Web Application Development

Produced by

David Drohan (ddrohan@wit.ie)

Department of Computing & Mathematics
Waterford Institute of Technology

http://www.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Vue.js

PART 3

APP ROUTING WITH VUE-ROUTER

Vue.js

Overall Section Outline

1. **Introduction** – Why you should be using VueJS
2. **Terminology & Overview** – The critical foundation for understanding
3. **Declarative Rendering & Reactivity** – Keeping track of changes (Data Binding)
4. **Components** – Reusable functionality (Templates, Props & Slots)
5. **Routing** – Navigating the view (vue-router)
6. **Directives**– Extending HTML
7. **Event Handling** – Dealing with User Interaction
8. **Filters** – Changing the way we see things
9. **Computed Properties & Watchers** – Reacting to Data Change
10. **Transitioning Effects** – I like your <style>
11. **Case Study** – Labs in action

Vue.js

Overall Section Outline

1. **Introduction** – Why you should be using VueJS
2. **Terminology & Overview** – The critical foundation for understanding
3. **Declarative Rendering & Reactivity** – Keeping track of changes (Data Binding)
4. **Components** – Reusable functionality (Templates, Props & Slots)
5. **Routing – Navigating the view (vue-router)**
6. **Directives**– Extending HTML
7. **Event Handling** – Dealing with User Interaction
8. **Filters** – Changing the way we see things
9. **Computed Properties & Watchers** – Reacting to Data Change
10. **Transitioning Effects** – I like your <style>
11. **Case Study – Labs in action**

Routing with vue-router

NAVIGATING THE VIEW

Introduction - Recap

As previously mentioned**, Routing** is a key part of all websites and **web applications** in one way or another. It plays a central role in static HTML pages as well as in the most complex **web applications**. **Routing** comes into play whenever you want to use a URL in your **application**.

In **Vue**, for most Single Page Applications, it's recommended to use the officially-supported **vue-router library** – which is what we'll be doing.

It is often more convenient to identify a route with a name, especially when linking to a route or performing navigations. You can give a route a name in the routes options while creating the Router instance:

(More on this later)

index.js

```
Vue.use(Router)

export default new Router({
  routes: [
    {
      path: '/',
      name: 'Home',
      component: Home
    },
    {
      path: '/donations',
      name: 'Donations',
      component: Donations
    },
    {name: 'Donate'...},
    {name: 'AboutUs'...},
    {name: 'ContactUs'...}
  ]
})
```

Routing in Depth

Vue is a great Javascript library that allows you to create some pretty cool, dynamic, front-end applications. Vue is also great for single page applications (SPA). SPAs work a little differently that your standard backend web application built in something like PHP. Instead of making requests to different routes on the backend and getting a fully rendered page as a response, a SPA does all the page rendering on the front-end and only sends requests to the server when new data is needed or needs to be refreshed or saved.

This can improve the responsiveness of your web application because you can render the pages once and display them dynamically based on the current context of the application. In order to make this work, you need a way to distinguish the different views or pages from each other. In SPAs this is done with a **router**. Luckily Vue has a fully supported first-party router library called **vue-router**.

vue-router

Vue Router is the official router for Vue.js. It deeply integrates with Vue.js core to make building Single Page Applications with Vue.js very straightforward. Features include:

- Nested route/view mapping
- Modular, component-based router configuration
- Route params, query, wildcards
- View transition effects powered by Vue.js' transition system
- Fine-grained navigation control
- Links with automatic active CSS classes
- HTML5 history mode or hash mode, with auto-fallback in IE9
- Customizable Scroll Behavior

Vue.js

Getting Started with `vue-router`

Creating a Single-page Application with Vue + Vue Router is very simple. With Vue.js, we are already composing our application with components. When adding Vue Router to the mix, all we need to do is map our components to the routes and let Vue Router know where to render them. Here's a basic example:

HTML

```html
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<script src="https://unpkg.com/vue-router/dist/vue-router.js"></script>

<div id="app">
  <h1>Hello App!</h1>
  <p>
    <!-- use router-link component for navigation. -->
    <!-- specify the link by passing the `to` prop. -->
    <!-- `<router-link>` will be rendered as an `<a>` tag by default -->
    <router-link to="/foo">Go to Foo</router-link>
    <router-link to="/bar">Go to Bar</router-link>
  </p>
  <!-- route outlet -->
  <!-- component matched by the route will render here -->
  <router-view></router-view>
</div>
```

Vue.js

Getting Started with `vue-router`

Creating a Single-page Application with Vue + Vue Router is very simple. With Vue.js, we are already composing our application with components. When adding Vue Router to the mix, all we need to do is map our components to the routes and let Vue Router know where to render them. Here's a basic example:

JavaScript

```
// 1. Define route components.
// These can be imported from other files
const Foo = { template: '<div>foo</div>' }
const Bar = { template: '<div>bar</div>' }

// 2. Define some routes
// Each route should map to a component. The "component" can
// either be an actual component constructor created via
// `Vue.extend()`, or just a component options object.
// We'll talk about nested routes later.
const routes = [
  { path: '/foo', component: Foo },
  { path: '/bar', component: Bar }
]
```

Vue.js

Getting Started with `vue-router`

Creating a Single-page Application with Vue + Vue Router is very simple. With Vue.js, we are already composing our application with components. When adding Vue Router to the mix, all we need to do is map our components to the routes and let Vue Router know where to render them. Here's a basic example:

JavaScript

```
// 3. Create the router instance and pass the `routes` option
// You can pass in additional options here, but let's
// keep it simple for now.
const router = new VueRouter({
  routes // short for `routes: routes`
})

// 4. Create and mount the root instance.
// Make sure to inject the router with the router option to make the
// whole app router-aware.
const app = new Vue({
  router
}).$mount('#app')

// Now the app has started!
```

Vue.js

Getting Started with `vue-router`

By injecting the router, we get access to it as **`this.$router`** as well as the current route as **`this.$route`** inside of any component:

Keep in mind that **`this.$router`** is exactly the same as using **`router`**.

The reason we use **`this.$router`** is because we don't want to import the router in every single component that needs to manipulate routing.

JavaScript

```
// Home.vue
export default {
  computed: {
    username () {
      // We will see what `params` is shortly
      return this.$route.params.username
    }
  },
  methods: {
    goBack () {
      window.history.length > 1
        ? this.$router.go(-1)
        : this.$router.push('/')
    }
  }
}
```

Vue.js

The Route Object

A **route object** represents the state of the current active route. It contains parsed information of the current URL and the **route records** matched by the URL.

The route object is immutable. Every successful navigation will result in a fresh route object.

The route object can be found in multiple places:

- Inside components as **this.$route**
- Inside **$route** watcher callbacks
- As the return value of calling **router.match(location)**
- Inside **navigation guards** as the first two arguments:
- Inside the scrollBehavior function as the first two arguments:

```
router.beforeEach((to, from, next) => {
  // `to` and `from` are both route objects
})
```

```
const router = new VueRouter({
  scrollBehavior (to, from, savedPosition) {
    // `to` and `from` are both route objects
  }
})
```

The Route Object

Route Object Properties

- **$route.path**
  - type: `string`

    A string that equals the path of the current route, always resolved as an absolute path. e.g. `"/foo/bar"`.

- **$route.params**
  - type: `Object`

    An object that contains key/value pairs of dynamic segments and star segments. If there are no params the value will be an empty object.

The Route Object

Route Object Properties

- **$route.query**
  - type: `Object`

    An object that contains key/value pairs of the query string. For example, for a path `/foo?user=1`, we get `$route.query.user == 1`. If there is no query the value will be an empty object.

- **$route.hash**
  - type: `string`

    The hash of the current route (with the `#`), if it has one. If no hash is present the value will be an empty string.

Vue.js

The Route Object

Route Object Properties

- **$route.fullPath**
  - type: `string`

    The full resolved URL including query and hash.

- **$route.matched**
  - type: `Array<RouteRecord>`

    An Array containing **route records** for all nested path segments of the current route. Route records are the copies of the objects in the `routes` configuration Array (and in `children` Arrays):

The Route Object

Route Object Properties

- **$route.name**

  The name of the current route, if it has one. (See Named Routes)

- **$route.redirectedFrom**

  The name of the route being redirected from, if there were one. (See Redirect and Alias)

Vue.js

Named Routes (what we use)

Sometimes it is more convenient to identify a route with a name, especially when linking to a route or performing navigations. You can give a route a name in the routes options while creating the Router instance:

```js
const router = new VueRouter({
  routes: [
    {
      path: '/user/:userId',
      name: 'user',
      component: User
    }
  ]
})
```

Vue.js

Named Routes

To link to a named route, you can pass an object to the router-link component's to prop:

```html
<router-link :to="{ name: 'user', params: { userId: 123 }}">User</router-link>
```

This is the exact same object used programmatically with router.push():

```js
router.push({ name: 'user', params: { userId: 123 }})
```

In both cases, the router will navigate to the path /user/123.

Vue.js

Named Views

Sometimes you need to display multiple views at the same time instead of nesting them, e.g. creating a layout with a sidebar view and a main view. This is where **named views** come in handy. Instead of having one single outlet in your view, you can have multiple and give each of them a name. A **router-view** without a name will be given **default** as its name.

```html
<router-view class="view one"></router-view>
<router-view class="view two" name="a"></router-view>
<router-view class="view three" name="b"></router-view>
```

Vue.js

Named Views

A view is rendered by using a component, therefore multiple views require multiple components for the same route. Make sure to use the **components** (with an s) option:

```js
const router = new VueRouter({
  routes: [
    {
      path: '/',
      components: {
        default: Foo,
        a: Bar,
        b: Baz
      }
    }
  ]
})
```

Vue.js

Route Meta Fields & Records

You can also include a meta field when defining a route:

```js
const router = new VueRouter({
  routes: [
    {
      path: '/foo',
      component: Foo,
      children: [
        {
          path: 'bar',
          component: Bar,
          // a meta field
          meta: { requiresAuth: true }
        }
      ]
    }
  ]
})
```

Route Meta Fields & Records

First, each route object in the routes configuration is called a **route record**. Route records may be nested. Therefore when a route is matched, it can potentially match more than one route record.

For example, with the previous route config, the URL **/foo/bar** will match both the parent route record and the child route record.

All route records matched by a route are exposed on the $route object (and also route objects in navigation guards) as the $route.matched Array. Therefore, we will need to iterate over $route.matched to check for meta fields in route records.

An example use case is checking for a meta field in the global **navigation guard**:

Vue.js

Route Meta Fields & Records

```js
router.beforeEach((to, from, next) => {
  if (to.matched.some(record => record.meta.requiresAuth)) {
    // this route requires auth, check if logged in
    // if not, redirect to login page.
    if (!auth.loggedIn()) {
      next({
        path: '/login',
        query: { redirect: to.fullPath }
      })
    } else {
      next()
    }
  } else {
    next() // make sure to always call next()!
  }
})
```

Redirect and Alias

Redirecting is also done in the **routes** configuration. To redirect from **/a** to **/b**:

```js
const router = new VueRouter({
  routes: [
    { path: '/a', redirect: '/b' }
  ]
})
```

The redirect can also be targeting a named route:

```js
const router = new VueRouter({
  routes: [
    { path: '/a', redirect: { name: 'foo' }}
  ]
})
```

Vue.js

Redirect and Alias

Or even use a function for dynamic redirecting:

```js
const router = new VueRouter({
  routes: [
    { path: '/a', redirect: to => {
      // the function receives the target route as the argument
      // return redirect path/location here.
    }}
  ]
})
```

Note that **Navigation Guards** are not applied on the route that redirects, only on its target. In the example above, adding a **beforeEnter** or **beforeLeave** guard to the **/a** route would have no effect.

Redirect and Alias

A redirect means when the user visits **/a**, the URL will be replaced by **/b**, and then matched as **/b**. But what is an alias?

An alias of /a as /b means when the user visits /b, the URL remains /b, but it will be matched as if the user is visiting /a.

The above can be expressed in the route configuration as:

```js
const router = new VueRouter({
  routes: [
    { path: '/a', component: A, alias: '/b' }
  ]
})
```

An alias gives you the freedom to map a UI structure to an arbitrary URL, instead of being constrained by the configuration's nesting structure.

Other Features worth Mentioning

- Transitions
- Data Fetching
- Scrolling Behaviour
- Lazy Loading Routes

Case Study

LABS IN ACTION

Analysing our Case Study

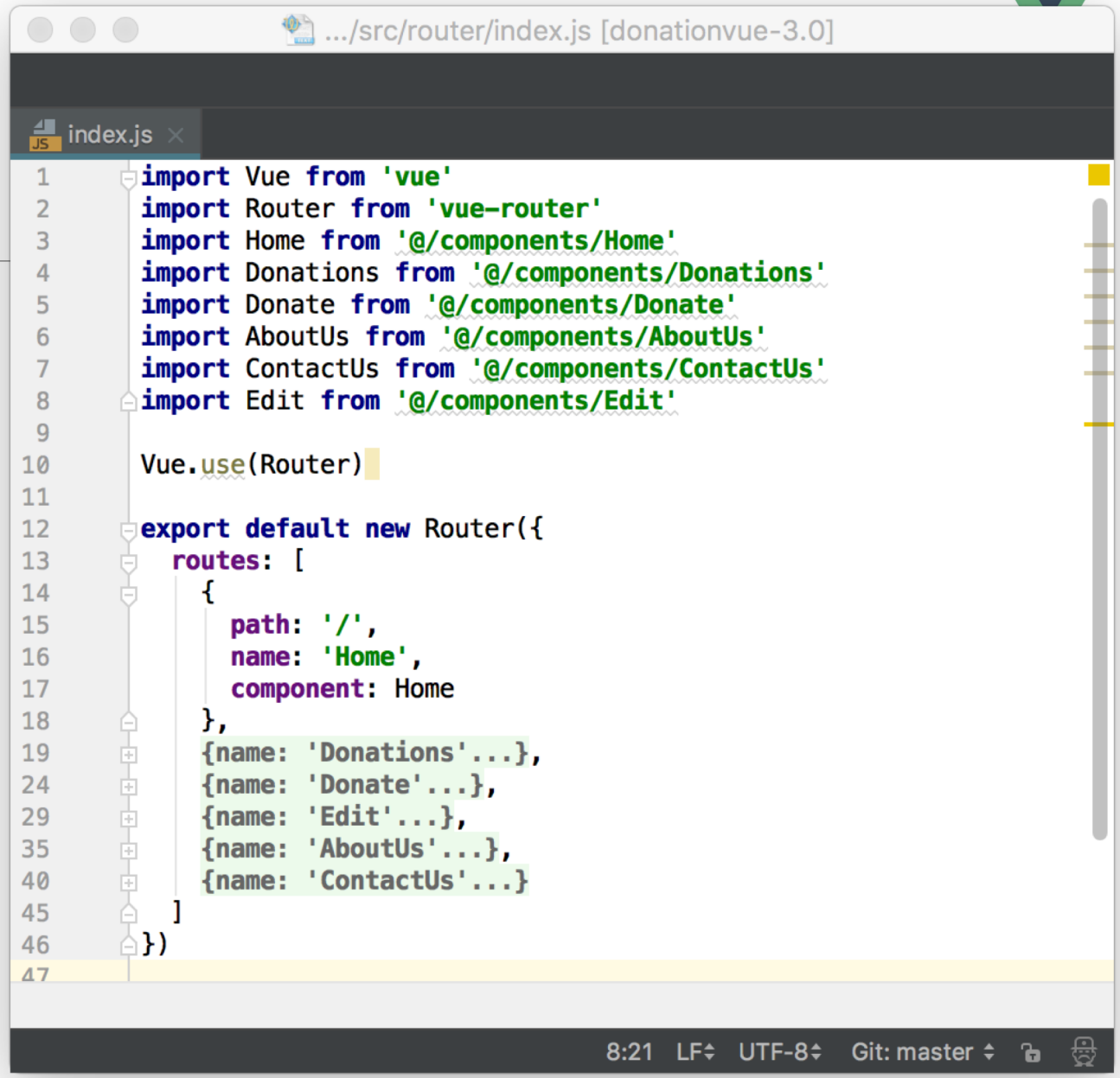
So now that we've covered some more detail about Routing and using **`vue-router`** to manage navigation in a Vue web app, let's take a closer look at how we us it in **DonationVue**.

The main file of note is **/router/index.js** so we'll go through that in the next few slides.

router/index.js

Here we have a fairly basic implementation of using **Named Routes** to manage our app navigation.

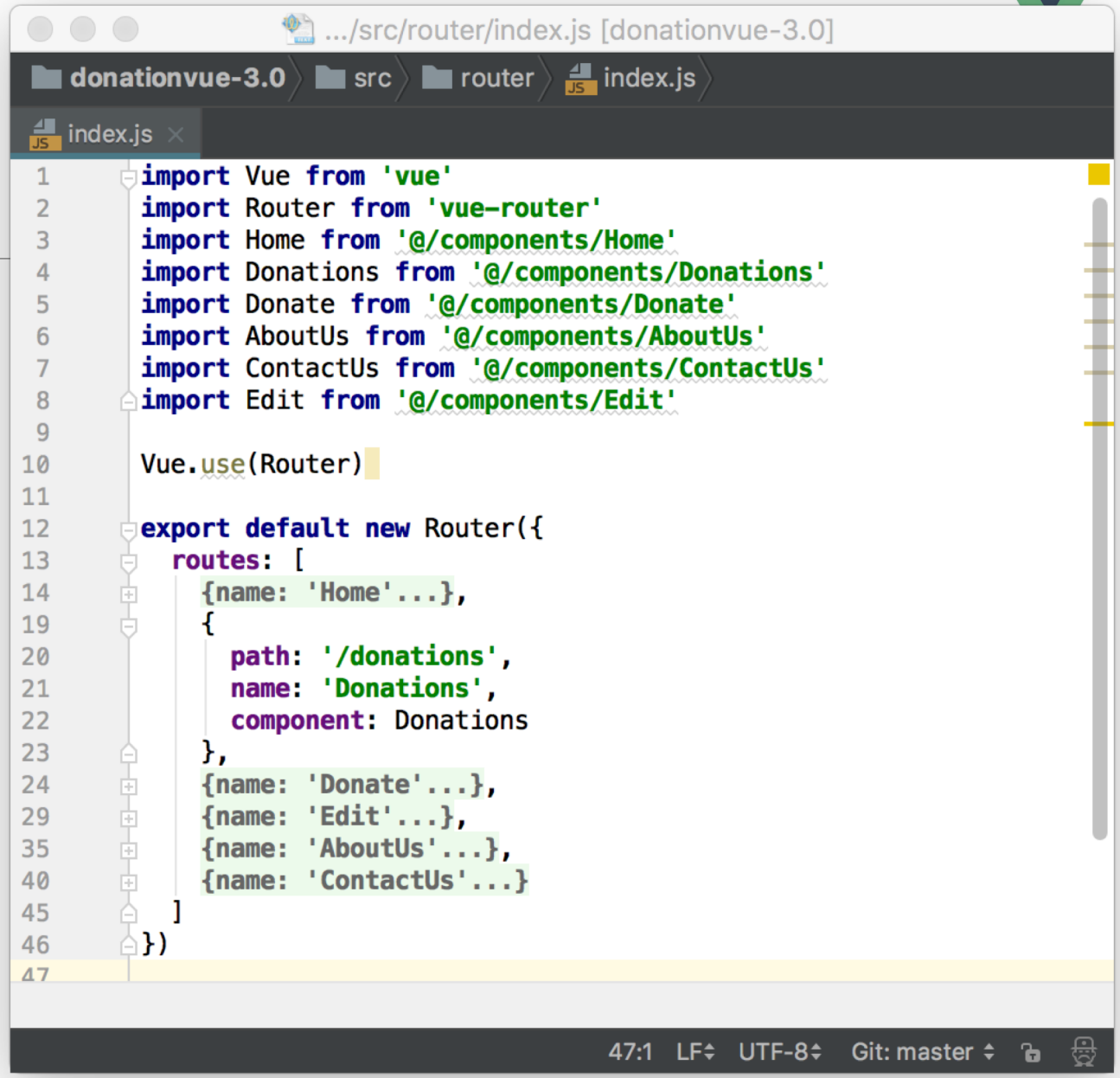
The big advantage of keeping all this in a separate file is that when we need to add a new route, we simply import the component and add a new route to our list of routes



router/index.js

Here we have a fairly basic implementation of using **Named Routes** to manage our app navigation.

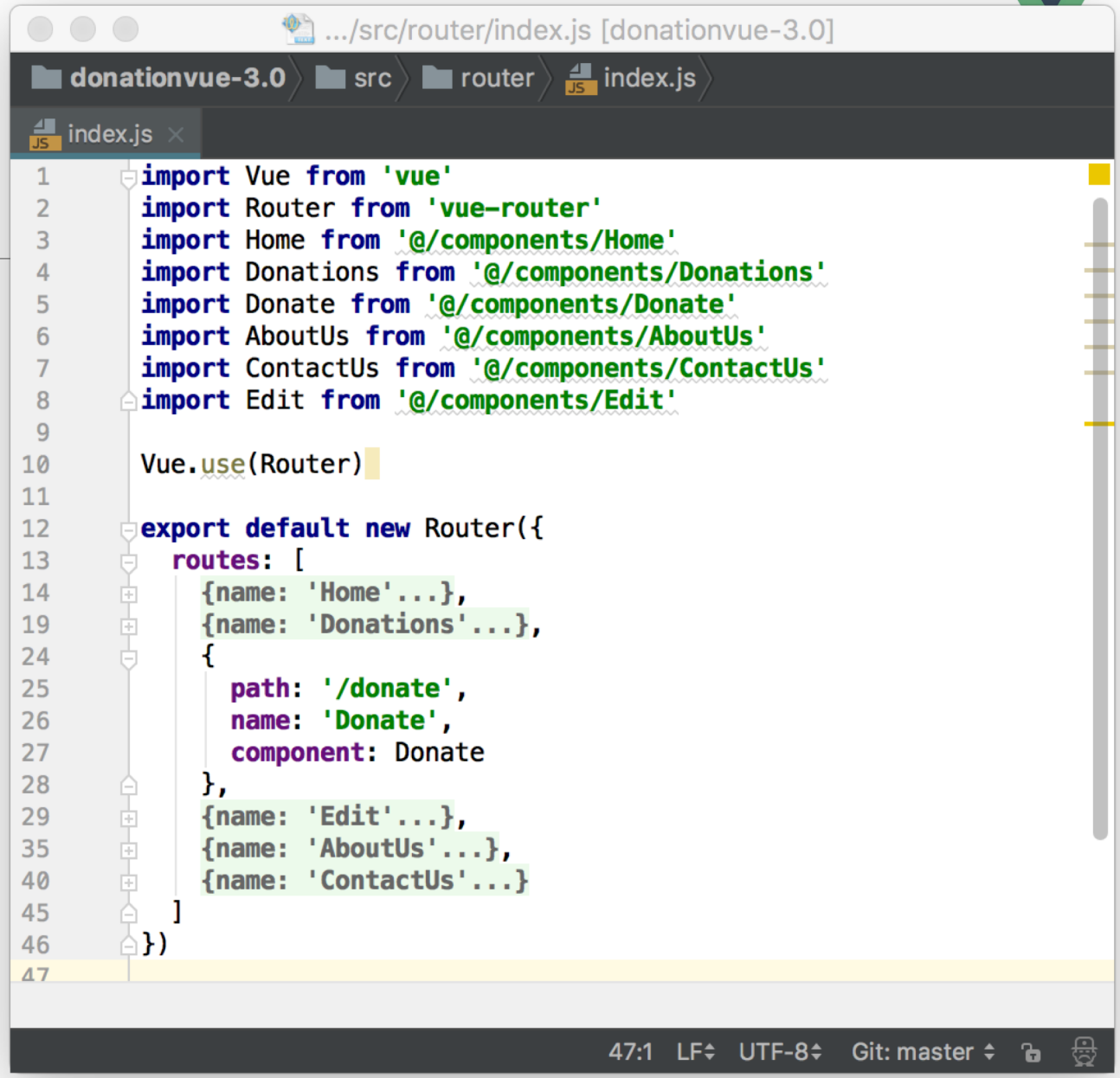
The big advantage of keeping all this in a separate file is that when we need to add a new route, we simply import the component and add a new route to our list of routes



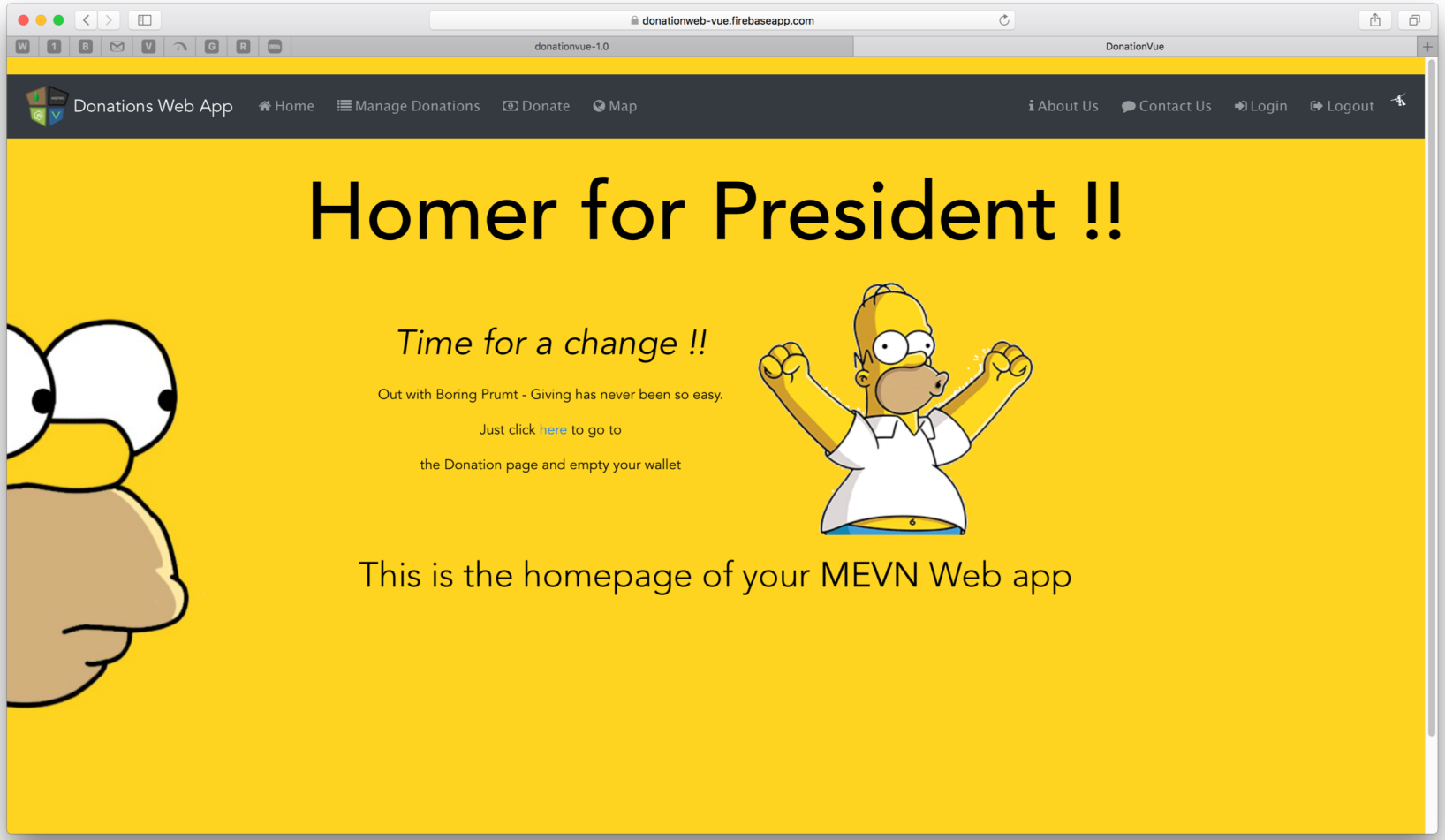
router/index.js

Here we have a fairly basic implementation of using **Named Routes** to manage our app navigation.

The big advantage of keeping all this in a separate file is that when we need to add a new route, we simply import the component and add a new route to our list of routes



Demo Application https://donationweb-vue.firebaseapp.com



References

- https://vuejs.org
- https://router.vuejs.org/guide
- https://scotch.io/tutorials/getting-started-with-vue-router

Questions?