CSC 480 / HCI 521 -22F Software Design

Postmarker
By Full Stack Attack

Software Requirements Specification Document

Prepared By: Noah Henwood, Jonathen Germakovski & Umang Patel

Version: (2) **Date: (9/30/2022)**

IEEE. *IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.* IEEE Computer Society, 1998. Obtained via Blackboard from Prof. Bastian Tenbergen

Table of Contents

Document Approvals

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

1. Introduction

1.1 Purpose

Postmarker is a discord bot that locates, records, and displays posts from users. It was developed for a class at SUNY Oswego to educate a class of CSC and HCI students.This SRS serves to document the Discord Bot, relevant DataBase, and WebService. The intended audience is the developers of said systems such that they will be able to implement these systems in future implementations.

1.2 Scope

This system entails the development of a Discord bot, Database, and Web Application, using OpenLiberty. The scope of the system is the following:

- The Discord bot shall regularly look for posts specifically marked with an emoji reaction to be offloaded to a web application
- The Web Application shall have a database for the posts
- The Web Application shall allow the sorting/filtering/categorization of the posts
- Users shall have access to the web application using discord login credentials.
- Users shall have the ability to look up important posts which could be regarding project information, homework, or work related to a specific category through the Web Application.

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	is an application used for text and video communication. Users make and react to posts within the channels of a discord server.
Discord 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/Dilbert (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 attached to a post or a reaction. Entities include information such as time, date, username, and channel.
Message	The communication method between the Discord bot and Database in order to specify that a post has been created or manipulated in some way.
TBD	To be determined
Authentication	prove who you are
Authorization	check what you are allowed to do
JSON	JavaScript Object Notation

JWT	JSON web token
DM	Direct message
API	Application programming interface
CRUD	Create, Read, Update, Delete. Refers to the functionality of API's.
Docker	Docker is an open source platform that enables developers to build, distribute, operate, update, and manage containers. Containers are standardized, executable components that integrate application source code with the operating system libraries and dependencies necessary to run that code in any environment.
OpenLiberty(Liberty)	A lightweight open framework for building fast and efficient cloud-native Java microservices

1.3.2 Index for Requirements

The requirements (section 3) will follow the conventions listed here:

FRQ-# Abstract Goals
FRQ-#.# Sub Goal
FRQ-#-#-# Solution Oriented Requirement
SCE-#-#-# Scenario

1.4 References

- Open Liberty - https://openliberty.io/
- Javacord https://javacord.org
- 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
- JavaScript: version ES6 https://developer.mozilla.org/en-US/docs/Web/JavaScript - The main programming language for the GUI to retrieve information on the webpage.
- HTML5 https://developer.mozilla.org/en-US/docs/Web/HTML - To build the layout of the webpage

- CSS3 https://developer.mozilla.org/en-US/docs/Web/CSS - To format how our program looks on the webpage
- OpenID Connect (OIDC) Diagram - Repurposed from https://auth0.com/docs/get-started/authentication-and-authorization-flow/authorization-code-flow

1.5 Overview

Section 2 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 Section 3. This section concerns users.

Section 3, is written primarily for the developers and describes in technical terms the details of the functionality of the product. Sections 2 and 3 describe the same software product in its entirety, but are intended for different audiences and thus use different language

Section 4 contains UML diagrams use cases, sequence diagrams, more diagrams, appendixes, and extra notes that may be useful to anyone.

2. The Overall Description

2.1 Product Perspective

2.1.1 System Interfaces

The software runs on the current version of Chrome or Firefox browser on Windows, Linux, and Mac as of October 2022.

The Discord bot will interface with a discord channel and post information therein using Discord API's and reaction emoji`s. Specific reaction emoji id's indicate that a post will be sent to the database on open liberty. The web application accesses the data from there to display it to users and admins, who get the ability to filter the content displayed.

2.1.2 Interfaces

The web application will be displayed to the user to put in their credentials using Discord 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 Discord bot interfaces with the database by sending user posts. The database interfaces with the web application to display the posts. The web application interfaces with the database and its entities to give the user options to filter posts.

2.1.4 Operations

Reacted to discord posts in a server that the discord bot is operating within are sent to the database. The Database puts the post into different tables corresponding to the category the reaction applies too. The database is accessed using a modified restful API by the web application. The web application displays the posts and can change what posts are shown by applying certain filters.

2.2 Product Functions

PF-1) Discord bot will listen for posts' reactions and send them to the database.
PF-2) Discord bot will remove a post from the database if it gets unreacted
PF-3) Users will be authenticated via Discord to access the web application
PF-4) Web application will not display posts that are no longer marked with a reaction.
PF-5) Web application will sort posts according to the categories
PF-6) Users will be able to search posts
PF-7) Users will be able to filter the search with entities of the post.
PF-8) Users will be able to send posts from the Web Application as a DM on discord
PF-8.1) Function is only intended to send DM within the same discord server, not to other servers or outside of the organization users.
PF-9) Admins will have access to slash commands to manually delete posts from the database
PF-10) Database shall store posts from the discord guild
PF-11) Database shall supply data to the web application
PF-12) Database shall allow changes in the database by Admins

2.3 User Characteristics

This system is intended for Discord users that would like to keep track of user-specified posts in a server and share these posts to users in the same discord server. These posts are not shareable across servers and are server specific.

Users

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

2.4 Constraints

2.4.1 Design constraints

- JWT must be used for a Web application to authenticate users
- All three microservice systems must run on separate OpenLiberty servers
- All backend systems must be written in JAVA
- Admins and required microservice must have access to interact with the database

2.4.2 Capability constraints

- If an admin removes a post from the database it will not remove the original reaction or post from Discord.
- The web application is designed for desktop use, and has no requirements for a responsive web page or mobile use.

2.5 Assumptions and Dependencies

- We are assuming that the Discord API Javacord is dependable
- We are assuming that Discord will maintain APIs used for making this software

3. Specific Requirements

FRQ – 1) Implement three systems using a microservice architecture.

FRQ – 1.2) All microservices should run on separate Liberty servers.

FRQ – 2) The Discord Bot shall find all Discord posts in a given channel that have specific reactions. REF: **4.1.1b**

FRQ – 2.1) The Discord Bot will constantly be Listening to all posts.

FRQ - 2.1.1) The discord bot “listens” to all reactions made to posts in a discord guild.

FRQ - 2.1.2) The discord bot “listens” to all deletions made to posts with reactions in a discord guild.

FRQ - 2.1.3) The discord bot “listens” to all edits made to posts in a discord guild.

FRQ - 2.1.4) The discord bot “listens” to all reaction removals made to posts in a discord guild.

FRQ - 2.2) The discord bot makes http requests to our API running on OpenLiberty. OpenLiberty accesses the database then returns the requested data. REF: **4.1.2**

FRQ - 2.3) Events listened to by the discord bot are associated with entities, such as date, time, channel, user name, and reaction type.

FRQ - 2.3.1) 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.

FRQ - 2.3.2) Different emojis shall be utilized to designate a post as interesting and also assign the post to a predetermined content category when it is reacted to.

FRQ - 2.4) The Discord Bot will be running on a Docker hosted with Open Liberty.

FRQ - 2.5) The Discord Bot message that is sent to the database 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 actual String/Chars)

FRQ - 2.6) The bot shall be able to list, add, and remove the specific emoji-content pairs via slash(/) commands in the discord guild.

FRQ - 2.6.1) /dictionary lists all of the specified emojis and their associated meanings.

FRQ - 2.6.2) /meaning *reaction* explains the meaning of the entered emoji.

FRQ - 2.6.3) /add pair *reaction meaning* sets the meaning of a new reaction.

FRQ - 2.6.4) /remove pair *reaction meaning* removes the specified emoji - meaning pair from the dictionary.

FRQ - 2.6.5) /remove message *id* removes a message from the database based off of the provided message id.

FRQ – 3) The database will store Discord posts with specific reactions. REF: **4.1.1**

FRQ – 3.1) Database must provide HTTP CRUD APIs for interaction.

FRQ – 3.2) Only database admins and the required microservices must be allowed to interact with the database.

FRQ - 3.3) The database uses MySQL to store and manage interesting discord posts and their associated entities in a series of tables, each containing different entities of the discord posts.

FRQ - 3.3.1) The tables are entitled: reactions, post content, authors, and dictionary

FRQ – 4) Create a web application that displays all the stored posts from the database on a web page.

FRQ – 4.1) Only display the web page to authenticated users.

FRQ – 4.1.1) Users should be authenticated via Discord. REF: **4.5**

FRQ – 4.1.1.1) When users login to the web app, Liberty goes through an OAuth flow and then makes another backend call to get information about the user associated with the access token.

FRQ - 4.1.1.2) Discord returns an access token to the user, but does not return back an ID token.

FRQ - 4.1.1.3) Liberty uses the access token from Discord to get information about the user and create the login session.

FRQ - 4.1.2) Must use JSON web tokens (JWT) for security.

FRQ – 4.2) Users shall be able to sort/filter/search for posts on the web page via the following methods.

FRQ - 4.2.1) The user name of the user who marked the post as interesting with a reaction emoji.

FRQ - 4.2.2) The date/time that the post was originally made, or when they were most recently edited.

FRQ - 4.2.3) The discord channel that the post was made in.

FRQ - 4.2.4) The predefined content category that the post was designated to when the reaction was added in discord.

FRQ - 4.3) The web application shall 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.

FRQ - 4.4) Find all interesting posts in a Discord guild (rather than just a specific channel).

FRQ - 4.5) Share posts via DM directly from the web page.

FRQ – 4.5.1) DM's sent from the web page shall only be sent to members of the original discord guild the post came from.

FRQ - 4.5.2) DM's sent to other users will be sent from the authenticated user's discord profile.

FRQ - 4.6) Do not display any posts that are no longer marked as interesting.

FRQ - 4.7) The Web App will make fetch requests to the database API. REF: **4.2.4**

FRQ - 4.7.1) Requests made by the web app will return a JSON file which contains information about posts and their entities.

FRQ - 4.7.2) The JSON file is then decoded to access the data sent from the database.

FRQ - 4.8) The Web App will use HTTPS with appropriate modifications for our implementation.

FRQ - 4.9) The Web App shall be available to users 24/7.

FRQ - 4.10) The web application shall be formatted for use on a desktop monitor, so smaller screens may have difficulties with usability.

FRQ - 5) Backend service(s) written in Java, such as coding related to the discord bot and database.

FRQ - 6) Users need a Discord Profile and Discord Server to utilize Postmarker.

FRQ - 6.1) Discord guild members make up all users interfacing with Postmarker.

FRQ - 6.2) All users shall interface with Postmarker solely via discord and with the web application. REF : **4.1.3**

FRQ - 6.3) Discord users defined in the discord server as “admins” will have access to admin only slash commands.

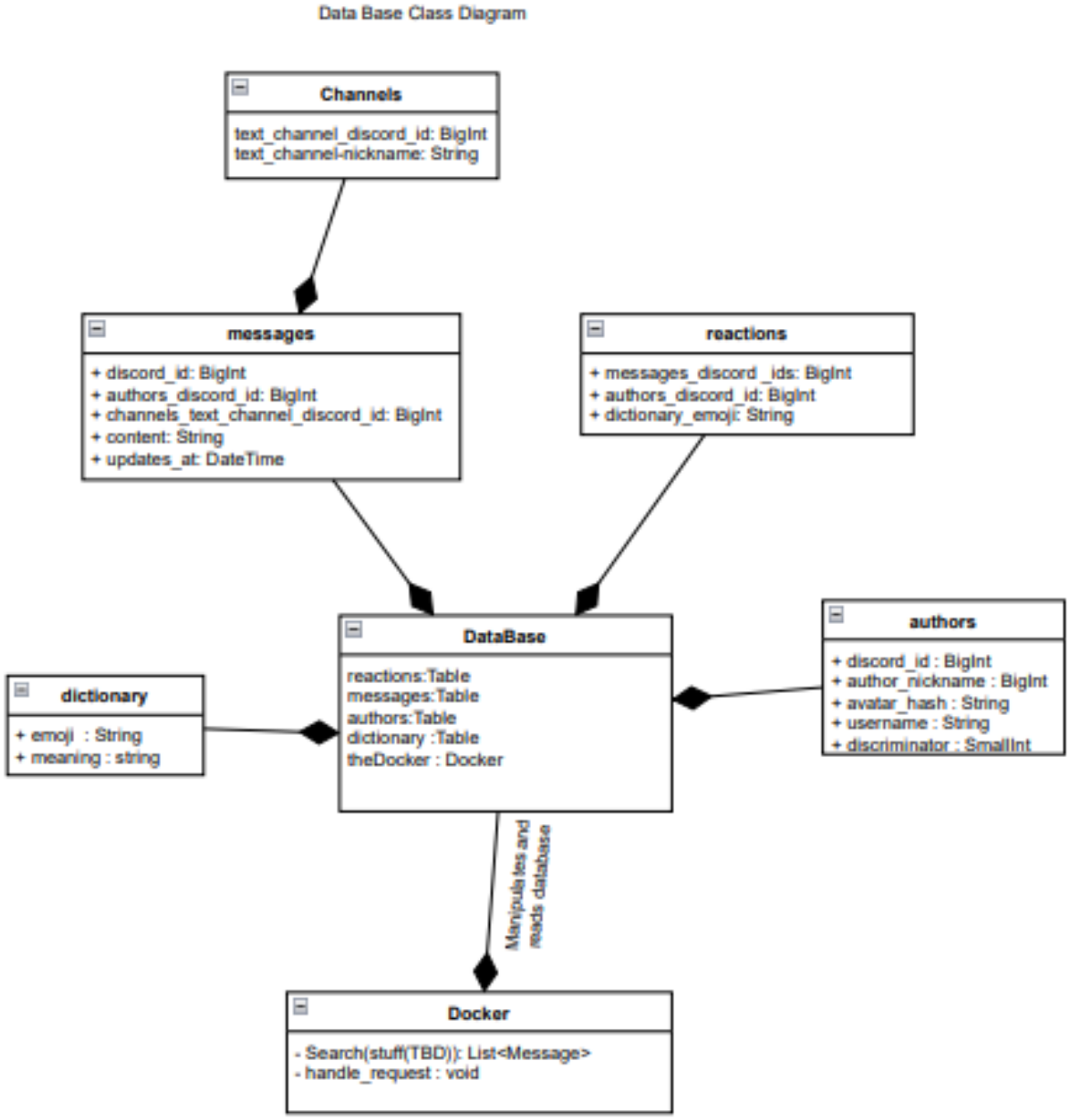
FRQ - 6.4) Admin users retain all the abilities of a normal user, with the addition of access to the add pair, remove pair, and remove message slash commands.

FRQ - 6.3.1) Admin users shall be able to delete posts from the database using /remove message, even if they have been reacted to on discord. This permission does not remove the reaction from the post in discord. REF : **4.1.3**

4 Supporting Information

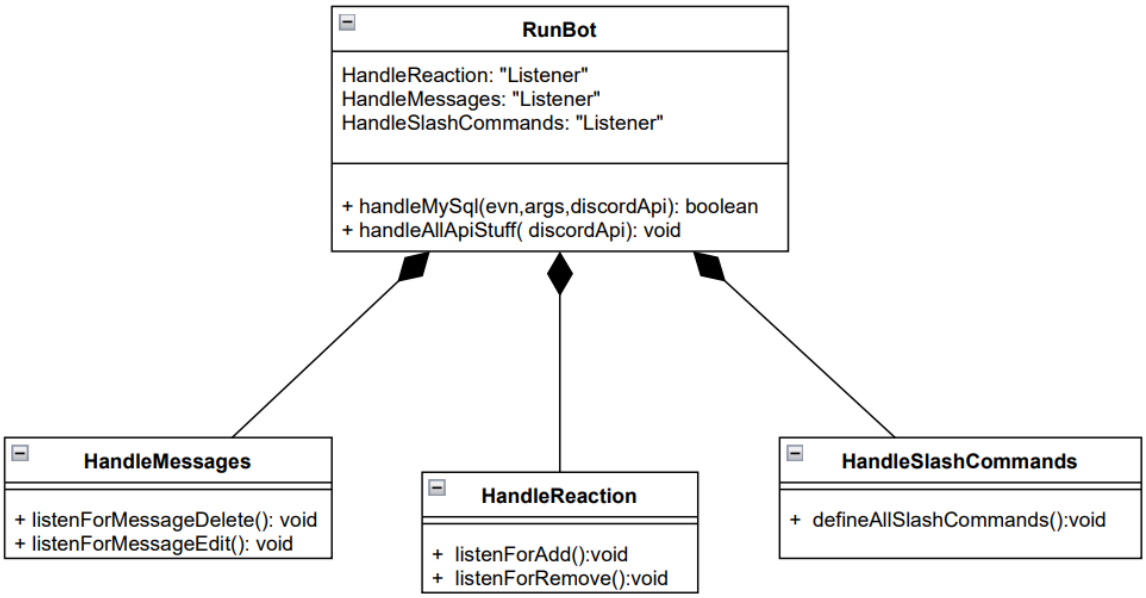
4.1 Class Diagrams

4.1.1 Database Class diagram



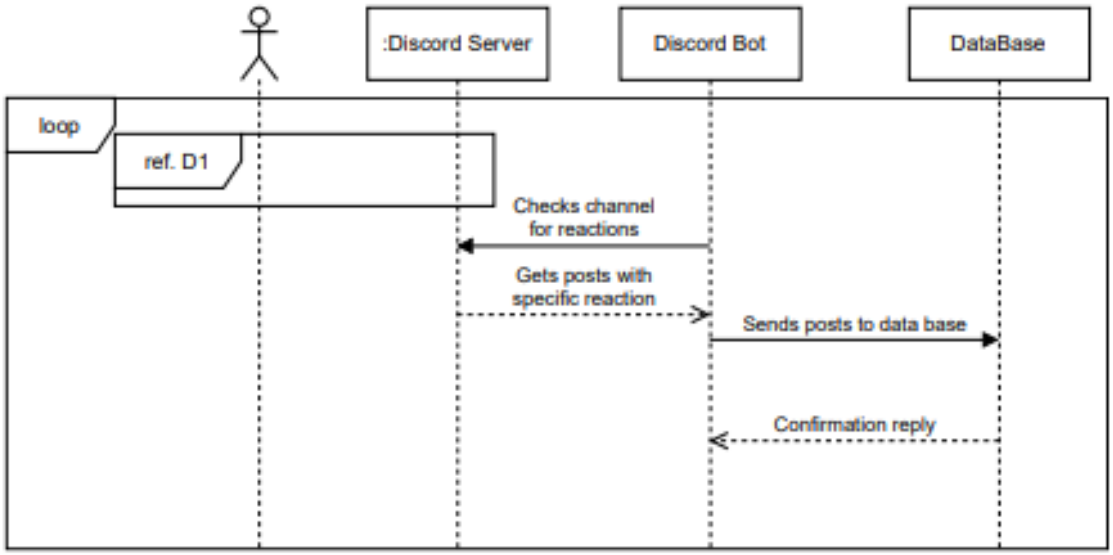
4.1.2 Discord Bot Class diagram

Discord Bot Class Diagram

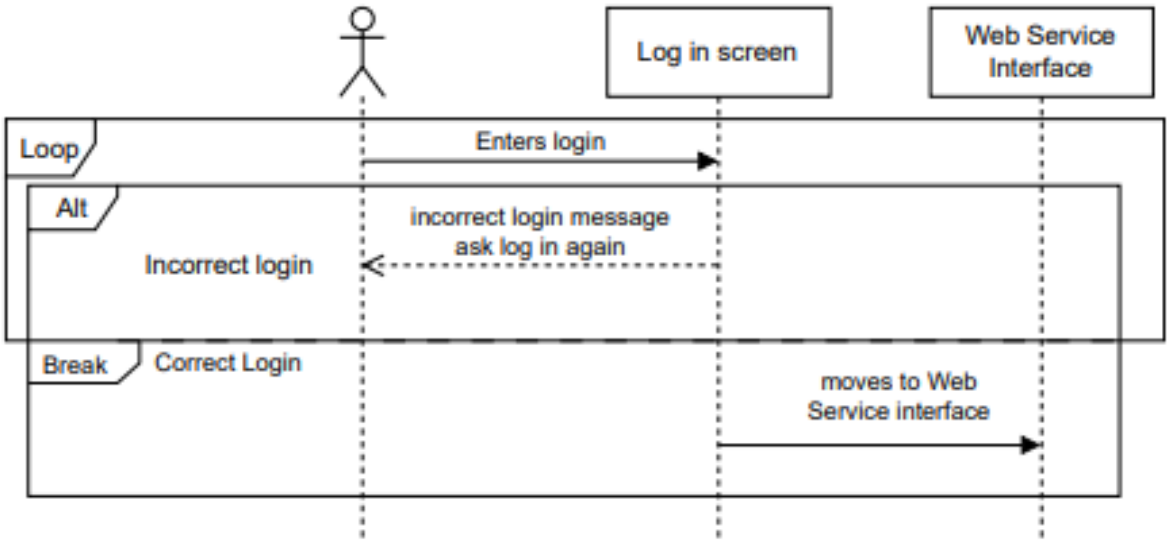


4.2 Sequence Diagram

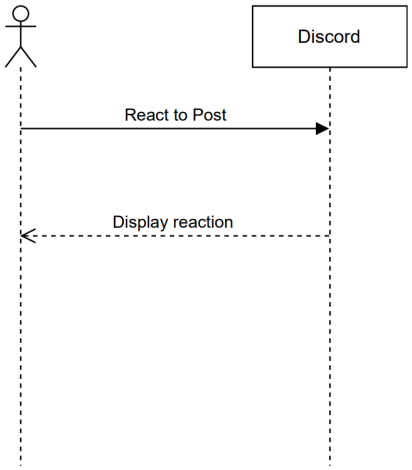
4.2.1 BSD



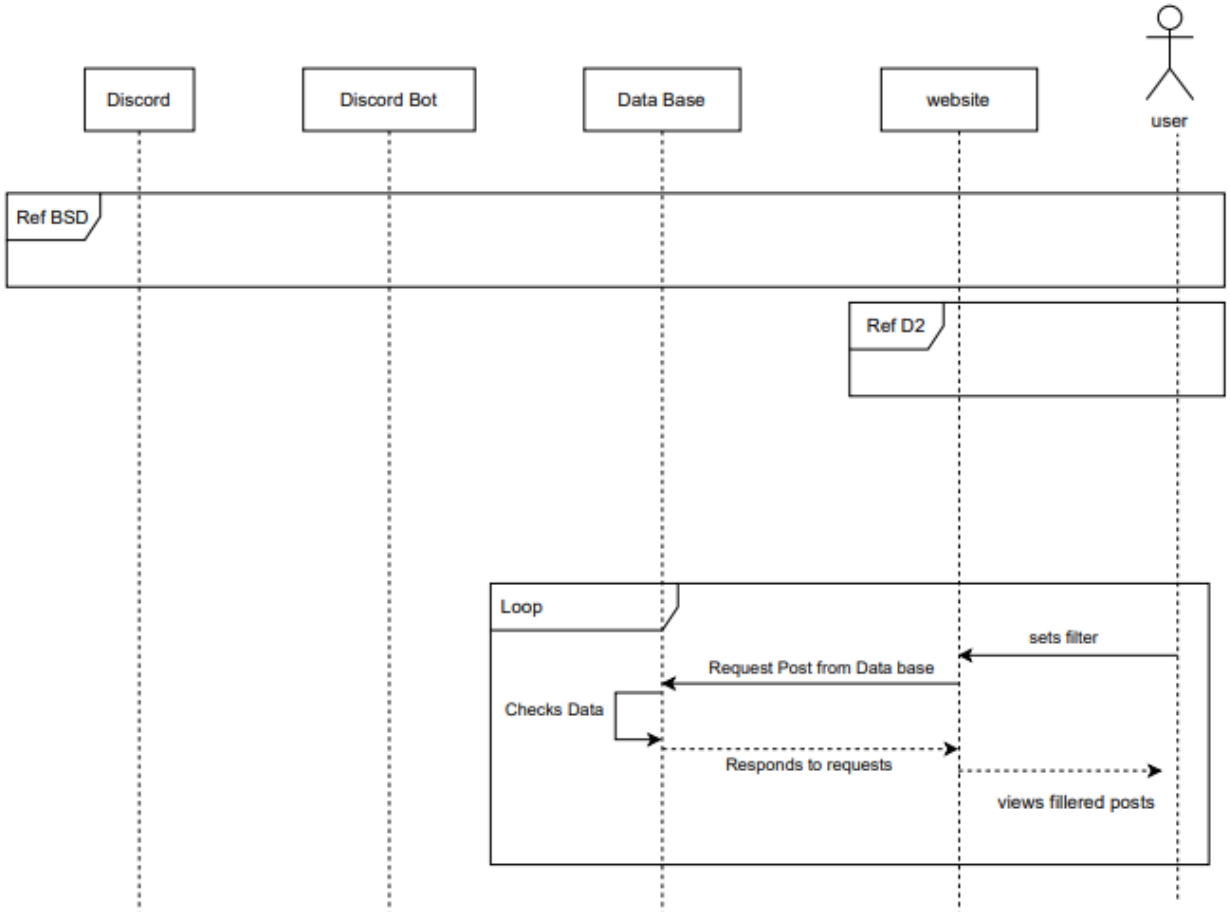
4.2.2 D2



4.2.3 D1



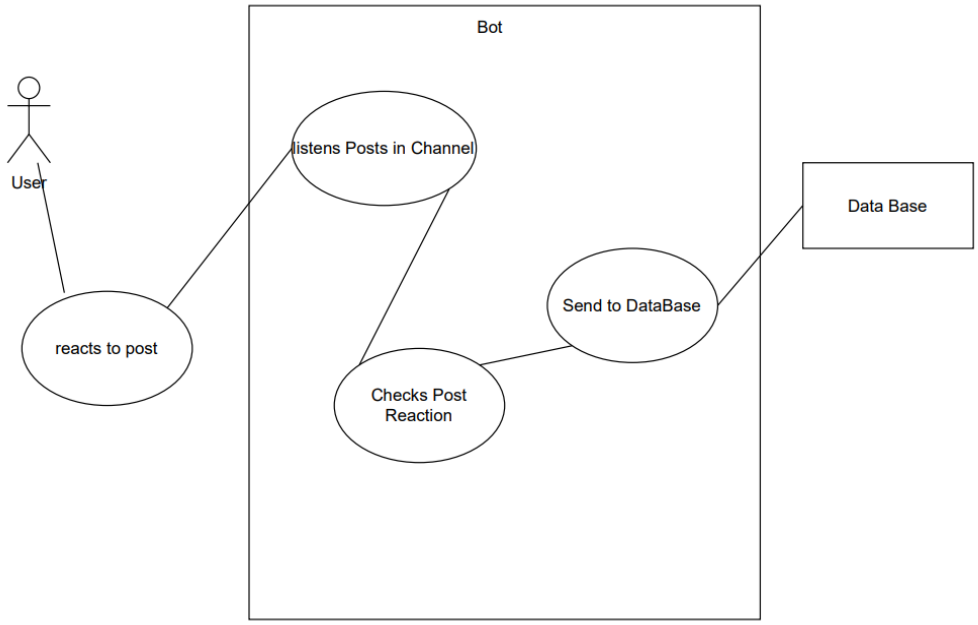
4.2.4 Main



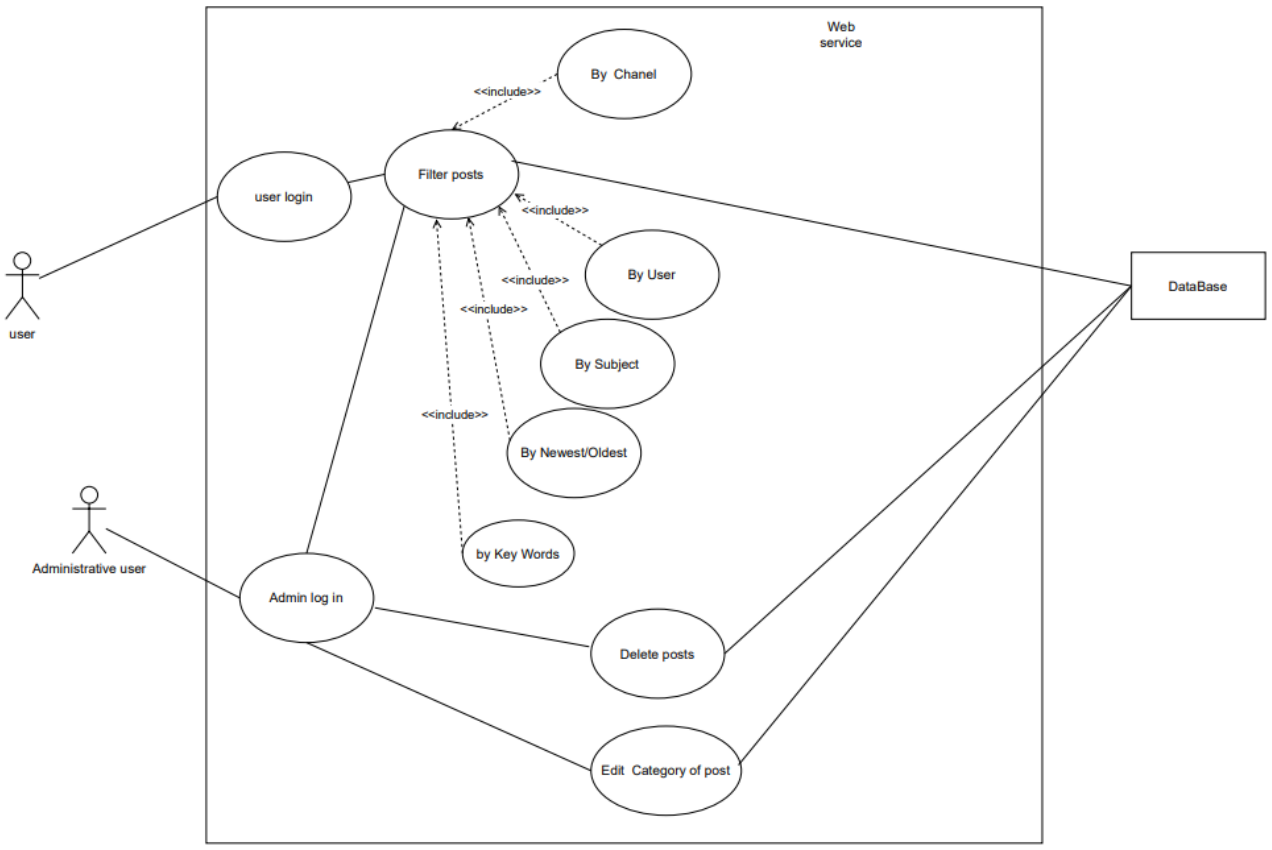
4.3 Use case Diagram

4.3.1 Data base

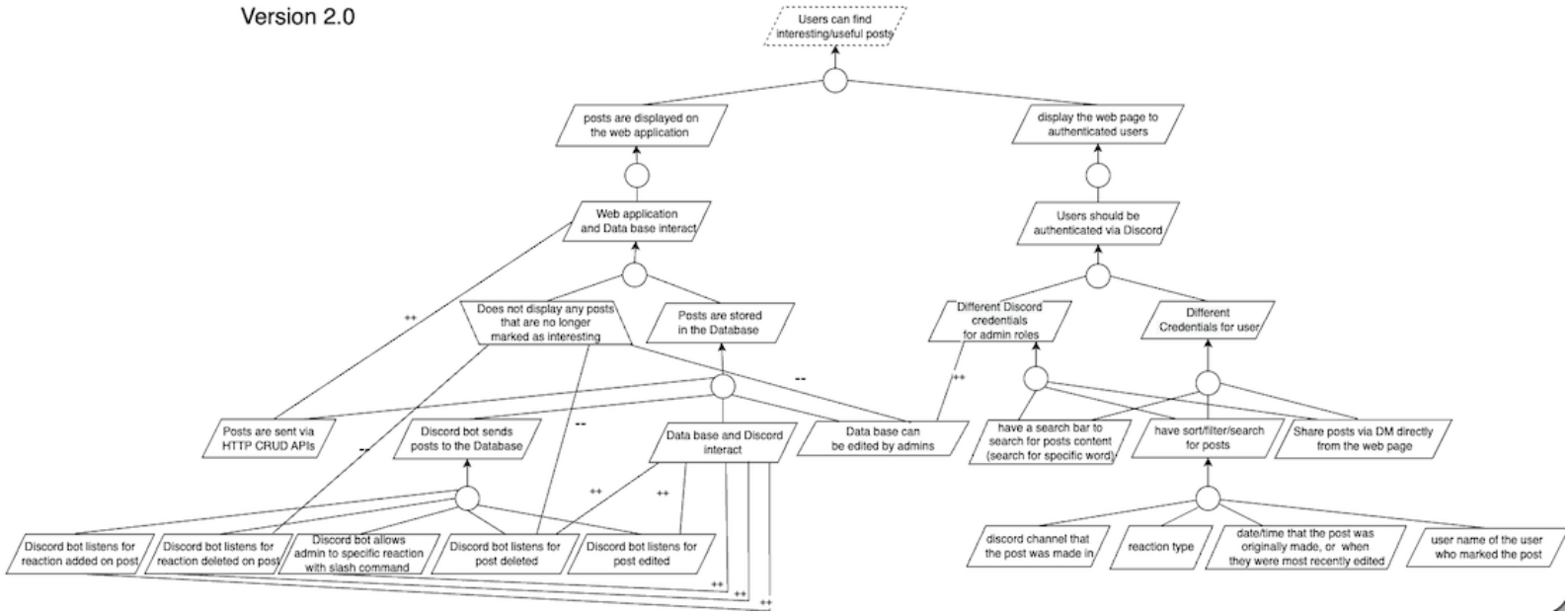
4.3.2 Bot



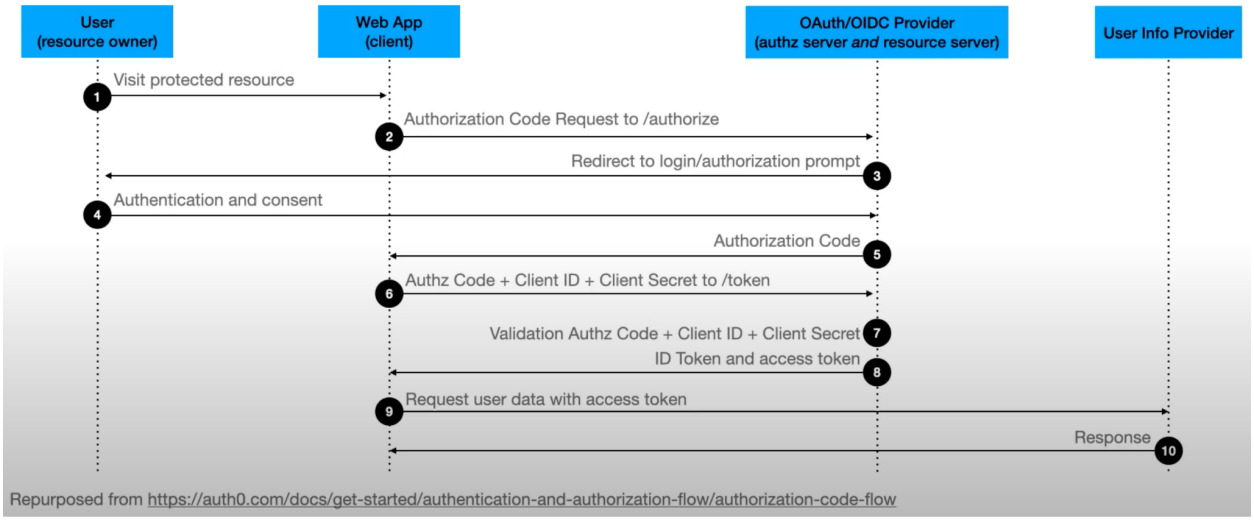
4.3.3 Web application



4.4 Kaos diagram (Goal)



4.5 OpenID Connect (OIDC) diagram



4.6 Goals from stakeholders

4.6.1 Goals

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.

4.6.1 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.