CRM Design Document

Date	Revision	Description	Author
6/22/2021	1.0	Initial Version	GROUP 3
6/29/2021	2.0	Flesh out design of all CRM systems	GROUP 3

1. Introduction

1.1. Overview

The purpose of this document is to review the design and implementation of a Customer Relationship Management (CRM) application.

Our CRM application has a simple goal. To help companies stay connected with their customers, streamline information and improve profitability.

The CRM system consists of a Server and a Client. The Server stores and processes data that connected Clients perform.

Such tasks that a Client could do are but not limited to: customer contacts, editing those customer connections by adding records and forming groups of customers based on some criteria.

1.2. Goals and Objectives

This document describes the overall CRM application stack and will detail the Client-Server architecture in place.

1.3. Statement of Scope

Decisions in this document are made based on the following priorities (most important first): Maintainability, Usability, Portability, Efficiency.

1.4. Constraints

- The system does not use any off-the-shelf software for data storage
- Administrators are responsible for creating subsequent users of the system
- Users cannot manipulate or edit the data of other users

1.5. Assumptions

- It is assumed that the system will have 3 levels of users: Administrator, Manager, User.
- It is assumed that Administrator-level users will create all users.
- It is assumed that data will persist for all connected users such that upon restarting or reaccessing the application, all their data will be present from the last session.
- It is assumed that User-level users can create/read/update/delete data pertaining to their customers.
- It is assumed that User-level users cannot interact with or manipulate data of other users
- of the system.
- It is assumed that each user has their own account and password.
- Groups depend on the user that created the group.
- Admin and manager can make changes to the account's username and password.

2. Design

The following section will go over each part of the CRM application and detail the Client-Server model being used.

The CRM will run as a Server which will be able to accept multiple-concurrent Clients. Each client will be able to interact with the server by passing and receiving messages. The server in turn processes each message which will be formatted as an operation to perform. Example operations can be (but not limited to):

- Add a new user
- Add/Remove/Update a client
- Group clients together

More operations are listed in detail in the CRM SRS Document. Method and class names are not set in stone but serve to detail the name of the action or actor being performed/doing the action.

2.1. System Architecture

The system architecture design describes the entire application as a whole and includes both the client side and the server side.

This section will describe the design of each component that makes up the CRM system, specifically, the client request, the command processor and the client data.

2.1.1. Client Request

A client request is an operation to be performed by the server at the request of the client.

Each operation performed by a client user for example: Adding a customer, adding a customer record or creating a new customer group and adding customers to said group are packaged as a client request that is sent to the server.

2.1.2. Command Processor

The command processor interprets client requests and performs the requested command. This command processor follows the Command pattern.

Commands are interpreted from the **Command Message** that is passed from the Client to the Server. Each message will have a command to execute and arguments for that command. Each **Command Processor** will know how to interpret the command and the arguments for that command.

2.1.2.1. User Commands

User commands relate to creating, editing or removing users from the CRM system.

2.1.2.1.1. AddUser

Insert a new user into the CRM server database.

2.1.2.1.2. RemoveUser

Remove a user from the CRM server database.

2.1.2.1.3. EditUser

Update a user in the CRM database.

2.1.2.2. Client Commands

Client commands relate creating, editing or removing clients from the CRM system.

2.1.2.2.1. AddClient

Insert a new client into the CRM server database.

2.1.2.2.2. RemoveClient

Remove a client from the CRM server database.

2.1.2.2.3. EditClient

Update a client in the CRM database.

2.1.2.3. Record Commands

Record commands relate to the creating, editing or removing records from the CRM system.

2.1.2.3.1. AddRecord

Insert a new record into the CRM server database.

2.1.2.3.2. RemoveRecord

Remove a record from the CRM server database.

2.1.2.3.3. EditRecord

Update a record in the CRM database.

2.1.2.4. Group Commands

Group commands relate to creating, editing or removing groups from the CRM system.

2.1.2.4.1. AddGroup

Insert a new group into the CRM server database.

2.1.2.4.2. RemoveGroup

Remove a group from the CRM server database.

2.1.2.4.3. EditGroup

Update a group in the CRM database.

2.1.2.5. Extras

Extra commands are additional commands that do not relate specifically to the CRM operations. These are support commands.

2.1.2.5.1. Import

Load saved data

2.1.2.5.2. Export

Export data

2.1.2.5.3. Save

Save data

2.1.3. Client Data

Client data is saved to a file and loaded on Server startup. Each inbound client request is validated and processed and will be synced to the datafile. This will ensure that all valid requests are in sync.

The current direction for how the data will be stored is currently as a CSV formatted file which will be read and parsed by the Server.

2.1.4. Models

2.1.4.1. User Models

2.1.4.1.1. Users

A generic base class that represents an entity

2.1.4.1.2. Admin

A top-level **User** with the ability to create new users, edit users, get users and delete users

2.1.4.1.3. SalesUser

A sales **User**. Has clients and the ability to add clients, add client records, edit records, group clients together, edit client groups, remove client groups

2.1.4.1.4. SalesRep

A variant of the **SalesUser**. Has a **User** manager.

2.1.4.1.5. SalesManager

A higher-level manager with the same capabilities as a **SalesRep** but has a team of **SalesRep** under them.

2.1.4.2. Message Model

2.1.4.2.1. Message

An object that represents a payload sent between the client and server. Messages are passed back and forth using **ObjectStreams**.

2.1.4.2.2. Authentication

An authentication **Message** which is used to notify the server that a login is requested.

2.1.4.2.3. Command

A command **Message** which notifies the server that a task is being requested to be performed. **Command Message** have an argument object which is passed to the appropriate command driver.

2.2. Client Side

2.2.1. Login

Method **userLogin** will be called on client application execution and request the user to input a username and password which will be passed to the server. This method will be passed to the server which will authenticate the client.

A dialog box will be displayed that will request the user to enter their CRM server address, CRM server port, username and password.

Following the submission, a new connection will be established with the **Server** if the CRM server address and server port are correct.

An **Authentication Message** will be sent to the **Server** in which the authentication message will be verified. Following authentication, the server will respond to the client.

2.2.2. GUI

Client interaction will be through an event-driven GUI. The GUI itself is written with Swing.

After successfully authenticating with the server, the GUI will be populated with GUI elements to graphically represent the user's data.

The GUI will provide the connected user with the ability to interact with the CRM system via a graphical interface. This interface will translate all events into a **Message** which will be sent to the **Server**.

Through the GUI, the Client has the ability to modify records, based on permissions and privileges by role sets, which is communicated between the server.

> Method **modData** will be called on the client application, which would prompt a template for basic information for the user to submit.
>
> The inputted data must match the record format, which would then display in the GUI a successful record that has been CREATED/MODIFIED/DELETED.

2.3. Server Side

The CRM operates on a client-server model.

2.3.1. Configuration File

Method **readConfigurationFile** will read and process a server configuration file to determine what port to bind, the storage file location and any other configuration details.

2.3.2. Client Handler

Class **ClientHandler** is a class that implements the **Runnable** interface which allows for handling multiple client connections.

Method **run** is the innermost handler for managing the client connection with the server. It sits and waits for incoming messages from the client. These messages are read and validated before being executed. A return message will be sent back to the client when the task is complete.

The client handler will interpret the message and determine the appropriate calls to make. For example, **Command** requests will be routed to the **CommandProcessor** to handle them.

2.3.3. Server

Method **startServer** will start the server process by creating a **ServerSocket** and waiting for clients to connect which will be handed off to **ClientHandler** and all client requests will be processed in a separate thread to support multiple clients.

The server will also start the **StorageWatcher** which will watch the **StorageQueue** and monitor incoming requests to save data to the datafile.

Method **recvClientMessage** retrieves a passed client message. All done via an **InputStream/ObjectInputStream**.

The **ClientHandler** will process the request and validate if it's a doable request. If the request is valid and doable, the request will be done and a save routine will be called to sync changes to the datafile.

Method **syncClient** will be called and sync updated changes that are stored locally on the server back into the data file for persistence. This synchronization calls upon the **StorageManager** to facilitate the process of saving data.

> The synchronization is non-blocking thus after the ClientHandler is able to perform the request, and acknowledge that the request is completed should be sent back to the Client and the UI should be flushed to reflect these changes (such as a new record appearing after we've created it).

2.3.4. Command Processor

The command processor reads incoming **Command Messages** and processes them depending on what specific command needs to be performed.

Interface **Command** contains a single method: **execute** that all commands are run through.

2.4. Storage Manager

The **StorageManager** class provides functions to monitor and save data to a persistent file object.

StorageWatcher runs in the background in its own thread to monitor for any requests to save data.

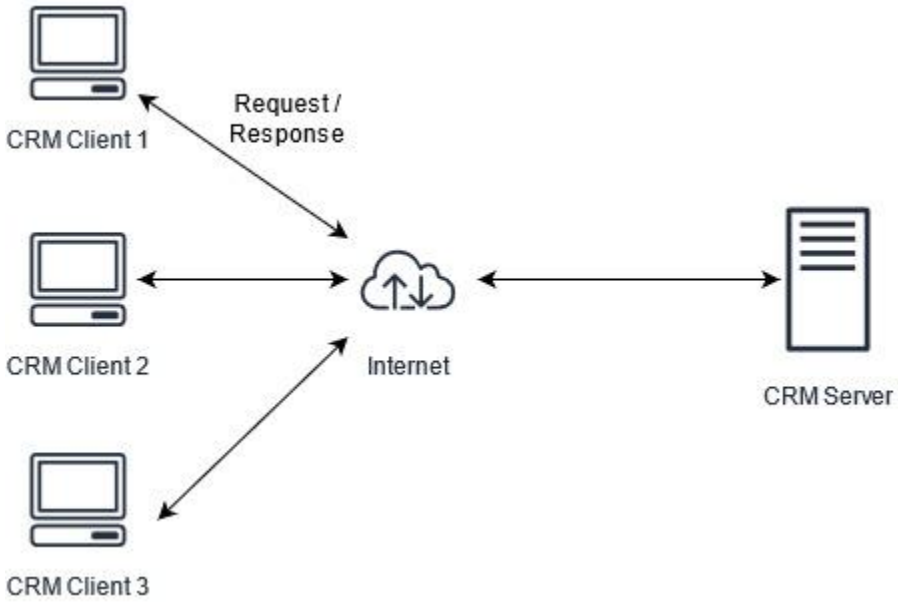
StorageQueue is a Queue which stores requests to save data on a First-In, First-out basis. This ensures that changes are saved in a specific order to preserve data consistency.

The data is saved to the datafile which allows the Server on startup to read and load from.

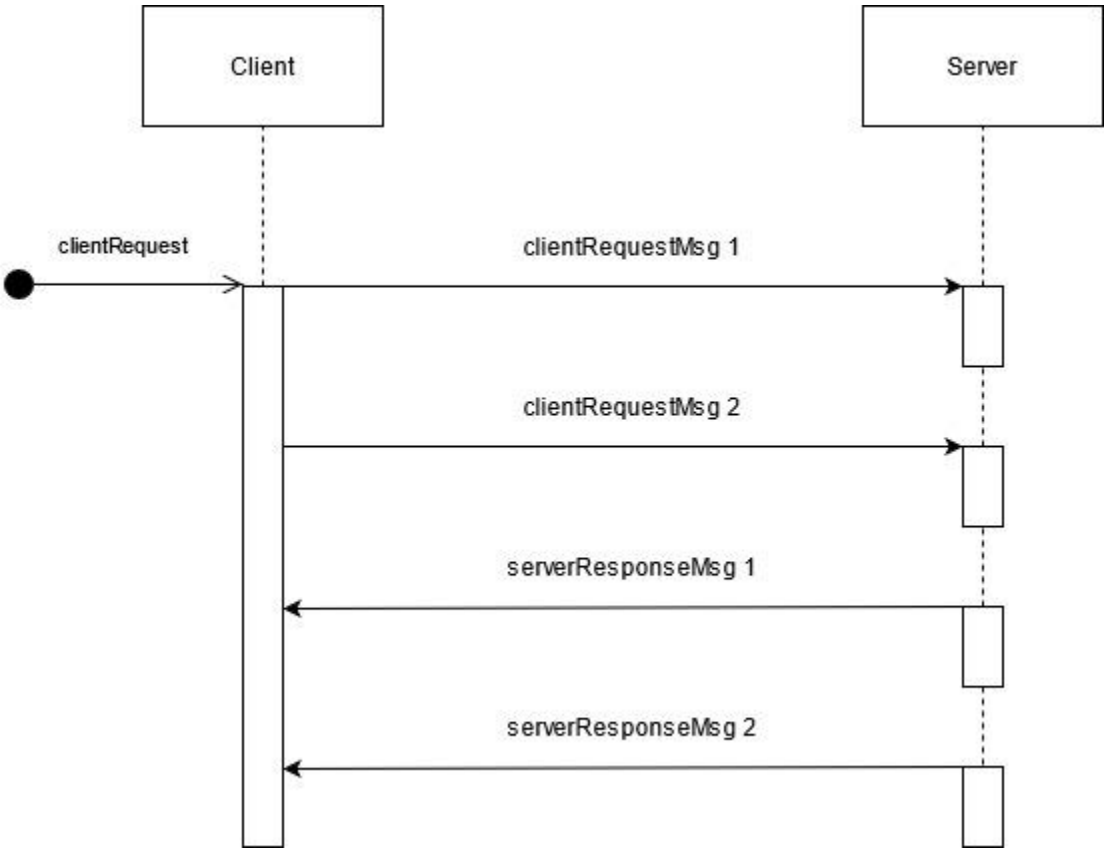
3. Architectural and Component-level Design

3.1. Client Server Architecture

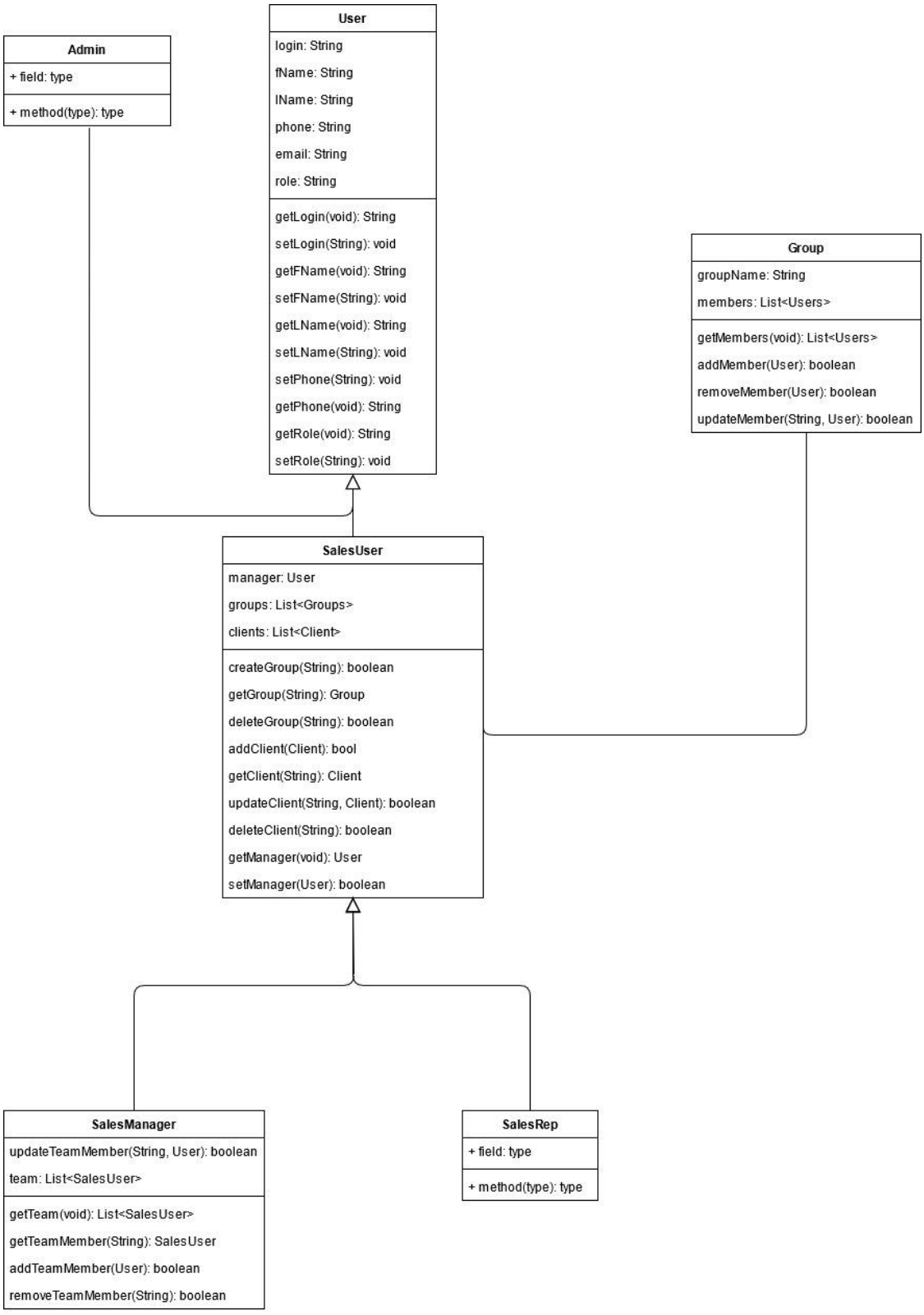
3.1.1. High Level Connection Model



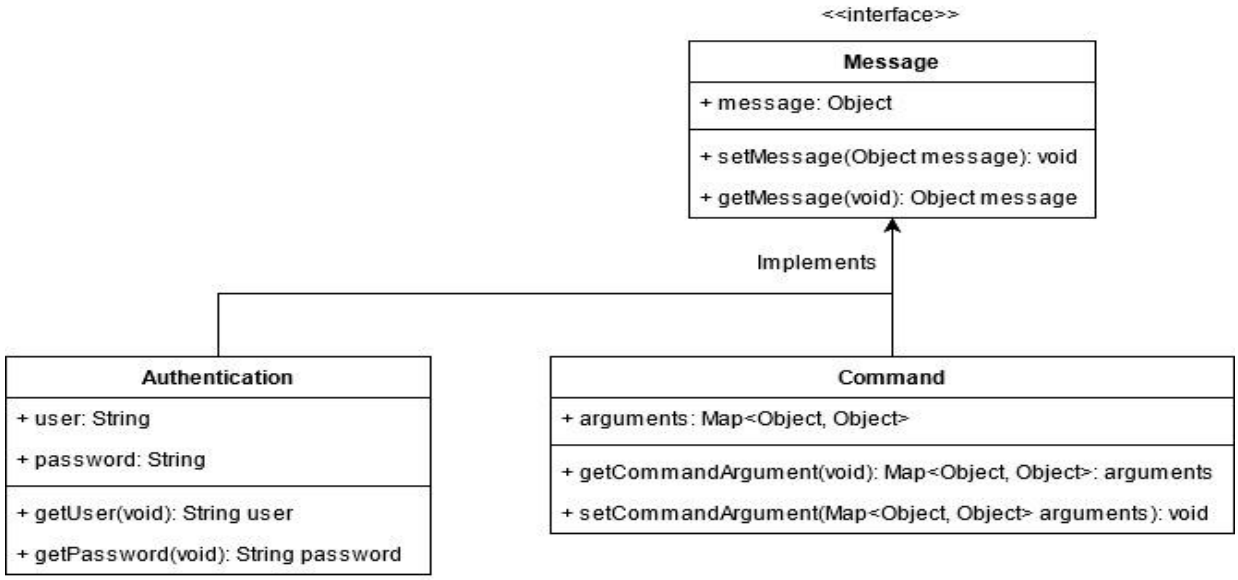
3.1.2. Client Request Model



3.1.3. User Models

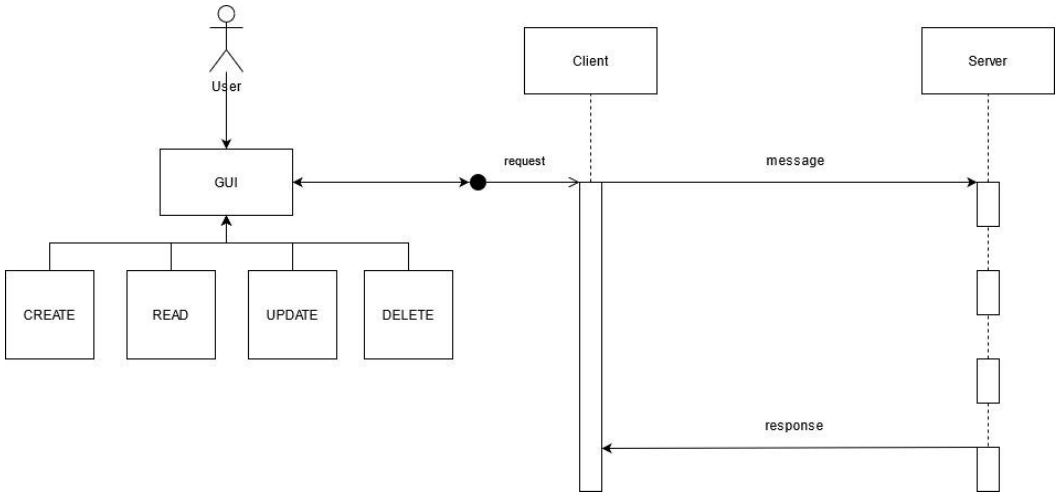


3.1.4. Messaging Model



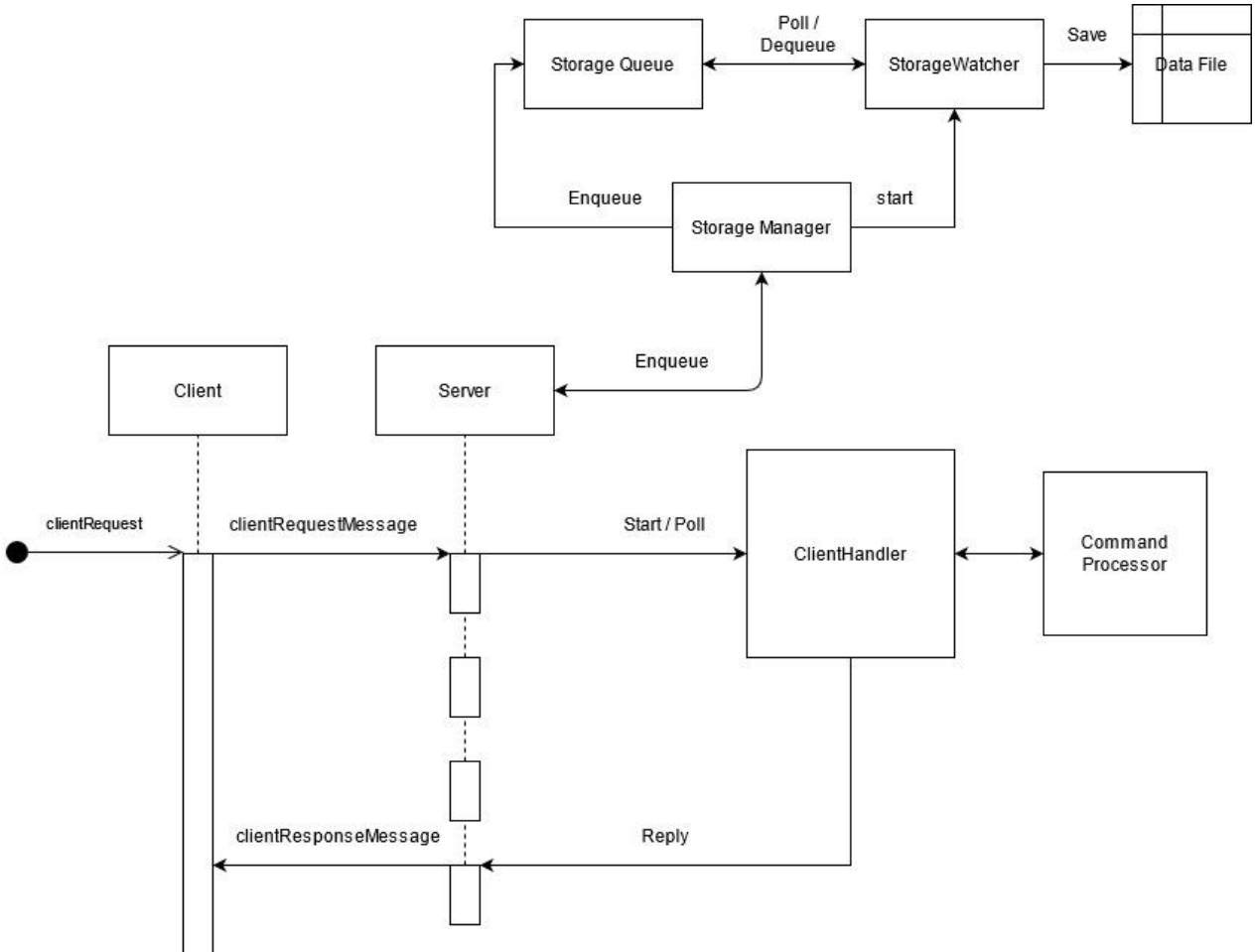
3.2. Client Architecture

3.2.1. GUI

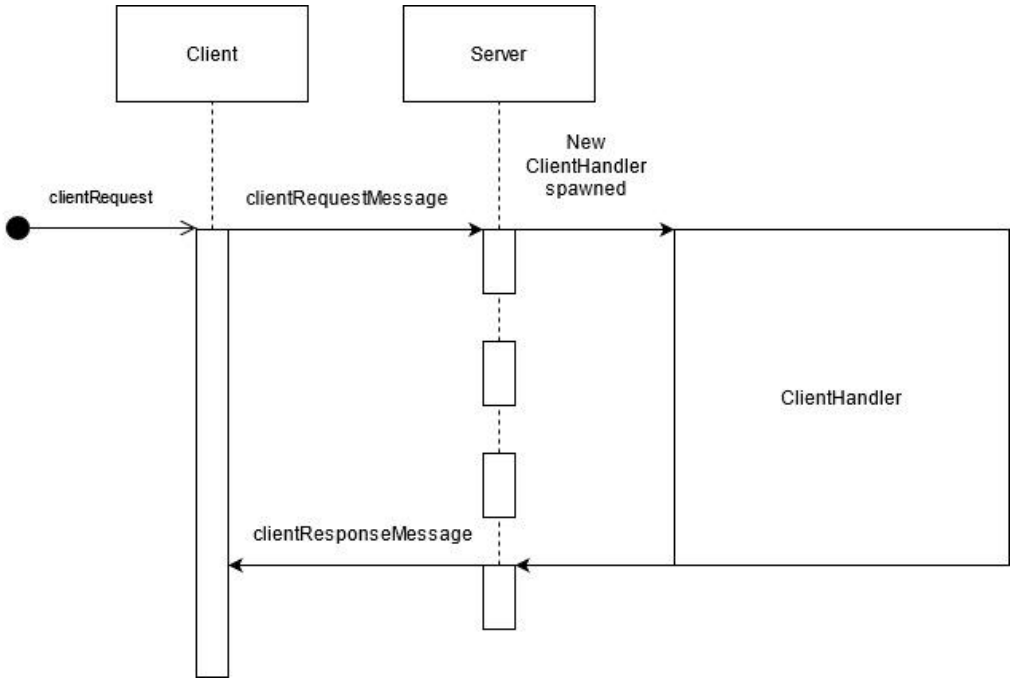


3.3. Server Architecture

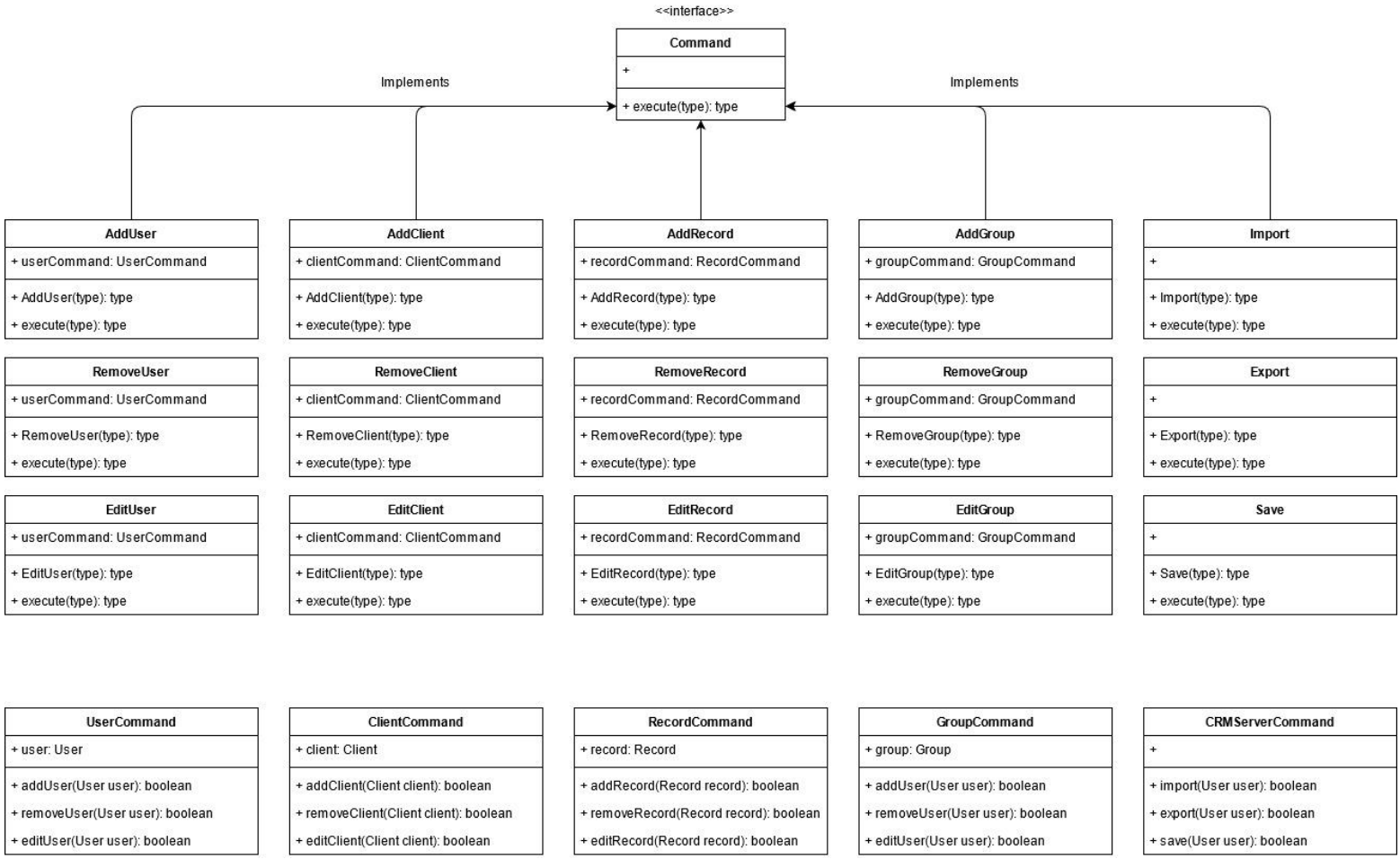
3.3.1. Server Side Architecture



3.3.2. Client Handler

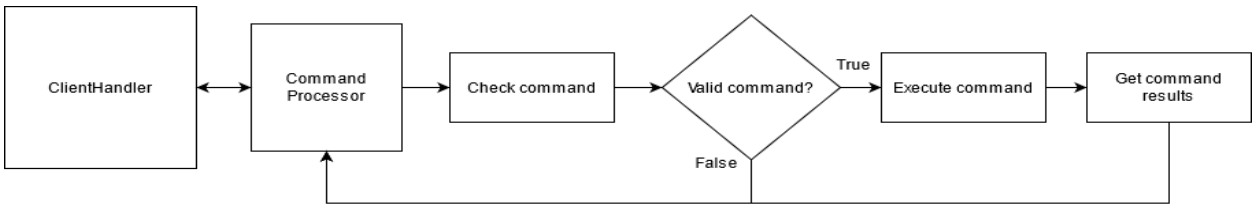


3.3.3. Command Models

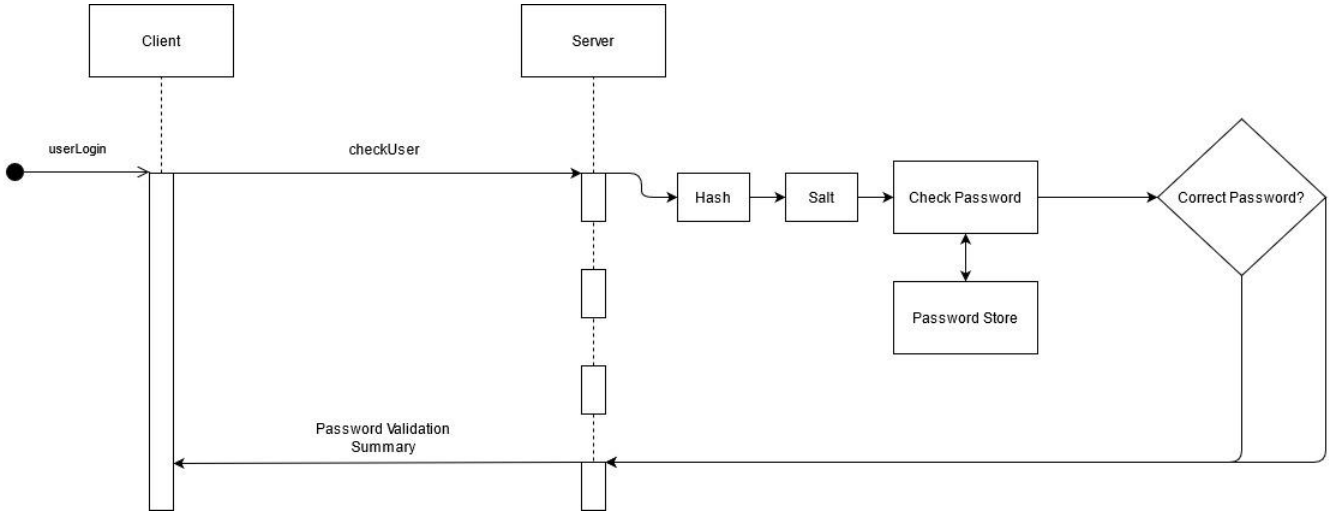


3.4. Sequence Models

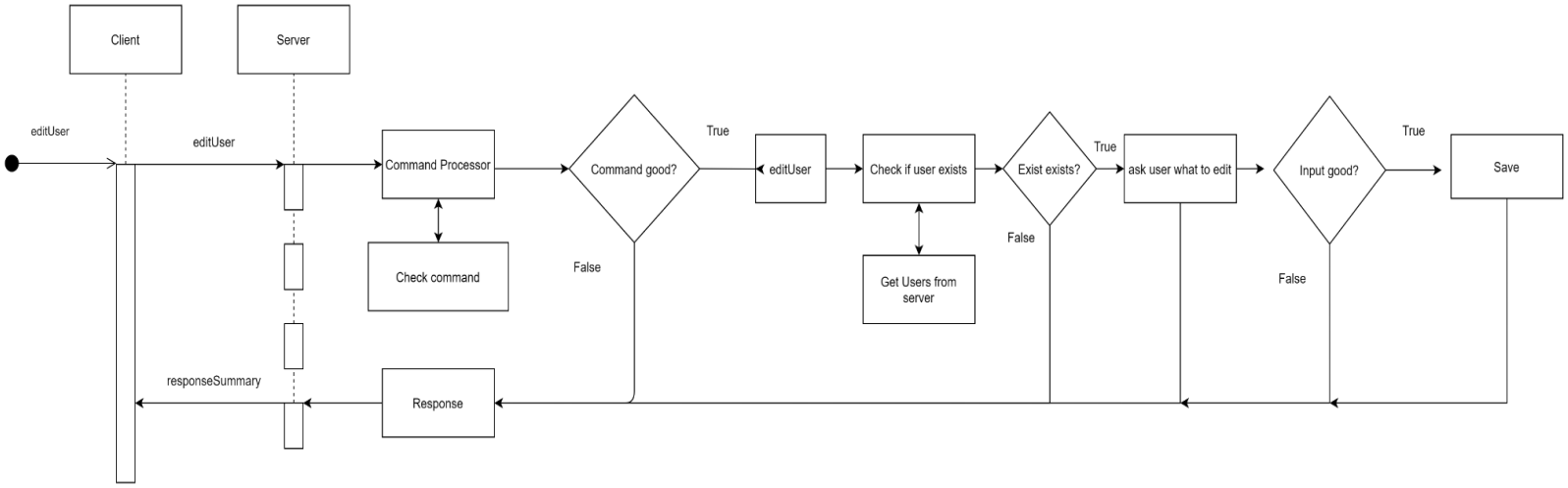
3.4.1. Command Processor



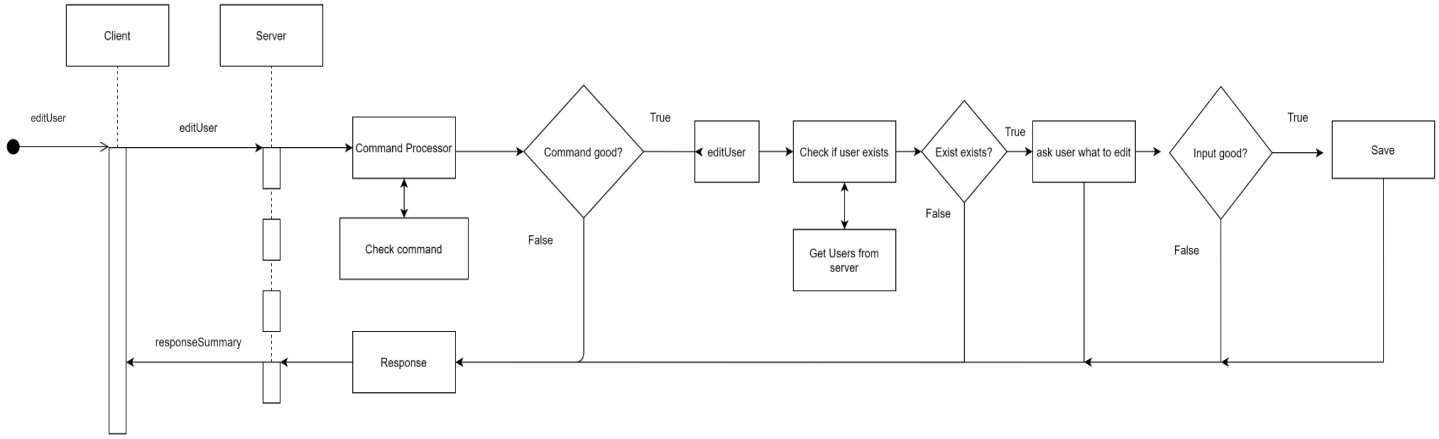
3.4.2. Authentication



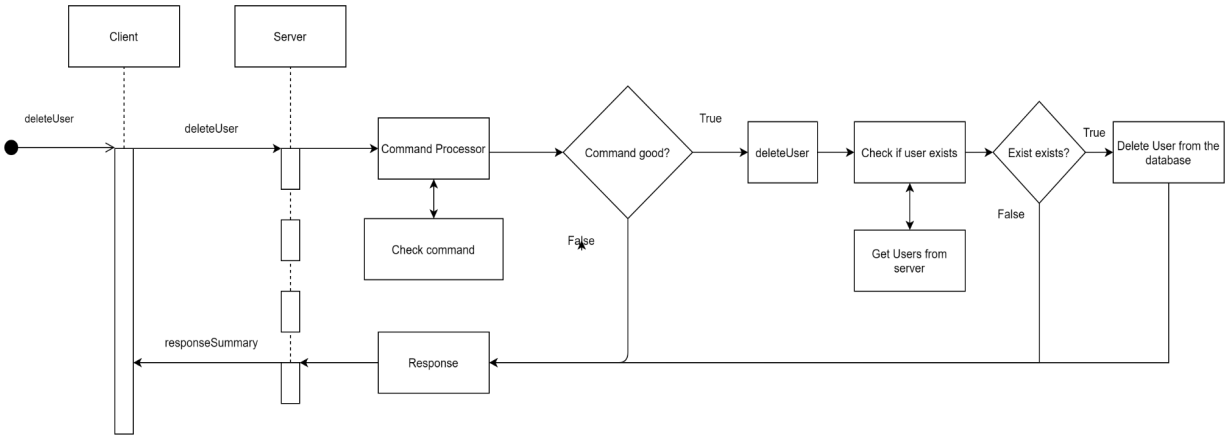
3.4.3. Client Login



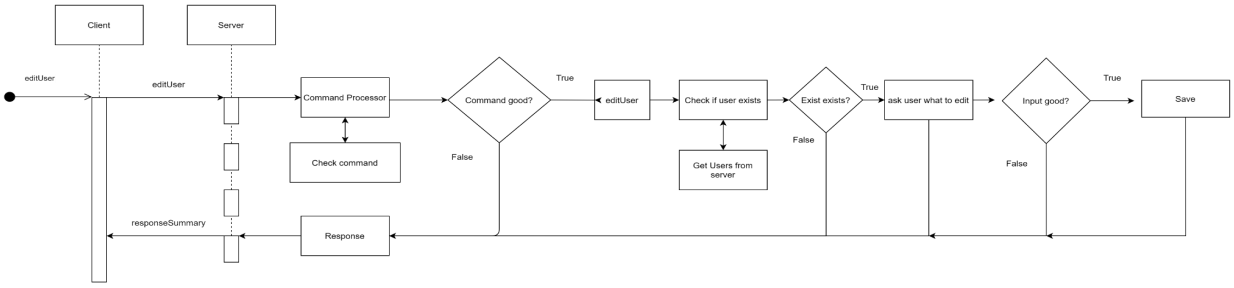
3.4.4. Adding user



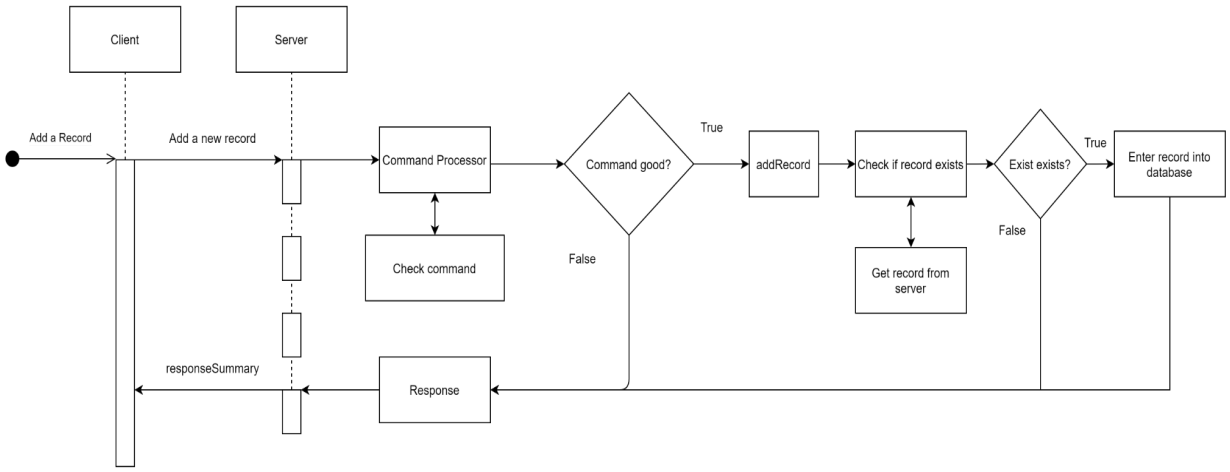
3.4.5. Remove User



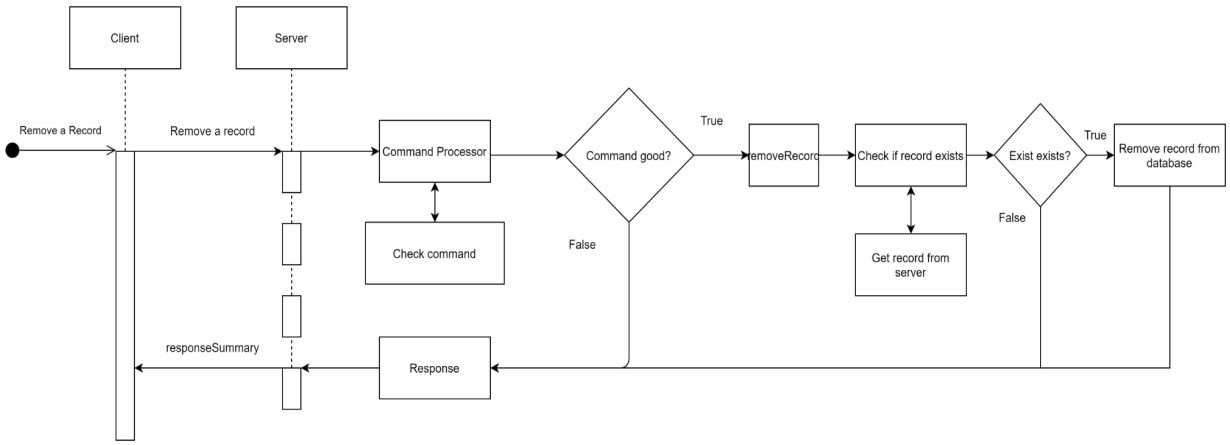
3.4.6. Edit User



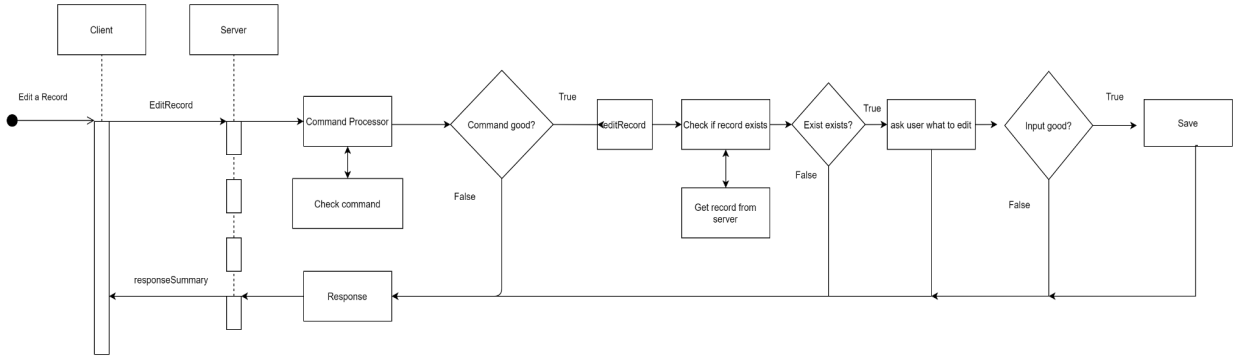
3.4.7. Add Record



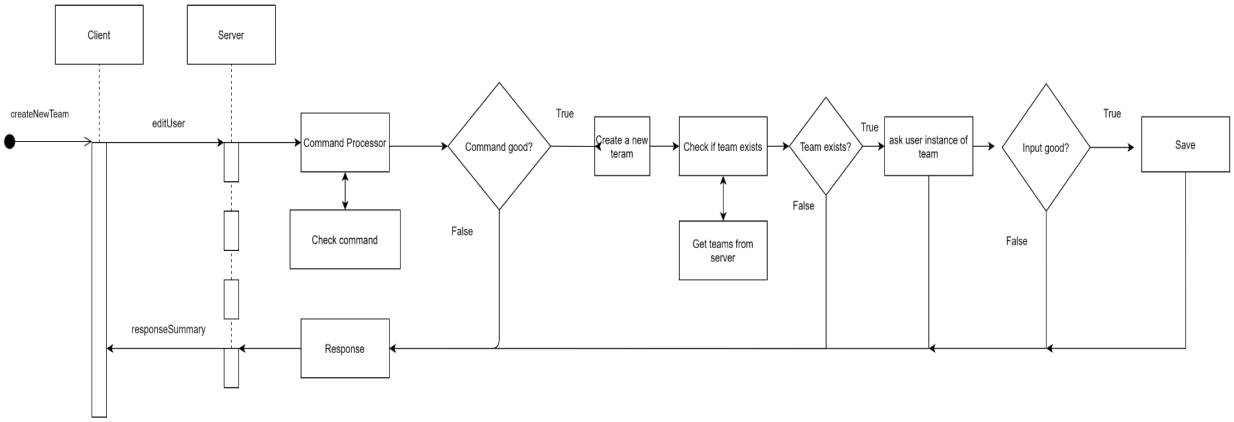
3.4.8. Remove Record



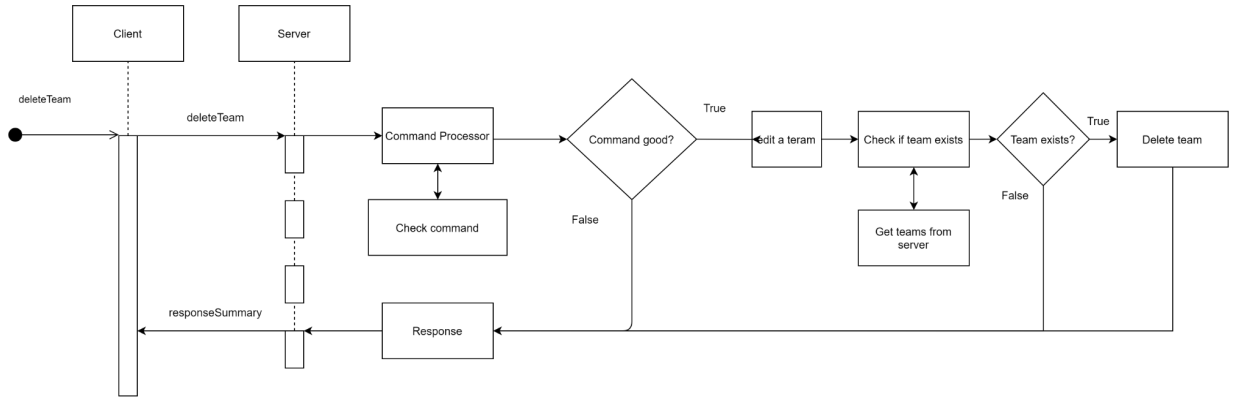
3.4.9. Edit Record



3.4.10. Add Group



3.4.11. Remove Group



3.4.12. Edit Group