Week 5: Vue.js

Last week we finished our back-end, and we moved on to the front-end.
We installed Vue.js, and created our front-end project.

Now, we will continue from last week onto:

0: Try running the app first

1. run the project by using `npm run serve` (slightly different from our back-end's command)
2. if you see a green message saying

```
    DONE Compiled successfully
```

everything is good to go! 😃

if not, please download the code from github, open it and run `npm install`, and `npm run serve`. If this doesn't work, please message me on slack and we can figure it out.

1. Installing a CSS library: Bulma

Now, we were introduced to CSS, but this is not a CSS course, and most people use third-party CSS libraries for their projects anyway. So this works out well.

The most commonly used one is: Bootstrap
However, Bulma is a nicer one to use for this course.

we install it by:

`> npm install --save bulma`

and in `main.js` add:

`import 'bulma' from 'bulma/css/bulma.min/css'`

Now, let's delete all css parts in:

1. `App.vue`
2. `HelloWorld.vue`

We do this, since we are using our Bulma.

You can see how nice Bulma is if you comment out:
`// import 'bulma' from 'bulma/css/bulma.min/css'`

Then, our page looks very old. So, we uncomment it to bring it back to modern age.

2. Creating our first Vue Component: Hero with Navbar

0. in `App.vue`, please add the following:

```
<style>
  .content-section {
    margin-top: -3.25rem;
    padding-top: 3.25rem;
  }
</style>
```

1. Let's create one of the main components called: `Home.vue` in `/components` folder.

```
<template>
    <div class="content-section">
        <section class="hero is-white is-fullheight-with-navbar">
            <div class="hero-body">
                <div class="container">
                    <h1 class="title
                    has-text-centered
                    is-unselectable">Backlogger</h1>
                    <h2 class="subtitle
                    has-text-centered
                    is-unselectable">keep your backlog organized</h2>
                </div>
            </div>
        </section>
    </div>
</template>
```

In Vue.js, all your HTML code must be in the element called: `<template>`

all your CSS code in the element called:
`<style>`

and, JavaScript in `<script>`.

Actually, the last two are not as suprising as that's where you would usually keep your CSS and JavaScript in normal HTML code.

2. Let's add a header to our page. Create a new file in the `/components` folder, called `Navbar.vue`

```
<template>
    <nav class="navbar is-transparent">
        <div class="navbar-brand">
            <div class="navbar-burger burger">
                <span></span>
                <span></span>
                <span></span>
            </div>
        </div>
        <div class="navbar-menu">
            <div class="navbar-start">
                <a class="navbar-item">Home</a>
                <a class="navbar-item">Dashboard</a>
                <a class="navbar-item">Add an Item</a>
            </div>
        </div>
    </nav>
</template>
```

But, now let's add some interactability.
Start by adding, the second part:

```
<script>
    export default {
        data: function() {
            return {

            }
        }
    }
</script>
```

Aside:
`data` would make sense to be a JavaScript Object conceptually, however, due to another conceptual restriction is why it should be a function that returns a JavaScript Object. The reason for that is to avoid having multiple instances of this component sharing the same data object.

e.g.

```
    <Navbar/>
    <Navbar/>
```

This would have a same `data` object which would have changes reflect in both instances, even if another class should not do that change.
Thus, the use-function solution makes all our instances of a same component exist independently of each other.

```
<template>
    <nav class="navbar is-transparent">
        <div class="navbar-brand">
            <div class="navbar-burger burger"
                :class="{'is-active':activeNavbar}"
                @click="toggleStatus">
                <span></span>
                <span></span>
                <span></span>
            </div>
        </div>
        <div class="navbar-menu"
            :class="{'is-active':activeNavbar}"
            @click="toggleStatus">
            <div class="navbar-start has-dropdown">
                <a class="navbar-item is-unselectable"
                   :class="{'is-active': itemIndex == 0}"
                    @click="toggleActiveItem(0)">Home</a>
                <a class="navbar-item is-unselectable"
                   :class="{'is-active': itemIndex == 1}"
                    @click="toggleActiveItem(1)">Dashboard</a>
                <a class="navbar-item is-unselectable"
                   :class="{'is-active': itemIndex == 2}"
                    @click="toggleActiveItem(2)">Add an Item</a>
            </div>
        </div>
    </nav>
</template>
<script>
export default {
    data: function() {
        return {
            activeNavbar: false,
            itemIndex: 0
        }
    },
    methods: {
        toggleStatus: function() {
            this.activeNavbar = !this.activeNavbar
        },
        toggleActiveItem: function(index) {
            this.itemIndex = index
        }
    }
}
</script>
```

Lastly, the action of giving indices seems very repetitive. What if we add a for-loop?

We do so like so:

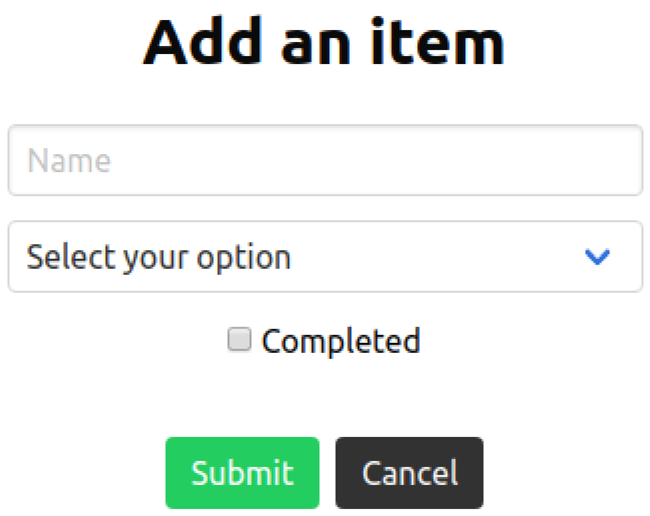
```
<template>
    <nav class="navbar is-fixed-top is-transparent">
        <div class="navbar-brand">
            <div class="navbar-burger burger"
                :class="{'is-active':activeNavbar}"
                @click="toggleStatus">
                <span></span>
                <span></span>
                <span></span>
            </div>
        </div>
        <div class="navbar-menu" :class="{'is-active':activeNavbar}"
            @click="toggleStatus">
            <div class="navbar-start has-dropdown">
                <a v-for="(option, index) in OPTIONS" :key="index"
                    class="navbar-item is-unselectable"
                    :class="{'is-active': optionIndex === index}"
                    @click="toggleActiveOption(index)"
                >{{ option }}</a>
            </div>
        </div>
    </nav>
</template>
<script>
export default {
    data: function() {
        return {
            OPTIONS: ['Home', 'Dashboard', 'Add an Item'],
            activeNavbar: false,
            optionIndex: 0
        }
    },
    methods: {
        toggleStatus: function() {
            this.activeNavbar = !this.activeNavbar
        },
        toggleActiveOption: function(index) {
            this.optionIndex = index
        }
    }
}
</script>
```

To summerize this section:

- we learned how to define components
- how to use `data` property
- how to use `methods` property
- how to use templating with {{ text }}
- how to use onClick events
- how to provide conditional classes

3. Defining Add an Item Component

Here is the final version:



Now define this component, we need to consider, will or can this component be reused in the future.
And the answer is yes. We can use reuse the same component when we are editing an existing item. So, it would be a good idea to make a very general form, and add it to the component.

So, let's make the form in `ItemForm.vue`

```
<template>
    <div class="container">
        <div class="tile is-ancestor">
            <div class="tile is-parent">
                <article class="tile is-child">
                    <div class="columns">
                        <form class="column"
                        :class="{'is-one-quarter is-offset-5': !isModal}">
                            <h5 class="title has-text-centered is-unselectable">
                            {{ title }}</h5>
                            <div class="field">
                                <div class="control">
                                    <input
                                        class="input"
                                        type="text"
                                        placeholder="Name"
                                        autoComplete="false"
                                        v-model="name.value"
                                        @blur="checkValidness('name')"
                                    />
                                </div>
                                <span v-if="!name.pristine">
                                    <p v-if="!name.value" class="help is-danger">
                                    name is required</p>
                                </span>
                            </div>
                            <div class="field">
                                <div class="select">
                                    <select
                                        v-model="type.value"
                                        @change="checkValidness('type')">
                                        <option value="" disabled selected>
                                        Select your option</option>
                                        <option
                                        v-for="(type, index) in types"
                                        :key="index">{{ type }}</option>
                                    </select>
                                </div>
                                <span v-if="!type.pristine">
                                    <p v-if="!type.value" class="help is-danger">
                                    type is required</p>
                                </span>
                            </div>
                            <div class="field has-text-centered" v-if="!isNew">
                                <label class="checkbox">
                                    <input type="checkbox" v-model="completed"/>
                                    Completed
                                </label>
                            </div>
                            <br/>
                            <div class="field">
                                <div class="control buttons is-centered">
                                    <button type="submit" class="button is-success">
                                    Submit</button>
                                    <button type="submit" class="button is-dark">
                                    Cancel</button>
                                </div>
                            </div>
                        </form>
                    </div>
                </article>
            </div>
        </div>
    </div>
</template>
<script>
import { ItemRuleSet, types } from './../rulesets/ItemRuleset'
export default {
    data: () => ({
        name: {
            value: '',
            pristine: true,
            valid: true
        },
        type: {
            value: '',
            pristine: true,
            valid: true
        },
        completed: false,
        types
    }),
    props: {
        item: Object,
        title: String,
        isNew: Boolean,
        isModal: {
            type: Boolean,
            default: false
        }
    },
    methods: {
        checkValidness(field) {
            const value = this[field].value
            const ruleset = ItemRuleSet({ [field]: value })

            this[field] = {
                valid: ruleset[field].valid,
                pristine: false,
                value: value
            }
        }
    }
}
</script>
<style lang="css">
    .select select:not([multiple]) {
        width: 100%
    }
    .select:not(.is-multiple) {
        width: 100%
    }
</style>
```

Now, we need to define what `types` are, and `ItemRuleSet`, we can define it it in the `/ruleset` directory in the `ItemRuleSet.js` file

Here's the code:

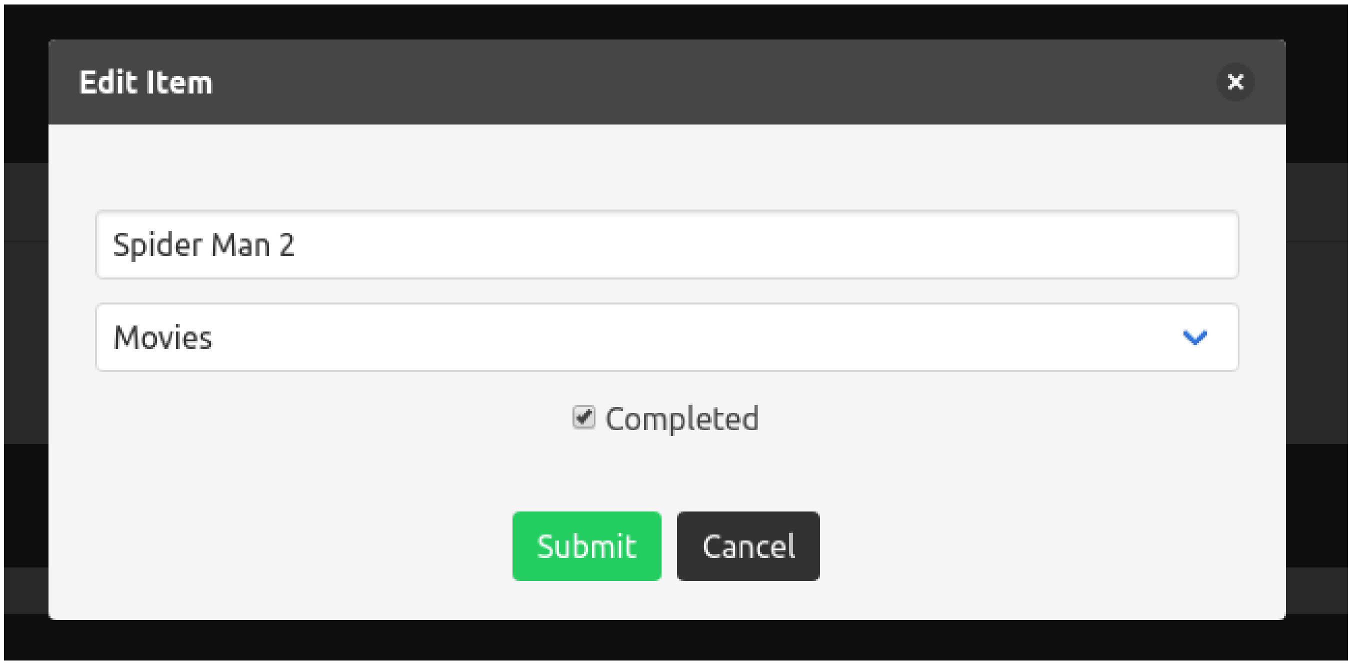
```
export const ItemRuleSet = function (object) {
    const { name, type } = object
    const ruleset = {
        'name': {
            valid: name && name.trim().length
        },
        'type': {
            valid: type && type.trim().length
        },
    }
    return ruleset
}
export const ItemRequiredFields = ['name', 'type']
export const types = ['Movies', 'TV Shows', 'Video Games', 'Books']
```

Now, our component can be defined for `AddItem.vue`

```
<template>
    <section class="hero is-fullheight-with-navbar is-white">
        <div class="hero-body">
            <ItemForm title="Add a new item" isNew="false" />
        </div>
    </section>
</template>
<script>
import ItemForm from './ItemForm'
export default {
    components: {
        ItemForm
    }
}
</script>
```

4. Defining the item modal

This should be the final result:



Now that we defined a form, defining a modal is really easy from now

so, let's make a new file called `ItemModal.vue`

```
<template>
    <section class="section" v-if="!!item && active">
        <div class="modal is-active">
            <div class="modal-background"></div>
            <div class="modal-content">
                <article class="message">
                    <div class="message-header">
                        <p>Edit Item</p>
                        <button class="delete" aria-label="delete"></button>
                    </div>
                    <div class="message-body has-text-centered">
                        <ItemForm
                            :item="item"
                            :isModal="true"
                        />
                    </div>
                </article>
            </div>
        </div>
    </section>
</template>
<script>
import ItemForm from './ItemForm'
export default {
    props: {
        item: Object,
        active: {
            type: Boolean,
            required: true
        }
    },
    components: {
        ItemForm
    }
}
</script>
```

5. Defining the Dashboard Component

To define `Dashboard` Component, we need to define a couple of components that will make it up:

1. `DeleteModal`
2. `ItemModal`

3. `Item`

DeleteModal.vue

```
<template>
    <section class="section" v-if="!!item && active">
        <div class="modal is-active">
            <div class="modal-background"></div>
            <div class="modal-content">
                <article class="message">
                    <div class="message-header">
                        <p class="is-unselectable">Delete this item?</p>
                        <button
                            class="delete"
                            aria-label="delete"></button>
                    </div>
                    <div class="message-body has-text-centered">
                        <p>{{ item.name }}</p>
                        <br/>
                        <div class="buttons is-centered">
                            <a class="button is-danger is-unselectable">Submit</a>
                            <a class="button is-dark is-unselectable">Cancel</a>
                        </div>
                    </div>
                </article>
            </div>
        </div>
    </section>
</template>
<script>
export default {
    props: {
        item: Object,
        active: {
            type: Boolean,
            required: true
        }
    }
}
</script>
```

ItemModal.vue

```
<template>
    <section class="section" v-if="!!item && active">
        <div class="modal is-active">
            <div class="modal-background"></div>
            <div class="modal-content">
                <article class="message">
                    <div class="message-header">
                        <p>Edit Item</p>
                        <button class="delete" aria-label="delete"></button>
                    </div>
                    <div class="message-body has-text-centered">
                        <ItemForm
                            :item="item"
                            :isModal="true"
                        />
                    </div>
                </article>
            </div>
        </div>
    </section>
</template>
<script>
import ItemForm from './ItemForm'
export default {
    props: {
        item: Object,
        active: {
            type: Boolean,
            required: true
        }
    },
    components: {
        ItemForm
    }
}
</script>
```

Item.vue

```
<template>
    <div class="notification is-dark">
        <button class="delete"></button>
        <p class="has-text-centered">{{ name }}</p>
    </div>
</template>
<script>
export default {
    props: {
        name: {
            type: String,
            required: true
        }
    }
}
</script>
```

Now, we can define the `Dashboard` component

Dashboard.vue

```
<template>
    <div class="content-section">
        <section>
            <div class="hero is-dark is-medium">
                <div class="hero-body">
                    <div class="container" >
                        <h1 class="title has-text-centered is-unselectable">
                        Welcome!
                        </h1>
                        <h1 class="subtitle has-text-centered is-unselectable">
                        Your Dashboard
                        </h1>
                    </div>
                </div>
            </div>
            <div class="tabs is-centered">
                <ul v-if="types">
                    <li v-for="(type, index) in types"
                        :key="index"
                        :class="{ 'is-active' : selectedType === index }"
                        @click="pickType(index)">
                        <a>{{ type }}</a>
                    </li>
                </ul>
            </div>
            <div class="tabs is-small is-toggle is-centered has-text-centered">
                <ul v-if="filters">
                    <li v-for="(filter, index) in filters"
                        :key="index"
                        :class="{ 'is-active' : selectedFilter === index }"
                        @click="pickFilter(index)">
                        <a>{{ filter }}</a>
                    </li>
                </ul>
            </div>
        </section>
        <section class="section" v-if="items.length">
            <Item
                v-for="(item, index) in items"
                :name="item.name"
                :key="index" />
        </section>
        <section class="section" v-else>
            <div class="notification is-white">
                <p class="has-text-centered is-unselectable">
                No Items to Show
                </p>
            </div>
        </section>
        <DeleteModal
            :active="activeDeleteModal"
            :item="selectedItem"/>
        <ItemModal
            :active="activeItemModal"
            :item="selectedItem"/>
    </div>
</template>
<script>
import ItemModal from './ItemModal'
import DeleteModal from './DeleteModal'
import { TYPES, FILTERS } from './../rulesets/ItemRuleset'
import Item from './Item'

export default {
    data: () => ({
        items: [{
            name: 'spider-man',
            type: 'Movies',
            completed: true
        }, {
            name: 'stranger things',
            type: 'TV Shows',
            completed: false
        }],
        types: TYPES,
        filters: FILTERS,
        selectedFilter: 0,
        selectedType: 0
    }),
    methods: {
        pickType (index) {
            this.selectedType = index
        },
        pickFilter (index) {
            this.selectedFilter = index
        }
    },
    components: {
        DeleteModal,
        ItemModal,
        Item
    }
}
</script>
```

6: Adding Routing

Now, we have added all the components that we will mainly use in our application, so now let's add some routing.

Earlier, in the `Navbar` we defined the following options:

- Home
- Dashboard
- Add an Item

Now, the main goal for us is to be able to navigate between the components. If you recall, those components took on the whole page. So it would make sense to give them their own URLs.

We are going to be use the following routes:

- `/`
- `/dashboard`
- `/add-item`

However, Vue.js doesn't come with routing out of the box, so we will need to install it and configure it.

1. please run in your terminal

```
npm install vue-router --save
```

2. Then, we will configure Vue-Router in its own file, to keep it more organized. Create a folder and a file index.js, like so: `/router/index.js`
3. Paste the following code:

```
import Vue from 'vue'
import VueRouter from 'vue-router'
import Home from '../components/Home.vue'
import Dashboard from '../components/Dashboard.vue'
import AddItem from '../components/AddItem.vue'

Vue.use(VueRouter)

export default new VueRouter({
    mode: 'history',
    routes: [
        { path: '/', component: Home },
        { path: '/dashboard', component: Dashboard },
        { path: '/add-friend', component: AddFriend },
        { path: '/add-item', component: AddItem }
    ],
    linkActiveClass: 'is-active'  // needed because of Bulma
})
```

4. in `main.js` use this code:

```
import Vue from 'vue'
import App from './App.vue'
import router from './router'

import 'bulma/css/bulma.min.css'
Vue.config.productionTip = false

new Vue({
  render: h => h(App),
  router
}).$mount('#app')
```

5. we can use the new component: `<router-view/>` in our App.vue component, like so:

```
<template>
  <div id="app">
    <Navbar />
    <router-view/>
  </div>
</template>

<script>
import Navbar from './components/Navbar'

export default {
  name: 'App',
  components: {
    Navbar
  }
}
</script>
<style>
  .content-section {
    margin-top: -3.25rem;
    padding-top: 3.25rem;
  }
</style>
```

Note: we do keep Navbar since we want that part to be consistent across the whole application.

6. Handling a case when nothing gets matched

Error.vue

```
<template>
  <div class="content-section">
        <section class="hero is-white is-fullheight-with-navbar">
            <div class="hero-body">
                <div class="container">
                    <h1 class="title has-text-centered is-unselectable">
                    404: Page Not Found
                    </h1>
                    <br/>
                    <div class="subtitle has-text-centered">
                        <div class="button is-dark">
                        Return to Previous Page
                        </div>
                    </div>
                </div>
            </div>
        </section>
    </div>
</template>
```

7: Adding interactability

How do we add interactability?

In this section we will:

1. add navigation between the pages
2. Make modals pop-up with pre-filled data, and close

1. add navigation between the pages

Navbar.vue

```
<template>
    <nav class="navbar is-fixed-top is-transparent">
        <div class="navbar-brand">
            <div class="navbar-burger burger"
            :class="{'is-active':activeNavbar}"
            @click="toggleStatus">
                <span></span>
                <span></span>
                <span></span>
            </div>
        </div>
        <div class="navbar-menu"
        :class="{'is-active':activeNavbar}"
        @click="toggleStatus">
            <div class="navbar-start has-dropdown">
                <router-link v-for="(option, index) in OPTIONS" :key="index"
                    class="navbar-item is-unselectable" :to="URL[index]"
                    @click="toggleActiveOption(index)">
                    {{ option }}
                </router-link>
            </div>
        </div>
    </nav>
</template>
<script>
export default {
    data: function() {
        return {
            OPTIONS: ['Home', 'Dashboard', 'Add an Item'],
            URL: ['/', '/dashboard', '/add-item'],
            activeNavbar: false,
            optionIndex: 0
        }
    },
    methods: {
        toggleStatus: function() {
            this.activeNavbar = !this.activeNavbar
        },
        toggleActiveOption: function(index) {
            this.optionIndex = index
        }
    }
}
</script>
```

Error.vue

```
<template>
  <div class="content-section">
        <section class="hero is-white is-fullheight-with-navbar">
            <div class="hero-body">
                <div class="container">
                    <h1 class="title has-text-centered is-unselectable">
                    404: Page Not Found
                    </h1>
                    <br/>
                    <div class="subtitle has-text-centered">
                        <div class="button is-dark"
                        @click="$router.go(-1)">
                        Return to Previous Page</div>
                    </div>
                </div>
            </div>
        </section>
    </div>
</template>
```

2. Make modals pop-up with pre-filled data, and close in Dashboard

Code will be posted later. There are a lot of changes in many files