```
import asyncio
import semantic_kernel as sk
from semantic_kernel.agents import ChatCompletionAgent
from semantic_kernel.connectors.ai.open_ai import AzureChatCompletion,
OpenAIChatPromptExecutionSettings

# Replace with your actual Azure OpenAI or OpenAI details
AZURE_OPENAI_ENDPOINT = "YOUR_AZURE_OPENAI_ENDPOINT"
AZURE_OPENAI_DEPLOYMENT_NAME =
"YOUR_AZURE_OPENAI_DEPLOYMENT_NAME"
AZURE_OPENAI_API_KEY = "YOUR_AZURE_OPENAI_API_KEY"

async def main():
    # Initialize the Kernel
    kernel = sk.Kernel()

    # Configure Azure OpenAI service
    kernel.add_service(AzureChatCompletion(
        service_id="azure_openai",
        deployment_name=AZURE_OPENAI_DEPLOYMENT_NAME,
        endpoint=AZURE_OPENAI_ENDPOINT,
        api_key=AZURE_OPENAI_API_KEY
    ))

    # Define agent instructions and names
    program_manager_instructions = """You are a program manager responsible for defining
project requirements and overseeing the development process."""
    developer_instructions = """You are a software developer tasked with writing code based
on the program manager's requirements."""
    tester_instructions = """You are a software tester responsible for testing the code
developed by the developer and reporting any bugs."""
    documentation_writer_instructions = """You are a technical writer responsible for creating
documentation for the software application."""
    client_liaison_instructions = """You are the client liaison, responsible for communicating
with the client and gathering their feedback."""
    database_admin_instructions = """You are a database administrator responsible for
designing and managing the application's database."""
    deployment_engineer_instructions = """You are a deployment engineer responsible for
deploying the application to the production environment."""

    # Create the agents
    program_manager_agent = ChatCompletionAgent(name="ProgramManager",
instructions=program_manager_instructions, kernel=kernel)
    developer_agent = ChatCompletionAgent(name="Developer",
instructions=developer_instructions, kernel=kernel)
    tester_agent = ChatCompletionAgent(name="Tester", instructions=tester_instructions,
kernel=kernel)
```

```
    documentation_writer_agent = ChatCompletionAgent(name="DocumentationWriter",
instructions=documentation_writer_instructions, kernel=kernel)
    client_liaison_agent = ChatCompletionAgent(name="ClientLiaison",
instructions=client_liaison_instructions, kernel=kernel)
    database_admin_agent = ChatCompletionAgent(name="DatabaseAdmin",
instructions=database_admin_instructions, kernel=kernel)
    deployment_engineer_agent = ChatCompletionAgent(name="DeploymentEngineer",
instructions=deployment_engineer_instructions, kernel=kernel)

    # Define the initial task
    initial_task = "We need to develop a simple to-do list application with user authentication
and the ability to add, delete, and mark tasks as complete."

    # Project Manager starts the process
    print(f"# User: {initial_task}")
    pm_response = await program_manager_agent.get_response(messages=f"The initial task
is: {initial_task}. Please outline the initial requirements and ask the developer for an
estimated timeline.")
    print(f"# {program_manager_agent.name}: {pm_response.content}")

    # Developer responds to the requirements
    dev_response = await developer_agent.get_response(messages=pm_response.content +
" Based on these requirements, what is your estimated timeline for development?")
    print(f"# {developer_agent.name}: {dev_response.content}")

    # Tester asks about testing strategy
    tester_response = await tester_agent.get_response(messages=f"Considering the
requirements and the developer's timeline: {dev_response.content}, what will be the testing
strategy?")
    print(f"# {tester_agent.name}: {tester_response.content}")

    # Documentation Writer asks for initial documentation plan
    doc_writer_response = await
documentation_writer_agent.get_response(messages=f"With the project starting, what is the
initial plan for creating user and technical documentation?")
    print(f"# {documentation_writer_agent.name}: {doc_writer_response.content}")

    # Client Liaison provides initial client feedback (simulated)
    client_feedback = "The client emphasizes the importance of a user-friendly interface and
secure authentication."
    client_response = await client_liaison_agent.get_response(messages=f"Here is the initial
client feedback: {client_feedback}. How should this be addressed?")
    print(f"# {client_liaison_agent.name}: {client_response.content}")

    # Database Admin asks about database design
    db_admin_response = await database_admin_agent.get_response(messages="Based on
the requirements, what will be the initial database design?")
    print(f"# {database_admin_agent.name}: {db_admin_response.content}")
```

```
    # Deployment Engineer asks about deployment environment
    deployment_response = await
deployment_engineer_agent.get_response(messages="What will be the target deployment
environment for this application?")
    print(f"# {deployment_engineer_agent.name}: {deployment_response.content}")

    # Further interactions can be orchestrated based on the needs of the application
    # For example, the developer might ask the database admin for schema details,
    # the tester might report bugs to the developer, etc.

if __name__ == "__main__":
    import asyncio
    asyncio.run(main())
```

Explanation:

1. **Import Libraries:** Imports the necessary libraries from the Semantic Kernel.
2. **Configure Kernel:** Initializes the Semantic Kernel and configures it to use Azure OpenAI (you can replace this with OpenAI or another supported service). **Remember to replace the placeholder values for your Azure OpenAI credentials.**
3. **Define Agent Instructions:** For each of the seven agents (Program Manager, Developer, Tester, Documentation Writer, Client Liaison, Database Admin, Deployment Engineer), clear and concise instructions are defined as strings. These instructions guide the behavior of each agent.
4. **Create Agents:** Instances of `ChatCompletionAgent` are created for each role, passing their respective names, instructions, and the initialized Kernel.
5. **Define Initial Task:** A string `initial_task` sets the stage for the multi-agent collaboration.
6. **Orchestrate Interactions:** The `main` function then simulates a basic workflow:
    - The `ProgramManager` agent starts by receiving the initial task and outlining requirements.
    - The `Developer` agent responds with an estimated timeline based on these requirements.
    - The `Tester`, `DocumentationWriter`, `ClientLiaison`, `DatabaseAdmin`, and `DeploymentEngineer` agents then chime in with their initial questions and considerations based on the project's kickoff.
7. **Print Interactions:** The code prints the messages exchanged between the user (simulated) and the agents to the console, making it easy to follow the flow of the conversation.

To Run This Code:

Install Semantic Kernel:

```
 Bash
pip install semantic-kernel
```

1.

Install OpenAI or Azure OpenAI Connector:

```
Bash
pip install semantic-kernel[azure]  # For Azure OpenAI
# OR
pip install semantic-kernel[openai] # For OpenAI
```

2.
3. **Replace Placeholders:** Update the `AZURE_OPENAI_ENDPOINT`, `AZURE_OPENAI_DEPLOYMENT_NAME`, and `AZURE_OPENAI_API_KEY` with your actual Azure OpenAI credentials. If you are using OpenAI, you would configure the `OpenAIChatCompletion` service accordingly with your OpenAI API key and model ID.

Run the Python script:

```
Bash
python your_script_name.py
```

4.

This code provides a basic framework for a multi-agent application. You can extend this by:

- Implementing more complex interaction patterns.
- Using planners to automate the flow of tasks between agents.
- Adding memory to agents to retain context across multiple interactions.
- Integrating plugins to allow agents to perform specific actions.
- Defining more sophisticated termination strategies for the multi-agent collaboration.