CRM Application

Developers: Harman Rai, Zeyu Zhang, Jacki Zhong, Nicholas Krone

CRM - Customer Relationship Management

Client / Server based application that allows sales representatives to maintain customer relationships via record tracking

- Username / Password based access to Server for each user
- Add, remove, delete or edit customers and records
- Group customers together in any way with a group name
- Persistent data and ability to import or export customer data

Functional Requirements

1. We need user to enter their username and password to verification.
2. Our system storing users and their clients' information.
3. Our system provide sorting, adding, removing or editing those features.
4. There is a database to store all of the informations above.
5. There is an Admin class to directly make changes to the database. Therefore, Admin can help the user change their password if some user forgot their password.

Non-functional Requirements

1. For security reasons, the salesman cannot directly make changes to the database. Only the manager or higher authorities can achieve that.
2. We assume the system might need to store a large number of data. We are going to use a hash-table to build our database to make the search quicker.
3. We will make a copy of our database, therefore, we can recover from failure quickly.

Classes

The high-level class overview of the CRM application

Users

- Admin, Manager, Sales, Customer, Group

Messenger

- Authentication: notifies the server that a login is requested.
- Command processor: notifies the server that a XYZ task is requested.

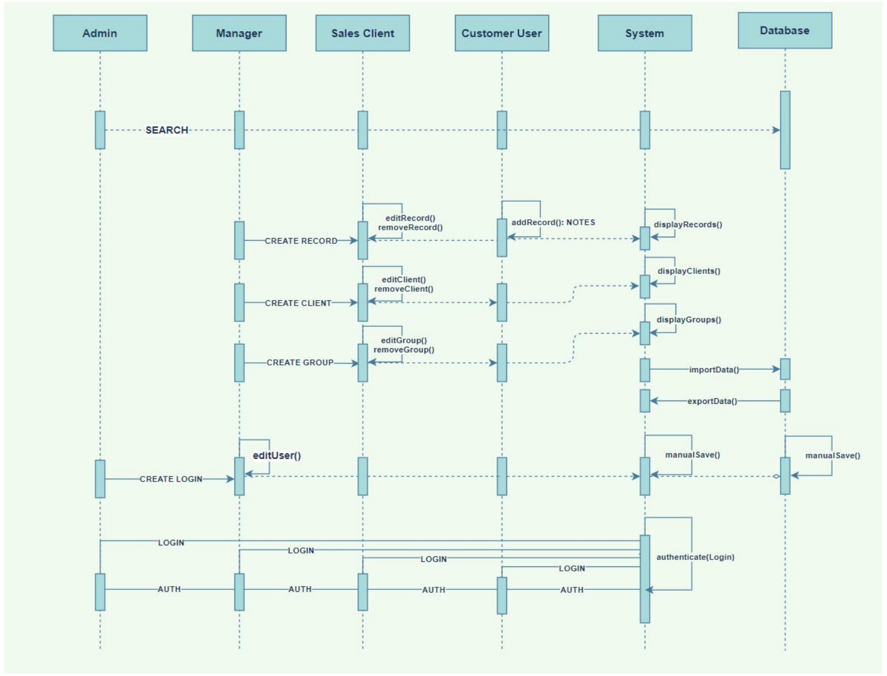
Client / Server side

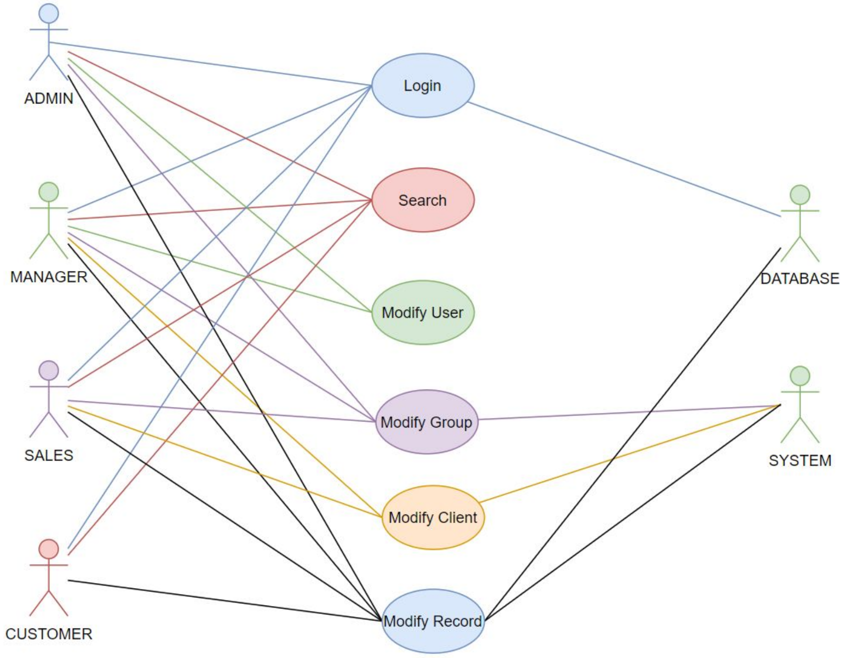
- Client: Login interaction, between the client and server. Communicates messages to the server, to verify and respond. (Confirmations, CRUD Records, Storage Commands))
- Server: Multithreaded and operates on standby. Command processes various messages through objects.

Storage Manager

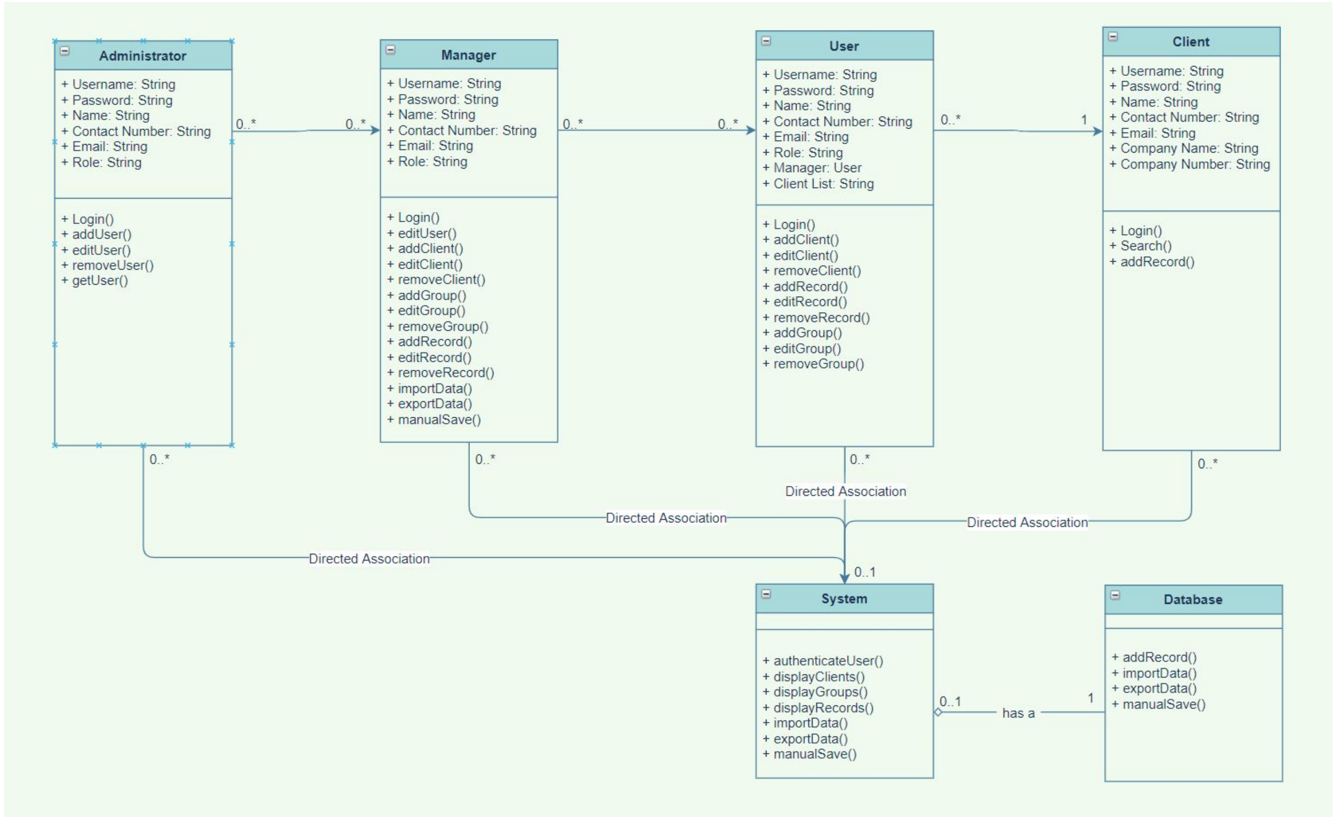
- Monitors and saves data to a file, CRUD.
- Queue structure, FIFO for requests to save data.

Interactions





Attributes

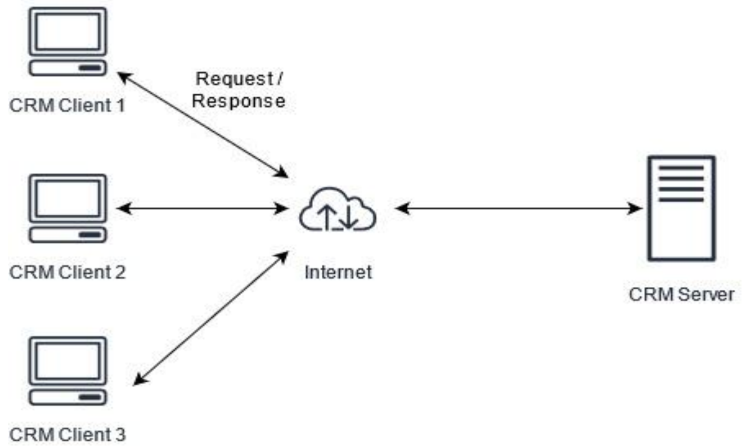

Administrator
+ Username: String
+ Password: String
+ Name: String
+ Contact Number: String
+ Email: String
+ Role: String
+ Login()
+ addUser()
+ editUser()
+ removeUser()
+ getUser()
Manager
+ Username: String
+ Password: String
+ Name: String
+ Contact Number: String
+ Email: String
+ Role: String
+ Login()
+ editUser()
+ addClient()
+ editClient()
+ removeClient()
+ addGroup()
+ editGroup()
+ removeGroup()
+ addRecord()
+ editRecord()
+ removeRecord()
+ importData()
+ exportData()
+ manualSave()
User
+ Username: String
+ Password: String
+ Name: String
+ Contact Number: String
+ Email: String
+ Role: String
+ Manager: User
+ Client List: String
+ Login()
+ addClient()
+ editClient()
+ removeClient()
+ addRecord()
+ editRecord()
+ removeRecord()
+ addGroup()
+ editGroup()
+ removeGroup()
Client
+ Username: String
+ Password: String
+ Name: String
+ Contact Number: String
+ Email: String
+ Company Name: String
+ Company Number: String
+ Login()
+ Search()
+ addRecord()
0..*
0..*
0..*
0..*
0..*
1
0..*
0..*
0..*
0..*
Directed Association
Directed Association
Directed Association
Directed Association
0..1
System
+ authenticateUser()
+ displayClients()
+ displayGroups()
+ displayRecords()
+ importData()
+ exportData()
+ manualSave()
0..1
has a
1
Database
+ addRecord()
+ importData()
+ exportData()
+ manualSave()

(High-level design) Design elements involving UML

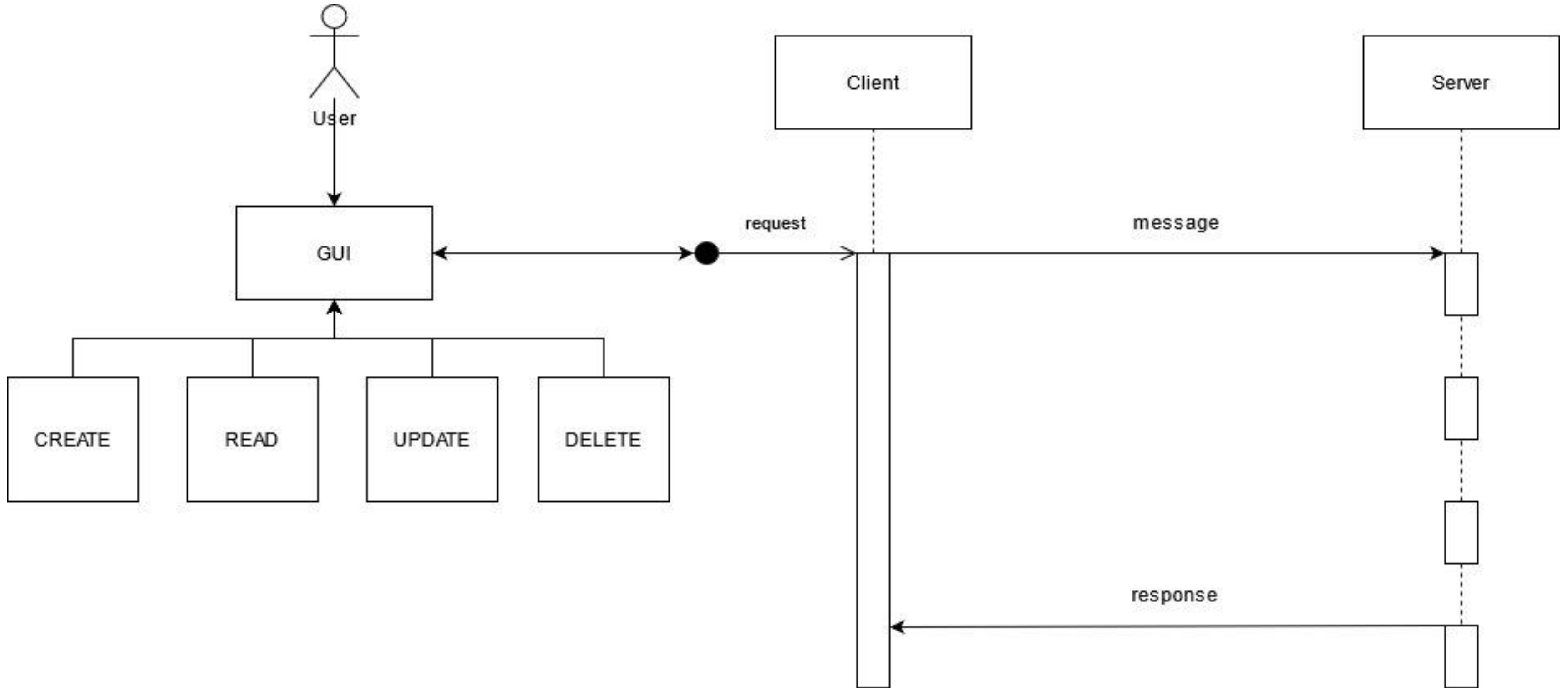
The CRM's project will run on a thread-based connection and interact over a simple graphical user interface(GUI).

Example operations are:

- creating a user account
- editing a client information
- grouping user

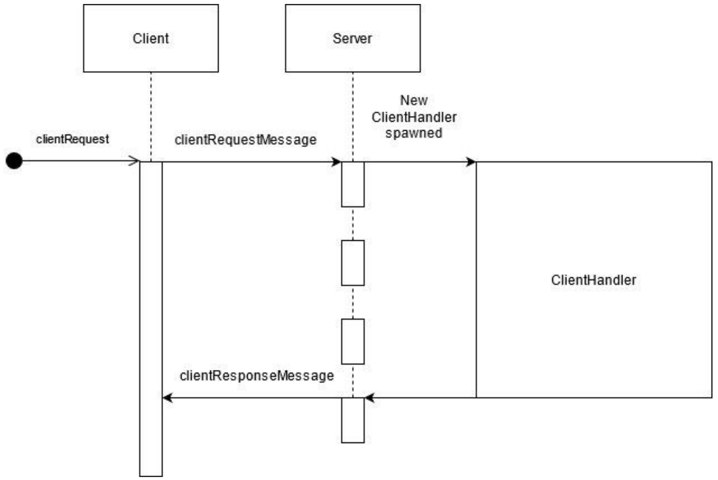


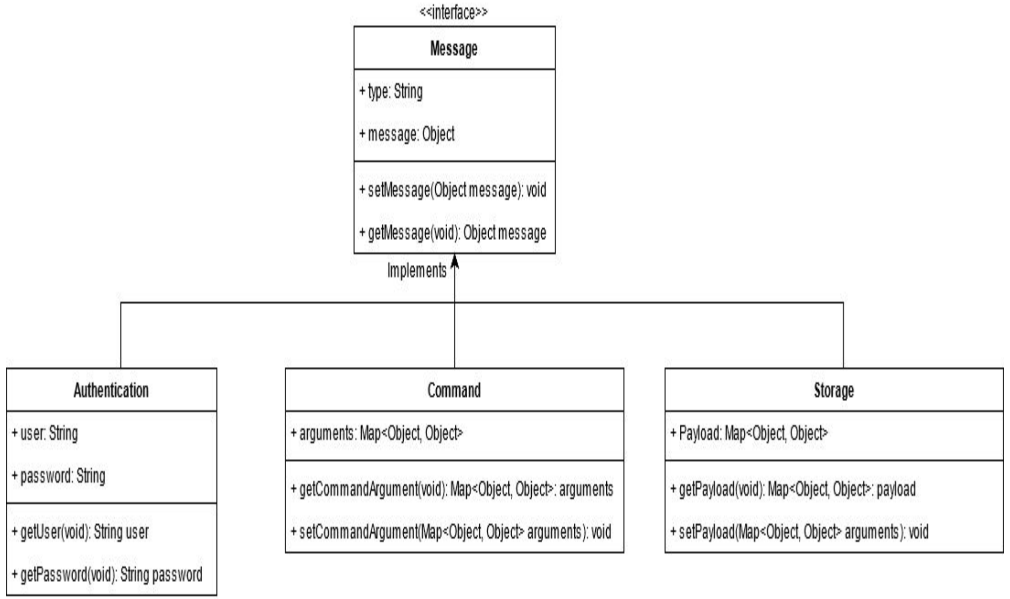
Graphical User Interface(GUI) UML



Messages

Message objects are transmitted between Client and Server who's type depends on the context



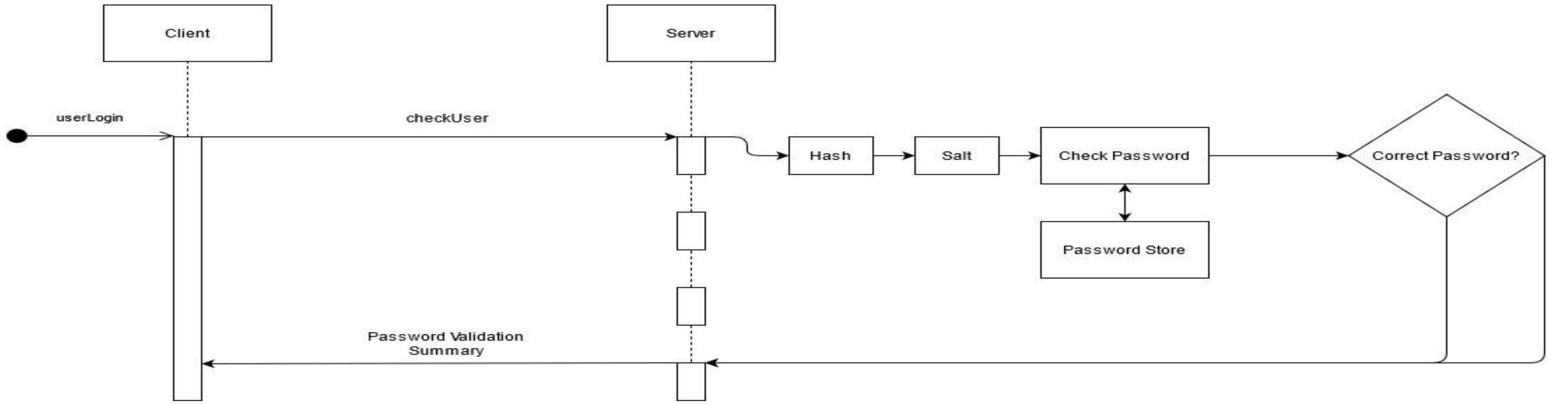

<<interface>>
Message
+ type: String
+ message: Object
+ setMessage(Object message): void
+ getMessage(void): Object message
Implements
Authentication
+ user: String
+ password: String
+ getUser(void): String user
+ getPassword(void): String password
Command
+ arguments: Map<Object, Object>
+ getCommandArgument(void): Map<Object, Object>: arguments
+ setCommandArgument(Map<Object, Object> arguments): void
Storage
+ Payload: Map<Object, Object>
+ getPayload(void): Map<Object, Object>: payload
+ setPayload(Map<Object, Object> arguments): void

Authentication Message

Client will send an **AuthenticationMessage** to Server on initial startup

Containing Username & Password - Server validates credentials

Whether credentials are good or bad - Server will send appropriate response back to the Client

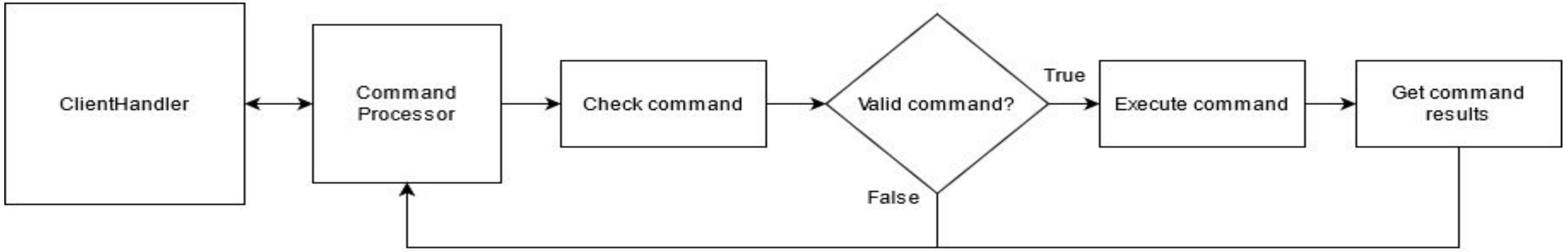


Command Message

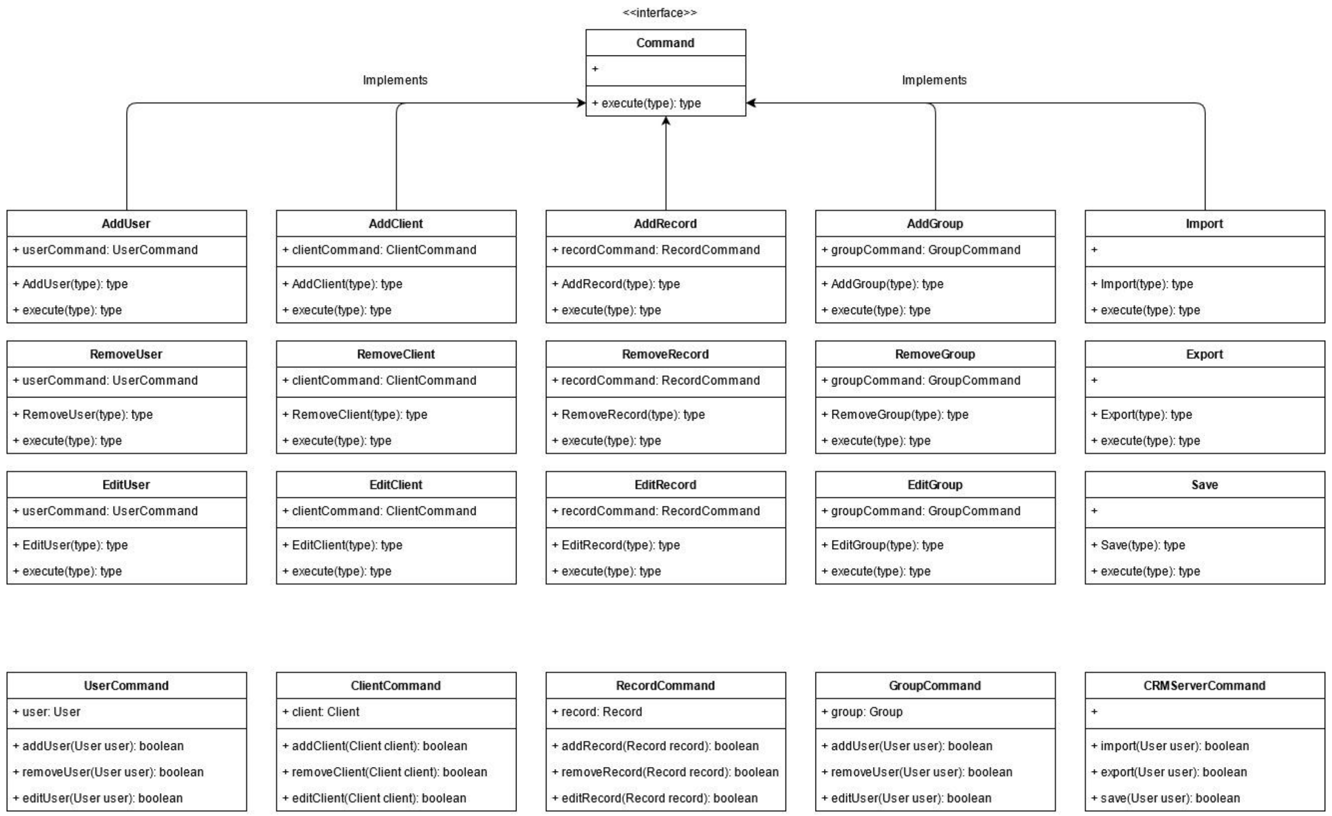
Client will send a **CommandMessage** when Client triggers an event such as: adding a new customer, creating a customer group, editing a record from the GUI.

Data is packaged in **CommandMessage** and sent to Server which passes off to **CommandProcessor**

CommandProcessor reads and interprets **CommandMessage** and performs the relative action



Commands

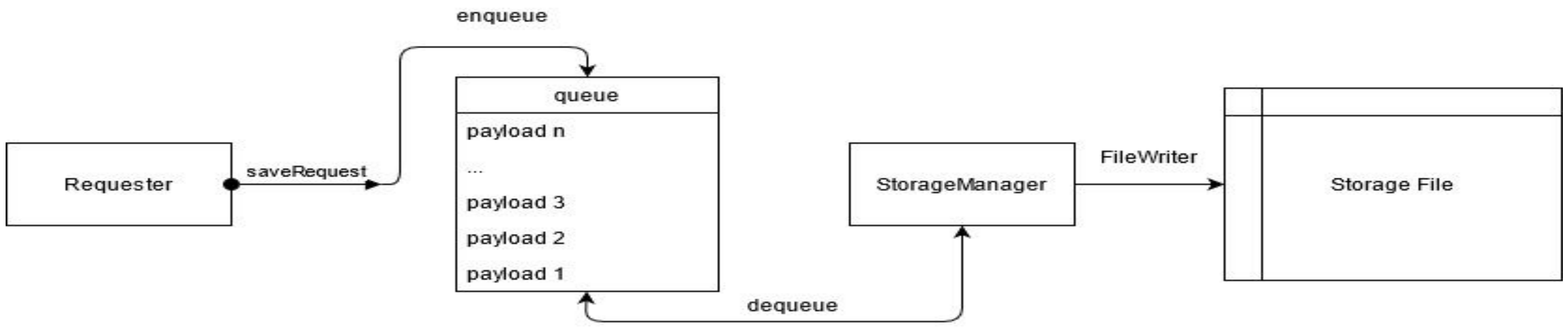

<<interface>>
Command
+
+ execute(type): type
Implements
Implements
AddUser
+ userCommand: UserCommand
+ AddUser(type): type
+ execute(type): type
AddClient
+ clientCommand: ClientCommand
+ AddClient(type): type
+ execute(type): type
AddRecord
+ recordCommand: RecordCommand
+ AddRecord(type): type
+ execute(type): type
AddGroup
+ groupCommand: GroupCommand
+ AddGroup(type): type
+ execute(type): type
Import
+
+ Import(type): type
+ execute(type): type
RemoveUser
+ userCommand: UserCommand
+ RemoveUser(type): type
+ execute(type): type
RemoveClient
+ clientCommand: ClientCommand
+ RemoveClient(type): type
+ execute(type): type
RemoveRecord
+ recordCommand: RecordCommand
+ RemoveRecord(type): type
+ execute(type): type
RemoveGroup
+ groupCommand: GroupCommand
+ RemoveGroup(type): type
+ execute(type): type
Export
+
+ Export(type): type
+ execute(type): type
EditUser
+ userCommand: UserCommand
+ EditUser(type): type
+ execute(type): type
EditClient
+ clientCommand: ClientCommand
+ EditClient(type): type
+ execute(type): type
EditRecord
+ recordCommand: RecordCommand
+ EditRecord(type): type
+ execute(type): type
EditGroup
+ groupCommand: GroupCommand
+ EditGroup(type): type
+ execute(type): type
Save
+
+ Save(type): type
+ execute(type): type
UserCommand
+ user: User
+ addUser(User user): boolean
+ removeUser(User user): boolean
+ editUser(User user): boolean
ClientCommand
+ client: Client
+ addClient(Client client): boolean
+ removeClient(Client client): boolean
+ editClient(Client client): boolean
RecordCommand
+ record: Record
+ addRecord(Record record): boolean
+ removeRecord(Record record): boolean
+ editRecord(Record record): boolean
GroupCommand
+ group: Group
+ addUser(User user): boolean
+ removeUser(User user): boolean
+ editUser(User user): boolean
CRMServerCommand
+
+ import(User user): boolean
+ export(User user): boolean
+ save(User user): boolean

Storage Message

The **Message** interface provides a way to reuse code to create a type called **StorageMessage** which is sent within the Server when data needs to be stored

StorageMessages are pushed to a **PriorityQueue** which orders **StorageMessages** by a timestamp

StorageManager reads **StorageMessages** from the queue and saves the data in the message to the database



Thank you