**CSC 480 / HCI 521 -22F Software Design**

**Postmarker**

**By Full Stack Attack**

**Software Requirements Specification**

**Document**

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

**Table of Contents**

**1. Introduction**

*1.1 Purpose*

*1.2 Scope*

*1.3 Definitions, Acronyms, and Abbreviations*

*1.4 References*

*1.5 Overview*

**2. The Overall Description**

*2.1 Product Perspective*

2.1.1 System Interfaces

2.1.2 Interfaces

2.1.3 Software Interfaces

2.1.4 Operations

*2.2 Product Functions*

*2.3 User Characteristics*

*2.4 Constraints*

2.4.1 Design Constraints

2.4.2 Capability Constraints

*2.5 Assumptions and Dependencies*

**3. Specific Requirements**

**4. Change Management Process**

**5. Document Approvals**

**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 OpenID Connect (OIDC) diagram*

*6.3 Goals from stakeholder*

# **1. Introduction**

## **1.1 Purpose**

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 which 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. |
| **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 |

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/>
* 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 and whatnot 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.
* Javacord https://javacord.org

## **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, 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 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 microservice interfaces 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 interface with the database and its entities to give the user more options to search.

### **2.1.**4 **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.

## **2.2 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) JWT will be used for authentication using valid Discord Credentials 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 messages within the same discord server, not to other servers or outside of the organization users.

PF-9) Admins will have access to Sort/categorize posts

PF-10) Admins will have access to the database to manually delete posts

PF-11) Admins will have access to edit posts on the database

PF-12) Database shall store posts from the discord guild

PF-13) Database shall supply data to the web application

PF-14) Database shall allow changes in the database by Admin`s

## **2.3 User Characteristics**

**Primary users**

* Users of Discord
* Developerswho 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**

**2.4.1 Design constraints**

* JWT must be used for Web application to Authenticate users
* All three microservice systems must run on separate OpenLiberty servers
* All backend systems must be written in JAVA
* The web application must display only the posts that were tagged as interesting by the authenticated user.
* Admins and required microservice must have access to interact with the database

**2.4.2 Capability constraints**

* If an admin deletes the post from the web application that means it will be deleted from the database but the bot will not remove the reaction from the post.

TBD - will add more after the testing is done for the system

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

## **2.5 Assumptions and Dependencies**

* We are assuming that the Discord API Javacord is dependable
* We are assuming that Discord will keep running open source and support developers

# **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: **6.1.1b**

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 will send messages to the database. REF : **6.1.2a**

FRQ - 2.2.1) The Discord Bot will send a message for posts with specific reactions when heard ref; FRQ - 2.1.1.

FRQ - 2.2.2) The Discord Bot will send a deletion message to the DataBase when heard ref; FRQ - 2.1.2.

FRQ - 2.2.3) The Discord Bot will send an edit message to the DataBase when heard ref; FRQ -2.1.3.

FRQ -2.2.4) The Discord Bot will send a removal message to the DataBase when heard ref;

FRQ - 2.1.4.

FRQ - 2.3) Events listened to by the discord bot are associated with entities, such as date, time, channel, user ID, 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 will use HTTPS with a modified post request.

FRQ - 2.7) The bot shall be able to list, add, and remove the specific emoji-content pairs via slash(/) commands in the discord guild.

FRQ - 2.7.1) /dictionary lists all of the specified emojis and their associated meanings.

FRQ - 2.7.2) /meaning *reaction* explains the meaning of the entered emoji.

FRQ - 2.7.3) /add pair *reaction meaning* sets the meaning of a new reaction.

FRQ - 2.7.4) /remove pair *reaction meaning* removes the specified emoji - meaning pair from the dictionary.

FRQ – 3) The database will store Discord posts with specific reactions. REF: **6.1.1a**

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, messages, 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.

FRQ - 4.1.1.1) Must use JSON web tokens (JWT) for security.

FRQ – 4.1.1.2) Using OpenID connect establishes a login session through Discord and provides the user with JWT.

FRQ - 4.1.1.3) Credentials differentiate between user and admin roles.

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 ID 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 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.

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

FRQ - 4.7) The Web App will send posts to the Database Docker.

FRQ - 4.8) The Web App will receive posts from the Database Docker.

FRQ - 4.9) The Web App will use HTTPS with appropriate modifications for our implementation.

FRQ - 4.10) The Web App shall be available to users 24/7.

FRQ - 4.11) 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) Normal users shall interface with Postmarker solely via discord and with the web application. REF : **6.1.3c**

FRQ - 6.3) Admin users retain all the abilities of a normal user, with the addition of some administrative permissions.

FRQ - 6.3.1) Admin users shall 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. REF : **6.1.3c**

FRQ - 6.3.2) Admin users shall 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. REF : **6.1.3c**

# **Change Management Process**

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.

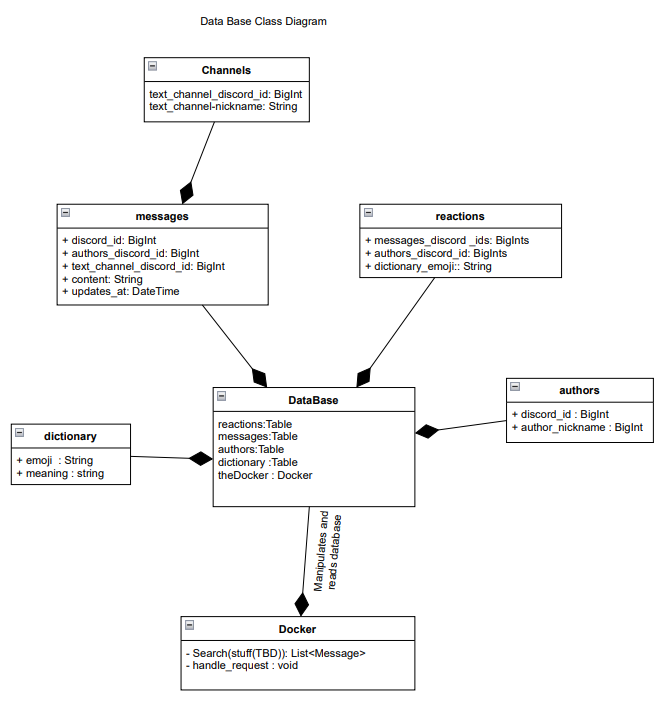
# **Document Approvals**

| **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 |

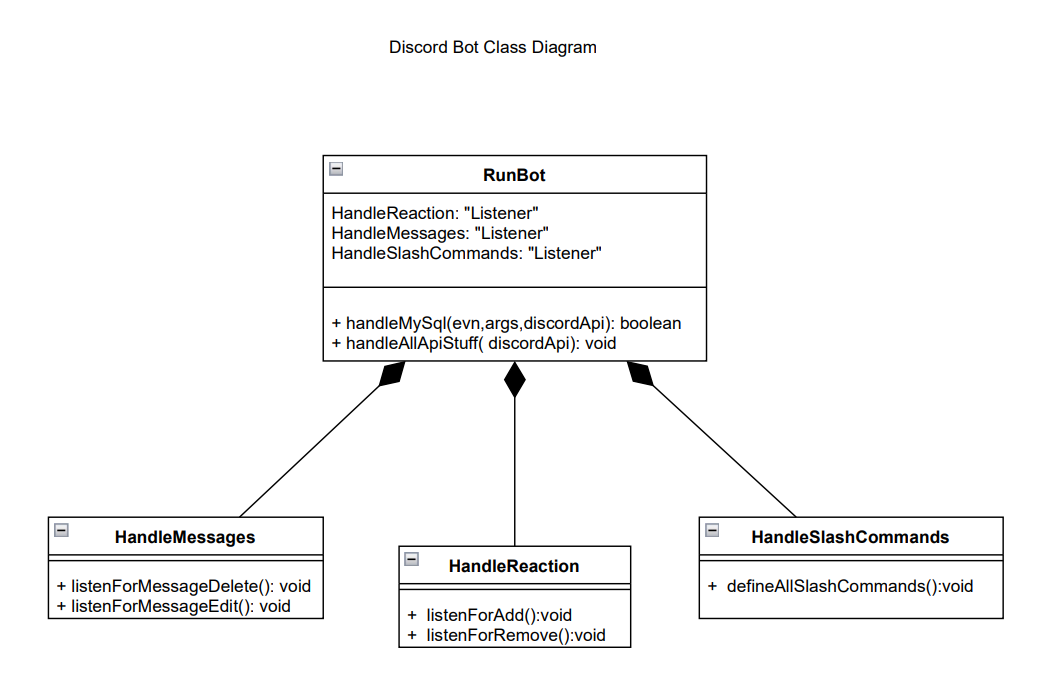
# 6 **Supporting Information**

**6.1 UML Diagrams**

**6.1.1a Database Class diagram**

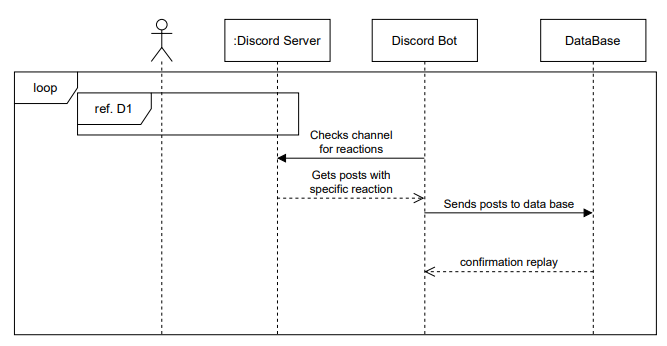
****

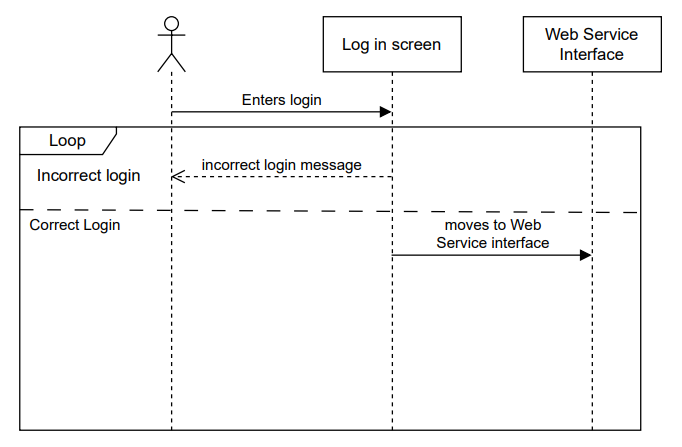
**6.1.1b Discord Bot Class diagram**

****

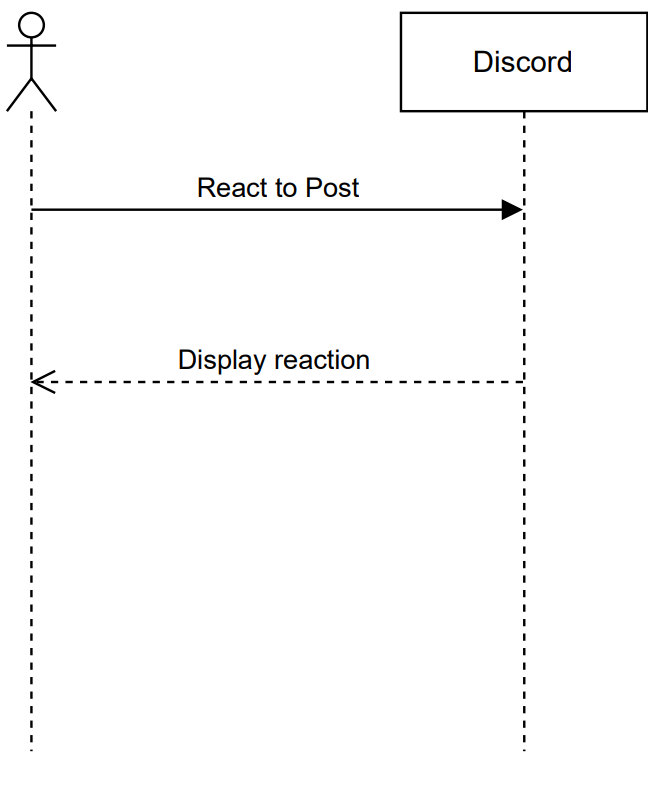
**6.1.2 Sequence Diagram**

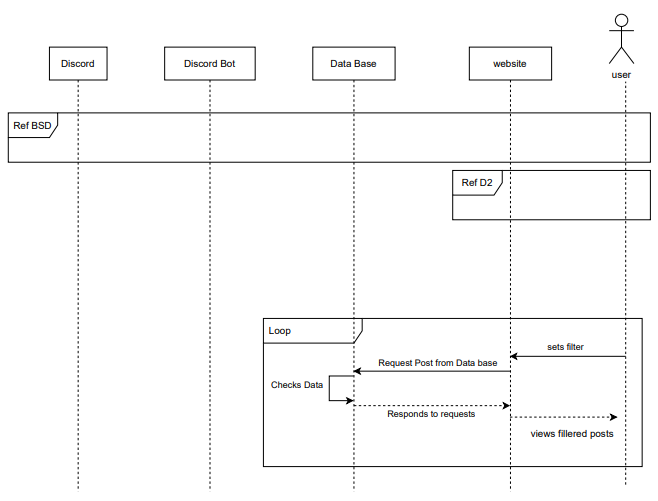
**6.1.2a BSD**

** 6.1.2b D2**

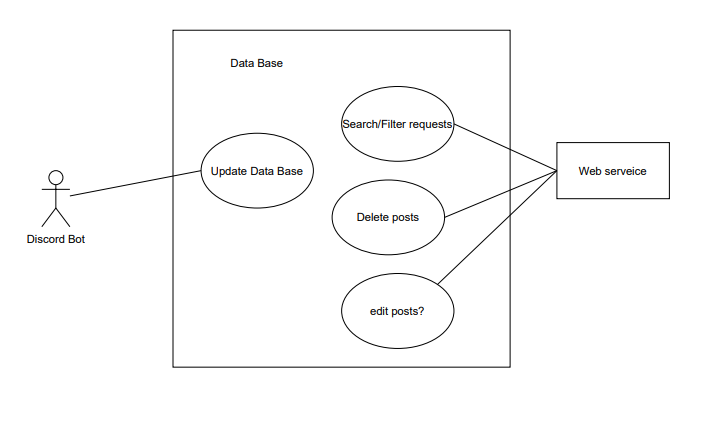
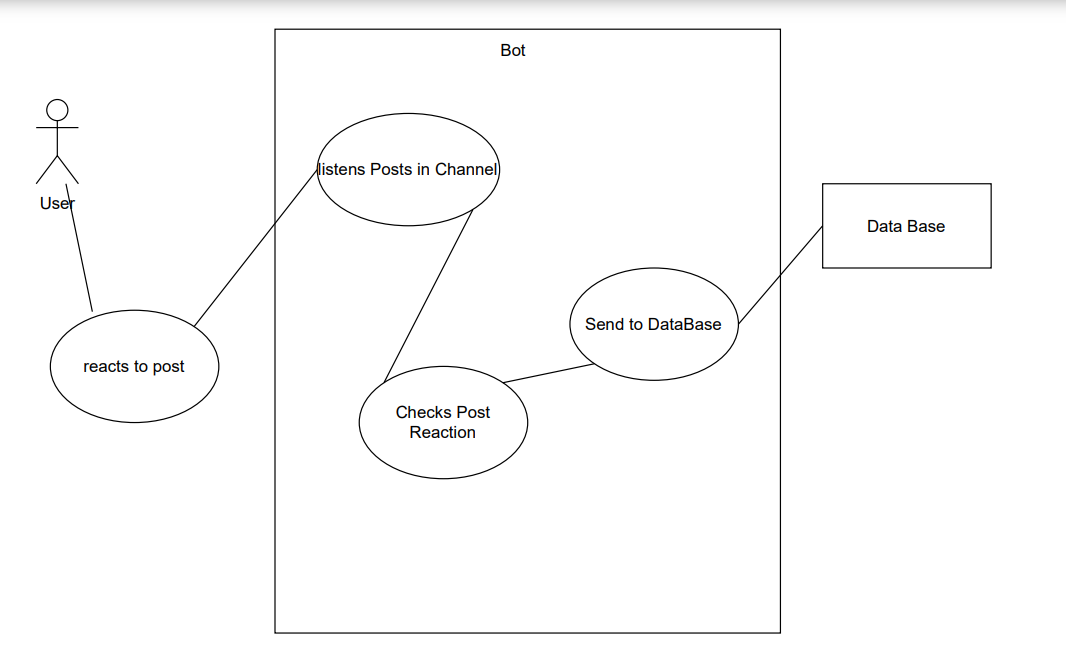
****

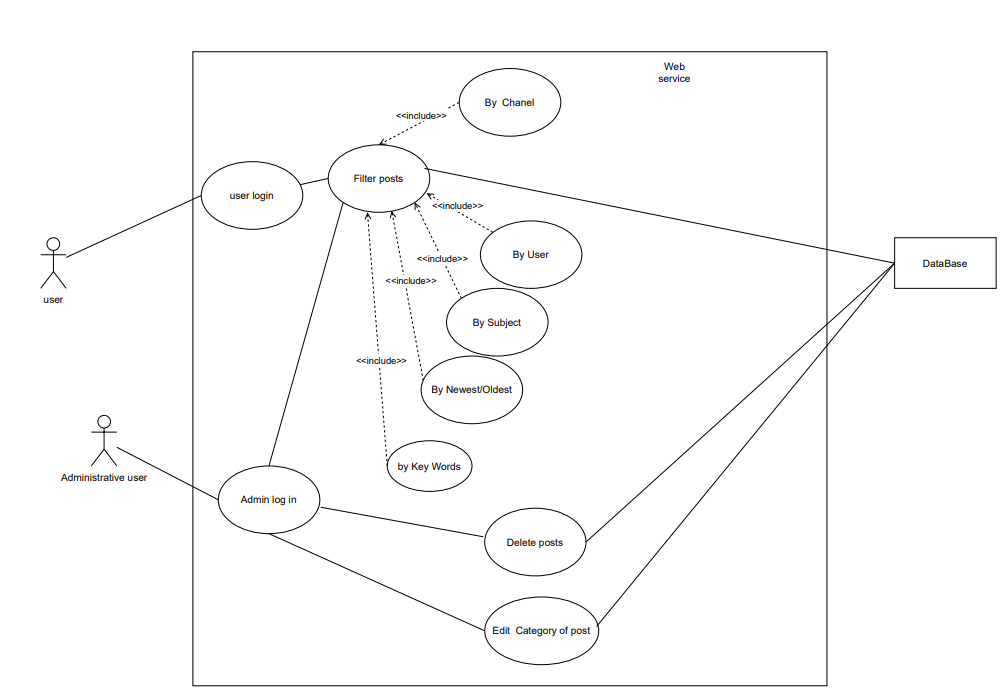
**6.1.2c D1**

****

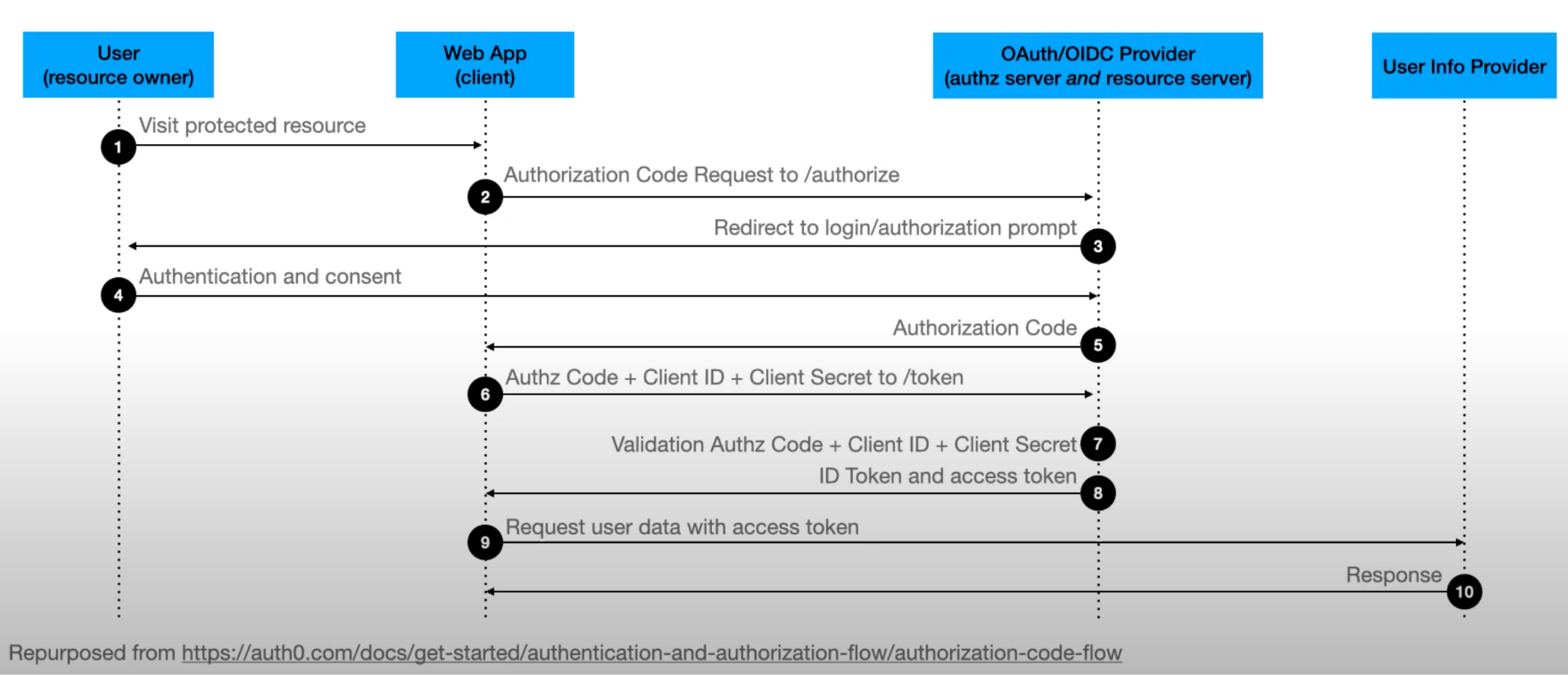
**6.1.2d Main**

**6.1.3 Use case Diagram**

**6.1.3a Data base  6.1.3b Bot**

**6.1.3c Web application**

**6.1.4 OpenID Connect (OIDC) diagram**

****

**6.3 Goals from stakeholders**

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.