COMP3850 Project Deliverable Certificate

Name of Deliverable	*D3- Updated Project Plan, Updated Requirements/Scoping Document, Design Document, Testing Document, and Prototype/MVP Document*
Date Submitted	*29 / 04 / 2024*
Project Group Number	*Team 26*
Rubric stream being followed for this deliverable	*SOFTWARE Rubric*

We, the undersigned members of the above Project Group, collectively and individually certify that the above Project Deliverable, as submitted, **is entirely our own work**, other than where explicitly indicated in the deliverable documentation.

INITIALS	SURNAME	GIVEN NAME	STUDENT NUMBER	SIGNATURE *(IN-PERSON OR DIGITAL)*
Rx	*Ruike*	*Xu*	*46271481*	*RuikeX*
KP	*Kritchanon*	*Prasobjaturaporn*	*46122206*	*KritchanonP*
LT	*Long*	*Trinh*	*46281657*	*LongT*
OB	*Bush*	*Oliver*	*46534032*	*OliverB*
KH	*Karl*	*Holzmann*	*47092475*	*KarlH*

List of tasks completed for the deliverable and activities since the last deliverable certificate with totals for each individual team member and the whole team

Performed by *(Student Names)*	Duration *(hrs)*	Complexity *(L, M, H)*	Name of task	Checked by *(Initials)*
Long Trinh	4h	H	Use Case Diagram	KP
Long Trinh	4h	H	Use Case Description	OB
Long Trinh	1h	L	User Stories	KP/OB
Long Trinh	1h	L	Edit Project Plan and SRS	OB
Long Trinh	1h	H	Class Diagram	OB
Oliver Bush	1h	L	Revision History	LT
Oliver Bush	4h	L	Document Formatting/Proofreading	LT
Oliver Bush	6h	H	Test Case Specifications	KP
Oliver Bush	5h	H	Test Plan	KH
Karl Holzmann	4h	H	System Design Document	OB
Karl Holzmann	3h	M	Data Definitions	LT/OB
Karl Horlzmann	4h	H	Initial Bot Development	RX
Kritchanon Prasojaturaporn	1h	L	Revision Project Plan and SRS	OB
Kritchanon Prasojaturaporn	4h	H	User Interface Layouts	LT
Kritchanon Prasojaturaporn	4h	H	Window Navigation Diagram	KH
Kritchanon Prasojaturaporn	1h	L	User Stories	LT
Kritchanon Prasobjaturaporn	6h	M	Handover Requirements	OB
Ruike Xu	4h	H	State Diagrams	KH
Ruike Xu	4h	H	Test Case Specifications	KP
Team Total	62h			

Table of Contents

PROJECT PLAN

1. Revisions History Table:

Version	Release Date	Changed by	Description
1.0	28/03/24	Nexus Nomads	Version 1, including; Introduction, Risk Management, Resource Management, Change Management, Quality Management, and Project Schedule.
1.1.1	29/04/24	Kritchanon Prasobjaturaporn	Version 1.1.1 changes include; Risk Analysis matrix priority rating adjustment
1.1.2	29/04/24	Long Trinh	Version 1.1.2 changes include; Resource Management, Quality Management, Tools and Services

2. Introduction:

This document represents the project plan for Rapid Summarisation (Summarisation Discord Bot). The goal is to construct a Discord bot that utilises the Rapid Analysis Large Language Model, through API, to summarise a Discord chat log.
Included in this document:

- **Introduction** provides insight into the scope of our project. Introducing the idea of what is to be achieved as well as the potential benefits.
- **Risk Management** demonstrates all aspects of risks associated with the project and the guide on how they are handled and managed.
- **Resources Management** displays the project resources usage; what they are and how they are used.
- **Change Management** illustrates a structured approach to address unforeseen adjustments.
- **Quality Management** presents a structured approach to quality control throughout the process of product development; such as quality assurance strategies, updating testing protocols, and creating a collaborative environment for ongoing improvement.

Project Plan Scope

The Large Language Model (LLM), when given a context, is a substantial tool for all-around text-based work tasks. The goal of our project is to improve workflow through simplification of team communication through LLM. Our project is heavily anchored to Agile methodology, an iterative, flexible approach to software development. Hence, as the development progresses, we anticipate refining planned features as well as introducing additional features.

Our current scope is two main text-oriented features. Firstly, 'Topic Thread', our product would analyse ongoing conversations and summarise them into topic threads with related comments, this way users can simply catch up by just simply reading the threads. Additionally, `Daily/Weekly Digest`, our product could generate, depending on the user's setting, a daily/weekly digest. A summary of the topics discussed throughout the day or week has the potential to further improve productivity, by further reducing hours spent on communication.

3. Risk Management:

3.1. Risk Identification:

i. Technical Risks:
- **Inadequate Technical Specifications**
- **Technology Obsolescence**
- **Cybersecurity Threats**

ii. People Risks:
- **Talent Shortage**
- **Underperformance**
- **Disengagement**
- **Leadership Gap**
- **Poor Management Practices**
- **Toxic Work Environment**
- **Ethical Misconduct**

iii. Organisational Risks
- **Market Competition**
- **Process Failures**
- **Legal Risks**
- **Financial Risks**

iv. Requirements Risks
- **Ambiguity and Unclear Requirements**
- **Changing Requirements**
- **Stakeholder Involvement and Communication**

v. Estimation Risks

- **Time Estimation Risks**
- **Cost Estimation Risks**
- **Resource Estimation Risks**
- **Technical and Operational Risks**
- **Planning and Methodology Risks**
- **Market and Environmental Risks**

3.2. Risk Analysis:

i. Technical Risks:

There are a variety of project risks, including Inadequate Technical specifications, which can lead to confusion and rework if the team does not have detailed technical specifications, which can lead to different ways of completing the project, and which are not interoperable with each other. Moreover, if the market changes too rapidly, even if the project is completed, it may lose its competitiveness because the technology is outdated.

Moreover, If team members do not fully carry out systematic learning of network security, it will also give some phishing emails and information security breaches the opportunity to cause information leakage and cyber attack-related risks to the organisation.

ii. People Risks

Both external and internal causes can lead to People Risks. External causes can be epidemics, natural disasters or even poaching by competing companies, all of which can lead to turnover, resulting in the loss of valuable employees and the inability to quickly replenish the team with the skilled personnel it needs, leading to the suspension of the project.

Additionally, internal causes such as overwork or mismatched wages can lead to negative performance, which affects the team's performance and prevents the project from being finalised at the expected time.

Moreover, poor leadership decisions or inadequate management skills may lower morale within the team or increase the likelihood of conflict among team members, resulting in a lack of proper strategic direction for the team as a whole. A negative team climate and different cultures may also lead to discrimination, bullying, or physical harassment among employees.

iii. Organisational Risks

Irregular changes in the market may lead to changes in market conditions and changes in customer preferences if the organisation is unable to make agile changes at this time, eventually, the team will not be able to adapt to the market environment, and then new competitors will take some market share, and market changes may also affect the existing project plan and lead to plan changes or outdated technology.

Moreover, if the organisation does not conduct a complete review of the project's processes, it can also reduce the efficiency of the business processes and lead to additional property damage.

Furthermore, when there is a change in government policies or regulations, the organisation may also have an impact because its operational strategy or monetisation strategy does not conform to existing policies/regulations.

Also, if there is a large change in the exchange rate of market currencies, it will lead to unforeseen risks in the assets of the organisation, and when users or partners are unwilling to fulfil financial obligations, it will also cause unpredictable risks to the organisation.

iv. Requirements Risks

When the client is unable to define what they need from the project, there is a high probability that the project will be carried out at a stage where the results do not match the client's objectives, thus wasting a lot of time and financial costs.

Also, when the target scope of the project is gradually expanded due to the client's requirements, failure to recalculate the time and resources for new project changes may result in a delay in the scheduled completion of the project beyond the original budget, and how to reallocate the extra resources after the project changes is also one of the risks.

Moreover, if there is no continuous and effective communication with the stakeholders during the project, the needs of the stakeholders may be ignored or misinterpreted, and the results of the project may be different from the expectations of the stakeholders.

v. Estimation Risks

Estimation Risks share some similarities with the previous ones in that additional costs due to external factors may significantly affect the project process if the leader or team is not satisfied with the amount of time required for the project, dependencies between projects, financial requirements, and additional costs due to external factors.

Additionally, misallocation of resources, incompatibility of team members' expertise with current project requirements, faulty project management methodologies, and changes in markets and policies can also affect the project process in the long run, resulting in a failure to deliver the project correctly or delivering a project with many defects.

Risk Analysis Matrix			
Risk Number	Probability	Impact	Priority
i.	Low	Catastrophic	1
ii.	Low	Serious	3

iii.	Moderate	Serious	2
iv.	Low	Tolerable	5
v	Low	Serious	4

3.3. Risk Planning:

Risk Number	Strategies
i.	Have regular meetings with stakeholders to discuss requirements and documentation. Try to differentiate ourselves from competitors in the market. Ensure all files containing sensitive credentials are excluded from version control, accounts all have multi-factor authentication and utilise GitHub's token leak scanner.
ii.	Ensure each team member gets the same training/documentation. Check-in/have meetings regularly to update the team on progress. Reschedule meetings or otherwise assist team members. Follow conflict resolution documentation in the team manual.
iii.	Try to differentiate ourselves from competitors in the market. Create and follow conflict resolution documentation in the team manual. Ensure we follow a code of conduct and follow conflict resolution if issues occur. Reduce code complexity to minimise costs.
iv.	Have regular meetings with stakeholders to discuss requirements and documentation.
v.	Check-in/have meetings regularly to update the team on progress. Communicate with stakeholders regularly to ensure accurate estimations are made by subject matter experts. Track costs. Regularly reevaluate estimations.

4. Resources Management:

Human Resources: Includes software developers, a project manager and a lead developer.
Roles and Responsibilities: Clearly defined to ensure efficient progress and accountability.
Technological Resources: Development environments (Discord API), version control (GitHub), collaboration platforms (Jira).

Digital and Informational Resources: Secure storage and management of code repositories and project documentation on platforms like GitHub.

- Utilise Jira to allocate assignments to developers according to their skills establishing accountabilities for each outcome. Arrange resources in accordance with project stages and key points to sustain advancement and achieve target dates.
- Make sure to check team workloads using Jira reports to ensure a good balance and avoid burnout or skills going underused.
- Keep track of team availability and time commitments using Google Docs so we can adjust resources easily as project needs change.
- Look for ways to make use of resources whenever possible by using automation tools, which can help free up people for challenging tasks.
- Ask our team for feedback, on how resources are being used and any obstacles they face to find ways to improve efficiency.
- Ensure that all parties involved are kept up to date on the availability of resources including any shortages or surpluses by providing updates and dashboard summaries.
- Foster transparent communication with the project sponsor and other stakeholders to guarantee that decisions regarding resources are made openly and collaboratively.

5. Change Management:

5.1. Managing Requirements and Scope Changes:

- Change Request Process: Establish a procedure wherein any suggested modifications to requirements or scope need to be submitted as a change request accompanied by thorough explanations and reasons.
- Change Approval: Other group members will then decide to approve of the change or disapprove. Choices with more votes will be accepted.
- Change Documentation: When a change request gets approval, we will ensure all project documents are updated to include these changes. Documents to change include, SRS, Project Plan, Design, Analysis, and prototype.

5.2. Version Control:

- Consider utilising a version control platform, like Git along with GitHub to effectively oversee and monitor modifications made to the project's codebase.
- Utilise the version history tool, in Google Docs to monitor modifications made to documents as time progresses enabling us to switch to versions if needed.
- Tagging should be utilised to indicate the release points, in the repository offering pointers to the versions that have been deployed or made available, to users.

6. Quality Management:

6.1. Quality Planning:

Establish clear, measurable quality goals aligned with the project requirements and user expectations, such as accuracy of summaries, user interface responsiveness, and system reliability.
Adhere to industry standards for coding, security, and data management.

6.2. Quality Assurance:

Implement QA processes across all development stages.
Conduct regular code reviews and system audits to ensure adherence to quality standards.

6.3. Quality Control:

Comprehensive testing strategy including unit testing, integration testing, system testing, and acceptance testing to validate the functionality, performance, and security of the system.
Monitor system performance continuously to detect and resolve issues.

6.4. Quality Improvement:

Integrate user feedback to gather user input on system performance and usability, facilitating continuous improvement.
Perform root cause analysis on issues to prevent recurrence.
Regularly review and refine development and quality processes based on lessons learned and best practices to enhance future project outcomes.

6.5. Documentation and Reporting:

Maintain up-to-date documentation on quality processes and standards accessible to all team members.
Develop quality metrics reports to keep stakeholders informed about the system's quality status and improvements.

7. Project Schedule:

7.1. Tasks:

- Task 1: Members meet each other, form the team, and figure out our communication plan, roles, responsibilities and skills. After that, we had time to talk to our sponsor from Rapid Analysis and got a deeper understanding of the project. Then we decided on our bot's main focus and feature: Summarisation on Discord conversation (22nd of February, 2024)
- Task 2: Members participate in team training to get a better understanding of each other's strengths, weaknesses and skills. After that, we decided on the approaches to the project and how the system will work. We created the skeleton version of the bot and added it to our discord server. Lastly, we made a Feasibility Report, Team Manual and Architecture Report (23rd of February, 2024 to 7th of March, 2024).
- Task 3: We will research and create the first version of the Project Plan and Software Requirement Specification (SRS). We will also research on the best feature to add to the bot besides chat summarisation (8th of March, 2024 to 28th of March, 2024).
- Task 4: We will research the bot's design, and prototyping, and start to add the initial version of summarisation + Rapid Analysis API integration to the bot. Then we will update the Project Plan and SRS with Design, Test Cases, and Prototype/MVP (29th of March, 2024 to 29th of April, 2024).
- Task 5: We will test and research on how the summarisation would work in different scenarios. We will add other features to the bot after that. Then we will update the documentation with the user/training manual (30th of April, 2024 to 16th of May, 2024)
- Task 6: We will test all the features and optimise the functionalities of the bot. In addition, we will also improve the bot's design and finalise it. Then we will create a Final Group Reflective Report (17th of May, 2024 to 30th of May, 2024)
- Task 7: We will demonstrate the bot, Final Report and presentation to the Rapid Analysis representative (30th of May, 2024).
- Task 8: We will polish the bot and make a Final Delivery of the Product to the Rapid Analysis as agreed in the handover (30th of May, 2024 to 13th of June, 2024).

7.2. Deliverables:

- Deliverable 1: Feasibility Study (Thursday, 7th of March, 2024)
- Deliverable 2: Project Plan and Requirements/Scoping Document (Thursday, 28th of March, 2024)
- Deliverable 3 - Increment 1: Updated Deliverable 2, plus Design, Test Cases, Prototype/MVP (Monday, 29th of April, 2024)
- Deliverable 4 - Increment 2: Updated Deliverable 3, plus user/training manual (Thursday, 16th of May, 2024)
- Deliverable 5: Final Group Reflective Report (Thursday, 30th of May, 2024)
- Deliverable 6: Project Presentation / Demonstration (Thursday, 30th of May, 2024)

- Deliverable 7: Final Delivery of the Product to the sponsor as agreed in handover (Thursday, 30th of May up to Thursday, 13th of June, 2024)

7.3. Timeline:

	Sch...	Task Name	Duration	Start	End	Compl...	Priority
1		Team Formation	1 day	22/02/2024	22/02/2024	100%	500
2		**Del 1: Feasibility Study**	**11 days**	**22/02/2024**	**7/03/2024**	**100%**	**500**
3		Feasibility Report	7 days	22/02/2024	2/03/2024	100%	500
4		Team Manual	2 days	2/03/2024	5/03/2024	100%	300
5		Architecture Report	2 days	5/03/2024	6/03/2024	100%	200
6		**Del 2: Project Plan and Requir...**	**15 days**	**8/03/2024**	**28/03/2024**	**100%**	**500**
7		SRS	7 days	8/03/2024	18/03/2024	100%	500
8		Project Plan	7 days	18/03/2024	26/03/2024	100%	400
9		Revised Team Manual	2 days	26/03/2024	27/03/2024	100%	200
10		**Del 3: Updated D2, plus Design...**	**22 days**	**29/03/2024**	**29/04/2024**	**0%**	**500**
11		Revised SRS	6 days	29/03/2024	5/04/2024	0%	400
12		Revised Project Plan	2 days	5/04/2024	8/04/2024	0%	300
13		Develop first version of summa...	10 days	8/04/2024	20/04/2024	0%	500
14		Designing & Prototyping/MVP	5 days	20/04/2024	28/04/2024	0%	400
15		**Del 4: Updated D3, plus user/tr...**	**13 days**	**30/04/2024**	**16/05/2024**	**0%**	**500**
16		Improve summarization	9 days	30/04/2024	11/05/2024	0%	500
17		Add features into the bot	6 days	8/05/2024	15/05/2024	0%	400
18		Revised SRS	4 days	30/04/2024	5/05/2024	0%	300
19		Revised Project Plan	3 days	5/05/2024	8/05/2024	0%	200
20		User/Training Manual	3 days	12/05/2024	15/05/2024	0%	300
21		**Del 5 & 6: Final Group Reflecti...**	**10 days**	**17/05/2024**	**30/05/2024**	**0%**	**500**
22		Final Group Reflective Report	3 days	17/05/2024	21/05/2024	0%	500
23		Project Presentation Preparation	4 days	21/05/2024	25/05/2024	0%	500
24		Improve the bot features	8 days	17/05/2024	28/05/2024	0%	400
25		Presentation & Demonstration...	1 day	30/05/2024	30/05/2024	0%	500
26		**Del 7: Final Delivery of the Pro...**	**10 days**	**31/05/2024**	**13/06/2024**	**0%**	**500**
27		Finalize the bot	6 days	31/05/2024	9/06/2024	0%	400
28		Handover to Sponsor	1 day	10/06/2024	10/06/2024	0%	500

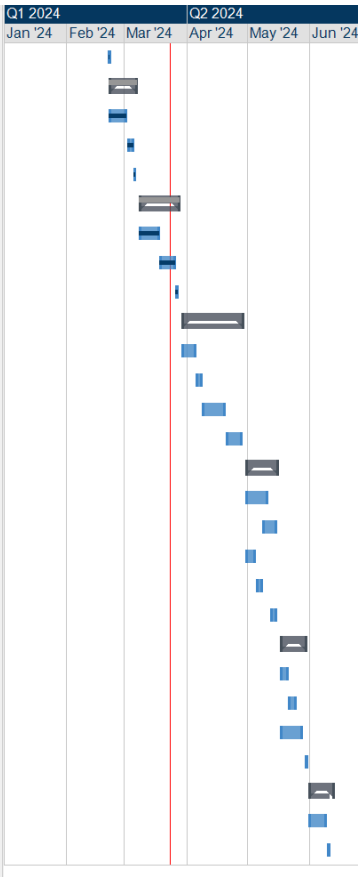

Fig 1: Project Timeline

7.4. Resources allocated:

7.4.1. People:

Our team has 5 members. Each sprint the roles will be rotated to different members. 3 main roles will be: Project Manager, Lead Developer and Developers.

7.4.2. Tools and Services:

- Discord Environment - Last Updated April, 2024
- Rapid Analysis API - Last Updated Dec 1, 2023
- Github - Version: 3.12.2 April 18, 2024
- Jira - Version: 7.5.4
- Gmail - Last Updated 25 Apr 2024

- Google Doc - Last Updated 25 Apr 2024

7.4.3. Time: Total: 14 weeks

- Researching phase: 6 weeks
- Development phase: 5 weeks
- Testing Phase: 4 weeks
- Deployment Phase: 2 weeks

7.5. Assumptions:

- Resource Availability: It is assumed that all team members and required tools will be accessible as scheduled throughout the project duration.
- API Accessibility: The project anticipates the availability of Discords API and Rapid Analysis's LLM API with rate limits to facilitate development and testing stages.
- Technological Stability: The project supposes that there will be no alterations, to the core technologies, such as updates to Discords platform or removal of APIs necessitating modifications, to the project scope or timeline.
- Budget Stability: It is assumed that the project will not be costly, and will not require the team to spend unexpectedly.
- Supports: The project will receive support from sponsors from Rapid Analysis and University if needed. Assistance can include academic help, tools, licences or servers from Rapid Analysis.
- Risk Management: The project plan operates under the assumption that any risks identified will be handled and minimised in line, with the strategies detailed in the risk management plan without leading to delays.

7.6. Templates/Standards:

- All of our templates are provided by the university. We can add and adjust the templates based on the sponsor's desire or team member's suggestion if approved.
- Our documentation standards will be based on the deliverable rubrics to meet the requirements.

8. Handover Requirements:

This section outlines the handover process to the client. It serves as a guide for the client to understand the functionalities, installation, operation, and maintenance of the bot.

8.1. Software and Data Handover

8.1.1. Final code

The handover of the final code for the Discord bot includes all source files. Additionally, we provide links to the version-controlled repository, in this case, GitHub, along with the necessary access permissions to ensure the client can retrieve updates or modify the code as required.

8.1.2. Dependencies

The bot's source code is developed using the Node.js framework, ensuring a robust and scalable application environment. Key libraries employed in the bot include Discord.js which provides an interface to interact with the Discord API, and node-fetch, utilised for making network requests. These dependencies are crucial for the bot's functionality. Hence, to facilitate a smooth setup, detailed documentation on installing and configuring these libraries will be provided, ensuring all dependencies are correctly managed to maintain optimal bot performance.

8.1.3. Database schema

We will be providing our database schema.The database will consist of a single table named rapidanalysis containing all of the data needing to be stored. This will consist of 3 columns, all being the type varchar: userID, apiKey and userPreferences. userID will store a specific user's Discord ID and will be the primary key of the table, apiKey will store that user's RapidAnalysis API key and userPreferences will store a JSON object containing preferences the user may modify. If this table does not exist, it will be created by the application.

8.2. Testing

This includes a thorough Documentation of Test Cases, where each test case is described in detail, specifying the expected inputs and outputs.

8.2.1. Test Cases

Acceptance

Acceptance testing is scheduled after completing the system testing. This phase aims to confirm the discord bot meets the acceptance criteria defined by the project's sponsor from Rapid Analysis. The following assessments will be performed in this stage:

- Test Scenario Execution: Execution of all pre-existing tests performed in earlier stages of the test plan.

- User Experience Assessment: Evaluation of the bot's user interface, responsiveness, and overall user experience. Feedback will be collected from stakeholders to identify any usability issues.
- Compatibility Testing: Verification of the bot's compatibility with different Discord client applications (web, desktop, mobile) and platforms.

8.2.2. Testing Tools/Environments

We plan to provide a comprehensive description of the testing tools and software that will be used throughout the project. While we have not yet finalised the specific tools and setup details, we are evaluating various options to ensure thorough and efficient testing. Once the tools and environments are selected, we will include detailed guidelines on how to set up these environments, essential for replicating tests and resolving any issues found.

8.2.3. Test schedule

Including a test schedule as part of the handover documentation emphasises the comprehensive approach we've taken to quality assurance throughout the development of the Discord bot. This schedule details each testing phase, from initial component tests to full system and acceptance testing, providing visibility into the rigorous process undertaken to ensure the bot's functionality and reliability.

8.3. Deployment and Maintenance

Deployment of the bot could be done automatically through a CI/CD pipeline using a service such as GitHub Actions. We will provide documentation on how this pipeline is configured, along with instructions on how to trigger deployments and manage the workflow. The bot will be deployed onto an Amazon Web Services EC2 instance provided by RapidAnalysis. Our documentation will include steps for setting up and managing the EC2 instance, as well as guidelines for ongoing monitoring and maintenance.

8.4. Documentation and Training Materials

8.4.1. Project Plan

The documentation articulates on the purpose, scope, and detailed execution strategy for the Discord bot. The goal is to provide the client with a transparent view of the project’s lifecycle and ensure they are aware of how each stage of development was managed to meet their requirements. The documentation reflects the methodologies used, particularly the agile approach. And it encapsulates the project's history, from conception through to completion. By reviewing the project plan, the client can gain insights into the decision-making processes, risk management strategies, and quality assurance practices that

were integral to the development of the Discord bot. Moreover, the schedule and resource allocation details provide a clear narrative of the project's progress.

8.4.2. Software Requirement Specification (SRS)

The SRS is included to give the client a complete overview of the Discord bot's functionalities and specifications. Adhering to the IEEE standard, the SRS serves as a clear agreement on the system's capabilities and requirements. It details the bot's intended use, operating conditions, and user documentation, and it lays out functional and non-functional requirements. This document ensures that the client fully understands the software's features and limitations

8.4.3. Design Documentation

The Design Documentation for our Discord bot project, built with object-oriented programming principles include a comprehensive System Design Document that outlines the architecture and high-level decisions. It features a User Interface Layout to visually guide the bot's interaction within Discord and a Window Navigation Diagram to show the flow between different user commands. Data Definitions are provided in tables with field names, types, and examples, ensuring clarity on data storage.

Included are Class Diagrams that detail the bot's structure, associations, and methods. Sequence Diagrams for each use case ensure the logic flow is clear and traceable. Finally, State Diagrams for complex objects give insight into their various states and behaviours.

8.4.4. Testing Documentation

Documentation outlines our structured approach to testing, detailing both strategies and specific test cases, which is crucial for verifying that the bot meets all functional and performance requirements.

8.4.5. Prototype / Product (MVP)

This documentation is critical as it not only reflects the iterative nature of our project development but also underscores the importance of sponsor feedback in refining our prototype.

8.4.6. Analysis Documentation

We include Analysis Documentation in our handover package to provide a foundational understanding of the application's functional requirements from a user perspective. It includes a Use Case Diagram for visual mapping of user interactions, detailed Use Case Descriptions for each functionality, and User Stories to succinctly capture the end-user perspective, ensuring the application's design and functionality align with user expectations.

8.4.7. Training Materials: User Manual

We are to prepare a comprehensive user manual designed to assist users in understanding and fully utilising the bot's capabilities. This manual is made to cater starting from users who may be entirely unfamiliar with the system to users with moderate computer knowledge.
It includes the following key sections:

- **Clear Table of Contents (TOC)**: An organised TOC that guides users to relevant sections of the manual.
- **Installation Guide**: Step-by-step instructions on installing the bot, complete with necessary screenshots to illustrate the process.
- **Configuration Settings**: Easy-to-follow configuration settings for the bot.
- **Usage Instructions**: Descriptions of how to use the bot's features, including command syntax and examples.
- **Troubleshooting and Support**: A troubleshooting guide to help users resolve common issues, along with information on how to obtain further help and support.

Software Requirements Specification

for

Discord chat summarising bot with Rapid Analysis Large Language Model through API

Version 1.2 approved

Prepared by Nexus Nomads

9/3/24

1. Revision History Table

Version	Release Date	Changed by	Description
1.0	28/03/24	Nexus Nomads	Version 1, including; Introduction, Overall Description, Requirements, and Sponsor Feedback.
1.1.1	29/04/24	Oliver Bush	Version 1.1.1 changes including; Document Conventions title and numbered points.
1.1.2	29/04/24	Oliver Bush	Version 1.1.2 changes including; User Class and Characteristics descriptive names provided.
1.1.3	29/04/24	Kritchanon Prasobjaturaporn	Version 1.1.3 changes including; Intended Audience and Reading Suggestion Lecturer removed.
1.1.4	29/04/24	Oliver Bush	Version 1.1.4 changes including; Design and Implementation Requirements, operation within discord

2. Introduction

2.1. Purpose

This Software Requirements Specifications (SRS) document outlines the requirements for the development of a Discord conversation summarisation bot, referred to as the "Rapid Summary Bot", designated as Version 1.0. The scope of the SRS includes the software requirements needed to implement only the core functionality of the Rapid Summary Bot in Discord. The development team, stakeholders, and everyone else with a stake in the project might turn to this SRS as an outline. It specifies the specifications for operation, expected use cases, technical constraints, and functional and non-functional requirements for the Rapid Summary Bot's development and deployment.

2.2. Document Conventions

2.2.1. Fonts and styles:

Title: Titles will be in Times New Roman. Their size is 32.
Headings: Headings will be in Times New Roman. Their sizes are from 18-14-12 depending on the hierarchy.

Statements: Statements will be in Arial. Their size is 11.

2.2.2. Figures: ***Fig 'num': Name*** layout in Arial, bold and Italic, size 11 below all figures. A Name is necessary and a description is optional.

2.2.3. Highlightings: **Key Terms and Important Concepts** will be emboldened.

2.3. Intended Audience and Reading Suggestions

The Rapid Summary Bot, Version 1.0 Software Requirements Specifications (SRS) document is created with the intention of appealing to a wide range of audiences:

- For **Developers/Lead Developers**, who are mostly interested in the technical specifications, the sections that address system features, interface requirements, and functions will be especially helpful.
- For **Project Managers**, who are mostly interested in the overall scope of the project, milestones and resources, the Introduction and Overview sections will be helpful for them to start.
- For **Sponsors**, who would be most interested in the project's goals, and desired final product, Overall Description will be the most crucial section for the sponsor to focus on. Before that, they should have a brief look at the Introduction.
- For **Users**, in order to understand the bot's capabilities and how to communicate with it, users need to read the overview first. Insights on how the bot could enhance their Discord experience can be gained by understanding the functional needs.

Document Organisation and Reading Sequence:

This SRS will be divided into 3 major parts:

- **Introduction:** The objective, scope, and target audience of the document are all summarised in the introduction.
- **Overall Description:** Provides audiences with an up-close view of the bot's capabilities and constraints by outlining a larger context of the product's requirements and most significant components.
- **System Features and Requirements:** Specifications for the bot's creation and operation are laid forth in the system requirements and features section, which delves into functional, non-functional, and interface requirements.

Reading the project's Introduction and Overall Description first will provide readers with an organised reading experience and a firm grasp of its essentials. After that, according to what was said before, readers should concentrate on the parts of the SRS that are most applicable to their interests and duties, so that they fully understand the material that is important to their position in the project. This method makes sure that all kinds of readers may quickly find and grasp the information that's important to them in the document.

2.4. Project Scope

By automating the generation of summaries of chat conversations, the Rapid Summary Bot improves user experience and makes it simpler to navigate topics on Discord. Confronting the problem of information overload, its creation is in line with the objective of enhancing community interaction and information accessibility on Discord.

Our Main Objectives:

- Make it simpler for people to join or rejoin ongoing conversations to increase engagement.
  Quickly find relevant talks without having to go through whole chat histories; this will save time for both users and admins.
- Foster Unity Within the Community: Facilitate a more welcoming community by making sure everyone can keep up with the latest news.
- The initiative's direct support for bigger business projects to enhance engagement in digital communities and user satisfaction leads to increased platform adoption and retention, among other benefits.

Vision Statement:
A unique solution that simplifies the consumption of large conversation data is offered by the Rapid Summary Bot, with the vision of redefining the way users connect with Discord. By using state-of-the-art natural language processing technology, this bot is designed to improve the Discord experience. It does this by summarising chat in a server in a concise and appropriate manner, so users can easily catch up on and join in on conversations. Therefore, our main objective is to make it easier for community members to stay united and put their effort into building a better server. In addition, it will also enhance members' experiences and the quality of Discord servers. Incorporating this vision into the Rapid Summary Bot will help us achieve our goal of enhancing digital communication and community engagement on the platform, which is to become an indispensable tool for both users and administrators.

Scope:
- **Initial Release (Version 1.0):** Gives top priority to developing tools that accurately summarise chat discussions in order to support English-speaking servers. Important considerations include providing understandable summaries, making it easy to install bots within servers, and ensuring data processing privacy and security.
- **Future Developments:** Future plans include adding more diverse functions into the bot, such as chat analysis, moderation or more features from Rapid Analysis API.
- **Target Audience:** Both Discord server owners looking to improve community management and users looking for a more streamlined method to interact with chat data may benefit from this.
- **Business Strategy Alignment:** The goal of developing the bot is to enhance digital communication technologies, namely Discord, so that it attracts more users and keeps the ones it already has happy.

2.5. References

- Discord Developer Documentation:
  + Author: Discord Inc.
  + Version: 10
  + Date: September 1, 2022
  + Location: https://discord.com/developers/docs
  + Description: The Rapid Summary Bot's development relies heavily on the information provided by this Discord API documentation, which includes development guidelines, authentication methods, rate restrictions, and terms of service.
- Rapid Analysis LLM API Documentation:
  + Author: Rapid Analysis

+ Version: 1.0
+ Date: Dec 1, 2023
+ Location: https://rapidanalysis.github.io/tos.html
+ Description: Describes the Rapid Analysis LLM API, which the bot will use to summarise texts, including its capabilities, limitations, and technical requirements.

3. Overall Description

3.1. Product Perspective

The Rapid Summary Bot is a new, self-contained product. It will allow users of the Discord chat platform to be able to interface with the LLM run by RapidAnalysis.

3.2. Product Functions

The Discord Chat Summary Bot will allow users to access many different functions through the Discord chat platform. Listed below are the features of the Discord Chat Summary Bot.

3.2.1. Automatic summarisation of Discord conversation threads.
3.2.2. Real-time generation of concise summaries highlighting main talking points.
3.2.3. Integration with Discord servers to provide summary services to users.
3.2.4. Personalised summarisation based on user preferences and server settings.
3.2.5. Dashboard functionality for server administrators to view analytics and insights.

3.3. User Classes and Characteristics

There are three different types of users for the Discord Chat Summary Bot:

3.3.1. Regular Discord users:

Regular Discord users are individuals who participate in Discord communities and have the opportunity to benefit from the summarisation services provided by the bot.

3.3.2. Server administrators:

Server administrators are users responsible for managing Discord servers who utilise the bot's dashboard functionality to gain insights into server activity.

3.3.3. Developers:

Developers are individuals involved in developing and maintaining the Rapid Summary Bot.

3.4. Operating Environment

The RapidSummary Bot will operate within the Discord platform, utilising Discord's API for seamless user interaction when added to a Discord server. It will be compatible with the various hardware platforms and operating systems supported by Discord, including web, desktop, and mobile environments. The bot will be designed to run on any machine that is capable of running the Node.js runtime and has a network connection but will ideally run on a Linux machine.

3.5. User Documentation

User documentation can be delivered in Discord with the use of a help command. This can return a paginated help message or a link to online documentation. While typing a command, Discord will also show a short description of the command and the parameters for the command, allowing users to autofill.

3.6. Assumptions and Dependencies

We will assume that each server has a sufficiently active user base to justify automatic summarisation and provide enough context to the LLM for coherent and accurate responses and that users have a basic understanding of how to use and navigate within the Discord app. We will also assume that the bot is provided all the permissions it requires and that users are given permission to use commands in their servers. The bot will depend on 3 libraries available on NPM. These libraries are discord.js to provide us with a native interface to the Discord API, and node-fetch to provide a simple way to perform network requests to the RapidAnalysis API and dotenv which allows us to load secret values from a file rather than hard coding them. We also depend on both the Discord and RapidAnalysis APIs to be stable and freely available.

4. Requirements

4.1. Functional Requirements

R.1. Chat Summarisation:

Description and Priority

- If a Discord server has this capability, the Rapid Summary Bot can automatically summarise chats inside the server. The goal is to provide them with a synopsis of the conversation by pulling out the main ideas and topics. This feature is highly prioritised since it is essential to the product's goal of improving the user experience by reducing information overload.

Chat Summarisation Requirements

- **R.1.1.Summary Generation**
    + Description: The system shall generate a summary based on user desired instructions, such as from a certain part or period of the conversation.
    + Error Response: The system shall send an error message to the user outlining the problem if summarisation is not possible because of incoherent data or any other reason.
- **R.1.2. Summary Presentation**
    + Description: The system shall give the user the summary inside Discord, but make it stand out from the rest of the discussion by utilising a clean, easy-to-read style.
    + Error Response: The system shall send an error message to the user to inform them if it cannot send messages or retrieve messages within the server because of no permission provided or any other problems.
- **R.1.3. Authentication and Authorisation**
    + Description: The system shall securely connect with Discord's API and gain the appropriate permissions from the server admins, in order to access the conversation logs for summarisation.
    + Error Response: The system shall notify the requesting user of the exact problem and offer suggestions for resolving it in the event that authentication fails or permissions are inadequate.
- **R.1.4. Integration with Discord Features**
    + Description: The system shall integrate seamlessly with Discord's existing features, such as roles and permissions, to respect user privacy and server settings.
    + Error Response: The system shall notify the user if they try to use a higher command than their current role or if there is a problem with the compatibility with Discord features and the bot
- **R.1.5. Interactive Summary Requests**
    + Description: The system shall enable users to request summaries in a dynamic way using chat commands, narrowing their search by subject, time range, or channel.
    + Error Response: The system shall notify users if there are problems or restrictions with handling interactive requests, such as commands that were not recognised or channels that were unreachable.

R.2. User Customisation and Preferences:

Description and Priority

- Users may adjust the Rapid Summary Bot's actions based on their own or the server's preferences, such as the amount of time between summary updates, how often they occur, and which subjects are most important to them. Medium, which plays a significant role in improving user happiness and engagement via customisation of the summary process to suit varied demands is the focus of this feature.

User Customisation and Preferences Requirements

- **R.2.1. Interface for Customisation**
    + Description: The system shall offer a user-friendly interface that lets users customise summary parameters, such as the length, frequency, and subjects that interest them. This may be done by commands in Discord or a graphical user interface that is available on the server.

    + Error Response: The system shall send a helpful message outlining legitimate customisation choices that will be shown by the bot in response to any user input that is deemed invalid.

- **R.2.2. Preference Persistence**
    + Description: The system shall keep user choices for personalisation safe by linking them to their Discord profile; this way, preferences will follow the user from session to session and server to server.
    + Error Response: The system shall notify users and try again shortly if there's a technical problem and storing preferences doesn't work.
- **R.2.3. Server-Level Default Settings**
    + Description: The system shall provide an option for server administrators to define default parameters for summarisation, which users may then customise according to their preferences.
    + Error Response: The system shall let the administrator know and offer other ones if the default settings don't save or don't work with the bot's capabilities.

R.3. Error Handling and Recovery:

Description and Priority

- Users may get to know the errors that they encountered and they will also be sent to the bot developers. The priority of this feature is below medium, it is crucial to get user feedback and to improve the bot’s functionalities.

Error Handling and Recovery Requirements

- **R.3.1. Error Detection and Logging**
    + Description: The system shall automatically detect and log errors. It will be sent to developers and feedback can be added by users for further information. This will help developers to have an in-depth understanding of the error and its condition.
    + Error Response: The system shall notify users if the error is unknown. Then, detailed information of the error will be sent to developers and users’ state will be reverted back to previous functional action.
- **R.3.2. Error Recovery**
    + Description: The system shall automatically revert back to the previous functional state to continue receiving the user's commands and show a message to ask the user to try again. This will help limit the error range and possibility for developers.
    + Error Response: The system shall notify users and try again shortly if it cannot revert back. If the second revert still did not work, the bot shall revert back to its first state (the “home” state).

4.2. Design and Implementation Requirements/Constraints

R.4. Operation within Discord

+ Description: The system shall operate within the Discord application. Including different applications and platforms.

R.5. Rapid Analysis Integration

+ Description: The system shall integrate its functionality through rapid analysis API

R.6. Implementation Language

+ Description: The system shall be implemented using Javascript

4.3. Usability Requirements

R.7. Command usage

+ Description: The system shall use a slash command within the discord platform before a command word for each system function.

R.8. User Documentation

+ Description: The system shall have clear and easily accessible user documentation for training.

4.4. Nonfunctional Requirements

R.9. Performance:

Description and Priority:

- Numerous Discord servers have extensive member count, exceeding 10,000 users, for instance. However, a closer inspection often reveals a minimal daily engagement, with perhaps only 20 to 30 members initiating conversation. Hence, an effective engagement metric to consider is the daily active users engaging through messaging. Specifically, having at least 250 unique users sending messages daily is indicative of an active community. This level of interaction is anticipated for our product's environment.

Performance Requirements:

- **R.9.1.** The system shall have a maximum return time of one minute irrespective of the volume of users.

R.10. Safety:

Description and Priority:

- Our product is built around our users' chat log information which may contain sensitive information about the users, and their acquaintances. Given this, the concern of sensitive data leakage is the most pressing matter regarding safety and security.
- Our framework prevents data leakage by ensuring that the bot does not store personal data beyond what is necessary for the summarisation process, as well as complying with relevant privacy laws. The bot will be compliant with the Australian Privacy Principles (APPs) under the Privacy Act 1988, ensuring the protection of personal information.

Safety Requirements

- **R.10.1.** The system shall not store personal data beyond what is required for the summarisation process.
- **R.10.2.** The system shall be compliant with the Australian Privacy Principles (APPs) under the Privacy Act 1988.

R.11. Security:

Description and Priority:

- Several requirements have been identified, to ensure the potential sensitivity of user data and prevent unauthorised and misuse of our product.
- First requirement, 'Discord Bot token' must be under maximum protection. Every Discord Bot has its own unique assigned code, those in possession of it are granted the access to the bot and the abilities to manipulate the bot's behaviour, access its permissions and information. Hence, the possible consequences of token code leakage is astronomically. Undeniably, the access to the token is limited, and distributed strictly on a need-to-know basis to ensure maximum security. Any file that may contain the token and any other sensitive information will be excluded, those files will be updated to `**gitignore**` file. This prevents any leakage during the version control process.
- Second requirement, our product must operate with server permission to the minimal extent necessary. This is to prevent our product from becoming over-privileged and allow for misuse for malicious activities.
- Lastly, the respect to the Discord rate limits. The software user request limit rate must be set according to the specified limits of Discord rules. This is to prevent our software from receiving a ban from Discord marketplace.

Security Requirements

- **R.11.1.** The system token shall be protected by maximum security.
- **R.11.2.** The system token shall be distributed strictly on a need-to-know basis
- **R.11.3.** The system shall operate under the least privilege necessary
- **R.11.4.** The system shall respect specified limits set by Discord

R.12. Software Quality Attributes:

Description and Priority:

- The successful deployment and operation of the product involves multiple software quality attributes. Here are the necessary attributes:
- **Adaptability**, our software is set to be highly adaptive, which means it must possess the ability to adjust to all types of work environments; such as from a small group of users with leisure interaction to a vast community with extensive interaction. A continuous communication with our partner to ensure the infrastructure can sustain high-traffic periods and scale according to user demand.
- **Availability,** our software definition of high availability is to be reliable, fails infrequently, and easily maintainable, can be easily and quickly repaired. Good coding practices are essential for easy **maintenance**, and extensive code reviews help prevent poor code from being committed.
- The **Correctness** of our program is to be measured through testing with users in real-world scenarios. Their feedback will be the criteria for correctness. Adjustments will be made accordingly.
- **Flexibility** is the degree of difficulty in refining or adding features to the current form of the software. Our software principle is built around API usage, essentially a bridge between two software with a streamlined interface for the users. As our LLM partner releases improvements and new features, our software, like a universal adapter, guarantees effortless integration.
- Our software exhibits a high degree of **usability** by the nature of Discord Bot. For a user that wishes to use our software, it is figuratively and literally like communicating with other users.

Software Quality Attributes

- **R.12.1.**The system shall adapt to all types of work environments, regardless of the usage
- **R.12.2.**The system shall maintain high availability, ensuring minimal instances of failure. In the rare event of a failure, swift and straightforward remediation will be ensured
- **R.12.3.**The system shall maintain a high level of flexibility such that any modification or improvement is effortlessly integrated.
- **R.12.4.**The system correctness shall be measured through the feedback of its users
- **R.12.5.**The system shall be user-friendly by providing clear, concise instructions and feedback to users for each transaction.

5. Sponsor Feedback:

- **Meeting date and time:** 28/03/2024
- **Feedback received:** Advice for further updates in Deliverable 3 and 4 regarding testing and prototyping
- **Team response/action points:** No action was needed

Analysis

1. Use Case Diagram

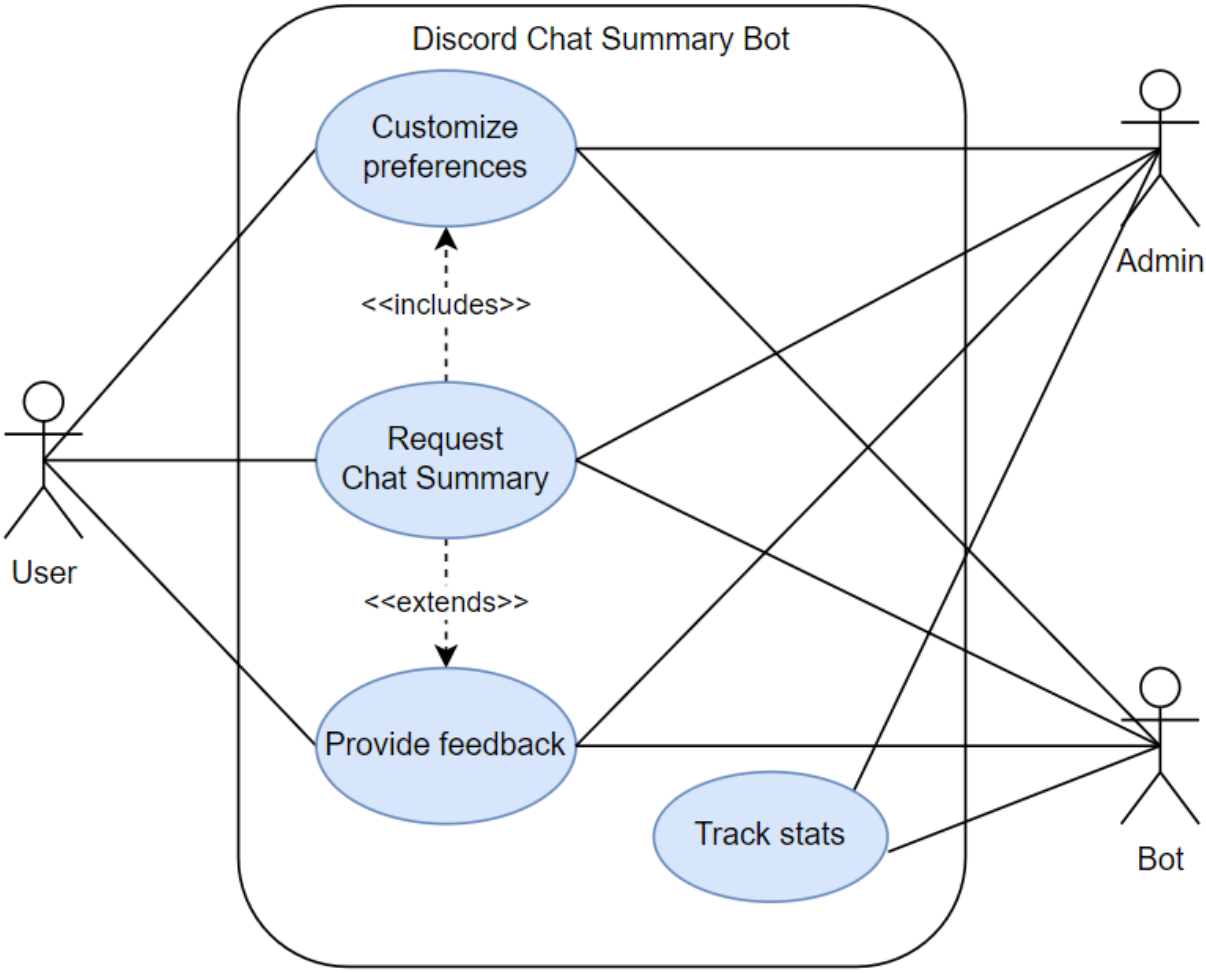

Fig 1: Use Case Diagram

2. Use Case Description

Use Case 1	UC01: Chat Summary	
Goal	Receive a concise summary of recent or specified chat conversations.	
Preconditions	The user is authenticated and a member of the server. The bot joins the server. The bot is on the same server as the command. The bot has permission to access chat logs.	
Success End Condition	The bot generates a precise summary of the latest default number of messages in the channel.	
Failed End Condition	The bot showed error messages. The bot did not show any content in the summary part. The bot showed an inaccurate summary of the conversation.	
Actors	Primary Actor: User, Bot Secondary Actor: Discord API, Rapid Analysis API	
Trigger	The user enters "/" into the channel chat bar.	
Description / Main Success Scenario	**Step**	**Action**
	1	The bot displays its recommended commands (main features) for users to choose from.
	2	The user clicks on the "/summarise" command or enters "summarise" into the chat bar and sends it. If the user wants to get a summary of a specific part of the channel chat logs, please refer to "**Alternative Flow 1: Specify Preference**". If the bot is not available, please refer to "**Alternative Flow 2: Bot is down**".
	3	The bot retrieves the latest default number of messages (set by users, initially is 20, from user preferences) from that channel with **Discord API**.

		If the user wants to set the default number, please refer to "**UC02: Customise Preferences**"
	4	The bot processes to summarise the messages with **Rapid Analysis API**.
	5	The bot sends the summary to that discord channel or sends it to the user privately (depending on user preferences).
	6	The user chooses to "**Provide feedback**" of the summary and send feedback to the bot. Please see "**UC03: Provide feedback**" for details.
	7	The use case ends.

Alternative Flow 1 Specify Preference	**Step**	**Branching Action**
	1	The user enters "summarise + <<numbers of latest messages>>" - summary of the last specific number of messages or "summarise + <<time starts>> + <<time ends>>" - summary of messages in specific time stamps or "summarise + <<number + 'h'>>" - summary of messages in the last hour(s) into the chat bar and sends it.
	2	The bot retrieves messages based on those specific preferences.
	3	Continue to step 4 of the main flow.

Alternative Flow 2 Bot is down.	**Step**	**Branching Action**
	1	Discord sends "The application did not respond" message.
	2	The user sends a message to the bot developers about the situation.
	3	Bot developers investigate and respond to the user after the bot is back online.

Use Case 2	UC02: Customise Preferences	
Goal	Personalise the summaries that the bot generates on the server.	
Preconditions	The user is authenticated and a member of the server. The bot joins the server. The bot is on the same server as the command. User preference storage is active.	
Success End Condition	User preferences are stored and applied to all subsequent summary requests.	
Failed End Condition	The bot showed error messages. The bot did not store any user preferences. The bot did not apply user preferences.	
Actors	Primary Actor: User Secondary Actor: User preference storage	
Trigger	The user enters “/” into the channel chat bar.	
Description / Main Success Scenario	**Step**	**Action**
	1	The bot displays its recommended commands (main features) for users to choose from.
	2	The user clicks on the “/preferences” command or enters “preferences” into the chat bar and sends it. If the bot is not available, please refer to “**Alternative Flow 1: Bot is down**”.
	3	The bot shows all the preferences that the user can set. Including: Default number of latest messages, Summary length limit, Summary privacy and “**Cancel**” button. If the user wants to cancel the operation, please refer to “**Alternative Flow 2: Cancel Operation**”.
	4	The user chooses one option.

	5	The bot displays the preference's description and prepares a template in the chat bar for users to enter the number. If the user wants to cancel the operation, please refer to "**Alternative Flow 2: Cancel Operation**".
	6	The user enters the desired numbers and sends it. If the user enters **not allowed numbers**, please refer to "**Alternative Flow 3: Not Allowed Number**"
	7	The bot saves user preference in User **Preferences Storage** and sends a message "Your Preferences is saved".
	8	The use case ends
Alternative Flow 1 **Bot is down.**	**Step**	**Branching Action**
	1	Discord sends "The application did not respond" message.
	2	The user sends a message to the bot developers about the situation.
	3	Bot developers investigate and respond to the user after the bot is back online.

Alternative Flow 2 **Cancel Operation**	**Step**	**Branching Action**
	1	The user clicks "/cancel" or enters cancel and sends the command.
	2	The bot cancels the current operation and goes back to the first state "/".

Alternative Flow 3 **Not Allowed Number**	**Step**	**Branching Action**
	1	The user enters a "not allowed number" and sends the commands.

	2	The bot detects the number and sends a message “You entered an unacceptable number. Please Try Again”.
	3	The bot reverts to the previous state. Continue to step 6 of the main flow

Use Case 3	UC03: Provide Feedback	
Goal	The user provides feedback after the summary and the bot successfully receives it.	
Preconditions	The user is authenticated and a member of the server. The chat summary is completed.	
Success End Condition	The bot receives the feedback and sends a success message to the user.	
Failed End Condition	The bot showed error messages. The bot did not show a feedback text field. The bot feedback cannot be saved after the user sends it.	
Actors	Primary Actor: User. Secondary Actor: Feedback storage.	
Trigger	The user receives a chat summary from the bot.	
Description / Main Success Scenario	**Step**	**Action**
	1	The bot displays a text field for the user to enter their feedback (Optional).
	2	The user enters their feedback or leaves it empty.
	3	The user clicks the “send” button or press Enter to send feedback. If the user left the field empty, please refer to “**Alternative Flow 1: Empty feedback**”
	4	The bot receives and stores the feedback into its storage.

	5	The bot sends the message “Feedback received. Thank you for using our service”.
	6	The use case ends.
Alternative Flow 1 Empty feedback	**Step**	**Branching Action**
	1	The user leaves the feedback field empty.
	2	The bot will count it as positive feedback without storing anything else in the storage.
	3	Continues to Step 5 of the main flow.

3. User Stories

US in this context stands for ‘**User Stories**’.

US 01.01 Request Chat Summary
As a Discord user, I want to request a chat summary from a certain part in a channel so that I can quickly catch up with the conversation without spending time scrolling through a large number of chat messages.

US 01.02 Customise User Preferences
As a Discord user, I want to customise my preferences for the default number of latest messages, summary length limit, and summary privacy so that the summary I receive meets my requirements and makes it easier for me to catch up with the conversation.

US 01.03 Provide Feedback on Summary
As a Discord user, I want to provide feedback on the summary that I receive so that the developers can improve the bot based on customer feedback.

US 01.04 Access Channel Statistics
As a Discord community administrator, I want to get statistics of the messages of a channel in my server so that I can understand the quality and the engagement of that channel.

Design

1. System Design Document

1.1. System Architecture

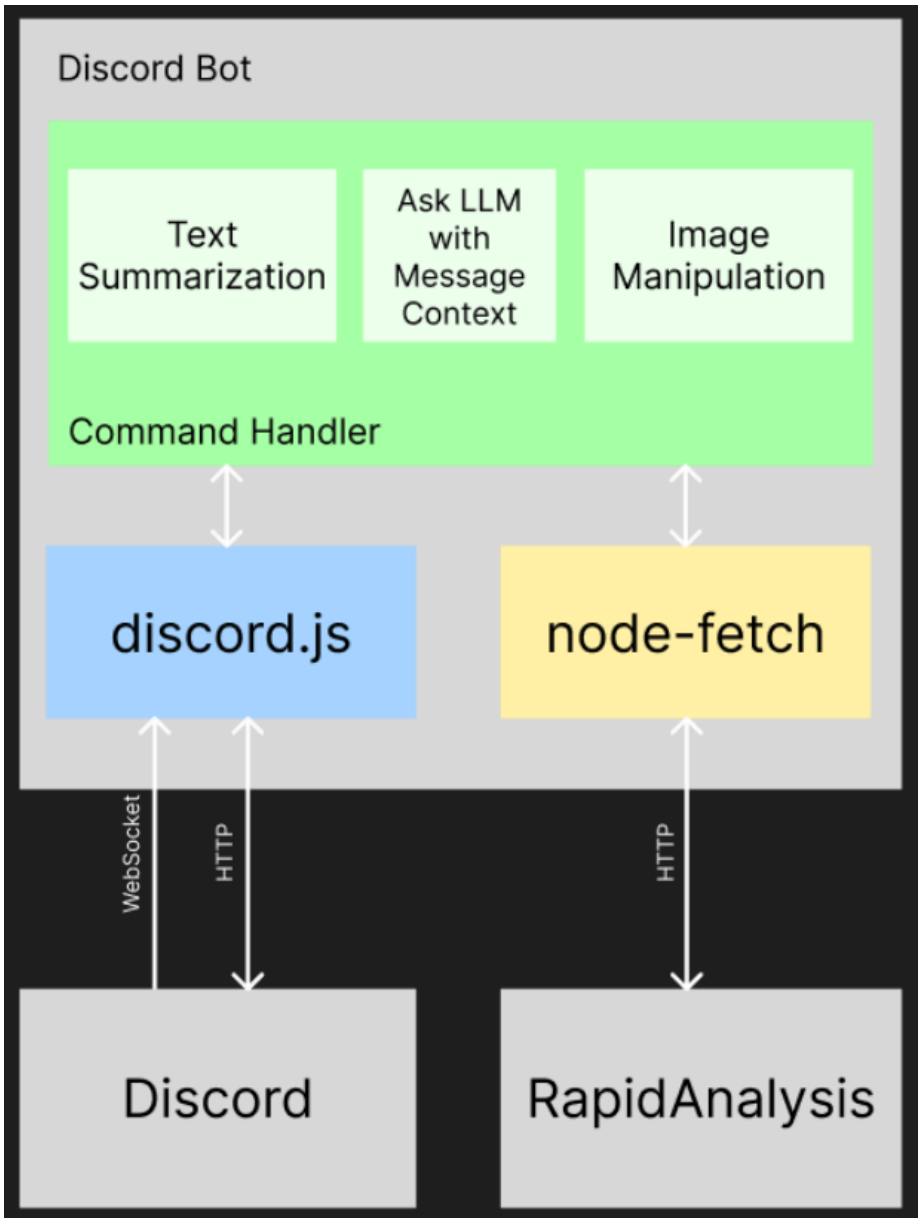

Fig 1: System architecture diagram

Discord fires Message Create events over the WebSocket connection for each message that gets sent in Guilds the bot is in. These messages can be parsed as commands. An alternative

approach is adding our own commands to the native command list in Discord. Using these would instead fire an Interaction Create event. These events are handled and emitted to our code by discord.js.

All other operations (sending messages, images, mass fetching messages) would occur over the REST API with discord.js methods.

1.2. Persistent Data Strategy

As each user/guild provides their own API keys to use the bot and can change personal settings within the bot, we will have to persist this data on disk. The strategy we have devised utilises a database engine, either MySQL/Postgres or the simpler SQLite.

1.3. Concurrent Processes

As JavaScript is a single-threaded language, very few tasks can be run concurrently. The event loop within JavaScript however allows us to defer long-running tasks such as network requests to the end of the loop, allowing us to handle other incoming events from the Discord API or perform other data processing.

1.4. User Interface Strategy

UIS 01.01 Integrate with Discord Interface
As a developer, I want the user interface to integrate seamlessly with Discord's existing interface, utilising text-based commands and interactions within Discord chat, to maintain consistency and ease of use.

UIS 01.02 Ensure Accessibility
As a developer, I want to ensure that all text outputs are accessible via screen readers and that commands are easy to learn and use, to provide a user-friendly experience for all users, including those with disabilities.

UIS 01.03 Implement Feedback Mechanism
As a developer, I want to implement simple commands that allow users to provide feedback directly through Discord chat, enhancing the ability to quickly gather and respond to user inputs.

1.5. Design Decisions

The decision to use Discord as the platform for the bot involves several design choices and trade-offs. Discord offers a large robust infrastructure and a large user base, providing a platform for the bot to operate within. It’s API allows for seamless integration and interaction with the messaging feature. However with the use of the Discord API this limits the bots may functionality with challenges like message length restrictions and rate limits, which may impact the bot's performance and scalability. Additionally Discords policies could be largely limiting.

2. User Interface Layouts

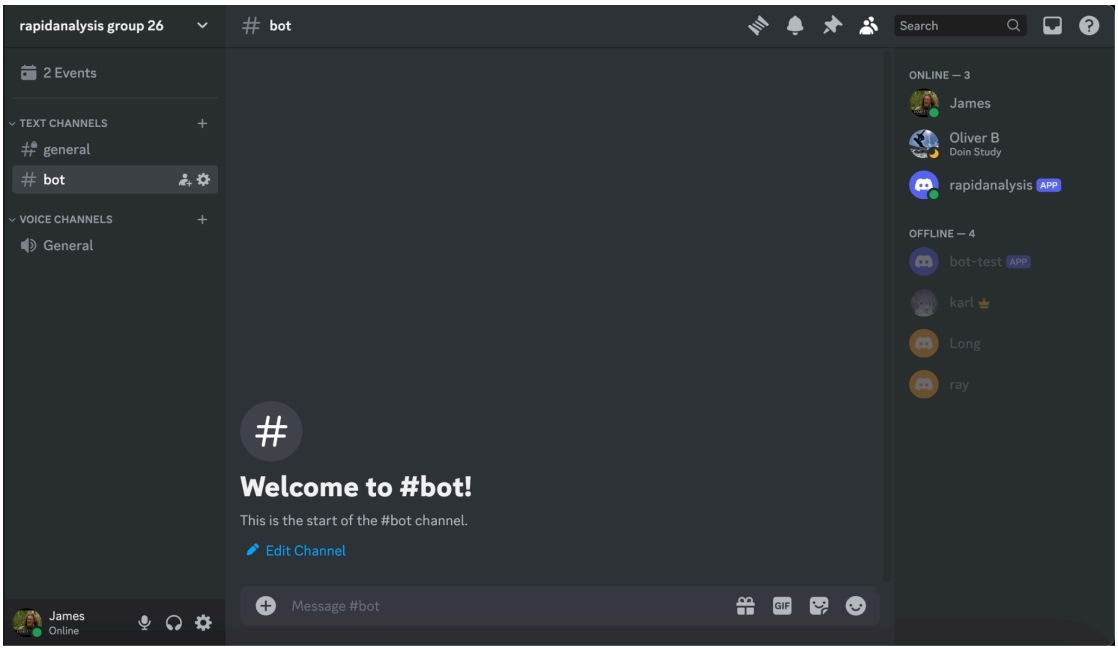

Fig 2: A snapshot of a discord channel, ‘bot’, on a discord server

The user interface presented is the classic Discord server interface, which is characterised by its user-friendly and intuitive design. The layout typically features a column on the far left that lists the server name at the top — in this case, ‘rapidanalysis group 26’ — followed by a list of text and voice channels. Here, there are two text channels: ‘general’ for regular conversations and ‘bot’ for interactions with the bot.

Creating a separate channel for bot queries, like the ‘bot’ channel seen here, is highly recommended to keep bot interactions organised. By creating a separate channel specifically for bot interactions, users can issue commands and receive summaries without cluttering other channels where different discussions are taking place.

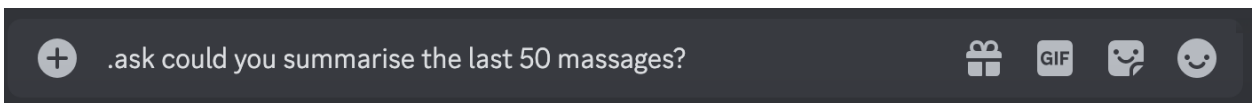

Fig 3: A snapshot of the chat text field with an example command (Beta version)

In the beta phase, interaction with the bot is initiated by typing '.ask' followed by your desired command. For the final release, to enhance the user experience: typing '/' into the chat text field will trigger a drop-down table displaying a range of commands, including 'summarise' and 'preference', for easy access and efficiency.

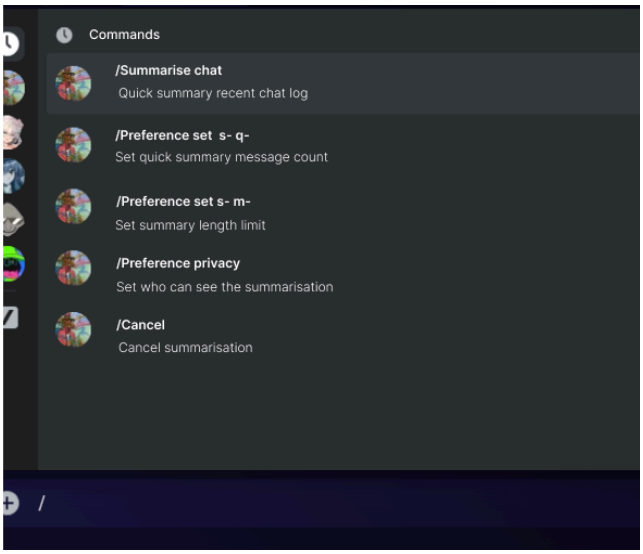

Fig 4: A working concept of drop-down table displaying a range of commands

A response from the bot will be a highlighted reply to its corresponding request message.

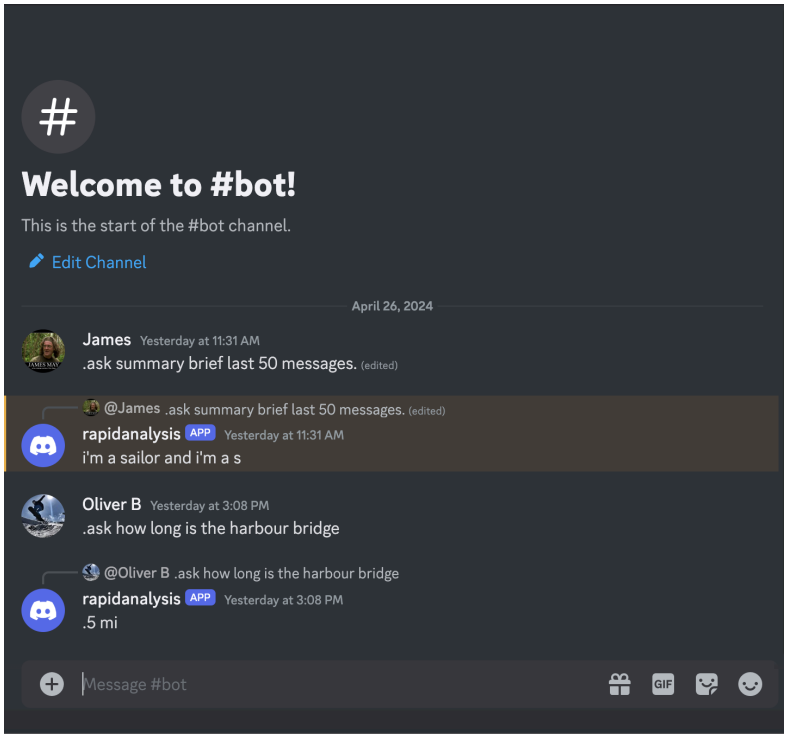

Fig 5: An example of the bot's responses to user requests

3. Window Navigation Diagram

Users can access the bot on any channel on their discord servers. Here, the user is accessing the bot in a bot-dedicated channel on a discord server.

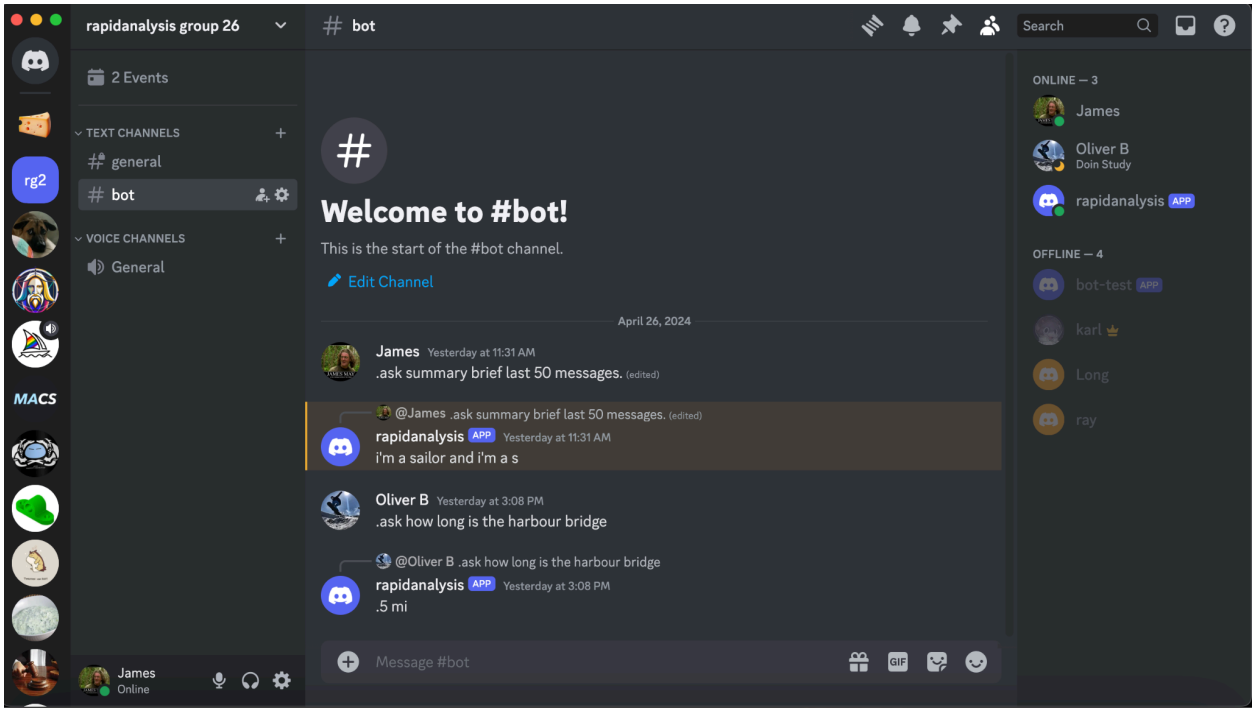

Fig 1:

To pass a request to the bot, users can type in their message text field, '.ask' and follow it with a desired command such as 'summarise'. In a final version, users can type '/' in their message text field to summon the bot's drop-down table menu. Where users can select various types of command.

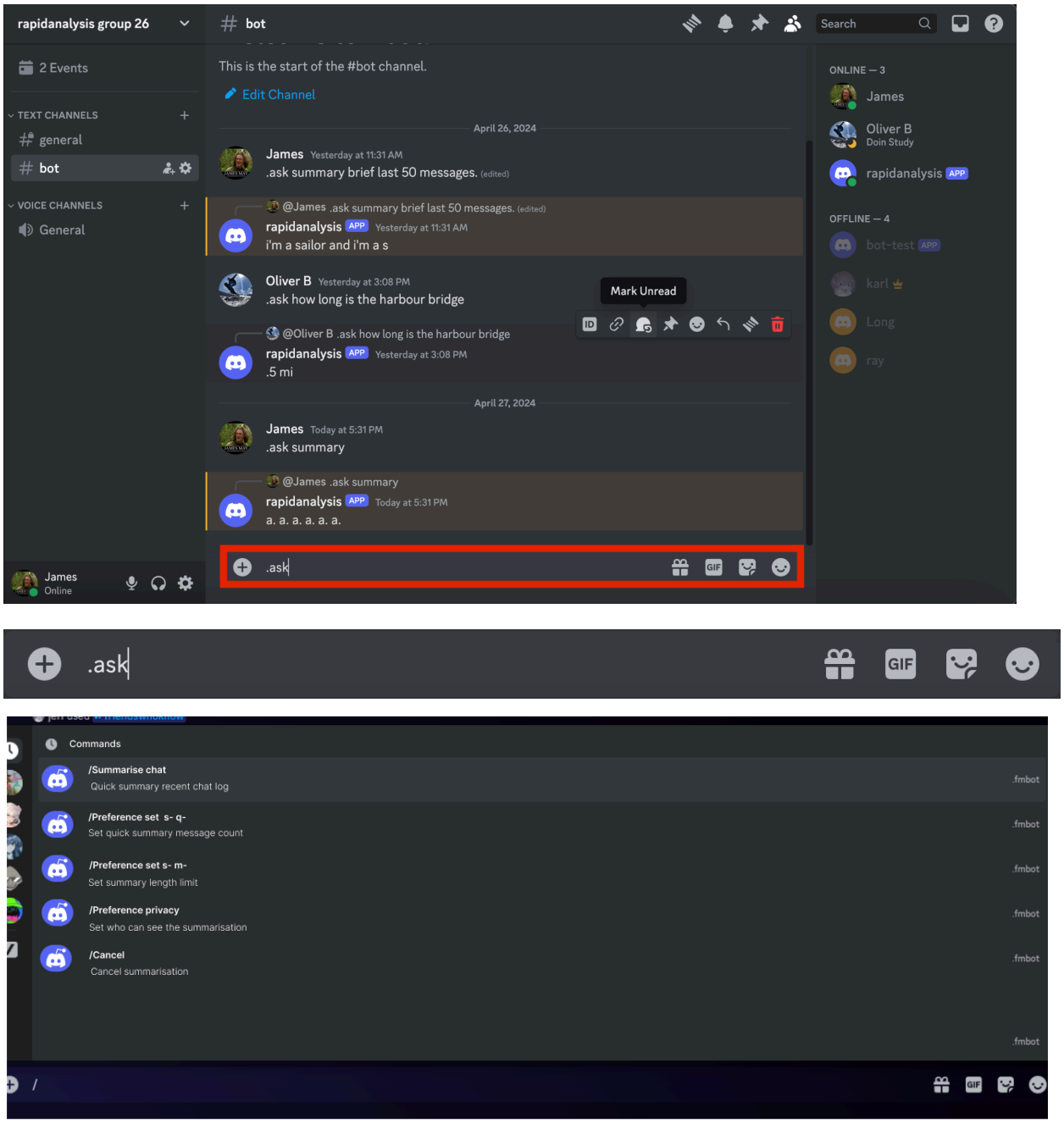

Fig 2: This figure contains multiple snapshots of the screen. The first snapshot shows the text field where users can communicate with the bot in the highlighted area. Second and third snapshot demonstrate the contrast between the beta and the working concept of the final version of the bot summoning method.

To submit your request message, simply press ‘enter’ on your keyboard. The bot will do a simple preprocessing process, then it will send an API request to Rapid Analysis Large Language model API endpoint to generate a response. When the response generation process

is completed, the response will be displayed as a reply to the corresponding user request message.

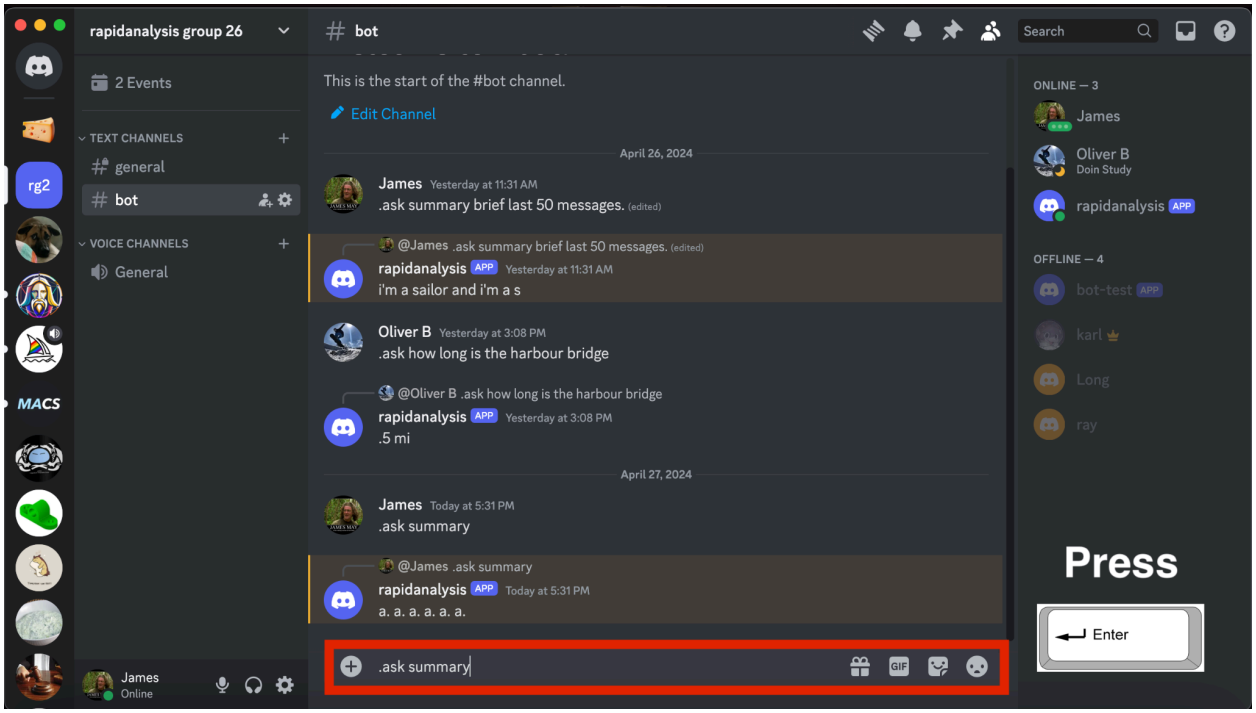

Fig 3: A snapshot of a screen with the input text field highlighted, and a prompt in the bottom right that includes the text 'press' and an image of the 'enter' key, indicating the button to use.

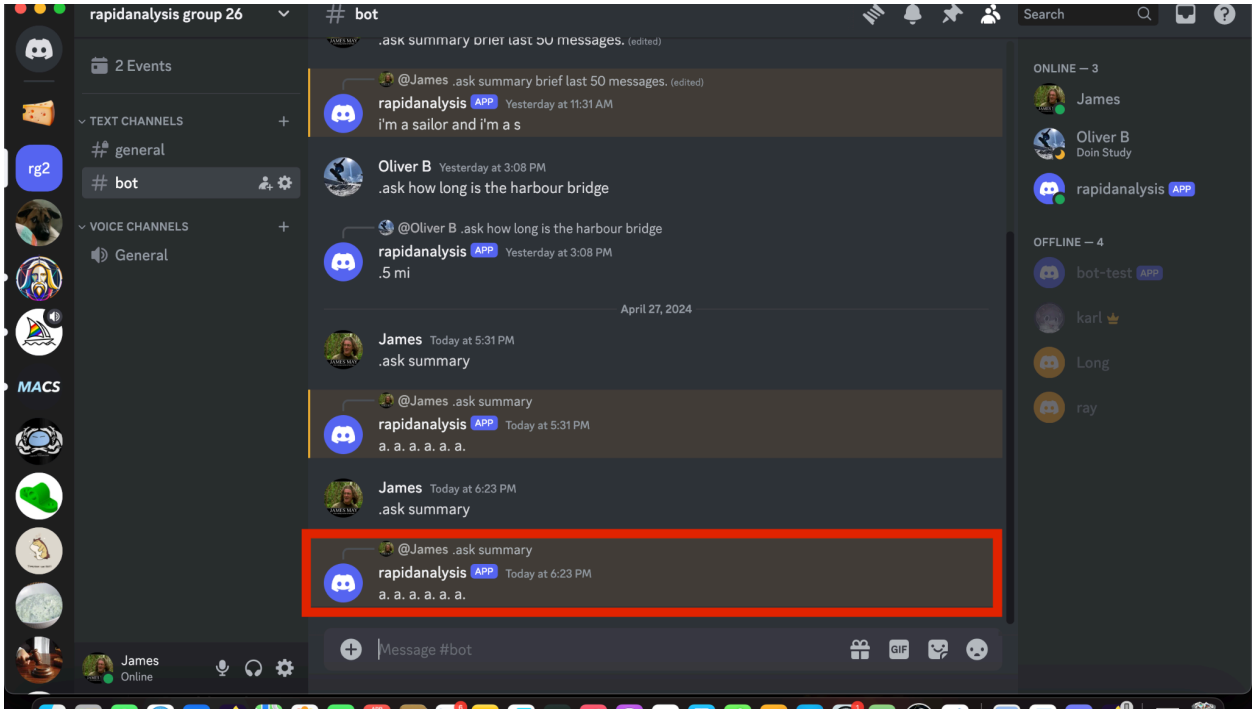

Fig 4: A snapshot of a screen with the bot response highlighted

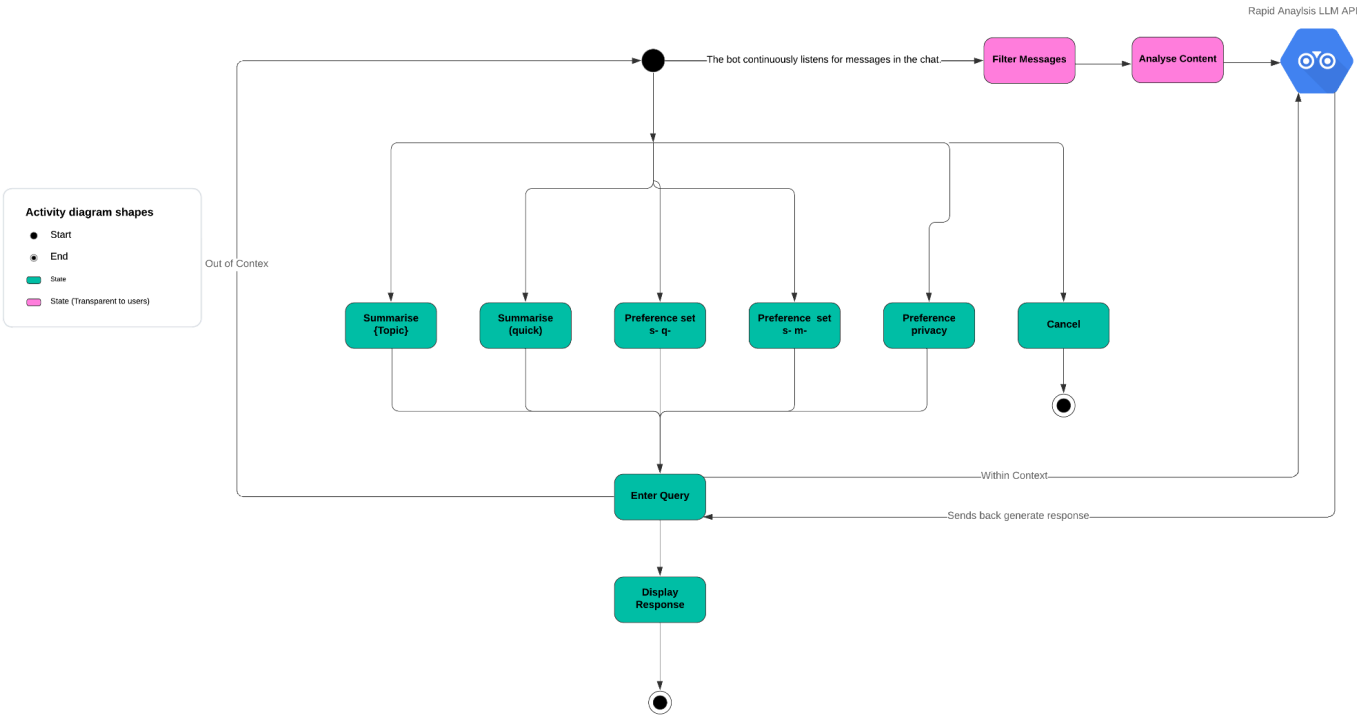

Fig 5: The activity diagram of Discord Text Summarisation Bot

Finally, having seen responses to summary commands in figure 4. Let's take a look at how the bot potentially responds to other commands such as 'preference' and 'cancel'.

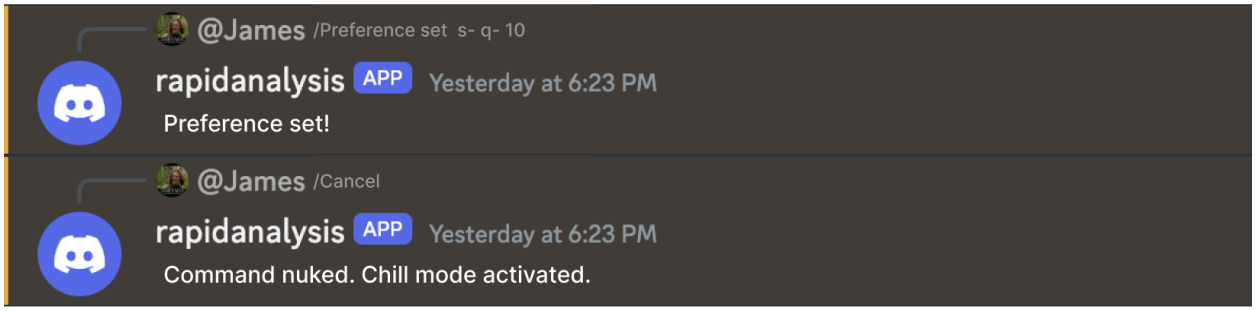

Fig 6: A snapshot of potential bot' responses to 'preference' and 'cancel' command.

4. Data Definitions

Data Field Name	Data Type	Description	Example
userID	varchar	Discord's unique user ID for a specific user	594677238327148545
apiKey	varchar	A user's RapidAnalysis API key	4UZNWQeSynOm8ob9aX79Ywf5nafYLC9NK17zsAIX7ltyT2KuSX
userPrefs	varchar	JSON object containing user preferences	`{summarisePercentage: 0.5}`

5. Class Diagram

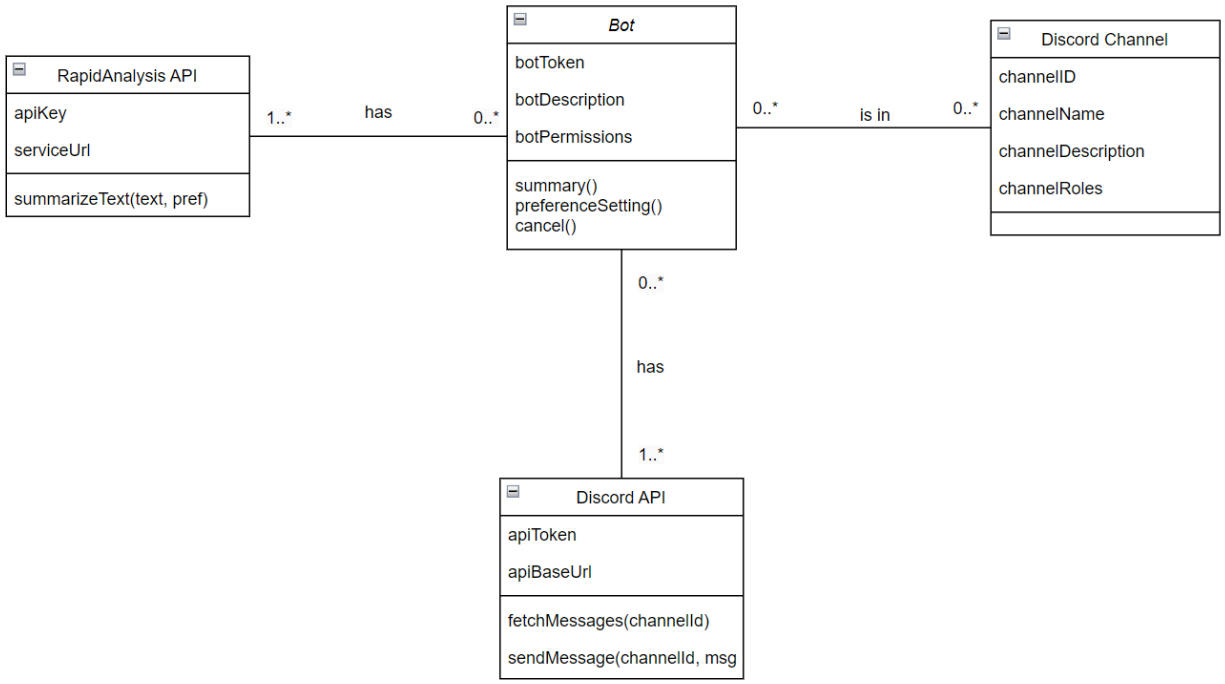

Fig 1: Class Diagram

6. State Diagrams

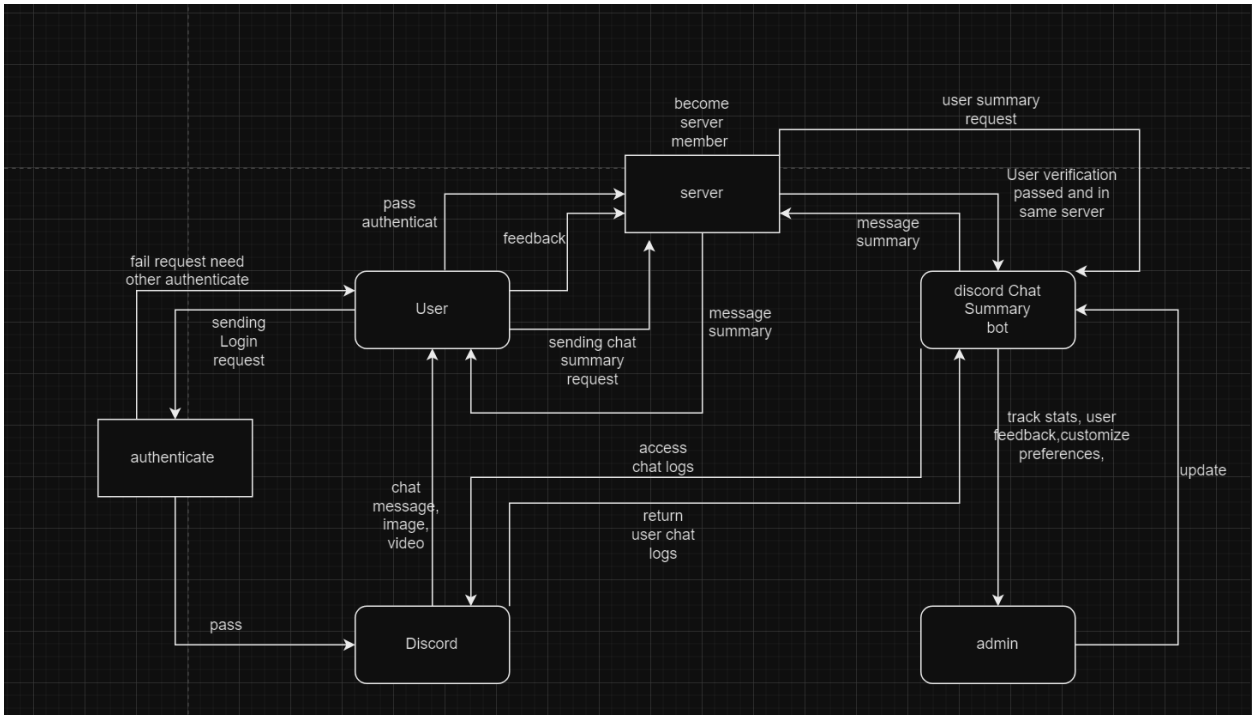

Fig 1: Structure diagram

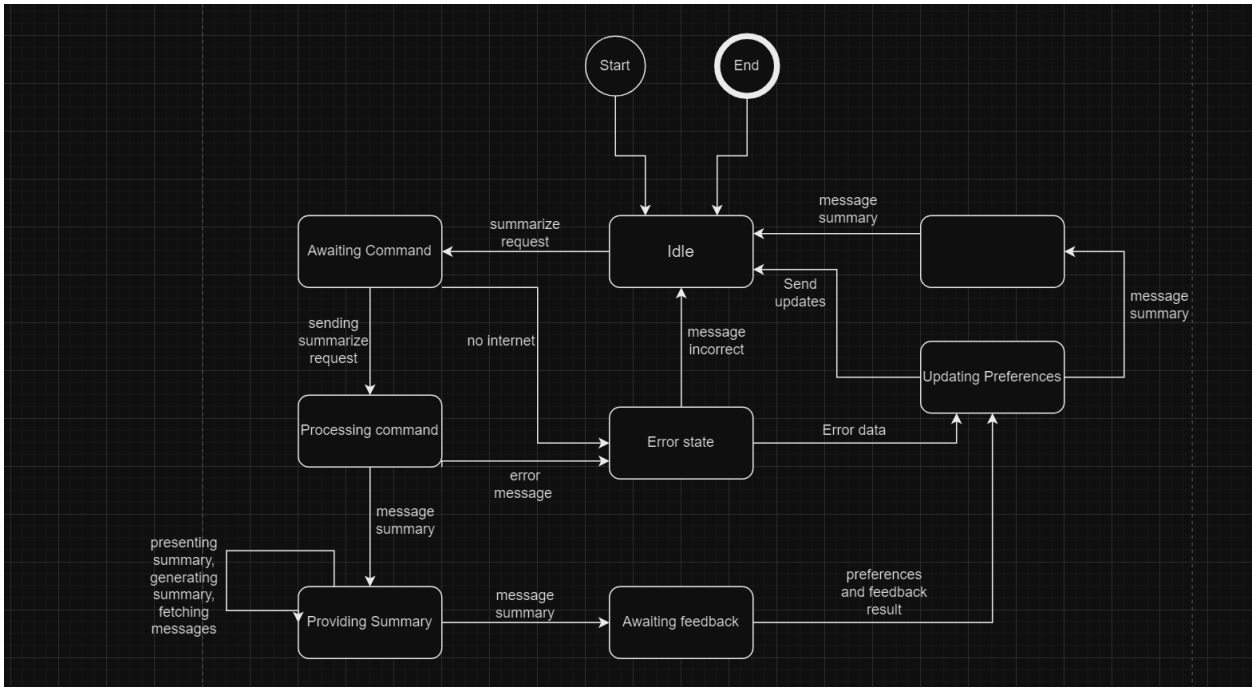

Fig 2: State diagram

7. Sequence Diagram

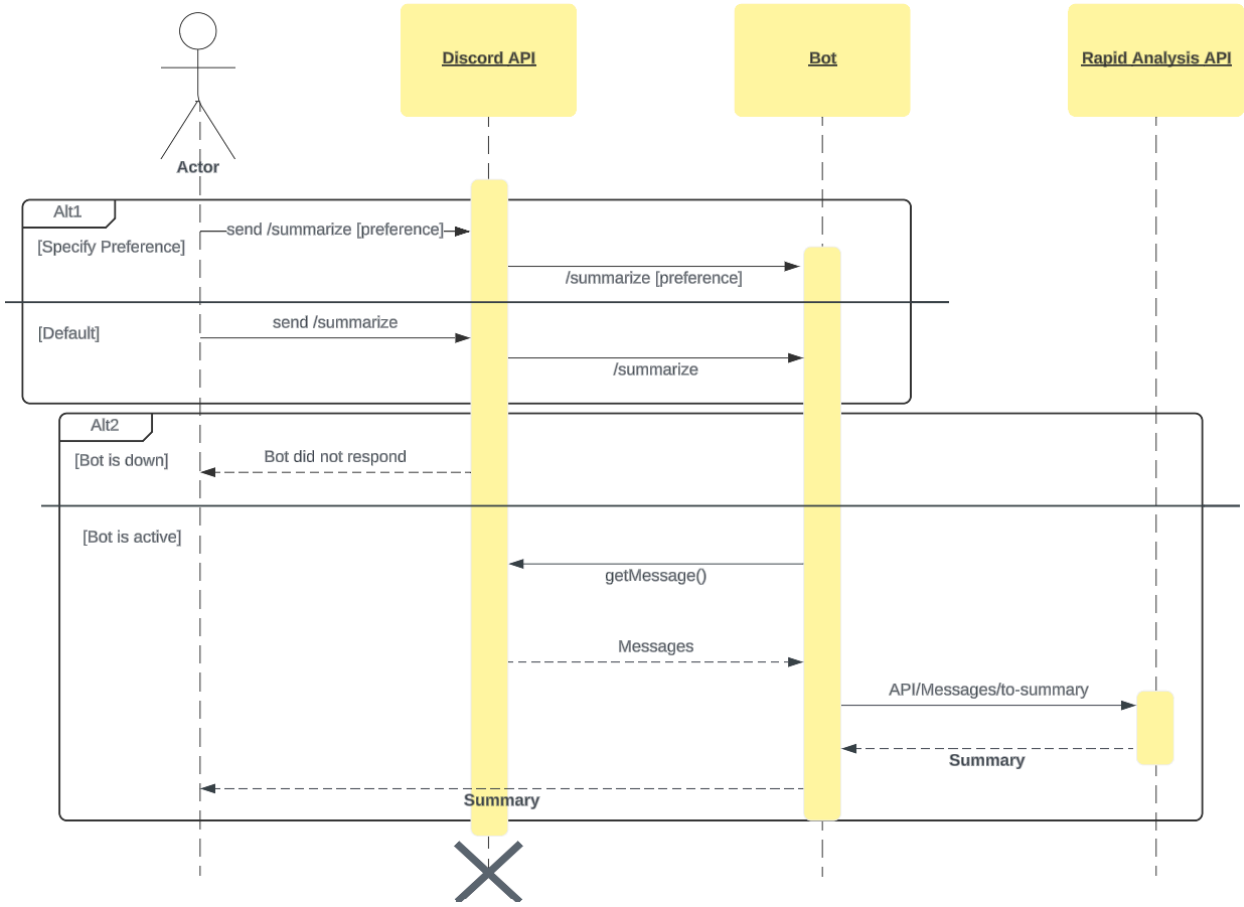

Fig 1: Use Case 1 Sequence Diagram

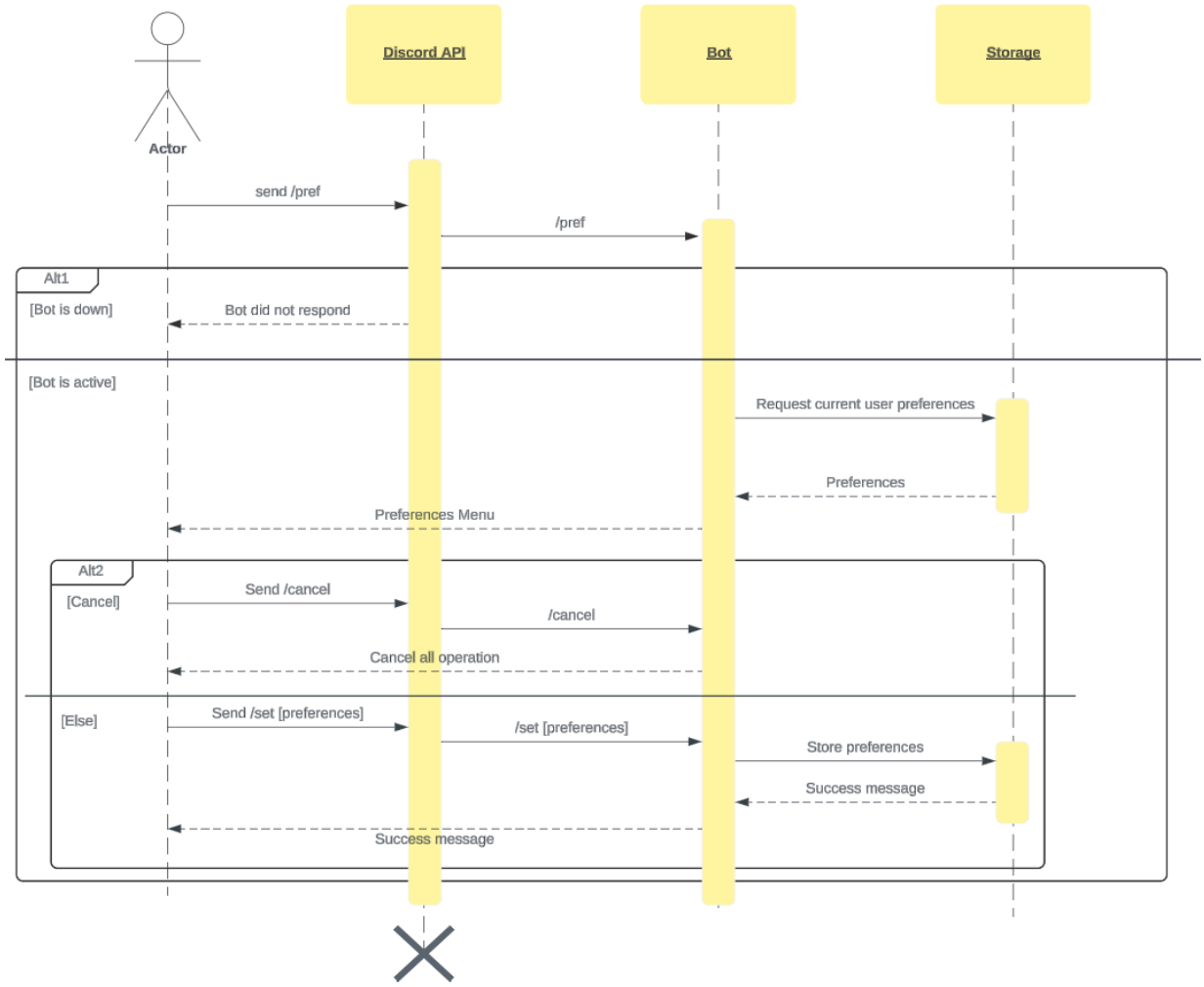

Fig 2: Use Case 2 Sequence Diagram

Prototype

Prototype:

Users can access the bot on any channel on their discord servers. Here, the user is accessing the bot in a bot-dedicated channel on a discord server.

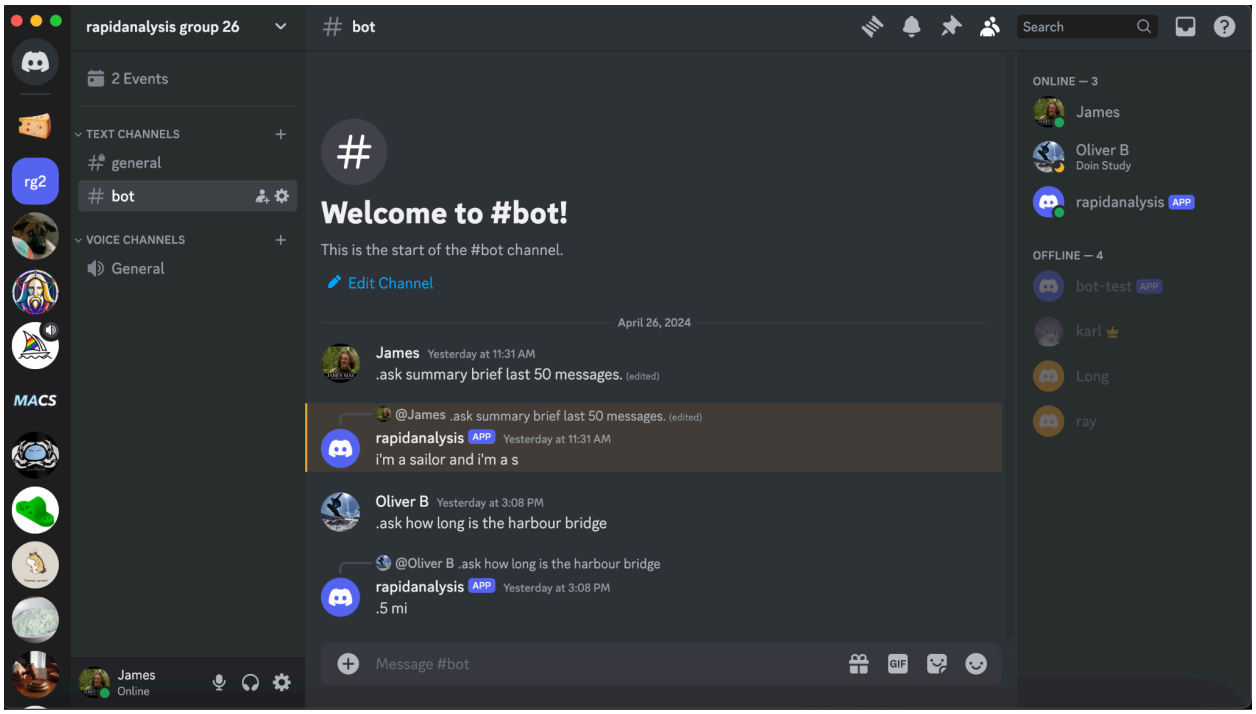

Fig 1:

To pass a request to the bot, users can type in their message text field, '.ask' and follow it with a desired command such as 'summarise'. In a final version, users can type '/' in their message text field to summon the bot's drop-down table menu. Where users can select various types of command.

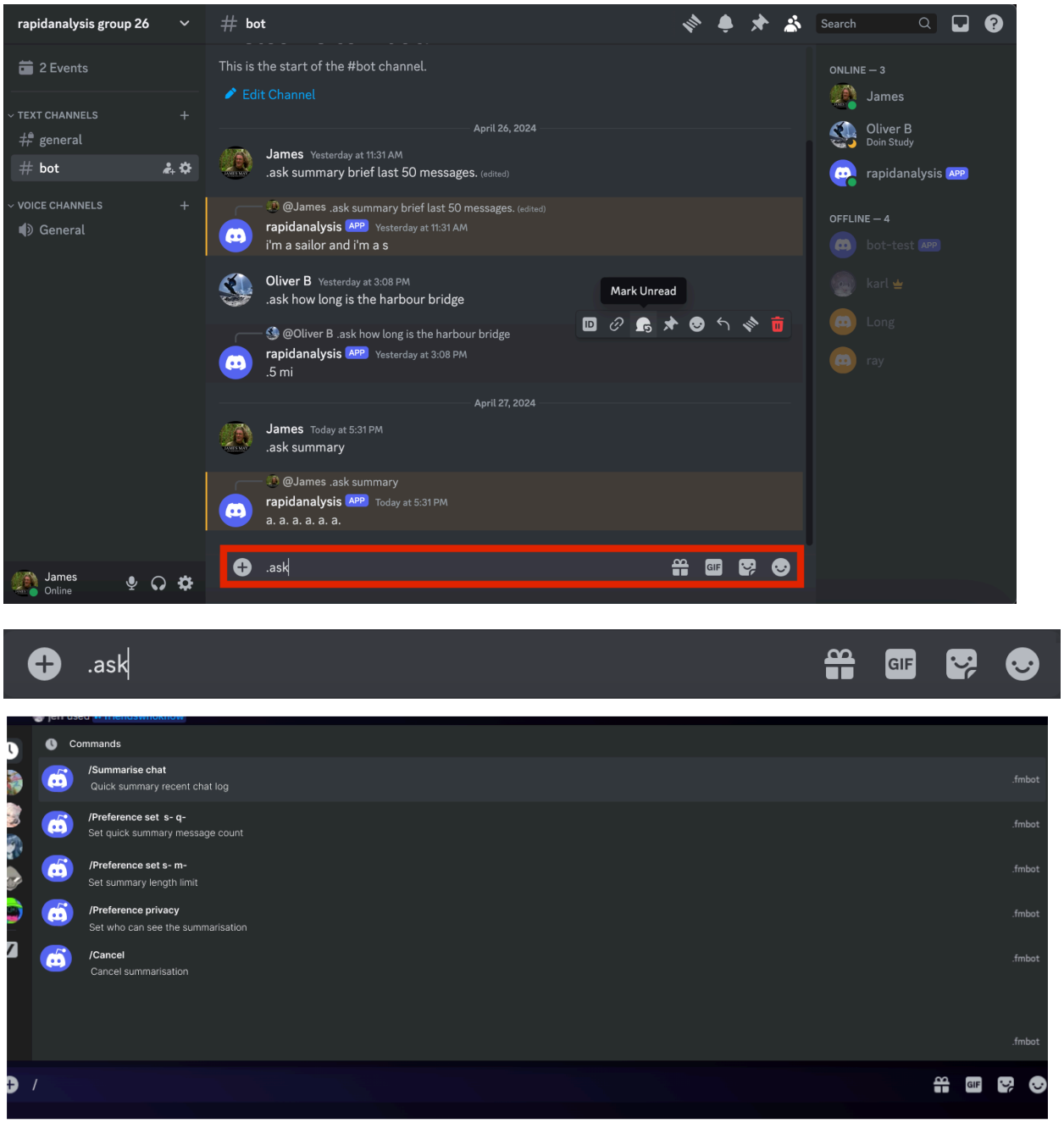

Fig 2: This figure contains multiple snapshots of the screen. The first snapshot shows the text field where users can communicate with the bot in the highlighted area. Second and third snapshot demonstrate the contrast between the beta and the working concept of the final version of the bot summoning method.

To submit your request message, simply press ‘enter’ on your keyboard. The bot will do a simple preprocessing process, then it will send an API request to Rapid Analysis Large Language model API endpoint to generate a response. When the response generation process

is completed, the response will be displayed as a reply to the corresponding user request message.

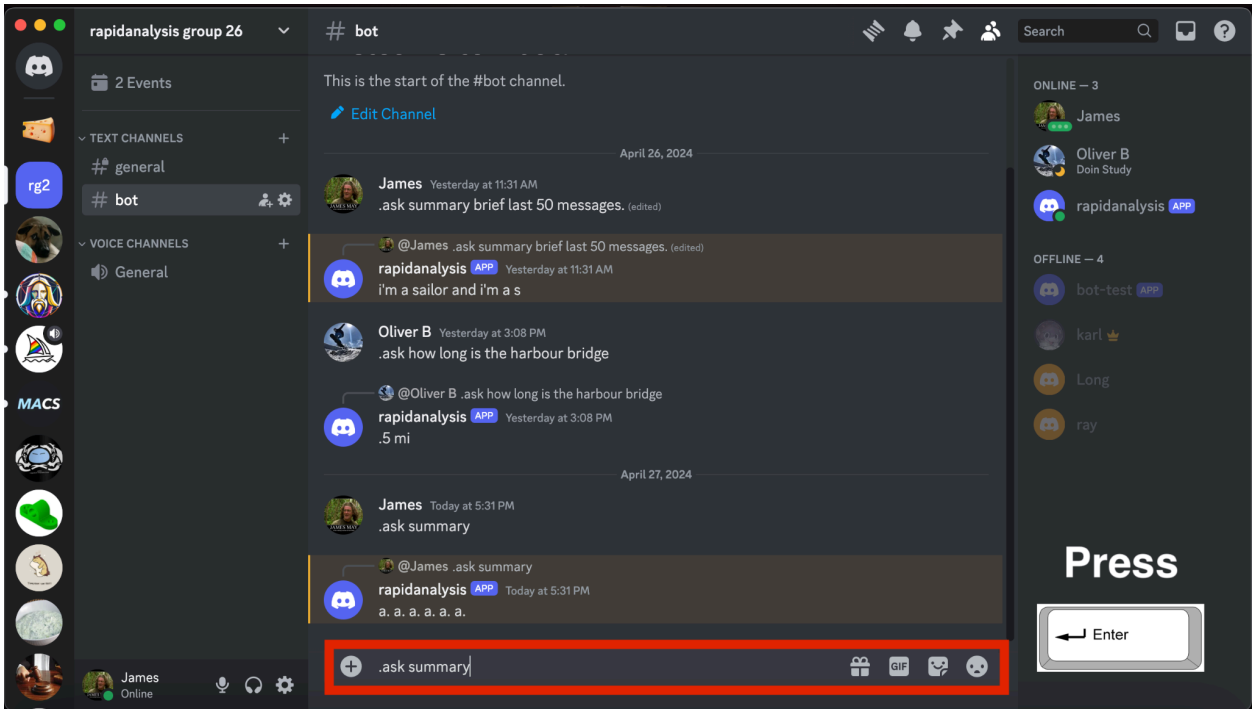

Fig 3: A snapshot of a screen with the input text field highlighted, and a prompt in the bottom right that includes the text 'press' and an image of the 'enter' key, indicating the button to use.

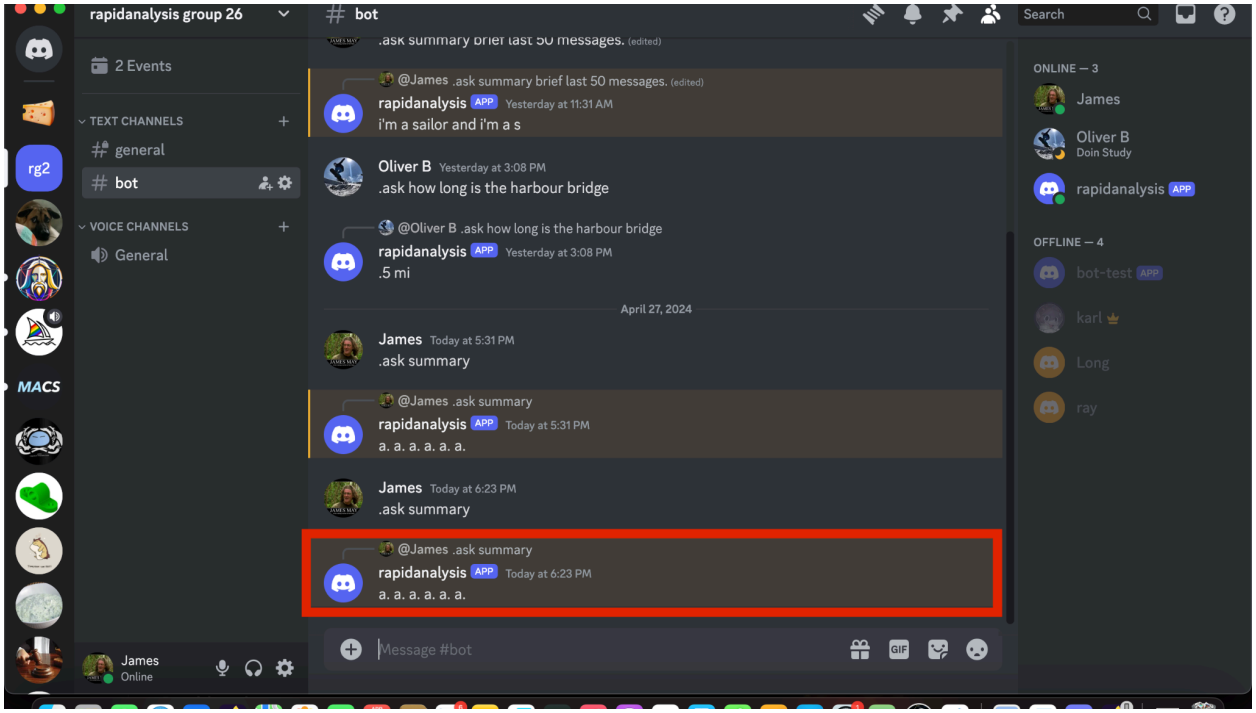

Fig 4: A snapshot of a screen with the bot response highlighted

Sponsor Meeting, Feedback and Response

Feedback: The sponsor complimented our works and the prototype. He said the prototype meets the requirements and is nice. Firstly, the bot is functional. Secondly, it communicates with RapidAnalysis API and Discord API quickly and smoothly. Thirdly, the documentation and diagrams are clear and easy to understand. Overall, he is very happy with the prototype and our works. In addition, we asked for suggestions on Testing documentation. He suggested adding some specific test cases and some methods to improve our testing documentation.
Response: We made the Testing documentation according to the sponsor suggestions. We tried our best to make our testing documentation clear, detailed but easy to understand. Additionally, we added extra details into functionalities that he complimented.

Testing

1. Test Plan

1.1. Introduction

The purpose of this test plan is to outline the approach that is taken for testing the Discord bot designed to perform chat summarisations within Discord. The Test Plan outlines the approach, objectives, schedule, and resources needed to finalise the functionality of the Discord Bot. Furthermore, the plan includes various testing stages, including component, integration, system, performance, and acceptance testing.

1.2. Test Items

1.2.1. Component

Each component within the system will be individually checked with a clearly defined test case specification. A component test is performed in a black box testing environment which allows the developer to validate the behaviour of the components without accessing their internal structures.
Components to be tested Include:

- Chat Summarisation
- User Interaction (customisation and preferences)

1.2.2. Integration /API test

The integration test is used to verify the bot integration with external APIs, specifically the RapidAnalysis API, to ensure seamless communication and data exchange. This test is a white box test, as the test requires knowledge of the authentication and authorisation system and in place to verify it is correctly operating.
The test objectives include:

- Authentication and authorisation
- API request handling
- Data formatting and encoding

1.2.3. System

System testing represents a comprehensive stage of which the aim is to very all aspects of the system's functionality. During this phase, each specific requirement is thoroughly checked across all components of the system. System testing occurs at the final stages of implementation to ensure that the completed software system complies with all specified requirements.

This test is performed in a white-box environment, allowing for an examination of the internal operation of the system. It includes assessments such as the performance of database management to ensure compliance with requirements.
The objectives of system testing include:

- Validating the system's functional and non-functional requirements.
- Assessing the system's reliability, and usability under realistic conditions.
- Identifying and resolving any defects or discrepancies between expected and actual outcomes.

1.2.4. Performance

Performance testing involves evaluating various metrics of software performance. The only metric included in this project's performance requirements is response time however other metrics like throughput and resource utilisation can be included in future versions. The test will assess the system for its maximum return time.

1.2.5. Acceptance

Acceptance testing is scheduled after completing the system testing. This phase aims to confirm the discord bot meets the acceptance criteria defined by the project's sponsor from Rapid Analysis. The following assessments will be performed in this stage:

- Compatibility Testing: Verification of the bot's compatibility with different Discord client applications (web, desktop, mobile) and platforms.
- Test Scenario Execution: Execution of all pre-existing tests performed in earlier stages of the test plan.
- User Experience Assessment: Evaluation of the bot's user interface, responsiveness, and overall user experience. Feedback will be collected from stakeholders to identify any usability issues.

1.3. Resources

1.3.1. Testing Team

All developers will be responsible for executing the test plan, documenting test results, and reporting defects.

1.3.2. Test Environment

The test environment is a private Discord server channel. The prototype versions of the Rapid Analysis Bot have been implemented in this server for testing replacing the “/” command with a “.” command.

1.3.3. Test Documentation

Test Documentation includes the Test Plan and Test Case Specifications. Within the Test Case Specifications, there are relevant test data sets, including sample chat logs and user interactions.

1.4. Schedule

	Task Name	Duration	Start	End	Work
1	Test Plan	22 days	29/03/2024	29/04/2024	2 hrs
2	**Testing Del 1: Component Tests**	**30 days**	**5/04/2024**	**16/05/2024**	**8 hrs**
3	Chat Summarisation	16 days	5/04/2024	26/04/2024	2 hrs
4	User Interaction	2 days	26/04/2024	29/04/2024	2 hrs
5	Error Handling	6 days	26/04/2024	5/05/2024	4 hrs
6	**Testing Del 2: API Integration Tests**	**14 days**	**29/04/2024**	**16/05/2024**	**5 hrs**
7	Authentication and Authorisation	5 days	29/04/2024	3/05/2024	1 hr
8	API request handling	10 days	29/04/2024	10/05/2024	2 hrs
9	Data formatting	14 days	29/04/2024	16/05/2024	2 hrs
10	**Testing Del 3: System Tests**	**14 days**	**29/04/2024**	**16/05/2024**	**19 hrs**
11	Functional Requirements	14 days	29/04/2024	16/05/2024	8 hrs
12	Integration Tests	14 days	29/04/2024	16/05/2024	5 hrs
13	Reliability and Usability	14 days	29/04/2024	16/05/2024	2 hrs
14	Defect correction	14 days	29/04/2024	16/05/2024	4 hrs
15	**Testing Del 4: Performance Tests**	**14 days**	**29/04/2024**	**16/05/2024**	**2 hrs**
16	Response Time	14 days	29/04/2024	16/05/2024	2 hrs
17	**Testing Del 5: Acceptance Tests**	**11 days**	**16/05/2024**	**30/05/2024**	**9 hrs**
18	Compatibility Test	4 days	16/05/2024	21/05/2024	4 hrs
19	User Experience Assessment	4 days	16/05/2024	21/05/2024	4 hrs
20	Handover Requirements Met	11 days	16/05/2024	30/05/2024	1 hr

Fig 1: Project Testing Schedule

2. Test Case Specifications

Application	Rapid Discord API bot	**Tester Name**	Oliver Bush
Release Version	Version 1.0	**Test Date**	28/04/24

Test Information			
Requirement	R1. Chat Summarisation	**Test Identifier**	TC01
Sub-requirements	R.1.1. Summary Generation R.1.2. Summary Presentation R.1.3. Authentication and Authorisation R.1.4. Integration with Discord Features R.1.5. Interactive Summary Requests		
Test Description	Type - Component Strategy - Black Box Tool - rapidanalysis Discord Bot Interface on RapidAnalysis channel		
Result Details			
User Input	**Expected Results**	**Actual Results**	**Pass/Fail**
User enters ".summarise" and send	The bot should clearly summarise the latest default number of messages	Bot does not respond	Fail
User enters ".summarise 100"	The bot provides a summary of the last 100 messages	Bot does not respond	Fail
User enters ".summarise 18:00 20:00"	The bot provides a summary of the chat messages between 6:00PM and 8:00PM	Bot does not respond	Fail
User enters ".summarise 2 h"	The bot provides a summary of the chat	Bot does not respond	Fail

	messages from the last 2 hours		

Application	Rapid Discord API bot	**Tester Name**	Oliver Bush
Release Version	Version 1.0	**Test Date**	28/04/24

Test Information			
Requirement	R.2. User Customisation and Preferences	**Test Identifier**	TC02
Sub-requirements	R.2.1. Interface for Customisation R.2.2. Preference Persistence R.2.3. Server-Level Default Settings		
Test Description	Type - Component Strategy - Black Box Tool - rapidanalysis Discord Bot Interface on RapidAnalysis channel		
Result Details			
User Input	**Expected Results**	**Actual Results**	**Pass/Fail**
User enters ".preferences"	The bot shows all the preferences that the user can set	Bot does not respond	Fail
User enters ".preferences" then ".cancel'	No results are returned	Bot does not respond	Fail
User enters ".preferences" then selects ".defaultNumberOfLatestMessages"	The bot displays the previous Default number of latest messages and a description of what it is	Bot does not respond	Fail
User enters "/preferences" then selects ".defaultNumberOfLatestMessages" then "20"	The bot displays the previous Default number of latest messages and a description of what it is	Bot does not respond	Fail

User enters “/preferences” then selects “.summaryLengthLimit”	The bot displays the previous Summary length limit and a description of what it is	Bot does not respond	Fail
User enters “/preferences” then selects Summary privacy	The bot displays the previous Summary privacy setting and a description of what it is	Bot does not respond	Fail

Application	Rapid Discord API bot	**Tester Name**	Oliver Bush
Release Version	Version 1.0	**Test Date**	28/04/24

Test Information			
Requirement	R.3. Error Handling and Recovery	**Test Identifier**	TC03
Sub-requirements	R.3.1. Error Detection and Logging R.3.2. Error Recovery		
Test Description	Type - Component Strategy - Black Box Tool - Discord Bot Interface on RapidAnalysis channel		
Result Details			
User Input	**Expected Results**	**Actual Results**	**Pass/Fail**
User enters “/summarise 101”	The bot should send an error message	Bot does not respond	Fail
User enters “/preferences” then selects Default number of latest messages then “101”	The bot displays the previous Default number of latest messages and a description of what it is	Bot does not respond	Fail

Application	Rapid Discord API bot	**Tester Name**	Oliver Bush
Release Version	Version 1.0	**Test Date**	28/04/24

Test Information			
Requirement	R.4. Operation within Discord	**Test Identifier**	TC04
Test Description	Type - Integration Strategy - White Box Tool - rapidanalysis Discord Bot Interface on RapidAnalysis channel		
Result Details			
User Input	**Expected Results**	**Actual Results**	**Pass/Fail**
Perform standard .ask command to see response	Discord bot responds in the text channel	Discord bot responds in the text channel	Pass

Application	Rapid Discord API bot	**Tester Name**	Oliver Bush
Release Version	Version 1.0	**Test Date**	

Test Information			
Requirement	R.5. Rapid Analysis Integration	**Test Identifier**	TC05
Test Description	Type - Integration Strategy - White Box Tool - rapidanalysis Discord Bot Interface on RapidAnalysis channel		
Result Details			
User Input	**Expected Results**	**Actual Results**	**Pass/Fail**
Perform standard .ask command to see response	Bot replies with standard text to chat response	Bot replies with standard text to chat response	Pass