Lesson 06: Secure Authentication System

Overview:
In this project, we need to build a **simple login application** using **React.js** on the frontend, **Express.js** on the backend, and **MongoDB** as the database. The app will have two types of users: **Admin** and **Customer**. Admin users can register and log in, while customers can also register and log in. The application will store hashed passwords and validate user credentials during login.

Technologies Used:

- **Frontend:**
  - **React.js**: For building the user interface.
  - **Axios**: To make HTTP requests to the backend.
- **Backend:**
  - **Express.js**: For building the backend API.
  - **MongoDB**: For storing user data (email, password, and role).
  - **bcrypt.js**: For hashing passwords securely.

Features:

1. **SignUp (Registration) for Admin and Customer:**
   - Admins and Customers can create accounts with their email and password.
   - Passwords will be hashed before storing in MongoDB using bcrypt.js.
2. **SignIn (Login) for Admin and Customer:**
   - Admins and Customers can log in by providing their email and password.
   - Passwords are compared with the hashed password stored in MongoDB for validation.
3. **Basic Role Management:**
   - The role of the user (either admin or customer) is saved in the database when signing up.
   - After login, we display a simple message based on the role of the user (e.g., "Welcome Admin" or "Welcome Customer").

Task:

The task was to build a simple authentication system where users can SignUp and SignIn as either Admin or Customer. This system does not require session management or JWT tokens but rather focuses on securely storing user data in a MongoDB database and validating login credentials. The core objectives were:

1. **SignUp (Registration):**
    - Create a registration system where users can sign up using their email, password, and role (admin or customer).
    - Hash the password before storing it in the database using bcrypt.js.
    - Ensure that duplicate email registrations are prevented.
2. **SignIn (Login):**
    - Create a login system where users can enter their email and password to authenticate.
    - Compare the provided password with the hashed password stored in MongoDB.
    - Display a role-specific message (e.g., "Welcome Admin" or "Welcome Customer") after a successful login.
3. **Backend Setup:**
    - Use Express.js to handle HTTP requests.
    - Set up MongoDB with Mongoose to store user data (email, password, and role).
    - Implement routes for SignUp and SignIn to interact with the frontend.
4. **Frontend Setup:**
    - Use React.js to create the SignUp and SignIn forms.
    - Use Axios to send HTTP requests to the backend for user registration and login.
    - Display appropriate messages based on the success or failure of the login attempt, depending on the user role (Admin or Customer).

Result:

The result of the project is a **working authentication system** with the following features:

1. **User Registration (SignUp)**:
    - Users (both Admin and Customer) can sign up using their **email** and **password**.
    - The password is hashed before being saved to MongoDB, ensuring secure storage.
    - The role of the user (admin or customer) is specified during registration.
    - Duplicate email registrations are handled with an error message.
2. **User Login (SignIn)**:
    - Users can log in using their **email** and **password**.
    - The password is compared to the hashed password stored in the database to validate the login attempt.
    - If the login is successful, the system displays a message such as **"Welcome Admin"** or **"Welcome Customer"** depending on the user's role.

    - Invalid login attempts (wrong password or non-existent email) are handled with appropriate error messages.
3. **Frontend-Backend Interaction**:
    - The **React.js** frontend communicates with the **Express.js** backend using **Axios** for handling registration and login.
    - After submitting the login form, the user receives a response indicating whether they were successfully logged in and which role they hold.
4. **Overall Outcome**:
    - A **functional authentication system** that allows for both **Admin** and **Customer** users to sign up and log in, with hashed password storage for security.
    - Basic **role-based messaging** is displayed after login (Admin or Customer).
    - This setup serves as a foundation for building more complex applications that may include features like session management or JWT-based authentication.

Rubric:
Your submission will be evaluated based on the following key criteria, each representing a crucial aspect. These criteria are:

Criteria	Description	Status
Functionality	- User registration (SignUp) and login (SignIn) work as expected. - Passwords are securely hashed using **bcrypt.js**. - Role-based authentication (Admin/Customer) is implemented.	
Code Structure & Organization	- Code is clean, well-organized, and modular. - Separation of concerns between frontend (React) and backend (Express).	
Security	- Password hashing and sensitive data handling are secure. - No sensitive data is exposed to the frontend.	
UI/UX	- User interface is simple, clean, and intuitive. - Clear feedback is given to the user on successful or failed actions (e.g., invalid login, successful registration).	
Documentation	- Basic **README** file with project setup and instructions is provided. - Code is adequately commented for clarity.	