Deliverable 2a

Johanna Bell
Luca Bosch
Niklas Maier
Sebastian Schwarz
Simon Vogelbacher
Yasmin Hoffman

Konstanz, July 11, 2020

Advised by Till Niese

Contents

List of Figures

Part I

System Design

1 Introduction

1.1 Intention of system

The product we are going to create is on the one hand supposed to help users track their mood and to get an overview and on the other hand it can be used to evaluate all the data gathered to gain further knowledge about the phenomenon called emotional contagion.
The product is intended to be used by someone who is interested in tracking their mood, and by someone who wants to make an experiment to see how the mood can be influenced by others or in some special situation.

1.2 Design goals

1.2.1 System performance

1.2.1.1 Response time

We want to make sure the app runs smoothly, so the reaction time should never exceed one second. It is also of importance, that this is true regardless of the quality of internet connection. It should be possible to have at least 100 simultaneously active users. Since our application is not very complicated, we estimate that it should run on every smartphone running Android 8+.

1.2.1.2 Storage requirements

The storage requirements should definitely not exceed 100 MB for the mobile app and the amount of data stored on the server should not exceed 1 GB, this is especially important since the firebase server will not be able to store more with the free plan. There should not be any problem to adhere to this restriction since we do not expect to have an exorbitant amount of participants.

1.2.2 Reliability

We will thoroughly test the application to ensure that there are no errors. If there happen to be reports of app crashes, etc. anyway, we will make sure to contact the customer as soon as possible, and we will do everything we can to fix the problems quickly. Since the server will be hosted by Google, we have no direct control over its reliability, we assume there will be no to little downtime and no data loss.

1.2.2.1 Robustness

General things like wrong user input can lead to unexpected behaviour in the system. To avoid crashes and errors we want to allow only valid user input. So the system will, for example, only allow email addresses which are in the right format when the user has to type his email address in. That way we can at least ensure that the input is an email address.

1.2.2.2 Availability

Our service will expectedly be available 24/7 except during power outages.

1.2.2.3 Security

The system will encrypt the passwords of the participants and the admin. Furthermore we will store all user data anonymously and won't store GPS locations, only GPS distances to Companions.

1.2.3 Maintainability

The mood tracking system is not intended to be maintained after the development is finished. However in the development of the application we want to make sure that the code is well written and understandable. To ensure that other developers may be able to maintain and extend our system in the future, we want to develop well commented code.

1.2.3.1 Extensibility

By using well commented and structured code, it will be easy for us or other development teams to pick the classes up and add additional functionalities to the app.

1.2.3.2 Portability

Since our app is programmed in Java, it should be fairly portable. For the moment we only plan an android app. As far as the web app is concerned, it will probably run on every currently available browser so it should be really portable.

1.2.3.3 Legibility

We will use Javadoc-comments for our methods and classes so even without having to interpret the whole code by yourself, you can deduce the return values from the text.

1.2.4 User criteria

The app will be intuitive and easy for information logging and the visualization tool. It will be self explanatory and provides the user a smooth, quick and intuitive navigation.

1.2.5 Costs

For this particular project, there will be no monetary cost. Instead the cost consists of all the hours of work our team puts into the creation of documents, models and plans. We will have to put a lot of effort into the code as well in order to meet all the requirements and there will be additional work, if the system is to be upgraded or maintained for a long period of time.

1.2.5.1 Development costs

Planning the app, that is, creating all of these deliverables will take a large percentage of the time (course start until the final deadline). Programming the app will surely be time consuming as well (some weeks).

1.2.5.2 Deployment costs

We don't expect to have any deployment costs. We will simply upload the finished system. Others will take care of the deployment.

1.2.5.3 Upgrade cost

Upgrade costs will be rather low, since our system, and therefore the code, will be simple and short respectively. So it shouldn't take too long of a time to modify anything.

1.2.5.4 Maintenance costs

Since we are using the free firebase plan, we are expecting the monetary maitenance costs to be 0. We might have to take the time to fix some undiscovered bugs though.

1.3 Contradicting design goals

We don't think there are any contradicting goals in particular, but we expect it to be hard to keep the UI intuitive and self-explanatory if there are a lot of functions the user has access to. For instance in the visualization screen, where we provide a lot of options, it can be hard to keep everything tidy and neat without making compromises regarding the functionality. We want to prioritize ease to use though and therefore, if we are unable to make a certain interface easy to use, we will try to split it in two or search for a way to put a part of the functionalities elsewhere.

1.4 Definitions, acronyms and abbreviations

Mood
: This refers to how good or bad a user is feeling. Mood will be measured on a five point scale from "very bad" to "very good".

Companion
: A user that has sent or been sent a friendship request, that was accepted. Each companion is associated a relationship by the other user. Users can see the mood changes of their companions after the end of an experimental cycle. It will be noted by the app if a companion is nearby while answering mood questions. Through this companion feature the phenomenon of emotional contagion will be explored.

Special Situations
: Events that take place in the daily life of a person (related to work, family, health, etc.) and are associated with with a negative or positive impact on their mood.

Emotional Contagion
: Happening when people's emotions and behaviours influence the emotions/behaviours of other people.

(G)UI
: *(Graphical) User Interface*. What the user visually perceives when using the application. All the different points where the user can interact with the system to make it do what he/she wants.

Intuitive
: By saying the application should be intuitive, we mean that the user should be able to understand how to use all of its functionalities without having to read an instruction manual beforehand.

1.5 Literature references

- Bernd Brügge, Allen H. Dutoit, *Object Oriented Software Engineering* Using UML, Patterns, and Java: International Version, 2009.

1.6 Overview

The mood tracker app we are developing is intended to be usable for scientific purposes in the field of psychology but also aims to be a user friendly mood tracking app for casual users to track their mood. The major point of the project is to track mood and relaxation levels in special situations and with companions in order to gain scientific insights on how people's moods affect each other. The goal of our development is to produce an application which is reliable and stable and which satisfies the requirements of a scientific environment. We are producing the application until delivery and after the development is finished the application will not be maintained by us. However we want to try our best to make the code as easy to maintain as possible. We are not payed for the project since it is in scope of a software project in our studies of computer science at the university of Constance.

2 Existing system

The client has no existing system, which can be used as code base to upgrade or change for our mood tracking system, however there are a few mood tracker apps available, which the client likes and that can be used as inspiration. However none of them offer the ability to see or research how an individual's emotions are influenced by the people around them. The client stated that she liked the design and stucture of the app called 'Daylio' a lot, therefore we could base parts of our work on it, though we try to avoid copying large parts as is. The process of recording and visualizing information and the design could be good inspiration, whereas the collection of data in an experiment is not part of the functionality.

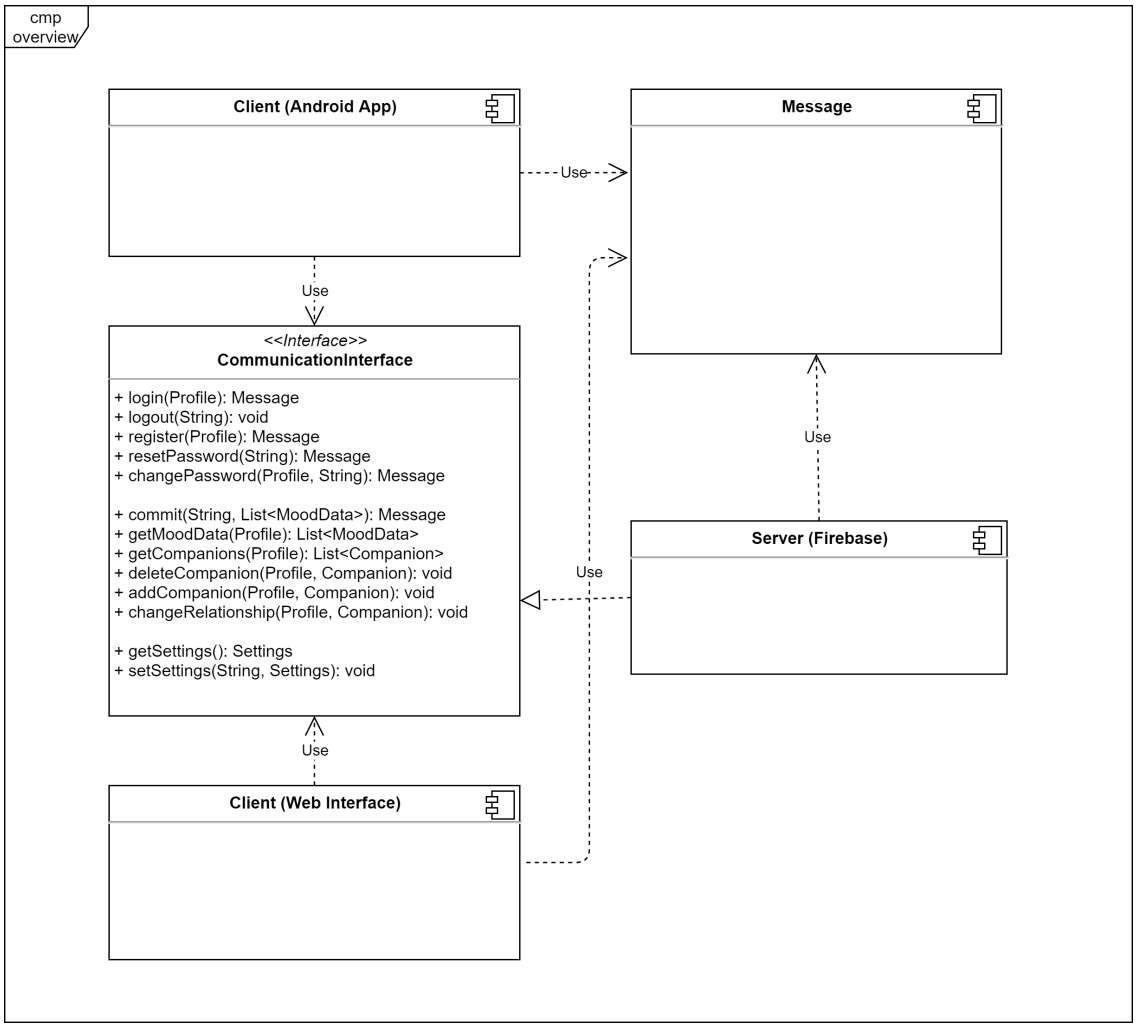
3 Suggested system

3.1 Overview

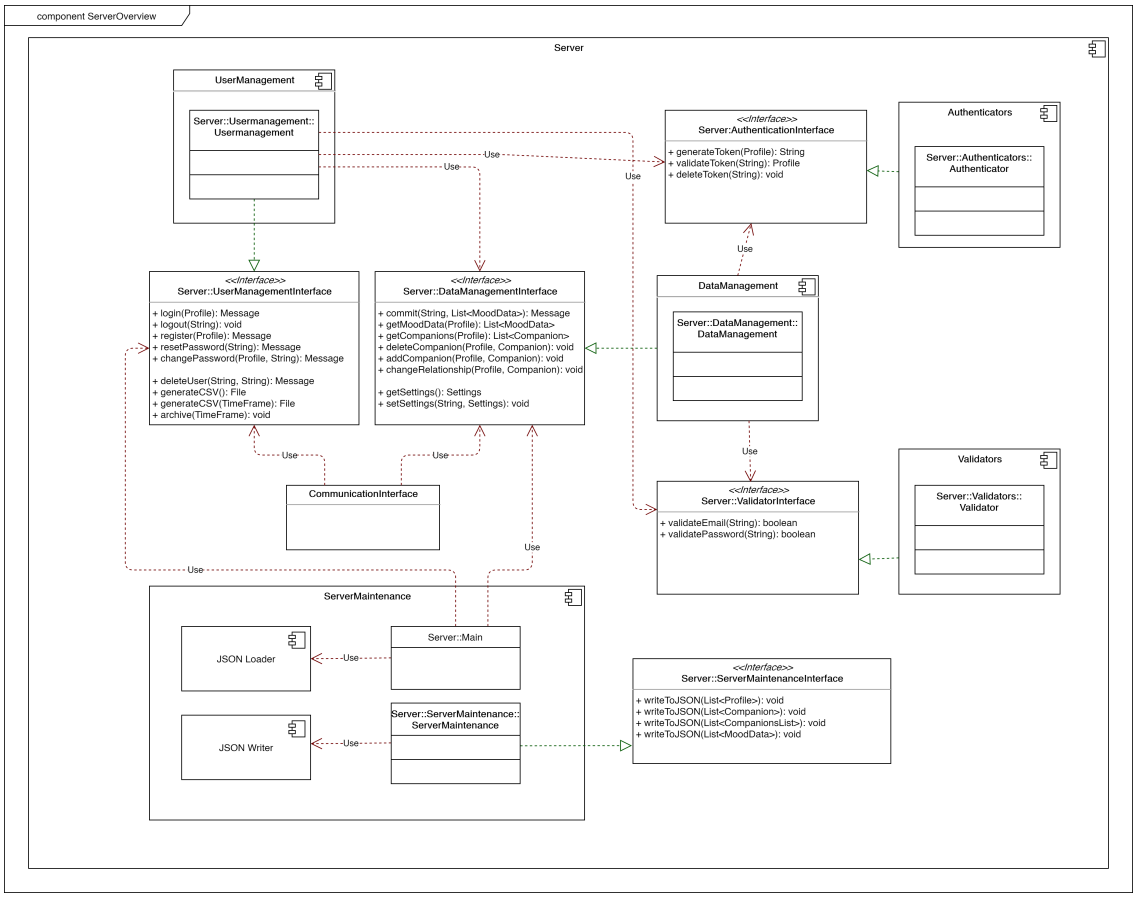
The suggested system is divided into three components. We have the android app, a server and a web interface. The app is the most extensive part of this project. The data gathered in this app, will be uploaded to the server and can be downloaded via the web interface by the leader of the experiment. This web interface will also be used by him or her to change the settings that are to be synchronized with the android app. For that purpose, we will again transfer the data via the server. Additionally, the users of the app can also download data from other people participating in the experiment after it has ended from the server.

3.2 Component segmentation and description of interfaces

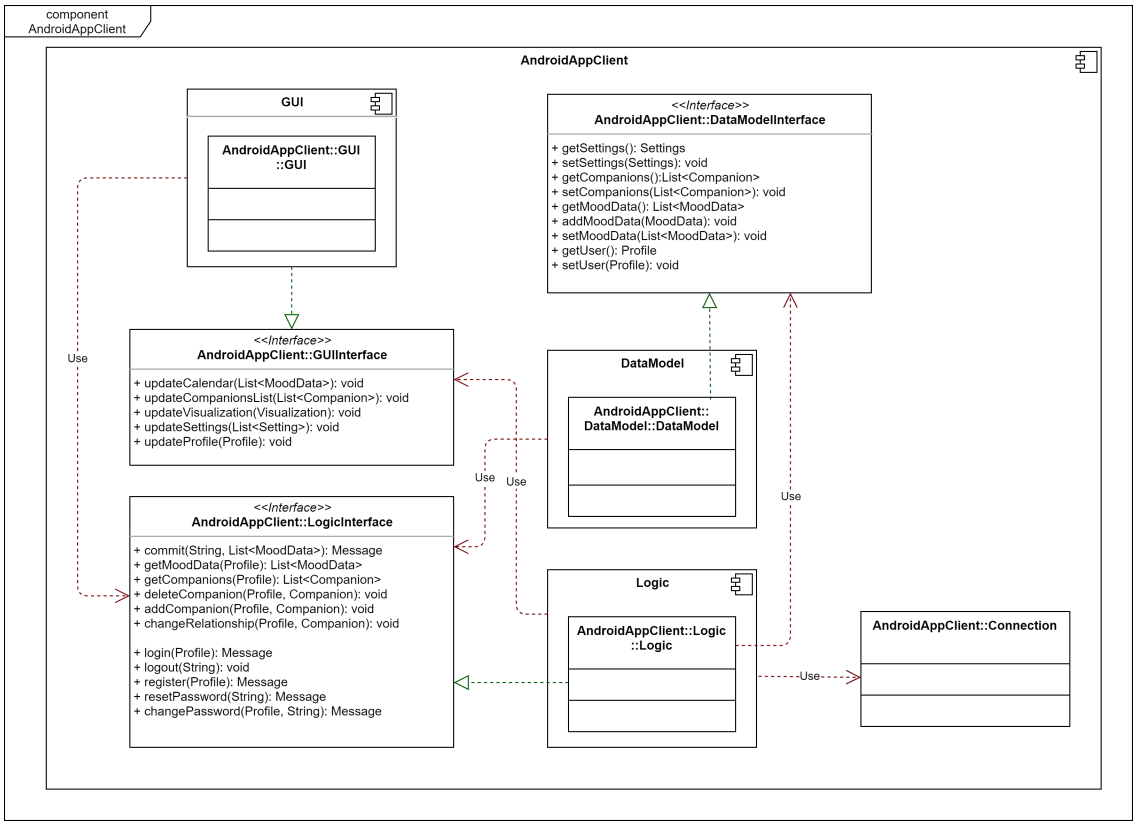
The following diagram gives an overview over the different components of our system and how they are going to communicate with each other. There are 2 client components (android and web application) and a server component that all communicate through the message component.

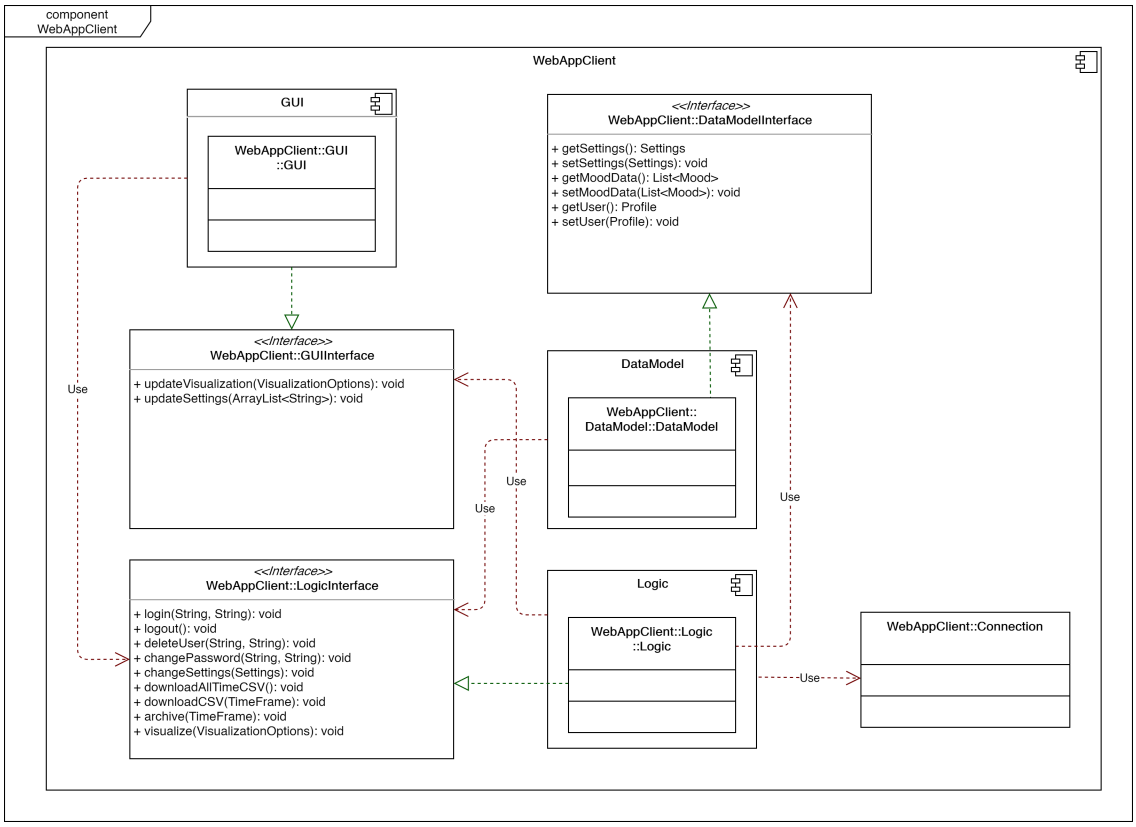


The server mainly consists of three components, the server maintenance component, the user management component and the data management component. There is a validators component and a authentication component used by both the data and user management component.



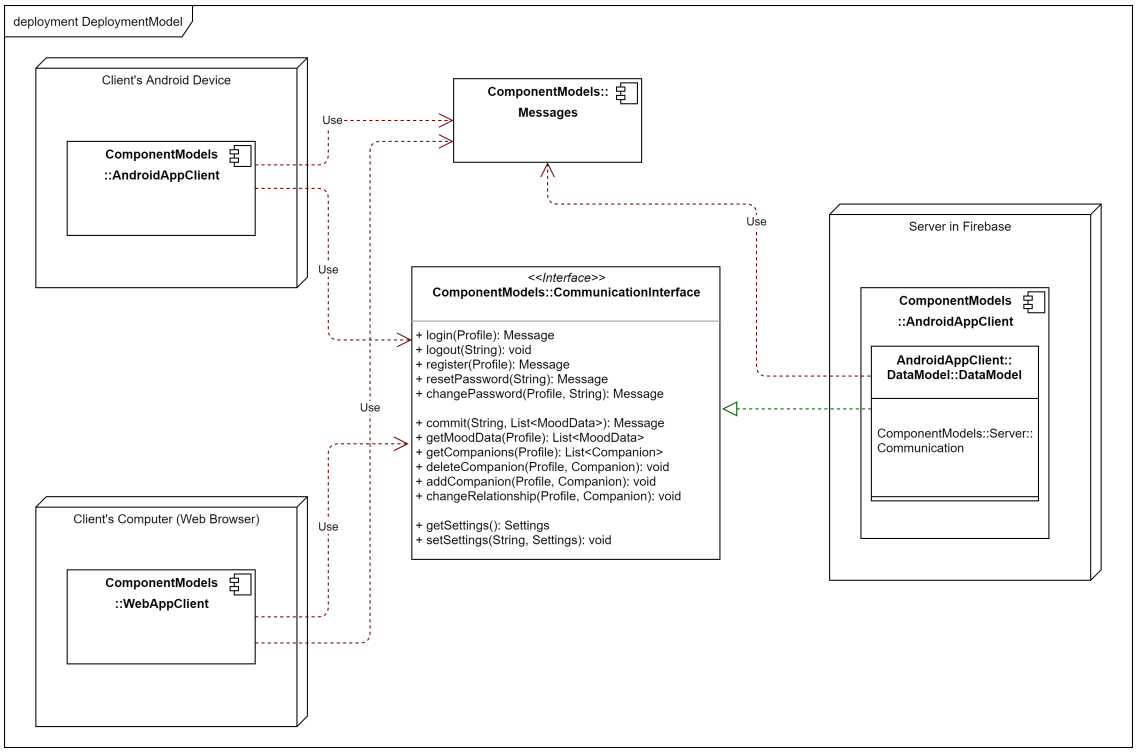
The android app and the web app consist of three main components each, one being the GUIInterface the participants and admin use to interact with the system, the LogicInterface summarizing the logic methods used to communicate with the server and the DataModelInterface listing methods used to manage locally cached data.




component
WebAppClient
WebAppClient
GUI
WebAppClient::GUI
::GUI
<<Interface>>
WebAppClient::DataModelInterface
+ getSettings(): Settings
+ setSettings(Settings): void
+ getMoodData(): List<Mood>
+ setMoodData(List<Mood>): void
+ getUser(): Profile
+ setUser(Profile): void
<<Interface>>
WebAppClient::GUIInterface
+ updateVisualization(VisualizationOptions): void
+ updateSettings(ArrayList<String>): void
DataModel
WebAppClient::
DataModel::DataModel
Use
Use
Use
Use
<<Interface>>
WebAppClient::LogicInterface
+ login(String, String): void
+ logout(): void
+ deleteUser(String, String): void
+ changePassword(String, String): void
+ changeSettings(Settings): void
+ downloadAllTimeCSV(): void
+ downloadCSV(TimeFrame): void
+ archive(TimeFrame): void
+ visualize(VisualizationOptions): void
Logic
WebAppClient::Logic
::Logic
Use
WebAppClient::Connection

3.3 Hardware/Software mapping

This following diagram shows how the different components are deployed. The Client Android App component is fully deployed in the Client's Android device and the Client Web Interface component is fully deployed in the Client's Web Browser. The Server component is fully deployed in a Firebase Server.



3.4 Management of persistent data

Client:
Daily mood data is saved locally on the users' devices until the end of the day. At the end of a day, the app will automatically attempt to synchronize data with the server by sending the locally stored data to the server. The app will keep the data on local storage until a connection was successfully established and the server responds with a message confirming that it has received and stored the data.
Other data including authentication token, project settings, companion data and past mood data of the logged in user is cached on the device. When launching the app, it automatically attempts to synchronize this data by fetching current values from the server. If no connection can be established, the cached data will be used.

Server:
The Firebase server saves all project data in a database utilizing the JSON (JavaScript object notation) format. This includes project settings, account and authentication data, user companion and mood data.
To circumvent data loss, requests from the clients (web interface, Android app) will always be responded to with a response code indicating either success or failure. The server immediately attempts to save the data on the database. The success code will be sent when a request was successfully processed and the respective data was saved on the database. If the request could not be successfully processed, a failure code will be sent in response to the request.

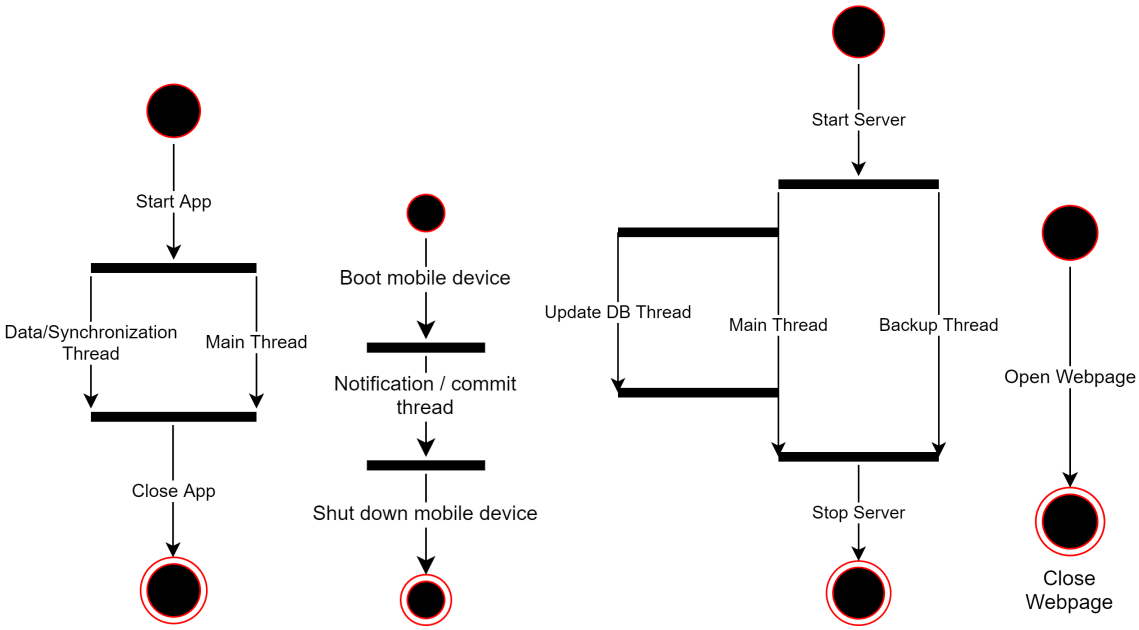
3.5 Access rights and access control

	Data Mangement	User Management
Everyone	getSettings() getVisualization() getMoodData()	login() logout() changePassword()
Participant	commitMood() getCompanions() addCompanion() deleteCompanion() changeRelationship()	register() resetPassword()
Admin	changeSettings() downloadCSV()	deleteAccount()

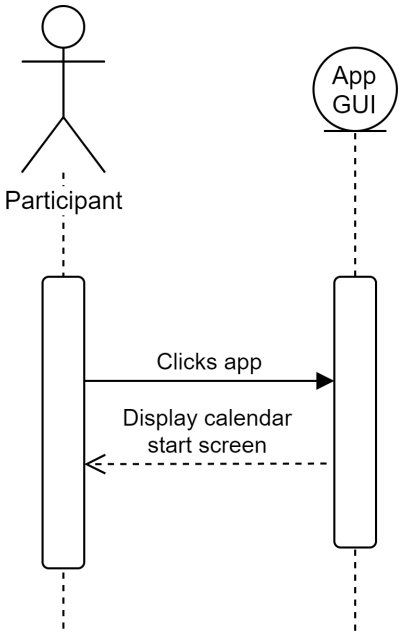
The admin will not be able to mess with the companions data, be it companions or recorded mood. However he/she will be able to change the settings of the app, since these also include the terms of use and experiment duration, which needs to be changed from the web interface. Even though the user can also set his settings (like notifications), he cannot change the experiment date for example, this option will not be displayed in the interface. The passwords will be stored encrypted using hashing so they will never be revealed to the public.

3.6 Global control flow

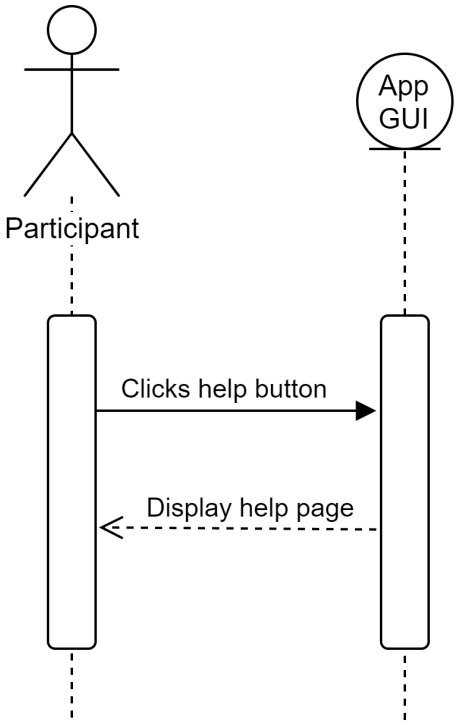
Our system is divided into three components: The server, the android app and the web app.
The server will have one main thread running and one backup thread, which will regularly save the data. To ensure that the data is always consistent, we need to ensure the atomarity of the commits. Therefore we will open a new thread to synchronize the server data with the data from the app users. Only if the data transmission process is completed successfully, the database will be updated. This is importance, since often the mobile network disconnects in remote areas or tunnels and we need to avoid incomplete entries at all cost.
However, since we are using firebase, we are confident that such mechanisms are taken care of.
The mobile application will work with one thread processing the saved data and the communication with the server, that is synchronizing the mood data and the settings and another thread that takes care of the GUI. This way, bad internet connection will not prevent the user from fluidly using the application. By using events for user inputs, we can assure that this division is possible.
Additionally, the app will run a permanent notification thread in the background for while the mobile device is running. This thread is responsible for showing the user randomized notifications, even while they're not using the app, as well as automatically commit locally saved mood data at the end of the day after the last notification.
The web app will run in a browser so an internet connection is permanently required. Since no data will be locally saved, we can work with one single thread taking care of everything.



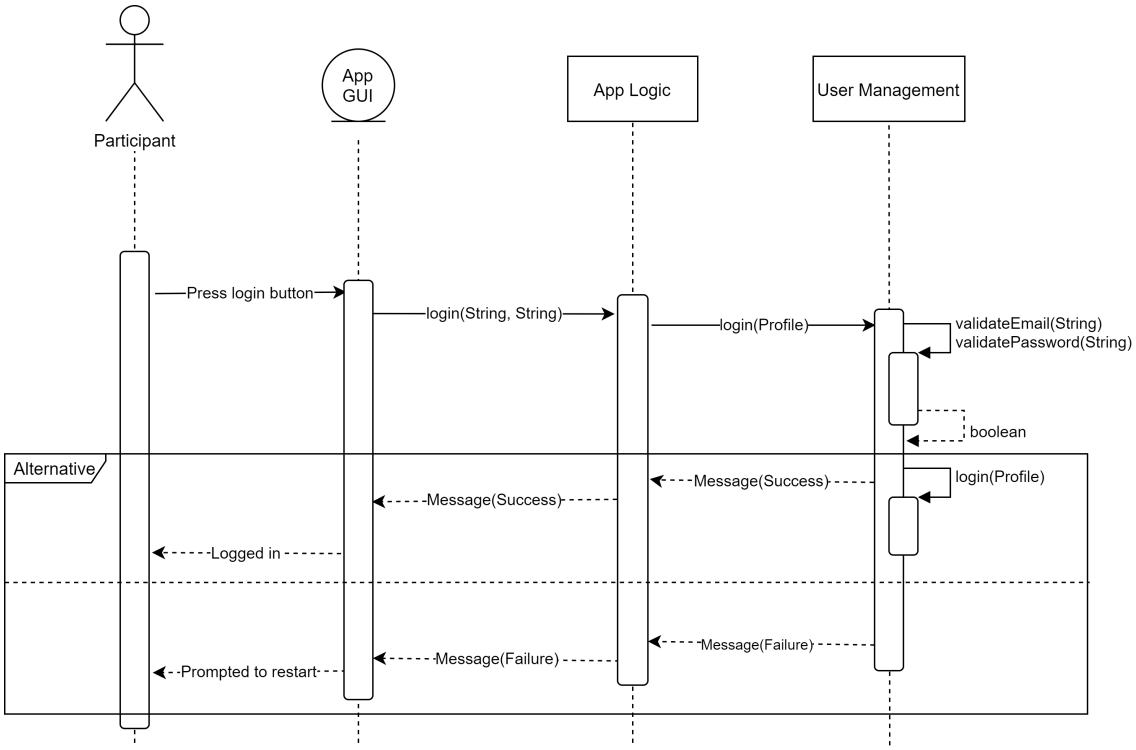
The following diagram displays the control flow of when the participant starts the application:



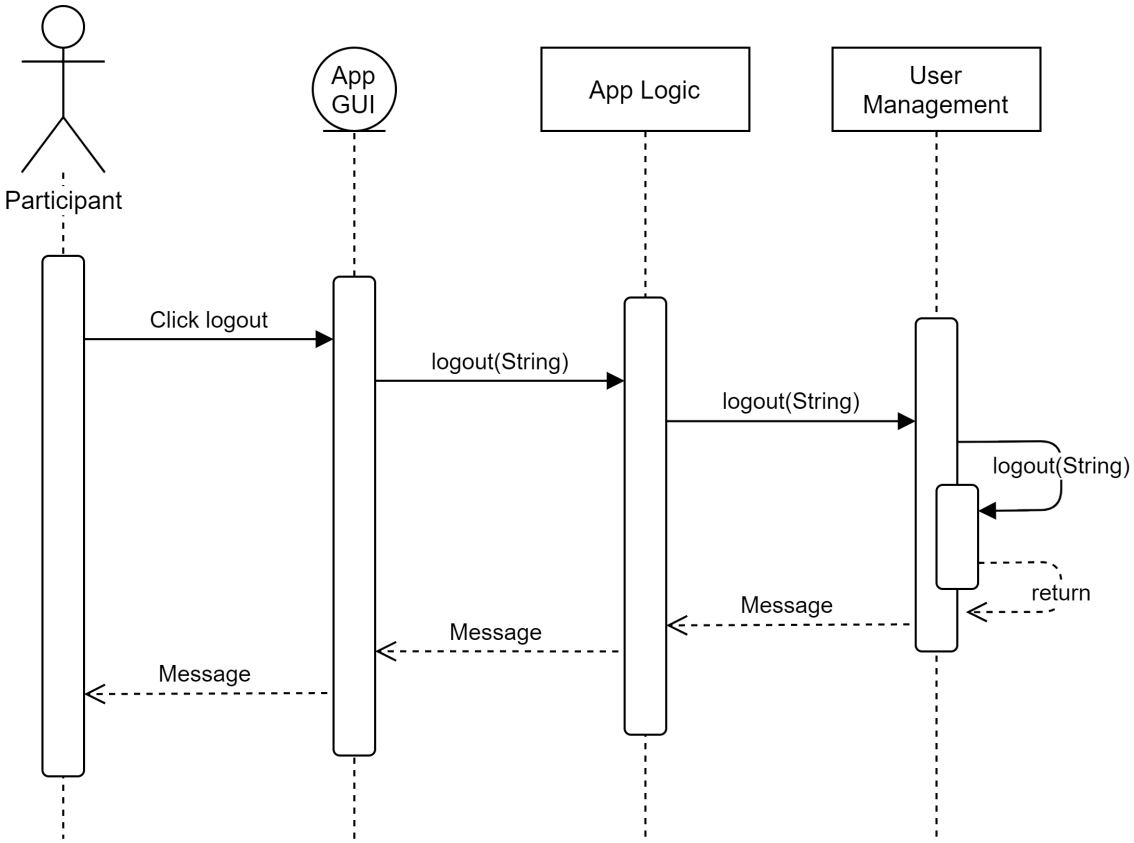
The following diagram displays the control flow of when the client presses the help button in the application:



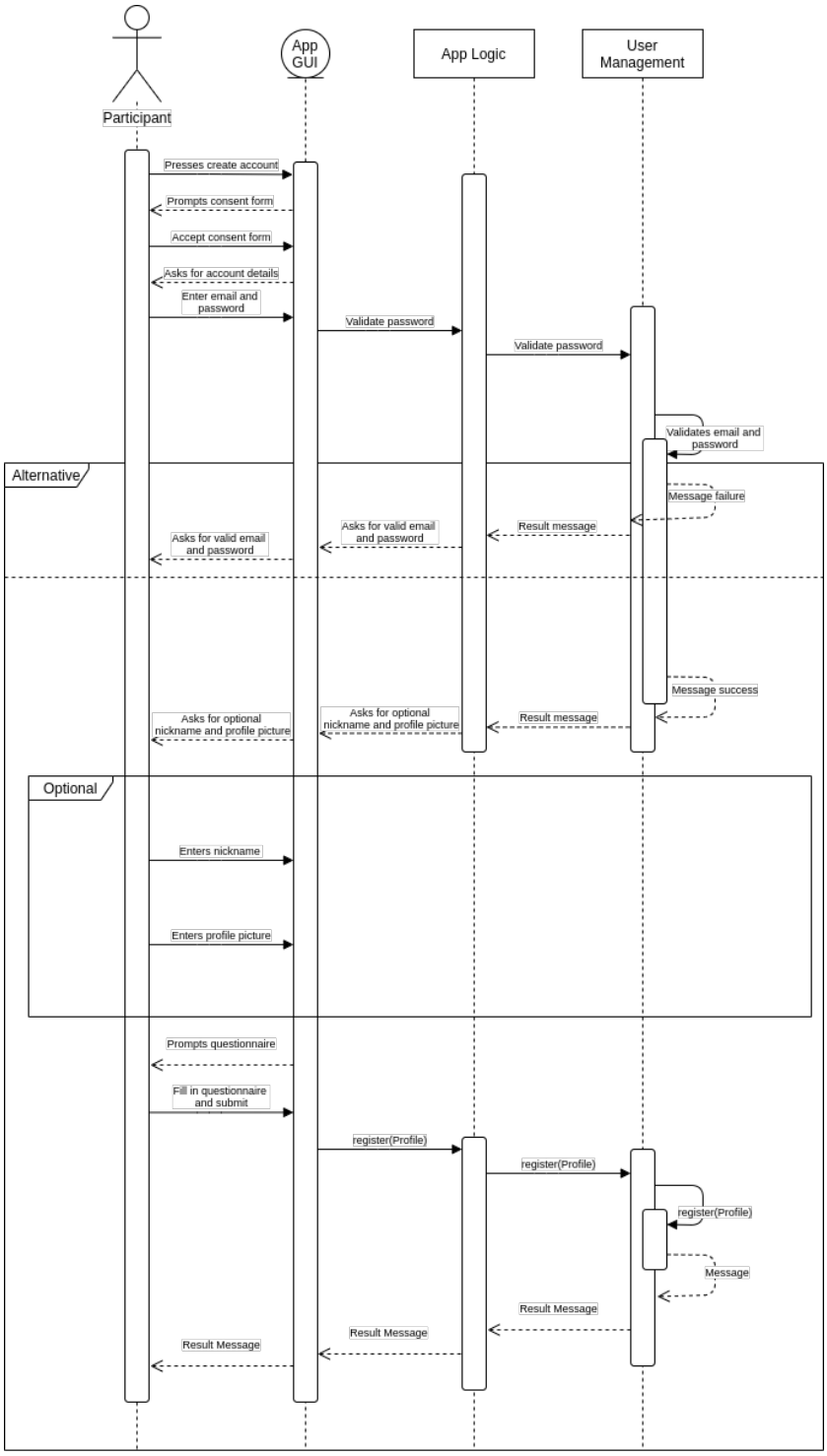
The following diagram displays the control flow of the client-invoked method call on the server when a participant logs in:



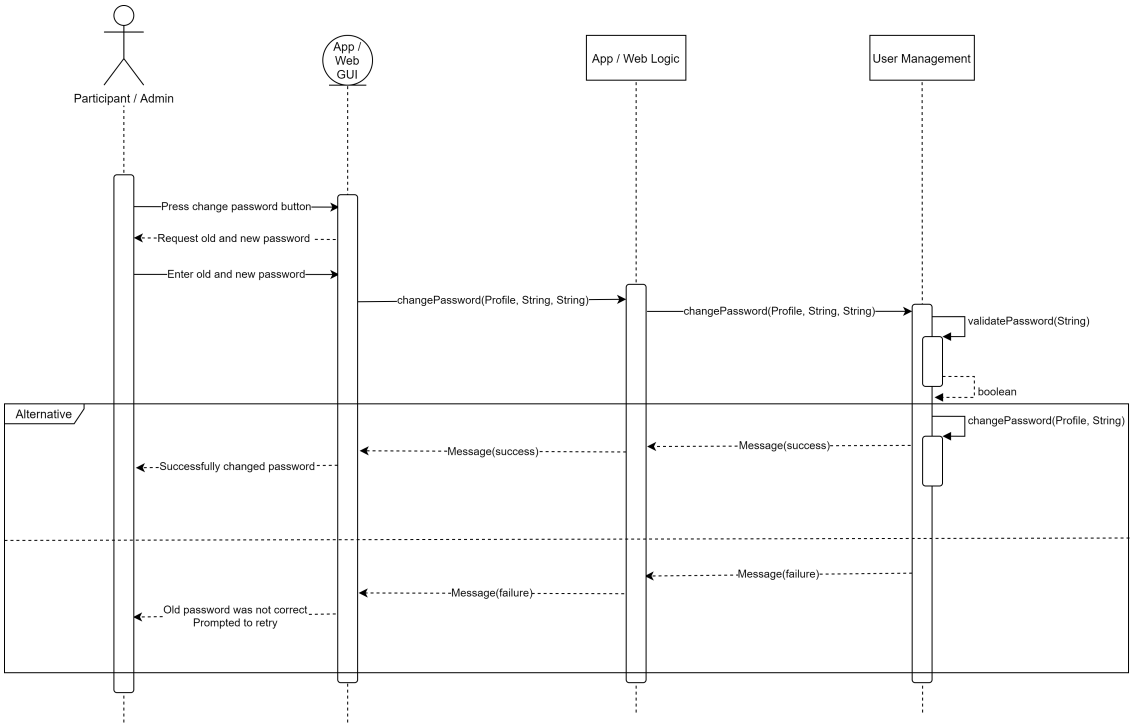
The following diagram displays the control flow of the client-invoked method call on the server when a participant logs out:



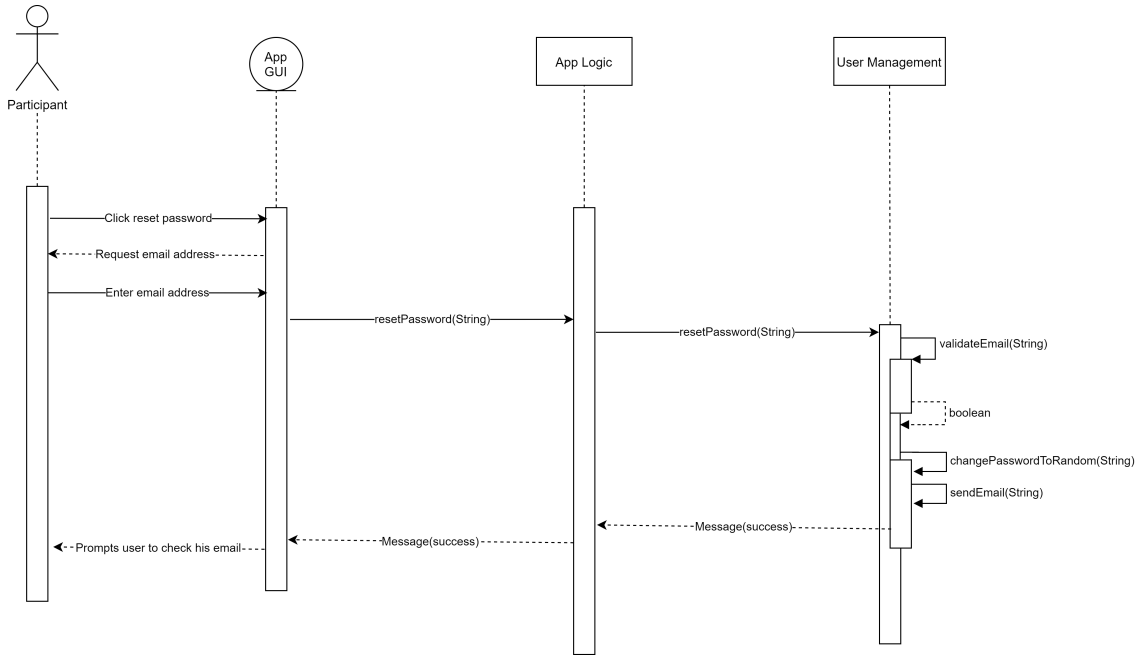
The following diagram displays the control flow of the client-invoked method call on the server when a participant signs up:



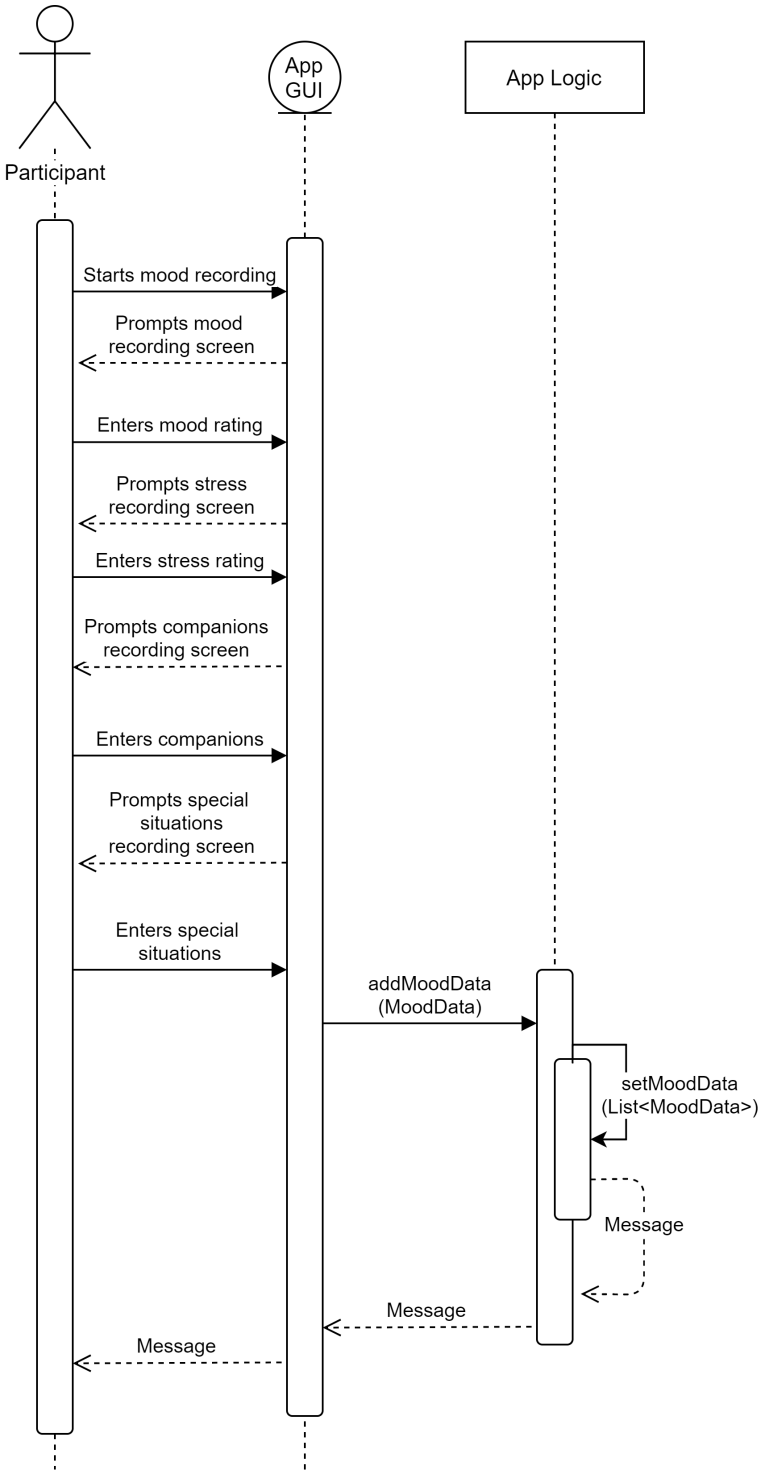
The following diagram displays the control flow of the client-invoked method call on the server when a participant or administrator changes his password:



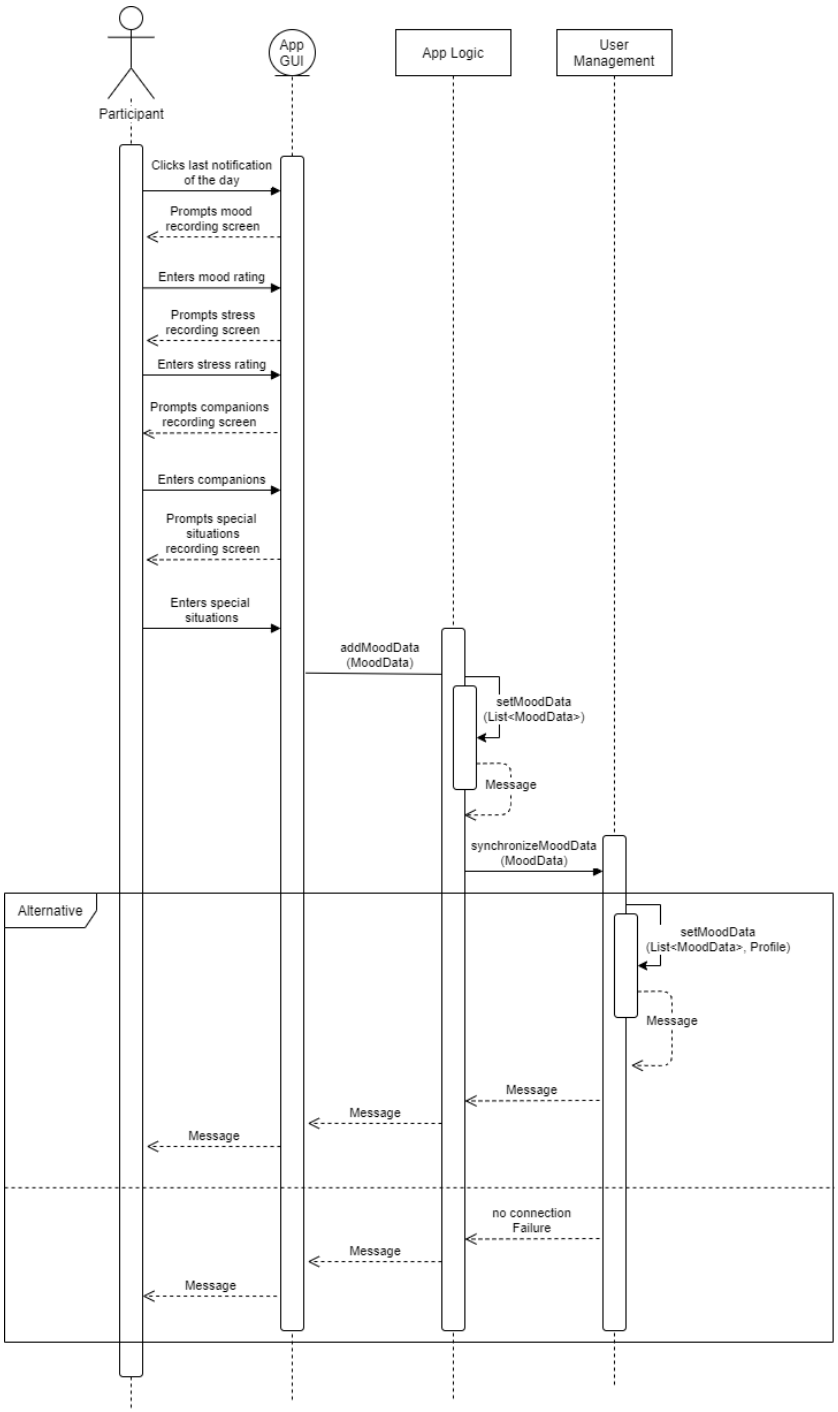
The following diagram displays the control flow of the client-invoked method call on the server when a participant resets his password:



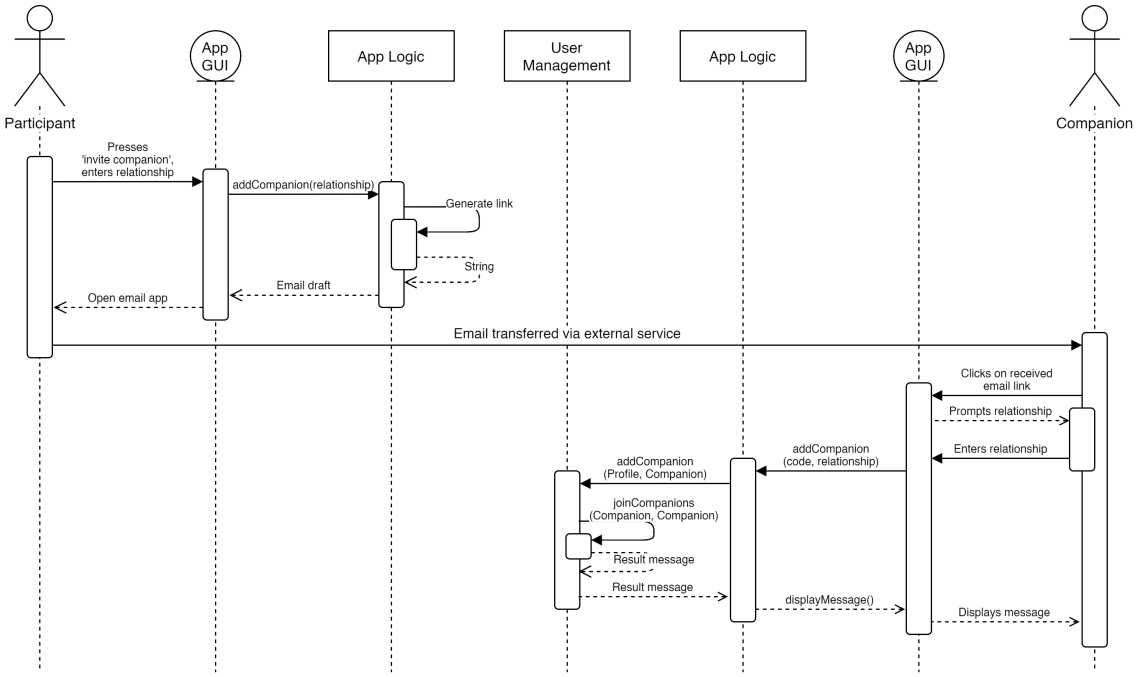
The following diagram displays the control flow of the client-invoked method call on the server when a participant makes a mood recording:



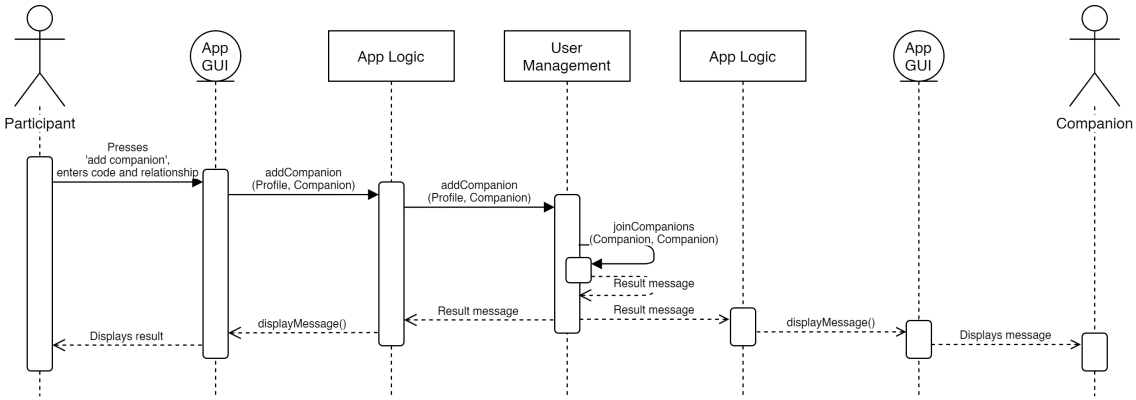
The following diagram displays the control flow of when the participant makes the last mood recording of the day and the data is synchronized with the server successfully or the data is not synchronized with the server, because of connection Failure:



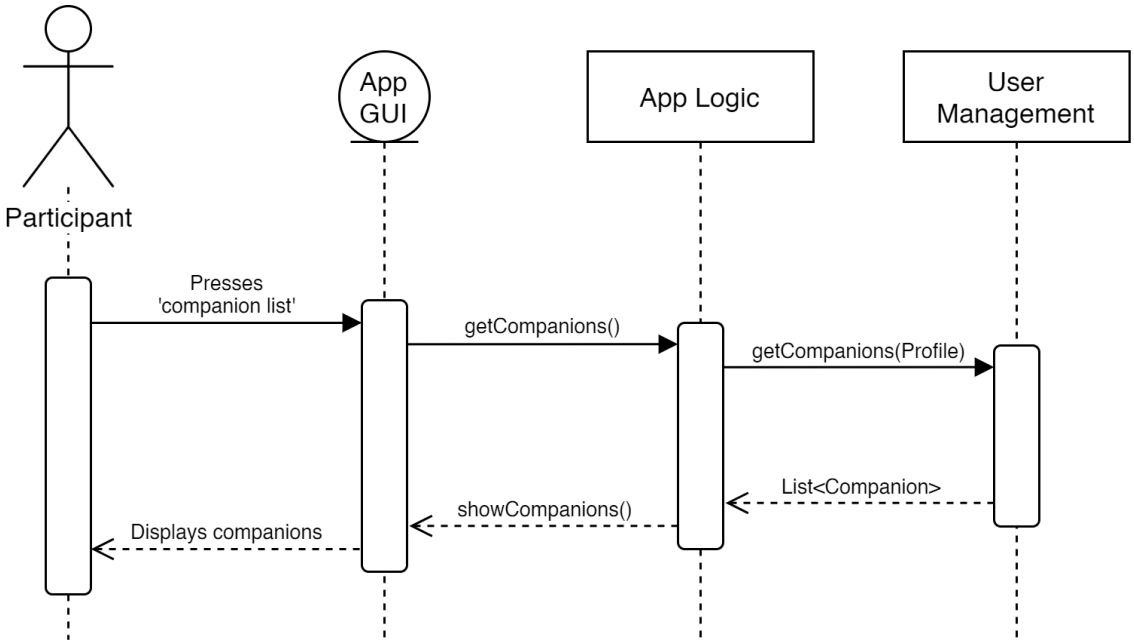
The following diagram displays the control flow of the client-invoked method calls on the server when a participant invites a companion:



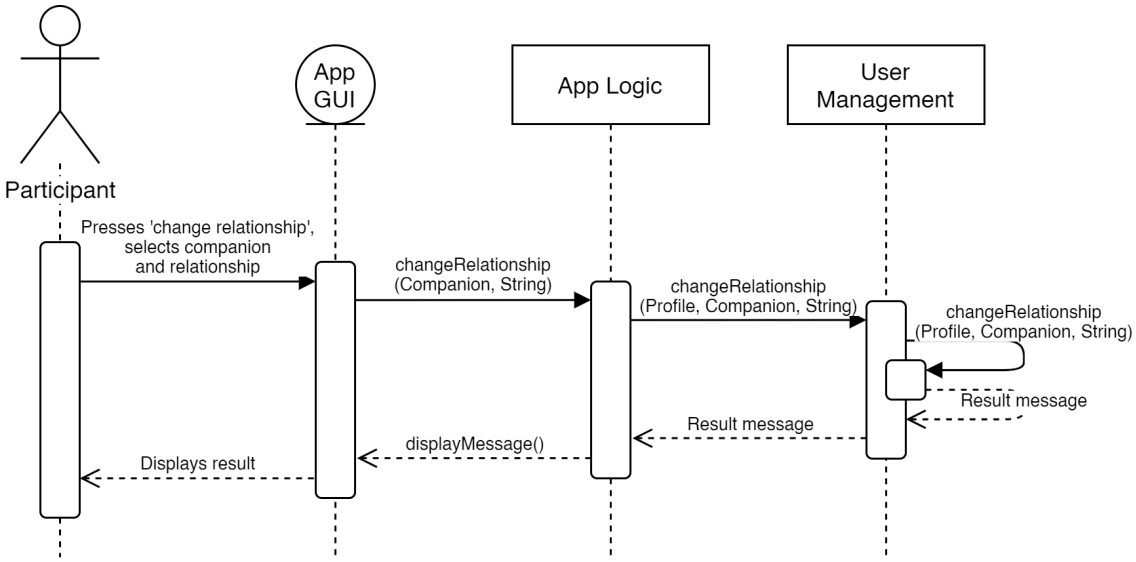
The following diagram displays the control flow of the client-invoked method calls on the server when a participant adds a companion:



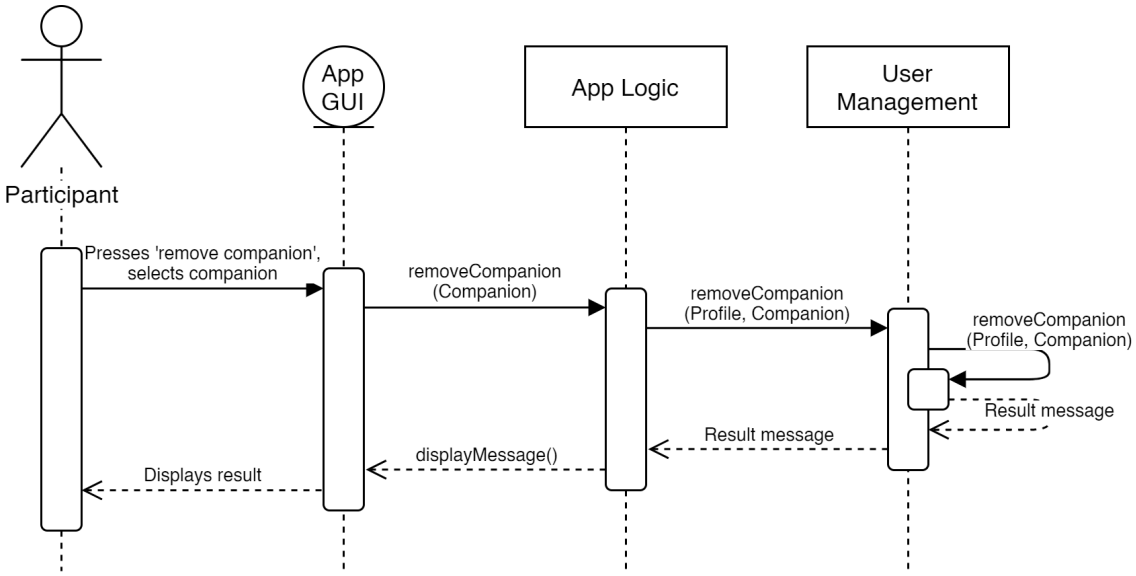
The following diagram displays the control flow of when a participant gets their list of companions:



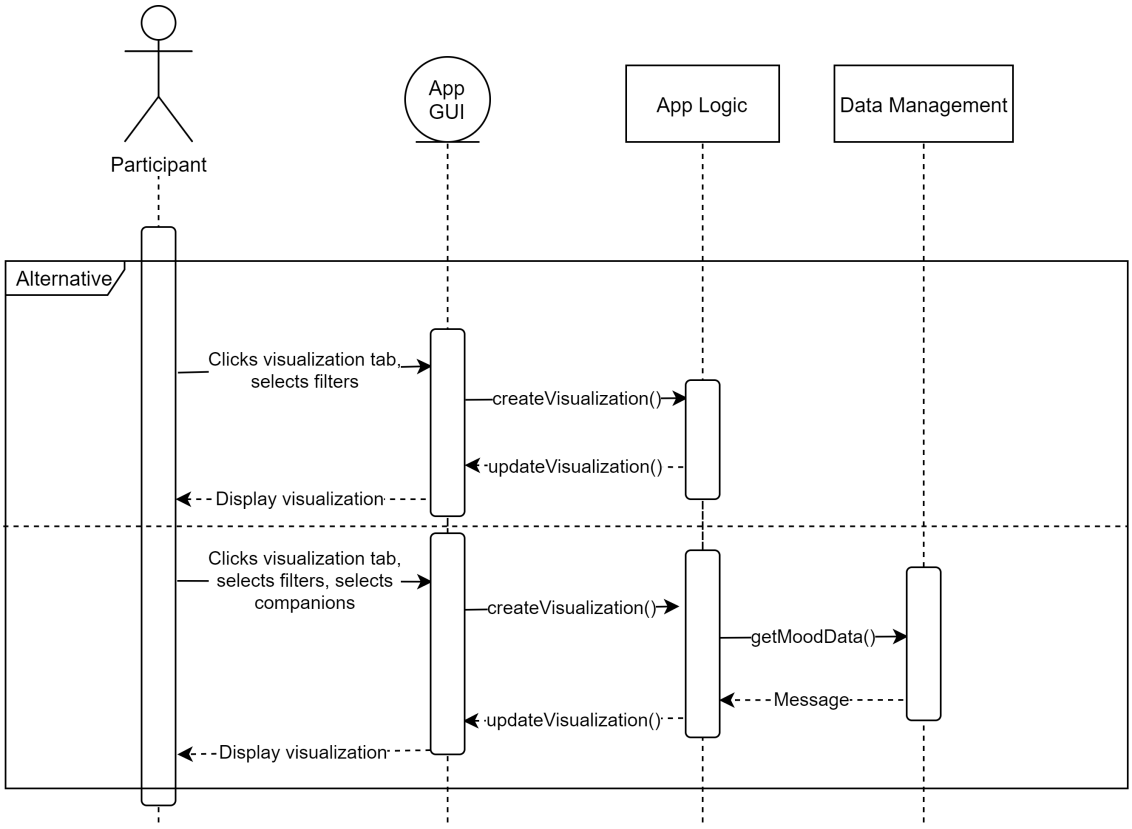
The following diagram displays the control flow of the client-invoked method calls on the server when a participant changes their relationship with a companion:



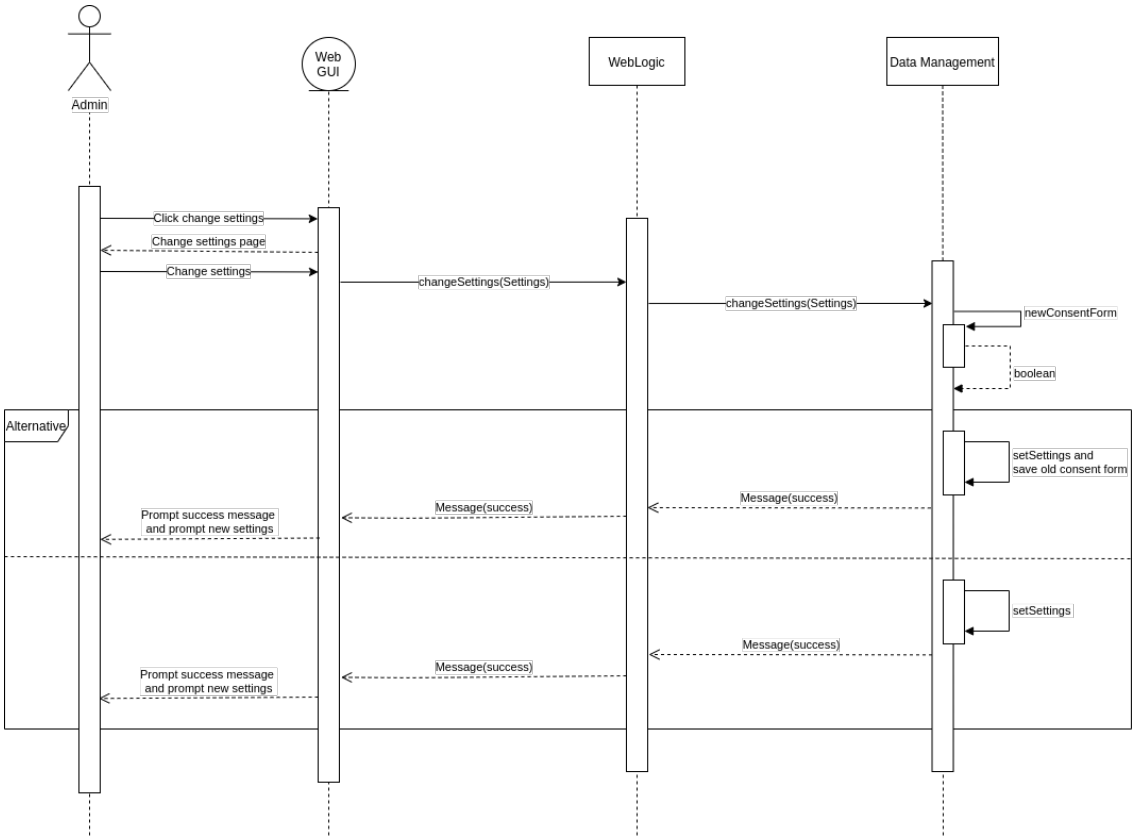
The following diagram displays the control flow of the client-invoked method calls on the server when a participant removes a companion:



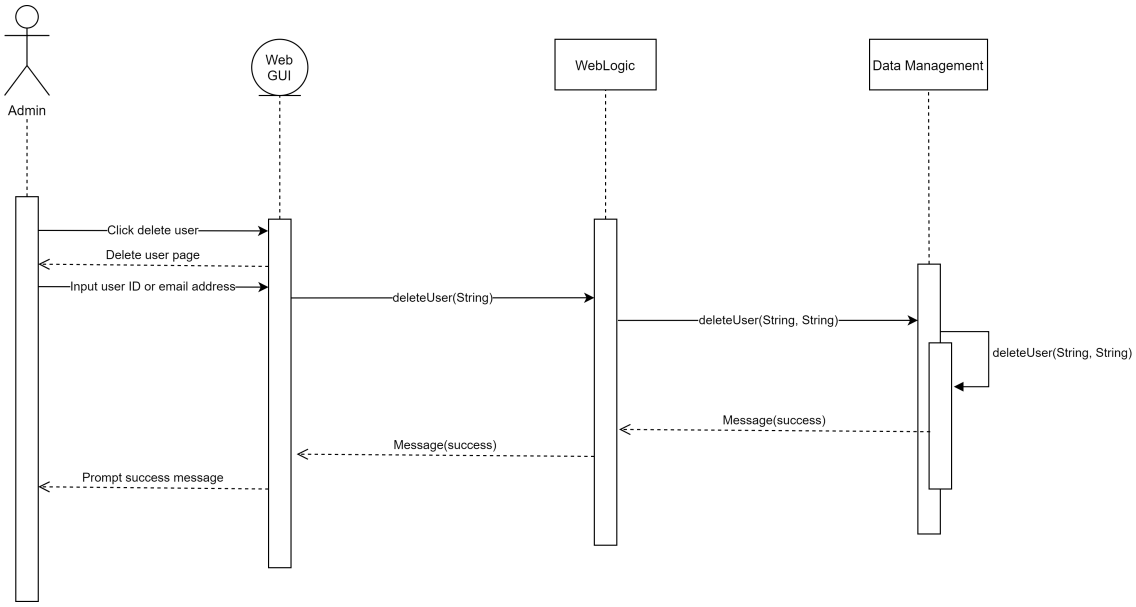
The following diagram displays the control flow of the client-invoked method call on the server when a participant wants to visualize mood data:



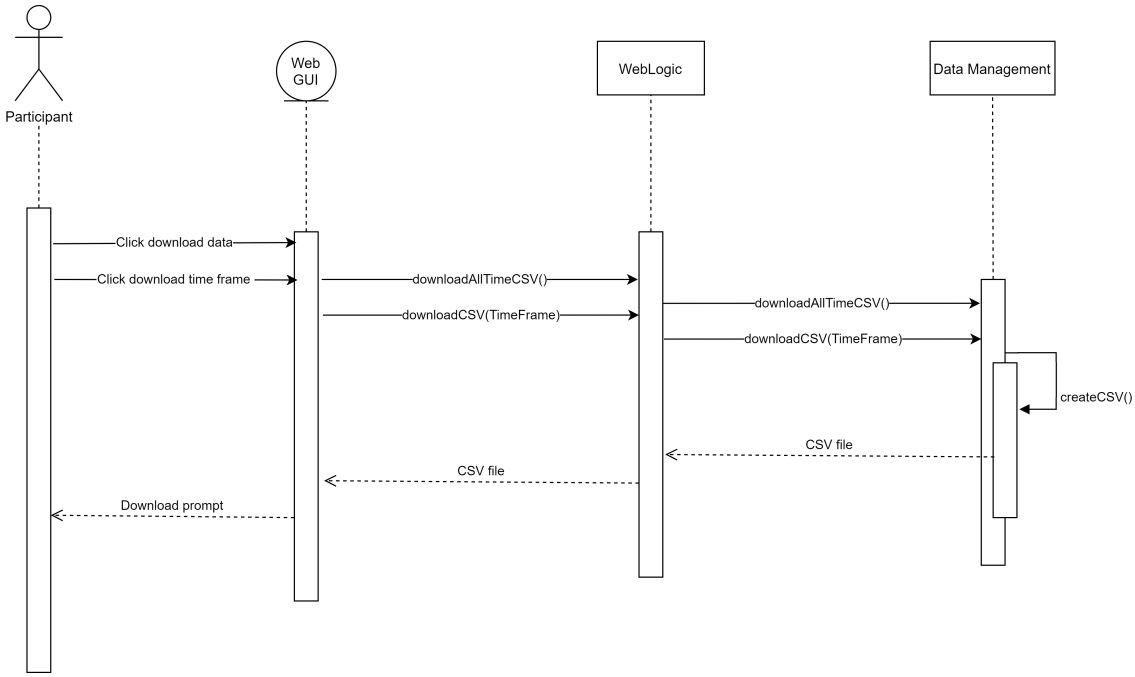
The following diagram displays the control flow of the client-invoked method call on the server when the administrator changes the project settings:



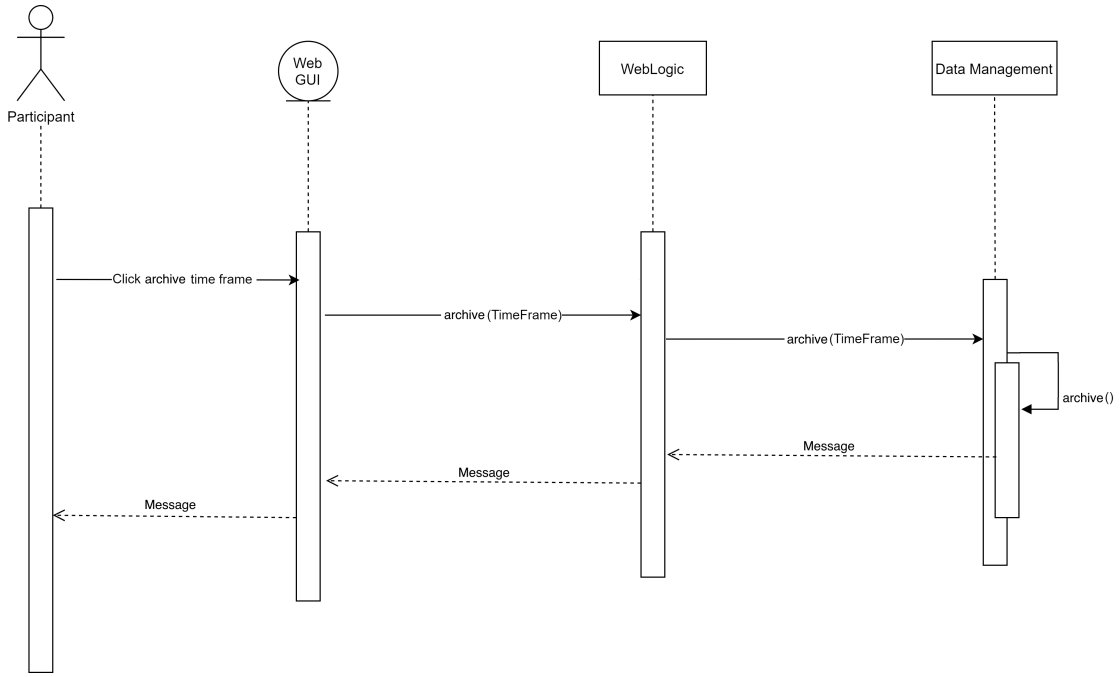
The following diagram displays the control flow of the client-invoked method call on the server when the administrator deletes a user:



The following diagram displays the control flow of the client-invoked method call on the server when the administrator wants to download mood data:

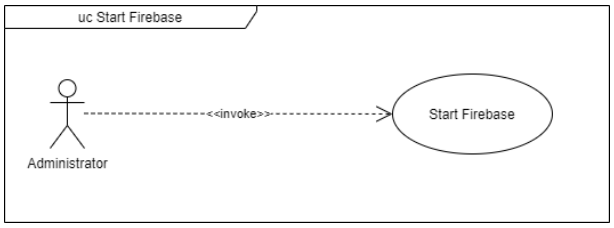


The following diagram displays the control flow of the client-invoked method call on the server when the administrator wants to archive mood data:

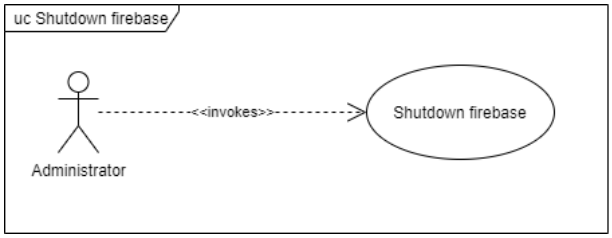


3.7 Boundary Conditions

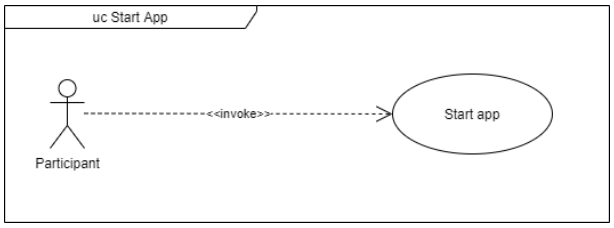
Use case name	`Start firebase`
Entry condition	the firebase server is not running
Flow of events	1. ADMINISTRATOR starts the firebase server
Exit condition	The server is online and it is possible to connect via a client to it.



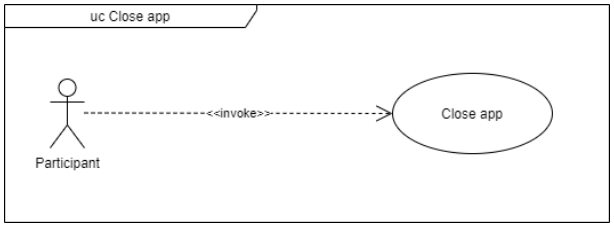
Use case name	`Shut down firebase`
Entry condition	the firebase server is running
Flow of events	1. ADMINISTRATOR stops the firebase server
Exit condition	The firebase server is no longer online. A client can't connect to it or looses the connection.



Use case name	`Start app`
Entry condition	the app is not running
Flow of events	1. Participant clicks on app symbol 2. The app starts
Exit condition	Participant is able to use the app



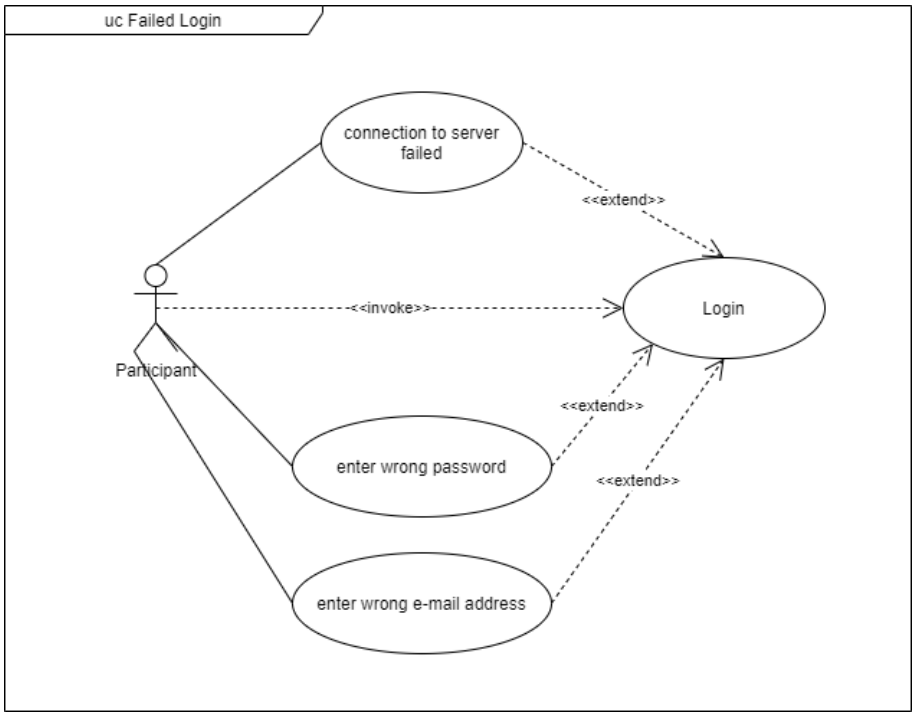
Use case name	`Close app`
Entry condition	The app is running
Flow of events	1. Participant leaves the app and removes it from the task manager 2. The app is terminated
Exit condition	Participant is no longer able to use the app



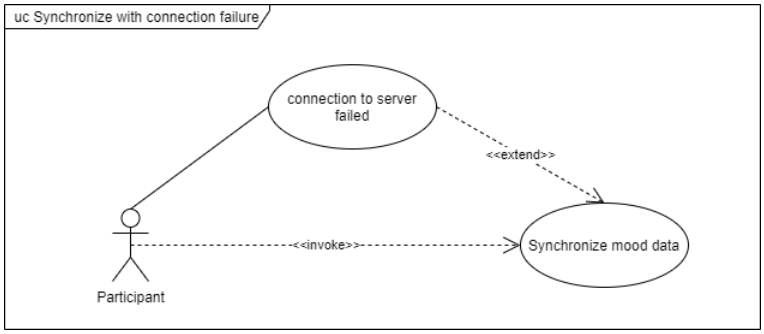
Use case name	`Login with wrong password`
Entry condition	The app is running on the smartphone
Flow of events	1. Participant enters the data and clicks on the login button 2. App connects to server and sends the entered login data 3. The server validates the login data and notices that the entered password is wrong 4. Server sends a wrong password message to the client 5. App displays the message and gives the possibility to enter the login data
Exit condition	The Participant is not logged in

Use case name	`Login with wrong e-mail address`
Entry condition	The app is running on the smartphone
Flow of events	1. Participant enters the data and clicks on the login button 2. App connects to server and sends the entered login data 3. The server validates the login data and notices that the entered e-mail address is wrong 4. Server sends a wrong e-mail address message to the client 5. App displays the message and gives the possibility to enter the login data again
Exit condition	The user is not logged in.

Use case name	`Login with connection failure`
Entry condition	The app is running on the smartphone
Flow of events	1. Participant enters the data and clicks on the login button 2. App tries to connect to the server 3. App can't connect to the server and displays warning that app is not able to connect to the server. 4. Participant can click on the login button, if he wants to try it again.
Exit condition	The Participant is not logged in.



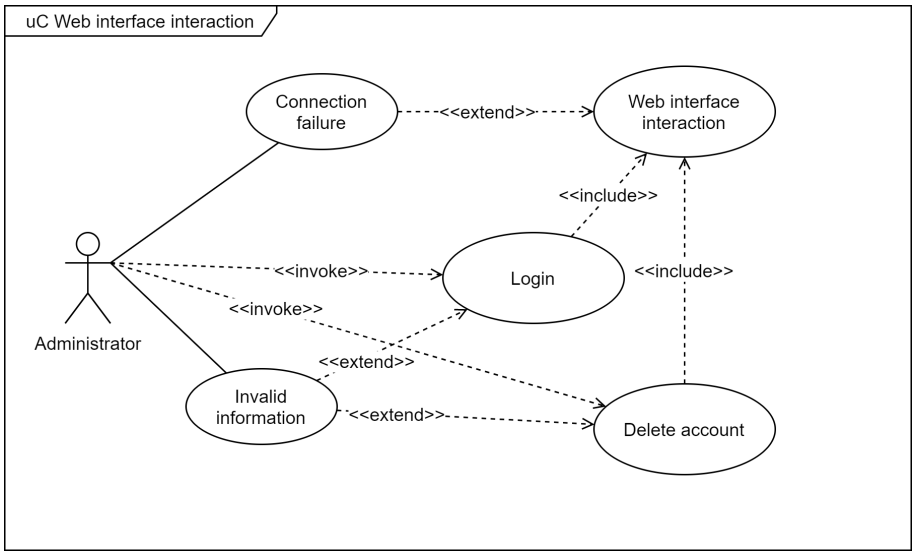
Use case name	`Synchronize with connection failure`
Entry condition	The last notification of the day has passed
Flow of events	1. The app attempts to connect to the server 2. The connection fails 3. No changes are done and the data remains locally saved 4. The app retries automatically committing the next day
Exit condition	No changes are done and the data remains locally saved



Use case name	`Admin login with wrong password`
Entry condition	The Admin has opened the web interface
Flow of events	1. Admin enters the password and clicks on the login button 2. Website sends the password to the server 3. The server validates the password and notices that the password is wrong 4. Server send a wrong password message to the ADMINISTRATOR 5. Website displays the message and gives the possibility to enter the password again
Exit condition	The Admin is not logged in

Use case name	`Web interface interaction with connection failure`
Entry condition	The Admin has opened the web interface
Flow of events	1. ADMINISTRATOR attempts to perform any action on the web interface 2. Website tries to connect to the server 3. Website can't connect to the server and displays warning that it is not able to connect to the server
Exit condition	Website displays the warning

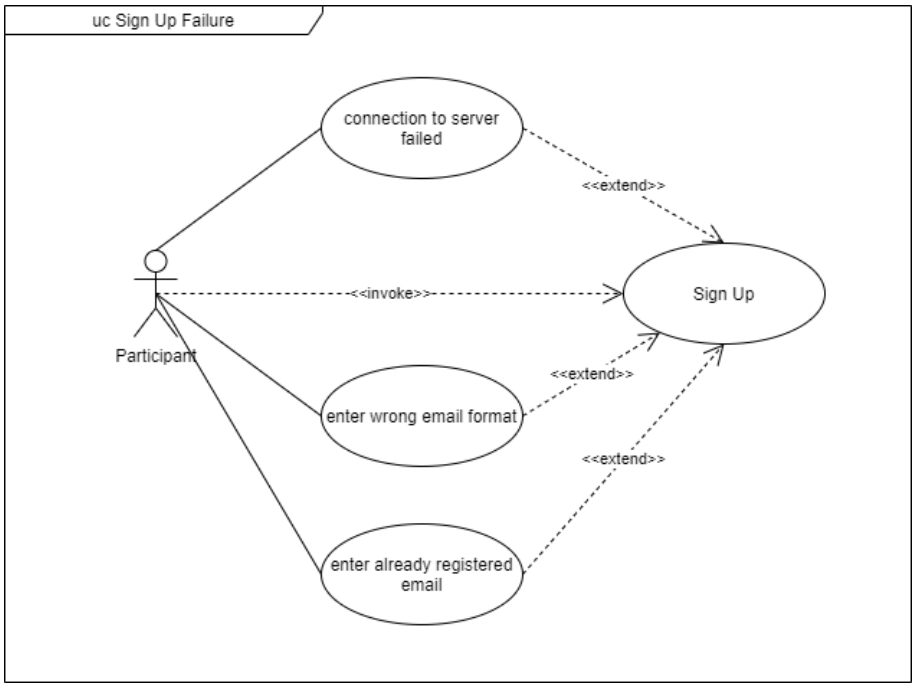
Use case name	`Delete user account with invalid information`
Entry condition	The Admin has opened the web interface and is logged in
Flow of events	1. ADMINISTRATOR clicks "delete user account" button on the web interface 2. ADMINISTRATOR enters invalid account information 3. Website displays warning that the account information is invalid
Exit condition	Website displays the warning, no changes are done



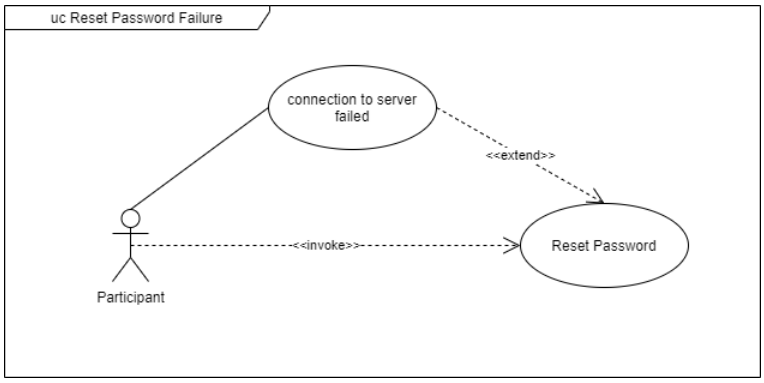
Use case name	`Sign up with wrong email format`
Entry condition	The Participant has the app installed
Flow of events	1. Participant enters email in wrong format 2. The app checks the email for the correct format and notices that the format is not correct 3. The app displays a warning and does not proceed until an email in the correct format is given
Exit condition	The Participant can't create an account

Use case name	`Sign up with already registered email-address`
Entry condition	The Participant has the app installed
Flow of events	1. Participant enters the same e-mail address, he has used before for another account 2. The app displays a warning and does not proceed until a new unused email-address is given.
Exit condition	The Participant can't create an account.

Use case name	`Sign up with connection failure`
Entry condition	The Participant has the app installed
Flow of events	1. Participant enters the credentials for creating an account 2. The app sends the new account details to the server 3. The app can't connect to the server and displays a warning 4. Participant can click on the make account button, if he wants to try again
Exit condition	The Participant can't create an account

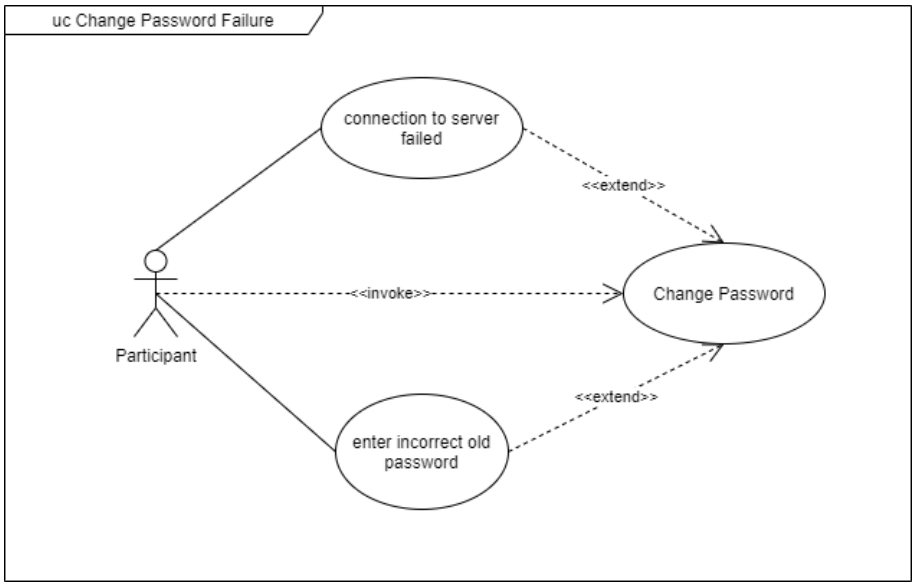


Use case name	`Reset password with connection failure`
Entry condition	The Participant has the app installed, an account and is not logged in
Flow of events	1. Participant chooses to log in 2. Participant clicks "reset password" button 3. Participant enters an email address 4. App tries to send data to the server 5. App can't connect to the server and displays a warning 6. Participant can click on the "reset password" button if he wants to try again
Exit condition	The Participant has not reset his password



Use case name	`Change password with connection failure`
Entry condition	The Participant has the app installed, an account and is logged in
Flow of events	1. Participant clicks "change password" button 2. Participant enters old password and a new password 3. App tries to send data to the server 4. App can't connect to the server and displays a warning 5. Participant can click on the "change password" button if he wants to try again
Exit condition	The Participant has not changed his password

Use case name	`Change password with wrong old password failure`
Entry condition	The Participant has the app installed, an account and is logged in
Flow of events	1. Participant clicks "change password" button 2. Participant enters incorrect old password and a new password 3. App sends data to the server 4. Server validates data and notices that old password is wrong 5. App displays warning, that old password is incorrect 6. Participant can click on the "change password" button if he wants to try again
Exit condition	The Participant has not changed his password

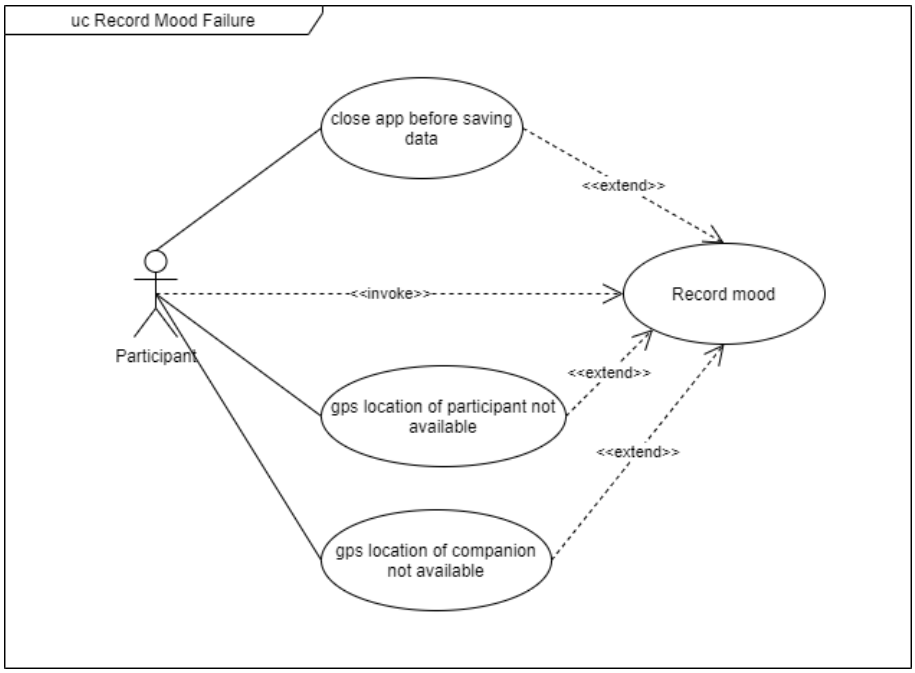


Use case name	`App gets closed during voluntary record mood`
Entry condition	The Participant is logged in
Flow of events	1. Participant clicks "record mood" button 2. Participant records his mood 3. Participant closes the app before saving the mood recording
Exit condition	The mood was not recorded

Use case name	`App gets closed during record mood by notification`
Entry condition	The Participant is logged in
Flow of events	1. Participant clicks on the notification 2. Participant records his mood 3. Participant closes the app before saving the mood recording
Exit condition	The mood was not recorded, the notification is still displayed on-screen

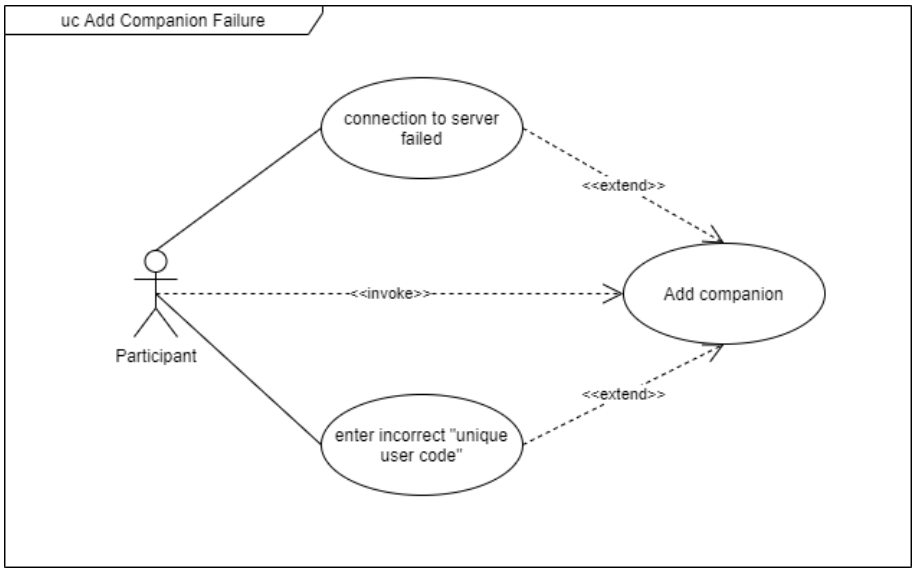
Use case name	`App can't get companion's GPS location during record mood`
Entry condition	The Participant is logged in
Flow of events	1. Participant clicks on the notification 2. Participant records his mood 3. App doesn't get GPS location of a single or multiple companions
Exit condition	-1 is noted as distance between Participant and Companions whose location data could not be fetched

Use case name	`App can't get participant's GPS location during record mood`
Entry condition	The Participant is logged in
Flow of events	1. Participant clicks on the notification 2. Participant records his mood 3. App doesn't get GPS location of the Participant
Exit condition	-1 is noted as distance between Participant and all Companions

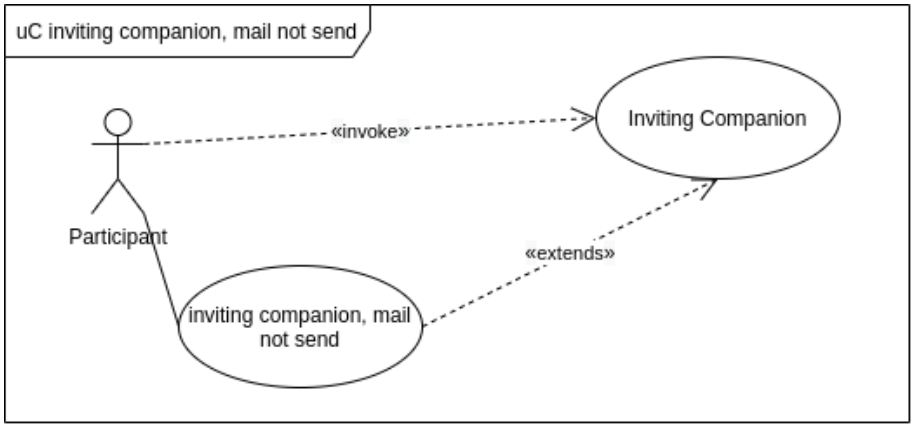


Use case name	`Adding companion with connection failure`
Entry condition	The Participant is logged in
Flow of events	1. Participant clicks on the "add Companion" button 2. Participant enters Companion code and relationship 3. App tries to send data to the server 4. App can't connect to server 5. App displays warning that app is not able to connect to server 6. Participant can click on the "add Companion" button, if he wants to try it again
Exit condition	Companion not added

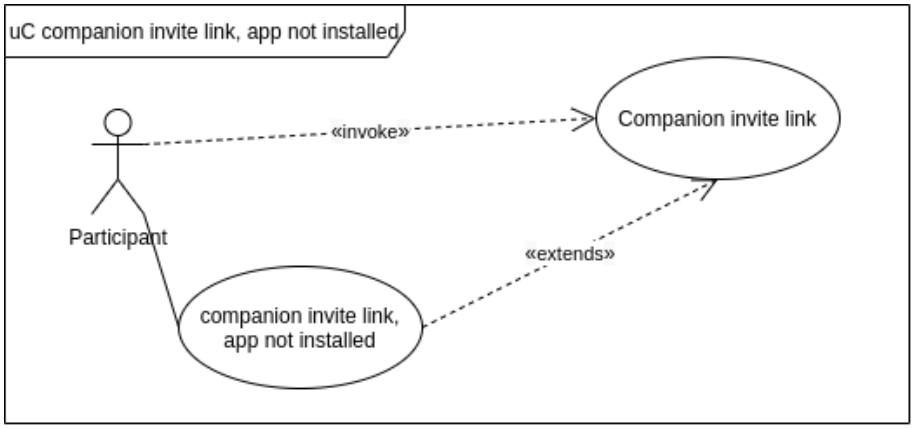
Use case name	`Adding companion with wrong "unique user code"`
Entry condition	The Participant is logged in
Flow of events	1. Participant clicks on the "add Companion" button 2. Participant enters incorrect "unique user code" and relationship 3. App connects to the server and sends the data 4. The server validates the data and notices that the entered "unique user code" is wrong 5. App displays warning that unique user code is incorrect 6. Participant can click on the "add Companion" button, if he wants to try it again
Exit condition	Companion not added



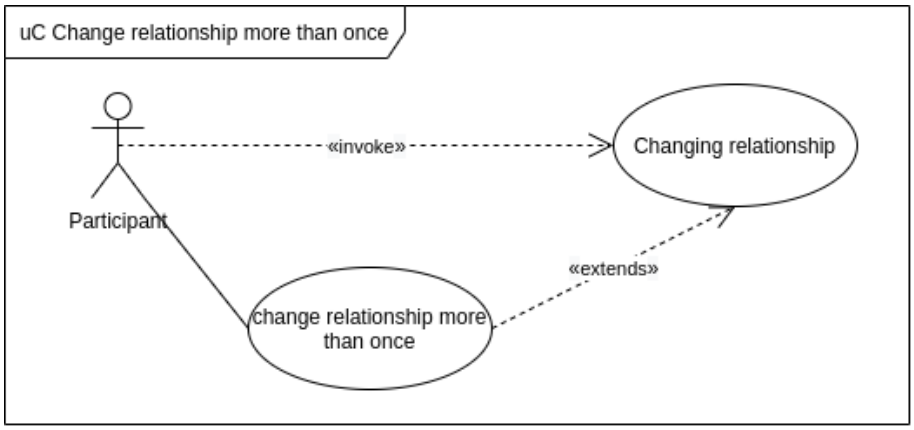
Use case name	`Inviting companion, mail not sent`
Entry condition	The Participant is logged in
Flow of events	1. Participant clicks on the "invite Companion" button 2. Participant enters relationship with invited person 3. App prompts user's email app with a message draft containing the generated links 4. The user doesn't send the mail (Cancelled or no connection)
Exit condition	Invitation not sent



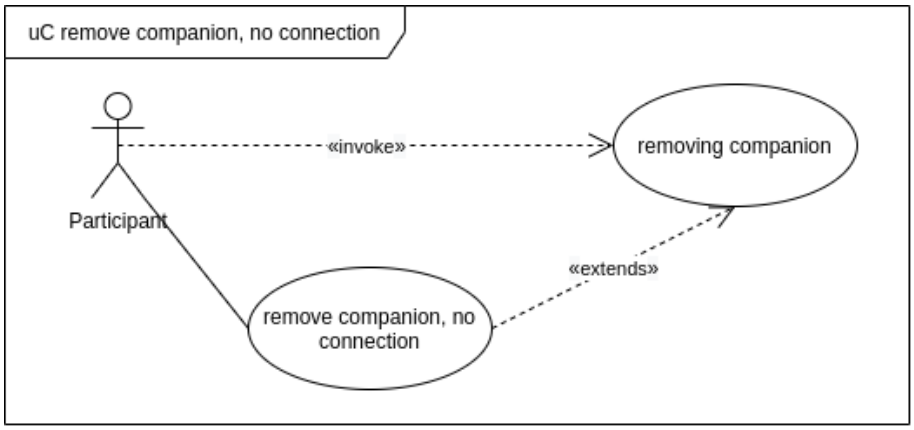
Use case name	`Companion invite link, app not installed`
Entry condition	The Participant has not installed the app by now
Flow of events	1. Participant clicks on the "add Companion" link in the invitation email they received 2. The app isn't installed, so the link is opened in the browser 3. The browser redirects the participant to the Google Play page of the app
Exit condition	The participant lands on the Google Play page of the app



Use case name	`Change relationship more than once`
Entry condition	The Participant has the app installed, and changed the relationship to that Companion already one time
Flow of events	1. Participant clicks on the "change Companion" button on the companion inside the app 2. App displays warning, that the Participant already changed the relationship with this companion
Exit condition	Relationship is not changed



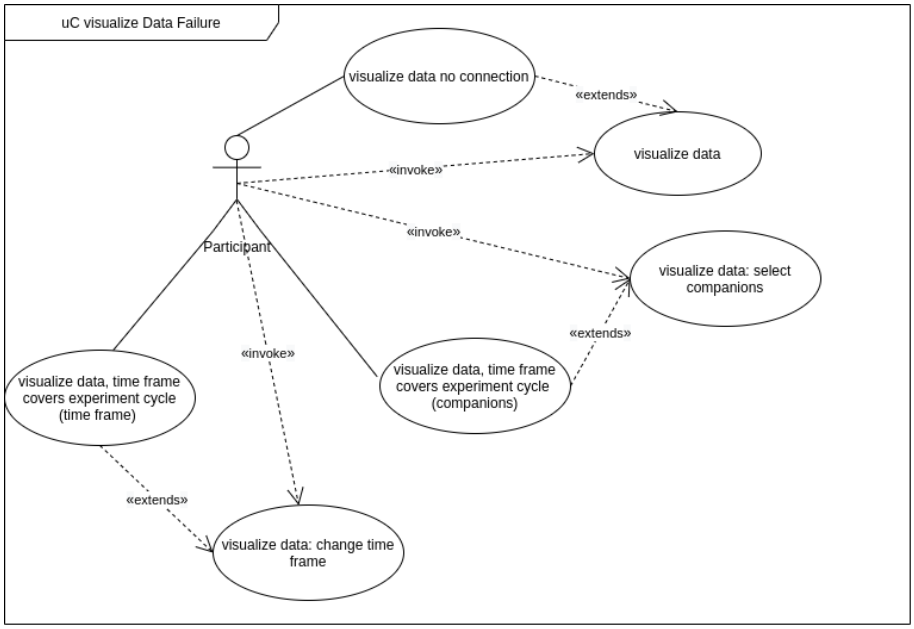
Use case name	`Remove companion, no connection`
Entry condition	The participant is logged in
Flow of events	1. Participant clicks on the "remove Companion" button on the companion inside the app 2. App attempts to connect to the server 3. App cannot connect to server 4. App displays an error message
Exit condition	Companion is not removed, participant is shown a warning



Use case name	`Visualize data: No connection`
Entry condition	The Participant has the app installed and is viewing the visualization screen
Flow of events	1. App attempts to load visualization data from server 2. App cannot connect to server 3. App displays an error message
Exit condition	Data is not visualized, participant is shown a warning

Use case name	`Visualize data: Time frame covers experiment cycle (Companions)`
Entry condition	The Participant has the app installed and is viewing the visualization screen
Flow of events	1. Participant attempts to change the selection of companions to visualize 2. At least one of the selected companions' active experiment cycle covers the currently selected time frame 3. App displays warning, that some of the selected companions cannot be visualized on the time frame
Exit condition	Data of companions whose experiment cycle cover the selected time frame are not visualized, participant is shown a warning

Use case name	`Visualize data: Time frame covers experiment cycle (Time frame)`
Entry condition	The Participant has the app installed and is viewing the visualization screen
Flow of events	1. Participant attempts to change the selected time frame 2. At least one of the selected companions' active experiment cycle covers the desired time frame 3. App displays warning, that some of the selected companions cannot be visualized on the chosen time frame
Exit condition	Visualization is not changed, participant is shown a warning



,

4 Glossary

RMI
: *Remote method invocation*: RMI resembles a method call on the server being invoked by a client system.

Admin/Administrator
: The person who will run the experiment and is able to access the data generated by the application and change its settings.

Mood data
: Mood data refers to the data input by the user, including mood level, relaxation level, near companions and special situations.

GPS
: *Global Positioning System* is used to determine the users position.

One-time consent form
: The one time constent form can be edited by the admin and has to be accepted once by every users before he can start use the app.

CSV
: *Comma-Separated Values*. A comma separated value (csv) file with all the data of each participant. Each row stands for one single user. Each row includes: an identifier that links the person to a consent form, the answers of the user's questionnaires, each relationship established (with ID to the other user and timestamp on when relationship started and/or ended), all entry log info (timestamp, voluntary flag), in case it was elicited: notification time, and data input time; mood, stress level, companions, special situations and GPS.
If a user didn't click on a notification,the entry only contains a timestamp,notification label,notification time, and a minus 1.

Experiment cycle
: After establishing a relation with a companion, the *experiment cycle* specifies the time frame which has to elapse before the corresponsive participants can view each other's data for said time frame.

Participant
: With participant, we refer to the people who will be partaking in the experiment, the users of our application.

qstnre.
: *Questionnaire*

mng.
: *Manage*

Logged in/out
: A user can only log in with his credentials, this assures that only people that are supposed to participate in the experiment are able to do so. A user can log out at any time, but will not be able to use the application.

Voluntarity flag
: Indicates whether the user entered the mood afte receiving a notification prompting them to record their mood or on their own.

User ID
: Every user will have a unique *User ID*, usually a string of letters and numbers which allows a program to identify him easily, even if there are multiple users with the same names, etc.

Part II

Documentation of Group Work

5 General organization

- Project Manager - Niklas Maier
- Customer Relationship Officer - Johanna Bell
- Documentation Lead - Luca Bosch
- Technical Lead - Sebastian Schwarz
- Repository Lead - Simon Vogelbacher
- Quality Assurance Manager - Yasmin Hoffmann

6 Individual contributions of each team member

6.1 Johanna Bell

- Sections 1, 2, 3.1, 3.2, 3.3, 3.4, 3.7
- Introduction, Existing System, Overview, Component segmentation and description of interface, Hardware/Software mapping, Management of persistent of data, boundary conditions

6.2 Luca Bosch

- Sections 1, 2, 3.1, 3.4, 3.5, 3.6, 3.7
- Introduction, Existing System, Overview, Management of persistent data, Access rights, Global control flow, Boundary condiditions, Documentation of group work

6.3 Niklas Maier

- Sections 1, 2, 3.1, 3.2, 3.3, 3.4, 3.7, 4
- Introduction, Existing System, Overview, Component segmentation and description of interface, Hardware/Software mapping, Management of persistent of data, boundary conditions, Glossary

6.4 Sebastian Schwarz

- Sections 1, 3.2, 3.3, 3.4, 3.6, 3.7
- Introduction, component segmentation, hardware/software mapping, management of persistent data, global control flow, boundary conditions

6.5 Simon Vogelbacher

- Sections 1, 2, 3.1, 3.2, 3.4, 3.7, 4
- Introduction, Existing System, Overview, Component segmentation and description of interface, Management of persistent data, Boundary conditions, Glossary

6.6 Yasmin Hoffman

- Sections 1,2,3.1,3.2,3.3,3.4,3.7,4
- Introduction, Existing System, Overview, Component segmentation and description of interface, Management of persistent data, Boundary Conditions, Global Control Flow

7 Protocols

7.1 Protocol on 28. May 2020

Datum: 28. May 2020 **Zeit:** 11:00-17:00

- Discussed Topics
  - We discussed the group feedback on D1
  - We considered what we wanted to add, change or delete
  - We talked about how we will proceed with the new assignment of D2a
  - Since we discussed the time for the advisor meeting tomorrow in the chat already, we do not need to discuss it again.
- Conclusions
  - We will edit the document together during this meeting until everyone is pleased with the outcome
  - We will specifically focus on the requirements, so they meet the IEEE standards
  - We will update the work protocol with the information, that we changed the final D1 document again
  - We will create a plan for the fourth week in the work plan to organize D2a right now
  - We will meet again on 1.6. 11:00

7.2 Protocol on 01. June 2020

Datum: 01. June 2020 **Zeit:** 11:00-17:00

- Discussed Topics
  - We talked about the work done on the weekend (Task 1.1-3.1) and found some minor mistakes
  - We searched for an online software where we could edit charts together
  - We looked for a timeslot where everyone had time to work on D2a.
- Conclusions
  - We will meet tomorrow, 2.6. 11:00 and try to collaboratively work on the next deliverable

7.3 Protocol on 09. June 2020

Datum: 09. June 2020 **Zeit:** 13:00-14:20

- Discussed Topics
  - We finished D2a and updated the work protocol together
  - We discussed the deadline change option mentioned in the email we all got
  - We considered changing some of the diagrams after recieving new information in Software Engineering
- Conclusions
  - We decided to consult our advisor about the deadline, since we just needed 2 weeks more time for the final deliverable, but the current ones are okay like they are
  - We decided to change the easily editable elements in D2a and correct the rest during the course of the week

7.4 Protocol on 11. June 2020

Datum: 11. June 2020 **Zeit:** 11:00-12:15

- Discussed Topics
  - After receiving a new email, we sat toghether again and tried to add some final text to the D2a deliverable
  - We discussed how we are going to adhere to the final deadline while also studying enough for the exams at the same time
  - We talked about what time would be the best for our upcoming advisor meeting
- Conclusions
  - The time for the meeting is not relevant, since everyone has time all day long, we will let the advisor decide

7.5 Protocol on 18. June 2020

Datum: 11. June 2020 **Zeit:** 15:00-15:30

- Discussed Topics
  - We figured out together how to work on the new deliverable, reading the script to understand what has to be done in the different stages, since there was no example this time
  - We created a very short work plan
  - We discussed how much time the steps in 'Instruction 4' takes, so we avoid working when we should be studying for exams
- Conclusions
  - We could not make a very detailed work plan, since we are still waiting on a feedback on the last deliverable, and since the next one is in large parts based on this one, we didn't want to build on incorrect diagrams.

7.6 Protocol on 22. June 2020

Datum: 11. June 2020 **Zeit:** 11:00-15:30

- Discussed Topics
  - We tried to work out the weekly work plan, but decided to review the previous deliverable again since we just got an email with feedback from our advisor
  - We discussed together which diagrams we want to change and what changes we want to do.
- Conclusions
  - We decided to add some new diagrams to the document, specifically new sequence diagrams to cover use-cases we did not yet cover with the existing ones.

Furthermore we had a lot of spontaneous meetings creating this deliverable, at least partly due to the corona situation and the different lectures of the team members. Though we have not documented these small meetings since there were only 2 or 3 people involved for short periods of time, we occasionally shared our conclusions in a text chat. Almost always they were simply new appointments for meeting again, so not really relevant.