**Multi-Agent Code Generation System**

**Project Overview:**

A multi-agent system that can produce, debug, optimize, and describe code in response to user input is deployed on Hugging Face Spaces in this project. For user interaction, it makes use of a Gradio interface and the OpenAI Chat Completions API.

**System Architecture:**

**1. Agent Modules (code\_generator.py, code\_debugger.py, code\_optimizer.py, code\_doc\_generator.py):**

* Distinct Python files, each having an agent in charge of a certain duty related to code processing.
* To create, improve, and document code, agents communicate with the OpenAI API through requests.

**2. Gradio Interface (app.py):**

* A web-based user interface for a Gradio program.
* Responds to user input (prompt and API key) and shows the agents' output.
* Contains status messages and error handling.

**3. Hugging Face Space:**

* Hosts the Gradio application and its dependencies.
* Manages environment variables (API key) securely via Secrets.

**4. OpenAI API:**

* Provides the language models used by the agents.

**5. Dependencies (requirements.txt):**

* Specifies required Python packages (gradio, requests).

**Functionality:**

* **User Input:** Using the Gradio interface, users enter their OpenAI API key and a code prompt.
* **Pipeline for Code Processing**:
  + the first code is produced by the code\_generator.
  + the code\_debugger looks for mistakes in the code that is produced.
  + the debugged code is optimized by the code\_optimizer.
  + the optimized code's documentation is produced by the code\_doc\_generator.
* **Output Display**: Within the Gradio interface, the outcomes from every agent are shown distinct tabs.
* **Error Handling**: The program shows helpful error/status messages and

manages API errors.

* **API Key Security**: A Hugging Face Secret is used to safely store the OpenAI API key.

**Deployment:**

1. Local Development: Create the application locally and test it there. Make requirements.txt, agent files, and app.py.
2. Hugging Face Space Creation: Establish Hugging Face a new Gradio Space.
3. File Upload: Use the web interface or Git to upload the local files to the Hugging Face Space.
4. Secret Management: In the Space's settings, set the OPENAI\_API\_KEY to a Secret.
5. Autonomous Deployment: Hugging Face Spaces launches app.py and installs dependencies automatically.

**Usage:**

1. Go to the URL for Hugging Face Space.
2. Type your OpenAI API key into a coding prompt.
3. The "Generate and Process Code" button should be clicked.
4. In the appropriate tabs, view the code that has been created, debugged, optimized, and documented.
5. Look for any error messages on the "Error/Status" tab.