**MODULE - 10**

**Assignment 20: Add Role-Based Access Control**

**Objective**

Implement role-based access control (RBAC) in the IELTS Speaking Test platform to restrict features and APIs based on user roles (e.g., admin, test taker). Provide separate dashboards and permissions for each role.

Optimized Approach for Implementing Role-Based Access Control

**1. Role-Based API Access**

a. Include User Roles in JWT Payload

During the user authentication process, ensure that the user's role(s) are included in the JWT token.

Structure the JWT payload to include user\_id, role, and expiration time.

Example:

{

"user\_id": 123,

"role": "admin",

"exp": 1700000000

}

**b. Middleware to Verify User Roles**

Implement backend middleware to enforce role-based access control.

Middleware will decode the JWT token and verify if the user has the required role for accessing specific API endpoints.

Steps:

Extract JWT token from the request headers.

Decode the token using a secret key and verify the role.

Return appropriate error message and status code if the user's role does not match the required role for the endpoint.

**2. Frontend Role-Based Features**

a. Creating Separate Dashboards

Develop two separate dashboard components: one for admins and one for test takers.

Each dashboard should have features and UI components relevant to the user's role.

b. Dynamic Rendering Based on User Role

Use React context or props to dynamically render components based on the user's role.

Example:

const Dashboard: React.FC<{ role: string }> = ({ role }) => {

return (

<div>

{role === 'admin' ? <AdminPanel /> : <TestTakerPanel />}

</div>

);

};

**3. Navigation and Permissions**

a. Configure Navigation Menus Based on User Role

Adjust navigation menus to show options relevant to the user's role using conditional rendering.

Example:

const Navigation: React.FC<{ role: string }> = ({ role }) => {

return (

<nav>

{role === 'admin' && <a href="/admin-dashboard">Admin Dashboard</a>}

{role === 'test\_taker' && <a href="/test-taker-dashboard">Test Taker Dashboard</a>}

</nav>

);

};

**b. Prevent Unauthorized Access to Restricted Routes**

Implement route guards to check user roles before allowing access to specific routes.

Redirect users to an appropriate page (e.g., access denied) if they attempt to access restricted areas.

**4. Error Handling**

a. Return Appropriate Error Messages for Unauthorized Access

Ensure the backend middleware returns meaningful error messages for unauthorized access attempts.

Example:

Status code 403

Message: "Unauthorized role"

**b. Display Frontend Notifications for Restricted Features**

Implement notifications or alert components on the frontend to inform users if they try to access restricted features.

Ensure the message is user-friendly and clearly indicates the issue.

**Testing**

1. API Testing

Test different API endpoints with various roles to confirm access restrictions.

Simulate unauthorized access attempts and verify that correct error messages and status codes are returned.

2. Frontend Testing

Log in as different users and verify that dashboards and navigation menus adjust dynamically based on the user's role.

Attempt to access restricted routes or features to ensure they are properly handled.

3. Edge Cases

Test scenarios where no role is provided or the JWT token is invalid.

Ensure the system correctly handles these cases, providing proper error feedback.

**Deliverables**

Backend Files: Include middleware for role-based API access.

Frontend Files: Submit React files with role-based dashboard implementation.

Testing Evidence: Provide test cases and evidence of role-based access control.

**Evaluation Criteria**

Role-Based API Access (40%): Enforce role restrictions correctly and securely.

Frontend Role Management (30%): Dynamic rendering of dashboards and features based on roles.

Error Handling (20%): Graceful handling of unauthorized access with proper user feedback.

Submission Completeness (10%): Include all required files and evidence of proper implementation.

.