CRM Design Document

Date	Revision	Description	Author
6/22/2021	1.0	Initial Version	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.

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 the user that created the group

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 will process 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. Data Design

2.1.1. Client Data

Client data is saved to a file and loaded on Server startup. Each inbound client request when validated and processed will be sync 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.

2.2. Client Side

Client interaction will be through an event-driven GUI. The GUI itself will use JavaFX (Swing is an alternative but is an older toolkit).

Client requests are packages as messages that are passed to the server.

Interactions with the GUI will cause a message to be sent to the Server which will be interpreted as an action to be performed.

Method **passClientMessage** passes a client operation from the client to the server. All done via an **OutputStream/ObjectOutputStream**.

Clients will be able to connect to the server component via a provided username and password.

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.

On successful login, the user's data will be displayed in the GUI which was any previous data that was saved on the server.

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 executed. A return message will be sent back to the client when the task is complete.

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.

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. Synchronization should be done in the form of a queue. All syncs should be performed first in, first out. This will ensure that the latest changes are made and should prevent data corruption.

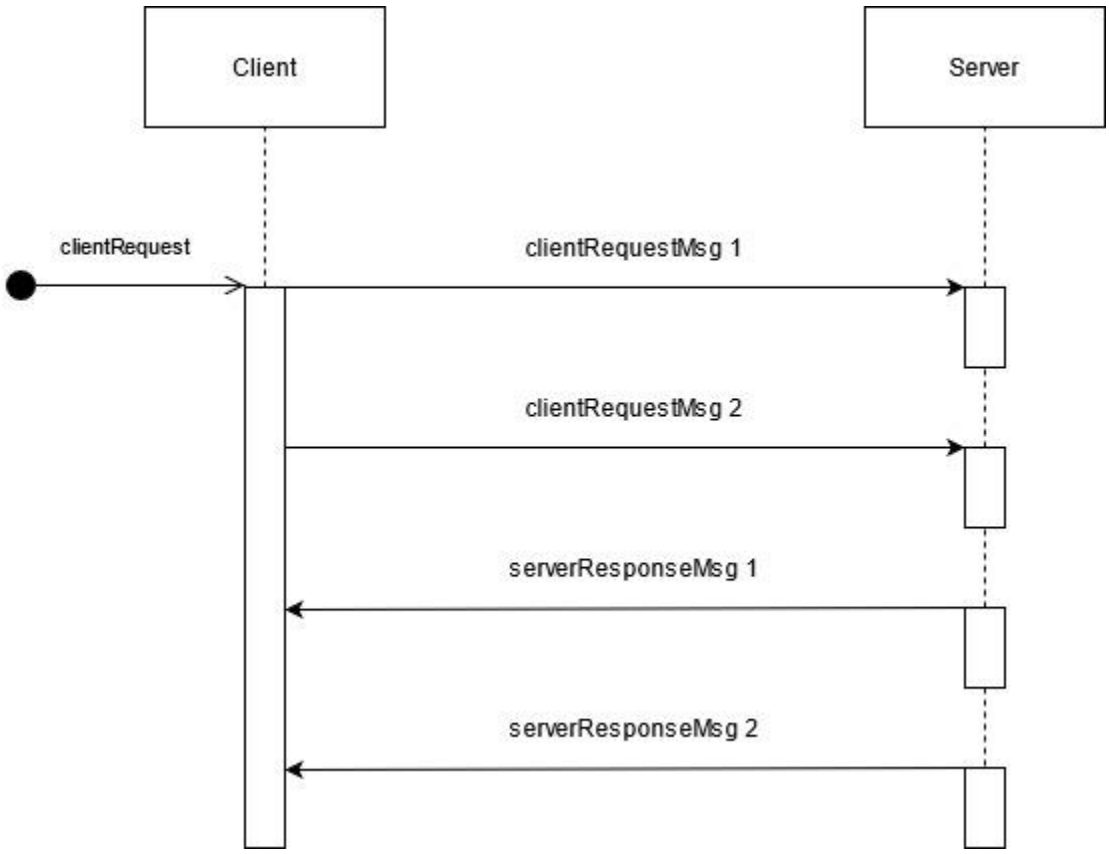
> The synchronization is non-blocking thus after the ClientHandler is able to perform the request, and acknowledge that the request is completed should be sent back to the Client and the UI should be flushed to reflect these changes (such as a new record appearing after we've created it).

3. Architectural and Component-level Design

3.1. Client Architecture

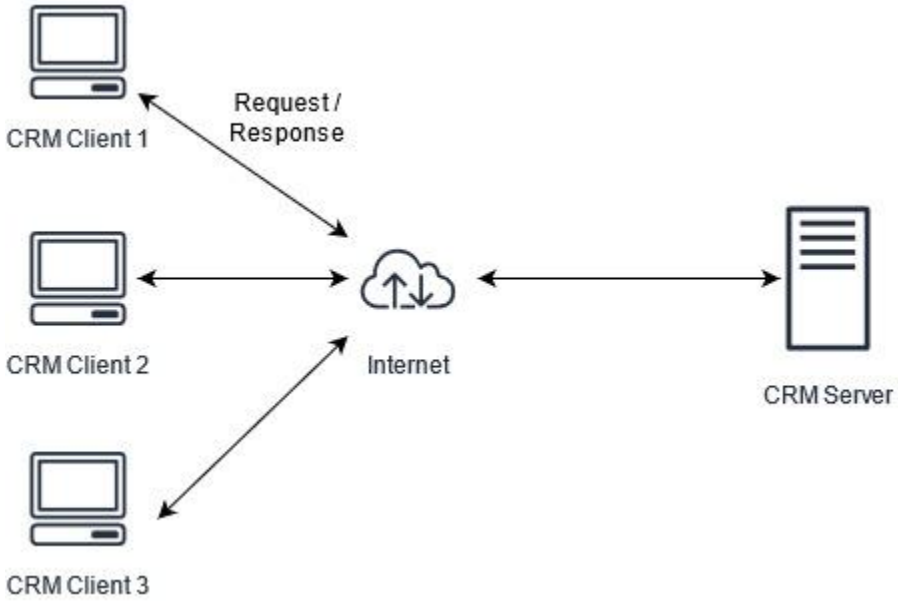
3.1.1. Client Request Model

The following diagram shows the client-server request model:



3.2. Server Architecture

3.2.1. Connection Model



3.2.2. Authentication Model