ulm university universität
uulm

Ulm University | 89069 Ulm | Germany

Faculty of Engineering, Computer Science and Psychology
Institute of Databases and Information Systems

Developing an approach to automate the building and deployment of configurable Progressive Web Applications

Master's thesis at Ulm University

Submitted by:
David Fraas
david.fraas@uni-ulm.de
1008637

Reviewer:
Prof. Dr. Manfred Reichert
Prof. Dr. Rüdiger Pryss

Supervisor:
Michael Stach

2020

Version of December 8, 2020

Satz: PDF-LaTeX 2ε

Abstract

The omnipresence of smartphones enables new methods of collecting data for research purposes on a certain research group. One possibility is the use of Ecological Momentary Assessments where a person completes assessments in his natural environment and chronologically close to the event he has to assess. This reduces the distortion of the research data compared to a retrospective assessment. Combined with Mobile Crowdsensing, where the sensors of the smartphone are used to collect additional context data, new insights on topics like chronic diseases can be gained.

However, there is no generic software solution to build and run EMA applications in combination with Mobile Crowdsensing to collect research data. In this thesis, a framework to automate the building and deployment process of configurable Progressive Web Applications (PWAs) is implemented. The thesis examines related projects to define the functional and non-functional requirements for the implementation. In the next step, a concept with technological and architectural aspects and an interface design for the web application are developed. The resulting implementation of the framework covers the processes of configuring, building and running the PWA, as well as the functionality of the PWA with notification scheduling, sensor usage and offline access. A comparison between the requirements and the actual implementation shows that the framework achieved the goal to develop an approach for building and deploying configurable PWAs.

Contents

1 Introduction

When it comes to the research of chronic diseases, the correctness of data is important to gain insight on the health status of a person. A lot of data can be collected by patients assessing on their disease. This assessment can be carried out in different ways. The point in time when the patient is assessed plays a major role.
Stone et al. [1] conducted a study to compare Ecological Momentary Assessment (EMA) with Retrospective Recall by examining the correspondence between short term (within 48 hours) retrospective coping reports and momentary reports close in time to when the stressor occurred. There was a notable difference between the two research methods. In fact, on average, "30% of the participants failed to retrospectively report using items they had endorsed on EMA assessments" [1]. The use of retrospective reports raises the questions whether patients are able to remember the symptoms of their disease and describe them accurately even after a certain period of time. Pryss et al. [2] state, that the limited validity in those reports has been shown in several studies and highlight the importance of Ecological Momentary Assessments because they can be used to prevent the problem of retrospective bias and distortion of medical data.
Mobile devices can be used to carry out these Ecological Momentary Assesssments and bring them closer to the patients. Additionally, mobile devices offer a range of built-in sensors that can be used to collect additional data. This paradigm of collecting sensor data from a group of people is called Mobile Crowdsensing (described in Section 2.2) and can be used to give a context to the patients assessment.

1.1 Problem statement

If a person who works with Ecological Momentary Assessments, for example a researcher, wants to provide an EMA application with Mobile Crowdsensing to a

group of people, for example patients with a certain disease, he will encounter the following problem: The implementation of an application for a mobile device (either native for iOS or Android, or as a web application) will require a certain level of knowledge in programming because for the development of Ecological Momentary Assessment (EMA) apps in combination with Mobile Crowdsensing (MCS) there is no generic software solution. So if the researcher wants his own custom EMA application he has to:

- learn how to program a mobile application, which will cost a lot of time to get an acceptable result, or
- pass the programming of the EMA application to one or more programmers who can implement the application for him, which will cost additional resources.

The goal of this thesis is to develop a software framework that lets, for example a researcher, generate and run his own EMA application. The framework offers the following benefits to the researcher:

- no programming knowledge required: The framework does not require any knowledge in programming.
- customization: The framework provides customization options to give the person who is working with it enough flexibility to create his custom EMA application.
- easy entry: The person who will use the framework does not have to spend a lot of time to read through instructions and installation guides before he can use it.
- automatic deployment: The framework does not only provide help for the implementation process, but also helps in the deployment process, so the EMA application is ready to use for the target group.

1.2 Subject area

The software that was developed in the course of this thesis will be referenced as “framework”. The people that use the framework are not specialists in the tech-

nical domain, like programmers, software developers or computer scientists, but rather experts in the medical domain who are familiar with Ecological Momentary Assessment and want to collect data on a certain field of research. In this thesis a person who uses the framework is referred to as an “expert“. Even though the framework does not require any programming knowledge, the expert should have basic knowledge on using a Command Line Interface (CLI) and editing JavaScript Object Notation (JSON) files.
The EMA application that the expert creates and deploys with the help of the framework will be referred to as the “PWA“. The target group of the PWA is specified by the expert, however in this thesis we refer to the person who uses the Progressive Web Application (PWA) by completing assessments as the “end user”.

1.3 Outline

The remaining seven chapters of the thesis are divided as follows: Chapter 2 describes the fundamentals of the thesis. Chapter 3 is about the related work including related projects and papers. In Chapter 4, the functional and non-functional requirements for the implementation are defined. The concept of the implementation is shown in Chapter 5. The implementation itself is described in Chapter 6. In Chapter 7, the requirements that are defined in Chapter 4 will be compared with the implementation. Chapter 8 is dedicated to the summary of the thesis and includes possible future work.

2 Fundamentals

In this chapter, we go over the fundamentals of Ecological Momentary Assessments and Mobile Crowdsensing, as well as technical fundamentals for developing a framework, like Progressive Web Applications and containerization.

2.1 Ecological Momentary Assessment

Ecological Momentary Assessment [3] describes a range of research methods that are characterized by Shiffman et al. [4] with the following features:

- Ecological: The data is collected in real world settings and environments and thus contributing to ecological validity.
- Momentary: The data is collected in current situations, in order to avoid a bias associated with retrospective assessments
- Strategic sampling: The timing for the assessments is selected by specific schemes like occuring events or by random.
- Longitudinal data: Assessment data is collected over a longer period of time in order to provide insight on how the state varies over time and across situations.

In conclusion, the main benefit of EMAs is collecting data with the absence of retrospective reports, where the subject can have a different bias because of a different situation, or a period of time lying between the report and the situation the report is about.

2.2 Mobile Crowdsensing

Mobile Crowdsensing [3] is a paradigm where a larger group of people (community) with mobile devices(e.g., smartphones) that have sensing capabilities (e.g., GPS, microphone) share and collect data. This data can then be used to gain information on common interests like healthcare. Unlike normal sensor networks, the main characteristic of MCS is the human involvement [5]. MCS can be split into two classes:

- Participatory sensing: The user actively participates in providing sensor data, for example, by choosing when, how, or what type of data is collected.
- Opportunistic sensing: The user does not have to be active, the data will be collected in the background, sometimes even without the users acknowledgment.

2.3 Progressive Web Applications

While normal web apps have a very high reach because they can be used by anyone with a device that has a modern browser installed, their functionality is limited. On the other hand, native apps provide high functionality, because they can make use of all the features that a device offers and work offline, but they are limited to the operating system that is installed on a device. PWAs [6] can be an alternative to both, because they are based on well-known web technologies like HTML, CSS and JavaScript and thus can be used with a browser, but they also provide extended functionality with APIs that provide, for example, offline accessibility, push notifications, or home-screen installation.

One main component of PWAs are service workers[1]. These service workers are scripts that are running in the background of the browser, separate from the web page. They enable features like offline functionality, background sync and push notifications. Figure 2.1 shows the life cycle of a service worker. When a service worker is registered in the websites JavaScript, the installation process will be started in the background when the web application is used for the first time. After

[1] https://developers.google.com/web/fundamentals/primers/service-workers

being installed, a service worker will get activated and fall into an idle state to listen to fetch and message events. When not in use, a service worker will terminate and restart (go into idle state) when it is next needed.

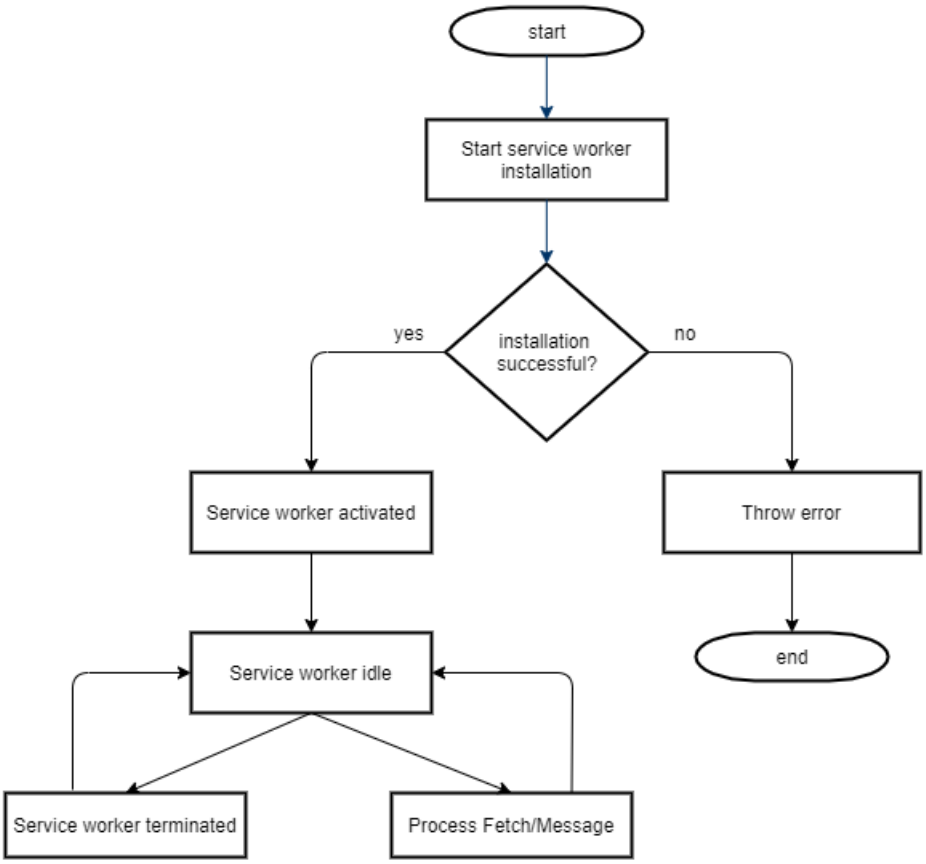

Figure 2.1: Life cycle of a service worker

2.4 Containerization

Containerization is a virtualization method of deploying and running software applications. Instead of running in a virtual machine, application containers run on a containerization engine that is installed on the host operating system. Each container includes all libraries and dependencies it needs to run an application.
Figure 2.2 (left) shows the structure of a virtual machine distribution with the infrastructure as the bottom layer and the hypervisor above. The infrastructure usually

consists of the hardware that the system is running on (e.g., a server). The hypervisor is a software that is installed directly on the infrastructure layer and controls multiple virtual machines. Each virtual machine has its own guest operating system and application(s) that run on it.
Figure 2.2 (right) shows the structure of a containerization distribution. Instead of a hypervisor, a host operating system is installed on the infrastructure. On the host operating system, a containerization engine is installed. The containerization engine controls the containers that run the applications for the system.

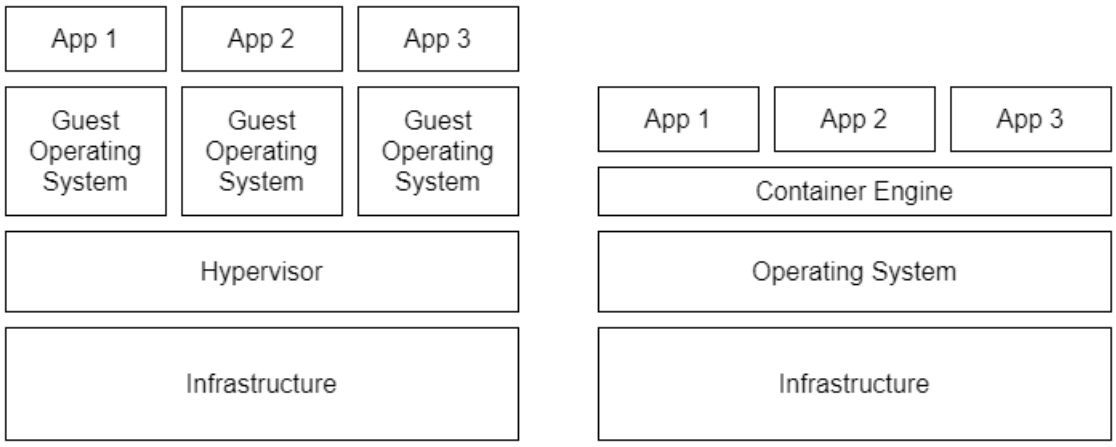

Figure 2.2: The structure of a virtual machine distribution versus the structure of a containerization distribution

2.5 Reverse proxy

A reverse proxy[2] is a (virtual) server that sits in front of one or more origin servers and redirects incoming requests to these servers. The requests come from clients (e.g., web browsers) and automatically get forwarded to the corresponding web server. The main benefits of reverse proxies are:

load balancing: If a website is deployed on a single server and millions of requests come in at once, it might not be able to handle all of the requests and fail. In order to prevent a single server from failing, the website can be split onto multiple servers

[2] https://www.cloudflare.com/learning/cdn/glossary/reverse-proxy/

and a reverse proxy will distribute incoming requests evenly across these servers to balance the workload.

protection from cyber attacks: Reverse proxies can protect servers from cyber attacks like denial-of-service attacks (DoS). Additionally, the origin server does not have to reveal its IP address which makes it harder to attack directly.

caching: Reverse proxies can cache website data to improve response time and reduce the workload of the origin server.

encryption: The task of TLS-encryption can be taken over by the reverse proxy and further reduce the workload of the origin server.

3 Related work

This chapter shows related work for applications in the EMA and MCS domain. Intersession-Online and TrackYourTinnitus both offer native applications for Android and iOS and give examples when it comes to developing EMA apps. Additionally Kraft et al. [3] wrote a paper about the combination of EMA and MCS in the health-care domain and propose recommendations as well as a reference architecture for developing mobile EMA applications. The thesis does not completely follow these recommendations but rather use them as inspiration for building a framework for EMA applications.

3.1 Intersession-Online

Intersession-Online [7] is a project by the research group for psychodynamic psychotherapy research (Arbeitsgruppe Psychodynamische Psychotherapieforschung) at the university of Klagenfurt in corporation with the Ulm University to examine the thoughts, feelings and memories that a patient has in between psychotherapy sessions (the so called “intersession processes“). The goal of the project is to collect data about intersession processes with the help of a smartphone application. The application is available as a native app for Android and iOS and additionally offers a website for registration and information purpose. Figure 3.1 shows the Android Application of the Intersession Online project.

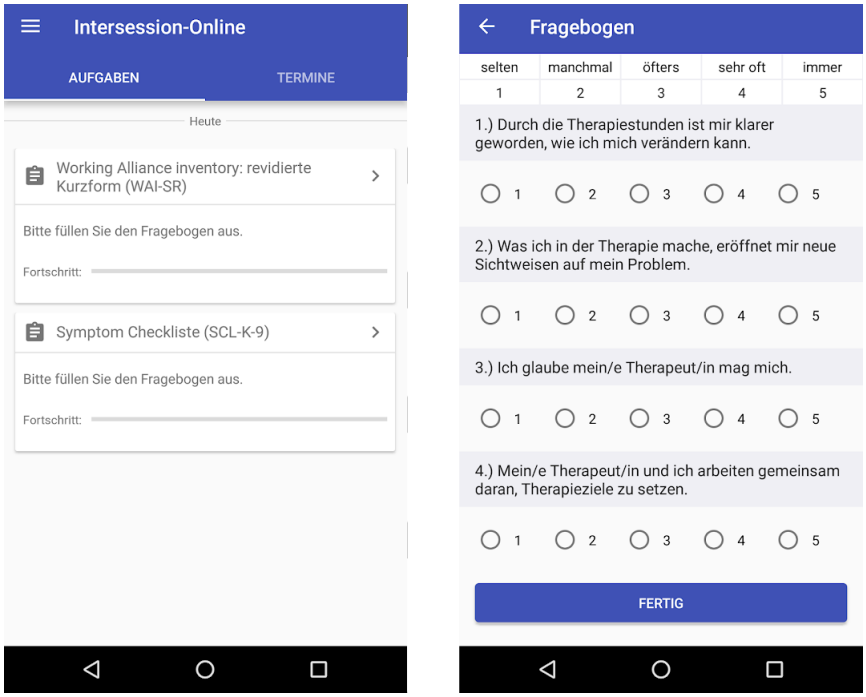

Figure 3.1: Screenshots of the Android Intersession App [8]

3.2 TrackYourTinnitus

TrackYourTinnitus (TYT) [9, 10, 11, 12] is a platform to collect data for tinnitus patients. It was developed by the Tinnitus Research Initiative (TRI) and the Institute of Database and Information Systems (DBIS) at Ulm University and is available and maintained since April 2014. TYT consists of a website for registration, a native iOS application, a native Android application and a central backend where the data is stored. The mobile applications offer the user assessments about their tinnitus in the style of an EMA. The user is prompted at random times during the day to complete his assessment. The questions of the assessment measure the tinnitus in eight different dimensions, including, for example, perception, loudness and distress. Additionally, the application measures the background noise in the users environment when the user is filling out an assignment. Figure 3.2 shows the Android Application of the TrackYourTinnitus platform.

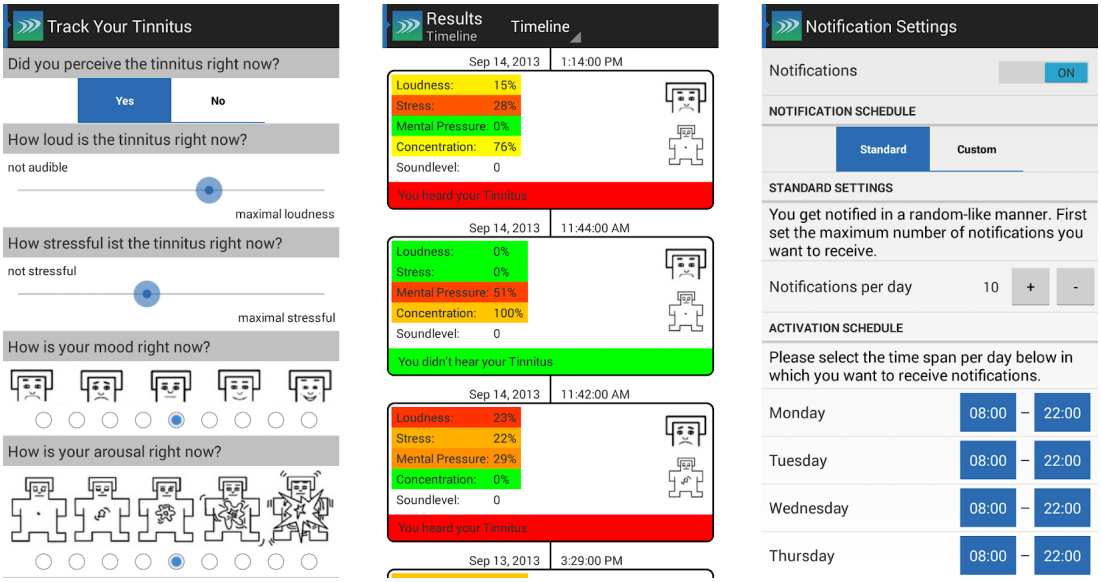

Figure 3.2: Screenshots of the Android TrackYourTinnitus App [13]

3.3 Combining MCS and EMA in the healthcare domain

Kraft et al. [3] examined the combination of Ecological Momentary Assessment and Mobile Crowdsensing in the healthcare domain. They analysed the state of the TrackYourTinnitus project, that is running for over 5 years and compared seven different EMA apps to derive a set of recommendations when building an application that combines EMA and MCS in the healthcare domain. These recommentations include:

User identity: Identify the user by using proper authentication and authorization.
Generic questionnaires: Handle generically defined questionnaires, including one-time and repeating questionnaires with various input types (e.g., multiple choice, text input).
Notifications: Prompt the user to complete assessments with a notification schedule that can be edited by the user. The notifications should be either at fixed times or created randomly by an algorithm.
Sensors and context-awareness: A set of sensor measurements should be defined, in order to record sensor data in the background.

Incentive mechanisms: Support the users adherence by including incentive mechanisms like feedback, gamification, or social features.
Groups, studies, and HCPs: The ability for the user to join one or more groups, for example for the representation of studies.
High availability and Performance: Best possible availability of the application and absence of performance issues.
Offline availability: Ability to use the application without connection to the internet.
Safety, security, and privacy: Presence of a high standard in safety, security and privacy, including consideration of region-specific regulations, encryption, identification of health risks and the existence of a security model for the platform.
Data quality: Compliance with data quality aspects including believability, relevancy, accuracy, interpretability, understandability, accessibility, objectivity, timeliness, completeness and (representational) consistency by using, for example, input validation.
Data analysis: Existence of data analysis functionality for the researcher, the healthcare provider and the user to review and analyze answers to questionnaires and sensor data.
Interoperability: Interoperability with other platforms by implementing common data exchange format standards, communication protocols and interfaces.

4 Requirements

When it comes to defining the requirements for the implementation of the framework, the following aspects should be considered:

- requirements for an EMA application
- requirements for Mobile Crowdsensing
- requirements for automation

According to Shiffman, 2007 [14] and Shiffman et al., 2008 [4] the implementation of EMA apps on mobile devices require the following functionality :

- Presentation of assessment content (questions and response option) to the user.
- Management of assessment logic (input validation and branching).
- Recording of time-stamps to determine when an assessment was completed.
- Storing assessment data.
- Management of user notifications (when should an assessment be made ?).
- Prompting the user to complete the assessment.

In order to fulfill the property of a “momentary“ assessment, the EMA app should be available whenever the user needs it. Therefore, offline support is mandatory, so that the user can complete his assessment even without a connection to the internet.
Concerning Mobile Crowdsensing, collecting sensor data requires the EMA app to have access to a various set of sensors. The availability of sensors depends on the mobile device that the user owns as well as the software (operating system, internet browser) that runs on the mobile device. EMA apps typically do not focus on a single type of Mobile Crowdsensing (mentioned in Section 2.2), but instead use

both, participatory sensing and opportunistic sensing, to some extend.
For automation, in order to keep the time and knowledge required by the expert to develop and run an EMA app at a minimum, the framework should do as much work as possible automatically. This includes automation in development (e.g., generating views and database) but also automation in deploying and running an EMA app (e.g., launching multiple components simultaneously).
Based on the requirements for EMA, MCS and automation, this chapter derives the functional and non-functional requirements for the implementation. Additionally, the recommendations from Kraft et al. [3] in Section 3.3 will be taken into consideration for the implementation, however, the implementation will focus on the collection of data only, therefore data analysis and incentive mechanisms will not be considered. The functional requirements are split between functions for the framework and functions for the PWA. A code is assigned to each requirement to refer to them. Additionally, each requirement has a description and a priority, which is assigned by using the MoSCoW method [15].

4.1 Framework functionality

The framework is used by the expert to configure, build and run the PWA. The following table shows all functions for the expert when using the framework.

Code	**Function**	**Priority**
Configuration		
F01	Configure assessment	MUST
F02	Configure database	MUST
F03	Configure registration	MUST
F04	Configure appearance	COULD
F05	Configure server	MUST
F06	Configure SMTP	SHOULD
F07	Configure sensordata	MUST
Deployment		
F08	Build PWA	MUST
F09	Run PWA	MUST

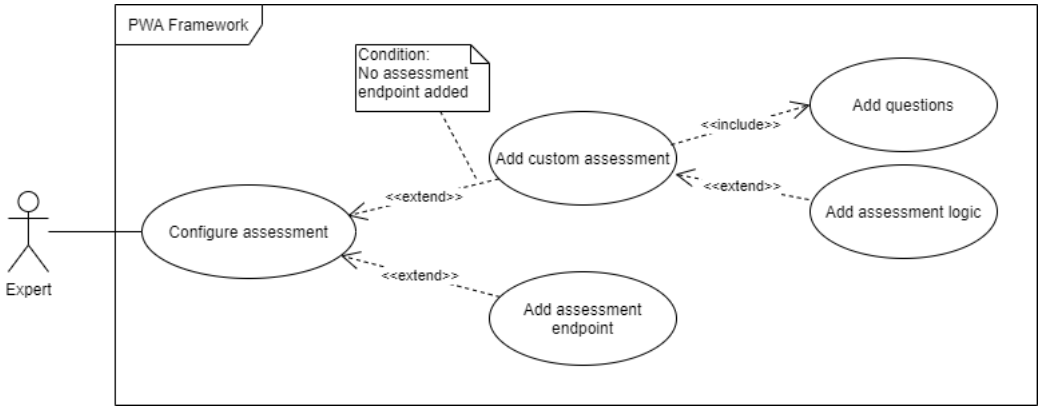

Figure 4.1: Use cases for configuring an assessment

F01 (Configure assessment): The expert can add an assessment, which contains questions for the end user to answer and an assessment logic, which controls the flow of the assessment (order of questions). The assessment can either be added directly to the configuration or an assessment endpoint can be specified, which then imports the assessment from an external source.

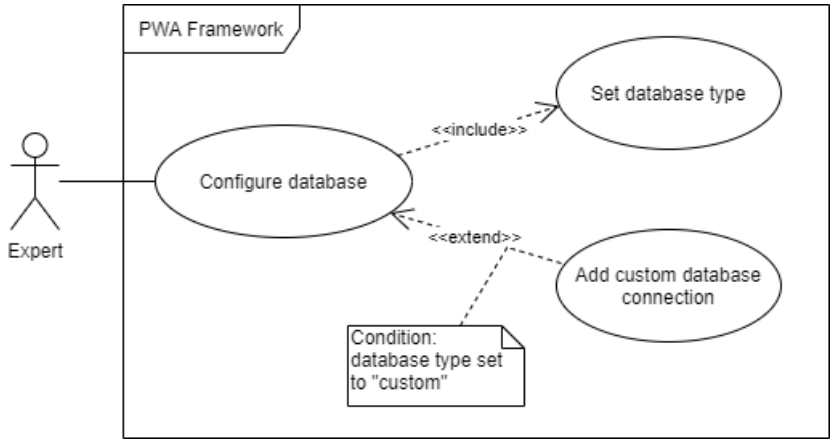

Figure 4.2: Use cases for configuring a database

F02 (Configure database): In order to store the data collected by the end user, the expert can add a database connection. The expert can either let the database be configured and created automatically, or add a custom database connection.

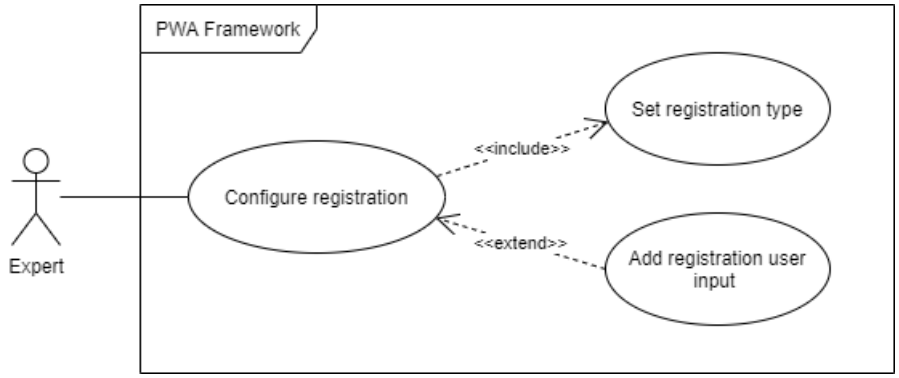

Figure 4.3: Use cases for configuring a registration

F03 (Configure registration): The expert can decide whether the end user is required to register an account or use the PWA without an account. Additionally, the expert can define one or more registration inputs (e.g., age, name) that are associated with the end user in order to get additional information on the end user.

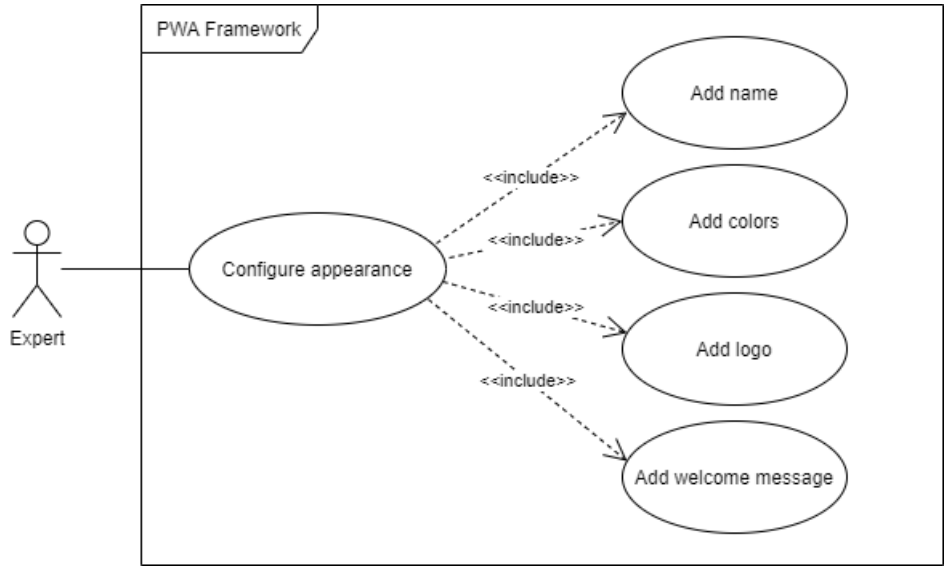

Figure 4.4: Use cases for configuring an appearance

F04 (Configure appearance): The expert can change the appearance of the PWA by adding a name, a logo and colors. Additionally he can specify a welcome message that is shown to the end user when he visits the PWA for the first time.

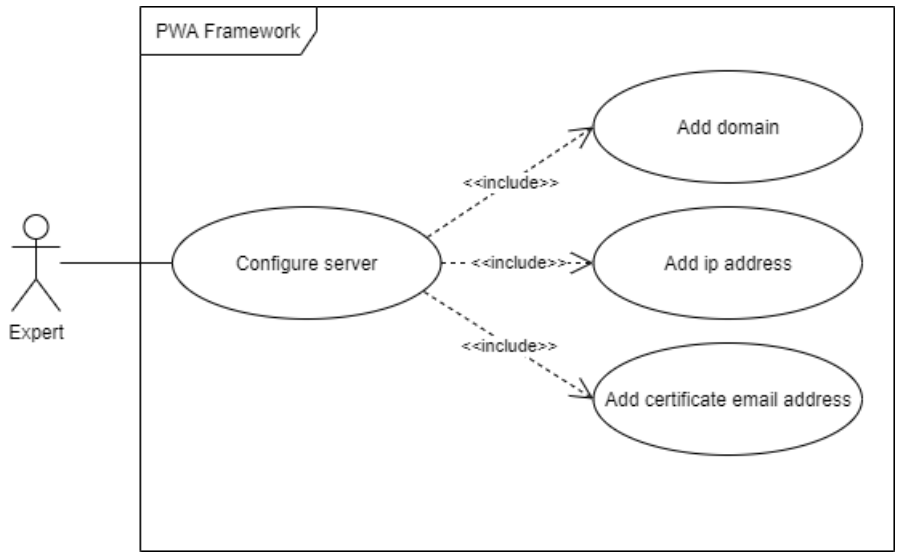

Figure 4.5: Use cases for configuring a server

F05 (Configure server): The Expert has to specify the domain and the ip address of the server where the framework is installed on. Additionally he has to provide an email address which is used to generate the certificate for a secure HTTPS connection.

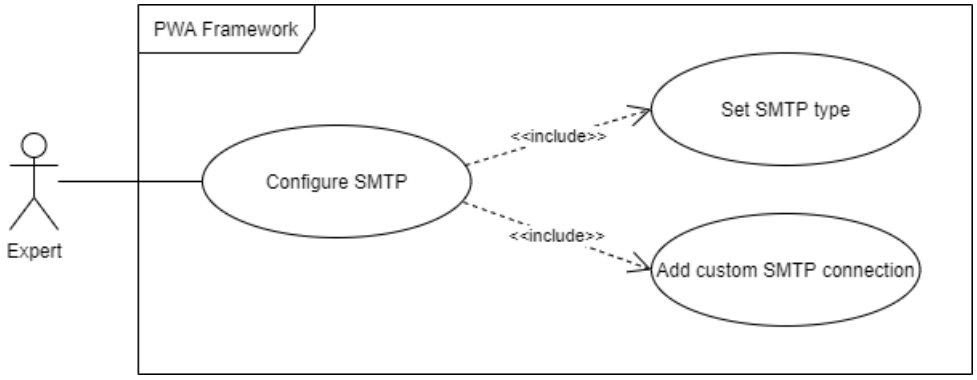

Figure 4.6: Use cases for configuring a SMTP server

F06 (Configure SMTP server): If the expert wants to use his own SMTP host for sending emails from the server, he can add a custom SMTP connection.

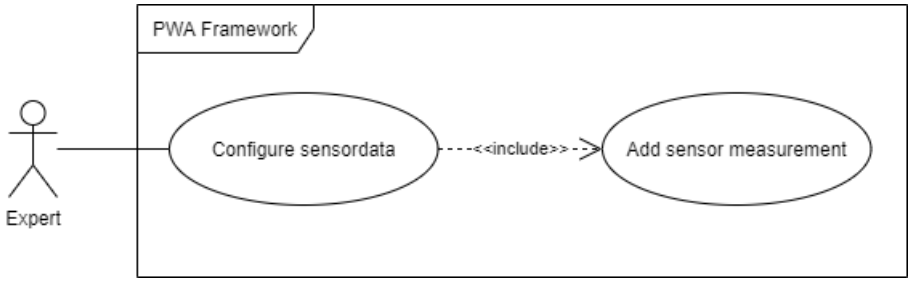

Figure 4.7: Use cases for configuring sensordata

F07 (Configure sensordata): The expert can add one or more sensor measurements that collect data from the end users device.

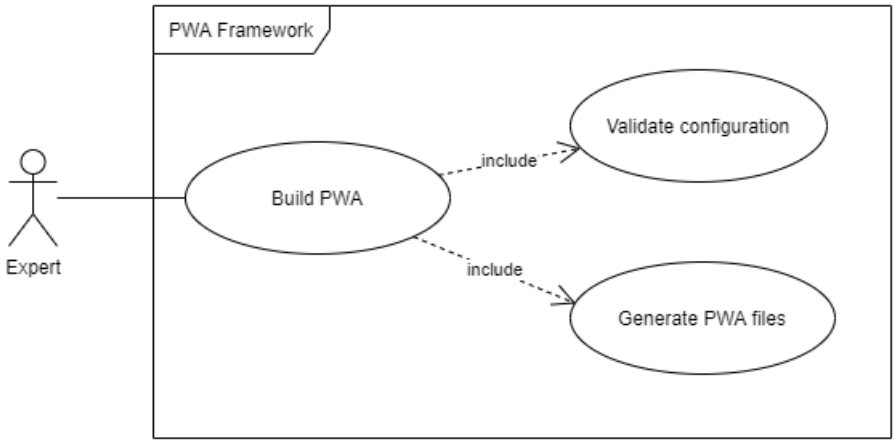

Figure 4.8: Use cases for building the PWA

F08 (Build PWA): This function starts the build process. Before the PWA can be build, the configuration is checked for errors (validate configuration). If there are no errors, the files for the PWA are generated.

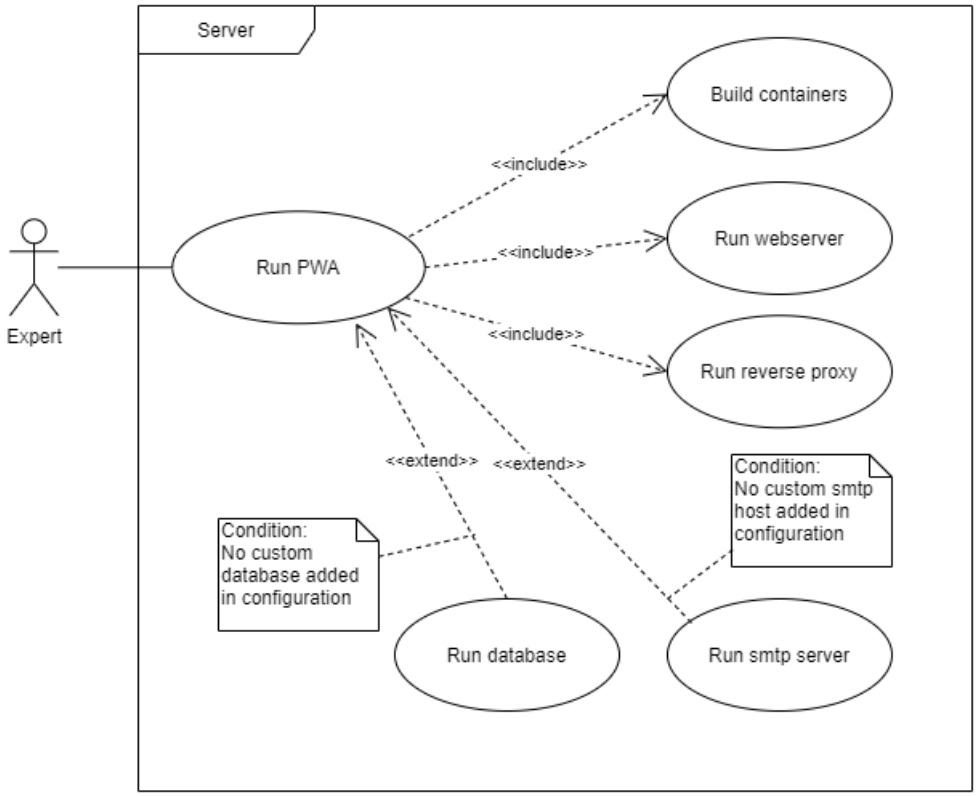

Figure 4.9: Use cases for running the PWA

F09 (Run PWA): When the expert runs the PWA, first, the containers are generated with the files from the previous build process (see F08: Build PWA). Afterwards, the web server and the reverse proxy will be run, and depending on the configuration, the database and SMTP server will be run.

4.2 PWA functionality

The PWA is used by the end user for account management and assessment related functions. The following table shows all functions for the end user when using the PWA.

Code	Function	Priority
P01	Create account	SHOULD
P02	Authorization	SHOULD
P03	Reset password	COULD
P04	Delete account	COULD
P05	Edit notification schedule	MUST
P06	Send notification	MUST
P07	Start assessment	MUST
P08	Do assessment	MUST
P09	Complete assessment	MUST

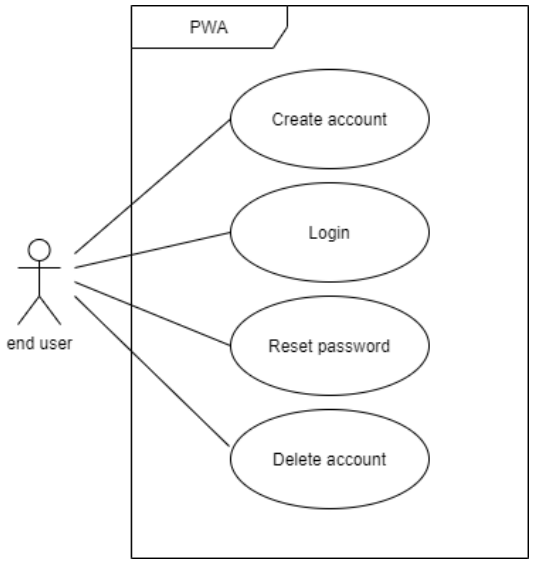

Figure 4.10: Use cases for the account management of the PWA

P01 (Create account): The end user can create an account to persistently store his assessment data on the server. The expert can decide if this step is mandatory and if the user will need an account to use the PWA, or if he can use it without an account.

P02 (Login): If the end user is not logged in, he can log in, which automatically loads his data from the server.

P03 (Reset password): The end user can reset his password by submitting his email address which sends an email to that email address with a link to a reset page where he can choose a new password.

P04 (Delete account): The end user can delete his account with all associated assessment data.

P05 (Edit notification schedule): The end user can decide when and how the PWA sends a notification. This can be either randomized or at fixed times.

P06 (Send notification): The PWA sends out a notification to the end user prompting him to do his assignment.

P07 (Start assessment): The end user can start a new assessment. If the expert has enabled the bluetooth sensor in the configuration, the end user has to choose a bluetooth device where the MCS data will be recorded.

P08 (Do assessment): The end user can go through the assessment by answering questions.

P09 (Complete assessment): When the end user completes an assessment, the assessment and the sensor data is stored. If the end user is logged in and the end users device is connected to the internet, the data will be stored on the server, otherwise it will be stored on the device.

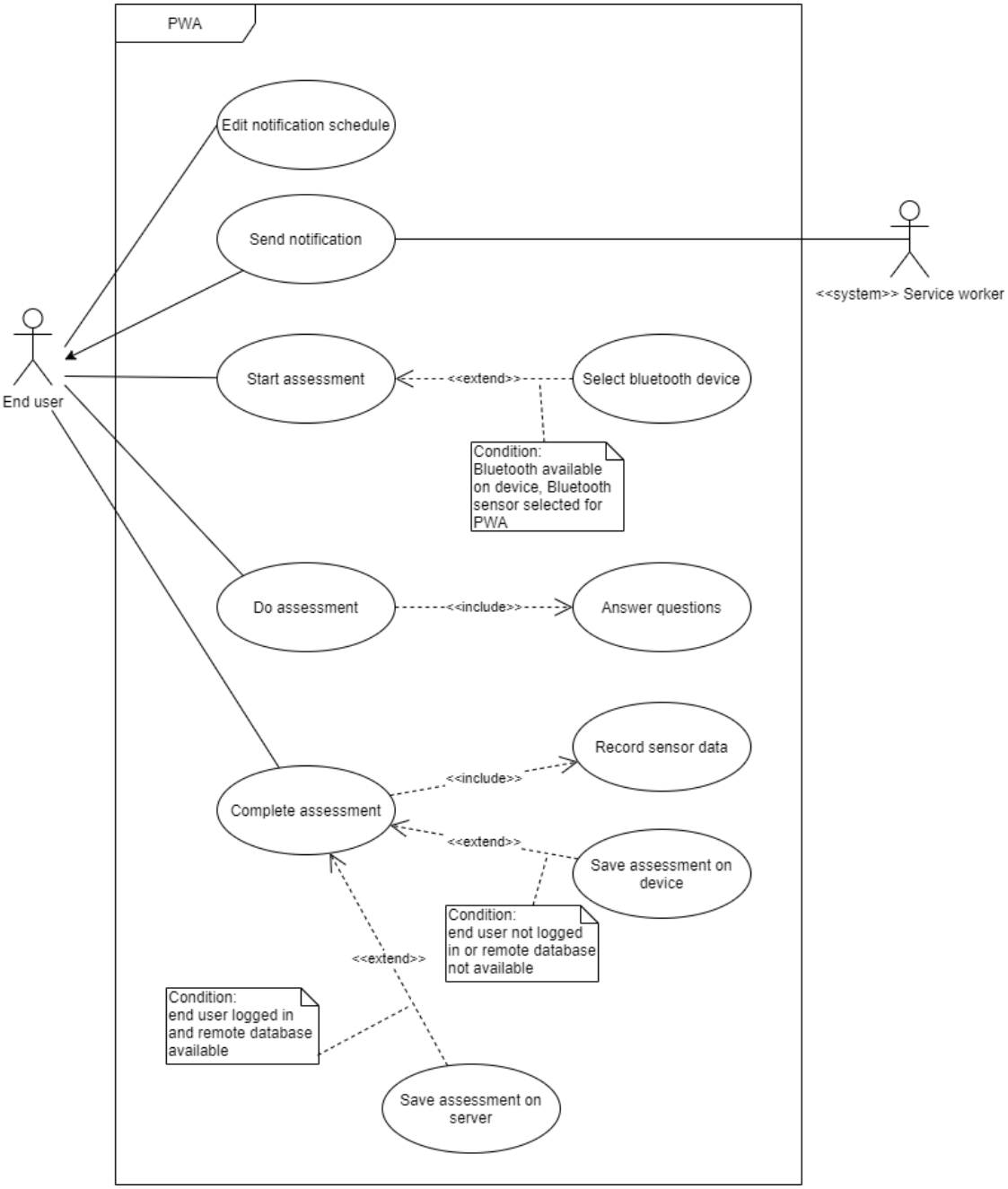

Figure 4.11: Use cases for the PWA

4.3 Non-functional requirements

Besides the functional requirements, the framework and PWA have additional requirements which cannot be described as functions. These non-functional requirements are shown in the following table.

Code	Requirement	Priority
NF1	Reliability	MUST
NF2	Responsive design	SHOULD
NF3	Robustness	SHOULD
NF4	User friendliness	SHOULD
NF5	Documentation	COULD

NF1 (Reliability): The PWA and the framework have to be reliable under any circumstances. Therefore, the user should be able to rely on the software in any situation.

NF2 (Robustness): The PWA and the framework should be able to detect errors like false user input and handle them accordingly by automatically correcting the error or displaying the error to the user.

NF3 (Responsive design): The PWA should be displayable on any device screen, including any screen size from a smartphone display to a desktop monitor. Therefore, the PWA has to adapt its user interface according to the screen size.

NF4 (User friendliness): From the perspective of the end user, the PWA should be easy to use and understand, because he wants to use the PWA on a daily basis and not waste time on learning how to use the PWA.

NF5 (Documentation): The framework should provide a documentation for the expert.

5 Concept

In this chapter, the concept of the software is explained. It includes the browser support, components and their relations and the interface design for the PWA.

5.1 Browser support

This section will compare different browsers for their compatibility with the features of the PWA. Because the PWA is implemented with the focus on mobile users, this comparison only includes mobile browsers. According to statcounter.com [16], 99,39% of browser usage in Europe in the last 12 month is split between Google Chrome, iOS Safari, Samsung Internet, Mozilla Firefox, Opera, UC Browser and Android Browser, therefore the comparison focuses on these seven browsers. Additionally, the comparison distinguishes between Android and iOS browsers, since the availability of some features also depends on the operating system that the end user uses, even when the browser can be the same on both systems (e.g., Google Chrome for Android versus Google Chrome for iOS). The comparison includes the following features:

- Service workers
- Notification API
- IndexedDB
- Web Bluetooth
- Geolocation API
- Sensor API
- MediaStream

	Google Chrome	Samsung Internet	Mozilla Firefox
Service workers [17]	●	●	●
Notification API [18]	●	●	●
IndexedDB [19]	●	●	●
Web Bluetooth [20]	●	●	●
Geolocation API [21]	●	●	●
Sensor API [22]	●	●	●
MediaStream API [23]	●	●	●
● = supported, ● = not supported, ● = no information			

Table 5.1: Browser feature support (part 1)

	UC Browser	Android Browser	Safari iOS	Opera
Service worker [17]	●	●	●	●
Notification API [18]	●	●	●	●
IndexedDB [19]	●	●	●	●
Web Bluetooth [20]	●	●	●	●
Geolocation API [21]	●	●	●	●
Sensor API [22]	●	●	●	●
MediaStream API [23]	●	●	●	●
● = supported, ● = not supported, ● = no information				

Table 5.2: Browser feature support (part 2)

Table 5.1 and Table 5.2 show the supported features for each of the previously mentioned browsers. As a conclusion, Google Chrome and Samsung Internet both support all features. Since Google Chrome has a bigger user base with 62.36% [16], the implementation will focus on this browser.

5.2 Components

The software is split into the server components and client components. The overview for all components is shown in Figure 5.1.

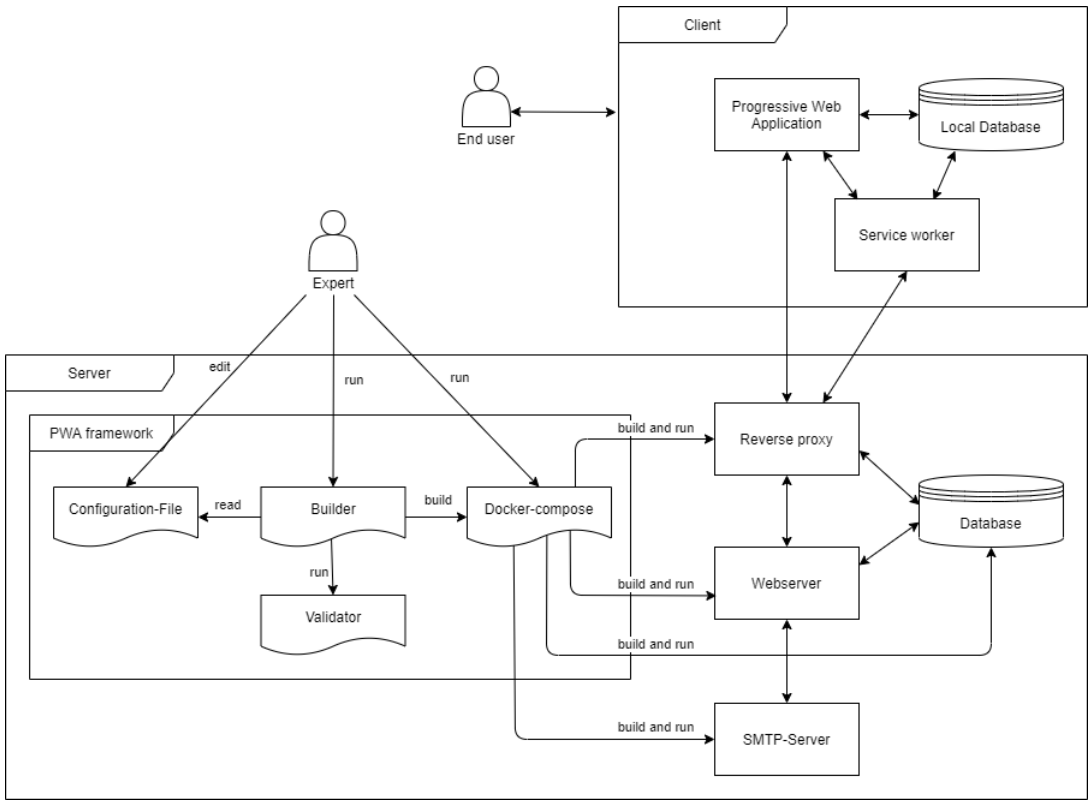

Figure 5.1: Components of the system

5.2.1 Server components

PWA framework: The PWA framework is installed on the server and contains all files to generate the containers that are used to deploy the PWA.

Configuration file: The configuration is edited by the expert and contains all the information about the PWA, for example, assessment questions, sensor configuration and database connection.

Builder: The builder script will be run by the expert after the configuration process is done. The builder reads the information from the configuration file and then runs the validator, before building the docker-compose script.

Validator: The validator script is used by the builder to check for errors in the configuration file.

Docker compose: The docker-compose script can run multiple containers at once. After the building process, the expert can run this script to build the container im-

ages and run all required containers simultaneously.

Reverse proxy: The reverse proxy automatically adds the TLS-certificate for encrypted communication between the server and the client over HTTPS. Additionally, it defines subdomains for the web server and the database and redirect incoming requests on these subdomains from the client to the web server and database.

Web server: The web server processes incoming requests and returns all necessary files to display the PWA on the end users device.

Database: The database stores the user information and assessment data of the end user.

SMTP server: The SMTP server is used to send out E-Mails to the end user when he wants to register an account or forgets his password.

5.2.2 Client components

PWA: The PWA is installed on the end users device and communicates with the local database, the service worker and the server.

Local database: The local database on the end users device stores assessments when the user does not have an account, or if the client is not connected to the internet. When the end user is logging in, or the client is connecting to the internet, all data on the local database is synchronized with the servers database.

Service Worker: The Service Worker is loading the cached website data if the client is offline. It also sends out the notification according to the notification schedule that is stored in the local database.

5.3 Web sensors

Modern browsers offer a wide array of sensors that can be accessed from a website. The sensordata can be accessed over an API (provided the device has that sensor built in). The following sensors are currently[1] available:

[1] As of December 8, 2020

Bluetooth API: The Web Bluetooth API[2] provides an interface to connect to Bluetooth Low Energy peripherals and communicate with them. It relies on the Generic Attribute Profile (GATT) protocol.

Geolocation API: The Geolocation API[3] can be used to get the current location of the end users device. Additionally, the speed and the direction where the device is heading can be tracked.

MediaDevices interface: The MediaDevices interface[4] provides access to the microphone and the camera of a device. It can also be used for screen sharing.

Sensor API: The Web Sensor API[5] provides access to the accelerometer, gyroscope and magnetometer, which can be used to determine the absolute (in relation to the Earth's reference coordinate system) or relative (without relation to the Earth's reference coordinate system) orientation of the device for example. Additionally, it provides access to the ambient light sensor to measure the luminosity in the area around the device.

The thesis will focus on a fixed set of sensors, therefore not all sensors will be included in the implementation. The following sensors will be included in the implementation:

- Microphone (MediaDevices interface): For recording background noise or other audio while the end user completes his assignment.
- Location, speed (Geolocation API): For recording the location and speed of the end user.
- Bluetooth: Let the end user connect to a bluetooth device (e.g., to record the heart rate with a corresponding bluetooth peripheral).
- Ambient light sensor (Sensor API): To record the luminosity.

[2] https://developer.mozilla.org/en-US/docs/Web/API/Web_Bluetooth_API
[3] https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API
[4] https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices
[5] https://developer.mozilla.org/en-US/docs/Web/API/Sensor_APIs

5.4 Interface Design for the PWA

Even though the PWA will focus on the end user with a mobile device, it will also be available for end users with other devices. For the PWA to be accessible on any device with a browser while still providing a user friendly interface, the interface of the PWA has to adjust to different screen sizes. This is accomplished by using a responsive design. This section will go over two versions of the interface: the mobile version for smartphones and the desktop version for bigger screens. Even though only two screen sizes are shown, the interface can adapt to any screen size.

5.4.1 Mobile version

The mobile version has a single column layout with an expandable navigation menu to use minimal space. Figure 5.2 (left) shows the navigation bar with the logo and the name of the PWA at the top of the page (1), the assessment content in the middle (2) and the navigation between the assessment question at the bottom (3). Figure 5.6 (right) shows the navigation menu (5) that can be collapsed and extended by using the menu button (4).

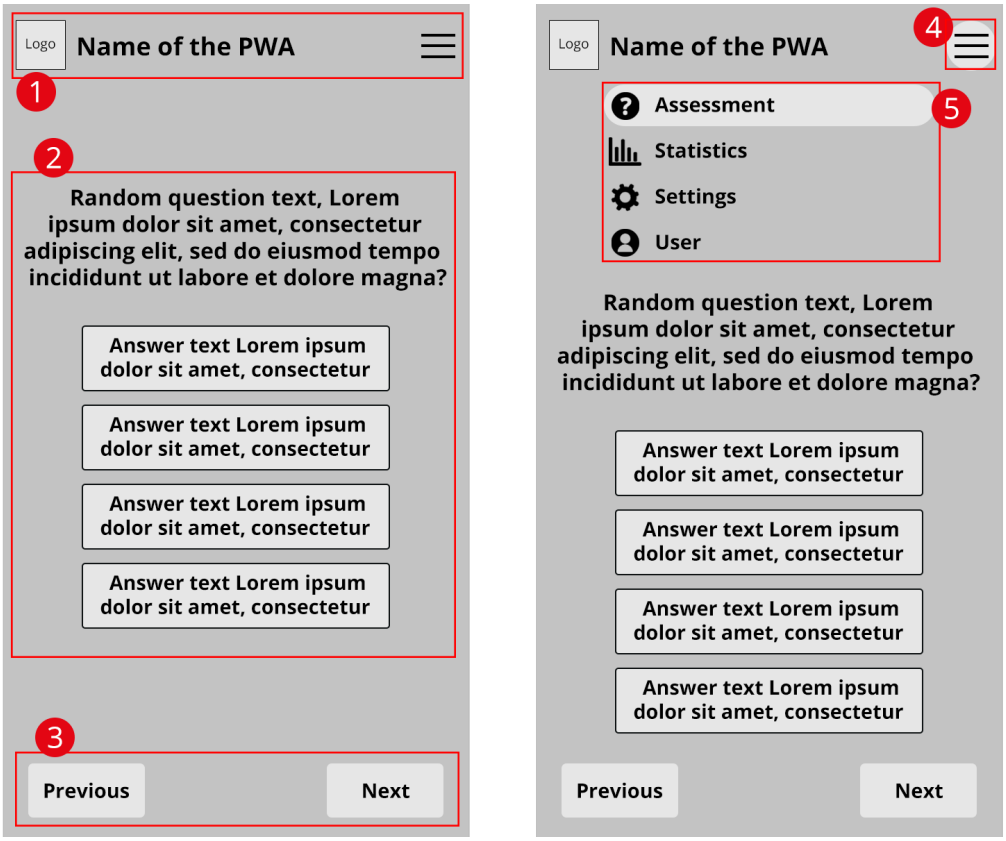

Figure 5.2: Wireframe for the mobile interface in assessment view

The statistics view (see Figure 5.3) has an assessment selector (6) on the top that toggles the list for all assessments (8) that were completed, ordered by date (newest first). When the end user selects an assessment from the list, the content of the assessment with all questions and answers is displayed (7).

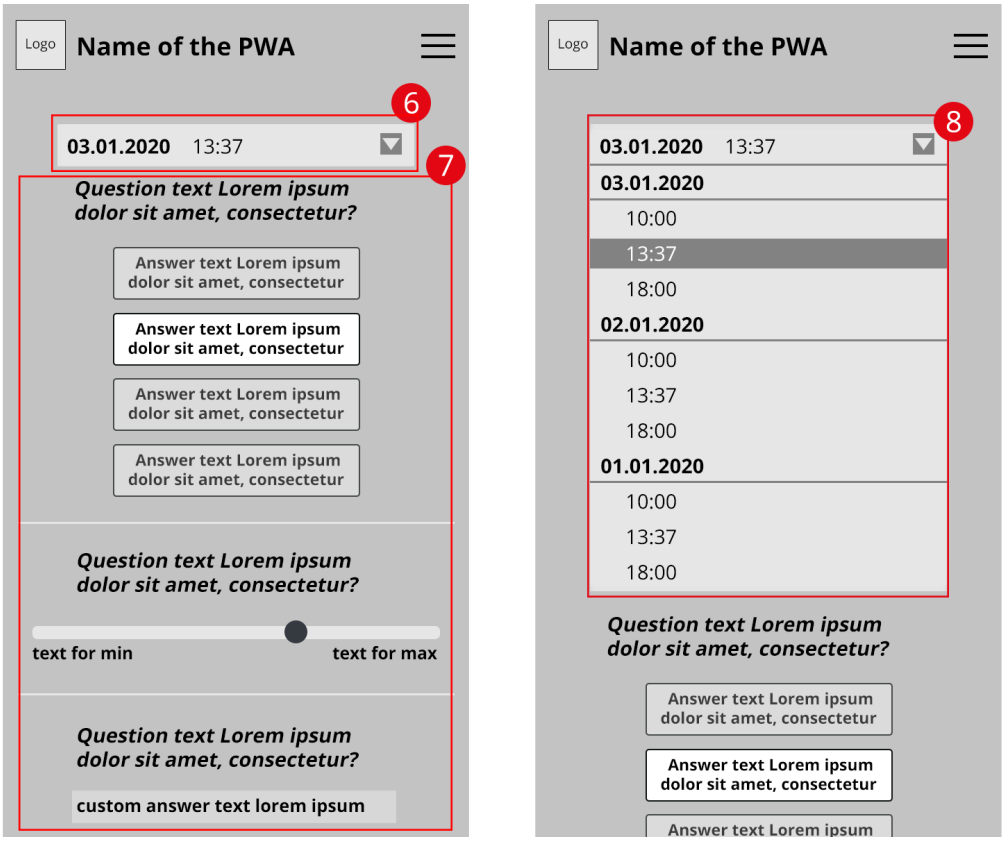

Figure 5.3: Wireframe for the mobile interface in statistics view

The settings menu (see Figure 5.3) consists of the notification type selector (9) where the user can switch between random and custom notifications. If the notification type “random“ is selected, the end user can choose the number of notifications per day (10) as well as the notification times for every day of the week (11). If the notification type is “custom“, he can choose the notification times (12) and add (13) or delete (14) additional notification times. The notification settings can be saved by pressing the save button (15) or discarded by closing the settings menu (16). Figure 5.5 shows the input form for the login (left) and the registration (right).

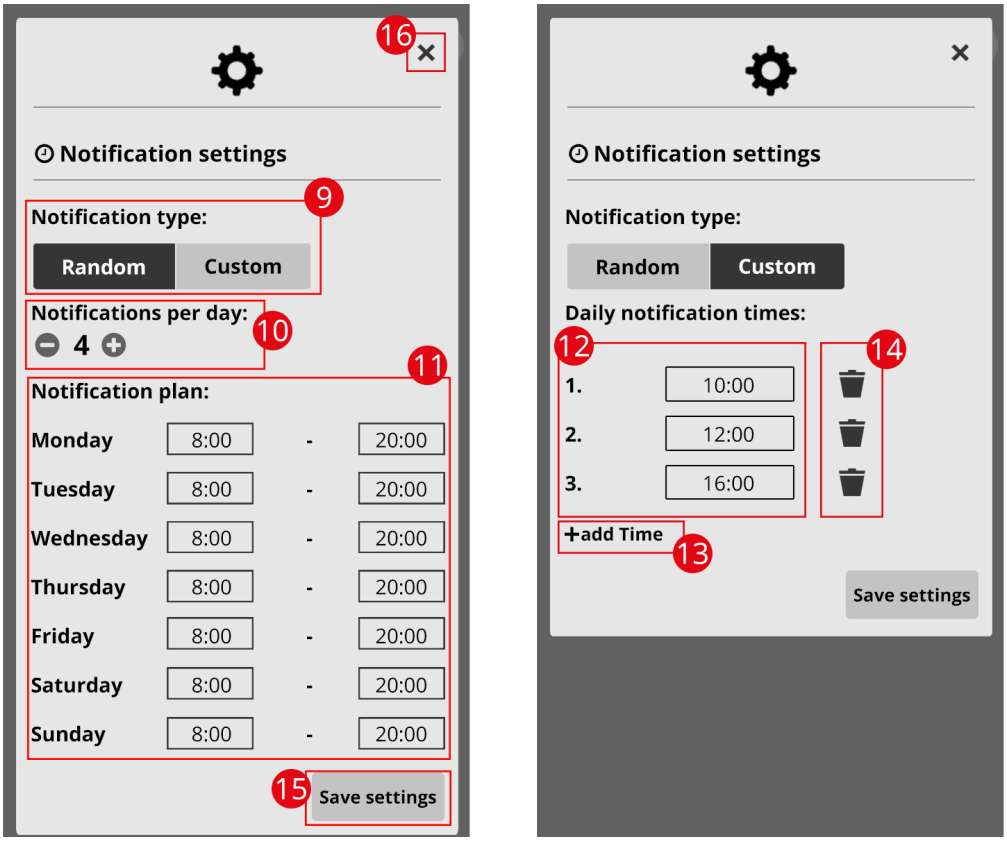

Figure 5.4: Wireframe for the mobile interface in settings menu view

Figure 5.5: Wireframe for the mobile interface in login menu view

5.4.2 Desktop version

The desktop version of the user interface is similar to the mobile interface, with the exception that it uses the wider screen size to display more content without having to hide information. Figure 5.6 shows the assessment view with the navigation bar on top that uses icons only.

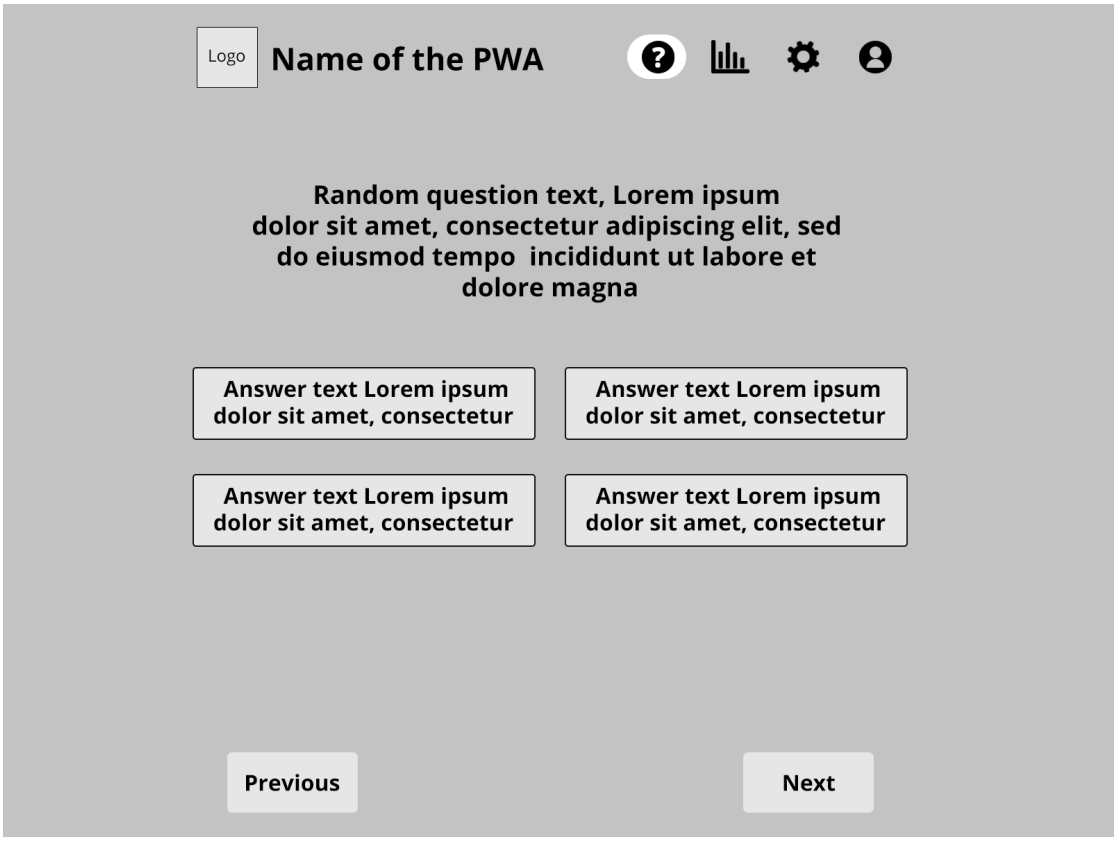

Figure 5.6: Wireframe for the desktop interface in assessment view

The statistics view (see Figure 5.7) displays the assessment list and the assessment content next to each other in contrast to the expandable assessment list in mobile view.

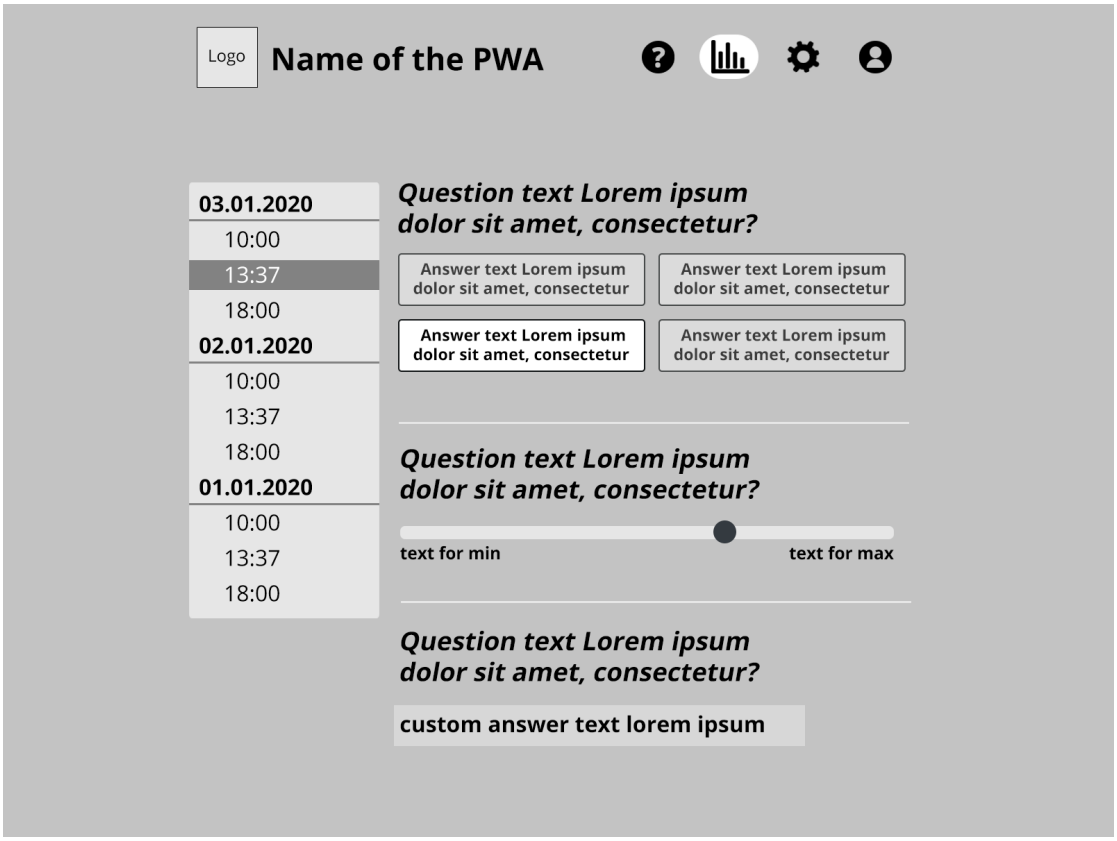

Figure 5.7: Wireframe for the desktop interface in statistics view

6 Implementation

In this chapter the implementation of the framework will be explained.

6.1 Notification scheduling implementation

The PWA will frequently remind the end user to complete his assessments. The notifications will we displayed with the help of the Notification API [1]. They can be scheduled in two different ways:

Random: The times for the notification will be picked randomly in a given period of time. The end user can pick the number of notifications and the time period for every day of the week in which the notifications can appear.

Fixed: The notification times will have a fixed time. The end user can decide when and how often he wants the notifications to appear on a daily basis.

The notification schedule is stored as a document in the local database of the end users device. Figure 6.1 shows the process of scheduling notifications. This process is carried out by the service worker and it is started whenever one of the following events occur:

- the end user logs in
- an assessment is completed
- the page is reloaded

First, the service worker checks if there is a notification schedule stored in the local database. If not, a new notification schedule is created. If there is a notification

[1] https://developer.mozilla.org/en-US/docs/Web/API/Notifications_API

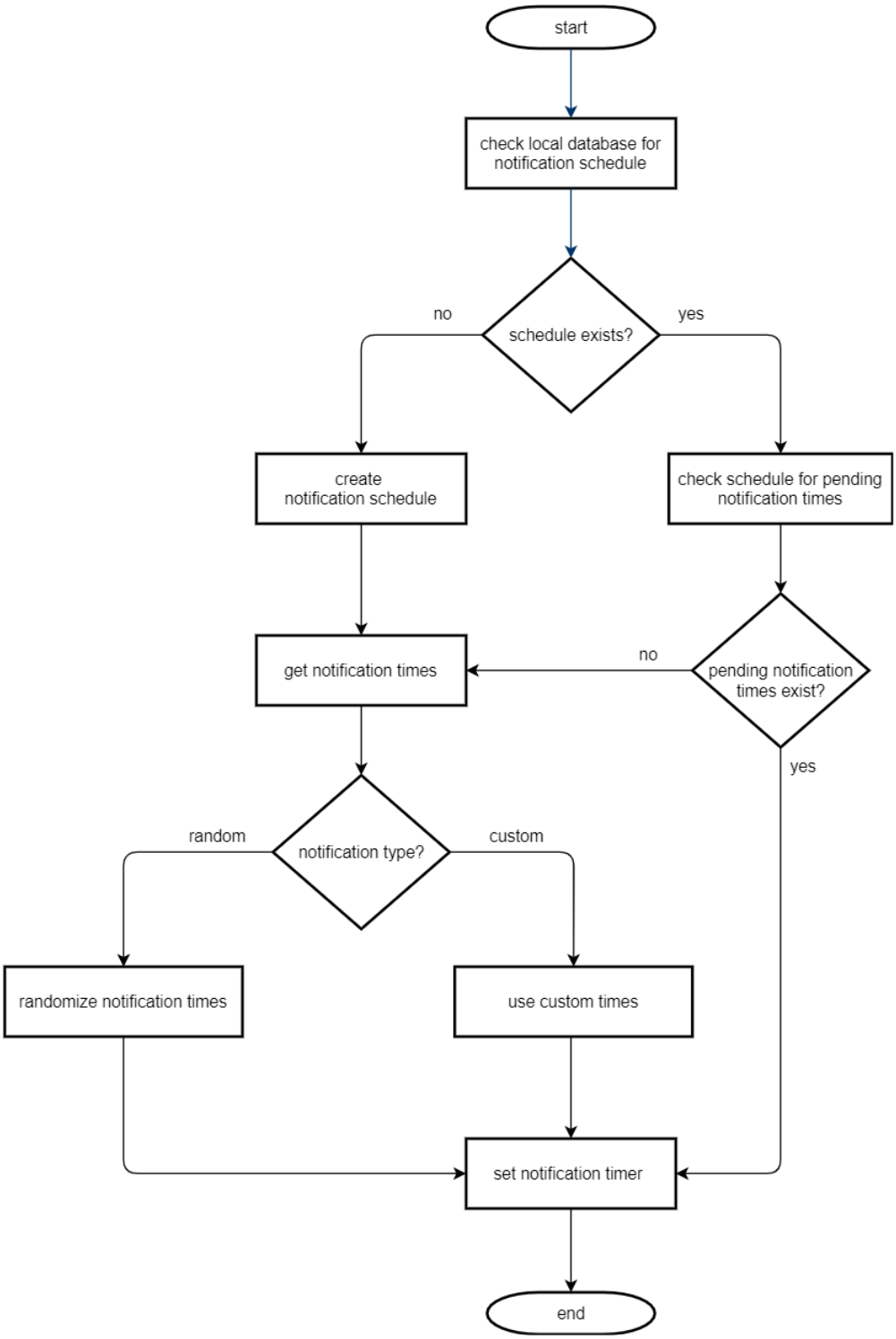

Figure 6.1: Notification scheduling flowchart

schedule in the local database, the service worker then checks if there are any pending notification times. This only includes date times that lie in the future, any notification times that are in the past are ignored. If there are any pending notification times, the notification timer will be set and the process is completed. If there are no pending notification times or a new notification schedule was created, the service worker will create a set of notification times based on the notification type and sets the notification timer afterwards.

Figure 6.2 shows the algorithm for the function that generates notification times if the end user chose the notification type “random“. The number of iterations equals the number of notifications that the end user selected (in the algorithm called “set“). The start- and end-time equal the first and last time of the day when a notification can be sent and also can be selected by the end-user. The function returns an array of date times.

6.2 Offline access implementation

Service workers make it possible to use websites and web applications offline by storing web files in the cache of the browser. Additionally, modern browsers provide local databases to persistently store website data on the client side. This section describes the implementation of the offline availability for the PWA and the local database.

6.2.1 Offline caching

The caching of website data is implemented with the help of Workbox[2] in the service worker script. Workbox is a JavaScript library that provides packages for routing and strategies. These packages are used to determine if and how the service worker should react to an incoming request from the PWA. The following strategies are used in the PWA:

- NetworkFirst: The service worker will first try to fetch the resource from the network. If the resource could not be fetched, the service worker will fall back

[2] https://developers.google.com/web/tools/workbox

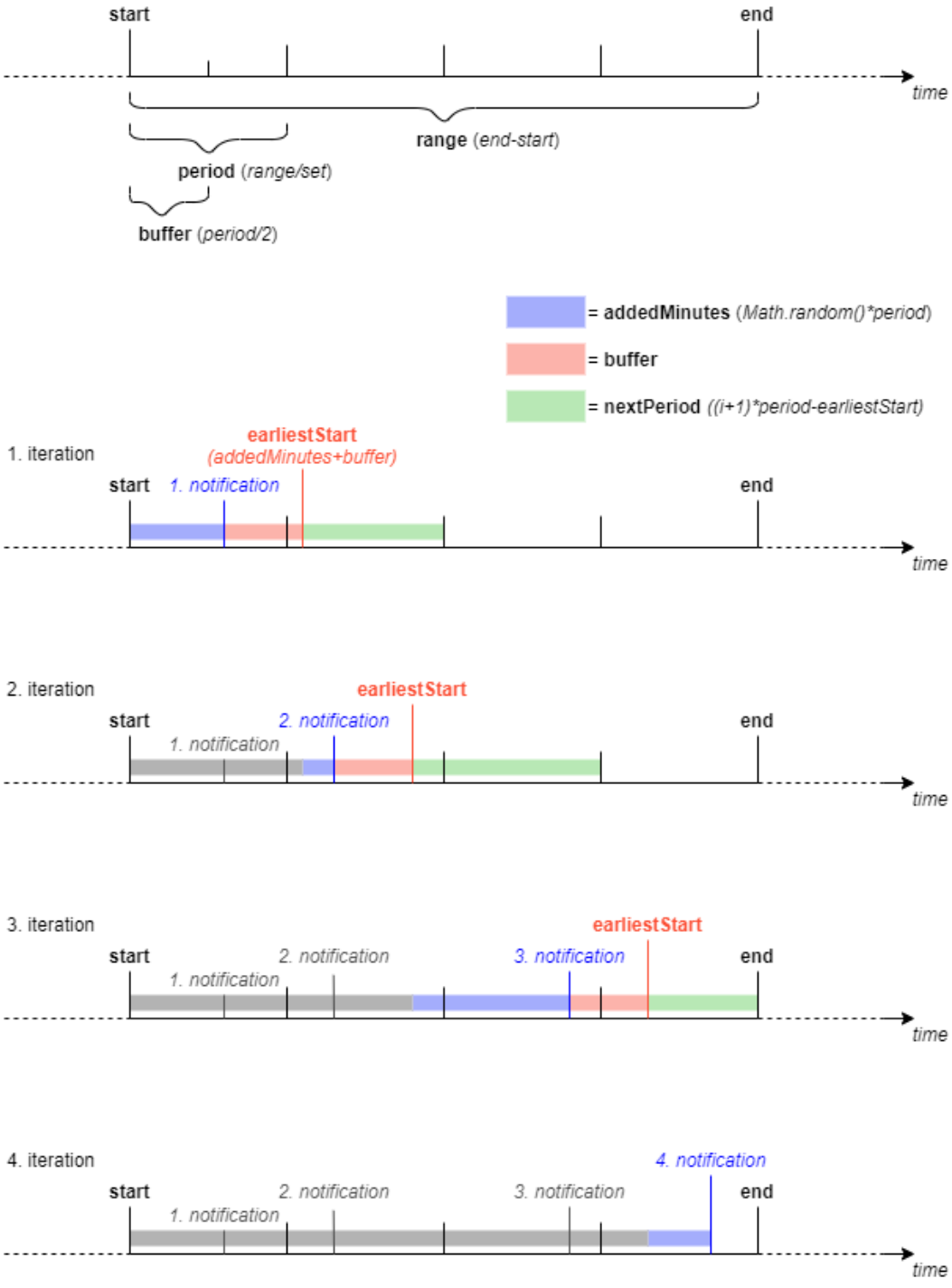

Figure 6.2: Notification time randomization process

to the resource that is stored in the cache.

- CacheFirst: The service worker will first try to fetch the resource from the cache and will fall back to the network if the resource could not be fetched.

The usage of a caching strategy is implemented by registering a route with the `registerRoute()` function. The route is defined by using a regular expression that includes all files of a type (e.g., all files that end with ".js") and assigning the caching strategy to that route (e.g., `NetworkFirst`).

6.2.2 Offline database

The offline mode for the database is implemented with the help of the JavaScript library pouchDB[3]. PouchDB can synchronize between the local database and the database on the server to ensure data integrity. It uses the indexedDB API[4] that modern browsers provide to store data on the end users device. Whenever an end user wants to store an assessment, the PWA checks if the user is online and logged in. If this is the case, the data will be stored on the server database and immediately synchronized with the local database. If the end user is offline, the data will be stored on the local database instead and will be synchronized with the server database as soon as the end users device connects to the internet. If the expert allowed the end user to use the PWA without an account (see Section 6.6.3), the assessment data will be stored on the local database only. If the end user then creates an account and logs in, the assessment data on the local database will be transferred to the server database.

6.3 Sensor implementation

The expert can define what sensor data should be recorded when the end user is completing an assessment. All sensors require the permission of the end user. This permission can be given by accepting the popup which appears when the end user starts an assessment for the first time.

[3] https://pouchdb.com/

[4] https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API

6.3.1 Microphone

The expert can set a limit for the time the audio will be recorded per assessment session. The maximum time limit is five minutes. This limit is given due to the storage capacity that the server database and local database have to avoid unnecessary data spam. The recording of the audio starts as soon as the end user starts a new assessment and accepts the permission popup. The recording of the audio will stop when one of the following events occur:

- The time limit that the expert has set has been reached
- The audio recording reaches five minutes
- The end user finishes the assessment

If the PWA is put in the background because the end user opens another tab or switches between applications on his device, the recording will be paused. The recording will be resumed as soon as the PWA is focused again. The audio file will be stored as a binary large object (BLOB) in the database.

6.3.2 Geolocation

The location of the end user can be recorded continuously throughout the assessment, at the start or at the end of the assessment. If the location is recorded continuously, the data that is stored with the assessment includes the first tracked position when the end user starts a new assessment, the last tracked position when the end user finishes his assessment as well as the speed and direction where the end user is heading. Otherwise, only the location at the start or the end will be stored with the assessment data.

6.3.3 Bluetooth

The expert has to specify a Generic Attribute Profile (GATT)[5] service that is later used to filter the list of nearby Bluetooth devices that the end user can choose from. The end user is prompted to select a Bluetooth device with the GATT service when

[5] https://www.bluetooth.com/specifications/gatt/

he starts a new assessment. The data from the Bluetooth service can be recorded continuously throughout the assessment, at the start or the end of the assessment.

6.3.4 Ambient light sensor

To record the brightness around the end user, the ambient light sensor can be set by the expert to either measure once at the start, at the end of the assessment or measure an average value over the time of the assessment.

6.4 Builder implementation

The builder is a node.js script that creates all files to run the PWA. Before the building process can start, the configuration of the PWA is checked for errors with the help of the validator. If the configuration does not have any errors, the building process will start. Otherwise, the errors will be displayed to the expert and the builder will exit.
The files that will be generated in the building process are based on templates. The data that is coming from the configuration file will be edited by escaping special coding characters (e.g., “<“ for HTML-markup) to prevent unwanted behaviour that can result in bugs in the PWA. The builder reads the template file and replaces a set of placeholder strings before saving the document as new file. The following files are created in the building process:

- Database Dockerfile
- docker-compose.yml
- index.js (web server)
- webmanifest
- service-worker.js
- HTML and CSS files

If the expert did not specify an external database connection, the builder generates random admin credentials for the database. The credentials consist of a ten character long random string for the username and the password and will be stored as

environment variables in the database Dockerfile.
The expert can place a logo in the template folder that will then be edited and saved in four different sizes. The different sizes cover the following browser requirements:

- 192x192 pixel, 512x512 pixel for the webmanifest when the end user installs the PWA on his device
- 180x180 pixel for iOS Safaris “add to homescreen“ feature
- 32x32 pixel for the default favicon in any modern browser

6.5 Docker implementation

The PWA framework uses Docker as containerization engine. Docker offers a tool called docker-compose which can be used to run multi-container applications like the PWA. Depending on the experts configuration, the PWA can contain two to four containers. The containers include the reverse proxy, the web server, the database and the SMTP server. The reverse proxy and the web server containers will always be included in the docker-compose, while the database and SMTP server containers can be excluded if the expert decides to use an external database or SMTP server that he can specify in the configuration.
Docker-compose uses Dockerfiles to build container images, which can then be run as containers. These Dockerfiles use container images from dockerhub[6] as base images, and build their own images on top of it.
The Dockerfile for the web server uses node as the base image and copies all PWA files, including the files that were created in the build process, into a directory in the web server container. Afterwards, all node packages for the web server will be installed in the container. When all packages are installed, the web server is started by running the `node index.js` command which starts the express.js application.
The Dockerfile for the database uses couchdb as base image and sets the admin credentials that were previously generated in the build process as environmental variables. The database configuration that was also created in the build process is then copied into a directory in the database container.
Figure 6.3 shows the sequence of events when running the docker-compose file.

[6] https://hub.docker.com/

After building the container images, the database and SMTP server containers will be started (provided that no external database or SMTP server have been specified). When these containers are running, the web server will be started and will then authorize itself for the database and the SMTP server via access credentials. In the last step, the reverse proxy container will be started and the PWA is accessible for the end user.

6.6 Configuration and validation implementation

The expert can configure the PWA by editing the configuration file. Appendix A shows an example for a valid PWA configuration file. The configuration can be split into these parts:

- Assessment configuration
- Database configuration
- Registration configuration
- Appearance configuration
- SMTP configuration
- Server configuration
- Sensor data configuration

The following sections will explain each configuration and the validation that will be checked by the validator in the build process.

6.6.1 Assessment

The expert can choose between specifying the assessment with all related questions in the configuration file, or provide an external assessment endpoint URL from where the assessment can be imported. In both cases, the assessment must have the given structure that is shown in Figure 6.4. If the assessment is imported from an external endpoint, the endpoint must provide the assessment in JSON format.

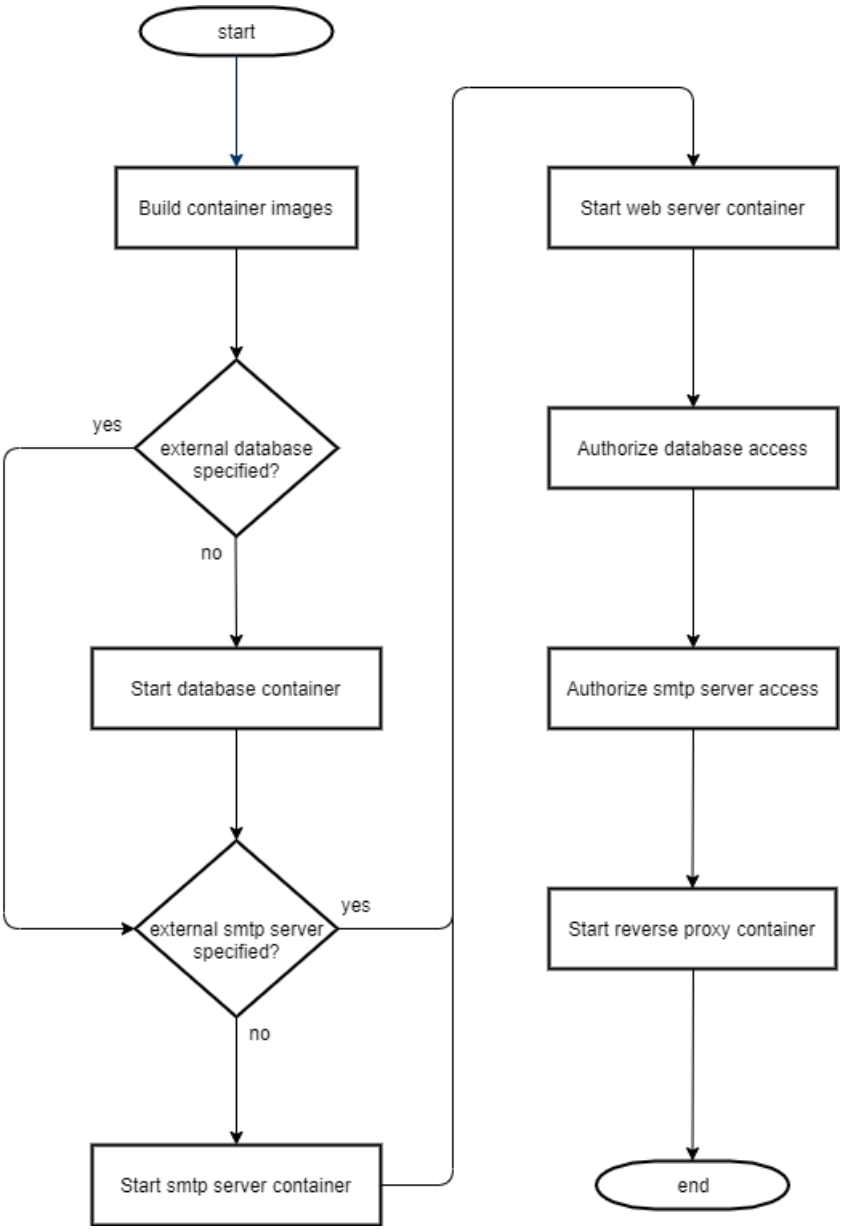

Figure 6.3: Flowchart for running the PWA

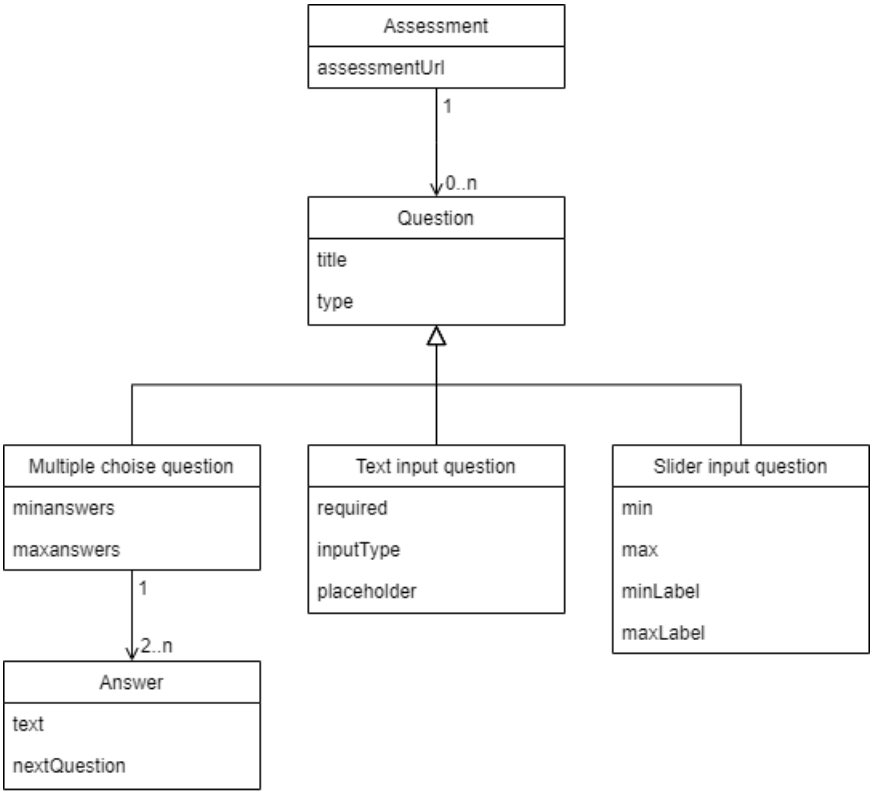

Figure 6.4: Class representation of the assessment configuration

An assessment consists of one or more questions. Every question has a `title` and a `type`. The title allows the expert to use custom HTML markup to embed additional media, for example images and videos. There are three different question types: multiple choice questions, text input questions and range input questions.

- Multiple choice questions who have two or more answers that the end user can choose from. Additionally, multiple choice questions have the attributes `minanswers` and `maxanswers`. `minanswers` defines the minimum number of answers that the end user has to select to be able to move on to the next question. `maxanswers` defines the maximum number of answers that the end user can select. Every answer has a title and an optional `nextQuestion` attribute that defines which question comes next when the answer is the only one that is selected by the end user. This gives the expert the possibility to control the flow of the assessment by creating followup questions depending on the answers of the end user.
- Range input questions that let the end user select a number in a range that is specified by the expert. This can be used, for example, to let the end user assess the intensity of a symptom on a scale of 1 to 10. The range selector has the attributes `min` and `max` to define the range of the scale as integers. Additionally, the expert can set the attributes `minLabel` and `maxLabel` to add a label to the minimum and maximum value of the range selector (e.g., “bad“ and “good“).
- Text input questions that let the end user write an answer in text form. The attribute `required` can be set to true if the expert wants the end user to give an answer before moving on to the next question. The `inputType` defines the type of answer that the end user can give. This includes numbers, dates (e.g., “10.10.2020“), datetimes (e.g.,“1.10.2020 20:14“) and times (e.g., “19:30“). The input will be validated before the user can move on to the next question and display an error if the end user entered an invalid value. The attribute `placeholder` can be used to give the end user additional information on the text input.

Listing 6.1 shows an example for the assessment configuration including four questions with three different question types.

6.6.2 Database

The database attribute `type` specifies whether the database that stores all the user and assessment data will be created in the build process and run as a container on the server (`type = auto`), or if the PWA will use an external database (`type = custom`). If an external database is used, the expert has to provide a database URL (`databaseUrl`) and user credentials (`user` and `password`). The user must have the rights to add, edit and delete users. Similar to the internal database (see Section 6.8.2), the external database must be a CouchDB database with the same configuration. The validator checks if the provided database URL and admin credentials are valid by sending a login request to the external database. If the external database can not be reached or the credentials are incorrect, an error message will be displayed to the expert.

6.6.3 Registration

With the `required` attribute, the expert can specify whether the end user needs an account. If `required` is set to `true`, the user is forced to register an account before he can use the PWA. If it is set to `false`, the end user can use the PWA without an account and the data will be stored on the local database on the end users device instead of on the servers database.
The registration form can be extended by adding additional input fields with the `userInfo` list. The list consists of one or more user information objects. Each object has a label and an optional `required` attribute which specifies if the end user must fill out the input field. An input field can also have a type. The valid types include numbers, dates, datetimes and times, similar to the input type in a text input question (see Subsection 6.6.1) and an additional type for email addresses (`type = email`). Listing 6.2 shows an example for a registration configuration with a user information where the end user has to enter his age as a number.

```
"assessment": {
    "questions":[
      { "type": "MULTIPLE_CHOICE",
        "title": "This is the first question",
        "minanswers": 1,
        "maxanswers":1,
        "answers":[
          {
            "text": "this is the first answer",
            "nextQuestion": 2
          },
          {
            "text": "this is the second answer"
          },
          {
            "text": "this is the third answer"
          }
        ]
      },
      { "type": 1,
        "title": "<h3>This is the second question with an embedded
          video</h3><iframe width=\"420\"  height=\"315\"
          src=\"http://url.to.video/\">
          </iframe>",
        "minanswers": 1,
        "maxanswers":1,
        "answers":[
          {
            "text": "this is the first answer",
            "nextQuestion": 2
          },
          {
            "text": "this is the second answer"
          },
          {
            "text": "this is the third answer"
          }
        ]
      },
      { "type": 2,
        "title": "this is the third question with a slider",
        "min":1,
        "max":10
      },
      { "type": 3,
        "required":true,
        "title": "this is the fourth question with text input",
        "placeholder":"placeholder text"
      }
    ]
}
```

Listing 6.1: Example for a assessment configuration

```
"registration": {
    "required": false,
    "userInfo":[
        {
          "label": "Age",
          "required": true,
          "type":"number"
        }
    ]
}
```

Listing 6.2: Example for a registration configuration

6.6.4 Appearance

The appearance of the PWA can be edited by adding a logo, specifying two colors, a name for the PWA as well as a welcome message and a notification message. Editing other design parameters like layout and typography are not included to prevent the expert from affecting the user friendliness (e.g., by decreasing the font size).
The name of the PWA is used in the navigation bar, the website title and in the webmanifest (e.g., for displaying the name on startup screens) and is limited to a length of 50 characters.
The two colors of the PWA can be specified with the attributes `mainColor` and `secondaryColor`. The main color is primarily used for the background of the PWA while the secondary color is used for buttons and navigation elements. Both colors must be specified as six digit hex color codes (e.g., `#1337ff`), otherwise the validator will throw an error.
The logo of the PWA is not specified in the configuration file, instead the expert places the logo in form of a JPEG, PNG or BMP file in the logo folder of the framework. The logo should come in square format with equal width and height of at least 512 pixels to avoid image distortion and blur after the scaling process in the builder. Similar to the name of the PWA, the logo is used in the navigation bar, the website title and in the webmanifest. The welcome message (`welcomeMessage`) is shown to the end user when he visits the PWA for the first time and can contain custom HTML markup. The notification message (`notificationMessage`) is shown in the notification for the end user. Listing 6.3 shows an example for an appearance configuration.

```
"appearance":{
    "name": "Testname PWA",
    "mainColor":"#184a6e",
    "secondaryColor" : "#7da468",
    "welcomeMessage": "<h3>This is a custom welcome message for the PWA which
      can be edited in the configuration</h3><h4 style=\"color:#d33\">
      Custom style is also possible</h4>",
    "notificationMessage": "Notification message to prompt the user for
      his assessment"
}
```

Listing 6.3: Example for a appearance configuration

6.6.5 SMTP server

Similar to the database configuration, the expert can provide an external SMTP server that will be used to send emails. If the expert wants to use an external SMTP server, he has to set the `type` to `custom` and specify a host URL (`hostUrl`) and user credentials (`user` and `password`), otherwise he can set the `type` to `auto`.

6.6.6 Server

In the server configuration, the expert has to provide basic information about the server that the framework is running on. This includes the IP address (`ipAddress`), the server domain (`domain`) and the email address that is used to generate the TLS certificate for the server (`certificateEmailAddress`) with the certificate authority Let's Encrypt.

6.6.7 Sensor data

In the sensor data configuration the expert can specify one or more sensors.
Figure 6.5 shows the structure of the sensor data configuration as a class diagram. With the `sensortype` attribute, the expert defines the type of the sensor, including: microphone, geolocation, lux or bluetooth. If the type is `microphone`, the expert can set the maximum length of the recording with optional attribute `maxLength`. This attribute must be a value between `0:01` and `5:00`, which equals the recording time in minutes.
For the geolocation, lux and bluetooth sensor, the attribute `captureTime` can be

set, which is used to specify the point in time when the data will be recorded. This attribute can have the values `start`, `end` and `continuous`. The bluetooth sensor has an additional attribute called `service` that consists of a valid bluetooth UUID[7]. Listing 6.4 shows an example for a sensordata configuration with a microphone sensor that records for ten seconds.

```
"sensordata":{[
        {
        "sensor":"microphone",
        "maxLength":"0:10"
        }
    ]
}
```

Listing 6.4: Example for a sensordata configuration

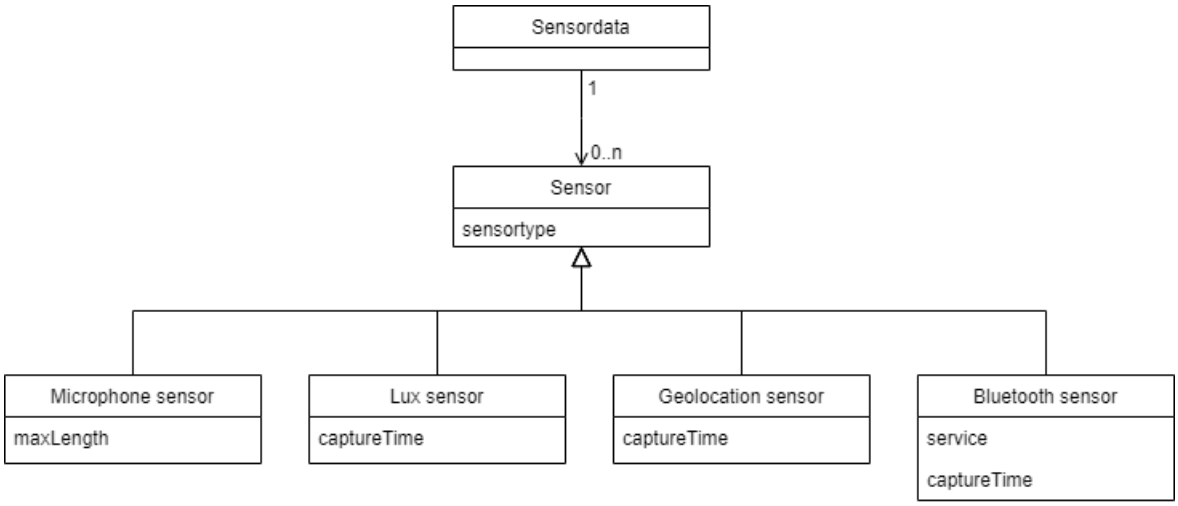

Figure 6.5: Class representation of the sensor data configuration

6.7 Reverse proxy implementation

For the reverse proxy, the framework uses Traefik[8]. It is used to assign subdomains for the web server and database and automatically generate TLS certificates for a

[7] https://www.bluetooth.com/specifications/assigned-numbers/service-discovery/
[8] https://doc.traefik.io/traefik/

secure connection over HTTPS. Traefik is configured in the docker-compose file. The ports where the reverse proxy listens for incoming traffic are called "EntryPoints". There are two EntryPoints defined for the reverse proxy: "web" with the port number 443 for secure traffic over HTTPS, and "webinsecure" with the port number 80 for traffic over HTTP. Since the PWA will run exclusively over HTTPS, the EntryPoint for HTTP traffic with the port number 80 is only there to redirect the end user to the secure EntryPoint with the port number 443.
For automatically generating a TLS certificate, Traefik uses a CertificateResolver. The CertificateResolver defines the type of challenge for generating and renewing ACME certificates (in this case a TLS challenge), the email address that is required to generate certificates (specified by the expert in the configuration file), the name of the folder where the certificate will be stored inside the container and the EntryPoint that is used for the HTTPS connection (EntryPoint "web" with the port 443).
After configuring the CertificateResolver, the web server and database container have to be connected to the reverse proxy. This also happens in the docker-compose file with the help of labels. The labels associate the container with the CertificateResolver and the HTTPS EntryPoint, and define a subdomain for each of the containers. The subdomains for the web server and the database are"pwa.domain.xy" and "db.domain.xy", where "domain.xy" is the domain of the server that the expert specified in the configuration file.

6.8 PWA implementation

When the build process is completed and all containers are running, the end user can access the PWA over the generated subdomain (see Section 6.7). The PWA can be split into two main parts: The frontend, where the content is presented and the end user interaction takes place, and the backend, where the requests from the end user get handled and the data is stored.

6.8.1 Frontend

In this thesis, the frontend refers to all components that operate on the client side (the device of the end user). This includes the PWA, the service worker and the lo-

cal database. While the service worker (see Section 6.2.1) and the local database (see Section 6.2.2) are described in a separate section, this section focuses on the PWA by going into detail on the user interface and frontend functionality with the help of screenshots from the final implementation.
The PWA uses basic web technologies like HTML, CSS and JavaScript in combination with the frameworks Bootstrap[9] and jQuery[10]. Bootstrap is a CSS framework and it is used for the responsive design for the PWA, while jQuery is a JavaScript framework that helps implementing the frontend functionality with DOM manipulation.
The configuration that is used for the PWA in the screenshots can be found in Appendix A, with the exception of Figure 6.6, where the screenshot shows two alternative configurations for the registration (see Section 6.6.3).

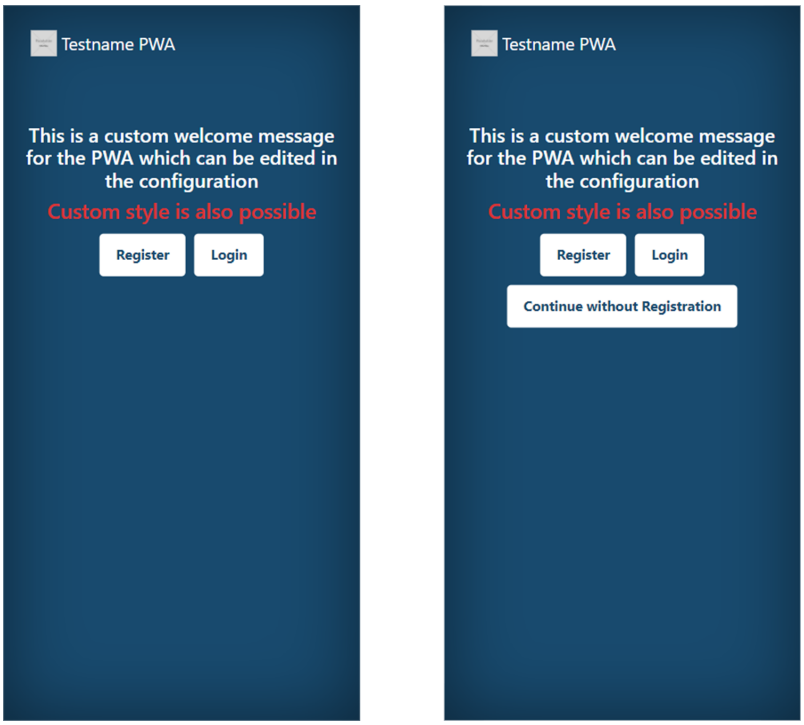

Figure 6.6: The startpage of the PWA

Figure 6.6 shows the welcome screen when the end user visits the PWA for the first time with a required registration on the left, and an optional registration on the

[9] https://getbootstrap.com/
[10] https://jquery.com/

right, where the end user can skip the registration by pressing the “Continue without Registration“ button. The text above the buttons shows the welcome message that the expert can define.

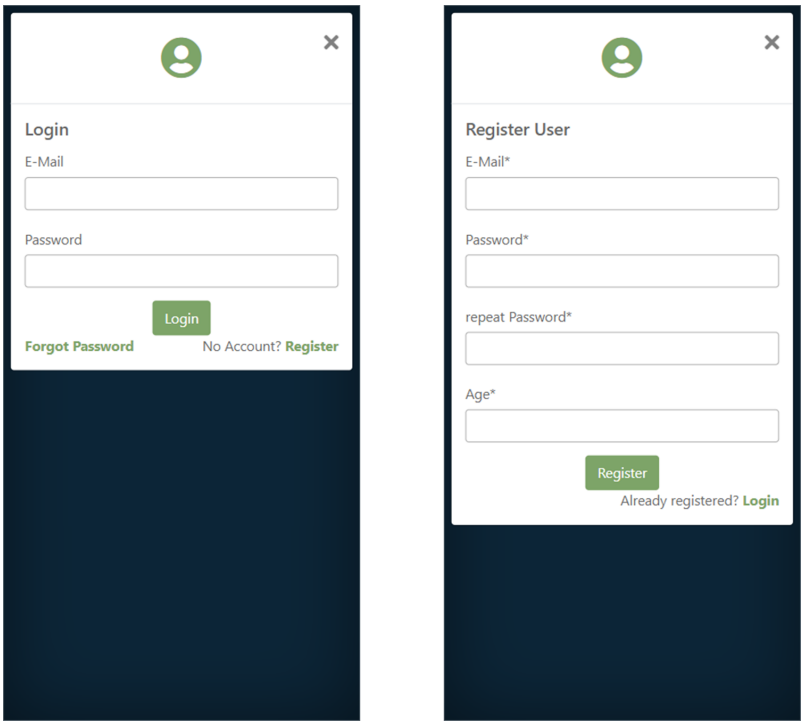

Figure 6.7: The login and registration view of the PWA

Figure 6.7 (left) shows the login form, where the end user can enter his email address and password to authenticate himself. The screenshot on the right side of the Figure shows the registration form where the end user has to enter his email address and password that he wants to use for the PWA. In the “repeat Password“ input, the end user has to enter the password a second time to prevent typing errors. Additionally, a required input field (marked with an asterisk) called “Age“ has to be filled out by the end user before he can complete the registration.

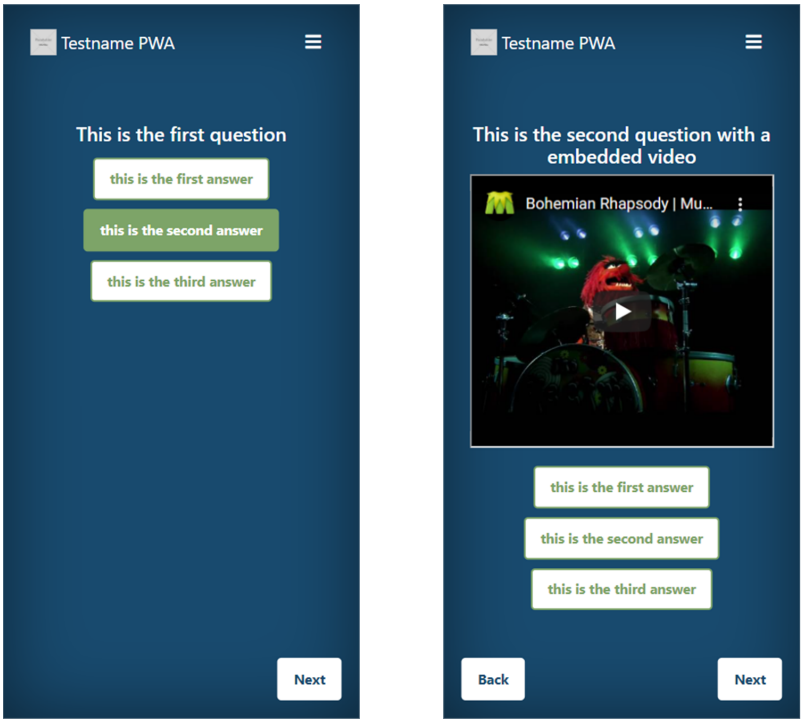

Figure 6.8: The question view with different multiple choice questions

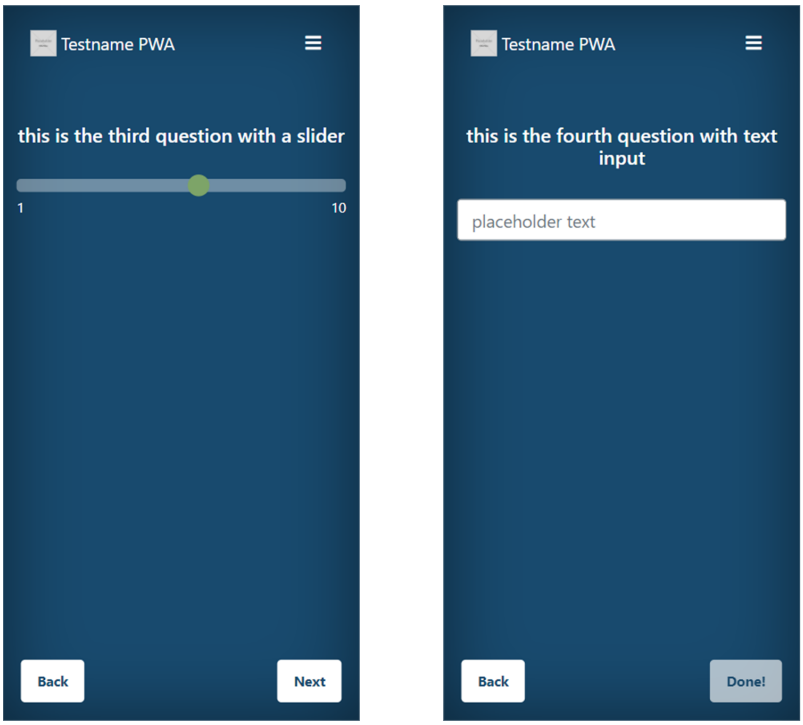

Figure 6.9: The question view with range question and text input question

Figure 6.8 (left) shows a multiple choice question with the question text and three answer options with the second answer currently selected. The possibility to use custom HTML markup in the question text lets the expert embed images or videos as shown in Figure 6.8 (right). The range question type shown in
Figure 6.9 (left) lets the end user select a value between one and ten and the text input question shown in Figure 6.9 (right) lets the end user enter a custom answer in the provided input field.

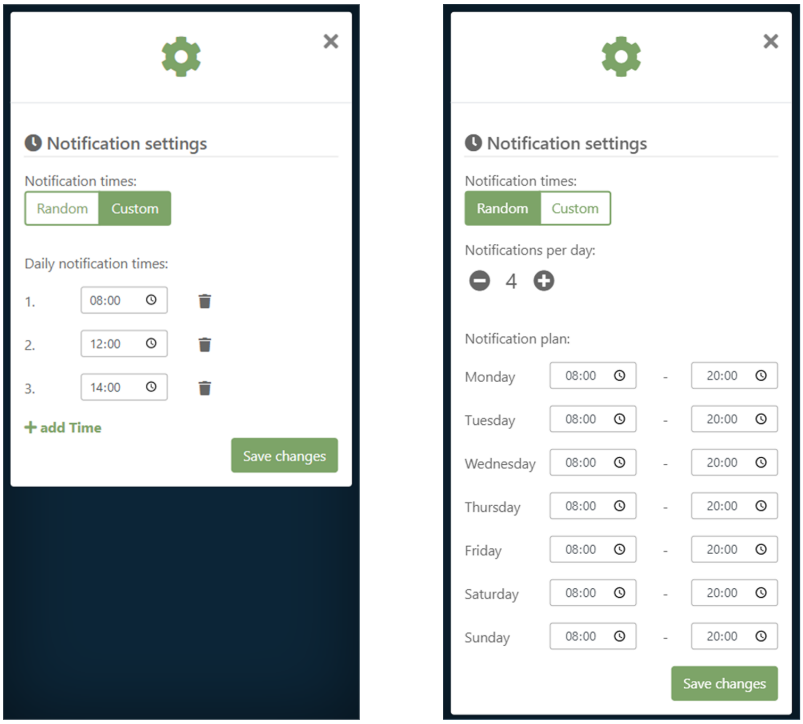

Figure 6.10: The settings view of the PWA

Figure 6.10 shows the settings view where the end user can choose between the notification modes “Custom“ and “Random“. The screenshot on the left shows the custom notification time list and the one on the right shows the settings menu for the random notification times with the number selector and the weekly schedule below.

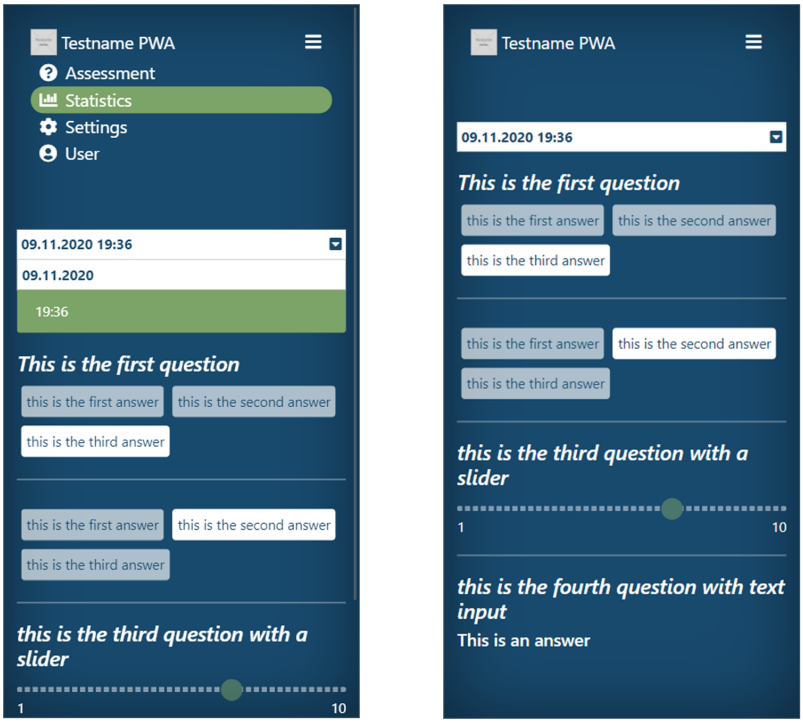

Figure 6.11: The statistics view of the PWA

The statistics view in Figure 6.11 shows a completed assessment with all questions and answers given by the end user. The screenshot on the left additionally shows the extended navigation menu and the extended assessment selector where the end user selects an assessment to display.

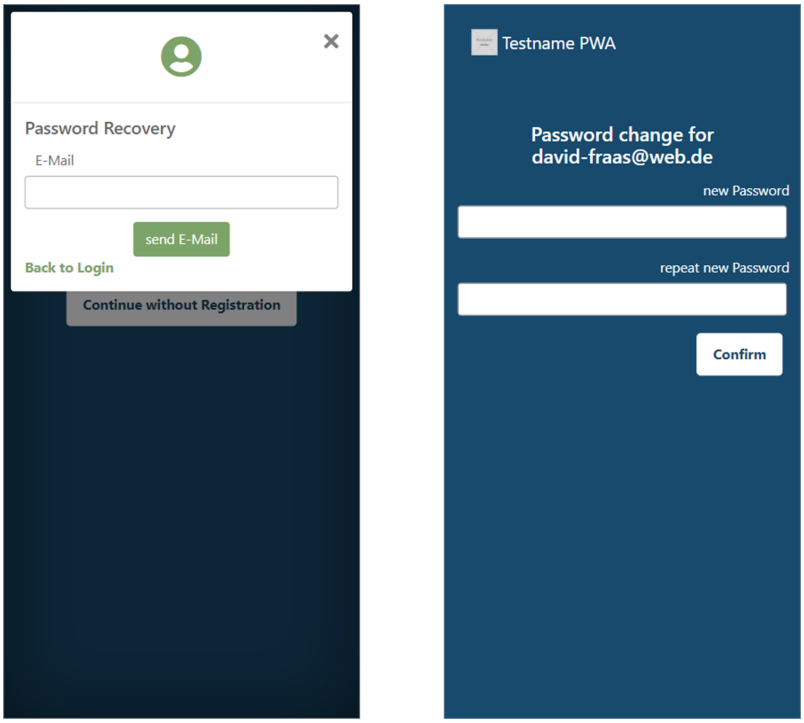

Figure 6.12: The password reset page of the PWA

Figure 6.12 (left) shows the password recovery form where the end user can send a request to reset the password of his account by providing his email address. Figure 6.12 (right) shows the reset page that the user can access by opening the link that is sent to his email address after he requested a password reset.

6.8.2 Backend

In this thesis, the backend refers to the components on the server that handle the requests from the end user and store the data persistently. This includes the web server and the database. In some functions, the frontend directly communicates with the database without interference of the web server (e.g., authentication). The web server is implemented as an express.js[11] node application and is responsible for the following tasks:

- serve website data (e.g., HTML, CSS and JavaScript files)

[11] https://expressjs.com/

- validate user and assessment data
- generate password reset tokens and send them via emails over the SMTP server

The database is implemented with the document-based database distribution CouchDB[12]. With the combination of CouchDB and the frontend JavaScript library pouchDB, the database on the server can be synchronized with the database on the end users device (see Section 6.2.2). The database consists of an account database where all user documents are stored and an additional per-user database for every account in order to store assessment data separately for every end user. The database is responsible for the following tasks:

- provide assessment data
- store user and assessment data
- create user databases
- authentication (session handling)

Figure 6.13 shows the process when the end user completes an assessment and wants to store it on the database of the server. The PWA sends a request to the web server with the assessment and the authentication token. Even though the frontend validates the assessment data, it will be validated again on the web server. This step is necessary to prevent faulty assessment data that can be injected over the REST interface of the web server. If the assessment data is correct, the web server sends a request to the database to store the assessment. The database validates the authentication token before storing the assessment and returning a success message to the web server, which will then send a success message to the PWA.
The authentication for the PWA is handled by the database. Figure 6.14 shows the sequence of the login process where the end user enters his username and password that are sent to the database. The database validates the login data and if the login data is correct, the database generates an authentication token and returns it to the PWA. The PWA then automatically sends another request with the authentication token to get the assessment data of the end user from the database.

[12] https://couchdb.apache.org/

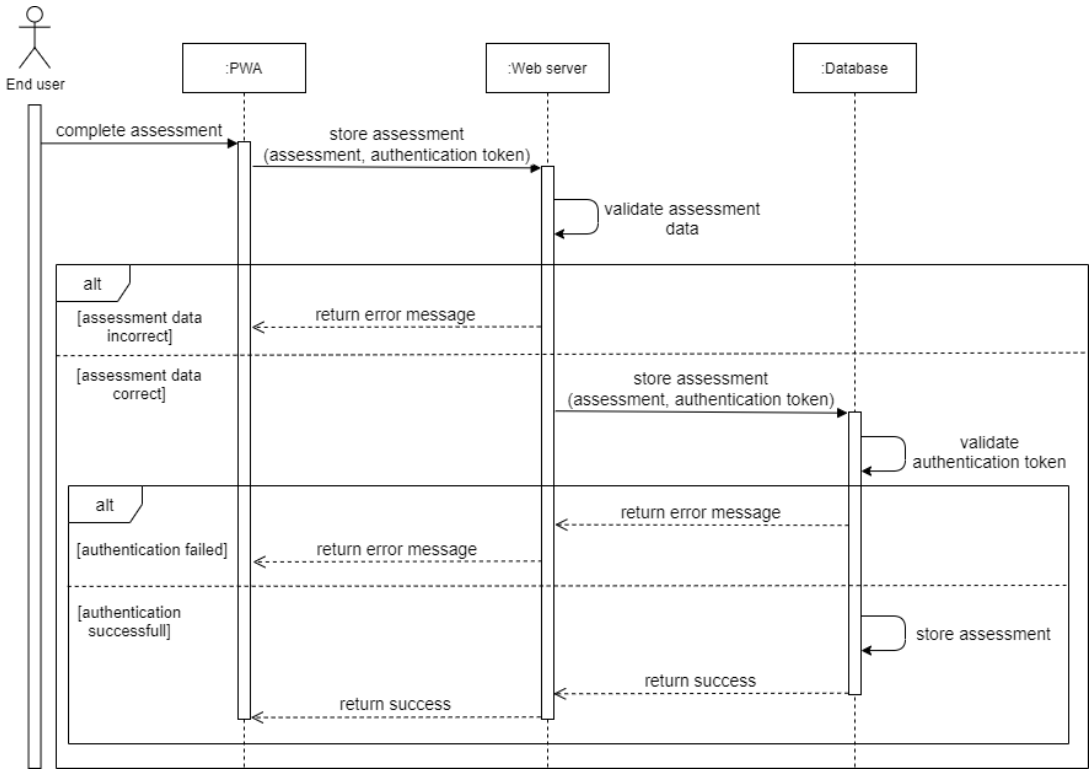

Figure 6.13: Sequence diagram of the assessment storing process

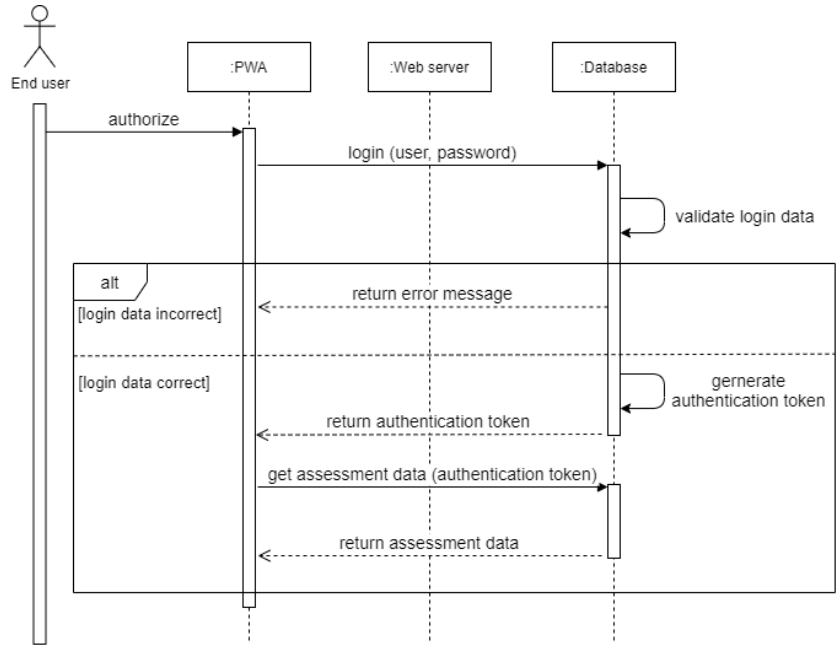

Figure 6.14: Sequence diagram of the login process

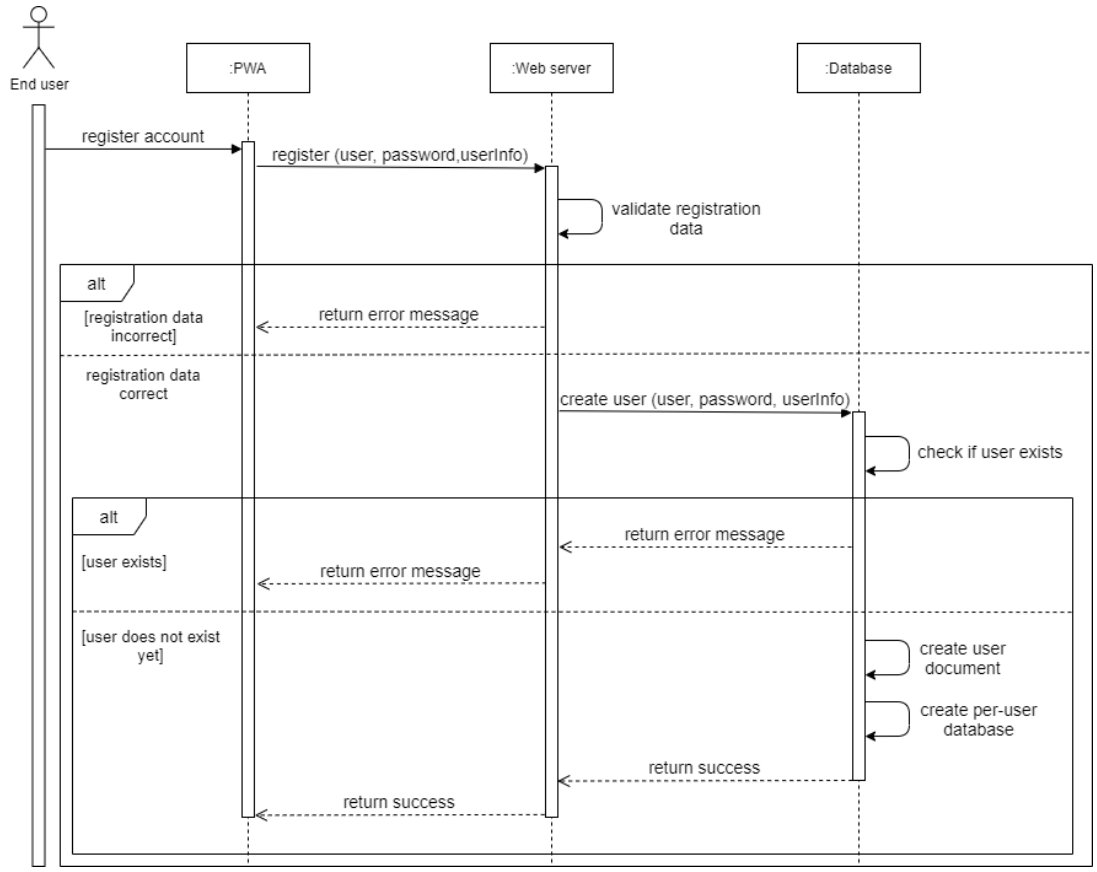

Figure 6.15: Sequence diagram of the registration process

Figure 6.15 shows the registration process. When the end user wants to create a new account, the PWA sends the registration data including email, password and additional user information to the web server. The web server then validates the registration data before forwarding it to the database. The database then checks if the user already exists. If this is not the case, the database creates a new user document and a per-user database where the assessment data will be stored before returning a success response to the web server, which is then again forwarded to the PWA.

Figure 6.16 shows the process when the end user wants to reset his password. This process consists of two parts, in the first part the email address of the account is send to the web server, which in return sends a request to the database to check if an account with that email address is registered. If this is the case, the web server generates a reset token and a timestamp when the token was created. Both token

and timestamp are stored in the database. Afterwards, the reset token is sent to the email address that was provided by the end user.
In the second part of the process, a request by the user to change the password is sent from the PWA to the web server. This request includes the new password and the reset token. The web server checks if the new password meets all password requirements. If the new password is valid, the web server requests the timestamp that is associated with the token from the database. If the token is valid (meaning the timestamp is not older than ten minutes) the password will be updated in the user document of the database and the timestamp of the token will be reset, so the password can not be changed a second time with the same token.

Figure 6.17 shows the process when the end user wants to delete his account. The PWA directly sends a request to delete the account to the database. The database then deletes the user related database before deleting the user data in the users document. Afterwards the database sends a success response to the PWA, which then in return deletes all user cookies and the local database.

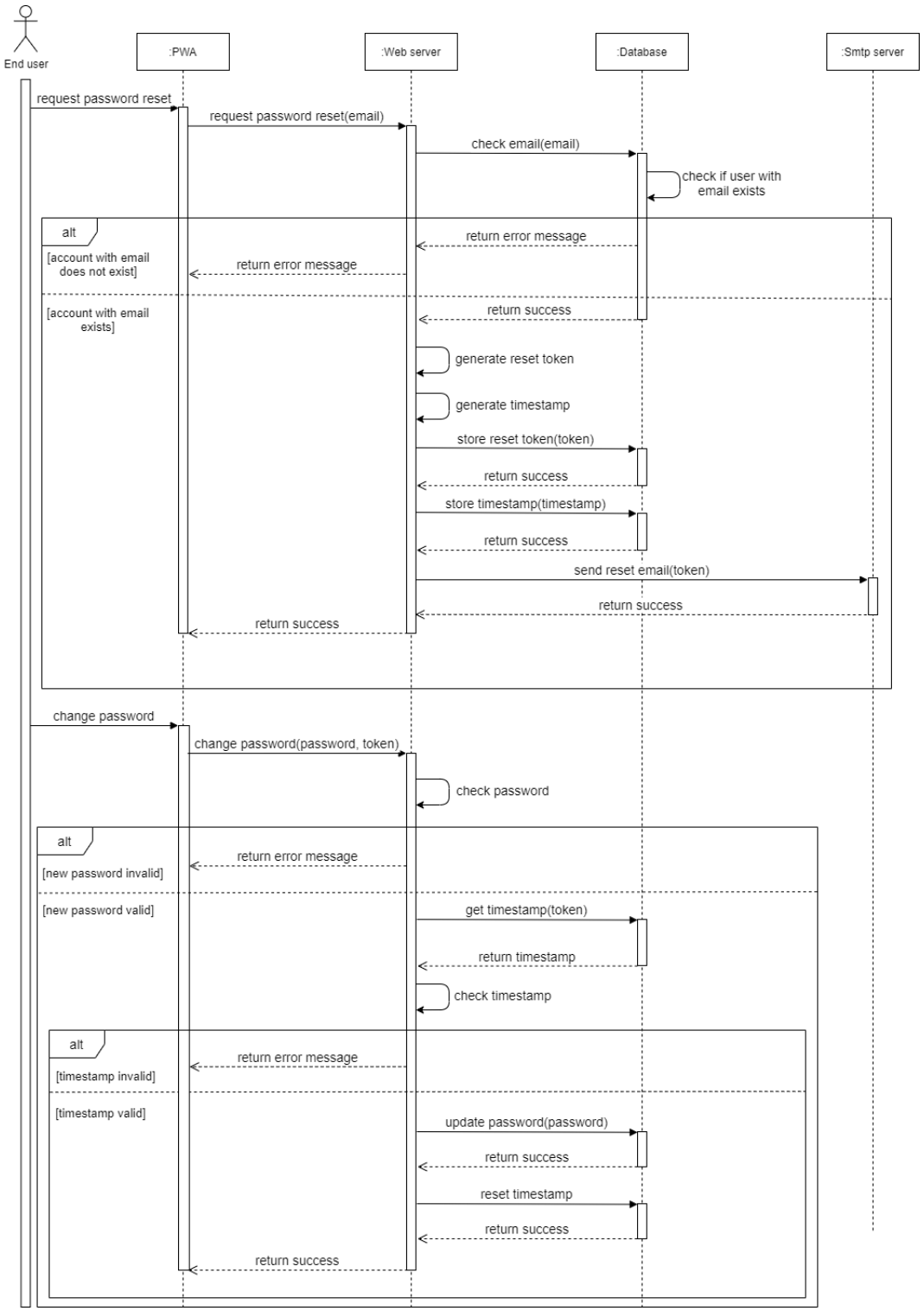

Figure 6.16: Sequence diagram of the password reset process

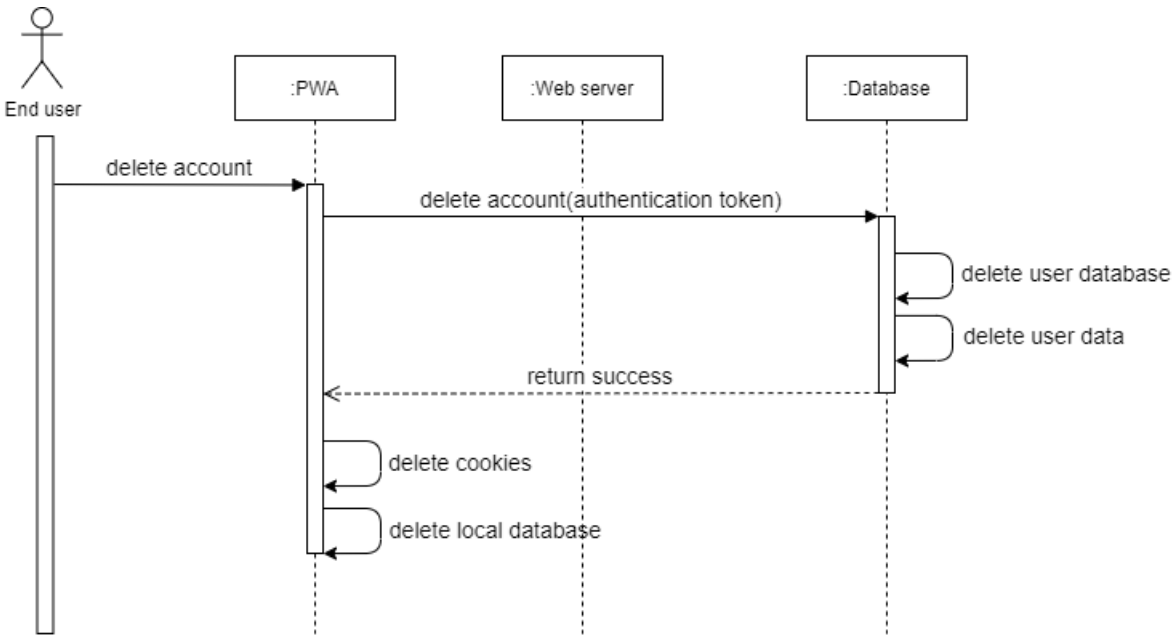

Figure 6.17: Sequence diagram of the account deletion process

7 Compliance with Requirements

In this chapter, the compliance of the framework requirements, PWA requirements and non-functional requirements with the implementation is checked. The measurement of the compliance for the framework requirements and PWA requirements depend on the implementation only, while the non-functional requirements also take the architecture into consideration. The following five levels are used to measure the compliance:
(5) The requirement was fully met.
(4) The requirement was met sufficiently.
(3) The requirement was met partially.
(2) The requirement was met insufficiently.
(1) The requirement was not met.

7.1 Framework requirements

The table below shows level of compliance for the framework requirements that have been defined in Section 4.1.

Code	Function	Priority	Fulfilled
Configuration			
F01	Configure assessment	MUST	5
F02	Configure database	MUST	5
F03	Configure registration	MUST	5
F04	Configure appearance	COULD	5
F05	Configure server	MUST	5
F06	Configure SMTP	SHOULD	4
F07	Configure sensordata	MUST	4
Deployment			
F08	Build PWA	MUST	5
F09	Run PWA	MUST	4

7.2 PWA requirements

The table below shows level of compliance for the PWA requirements that have been defined in Section 4.2.The function to reset the password of the end user (P03: Reset password) and the function to delete the account (P04: Delete account) were met partially or not met because of time reasons.

Code	Function	Priority	Fulfilled
P01	Create account	SHOULD	4
P02	Authorization	SHOULD	5
P03	Reset password	COULD	3
P04	Delete account	COULD	1
P05	Edit notification schedule	MUST	4
P06	Send notification	MUST	4
P07	Start assessment	MUST	5
P08	Do assessment	MUST	5
P09	Complete assessment	MUST	5

7.3 Non-functional requirements

The table below shows level of compliance for the PWA requirements that have been defined in Section 4.3. The robustness (NF4) was met insufficiently, because the implementation process did not include software tests, therefore, the ability to cope with errors and erroneous input can not be guaranteed.

Code	**Requirement**	**Priority**	**Fulfilled**
NF1	Reliability	MUST	4
NF2	Responsive design	SHOULD	4
NF3	Robustness	SHOULD	2
NF4	User friendliness	SHOULD	4
NF5	Documentation	COULD	5

8 Summary

This chapter consists of the conclusion which sums up the thesis, and the future work.

8.1 Conclusion

In this thesis, a framework for developing and deploying EMA apps was developed. The goal was to give an expert the possibility to create his own EMA application as Progressive Web Application to collect data from a certain group of users with additional environmental data that is collected by the sensors on the end users device (Mobile Crowdsensing). Furthermore, the expert is able to use the framework without any programming knowledge, and the framework does most of the work in building and running the PWA, so the expert can focus on configuration and does not have to worry about how to make the EMA application accessible for the end users.
The framework takes the configuration of the expert as input and builds software containers. These software containers include the reverse proxy (implemented with traefik) that is used for secure connection via HTTPS, which is required by some frontend APIs that are used in the PWA and creating subdomains for the other containers to be reachable. The second container is the web server, which provides the PWA data for the end user. The third and fourth containers which include the database and the SMTP server are optional, because the expert can decide to use his own database or SMTP server. The database container consists of a CouchDB instance, which stores the incoming data as JSON documents.
For the PWA that is used by the end user, a user interface with focus on mobile devices was developed. The framework uses template files that are completed with

the configuration data from the expert to generate the PWA which is used to answer assessment questions, record sensor data and remind the user to complete assessments. The PWA uses service workers and PouchDB to enable offline usage. Additionally, it uses APIs to collect sensor data and send notifications to the end user. The sensor data can be the geolocation of the user, the noise around the user (recorded with the microphone of the device), the brightness in the users environment, or any data that is coming from a bluetooth device which can be connected to the PWA. The end user can decide whether he wants to get notified at random times or at fixed times by the PWA to complete an assessment.
All in all, the developed framework is suitable to generate a PWA to collect data from end users.

8.2 Future work

The focus of the PWA that is generated by the framework was to collect data from an end user with the requirement that the end user actually completes assessments frequently. This is not always the case, since the end user is not always motivated to complete an assessment. Therefore the future work focuses on incentive mechanisms that keep the end user motivated. As stated by Kraft et al. [3], these incentive mechanisms can be feedback, gamification or social features. The PWA already provides feedback to the end user by displaying the history of all completed assessments in the PWA (see Figure 6.11), however, this can be expanded with graphs and charts for a better representation of the collected data, or even chat bots that give instant feedback by analysing the answers of the end user. For gamification, a reward system with achievements for completing a total amount of assessments, or completing a streak of assessments for a certain number of days can be implemented. Another feature for gamification is a leader board, which displays the end users with the most completed assessments. A third incentive mechanism is the implementation of social features, for example, user profiles and sharing functions. Additionally, the registration process can be completed with social logins, for example with the help of OpenID[1].
Another topic that was mentioned by Kraft et al. [3] is the data analysis by the ex-

[1] https://openid.net

pert. This could include a web application for the expert, that displays the collected data from the database with charts and diagrams to compare the data from different end users. Additionally, the expert should be able to export the collected data in common formats like CSV or PDF.

Bibliography

[1] Arthur A. Stone, Joseph E. Schwartz, John M. Neale, Saul Shiffman, Christine A. Marco, Mary Hickcox, Jean Paty, Laura S. Porter, and Laura J. Cruise. “A comparison of coping assessed by Ecological Momentary Assessment and retrospective recall.” In: *Journal of Personality and Social Psychology* 74.6 (1998), pp. 1670–1680. ISSN: 0022-3514. DOI: 10.1037//0022-3514.74.6.1670. URL: https://pubmed.ncbi.nlm.nih.gov/9654765/.

[2] Rüdiger Pryss, Thomas Probst, · Winfried Schlee, Johannes Schobel, · Berthold Langguth, Patrick Neff, Myra Spiliopoulou, Manfred Reichert, Winfried Schlee, and Berthold Langguth. “Prospective crowdsensing versus retrospective ratings of tinnitus variability and tinnitus-stress associations based on the TrackYourTinnitus mobile platform”. In: *International Journal of Data Science and Analytics* 8 (2019), pp. 327–338. DOI: 10.1007/s41060-018-0111-4. URL: https://doi.org/10.1007/s41060-018-0111-4.

[3] Robin Kraft, Winfried Schlee, Michael Stach, Manfred Reichert, Berthold Langguth, Harald Baumeister, Thomas Probst, Ronny Hannemann, and Rüdiger Pryss. “Combining Mobile Crowdsensing and Ecological Momentary Assessments in the Healthcare Domain”. In: *Frontiers in Neuroscience* 14 (Feb. 2020), p. 164. URL: http://dbis.eprints.uni-ulm.de/1879/.

[4] Saul Shiffman, Arthur A Stone, and Michael R Hufford. “Ecological Momentary Assessment”. In: *Annu. Rev. Clin. Psychol.* 4 (2008), pp. 1–32.

[5] Huadong Ma, Dong Zhao, and Peiyan Yuan. “Opportunities in Mobile Crowdsensing”. In: *IEEE Communications Magazine* 52.8 (2014), pp. 29–35.

[6] Andreas Biørn-Hansen, Tim A. Majchrzak, and Tor-Morten Grønli . “Progressive Web Apps: The Possible Web-native Unifier for Mobile Development”. In: Jan. 2017, pp. 344–351. DOI: 10.5220/0006353703440351.

[7] Michael Stach, Carsten Vogel, Thorsten-Christian Gablonski, Sylke Andreas, Thomas Probst, Manfred Reichert, Marc Schickler, and Rüdiger Pryss. *Technical Challenges of a Mobile Application Supporting Intersession Processes in Psychotherapy*. Tech. rep. URL: https://www.sciencedirect.com/science/article/pii/S187705092031718X.

[8] DBIS Uni Ulm. *Intersession-Online*. visited on 2020-11-30. URL: https://play.google.com/store/apps/details?id=de.intersession_online.intersession_online.

[9] Thomas Probst, Rüdiger Pryss, Berthold Langguth, and Winfried Schlee. "Emotional states as mediators between tinnitus loudness and tinnitus distress in daily life: Results from the TrackYourTinnitus application". In: *Scientific Reports* 6 (Feb. 2016). URL: http://dbis.eprints.uni-ulm.de/1396/.

[10] Rüdiger Pryss, Winfried Schlee, Berthold Langguth, and Manfred Reichert. "Mobile Crowdsensing Services for Tinnitus Assessment and Patient Feedback". In: *6th IEEE International Conference on AI & Mobile Services (IEEE AIMS 2017)*. IEEE Computer Society Press, June 2017, pp. 22–29. URL: http://dbis.eprints.uni-ulm.de/1521/.

[11] Thomas Probst, Rüdiger Pryss, Berthold Langguth, and Winfried Schlee. "Emotion dynamics and tinnitus: Daily life data from the "TrackYourTinnitus" application". In: *Scientific Reports* 6.31166 (2016). URL: http://dbis.eprints.uni-ulm.de/1718/.

[12] Robin Kraft, Michael Stach, Manfred Reichert, Winfried Schlee, Thomas Probst, Berthold Langguth, Marc Schickler, Harald Baumeister, and Rüdiger Pryss. "Comprehensive insights into the TrackYourTinnitus database". In: *17th International Conference on Mobile Systems and Pervasive Computing (MobiSPC)*. Procedia Computer Science 175. Elsevier, May 2020, pp. 28–35. URL: http://dbis.eprints.uni-ulm.de/1955/.

[13] TRI Initiative in cooperation with UKW Würzburg. *Track Your Tinnitus*. visited on 2020-11-30. URL: https://play.google.com/store/apps/details?id=com.jochenherrmann.trackyourtinnitus.

[14] Saul Shiffman. “Designing Protocols for Ecological Momentary Assessment”. In: *The science of real-time data capture: Self-reports in health research* (2007), pp. 27–53.

[15] Kevin Brennan et al. *A Guide to the Business Analysis Body of Knowledger*. Iiba, 2009.

[16] Statcounter.com. *Mobile Browser Market Share Europe StatCounter Global Stats*. visited on 2020-11-29. URL: https://gs.statcounter.com/browser-market-share/mobile/europe/#monthly-201910-202010-bar.

[17] Jake Archibald. *Is service worker ready?* visited on 2020-11-28. URL: https://jakearchibald.github.io/isserviceworkerready/.

[18] caniuse.com. *Can I use notification?* visited on 2020-11-28. URL: https://caniuse.com/?search=notification.

[19] caniuse.com. *Can I use IndexedDB?* visited on 2020-11-28. URL: https://caniuse.com/indexeddb.

[20] caniuse.com. *Can I use Web Bluetooth?* visited on 2020-11-28. URL: https://caniuse.com/web-bluetooth.

[21] caniuse.com. *Can I use Geolocation?* visited on 2020-11-28. URL: https://caniuse.com/geolocation.

[22] developer.mozilla.org. *Sensor APIs - Web APIs | MDN*. visited on 2020-11-28. URL: https://developer.mozilla.org/en-US/docs/Web/API/Sensor_APIs.

[23] developer.mozilla.org. *MediaStream Recording API - Web APIs | MDN*. visited on 2020-11-28. URL: https://developer.mozilla.org/en-US/docs/Web/API/MediaStream_Recording_API.

A Configuration Example

```
"assessment": {
    "questions":[
      {
        "type": "MULTIPLE_CHOICE",
        "title": "This is the first question",
        "minanswers": 1,
        "maxanswers":1,
        "answers":[
          {
            "text": "this is the first answer",
            "nextQuestion": 2
          },
          {
            "text": "this is the second answer"
          },
          {
            "text": "this is the third answer"
          }
        ]
      },
      {
        "type": 1,
        "title": "<h3>This is the second question with an embedded
          video</h3><iframe width=\"420\"  height=\"315\"
          src=\"http://url.to.video/">
          </iframe>",
        "minanswers": 1,
        "maxanswers":1,
        "answers":[
          {
            "text": "this is the first answer",
            "nextQuestion": 2
          },
          {
            "text": "this is the second answer"
          },
          {
            "text": "this is the third answer"
          }
        ]
      },
      {
```

```
        "type": 2,
        "title": "this is the third question with a slider",
        "min":1,
        "max":10
      },
      {
        "type": 3,
        "required":true,
        "title": "this is the fourth question with text input",
        "placeholder":"placeholder text"
      }
    ]
},
"database":{
    "type":"auto"
},
"registration": {
    "required": false,
    "userInfo":[
        {
          "label": "Age",
          "required": true,
          "type":"number"
        }
    ]
},
"appearance":{
    "name": "Testname PWA",
    "mainColor":"#184a6e",
    "secondaryColor" : "#7da468",
    "welcomeMessage": "<h3>This is a custom welcome message for the PWA which
      can be edited in the configuration</h3><h4 style=\"color:#d33\">Custom
      style is also possible</h4>",
    "notificationMessage": "Notification message to prompt the user for
      his assessment"
},
"smtp":{
    "type":"auto"
},
{
"server":{
    "domain":"pwatest.de",
    "ip":"123.456.789",
    "certificateEmailAddress":"test@email.de"
},
"sensordata":{[
        {
        "sensor":"microphone",
        "maxLength":"0:10"
        }
    ]
}
```

B Abbreviations

CLI Command Line Interface

EMA Ecological Momentary Assessment

GATT Generic Attribute Profile

JSON JavaScript Object Notation

MCS Mobile Crowdsensing

PWA Progressive Web Application

Name: David Fraas Matrikelnummer: 1008637

Erklärung

Ich erkläre, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den 08.12.2020 D. Fraas

David Fraas