Lab 8: Networking in VueJS

Learning Objectives

- Get our VueJS applications interacting with our back ends
- Implement authentication handling for our VueJS applications

Note: during this lab you will be working on your assignment code. Do not let others see or copy your code. Plagiarism rules apply.

Exercise 1: Before you start…

This lab assumes that you have completed Lab 7. More specifically, you should have the following:

1. An application already set ready for your assignment
2. A router that can router between a home page and a login page

Exercise 2: Getting Organised

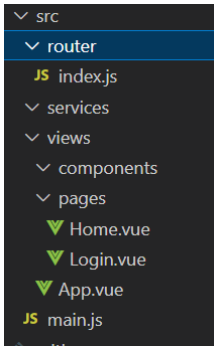
By the end of lab 7, you will have noticed your 'src' directory was getting a bit messy. A bit of pages, small components, and your router files. We'll start today by making sure that we are organised.

In your 'src' directory, create the following directory structure:

- src
  - views
    - pages
    - components
  - router
  - services

Your router directory is already set up from Exercise 1. You should also:

1) Add your App.vue file to the "views" directory
2) Add your "Home" and "Login" components to the "pages" directory
3) Update all of your code references to reflect the new structure
4) Test that everything still works.



How will the directories be used?

- **Pages**: stores all the page components (e.g., Home)
- **Components**: stores all the helper components (e.g., SingleComment)

- **Router**: stores all routing functionality
- **Services**: stores libraries for making API requests

Exercise 3: Getting a list of all articles (Home page)

1. In the services directory, create a file called “article.service.js”
2. In this file, add the below function for submitting a GET request to the /articles end-point (explained in the lecture)

```
const getAll = () => {
    return fetch("http://localhost:3333/articles")
    .then((response) => {
        if(response.status === 200){
            return response.json();
        }else{
            throw "Something went wrong"
        }
    })
    .then((resJson) => {
        return resJson
    })
    .catch((error) => {
        console.log("Err", error)
        return Promise.reject(error)
    })
}
```

3. We also need to export the function so that it is visible to other files. Add the below to the bottom of the file.

```
export const articleService = {
    getAll,
}
```

4. In the script section of your Home component, import the articles service. We can then call the getAll() function inside the components mounted() hook.

```
<script>
    import { articleService } from "../../services/article.service"

    export default {
        data() {
            return {
                articles: [],
                error: "",
                loading: true
            }
        },
        mounted() {
            articleService.getAll()
            .then(articles => {
                this.articles = articles
                this.loading = false
            })
            .catch(error => this.error = error);
        }
    }
</script>
```

5. Finally, in your template section, look through the articles array and display each article in a list:

```
<div>
    <h1>Welcome to my blog!</h1>

    <em v-if="loading">Loading articles...</em>

    <ul v-if="articles.length">
        <li v-for="article in articles" :key="article.article_id">
            {{ article.title + ' by ' + article.author }}
        </li>
    </ul>

    <div v-if="error">
        {{ error }}
    </div>
</div>
```

6. Test in your browser.

If it isn't working:

- Do you have your server running?
- If you open the developer tools, can you see a CORS error? (Refer to the lecture slides to fix)

Exercise 4: Getting a single article

1. In your page's directory, create a new component called "Article"
2. Add this component to your router:

```
{ path: "/article/:id", component: Article },
```

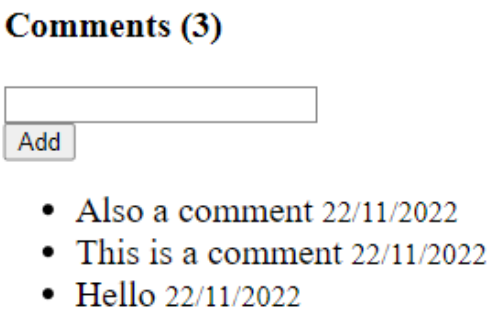
3. Now in your Home component template, alter it so that the list items now link through to the Article endpoint, passing in the article ID:

```
<li v-for="article in articles" :key="article.article_id">
    <router-link :to="'/article/' + article.article_id">
        {{ article.title + ' by ' + article.author }}
    </router-link>
</li>
```

4. Using the lecture slides to help you, implement the rest of the Article component. Can you also get and display the list of comments for the article?

Exercise 5: Adding comments

1. In your Article component, create a form before you display your list of comments.
2. On submit of your form, validate the input
3. If the form passes validation, create and call a function that sends a POST request to the /comments/:id end point. You will need to implement this function in the comments.service.js file that you should have created as part of Exercise 4
4. If the POST request is successful, send a GET request to get all of the comments and update the array.
5. Add a count of comments on the screen



Exercise 6: Logging in

1. Last week, you should have created a login form that validates the inputs when a button is clicked. Edit this function so that it also sends a POST request to the /login endpoint.
2. If the POST request is successful, store the user_id and session_token in the browser's localStorage. Refer to the lecture slides to help you with this.
3. If the user successfully logs in, you should redirect them to the "Dashboard" component (you will need to create this component and add it to your router first)

Exercise 7: Handling Authentication

The dashboard page should only be accessed by users who are logged in. We know if a user is logged in when a session_token exists for them in their browser storage. Using the lecture slides and last week's lab sheet, edit your router so that you can only access the dashboard when logged in.

Exercise 8: Logging out

Add a button to your dashboard. Clicking the button should log the user out (POST /logout) and remove the users details from the localStorage.

Exercise 9: Get on with it…

Look at the API specification in swagger, you have implemented six of the twelve endpoints in this lab. The remaining six require you to be authenticated, so should all be accessed through the dashboard in some way. How you design and implement the rest of the assignment is up to you. You have all the information and code needed to finish off all of the assignment's functionality now. So go ahead, finish it 😊