Requirements Engineering

Instant Chat App

Group 9 - Comp 680

About Team

Team Members

Jeel Patel

project proposal
system requirements
Project Report
User Stories
Use Case
Sequence Diagrams
Code
Video Recording
Video Editing
UI Design
Traceability Matrix
System Architecture

Carlos Lima

project proposal
system requirements
User Stories
Use Case
Sequence Diagrams
Code
Video Recording
Unit Testing
Schema for Use Cases

Hossein Alishah

project proposal
system requirements
Project Report
User Stories
Use Case
Sequence Diagrams
Code
Video Recording
UI Design
Unit Testing
Schema for Use Cases
App Demo

Harisha Dheeravath

project proposal
system requirements
User Stories
Use Case
Sequence Diagrams
Testing
Video Recording
Schema for Use Cases

Table Of Contents

Summary of Changes

1. Using two different pages for sign up instead of a single page. First, the user puts the valid email address and password, then on the second page, select profile picture and username.

2. Added contact page which user can use to start a new conversation with any contact which has a valid email address that already signed up in the app.

3. Added the image button on the home page using which a user without going to chatroom or making a new chatroom, can send pictures to another user.

Project Proposal

The topic for our project is a real time group chat application to improve communication and collaboration for students and professionals. This app would have an easy to use interface and send messages in real time to a single user or a group. All conversations the user has been a part of will be available to view at any time. Each user will need to authenticate themselves by creating an account and logging in so we can validate where all of the messages are coming from.

In this age of virtual work and schooling, collaboration and communication has become more difficult. Groups and teams need a way to communicate, but sending an email or a phone call is not always the best choice. Online communication has become the preferred method of communication today and we want to help people easily connect online. This application would benefit students, professionals and anyone looking for a way to communicate with others. This topic was interesting to our group because it is a helpful tool for many people and requires a lot of technical planning in order to complete.

The main problem this application will solve is miscommunication and disunity in school and professional environments. When people are in the same location it can be easy to go up to someone and ask a question or update your status on something, but when you are behind a computer in a different location this can be more difficult. We want to create a platform that allows individuals to communicate in real time as if they were together while they are in different locations. The traditional email doesn't always get an immediate reply and people can be left waiting for a reply for something very simple. This group chat application will provide a method of communication that is less formal than the traditional email but implies some urgency like a face to face interaction.

Tech Stack

Node JS, React JS, Firebase DB

In order to accomplish this our team plans to create a UI using React JS and a backend using Node JS. We plan to use Firebase for our data storage needs which will be easier to scale.

System Requirements

Enumerated Functional Requirements

Identifier	Priority	Requirements
REQ1	4	The application shall start with the log-in page that gets the username and password from the user.
REQ2	4	The application shall have a forget password bottom for a user who has forgotten the account password.
REQ3	3	The application shall connect to a database system such as firebase to store all the user information and avoid duplication.
REQ4	5	The application will connect to a database where all messages will be stored
REQ5	4	The application will add every message sent to the database when it is sent
REQ6	4	The application will retrieve the list of messages for the selected conversation to display them
REQ7	2	The user will be able to delete their own messages from the database so they are no longer displayed in the conversation
REQ8	2	When the user deletes the conversation both parties will lose access to the previous messages in that conversation
REQ9	2	If a user deletes conversations, the system will delete all messages that are part of that conversation from the DB.
REQ10	5	The app provides a text input area to type the text message.
REQ11	5	The app shall provide a Send button to send the typed message to the receiving user/s.
REQ12	4	The app provides a button to select one or multiple files/images from the device to send them in the chat.
REQ13	5	The app shall create Voice messages,voice notes and group calls.
REQ14	3	The app should be able to revert all actions done through the delete feature.

Enumerated Non-Functional Requirements

Identifier	Priority	Requirements
REQ15	5	The application shall lock an account for 24 hours if the user entered the wrong password more than three times.
REQ16	3	The application shall ask the new user some personal information such as first name, last name, email address, phone number, password, and confirm password to sign up.
REQ17	3	The retrieval of messages for each conversation will be quick and limit the total number of messages loaded until the user wants to load more messages
REQ18	5	The app shall provide a list of messages previously sent in a chronological order.
REQ19	4	The user shall be displayed with the sent message almost instantly and with minimum lag, since this app is an instant chat app.
REQ20	3	The app shall not send "empty text" messages.
REQ21	5	The app shall provide functionality to save the received file/image in local storage of the device.

Enumerated User Interface Requirements

Identifier	Priority	Requirements
REQ22	3	The application must ask the new user to put a proper email address, phone number, and password and show the error notification for inappropriate information.
REQ23	4	In the Messenger UI for each conversation there will be previous messages that the user can view
REQ24	2	The application will allow the user to delete conversation they are a part of at any time from the UI
REQ25	4	The app displays messages received on the left side and messages sent on the right side.
REQ26	3	The app shall display the message sent time.

REQ27	5	The app shall display the sent images/files in the chat messages area.
REQ28	3	The app has a function called join room where new users can join the room.
REQ29	3	The app has a function called join room after joining the room users can leave after conversations.
REQ30	2	The application should make a sound when the user receives a notification so they can be alerted even if they are not looking at the messenger at the moment
REQ31	1	The Application should display a visual when they receive a notification so that they can see the alert even if the sound is muted.

FunctionalRequirements

Actors and Goals

Actor	Actor's goal	Use Case Name
App Interface	Show login/signup page to the user upon opening the app	(UC-1) start page
App Interface	Show Forgot password button upon incorrect password entry	(UC-2) Forgot password
App Interface	Make a connection with the DataBase to access and store all user data	(UC-3) Connection with DataBase
DataBase	Remove deleted messages from the DataBase.	(UC-4) Delete Messages
User	Selects one or multiple files/images to send to the receiver.	(UC-5) Send File
Storage	Presses button To Store the files/images received	(UC-6) Storing Locally
Speaker	Turns on notifications To Make a sound when new messages arrives	(UC-7) Notification
User	Types a message and press the send button to send a message.	(UC-8) sending message
User	Scrolls the message list To view previous messages from a conversation	(UC-9) see messages
User	Scrolls the chat list To see the messages from different conversations at any time	(UC-10) see conversations
User	Presses button To delete messages from a conversation	(UC-11) delete message
User	Presses button To delete an entire conversation	(UC-12) delete conversation
User	Press the button To save every message sent, AKA export the conversation.	(UC-13) save message

Casual Use Case Description

(UC-1) start page
Related Requirements: REQ-1 REQ-2 REQ-15 REQ-16 REQ-22 Initiating Actor: User Actors goal: To open the app to use it for messaging Participating Actors: API, UI Preconditions: Working internet connection, Server must be up Flow of Events: ● User clicks on the app icon ● App launches and loads the start/login page

(UC-2) Forgot password
Related Requirements: REQ-1 REQ-2 REQ-15 REQ-16 Initiating Actor: user Actors goal: to reset the credentials for their account Participating Actors: API, DB Preconditions: None Flow of Events: ● User tries to login but can't remember the password ● They clicks on the forget password button ● The API will send a reset link to the associated email ● User uses the reset link to create the new password

(UC-3) Connection with DataBase
Related Requirements: REQ-3 REQ-4 Initiating Actor: API Actors goal: To make a working connection with a database Participating Actors: BD Preconditions: API server must be up, database server must be up Flow of Events: ● The API uses the database credentials and connects with the database server ● Database makes a connection with the API server.

(UC-4) Delete Messages
Related Requirements: REQ-7 REQ-8 REQ-9 Initiating Actor: user Actors goal: to remove the previously sent messages

Participating Actors: API, DB
Preconditions: API server and DB server must be up, the sent message must be sent by current user
Flow of Events:

- The user will select the message
- User will click on the delete button
- Selected message will be deleted from the DB

(UC-5) Send File

Related Requirements: REQ-12 REQ-27
Initiating Actor: user
Actors goal: to send a image or a file to the receiver party or a chat croup
Participating Actors: API, Local Storage, BD
Preconditions: API and DB server must be up, access permission of local storage
Flow of Events:

- User will click on “send file” button
- User will select one or multiple files/images
- The selected file will be uploaded to the BD
- The receiver will receiver the send files/images

(UC-6) Storing Locally

Related Requirements: REQ-8 REQ-21
Initiating Actor: user
Actors goal: to save the received message, image or files to the local storage
Participating Actors: local storage, DB
Preconditions: DB server must be up, access to the local storage
Flow of Events:

- User selects the messages, images or files from the chat list of received messages
- The app will ask for the access permission for the local storage
- The selected messages will be downloaded from the DB
- The messages will be saved to the local storage of users device

(UC-7) Notification

Related Requirements: REQ-30 REQ-31
Initiating Actor: API
Actors goal: to inform about new received message
Participating Actors: DB
Preconditions: API and DB server must be up and running
Flow of Events:

- Some other user sent a message to current user
- The API will inform the DB about the new message to store it no the DB
- The API also inform the UI to sound a notification for the user's device
- The notification will contain sent message and sender's name

(UC-8) sending message

Related Requirements: REQ-4 REQ-5 REQ-10 REQ-11 REQ-12 REQ-13 REQ-20
Initiating Actor: user
Actors goal: to send a text, image, file message to the group chat or a individual user
Participating Actors: API, DB, Local Storage
Preconditions: API and DB server must be up
Flow of Events:

- The user will type the text message in the message input bar
- The user will select an image or file if they want to send it.
- The message will be sent to the receiver party
- The file, image will be uploaded to the DB and sent to the receiver party

(UC-9) see messages

Related Requirements: REQ-4 REQ-17 REQ-18 REQ-19 REQ-23 REQ-25 REQ-26 REQ-27
Initiating Actor: user
Actors goal: to see a list or sent and received messages
Participating Actors: API, DB
Preconditions: API and DB server must be up
Flow of Events:

- The user will open the chat
- The API will load the chat messages from the DB
- The user will be able to see and scroll through the sent and received messages

(UC-10) see conversations

Related Requirements: REQ-4 REQ-6 REQ-17 REQ-18 REQ-25 REQ-26
Initiating Actor: user
Actors goal: to see all group names and individual user's chat listing
Participating Actors: app UI, DB
Preconditions: user must be logged in, API and DB server must be up
Flow of Events:

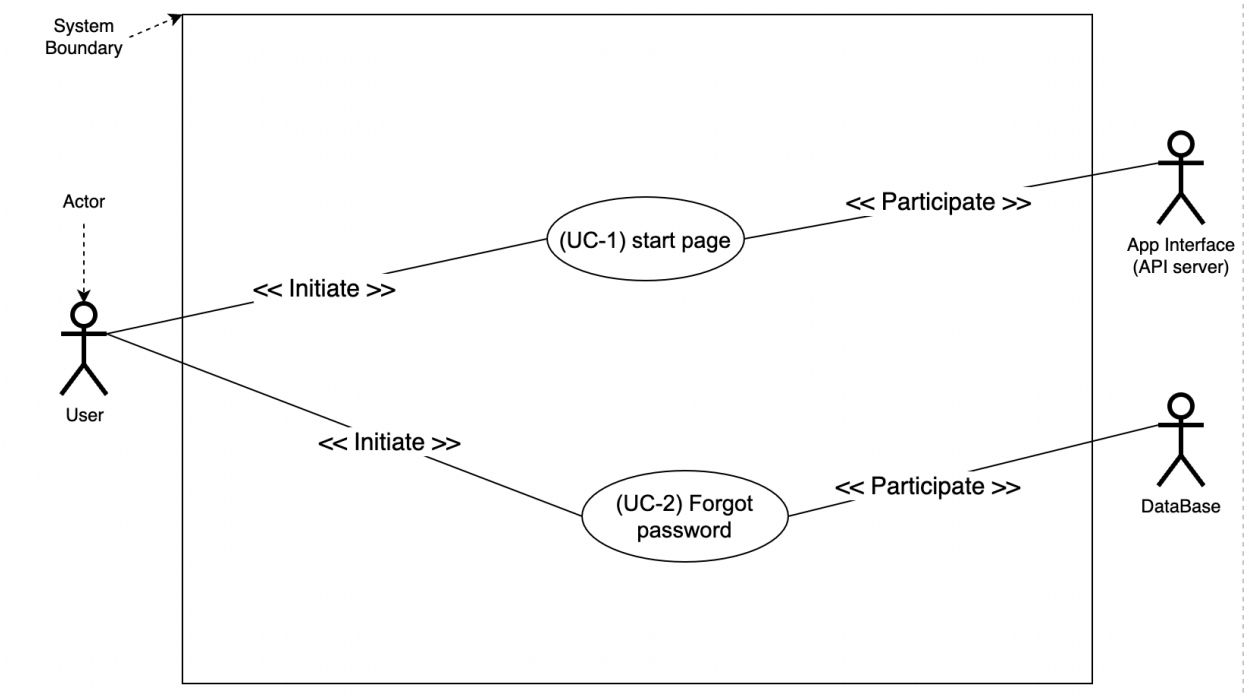
- The user will click on the app icon and open the app
- The user will login/signup for the app
- The app will load and list the conversations list

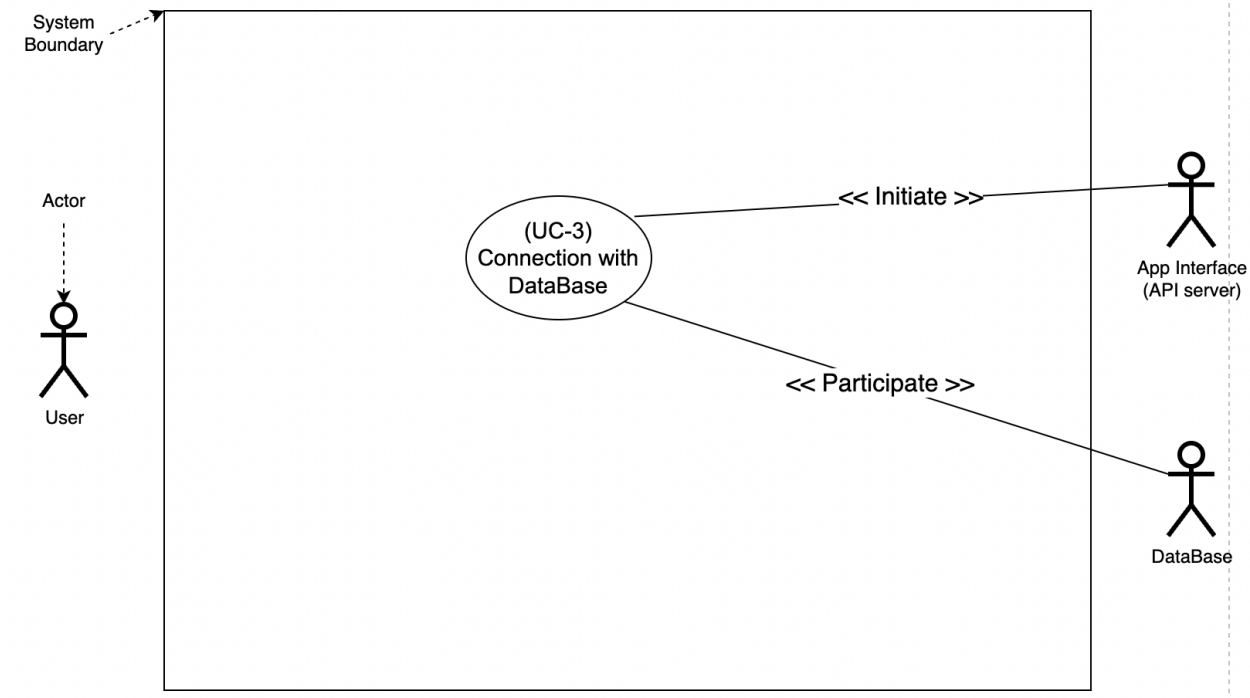
(UC-11) delete message
Related Requirements: REQ-7 REQ-9 REQ-14 Initiating Actor: user Actors goal: to remove previously sent text, file or image message Participating Actors: app UI, DB Preconditions: user must have selected a message they sent previously, DB server must be up Flow of Events: ● The user will select a previously sent message ● The user will click on the delete button ● The API will request the DB to delete the message

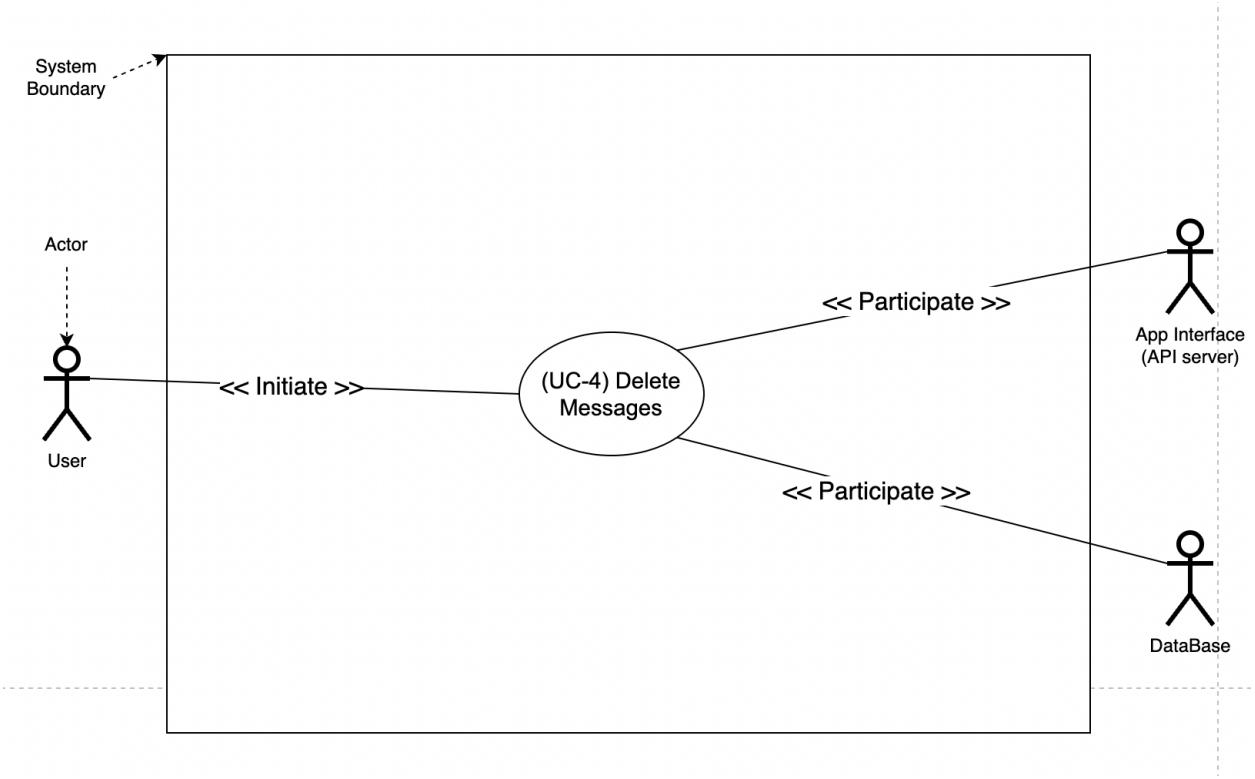
(UC-12) delete conversation
Related Requirements: REQ-7 REQ-8 REQ-9 REQ-14 REQ-24 Initiating Actor: Messenger user Actors goal: Delete conversations permanently. Participating Actors: Messenger user,Messenger DB,Messenger system. Preconditions:User has created an account,started a conversation and sent or received messages. Flow of Events: ● User selects a conversation to delete and clicks delete conversation ● Triggers an call to the messenger system to delete the conversation and all messages ● Messages for the conversation deleted from the DB

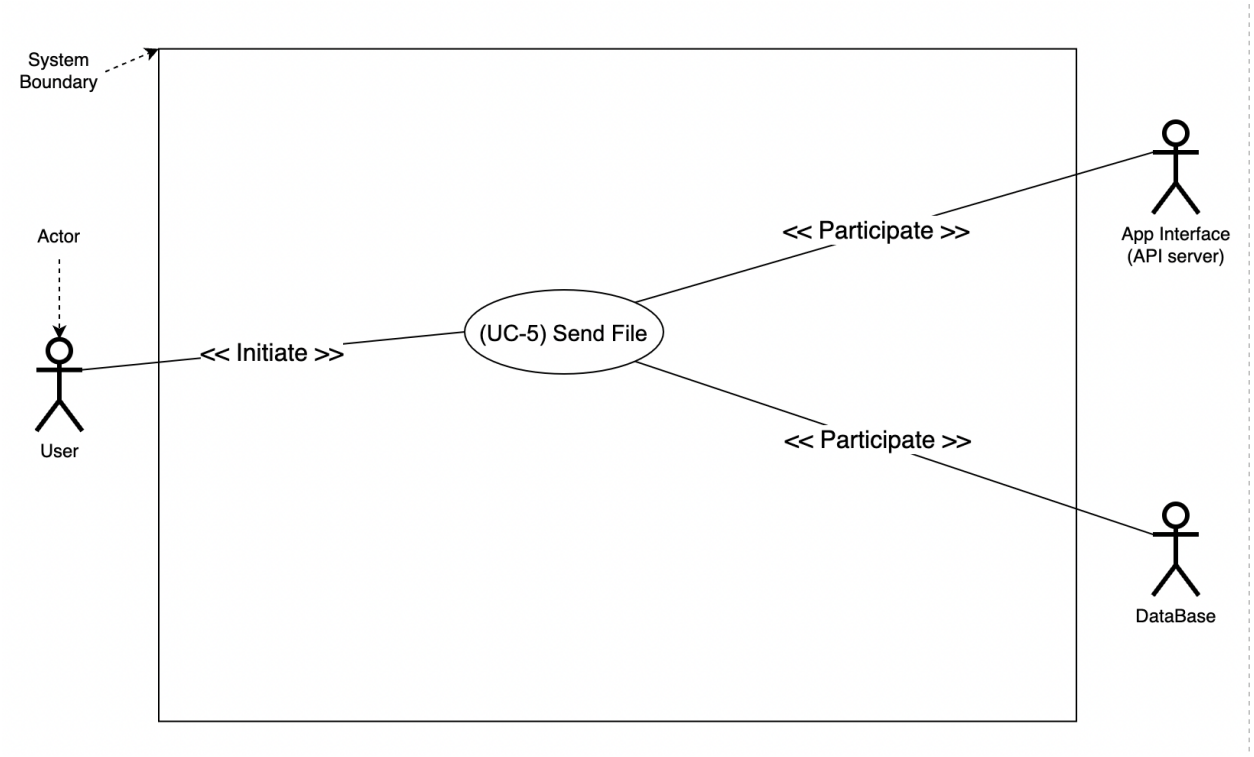
(UC-13) save message
Related Requirements: REQ-4 REQ-5 REQ-21 Initiating Actor: user Actors goal: to export the messages of the conversation to the local storage of the device Participating Actors: DB Preconditions: DB server must be up, access to the local storage Flow of Events: ● The user will click on “export chat” button ● The API will download the conversation form the DB ● The downloaded messages will be saved on the user’s local storage of the device

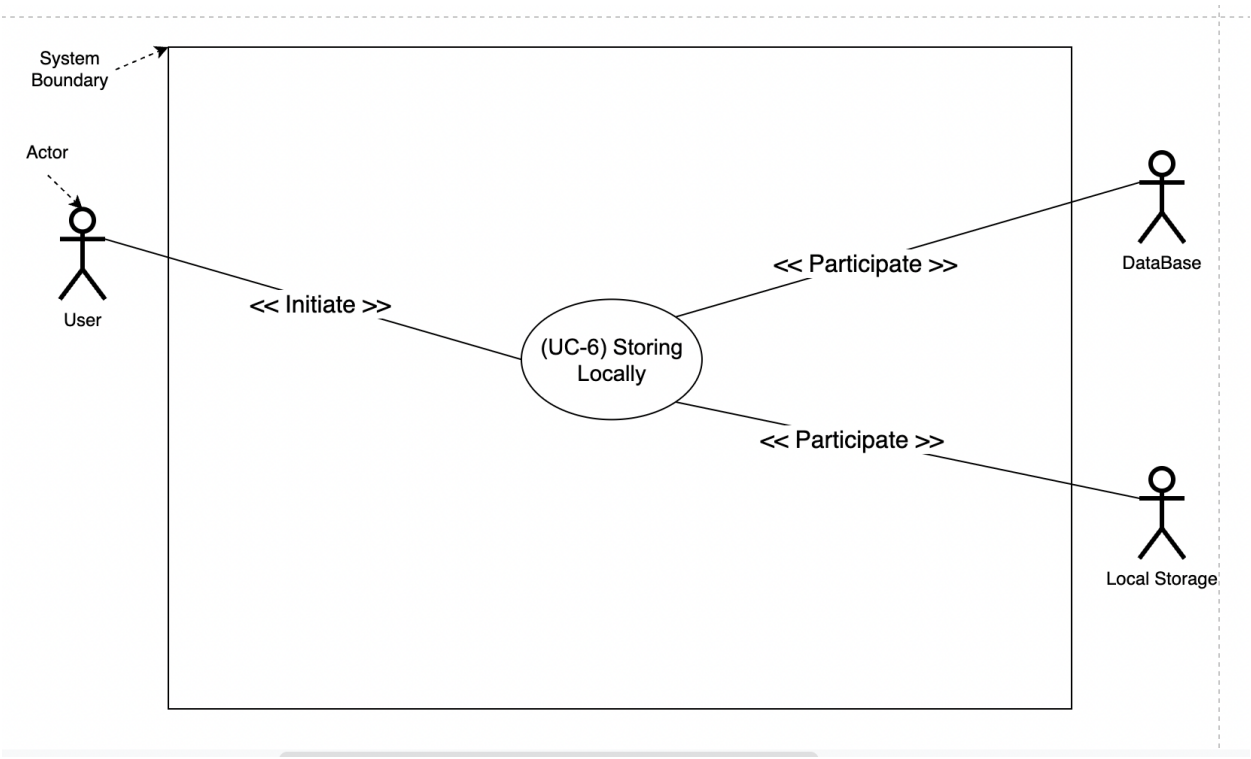
Use Case Diagrams

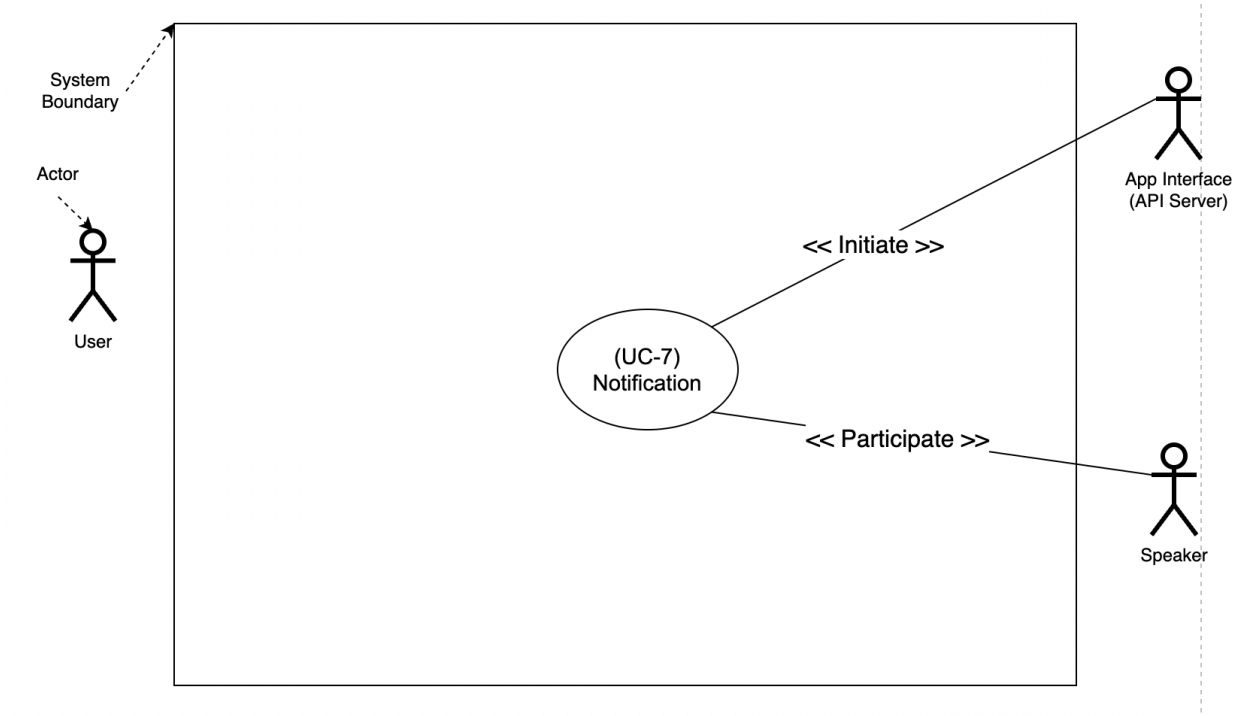


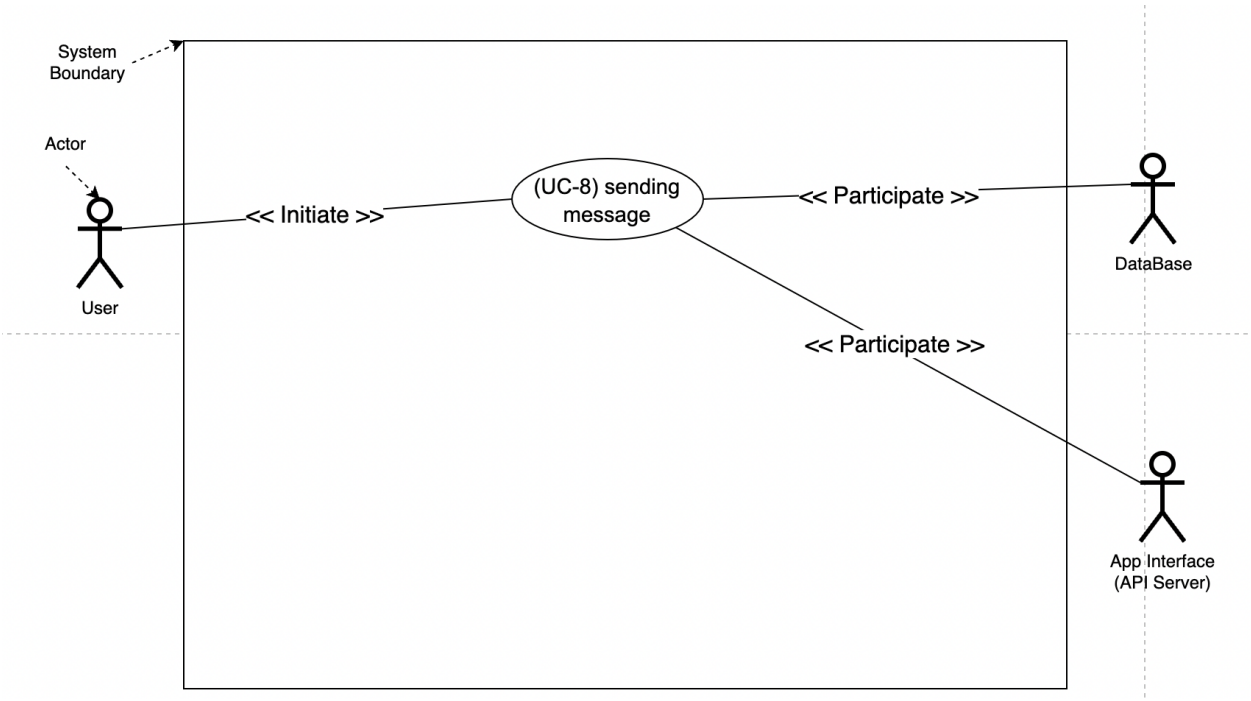


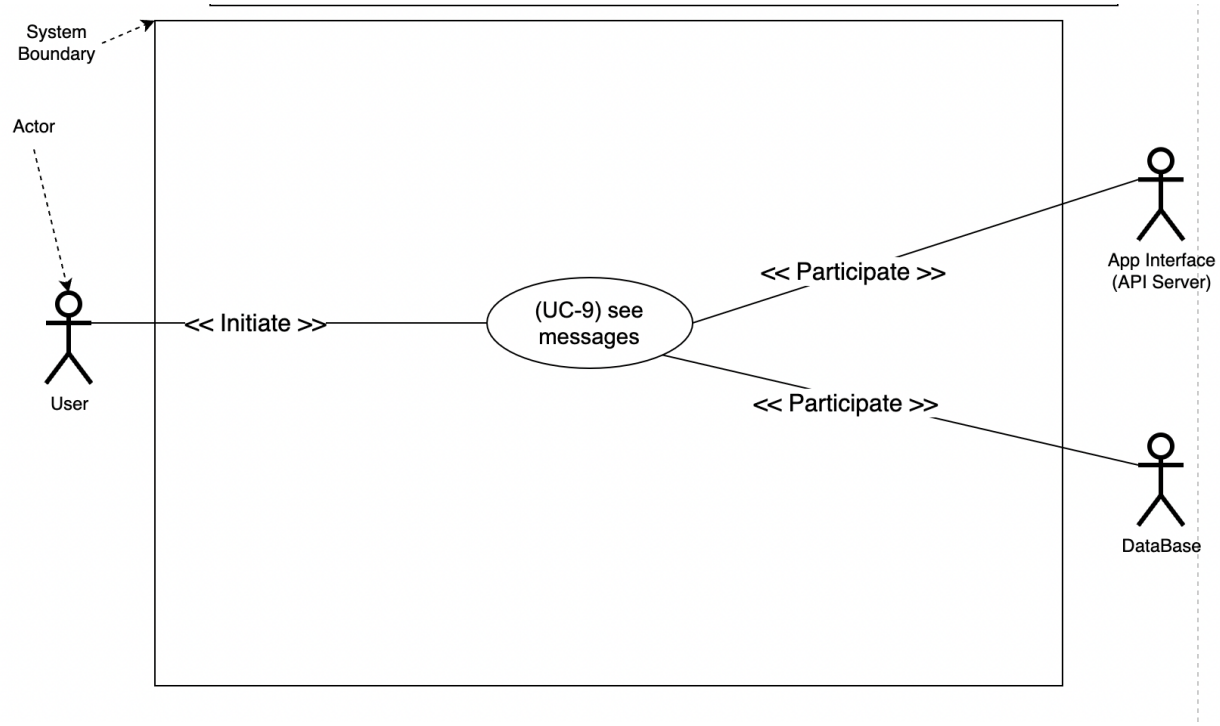

System
Boundary
Actor
User
<< Initiate >>
(UC-4) Delete
Messages
<< Participate >>
App Interface
(API server)
<< Participate >>
DataBase

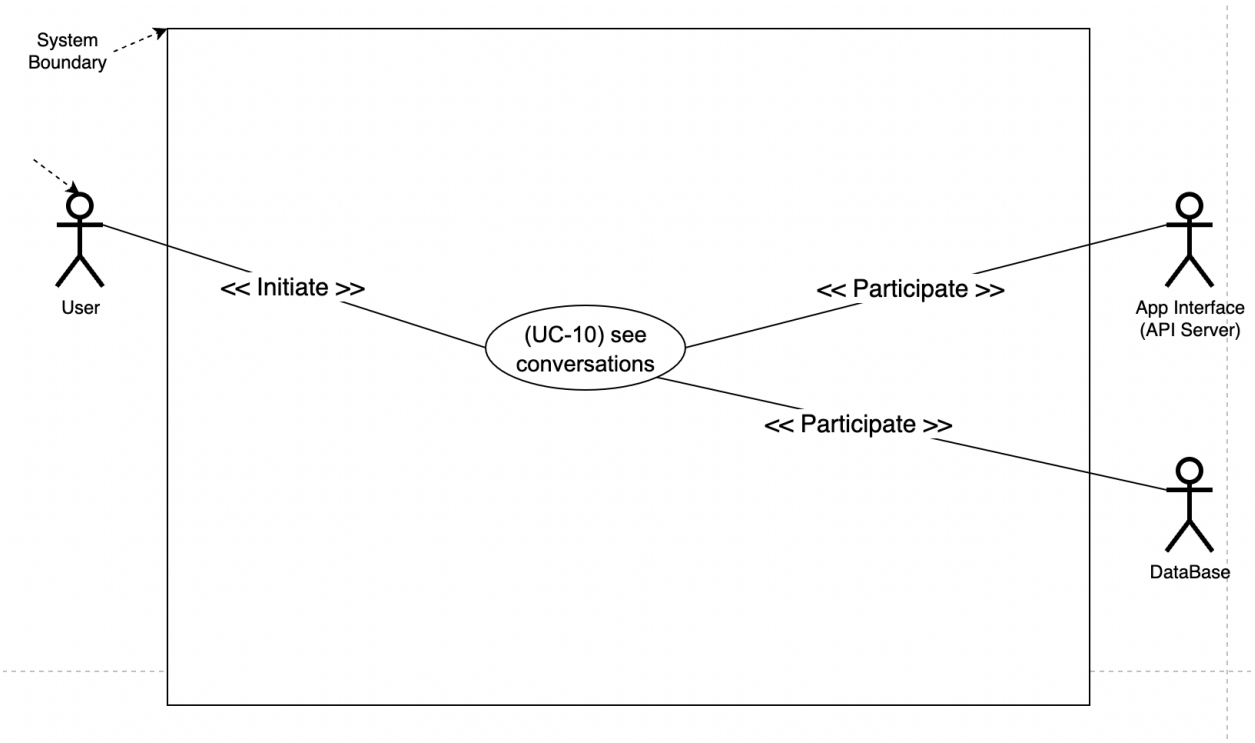

System
Boundary
Actor
User
<< Initiate >>
(UC-5) Send File
<< Participate >>
App Interface
(API server)
<< Participate >>
DataBase

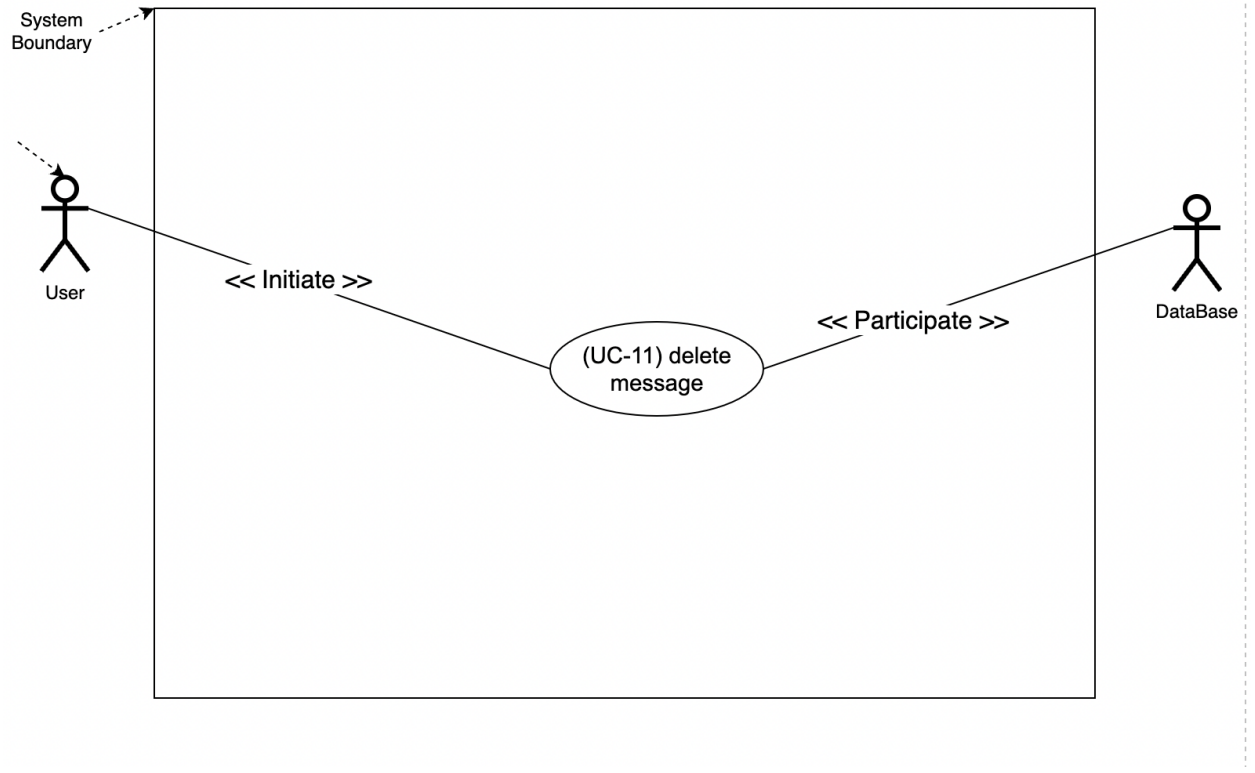

System Boundary
Actor
User
<< Initiate >>
(UC-6) Storing Locally
<< Participate >>
DataBase
<< Participate >>
Local Storage

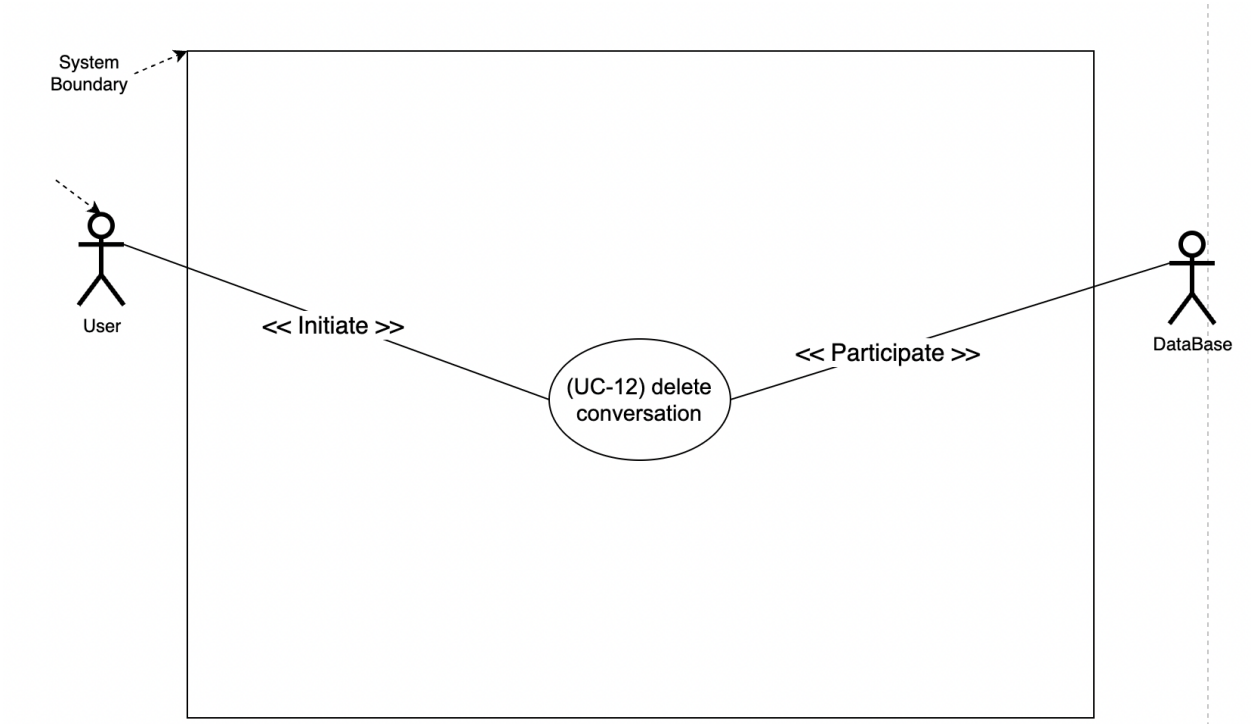

System Boundary
Actor
User
<< Initiate >>
(UC-7) Notification
<< Participate >>
App Interface (API Server)
Speaker

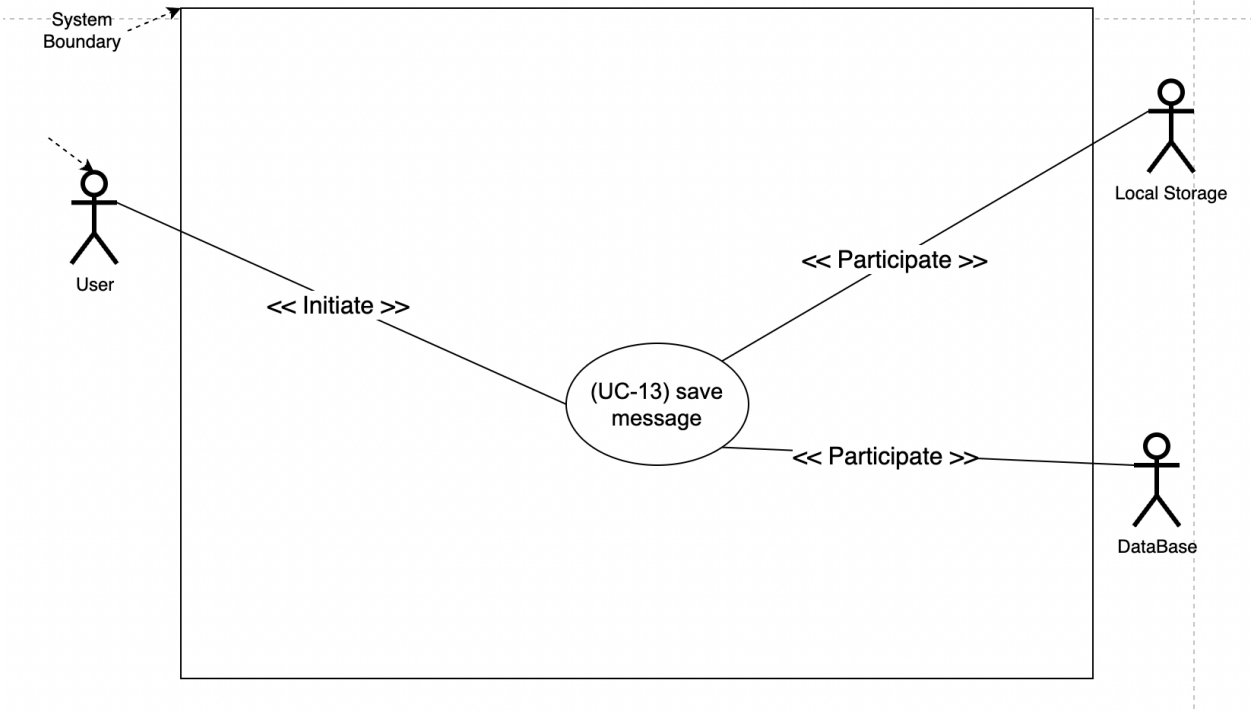

System
Boundary
Actor
User
<< Initiate >>
(UC-8) sending
message
<< Participate >>
DataBase
<< Participate >>
App Interface
(API Server)


System
Boundary
Actor
User
<< Initiate >>
(UC-9) see
messages
<< Participate >>
App Interface
(API Server)
<< Participate >>
DataBase


System
Boundary
User
<< Initiate >>
(UC-10) see
conversations
<< Participate >>
App Interface
(API Server)
<< Participate >>
DataBase


System
Boundary
User
<< Initiate >>
(UC-11) delete
message
<< Participate >>
DataBase


System
Boundary
User
<< Initiate >>
(UC-12) delete
conversation
<< Participate >>
DataBase


System
Boundary
User
<< Initiate >>
(UC-13) save
message
<< Participate >>
Local Storage
<< Participate >>
DataBase

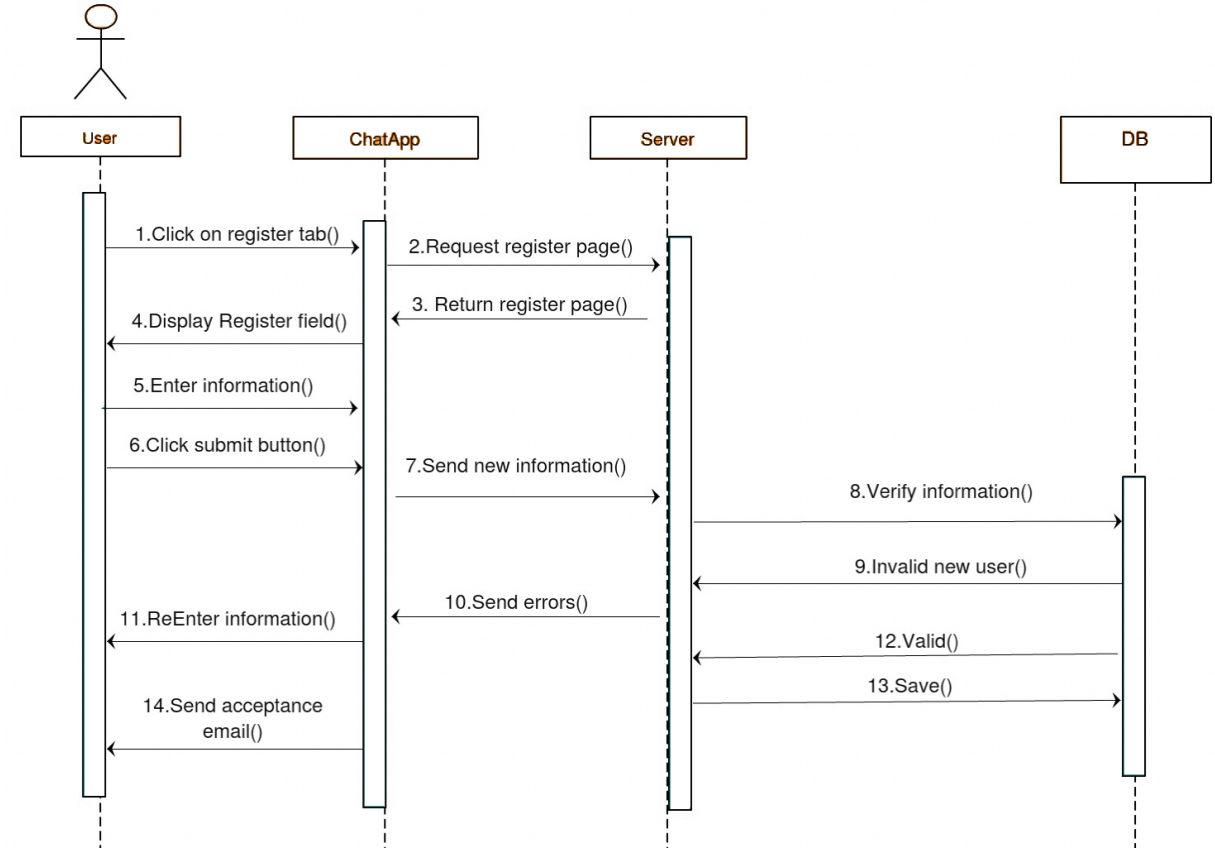
Traceability Matrix

Req't	PW	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	UC11	UC12	UC13
REQ1	4	x	x											
REQ2	4	x	x											
REQ3	3			x										
REQ4	5			x					x	x	x			x
REQ5	4								x					x
REQ6	4										x			
REQ7	3				x							x	x	
REQ8	3				x		x						x	
REQ9	3				x							x	x	
REQ10	5								x					
REQ11	5								x					
REQ12	4					x			x					
REQ13	5								x					
REQ14	3											x	x	
REQ15	5	x	x											
REQ16	3	x	x											
REQ17	3									x	x			
REQ18	5									x	x			
REQ19	4									x				
REQ20	2								x					
REQ21	5						x							x
REQ22	3	x												
REQ23	4									x				
REQ24	2												x	

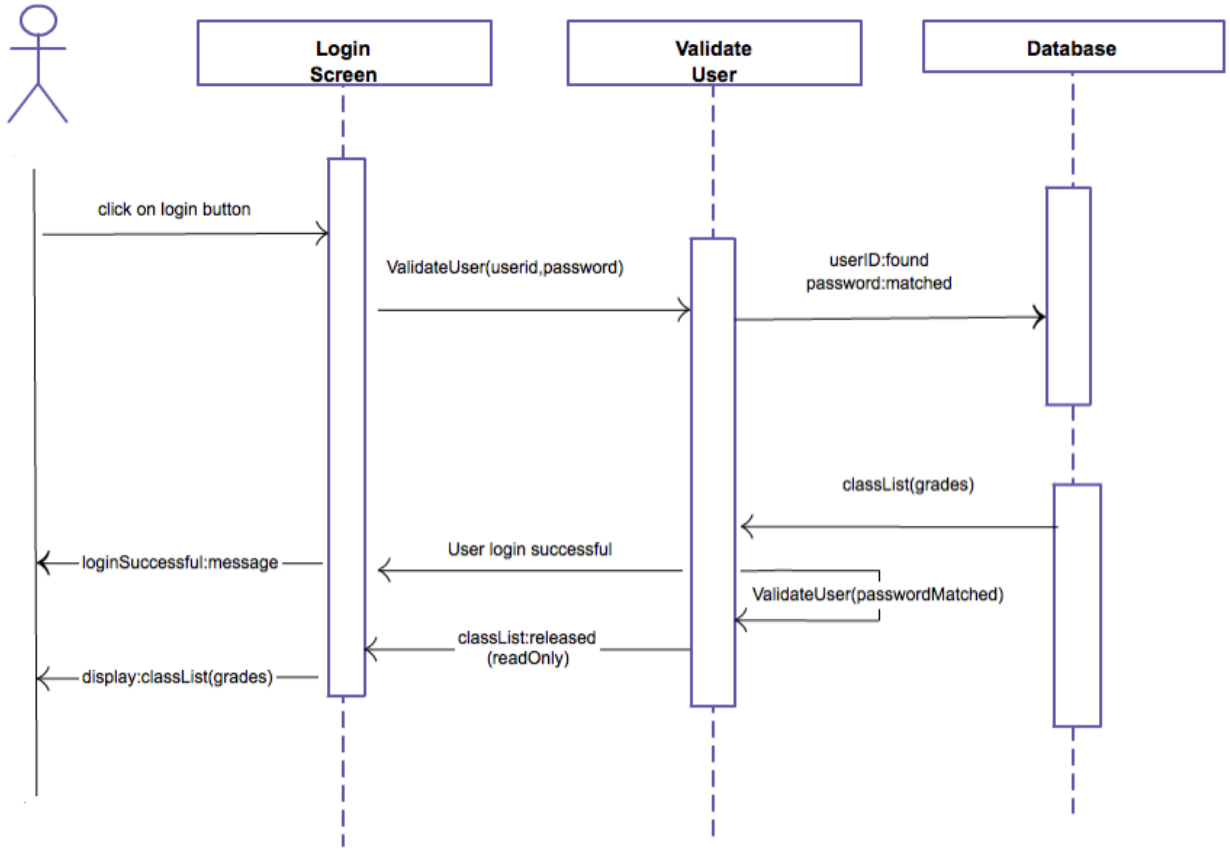
REQ25	4									x	x			
REQ26	3									x	x			
REQ27	5					x				x				
REQ28	3													
REQ29	3													
REQ30	2							x						
REQ31	1							x						
Max PW		5	5	5	3	5	5	2	5	5	5	5	3	5
Total PW		19	16	8	9	9	8	3	30	29	24	11	14	14

System Sequence Diagrams

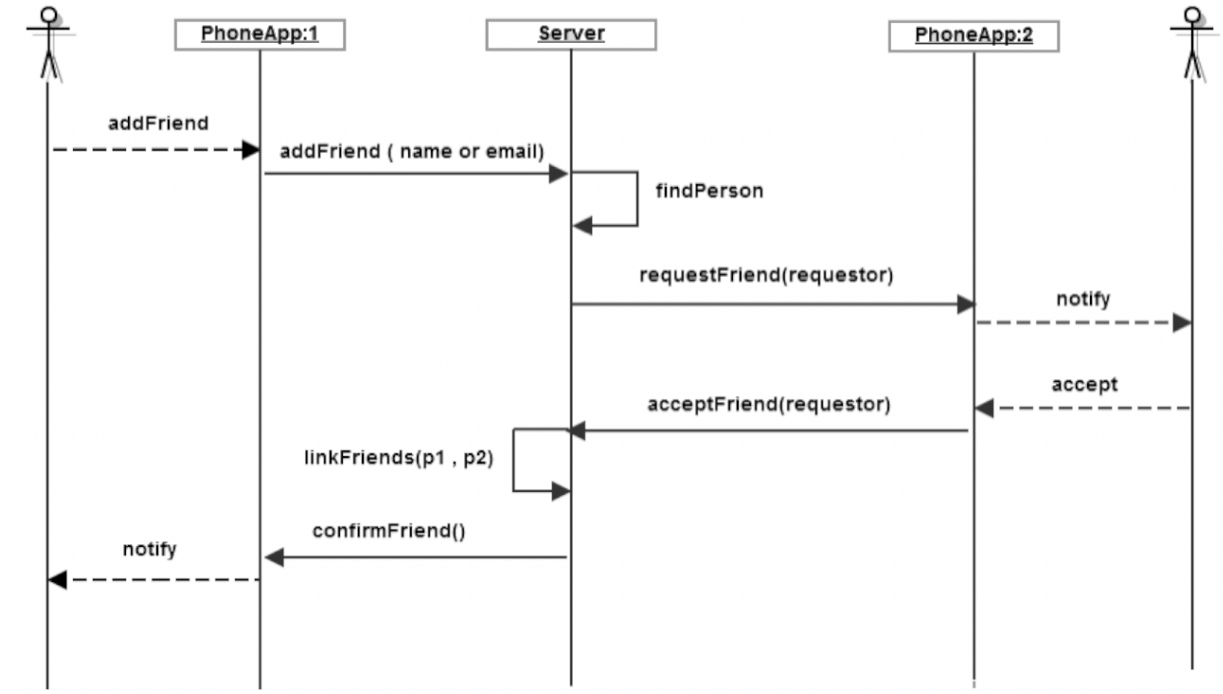
Sequence diagram of registration



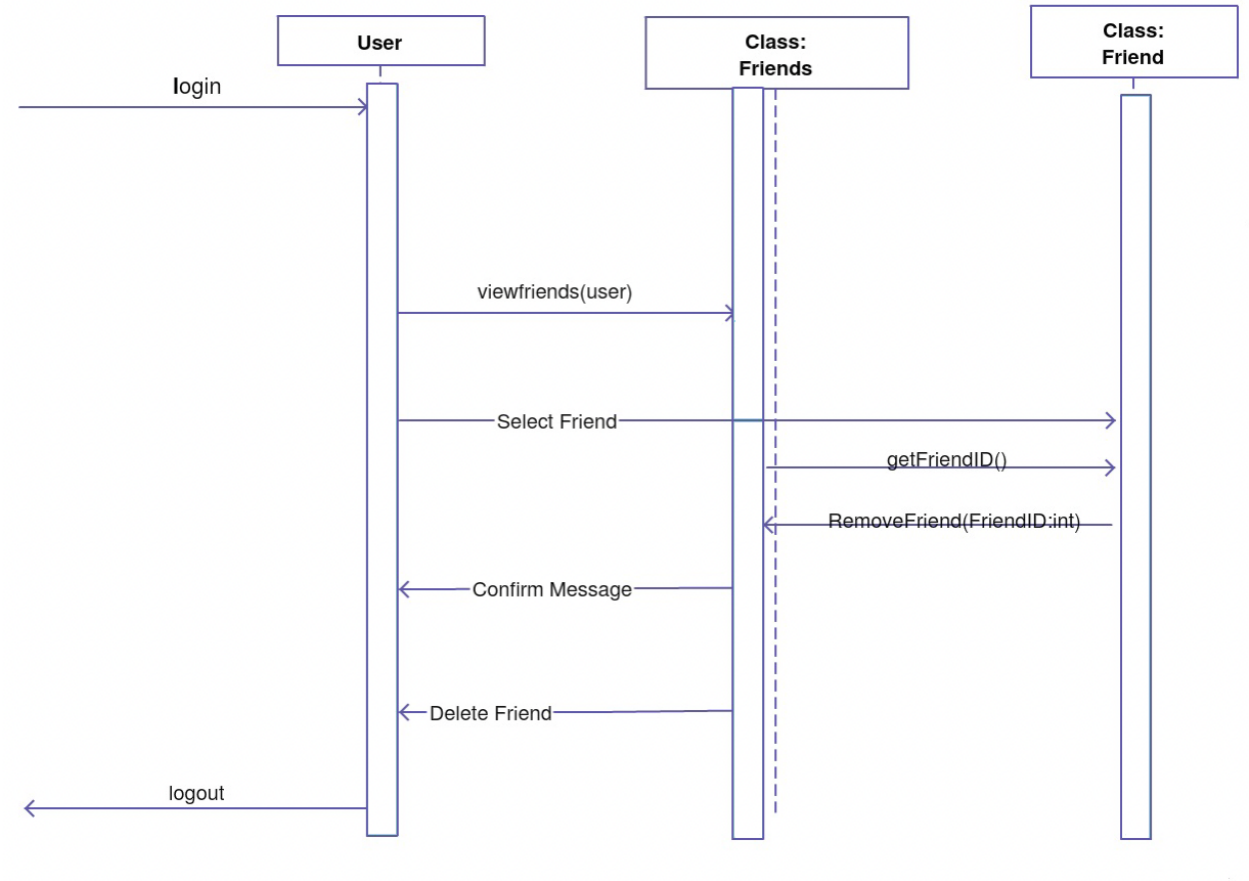
Sequence diagram of log in



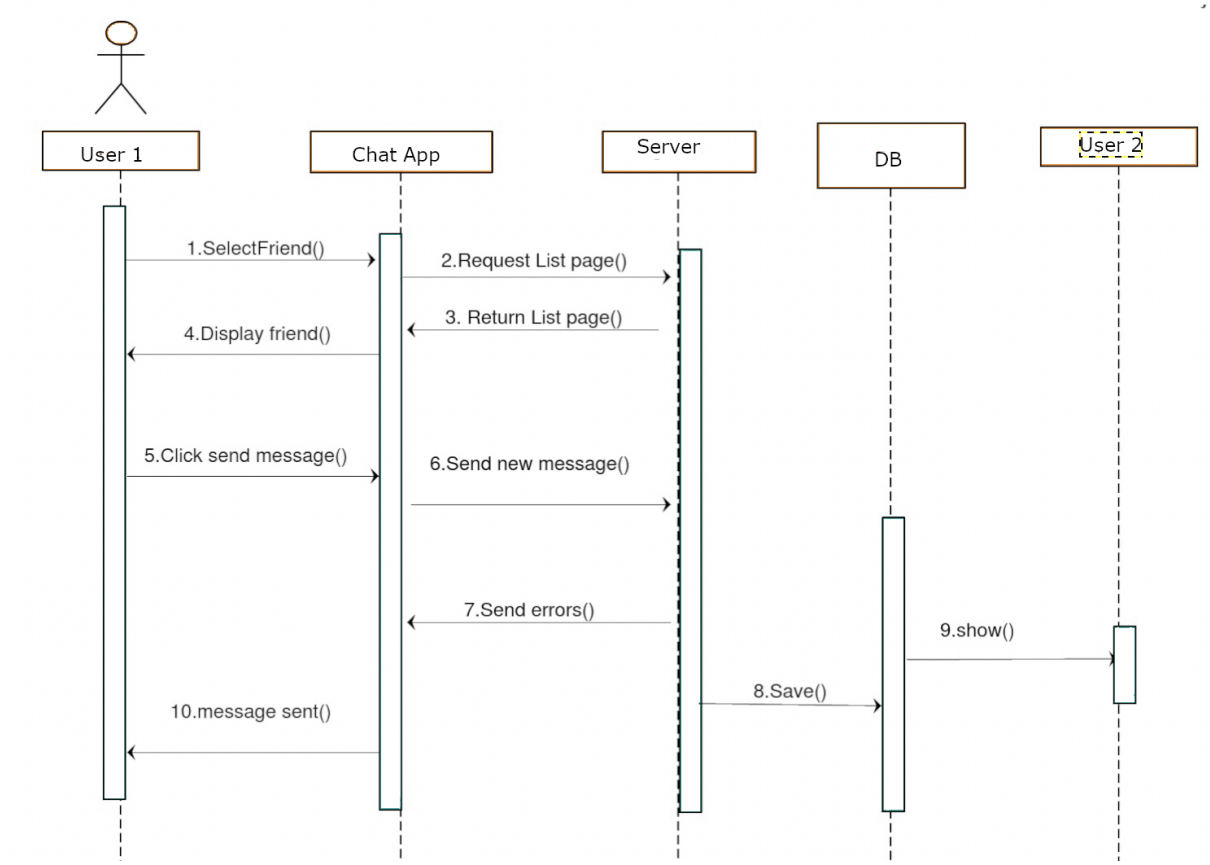
Sequence diagram of Adding Friend



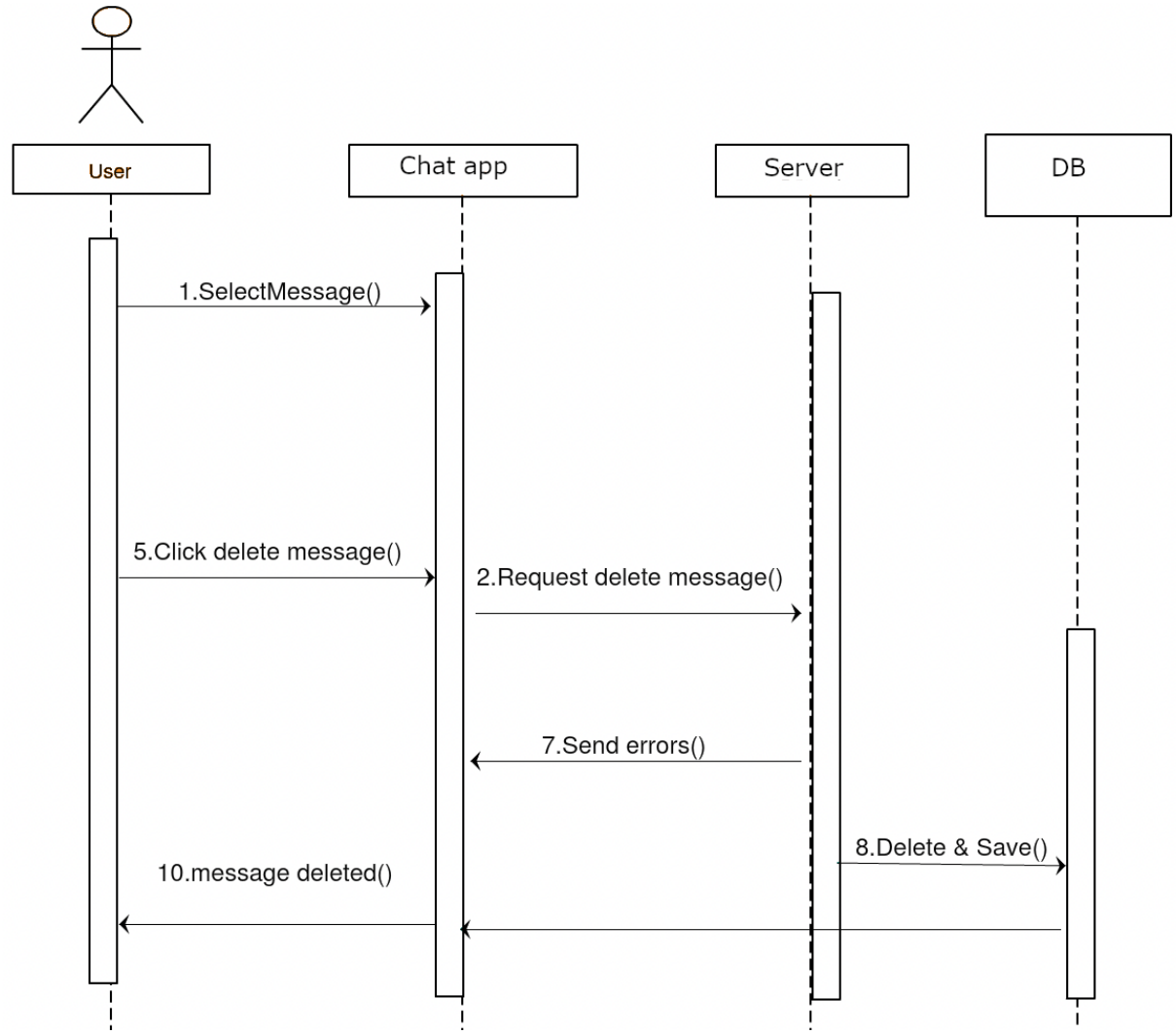
Sequence diagram of removing Friend



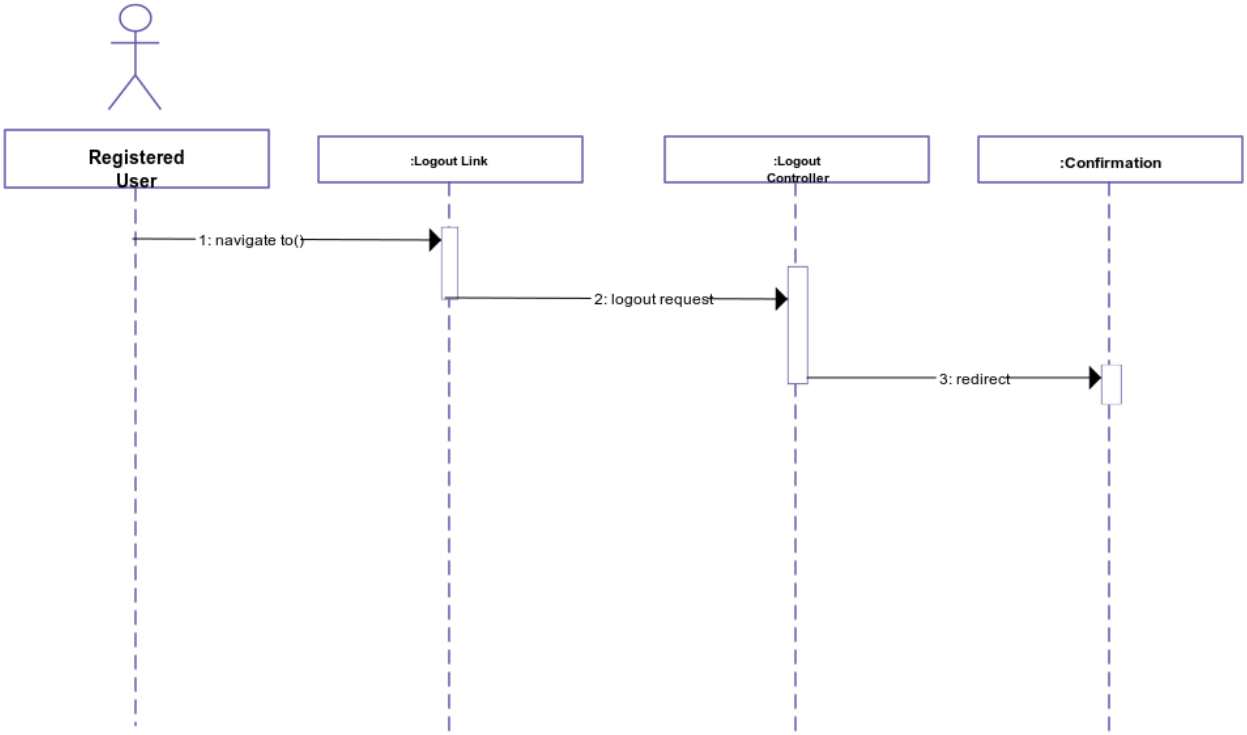
Sequence diagram for sending messages



Sequence diagram of deleting message



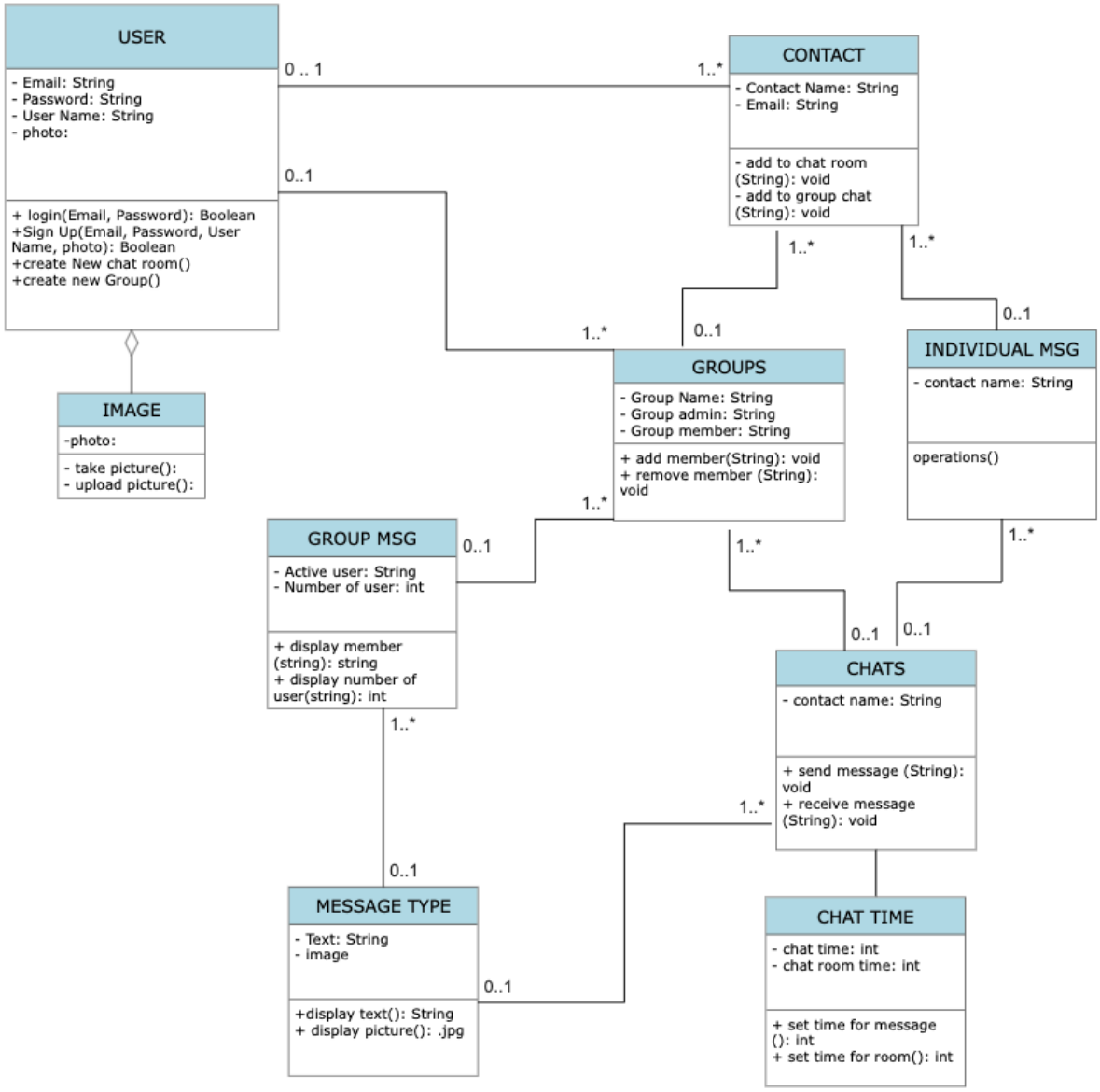
Sequence diagram of log out



Effort Estimation Using Use Case Points

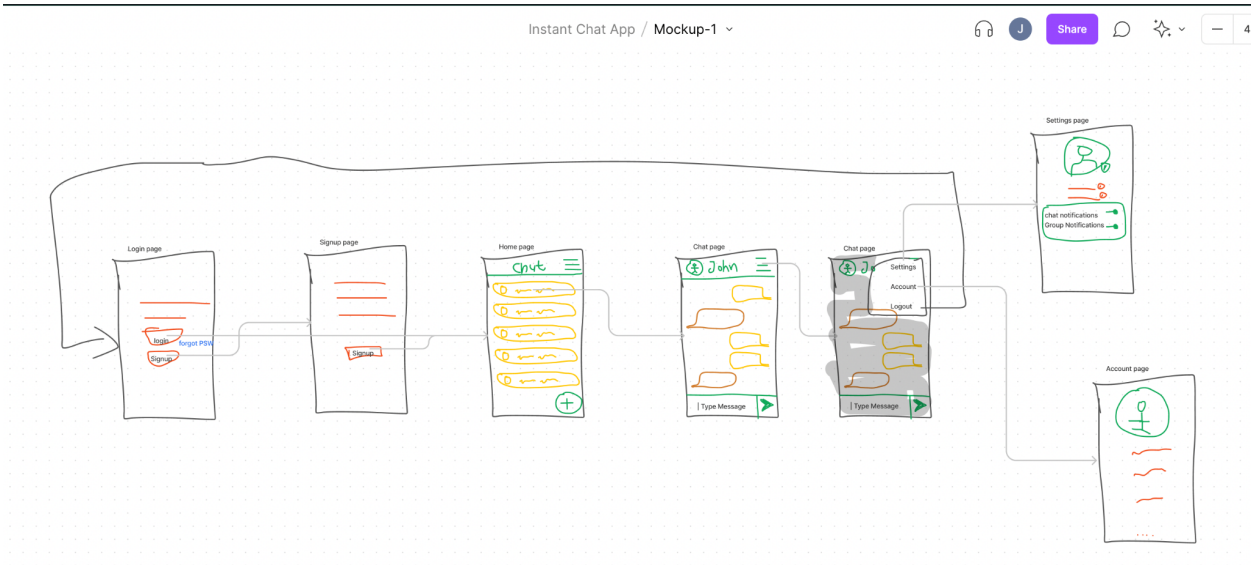
Use Cases	Points
UC1	19
UC2	16
UC3	8
UC4	9
UC5	9
UC6	8
UC7	3
UC8	30
UC9	29
UC10	24
UC11	11
UC12	14
UC13	14
Total	**194**

Class Diagrams and Interface Specifications



User Interface Design and Implementation

Mockup-1



User Interface and Code Snippets

We have designed our application in a way that all screens are navigated by App.js class. This class helps to navigate between all pages and screens in our app. In what follow, you have access to **App.js** code:

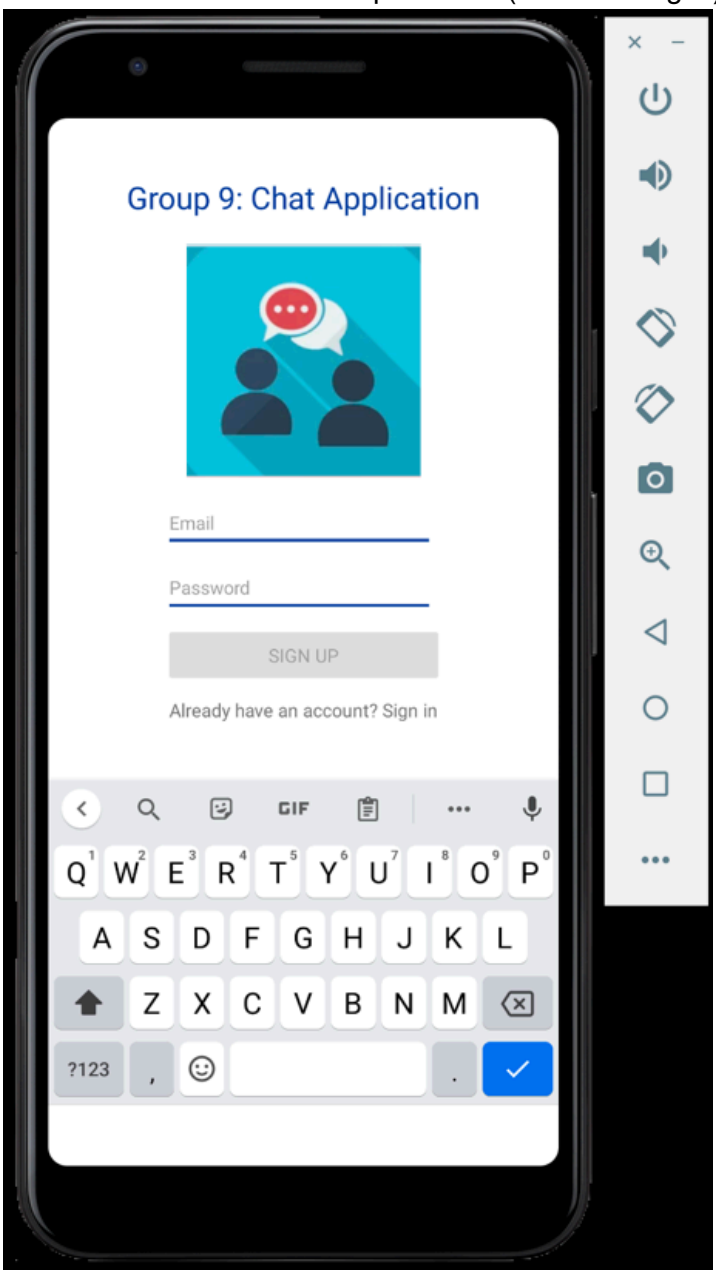
```
import React, { useState, useEffect, useContext } from "react";
import { Text, View, LogBox } from "react-native";
import { useAssets } from "expo-asset";
import { onAuthStateChanged } from "firebase/auth";
import { auth } from "./firebase";
import { NavigationContainer } from "@react-navigation/native";
import { createStackNavigator } from "@react-navigation/stack";
import { createMaterialTopTabNavigator } from "@react-navigation/material-top-tabs";
import SignIn from "./screens/SignIn";
import ContextWrapper from "./context/ContextWrapper";
import Context from "./context/Context";
import Profile from "./screens/Profile";
import Chats from "./screens/Chats";
import Photo from "./screens/Photo";
import { Ionicons } from "@expo/vector-icons";
import Contacts from "./screens/Contacts";
import Chat from './screens/Chat'
import ChatHeader from './components/ChatHeader'
LogBox.ignoreLogs([
  "Setting a timer",
  "AsyncStorage has been extracted from react-native core and will be removed in a future release.",
]);

const Stack = createStackNavigator();
const Tab = createMaterialTopTabNavigator();

function App() {
  const [currUser, setCurrUser] = useState(null);
  const [loading, setLoading] = useState(true);
  const {
    theme: { colors },
  } = useContext(Context);

  useEffect(() => {
    const unsubscribe = onAuthStateChanged(auth, (user) => {
      setLoading(false);
      if (user) {
        setCurrUser(user);
      }
    });
    return () => unsubscribe();
  }, []);

  if (loading) {
    return <Text>Loading...</Text>;
  }

  return (
    <NavigationContainer>
      {!currUser ? (
        <Stack.Navigator screenOptions={{ headerShown: false }}>
          <Stack.Screen name="signIn" component={SignIn} />
        </Stack.Navigator>
      ) : (
        <Stack.Navigator
          screenOptions={{
```

```
            headerStyle: {
              backgroundColor: colors.foreground,
              shadowOpacity: 0,
              elevation: 0,
            },
            headerTintColor: colors.white,
          }}
        >
          {!currUser.displayName && (
            <Stack.Screen
              name="profile"
              component={Profile}
              options={{ headerShown: false }}
            />
          )}
          <Stack.Screen
            name="home"
            options={{ title: "Chat App" }}
            component={Home}
          />
          <Stack.Screen
            name="contacts"
            options={{ title: "Select Contacts" }}
            component={Contacts}
          />
          <Stack.Screen name="chat" component={Chat} options={{headerTitle: (props) => <ChatHeader {...prop
        </Stack.Navigator>
      )}
    </NavigationContainer>
  );
}
function Home() {
  const {
    theme: { colors },
  } = useContext(Context);
  return (
    <Tab.Navigator
      screenOptions={({ route }) => {
        return {
          tabBarLabel: () => {
            if (route.name === "photo") {
              return <Ionicons name="camera" size={20} color={colors.white} />;
            } else {
              return (
                <Text style={{ color: colors.white }}>
                  {route.name.toLocaleUpperCase()}
                </Text>
              );
            }
          },
          tabBarShowIcon: true,
          tabBarLabelStyle: {
            color: colors.white,
          },
          tabBarIndicatorStyle: {
            backgroundColor: colors.white,
```

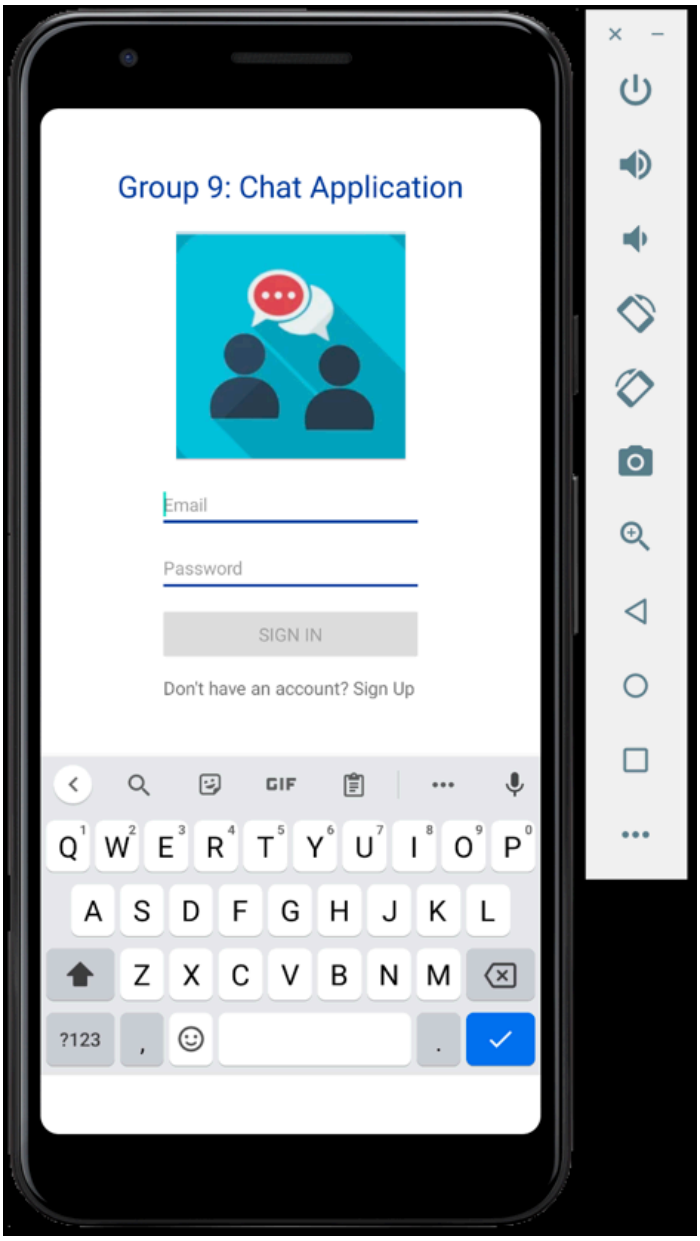
```
            },
            tabBarStyle: {
              backgroundColor: colors.foreground,
            },
          };
        }}
        initialRouteName="chats"
      >
        <Tab.Screen name="photo" component={Photo} />
        <Tab.Screen name="chats" component={Chats} />
      </Tab.Navigator>
    );
  }

  function Main() {
    const [assets] = useAssets(
      require("./assets/icon-square.png"),
      require("./assets/chatbg.png"),
      require("./assets/user-icon.png"),
      require("./assets/welcome1-img.png")
    );
    if (!assets) {
      return <Text>Loading ..</Text>;
    }
    return (
      <ContextWrapper>
        <App />
      </ContextWrapper>
    );
  }

  export default Main;
```

Welcome page:

the start screen on our app is the sign in/sign up page. As you can see on the following picture, users can sign up with a valid email address and password (at least 6 digits).



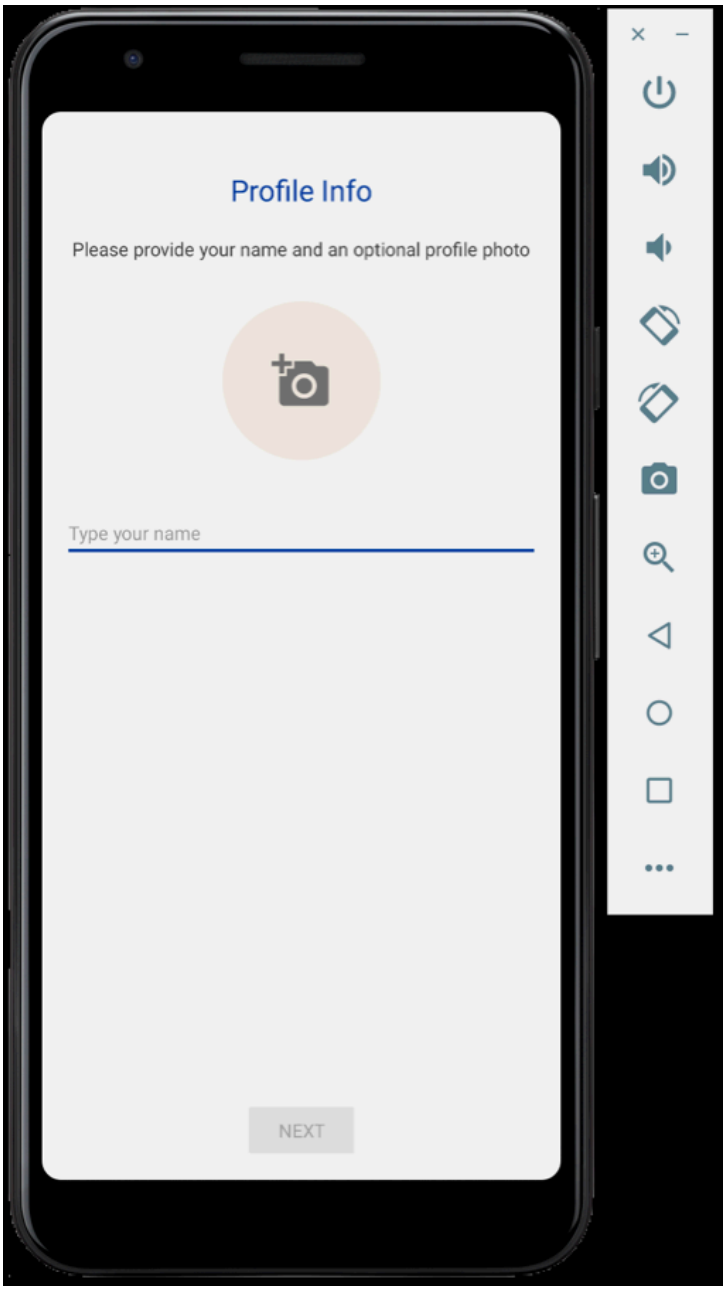
If User already has an account in the app, by clicking on Sign in button, can access to **sign in page:**



If the user puts in an invalid email or password, a notification pops up and the user must provide valid information to sign in or sign up.

Sign up:

First, user must put email and password and click on sign up button to go through to the next page to upload profile picture and set a username:



Snippet: let see the code related to the sign-in and sign-up feature in our app:

SignIn.js

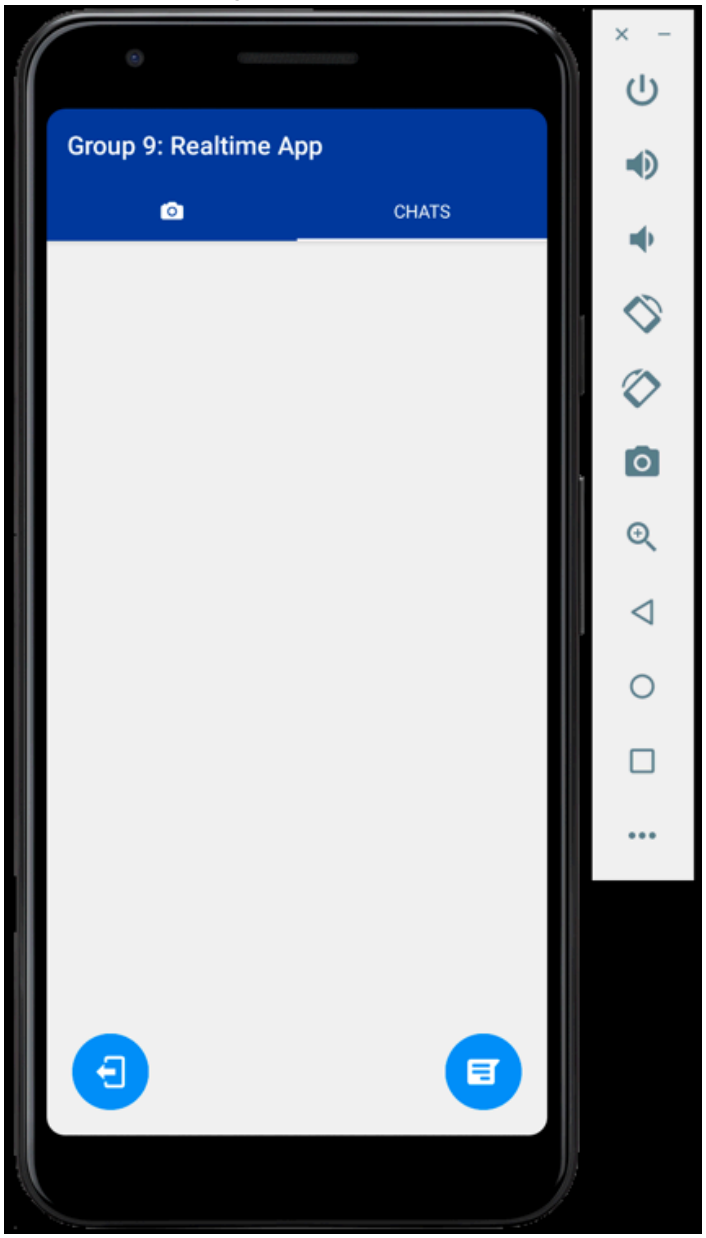
```
import React, { useContext, useState } from "react";
import { View, Text, Image, TextInput, Button } from "react-native";
import { TouchableOpacity } from "react-native-gesture-handler";
import Context from "../context/Context";
import { signIn, signUp } from "../firebase";
export default function SignIn() {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [mode, setMode] = useState("signUp");
  const {
    theme: { colors },
  } = useContext(Context);

  async function handlePress() {
    if (mode === "signUp") {
      await signUp(email, password);
    }
    if (mode === "signIn") {
      await signIn(email, password);
    }
  }
```

```
  return (
    <View
      style={{
        justifyContent: "center",
        alignItems: "center",
        flex: 1,
        backgroundColor: colors.white,
      }}
    >
      <Text
        style={{ color: colors.foreground, fontSize: 24, marginBottom: 20 }}
      >
        Group 9: Chat Application
      </Text>
      <Image
        source={require("../assets/welcome1-img.png")}
        style={{ width: 180, height: 180 }}
        resizeMode="cover"
      />
      <View style={{ marginTop: 20 }}>
        <TextInput
          placeholder="Email"
          value={email}
          onChangeText={setEmail}
          style={{
            borderBottomColor: colors.primary,
            borderBottomWidth: 2,
            width: 200,
          }}
        />
        <TextInput
          placeholder="Password"
          value={password}
          onChangeText={setPassword}
          secureTextEntry={true}
          style={{
            borderBottomColor: colors.primary,
            borderBottomWidth: 2,
            width: 200,
            marginTop: 20,
          }}
        />
        <View style={{ marginTop: 20 }}>
          <Button
            title={mode === "signUp" ? "Sign Up" : "Sign in"}
            disabled={!password || !email}
            color={colors.secondary}
            onPress={handlePress}
          />
        </View>
        <TouchableOpacity
          style={{ marginTop: 15 }}
          onPress={() =>
            mode === "signUp" ? setMode("signIn") : setMode("signUp")
          }
        >
```

```
78              <Text style={{ color: colors.secondaryText }}>
79                {mode === "signUp"
80                  ? "Already have an account? Sign in"
81                  : "Don't have an account? Sign Up"}
82              </Text>
83            </TouchableOpacity>
84          </View>
85        </View>
86      );
87    }
```

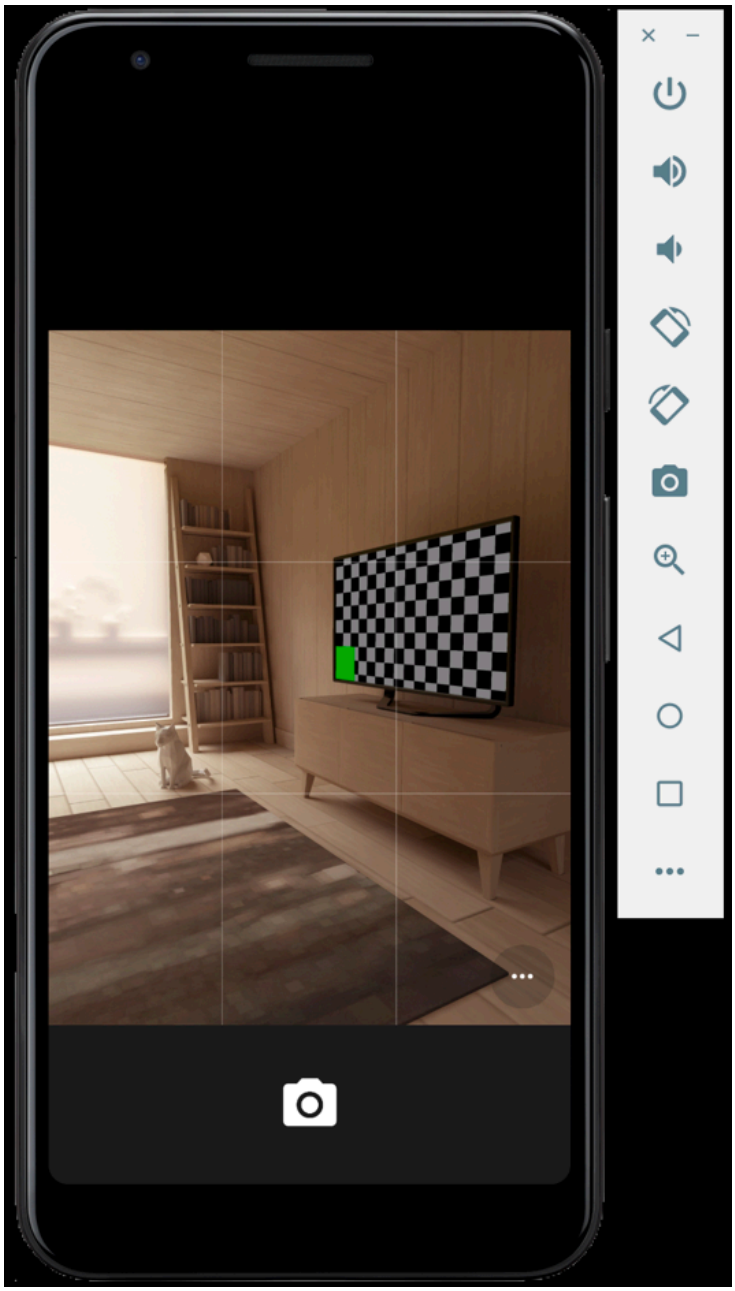
Home page:

after signing into the app, the home page or chat app shows that has different parts:



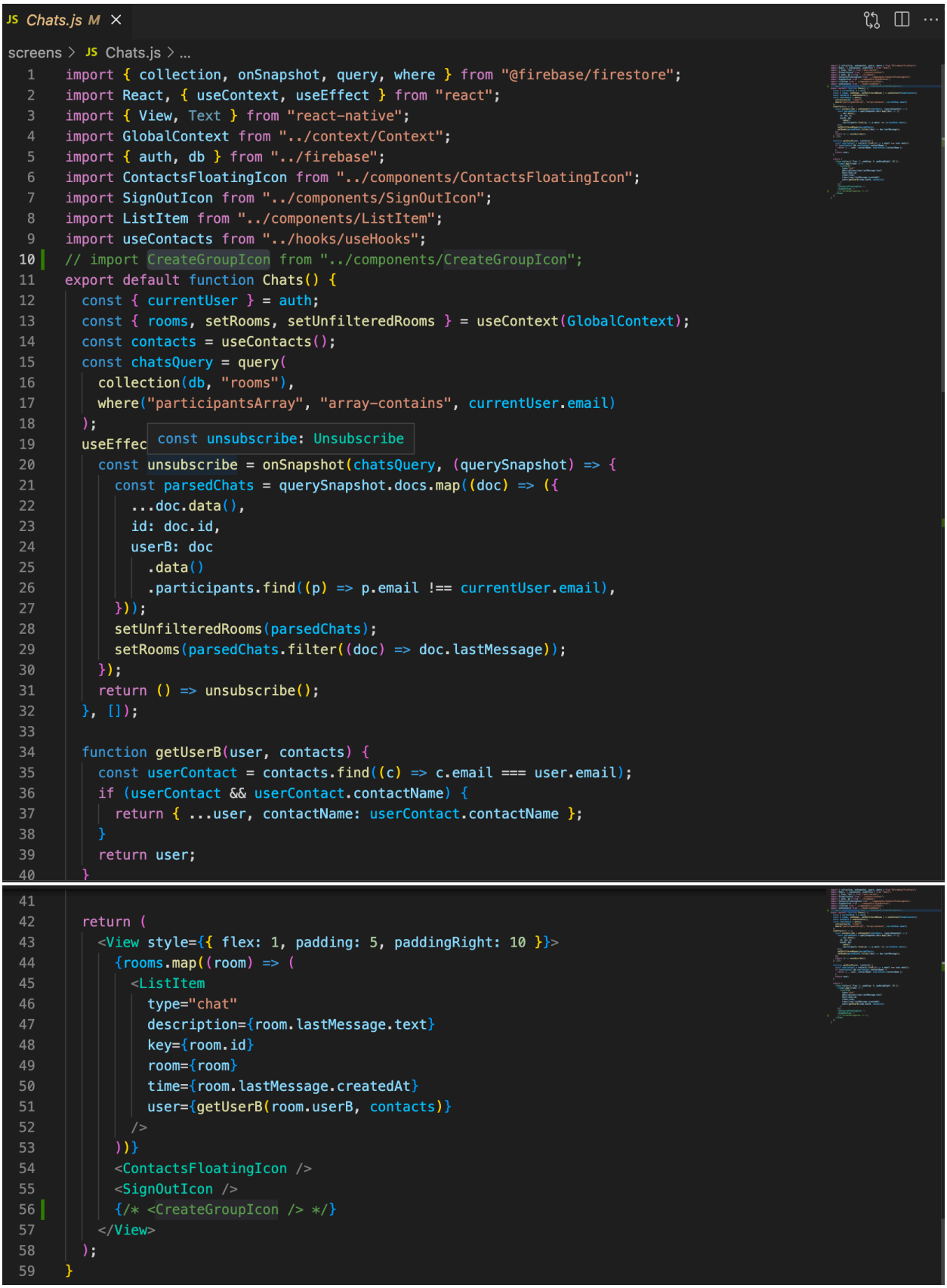
This page has two major parts: image and Chats

One image, first you must take a picture with your device and send it to the user that you already saved his/her information in contacts:



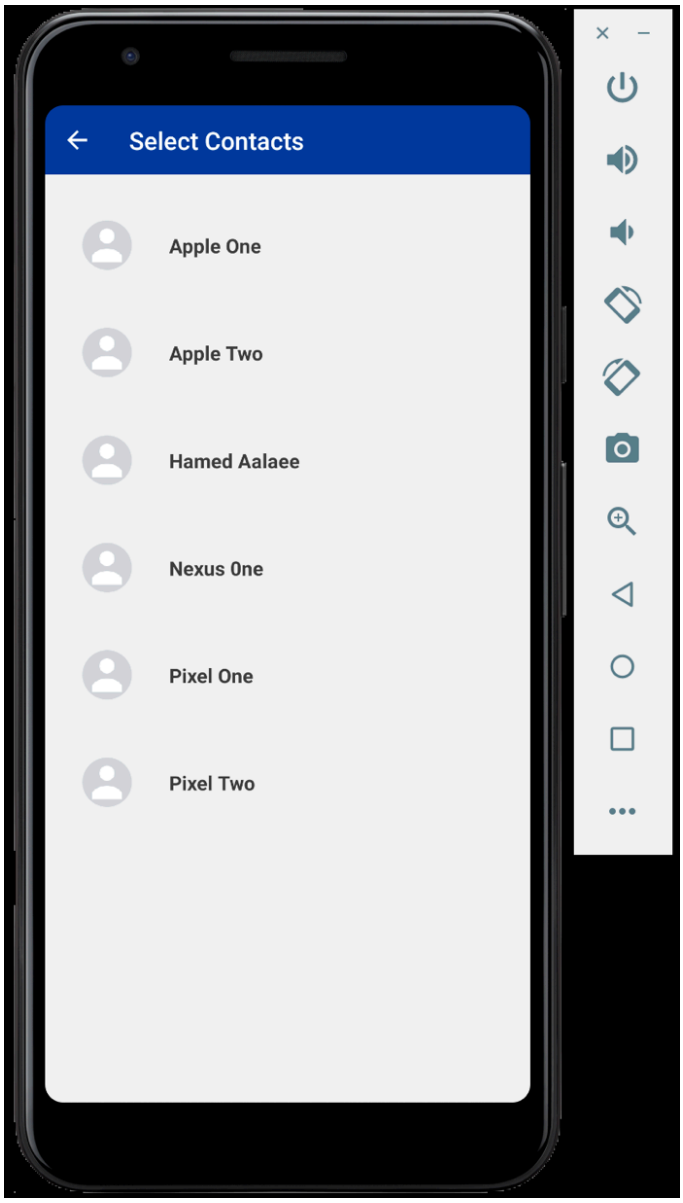
Chats page:

This page has three different components:
There is a class that handle all these three features, chats.js


Chats.js M
screens > Chats.js > ...
1 import { collection, onSnapshot, query, where } from "@firebase/firestore";
2 import React, { useContext, useEffect } from "react";
3 import { View, Text } from "react-native";
4 import GlobalContext from "../context/Context";
5 import { auth, db } from "../firebase";
6 import ContactsFloatingIcon from "../components/ContactsFloatingIcon";
7 import SignOutIcon from "../components/SignOutIcon";
8 import ListItem from "../components/ListItem";
9 import useContacts from "../hooks/useHooks";
10 // import CreateGroupIcon from "../components/CreateGroupIcon";
11 export default function Chats() {
12 const { currentUser } = auth;
13 const { rooms, setRooms, setUnfilteredRooms } = useContext(GlobalContext);
14 const contacts = useContacts();
15 const chatsQuery = query(
16 collection(db, "rooms"),
17 where("participantsArray", "array-contains", currentUser.email)
18);
19 useEffec const unsubscribe: Unsubscribe
20 const unsubscribe = onSnapshot(chatsQuery, (querySnapshot) => {
21 const parsedChats = querySnapshot.docs.map((doc) => ({
22 ...doc.data(),
23 id: doc.id,
24 userB: doc
25 .data()
26 .participants.find((p) => p.email !== currentUser.email),
27 }));
28 setUnfilteredRooms(parsedChats);
29 setRooms(parsedChats.filter((doc) => doc.lastMessage));
30 });
31 return () => unsubscribe();
32 }, []);
33
34 function getUserB(user, contacts) {
35 const userContact = contacts.find((c) => c.email === user.email);
36 if (userContact && userContact.contactName) {
37 return { ...user, contactName: userContact.contactName };
38 }
39 return user;
40 }
41
42 return (
43 <View style={{ flex: 1, padding: 5, paddingRight: 10 }}>
44 {rooms.map((room) => (
45 <ListItem
46 type="chat"
47 description={room.lastMessage.text}
48 key={room.id}
49 room={room}
50 time={room.lastMessage.createdAt}
51 user={getUserB(room.userB, contacts)}
52 />
53))}
54 <ContactsFloatingIcon />
55 <SignOutIcon />
56 {/* <CreateGroupIcon /> */}
57 </View>
58);
59 }

Contact:

to start new conversation with another user
The contact class in **contact.js** and **contacsFloatingIcon.js**


Select Contacts
Apple One
Apple Two
Hamed Aalaee
Nexus 0ne
Pixel One
Pixel Two

Contacts.js

screens > Contacts.js > ...

```
import { collection, onSnapshot, query, where } from "@firebase/firestore";
import { useRoute } from "@react-navigation/native";
import React, { useContext, useEffect, useState } from "react";
import { View, Text, FlatList } from "react-native";
import ListItem from "../components/ListItem";
import GlobalContext from "../context/Context";
import { db } from "../firebase";
import useContacts from "../hooks/useHooks";

export default function Contacts() {
  const contacts = useContacts();
  const route = useRoute();
  const image = route.params && route.params.image;
  return (
    <FlatList
      style={{ flex: 1, padding: 10 }}
      data={contacts}
      keyExtractor={(_, i) => i}
      renderItem={({ item }) => <ContactPreview contact={item} image={image} />}
    />
  );
}

function ContactPreview({ contact, image }) {
  const { unfilteredRooms, rooms } = useContext(GlobalContext);
  const [user, setUser] = useState(contact);

  useEffect(() => {
    const q = query(
      collection(db, "users"),
      where("email", "==", contact.email)
    );
    const unsubscribe = onSnapshot(q, (snapshot) => {
      if (snapshot.docs.length) {
        const userDoc = snapshot.docs[0].data();
        setUser((prevUser) => ({ ...prevUser, userDoc }));
      }
    });
    return () => unsubscribe();
  }, []);
  return (
    <ListItem
      style={{ marginTop: 7 }}
      type="contacts"
      user={user}
      image={image}
      room={unfilteredRooms.find((room) =>
        room.participantsArray.includes(contact.email)
      )}
    />
  );
}
```

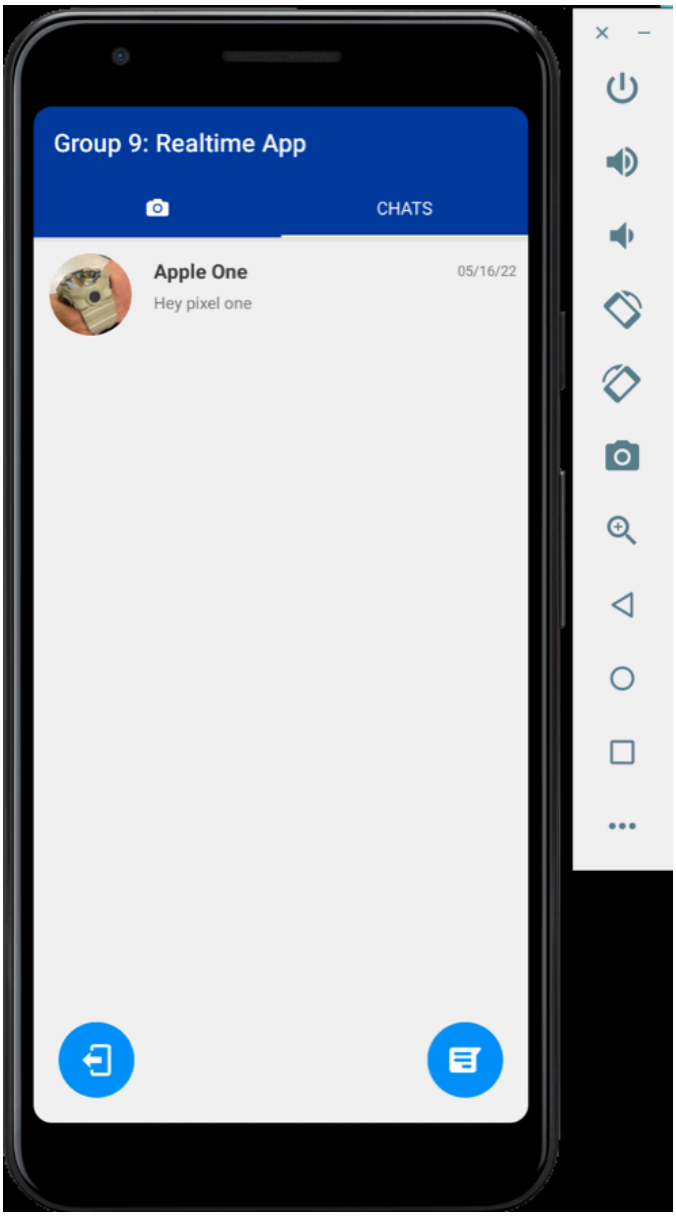
JS ContactsFloatingIcon.js ×

components > JS ContactsFloatingIcon.js > ContactsFloatingIcon

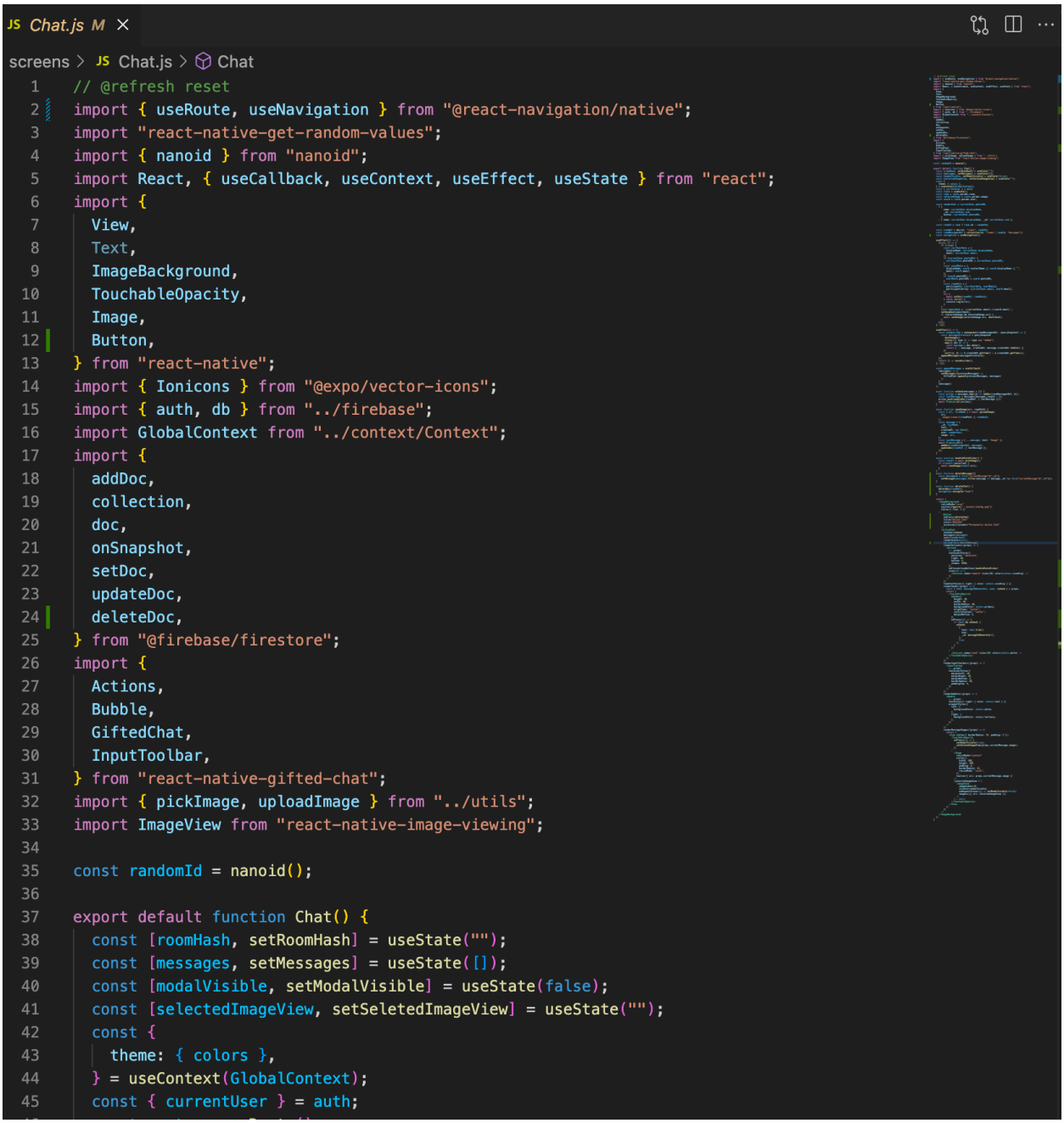
```
1   import React, { useContext } from "react";
2   import { TouchableOpacity } from "react-native";
3   import { MaterialCommunityIcons } from "@expo/vector-icons";
4   import GlobalContext from "../context/Context";
5   import { useNavigation } from "@react-navigation/native";
6   export default function ContactsFloatingIcon() {
7     const {
8       theme: { colors },
9     } = useContext(GlobalContext);
10    const navigation = useNavigation();
11    return (
12      <TouchableOpacity
13        onPress={() => navigation.navigate("contacts")}
14        style={{
15          position: "absolute",
16          right: 20,
17          bottom: 20,
18          borderRadius: 60,
19          width: 60,
20          height: 60,
21          backgroundColor: colors.secondary,
22          alignItems: "center",
23          justifyContent: "center",
24        }}
25      >
26        <MaterialCommunityIcons
27          name="android-messages"
28          size={30}
29          color="white"
30          style={{ transform: [{ scaleX: -1 }] }}
31        />
32      </TouchableOpacity>
33    );
34  }
35
```

Chatrooms:

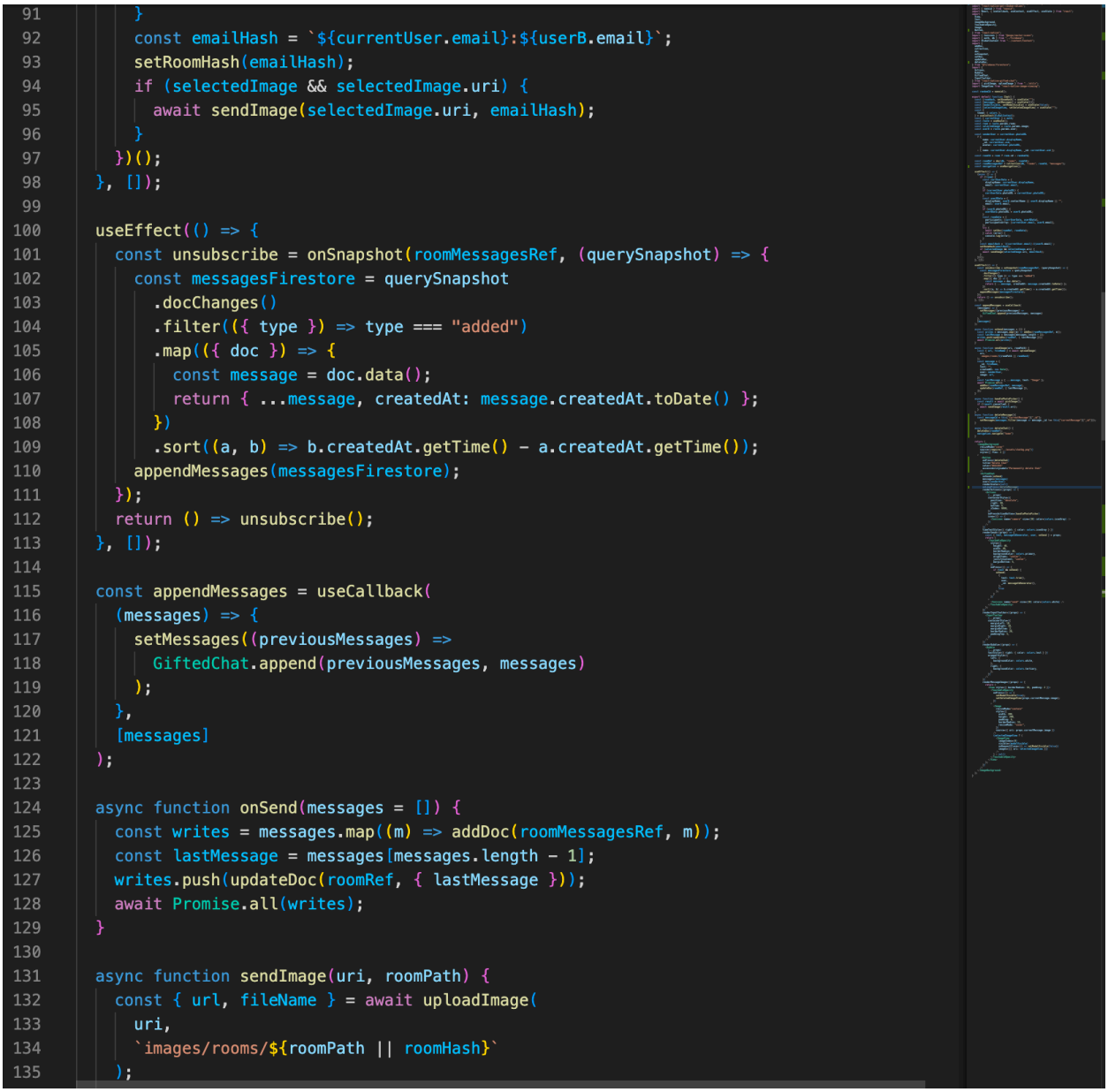
to show previous conversation



The snippet related to chat feature in in **chat.js**:


Chat.js M
screens > Chat.js > Chat
// @refresh reset
import { useRoute, useNavigation } from "@react-navigation/native";
import "react-native-get-random-values";
import { nanoid } from "nanoid";
import React, { useCallback, useContext, useEffect, useState } from "react";
import {
View,
Text,
ImageBackground,
TouchableOpacity,
Image,
Button,
} from "react-native";
import { Ionicons } from "@expo/vector-icons";
import { auth, db } from "../firebase";
import GlobalContext from "../context/Context";
import {
addDoc,
collection,
doc,
onSnapshot,
setDoc,
updateDoc,
deleteDoc,
} from "@firebase/firestore";
import {
Actions,
Bubble,
GiftedChat,
InputToolbar,
} from "react-native-gifted-chat";
import { pickImage, uploadImage } from "../utils";
import ImageView from "react-native-image-viewing";
const randomId = nanoid();
export default function Chat() {
const [roomHash, setRoomHash] = useState("");
const [messages, setMessages] = useState([]);
const [modalVisible, setModalVisible] = useState(false);
const [selectedImageView, setSeletedImageView] = useState("");
const {
theme: { colors },
} = useContext(GlobalContext);
const { currentUser } = auth;

```
const route = useRoute();
const room = route.params.room;
const selectedImage = route.params.image;
const userB = route.params.user;

const senderUser = currentUser.photoURL
  ? {
      name: currentUser.displayName,
      _id: currentUser.uid,
      avatar: currentUser.photoURL,
    }
  : { name: currentUser.displayName, _id: currentUser.uid };

const roomId = room ? room.id : randomId;

const roomRef = doc(db, "rooms", roomId);
const roomMessagesRef = collection(db, "rooms", roomId, "messages");
const navigation = useNavigation();

useEffect(() => {
  (async () => {
    if (!room) {
      const currUserData = {
        displayName: currentUser.displayName,
        email: currentUser.email,
      };
      if (currentUser.photoURL) {
        currUserData.photoURL = currentUser.photoURL;
      }
      const userBData = {
        displayName: userB.contactName || userB.displayName || "",
        email: userB.email,
      };
      if (userB.photoURL) {
        userBData.photoURL = userB.photoURL;
      }
      const roomData = {
        participants: [currUserData, userBData],
        participantsArray: [currentUser.email, userB.email],
      };
      try {
        await setDoc(roomRef, roomData);
      } catch (error) {
        console.log(error);
      }
```



```
      }
      const emailHash = `${currentUser.email}:${userB.email}`;
      setRoomHash(emailHash);
      if (selectedImage && selectedImage.uri) {
        await sendImage(selectedImage.uri, emailHash);
      }
    })();
  }, []);

  useEffect(() => {
    const unsubscribe = onSnapshot(roomMessagesRef, (querySnapshot) => {
      const messagesFirestore = querySnapshot
        .docChanges()
        .filter(({ type }) => type === "added")
        .map(({ doc }) => {
          const message = doc.data();
          return { ...message, createdAt: message.createdAt.toDate() };
        })
        .sort((a, b) => b.createdAt.getTime() - a.createdAt.getTime());
      appendMessages(messagesFirestore);
    });
    return () => unsubscribe();
  }, []);

  const appendMessages = useCallback(
    (messages) => {
      setMessages((previousMessages) =>
        GiftedChat.append(previousMessages, messages)
      );
    },
    [messages]
  );

  async function onSend(messages = []) {
    const writes = messages.map((m) => addDoc(roomMessagesRef, m));
    const lastMessage = messages[messages.length - 1];
    writes.push(updateDoc(roomRef, { lastMessage }));
    await Promise.all(writes);
  }

  async function sendImage(uri, roomPath) {
    const { url, fileName } = await uploadImage(
      uri,
      `images/rooms/${roomPath || roomHash}`
    );
```

Log-out button:

sign out from the app
After clicking on this button, the user goes back to the sign in button.
The classes related to this feature are **SignOutIcon.js** and line of codes in **chats.js** and **signOut.js** and **signIn.js**:

SignOutIcon.js M

components > SignOutIcon.js > SignOutIcon

```
import React, { useContext } from "react";
import { TouchableOpacity } from "react-native";
import { MaterialCommunityIcons } from "@expo/vector-icons";
import GlobalContext from "../context/Context";
import { useNavigation } from "@react-navigation/native";
import { signOutApp } from "../signIn";

export default function SignOutIcon() {
  const {
    theme: { colors },
  } = useContext(GlobalContext);
  const navigation = useNavigation();
  async function handlePress() {
    await signOutApp();
    console.log("signed out, now trying to navigate to signin page.");
    navigation.navigate("signIn");
  }
  return (
    <TouchableOpacity
      onPress={() => handlePress()}
      style={{
        position: "absolute",
        left: 20,
        bottom: 20,
        borderRadius: 60,
        width: 60,
        height: 60,
        backgroundColor: colors.secondary,
        alignItems: "center",
        justifyContent: "center",
      }}
    >
      <MaterialCommunityIcons
        name="logout"
        size={30}
        color="white"
        style={{ transform: [{ scaleX: -1 }] }}
      />
    </TouchableOpacity>
  );
}

```

signOut.js

signOut.js

```
import { signOut } from 'firebase/auth';
port { auth } from "./firebase";


```

signIn.js

signIn.js > signOutApp

```
1  import { signInWithEmailAndPassword, signOut  } from 'firebase/auth';
2  import { auth } from "./firebase";
3
4
5  export function signIn(email, password) {
6    return signInWithEmailAndPassword(auth, email, password);
7  }
8
9
10 export function signOutApp() {
11   return signOut(auth);
12 }
```

Test Cases

1) Login Form Test Case

- Test-case Identifier: - Use Case Tested: - Pass/fail Criteria: - Input Data:	- TC-1 - UC-1 - API Response with 200 or 400 HTTP response - user credentials (username, email, password)
Test Procedure:	Expected Result:
1. In the login form, type the correct username and password. 2. Type unregistered username and password 3. Type invalid username like “us%^fjf”	1. 1We got 200 HTTP responses with a user token. 2. We got 400 HTTP response with Error message 3. We got 400 HTTP response with Error message

2) Adding a Friend Test Case

- Test-case Identifier: - Use Case Tested: - Pass/fail Criteria: - Input Data:	- TC-2 - UC-3 and UC-5 - API Response with 200 or 400 HTTP response - Username or email
Test Procedure:	Expected Result:
1. Type registered user’s name or email 2. Type unregistered username or email 3. Type invalid username like “us%^fjf”	1. We got 200 HTTP responses with a Success message. 2. We got 400 HTTP response with Error message 3. We got a 404 HTTP response with Error message

3) Sending a message Test Case

- Test-case Identifier: - Use Case Tested: - Pass/fail Criteria: - Input Data:	- TC-5 - UC-5 and UC-8 - API Response with 200 or 400 HTTP response - Token and Message Text or Media
Test Procedure:	Expected Result:
1. Type a valid message text string or select media (image, video) and press send button 2. Types or selects nothing and	1. We got 200 HTTP responses with a success message. 2. We won’t send an API request and display a pop-up message to enter the message before pressing the send button.

press send button 3. Type unsupported emojis or special characters or too much large sized media file	3. We got 400 HTTP response with Error message

4) Deleting a message Test Case

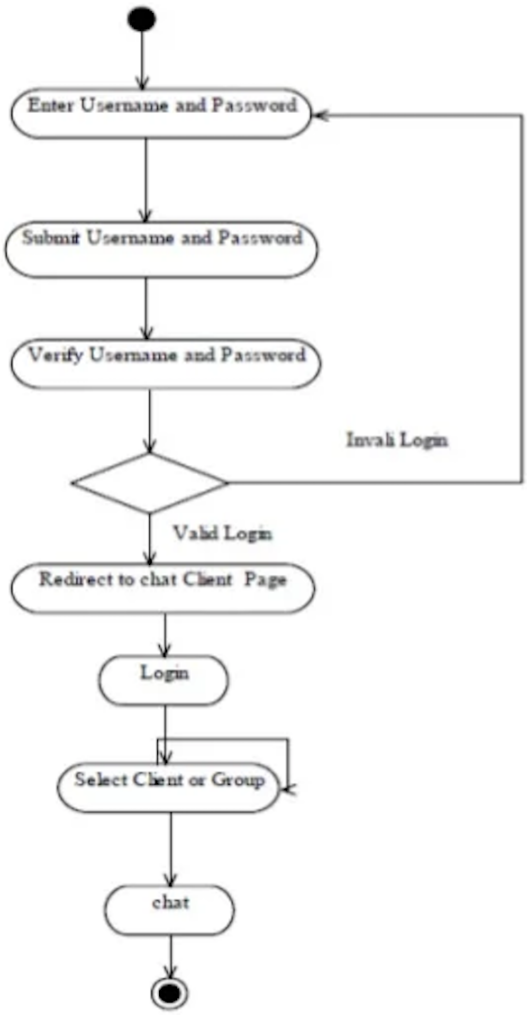
- Test-case Identifier: - Use Case Tested: - Pass/fail Criteria: - Input Data:	- TC-6 - UC-4 and UC-11, UC-12 - API Response with 200 or 400 HTTP response - User Token
Test Procedure:	Expected Result:
1. Sending DELETE request to API with valid User Token 2. Sending DELETE request to API with valid User Token	1. We got 200 HTTP response with a success message. 2. We got 400 HTTP response with Error message

5) Log out Test Case

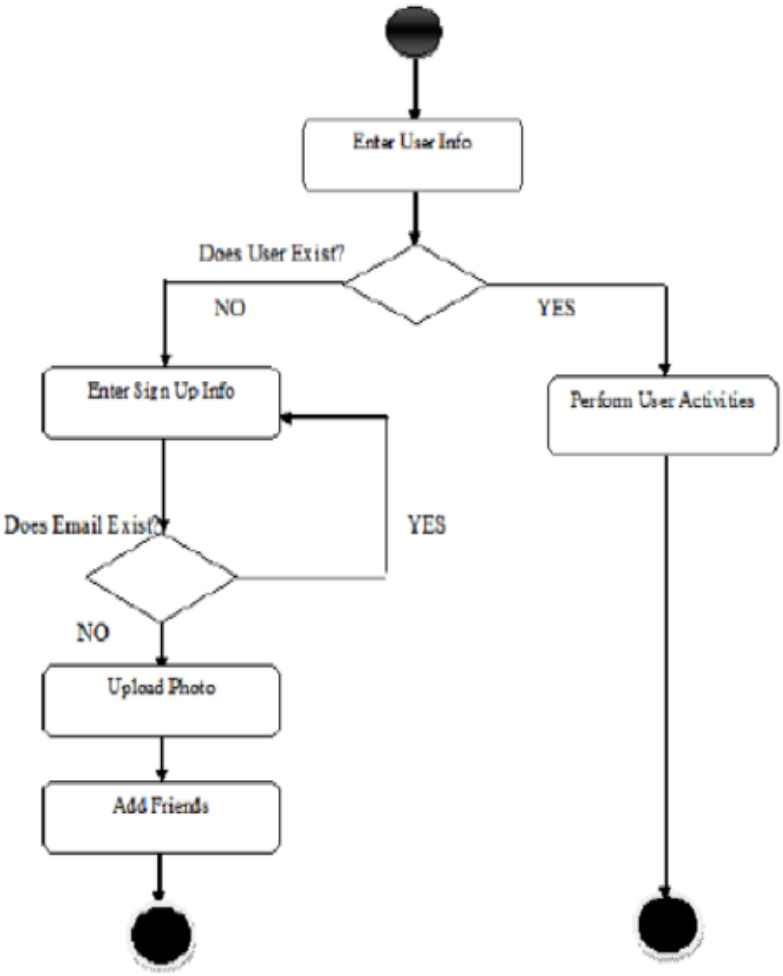
- Test-case Identifier: - Use Case Tested: - Pass/fail Criteria: - Input Data:	- TC-7 - - API Response with 200 or 400 HTTP response - User Token
Test Procedure:	Expected Result:
1. Sending request to API with valid User Token 2. Sending request to API with valid User Token	1. We got 200 HTTP response with a success message. 2. We got 400 HTTP response with Error message

state-based testing

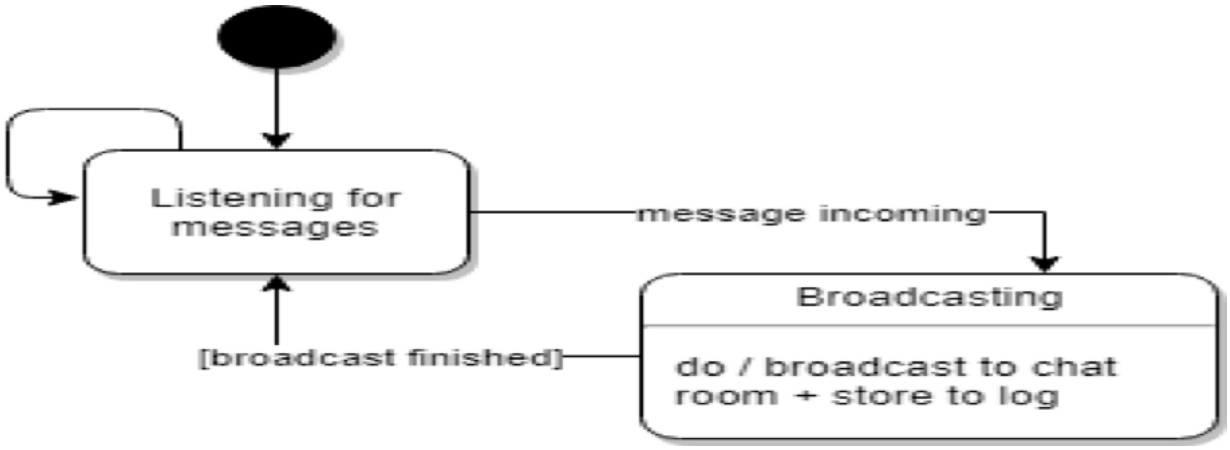
Login



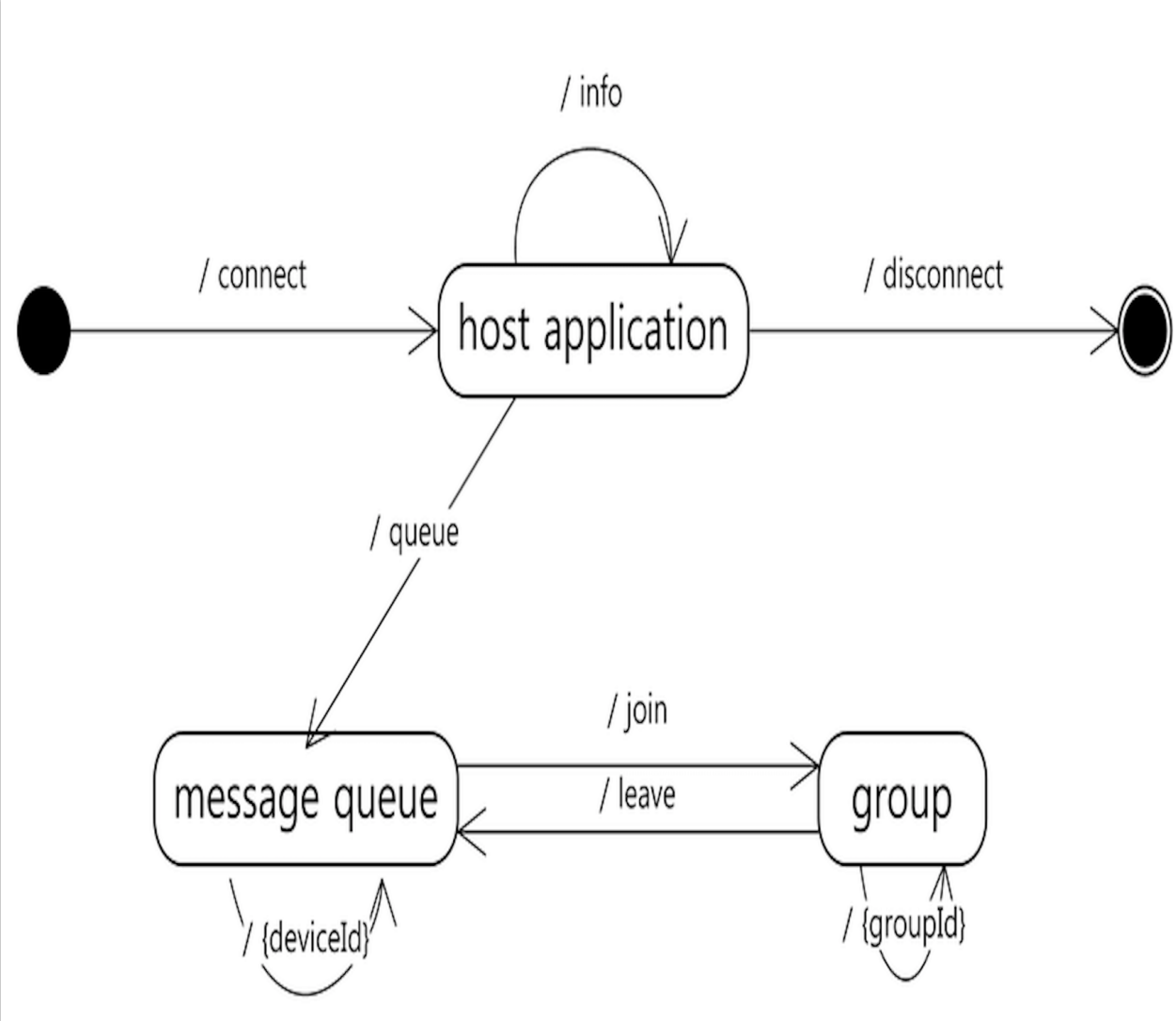
State diagram of adding friends



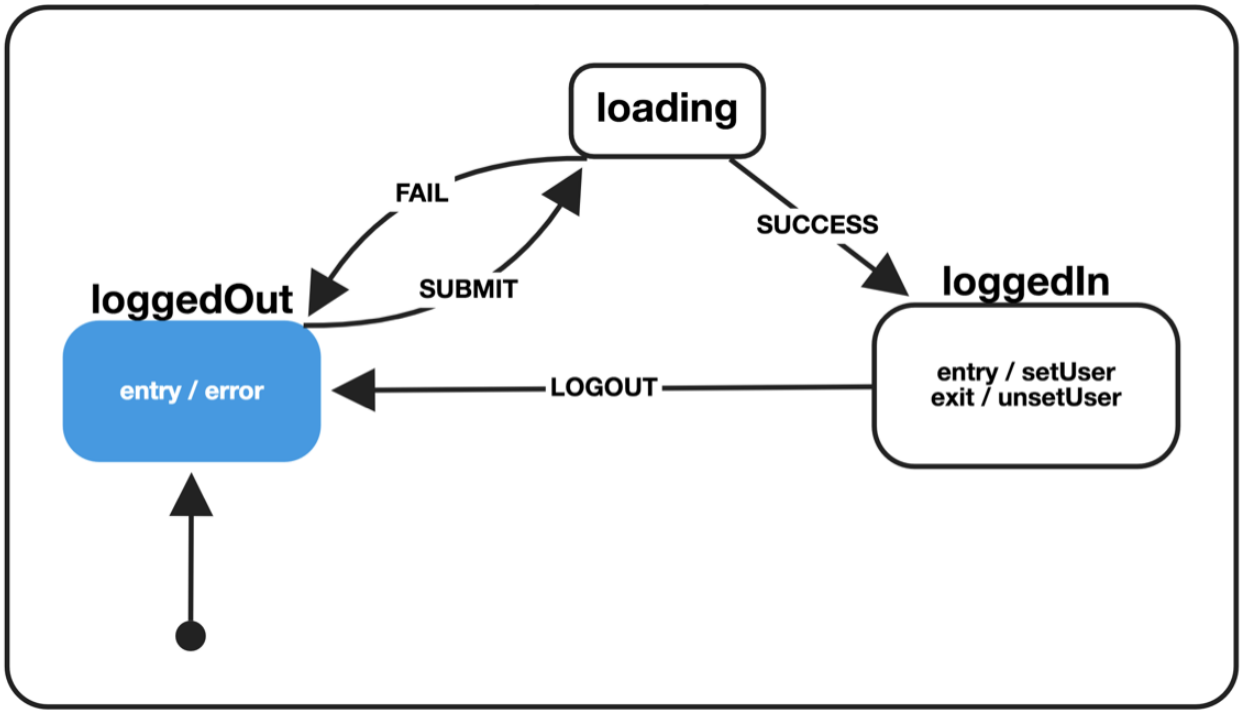
State diagram of broadcasting messages



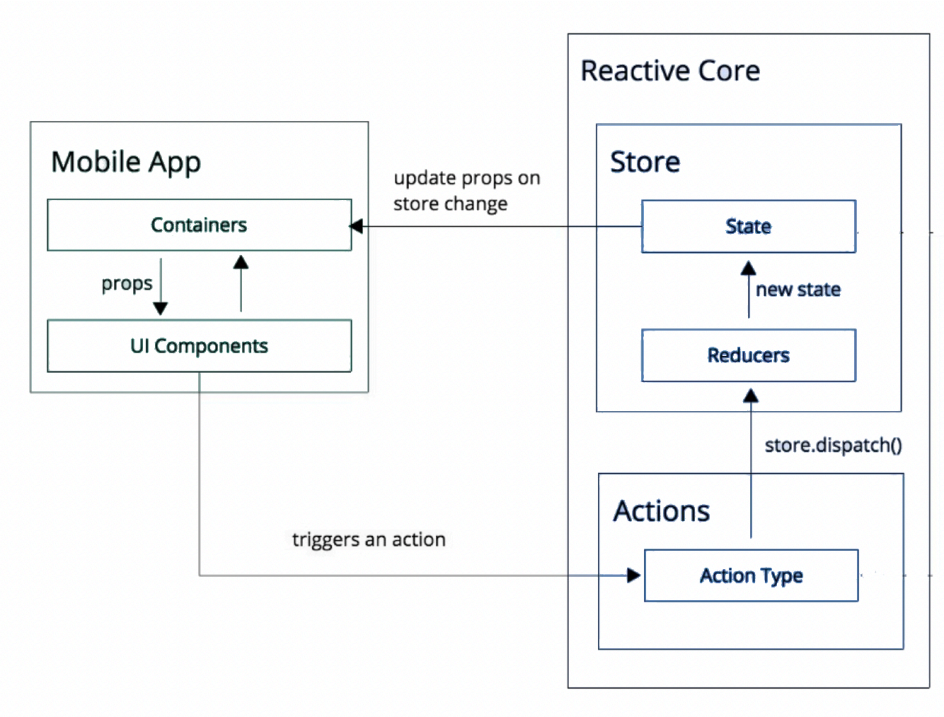
State diagram of join and leave messages and group



State diagram of logout

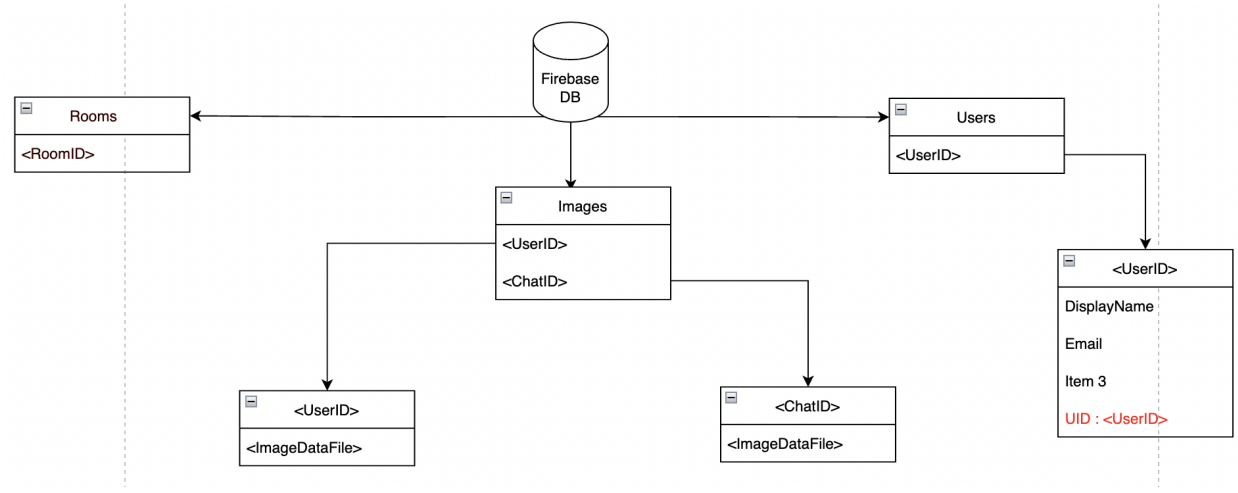


System Architecture and System Design



Persistent Data Storage

We are Using Firebase Realtime Database to Store Chat Data and User Data to persistent data storage.



Hardware requirements

Android Phone (AndroidOS 7+) or iPhone Phone (IOS 10+)

User Story

Identifier	User Story	Size
ST-1	As a user, I can log in to my account.	5
ST-2	As a new user, I can make a new account.	6
ST-3	As a user, I can log in even if I've forgotten the password.	3
ST-4	As a user I want to have all my messages stored so that I can view them at a later time	3
ST-5	As a user I want to see previous messages when I click a conversation so that I can read through previous messages	3
ST-6	As a user I want to delete a message from a conversation so that no one can view it.	3
ST-7	As a user I want to delete a conversation so that the messages are not available to anyone anymore	3
ST-8	I can see my chat with my friend on the chat section in chronological order.	2
ST-9	I can see my messages on the right and received messages on the left, so it is easy to differentiate.	2
ST-10	I can type my message in the text box to send it.	3
ST-11	I can send my message which can be text or image or file using the send button.	4
ST-12	I can receive the sent messages almost instantly, which makes the conversation flow better.	5
ST-13	I can see the time at which the message was sent or received.	3
ST-14	I can select my pictures or any document files I want to share in a group or with a person and send it in the chat.	5

ST-15	I can see the received or sent images or documents files in the chat.	4
ST-16	I can download the shared file or image from the chat.	4
ST-17	As a user I wanted to chat with multiple people in my class to share notes and delete unnecessary shared notes.	5
ST-18	As a user, I want to make audio calls and text messages that can leave after the conversation.	4
ST-19	As a user, a message to welcome the room user will be shown to the user and leave after the meeting.	4
ST-20	As a user a message username has joined will be broadcasted to all other users except the user who joined the room.	3
ST-21	As a user in function chat . if a user leaves the chat a disconnected message is broadcasted to all other rooms.	3
ST-22	As a user I want to receive notifications when there are new messages so that I can stay updated on my conversations	2
ST-23	As a user I want visual notifications and a sound so that I can be alerted of new messages even if I am not looking or if the sound is off.	1

History of Work

We have worked on web1 app projects using JS mainly also we have worked on web2 projects using JS frameworks. We planned to develop a webapp for this project since a website can be accessed from Computer, phone or other devices, it’s more accessible. But sice this is a realtime chat application and most of our user base will be phone users, we need the native feel and performance so we changed our proposal to a native app.

Now instead of developing separate applications for IOS and Android we decided to use a hybrid technology, React Native, which uses JavaScript as a programming language, from which we are familiar with.
So using React Native we can develop IOS, Android as well as Web app for our chat app project.

Now as a DataBase we were planning to use MongoDB but that would require us to create some API endpoints, so we decided to use FireBase from Google, which provides many services like realtime database and Authentication, etc.
