Final Project – Full-Stack CRUD Application

DUE DATE

- **Due on Thursday, 5/15, at 11:59 PM**
- **You can work in groups or individually. Maximum 4 students per group. Working in groups is strongly recommended and encouraged.**

HOW TO SUBMIT

- **Brightspace:** Submit (1) project document file in PDF format, (2) the link to your GitHub repository for the client-side (front-end) application, (3) the link to your GitHub repository for the server-side (back-end) application, and (4) your group member names on Brightspace by 11:59 PM on the due day. Each student must submit the assignment individually on Brightspace—even when working in a group.
- **GitHub:**
    - In the README section of GitHub repository for the client-side (front-end) application, please list your group member names and GitHub usernames.
    - In the README section of GitHub repository for the server-side (back-end) application, please list your group member names and GitHub usernames.

GRADING

This assignment is worth **50%** of your grade.

- **5%** - Project document in PDF format (i.e., feature requirements, application architecture description and diagram, epics, user stories, acceptance criteria, and project schedule chart).
- **40%** - Assignment functionality (i.e., actual functioning website and content).
- **5%** - Code organization (i.e., the code is clean, well-formatted, commented, and easy-to-read). Git version control such as following Git feature branch workflow, creating pull requests when merging feature branches, making small and frequent commits with appropriate commit messages, etc.

GOAL

The goal is to offer students the opportunity to gain hands-on experience in the development of full-stack web application by completing the following tasks:

- Building a RESTful full-stack web application to manage students and campuses using Node, Express, React, Redux, PostgreSQL, and Sequelize (an ORM)
- Implementing the CRUD operations of database: Create, Read, Update, and Delete
- Writing models, querying a database with Object-Relational Mapping (ORM), designing routes/endpoints and handler functions to process user requests and generate responses
- Developing React components, managing the application state with React Redux, and much more

- Creating two individual repositories for client-side (front-end) and server-side (back-end) applications (i.e., a separate client and a separate server), for separation of concerns and modularity

ASSIGNMENT

In this assignment, you will develop a Campus Management System using the full-stack web technologies of PostgreSQL, Express, React, and Node.js (i.e., the PERN technology stack). The system consists of both client-side (front-end) and server-side (back-end) applications.

Complete the Following User Stories

As a user, I:

Home Page View

1. will land on a visually pleasing home page by default, which allows navigation to view **all campuses** and **all students**

All Campuses View

2. can navigate to the **All Campuses View**, and
    - see a list of all campuses in the database
    - see an informative message if no campuses exist
    - add a new campus
    - delete a campus (e.g., via link/button and optionally, this can be part of the **Single Campus View**)

Single Campus View

3. can navigate to the **Single Campus View**, and
    - see details about a single campus, with all data fields, including enrolled students (if any)
    - see an informative message if no students are enrolled at that campus
    - navigate to the **Single Student View** and see any student's information
    - add new/existing students to the campus (e.g., via link/button)
    - delete students from the campus (e.g., via link/button)
    - navigate to the **Edit Campus View** and edit the campus information
    - delete the campus (e.g., via link/button and optionally, this can be part of the **All Campuses View**)

Add Campus View

4. can navigate to the **Add Campus View**, and
    - enter the campus information using a form
        - including data fields: name, address, description, and image URL.
        - with a validated form displaying real-time error messages (e.g., for an invalid input)

Edit Campus View

5. can navigate to the **Edit Campus View**, and
    - edit the campus information using a form
        - including data fields: name, address, description, and image URL.

        - with a validated form displaying real-time error messages (e.g., for an invalid input)

All Students View

6. can navigate to the **All Students View**, and
    - see a list of all students in the database
    - see an informative message if no students exist
    - add a new student
    - delete a student (e.g., via link/button and optionally, this can be part of the **Single Student View**)

Single Student View

7. can navigate to the **Single Student View**, and
    - see details about a single student, with all data fields, including the campus at which the student is enrolled (if exists)
    - see an informative message if the student is not enrolled at a campus
    - navigate to the **Single Campus View** of the student's enrolled campus
    - navigate to the **Edit Student View** and edit the student's information
    - delete the student (e.g., via link/button and optionally, this can be part of the **All Students View**)

Add Student View

8. can navigate to the **Add Student View**, and
    - enter the student information using a form
        - including data fields: first name, last name, email, image URL, and GPA. (You may also include campus ID/name depending on the implementation.)
        - with a validated form displaying real-time error messages (e.g., for an invalid input)

Edit Student View

9. can navigate to the **Edit Student View**, and
    - edit the student information using a form
        - including data fields: first name, last name, email, image URL, GPA, and campus ID/name.
        - with a validated form displaying real-time error messages (e.g., for an invalid input)

Requirements and Functionalities

All Campuses and All Students

UI (React) – front-end application

1. Write a component to display a list of all campuses in the **All Campuses View**, with at least the following information of each campus: campus name and image (can be a default image)
2. Write a component to display a list of all students in the **All Students View**, with at least the following information of each student: student name

Client-Side Routing (React Router) – front-end application

1. Display the **All Campuses View** component when the URL matches "/campuses"
2. Display the **All Students View** component when the URL matches "/students"
3. Add links to the navigation bar that can be used to navigate to the **All Campuses View** and **All Students View**

State Management (Redux) – front-end application

1. Write a "campuses" sub-reducer to manage campuses in the Redux Store
2. Write a "students" sub-reducer to manage students in the Redux Store

Database (Sequelize) – back-end application

1. Write a "campus" model with the following information:
    - name - not allow null/empty
    - address - not allow null/empty
    - description - large text string, allow null/empty
    - imageUrl - with a default value, allow null/empty
2. Write a "student" model with the following information:
    - firstName - not allow null/empty
    - lastName - not allow null/empty
    - email - not allow null/empty
    - imageUrl - with a default value, allow null/empty
    - gpa - decimal between 0.0 and 4.0, allow null/empty
3. Student can be associated with at most one campus
4. Campus can be associated with many students

API/Server-Side Routing (Express, Sequelize) – back-end application

1. Write a route to serve up all students
2. Write a route to serve up all campuses

Single Campus and Single Student

UI (React) – front-end application

1. Write a component to display a single campus in the **Single Campus View** with the following information:
    - The campus's name, image, address, and description
    - A list of the names of all students in that campus (or a helpful message if it doesn't have any students)
2. Write a component to display a single student in the **Single Student View** with the following information:
    - The student's full name (first and last names), email, image, and GPA
    - The name of the student's campus (or a helpful message if the student doesn't have one)

Client-Side Routing (React Router) – front-end application

1. Display the specific campus's information when the URL matches "/campus/:campusId"

2. Display the specific student's information when the URL matches "/student/:studentId"
3. Clicking on the name of a campus from the **All Campuses View** should navigate to show that campus in the **Single Campus View**
4. Clicking on the name of a student from the **All Students View** should navigate to show that student in the **Single Student View**
5. Clicking on the name of a student in the **Single Campus View** should navigate to show that student in the **Single Student View**
6. Clicking on the name of a campus in the **Single Student View** should navigate to show that campus in the **Single Campus View**

API/Server-Side Routing (Express, Sequelize) – back-end application

1. Write a route to serve up a single campus (based on the campus id), including that campus's students
2. Write a route to serve up a single student (based on the student id), including that student's campus

Adding a Campus and Adding a Student

UI (React) – front-end application

1. Write a component to display a form in the **Add Campus View** for adding a new campus, which contains input fields for all campus information
2. Submitting the form with valid inputs should:
    - Make a request that causes the new campus to be persisted in the database
    - Add the new campus to the list of campuses without needing to refresh the web page
3. Write a component to display a form in the **Add Student View** for adding a new student, which contains input fields for all student information
4. Submitting the form with valid inputs should:
    - Make a request that causes the new student to be persisted in the database
    - Add the new student to the list of students without needing to refresh the web page

API/Server-Side Routing (Express, Sequelize) – back-end application

1. Write a route to add a new campus
2. Write a route to add a new student

Editing a Campus and Editing a Student

UI (React) – front-end application

1. Write a component to display a form in the **Edit Campus View** for editing a campus, which contains fields for all campus information
2. Submitting the form with valid inputs should:
    - Make a request that causes the campus to be updated in the database
    - Display the updated campus information without needing to refresh the web page
3. Write a component to display a form in the **Edit Student View** for editing a student, which contains fields for all student information
4. Submitting the form with valid inputs should:

    - Make a request that causes the student to be updated in the database
    - Display the updated student information without needing to refresh the web page

API/Server-Side Routing (Express, Sequelize) – back-end application

1. Write a route to edit a campus (based on the campus id)
2. Write a route to edit a student (based on the student id)

Deleting a Campus and Deleting a Student

UI (React) – front-end application

1. In the **All Campuses View**, include a "Delete" button next to each campus
2. Clicking the "Delete" button should:
    - Make a request that causes that campus to be deleted from the database
    - Delete the campus from the list of campuses without needing to refresh the web page
3. In the **All Students View**, include a "Delete" button next to each student
4. Clicking the "Delete" button should:
    - Make a request that causes that student to be deleted from the database
    - Delete the student from the list of students without needing to refresh the web page

API/Server-Side Routing (Express, Sequelize) – back-end application

1. Write a route to delete a campus (based on the campus id)
2. Write a route to delete a student (based on the student id)

Project Document

1. Business requirements, functional requirements, and non-functional requirements
2. Software application architecture description and diagram
3. Epics, user stories, and acceptance criteria
4. Project schedule chart (Gantt chart)

FEATURES IMPLEMENTED IN THE STARTER CODES

This section highlights the features that are either completed or partially completed in the starter codes for the front-end and back-end applications.

Green color identifies a feature that is completed. Yellow color identifies a feature that is partially completed.

Complete the Following User Stories

As a user, I:

Home Page View

1. will land on a visually pleasing home page by default, which allows navigation to view **all campuses** and **all students**

All Campuses View

2. can navigate to the **All Campuses View**, and
    - see a list of all campuses in the database
    - see an informative message if no campuses exist
    - add a new campus
    - delete a campus (e.g., via link/button and optionally, this can be part of the **Single Campus View**)

Single Campus View

3. can navigate to the **Single Campus View**, and
    - see details about a single campus, with all data fields, including enrolled students (if any)
    - see an informative message if no students are enrolled at that campus
    - navigate to the **Single Student View** and see any student's information
    - add new/existing students to the campus (e.g., via link/button)
    - delete students from the campus (e.g., via link/button)
    - navigate to the **Edit Campus View** and edit the campus information
    - delete the campus (e.g., via link/button and optionally, this can be part of the **All Campuses View**)

Add Campus View

4. can navigate to the **Add Campus View**, and
    - enter the campus information using a form
        - including data fields: name, address, description, and image URL.
        - with a validated form displaying real-time error messages (e.g., for an invalid input)

Edit Campus View

5. can navigate to the **Edit Campus View**, and
    - edit the campus information using a form
        - including data fields: name, address, description, and image URL.
        - with a validated form displaying real-time error messages (e.g., for an invalid input)

All Students View

6. can navigate to the **All Students View**, and
    - see a list of all students in the database
    - see an informative message if no students exist
    - add a new student
    - delete a student (e.g., via link/button and optionally, this can be part of the **Single Student View**)

Single Student View

7. can navigate to the **Single Student View**, and
    - see details about a single student, with all data fields, including the campus at which the student is enrolled (if exists)
    - see an informative message if the student is not enrolled at a campus
    - navigate to the **Single Campus View** of the student's enrolled campus
    - navigate to the **Edit Student View** and edit the student's information
    - delete the student (e.g., via link/button and optionally, this can be part of the **All Students View**)

Add Student View

8. can navigate to the **Add Student View**, and
    - enter the student information using a form
        - including data fields: first name, last name, email, image URL, and GPA. (May also include campus ID/name depending on the implementation.)
        - with a validated form displaying real-time error messages (e.g., for an invalid input)

Edit Student View

9. can navigate to the **Edit Student View**, and
    - edit the student information using a form
        - including data fields: first name, last name, email, image URL, GPA, and campus ID/name.
        - with a validated form displaying real-time error messages (e.g., for an invalid input)

Technical Requirements and Functionalities

All Campuses and All Students

UI (React) – front-end application

1. Write a component to display a list of all campuses in the **All Campuses View**, with at least the following information of each campus: campus name and image (can be a default image)
2. Write a component to display a list of all students in the **All Students View**, with at least the following information of each student: student name

Client-Side Routing (React Router) – front-end application

1. Display the **All Campuses View** component when the URL matches "/campuses"

2. Display the **All Students View** component when the URL matches "/students"
3. Add links to the navigation bar that can be used to navigate to the **All Campuses View** and **All Students View**

State Management (Redux) – front-end application

1. Write a "campuses" sub-reducer to manage campuses in the Redux Store
2. Write a "students" sub-reducer to manage students in the Redux Store

Database (Sequelize) – back-end application

1. Write a "campus" model with the following information:
   - name - not allow null/empty
   - address - not allow null/empty
   - description - large text string, allow null/empty
   - imageUrl - with a default value, allow null/empty
2. Write a "student" model with the following information:
   - firstName - not allow null/empty
   - lastName - not allow null/empty
   - email - not allow null/empty
   - imageUrl - with a default value, allow null/empty
   - gpa - decimal between 0.0 and 4.0, allow null/empty
3. Student can be associated with at most one campus
4. Campus can be associated with many students

API/Server-Side Routing (Express, Sequelize) – back-end application

1. Write a route to serve up all students
2. Write a route to serve up all campuses

Single Campus and Single Student

UI (React) – front-end application

1. Write a component to display a single campus in the **Single Campus View** with the following information:
   - The campus's name, image, address, and description
   - A list of the names of all students in that campus (or a helpful message if it doesn't have any students)
2. Write a component to display a single student in the **Single Student View** with the following information:
   - The student's full name (first and last names), email, image, and GPA
   - The name of the student's campus (or a helpful message if the student doesn't have one)

Client-Side Routing (React Router) – front-end application

1. Display the specific campus's information when the URL matches "/campus/:campusId"
2. Display the specific student's information when the URL matches "/student/:studentId"
3. Clicking on the name of a campus from the **All Campuses View** should navigate to show that campus in the **Single Campus View**

4. Clicking on the name of a student from the **All Students View** should navigate to show that student in the **Single Student View**
5. Clicking on the name of a student in the **Single Campus View** should navigate to show that student in the **Single Student View**
6. Clicking on the name of a campus in the **Single Student View** should navigate to show that campus in the **Single Campus View**

API/Server-Side Routing (Express, Sequelize) – back-end application

1. Write a route to serve up a single campus (based on the campus id), including that campus's students
2. Write a route to serve up a single student (based on the student id), including that student's campus

Adding a Campus and Adding a Student

UI (React) – front-end application

1. Write a component to display a form in the **Add Campus View** for adding a new campus, which contains input fields for all campus information
2. Submitting the form with valid inputs should:
   - Make a request that causes the new campus to be persisted in the database
   - Add the new campus to the list of campuses without needing to refresh the web page
3. Write a component to display a form in the **Add Student View** for adding a new student, which contains input fields for all student information
4. Submitting the form with valid inputs should:
   - Make a request that causes the new student to be persisted in the database
   - Add the new student to the list of students without needing to refresh the web page

API/Server-Side Routing (Express, Sequelize) – back-end application

1. Write a route to add a new campus
2. Write a route to add a new student

Editing a Campus and Editing a Student

UI (React) – front-end application

1. Write a component to display a form in the **Edit Campus View** for editing a campus, which contains fields for all campus information
2. Submitting the form with valid inputs should:
   - Make a request that causes the campus to be updated in the database
   - Display the updated campus information without needing to refresh the web page
3. Write a component to display a form in the **Edit Student View** for editing a student, which contains fields for all student information
4. Submitting the form with valid inputs should:
   - Make a request that causes the student to be updated in the database
   - Display the updated student information without needing to refresh the web page

API/Server-Side Routing (Express, Sequelize) – back-end application

1. Write a route to edit a campus (based on the campus id)
2. Write a route to edit a student (based on the student id)

Deleting a Campus and Deleting a Student

UI (React) – front-end application

1. In the **All Campuses View**, include a "Delete" button next to each campus
2. Clicking the "Delete" button should:
    - Make a request that causes that campus to be deleted from the database
    - Delete the campus from the list of campuses without needing to refresh the web page
3. In the **All Students View**, include a "Delete" button next to each student
4. Clicking the "Delete" button should:
    - Make a request that causes that student to be deleted from the database
    - Delete the student from the list of students without needing to refresh the web page

API/Server-Side Routing (Express, Sequelize) – back-end application

1. Write a route to delete a campus (based on the campus id)
2. Write a route to delete a student (based on the student id)