Horizon Europe Programme
Research and Innovation Action

This project has received funding from European Union Horizon Europe programme, under Grant Agreement N°101080923

Start date of project: 1st May 2023

Duration: 42 months

D6.4 Awareness APP Report and Software

Deliverable details	
Work Package Title	**Digital Integrations and Data Operations**
Task Number	**T6.5**
Deliverable Number	**D6.4**
Deliverable Title	**Awareness APP Report and Software**
Revision Number	**3.0**
Responsible Organization	**RDIUP**
Author(s)	**Dah Diarra (RDIUP), Habib Nasser (RDIUP)**
Due Date	**30 June 2025**
Delivered Date	**30 June 2025**
Reviewed by	**Jevgenijs Danilins (NURO) & Zouhair Haddi (NION)**
Dissemination level	**Public**
Please cite as	**D6.4 Companion APP**
Contact person EC	**Habib Nasser**

Version	Authors	Status
1	Habib Nasser (RDIUP), Dah Diarra (RDIUP)	To be reviewed by NION and NURO
2	Zouhair Haddi (NION), Jevgenijs Danilins (NURO)	To be reviewed by RDIUP internally
3	Habib Nasser (RDIUP), Dah Diarra (RDIUP)	To be submitted

Contributing partners	
1.	FTK - FORSCHUNGSINSTITUT FUR TELEKOMMUNIKATION UND KOOPERATION E.V.
2.	RDIUP
3.	UNIVERSITATSKLINIKUM HEIDELBERG
4.	NUROGAMES GMBH
5.	ISTITUTO DI RICOVERO E CURA A CARATTERE SCIENTIFICO – AZIENDA OSPEDALIERO – UNIVERSITARIA DI BOLOGNA
6.	UNIVERZA V MARIBORU
7.	MESTNA OBCINA MARIBOR
8.	C.I.P. CITIZENS IN POWER
9.	N VISION SYSTEMS AND TECHNOLOGIES SL
10.	WIZ DEVELOPMENT & SERVICES SRL
11.	SWPS UNIWERSYTET HUMANISTYCZNOSPOLECZNY
12.	FUNDACION INTRAS
13.	UNIVERSITY OF BOLOGNA

Associated partners	
1.	HERIOT-WATT UNIVERSITY
2.	THE UNIVERSITY OF EDINBURGH

Disclaimer: SMILE is a project co-funded by the European Commission under the Horizon Europe Programme - Call: HORIZON-HLTH-2022-STAYHLTH-01-two-stage under Grant Agreement No. 101080923.

List Of Terminology	
Adolescents	We mean by that: older children, adolescents, teenagers and young adults. Three groups are considered in SMILE: early, middle, and late adolescence (10-24)
APP	APP or App refer both to mobile and web application
Digital mental health technology	Multiple technologies refer to digital technology in personalized and/or group mental health initiative
GAD7	The Generalized Anxiety Disorder 7-item scale (GAD-7) is a widely used self-administered diagnostic tool designed to screen for and assess the severity of generalized anxiety disorder (GAD)
Participants	We mean by that: older children, adolescents, teenagers and young adults participating in the piloting. Three groups are considered in SMILE: early, middle, and late adolescence (10-24)
PHQ9	The nine-item Patient Health Questionnaire (PHQ-9) is a depressive symptom scale and diagnostic tool introduced in 2001 to screen adult patients in primary care settings
Pilot administrator	Pilot staff managing the piloting and responsible of participant.
Stakeholders	Individuals, groups, or entities who have an interest, concern, or investment in a particular project, organization, or system. They can be internal or external and can have varying degrees of influence and involvement.
Well-being	Well-being, or well-being can refer to both negative and positive well-being and is what is intrinsically valuable relative to the individual.
DevOps	A set of practices, and collection of tools that aims to integrate and automate the processes between software development ("Dev") and IT operations ("Ops") teams
Feri	Refer to UoM dedicated API server interfacing FHIR and SAMF

List of abbreviations	
API	Application Programing Interface
APP	Application (Either web or mobile)
HTTP (S)	Hyper Text Transfer Protocol (Secure)
JSON	JavaScript Object Notations
FHIR	Fast Healthcare Interoperability Resources
REST	Representational State Transfer
UI	User Interface
UX	User Experience
ESM	Experience Sampling Methodology
RPC	Remote Procedural Call
CI	Continuous Integration
CD	Continuous Delivery
SAMF	Self-Assessment and Monitoring Framework
OKP	Open Knowledge Platform
DSS	Decision Support System
XP	Experience Points

Table of Contents

List of figures

Executive Summary

The SMILE Companion App, a pivotal software component of the broader SMILE project and the main output of T6.5, is designed to support the SMILE Game App and advance mental health research. This initiative is funded by the European Union's Horizon Europe Program under Grant Agreement No. 101080923. Companion App's primary goal is to increase participants' self-awareness and engagement within the study. This report outlines the app's core purpose, key functionalities, UX/UI design, and underlying technical architecture. The app serves the following main functions:

- **Data Collection:** Periodically collects comprehensive mental health data from participants using standardized questionnaires, Experience Sampling Methodology (ESM), and diary video recordings.

- **Peer Support:** While initially intended to offer peer support through digital room discussions, this feature was ultimately not developed for the final version of the app due to clinical decisions, as it was deemed potentially interfering with result analysis during the study.

- **Engagement and Feedback:** To ensure participant engagement with activity items, in-app push notifications are sent as reminders, both before an activity's scheduled start time and again prior to its expiration. The app also provides visual feedback on completed activities (both within the app and the game), empowering users to analyze their thoughts throughout the study.

For seamless and secure operation, the Companion App integrates directly with the SMILE Game App through a purpose-built SMILE API, ensuring efficient and secure data exchange between the Open Knowledge Platform (OKP) components. From a technical standpoint, the solution is built with a React Native mobile application (enabling broad deployment across iOS and Android platforms via AppStore and Play Store, respectively) and a powerful Python backend API. This backend, hosted on a private cloud server in a Paris Datacenter, not only serves as the robust data backbone for the mobile app but also acts as the central hub for secure integration with other SMILE OKP components. Crucially, both the mobile app and backend leverage Keycloak for authentication and user management, unifying user management across all OKP components. A comprehensive explanation of the SMILE OKP components and their interaction can be found in D5.1.

The app's UX/UI design evolved through multiple stages using a co-creation methodology, involving living lab and public events with relevant stakeholders. This iterative process led to two major design iterations, with the second being a direct result of collected feedback. This user-centric approach has been highly successful, receiving positive feedback from participants who feel a strong sense of ownership in the app's development.

For security and privacy, and aligning with the study's specific requirements, user access to the app is controlled: participants can only use the app after being onboarded by one of the seven designated pilot administrators, limiting authentication to login only (no direct sign-up).

Given its flexible architecture, the app can be adapted for use in other scientific research studies to securely collect rich data from participants, enhancing their self-awareness through curated visual feedback.

1 Introduction

The escalating prevalence of mental health conditions among young people has emerged as a substantial global public health burden in recent years, particularly exacerbated by the COVID-19 pandemic. While the issue is widespread, regional-specific estimates are most informative for developing targeted public health policies. Notably, in Europe, a high-income continent, mental disorders contribute a greater percentage to the overall health burden [1]. This is a striking paradox, given Europe's comparatively greater financial capacity and access to evidence-based interventions. A significant barrier to effective intervention lies in the isolation many young people experience when grappling with these conditions, often exacerbated by parents' difficulty in understanding and adequately supporting their children's mental health struggles. This fundamental lack of comprehension can further hinder help-seeking behaviors and access to vital care.

Addressing this critical gap, SMILE is a collaborative EU-funded project that seeks to harness digital technologies to advance mental health and foster well-being and resilience in young people. The project will undertake comprehensive study tests across seven European pilot sites. Central to this initiative is the development of an OKP that innovatively integrates gamification and artificial intelligence to deliver a suite of interconnected tools. This platform includes user-facing applications, the Game App and Companion App for participants, and a Web App for pilot administrators, alongside a Decision Support System (DSS) and a Self-Assessment Monitoring Framework (SAMF), which are vital for processing and analyzing participant data collected by the apps. A detailed overview of the OKP architecture and its components is presented in D5.1.

As part of T6.5 Development of Mobile Awareness App, this report focuses specifically on the Awareness App now renamed the “Companion App” by consortium partners as more user-friendly, detailing its pivotal role within the SMILE Ecosystem and providing a comprehensive technical implementation detail. As a mobile application, its primary function is to capture real-world mental health data from participants via standardized questionnaires, weekly diary recordings, and ESM. This rich dataset is instrumental for two key purposes: providing personalized feedback directly to users and supplying critical input to the broader SMILE system, enabling the DSS and SAMF to derive actionable insights, thereby contributing to more accurate mental health research and the provision of tailored support.

2 App Overview

In alignment with the introduction, the app's central purpose is to efficiently capture participants' mental health data and provide them with meaningful feedback. This section offers a

comprehensive overview of the app's fundamental features, requirements and its underlying technical architecture, thereby illustrating the mechanisms through which real-world mental health state data is acquired, processed, and presented to users.

2.1 Functional features

The app's core features encompass the following elements:

- **Authentication**: Participants can log in to the app via a centralized Keycloak server web application. User accounts are securely created by pilot administrators, ensuring controlled access.
- **Onboarding**: Upon their first login, users complete a guided onboarding process that introduces them to the project and sets up initial requirements, such as necessary permissions and availability preferences.
- **Data Collection**: As the most critical feature, this functionality focuses on delivering a variety of activities to users. These include standardized questionnaires (e.g., PHQ9, GAD7, etc.), five-times-a-day ESM questionnaires, and weekly diary recordings. Users respond based on their availability and specific time constraints defined by research protocol designers.
- **Feedback**: This feature is divided into two parts. The first provides users with progress data, while the second offers reflective exercises designed to help them understand their experiences while playing the game.
- **Account Management**: Users can manage their account information, including username and email, change their avatar, and adjust default settings such as enabling haptic feedback, app theme, date format, and sound effects.
- **Notifications**: The app provides features that allow users to accept in-app push notifications and define their preferred timing for receiving them.
- **Rewarding System**: To gamify the experience and encourage engagement, the app rewards users for participating in activities. These rewards include leveling up, streaks, and achievement badges.
- **Multilingual Support**: The app will support the seven languages corresponding to each of the pilot sites, ensuring accessibility and usability across diverse European regions.

2.2 Non functional features

Beyond these core functionalities, the application must meet some key requirements:

- **Public Availability**: The app will be publicly available for iOS through the App Store and for Android through the Play Store.
- **Device Support**: The app was developed to support the lower Android and iOS versions so that lower income participants will be able to participate in the study and not be an exclusion criteria. Tablet and Ipad should also be supported.
- **Availability:** the app should be able to support 3000 participants concurrently during the study.

2.3 Technical architecture

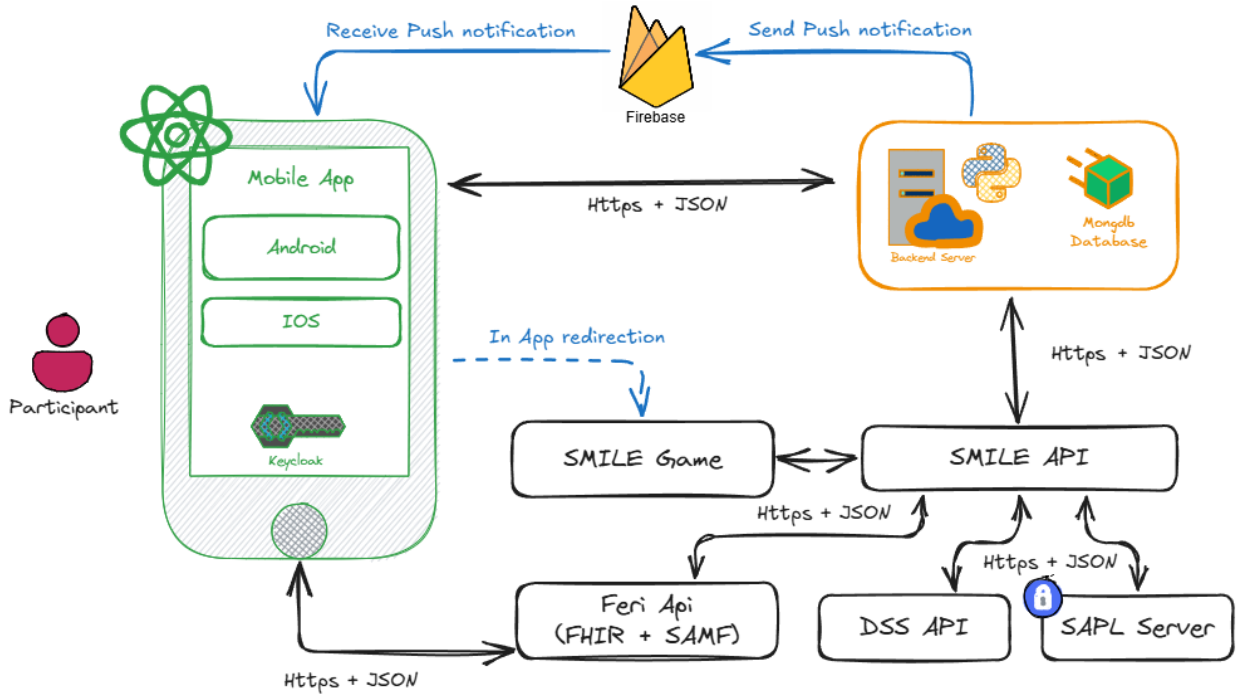

Figure 1: Companion App Architecture

The Companion App architecture, as illustrated in Figure 1, consists of two primary components: the front-end application and the back-end server. The frontend is the participant-facing interface, built with React Native to support both Android and iOS devices. The backend server acts as crucial middleware, simplifying the app's integration with other SMILE OKP components.

The backend facilitates communication by sending reminders and near real-time updates to the app via Firebase. Firebase, a free Google service, enables developers to send notifications to both Android and iOS applications. Importantly, no sensitive participant data is sent through Firebase, ensuring privacy.

For efficient retrieval of questionnaire items, the app directly connects to the Feri API chatbot, minimizing latency. Furthermore, the app redirects (via deep link) participants to the SMILE game App when a game module needs to be completed before starting activities. The backend also utilizes a MongoDB server to store metadata not directly relevant to the core SMILE ecosystem.

In the following two sections, we will delve into the details of the backend and mobile components, respectively.

3 Backend

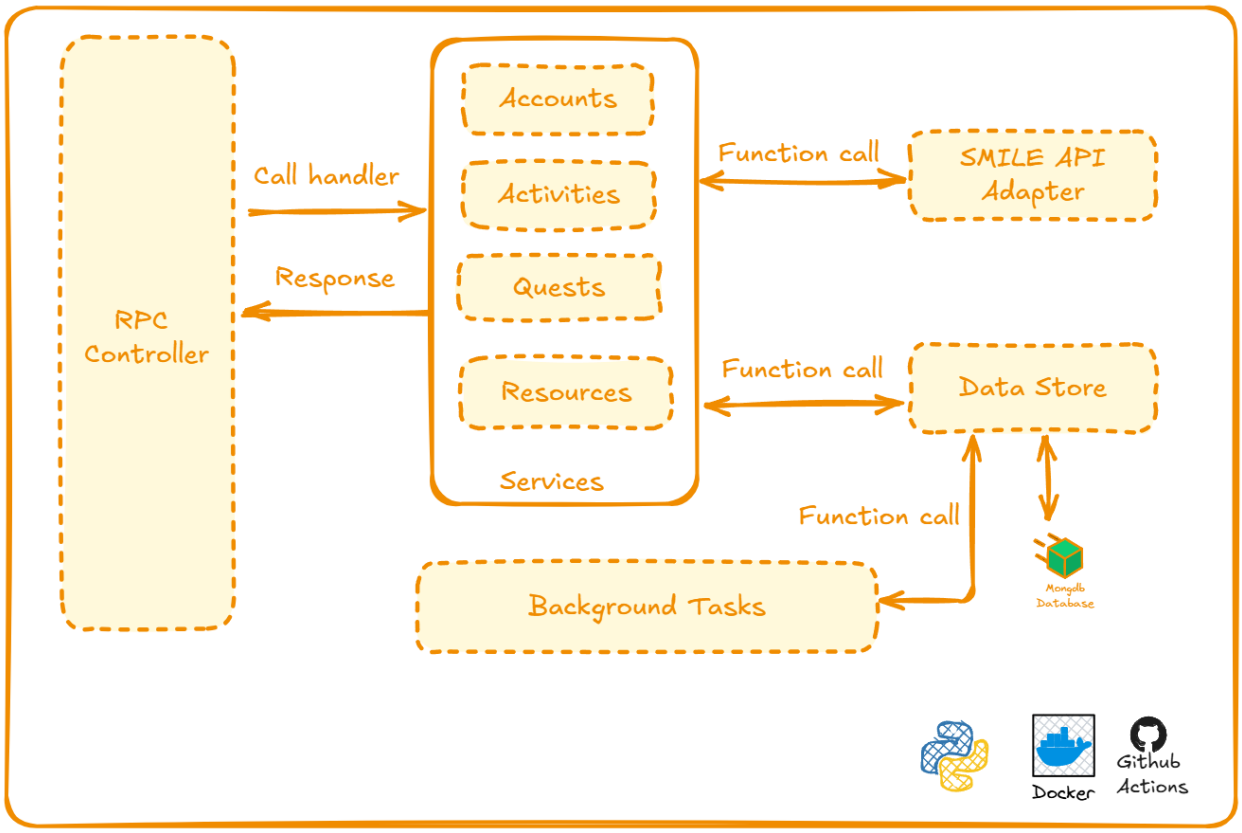

Figure 2: Companion Backend Architecture

The backend server (Figure 2) is the backbone of the Companion Application, meticulously designed to handle all resources management operations, encompassing creation, updates, and deletions. This architectural approach effectively centralizes the complex data logic on the server, thereby allowing the mobile App to maintain a streamlined focus solely on displaying information to the user and collecting their inputs.

3.1 RPC controller

The Remote Procedural Call (RPC) Controller serves as the core module responsible for the pre-processing and orchestration of all incoming client requests. Built on top of Starlette, a lightweight asynchronous Python web framework, it efficiently manages the lifecycle of each request.

Its primary responsibilities include:

- **Request Parsing and Input Extraction:** The controller meticulously parses raw incoming requests, extracting all required and optional input parameters from the JSON payload. This ensures that the subsequent processing logic receives well-formed and validated data.
- **Authentication and Authorization:** A critical function of the RPC Controller is to perform robust authentication, and authorization checks for every incoming request. This step

verifies the client's identity and ensures they have the necessary permissions to execute the requested operation, thereby enforcing security policies at the entry point.

- **Request Delegation:** After successful parsing and authentication, the controller intelligently delegates the processing of the request to the appropriate internal handler. This handler is specifically designed to fulfill the business logic associated with the requested RPC method.
- **Response Management:** The controller then asynchronously awaits the response from the designated handler. Once received, it meticulously prepares the final response for the client, ensuring it adheres to the predefined JSON format and includes all necessary data and status indicators.

For client interaction, the RPC Controller exposes a single, unified endpoint: ***https://{{hostname}}/rpc***. All communication with this endpoint is exclusively performed using the HTTP POST method, with data exchanged strictly in JSON format. This standardized and minimal interface simplifies client integration and enhances security by reducing the attack surface.

RPC controller input format.

```
1. {
2.    “method”: “string”,
3.    “version”: “string”,
4.    “trace_uid”: “string”
5.    “meta”: “json-object”, // some meta data.
6.    “params”: “json-object” // method input parameters
7. }
```

RPC controller response format.

```
1. {
2.   “succeed”: “boolean”,  // If succeed data is null and error is provided
3.   “data”: “json-object”,
4.   “error”: {
5.     “code”: “string”,
6.     “detail”: “string”,
7.     “attr”: “optional string”,
8.     “subs”: “list of error”
9.   }
10. }
```

3.2 Services

The services layer is the architectural component where all application-specific and non-generic features of the Companion App are implemented. It is meticulously designed to separate concerns, comprising two distinct types of Python functions:

- **Handlers:** These functions are responsible for directly processing client requests originating from the RPC Controller. Their design ensures that critical domain rules and business logic are consistently applied and maintained across all changes, guaranteeing data integrity and application coherence.

- **Event Reactors:** These functions are designed to react to specific events raised by the Handlers or other parts of the system. This event-driven approach fosters a highly decoupled architecture, promoting flexibility and scalability. Event Reactors can also raise new events, enabling complex, chained workflows.

This clear separation of responsibilities provides a powerful architectural advantage, enhancing maintainability, testability, and scalability. Crucially, this layer adheres to the Dependency Inversion Principle, relying on abstractions rather than concrete implementations for its external interactions. Specifically, it uses the Datastore for data persistence operations and the SMILE API Adapter for seamless communication with external SMILE OKP components. This design allows for flexible and testable interactions, facilitating mock implementations during testing and easy adaptation to evolving external services.

Furthermore, the functions within the services layer are logically grouped by resource type, optimizing for the atomicity of write operations and providing clear organization of domain logic. Below are the details of the available methods, categorized by their respective resource types.

3.2.1 Accounts Resource

The Accounts resource manages user-centric data and interactions from the client's perspective, providing functions to manage participant profiles and preferences.

- ***User.Authenticate*** **(RPC Method Handler):** Handles user authentication requests. It requires a Keycloak token received from the Keycloak server via the Companion App. This handler then leverages the SMILE API to validate the token and retrieve comprehensive user details. Upon successful validation, it returns internal access and refresh token to the client, which can be subsequently used to access other protected resources.
- ***User.Get*** **(RPC Method Handler):** Allows retrieval of a participant's profile information. These data are fetched from the SMILE API and subsequently merged with relevant local metadata stored in the Datastore.
- ***User.RefreshToken*** **(RPC Method Handler):** Used to exchange a valid refresh token for a new access token, ensuring continuous user sessions without re-authentication.
- ***User.Edit*** **(RPC Method Handler):** Facilitates the editing of participant details, such as username and avatar.
- ***User.EditDevice*** **(RPC Method Handler):** Enables updates to push notification preferences and device tokens, ensuring participants receive timely reminders and updates.
- ***User.Logout*** **(RPC Method Handler):** Manages the user logout process, invalidating current sessions and tokens.
- ***Point.Added*** **(Event Reactor):** An event raised when a participant earns points. A subscriber function reacts to this event by incrementally updating the participant's current balance based on the amount defined in the event data.
- ***Activity.Completed*** **(Event Reactor):** Triggered upon the successful completion of an activity. A dedicated function is invoked to calculate and add earned points to the participant's portfolio.

3.2.2 Activities Resource

The Activities resource manages the set of activities or questionnaire items that participants are required to complete. It provides the following functions

- ***Activity.Initiate*** **(RPC Method Handler):** Allows for the initiation of a new activity that a participant should complete.
- ***Activity.Start*** **(RPC Method Handler):** Marks an activity as "started." This method is called by the Companion App when a participant begins answering questions for a specific activity.
- ***Activity.Complete*** **(RPC Method Handler):** Executed once a participant finishes responding to all questions for an activity. It takes the user's responses as input, which are then sent to the SMILE API for forwarding and storage within FHIR.
- ***Activity.GetList*** **(RPC Method Handler):** Used by the Companion App to retrieve a list of a given participant's active activities, with support for time-based filtering.
- ***Activity.Get*** **(RPC Method Handler):** Used to retrieve detailed information for a single activity.
- ***Activity.Started*** **(Event Reactor):** Once an activity is marked as started, a subscriber function is triggered to update the activity's status in FHIR via the SMILE API.
- ***Activity.Completed*** **(Event Reactor):** Upon a participant's completion of an activity, a subscriber function is executed to check user progress and, if necessary, initiate the creation of subsequent activities.
- ***User.Onboarded*** **(Event Reactor):** After a participant completes the onboarding process, a subscriber function is used to initiate baseline activities.

3.2.3 Quests Resource

The Quests resource is designed to manage weekly and daily quests assigned to participants, serving as a motivational tool by rewarding points and badges.

- ***Quest.GetList*** **(RPC Method Handler):** Used to retrieve a list of active quests for a given participant. This method is not strictly read-only; it may also trigger the creation of new quests if none exist for the participant.
- ***Quest.Completed*** **(Event Reactor):** Executed to fulfill all business requirements upon quest completion. This includes actions such as adding points for daily quests and awarding badges and "smiles" (a motivational currency) for weekly quests.
- ***Point.Added*** **(Event Reactor):** Used to update the progress of weekly quests when points are earned by the participant.

3.3 Datastore

The data store layer is a data operation wrapper designed to simplify server data access. It provides methods to save Python objects in MongoDB, automatically handling object serialization

and deserialization. It also offers a built-in query interface that simplifies queries from client code. This data wrapper accelerates the development process, as for new resources, we only need to define a Python object that implements the main concept; this layer then handles storage and retrieval. It exposes seven main functions: loading one or multiple objects with query-based filtering, saving one or multiple objects at once, checking if an object exists based on a query filter, and deleting objects based on a query filter.

3.4 SMILE API Adapter

The SMILE API serves as a proxy, significantly simplifying inter-component communication within the SMILE OKP. This specific component, built from a collection of Python classes and protocols, provides the interface to the SMILE API. A key aspect of its design is the ability to seamlessly swap between the actual SMILE API server and the underlying component servers it encapsulates. This 'adapter' pattern enables isolated and efficient testing sessions for individual components; developers can simply plug in the relevant adapter without needing to alter core code. Ultimately, this approach accelerates the integration process, especially in cases where the SMILE API's full implementation for certain functionalities is still pending.

3.5 Background tasks

As of the writing of this document, this component comprises two primary background tasks, implemented as Python functions that execute periodically to perform specific actions:

- **Activity Reminder Task:** This function runs every three minutes to identify uncompleted activities that would benefit from a reminder. It then dispatches these reminders to participants via Firebase notifications within the mobile application.
- **Reflective Feedback Task:** This function executes every minute to check if a participant has completed a game checkpoint that necessitates reflective feedback. Upon detection, it generates new feedback prompts and sends a corresponding notification to the participant.

3.6 DevOps

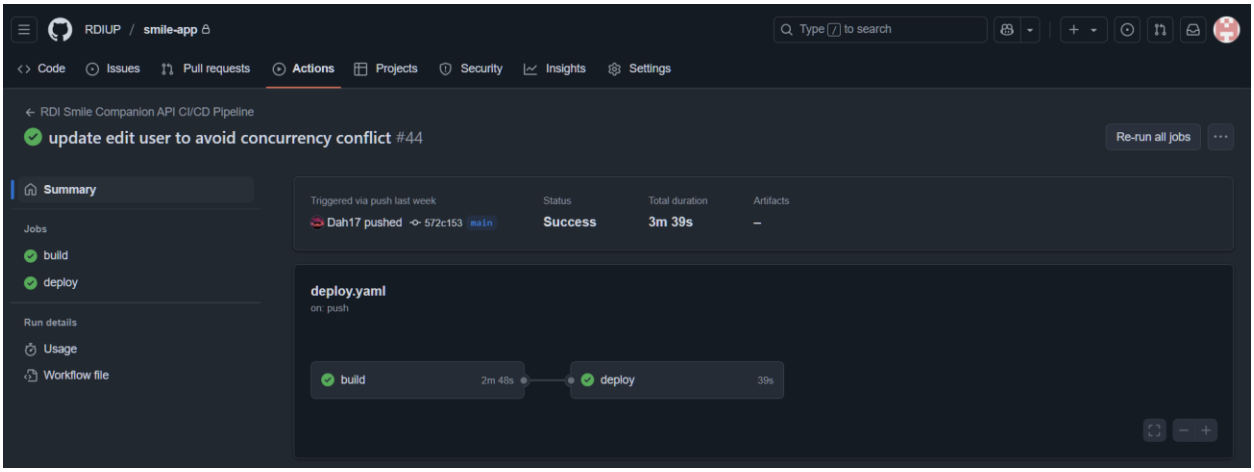

Figure 3: Backend DevOps Pipeline on GitHub

The backend API DevOps (See Figure 3) strategy is centered on automation, containerization, and secure remote deployment. We utilize GitHub Actions as our Continuous Integration/Continuous Deployment (CI/CD) platform to streamline the process from code commit to a running Docker container on the host machine. This allows us to correct issues and deliver them to the user in a matter of minutes.

3.6.1 Continuous Integration (CI)

Every code change committed to the backend repository automatically triggers a CI pipeline orchestrated by GitHub Actions, defined in *".github/workflows"* YAML files. This pipeline includes the following automated steps:

- **Code Checkout:** The latest backend code is retrieved from the repository.
- **Dependency Installation:** All necessary Node.js (or relevant language) project dependencies for the backend are installed.
- **Build:** The backend application code is compiled or prepared for execution.
- **Testing:** Comprehensive unit and integration tests are executed to validate the backend's functionality and stability. This ensures that new changes do not introduce regressions.
- **Docker Image Build:** Upon successful completion of tests, a new Docker image for the backend application is built. This image encapsulates the application code, its dependencies, and a lightweight operating system, ensuring a consistent and isolated runtime environment.
- **Docker Image Tagging:** The newly built Docker image is tagged appropriately, typically with a version number or a commit SHA for traceability and push to a private docker image hub for remote access.

If any of these steps fail, the pipeline halts, providing immediate feedback to the development team, allowing for rapid identification and resolution of issues.

3.6.2 Continuous Deployment (CD)

Upon successful completion of the CI pipeline and the Docker image build, the CD process is initiated. Our CD strategy focuses on automated deployment of the Docker image to a targeted host machine via SSH. This process is triggered by commits to designated branches (e.g., main or release branches) and includes:

- **Secure SSH Connection:** GitHub Actions establishes a secure SSH connection to the designated host machine where the backend application will run. SSH keys are securely managed within GitHub Actions secrets and are never exposed in the repository.
- **Remote Docker Operations:** Via the SSH connection, the following Docker commands are executed on the host machine:
    - **Pull Latest Image:** The latest tagged Docker image from our container registry (or if pushed, directly from the build step) is pulled to the host machine.
    - **Stop Existing Container:** Any currently running Docker container for the backend application is gracefully stopped to avoid conflicts.

  - **Remove Old Container:** The old, stopped Docker container is removed to free up resources.
  - **Start New Container:** A new Docker container is started from the freshly pulled image. This command also ensures proper port mappings and environment variable configurations are applied.

- **Post-Deployment Verification:** Automated checks are performed on the host machine to ensure the newly deployed backend Docker container is running as expected, listening on the correct ports, and responding to health checks.

3.7 Hosting & Security

As with this report's writing, Companion App remains under active development and in its internal testing phase with consortium partners. During this period, the API is deployed on an AWS cloud server in a Paris Datacenter, aligning with RDIUP's location as the app's developer. For the subsequent study phase, the server is designed for seamless migration to new infrastructure provided by the SMILE consortium, guaranteeing adherence to all data privacy requirements. Due to our automated deployment process, this transition will be a simple update of the host machine's IP address within our CI/CD pipeline configuration file. The sole prerequisite for the new server is the installation of Docker.

The backend API is engineered for security by default, strictly limiting access to only the SMILE API and the Companion mobile application. All communications are conducted over the HTTPS protocol, ensuring data encryption in transit. Furthermore, communication between the backend and the SMILE API is fortified by the Streaming Attribute Policy Language (SAPL), developed by FTK, which precisely controls access, thereby enhancing data confidentiality and integrity.

4 Frontend

The frontend of the Companion App was meticulously designed with a singular focus: to deliver exceptional user experience and intuitive interface for study participants. Unlike the backend system, the frontend's development has been a deeply collaborative process, shaped by co-creation with initial participants through a Living Lab methodology and rigorous internal testing with our consortium's clinical partners, resulting in two distinct design iteration phases, the second being the final version intended for the study.

This section will first detail the functionalities offered by the app during both phases, highlighting how feedback from the initial iteration directly informed and shaped the subsequent, refined version. Following this, we will delve into the technical implementation details of the frontend.

4.1 First Design Iteration

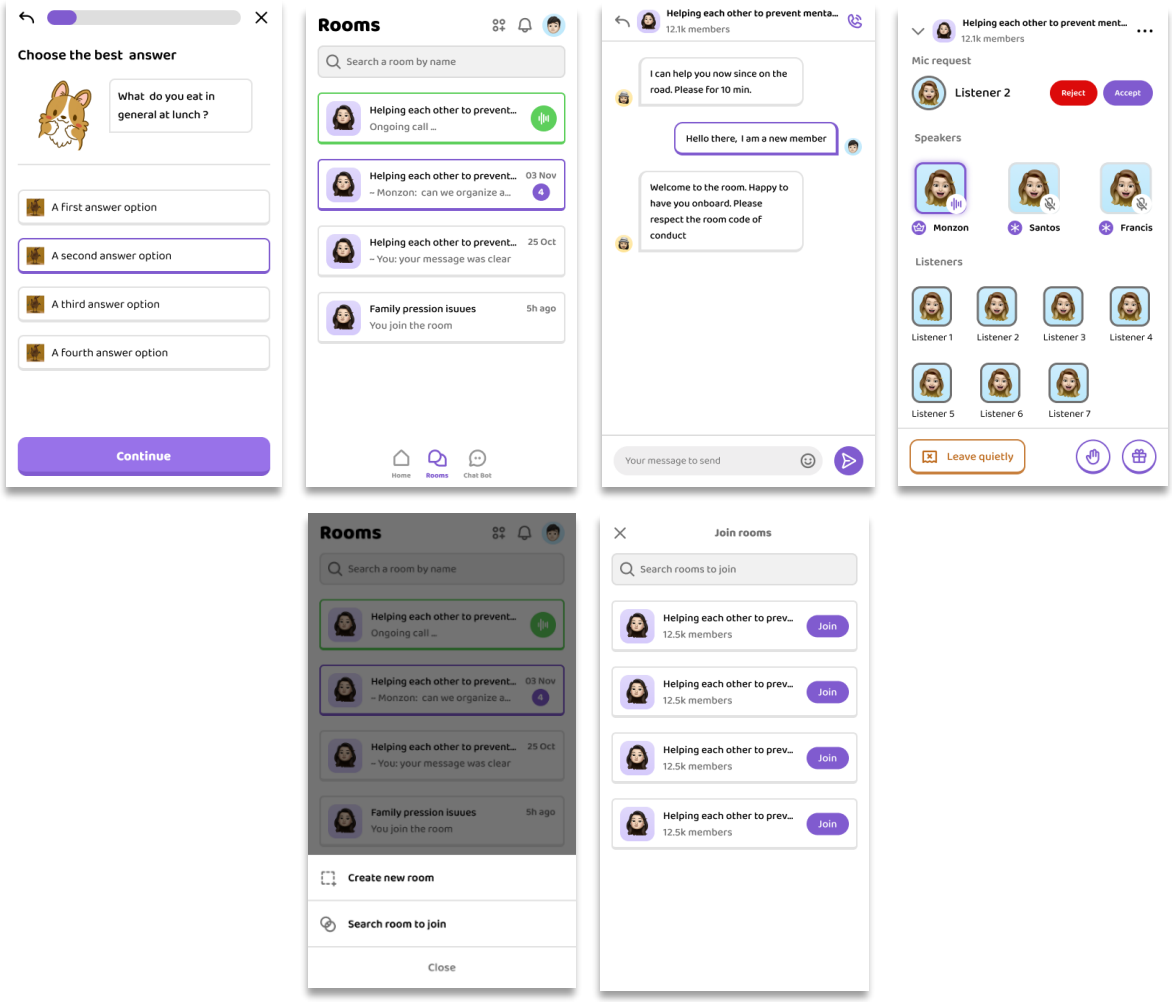

Figure 4: Initial Figma Design

Our development journey began with an initial design phase using Figma (Figure 4), a powerful prototyping tool. This early design showcased two main features: standardized questionnaire answering and a peer-to-peer review discussion room. For the discussion rooms, we visualized how users would join, engage in live voice calls, and participate in room chats.

This initial prototyping allowed us to quickly visualize concepts and gather crucial early feedback from stakeholders. These insights were immediately integrated, enabling us to rapidly develop core features of the application, therefore preparing for comprehensive Living Lab testing.

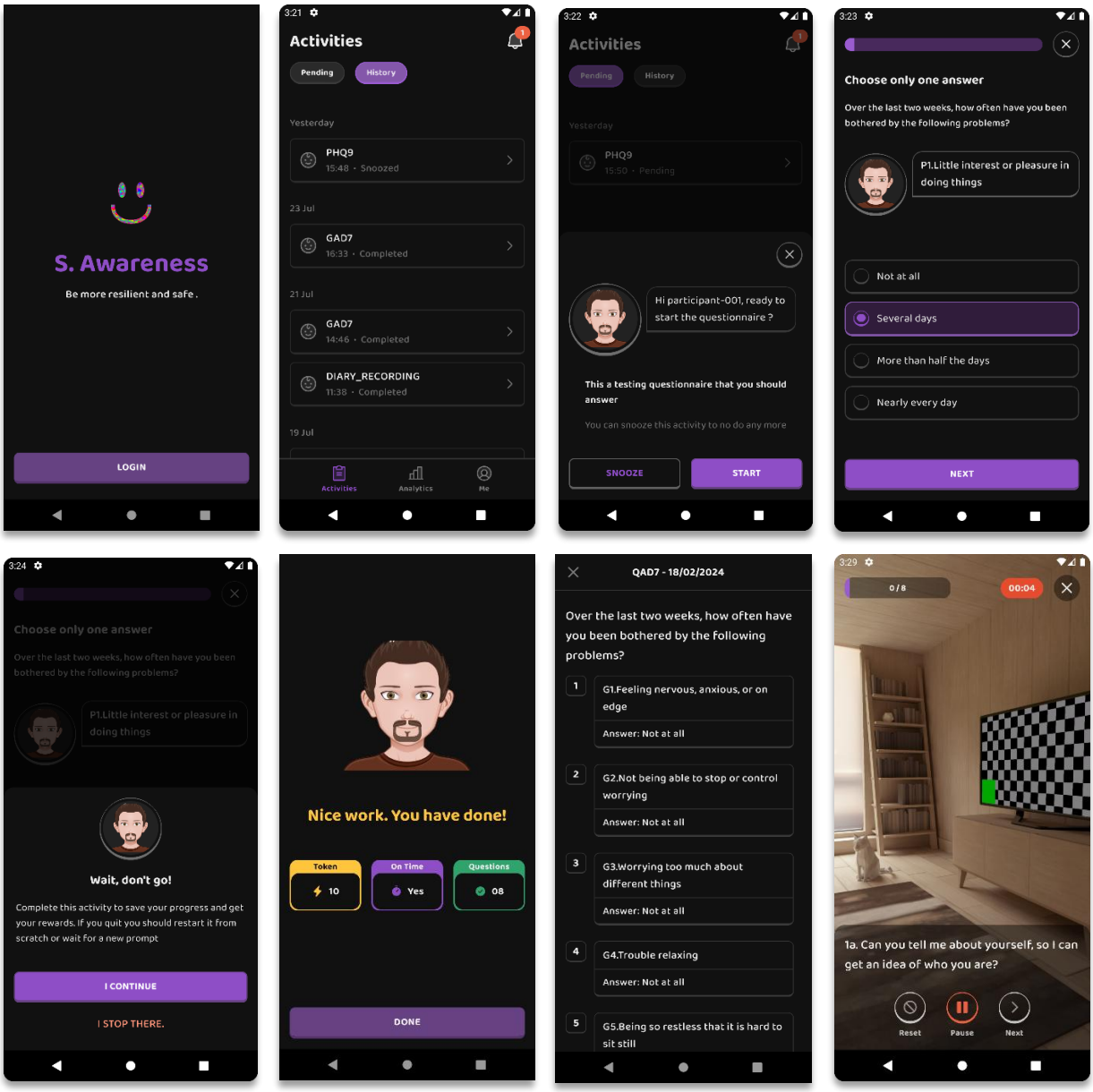

Figure 5: First Design Iteration Screenshot Part 1

As shown in Figure 5, this iteration of the application includes features that allow participants to: Log in, visualize assigned questionnaires (both pending and historical), start or snooze questionnaires, answer questions, view answering progress, view questionnaire results. When participants decide to leave the questionnaires, a modal is displayed for confirmation since the progress is not preserved.

It also provides an initial version of the diary video recording activity, complete with options to pause, resume, advance to the next question, and even reset the recording. Upon successful completion, a dedicated screen displays the Experience Points (XP) earned by the participant with a personalized completion message.

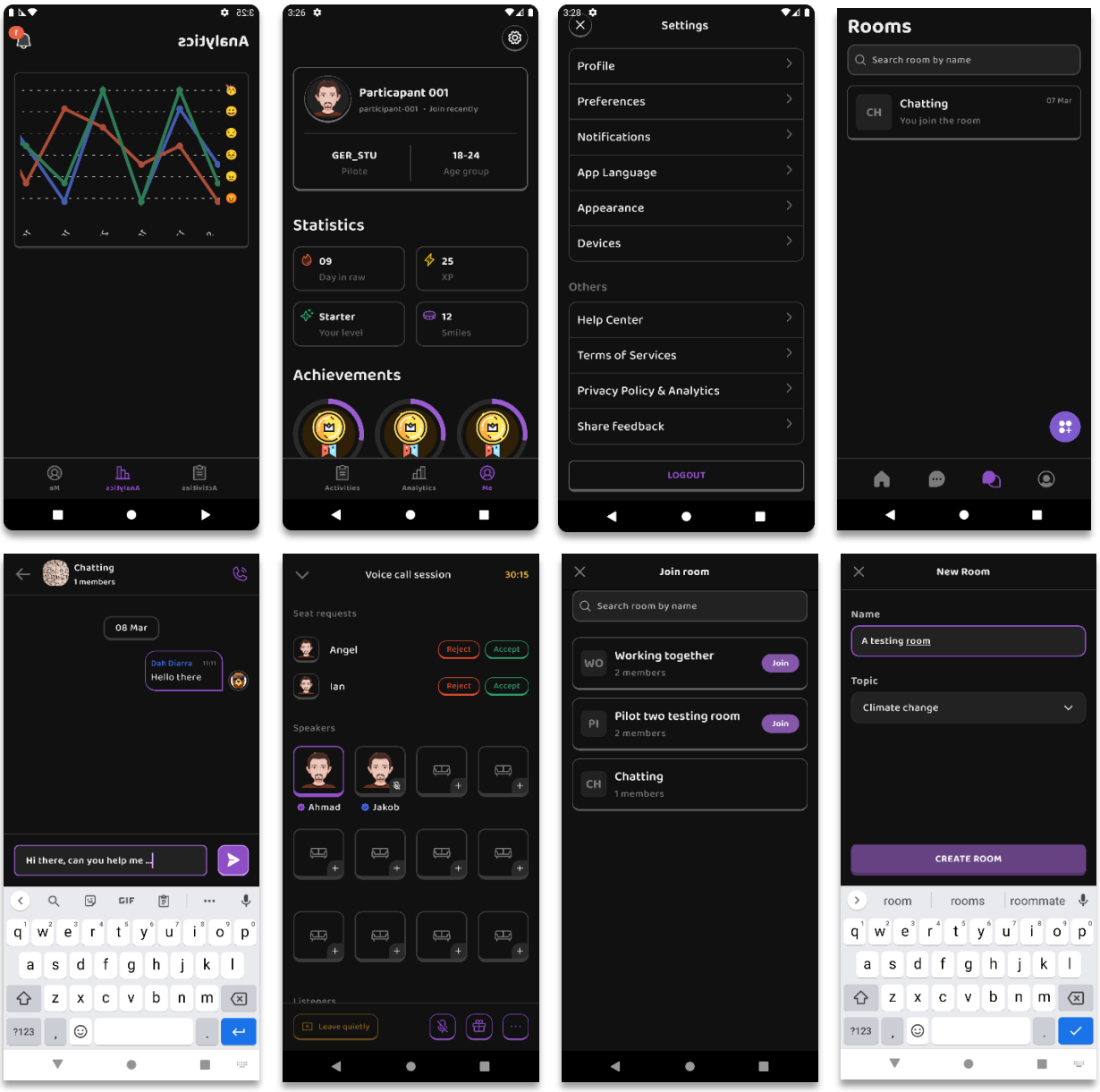

Figure 6: First Design Iteration Screenshots Part 2

These screenshots (shown in Figure 6) showcase the comprehensive functionalities available to users for managing their account and personalizing their application experience. Participants can:

- **View Basic Progress and Profile Information:** Get an overview of their journey and personal details.
- **Personalized App Appearance:** Adjust the app's visual theme (e.g., light, dark, system default).
- **Language:** Select their preferred language for the interface.
- **Notification Preferences:** Manage settings for push notifications.

- **Availability Preferences:** Define periods for activity assignments and notifications.
- **Profile Management:** Change their username and avatar.
- **Device Management:** Log out from specific devices and manage their list of active devices.

A basic feedback section was also integrated to visually represent mood fluctuations over time.

Additionally, participants can interact with discussion rooms, where they can:

- Visualize the rooms they are part of and view chat messages.
- Send text messages within the chat.
- Start or join live voice calls.
- Search for rooms to join based on their interests or thematic focus.
- Participants with appropriate permissions are also able to create new discussion rooms.

This first iteration proved invaluable, delivering a working prototype that provided the application's core features. This enabled us to conduct real-environment testing using the Living Lab methodology across all seven pilots. From this extensive testing, we gathered comprehensive feedback from each pilot, which was crucial in guiding the design for the second phase.

4.2 Second design Iteration

This second design phase focused mainly on implementing the valuable feedback gathered from the Living Lab during the first design iteration. The Living Lab phase proved invaluable; through direct engagement and observation, we collected extensive participant feedback. This allowed us to clearly identify both successful design elements and areas that required significant improvement.

This iterative process ends in a complete refactoring and redesign of the User Experience (UX) and User Interface (UI). This new design was directly driven by participant insights, ensuring the final application truly resonates with their needs and preferences. We diligently rectified elements that were not providing a smooth or great user experience, transforming them into intuitive and effective interactions.

This refined version introduces new features while also strategically removing others. For instance, the peer-to-peer discussion room feature was ultimately removed, as it was determined that it could have a negative impact on the study's results.

The following screenshots and descriptions detail the successful outcomes of this comprehensive redesign.

4.2.1 Authentication and Onboarding

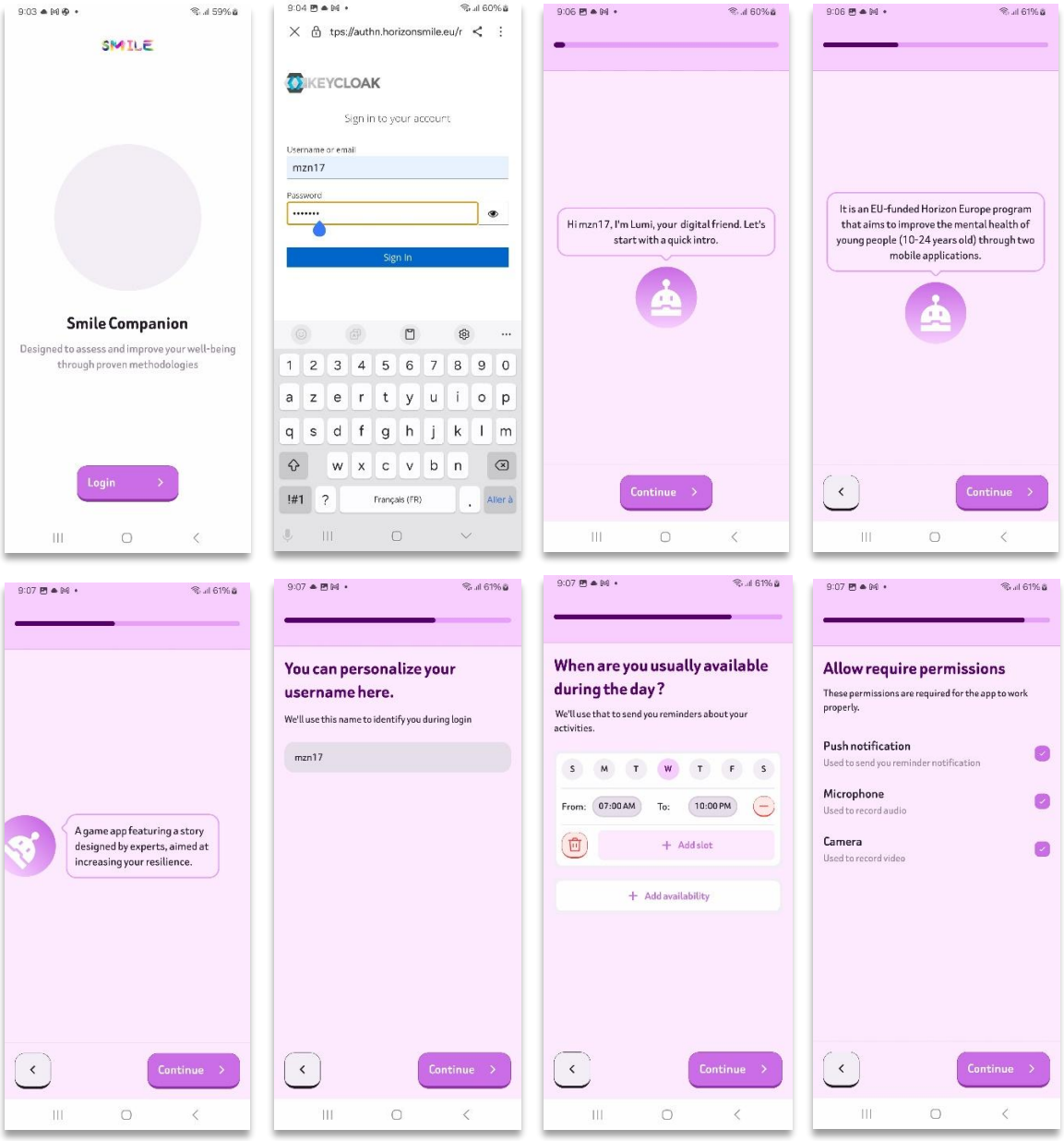

Figure 7: Authentication and Onboarding Screenshots

To begin using the Companion Application, participants must first log in. Upon initial installation or after logging out, users are prompted to perform this login. The application provides an in-app browser to load the Keycloak web client, allowing the user to securely input their credentials. Once authenticated by Keycloak and the credentials are submitted, the app retrieves the required authentication token and starts authenticating participant in Companion Backend.

If authentication fails, either with Keycloak or during backend validation, the user will receive a clear error message explaining the reason. If a user already has an active session on another

device, a pop-up will inform them that concurrent sessions are not permitted, offering a button to switch the active session to the new device if desired.

After successful authentication, if it's the user's first time using the app, they will be seamlessly redirected to the **onboarding workflow (Figure 7)**. This guided process includes:

- **Introduction Messages:** Welcoming the user and providing an overview of the app.
- **Username Personalization:** Allowing the user to define their preferred username.
- **Availability Definition:** Enabling the user to set their availability, which is crucial for personalizing the assignment periods of activities.
- **Permissions Setup:** Guiding the user through the process of setting up their required permissions.

4.2.2 Activities

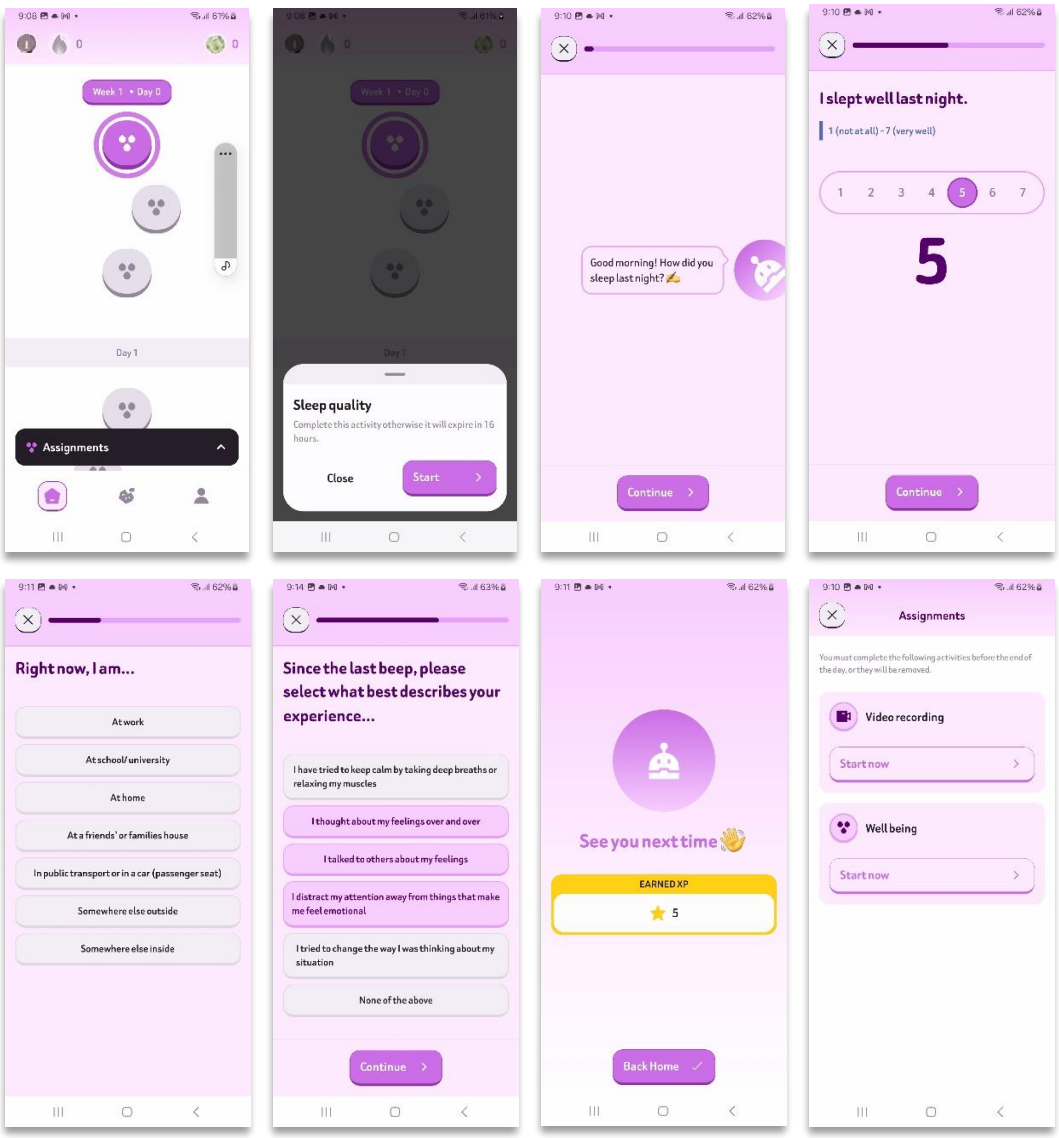

Figure 8: Activities Screenshots

As shown in the Figure 8, once a user successfully completes the authentication and onboarding phases, they're directed to the main screen. This screen primarily displays ESM activities for the current week, presented within a clear pathway. Each point on this path represents an activity, with activities thoughtfully grouped by day. Typically, there are five activities scheduled per day.

When a user taps on an activity, a pop-up appears, its content varying based on the activity's status:

- **Active Activity:** If the activity is active and ready to begin, the user sees a "Start" button, allowing them to initiate it.
- **Active, Not Yet Started:** If the activity is active but its designated start time hasn't arrived, a message indicating this appears, and the "Start" button remains disabled.

- **Completed Activity:** For activities already finished, a button is available to review the results.
- **Expired Activity:** Users will see a message indicating that the activity has expired.

When a user decides to start an activity, the activity forms screen is displayed as a full-screen modal, allowing them to answer all questions. Upon completion, a completion screen appears, showing the XP reward earned for finishing the activity. Only one activity can be active at a time. If an activity isn't completed within its defined period, it's marked as missed. For other activity types, such as standardized questionnaires, diary recordings, and reflective exercises, a dedicated button at the bottom of the main screen navigates the user to a separate screen where these activities are listed. If no such activity is scheduled for the current day, the corresponding button will simply disappear. Participants can start any available activities in any order, and activities will automatically disappear once they expire.

The top bar of the main screen features three key buttons:

- **Current Level:** Tapping this button directs the user to the level screen, showing their progress.
- **Active Streaks:** This button leads to the streaks screen, highlighting their consecutive activity completions.
- **Current Smiles Balance:** Clicking this takes the user to the Smiles Balance screen, displaying their earned motivational currency from the game.

4.2.3 Diary recording

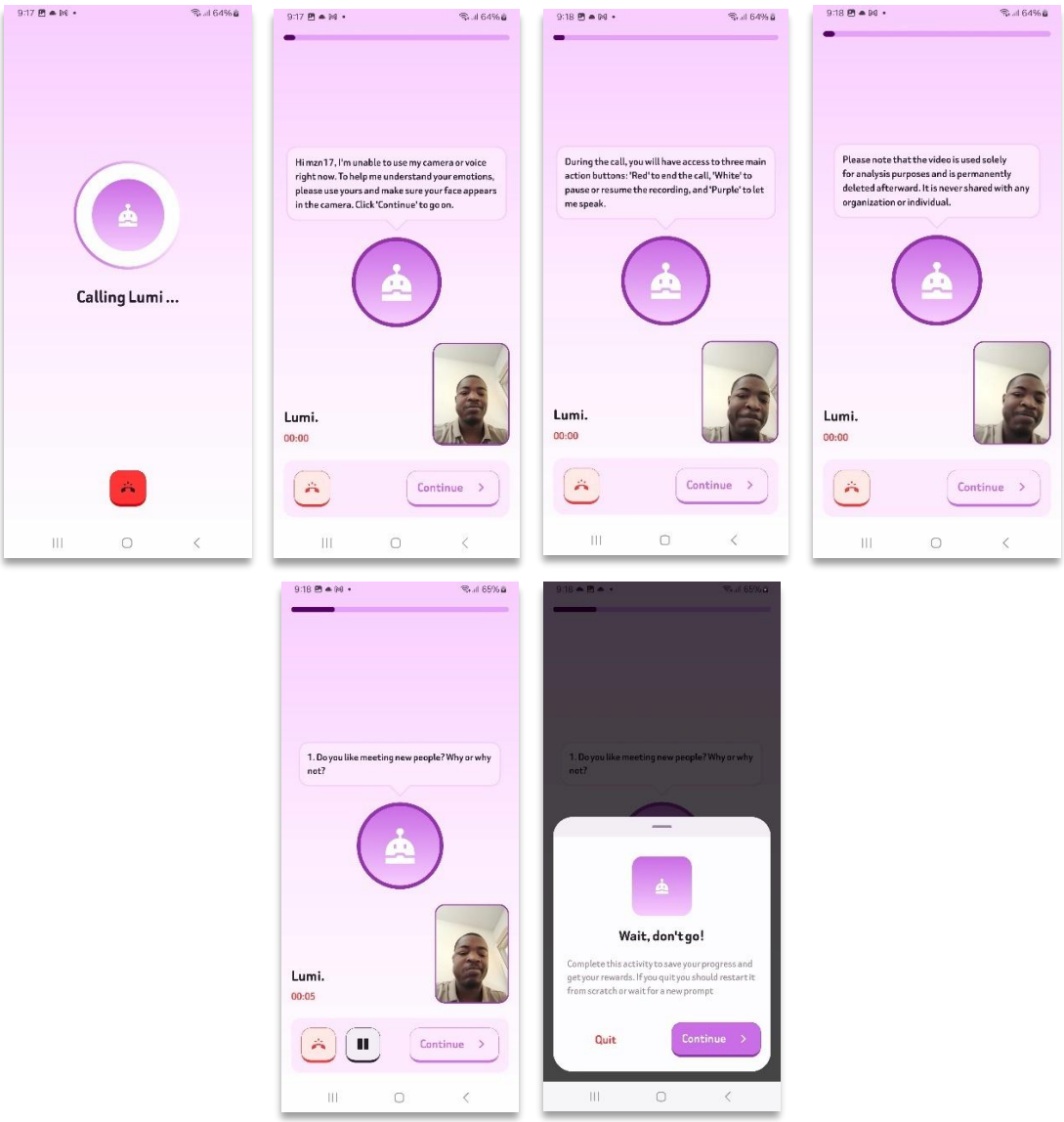

Figure 9: Video Recording Screenshots

As shown in Figure 9, since diary recording focuses on capturing the user's voice or video, its interface differs significantly from other activities. It begins with a ringing screen, simulating a call to a companion bot that the user will interact with. Next, the user is directed to the main call screen. If it's their first time using this feature, four introductory messages display to explain how the recording works and the function of each button on the screen.

The bot communicates with the participant via text, but the participant must use their direct voice and have their camera on. There are three main buttons: a red button to cancel the call at any time, a gray button to pause or resume the call, and a purple button to move forward, prompting the bot to ask the next question. Based on testing feedback, we will introduce an option to make the camera optional for participants who aren't comfortable being filmed. Upon completion, a screen appears, presenting the earned XP.

4.2.4 Feedback

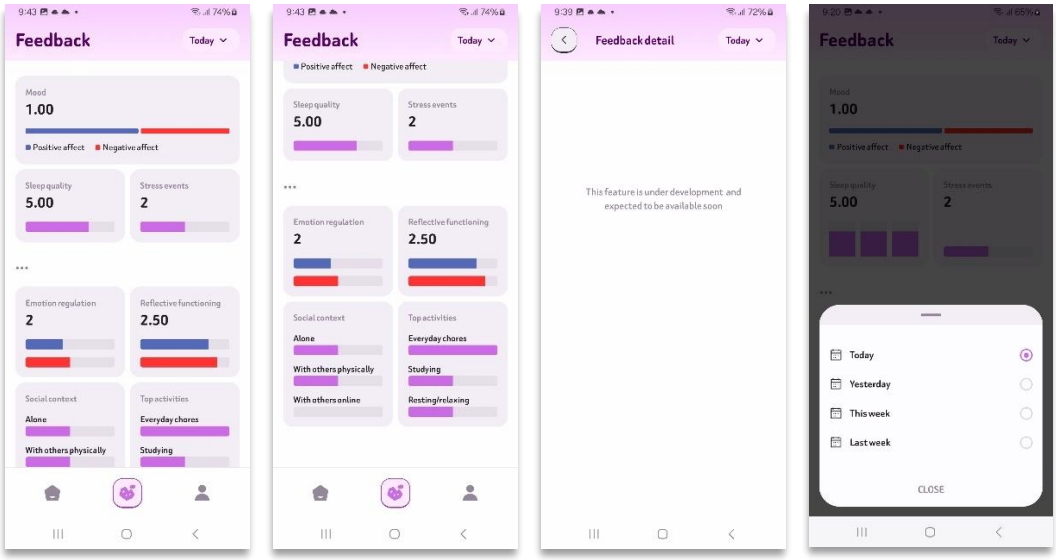

Figure 10: Feedback Screenshots

As illustrated in Figure 10, the feedback section of the Companion App provides participants with a comprehensive and insightful visualization of their progress across various areas, primarily derived from their ESM activities and game results. For other types of activities, a discussion is currently underway to determine the safest and most effective methods for displaying their results to participants.

As illustrated in the provided screenshots, the main feedback screen presents an overview of key metrics over a specified period:

- **Mood:** This metric is visualized by comparing positive affect against negative affect, with the difference between these two values representing the user's current mood. Each affect value ranges from 1 to 7. Mood is measured across all ESM activities, except for the sleep quality check-in.
- **Sleep Quality:** This metric, ranging from 1 to 7, provides insight into the user's sleep patterns.
- **Stress Event Count:** The total number of reported stress events is clearly displayed.
- **Emotion Regulation & Reflective Functioning:** These metrics are computed from specific ESM items where participant choices contribute positive or negative points. The overview box for these areas shows the net difference between positive and negative points, along with a percentage obtained, visualized as a progress bar.
- **Social Context & Top Activities:** These sections highlight the top three elements based on participant data, showing their percentage contribution over the total "beeps" (ESM prompts) within the selected period.

Game metrics are currently under development, delayed by game integration with SMILE API.

Participants have the flexibility to adjust the displayed period for their feedback, choosing between "Today," "Yesterday," "This Week," and "Last Week." We are also actively working on extending this functionality to allow participants to define customized periods by selecting specific start and end dates.

Upon tapping on any of the overview boxes, participants will be directed to a detailed screen dedicated to that specific metric. These detail screens will feature in-depth visualizations, including charts, and provide personalized messages to further explain their progress. This detailed feedback feature is also currently under development.

4.2.5 Rewarding System

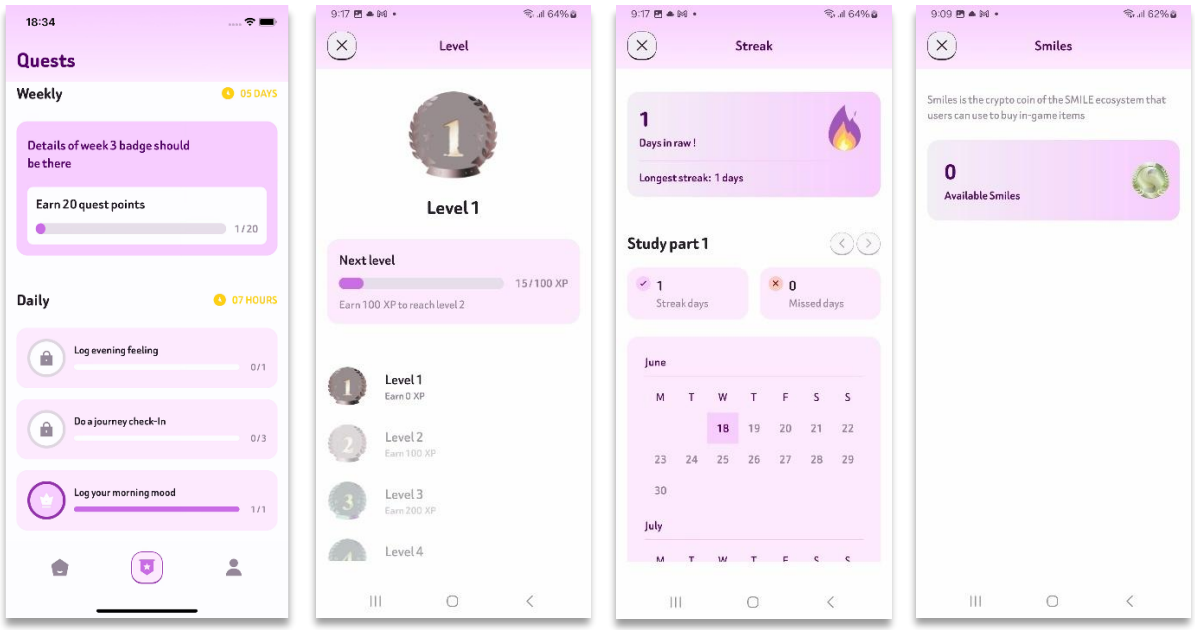

Figure 11: Quest and Rewarding Screenshots

Currently under active development and refinement, the rewarding system (Figure 11) is designed with a core objective: to significantly increase user engagement and motivation throughout the study. We employ several gamification features to achieve this:

- **Streaks:** Recognizing streaks as a highly effective engagement technique, we encourage participants to return to the app daily to maintain their continuous activity streaks. To further enhance motivation and social interaction, participants are also empowered to share their streaks with peers.
- **XP and Levels:** Participants earn Experience Points (XP) by successfully completing activities within both the SMILE game and the Companion Apps. These accumulated XP contribute to their progression through ten distinct levels. Each new level attained is rewarded with a unique badge, which participants can proudly share with others, fostering a sense of accomplishment and community.

- **Smiles (In-App Currency):** We introduce an in-app cryptocurrency, named "Smiles," which participants can earn throughout their journey. Their current balance of Smiles is prominently displayed within the app, providing a clear overview of their accumulated rewards.
- **Weekly and Daily Quests:** To drive deeper engagement, we've integrated weekly quests. Successful completion of these quests earns participants special, limited-edition badges. Progression in a weekly quest is tied to completing daily quests, which, in turn, reward "quest points." Participants must accumulate a predetermined number of quest points to successfully complete a weekly quest. Daily quests themselves are fulfilled by completing specific activities assigned for that day.

The initial implementation of this rewarding system is currently undergoing an update. Feedback received from preliminary testing indicated a need for revision to make certain mechanics clearer and more evident for participants, ensuring a seamless and intuitive experience. This iterative refinement process is crucial for maximizing the system's impact on user engagement and retention

4.2.6 Account management

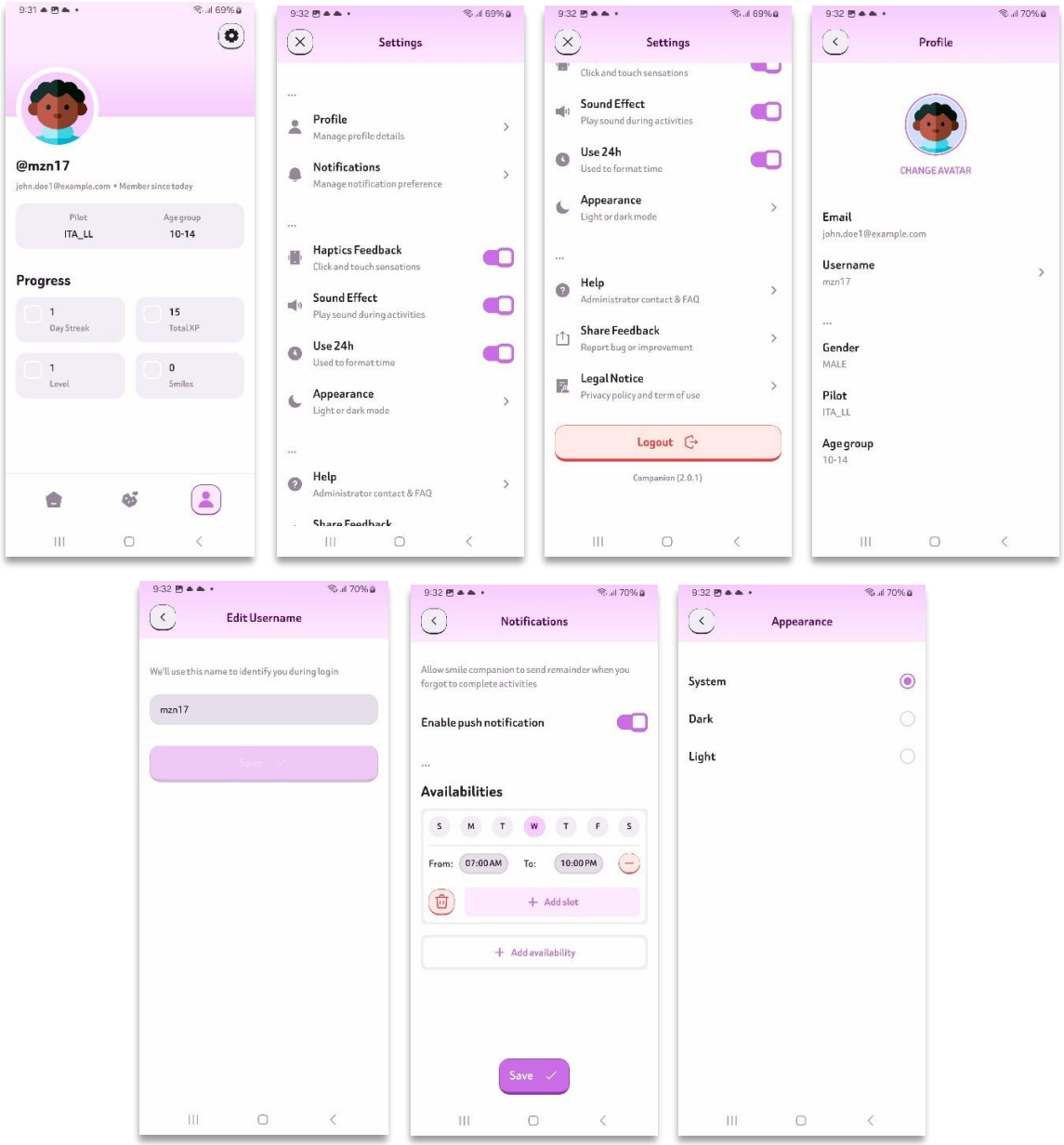

Figure 12: Account Management Screenshots

It features a comprehensive set of functionalities designed to provide users with an overview of their progress, manage personal information, and customize their application experience (Figure 12). Users can perform various actions, including:

- **Enabling Haptic Feedback:** Allowing users to feel tactile responses when interacting with UI elements.
- **Changing Date Formatting:** Customizing how dates are displayed within the application.

- **Customizing App Appearance:** Switching between system default, light, or dark themes. The system default theme aligns with the device's operating system settings.
- **Managing Notifications:** Users can enable or disable push notifications and set their preferred availability periods for receiving them.
- **Personalizing Profile:** Users have the flexibility to change their avatar and username.

It is important to note that users cannot change the application's language directly within the settings, as the language is pre-determined based on the specific pilot program the user belongs to.

The first test of this new design was exceptionally well received. Participants expressed high satisfaction, finding the updated interface not only valuable for their study participation but also deeply appreciative of the collective effort invested by both the development team and themselves in shaping a truly user-centric application. This positive feedback underscores the relevance of our co-creation strategy and its direct impact on delivering a highly engaging and effective digital companion. This co-creative approach ensures that Companion App's frontend is not just functional, but genuinely user-centric, engaging, and easy to navigate, ultimately enhancing participant retention and the quality of collected data throughout the study.

4.3 Architecture

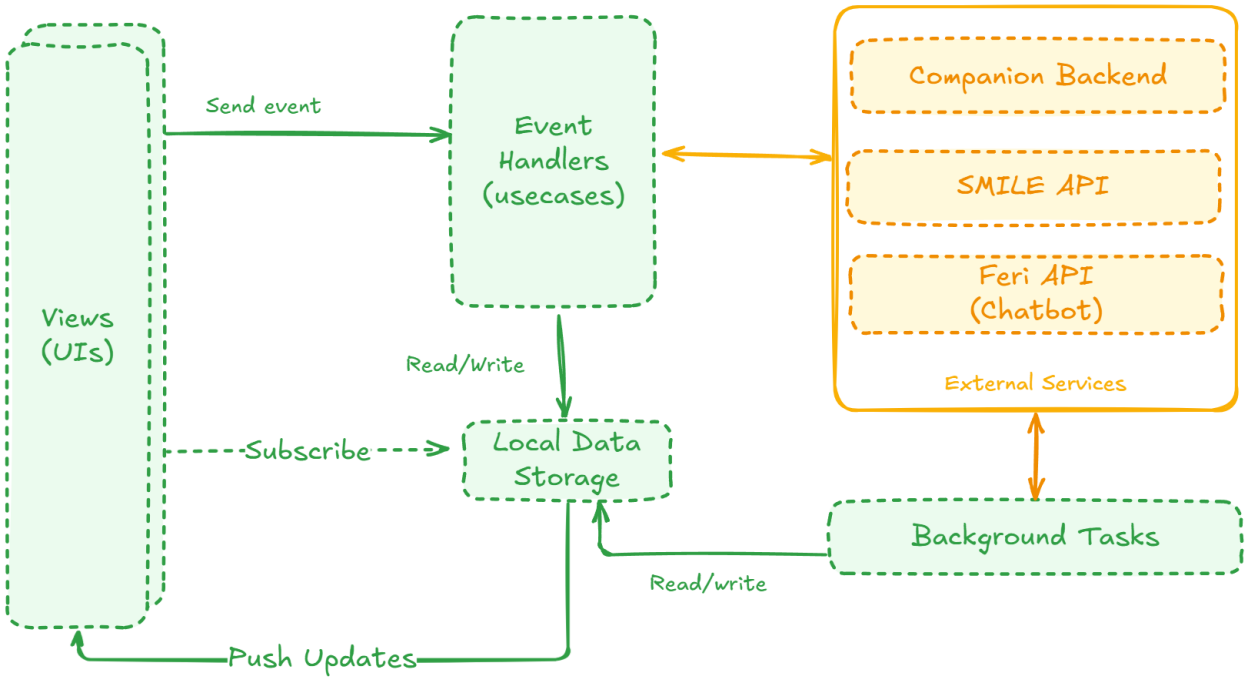

Figure 13: Companion Frontend Architecture

In terms of technical design (Figure 13), the Companion App's frontend adopts a reactive pattern. This allows for immediate visual updates across different views of the entire application as data

changes, greatly simplifying state management and ensuring a consistent data presentation to the user.

To minimize the dependency on constant internet connectivity and enhance responsiveness when interacting with external services, the app was designed with a local-first approach in mind. This means that crucial data is stored locally on the device, and UI elements (views) are engineered to listen to data changes directly from this local storage. While not all data is stored locally (to prevent excessive storage consumption), a subset of anticipated required data—such as user's personal information, active activities, and quests to complete—is persistently stored on the local data storage. Larger datasets, like real-time game measurements, are retrieved on demand from SMILE API to conserve local storage.

This local-first pattern significantly reduces the application's boot time, as there is no initial prefetching of extensive information. All data updates and synchronizations with the Companion backend are efficiently performed in the background upon app startup, ensuring that the user always has access to the most relevant information without noticeable delays.

Crucially, static resources such as translations are sync with the backend server. This strategic decision enables instant updates to these resources, eliminating the need for a full application rebuild and redeployment for changes to take effect. Given that there are seven pilot programs, each with its own language requirements, the system is susceptible to numerous potential translation issues. This backend-driven approach for static content, including activity metadata, allows for easy and rapid updates of translations and other static assets, ensuring immediate corrections and consistent localization across all pilots without disrupting the user experience with app store updates.

4.4 Development

Given the requirement for multi-device support, we strategically selected React Native as the development framework for the Companion App's frontend. This choice represented an optimal balance between performance and development efficiency, as its cross-platform nature eliminated the need to build two separate native applications for Android and iOS. This decision proved highly beneficial, as React Native fully supported all required application features. Furthermore, we leveraged its vibrant open-source ecosystem, integrating various libraries for complex functionalities such as video recording, data visualization (charts), background task execution, and Firebase for push notifications.

4.5 Deployment

As of writing this report, a fully automated CI/CD pipeline for the frontend is not yet in place, primarily due to the relatively simple and rapid build and delivery processes. During the internal testing phase, Android APKs were generated from the build output and published to a private drive, allowing testers to download and use new versions almost instantly. This approach significantly accelerated the delivery process by bypassing the typical store provider review cycles. A similar process was adopted for iOS, utilizing Apple's TestFlight, which required a review only for the initial version's validation by Apple.

Currently, a public Android version is available on the Google Play Store, though it may not reflect the absolute latest release due to the Google review process. For iOS, we are actively working through the necessary steps to comply with Apple's requirements for publishing the app on the App Store.

5 Conclusion

The SMILE Companion App stands as a vital and innovative component of the broader SMILE Project, directly addressing the critical need for advanced mental health research in young people. By leveraging a robust React Native mobile application and a powerful Python backend API, the app effectively collects rich mental health data through standardized questionnaires, Experience Sampling Methodology (ESM), and diary recordings. Its seamless integration with other OKP components via a purpose-built SMILE API ensures secure and efficient data exchange.

Crucially, the app's user-centric UX/UI design, developed through iterative co-creation with stakeholders, has resulted in a highly intuitive and well-received tool that empowers participants to analyze their thoughts. With Keycloak providing unified authentication and user management, the Companion App is not only a technologically sound solution but also a secure and private platform for groundbreaking mental health research, poised to significantly contribute to promoting well-being and resilience among young people.

While some application features are still under development and refinement, the core functionalities required for the pilot study are already fully implemented, allowing the application to be deployed for the pilot phase. Most of the pending features are expected to be released by the end of July, which is two months ahead of the first piloting sequence.

[2] A link to a web page providing a QR code for downloading and installing the app on iOS and Android devices.

6 Reference

[1] Neufeld SAS. The burden of young people's mental health conditions in Europe: No cause for complacency. Lancet Reg Health Eur. 2022 Mar 24;16:100364. doi: 10.1016/j.lanepe.2022.100364. PMID: 35345644; PMCID: PMC8956936.

[2] https://www.horizonsmile.eu/testing

7 Contact

Project Coordinator: Dr. Dominic Heutelbeck

FTK - Forschungsinstitut für Telekommunikation und Kooperation e.V.

Wandweg 3, 44149 Dortmund, 0231 - 975056-0

www.horizonsmile.eu

@HorizonSmile

www.linkedin.com/company/HorizonSmile/

@HESmile_project

Funded by the European Union under Grant Agreement No°101080923. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.