Orientation

In this Vue Mastery course, we'll be building a production-level app using Vue 3. We'll start off by creating the project using the Vue CLI. Then we'll learn about single file .vue components and how they can be used to create a single page application. We'll cover the fundamentals of Vue Router so we can navigate between the different views of our app, and we'll even fetch real external data using API calls with Axios. We'll end by learning about the build process and how to deploy our app into production.

If you don't know the fundamentals of Vue.js syntax, you'll want to first take our Intro to Vue 3 course.

If you're ready to build a Real World Vue 3 app, I'll see you in the rest of the course.

Vue CLI - Creating the Project

In this tutorial, we'll create our project using the Vue CLI. We'll then look at the Vue UI, a graphical user interface for managing our project. We'll end by touring the project that the CLI generates for us to get comfortable working within these files and folders.

Why a CLI?

As you probably know, CLI stands for Command Line Interface, and the Vue CLI provides a full system for rapid Vue.js development. This means it does a lot of tedious work for us and provides us with valuable features out-of-the-box.

It allows us to select which libraries our project will be using

Then it automatically plugs them into the project.

It Configures Webpack

When we build our app with Webpack, all of our JavaScript files, our CSS, and our dependencies get properly bundled together, minified and optimized.

It allows us to write our HTML, CSS & JavaScript however we like

We can use single-file .vue components, TypeScript, SCSS, Pug, the latest versions of ECMAScript, etc.

It enables Hot Module Replacement (HMR)

When you save your project, changes appear instantly in the browser.

Installing the CLI

In order to use the CLI, you'll need to have Node.js version 8.9 or above (v10+ recommended).

To install the CLI, run this command in your terminal:

```
npm i -g @vue/cli
# OR
yarn global add @vue/cli
```

Once it is installed, you'll have access to the `vue` binary in your command line. We'll use this to create our project.

Creating a Vue project

There are two ways we can create our project. With the Vue UI, or directly from the command line, which we'll do now by running this command in our terminal:

```
vue create real-world-vue
```

This command will start the creation of a Vue project, with the name of 'real-world-vue'.

We'll then be prompted with the option to pick a default preset or to manually select features. Using the down arrow key, we'll highlight **Manually select features**, then hit enter.

We'll then be presented with a list of feature options. Using the down arrow key, we'll move down and use the spacebar to select **Router**, **Vuex** and **Linter / Formatter**, then hit enter.

Next up, we'll select the version of Vue.js that we want to use. Of course, we'll select Vue 3 here.

Then we'll say **Y** (yes) to using History mode for Vue Router.

We'll then be asked to choose a Linter / Formatter, which is entirely up to you. I'll go ahead and choose **ESLint + Prettier** and tell it to **Lint on save**.

And for the sake of this course, I'll choose to have dedicated config files, but we could also keep them in package.json. Again, this is totally up to you.

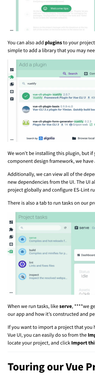
We have the option to save all of these settings as a preset. I'll choose not to with **n**.

If you'd like to save this as a preset, however, it will be stored in a JSON file named `.vuerc` in your user home directory.

When we hit enter, our project will be created automatically.

Serving our Project

Once our project is done being created, we can `cd` into it. In order to view it live in our browser, we'll run the command `npm run serve`, which compiles the app and serves it live at a local host.



Above is our app, running live in the browser. It already has two pages, the **Home** page and the **About** page, which we can navigate between because it's using Vue Router.

Vue UI

Now that we understand how to create a Vue project from the command line, let's repeat this same process but with the Vue UI instead, which is an intuitive visual way to manage our Vue projects.

Since we already have access to the `vue` binary, we can type `vue ui` in our terminal, which will start up the Vue UI in our browser.

To create a project in here, we'd click the **Create** tab, select the location where we want to save our project, then click **Create a new project here**. This will guide us through all of the project configuration steps we just went through from the terminal.

Vue UI Features

From the UI dashboard, we can do things such as monitor for plugin and dependency updates for our project and perform vulnerability checks.

You can also add **plugins** to your project from the Vue UI, which makes it very simple to add a library that you may need, such as Vuetify.

We won't be installing this plugin, but if you're interested in learning about this component design framework, we have an entire Vuetify course.

Additionally, we can view all of the dependencies our project is using and add new dependencies from here. The UI also has a tab for us to configure the project globally and configure ES-Lint rules and more.

There is also a tab to run tasks on our project.

When we run tasks, like **serve**, ******** we get a lot of helpful visual feedback about our app and how it's constructed and performing.

If you want to import a project that you hadn't originally created from within the Vue UI, you can easily do so by the **Import** tab of the Project Manager. Just locate your project, and click **import this folder**.

Touring our Vue Project

Now that we know how to create our project from the terminal and also from the UI, let's take a look at the project that was created for us.

The **node_modules** directory is where all of the libraries we need to build Vue are stored.

In the **public** directory, you can place any static assets you don't want to be run through Webpack when the project is built.

The **src** directory is where you'll spend most of your time since it houses all of the application code.

You'll want to put the majority of your assets, such as images and fonts, in the **assets** directory so they can be optimized by Webpack.

The **components** directory is where we store the components, or building blocks, of our Vue app.

The **router** folder is used for Vue Router, which enables our site navigation. We use Vue Router to pull up the different "views" of our single page application.

The **store** is where we put Vuex code, which handles state management throughout the app. By the end of this course, you'll have a basic understanding of what Vuex is for, but we won't be implementing any Vuex code. This course serves as a foundational course that prepares you for Vuex course.

The **views** directory is where we store component files for the different views of our app, which Vue Router loads up.

The **App.vue** file is the root component that all other components are nested within.

The **main.js** file is what renders our **App.vue** component (and everything nested within it) and mounts it to the DOM.

Finally, we have a **.gitignore** file where we can specify what we want git to ignore, along with a **babel.config.js** file and our **package.json**, which helps npm identify the project and handle its dependencies, and a **README.md**.

How the App is Loaded

You might be wondering now, how is the app being loaded? Let's take a look at that process.

📄src/main.js

```
import { createApp } from "vue";
import App from "./App.vue";
import router from "./router";
import store from "./store";

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

In our **main.js** file, we're importing the `createApp` method from Vue, along with our **App.js** component. We're then running that method, feeding in the `App` (the root component that includes all of our application code since all other components are nested within it).

As the method name explicitly states, this creates the app and we're telling it to `use` the `Router` and `Store` that are imported above. Finally, the app is mounted to the DOM via the `mount` method, which takes in an argument to specify where in the DOM the app should be mounted. But where exactly is this of `#app`?

If we peek inside our **index.html** file, we can see there's a div with the id of "app", with a helpful comment below it.

📄public/index.html

```
<div id="app">
  <!-- built files will be auto injected -->
</div>
```

Ah. So this is where our Vue app is being mounted. Later, we'll gain a deeper understanding of how this **index.html** serves as the "single page" of our single page application.

Putting it all together

Let's take a look at this process more visually:

Wrapping Up

You should now have an understanding of how we can create a Vue project and how to manage it from the Vue UI. We also explored the project that was created for us to get ready to start customizing this project. In the next lesson, we'll build our first single file vue component.