Software Design
CSC 480 / HCI 521 -22F

A Tool to collect useful posts from Discord

Software Requirements Specification

Document

Version: (1) **Date: (9/16/2022)**

Table of Contents

6. Supporting Information

6.1 UML diagram

- 6.1.1 Database Class diagram
- 6.1.2 Sequence Diagram
  - 6.1.2a BSD
  - 6.1.2b D2
  - 6.1.2c D1
  - 6.1.2d Main
- 6.1.3 Use case Diagram
  - 6.1.3a Database
  - 6.1.3b Bot
  - 6.1.3c Web Application

6.2 Goals from stakeholder

1. Introduction

1.1 Purpose

The purpose of this document is to present a detailed description of our engineering requirements for making a tool to collect useful posts from Discord; Specifically looking at three microservices: a database, a web application, and a discord bot.

The document will explain the purpose and features of the system. What the system will do, the constraints under which it must operate, and how the system will react to external stimuli and handle errors. This document is intended for stakeholders, developers, and users of the systems.

1.2 Scope

The objective of this project is to develop a Discord bot that works with a web app that consolidates specified posts from Discord. A Discord bot will regularly look for posts specifically marked with an emoji reaction to be offloaded to a web application. The Web application will have a database for the posts, and allow the categorization of the posts. Users can have access to the web application to look up important posts which could be regarding project information, homework, or work related to a specific category. The project can also be developed further by adding more entities to the data and having more categories. Keep it simple to start and expand on it later.

1.3 Definitions, Acronyms, and Abbreviations.

SRS (Software requirement specification document)	A document that completely describes all of the functions of a proposed system and the constraints under which it must operate. For example, this document.
Discord	An application used for text and video communication. Users make and react to posts within the channels of a discord server.
Server/Guild	The spaces on Discord. They are made by specific communities and friend groups. The vast majority of servers are small and invitation-only.

Channel	A small section of a discord server that usually relates to a specific purpose or topic as designated by members of the server.
Bot (Discord Bot)	is a program that operates automated tasks over the Internet as an agent for a user or another program or simulates a human activity.
DB (Database)	Collection of all the information monitored by this system.
Stakeholder	Any person with an interest in the project who is not a developer.
Web Application/Web Service	Web application and Web service refer to the web interface where users can view interesting posts from the database.
Users	are the members of any discord server that will be interacting with the discord bot.
Post	A post is a single message/image/file sent by a user into a discord channel, and any reaction emojis associated with it.
Reaction	A reaction is an emoji that can be added to a post after the post is sent.
Entities	An entity is a piece of data that is attached to a post or a reaction. Entities include information such as: time, date, username, and channel.
TBD	To be determined

1.4 References

- Open Liberty - https://openliberty.io/
- Discord developer documentation - https://discord.com/developers/docs/intro
- Github - https://github.com/PavlAvstin/OZ-CSC-480-HCI-521-Fall-2022
- IEEE. *IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.* IEEE Computer Society, 1998. Obtained via Blackboard

1.5 Overview

The next(second) section, the overall description section, of this document gives an overview of the product's functionality. It describes the informal requirements and is used to establish a context for the technical requirements specification in the next chapter. This section concerns users.

The third section, the Requirements Specification section, of this document is written primarily for the developers and describes in technical terms the details of the functionality of the product. Both sections of the document describe the same software product in its entirety, but are intended for different audiences and thus use different language

The fourth section explains how another team may go about getting changes in the requirements and interact with the requirements engineers as well developers. This section concerns customers, clients, and anyone interested in speaking to the team about the system changes.

The fifth section consists of the log with the version number of the requirement developer and stakeholders to approve the srs.

The sixth section contains UML diagrams use cases, sequence diagrams, more diagrams, stakeholder goals, and extra notes that may be useful to anyone.

2. The Overall Description

2.1 Product Perspective

The system consists of three main components: a Discord chat listener, a web application, and a database to store posts.

The system starts with a bot crawling on a Discord channel picking specific posts which are reacted to with a specific emoji. That chat is picked and sent to a web application database to be stored. The web application can be accessed by authenticated users and it's controlled by admins who have access to deleting and adding a post in and out of the database, Users have access to search for posts by the username of who posted it or search through a specific category of posts to view it.

Discord Bot

- Listening to the chat to pick reacted posts

Web Application
- Authenticate users
- Display posts
- Search posts
- Filter posts
- Sort posts

Database
- Add posts
- Delete posts
- Update posts
- Categorize posts

2.1.1 System Interfaces

The software runs in the latest version of Chrome or Firefox browser on Windows, Linux, and Mac.

The bot will interface with the discord channel, and post information using Discord API and reaction emoji`s id to send it over to the database on open liberty. The web application accesses the data from there to display it to users and admins, who get control over the content displayed.

2.1.2 Interfaces

The web application will be displayed to the user to put in their credentials using Discord or Google Authentication to log in. Users and admins have different features to access. The web application will have a search tab to search different entities supported by the posts picked from the discord guild, such as name, date, and specific tags which could be school, homework, assignment, work-related, important announcement, etc.

2.1.3 Software Interfaces

The three main microservices interface with each other. The Discord bot will be interfacing with the database loading posts that are reacted to sent and the database will be used to display the posts on the web application. The web application will be interfacing with the database and its entities to give the user more options to search.

2.1.4 Memory Constraints

TBD

Maybe in future memory can written depending on the engine team such as database memory limit or server limit

2.1.5 Operations

The post is immediately sent to the database to be displayed immediately on the web application. If the system needs an update it will show up on the screen on the dashboard with instructions to follow as per the update of a specific component or if the web application will be offline for moments.

2.1.6 Site Adaptation Requirements

- Users would need a Discord profile and chat channel
- User will have to be authenticated to access the web application
- User manual will be available after the development process which will instruct the user on how to download and set the system on their Discord server.

2.2 Product Functions

Discord Bot

- Get an active listening Discord bot to check posts for reaction
  - if the post is reacted with a specific emoji and send it to the database,
  - if the reaction is removed the bot will remove the post from the database.
- Adding posts to the database
- Removing mismatched(not reacted to on discord, but still located on the database) posts from the database.

Web application

- JWT will be needed for authentication using Discord or Gmail to access the web application.
- Do not display posts that are no longer marked with a reaction. A good way to implement this would be for the bot to check for mismatches.
- Sort page according to the category

Users

- Search posts
- Filter search with entities of the post
- Send posts in the database to other users in the discord server as a DM.
- Function is only intended to send messages within the same discord server, not to other servers or outside of the organization users.
- Make sure that users don’t have access to those API’s.

Admins

- Sort/categorize posts
- Database admins are only allowed to access the database to manually change it. This role is to ensure that normal users won't be able to access API calls.
- Delete posts
- Edit posts on the database

Database

- Store posts from the discord guild
- supply data to the web application
- allow the change in the database i.e. edit/delete a post from the web application

2.3 User Characteristics

Primary users

- Users of Discord
- Developers who would like to make changes to this software.

Secondary users

- It could be used in the educational discord server, or workspace discord server

2.4 Constraints

- Interface to other applications - The web application login must allow authentication via discord and google. That is to say, allow users to continue to the web app with their discord credentials or google credentials. No other external application interfacing is required.
- Parallel operation - The software must have a microservices architecture, such that the discord bot, database, and web application all run on separate OpenLiberty servers to support statelessness. Additionally, separate bots are required for each discord server that they are listening to for interesting posts.
- Language requirements - Backend services, such as the discord bot, must be written using Java.
- Security considerations - JSON web tokens (JWT) will be used to authenticate users when logging into the web application.
- Server - IBM's OpenLiberty servers must be used to host each of the three microservices.

2.5 Assumptions and Dependencies

- Using JWT for authentication
- Using open liberty for microservices
- Using SQL for Database
- Using supported discord APIs

2.6 Apportioning of Requirements.

Future requirements to expand the software

- first, make it simply functional to get the three main systems functioning effectively
- implement either one types of authentication via Gmail or discord
- have feature emoji represent the specific category in the database
- have more different ways to send posts in dm`s
- having more entities in the DB to sort and search according to it

3. Specific Requirements

3.1 External Interfaces

- Discord bot interfaces with chat posts as an active listener on the discord server. It stores all posts from the server but only reacted-to posts are sent to the database, such that they will then be able to be viewed in the web application.
- Database is using SQL queries to store posts and their entities which interface with the bot and web application.
- Security interfaces with the web application via Google or Discord Authentication. JWT web tokens will be used for user authentication to login through those credentials and have access to the web application. Those credentials also differentiate between user and admin roles, where admins have access to more functions, such as delete, sort, and categorize posts.
- The web application interfaces with the database to get search results or look up a post in a specified category.

3.1.1 Human Actors

- **Normal Users:** Discord guild members make up all normal human actors(users) interfacing with the software.
    - Actors will interface solely with the web application.
    - Login to the webpage can be completed with user discord or google credentials.
    - Users should be able to filter the posts collected therein by usernames, discord channel, post content, post date/time, and by keywords.
    - Users should be able to search posts within the web application.
    - Users should be able to send posts from the web application to other users in the same discord server as a direct message(DM).

- **Admin Users:** Admin users retain all the abilities of a normal user, with the addition of some administrative permissions.

    - Admin users should be able to manually delete posts from the database, even if they have been reacted to on discord

        - This permission does not remove the reaction from the post in discord.

    - Admin users should be able to manually change the content category of a post within the database.
        - This permission does not change the originally designated content category of the reacted-to post in discord.

3.1.2 Software System Actors

- **Discord:** Discord is an application used for text and video communication. Users make and react to posts within the channels of a discord server.
    - **Discord Bot:** The discord bot "listens" to all posts and reactions made in a discord server as a whole.
    - Whenever a post is made, or a reaction is added/removed from a post, the discord bot records that event along with a series of entities, such as date, time, channel, username, and reaction type.
    - If the reaction type on a post is one of the emojis that designate a post as "interesting", that post is added to the database, and vice versa for removal of a reaction.
    - Different emojis will be utilized to designate a post as interesting and also assign the post to a predetermined content category.

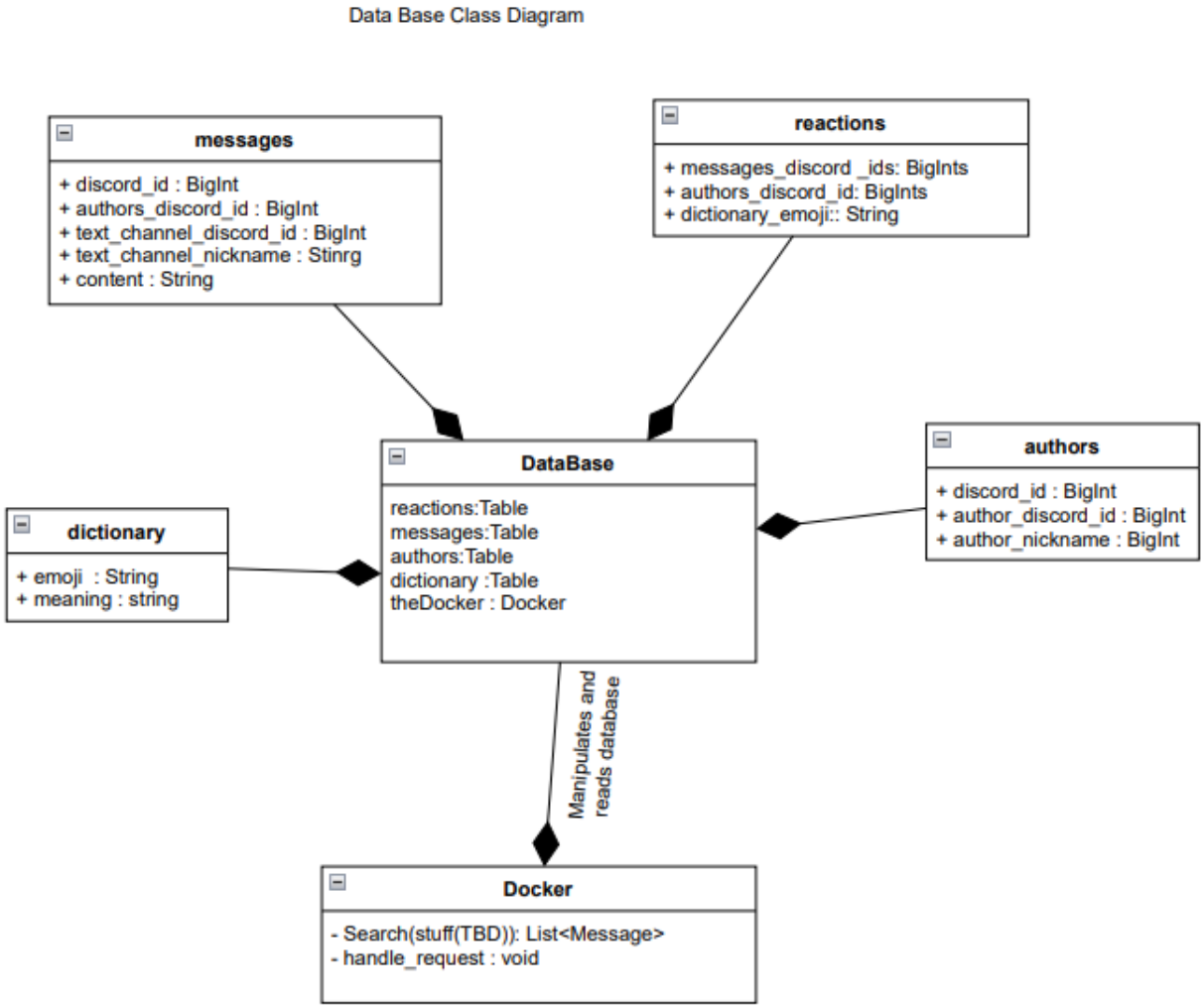
- **Database:** The database is a collection of all the information recorded by the discord bot. Our database utilizes SQL to manage that data.
    - The database uses SQL to store and manage interesting discord posts and their associated entities in a series of tables, each containing different entities of the discord posts.
    - The tables are entitled: reactions, messages, authors, and dictionary
    - Multiple tables are used to connect different aspects of posts with others in the database.

- **Web Application:** The web application is a web page that users can access to view, sort, filter, and search through all the information in the database with a user interface.
    - Access to the application will be granted with a login using user's discord or google credentials and JWT for security.
    - All posts marked as interesting that are currently in the database are displayed in the web application
    - The web application needs to contain functionality to sort and filter these posts by:
        - The username of the user who marked the post as interesting with a reaction emoji.
        - The date/time that the post was originally made.
        - The discord channel that the post was made in.

        - The content category that the post was designated to when the reaction was added in discord.
    - The web application must have a search bar to search for posts that contain keywords, and/or have one of the entities associated with them as described by the above filters.
    - The web application must provide an option to send singular posts back into the same discord server as a direct message(DM).

3.2 Diagram Derived Description

Database Class Diagram



For other diagram look at section 6

3.2.1 Discord Bot

The Discord Bot will be running on a Docker
The Discord Bot will have 4 listening parts;

1. When a Post is Reacted too.
2. When a Post is deleted.
3. When a Post reaction is removed.

The Discord Bot will be constantly listening.
The Discord Bot shall send a message to the Database Docker when it hears something it was listening for.
The Discord Bot message shall be made up of 5 Attributes;

1. Discord Id
2. Author discord id
3. Text channel discord id
4. Text channel nickname
5. Content of post (the actually String/Chars)

The Bot will use HTTPS with appropriate modifications for our implementation.

3.2.2 Database

The Database will be written in SQL.
The Database will have 4 tables;

1. Reaction
2. Messages
3. Authors
4. Dictionary

There will be 3 Dockers

1. Hosting the database will
2. Hosting OpenLiberty
3. Hosting the Bot

The database will have a Docker that manages incoming/outgoing requests.
The Database Docker will use HTTPS with appropriate modifications for our implementation.
The Database Docker will communicate with the Web App and Discord Bot.

3.2.3 Web Applications

The Web App will have a login interface.
The Web App will have a way to filter posts.
The filters Are as follows;

1. User defined Category
2. Post date/time
3. User
4. Channel

The Web App will have a search feature.
The Web App will send messages to the Database Docker.
The Web App will receive messages from the Database Docker.

The Web App will use HTTPS with appropriate modifications for our implementation.

The Web App will have admin users.

Admin users will be able to delete posts

Admin users will be able to edit Categories of posts.

3.3 Performance Requirements

TBD

3.4 Design Constraints

- No cost constraints for hardware,
- If an admin deletes a post from the web application it will be deleted on the database but the bot will not be able to undo the reaction on the Discord server.

3.4.1 Standards Compliance

- Database entities must have the following entities Username, Discord channel, content of the post, and date of post

3.5 Software System Attributes

3.5.1 Reliability

TBD

3.5.2 Availability

- Discord bot should actively listen to the Discord server 24/7 sending reacted posts to the database.
- The web application should be available 24/7 for users to login and view interesting posts.
    - If there is any update or maintenance going on with the page an error page would show up describing the issue.
- Database should be updated by the bot to add post and the admin on the page should have access of all the features 24/7

3.5.3 Security

- Using Google or Discord Authentication to protect the web application from unauthorized users.
    - TBD

3.5.4 Maintainability

- Web application - Will be maintained by the admins of the page.
    - Admins have control and access to the web application to filter, sort, and delete posts that the discord bot puts into the database. Admins will be the "human filter" to the database made by the bot.
- Database - can be maintained by a developer looking at the ER diagram of the database and seeing that discord puts in the most data but the admins access to make changes to it
- Discord bot is on a discord server actively listening and interfacing with discord API.

3.5.5 Portability

- Host needs to set up a server for the web application and database.
- Users can react and post on discord from any device anywhere given availability of internet connection.
- Users can access the web application from any device with an internet connection.
- The web application is formatted for use on a desktop monitor, so smaller screens may have difficulties with usability.

3.6 Additional Comments

- An improved version of this document with more detailed specifications will be available as version 2. Until then if there is any specific question or constraint related to this document, that can be resolved through meetings with the requirements team.

4. Change Management Process

During sprint 2, all teams are encouraged to voice questions, comments, and concerns regarding all pieces of the SRS in the designated “SRS Q&A” thread on the requirements text channel. If a team requests for changes to specific software requirements, arranging a meeting with the requirements team is all that is needed for those changes to be heard and evaluated.

After sprint 2, if another team requests for changes to be made to the software requirements and/or has new requirements, they will have to make an appointment to meet all members of the requirements team. A formal written descriptive document detailing what they would like changed/added shall be submitted while making the appointment. A properly written document should include: The specific requirement that needs change (if there is any), a well written proposed requirement, and the reason/s for the change or addition. Only then will the other team's demands be heard and an evaluation on if they can be implemented will occur. We will have to double check if there are any other constraints or stakeholder demands that encourage or inhibit the request and if the requirement/feature is possible and compatible with the system.

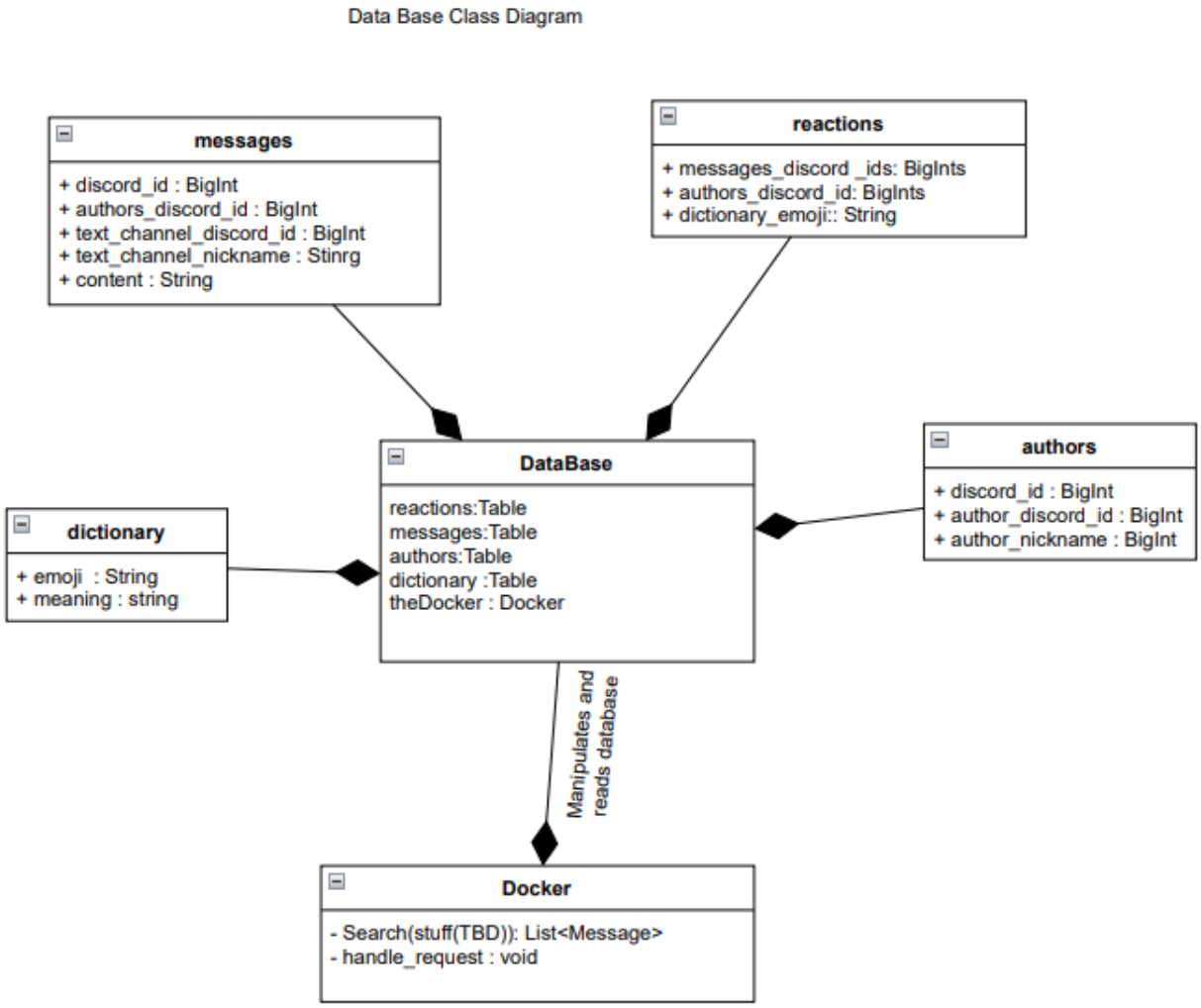
5. Document Approvals

SRS version	Name	Signature	Date
1	Noah Henwood (developer)		/ / 2022
1	Jonathen Germakovski (developer)		/ /2022
1	Umang Patel (developer)		/ /2022
1	Paul Austin (stakeholder)		/ /2022
1	Adam Yoho (stakeholder)		/ /2022
1	Rumana Haque (stakeholder)		/ /2022

6 Supporting Information

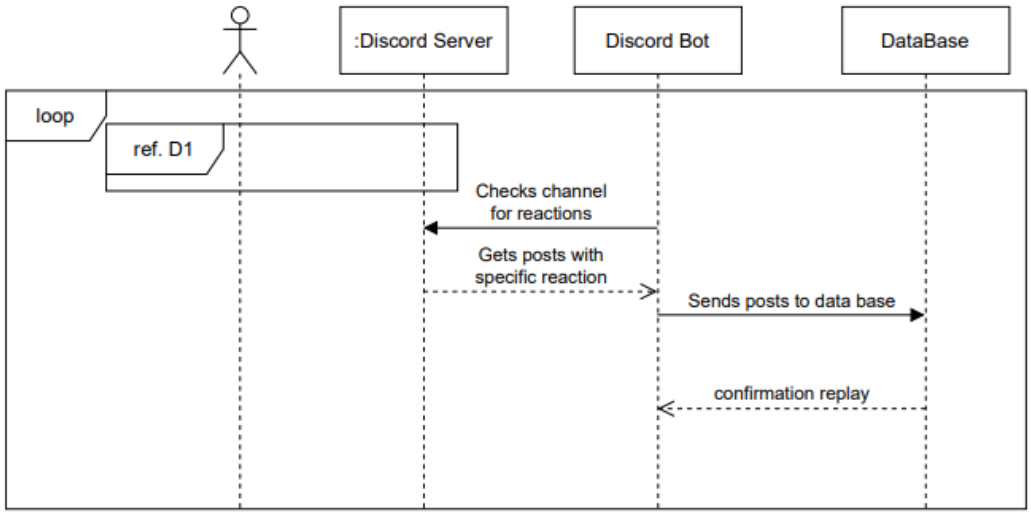
6.1 UML Diagrams

6.1.1 Database Class diagram

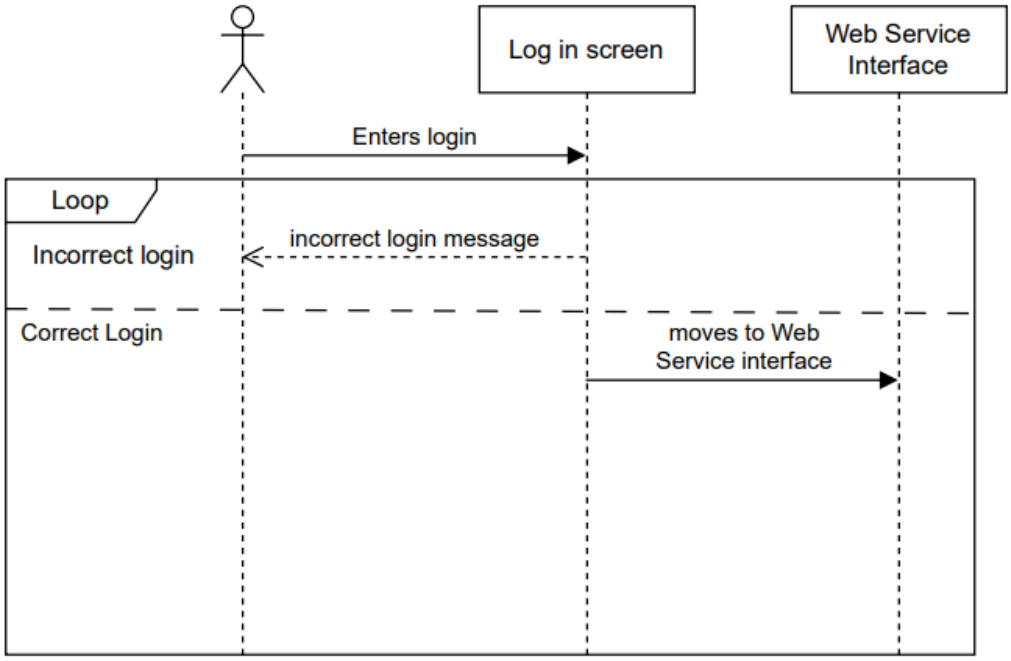


6.1.2 Sequence Diagram

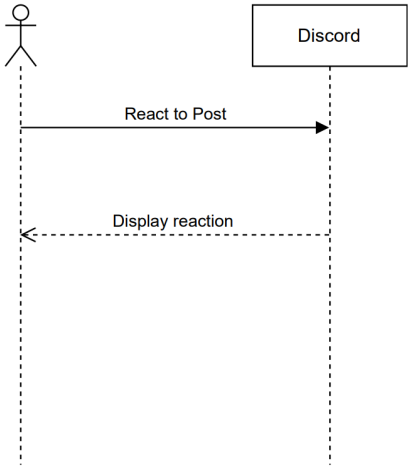
6.1.2a BSD



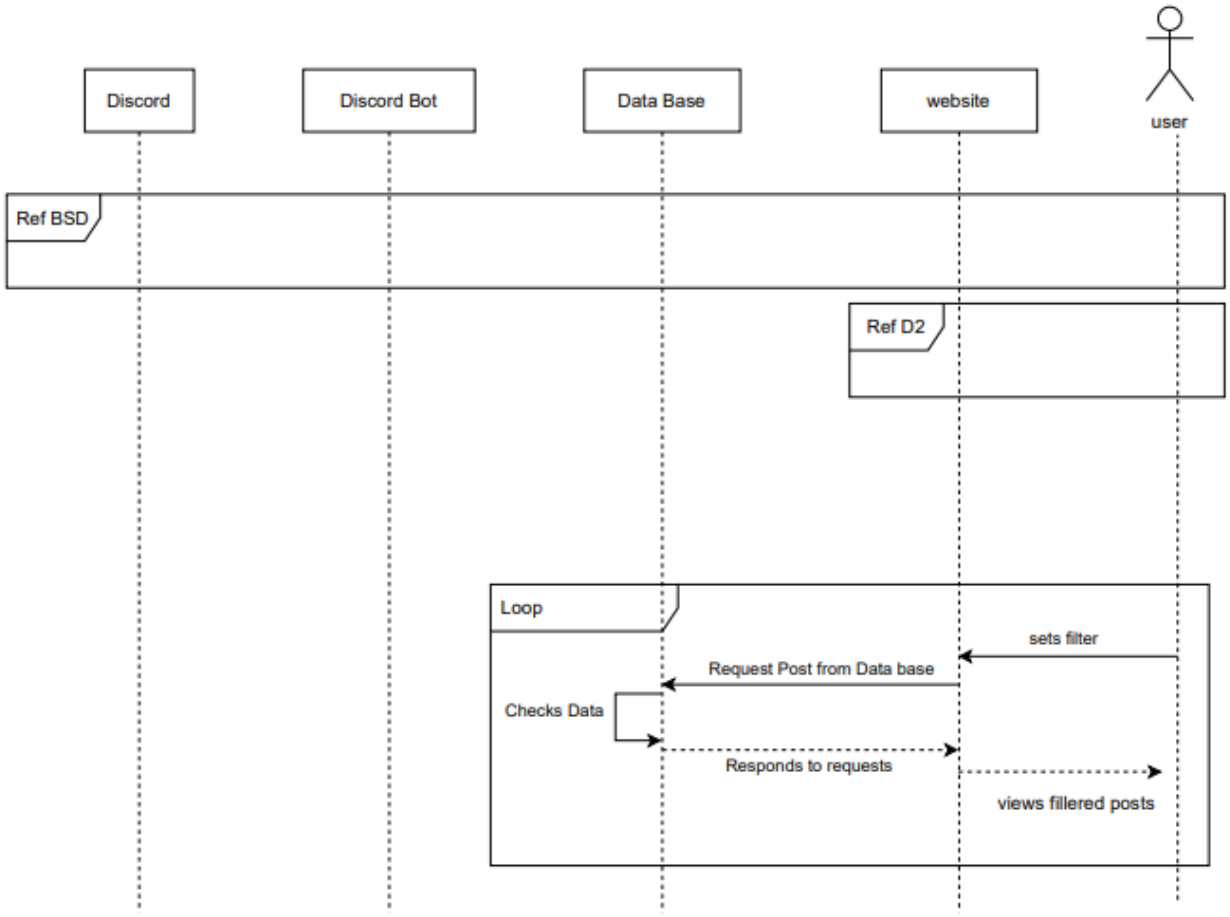
6.1.2b D2



6.1.2c D1

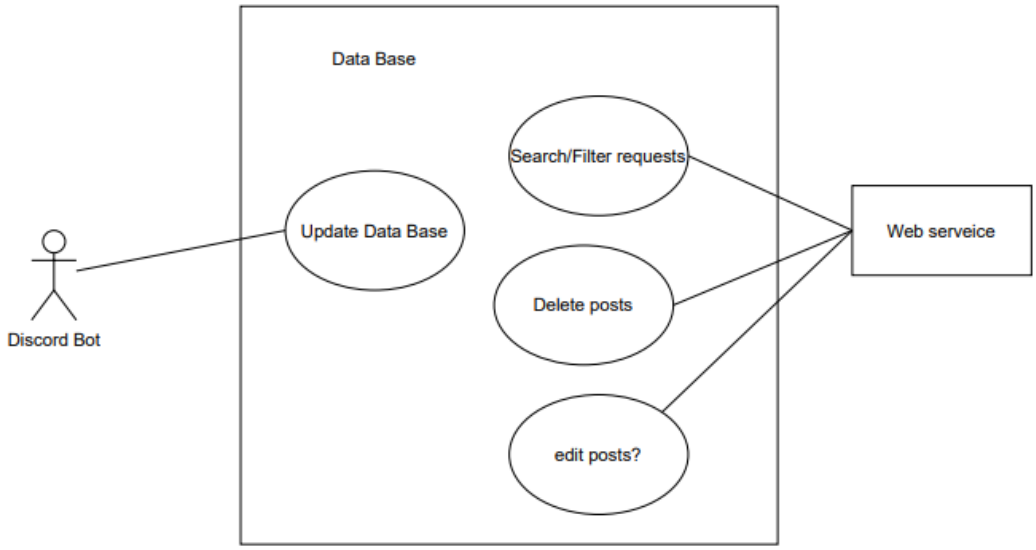


6.1.2d Main

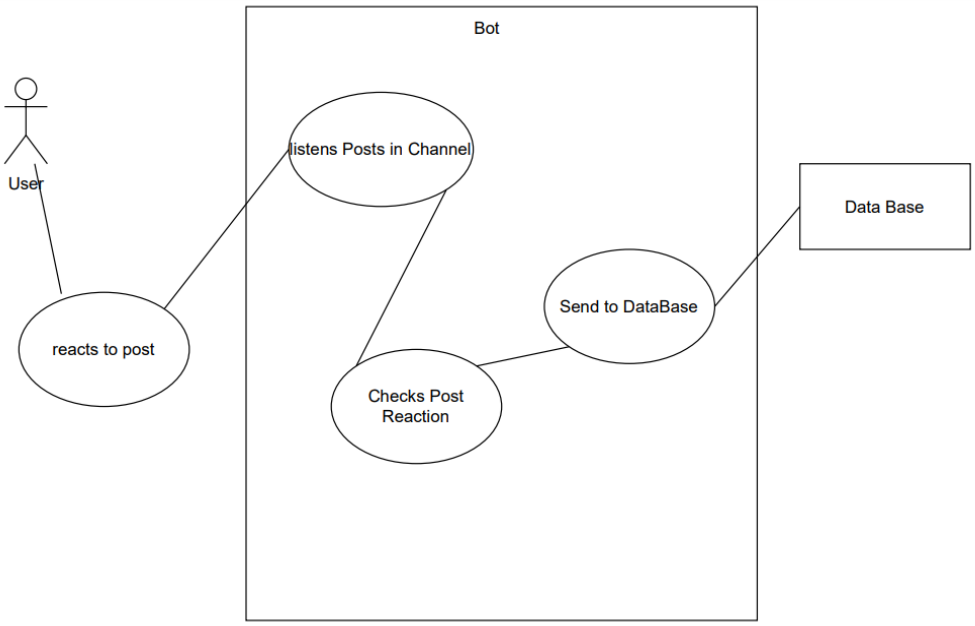


6.1.3 Use case Diagram

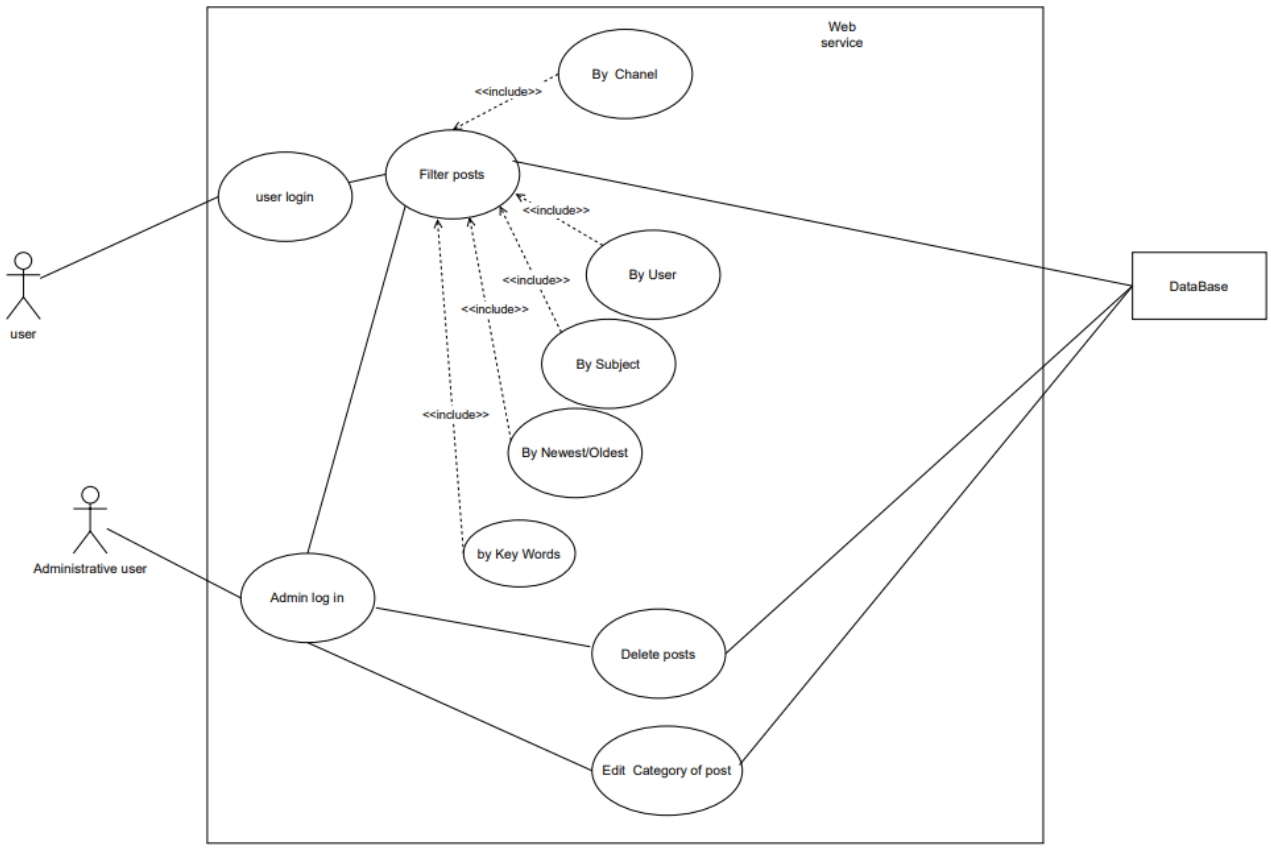
6.1.3a Data base



6.1.3b Bot



6.1.3c Web application



6.2 Goals from stakeholder

1. Implement three systems using a microservice architecture.
2. Create a system to find all Discord posts in a given channel that have a specific reaction.
3. Periodically search Discord to find appropriate posts and store them in a database.
4. Database must provide HTTP CRUD APIs for interaction.
5. Only database admins and the required microservices must be allowed to interact with the database.
6. Display all interesting posts on a web page.
7. Only display the web page to authenticated users.
8. Users should be authenticated via Discord or Google.
9. All microservices should run on separate Liberty servers.
10. Display only the posts that were tagged as interesting by a specific user.
11. Display only the posts that were tagged as interesting by the authenticated user.
12. Allow the system to categorize posts (e.g. This post relates to homework, meetings, technical details, etc.)
13. Find all interesting posts in a Discord guild (rather than just a specific channel).
14. Allow the user to sort/filter/search posts on the UI.
15. Share posts via DM directly from the web page.
16. Do not display any posts that are no longer marked as interesting.

Technical Requirements:
1. Use IBM's Open Liberty application server to host your application.
2. This will be a web app - no mobile requirements.
3. Use microservices architecture with at least the suggested microservices.
4. Backend service(s) written in Java.
5. Frontend service(s) written using any library you prefer.
6. Use JSON web tokens (JWT) for security.