**MODULE-7**

**Assignment 13: Add Real-Time AI Question Generation**

**Objective**

**Integrate Azure OpenAI into the IELTS Speaking Test platform to dynamically generate and fetch speaking test questions. Implement an API endpoint in the Flask backend to serve these AI-generated questions to the frontend.**

Sure! Here’s a step-by-step approach to integrating Azure OpenAI into the IELTS Speaking Test platform to dynamically generate and fetch speaking test questions, ensuring an optimized solution that can handle competitive programming test cases:

**Step-by-Step Approach**

**1. Azure OpenAI Setup**

1. **Configure Azure OpenAI**:
   * **Sign Up and Create API Key**: Sign up for Azure OpenAI service and create an API key.
   * **Get Endpoint URL**: Ensure you have the correct endpoint URL to access the OpenAI model.
2. **Secure API Keys Using Environment Variables**:
   * **Store API Key and Endpoint**: Use environment variables to securely store the API key and endpoint URL.
   * **Environment Variable Setup**: Create a .env file to manage these configurations and load them into your Flask application.

**2. Backend Implementation**

1. **Set Up Flask Application**:
   * **Project Structure**: Organize your application’s structure with necessary files like app.py, config.py, and route handlers.
   * **Load Environment Variables**: Use a library like python-dotenv to load environment variables.
2. **Create an API Endpoint**:
   * **Define /api/ai-questions Endpoint**: Create a new route in app.py to define your endpoint.
   * **Request Handling**: Ensure the endpoint can accept parameters such as question type and difficulty from the request.
3. **Integrate Azure OpenAI for Question Generation**:
   * **API Request**: Use the requests library to make HTTP requests to Azure OpenAI. Send required parameters and fetch the AI-generated question.
   * **Process API Response**: Extract the generated question from the response, and structure it in JSON format to send back to the frontend.

**3. Error Handling**

1. **Validation**:
   * **Validate Input Parameters**: Ensure that all required input parameters are provided and are valid.
   * **Verify API Response**: Validate the response from Azure OpenAI for correctness and completeness.
2. **Handle API Errors**:
   * **Manage API Failures**: Implement error handling for scenarios such as network issues, rate limiting, or unexpected API responses.
   * **Return Meaningful Error Messages**: Format error responses in JSON, providing clear and informative messages.

**4. Testing**

1. **Unit Testing**:
   * **Create Test Cases**: Develop test cases for both successful and invalid input scenarios.
   * **Mock API Calls**: Use tools to mock Azure OpenAI API responses for testing without making actual API calls.
2. **Integration Testing**:
   * **End-to-End Testing**: Test the full flow from sending a request to receiving a dynamically generated question.
   * **Load Testing**: Ensure the endpoint can handle multiple simultaneous requests and return results efficiently.
3. **Testing Evidence**:
   * **Document Results**: Gather evidence of successful tests and error handling. Save generated questions as examples.

**Deliverables**

1. **Updated Backend Files**:
   * Submit config.py, app.py, and other relevant backend files, showing the integration with Azure OpenAI and the new endpoint implementation.
2. **Test Results**:
   * Provide documented test results for both successful and failed API calls, showing evidence of correct functionality and error handling.
3. **Example Questions**:
   * Share examples of questions generated by the AI to demonstrate the successful integration.

**Evaluation Criteria**

1. **Integration with Azure OpenAI (40%)**:
   * Correct configuration and usage of the Azure OpenAI API are essential for optimal performance.
2. **Endpoint Functionality (30%)**:
   * Ensure that /api/ai-questions generates and returns valid test questions dynamically based on inputs.
3. **Error Handling (20%)**:
   * Provide clear and meaningful error messages for invalid inputs or failed API calls, ensuring robustness.
4. **Submission Completeness (10%)**:
   * Include all required files, sample test results, and example generated questions.

**Key Considerations for Future Enhancements**

* **Extensibility**:
  + Design the endpoint to allow for future enhancements, such as specifying difficulty levels or different question categories.
* **Security**:
  + Ensure the environment variables for API keys and endpoints are securely managed, avoiding hard-coded sensitive information.

By following these steps and considerations, you can achieve a robust and scalable integration of Azure OpenAI into the IELTS Speaking Test platform, providing dynamic and varied content for the test takers while ensuring optimal performance and error handling.