## Page:1

# **Project 1**

**Movie Projects**

**Objective:** Create a full-stack web application titled "Movie Projects" using **React.js** for the frontend and **Node.js**, **Express**, and **MongoDB** for the backend. The application should allow users to perform CRUD (Create, Read, Update, Delete) operations on a collection of movies and implement authentication and authorization using **JWT** and **bcrypt**.

**Requirements:**

**Frontend (React.js):**

Build a user interface that allows the following functionalities:

1. **Authentication**:
   * **Login Page**: A form for users to log in.
   * **Registration Page**: A form to register new users.
   * Store JWT token in localStorage for authenticated users.
2. **Movie Operations**:
   * **Create a New Movie**: A form to add new movies (available only for authenticated users).
   * **View All Movies**: Display a list of all movies in the database with details like title, genre, director, release year, and description.
   * **Update a Movie**: A form to update details of an existing movie (available only for authenticated users).
   * **Delete a Movie**: A button to delete a movie (available only for authenticated users).
3. The frontend should be responsive and styled using **CSS** or **Bootstrap**.

**Backend (Node.js, Express, MongoDB):**

Create an Express API to handle the CRUD operations and implement authentication and authorization using **JWT** and **bcrypt**.

1. **Authentication and Authorization**:
   * **POST /auth/register**: Register a new user. Passwords should be hashed using bcrypt.
   * **POST /auth/login**: Authenticate a user and generate a JWT token.

## Page:2

* + **Protected Routes**: Access to certain routes should be restricted to authenticated users via JWT.

1. **Movie CRUD Operations**:
   * **GET /movies**: Fetch all movies from the MongoDB database.
   * **POST /movies**: Add a new movie to the database (authentication required).
   * **PUT /movies/**: Update an existing movie by its ID (authentication required).
   * **DELETE /movies/**: Delete a movie by its ID (authentication required).
2. Use **Mongoose** to define the movie schema and interact with MongoDB.

**MongoDB:**

The **Movie** model should have the following fields:

* **Title** (String)
* **Genre** (String)
* **Director** (String)
* **Release Year** (Number)
* **Description** (String)

The **User** model should have the following fields:

* **Username** (String)
* **Email** (String)
* **Password** (hashed using bcrypt)
* **Role** (optional, for role-based access control)

**Routing & Communication:**

* Set up **Axios** or **Fetch API** to communicate between the React frontend and Express backend.
* Ensure that the frontend correctly handles server responses (e.g., displaying success/error messages).

**Instructions:**

**Backend Setup (Node.js, Express, MongoDB):**

## Page:3

1. Initialize a new Node.js project and install the required dependencies (express, mongoose, cors, dotenv, bcryptjs, jsonwebtoken, etc.).
2. Create routes and controllers to handle CRUD operations and authentication.
3. Set up a MongoDB connection using Mongoose.
4. Implement authentication using **JWT** and password hashing using **bcrypt**.

**Frontend Setup (React.js):**

1. Initialize a React app using **Create React App**.
2. Install Axios or use Fetch API for HTTP requests.
3. Create components for:
   * **Movie List**: Displays all movies.
   * **Movie Form**: For adding and updating movies.
   * **Movie Details**: Displays detailed information about a movie.
   * **Login** and **Register**: Forms for user authentication.
4. Implement routing using **React Router**.
5. Use protected routes to secure access to certain pages based on authentication.

**Folder Structure:**

**Backend:**

Copy code

backend/

├── controllers/

│ └──

bash

Copy code

│ └── movieController.js

│ └── authController.js

├── models/

│ └── movie.js

## Page:4

│ └── user.js

├── routes/

│ └── movieRoutes.js

│ └── authRoutes.js

├── middleware/

│ └── authMiddleware.js

├── app.js

└── .env

**Frontend (React):**

java

Copy code

frontend/

├── src/

│ ├── components/

│ │ ├── MovieList.js

│ │ ├── MovieForm.js

│ │ ├── MovieDetails.js

│ │ ├── Login.js

│ │ └── Register.js

│ ├── App.js

│ ├── index.js

├── public/

└── package.json

**Submission:**

1. Upload the project to GitHub (both frontend and backend).
2. Submit a **README** file with instructions on how to run the project.

## Page:5

**Evaluation Criteria:**

* **Functionality (30%)**: Full CRUD functionality and authentication using JWT and bcrypt are implemented.
* **Code Structure (25%)**: Proper project structure with clean and modular code.
* **UI/UX (20%)**: Frontend is responsive and user-friendly.
* **Database Integration (15%)**: Correct implementation of MongoDB using Mongoose.
* **Bonus Features (10%)**: Search, pagination, Redux, role-based access control, etc.

**Deadline:**

Provide the due date here.

**Project 2**

**Convert HTML to Node js**

**Objective:**

Create a multi-page web application with a frontend using EJS (Embedded JavaScript) for templating and a backend built with Node.js, Express, and MongoDB. The project should follow the **MVC (Model-View-Controller)** architecture.

**Requirements:**

1. **Pages to Implement:**
   * **Home** (e.g., a welcoming page with an introduction to the website).
   * **About** (a page with information about the website/company).
   * **Contact** (a form to collect user queries; data should be saved in MongoDB).
   * **Product** (a list of products retrieved from MongoDB, displayed on the frontend).
2. **Backend:**
   * Use **Node.js** and **Express** to handle server-side logic.

## Page:6

* + **MongoDB** should be used for storing dynamic data (like product information or contact form submissions).
  + Follow the **MVC pattern** to structure your application:
    - **Models:** Handle the database schema and interactions.
    - **Views:** Create EJS templates for each page.
    - **Controllers:** Manage the logic for each route and interact with the models.
  + Set up the application so that the root URL ("/") serves the Home page.

1. **Frontend (EJS):**
   * Use **EJS** to render dynamic content on all pages.
   * Pass data from the controllers to the views using EJS.
   * Ensure that each page (Home, About, Contact, Product) has a consistent layout (e.g., a common header and footer).
2. **Database (MongoDB):**
   * Store dynamic data like:
     + **Products**: Each product should have fields like name, price, description, and image URL.
     + **Contact submissions**: Save user queries in a collection with fields like name, email, and message.
   * Set up a **MongoDB connection** using the mongoose library.
3. **Routing:**
   * Implement routing using Express for each page:
     + GET / → Home Page
     + GET /about → About Page
     + GET /contact → Contact Page
     + POST /contact → Handle form submission and save it to MongoDB.
     + GET /products → Display a list of products from MongoDB..

**Instructions:**

1. **Setup:**
   * Initialize your project using npm init.
   * Install the necessary dependencies: express, mongoose, ejs, and any other libraries you may need (e.g., dotenv).

## Page:7

* + Create a .gitignore file to exclude node\_modules and sensitive files (e.g., .env).

1. **Folder Structure (MVC Pattern):**
2. arduino
3. Copy code
4. project-folder/
5. ├── models/
6. │ ├── product.js
7. │ └── contact.js
8. ├── views/
9. │ ├── home.ejs
10. │ ├── about.ejs
11. │ ├── contact.ejs
12. │ └── products.ejs
13. ├── controllers/
14. │ ├── productController.js
15. │ └── contactController.js
16. ├── routes/
17. │ ├── index.js
18. ├── public/ (for static assets like CSS, images)
19. ├── app.js (or server.js)
20. ├── .env (for environment variables)
21. └── package.json
22. **Submission:**
    * Upload your project to GitHub.
    * Provide a link to your GitHub repository along with any additional instructions on how to run the project locally.

**Evaluation Criteria:**

1. **Correct Implementation of MVC Pattern (25%)**

## Page:8

1. **Proper Use of EJS for Templating (20%)**
2. **Working MongoDB Integration (20%)**
3. **Routing and Form Handling (20%)**
4. **Code Quality and Project Structure (15%)**

**Project 3**

**Book Store Details**

**Objective:**

Create a full-stack web application titled **"Book Store Details"** using **React.js** for the frontend and **Node.js, Express, MongoDB** for the backend. The application should allow users to perform **CRUD (Create, Read, Update, Delete)** operations on a collection of books.

**Requirements:**

1. **Frontend (React.js):**
   * Build a user interface that allows the following functionalities:
     + **Create a New Book**: A form to add new books.
     + **View All Books**: Display a list of all books in the store with details like title, author, price, and description.
     + **Update a Book**: A form to update details of an existing book.
     + **Delete a Book**: A button to delete a book from the store.
   * The frontend should be responsive and styled using **CSS** or **Bootstrap**.
2. **Backend (Node.js, Express, MongoDB):**
   * Create an **Express** API to handle the CRUD operations:
     + **GET /books**: Fetch all books from the MongoDB database.
     + **POST /books**: Add a new book to the database.
     + **PUT /books/**: Update an existing book by its ID.
     + **DELETE /books/**: Delete a book by its ID.
   * Use **Mongoose** to define the book schema and interact with MongoDB.

## Page:9

1. **MongoDB**:
   * The **Book** model should have the following fields:
     + Title (String)
     + Author (String)
     + Price (Number)
     + Description (String)
     + ISBN (String)
   * Store the book details in a MongoDB database.
2. **Routing & Communication**:
   * Set up **Axios** or **Fetch API** to communicate between the React frontend and Express backend.
   * Ensure that the frontend correctly handles server responses (e.g., displaying success/error messages).

**Instructions:**

1. **Backend Setup (Node.js, Express, MongoDB):**
   * Initialize a new Node.js project and install the required dependencies (express, mongoose, cors, dotenv, etc.).
   * Create routes and controllers to handle CRUD operations.
   * Set up a MongoDB connection using **Mongoose**.
2. **Frontend Setup (React.js):**
   * Initialize a React app using **Create React App**.
   * Install Axios or use Fetch API for HTTP requests.
   * Create components for:
     + **Book List**: Displays all books.
     + **Book Form**: For adding and updating books.
     + **Book Details**: Displays detailed information about a book.
   * Implement routing using **React Router**.

**Folder Structure:**

* **Backend:**
* Bash

## Page:10

* Copy code
* backend/
* ├── controllers/
* │ └── bookController.js
* ├── models/
* │ └── book.js
* ├── routes/
* │ └── bookRoutes.js
* ├── app.js
* └── .env
* **Frontend (React):**
* java
* Copy code
* frontend/
* ├── src/
* │ ├── components/
* │ │ ├── BookList.js
* │ │ ├── BookForm.js
* │ │ └── BookDetails.js
* │ ├── App.js
* │ ├── index.js
* ├── public/
* └── package.json

**Submission:**

* Upload the project to GitHub (both frontend and backend).
* Submit a README file with instructions on how to run the project.

**Evaluation Criteria:**

## Page:11

1. **Functionality (30%)**: Full CRUD functionality is implemented.
2. **Code Structure (25%)**: Proper project structure with clean code.
3. **UI/UX (20%)**: Frontend is responsive and user-friendly.
4. **Database Integration (15%)**: Correct implementation of MongoDB using Mongoose.