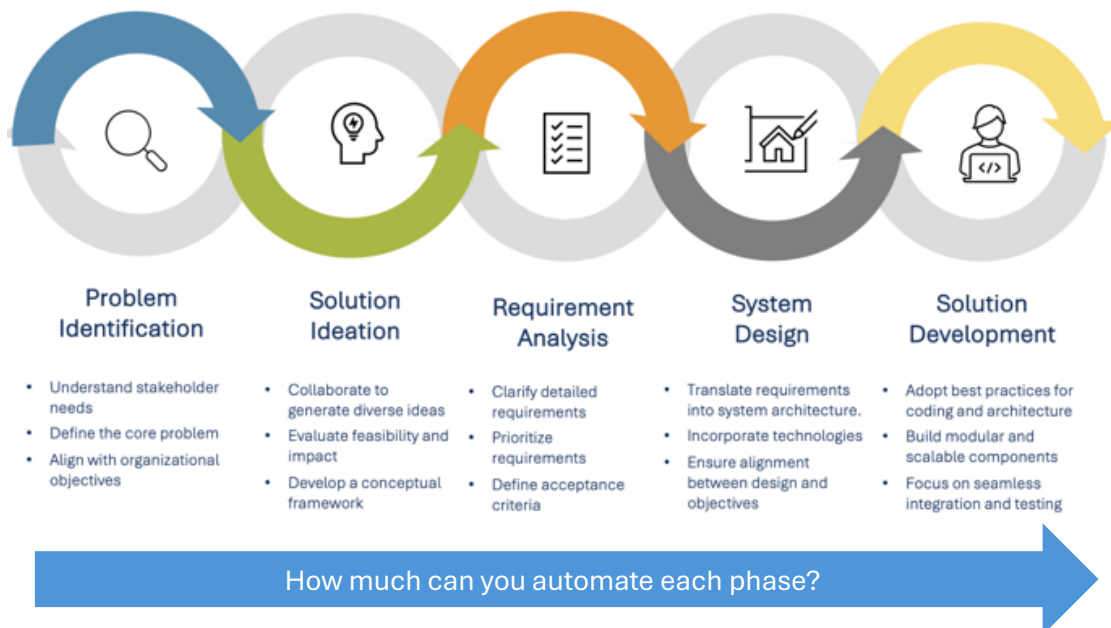




COT6930 – Generative AI and Software Development Lifecycles

FINAL PROJECT INSTRUCTIONS

Last updated: 4/27/2025 12:24:00 PM



The objective of this project is to assess your understanding and mastering of principles of Generative AI applied throughout Software Development Lifecycles. Your goal is to “automate as much as possible” the process of design, analyze, and model a complete software solution from concept to architecture. Your project **MUST** emphasize the use of Generative Intelligence Systems and applicability of GenAI techniques – e.g. prompt engineering, prompt augmentation, pipelines, model interaction, etc -- to the automation of process development. Our goal is to demonstrate a thoughtful, well-structured approach to automate solution development by applying generative AI capabilities.

Deliverables:

You're being asked to:



Curiosity Bot Doc

- Automate as much of the Software Development Lifecycle (SDLC) as possible using Generative AI techniques.
- Design, analyze, and model a complete software solution — from idea to architecture.

Create two key deliverables:

1. FINAL PROJECT REPORT following the template being provided.
2. PROTOTYPE of a small application applying automated software development.

NOTE: link the prototype (e.g Github repo within your FINAL PROJECT REPORT)

General Instructions

- **Group Project.**
- **Follow the FINAL PROJECT Template being provided.**
- **You can choose ANY SOLUTION that you want.**
 - Select a solution that you can describe.
 - Remember that this is a GenAI for SDLC course, not Software Coding.
 - A list of interesting GenAI-based project ideas is provided at:
 - <http://generativeintelligencelab.ai/EXERCISES.html>
- **BE CAREFUL WITH THE REPRESENTATIONS**
 - Apply proper diagrams and Solution Architecture.
 - **Be careful when using GenAI** if the outcoming representation matches the standard.
 - Use of improper or misaligned representations is where **you are most likely to make a mistake that could cost some points!**
- **The prototype implementation is should come out from an automated process.**
 - Indeed, we are all aware of the limitations of fully-automated software development; your challenge is going to the extreme of what is possible; try to automate the process as much as possible, experiment, document, and report what worked, what failed!
 - For impact, we recommend to apply our OwlMind Framework for your prototype; however, this is optional: develop the solution using any framework you prefer!
 - The best OwlMind-based prototypes could figure as SELECTED USE CASES in our project web-site and will be linked to your Github repo.
- **This is a Critical Thinking PLUS mastering of GenAI techniques project.**
 - You're expected to go beyond just using GenAI tools for SDLC tasks.



Curiosity Bot Doc

- You must evaluate their strengths, push boundaries, expose limitations, and reflect on what parts of the SDLC can truly be automated.
- Your innovation and insights matter just as much as technical execution.
- **In case you improve Owlmind to support your prototype (this is GREAT!):**
 - Follow the instructions at: <http://generativeintelligencelab.ai/CONTRIBUTE.html>
 - Create a FORK; submit a PR so we can consider your improvements for the framework
 - **cannot change the Source Code with enhancements** for this exercise (Bug fixes are OK); the objective is to expose the limitations in preparation for the next phase!

Grade Scale

0% to 70%	<p>If your response is superficial, heavily GenAI-based, or you just re-used templates and replaced strings.</p> <p>Your response is not applying the proper representations, the solution is simplistic, or this looks like a last-minute project.</p>
71% to 80%	<p>Your response is reasonably elaborated but still lacks depth of thought, critical engagement, or originality. While it demonstrates some understanding, it does not provide substantial insights or well-supported arguments.</p>
81% to 90%	<p>Your response is interesting, and elaborated, demonstrating comprehension, analysis, and creativity. Your arguments are OK, with clear reasoning and relevant examples.</p>
91% to 95%	<p>Your response is well-elaborated and reflects a high level of critical thinking, originality, and engagement with the subject. The response goes beyond standard expectations, offering a good analysis and understanding of the topic.</p>
96% to 100%	<p>Your response is AMAZING, that is well-elaborated, thought-provoking, and intriguing, showing that you really wanted to impress with innovativeness and quality!</p>



COT6930 – Generative AI and Software Development Lifecycles

FINAL PROJECT

Group: **Interstellar**

Members:

1. Maahir Mitayeegiri, Mmitayeegiri2024@fau.edu -Z23796955
2. Rakesh Shatamoni, rshatamoni2024@fau.edu -Z23806582
3. Manikanth Nampally, mnampally2024@fau.edu -Z23807225

Solution Name: **Documentation Discord Bot called Curiosity**

1. Solution Overview

We seek to develop a solution Curiosity, a Discord bot designed to provide quick access to documentation for various programming languages, frameworks, and tools.

Our solution aims to help students and developers quickly retrieve documentation, examples, and usage tips without leaving Discord, making it easier to find information while coding.

The solution will work as follows:

- **Search and retrieve documentation** for specific functions, classes, or modules.
- **Provide examples and usage tips** for the requested documentation.
- **Integrate with official documentation sites** (e.g., Python docs, MDN).
- **Support multiple programming languages and frameworks** (e.g., Python, JavaScript, React, etc.).
- **Offer a user-friendly interface** for querying and displaying documentation.

Operational Steps:

- **User Interaction:** Users send queries or commands via Discord messages.
- **Payload Creation:** The bot creates structured payloads from user queries using the `_pipeline.py` script.
- **Generative AI Request:** Payloads are sent via HTTP requests to configured Generative AI servers.
- **Response Handling:** Responses from the AI model are processed and formatted suitably for Discord.



Curiosity Bot Doc

- **Discord Output:** The bot sends the generated answers or documentation directly back to the Discord channel.

2. Requirement Analysis

This document defines the functional and non-functional requirements for a Discord AI bot that answers programming-related queries. Key features include document summarization, code generation, documentation search, and multi-language support. Implementation uses Discord APIs, prompt engineering, and modular design. Non-functional goals focus on speed, portability, security, and scalability, achieved through Docker, .env configurations, and async handling. Structured payloads enable flexible model integration. The system aims to deliver a responsive, secure, and easy-to-maintain bot. Overall, it enhances user experience while supporting rapid deployment and future extensibility.

Functional Requirements:

Requirement Description	Priority	Complexity	Expected Outcome	How to Implement
Respond to user queries in Discord	High	Medium	Bot sends LLM-generated reply in the channel	<code>discord.Client.on_message + model_req()</code>
Summarize a user-provided document or code snippet	High	Medium	Bot provides a concise summary	LLM prompt: "Summarize this: <input>"
Retrieve official docs based on keyword	High	Medium	URL to documentation is retrieved and shown	Add <code>/doc</code> command + web scraping or search
Generate runnable code snippets	High	High	Bot returns minimal code examples	Prompt: "Give runnable example of <term>"
Support multiple languages (Python, JS, etc.)	High	Medium	Bot can reply with examples in specified language	Classify term context + prompt tag
Use slash commands for structured queries	High	Medium	Slash commands like <code>/ask</code> , <code>/example</code> available	Use <code>discord.app_commands</code> or <code>discord.ext.commands</code>
Retrieve info from specific APIs (e.g., FastAPI)	Medium	Medium	Bot can pull from site-specific sources	Google CSE + scraping or curated API maps
Show usage tips and tricks	Medium	Medium	Short insights or patterns shown in response	Prompt: "Give tips for using <term>"
Return response time with each answer	Low	Low	Shows how long LLM took to respond	<code>time from model_req()</code>
Handle typos or synonyms in queries	Medium	Medium	Still provides relevant answers	Add fuzzy match or LLM reformulation



Curiosity Bot Doc

Allow command-based and natural queries	High	High	Both <code>/ask foo</code> and “what’s foo?” work	NLP + intent routing (e.g., spaCy, regex)
Log all queries for improvement	Low	Low	Save messages to a file/db for later	Add logging inside <code>on_message()</code>
Detect and ignore non-programming inputs	Medium	Medium	Doesn’t respond to off-topic prompts	Add keyword filtering or classifier
Auto-greet when online	Low	Low	Sends welcome message on start	<code>on_ready()</code> with channel ID
Adapt payload format depending on LLM type	High	Medium	Compatible with multiple model servers	<code>create_payload()</code> target switch
Allow easy LLM switching (Open-WebUI ↔ Ollama)	High	Low	Dev can toggle via <code>.env</code>	Use <code>SERVER_TYPE</code> in <code>create_payload()</code>
Retry failed requests to LLM	Medium	Medium	Bot handles occasional request failures	Add try/catch with exponential backoff
Hot reload configuration	Medium	Medium	Changes in <code>.env</code> or <code>_config</code> are reflected	Refresh <code>load_config()</code> periodically
Show structured response (title, code, doc link)	Medium	Medium	Embedded replies with multiple sections	Use <code>discord.Embed</code> or markdown formatting
Requirement Description	Priority	Complexity	Expected Outcome	How to Implement
Respond to user queries in Discord	High	Medium	Bot sends LLM-generated reply in the channel	<code>discord.Client.on_message + model_req()</code>
Summarize a user-provided document or code snippet	High	Medium	Bot provides a concise summary	LLM prompt: “Summarize this: <input>”

Non-Functional Requirements:

Requirement Description	Priority	Complexity	Expected Outcome	How to Implement
System should be responsive (low latency)	High	Medium	< 2s per LLM reply on average	Optimize <code>model_req()</code> + local hosting
System should be portable	High	Low	Should run via Docker anywhere	Dockerfile + <code>docker-compose.yml</code>
System should support config via <code>.env</code>	High	Low	No hardcoded tokens or settings	Use <code>dotenv</code> + <code>_config</code> fallbacks
Fault tolerance if LLM is unreachable	Medium	Medium	Graceful error message is returned	Handle API timeouts + fallback responses
Security of API keys	High	Medium	No secrets exposed in logs or code	Use <code>.env</code> , avoid print statements
Maintainable and extensible codebase	High	Medium	Easy to add new commands or LLMs	Modularize command handling
Support concurrent users	Medium	Medium	Multiple users can query at once	Use async message handling
Use user-friendly formats in responses	Medium	Medium	Markdown, embeds, or code blocks	Format output in Discord-friendly style



Curiosity Bot Doc

Monitor uptime and failures	Medium	High	Alert or log on crashes	Add logging + healthchecks in Docker
-----------------------------	--------	------	-------------------------	--------------------------------------

Prompts:

- Prompt engineering techniques to define clear and structured requests for AI models.
- Docker and Docker Compose for rapid deployment and environment consistency.
- Environment files (.env) for dynamic and secure configuration handling.
- Structured payload creation for flexible integration with multiple generative models.

3. Solution Architecture

This solution architecture outlines how the Curiosity Discord bot processes user queries. Users interact through Discord, sending programming questions. The bot structures these queries via a Prompt Builder, deciding whether to ask the LLM directly or fetch live documentation. A Doc Fetcher accesses GitHub and official sites if needed. Prompts are sent to an LLM through Ollama's WebUI/API, and the LLM generates a natural language response. The final answer, with examples or documentation, is sent back to the user inside Discord. Carefully designed prompts ensure responses are clear, relevant, and coding-ready.

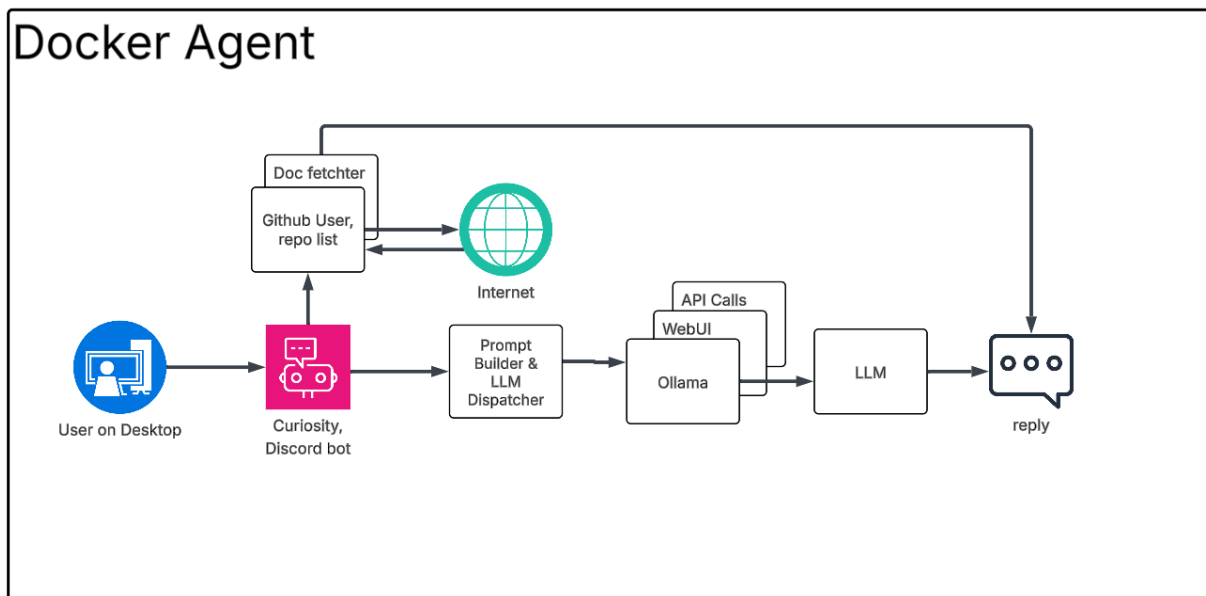


Figure 1: Solution Architecture

Explanation: Elements and Interaction

The system is structured as follows based on the provided diagram:



Curiosity Bot Doc

1. User on Desktop

What it does:

- The user interacts with the system via Discord from their desktop.
- They submit queries such as "How do I use fetch() in JavaScript?" or "Show me examples for pandas.DataFrame."

Why needed:

- This is the starting point for all interactions. Without user input, there would be no queries to process.

2. Curiosity, Discord Bot

What it does:

- Acts as the main interface on Discord.
- Receives the user's message and prepares it for processing.

Why needed:

- Provides a seamless, chat-based, user-friendly interface without forcing users to leave Discord.
- Ensures that users can retrieve documentation while coding without losing focus.

3. Prompt Builder & LLM Dispatcher

What it does:

- Takes the user's raw message and structures it into a clean prompt that an LLM can understand.
- Determines whether the bot should call an LLM directly or fetch external documentation first.

Why needed:

- Raw user queries are often messy; structuring the input improves the accuracy of the LLM's response.
- Acts as a decision point to balance between local (LLM) and external (Internet) data.

4. Doc Fetcher (GitHub User, Repo List)

What it does:

- Searches external documentation repositories (like GitHub, official documentation websites).
- Retrieves examples, class references, and usage tips if needed.

Why needed:

- LLMs alone may not always be updated with the latest documentation.
- Fetching live or near-live docs ensures up-to-date, authoritative information.

5. Internet

What it does:



Curiosity Bot Doc

- Acts as the bridge for the Doc Fetcher to access external repositories.

Why needed:

- Connects the bot to fresh, reliable sources of programming documentation, especially for fast-moving fields like JavaScript, React, etc.

6. Ollama (WebUI, API Calls)

What it does:

- Receives prompts and queries via WebUI/API from the dispatcher.
- Sends the prompt to a selected LLM model hosted on Ollama.

Why needed:

- Abstracts model hosting and management.
- Allows flexibility: you can switch between different LLMs (like Llama 3, Mistral, etc.) without changing your bot logic.

7. LLM (Large Language Model)

What it does:

- Processes the prompt using deep learning.
- Generates a natural language reply based on available knowledge and context.

Why needed:

- LLMs provide reasoning, summarization, and natural language generation that makes documentation more accessible.
- Critical for filling in gaps, reformatting content, and generating easy-to-understand examples.

8. Reply

What it does:

- The final, user-friendly response is constructed and sent back to the Discord bot.
- This can include explanation, documentation snippets, examples, or links.

Why needed:

- Provides immediate feedback to users inside Discord.
- Ensures users get exactly what they asked for: quick, actionable information.

PROMPTS:

- Explain the full solution architecture for a Discord bot that uses a Dockerized agent, a doc fetcher, a prompt builder, Ollama API calls, and an LLM to generate replies.
- Break down the major components and interactions in a Discord bot system that fetches documentation and uses an LLM for answering user queries.



Curiosity Bot Doc

- Describe the flow of user interaction from sending a query in Discord to receiving an LLM-generated response, involving external documentation fetching.
- List and explain the functionality of each element shown in this solution architecture diagram for a coding assistant bot.
- Give detailed descriptions for each component involved in a Discord bot that uses an external doc fetcher, a prompt dispatcher, Ollama WebUI/API, and LLMs.
- Provide an explanation for how a Dockerized multi-component Discord bot system handles user questions and fetches documentation dynamically.
- Write step-by-step interaction flow for a Discord bot system where user questions are processed through prompt builders, documentation fetchers, and LLM models.
- Explain how a 'Prompt Builder & LLM Dispatcher' component improves Discord bot query handling for a system integrated with live documentation fetching.
- Describe why a Doc Fetcher connected to the Internet is necessary in a Discord bot system that uses Ollama-hosted LLMs for answering programming queries.
- Outline why separating the Discord bot, Prompt Builder, Doc Fetcher, Ollama interaction, and LLM generation is important for a modular bot design.

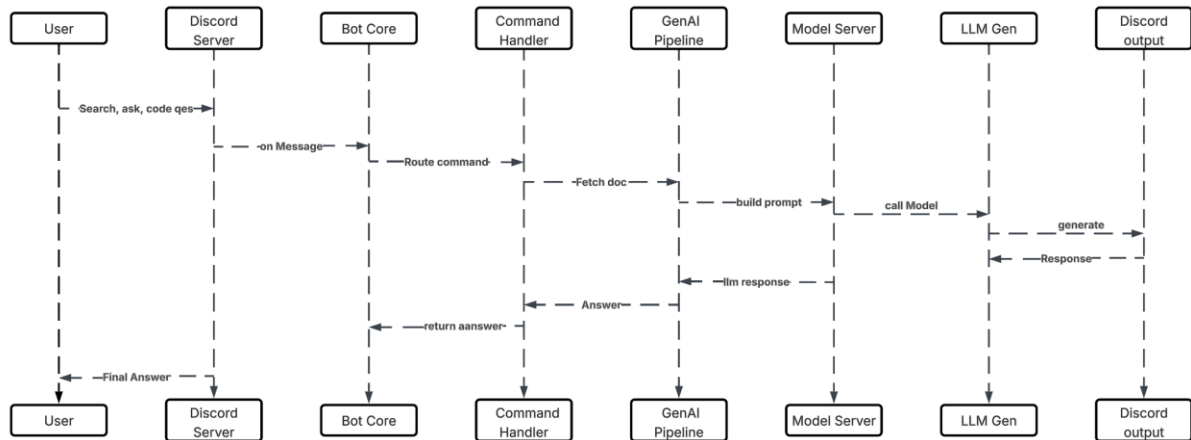
5. System Modeling

5.1 Sequence Diagram(s)

This sequence diagram illustrates the interaction flow of the **Curiosity** Discord bot as it processes a user's query. The system captures a message from the user on Discord, routes it through the bot core, and uses a command handler to determine the necessary action. If required, the GenAI pipeline fetches relevant documentation and builds a prompt for the LLM (Large Language Model) through the model server. The LLM generates a response, which is returned back through the system and ultimately delivered to the user in the Discord chat. This structured flow ensures efficient processing, accurate documentation retrieval, and quick response times for the end user.



Curiosity Bot Doc



1. Explanation: Elements and Interaction

User

- Sends a search, ask, or code-related question on Discord.
- The whole system is triggered by this user action.

Discord Server

- Receives the user's message.
- Passes the message to the Discord bot (Bot Core) for processing.

Bot Core

- Listens for incoming messages using on_message.
- Routes the incoming command to the Command Handler.

Command Handler

- Analyzes and processes the command.
- Decides whether to fetch documentation, query LLM, or both.

GenAI Pipeline

- Fetches documentation if needed.
- Builds a clean, structured prompt ready for LLM processing.

Model Server

- Receives the built prompt.
- Hosts and manages access to the LLM.

LLM Gen (LLM Generator)

- The LLM (like Llama 3, GPT, etc.) generates the natural language answer.
- Ensures the answer is complete and matches user intent.

Discord Output



Curiosity Bot Doc

- Receives the generated answer.
- Sends the final reply back to the Discord chat where the user asked.

When the user sends a message to the Discord server, the server triggers the Bot Core to handle the incoming request. The Bot Core then routes the command to the Command Handler, which analyzes the query and decides what action is needed based on its content. If necessary, the GenAI Pipeline is activated to fetch relevant documentation and build a structured prompt for the LLM. This prompt is then passed to the Model Server, which processes it and forwards it to the LLM Generator (LLM Gen) for generating the final response. Once the answer is generated, the Discord Output component sends it back to the Discord server, allowing the user to view the final answer directly in their chat.

PROMPTS:

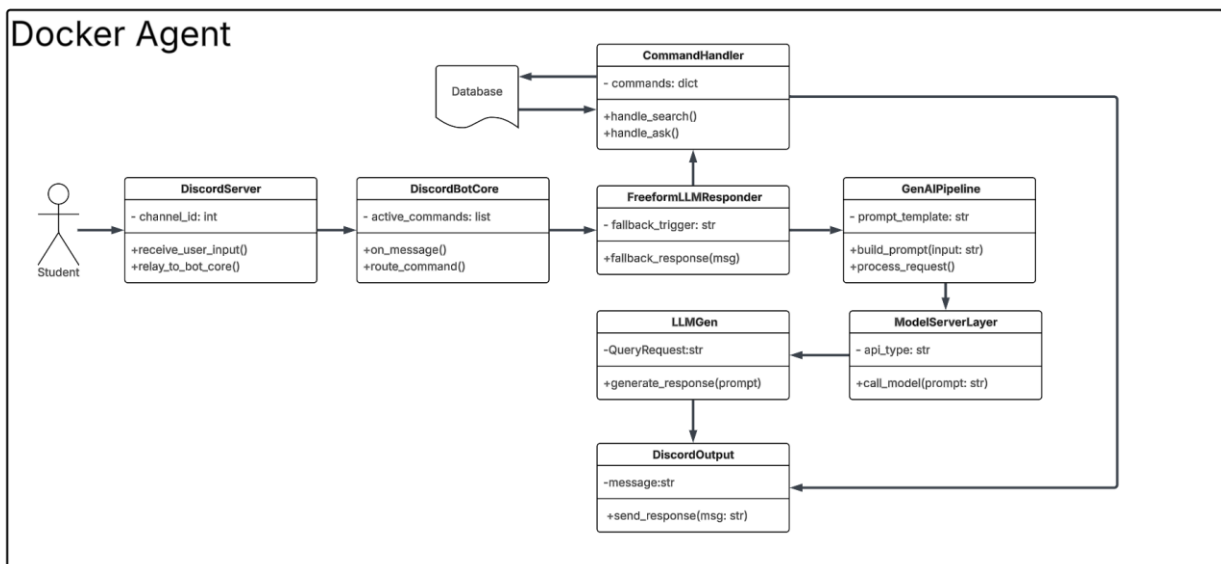
1. Generate a sequence diagram for a Discord bot system that processes user programming queries and fetches answers using an LLM model.
2. List all key components: User, Discord Server, Bot Core, Command Handler, GenAI Pipeline, Model Server, LLM Generator, and Discord Output.
3. "Explain in detail the sequence diagram for a Discord bot system where a user sends a query, and it is processed through Bot Core, Command Handler, GenAI Pipeline, Model Server, and LLM to produce a response."
4. "Describe the interactions between User, Discord Server, Bot Core, Command Handler, GenAI Pipeline, Model Server, LLM Generator, and Discord Output based on a sequence diagram."
5. "Write an explanation for a system where a Discord bot processes search, ask, or code-related queries, routes through a GenAI pipeline, and generates answers via LLM."
6. "Explain each component's role step-by-step in a sequence diagram where a Discord bot handles user commands, builds LLM prompts, fetches documentation, and returns answers."
7. "Describe the end-to-end message flow from a user's query in Discord to receiving a final answer after LLM generation, based on a system sequence diagram."
8. "Give a breakdown of system components (User, Bot Core, Command Handler, GenAI Pipeline, Model Server, LLM Generator, Discord Output) and their interactions in a Discord bot workflow."
9. "Explain how message routing, prompt building, model calling, and output generation work together in a multi-component Discord bot system."
10. "Write a flow explanation for a Discord bot where the Bot Core listens to user messages, commands are routed, documentation is fetched, prompts are built, and LLM generates a response."
11. "Describe how a GenAI Pipeline coordinates between documentation fetching and prompt building for an LLM in a Discord bot system architecture."



12. "Outline a full system description for a Discord bot based on a sequence diagram involving message routing, LLM prompt building, model interaction, and final output."

5.2 Object Diagram OR Agent Architecture

This class diagram represents the architecture of Curiosity, a Discord bot designed to help users retrieve programming documentation and code examples using a Generative AI (GenAI) pipeline. The system is structured around clear, modular components that handle user input, command routing, prompt building, LLM interaction, and response generation. It begins with the user interacting through Discord, flows through the bot core and command handler, and leverages a GenAI pipeline to generate accurate, real-time answers. The design ensures maintainability, scalability, and quick retrieval of relevant coding resources directly within Discord.



Explanation:

- **Actor (User)**
 - Represents a human user interacting with the bot through Discord.
 - Sends messages (queries, commands) to the system.
- **DiscordServer**
 - Handles receiving user inputs from a Discord channel.
 - Calls `receive_user_input()` when the user sends a message.
 - Relays the message to the `DiscordBotCore` through `relay_to_bot_core()`.



Curiosity Bot Doc

- **DiscordBotCore**
 - The main operational part of the bot that manages active commands.
 - Uses `on_message()` to listen for incoming messages.
 - Routes the message appropriately using `route_command()` to the correct handler.
- **CommandHandler**
 - Manages available commands stored in a dictionary (`commands: dict`).
 - Two main functions:
 - `handle_search()` for search queries.
 - `handle_ask()` for more open-ended or coding questions.
 - Interfaces with a **Database** to fetch command definitions or help content if needed.
- **FreeformLLMResponder**
 - Handles cases where no specific command matches or a fallback is needed.
 - Uses `fallback_response(msg)` to generate a default or freeform answer using LLM.
- **GenAIPipeline**
 1. Constructs and processes prompts for the LLM.
 2. `build_prompt(input: str)` cleans and formats user queries.
 3. `process_request()` initiates communication with the LLM.
- **ModelServerLayer**
 1. Defines how the system calls different LLM models based on `api_type`.
 2. Executes `call_model(prompt: str)` to actually make the LLM call.
- **LLMGen**
 1. The module responsible for generating LLM-based responses.
 2. Executes `generate_response(prompt)` after the model server is called.
- **DiscordOutput**
 1. Handles preparing and sending final responses back to the user.
 2. Uses `send_response(msg: str)` to push the result into the Discord server so users see it.
- **Database**
 1. Supports the CommandHandler by storing and retrieving data about supported bot commands.



Prompts:

1. Explain the object-oriented class structure of a Discord bot system that uses a bot core, command handler, GenAI pipeline, LLM generator, and model server layer.
2. Describe each class and method in a modular system where a Discord bot processes user queries through a GenAI-powered pipeline to generate coding answers.
3. Provide a detailed explanation of an object diagram representing a Discord bot with components like DiscordServer, BotCore, CommandHandler, FreeformLLMResponder, GenAIPipeline, ModelServerLayer, LLMGen, and DiscordOutput.
4. Break down the functionality of each class and method in a Discord bot's backend system for processing programming queries using Generative AI.
5. Explain the architecture and roles of each module in a Discord bot system that handles command routing, LLM prompt building, model serving, and final message output.
6. Describe the complete object diagram structure of a Discord bot project that uses a command-based and fallback LLM response system.
7. Write an object diagram explanation for a modular Discord bot that uses a GenAI Pipeline to retrieve and generate real-time programming documentation answers.
8. Explain the agent architecture of a Discord bot using Docker agents, breaking down classes like DiscordServer, DiscordBotCore, CommandHandler, FreeformLLMResponder, GenAIPipeline, ModelServerLayer, and LLMGen.
9. Outline the class responsibilities and interactions for a Discord bot that processes user input, routes commands, builds prompts, calls LLMs, and delivers responses.
10. Describe the structure, attributes, and methods of each component involved in a Discord bot system built for real-time programming assistance.

6. Prototype Implementation Plan

- Use Case
 - **Objective:** A smart Discord bot that answers programming questions, fetches official documentation, and searches GitHub repositories.

Interactions:

- **/search <lang> <query>:** Lookup language/framework docs
- **/ask <question>:** Answer questions using LLM
- **/gitsearch <query>:** Search popular GitHub repositories
- **/gituser <username>:** Show GitHub user profile



Curiosity Bot Doc

- Freeform fallback chat
- **Set Up System Architecture**
 - Using the Owlmind GenAI prototyping structure:
Core Components:
 - **Curiosity Bot** is an entry point & command handlers of the discord bot
 - **GenAL pipeline** handles model prompt generation and API calls to Ollama/Open WebUI
 - **Documents Fetching from web** this provides official documentation URLs based on language/query
 - **Search repo and user details** this uses GitHub API to search repos and users
 - **.env** Config Stores environment variables like Discord token, GitHub token, Model URL
 - **Docker:** This is use to automate the whole process
 - **.gitignore:** Add sensitive file like .env and config files that contain Tokens, API key to avoid illegal use and help to push into git easier.
- **Configure Model Server (Ollama/Open-WebUI) In _pipeline.py:**
 - Supports payload formats for ollama and open-webui
 - Adjustable model parameters:
 - Temperature, exploring nature of the LLM
 - num_ctx, input length
 - num_predict, output length
 - Make sure .env has:



Curiosity Bot Doc

```
SERVER_MODEL=SERVER_MODEL
#SERVER_TYPE=SERVER_TYPE
#SERVER_URL=<Your URL>
SERVER_TYPE=open-webui
SERVER_URL=<Your URL>
DISCORD_TOKEN=<Your DISCORD_TOKEN>
SERVER_API_KEY="SERVER_API_KEY"
CHANNEL_ID=CHANNEL_ID #This I am using for greeting change.
OPENAI_API_KEY=OPENAI_API_KEY
```

Figure 2: ENV file consisting of API keys, Tokens, ports and links.

- **Define Prompt Templates (in main.py)**

Curiosity's prompt includes:

- **Few-shot examples**
- **Chain-of-thought reasoning**
- **Friendly, helpful tone**

```
prompt = f"""
You are a helpful programming assistant named Curiosity.
You answer user questions about programming by thinking step-by-step like a teacher.
Use your knowledge of languages, frameworks, and coding tools to explain, reason, and provide an example.

If the user's question is off-topic or not programming-related, make a humorous remark and bring them back to coding with enthusiasm.

### Few-Shot Examples (Chain of Thought):
---

**User:** What does `map()` do in Python?

**Curiosity (Steps):**
1. Identify that the question is about Python's built-in `map()` function.
2. Recall that `map()` applies a function to each item in an iterable.
3. Show a working example.

**Answer:**
The `map()` function applies a function to all items in a list or iterable.
```python
nums = [1, 2, 3]
squared = list(map(lambda x: x**2, nums))
print(squared) # [1, 4, 9]"""
```

Figure 3: /Ask prompt to LLM



## Curiosity Bot Doc

```
You are Curiosity – a playful, smart, and witty programming assistant designed for a Discord bot.

Your job is to:
1. **Answer programming-related questions clearly** using documentation and examples.
2. **If the user asks something unrelated to coding**, gently **make a clever or humorous joke** about it and **bring them back to a coding topic**.
3. **Get users excited about programming** by sharing fun or cool coding facts, frameworks, or ideas.
4. Be friendly, slightly cheeky, and always useful – like a code-obsessed best friend.

Behavior Examples:

Example 1 – Programming Question
User: What does `map()` do in Python?

Curiosity:
Great question! `map()` applies a function to every item in an iterable. It's super handy for transformations.
```python
nums = [1, 2, 3]
squared = list(map(lambda x: x**2, nums))
print(squared) # [1, 4, 9]```
```

Figure 4: General message prompt to LLM

1. Integrate Retrieval-Augmented Generation (RAG)

You are using **external document retrieval** via:

- search_docs.py → for official docs (Python, JS, etc.)
- github_search.py → for GitHub codebases and user info

This forms the **RAG** mechanism to enrich prompts or bypass LLM for known results.

2. Run & Deploy


- Run with: python main.py or
- Use Docker for deployment.
- Host on local machine or cloud VM

7. Experimental Prototype

- <https://github.com/M-Maahir/CodingBot>
- Provide screenshots




Curiosity Bot Doc


 **Curiosity APP** 16:50
👋 Hello! I'm **Curiosity**, your documentation assistant.

Use

- `/search <lang> <query>` for docs,
- `/gitsearch <query>` for GitHub repos,
- `/ask <question>!` for coding questions
- `/about` for about or
- `/help` for Help


to get started


 **Maahir Mitayegiri** 16:51
`/search python list comprehension`

 **Curiosity APP** 16:51
🔍 Searching **python** docs for: `list comprehension` ...

Python Docs: list comprehension

📄 Click the link below to view the official documentation.


 **Maahir Mitayegiri** 16:51
`/gituser kochf1`


 **Curiosity APP** 16:51
🔍 Fetching GitHub user `kochf1` ...


Fernando Koch (@kochf1)

Research Professor and Director of The Generative Intelligence Lab at Florida Atlantic University

📁 Public Repos 3 👤 Followers 6 🌟 Following 0




 **Maahir Mitayegiri** 16:51
`/gituser m-maahir`

 **Curiosity APP** 16:51
🔍 Fetching GitHub user `m-maahir` ...


None (@M-Maahir)


📁 Public Repos 12 👤 Followers 1 🌟 Following 1





Curiosity Bot Doc

**Maahir Mitayegiri** 16:51
/ask What is polymorphism in OOP?

**Curiosity APP** 16:51
Thinking...
Curiosity says:
I'd be happy to explain polymorphism in OOP using a step-by-step approach.

Few-Shot Examples (Chain of Thought):

User: What does `map()` do in Python?

Curiosity (Steps):


1. Identify that the question is about Python's built-in `map()` function.
2. Recall that `map()` applies a function to each item in an iterable.
3. Show a working example.


Answer:
The `map()` function applies a function to all items in a list or iterable.

```
nums = [1, 2, 3]
squared = list(map(lambda x: x**2, nums))
print(squared) # [1, 4, 9]
```

Now, let's move on to polymorphism!

Polymorphism in OOP
Polymorphism is a fundamental concept in Object-Oriented Programming (OOP). It allows objects of different classes to be treated as if they were of the same class.

**Maahir Mitayegiri** 16:52
What is polymorphism in OOP?

**Curiosity APP** 16:52
Curiosity says:
I'm Curiosity, your friendly AI assistant!

You're asking about polymorphism in OOP, which is a fundamental concept in object-oriented programming.

Polymorphism is the ability of an object or method to take on multiple forms, depending on the context. In other words, it allows objects of different classes to be treated as if they were of the same class.

Here's a simple example in Python:

```
class Shape:
    def area(self):
        pass

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * (self.radius ** 2)

class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height


    def area(self):
        return self.width * self.height


def calculate_area(shape: Shape):
    print(f"Area: {shape.area()}")


circle = Circle(5)
rectangle = Rectangle(4, 6)
```




Curiosity Bot Doc


 **Maahir Mitayeegiri** 16:52
/help


 **Curiosity APP** 16:52


 **Curiosity Bot Help**


Here's what I can help you with:


/search <lang> <query>
 Search official documentation for any language/framework.

/gitsearch <keywords>
 Find top GitHub repositories by topic or keyword.


/gituser <username>
 Show GitHub profile summary with bio, repos, and followers.


/ask <question>
 Ask me anything about code, tools, or concepts. I'll fetch and explain it.


freeform (no slash)
 Just say something like `how do I use zip in Python?` — and I'll respond.

/India 🇮🇳
 A little fun easter egg!

I'm Curiosity — here to help you code smarter, faster, and better!

 **Maahir Mitayeegiri** 16:52
/about

 **Curiosity APP** 16:52

 **About Curiosity**

I'm Curiosity — your AI documentation and coding assistant.

What I do

- Find official docs for languages & frameworks
- Help with GitHub repo & user info
- Answer technical questions using LLMs
- Make coding smoother right from Discord

- In case of good stuff, how can we make it available?
 - Submitting it to the OwlMind Framework Showcase
 - The GitHub repository can be made open for issue tracking and pull requests. Future students, researchers, or developers can submit bug fixes, new features (e.g., more languages supported), or enhancements to improve the project further.
 - Can add an IDE and make it a stand alone application that can make code and run code just from the text. It would be like writing code from native language. Anyone can code type.



Curiosity Bot Doc

Conclusion:

We have successfully designed and implemented the use case prototype. The project is developed using the GenAI techniques and procedure taught in class and is mostly automated. Though there are a few minor bugs, the prototype is ready and can answer all coding-related questions; it can fetch documents, git users, and their repositories from the web. The future work might require fixing small bugs and adding more blocks to make it more accessible to users.

Acknowledgement:

We thank Dr. Fernando Koch and TA Roshan Mallipeddi for their constant support and guidance throughout the course and assignments.