## MERN STACK - Task 4: Login Page Development

---

## Objective

Develop a basic login page where users can input their credentials to log in. This task involves creating a ReactJS frontend, an ExpressJS backend, and integrating MongoDB for storing and verifying user credentials.

---

## Project Steps

### 1. Frontend Development with ReactJS

- **Features**:
    - A login form with fields for **username/email** and **password**.
    - A submit button to send the credentials to the backend API.
    - Error messages for validation failures, such as missing fields or incorrect credentials.
- **Implementation**:
    - **Client-side Validation**:
        - Check that all fields are filled before submission.
        - Highlight empty fields with error messages.
    - **Styling**:
        - Use a CSS framework like **Bootstrap** or **Tailwind CSS** for a professional design.
        - Arrange the form in a visually appealing layout with responsive design.
- **Flow**:
    - On form submission:
        - Validate inputs.
        - Send a POST request to the backend API with the credentials.
        - Handle the response:
            - Display a success message or redirect to the dashboard on a successful login.
            - Show an error message for failed attempts.

---

### 2. Backend Development with ExpressJS

- **Setup**:
    - Install necessary modules: `express`, `body-parser`, `mongoose`, `cors`.
    - Configure middleware to handle JSON requests and enable CORS.
- **API Endpoint**:
    - Create a POST `/login` endpoint.
    - Parse the received credentials (username/email and password).
- **Logic**:
    - Validate credentials against the database.
    - Return a success response (e.g., `{ status: "success", message: "Login successful" }`) or failure response (e.g., `{ status: "fail", message: "Invalid credentials" }`).

---

**3. Database Integration with MongoDB**

- **Setup**:
    - Use MongoDB Atlas or a local MongoDB instance.
    - Create a `users` collection with fields like `email`, `username`, and `password`.
- **Password Handling**:
    - Store passwords securely using a hashing library like **bcrypt**.
    - During login, hash the input password and compare it to the stored hashed password.
- **Sample Data**:

Insert a few test users into the database:
{ "username": "testuser", "email": "test@example.com", "password": "hashed_password" }

    -

---

**4. Testing and Debugging**

- **Frontend Testing**:
    - Check input validation.
    - Test the form submission and API communication.
    - Verify UI behavior for both success and error responses.

- **Backend Testing**:
    - Test API endpoints with tools like **Postman** or **Thunder Client**.
    - Check database queries to ensure correct validation of credentials.
- **End-to-End Testing**:
    - Run the application and verify:
        - Correct data flow from the frontend to the backend.
        - Proper database operations (queries, updates).

---

**5. Documentation**

- **Setup Instructions**:
    - Steps to clone the repository and install dependencies.
    - Configuration details for connecting the backend to MongoDB.
    - Commands for starting the frontend and backend servers.
- **Data Flow Description**:
    - User submits credentials → Frontend sends data to backend → Backend queries database → Response sent to frontend → Result displayed.
- **API Details**:
    - Endpoint: `/login`
    - Method: POST
    - Request Body: `{ "email": "user@example.com", "password": "password123" }`
    - Response:
        - Success: `{ "status": "success", "message": "Login successful" }`
        - Failure: `{ "status": "fail", "message": "Invalid credentials" }`

---

## Deliverables

1. **Fully Functional Login Page**:
    - Responsive UI for the login form.
    - Working login process from frontend to backend.
2. **API Documentation**:
    - Details on the login endpoint and sample requests/responses.

3. **Database Integration**:
    - MongoDB setup with sample user data.
4. **Testing Report**:
    - Screenshots and details of testing (e.g., API responses, form validation results).

Would you like guidance on implementing specific parts of this task, such as the database connection or error handling?

## Deadline Compliance

- **Restriction**: **Submit the project within 7 days** from the start date.
- **Reason**: Meeting deadlines is crucial in the real-world software development environment. This restriction helps students practice **time management** and **task prioritization**. In professional settings, tight deadlines are often the norm, and learning to meet them without compromising quality is an essential skill.
- **Learning Outcome**: Students will learn to manage their time effectively, complete projects under pressure, and **deliver results on time**, which are all important skills in the workplace.