# *COMP3850 Project Deliverable Certificate*

| | |
|---|---|
| Name of Deliverable | *Deliverable4 - Updated D3, plus user/training manual* |
| Date Submitted | *16 / 05 / 2024* |
| Project Group Number | *Team 25* |
| Rubric stream being followed for this deliverable (highlight one)<br>*Note: the feasibility study has the same rubric for all streams.* | *SOFTWARE Rubric*<br>*GAMES Rubric*<br>*CYBERSECURITY Rubric*<br>*DATA SCIENCE 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)* |
|---|---|---|---|---|
| YL | Lee | Yasmin | 46366377 | YLee |
| FG | Gelwyn | Faith | 46421114 | FGelwyn |
| DL | Lee | Donghyun | 45773602 | Lee |
| BB | Bostami | MD N S Baizid | 46128980 | baizid |
| IW | Weston | Isabel | 46570616 | IWeston |
| SR | Rasmussen | Sebastian | 47223421 | Sebastian R |

| Performed by *(Student Names)* | Duration *(hrs)* | Complexity *(L, M, H)* | Name of task | Checked by *(Initials)* |
|---|---|---|---|---|
| All group members | 3.5 | M | Team meeting | IW, DL, FG, BB, SR, YL |
| Yasmin Lee | 2 | M | Revise Use case diagram and use cases, Overleaf formatting | DL,IW, FG ,BB, SR |
| Yasmin Lee | 0.5 | L | Revise Gantt Chart in Project Plan | DL,IW, FG ,BB, SR |
| Faith Gelwyn | 2 | L | User manual | DL,IW, BB, SR, YL |
| Baizid Bostami | 2.5 | M | Updated Prototype document, User Interface Overview section, Contacting Support section, | DL,IW, FG , SR, YL |
| Donghyun Lee | 2 | M | User manual - FAQ & Troubleshooting | DL,IW, FG, SR,BB, YL |
| Donghyun Lee | 1.5 | L | Set up Jira & Contact with Client | DL,BB, YL, SR |
| Donghyun Lee | 0.5 | L | Documentation | DL, YL, SR |
| Sebastian Rasmussen | 0.5 | L | Prototype Video for Client | DL,IW, FG, BB, YL |
| Sebastian Rasmussen | 1 | M | Programming Prototype | DL,IW, FG,BB, YL |
| Sebastian Rasmussen | 0.5 | L | Overleaf formatting | DL,BB, YL, FG |
| Sebastian Rasmussen | 1.5 | L | User Manual | DL,BB, YL, FG |
| Isabel Weston | 2.5 | | User manual, updated use cases, Overleaf formatting | DL,IW, FG, BB, YL, SR |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| Yasmin Lee Total | 6 | | | |
| Faith Gelwyn Total | 5.5 | | | |
| Baizid Bostami Total | 6 | | | |
| Donghyun Lee Total | 7.5 | | | |
| Sebastian Rasmussen Total | 3.5 | | | |
| Isabel Weston Total | 6 | | | |
| Team Total | 34 | | | |


# Infobyte

Team 25

## Project Plan

Slack app with RapidAnalysis API

Version 3.0 16.05.2024

## Contents

Revision History

| Name | Date | Reason for Change | Revision |
|---|---|---|---|
| FG | 17.04.24 | Updating Change Management 4.1 Requirements and Scope Change feedback | 1.1 |
| FG | 18.04.24 | Adding Handover Requirements | 1.2 |
| FG | 19.04.24 | Revised Project Schedule | 1.3 |
| BB | 28.04.24 | Approval from Project Manager | 2.0 |
| DL | 16.05.24 | Updating Figure 2(Gantt chart) | 3.0 |

# 1 Statement of Purpose

The purpose of this document is to present a plan for the development and deployment of an AI chatbot in Slack using the RapidAnalysis API.

The team, InfoByte, will develop a Slack app for the client, RapidAnalysis. The chatbot will address common issues in team communication over Slack by summarising long messages and giving users intelligent, context-aware responses to their prompts. The final product will be a user-friendly solution that is monetisable by RapidAnalysis and extensible by other developers in the future.

This project plan will outline the management of risks, resources, changes, and quality of the project, as well as the schedule for the project, including a timeline of deliverables, overview of tasks with specific examples taken from Jira, resources allocated and assumptions for the duration of the project. It will assume a familiarity with basic acronyms involved in the project, used in other documents like the Team Manual.

# 2 Risk Management

Risk management is essential for discovering, assessing, and managing possible threats to a project's objectives and success and informing decision-making. A three-step risk management strategy has been established for this project:

1. Risk identification
2. A matrix that categorises risks according to type, asset, severity, likelihood, and impact
3. Mitigation strategies

## 2.1 Identification

Risks to the project have been identified in the following areas.

### 2.1.1 Technology

1. Dependency on third-party tools, namely APIs and libraries.
2. Data breaches, unauthorised access, and compromised user information
3. Delays in development, poor quality code.
4. Integrating RapidAnalysis API with Slack could lead to complexity.
5. User acceptance testing (UAT) issues can lead to severe functional and non-functional problems.
6. A lack of scalability can slow down the system
7. After development, the API may face legal and compliance risks as there are already several other chatbots in the market, e.g., ChatGPT, Microsoft Copilot, and Google Gemini.
8. As the team members are University students, the team might have skill gaps.
9. Ample data is not provided to the language model, so the LLM can't provide enough response.

10. Chatbot infrastructure and maintenance can cause bot downtime.
11. Users can input spam or abusive material to the chatbot.

#### 2.1.2 Human

1. Loss of contact.
2. Inability to attend the meeting.
3. Withdrawal from the project.
4. Conflict within the team.
5. Incomplete tasks due to lack of ability.
6. Incomplete tasks due to lack of motivation.
7. Incomplete tasks due to illness.

#### 2.1.3 Organisation

1. Unmet expectations can make the stakeholders dissatisfied.
2. Insufficient resources can hinder project success.
3. Resistance to change may hamper the progress of the project.

#### 2.1.4 Estimation

1. We miss deliverables in development.
2. We are behind schedule according to the timeline.
3. We continually fail to arrange meetings.

## 2.2 Matrix

| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 1 | IT RISK ASSESSMENT MATRIX | | | | | | | | |
| 2 | REF/ID | RISK TYPE | ASSET | RISK DESCRIPTION | RISK SEVERITY | RISK LIKELIHOOD | INTERNAL IMPACT | USER IMPACT | TRIGGER |
| 3 | R1 | Illness | INSIDER | Any team members are ill | ACCEPTABLE | POSSIBLE | LOW | MEDIUM | Delays in production, interruption in project |
| 4 | R2 | Technology | EXTERNAL | Dependency on third party tools e.g API, libraries etc | TOLERABLE | IMPROBABLE | MEDIUM | MEDIUM | Delays in production, |
| 5 | R3 | Technology | INSIDER | Data breaches, Unauthorised access and compromised user information | INTOLERABLE | POSSIBLE | EXTREME | EXTREME | Reputation damage, legal consequences, and loss of user trust |
| 6 | R4 | Technology | INSIDER | Delays in development, poor quality code | TOLERABLE | PROBABLE | MEDIUM | MEDIUM | Rushed work or lack of adherence to coding standards. |
| 7 | R5 | Health | EXTERNAL | Aggressive deadlines can make stress and potential burnout | TOLERABLE | IMPROBABLE | LOW | LOW | Unrealistic project planning. |
| 8 | R6 | Business | TRUSTED INSIDER | Unmet Expectations can make the stakeholder disstatisfied | ACCEPTABLE | IMPROBABLE | MEDIUM | LOW | Inaccurate estimations. |
| 9 | R7 | Health | INSIDER | Low productivity of team members can occur delays and missed deadlines | TOLERABLE | IMPROBABLE | MEDIUM | LOW | Delays, employee burnout. |
| 10 | R8 | Business | EXTERNAL | Lack of user adoption can lead to disappointment | TOLERABLE | IMPROBABLE | MEDIUM | MEDIUM | Poor user experience or inadequate marketing. |
| 11 | R9 | Health | INSIDER | Conflict between team members | UNDESIRABLE | IMPROBABLE | MEDIUM | LOW | Unwanted decisitions, fall of team structure |
| 12 | R10 | Technology | INSIDER | While integrating with the RapidAPI to slack, it cloud lead to complexity | TOLERABLE | POSSIBLE | HIGH | LOW | Complex APIs or legacy systems |
| 13 | R11 | Technology | INSIDER | User Acceptance Testing (UAT) Issues can lead to sevear functional and non functional problems | TOLERABLE | PROBABLE | HIGH | LOW | Misaligned expectations during UAT. |
| 14 | R12 | Technology | INSIDER | Lack of Scalability can slow down the system | TOLERABLE | POSSIBLE | HIGH | MEDIUM | Underestimating future user load. |
| 15 | R13 | Technology | EXTERNAL | After development, the api can face Legal and Compliance Risks as there are several other chatboat in the market already. E.g Chatgpt, Microsoft Copilot, Google gemini | UNDESIRABLE | POSSIBLE | HIGH | HIGH | Violation of data protection laws. |
| 16 | R14 | Technology | INSIDER | As the team members are Univeristy students, there might be skill gaps in the team | ACCEPTABLE | IMPROBABLE | MEDIUM | LOW | Unexpected team departures. |
| 17 | R15 | Technology | INSIDER | Ample data is not provided to the language model, so the LLM can't provide enough response | TOLERABLE | POSSIBLE | HIGH | HIGH | Insufficient user training materials. |
| 18 | R16 | Technology | INSIDER | Chatbot infrastructure and maintenance can cause downtime to the bot | TOLERABLE | POSSIBLE | HIGH | HIGH | Hardware or software failures, system outages |
| 19 | R17 | Technology | INSIDER | User can input spam or abusive material to the chatbot | TOLERABLE | POSSIBLE | HIGH | HIGH | Malicious actors exploiting the chatbot for spamming or other abusive behavior. |
| 20 | R18 | Time | INSIDER | Missing deliverables in development | TOLERABLE | POSSIBLE | HIGH | LOW | The deliverables are not submiited in right time |
| 21 | R19 | Time | INSIDER | We are behind schedule according to the timeline | TOLERABLE | IMPROBABLE | HIGH | MEDIUM | We can't keep up with the pace and continuously missing timeframe |
| 22 | R20 | Time | INSIDER | failing to arrange team meetings. | UNDESIRABLE | IMPROBABLE | HIGH | LOW | No team communication, missing deadlines |

Figure 1: Risk assessment matrix with scores for severity, likelihood and impact assigned to each risk item, as well as identified potential triggers.

## 2.3 Mitigation

The following measures will mitigate the risks to the project in each identified area.

### 2.3.1 Technology

1. Dependency on third-party tools: Discuss the best solution with the team and client before proceeding, like using alternative libraries.
2. Data breaches, unauthorised access, and compromised user information:
   (a) Implement robust security practices.
   (b) Conduct regular vulnerability assessments and follow best practices.
   (c) Do not keep user information in the cloud or any other storage.
3. Delayed development with poor quality code:
   (a) Frequent code testing.
   (b) Resolve bugs and logical errors promptly.
   (c) Establish coding standards and best practices.
4. Complexity of integrating RapidAnalysis API with Slack:
   (a) Thoroughly understand integration requirements.
   (b) Test integrations early and often.
5. UAT issues:
   (a) Involve stakeholders in UAT planning.
   (b) Clearly define acceptance criteria.
6. Lack of scalability:
   (a) Design for scalability from the outset.
   (b) Regularly assess performance metrics.
7. Legal and compliance risks with other chatbots:
   (a) Consult legal experts team and stakeholders.
   (b) Ensure compliance with relevant regulations.
8. Skill gaps:
   (a) Ensure proper training for team members.
   (b) Document critical knowledge and everyone's strengths and skills to align with the project.
9. Unsatisfactory chatbot response:
   (a) Develop user-friendly documentation.
   (b) Conduct training sessions for end-users.

10. Chatbot downtime:
    (a) Implement a high-availability infrastructure for the chatbot.
    (b) Develop a robust monitoring and alerting system for potential issues.
    (c) Have a backup plan in place to restore functionality in case of failures.
11. Spam or abusive input:
    (a) Implement spam filters and user authentication mechanisms.
    (b) Define clear rules and guidelines for chatbot interaction.
    (c) Monitor chatbot activity for suspicious behavior and take corrective actions as needed.

#### 2.3.2 Human

1. Illness, loss of contact, or inattendance: Another team member will get the handover and continue the task.
2. Withdrawal: The project plan changes and other team members will be allocated based on their strengths to fill the position.
3. Conflict: All team members will discuss the point of conflict, and the project manager will lead the discussion to find the solution.
4. Incomplete tasks due to lack of ability or motivation:
    (a) We are avoiding any excessive workloads in our project. We also have well-paced project planning.
    (b) Per our team structure and guidelines, if any team member cannot complete their assigned task, they must inform the team. Subsequently, the team collaborates with the project manager to find a solution and ensure the completion of any remaining tasks.

#### 2.3.3 Organisation

1. Unmet stakeholder expectations: We have established clear communication channels and set realistic goals to address this issue. We also follow weekly updates and transparent reporting to manage expectations effectively.
2. Insufficient resources: Our approach involves comprehensively evaluating resources and prioritising tasks according to their importance for the project's success. In our feasibility Report, we have also discussed and recorded various alternative strategies to address the issue.
3. Resistance to changes: We have involved our stakeholders in the change process, providing them with the rationale behind changes and how they will benefit the project. We will also ensure training and support within our team to facilitate smoother transitions.

#### 2.3.4 Estimation

1. Missed deliverables:
   (a) Conduct regular reviews to track progress and identify any deviations from the plan.
   (b) Ensure well-defined requirements to minimise misunderstandings and scope creep.
   (c) Have backup plans in place to address unexpected delays.
2. Behind schedule:
   (a) Set realistic timelines based on data and team capacity.
   (b) Allocate extra time for unforeseen issues or unexpected delays.
   (c) Keep stakeholders informed about any schedule changes promptly.
3. Failure to arrange meetings:
   (a) Set reminders to remember to arrange and attend Discord meetings.
   (b) Clearly define the purpose and agenda for each meeting.
   (c) Ensure all participants commit to attend the scheduled weekly meeting.

# 3 Resource Management

Resource management encompasses the efficient allocation, scheduling, and continuous monitoring of various resources.

## 3.1 Human

The team consists of six students under the consultancy name "InfoByte" working collaboratively on the project. Roles and responsibilities are allocated to team members effectively based on their strengths, skills, and availability. Team members are given one of three formal roles defined in the team manual: project manager, lead developer, and developer. Role rotation ensures that everyone experiences all three sets of responsibilities. Workloads are balanced by evenly assigning Jira tasks to team members.

## 3.2 Technology

There are no specific hardware resources other than standard development machines (laptops or desktops) belonging to the team members which have sufficient processing power, memory, and storage for the project. The system requirements for using Slack are MacOS 11 or above, Windows 10 version 21H2 or above, Ubuntu LTS releases 20.04 or above, or Red Hat Enterprise Linux 8.0 or above.

The software stack includes the following tools, which have been thoughtfully selected to ensure effective resource management across the team:

- IDE: Microsoft Visual Studio Code (VSCode), a versatile code editor with features like syntax highlighting, debugging, and extensions for enhanced productivity.
- Version control: Git for code, with GitHub to host the repository, and Google Docs and Overleaf to save and iterate on notes, drafts, and completed documents.

- Communication channels: Discord for messaging and meetings, and Slack to test the chatbot features.
- APIs: RapidAnalysis API for LLM capabilities and Slack API for Slack app development.
- Project management: Jira project for assigning tasks and viewing progress, using Kanban board and timeline views.

### 3.3 Information

The team uses formal documentation and the Discord server to save relevant information relating to the project. This includes the RapidAnalysis website, which demonstrates how the client expects their AI smart utilities and APIs to appear to users and developers, and important pages on Slack app development and AWS.

### 3.4 Time

The PM schedules weekly client meetings to discuss project progress and address client-specific requirements.

Weekly team meetings are a priority to understand upcoming deliverables and allocate tasks. These are held once or twice a week, considering the availability of team members.

Jira provides a timeline view of deliverables and subtasks. The PM prioritises tasks to ensure timely completion.

### 3.5 Finances

The RapidAnalysis API and Slack are currently free for the developers and users.

## 4 Change Management

Change management for this project involves

1. managing requirements and scope change, and
2. version control for code and documents.

### 4.1 Requirements and scope change

The team recognises and understands that change in the project is inevitable and these changes must be communicated due to the potential impacts for the client and the development team. For these reasons, it is crucial for the team to document and clearly define the project requirements.

The Project Manager (PM) should regularly review and validate requirements with the client, allowing the team to understand the expectations of the client and also preemptively identifying any discrepancies or evolving needs.

Finally, the team expects that any changes are promptly communicated, whether this is during the weekly team meeting or in Discord messages, viewable to the rest of the team. This allows the team to openly discuss the changes and provide feedback, seek clarification and re-evaluate the work collaboratively. Furthermore private, closed-door decisions contradict the change management plan outlined in the team manual.

Communicating the changes between the client and the team is the most important aspect of requirement and scope change. The changes must be clear to all team members to facilitate the work required to adopt the new changes.

### 4.2 Version control

Developers are expected to document code changes accurately using Git, assisted by the LD who reviews pull requests and merges them into the GitHub project once accepted.

All deliverable documents are completed in collaborative environments: Google Docs and Overleaf. This ensures that team members iterate on the latest version and are able to view changes or restore an older version in case of accidental change.

## 5 Quality Management

The quality of this project depends on the communication within the team, dictated by the team manual with clearly defined roles and responsibilities, and with the client, whose feedback and expectations are received in weekly emails and/or meetings with the PM. Specific quality monitoring and assurance strategies will include the following.

1. Error checking mechanisms to ensure the chatbot operates smoothly and without errors, which could involve validating user inputs, checking for null and corner values, and handling unexpected responses from APIs or databases.
2. Performance optimisation by monitoring stress under different loads to ensure the chatbot can handle many simultaneous users or requests.
3. Comprehensive, up-to-date documentation to facilitate collaboration among developers, understand the system, and help in troubleshooting and maintenance.
4. Regular testing to ensure the chatbot works as expected, including unit testing individual components, integration testing the whole system, and user acceptance testing to validate the chatbot from the perspective of the end user.

Tester fatigue should be taken into account when the developers assign Jira tasks to themselves and communicated with the LD, as per the communication policy for all technical tasks.

## 6 Project Schedule

The project and sprints are structured around the deliverables, which are summarised in Table 1. This is reflected in the timeline view of the Jira project in Figure 2.

| **Deliverable** | **Due** |
|---|---|
| D1: Feasibility study and team manual | 07/03/2024 |
| D2: Project plan, requirements document, updated team manual | 28/03/2024 |
| D3: Updated documents and minimum viable product (MVP) | 29/04/2024 |
| D3: PM changeover | 17/04/2024 |
| D4: Updated deliverables and user manual | 16/05/2024 |
| D5: Final report | 30/05/2024 |
| D6: Presentation | 30/05/2024 |
| D7: Final product | 30/05/2024 |
| D8: Exam | TBC |

Table 1: Scheduled deliverables for the project.

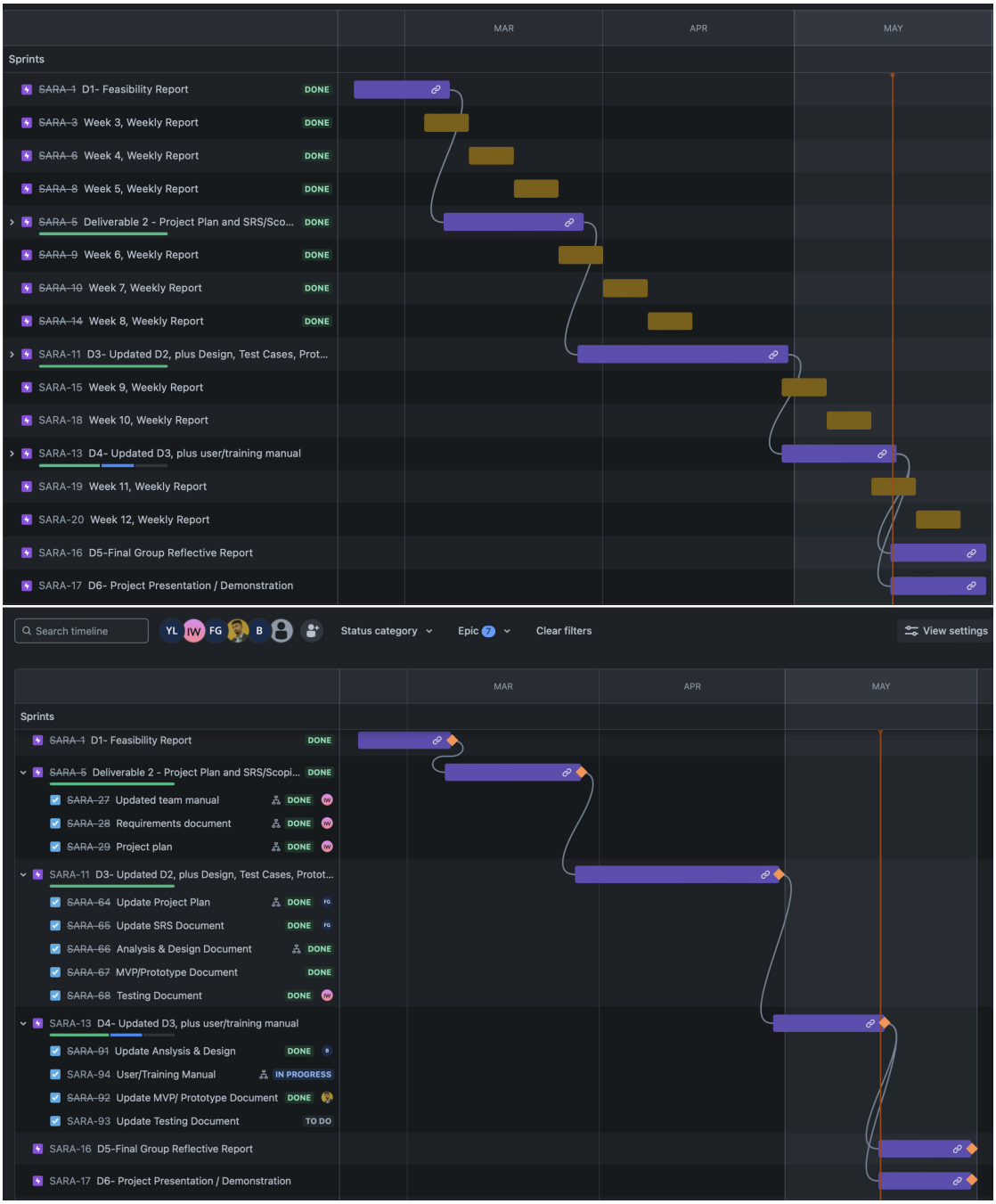



Figure 2: Timeline view of the Jira project, Slack App for RapidAnalysis (abbreviated to SARA). In the first image, the orange bars indicate progress reports which are submitted once every week, excluding the recess. The purple bars represent sprints; clicking on these expands the details of that deliverable. The second image just shows these deliverables and their expanded tasks along with orange diamonds to represent milestones.

## 6.1 Tasks

As an agile team of 6 members with rotating roles in each sprint, a flexible approach to task allocation is crucial. The PM sets up specific issues under the deliverables on Jira; see the examples in Figures 3 and 4. There are also recurring core tasks for each sprint which the team must complete to ensure the development of the project, which are divided into the following areas.

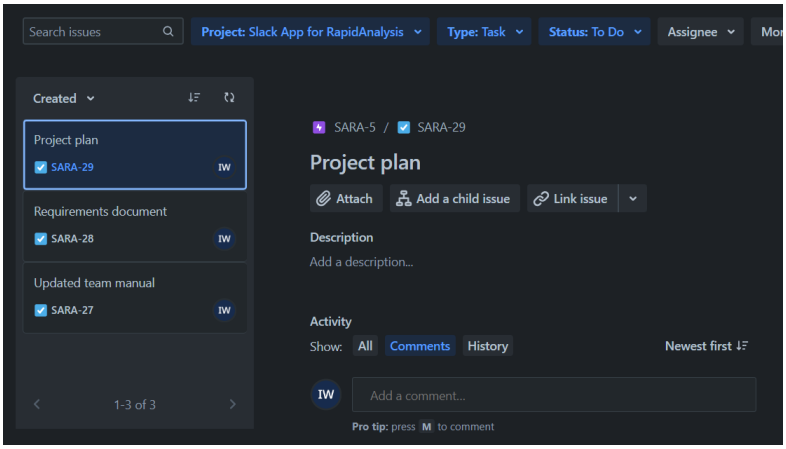



Figure 3: Example of a Jira task. The sidebar to the left shows all tasks listed under D2 (uniquely identified as SARA-5) which have all been assigned to the PM for that sprint (see initials).

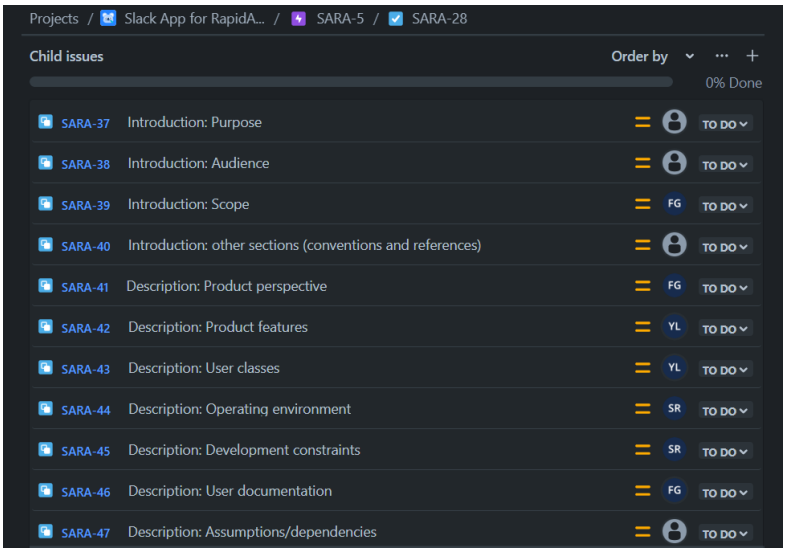



Figure 4: Example of child issues listed under a task. Some team members have already assigned the task to themselves. The completion status can also be updated from "TO DO" to "IN PROGRESS" or "DONE".

### 6.1.1 Planning and Management

At the beginning of each sprint, the designated PM initiates communications with the client over email and arranges a meeting if necessary, and schedules a team meeting based on shared availability.

All team members attend the weekly meeting to collaboratively define sprint goals, share ideas and notes on the upcoming deliverables, and accept Jira tasks set by the PM.

The PM updates and maintains the Jira tasks, and checks that the team is on track to complete tasks for deliverables with input from the LD.

### 6.1.2 Development and Implementation

Each developer will focus on their assigned tasks, which could include building specific chatbot functionalities and integrating the chatbot into Slack.

The LD provides guidance to developers through paired programming, conducting code reviews, approving changes, and overseeing the overall technical direction of the project in accordance with the demands of the PM and, ultimately, the client.

### 6.1.3 Testing and Deployment

All team members can participate in beta testing, unit testing and integration testing for quality assurance. Most importantly, feedback for improvement should be given in a Discord message or Jira issue. The team will collaborate on deployment strategies, including choosing a Slack channel for user testing and establishing a feedback mechanism.

### 6.1.4 Documentation and Support

Throughout the project, everyone will contribute to documenting the development process. This includes deliverable documents like requirements and specifications and the user manual, as well as weekly progress reports. Using text-based communication channels like Discord messaging and Jira issue comments are also a form of documentation and are crucial for team members to support the overall project progress.

### 6.1.5 Updates

Since version 1.0 there has been a Project Manager and Lead Developer changeover as the D3 timeline is the longest of the project. We plan to meet weekly with the client to show multiple iterations of the prototype.

## 6.2 Resources

For the duration of the project, Team 25 is allocated the following key resources.

- 6 team members, assuming no unexpected circumstances like withdrawal from the project, with their own development devices
- Discord server and Jira project accessible to all team members
- GitHub repository provided by the client for developers to work on
- Amazon Web Services provided by the client

### 6.3 Assumptions

In completing the client's desired Slack application with RapidAnalysis AI smart utilities integration, it is assumed that our team will have access to sufficient resources including:

- access to APIs (RapidAnalysis API, Slack API) and their documentation
- hosting services (Amazon Web Services)
- project management resources (Jira)
- repository for version control (GitHub).

It is also assumed that the RapidAnalysis API will adequately facilitate information exchange such that long chat histories can be analysed by LLM.

## 7 Handover Requirements

The client:

- has access to the code, because they own the GitHub repository
- is aware of the programming language and version being used
- will receive permission to access the AWS Lambda service which hosts the code
- will receive all documentation, enabling them to use the project.

## 8 Appendices

1. Risk Management Template by Dan Martin of ClearRisk
2. Document Standard Header Template by InfoByte
3. SRS Template


# Infobyte

Team 25

## Software Requirements Specification

Slack app with RapidAnalysis API

Version 3.0 16.05.2024

## Contents

# Revision history

| Name | Date | Reasons for changes | Version |
|---|---|---|---|
| FG | 18/04/2024 | Adding defintions at first mention throughout documentation | 1.1 |
| FG | 22/04/2024 | Updating Functional Requirements (D2 Feedback) | 1.2 |
| BB | 28/04/2024 | Approval from Project Manager | 2.0 |

# 1 Introduction

## 1.1 Overview

### 1.1.1 Document Conventions

For improved readability, this document uses numbered, bold headings and subheadings, lists, and captioned figures. Other specific conventions include the following.

1. Code is written in monospace format like `code`.
2. Proposed commands for users to access the chatbot in Slack are written as `/command`.
3. The Definitions, Acronyms, and Abbreviations emphasises key terms in bold like **SRS**.

### 1.1.2 Intended Audience and Reading Suggestions

This document is intended for current and future project stakeholders, namely the client, RapidAnalysis, but also other teams or individuals who will continue to expand the scope of the Slack app, develop and test new functionalities, and add documentation. This document will help readers to understand the purpose, features, and functional requirements of the project. The authors have the following suggestions to readers.

- For a high-view understanding of the project, refer to the Introduction and Overall Description, which includes diagrams and hyperlinks to further resources.
- For deeper understanding of the product's functionality, see 3.1 and 3.2. For deeper understanding of the project requirements, see 3.3 Usability Requirements and 3.4 Other Nonfunctional Requirements.
- Read Client Feedback to understand how the expectations of the client informed this document.
- Readers should continuously refer to section 1.4 Definitions, Acronyms and Abbreviations, which contains definitions of specialised terms and acronyms to better understand this document.

## 1.2 Purpose

The client, RapidAnalysis [1], is developing their AI (artificial Intelligence) smart utilities with Machine Learning (ML) endpoints for a wide range of use cases. Currently, users can create free developer accounts and interact with their AI smart utilities with a monthly limit of 5,000 API (Application Programming Interface) calls, but the AI smart utilities will eventually be monetised.

This Software Requirements Specification (SRS) version 1.0 describes the specific project of integrating RapidAnalysis capabilities in a Slack app. It communicates the intention of the project's development team, InfoByte, to other stakeholders, namely the client. It conforms to the ISO/IEC/IEEE 29148:2011 standard [2].

## 1.3 Project Scope

RapidAnalysis is focused on delivering customisable business solutions using their machine learning APIs. For this project InfoByte will be creating an AI summarising chatbot that can be downloaded to Slack user's workspace. Slack, a messaging application for organisation, has a dedicated Marketplace where users can download third-party applications [3]. After installing the application, a user will be able to interact with the AI bot to summarise the channels in their workspace, using RapidAnalysis's machine learning API to do so.

Slack users will be able to save time and confusion by utilising the AI bot. Instead of reading through a backlog of messages in multiple channels, the AI bot will be able to read the channel and give a comprehensive synopsis of each channel when requested. This increases productivity in an organisation, allowing users to spend more time on the actionables rather than backtracking and trying to understand what has been spoken about when they may not have been available.

Having the application available on Slack opens the doors for exposure and potential monetisation of the API for RapidAnalysis. For this project it is intended to have a functioning application that is deployed on the Slack Marketplace, but also to leave it in a state that can be further improved by other developers in the future.

## 1.4 Definitions, Acronyms, and Abbreviations

- **AI** Artificial Intelligence
- **API** Application Programming Interface
- **AWS** Amazon Web Services
- **DM** Direct Message
- **HTTP** HyperText Transfer Protocol
- **HTTPS** HTTP Secure
- **JSON** JavaScript Object Notation
- **LLM** Large Language Model
- **ML** Machine Learning
- **OS** Operating System
- **PEP** Python Enhancement Proposal
- **PM** Private Message
- **REST** Representational State Transfer
- **SRS** Software Requirements Specification

## 1.5 References

[1] RapidAnalysis. "Rapidanalysis." Accessed on March 14, 2024. (Feb. 2024), [Online]. Available: https://rapidanalysis.github.io/.

[2] "ISO/IEC/IEEE international standard - systems and software engineering – life cycle processes –requirements engineering," *ISO/IEC/IEEE 29148:2011(E)*, pp. 1–94, 2011, Accessed on March 14, 2024. DOI: 10.1109/IEEESTD.2011.6146379.

[3] Slack. "Add apps, get work done." Accessed on March 14, 2024. (), [Online]. Available: https://slack.com/intl/en-au/apps.

[4] Slack. "Unlock your productivity potential with Slack Platform." Accessed on March 14, 2024. (Mar. 2024), [Online]. Available: https://api.slack.com/.

# 2 Overall Description

## 2.1 Product Perspective

Previously, RapidAnalysis has created an add-on for Google Sheets using their machine learning API and are looking to expand the applications of their technology. InfoByte has been tasked with implementing the machine learning API with Slack. Slack has its own API for developers to use when creating new applications [4], allowing them to upload the application to their (Slack) own marketplace where users can download the applications and extensions to their workspace.

The goal for this product is to have a summarising AI bot that can be downloaded from the Slack Marketplace. The scope of the application has been reduced to ensure that it will be executed well, leaving the project extendable for other developers to build on.

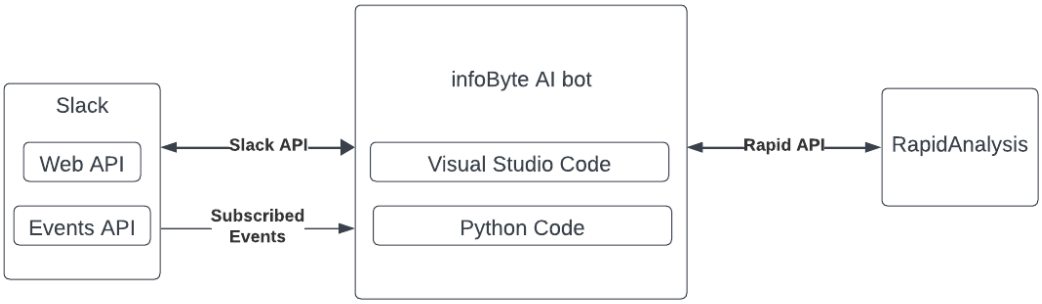



Figure 1: Architecture diagram showing how the chatbot, created with Python in VSCode, will integrate RapidAnalysis AI smart utilities in Slack.

## 2.2 Product Functions

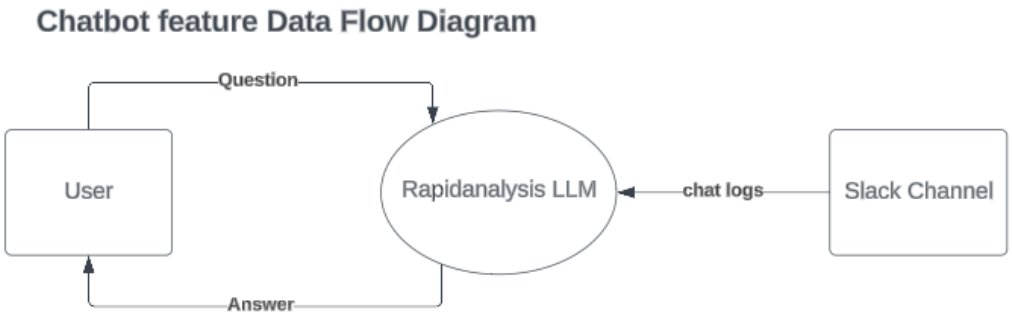



Figure 2: Data flow diagram showing how the user will interact with the chatbot, ultimately the RapidAnalysis LLM (large language model), which receives chat logs from a Slack channel.

Users can ask the chatbot questions about conversations in a Slack channel and it will provide a summary of information relevant to the user request based on the channel logs it has been given; see Figure 2. Developers can expand upon the functionality of the chatbot using the RapidAnalysis API.

## 2.3 User Classes and Characteristics

The target audience of Slack is any organisational employee, whether in small teams to multinational organisations, typically in "office" jobs or work-from-home roles. Freelancers, independent contractors, non-profit organisations, and student organisations may also use Slack.

Most of the users will be employees of organisations who already use Slack for communication within teams every day during working hours. They only need a basic understanding of Slack to download the app from the Slack App Directory and access the chatbot.

1. As an employee, I would like to be able to automate message summarisation and search for information more easily in Slack channels so that I don't have to waste time finding it.
2. As a team leader, I would like to ensure that my team is communicating clearly and efficiently, and maximise their understanding of my messages.
3. As a business leader, I would like to use the summaries of conversations to make informed business decisions.

Some of the user base will be developers who would like to expand upon the app. This will be less common and less frequent. They will have a high level of technical expertise to be able to develop the app further.

## 2.4 Operating Environment

The software's only dependencies are Python version 3.12, and the support for HTTP (HyperText Transfer Protocol) to interact with the Slack and RapidAnalysis API. Thus the software requires the same hardware requirements as Python, where the OS (operating system) can be Microsoft Windows, MacOS, or Linux.

The Python application will operate with 2 packages, and their dependencies, which are (in the form `'package_name' == 'version'`):

- `slack-bolt == 1.18.1`, for interacting with the Slack API.
- `requests == 2.31.0`, for creating HTTP requests, specifically for interacting with the RapidAnalysis API.

The software must run on the AWS (Amazon Web Services) Lambda Service, and thus must coexist with all its components.

## 2.5 User Documentation

The software will be delivered with a user manual and help guides. As the software is being launched on the Slack Marketplace, there is a dedicated space for a description and links under 'Learn more & support' to help users with the app. These links lead the user to the developer's website where there is documentation on how to use their software. RapidAnalysis has a dedicated section on their website for their Slack App information to be updated on the completion of the project and will be linked in the Slack App Directory description.

## 2.6 Assumptions and Dependencies

The team assumes the following for this project.

1. Access to Slack API and corresponding documentation
2. Access to RapidAnalysis API and corresponding documentation
3. AWS Lambda Service for hosting the Python application, provided by the client
4. Jira for project management
5. GitHub repository

# 3 Requirements

## 3.1 Functional Requirements

### 3.1.1 Interact with RapidAnalysis API

**Description and Priority**

**Priority**: High.

Interacting with APIs is crucial for the application's core functionality. Without this feature, the application would lack the ability to integrate external services and functionalities, severely limiting its usefulness and value to users. Therefore, ensuring the successful implementation of API interactions is of utmost importance.

**Stimulus/Responses Sequences**

1. User requests interaction with API through application interface.
2. Application sends API request with relevant parameters.
3. API processes request and returns response to application.
4. Application receives response and presents it to the user.

**Functional Requirements**

The application will provide an interface for users to initiate interactions with external APIs.

Users will be able to specify parameters and data required for API requests. The application will handle authentication and authorisation mechanisms required for securely accessing external APIs.

1. Error handling: The application will appropriately handle errors and exceptions returned by the APIs, providing meaningful feedback to the user.
2. Input validation: Ensure that user-provided data for API requests is validated to prevent invalid or malicious inputs.
3. Logging: The application will log API requests and responses for debugging and auditing purposes.
4. Asynchronous processing: The application should support asynchronous API calls to prevent blocking user interactions during long-running operations.

REQ-1: The application will support various HTTP methods (GET, POST, PUT, DELETE) for interacting with APIs.

REQ-2: API responses will be parsed and formatted appropriately before presenting to the user.

REQ-3: Implement rate limiting mechanisms to prevent abuse and ensure fair usage of APIs.

REQ-4: Ensure compatibility with different API versions and specifications to accommodate changes and updates by API providers.

### 3.1.2 Summarising Messages and Asking Questions

**Description and Priority**

**Priority**: Medium.

This feature enables the system to summarise messages and ask questions based on the conversation context. It plays a pivotal role in enhancing communication efficiency and comprehension within the application.

**Justification**

The ability to summarise messages and ask relevant questions enhances user engagement and comprehension within the application. By providing concise summaries and prompting further discussion through questions, this feature contributes to a more interactive and productive user experience.

**Stimulus/Response Sequences**

**Summarising Messages:**

1. The system monitors ongoing conversations.
2. The system will be able to analyse conversation content in real-time.
3. When prompted or at designated intervals, the system analyses the conversation content.
4. Based on the analysis, the system generates a concise summary of the conversation's key points.
5. The summary is presented to the user either within the conversation thread or as a separate notification.
6. Summaries will be generated accurately and efficiently, considering the context and relevance of the messages.

**Asking Questions:**

1. Upon detecting potential areas for clarification or further discussion, the system formulates relevant questions.
2. These questions are designed to encourage users to elaborate on specific topics or provide additional context.
3. Questions are seamlessly integrated into the conversation flow, prompting users to respond and engage further.

**Question Generation:**

1. The system will have predefined templates or algorithms for generating questions based on conversation context.
2. Questions will be formulated in a natural and conversational tone to encourage user engagement.
3. The system must ensure that questions are contextually relevant and align with the ongoing conversation.

**User Interaction:**

1. Users will have the option to enable or disable message summarisation and question prompts.
2. They will be able to provide feedback on the accuracy and usefulness of summaries and questions.

3. The system will adapt its summarisation and questioning strategies based on user preferences and feedback.

**Integration and Compatibility:**

1. This feature will seamlessly integrate with other system functionalities, such as messaging interfaces and user profiles.
2. It will be compatible with various messaging platforms and communication channels supported by the application.

## 3.2 Design and Implementation Constraints

Hardware Limitations

- The application must be able to respond to a user's request in a reasonable amount of time to be perceived as real time.
- There exists no memory requirements, although a lightweight application is preferred to reduce server load and cost.

Interfaces to other Applications

- The application is required to access and interact with the REST (Representational State Transfer) APIs provided by RapidAnalysis and Slack.

Specific Technologies, Tools, and Databases to be used

- GitHub, for a version control code repository.

Parallel Operations

- The application must be able to handle multiple user requests simultaneously while interacting with both used APIs.

Language Requirements

- Python, for the programming language to be used in constructing the application.

Communications Protocols

- HTTP, when interacting with APIs using both HTTP POST and HTTP GET requests when needed.

Design Conventions or Programming Standards

- PEP8 (Python Enhancement Proposal 8), a Style Guide for Python Code to conform to industry standards.
- PEP257 (Python Enhancement Proposal 257), Docstring Conventions for Python to conform to industry standards.

## 3.3 Usability Requirements

### 3.3.1 User Interfaces

1. **Pop-up Chat (Help Bot):**
   - Users can interact with the chatbot using a trigger phrase like "`/infobyte`,".
   - When a user types "`/infobyte`," the chatbot replies with relevant information or assistance.
   - This approach allows users to quickly access help or information without leaving the current channel. So it acts like a shortcut.
   - Consider adding a consistent message prefix (e.g., "This message was generated by [Chatbot Name]") to distinguish chatbot responses from human messages.
2. **Private Message (PM) Interaction:**
   - Users can open a direct message (DM) with the chatbot.
   - In the DM, users can send commands or queries to the chatbot.
   - The chatbot responds directly in the DM, providing personalised assistance.
   - Advantages: Privacy, focused interaction, and no interference with other channel conversations.
   - We are considering implementing a command prefix (e.g., "`/help`" or "`/info`") to initiate chatbot interactions.
3. **Hidden Command Approach (e.g., `/infobyte`):**
   - Users can send hidden commands to the chatbot within regular conversations.
   - For example, a user might type, "I need information about project deadlines. `/infobyte`"
   - The chatbot detects the hidden command and responds accordingly.
   - This approach integrates chatbot functionality seamlessly into ongoing discussions.
   - Ensure that the hidden command is unique and unlikely to be used accidentally.
4. **Accessing Channel Message History:**
   - To provide context, the chatbot can ask users to refer to specific channels or threads.
   - For example, a user might ask, "What was discussed in #project-planning?"
   - The chatbot can retrieve relevant information from the specified channel's message history.
   - We are considering using Slack's API to access channel history programmatically.
5. **GUI Standards:**
   - **Sample Screen Images:**
     - We will be using Figma to create mockups or wireframes of our chatbot's interface. These visual representations help convey the layout, components, and interactions.
     - We will Define a consistent visual style for our chatbot consider Slack's existing design patterns and adhere to them.

  - We will Avoid cluttered layouts to prioritise essential content.
- **Standard Buttons and Functions:**
  - The interface will Include standard buttons or interactive elements for common actions:
    1. **Help**: A button or command to access chatbot documentation or FAQs.
    2. **Cancel**: For interrupting ongoing processes.
    3. **Submit**: To confirm or submit user input.
    4. **Back/Next**: For multi-step interactions.
  - Use consistent labels and icons for these buttons.
- **Keyboard Shortcuts:**
  - Slack supports keyboard shortcuts. So we will implement shortcuts for common chatbot actions:
    1. E.g., `/help` for accessing help, `/info` for information recovery.

### 3.3.2 Hardware Interfaces

Since the application being developed is intended to run within Slack, the hardware interfaces required will match that of the Slack requirements. Slack is a cross-platform application and is able to run on any hardware that supports a browser, or uses Linux, Windows, MacOS, Android, or iOS.

### 3.3.3 Software Interfaces

**API Protocols**

- **Slack API**: Used to integrate the chatbot with Slack, enabling message handling, events, and interactive features.
- **RapidAnalysis**: Used to make analytical responses for message handling and interactive features.

**Database Connection**: The message history will not be saved on any external database for security concerns.

**Programming Languages**: We will use the latest Python (3.10) version for our project. Python has a rich set of Natural Language Processing (NLP), and Machine Learning libraries. Moreover, the scalability to integrate with other technologies and frameworks made it suitable for our project.

**Operating System**

The application will be hosted on Amazon AWS for serverless web applications. We chose AWS according to our client's recommendation as it runs and manages compute resources automatically. It is also safe and secure to maintain and monitor the APIs.

**Services and Communication**

- **Authentication Service**: Ensuring secure communication is paramount for authentication. This service will be employed to confirm the chatbot's identity when accessing APIs.

- **Analysis Service**: This service aids the chatbot in submitting data analysis requests to the RapidAnalysis API and receiving the processed results during communication exchanges.
- **Result Generation Service**:Based on designated keywords, parameters, or data inputs, the chatbot will produce detailed analytical reports.

**Data Sharing Mechanism**

It involves transferring various data and messages between the chatbot and RapidAnalysis API.

- Input Data:
  - The user requests for data analysis or report generation using specific parameters such as text, commands, etc. The authentication credentials are kept secure and transmitted via API access.
- Output Data:
  - It is processed data that generates formatted output by analysis findings.
  - If there is a failed request, it will show error messages during processing.

**Implementation Constraints**

- **Global Data area**: Used to share common data such as user information, histories, configuration and machine learning models across different parts of the chatbot. It will be implemented in AWS cloud storage for concurrent access and data persistence. We will also implement security measures to protect sensitive information.
- **Secure transmission**: Data Transfer between the chatbot and Rapidanalysis API must attach to secure transmission protocols such as HTTPS (HyperText Transfer Protocol Secure) to provide data confidentiality and integrity.
- **Data validation**: When the user inputs any data, it must be validated to prevent spam requests to ensure accurate output.
- **Error Handling**: The system will use a strong error-handling procedure to manage exceptions and unpredictable responses from the API.

### 3.3.4 Communication Interfaces

The project utilises two REST-compliant API's, namely the RapidAnalysis API, and Slack API, each with specifications as below. All HTTP methods sent to each REST API are served over HTTPS. RapidAnalysis API:

- The RapidAnalysis API methods use HTTP GET, POST, PUT, and DELETE, with a JSON (JavaScript Object Notation) payload of data pairs expected by the server. A response from the server will be structured by the following, where the output value is the expected output from the method.

```
{
     ...,
     "output": ...
 }
```

Slack API [4]:

- Slack's Web API uses HTTP POST, and GET methods for the application to access, though majority of the barebones HTTP requests are hidden behind the 'slack-bolt' Python package used for this project.
- The Event API allows for the implementation of a static web server to receive HTTP POST requests consisting of the event's JSON-based payloads from Slack on the application's event triggers. The AWS Lambda Service allows for an API Gateway to forward the event's payload in a consistent format to the application.

## 3.4 Other Nonfunctional Requirements

### 3.4.1 Performance Requirements

The chatbot will respond to user prompts within 5 seconds so that users perceive the chatbot as interactive, responsive and available. Longer than this may negatively affect the user experience.

The application will support asynchronous API calls to prevent blocking user interactions during long-running operations. In case of other factors that are slowing the chatbot, it will notify the user after the 5-second mark.

Regular performance testing is necessary for this project. The team will also implement monitoring tools to track response times.

### 3.4.2 Security Requirements

**Risk Identification and Management:**

- The product needs to implement a process for identifying and managing risk associated with product usage.
- Develop contingency plans for potential threats both internally and externally to the system to address security issues.

**Privacy Protection:**

When collecting, storing, processing, and transmitting user's personal information. The product must take appropriate security measures to protect it.

**User Identification and Authentication:**

Introduce mechanisms to verify and authenticate the identity of users. This enhances account security and prevents unauthorised access.

**Privacy and Terms of User**

Personal Information Collection and Use:

- Clearly specify the purposes of use before collecting the user's personal information.
- Personal information will only be collected and used when legally required or with the user's consent

Information Security**:**

- Implement appropriate technical and organisational security measures to securely protect user's personal information.
- Implement security features such as data encryption, access control, and prevention of unauthorised access.

Terms of Use:

- Provide users with clear and easy-to-understand terms of use.
- The terms of use will include service terms, privacy policies, responsibilities, and liability limitations

The product will not introduce the risk of security issues for users.

- No user information or messages are stored by the app.
- Only the user can view their messages to and from the chatbot.
- The connection to the RapidAnalysis API is authenticated with an API key. Each user has their own API key which no one else will be able to access, like a password.
- The app does not store API keys directly in main code, but allows users to securely input it into environmental variables. The user manual will explain in more detail, and the app should guide users through this one-time setup process.

The terms of use include the following.

- The user is responsible for maintaining the confidentiality of their Slack account credentials and preventing unauthorised access to their devices logged into the accounts.
- The user is responsible for maintaining the confidentiality of their RapidAnalysis API key and preventing other agents from interacting with the LLM on their behalf.
- The user will follow the other Slack terms of use.

### 3.4.3 Software Quality Attributes

The quality attributes are derived from the target audience of the product. As explained in the User Classes and Characteristics section, the product is primarily for professionals who use Slack to communicate with others in their organisation, and developers may extend on this project in the future. Therefore the product must be easy to use (for Slack users) and extensible (for future developers who want to implement more functionalities).

1. Ease of use: The target audience (professionals using Slack for team communications) has a positive user experience. Users find that using the chatbot is intuitive. Within any Slack channel, users simply send the correct command from the messaging box. Ease of use also encompasses other aspects of usability, e.g. continuous availability is a prerequisite for use.
2. Extensibility: Developers can extend the project to integrate more capabilities into Slack. The product information encourages development and includes hyperlinks to highly readable and comprehensive developer guides. The codebase is well-documented and modular for easy understanding.

# 4 Other Requirements

## Client feedback

Date and time: 10:00, 22/03/2024

### Feedback received

1. Include monetisation opportunity for RapidAnalysis in Project Scope.
2. Include a short user story.
3. State the Python version and package requirements.
4. There are no corporate or regulatory policies for this project.
5. Recommendation: Host the Python code on a AWS Lambda Service which is a serverless endpoint. Alternative recommendation: Run an Amazon Elastic Compute Cloud virtual server instance which allows more flexibility but requires management of the API wrapper and firewall access.
6. Outline where and how users will store their RapidAnalysis API key (or maybe OpenAI Key in the future) using simple language for non-technical users.

### Response

1. Project Scope notes the "potential monetisation of the API for RapidAnalysis".
2. User Classes and Characteristics presents brief user stories in bullet points.
3. Operating Environment states the required Python version, along with package requirements and their versions.
4. Omitted corporate and regulatory policies from Design and Implementation Constraints.
5. The team has chosen to use the AWS Lambda Service, and updated accordingly.
6. Security Requirements notes that users store their API key privately.


# Infobyte

Team 25

## Analysis and Design

Slack app with RapidAnalysis API

Version 2.0 16.05.2024

# Contents



# Revision history

| Name | Date | Reasons for changes | Version |
|---|---|---|---|
| IW | 11/05/2024 | Edited use cases to have IDs, bullet-point format, and consistent Actor names. | 2.0 |
| YL | 15/05/2024 | Updated Figure 1 to reflect the above changes. | 2.0 |

# 1 Analysis

## 1.1 Use Case Diagram

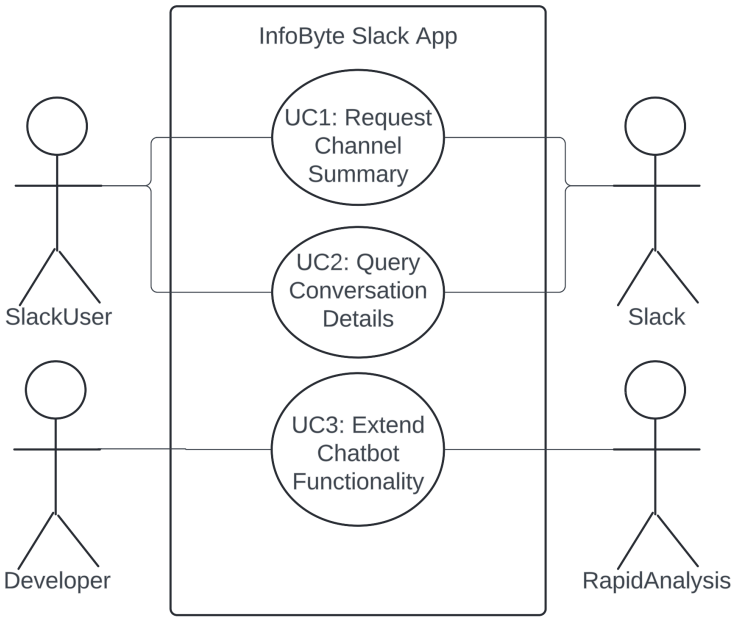



Figure 1: Use Case Diagram

## 1.2 Use Case Description

Use cases UC1, UC2 and UC3 are identified below.

**UC1: Request Channel Summary**

- **Actor**: Slack User
- **Description**: The user interacts with the chatbot by requesting a summary of messages from a specific Slack channel.
- **Preconditions**: The user must have installed the chatbot in their Slack workspace.
- **Postconditions**: The chatbot generates and returns a summarised overview of the messages in the requested channel.
- **Main Flow:**
  1. User initiates a conversation with the chatbot.
  2. User specifies the Slack channel for which they want a summary.
  3. Chatbot utilises RapidAnalysis's API to process and summarise messages from the specified channel.

4. Chatbot presents the summarised information to the user within the Slack interface.

- **Alternative Flow:** If the specified channel does not exist or is inaccessible, the chatbot notifies the user accordingly.

**UC2: Query Conversation Details**

- **Actor:** Slack User
- **Description:** The user queries the chatbot for specific details or keywords within a Slack channel's conversation.
- **Preconditions:** The user has access to the chatbot within their Slack workspace.
- **Postconditions:** The chatbot provides relevant information based on the user's query.
- **Main Flow:**
  1. User interacts with the chatbot and submits a query related to a specific conversation topic or keyword.
  2. Chatbot processes the query using RapidAnalysis's API to extract relevant details from the channel's messages.
  3. Chatbot returns the requested information (e.g., key points, discussions) to the user.
- **Alternative Flow:** If the chatbot cannot identify relevant information based on the query, it notifies the user or suggests refining the query.

**UC3: Extend Chatbot Functionality**

- **Actor:** Developer
- **Description:** A developer extends the functionality of the chatbot using RapidAnalysis's API.
- **Preconditions:** The developer has access to the chatbot's source code and RapidAnalysis's API documentation.
- **Postconditions:** The chatbot's capabilities are enhanced or customised to meet specific requirements.
- **Main Flow:**
  1. The developer accesses the chatbot's codebase and relevant documentation.
  2. The developer integrates additional features or custom logic using RapidAnalysis's API (e.g., advanced summarization techniques, and natural language processing).
  3. The developer tests and deploys the updated chatbot version to the Slack Marketplace.
- **Alternative Flow:** If integration with RapidAnalysis's API encounters issues, the developer troubleshoots and resolves them before deployment.

## 1.3 User Stories

As a Slack user, I want to request a summary of messages from a specific channel, so that I can quickly understand recent discussions without having to read through all the messages.

**Acceptance Criteria:**

- The chatbot should be able to generate a concise summary of messages from the specified channel.
- The summary should cover key points and discussions within the selected timeframe.

As a Slack user, I want to query the chatbot for specific details or keywords within a channel's conversation, so that I can find relevant information efficiently.

**Acceptance Criteria:**

- The chatbot should be able to process natural language queries and extract relevant details from the channel's messages.
- The chatbot's response should include information related to the queried topic or keyword.

As a business leader (manager), I want to use the chatbot's summaries to make informed decisions based on past discussions within our Slack channels.

**Acceptance Criteria:**

- The chatbot's summaries should provide actionable insights and key takeaways from channel conversations.
- The summaries should help in understanding trends, issues, and progress discussed within the team.

As a developer, I want to extend the chatbot's functionality using RapidAnalysis's API, so that I can enhance its capabilities and offer additional features.

**Acceptance Criteria:**

- The chatbot's codebase should be well-documented and modular to facilitate integration with RapidAnalysis's API.
- The extended functionality should align with user needs and improve the overall utility of the chatbot.

As a Slack user, I want the chatbot to be intuitive and easy to use, so that I can quickly access summarised information without a steep learning curve.

**Acceptance Criteria:**

- The chatbot should have clear commands and responses that are easy to understand.
- The user interface should be user-friendly and accessible within the Slack environment.

As a developer, I want comprehensive documentation and support resources for extending the chatbot's functionality to integrate new features and troubleshoot issues efficiently.

**Acceptance Criteria:**

- The documentation should include API integration guidelines, code samples, and usage scenarios.
- Support resources such as forums or helpdesk should be available for developers to seek assistance when needed.

# 2 Design

## 2.1 System Design Document

### 2.1.1 System Architecture

The system architecture for our application relies heavily on AWS (Amazon Web Services) services and packages within to interact with Slack. User's main interaction and Graphical User Interface will be within Slack itself where they can use the commands to execute functions.

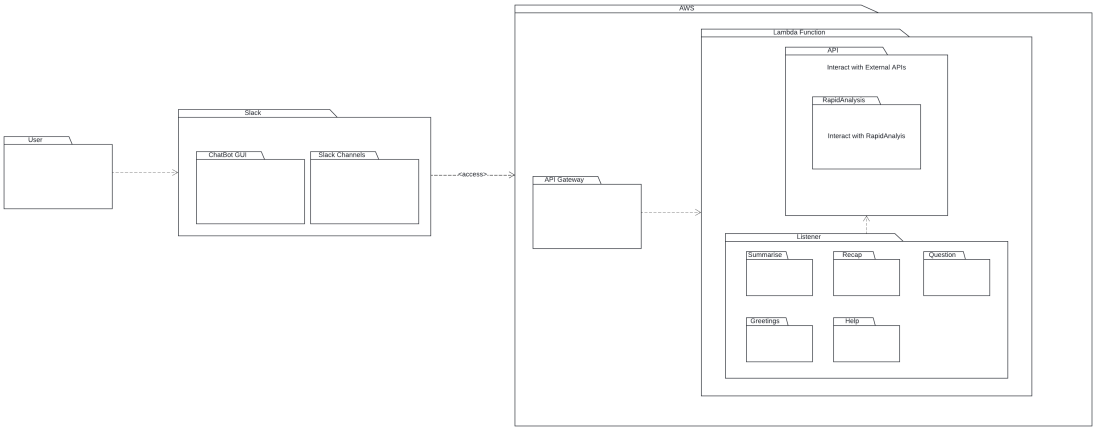



Figure 2: System Architecture

### 2.1.2 Storage/Persistent Data Strategy

The project requires persistent storage of consumer API keys for accessing RapidAnalysis, with each workspace requiring a key. The following describes varying strategies with their respective advantages and disadvantages. In addition, the Slack application requires a signing token provided by Slack, and a bot token for each workspace in which the application is installed.

**Parameter Store**
AWS provides the service 'Parameter Store' allowing for the storage of key, value pairs, and as such can provide a means of storing API keys for each unique workspace designated by a workspace identifier provided by Slack. The service allows for both unencrypted storage and

encrypted storage with the use of AWS KMS (Key Management Service). In addition, the signing token from Slack can be stored within its own parameter.

**Advantages:**

- Without AWS KMS
  - Provides an inbuilt method of accessing values, allowing for safe access given roles have been established with correct permissions.
  - Free AWS service.
- With AWS KMS
  - Provides encryption of keys, preventing intruders from easily accessing the values, provided they do not have permissions. This requires that AWS user, and service roles have been established removing their ability to leak this information.

**Disadvantages:**

- Without AWS KMS:
  - Values are not encrypted, thus can be accessed more easily, which compromises security.
- With AWS KMS:
  - Implementation incurs varying costs depending on the load of the Slack application:
    - Non-AWS managed keys are stored at a cost of $1 per month.
    - Access to the KMS key incurs a charge of $0.03 per 10,000 requests, after the first 20,000 free requests.

**Secrets Manager**

The secrets manager allows for storage of API keys, providing a perfect option for storing the Slack application's signing token.

**Advantages:** Improved security by removing hard-coded credentials and replacing with dynamically retrieved values from the Secrets Manager Service when authorised access is provided.

**Disadvantages:** Each secret added costs $0.40 per month for storage, and $0.05 per 10,000 API calls.

### 2.1.3 Concurrent Processes

All concurrent requests to the Slack application will be sent to an AWS API Gateway, which redirects requests along with their data to a Python Lambda function, which proceeds to process each request individually (although concurrently), and provide a response to Slack and thus the user.

### 2.1.4 User Interface Strategy

The main goal of the user interface is to present an easy and smooth interaction in Slack for the user and the developer. It is aimed at making it a pretty easy way to get the functionality of the chatbot for summarisation, query, and help in the easy extension of the chatbot by the developer.

**Key Features**

- **Interactivity:** This feature allows instant user-to-user feedback in a familiar context of Slack command interactions.
- **Simplicity:** The simplicity approach for the UI eliminates complexity with simple and clear options for commands and outputs.
- **Accessibility:** Search and download right through Slack—no need for any other installations or setup.

**Design Approach**

- **Command-Based Interaction:** The user interaction with the chatbot should be based on simple text commands, which start with `/help`, `/recap`, `/summarise` etc.
- **Feedback Mechanisms:** Users receive textual and visual feedback after commands are issued.
- **Error Handling:** All errors or absence of data are handled with meaningful messages through the UI.

**Integration with Slack**

- **Design Consistency:**
  - The UI should adhere to design guidelines, ensuring a native feel in the Slack app.
  - Utilises the modals and buttons provided by Slack to attain an interactive experience.
- **Adaptability and User Experience**
  - **Scalability:** Can be adapted for different loads and maintain performance with an increase in the number of users.
  - **User Experience:** Help users achieve their goals with as little time and difficulty as possible, while they must also be written in simplified language to foster quick interactions.

**Tools and Technologies**

- **Figma:** For prototyping and designing the UI for both web and mobile.
- **Slack APIs:** To roll our own interactive elements.

### 2.1.5 Design Decision Choices and Trade-Offs

**Command-Based Interaction**

**Decision:** Accept the use of command-based interaction as proposed by Slack.
**Pros:**Simplifies the cognitive load for users; hence, it is not complex.
**Cons:** Limited support in terms of hardware and software to get involved against Graphic Users.

**Feedback Mechanism**

**Decision:** This decision has taken into consideration both text and visual feedback.
**Pros:**

- Clear use of voice that gives users great feedback.
- It's more humanly oriented; and reduces environmental impact.

**Cons:** Additional development effort and potential visual clutter.

**Error Handling**

**Decision:** Use clear messages and fail-fast error handling.
**Pros:**

- The guidance and satisfaction of the user are higher.
- Clean and easy-to-read code.

**Cons:** Requires extensive testing and increases code complexity.

**Use of Slack's Interactive Components**

**Decision:** Extensive use of Slack's inbuilt components like buttons and modals.
**Pros:** Engaging and structured user experience.
**Cons:** It might have unnecessary complexity since it requires the platform of Slack.

**Scalability and Customisability**

**Decision:** Will use an adaptable architecture that is highly flexible.
**Pros:** It allows third-party plugins and further expansion.
**Cons:** It has a complex initial architecture and risk of potential security.

**Tool Choices (Figma and Slack APIs)**

**Decision:** The tools that we would develop with are Figma and Slack.
**Pros:** It integrates with nearly seamless rapid prototyping and has a modern and simple UI.
**Cons:**

- Some of the features are quite limiting and require a subscription.
- These are the strategic decisions we made to balance user experience with development efficiency and adaptability in a way that the developed chatbot is not only user-friendly but also robust within the Slack ecosystem.

## 2.2 User Interface Layouts

We have used Figma to create the User Interface for our Slack app. In the Slack app, we have multiple screens to use different commands. The bot will be replying to the commands accordingly. We have created the User Interface for both Mobile and for the web:

### Mobile UI

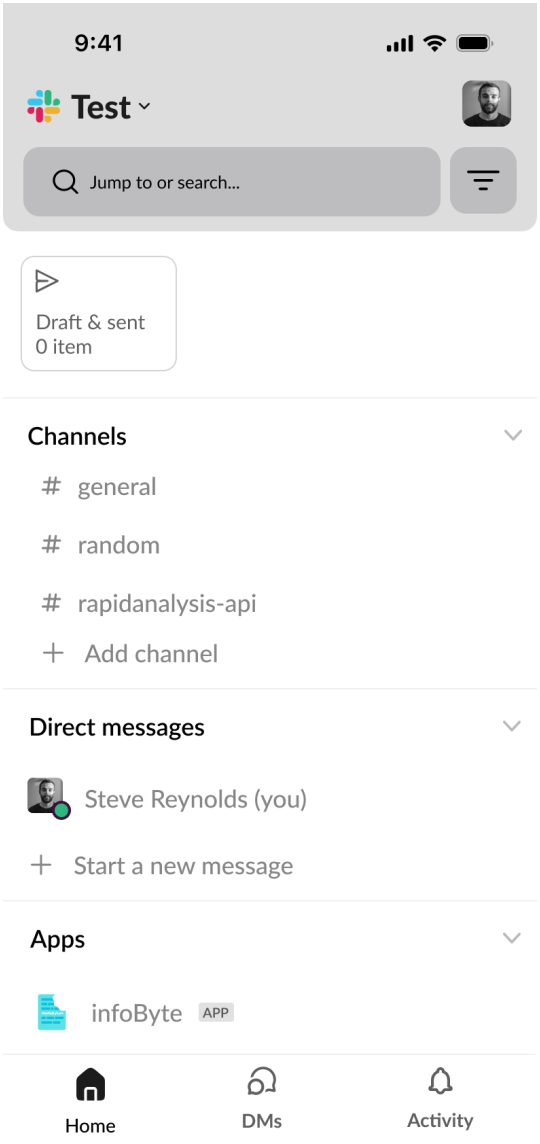



Figure 3: Home Page

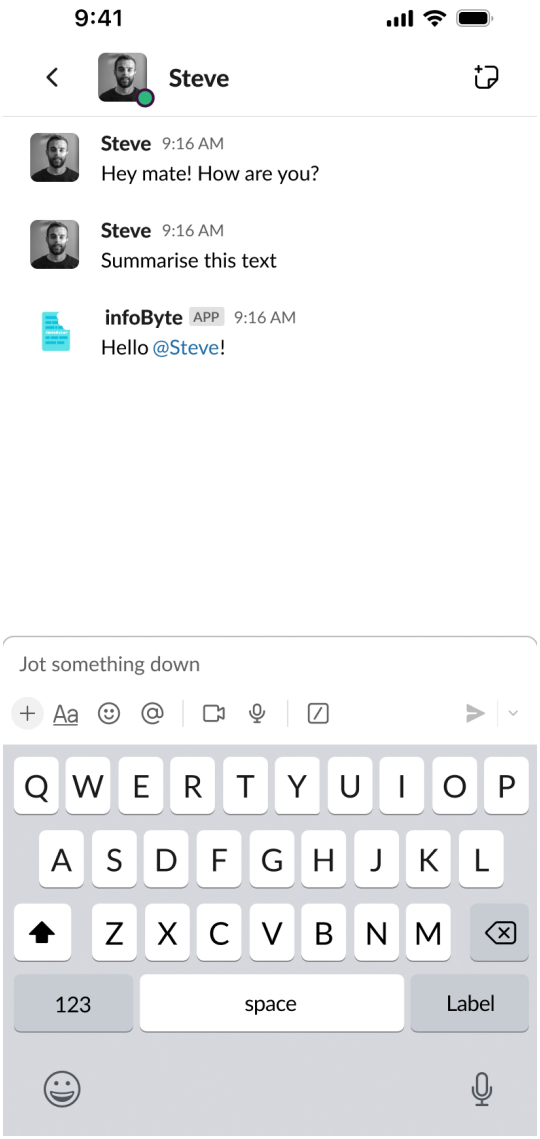



Figure 4: Main screen with keyboard

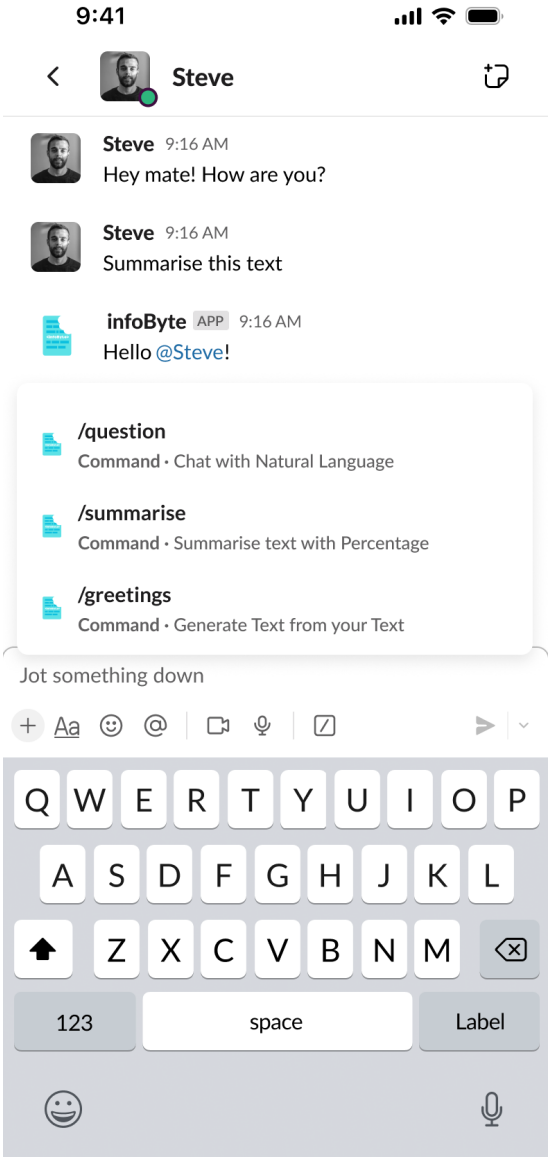



Figure 5: Pop up command

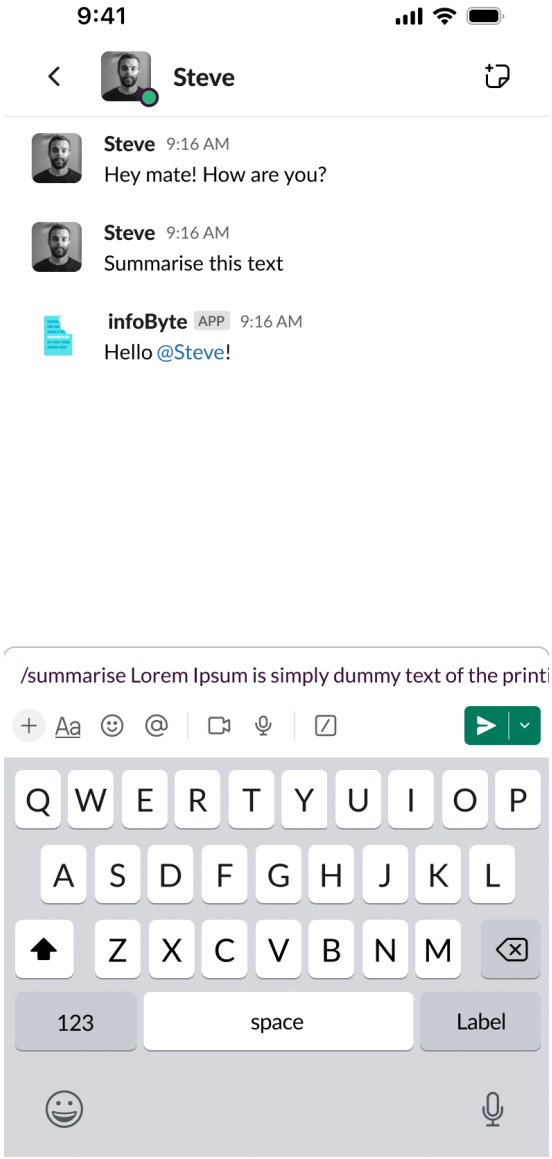



Figure 6: Written Message

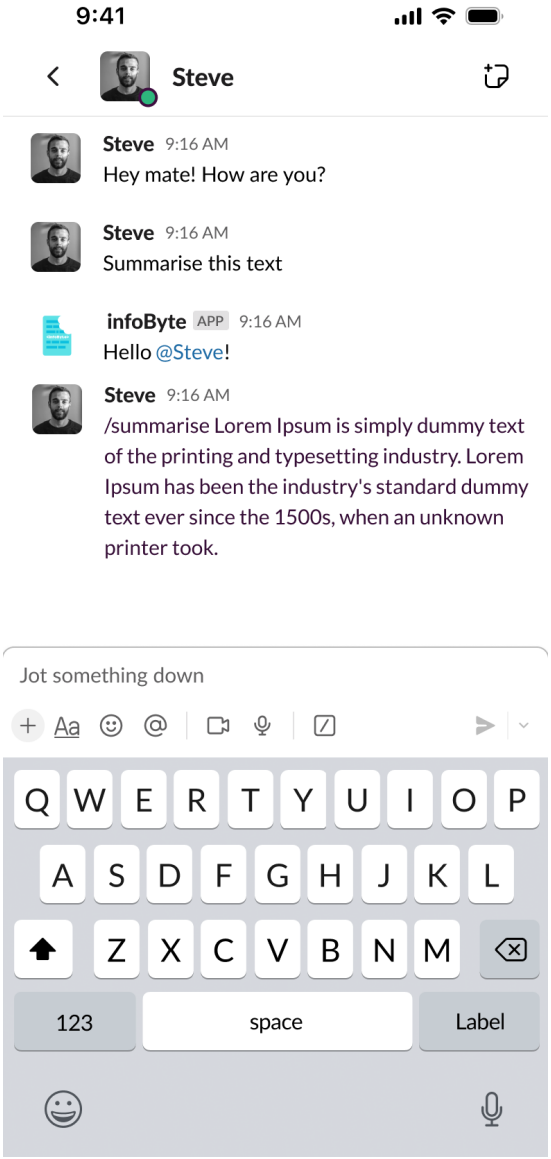



Figure 7: Sent message

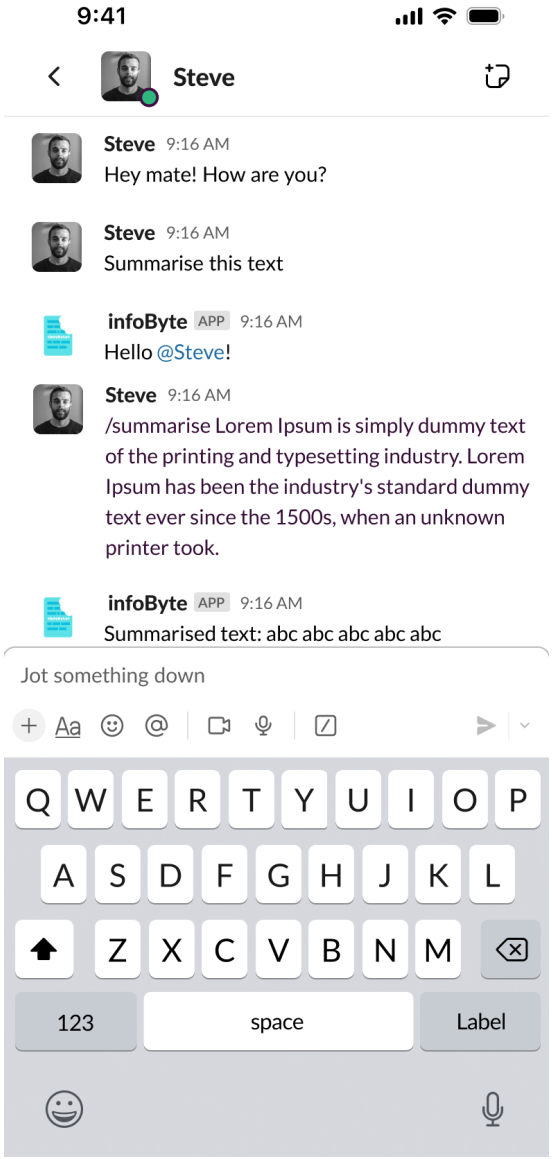



Figure 8: Received Reply

## Web UI

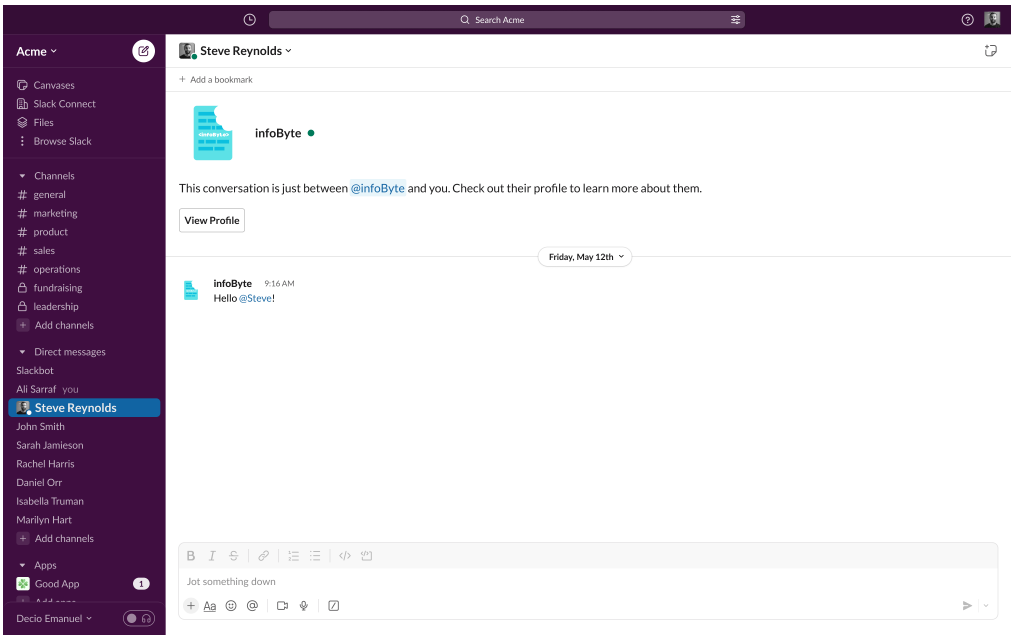



Figure 9: Homepage

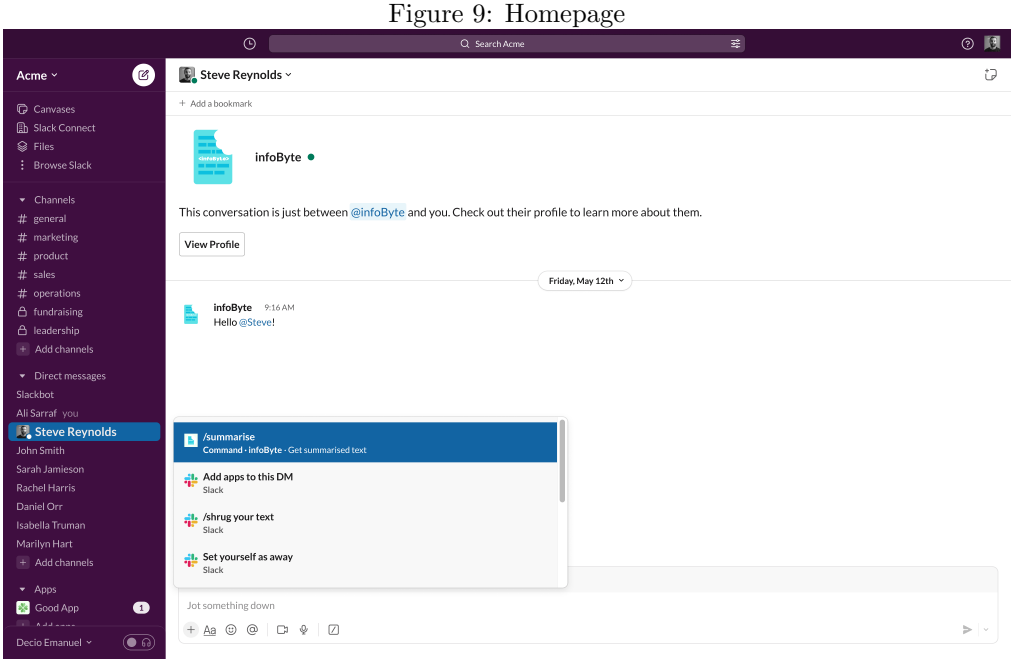



Figure 10: Commands

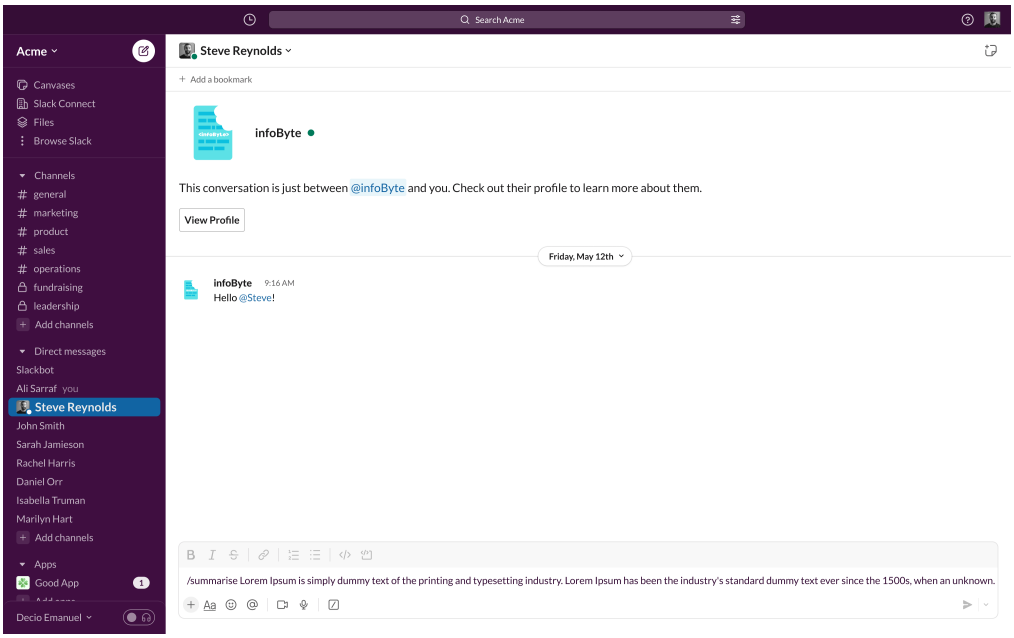



Figure 11: Written commands

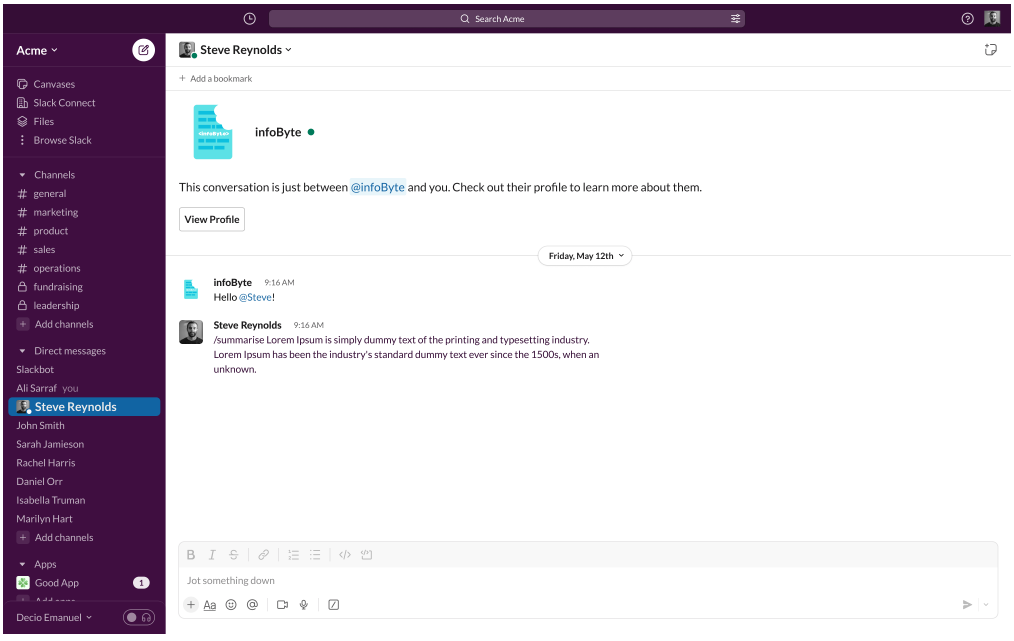



Figure 12: Sent message

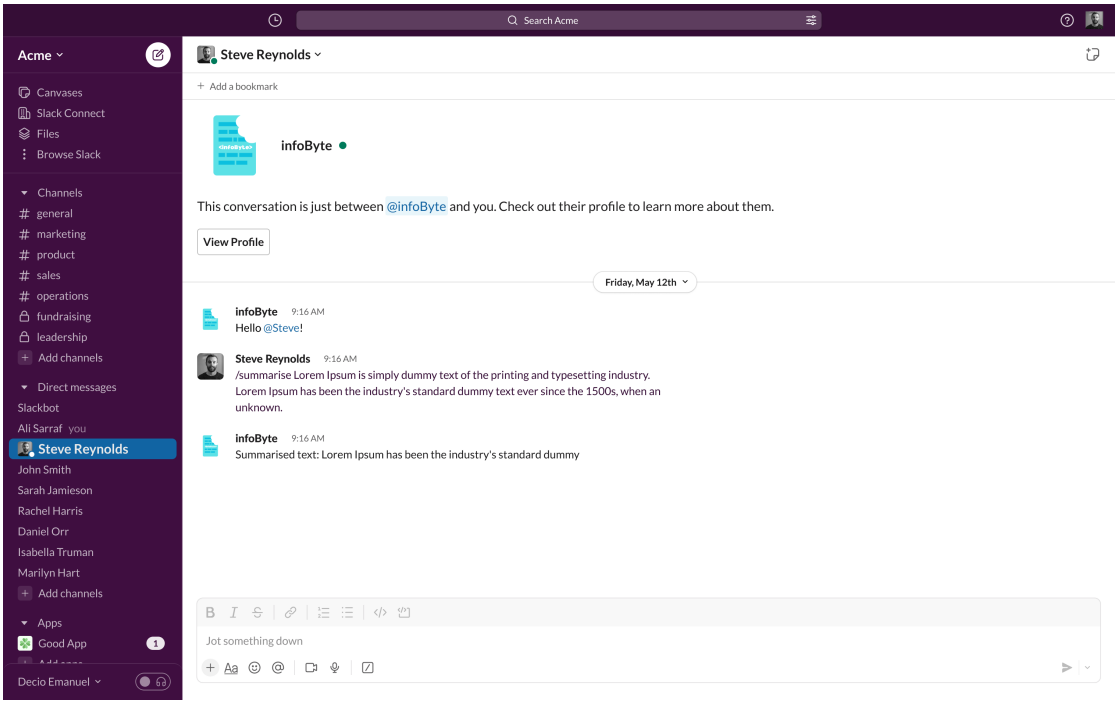



Figure 13: Received Reply

## 2.3 Window Navigation Diagram

The Window navigation diagrams look like below:

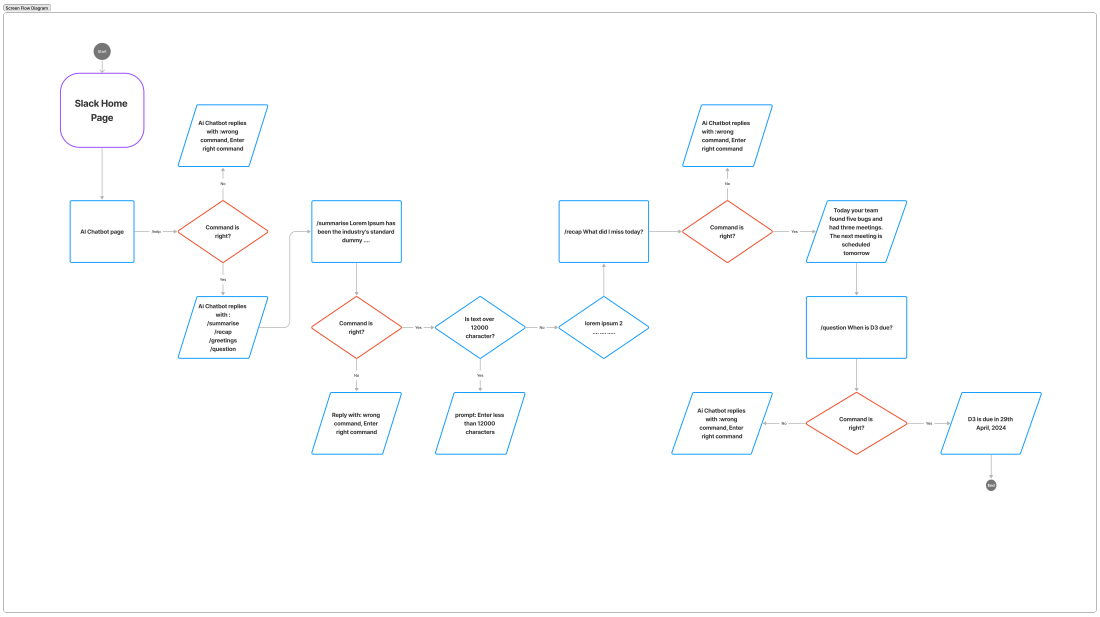



Figure 14: Window Navigation Diagram

## 2.4 Data Definitions

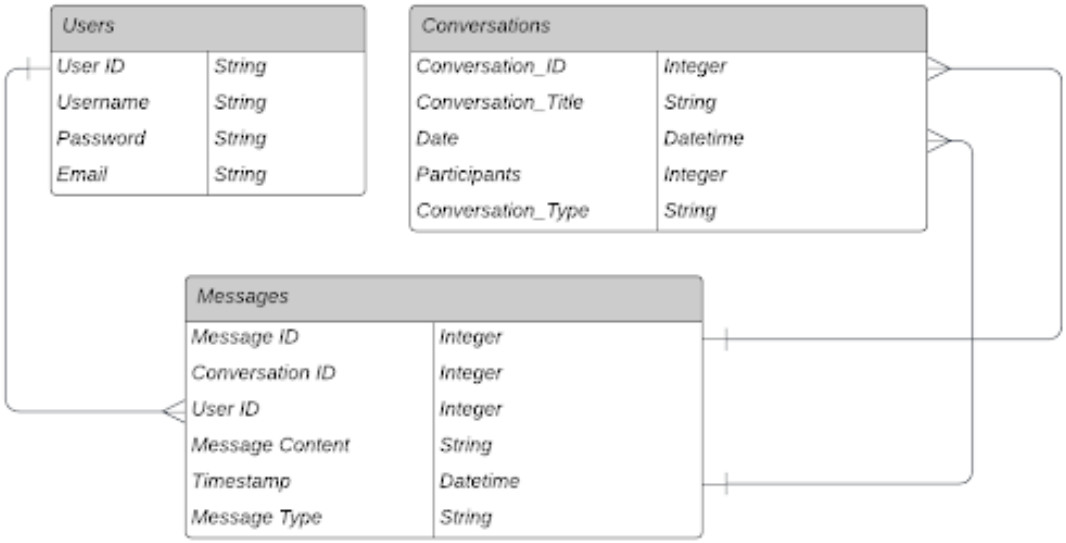



Figure 15: Data Definitions

**Table Names:**

1. Users

2. Conversations
3. Messages

**Data Field Names, Types, Examples:**

- Users
  - User ID: Integer, Example: “ben96”
  - Username: String, Example: ”Donghyun_Lee”
  - Password: String, Example: “1q2w3e4r”
  - Email Address: String, Example: ”dlehdgus0987@gmail.com”
- Conversations
  - Conversation ID: Integer, Example: 20240101
  - Conversation Title: String, Example: ”Slack App with Rapid Analysis”
  - Creation Time: Datetime, Example: ”2024-04-21 11:30:00”
  - Participants: Integer, Example: “4”
  - Conversation Type: String, Example: ”Private Chat”, ”Group Chat”
- Messages
  - Message ID: Integer, Example: 1
  - Conversation ID: Integer, Example: 20240101
  - User ID: String, Example: “ben96”
  - Message Content: String, Example: ”We need to adjust the project meeting time.”
  - Timestamp: Datetime, Example: ”2024-04-21 11:30:00”
  - Message Type: String, Example: ”Text”, ”Emoji”, ”File Attachment”

## 2.5 Class Diagrams

This project does not require class diagrams as no classes are necessary for the design and implementation of the application. This is a result of Python’s structure, where standalone functions are allowed, and modules are implemented as per the system architecture.

## 2.6 Activity Diagrams

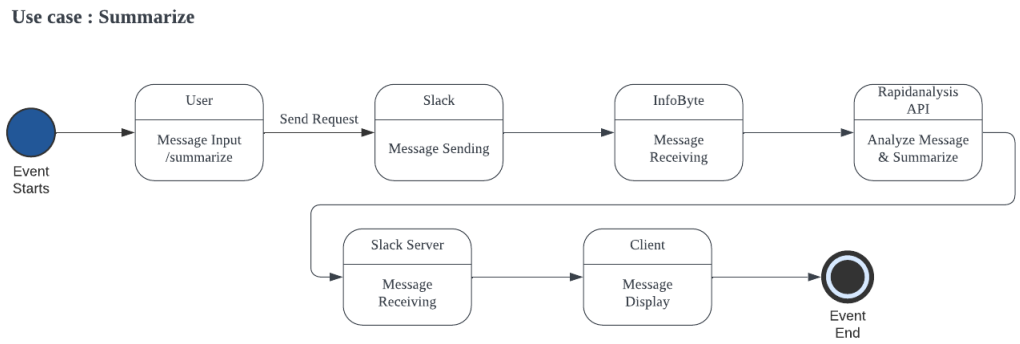



Figure 16: Use Case-Summarise

1. The user inputs the command `/summarise` in the Slack chatroom and presses send.
2. The request is sent to the server.
3. The "infobyte" component receives the request and collects the previous conversation.
4. "infobyte" utilises the Rapidanalysis API to analyze and summarize the messages.
5. The summarised result is generated and sent back to the server.
6. The result is displayed on the user's screen.

### 1.2 Query Conversation Details

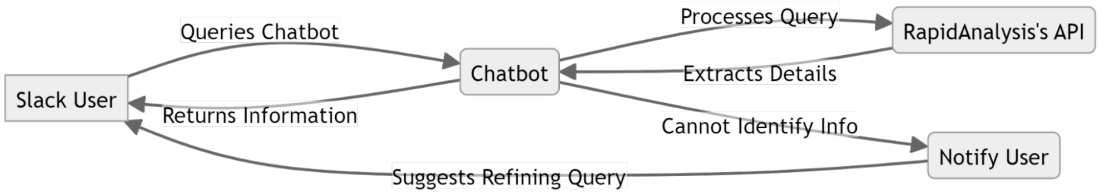



Figure 17: Use Case- Query Conversation Details

### 1.2 Request Channel Summary


Figure 18: Use Case- Request Channel Summary

### 1.2 Extend Chatbot Functionality

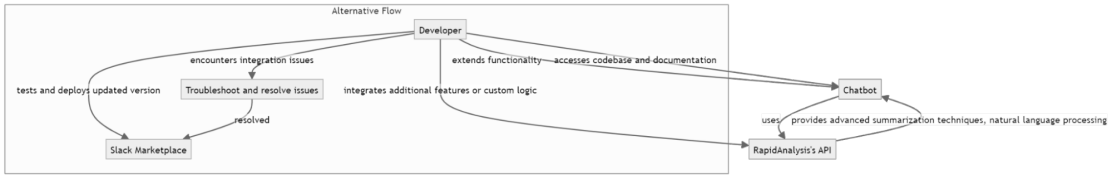



Figure 19: Use Case - Extend Chatbot Functionality

## 2.7 Assumptions

1. It is assumed that the Slack API and RapidAnalysis API are functioning reliably and can meet the requirements of the project.
2. Access to users' Slack workspaces is authenticated with the correct API keys, and users are assumed to have the necessary access permissions to utilize the chatbot's functionalities.
3. The system is expected to have sufficient memory and processing capabilities to handle large volumes of messages without causing performance issues.
4. The database is designed with a structure that can securely store large amounts of data and retrieve it quickly, while maintaining the integrity and security of the data.
5. The network connection of the system is assumed to be secure and stable, meeting all the requirements for communication with users.
6. Developers are assumed to have access to sufficient documentation and development support resources to integrate new features and debug, with comprehensive guidelines provided for API integration and feature extension.
7. Legitimate and appropriate permissions are required for users to use chatbot commands in Slack channels, thereby preventing unauthorized access to data.
8. The system is expected to be continually managed with appropriate monitoring and management mechanisms to ensure high availability and performance.
9. The system is expected to comply with the latest security regulations and laws to protect users' data and personal information.

These assumptions should be considered throughout the project's design and development phases, and each assumption should be periodically reviewed for validity as the project progresses.


# Infobyte

Team 25

## Prototype Document

Slack app with RapidAnalysis API

Version 2.0 16.05.2024

## Contents



## Revision history

| Name | Date | Reasons for changes | Version |
|---|---|---|---|
| BB | 16/05/2024 | Updated Sponsor Meeting section | 2.0 |

# 1 Prototype Screenshot

Our team has developed the initial prototype as planned for Increments 1 and 2. This section of the document presents the prototype, including screenshots and a detailed explanation of how the product looks and functions.

## 1.1 Greetings Command

Application responds to the user with the greeting “Hey there @user!”.

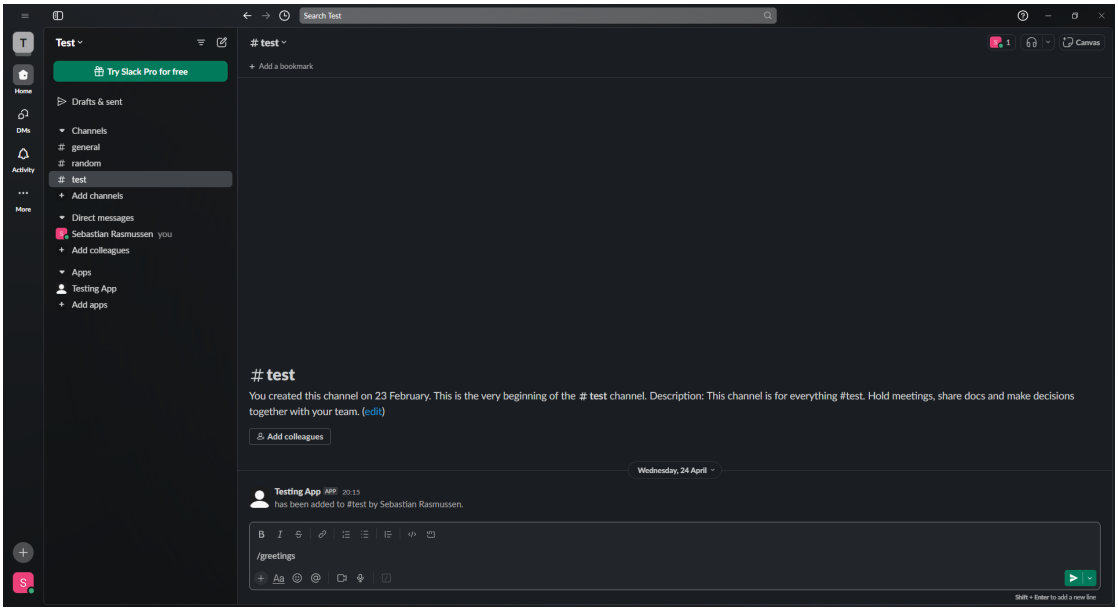



Figure 1: Greetings Command

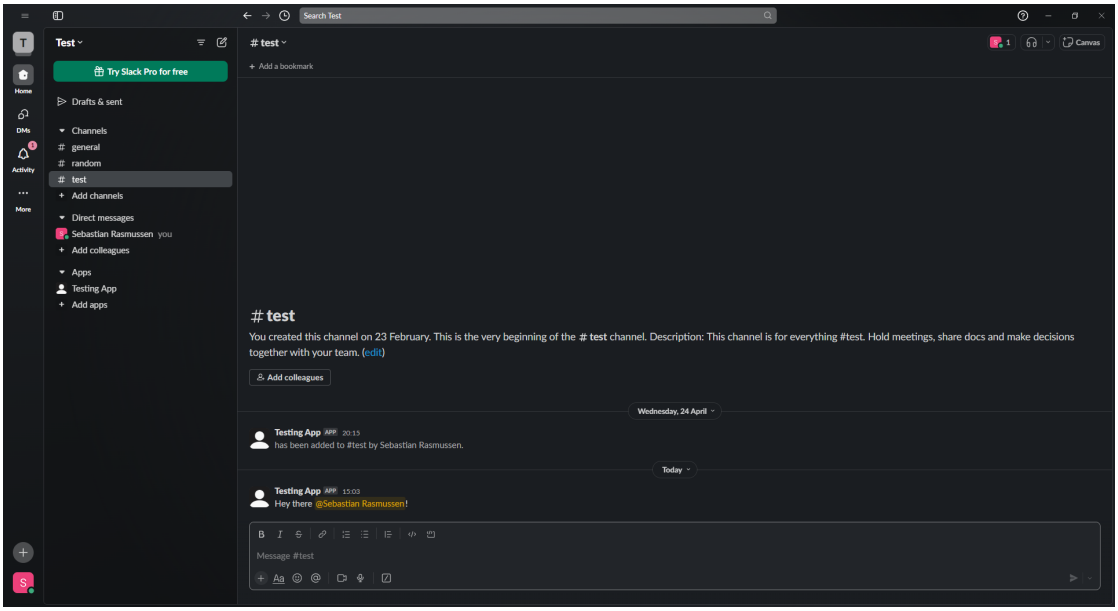



Figure 2: Reply

## 1.2 Summarise Command

*text*

Application responds to the user with a summary of the text provided, hidden from others

in the channel. Currently provides a response with a slight delay.

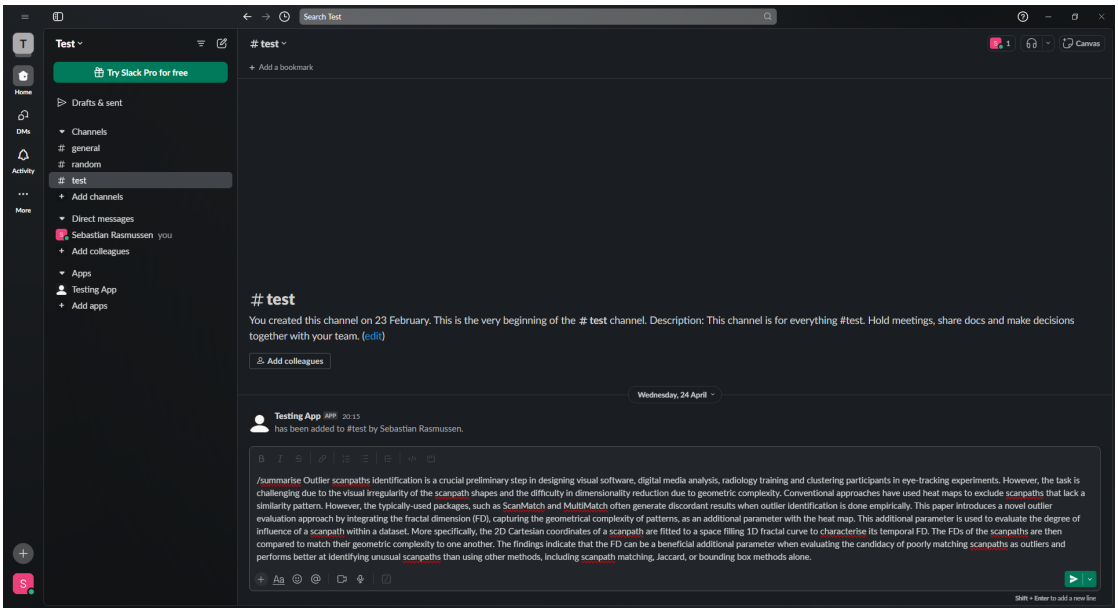



Figure 3: Summarise Command

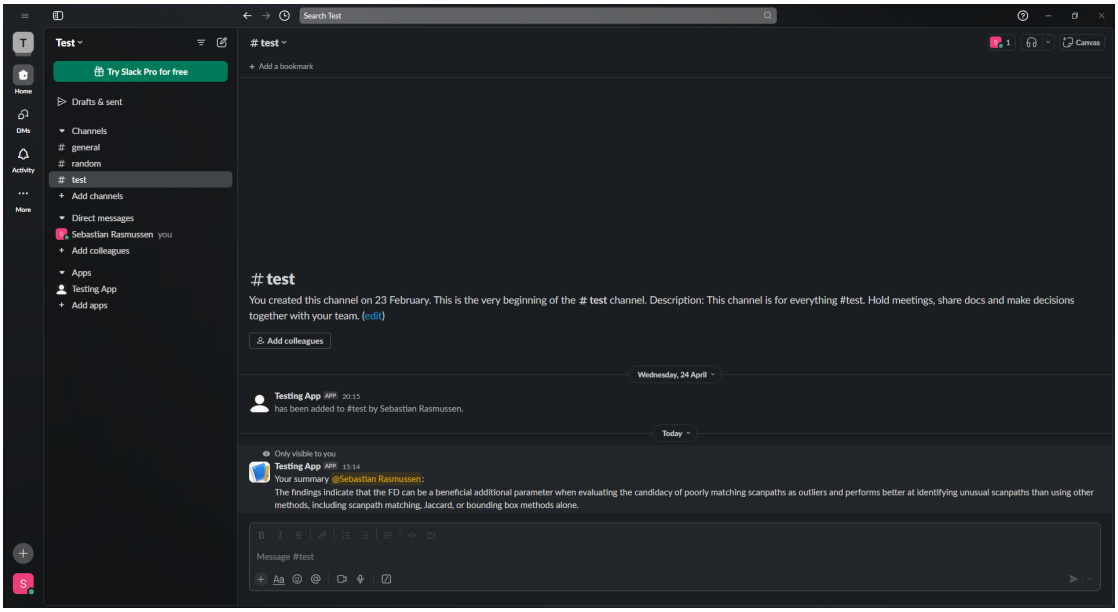



Figure 4: Reply

## 1.3 Summarise Command

*Pastebin link*

Application responds to the user with a summary of the text provided, hidden from others in the channel. Currently provides a response with a slight delay.

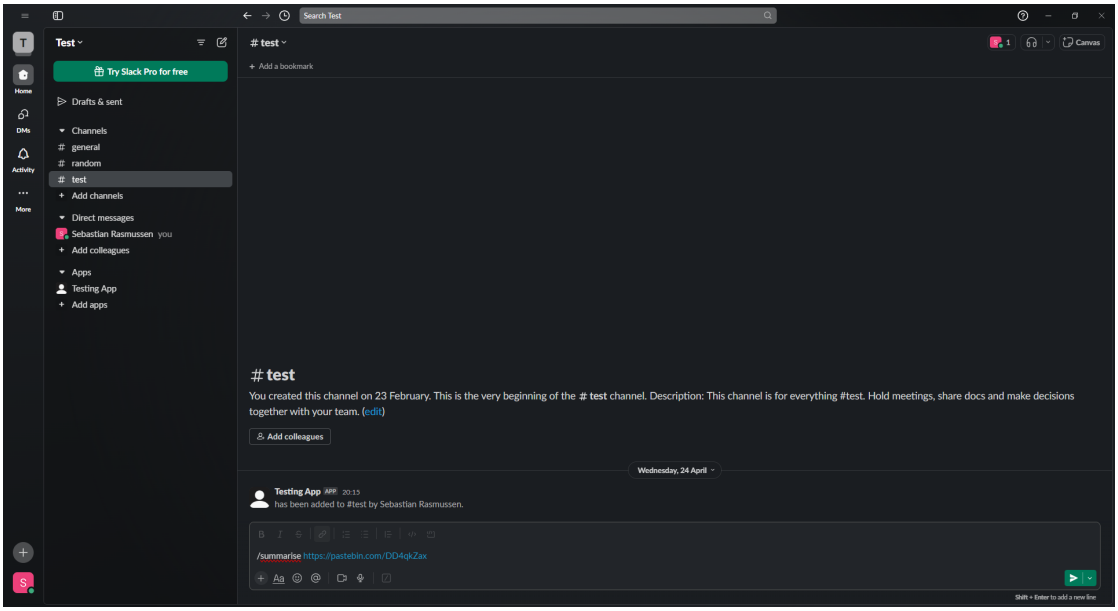



Figure 5: Summarise Command

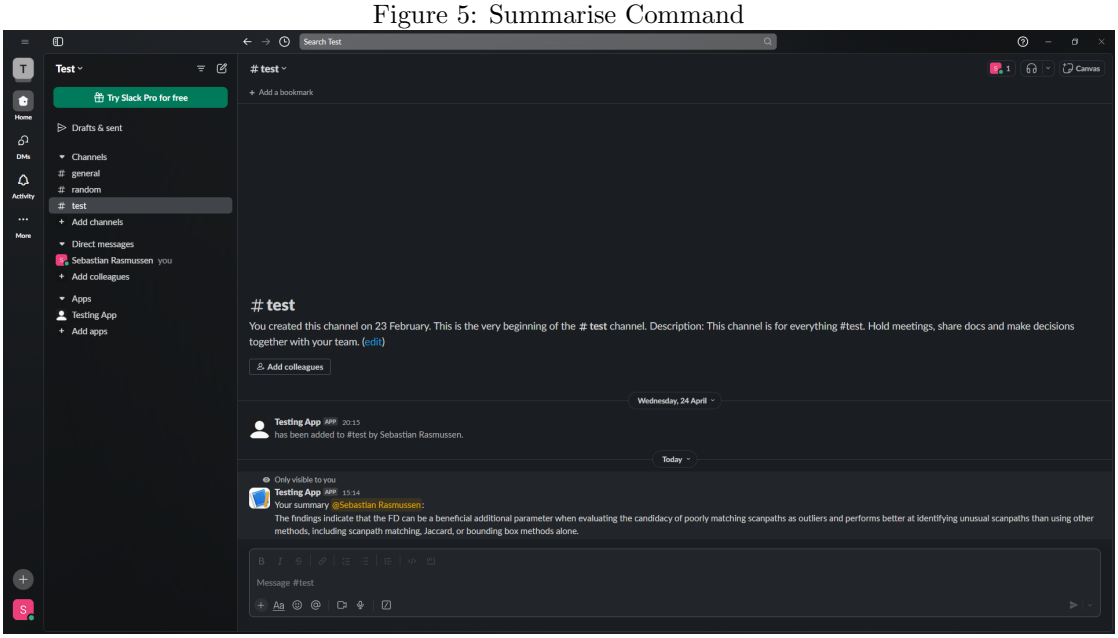



Figure 6: Reply

## 1.4 Recap Command

Responds to the user with a summary of the channels messages, hidden from others in the channel. Currently the command provides an extremely delayed response.

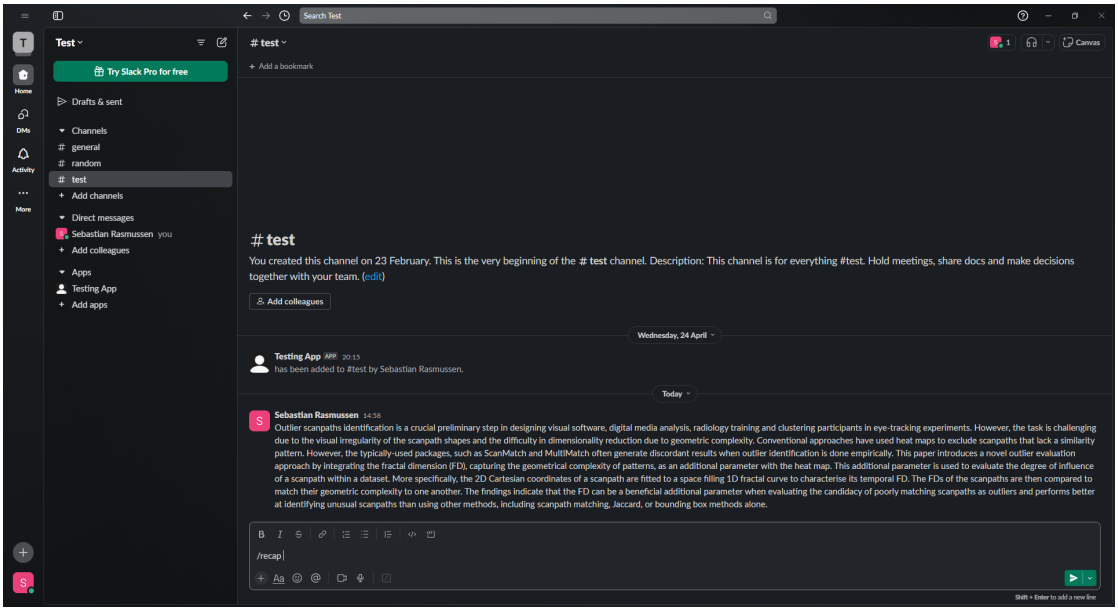



Figure 7: Recap Command

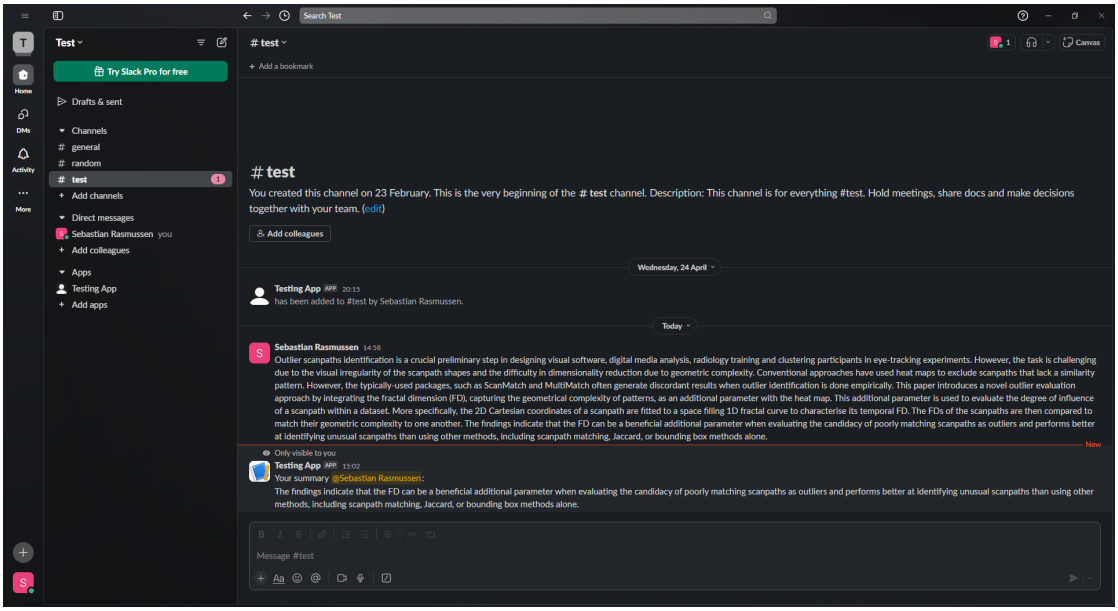



Figure 8: Reply

# 2 Sponsor Meeting

Following our proactive communication during the mid-semester break and preliminary sharing of our prototype details, we scheduled two additional meetings with our sponsor, aiming to gather direct feedback and further align our development with the sponsor's expectations. Unfortunately, despite confirming the meeting dates and times, our sponsor was unable to attend the scheduled meetings. The reasons for their absence were not communicated to us, which prevented us from rescheduling promptly.

In an effort to maintain momentum and continue our engagement, we created a detailed video demonstration of our prototype. This video was designed to visually convey the functionalities, user interactions, and potential impact of our prototype, ensuring that our sponsor could review it at their convenience. The video was sent via email by our project manager, accompanied by a comprehensive description and a request for feedback, either written or as a scheduled follow-up discussion.

As of the submission of this document, we have not received a response from our sponsor regarding the video demonstration. This lack of communication has been challenging, yet we remain committed to our project's success and to accommodating our sponsor's schedule and preferences for communication.

Given the circumstances, our team plans to undertake the following steps to ensure continuity and productive sponsor interaction:

1. Scheduled Follow-up: We will attempt to reach out again to reschedule the meeting, providing multiple time options and seeking confirmation or an alternative communication method preferred by the sponsor.

2. Alternative Communication Channels: If direct meetings continue to be challenging, we will propose the use of asynchronous communication methods, such as email feedback forms or comments directly on shared documents, to ensure that our sponsor can contribute effectively at their convenience.

3. Intermediary Updates: We will continue to send brief, regular updates via email to keep the sponsor informed and engaged with the project's progress, reducing the need for extensive catch-up during meetings and allowing for more focused feedback sessions.

Our commitment to adapting our communication and engagement strategies exemplifies our dedication to the project's success and our responsiveness to the challenges of collaborating in a dynamic project environment. We anticipate that these efforts will encourage more active participation from our sponsor and lead to valuable insights that will enhance our project outcome.


# Infobyte

Team 25

## Testing Document

Slack app with RapidAnalysis API

Version 2.0 16.05.2024

# Contents

# 1 Test Plans

## 1.1 Strategy

The project must correctly handle interactions between Slack users, the InfoByte chatbot, and the underlying RapidAnalysis AI Smart Utilities through various web requests and application programming interfaces (APIs). The development team will test the prototype to verify that it functions as intended.

Developers will design and execute test cases. The lead developer (LD) will oversee the creation of new tests to ensure that they are valid, i.e. they test the expected behaviour of the product defined in the Software Requirements Specification (SRS). The overall test strategy for this project includes the following:

1. **White-box testing** will focus on individual components of the chatbot, such as its summarising ability and integration with the Slack API, allowing developers to quickly identify and resolve small, potential problems in the prototype.
2. **Black-box testing** will address system-level performance, ensuring that the product correctly functions from the perspective of a user with no access to the internal codebase. This includes testing the chatbot's ability to understand user input, provide appropriate responses, test corner cases, and null values, and integrate seamlessly with the Slack platform.

## 1.2 Types of Tests

The development team plans to perform the following various, specific types of tests:

- **Unit testing** to test individual components or functions of the chatbot code.
- **Integration testing** to test the integration of different components and services, like the Slack API, RapidAnalysis API and AWS services.
- **System testing** to test the complete chatbot system in a production environment in various systems, including MacOS, Windows, iOS, and Android.
- **Regression testing** to test whether new iterations of the app break existing functionality.
- **Performance testing** to test the response time, timeout limit, throughput, and scalability under different load conditions.
- **Security testing** to test the security of the app and find vulnerabilities, so that the development team can quickly resolve them and ensure data privacy and protection of users.

## 1.3 Schedule

In preparing for D3, the developers focus on creating the prototype and presenting it to the client for feedback, while beginning to create test case specifications to test basic functionality defined in the SRS, for example testing whether the chatbot can send a summary of a long message to the user. After the client approves the initial prototype, the developers can begin to execute test cases and extend test case specifications to handle unusual behaviour and edge cases. This allows the development team to iterate on the product according to client expectations before creating extensive test cases.

In the next sprint for D4, the team will have received adequate feedback from the client to expand testing. The LD defines functional requirements and specific development and testing milestones for that sprint, according to the testing strategy of this project.

## 1.4 Tools and Resources

Developers will primarily use Python testing frameworks (`pytest`) and AWS testing services (lambdas). They will combine suggestions from the LD with the information in this document, including types of tests, to define and execute suitable tests for new functionality.

## 1.5 Test Case Design

The development team will comprehensively document the testing process by creating test case specifications which must include the following details:

- A unique identifier (ID) for the test case. Examples: `T_001`, `T_002`.
- Other general information for traceability purposes, including who and when the test was designed and executed.
- A description that identifies the specific functionality or scenario tested by this test case. Examples: specific Slack commands, different text to summarise which vary in length, complexity and ambiguity, and edge-case user queries like unexpected null values.
- A description of any assumed pre-conditions.
- A description of the precise steps taken to execute the test case, and the expected output or behaviour which the tester can interpret as the correctly functioning of the app.
- Real test inputs used by testers. Example: text message containing non-ASCII characters, or an invalid URL.

## 1.6 Template for Test Case Specifications

The following template can be used to ensure all test case specifications follow the same standard. Note that the subsection heading should match the test title.

**General Information**

| | |
|---|---|
| **Test Case ID:** T_XXX | **Test Designed by:** |
| **Test Priority:** Low/Medium/High | **Date of Design:** |
| **Test Title:** | **Test Executed by:** |
| **Description:** | **Date of Execution:** |
| **Pre-conditions:** | |

**Test Activities**

| **Step No:** | **Step Description:** | **Expected Result:** | **Actual Result:** |
|---|---|---|---|
| 1 | | | |
| 2 | | | |

**Test Data Sets**

| **Data Type:** | **Data Set 1:** |
|---|---|
| | |
| **Data Type:** | **Data Set 2:** |
| | |

# 2 Test Case Specifications

## 2.1 Command text boundary

### General Information

| | |
|---|---|
| **Test Case ID:** T_001 | **Test Designed by:** Sebastian |
| **Test Priority:** Medium | **Date of Design:** 14.04.24 |
| **Test Title:** Command text boundary | **Test Executed by:** Isabel |
| **Description:** Test the slack command's boundary cases with no characters, and max characters | **Date of Execution:** 16.05.24 |
| **Pre-conditions:** Access to Slack channel with Slack App | |

### Test Activities

| Step No: | Step Description: | Expected Result: | Actual Result: |
|---|---|---|---|
| 1 | Enter command '/summarise Text' | Receive a response with a length ≈ 20% of the original message. | Received a response with a length 10% to 25% of the original message. |
| 2 | Enter command '/summarise' | Receive a usage message on how to use the command. | Received a response: "None" |

### Test Data Sets

| Data Type: | Data Set 1: |
|---|---|
| Text | Contents of this lorem ipsum explainer |

## 2.2 Slack sign-in

**General Information**

| | |
|---|---|
| **Test Case ID:** T_002 | **Test Designed by:** Baizid |
| **Test Priority:** Medium | **Date of Design:** 14.04.24 |
| **Test Title:** Slack sign-in | **Test Executed by:** |
| **Description:** Test successful new user sign-in to Slack app | **Date of Execution:** |
| **Pre-conditions:** Have a Gmail or email account | |

**Test Activities**

| Step No: | Step Description: | Expected Result: | Actual Result: |
|---|---|---|---|
| 1 | Open Slack app on desktop/ mobile | App is launched | |
| 2 | Sign up with email | New account created, confirmation email sent | |
| 3 | Sign in to Slack with credentials | Signed into Slack | |

**Test Data Sets**

| Data Type: | Data Set 1: |
|---|---|
| | |

## 2.3 Greetings behaviour

**General Information**

| | |
|---|---|
| **Test Case ID:** T_003 | **Test Designed by:** Baizid |
| **Test Priority:** Medium | **Date of Design:** 14.04.24 |
| **Test Title:** Greetings behaviour | **Test Executed by:** Isabel |
| **Description:** The chatbot should greet the user when prompted. | **Date of Execution:** 16.05.24 |
| **Pre-conditions:** Access to Slack channel with Slack App | |

**Test Activities**

| Step No: | Step Description: | Expected Result: | Actual Result: |
|---|---|---|---|
| 1 | Send the message '/greetings' in Slack channel with the app enabled | Chatbot responds with the message 'Hey there @user!' | Chatbot responds with the message 'Hey there @user!' |

**Test Data Sets**

| Data Type: | Data Set 1: |
|---|---|
| Text | '/greetings' command |
| **Data Type:** | **Data Set 2:** |
| Text | '/greetings stranger', command with additional text |

## 2.4 Help behaviour

**General Information**

| | |
|---|---|
| **Test Case ID:** T_004 | **Test Designed by:** Baizid |
| **Test Priority:** Medium | **Date of Design:** 16.04.24 |
| **Test Title:** Help behaviour | **Test Executed by:** |
| **Description:** The chatbot can explain its capabilities when prompted. | **Date of Execution:** |
| **Pre-conditions:** Access to Slack channel with Slack App | |

**Test Activities**

| Step No: | Step Description: | Expected Result: | Actual Result: |
|---|---|---|---|
| 1 | Send the message '/help' in Slack channel with the app enabled | Chatbot explains its valid commands and how to use them: '/greetings', '/summarise', etc | |

**Test Data Sets**

| Data Type: | Data Set 1: |
|---|---|
| Text | '/help' command |
| **Data Type:** | **Data Set 2:** |
| Text | '/help please', command with additional text |

## 2.5 Pastebin summary

**General Information**

| | |
|---|---|
| **Test Case ID:** T_005 | **Test Designed by:** Sebastian |
| **Test Priority:** Medium | **Date of Design:** 27.04.24 |
| **Test Title:** Pastebin summary | **Test Executed by:** Isabel |
| **Description:** Test the summarise command's ability to summarise from a pastebin link. | **Date of Execution:** 16.05.24 |
| **Pre-conditions:** Access to Slack channel with Slack App | |

**Test Activities**

| Step No: | Step Description: | Expected Result: | Actual Result: |
|---|---|---|---|
| 1 | Send the message '/summarise URL' | Receive a response from the application with a length ≈ 20% of the original message | Received a response from the application with a length ≈ 20% of the original message |

**Test Data Sets**

| Data Type: | Data Set 1: |
|---|---|
| Text | '/summarise https://pastebin.com/DD4qkZax' |


# Infobyte

Team 25

## User and Training Manual

Slack app with RapidAnalysis API

Version 1.0 16.05.2024

# Contents

# 1 Introduction

This document shows users how to get started with the InfoByte application for Slack at this early stage of development.

Users should check System Requirements before following the Installation Instructions. The User Interface Overview helps users to navigate the Chatbot functionality. More detailed steps are also available in Using InfoByte Slack Application. For help, view the sections in this order: FAQs, Troubleshooting, then Contact Support.

# 2 System Requirements

Ensure that your system meets the minimum requirements for Slack before you try to install Slack and the InfoByte app. You can find out the requirements at the Slack help centre. As of May, 2024, Slack has the following minimum requirements:

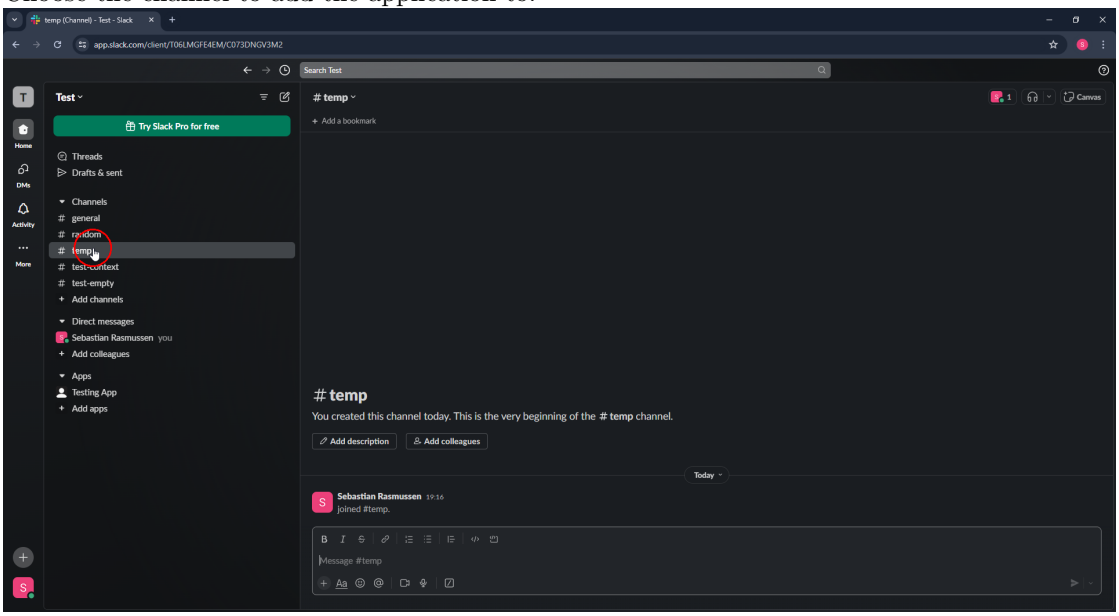
- Desktop
  - Mac OS X: MacOS 11 or above
  - Windows: Windows 10 version 21H2 or above, or Windows Server 2016
  - Linux: Ubuntu LTS releases 20.04 or above, or Red Hat Enterprise Linux 8.0 or above
- Mobile
  - iOS: iOS 16 or above
  - Android: Android 11 or above, assuming the device has Google Play Services installed
- Web browser:
  - Chrome: Version 117 or above
  - Firefox: Version 118 or above
  - Safari: Version 17 or above

# 3 Installation Instructions

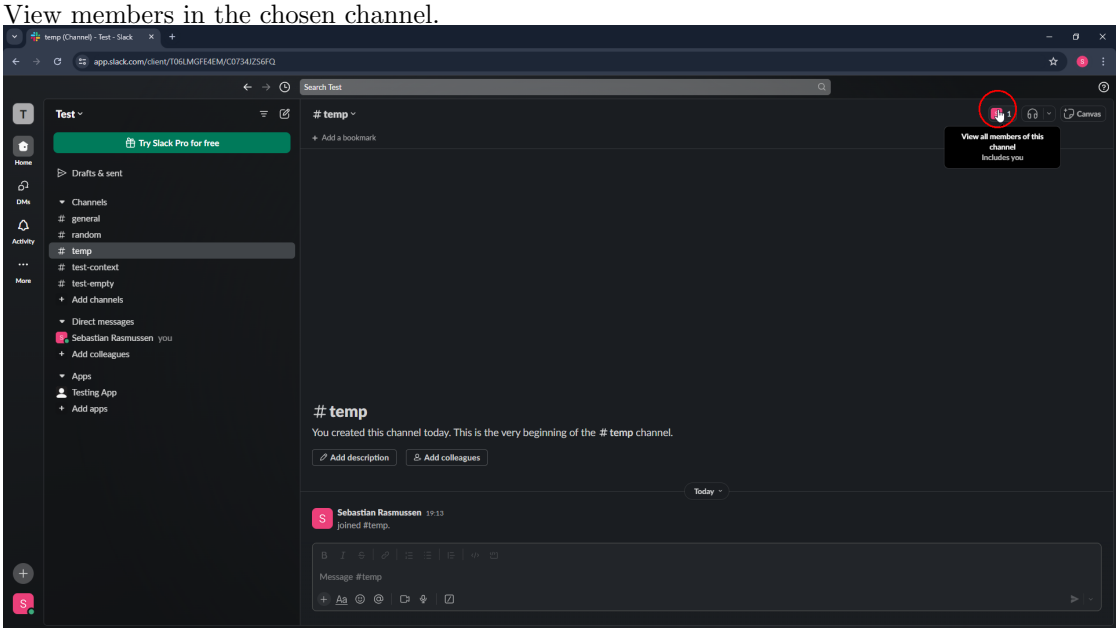
To install the Infobyte Chatbot to the workspace please follow these steps:

**Add the Slack Application to the channel**

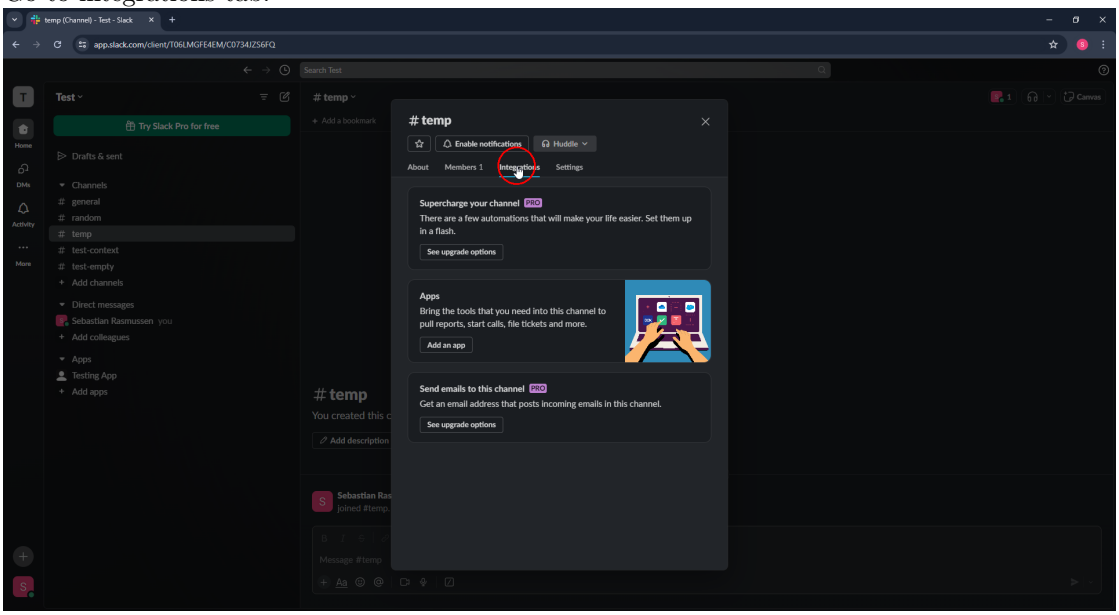
1. Choose the channel to add the application to.



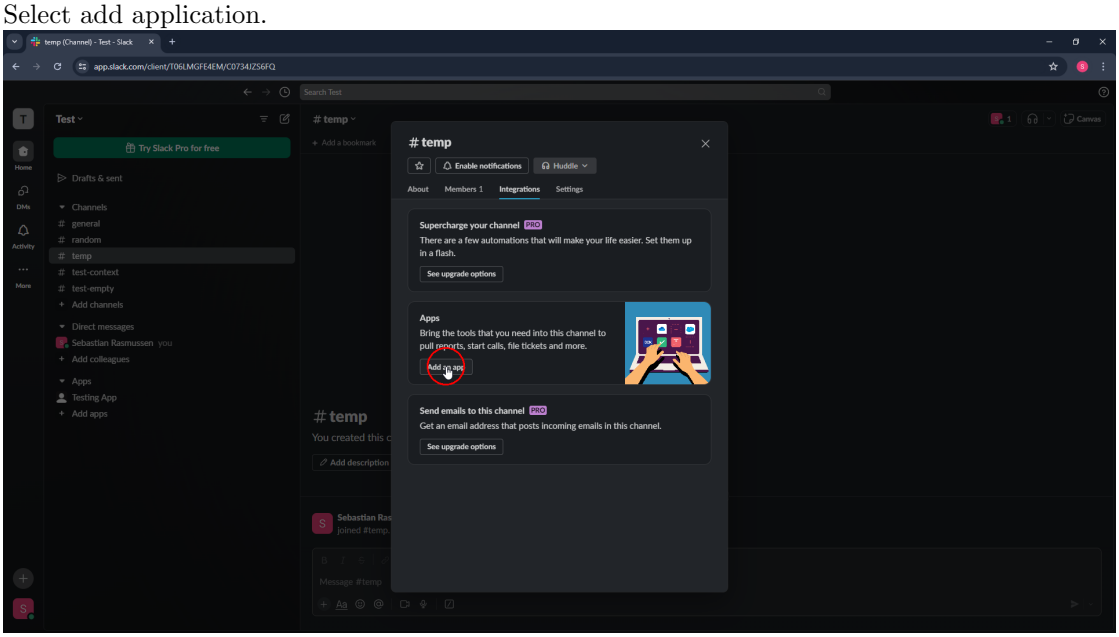

2. View members in the chosen channel.



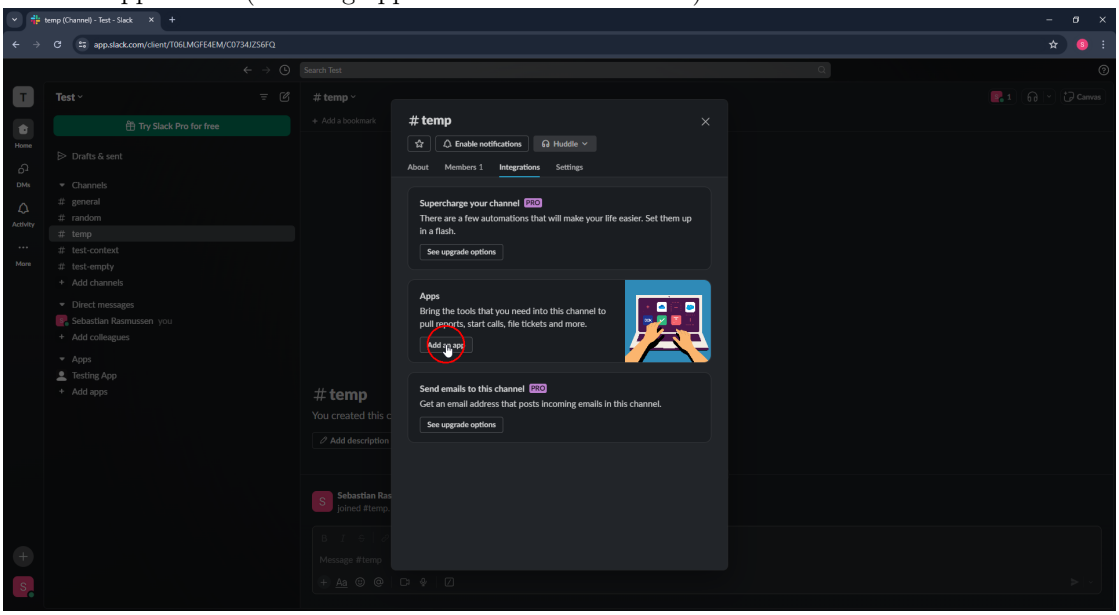
3. Go to integrations tab.




4. Select add application.



5. Add the application (a testing application is shown for this)



# 4 User Interface Overview

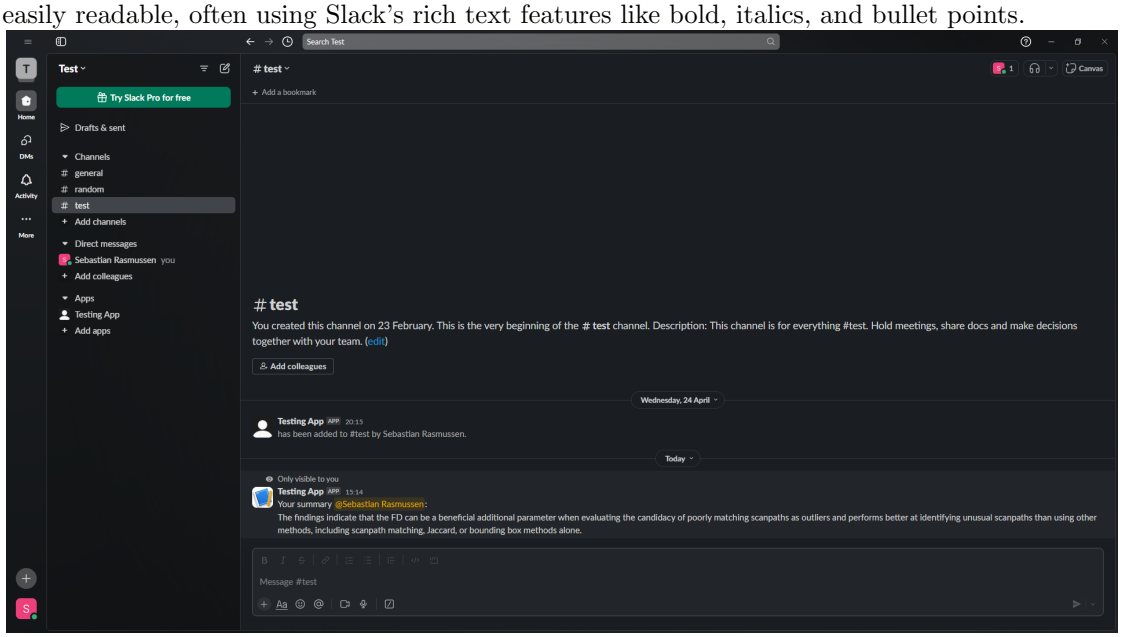
The Infobyte Chatbot for Slack is designed to integrate seamlessly into the Slack interface, providing an intuitive and easy-to-use experience that leverages Slack's existing UI components. Below is a detailed overview of the different parts of the Infobyte Chatbot interface, complemented by annotated screenshots to help users navigate and utilize the Chatbot effectively.

**Chat Commands Slash Commands:** Infobyte Chatbot operates primarily through slash commands within Slack. Users can interact with the Chatbot by typing commands directly into the message input box in any channel or direct message where the Chatbot is installed. Examples of these commands include:

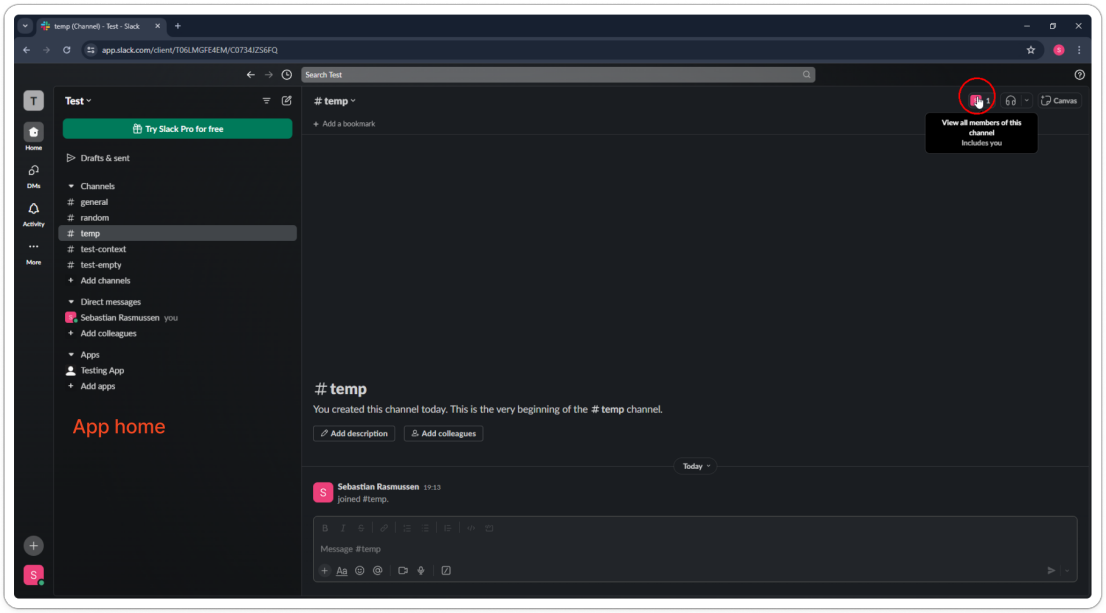
- */greet* for greetings,
- */summarize* to summarize text,
- */recap* for channel recaps,
- */question* to ask questions based on channel history.

**Interaction Flows:**

**Command Responses:** When a command is entered, the Chatbot processes the request and posts a response directly in the channel or direct message. The responses are formatted to be easily readable, often using Slack's rich text features like bold, italics, and bullet points.



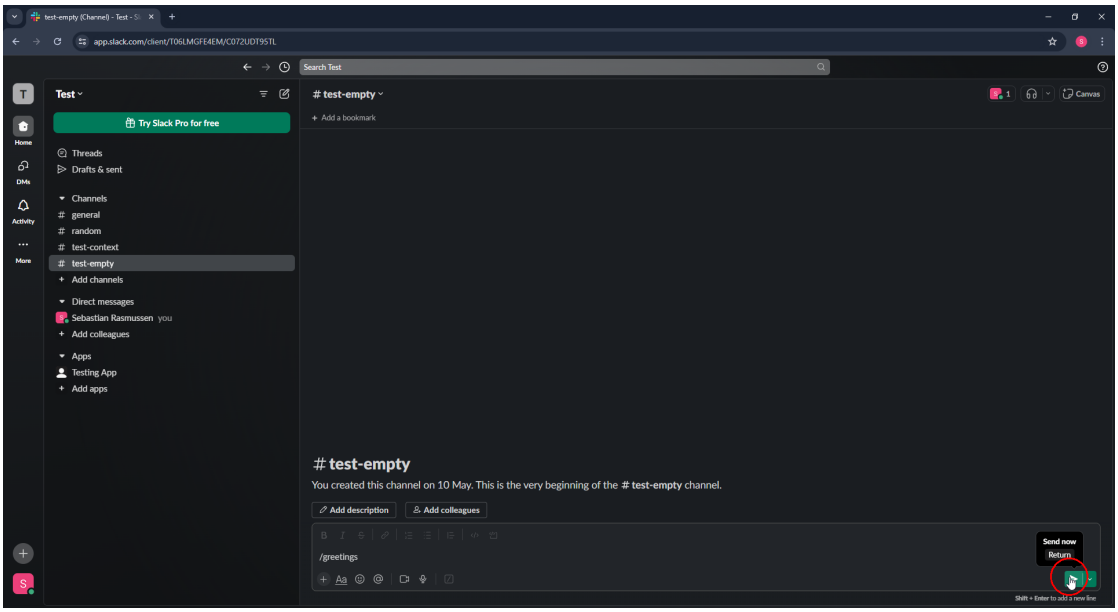
**App Home Tab:** Users can visit the App Home for Infobyte Chatbot in Slack by clicking on the Chatbot under 'Apps' in the Slack sidebar. The App Home provides a centralized interface where users can view a summary of their interactions with the Chatbot, access frequently used commands, and adjust settings.



# 5 Using InfoByte Chatbot

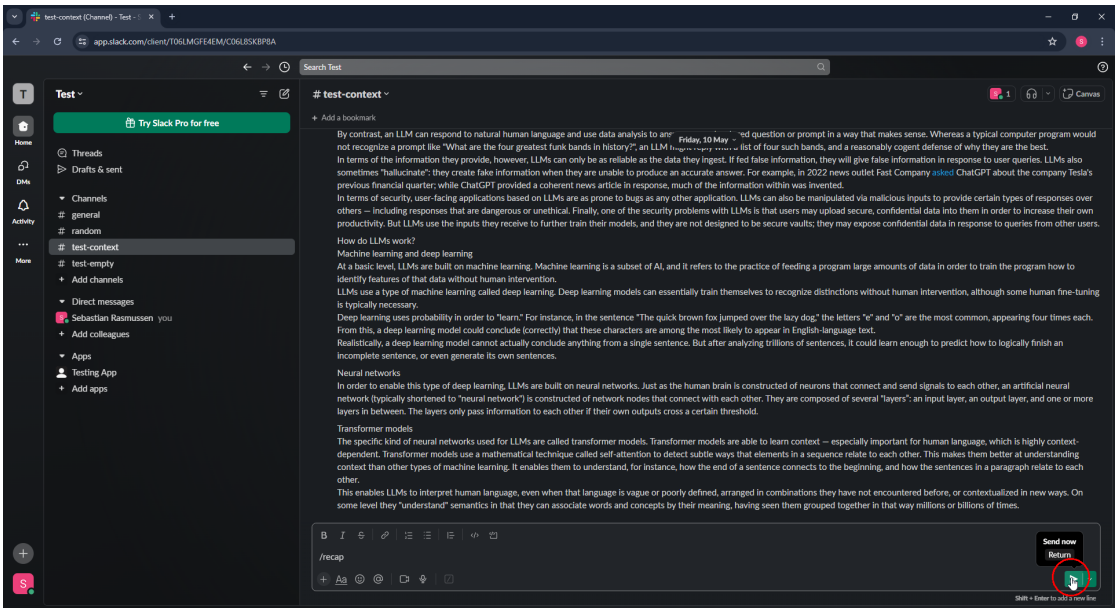
## 5.1 For Users

Command 1 - Greeting
Executing this command results in the application responding to the user with a greeting in the form “Hey there @user”.



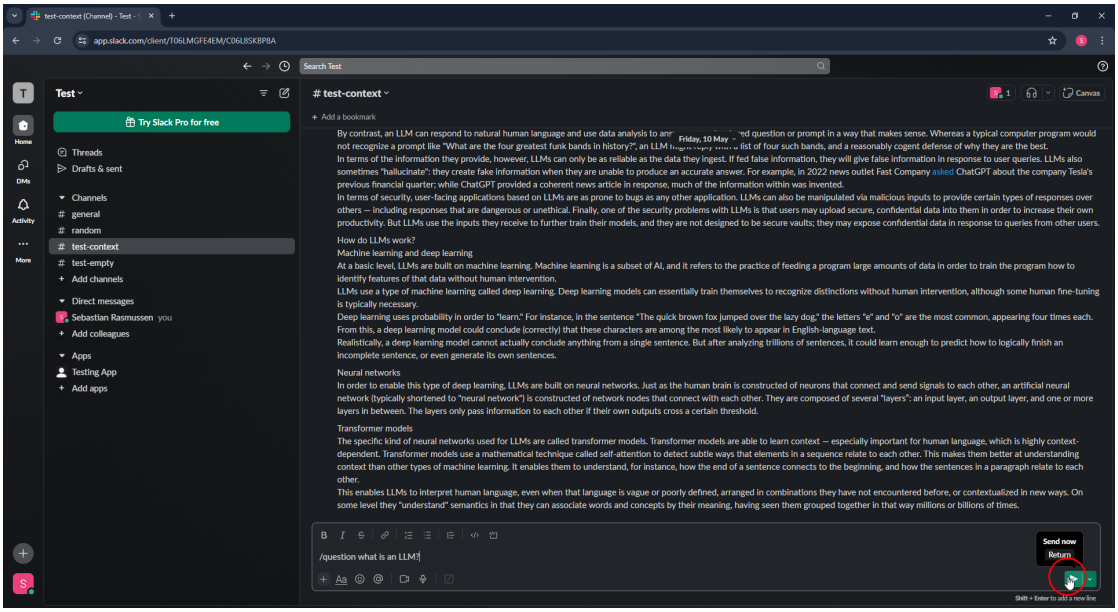

Type ‘/greetings’ and send the message to the channel.

Command 2 - Recap
Provides a summary of the channel at about 20% of the original length in a message only you can see.



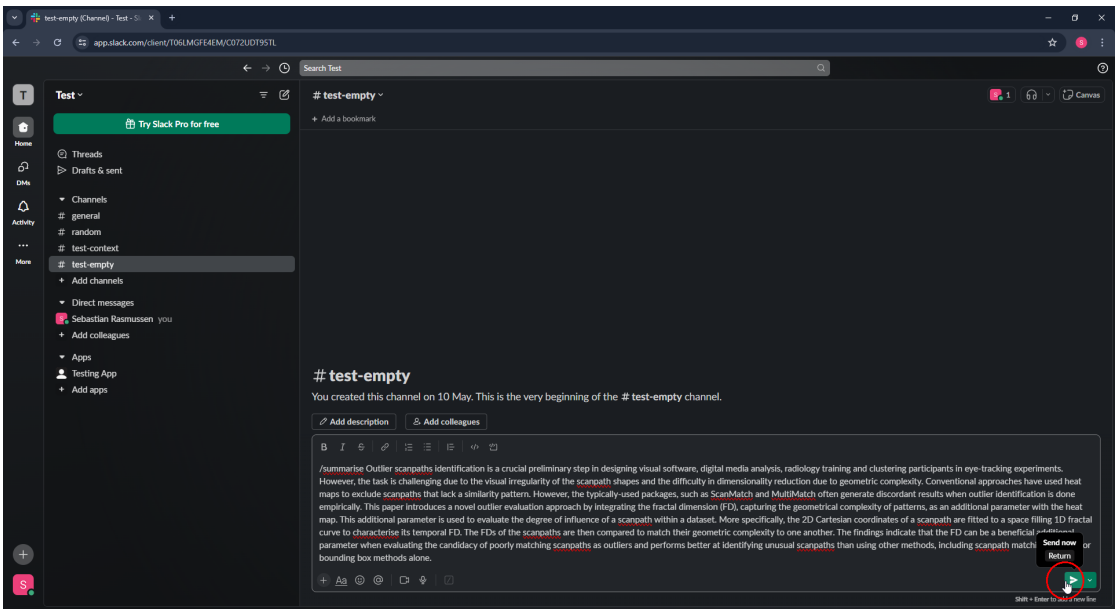

Send '/recap' to the slack channel.

Command 3 - Question
Responds to the user's question using the channel's history as context.




Send '/question' followed by a prompt.

Command 4 - Summarise
Provides a summary of the provided text (or Pastebin URL in the case of longer texts) at about 20% of the length of the original text. The slash command can be executed by entering '/summarise text—URL' in any channel that the application can access.




Type '/summarise' followed by the text to be summarised, sending the message to the channel.

## 5.2 For Developers

Ensure that you have set up an AWS account and environment. You can follow the steps here

Setup an AWS Lambda function (with invokeFunction and getFunction privileges) with an API Gateway. See here.

The repository can be cloned from github here. From here you can use your preferred IDE to edit the code.

In the created Lambda function, upload a zip file with the contents of the repository along with the required packages by running the following command in the root directory:

```
pip install --target . -r requirements.txt
```

Set the environment variables for the lambda function as follows:
RAPID_API_KEY : RapidAnalysis API Key
SLACK_BOT_TOKEN : Slack bot token for the installed workspace
SLACK_SIGNING_SECRET : Slack app signing token

# 6 Frequently Asked Questions (FAQs)

**Question 1: How do I start using InfoByte in Slack?**
Answer: To start using InfoByte, ensure it is installed in your Slack workspace. Once installed, you can interact with InfoByte by typing commands directly into the message input box in any channel or direct message where the Chatbot is present. For example, use the /greet command to receive a greeting from InfoByte.

**Question 2: Can InfoByte summarize messages from specific channels?**
Answer: Yes, InfoByte can summarize messages from specific channels. Use the /summarize command followed by the text you want to be summarized. For channel-wide summaries, use the /recap command to get a summary of the entire channel's conversation history.

**Question 3: Is there a character limit for texts that InfoByte can summarize?**
Answer: While InfoByte is capable of summarizing lengthy texts, there is an optimal limit to ensure accuracy and performance. For longer texts, consider using a Pastebin URL or breaking the text into smaller sections.

**Question 4: What should I do if InfoByte is not responding to commands?**
Answer: If InfoByte is not responding, ensure that it is correctly installed and active in your Slack workspace. Try mentioning @infobyte again or refreshing your Slack client. If the issue persists, refer to the troubleshooting section or contact support.

**Question 5: How can I configure InfoByte for my Slack workspace?**
Answer: Configuration of InfoByte can be done through the App Home tab in Slack. Here, you can adjust settings and view a summary of interactions with the Chatbot. For advanced configurations, such as API keys and integration settings, please refer to the installation and setup instructions.

**Question 6: How do I handle errors or unexpected behavior from InfoByte?**
Answer: If you encounter errors or unexpected behavior, ensure your commands are correctly formatted and that the provided text is coherent. Refer to the troubleshooting section for common issues and solutions. If problems persist, contact our support team for assistance.

**Question 7: How do I update or extend the functionality of InfoByte?**
Answer: Developers can update or extend InfoByte's functionality by accessing the source code on GitHub. Follow the provided instructions for setting up an AWS Lambda function and integrating the necessary API keys. For detailed guidance, refer to the developers' section in this manual.

**Question 8: Where can I find more information or get help with using InfoByte?**
Answer: For more information, you can refer to the user manual, the App Home tab in Slack, or contact our support team. For issues related to the RapidAnalysis API, visit the RapidAnalysis website or contact their support team.

# 7 Troubleshooting

If you encounter any issues while using InfoByte Chatbot, please refer to this section for troubleshooting advice. If the issue persists, please contact support (see next section).

**Issue: InfoByte is not responding to commands.**
Solution: Ensure that InfoByte is installed and active in your Slack workspace. If it is, try mentioning "@infobyte" again or refreshing your Slack client to prompt a response.

**Issue: The summary provided by InfoByte is incomplete or inaccurate.**
Solution: Verify that the text provided for summarisation is well-formatted and coherent. If the issue persists, try summarising smaller sections of the text or providing a Pastebin URL for longer texts.

**Issue: InfoByte is responding slowly.**
Solution: InfoByte's response time can vary based on server load and network conditions. If the delay is significant, please be patient and wait for the summary to be generated. If the problem persists, contact our support team for further assistance.

**Issue: I'm unsure how to use a specific command.**
Solution: Refer to the command syntax and usage guidelines provided in the user manual. If you still have questions, feel free to reach out to our support team for clarification.

**Issue: InfoByte is not summarizing messages as expected.**
Solution: Review the command parameters and ensure they are correctly formatted. If you're still experiencing issues, contact our support team with details about the problem for further assistance.

# 8 Contact Support

This prototype/MVP was created by a group of students of Macquarie University as their final year PACE project with the help of Rapidanalysis.. So if you need any assistance, please contact our team below:

1. Sebastian Rasmussen: **sebastian.rasmussen@students.mq.edu.au**
2. Faith Gelwyn: **faith.gelwyn@students.mq.edu.au**
3. Isabel Weston: **isabel.weston@students.mq.edu.au**
4. Baizid Bostami: **mdnsbaizid.bostami@students.mq.edu.au**
5. Dong Hyun Lee: **donghyun.lee2@students.mq.edu.au**
6. Yasmin Lee: **yasmin.lee@students.mq.edu.au**

If you need further assistance with the rapidanalysis API, please see below:

Website: RapidAnalysis
Email: **info@rapidanalysis.com**
Address: 8 Hadenfeld Ave,
Macquarie Park NSW 2113
Sydney, Australia