Module 10 Assignment: Security and Authentication

Objective

To secure the IELTS Speaking Test platform's APIs using robust authentication and authorization mechanisms, including password hashing, JWT-based authentication, and role-based access control.

Scenario

The IELTS Speaking Test platform needs secure access for both test takers and administrators. As a developer, you are tasked with implementing a secure authentication system on the backend and ensuring authorized access to specific routes on the frontend. This includes password hashing, session management, and role-based access control.

Requirements

1. **Backend Security**:
   - **Password Hashing**:
     - Use `werkzeug.security` to securely hash and verify user passwords.
     - Store hashed passwords in the database when users register.
   - **Token-Based Authentication**:
     - Implement JWT for authentication.
     - Create the following API endpoints:
       - **`POST /api/register`**: Register a new user with fields `name`, `email`, `password`, and `role`. Hash the password before storing it.
       - **`POST /api/login`**: Authenticate the user by verifying the email and password. If valid, return a JWT containing the user's ID and role.
       - **`GET /api/profile`**: Return the authenticated user's profile information. Require a valid JWT in the request header.
   - **Middleware**:
     - Add middleware to verify JWTs on protected endpoints.
     - Return appropriate error messages for expired or invalid tokens.
2. **Frontend Authorization**:
   - **Route Protection**:
     - Use React to create protected routes that are accessible only to authenticated users.
     - Redirect unauthenticated users to the login page.
   - **Role-Based Access Control**:
     - Restrict admin-specific pages (e.g., user management) to users with the `admin` role.

        - Allow test takers to access only test-related pages (e.g., dashboard, test sections).
    - **Session Management**:
        - Store the JWT securely in session storage or local storage on the client side.
        - Add logic to auto-expire sessions based on the token's expiration time.
3. **Error Handling and Security**:
    - Handle authentication errors such as invalid credentials or expired tokens gracefully on both frontend and backend.
    - Ensure sensitive data (e.g., passwords, tokens) is never exposed in error messages or logs.

Deliverables

1. **Backend**:
    - **`app.py`**: Contains routes for registration, login, and profile retrieval.
    - **`middleware.py`**: Middleware for JWT verification.
    - **Database**: Updated schema to include user roles (`admin`, `test_taker`).
2. **Frontend**:
    - **LoginPage.tsx**: A login form that sends email and password to the backend and stores the JWT on success.
    - **ProtectedRoutes.tsx**: A higher-order component or utility to enforce route protection.
    - **AdminDashboard.tsx**: A page accessible only to admins, displaying user management features.
    - **TestDashboard.tsx**: A page accessible only to test takers, displaying test details.
3. **Postman collection or cURL commands** to test backend endpoints.
4. A **README** file explaining:
    - How to set up and test the authentication system.
    - Instructions for testing role-based access control on the frontend.

Submission Guidelines

1. Submit the project in a zipped folder named `Module10_Assignment_<YourName>.zip`.
2. Include detailed documentation for setting up both backend and frontend.
3. Provide screenshots or screen recordings of:
    - Registering a user and logging in.
    - Accessing protected routes with appropriate permissions.
    - Error messages for invalid or expired tokens.

Evaluation Criteria

1. **Backend Security (40%)**:
    - Correct implementation of password hashing and JWT-based authentication.
    - Middleware effectively validates tokens and handles errors.
2. **Frontend Authorization (35%)**:
    - Routes are protected and redirect unauthenticated users correctly.
    - Role-based access control is implemented as specified.
3. **Error Handling and Documentation (15%)**:
    - Authentication errors are handled gracefully on both frontend and backend.
    - README provides clear setup and usage instructions.
4. **Code Quality (10%)**:
    - Proper structure, meaningful variable names, and comments for clarity.