Deliverable 3: Final Document

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

August 14, 2020

Advised by Till Niese

Contents

List of Figures

Part I

Change History

Date	Version	Revision Class	Description
14.08.2020	1.1	Minor	changed Sign-Up, Change password and Reset Password sequence diagrams from D2a
14.08.2020	1.1	Minor	added requirement FR22: Users can delete their account.

Annotation from our group:

In the final implementation phase of our project we realised, that the code structure and implemented app were different from what we originally had planned. This was because nobody of our team knew how the implementation of an android app exactly would look like in the end. Because of time limitations in the end we weren't able to adapt all the contents of our documents to the final code. Therefore our class diagrams and pseudo code are not exactly representing the structure of our code.
The requirements and use cases should still reflect how our app works and we were able to implement most of the requirements we made in the beginning. Still there are some requirements / features which we weren't able to get to work in the end, mostly just because we ran out of time. In the following we want to give a overview of the missing features.

The biggest feature we couldn't implement is the GPS feature for calculating the distance between the user and their companions (FR 50). We have not found a way to implement this feature while maintaining the integrity of the data, our code and the security of our users. Furthermore, our app doesn't properly work when there's no internet connection, because we had no time to figure out how to store and read files locally (FR 60). Recorded mood data is uploaded immediately and none of the project data (settings) and user data (companions, etc) is stored locally.
Other missing features are the user's profile pic (FR 18), the feature of being able to change the companion relationship once (FR 19), and the visualisation of up to 5 companions mood data (FR 78) after the experiment cycle and the function of the experiment cycle itself (FR 93). We have specifically decided not to work on the visualization of the 5 companions because we as a team would find a graph filled with this many curves extremely confusing. Another small feature which is missing is about the notifications. We only send notifications to each user which are then randomly made in the application of the user. We can't send the same random notification time to all the users.
Bonus tasks: We implemented a part of bonus A, so far that the admin can enter a user id and see the user's visualization as it was described in 1.4 in the Project Description.
No other bonus tasks were implemented.

Part II

Requirement Analysis

1 Introduction

1.1 Intention of system

The goal of this project is to track the mood of individuals and groups (couples, co-workers, family members, etc.) to evaluate them from a scientific point of view. The final aim of the developed software will be to collect data about how individuals' moods are affected by the presence and moods of other individuals in a group (emotional contagion).

1.2 Extensiveness of system

The project consists of an android application for the users whose mood will be monitored, an interactive website for the client who will be able to access the data that the users enter, and those two systems will be connected via a web server. The functionalities of the app should be to provide an easy way for the users to track their mood and stress level, an overview and visualization of past activity and the ability to share their data with companions. The system should remind users regularly by making use of notifications and provide some basic settings like setting a profile picture or username. The application furthermore needs to collect the users data and upload it to a server.
The web application is only intended to be used by the client to download and manage the data from all participants and change settings that will then be synchronized with the mobile application (for instance the experimental cycle). Optional features, that are not necessary to successfully complete the project are additional visualization features for the website as well as the option that allows the client to change the frequency and content of questions for the individual users.

1.3 Goals and success criteria of project

- May 22^{nd}, 2020 (Deliverable 1)
  Requirement Analysis
- June 12^{th}, 2020 (Deliverable 2a)
  Design Document: Software Architecture
- July 3^{rd}, 2020 (Deliverable 2b)
  Design Document: Detailed Design
- July 31^{st}, 2020 (Deliverable 3)
  Implementation

The project is successfully completed if it passes the specified customer acceptance tests.

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. User 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

- Stress/Relaxation level
  The level of stress/relaxation that a user is experiencing. Measured on a five point scale from "very stressed" to "very relaxed'

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

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 Proposed System

3.1 Overview

The software which has to be developed is a mood tracker in form of an android application, which is intended to capture the feelings of the users in order to evaluate them later from a scientific point of view. For this purpose, the feelings of several users are to be recorded in order to investigate the influence of the feelings of each user on the other users. In this way, insights shall be gained into how the feelings of the users are influenced by other people (and their feelings).

3.2 Functional Requirements

3.2.1 Required Functions

/FR 10/
Users are able to register to the system by providing a valid e-mail address and a password

/FR 11/
Users can login with the created account

/FR 12/
Users can change their password

/FR 13/
Users must fill-in a "one-time consent form" when first signing into the app

/FR 14/
When the user is logged in and starts the app, the calendar screen is opened.

/FR 15/
Users can send "unique user codes" to other people who have the app in order to add them as couple, friend, family, coworker, roommate, other

/FR 16/
Users can invite other people as couple, friend, family, coworker, roommate, other who don't have the app, via e-mail

/FR 17/
Users have to declare their relationship (couple, friend, family, coworker, roommate, other) when adding a friend via "unique user code" or via email

/FR 18/
Users can optionally choose a nickname and profile picture

/FR 19/
Users can change their relationship with companions up to one time each

/FR 20/
Users can click on days in the calendar to visualize the mood and stress level, companions

and special situations they recorded on that day

/FR 21/
Users can click on a button with the calendar where they can enter their current mood, stress level, companions and special situations (which needs to have a timestamp of the recorded time)

/FR 21/
Users can delete their account.

/FR 30/
The app must notify the user randomly, multiple times a day to record their current mood, stress level, companions and special situations

/FR 31/
The app must align with the users current status for the notification settings

/FR 40/
Users can record mood ratings, stress/relaxtion level ratings, which companions they met and special situations they're experiencing throughout the day

/FR 41/
The app has to display the mood recording screen when the user opens the app following a notification

/FR 42/
Users have to enter a mood rating inside the mood recording screen

/FR 43/
The app has to display a stress/relaxation screen after the mood recording where the user has to declare a level of stress on a five-point scale

/FR 44/
The app has to display the companions screen after the stress / relaxation screen

/FR 45/
Users must add a nearby companion, companions they've met in the last half hour, or none (when they haven't met anyone in the last hour) in the companion screen

/FR 46/
The app has to display a special situations recording screen after the companions recording screen in case of the first recording of the day that was issued by a notification

/FR 47/
The app has to display the special situations recording screen every time when the user voluntarily records his mood

/FR 48/
Users can choose from a preset of special situations, adding positive or negative connotations

/FR 50/
The system shall calculate the distance between the user and all their contacts through GPS each time the user records mood data

/FR 60/
The recorded mood data needs to be stored locally on the user's android device first

/FR 61/
The mood data needs to have a time stamp, and information about if it was recorded voluntarily or by notification

/FR 62/
The locally stored recorded mood data should be synchronized once a day (after the last reminder)

/FR 70/
The visualization for mood and stress changes within a day is accessible from the calendar screen

/FR 71/
The visualization for mood and stress corresponds to a choosable time window

/FR 72/
The visualization should have a x-axis which represents time

/FR 73/
The visualization should have a y-axis which represents mood/stress level

/FR 74/
The visualization tool should allow filters that can be turned off and on

/FR 75/
The visualization tool should have a filter to show negative or positive events of the user

/FR 76/
The visualization tool should have a filter to display meetings with other people

/FR 77/
The visualization tool should have a filter to display mood or stress levels

/FR 78/
The visualization tool allows to visualize data of up to 5 companions, given the selected time window does not cover the current experiment cycle

/FR 80/
The app should have a help page to explain the different functionalities of the app to the user

/FR 90/
The administrator can log-in with given credentials to the admin website

/FR 91/
The administrator can change the settings which are stored in the server (including administrator password, notification timing parameters, length of experimental cycle, consent form text)

/FR 92/
The administrator has the option to change if and how many times a day the users get notifications

/FR 93/
The administrator has the option to change the experiment cycle duration in days

/FR 94/
The administrator has the option to change the one time consent form

/FR 95/
The system should store a history of the changed consent form texts

/FR 96/
Users have to be signed to a version of the consent form text which they accepted

/FR 100/
The administrator can change the settings of the firebase in the web interface

/FR 101/
The administrator can download recorded user data of all users through the admin website

/FR 102/
The data for download can be selected in range of dates

/FR 103/
The data can be downloaded all in once (all available data)

/FR 104/
The administrator can archive recorded user data in the web interface

/FR 110/
The downloaded data is contained in a single CSV file with one row per user data

/FR 111/
The CSV file consists of rows which don't contain user names but consent form identifiers on which the users agreed

/FR 112/
The CSV file consists of rows which contain questionnaire answers, relationships established (with user ID and timestamp)

/FR 113/
The CSV file consists of rows which contain data entered along notification and enter timestamps and companions detected by GPS.

/FR 114/
The CSV file row consists of a flag "voluntary" and indicator value -1 when the user omits a notification.

3.2.2 Desired Functions

/DF 10/

Users can filter the visualization for relationships, variables of the questionnaire, personality, emotion contagion score, special situations, time of the day or a combination of several filters

/DF 20/

The administrator can visualize the users data in the website using different filters

/DF 30/

The administrator can change, edit or erase the in-app questions which are displayed to the users, to trigger which kind of information users log.

/DF 40/
The admin can remove questions, add new questions, and select the order in which they appear

/DF 50/
The administrator can change if all questions appear on every notification or just once a day or in special situations

/DF 60/
The administrator can edit/add/erase questionnaires that need to be answered by the users upon the first log-in to the app

/DF 61/
The admin can select if the user has to sign the questionnaires at sign up or at a specific point in time

/DF 70/
The data transmission from the apps to the server is encrypted

3.3 Non-functional Requirements

3.3.1 Reliability

- The notifications and information logging should be extremely reliable and robust while the user is offline, without battery, etc.

3.3.2 Usability

- The information logging and visualization tool should be intuitive and easy to use for the user
- The application should be self explanatory and accessible to everyone
- The application should provide the user with a visually pleasing experience
- The application should provide a quick and intuitive navigation to avoid annoyance
- The user should personally benefit from using the app in order to get motivated to track his moods more regularly

- The user should be able to gain knowledge from the provided visualizations

3.3.3 Efficiency

- The app should run smoothly and quickly, to avoid burdening the users

3.3.4 Performance

- The system should support at least 100 users at the same time

3.4 System models

3.4.1 Scenarios

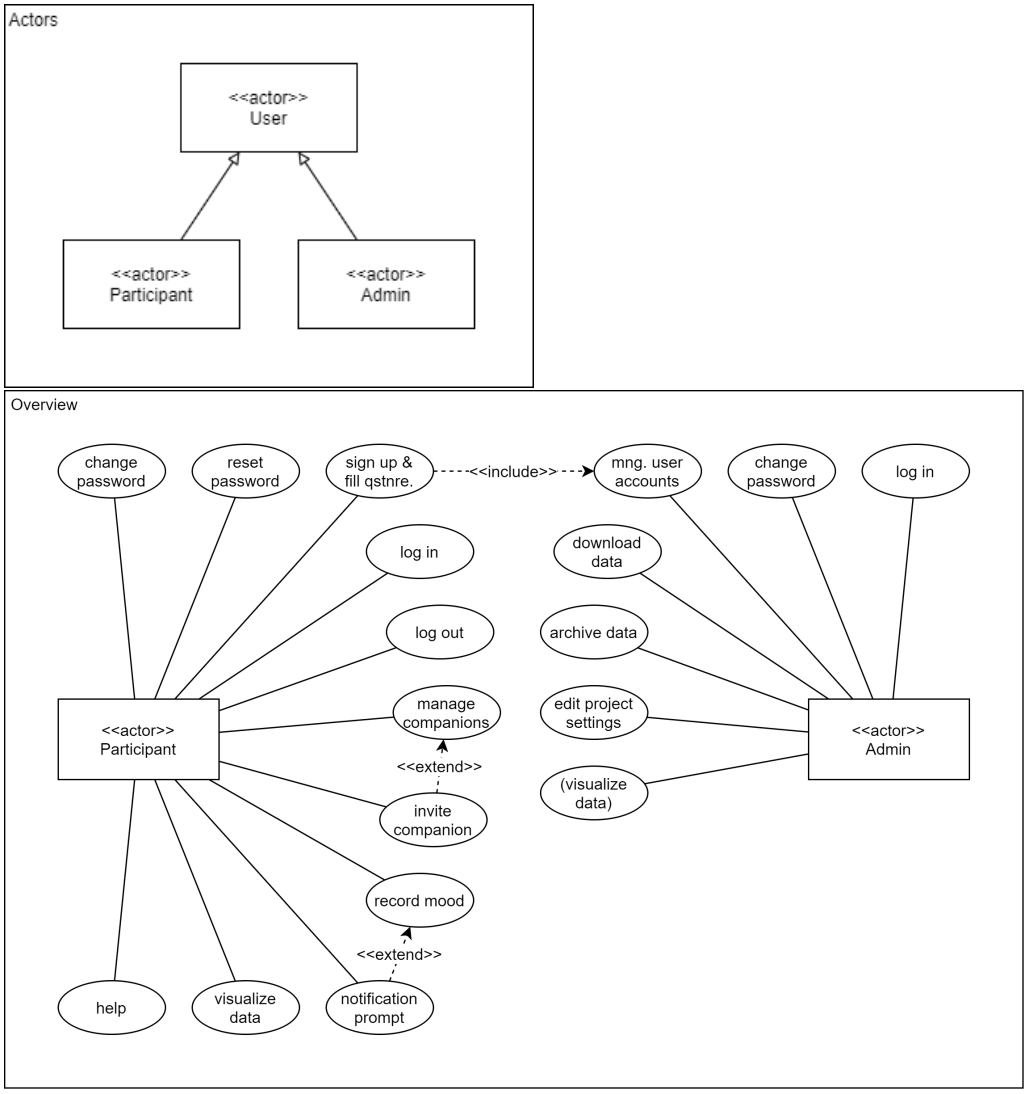
3.4.1.1 General sample scenario

Actors: Admin, 3 participants (friends)

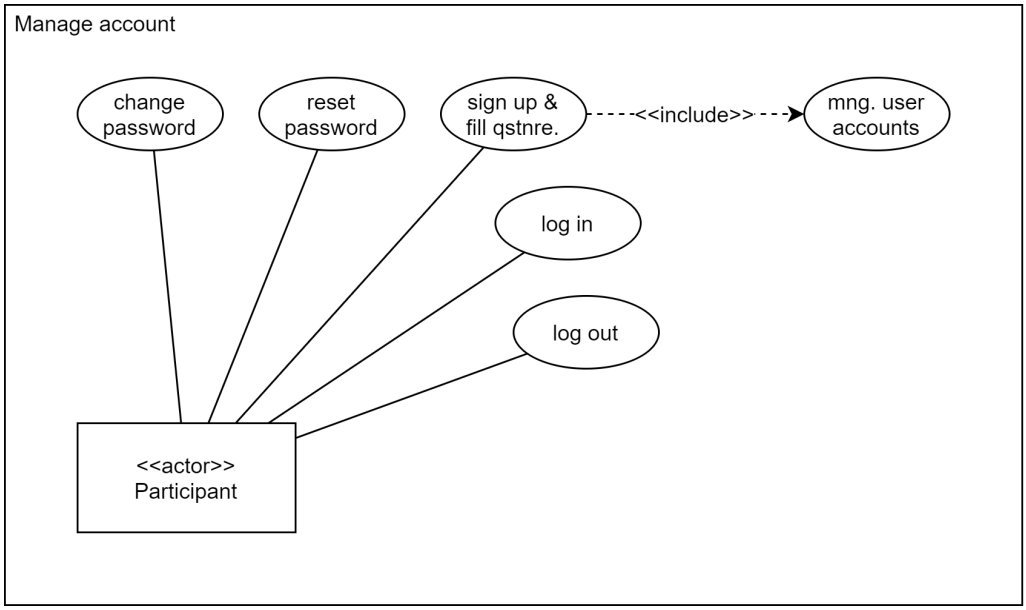
Events:
Participant A downloads the app and creates an account.
Participant A then invites participants B and C via Email through the app, who download it and create an account each.
Participant A shares their unique user code with both B and C, who each add A as a friend to their companions.
Over a couple of days, all participants fill the mood prompt whenever they receive notifications.
After two weeks, the admin downloads the recorded data from the server.
The admin analyses the gathered data to examine how the participants' moods develop depending on another.

3.4.2 Use cases

3.4.2.1 Use cases overview



3.4.2.2 Participant account management



Use case name	Sign up
Actors	Participant
Starting condition	The app is installed on the participant's phone and the participant has yet to create an account.
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1. The participant opens the app for the first time and is prompted to log in or create an account. 2. The participant chooses to create an account and is presented a consent form. 3. Upon accepting, the participant is prompted for account details. 4. Upon entering valid email and password, the participant is asked to set an optional nickname and profile picture. 5. Upon completion, the participant is presented questionnaires they have to fill in. The version of the consent form on which the participant accepts is stored in the server. 6. Upon finishing, the app synchronizes data with the server and the participant is logged in.
Completion condition	The registration is successful and the participant is now logged in.

Use case name	Sign up failure
Actors	Participant
Starting condition	The app is installed on the participant's phone and the participant has yet to create an account.
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1. The participant opens the app for the first time and is prompted to log in or create an account. 2. The participant chooses to create an account and is presented a consent form. 3. Upon accepting, the participant is prompted for account details. 4. Upon filling in invalid credentials, the participant is shown a warning.
Completion condition	No changes are done and the participant is shown a warning.

Use case name	Log in
Actors	Participant
Starting condition	The participant has an account but is not logged in. The app is installed on their phone.
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1. The participant opens the app and is prompted to log in or create an account. 2. The participant chooses to log in and is presented a log in form. 3. Upon filling in valid credentials, the app compares provided data with server data. 4. The data is validated and the participant is now logged in.
Completion condition	Logging in is successful.

Use case name	Log in failure
Actors	Participant
Starting condition	The participant has an account but is not logged in. The app is installed on their phone.
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1. The participant opens the app and is prompted to log in or create an account. 2. The participant chooses to log in and is presented a log in form. 3. Upon filling in invalid credentials, the app compares provided data with server data. 4. The data is refused and the participant is shown a warning.
Completion condition	No changes are done and the participant is shown a warning.

Use case name	Open app
Actors	Participant
Starting condition	The participant is logged in and opens the app .
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1.The participant opens the app. 2.The app shows the calender screen.
Completion condition	The participant is shown the calender screen.

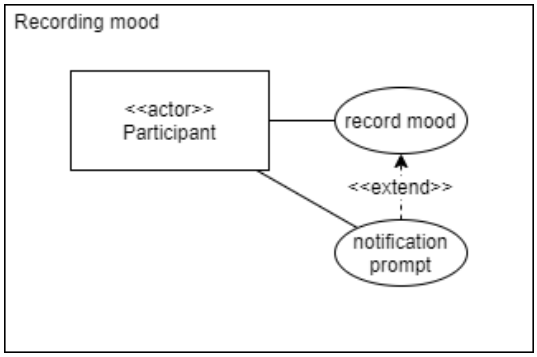
Use case name	Reset password
Actors	Participant
Starting condition	The participant has an account but is not logged in. The app is installed on their phone.
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1. The participant opens the app and is prompted to log in or create an account. 2. The participant chooses to log in and is presented a log in form. 3. The participant uses the "reset password" option on the login screen and is prompted to enter their email address. 4. Upon entering, the app requests the server to send a password reset email to the entered address, given it's registered.
Completion condition	The provided email address is valid and a reset link is sent.

Use case name	Log out
Actors	Participant
Starting condition	The participant has an account and is logged in.
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1. The participant clicks the log out button inside the app. 2. The app removes the saved account data and the participant is logged out.
Completion condition	The participant has logged out successfully.

Use case name	Change password
Actors	Participant
Starting condition	The participant has an account and is logged in.
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1. The participant clicks the change password button inside the app. 2. The participant is presented a password changing form, asking for the current and a new password. 3. The participant enters the correct current password and a new password. 4. The data is validated and the password changes are reflected on the server.
Completion condition	The password is changed and the new password is saved on the server.

Use case name	Change password failure
Actors	Participant
Starting condition	The participant has an account and is logged in.
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1. The participant clicks the change password button inside the app. 2. The participant is presented a password changing form, asking for the current and a new password. 3. The participant enters an incorrect current password and a new password. 4. The data is refused and the participant is shown a warning.
Completion condition	No changes are done and the participant is shown a warning.

3.4.2.3 Recording mood



Use case name	Record mood
Actors	Participant
Starting condition	The participant has an account and is logged in.
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1. The participant clicks the record mood button (near calendar) either inside the app or on a notification. 2. The participant is presented a flow of questions(mood recording screen): 2.1. The participant is asked to enter their mood on a scale of 1-5. 2.2. The participant is asked to enter their relaxation on a scale of 1-5. 2.3 The participant is asked to select all nearby companions, companions they have met in the last half hour, or none (when they haven't met anyone in the last hour). 2.4. The participant is asked to select from a preset of special situations and state their positivity. 3. The app fires a GPS location collection for the participant and all their companions and records the distance between them 4. The app saves the recorded data locally with timestamp and voluntarity flag.
Completion condition	The recorded data is saved on the device.

Use case name	Synchronize mood data
Actors	Participant
Starting condition	The participant has an account and is logged in. They have unsynchronized data on their device.
Quality requirements	The process runs automatically and reliably.
Events	1. After the last notification of a day, the app automatically attempts to synchronize data with the server. 2. The app establishes a connection with the server and transmits the locally saved data. 3. The app marks the data as synchronized locally.
Completion condition	The locally recorded data is saved on the server and the local storage is updated.

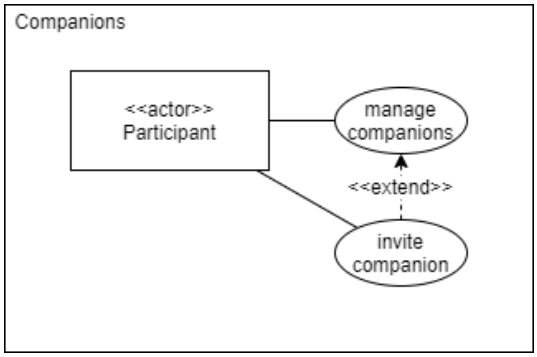
Use case name	Synchronize mood data failure
Actors	Participant
Starting condition	The participant has an account and is logged in. They have unsynchronized data on their device.
Quality requirements	The process runs automatically and reliably.
Events	1. After the last notification of a day, the app automatically attempts to synchronize data with the server. 2. The app cannot establish a connection with the server. 3. No changes are done, the app will attempt to synchronize the data on the next day.
Completion condition	No changes are done.

Use case name	Notification prompt accepted
Actors	Participant
Starting condition	The participant has an account and is logged in. The notification settings allow for sending messages to the user. The app is installed on their phone.
Quality requirements	The notification is well visible.
Events	1. The participant receives a notification prompting them to record their mood. 2. The participant accepts the prompt and is taken to the *Record mood* case. 3. If this is not the first notification of the day, the situations screen in the Record mood use case (2.4.) is skipped.
Completion condition	The recorded data is saved on the device.

Use case name	Notification prompt ignored
Actors	Participant
Starting condition	The participant has an account and is logged in. The notification settings allow for sending messages to the user. The app is installed on their phone.
Quality requirements	The notification is well visible.
Events	1. The participant receives a notification prompting them to record their mood. 2. The participant ignores the prompt. 3. The notification remains on-screen until the next notification appears. 4. A data entry is saved for a missed notification.
Completion condition	The entry is saved on the device.

Use case name	Multiple notification prompts ignored
Actors	Participant
Starting condition	The participant has an account and is logged in. The app is installed on their phone.
Quality requirements	The notification is well visible.
Events	1. The participant has ignored or missed all notifications of a day. 2. The participant is sent an email reminder.
Completion condition	The participant is sent an email.

3.4.2.4 Companions



Use case name	Adding companion
Actors	Participant
Starting condition	The participant has an account and is logged in. The participant knows the unique user code of their companion.
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1. The participant clicks the add companion button inside the app. 2. The participant is prompted for unique user code and relationship. 3. Upon filling in with a valid code, the user has to declare a relationship to the new companion, the companion is added to their list, and the data along with a timestamp is saved on the server.
Completion condition	The companion data is saved on the server.

Use case name	Adding companion failure
Actors	Participant
Starting condition	The participant has an account and is logged in. The participant knows the unique user code of their companion.
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1. The participant clicks the add companion button inside the app. 2. The participant is prompted for unique user code and relationship. 3. Upon filling in with an invalid code, the participant is shown a warning.
Completion condition	No changes are done and the participant is shown a warning.

Use case name	Inviting companion
Actors	Participant
Starting condition	The participant has an account and is logged in.
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1. The participant clicks the invite companion button inside the app. 2. The participant is asked for the relationship to the invited person. 3. Upon selecting, the participant has the option to send a message with generated links (app download link and add companion link) via email.
Completion condition	The email is sent.

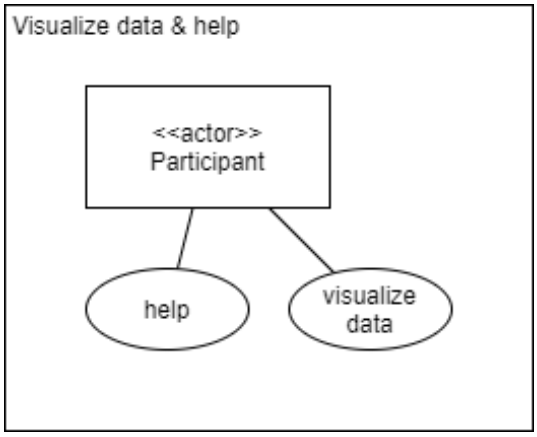
Use case name	Companion invite link
Actors	Participant
Starting condition	The participant has an account and is logged in.
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1. The participant clicks a companion invite link they received via email. 2. The app is opened and the user is prompted to enter the relationship to the inviter. 3. The participant selects a relationship 4. The inviter is sent a notification and changes are saved on the server.
Completion condition	The inviter is sent a notification and changes are saved on the server.

Use case name	Changing relationship
Actors	Participant
Starting condition	The participant has an account and is logged in. The participant has not changed the relationship with specified companion before. This is only possible once.
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1. The participant clicks the change relationship button on a companion inside the app. 2. The participant is prompted for the new relationship status. 3. Upon selecting, the changed data is saved on the server.
Completion condition	The changed data is saved on the server.

Use case name	Changing relationship prohibited
Actors	Participant
Starting condition	The participant has an account and is logged in. The participant has changed the relationship with specified companion before. This is only possible once.
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1. The participant clicks the change relationship button on a companion inside the app. 2. The participant is shown a warning mentioning that they have already changed the relationship with this companion.
Completion condition	No changes are done and the participant is shown a warning.

Use case name	Removing companion
Actors	Participant
Starting condition	The participant has an account and is logged in.
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1. The participant clicks the remove companion button on a companion inside the app. 2. The changed data along with a timestamp is saved on the server.
Completion condition	The changed data is saved on the server.

3.4.2.5 Visualize data and help



Use case name	Visualize data
Actors	Participant
Starting condition	The participant has an account and is logged in.
Quality requirements	The interface is easy to understand and use. The visualization is rich and clear.
Events	1. The participant clicks on a calendar day inside the app. 2. A visualization graph is shown, containing information about mood, relaxation, companions and situations recorded on the y-axis and the time on the x-axis.
Completion condition	The graph is displayed properly.

Use case name	Visualize data: Toggle information
Actors	Participant
Starting condition	The participant is viewing the visualization screen.
Quality requirements	The interface is easy to understand and use. The visualization is rich and clear.
Events	1. The participant changes the visualized data (mood, relaxation, companions, situations) with filters 2. The visualization is updated.
Completion condition	The graph is updated properly.

Use case name	Visualize data: Change time frame
Actors	Participant
Starting condition	The participant is viewing the visualization screen. The desired time frame does not cover the current experiment cycle with any of the selected companions.
Quality requirements	The interface is easy to understand and use. The visualization is rich and clear.
Events	1. The participant clicks on the time frame selection and specifies a new time frame. 2. The visualization is updated.
Completion condition	The graph is updated properly.

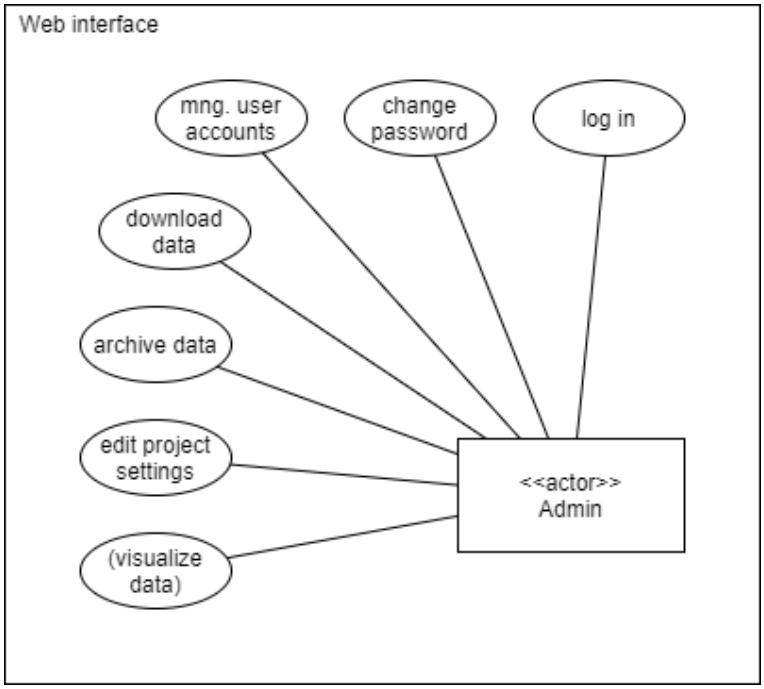
Use case name	Visualize data: Change time frame failure
Actors	Participant
Starting condition	The participant is viewing the visualization screen. The desired time frame does cover the current experiment cycle with any of the selected companions.
Quality requirements	The interface is easy to understand and use. The visualization is rich and clear.
Events	1. The participant clicks on the time frame selection and specifies a new time frame. 2. The participant is shown a warning mentioning that the selected time frame covers a running experiment cycle with a selected companion.
Completion condition	No changes are done and the participant is shown a warning.

Use case name	Visualize data: Select companions
Actors	Participant
Starting condition	The participant is viewing the visualization screen. The current time frame does not cover the current experiment cycle with any of the desired companions.
Quality requirements	The interface is easy to understand and use. The visualization is rich and clear.
Events	1. The participant clicks on the companion selection and selects a new list of up to 5 companions. 2. The visualization is updated.
Completion condition	No changes are done and the participant is shown a warning.

Use case name	Visualize data: Select companions failure
Actors	Participant
Starting condition	The participant is viewing the visualization screen. The current time frame does cover the current experiment cycle with one of the desired companions.
Quality requirements	The interface is easy to understand and use. The visualization is rich and clear.
Events	1. The participant clicks on the companion selection and selects a new list of companions. 2. The participant is shown a warning mentioning that the selected time frame covers a running experiment cycle with a selected companion.
Completion condition	No changes are done and the participant is shown a warning.

Use case name	Help page
Actors	Participant
Starting condition	The participant has an account and is logged in.
Quality requirements	The interface is easy to understand and use. The help page is informative.
Events	1. The participant clicks the help button inside the app. 2. A help page containing information about the app and its usage is presented.
Completion condition	The help page is displayed properly.

3.4.2.6 Web interface



Use case name	Admin log in
Actors	Admin
Starting condition	The admin has a functioning web browser and knows the address of the web interface.
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1. The admin loads the web interface and is presented a log in form. 2. Upon filling in valid credentials, the interface compares provided data with server data. 3. The data is validated and the admin is now logged in.
Completion condition	Logging in is successful.

Use case name	Admin log in failure
Actors	Admin
Starting condition	The admin has a functioning web browser and knows the address of the web interface.
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1. The admin loads the web interface and is presented a log in form. 2. Upon filling in invalid credentials, the interface compares provided data with server data. 3. The data is refused and the admin is shown a warning.
Completion condition	No changes are done and the admin is shown a warning.

Use case name	Change admin password
Actors	Admin
Starting condition	The admin is logged in to the web interface.
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1. The admin clicks the change password button on the web interface. 2. The admin is presented a password changing form and fills in the fields with valid credentials. 3. The data is validated and the password changes are reflected on the server.
Completion condition	The password is changed and the new password is saved on the server.

Use case name	Change admin password failure
Actors	Admin
Starting condition	The admin is logged in to the web interface.
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1. The admin clicks the change password button on the web interface. 2. The admin is presented a password changing form and fills in the fields with invalid credentials. 3. The data is refused and the admin is shown a warning.
Completion condition	No changers are done and the admin is shown a warning.

Use case name	Project settings
Actors	Admin
Starting condition	The admin is logged in to the web interface.
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1. The admin clicks the edit project settings button on the web interface. 2. The admin can choose to edit notification frequency and synchrony, length of experimental cycle and consent form text.
Completion condition	The admin is shown a settings screen.

Use case name	Project settings: Notifications
Actors	Admin
Starting condition	The admin is viewing the web settings screen.
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1. The admin clicks the edit notification settings button on the web interface. 2. The admin enters new notification frequency and synchrony data (how many notifications a day, time range, whether all participants receive notifications at the same time). 3. Changes are saved on the server.
Completion condition	Changes are saved on the server.

Use case name	Project settings: Experimental cycle duration
Actors	Admin
Starting condition	The admin is viewing the web settings screen.
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1. The admin clicks the edit cycle duration settings button on the web interface. 2. The admin enters a new cycle duration in days. 3. Changes are saved on the server.
Completion condition	Changes are saved on the server.

Use case name	Project settings: Consent form text
Actors	Admin
Starting condition	The admin is viewing the web settings screen.
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1. The admin clicks the edit consent form settings button on the web interface. 2. The admin enters a consent form text. 3. All previous and the new consent form texts are assigned an ID and saved on the server.
Completion condition	Changes are saved on the server.

Use case name	Delete user account
Actors	Admin
Starting condition	The admin is logged in to the web interface.
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1. The admin received a request to delete a user account. 2. The admin chooses to remove a user account from within the web interface. 3. The admin is prompted for either the user id or email address of the account to be deleted. 4. The admin enters valid account information (id / email). 5. The data is validated and the corresponding account is deleted.
Completion condition	All account data for the specified account is deleted from the server.

Use case name	Delete user account failure
Actors	Admin
Starting condition	The admin is logged in to the web interface.
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1. The admin received a request to delete a user account. 2. The admin chooses to remove a user account from within the web interface. 3. The admin is prompted for either the user id or email address of the account to be deleted. 4. The admin enters invalid account information (id / email). 5. The data is refused and the admin is shown a warning.
Completion condition	No changes are done and the admin is shown a warning.

Use case name	Download all data
Actors	Admin
Starting condition	The admin is logged in to the web interface.
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1. The admin clicks the download data button on the web interface. 2. A single CSV file containing all data with one line per user is generated and a download is prompted.
Completion condition	A CSV file is generated and downloaded.

Use case name	Download data for time frame
Actors	Admin
Starting condition	The admin is logged in to the web interface.
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1. The admin clicks the download data for time frame button on the web interface. 2. The admin is prompted for and selects a time frame. 3. A single CSV file containing the data for the specified time frame with one line per user is generated and a download is prompted.
Completion condition	A CSV file is generated and downloaded.

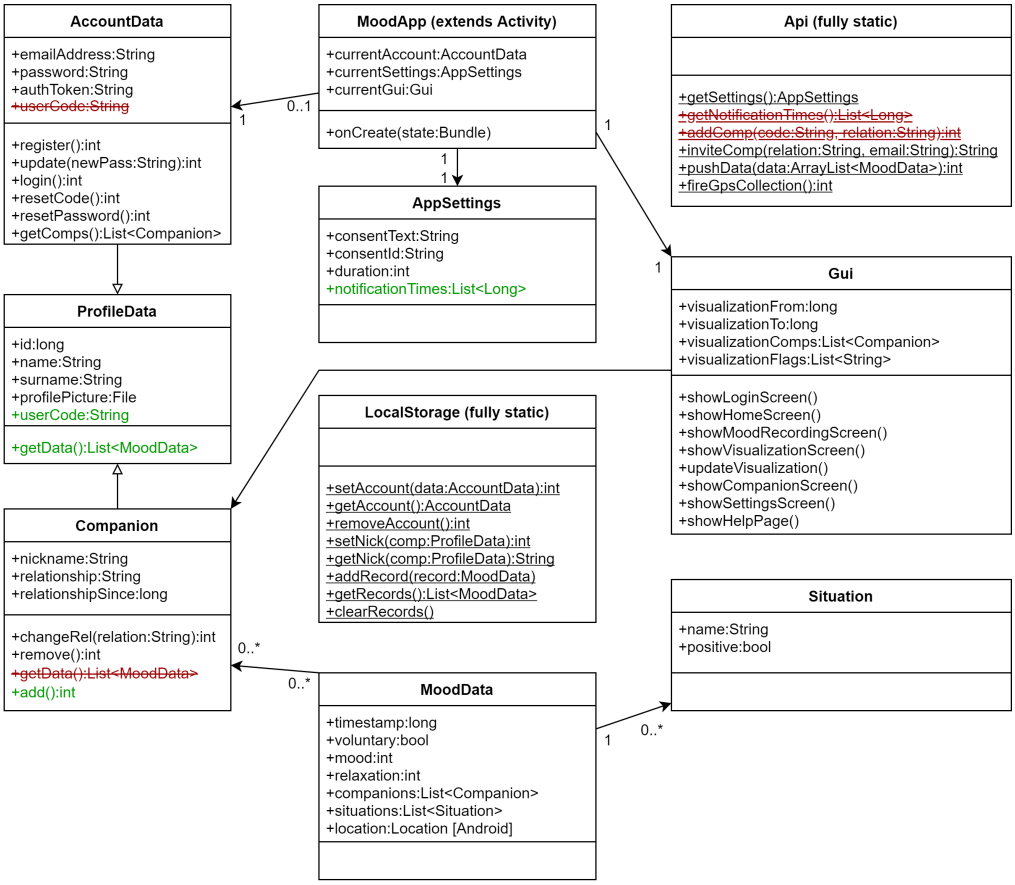
Use case name	Archive data
Actors	Admin
Starting condition	The admin is logged in to the web interface.
Quality requirements	The interface is easy to understand and use. The operations run seamlessly and quickly.
Events	1. The admin clicks the archive data button on the web interface. 2. The admin is prompted for and selects a time frame. 3. The specified time frame is marked as archived on the server. 4. This change is visible in the web interface in the future.
Completion condition	The changes are saved on the server.

Use case name	Web visualize data
Actors	Admin
Starting condition	The admin is logged in to the web interface.
Quality requirements	The interface is easy to understand and use. The visualization is rich and clear.
Events	1. The admin clicks the visualize data button on the web interface. 2. A visualization graph is shown, containing information about mood, relaxation, companions and situations.
Completion condition	The graph is displayed properly.

Use case name	Web visualize data: Select time frame
Actors	Admin
Starting condition	The admin is viewing the visualization screen.
Quality requirements	The interface is easy to understand and use. The visualization is rich and clear.
Events	1. The admin picks a new time frame. 2. The visualization is updated.
Completion condition	The graph is updated properly.

Use case name	Web visualize data: Select participant
Actors	Admin
Starting condition	The admin is viewing the visualization screen.
Quality requirements	The interface is easy to understand and use. The visualization is rich and clear.
Events	1. The admin picks a new participant to visualize. 2. The visualization is updated.
Completion condition	The graph is updated properly.

3.4.3 Class model



3.4.4 UI prototypes

Figure 1: UI prototype 1

Figure 2: UI prototype 2

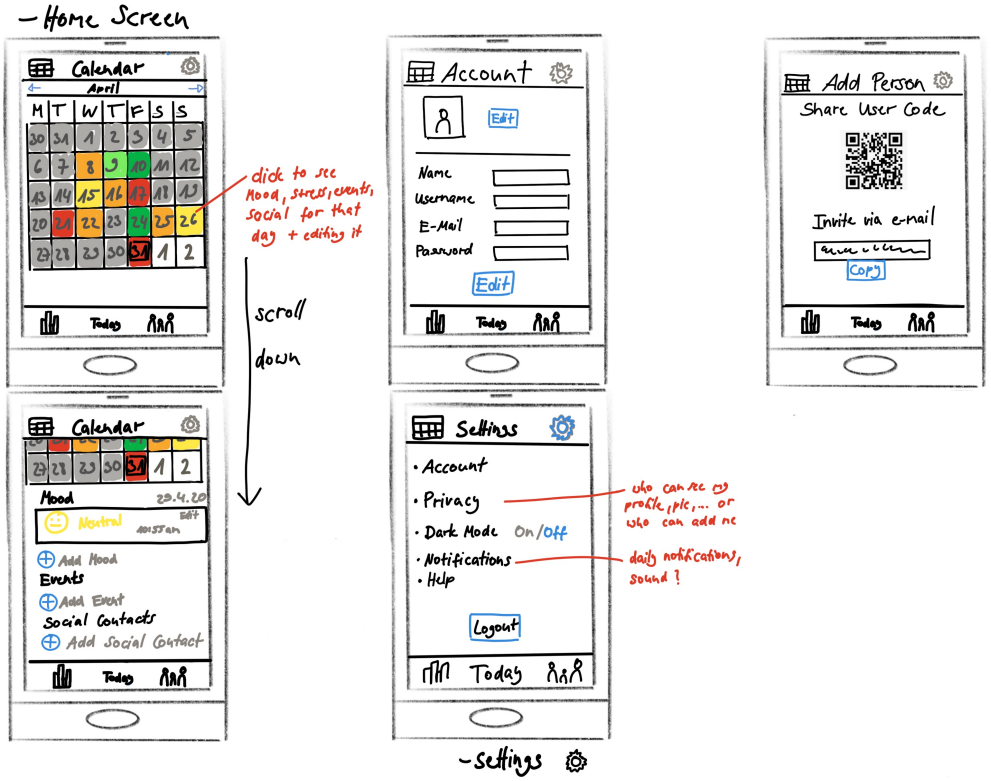


Figure 3: UI prototype 3

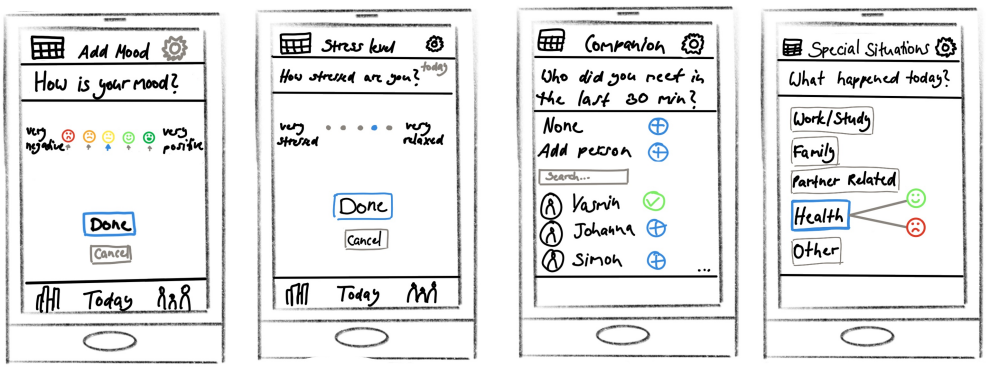


Figure 4: UI prototype 4

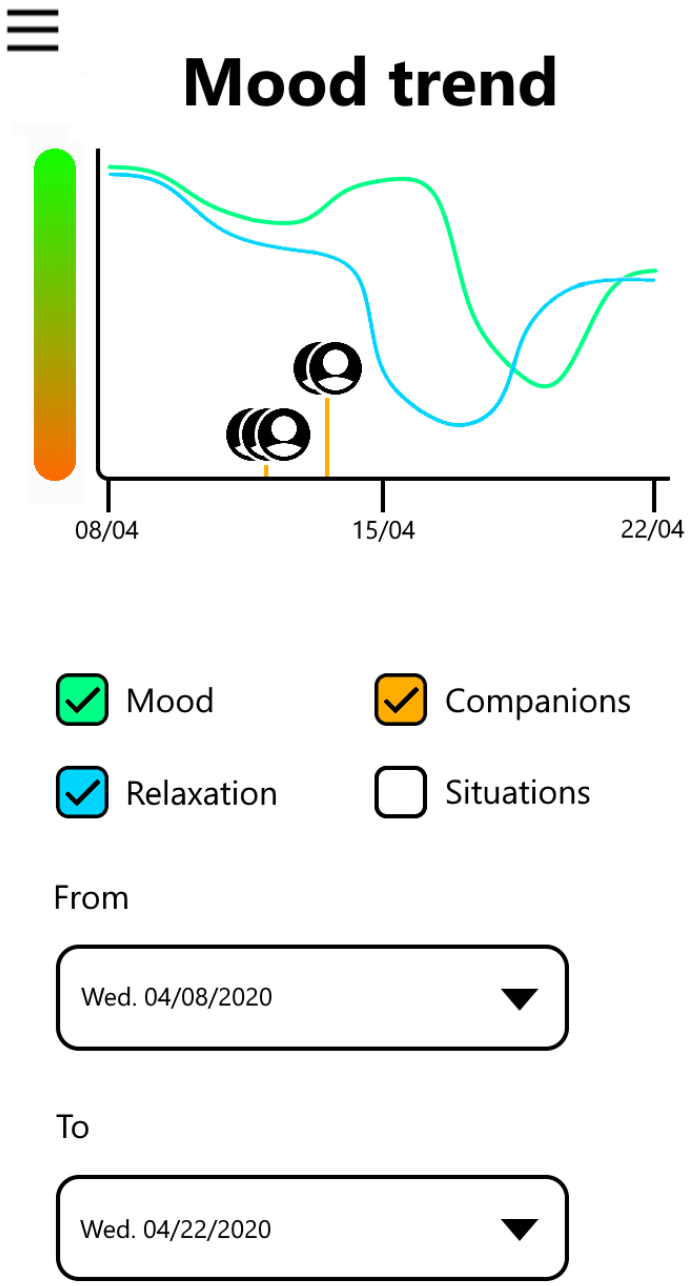


Figure 5: Web UI prototype 1

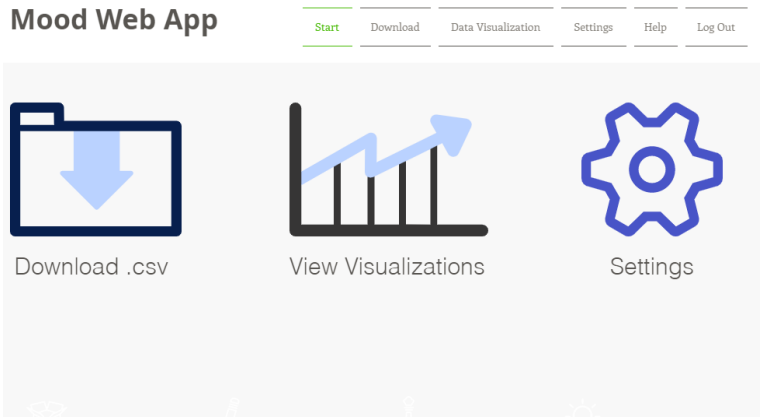


Figure 6: Web UI prototype 2

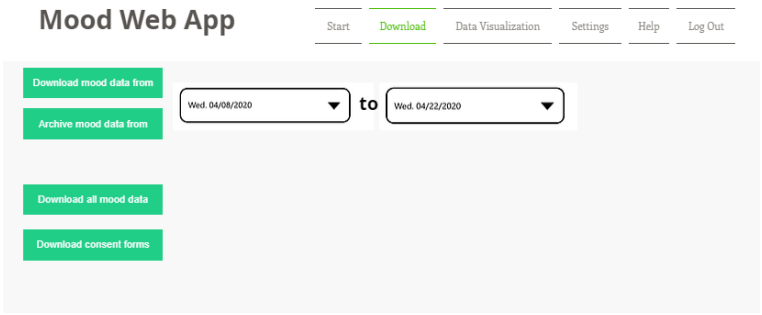


Figure 7: Web UI prototype 3

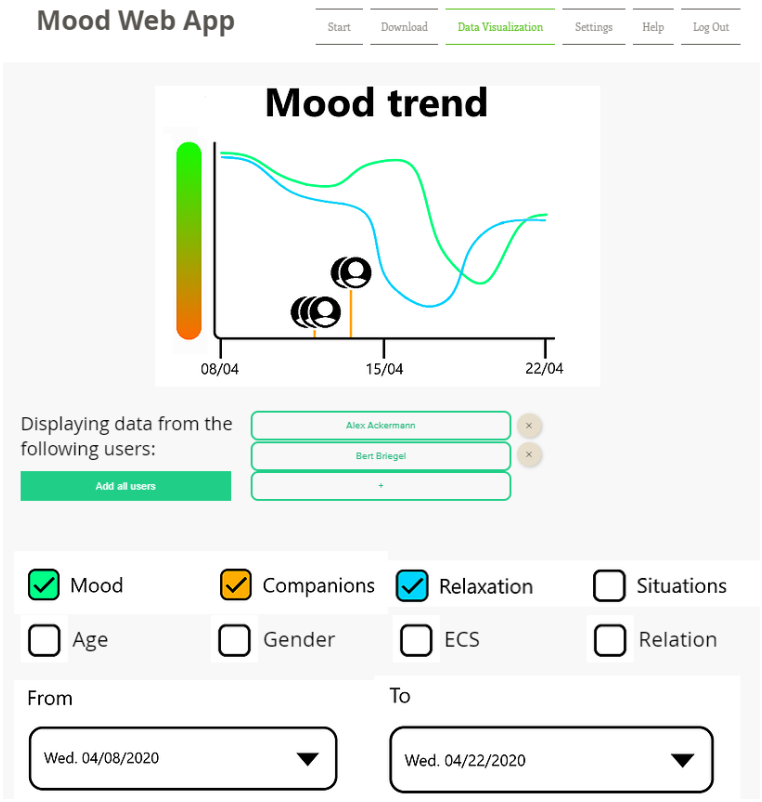
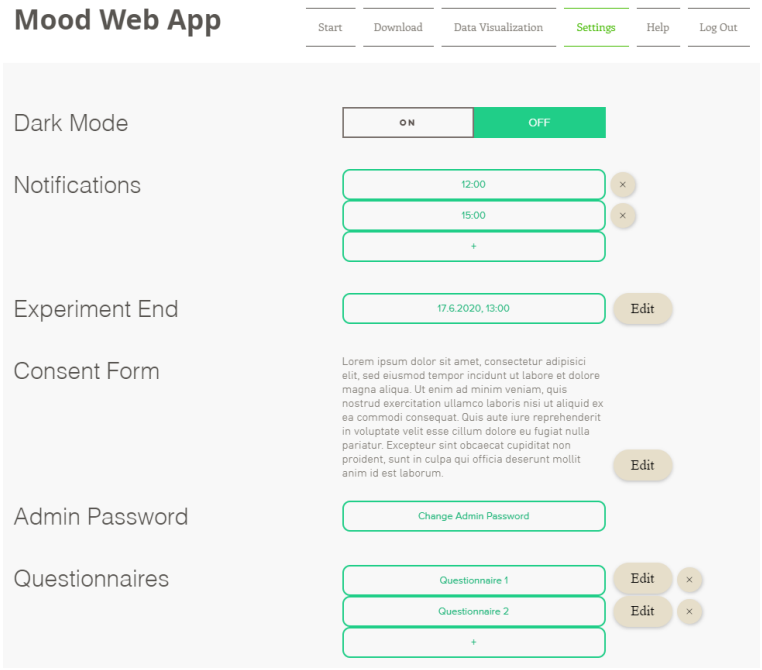


Figure 8: Web UI prototype 4



4 Glossary

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

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.

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 III

System Design

5 Introduction

5.1 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.

5.2 Design goals

5.2.1 System performance

5.2.1.1 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+.

5.2.1.2 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.

5.2.2 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.

5.2.2.1 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.

5.2.2.2 Availability

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

5.2.2.3 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.

5.2.3 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.

5.2.3.1 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.

5.2.3.2 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.

5.2.3.3 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.

5.2.4 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.

5.2.5 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.

5.2.5.1 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).

5.2.5.2 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.

5.2.5.3 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.

5.2.5.4 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.

5.3 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.

5.4 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.

5.5 Literature references

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

5.6 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.

6 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.

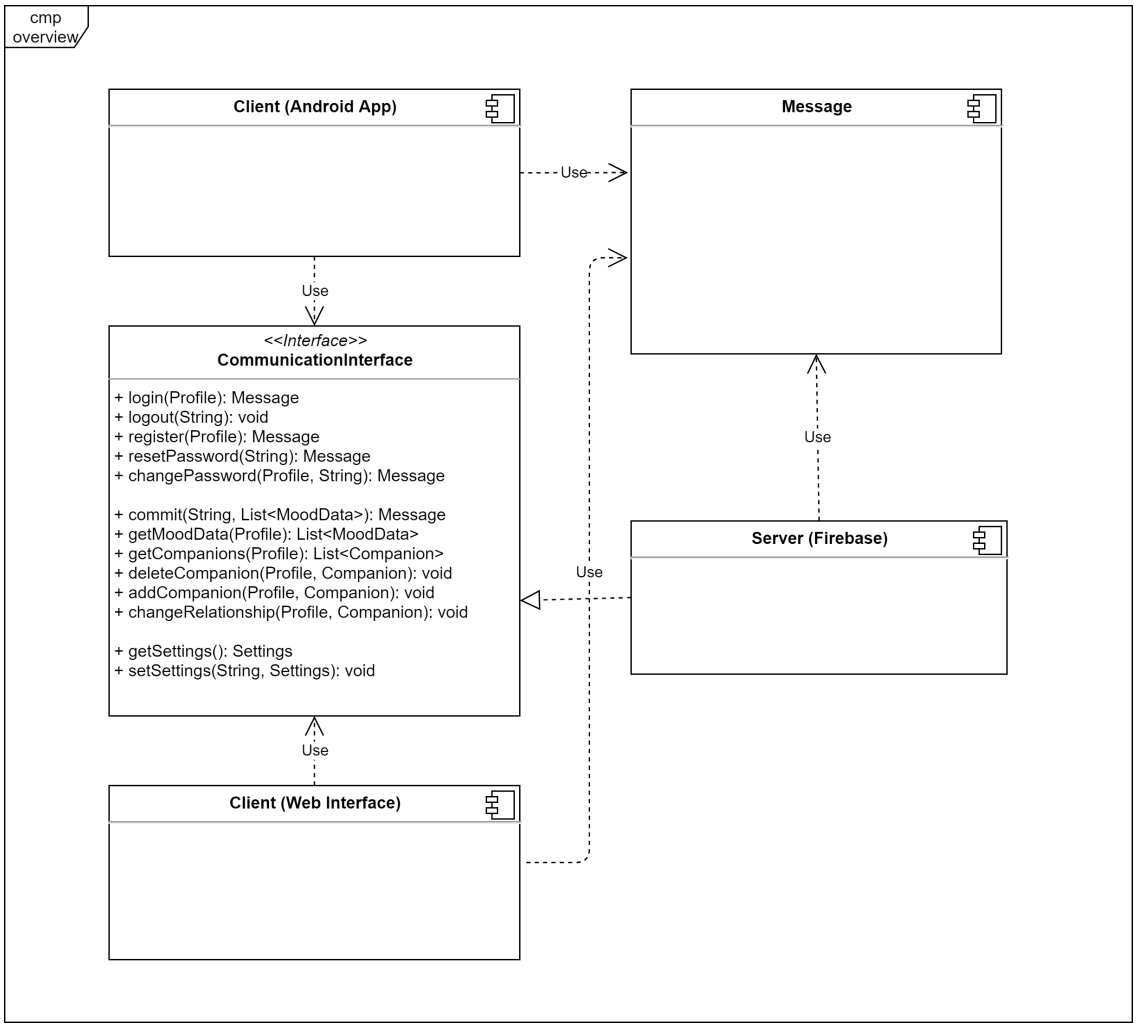
7 Suggested system

7.1 Overview

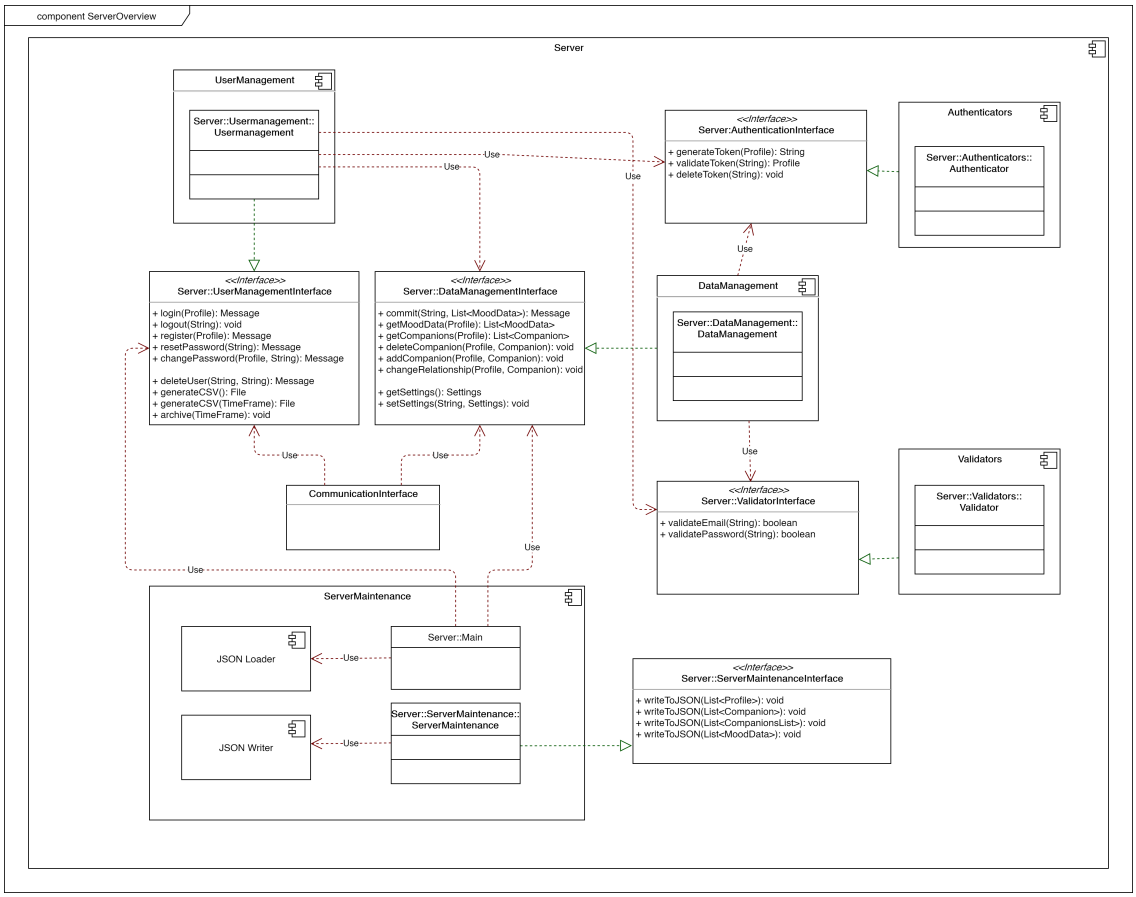
The suggested system is divided into three components. We have the android app, a server and a web interface. The app is the most extensive part of this project. The data gathered in this app, will be uploaded to the server and can be downloaded via the web interface by the leader of the experiment. 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.

7.2 Component segmentation and description of interfaces

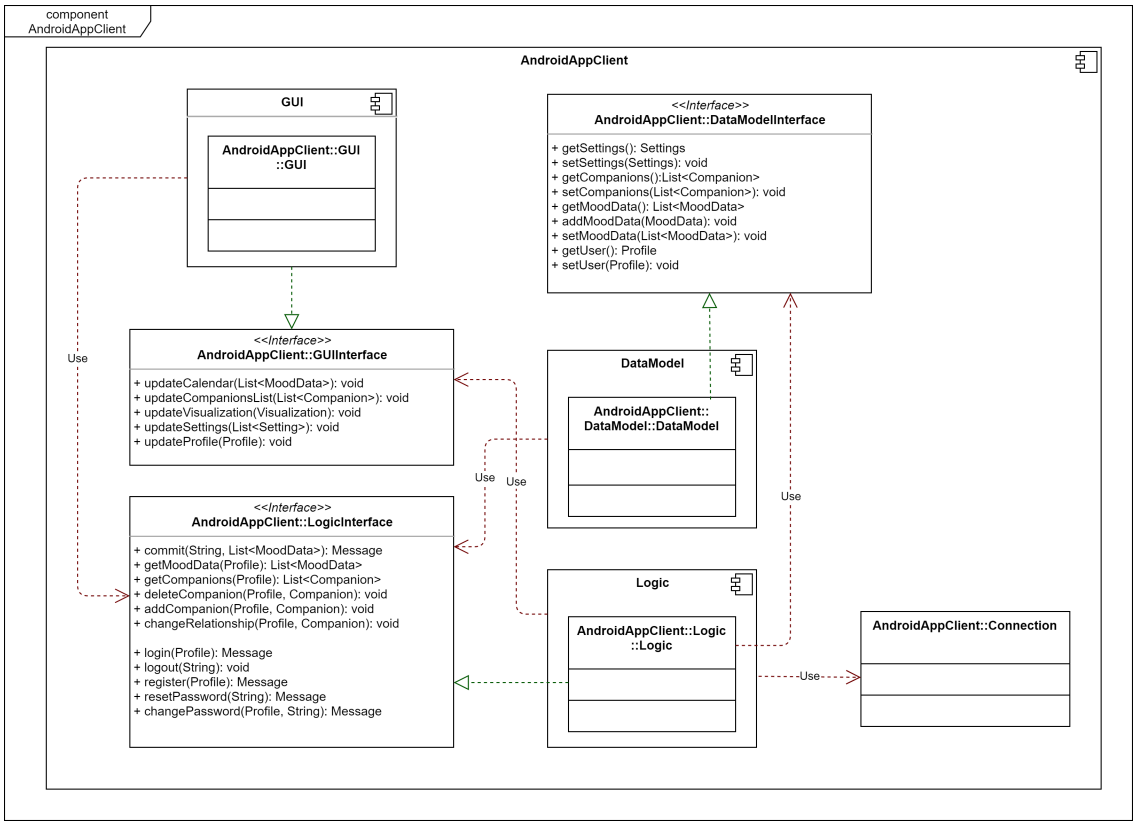
The following diagram gives an overview over the different components of our system and how they are going to communicate with each other. There are 2 client components (android and web application) and a server component that all communicate through the message component.

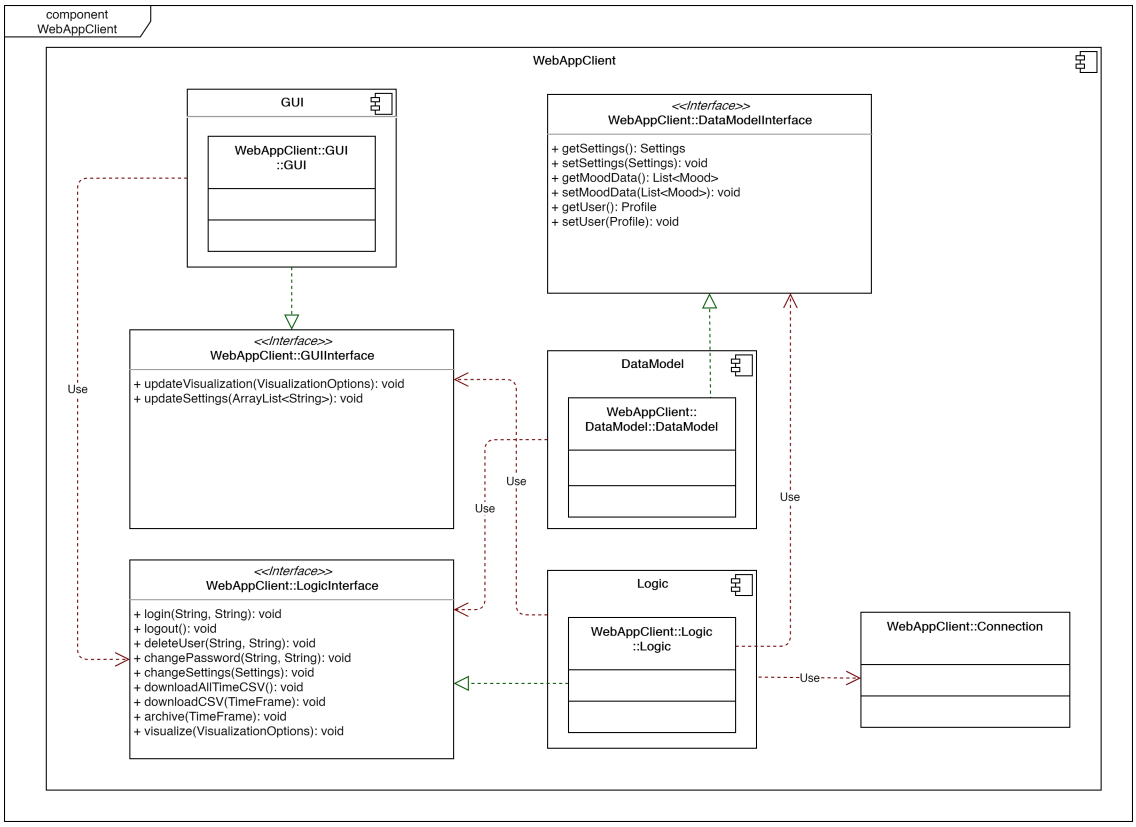


The server mainly consists of three components, the server maintenance component, the user management component and the data management component. There is a validators component and a authentication component used by both the data and user management component.



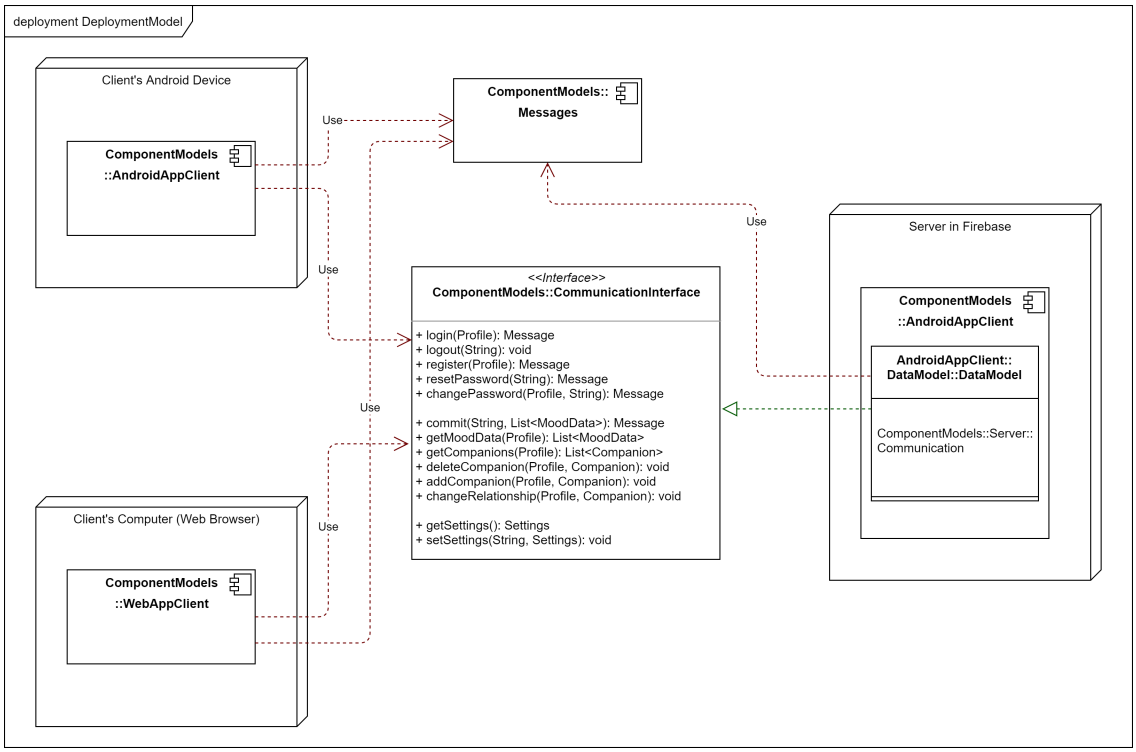
The android app and the web app consist of three main components each, one being the GUIInterface the participants and admin use to interact with the system, the LogicInterface summarizing the logic methods used to communicate with the server and the DataModelInterface listing methods used to manage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

7.3 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.



7.4 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.

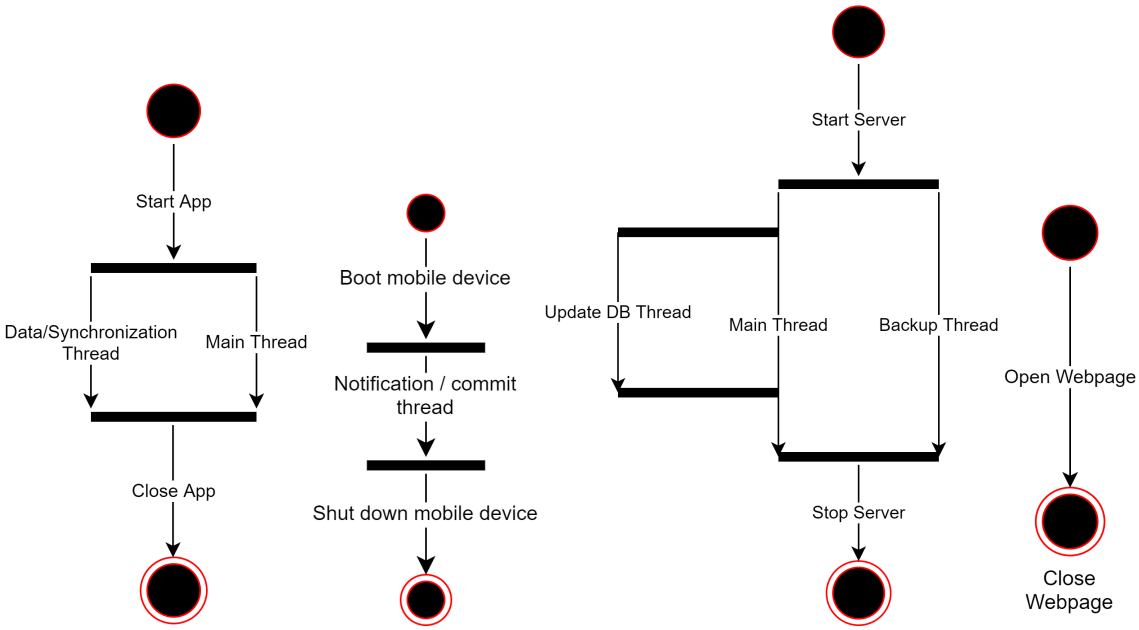
7.5 Access rights and access control

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

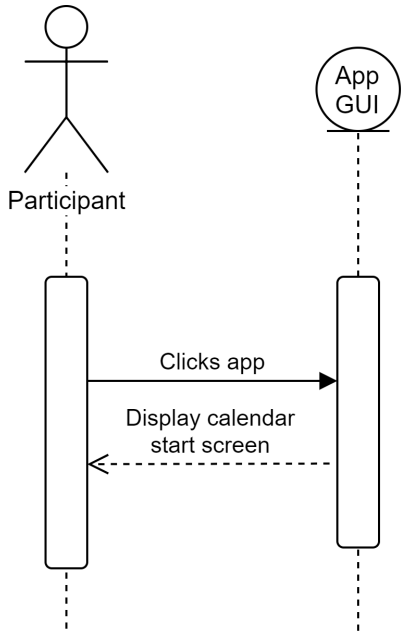
The admin will not be able to mess with the companions data, be it companions or recorded mood. However he/she will be able to change the settings of the app, since these also include the terms of use and experiment duration, which needs to be changed from the web interface. Even though the user can 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.

7.6 Global control flow

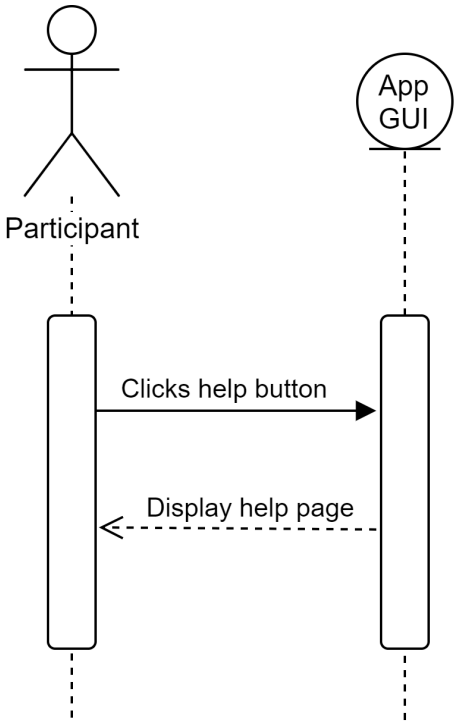
Our system is divided into three components: The server, the android app and the web app.
The server will have one main thread running and one backup thread, which will regularly save the data. To ensure that the data is always consistent, we need to ensure the atomarity of the commits. Therefore we 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.



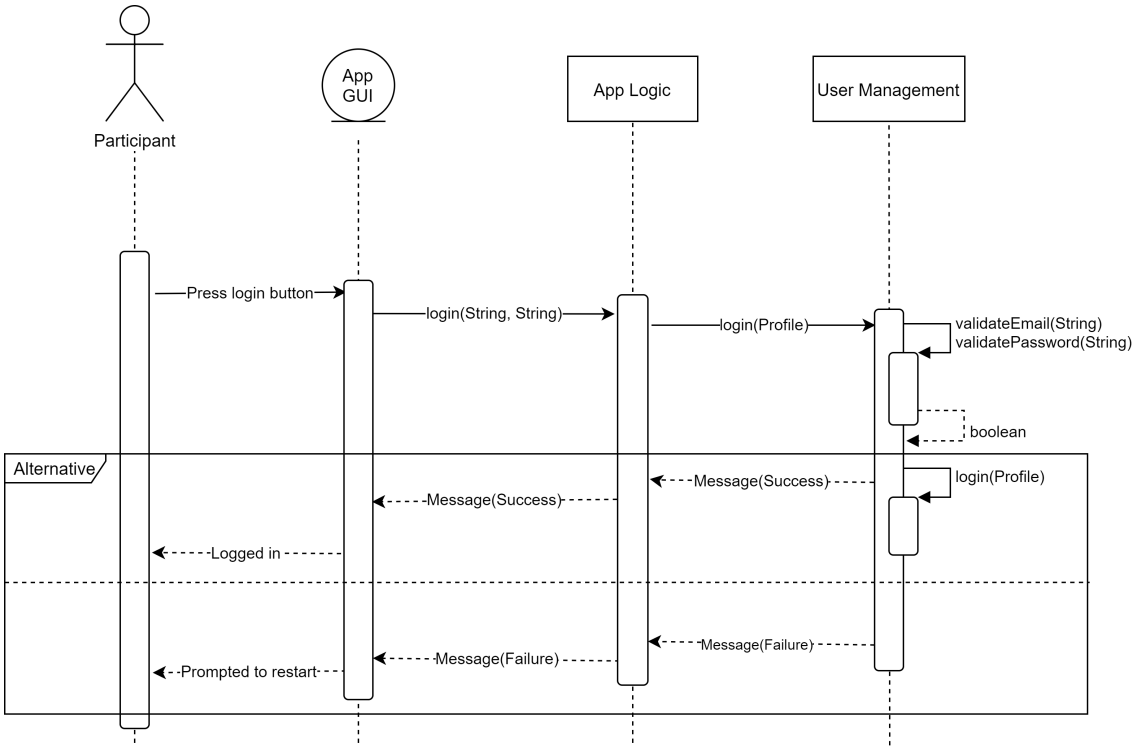
The following diagram displays the control flow of when the participant starts the application:



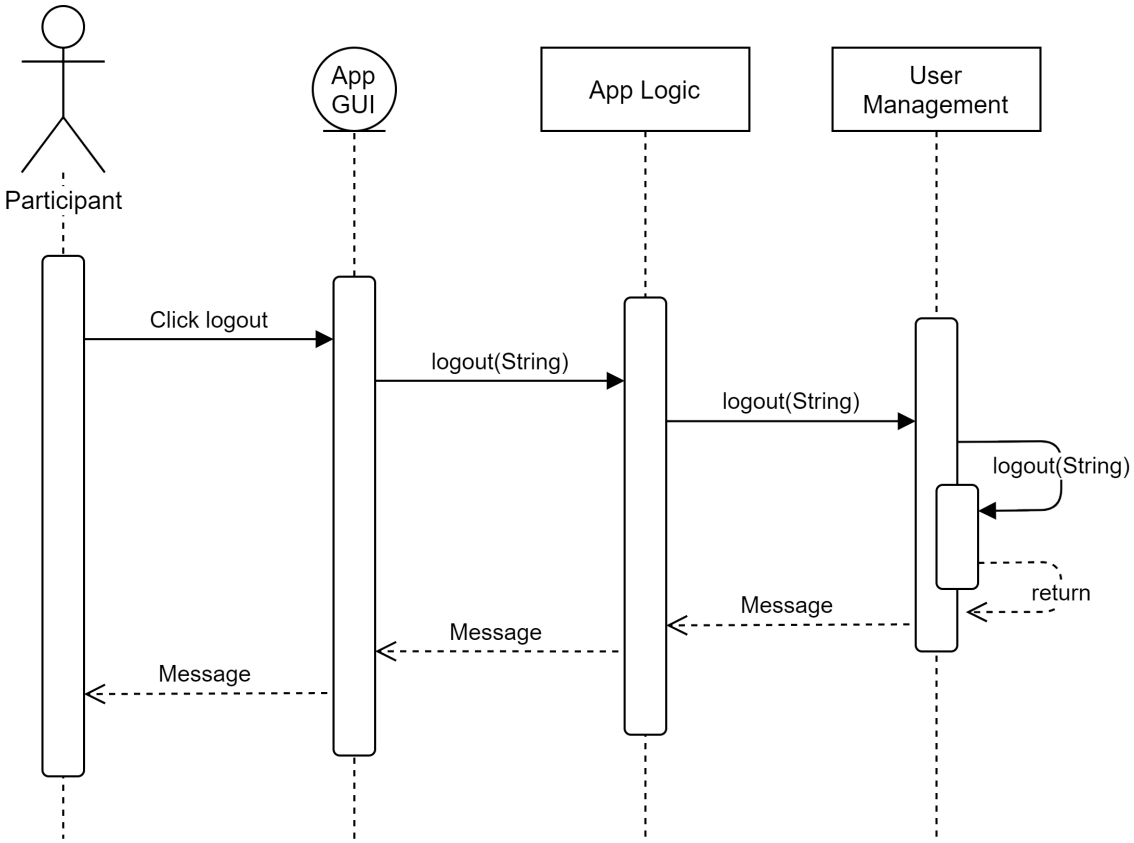
The following diagram displays the control flow of when the client presses the help button in the application:



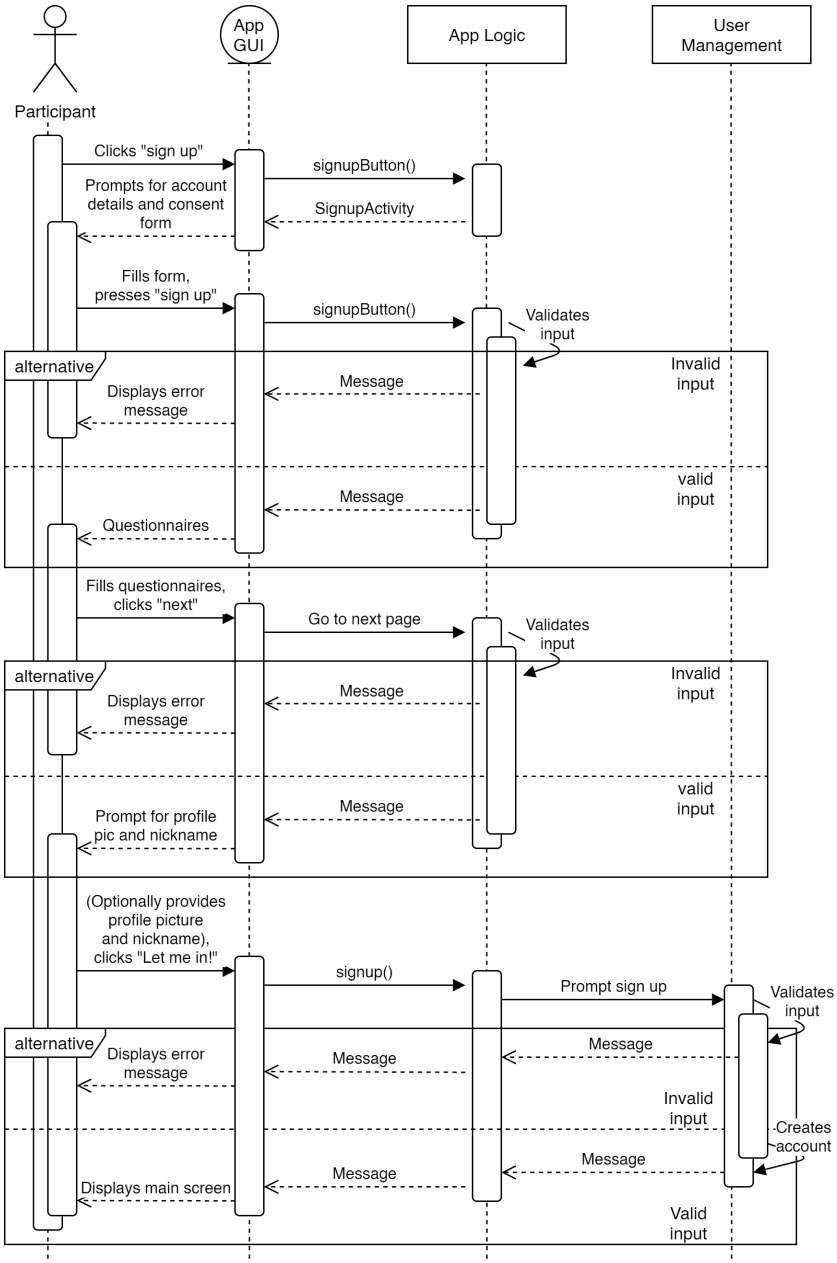
The following diagram displays the control flow of the client-invoked method call on the server when a participant logs in:



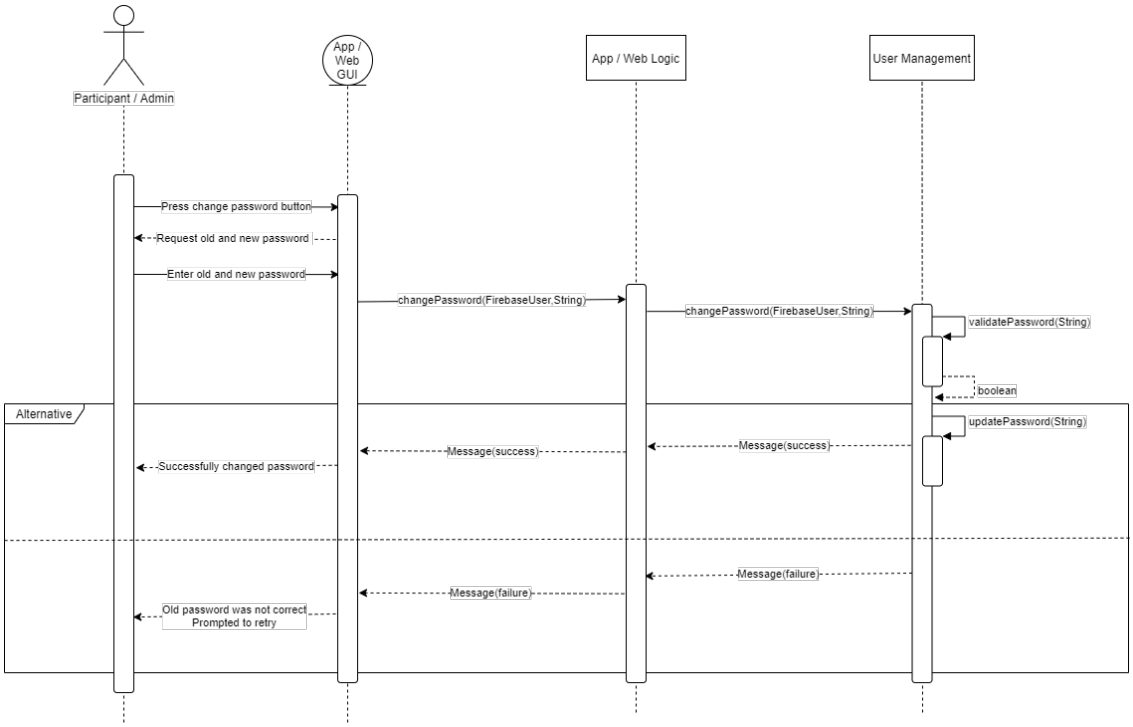
The following diagram displays the control flow of the client-invoked method call on the server when a participant logs out:



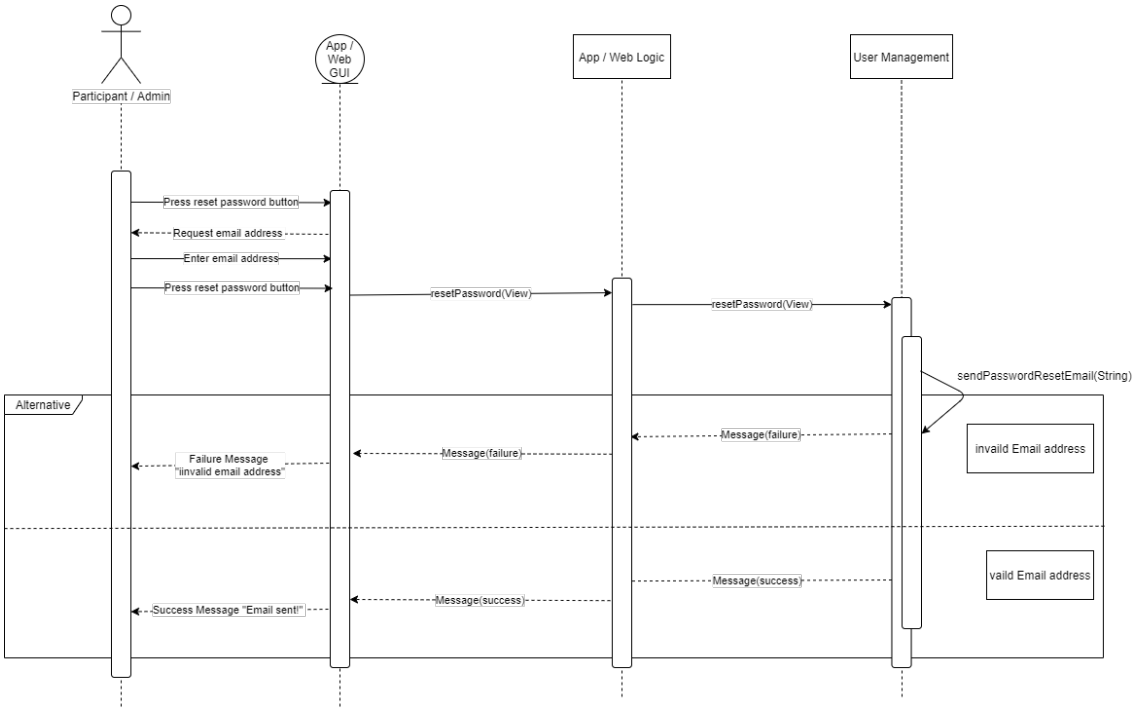
The following diagram displays the control flow of the client-invoked method call on the server when a participant signs up:



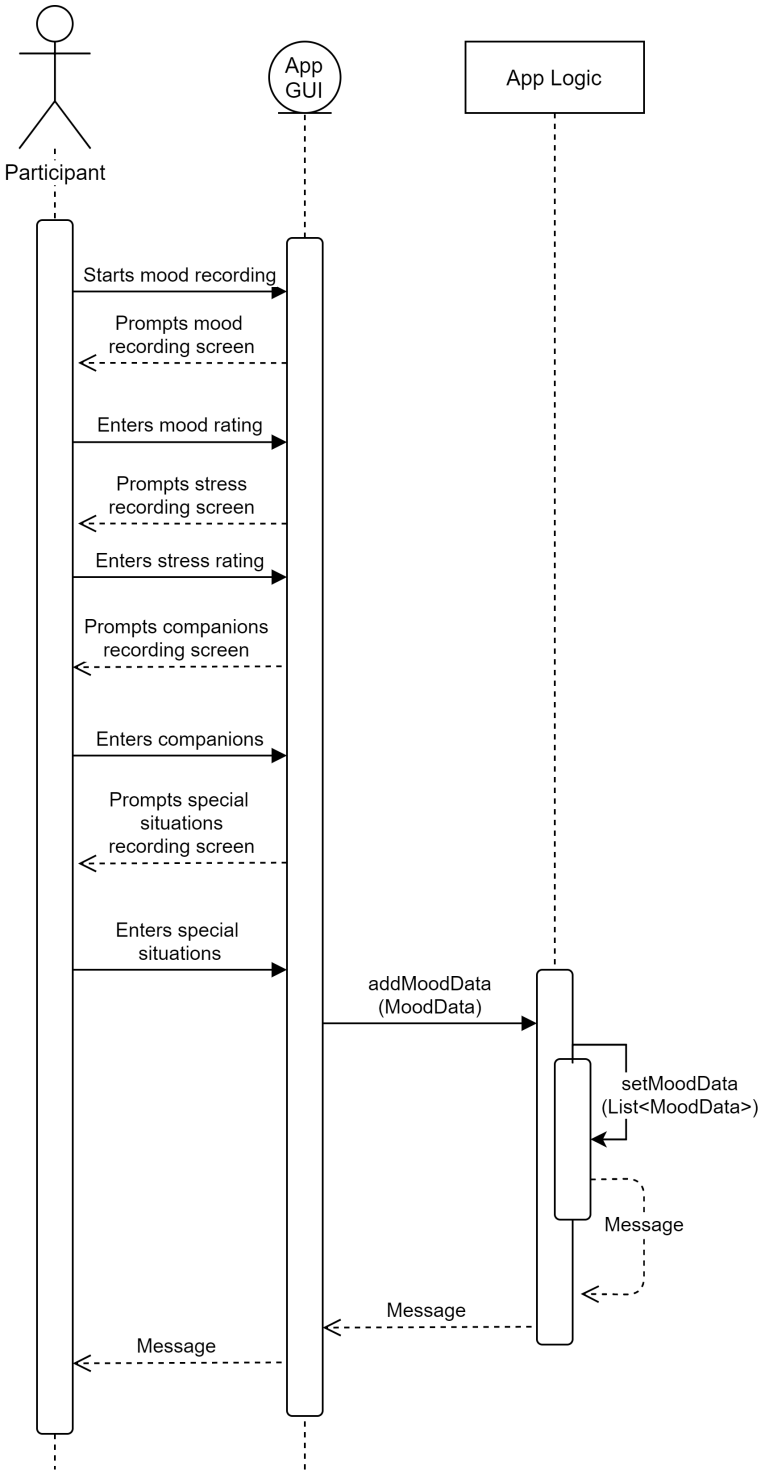
The following diagram displays the control flow of the client-invoked method call on the server when a participant or administrator changes their password:



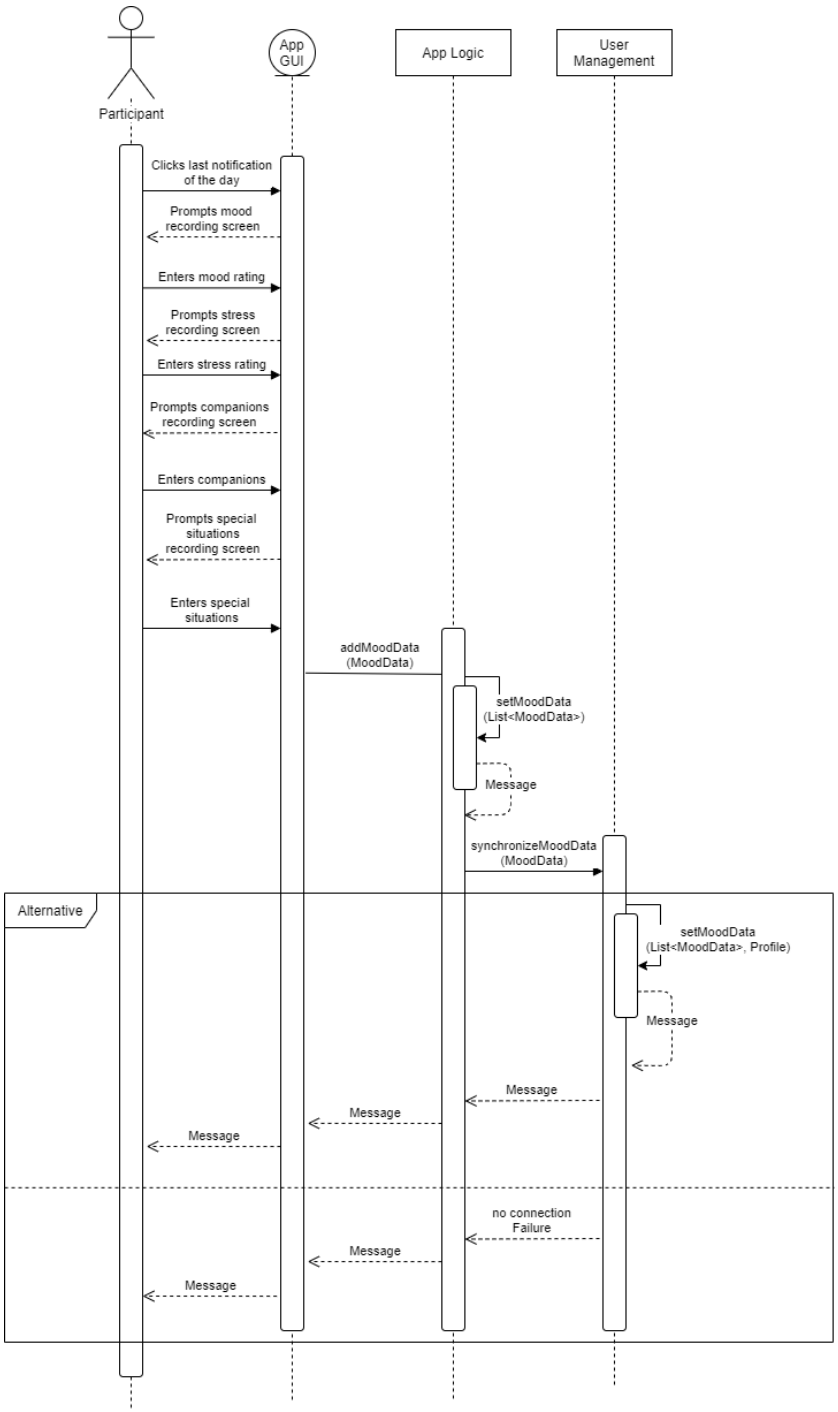
The following diagram displays the control flow of the client-invoked method call on the server when a participant resets his password:



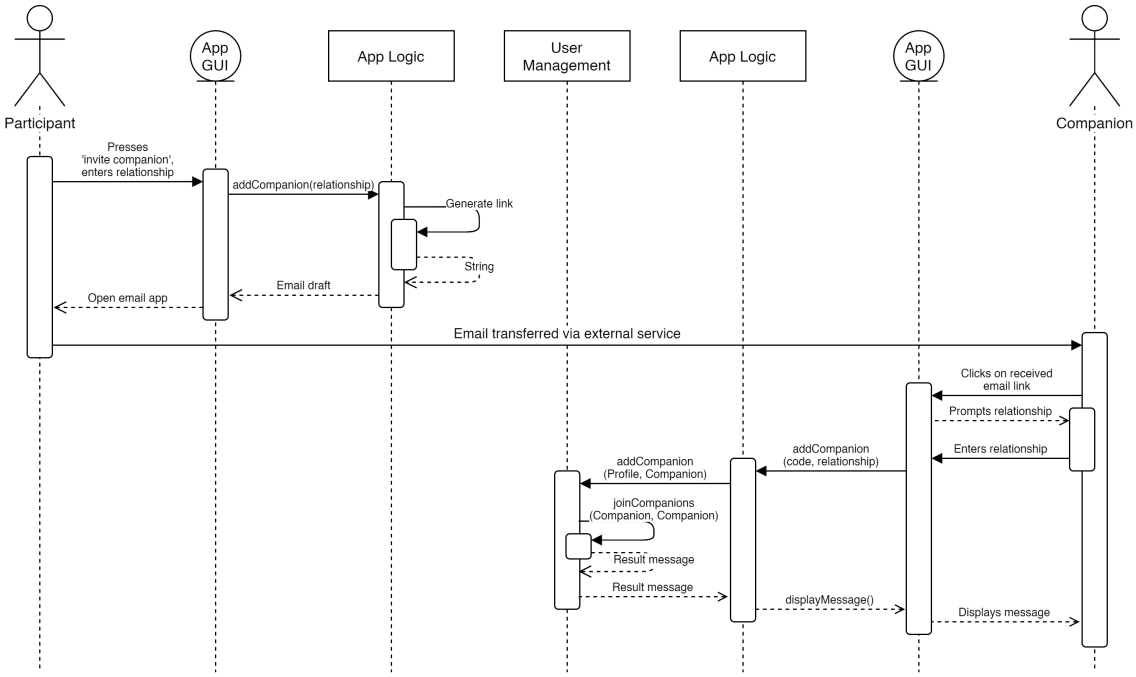
The following diagram displays the control flow of the client-invoked method call on the server when a participant makes a mood recording:



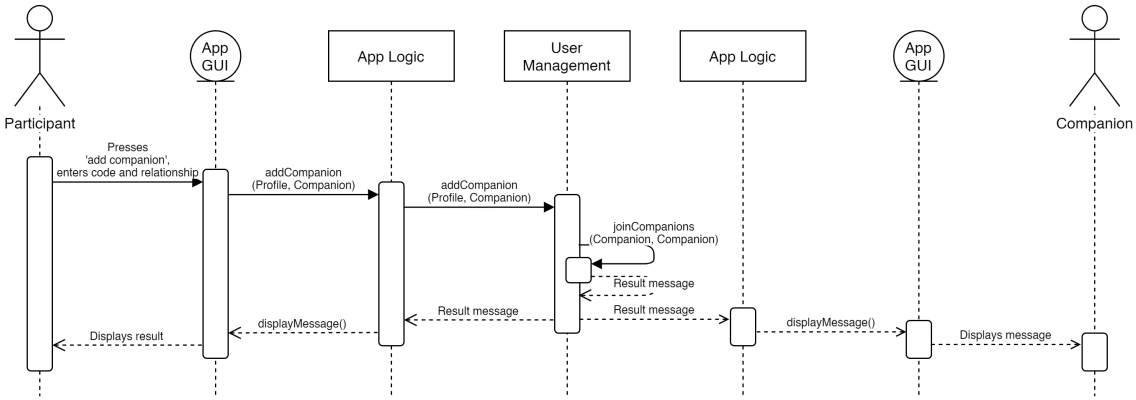
The following diagram displays the control flow of when the participant makes the last mood recording of the day and the data is synchronized with the server successfully or the data is not synchronized with the server, because of connection Failure:



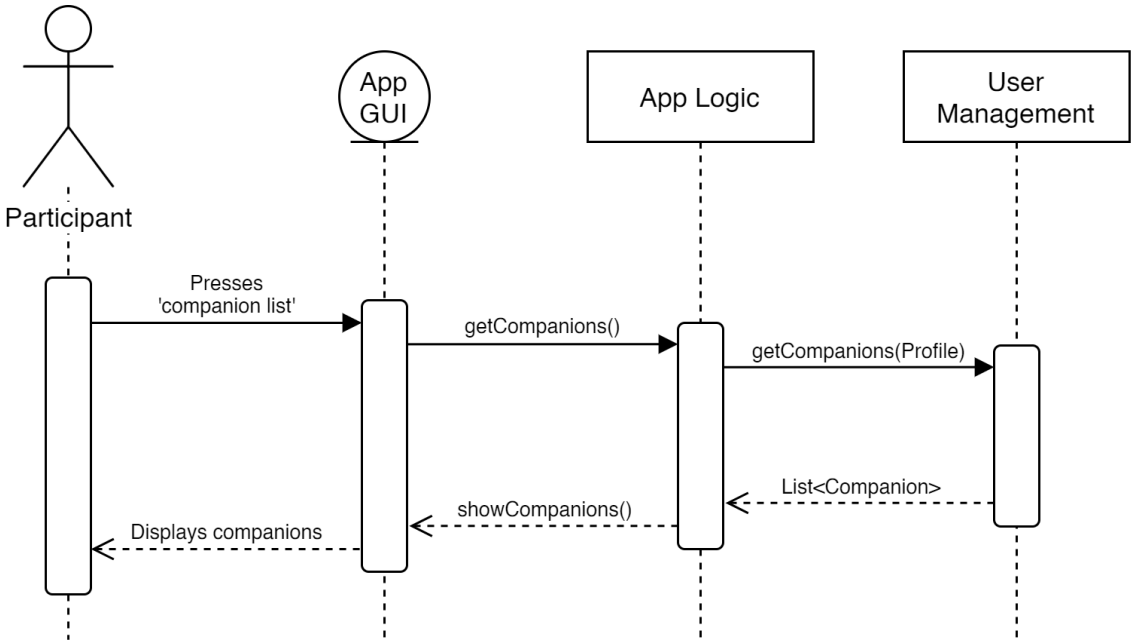
The following diagram displays the control flow of the client-invoked method calls on the server when a participant invites a companion:



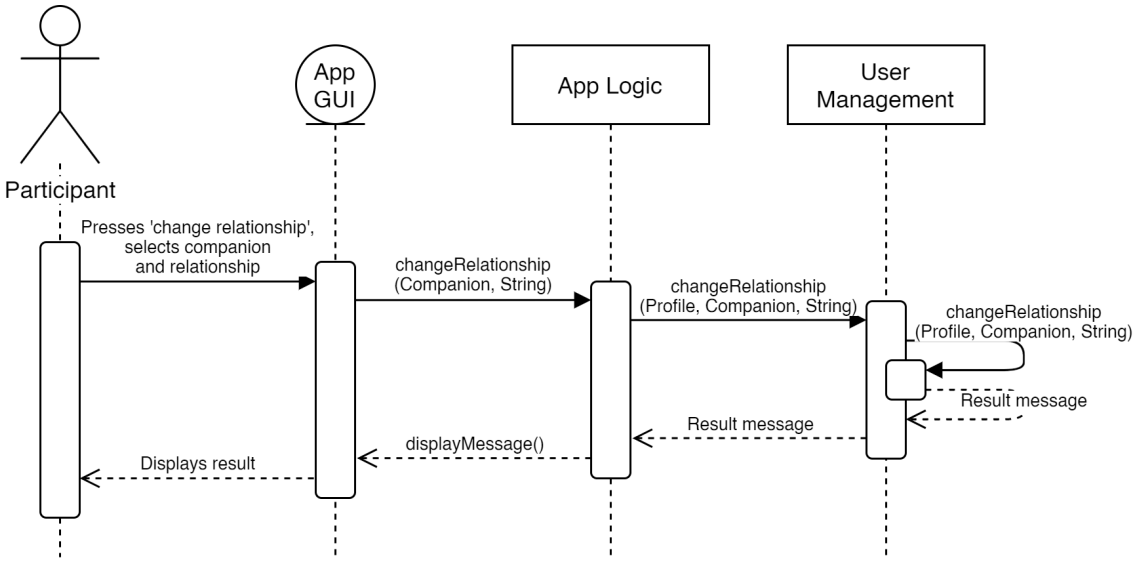
The following diagram displays the control flow of the client-invoked method calls on the server when a participant adds a companion:



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

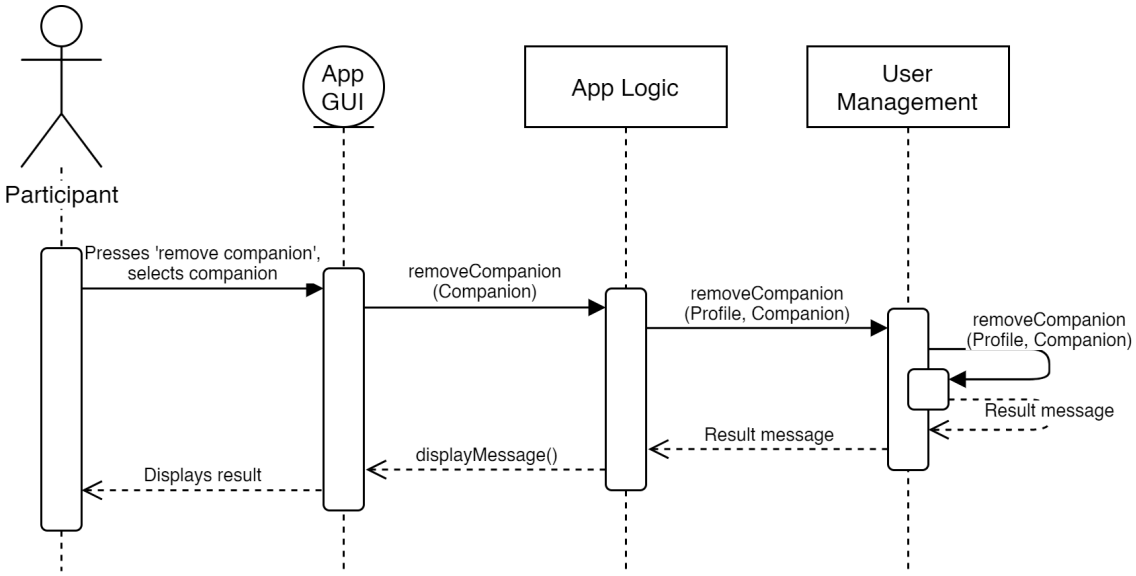


The following diagram displays the control flow of the client-invoked method calls on the server when a participant changes their relationship with a companion:

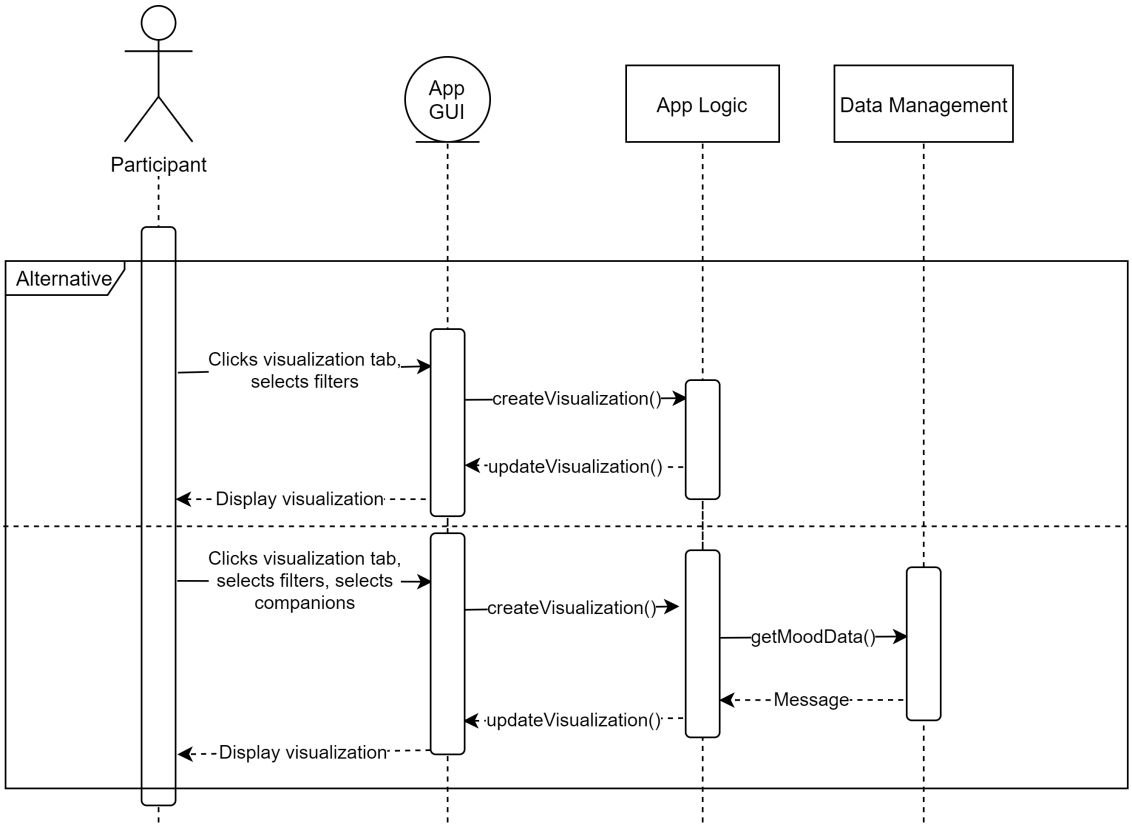


This feature was not implemented into the app due to time limits. Changing the relationship can still be achieved by removing and re-adding a companion.

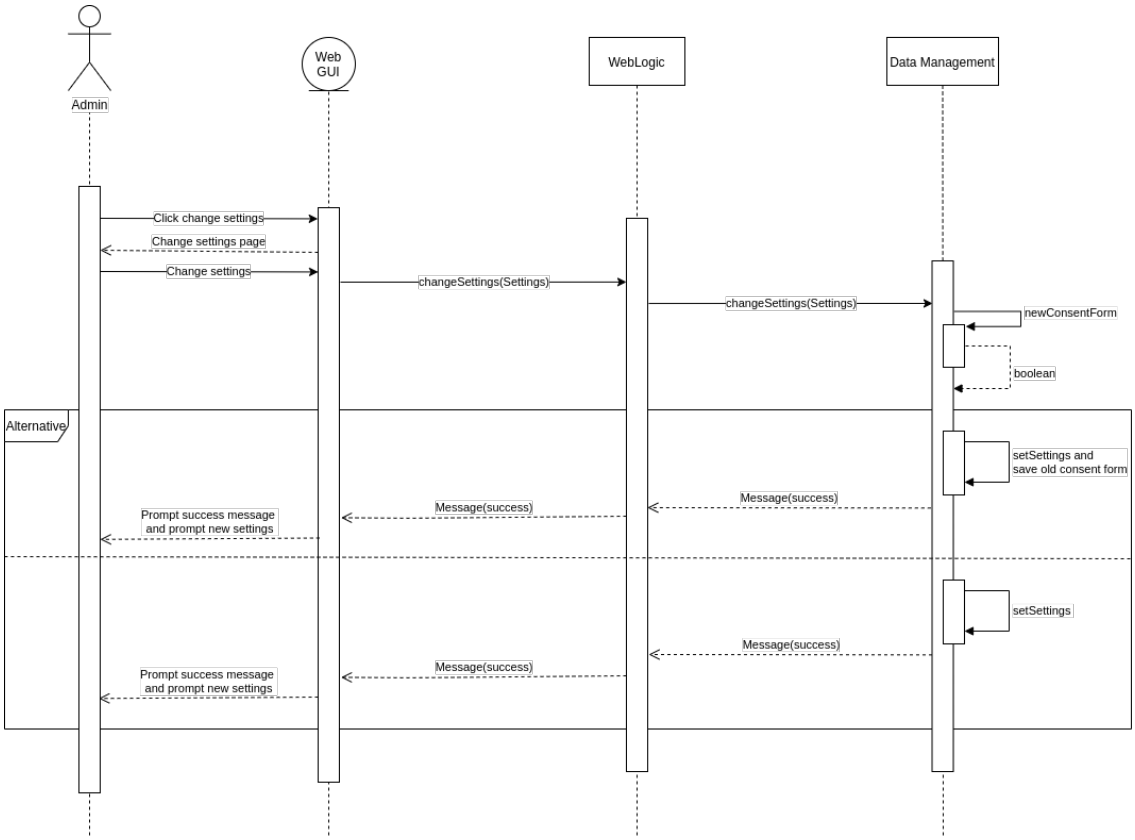
The following diagram displays the control flow of the client-invoked method calls on the server when a participant removes a companion:



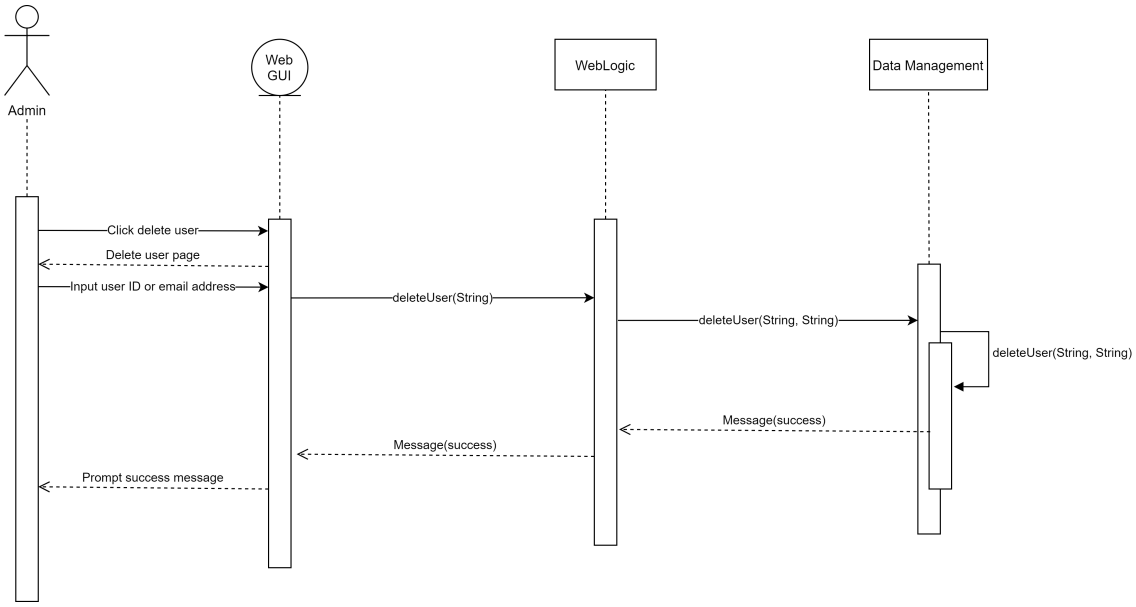
The following diagram displays the control flow of the client-invoked method call on the server when a participant wants to visualize mood data:



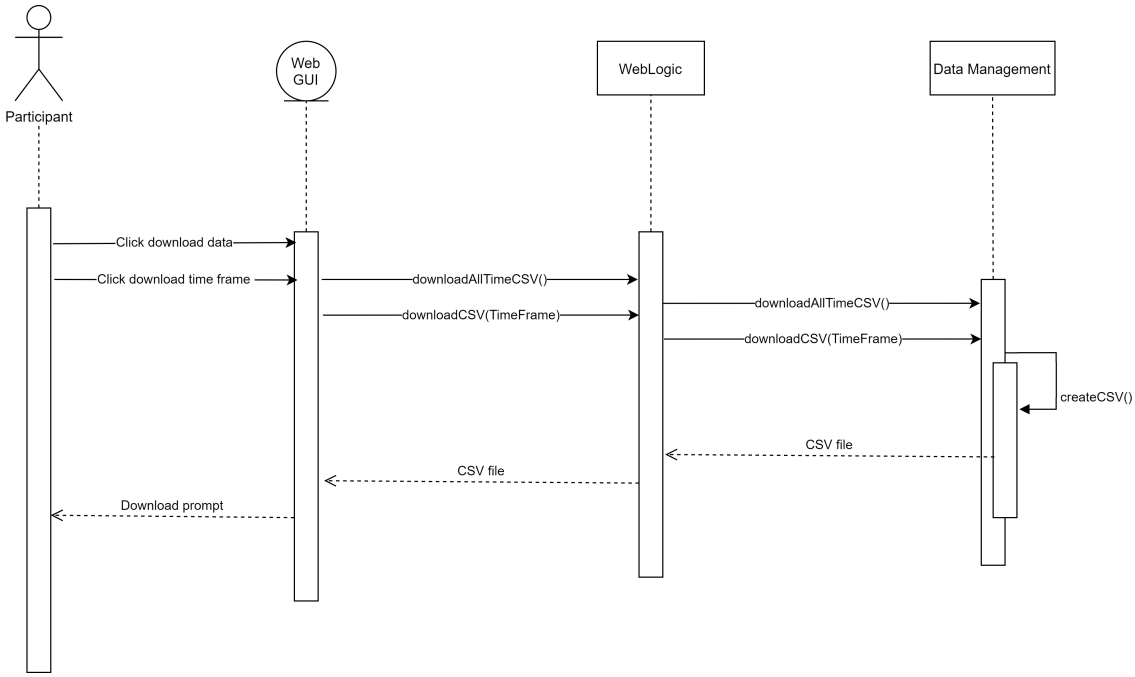
The following diagram displays the control flow of the client-invoked method call on the server when the administrator changes the project settings:



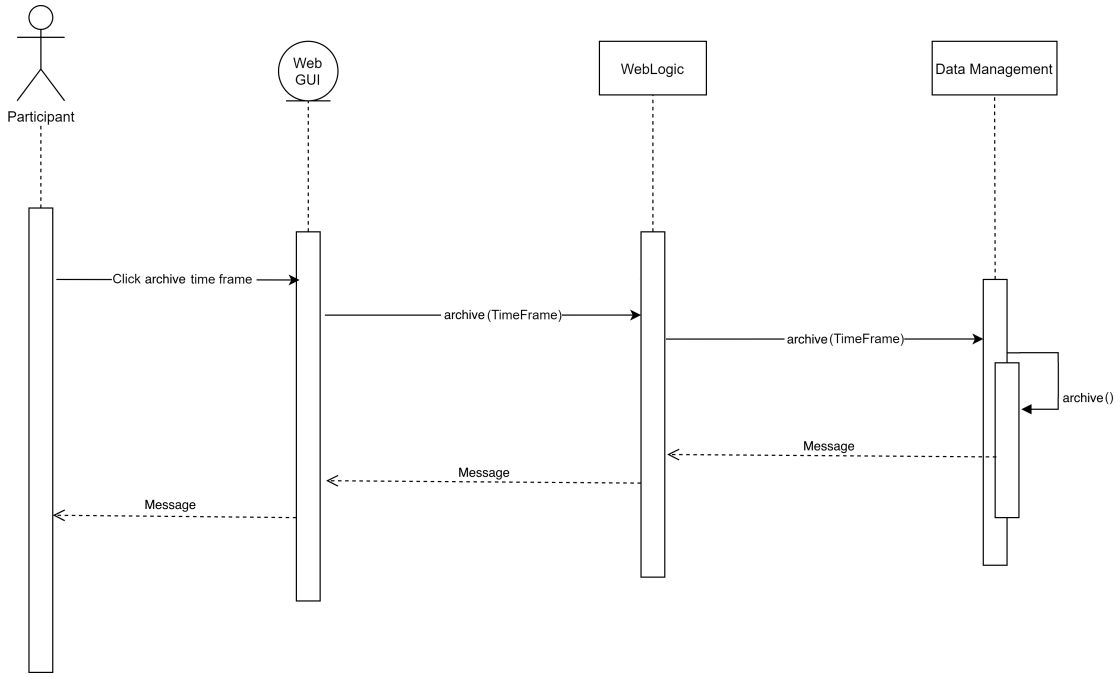
The following diagram displays the control flow of the client-invoked method call on the server when the administrator deletes a user:



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

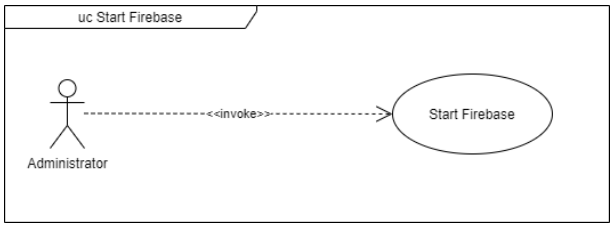


The following diagram displays the control flow of the client-invoked method call on the server when the administrator wants to archive mood data:

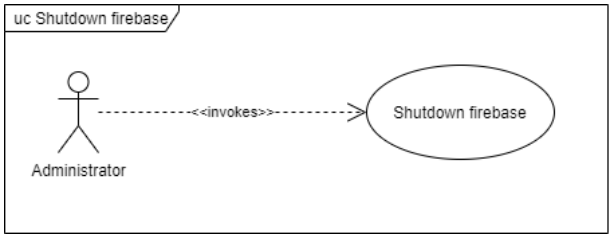


7.7 Boundary Conditions

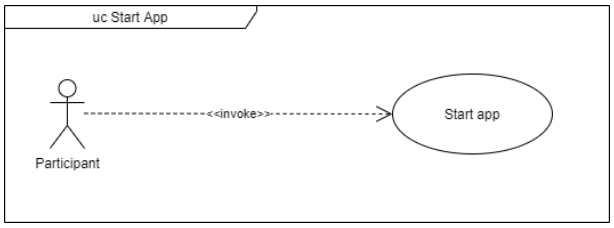
Use case name	`Start firebase`
Entry condition	the firebase server is not running
Flow of events	1. ADMINISTRATORstarts the firebase server
Exit condition	The server is online and it is possible to connect via a client to it.



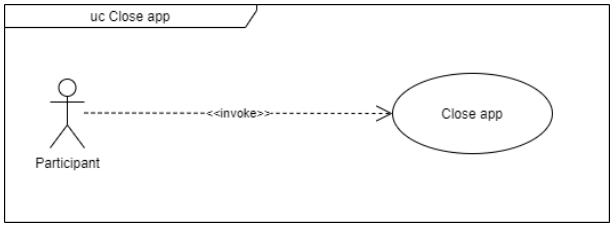
Use case name	`Shut down firebase`
Entry condition	the firebase server is running
Flow of events	1. ADMINISTRATORstops the firebase server
Exit condition	The firebase server is no longer online. A client can't connect to it or looses the connection.



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



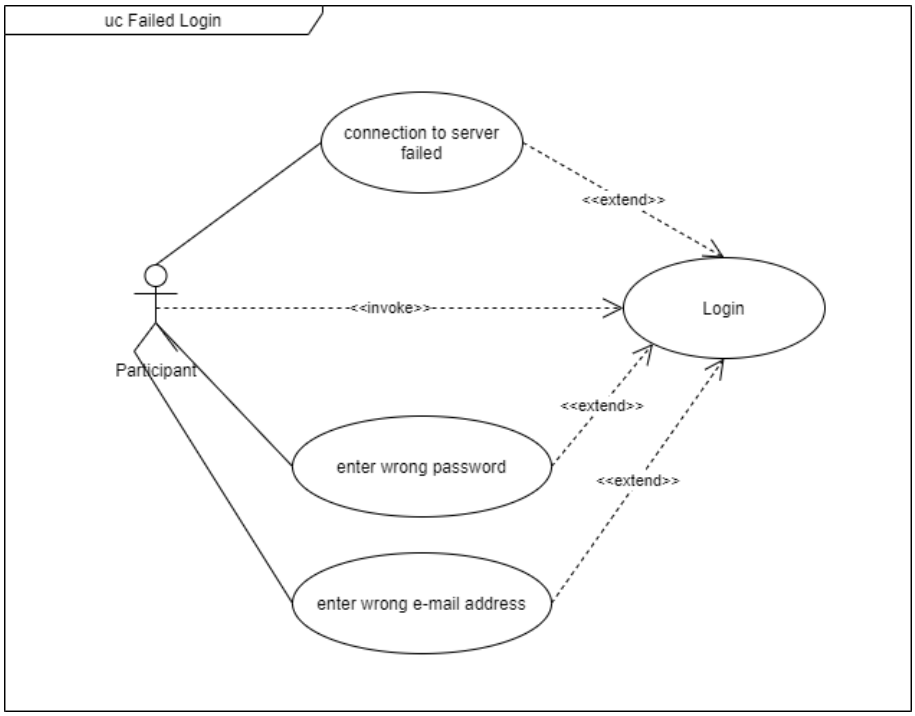
Use case name	`Close app`
Entry condition	The app is running
Flow of events	1. Participant leaves the app and removes it from the task manager 2. The app is terminated
Exit condition	Participant is no longer able to use the app



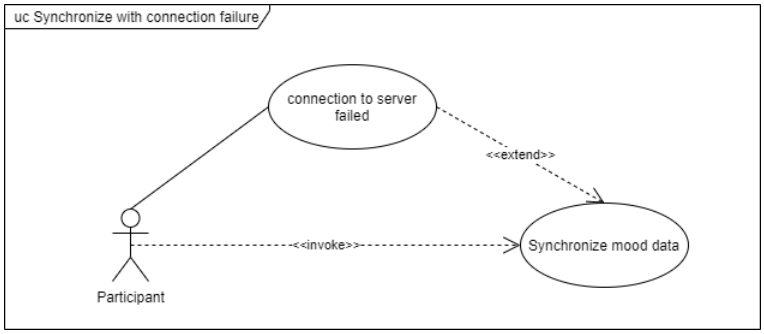
Use case name	`Login with wrong password`
Entry condition	The app is running on the smartphone
Flow of events	1. Participant enters the data and clicks on the login button 2. App connects to server and sends the entered login data 3. The server validates the login data and notices that the entered password 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.



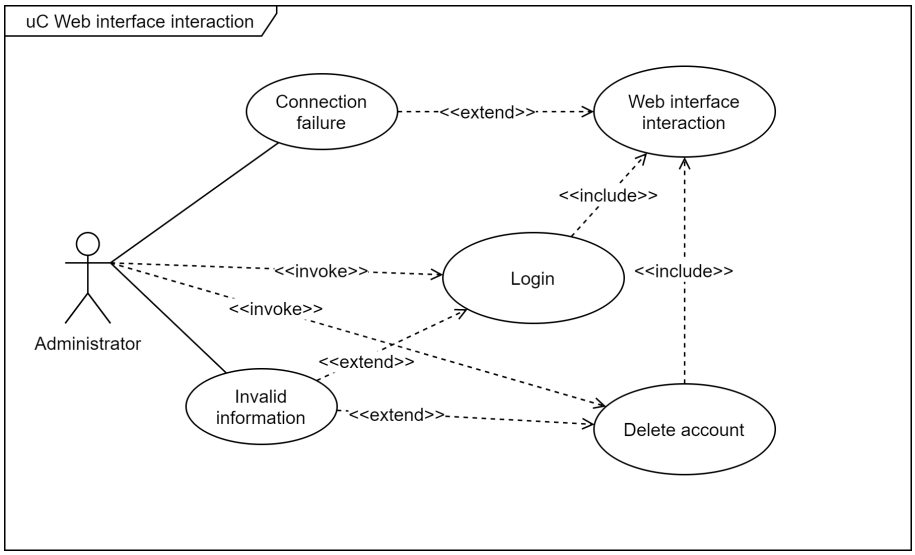
Use case name	`Synchronize with connection failure`
Entry condition	The last notification of the day has passed
Flow of events	1. The app attempts to connect to the server 2. The connection fails 3. No changes are done and the data remains locally saved 4. The app retries automatically committing the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

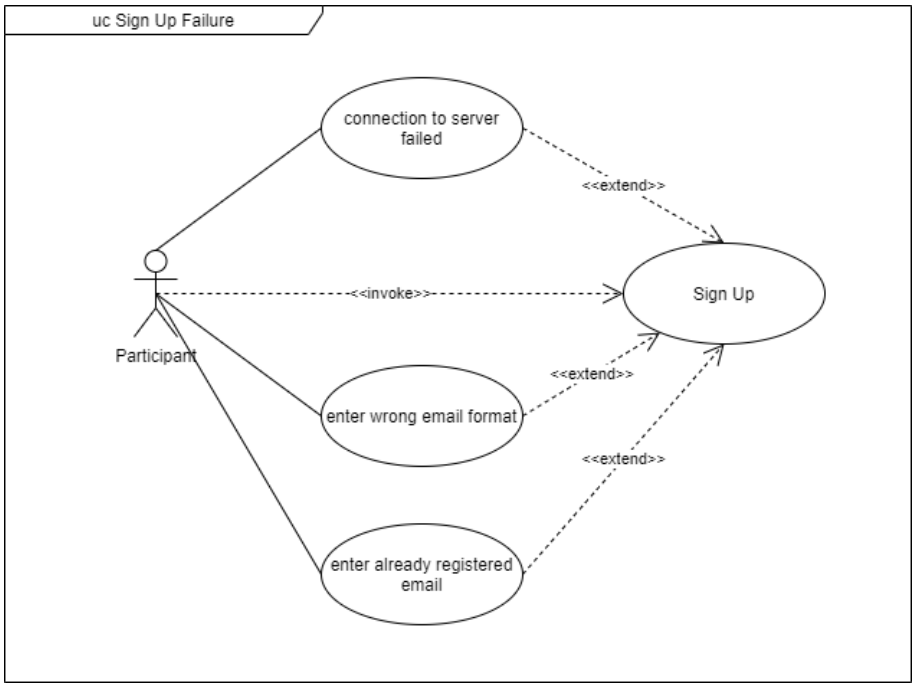
Use case name	`Delete user account with invalid information`
Entry condition	The Admin has opened the web interface and is logged in
Flow of events	1. ADMINISTRATORclicks "delete user account" button on the web interface 2. ADMINISTRATORenters invalid account information 3. Website displays warning that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



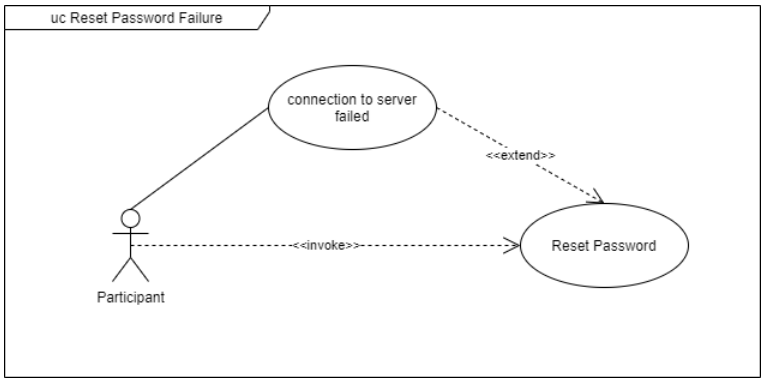
Use case name	`Sign up with wrong email format`
Entry condition	The Participant has the app installed
Flow of events	1. Participant enters email in wrong format 2. The app checks the email for the correct format and notices that the format is not correct 3. The app displays a warning and does not proceed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

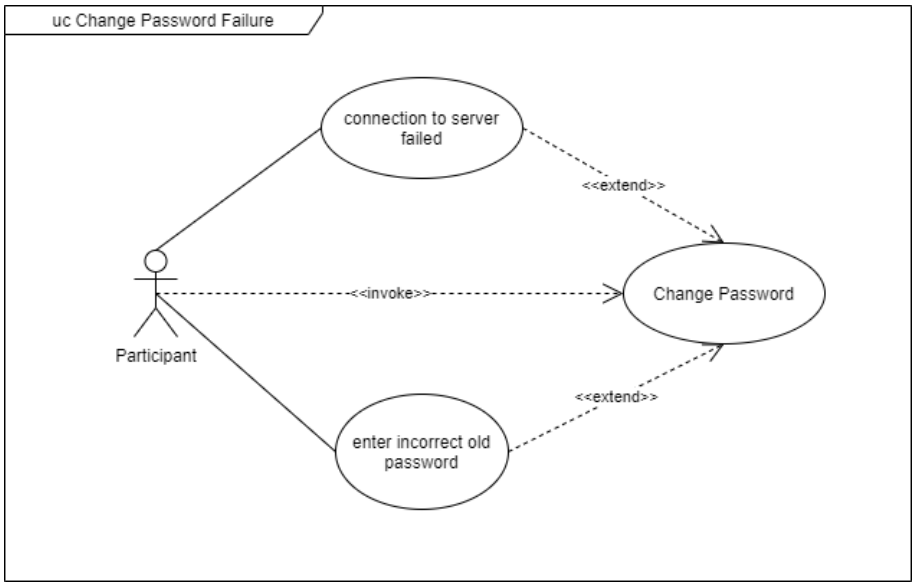


Use case name	`Reset password with connection failure`
Entry condition	The Participant has the app installed, an account and is not logged in
Flow of events	1. Participant chooses to log in 2. Participant clicks "reset password" button 3. Participant enters an email address 4. App tries to send data 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

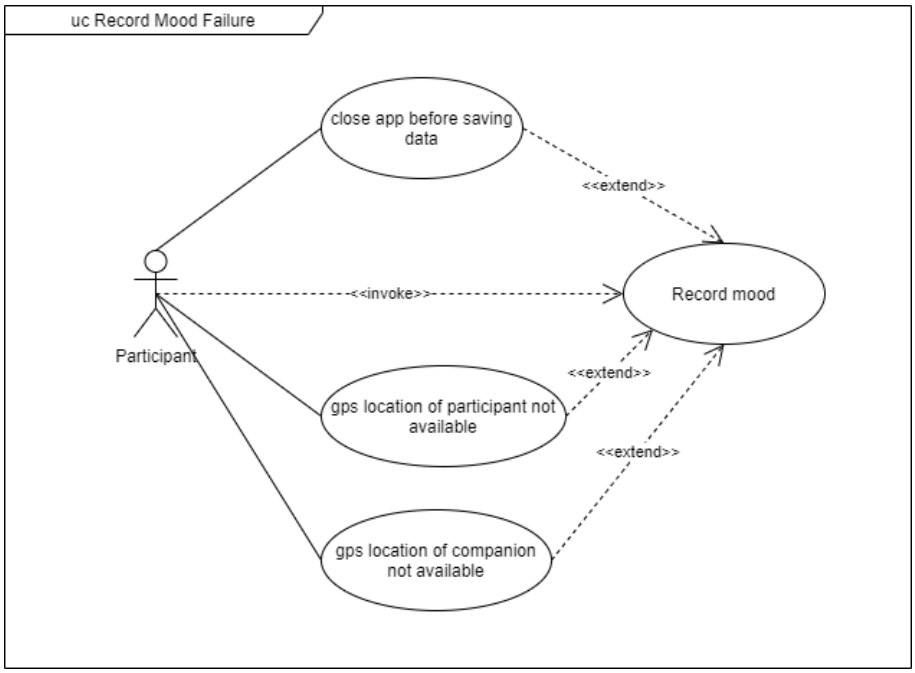


Use case name	`App gets closed during voluntary record mood`
Entry condition	The Participant is logged in
Flow of events	1. Participant clicks "record mood" button 2. Participant records his mood 3. Participant closes the app before saving the mood recording
Exit condition	The mood was not recorded

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

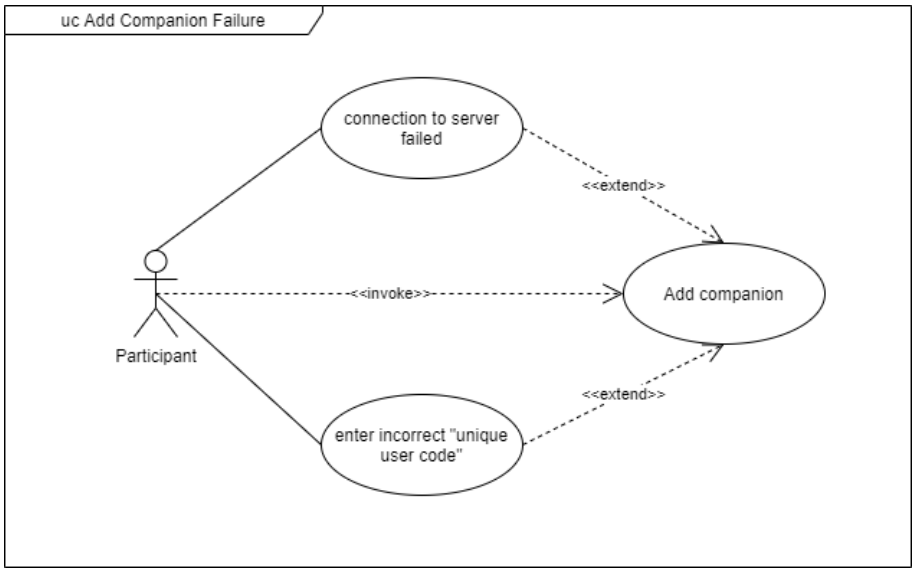
Use case name	`App can't get companion's GPS location during record mood`
Entry condition	The Participant is logged in
Flow of events	1. Participant clicks on the notification 2. Participant records his mood 3. App doesn't get GPS location of a single or multiple companions
Exit condition	-1 is noted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

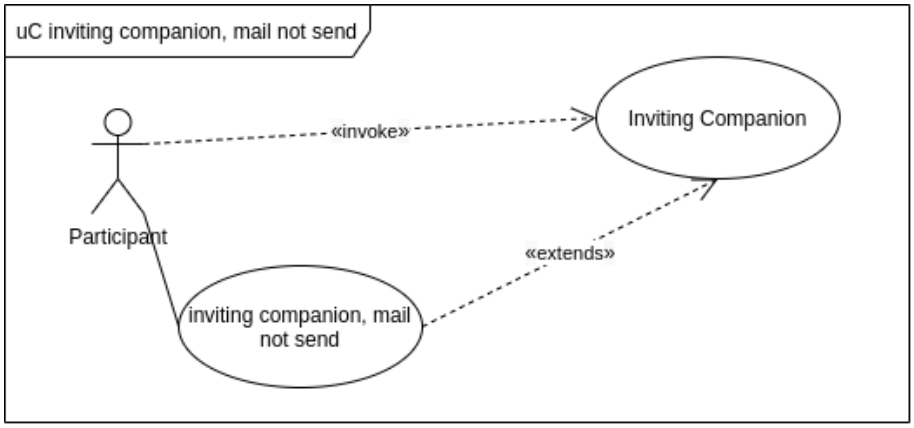


Use case name	`Adding companion with connection failure`
Entry condition	The Participant is logged in
Flow of events	1. Participant clicks on the "add Companion" button 2. Participant enters Companion code and relationship 3. App tries to send data to the server 4. App can't connect to server 5. App 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

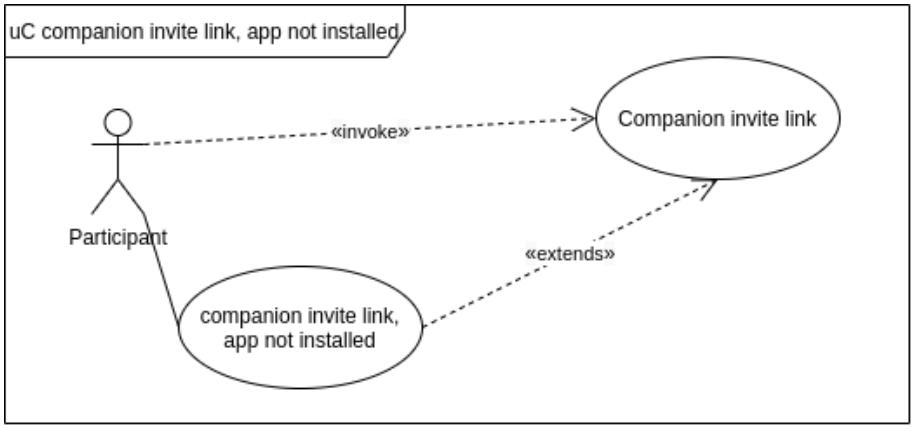
Use case name	`Adding companion with wrong "unique user code"`
Entry condition	The Participant is logged in
Flow of events	1. Participant clicks on the "add Companion" button 2. Participant enters incorrect "unique user code" and relationship 3. App connects to the server and sends the data 4. The server 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



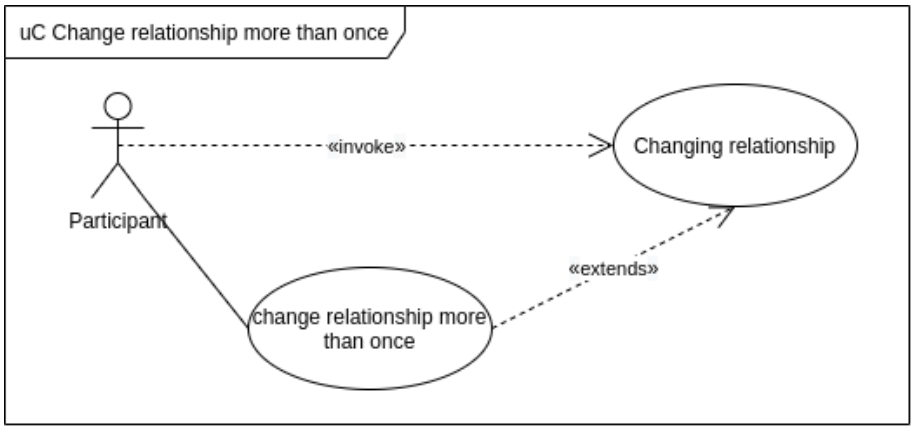
Use case name	`Inviting companion, mail not sent`
Entry condition	The Participant is logged in
Flow of events	1. Participant clicks on the "invite Companion" button 2. Participant enters relationship with invited person 3. App prompts user's email app with a message draft containing the generated links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



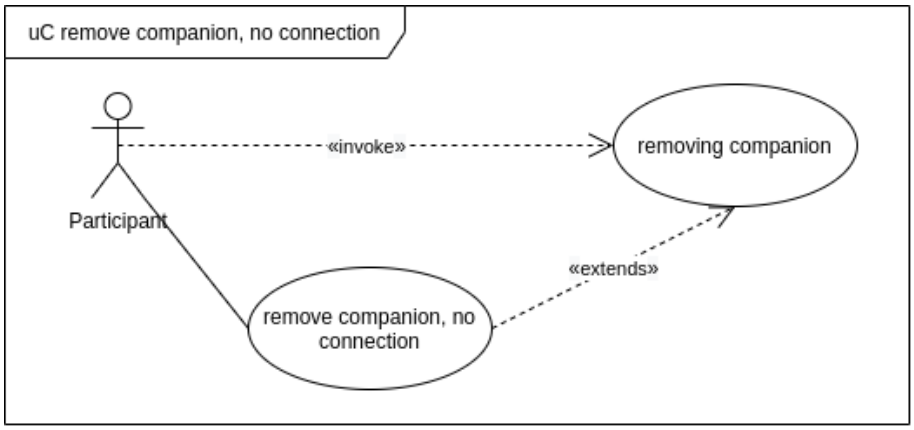
Use case name	`Companion invite link, app not installed`
Entry condition	The Participant has not installed the app by now
Flow of events	1. Participant clicks on the "add Companion" link in the invitation email they received 2. The app isn't installed, so the link is opened in the browser 3. The browser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



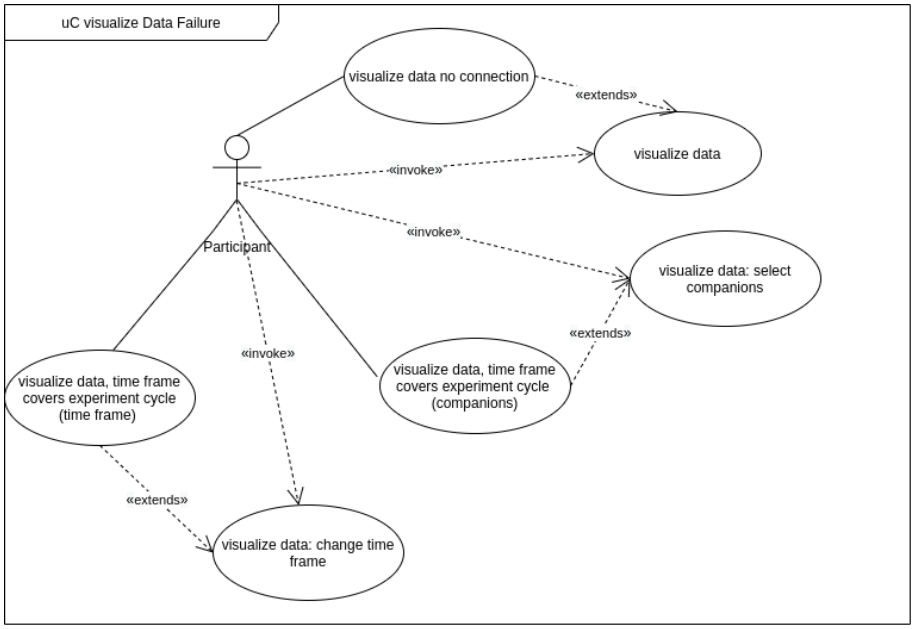
Use case name	`Remove companion, no connection`
Entry condition	The participant is logged in
Flow of events	1. Participant clicks on the "remove Companion" button on the companion inside the app 2. App attempts to connect to the server 3. App cannot connect to server 4. App displays an error message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



,

8 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.

Contents

List of Figures

Part IV

Object Design

9 Introduction

9.1 Guidelines of interface documentation

We, Group 12, use the following naming conventions that will help structuring and writing clear and readable code:

- Facades will be named with the suffix "Facade" and a prefix which describes what the Facade is for.
- Every class which includes logic will be named with the suffix "Handler" and a prefix which describes what the Handle handles.
- Classes which interact with the database will be named with the suffix "Manager" and a prefix which describes which data the Manager handles.
- Class names will be combinations of nouns.
- All parameters and methods will be given names which describes their purpose.

9.2 Literature references

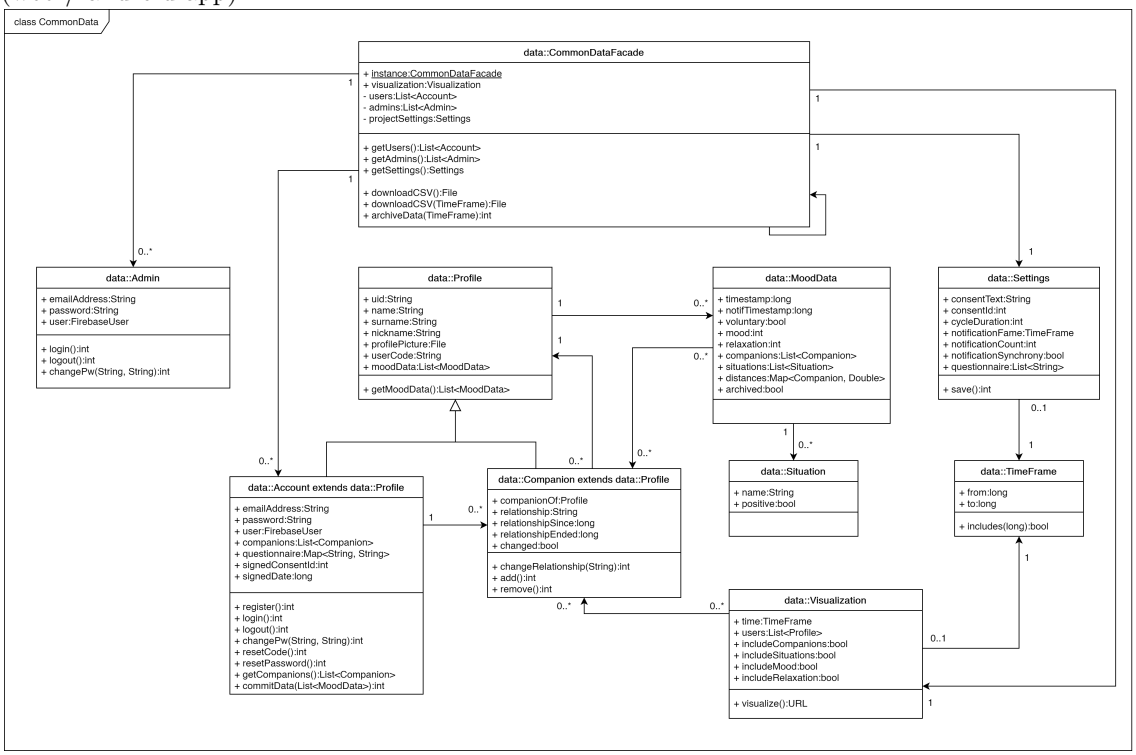
We will lightly use the example submission file "Example_D2b.pdf" as a template and our previously created documentation as reference.

10 Packages

This section defines a list of data and class models to be implemented to form the proposed system.

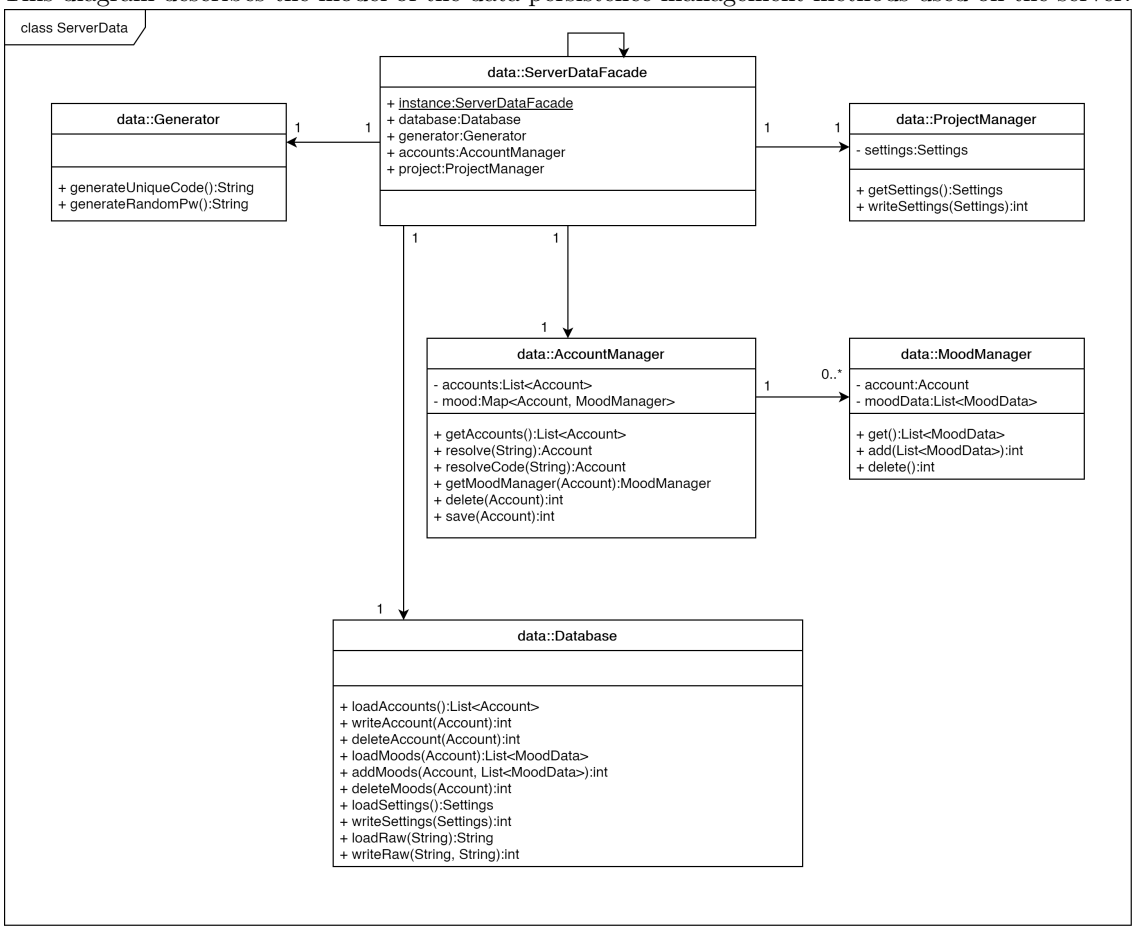
10.1 CommonData

This diagram describes the model of custom data types used both on the server and the clients (web / android app).



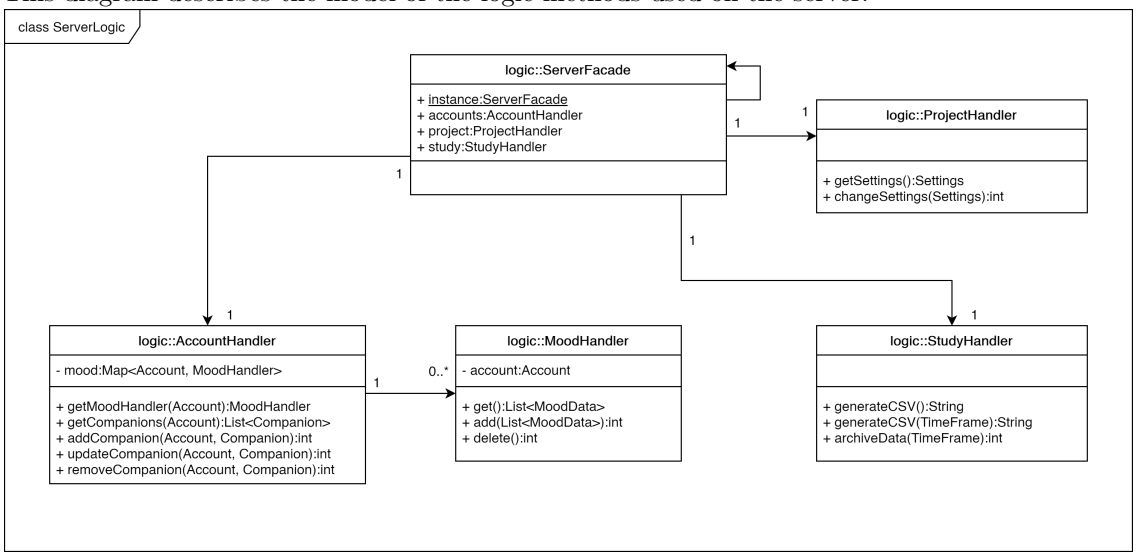
10.2 ServerData

This diagram describes the model of the data persistence management methods used on the server.



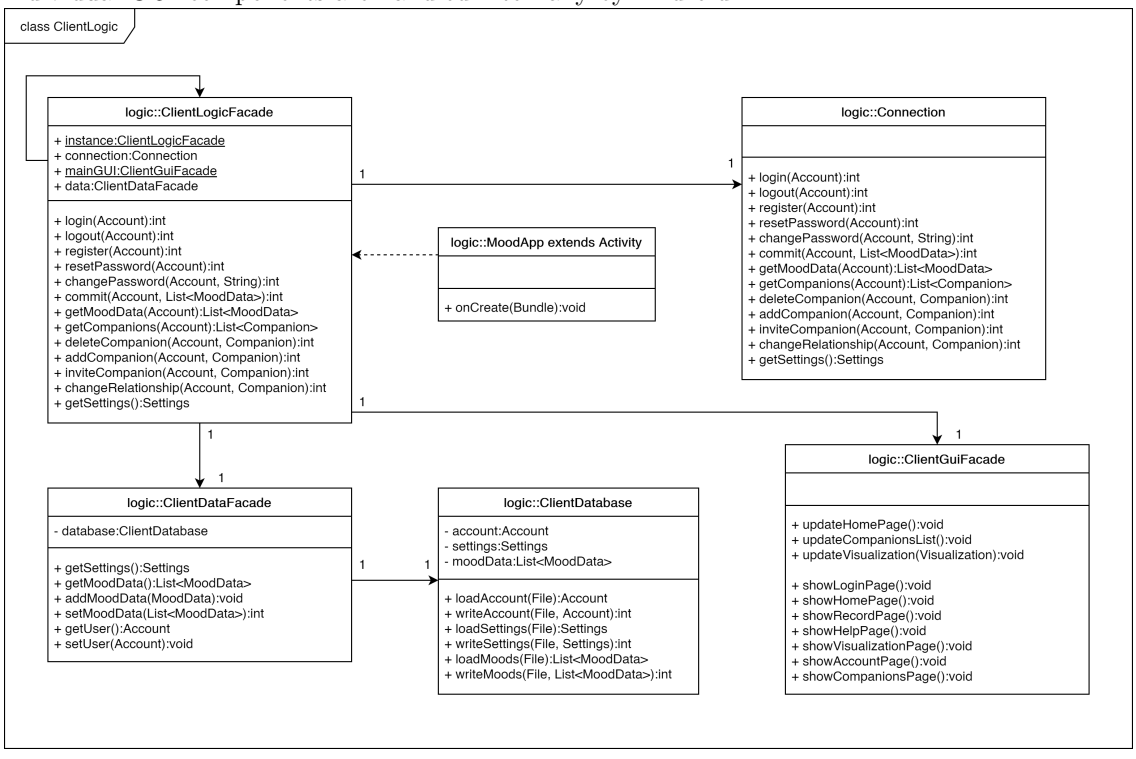
10.3 ServerLogic

This diagram describes the model of the logic methods used on the server.



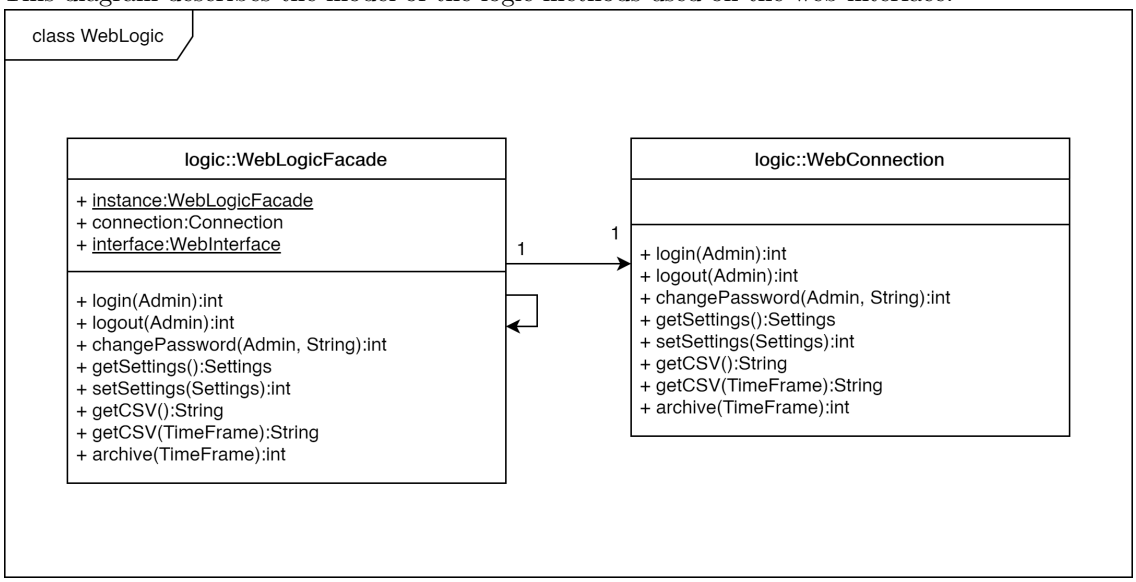
10.4 ClientLogic

This diagram describes the model of the logic methods used on the client. Additionally, since not much data is saved on the client, this diagram also describes the client database.
Furthermore, this diagram also describes the basic navigation methods for the GUI since the individual GUI components are handled internally by Android.



10.5 WebLogic

This diagram describes the model of the logic methods used on the web interface.



11 Class Interfaces

The algorithms in the following interfaces are written in Java-like pseudo code.
Many of the algorithms or methods we use are implemented by Firebase itself. These are not documented here.

11.1 CommonDataFacade

Methods within the CommonDataFacade or classes which are part of it merely call the corresponding methods on either the ClientLogic or ServerLogic, depending on which instance they belong to, with `this` as self-reference if applicable.

11.1.1 TimeFrame

Algorithm 1 includes(long time) : bool

Constraints:
post: true is returned iff the time is within the timeframe of the current object

Procedure:
return this.from <= time && this.to >= time;

11.2 ServerData

11.2.1 Database

The Database class is responsible for interacting with the Firebase Real-Time-Database. Given methods simply perform get or set operations on the corresponding fields.

11.2.2 Generator

Algorithm 2 generateUniqueCode() : String

Constraints:
post: returned value is a unique code not assigned to any account yet

Procedure:

```
String code = this.generateRandomPw();
while ServerDataFacade.instance.accounts.resolveCode(code) != null do
  code = this.generateRandomPw();
end while
return code
```

Algorithm 3 generateRandomPw() : String

Constraints:
post: a random password of 12 characters is returned

Procedure:

```
StringBuilder sb ← new StringBuilder();
for int i←0; i<12; i++ do
  sb.append((char)(Math.random() * 93 + 33));
end for
return sb.toString();
```

11.2.3 AccountManager

Algorithm 4 getAccounts() : List<Account>

Constraints:
post: a list of all accounts is returned

Procedure:
```
if this.accounts = null then
  this.accounts ← ServerDataFacade.instance.database.loadAccounts();
end if
return this.accounts;
```

Algorithm 5 resolve(String id) : Account

Constraints:
post: the account matching the id is returned

Procedure:
```
Iterator<Account> it ← this.getAccounts().iterator();
while it.hasNext() do
  Account current ← it.next();
  if current.email = id || current.uid = id then
    return current;
  end if
end while
```

Algorithm 6 resolveCode(String code) : Account

Constraints:
post: the account to the given code is returned

Procedure:
```
Iterator<Account> it ← this.getAccounts().iterator();
while it.hasNext() do
  Account current ← it.next();
  if current.userCode = code then
    return current;
  end if
end while
```

Algorithm 7 getMoodManager(Account account) : MoodManager

Constraints:
post: the mood manager for the given account is returned

Procedure:
```
if this.mood.get(account) = null then
    this.mood.put(account, new MoodManager(account));
end if
return this.mood.get(account);
```

Algorithm 8 delete(Account account) : int

Constraints:
pre: the account is known (in the list)
post: the account is removed from the list and database

Procedure:
```
int index ← this.getAccounts().indexOf(account);
this.accounts.remove(index);
this.getMoodManager(account).delete();
return ServerDataFacade.instance.database.deleteAccount(Account);
```

Algorithm 9 save(Account account) : int

Constraints:
post: the account is added to the list and database if not yet present, otherwise the account present is updated

Procedure:
```
int index ← this.getAccounts().indexOf(account);
if index < 0 then
    this.accounts.add(account);
else
    this.accounts.set(index, account);
end if
return ServerDataFacade.instance.database.writeAccount(account);
```

Algorithm 10 delete(Account account) : int

Constraints:
post: the account is removed from the list and database if known

Procedure:
```
int index ← this.getAccounts().indexOf(account);
if index < 0 then
    return -1;
end if
this.accounts.remove(index);
return ServerDataFacade.instance.database.deleteAccount(account);
```

11.2.4 MoodManager

Algorithm 11 get() : List<MoodData>

Constraints:
post: the mood data is returned

Procedure:
```
if this.moodData = null then
  this.moodData ← ServerDataFacade.instance.database.loadMoods(this.account);
end if
return this.moodData;
```

Algorithm 12 add(List<MoodData> moods) : List<MoodData>

Constraints:
post: the mood data is added to the list and database

Procedure:
```
this.get().addAll(moods);
return ServerDataFacade.instance.database.addMoods(this.account, moods);
```

Algorithm 13 delete() : int

Constraints:
post: the mood data is deleted from the list and database

Procedure:
```
for int i←this.get().size(); i>0; i– do
  this.moodData.remove(0);
end for
return ServerDataFacade.instance.database.deleteMoods(this.account);
```

11.2.5 ProjectManager

Algorithm 14 getSettings() : Settings

Constraints:
post: settings are returned

Procedure:

```
if this.settings = null then
  this.settings ← ServerDataFacade.instance.database.loadSettings();
end if
return this.settings;
```

Algorithm 15 writeSettings(Settings settings) : int

Constraints:
post: the new settings are saved and if the new consentForm

Procedure:

```
Settings old ← this.getSettings();
Database db ← ServerDataFacade.instance.database
for int i=1; i<=db.consentForms.length(); i++ do
  if db.loadRaw("consent_" + i) = settings.consentText then
    settings.consentId ← i;
    break;
  else if i = old.consentId then
    settings.consentId ← i + 1;
    db.writeRaw("consent_" + (i + 1), settings.consentText);
  end if
end for
this.settings ← settings;
db.writeSettings(this.settings);
return 20;
```

11.3 ServerLogic

11.3.1 AccountHandler

Algorithm 16 getMoodHandler(Account account) : MoodHandler

Constraints:
post: the mood handler for the account is returned

Procedure:
```
if this.mood.get(account) = null then
  this.mood.put(account, new MoodHandler(account));
end if
return this.mood.get(account);
```

Algorithm 17 getCcompanions(Account account) : List<Companion>

Constraints:
post List of Companions of Account is returned

Procedure:
```
return ServerDataFacade.instance.accounts.resolve(account.uid).companions;
```

Algorithm 18 addCompanion(Account account, Companion comp) : int

Constraints:
post Companion is added to accoun

Procedure:
```
Account current ← ServerDataFacade.instance.accounts.resolve(account.uid);
current.companions.add(comp);
return ServerDataFacade.instance.database.writeAccount(crrent);
```

Algorithm 19 updateCompanion(Account account, Companion comp) : int

Constraints:
pre: relationship has not been changed before
post: companion data is updated on the database

Procedure:

```
Account current ← ServerDataFacade.instance.accounts.resolve(account.uid);
int index ← current.companions.indexOf(comp);
if index < 0 then
  return -1;
end if
current.companions.set(index, comp);
return ServerDataFacade.instance.database.writeAccount(crrent);
```

Algorithm 20 removeCompanion(Account account, Companion comp) : int

Constraints:
pre: relationship was not terminated yet
post: relationship is terminated by providing a termination timestamp

Procedure:

```
if comp.relationshipEnded ≥ 0 then
  return -1;
end if
comp.relationshipEnded ← Date.now();
return ServerDataFacade.instance.database.writeAccount(acount);
```

11.3.2 MoodHandler

Algorithm 21 get() : List<MoodData>

Constraints:
post: MoodData is returned

Procedure:

```
return ServerDataFacade.instance.accounts.getMoodManager(this.account).get();
```

Algorithm 22 add(List<MoodData> moods) : int

Constraints:
post MoodData ist added

Procedure:
return ServerDataFacade.instance.accounts.getMoodManager(this.account).add(moods);

Algorithm 23 delete() : int

Constraints:
post: account is delted

Procedure:
return ServerDataFacade.instance.accounts.getMoodManager(this.account).delete();

11.3.3 ProjectHandler

Algorithm 24 getSettings() : Settings

Constraints:
post: settings are returned

Procedure:
return ServerDataFacade.instance.project.getSettings();

Algorithm 25 changeSettings(Settings settings) : int

Constraints:
post: settings are changed

Procedure:
return ServerDataFacade.instance.project.writeSettings(settings);

11.3.4 StudyHandler

Algorithm 26 generateCSV(TimeFrame time) : String

Constraints:
post: Returned string contains all mood data for the selected time frame with one line per user.

Definition:
This algorithm returns recorded mood data for a selected time frame with one line per user.
Each line uses the following comma-separated entries:

- ID of user
- Join date of user
- Signed consent ID of user
- Questionnaire answers in the format q1:a1&q2:a2&...
- Relationships in the format r1id:r1type:r1start:r1end&r2id:r2type:r2start:r2end&...
- Mood data block 1 in the format m1time|m1notifTime|m1mood|m1relax|m1volunt|m1archived
  |selected companions in the format c1id&c2id&...
  |GPS companion distances in the format c1id:c1distance&c2id:c2distance&...
- Mood data block 2 in the above format
- ...

Algorithm 27 generateCSV() : String

Constraints:
post: CSV with all the data is generated and returned

Procedure:
return this.generateCSV(new TimeFrame(0, Long.MAX_VALUE));

11.4 ClientLogic

11.4.1 Connection

The connection is responsible for calling the corresponding methods on the server.

11.4.2 ClientDataFacade

11.4.3 ClientLogicFacade

Algorithm 28 register(Account account) : int

Constraints:
pre: user has no account, has filled in the signup form (consent form, questionnaire, personal information, login credentials)
post: user is registered

Procedure:

```
firebaseAuth.createUserWithEmailAndPassword(account.email, account.password);
if task.isSuccessful() then
    updateUI(account);
    return ServerDataFacade.instance.accounts.save(account);
else
    return -1;
end if
```

Algorithm 29 login(Account account) : int

Constraints:
pre: user is logged out
post: user is logged in

Procedure:

```
firebaseAuth.signInWithEmailAndPassword(account.email, account.password);
if task.isSuccessful() then
    updateUI(account);
    return 1;
else
    log("Authentication failed");
    return -1;
end if
```

Algorithm 30 resetPassword(Account account) : int

Constraints:
pre: user is logged in
post: password reset email is sent

Procedure:

```
firebaseAuth.sendPasswordResetEmail(account.email);
if task.isSuccessful() then
  log("Email has been sent!");
  return 1;
else
  return -1;
end if
```

Algorithm 31 changePassword(Account account, String newPassword) : int

Constraints:
pre: user is logged in
post: password is changed

Procedure:

```
firebaseUser.updatePassword(newPassword);
if task.isSuccessful() then
  log("Password has been changed!");
  return 1;
else
  return -1;
end if
```

Algorithm 32 logOut(Account account) : int

Constraints:
pre: user is logged in
post: user is logged out

Procedure:

```
firebaseAuth.signOut();
if task.isSuccessful() then
  return 1;
else
  return -1;
end if
```

Algorithm 33 getMoodData(Account account) : List<MoodData> data

Constraints:
post: mood data is sent to the app

Procedure:

```
return getMoodHandler(account).get();
```

Algorithm 34 getCompanions(Account account) : List<Companion> companionsList

Constraints:
post: companion data is sent to the app

Procedure:
return ServerLogicFacade.instance.accounts.getCompanions(account);

Algorithm 35 deleteCompanion(Account account, Companion companion) : int

Constraints:
post: companion is deleted from users companion list

Procedure:
return ServerLogicFacade.instance.accounts.removeCompanion(account, companion);

Algorithm 36 addCompanion(Account account, Companion companion) : int

Constraints:
post: companion added to users companion list

Procedure:
return ServerLogicFacade.instance.accounts.addCompanion(account, companion);

Algorithm 37 addCompanionByCode(Account account, String friendcode, String relationship) : int

Constraints:
pre: user wants to invite the companion with the friend code
post: user with the friendcode is invited and added to the friend list

Procedure:
companion ← new Companion(ServerDataFacade.instance.accounts.resolveCode(friendcode), relationship);
return ServerLogicFacade.instance.accounts.addCompanion(account, companion);

Algorithm 38 inviteCompanion(Account account) : int

Constraints:
pre: user wants to invite a companion by email

Method:
This method generates an invite email draft containing an invite link which can be used to download the app if the recipient doesn't have the app installed yet and otherwise add the invitee as a companion. The user is prompted to send the generated email from their primary installed email client.

Algorithm 39 changeRelationship(Account account, Companion companion) : int

Constraints:
post: users relationship to companion is changed

Procedure:
return ServerLogicFacade.instance.accounts.updateCompanion(account, companion);

Algorithm 40 getSettings() : settings

Constraints:
post: user get's the settings

Procedure:
return ServerLogicFacade.instance.project.getSettings();

11.4.4 ClientDatabase

The ClientDatabase class is responsible for storing data on the user devices (phones). This data includes the currently logged in account, cached project settings and mood data.

11.4.5 ClientGuiFacade

The ClientGuiFacade class is responsible for updating and presenting the user interface of the Android app.

Algorithm 41 updateHomePage() : void

Constraints:
post: the calendar on the homepage reflects the most up-to-date data

Method:
The calendar on the homepage displays an overview of the signed in user's mood of the current month. This method reads the mood data of the user and updates the displayed calendar.

Algorithm 42 updateCompanionsList() : void

Constraints:
post: the companions list contains a list of GUI elements for each current companion

Method:
The companions of the signed in user are read from the server database if possible, else from the client cache and the GUI list is updated to contain an element for each known companion. The design will be according to the corresponding UI prototype from the first Milestone.

Algorithm 43 createVisualization() : void

Constraints:
post: the visualization frame contains a visualization matching the current options

Method:
The visualization is fetched in form of a web page and displayed in a WebView frame.
We use D3 to generate the visualizations on the page.

Algorithm 44 updateVisualization() : void

Constraints:
post: the visualization frame contains a visualization matching the current options

Method:
The visualization is fetched in form of a web page and displayed in a WebView frame.
We use D3 to generate the visualizations on the page.

Algorithm 45 showLoginPage() : void

Constraints:
pre: user is not logged in

Method:
Displays the login page, where the user can decide whether to log in to an existing account or register a new account. If they pick to create a new account, they will have to accept the consent form and fill the questionnaire first.

Algorithm 46 showHomePage() : void

Constraints:
pre: user is logged in

Method:
Displays the home page with calendar and overview.

Algorithm 47 showRecordPage() : void

Constraints:
pre: user is logged in

Method:
Displays the mood recording screen where the user can enter their current mood.

Algorithm 48 showHelpPage() : void

Constraints:
None

Method:
Displays the help page with information about the app.

Algorithm 49 showRecordPage() : void

Constraints:
pre: user is logged in

Method:
Displays the mood recording screen where the user can enter their current mood.

Algorithm 50 showVisualizationPage() : void

Constraints:
pre: user is logged in

Method:
Displays the visualization screen where the user can change parameters to view different visualizations of their own and their companions' mood history.

Algorithm 51 showAccountPage() : void

Constraints:
pre: user is logged in

Method:
Displays the account management screen where the user can change their password or log out.

Algorithm 52 showCompanionPage() : void

Constraints:
pre: user is logged in

Method:
Displays the companion list screen where the user can add, invite, change and remove companions.

11.5 WebLogic

11.5.1 WebConnection

Algorithm 53 login(Admin admin) : int

Constraints:
pre: Admin is not logged in.
post: Admin is logged in.

Procedure:

```
try{
await firebase.auth().signInWithEmailAndPassword(admin.email, admin.password);
return 1;
}
catch(Exception e){
return -1;
}
```

Algorithm 54 logout(Admin admin) : int

Constraints:
pre: Admin is logged in.
post: Admin is not logged in.

Procedure:

```
try{
await firebase.auth().signOut();
return 1;
}
catch(Exception e){
return -1;
}
```

Algorithm 55 changePassword(Admin admin, String newPw) : int

Constraints:
pre: Admin is logged in.
post: Password is changed.

Procedure:

```
try{
await firebase.auth().currentUser.updatePassword(newPw);
return 1;
}
catch(Exception e){
return -1;
}
```

Algorithm 56 getSettings() : String

Constraints:
pre: Admin is logged in.
post: Settings in the DB are synchronized with the ones shown in the Web App (overwrite local data).

Procedure:

```
firebase.database().ref('Settings').once('value').then(;
return snapshot.val();
);
```

Algorithm 57 setSettings(setting, value) : int

Constraints:
pre: Admin is logged in.
post: Settings in the DB are synchronized with the ones entered in the Web App (overwrite server data).

Procedure:

```
await firebase.database().ref('Settings').update(
setting: value
return 1;
);
catch(Exception e){
return -1;
}
```

Algorithm 58 getCSV() : String

The following algorithm will be implemented with Firebase web functions:
Constraints:
pre: Admin is logged in.
post: Computer gets CSV data from cloud function page.

Procedure:

```
window.open('https://us-central1-angrynerds-dac9e.cloudfunctions.net/getCSV');
```

Algorithm 59 getCSV(timeframeStart, timeframeEnd) : void

The following algorithm will be implemented with Firebase web functions:

Constraints:

pre: Admin is logged in.

post: Computer gets CSV data from cloud function page.

post: Only data between the selected dates is returned.

Procedure:

```
window.open('https://us-central1-angrynerds-dac9e.cloudfunctions.net/getCSV?timestamp1=' +
timeframeStart + '&timestamp2=' + timeframeEnd);
```

Algorithm 60 archive(timeframeStart, timeframeEnd) : String

The following algorithm will be implemented with Firebase web functions:

Constraints:

pre: Admin is logged in.

post: Entries within the specified timeframe get an archived flag.

Procedure:

```
window.open('https://us-central1-angrynerds-dac9e.cloudfunctions.net/archiveData?timestamp1='
+ timeframeStart + '&timestamp2=' + timeframeEnd');
```

12 Third-party libraries

- Firebase
  - Authorization
  - Database
  - Functions
- Node.js
- jQuery
- d3
- java.util.*
- Bootstrap

13 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 consent 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.

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 after 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.

DB : *Database*

Part V

System and Client Acceptance Tests

14 Client acceptance tests

14.1 Introduction

For the execution of the test we will use 2 android phones (AP1, AP2) and one laptop (L1). AP1 and AP2 will be used to test the functions of the users in the application and L1 will be used to test the web interface.

14.1.1 Hardware and Test-Software

The different hardware on which we will run the tests is capable to support the demand of our system. The operating system of L1 is Windows 10 with the latest version of Google Chrome and Firefox browser installed where we will run our web interface of the application. AP1 and AP2 are running on Android 8+ with our application installed.

14.1.2 Test data

The application will be tested with the standard start configuration without predefined data.

14.1.3 Personnel required

All members of the group will take part in the acceptance tests, as specified in the according test suite.

14.2 Acceptance criteria

14.2.1 Criteria for acceptance / rejection

Acceptable criteria: The test case returns the predefined result for that specific test.
Unacceptable criteria: The test case returns an unexpected result or error for that specific test.

14.2.2 Interruptions

Unacceptable criteria that are not caused by the system.

14.2.3 Continuation of the test

The test case should be restarted and tested again.

14.2.4 Testcases Component 1

14.2.4.1 Testsuite: Android application user registration and login

14.2.4.2 Test Case C1-1

Input	The user inputs the info to create a new user and clicks the button to sign-in
Operation	Create a new user
Output	User account is registered in DB

14.2.4.3 Test Case C1-2

Input	username, password
Operation	Log in to account
Output	User is logged into the application

14.2.4.4 Test Case C1-3

Input	password, new password
Operation	Change password
Output	The password of the user is changed

14.2.4.5 Test Case C1-4

Input	email adress
Operation	Reset password
Output	The password of the user is changed and the new password is sent to the email adress given by the user

14.2.4.6 Test Case C1-5

Input	logout request
Operation	Log out
Output	The currently logged in user is logged out of the system

14.2.4.7 Test Case C1-6

Input	delete user request
Operation	Delete user
Output	A selected user is deleted from the system

14.2.5 Testcases Component 2

14.2.5.1 Testsuite: Android application mood recording

14.2.5.2 Test Case C2-1

Input	mood on a scale of 1-5, relaxation on a scale of 1-5, companions that are near, special situations
Operation	Mood recording
Output	The mood of the user is recorded and locally stored on the android device

14.2.5.3 Test Case C2-2

Input	accept notification
Operation	Notification prompt
Output	The user is directed to the mood recording after clicking on the notification

14.2.6 Testcases Component 3

14.2.6.1 Testsuite: Android application companion/friend management

14.2.6.2 Test Case C3-1

Input	unique user code
Operation	Adding companion
Output	The companion is added to the friend list of the user

14.2.6.3 Test Case C3-2

Input	relationship to the invited person
Operation	Inviting companion
Output	An invitation email is send to the invited user

14.2.6.4 Test Case C3-3

Input	new relationship status
Operation	Changing relationship status to companion
Output	The current relationship status to the friend is changed

14.2.6.5 Test Case C3-4

Input	removing companion operation
Operation	Remove companion from companions list
Output	The companion which has to be removed is removed from the companions list

14.2.7 Testcases Component 4

14.2.7.1 Testsuite: Android application data visualization

14.2.7.2 Test Case C4-1

Input	time frame, data to visualize, up to 5 companions whose mood and relaxation will be included in the visualization.
Operation	Visualize mood data
Output	A visualization with the user defined parameters

14.2.8 Testcases Component 5

14.2.8.1 Testsuite: Android application help page

14.2.8.2 Test Case C5-1

Input	help page request
Operation	Show help page
Output	The help page is prompted to the user

14.2.9 Testcases Component 6

14.2.9.1 Testsuite: Web interface

14.2.9.2 Test Case C6-1

Input	admin, admin-password
Operation	Log in to Web interface
Output	Admin is logged into the Web interface

14.2.9.3 Test Case C6-2

Input	admin-password, new admin-password
Operation	Change admin-password
Output	The password of the admin is changed

14.2.9.4 Test Case C6-3

Input	change settings request
Operation	Change project settings
Output	The changes made by the administrator are saved on the server

14.2.9.5 Test Case C6-4

Input	delete user request
Operation	Delete user
Output	All account data for the specified account is deleted from the server

14.2.9.6 Test Case C6-5

Input	download data request
Operation	Download data
Output	A csv file is generated and downloaded

14.2.9.7 Test Case C6-6

Input	archive data request
Operation	Archive data
Output	Changes are saved on the server

14.2.9.8 Test Case C6-7

Input	visualize data request
Operation	Visualize data
Output	The data is visualized in a graph

15 System Tests

15.1 Introduction

15.1.1 Purpose of the tests

The main purpose of the system tests is to test the whole system, and thus discover all possible errors that may occur.

15.1.2 Test coverage area

The complete system will be tested.

15.2 Test environment

15.2.1 Introduction

The test environment is based on the systems we will use to create the system.

15.2.2 Hardware and Test-Software

The hardware on which we are going to run the system tests is capable to support the demand of the system. We will use Android 8+ for the mobile app and modern browsers (Chrome, Firefox) for the web interface.

15.2.3 Test-Files

The program will be tested with a test database. We will use several accounts for different test sections.

15.2.4 Manpower requirements

All members of the group will take part in the system tests. We will, separately and together, test the different aspects of the system as specified by the according test suites.

15.3 Acceptance criteria

15.3.1 Acceptable and unacceptable criteria

Acceptable criteria: The test case returns the specified result for that case. Unacceptable criteria: The test case returns an error, due to a function or component which has not returned the specified result.

15.3.2 Interrupts

Unacceptable criteria that are not caused by the system

15.3.3 Resumption of test

The test section should be executed over again.

15.4 Test section 1: User accounts

15.4.1 Introduction

Test section 1 from the system tests is going to validate if the registration, login and logout work correctly.

15.4.2 Test suites

15.4.2.1 Test suite: Registration tests

15.4.2.2 Test Case TG1-1-1

Entry conditions	User has app installed and is not logged in
Input	Valid account information
Operation	Registration
Output	Account created on server
Exit conditions	Successfully created account

15.4.2.3 Test Case TG1-1-2

Entry conditions	User has app installed and is not logged in
Input	Account information with invalid email
Operation	Registration
Output	Error message "Invalid email address"
Exit conditions	Account is not created

15.4.2.4 Test suite: Log in tests

15.4.2.5 Test Case TG1-2-1

Entry conditions	User has app installed and is not logged in
Input	Valid account credentials
Operation	Log in
Output	Account is saved on device
Exit conditions	User is logged in

15.4.2.6 Test Case TG1-2-2

Entry conditions	User has app installed and is not logged in
Input	Invalid account credentials
Operation	Log in
Output	Error message "Invalid credentials"
Exit conditions	User is not logged in

15.4.2.7 Test Case TG1-2-3

Entry conditions	User has app installed and is not logged in
Input	Valid email address
Operation	Reset password
Output	Message "If this account exists, an email was sent"
Exit conditions	Email with reset link was sent

15.4.2.8 Test Case TG1-2-4

Entry conditions	User has app installed and is not logged in
Input	Invalid email address
Operation	Reset password
Output	Message "If this account exists, an email was sent"
Exit conditions	No email was sent

15.4.2.9 Test suite: Log out tests

15.4.2.10 Test Case TG1-3-1

Entry conditions	User has app installed and is logged in
Input	-
Operation	Log out
Output	Account is removed from the device
Exit conditions	User is logged out

15.4.2.11 Test suite: Password changing tests

15.4.2.12 Test Case TG1-4-1

Entry conditions	User has app installed and is logged in
Input	Correct old and new password
Operation	Change password
Output	Changes are reflected on the server
Exit conditions	Password is changed

15.4.2.13 Test Case TG1-4-2

Entry conditions	User has app installed and is logged in
Input	Wrong old and new password
Operation	Change password
Output	Error message "Wrong current password"
Exit conditions	No changes are done

15.5 Test section 2: Recording mood

15.5.1 Introduction

Test section 2 from the system tests is going to validate if recording mood both voluntarily and by notifications including all features work correctly.

15.5.2 Test suites

15.5.2.1 Test suite: Record mood tests

15.5.2.2 Test Case TG2-1-1

Entry conditions	User has app installed and is logged in
Input	Mood, relaxation, near companions, special situations
Operation	Record mood with special situations (voluntary)
Output	Data is saved on the device
Exit conditions	Data is successfully saved

15.5.2.3 Test Case TG2-1-2

Entry conditions	User has app installed and is logged in
Input	Mood, relaxation, near companions
Operation	Record mood without special situations (only by notification)
Output	Data is saved on the device
Exit conditions	Data is successfully saved

15.5.2.4 Test Case TG2-1-3

Entry conditions	User has app installed and is logged in
Input	Last notification of the day
Operation	Automatic data commit attempt
Output	Data is saved on the server
Exit conditions	Data is successfully commited

15.5.2.5 Test suite: Notification tests

15.5.2.6 Test Case TG2-2-1

Entry conditions	User has app installed and is logged in
Input	First notification prompt of the day
Operation	Accept notification
Output	Record mood screen
Exit conditions	User is prompted to record mood with special situations

15.5.2.7 Test Case TG2-2-2

Entry conditions	User has app installed and is logged in
Input	Not first notification prompt of the day
Operation	Accept notification
Output	Record mood screen
Exit conditions	User is prompted to record mood without special situations

15.5.2.8 Test Case TG2-2-3

Entry conditions	User has app installed and is logged in
Input	Notification prompt
Operation	Miss notification
Output	Data with miss flag is saved on the device upon next notification
Exit conditions	Data is successfully saved

15.5.2.9 Test Case TG2-2-4

Entry conditions	User has app installed and is logged in
Input	Notification prompts throughout a day
Operation	Miss all notification prompts
Output	User is sent an email reminding them that they missed all notifications for a full day
Exit conditions	Email is successfully sent

15.6 Test section 3: Companions

15.6.1 Introduction

Test section 3 from the system tests is going to validate if adding, removing, changing and inviting companions via Email work correctly.

15.6.2 Test suites

15.6.2.1 Test suite: Adding and inviting companion tests

15.6.2.2 Test Case TG3-1-1

Entry conditions	User has app installed and is logged in, user knows their companion's unique code
Input	Relation, valid unique code
Operation	Add companion by code
Output	Data is saved on the server
Exit conditions	Companion is successfully added

15.6.2.3 Test Case TG3-1-2

Entry conditions	User has app installed and is logged in, user knows thier companion's unique code
Input	Relation, invalid unique code
Operation	Add companion by code
Output	Error message "Invalid user code"
Exit conditions	No changes are done

15.6.2.4 Test Case TG3-1-3

Entry conditions	User has app installed and is logged in
Input	Relation
Operation	Invite by email
Output	User is prompted with a generated email draft
Exit conditions	User is able to send email

15.6.2.5 Test Case TG3-1-4

Entry conditions	User has the app installed
Input	Invitation link
Operation	Accept invite
Output	User is prompted for relationship
Exit conditions	Companion can be added

15.6.2.6 Test Case TG3-1-5

Entry conditions	User doesn't have the app installed
Input	Invitation link
Operation	Accept invite
Output	User is taken to the Google Play page
Exit conditions	User can download app

15.6.2.7 Test suite: Changing and removing companion tests

15.6.2.8 Test Case TG3-2-1

Entry conditions	User has app installed and is logged in, user has not changed the relation before
Input	Companion, new relation
Operation	Change relation
Output	Changes are saved on the server
Exit conditions	Relation is successfully changed

15.6.2.9 Test Case TG3-2-2

Entry conditions	User has app installed and is logged in, user has changed the relation before
Input	Companion, new relation
Operation	Change relation
Output	Error message "Relation with this user cannot be changed again"
Exit conditions	No changes are done

15.6.2.10 Test Case TG3-2-3

Entry conditions	User has app installed and is logged in, user has companion on their list
Input	Companion
Operation	Remove companion
Output	Data is saved on the server
Exit conditions	Companion is successfully removed

15.7 Test section 4: Visualize data and help

15.7.1 Introduction

Test section 4 from the system tests is going to validate if the dynamic data visualization and displaying of the help page work correctly.

15.7.2 Test suites

15.7.2.1 Test suite: Visualize own data tests

15.7.2.2 Test Case TG4-1-1

Entry conditions	User has app installed and is logged in, no companions are selected
Input	New time window
Operation	Change time window
Output	Visualization graph is updated
Exit conditions	New time window is visualized

15.7.2.3 Test Case TG4-1-2

Entry conditions	User has app installed and is logged in
Input	Select data (mood, relaxation, companions, situations)
Operation	Change visualized data
Output	Visualization graph is updated
Exit conditions	Changes are visualized

15.7.2.4 Test suite: Visualize companion data tests

15.7.2.5 Test Case TG4-2-1

Entry conditions	User has app installed and is logged in, time window does not cover current experiment cycle
Input	Select companion
Operation	Visualize companion data
Output	Visualization graph is updated
Exit conditions	Companion data is visualized

15.7.2.6 Test Case TG4-2-2

Entry conditions	User has app installed and is logged in, time window covers current experiment cycle
Input	Select companion
Operation	Visualize companion data
Output	Error message "Time window covers current experiment cycle with this companion"
Exit conditions	No changes are done

15.7.2.7 Test Case TG4-2-3

Entry conditions	User has app installed and is logged in, time window does not cover current experiment cycle
Input	New time window
Operation	Change time window
Output	Visualization graph is updated
Exit conditions	New time window is visualized

15.7.2.8 Test Case TG4-2-4

Entry conditions	User has app installed and is logged in, time window covers current experiment cycle
Input	New time window
Operation	Change time window
Output	Error message "Time window covers current experiment cycle with selected companions"
Exit conditions	No changes are done

15.7.2.9 Test suite: Help page

15.7.2.10 Test Case TG4-3-1

Entry conditions	User has app installed and is logged in
Input	-
Operation	Open help page
Output	User is shown the help page
Exit conditions	User is shown the help page

15.8 Test section 5: Web interface

15.8.1 Introduction

Test section 5 from the system tests is going to validate if the web interface works correctly.

15.8.2 Test suites

15.8.2.1 Test suite: Admin log in tests

15.8.2.2 Test Case TG5-1-1

Entry conditions	Admin has access to the web interface
Input	Valid admin account credentials
Operation	Admin log in
Output	Admin account data is saved in cache
Exit conditions	Admin is logged in

15.8.2.3 Test Case TG5-1-2

Entry conditions	Admin has access to the web interface
Input	Invalid admin account credentials
Operation	Admin log in
Output	Error message "Invalid credentials"
Exit conditions	Admin is not logged in

15.8.2.4 Test suite: Change admin password tests

15.8.2.5 Test Case TG5-2-1

Entry conditions	Admin is logged in to the web interface
Input	Correct old and new password
Operation	Change admin password
Output	Changes are saved on the server
Exit conditions	Admin password is changed

15.8.2.6 Test Case TG5-2-2

Entry conditions	Admin is logged in to the web interface
Input	Wrong old and new password
Operation	Change admin password
Output	Error message "Wrong current password"
Exit conditions	No changes are done

15.8.2.7 Test suite: Project settings tests

15.8.2.8 Test Case TG5-3-1

Entry conditions	Admin is logged in to the web interface
Input	New consent text
Operation	Change consent text
Output	Changes are saved on the server
Exit conditions	Consent text is changed, old consent text is archived

15.8.2.9 Test Case TG5-3-2

Entry conditions	Admin is logged in to the web interface
Input	New frequency options
Operation	Change notification frequency
Output	Changes are saved on the server
Exit conditions	Notification frequency is changed

15.8.2.10 Test Case TG5-3-3

Entry conditions	Admin is logged in to the web interface
Input	Check / uncheck notification synchrony
Operation	Change notification synchrony
Output	Changes are saved on the server
Exit conditions	Notification synchrony is changed

15.8.2.11 Test Case TG5-3-4

Entry conditions	Admin is logged in to the web interface
Input	New cycle duration in days
Operation	Change experiment cycle
Output	Changes are saved on the server
Exit conditions	Experiment cycle duration is changed

15.8.2.12 Test suite: Manage user accounts tests

15.8.2.13 Test Case TG5-4-1

Entry conditions	Admin is logged in to the web interface
Input	User credentials (id, email)
Operation	Delete user account
Output	All user data is removed from the server
Exit conditions	User account is deleted

15.8.2.14 Test suite: Download and archive data tests

15.8.2.15 Test Case TG5-5-1

Entry conditions	Admin is logged in to the web interface
Input	-
Operation	Download all data
Output	CSV file with all user data
Exit conditions	Data is downloaded

15.8.2.16 Test Case TG5-5-2

Entry conditions	Admin is logged in to the web interface
Input	Time window
Operation	Download data by time window
Output	CSV file with user data for time window
Exit conditions	Data is downloaded

15.8.2.17 Test Case TG5-5-3

Entry conditions	Admin is logged in to the web interface
Input	Time window
Operation	Archive data by time window
Output	Data is flagged as archived on the server
Exit conditions	Data is marked as archived

15.8.2.18 Test suite: Web data visualization tests

15.8.2.19 Test Case TG5-6-1

Entry conditions	Admin is logged in to the web interface
Input	Time window
Operation	Change time window
Output	Visualization graph is updated
Exit conditions	New time window is visualized

15.8.2.20 Test Case TG5-6-2

Entry conditions	Admin is logged in to the web interface
Input	Select data (mood, relaxation, companions, situations)
Operation	Change visualized data
Output	Visualization graph is updated
Exit conditions	Changes are visualized

15.8.2.21 Test Case TG5-6-3

Entry conditions	Admin is logged in to the web interface
Input	User (id, email)
Operation	Change user to visualize
Output	Visualization graph is updated
Exit conditions	New user data is visualized

Part VI

Integration Tests

We will approach the integration testing via bottom-up technique : we will first test the data layers, then the logic layers and lastly the GUI.

15.9 CommonData

15.9.1 Testing of the CommonDataFacade Class includes:

- get*(); * stands for Users,Admins,Settings

The tests are successful if the correct values (Accounts,Admins and Settings) are returned.

- archiveData(TimeFrame)

The test is successful if all data of the correct time frame is stored in the database.

- downloadCSV();with or without a TimeFrame

The tests are successful if a correct CSV file for the correct time frame is downloaded.

15.9.2 Testing of the Admin Class includes:

- login()

The test is successful if the admin is logged in to the system with the correct credentials.

- logout()

The test is successful if the admin is logged out of the system.

- changePw(String,String)

The test is successful if,given a correct old and valid new password, the admin password is changed.

15.9.3 Testing of the Account class includes:

- register()

The test is successful if the user is registered in the system.

- login()

The test is successful if the user is logged in to the system with the correct credentials.

- logout()

The test is successful if the user is logged out of the system.

- changePw(String,String)

The test is successful if, given a correct old and valid new password, the user password is changed.

- resetCode()

The test is successful if the user's unique code is resetted and replaced with a new one.

- resetPassword()

The test is successful if an email with a password reset link is sent to the users email.

- commitData(List<MoodData>)

The test is successful if the list of MoodData is commited to the server.
- getCompanions()

The test is successful if the correct companion list is returned.

15.9.4 Testing of the Profile class includes:

- getMoodData()

The test is successful if the correct MoodData list is returned.

15.9.5 Testing of the Companion class includes:

- changeRelationship(String)

The test is successful if the relationship status in the databse is changed correctly.
- add()

The test is successful if the companion is added to the users companions list.
- remove()

The test is successful if the selected companion is flagged, as not being a companion anymore, in the users companions list in the database.

15.9.6 Testing of the Visualization class includes:

- visualize()

The test is successful if an URL with the visualization is returned. The visualization itself should be correct too.

15.9.7 Testing of the Settings class includes:

- save()

The test is successful if the settings are correctly stored in the database.

15.9.8 Testing of the TimeFrame class includes:

- includes(long)

The test is successful if the test returns the correct boolean value for the given time frame.

15.10 ServerData

15.10.1 Testing of the Database Class includes:

- load*(); *stands for: Accounts, Moods, Settings, Raw
- write*(); *stands for: Account, Settings, Raw
- delete*(); *stands for: Account,Moods
- addMoods()

The tests on the database are successful if the values of the objects are loaded, written, deleted or added correctly.

15.10.2 Testing of the AccountManager Class includes:

- get*(); *stands for: Accounts, MoodManager
- resolve(String);
- resolveCode(String)
- save(Account)

The tests on the AccountManager are successful if the correct values (Account and MoodManager) are returned and the correct Account is saved.

15.10.3 Testing of the MoodManager Class includes:

- get()
- add(List<MoodData>)
- delete()

The tests on the MoodManager are successful if the correct values (Moods) are returned, added to the database or deleted.

15.10.4 Testing of the ProjectManager Class includes:

- getSettings()
- writeSettings(Settings)

The tests on the ProjectManager are successful if the correct setting values are returned, or written to the database.

15.10.5 Testing of the Generator Class includes:

- generateUniqueCode()
- generateRandomPw()

The tests on the Generator are successful if a unique user code or random password are generated correctly.

15.11 ServerLogic

15.11.1 Testing of the AccountHandler Class includes:

- get*(Account); *stands for: MoodHandler, Companions

The tests are successful if the correct values are returned.

- addCompanion(Account, Companion)

The test is successful if the requested companion object is added to the correct user account.

- updateCompanion(Account,Companion)

The test is successful if all the values of the user's companion object are updated correctly.

- removeCompanion(Account,Companion)

The test is successful if the selected companion is flagged as not being a companion anymore in the users companions list in the database.

15.11.2 Testing of the MoodHandler Class includes:

- get()
- add()
- delete()

The tests on the MoodHandler are successful if the correct values (Moods) are returned, added to the database or deleted.

15.11.3 Testing of the StudyHandler Class includes:

- generateCSV(); with selected time frame or without.

The test is successful if a correct CSV file with all user data for the correct time frame is generated.

- archiveData(TimeFrame)

The test is successful if all data of the correct time frame is stored in the database.

15.11.4 Testing of the ProjectHandler Class includes:

- getSettings()

The test is successful if the current project settings are retrieved from the database.

- changeSettings(Settings)

The test is successful if the project settings are changed.

15.12 ClientLogic

15.12.1 Testing of the ClientLogicFacade Class includes:

- login(Account)

The test is successful if the account is logged in and a FirebaseUser is assigned.
- logout(Account)

The test is successful if the account is logged out.
- register(Account)

The test is successful if the account is registered and saved on the database.
- resetPassword(Account)

The test is successful if an email containing a reset link was sent to the account's email address.
- changePassword(Account, String)

The test is successful if the account's password was changed successfully.
- commit(Account, List¡MoodData¿)

The test is successful if the list of mood data is saved on the database and removed from local cache.
- getMoodData(Account)

The test is successful if a list of mood data is retrieved from the database.
- getCompanions(Account)

The test is successful if a list of companions for the account is retrieved from the database.
- deleteCompanion(Account, Companion)

The test is successful if the relationship between the account and the companion is marked as ended on the database.
- addCompanion(Account, Companion)

The test is successful if a companion relationship between the account and the companion is created and saved on the database and the companion received a notification.
- inviteCcompanion(Account, Companion)

The test is successful if the user is prompted to send a generated email draft in their email app.
- changeRelationship(Account, Companion)

The test is successful if the desired relationship changes are saved on the database, or the user is shown a warning if they have previously changed their relationship with this companion.
- getSettings()

The test is successful if the current project settings are retrieved from the database.

Testing of the ClientLogicFacade Class inherit the testing of the Connection Class.

15.12.2 Testing of the ClientDataFacade Class includes:

- getSettings()

The test is successful if cached project settings are retrieved from local storage.
- getMoodData()

The test is successful if cached mood data is retrieved from local storage.
- addMoodData(MoodData)

The test is successful if the mood data was added to the local cache storage.
- setMoodData(List¡MoodData¿)

The test is successful if the list of mood data was saved on the local storage.
- getUser()

The test is successful if the currently logged in user is retrieved from local storage, if given.
- setUser(Account)

The test is successful if the account is saved as currently logged in account on the local storage.

Testing of the ClientDataFacade Class inherits testing of the ClientDatabase Class, which is responsible for reading and storing this information on the phone's local storage.

15.12.3 ClientGuiFacade

- updateHomePage()

The test is successful if the home page content reflects the up-to-date data of the logged in user.
- updateCompanionsList()

The test is successful if the companion list displays the list of the current companions of the logged in user.
- updateVisualization()

The test is successful if the visualization displays data corresponding to the logged in user and their visualization preferences.
- show*Page(); * stands for Login, Home, Record, Help, Visualization, Account, Companions

The tests are successful if the corresponding page is shown on the device.

15.13 WebLogic

15.13.1 Testing of the WebConnection Class includes:

1. Operations on the admin account
   - login(Admin)
     The test is successful, if, after providing valid credentials, the admin is logged in with Firebase.
   - logout(Admin)
     The test is successful if the admin is logged in with Firebase afterwards.
   - changePassword(Admin, String)
     The test is successful, if the admins password is changed to the specified value in Firebase.
2. Interaction with the project settings
   - getSettings()
     The test is successful if the settings are correctly loaded from the realtime database.
   - setSetting(Settings)
     The test is successful if the settings are correctly written to the realtime database.
3. Interaction with the user data in the DB
   - getCSV()
     The test is successful if the function returns a String in CSV format containing all of the required information.
   - getCSV(TimeFrame)
     The test is successful if the function returns a String in CSV format containing all of the required information, but only in the specified timeframe.
   - archive(TimeFrame)
     The test is successful if all accounts in the specified timeframe are flagged as archived in the database.

Part VII

Test Protocols

Date: 13.08.2020
Phone: Pixel3
Android Version: API 29

Client acceptance tests

Testsuite: Android application user registration and login:

Test Case C1-1 Create a new user
The software passed the test (after 10 seconds, but user can't select profile pic)

Test Case C1-2 Log in to account
The software passed the test

Test Case C1-3 Change password
The software passed the test

Test Case C1-4 Reset password
The software passed the test

Test Case C1-5 Log out
The software passed the test

Test Case C1-6 Delete user
The software passed the test

Testsuite: Android application mood recording:

Test Case C2-1 Mood recording
The software hasn't fully passed the test.
The mood of the user is recorded but not locally stored on the android device, it gets stored in the database.

Test Case C2-2 Notification prompt
The software passed the test

Testsuite: Android application companion/friend management:

Test Case C3-1 Adding companion
The software passed the test

Test Case C3-2 Inviting companion
The software passed the test

Test Case C3-3 Changing relationship status to companion
The software didn't pass the test
(feature not implemented, but there is the possibility of changing a relationship with a companion by deleting and readding with new relation)

Test Case C3-4 Remove companion from companions list
The software passed the test

Testsuite: Android application data visualization:

Test Case C4-1 Visualize mood data
The software passed the test for the user visualization. So the user can visualize his mood data with time frame and filters, but he can not visualize another companion or up to 5 other companions.

Testsuite: Android application help page

Test Case C5-1 Show help page
The software passed the test

Testsuite: Web interface:

Test Case C6-1 Log in to Web interface
The software passed the test

Test Case C6-2 Change admin-password
The software passed the test

Test Case C6-3 Change project settings
The software passed the test, except of the change of the experimental cycle. Cycle can be changed but is not saved/ has no function

Test Case C6-4 Delete user
The software passed the test

Test Case C6-5 Download data
The software passed the test (after some seconds)

Test Case C6-6 Archive data
The software passed the test

Test Case C6-7 Visualize data
The software passed the test

System tests

Test suite: Registration tests:

Test Case TG1-1-1 Registration with valid account information
The software passed the test

Test Case TG1-1-2 Registration with invalid email
The software passed the test (different error message: "Authentication failed")

Test suite: Log in tests:

Test Case TG1-2-1 Log in with valid account credentials
The software passed the test

Test Case TG1-2-2 Log in with invalid account credentials
The software passed the test (different error message: "Authentication failed")

Test Case TG1-2-3 Reset password with valid email address
The software passed the test

Test Case TG1-2-4 Reset password with invalid email address
The software passed the test

Test suite: Log out tests:

Test Case TG1-3-1 Log out
The software passed the test

Test suite: Password changing tests:

Test Case TG1-4-1 Change password with correct old and new password
The software passed the test

Test Case TG1-4-2 Change password with wrong old and new password
The software passed the test and displays the error: "Current password is incorrect"

Test suite: Record mood tests:

Test Case TG2-1-1 Record mood with special situations (voluntary)
The software passed the test

Test Case TG2-1-2 Record mood without special situations (only by notification)
The software passed the test

Test Case TG2-1-3 Automatic data commit attempt after last notification
The software passed the test (because data is automatically saved on server after every recording, when internet connection is available)

Test suite: Notification tests:

Test Case TG2-2-1 Accept notification from the first notification prompt of the day
The software passed the test

Test Case TG2-2-2 Accept notification from not the first notification prompt of the day
The software passed the test

Test Case TG2-2-3 Miss notification
The software passed the test

Test Case TG2-2-4 Miss all notification prompts of the day
The software didn't pass the test
Reason: feature not implemented

Test suite: Adding and inviting companion tests:

Test Case TG3-1-1 Add companion with valid unique code
The software passed the test

Test Case TG3-1-2 Add companion with invalid unique code
The software passed the test

Test Case TG3-1-3 Invite by email
The software passed the test

Test Case TG3-1-4 Accept invite, when user has the app installed
The software passed the test

Test Case TG3-1-5 Accept invite, when user doesn't have the app installed
The software didnt't pass the test
Reason: Not implemented because App is not added to Google Play Store yet

Test suite: Changing and removing companion tests:

Test Case TG3-2-1 Change relation and changes get saved on the server
The software didnt't pass the test
Reason: Not implemented, changes to relationship cannot be made directly (except for deleting companion and readding with new relation)

Test Case TG3-2-2 Change relation and get a error message "Relation with this user cannot be changed again"
The software didnt't pass the test
Reason: Not implemented, changes to relationship cannot be made (except for deleting and readding companion with new relation)

Test Case TG3-2-3 Remove companion
The software passed the test

Test suite: Visualize own data tests:

Test Case TG4-1-1 Change time window
The software passed the test

Test Case TG4-1-2 Change visualized data
The software passed the test

Test suite: Visualize companion data tests:

Test Case TG4-2-1 Visualize companion data and Visualization graph is updated
The system didn't pass the test
Reason: experiment cycle is not considered in visualization

Test Case TG4-2-2 Visualize companion and receive a error message "Time window covers current experiment cycle with this companion"
The system didn't pass the test
Reason: experiment cycle is not considered in visualization

Test Case TG4-2-3 Change time window and Visualization graph is updated
The system didn't pass the test
Reason: experiment cycle is not considered in visualization

Test Case TG4-2-4 Change time window and receive a error message "Time window covers current experiment cycle with selected companions"
The system didn't pass the test
Reason: experiment cycle is not considered in visualization

Test suite: Help page:

Test Case TG4-3-1 Open help page
The software passed the test

Test suite: Admin log in tests:

Test Case TG5-1-1 Admin log in with valid admin account credentials
The software passed the test

Test Case TG5-1-2 Admin log in with invalid admin account credentials
The software passed the test (different error message: "The password is invalid or the user does not have a password."

Test suite: Change admin password tests:

Test Case TG5-2-1 Change admin password with correct old and new password
The software passed the test

Test Case TG5-2-2 Change admin password with wrong old and new password
The software passed the test (different error message: "Reauthentication failed. Make sure that the password you entered is correct."

Test suite: Project settings tests:

Test Case TG5-3-1 Change consent text
The software passed the test

Test Case TG5-3-2 Change notification frequency
The software passed the test

Test Case TG5-3-3 Change notification synchrony

The software didnt't pass the test
Reason: Synchrony button exists, but has no function

Test Case TG5-3-4 Change experiment cycle
The software didn't pass the test
Reason: Cycle can be changed but is not saved/ has no function

Test suite: Manage user accounts tests:

Test Case TG5-4-1 Admin deletes user account
The software passed the test (email not necessary as input)

Test suite: Download and archive data tests:

Test Case TG5-5-1 Download all data
The software passed the test

Test Case TG5-5-2 Download data by time window
The software passed the test

Test Case TG5-5-3 Archive data by time window
The software passed the test

Test suite: Web data visualization tests:

Test Case TG5-6-1 Change time window
The software passed the test (for specific user id)

Test Case TG5-6-2 Change visualized data
The software passed the test (for specific user id)

Test Case TG5-6-3 Change user to visualize
The software passed the test (email not necessary as input)

Integration tests

The methods we coded differentiate widely from our initial drafts documented previously. All of our newly written methods have been tested positively via the interface to elaborate whether they satisfy the desired behaviour.

Part VIII

Documentation of Group Work for D1

16 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

17 Individual tasks of the team members

17.1 Johanna Bell

- Sections 3.1, 3.2, 3.3, 5.1, 5.2
- Requirements and Client acceptance tests

17.2 Luca Bosch

- Sections 1.3, 7.1-7.6, 8.1-8.5
- Introduction and Documentation of the teamwork

17.3 Niklas Maier

- Sections 1.1, 1.2, 1.3, 1.4, 6.1
- Introduction and System tests

17.4 Sebastian Schwarz

- Sections 3.4, 6
- Use cases, system models and system tests

17.5 Simon Vogelbacher

- Sections 3.1, 3.2, 3.3, 5.1, 5.2
- Requirements and Client acceptance tests

17.6 Yasmin Hoffman

- Sections 1.1, 1.2, 1.3, 1.4, 6.1
- Introduction and System tests

18 Protocols

18.1 Protocol from April 27th

Date: 27.04.20 **Time:** 16:00-17:00

- Discussed Topics
  - In which form are we going to hold regular online meetings in the future?
  - Who is going to create the protocols of these meetings?
  - What are the roles of each team member and who is going to be the project manager?
  - How do we want to call our team?
  - Who is going to schedule a meeting with our advisor?
  - Where are we going to gather questions for said meeting?
- Conclusions
  - Future meetings will be held on a Discord channel.
  - Everyone can contribute to the protocol, the responsibility for it will be taken by the Documentation Lead
  - Roles were distributed.
  - Meeting with advisor booked by the Project Manager for 13:00-13:30 on Tue. 28/04/20.
  - Qustions for the meeting will be gathered in a Discord channel.
  - We have decided on the team name 'Angry Nerds'.

18.2 Protocol from May 1st

Date: 01.05.20 **Time:** 11:00-13:00

- Discussed Topics
  - We talked about the meeting with our advisor and discussed the new information gained by getting answers to our questions
  - We discussed the design choices regarding the application layout
  - We discussed how the steps of mood selection should take place.
  - Next meeting Monday 16:00.
- Conclusions
  - We decided what to do this week
  - We gathered some more questions
  - We are going to use a bottom bar
  - We decided which elements to include in each page of the app and how the sliders, switches and other interactive buttons are going to look like. For now there will be a home screen, where you add moods, companion screen, where you add friends and relatives, settings and stats screen, where you can visualize past entries.

18.3 Protocol from May 7th

Date: 07.05.20 **Time:** 17:00-18:45

- Discussed Topics
  - We discussed our solutions to the requirement catalogue
  - Meeting with the client timeslot on friday
  - We discussed how we are going to talk to the client
- Conclusions
  - The customer relations officer will introduce our group to the client
  - We merged our solutions from the requirement catalogues, removed duplicates and refactored the text
  - We are flexible regarding the meeting on friday

18.4 Protocol from May 11th

Date: 07.05.20 **Time:** 15:00-15:20

- Discussed Topics
  - Current progress and what we are going to do next
  - We agreed that Sebastians work on the System Requirements (3.4) was to our liking
- Conclusions
  - We will meet again on Friday, 12:00 pm so we can get a look at more finished work and put it together

18.5 Protocol from May 15th

Date: 07.05.20 **Time:** 15:00-15:20

- Discussed Topics
  - We discussed work our team did this week
  - We created the work protocol for this week online
  - We discussed errors occurring in the LaTeX-Template for D1
- Conclusions
  - We will ask our advisor about how we can fix the errors
  - Next meeting 19.5. 17:00

Part IX

Documentation of Group Work for D2a

19 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

20 Individual contributions of each team member

20.1 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

20.2 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

20.3 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

20.4 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

20.5 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

20.6 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

21 Protocols

21.1 Protocol on 28. May 2020

Date: 28. May 2020 **Time:** 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

21.2 Protocol on 01. June 2020

Date: 01. June 2020 **Time:** 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

21.3 Protocol on 09. June 2020

Date: 09. June 2020 **Time:** 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

21.4 Protocol on 11. June 2020

Date: 11. June 2020 **Time:** 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

21.5 Protocol on 18. June 2020

Date: 11. June 2020 **Time:** 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.

21.6 Protocol on 22. June 2020

Date: 11. June 2020 **Time:** 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.

Part X

Documentation of Group Work for D2b

22 General organization of group

- 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

23 Individual tasks of group members

In this deliverable we had to coordinate our work a lot, since the functions and diagrams are heavily interwoven. Thus we did all of the work together, there were not really any *individual* Tasks.

23.1 Johanna Bell

- Sections Part II - 1, 2, 3, 4, 5, Part III
- Introduction, Packages, Class Interfaces, Third-Party Libraries, Glossary, Integration Tests

23.2 Luca Bosch

- Sections Part II - 1, 2, 3, 4, 5, Part III, Part IV
- Introduction, Packages, Class Interfaces, Third-Party Libraries, Glossary, Integration Tests, Documentation of Group Work

23.3 Niklas Maier

- Sections Part II - 1, 2, 3, 4, 5, Part III
- Introduction, Packages, Class Interfaces, Third-Party Libraries, Glossary, Integration Tests

23.4 Sebastian Schwarz

- Sections Part II - 1, 2, 3, 4, 5, Part III
- Introduction, Packages, Class Interfaces, Third-Party Libraries, Glossary, Integration Tests

23.5 Simon Vogelbacher

- Sections Part II - 1, 2, 3, 4, 5, Part III
- Introduction, Packages, Class Interfaces, Third-Party Libraries, Glossary, Integration Tests

23.6 Yasmin Hoffman

- Sections Part II - 1, 2, 3, 4, 5, Part III
- Introduction, Packages, Class Interfaces, Third-Party Libraries, Glossary, Integration Tests

24 Meeting protocols

24.1 Protocol on 26. June 2020

Date: 11. June 2020 **Time:** 11:00-13:30

- Discussed Topics
  - After getting feedback on the updated diagrams on D2a, we realized that we changed the wrong things and discussed how to finally get them right
  - After getting feedback on the updated diagrams on D2a, we realized that we changed the wrong things and discussed how to finally get them right
  - We brainstormed about where to start with the next deliverable, using the existing class diagrams
  - We thought about which set of deadlines we are going to stick to
- Conclusions
  - We wrote some of the pseudocode in D2b for methods that were easily understandable and clearest in their function
  - We decided to try and get back some of the time we had to invest into D2a by working even harder on D2b and finishing it quickly

24.2 Protocol on 29. June 2020

Date: 11. June 2020 **Time:** 11:00-13:30

- Discussed Topics
  - After getting feedback on the updated diagrams on D2a, we realized that we changed the wrong things and discussed how to finally get them right
  - We needed to make a new work plan and schedule our advisor meeting
  - Yesterday, we found out that we misunderstood the functionality of Firebase and hat to adjust our documents accordingly
- Conclusions
  - We scheduled an advisor meeting for Wed. 10:30
  - We tried to understand the firebase login process together
  - We created the new work plan and uploaded it to git
  - We decided to meet again after the advisor meeting to discuss how we want to proceed

24.3 Protocol on 7. July 2020

Date: 11. June 2020 **Time:** 13:00-17:30

- Discussed Topics
  - We discussed how we want to modify the class diagrams so they would be as accurate as possible compared with the final code
  - We figured out how the realtime database can be accessed via functions and how the functions there can be called dynamically
- Conclusions
  - We removed most of the authentication parts of the diagrams since those things will be handled by Firebase
  - We decided to meet again tomorrow to finish the deliverable, since we will be studying for the exams the next days

Part XI

Documentation of Group Work for D3

25 General organization of group

- 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

26 Individual tasks of group members

In this deliverable we had to coordinate our work a lot, since the coding is heavily interwoven. Thus we did most of the work together, there were very few *individual* Tasks.

26.1 Johanna Bell

- Sign-up process, notifications, mood recording, app settings
- System Tests, Client Acceptance Tests, Integration Tests

26.2 Luca Bosch

- Web interface, visualization, mood recording, cloud functions, sign up process
- System Tests, Client Acceptance Tests, Integration Tests

26.3 Niklas Maier

- Visualization, notifications, sign up process, web interface, app home screen, refining documents (D3, final document)
- System Tests, Client Acceptance Test, Integration Tests

26.4 Sebastian Schwarz

- Sign up process, companions management, mood recording, help page, web visualization, cloud functions, refining documents (D3 final document)
- System Tests, Client Acceptance Tests, Integration Tests

26.5 Simon Vogelbacher

- Mood recording, sign-up, notifications, alarm management, visualization, app settings, web interface, refining documents
- System Tests, Client Acceptance Tests, Integration Tests

26.6 Yasmin Hoffman

- Sign-Up process, notifications, mood recording, App Settings, app home Screen, refining documents (D3,final document)
- System Tests, Client Acceptance Tests, Integration Tests