# Bethe OPS Project

## Developer Instructions: Users

**May 16, 2019**

# Table of Contents

## 1 Overview

This document describes the technical part of the functionalities of the three different types of users (students, event leaders, and admins) on the website and explains how to modify such functionalities. For details on the user flow of these functionalities, please refer to "User Instructions".

The styles for all the functionalities are encapsulated with the file ./**public/css/style.css ./public/css/materialize.css,** and **./public/css/materialize.min.css**. The functions that load all the elements are all contained in **./routes/students.js, ./routes/eventleader.js, ./routes/admin.js, ./public/js/sidebar.js**, **./public/js/init.js, ./public/js/materialize.js**, and **./public/js/materialize.min.js**. Details on any of the functionality from Materialize (both the CSS files and the JS files) can be found on Materialize CSS's website.

## 2 Required Technologies

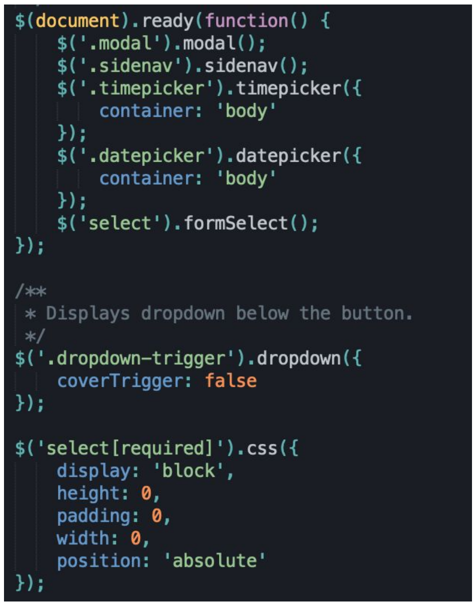
The required technologies for operating all users are:

- EJS
- JavaScript
- jQuery
- CSS
    - Materialize CSS framework

## 3 How It Works

For each user, there are three components: the navigation bar, the event profile, and an identical sidebar. This section will discuss the technical parts of the navigation bar and the event profile for the navigation bar and the event profile, focusing on the differences in the implementation of each user. Details on the functionality of the sidebar are in a separate document called "Sidebar".

In all user types, the front-end is implemented using EJS, which is a templating language that generates HTML markup with plain JavaScript, and CSS. All the front-end for the student landing page is found in **./views/pages/index.ejs**, the event leader landing page is found in **./views/pages/eventleader.ejs**, and the admin landing page in **./views/pages/admin.ejs**. The back-end encompasses the functionality of the landing page and handles dynamic changes made on the page; the back-end is implemented using JavaScript and jQuery. The back-end that initializes the modal, along with the other elements of the page, including the featured event, dropdowns, etc. are all found in **init.js**.

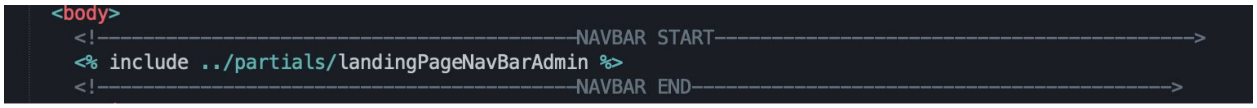


```
$(document).ready(function() {
    $('.modal').modal();
    $('.sidenav').sidenav();
    $('.timepicker').timepicker({
        container: 'body'
    });
    $('.datepicker').datepicker({
        container: 'body'
    });
    $('select').formSelect();
});

/**
 * Displays dropdown below the button.
 */
$('.dropdown-trigger').dropdown({
    coverTrigger: false
});

$('select[required]').css({
    display: 'block',
    height: 0,
    padding: 0,
    width: 0,
    position: 'absolute'
});
```

*Initializes the elements of the page*

## 3.1 Navigation Bar

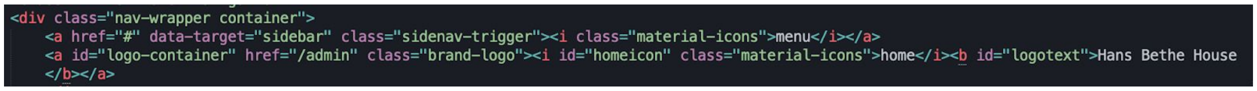
The navigation bar is a **partial** in each user EJS file. In the EJS files for the landing pages, there is a section near the top of the file that calls the navigation bar partial as so:



```
<body>
  <!------------------------------------------NAVBAR START------------------------------------------>
  <% include ../partials/landingPageNavBarAdmin %>
  <!------------------------------------------NAVBAR END------------------------------------------>
```

*Attaching the navigation bar to the landing page front-end*

The elements that all three users share on the navbar are the "Hans Bethe House" logo on the top left corner of the screen, and the side navigation bar trigger for the sidebar when the screen sizes are smaller. Clicking on the "Hans Bethe House" logo will redirect the user back to the default landing page of the current user. The trigger for the sidebar will only show up if screen sizes are smaller than 600px, and clicking on this trigger will open the sidebar from the left. The backend for this functionality is contained within the file **materialize.js**.

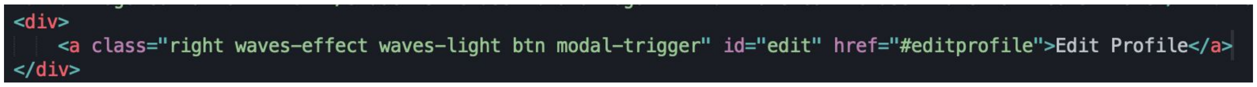


```
<div class="nav-wrapper container">
    <a href="#" data-target="sidebar" class="sidenav-trigger"><i class="material-icons">menu</i></a>
    <a id="logo-container" href="/admin" class="brand-logo"><i id="homeicon" class="material-icons">home</i><b id="logotext">Hans Bethe House
    </b></a>
```

*Displaying the sidebar trigger and page logo*

### 3.1.1 Student

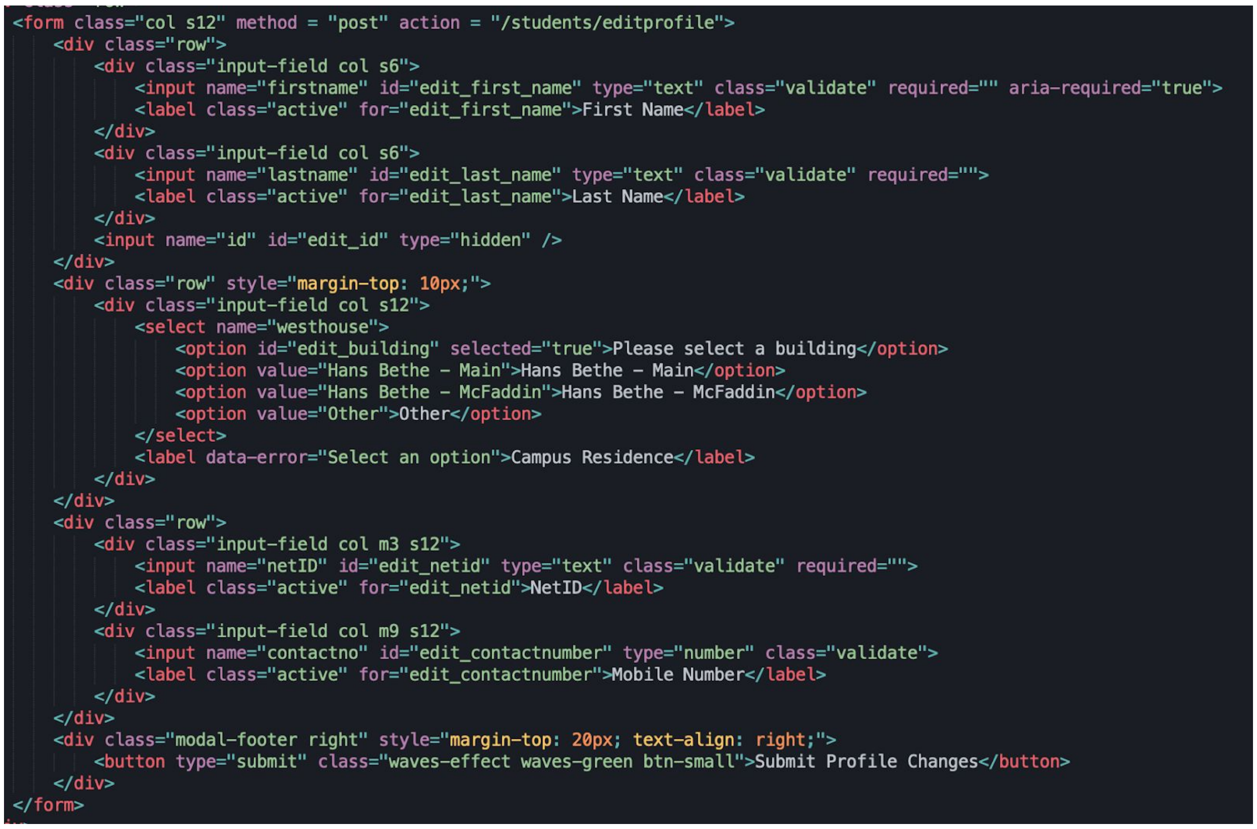
On the right corner of the navigation bar for the **student** landing page, there is a button labeled "Edit Profile". Clicking this button will open a modal, which dynamically loads the logged in student's information.



```
<div>
    <a class="right waves-effect waves-light btn modal-trigger" id="edit" href="#editprofile">Edit Profile</a>
</div>
```

*Displaying the "Edit Profile" button*

The modal that is opened contains a form that allows the user to edit the contents of each input. The inputs, such as first name, last name, etc. are separated into rows. At the bottom of the modal, there is a button labeled "Submit Profile Changes" that the student clicks to update the profile.

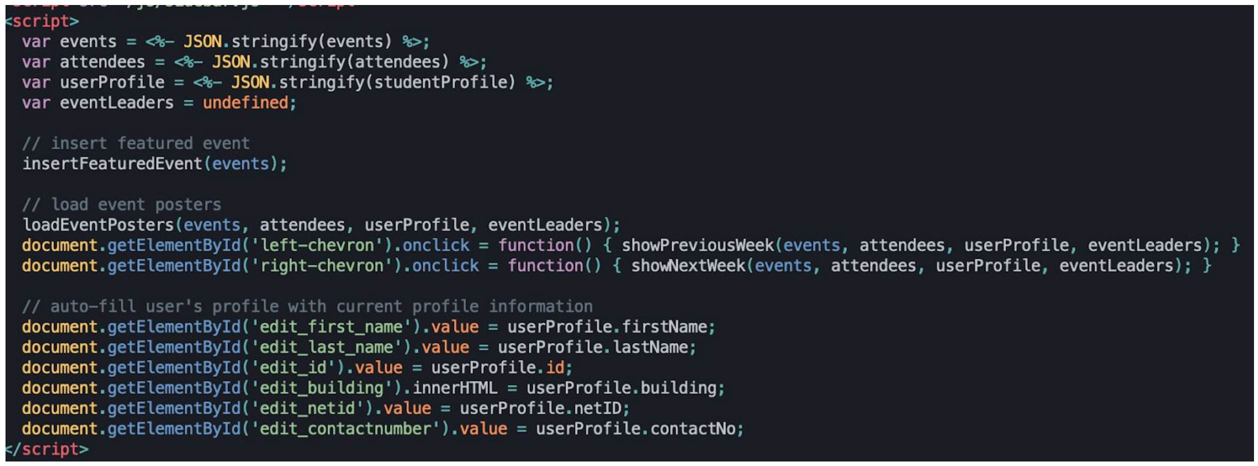


```
<form class="col s12" method = "post" action = "/students/editprofile">
    <div class="row">
        <div class="input-field col s6">
            <input name="firstname" id="edit_first_name" type="text" class="validate" required="" aria-required="true">
            <label class="active" for="edit_first_name">First Name</label>
        </div>
        <div class="input-field col s6">
            <input name="lastname" id="edit_last_name" type="text" class="validate" required="">
            <label class="active" for="edit_last_name">Last Name</label>
        </div>
        <input name="id" id="edit_id" type="hidden" />
    </div>
    <div class="row" style="margin-top: 10px;">
        <div class="input-field col s12">
            <select name="westhouse">
                <option id="edit_building" selected="true">Please select a building</option>
                <option value="Hans Bethe - Main">Hans Bethe - Main</option>
                <option value="Hans Bethe - McFaddin">Hans Bethe - McFaddin</option>
                <option value="Other">Other</option>
            </select>
            <label data-error="Select an option">Campus Residence</label>
        </div>
    </div>
    <div class="row">
        <div class="input-field col m3 s12">
            <input name="netID" id="edit_netid" type="text" class="validate" required="">
            <label class="active" for="edit_netid">NetID</label>
        </div>
        <div class="input-field col m9 s12">
            <input name="contactno" id="edit_contactnumber" type="number" class="validate">
            <label class="active" for="edit_contactnumber">Mobile Number</label>
        </div>
    </div>
    <div class="modal-footer right" style="margin-top: 20px; text-align: right;">
        <button type="submit" class="waves-effect waves-green btn-small">Submit Profile Changes</button>
    </div>
</form>
```

*Displaying the "Edit Profile" Modal*

All fields are normal text inputs except the student building of residence input. This is shown as a dropdown with the three options for buildings: "Bethe - Main", "Bethe -

McFaddin", and "Other". The fields are loaded with the student's information at the bottom of **index.ejs**.



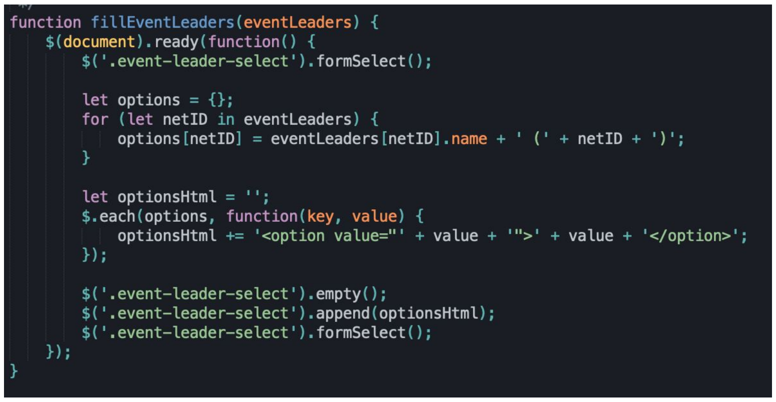
```
<script>
  var events = <%- JSON.stringify(events) %>;
  var attendees = <%- JSON.stringify(attendees) %>;
  var userProfile = <%- JSON.stringify(studentProfile) %>;
  var eventLeaders = undefined;

  // insert featured event
  insertFeaturedEvent(events);

  // load event posters
  loadEventPosters(events, attendees, userProfile, eventLeaders);
  document.getElementById('left-chevron').onclick = function() { showPreviousWeek(events, attendees, userProfile, eventLeaders); }
  document.getElementById('right-chevron').onclick = function() { showNextWeek(events, attendees, userProfile, eventLeaders); }

  // auto-fill user's profile with current profile information
  document.getElementById('edit_first_name').value = userProfile.firstName;
  document.getElementById('edit_last_name').value = userProfile.lastName;
  document.getElementById('edit_id').value = userProfile.id;
  document.getElementById('edit_building').innerHTML = userProfile.building;
  document.getElementById('edit_netid').value = userProfile.netID;
  document.getElementById('edit_contactnumber').value = userProfile.contactNo;
</script>
```

*Dynamically loading the editable fields in the "Edit Profile" modal*

The user's current information is auto-filled based on the information of the user profile, which is grabbed from the database. The server uses the person id stored in cookies to query for the user record in the people table and this record is sent back to the newly rendered page. Then, the inputs initial values are set to the information sent from the backend, which the student can override by typing inside of the inputs. When the student clicks the "Submit Profile Changes", this sends a POST request to the route, /students/editprofile (as indicated in the <form> tag depicted in "Displaying the 'Edit Profile' Modal" figure above); the code for processing the request is located in **./routes/student.js**. The code updates the current user record in the people table, if any changes has been made.

### 3.1.2 Event Leader

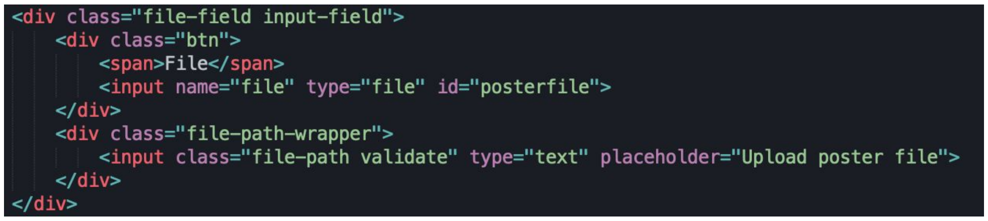
The only difference between the navigation bar for a student and the navigation bar for event leaders and admins, is that the "Edit Profile" button is replaced by a dropdown with multiple options. The EJS code for the navigation bar of the event leader is found in **./views/partials/landingPageNavBarEL.ejs**. For an event leader, this includes "Edit Profile", "Create Event", "Add Student", and "Remove Student". Each of these dropdown options will open up a modal in which the event leader enters values in the fields given in order to fulfill the action. All the EJS for the modals can be found in the same file, under the "Edit Profile" modal, and the code is separated by comments. Similar to the "Edit Profile" modal, clicking the "Submit" button at the bottom of the modal will send a POST request to the route, /eventleader/editprofile; the code to process this request is located in **./routes/eventleader.js**. The code updates the current user record in the people table, if any changes has been made. The “Create Event” modal contains a dropdown for event leaders, which will display all the event leaders currently in the system. The names and net IDs of the event leaders are dynamically loaded in the **./public/js/init.js** file, under the function, fillEventLeaders( ).



```
function fillEventLeaders(eventLeaders) {
    $(document).ready(function() {
        $('.event-leader-select').formSelect();

        let options = {};
        for (let netID in eventLeaders) {
            options[netID] = eventLeaders[netID].name + ' (' + netID + ')';
        }

        let optionsHtml = '';
        $.each(options, function(key, value) {
            optionsHtml += '<option value="' + value + '">' + value + '</option>';
        });

        $('.event-leader-select').empty();
        $('.event-leader-select').append(optionsHtml);
        $('.event-leader-select').formSelect();
    });
}
```

*Dynamically loading the dropdown options for event leaders in the “Create Event” modal*

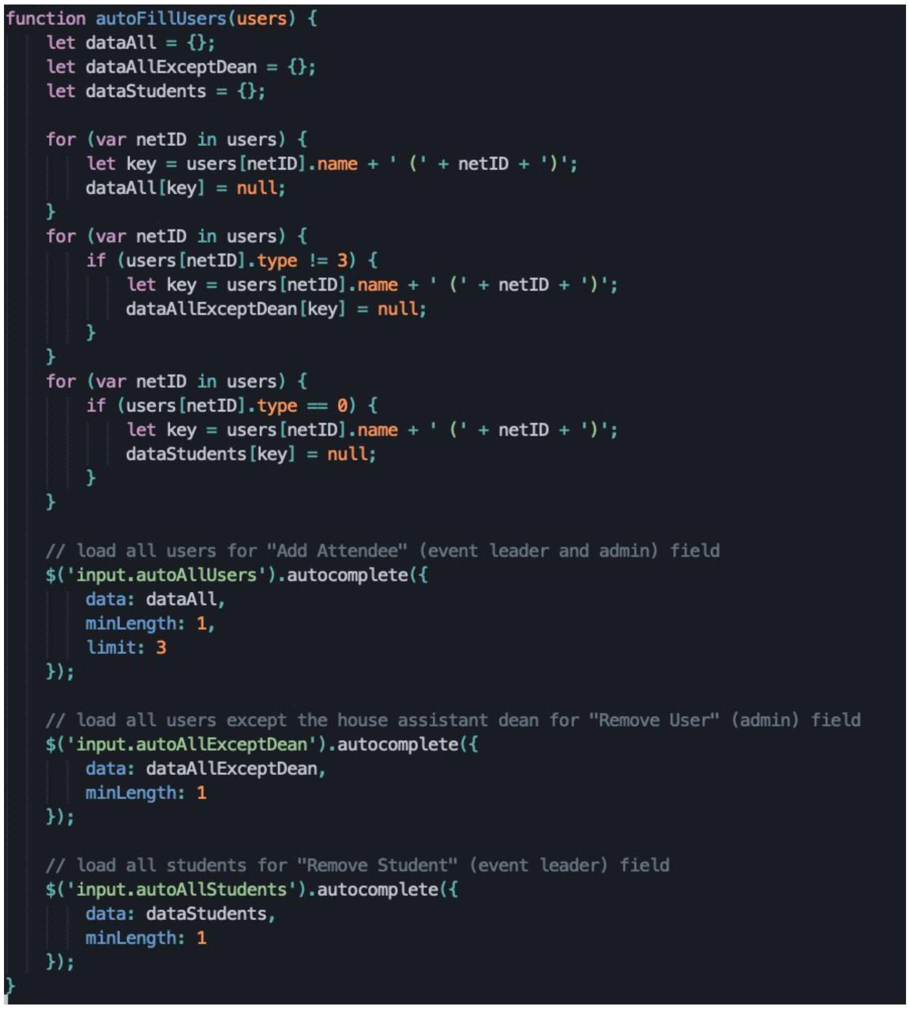
The “Create Event” modal also has an input field that allows the event leader to upload an image as the event poster, which is displayed in both the sidebar and the event profile. The image is uploaded, using the multer library. Then, the multer library is used to create a disk storage object, which indicates where all images should be stored and what all stored images should be called. Afterwards, an upload object is created with this disk storage object. On the routes that need to store an image, the upload.single() object is passed in as an object. The image is stored in **./public/images**; the functionality for file uploads is located in **./exports/storage.js**.



```
<div class="file-field input-field">
    <div class="btn">
        <span>File</span>
        <input name="file" type="file" id="posterfile">
    </div>
    <div class="file-path-wrapper">
        <input class="file-path validate" type="text" placeholder="Upload poster file">
    </div>
</div>
```

*Input field for uploading an event poster*

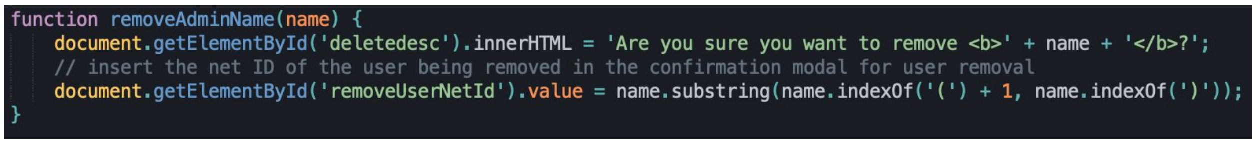
Clicking the “Add Student” option opens a modal with a form identical to the form that is displayed when a student first logs in and is redirected to a sign-up screen. Clicking the “Remove Student” option opens a modal with an autocomplete text input. Typing in letters will display a dropdown with student names and net IDs. The maximum number of dropdown options shown is three so not to clutter the modal. The list of student names and net IDs are dynamically loaded in **init.js** in the function autoFillUsers( ), under the autocomplete for *autoAllStudents*. Note the limit of the autofill list displayed can be changed by changing the value of the field, *limit*.



```
function autoFillUsers(users) {
    let dataAll = {};
    let dataAllExceptDean = {};
    let dataStudents = {};

    for (var netID in users) {
        let key = users[netID].name + ' (' + netID + ')';
        dataAll[key] = null;
    }
    for (var netID in users) {
        if (users[netID].type != 3) {
            let key = users[netID].name + ' (' + netID + ')';
            dataAllExceptDean[key] = null;
        }
    }
    for (var netID in users) {
        if (users[netID].type == 0) {
            let key = users[netID].name + ' (' + netID + ')';
            dataStudents[key] = null;
        }
    }

    // load all users for "Add Attendee" (event leader and admin) field
    $('input.autoAllUsers').autocomplete({
        data: dataAll,
        minLength: 1,
        limit: 3
    });

    // load all users except the house assistant dean for "Remove User" (admin) field
    $('input.autoAllExceptDean').autocomplete({
        data: dataAllExceptDean,
        minLength: 1
    });

    // load all students for "Remove Student" (event leader) field
    $('input.autoAllStudents').autocomplete({
        data: dataStudents,
        minLength: 1
    });
}
```

*Function that dynamically loads the list of students and their net IDs for autocomplete inputs*

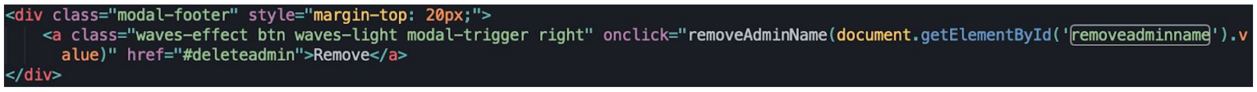
Removing a student is permanent so clicking the “Remove” button at the bottom of the “Remove Student” modal will first open another modal that warns the event leader that the action will be permanent. The name that is displayed in this modal is dynamically loaded in the function removeAdminName( ).



```
function removeAdminName(name) {
    document.getElementById('deletedesc').innerHTML = 'Are you sure you want to remove <b>' + name + '</b>?';
    // insert the net ID of the user being removed in the confirmation modal for user removal
    document.getElementById('removeUserNetId').value = name.substring(name.indexOf('(') + 1, name.indexOf(')'));
}
```

*Dynamically loads the student name and netid that is being removed from the system*

The required parameter *name* is passed in when the “Remove” button is clicked, and the name of the student is grabbed from the value of the input field of the “Remove Student” modal (which has the id “removeadminname”).

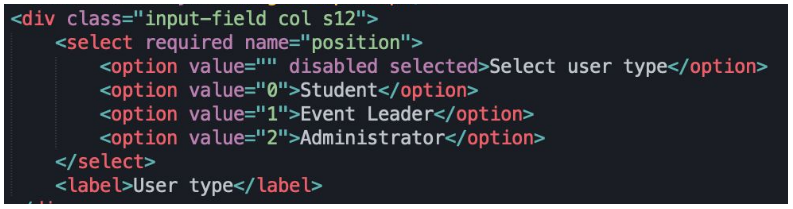


```
<div class="modal-footer" style="margin-top: 20px;">
    <a class="waves-effect btn waves-light modal-trigger right" onclick="removeAdminName(document.getElementById('removeadminname').v
      alue)" href="#deleteadmin">Remove</a>
</div>
```

*Passing in the name of the removed student into removeAdminName*

### 3.1.3 Admin

The EJS code can be found in **./views/partials/landingPageNavBarAdmin.ejs**. The navigation bar for admins is nearly identical to the navigation bar for event leaders. The only difference is that the dropdown menu will have “Add User” and “Remove User” options instead of “Add Student” and “Remove Student”. The “Add User” modal will be similar to the “Add Student” modal for event leaders except there will be an added dropdown field in which the admin chooses what type of user the added user should be. The three options are “Student”, “Event Leader”, and “Administrator”.



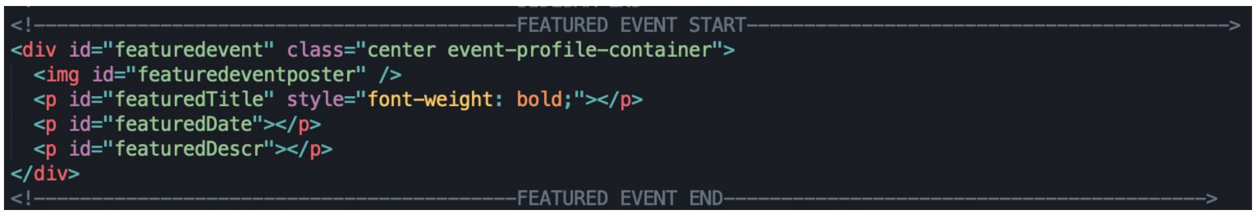
```
<div class="input-field col s12">
    <select required name="position">
        <option value="" disabled selected>Select user type</option>
        <option value="0">Student</option>
        <option value="1">Event Leader</option>
        <option value="2">Administrator</option>
    </select>
    <label>User type</label>
```

*Dropdown for User Type*

The functionality for the “Remove User” modal is the same as the “Remove Student” modal for event leaders. The admin types in a name or net ID, selects a name from the autocomplete list, clicks the “Remove” button, and confirms the action. The list that shows up includes all types of users, including event leaders and admins, with the exception of the assistant dean, Erica Ostermann. The list of users is dynamically loaded in autoFillUsers( ), under the autocomplete for *autoAllUsers*.

## 3.2 Event Profile

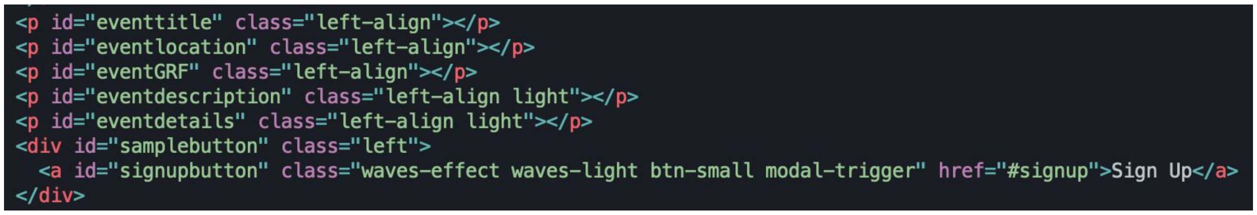
The EJS for the event profile displayed is all contained in the respective EJS files (**index.ejs, eventleader.ejs, admin.ejs**). The default event profile when first visiting the landing page for all users is the featured event. The featured event is the next upcoming event within the next two weeks starting on the current day (i.e., today). Details on how the featured event details are dynamically loaded are in the document “Sidebar”.



```
<!-----------------------------------------FEATURED EVENT START------------------------------------------->
<div id="featuredevent" class="center event-profile-container">
  <img id="featuredeventposter" />
  <p id="featuredTitle" style="font-weight: bold;"></p>
  <p id="featuredDate"></p>
  <p id="featuredDescr"></p>
</div>
<!-----------------------------------------FEATURED EVENT END-------------------------------------------->
```

*Displaying the featured event*

The event profile for all users after clicking on an event in the sidebar contains the event poster, title, location, GRF/event leader, description, additional details, and a button to sign up for the event. All this information is dynamically loaded in **./public/js/sidebar.js**. This process is further elaborated in the document “Sidebar”.

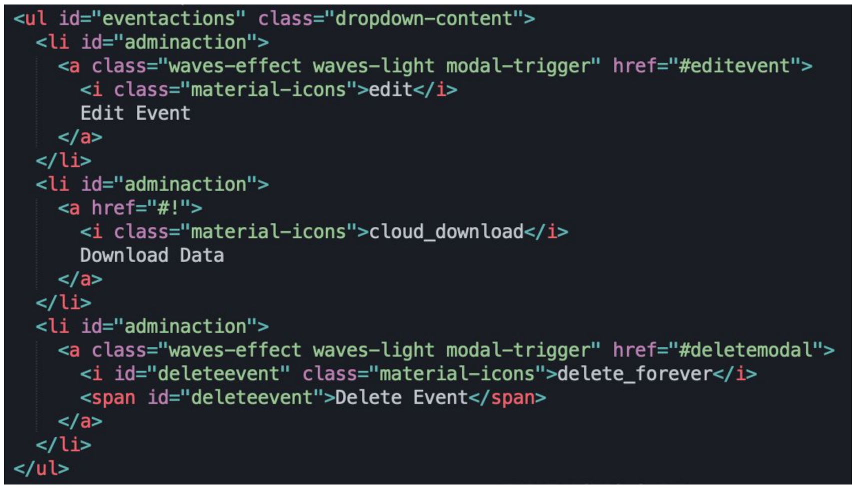


```
<p id="eventtitle" class="left-align"></p>
<p id="eventlocation" class="left-align"></p>
<p id="eventGRF" class="left-align"></p>
<p id="eventdescription" class="left-align light"></p>
<p id="eventdetails" class="left-align light"></p>
<div id="samplebutton" class="left">
  <a id="signupbutton" class="waves-effect waves-light btn-small modal-trigger" href="#signup">Sign Up</a>
</div>
```

*Displaying the event information*

Clicking the “Sign Up” button opens a modal with only one optional field “Other Comments”. These comments are stored in the database for admins to see but will not be displayed in the interface anywhere. When clicking the “Sign Up” button at the bottom of this modal, a POST request is sent to add the user signing up to the attendee list of the event. In the backend, the server creates a new record in signups table with the user id, the event id(passed to server as a hidden input in the submitted form), the current time down to milliseconds, and any comments the user may have.

For event leaders and admins, there will also be a dropdown in the upper-right corner of the event profile called “Event Actions”. These actions include “Edit Event”, “Download Data”, and “Delete Event” (NOTE: “Download Data” is currently NOT implemented). Clicking on “Edit Event” or “Delete Event” will open a modal for the action.



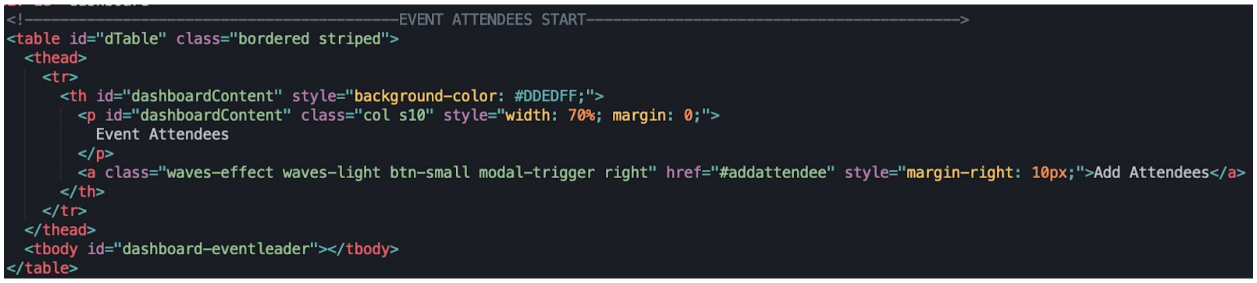
```html
<ul id="eventactions" class="dropdown-content">
  <li id="adminaction">
    <a class="waves-effect waves-light modal-trigger" href="#editevent">
      <i class="material-icons">edit</i>
      Edit Event
    </a>
  </li>
  <li id="adminaction">
    <a href="#!">
      <i class="material-icons">cloud_download</i>
      Download Data
    </a>
  </li>
  <li id="adminaction">
    <a class="waves-effect waves-light modal-trigger" href="#deletemodal">
      <i id="deleteevent" class="material-icons">delete_forever</i>
      <span id="deleteevent">Delete Event</span>
    </a>
  </li>
</ul>
```

*Displaying the "Event Actions" dropdown*

The EJS for the modals for "Edit Event" and "Delete Event" can be found under the "Event Actions" dropdown EJS.

The fields of the "Edit Event" modal are dynamically loaded in the file **sidebar.js**. The details of this are elaborated in the document "Sidebar". When the user clicks the button "Save Event" at the bottom of the modal, this sends a POST request to the respective JS file in **./routes** (**eventleader.js** for event leaders and **admin.js** for admins). In the backend, the server creates an update query to update the current event record in the events table, if necessary.

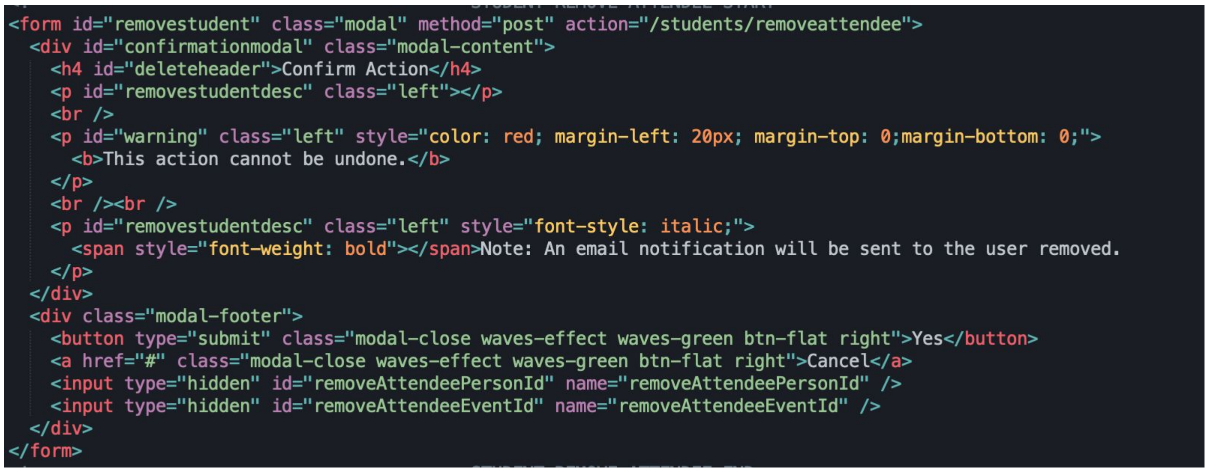
Below the event details, there is a table that shows all the attendees that are currently signed up for the clicked event. This table will not show up on the student landing page if upon creating an event, the event leader or admin chose the option to hide the attendees list. The attendees list will always show up for the event leader landing page and the admin landing page. The attendee list is dynamically loaded and details on this are also elaborated in the document "Sidebar".



```html
<!-------------------------------------------EVENT ATTENDEES START------------------------------------------->
<table id="dTable" class="bordered striped">
  <thead>
    <tr>
      <th id="dashboardContent" style="background-color: #DDEDFF;">
        <p id="dashboardContent" class="col s10" style="width: 70%; margin: 0;">
          Event Attendees
        </p>
        <a class="waves-effect waves-light btn-small modal-trigger right" href="#addattendee" style="margin-right: 10px;">Add Attendees</a>
      </th>
    </tr>
  </thead>
  <tbody id="dashboard-eventleader"></tbody>
</table>
```

*Displaying event attendees*

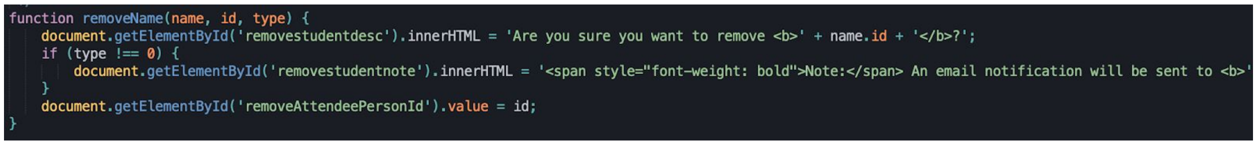
In the event attendees table, the user currently logged in is able to remove him/herself from the event. Next to the user's name is a red "X" icon. Students can remove themselves from an event whereas event leaders and admins are able to remove any attendee from an event. Clicking this icon opens a "Confirm Action" modal, similar to the modal that is opened when an event leader or admin wants to remove a user from the system.



```
<form id="removestudent" class="modal" method="post" action="/students/removeattendee">
  <div id="confirmationmodal" class="modal-content">
    <h4 id="deleteheader">Confirm Action</h4>
    <p id="removestudentdesc" class="left"></p>
    <br />
    <p id="warning" class="left" style="color: red; margin-left: 20px; margin-top: 0;margin-bottom: 0;">
      <b>This action cannot be undone.</b>
    </p>
    <br /><br />
    <p id="removestudentdesc" class="left" style="font-style: italic;">
      <span style="font-weight: bold"></span>Note: An email notification will be sent to the user removed.
    </p>
  </div>
  <div class="modal-footer">
    <button type="submit" class="modal-close waves-effect waves-green btn-flat right">Yes</button>
    <a href="#" class="modal-close waves-effect waves-green btn-flat right">Cancel</a>
    <input type="hidden" id="removeAttendeePersonId" name="removeAttendeePersonId" />
    <input type="hidden" id="removeAttendeeEventId" name="removeAttendeeEventId" />
  </div>
</form>
```

*Display the confirmation modal when trying to remove an attendee*

The name that is displayed in this modal is dynamically loaded in **init.js** in the function removeName( ).



```
function removeName(name, id, type) {
    document.getElementById('removestudentdesc').innerHTML = 'Are you sure you want to remove <b>' + name.id + '</b>?';
    if (type !== 0) {
        document.getElementById('removestudentnote').innerHTML = '<span style="font-weight: bold">Note:</span> An email notification will be sent to <b>'
    }
    document.getElementById('removeAttendeePersonId').value = id;
}
```

*Dynamically loading the name of the attendee being removed from an event*

The three required parameters are the name of the attendee being removed, the personal ID of the attendee, and the user type of the attendee.

The event leader and admin landing pages also have an "Add Attendees" button at the upper right corner of the attendees table. Clicking this will open another modal. The modal is similar to the modals that are opened when removing a user from the system, and the EJS code can be found in the respective EJS files for the landing pages, labeled with comments. There is an autocomplete input in which the event leader or admin can type in the name or net ID of the user (s)he wants to add to the event. The list of users that can be added are dynamically loaded in the function autoFillUsers( ) in **init.js**.