NEW EDITION

LARGE SCALE APPS WITH VUE 3 AND TYPESCRIPT

DAMIANO FUSCO

Large Scale Apps with Vue 3 and TypeScript

Build Large and Scalable front-ends that leverage component isolation, internationalization, localization, a modular Vuex store, Custom Component Libraries, API-client code that easily can switch between mocked data and live data and more.

Damiano Fusco

This book is for sale at http://leanpub.com/vue-typescript

This version was published on 2020-11-18

Leanpub

* * * * *

feedback, pivot until you have the right book and build traction once you do.

* * * * *

ISBN for EPUB version: 978-1-63684-544-9

Table of Contents

LARGE SCALE APPS WITH VUE 3 AND TYPESCRIPT

Build front-ends that can grow to a large code base that is organized and easy to expand and maintain using technique like:

- **Development of UI components in isolation** using an API client that can easily serve live data or mocked data
- **A modular Vuex store** organized into application domains
- **Internationalization** and **Localization** for language translation and number/dates formatting according to a specific culture
- **TypeScript type-checking** at development time to decrease run-time bugs or errors
- **Directory structure, file, and code naming conventions**
- **Unit tests** for models and components
- **Components Libraries**
- And more

Companion Code: **github.com/damianof/large-scale-apps-my-vue3-project**

Running examples and other things: **largescaleapps.com**

Damiano Fusco

Web: damianofusco.com

Email: me@damianofusco.com

Twitter: @damianome

GitHub: github.com/damianof

GumRoad: gumroad.com/l/vue-typescript

ProductHunt: producthunt.com/@damianome

Preface

Thanks

First, I would like to thank **Evan You**[1], the creator of **Vue.js**[2]. Thank you for not giving up on this project a few years ago when it was still small and people were still skeptical about it. You gave us an amazing gift with such a lightweight and performing framework, which is progressive and not opinionated like many others out there.

I also want to thank my son for helping me proof read and validate the steps in each chapter by building the same project.

I want to thank my wife **Paola** for translating the first 5 chapters of the Italian edition, **Stefano Rampazzo** for translating the resto of the chapters of the Italian edition, **Alexey Pyltsyn** for translating the Russian edition, and **Victor Borràs Castell** for translating the Spanish edition.

Audience

The audience for this book is from beginners with some experience in **MV*** applications, to advanced developers. The format is similar to a cookbook, but instead of individual recipes we’ll go through creating a project and keep enhancing, refactoring, and make it better as we move forward to more advanced chapters to show different patterns, architecture, technologies.

Note: Some of the patterns illustrated here are not specific to **Vue**, but can applied in any application written in

TypeScript or **JavaScript**. For example, code from Chapters 3, 7, 8, 14 can also be used in **React/Angular** or other front-end apps. Similarly, code from Chapters 3 and 14 can also be used in **NodeJS** apps.

Goal

The primary goal of this book is to show you how to structure a Vue.js project, its directories and files, adopt naming conventions, follow state management patterns, enforce type checking at development time through TypeScript, write unit tests, and more, in proven ways that offer a solid foundation for building a large-scale application that is more easily to expand and maintain.

We’ll start creating a simple project and grow this throughout the chapters to show how some patterns, naming conventions, and strategies, will help create a more solid foundation and keeping the code organized and avoid cluttering.

We’ll create a TypeScript API client that can easily serve static mocked data or switch to communicate with a live API once that is available, thus enabling to develop our front-end early even before we (or another team) has completed their back-end coding for the API. We’ll integrate internationalization and localization, learn how we can create plugins, and more.

IMPORTANT: We will initially write code that allows us to achieve the desired functionality quickly, even if it requires more code, but then we constantly “rework” it (**refactoring**) to improve it and find solutions that allow us to reduce the amount of code used, or to organize it in a clear and easy way to is easy to expand and maintain. So arm yourself with a lot of patience!

Text Conventions

Going forward, I would refer to **Vue.js** simply with **Vue**. I will highlight most terms or names in bold, rather than define different fonts/styles depending on whether a term is code, or a directory name or something else.

Prerequisites

This book assumes that you are familiar with the **terminal** (**command prompt** on Windows), have already worked with the **Node.js** and **NPM** (**Node Package Manager**), know how to install packages, and are familiar with the **package.json** file.

It also assumes you are familiar with **JavaScript, HTML, CSS** and in particular with **HTML DOM** elements properties and events.

It will also help if you have some preliminary knowledge of **TypeScript**[1] as we won't get into details about the language itself or all of its features but mostly illustrate how to enforce type checking at development time with it.

You will need a text editor like **VS Code** or **Sublime Text**, better if you have extensions/plugins installed that can highlight Vue code syntax (like in **.vue** single files). For VS Code for example, you could use extensions like **Vetur**[2] (just search for it within the VS code extensions tab).

Companion Code

The entire companion code for the book can be found on **GitHub** at: github.com/damianof/large-scale-apps-my-vue3-project

Running examples and other things: largescaleapps.com

If you find any errors, or have difficulty completing any of the steps described in the book, please report them to me through the GitHub **issues** section here: github.com/damianof/large-scale-apps-my-vue3-project/issues

You are also free to reach out to me directly through Twitter at: @damianome

Chapter 1 - Setting Up The Project With The Vue-Cli

IMPORTANT: This chapter assumes that you already have installed a recent version of **Node.js** on your computer. If you do not have it yet, you can download it here: https://nodejs.org/en/download/

To set up the project you will use the terminal and the **vue-cli**[1]. If you do not have this installed on your computer yet, you can install it globally using **npm**:

```
npm install -g @vue/cli@next
```

Create Project Wizard

To create our project, do[2]:

```
vue create my-vue3-project
```

Next it will ask you to choose between these default presets:

- Default Vue 2
- Default Vue 3
- Manually Select Features

Select the **Manually Select Features** option (using arrows up/down) and hit enter:

```
Vue CLI v4.5.3
? Please pick a preset:
Default (\[Vue 2\] babel, eslint)
Default (Vue 3 Preview) (\[Vue 3\] babel, eslint)
❯ Manually select features
```

Next, it will ask you which features you want. Use the arrow keys to move up/down and press the space-bar to check/uncheck the features. For this project select only the **TypeScript**, **Router**, **Vuex**, **CSS Pre-processors**, **Unit Testing** features, then hit the Enter key:

```
? Check the features needed for your project:
 ◉ Choose Vue version
 ○ Babel
 ◉ TypeScript
 ○ Progressive Web App (PWA) Support
 ◉ Router
 ◉ Vuex
 ◉ CSS Pre-processors
 ○ Linter / Formatter
 ❯ ◉ Unit Testing
 ○ E2E Testing
```

Next, it will ask you which version of Vue you want to use. Use the arrow key to move up/down and select **3.x (preview)** by pressing the enter key:

```
? Choose a version of Vue.js that you want to start the project with
 2.x
 ❯ 3.x (Preview)
```

It will then ask you a series of questions.
Make the following choices:

Use class-style component syntax? N *(NOTE: we will be using the **Vue Composition API style** in this book, but we will give an example on how to use the **class-style syntax** as well in one of the advanced chapters)*

```
? Use class-style component syntax? (y/N) N
```

Use Babel alongside TypeScript? No

```
? Use Babel alongside TypeScript (required for modern mode, auto-detected
polyfills,\
 transpiling JSX)? (y/N) N
```

Use history mode for router? No

```
? Use history mode for router? (Requires proper server setup for index
fallback in p\
roduction) (Y/n) n
```

Pick a CSS pre-processor: Sass/SCSS (with node-sass)

```
? Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are
supported by d\
efault):
 Sass/SCSS (with dart-sass)
 ❯ Sass/SCSS (with node-sass)
 Less
 Stylus
```

Pick a unit testing solution: Mocha + Chai

```
? Pick a unit testing solution: (Use arrow keys)
 ❯ Mocha + Chai
 Jest
```

Where do you prefer placing config … ? In dedicated config files

```
? Where do you prefer placing config for Babel, ESLint, etc.? (Use arrow keys)
 ❯ In dedicated config files
 In package.json
```

The final choice will prompt you to save all the previous selections as a preset:

```
? Save this as a preset for future projects? (y/N) N
```

It’s ok to say No here, but if later you wish to create another project with the exact same features as this one and not having to go through all the steps, then go give it a name and save it.

NOTE: At this point, it might also ask you if you want to use yarn or npm as your package manager. If so, please choose npm as that is what we’ll be using in the book and thus will be easier to follow the examples. Of course, you are always free to choose yarn if that is your favorite tool.

The **vue-cli** will now create the project, install all the required NPM packages, create the configuration files, and stub some preliminary code (**Home** and **About** views, a simple **HelloWorld** component, **Vuex** store, etc)

At the end it should display a message similar to this:

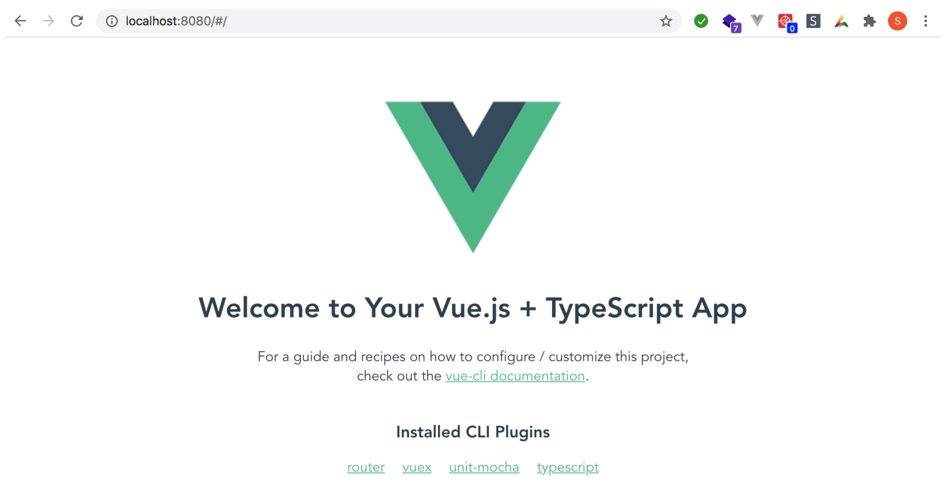
```
 Successfully created project my-vue3-project.
 Get started with the following commands:
$ cd my-vue3-project
$ npm run serve
```

The first command will navigate to the current sub-directory called **my-vue3-project**, the second will serve the app with the **vue-cli-service**. You'll see a message similar to this displayed:

```
DONE Compiled successfully in 3767ms 7:39:17 AM
App running at:
- Local: [http://localhost:8080/](http://localhost:8080/)
- Network: [http://192.168.1.3:8080/](http://192.168.1.3:8080/)

Note that the development build is not optimized.
To create a production build, run npm run build.
Issues checking in progress\...
No issues found.
```

NOTE: with vue-cli 4.5.3 I got a compile time error complaining about the router/index.ts Home component assignment type. If that should happen to you, just changed it like this "component: <any>Home" and should be able to pass that error. I did not experience this in existing Vue 2 projects upgraded to Vue 3 via "vue add vue-next". Hopefully as the Vue team gets closer to the official Vue 3 release, issues like these will be better addressed.

From the web browser, navigate to the http://localhost:8080/ address and you'll see application home page rendered:


localhost:8080/#/
Welcome to Your Vue.js + TypeScript App
For a guide and recipes on how to configure / customize this project,
check out the vue-cli documentation.
Installed CLI Plugins
router
vuex
unit-mocha
typescript

BrowserImage

The **my-vue3-project** has been created with two stubbed views, **Home.vue** and **About.vue** and two routes have been added to the **Vue** router to allow navigation to those two views.

Now stop the app with CTRL+C in the terminal. Run this command to install the package **cross-env**[3]:

```
npm install cross-env --save-dev
```

cross-env allows to set environment variables across different operating systems in the same way.

If you want to know more details about this, see the next section **Differences Between Operating Systems**, otherwise you can skip to Chapter 2.

Differences Between Operating Systems

There are some difference between different operating systems in the way we set environment variables in the “scripts” commands of our **package.json** file.

You can see the use of the keyword **export**. This works on **OSX** and **Linux** environment, but **Windows** uses a different keyword and syntax.

For example this command:

```
"serve": "export VUE_APP_API_CLIENT=mock; vue-cli-service serve --mode
development"
```

On Windows it would have to be converted to this:

```
"serve": "set \"VUE_APP_API_CLIENT=mock\" && vue-cli-service serve --mode
developmen\
t"
```

Note how the **export** keyword becomes **set** and the **semi-colon** becomes **&&**. It is also recommended to wrap the **variable=value** expression within escaped quotes, like in **\“VUE_APP_API_CLIENT=mock\“**

You could add additional shortcuts that are **Windows** specific by prefixing them with **win-** like here fore example:

```
"win-serve": "set \"VUE_APP_API_CLIENT=mock\" && vue-cli-service serve --mode
develo\
pment",
```

However, thanks to the **cross-env** package, all I had to do is to update the commands to use the following syntax and stopped worrying about specific operating systems:

```
 "serve": "cross-env VUE_APP_API_CLIENT=mock vue-cli-service serve \--mode
developme\
nt",
 "build": "cross-env VUE_APP_API_CLIENT=live vue-cli-service build \--mode
productio\
n",
 "build-mock": "cross-env VUE_APP_API_CLIENT=mock vue-cli-service build \--
mode prod\
uction"
```

Just remember to be aware that these differences exists if you do not use something like **cross-env**.

Chapter 1 Recap

What We Learned

How to create the basic plumbing for a **Vue 3** app using the **vue-cli**

- How to serve the app using the **vue-cli** service through the command **npm run serve**

Observations

- The app has been created with a preliminary router, routes, and views
- The app does not do much yet, has only two very simple views with static html in them

Based on these observations, there are a few improvements that will be making into the next chapter:

Improvements

- Expand our app functionality by creating our first component

```
<code></code><code class="p"><</code><code
class="nt">template</code><code class="p">></code>

   <code class="p"><</code><code
class="nt">div</code><code class="p">></code>

   <code class="p"><</code><code
class="nt">h3</code><code class="p">></code>Items:
<code class="p"></</code><code class="nt">h3</code>
<code class="p">></code>

   <code class="p"><</code><code class="nt">ul</code>
<code class="p">></code>

   <code class="p"><</code><code class="nt">li</code>
<code class="na">v-for</code><code class="o">=
</code><code class="s">"item in items"</code> <code
class="na">:key</code><code class="o">=</code>
<code class="s">"item.id"</code><code class="p">>
</code>

   {{ item.name }}

   <code class="p"></</code><code class="nt">li</code>
<code class="p">></code>

   <code class="p"></</code><code
class="nt">ul</code><code class="p">></code>

   <code class="p"></</code><code
class="nt">div</code><code class="p">></code>

<code class="p"></</code><code
class="nt">template</code><code class="p">></code>
```

```
<code class="p"><</code><code class="nt">script</code> <code class="na">lang</code> <code class="o">=</code><code class="s">"ts"</code> <code class="p">></code>

   <code class="kr">import</code> <code class="p">{</code> <code class="nx">defineComponent</code> <code class="p">,</code> <code class="nx">PropType</code> <code class="p">}</code> <code class="nx">from</code> <code class="s1">'vue'</code>



   <code class="kr">export</code> <code class="k">default</code> <code class="nx">defineComponent</code><code class="p">({</code>

   <code class="nx">props</code><code class="o">:</code> <code class="p">{</code>

   <code class="nx">items</code><code class="o">:</code> <code class="p">{</code>

   <code class="nx">type</code><code class="o">:</code> <code class="nb">Array</code> <code class="nx">as</code> <code class="nx">PropType</code><code class="o"><</code> <code class="nx">any</code><code class="p">[]</code><code class="o">></code>

   <code class="p">}</code>

   <code class="p">}</code>

   <code class="p">})</code>
```

```
<code class="p"></</code><code class="nt">script</code><code class="p">></code>

<code></code><code class="p">...</code>

   <code class="nx">props</code><code class="o">: </code> <code class="p">{</code>

   <code class="nx">items</code><code class="o">: </code> <code class="p">{</code>

   <code class="nx">type</code>: <code class="kt">Array</code> <code class="kr">as</code> <code class="nx">PropType</code><code class="o"><</code><code class="nx">any</code><code class="p">[]</code><code class="o">></code> <code class="c1">// avoid using "any"</code>

   <code class="p">}</code>

   <code class="p">}</code>

<code class="p">...</code>

<code></code><code class="p"><</code><code class="nt">template</code><code class="p">></code>

<code class="p"><</code><code class="nt">div</code> <code class="na">class</code><code class="o">=</code><code class="s">"home"</code><code class="p">></code>

<code class="p"></</code><code class="nt">div</code> <code class="p">></code>

<code class="p"></</code><code class="nt">template</code><code class="p">></code>
```

```
<code class="p"><</code><code
class="nt">script</code> <code class="na">lang</code>
<code class="o">=</code><code class="s">"ts"</code>
<code class="p">></code>

<code class="p"></</code><code
class="nt">script</code><code class="p">></code>

<code></code><code class="p">...</code>

<code class="kr">import</code> <code class="p">
{</code> <code class="nx">defineComponent</code>
<code class="p">}</code> <code
class="nx">from</code> <code class="s1">'vue'</code>

<code class="p">...</code>

<code class="kr">export</code> <code
class="k">default</code> <code
class="nx">defineComponent</code><code class="p">
({</code>

   <code class="nx">name</code><code class="o">:
</code> <code class="s1">'Home'</code>

<code class="p">})</code>

<code></code> <code class="kr">import</code> <code
class="p">{</code> <code
class="nx">defineComponent</code> <code class="p">}
</code> <code class="nx">from</code> <code
class="s1">'vue'</code>

<code class="kr">import</code> <code
class="nx">ItemsListComponent</code> <code
class="nx">from</code> <code
class="s1">'@/components/items/</code>
```

```
<code class="s1"> export default
defineComponent({</code>

<code class="s1"> name: '</code><code
class="nx">Home</code><code class="err">'</code>
<code class="p">,</code>

   <code class="nx">components</code><code
class="o">:</code> <code class="p">{</code>

   <code class="nx">ItemsListComponent</code>

   <code class="p">}</code>

<code class="p">})</code>

<code></code><code class="p">...</code>

<code class="kr">export</code> <code
class="k">default</code> <code
class="nx">defineComponent</code><code class="p">
({</code>

<code class="nx">name</code><code class="o">:
</code> <code class="s1">'Home'</code><code
class="p">,</code>

<code class="nx">components</code><code class="o">:
</code> <code class="p">{</code>

   <code class="nx">ItemsListComponent</code>

<code class="p">},</code>

<code class="nx">setup() {</code>

   <code class="kr">const</code> <code
class="nx">items</code>: <code class="kt">any</code>
<code class="p">[]</code> <code class="o">=</code>
<code class="p">[{</code>

   <code class="nx">id</code>: <code
class="kt">1</code><code class="p">,</code>

   <code class="nx">name</code><code class="o">:
</code> <code class="s1">'Item 1'</code>

   <code class="p">},</code> <code class="p">{</code>

   <code class="nx">id</code>: <code
class="kt">2</code><code class="p">,</code>

   <code class="nx">name</code><code class="o">:
</code> <code class="s1">'Item 2'</code>

   <code class="p">},</code> <code class="p">{</code>

   <code class="nx">id</code>: <code
class="kt">3</code><code class="p">,</code>

   <code class="nx">name</code><code class="o">:
</code> <code class="s1">'Item 3'</code>

   <code class="p">}]</code>

   <code class="k">return</code> <code class="p">
{</code>

   <code class="nx">items</code>

   <code class="p">}</code>

   <code class="p">}</code>

<code class="p">}</code>
```

```
<code></code><code class="o"><</code><code
class="nx">template</code><code class="o">></code>

   <code class="o"><</code><code
class="nx">div</code> <code class="kr">class</code>
<code class="o">=</code><code class="s2">"home"
</code><code class="o">></code>

   <code class="o"><</code><code
class="nx">ItemsListComponent</code> :<code
class="kt">items</code><code class="o">=</code>
<code class="s2">"items"</code> <code class="o">/>
</code>

   <code class="o"><</code><code class="err">/div>
</code>

<code class="o"><</code><code class="err">/template>
</code>



<code class="o"><</code><code
class="nx">script</code> <code class="nx">lang</code>
<code class="o">=</code><code class="s2">"ts"
</code><code class="o">></code>

   <code class="kr">import</code> <code class="p">
{</code> <code class="nx">defineComponent</code>
<code class="p">}</code> <code
class="nx">from</code> <code class="s1">'vue'</code>

   <code class="kr">import</code> <code
class="nx">ItemsListComponent</code> <code
class="nx">from</code> <code
class="s1">'@/components/items/ItemsList.component.vue'
</code>
```

<code class="kr">export</code> <code class="k">default</code> <code class="nx">defineComponent</code><code class="p">({</code>

<code class="nx">name</code><code class="o">:</code> <code class="s1">'Home'</code><code class="p">,</code>

<code class="nx">components</code><code class="o">:</code> <code class="p">{</code>

<code class="nx">ItemsListComponent</code>

<code class="p">},</code>

<code class="nx">setup() {</code>

<code class="kr">const</code> <code class="nx">items</code>: <code class="kt">any</code> <code class="p">[]</code> <code class="o">=</code> <code class="p">[{</code>

<code class="nx">id</code>: <code class="kt">1</code><code class="p">,</code>

<code class="nx">name</code><code class="o">:</code> <code class="s1">'Item 1'</code>

<code class="p">},</code> <code class="p">{</code>

<code class="nx">id</code>: <code class="kt">2</code><code class="p">,</code>

```
   <code class="nx">name</code><code class="o">:
</code> <code class="s1">'Item 2'</code>

   <code class="p">},</code> <code class="p">{</code>

   <code class="nx">id</code>: <code
class="kt">3</code><code class="p">,</code>

   <code class="nx">name</code><code class="o">:
</code> <code class="s1">'Item 3'</code>

   <code class="p">}]</code>



   <code class="k">return</code> <code class="p">
{</code>

   <code class="nx">items</code>

   <code class="p">}</code>

   <code class="p">}</code>

   <code class="p">})</code>

<code class="o"><</code><code class="err">/script>
</code>
```

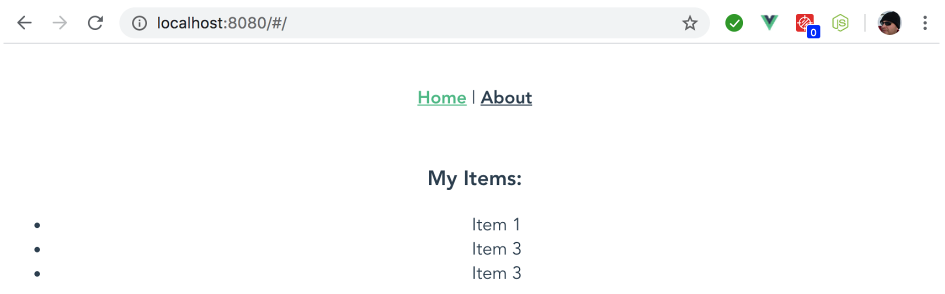
```
<code></code><code class="p"><</code><code
class="nt">template</code><code class="p">></code>

   <code class="p"><</code><code
class="nt">div</code> <code class="na">id</code>
<code class="o">=</code><code class="s">"app"
</code><code class="p">></code>

   <code class="p"><</code><code
class="nt">h2</code><code class="p">></code>My Vue
3 Project<code class="p"></</code><code
class="nt">h2</code><code class="p">></code>

   <code class="p"><</code><code
class="nt">div</code> <code class="na">id</code>
<code class="o">=</code><code class="s">"nav"
</code> <code class="na">class</code><code
class="o">=</code><code class="s">"nav"</code>
<code class="p">></code>

   <code class="p"><</code><code class="nt">router-
link</code> <code class="na">to</code><code
class="o">=</code><code class="s">"/"</code><code
class="p">></code>Home<code class="p"></</code>
<code class="nt">router-link</code><code class="p">>
</code> |

   <code class="p"><</code><code class="nt">router-
link</code> <code class="na">to</code><code
class="o">=</code><code class="s">"/about"</code>
<code class="p">></code>About<code class="p">
</</code><code class="nt">router-link</code><code
class="p">></code>

   <code class="p"></</code><code
class="nt">div</code><code class="p">></code>
```

```
   <code class="p"><</code><code class="nt">router-
view</code> <code class="p">/></code>

   <code class="p"></</code><code
class="nt">div</code><code class="p">></code>

<code class="p"></</code><code
class="nt">template</code><code class="p">></code>

<code></code><code class="o"><</code><code
class="nx">script</code> <code class="nx">lang</code>
<code class="o">=</code><code class="s2">"ts"
</code><code class="o">></code>

   <code class="kr">import</code> <code class="p">
{</code> <code class="nx">defineComponent</code>
<code class="p">}</code> <code
class="nx">from</code> <code class="s1">'vue'</code>



   <code class="kr">export</code> <code
class="k">default</code> <code
class="nx">defineComponent</code><code class="p">
({</code>

   <code class="nx">name</code><code class="o">:
</code> <code class="s1">'App'</code>

   <code class="p">})</code>

<code class="o"><</code><code class="err">/script>
</code>

<code></code><code class="o"><</code><code
class="nt">style</code> <code class="nt">lang</code>
```

```
<code class="o">=</code><code class="s2">"scss"
</code><code class="o">></code>

   <code class="nn">#app</code> <code class="p">
{</code>

   <code class="nt">font-family</code><code
class="nd">:</code> <code class="nt">Avenir</code>
<code class="o">,</code> <code
class="nt">Helvetica</code><code class="o">,</code>
<code class="nt">Arial</code><code class="o">,</code>
<code class="nt">sans-serif</code><code class="o">;
</code>

   <code class="nt">-webkit-font-smoothing</code><code
class="nd">:</code> <code
class="nt">antialiased</code><code class="o">;</code>

   <code class="nt">-moz-osx-font-smoothing</code>
<code class="nd">:</code> <code
class="nt">grayscale</code><code class="o">;</code>

   <code class="nt">text-align</code><code class="nd">:
</code> <code class="nt">center</code><code
class="o">;</code>

   <code class="nt">color</code><code class="nd">:
</code> <code class="nn">#2c3e50</code><code
class="o">;</code>

   <code class="nt">h2</code> <code class="p">
{</code>

   <code class="nt">margin</code><code class="nd">:
</code> <code class="nt">0</code><code class="o">;
</code>
```

```
   <code class="p">}</code>

   <code class="p">}</code>



   <code class="nn">#nav</code> <code class="p">
{</code>

   <code class="nt">padding</code><code class="nd">:
</code> <code class="nt">30px</code><code
class="o">;</code>

   <code class="nt">a</code> <code class="p">{</code>

   <code class="nt">font-weight</code><code
class="nd">:</code> <code class="nt">bold</code>
<code class="o">;</code>

   <code class="nt">color</code><code class="nd">:
</code> <code class="nn">#2c3e50</code><code
class="o">;</code>

   <code class="k">&</code><code class="nc">.router-
link-exact-active</code> <code class="p">{</code>

   <code class="nt">color</code><code class="nd">:
</code> <code class="nn">#42b983</code><code
class="o">;</code>

   <code class="p">}</code>

   <code class="p">}</code>

   <code class="p">}</code>

<code class="o"></</code><code
class="nt">style</code><code class="o">></code>
```

Save the file. The web browser will refresh and display our preliminary items list being rendered more or less like this:



My Items:

- Item 1
- Item 3
- Item 3

Chapter 2 Recap

What We Learned

- How to create a basic component that displays a list of items
- How to consume that component in a view

Observations

- The **items** property within the **ItemsList.component.vue** is declared as an array of type **any**
- The **Home.vue** view contains hard-coded data (items) which is also declared as an array of **any**
- This means we are not leveraging strong-type checking at development time using TypeScript interfaces/models/types

Based on these observations, there are a few improvements that we will make in the next chapters:

Improvements

- Create a TypeScript interface called **ItemInterface** for enforcing type checking at development time for our **items** data
- Update our code so it uses the new **ItemInterface** interface

Chapter 3 - Data Model Interfaces

In this chapter, we will keep building and improving our Vue project by starting to leverage TypeScript interfaces for strong-type checking at development time. One of the disadvantage or pure JavaScript is that is **loosely typed**, and this might cause issues at run-time as there are no checks on the type and or the expected properties of a value or object passed around through our code. TypeScript main advantage is the ability to enforce **strong-type checking** at development time through the use of **interfaces**, types, **classes**, and more.

Models Directory

Start by creating a new sub-directory under **src** called **models**. Since the focus of this book is on building a foundation for large applications, we'll keep structuring our **files** and directories in a consistent way, following our own **convention**. You and your team are free to decide what your standards will be, but it's important to have both files/directory naming conventions and structuring in place as soon as you start building a large application. This will save you a lot of confusion and headaches later as the application grows exponentially and the number of source files and directories grows with it.

Interface ItemInterface

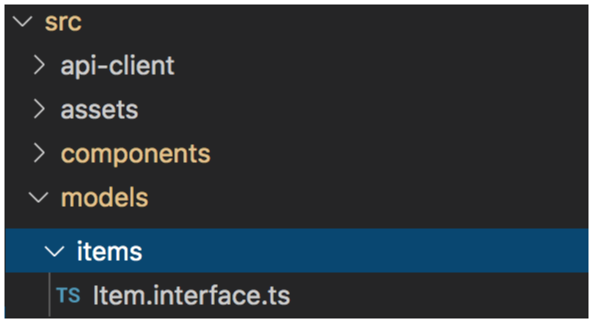
Create the directory **src/models/items** and here add a new TypeScript file called **Item.interface.ts**.

*Note: we'll be following a naming convention for TypeScript files that represents the interface by adding the **Interface***

suffix *(in the first edition of this book I used the "**I**" prefix was like in **IItem** but this is definitely harder especially on people coming from cultures other than the US which are less used to abbreviations and acronyms in code).*

In this version of the book, we’ll save each interface into its own file with a file name convention like **Item.interface.ts**. (see the Naming Conventions section at the end of this book for more information)

Your directory structure should now look similar to this:



Let’s write an interface that represents one item that will be rendered in our **Item** component. Our interface will have three properties:

- **id**: this is a unique number for each item in the list
- **name**: is a string containing the name of the item
- **selected**: is a boolean value that shows if the user has selected the item

The code for your interface should look like this:

```
export interface ItemInterface {
  id: number
  name: string
  selected: boolean
}
```

For now, that is all we need. Since this will only represent a piece of data, we do not need to implement a class.

NOTE: In this case our ItemInterface only holds fields, but no methods. You can think of this more like the type struct in language like C or C#. Unfortunately TypeScript does not have an explicit struct type[1] *and their guidance is to use interfaces for this.*

ItemsList Component

Now that we have our interface, we can finally leverage TypeScript type checking ability by changing our items property on the items component from **any[]** to **ItemInterface[]**. First, import a reference for **ItemInterface**:

```
<script lang="ts">
  import { defineComponent, PropType } from 'vue'
  import { ItemInterface } from '@/models/items/Item.interface'
```

Then modify our items property declaration from any[] to ItemInterface[]:

```
export default defineComponent({
    props: {
      items: {
        type: Array as PropType<ItemInterface[]>
      }
    }
  }
  ...
```

The complete update code should look like this:

```
<template>
  <div>
    <h3>Items:</h3>
     <ul>
       <li v-for="item in items" :key="item.id">
         {{ item.name }}
       </li>
     </ul>
  </div>
</template>
<script lang="ts">
```

```
  import { defineComponent, PropType } from 'vue'
  import { ItemInterface } from '@/models/items/Item.interface'

  export default defineComponent({
    props: {
      items: {
        type: Array as PropType<ItemInterface[]>
      }
    }
  })
</script>
```

Make sure the terminal does not display any error, and that the web browser refreshed and no error are displayed in the browser console.

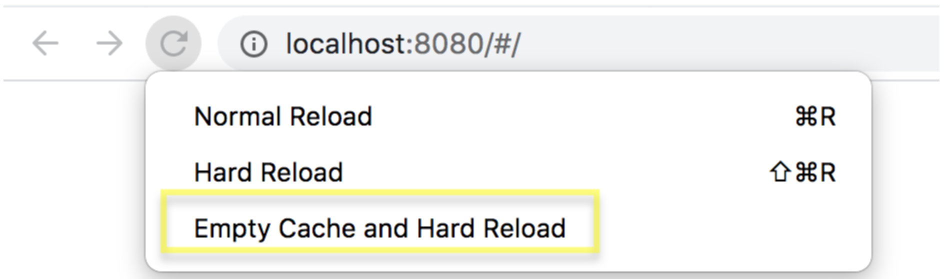
Home View

We should also update the **Home.vue** code so it uses the **ItemInterface** interface for the locally private property also called **items**.

Please note, that as soon as you change the items property from any[] to ItemInterface[] it will complain that each item does not correctly implement the interface. This is because we did not initially include the selected property required by the interface. This is one of the powerful things of using TypeScript correctly. It will help catching errors like this at development time rather than run time and increase the code quality and have it less prone to bugs.

```
<script lang="ts">
  import { defineComponent } from 'vue'
  import ItemsListComponent from '@/components/items/ItemsList.component.vue'
  import { ItemInterface } from '@/models/items/Item.interface'

  export default defineComponent({
    name: 'Home',
    components: {
      ItemsListComponent
    },
    setup() {
      const items: ItemInterface[] = [{
        id: 1,
        name: 'Item 1',
        selected: false
      }, {
        id: 2,
        name: 'Item 2',
        selected: false
      }, {
        id: 3,
        name: 'Item 3',
        selected: false
      }]

      return {
        items
      }
    }
  })
</script>
```

Again, make sure the terminal does not display any errors, and that the web browser refreshed and no error are displayed in the browser console. As you make changes is also a good idea occasionally to do an **Empty Cache and Hard Reload** by right clicking on the Chrome refresh icon and selecting the last option:



Chapter 3 Recap

What We Learned

- It’s important to follow files and directories naming convention and structure convention
- How to leverage **TypeScript** interfaces and avoid using **any** so that strong-type checking is enforced at development time and avoiding potential runtime errors or hidden bugs

Observations

- The **Home.vue** contains a local property that holds hard-coded mocked data that enabled us to prototype our component quickly
- **ItemsList.component.vue** just display the list of items, but the user has still no ability to click on them to change their **selected** property

Based on these observations, there are a few improvements that we will make into the next chapter:

Improvements

- Update our component so that when a user clicks on an item displayed on the page, the item selected property will toggle from false to true (and vice versa)

Chapter 4 - Adding Events To the Items Component

In this chapter we keep building our **ItemsList.component.vue** so we can handle when the user clicks on an item in the list.

ItemsList Component

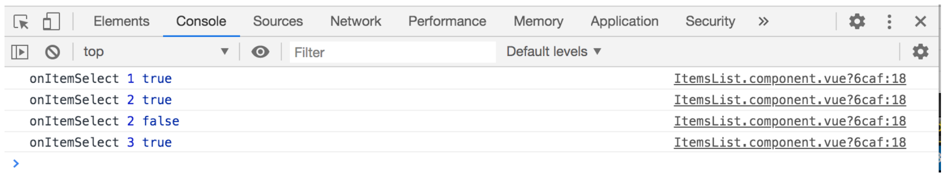
Start by updating the **<template>** section by adding a **@click**[1] attribute to the **<li>** element, pointing to an handler called **onItemClick** and passing a reference to **item** as the argument:

```
<li v-for="item in items" :key="item.id" @click="onItemSelect(item)">
  {{ item.name }}
</li>
```

Then within the **<script>** section add a function that implements our **onItemSelect** handler, toggles the **item.selected** property from true to false or vice versa and logs the item **id** and **selected** properties to the console for preliminary debugging. Note we are doing this in the **setup()** method of the **Vue** component by setting the **onItemSelect** on a variable and returning an object that contains the method. There are other ways of doing this, but for now will stick to using the **setup()** method:

```
export default defineComponent({
    props: {
      items: {
        type: Array as PropType<ItemInterface[]>
      }
    },
    setup() {
      const onItemSelect = (item: ItemInterface) => {
        item.selected = !item.selected
        console.log('onItemSelect', item.id, item.selected)
      }
      return {
        onItemSelect
      }
    }
  }
...
```

Then, the web browser should have refreshed, and when clicking on the items in the list you should see the message being displayed in the browser developer console, and when clicking multiple time on the same item it should print true then false etc showing that toggling is working:



```
onItemSelect 1 true                                    ItemsList.component.vue?6caf:18
onItemSelect 2 true                                    ItemsList.component.vue?6caf:18
onItemSelect 2 false                                   ItemsList.component.vue?6caf:18
onItemSelect 3 true                                    ItemsList.component.vue?6caf:18
```

Now, we learned how to add a click handler to our component and changing the data item selected property that way. However, in an app that will grow large, have many components/views that will need to be aware of each other state, or the state of the data they manipulate, this is not the best pattern to use.

In the next chapter we'll introduce **Vuex** to manage our state in a more appropriate pattern where changes in state are done in a centralized place (State Management Pattern) and offers a way to separate state from views and actions.

Here is the official **Vuex** definition from the official website[2]:

> *"Vuex is a **state management pattern + library** for Vue.js applications. It serves as a centralized store for all the components in an application, with rules ensuring that the state can only be mutated in a predictable fashion. It also integrates with Vue's official devtools*

extension to provide advanced features such as zero-config time-travel debugging and state snapshot export/import."

Chapter 4 Recap

What We Learned

- How to add a click handler to our **ItemsList** component
- How to manipulate the **item.selected** property through our click handler

Observations

- The **items selected** property is being manipulated directly within our component
- We need a more centralized way to handle changes on the data and state of the application

Based on these observations, there are a few improvements that we will make in the next chapters:

Improvements

- Implement a **Vuex** state store and commit the changes to our items in a centralized place

Chapter 5 - Intro to Unit Testing While Refactoring a Bit

We will now see how to add some unit tests to our project. There are two main categories of unit tests that we will write in this book:

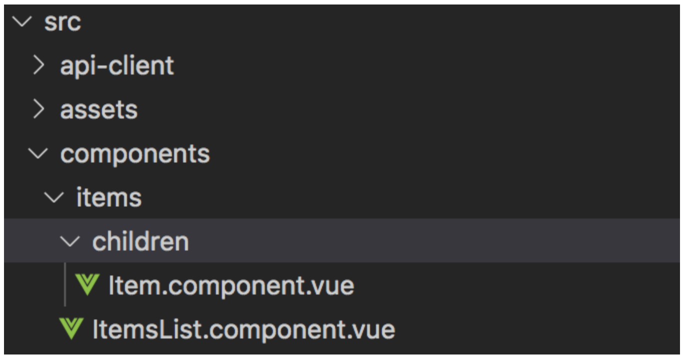
- unit tests for models/classes/structures/interfaces (i.e. Api client, Vuex mutations and actions, Api etc)
- unit tests for Vue components

Note: there is also a 3rd category called e2e (end to end) tests, but we will not be getting into those in this book.

Since we coded our first component already, we should start with the second category for now. We will add our first unit tests for a component and while doing so, we'll refactor our **ItemsList** component a little and we can use the unit tests to validate our changes.

ItemComponent

Remember how in our **ItemsList** component we have a **v-for** loop that creates individual **<li>** for each item in our items property? Let's extract the code for the **<li>** element and create a child component just for that. Let's start by adding a new file called **Item.component.vue** under the **src/components/items/children** directory:



Now, paste the following code at the top for the **template** section:

```
<template>
  <li :class="css" @click="onClick">
    <div class="selected-indicator">*</div>
    <div class="name">{{ model.name }}</div>
  </li>
</template>
...
```

For the **<script>** section paste the following code after the template section:

```
...
<script lang="ts">
  import { defineComponent, computed, PropType } from 'vue'
  import { ItemInterface } from '@/models/items/Item.interface'

  export default defineComponent({
    props: {
      model: {
        type: Object as PropType<ItemInterface>
      }
    },
    setup(props, { emit }) {

      const css = computed(() => {
        let css = 'item'
        if (props.model?.selected) {
          css += ' selected'
        }
        return css.trim()
      })

      const onClick = () => {
        emit('select', props.model)
      }

      return {
        css,
        onClick
      }
    }
  })
</script>
```

We just created a template for a single **<li>** element. We also enhanced this a bit by replacing the rendering of the name with binding **}** with two child **<div>** elements:

- one to display the Item name
- one that will show a star icon (*we are just using a char here, but in the next chapters we'll be replacing this with real icons from the font library material-icons*)

Then we added a computed property called **css** that will return the string "**item**" or "**item selected**". We then bind this to the **<li> class** attribute through the **:class** binding, based on whether the **model.selected** property is true or false: **<li :class="css" @click="onClick">**

This will have the effect to render the **<li>** element in two possible ways:

- <li class="item"> (when not selected)
- <li class="item selected"> (when selected)

We also bind to the click event with **@click** binding and in the local **onClick** handler we just emit the custom **select** event and pass the model as the argument (props.model). We will then handle this in the parent component (**ItemsList** component).

Also add some custom CSS to make it look nicer:

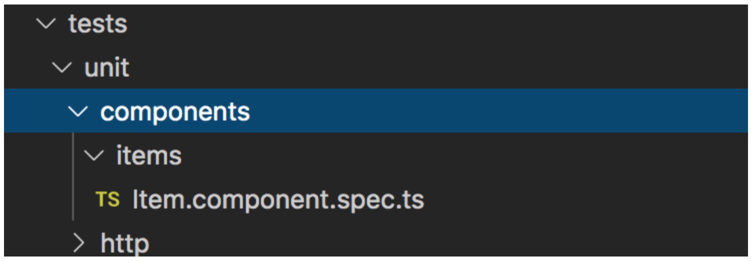
```
...
<style lang="scss">
  li.item {
    padding: 0;
    outline: solid 1px #eee;
    display: grid;
    grid-template-columns: 1.7em auto;
    cursor: pointer;
    transition: background-color 0.3s ease;

    .name {
      padding: 5px;
      text-align: left;
    }
    .selected-indicator {
      padding: 5px;
      font-size: 3em;
      line-height: 0.5em;
      padding: 5px;
      color: lightgray;
    }
    &.selected {
      .selected-indicator {
        color: skyblue;
      }
    }
    &:hover {
      background-color: #eee;
    }
  }
</style>
```

NOTE: I am not suggesting here that CSS should be part of the single .vue files. This is one way of doing this. I am doing it for now because is the fastest way to style a component in our project without using a css framework or similar technique. In later chapters I'll show you how to add a CSS framework using a plugin, instead of writing CSS within the components.

ItemComponent Unit Tests

Now, let's add a unit test for our newly created component.

Create the directory **tests/unit/components/items** and here add a file called **Item.component.spec.ts**[1]



Within the unit tests file, paste the following code:

```
import { expect } from 'chai'
import { shallowMount } from '@vue/test-utils'
import ItemComponent from '@/components/items/children/Item.component.vue'
import { ItemInterface } from '@/models/items/Item.interface'

describe('Item.component.vue', () => {
  it('renders an Item correctly', () => {
    const model: ItemInterface = {
      id: 1,
      name: 'Unit test item 1',
      selected: false
    }

    const wrapper = shallowMount(ItemComponent, {
      props: {
        model: model
      }
    })

    expect(wrapper.text()).to.include('Unit test item 1')
  })
})
```

Here we test that the component renders the data model properties as expected. For now, we are checking if the entire text rendered by the component contains the model.name. This is not very precise as our component later might render additional labels and our test might match these instead resulting in possible false positives.

A better and more precise way to test what our component has rendered, is to use the wrapper.find utility to select specific html DOM elements and then check if those have render the expected text (or even check for other properties like if they exist, they are visible etc).

Run our unit tests from the terminal with this command:

```
npm run test:unit
```

It should run the unit tests and print the results on the terminal, similar to this:

```
 DONE  Compiled successfully in 2779ms
  [=========================] 100% (completed)

 WEBPACK  Compiled successfully in 2779ms
 MOCHA  Testing...

  HelloWorld.vue
    ✓ renders props.msg when passed

  Item.component.vue
    ✓ renders an Item correctly

  2 passing (49ms)

 MOCHA  Tests completed successfully
```

Let’s now improve our unit test so we can select the **<div>** with the class "**name**" and check that it renders our **model.name**:

```
import { expect } from 'chai'
import { shallowMount } from '@vue/test-utils'
import ItemComponent from '@/components/items/children/Item.component.vue'
import { ItemInterface } from '@/models/items/Item.interface'

describe('Item.component.vue', () => {
  it('renders an Item correctly', () => {

    const model: ItemInterface = {
      id: 1,
      name: 'Unit test item 1',
      selected: false
    }

    const wrapper = shallowMount(ItemComponent, {
      props: {
        model: model
      }
    })

    // this just tests that the entire text rendered by the component
somewhere rend\
ered
    // 'Unit test item 1', but this is not very precise.
    expect(wrapper.text()).to.include('Unit test item 1')

    // this is more precise as we are selecting the div with the class name
and chec\
k if it rendered the correct text
    let domEl = wrapper.find('div.name')
    expect(domEl.text()).to.equal('Unit test item 1')
  })
})
```

Let’s add two more tests within the same **describe** section to check that the component has the expected CSS classes (in this case the text "item", and when the **model.selected** property is **false** it should not have the text "selected"):

```
  it('has expected css class when selected is false', () => {
    const model: ItemInterface = {
      id: 2,
      name: 'Unit test item 2',
      selected: false
    }

    const wrapper = shallowMount(ItemComponent, {
      props: {
        model: model
      }
    })

    // check component css classes list
    const classes = wrapper.classes()
    expect(classes).to.be.an('array')
      .that.includes('item')
    expect(classes).to.be.an('array')
      .that.does.not.includes('selected')
  })
```

(here too the text "item", and when the **model.selected** property is **true** it should also have the text "selected"):

```
  it('has selected css class when selected is true', () => {
    const model: ItemInterface = {
      id: 3,
      name: 'Unit test item 3',
      selected: true /* setting selected = true here */
    }

    const wrapper = shallowMount(ItemComponent, {
      props: {
        model: model
      }
    })

    // check component css classes list
    const classes = wrapper.classes()
    expect(classes).to.be.an('array')
      .that.includes('item')
    expect(classes).to.be.an('array')
      .that.includes('selected')
  })
```

We could also trigger a click with **wrapper.trigger('click')** but our component only emits the event to be handled by a parent component, so we might have to do that in a unit tests that will write for our ItemsList component later in the book. For now, let's stop here with unit tests, we'll get into more advanced testing later in the book.

If you execute **npm run test:unit** again, this time should print results similar to these:

```
 Item.component.vue
    ✓ renders an Item correctly
    ✓ has expected css class when selected is false
    ✓ has selected css class when selected is true

  3 passing (29ms)
```

ItemsList component

We now have to change our **ItemsList.component.vue** so that it uses our newly created **Item.component.vue** in place of the **<li>** element. Here are the changes required in our **ItemsList.component.vue** code:

- Import a reference to our **ItemComponent**
- Add a reference within the **components** section
- Remove the existing **<li>** code within the <template> section and replace it with our imported Item component

Here is the **<template>** section with the highlighted changes:

```
<template>
  <div>
    <h3>Items:</h3>
     <ul>
       <!-- Remove this existing <li> block: -->
       <li v-for="item in items" :key="item.id" @click="onItemClick(item)">
         {{ item.name }}
       </li>
```

```
      <!-- add this: -->
      <ItemComponent v-for="item in items"
         :key="item.id"
         :model="item"
         @select="onItemSelect" />
    </ul>
  </div>
</template>
```

Here is the **<script>** section with the highlighted changes:

```
<script lang="ts">
  import { defineComponent, PropType } from 'vue'
  import { ItemInterface } from '@/models/items/Item.interface'
  import ItemComponent from './children/Item.component.vue' // <-- add this

  export default defineComponent({
    // add the components block here:
    components: {
      ItemComponent
    },
    props: {
      items: {
        type: Array as PropType<ItemInterface[]>
      }
    },
    setup() {
      const onItemSelect = (item: ItemInterface) => {
        item.selected = !item.selected
        console.log('onItemSelect', item.id, item.selected)
      }

      return {
        onItemSelect
      }
    }
  })
</script>
```

And add some custom css here to make the whole **<ul>** looks a little better:

```
<style lang="scss">
  ul {
    list-style-type: none;
    margin-block-start: 0;
    margin-block-end: 0;
    margin-inline-start: 0px;
    margin-inline-end: 0px;
    padding-inline-start: 0px;
  }
</style>
```

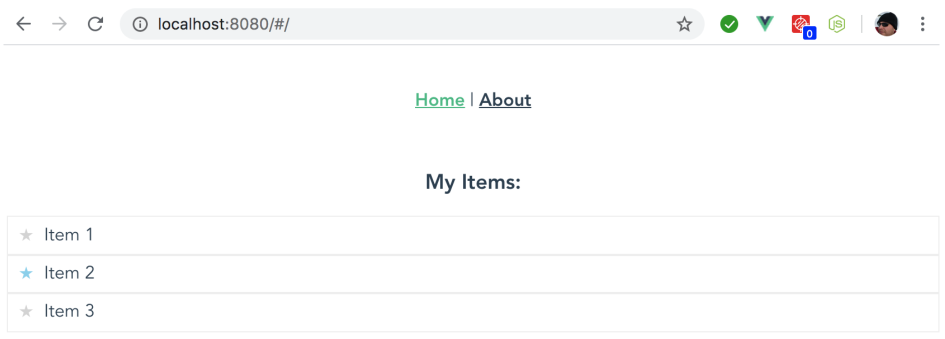
If you refresh the web browser, the list should render with the new style applied. However, clicking on the items does not seem to change their state in the UI. This is because we did not make the **items** property in our **Home** page **reactive** yet.

Home.vue updates

Open **Home.vue** and make these changes. Add **reactive** to the import list from **‘vue’**, and then wrap the mock data array with **reactive(…)**:

```
<script lang="ts">
  import { defineComponent, reactive } from 'vue'
  import ItemsListComponent from '@/components/items/ItemsList.component.vue'
  import { ItemInterface } from '@/models/items/Item.interface'

  export default defineComponent({
    name: 'Home',
    components: {
      ItemsListComponent
    },
    setup() {

      const items: ItemInterface[] = reactive([{
        id: 1,
        name: 'Item 1',
        selected: false
      }, {
        id: 2,
        name: 'Item 2',
        selected: false
      }, {
        id: 3,
        name: 'Item 3',
        selected: false
      }])

      return {
        items
      }
    }
  })
</script>
```

From the web browser, now the list should render similar to this (here we are showing it with the 2nd item element being clicked):



My Items:

★ Item 1

★ Item 2

★ Item 3

Chapter 5 Recap

What We Learned

- How to write unit tests against a component
- How to test that DOM elements render specific text, or have specific attributes like CSS classes, etc.
- How to re-factor parts of a component to create a child component and use unit tests to validate our changes

Observations

- We did not test user interaction with our component

Based on these observations, there are a few improvements that will make in the next chapters when we get into more advanced unit tests:

Improvements

- Add unit tests also for events like click etc to test user interaction

Chapter 6 - Introducing Vex

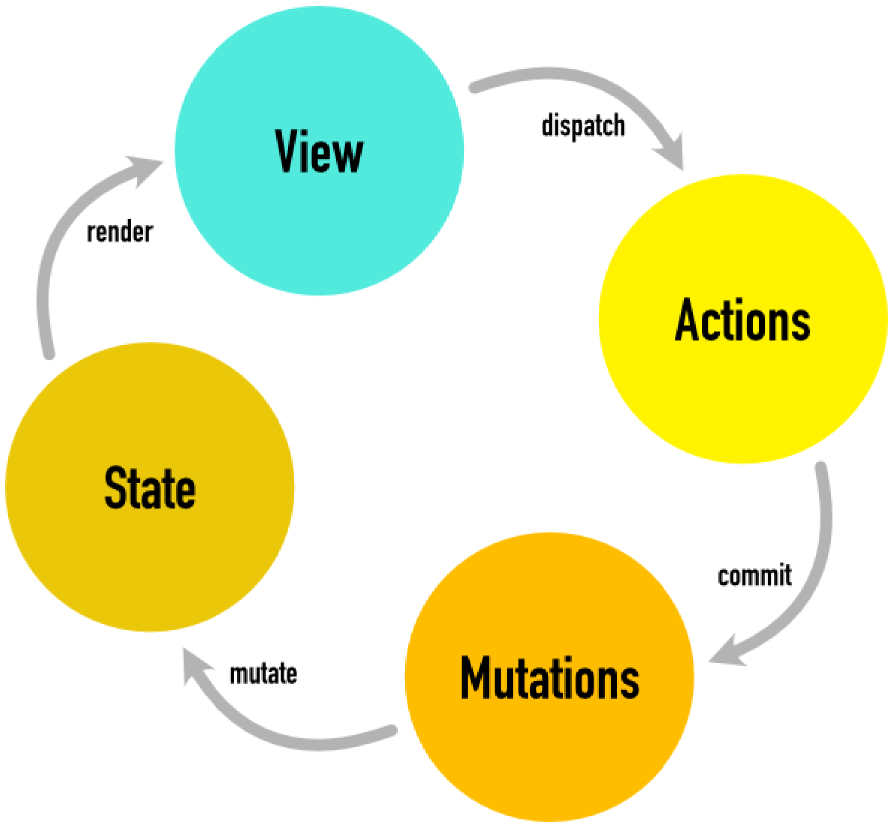
As mentioned at the end of the previous chapter, **Vuex** offer a **State Management Pattern** to keep views/state/actions separated and allow to manage state changes from a centralized place, the **store**, which becomes the **single source of truth** for our application state.

As long as we follow Vuex recommended patterns and read or modify the state only through Vuex actions and mutations, our application will always work. On the other hand, if we attempt to modify the state outside of Vuex actions and mutations, we might start experiencing problems.

There are also examples online where you can create your own state management mechanism. These too can be valid solutions. As a general rule, for applications that are set to grow a lot, it’s best to stick to Vuex’s functionality as it’s well tested and supported. Plus, if you need help, there are plenty of online resources for Vuex. Of course, in a real work situation you will need to evaluate and make the best decision on which approach to use.

Vuex Overview

The main concept around **Vuex** State Management Pattern is that of **one-way data flow** as illustrated in the following diagram:



- The View dispatch a request for an action to the store
- The action performs some work and then commit one or mutations
- The mutations change the state
- The view renders the updated state

Main Vuex Store (file: store/index.ts)

When we initially created our project in Chapter 1, the **vue-cli** also created a preliminary stub for the **Vuex** state in the file **src/store/index.ts**

Open this file. You can see that the the current code looks like this:

```
import { createStore } from 'vuex'

export default createStore({
  state: {
  },
  mutations: {
  },
  actions: {
  },
  modules: {
  }
})
```

Let’s add some code to implement a simple store that enables to keep track of our data and state and its mutations in one place.

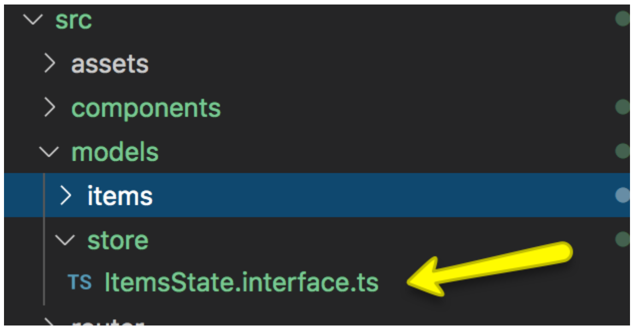
First, we need to add a property to hold our items, which will be an array of items. For this we could keep building on the stubbed code above:

```
    state: {
        items: []
    },
```

However, remember how in Chapter 4 we introduced the use of TypeScript interfaces to enforce type checking at development time and avoid possible run-time errors. In this book our focus is to build a strong foundation that can better serve building large-scale applications. So let’s take a step back and add an interface that will represent our entire state for the items list data and component.

Interface ItemsStateInterface

Create directory **src/models/store** and here add a file called **ItemsState.interface.ts**. Our directory structure should look like this:



Inside the **ItemsState.interface.ts** file, paste the following code:

```
import { ItemInterface } from '@/models/items/Item.interface'

/**
 * @name ItemsStateInterface
 * @description
 * Interface for the Items state
 */
export interface ItemsStateInterface {
  loading: boolean
  items: ItemInterface[]
}
```

As you can see in the code above, our state has a property called **items** and one called **loading**. The first one holds our data, the second one is used to give feedback to the user in the UI by showing loader/spinner image during the loading operation.

Back to the file: store/index.ts

Now we have to import a reference for the new interface into the **store/index.ts** file, and change our **state** instance so that uses our new **ItemsStateInterface**. Note that we also extracted the inline **state** instantiation from **createStore** into its own **const state** variable to make the code a bit more readable:

```
import { createStore } from 'vuex'
import { ItemInterface } from '@/models/items/Item.interface'
import { ItemsStateInterface } from './ItemsState.interface'
```

```
// our initial state:
const state: ItemsStateInterface = {
  loading: false,
  items: []
}

export default createStore({
  state: state,
  mutations: {
  },
  actions: {
  },
  modules: {
  }
})
```

Now we need to create some **actions** that will change our state. In **Vuex**, actions are simply functions that perform some tasks like loading/saving data from/to an API end-point and then commit the mutations to the state. First, we need an action that starts the loading of our data. We'll call this **loadItems**::

```
…
  mutations: {
  },
  actions: {
    loadItems({ commit, state }) {
      commit('loadingItems')
    }
  },
```

NOTE: in the next chapter we'll be replace hard-coded strings like 'loadingItems' with constant names in an organized way.

This action will commit a simple mutation called **loadingItems** (note the small difference: the **loadItems action** asks the store to load the data, while the **loadingItems mutation** sets the **loading** property to **true** to indicate that the operation is in progress). We need to add also a mutation function called **loadingItems** within the **mutations** section. Within the mutation, we can clear existing data from our **items** array and set the **loading** property to true. So add the following code within the mutations block:

```
…
  mutations: {
    loadingItems(state: ItemsStateInterface) {
      state.loading = true
      state.items = []
    }
  },
…
```

Now, we need to add more code within our action **loadItems** so we can load our mocked data. Let’s get the hard-coded data we initially stored within our **Home.vue** code and put it within our **loadItems** action. Here we call the variable **mockItems** and insert right after our **commit(‘loadingItems’)** line:

```
  actions: {
    loadItems({ commit, state }) {
      commit('loadingItems')

      // mock some data
      const mockItems: ItemInterface[] = [{
        id: 1,
        name: 'Item 1',
        selected: false
      }, {
        id: 2,
        name: 'Item 2',
        selected: false
      }, {
        id: 3,
        name: 'Item 3',
        selected: false
      }]
      …
```

Lets use a **setTimeout** to simulate a delay and commit a new mutation called **loadedItems** in which we also pass our **mockItems** data as the second argument:

```
    …

      // let's pretend we called some API end-point
      // and it takes 1 second to return the data
      // by using javascript setTimeout with 1000 for the milliseconds option
      setTimeout(() => {
        commit('loadedItems', mockItems)
      }, 1000)
    …
```

Then add a new function called **loadedItems** within the **mutations** block where we set the **state.items** to the loaded data, and set the **loading** property back to false (this indicates that the load items operation has completed):

```
…
mutations: {
    loadingItems(state: ItemsStateInterface) {
      state.loading = true
      state.items = []
    },
    loadedItems(state: ItemsStateInterface, items: ItemInterface[]) {
      state.items = items
      state.loading = false
    }
  },
…
```

The complete code including the new **loadingItems** mutation will look like this:

```
import { createStore } from 'vuex'
import { ItemInterface } from '@/models/items/Item.interface'
import { ItemsStateInterface } from './ItemsState.interface'

// our initial state:
const state: ItemsStateInterface = {
  loading: false,
  items: []
}

export default createStore({
  state: state,
  mutations: {
    loadingItems(state: ItemsStateInterface) {
      state.loading = true
      state.items = []
    },
    loadedItems(state: ItemsStateInterface, items: ItemInterface[]) {
      state.items = items
      state.loading = false
    }
  },
  …
```

```
  …
  actions: {
    loadItems({ commit, state }) {
      commit('loadingItems')
      // mock some data
      const mockItems: ItemInterface[] = [{
        id: 1,
        name: 'Item 1',
        selected: false
      }, {
        id: 2,
        name: 'Item 2',
        selected: false
      }, {
        id: 3,
        name: 'Item 3',
        selected: false
      }]

      // let's pretend we called some API end-point
      // and it takes 1 second to return the data
      // by using javascript setTimeout with 1000 for the milliseconds option
      setTimeout(() => {
        commit('loadedItems', mockItems)
      }, 1000)

    }
  },
  modules: {
  }
})
```

Home View

In the code within the file **src/views/Home.vue** we need to re-work a bit the **items** property. Remove the hard-coded data and change this to a **computed** property that just return the data from the **store state**:

```
<script lang="ts">
  import { defineComponent, reactive, computed } from 'vue'
  …

  export default defineComponent({
    …
    setup() {

      // begin: remove code
~~      const items: ItemInterface[] = reactive([{~~
~~        id: 1,~~
~~        name: 'Item 1',~~
~~        selected: false~~
~~      }, {~~
~~        id: 2,~~
~~        name: 'Item 2',~~
~~        selected: false~~
~~      }, {~~
~~        id: 3,~~
~~        name: 'Item 3',~~
~~        selected: false~~
~~      }])~~
      // end: remove code

      // begin: add code
      const items = computed(() => {
        return store.state.items
      })
      // end: add code

      return {
        items
      }
    })
…
```

Then dispatch our **loadItems** action in the view **mounted** event handler so it will ask our **Vuex** to load our items. The complete code for the **<script>** section is this:

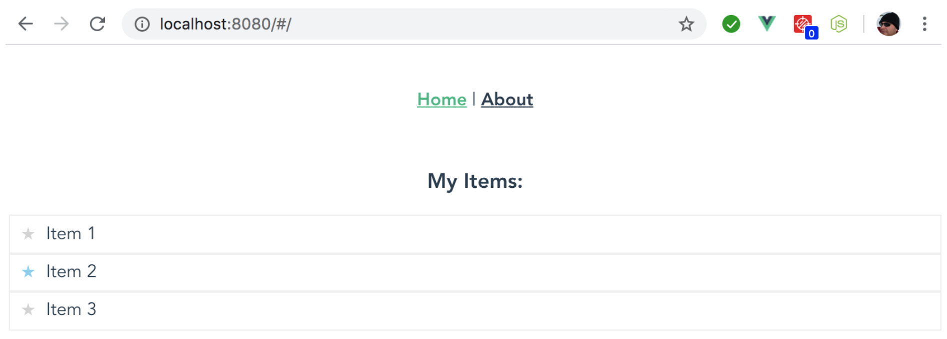
```
<script lang="ts">
  import { defineComponent, computed, onMounted } from 'vue' // <-- add
onMounted he\
re
  import store from '@/store'
  import ItemsListComponent from '@/components/items/ItemsList.component.vue'
  import { ItemInterface } from '@/models/items/Item.interface'

  export default defineComponent({
    name: 'Home',
    components: {
      ItemsListComponent
    },
    setup() {
      const items = computed(() => {
        return store.state.items
      })

      // begin: add code
      onMounted(() => {
        store.dispatch('loadItems')
      })
      // end: add code

      return {
        items
      }
    }
  })
</script>
```

When the user browses to our Home view, this will dispatch a request for the action **loadItems** to the **Vuex** store. In turn, the store will:

1. commit a **loadingItems** mutation where we clear the **items** array and set the **loading** property to true
2. load the items data and commit a **loadedItems** mutation where we set our **items** property to the data loaded, and set the **loading** property to false

Web Browser

Refresh the Home page on the web browser and notice that now it will take about 1 second before the items will be rendered. However, there is no way to tell what it's happening while the data is being loaded:



So let's enhance a bit more the **ItemsList.component.vue** code so we can start getting some feedback in the UI.

ItemsList.component.vue

Add a property called **loading** of type **boolean** to the **ItemsList** component:

```
export default defineComponent({
    components: {
      ItemComponent
    },
    props: {
      items: {
        type: Array as PropType<ItemInterface[]>
      },
      // begin: add code
      loading: {
        type: Boolean
      }
      // end: add code
    },

    …
```

Now within the **<h3>** element, add a one-way binding using the double curly braces to print out the value of the **loading** property:

```
<template>
  <div>
    <h3>Items - loading: {{ loading }}:</h3>
    …
```

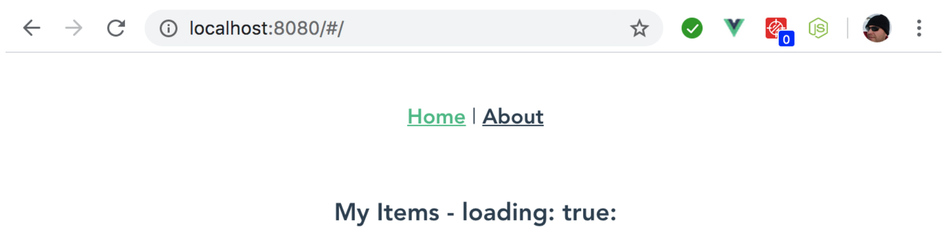
Home.vue

In the **Home** view, now we have to add a computed property called **loading** that returns the value from our **state.loading** and bind it to the :**loading** attribute of the **ItemsListComponent**:

```
<template>
  <div class="home">
    <ItemsListComponent
      :items="items"
      :loading="loading" /> <!-- add :loading="loading" -->
  </div>
</template>

<script lang="ts">
  …
  export default defineComponent({
    …
    setup() {
      const items = computed(() => {
        return store.state.items
      })

      // begin: add code
      const loading = computed(() => {
        return store.state.loading
      })
      // end: add code

      onMounted(() => {
        store.dispatch('loadItems')
      })

      return {
        items,
        loading // <-- add this
      }
    }
  })
</script>
```

Web Browser

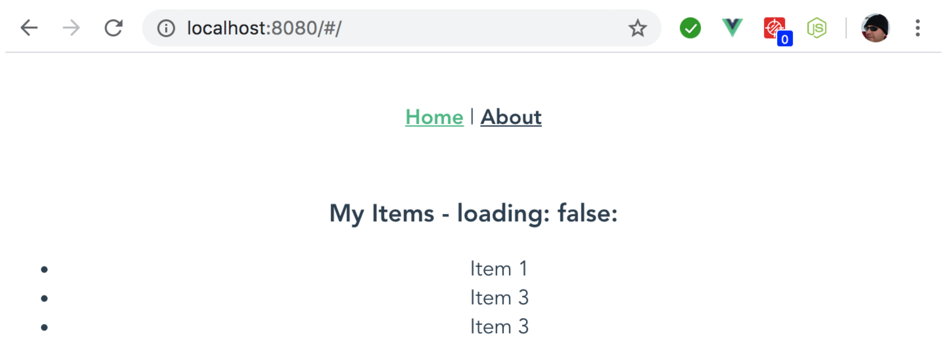
Now, when we refresh the browser, we'll first see a blank list, but in the header we'll see the text **My items - loading: true**:


localhost:8080/#/

Home | About

My Items - loading: true:

After 1 second the items will render and the h3 element will display the text **My items - loading: false**:


localhost:8080/#/

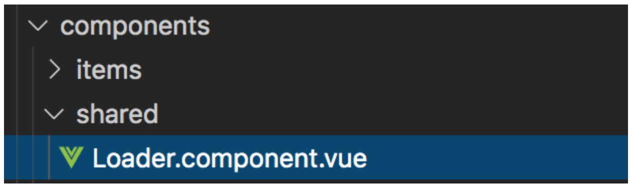
Home | About

My Items - loading: false:

- Item 1
- Item 3
- Item 3

Loader Component

Create the directory **src/components/shared**. Within this directory create a file called **Loader.component.vue**. Your directory structure should now look like this:



Within the **Loader.component.vue** file, paste the following code:

```
<template>
  <div class="loader" v-show="show">
    <div class="bounceball"></div>
  </div>
</template>

<script lang="ts">
  import { defineComponent } from 'vue'

  export default defineComponent({
    props: {
      show: {
        type: Boolean
      }
    }
  })
</script>
```

Also add this <style> section at the bottom of the file:

```
<style lang="scss">
  .loader {
    $color: #42b983;
    $width: 30px;
    $height: 30px;
    $bounceHeight: 60px;
    display: inline-block;

    .bounceball {
      position: relative;
      width: $width;
      &:before {
        position: absolute;
        content: '';
        top: 0;
        width: $width;
        height: $height;
        border-radius: 50%;
        background-color: $color;
        transform-origin: 50%;
        animation: bounce 500ms alternate infinite ease;
      }
    }
    …
```

```
    …

    @keyframes bounce {
      0% {
        top: $bounceHeight;
        height: 10px;
        border-radius: 60px 60px 20px 20px;
        transform: scaleX(2);
      }
      25% {
        height: $height;
        border-radius: 50%;
        transform: scaleX(1);
      }
      100% {
        top: 0;
      }
    }
  }
</style>
```

This provides a basic loader that uses pure CSS for the animation. You are free to use an animated **gif**, or **svg** image, or **font-icon** etc.

All the needed CSS (we are using SASS here, as you can see from the style **lang** attribute value **scss**) is contained within the same file[1].

ItemsList Component

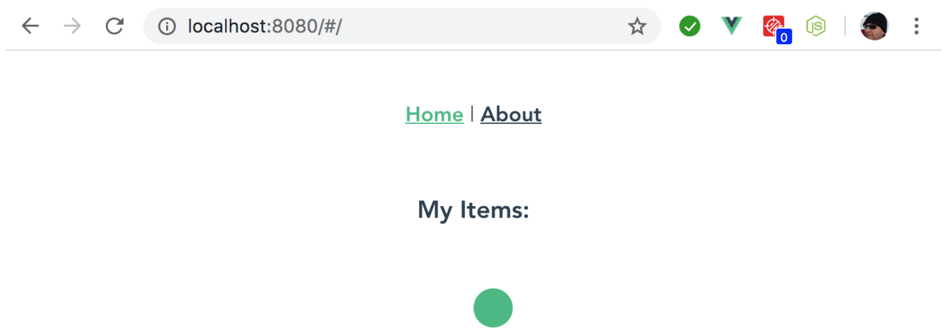
Now, lets go back into our **ItemsList.component.vue** code and import a reference to our new **Loader** component:

```
<script lang="ts">
  import { defineComponent, PropType } from 'vue'
  import { ItemInterface } from '@/models/items/Item.interface'
  import ItemComponent from './children/Item.component.vue'
  import Loader from '@/components/shared/Loader.component.vue' // <-- add
this line

  export default defineComponent({
    components: {
      ItemComponent,
      Loader // <-- add this line
    },

    …
```

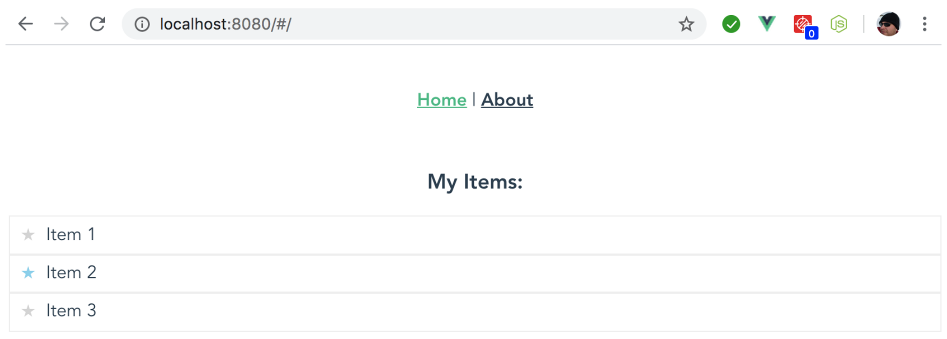
Within the **<template>** section add an instance of **<Loader>** and set its visibility through the **v-show** binding with the **loading** property. Also add a **v-show** binding to **!loading** to our **<ul>** element so that is not visible when **loading** is false, and vice versa:

```
<template>
  <div>
    <h3>My Items - loading: {{ loading }}:</h3>
    <Loader v-show="loading" /> <!-- add the Loader markup here binding v-show
to lo\
ading -->
    <ul v-show="!loading"> <!-- add v-show="!loading" -->
      <ItemComponent v-for="item in items"
        :key="item.id"
        :model="item"
        @select="onItemSelect" />
     </ul>
  </div>
</template>
```

Now refresh the web page you'll see our loader showing for about 1 second before it renders the items:



Then the loader will hide and the items list is rendered:



ItemsList Component - enhancing Item click handler

We need to do one more thing. Remember how in Chapter 4 our Item component did not exist yet and in the **ItemsList** component we were just updating each item **selected** property directly in our **onItemSelect** handler there? Here is the old code:

```
…
setup() {
  const onItemSelect = (item: ItemInterface) => {
    item.selected = !item.selected
    console.log('onItemSelect', item.id, item.selected)
  }

…
```

As a general rule, we should never change the application state/data outside our **Vuex** store, but only through the store actions/mutations. This means we need to change the code above to notify our parent view (Home.vue) that the user clicked on an element and wants to select an item. For this we emit a custom event called **selectItem**. Then the **Home.vue** code will receive this event and dispatch the **selectItem** action to our **Vuex** store.

Let’s change the code so we emit a custom event called **selectItem**. Remove the lines in red, and add the lines in yellow:

```
    setup({ emit }) { // <-- add { emit }
      const onItemSelect = (item: ItemInterface) => {
        item.selected = !item.selected // <-- remove this line
        console.log('onItemSelect', item.id, item.selected) <-- remove this
line
        emit('selectItem', item) <-- add this line
      }
```

Now in the **Home.vue** code, within the template section, we need to add a binding receive the **selectItem** event from the **ItemsListComponent** element:

```
<template>
  <div class="home">
    <ItemsListComponent
      :items="items"
      :loading="loading"
      @selectItem="onSelectItem" /> <!-- add @selectItem="onSelectItem" -->
  </div>
</template>
```

Within our **setup()** we need to add the receiving method called **onSelectItem**, and here we'll dispatch the action to our **Vuex** store passing an object parameter with the necessary values:

```
setup() {
      …
      // begin: add code
      const onSelectItem = (item: ItemInterface) => {
        store.dispatch('selectItem', {
          id: item.id,
          selected: !item.selected
        })
      }
      // end: add code

      return {
        items,
        loading,
        onSelectItem // <-- add this line
      }

      …
```

Note how we pass a parameter that is an anonymous object with two properties. The **item.id** value and the inverse of the current **item.selected** value (which we create by using the ! prefix to toggle the current value from false to true or vice-versa):

```
{
  id: item.id,
  selected: !item.selected
}
```

Now when you click on an item in the browser, it should log a **JavaScript error** in the console:

```
[vuex] unknown action type: selectItem
```

This is because we did not add a **selectItem** action to our **Vuex** store yet. So let’s do that.

Vuex Store: selectItem Mutation

Within the **store/index.ts** code we need to add a new mutation called **selectItem***.* We extract the **id** and **selected** values from our **params** object (using **JavaScript Destructuring**[2]), then find the item instance from our **state.items** (using **JavaScript Array.find**[3])**,** and finally update its **selected** property:

```
mutations: {
    …
    // begin: add code
    selectItem(state: ItemsStateInterface, params: {
      id: number,
      selected: boolean
    }) {
      const { id, selected } = params
      const item = state.items.find(o => o.id === id)
      if (item) {
        item.selected = selected
      }
    }
    // end: add code

    …
```

Vuex Store: selectItem Action

Add the **selectItem** action which will receive the **params** argument and just commit the **selectedItem** mutation which will take care of updating the state:

```
actions: {

    …,
    // begin: add code
    selectItem({ commit }, params: {
        id: number
        selected: boolean
      }) {
      commit('selectItem', params)
    }
    // end: add code
    …
```

Refresh the browser and click on the items, the **item.selected** property will be now updated through the **Vuex** state as recommended.

Congratulations on completing this chapter and learning how to use Vuex to centrally manage the application state. It’s a long chapter, the concepts outlined here require a lot of code to implement, and not everyone gets through it in a straightforward fashion the first time around. In the next chapters we will try to improve this code even more so arm yourself with a lot of patience!

Chapter 6 Recap

What We Learned

- How to use **Vuex** to manage our **Items** state
- How to create **Vuex actions** and **mutations**
- How to commit **state mutations** from within the **actions**
- How to **dispatch** (invoke) a state action
- How to use a **loading** property on our state to provide feedback to the user about long-running processes through a **loader** (animation)
- How to create a simple and reusable **Loader** component

Observations

- We are using hard-coded strings for our mutations names, which is considered a bad practice in a large-code base
- We are still using hard-coded data (**mockItems** within our **store/index.ts** file), instead of loading the data through an API client

Based on these observations, there are a few improvements we will make in the next chapters:

Improvements

- Replace the hard-coded string for the actions/mutations names and replace them with TypeScript constants
- Create an API client that can serve mocked data for quick front-end development and prototyping, and an API client that can communicate with real API end-points

Chapter 7 - Api Client

So far we have worked with **Vuex** and managed the app state through our **Vuex** store. However, we are still "pretending" to load data by using a **mockItems** variable with hard-coded data within our store **loadItems** action, and using the **setTimeout** trick to add a bit of delay before returning the data (so we have at least 1 second to show our Loader to the user).

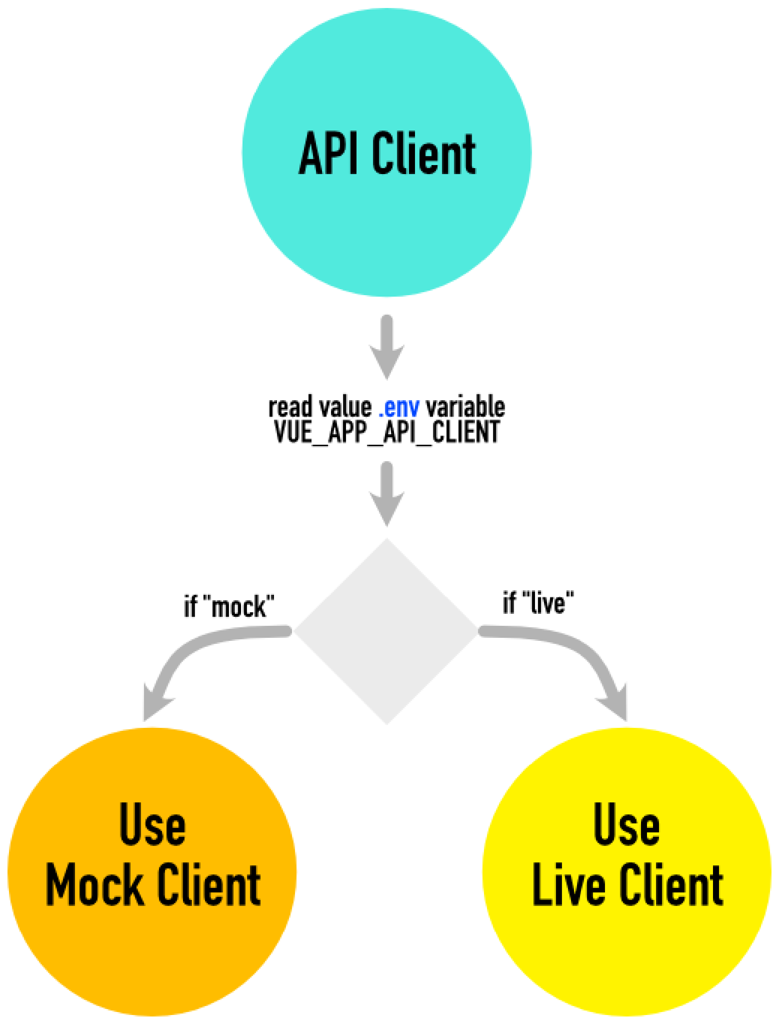
In the real world, we’ll be most likely writing a component that has to get the data from a serve API end-point. At the same time, we do not want to lose our ability to do quick prototyping and development of our front-end, even if the server API has not been developed yet. Now there are different ways of accomplishing this. Some people like to use mock data returned by a real API (there are packages and services out there that do just this[1]). Others prefer to have 2 different implementations for each API client, one that returns the mocked data (either by loading from disk or invoking a mocked API service), and one that returns the live data from the real server API. We’ll be implementing the latter pattern in this chapter so we have better control on our data and also have better control on different scenarios (TODO see our mocking of a successful or failing authentication in [chapter])

Another pattern is to create a separate API client for each area of our application. This will enable for better separation of concerns, avoid code cluttering, more easily write unit tests against each client. This is the pattern we’ll be following in this book, but remember this is not the only way to accomplish this. You should always evaluate what is the best solution for your specific requirements and evaluate that it fits your needs.

You should also read about **Domain Driver Design**, even though this book is not strictly following DDD principles, still the overall idea here is to try to keep code organized by application domain.

API Client Overview

Here is an overview of our API client architecture:



API Client module will read the env variable VUE_APP_API_CLIENT and there are two possible outcomes:

- when VUE_APP_API_CLIENT is Mock: it will return the Mock API Client
- when VUE_APP_API_CLIENT is Live: it will return the Live API Client

Domains

We'll create a global **ApiClient** that wraps additional clients organized by application domain. Our **ApiClient** will have for example a property called items which is the actual API client for the **Items** domain. As our application grows, we'll be adding more domains specific API clients.

Our goal is to eventually consume our API client code from our **Vuex** state in this way:

```
apiClient
  .items
  .fetchItems()
```

Here we have an instance of our main **ApiClientInterface**. We then access its **items** property which is the domain-specific API client (of type **ItemsApiClientInterface**) and call its methods or access its properties.

Later, if for example need to add a new **people** domain, we will add a **people** property to our main **ApiClientInterface** that points to an instance of **PeopleApiClientInterface**. Then we will be able to call its methods like this:

```
apiClient
  .people
  .fetchPeople()
```

As you can see, this makes the code much more concise and readable.

NOTE: This might seem to complicate things at first. However, remember that the scope of this book is to build a foundation for large-scale applications. Our primary goal is a solid code organization and structuring to avoid cluttering as the code might grow very large with many files.

The Main ApiClient

Create the directory **src/models/api-client**. Inside this directory, create the file **ApiClient.interface.ts** with the following code:

```
import { ItemsApiClientInterface } from './items'

/**
 * @Name ApiClientInterface
 * @description
 * Interface wraps all api client modules into one places for keeping code
organized.
 */
export interface ApiClientInterface {
  items: ItemsApiClientInterface
}
```

As you can see in the code above, our ApiClient will have a property called items of type **ItemsApiClientInterface,** which will be the API client specific to the **Items** domain.

Now let’s create the the **Items** API client.

Items Api Client

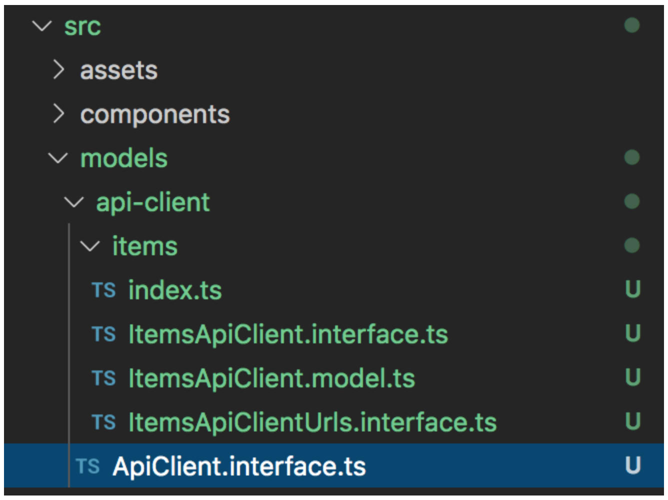
Now we create the interfaces and model that defines a domain-specific API client.

Create the directory **src/models/api-client/items**. Inside the **src/models/api-client/items** directory, create the following files:

- index.ts
- ItemsApiClient.interface.ts
- ItemsApiClient.model.ts

- ItemsApiClientUrls.interface.ts

Your directory structure will look like this:



```
src
  assets
  components
  models
    api-client
      items
        index.ts                          U
        ItemsApiClient.interface.ts       U
        ItemsApiClient.model.ts           U
        ItemsApiClientUrls.interface.ts   U
      ApiClient.interface.ts              U
```

Following is the the description and code for each of the files.

ItemsApiClientUrls.interface.ts

In order to avoid using hard-coded strings, and to enforce type-checking at development time, we'll be using interface **ItemsApiClientUrlsInterface** for the Urls/paths values that indicates the API end-points consumed by the **ItemsApiClient**. Here is the code to paste into **ItemsApiClientUrls.interface.ts**:

```
/**
 * @Name ItemsApiClientUrlsInterface
 * @description
 * Interface for the Items urls used to avoid hard-coded strings
 */
export interface ItemsApiClientUrlsInterface {
  fetchItems: string
}
```

ItemsApiClient.interface.ts

This is the interface for our **ItemsApiClient**. Our interface requires implementing a method called **fetchItems** the will return a list of items. Here is the code to paste into **ItemsApiClient.interface.ts**:

```
import { ItemInterface } from '@/models/items/Item.interface'

/**
 * @Name ItemsApiClientInterface
 * @description
 * Interface for the Items api client module
 */
export interface ItemsApiClientInterface {
  fetchItems: () => Promise<ItemInterface[]>
}
```

ItemsApiClient.model.ts

This is the model (class) for our **ItemsApiClient** which implements our Items API client interface.

For the initial version of this, we will be using a third-part open-source NPM package called **axios**. This is just a library that allows to make Ajax call in a much easier way. Let’s go back to the terminal, from within **my-vue3-project** directory, and install **axios** with the command:

```
npm install axios –save
```

NOTE: we will improve this even more later (see [TODO chapter]) to avoid having references to a third-party NPM package spread throughout the code.

Back to the editor, open **ItemsApiClient.model.ts** and start importing all the things we need:

```
import axios, { AxiosRequestConfig, AxiosError, AxiosResponse } from 'axios'

import { ItemsApiClientUrlsInterface } from './ItemsApiClientUrls.interface'
import { ItemsApiClientInterface } from './ItemsApiClient.interface'
import { ItemInterface } from '@/models/items/Item.interface'

…
```

And here is the class that implement our **ItemsApiClientInterface**:

```
/**
 * @Name ItemsApiClientModel
 * @description
 * Implements the ItemsApiClientInterface interface
 */
export class ItemsApiClientModel implements ItemsApiClientInterface {
  private readonly urls!: ItemsApiClientUrlsInterface

  constructor(urls: ItemsApiClientUrlsInterface) {
    this.urls = urls
  }

  fetchItems(): Promise<ItemInterface[]> {
    return new Promise<ItemInterface[]>((resolve) => {
      const url = this.urls.fetchItems

      // axios options
      const options: AxiosRequestConfig = {
        headers: {
        }
      }

      axios
        .get(url, options)
        .then((response: AxiosResponse) => {
            resolve(response.data as ItemInterface[])
        })
        .catch((error: any) => {
            console.error('ItemsApiClient: HttpClient: Get: error', error)
        })
    })
  }
}
```

index.ts (barrel file)

This just exports all our interfaces and models under items/ so that we can more easily import them later in other parts of the code:

```
export * from './ItemsApiClientUrls.interface'
export * from './ItemsApiClient.interface'
export * from './ItemsApiClient.model'
```

Mock and Live Api Clients

Now that we have defined our models for **ApiClientInterface** and **ItemsApiClientInterface**, let's implement a mechanism that will allow us to either use a

mock api-client that returns static **json** data, or a **live** api-client that returns data from as real API.

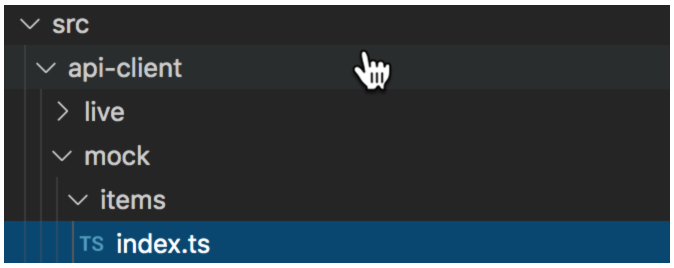
Under the **src** directory, create a sub-directory called **api-client**. Within this directory create two new sub-directories called:

- **mock** (this will contain our mock implementations to return static **json** data)
- **live** (this will contain the implementation that call the real API end-points)

We'll be writing two implementations of our **ApiClientInterface** and its child **ItemsApiClientInterface**. One for the mock and one for the live API.

Mock Api Client

Let's start with the mock first. Within the **mock** directory, add a child directory called **items**, and within that one create a new file named **index.ts**. Your directory structure should look like this:



Inside the **src/api-client/mock/items/index.ts** file, paste the following code:

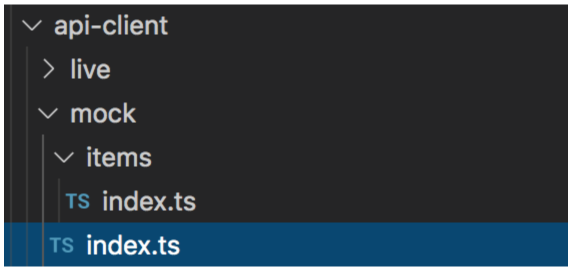
```
import {
  ItemsApiClientUrlsInterface,
  ItemsApiClientInterface,
  ItemsApiClientModel
} from '@/models/api-client/items'

const urls: ItemsApiClientUrlsInterface = {
  fetchItems: '/static/data/items.json'
}

// instantiate the ItemsApiClient pointing at the url that returns static json
mock \
data
const itemsApiClient: ItemsApiClientInterface = new ItemsApiClientModel(urls)
// export our instance
export default itemsApiClient
```

Here we import all our interfaces and models, then we instantiate a variable called **urls** of type **ItemsApiClientUrlsInterface** that holds the path to the API end-point that returns the **Items** data. In this case, since this is the mock implementation, we will point to some static **json** file with the mock data. Note that we have only **fetchItems**, but we could have multiple end-points. For now we'll focus only on returning data. Later, in more advanced chapter I'll show you how to do something similar for CRUD operations.

We then create an instance of our **ItemsApiClient** class by passing our **urls** instance into the constructor (as you can see, later in our live implementation we'll pass an instance of ItemsApiClientUrlsInterface that contains the paths/urls to the real end-points)

Finally, we just export our instance called **itemsApiClient**.

Now let's move one directory up, under **src/api-client/mock** and create another **index.ts** file here. Your directory structure should look like this:



Inside the **src/api-client/mock/index.ts** file, paste the following code:

```
import { ApiClientInterface } from '@/models/api-client/ApiClient.interface'
import itemsApiClient from './items'

// create an instance of our main ApiClient that wraps the mock child clients
const apiMockClient: ApiClientInterface = {
  items: itemsApiClient
}
// export our instance
export default apiMockClient
```

This is the mock implementation of our main ApiClient that wraps that items client.

Here we import our **ApiClientInterface** interface, and our mock instance of **ItemsApiClient**. We then create an instance of our **ApiClientInterface** that is called **apiMockClient** because it will use the mock implementation of the **ItemsApiClient**.

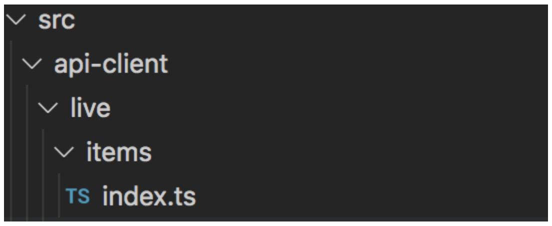
Live Api Client

Similar to what we did with our mock api client, we'll be implementing the live api client now. Note that the **live** directory structure will be the same as the **mock** directory structure.

*NOTE: Here we'll be following similar steps as per the mock one. Alternatively, you can copy all the files under **api-***

client/mock *to* ***api-client/live*** *and make the necessary changes.*

Create directory **src/api-client/live/items** and here add a new file named **index.ts**. Your directory structure should look like this:

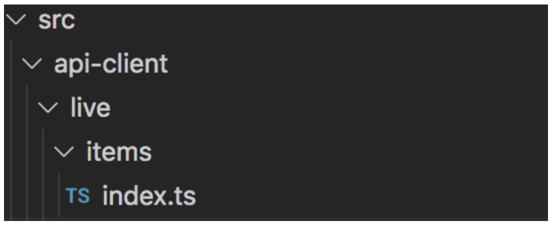


Inside the **src/api-client/live/items/index.ts** file, paste the following code:

```
import {
  ItemsApiClientUrlsInterface,
  ItemsApiClientInterface,
  ItemsApiClientModel
} from '@/models/api-client/items'

const urls: ItemsApiClientUrlsInterface = {
  fetchItems: '/static/data/items.json'
}

// instantiate the ItemsApiClient pointing at the url that returns static json
mock \
data
const itemsApiClient: ItemsApiClientInterface = new ItemsApiClientModel(urls)
// export our instance
export default itemsApiClient
```

NOTE: this code is almost exactly the same as the mock client. The only difference is the ***fetchItems*** *property that here says for now "/path/to/your/real/api/and-point". You'll replace this with the actual value of your real server API end-point url/path. If you do not have one yet, leave the current value as a place holder and updated once in the future you'll have your server API ready.*

Now let’s move one directory up, under **src/api-client/live** and create another **index.ts** file here. Your directory structure should look like this:



Inside the **src/api-client/live/index.ts** file, paste the following code:

```
import { ApiClientInterface } from '@/models/api-client/ApiClient.interface'
import itemsApiClient from './items'

// create an instance of our main ApiClient that wraps the live child clients
const apiMockClient: ApiClientInterface = {
  items: itemsApiClient
}
// export our instance
export default apiMockClient
```

This code is also almost identical to the related mock **index.ts** file. The only exceptions are:

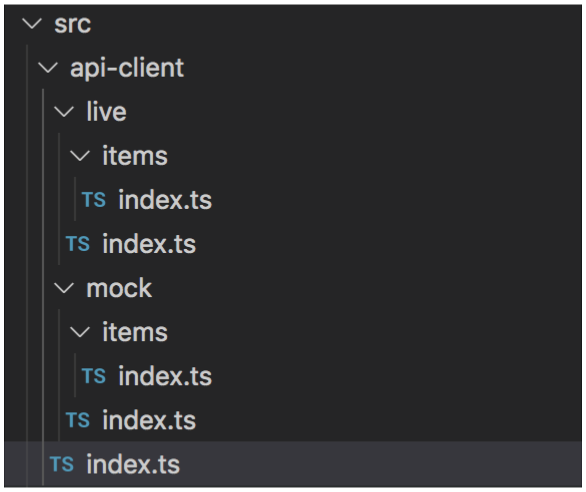
1. We use the live **ItemsApiClient** from **api-client/live-items**
2. We name the instance **apiLiveClient** for more clarity

We then just export our **apiLiveClient** instance.

Now we need one final **index.ts** that will server our main API client factory and return either the **mock** or the **live** API client based on an environment variable (later you might find easier to drive this with different configuration files).

Api Client Factory

Let’s move up one more directory. Under **src/api-client** let’s add another **index.ts** file. Your directory structure should look like this:


```
src
  api-client
    live
      items
        index.ts
      index.ts
    mock
      items
        index.ts
      index.ts
    index.ts
```

Inside the **src/api-client/index.ts** file, start by importing a reference to our **ApiClientInterface** interface, and both the instances for the **mock** and the **live** clients:

```
import { ApiClientInterface } from '@/models/api-client/ApiClient.interface'
import apiMockClient from './mock'
import apiLiveClient from './live'
```

Now we will add some code that will export either the **mock** or **live** clients based on an environment variable. The complete code will look like this:

```
import { ApiClientInterface } from '@/models/api-client/ApiClient.interface'
import apiMockClient from './mock'
import apiLiveClient from './live'

let env: string = 'mock'
if (process.env && process.env.VUE_APP_API_CLIENT) {
  env = process.env.VUE_APP_API_CLIENT.trim()
}
// return either the live or the mock client
let apiClient: ApiClientInterface
if (env === 'live') {
    apiClient = apiLiveClient
} else {
    apiClient = apiMockClient
}
```

```
export default apiClient
```

Depending on your TypeScript and **ESlint** configurations, you might have to add also a declaration for the process.env types within the **src/shims-vue.d.ts** file:

```
declare interface process {
  env: {
    VUE_APP_API_CLIENT: string
  }
}
```

Finally, we have to create an **.env** file that will store our default environment variables. Create this file under the root of your project directory and add the following content:

```
VUE_APP_API_CLIENT=mock
```

We also need to also update the **scripts** section within the **package.json** file so it will correctly set the expected environment variables when running locally for development with **npm run serve**, or when building for production with **npm run build**. The current content of your script section should be like this:

```
  "scripts": {
    "serve": "vue-cli-service serve",
    "build": "vue-cli-service build",
    "test:unit": "vue-cli-service test:unit"
  },
```

Change the **serve** command to:
*(Note: remember we are using **cross-env** npm package here that works across different operating systems. See the end of Chapter 1 for more details)*

```
"serve": "cross-env VUE_APP_API_CLIENT=mock vue-cli-service serve --mode
development\
",
```

Change the **build** command to:

```
"build": "cross-env VUE_APP_API_CLIENT=live vue-cli-service build --mode
production",
```

Optional: You could also add a **build-mock** command that uses the mock api client, if you are do not plan to have a real API in your project, or maybe to test new front-end functionality in production when the server API is not yet ready:

```
"build-mock": "cross-env VUE_APP_API_CLIENT=mock vue-cli-service build --mode
produc\
tion",
```

Similarly, you could also have a serve-mock or serve-live, with the environment variables set as you wish.

Vuex Store Instance updates

Back into our **store/index.ts** code, we can now finally remove the reference to the hard-coded data and use our new API client to retrieve these data. Start by adding an import for our **apiClient** (note how we no longer have to worry about using the **mock** or the **live** one, the system we'll handle that automatically based on the **VUE_APP_API_CLIENT** environment variable we created earlier):

```
import { createStore } from 'vuex'
import { ItemInterface } from '@/models/items/Item.interface'
import { ItemsStateInterface } from '@/models/store/ItemsState.interface'
import apiClient from '@/api-client'
```

Then, within the **loadItems** action, remove the hard-coded **mockItems** variable and its data. Then remove the **commit('loadedItems', mockItems)** line within the **setTimeout** and replace it with a call to our **apiClient.items.fetchItems** and this time commit **loadedItems** with the **data** returned by our **fetchItems**:

```
actions: {
    loadItems({ commit, state }) {
      commit('loadingItems')

                        // begin: remove code
      ~~// mock some data~~
      ~~const mockItems: ItemInterface[] = [{~~
        ~~id: 1,~~
        ~~name: 'Item 1',~~
        ~~selected: false~~
      ~~}, {~~
        ~~id: 2,~~
        ~~name: 'Item 2',~~
        ~~selected: false~~
      ~~}, {~~
        ~~id: 3,~~
        ~~name: 'Item 3',~~
        ~~selected: false~~
      ~~}]~~
      // end: remove code

      // let's pretend we called some API end-point
      // and it takes 1 second to return the data
      // by using javascript setTimeout with 1000 for the milliseconds option
      setTimeout(() => {
        commit('loadedItems', mockItems) <!-- remove line -->
        // begin: add code
        apiClient
          .items
          .fetchItems().then((data: ItemInterface[]) => {
            commit('loadedItems', data)
          })
       // end: add code
      }, 1000)

    },
    ...
```

We also need to create data folder from where our **mock** api-client will load the static **json** files.

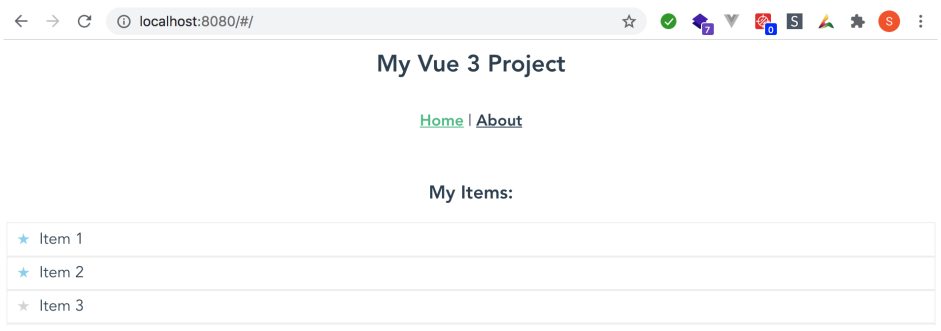
If you remember, earlier during our Mock Api Client implementation we set the **urls fetchItems** end-point path to be **/static/data/items.json**.

We need to create a directory called **static** under our **public** folder, because that is what **Vue** consider our root directory when running the application. And within the static directory create one called **data**. Here create a file called **items.json**.

Within the **items.json** files and paste in the following data:

```
# File: public/static/data/items.json:
[{
  "id": 1,
  "name": "Item 1",
  "selected": false
}, {
  "id": 2,
  "name": "Item 2",
  "selected": false
}, {
  "id": 3,
  "name": "Item 3",
  "selected": false
}]
```

Make sure there are no errors in the terminal. If needed stop it with **CTRL-C** and run again with **npm run serve**. The browser should display a loader, then render our items list as before:



Alternatives

There are other ways in which you could use a mocked API. There are services or libraries out there that can help you build a mocked API, and you could simplify the code here by having only one **apiClient** that uses either mock or live API end-points based on environment variables only etc.

I opted to show you how you can do this using static **.json** files that are located in the same project under **public/static/data** as this gives you a lot of flexibility to play around with different things when you are starting out.

The other thing is that by having a specific implementation of the mock **apiClient** you do not have to necessarily return the .json files, but you could simulate fake responses or pretend to have saved or delete an item without actually modifying any static data (so it will be just in memory, and when you refresh the web browser the data would bd reload as in its original state).

I will leave it up to you to explore the alternatives as you see fit.

Chapter 7 Recap

What We Learned

- How to implement an **apiClient** that automatically can serve either mock or real data
- How to continue enforcing type checking at development time with TypeScript interfaces and models
- How to structure directories and files in an organized way
- How to invoke our api client from the Vuex store

Observations

- We have a reference to a third NPM package (**axios**) in our **ItemsApiClient** mode and if we keep following this pattern we'll keep polluting new api client implementations for different areas with references to this NPM package in several parts of our code. This will cause a build up in technical debt that will make it harder to later replace **axios** with something else one day we'll have to. This might happen either because **axios** will no longer be supported, or maybe better NPM packages will be available that we want o use in its place. Either way, we should structure our code in a way so that we can more easily replace **axios** with something else without having to change a lot of code in too many places.
- Our **Vuex** store is still contained all in one file, **store/index.ts**, and this will quickly become cluttered as we add more properties, actions, mutations. We need a better way to structure our **Vuex** store so that can scale up in a cleaner and more maintainable way.

Based on these observations, there are a few improvements that will be making into the next two chapters:

Improvements

- Create an **HttpClient** model that implements an **HttpClientInterface** where we can encapsulate the reference to **axios** all in one place and make it easier to change later if we find the need to use a different NPM package.
- Split the **Vuex** store into modules so that code can be more easily maintained and unit tested.

Chapter 8 - Enhance the Api Client

From the previous chapter recap, we observed that the **ItemsApiClient** contains hard-coded references to the **axios** NPM package. We understand that is not a good practice to follow as, when adding more API clients, we do not want to have references to a 3rd party NPM packages spread throughout our code. What we need to do is abstract the generic client get/post/etc methods into their own implementation that we can then consume from our **ItemsApiClient** and future API clients implementations that we'll be adding later.

There are multiple ways we could do this. You could create an Http client factory, use dependency injection, but it should be enough to just wrap our calls done with **axios** in one place, within an **HttpClient** class. If later we have to switch to a different NPM package, all the code that needs to be updated will be inside this class and as long as we do not change the signature of our HttpClient get/post/etc methods, everything should still work as before.

HttpClient Interfaces and Models

Create the directory **src/models/http-client**. Within this directory, create the following files

HttpClient.ts

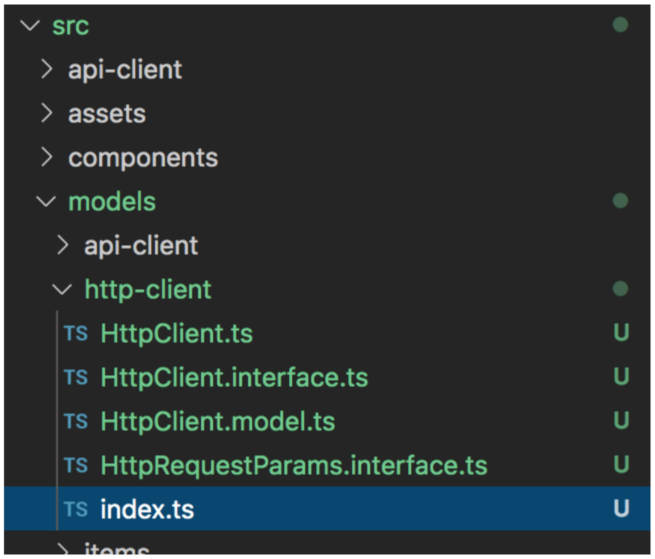
HttpClient.interface.ts

HttpClient.model.ts

HttpRequestParams.interface.ts

index.ts

Your directory structure will look like this:



Following is the the description and code for each of the files.

HttpRequestParams.interface.ts

The **HttpRequestParamsInterface** will allow us to pass parameters to the HttpClient methods. These are things like the API end point **url**, an optional **payload** (if POST or PUT), and a flag that indicates if the request requires an authentication token.

```
/**
 * @name HttpRequestParamsInterface
 * @description
 * HttpClient requests parameters for get/post/put etc operations
 */
export interface HttpRequestParamsInterface {
  url: string
  requiresToken: boolean
  payload?: any
}
```

*NOTE: If you want to be even more strict, you could make the payload a generic type by changing the interface declaration to **HttpRequestParamsInterface <T | any | undefined>** and the property to **payload?: T | any | undefined***

HttpClient.interface.ts

The **HttpClientInterface** is a generic interface that will define the methods out **HttpClient** will have to implement (i.e. **get/post/put/delete** etc):

```
import { HttpRequestParamsInterface } from './HttpRequestParams.interface'

/**
 * @Name HttpClientInterface
 * @description
 * Interface for our HttpClient wrapper
 */
export interface HttpClientInterface {
  get<T>(parameters: HttpRequestParamsInterface): Promise<T>
  post<T>(parameters: HttpRequestParamsInterface): Promise<T>
}
```

(Note: you can later refactor this and just use one method called request<T> for all type of requests if you wish)

HttpClient.model.ts

The **HttpClientModel** is the class that implements our **HttpClientInterface** methods (for now just **get** and **post**). Since the code here is longer, let me split in multiple parts.

First the import section:

```
import axios, {
  AxiosRequestConfig,
  AxiosResponse
} from 'axios'

import { HttpRequestParamsInterface } from './HttpRequestParams.interface'
import { HttpClientInterface } from './HttpClient.interface'
...
```

First part of the class and its constructor:

```
/**
 * @name HttpClientModel
 * @description
 * Wraps http client functionality to avoid directly using a third party npm
package\
 like axios
 * and simplify replacement in the future if such npm package would stop being
devel\
oped or other reasons
 */
export class HttpClientModel implements HttpClientInterface {
  private getToken(): string {
    const TOKEN_KEY =
      process.env && process.env.VUE_APP_TOKEN_KEY
        ? process.env.VUE_APP_TOKEN_KEY
        : 'myapp-token'
    const token = localStorage.getItem(TOKEN_KEY) || ''
    return token
  }

  constructor() {
    // OPTIONAL for now: Add request interceptor to handle errors or other
things fo\
r each request in one place
  }

  …
```

The **get<T>** implementation:

```
  …

  get<T>(parameters: HttpRequestParamsInterface): Promise<T> {
    return new Promise<T>((resolve, reject) => {
      const { url, requiresToken } = parameters

      // axios options
      const options: AxiosRequestConfig = {
        headers: {}
      }

      if (requiresToken) {
        const token = this.getToken()
        options.headers.RequestVerificationToken = token
      }

      axios
        .get(url, options)
        .then((response: AxiosResponse) => {
          resolve(response.data as T)
        })
        .catch((response: AxiosResponse) => {
          console.info('------ rejecting ----')
          reject(response)
        })
    })
  }

  …
```

The **post<T>** implementation and end of file:

```
  …

  post<T>(parameters: HttpRequestParamsInterface): Promise<T> {
    return new Promise<T>((resolve, reject) => {
      const { url, requiresToken, payload } = parameters

      // axios options
      const options: AxiosRequestConfig = {
        headers: {}
      }

      if (requiresToken) {
        const token = this.getToken()
        options.headers.RequestVerificationToken = token
      }

      axios
        .post(url, payload, options)
        .then((response: AxiosResponse) => {
          resolve(response.data as T)
        })
        .catch((response: AxiosResponse) => {
          reject(response)
        })
    })
  }
}
```

*NOTE: we added a preliminary implementation of the private **getToken** method that retrieves a token from local storage. For now this does not do anything, but when we'll get to consume a real API, we could store an authentication or JWT token into local storage during the login process with the same key as in VUE_APP_TOKEN_KEY. We also need to add this new env variables to our shims-vue.d.ts like we did for VUE_APP_API_CLIENT.*

HttpClient.ts

This file contains the export of a single instance of our **HttpClientModel** as **HttpClient**. This is what we'll be consuming in our API client:

```
import { HttpClientInterface } from './HttpClient.interface'
import { HttpClientModel } from './HttpClient.model'
```

```
// export instance of HttpClientModel
export const HttpClient: HttpClientInterface = new HttpClientModel()
```

index.ts

This just exports all our interfaces and models under **http-client/** so that we can more easily import them later in other parts of the code:

```
export * from './HttpRequestParams.interface'
export * from './HttpClient.interface'
export * from './HttpClient.model'
export * from './HttpClient'
```

HttpClient Unit Tests

Now we need to first add unit tests against our newly create **HttpClient** before we can re-factor the **ItemApiClient** code to use it in place of **axios**.

Start by installing an **NPM** package called **sinon** (and its TypeScript types) which we'll use for start using **mocks** and **stubs**[1]:

```
npm install --save-dev sinon @types/sinon
```

Then create the directory **tests/unit/http** and within this directory add a new file called **MockedPromiseFactory.ts**. Within the file, let's implement a factory that returns a mocked **promise**[2] that we can use as an helper for our unit tests:

```
import axios, { AxiosRequestConfig, AxiosResponse } from 'axios'

export interface MockedPromiseFactoryParamsInterface {
  url: string
  requestConfig: AxiosRequestConfig
  statusCode: number
  statusText: string
  data: any
  reject: boolean
}

export const MockedPromiseFactory = (
  params: MockedPromiseFactoryParamsInterface
): Promise<AxiosResponse<string>> => {
  return new Promise<AxiosResponse<string>>((resolve, reject) => {
    setTimeout(() => {
      const response: AxiosResponse = {
        data: params.data,
        status: params.statusCode,
        statusText: params.statusText,
        headers: [],
        config: params.requestConfig
      }

      if (params.reject) {
        reject(response)
      } else {
        resolve(response)
      }
    }, 100)
  })
}
```

This allows us to avoid code duplication in the unit tests as will mock both successful and unsuccessful responses.

Note how we resolve the promise by default, but if the **params.reject** is true we reject it. This gives us the option to simulate a **Promise** that will reject.

Testing a successful "get" response

Still within the **tests/unit/http** directory add a new file called **HttpClient.get.200.spec.ts**. Within the file, start by importing all the things we need:

```
import { expect } from 'chai'
import sinon from 'sinon'
import axios, { AxiosRequestConfig } from 'axios'
import { HttpClient, HttpRequestParamsInterface } from '@/models/http-client'
import { MockedPromiseFactory } from './MockedPromiseFactory'
...
```

Then create an instance of **HttpRequestParamsInterface**:

```
...
const mockParams: HttpRequestParamsInterface = {
  url: 'path/to/a/get/api/endpoint',
  requiresToken: false
}
...
```

Then let’s start writing the core of our unit tests by adding a **before** and **after** function (these are special **mocha** hook functions that will be run before the actual unit tests[3]). Within the **before** we will create an instance of **AxiosRequestConfig**, then use our **MockedPromiseFactory** to create a fake promise, then stub the **axios.get** call with **sinon** so that it will return our mockedPromise. Within the **after**, we just call **sinon.restore()** to clear our mocks and stubs:

```
...
describe('HttpClient.get', () => {
  before(() => {
    const mockedRequestConfig = {
      headers: {}
    } as AxiosRequestConfig

    const mockedPromise = MockedPromiseFactory({
      url: mockParams.url,
      statusCode: 200,
      statusText: 'Success',
      requestConfig: mockedRequestConfig,
      data: 'get completed',
      reject: false
    })

    // since HttpClient uses axios internally, stub axios here
    sinon
      .stub(axios, 'get')
      .withArgs(mockParams.url, mockedRequestConfig)
      .returns(mockedPromise)
  })

  after(() => {
    sinon.restore()
  })
...
```

Now, let’s write the unit test that invokes the **httpClient.get** method passing the **mockParams** and check that the response equals our expected value:

```
...
  it('should succeed and return data', (done) => {
    HttpClient.get<string>(mockParams)
      .then((response: string) => {
        expect(response).to.equal('get completed')
        done()
      })
  })
...
```

We can now add even more tests for different conditions and be able to use our **MockedPromiseFactory** to simulate the return of different **http** responses.

Testing an unsuccessful "get" response

Create a file called **HttpClient.get.400.spec.ts**. Within the file, start by importing all the things we need:

```
import { expect } from 'chai'
import sinon from 'sinon'
import axios, { AxiosRequestConfig } from 'axios'
import { HttpClient, HttpRequestParamsInterface } from '@/models/http-client'
import { MockedPromiseFactory } from './MockedPromiseFactory'
...
```

Then, create an instance of **HttpRequestParamsInterface**:

```
...
const mockParams: HttpRequestParamsInterface = {
    url: 'path/to/a/get/api/endpoint',
    requiresToken: false
}
...
```

This time we stub a **400** response within the **before** block (note the **reject** parameter set to true):

```
...
describe('HttpClient.get', () => {
  before(() => {
    const mockedRequestConfig = {
        headers: {
        }
    } as AxiosRequestConfig

    const mockedPromise = MockedPromiseFactory({
        url: mockParams.url,
        statusCode: 400,
        statusText: 'Error',
        requestConfig: mockedRequestConfig,
        data: 'get completed with errors',
        reject: true
    })

    // since HttpClient uses axios internally, stub axios here
    sinon.stub(axios, 'get')
      .withArgs(mockParams.url, mockedRequestConfig)
      .returns(mockedPromise)
  })

  after(() => {
    sinon.restore()
  })
...
```

Again write a unit test that invokes the **httpClient.get** method passing the **mockParams** but this time also add a **catch** block as we are expecting our mocked promise to reject and return a **response.status** to be **400**:

```
...
  it('should reject and return 400', (done) => {
    HttpClient.get<string>(mockParams)
      .then(() => {
        // should not get in here
      })
      .catch((response: AxiosResponse) => {
        expect(response.status).to.equal(400)
        done()
      })
  })
...
```

*NOTE: In a similar way, you could also write the code to test the post method with files **HttpClient.post.200.spec.ts** and **HttpClient.post.400.spec.ts** etc.*

We can now finally change our **ItemApiClient** so it uses our newly implemented HttpClient instead of axios.

ItemsApiClientModel Update

Open the file **src/models/api-client/items/ItemsApiClient.model.ts**.\
Remove the import **axios** line and replace it with an import for our **HttpClient** instance and the **HttpRequestParamsInterface**:

```
import axios, { AxiosRequestConfig, AxiosError, AxiosResponse } from 'axios'
// <-- \
remove this line
import { HttpClient, HttpRequestParamsInterface } from '@/models/http-client'
// <--\
 add this line
```

Then change the **fetchItems** implementation but removing the whole blocks that returns the new **Promise**:

```
fetchItems(): Promise<ItemInterface[]> {
    // begin: remove code
    ~~return new Promise<ItemInterface[]>(resolve => {~~
      ~~const url = this.urls.fetchItems~~

      ~~// axios options~~
      ~~const options: AxiosRequestConfig = {~~
        ~~headers: {}~~
      ~~}~~

      ~~axios~~
        ~~.get(url, options)~~
        ~~.then((response: AxiosResponse) => {~~
          ~~resolve(response.data as ItemInterface[])~~
        ~~})~~
        ~~.catch((error: AxiosResponse) => {~~
          ~~console.error('ItemsApiClient: HttpClient: Get: error', error)~~
        ~~})~~
    ~~})~~
    // end: remove code
  }
```

Replace it with the following code. This creates a const variable to hold our **HttpRequestParamsInterface** parameters, and then return the call to **httpClient.get** (which is already a Promise, so we do not have to do anything else here):

```
fetchItems(): Promise<ItemInterface[]> {
    // begin: add code
    const getParameters: HttpRequestParamsInterface = {
      url: this.urls.fetchItems,
      requiresToken: false
    }

    return HttpClient.get<ItemInterface[]>(getParameters)
    // end: add code
  }
```

Now, make sure there are no errors in the terminal and the browser refreshes correctly and load the data correctly.

Chapter 8 Recap

What We Learned

- How to abstract and encapsulate functionality into a **wrapper** class that ultimately uses a 3rd party package (our **HttpClient** wraps all the code consuming **axios** in one place)
- How to use **sinon** for stubs and a helper factory that returns a **Promise** to simulate different kinds of **Http** responses so we can unit tests different scenarios.

Observations

- We did not write unit tests against the **HttpClient** "post" method
- We did not write unit tests against the ItemsApiClientModel

Based on these observations, there are a few improvements that you could make, but we’ll not be doing them as part of this book:

Improvements

- Add unit tests against the **HttpClient** post method as well
- Add unit tests against the **ItemsApiClient** methods as well

Chapter 9 - Modularize the Vuex Store

Most basic examples and articles you can find on the web about **Vuex** usually show only how to create a monolithic store that contains all the app state/mutations/actions. All this code is contained in the same **store/index.ts** file. This is the same pattern we have followed so far, as we kept building on the default code stubbed initially by the **vue-cli** when we created the project.

We will now break apart our **store** into **modules** and show how we can better separate concerns and keep code that is related to one area of our application in one place. We will do this by using multiple sub-directories under the directory **src/store** and following our established naming conventions.

While doing this re-factoring, we'll also add unit testings against our store so we can validate our changes as we break the current code into modules.

NOTE: this is going to be a longer chapter and you'll be adding quite a few files and interfaces/models/namespaces. If you encounter any trouble, always refer to the GitHub[1] *code companion to this book, or feel free to reach out to me on Twitter with questions or comments.*

Current Vuex Store instance (src/store/index.ts)

In **Chapter 6** we expanded on top of the default code that the **vue-cli** initially stubbed when we first created our project. We added **actions** and **mutations** to load our items and to select/unselect items.

The code looks more or less like this (I omitted the functions within actions and mutations to keep the example short here):

```
import { createStore } from 'vuex'
import { ItemInterface } from '@/models/items/Item.interface'
import { ItemsStateInterface } from './ItemsState.interface'
import apiClient from '@/api-client'

// our initial state:
const state: ItemsStateInterface = {
  loading: false,
  items: []
}

export default createStore({
  state: state,
  mutations: {
    …
  },
  actions: {
    …
  },
  modules: {}
})
```

Currently, we have the code all in one place, and the mutations and actions here revolve around changing (or if you prefer, mutating) the **ItemsStateInterface** only.

In a large code base that will have many high-level components, this approach would lead to a gigantic **store/index.ts** file with thousands of lines of code. This will make it very hard to read, maintain, troubleshoot and unit tests against it.

The solution is to modularize our store. Thankfully, **Vuex** provides already functionality to do that. We have to only make some decisions on how to structure our files and code and follow these standard consistently as we are doing for the rest of the application code.

Store and State Models

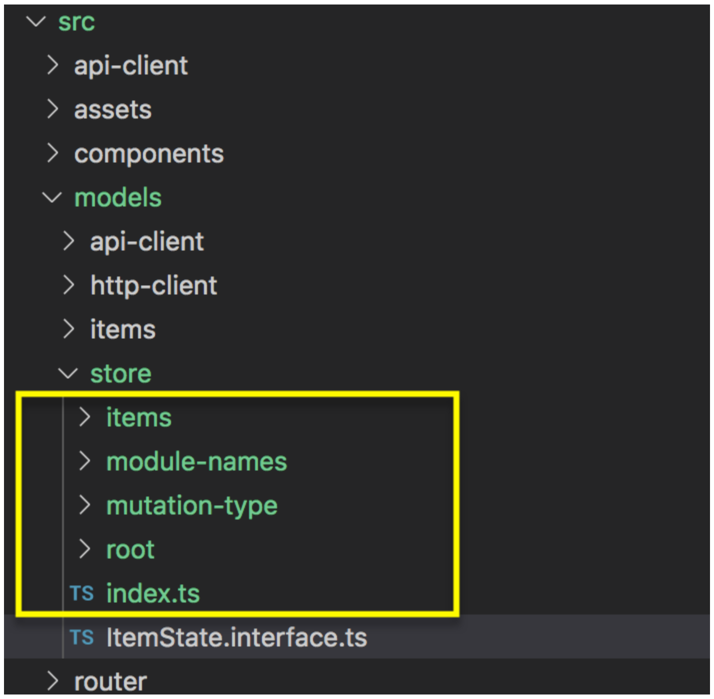
Let’s start by taking care of our **models** and **interfaces** first. We’ll need to create an interface for the main store (Root) and add an additional module for the Items domain. Later, in the next chapters, we’ll be adding even more domain-specific modules as we grow our project larger.

Let’s create sub-directories within our **src/models/store** to keep things more organized.

Add a file called **index.ts** and the following sub-directories:

- module-names
- mutation-type
- root
- items

Your **src/models/store** sub-directory structure should look like this:



```
src
  api-client
  assets
  components
  models
    api-client
    http-client
    items
    store
      items
      module-names
      mutation-type
      root
      index.ts
    ItemState.interface.ts
  router
```

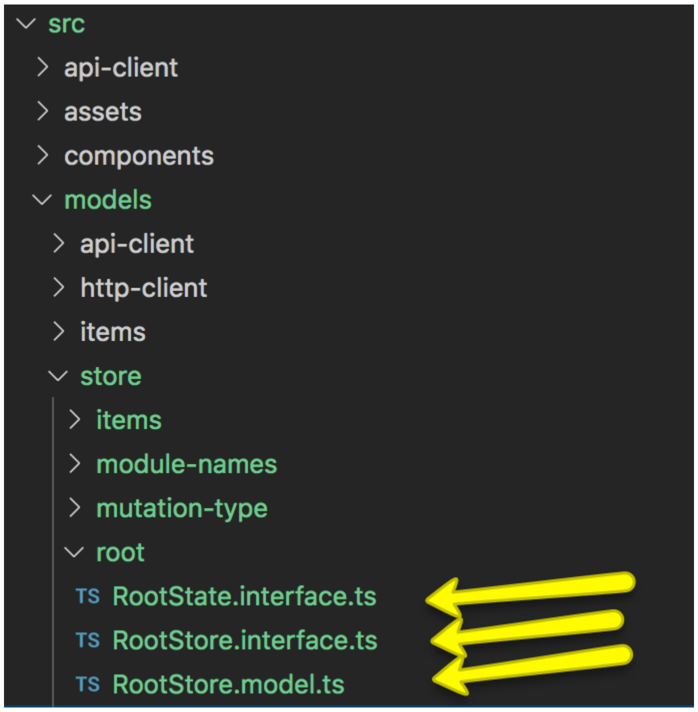
Now let’s create our Root store interfaces and model.

Root Store

Inside the **src/models/store/root** directory, add these 3 files:

- RootState.interface.ts
- RootStore.interface.ts
- RootStore.model.ts

Your **src/models/store/root** file structure should look like this:



```
src
  api-client
  assets
  components
  models
    api-client
    http-client
    items
    store
      items
      module-names
      mutation-type
      root
        RootState.interface.ts
        RootStore.interface.ts
        RootStore.model.ts
```

Following, is a description and the code for each of the **root** store files.

RootState.interface.ts

The **RootStateInterface** represent the **root state** of our application. We do not have a plan to use any specific root state for now, but for the sake of providing an example complete enough, let’s say that our root state will have one property called **loading** that we can set to true when the whole application is initially loading (if we want to, but we would not do it as part of this book).

```
/**
 * @name RootStateInterface
 * @description
 * Interface for the Root state
 */
export interface RootStateInterface {
  loading: boolean
}
```

RootStore.interface.ts

The **RootStoreInterface** just wraps all the other store modules in one place. This is similar to what we did with the **ApiClientInterface**. For now, we will only have one module represented by the **ItemsStateInterface**.

```
import {
  ItemsStateInterface
} from '../items/ItemsState.interface'

/**
 * @name RootStoreInterface
 * @description
 * Wraps together each store module interface in one place
 */
export interface RootStoreInterface {
  itemsState: ItemsStateInterface
// additional domain-specific module interfaces we'll be added here in the
next book\
 chapters
}
```

RootStore.model.ts

The **RootStoreModel** extends the **Vuex.Store** and implements our **RootStoreInterface**. This model represent the root store that we will be consuming in our application.

```
import Vuex from 'vuex'
import { RootStoreInterface } from './RootStore.interface'

/**
 * @name RootStoreModel
 * @description
 * Extends Vuex Store declaration with our RootStoreInterface
 */
export class RootStoreModel<S> extends Vuex.Store<RootStoreInterface> {
}
```

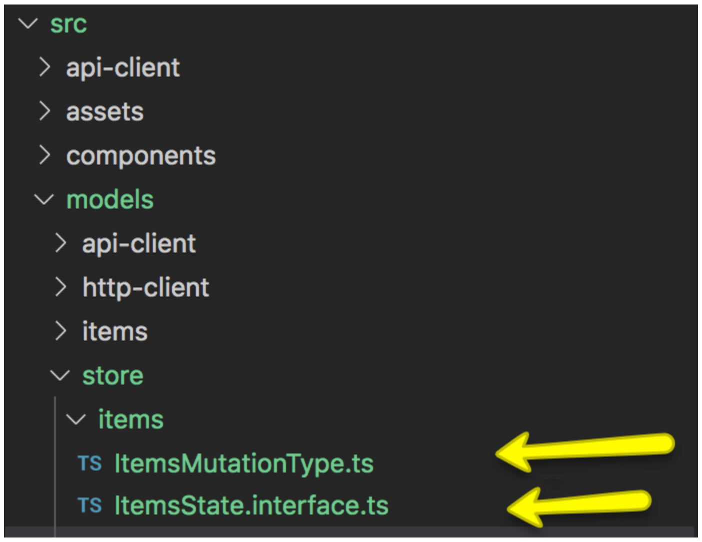
Items Store

NOTE: most of the code here is similar to the one in the first edition of this book (Vue 2), but here we reduced the number of files and code we create by leveraging TypeSCript namespaces instead of using additional interfaces for constant names wrapper like ItemsMutationType, MutationType, and StoreModuleNames.

Inside the **src/models/store/items** directory, add these 2 files:

- ItemsMutationType.ts
- ItemsState.interface.ts

Your **src/models/store/items** file structure should look like this:



```
src
  api-client
  assets
  components
  models
    api-client
    http-client
    items
    store
      items
        TS ItemsMutationType.ts
        TS ItemsState.interface.ts
```

Following is a description and the code for each of the **items** store files.

ItemsMutationType.ts

The **ItemsMutationType** is a namespace that export the **Items** mutation type names constants. We will use these later to replace the hard-coded strings we initially used in Chapter 6. As you add more mutation type, you will be adding to this additional constants as well:

```
// group our constants in a namespace
export namespace ItemsMutationType {
  export const loadItems: string = 'loadItems'
  export const loadingItems: string = 'loadingItems'
  export const loadedItems: string = 'loadedItems'
  export const selectItem: string = 'selectItem'
  // as you add new mutations to the Items store module, keep adding their
names her\
e as above
}
```

(NOTE: MutationType constants are used to invoke Vuex actions and mutations. You could call it ActionType, or however you like, but I had to pick a name and MutationType made more sense to me as, ultimately, a mutation is committed to the state, while an action is only an intermediary step int he Vuex flow)

ItemsState.interface.ts

We already created file in Chapter 6 when we first introduced Vuex. You can just move it from **src/models/store/ItemsState.interface.ts** to **src/models/store/items/ItemsState.interface.ts**. For completeness, here is the code for it again:

```
import { ItemInterface } from '@/models/items/Item.interface'

/**
 * @name ItemsStateInterface
 * @description
 * Interface for the Items state
 */
export interface ItemsStateInterface {
  loading: boolean
  items: ItemInterface[]
}
```

Main MutationType Namespace

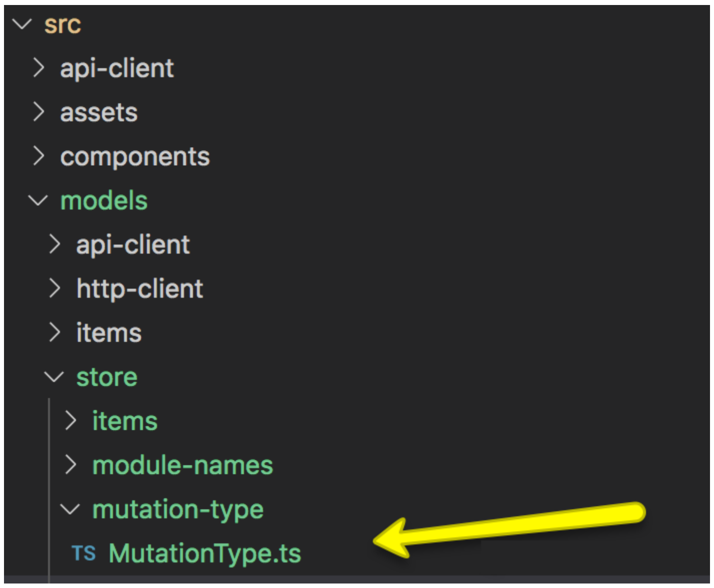
We now need to add the main **MutationType** namespace. This just wraps all the mutation types for each domain (i.e. **ItemsMutationType**) in one place.

Again, for now we have only the Items domain. But this enables us access our mutation type constants in a clear and organized way. I.e.: **MutationType.items.selectItem**.

Inside the **src/models/store/mutation-type** directory, add 1 file:

- MutationType.ts

Your **src/models/store/mutation-type** file structure should look like this:


```
src
  api-client
  assets
  components
  models
    api-client
    http-client
    items
    store
      items
      module-names
      mutation-type
        TS MutationType.ts
```

Following is a description and the code for the **mutation-type/MutationType.ts** file.

MutationType.ts

The **MutationType** is a namespace the groups and exports our domain specific mutation types like **ItemsMutationType** etc in an organized way:

```
// group our constants in a namespace
import { ItemsMutationType } from '../items/ItemsMutationType'

export namespace MutationType {
  export const items = ItemsMutationType
  // as you add new state modules, add additional exports here following the
same co\
nvention
}
```

The way we'll consume this later is to replace hard-coded strings like “loadItems" for example:

```
// source: src/views/Home.vue
...
onMounted(() => {
  store.dispatch('loadItems')
})
...
```

With our constant value **MutationType.items.loadItems**:

```
// source: src/views/Home.vue
...
onMounted(() => {
  store.dispatch(MutationType.items.loadItems)
})
...
```

As you can see this is much more readable than using constants like **MUTATION_LOAD_ITEM** or similar as you see often in trivial example on blog posts on the web.

Module Names Namespace

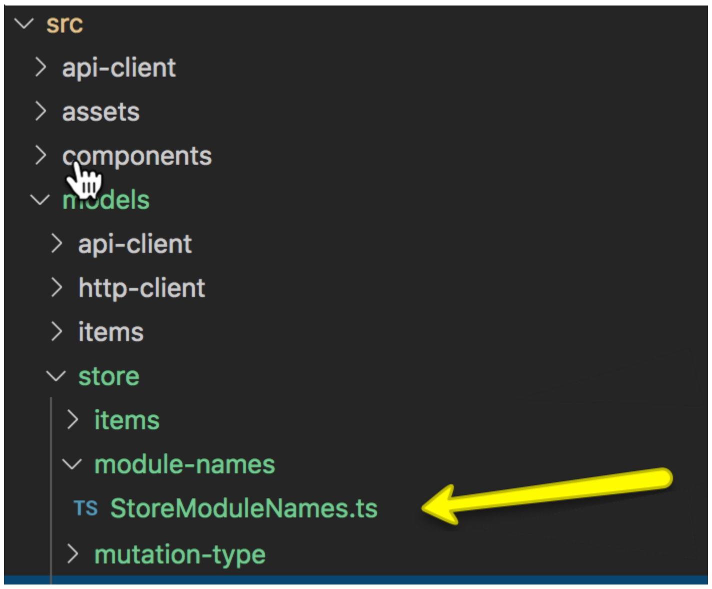
In a similar way, we'll be using constants for the **Vuex** module names. For this too we'll need a namespace that groups and exports the actual values in an organized way.

Again, the goal is to avoid hard-coded strings and as you will see in a little bit, with a modular Vuex store we'll be *dispatching actions/mutations with a name-spaced syntax.*

Inside the **src/models/store/module-names** directory, add 1 file:

- StoreModuleNames.ts

Your **src/models/store/module-names** file structure should look like this:



Following is a description and the code for the **module-names/StoreModuleNames.ts** file.

StoreModuleNames.ts

The **StoreModuleNames** is the namespace that groups and exports all our **Vuex** module names constants. As you add more **Vuex** store modules, you'll be adding here their names as well:

```
// group our constants in a namespace:
export namespace StoreModuleNames {
  export const itemsState: string = 'itemsState'
  // as you add new state modules, add additional properties here following
the same\
 convention
}
```

File src/models/store/index.ts

Finally the **src/models/store/index.ts** file will just export all our state models and interfaces:

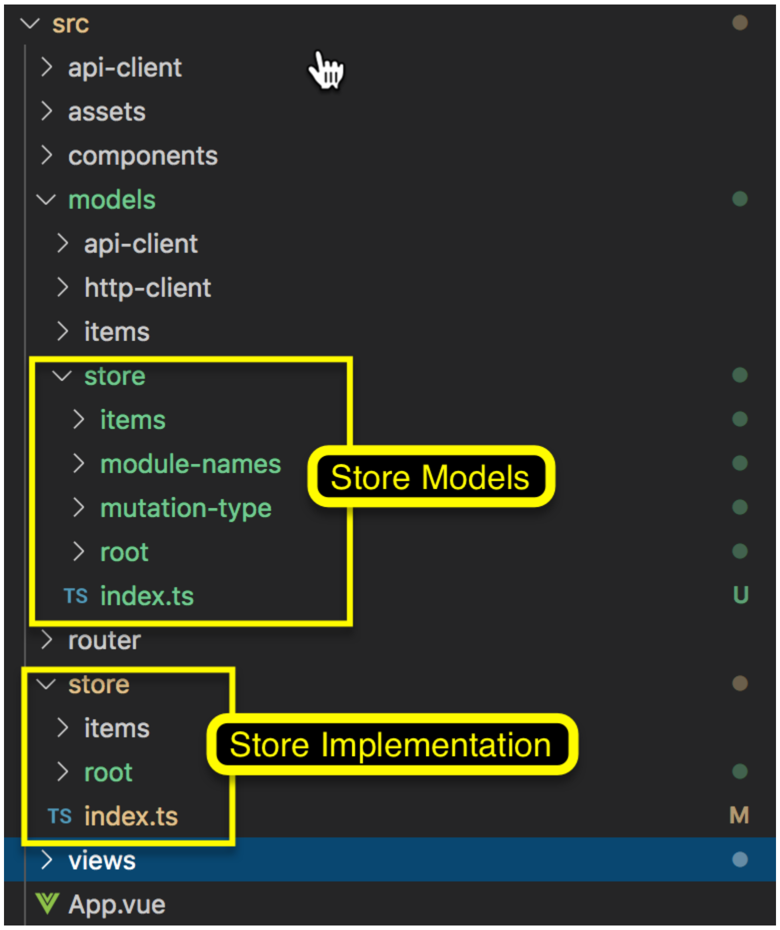
```
// export the MutationType namespace
export * from './mutation-type/MutationType'
// export the StoreModuleNames namespace
export * from './module-names/StoreModuleNames'
// export RooState and RootStore interfaces
export * from './root/RootState.interface'
export * from './root/RootStore.interface'
// export the RootStore model
export * from './root/RootStore.model'
// GEN-EXPORTS
// as you add more state modules, add additional exports for those here as
well
export * from './items/ItemsState.interface'
```

Let’s now move to the store implementation code under **src/store** directory.

Store Instance

Similarly to what we did in **src/models/store**, we need to organize our folder/file structure for the actual store implementation files under **src/store**.

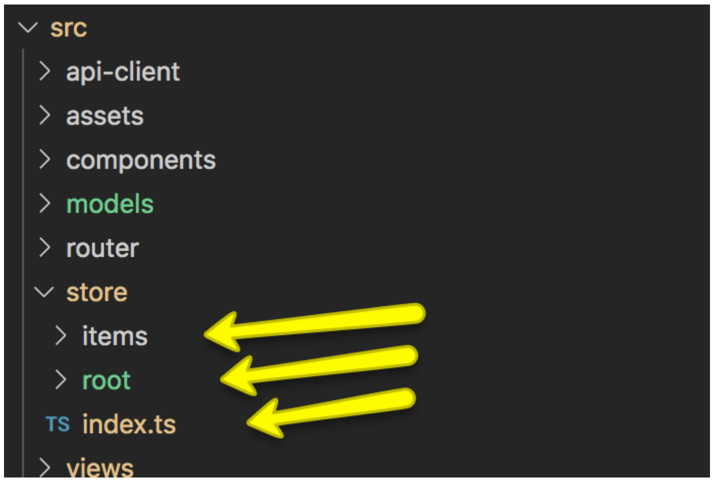
This might be confusing, so here is a screenshot to illustrate the locations for the models/interfaces and the one for the actual implementation:



Start by adding a file called **index.ts** and the following sub-directories:

- root
- items

Your **src/store** sub-directory structure should look like this:



Items Store Instance

The way a **Vuex** module is structured is like how the root store is structured, except a few additional things. We'll be using the **Module** interface from **Vuex** and compose a module like this:

```
Module<ItemsStateInterface, RootStateInterface> = {

namespaced,
state,

getters,
actions,

mutations
}
```

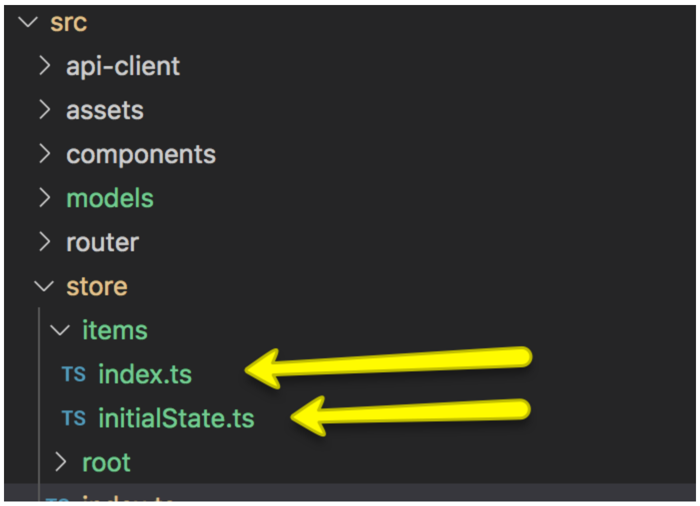
We'll be using our **TypeScript** interfaces here so we define our module with strong types by specifying both the module state interface (**ItemsStateInterface** in this case) and the interface for the root state (**RootStateInterface**):

Module<ItemsStateInterface, RootStateInterface>

Under the **src/store/items** folder, create the following 2 files:

- initialState.ts
- index.ts

Your **src/store/items** file structure should look like this:



Following is a description of each file and their code.

initialState.ts

Here we create and export an instance of our **ItemsStateInterface** with the initial state values:

```
import { ItemsStateInterface } from '@/models/store'

/**
 * @name initialItemsState
 * @description
 * The Items state instance with the initial default values
 */
export const initialItemsState: ItemsStateInterface = {
  loading: false,
  items: []
}
```

index.ts

Here we create our **Items** module instance by first importing **Module**, **MutationTree**, **ActionTree** and **GetterTree** from **Vuex**:

```
import { Module, MutationTree, ActionTree, GetterTree } from 'vuex'
...
```

Then import our **MutationType**, **RootStateInterface**, **ItemsStateInterface** and our **initialItemsState**:

```
...
import {
  MutationType,
  RootStateInterface,
  ItemsStateInterface
} from '@/models/store'

import {
  initialItemsState
} from './initialState'
...
```

Import also our **ItemInterface** and the **apiClient**:

```
...
import { ItemInterface } from '@/models/items/Item.interface'
import apiClient from '@/api-client'
...
...
```

Note: Some of the next code could also be extracted by the older file src/store/index.ts (as you implemented in Chapter 6)

Now let’s create our **mutations** as an instance of **MutationTree<ItemsStateInterface>** *(note that the code for the mutations methods like loadingItems etc is the same as in your older src/store/index.ts file)*:

```
...
export const mutations: MutationTree<ItemsStateInterface> = {
  loadingItems(state: ItemsStateInterface) {
    state.loading = true
  },
  loadedItems(state: ItemsStateInterface, items: ItemInterface[]) {
    state.items = items
    state.loading = false
  },
  selectItem(state: ItemsStateInterface, params: {
      id: number
      selected: boolean
    }) {
    const { id, selected } = params
    const item = state.items.find(o => o.id === id)
    if (item) {
      item.selected = selected
    }
  }
}
...
```

Next, let’s create our **actions** as an instance of **ActionTree<ItemsStateInterface, RootStateInterface>** *(note that the code for the actions methods like loadItems etc is very similar as in your older src/store/index.ts file. But note that we replaced the hard-coded strings like ‘loadedItems’ etc for the commits calls, with our MutationType constants)*:

```
...
export const actions: ActionTree<ItemsStateInterface, RootStateInterface> = {
  loadItems({ commit }) {
    commit(MutationType.items.loadingItems)

    // let's pretend we called some API end-point
    // and it takes 1 second to return the data
    // by using javascript setTimeout with 1000 for the milliseconds option
    setTimeout(() => {
      apiClient.items.fetchItems().then((data: ItemInterface[]) => {
        commit(MutationType.items.loadedItems, data)
      })
    }, 1000)
  },
  selectItem(
    { commit },
    params: {
      id: number
      selected: boolean
    }
  ) {
    commit(MutationType.items.selectItem, params)
  }
}
...
```

Now, let’s create our **getters** as an instance of **GetterTree<ItemsStateInterface, RootStateInterface>**

(note: this for now is blank as we have not coded any getters yet, however we should include it for completeness):

```
...
export const getters: GetterTree<ItemsStateInterface, RootStateInterface> = {}
...
```

Finally compose our **Module** by putting together our **mutations**, **actions**, **getters**. We will use our **initialItemsState** for the **state** and also tell **Vuex** that this module has to be name-spaced by setting the **namespaced** flag to true:

```
...
// create our Items store instance
const namespaced: boolean = true
const state: ItemsStateInterface = initialItemsState

export const itemsState: Module<ItemsStateInterface, RootStateInterface> = {
  namespaced,
  state,
  getters,
  actions,
  mutations
}
```

We are using **namespaced** true because we want our module to be more **self-contained** and **reusable** as per the official https://vuex.vuejs.org/ guidance[2]:

> Namespacing
> By default, actions, mutations and getters inside modules are still registered under the global namespace - this allows multiple modules to react to the same mutation/action type.
> If you want your modules to be more self-contained or reusable, you can mark it as namespaced with namespaced: true. When the module is registered, all of its getters, actions and mutations will be automatically namespaced based on the path the module is registered at.

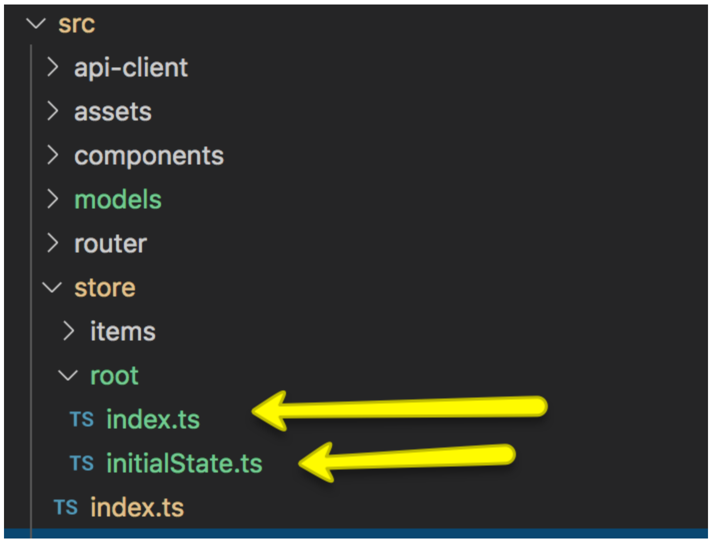
This has implications on how we'll dispatch actions to our **Vuex** module in our components because it will require us to use the "module-name/mutation-name" syntax.

Root Store Instance

Under the **src/store/root** folder, create the following 2 files:

- initialState.ts
- index.ts

Your **src/store/root** file structure should look like this:



Following is a description of each file and their code.

initialState.ts

Here we create and export an instance of our **RootStateInterface** with the initial state values:

```
import { RootStateInterface } from '@/models/store'

/**
 * @name initialRootState
 * @description
 * The Root state instance with the initial default values
 */
export const initialRootState: RootStateInterface = {
  loading: false
}
```

index.ts

Here we create our **Vuex** root instance which will implement our **RootStoreModel<RootStateInterface>** which put all our modules together:

```
import { createStore, StoreOptions } from 'vuex'

import {
  RootStateInterface,
  RootStoreInterface,
  RootStoreModel
} from '@/models/store'

import { initialRootState } from './initialState'

// import each Vuex module
import { itemsState } from '@/store/items'

// Vuex store options to build our modularized namespaced store
const storeOptions: StoreOptions<RootStateInterface> = {
  state: initialRootState,

  modules: {
    itemsState
    // as you create additional modules, you'lla dd them here similar to the
itemsSt\
ate
  }
}

// Vuex Root store instance
export const store: RootStoreModel<RootStateInterface> =
<any>createStore(storeOptio\
ns)
```

Update file src/store/index.ts

Finally lets open again the older Vuex Store instance code under src/store/index.ts and replace all the code in there with one line that export our **root** instance:

```
export * from './root'
```

Let’s now change our **Vue** components to leverage our store changes.

Update src/main.ts

Note: we are no longer exporting **store** as default, so we need to import with:

```
import { createApp } from 'vue'
import App from './App.vue'
import router from './router'
import store from './store' // <-- remove this line
import { store } from './store' // <-- add this line

createApp(App)
  .use(store)
  .use(router)
  .mount('#app')
```

Update Home.vue

Let's add an import for our **MutationType**, **StoreModuleNames**, and **store** instance (*Note: we are no longer exporting* ***store*** *as default, so we need to import with*):

```
...
<script lang="ts">
  import { defineComponent, computed, onMounted } from 'vue'
  import store from '@/store' // <-- remove this line
  // add these two lines
  import { store } from '@/store'
  import { MutationType, StoreModuleNames } from '@/models/store'
...
```

In the **setup()** method, within the computed properties, our access to **store.state.items** will no longer work. We have to change it to **store.state.itemsState.items** (similar thing for state.loading):

```
...
    setup() {
      const items = computed(() => {
        return store.state.items // <-- remove this line
        // add this line:
        return store.state.itemsState.items
      })
      const loading = computed(() => {
        return store.state.loading // <-- remove this line
        // add this line:
        return store.state.itemsState.loading
      })
...
```

We need to change how we dispatch the loadItems action within the onMounted to use the **Vuex** name-space syntax ('store-modulename/mutation-name'). Here I am using JavaScript string interpolation to build the fully qualified mutation name:

```
...
      onMounted(() => {
        store.dispatch(MutationType.items.loadItems) // <-- remove this line
        // add this line:

store.dispatch(`${StoreModuleNames.itemsState}/${MutationType.items.loadItem\
s}`)
      })
...
```

Similarly, we need to also change the code that dispatch the **selectItem** action:

```
...
      const onSelectItem = (item: ItemInterface) => {
        store.dispatch(MutationType.items.selectItem, {  // <-- remove this
line

store.dispatch(`${StoreModuleNames.itemsState}/${MutationType.items.selectIt\
em}`, { // <-- add this line
          id: item.id,
          selected: !item.selected
        })
      }
...
```

Note how we have to use the namespace prefix **itemsState** with the convention [**store-modulename]/[mutation-name]** when dispatching our action. Also, for the name of the **action** we replaced the hard-coded string with **MutationType.items.selectingItem**.

The ugly module/mutation string interpolation syntax

For most of us, the ugly syntax with the string interpolation is a pain in the butt. We can do better than that.

Let’s get back into the file **src/store/root/index.ts** and update the import sections (note we are also adding /module to the path for **itemsState** import):

```
import { createStore, StoreOptions } from 'vuex'

import {
  RootStateInterface,
  RootStoreInterface, // <-- remove this line
  RootStoreModel
} from '@/models/store'

…
import { itemsState } from '@/store/items/module' // <-- append "/module" here
…
```

At the end of the file, rename the exported const variable **store** to **rootStore,** and after this line, add an helper function called **dispatchModuleAction<T>** that takes 3 parameters:

- **moduleName**: string
- **actionName**: string
- **params?**: T

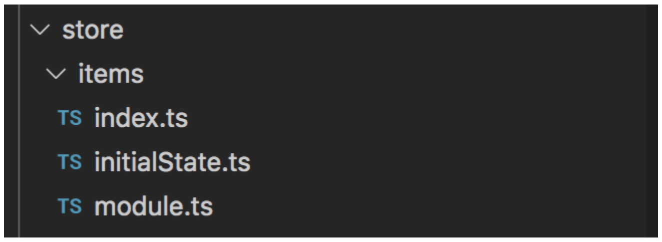
This enable us to wrap the ugly string interpolation syntax in one place. Here is the code:

```
…
// Vuex Root store instance
// delete this line:
export const store RootStoreModel<RootStateInterface> =
<any>createStore(storeOption\
s)
// add this line:
export const rootStore: RootStoreModel<RootStateInterface> =
<any>createStore(storeO\
ptions)
...
// begin: add code
// Private helper to dispatch an action to a Vuex module from one place so we
keep t\
he string interpolation for `${moduleName}/${actionName}` in one place only
and be a\
ble to dispatch action with less code in a strongly-type way
export function dispatchModuleAction<T>(moduleName: string, actionName:
string, para\
ms?: T): void {
  // rename store.dispatch here to rootStore.dispatch
  rootStore.dispatch(`${moduleName}/${actionName}`, params)
}
// end: add code
```

Files: store/items/module.ts and store/items/index.ts

Rename **store/items/index.ts** to **store/items/module.ts** Then add a new blank **index.ts** file.

Your **src/store/root** file structure now will look like this:



Within the new blank **store/items/index.ts** file, add a custom object called **itemsStore** that represent what we'll actually need to consume later in our views/components for the **Items** store. We also export the **useItemsStore** function (as per composition API style) which will return our **itemsStore**:

```
import { rootStore, dispatchModuleAction } from '../root'
import { StoreModuleNames, ItemsStateInterface } from '@/models/store'

/**
 * @name itemsStore
 * @description
 * The items store wrapper that returns the itemsState and exposes a generic
action<\
T> method
 */
const itemsStore = {
  get state(): ItemsStateInterface {
    return rootStore.state.itemsState
  },
  action<T>(actionName: string, params?: T): void {
    dispatchModuleAction(StoreModuleNames.itemsState, actionName, params)
  }
}
// export our wrapper using the composition API convention (i.e. useXYZ)
export const useItemsStore = () => {
  return itemsStore
}
```

Update src/main.ts

Update the the code to use the new name **rootStore**:

```
…
import { store } from './store' // <-- delete this line
import { rootStore } from './store' // <-- add this line
…
createApp(App)
  .use(store) // <-- delete this line
  .use(rootStore) // <-- add this line
  .use(router)
…
```

Home.vue changes

Now back in the **Home.vue** let's import a reference to the **useItemsStore**:

```
…
<script lang="ts">
  import { defineComponent, computed, onMounted } from 'vue'
  import { store } from '@/store' // <-- delete this line
  // add this line:
  import { useItemsStore } from '@/store/items'
…
```

Then create a reference for our **itemsStore** in the setup method and start consuming its state:

```
...
    setup() {
      const itemsStore = useItemsStore() // <-- add this line
      const items = computed(() => {
        return store.state.itemsState.items // <-- remove this line
        return itemsStore.state.items // <-- add this line
      })

      const loading = computed(() => {
        return store.state.itemsState.loading // <-- remove this line
        return itemsStore.state.loading // <-- add this line
      })

...
```

Also, still in **Home.vue** code, we can finally replace the code for **store.dispatch** that uses the ugly string-interpolation syntax by using our new **itemsStore.action** method instead:

```
...
      onMounted(() => {
        // remove this line:

store.dispatch(`${StoreModuleNames.itemsState}/${MutationType.items.loadItem\
s}`)
        // add this line:
        itemsStore.action(MutationType.items.loadItems)
      })
...
      const onSelectItem = (item: ItemInterface) => {
        // begin: remove code
~~        ~~
~~store.dispatch(`${StoreModuleNames.itemsState}/${MutationType.items.selectIt\~~
~~em}`, {~~
~~          id: item.id,~~
~~          selected: !item.selected~~
~~        })~~
        // end: remove code
        // begin: add code
        itemsStore.action(MutationType.items.selectItem, {
          id: item.id,
          selected: !item.selected
        })
        // end: add code
      }
...
```

Summary

All this might seem overkill especially to less experienced developers. It might initially create a bit of confusion when accessing your store. But in a large code base, it will help keeping things encapsulated by app domain and ultimately the benefits of better code readability, maintainability, unit testing etc will out-weight the bit of extra work that this requires.

Furthermore, it is possible to eventually auto-generate code for the API client and store modules interfaces/models and constants namespaces with tools like **plop** or **Swagger** (maybe I will either add a Chapter on how to do that, or write a blog post about it).

If you encounter any trouble, please refer to the companion code on GitHub[3], or reach me on twitter with questions or comments.

Chapter 9 Recap

What We Learned

- How to create a modularized **Vuex** store
- How to use a **MutationType** object to keep all the mutation types related to each store module in one place
- Make code more readable by avoid using hard-coded strings and enforcing constant values checks through interfaces
- How to wrap what we need (state and action<T>) into an object that represent the store module (itemsStore) and avoid using strings for dispatching actions

Observations

- We did not write unit tests against the **RootStore actions** and **mutations**
- Based on these observations, there are a few improvements that you can do.

Improvements

- Add unit tests against the **RootStore actions** and **mutations**

Chapter 10 - Localization and Internationalization - Language Localization - Part 1

"Localization refers to the adaptation of a product, application or document content to meet the language, cultural and other requirements of a specific target market (a locale)…"

"…Internationalization is the design and development of a product, application or document content that enables easy localization for target audiences that vary in culture, region, or language" [1]

NOTE: This chapter applies to you only if the application you are working on will be used or marketed to multiple countries and it is desired to present labels in the local language, as well as date/number formats in the local culture.

Most modern applications that target multiple countries or cultures are architected in a way that is easy to present the UI in different languages and also present values like numbers or dates formatted as expected by the culture specific to that country (hence, localized).

In this book we'll first leverage a plugin that allows us to present labels in different languages (**vue-i18n**) and later we'll add also a custom plugin based on the **Intl API** (supported by most modern browsers) to provide for numbers/date formatting functionality based on different locales (cultures).

*NOTE: **vue-18n-next** offers also formatting, but is still under work and we would rather show you here how to use the browser built-in **Intl API** instead. Later, as **vue-i18n-next** formatting capabilities are more solid, you might just use those if you prefer.*

Plugin: vue-i18n

There are many JavaScript libraries out there that simplify localization of a frontend app, but the most widely used is the **i18n** library. User **Kazuya Kawaguchi** (**kazupon)**[2] maintains a very nice **Vue** plugin called **vue-i18n,** which is published on NPM here [https://www.npmjs.com/package/vue-i18n

In this book we'll be creating an additional plugin that wrap around the **vue-i18n** which will allow us to avoid code cluttering and greatly simplify how we localize our components in our **Vue** application.

You are free to call the additional plugin as you prefer if you want to avoid confusion with other existing plugins out there. In this book we'll call it **vue-i18n-next-plugin** to make it easier to remember what really is.

Let's start by first adding the **vue-i18n** NPM package to our application. We need to use the command **npm install -save vue-i18n@next** since we are using **Vue 3** (*NOTE: The **vue-i18n next** is currently at version **9.0.0-beta.8** as of this writing, but you might get a newer version. If you want to avoid errors, you can get the exact same version with "**npm install -save vue-i18n@9.0.0-beta.8**")*[3].

We need to use the **-save** option as we want this to be saved as part of the app "dependencies" in the **package.json**:

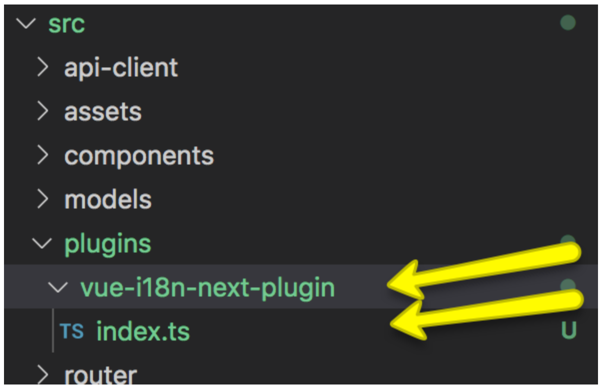
```
npm install --save vue-i18n@next
```

Let’s also install the **vue-i18n** types (these are only for TypeScript and not required to run the application, so we can use **—save-dev** to add only to the **devDependencies**):

```
npm install --save-dev @types/vue-i18n
```

NOTE: You might not need @types/vue-i18n as vue-i18n-next includes them but it seem to still be behind a few things so that is why I added as an extra step here.

Then we need to add a new folder called **plugins** within our project src directory. Inside the plugins folder we’ll add another folder called **vue-i18n-next-plugin** and inside this add a TypeScript file called **index.ts**. The directory structure should look like this:



For the **index.ts** code, let’s start importing **createI18n** and **LocaleMessages** from **vue-i18n**. Let’s also declare an interface called **LocalesDataInterface** that defines a property called **messages** of the **vue-18n** type **LocaleMessages**. This will simplify things later when we’ll be loading the messages from static **JSON** files.

```
import { createI18n, LocaleMessages, VueMessageType } from 'vue-i18n'

interface LocalesDataInterface {
  messages: LocaleMessages<VueMessageType>
}

…
```

For now let’s instantiate a const called data with some initial data for a welcome message that we’ll display in a different language for 4 different locales:

```
…

const data: LocalesDataInterface = {
  messages: {
    'en-US': {
      welcome: 'Welcome: this message is localized in English'
    },
    'it-IT': {
      welcome: 'Benvenuti: this message is localized in Italian'
    },
    'fr-FR': {
      welcome: 'Bienvenue: this message is localized in French'
    },
    'es-ES': {
      welcome: 'Bienvenido: this message is localized in Spanish'
    }
  }
}
```

Note how messages is a key-value pair lookup (or strategy pattern) that allows the **vue-i18n** plugin to know about localized text strings for specific "locales" (or cultures).

NOTE: You can learn more about locales and their standard and definitions here [https://en.wikipedia.org/wiki/Language_localisation and here [https://github.com/ladjs/i18n-locales or here [https://github.com/mashpie/i18n-node)

In our code above, we added four entries, each one related to a specific culture (or language):

- English (en-us)
- Italian (it-IT)
- French (fr-FR)
- Spanish (es-ES)

And each of them has only one key called **welcome** which holds the value for each specific locale.

Finally we export a **const** named just **i18n** for brevity which will hold a reference to a new instance of **VueI18n**. Note that the locale property is set to 'it-IT' and this will be the currently selected locale:

```
…

export const i18n = createI18n({
  locale: 'it-IT',
  fallbackLocale: 'en-US',
  messages: data.messages
})
```

*Note that we are only going to need one instance of our plugin and we could also use Object.freeze if we want to be extra careful, but since it is a **const** we would be warned by TypeScript if we accidentally try to somehow overwrite this in other parts of our code.*

Let’s now switch to the main TypeScript file for our **Vue** app which is located at **src/main.ts** and import a reference to our **vue-i18n-next-plugin** and let the **Vue** app instance know about it:

```
import { createApp } from 'vue'
import App from './App.vue'
import router from './router'
import { store } from './store'
// add following line:
import { i18n } from ‘./plugins/vue-i18n-next-plugin'

createApp(App)
  .use(store)
  .use(router)
  .use(i18n) // <-- add this line
  .mount('#app')
```

Now if you serve the app with **npm run serve** it should build and run without errors. But, of course, the plugin is not doing anything yet.

Now, we’ll soon be adding more code in the **vue-i18n-next-plugin** directory to better structure and define the different locales as this book is about writing code that can grow large and is as clean as possible, but for the time being let’s go change one of our views to demonstrate how it works.

App.vue

Let’s open our main **App.vue** file and within the **<script>** section let’s add an import to **useI18n** from **vue-18n**:

```
<script lang="ts">
  import { defineComponent } from 'vue'
  // add this line:
  import { useI18n } from 'vue-i18n'
  …
```

Within our component definition, add a **setup()** method and within this get a reference to **i18n** with **useI18n()** and return it as part of the component properties:

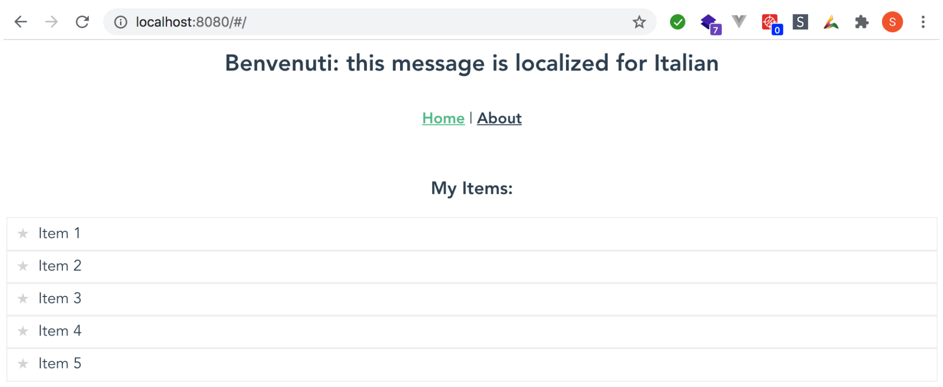
```
…

  export default defineComponent({
    name: 'App',
    // begin: add code block
    setup() {
      const i18n = useI18n()

      return {
        i18n
      }
    }
    // end: add code block
  })
</script>
```

Within the **<template>** section, just before the navigation element there should be either a **<h1>** or **<h2>** element. Replace the hard-coded text message with the one from the **i18n** plugin using the handle-bars binding and the **i18n.t()** function **}**:

```
<template>
  <div id="app">
    <h2>{{ i18n.t('welcome') }}</h2>
    <div id="nav" class="nav">
```

```
      <router-link to="/">Home</router-link> |
      <router-link to="/about">About</router-link>
    </div>
    <router-view />
  </div>
</template>
```

Save the file again, and make sure this time the **message** from **it-IT** is displayed *(NOTE: it default to it-IT because we specified this as the default locale within **createI18n()** in our **vue-i18n-next-plugin**)*:


localhost:8080/#/

Benvenuti: this message is localized for Italian

Home | About

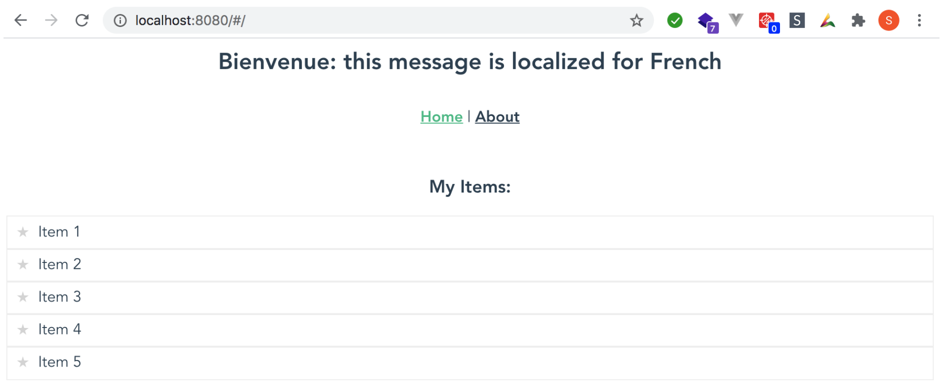
My Items:

- Item 1
- Item 2
- Item 3
- Item 4
- Item 5

In the next chapter we'll build a simple component that will allow us to easily switch locale from the UI so we can quickly test the different locale messages. For the time being, let's edit the **vue-i18n-next-plugin/index.ts** code once more so that it uses the **French** locale:

```
export const i18n = new VueI18n({
    locale: 'fr-FR', // change this temporarily to fr-FR and save
    fallbackLocale: 'en-US',
    messages:
    ...
```

Again save the file, and you should see the French version of the message in the browser:


localhost:8080/#/

Bienvenue: this message is localized for French

Home | About

My Items:

★ Item 1

★ Item 2

★ Item 3

★ Item 4

★ Item 5

Chapter 10 Recap

What We Learned

- How to add the **vue-i18n** plugin to our application using an additional wrapper plugin
- How to use multiple **locale** settings for text translation in order to localize our UI labels
- How to switch to test a different locale by manually (for now) updating the default locale parameter passed as one of the options to the **vue-i18n createI18n()**

Observations

- We hard-coded the localized strings for each locale
- We do not have a dynamic way to switch to a different locale at run-time

Based on these observations, there are a few improvements that we can do in the next two chapters:

Improvements

- Better structure our **vue-i18n-next-plugin** code so that it can load each locale messages from a static **JSON** file automatically
- Build a small component that allow to switch locale at run-time to visually test the different locales

```
<code></code><code class="kr">export</code> <code
class="kr">interface</code> <code
class="nx">LocaleInfoInterface</code> <code class="p">
{</code>

   <code class="nx">name</code>: <code
class="kt">string</code> <code class="c1">// the friendly
name of the locale, i.e. USA</code>

   <code class="nx">locale</code>: <code
class="kt">string</code> <code class="c1">// the locale
code, i.e. en-US</code>

   <code class="nx">flag</code>: <code
class="kt">string</code> <code class="c1">// the 2 char
code used to build the icon name</code>

   <code class="nx">selected</code>: <code
class="kt">boolean</code> <code class="c1">// if this
locale is currently</code>

<code class="p">}</code>

<code></code><code class="c1">// group our constants
in a namespace</code>

<code class="kr">export</code> <code
class="nx">namespace</code> <code
class="nx">LocalesMutationType</code> <code
class="p">{</code>

   <code class="kr">export</code> <code
class="kr">const</code> <code
class="nx">selectLocale</code>: <code
class="kt">string</code> <code class="o">=</code>
<code class="s1">'selectLocale'</code>
```

```
   <code class="c1">// as you add more mutations to the
Locales store module, keep adding their names \</code>

<code class="nx">here</code> <code
class="kr">as</code> <code class="nx">above</code>

<code class="p">}</code>

<code></code><code class="kr">import</code> <code
class="p">{</code> <code
class="nx">LocaleInfoInterface</code> <code
class="p">}</code> <code class="nx">from</code>
<code
class="s1">'@/models/localization/LocaleInfo.interface'</co
de>



<code class="cm">/**</code>

<code class="cm"> * @name
LocalesStateInterface</code>

<code class="cm"> * @description</code>

<code class="cm"> * Interface for the Locales
state</code>

<code class="cm"> */</code>

<code class="kr">export</code> <code
class="kr">interface</code> <code
class="nx">LocalesStateInterface</code> <code
class="p">{</code>

   <code class="nx">availableLocales</code>: <code
class="kt">LocaleInfoInterface</code><code class="p">[]
```

</code>

<code class="p">}</code>

<code></code><code class="p">...</code>

<code class="c1">// as you add more state modules, add additional exports for those here as well</code>

<code class="kr">export</code> <code class="o">*</code> <code class="nx">from</code> <code class="s1">'./items/ItemsState.interface'</code>

<code class="kr">export</code> <code class="o">*</code> <code class="nx">from</code> <code class="s1">'./locales/LocalesState.interface'</code> <code class="c1">// <-- add this line</code>

<code></code><code class="c1">// group our constants in a namespace:</code>

<code class="kr">export</code> <code class="nx">namespace</code> <code class="nx">StoreModuleNames</code> <code class="p">{</code>

   <code class="kr">export</code> <code class="kr">const</code> <code class="nx">itemsState</code>: <code class="kt">string</code> <code class="o">=</code> <code class="s1">'itemsState'</code>

   <code class="kr">export</code> <code class="kr">const</code> <code class="nx">localesState</code>: <code class="kt">string</code> <code class="o">=</code>

<code class="s1">'localesState'</code> <code class="c1">// <-- add this line</code>

  <code class="c1">// as you add more state modules, add additional properties here following the sam\</code>

<code class="nx">e</code> <code class="nx">convention</code>

<code class="p">}</code>

<code></code><code class="c1">// group our constants in a namespace</code>

<code class="kr">import</code> <code class="p">{</code> <code class="nx">ItemsMutationType</code> <code class="p">}</code> <code class="nx">from</code> <code class="s1">'../items/ItemsMutationType'</code>

<code class="kr">import</code> <code class="p">{</code> <code class="nx">LocalesMutationType</code> <code class="p">}</code> <code class="nx">from</code> <code class="s1">'../locales/LocalesMutationType'</code>

<code class="kr">export</code> <code class="nx">namespace</code> <code class="nx">MutationType</code> <code class="p">{</code>

  <code class="kr">export</code> <code class="kr">const</code> <code class="nx">items</code> <code class="o">=</code> <code class="nx">ItemsMutationType</code>

<code class="kr">export</code> <code class="kr">const</code> <code class="nx">locales</code> <code class="o">=</code> <code class="nx">LocalesMutationType</code>

<code class="c1">// as you add more domain-specific mutation types, add them here following the sam\</code>

<code class="nx">e</code> <code class="nx">convention</code>

<code class="p">}</code>

<code></code><code class="kr">import</code> <code class="p">{</code> <code class="nx">ItemsStateInterface</code> <code class="p">}</code> <code class="nx">from</code> <code class="s1">'../items/ItemsState.interface'</code>

<code class="kr">import</code> <code class="p">{</code> <code class="nx">LocalesStateInterface</code> <code class="p">}</code> <code class="nx">from</code> <code class="s1">'../locales/LocalesState.interface'</code> <code class="c1">// <-- add\</code>

<code class="k">this</code> <code class="nx">line</code>

<code class="cm">/**</code>

<code class="cm"> * @name RootStoreInterface</code>

<code class="cm"> * @description</code>

<code class="cm"> * Wraps together each store module interface in one place</code>

<code class="cm"> */</code>

<code class="kr">export</code> <code class="kr">interface</code> <code class="nx">RootStoreInterface</code> <code class="p">{</code>

   <code class="nx">itemsState</code>: <code class="kt">ItemsStateInterface</code><code class="p">,</code>

   <code class="nx">localesState</code>: <code class="kt">LocalesStateInterface</code> <code class="c1">// <-- add this line</code>

   <code class="c1">// additional domain-specific module interfaces we'll be added here in the next bo\</code>

<code class="nx">ok</code> <code class="nx">chapters</code>

<code class="p">}</code>

<code></code><code class="kr">import</code> <code class="p">{</code> <code class="nx">LocalesStateInterface</code> <code class="p">}</code> <code class="nx">from</code> <code class="s1">'@/models/store'</code>

<code class="cm">/**</code>

<code class="cm"> * @name initialLocalesState</code>

```
<code class="cm"> * @description</code>

<code class="cm"> * The Locales state instance</code>

<code class="cm"> */</code>

<code class="kr">export</code> <code
class="kr">const</code> <code
class="nx">initialLocalesState</code>: <code
class="kt">LocalesStateInterface</code> <code
class="o">=</code> <code class="p">{</code>

   <code class="nx">availableLocales</code><code
class="o">:</code> <code class="p">[</code>

   <code class="p">{</code>

   <code class="nx">name</code><code class="o">:
</code> <code class="s1">'USA'</code><code
class="p">,</code>

   <code class="nx">locale</code><code class="o">:
</code> <code class="s1">'en-US'</code><code
class="p">,</code>

   <code class="nx">flag</code><code class="o">:
</code> <code class="s1">'us'</code><code class="p">,
</code>

   <code class="nx">selected</code>: <code
class="kt">false</code>

   <code class="p">},</code>

   <code class="p">{</code>

   <code class="nx">name</code><code class="o">:
</code> <code class="s1">'Italy'</code><code
```

class="p">,</code>

   <code class="nx">locale</code><code class="o">:
</code> <code class="s1">'it-IT'</code><code
class="p">,</code>

   <code class="nx">flag</code><code class="o">:
</code> <code class="s1">'it'</code><code class="p">,
</code>

   <code class="nx">selected</code>: <code
class="kt">true</code> <code class="c1">// this is
selected by default</code>

   <code class="p">},</code>

   <code class="p">{</code>

   <code class="nx">name</code><code class="o">:
</code> <code class="s1">'Spain'</code><code
class="p">,</code>

   <code class="nx">locale</code><code class="o">:
</code> <code class="s1">'es-ES'</code><code
class="p">,</code>

   <code class="nx">flag</code><code class="o">:
</code> <code class="s1">'es'</code><code class="p">,
</code>

   <code class="nx">selected</code>: <code
class="kt">false</code>

   <code class="p">},</code>

   <code class="p">{</code>

```
   <code class="nx">name</code><code class="o">:
</code> <code class="s1">'France'</code><code
class="p">,</code>

   <code class="nx">locale</code><code class="o">:
</code> <code class="s1">'fr-FR'</code><code
class="p">,</code>

   <code class="nx">flag</code><code class="o">:
</code> <code class="s1">'fr'</code><code class="p">,
</code>

   <code class="nx">selected</code>: <code
class="kt">false</code>

   <code class="p">}</code>

   <code class="p">]</code>

<code class="p">}</code>

<code></code><code class="kr">import</code> <code
class="p">{</code> <code class="nx">Module</code>
<code class="p">,</code> <code
class="nx">MutationTree</code><code class="p">,
</code> <code class="nx">ActionTree</code><code
class="p">,</code> <code class="nx">GetterTree</code>
<code class="p">}</code> <code
class="nx">from</code> <code
class="s1">'vuex'</code>



<code class="kr">import</code> <code class="p">
{</code>
```

```
   <code class="nx">MutationType</code><code
class="p">,</code>

   <code class="nx">RootStateInterface</code><code
class="p">,</code>

   <code class="nx">LocalesStateInterface</code>

<code class="p">}</code> <code
class="nx">from</code> <code
class="s1">'@/models/store'</code>



<code class="kr">import</code> <code class="p">
{</code>

   <code class="nx">initialLocalesState</code>

<code class="p">}</code> <code
class="nx">from</code> <code
class="s1">'./initialState'</code>



<code class="kr">import</code> <code class="p">
{</code> <code class="nx">i18n</code> <code
class="p">}</code> <code class="nx">from</code>
<code class="s1">'@/plugins/vue-i18n-next-plugin'</code>

<code class="p">...</code>

<code></code><code class="p">...</code>

<code class="c1">// Vuex Locales mutations</code>

<code class="kr">export</code> <code
class="kr">const</code> <code
```

class="nx">mutations</code>: <code class="kt">MutationTree</code><code class="o"><</code><code class="nx">LocalesStateInterface</code><code class="o">></code> <code class="o">=</code> <code class="p">{</code>

  <code class="nx">selectLocale</code><code class="p">(</code><code class="nx">state</code>: <code class="kt">LocalesStateInterface</code><code class="p">,</code> <code class="nx">localeId</code>: <code class="kt">string</code><code class="p">)</code> <code class="p">{</code>

  <code class="c1">// set only the model selected to true</code>

  <code class="nx">state</code><code class="p">.</code><code class="nx">availableLocales</code><code class="p">.</code><code class="nx">forEach</code><code class="p">(</code><code class="nx">localeInfo</code> <code class="o">=></code> <code class="p">{</code>

  <code class="nx">localeInfo</code><code class="p">.</code><code class="nx">selected</code> <code class="o">=</code> <code class="nx">localeInfo</code><code class="p">.</code><code class="nx">locale</code> <code class="o">===</code> <code class="nx">localeId</code>

  <code class="k">if</code> <code class="p">(</code><code class="nx">localeInfo</code><code class="p">.</code><code class="nx">selected</code><code class="p">)</code> <code class="p">{</code>

  <code class="c1">// switch i18n selected locale</code>

```
   <code class="nx">i18n</code><code class="p">.
</code><code class="nx">global</code><code
class="p">.</code><code class="nx">locale</code>
<code class="p">.</code><code
class="nx">value</code> <code class="o">=</code>
<code class="nx">localeInfo</code><code class="p">.
</code><code class="nx">locale</code>

   <code class="p">}</code>

   <code class="p">})</code>

   <code class="p">}</code>

<code class="p">}</code>



<code class="c1">// Vuex Locales actions</code>

<code class="kr">export</code> <code
class="kr">const</code> <code
class="nx">actions</code>: <code
class="kt">ActionTree</code><code class="o"><</code>
<code class="nx">LocalesStateInterface</code><code
class="p">,</code> <code
class="nx">RootStateInterface</code><code class="o">>
</code> <code class="o">=</code> <code class="p">
{</code>

   <code class="nx">selectLocale</code><code
class="p">({</code> <code class="nx">commit</code>
<code class="p">},</code> <code
class="nx">localeId</code>: <code
class="kt">string</code><code class="p">)</code>
<code class="p">{</code>
```

```
   <code class="nx">commit</code><code class="p">(</code><code class="nx">MutationType</code><code class="p">.</code><code class="nx">locales</code><code class="p">.</code><code class="nx">selectLocale</code><code class="p">,</code> <code class="nx">localeId</code><code class="p">)</code>

   <code class="p">}</code>

<code class="p">}</code>



<code class="kr">export</code> <code class="kr">const</code> <code class="nx">getters</code>: <code class="kt">GetterTree</code><code class="o"><</code><code class="nx">LocalesStateInterface</code><code class="p">,</code> <code class="nx">RootStateInterface</code><code class="o">></code> <code class="o">=</code> <code class="p">{}</code>

<code class="p">...</code>

<code></code><code class="p">...</code>

<code class="kr">const</code> <code class="nx">namespaced</code>: <code class="kt">boolean</code> <code class="o">=</code> <code class="kc">true</code>

<code class="kr">const</code> <code class="nx">state</code>: <code class="kt">LocalesStateInterface</code> <code
```

```
class="o">=</code> <code
class="nx">initialLocalesState</code>



<code class="kr">export</code> <code
class="kr">const</code> <code
class="nx">localesState</code>: <code
class="kt">Module</code><code class="o"><</code>
<code class="nx">LocalesStateInterface</code><code
class="p">,</code> <code
class="nx">RootStateInterface</code><code class="o">>
</code> <code class="o">=</code> <code class="p">
{</code>

  <code class="nx">namespaced</code><code
class="p">,</code>

  <code class="nx">state</code><code class="p">,
</code>

  <code class="nx">getters</code><code class="p">,
</code>

  <code class="nx">actions</code><code class="p">,
</code>

  <code class="nx">mutations</code>

<code class="p">}</code>

<code></code><code class="kr">import</code> <code
class="p">{</code> <code class="nx">rootStore</code>
<code class="p">,</code> <code
class="nx">dispatchModuleAction</code> <code
class="p">}</code> <code class="nx">from</code>
<code class="s1">'../root'</code>
```

<code class="kr">import</code> <code class="p">{</code> <code class="nx">MutationType</code><code class="p">,</code> <code class="nx">StoreModuleNames</code><code class="p">,</code> <code class="nx">LocalesStateInterface</code> <code class="p">}</code> <code class="nx">from</code> <code class="s1">'@/models/stor\</code>

<code class="s1">e'</code>

<code class="kr">import</code> <code class="p">{</code> <code class="nx">LocalStorageKeys</code> <code class="p">}</code> <code class="nx">from</code> <code class="s1">'@/models/local-storage/LocalStorageKeys'</code>

<code class="nx">The</code> <code class="nx">locales</code> <code class="nx">store</code> <code class="nx">wrapper</code> <code class="nx">that</code> <code class="nx">returns</code> <code class="nx">the</code> <code class="nx">localesState</code> <code class="nx">and</code> <code class="nx">exposes</code> <code class="nx">a</code> <code class="nx">generic</code> <code class="nx">action</code><code class="o">\</code>

<code class="o"><</code><code class="nx">T</code> <code class="o">></code> <code class="nx">method</code>

```
<code class="kr">const</code> <code class="nx">localesStore</code> <code class="o">=</code> <code class="p">{</code>

   <code class="nx">get</code> <code class="nx">state</code><code class="p">()</code> <code class="o">:</code> <code class="nx">LocalesStateInterface</code> <code class="p">{</code>

   <code class="k">return</code> <code class="nx">rootStore</code><code class="p">.</code> <code class="nx">state</code><code class="p">.</code><code class="nx">localesState</code>

   <code class="p">},</code>

   <code class="nx">action</code><code class="o"><</code><code class="nx">T</code><code class="o">></code><code class="p">(</code><code class="nx">actionName</code>: <code class="kt">string</code><code class="p">,</code> <code class="nx">params?</code>: <code class="kt">T</code><code class="p">)</code><code class="o">:</code> <code class="k">void</code> <code class="p">{</code>

   <code class="nx">dispatchModuleAction</code><code class="p">(</code><code class="nx">StoreModuleNames</code><code class="p">.</code><code class="nx">localesState</code><code class="p">,</code> <code class="nx">actionName</code><code class="p">,</code> <code class="nx">params</code><code class="p">)</code>

   <code class="p">}</code>
```

<code class="p">}</code>

<code class="c1">// export our wrapper using the composition API convention (i.e. useXYZ)</code>

<code class="kr">export</code> <code class="kr">const</code> <code class="nx">useLocalesStore</code> <code class="o">=</code> <code class="p">()</code> <code class="o">=></code> <code class="p">{</code>

   <code class="k">return</code> <code class="nx">localesStore</code>

<code class="p">}</code>

<code></code><code class="p">...</code>

<code class="kr">import</code> <code class="p">{</code>

   <code class="nx">RootStateInterface</code><code class="p">,</code>

   <code class="nx">RootStoreInterface</code><code class="p">,</code>

   <code class="nx">RootStoreModel</code><code class="p">,</code>

   <code class="nx">StoreModuleNames</code><code class="p">,</code>

   <code class="nx">MutationType</code><code class="p">,</code>

<code class="nx">ItemsStateInterface</code><code class="p">,</code>

<code class="nx">LocalesStateInterface</code> <code class="c1">// <-- add this line</code>

<code class="p">}</code> <code class="nx">from</code> <code class="s1">'@/models/store'</code>

<code class="p">...</code>

<code class="c1">// import each Vuex module</code>

<code class="kr">import</code> <code class="p">{</code> <code class="nx">itemsState</code> <code class="p">}</code> <code class="nx">from</code> <code class="err">‘@</code><code class="o">/</code> <code class="nx">store</code><code class="o">/</code><code class="nx">items</code> <code class="o">/</code><code class="nx">module</code><code class="s1">'</code>

<code class="s1">import { localesState } from ‘@/store/locales/module’ // <-- add this line</code>

<code class="s1">// Vuex store options to build our modularized namespaced store</code>

<code class="s1">const storeOptions: StoreOptions<RootStateInterface> = {</code>

<code class="s1"> state: initialRootState,</code>

```
<code class="s1"> modules: {</code>

<code class="s1"> itemsState,</code>

<code class="s1"> localesState // <-- add this line</code>

<code class="s1"> // as you create additional modules,
you'</code><code class="nx">ll</code> <code
class="nx">add</code> <code class="nx">them</code>
<code class="nx">here</code> <code
class="nx">similar</code> <code class="nx">to</code>
<code class="nx">the</code> <code
class="nx">itemsSt</code><code class="o">\</code>

<code class="nx">ate</code>

   <code class="p">}</code>

<code class="p">}</code>

<code class="p">...</code>

<code></code><code class="p"><</code><code
class="nt">template</code><code class="p">></code>

   <code class="p"><</code><code
class="nt">div</code> <code class="na">class</code>
<code class="o">=</code><code class="s">"locale-
selector"</code><code class="p">></code>

   <code class="p"><</code><code
class="nt">div</code> <code class="na">class</code>
<code class="o">=</code><code class="s">"locale-radio-
group"</code><code class="p">></code>

   <code class="p"><</code><code
class="nt">LocaleFlagRadio</code>
```

```
   <code class="na">v-for</code><code class="o">=
</code><code class="s">"(localeInfo, index) in
availableLocales"</code>

   <code class="na">:key</code><code class="o">=
</code><code class="s">"index"</code>

   <code class="na">:localeInfo</code><code class="o">=
</code><code class="s">"localeInfo"</code>

   <code class="err">@</code><code
class="na">clicked</code><code class="o">=</code>
<code class="s">"onFlagClicked"</code>

   <code class="p">/></code>

   <code class="p"></</code><code
class="nt">div</code><code class="p">></code>

   <code class="p"></</code><code
class="nt">div</code><code class="p">></code>

<code class="p"></</code><code
class="nt">template</code><code class="p">></code>

...

<code></code><code class="p">...</code>

<code class="o"><</code><code
class="nx">script</code> <code class="nx">lang</code>
<code class="o">=</code><code class="s2">"ts"
</code><code class="o">></code>

   <code class="kr">import</code> <code class="p">
{</code> <code class="nx">defineComponent</code>
<code class="p">,</code> <code
```

class="nx">reactive</code><code class="p">,</code> <code class="nx">computed</code><code class="p">, </code> <code class="nx">ref</code> <code class="p">}</code> <code class="nx">from</code> <code class="s1">'vue'</code>

<code class="kr">import</code> <code class="p">{</code> <code class="nx">useI18n</code> <code class="p">}</code> <code class="nx">from</code> <code class="s1">'vue-i18n'</code>

<code class="kr">import</code> <code class="p">{</code> <code class="nx">LocaleInfoInterface</code> <code class="p">}</code> <code class="nx">from</code> <code class="s1">'@/models/localization/LocaleInfo.interface'</code>

<code class="kr">import</code> <code class="nx">LocaleFlagRadio</code> <code class="nx">from</code> <code class="s1">'./LocaleFlagRadio.component.vue'</code>

<code class="kr">export</code> <code class="k">default</code> <code class="nx">defineComponent</code><code class="p">({</code>

<code class="nx">components</code><code class="o">:</code> <code class="p">{</code>

<code class="nx">LocaleFlagRadio</code>

<code class="p">},</code>

<code class="nx">props</code><code class="o">:
</code> <code class="p">{</code>

<code class="nx">availableLocales</code><code
class="o">:</code> <code class="p">{</code>

<code class="nx">type</code>: <code
class="kt">Array</code>

<code class="p">}</code>

<code class="p">},</code>

<code class="nx">setup</code><code class="p">
(</code><code class="nx">props</code><code
class="p">,</code> <code class="p">{</code> <code
class="nx">emit</code> <code class="p">})</code>
<code class="p">{</code>

<code class="kr">const</code> <code
class="nx">i18n</code> <code class="o">=</code>
<code class="nx">useI18n</code><code class="p">()
</code>

<code class="kr">const</code> <code
class="nx">onFlagClicked</code> <code class="o">=
</code> <code class="p">(</code><code
class="nx">localeInfo</code>: <code
class="kt">LocaleInfoInterface</code><code class="p">)
</code> <code class="o">=></code> <code class="p">
{</code>

<code class="nx">emit</code><code class="p">
(</code><code class="s1">'clicked'</code><code

```
class="p">,</code> <code class="nx">localeInfo</code>
<code class="p">)</code>

   <code class="p">}</code>



   <code class="k">return</code> <code class="p">
{</code>

   <code class="nx">onFlagClicked</code>

   <code class="p">}</code>

   <code class="p">}</code>

   <code class="p">})</code>

<code class="o"><</code><code class="err">/script>
</code>

<code class="p">...</code>

<code></code><code class="o"><</code><code
class="nt">style</code> <code class="nt">lang</code>
<code class="o">=</code><code class="s2">"scss"
</code><code class="o">></code>

   <code class="nc">.locale-selector</code> <code
class="p">{</code>

   <code class="nt">display</code><code class="nd">:
</code> <code class="nt">inline-flex</code><code
class="o">;</code>
```

```
   <code class="nc">.locale-radio-group</code> <code
class="p">{</code>

   <code class="nt">display</code><code class="nd">:
</code> <code class="nt">inline-flex</code><code
class="o">;</code>

   <code class="nt">justify-content</code><code
class="nd">:</code> <code class="nt">center</code>
<code class="o">;</code>



   <code class="nt">label</code><code
class="nc">.locale-radio</code> <code class="p">
{</code>

   <code class="nt">cursor</code><code class="nd">:
</code> <code class="nt">pointer</code><code
class="o">;</code>

   <code class="nt">display</code><code class="nd">:
</code> <code class="nt">block</code><code
class="o">;</code>

   <code class="nt">padding</code><code class="nd">:
</code> <code class="nt">5px</code><code class="o">;
</code>



   <code class="k">&</code><code
class="nc">.selected</code> <code class="p">{</code>

   <code class="nt">border-bottom</code><code
class="nd">:</code> <code class="nt">solid</code>
<code class="nt">5px</code> <code
class="nn">#42b983</code><code class="o">;</code>

   <code class="p">}</code>

   <code class="p">}</code>



   <code class="nt">input</code> <code class="p">
{</code>

   <code class="nt">display</code><code class="nd">:
</code> <code class="nt">none</code><code
class="o">;</code>

   <code class="p">}</code>

   <code class="p">}</code>

   <code class="p">}</code>

<code class="o"></</code><code
class="nt">style</code><code class="o">></code>

<code></code><code class="p"><</code><code
class="nt">template</code><code class="p">></code>

   <code class="p"><</code><code
class="nt">label</code>

   <code class="na">role</code><code class="o">=
</code><code class="s">"radio"</code>

   <code class="na">:class</code><code class="o">=
</code><code class="s">"`locale-radio
${localeInfo.selected ? 'selected' : ''}`.trim()"</code>
```

```
   <code class="p">></code>

   <code class="p"><</code><code class="nt">i</code>
<code class="na">:class</code><code class="o">=
</code><code class="s">"`flag-icons ${localeInfo.flag}`"
</code><code class="p">></</code><code
class="nt">i</code><code class="p">></code>

   <code class="p"><</code><code
class="nt">input</code>

   <code class="na">type</code><code class="o">=
</code><code class="s">"radio"</code>

   <code class="na">class</code><code class="o">=
</code><code class="s">"icon-button"</code>

   <code class="na">name</code><code class="o">=
</code><code class="s">"locale"</code>

   <code class="na">:value</code><code class="o">=
</code><code class="s">"localeInfo.selected"</code>

   <code class="na">v-model</code><code class="o">=
</code><code class="s">"localeInfo.selected"</code>

   <code class="err">@</code><code
class="na">click</code><code class="o">=</code>
<code class="s">"onClick"</code>

   <code class="p">/></code>

   <code class="p"></</code><code
class="nt">label</code><code class="p">></code>

<code class="p"></</code><code
class="nt">template</code><code class="p">></code>
```

...

<code></code><code class="p">...</code>

<code class="o"><</code><code class="nx">script</code> <code class="nx">lang</code> <code class="o">=</code><code class="s2">"ts"</code><code class="o">></code>

   <code class="kr">import</code> <code class="p">{</code> <code class="nx">defineComponent</code> <code class="p">,</code> <code class="nx">reactive</code><code class="p">,</code> <code class="nx">computed</code><code class="p">,</code> <code class="nx">ref</code> <code class="p">}</code> <code class="nx">from</code> <code class="s1">'vue'</code>

   <code class="kr">import</code> <code class="p">{</code> <code class="nx">useI18n</code> <code class="p">}</code> <code class="nx">from</code> <code class="s1">'vue-i18n'</code>

   <code class="kr">import</code> <code class="p">{</code> <code class="nx">LocaleInfoInterface</code> <code class="p">}</code> <code class="nx">from</code> <code class="s1">'@/models/localization/LocaleInfo.interface'</code>

   <code class="kr">import</code> <code class="p">{</code> <code class="nx">i18n</code> <code class="p">}</code> <code class="nx">from</code> <code class="s1">'@/plugins/vue-i18n-next-plugin'</code>

```
   <code class="kr">export</code> <code
class="k">default</code> <code
class="nx">defineComponent</code><code class="p">
({</code>

   <code class="nx">props</code><code class="o">:
</code> <code class="p">{</code>

   <code class="nx">localeInfo</code><code class="o">:
</code> <code class="p">{</code>

   <code class="nx">type</code>: <code
class="kt">Object</code>

   <code class="p">}</code>

   <code class="p">},</code>

   <code class="nx">setup</code><code class="p">
(</code><code class="nx">props</code><code
class="p">,</code> <code class="p">{</code> <code
class="nx">emit</code> <code class="p">})</code>
<code class="p">{</code>

   <code class="kr">const</code> <code
class="nx">i18n</code> <code class="o">=</code>
<code class="nx">useI18n</code><code class="p">()
</code>



   <code class="kr">const</code> <code
class="nx">onClick</code> <code class="o">=</code>
<code class="p">()</code> <code class="o">=></code>
<code class="p">{</code>
```

```
  <code class="nx">emit</code><code class="p">(</code><code class="s1">'clicked'</code><code class="p">,</code> <code class="nx">props</code><code class="p">.</code><code class="nx">localeInfo</code><code class="p">)</code>

  <code class="p">}</code>



  <code class="k">return</code> <code class="p">{</code>

  <code class="nx">onClick</code>

  <code class="p">}</code>

  <code class="p">}</code>

  <code class="p">})</code>

<code class="o"><</code><code class="err">/script></code>

<code></code><code class="kr">import</code> <code class="p">{</code> <code class="nx">createApp</code> <code class="p">}</code> <code class="nx">from</code> <code class="s1">'vue'</code>

<code class="kr">import</code> <code class="nx">App</code> <code class="nx">from</code> <code class="s1">'./App.vue'</code>

<code class="kr">import</code> <code class="nx">router</code> <code class="nx">from</code> <code class="s1">'./router'</code>
```

```
<code class="kr">import</code> <code class="p">
{</code> <code class="nx">store</code> <code
class="p">}</code> <code class="nx">from</code>
<code class="s1">'./store'</code>

<code class="kr">import</code> <code class="p">
{</code> <code class="nx">i18n</code> <code
class="p">}</code> <code class="nx">from</code>
<code class="s1">'@/plugins/vue-i18n-next-plugin'</code>

<code class="kr">import</code> <code class="p">
{</code> <code class="nx">FlagIconsScss</code> <code
class="p">}</code> <code class="nx">from</code>
<code class="s1">'@/plugins/flags-icons/'</code>



<code class="nx">createApp</code><code class="p">
(</code><code class="nx">App</code><code
class="p">)</code>

   <code class="p">.</code><code
class="nx">use</code><code class="p">(</code><code
class="nx">store</code><code class="p">)</code>

   <code class="p">.</code><code
class="nx">use</code><code class="p">(</code><code
class="nx">router</code><code class="p">)</code>

   <code class="p">.</code><code
class="nx">use</code><code class="p">(</code><code
class="nx">i18n</code><code class="p">)</code>

   <code class="p">.</code><code
class="nx">use</code><code class="p">(</code><code
class="nx">FlagIconsScss</code><code class="p">)
</code>
```

```
   <code class="p">.</code><code
class="nx">mount</code><code class="p">(</code>
<code class="s1">'#app'</code><code class="p">)
</code>

<code></code><code class="p"><</code><code
class="nt">template</code><code class="p">></code>

   <code class="p"><</code><code
class="nt">div</code> <code class="na">id</code>
<code class="o">=</code><code class="s">"app"
</code><code class="p">></code>

   <code class="p"><</code><code
class="nt">h2</code><code class="p">></code>{{
i18n.t('welcome') }}<code class="p"></</code><code
class="nt">h2</code><code class="p">></code>

   <code class="c"><!-- begin: add block --></code>

   <code class="p"><</code><code
class="nt">LocaleSelector</code>

   <code class="na">:availableLocales</code><code
class="o">=</code><code class="s">"availableLocales"
</code>

   <code class="err">@</code><code
class="na">clicked</code><code class="o">=</code>
<code class="s">"onLocaleClicked"</code>

   <code class="p">/></code>

   <code class="c"><!-- end: add block --></code>

   <code class="p"><</code><code
class="nt">div</code> <code class="na">id</code>
```

```
<code class="o">=</code><code class="s">"nav"
</code> <code class="na">class</code><code
class="o">=</code><code class="s">"nav"</code>
<code class="p">></code>

   <code class="p"><</code><code class="nt">router-
link</code> <code class="na">to</code><code
class="o">=</code><code class="s">"/"</code><code
class="p">></code>Home<code class="p"></</code>
<code class="nt">router-link</code><code class="p">>
</code> |

   <code class="p"><</code><code class="nt">router-
link</code> <code class="na">to</code><code
class="o">=</code><code class="s">"/about"</code>
<code class="p">></code>About<code class="p">
</</code><code class="nt">router-link</code><code
class="p">></code>

   <code class="p"></</code><code
class="nt">div</code><code class="p">></code>

   <code class="p"><</code><code class="nt">router-
view</code> <code class="p">/></code>

   <code class="p"></</code><code
class="nt">div</code><code class="p">></code>

<code class="p"></</code><code
class="nt">template</code><code class="p">></code>



...

<code></code><code class="p">...</code>
```

```
<code class="o"><</code><code
class="nx">script</code> <code class="nx">lang</code>
<code class="o">=</code><code class="s2">"ts"
</code><code class="o">></code>

   <code class="kr">import</code> <code class="p">
{</code> <code class="nx">defineComponent</code>
<code class="p">,</code> <code
class="nx">computed</code> <code class="p">}</code>
<code class="nx">from</code> <code
class="s1">'vue'</code> <code class="c1">// add
"computed" in the import li\</code>

<code class="nx">st</code>

<code class="p">...</code>

   <code class="c1">// begin: add lines</code>

   <code class="kr">import</code> <code class="p">
{</code> <code class="nx">MutationType</code> <code
class="p">}</code> <code class="nx">from</code>
<code class="s1">'@/models/store'</code>

   <code class="kr">import</code> <code class="p">
{</code> <code class="nx">useLocalesStore</code>
<code class="p">}</code> <code
class="nx">from</code> <code class="err">‘@</code>
<code class="o">/</code><code
class="nx">store</code><code class="o">/</code><code
class="nx">locales</code><code class="s1">'</code>

<code class="s1"> import { LocaleInfoInterface } from
'</code><code class="err">@</code><code
class="o">/</code><code class="nx">models</code>
<code class="o">/</code><code
class="nx">localization</code><code class="o">/</code>
```

<code class="nx">LocaleInfo</code><code class="p">.
</code><code class="kr">interface</code><code
class="s1">'</code>

<code class="s1"> import LocaleSelector from '</code>
<code class="err">@</code><code class="o">/</code>
<code class="nx">components</code><code
class="o">/</code><code class="nx">locale</code>
<code class="o">-</code><code
class="nx">selector</code><code class="o">/</code>
<code class="nx">LocaleSelector</code><code
class="p">.</code><code class="nx">component</code>
<code class="p">.</code><code class="o">\</code>

<code class="nx">vue</code><code
class="err">'</code>

   <code class="c1">// end: add lines</code>

<code class="p">...</code>

<code></code> <code class="p">...</code>

   <code class="kr">export</code> <code
class="k">default</code> <code
class="nx">defineComponent</code><code class="p">
({</code>

   <code class="nx">name</code><code class="o">:
</code> <code class="s1">'App'</code><code
class="p">,</code>

   <code class="nx">components</code><code
class="o">:</code> <code class="p">{</code>

   <code class="nx">LocaleSelector</code>

```
<code class="p">},</code>

<code class="nx">setup() {</code>

<code class="kr">const</code> <code class="nx">i18n</code> <code class="o">=</code> <code class="nx">useI18n</code><code class="p">()</code>

<code class="kr">const</code> <code class="nx">localesStore</code> <code class="o">=</code> <code class="nx">useLocalesStore</code><code class="p">()</code>



<code class="kr">const</code> <code class="nx">availableLocales</code> <code class="o">=</code> <code class="nx">computed</code><code class="p">(()</code> <code class="o">=></code> <code class="p">{</code>

<code class="k">return</code> <code class="nx">localesStore</code><code class="p">.</code><code class="nx">state</code><code class="p">.</code><code class="nx">availableLocales</code>

<code class="p">})</code>



<code class="kr">const</code> <code class="nx">onLocaleClicked</code> <code class="o">=</code> <code class="p">(</code><code class="nx">localeInfo</code>: <code class="kt">LocaleInfoInterface</code><code class="p">)
```

</code> <code class="o">=></code> <code class="p">{</code>

<code class="nx">localesStore</code><code class="p">.</code><code class="nx">action</code> <code class="p">(</code>

<code class="nx">MutationType</code><code class="p">.</code><code class="nx">locales</code> <code class="p">.</code><code class="nx">selectLocale</code><code class="p">, </code>

<code class="nx">localeInfo</code><code class="p">. </code><code class="nx">locale</code>

<code class="p">)</code>

<code class="p">}</code>

<code class="k">return</code> <code class="p">{</code>

<code class="nx">i18n</code><code class="p">, </code>

<code class="nx">availableLocales</code><code class="p">,</code>

<code class="nx">onLocaleClicked</code>

<code class="p">}</code>

<code class="p">}</code>

<code class="p">})</code>

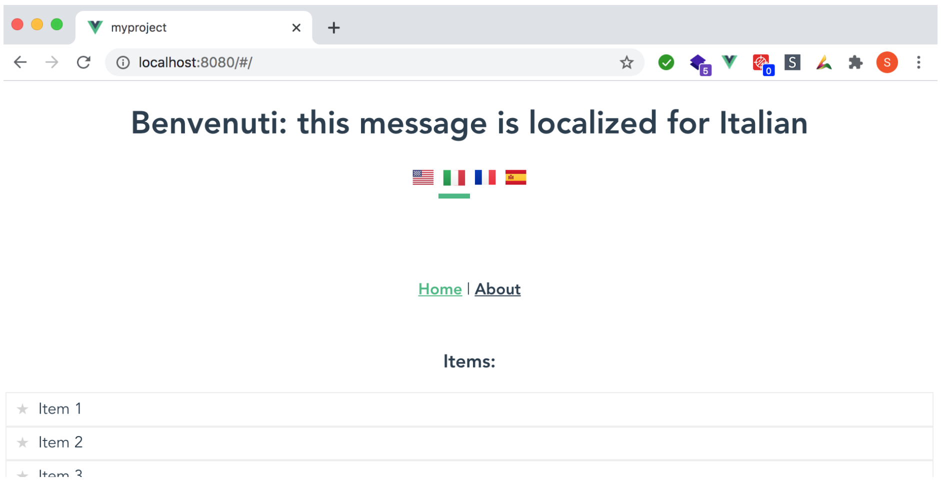
<code class="o"><</code><code class="err">/script></code>

Note that you could display this component anywhere you like, but I decided to show in the main App.vue so it is always available through the whole app even when navigating to other views.

Browser

Run the app again with **npm run serve**

If everything is in the right place, now you should see the flag icons displayed at the top and be able to click on the different country flags to quickly switch between different locales:



Benvenuti: this message is localized for Italian

Home | About

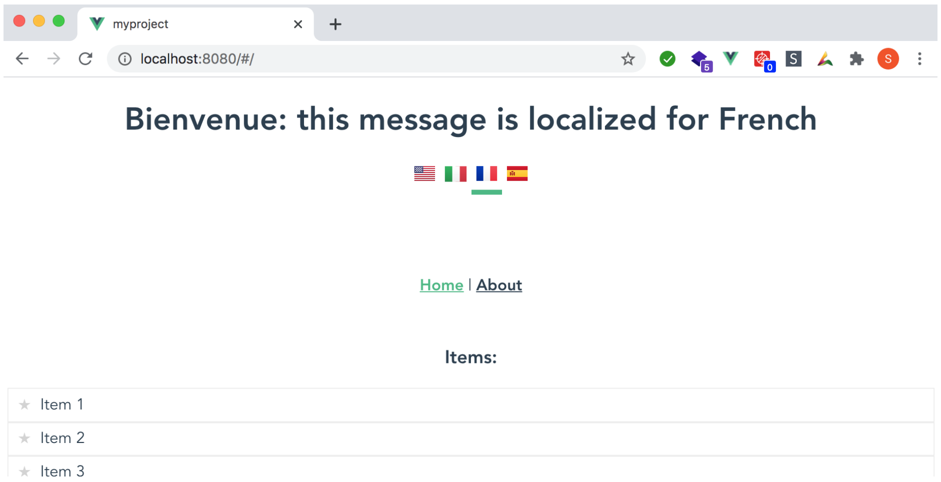
Items:

Item 1

Item 2

Item 3

Try clicking on the French flag for example , and it should display the welcome message at the top in French:



Bienvenue: this message is localized for French

Home | About

Items:

★ Item 1

★ Item 2

★ Item 3

Chapter 11 Recap

What We Learned

- How to add a custom component with custom radio buttons that allow selecting a different **Locale**
- How to use an external **SCSS** file with **.svg** icons
- How to setup an additional **Vuex** module to handle the selected Locale state and add ability to switch to different locales

Observations

- We did not add unit tests for our new **LocaleFlagRadio** and **LocaleSelector** components
- Our messages data of type **LocalesDataInterface** is hard-coded within the **vue-i18n-next-plugin/index.ts** and this would make it harder to maintain as we add more translated messages

Based on these observations, there are a few improvements that you could do on your own:

Improvements

- Add a unit test for the **LocaleFlagRadio** to ensure it renders the correct flag
- Add a unit test for the **LocaleSelector** component to ensure it renders a list of **LocalFlagRadio** buttons with the correct flags
- Improve how we create our instance of **LocalesDataInterface** data with the translated messages.

<code></code><code class="p">...</code> <code class="kr">interface</code> <code class="nx">LocalesDataInterface</code> <code class="p">{</code> <code class="nx">messages</code>: <code class="kt">LocaleMessages</code><code class="o"><</code><code class="nx">VueMessageType</code> <code class="o">></code> <code class="p">}</code>

<code class="kr">const</code> <code class="nx">data</code>: <code class="kt">LocalesDataInterface</code> <code class="o">=</code> <code class="p">{</code> <code class="nx">messages</code><code class="o">:</code> <code class="p">{</code> <code class="s1">'en-US'</code><code class="o">:</code> <code class="p">{</code> <code class="nx">welcome</code><code class="o">:</code> <code class="s1">'Welcome: this message is localized in English'</code> <code class="p">},</code> <code class="s1">'it-IT'</code> <code class="o">:</code> <code class="p">{</code> <code class="nx">welcome</code><code class="o">:</code> <code class="s1">'Benvenuti: this message is localized in Italian'</code> <code class="p">},</code> <code class="s1">'fr-FR'</code><code class="o">:</code> <code class="p">{</code> <code class="nx">welcome</code><code class="o">:</code> <code class="s1">'Bienvenue: this message is localized in French'</code> <code class="p">},</code> <code class="s1">'es-ES'</code><code class="o">:</code> <code class="p">{</code> <code class="nx">welcome</code><code class="o">:</code>

<code class="s1">'Bienvenido: this message is localized in Spanish'</code> <code class="p">}</code>

   <code class="p">}</code>

<code class="p">}</code>

<code class="p">...</code>

<code></code><code class="p">{</code> <code class="nt">"messages"</code><code class="p">:</code> <code class="p">{</code> <code class="nt">"welcome"</code><code class="p">:</code> <code class="s2">"Welcome: this message is localized in English"</code> <code class="p">}</code>

<code class="p">}</code>

<code></code><code class="p">{</code> <code class="nt">"messages"</code><code class="p">:</code> <code class="p">{</code> <code class="nt">"welcome"</code><code class="p">:</code> <code class="s2">"Benvenuti: this message is localized in Italian"</code> <code class="p">}</code>

<code class="p">}</code>

<code></code><code class="p">{</code> <code class="nt">"messages"</code><code class="p">:</code> <code class="p">{</code> <code class="nt">"welcome"</code><code class="p">:</code> <code class="s2">"Bienvenue: this message is localized in French"</code> <code class="p">}</code>

<code class="p">}</code>

<code></code><code class="p">{</code> <code class="nt">"messages"</code><code class="p">:</code> <code class="p">{</code> <code class="nt">"welcome"</code><code class="p">:</code> <code class="s2">"Bienvenido: this message is localized in Spanish"</code> <code class="p">}</code>

<code class="p">}</code>

<code></code><code class="kr">import</code> <code class="p">{</code> <code class="nx">createI18n</code><code class="p">,</code> <code class="nx">LocaleMessages</code><code class="p">,</code> <code class="nx">VueMessageType</code> <code class="p">}</code> <code class="nx">from</code> <code class="s1">'vue-i18n'</code>

<code class="kr">interface</code> <code class="nx">LocalesDataInterface</code> <code class="p">{</code> <code class="nx">messages</code>: <code class="kt">LocaleMessages</code><code class="o"><</code><code class="nx">VueMessageType</code><code class="o">></code> <code class="p">}</code>

<code class="cm">/**</code>

<code class="cm"> * @name: getLocalesData</code> <code class="cm"> * @description: Helper to load the locale json files …</code> <code class="cm"> */</code>

<code class="kr">const</code> <code class="nx">getLocalesData</code> <code class="o">=</code> <code class="p">()</code><code class="o">:

```
</code> <code class="nx">LocalesDataInterface</code>
<code class="o">=></code> <code class="p">{</code>
<code class="c1">// use require.context to get all the .json
files ...</code> <code class="kr">const</code> <code
class="nx">files</code> <code class="o">=</code>
<code class="p">(</code><code
class="nx">require</code> <code class="kr">as</code>
<code class="nx">any</code><code class="p">).
</code><code class="nx">context</code><code
class="p">(</code><code class="s1">'./locales'</code>
<code class="p">,</code> <code
class="kc">true</code><code class="p">,</code> <code
class="sr">/[A-Za-z0-9-_,\s]+\.json\</code> <code
class="sr">$/i</code><code class="p">)</code> <code
class="c1">// create the instance that will hold the loaded
data</code> <code class="kr">const</code> <code
class="nx">localeData</code>: <code
class="kt">LocalesDataInterface</code> <code
class="o">=</code> <code class="p">{</code> <code
class="nx">messages</code><code class="o">:</code>
<code class="p">{}</code> <code class="p">}</code>

   <code class="c1">// loop through all the files</code>
<code class="kr">const</code> <code
class="nx">keys</code>: <code
class="kt">string</code><code class="p">[]</code>
<code class="o">=</code> <code
class="nx">files</code><code class="p">.</code><code
class="nx">keys</code><code class="p">()</code>
<code class="nx">keys</code><code class="p">.
</code><code class="nx">forEach</code><code
class="p">((</code><code class="nx">key</code>:
<code class="kt">string</code><code class="p">)
</code> <code class="o">=></code> <code class="p">
{</code> <code class="c1">// extract name without
extension</code> <code class="kr">const</code> <code
```

class="nx">matched</code> <code class="o">=</code> <code class="nx">key</code><code class="p">.</code> <code class="nx">match</code><code class="p">(</code><code class="sr">/([A-Za-z0-9-_]+)\./i</code> <code class="p">)</code> <code class="k">if</code> <code class="p">(</code><code class="nx">matched</code> <code class="o">&&</code> <code class="nx">matched</code><code class="p">.</code><code class="nx">length</code> <code class="o">></code> <code class="mi">1</code> <code class="p">)</code> <code class="p">{</code> <code class="kr">const</code> <code class="nx">localeId</code> <code class="o">=</code> <code class="nx">matched</code><code class="p">[</code><code class="mi">1</code><code class="p">]</code> <code class="c1">// from each file, set the related messages property </code> <code class="nx">localeData</code><code class="p">.</code> <code class="nx">messages</code><code class="p">[</code><code class="nx">localeId</code><code class="p">]</code> <code class="o">=</code> <code class="nx">files</code><code class="p">(</code><code class="nx">key</code><code class="p">).</code><code class="nx">messages</code> <code class="p">}</code>

<code class="p">})</code>

<code class="k">return</code> <code class="nx">localeData</code> <code class="p">}</code>

<code class="err">…</code>

...

// begin: remove code block

```
<code class="udl">const data: LocalesDataInterface = {

</code><code class="udl"> messages: {

</code><code class="udl"> 'en-US': {

</code><code class="udl"> welcome: 'Welcome: this message is localized in English'

</code><code class="udl"> },

</code><code class="udl"> 'it-IT': {

</code><code class="udl"> welcome: 'Benvenuti: this message is localized in Italian'

</code><code class="udl"> },

</code><code class="udl"> 'fr-FR': {

</code><code class="udl"> welcome: 'Bienvenue: this message is localized in French'

</code><code class="udl"> },

</code><code class="udl"> 'es-ES': {

</code><code class="udl"> welcome: 'Bienvenido: this message is localized in Spanish'

</code><code class="udl"> }

</code><code class="udl"> }

</code><code class="udl">}

</code>// end: remove code block
```

<code class="uil">// create our data dynamically by loading the JSON files through our getLocalesData \ </code><code class="uil">helper

</code><code class="uil">const data: LocalesDataInterface = getLocalesData() </code>...

<code></code><code class="p"><</code><code class="nt">template</code><code class="p">></code> <code class="p"><</code><code class="nt">div</code> <code class="na">id</code><code class="o">=</code> <code class="s">"app"</code><code class="p">></code> <code class="p"><</code><code class="nt">h2</code><code class="p">></code>{{ i18n.t('welcome') }}<code class="p"></</code><code class="nt">h2</code><code class="p">></code> <code class="p"><</code><code class="nt">div</code> <code class="na">id</code><code class="o">=</code><code class="s">"nav"</code> <code class="na">class</code> <code class="o">=</code><code class="s">"nav" </code><code class="p">></code> <code class="p"> <</code><code class="nt">router-link</code> <code class="na">to</code><code class="o">=</code><code class="s">"/"</code><code class="p">> </code>Home<code class="p"></</code><code class="nt">router-link</code><code class="p">></code> |

   <code class="p"><</code><code class="nt">router-link</code> <code class="na">to</code><code class="o">=</code><code class="s">"/about"</code> <code class="p">></code>About<code class="p"> </</code><code class="nt">router-link</code><code class="p">></code> <code class="p"></</code><code class="nt">div</code><code class="p">></code> ...

```
{

   "messages": {

   "welcome": "Welcome: this message is localized in
English", <code class="uil"> "navigation": {

</code><code class="uil"> "home": "Home",

</code><code class="uil"> "about": "About"

</code><code class="uil"> }

</code> }

}

{

   "messages": {

   "welcome": "Benvenuti: this message is localized in
Italian", <code class="uil"> "navigation": {

</code><code class="uil"> "home": "Home",

</code><code class="uil"> "about": "Chi Siamo"

</code><code class="uil"> }

</code> }

}

{

   "messages": {
```

```
   "welcome": "Bienvenue: this message is localized in
French", <code class="uil"> "navigation": {

</code><code class="uil"> "home": "Accueil",

</code><code class="uil"> "about": "À propos de nous"

</code><code class="uil"> }

</code> }

}

{

   "messages": {

   "welcome": "Bienvenido: this message is localized in
Spanish", <code class="uil"> "navigation": {

</code><code class="uil"> "home": "Inicio",

</code><code class="uil"> "about": "Acerca de"

</code><code class="uil"> }

</code> }

}

<template>

   <div id="app">

   <h2>{{ i18n.t('welcome') }}</h2> <div id="nav"
class="nav"> <code class="udl"> <!-- begin: remove code
block --> </code><code class="udl"> <router-link
to="/">Home</router-link> |
</code><code class="udl"> <router-link
to="/about">About</router-link> </code><code
class="udl"> <!-- end: remove code block --> </code>
<code class="uil"> <!-- begin: add code block --> </code>
<code class="uil"> <router-link to="/">{{
i18n.t('navigation.home') }}</router-link> |
```

```
</code><code class="uil"> <router-link to="/about">{{
i18n.t('navigation.about') }}</router-link> </code><code
class="uil"> <!-- end: add code block --> </code> </div>
```

...

```
<code></code><code class="p"><</code><code
class="nt">template</code><code class="p">></code>
<code class="p"><</code><code class="nt">div</code>
<code class="p">></code> <code class="p"><</code>
<code class="nt">h3</code><code class="p">>
</code>My Items:<code class="p"></</code><code
class="nt">h3</code><code class="p">></code> <code
class="p"><</code><code class="nt">Loader</code>
<code class="na">v-show</code><code class="o">=
</code><code class="s">"loading"</code> <code
class="p">/></code> ...
```

```
{

  "messages": {

  "welcome": "Welcome: this message is localized in
English", "navigation": {

  "home": "Home",

  "about": "About"

  },
<code class="uil"> "items": {

</code><code class="uil"> "list": {

</code><code class="uil"> "header": "My Items"

</code><code class="uil"> }

</code><code class="uil"> }

</code> }

}

{

   "messages": {

   "welcome": "Benvenuti: this message is localized in
Italian", "navigation": {

   "home": "Home",

   "about": "Chi Siamo"

   },

<code class="uil"> "items": {

</code><code class="uil"> "list": {

</code><code class="uil"> "header": "I miei articoli"

</code><code class="uil"> }

</code><code class="uil"> }

</code> }
```

```
}

{

   "messages": {

   "welcome": "Bienvenue: this message is localized in
French", "navigation": {

   "home": "Accueil",

   "about": "À propos de nous"

   },

<code class="uil"> "items": {

</code><code class="uil"> "list": {

</code><code class="uil"> "header": "Mes articles"

</code><code class="uil"> }

</code><code class="uil"> }

</code> }

}

{

   "messages": {

   "welcome": "Bienvenido: this message is localized in
Spanish", "navigation": {

   "home": "Inicio",

   "about": "Acerca de"
   },

<code class="uil"> "items": {

</code><code class="uil"> "list": {

</code><code class="uil"> "header": "Mis cosas"

</code><code class="uil"> }

</code><code class="uil"> }

</code> }

}

<template>

   <div>

<code class="udl"> <h3>My Items:</h3> <!-- delete this
line --> </code><code class="uil"> <h3>{{
i18n.t('items.list.header') }}:</h3> <!-- add this line -->
</code> <Loader v-show="loading" />

…



<script lang="ts">

   import { defineComponent, PropType } from 'vue'

   import { ItemInterface } from
'@/models/items/Item.interface'

   import ItemComponent from
'@/components/items/children/Item.component.vue'
```

import Loader from
'@/components/shared/Loader.component.vue'

<code class="uil"> import { useI18n } from 'vue-i18n' // <-- add this line </code>...

...

setup(props, { emit }) {

<code class="uil"> const i18n = useI18n() // <-- add this line </code> ...

return {

<code class="uil"> i18n, // <-- add this line

</code> onItemSelect

}

}

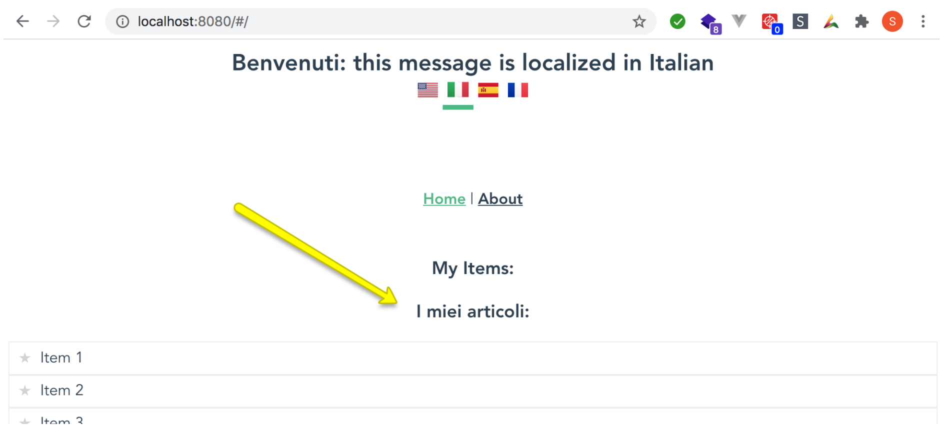
...

Not how here we have to use key **items.list.header** as that is how we structured our **JSON** data for that. We are doing this way to illustrate how to better organize all the locale messages by app domain. If you prefer you can organized them in other way. Mind you, there is not going to be a perfect way that serves all scenarios, but if your code base will grow large with many components, I suggest you start using this way or some way of organizing them or your code will quickly become hard to read and maintain.

Browser

Head over your web browser and verify the page is displaying the **ItemsList** component header in the different locales.

Here is in **Italian**:



localhost:8080/#/

Benvenuti: this message is localized in Italian

Home | About

My Items:

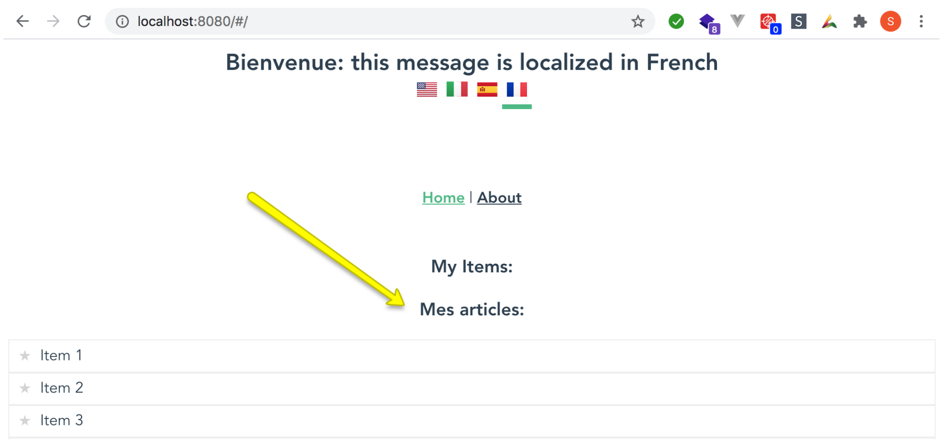
I miei articoli:

Item 1

Item 2

Item 3

Here is in **French**:



localhost:8080/#/

Bienvenue: this message is localized in French

Home | About

My Items:

Mes articles:

Item 1

Item 2

Item 3

And verify the other locales as well.

Chapter 12 Recap

What We Learned

- We learned how to localize labels for a component so that they are displayed in different languages
- We learned how to expand our locales **JSON** files in a structure and organized way

Observations

- We did not localize any number or date based on the selected **locale** yet
- We did not write any unit tests against our **getLocalesData** helper function
- We did not write any unit tests against our static **locales JSON** files to validate their structure and content

Based on these observations, there are a few improvements that we can:

Improvements

- In the next chapters we'll be adding a way to localize also numbers and dates formats
- Optional: You can write unit tests to ensure the **getLocalesData** helper function loads the **JSON** files correctly and set the messages as expected. You could also write some unit tests against the **JSON** files directly to make sure they contain the correct data structure.

{

<code class="uil"> "datetimeFormats": {

</code><code class="uil"> "long": {

</code><code class="uil"> "year": "numeric", </code><code class="uil"> "month": "2-digit", </code><code class="uil"> "day": "2-digit", </code><code class="uil"> "hour": "2-digit", </code><code class="uil"> "minute": "2-digit", </code><code class="uil"> "second": "2-digit", </code><code class="uil"> "hour12": true </code><code class="uil"> },

</code><code class="uil"> "short": {

</code><code class="uil"> "year": "numeric", </code><code class="uil"> "month": "2-digit", </code><code class="uil"> "day": "2-digit"

</code><code class="uil"> }

</code><code class="uil"> },

</code><code class="uil"> "numberFormats": {

</code><code class="uil"> "currency": {

</code><code class="uil"> "style": "currency", </code><code class="uil"> "currency": "USD", </code><code class="uil"> "currencyDisplay": "symbol", </code><code class="uil"> "minimumFractionDigits": 0, </code><code class="uil"> "maximumFractionDigits": 2

</code><code class="uil"> },

</code><code class="uil"> "decimal": {

```
</code><code class="uil"> "style": "decimal", </code>
<code class="uil"> "useGrouping": false, </code><code
class="uil"> "minimumFractionDigits": 0, </code><code
class="uil"> "maximumFractionDigits": 2

</code><code class="uil"> },

</code><code class="uil"> "numeric": {

</code><code class="uil"> "style": "decimal", </code>
<code class="uil"> "useGrouping": false, </code><code
class="uil"> "minimumFractionDigits": 0, </code><code
class="uil"> "maximumFractionDigits": 0

</code><code class="uil"> },

</code><code class="uil"> "percent": {

</code><code class="uil"> "style": "percent", </code>
<code class="uil"> "useGrouping": false, </code><code
class="uil"> "minimumFractionDigits": 0, </code><code
class="uil"> "maximumFractionDigits": 0

</code><code class="uil"> }

</code><code class="uil"> },

</code> "messages": {

  "welcome": "Welcome: this message is localized in
English", ...

{

<code class="uil"> "datetimeFormats": {

</code><code class="uil"> "long": {
```

```
</code><code class="uil"> "year": "numeric", </code>
<code class="uil"> "month": "2-digit", </code><code
class="uil"> "day": "2-digit", </code><code class="uil">
"hour": "2-digit", </code><code class="uil"> "minute": "2-
digit", </code><code class="uil"> "second": "2-digit",
</code><code class="uil"> "hour12": false </code><code
class="uil"> },

</code><code class="uil"> "short": {

</code><code class="uil"> "year": "numeric", </code>
<code class="uil"> "month": "2-digit", </code><code
class="uil"> "day": "2-digit"

</code><code class="uil"> }

</code><code class="uil"> },

</code><code class="uil"> "numberFormats": {

</code><code class="uil"> "currency": {

</code><code class="uil"> "style": "currency", </code>
<code class="uil"> "currency": "EUR", </code><code
class="uil"> "currencyDisplay": "symbol", </code><code
class="uil"> "minimumFractionDigits": 0, </code><code
class="uil"> "maximumFractionDigits": 2

</code><code class="uil"> },

</code><code class="uil"> "decimal": {

</code><code class="uil"> "style": "decimal", </code>
<code class="uil"> "useGrouping": false, </code><code
class="uil"> "minimumFractionDigits": 0, </code><code
class="uil"> "maximumFractionDigits": 2
```

```
</code><code class="uil"> },

</code><code class="uil"> "numeric": {

</code><code class="uil"> "style": "decimal", </code>
<code class="uil"> "useGrouping": false, </code><code
class="uil"> "minimumFractionDigits": 0, </code><code
class="uil"> "maximumFractionDigits": 0

</code><code class="uil"> },

</code><code class="uil"> "percent": {

</code><code class="uil"> "style": "percent", </code>
<code class="uil"> "useGrouping": false, </code><code
class="uil"> "minimumFractionDigits": 0, </code><code
class="uil"> "maximumFractionDigits": 0

</code><code class="uil"> }

</code><code class="uil"> },

</code> "messages": {

   "welcome": "Benvenuti: this message is localized in
Italian", ...

{

<code class="uil"> "datetimeFormats": {

</code><code class="uil"> "long": {

</code><code class="uil"> "year": "numeric", </code>
<code class="uil"> "month": "2-digit", </code><code
class="uil"> "day": "2-digit", </code><code class="uil">
"hour": "2-digit", </code><code class="uil"> "minute": "2-
digit", </code><code class="uil"> "second": "2-digit",
</code><code class="uil"> "hour12": false </code><code
class="uil"> },

</code><code class="uil"> "short": {

</code><code class="uil"> "year": "numeric", </code>
<code class="uil"> "month": "2-digit", </code><code
class="uil"> "day": "2-digit"

</code><code class="uil"> }

</code><code class="uil"> },

</code><code class="uil"> "numberFormats": {

</code><code class="uil"> "currency": {

</code><code class="uil"> "style": "currency", </code>
<code class="uil"> "currency": "EUR", </code><code
class="uil"> "currencyDisplay": "symbol", </code><code
class="uil"> "minimumFractionDigits": 0, </code><code
class="uil"> "maximumFractionDigits": 2

</code><code class="uil"> },

</code><code class="uil"> "decimal": {

</code><code class="uil"> "style": "decimal", </code>
<code class="uil"> "useGrouping": false, </code><code
class="uil"> "minimumFractionDigits": 0, </code><code
class="uil"> "maximumFractionDigits": 2

</code><code class="uil"> },

</code><code class="uil"> "numeric": {

</code><code class="uil"> "style": "decimal", </code>
<code class="uil"> "useGrouping": false, </code><code
```

class="uil"> "minimumFractionDigits": 0, </code><code class="uil"> "maximumFractionDigits": 0

</code><code class="uil"> },

</code><code class="uil"> "percent": {

</code><code class="uil"> "style": "percent", </code> <code class="uil"> "useGrouping": false, </code><code class="uil"> "minimumFractionDigits": 0, </code><code class="uil"> "maximumFractionDigits": 0

</code><code class="uil"> }

</code><code class="uil"> },

</code> "messages": {

  "welcome": "Bienvenue: this message is localized in French", ...

{

<code class="uil"> "datetimeFormats": {

</code><code class="uil"> "long": {

</code><code class="uil"> "year": "numeric", </code> <code class="uil"> "month": "2-digit", </code><code class="uil"> "day": "2-digit", </code><code class="uil"> "hour": "2-digit", </code><code class="uil"> "minute": "2-digit", </code><code class="uil"> "second": "2-digit", </code><code class="uil"> "hour12": false </code><code class="uil"> },

</code><code class="uil"> "short": {

```
</code><code class="uil"> "year": "numeric", </code>
<code class="uil"> "month": "2-digit", </code><code
class="uil"> "day": "2-digit"

</code><code class="uil"> }

</code><code class="uil"> },

</code><code class="uil"> "numberFormats": {

</code><code class="uil"> "currency": {

</code><code class="uil"> "style": "currency", </code>
<code class="uil"> "currency": "EUR", </code><code
class="uil"> "currencyDisplay": "symbol", </code><code
class="uil"> "minimumFractionDigits": 0, </code><code
class="uil"> "maximumFractionDigits": 2

</code><code class="uil"> },

</code><code class="uil"> "decimal": {

</code><code class="uil"> "style": "decimal", </code>
<code class="uil"> "useGrouping": false, </code><code
class="uil"> "minimumFractionDigits": 0, </code><code
class="uil"> "maximumFractionDigits": 2

</code><code class="uil"> },

</code><code class="uil"> "numeric": {

</code><code class="uil"> "style": "decimal", </code>
<code class="uil"> "useGrouping": false, </code><code
class="uil"> "minimumFractionDigits": 0, </code><code
class="uil"> "maximumFractionDigits": 0

</code><code class="uil"> },
</code><code class="uil"> "percent": {

</code><code class="uil"> "style": "percent", </code>
<code class="uil"> "useGrouping": false, </code><code
class="uil"> "minimumFractionDigits": 0, </code><code
class="uil"> "maximumFractionDigits": 0

</code><code class="uil"> }

</code><code class="uil"> },

</code> "messages": {

   "welcome": "Bienvenido: this message is localized in
Spanish", ...

import { createI18n } from 'vue-i18n'



interface LocalesDataInterface {

<code class="uil"> datetimeFormats: any // <-- add this
line </code><code class="uil"> numberFormats: any // <--
add this line </code> messages: any

}



...

...

/**

* @name: getLocalesData
* @description: Helper to load the locale json files…

*/

const getLocalesData = (): LocalesDataInterface => {

   // use require.context to get all the json files…

   const files = (require as any).context('./locales', true, /[A-
Za-z0-9-_,\s]+\.json\ $/i)

   // create the instance that will hold the loaded data const
localeData: LocalesDataInterface = {

<code class="uil"> datetimeFormats: {}, // <-- add this line
</code><code class="uil"> numberFormats: {}, // <-- add
this line </code> messages: {}

   }

   // loop through all the files const keys: string[] =
files.keys() keys.forEach((key: string) => {

   // extract name without extension const matched =
key.match(/([A-Za-z0-9-_]+)\./i) if (matched &&
matched.length > 1) {

   const localeId = matched[1]

   // from each file, set the related messages property <code
class="uil"> localeData.datetimeFormats[localeId] =
files(key).datetimeFormats // <-- add t\ </code><code
class="uil">his line

</code><code class="uil">
localeData.numberFormats[localeId] =
files(key).numberFormats // <-- add this \ </code><code
class="uil">line
</code> localeData.messages[localeId] =
files(key).messages }

   })



   return localeData

}



...

...



// create our data dynamically by loading the JSON files…

const data: LocalesDataInterface = getLocalesData()

// create out vue-18n instance

export const i18n = createI18n({

   locale: 'it-IT',

   fallbackLocale: 'en-US',

   messages: data.messages,

<code class="uil"> datetimeFormats:
data.datetimeFormats, // <-- add this line </code><code
class="uil"> numberFormats: data.numberFormats // <--
add this line </code>})
```

```
<template>

   <div id="app">

<code class="uil"> <div class="long-date">{{ i18n.d((new
Date()), 'long') }}</div> <!-- add this li\ </code><code
class="uil">ne -->

</code> <h2>{{ i18n.t('welcome') }}</h2> ...

<code></code><code class="o"><</code><code
class="nt">style</code> <code class="nt">lang</code>
<code class="o">=</code><code class="s2">"scss"
</code><code class="o">></code> <code
class="nn">#app</code> <code class="p">{</code>
<code class="nt">font-family</code><code class="nd">:
</code> <code class="nt">Avenir</code><code
class="o">,</code> <code class="nt">Helvetica</code>
<code class="o">,</code> <code
class="nt">Arial</code><code class="o">,</code> <code
class="nt">sans-serif</code><code class="o">;</code>
<code class="nt">-webkit-font-smoothing</code><code
class="nd">:</code> <code
class="nt">antialiased</code><code class="o">;</code>
<code class="nt">-moz-osx-font-smoothing</code><code
class="nd">:</code> <code class="nt">grayscale</code>
<code class="o">;</code> <code
class="nt">padding</code><code class="nd">:</code>
<code class="nt">10px</code><code class="o">;
</code> <code class="nt">color</code><code
class="nd">:</code> <code class="nn">#2c3e50</code>
<code class="o">;</code> <code class="nt">h2</code>
<code class="p">{</code> <code
class="nt">margin</code><code class="nd">:</code>
<code class="nt">0</code><code class="o">;</code>
<code class="p">}</code>
```

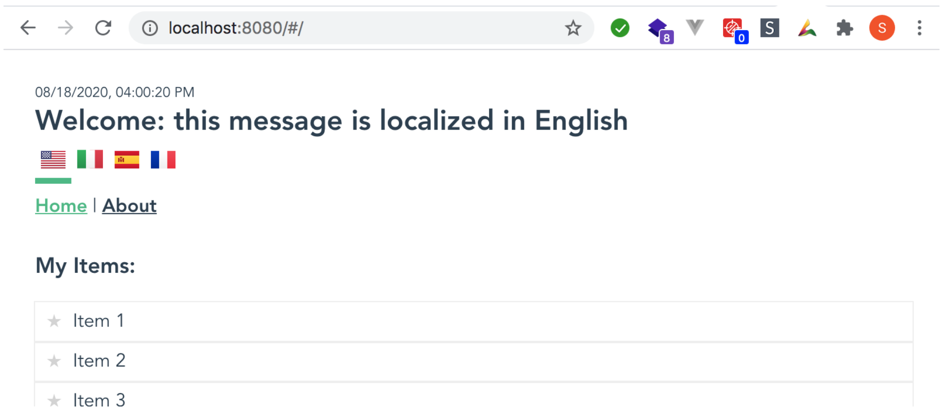
<code class="nc">.long-date</code> <code class="p">{</code> <code class="nt">font-size</code><code class="nd">:</code> <code class="nt">12px</code> <code class="o">;</code> <code class="p">}</code> <code class="p">}</code>

<code class="nn">#nav</code> <code class="p">{</code> <code class="nt">padding</code><code class="nd">:</code> <code class="nt">8px</code> <code class="nt">0</code><code class="o">;</code> <code class="nt">a</code> <code class="p">{</code> <code class="nt">font-weight</code><code class="nd">:</code> <code class="nt">bold</code><code class="o">;</code> <code class="nt">color</code> <code class="nd">:</code> <code class="nn">#2c3e50</code><code class="o">;</code> <code class="k">&</code><code class="nc">.router-link-exact-active</code> <code class="p">{</code> <code class="nt">color</code><code class="nd">:</code> <code class="nn">#42b983</code><code class="o">;</code> <code class="p">}</code> <code class="p">}</code> <code class="p">}</code> <code class="o"></</code><code class="nt">style</code><code class="o">></code>

Web Browser

The web page should have now refreshed, and the date and time should be displaying on the top.

Here is when clicking on the **US** icon:


localhost:8080/#/

08/18/2020, 04:00:20 PM

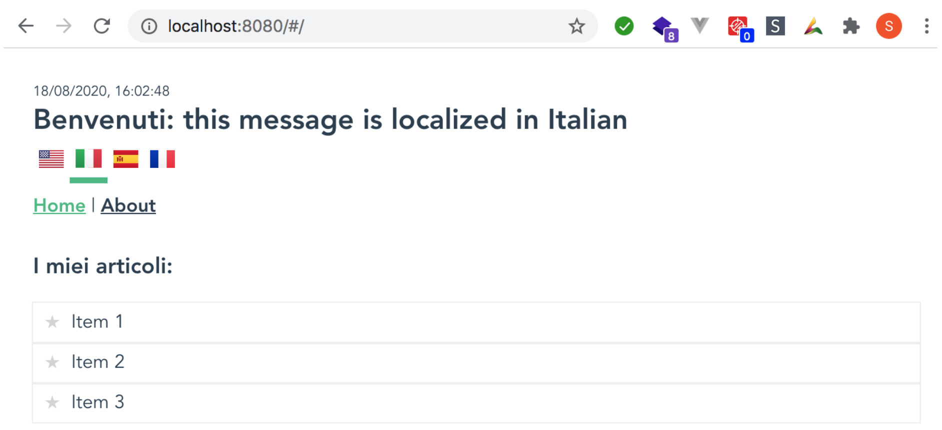
Welcome: this message is localized in English

Home | About

My Items:

★ Item 1
★ Item 2
★ Item 3

Here is when clicking on the icon for **Italy**:


localhost:8080/#/

18/08/2020, 16:02:48

Benvenuti: this message is localized in Italian

Home | About

I miei articoli:

★ Item 1
★ Item 2
★ Item 3

Chapter 13 Recap

What We Learned

- We learned how to apply additional settings to our **vue-i18n-next-plugin** implementation to display **numbers** and **dates** in the **locale** formats

Observations

- We did not write any unit tests against our **getLocalesData** helper function
- We did not write any unit tests against our static **locales JSON** files to validate their structure and content

Based on these observations, there are a few improvements that we can:

Improvements

- Optional: You can write unit tests to ensure the **getLocalesData** helper function loads the **JSON** files correctly and set the messages as expected. You could also write some unit tests against the **JSON** files directly to make sure they contain the correct data structure.

```
declare module '*.vue' {

   import { defineComponent } from 'vue'

   const component: ReturnType<typeof defineComponent>
export default component

}



declare interface process {

   env: {

<code class="udl"> VUE_APP_API_CLIENT: string // <--
remove this line </code><code class="udl">
VUE_APP_TOKEN_KEY: string // <-- remove this line </code>
<code class="uil"> VUE_APP_CONFIG: string // <-- add this
line </code> }

}

<code></code><code class="kr">import</code> <code
class="p">{</code> <code
class="nx">ItemsApiClientUrlsInterface</code> <code
class="p">}</code> <code class="nx">from</code>
<code class="s1">'@/models/api-client/items'</code>

<code class="cm">/**</code>

<code class="cm"> * @Name ConfigInterface</code>

<code class="cm"> * @description</code>

<code class="cm"> * </code>
```

<code class="cm"> */</code>

<code class="kr">export</code> <code class="kr">interface</code> <code class="nx">ConfigInterface</code> <code class="p">{</code> <code class="nx">global</code><code class="o">:</code> <code class="p">{</code> <code class="c1">// ... things that are not specific to a single app domain</code> <code class="p">}</code>

   <code class="nx">httpClient</code><code class="o">:</code> <code class="p">{</code> <code class="nx">tokenKey</code>: <code class="kt">string</code> <code class="p">}</code>

   <code class="nx">apiClient</code><code class="o">:</code> <code class="p">{</code> <code class="nx">type</code>: <code class="kt">string</code> <code class="p">}</code>

   <code class="nx">items</code><code class="o">:</code> <code class="p">{</code> <code class="nx">apiUrls</code>: <code class="kt">ItemsApiClientUrlsInterface</code> <code class="p">}</code>

<code class="p">}</code>

<code></code><code class="p">{</code>

   <code class="nt">"global"</code><code class="p">:</code> <code class="p">{</code> <code class="p">},</code>

   <code class="nt">"httpClient"</code><code class="p">:</code> <code class="p">{</code> <code class="nt">"tokenKey"</code><code class="p">:</code>

```
<code class="s2">"myapp-token"</code> <code
class="p">},</code>

   <code class="nt">"apiClient"</code><code class="p">:
</code> <code class="p">{</code> <code
class="nt">"type"</code><code class="p">:</code>
<code class="s2">"mock"</code> <code class="p">},
</code>

   <code class="nt">"items"</code><code class="p">:
</code> <code class="p">{</code> <code
class="nt">"apiUrls"</code><code class="p">:</code>
<code class="p">{</code> <code
class="nt">"fetchItems"</code><code class="p">:
</code> <code class="s2">"/static/data/items.json"
</code> <code class="p">}</code>

   <code class="p">}</code>

<code class="p">}</code>

<code></code><code class="p">{</code>

   <code class="nt">"global"</code><code class="p">:
</code> <code class="p">{</code> <code class="p">},
</code>

   <code class="nt">"httpClient"</code><code class="p">:
</code> <code class="p">{</code> <code
class="nt">"tokenKey"</code><code class="p">:</code>
<code class="s2">"myapp-token"</code> <code
class="p">},</code>

   <code class="nt">"apiClient"</code><code class="p">:
</code> <code class="p">{</code> <code
class="nt">"type"</code><code class="p">:</code>
```

```
<code class="s2">"live"</code> <code class="p">},
</code>

   <code class="nt">"items"</code><code class="p">:
</code> <code class="p">{</code> <code
class="nt">"apiUrls"</code><code class="p">:</code>
<code class="p">{</code> <code
class="nt">"fetchItems"</code><code class="p">:
</code> <code
class="s2">"/path/to/your/real/LOCAL/api/and-point"
</code> <code class="p">}</code>

   <code class="p">}</code>

<code class="p">}</code>

<code></code><code class="p">{</code>

   <code class="nt">"global"</code><code class="p">:
</code> <code class="p">{</code> <code class="p">},
</code>

   <code class="nt">"httpClient"</code><code class="p">:
</code> <code class="p">{</code> <code
class="nt">"tokenKey"</code><code class="p">:</code>
<code class="s2">"myapp-token"</code> <code
class="p">},</code>

   <code class="nt">"apiClient"</code><code class="p">:
</code> <code class="p">{</code> <code
class="nt">"type"</code><code class="p">:</code>
<code class="s2">"live"</code> <code class="p">},
</code>

   <code class="nt">"items"</code><code class="p">:
</code> <code class="p">{</code> <code
class="nt">"apiUrls"</code><code class="p">:</code>
```

```
<code class="p">{</code> <code
class="nt">"fetchItems"</code><code class="p">:
</code> <code
class="s2">"/path/to/your/real/BETA/api/and-point"</code>
<code class="p">}</code>

   <code class="p">}</code>

<code class="p">}</code>

<code></code><code class="p">{</code>

   <code class="nt">"global"</code><code class="p">:
</code> <code class="p">{</code> <code class="p">},
</code>

   <code class="nt">"httpClient"</code><code class="p">:
</code> <code class="p">{</code> <code
class="nt">"tokenKey"</code><code class="p">:</code>
<code class="s2">"myapp-token"</code> <code
class="p">},</code>

   <code class="nt">"apiClient"</code><code class="p">:
</code> <code class="p">{</code> <code
class="nt">"type"</code><code class="p">:</code>
<code class="s2">"live"</code> <code class="p">},
</code>

   <code class="nt">"items"</code><code class="p">:
</code> <code class="p">{</code> <code
class="nt">"apiUrls"</code><code class="p">:</code>
<code class="p">{</code> <code
class="nt">"fetchItems"</code><code class="p">:
</code> <code
class="s2">"/path/to/your/real/LIVE/api/and-point"</code>
<code class="p">}</code>
```

```
   <code class="p">}</code>

<code class="p">}</code>

{

   "compilerOptions": {



...



   "allowSyntheticDefaultImports": true,

<code class="uil"> "resolveJsonModule": true, /* this allows
to import .json file as if they were .\ </code><code
class="uil">ts files: using to load config files */

</code> "sourceMap": true,



...

<code></code><code class="kr">import</code> <code
class="p">{</code> <code
class="nx">ConfigInterface</code> <code class="p">}
</code> <code class="nx">from</code> <code
class="s1">'./Config.interface'</code>

<code class="c1">// individual environments configs:
</code> <code class="kr">import</code> <code
class="nx">configMock</code> <code
class="nx">from</code> <code class="s1">'./config-
files/mock.json'</code> <code class="kr">import</code>
<code class="nx">configLocal</code> <code
```

class="nx">from</code> <code class="s1">'./config-files/local.json'</code> <code class="kr">import</code> <code class="nx">configBeta</code> <code class="nx">from</code> <code class="s1">'./config-files/beta.json'</code> <code class="kr">import</code> <code class="nx">configLive</code> <code class="nx">from</code> <code class="s1">'./config-files/live.json'</code>

<code></code><code class="p">...</code>

<code class="c1">// return appropriate config based on env VUE_APP_CONFIG</code> <code class="kd">let</code> <code class="nx">env</code>: <code class="kt">string</code> <code class="o">=</code> <code class="s1">'mock'</code> <code class="cm">/* by default we return the mock configuration */</code>

<code class="c1">// if our env VUE_APP_CONFIG variable is set, get its value</code> <code class="k">if</code> <code class="p">(</code><code class="nx">process</code><code class="p">.</code> <code class="nx">env</code> <code class="o">&&</code> <code class="nx">process</code><code class="p">.</code><code class="nx">env</code><code class="p">.</code><code class="nx">VUE_APP_CONFIG</code><code class="p">)</code> <code class="p">{</code> <code class="nx">env</code> <code class="o">=</code> <code class="nx">process</code><code class="p">.</code><code class="nx">env</code><code class="p">.</code><code class="nx">VUE_APP_CONFIG</code> <code class="p">.</code><code class="nx">trim</code> <code class="p">().</code><code

class="nx">toLowerCase</code><code class="p">()
</code> <code class="p">}</code>

<code class="p">...</code>

<code></code><code class="p">...</code>

<code class="c1">// you can use a strategy pattern, or a javascript Map()</code> <code class="kr">export</code> <code class="kr">const</code> <code class="nx">configsMap</code>: <code class="kt">Map</code><code class="o"><</code><code class="kt">string</code><code class="p">,</code> <code class="nx">ConfigInterface</code><code class="o">></code> <code class="o">=</code> <code class="k">new</code> <code class="nx">Map</code> <code class="o"><</code><code class="kt">string</code><code class="p">,</code> <code class="nx">ConfigInterf</code><code class="o">\
</code> <code class="nx">ace</code><code class="o">></code><code class="p">([</code> <code class="p">[</code><code class="s1">'mock'</code> <code class="p">,</code> <code class="nx">configMock</code><code class="p">],
</code> <code class="p">[</code><code class="s1">'local'</code><code class="p">,</code> <code class="nx">configLocal</code><code class="p">],
</code> <code class="p">[</code><code class="s1">'beta'</code><code class="p">,</code> <code class="nx">configBeta</code><code class="p">],
</code> <code class="p">[</code><code class="s1">'live'</code><code class="p">,</code>

<code class="nx">configLive</code><code class="p">]
</code> <code class="p">])</code>

<code class="k">if</code> <code class="p">(</code>
<code class="o">!</code><code
class="nx">configsMap</code><code class="p">.</code>
<code class="nx">has</code><code class="p">(</code>
<code class="nx">env</code><code class="p">))
</code> <code class="p">{</code> <code
class="k">throw</code> <code class="nb">Error</code>
<code class="p">(</code><code class="sb">`Could not
find config for VUE_APP_CONFIG key "</code><code
class="si">${</code> <code class="nx">env</code>
<code class="si">}</code><code class="sb">"`</code>
<code class="p">)</code> <code class="p">}</code>

<code class="kr">export</code> <code
class="kr">const</code> <code
class="nx">config</code>: <code
class="kt">ConfigInterface</code> <code class="o">=
</code> <code class="nx">configsMap</code><code
class="p">.</code><code class="nx">get</code><code
class="p">(</code><code class="nx">env</code><code
class="p">)</code> <code class="kr">as</code> <code
class="nx">ConfigInterface</code>

<code></code><code class="kr">import</code> <code
class="p">{</code> <code class="nx">expect</code>
<code class="p">}</code> <code
class="nx">from</code> <code class="s1">'chai'</code>
<code class="kr">import</code> <code class="p">
{</code> <code class="nx">configsMap</code> <code

```
class="p">}</code> <code class="nx">from</code>
<code class="s1">'@/config'</code>

<code class="nx">describe</code><code class="p">
(</code><code class="s1">'configsMap'</code><code
class="p">,</code> <code class="p">()</code> <code
class="o">=></code> <code class="p">{</code>

   <code class="nx">it</code><code class="p">(</code>
<code class="s1">'instance should have "mock"
key'</code><code class="p">,</code> <code class="p">
()</code> <code class="o">=></code> <code class="p">
{</code> <code class="nx">expect</code><code
class="p">(</code><code class="nx">configsMap</code>
<code class="p">.</code><code class="nx">has</code>
<code class="p">(</code><code
class="s1">'mock'</code><code class="p">)).</code>
<code class="nx">to</code><code class="p">.</code>
<code class="nx">equal</code><code class="p">
(</code><code class="kc">true</code><code
class="p">)</code> <code class="p">})</code>



   <code class="nx">it</code><code class="p">(</code>
<code class="s1">'instance should have "local"
key'</code><code class="p">,</code> <code class="p">
()</code> <code class="o">=></code> <code class="p">
{</code> <code class="nx">expect</code><code
class="p">(</code><code class="nx">configsMap</code>
<code class="p">.</code><code class="nx">has</code>
<code class="p">(</code><code
class="s1">'local'</code><code class="p">)).</code>
<code class="nx">to</code><code class="p">.</code>
<code class="nx">equal</code><code class="p">
(</code><code class="kc">true</code><code
class="p">)</code> <code class="p">})</code>



  <code class="nx">it</code><code class="p">(</code>
<code class="s1">'instance should have "beta"
key'</code><code class="p">,</code> <code class="p">
()</code> <code class="o">=></code> <code class="p">
{</code> <code class="nx">expect</code><code
class="p">(</code><code class="nx">configsMap</code>
<code class="p">.</code><code class="nx">has</code>
<code class="p">(</code><code
class="s1">'beta'</code><code class="p">)).</code>
<code class="nx">to</code><code class="p">.</code>
<code class="nx">equal</code><code class="p">
(</code><code class="kc">true</code><code
class="p">)</code> <code class="p">})</code>



  <code class="nx">it</code><code class="p">(</code>
<code class="s1">'instance should have "live" key'</code>
<code class="p">,</code> <code class="p">()</code>
<code class="o">=></code> <code class="p">{</code>
<code class="nx">expect</code><code class="p">
(</code><code class="nx">configsMap</code><code
class="p">.</code><code class="nx">has</code><code
class="p">(</code><code class="s1">'live'</code><code
class="p">)).</code><code class="nx">to</code><code
class="p">.</code><code class="nx">equal</code>
<code class="p">(</code><code class="kc">true</code>
<code class="p">)</code> <code class="p">})</code>

<code class="p">})</code>
```

```
<code></code><code class="kr">import</code> <code
class="p">{</code> <code class="nx">expect</code>
<code class="p">}</code> <code
class="nx">from</code> <code class="s1">'chai'</code>
<code class="kr">import</code> <code class="p">
{</code> <code class="nx">config</code> <code
class="p">}</code> <code class="nx">from</code>
<code class="s1">'@/config'</code>

<code class="nx">describe</code><code class="p">
(</code><code class="s1">'config'</code><code
class="p">,</code> <code class="p">()</code> <code
class="o">=></code> <code class="p">{</code> <code
class="nx">it</code><code class="p">(</code><code
class="s1">'instance should have "global" section'</code>
<code class="p">,</code> <code class="p">()</code>
<code class="o">=></code> <code class="p">{</code>
<code class="nx">expect</code><code class="p">
(</code><code class="nx">config</code><code
class="p">).</code><code class="nx">to</code><code
class="p">.</code><code class="nx">have</code><code
class="p">.</code><code class="nx">own</code><code
class="p">.</code><code class="nx">property</code>
<code class="p">(</code><code
class="s1">'global'</code><code class="p">)</code>
<code class="p">})</code>



   <code class="nx">it</code><code class="p">(</code>
<code class="s1">'instance should have "httpClient"
section'</code><code class="p">,</code> <code
class="p">()</code> <code class="o">=></code> <code
class="p">{</code> <code class="nx">expect</code>
<code class="p">(</code><code
class="nx">config</code><code class="p">).</code>
```

<code class="nx">to</code><code class="p">.</code> <code class="nx">have</code><code class="p">. </code><code class="nx">own</code><code class="p">. </code><code class="nx">property</code><code class="p">(</code><code class="s1">'httpClient'</code> <code class="p">)</code> <code class="p">})</code>

<code class="nx">it</code><code class="p">(</code> <code class="s1">'instance should have "items" section'</code><code class="p">,</code> <code class="p">()</code> <code class="o">=></code> <code class="p">{</code> <code class="nx">expect</code> <code class="p">(</code><code class="nx">config</code><code class="p">).</code> <code class="nx">to</code><code class="p">.</code> <code class="nx">have</code><code class="p">. </code><code class="nx">own</code><code class="p">. </code><code class="nx">property</code><code class="p">(</code><code class="s1">'items'</code> <code class="p">)</code> <code class="p">})</code>

<code class="nx">describe</code><code class="p"> (</code><code class="s1">'global'</code><code class="p">,</code> <code class="p">()</code> <code class="o">=></code> <code class="p">{</code> <code class="kr">const</code> <code class="nx">section</code> <code class="o">=</code> <code class="nx">config</code><code class="p">. </code><code class="nx">global</code> <code class="c1">// tests against the global section eventually go here</code> <code class="p">})</code>

```
   <code class="nx">describe</code><code class="p">
(</code><code class="s1">'httpClient'</code><code
class="p">,</code> <code class="p">()</code> <code
class="o">=></code> <code class="p">{</code> <code
class="kr">const</code> <code
class="nx">section</code> <code class="o">=</code>
<code class="nx">config</code><code class="p">.
</code><code class="nx">httpClient</code> <code
class="nx">it</code><code class="p">(</code><code
class="s1">'section should have "tokenKey"
property'</code><code class="p">,</code> <code
class="p">()</code> <code class="o">=></code> <code
class="p">{</code> <code class="nx">expect</code>
<code class="p">(</code><code
class="nx">section</code><code class="p">).</code>
<code class="nx">to</code><code class="p">.</code>
<code class="nx">have</code><code class="p">.
</code><code class="nx">own</code><code class="p">.
</code><code class="nx">property</code><code
class="p">(</code><code class="s1">'tokenKey'</code>
<code class="p">)</code> <code class="p">})</code>

   <code class="p">})</code>



<code class="p">...</code>

<code></code><code class="p">...</code>



<code class="nx">describe</code><code class="p">
(</code><code class="s1">'apiClient'</code><code
class="p">,</code> <code class="p">()</code> <code
```

class="o">=></code> <code class="p">{</code> <code class="kr">const</code> <code class="nx">section</code> <code class="o">=</code> <code class="nx">config</code><code class="p">.</code><code class="nx">apiClient</code> <code class="nx">it</code><code class="p">(</code><code class="s1">'section should have "type" property'</code> <code class="p">,</code> <code class="p">()</code> <code class="o">=></code> <code class="p">{</code> <code class="nx">expect</code><code class="p">(</code><code class="nx">section</code><code class="p">).</code><code class="nx">to</code><code class="p">.</code><code class="nx">have</code><code class="p">.</code><code class="nx">own</code><code class="p">.</code><code class="nx">property</code> <code class="p">(</code><code class="s1">'type'</code><code class="p">)</code> <code class="p">})</code>

<code class="p">})</code>

<code class="nx">describe</code><code class="p">(</code><code class="s1">'items'</code><code class="p">,</code> <code class="p">()</code> <code class="o">=></code> <code class="p">{</code> <code class="kr">const</code> <code class="nx">section</code> <code class="o">=</code> <code class="nx">config</code><code class="p">.</code><code class="nx">items</code>

<code class="nx">it</code><code class="p">(</code> <code class="s1">'section should have "apiUrls" property'</code><code class="p">,</code> <code class="p">()</code> <code class="o">=></code> <code

class="p">{</code> <code class="nx">expect</code>
<code class="p">(</code><code
class="nx">section</code><code class="p">).</code>
<code class="nx">to</code><code class="p">.</code>
<code class="nx">have</code><code class="p">.
</code><code class="nx">own</code><code class="p">.
</code><code class="nx">property</code><code
class="p">(</code><code class="s1">'apiUrls'</code>
<code class="p">)</code> <code class="p">})</code>

<code class="nx">describe</code><code class="p">
(</code><code class="s1">'apiUrls'</code><code
class="p">,</code> <code class="p">()</code> <code
class="o">=></code> <code class="p">{</code> <code
class="kr">const</code> <code
class="nx">apiUrls</code> <code class="o">=</code>
<code class="nx">section</code><code class="p">.
</code><code class="nx">apiUrls</code>

<code class="nx">it</code><code class="p">(</code>
<code class="s1">'section should have "fetchItems"
property'</code><code class="p">,</code> <code
class="p">()</code> <code class="o">=></code> <code
class="p">{</code> <code class="nx">expect</code>
<code class="p">(</code><code
class="nx">apiUrls</code><code class="p">).</code>
<code class="nx">to</code><code class="p">.</code>
<code class="nx">have</code><code class="p">.
</code><code class="nx">own</code><code class="p">.
</code><code class="nx">property</code><code
class="p">(</code><code
class="s1">'fetchItems'</code><code class="p">)
</code> <code class="c1">// verify that fetchItems url is a
string and has a reasonable length</code> <code

```
class="nx">expect</code><code class="p">(</code>
<code class="nx">apiUrls</code><code class="p">.
</code><code class="nx">fetchItems</code><code
class="p">).</code><code class="nx">to</code><code
class="p">.</code><code class="nx">be</code><code
class="p">.</code><code class="nx">a</code><code
class="p">(</code><code class="s1">'string'</code>
<code class="p">)</code> <code
class="nx">expect</code><code class="p">(</code>
<code class="nx">apiUrls</code><code class="p">.
</code><code class="nx">fetchItems</code><code
class="p">).</code><code class="nx">to</code><code
class="p">.</code><code class="nx">have</code><code
class="p">.</code><code class="nx">length</code>
<code class="p">.</code><code
class="nx">greaterThan</code><code class="p">
(</code><code class="mi">10</code><code class="p">)
</code> <code class="p">})</code>

   <code class="p">})</code>

   <code class="p">})</code>

<code class="p">})</code>
```

...

```
   "scripts": {

   "serve": "cross-env VUE_APP_API_CLIENT=mock vue-cli-
service serve --mode develop\ ment",

   "build": "cross-env VUE_APP_API_CLIENT=live vue-cli-
service build --mode product\ ion",
```

"build-mock": "cross-env VUE_APP_API_CLIENT=mock vue-cli-service build --mode pr\ oduction",

<code class="udl"> "test:unit": "vue-cli-service test:unit", <!-- remove this line --> </code><code class="uil"> <!-- add the following 4 lines --> </code><code class="uil"> "test-mock": "cross-env VUE_APP_CONFIG=mock vue-cli-service test:unit", </code><code class="uil"> "test-local": "cross-env VUE_APP_CONFIG=local vue-cli-service test:unit", </code><code class="uil"> "test-beta": "cross-env VUE_APP_CONFIG=beta vue-cli-service test:unit", </code> <code class="uil"> "test-live": "cross-env VUE_APP_CONFIG=live vue-cli-service test:unit", </code>

...

<code></code>DONE Compiled successfully in 3172ms

...

MOCHA Testing...

configsMap

✓ instance should have "mock" key

✓ instance should have "local" key

✓ instance should have "beta" key

✓ instance should have "live" key

```
config

✓ instance should have "global" section

✓ instance should have "items" section

items

✓ section should have "apiUrls" property

apiUrls

✓ section should have "fetchItems" property


HttpClient.get

✓ should succeed and return data (102ms)


HttpClient.get

✓ should reject and return 400 (103ms)


Item.component.vue

✓ renders an Item correctly

✓ has expected css class when selected is false ✓ has
selected css class when selected is true


13 passing (231ms)
```

MOCHA Tests completed successfully

<code></code><code class="err">...</code>

<code class="s2">"scripts"</code><code class="err">:</code> <code class="p">{</code> <code class="nt">"serve"</code><code class="p">:</code> <code class="s2">"cross-env VUE_APP_CONFIG=mock vue-cli-service serve --mode development\</code> <code class="s2">"</code><code class="p">,</code> <code class="nt">"build"</code><code class="p">:</code> <code class="s2">"cross-env VUE_APP_CONFIG=live vue-cli-service build --mode production"</code><code class="p">,</code> <code class="nt">"build-mock"</code><code class="p">:</code> <code class="s2">"cross-env VUE_APP_CONFIG=mock vue-cli-service build --mode produc\</code> <code class="s2">tion"</code><code class="p">,</code>

<code class="err">...</code>

...

<code class="uil">import { config } from '@/config' // <-- add this line </code>...

```
export class HttpClientModel implements HttpClientInterface
{

  private getToken(): string {
```

```
<code class="udl"> // begin: remove code block

</code><code class="udl"> const TOKEN_KEY =

</code><code class="udl"> process.env &&
process.env.VUE_APP_TOKEN_KEY

</code><code class="udl"> ?
process.env.VUE_APP_TOKEN_KEY

</code><code class="udl"> : 'myapp-token'

</code><code class="udl"> // end: remove code block

</code><code class="uil"> const TOKEN_KEY =
config.httpClient.tokenKey || 'myapp-token' // <-- add this li\
</code><code class="uil">ne

</code> const token = localStorage.getItem(TOKEN_KEY) ||
''

  return token

  }



...

...

<code class="uil">import { config } from '@/config' // <--
add this line </code>...

<code class="udl">// begin: remove code block

</code><code class="udl">let env: string = 'mock'
```

```
</code><code class="udl">if (process.env &&
process.env.VUE_APP_API_CLIENT) {

</code><code class="udl"> env =
process.env.VUE_APP_API_CLIENT.trim() </code><code
class="udl">}

</code><code class="udl">// end: remove code block

</code>// return either the live or the mock client

<code class="udl">// begin: remove code block

</code><code class="udl">let apiClient: ApiClientInterface
</code><code class="udl">if (env === 'live') {

</code><code class="udl"> apiClient = apiLiveClient

</code><code class="udl">} else {

</code><code class="udl"> apiClient = apiMockClient

</code><code class="udl">}

</code><code class="udl">// end: remove code block

</code>if (config.apiClient.type === 'live') {

  apiClient = apiLiveClient

} else if (config.apiClient.type === 'mock') {

  apiClient = apiMockClient

} else {

  throw Error('Invalid or undefined config.apiClient.type') }
```

```
export default apiClient

...

<code class="uil">import { config } from '@/config' // <--
add this line </code>// urls for this API client

<code class="udl">// begin: remove code block

</code><code class="udl">const urls:
ItemsApiClientUrlsInterface = {

</code><code class="udl"> fetchItems:
'/path/to/your/real/api/and-point'

</code><code class="udl">}

</code><code class="udl">// end: remove code block

</code><code class="uil">const urls:
ItemsApiClientUrlsInterface = config.items.apiUrls // <-- add
this line </code>// instantiate the ItemsApiClient pointing at
the url that returns the live data fro\ m an API server

const itemsApiClient: ItemsApiClientInterface = new
ItemsApiClientModel(urls) // export our instance

export default itemsApiClient

...

<code class="uil">import { config } from '@/config' // <--
add this line </code>// urls for this API client

<code class="udl">// begin: remove code block
```

</code><code class="udl">const urls: ItemsApiClientUrlsInterface = {

</code><code class="udl"> fetchItems: '/static/data/items.json'

</code><code class="udl">}

</code><code class="udl">// end: remove code block

</code>const urls: ItemsApiClientUrlsInterface = config.items.apiUrls // <-- add this line

// instantiate the ItemsApiClient pointing at the url that returns static json mock \ data

const itemsApiClient: ItemsApiClientInterface = new ItemsApiClientModel(urls) // export our instance

export default itemsApiClient

*IMPORTANT: At this point, thanks to the new way of driving things through the **config**, the code in both files **src/api-client/live/items/index.ts** and **src/api-client/mock/items/index.ts** is identical. In later chapters we will simplify and reduce the amount of code we initially created to serve either **mock** or **live** data. But for now, we'll keep the duplicated code to avoid making this chapter too long.*

Now make sure you run all the unit tests again, then serve the app again to make sure all compiles and works as before.

Chapter 14 Recap

What We Learned

- We learned how to use static **JSON** files to have multiple configuration settings, one for each environment
- How to dynamically return the appropriate config file based on the new environment variable **VUE_APP_CONFIG**
- How to add option **resolveJsonModule** to the the **TypeScript tsconfig.json** file, section **compilerOptions** to allow importing static **JSON** files through import statement
- How to write unit tests against our configuration

Observations

- For now our configuration is pretty small, but might grow larger as the application itself grows and we need to add more configurable options.

Improvements

- Going forward we’ll be expanding the configuration as we keep growing our application components and logic.

Chapter 15 - Using CSS/SASS/SCSS Libraries

So far we have used very simple **SCSS** included within the **<style>** section of our **.vue** files. There is nothing wrong with this approach, but if we want to be able to apply a uniform style and branding throughout our entire application, it makes more sense to add the SCSS through the plugins.

This has a few advantages. The most important advantage is the ability to easily witch between CSS frameworks in the future, should we need to. Switching from on CSS framework to another it is not necessarily easy as it also depends on the html structure and the names of the CSS classes applied to your elements. However, in this book we’ll use the plugin approach to at least make it easy in your **Vue** app to switch between SCSS libraries (or use multiple libraries, like in the case of the **flag-icons** SCSS).

We start by removing all the SCSS code currently within **<style>** tags, and create our own custom SCSS library called **MyAppScss** (later in other projects you are free to use any SCSS library you like).

Since there is quite a bit of SCSS code here, it us not practical to try explaining all the changes with examples in this chapter. But you can download the latest version of the SCSS from the book companion code on GitHub[1].

SCSS Library

Download the content of the **src/assets/scss** directory from the GitHub[2] repo and put it under your project at the same path.

You can see that the **scss** directory contains two folders:

- **flag-icons** (here I moved the previous **flag-icons.scss** file and renamed it **index.scss** to keep things organized in a consistent way)
- **myapp-scss** (here I put all the SCSS I extracted from each component **.vue** files, which has also been organized in a certain way plus updated to support different themes)

Remove all the <style> code from the components

Remove all the <style> sections and their content from all the components where we have it. For example, **Loader.component.vue, ItemsList.component.vuew, Ite.component.vue**, etc.

Flag-Icons Plugin Code update

Update the code within **src/plugins/flag-icons/index.ts** to refer to the new location for the **flag-icons scss**:

```
export const FlagIconsScss = {
  install() {
markua-begin-delete
    require('../../assets/scss/flag-icons.scss') // <-- remove this line
markua-begin-insert
    require('../../assets/scss/flag-icons/index.scss') // <-- add this line
  }
}
```

MyAppScss Plugin

Create the directory **src/plugins/myapp-scss** and inside here create a file called **index.ts** with the following code:

```
export const MyAppScss = {
  install() {
    require('../../assets/scss/myapp-scss/index.scss')
  }
}
```

All we are doing here is use the install method of the plugin to include the main **index.scss** from our SCSS library, which is located within the **src/assets/scss/myapp-scss** folder.

main.ts Updates

Update the **src/main.ts** to import the new **MyAppScss** plugin and tell Vue to "use" it:

```
import { createApp } from 'vue'
import App from './App.vue'
import router from './router'
import { store } from './store'
import { i18n } from '@/plugins/vue-i18n-next-plugin'
markua-begin-insert
import { MyAppScss } from '@/plugins/myapp-scss' // <-- add this line
import { FlagIconsScss } from '@/plugins/flags-icons/'

createApp(App)
  .use(store)
  .use(router)
  .use(i18n)
markua-begin-insert
  .use(MyAppScss) // <-- add this line
  .use(FlagIconsScss)
  .mount('#app')
```

public/index.html Updates

We could do this in a different way, but since we'll be using the font **Open Sans** and **Material Icons** let's just reference them from **fonts.googleapis.com**:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
    <link rel="icon" href="<%= BASE_URL %>favicon.ico">
    <title><%= htmlWebpackPlugin.options.title %></title>
markua-begin-insert
    <link rel="stylesheet" href="https://fonts.googleapis.com/css?
family=Open+Sans:3\
00,400,600,700|Material+Icons&lang=en&display=swap"/> <!-- add this line -->
...
```

Now, if you run the app again it should all still work and use the centralized SCSS thanks to the **MyAppScss** plugin we created.

ThemeSelector component

Another thing we can do for fun, is to create a component called **ThemeSelector** that will allow us to switch between different themes.

Create the directory **src/components/theme-selector** and here add a new file called **ThemeSelector.component.vue** with the following **<template>** code:

```
<template>
  <div class="theme-selector">
    <div class="theme-radio-group">
      <label
        role="radio"
        v-for="(theme, index) in themes"
        :key="index"
        :class="`theme-radio ${ theme.name } ${theme.selected ? 'selected' :
''}`.tr\
im()"
      >
        <i class="material-icons">color_lens</i>
        <input
          type="radio"
          class="icon-button"
          name="locale"
          :value="theme.selected"
          v-model="theme.selected"
          @click="onThemeClicked(theme)"
        />
      </label>
    </div>
  </div>
</template>
...
```

Here we are creating a group of **radio** buttons using a **v-for** binding and setting their CSS **class** attribute dynamically adding the theme **name** and the **selected** class for the one that is currently selected.

For the **<script>** section, we define an interface called **ThemeInfoInterface** which defines the properties for each of our themes. These are

- **name** (the friendly name of our theme)
- **selected** (to track which one is currently selected)
- **bodyCss** (the name of the **body css class** we need to apply the the <body> element in order to change theme)

We also define an array called **themes** that hold information for our 3 different themes (**light, dark, navy**):

```
...

<script lang="ts">
  import { defineComponent, reactive, onMounted, ref } from 'vue'

  interface ThemeInfoInterface {
    selected: boolean
    name: string
    bodyCss: string
  }

  const themes: ThemeInfoInterface[] = reactive([{
    selected: false,
    name: 'light',
    bodyCss: 'default'
  }, {
    selected: false,
    name: 'dark',
    bodyCss: 'dark-theme'
  }, {
    selected: false,
    name: 'navy',
    bodyCss: 'navy-theme'
  }])

...
```

Finally the component declaration. Here we exposes the **themes** array and the handler **onThemeClicked** to the template. Within the **click handler** we set the **document.body** class to the clicked theme **bodyCss** value, and also ensure that only the clicked theme is set to **selected**:

```
...

  export default defineComponent({
    components: {
    },
    setup(props, { emit }) {

      const onThemeClicked = (themeClicked: ThemeInfoInterface) => {
        document.body.className = ''
        document.body.classList.add(themeClicked.bodyCss)
        // select only the clicked theme
        themes.forEach(theme => {
          theme.selected = theme.name === themeClicked.name
        })
      }

      onMounted(() => {
        const defaultTheme = themes.find(theme => theme.name === 'dark')
        if (defaultTheme) {
          onThemeClicked(defaultTheme)
        }
      })

      return {
        themes,
        onThemeClicked
      }
    }
  })
</script>
```

Note how we also use **onMounted** to preselect a **default theme** (**dark**).

App.vue updates

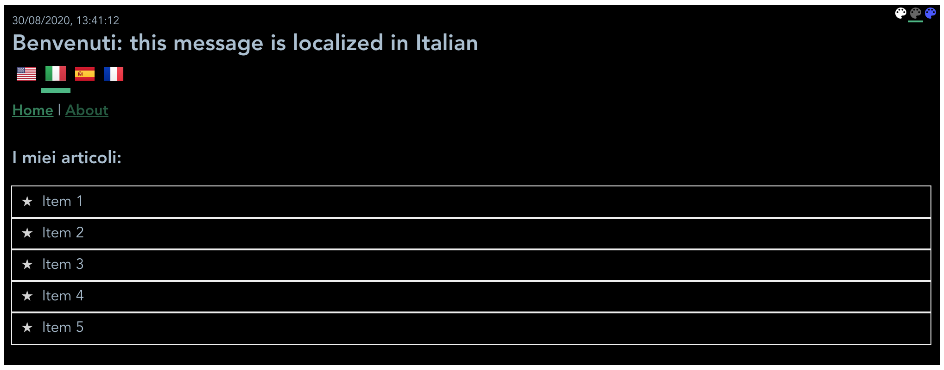
Now let’s import and use the **ThemeComponent**. Modify the **App.vue** file **<template>** section as follows:

```
<template>
  <div id="app">
    <div class="long-date">{{ i18n.d(new Date(), 'long') }}</div>
    <h2>{{ i18n.t('welcome') }}</h2>
    <LocaleSelector
      :availableLocales="availableLocales"
      @clicked="onLocaleClicked"
    />
markua-begin-insert
    <ThemeSelector /> <!-- add this line -->
    <div id="nav" class="nav">
...
```

And the **<script>** section as follows:

```
...

<script lang="ts">
  ...
markua-begin-insert
  // add this line:
  import ThemeSelector from '@/components/theme-
selector/ThemeSelector.component.vue'

  export default defineComponent({
    name: 'App',
    components: {
      LocaleSelector,
markua-begin-insert
      ThemeSelector // <!-- add this line
    },
    setup() {

...
```

If you now run the app and refresh your browser, it should default to the **dark** theme:



30/08/2020, 13:41:12

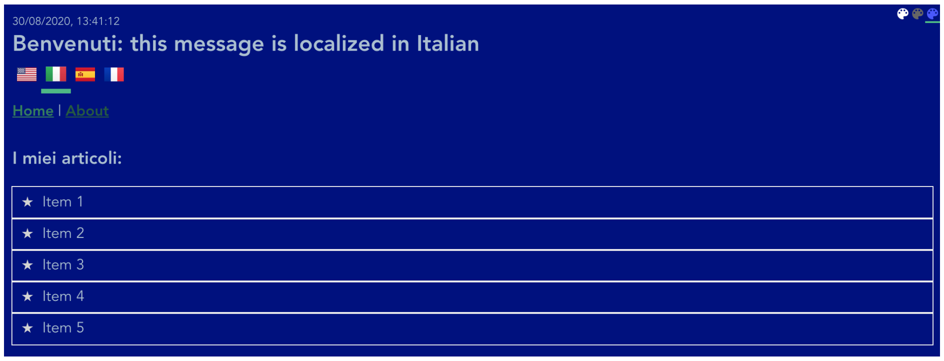
Benvenuti: this message is localized in Italian

Home | About

I miei articoli:

★ Item 1
★ Item 2
★ Item 3
★ Item 4
★ Item 5

You can then click on the top-right icons to switch between themes. For example, here is after clicking on the icon for the **blue** theme:


30/08/2020, 13:41:12

Benvenuti: this message is localized in Italian

Home | About

I miei articoli:

★ Item 1
★ Item 2
★ Item 3
★ Item 4
★ Item 5

Chapter 15 Recap

What We Learned

- We learned on how to organize SCSS libraries under our **src/assets/scss** folder
- We learned how to include those libraries by creating a plugin
- We learned hoe to create a component to switch between themes

Observations

- We did not write unit tests
- We did not show how to add a third-party SCSS library

Based on these observations, there are a few improvements that you could do on your own:

Improvements

- Write unit tests against the ThemeSelector component
- Try adding a third-party SCSS library and consume it with a custom plugin like we did for **MyAppCss**
- Drive both Locale and Theme from configuration **(see branch locales-themes-state in GitHub**[3] **repo)**
- Use Vuex state to handle the Theme selection **(see branch locales-themes-state in GitHub repo)**
- Use localStorage to remember the user preference for both Locale and Theme selected **(see branch locales-themes-state in GitHub repo)**

```
<code></code><code class="kr">export</code> <code
class="kr">interface</code> <code
class="nx">ThemeInfoInterface</code> <code class="p">
{</code>

   <code class="nx">id</code>: <code
class="kt">string</code>

   <code class="nx">name</code>: <code
class="kt">string</code>

   <code class="nx">icon</code>: <code
class="kt">string</code>

   <code class="nx">selected</code>: <code
class="kt">boolean</code>

<code class="p">}</code>

<code></code><code class="kr">export</code> <code
class="o">*</code> <code class="nx">from</code>
<code class="s1">'./ThemeInfo.interface'</code>

<code></code><code class="kr">export</code> <code
class="o">*</code> <code class="nx">from</code>
<code class="s1">'./LocalStorageKeys'</code>

<code class="kr">export</code> <code class="o">*
</code> <code class="nx">from</code> <code
class="s1">'./Mutations.model'</code>

<code class="kr">export</code> <code class="o">*
</code> <code class="nx">from</code> <code
class="s1">'./Actions.model'</code>
```

```
<code class="kr">export</code> <code class="o">*
</code> <code class="nx">from</code> <code
class="s1">'./State.interface'</code>

<code class="kr">export</code> <code class="o">*
</code> <code class="nx">from</code> <code
class="s1">'./Store.model'</code>

<code></code><code class="kr">export</code> <code
class="nx">namespace</code> <code
class="nx">LocalStorageKeys</code> <code class="p">
{</code>

   <code class="kr">export</code> <code
class="kr">const</code> <code
class="nx">theme</code> <code class="o">=</code>
<code class="s1">'theme'</code>

<code class="p">}</code>

<code></code><code class="kr">import</code> <code
class="p">{</code> <code
class="nx">ThemeInfoInterface</code> <code
class="p">}</code> <code class="nx">from</code>
<code class="s1">'../models/ThemeInfo.interface'</code>



<code class="cm">/**</code>

<code class="cm"> * @name StateInterface</code>

<code class="cm"> * @description</code>

<code class="cm"> * Interface for the Themes
state</code>
```

<code class="cm"> */</code>

<code class="kr">export</code> <code class="kr">interface</code> <code class="nx">StateInterface</code> <code class="p">{</code>

   <code class="nx">themes</code>: <code class="kt">ThemeInfoInterface</code><code class="p">[]</code>

<code class="p">}</code>

<code></code><code class="kr">import</code> <code class="p">{</code> <code class="nx">ThemeInfoInterface</code> <code class="p">}</code> <code class="nx">from</code> <code class="s1">'../models'</code>

<code class="kr">import</code> <code class="p">{</code> <code class="nx">StateInterface</code> <code class="p">}</code> <code class="nx">from</code> <code class="s1">'./State.interface'</code>

<code class="cm">/**</code>

<code class="cm"> * @name MutationsInterface</code>

<code class="cm"> */</code>

<code class="kr">export</code> <code class="kr">interface</code> <code class="nx">MutationsInterface</code> <code class="p">{</code>

```
   <code class="nx">loadThemes</code><code
class="p">(</code><code class="nx">themes</code>:
<code class="kt">ThemeInfoInterface</code><code
class="p">[])</code><code class="o">:</code> <code
class="k">void</code>

   <code class="nx">selectTheme</code><code
class="p">(</code><code class="nx">themeId</code>:
<code class="kt">string</code><code class="p">)
</code><code class="o">:</code> <code
class="k">void</code>

<code class="p">}</code>



<code class="cm">/**</code>

<code class="cm"> * @name MutationsModel</code>

<code class="cm"> */</code>

<code class="kr">export</code> <code
class="kr">class</code> <code
class="nx">MutationsModel</code> <code
class="kr">implements</code> <code
class="nx">MutationsInterface</code> <code class="p">
{</code>

   <code class="kr">private</code> <code
class="nx">state</code><code class="o">!:</code>
<code class="nx">StateInterface</code>



   <code class="kr">constructor</code><code class="p">
(</code><code class="nx">state</code>: <code
class="kt">StateInterface</code><code class="p">)
</code> <code class="p">{</code>

   <code class="k">this</code><code class="p">.</code>
<code class="nx">state</code> <code class="o">=
</code> <code class="nx">state</code>

   <code class="p">}</code>



   <code class="nx">loadThemes</code><code
class="p">(</code><code class="nx">themes</code>:
<code class="kt">ThemeInfoInterface</code><code
class="p">[])</code> <code class="p">{</code>

   <code class="k">this</code><code class="p">.</code>
<code class="nx">state</code><code class="p">.
</code><code class="nx">themes</code> <code
class="o">=</code> <code class="nx">themes</code>

   <code class="p">}</code>



   <code class="nx">selectTheme</code><code
class="p">(</code><code class="nx">themeId</code>:
<code class="kt">string</code><code class="p">)
</code> <code class="p">{</code>

   <code class="k">this</code><code class="p">.</code>
<code class="nx">state</code><code class="p">.
</code><code class="nx">themes</code><code
class="p">.</code><code class="nx">forEach</code>
<code class="p">((</code><code
class="nx">theme</code><code class="p">)</code>
<code class="o">=></code> <code class="p">{</code>
```

```
   <code class="nx">theme</code><code class="p">.
</code><code class="nx">selected</code> <code
class="o">=</code> <code class="p">(</code><code
class="nx">theme</code><code class="p">.</code>
<code class="nx">id</code> <code class="o">===
</code> <code class="nx">themeId</code><code
class="p">)</code>

   <code class="p">})</code>

   <code class="p">}</code>

<code class="p">}</code>

<code></code><code class="kr">import</code> <code
class="p">{</code> <code
class="nx">ThemeInfoInterface</code> <code
class="p">}</code> <code class="nx">from</code>
<code class="s1">'../models'</code>

<code class="kr">import</code> <code class="p">
{</code> <code class="nx">StoreInterface</code> <code
class="p">}</code> <code class="nx">from</code>
<code class="s1">'./Store.model'</code>

<code class="kr">import</code> <code class="p">
{</code> <code class="nx">LocalStorageKeys</code>
<code class="p">}</code> <code
class="nx">from</code> <code
class="s1">'./LocalStorageKeys'</code>



<code class="cm">/**</code>

<code class="cm"> * @name ActionsInterface</code>
```

<code class="cm"> */</code>

<code class="kr">export</code> <code class="kr">interface</code> <code class="nx">ActionsInterface</code> <code class="p">{</code>

  <code class="nx">loadThemes</code><code class="p">(</code><code class="nx">themes</code>: <code class="kt">ThemeInfoInterface</code><code class="p">[])</code><code class="o">:</code> <code class="k">void</code>

  <code class="nx">selectTheme</code><code class="p">(</code><code class="nx">themeId</code>: <code class="kt">string</code><code class="p">)</code><code class="o">:</code> <code class="k">void</code>

<code class="p">}</code>

<code class="p">...</code>

<code></code><code class="p">...</code>

<code class="cm">/**</code>

<code class="cm"> * @name ActionsModel</code>

<code class="cm"> */</code>

<code class="kr">export</code> <code class="kr">class</code> <code class="nx">ActionsModel</code> <code

class="kr">implements</code> <code class="nx">ActionsInterface</code> <code class="p">{</code>

<code class="kr">private</code> <code class="nx">readonly</code> <code class="nx">store</code><code class="o">!:</code> <code class="nx">StoreInterface</code>

<code class="kr">constructor</code><code class="p">(</code><code class="nx">store</code>: <code class="kt">StoreInterface</code><code class="p">)</code> <code class="p">{</code>

<code class="k">this</code><code class="p">.</code> <code class="nx">store</code> <code class="o">=</code> <code class="nx">store</code>

<code class="p">}</code>

<code class="nx">loadThemes</code><code class="p">(</code><code class="nx">themes</code>: <code class="kt">ThemeInfoInterface</code><code class="p">[])</code> <code class="p">{</code>

<code class="k">this</code><code class="p">.</code> <code class="nx">store</code><code class="p">.</code><code class="nx">mutations</code><code class="p">.</code><code class="nx">loadThemes</code><code class="p">(</code><code class="nx">themes</code><code class="p">)</code>

<code class="p">}</code>

<code class="nx">selectTheme</code><code class="p">(</code><code class="nx">themeId</code>: <code class="kt">string</code><code class="p">)</code> <code class="p">{</code>

<code class="k">if</code> <code class="p">(</code> <code class="nb">document</code><code class="p">.</code><code class="nx">body</code><code class="p">.</code><code class="nx">className</code> <code class="p">.</code><code class="nx">indexOf</code><code class="p">(</code> <code class="s1">'-theme'</code><code class="p">)</code> <code class="o">></code> <code class="mi">0</code><code class="p">)</code> <code class="p">{</code>

<code class="nb">document</code><code class="p">.</code><code class="nx">body</code><code class="p">.</code><code class="nx">className</code> <code class="o">=</code> <code class="nb">document</code><code class="p">.</code> <code class="nx">body</code><code class="p">.</code><code class="nx">className</code><code class="p">.</code><code class="nx">replace</code> <code class="p">(</code>

<code class="sr">/[\w^-]+-theme+/gi</code><code class="p">,</code>

<code class="nx">themeId</code>

<code class="p">)</code>

```
   <code class="p">}</code> <code
class="k">else</code> <code class="p">{</code>

   <code class="nb">document</code><code class="p">.
</code><code class="nx">body</code><code
class="p">.</code><code class="nx">classList</code>
<code class="p">.</code><code class="nx">add</code>
<code class="p">(</code><code
class="nx">themeId</code><code class="p">)</code>

   <code class="p">}</code>

   <code class="nx">localStorage</code><code
class="p">.</code><code class="nx">setItem</code>
<code class="p">(</code><code
class="nx">LocalStorageKeys</code><code class="p">.
</code><code class="nx">theme</code><code
class="p">,</code> <code class="nx">themeId</code>
<code class="p">)</code>

   <code class="k">this</code><code class="p">.</code>
<code class="nx">store</code><code class="p">.
</code><code class="nx">mutations</code><code
class="p">.</code><code
class="nx">selectTheme</code><code class="p">
(</code><code class="nx">themeId</code><code
class="p">)</code>

   <code class="p">}</code>

<code class="p">}</code>

<code></code><code class="kr">import</code> <code
class="p">{</code> <code
class="nx">MutationsInterface</code><code class="p">,
</code> <code class="nx">MutationsModel</code>
<code class="p">}</code> <code
```

```
class="nx">from</code> <code
class="s1">'./Mutations.model'</code>

<code class="kr">import</code> <code class="p">
{</code> <code class="nx">ActionsInterface</code>
<code class="p">,</code> <code
class="nx">ActionsModel</code> <code class="p">}
</code> <code class="nx">from</code> <code
class="s1">'./Actions.model'</code>

<code class="kr">import</code> <code class="p">
{</code> <code class="nx">StateInterface</code> <code
class="p">}</code> <code class="nx">from</code>
<code class="s1">'./State.interface'</code>



<code class="cm">/**</code>

<code class="cm"> * @name Store interface</code>

<code class="cm"> */</code>

<code class="kr">export</code> <code
class="kr">interface</code> <code
class="nx">StoreInterface</code> <code class="p">
{</code>

   <code class="nx">state</code>: <code
class="kt">StateInterface</code>

   <code class="nx">mutations</code>: <code
class="kt">MutationsInterface</code>

   <code class="nx">actions</code>: <code
class="kt">ActionsInterface</code>
```

<code class="p">}</code>

<code class="cm">/**</code>

<code class="cm"> * @name Store model</code>

<code class="cm"> */</code>

<code class="kr">export</code> <code class="kr">class</code> <code class="nx">StoreModel</code> <code class="kr">implements</code> <code class="nx">StoreInterface</code> <code class="p">{</code>

  <code class="nx">readonly</code> <code class="nx">state</code><code class="o">!:</code> <code class="nx">StateInterface</code>

  <code class="nx">readonly</code> <code class="nx">mutations</code><code class="o">!:</code> <code class="nx">MutationsInterface</code>

  <code class="nx">readonly</code> <code class="nx">actions</code><code class="o">!:</code> <code class="nx">ActionsInterface</code>

  <code class="kr">constructor</code><code class="p">(</code><code class="nx">state</code>: <code class="kt">StateInterface</code><code class="p">)</code> <code class="p">{</code>

```
   <code class="k">this</code><code class="p">.</code>
<code class="nx">state</code> <code class="o">=
</code> <code class="nx">state</code>

   <code class="k">this</code><code class="p">.</code>
<code class="nx">mutations</code> <code class="o">=
</code> <code class="k">new</code> <code
class="nx">MutationsModel</code><code class="p">
(</code><code class="k">this</code><code class="p">.
</code><code class="nx">state</code><code
class="p">)</code>

   <code class="k">this</code><code class="p">.</code>
<code class="nx">actions</code> <code class="o">=
</code> <code class="k">new</code> <code
class="nx">ActionsModel</code><code class="p">
(</code><code class="k">this</code><code class="p">)
</code>

   <code class="p">}</code>

<code class="p">}</code>

<code></code><code class="p"><</code><code
class="nt">template</code><code class="p">></code>

   <code class="p"><</code><code
class="nt">label</code>

   <code class="na">role</code><code class="o">=
</code><code class="s">"radio"</code>

   <code class="na">:class</code><code class="o">=
</code><code class="s">"cssClass"</code><code
class="p">></code>
```

```
   <code class="p"><</code><code class="nt">i</code>
<code class="na">class</code><code class="o">=
</code><code class="s">"material-icons"</code><code
class="p">></code>{{ themeInfo.icon }}<code
class="p"></</code><code class="nt">i</code><code
class="p">></code>

   <code class="p"><</code><code
class="nt">input</code> <code class="na">type</code>
<code class="o">=</code><code class="s">"radio"
</code> <code class="na">class</code><code
class="o">=</code><code class="s">"icon-button"
</code> <code class="na">name</code><code
class="o">=</code><code class="s">"theme"</code>

   <code class="na">:value</code><code class="o">=
</code><code class="s">"themeInfo.selected"</code>

   <code class="na">v-model</code><code class="o">=
</code><code class="s">"themeInfo.selected"</code>

   <code class="err">@</code><code
class="na">click</code><code class="o">=</code>
<code class="s">"onClick"</code>

   <code class="p">/></code>

   <code class="p"></</code><code
class="nt">label</code><code class="p">></code>

<code class="p"></</code><code
class="nt">template</code><code class="p">></code>



...
```

```
<code></code><code class="p">...</code>



<code class="o"><</code><code
class="nx">script</code> <code class="nx">lang</code>
<code class="o">=</code><code class="s2">"ts"
</code><code class="o">></code>

   <code class="kr">import</code> <code class="p">
{</code> <code class="nx">defineComponent</code>
<code class="p">,</code> <code
class="nx">reactive</code><code class="p">,</code>
<code class="nx">computed</code><code class="p">,
</code> <code class="nx">PropType</code> <code
class="p">}</code> <code class="nx">from</code>
<code class="s1">'vue'</code>

   <code class="kr">import</code> <code class="p">
{</code> <code class="nx">ThemeInfoInterface</code>
<code class="p">}</code> <code
class="nx">from</code> <code
class="s1">'./models/ThemeInfo.interface'</code>



   <code class="kr">export</code> <code
class="k">default</code> <code
class="nx">defineComponent</code><code class="p">
({</code>

   <code class="nx">props</code><code class="o">:
</code> <code class="p">{</code>

   <code class="nx">themeInfo</code><code class="o">:
</code> <code class="p">{</code>
```

<code class="nx">type</code>: <code class="kt">Object</code> <code class="kr">as</code> <code class="nx">PropType</code><code class="o"><</code><code class="nx">ThemeInfoInterface</code> <code class="o">></code>

<code class="p">}</code>

<code class="p">},</code>

<code class="nx">computed</code><code class="o">:</code> <code class="p">{</code>

<code class="nx">cssClass</code><code class="p">()</code><code class="o">:</code> <code class="kt">string</code> <code class="p">{</code>

<code class="kr">const</code> <code class="nx">themeInfo</code> <code class="o">=</code> <code class="k">this</code><code class="p">.</code><code class="nx">$props</code><code class="p">.</code><code class="nx">themeInfo</code>

<code class="k">return</code> <code class="sb">`theme-radio </code><code class="si">${</code><code class="nx">themeInfo</code><code class="o">?</code><code class="p">.</code><code class="nx">id</code><code class="si">}</code><code class="sb"> </code><code class="si">${</code> <code class="nx">themeInfo</code><code class="o">?</code><code class="p">.</code><code class="nx">selected</code> <code class="o">?</code><code class="s1">'selected'</code> <code class="o">:</code> <code class="s1">'\</code>

```
<code class="s1">'</code> <code class="si">}</code>
<code class="sb">`</code><code class="p">.</code>
<code class="nx">trim</code><code class="p">()
</code>

   <code class="p">}</code>

   <code class="p">},</code>

   <code class="nx">setup</code><code class="p">
(</code><code class="nx">props</code><code
class="p">,</code> <code class="p">{</code> <code
class="nx">emit</code> <code class="p">})</code>
<code class="p">{</code>

   <code class="kr">const</code> <code
class="nx">onClick</code> <code class="o">=</code>
<code class="p">()</code> <code class="o">=></code>
<code class="p">{</code>

   <code class="nx">emit</code><code class="p">
(</code><code class="s1">'clicked'</code><code
class="p">,</code> <code class="nx">props</code>
<code class="p">.</code><code
class="nx">themeInfo</code><code class="o">?</code>
<code class="p">.</code><code class="nx">id</code>
<code class="p">)</code>

   <code class="p">}</code>



   <code class="k">return</code> <code class="p">
{</code>

   <code class="nx">onClick</code>
   <code class="p">}</code>

   <code class="p">}</code>

   <code class="p">})</code>

<code class="o"><</code><code class="err">/script>
</code>

<code></code><code class="p"><</code><code
class="nt">template</code><code class="p">></code>

   <code class="p"><</code><code
class="nt">div</code> <code class="na">class</code>
<code class="o">=</code><code class="s">"theme-
selector"</code><code class="p">></code>

   <code class="p"><</code><code
class="nt">div</code> <code class="na">class</code>
<code class="o">=</code><code class="s">"theme-
radio-group"</code><code class="p">></code>

   <code class="p"><</code><code
class="nt">ThemeRadio</code> <code class="na">v-
for</code><code class="o">=</code><code class="s">"
(theme, index) in themes"</code> <code
class="na">:key</code><code class="o">=</code>
<code class="s">"index"</code> <code
class="na">:themeInfo</code><code class="o">=
</code><code class="s">"theme"</code> <code
class="err">@\</code>

<code class="na">clicked</code><code class="o">=
</code><code class="s">"changeTheme"</code> <code
class="p">/></code>
```

  <code class="p"></</code><code class="nt">div</code><code class="p">></code>

  <code class="p"></</code><code class="nt">div</code><code class="p">></code>

<code class="p"></</code><code class="nt">template</code><code class="p">></code>

...

<code></code><code class="p">...</code>

<code class="o"><</code><code class="nx">script</code> <code class="nx">lang</code> <code class="o">=</code><code class="s2">"ts"</code><code class="o">></code>

  <code class="kr">import</code> <code class="p">{</code> <code class="nx">defineComponent</code> <code class="p">,</code> <code class="nx">PropType</code><code class="p">,</code> <code class="nx">computed</code><code class="p">,</code> <code class="nx">onMounted</code> <code class="p">}</code> <code class="nx">from</code> <code class="s1">'vue'</code>

  <code class="kr">import</code> <code class="p">{</code> <code class="nx">ThemeInfoInterface</code> <code class="p">}</code> <code class="nx">from</code> <code class="s1">'./models/ThemeInfo.interface'</code>

```
<code class="kr">import</code> <code class="p">
{</code> <code class="nx">Store</code> <code
class="p">}</code> <code class="nx">from</code>
<code class="s1">'./ThemesStore'</code>

<code class="kr">import</code> <code class="p">
{</code> <code class="nx">LocalStorageKeys</code>
<code class="p">}</code> <code
class="nx">from</code> <code class="s1">'./store-
models'</code>

<code class="kr">import</code> <code
class="nx">ThemeRadio</code> <code
class="nx">from</code> <code
class="s1">'./ThemeRadio.component.vue'</code>



<code class="kr">export</code> <code
class="k">default</code> <code
class="nx">defineComponent</code><code class="p">
({</code>

<code class="nx">components</code><code
class="o">:</code> <code class="p">{</code>

<code class="nx">ThemeRadio</code>

<code class="p">},</code>

<code class="nx">props</code><code class="o">:
</code> <code class="p">{</code>

<code class="nx">availableThemes</code><code
class="o">:</code> <code class="p">{</code>
```

```
  <code class="nx">type</code>: <code
class="kt">Array</code> <code class="kr">as</code>
<code class="nx">PropType</code><code class="o">
<</code><code class="nx">ThemeInfoInterface</code>
<code class="p">[]</code><code class="o">></code>
<code class="p">,</code>

  <code class="k">default</code><code class="o">:
</code> <code class="p">[]</code>

  <code class="p">}</code>

  <code class="p">},</code>

  <code class="nx">computed</code><code class="o">:
</code> <code class="p">{</code>

  <code class="nx">themes</code><code class="o">:
</code> <code class="p">()</code> <code class="o">=>
</code> <code class="p">{</code>

  <code class="k">return</code> <code
class="nx">Store</code><code class="p">.</code>
<code class="nx">state</code><code class="p">.
</code><code class="nx">themes</code>

  <code class="p">}</code>

  <code class="p">},</code>

  <code class="nx">setup</code><code class="p">
(</code><code class="nx">props</code><code
class="p">,</code> <code class="p">{</code> <code
class="nx">emit</code> <code class="p">})</code>
<code class="p">{</code>

  <code class="c1">// handles theme radio click:</code>
```

```
   <code class="kr">const</code> <code
class="nx">changeTheme</code> <code class="o">=
</code> <code class="p">(</code><code
class="nx">themeId</code>: <code
class="kt">string</code><code class="p">)</code>
<code class="o">=></code> <code class="p">{</code>

   <code class="nx">emit</code><code class="p">
(</code><code class="s1">'themeClicked'</code><code
class="p">,</code> <code class="nx">themeId</code>
<code class="p">)</code>

   <code class="nx">Store</code><code class="p">.
</code><code class="nx">actions</code><code
class="p">.</code><code
class="nx">selectTheme</code><code class="p">
(</code><code class="nx">themeId</code><code
class="p">)</code>

   <code class="nx">emit</code><code class="p">
(</code><code class="s1">'themeChanged'</code><code
class="p">,</code> <code class="nx">themeId</code>
<code class="p">)</code>

   <code class="p">}</code>

<code class="p">...</code>

<code></code><code class="p">...</code>



   <code class="c1">// on mounted, initialize the themes,
and make sure we select a theme</code>

   <code class="nx">onMounted</code><code class="p">
(()</code> <code class="o">=></code> <code
```

class="p">{</code>

   <code class="nx">Store</code><code class="p">.
</code><code class="nx">actions</code><code
class="p">.</code><code
class="nx">loadThemes</code><code class="p">
(</code><code class="nx">props</code><code
class="p">.</code><code
class="nx">availableThemes</code><code class="p">)
</code>

   <code class="c1">// for the current theme, try reading
from localStorage</code>

   <code class="kr">const</code> <code
class="nx">userPreferredThemeId</code>: <code
class="kt">string</code> <code class="o">=</code>
<code class="nx">localStorage</code><code class="p">.
</code><code class="nx">getItem</code><code
class="p">(</code><code
class="nx">LocalStorageKeys</code><code class="p">.
</code><code class="nx">t</code><code class="o">\
</code>

<code class="nx">heme</code><code class="p">)
</code> <code class="o">||</code> <code
class="s1">''</code>

   <code class="k">if</code> <code class="p">(</code>
<code class="nx">userPreferredThemeId</code><code
class="p">.</code><code class="nx">length</code>
<code class="o">></code> <code class="mi">0</code>
<code class="p">)</code> <code class="p">{</code>

```
   <code class="c1">// select from user preference saved in
localStorage</code>

   <code class="nx">changeTheme</code><code
class="p">(</code><code
class="nx">userPreferredThemeId</code><code
class="p">)</code>

   <code class="p">}</code> <code
class="k">else</code> <code class="p">{</code>

   <code class="k">if</code> <code class="p">(</code>
<code class="nx">props</code><code class="p">.
</code><code class="nx">availableThemes</code><code
class="p">.</code><code class="nx">length</code>
<code class="o">></code> <code class="mi">0</code>
<code class="p">)</code> <code class="p">{</code>

   <code class="kr">const</code> <code
class="nx">selected</code> <code class="o">=</code>
<code class="nx">props</code><code class="p">.
</code><code class="nx">availableThemes</code><code
class="p">.</code><code class="nx">find</code><code
class="p">(</code><code class="nx">item</code>
<code class="o">=></code> <code
class="nx">item</code><code class="p">.</code><code
class="nx">selected</code><code class="p">)</code>

   <code class="k">if</code> <code class="p">(</code>
<code class="nx">selected</code><code class="p">)
</code> <code class="p">{</code>

   <code class="nx">changeTheme</code><code
class="p">(</code><code class="nx">selected</code>
<code class="p">.</code><code class="nx">id</code>
<code class="p">)</code>
```

```
   <code class="p">}</code> <code
class="k">else</code> <code class="p">{</code>

   <code class="c1">// select the first one</code>

   <code class="nx">changeTheme</code><code
class="p">(</code><code class="nx">props</code>
<code class="p">.</code><code
class="nx">availableThemes</code><code class="p">
[</code><code class="mi">0</code><code class="p">].
</code><code class="nx">id</code><code class="p">)
</code>

   <code class="p">}</code>

   <code class="p">}</code>

   <code class="p">}</code>

   <code class="p">})</code>



   <code class="k">return</code> <code class="p">
{</code>

   <code class="nx">changeTheme</code>

   <code class="p">}</code>

   <code class="p">}</code>

   <code class="p">})</code>

<code class="o"><</code><code class="err">/script>
</code>

<code></code><code class="p">...</code>
```

```
  <code class="kr">const</code> <code class="nx">changeTheme</code> <code class="o">=</code> <code class="p">(</code><code class="nx">themeId</code>: <code class="kt">string</code><code class="p">)</code> <code class="o">=></code> <code class="p">{</code>

  <code class="nx">emit</code><code class="p">(</code><code class="s1">'themeClicked'</code><code class="p">,</code> <code class="nx">themeId</code> <code class="p">)</code>

  <code class="nx">Store</code><code class="p">.</code><code class="nx">actions</code><code class="p">.</code><code class="nx">selectTheme</code><code class="p">(</code><code class="nx">themeId</code><code class="p">)</code>

  <code class="nx">emit</code><code class="p">(</code><code class="s1">'themeChanged'</code><code class="p">,</code> <code class="nx">themeId</code> <code class="p">)</code>

  <code class="p">}</code>

<code class="p">...</code>

<code></code><code class="kr">import</code> <code class="p">{</code>

  <code class="nx">ThemeInfoInterface</code>

<code class="p">}</code> <code class="nx">from</code> <code class="s1">'./theme-selector/models'</code>
```

```
<code class="kr">import</code> <code
class="nx">ThemeSelector</code> <code
class="nx">from</code> <code class="s1">'./theme-
selector/ThemeSelector.component.vue'</code>



<code class="kr">export</code> <code class="p">
{</code>

   <code class="nx">ThemeInfoInterface</code><code
class="p">,</code>

   <code class="nx">ThemeSelector</code>

<code class="p">}</code>

<template>

   <div id="app">

   <div class="long-date">{{ i18n.d(new Date(), 'long') }}
</div>

   <h2>{{ i18n.t('welcome') }}</h2>

   <LocaleSelector :availableLocales="availableLocales"
@clicked="onLocaleClicked" \

/>

<code class="udl"> <ThemeSelector /> <!-- remove this
line -->

</code><code class="uil"> <ThemeSelector
:availableThemes="availableThemes" /> <!-- add this line --
>
```

</code> <div id="nav" class="nav">

   <router-link to="/">{{ i18n.t('navigation.home') }}
</router-link> |

   <router-link to="/about">{{ i18n.t('navigation.about') }}
</router-link>

   </div>

...

...

<script lang="ts">

   ...

<code class="udl"> import { ThemeSelector } from
'@/components/theme-selector/

</code><code class="udl">ThemeSelector.component.vue'
// <-- remove this line

</code><code class="uil"> import { ThemeSelector } from
'@/components-standalone' // <-- add this line

</code><code class="uil"> import { config } from
'@/config' // <-- add this line

</code>

...

```
...

   setup() {

   ...

<code class="uil"> const availableThemes = config.themes
// <-- add this line

</code> ...



   return {

   i18n,

   availableLocales,

   onLocaleClicked,

<code class="uil"> availableThemes // <-- add this line

</code> }

   }



...

<code></code><code class="err">...</code>



<code class="s2">"themes"</code><code class="err">:
</code> <code class="p">[{</code>
```

<code class="nt">"id"</code><code class="p">:</code> <code class="s2">"default-theme"</code><code class="p">,</code>

<code class="nt">"name"</code><code class="p">:</code> <code class="s2">"Default"</code><code class="p">,</code>

<code class="nt">"icon"</code><code class="p">:</code> <code class="s2">"color_lens"</code><code class="p">,</code>

<code class="nt">"selected"</code><code class="p">:</code> <code class="kc">false</code>

<code class="p">},</code> <code class="p">{</code>

<code class="nt">"id"</code><code class="p">:</code> <code class="s2">"dark-theme"</code><code class="p">,</code>

<code class="nt">"name"</code><code class="p">:</code> <code class="s2">"Dark"</code><code class="p">,</code>

<code class="nt">"icon"</code><code class="p">:</code> <code class="s2">"color_lens"</code><code class="p">,</code>

<code class="nt">"selected"</code><code class="p">:</code> <code class="kc">true</code>

<code class="p">},</code>

<code class="p">{</code>

```
   <code class="nt">"id"</code><code class="p">:</code> <code class="s2">"navy-theme"</code><code class="p">,</code>

   <code class="nt">"name"</code><code class="p">:</code> <code class="s2">"Navy Dark"</code><code class="p">,</code>

   <code class="nt">"icon"</code><code class="p">:</code> <code class="s2">"color_lens"</code><code class="p">,</code>

   <code class="nt">"selected"</code><code class="p">:</code> <code class="kc">false</code>

   <code class="p">}]</code><code class="err">,</code>



<code class="err">...</code>
```

If you do not want to add them to the config, you can just add a similar array in the **App.vue setup()**, so instead of **const availableThemes = config.themes** you would have something like **const availableThemes = yourarrayhere**

If you refresh the browser page, the **ThemeSelector** button should render as before, but this time after you select a different theme in the UI, it will remember your selection next time you visit the same page (or just press **F5** to refresh the entire page) as it reads it from **localStorage**.

Chapter 16 Recap

What We Learned

- We learned how to create a **standalone component** that uses a **local custom state** created with Vue’s **reactive**
- We learned how to drive the themes from the config **json** files

Observations

- We did not write unit tests for the **ThemeSelelector custom state**
- We did not write unit tests for the **ThemeSelelector** component
- I did not show you how to package your standalone component for distribution through **npm** or as a **tgz compressed** file

Based on these observations, there are a few improvements that can be done:

Improvements

- You should write unit tests for the **ThemeSelelector custom state actions** and **mutations**
- You should write unit tests for the **ThemeSelector** component

- In the next chapter I will show you how to package your standalone component for distribution through **npm** or as a **tgz compressed** file so you can consume it more easily across different apps

Chapter 17 - Packaging Component Libraries for Distribution

Let’s start by creating a new project with the **vue-cli** that will contain only **standalone** components. This could contain more than one component if you wish, as we’ll publish it as a **library package**. But in this chapter we’ll only add the **ThemeSelector** component to it.

First make sure you have the latest vue-cli:

```
npm i -g @vue/cli
```

Then in your home or documents directory, create a brand new project like this:

```
vue create my-component-library
```

Select option **Manually select features**:

```
Vue CLI v4.5.6
? Please pick a preset:
  Default ([Vue 2] babel, eslint)
  Default (Vue 3 Preview) ([Vue 3] babel, eslint)
❯ Manually select features
```

The select only the **Babel**, **TypeScript** and **UnitTesting features** *(Note: if you’ll be creating standalone components that include CSS in the <style> section of the .vue single files, then also select CSS Pre-processors):*

```
Vue CLI v4.5.6
? ...
? Check the features needed for your project:
 ◉ Choose Vue version
 ◉ Babel
 ◉ TypeScript
 ○ Progressive Web App (PWA) Support
 ○ Router
 ○ Vuex
 ○ CSS Pre-processors
 ○ Linter / Formatter
❯◉ Unit Testing
 ○ E2E Testing
```

Select **3.x (Preview)** for the Vue version:

```
Vue CLI v4.5.6
? ...
? Choose a version of Vue.js that you want to start the project with
  2.x
❯ 3.x (Preview)
```

Answer No to **Use class-style component syntax?**:

```
Vue CLI v4.5.6
? ...
? Use class-style component syntax? (y/N) N
```

Answer Yes to **Use Babel alongside TypeScript…?**:

```
Vue CLI v4.5.6
? ...
? Use Babel alongside TypeScript (required for modern mode, auto-detected
polyfi
lls, transpiling JSX)? (Y/n) Y
```

Select **Mocha+Chai** for the unit testing solution:

```
Vue CLI v4.5.6
? ...
? Pick a unit testing solution: (Use arrow keys)
❯ Mocha + Chai
  Jest
```

Select **In dedicated config files** for the config:

```
Vue CLI v4.5.6
? ...
? Pick a unit testing solution: Mocha
? Where do you prefer placing config for Babel, ESLint, etc.? (Use arrow keys)
❯ In dedicated config files
  In package.json
```

Answer NO for **Save this as a preset for future projects?**:

```
Vue CLI v4.5.6
? ...
? Save this as a preset for future projects? (y/N) N
```

Wait till the vue-cli creates all the project files, then navigate to the my-component-library directory:

```
cd my-component-library/
```

In the next section we'll open the folder in VS Code and start making a few changes.

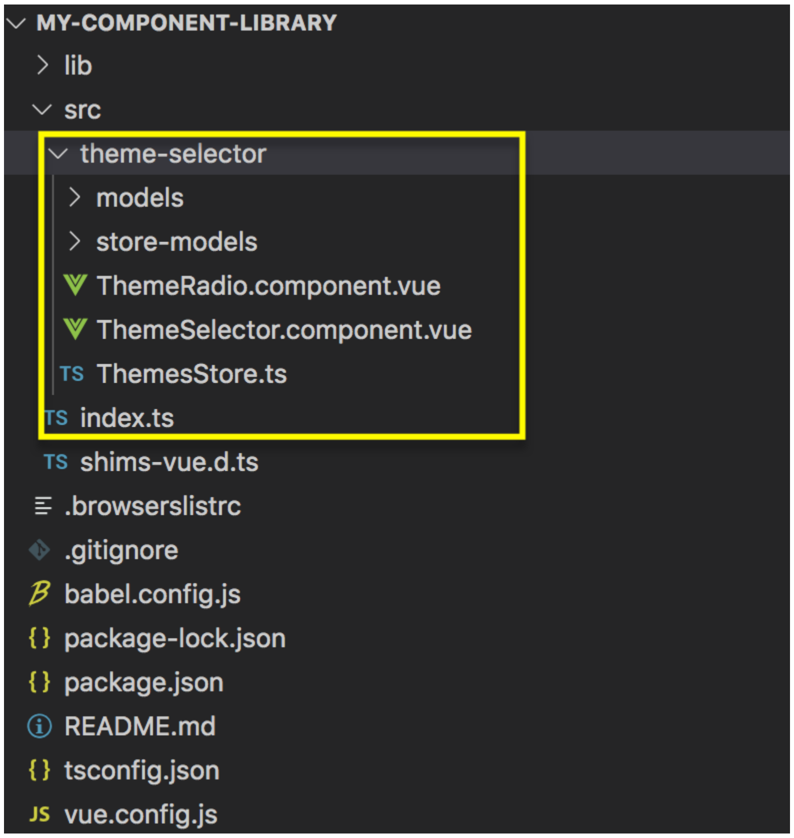
Remove files that are not needed

Remove all of these:

- The **public** folder
- The **src/assets**
- The **src/App.vue** and **src/main.ts** files
- The **components** directory
- The **tests/unit/HelloWorld.spec.ts** file

Bring the ThemeSelector files into this project

From the main book project located at **my-project**, bring all the files currently inside **src/components-standalone** into the **my-component-library/src/** directory.



```
MY-COMPONENT-LIBRARY
  lib
  src
    theme-selector
      models
      store-models
      ThemeRadio.component.vue
      ThemeSelector.component.vue
      ThemesStore.ts
    index.ts
    shims-vue.d.ts
  .browserslistrc
  .gitignore
  babel.config.js
  package-lock.json
  package.json
  README.md
  tsconfig.json
  vue.config.js
```

Update tsconfig.json

Update the **tsconfig.json** file **compilerOptions** by adding **declaration** and **outDir** options, and setting the **sourceMap** one to false like this:

```
{
  "compilerOptions": {
    "target": "esnext",
    "module": "esnext",
    "strict": true,
    "declaration": true, // <-- add this line
    "outDir": "lib", // <-- add this line
    "jsx": "preserve",
...
~~    "sourceMap": true,  // <-- remove this line~~
    "sourceMap": false,  // <-- add this line
...
```

Also make sure the **include** and **exclude** sections are replaced with this:

```
...
  "include": [
    "src/**/*.ts",
    "src/**/*.tsx",
    "src/**/*.vue"
  ],
  "exclude": [
    "node_modules",
    "src/main.ts",
    "src/App.vue",
    "tests/**/*.ts",
    "tests/**/*.tsx"
  ]
...
```

Update package.json

Add **main** and **types**:

```
...
  "main": "lib/my-component-library.umd.js",
  "types": "lib/index.d.ts",

...
```

Update the **scripts** section like this:

```
...
  "scripts": {
    "build-lib": "vue-cli-service build --target lib src/index.ts && rm
lib/demo.htm\
l",
    "build-types": "tsc --emitDeclarationOnly",
    "build": "npm run build-lib && npm run build-types",
    "pack": "npm pack && mv ./my-component-library-0.1.0.tgz ../my-component-
library\
-0.1.0.tgz
",
    "all": "npm run build && npm run pack"
  },

...
```

Add vue.config.js file

Add a new file called **vue.config.js** at the root of **my-component-library** directory with the following code:

```
const path = require('path');

module.exports = {
  productionSourceMap: false,
  outputDir: './lib',
  /* publicPath is used when we execute "npm run build" to prefix references
for scr\
ipt/css files */
  publicPath: './',
  /* additional webpack configuration */
  chainWebpack: (config) => {
    config.resolve.alias.set('@', path.resolve(__dirname, 'src'))

    if (process.env.NODE_ENV === 'production') {
      // if we need to make exception for some things that should not go into
the li\
b folder
    }
  },
  configureWebpack: {
    output: {
      filename: '[name].js',
      chunkFilename: '[name].js'
    },
    resolve: {
      alias: {
        "@": path.resolve(__dirname, 'src')
      }
    }
  }
}
```

Try to build

Now let’s try to build the library in steps. First execute the command **npm run build-lib**. If all goes well, it should create the **umd.js** and **common.js** files under **lib** output directory. In the terminal, it should print something like this:

```
DONE Compiled successfully in 3223ms
\
  10:28:17 AM

  File                                 Size                         Gzipped

  lib/my-component-library.umd.min.js    27.18 KiB                    9.89 KiB
  lib/my-component-library.umd.js        89.10 KiB                    21.43 KiB
  lib/my-component-library.common.js     88.60 KiB                    21.25 KiB

  Images and other types of assets omitted.
```

Then execute the command **npm run build-types**. If all goes well, it should create the **TS declaration (d.ts) files** under **lib** directory. In the terminal, it should print something like this:

```
> my-component-library@0.1.0 build-types /Users/damiano/Documents/no-repo/my-
compone\
nt-library
> tsc --emitDeclarationOnly
```

Finally execute the command **npm run pack**. If all goes well, it should pack the package and create the file **my-component-library-0.1.0.tgz** and move it one directory outside the library directory. In the terminal, it should print something like this:

```
...
npm notice
npm notice □  my-component-library@0.1.0
npm notice === Tarball Contents ===
npm notice 30B     .browserslistrc
npm notice 73B     babel.config.js
npm notice 90.7kB  lib/my-component-library.common.js
npm notice 91.2kB  lib/my-component-library
...
npm notice === Tarball Details ===
npm notice name:          my-component-library
npm notice version:       0.1.0
npm notice filename:      my-component-library-0.1.0.tgz
npm notice package size:  228.3 kB
npm notice unpacked size: 393.3 kB
...
npm notice total files:   33
npm notice
my-component-library-0.1.0.tgz
```

*Note: there is also a shortcut in the scripts section call "all", so you could run all the build and pack steps at one time with **npm run all** in the future. It is useful to also have them separated for easier debugging if there are are build errors.*

In the next section we can switch to **my-project** again and start using the **tgz** package we build.

Updates to my-project

Remove the entire **components-standalone** directory. Then run this command to install the local **tgz** compressed package we created in the previous section *(note: the ../*

path is relative to where the tgz file is, so make sure it is in the right place):

```
npm install --save ../my-component-library-0.1.0.tgz
```

Note: The previous install command will add an entry to your dependencies section in the package.json file like this:

```
...
  "dependencies": {
...
    "my-component-library": "file:../my-component-library-0.1.0.tgz",
...
```

Updates to ConfigInterface

Open the **src/config/Config.interface.ts** file and modify the import for the **ThemeInfoInterface** to now read it from the imported package:

```
...
~~import { ThemeInfoInterface } from '@/components-standalone' // <-- remove this line~~
import { ThemeInfoInterface } from 'my-component-library' // <-- add this line
...
```

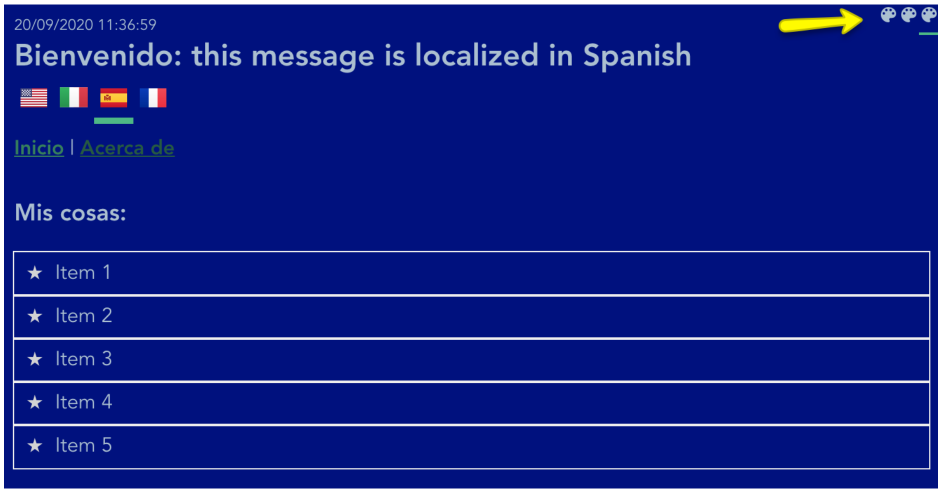
Updates to App.vue

Open the **src/App.vue** file and here too modify the import for the **ThemeInfoInterface** to now read it from the imported package:

```
...
~~  import { ThemeSelector } from '@/components-standalone' // <-- remove this line~~
  import { ThemeSelector } from 'my-component-library' // <-- add this line
...
```

Finally execute **npm run serve** to run the my-project app again and verify that every runs without errors in the terminal and in the browser.

The **ThemeSelector** should work exactly like before. Also if you refresh the whole page with **F5** or close the browser and reopen it, it will remember the theme selected as the **ThemeSelector** custom store will read the value previously save in **localStorage**.


20/09/2020 11:36:59

Bienvenido: this message is localized in Spanish

Inicio | Acerca de

Mis cosas:

★ Item 1
★ Item 2
★ Item 3
★ Item 4
★ Item 5

Chapter 17 Recap

What We Learned

- We learned how to create a **component library** that we can package in a compressed **tgz** file for distribution
- We learned how to consume the **tgz** file through **npm install**

Observations

- We did not write **unit tests** against the components in the library
- We did not publish the package on **npmjs** or similar service for easier distribution

Based on these observations, there are a few improvements that can be done:

Improvements

- You should write **unit tests** against the components in the library
- You could publish the **my-component-library** to npm (or similar service) and consume it from there instead of a local **tgz** file

```
<code></code><code class="kr">const</code> <code
class="nx">path</code> <code class="o">=</code>
<code class="nx">require</code><code class="p">
(</code><code class="s1">'path'</code><code
class="p">);</code>



<code class="nx">module</code><code class="p">.
</code><code class="nx">exports</code> <code
class="o">=</code> <code class="p">{</code>

  <code class="nx">devServer</code><code class="o">:
</code> <code class="p">{</code>

  <code class="nx">port</code><code class="o">:
</code> <code class="mi">8080</code>

  <code class="p">},</code>

  <code class="nx">productionSourceMap</code><code
class="o">:</code> <code class="kc">true</code><code
class="p">,</code>

  <code class="nx">outputDir</code><code class="o">:
</code> <code class="s1">'dist'</code><code
class="p">,</code>

  <code class="cm">/* publicPath is used when we execute
"npm run build" to prefix references for scr\</code>

<code class="cm">ipt/css files */</code>

  <code class="nx">publicPath</code><code class="o">:
</code> <code class="s1">'./'</code><code class="p">,
</code>
```

```
   <code class="cm">/* chainWebpack/configureWebpack
so that npm run build does not produce hashed fil\</code>

<code class="cm">e names under dist/ folder */</code>

   <code class="nx">chainWebpack</code><code
class="o">:</code> <code class="p">(</code><code
class="nx">config</code><code class="p">)</code>
<code class="p">=></code> <code class="p">{</code>

   <code class="nx">config</code><code class="p">.
</code><code class="nx">resolve</code><code
class="p">.</code><code class="nx">alias</code><code
class="p">.</code><code class="nx">set</code><code
class="p">(</code><code class="s1">'@'</code><code
class="p">,</code> <code class="nx">path</code>
<code class="p">.</code><code
class="nx">resolve</code><code class="p">(</code>
<code class="nx">__dirname</code><code class="p">,
</code> <code class="s1">'src'</code><code
class="p">))</code>



   <code class="k">if</code> <code class="p">(</code>
<code class="nx">config</code><code class="p">.
</code><code class="nx">plugins</code><code
class="p">.</code><code class="nx">has</code><code
class="p">(</code><code class="s1">'html'</code>
<code class="p">))</code> <code class="p">{</code>

   <code class="kr">const</code> <code
class="nx">htmlPlugin</code> <code class="o">=
</code> <code class="nx">config</code><code
class="p">.</code><code class="nx">plugin</code>
<code class="p">(</code><code
class="s1">'html'</code><code class="p">);</code>
```

```
  <code class="k">if</code> <code class="p">(</code>
<code class="nx">htmlPlugin</code><code class="p">)
</code> <code class="p">{</code>

  <code class="nx">htmlPlugin</code><code class="p">.
</code><code class="nx">tap</code><code class="p">
(</code><code class="nx">args</code> <code
class="p">=></code> <code class="p">{</code>

  <code class="nx">args</code><code class="p">
[</code><code class="mi">0</code><code class="p">].
</code><code class="nx">title</code> <code
class="o">=</code> <code class="s1">'Vue 3 Large Scale
Apps with TypeScript Sample Project'</code><code
class="p">;</code>

  <code class="k">return</code> <code
class="nx">args</code><code class="p">;</code>

  <code class="p">});</code>

  <code class="p">}</code>

  <code class="p">}</code>

  <code class="err">…</code>

<code></code> <code class="err">…</code>



  <code class="k">if</code> <code class="p">(</code>
<code class="nx">config</code><code class="p">.
</code><code class="nx">plugins</code><code
class="p">.</code><code class="nx">has</code><code
class="p">(</code><code class="s1">'extract-
css'</code><code class="p">))</code> <code class="p">
{</code>

   <code class="kr">const</code> <code
class="nx">extractCSSPlugin</code> <code class="o">=
</code> <code class="nx">config</code><code
class="p">.</code><code class="nx">plugin</code>
<code class="p">(</code><code class="s1">'extract-
css'</code><code class="p">);</code>

   <code class="nx">extractCSSPlugin</code> <code
class="o">&&</code> <code
class="nx">extractCSSPlugin</code><code class="p">.
</code><code class="nx">tap</code><code class="p">
(()</code> <code class="p">=></code> <code
class="p">[{</code>

   <code class="nx">filename</code><code class="o">:
</code> <code class="s1">'[name].css'</code><code
class="p">,</code>

   <code class="nx">chunkFilename</code><code
class="o">:</code> <code
class="s1">'[name].css'</code>

   <code class="p">}])</code>

   <code class="p">}</code>



   <code class="k">if</code> <code class="p">(</code>
<code class="nx">process</code><code class="p">.
</code><code class="nx">env</code><code class="p">.
</code><code class="nx">NODE_ENV</code> <code
class="o">===</code> <code
```

```
class="s1">'production'</code><code class="p">)
</code> <code class="p">{</code>

   <code class="c1">// if we need to make exception for
some things that should not go into the di\</code>

<code class="nx">st</code> <code
class="nx">folder</code>

   <code class="p">}</code>

   <code class="p">},</code>

   <code class="nx">configureWebpack</code><code
class="o">:</code> <code class="p">{</code>

   <code class="nx">output</code><code class="o">:
</code> <code class="p">{</code>

   <code class="nx">filename</code><code class="o">:
</code> <code class="s1">'[name].js'</code><code
class="p">,</code>

   <code class="nx">chunkFilename</code><code
class="o">:</code> <code class="s1">'[name].js'</code>

   <code class="p">},</code>

   <code class="nx">resolve</code><code class="o">:
</code> <code class="p">{</code>

   <code class="nx">alias</code><code class="o">:
</code> <code class="p">{</code>

   <code class="s2">"@"</code><code class="o">:
</code> <code class="nx">path</code><code
class="p">.</code><code class="nx">resolve</code>
<code class="p">(</code><code
```

class="nx">__dirname</code><code class="p">,</code> <code class="s1">'src'</code><code class="p">)</code>

    <code class="p">}</code>

    <code class="p">}</code>

    <code class="p">}</code>

<code class="p">}</code>

Naming Conventions

In this book we have been providing some direction on both naming standards for code elements like interface, classes etc, as well as for directory and file names. Here is a detailed description of the standard we followed in this book.

NOTE: These are mostly suggestions, a starting point. You should always agree with your team on the naming conventions and directory structure standards that will work best for your company. Then the whole team should commit to follow those conventions.

Coding Standards

TypeScript any

Avoid using **any** and rather always choose an interface/type/class

Interfaces

Interfaces are named with an **Interface** suffix. For example, an interface representing **Item** will be named **ItemInterface**.

Each interface will be contained in its own file. The file naming convention will be **Item.interface.ts**.

Directory/File Naming and Structure

Directory Names

In general, we’ll use lower-case for naming directories. When this contains multiple words, they will be separated by a hyphen (dash). I.e. **items-with-children**

We try to keep code files organized in directories by category (i.e. **components**, **models**, **plugins**) and sub-directories

Sub-directories are organized by app domain for models, i.e. **models/items**, **models/customers**, **models/locales** etc

For components, they are organized by component domain or functionality, i.e. **components/items**, **components/locales** etc.

In general, if a model or a component is used across all domains, then the sub-directory name is **shared** (or **common** if you prefer), i.e. **components/shared**

File Names

In general, files will be named with a pascal-case convention, I.e. **ItemsWithChildren.ts**

Interface File Names

Files containining interfaces will follow the convention **[Name].interface.ts**, i.e. **Item.interface.ts**.

Vue single-file Components File Names

Vue components files will be under **src/components** directory.\
Their names follow the convention **[ComponentName].component.vue**. I.e. **ItemsList.component.vue**

Vue single-file Views File Names

Vue views files will be under **src/views** directory.\
Their names follow the convention **[ViewName].vue**, without adding any suffix like for the components (*NOTE: in **Vue**, everything is really a component, including views. The separation is mostly for organization purposes. The way we consume views and components differs and we talk more about this throughout the book*).

Unit Tests file names

For unit tests, we'll follow the convention **[ClassOrComponentBeingTested].spec.ts**. I.e. **ItemsListComponent.spec.ts**

NOTE: If you have to write many unit tests against the same class or component to test specific areas (i.e. security, permissions etc) might be a good idea to also split the code into additional files named with additional suffixes (as long as you adopt a standard that makes sense and it's easy to follow).

This could be a convention to follow in that case: **[ClassOrComponentBeingTested].[area-tested].[condition].[value].spec.ts** and here are a couple of examples:

- **ItemsListComponent.permissions.view.no.vue** (to test when user does not have View permisions)
- **ItemsListComponent.permissions.view.yes.vue** (to test when user has iew permisions)

Directory src

Contains all the **Vue** source code

- src/assets: contains static assets like image files etc

    - src/assets/images: contains all the static images

src/api: contains the API clients implementations

- **src/api/mock**: contains the API clients that return mock data
- **src/api/live**: contains the API clients that communicate with the real API end-points

src/components: contains all the **Vue** single file components (extension **.vue**)

- **src/components/[lowercase-component-name]**: contains all the files that make up a specific component. I.e. **src/components/items**
    - **src/components/[lowercase-component-name]/children**: contains all the sub-components, if any, consumed only by our main component. I.e. **src/components/items/children**

src/models: contains all the pure TypeScript interface/types/classes/models/etc (extension .ts), with the exception of the Vue **main.ts** file which is located directly under src/

- **src/models/[domain]**: contains all the interfaces/classes/etc that are related to a particular domain, I.e. **items**
- **src/models/store**: contains all the interfaces for our **Vuex** store and store modules organized in sub-directories by specific domain:
    - **src/models/store/[domain]** contains all the interfaces for a specific **Vuex** store module

src/router: contains the **Vue** router code implementation

src/store: contains the **Vuex** store implementation

- **src/store/[domain]**: contains the store module implementation for a specific domain, I.e. **items**

src/views: contains all the **Vue** single file views (extension **.vue**), except for the **App.vue** which is located directly under **src/**

Directory tests/unit

Contains all the unit and end-to-end tests.

- **tests/unit**: contains the unit tests
  - **tests/unit/components**: contains all the unit tests against Vue components
  - **tests/unit/models**: contains all the unit tests against the models/interfaces/types etc
- **tests/end-to-end**: contains the end-to-end tests (not used in this book)

(More Chapters Coming Soon)

Notes

Preface

1 *Evan You on Twitter*: https://twitter.com/youyuxi*↩

2 *Official website:* https://vuejs.org↩

Prerequisites

1 https://www.typescriptlang.org↩

2 https://vuejs.github.io/vetur↩

Chapter 1 - Setting Up The Project With The Vue-Cli

1 *vue-cli version used at the time of this writing is 4.5.3*↩

2 *reference* https://cli.vuejs.org/guide/creating-a-project.html↩

3 https://www.npmjs.com/package/cross-env↩

Chapter 2 - Your First Component

1 *We are following a file naming convention where Vue components are pascal-case and follow this format [ComponentName].component.vue (Reference: Naming Conventions section at the end of this book)*↩

2 *With ‘any’, TypeScript does not enforce type-checking on a property or variable. However, this is considered a bad practice as we lose the main benefit of TypeScript. There might be exceptions to this rule when using older 3rd party packages/libraries/plugins that do not offer type definitions. However, even in those cases it would be strongly recommended to provide interfaces and types so that you can still avoid using ‘any’.*↩

3 *Note: using hard-coded data is a bad practice and here we are only doing it to first illustrate how things flow, and later in the next chapters will remove in favor of best practices and patterns (see Chapter 5)*↩

4 *In Vue, the colon prefix on a DOM element attribute is a shorthand for the v-bind directive. In this case :items works exactly as v-bind:items but we prefer the shorthand as this makes the code less cluttered. Reference:* https://vuejs.org/v2/guide/syntax.html#v-bind-Shorthand↩

Chapter 3 - Data Model Interfaces

1 *There have been suggestions presented, but I do not think they will ever add a struct type. See the TypeScript team answers here:* https://github.com/microsoft/TypeScript/issues/22101↩

Chapter 4 - Adding Events To the Items Component

1 *In Vue, the @ prefix on a DOM element attribute is a shorthand for the v-on directive. Here, @click works exactly as v-on:click, but we prefer the shorthand as this makes the code less cluttered. Reference:* https://vuejs.org/v2/guide/syntax.html#v-on-Shorthand↩

2 *Reference:* https://vuex.vuejs.org↩

Chapter 5 - Intro to Unit Testing While Refactoring a Bit

1 *It’s customary to use the suffix .spec on file names that represent Mocha tests. Other testing framework also following similar convention, i.e. Jasmine*↩

Chapter 6 - Introducing Vex

1 *You do not have to include CSS as part of the component if you do not want to, or you already use a CSS framework with centralized files*↩

2 https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment↩

3 https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/find↩

Chapter 7 - Api Client

1 https://jsonplaceholder.typicode.com *or* https://miragejs.com *for example*↩

Chapter 8 - Enhance the Api Client

1 https://sinonjs.org↩

2 https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise↩

3 https://mochajs.org/#hooks↩

Chapter 9 - Modularize the Vuex Store

1 https://github.com/damianof/large-scale-apps-my-vue3-project↩

2 https://vuex.vuejs.org/guide/modules.html↩

3 https://github.com/damianof/large-scale-apps-my-vue3-project↩

Chapter 10 - Localization and Internationalization - Language Localization - Part 1

1 *Reference:* https://www.w3.org/International/questions/qa-i18n↩

2 https://github.com/kazupon↩

3 *See* https://github.com/intlify/vue-i18n-next↩

Chapter 11 - Localization and Internationalization - Language Localization - Part 2

1 [https://github.com/yammadev/flag-icons↩](https://github.com/yammadev/flag-icons)

2 [https://github.com/damianof/large-scale-apps-my-vue3-project↩](https://github.com/damianof/large-scale-apps-my-vue3-project)

Chapter 13 - Localization and Internationalization - Number and DateTime Formats

1 https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Intl↩

Chapter 15 - Using CSS/SASS/SCSS Libraries

1 https://github.com/damianof/large-scale-apps-my-vue3-project↩

2 https://github.com/damianof/large-scale-apps-my-vue3-project↩

3 https://github.com/damianof/large-scale-apps-my-vue3-project↩

Chapter 16 - Creating Stand-Alone Components

1 *"Barrels" is the names of the index.ts files that just export everything from one folder*↩

2 *You could get away by doing the same thing we are doing by just using Actions. But we wanted to follow a clean pattern similar to Vuex for better encapsulation and best practice as it offers many advantages*↩

3 https://github.com/damianof/large-scale-apps-my-vue3-project↩

4 *You will have to add an import at the top of the Config.interface.ts code like this:*\↩

The Vue.config.js File

1 [https://github.com/damianof/large-scale-apps-my-vue3-project]↩