**MODULE - 10**

**Assignment 19: Implement JWT Authentication**

**Objective**

Secure the backend APIs of the IELTS Speaking Test platform by implementing JWT-based authentication. Use JSON Web Tokens to manage login and access control for test takers and admins.

**Step-by-Step Instruction to Implement JWT Authentication**

**Setup and Initialization:**

Ensure you have your development environment ready with the necessary dependencies.

Use a framework like Flask, Express, or Django for your backend APIs.

Install JWT libraries like PyJWT for Python, jsonwebtoken for Node.js.

**User Login API:**

Create an endpoint /login to handle user authentication.

Accept user credentials (username and password) in the request body.

Verify the credentials against your stored user data (e.g., database).

If authentication is successful, generate a JWT:

The JWT payload should include:

user\_id: A unique identifier for the user.

role: The role of the user (e.g., admin, test taker).

exp: An expiration time for the token.

Return the JWT to the client along with a successful response.

**Token Verification Middleware:**

Create middleware to intercept incoming requests to protected endpoints.

Extract the token from the Authorization header.

Verify the token using the secret key:

If the token is missing or invalid, return a 401 Unauthorized response.

If the token is valid, decode it to retrieve the payload.

Attach the decoded token data (user details) to the request object to be accessible in the endpoint.

**Role-Based Access Control (RBAC):**

Enhance the middleware or create additional wrappers to check the user’s role.

Ensure that protected endpoints check the user’s role and restrict access based on it.

Example (pseudo-code):

If an endpoint requires admin access, verify user\_data['role'] == 'admin'.

**Token Management:**

Set an expiration time for tokens (e.g., 1 hour).

Implement a refresh token mechanism:

Usually, this involves a separate endpoint to issue a new token if a valid (non-expired) refresh token is presented.

The refresh token should have a longer expiration time than the access token.

**Error Handling:**

Implement comprehensive error handling within the middleware and endpoints:

Handle missing tokens, invalid tokens, expired tokens.

Return clear and appropriate error responses (e.g., JSON responses with error messages).

Ensure unauthorized requests return a 401 status code with a relevant message.

**Testing:**

**Login API:**

Test the /login endpoint with valid and invalid credentials.

Verify that a valid JWT is returned with successful logins.

Protected Endpoints:

Test accessing protected endpoints with and without the token.

Ensure unauthorized access is properly restricted.

Validate role-based access by checking endpoints with users having different roles.

Error Handling:

Test scenarios with expired tokens, invalid tokens, and missing tokens.

Confirm that appropriate error messages and statuses are returned.

**Documentation and Submission:**

Document your JWT structure and the role-based restrictions implemented.

Prepare test cases and evidence showcasing secure access to protected endpoints.

Submit the updated backend files, including the login endpoint, middleware, and any other related configurations.

**Optimized Approach Summary**

Efficiency: By using JWTs, you avoid excessive database queries, keeping authentication lightweight and scalable.

Security: Include expiration and refresh mechanisms to maintain secure token usage, minimizing risks associated with token theft.

Modularity: Use middleware to keep your authentication logic separate and reusable across different endpoints.

Role Management: Embed role information in the token and validate roles at request time to enforce RBAC smoothly.

This approach ensures a secure, scalable, and maintainable solution for managing authentication and access control using JWTs in your backend application.