**Lab Assignment #1 – Designing and implementing a modern web app using MERN stack**

Due Date: Sunday midnight, Week 4.

Purpose: The purpose of this assignment is to provide software engineering students with an opportunity to develop a student/course system using the MERN stack, while gaming students can apply their skills in the context of a game library management system and a UI with visually appealing 3D graphics and interactive elements.

References: Read the reference textbooks, lecture slides, and class examples. This material provides the necessary information that you need to complete the exercises.

Be sure to read the following general instructions carefully:

- This assignment must be completed individually by all the students.

- See the naming and **submission rules** at the end of this document

- You will have to **provide an demonstration for your solution** in class and upload the solution on eCentennial through the assignment link.

**Exercise 1: MERN Stack - Student/Course System (for SET students)**

Create a comprehensive **student/course system** using the MERN stack. Develop an Express REST API to expose CRUD functionalities for managing student and course information. Implement a React front-end to allow students to interact with the system and an admin user to manage student records.

**Requirements:**

1. **Express REST API:**

* Implement CRUD functionalities for managing student and course information.
  + Create a **student** model with the following fields:
    - Student number
    - Password
    - First name
    - Last name
    - Address
    - City
    - Phone number
    - Email
    - Program
    - Add two additional custom fields for student-specific information, for example: favorite topic, favorite assignment, strongest technical skill obtained in this course, etc.
  + Create a **course** model with the following fields:
    - Course code
    - Course name
    - Section
    - Semester
    - Students. This field can be defined as: **students: [{ type: Schema.Types.ObjectId, ref: 'Student' }]**

Store the information in a MongoDB database. You may use the **ref** feature to establish references between student and course documents.

Implement authentication/authorization capabilities using **JWT** and **HTTPOnly** cookies.

1. **React Front-end:**

* Develop a user-friendly UI using **functional components, composition, and React Hooks**.
* Provide a **login functionality** for students.
* Implement the following features for students:
  + **Add** a course
  + **Update** a course (e.g., change section)
  + **Drop** a course
  + **List** all courses taken by a student
* Implement the following features for admin user:
  + Add a student
  + List all students
  + List all courses
  + List all students taking a specific course

1. **MVC Architecture:**

Apply the MVC (Model-View-Controller) principles for the Express API to ensure a structured and maintainable codebase.

Design a nice and friendly UI.

(10 marks)

**Evaluation:**

|  |  |
| --- | --- |
| **Functionality(including code explanation):** |  |
| **React** **front-end (**Correct **components, forms, event handling, login)** | 30% |
| **MongoDB database** (config files, models) | 10% |
| **Express Rest API (**Correct **models**, **controllers**, **routes** code, Correct implementation of MVC architecture, **authentication/authorization,** Correct **server.js**, **express.js**, **mongoose.js** and **config.js** files) | 30% |
| **Friendliness** (using CSS to align the React elements, React-Bootstrap, etc.) | 10% |
| **Innovative features** (extensive use of ES6+ syntax, UI design, etc.) | 10% |
| In class Code demonstration with explanation | 10% |
| **Total** | **100%** |

**Exercise 2: Game Library Management System with Enhanced UI (for G-P students)**

**Requirements:**

1. **Express REST API:**

* Develop an Express REST API that supports CRUD operations for managing games and user collections.
* Create the following models with their respective fields:
  + Game Model:
    - title (string): The title of the game.
    - genre (string): The genre of the game.
    - platform (string): The platform(s) on which the game can be played.
    - releaseYear (number): The year the game was released.
    - developer (string): The developer of the game.
    - rating (number): The rating of the game.
    - description (string): A description of the game.
  + User Model:
    - username (string): The username of the user.
    - password (string): The password of the user.
    - games (array of game IDs): The collection of games owned by the user. This field in the game model can be defined as: **games: [{ type: Schema.Types.ObjectId, ref: Game' }]**

1. **React Front-end:**

* Design a user-friendly UI using functional components, composition, and React Hooks.
* Implement the following features for users:
  + Register: Allow users to register an account with a unique username and password.
  + Login: Provide a login form for users to access their game library.
  + Add games: Allow users to add games to their collection by searching and selecting from the available games.
  + Remove games: Enable users to remove games from their collection.
  + Display game details: Show detailed information about each game in the user's collection.
  + List all games: Display a list of all available games in the library.
* Integrate three.js into the React UI:
  + Incorporate three.js to enhance the UI of the game library management system.
  + Utilize three.js features to create visually appealing elements such as 3D buttons, dynamic backgrounds, or animated transitions.
  + Focus on incorporating simpler three.js elements that enhance the user experience.

1. **MVC Architecture:**

* Apply the MVC principles to maintain a structured codebase.

**Evaluation:**

|  |  |
| --- | --- |
| **Functionality(including code explanation):** |  |
| **React** **front-end (**Correct **components, forms, event handling, login, interactive graphics)** | 35% |
| **MongoDB database** (config files, models) | 5% |
| **Express Rest API (**Correct **models**, **controllers**, **routes** code, correct implementation of MVC architecture, **authentication/authorization,** Correct **server.js**, **express.js**, **mongoose.js** and **config.js** files) | 25% |
| **Friendliness** (using CSS to align the React elements, React-Bootstrap, etc.) | 10% |
| **Innovative features** (extensive use of ES6+ syntax, UI design, Three.js, etc.) | 15% |
| In class Code demonstration with explanation | 10% |
| **Total** | **100%** |

**VS Code Project Naming rules:**

You must name your **VS Code** project/folder according to the following rule:

**YourFullName\_COMP308LabNumber\_ ExNumber**.

Example: **JohnSmith\_COMP308Lab1\_ Ex1**

**Submission rules:**

**Remove the node\_modules folder before zipping the project.** Submit your project as a **zip file** that is named according to the following rule:

**YourFullName \_COMP308LabNumber\_ExNumber.zip**

Example: **JohnSmith\_COMP308Lab1\_Ex1.zip**

**Include the names of both pair programmers.**

**DO NOT use RAR or other types of archives.**