**MODULE - 9**

**Assignment 18: Implement Real-Time Updates**

**Objective**

Use WebSocket connections to implement real-time updates for the IELTS Speaking Test platform, allowing users to receive live updates on test status or system events.

**Step-by-Step Approach for Implementing Real-Time Updates using WebSockets**

**1. Backend Setup with Flask-SocketIO**

Install Flask-SocketIO: Ensure Flask-SocketIO is installed in your project.

Initialize SocketIO: Integrate SocketIO with your Flask app.

Create a SocketIO instance.

Set up event handlers for connect, disconnect, and custom events such as test\_started, timer\_updates, and admin\_notifications.

Broadcasting Events: Implement functions to broadcast test status updates and other real-time events.

Testing Back-End: Use tools like Postman or WebSocket testing tools to test WebSocket connections and event broadcasting.

**2. Frontend Integration with React using Socket.io-client**

Install Socket.io-client: Ensure socket.io-client is installed in your React project.

Establish WebSocket Connections: Create a connection from the React frontend to the Flask backend.

Configure the WebSocket connection, typically in a component or a custom hook.

Write event listeners for real-time updates (e.g., test\_started, timer\_updates, admin\_notifications).

Update the UI: Implement logic to dynamically update the UI based on incoming WebSocket messages.

Modify React state or use context to propagate updates.

Use components like notification bars or modals to display real-time updates.

Testing Front-End: Simulate backend events and ensure the frontend updates dynamically without page refreshes.

**3. Real-Time Updates Display**

Test Status Updates: Implement a system that displays test status changes (e.g., “Test Starting in 2 Minutes”) in real-time.

Admin Notifications: Develop a notification system for real-time admin messages.

Ensure updates are visually noticeable using elements such as notification bars, pop-ups, or banners.

**4. Error Handling**

Connection Failures: Implement logic to handle WebSocket connection failures.

Show user-friendly error messages.

Attempt automatic reconnections after a defined interval.

Fallback Mechanisms: Ensure critical updates are communicated even if WebSocket fails.

Implement alternative methods like periodic polling via HTTP requests as a fallback.

Testing Edge Cases: Validate reconnection logic and fallback mechanisms by simulating connection disruptions.

**5. Styling and UI Design**

Visual Indicators: Design visual elements to highlight real-time updates.

Consistency: Ensure the updates are consistent with the rest of the UI for a seamless user experience.

**6. Submission Guidelines**

Backend Files: Submit the updated backend files with WebSocket configuration and event handlers.

React Components: Provide React components used for WebSocket integration.

Testing Evidence: Include test cases and evidence (screenshots or recordings) showing successful real-time communication.

Completeness: Ensure all required files and documentation are included in the submission.

**7. Evaluation Criteria**

WebSocket Implementation (40%): Verify backend and frontend WebSocket connections work seamlessly.

Real-Time Updates (30%): Ensure the UI dynamically reflects real-time changes without requiring page refreshes.

Error Handling (20%): Confirm robust handling of connection issues or failures.

Submission Completeness (10%): Ensure all required files and evidence are included.

Final Notes

Testing Tools: Utilize WebSocket testing tools or Postman for backend testing; simulate backend events for frontend testing.

Edge Cases: Thoroughly test edge cases including connection disruptions and reconnection logic to ensure robustness.

Performance Optimization: Ensure the solution is optimized to handle multiple simultaneous connections and real-time events.