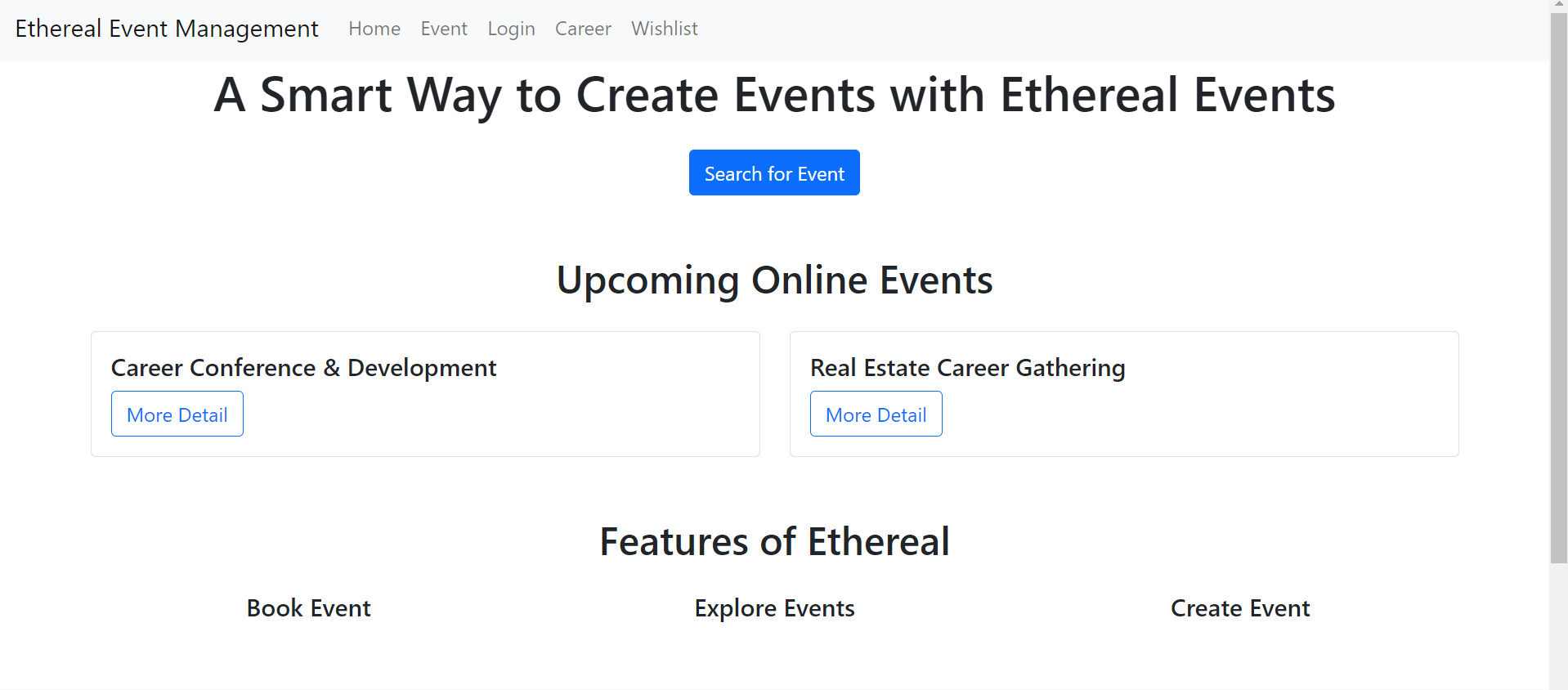
**Task 6.2 : Report on Custom Web of Event Management Application**

Student Name: Michelle Angelica Lautan

Student ID: 103524325

Final Design and implementation

1. Home Page



A screenshot of a computer

Description automatically generated

Ethereal event management home page include a navigation bar which is style using bootstrap using the following code:



Figure2: Navigation bar

To make a navigation bar that is responsive, I used Bootstrap. The navbar, navbar-expand-lg, and navbar-light bg-light classes are used to design the <nav> element in order to define the navbar, make it expandable on big screens, and give it a light background. The brand link (navbar-brand), toggler button (navbar-toggler) for navbar collapsing on smaller devices, and collapsible content section (collapse navbar-collapse) with unordered list (navbar-nav) are all included inside a container-fluid for full-width layout. There are router-link components stylised as navigation links (nav-link) in every list item (nav-item). This configuration makes use of Bootstrap's utility classes and responsive design concepts to provide a mobile-friendly, accessible navigation bar that adjusts to various screen sizes.

The information of the home pages inside Bootstrap classes was organised using a template string for layout and aesthetic design. The whole page content is contained within the `home-page` div, which begins with a centred hero section with a headline and a `router-link} button that is styled using the `mt-3} and `btn btn-primary} classes from Bootstrap. The next section is about upcoming events. It uses a Bootstrap `container} with a `row} of two columns ({col-md-6}), each having a `router-link} button styled as `btn btn-outline-primary} and a card ({card}) with information about the event. Similar in construction, the features section consists of a `container} containing a `row} of three columns ({col-md-4}), each containing a basic feature card.

The home page includes a search for event button to allow users to be directed directly to the event page. The home page also includes information regarding what ethereal management is about and what features are being provided by our website. Our website also provides a footer to allow users to access other pages easily without going back to the top page.

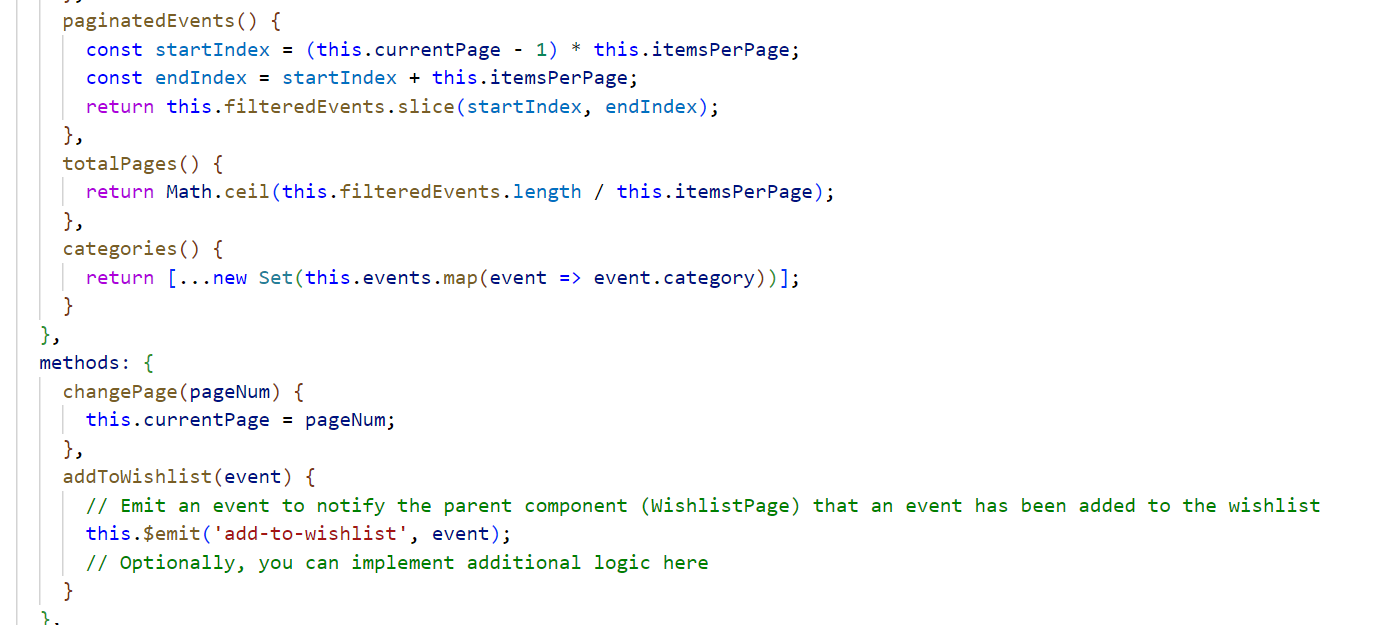
1. Event Page

Event Page is the main page in this Ethereal Event Website. It allows user to view available events, create events, edit events, and add their liking event to their Wishlist.

The purpose of the Event Page component in Ethereal Event is to provide an events list with search, filtering, and pagination capabilities. Several states, including events, wishlist, currentPage, itemsPerPage, searchQuery, and filterCriteria, are initialized by the data function. It returns only the relevant events by dynamically filtering them using computed properties in response to search queries, chosen categories, locations, and dates as shown in Figure 3. The filtered list is then sliced using the “paginatedEvents” computed property as shown in Figure 4 to show only the events for the current page, while “totalPages” determines how many pages are needed. A computed property that deduplicates the categories from the events list is used to extract the categories. Pagination is accomplished via methods like changePage and addToWishlist, which enable the incorporation of events into a wishlist and send an event to notify the parent component. A paginated view with buttons to flip between pages and Bootstrap class styling is used to achieve pagination. The pagination component supports navigation by using changePage with the correct page number. It does this by using computed properties to determine which events to display based on the current page.

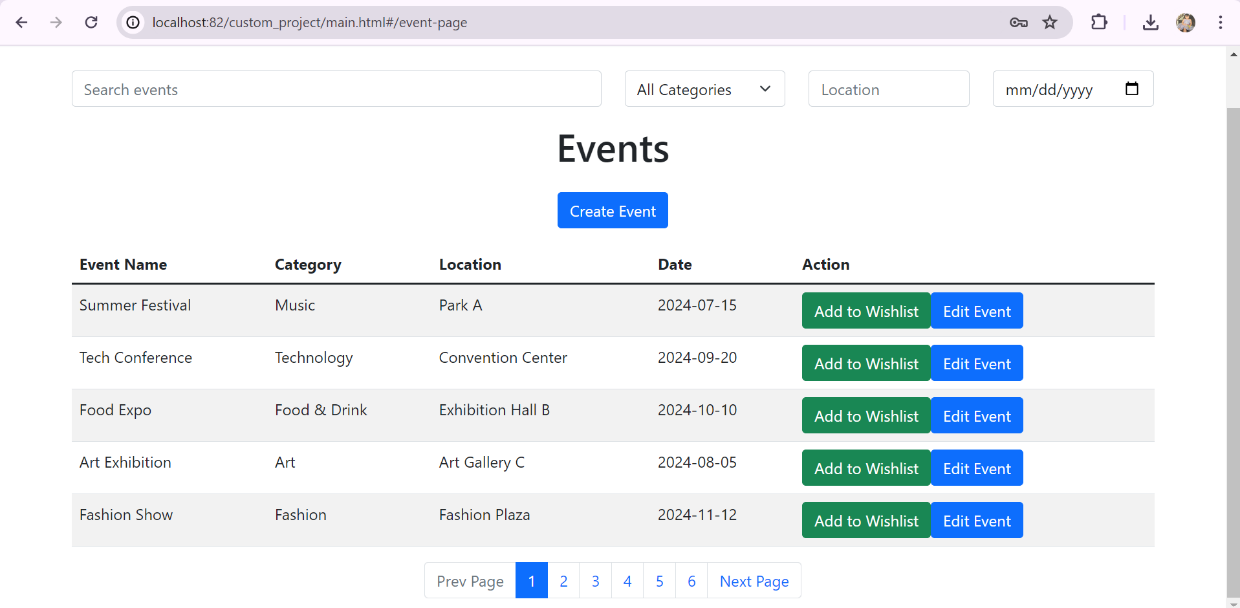


**Figure 3: Computed Properties for Filtering the data**



**Figure 4: Pagination Implementation**

**Pagination Output:**

****

**Figure 5: Pagination Implementation Result**

**Filtering Output:**

1. Filter By Search

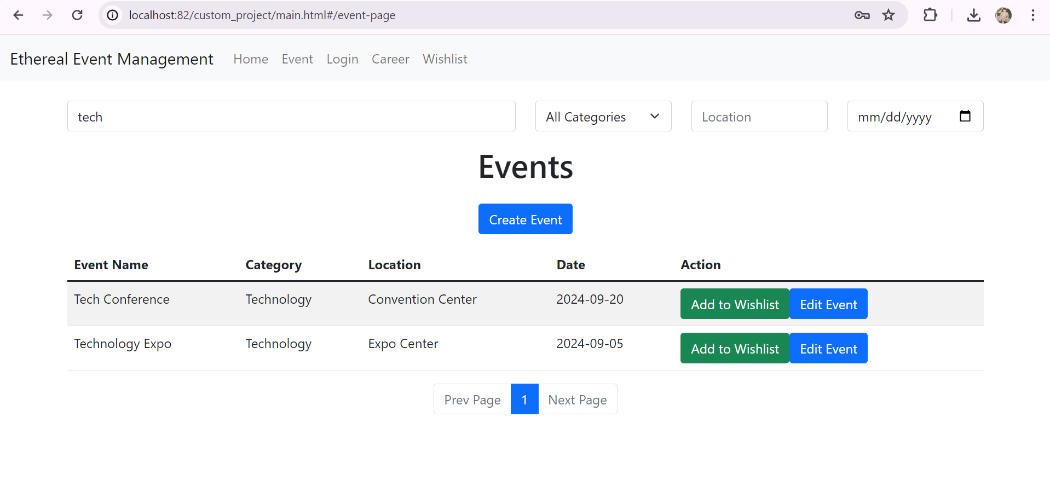


Figure 6: Filter by Search Result

1. Filter By Category (Drop Down)

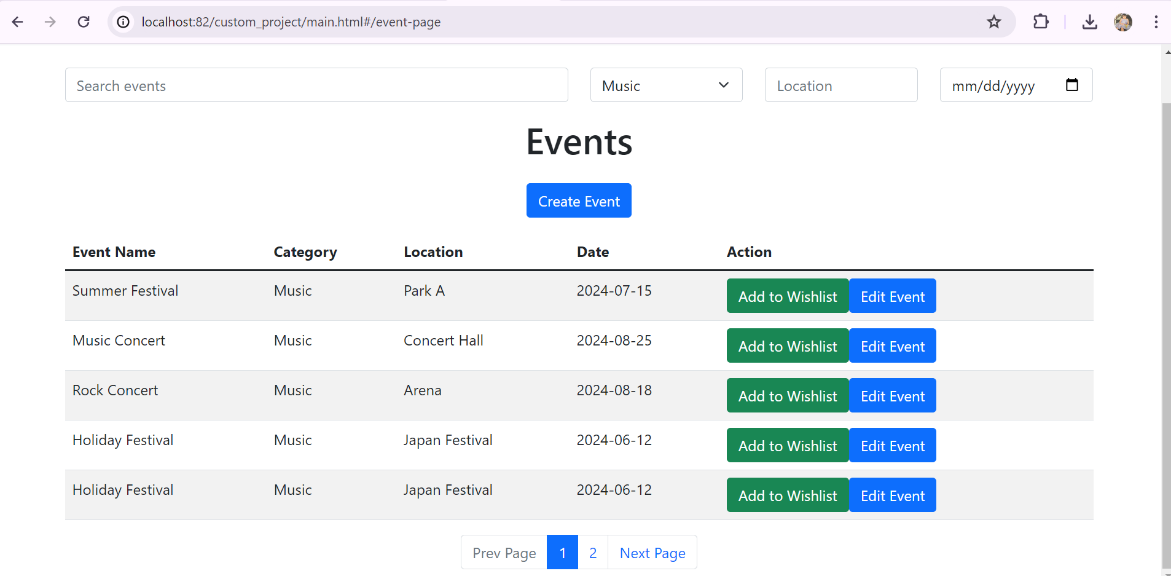


Figure 7: Filter event data by category

1. Filter By Date and Location

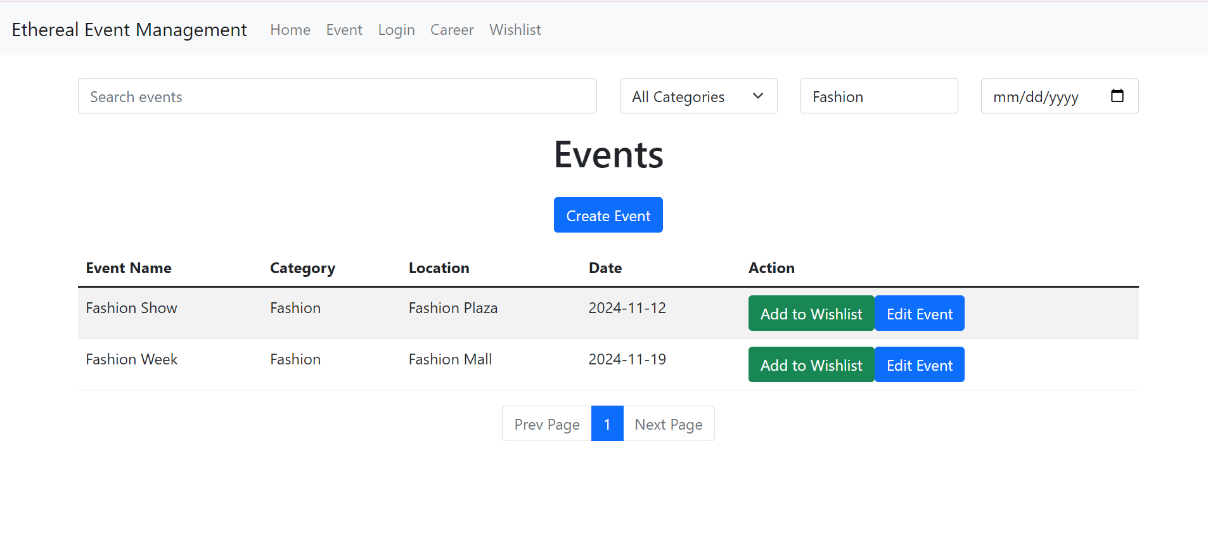


Figure 8: Filter by data’s date and location result

When the component has been built, data is collected from an external API by utilising the Fetch API to obtain event data from a given URL as shown in Figure 9. Any mistakes are recorded in errorMessage, and the events are kept in the events array. By connecting input elements to Vue's v-model directive, which updates the component's state in real-time, the search and filter features are implemented.



Figure 9: Fetching external API from the external database

Figure 10 below illustrates the API that has been created by the database in PhpMyAdmin after being configured in the PHP file:



Figure 10: External Event Data API from PhpMyAdmin MySQL

These are the database that has been made in PhpMyAdmin through the create and insert command:

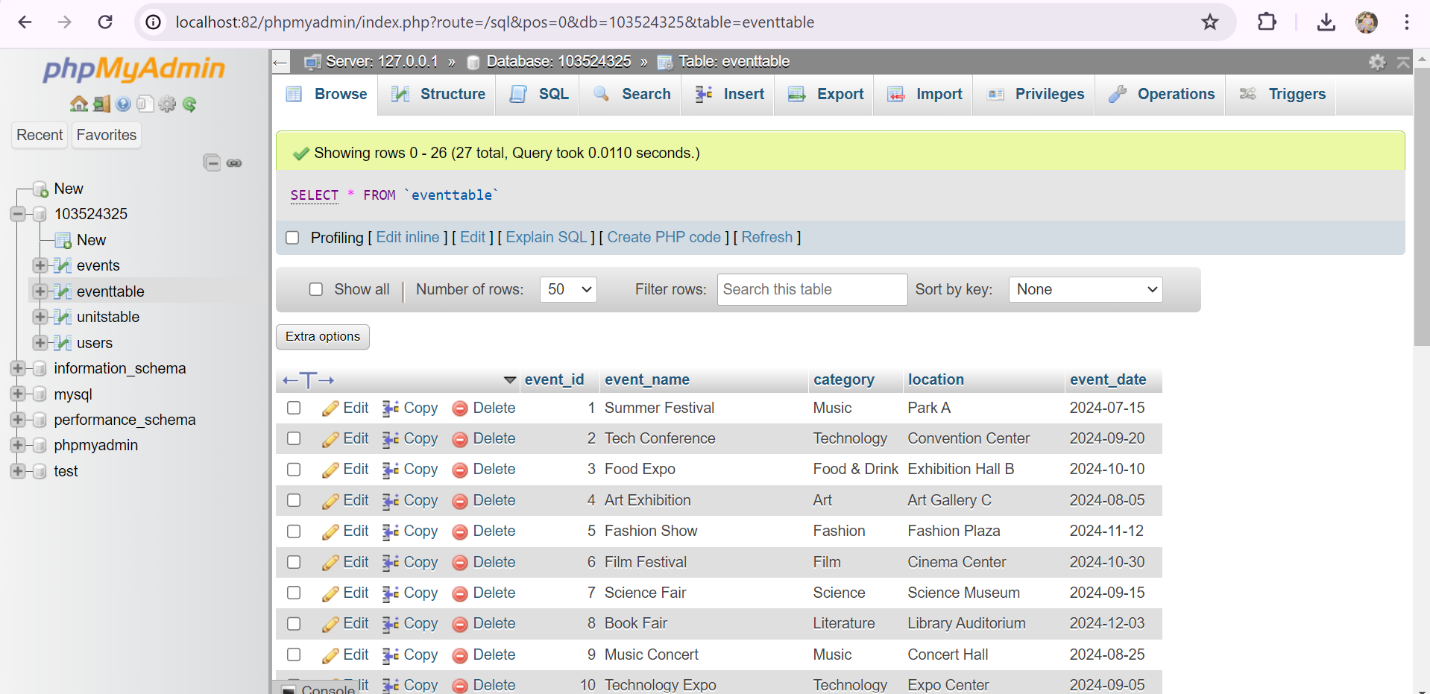


Figure 11: Event Page Database

The commands used in PhpMyAdmin to develop the table and insert data into “eventtable” are shown in Figure 12.

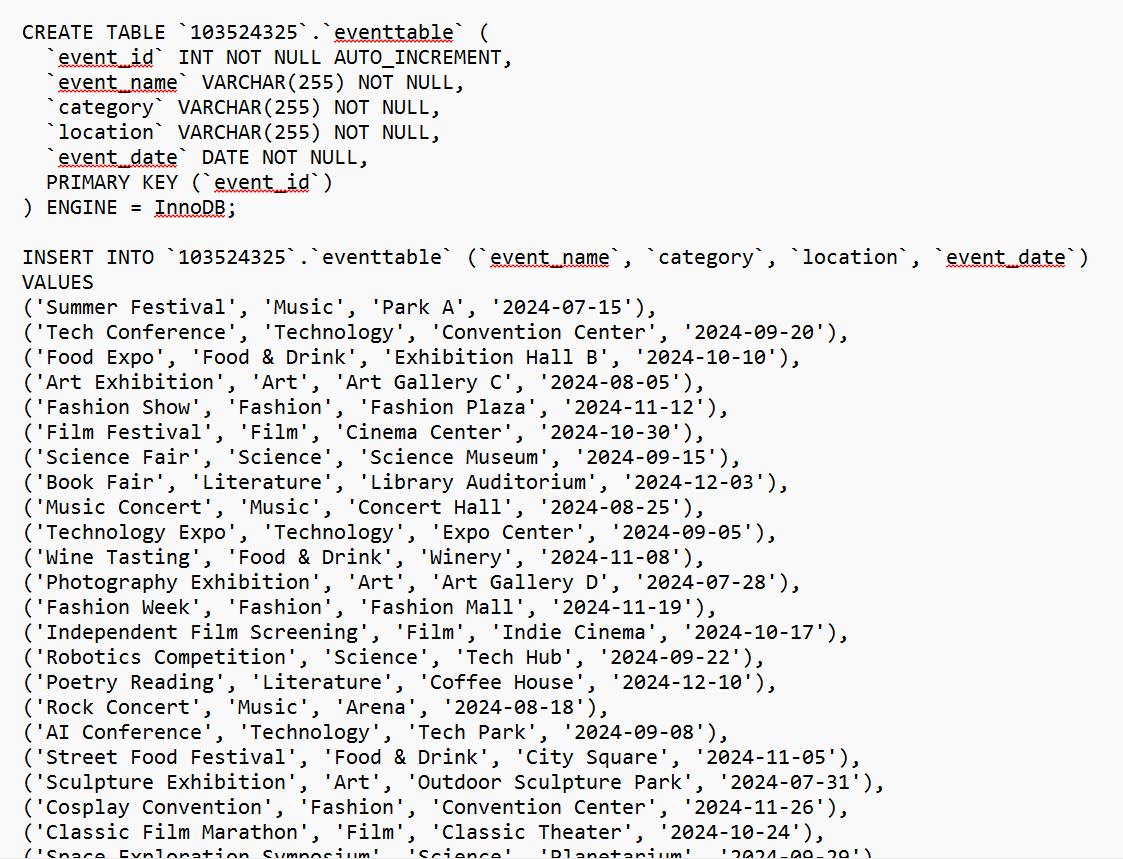


Figure 12: SQL Commands

1. Create Event Page

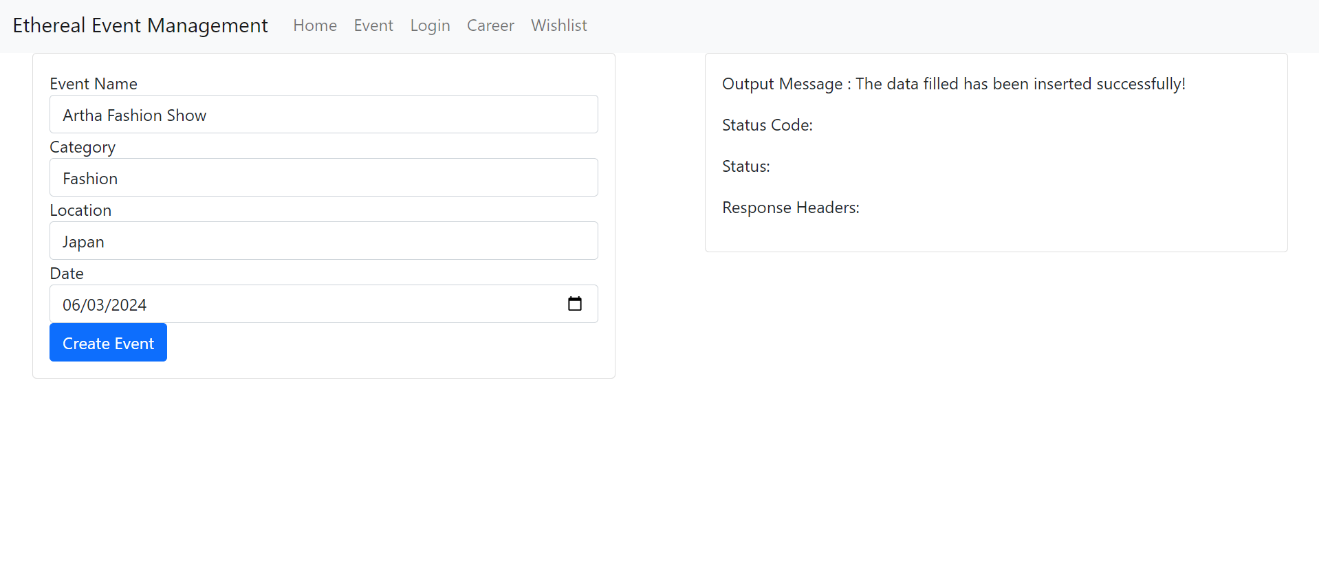


Figure 13: Create Event Page

Once user click the create event button in the event main page, it will redirect them to the create event page where user able to fill in their desired event name, event category, event location that want to be held and the date of the event. Once user has completed the form, the data they entered will be inserted to the external database and displayed in the event main page. To insert the data into the external database, a POST request was made to the server. It then sends this request to the URL of "./resources/apis.php/ " using the FETCH API. It tries to parse the answer as JSON when it has received one. It sets a message indicating success if the request is successful and a message stating the error if there is a problem as shown in Figure 13.

The data inserted from Figure 13 can be seen in the main event page as shown in Figure 14.

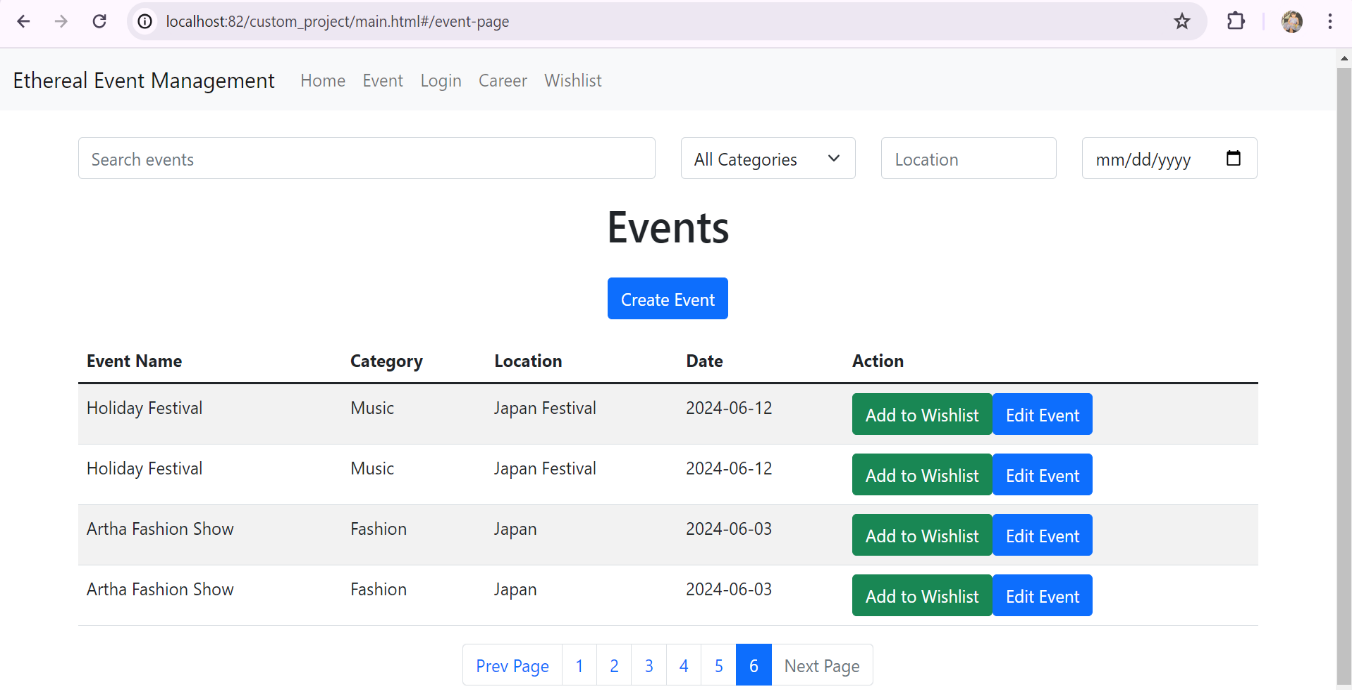
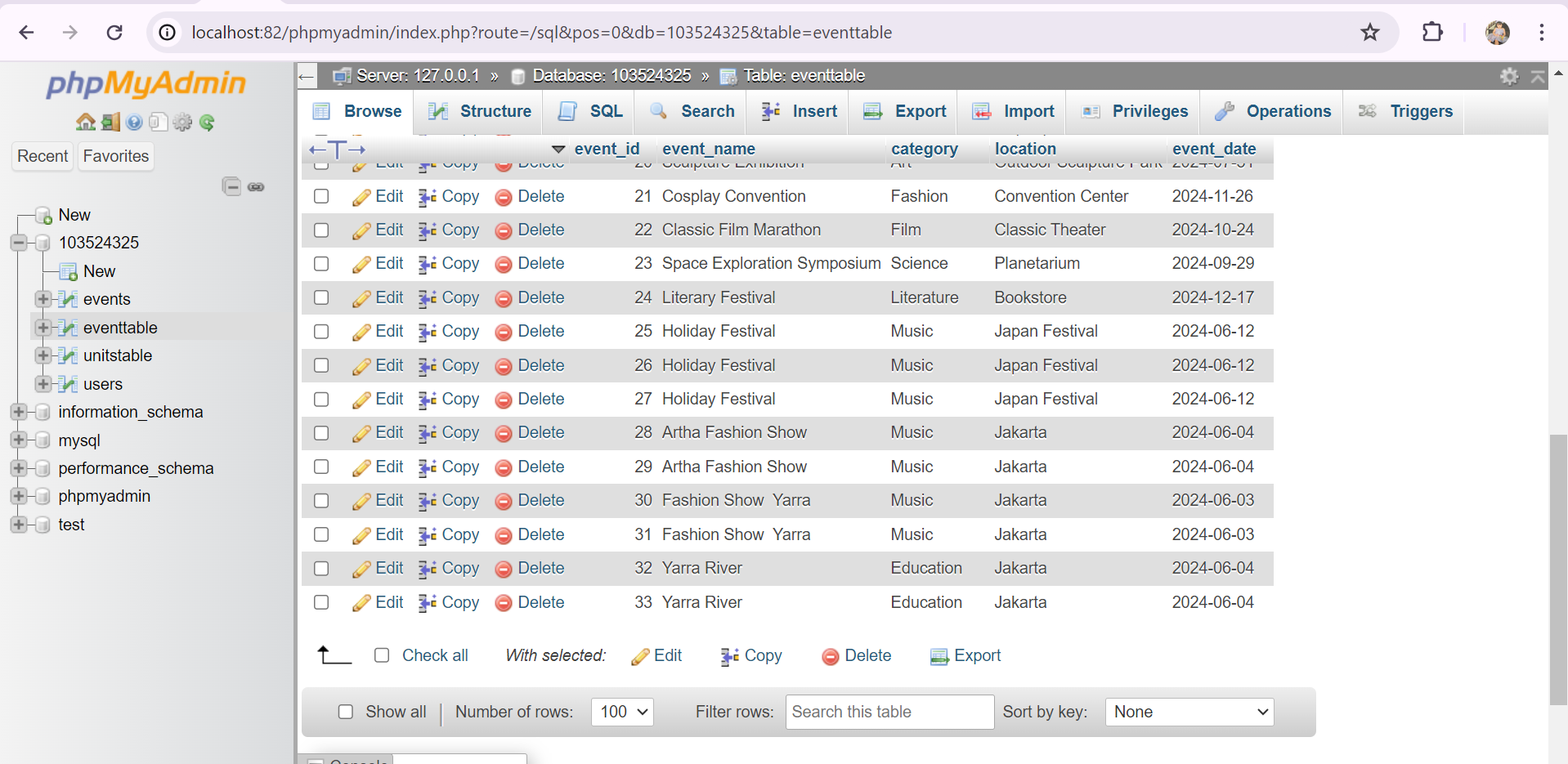


Figure 14: Create Event Result

***Database Result:***



1. Edit Event

User can also edit the event by clicking on the edit event button where it will also affect the External database by using a PUT request to the server as shown in Figure 15.

A screen shot of a computer

Description automatically generated

Figure 15: PUT Request made to the server to update the data in the database

1. Login Page

The `LoginPage` provides a user login interface with form validation. The `data` function of the component initializes state variables for the password, username, error messages associated with each, and a submitted flag. The `validateForm` method uses the `validateUsername` and `validatePassword` methods to check for issues in the username and password fields. If any fields are empty, the method sets the relevant error messages. If there are no errors, it uses Vue Router's `push` function to simulate a successful login by redirecting to the event page. The template includes a form with Bootstrap-styled username and password input boxes. Validation feedback is provided by displaying error warnings and utilizing Vue's conditional class bindings. When the user has successfully login, user will be redirected to the event main page.

Figure 16 illustrates the login page of the Ethereal Event Application.

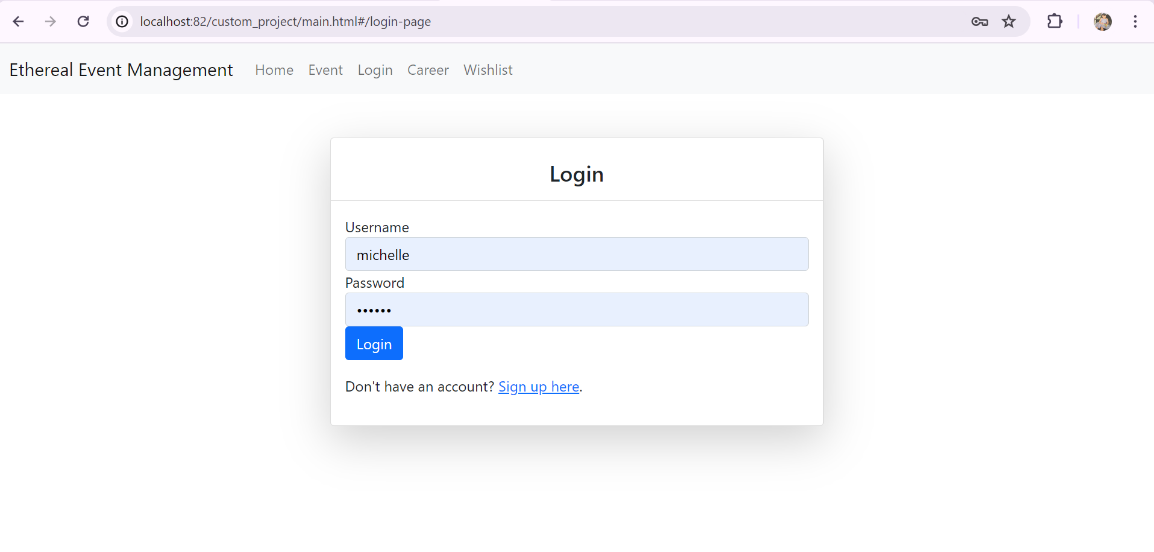


Figure 16: Login Page

1. Sign Up Page

The sign up allows user to register their account by filling up the form below in Figure 17.

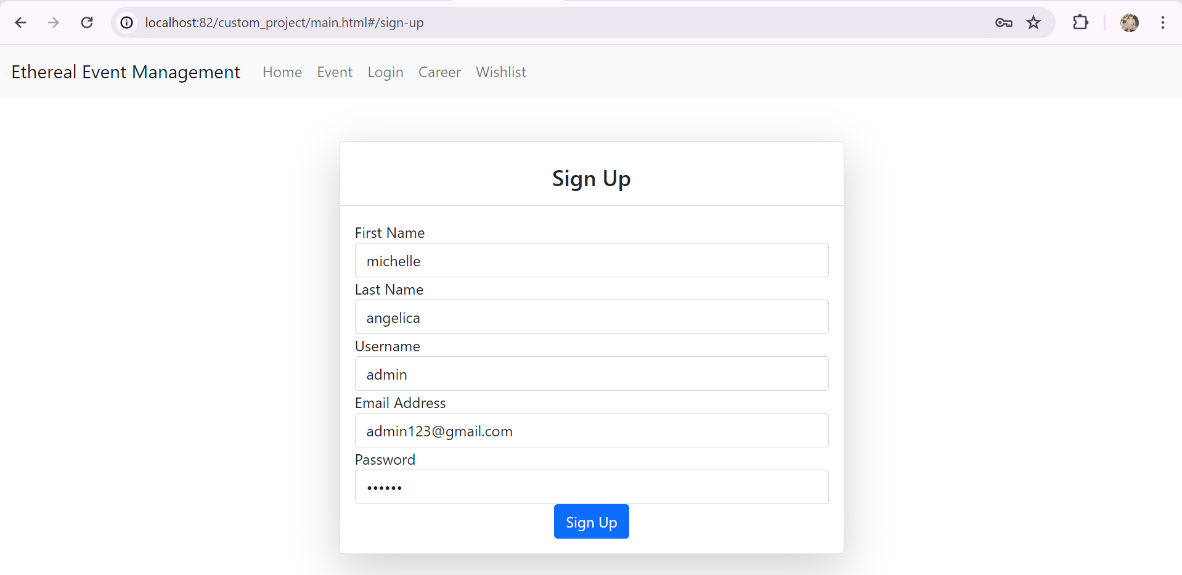


Figure 17: Sign Up Page

The implementation method of sign up validation:

A screenshot of a computer code

Description automatically generated

Figure 18: Form Validation of Sign Up Page

FirstName, lastName, email, and password, along with associated error message fields such as firstNameError and lastNameError, are initialized within the data method, along with a submitted flag to indicate if the form has been submitted. Functions for form validation, such as validateFirstName, validateLastName, validateUsername, validateEmail, and validatePassword, are defined within the methods object as depicted in Figure 18. These functions verify specific requirements for the fields they correspond to and appropriately update the relevant error message fields. All of the various validation routines are triggered, and the submitted flag is set to true by the validateForm method. The sign-up logic is executed if all validations are successful, represented in this sign-up page by an alert message indicating successful form submission.

The sign-up validation results are illustrated in Figure 19.

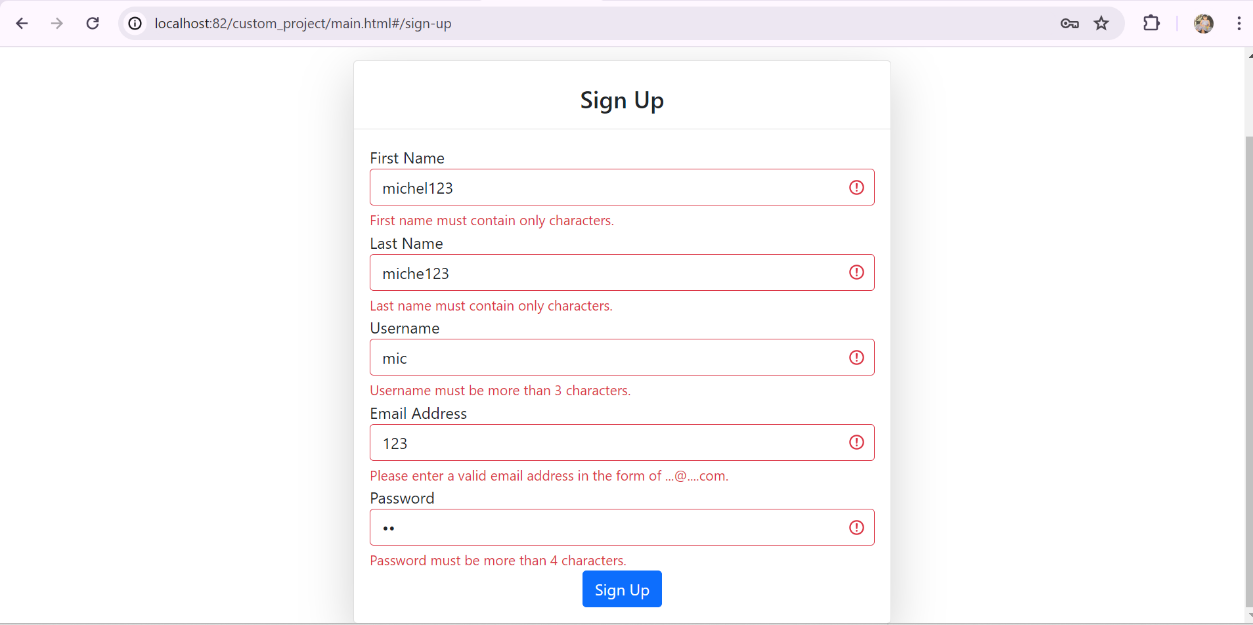


Figure 19: Sign Up Form Validation

1. Routing to Pages

The router link component in the home page's navigation bar in Figure 20 directs users to a specific page that is defined in the router configuration; upon clicking any of the links in the navigation bar, the user will be taken to the corresponding page component by the Vue Router.



Figure 20: Router Link in the Navigation Bar

Initially, we must initialise the routes for the various pages, where each route specifies a different path and the corresponding component to render as shown in Figure 21.These routes are then used to create an instance of the Vue Router, which controls the navigation between these pages through hash-based history. After the Vue application is created, the router is added as a plugin before the application is mounted to the DOM element with the id "app."

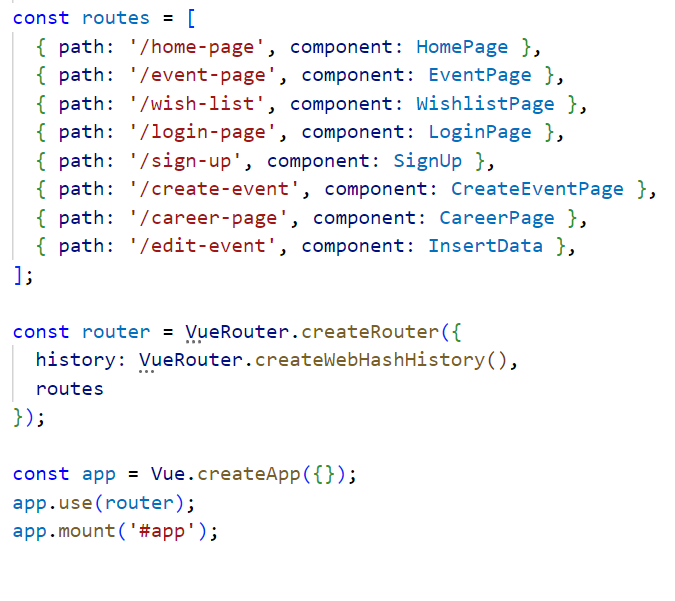


Figure 21: Vue Router Configuration