IT 314 - Lab 06

Name and IDs:

202001062 – Boricha Vinal
202001089 – Priyank Pitliya
202001094 – Deep Rakhasiya
202001099 – Raj Aditya
202001100 – Shobhit Verma
202001103 – Dhruv Prajapati
202001108 – Nikhil Jethanandani
202001110 – Vihar Shah (Group Representative)
202001116 – Gaurav Shah

Lab Date:

22/03/2023

1. Domain Analysis Models

In Domain Analysis, we try to identify as much as possible on the project domain and figure out the structure of various functionalities of existing systems in the same domain.

1.1. Introduction

The domain for the system can be taken as a "Recipe Sharing Platform". The motivation behind developing this system is to create a platform for aspiring chefs to share their recipe ideas with everyone.

1.2. General Information about the domain

While searching for recipes, users usually want to see a step-by-step transcript of the recipe, a proper list of ingredients required with measurements, a video tutorial, some images to illustrate the end product so that the user can compare to know if they are doing it correctly.

Such a system will have users from all over the world, leading to a high possibility of large amounts of requests hitting the website simultaneously. So the website will need efficient scalability to handle such high load levels.

1.3. Users

The users involved in system interaction are the end-user interacting with the front-end and database, responding to the backend calls.

The end-user needs no training to access the system the first time.

The backend system will perform the necessary database calls when API calls to the backend are made.

1.4. Existing Software

Due to the universality of the domain name, there are many existing systems in the market. Each system has various functionalities like searching, nutritional details, bookmarking recipes, rating and feedback, categories of food, and many more.

The system to be developed must compete with all the existing systems in the market regarding load balancing, availability, usability and more.

2. Boundary, Entity, and Control Object

2.1. Boundary, Entity, and Country Object

1) A **boundary object** is an object that acts as a link or middleman between various communities or groups of people with various viewpoints, values, or behaviours. It is a shared resource that makes cross-border communication and cooperation possible. A boundary object in the context of a project could be any real or intangible asset that helps project stakeholders with various backgrounds and interests work together and understand one another, including a document, tool, model, or other boundary items.

    - Web Browser: The user interacts with the system through a web browser.
    - Mobile Device: The user can also access the system through a mobile device.
    - Recipe Database: The system might access a recipe database to provide recipe information to the user.

2) Conversely, an **entity object** is an abstraction of a real-world item or notion with properties or attributes. An entity object represents a data entity with a distinct identifier and a collection of attributes that specify its properties or behaviours in software engineering or database architecture. An entity object in the context of a project could be any tangible or ethereal object pertinent to the project's objectives and scope.

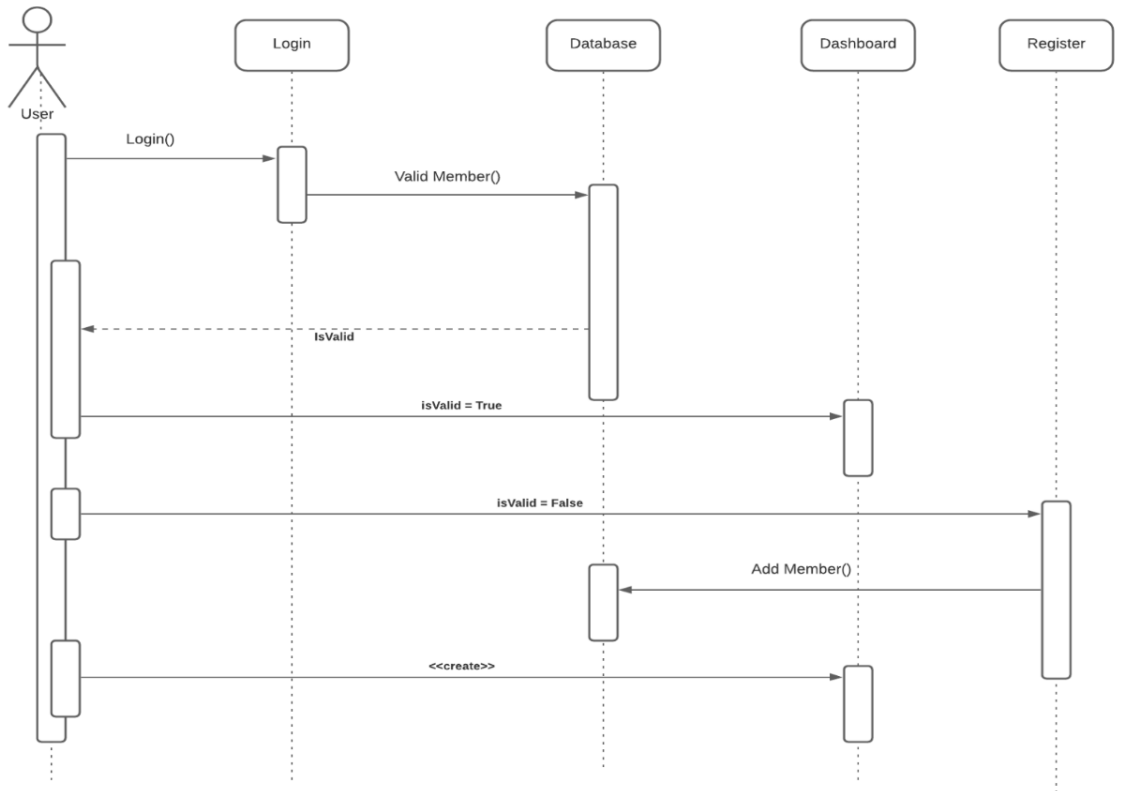
    For "The Naive Baker," some examples of entity objects might include:
    - User: Represents the user of the system.
    - Recipe database: Represents a recipe stored in the system.

3) **Control objects** are the parts of a software system that guarantee the system works properly and according to expectations. Control objects manage and regulate other objects' behaviour in the system. They are often included in the system architecture and are created to preserve the system's integrity, stability, and performance.

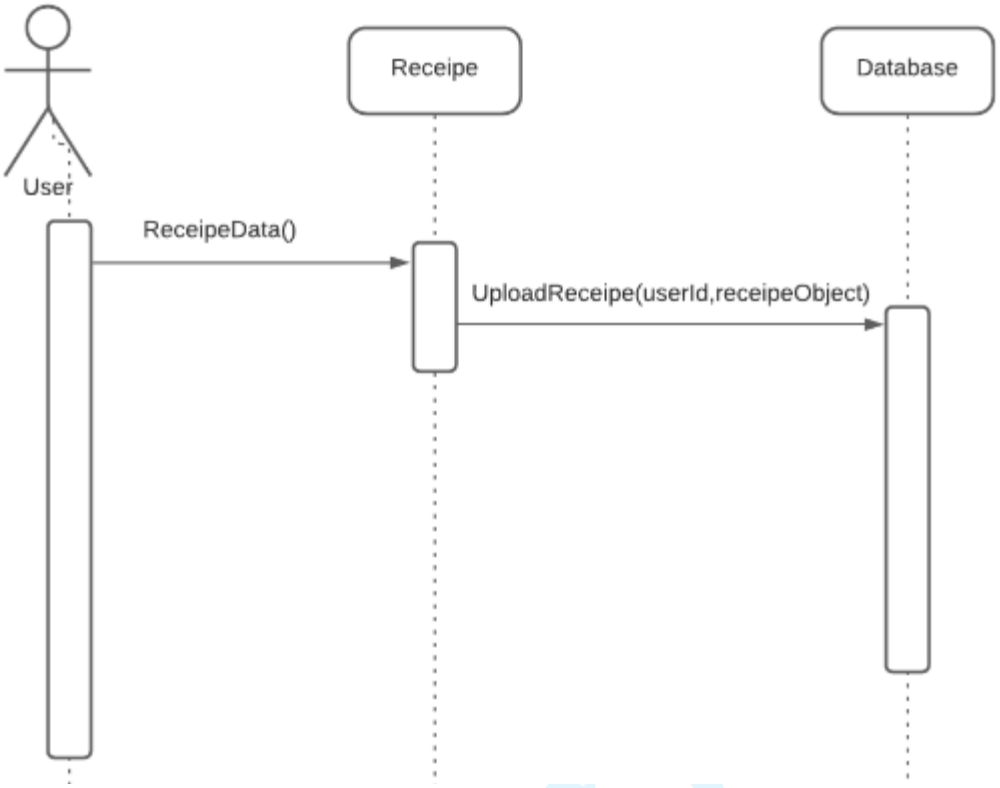
- Login: A control object allowing users to log in to the system and access their accounts.
- Search Recipes: A control object that allows the user to search for recipes in the system.
- Upload Recipe: A control object that allows the user to upload a recipe to the system.
- Remove Recipe: A control object that allows the user to remove a recipe from the system.
- View dashboard: A control object that allows the user to view their dashboard, which contains information about their uploaded recipes, saved recipes, etc.
- Rate a recipe: A control object that allows the user to rate a recipe in the system.
- Save a recipe: A control object that allows the user to save a recipe to their account.
- Report a recipe: A control object that allows users to report a recipe if they find it inappropriate or violating any rules.

2.2. Sequence Diagram

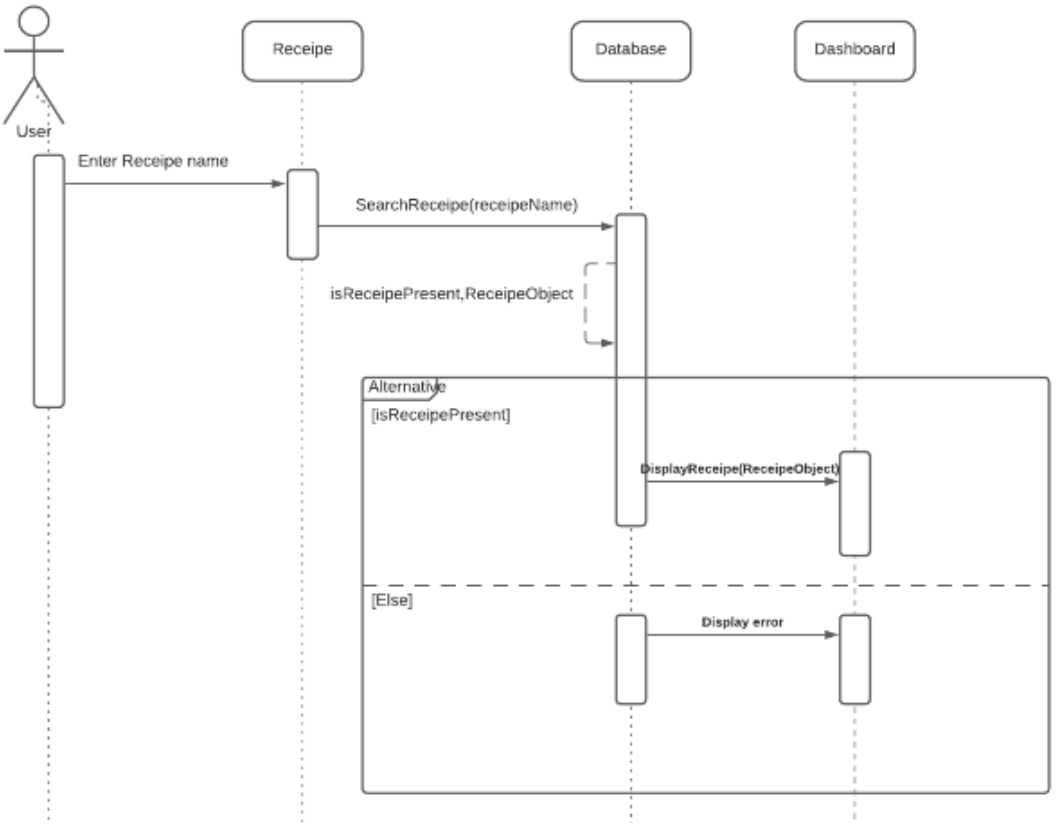
1) Login:



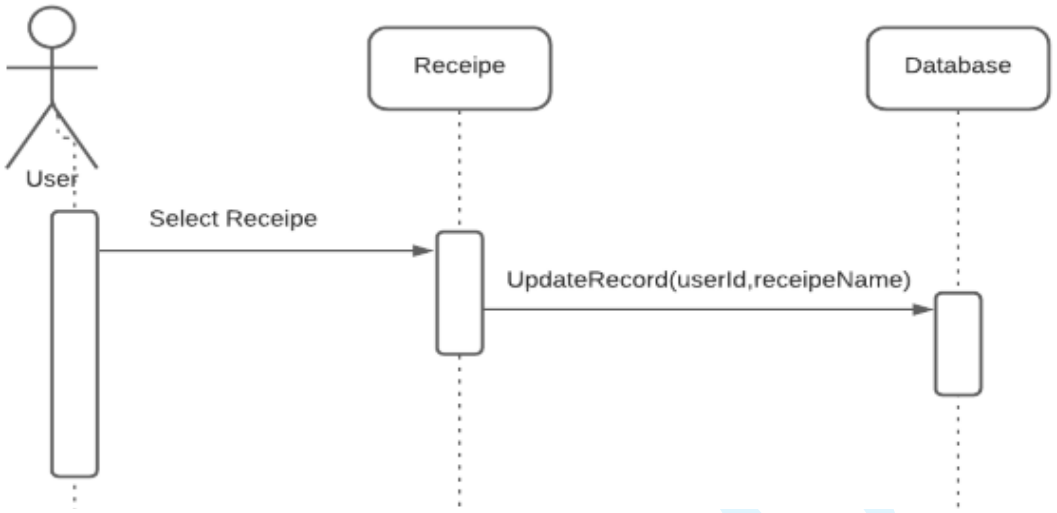
2) Upload Recipe:



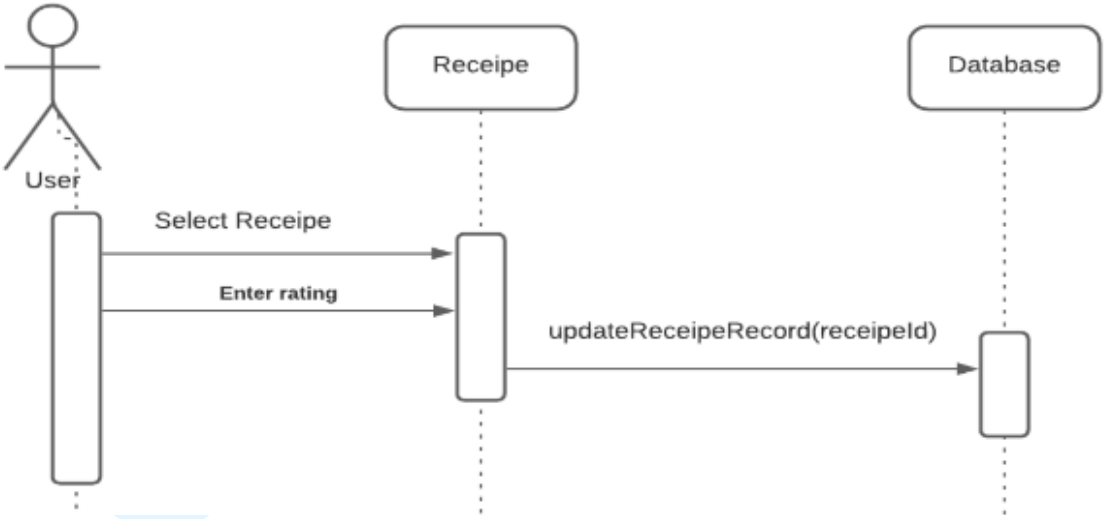
3) Search Recipe



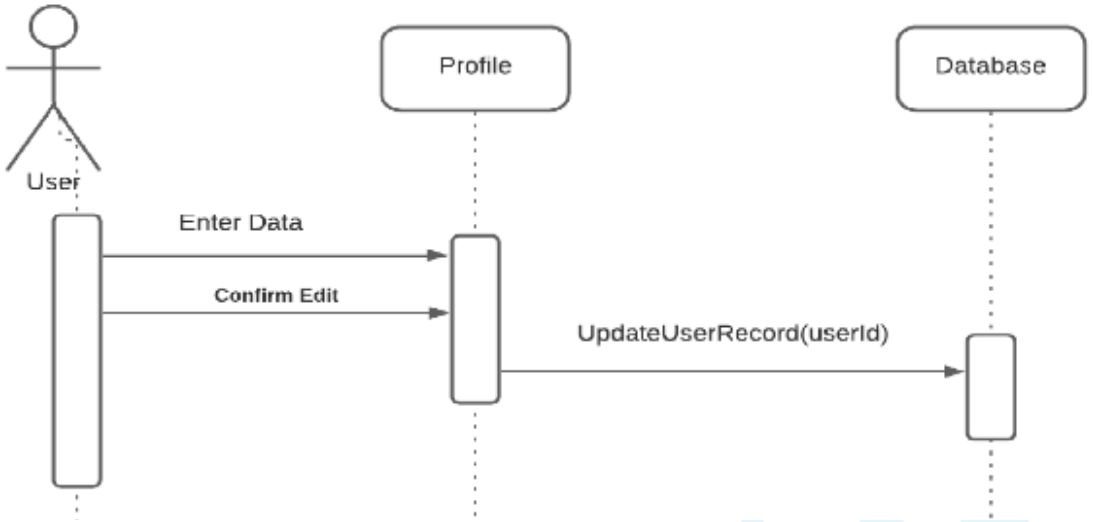
4) Save a recipe



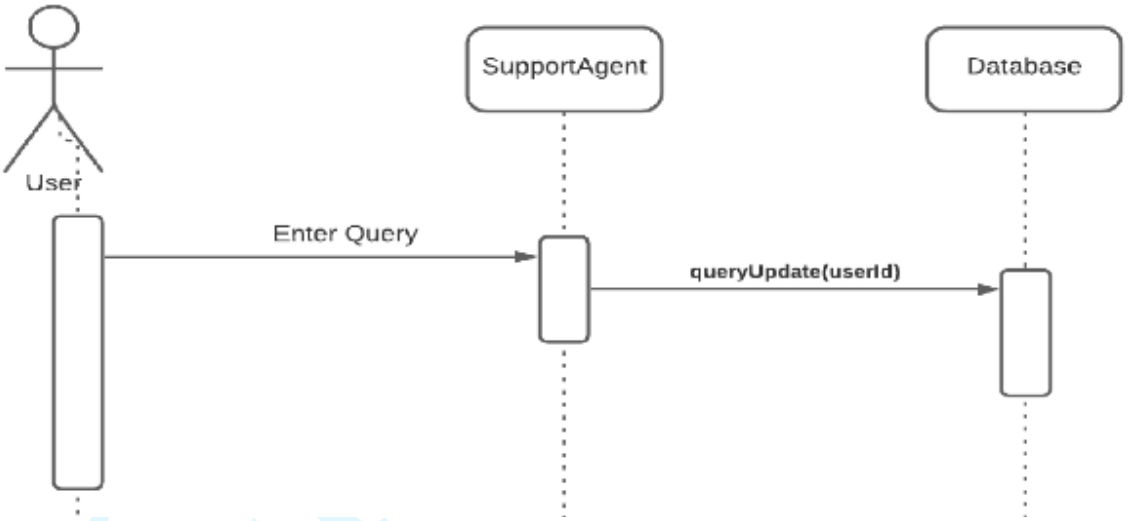
5) Rate a recipe



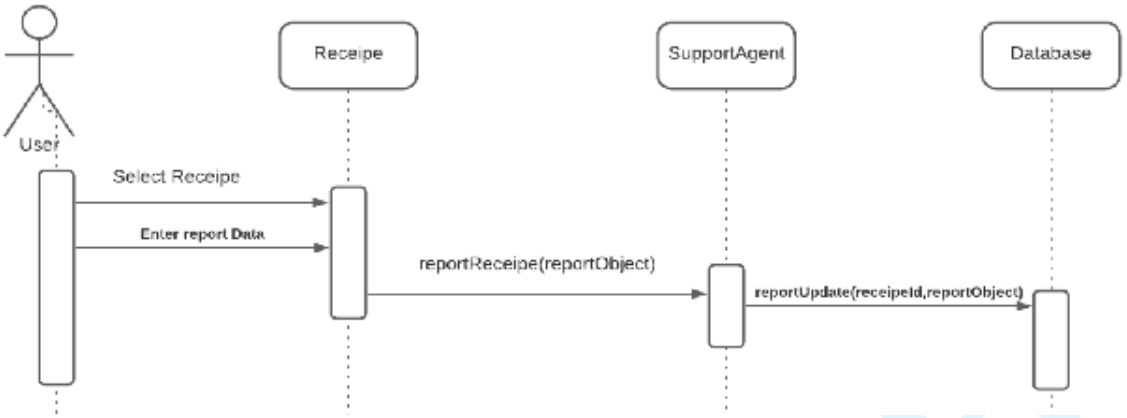
6) Edit profile



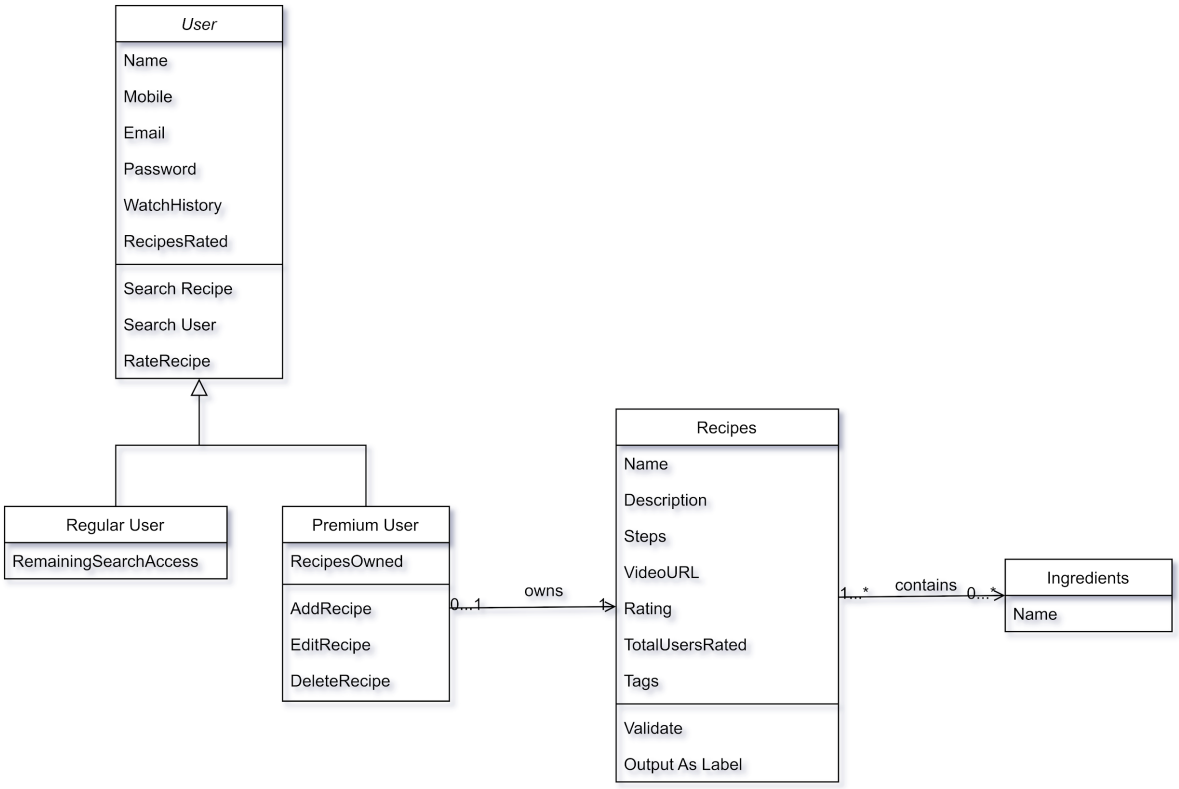
7) Request support



8) Report recipe



2.3. Class Diagram



2.4. Identifying Design Goals

The design goals of our naive baker model system include:

1. **Functionality**: Our system should be able to perform different functions like search recipes and its subcategories and general functionaries of Dashboard. Also, the sign-in/sign-up options work correctly as they are designed to do so.

2. **Security**: Our system should be designed so that when the user enters the password, our system will encrypt the password. The user's request will be authenticated using the authentication tokens in cookies for validation. Also, the user information will be authenticated when the user attempts to log in.

3. **Maintainability**: The system will use the functional react components for maintenance. This will make the system highly reusable and ensure the components' independence.

4. **Testing**: To make the system more reliable and error-free, the system will be tested through static code analysis tools, which will help the admin/developer to identify minor bugs and make the code more readable. Also, the Testing Team will use various test cases to test in every corner of the system.

5. **Ease Of Use:** The system's overall user interface will be very handy for the user to use the system's functionalities. This will also reflect functionalities that involve editing and modifying the user's data.
6. **Minimum number of errors:** The system aims to make minimum errors in providing the results the user expects. Also, in a scenario where multiple requests are coming to the system, the system should be able to fetch the correct results for all the requests.
7. **Reliability:** The system will be designed in such a way that it will be able to handle multiple requests at a time and provide expected results.

3. High-Level System Design

1) Architecture:

This application will use a client-server architecture. We will use a 3 tier application architecture that consists of a presentation tier, an application tier and a data tier.

The data tier stores information, the application tier handles logic, and the presentation tier is a graphical user interface (GUI).

Presentation tier: This layer is distributed to a computing device using a web browser or a web-based application and is constructed with ReactJS. Application programme interface (API) calls are the primary communication between the presentation tier and the other levels.

Application tier: The application tier, also called the logic tier, is written in Node.js and contains the business logic that supports the application's core functions.
This choice is because it allows the app to be used by many users and provides flexibility for future development.

Data tier: A database and software for controlling read and write access to a database make up the data tier. We will be using MongoDb for this tier.

2) IDENTIFYING SUB-SYSTEM: