<https://priteshs-organization.gitbook.io/assignment-2>

**Home**

This assignment combines a full-stack development project using a variety of popular tools and practices.

**COMP 3123 | Assignment – 2 (12%)**

COMP 3123 | Full Stack Development – I

*This assignment combines a full-stack development project using a variety of popular tools and practices.*

**Important Note: No extensions will be granted for this assignment, as they may negatively impact other coursework and evaluations.**

*Utilize ReactJS concepts such as states, props, Material Design, React-Bootstrap, Fetch/Axios, React Router, class components, function components, and hooks. Research to ensure your work adheres to industry standards. Use screen designs as references but create*

**Assignment**

This assignment combines a full-stack development project using a variety of popular tools and practices. Here’s a breakdown of the main tasks and guidelines for each part to help you get started:

**Part I: Backend with NodeJS, Express, and MongoDB / Assignment 1**

1. **API Development:** You will refer to **Assignment 1** to create a backend application using NodeJS, Express and MongoDB. This application should expose RESTful APIs for performing CRUD operations on user and employee data (for example, signup, login, adding, viewing, updating, and deleting employee records).

* Build a RESTful API using Node.js, Express, and MongoDB.
* Set up endpoints for CRUD operations on user and employee data, such as:
  + **User Endpoints**: POST /signup, POST /login
  + **Employee Endpoints**: POST /employees, GET /employees, GET /employees/:id, PUT /employees/:id, DELETE /employees/:id, /search
* Use validation to handle various data inputs and error scenarios.

1. **Axios for API Consumption**

* On the ReactJS frontend, make HTTP requests to your backend API using Axios. This ensures separation between frontend and backend and allows you to efficiently make data requests to the backend.

**Important: Ensure the MongoDB connection is not established on the frontend; it should only reside in the backend.**

1. **Docker Setup or Cloud Deployment**

* [**Docker**](https://priteshs-organization.gitbook.io/assignment-2/docker-reference): Create a docker-compose.yml file to orchestrate MongoDB, the backend, and the frontend. Each service (frontend, backend, MongoDB) should have its own Dockerfile if deploying with Docker Compose.

**OR**

* **Cloud Deployment**: Alternatively, deploy the backend and frontend on a cloud platform (e.g., [Heroku](https://id.heroku.com/login) / [Vercel](https://vercel.com/new) / [Render](https://render.com/)) for live access.

***Update the backend (assignment 1) whenever required to accommodate the requirement of the frontend.***

**Part II: Frontend with ReactJS**

1. **Application Setup and GitHub repository**
   * Initialize a ReactJS app named in the specified format (studentID\_comp3123\_assignment2\_reactjs) using create-react-app Then, *set up your GitHub repository with the same name and commit/push your changes regularly with descriptive messages.*
   * *NOTE: if you are using* [*docker-compose*](https://priteshs-organization.gitbook.io/assignment-2/docker-reference) *then make a single folder to keep the frontend and backend. For example,* Organize the project structure in a single folder if using Docker:

Copy

studentID\_comp3123\_assignment/

├── docker-compose.yml

├── frontend/

└── backend/

1. **Routing and Navigation**
   * Use react-router-dom for screen navigation:
     + **Login**
     + **Signup**
     + **Employee components**
2. **Login and Signup Screens**
   * Implement login and signup screens with form validations.
   * Use controlled components to manage form input states.
3. **Session Management**
   * Store a user session token in localStorage to persist user authentication or Context API or Redux.

*localStorage.setItem('token', response.data.token);*

1. **Employee Management**
   * After login, navigate users to an **Employee List** screen displaying all employees in a table format.
   * Provide CRUD operations:
     + **Add Employee**: A form to add new employees.
     + **View Details**: A screen/modal displaying specific employee details.
     + **Update Information**: A form for editing employee information.
     + **Delete Employee**: A button to remove an employee record.
2. **Search Functionality**

Add employee search criteria for department or position and display them in the appropriate table format. This feature will allow users to search for employees based on department or position, showcasing your ability to handle dynamic queries and responsive UI. (***Add new REST API to backend***)

1. **User Experience**
   * Focus on professional UI/UX design:
     + Use Material-UI, Bootstrap, or custom CSS for styling.
     + Make sure components are responsive.
2. **Logout and Redirect**
   * Implement a logout button to clear user sessions and redirect users to the login page.

**Reference Sample Screen Designs**

**Sample Design Screens for reference only. Based on the fields of user and employee please update the screens. Make sure to design your own search result screen**

1. List all employees after login

A screenshot of a computer

Description automatically generated

1. On click of “Add Employee” add a new employee by calling API and Delete the employee record when you press the delete button

A screenshot of a computer

Description automatically generated

1. View Employee details when you click on the View button

A screenshot of a computer

Description automatically generated

1. Update employee information when you click the Update button

A screenshot of a computer

Description automatically generated

**Evaluation Criteria**

* **Part I (10%):** Successful deployment of the backend or creation of a Docker Compose file or on any cloud platform. Failure to do so results in a zero.
* **Part II (80%):** Each correctly implemented screen must follow professional design standards. Errors or incomplete features will result in a zero.
* **GitHub Repository (10%):** Proper naming conventions and valid commits are essential. Failure to submit the repository correctly will result in a zero.
* **Validation:** Ensure that your application displays appropriate error messages for all validations.

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Component** | **Points** |
| 1 | Deploy backend and frontend to Heroku/Vercel or OTHER hosting platform or creating docker compose | 10 |
| 2 | Working Signup component screen | 07 |
| 3 | Working Login component and Logout screen | 08 |
| 4 | List all Employee component with good design and theme | 10 |
| 5 | Add New Employee screen with good design and theme and with all validation messages | 10 |
| 6 | View and Update Employee component with good design and theme and with all validation messages | 10 |
| 7 | Delete Employee | 10 |
| 8 | Search employee by department or position | 10 |
| 9 | Accepted UI/UX using material design OR bootstrap, etc. | 10 |
| 10 | Code organization into modules and services | 05 |
| 11 | GitHub repository with valid commits and readme file. Screenshot submission for validation | 10 |