
# CSC 431

## College Social App

System Architecture Specification (SAS)

**Team 14**


| | |
|---|---|
| Austin Schladant | Scrum Master |
| Kyle Riley | Scrum Development Team Member |
| Bo Fei Wang | Scrum Development Team Member |

# Version History

| Version | Date | Author(s) | Change Comments |
|---|---|---|---|
| **2.0** | 4/13/21 | Austin Schladant, Kyle Riley | Final Draft |
| **1.0** | 4/1/21 | Austin Schladant, Kyle Riley, BoFei Wang | First Draft |

# Table of Contents

# Table of Tables

<Generate table here>

# Table of Figures

# 1. System Analysis

## 1.1. System Overview

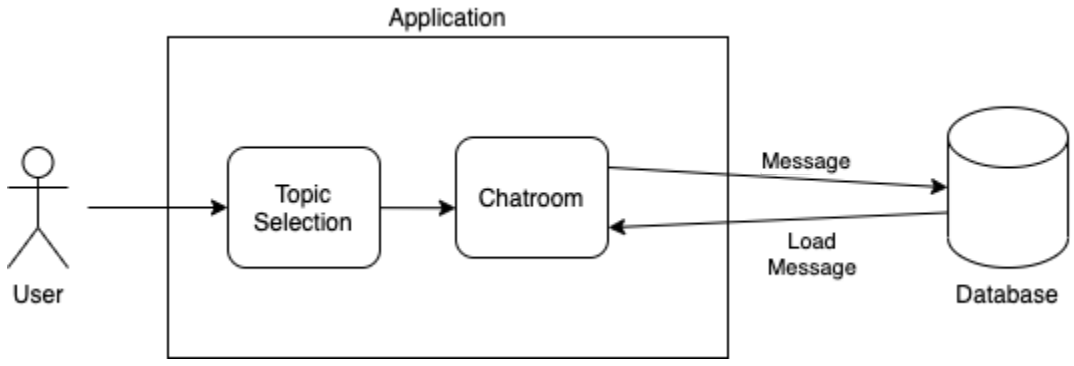
The system will consist of two main parts: the chatroom selection and  the chatroom itself. The chatroom selection will allow the user to view all the chat rooms that are stored using firebase. While the chatroom will allow users to actively send messages to the chatroom and read any incoming messages from other users.

The pipeline of the application is: (1) the user logs in, or signs up if an account isn't made, (2) the user edits their profile if they need to change password or update/add profile picture (3) the user explores chat rooms depending on what subject they select, (4) the user enters a chat room, (5) the user sends and reads messages in the chat room, (6) the user saves the chat room if they want.

The chatroom will be designed using a Model-View Controller architecture, while the navigation of the application will be designed using an Event-Driven architecture. Using a Model-View-Controller architecture will allow us to implement that functionality of sending messages, saving the messages onto the database, and then showing the messages in the chatrooms, and an Event-Driven architecture will allow us to easily program navigation through the application.

## 1.2. System Diagram

Figure 1. Application System Diagram




## 1.3. Actor Identification

There are two types of human actors - those that have an account, and those that don't. Those that do not have an account can only access the account creation and login functionality of the app, while those

that have an account have those functions and all other functions of the app, such as sending and receiving messages from various group chats. Additionally, there is a non-human actor, which is the cloud storage where the user information and chatroom messages will be saved.

## 1.4. Design Rationale

### 1.4.1. Architectural Style

In this application, the user will navigate themselves through various topics in order to find a chatroom that they want to join. Since the main functionality of the other states in our application are to navigate the application in order to find a chatroom, we decided that using an **event-driven architecture** to design these states would be best. Depending on the event caused by the user, i.e. the user pressing a button, the application will then act accordingly. For example, the user will select a topic by pressing a button, this will trigger the application to display the rooms pertaining to the topic, then once the user selects a room, the application will enter the chatroom selected.

The chatroom will have a **model-view-controller (MVC) architecture**. In this architecture, the controller represents the text field where the user inputs their message, the model represents the database where the messages are stored, which is firebase, and the view represents the text box where the messages show up in the chat room. The flow of this architecture is: the user inputs a message into the controller, the controller then handles this message and sends it to the model, then the model saves the message, and finally, the view grabs the message and displays it in the chatroom.

### 1.4.2. Design Pattern(s)

We anticipate that we are going to use the following design patterns:

1. The chatrooms are going to be designed using the **mediator pattern**. This object is going to encapsulate the User object and the Database base and add functionality between the two.
2. The chatroom selection page is going to be designed using the **factory method**. No matter the topic, the chatroom selection page is always going to be a list of chatrooms, but depending on what the user selected, there are going to be different chatrooms displayed. The factory method allows us to implement the structure for a Chatroom Selection Object, then depending on the user input, the application will implement an interface of this object. For example, if the user selects the “Academics” topic, the Academic Chatroom Selection Object will be instantiated.

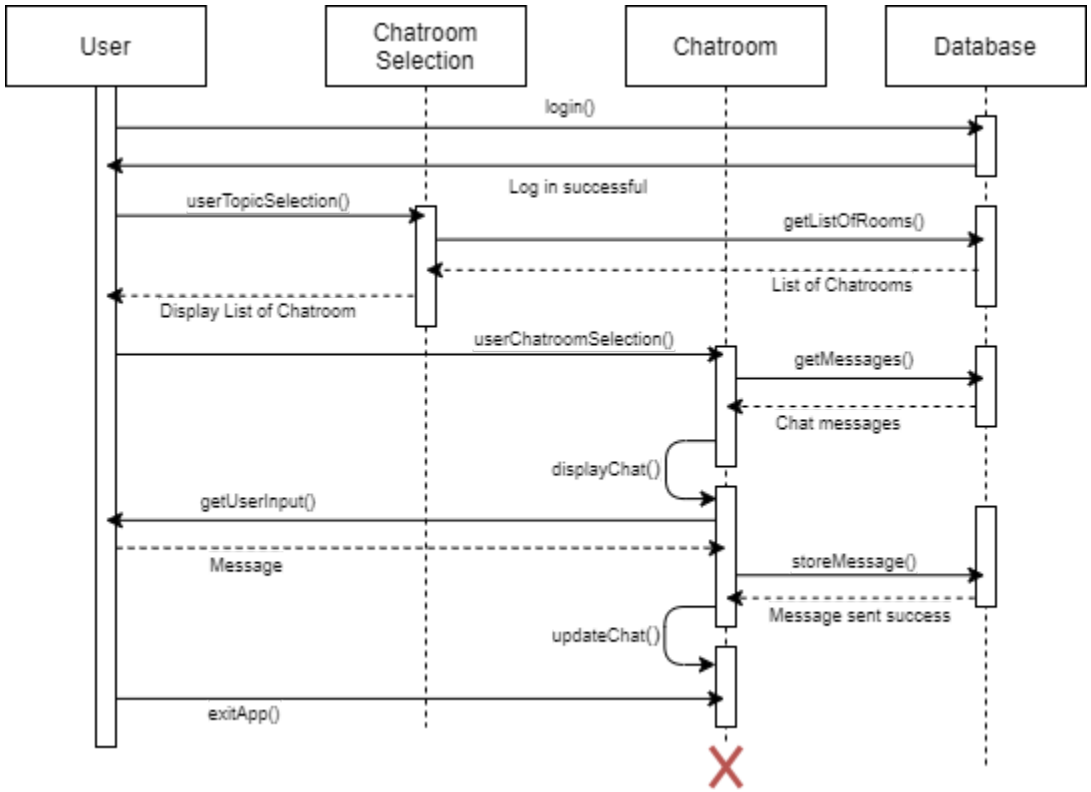
### 1.4.3. Framework

This application is going to be using the Flutter UI toolkit and the Firebase API. Using Flutter and Dart will allow the program to interact with Firebase storage to allow users to message each other in chat rooms.

# 2. Functional Design

## 2.1. Sequence Diagram

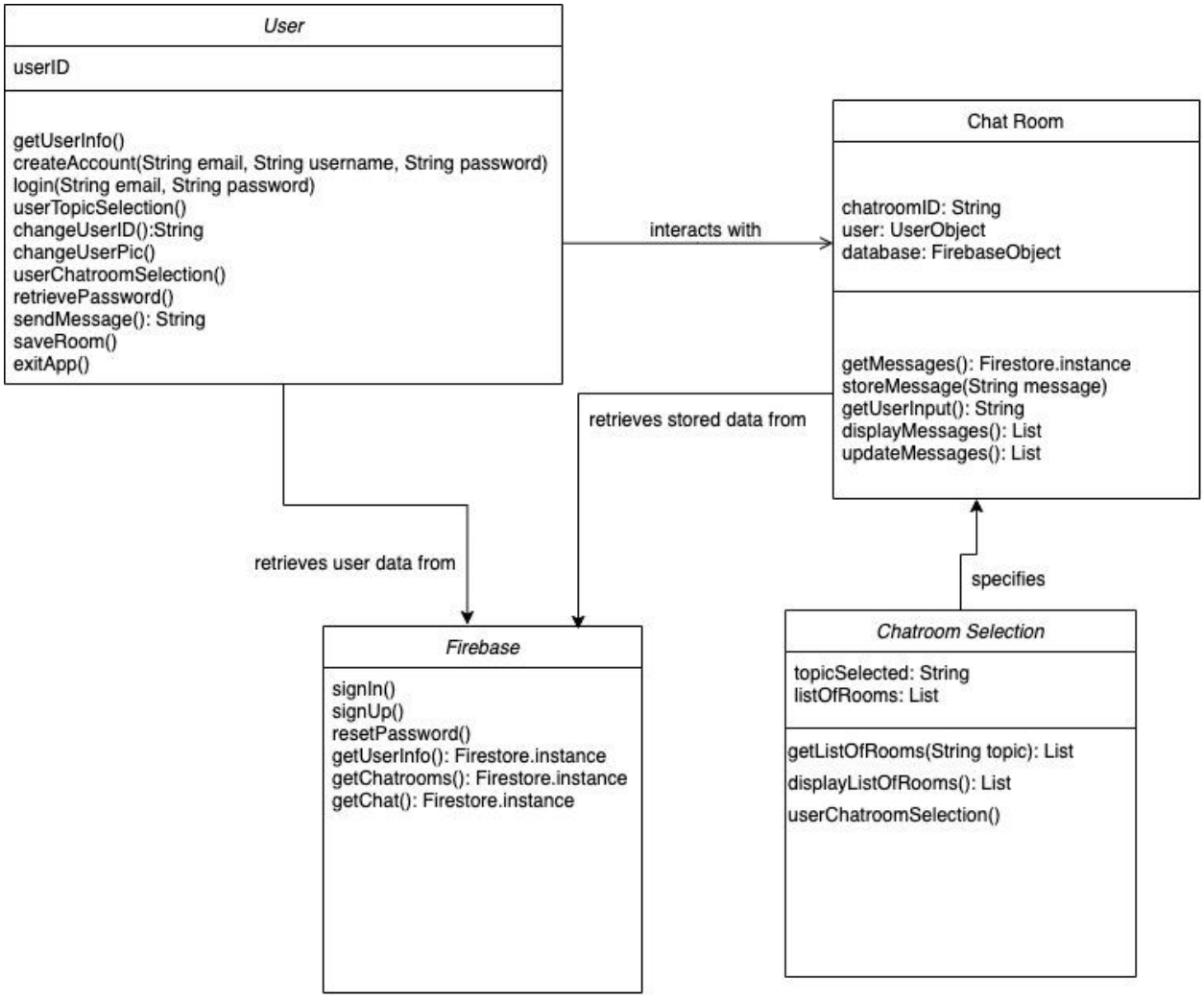
Figure 2: Sequence Diagram




- When the user logs in, their email and password authenticated using the Firebase API.
- After the user logs in, they have the option to select a topic, and once they do that, the Chatroom Selection then accesses Firebase to get the list of chatrooms under that topic and displays it to the user.
- The user then has the option to select any of the chatrooms that are displayed. Once selected, the Chatroom object is called, and the Chatroom gets the messages from Firebase and displays the chatroom.
- The Chatroom object then waits for the user input. When the user inputs a message, the Chatroom object then stores the message in Firebase, and updates the chat.

# 3. Structural Design

Figure 3: Class Diagram




- Firebase will serve as the database that stores information.