Cian Gilmore

N00182300

Vue Application with a REST API

Web application frameworks

# NavBar

There is a simple navbar that is mounted to the App.vue file. This allows the navbar to be present on every single page. It contains links to the index pages for the courses, enrolments, and lecturers. There is also a logout button so that you can logout at any time.

The logout function will remove the token from the local storage, as each json needs the user to have a token, they will not be able to access any of the functionality. I initially had a button to getCourses which would display all the courses on the index page, but I later mounted the function to make it easier to test. If you make it to the index page while logged out the courses will not display.

# Vue Router

Vue Router is the official router for Vue.js. It allowed us to easily create routes for the different views for pages of the application. It meant that if the user goes to a certain url path such as “/courses” then it will return the CoursesIndex component.

The router also allows you to specify the name of the route instead. Having a link that goes to “{ name: ‘courses\_index’}” will be the same as “/courses” as a pathway.

It is installed by typing “npm i vue-router” in the command prompt.

# Structure

There are three folders inside the views folder. They are for courses, enrolments, and lecturers with each of them containing a Create, Edit, Show and Index files.

The Courses Index view page will display all the courses. This is where you will be immediately redirected to when you login from the homepage. A table displays each course with the title, code, points, and level. These are sortable by ascending or descending.

Clicking on the Create Course button will bring you to the Create Course form, where you can enter the details and add a new course. When you click submit you will be redirected to the Courses Index page.

By clicking on the course title, you can be brought to its specific page where you can then update or delete it. You can also see all the enrolments for that specific course underneath.

The edit course page will have a form that is already pre filled out, this is done by getting the course with the right id and filling the form with its information for you to edit and then resubmit it again.

This structure is closely followed by both the lecturers and enrolments. You can view a specific lecturer and see their contact details. From here you can again, edit or delete it.

Enrolments is similar except when viewing the enrolments page for a specific class, you get the class information, the enrolment information such as the time and which lecturer, then that lecturers contact details. When editing, the form uses bootstrap for date and time pickers. There are also radio buttons to select the status. Finally, there are drop down menus for the Course ID and the Lecturer ID.

# Bootstrap Vue

I did some minimal styling using Bootstrap for Vue. It was fairly similar to other versions that I have used before but it also included Vue functionality.

The main features used were for forms and buttons. Although I did find it difficult sometimes to implement the Vue functionality with the Bootstrap elements, for example, it is supposed to support the router-links but I could not for the life of me get the href to work properly with the correct name for the component link. I’m sure the solution is simple but I ran out of time. This unfortunately has lead to an inconsistency with the design as the buttons on the index pages do not use Bootstrap but just button tags wrapped around the link.

A work around could have been done by creating a method at the bottom of the page that would redirect on click.

I found that adding classes to certain elements did not implement the Bootstrap design to them. I set my navbar to be blue by changing the variant attribute to “primary” but then the links all appeared the same colour as the background, when reading the Bootstrap documentation, it didn’t provide the solution I needed. This is also true for the href issue discussed above.

# Axios

Axios was used to consume the College API. It is a HTTP client library. It allowed me to use GET, POST, PUT, DELETE requests to the API which is hosted on a virtual machine acting as a server and then it can be rendered in the Vue application.

axios.get('http://college.api:8000/api/courses', {

headers: {Authorization: "Bearer "+token}

})

Above is a code snippet of what is being run when the getCourses method is executed. Authorization is required for the json so in the header the token is set.

It uses the GET request from axios to retrieve all the courses from the API.

# Insomnia

Insomnia was used to test the application and get a better understanding of the json files in a digestible way.

It made it clear what information was needed when using forms because it displayed all the properties.

Using a POST method at college.api:8000/api/login , you can see that you need authorization to login. If you fill out the json properly then you will get access to the token which is then used in the header for the other requests.

Once logged in you can get all the courses by sending a GET request to college.api:8000/api/courses, which fill return the json file with everything inside in the preview. It also gives the response code which is 200 for success, the time it took, and the data size.

Errors it showed included 404 Not Found and 422 Unprocessable Entity.

## GitHub link:

https://github.com/n00182300/WAFCA2.git