# **SWE 3313**

**Software Requirements Design Document**

**Project Name: ARIA Active Responsive Integrated AI**

Team #4

Jordyn Jones, Lewis Dante, Will Martin, Jackson Nuckles, Joshua Onuoha, Phillip Pham, Miguel Ramirez

# **Team Members:**

A: All team members contributed to the document.

TABLE OF CONTENTS

1. Introduction …………………………………………………………………………………. 3

1.1 Purpose

1.2 Scope

1.3 Overview

1.4 Reference Material

1.5 Definitions and Acronyms

2. SYSTEM OVERVIEW ……………………………………………………………………. 4

3. System Architecture ………………………………………..…………………………….. 4

3.1 Architectural design

3.2 Technical Design

3.3 Design Rationale

4. Detailed Design …………………………………………………………………………….. 9

4.1 Class Diagrams

4.2 Other Diagrams

5. Data (Database) Design ……….…………………………………………………………. 11

6. Human Interface Design …………………………………………………………………. 12

6.1 UI design

6.2 Graphical UI design

7. Requirements Matrix …………………………………………………………………… 14

1. INTRODUCTION

1.1 Purpose

This Software Requirements Design Document describes the architecture and system design of ARIA (Active Responsive Integrated AI). It is intended for the project development team, providing an overview of the system's design.

1.2

Scope

The scope of this project is to develop ARIA, an Instagram bot that reads direct messages (DMs) from Instagram and responds in real-time using the ChatGPT API. ARIA allows users to send direct messages, generate text responses, create Instagram posts, read direct messages, and manage user requests. The document outlines the high-level design of ARIA.

1.3

Overview

Provide an overview of this document and its organization.

1.4

Reference Material

This section is optional.

List any documents, if any, which were used as sources of information for the test plan.

1.5

Definitions and Acronyms

This section is optional.

Provide definitions of all terms, acronyms, and abbreviations that might exist to properly interpret the SDD. These definitions should be items used in the SDD that are most likely not known to the audience.

2. SYSTEM OVERVIEW

Give a general description of the environment, functionality, context and design of your project.

Provide any background information if necessary.

3. SYSTEM ARCHITECTURE

3.1

Architectural Design

High-Level System Architecture Overview:

Our Discord-Instagram bot, named ARIA (Active Responsive Integrated AI), is designed to facilitate communication between Discord, Instagram, and external AI services. It allows users to interact with it through Discord by sending messages and images, and it generates responses using the ChatGPT API. Additionally, users can create and publish Instagram posts through the bot.

Major Components and Subsystems:

Discord Interface: This subsystem is responsible for handling interactions with Discord. It listens for messages from Discord users and routes them to the appropriate processing components. It also sends responses generated by the bot back to Discord.

Instagram Interface: This component manages interactions with Instagram. It is responsible for reading direct messages, creating Instagram posts, and performing other actions on Instagram.

ChatGPT Integration: This subsystem integrates with the ChatGPT API to generate text responses based on user messages. It processes incoming messages, sends them to the ChatGPT API, and then sends the generated responses back to the user.

Image Handling: This component manages the receipt and processing of images sent by users. It checks the received images, and if valid, they are sent for AI processing.

AI Processing: This subsystem is responsible for processing user-provided text prompts and images. It communicates with both the ChatGPT API and external image processing services.

Database: If required, a database stores user-specific data like user settings, logs, and Instagram credentials.

Collaboration Between Subsystems:

Users send messages and images to the Discord Interface.

The Discord Interface routes these messages to the appropriate processing components.

The ChatGPT Integration processes user messages by forwarding them to the ChatGPT API.

The AI Processing component manages the processing of both text prompts and images and communicates with external AI services.

The Instagram Interface handles interactions with Instagram, including reading direct messages and posting content.

3.2

Technical design

High-Level Technologies Used:

Discord API: The Discord bot interacts with the Discord API to send and receive messages in Discord servers and channels.

Instagram API: For Instagram interactions, the system uses the Instagram Graph API to read direct messages, create posts, and perform other actions on Instagram.

ChatGPT API: The ChatGPT API, provided by OpenAI, is utilized for generating text responses based on user messages. It processes incoming messages and sends them to the ChatGPT API to receive generated responses.

External Image Processing Services: The system might use external image processing services or libraries to validate, manipulate, or analyze images sent by users.

Database System (Optional): If necessary, a database management system (DBMS) like MySQL or PostgreSQL is used to store user-specific data such as settings, logs, and Instagram credentials.

Programming Languages: The bot is implemented using Python for its flexibility and extensive libraries for working with APIs, handling images, and AI integration.

Web Frameworks (Optional): Web frameworks like Flask or Django can be used for creating web services to manage the communication between different parts of the system.

Hosting Services (Optional): If continuous bot operation is required, cloud hosting services like AWS, Heroku, or others can be used to host the bot 24/7.

Technical Design Overview:

Discord API Integration: The Discord API integration is achieved through a Python library called discord.py. The bot connects to Discord servers, listens for messages, and sends responses to specific channels.

Instagram API Integration: The Instagram Graph API is used to manage interactions with Instagram. The bot communicates with Instagram servers using HTTP requests. It sends messages, posts content, and retrieves user data.

ChatGPT Integration: The ChatGPT API integration is done via HTTP requests. The bot sends user messages to the ChatGPT API and receives responses in real-time. The response is then sent back to the user on Discord.

External Image Processing: For image processing, the system might utilize image libraries like Pillow or external image recognition APIs. Images sent by users are validated, and AI processing may be applied if needed.

Database (Optional): If a database is part of the system, it's integrated using appropriate database connectors or Object-Relational Mapping (ORM) libraries. This allows the bot to store and retrieve user-specific data efficiently.

Programming Logic: Python code manages the logic that glues all components together. It routes messages between Discord, Instagram, and AI services, manages user interactions, and orchestrates responses.

Web Frameworks (Optional): If web services are used, Flask or Django is employed to create APIs that enable communication between different parts of the system. These frameworks manage routes, views, and responses.

3.3

Design Rationale

Our architecture for the Discord-Instagram bot was carefully selected based on a combination of factors, including functionality, flexibility, and ease of development. Here's the rationale behind the architectural choices:

1. Discord Bot Architecture:

Choice: We opted for a modular architecture with a Discord bot at its core.

Rationale: Discord is a widely used platform for communication, and its API (via the discord.py library) allows for easy integration. By building our system around a Discord bot, we can tap into a large user base and provide seamless interactions.

2. Instagram API Integration:

Choice: We integrated Instagram API for Instagram interactions.

Rationale: Instagram is a popular social media platform, and the Instagram Graph API provides the necessary functionality to interact with Instagram. This choice allows users to send messages to Instagram and create Instagram posts directly from Discord.

3. ChatGPT API Integration:

Choice: We integrated the ChatGPT API from OpenAI for text generation.

Rationale: The ChatGPT API is state-of-the-art in natural language understanding and generation. It allows the bot to provide contextually relevant and human-like responses to user messages. This choice ensures that the bot is engaging and user-friendly.

4. Flexibility and Extendability:

Choice: We designed the system to be modular, allowing for easy addition of new features or integrations.

Rationale: To adapt to changing user needs and technological advancements, we opted for an architecture that is easy to extend. Additional features, such as image recognition or language translation, can be added without major disruptions.

5. Trade-offs:

Integration Complexity: While integrating multiple APIs, we considered the trade-off between functionality and complexity. By adding Instagram, Discord, and ChatGPT, we introduce potential points of failure. However, these integrations are essential for providing the desired features.

Bot Reliability: Running a bot continuously poses reliability challenges, as it must be online 24/7 to serve users. Due to hardware limitations, we might face occasional downtimes, which could affect availability.

6. Alternative Architectures:

Monolithic Architecture: We considered a monolithic architecture where all functionalities would be within a single codebase. However, this would result in a complex and less maintainable codebase, which goes against our aim for a modular and maintainable system.

Serverless Architecture: We explored a serverless architecture for cost-efficiency. However, serverless has limitations in terms of connection times and resource constraints, which may not be suitable for real-time interactions.

7. Scalability Considerations:

While our architecture is designed for moderate-scale interactions, we acknowledged that for extreme scalability, especially in the case of Instagram and ChatGPT, we would need a more complex distribution of processing, including load balancing, caching, and microservices. Ok

4. DETAILED DESIGN

In this section, provide details (with respect to 3.1) at what each component does in a more systematic way. In other words, describe the low-level design for each of the system components identified in the previous section (3.1). For each component, you should provide details in the following UML diagrams to show its internal structure.

4.1

Class diagrams

| **Discord-Instagram Bot** |
| --- |
| * DiscordClient: Client |
| * InstagramClient: Client |
| * ChatGPTClient: Client |
| * initialize() |
| * start() |
| * on\_message\_received() |
| * on\_instagram\_interaction() |
| * on\_discord\_interaction() |
| * on\_chatgpt\_response |
| * handle\_user\_requests() |

Discord-Instagram Bot: This is the core class representing the Discord-Instagram bot. It has three client instances, one for Discord, one for Instagram, and one for ChatGPT. These clients handle communication with their respective services.

DiscordClient: Manages interactions with Discord, such as sending and receiving messages in Discord text channels.

InstagramClient: Handles interactions with Instagram, including sending direct messages and creating Instagram posts.

ChatGPTClient: Manages communication with the ChatGPT API for generating text responses to user messages.

initialize(): Initializes the bot and its client instances.

start(): Initiates the bot's operations and connects to Discord, Instagram, and ChatGPT.

on\_message\_received(): Handles messages received from Discord users and initiates appropriate actions.

on\_instagram\_interaction(): Manages interactions with Instagram, including sending and receiving messages.

on\_discord\_interaction(): Handles interactions and responses in Discord.

on\_chatgpt\_response(): Processes responses generated by the ChatGPT API.

handle\_user\_requests(): Manages and responds to user requests and interactions.

4.2

Other diagrams

Detailed diagrams for subsystem components, such as DiscordClient, InstagramClient, and ChatGPTClient, are available upon request. Remember this is optional

5. DATA (DATABASE) DESIGN

Our system doesn't rely on traditional databases like SQL or NoSQL databases, as the data requirements are relatively simple and don't involve complex data storage. Instead, we utilize in-memory data structures, temporary caches, and queues to support the bot's real-time functionality. These structures are sufficient for the purpose of the bot, which primarily focuses on facilitating interactions and content sharing between Discord and Instagram users.

Data Structures:

User Data Cache: We maintain a simple in-memory cache to store user data temporarily. This cache stores basic user information, such as user IDs, usernames, and session details. This allows the bot to keep track of user interactions and respond appropriately. It's worth noting that this cache is transient and doesn't persist data across bot restarts.

Message Queue: To ensure efficient message processing and response generation, we implement a message queue system. This queue temporarily stores incoming messages from Discord and Instagram users. The queue is structured to maintain the order of messages, ensuring that responses are generated in a first-in, first-out (FIFO) manner. The message queue helps manage bot responsiveness during periods of high message traffic.

Instagram Post Queue: This data structure facilitates the creation and scheduling of Instagram posts. Users can submit posts with captions, images, and scheduling information. The Instagram post queue stores these posts and their associated data, allowing the bot to automatically publish posts at the specified times.

Data Processing:

Message Processing: When a message is received from Discord or Instagram, it's first placed in the message queue. The bot processes messages in the order they were received. It reads the message, identifies the user, and manages the response process. The ChatGPT API is utilized for generating text responses.

User Interaction Tracking: The user data cache stores basic information about users, such as their user IDs and usernames. This information is updated as users interact with the bot. For example, the bot tracks the state of conversations with users.

Instagram Post Creation: Users can submit Instagram posts to the bot. These posts, along with associated data like captions and scheduling information, are placed in the Instagram post queue. The bot checks this queue and, at the scheduled times, creates and publishes the posts on the user's Instagram profile.

6. HUMAN INTERFACE DESIGN

6.1

For UI design, the interaction with our bot occurs within the Discord and Instagram chat interfaces. Users communicate with the bot by sending text messages on these platforms. The bot, in turn, responds with text-based messages. There are no visual elements or graphical screens to design.

6.2

While we don't have a graphical user interface to design, we place a strong emphasis on the user experience in our bot. The user experience is facilitated through the following principles:

1. Seamless Interaction: The bot is designed to provide a seamless and natural interaction experience for users. It understands and processes user messages, making the conversation feel as if they are chatting with a human.

2. Responsiveness: The bot is programmed to respond promptly to user messages, ensuring that users don't experience significant delays in communication.

3. User Guidance: The bot provides clear instructions and guidance to users. For example, it prompts users to send an image along with a specific command to generate Instagram posts.

4. Error Handling:In cases where users send invalid messages or requests, the bot handles errors gracefully. It sends informative messages to guide users on the correct usage.

5. Content Sharing:Users can easily share images and text, which are the primary content types for the bot. The bot assists users in generating content for Instagram posts based on their inputs.

6. Scheduled Posts: The bot allows users to schedule Instagram posts, providing a convenient way to manage their content.

7. REQUIREMENTS MATRIX

| Item 1 | Discord Interface: we need a way to access mid journey to generate images for our product, we will use the discord interface |
| --- | --- |
| Item 2 | Instagram Interface: we need access to the instagram platform in order to write messages to the end user |
| Item 3 | Chat GPT API Integration: we will be using chatGPT API to power the automated responses for our bot |
| Item 4 | Data Base(optional): We will need someone to store user messages and bot responses to make conversations flow seamlessly |
| Item 5 | Message Queue: You will always need a queue to determine which message gets sent next and implementing a message queue or similar system will be essential |