AI Agent and AI Agent UI Development Plan

1. Overview

This document outlines the steps and tasks required to develop both the AI Agent and the AI Agent UI. The AI Agent is responsible for processing user inputs, interacting with external services (QuestDB, Grafana, VS Code), and generating responses. The AI Agent UI serves as the user interface for interacting with the AI Agent.

2. Development Environment Setup

2.1. Docker and Docker Compose Configuration

- **Task**: Update the existing `docker-compose.yml` file to include both the `ai-agent` and `ai-agent-ui` services.
- **Details**:
  - Add the `ai-agent` and `ai-agent-ui` service definitions.
  - Configure environment variables (e.g., `OPENAI_API_KEY`, `QUESTDB_PG_USER`, `QUESTDB_PG_PASSWORD`, `GRAFANA_API_KEY`, `VSCODE_API_KEY`).
  - Set up dependencies for `ai-agent` on QuestDB, Grafana, and VS Code, and for `ai-agent-ui` on `ai-agent`.
  - Ensure both services are correctly configured and linked.

2.2. Dockerfile Creation

- **Task**: Create Dockerfiles for both the AI Agent and AI Agent UI services.
- **Details**:
  - **AI Agent**:
    - Base the image on a suitable Python image (`python:3.x`).
    - Install required dependencies (e.g., `langchain`, `psycopg2`, `requests`).
    - Copy the AI Agent code and configuration files into the Docker image.
    - Set the appropriate entry point for the AI Agent.
  - **AI Agent UI**:
    - Base the image on a suitable Python image (`python:3.x`).
    - Install Flask and other necessary Python packages.
    - Copy the UI code into the Docker image.
    - Set the appropriate entry point for the Flask application.

2.3. Local Environment Setup (Optional)

- **Task**: Set up a local Python virtual environment for development.
- **Details**:
  - Use `venv` or `conda` to create a virtual environment.
  - Install necessary Python packages (as listed in `requirements.txt`).
  - Set up environment variables locally for testing and development.

3. AI Agent Development

3.1. Input Processing

- **Task**: Implement logic to handle user inputs received from the AI Agent UI.
- **Details**:
  - Define input formats (e.g., JSON).
  - Implement validation and sanitization of inputs.
  - Parse the input to determine the user's intent and required action.

3.2. Interaction with OpenAI API

- **Task**: Integrate the OpenAI API to process natural language inputs.
- **Details**:
  - Use the OpenAI Python SDK to send inputs to the GPT model.
  - Handle API responses, including error management and retry logic.
  - Extract and format the relevant information from the API response to send back to the UI.

3.3. Database Interaction with QuestDB

- **Task**: Implement database queries and interactions using QuestDB.
- **Details**:

3.3.1. PostgreSQL Wire Protocol

  - **Use Case**: For applications where you need to perform standard SQL operations (INSERT, UPDATE, SELECT) efficiently.
  - **Tools**:
    - **psycopg2**: A PostgreSQL adapter for Python that can be used to interact with QuestDB via the PostgreSQL wire protocol.
    - **SQLAlchemy**: An ORM (Object-Relational Mapper) for abstracting and simplifying database interactions, which can be configured to work with QuestDB.
  - **Implementation**:
    - Establish a connection to QuestDB using the PostgreSQL wire protocol.
    - Write and execute SQL queries for inserting, updating, and selecting data.
    - Ensure connection pooling and proper handling of database transactions.

3.3.2. REST API

  - **Use Case**: When you prefer a more lightweight interaction or need to execute specific queries via HTTP requests.
  - **Tools**:
    - **Requests Library**: Use Python's `requests` library to interact with QuestDB's REST API.
  - **Implementation**:
    - Send HTTP requests to QuestDB's REST API endpoints for SQL queries.
    - Handle responses, including parsing JSON data returned by SELECT queries.
    - Implement error handling and retry logic for robust interactions.

3.3.3. Query Optimization and Best Practices

- **Use Case**: Optimize performance for data retrieval and manipulation.
- **Implementation**:
  - Optimize SQL queries by indexing frequently accessed columns.
  - Use batch inserts and updates when dealing with large datasets to reduce overhead.
  - Monitor and analyze query performance, adjusting queries as needed to ensure efficiency.

3.4. Integration with Grafana

- **Task**: Set up communication between the AI Agent and Grafana for data visualization.
- **Details**:
  - Use Grafana's API to fetch or update dashboards.
  - Implement logic to trigger Grafana visualizations based on user queries.
  - Handle API authentication using the `GRAFANA_API_KEY`.

3.5. Integration with VS Code

- **Task**: Implement code-related queries or tasks that interact with VS Code.
- **Details**:
  - Use VS Code API (or custom implementation) to handle code execution, file editing, etc.
  - Manage authentication using the `VSCODE_API_KEY`.
  - Define input/output formats for code-related tasks.

4. AI Agent UI Development

4.1. UI Design and Setup

- **Task**: Design the user interface layout and set up the Flask application.
- **Details**:
  - Create wireframes or mockups to outline the UI.
  - Set up the Flask project structure, including templates and static files.

4.2. Front-End Development

- **Task**: Implement the HTML, CSS, and JavaScript for the UI.
- **Details**:
  - Use a CSS framework (e.g., Bootstrap or Tailwind CSS) for styling.
  - Implement the chatbot interface using HTML and JavaScript.
  - Integrate React.js (optional) for more dynamic and interactive components.

4.3. Back-End Development

- **Task**: Set up Flask routes and API endpoints for the UI.
- **Details**:
  - Create Flask routes to handle user inputs and send them to the AI Agent.
  - Implement WebSocket communication (optional) for real-time updates.
  - Ensure proper session management and authentication.

4.4. Integration with AI Agent

- **Task**: Connect the AI Agent UI with the AI Agent via RESTful APIs.
- **Details**:
    - Define API endpoints for sending user queries and receiving responses.
    - Handle different response types (e.g., text, links, images) in the UI.
    - Implement error handling and feedback mechanisms for the user.

4.5. Testing and Debugging

- **Task**: Test the UI for functionality, usability, and responsiveness.
- **Details**:
    - Write unit tests for Flask routes and API endpoints.
    - Test the front-end for cross-browser compatibility and responsiveness.
    - Debug any issues that arise during testing.

5. Performance and Optimization

5.1. API Efficiency

- **Task**: Optimize API calls between the AI Agent and external services.
- **Details**:
    - Implement request batching if possible.
    - Use caching mechanisms for frequently requested data or API calls.
    - Handle rate limiting gracefully.

5.2. Database Query Optimization

- **Task**: Optimize database interactions to ensure quick response times.
- **Details**:
    - Review and optimize SQL queries.
    - Implement indexing or other performance-enhancing database techniques.
    - Consider query caching where applicable.

5.3. Background Task Processing (Optional)

- **Task**: Implement background task processing for long-running tasks.
- **Details**:
    - Use Celery or another task queue to offload long-running processes.
    - Ensure tasks can be queued, executed asynchronously, and results communicated back to the AI Agent UI.

6. Documentation

6.1. Code Documentation

- **Task**: Document the AI Agent and AI Agent UI codebase for maintainability.
- **Details**:
    - Add docstrings to functions, classes, and modules.
    - Comment on complex or non-obvious code sections.

6.2. User Documentation

- **Task**: Write usage instructions for the AI Agent and AI Agent UI.
- **Details**:
  - Create a `README.md` detailing setup, configuration, and usage.
  - Provide examples of input formats and expected outputs.

6.3. API Documentation

- **Task**: Document the API endpoints exposed by the AI Agent.
- **Details**:
  - List available API routes, parameters, and expected responses.
  - Provide examples for common API interactions.

7. Final Steps

7.1. Deployment Preparation

- **Task**: Prepare the AI Agent and AI Agent UI for deployment.
- **Details**:
  - Finalize Docker images and push to the Docker registry.
  - Ensure all environment variables and configurations are correctly set for production.

7.2. Final Testing

- **Task**: Conduct final testing in a staging environment.
- **Details**:
  - Verify all functionalities in an environment that mirrors production.
  - Conduct load testing to ensure performance under expected user load.

7.3. Deployment

- **Task**: Deploy the AI Agent and AI Agent UI to the production environment.
- **Details**:
  - Use the updated Docker Compose configuration to bring up all services.
  - Monitor logs and performance post-deployment to ensure stability.

Estimated Time for Completion

- **AI Agent Development**: 46-81 hours
- **AI Agent UI Development**: 55-85 hours
- **Total Estimated Time**: 101-166 hours