COMP-1640

Enterprise Web Software Development

Contents

1. System Implementation

1.1 Techniques

1.1.1 Front-end

A JavaScript framework called Vue is used to create user interfaces. It offers a declarative, component-based programming architecture that makes it easy to create user interfaces of unlimited complexity by building on top of common HTML, CSS, and JavaScript (Introduction).

I chose to employ vuejs once I become familiar with the font-end coding technology. Due to Vue.js's ease of usage and learning. Like myself, new users may easily get started designing applications without much experience because to the straightforward syntax and comprehensive documentation.

Advantage:

Vue.js's strength lies in its simple, intuitive syntax, which makes it simple for novices to use and become accustomed to. Because of its utilization of Virtual DOM and strong reactivity system, which maximize application speed and offer a seamless user experience, it is also highly regarded for performance. The adaptability of Vue.js is another asset. It enables the application to be divided into separate components for simple maintenance and reuse, and it facilitates seamless integration into current projects without adding unnecessary complexity. In addition, Vue.js boasts a vibrant and inventive community along with a varied extension ecosystem with a wide range of extension libraries and plugins.

Disadvantage:

Vue.js is not without flaws either. It might not offer the organizational structure and features that larger frameworks like Angular or React do in large-scale, sophisticated applications. Furthermore, because of its extreme flexibility, it might be challenging to guarantee that Vue.js applications are constructed in compliance with certain guidelines and standards. While the Vue.js community is expanding, it is still smaller than that of other frameworks, and some of the support and documentation might not be as good as one might anticipate.

Conclusion:

In short, because of its versatility, great performance, and ease of use, Vue.js is a suitable option for small to medium-sized online applications. Still, shortcomings must be taken into account to make sure it fits the project's particular needs.

Vuejs structure

1. Modules

Every module in the program serves a distinct and reusable purpose. With Vue.js, modules facilitate the division of functionality and user interface into more manageable and maintainable components. This lowers application complexity and promotes code reuse. Depending on the particular requirements of the application, each Vue.js module may comprise components, directives, filters, mixins, and other elements. You may arrange your code in a methodical, adaptable, and manageable manner by utilizing modules. Additionally, modules provide the establishment of connections between various application components, enabling simple access to information and features from other modules.

2. Components

Every Vue.js component creates a separate, reusable user interface element. Vue.js components assist in breaking up the user interface into smaller elements, each in charge of carrying out a certain task. This facilitates easier application maintenance, lowers complexity, and increases code reuse. The HTML, CSS, and JavaScript required for display and user interaction can be included in any component. With the help of Vue.js, you can define and use components with ease and flexibility, enabling you to construct a wide range of complex and colorful user interfaces.

3. Templates

Templates are used to specify how the logic and data of a component will be presented to and utilized by the user. In Vue.js, templates are usually written in HTML using special tags called "directives" from Vue that tie the functionality and data of the component to the user interface. You may do tasks like presenting dynamic data, managing user events, repeating items, and controlling the display of components depending on circumstances with the aid of directives like {{ }}, v-bind, v-on, v-for, v-if,... The application logic and user interface are kept apart in the code by using templates, which facilitates simpler reading and maintenance. Additionally, it makes it possible to reuse patterns across a wide range of components, increasing code reuse and lowering application complexity.

4. Directives

Vue provides unique characteristics that start with v- that are used to change the DOM declaratively. They are employed in Vue.js templates to give HTML elements dynamic behavior. Among the often used directives are v-if, v-for, v-bind, v-on, etc. With the help of these directives, developers may handle events, conditionally render elements, bind data to the DOM, and modify styles and attributes in response to changes in data. All things considered, directives improve the responsiveness and interactivity of Vue.js apps by offering an extensive set of DOM modification tools.

5. Services

While Vue.js lacks an integrated idea of services, akin to some other frameworks such as Angular, developers frequently employ diverse patterns and strategies to incorporate services. A wide range of operations, including as data retrieval via APIs, state management, validation, authentication, utility functions, and more, may be included in services in Vue.js. By concentrating similar logic and functionality in one location, they aid in the promotion of reuse, maintainability, and code structure. A straightforward JavaScript module, a Vuex module for state management, a Vue plugin, a mixin, or a global event bus are just a few of the methods that may be used to develop services. The application's particular needs as well as the developer's or team's choices determine the solution to be used.

6. Dependency Injection

Services, libraries, and other items that components require to carry out particular tasks might be considered dependencies. Vue.js's Dependency Injection method facilitates easy control over application dependencies, minimizes restrictions between components, and increases code reuse. DI enables us to supply external dependencies to components at the time of instantiation or as arguments, bypassing the need for direct dependency declaration and initialization in components. via setup. Using props to transfer data from parent to child components, inject and provide in Vue.js to share data between components without using props, and using state management libraries like Vuex to control the application's global state are a few ways to implement dependency injection in Vue.js.

2. Front-end

Login

First, it uses the Axios library to make HTTP requests, in this case a POST request to an API endpoint.

The code defines a Vue component, with data and methods to process the data and interact with the API.

Data defined in the component includes:

results: An object to store the results returned from the API, however, in this code there is no use of results.

login: An object containing the user's login information, including email and password.

loginError: A string to store the login error message.

There are two Vue lifecycle hooks used:

create(): Called when the Vue instance is created.

mounted(): Called when the Vue instance has been mounted to the DOM.

There is a LoginData() method defined to make the login request:

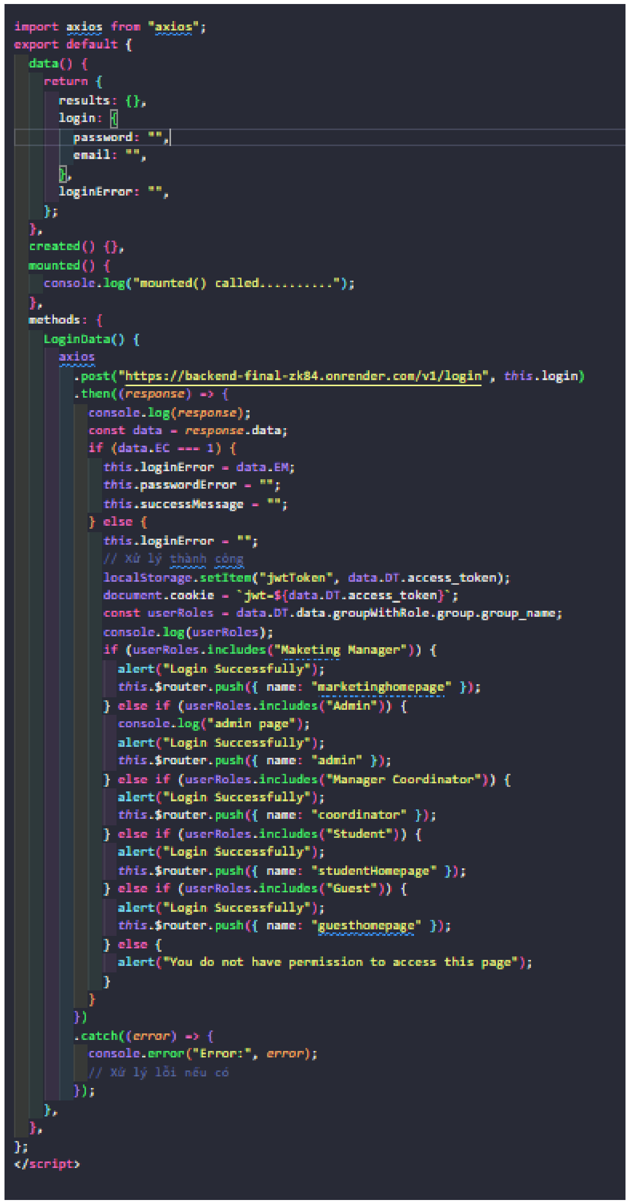
Use Axios to send a POST request to API endpoint "/v1/login" with login data from login.

Handling results returned from the API:

If there is an error (EC = 1), display an error message.

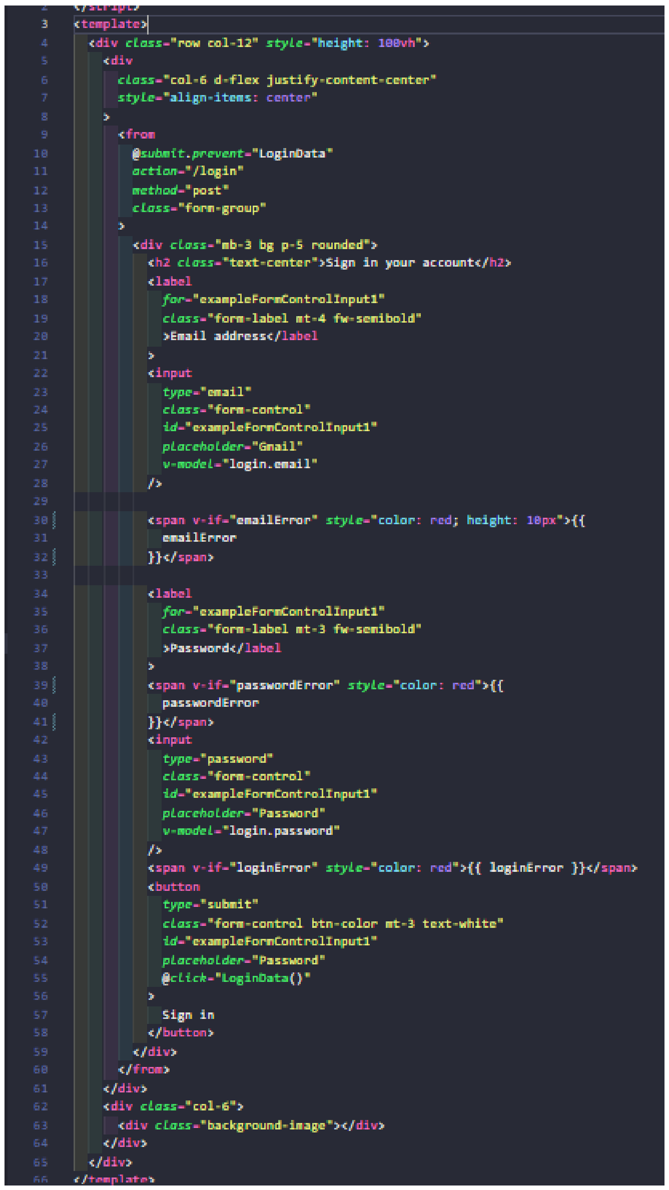
If there are no errors, store the access token in localStorage and cookies, then based on the user's role, redirect to different pages using Vue's router ($router.push()).

If there is an error while sending the request, an error message will be displayed in the console.



```
import axios from "axios";
export default {
  data() {
    return {
      results: {},
      login: {
        password: "",
        email: "",
      },
      loginError: "",
    };
  },
  created() {},
  mounted() {
    console.log("mounted() called..........");
  },
  methods: {
    LoginData() {
      axios
        .post("https://backend-final-zk84.onrender.com/v1/login", this.login)
        .then((response) => {
          console.log(response);
          const data = response.data;
          if (data.EC === 1) {
            this.loginError = data.EM;
            this.passwordError = "";
            this.successMessage = "";
          } else {
            this.loginError = "";
            // Xử lý thành công
            localStorage.setItem("jwtToken", data.DT.access_token);
            document.cookie = `jwt-${data.DT.access_token}`;
            const userRoles = data.DT.data.groupWithRole.group.group_name;
            console.log(userRoles);
            if (userRoles.includes("Maketing Manager")) {
              alert("Login Successfully");
              this.$router.push({ name: "marketinghomepage" });
            } else if (userRoles.includes("Admin")) {
              console.log("admin page");
              alert("Login Successfully");
              this.$router.push({ name: "admin" });
            } else if (userRoles.includes("Manager Coordinator")) {
              alert("Login Successfully");
              this.$router.push({ name: "coordinator" });
            } else if (userRoles.includes("Student")) {
              alert("Login Successfully");
              this.$router.push({ name: "studentHomepage" });
            } else if (userRoles.includes("Guest")) {
              alert("Login Successfully");
              this.$router.push({ name: "guesthomepage" });
            } else {
              alert("You do not have permission to access this page");
            }
          }
        })
        .catch((error) => {
          console.error("Error:", error);
          // Xử lý lỗi nếu có
        });
    },
  },
};
</script>
```

Figure 1: Function Login



```
</script>
<template>
  <div class="row col-12" style="height: 100vh">
    <div
      class="col-6 d-flex justify-content-center"
      style="align-items: center"
    >
      <from
        @submit.prevent="LoginData"
        action="/login"
        method="post"
        class="form-group"
      >
        <div class="mb-3 bg p-5 rounded">
          <h2 class="text-center">Sign in your account</h2>
          <label
            for="exampleFormControlInput1"
            class="form-label mt-4 fw-semibold"
            >Email address</label
          >
          <input
            type="email"
            class="form-control"
            id="exampleFormControlInput1"
            placeholder="Gmail"
            v-model="login.email"
          />

          <span v-if="emailError" style="color: red; height: 10px">{{
            emailError
          }}</span>

          <label
            for="exampleFormControlInput1"
            class="form-label mt-3 fw-semibold"
            >Password</label
          >
          <span v-if="passwordError" style="color: red">{{
            passwordError
          }}</span>
          <input
            type="password"
            class="form-control"
            id="exampleFormControlInput1"
            placeholder="Password"
            v-model="login.password"
          />
          <span v-if="loginError" style="color: red">{{ loginError }}</span>
          <button
            type="submit"
            class="form-control btn-color mt-3 text-white"
            id="exampleFormControlInput1"
            placeholder="Password"
            @click="LoginData()"
          >
            Sign in
          </button>
        </div>
      </from>
    </div>
    <div class="col-6">
      <div class="background-image"></div>
    </div>
  </div>
</template>
```

Figure 2: Login Interface code

The above code is part of the template of a Vue component, which is responsible for displaying a login form to the user and handling the login event.

This code uses Bootstrap's structure to divide the layout into two parts:

A section on the left with class "col-6" contains the login form.

The part on the right also has class "col-6" to display the background image.

In the login form:

The form's submit event is caught using @submit.prevent="LoginData", which prevents the form's default action of sending a request to the server and instead calls the LoginData() function to handle the login.

Inputs for email and password are linked to data in the Vue component via v-model="login.email" and v-model="login.password".

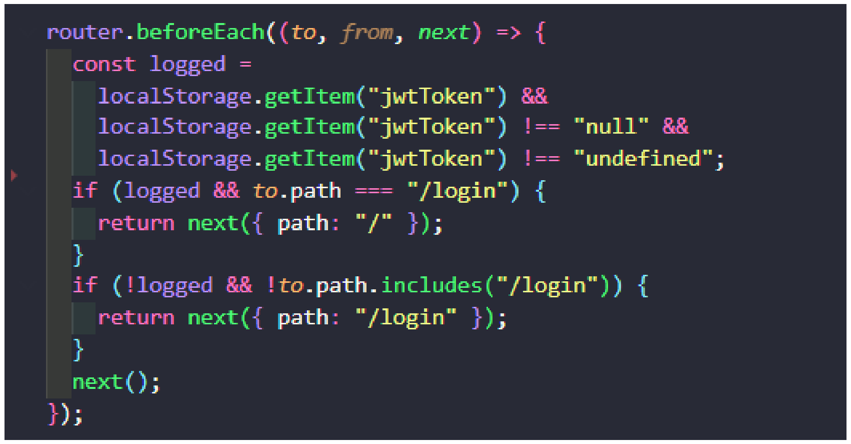
There are <span> tags used to display error messages for email, password, and login errors.

The "Sign in" button is pressed to call the LoginData() function when pressed.

The CSS section is embedded directly into the style attributes of HTML elements to customize the look and feel.

The right part has the class "background-image" which is where the background image is displayed.

With this structure, users can fill in their login information into the form and submit it. When the form is submitted, the LoginData() function is called to process the login information and interact with the API to check and validate the login information.



```
router.beforeEach((to, from, next) => {
  const logged =
    localStorage.getItem("jwtToken") &&
    localStorage.getItem("jwtToken") !== "null" &&
    localStorage.getItem("jwtToken") !== "undefined";
  if (logged && to.path === "/login") {
    return next({ path: "/" });
  }
  if (!logged && !to.path.includes("/login")) {
    return next({ path: "/login" });
  }
  next();
});
```

Figure 3: Function middleware

Check Login Status:

This function first checks whether there are credentials saved in localStorage by checking the value of jwtToken.

If yes, the user is logged in (logged is true).

Handling Redirects:

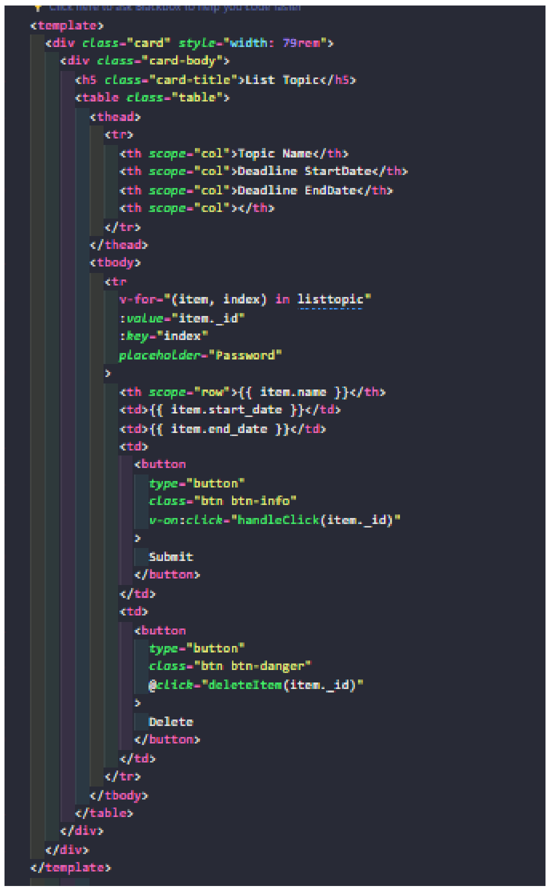
If the user is logged in (logged is true) and they try to access the login page (to.path === "/login"), this function will redirect the user to the main page (next({ path: "/" })).

If the user is not logged in (logged is false) and they try to access a page other than the login page (!to.path.includes("/login")), this function will redirect the user to the login page enter (next({ path: "/login" })).

Continue Navigation:

If any of the above conditions are not met, the function will allow navigation to continue by calling next() without parameters.

List Topic:



```
<template>
  <div class="card" style="width: 79rem">
    <div class="card-body">
      <h5 class="card-title">List Topic</h5>
      <table class="table">
        <thead>
          <tr>
            <th scope="col">Topic Name</th>
            <th scope="col">Deadline StartDate</th>
            <th scope="col">Deadline EndDate</th>
            <th scope="col"></th>
          </tr>
        </thead>
        <tbody>
          <tr
            v-for="(item, index) in listtopic"
            :value="item._id"
            :key="index"
            placeholder="Password"
          >
            <th scope="row">{{ item.name }}</th>
            <td>{{ item.start_date }}</td>
            <td>{{ item.end_date }}</td>
            <td>
              <button
                type="button"
                class="btn btn-info"
                v-on:click="handleClick(item._id)"
              >
                Submit
              </button>
            </td>
            <td>
              <button
                type="button"
                class="btn btn-danger"
                @click="deleteItem(item._id)"
              >
                Delete
              </button>
            </td>
          </tr>
        </tbody>
      </table>
    </div>
  </div>
</template>
```

Figure 4:List topic

This snippet is part of the app's user interface, written in HTML and uses Vue.js to interact with data.

This code displays a list of topics in a card, with information such as topic name, start date, and end date.

Structure of the card:

card: Is a Bootstrap tag to create a card interface.

card-body: Is the part of the card that contains the content of the card.

card-title: Is the title of the card, in this case "List Topic".

Inside the card is a table to display a list of topics (listtopic), with the columns being "Topic Name", "Deadline StartDate", "Deadline EndDate", and the last two columns are the "Submit" button. " and "Delete" for each topic.

The code uses v-for to iterate over each element in the listtopic array and display each topic's information in a single line (<tr>).

Each topic has a corresponding "Submit" and "Delete" button to perform corresponding actions.

Each topic's data is taken from item in each v-for loop:

item.name: Displays the name of the topic.

item.start_date: Displays the topic's start date.

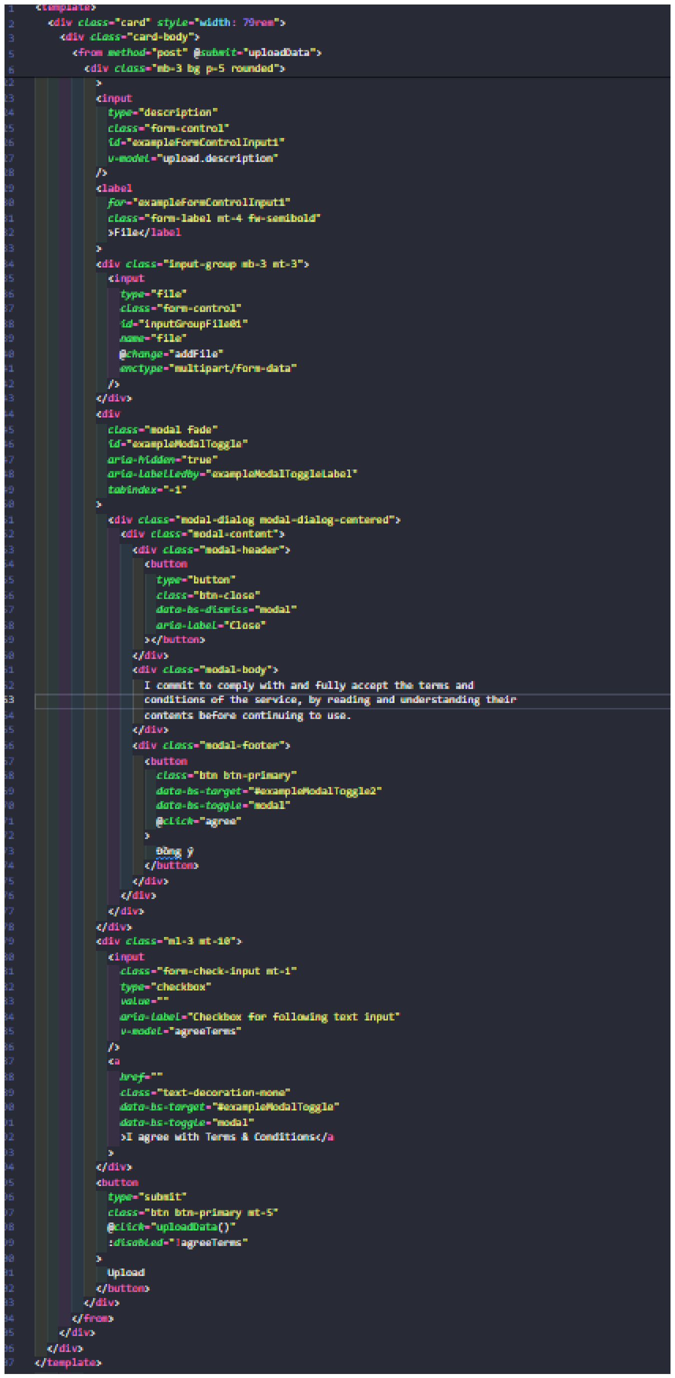
item.end_date: Displays the end date of the topic.

The v-on:click event is used to call the handleClick(item._id) method when the user presses the "Submit" button.

handleClick(item._id): This method is often used to handle sending information or redirecting to another page.

The @click event is used to call the deleteItem(item._id) method when the user presses the "Delete" button.

deleteItem(item._id): This method is often used to delete an item from a list.



```
<template>
  <div class="card" style="width: 79rem">
    <div class="card-body">
      <from method="post" @submit="uploadData">
        <div class="mb-3 bg p-5 rounded">
          >
          <input
            type="description"
            class="form-control"
            id="exampleFormControlInput1"
            v-model="upload.description"
          />
          <label
            for="exampleFormControlInput1"
            class="form-label mt-4 fw-semibold"
            >File</label
          >
          <div class="input-group mb-3 mt-3">
            <input
              type="file"
              class="form-control"
              id="inputGroupFile01"
              name="file"
              @change="addFile"
              enctype="multipart/form-data"
            />
          </div>
          <div
            class="modal fade"
            id="exampleModalToggle"
            aria-hidden="true"
            aria-labelledby="exampleModalToggleLabel"
            tabindex="-1"
          >
            <div class="modal-dialog modal-dialog-centered">
              <div class="modal-content">
                <div class="modal-header">
                  <button
                    type="button"
                    class="btn-close"
                    data-bs-dismiss="modal"
                    aria-label="Close"
                  ></button>
                </div>
                <div class="modal-body">
                  I commit to comply with and fully accept the terms and
                  conditions of the service, by reading and understanding their
                  contents before continuing to use.
                </div>
                <div class="modal-footer">
                  <button
                    class="btn btn-primary"
                    data-bs-target="#exampleModalToggle2"
                    data-bs-toggle="modal"
                    @click="agree"
                  >
                    Đồng ý
                  </button>
                </div>
              </div>
            </div>
          </div>
          <div class="ml-3 mt-10">
            <input
              class="form-check-input mt-1"
              type="checkbox"
              value=""
              aria-label="Checkbox for following text input"
              v-model="agreeTerms"
            />
            <a
              href=""
              class="text-decoration-none"
              data-bs-target="#exampleModalToggle"
              data-bs-toggle="modal"
              >I agree with Terms & Conditions</a
            >
          </div>
          <button
            type="submit"
            class="btn btn-primary mt-5"
            @click="uploadData()"
            :disabled="!agreeTerms"
          >
            Upload
          </button>
        </div>
      </from>
    </div>
  </div>
</template>
```

Figure 5: Interface Name, Description, File, terms and conditions

In this code, there is an HTML form where users can upload information and files. Here is a detailed analysis:

A div tag with class "card" is used to create a card interface to contain the form's content.

The title of the card tag is placed in an h5 tag with the class "card-title". In this case, the title is "Submit My Post".

The HTML form is placed within a form tag, with a post method and @submit event attached to the uploadData method, which means that when the user submits the form, the uploadData method will be called.

In the form, there are fields for entering information:

Name: Use input with type="name" and v-model="upload.name" to link to upload.name's data.

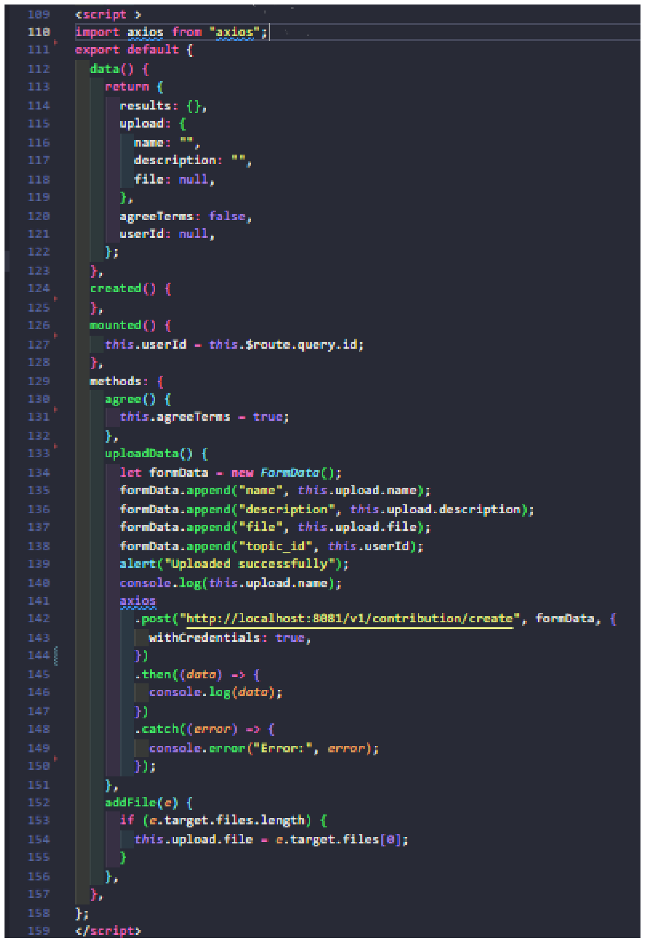
Description: Same as above, but data is associated with upload.description.

File: Use input with type="file" and @change="addFile" so that when the user selects a file, the addFile function will be called to save the file to the upload.file variable.

There is a modal (<div class="modal">) used to display the terms and conditions of the service. Users need to agree to these terms before uploading files.

There is a checkbox where users can agree to the terms and conditions, and an "I agree with Terms & Conditions" link that opens the modal when clicked.

The "Upload" button is used to submit the form. This button has an @click="uploadData()" event, but is already set in the uploadData method above. Additionally, the button will be disabled if the user has not agreed to the terms and conditions.



```
<script >
import axios from "axios";
export default {
  data() {
    return {
      results: {},
      upload: {
        name: "",
        description: "",
        file: null,
      },
      agreeTerms: false,
      userId: null,
    };
  },
  created() {
  },
  mounted() {
    this.userId = this.$route.query.id;
  },
  methods: {
    agree() {
      this.agreeTerms = true;
    },
    uploadData() {
      let formData = new FormData();
      formData.append("name", this.upload.name);
      formData.append("description", this.upload.description);
      formData.append("file", this.upload.file);
      formData.append("topic_id", this.userId);
      alert("Uploaded successfully");
      console.log(this.upload.name);
      axios
        .post("http://localhost:8081/v1/contribution/create", formData, {
          withCredentials: true,
        })
        .then((data) => {
          console.log(data);
        })
        .catch((error) => {
          console.error("Error:", error);
        });
    },
    addFile(e) {
      if (e.target.files.length) {
        this.upload.file = e.target.files[0];
      }
    },
  },
};
</script>
```

Figure 6: uploadData function

This code is another Vue component, whose task is to display a form for users to upload a file and send information related to that file to the server via an HTTP POST request.

The upload object in the Vue component's data contains data fields related to file uploads:

name: Name of the file.

description: Description of the file.

file: The file object selected for upload.

There is an agreeTerms variable to track whether the user has agreed to the terms.

The userId variable is used to store the user's id, extracted from the URL's query parameter, via $route.query.id in the mounted() method. This helps determine which current user is uploading the file.

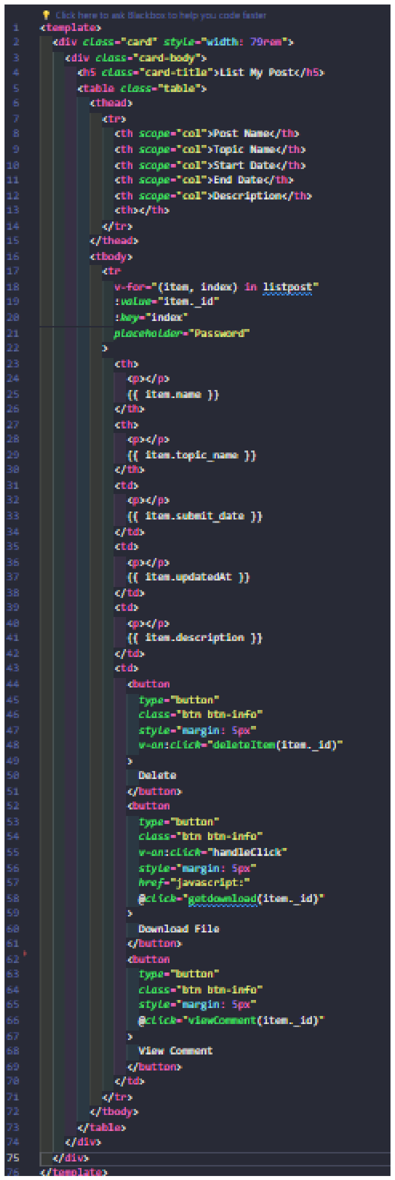
There are two methods:

agree(): Called when the user agrees to the terms. Update makes agreeTerms true.

uploadData(): Called when the user presses a button to upload a file. Create a FormData object to encapsulate the data fields and files. Send a POST request to the "/v1/contribution/create" endpoint on the server, along with the packaged data using the Axios POST method.

The addFile(e) method is called when the user selects a file to upload. It checks to see if the user has selected a file, and if so, it saves the file to the upload.file variable.

When the user fills in all information and files and presses the "Upload" button, the message "Uploaded successfully" will appear, then the data will be sent to the server through a POST request. If there is an error during this process, the error will be displayed in the console.



```
💡 Click here to ask Blackbox to help you code faster
<template>
  <div class="card" style="width: 79rem">
    <div class="card-body">
      <h5 class="card-title">List My Post</h5>
      <table class="table">
        <thead>
          <tr>
            <th scope="col">Post Name</th>
            <th scope="col">Topic Name</th>
            <th scope="col">Start Date</th>
            <th scope="col">End Date</th>
            <th scope="col">Description</th>
            <th></th>
          </tr>
        </thead>
        <tbody>
          <tr
            v-for="(item, index) in listpost"
            :value="item._id"
            :key="index"
            placeholder="Password"
          >
            <th>
              <p></p>
              {{ item.name }}
            </th>
            <th>
              <p></p>
              {{ item.topic_name }}
            </th>
            <td>
              <p></p>
              {{ item.submit_date }}
            </td>
            <td>
              <p></p>
              {{ item.updatedAt }}
            </td>
            <td>
              <p></p>
              {{ item.description }}
            </td>
            <td>
              <button
                type="button"
                class="btn btn-info"
                style="margin: 5px"
                v-on:click="deleteItem(item._id)"
              >
                Delete
              </button>
              <button
                type="button"
                class="btn btn-info"
                v-on:click="handleClick"
                style="margin: 5px"
                href="javascript:"
                @click="getdownload(item._id)"
              >
                Download File
              </button>
              <button
                type="button"
                class="btn btn-info"
                style="margin: 5px"
                @click="viewComment(item._id)"
              >
                View Comment
              </button>
            </td>
          </tr>
        </tbody>
      </table>
    </div>
  </div>
</template>
```

Figure 7: Interface My Post

This snippet is part of the app's user interface, written in HTML and uses Vue.js to interact with data.

A div tag with class "card" is used to create a card interface to contain the content of the post list table.

The title of the card tag is placed in an h5 tag with the class "card-title". In this case, the title is "List My Post".

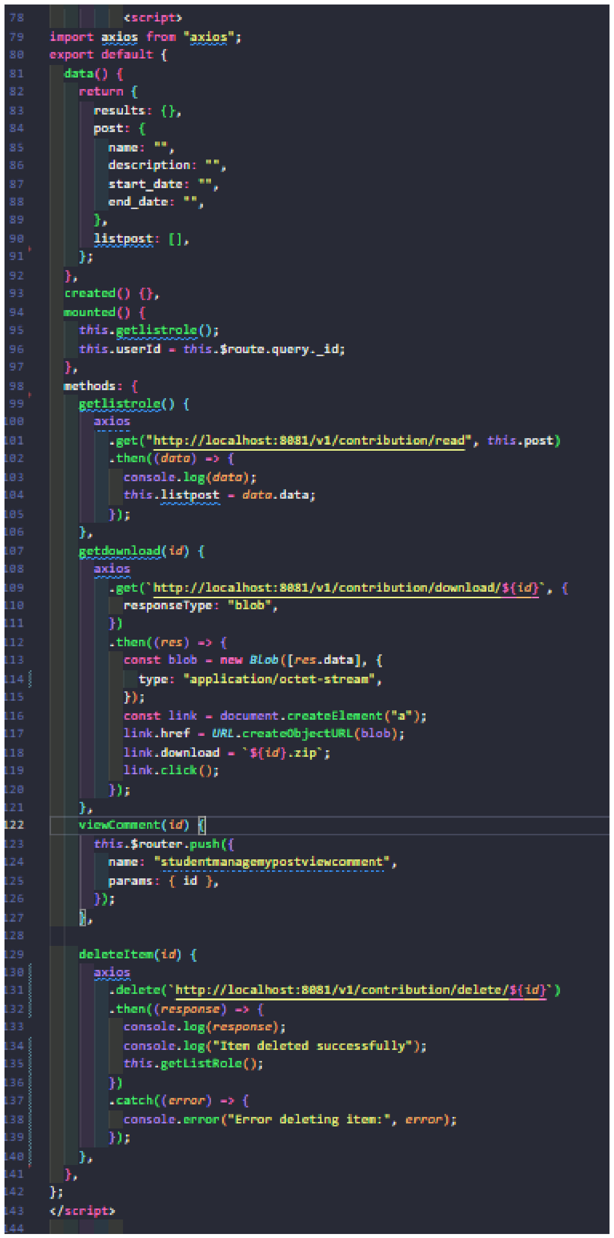
The HTML table is created using the table tag, with columns "Post Name", "Topic Name", "Start Date", "End Date", "Description" and an empty column to contain function buttons.

The code uses v-for to loop through each post in the listpost array and display each post's information in a single line (<tr>).

Each post will have columns corresponding to its information such as post name, topic name, start date, end date, description.

Function buttons are displayed in the last column of each row, including a "Delete" button to delete the post, a "Download File" button to download the post's attachment, and a "View Comment" button to view comments comment of the post.

Each button has a v-on:click or @click event that calls the corresponding Vue methods when the user interacts with the button.



```
<script>
import axios from "axios";
export default {
  data() {
    return {
      results: {},
      post: {
        name: "",
        description: "",
        start_date: "",
        end_date: "",
      },
      listpost: [],
    };
  },
  created() {},
  mounted() {
    this.getlistrole();
    this.userId = this.$route.query._id;
  },
  methods: {
    getlistrole() {
      axios
        .get("http://localhost:8081/v1/contribution/read", this.post)
        .then((data) => {
          console.log(data);
          this.listpost = data.data;
        });
    },
    getdownload(id) {
      axios
        .get(`http://localhost:8081/v1/contribution/download/${id}`, {
          responseType: "blob",
        })
        .then((res) => {
          const blob = new Blob([res.data], {
            type: "application/octet-stream",
          });
          const link = document.createElement("a");
          link.href = URL.createObjectURL(blob);
          link.download = `${id}.zip`;
          link.click();
        });
    },
    viewComment(id) {
      this.$router.push({
        name: "studentmanagemypostviewcomment",
        params: { id },
      });
    },

    deleteItem(id) {
      axios
        .delete(`http://localhost:8081/v1/contribution/delete/${id}`)
        .then((response) => {
          console.log(response);
          console.log("Item deleted successfully");
          this.getListRole();
        })
        .catch((error) => {
          console.error("Error deleting item:", error);
        });
    },
  },
};
</script>
```

Figure 8: getlistrole, getdownload, viewComment, deleteItem function my post

This snippet is a Vue component used to manage posts, including functions like displaying post list, downloading files, viewing comments, and deleting posts.

Data declaration section:

results: An object used to store results from the API. However, in this code, this data is not used.

post: An object to hold information about the post, including name, description, start date, and end date.

listpost: An array to store a list of posts.

mounted() method:

In this method, two actions are performed:

Call the getlistrole() function: This function calls the API to get the list of posts and assigns the results to the listpost variable.

Assign the value of the _id query parameter from the URL route to the userId variable.

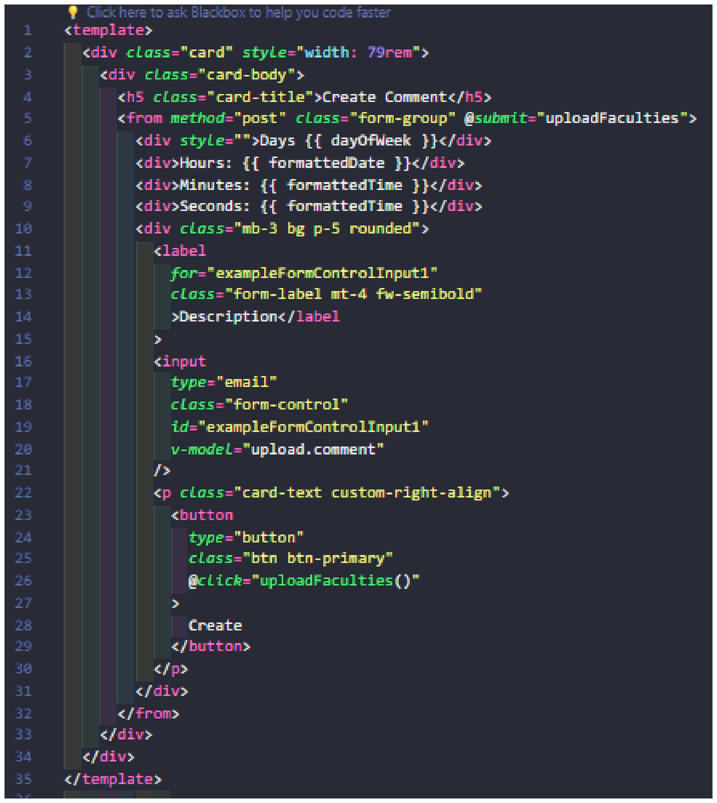
Methods:

getlistrole(): Call the API to get a list of posts from the server and assign the results to the listpost variable.

getdownload(id): Call the API to download a file based on the post's ID. When the data is downloaded, it is processed to create a Blob object and create a URL to download the file.

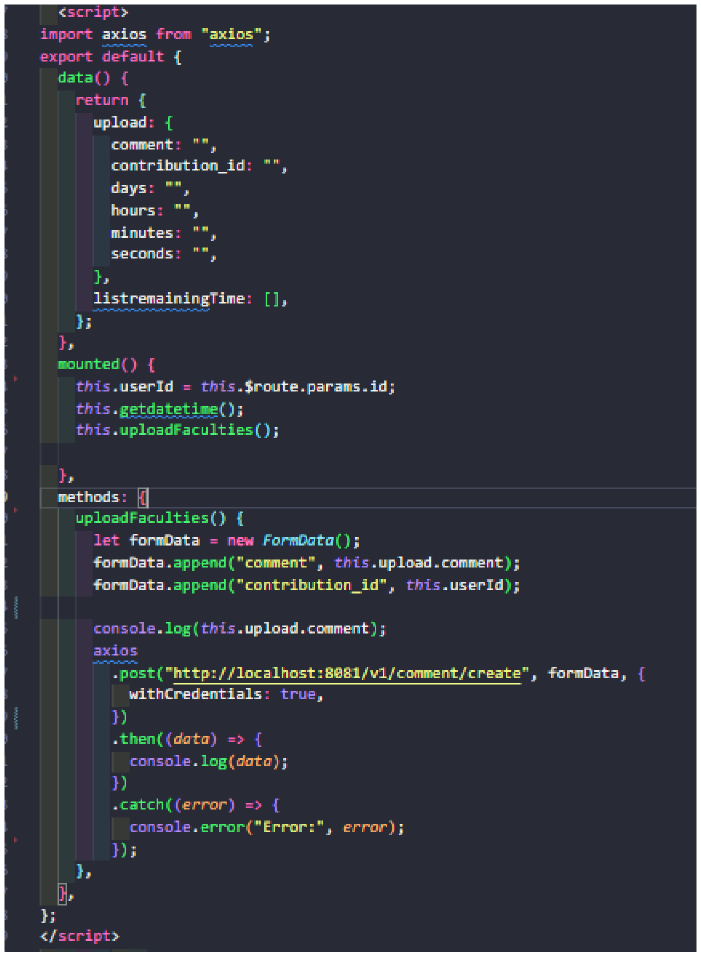
viewComment(id): Redirects users to the comments view page for a specific post, based on the post's ID.

deleteItem(id): Call the API to delete a post based on its ID. After successful deletion, the post list is updated by calling the getlistrole() function again. If an error occurs during deletion, it will be displayed in the console.



```
<template>
  <div class="card" style="width: 79rem">
    <div class="card-body">
      <h5 class="card-title">Create Comment</h5>
      <from method="post" class="form-group" @submit="uploadFaculties">
        <div style="">Days {{ dayOfWeek }}</div>
        <div>Hours: {{ formattedDate }}</div>
        <div>Minutes: {{ formattedTime }}</div>
        <div>Seconds: {{ formattedTime }}</div>
        <div class="mb-3 bg p-5 rounded">
          <label
            for="exampleFormControlInput1"
            class="form-label mt-4 fw-semibold"
            >Description</label
          >
          <input
            type="email"
            class="form-control"
            id="exampleFormControlInput1"
            v-model="upload.comment"
          />
          <p class="card-text custom-right-align">
            <button
              type="button"
              class="btn btn-primary"
              @click="uploadFaculties()"
            >
              Create
            </button>
          </p>
        </div>
      </from>
    </div>
  </div>
</template>
```

Figure 9: Interface Input Comment



```
<script>
import axios from "axios";
export default {
  data() {
    return {
      upload: {
        comment: "",
        contribution_id: "",
        days: "",
        hours: "",
        minutes: "",
        seconds: "",
      },
      listremainingTime: [],
    };
  },
  mounted() {
    this.userId = this.$route.params.id;
    this.getdatetime();
    this.uploadFaculties();

  },
  methods: {
    uploadFaculties() {
      let formData = new FormData();
      formData.append("comment", this.upload.comment);
      formData.append("contribution_id", this.userId);

      console.log(this.upload.comment);
      axios
        .post("http://localhost:8081/v1/comment/create", formData, {
          withCredentials: true,
        })
        .then((data) => {
          console.log(data);
        })
        .catch((error) => {
          console.error("Error:", error);
        });
    },
  },
};
</script>
```

Figure 10: UploadComment Function

let formData = new FormData();: Create a FormData object to contain the data of the comment to be sent.

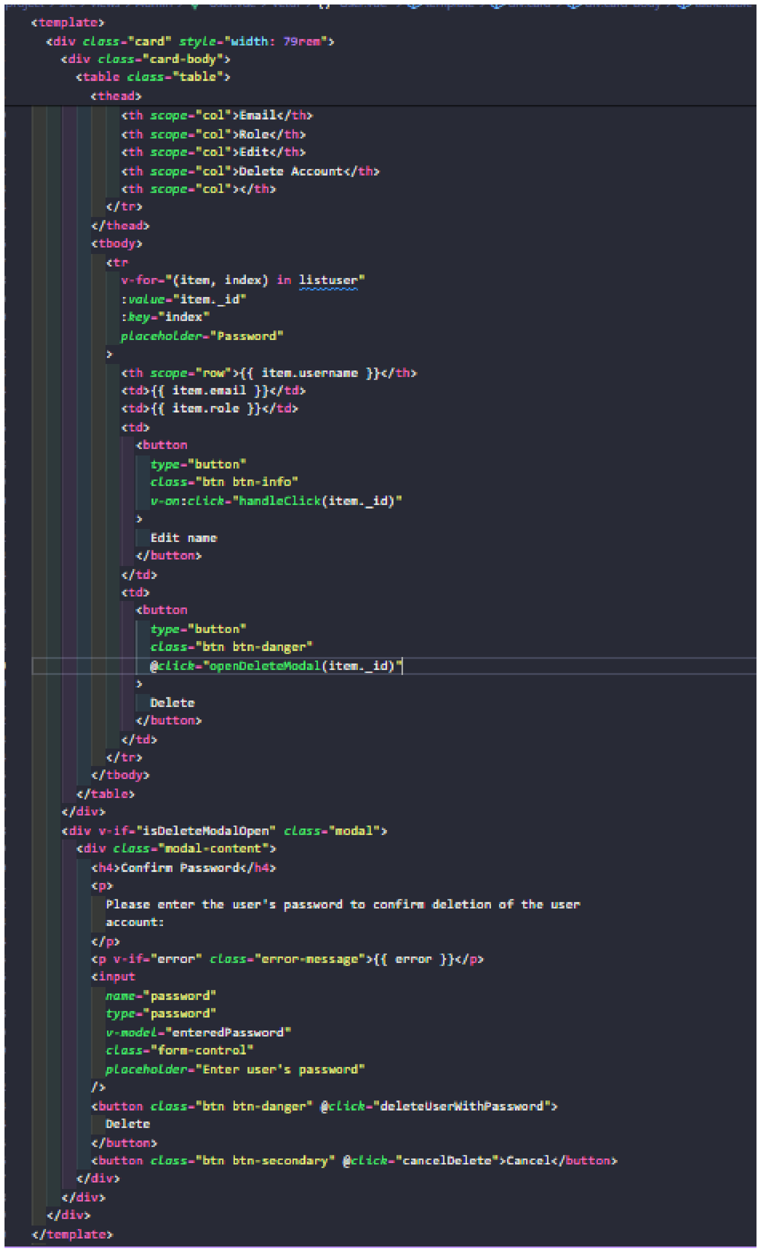
formData.append("comment", this.upload.comment);: Add the content of the comment (this.upload.comment) to FormData with the key "comment". This will ensure that comment data will be sent with the "comment" key when the POST request is made.

formData.append("contribution_id", this.userId);: Add the id of the post or element the comment belongs to to FormData with the key "contribution_id". This will ensure that the comment will be linked to the corresponding post or element.

axios.post("http://localhost:8081/v1/comment/create", formData, { withCredentials: true }): Send a POST request to the address "http://localhost:8081/v1/comment/ create", with the data encapsulated in formData. The { withCredentials: true } parameter is used to ensure that credentials (if any) are sent with the request.

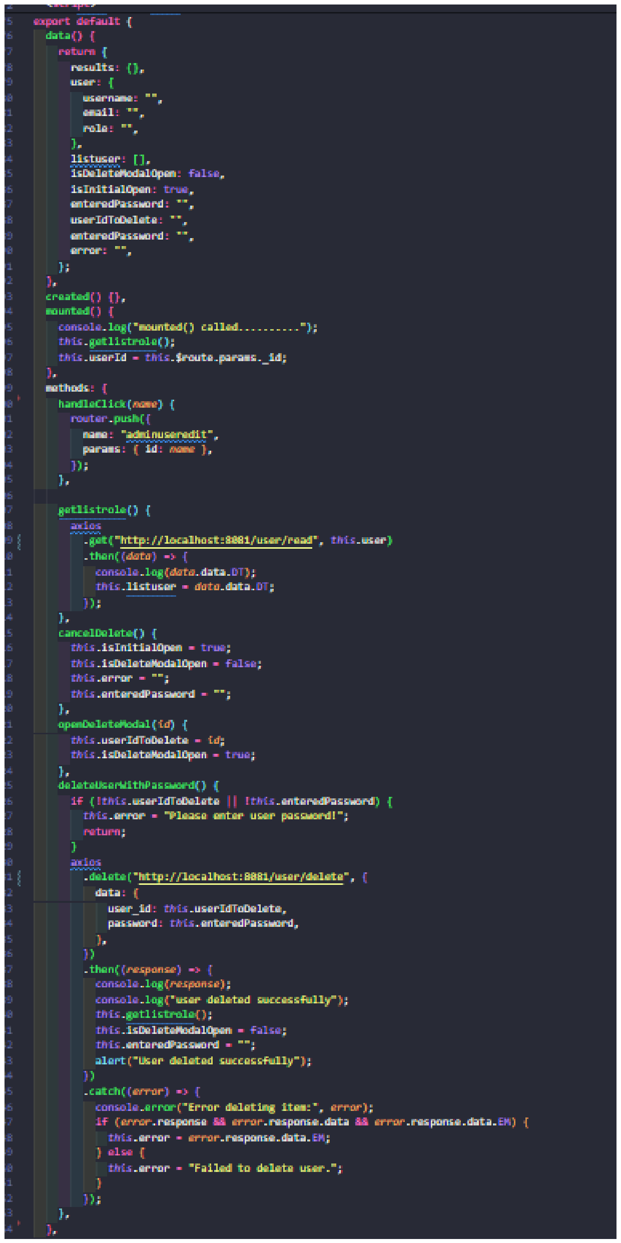
.then((data) => { console.log(data); }): Handles the results returned from the POST request. In this case, the returned data is printed to the console for testing.

.catch((error) => { console.error("Error:", error); }): Handle errors if any by printing error messages to the console.



```
<template>
  <div class="card" style="width: 79rem">
    <div class="card-body">
      <table class="table">
        <thead>
            <th scope="col">Email</th>
            <th scope="col">Role</th>
            <th scope="col">Edit</th>
            <th scope="col">Delete Account</th>
            <th scope="col"></th>
          </tr>
        </thead>
        <tbody>
          <tr
            v-for="(item, index) in listuser"
            :value="item._id"
            :key="index"
            placeholder="Password"
          >
            <th scope="row">{{ item.username }}</th>
            <td>{{ item.email }}</td>
            <td>{{ item.role }}</td>
            <td>
              <button
                type="button"
                class="btn btn-info"
                v-on:click="handleClick(item._id)"
              >
                Edit name
              </button>
            </td>
            <td>
              <button
                type="button"
                class="btn btn-danger"
                @click="openDeleteModal(item._id)"
              >
                Delete
              </button>
            </td>
          </tr>
        </tbody>
      </table>
    </div>
    <div v-if="isDeleteModalOpen" class="modal">
      <div class="modal-content">
        <h4>Confirm Password</h4>
        <p>
          Please enter the user's password to confirm deletion of the user
          account:
        </p>
        <p v-if="error" class="error-message">{{ error }}</p>
        <input
          name="password"
          type="password"
          v-model="enteredPassword"
          class="form-control"
          placeholder="Enter user's password"
        />
        <button class="btn btn-danger" @click="deleteUserWithPassword">
          Delete
        </button>
        <button class="btn btn-secondary" @click="cancelDelete">Cancel</button>
      </div>
    </div>
  </div>
</template>
```

Figure 11: Interface list account confirm Password



```
export default {
  data() {
    return {
      results: {},
      user: {
        username: "",
        email: "",
        role: "",
      },
      listuser: [],
      isDeleteModalOpen: false,
      isInitialOpen: true,
      enteredPassword: "",
      userIdToDelete: "",
      enteredPassword: "",
      error: "",
    };
  },
  created() {},
  mounted() {
    console.log("mounted() called..........");
    this.getlistrole();
    this.userId = this.$route.params._id;
  },
  methods: {
    handleClick(name) {
      router.push({
        name: "adminuseredit",
        params: { id: name },
      });
    },

    getlistrole() {
      axios
        .get("http://localhost:8081/user/read", this.user)
        .then((data) => {
          console.log(data.data.DT);
          this.listuser = data.data.DT;
        });
    },
    cancelDelete() {
      this.isInitialOpen = true;
      this.isDeleteModalOpen = false;
      this.error = "";
      this.enteredPassword = "";
    },
    openDeleteModal(id) {
      this.userIdToDelete = id;
      this.isDeleteModalOpen = true;
    },
    deleteUserWithPassword() {
      if (!this.userIdToDelete || !this.enteredPassword) {
        this.error = "Please enter user password!";
        return;
      }
      axios
        .delete("http://localhost:8081/user/delete", {
          data: {
            user_id: this.userIdToDelete,
            password: this.enteredPassword,
          },
        })
        .then((response) => {
          console.log(response);
          console.log("user deleted successfully");
          this.getlistrole();
          this.isDeleteModalOpen = false;
          this.enteredPassword = "";
          alert("User deleted successfully");
        })
        .catch((error) => {
          console.error("Error deleting item:", error);
          if (error.response && error.response.data && error.response.data.EM) {
            this.error = error.response.data.EM;
          } else {
            this.error = "Failed to delete user.";
          }
        });
    },
  },
```

Figure 12: handleClick, cancelDelete, OpenDeleteModal, DeleteUserWithPassword function

The handleClick(name) method is used to redirect the user from one page to another. In this case, it is used to redirect users to the admin's user information edit page when they click on a specific user in the list.

router.push({ name: "adminuseredit", params: { id: name } }): Uses Vue Router's router object to perform redirects. The push() method is used to add a new browsing history to the application's browsing history, and navigate the user to a specific address. In this case:

name: "adminuseredit": Specifies the name of the route I want to redirect users to. In this case, it's the admin user information editing page.

params: { id: name }: Pass a parameter to the route, in this case the id of the specific user the user clicked on. Through params, I can pass data from one page to another for use when needed, for example displaying user details that the user has chosen to edit.

The cancelDelete() method is designed to cancel item deletion. When an admin wants to delete an item, a confirmation dialog will appear to ensure that they don't accidentally delete the item. If admin wants to cancel the deletion action, they can use this cancelDelete() method.

this.isInitialOpen = true;: Set the value of the isInitialOpen variable to true. It can be assumed that this variable is used to control the display of the delete dialog or display of other content on the user interface.

this.isDeleteModalOpen = false;: Set the value of the isDeleteModalOpen variable to false. Let's assume that this variable is used to control showing or hiding the delete dialog on the user interface.

this.error = "";: Set the value of the error variable to an empty string. It can be assumed that this variable is used to display error messages or information related to the deletion of a certain item.

this.enteredPassword = "";: Set the value of the enteredPassword variable to an empty string. Suppose that this variable is used to store the password that the user enters to confirm the deletion action.

The openDeleteModal(id) method is designed to open the confirmation modal to delete an item. When an admin wants to delete an item, this method is called to display the deletion confirmation modal and allow the admin to confirm or cancel the deletion action.

this.userIdToDelete = id;: Assign the id value of the item to be deleted to the variable userIdToDelete. Suppose that this variable is used to store the ID of the item to be deleted, so that it can later be used in confirming the delete or non-delete action.

this.isDeleteModalOpen = true;: Set the variable isDeleteModalOpen to true. Let's assume that this variable is used to control showing or hiding the delete confirmation modal on the user interface. When set to true, a modal will be displayed, allowing the user to confirm or cancel the deletion action.

The deleteUserWithPassword() method is designed to delete a user from the system providing a password confirmation. This method is usually called when the user confirms the deletion of a user via the confirmation modal.

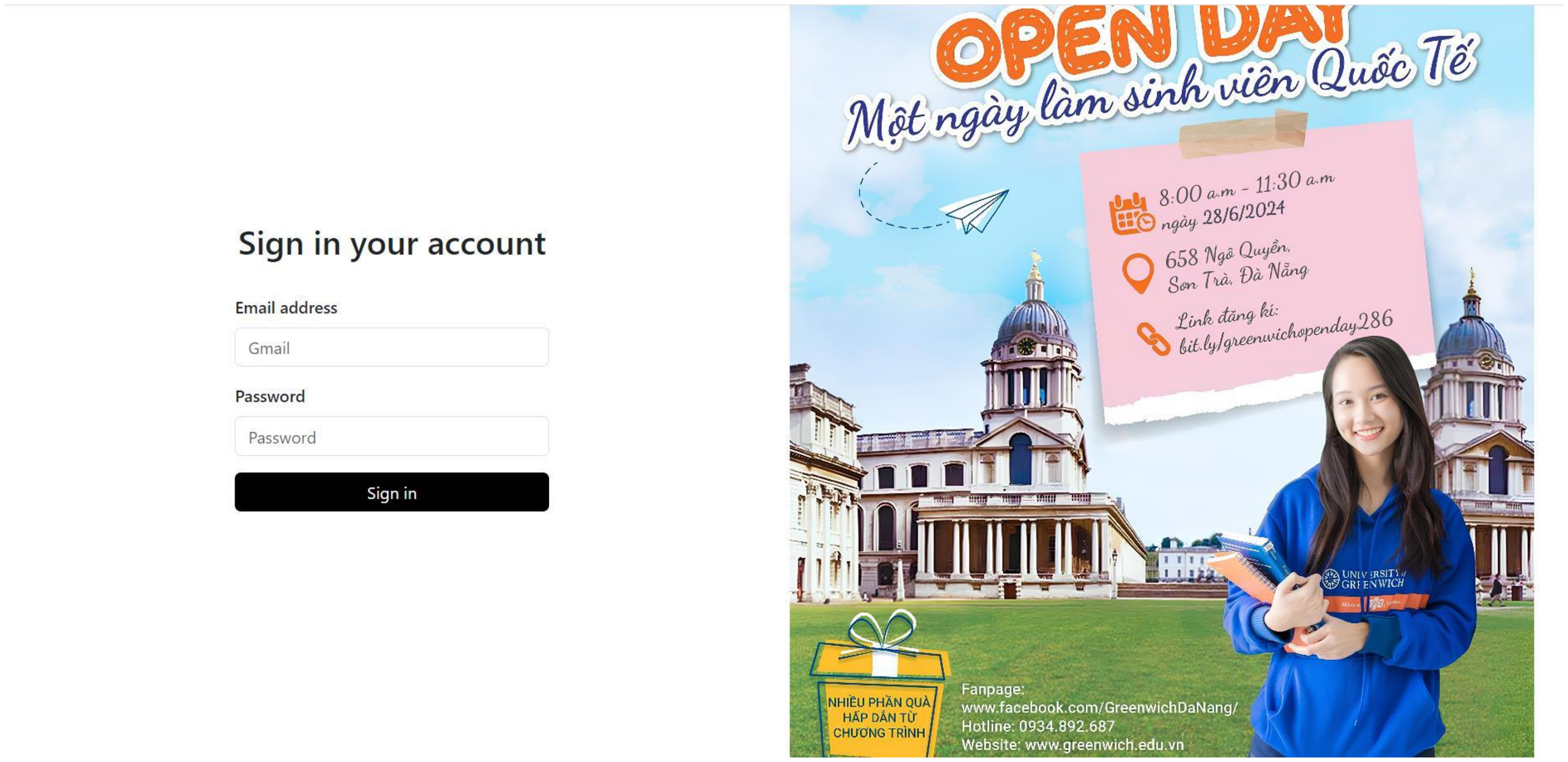
Check if userIdToDelete and enteredPassword exist. If not, the method assigns an error message to the error variable and ends the function.

Use the Axios library to make an HTTP DELETE request to the "http://localhost:8081/user/delete" endpoint to delete the user. The data sent in the body of the request includes user_id (ID of the user to be deleted) and password (confirmation password).

If the deletion request is successful, display the message "User deleted successfully", and refresh the role list and close the deletion confirmation modal.

If an error occurs during deletion, the corresponding error message is displayed. If there is error data returned from the server response, the error message will be taken from the EM field of the error data. Otherwise, the default error message will be displayed.

System interface(Screen Shot)



Sign in your account

Email address

Gmail

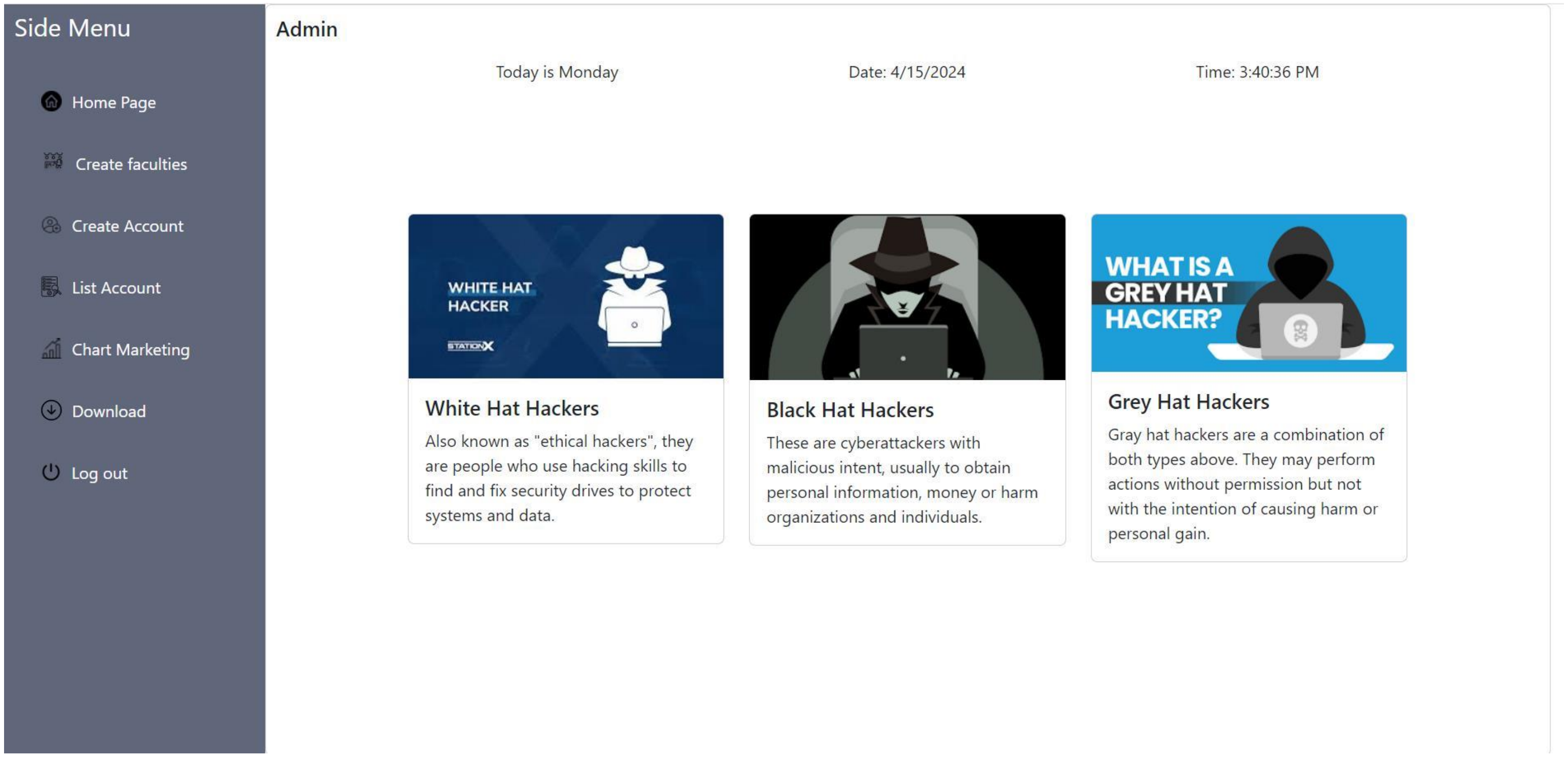
Password

Password

Sign in
OPEN DAY
Một ngày làm sinh viên Quốc Tế
8:00 a.m - 11:30 a.m
ngày 28/6/2024
658 Ngô Quyền, Sơn Trà, Đà Nẵng
Link đăng kí: bit.ly/greenwichopenday286
NHIỀU PHẦN QUÀ HẤP DẪN TỪ CHƯƠNG TRÌNH
Fanpage: www.facebook.com/GreenwichDaNang/
Hotline: 0934.892.687
Website: www.greenwich.edu.vn

Figure 13: Login interface

This is the login page, users will log in when the admin grants an account


Side Menu

- Home Page
- Create faculties
- Create Account
- List Account
- Chart Marketing
- Download
- Log out

Admin

Today is Monday Date: 4/15/2024 Time: 3:40:36 PM

White Hat Hackers

Also known as "ethical hackers", they are people who use hacking skills to find and fix security drives to protect systems and data.

Black Hat Hackers

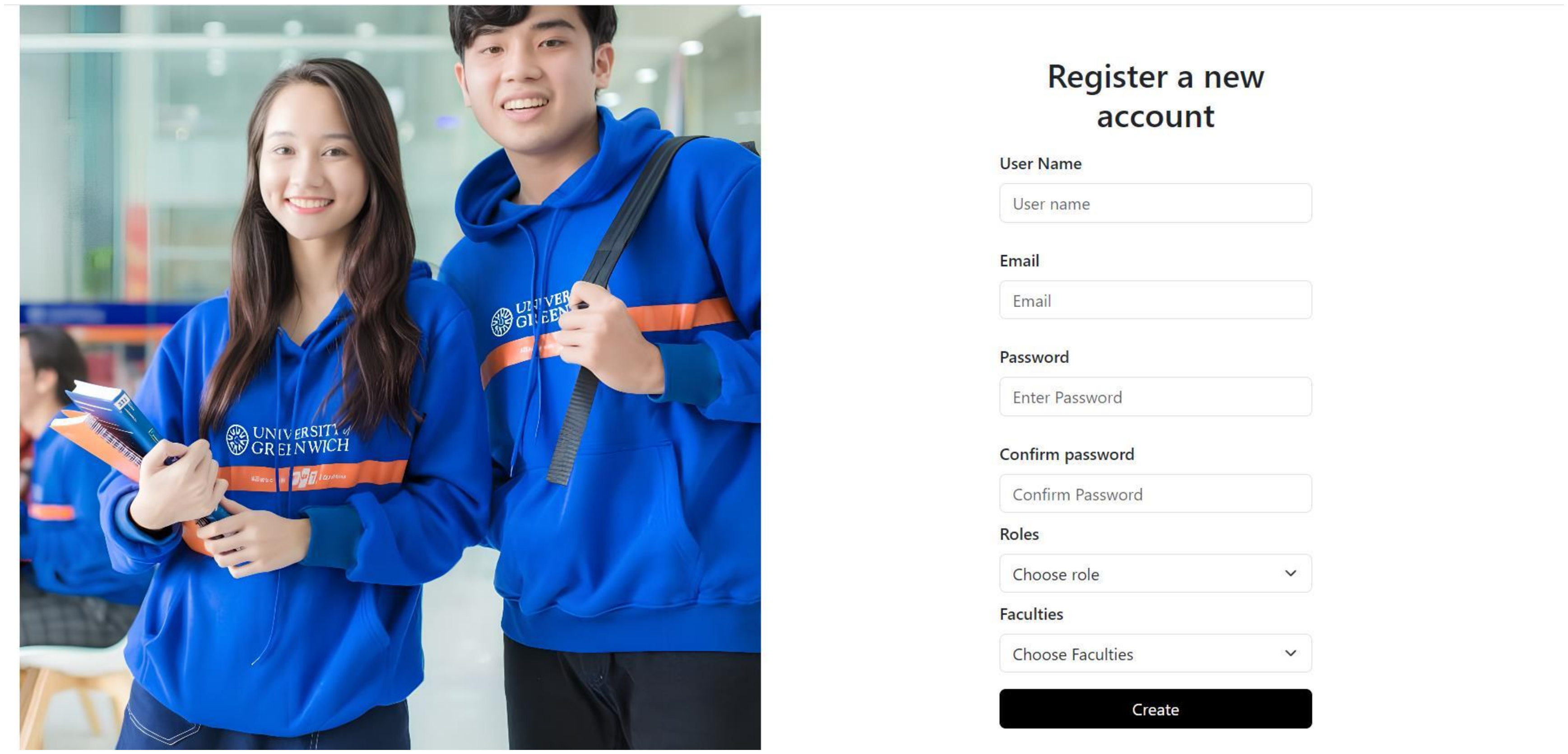
These are cyberattackers with malicious intent, usually to obtain personal information, money or harm organizations and individuals.

Grey Hat Hackers

Gray hat hackers are a combination of both types above. They may perform actions without permission but not with the intention of causing harm or personal gain.

Figure 14: Home page admin interface

Provide new information


Register a new account

User Name

User name

Email

Email

Password

Enter Password

Confirm password

Confirm Password

Roles

Choose role

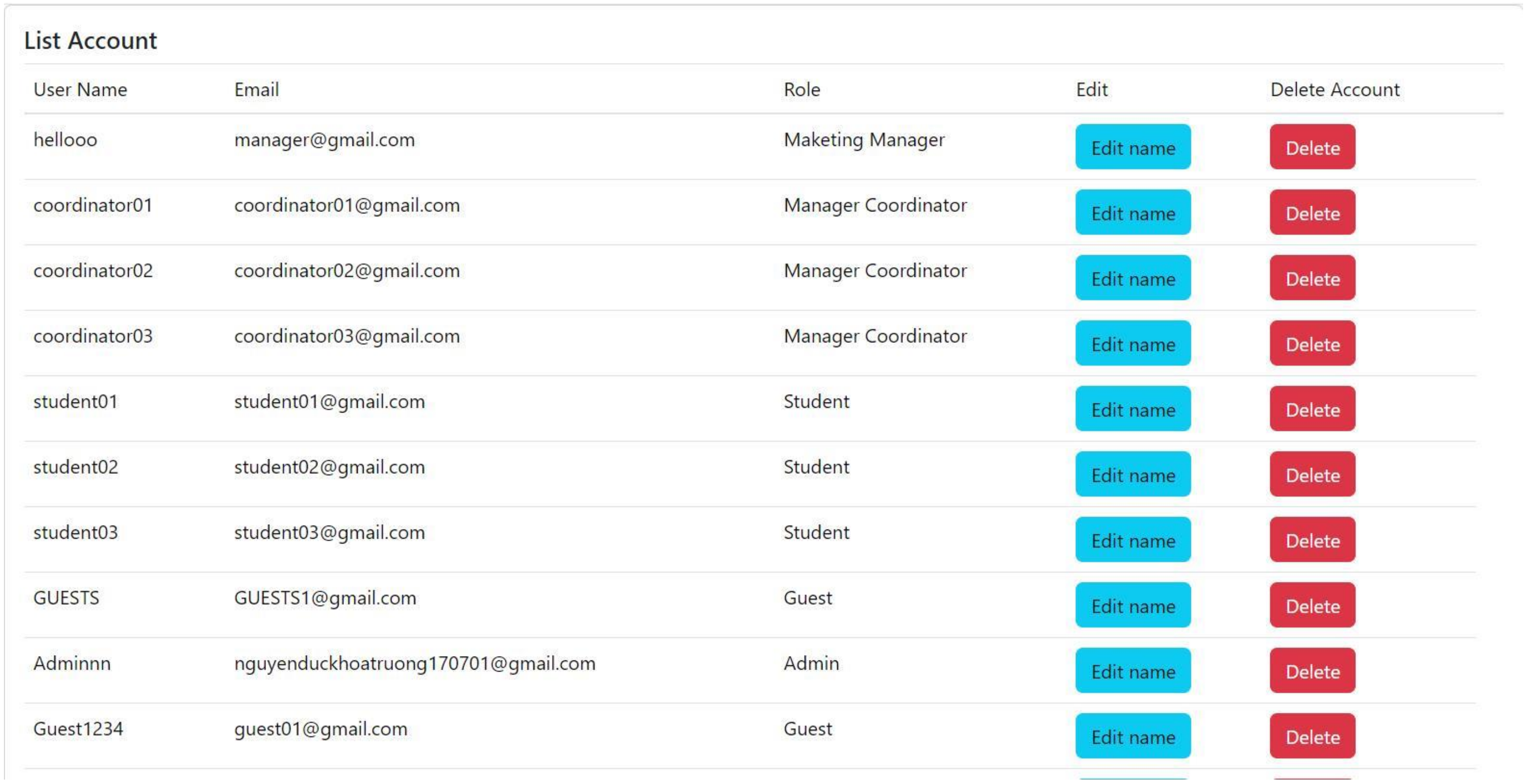
Faculties

Choose Faculties

Create

Figure 15: Create Acount

This account creation page will be created by the admin in which the admin will assign roles and Faculties.



List Account

User Name	Email	Role	Edit	Delete Account
hellooo	manager@gmail.com	Maketing Manager	Edit name	Delete
coordinator01	coordinator01@gmail.com	Manager Coordinator	Edit name	Delete
coordinator02	coordinator02@gmail.com	Manager Coordinator	Edit name	Delete
coordinator03	coordinator03@gmail.com	Manager Coordinator	Edit name	Delete
student01	student01@gmail.com	Student	Edit name	Delete
student02	student02@gmail.com	Student	Edit name	Delete
student03	student03@gmail.com	Student	Edit name	Delete
GUESTS	GUESTS1@gmail.com	Guest	Edit name	Delete
Adminnn	nguyenduckhoatruong170701@gmail.com	Admin	Edit name	Delete
Guest1234	guest01@gmail.com	Guest	Edit name	Delete

Figure 16: List Account

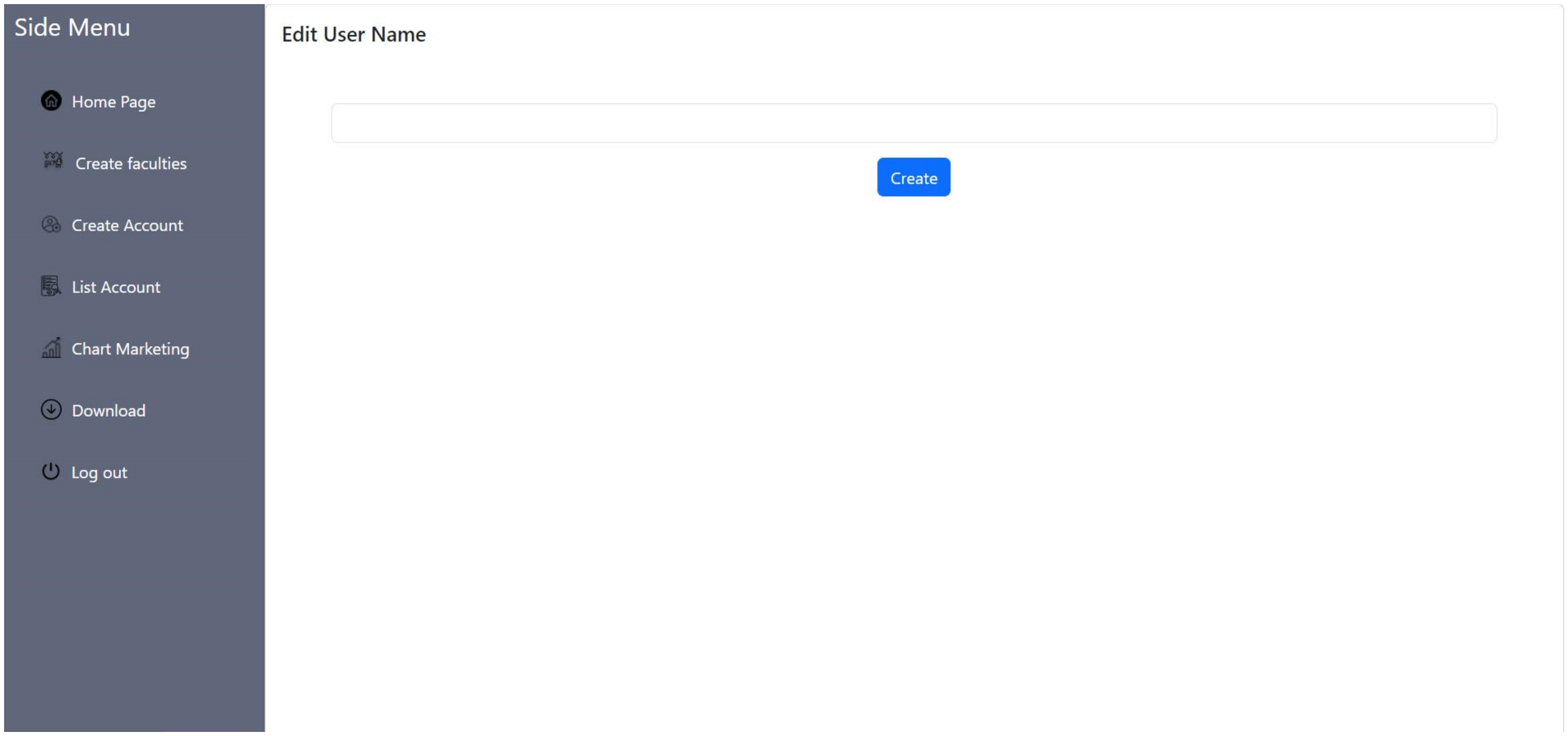
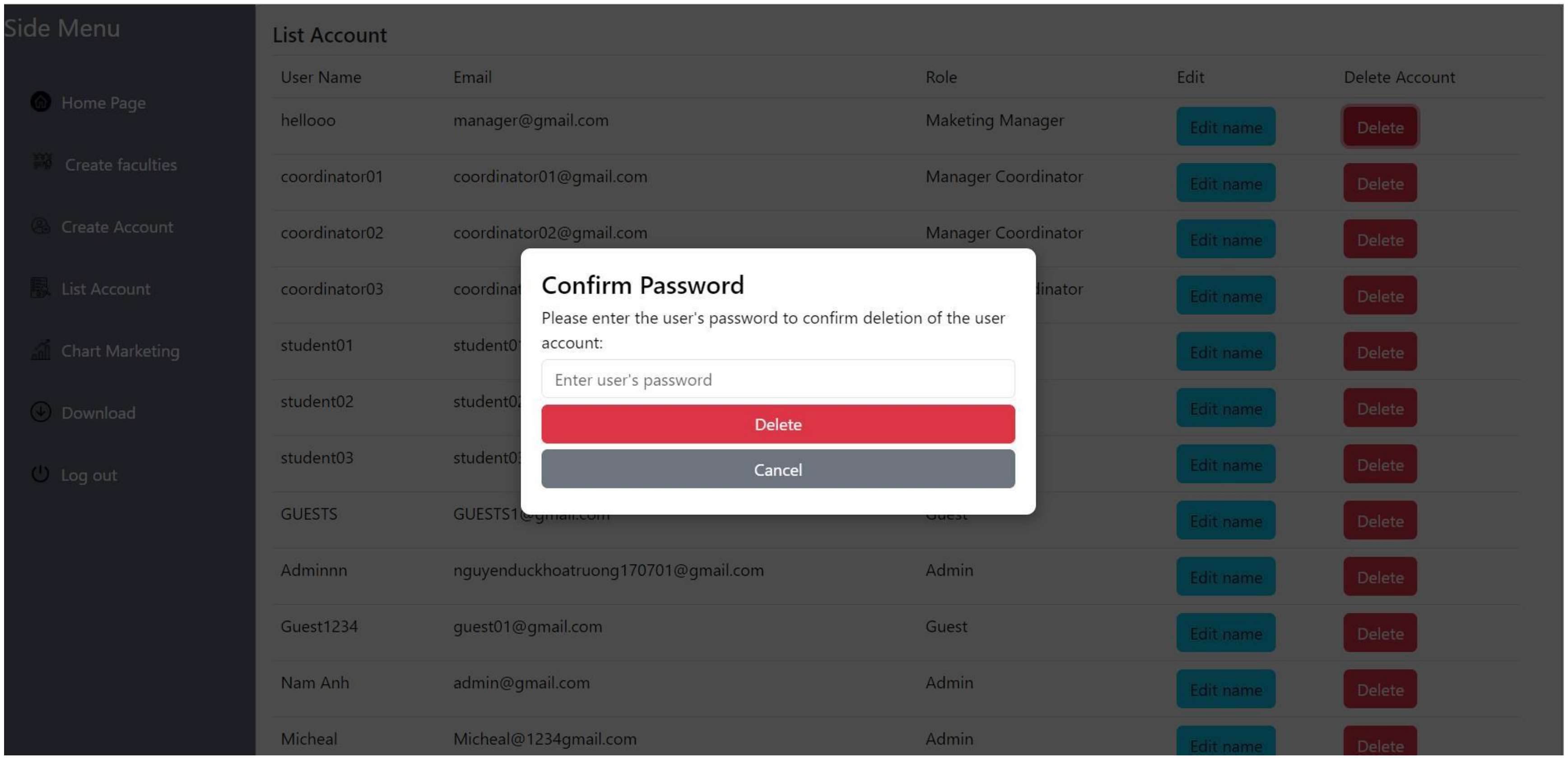


Figure 17: Edit user name



List Account

User Name	Email	Role	Edit	Delete Account
hellooo	manager@gmail.com	Maketing Manager	Edit name	Delete
coordinator01	coordinator01@gmail.com	Manager Coordinator	Edit name	Delete
coordinator02	coordinator02@gmail.com	Manager Coordinator	Edit name	Delete
coordinator03	coordinat	inator	Edit name	Delete
student01	student0		Edit name	Delete
student02	student0		Edit name	Delete
student03	student0		Edit name	Delete
GUESTS	GUESTS1@gmail.com	Guest	Edit name	Delete
Adminnn	nguyenduckhoatruong170701@gmail.com	Admin	Edit name	Delete
Guest1234	guest01@gmail.com	Guest	Edit name	Delete
Nam Anh	admin@gmail.com	Admin	Edit name	Delete
Micheal	Micheal@1234gmail.com	Admin	Edit name	Delete

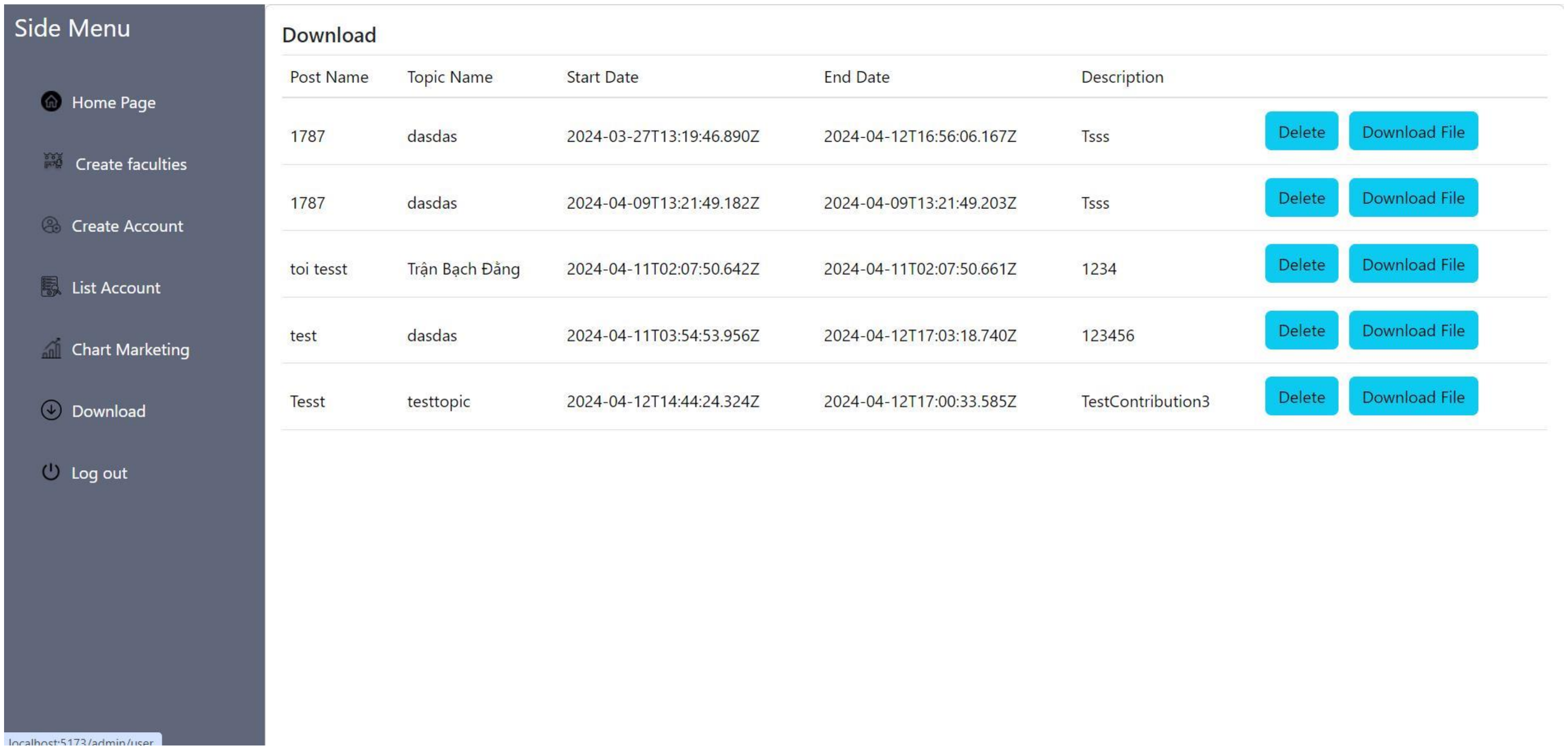
Confirm Password

Please enter the user's password to confirm deletion of the user account:

Enter user's password

Figure 18: Confirm Password

When the admin wants to delete an account, he or she must enter the user password before deleting it.

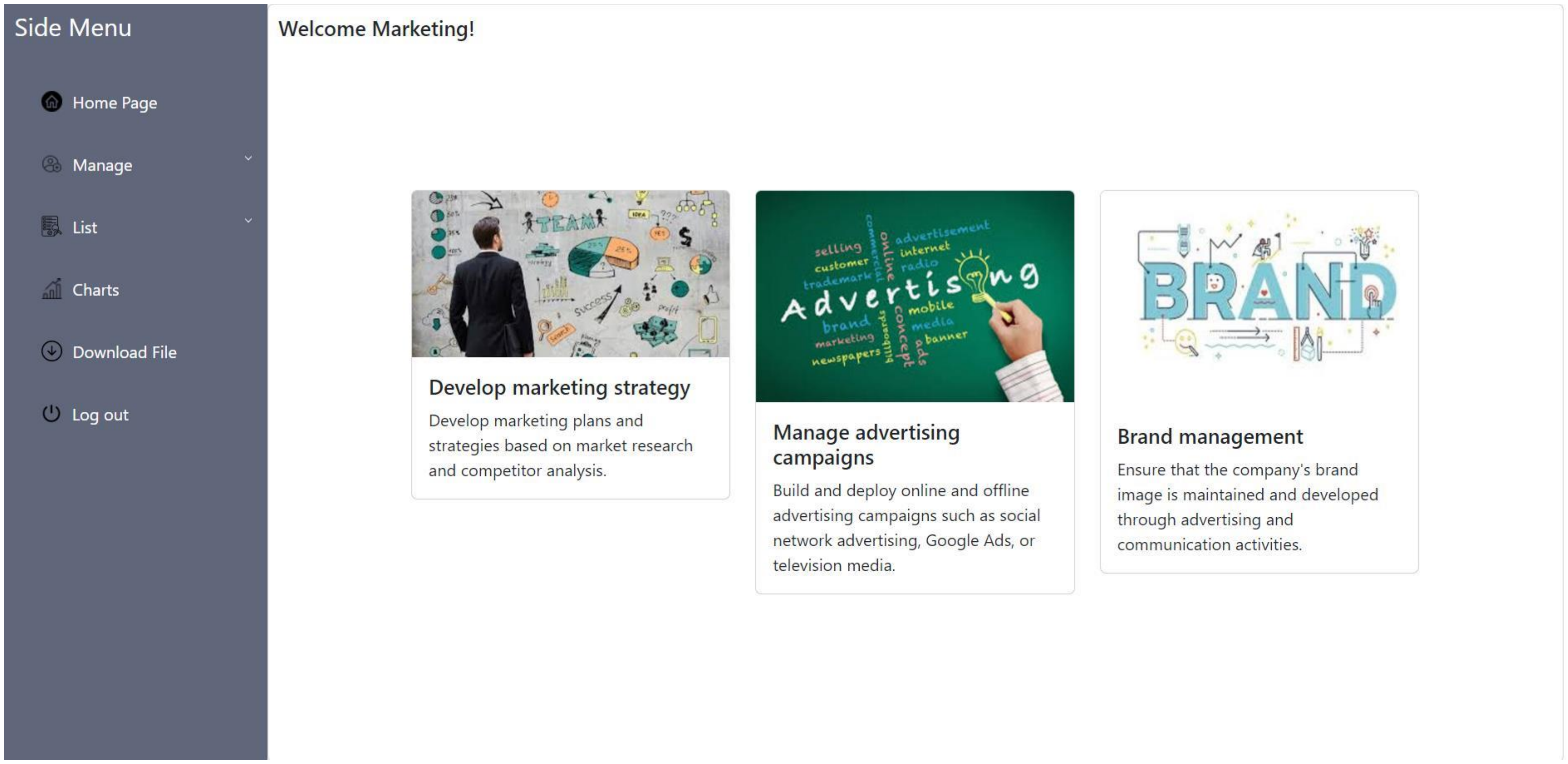


Download

Post Name	Topic Name	Start Date	End Date	Description	
1787	dasdas	2024-03-27T13:19:46.890Z	2024-04-12T16:56:06.167Z	Tsss	Delete Download File
1787	dasdas	2024-04-09T13:21:49.182Z	2024-04-09T13:21:49.203Z	Tsss	Delete Download File
toi tesst	Trận Bạch Đằng	2024-04-11T02:07:50.642Z	2024-04-11T02:07:50.661Z	1234	Delete Download File
test	dasdas	2024-04-11T03:54:53.956Z	2024-04-12T17:03:18.740Z	123456	Delete Download File
Tesst	testtopic	2024-04-12T14:44:24.324Z	2024-04-12T17:00:33.585Z	TestContribution3	Delete Download File

Figure 19: Download file

Admin can download student contributions



Welcome Marketing!

Develop marketing strategy

Develop marketing plans and strategies based on market research and competitor analysis.

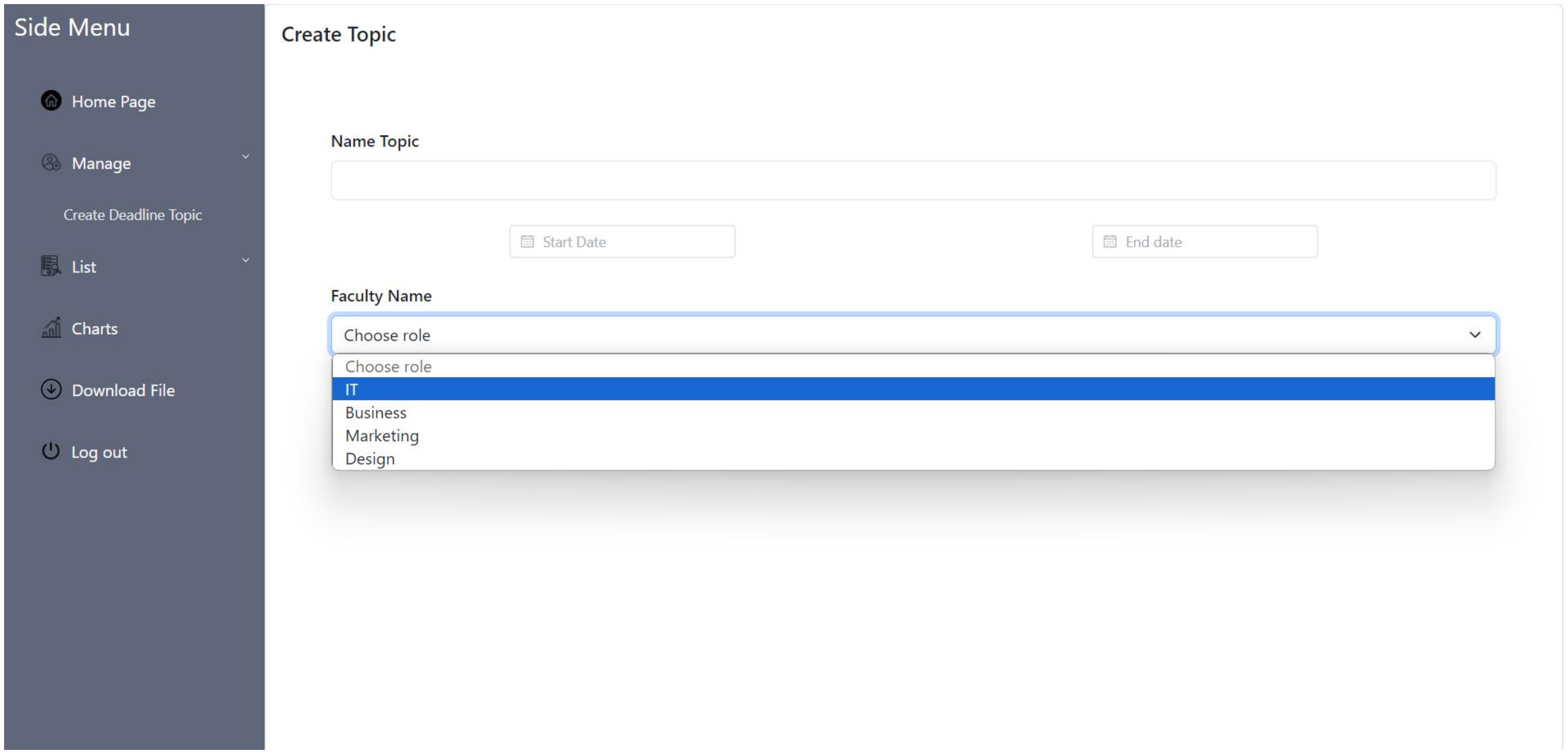
Manage advertising campaigns

Build and deploy online and offline advertising campaigns such as social network advertising, Google Ads, or television media.

Brand management

Ensure that the company's brand image is maintained and developed through advertising and communication activities.

Figure 20: Homepage manage



Create Topic

Name Topic

Start Date

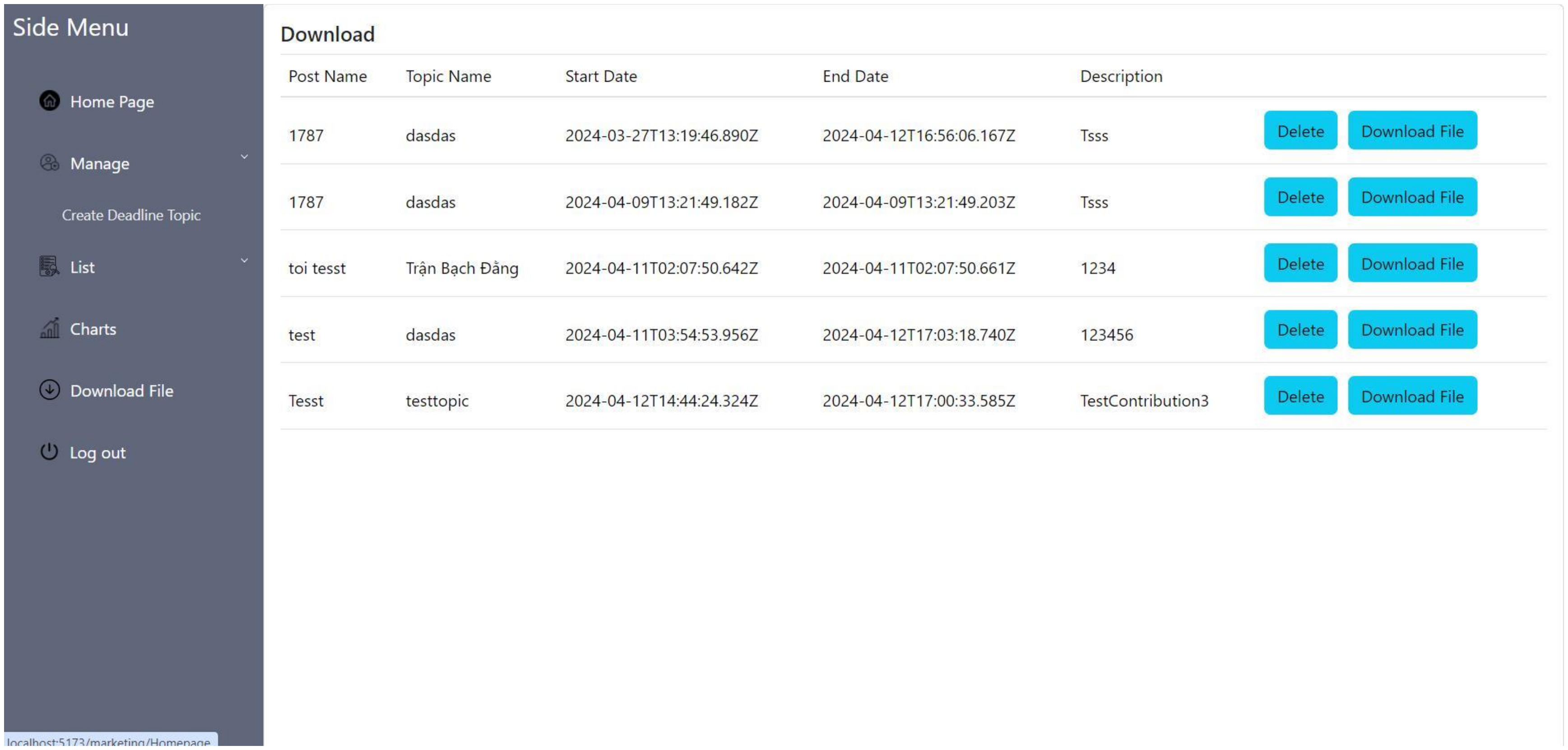
End date

Faculty Name

Choose role

- Choose role
- IT
- Business
- Marketing
- Design

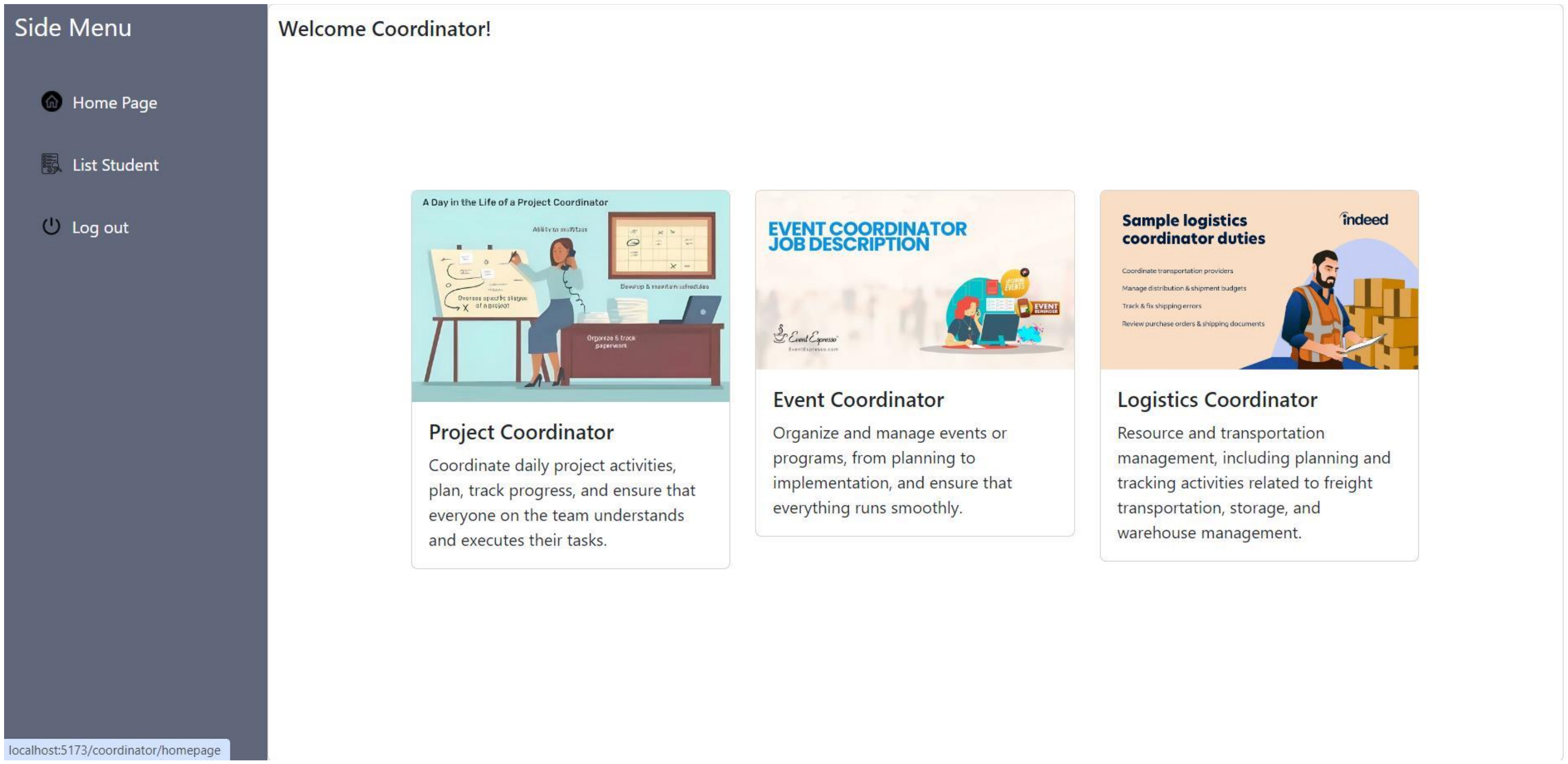
Figure 21: Marketing create topic



Download

Post Name	Topic Name	Start Date	End Date	Description	
1787	dasdas	2024-03-27T13:19:46.890Z	2024-04-12T16:56:06.167Z	Tsss	Delete Download File
1787	dasdas	2024-04-09T13:21:49.182Z	2024-04-09T13:21:49.203Z	Tsss	Delete Download File
toi tesst	Trận Bạch Đằng	2024-04-11T02:07:50.642Z	2024-04-11T02:07:50.661Z	1234	Delete Download File
test	dasdas	2024-04-11T03:54:53.956Z	2024-04-12T17:03:18.740Z	123456	Delete Download File
Tesst	testtopic	2024-04-12T14:44:24.324Z	2024-04-12T17:00:33.585Z	TestContribution3	Delete Download File

Figure 22: Dowload file marketing



Side Menu

Home Page

List Student

Log out

localhost:5173/coordinator/homepage

Welcome Coordinator!

A Day in the Life of a Project Coordinator

Project Coordinator

Coordinate daily project activities, plan, track progress, and ensure that everyone on the team understands and executes their tasks.

EVENT COORDINATOR JOB DESCRIPTION

Event Coordinator

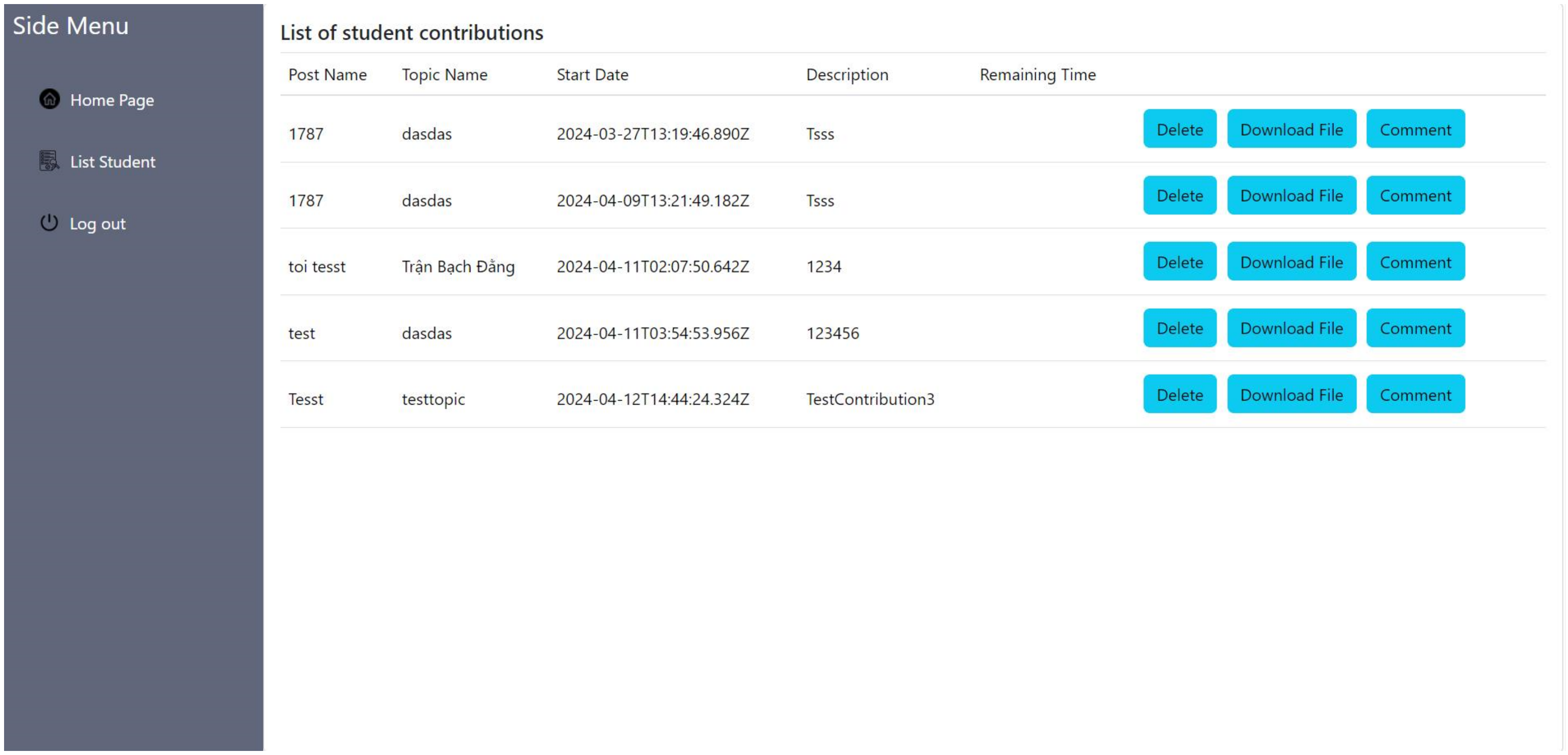
Organize and manage events or programs, from planning to implementation, and ensure that everything runs smoothly.

Sample logistics coordinator duties
indeed
Coordinate transportation providers
Manage distribution & shipment budgets
Track & fix shipping errors
Review purchase orders & shipping documents

Logistics Coordinator

Resource and transportation management, including planning and tracking activities related to freight transportation, storage, and warehouse management.

Figure 23: Home page Coordinator

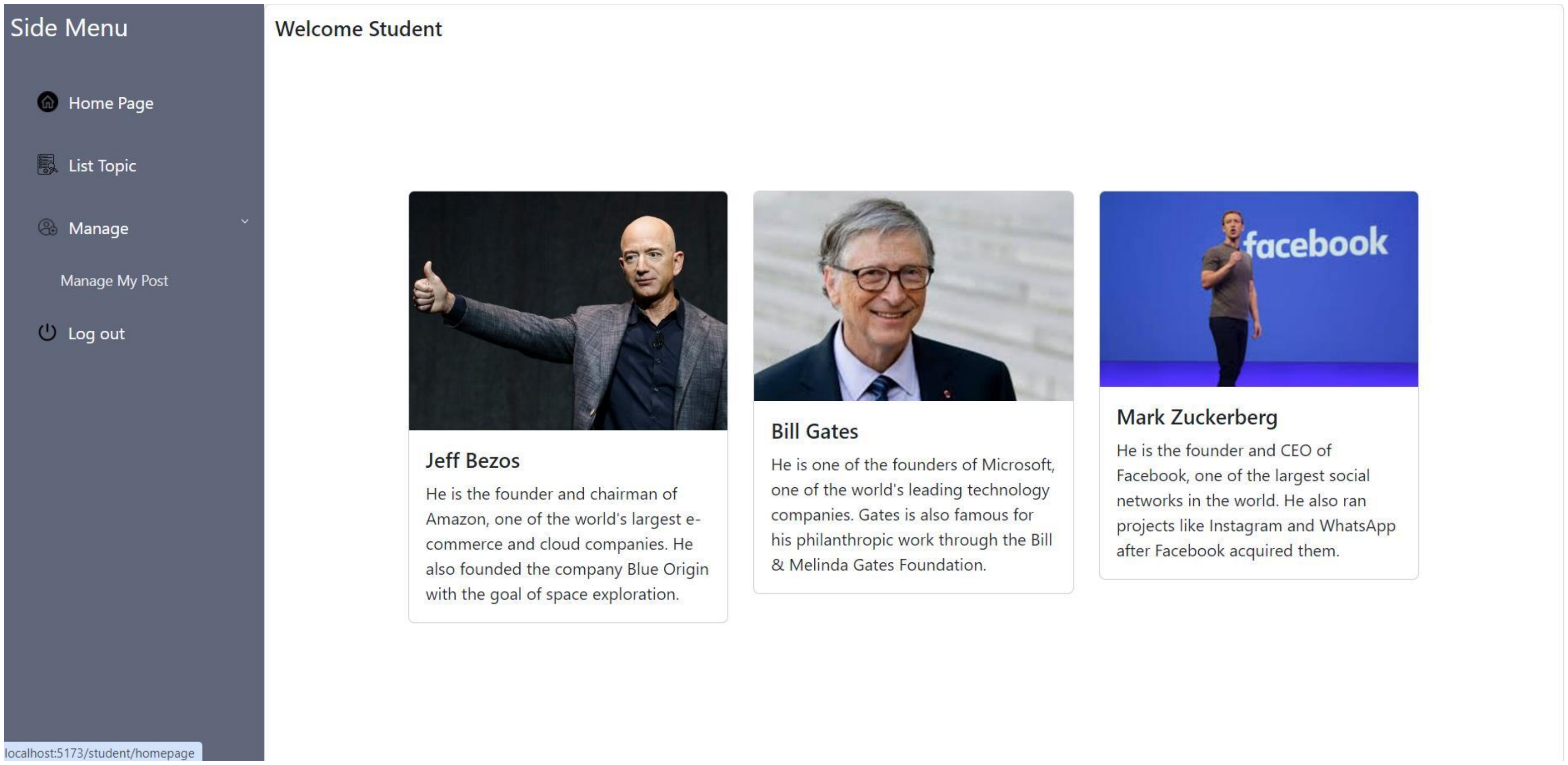


List of student contributions

Post Name	Topic Name	Start Date	Description	Remaining Time	
1787	dasdas	2024-03-27T13:19:46.890Z	Tsss		Delete Download File Comment
1787	dasdas	2024-04-09T13:21:49.182Z	Tsss		Delete Download File Comment
toi tesst	Trận Bạch Đằng	2024-04-11T02:07:50.642Z	1234		Delete Download File Comment
test	dasdas	2024-04-11T03:54:53.956Z	123456		Delete Download File Comment
Tesst	testtopic	2024-04-12T14:44:24.324Z	TestContribution3		Delete Download File Comment

Figure 24: List of student contribution

Coordinator can comment on student contributions



Welcome Student

Jeff Bezos

He is the founder and chairman of Amazon, one of the world's largest e-commerce and cloud companies. He also founded the company Blue Origin with the goal of space exploration.

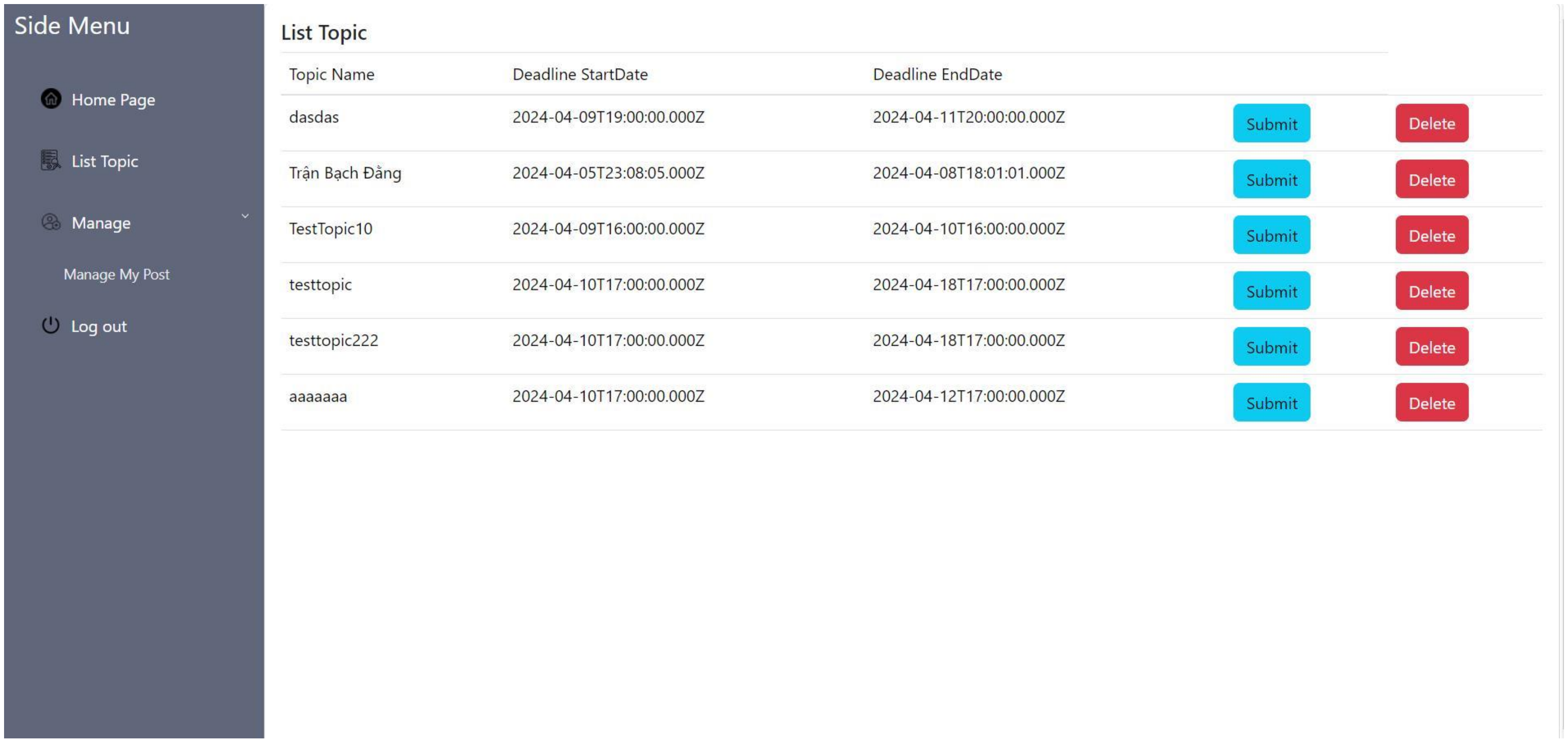
Bill Gates

He is one of the founders of Microsoft, one of the world's leading technology companies. Gates is also famous for his philanthropic work through the Bill & Melinda Gates Foundation.

Mark Zuckerberg

He is the founder and CEO of Facebook, one of the largest social networks in the world. He also ran projects like Instagram and WhatsApp after Facebook acquired them.

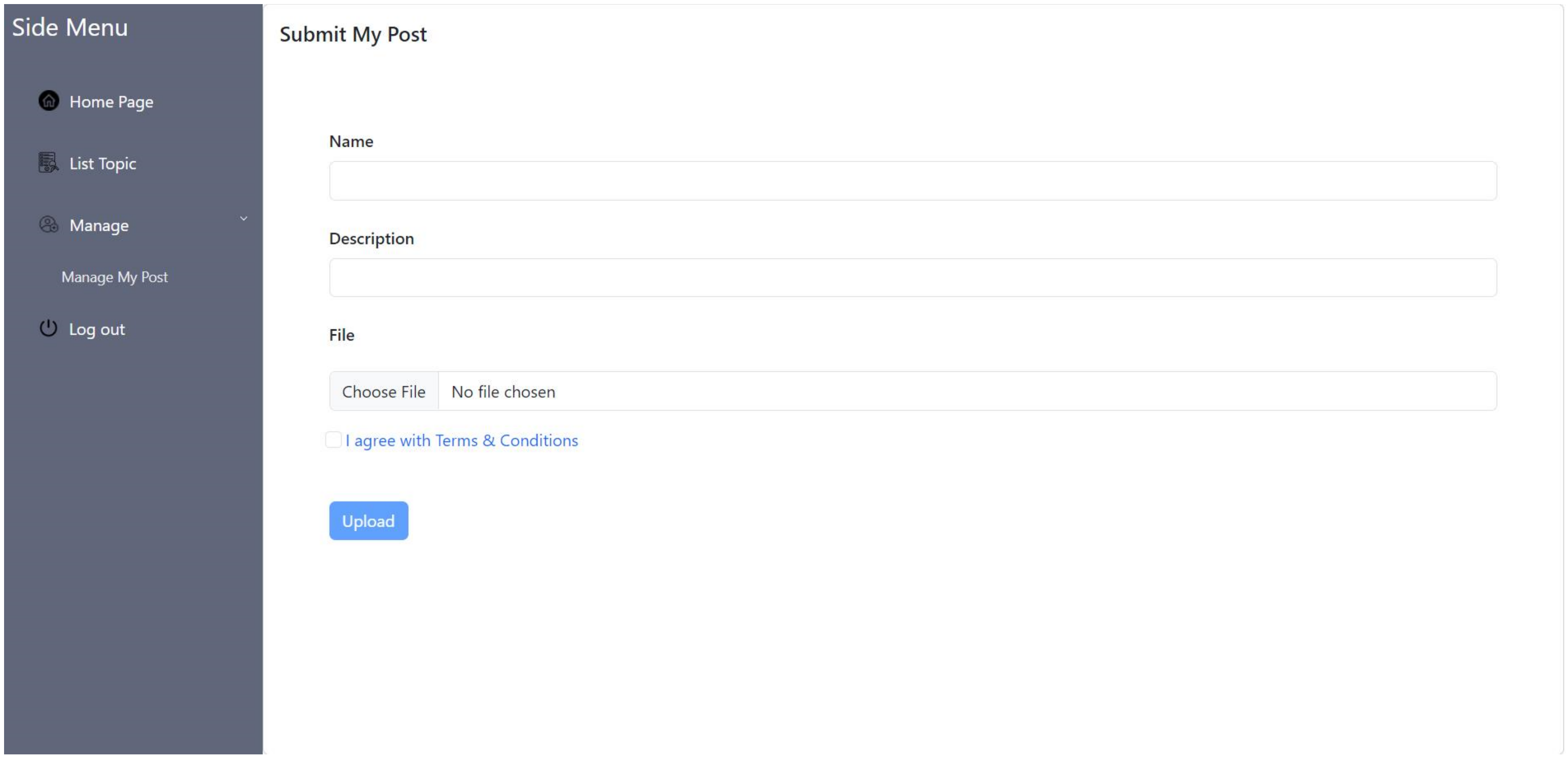
Figure 25: Home page Student



List Topic

Topic Name	Deadline StartDate	Deadline EndDate		
dasdas	2024-04-09T19:00:00.000Z	2024-04-11T20:00:00.000Z	Submit	Delete
Trận Bạch Đằng	2024-04-05T23:08:05.000Z	2024-04-08T18:01:01.000Z	Submit	Delete
TestTopic10	2024-04-09T16:00:00.000Z	2024-04-10T16:00:00.000Z	Submit	Delete
testtopic	2024-04-10T17:00:00.000Z	2024-04-18T17:00:00.000Z	Submit	Delete
testtopic222	2024-04-10T17:00:00.000Z	2024-04-18T17:00:00.000Z	Submit	Delete
aaaaaaa	2024-04-10T17:00:00.000Z	2024-04-12T17:00:00.000Z	Submit	Delete

Figure 26: List Topic



Submit My Post

Name

Description

File

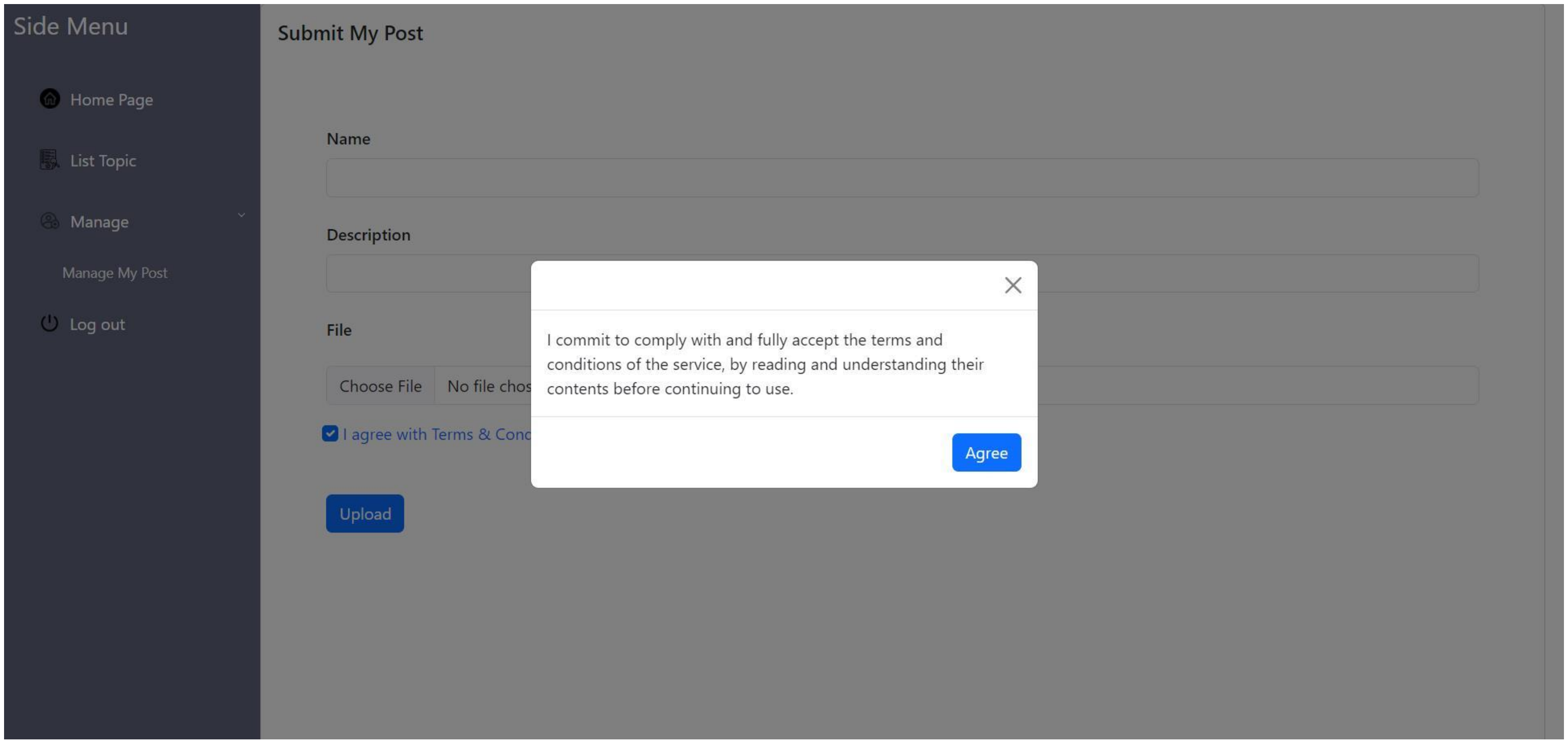
Choose File No file chosen

I agree with Terms & Conditions

Upload

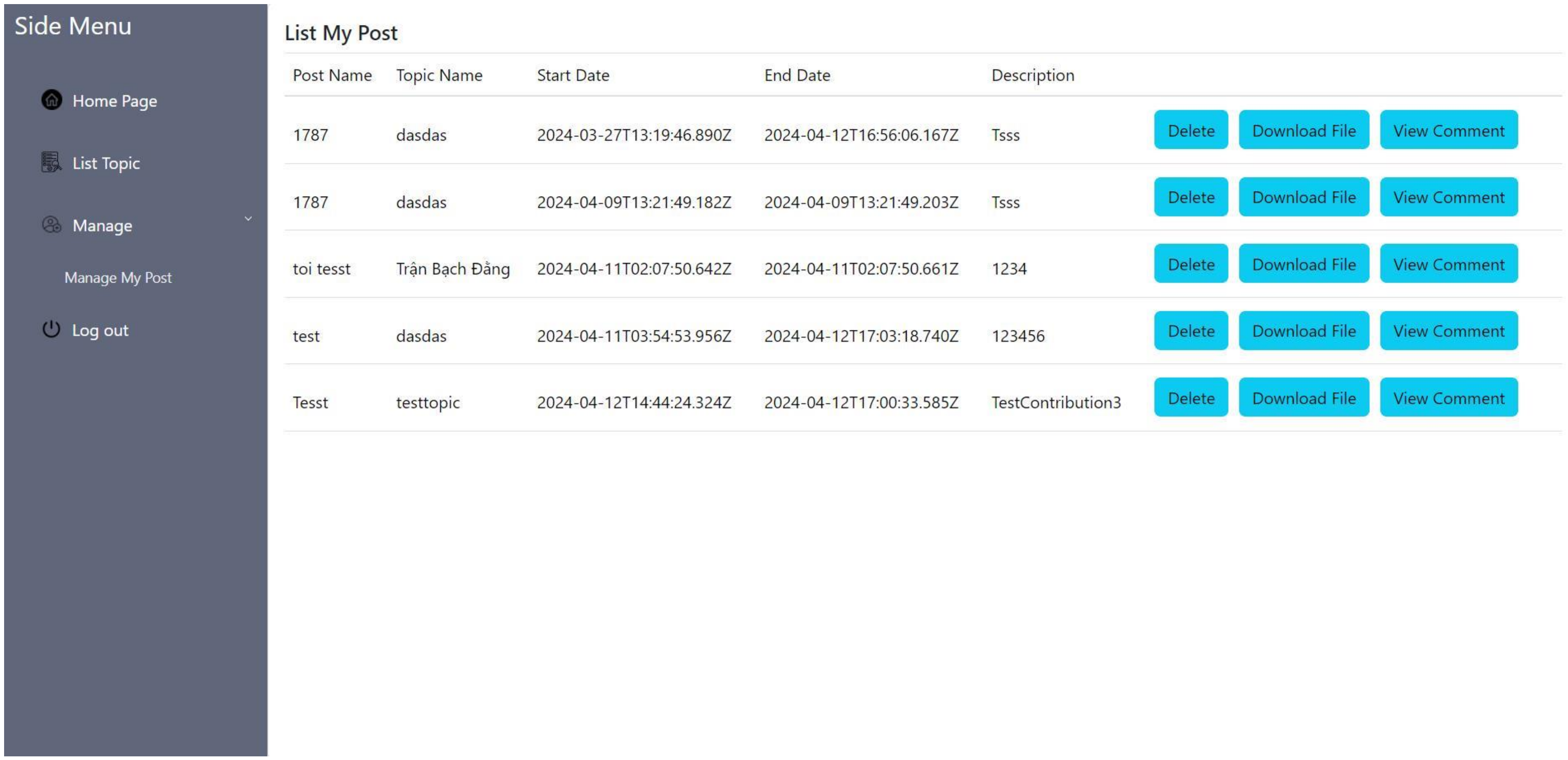
Figure 27: Submit topic

Students can contribute to the topic created by marketing



I commit to comply with and fully accept the terms and conditions of the service, by reading and understanding their contents before continuing to use.

Figure 28: Terms & Conditions

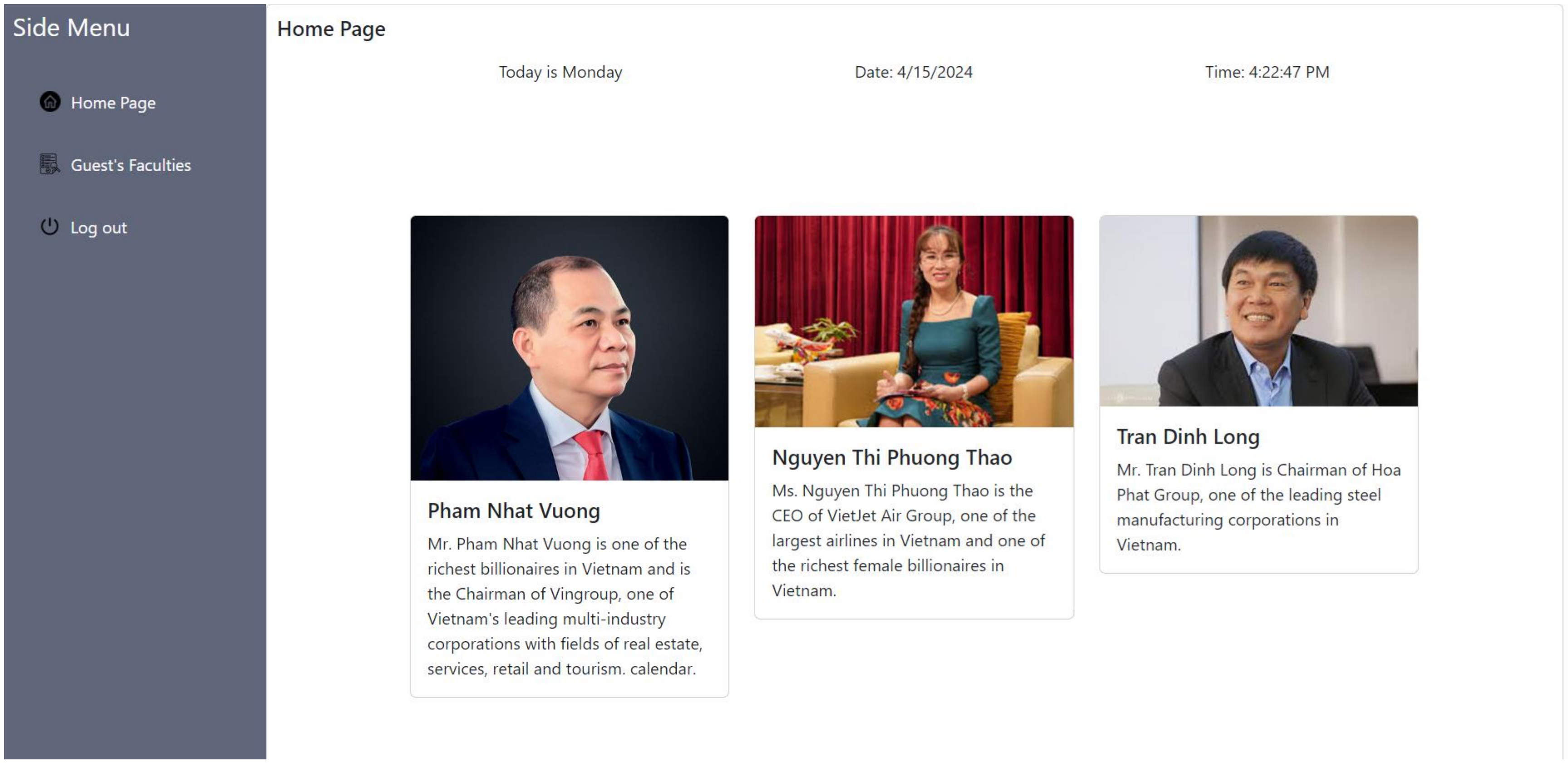


List My Post

Post Name	Topic Name	Start Date	End Date	Description	
1787	dasdas	2024-03-27T13:19:46.890Z	2024-04-12T16:56:06.167Z	Tsss	Delete Download File View Comment
1787	dasdas	2024-04-09T13:21:49.182Z	2024-04-09T13:21:49.203Z	Tsss	Delete Download File View Comment
toi tesst	Trận Bạch Đằng	2024-04-11T02:07:50.642Z	2024-04-11T02:07:50.661Z	1234	Delete Download File View Comment
test	dasdas	2024-04-11T03:54:53.956Z	2024-04-12T17:03:18.740Z	123456	Delete Download File View Comment
Tesst	testtopic	2024-04-12T14:44:24.324Z	2024-04-12T17:00:33.585Z	TestContribution3	Delete Download File View Comment

Figure 29: List Student Post

Students can see the comments that the coordinator comments on the contribution



Today is Monday Date: 4/15/2024 Time: 4:22:47 PM

Pham Nhat Vuong

Mr. Pham Nhat Vuong is one of the richest billionaires in Vietnam and is the Chairman of Vingroup, one of Vietnam's leading multi-industry corporations with fields of real estate, services, retail and tourism. calendar.

Nguyen Thi Phuong Thao

Ms. Nguyen Thi Phuong Thao is the CEO of VietJet Air Group, one of the largest airlines in Vietnam and one of the richest female billionaires in Vietnam.

Tran Dinh Long

Mr. Tran Dinh Long is Chairman of Hoa Phat Group, one of the leading steel manufacturing corporations in Vietnam.

Figure 30: Home page Guest

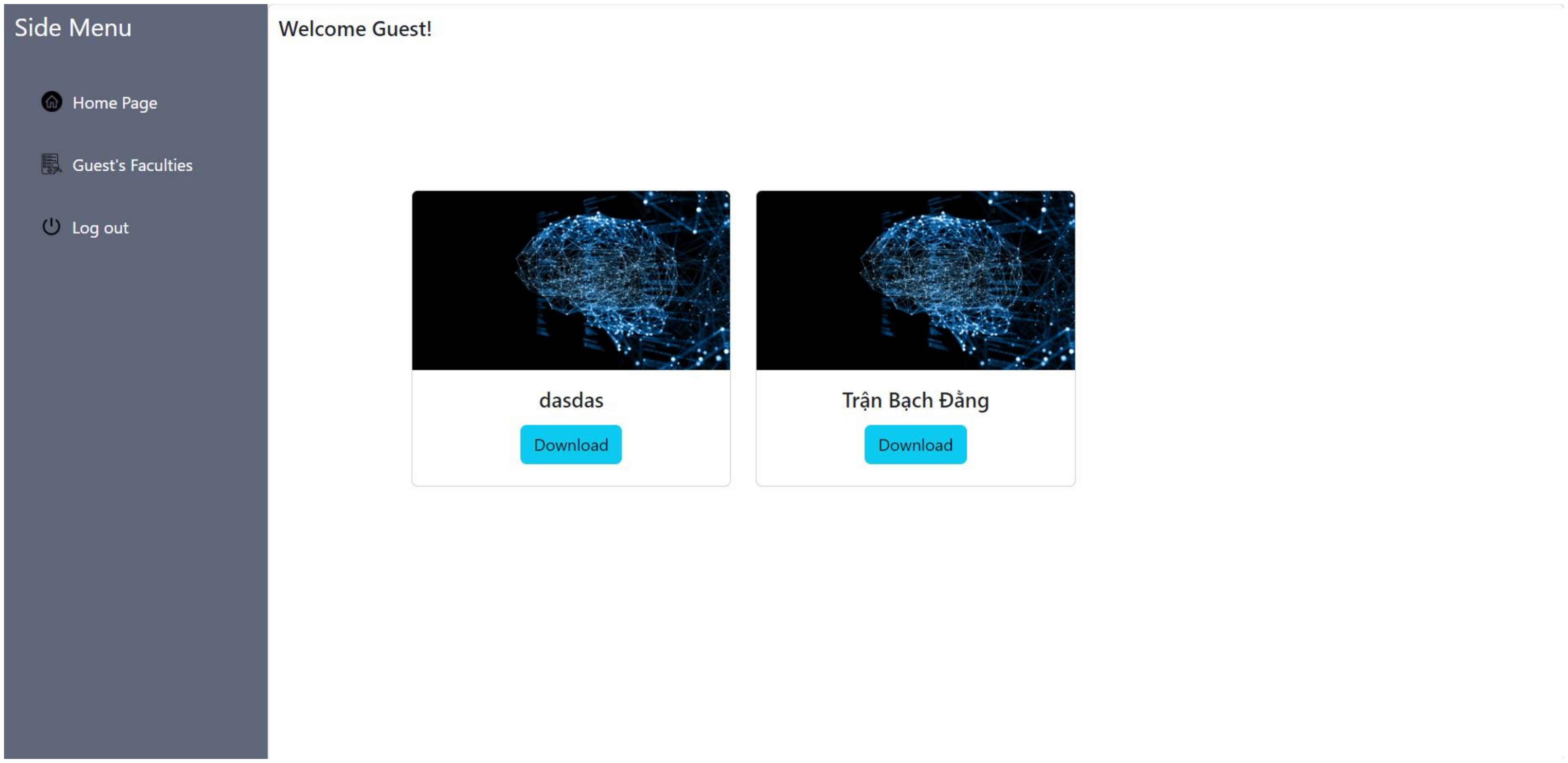


Figure 31: view contribution guest