Project Title: Smart Task Manager

Goal: Create an API that allows users to manage tasks, leveraging AI for task categorization and summary generation.

Functionality:

The API should provide the following endpoints:

1. **/tasks (GET):**
    - Retrieves a list of all tasks stored in Firestore.
    - Each task should have at least the following fields: id (generated by Firestore), title (string), description (string), category (string, initially empty), summary (string, initially empty), created_at (timestamp).
2. **/tasks (POST):**
    - Creates a new task.
    - Accepts a JSON body with title (string) and description (string).
    - Upon task creation:
        - Generates a unique ID for the task (handled by Firestore).
        - Saves the task details to Firestore.
        - **AI Integration 1: Task Categorization:** Uses the OpenAI API to categorize the task based on its title and description. Updates the category field in Firestore with the generated category.
        - Returns the newly created task (including the generated ID and category).
    -
3. **/tasks/{task_id} (GET):**
    - Retrieves a specific task by its task_id.
    - Returns the task details if found, otherwise, returns an appropriate error response (e.g., 404 Not Found).
4. **/tasks/{task_id}/summarize (POST):**
    - Takes a task_id as path parameter.
    - **AI Integration 2: Summary Generation:** Uses the OpenAI API to generate a summary of the task's description. Updates the summary field of the specified task in Firestore with the generated summary.
    - Returns the updated task (including the generated summary).

Technical Requirements:

- **FastAPI:** Use FastAPI to build the API.
- **Firestore:** Use the Google Cloud Firestore Python client library to interact with Firestore for data persistence.
    - Assume a Firestore collection named "tasks" will be used.
- **OpenAI API:** Use the OpenAI Python library to interact with the OpenAI API for task categorization and summarization.
    - Use the ***gpt-4o-mini*** model.
    - Provide clear prompts to the AI for desired behavior.
    - Handle API key management securely (e.g., via environment variables).
- **Error Handling:** Implement appropriate error handling for API requests, Firestore operations, and OpenAI API calls.
- **Data Validation:** Implement basic data validation (e.g., checking for missing fields).
- **Dependencies:** Manage project dependencies using requirements.txt.
- **Code Quality:** Write clean, well-documented, and organized code.

Deliverables:

- The complete Python code for the FastAPI application.
- A requirements.txt file listing all dependencies.
- A README.md file that includes:
    - Instructions on how to set up the environment (including how to connect to Firestore and configure the OpenAI API).
    - Clear instructions on how to run the application and use the API endpoints (provide examples).
    - Any assumptions made.
- Dockerfile