Objective:

The task is to develop a small web service that integrates with OpenAI's GPT-4 API and exposes an endpoint for querying the GPT model. The service should process user input, modify the prompt dynamically based on user type, and return the response in a structured format.

Task Overview:

```
1.  Develop a Backend Service (using Python and FastAPI):
    •   Create a REST API with at least one POST endpoint that accepts a
JSON payload with a user query.
    •   The API should interact with OpenAI's GPT-4 API to generate
responses.
2.  Role-Based Behavior:
    •   The user should provide their role (e.g., "admin" or "user") in
the request.
    •   Based on the role, modify the prompt to change how the model
responds. For example:
    •   Admin: The GPT response should be more formal and provide more
detailed explanations.
    •   User: The GPT response should be simpler and more casual.
3.  Query Parameters:
    •   The request payload should include:
        •   role: a string to define the user role (admin/user).
        •   query: a string representing the user's question or input.
4.  Response Structure:
    •   The API response should be structured as follows:
```

```
{
"original_query": "User's original query",
"gpt_response": "GPT-generated response",
"timestamp": "Time when the response was generated"
}
```

```
5.  Bonus:
    •   Implement logging that tracks the original query, GPT's response,
and the user role.
    •   Add error handling for potential issues such as missing inputs,
invalid API keys, or network errors.
```

Instructions for Submission:

```
•   Language: Use Python for the backend.
•   Framework: FastAPI for building the API.
•   OpenAI API: Integrate with the GPT-4 API using OpenAI's official
library.
•   Documentation: Provide clear instructions for how to:
•   Set up the environment.
•   Run the application locally.
•   Test the API using sample curl commands or a tool like Postman.
•   Time Estimate: ~2-4 hours.
•   Bonus Tasks: Optional but highly recommended to attempt for extra
credit.
```

Sample Input and Output:

Request:

```
{
"role": "admin",
"query": "What is the current status of artificial intelligence?"
}
```

Response:

```
{
"original_query": "What is the current status of artificial
intelligence?",
"gpt_response": "As of 2024, artificial intelligence continues to advance
in various fields, including natural language processing, robotics, and
healthcare...",
"timestamp": "2024-10-07T10:15:30Z"
}
```

Evaluation Criteria:

```
1.  API Functionality (40%):
    •   Does the API work as expected? Is it able to correctly handle
role-based modifications to the prompt?
    •   Are the responses from GPT accurate and well-structured?
2.  Code Quality (25%):
    •   Is the code clean, modular, and well-organized?
    •   Is there proper error handling and logging?
3.  Documentation (10%):
    •   Are the setup instructions clear?
    •   Can the reviewer easily run the code locally?
4.  Bonus Tasks (Optional) (15%):
    •   Logging of requests and responses.
    •   Handling API failures gracefully (e.g., retry logic or informative
error messages).
5.  Creativity and Efficiency (10%):
    •   Did the candidate go beyond the basic requirements?
    •   Is the solution efficient and scalable?
```

Deliverables:

```
•   A zip file containing the project source code.
•   Clear documentation in the README.md file on how to set up and run the
application.
•   Any additional notes on potential improvements or challenges faced
during development.
```

Be prepared to answer questions about building custom GPT models, API design, and error handling strategies.