Universität Ulm | 89069 Ulm | Germany

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

Conceptualization and Realization of a Generic Mobile App Framework to support Interventional and Sensor-driven as well as Mobile Crowdsensing based Clinical studies

Master's thesis at Universität Ulm

Submitted by:
Robin Bird
robin.bird@uni-ulm.de

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

Supervisor:
Dr. Rüdiger Pryss

2019

Version from August 12, 2019

Composition: PDF-$\LaTeX\,2_\varepsilon$

Abstract

Creating medical questionnaires, scientific studies and carry them to execution is very labor- and cost intensive. New trends like the digitization of business processes and crowd-sensed data collection support professionals and scientists in their research and allow up-scaling of studies to reach more patients and gain more precise scientific findings. The use of mobile apps to support scientific studies allows the integration of vital sensors that improve the quality of scientific studies. One issue remains though, the creation of a mobile app for a study may still be very cost and time consuming. The QuestionSys framework is a framework developed in the context of the research by the Institute of Databases and Information Systems of Ulm University, that allows researchers as well as clinicians to develop their own mobile apps and allows them to collect mobile data without programming skills. This project aims to assist scientist as well as professionals in medical domains to improve patient healthcare and medical research.

Therefore in the context of this thesis, a mobile framework is created that implements a process engine that can read and execute generic questionnaires created by the QuestionSys Configurator. The main goal is to reduce the costs, time effort and the qualification barrier to design and create new scientific studies. For the framework, the support for internal as well as external bluetooth sensors is integrated and custom questionnaire elements of the QuestionSys Configurator are used to configure the sensors. The framework is developed to easily allow extending functionality for new sensors and allow adding custom functionality that may be needed in the context of new studies. To prove the practicability, a case study "Mindful Walking" is implemented using the developed framework. The study is implemented in the form of a multi-session study. It supports sensor logging for a bluetooth heart rate sensor, a GPS speed sensor and a GPS distance sensor and allows the export of all sensor logs. In addition, a demo application is developed to prepare the framework for integration of a future RESTful Api that may load studies and upload study results. The application is designed to cache

media files and to execute studies offline. This allows to reach parts of the world with limited access to the Internet.

Acknowledgment

I would like to express my gratitude to everyone who supported me during this master's thesis.

I would first like to thank my thesis adviser Dr. Rüdiger Pryss for his invaluable support, commitment and assistance during my master's thesis.

I would also like to thank Dr. Johannes Schobel very much for supporting me during my master's thesis and assisting me with the usage of the QuestionSys backend.

Furthermore I acknowledge the contribution of my proofreaders for their help to improve the writing and wording of this master's thesis.

Contents

1 Introduction

These days, there are many new IT technologies that can be used in medical domains to either improve patient healthcare or improve medical research.

One of these trends is the use of vital sensors in mobile healthcare applications. There are multiple scientific papers that cover the implementation of a sensor framework for mobile applications and the connection of bluetooth enabled sensors with mobile phones in order to measure the patient's health data.

Another trend is the digitization of business processes. Digitization can also be used to improve medical research and clinical studies. The development of multiple apps which support crowed sensed data collection enables clinical studies to scale up and reach parts of the world that are usually hard to access. One of these clinical studies is the *Mindful Walking* application, which studies the effects on "Mindfulness" on stress.

1.1 Problem statement

The described new technologies may improve the quality of clinical studies and the speed of creating studies significantly. The use of smart phones for the execution of clinical studies also enables the chance of integrating vital sensors into clinical studies. Therefore feedback and statistics can be given to the user instantly and patients may participate in studies from their home or data may be collected over a long period of time.

However, none of these solutions address remaining massive costs and effort in creating a mobile application for new clinical studies. If a generic framework like QuestionSys that is described in section 2.1 is used to create and deliver questionnaires to the user's phone, there is still no way to make the App generic in order to integrate vital sensor data collection and other functional additions to questionnaire pages.

The basic idea of this master's thesis is to combine all these technologies and functionality that is needed for studies and to create a concept for a generic mobile app framework to support interventional clinical studies. In this framework the functionality for sensor-driven data collection is included and it enables crowd-sensed data collection as well.

1.2 Objective

In the context of the research of the Institute of Databases and Information Systems, it is the objective of this work to design and implement the framework which is described above. As the study input, questionnaires should be created by making use of the *QuestionSys Configurator*. The *QuestionSys Configurator* was designed to create psychological questionnaires, so additional changes have to be made in the questionnaires to use the output of the *QuestionSys Configurator* for generic studies. Therefore the Custom Elements provided by the *QuestionSys Configurator* will be used to add action views, sensor configuration and session-aware meta data to the framework. The framework will be implemented for Android and should be able to process the output provided by the QuestionSys Configurator when exporting studies.

The framework should also have an export functionality to export study results to a JSON formatted file that may be uploaded using a RESTful Api.

To prove the practicability of the theoretical approach of this master's thesis, an android application that embeds the mentioned framework is implemented. Besides, the developed application is also used to execute the Mindful Walking study and to discuss and present the challenges of that study.

1.3 Structure of the thesis

This work is structured into nine chapters. The second chapter 2 introduces the *QuestionSys Configurator* as well as other related work to improve the reader's understanding about this master's thesis. Besides, the use cases for the framework in this work are discussed. The third chapter 3 defines the requirements for the framework of this work and the fourth chapter 4 discusses the architectural aspects of the framework. First, the overall application is introduced, then the class structure is outlined. The data model and persistence as well as the input data structure are discussed. The end of chapter four contains a discussion about the results structure. In the fifth chapter 5 selected implementation aspects like layout aspects, the view adapter used for providing the user interface, the persistence manager and also saving of meta data are shown. Finally the sensor integration of bluetooth sensors as well as internal sensors is discussed. It is also shown how sensor data is logged and persisted. Chapter six 6 introduces the case study *Mindful Walking* with a process model. Then the process model is transferred into a technical model in the *QuestionSys Configurator*. The challenges when transferring the process model of the case study into the technical model are discussed. In the seventh chapter 7 the implemented mobile application is presented by screen-shots and in the eighth chapter 8 the previously defined requirements are checked. Finally the ninth chapter 9 is a conclusion and shows the prospects of the study.

2 Fundamentals

This chapter introduces basic fundamentals which are required for this work. First, the *QuestionSys Configurator* that is used as the platform to create clinical studies is introduced. Afterward the related work is described in a more accurate way.

2.1 QuestionSys

The *QuestionSys* framework is a framework that allows researchers as well as clinicians to develop their own mobile apps to collect mobile data without programming skills [17]. Psychological and social sciences in various situations rely on self-report questionnaires to collect data. Some researches are unaware of the capabilities and opportunities offered by smart mobile devices in their respective domain [17] and already existing data collection apps do not adequately support researchers [17]. In addition, implementing sophisticated mobile data collection apps usually requires considerable communication efforts between researchers and mobile app developers [17]. Therefor a model-driven, end-user programming approach was developed to create psychological or clinical studies. *QuestionSys* aims to provide a generic and flexible questionnaire system to enable process-driven mobile data collection. It encompasses three parts:

1. a questionnaire configurator for defining questions as well as for aggregating them to pages within the questionnaire. The latter will then be displayed on the smart mobile device.
2. a questionnaire application enabling the use of smart mobile devices to gather data,

3. a middleware for secure data exchange and data management (including cloud-based services)

[28]

In the context of this master's thesis the *QuestionSys Configurator* is used to create questionnaires. To create fully-fledged clinical studies, some extension have been made using the given functionality by the *QuestionSys Configurator*. The generated output of the *QuestionSys Configurator* is used as input for the *QuestionSys studies* framework that is developed in context of this thesis.

The Configurator was developed as a web application. It provides a graphical user interface to create process models that represent questionnaires. Therefor the web application is divided into three parts. The left part of the application provides the user multiple elements which can be used to build a process model. Nodes are the basic elements in a technical model. Nodes can be structural elements or elements with a user interface that is presented to the participant of the study. The following structure elements are available:

1. **Pages** can be used to split UI elements of the study into different pages. These are illustrated as grey boxes that contain nodes with UI elements. Every page in the technical model is equivalent to a single page of the questionnaire which can be displayed to the user in a mobile application.
2. **XOR Splits** split the model in different lanes by specific conditions. These are illustrated as inclined squares.
3. **XOR Joins** join the previously split lanes together.

Since all elements that are attached to an UI element belong to a specific page in the questionnaire, these elements are called page elements and have to be dropped within a page. The following page elements are available:

1. **Headlines** are the headlines of a page and illustrated as light blue rectangles.
2. **Questions** are the question elements of a page and illustrated as pink rectangles.

3. **Text elements** are used to show formatted text on a page and illustrated as orange rectangles.
4. **Media elements** are used to show media objects on a page and illustrated as yellow rectangles. Media objects may be images, audio files or videos.
5. **Custom elements** are used to define custom JSON content. In the context of this master's thesis these are used to define additional logic or page elements.

The middle part of the application contains the graphical representation of the process model that is created using the structure elements and the page elements. To adapt the process model every element can be added or moved by drag and drop gestures.

The right part of the *QuestionSys Configurator* contains the details of each element. When an element is selected, the details of that element can be configured. For a question element that might be the question type and the answer possibilities. For a XOR split that might be the branches and the conditions following the split.

In the context of this master's thesis the *QuestionSys* back-end and the *QuestionSys Configurator* have been used to create questionnaires and to export these questionnaires to a JSON formatted text file that can be used as the input for the *QuestionSys studies* framework.

2.2 Related Work

This section presents related work to improve the readers' understanding about the precise result of this master's thesis. Therefore, clinical study apps as well as sensor-enabled mobile applications are discussed.

2.2.1 QuestionSys mobile application

In the scope of the bachelor's thesis *Developing a Complex User Interface for Mobile Data Collection Applications* of Martin, Robin a sophisticated user interface for mobile applications to collect mobile data has been developed [3]. The main target was to transfer paper based questionnaires that have several drawbacks like costs, data quality and logistical issues into an application that is able to show and collect data in large-scale scenarios. Therefore, specific rules, user interface guidelines for mobile devices, the style and usability guidelines of the two platforms Android and iOS, and aspects like visual design, interaction design and navigation in mobile applications were taken into consideration for developing an UI that can present digitized questionnaires. Figure 2.1 shows the basic structure of a questionnaire page with the corresponding UI developed in the context of the bachelor's thesis.

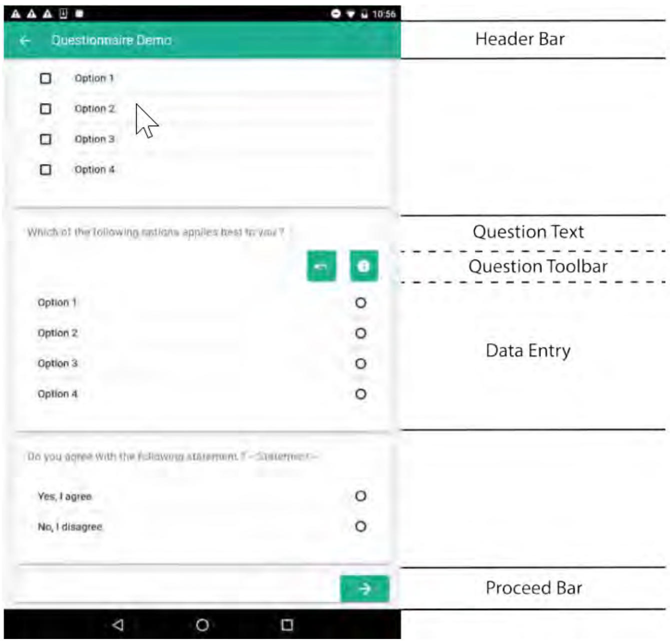


Figure 2.1: Basic Structure of a Questionnaire Page in the Questionnaire Interface application [3]

2.2.2 Clinical study apps

In the scope of the bachelor's thesis *Conception and implementation of a mobile application to conduct a Mindful Walking Study regarding Clinical Psychology* of Müller, Dominik a mobile application was developed that may be used as a supporting tool when walking mindfully [5]. Therefore the support for the Apple Watch was integrated to use the combined data of an iPhone and an Apple Watch. Also the app allows users to participate in a study of Mindful Walking that researches whether Mindful Walking might be helpful to people in their daily lifes [5].

For the application several requirements have been defined. One requirement of the application for example was to show and select all questionnaires that belong to an active study the user participates in. The user should also be able to complete and correct questionnaires. Notifications were used to remind the participant to complete the questionnaire.
The Apple Watch works as a companion for the mobile application and should be able to start and stop a Mindful Walk. Also the user should be able to select a target reduction speed, calibrate the walking speed and show Mindful Walking statistics on the Apple Watch.

For this functionality a user interface for the iPhone and the Apple Watch has been created that guides the user through the Mindful Walk study. For the questionnaires that belong to the study a question view has been created in which the user can answer the questions that belong to the questionnaire. For the Apple Watch several views have been created that guide the user through the Mindful Walk. After calibrating the Apple Watch, the user has to walk for at least five minutes. Therefor a countdown and an instruction text is shown. After the calibration the user can select a reduction in the walking speed and the actual Mindful Walk starts. During the Mindful Walk the Apple Watch shows the distance, the pace, the heart rate and the active energy burned.

The App is designed to be a companion for a clinical study and allows to collect crowd-sensed based data from users that participate in the Mindful Walking study.

2.2.3 Sensor-enabled mobile apps

Runtastic

One of the best known app that targets end users and integrates internal as well as external bluetooth sensors is the mobile application "Runtastic" [29]. Runtastic is a fitness tracker app that supports tracking outdoor activities, tracking training progress and reaching fitness goals [30]. Figure 2.2 shows a workout sessions in the Android version of the runtastic application. The workout session screen shows a chronometer, the walked distance, the burned calories, and the pace. It also brings the functionality to track the GPS route and illustrate it on a map. The application offers the ability to connect different pulse sensors to the mobile application. Both, the sensor readings as well as the GPS tracks can be logged and analyzed with the runtastic platform.

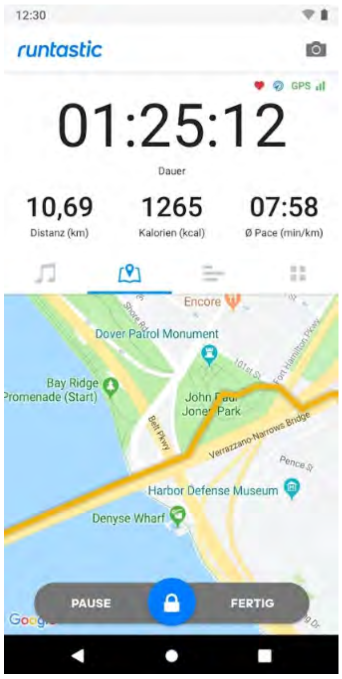


Figure 2.2: Workout session in the mobile app "runtastic" showing a chronometer and several sensor readings [30]

XFitXtreme

The XFitXtreme application is a real-world mobile business application from the fitness domain that has been developed by the Institute of Databases and Information Systems at University of Ulm [29]. To integrate bluetooth sensors a sensor framework has been developed that can easily be integrated into Android applications. The mentioned framework was also used in the XFitXtreme Application. It encapsulates the heterogeneity of data structure of different sensors as well as the communication between the external sensor and the smart phone [29]. Furthermore, the application provides functionality to connect to a sensor as well as receiving data from the sensor. XFitXtreme offers functionality to monitor, record and graphically evaluate the sensor values provided by the framework. It may be used by athletes that train CrossFit [29]. Figure 2.3 shows how the vital signs are visualized during the workout.

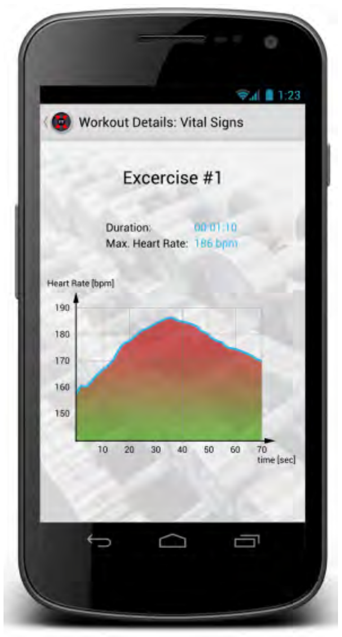


Figure 2.3: Visualization of vital signs during the workout in the XFitXtreme mobile application [29]

2.2.4 Discussion

Several apps have been developed that either support clinical studies or integrate internal or external sensors to collect and log vital signs. The app introduced in 2.2.1 integrates an user interface that may be used to digitize paper-based questionnaires and distribute them to mobile devices. For each page of a questionnaire, the app provides a virtual page and each question is shown in the design of a small card. The evaluated user interface provides functionality to deliver questionnaires to mobile apps in a generic way.

The Mindful Walking app introduced in section 2.2.2 provides a functionality to participate in a study that is collecting the participants answers to the questionnaires and the sensor values of the Apple Watch for research purpose. Although the app integrates questionnaires that belong to the study and uses sensor values of the external Apple Watch, the app cannot be used to participate in generic studies.

The Fitness apps Runtastic and XFitXtreme introduced in 2.2.3 provide functionality to connect to different external bluetooth sensors. They also implement an easy to use user interface that provides an overview over the current sensor values and vital signs. However, these apps are not designed to create and start generic workouts and do not support scientific studies.

The objective of the following chapters is to develop a generic app that can run and support any clinical study and provides functionality for generic questionnaires, generic actions and can read and monitor internal sensors and external bluetooth sensors.

3 Requirements

In this chapter the functional and non functional requirements of the framework are introduced.

3.1 Functional requirements

1. **QuestionSys as interface for generic studies:** The *QuestionSys Configurator* should be used as a interface for generic questionnaires, studies and exercises that include sensor values.
2. **Multiple session studies:** The framework should be able to handle studies that consist of multiple sessions that are executed at different days.
3. **Meta data:** Since the framework should support multiple studies containing multiple sessions, the framework should also support saving and reading of meta data. Meta data means data, that is not linked to a specific session of the study but to the study itself and may be used to read results of previous sessions.
4. **Multiple studies:** The framework should be able to list, handle and execute multiple studies. For the study input JSON formatted content should be passed to the framework.
5. **Elements:** A study consists of one or more study sessions and each study session consists of one or more study pages. A study page consists of nodes and a node may be a headline, a text element, a media element, a question or a custom element. Elements are classic questionnaire elements like likert scales or drop down menus.

6. **Questions:** The framework should support and provide UI elements for drop down questions, multiple choice questions, single choice questions, yes/no questions, date input, single line text input, multi line text input, likert scales, likert scales with images, single sliders and slider ranges.
7. **Logic elements:** The framework should support logic elements like XOR splits and JOIN elements. XOR split elements split the questionnaire into multiple paths based on either node result data or meta data input.
8. **Custom elements:** The framework should support custom elements. Custom elements are elements that extend the functionality of *QuestionSys* to functions like *Sensor Configuration*, *Sensor Start*, *Sensor Stop* or *Action Elements*.
9. **Action elements:** The framework should support action elements. Action elements are exercises that are included in the study and request the user to do an exercise. When an action has started, previously configured sensor data should be logged and shown to the user. Also there should be chronometer and a timer for action elements.
10. **Sensors :** The framework should support smart-phone internal sensors as well as external bluetooth sensors that support the Bluetooth Low Energy protocol.
11. **Sensor Notifications:** During an exercise the user should get notifications if the sensor values are exceeding a maximum limit or fall below a minimum limit.
12. **Pausable:** A study should be pausable at any time and input data should remain persisted.
13. **Notifications:** The framework should support notifications that remind the user to start a new study session. Study sessions may be scheduled by the creator.
14. **Results:** The framework should be able to export study results after the study has been finished and upload to a RESTful API.
15. **Statistics:** The framework should be able to create statistics and feedback for the user. Particularly it should show sensor values of the exercises included in a study.

16. **Log-in:** The app should provide an user log-in and the functionality to download user specific studies.
17. **Multi-language support:** The app should support multiple languages. This includes user interfaces, questionnaires and user instructions.

3.2 Non-functional requirements

1. **Generic:** The framework should be generic and work with different kind of studies.
2. **Android compatible:** The framework should be compatible to the Android mobile system and should work with all kind of Android devices.
3. **Well structured input data:** Input data for studies should be well formatted and has to be equal to the JSON syntax to be platform independent and to be reusable by all kind of applications.
4. **Well structured output data:** Study result data should be well formatted and has to be equal to the JSON syntax to be platform independent and reusable by all kinds of applications. Result data should be machine processable and analyzable to create statistics and study results.
5. **Loose coupling:** The framework should be able to be embedded into other Android applications.
6. **Modularity:** The framework should be modular to easily allow exchanging parts of the framework.
7. **Extensibility:** The framework should easily be extendable by new questionnaire elements or sensor devices.
8. **Offline mode:** The framework should be able to work without internet connection. Studies should be downloaded and saved on the device, media objects should be cached for offline use.
9. **App feeling:** The app should be usable by all kind of users. It should instruct the user during the execution of a clinical study.

4 Architecture

This chapter introduces the architecture of the *QuestionSys studies* framework. First, the Quengine's process is illustrated, then the class structure, the data model and persistence is shown. In the end the input data structure and the results structure are explained.

4.1 Quengine Process

The *Quengine* is the heart of the *QuestionSys studies* framework and handles the whole process that belongs to the execution of a study. It serves as the main entry point and can be embedded in any Android fragment of any application by passing an Android viewgroup to the *Quengine*. The Android viewgroup is used as a container in which the study is loaded and presented.

Figure 4.1 illustrates a BPMN model that represents the basic execution process of a study. The process also contains the interaction between the user and the *Quengine*. The model is separated into two BPMN pools: the user pool on the top and the *Quengine* pool on the bottom.

The process starts when the user loads a study into the *Quengine*. The *Quengine* initializes the study as well as the UI elements and informs the user by showing a start button. Now the user can start the study by pressing the start button. As soon as the user selects the next page, the *Quengine* gets notified and increments the *current page* object.

Afterwards all nodes that belong to the current page are loaded and handled. That means, if the nodes contain UI elements, these nodes are added to an array of UI element nodes. In contrast, if they contain logical elements like XOR Splits or Joins with conditions, these conditions are evaluated and the next page is calculated and written to the *current page* object. Then the nodes are processed again until there are only nodes with UI elements left.

The UI elements are loaded into a viewadapter that presents all UI elements to the user. The user then has to fill in the corresponding data. After the data is filled in, the *Quengine* checks if all input is complete and correct. If data is missing or incorrect, these UI elements highlighted and the user has to fill in the data again. This process is repeated until all necessary information is available and correct.

After a page is finished, the user may then choose to pause or to finish the study. If one chooses to finish the study, the *Quengine* saves the meta data that contains the study cycle and the execution time-stamp. Otherwise the user can access the previous or the next page of the questionnaire.

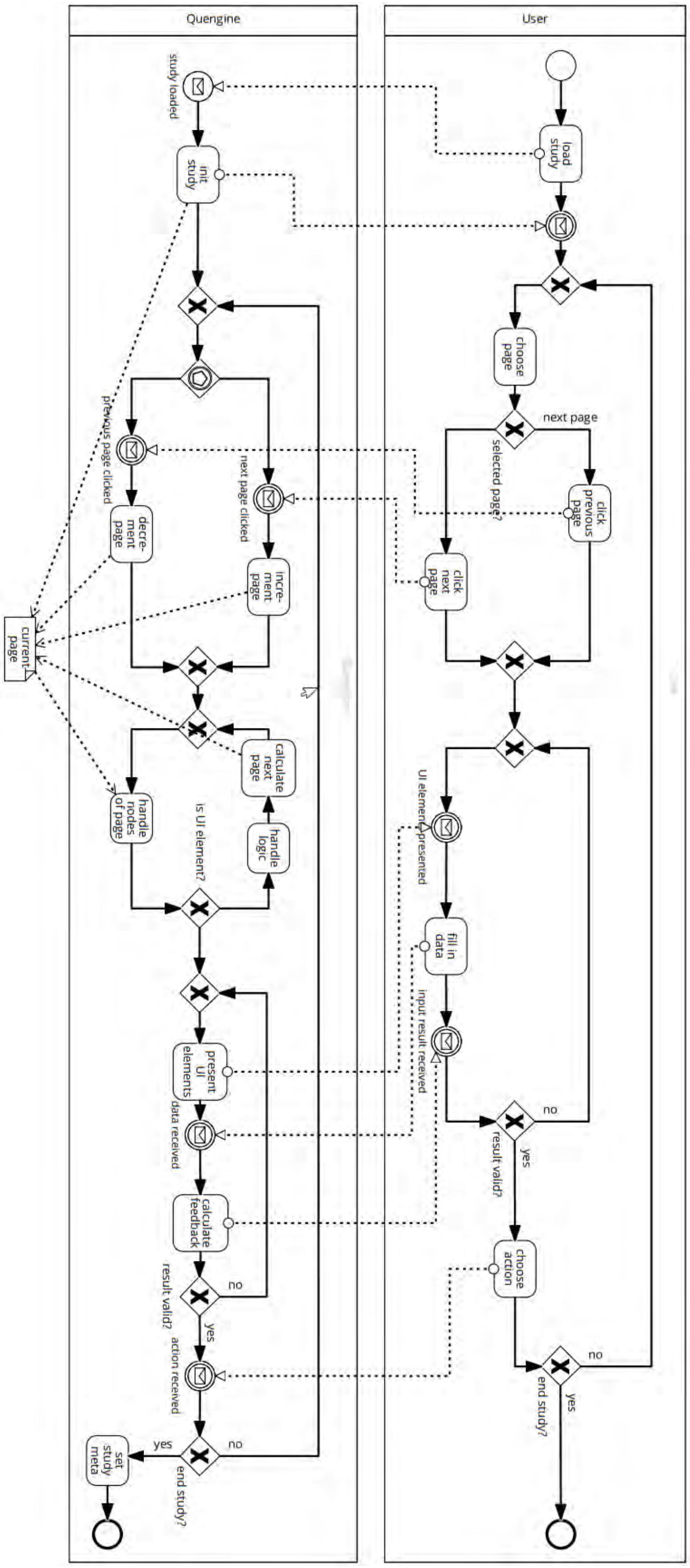


Figure 4.1: Business logic of a study execution containing interaction between the User and the Quengine

4.2 Class architecture

Figure 4.2 illustrates the class architecture of the *QuestionSys studies* framework. As mentioned before, the *Quengine* on the top left of the diagram is the main entry point of the framework. The architecture's design objective was to implement the framework as modular and expandable as possible. Keeping the framework modular was the reason to separate functionality that belongs to the user interface, the sensors or the persistence of the study outcomes.

The user interface is handled by the *StudyPageAdapter* which instantiates all UI classes that belong to a specific page of the study. All UI classes that may be used in the study have to implement the interface *QView* which itself implements checks for user input and marking the correctness of the user input. Using an Android Adapter ensures asynchronous loading and lazy loading for UI elements, which means the user does not have to wait while study pages that contain many UI elements are loading.

The *Quengine* also holds the *StudyResultsEngine* which is responsible for the persistence of the user input as well as other study outcomes. Therefor the *StudyResultsEngine* as well as the *SensorLoggingEngine* extend the abstract class *PersistenceEngine*, which implements all database functionality.

The *SensorManager* which is also a component of the *Quengine* is responsible for the sensor functionality. It can be distinguished into internal and external sensors. In this context, all sensors that are included in the smartphone, such as the location sensor, gyroscope, accelerometer, etc. are defined as *internal sensor*s. The term *external sensor*s includes all sensors, that are connected via Bluetooth Low Energy. An example is the heart rate sensor in smart watches. The *SensorManager* may instantiate an arbitrary amount of *InternalSensor* devices and *BluetoothDevice*s. The *InternalSensor* devices need to implement the interface *InternalSensor* and for the *BluetoothDevice*s, each *BluetoothDevice* is managed by a *BluetoothDeviceService*.

Both, the *BluetoothDevice* and the *InternalSensorDevice* implement the *LoggableAttribute* which exports an attribute that is used by the *SensorLoggingEngine* to log the sensor values. The *LoggableAttribute* is also used by the *NotificationEngine* to check if the sensor values exceed or fall below the notification thresholds.

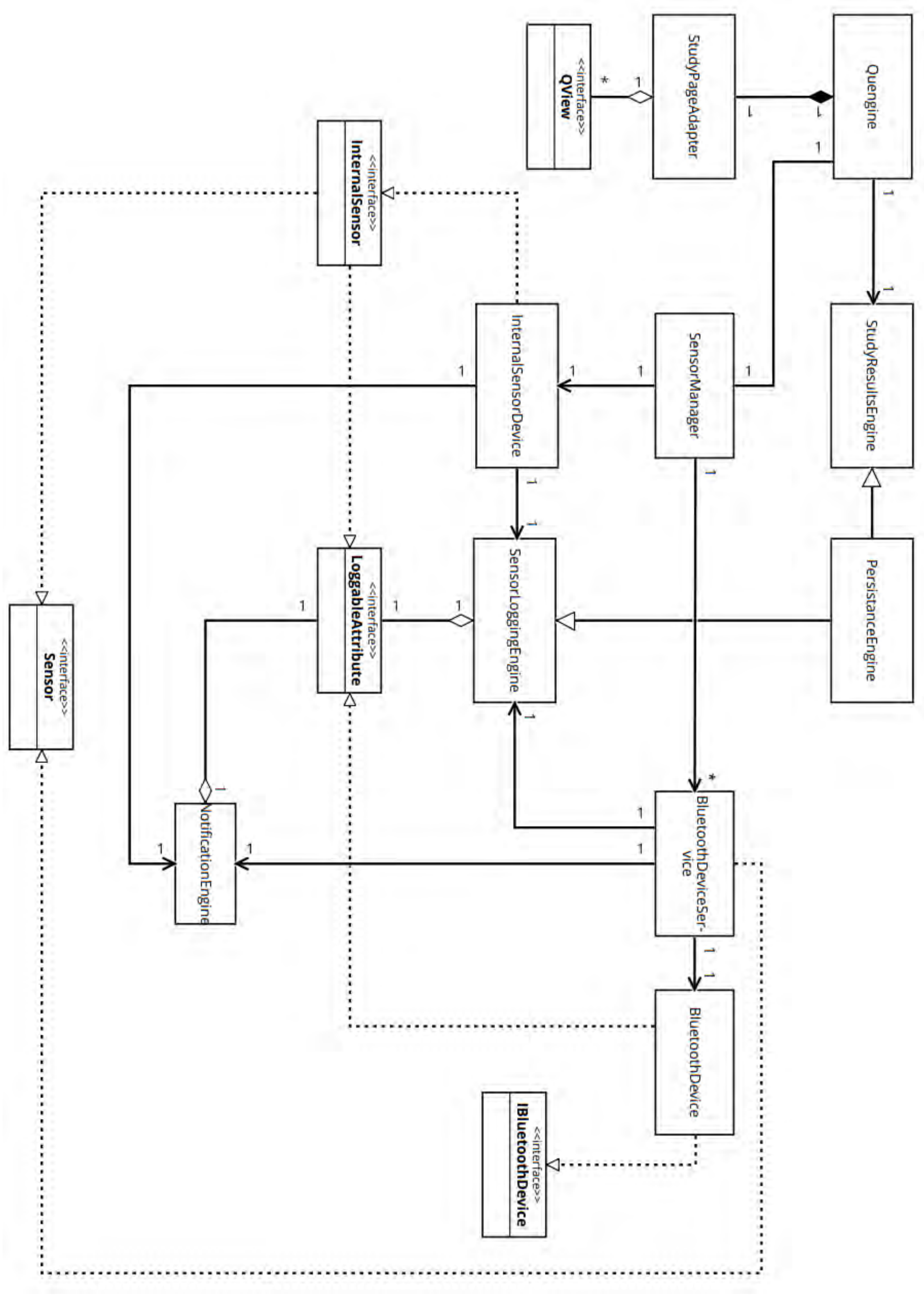


Figure 4.2: Class architecture of the *Quengine* including *SensorManager*, *PersistenceEngine* and *NotificationEngine*

4.3 Data Model and Persistence

In this section the data model that is used by the *Quengine* is explained. Figure 4.3 illustrates the complete data model that is used as the input model for every study that is executed.

Every study is represented by a *Study* object which contains a *Meta* object with meta data, an arbitrary amount of *MediaObjects* and one *Model* object. The *MediaObjects* contain information to gain access to media that is included in the study. That may be images, videos and audio files. The *Model* object is used as a container for *Nodes* and *LinkData* Objects. A *LinkData* Object marks possible paths from *Node* to *Node* by providing a *from object* and a *to object*. These objects contain the keys of the *nodes*, that are included in the study.

Each *node* has a key, a group id (that is used by the *Quengine* to assign the node to groups like study pages), a *correspondingNodeKey*, a boolean *isGroup*, a *text*, a *category*, and an *ExecutableComponent*. The *ExecutableComponent* specifies the type of the node. Types may be *PageComponents*, *XORSplitComponents* or *XORJoinComponents*. Every node may contain an *Element* which contains detailed information for the component if needed.

The attributes of an *Element* can be distinguished into those that are for specifying questions with UI elements, and those that contain logical elements.

For specifying questions *Elements* contain:

1. a question type
2. an export name, which can be set by the creator of the study and is used by the framework as the identifier for the element
3. a flag that indicates, if the question is mandatory to fill in
4. an instruction for the user
5. a caption for media objects, if there are media objects included
6. a unit for questions that need to be filled in

7. a media id, if there are media objects included
8. a flag that indicates, if the question is linked to a *MediaObject*
9. a minimum and maximum length attribute if the user has to type in free text
10. a start number
11. an end number
12. a step size

The start number, end number and the step size are mainly used for sliders. Question types may be Likertscale, Likertscale with Images, SingleSlider, FreeText, etc.

For the logical *Elements* a gateway type is defined which represents the logic element like a XORSplit element. In case of a XORSplit *Element*, the *Element* also contains one or more *Branches*. A *Branch* is a split in the process model. It indicates the direction in which the process engine may continue.

A *Branch* contains a name, the key for the next *Node*, a indicator if the *Branch* is the default *Branch* and a condition string or one or more *Variable mappings*. A condition string is a predefined string that is evaluated by the process engine and may access meta data like the current *study cycle* or *time stamp*. A *VariableMapping* is a mapping to the outcome of a question and compares the output with a given value. Only if all *VariableMapping*s are evaluated and all conditions are true, the *Branch* conditions are fulfilled.

For the *Elements* that belong to a question, the *Element* may also include *Items* that represent options that the user can select in case of Single Choice or Multiple Choice question.

For the sensor configuration and action view configuration the *CustomConfiguration* and *CustomContent* models have been added. The *CustomConfiguration* contains a type and a sensor id if the *Element* references a sensor.
The *CustomContent* model contains a day attribute that may be used to schedule notifications. For reading and writing of meta values it also contains one or more meta objects that have a name and may also have a value. For the action views, a action

config is included that defines the headline and an instruction for the user. It also defines a countdown for the action and provides the seconds for the countdown.

For the sensor configuration there are one or more *Sensor* models that contain an id, a name, a sensor type and a sensor attribute. It also contains the configuration for the sensor value logging and the user notifications. The log config contains a flag that defines if logging for the sensor is enabled and the frequency of the logging in seconds.

The *FluidNotification* model also contains a flag, that indicates if the notifications are enabled as well as a notification frequency. Also the minimum and the maximum threshold of the notifications are defined in the *NotificationThresholdValue*. In this context *FluidNotification* means, that the user is constantly notified about the current sensor values while the sensors are running.

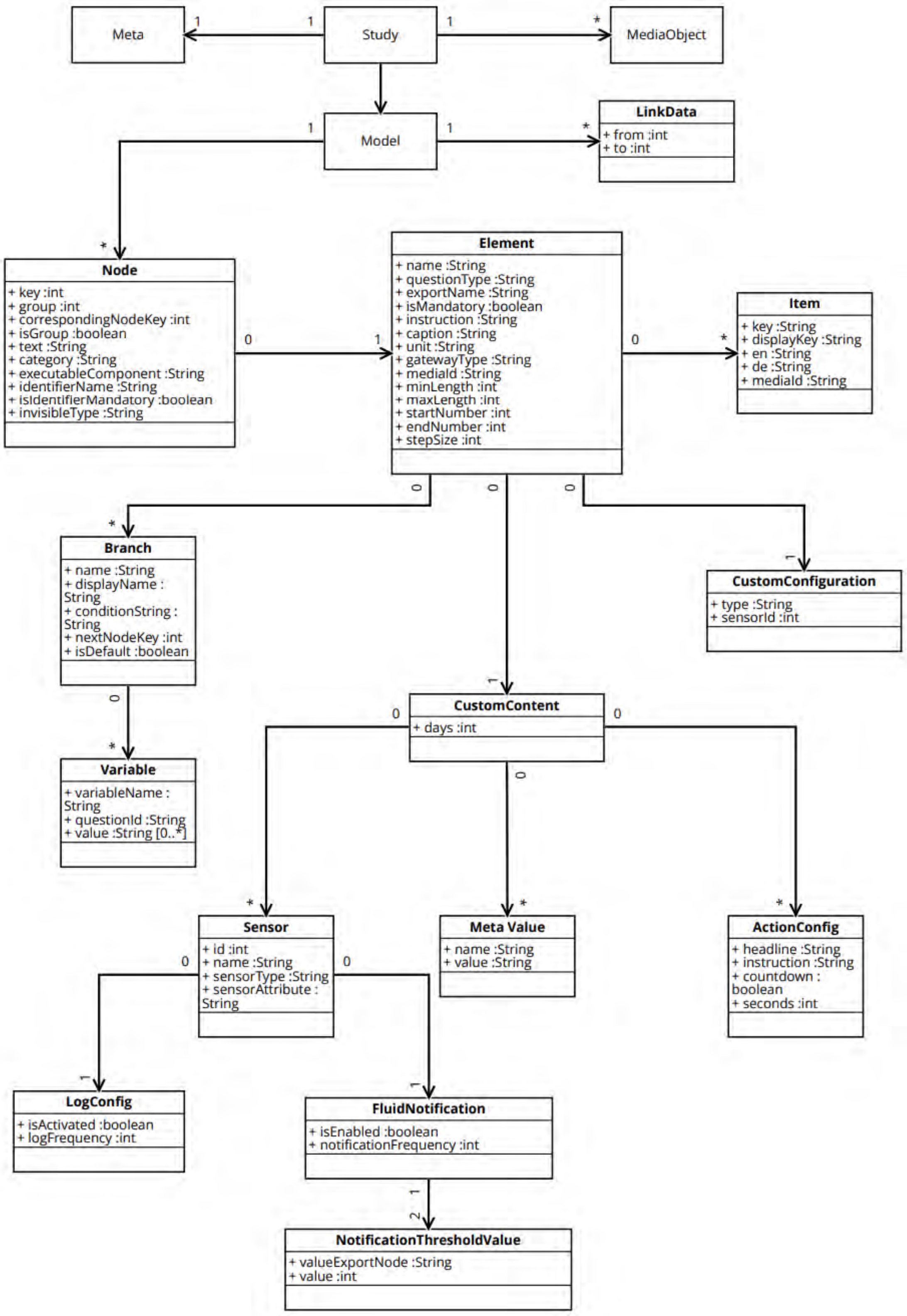


Figure 4.3: Data model of a study processed by the Quengine

4.4 Input Data structure

The exported data of the *QuestionSys Configurator* is used as the input model for each study. The *QuestionSys Configurator* doesn't support configuring sensors and action elements. The sensor configuration and other functionality that had been added to the framework have been implemented by using a *Custom Element* in the *QuestionSys Configurator* and adding the data to the *Custom Configuration* attribute and *Custom Content* attribute of that element. The input data is provided as a JSON formatted string and can easily be translated into the data model shown in section 4.3. No modifications have been made to the JSON export, so the JSON can directly be processed by the *Quengine*.

However, at this point the API integration has not yet been implemented, since the API has not been finished yet. Nevertheless, preparations have been made. In the sample app there is a fragment showing a list of all available studies the user has access to and since a JSON formatted file is used as input for the study data, the integration with the API is prepared.

Listing 4.1 shows a sample study input in JSON format. For demonstration purposes the listing has been shortened. The JSON contains three root objects: *meta*, *model* and *media*. *Meta* includes all meta data belonging to the study like the contact person, the name and version. The *media* object contains all media objects that can be downloaded from the server. The *model* is the main part of the study and contains a *node data* array and a *link data* array. The *link data* array is just an array with references from one *node* to another *node*. Every object represents a path in the process model of the study.

The *node data* array contains all *node*s of the study. A *node* may be a logical element or contain UI elements like questions. In Listing 4.1 the *node* represents a Likert question. The group is used to link the node to a specific page of the study. In the *element* object the information about the question is saved. The *Items* array contains all possible answers the user can select. The study is fully multilingual and supports English as well as German. Additional languages may be added easily.

```
{
  "meta": {
    "contact": "Hollie Nikolaus",
    [...]
    "version": "2.0.0"
  },
  "model": {
  "nodeDataArray":
    [
      {
        "text": "WHO-5.4",
        "category": "Question",
        "group": -140,
        "element": {
          "name": "WHO-5.4",
          "question": {
            "de": "In den letzten zwei Wochen ...",
            "en": "Over the past 2 weeks ..."
          },
          "questionType": "Likert",
          "exportName": "WHO-5.4",
          "isMandatory": true,
          "instruction": {
            "de": "... habe ich mich beim Aufwachen frisch [...]
            "en": "... I woke up feeling fresh and rested."
          },
          "items": [
            {
              "key": "1561237454541497",
              "displayKey": "1",
              "de": "Die ganze Zeit",
              "en": "All of the time"
            },
            [...]
          ]
        },
        "key": -148
      }
    ],
  "linkDataArray":
    [
      {
        "from": 1,
        "to": -5
      },
      [...]
    ]
  }
}
```

Listing 4.1: Sample study input formatted as JSON formatted string

4.5 Results structure

The objective of this master's thesis is to provide a simple, standardized and machine processable export function to export all study outcomes like user inputs and sensor logging data. Another goal of this scientific work is to provide an easy functionality to create statistics about the study results and to be able to process huge amount of data. Therefore the JSON format has been chosen as the study results export format.

Listing 4.2 shows a sample result of a user participating in a study:

```
{
  "studyCycles":
```

```
[
  {
    "cycle": 1,
    "questions":
    [
      {
        "nodeKey": "-120",
        "questionTitle": "Over the past 2 weeks ...",
        "resultValue": "1561237592235256",
        "resultValues": []
      },
      [...]
    ],
    "sensorResults":
    [
      {
        "sensorName": "HEARTRATE",
        "sensorRuns":
        [
          {
            "sensorValues": [
              61.0,
              61.5,
              [...]
            ],
            "sensor_run": 1
          }
        ],
        "timestamp": "2019-06-24"
      }
    ]
```

```
      }
   ]
}
```

Listing 4.2: Sample result of a user participating in a study

The export starts with a *study cycles* array that contains all cycles of a study that have been performed by the participant. Every study cycle object contains the cycle number, a time-stamp of the cycle, a *questions* array that contains the outcome of all questions the participant has answered and an array *sensorResults* containing the sensor logs.

Every *question object* contains the node key of the question, the question title and either a question value or an array with question values if the question allows or requires multiple result values. The result value represents the result option key of the corresponding question. The node key may be used to allocate the corresponding question and the result options that are available.

Since the framework allows logging of multiple sensors for each study, each object in the *sensor results* array represents the logs of a single sensor. It contains the sensor name of the sensor that gets logged and a *sensor runs* array. The *sensor runs* array is necessary, because in each study cycle the sensors may be started and stopped multiple times. Each time the sensor restarts another object in the *sensor runs* array is pushed. Each object contains the sensor run value and an array *sensorValues* of the actual values.

5 Implementation & Implementation Aspects

In this chapter selected implementation aspects are presented and discussed. First, layout aspects are discussed, then the view adapter and the *QViews* are shown. Third, the persistence manager is illustrated. Then meta data is discussed and last the sensor integration is shown.

5.1 Layout Aspects

It was a requirement of the study to create a clean, tidy and uniform feel and look without a lot of unnecessary overhead, so that any participant of any country and culture can use the application.

Each study consists of one or more study pages that contain different types of questions or other UI elements for the participant. To create a uniform look and feel it was the main idea to create an interface that can be embedded into any Android View and automatically generates the UI elements that belong to the study.

Integrating the framework into a sample app requires a Android fragment with at least two Android view groups. One view group is used for adding the meta layout containing the previous, start and next page buttons, the other view group is used as a container for the actual UI elements that belong to the study. Usually, the second view group is nested into the first one. It is also possible to customize the first view group and add additional UI elements or a background image. In the sample app every study includes a background image and the meta data as well as the UI elements overlay that background image.

Listing 5.1 shows the constructor of the *Quengine*. The constructor is called in the fragment that embeds the *Quengine*:

```java
public Quengine(Context context,
String studyJSON, EngineHolder engineHolder) {
        [...]
}
```

Listing 5.1: *Quengine* constructor

The constructor expects three different data objects: the Context, a JSON formatted string that contains information about the study and the *EngineHolder*. The *EngineHolder* is the fragment that should hold the UI elements to the *Quengine*. The rest is completely managed by the *Quengine*.

The fragment that is used to render the study elements must implement the *EngineHolder* interface:

```java
public interface EngineHolder {
    public void addMetaLayout(View view);
    public Fragment getFragment();
    public void setAdapter(RecyclerView.Adapter adapter);
    public void clearAdapter();
    public void endStudy();
    public void pauseStudy();
}
```

Listing 5.2: *EngineHolder* Interface

The interface provides a method to add the meta layout to the fragment. The meta layout is needed for the page counter and the meta buttons. It also provides methods to get the fragment that is needed for app permission handling, to set the view adapter that is discussed in section 5.2, to clear the adapter (when the page is changed), to end the study and to start or to pause the study.

To create a uniform look and feel, every UI element that belongs to a study and requires interaction with the user, has a white and semi-transparent background layer that lets the background image of the study shine through but also improves contrast and readability. The design language is referred from a white card.

This card element is split into a headline element with white text color that is drawn on a background in the configurable accent color of the app and the body element containing an user instruction and the interaction elements. The interaction elements can be various types of elements like text inputs, radio buttons, check-boxes, sliders, media elements and also complex elements like action elements. Each page may have any amount of questions and becomes scrollable if the cards do not fit the screens' height.

Figure 7.19 shows a typical page of a study containing multiple questions. Each question is represented by a card element and each card element contains a headline on the top of the card and the UI elements that belong to the question in the body of the card.

5.2 View Adapter & QViews

The *StudyPageAdapter* is responsible for loading and rendering all UI elements that belong to the study. The *Quengine* has exactly one *StudyPageAdapter*, which is passed to the fragment that serves as the container for all UI elements that belong to the study. The adapter handles all UI elements asynchronously so that the UI is only rendered at the time the user scrolled to that UI element. The *StudyPageAdapter* extends the Android native *RecyclerView.Adapter* and overrides the methods *getItemCount*, *getItemViewType*, *onCreateViewHolder* and *onBindViewHolder*. It also has a method *completePage*.

At the time of the creation, the adapter receives a list of nodes. That list contains the UI elements that need to be rendered on the page. Every time the user starts the study or switches the study page, a new page needs to be rendered and the adapter is recreated with a new list of nodes.

The *getItemCount* method returns the size of the node array that got passed to the *StudyPageAdapter*. The *getItemViewType* method returns an integer that represents the *viewtype* of the node at the selected position. Every question type, action or sensor configuration view is linked to a *viewtype* defined as a final int value. The *onCreateViewHolder* method then accesses the *viewtype* and creates the *QView* based on the *viewtype*.

The action view, the sensor configuration view and each question type implement the *QView* Interface and extend the Android View class. The *onBindViewHolder* method is called when an UI element becomes visible to the user and is used to initialize that view. For initialization the *init* method of the corresponding *QView* gets called with a *QConfig* as parameter.

The *QConfig* contains all information that is needed for the *QView* to provide the UI for the user. This includes the *Context*, the *Element*, the *Node* and the selected *Key/Keys*. Every *QView* needs different content to provide the UI for the user, so the *QConfig* was implemented in the Builder pattern and is instantiated using a static Builder class. The information that is relevant for each specific *QView* is passed to the corresponding *QConfig*.

During instantiation each *QView* loads the UI that belongs to the node, then reads the default data if there is any default data. If the user already set data for the node before, the saved data is loaded from the *realm database* and shown in the UI. The realm database is an object-based, light-weight and open source database available for Android and other plattforms [31]. All database operations in the framework are implemented asynchronously, so that there are no lags when scrolling through the nodes of a page.

Figure 7.17 shows the action view as an example for a *QView*. As explained in section 5.1 the *QView* is shown as a card layout that contains a headline and a card body. In the card body the actual UI elements are shown. The headline of the action *QView* is always "Action" as there is another headline inside the card body.

Under the headline in the card body there is a countdown timer or depending on the configuration alternatively a chronometer. The Action view can be configured to either show a countdown timer or a chronometer. If a countdown timer is shown, then the user has to start the timer and cannot complete the action view before the countdown has completed. Below the countdown timer there is an Android grid view which is used to show all configured sensor values in a grid. Each sensor features an icon, a value and the unit of the value. On the bottom there is space for an additional text like an instruction and a start/stop bottom to start and stop the timer, respective the chronometer.

When the user goes to the next page, first the current *StudyPageAdapter* finishes all UI elements and checks if the user input is valid. Therefor the *QView* interface provides the methods *finishNode* and *markInputValid*. First the *markInputValid* method is called for every *QView* and each *QView* checks if the user input is complete and valid. If the input is not valid, the view marks the invalid input and the method returns "false". The page is not changed then. If all inputs are valid, for each *QView* the method *finishNode* is called. Each node finishes ongoing processes and saves the user input. The action view, for instance stops all running sensors.

5.3 Persistence Manager

One of the main elements of the framework is the functionality to save and persist user input and sensor values. In order to provide an easy and centralized way of saving study result data, the *PersistenceManager* was created. Figure 5.1 shows the class diagram of the *PersistenceManager*, the *SensorLoggingEngine* and the *StudyResultsEngine*, which both extend the *PersistenceManager*.

The *PersistenceManager* uses a realm database to persist study results and sensor logs. Therefore it provides the Realm instances that are used to read and save data. Since every node may have different requirements for saving data and uses different data types, a generic and extendable way of saving data was needed. For that reason the *SaveDataObject* that extends *RealmObject* was created. A *SaveDataObject* provides several fields for saving data. Different fields are used by different question types as the data type may vary. A user input for example may save a string, a slider may save an Integer. Each *SaveDataObject* contains fields for assigning the *SaveDataObject* to a *studyId*, a *attributeName* and saving a *timestamp*. The *PersistenceManager* uses a realm database to persist study results and sensor logs. Therefor it provides the Realm

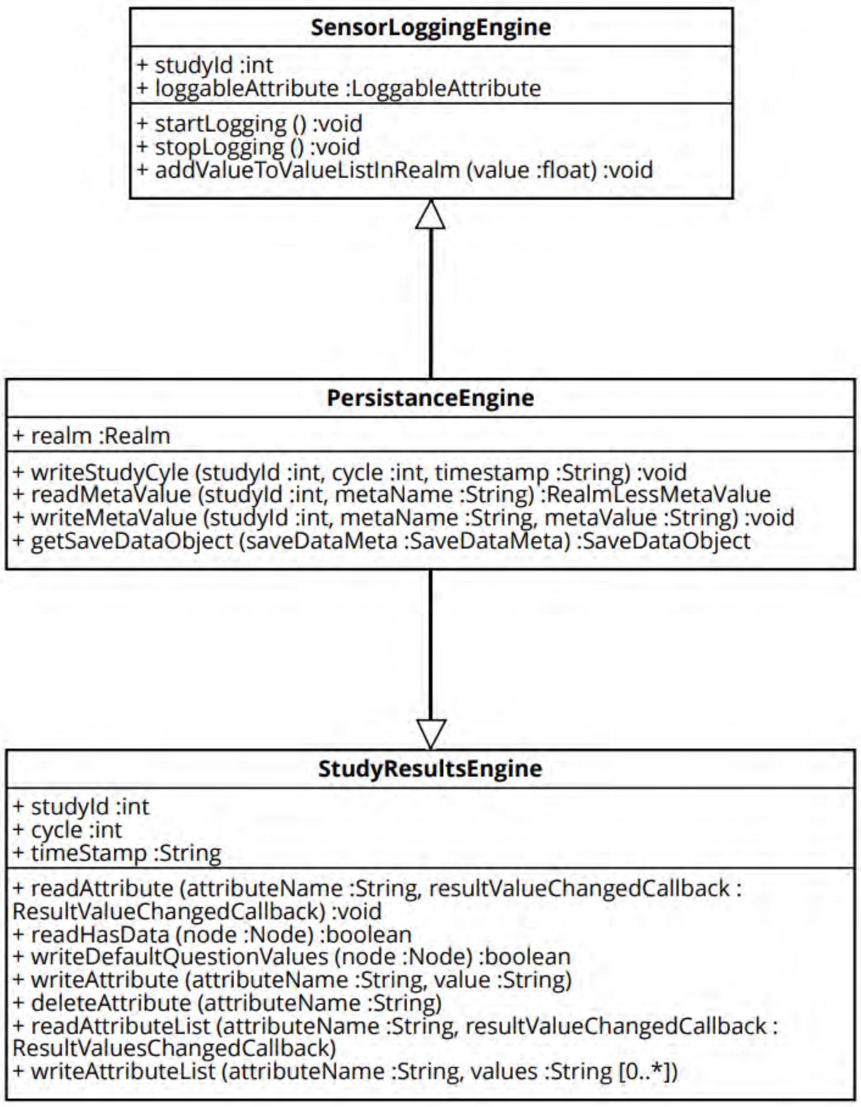

SensorLoggingEngine
+ studyId :int
+ loggableAttribute :LoggableAttribute
+ startLogging () :void
+ stopLogging () :void
+ addValueToValueListInRealm (value :float) :void
PersistanceEngine
+ realm :Realm
+ writeStudyCyle (studyId :int, cycle :int, timestamp :String) :void
+ readMetaValue (studyId :int, metaName :String) :RealmLessMetaValue
+ writeMetaValue (studyId :int, metaName :String, metaValue :String) :void
+ getSaveDataObject (saveDataMeta :SaveDataMeta) :SaveDataObject
StudyResultsEngine
+ studyId :int
+ cycle :int
+ timeStamp :String
+ readAttribute (attributeName :String, resultValueChangedCallback : ResultValueChangedCallback) :void
+ readHasData (node :Node) :boolean
+ writeDefaultQuestionValues (node :Node) :boolean
+ writeAttribute (attributeName :String, value :String)
+ deleteAttribute (attributeName :String)
+ readAttributeList (attributeName :String, resultValueChangedCallback : ResultValuesChangedCallback)
+ writeAttributeList (attributeName :String, values :String [0..*])

Figure 5.1: Class diagram of the *PersistenceManager*, the *SensorLoggingEngine* and the *StudyResultsEngine*. Both extend the *PersistenceManager*.

instances that are used to read and save data. Since every node may have different requirements for saving data and uses different data types, a generic and extendable way of saving data was needed. For that reason the *SaveDataObject* that extends *RealmObject* was created. A *SaveDataObject* provides several fields for saving data. Different fields are used by different question types as the data type may vary. A user input for example may save a string, a slider may save an Integer. Each *SaveDataObject* contains fields for assigning the *SaveDataObject* to a *studyId*, a *attributeName* and saving a *timestamp*.

To save user inputs, two result value fields are provided as well as a list of result values. To save sensor logs, a field for the current *sensor run* is provided as well as a list of *sensor values*. The *sensor run* field is mandatory, since each study cycle may have multiple action views that may log the sensors multiple times. In this context a *sensor run* means the log of a sensor during a single action view. If there is another action view in a study, the sensors are started again in a new *sensor run*.

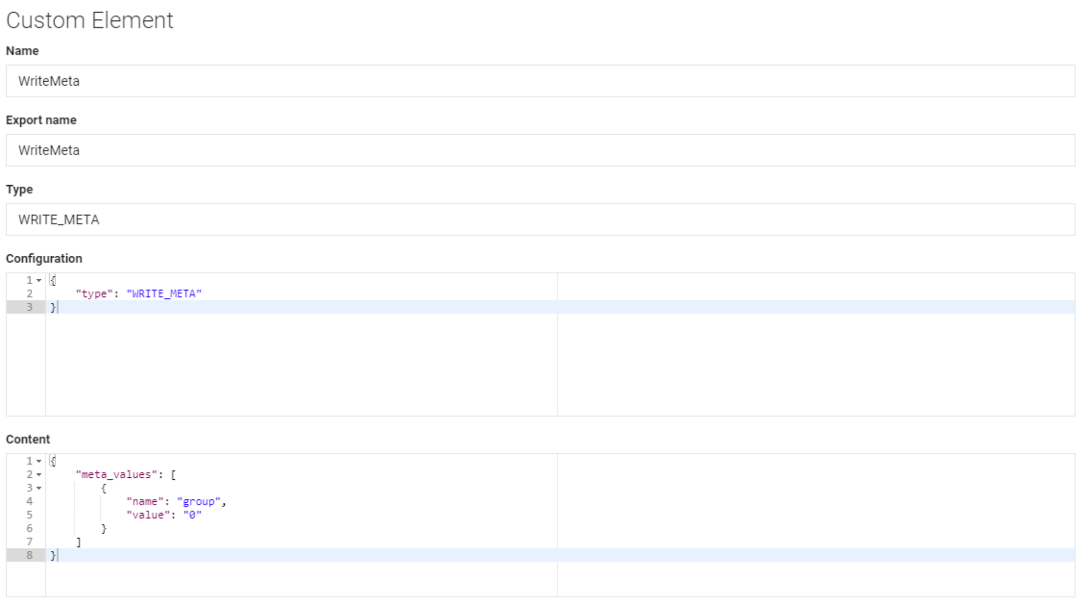
The *PersistenceManager* provides methods for getting *SaveDataObjects* and getting *MetaDataObjects*. These methods are called within the *StudyResultsEngine* and the *SensorLoggingEngine*. The *StudyResultsEngine* extends the *PersistenceManager* and adds the methods that are used to read and write user input. In contrast, the *Sensor-LoggingEngine* adds the methods that are used to read and write sensor logs.

5.4 Meta Data

As the framework supports multiple study cycles per study, it was a requirement to save user input that should be accessible not only within the current cycle of the study, but within all cycles. Therefore the framework supports reading and writing meta data that is valid for all cycles of a study. Meta data may be the age, the weight or other attributes of the participant, that usually don't change during the participation of a study, even if the study is performed multiple times with a break of several days.

For saving meta data, methods have been added to the *PersistenceManager* that support the reading and saving of meta values. Meta values are objects that consist of a name and a value. The method *readMetaValue* returns the Meta Object of a given *study id* and a given *meta value*. The method *writeMetaValue* writes the *meta value* for a given *meta name* and *study id*. *Study id*, *meta name* and *meta value* are all passed as parameters to the method.

If the study creator wants the *Quengine* to persist *meta values*, a custom element has to be used. The full configuration is illustrated in Figure 5.2. The figure shows how a custom element may be configured to direct the *Quengine* saving a custom value with the name **group** and the value **0**. The types' value has to be "WRITE_META". Furthermore it needs to be set in the configuration field. The content field contains the *meta values* as an array of JSON objects. Each object contains the name and the value of the *meta object* that has to be written.



Custom Element

Name: WriteMeta

Export name: WriteMeta

Type: WRITE_META

Configuration

```
{
    "type": "WRITE_META"
}
```

Content

```
{
    "meta_values": [
        {
            "name": "group",
            "value": "0"
        }
    ]
}
```

Figure 5.2: Custom node that is configured to write a meta value for a study

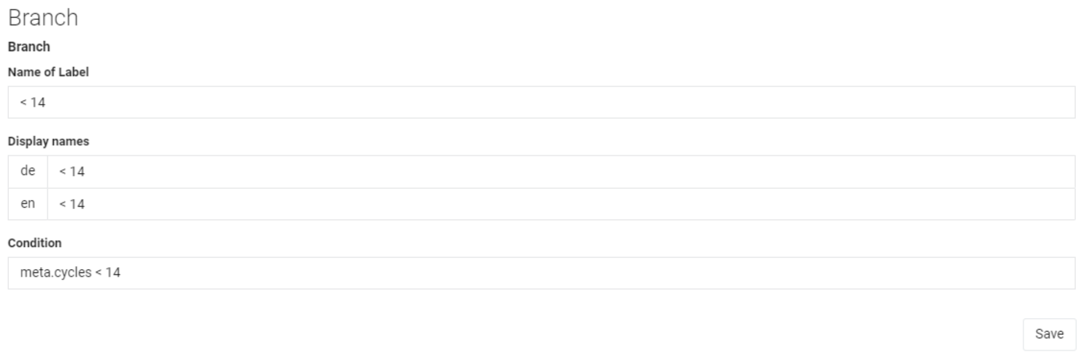
Listing 5.3 shows the method of the *PersistenceManager* that persists the meta data that gets passed to it. Before the method executes the realm transaction it checks whether the meta objects are available or not. If the object is already saved, then the value of the object becomes overwritten. If it doesn't exist yet, then the meta object is created and saved.

```
public void writeMetaValue(int studyId, String metaName, String
    metaValue) {
  Realm _realm = RealmHelper.getNewRealmInstance();
  _realm.executeTransactionAsync(new Realm.Transaction() {
    @Override
    public void execute(Realm _realm) {
      RealmList<MetaValue> metaValues = new RealmList<>();
      StudyMetaObject studyMetaObject
        = getStudyMetaObjectSynchronously(studyId, _realm);
      if (studyMetaObject.metaValues != null)
        metaValues = studyMetaObject.metaValues;
      boolean found = false;
      for (MetaValue val : metaValues) {
        if (val.name.equals(metaName)) {
          val.value = metaValue;
          found = true;
          break;
        }
      }
      if(!found){
        MetaValue metaValueObject = new MetaValue();
        metaValueObject.name = metaName;
        metaValueObject.value = metaValue;
        metaValues.add(metaValueObject);
      }
      studyMetaObject.metaValues = metaValues;
    }
  }, [...]);
}
```

Listing 5.3: *WriteMetaValue* Method of the *PersistenceManager* that persists the meta values

Meta values can be used to save data that belongs to more than one study cycle, but can also be used to control the process flow of a study and switch to different branches within the study based on meta values. If the study creator wants to separate study participants into two groups and let the participant perform different actions based on the participant group, one would save the assigned group as a meta value. One would then create a XOR node with two branches and set the condition for one branch to match the desired participant group. The *QuestionSys Configurator* brings the functionality to create branches and set conditions for each branch. To access previously saved meta values, one has to use the pattern: *meta.METANAME*. The framework automatically reads the value of an meta object with the *METANAME* after the prefix "*meta.*".

Figure 5.3 shows a sample for a branch condition that accesses a meta value.


Branch
Branch
Name of Label
< 14
Display names
de	< 14
en	< 14

Condition
meta.cycles < 14

Figure 5.3: Branch condition that accesses a meta value

5.5 Sensor integration

One of the main functions of the framework is the integration of both internal and external sensors in studies. Internal sensors are sensors that are integrated in the smart phone, external sensors are sensors that are connected via bluetooth to the smart phone.

It was goal of this master's thesis to implement a generic way to connect sensors, access sensor values and add new sensor types to the framework. Therefore internal and external sensor classes have been split. Both implement the shared interface *LoggableAttribute* and *Sensor* which functions as the interface to the *PersistenceManager* and the *NotificationManager*. The following two subsections introduce implementation aspects of the external sensors, the internal sensors and the sensor logging.

5.5.1 Bluetooth sensors

External sensors like step counters, heart rate sensors, etc. have to be bluetooth devices that use the Bluetooth Low Energy Standard and provide GATT attributes. As a case study a heart rate device that uses Bluetooth LE has been integrated in the framework and tested in the Mindful Walking study in chapter 6.

Figure 5.4 shows a class diagram that illustrates how external sensors have been integrated into the framework. The *SensorManager* holds an arbitrary amount of *BluetoothDeviceServices*. For every bluetooth device that is connected to the user device and is in the study one *BluetoothDeviceService* is instantiated and assigned to the device.

The *BluetoothDeviceService* holds a *BluetoothDeviceSetting*, a *BluetoothDevice* and overrides methods to start and stop a sensor. Therefor it implements the interface *Sensor*. Each *BluetoothDevice* has a *BluetoothDeviceSetting* that configures the sensor's attribute that should be read, the name of the bluetooth device and the MAC-address of the bluetooth device. This information is used by the *BluetoothDevice* class to connect to the physical device. The *BluetoothDeviceService* also provides a method to discover bluetooth devices in range, so that the user can configure the device to use.

The *BluetoothDevice* class is implemented as an abstract class and implements the interfaces *IBluetoothDevice* and *LoggableAttribute*. It has an attribute *currentSensorValue* which is a float value that represents the current value of the sensor. In case of a heart rate device this value would be the heart rate as the beats per minute.

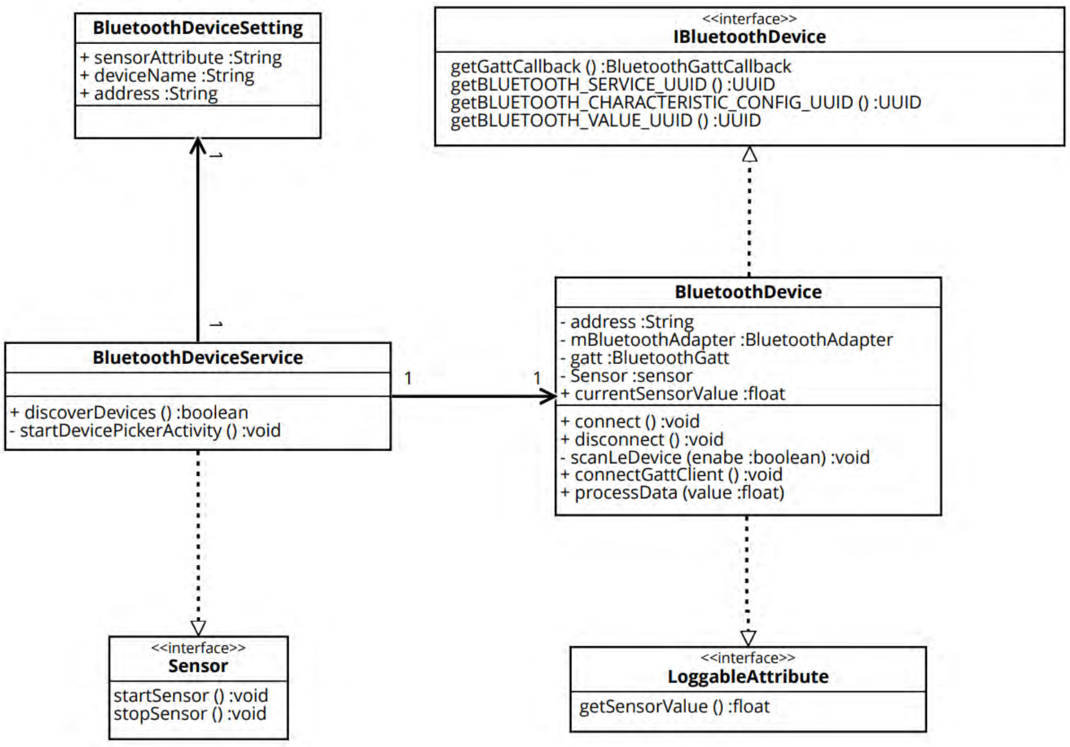

BluetoothDeviceSetting
+ sensorAttribute :String
+ deviceName :String
+ address :String
<<interface>>
IBluetoothDevice
getGattCallback () :BluetoothGattCallback
getBLUETOOTH_SERVICE_UUID () :UUID
getBLUETOOTH_CHARACTERISTIC_CONFIG_UUID () :UUID
getBLUETOOTH_VALUE_UUID () :UUID
BluetoothDevice
- address :String
- mBluetoothAdapter :BluetoothAdapter
- gatt :BluetoothGatt
- Sensor :sensor
+ currentSensorValue :float
+ connect () :void
+ disconnect () :void
- scanLeDevice (enabe :boolean) :void
+ connectGattClient () :void
+ processData (value :float)
BluetoothDeviceService
+ discoverDevices () :boolean
- startDevicePickerActivity () :void
1
1
<<interface>>
Sensor
startSensor () :void
stopSensor () :void
<<interface>>
LoggableAttribute
getSensorValue () :float

Figure 5.4: The class diagram illustrates the integration of external bluetooth sensors into the framework

The abstract class also provides methods for connecting and disconnecting bluetooth devices as well as scanning and searching for bluetooth devices. It also provides a method to connect to a GATT client. The *getSensorValue* method of the *LoggableAttribute* interface has been overridden and just returns the current sensor value.

Since the *BluetoothDevice* class only provides the methods to handle the bluetooth devices, but does not have any information about the different sensor devices, to integrate a real sensor into the framework one must create a class for every type of bluetooth device. That class must extend the abstract class *BluetoothDevice* and override the methods *getGattCallback*, *getBLUETOOTH_SERVICE_UUID*, *getBLUETOOTH_CHARACTERISTIC_CONFIG_UUID*, and *getBLUETOOTH_VALUE_UUID*. This is the only position in the code where one must register a new type of Bluetooth Sensor since the rest of the functionality is provided by the *BluetoothDevice* class.

Listing 5.4 shows how the *HeartRateDevice* class has been implemented to support reading of the heart rate using a Bluetooth LE heart rate sensor:

```java
public class HeartRateDevice extends BluetoothDevice {
    BluetoothGattCallback gattCallback =
    new BluetoothGattCallback()
  {
    @Override
    public void onServicesDiscovered(BluetoothGatt gatt,
      int status)
    {
      super.onServicesDiscovered(gatt, status);
      BluetoothGattCharacteristic characteristic =
        gatt.getService(getBLUETOOTH_SERVICE_UUID())
        .getCharacteristic(getBLUETOOTH_VALUE_UUID());
      gatt.setCharacteristicNotification(characteristic,
        true);
      BluetoothGattDescriptor descriptor =
        characteristic.getDescriptor(
          getBLUETOOTH_CHARACTERISTIC_CONFIG_UUID());
      descriptor.setValue(
        BluetoothGattDescriptor
          .ENABLE_NOTIFICATION_VALUE);
      gatt.writeDescriptor(descriptor);
    }
    [...]
    @Override
    public void onCharacteristicChanged(BluetoothGatt gatt,
      BluetoothGattCharacteristic characteristic)
    {
      super.onCharacteristicChanged(gatt, characteristic);
      int value = characteristic.getIntValue(
        BluetoothGattCharacteristic.FORMAT_UINT8, 1);
      processData(value);
    }
  };

    @Override
    public BluetoothGattCallback getGattCallback() {
        return gattCallback;
    }

    @Override
    public UUID getBLUETOOTH_SERVICE_UUID() {
        return convertFromInteger(0x180D);
    }

    @Override
    public UUID getBLUETOOTH_CHARACTERISTIC_CONFIG_UUID() {
        return convertFromInteger(0x2902);
    }

    @Override
    public UUID getBLUETOOTH_VALUE_UUID() {
        return convertFromInteger(0x2A37);
    }

    public HeartRateDevice(Context context, String address,
    Sensor sensor) {
      super(context, address, sensor);
    }
}
```

Listing 5.4: The class that represents a *heart rate device* and extends the *BluetoothDevice* class

The method *getGattCallback* returns the *BluetoothGATT* callback that is used by the *BluetoothDevice* class. Based on the type of device the callback must be adapted. *Bluetooth GATT* is an abbreviation of Bluetooth Generic Attributes and a part of the Bluetooth Low Energy Protocol. It establishes common operations and a framework for the data transported and stored by the Attribute Protocol [27]. In this context the bluetooth gatt characteristic is the actual sensor value. The *onCharacteristicChanged* of the GATT callback is called every time the bluetooth gatt characteristic changes. In that method one has to process the data and set the current sensor value to the value of the bluetooth gatt characteristic.

Every device that supports BluetoothGATT has at least one bluetooth service uuid, one characteristic config uuid and one bluetooth value uuid. These uuids depend on the sensor type one wants to read and are defined by the Bluetooth SIG [27].

In case of the heart rate device the service 0x180D represents the Heart Rate service defined by the Bluetooth SIG. The BluetoothGATTDescriptor with the uuid 0x2902 represents the Client Characteristic Configuration and is used in the BluetoothGattCallback to enable notifications when the characteristic value has been changed. The GATT characteristic 0x2A37 represents the Heart Rate Measurement characteristic and contains the actual heart rate value.

To add functionality for a new external sensor type, all three UUID's have to be looked up and the methods have to be adapted to return the correct uuid's.

5.5.2 Internal sensors

Internal sensors mean the sensors that are integrated in the user's device. Most smart phones have an internal GPS sensor, an accelerometer, a gyroscope and other sensors like brightness sensors.

An internal sensor in the context of a clinical study is not the raw hardware sensor output but the processed value one wants to log and present to the user. This may be a step counter that uses a combination of multiple hardware sensors, a distance sensor that uses the GPS sensor of the device or a speed sensor that calculates the speed in which the user is walking.

For the case study a speed sensor and a distance sensor have been implemented using the GPS sensors of the device to calculate the speed of the user and the distance the user has moved.

Figure 5.5 illustrates the implementation of the internal speed sensor and the distance sensor as a class diagram.

The class *SpeedmeterSensor* is the central class of the speed and distance sensors and implements the *InternalSensor* interface, which then implements the *Sensor* and the *LoggableAttribute* interfaces. As the *SpeedmeterSensor* class implements the *LoggableAttribute* interface and the user's speed as well as the distance are two independent sensor values, to integrate both of them at the same time, two *SpeedmeterSensor* classes have to be instantiated. The config that is passed to the constructor defines, if it provides the value for the speed or the distance.

Each *SpeedmeterSensor* class holds a *Data* object and a *GPSServices* class. The *Data* class contains all sensor values that are calculated from the GPS sensors. The *GPSServices* class extends an Android Service and implements a listener for the user location. It

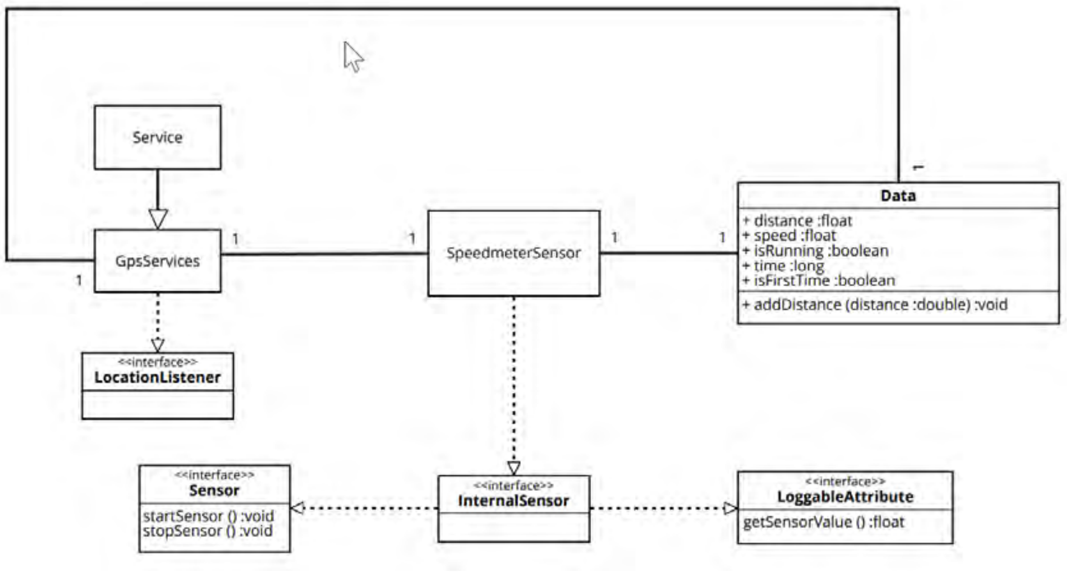


Figure 5.5: The class diagram illustrates the integration of a internal speedmeter sensor into the framework

contains all functionality to calculate the speed and distance the user has moved based on the raw GPS sensor values. As for the external sensors the *BluetoothDeviceService* provides the methods for start and stop the sensor, for the internal speed and distance sensors the *SpeedmeterSensor* class provides these methods. Listing 5.5 shows the method to start the sensor:

```
@Override
public void startSensor()
{
  if(!data.isRunning())
  {
    data.setRunning(true);
    data.setFirstTime(true);
    context.startService(new Intent(context,
      GpsServices.class));
  }
  if (sensor.logConfig.isActivated) {
    sensorLoggingEngine = new SensorLoggingEngine(0,
      sensor, this);
    sensorLoggingEngine.startLogging();
  }
  if (sensor.fluidNotification.isEnabled) {
    notificationEngine =  NotificationEngine
        .newBuilder()
        .Context(context)
        .StudyResultsEngine(studyResultsEngine)
        .FluidNotification(sensor.fluidNotification)
        .LoggableAttribute(this)
        .Nodes(nodes)
        .build();
    notificationEngine.startFluidNotification();
  }
}
```

Listing 5.5: *StartSensor* method of the *Speedmeter* sensor that starts the speed and distance sensors

To start a sensor, first it is checked whether the sensor is already running. If it is running, is doesn't get started again. Otherwise, the *Data* object, that contains the sensor values, is marked as *running*. The *GPSServices* Android service, that listens for location changes, is started. After the service is started, it is checked if the sensor that is used in the study is configured to log the sensor values and in case the logging is enabled, a new *SensorLoggingEngine* is started. The same goes for the user notifications and the *NotificationEngine*.

5.5.3 Sensor Logging

The main reason for sensor integration within clinical studies is to log and analyze sensor logs during user actions. For example, in the Mindful Walking case study the moving speed and the heart rate of the user is logged and may be used to show the progress of the user in the Mindful Walk sessions. Therefore the *SensorLoggingEngine* was implemented to log and persist the sensor values while the sensor is started.

Every sensor for that logging is enabled, holds a *SensorLoggingEngine* that is responsible for logging the values of the sensor. Since both external and internal sensors implement the *LoggableAttribute* Interface, the *SensorLoggingEngine* accesses the *getSensorValue* method that is overridden in the *Sensor* classes to access the actual sensor values. When the logging process starts, a Java thread becomes started which persists the sensor values to the realm database in a configured interval. Listing 5.6 shows the method in which the logging gets started.

```
public void startLogging()
{
  loggingTask = new Runnable()
  {
    @Override
    public void run()
    {
      addValueToValueListInRealm(loggableAttribute
        .getSensorValue());
      handler.postDelayed(loggingTask,
        sensor.logConfig.logFrequency * 1000);
    }
  };
  handler.post(loggingTask);
}
```

Listing 5.6: *StartLogging* method of the *SensorLoggingEngine* starts logging the sensor values

Within the *startLogging* method a new Java Runnable is created. The sensor value of the *loggableAttribute* is being added to the Realm database. For saving the realm entries, the methods provided by the abstract class *PersistenceManager*, which the

SensorLoggingEngine extends, are used. The task is called again after the given frequency of the *logConfig* of the sensor that becomes logged.

6 Case Study: Mindful Walking

In order to investigate if the designed framework can replace an individually created app for one clinical study and measure the typical effort to design a new study, a case study that is called Mindful Walking study has been created using the *QuestionSys Configurator*.

The main goal of the Mindful Walking study is to increase the level of Mindfulness the user experiences. Mindfulness has been defined as the intentional and nonjudgmental attention to experiences of the present moment [23].

For the Mindful Walking study, Walking activities in the study are integrated for the sake of increasing the Mindfulness of the user [5]. The study and the app should guide the user as well as possible through the process to gain a maximum result.

This chapter introduces a BPMN process model which builds the fundamentals of the technical model of the Mindful Walking study. The arised challenges while the process models was transformed to the technical model are discussed afterwards.

6.1 Process Model

The Process Model in Figure 6.1 and in Figure 6.2 is referred from the Bachelor's thesis *Conception and implementation of a mobile application to conduct a Mindful Walking Study regarding Clinical Psychology* [5], the Bachelor's thesis *Konzeption und Realisierung einer mobilen Anwendung zur Unterstützung von gestressten Patienten mithilfe des "Mindful Walking Gedankens* [25] and the Bachelor's thesis *A personalized* *support tool for the training of mindful walking: The mobile “MindfulWalk” application* [2].

For the questionnaires the German translation of the Five Facet Mindfulness Questionnaires has been integrated [18].

The Process Model Mindful Walking 6.1 is the study illustrated as an overall Business Process in BPMN. That means, for every user the process is only started and passed through once, even if the user participates in the study at different days and starts it multiple times in the app. These multiple cycles are modeled as loops within the process model.

The process of the Mindful Walking Study 6.1 starts by accepting the terms and agreements. If the user doesn’t accept the terms and agreements, the process immediately ends. If one agrees then one should fill in the WHO-5 (Five Well-being index), the PSQ (Perceived Stress Questionnaire) and the FFMQ (Five Facet Mindfulness Questionnaire) questionnaires. After filling in the questionnaires the participant is randomly assigned to a participant group (control group or experimental group). For each of the participant groups there’s another lane in the study.

For the control group one is first informed about the assignation and then has to wait for two weeks until one has to fill in the WHO-5, the PSQ and the FFMQ questionnaires again. Then the study has finished and the participant gets an notification that the study has been finished successfully.

For the experimental group the participant is also informed about the assignation, but then after one day the first Mindful Walk sub process can be started. Figure 6.2 shows the sub process of a single Mindful Walk.

A single Mindful Walk starts with an instruction shown to the participant. Afterwards one has to fill in the body scale State Mindfulness Scale (SMS). Then one has to determine the regular walking speed. That means one has to walk for a some time and measure the regular walking speed. After the regular walking speed has been measured, one has to choose a target speed. The target speed may be an reduction of 10%, 25% or 50% of the regular walking speed.

After choosing the target speed the Mindful Walk is performed. That means the participant has to walk for at least five minutes in the target speed, while speed, distance and heart rate are measured and logged. After the Mindful walk, the body scale SMS has to be filled in again as well as a specific Mindful Walk questionnaire provided by Ulm University. Then a variable *Mindful Walk cycles* becomes incremented and the Mindful Walk sub process is finished.

This sub process is performed fourteen times in an one day interval and after fourteen days the WHO-5, the PSQ and the FFMQ questionnaires are filled in again. Then the study is finished and a notification is sent to the participant.

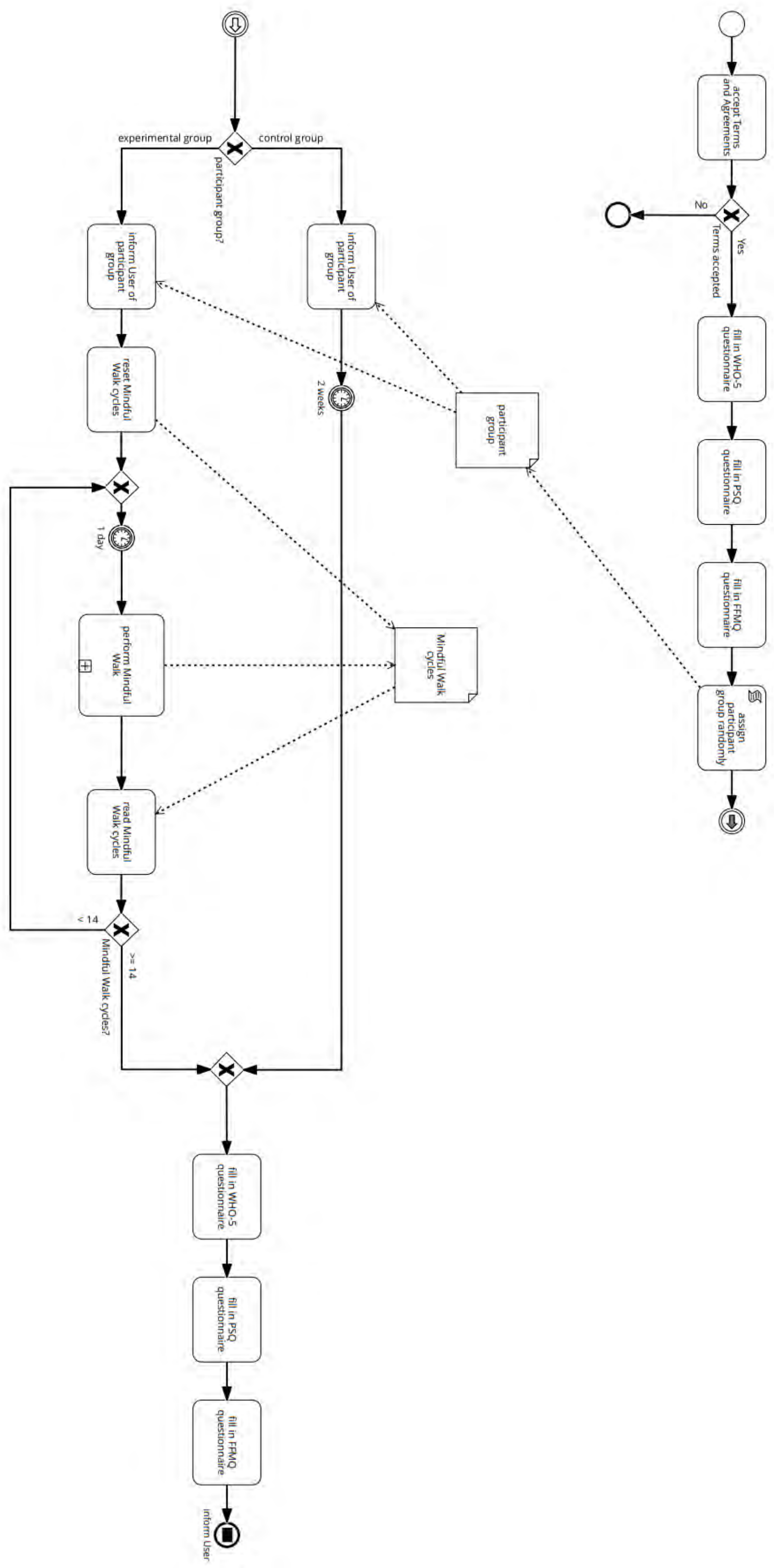

accept Terms and Agreements
Terms accepted
Yes
No
fill in WHO-5 questionnaire
fill in PSQ questionnaire
fill in FFMQ questionnaire
assign participant group randomly
participant group
participant group?
experimental group
control group
inform User of participant group
inform User of participant group
2 weeks
reset Mindful Walk cycles
1 day
perform Mindful Walk
Mindful Walk cycles
read Mindful Walk cycles
< 14
>= 14
Mindful Walk cycles?
fill in WHO-5 questionnaire
fill in PSQ questionnaire
fill in FFMQ questionnaire
inform User

Figure 6.1: BPMN Model of the Mindful Walking Study

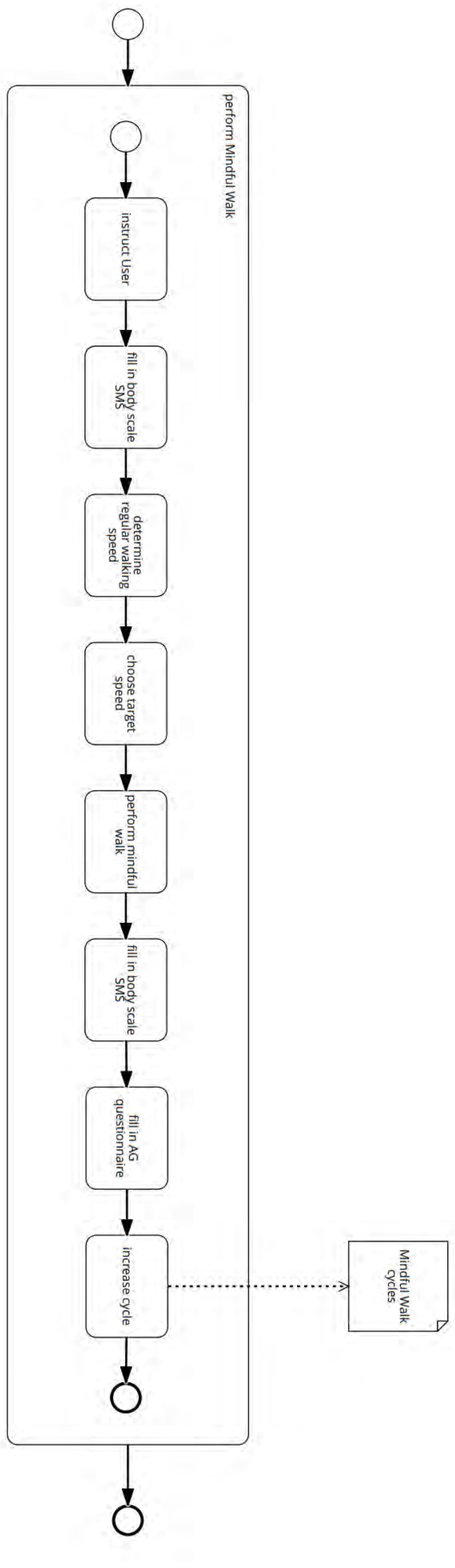


Figure 6.2: Subprocess of a single Mindful Walk action

6.2 Technical Model

The technical model of the Mindful Walking study is the model created in the *QuestionSys Configurator* and based on the process model in chapter 6.1. The JSON export of the *QuestionSys Configurator* is imported into the framework without any further adaptions. The technical model includes the complete functionality of the study as well as all changes that have to be made when transforming the process model to the technical model.

The technical model in Figure 6.3 and Figure 6.4 starts with a XOR-Split Gateway that splits the study into two lanes: first cycle and the second and following cycles.

If one starts the study the first time, one will get into the left line and the study starts with a welcome page. After the participant has been welcomed, another page is shown containing a headline, the WHO-5 questionnaire instruction and the WHO-5 questions. In Figure 6.3 and 6.4 besides one exemplary question, all questions have been removed for clarity. The WHO-5 questionnaire is followed by the PSQ and the FFMQ questionnaires.

After the questionnaires have been filled in, the participant has to choose the participant group. The answer chosen in that question is used as a condition in the following XOR-split. Depending on the answer given, different lanes are followed.

The first lane starts with a custom element that writes the value **"0"** for the meta key **"group"**. In the text node the participant gets an information that one has been assigned to the experimental group and that one should for fourteen days participate each day in a Mindful Walk. The third element is a custom element of type *NOTIFICATION* and the content **"days": 1**.

The participant will receive a notification after one day that another Mindful Walk is available.

The second lane starts with a custom element that writes the value **"1"** for the meta key **"group"**. In the text node the participant gets an information that one has been assigned to the control group and that one should fill in another questionnaire in fourteen days. The third element again is a custom element of the type *NOTIFICATION* and the content **"days": 14**.

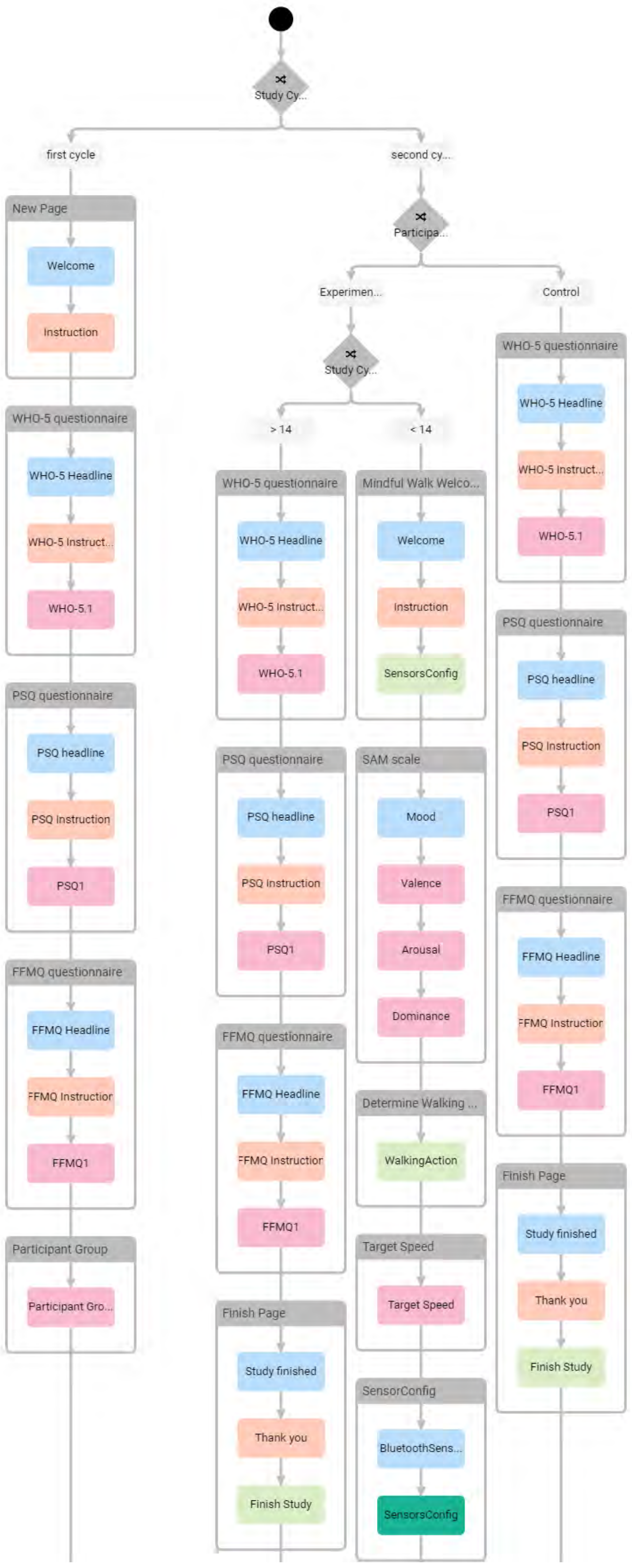


Figure 6.3: Technical model of the Mindful Walking study - Top part

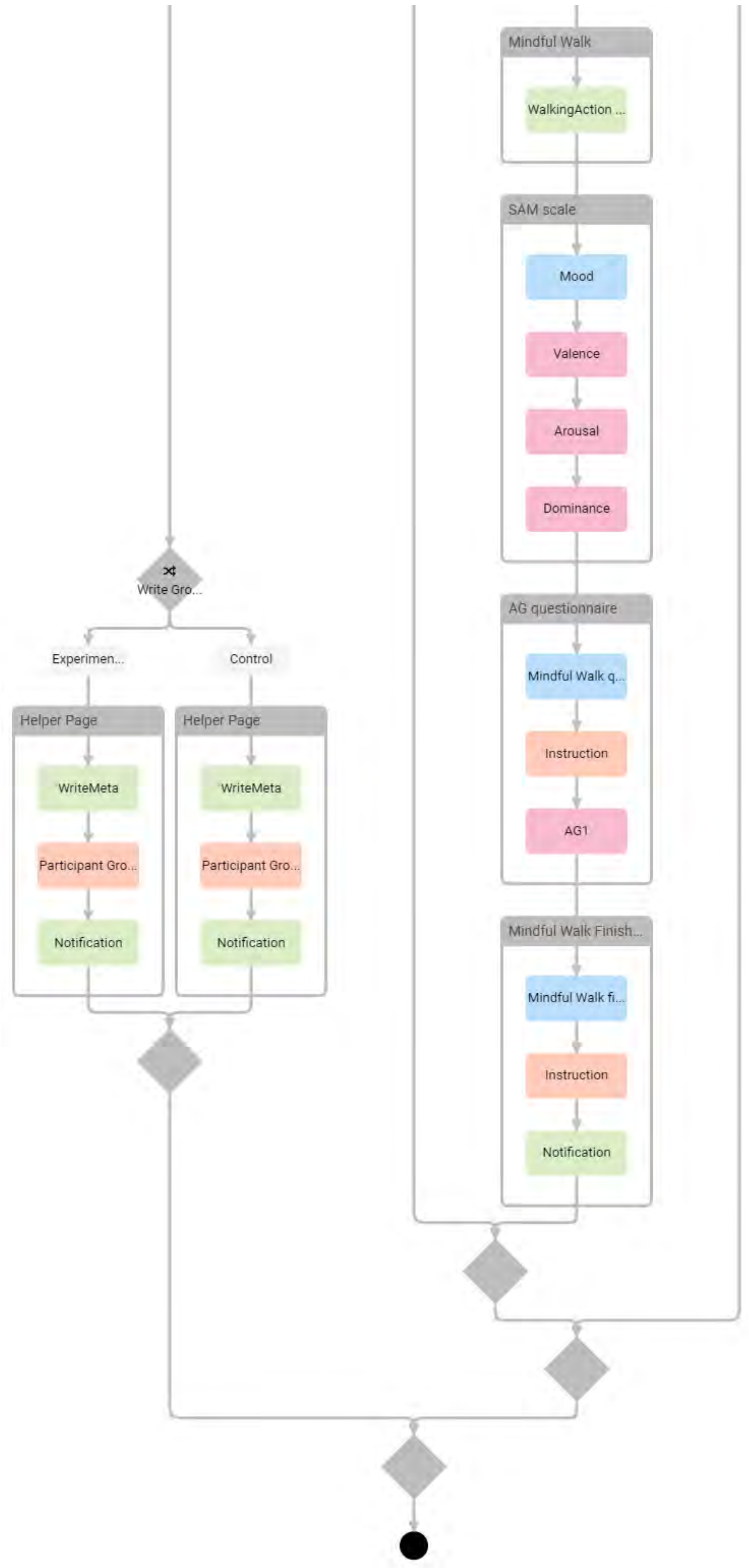


Figure 6.4: Technical model of the Mindful Walking study - Bottom part

The participant will receive a notification after fourteen days that another questionnaire to fill in is available.

Coming from the study entry, the second lane starts with another XOR-Split that splits the lane into the two participant groups: control group (right lane) and experimental group (left lane).

If the participant is assigned to the control group, the line starts with the WHO-5 questionnaire page, the PSQ questionnaire page and the FFMQ questionnaire page. Then a finish page is shown. That page contains a headline, a text element that informs the participant that one has successfully finished the study and finally a custom element that writes the value **"true"** for the meta key **"finished"**. At that point the participation in the Mindful Walking study is fully finished and the participant may not start over again.

If the participant is assigned to the experimental group the lane is split into two lanes by another XOR-Gateway. If one has participated in less than fourteen cycles, then a Mindful Walk starts. That means that first a Mindful Walk Welcome Page is shown. That page contains a headline, a welcome message for the user and a custom element. The custom element is of type *SENSOR_CONFIG* and configures the speed sensor, that is used in the first action element.

The next page is the SAM scale (Self-Assessment Manikin) containing the body scale SAM questions. Then on the following page the first custom element of type *ACTION* is shown. That is the custom element to determine the regular walking speed. Since there is no minimum time limit the participant has to walk, the *countdown* is set to *false*. The complete configuration is shown in Listing 6.3.

```
{
    "actionconfig":
    {
        "headline": {
            "de": "Geschwindigkeit ermitteln",
            "en": "Determine Speed"
        },
        "instruction": {
```

```
        "de": "Bitte laufen Sie in
            Ihrer normalen Gehgeschwindigkeit
            und ermitteln Sie die Geschwindigkeit.",
        "en": "Please walk in your regular Walking Speed
            and determine the speed"
          },
          "countdown": false,
          "seconds": 0
    }
}
```

Listing 6.1: Configuration for the action element that allows the participant to determine the regular walking speed

On the next page one is asked for the target speed. A helper page follows to configure the sensors for the actual Mindful Walk. At that point the bluetooth sensor for the *Heart Rate*, the *Speed Sensor* and the *Distance Sensor* are configured, automatically started and shown in the next action element. The logging for all of the sensors is set to *true* with an interval of three, respective ten seconds.

Also the user notifications for each sensors are configured in the attribute *fluidNotification*. The notification is enabled for the heart rate sensor and the speed sensor and the notification frequency is set to ten or five second. That means one may get notifications for the walking speed every five seconds. For each notification config a maximum and/or a minimum amount is set. For the heart rate one gets a notification if the heart rate drops down below an absolute value of 50 bpm or if the heart rate exceeds an absolute value of 100 bpm.

For the speed sensor a maximum value is set, so that one only gets a notification if the walking speed exceeds the target speed. Therefor the attribute *valueExportNode* is set to *TargetSpeed*, which means that the maximum value is set to the result of the previously set node with the export name *TargetSpeed*.

The complete sensor configuration is shown in Listing 6.2.

```
{
    "sensors":
    [
        {
            "id": 0,
            "name": "Heartrate",
            "sensorType": "EXTERNAL",
            "sensorAttribute": "HEARTRATE",
            "logConfig": {
                "isActivated": true,
                "logFrequency": 3
            },
            "fluidNotification": {
                "isEnabled": true,
                "notificationFrequency": 10,
                "maxValue": {
                    "valueExportNode": null,
                    "value": 100
                },
                "minValue": {
                    "valueExportNode": null,
                    "value": 50
                }
            }
        },
        {
            "id": 1,
            "name": "Walking Speed",
            "sensorType": "INTERNAL",
            "sensorAttribute": "SPEED",
            "logConfig": {
                "isActivated": true,
                "logFrequency": 10
            },
            "fluidNotification": {
                "isEnabled": true,
                "notificationFrequency": 5,
                "maxValue": {
                    "valueExportNode": "TargetSpeed",
                    "value": 0
                },
                "minValue": {
                    "valueExportNode": null,
                    "value": 0
                }
            }
        },
        {
            "id": 2,
            "name": "Distance",
            "sensorType": "INTERNAL",
            "sensorAttribute": "DISTANCE",
            "logConfig": {
                "isActivated": true,
                "logFrequency": 10
            },
            "fluidNotification": {
                "isEnabled": false
            }
        }
    ]
}
```

Listing 6.2: Configuration for the sensors in the Mindful Walk study. The sensor logging is activated for the heart rate sensor, the speed sensor and the distance sensor. Notifications are enabled for the heart rate sensor and the speed sensor.

The next page contains the actual Mindful Walk as a custom element of type *ACTION*. In that action element the participant is instructed for the Mindful Walk. The *countdown* attribute is set to **"true"** and the *"seconds"* attribute is set to **"300"**. That means, that one has to perform at least five minutes of mindful walking. As soon as the Mindful Walk is started, the sensors are started automatically and after the action element has finished, the sensors are stopped automatically.

The complete configuration is shown in Listing 6.2.

```
{
    "actionconfig": {
        "headline": {
            "de": "Mindful Walk",
            "en": "Mindful Walk"
        },
        "instruction": {
            "de": "Bitte klicken Sie auf Start um min.
              5 Minuten Mindful Walk durchzufuehren.
              Die App wird Sie benachrichtigen,
              falls Sie ihre Zielgeschwindigkeit
              ueberschreiten.",
            "en": "Please click on start to start
              a mindful walk.
              The app will notificate you,
              if you exceed the target speed."
        },
        "countdown": true,
            "seconds": 300
        }
}
```

Listing 6.3: Configuration for the action element that allows the participant to determine the regular walking speed

After the Mindful Walk has finished, the SAM Scale questionnaire page is shown another time, then the specific Mindful Walk questionnaire. In the end a finish page is shown. That contains a message for the participant and a custom element that schedules an user notification for the next day. At that point one Mindful Walk has been finished and the participant comes back on the next day.

If one has participated in more than fourteen cycles, then the study is finished in the same way like in the control group lane.

6.3 Challenges

When transforming the business logic (process model) of the Mindful Walking study to the technical model in order to run the study in an app, one can easily see, that the model had to be adapted and functionality had to be added. The following listing points out the challenges that occurred during the implementation of the Mindful Walking study:

1. **Multiple study cycles:** The most obvious difference between the process model and the technical model one can see, is that the process model is only executed once per participant and the cycles of the Mindful Walk sub process are modeled within the process model as loops with timer events. Since for the technical implementation it would not be practical, that the process remains opened over a long period of time, another solution had to be found. In order to allow studies that are executed on different days, a functionality had to be added that the engine can keep track about the study cycles. After every completed study cycle, meta data is saved that contains the count of cycles the study has been executed. Also for every study cycle, the time stamp is saved. Since the study does not complete

if one cycle ends, another meta data had to be added, that contains a flag, that indicates if the study has been fully completed. That attribute had to be set in the end of the study participation. In case of the Mindful Walking study that flag is set after fourteen cycles of execution.

2. **Multiple sensor configurations:** The Mindful Walking study contains two action elements during each study cycle. One is to determine the regular walking speed and the other one is for the actual Mindful Walk. For each action element another set of sensors is needed. The first action element only uses the speed sensor and is configured to not log the sensor values nor notify the participant about the sensor values. The second action element uses the speed sensor, the distance sensor and the heart rate sensor. Logging and notifications are enabled. Therefore two configuration elements had to be added and the second sensor configuration has to overwrite the first sensor configuration.
3. **Scheduled study cycles:** Since the Mindful Walk study needs to be executed on a daily schedule, the study execution has to be blocked if the time gap between two cycles has not passed yet. Therefore the framework saves a time stamp for every passed study cycle.
4. **Meta Data:** Some input is used not only in the current study cycle, but in all cycles of the study. For example the participant group is set once and used in all cycles. Therefor functionality had to be added to save meta attributes.
5. **Conditions based on meta data:** Meta data has to be used as input for conditions. The Mindful Walking study varies depending on the participant group. The functionality for meta conditions had been added and conditions may access meta values by using the prefix *"meta."*.
6. **Scheduled notifications:** After every Mindful Walk in the case study, a notification is scheduled for the next day. Therefore a custom element of type *NOTIFICATION* has been added to schedule notifications.
7. **Accessing data of other nodes in action element:** Within the Mindful Walk sub process the user is notified if the target speed is exceeded. The target speed has been set in a previously shown question element. To access the user input

within the sensor config an attribute *valueExportNode* had to be added that links to the result of another node.

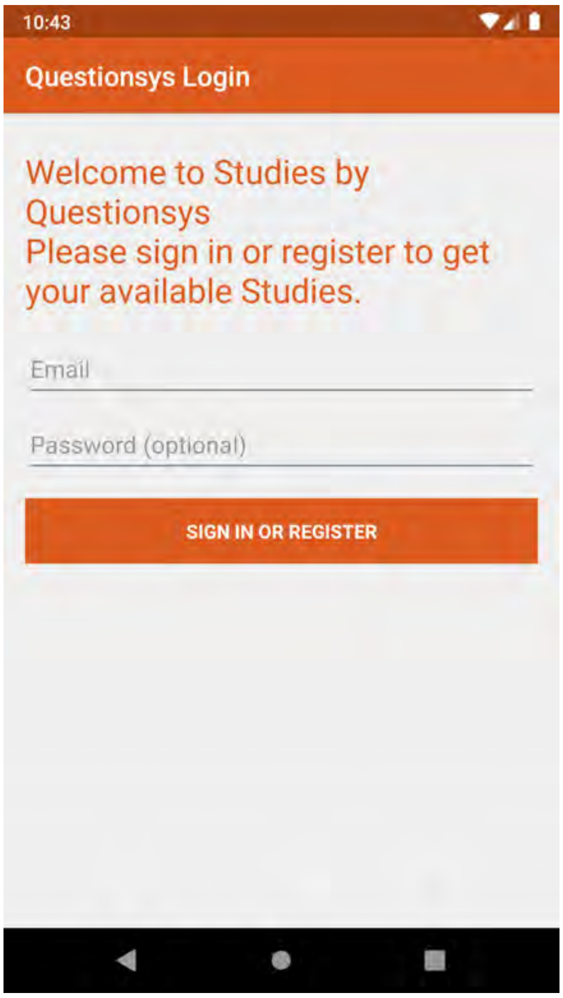
7

Presentation of the mobile application

This chapter introduces the application and shows which UI elements the questionnaires support as well as how the case study Mindful Walking has been implemented.

7.1 Demo App

This section shows how an app that embeds the QuestionSys studies framework may be developed. For demonstration purpose a demo app with basic functionality has been implemented. This implementation may be used as a template for future applications. The app starts with a login screen. The user may login or register for an account by typing in the user name and the password. Figure 7.1 shows the login screen.



Welcome to Studies by Questionsys
Please sign in or register to get your available Studies.

Figure 7.1: Login Screen

The demo app brings a sliding menu from that the different features can be selected. To configure and connect the external bluetooth sensors there is an external sensor setting menu that can be seen in Figure 7.2. For each supported sensor type there is a list item that can be tapped to search for a bluetooth device and connect it.

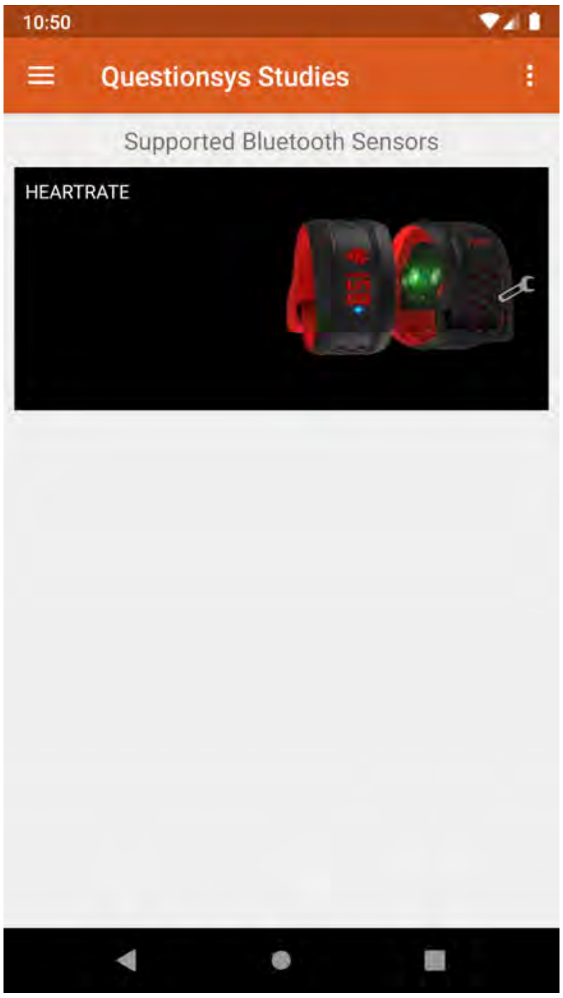


Figure 7.2: External Sensor settings

The study list in Figure 7.3 shows all studies that are available for the user. Tapping on a study loads and starts a cycle of the study.

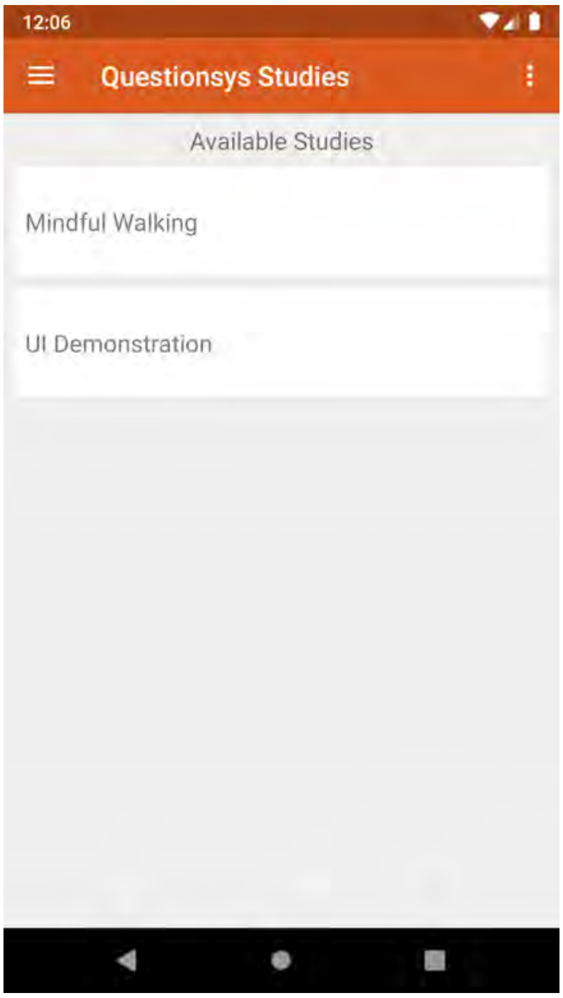


Figure 7.3: Study List

7.2 Mindful Walking

This section presents the UI of the case study Mindful Walking that has been implemented.

When the study has been selected, a start screen with a play button is shown (Figure 7.4). Clicking on the play button starts the study cycle.

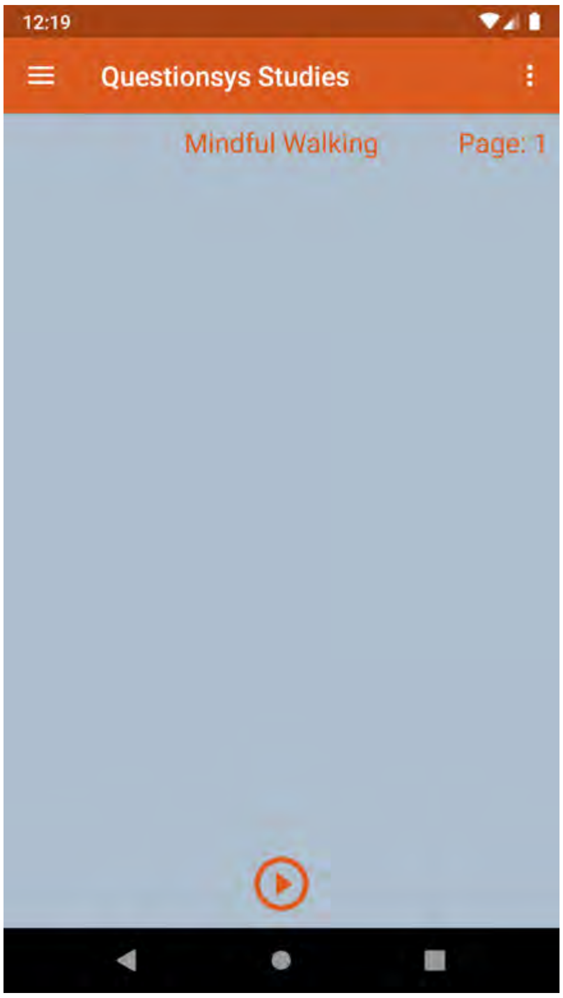
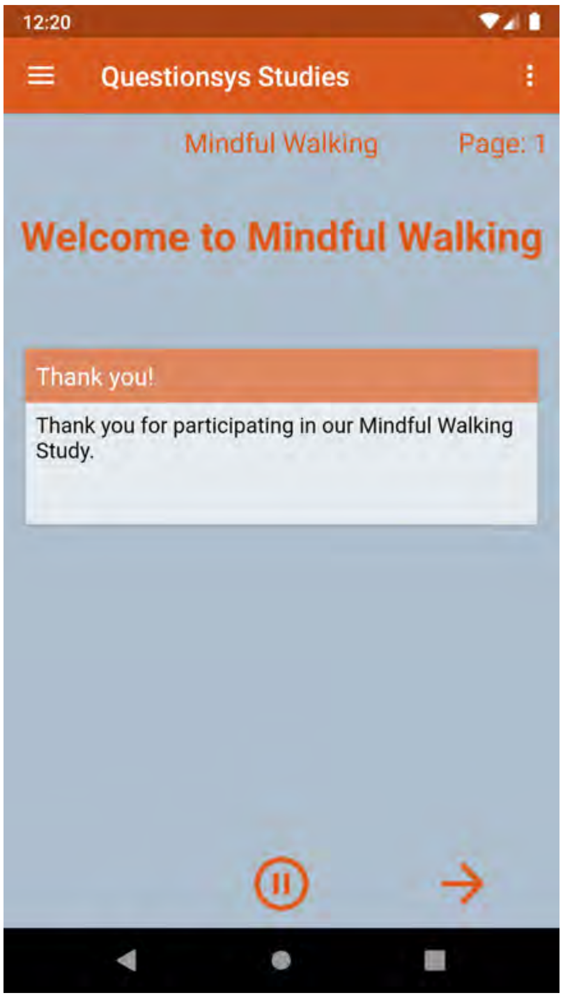


Figure 7.4: Start Screen after the a study was selected

The study begins with a welcome screen containing a headline and a text element thanking the user for the participation. The welcome screen can be seen in Figure 7.5.



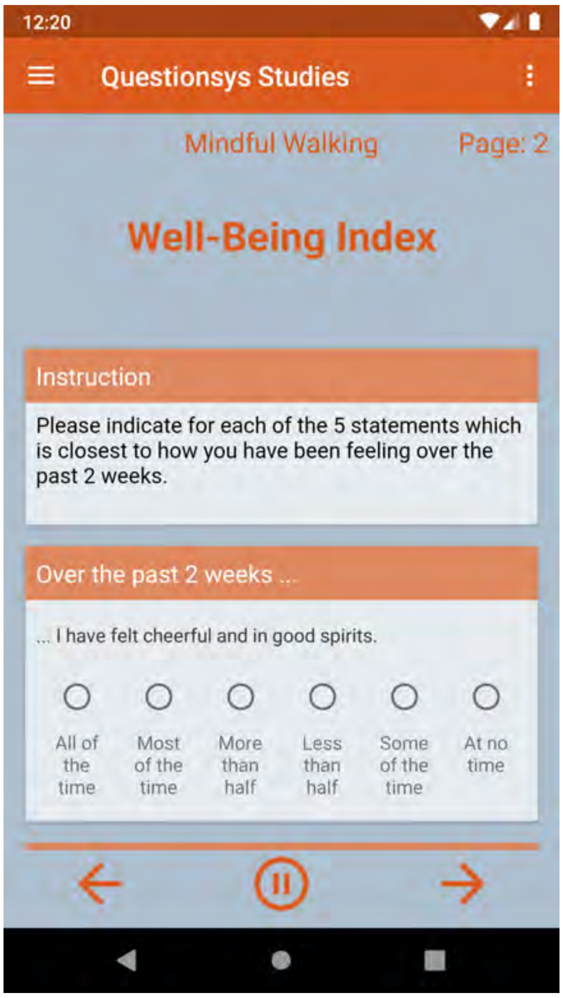
Welcome to Mindful Walking

Thank you!

Thank you for participating in our Mindful Walking Study.

Figure 7.5: Mindful Walking study: Welcome Screen

Figure 7.6 shows the Well-Being Index questionnaire that is shown to the user in the first study cycle. The page consists of a headline, an instruction and the questions. Each question of the questionnaire contains a likert scale, so the user has to select the answer that fits best from the given options.



Well-Being Index

Instruction

Please indicate for each of the 5 statements which is closest to how you have been feeling over the past 2 weeks.

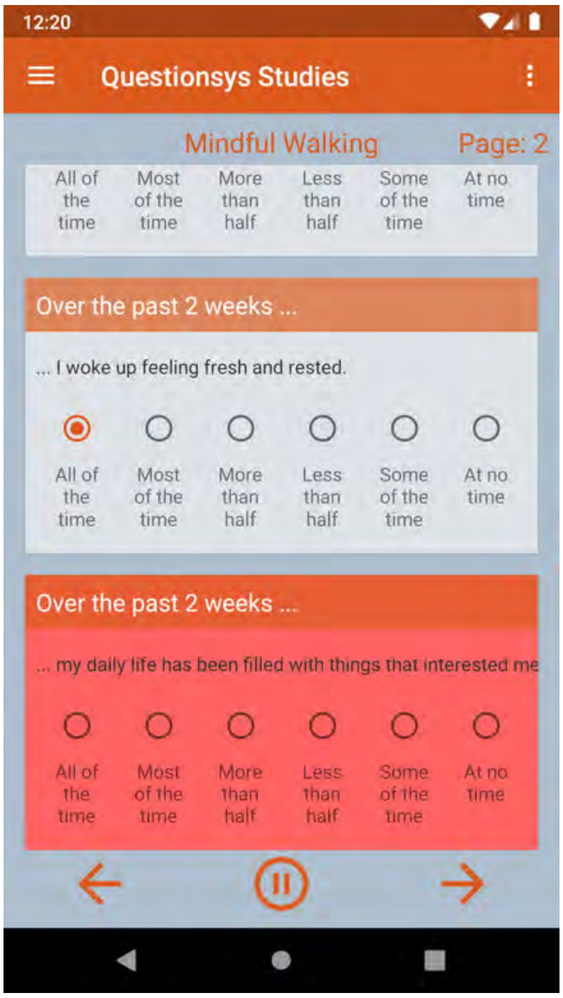
Over the past 2 weeks ...

... I have felt cheerful and in good spirits.

○	○	○	○	○	○
All of the time	Most of the time	More than half	Less than half	Some of the time	At no time

Figure 7.6: Mindful Walking study: Well-Being Index questionnaire as Likert Scale

The user has to completely fill in the questionnaire. If a question has not been filled in and the user wants to go to the next page, the question becomes highlighted, as shown in Figure 7.7.


12:20

Questionsys Studies

Mindful Walking Page: 2

All of the time	Most of the time	More than half	Less than half	Some of the time	At no time

Over the past 2 weeks ...

... I woke up feeling fresh and rested.

[✓ All of the time] Most of the time More than half Less than half Some of the time At no time

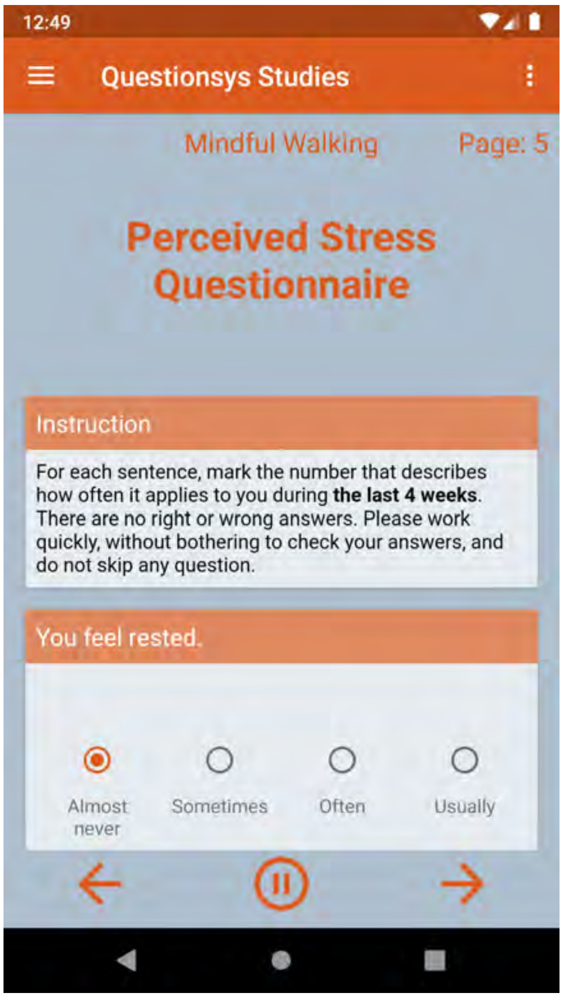
Over the past 2 weeks ...

... my daily life has been filled with things that interested me

All of the time Most of the time More than half Less than half Some of the time At no time

Figure 7.7: Mindful Walking study: Incorrectly or incompletely filled in questionnaire elements are highlighted

Two more questionnaires follow (Figure 7.8), then the user has to select the participant group in Figure 7.9.



Perceived Stress Questionnaire

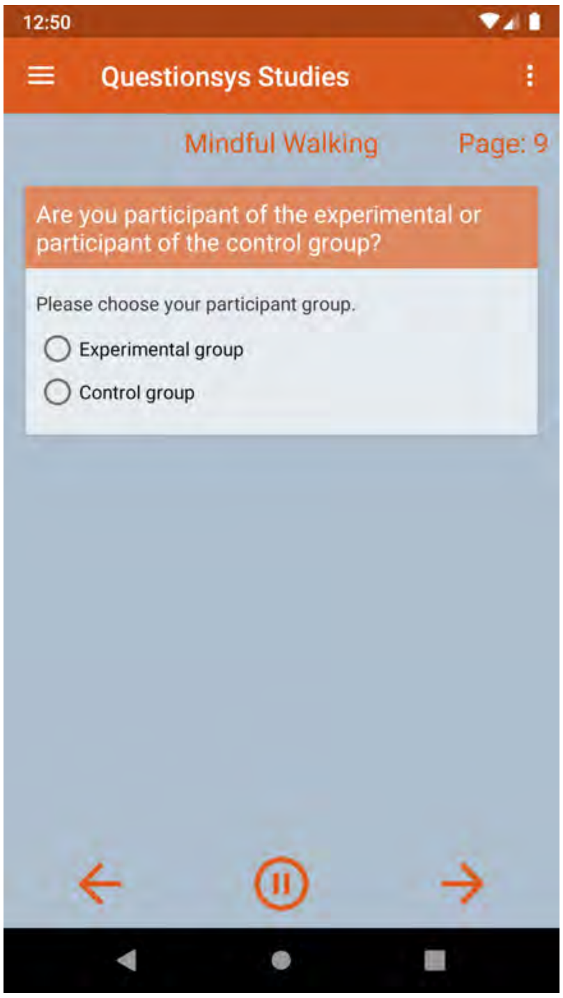
Instruction

For each sentence, mark the number that describes how often it applies to you during **the last 4 weeks**. There are no right or wrong answers. Please work quickly, without bothering to check your answers, and do not skip any question.

You feel rested.

[✓ Almost never] Sometimes Often Usually

Figure 7.8: Mindful Walking study: Perceived Stress Questionnaire



Are you participant of the experimental or participant of the control group?

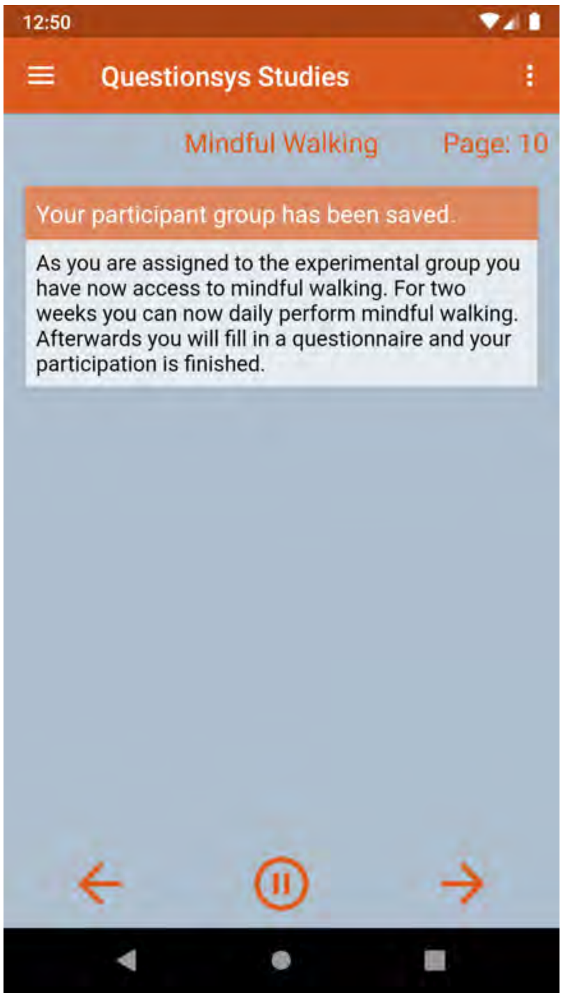
Please choose your participant group.

◯ Experimental group

◯ Control group

Figure 7.9: Mindful Walking study: Control group selection

After the selection, a confirmation is shown (Figure 7.10) and based on the selection, the content of the next study cycles differ.

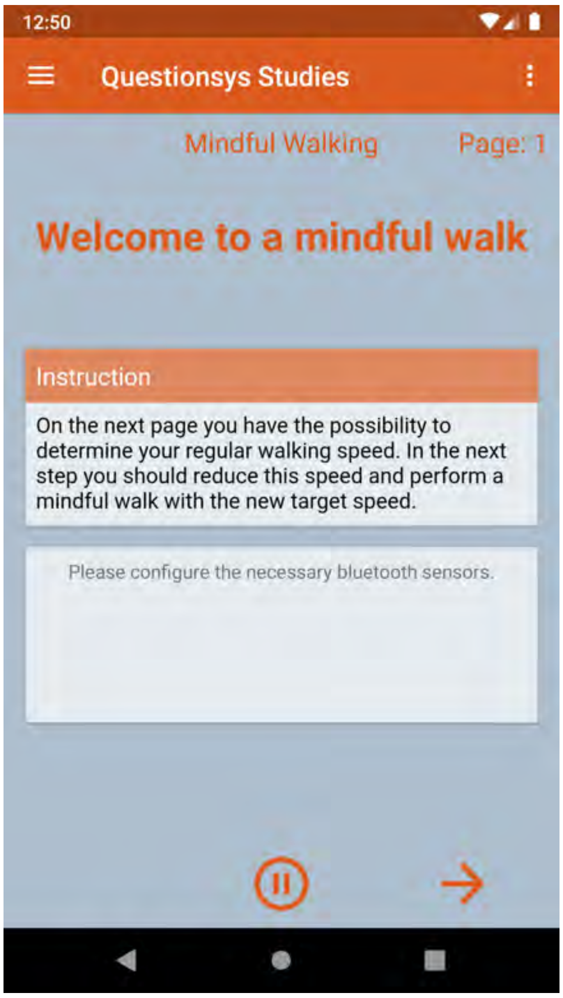


Your participant group has been saved.

As you are assigned to the experimental group you have now access to mindful walking. For two weeks you can now daily perform mindful walking. Afterwards you will fill in a questionnaire and your participation is finished.

Figure 7.10: Mindful Walking study: Control group selection confirmation

In the experimental group the next study cycle contains a Mindful Walk session. Figure 7.11 shows the welcome screen and instruction for the Mindful Walk.


12:50

Questionsys Studies

Mindful Walking Page: 1

Welcome to a mindful walk

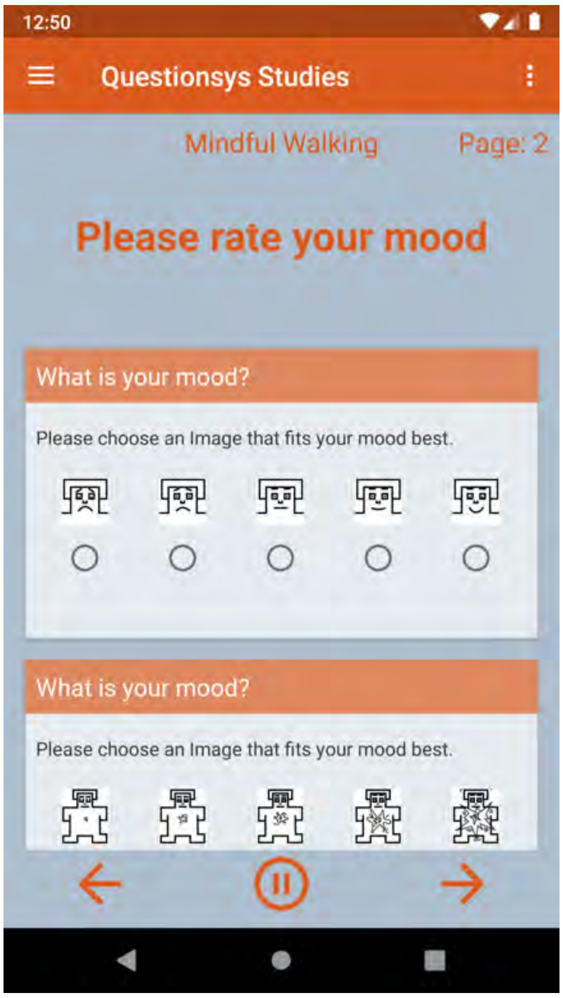
Instruction

On the next page you have the possibility to determine your regular walking speed. In the next step you should reduce this speed and perform a mindful walk with the new target speed.

Please configure the necessary bluetooth sensors.

Figure 7.11: Mindful Walking study: Welcome screen for the daily Mindful Walk

First, some questionnaires that contain Likert scales with images are shown. An example can be seen in Figure 7.12.


12:50

Questionsys Studies

Mindful Walking Page: 2

Please rate your mood

What is your mood?

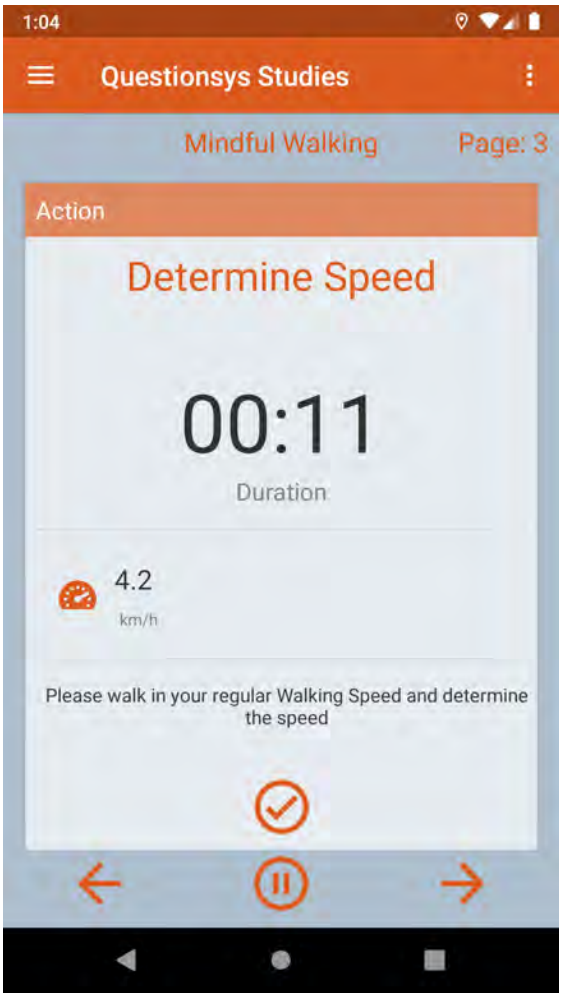
Please choose an Image that fits your mood best.

What is your mood?

Please choose an Image that fits your mood best.

Figure 7.12: Mindful Walking study: Mood questionnaire as a Liker Scale with images

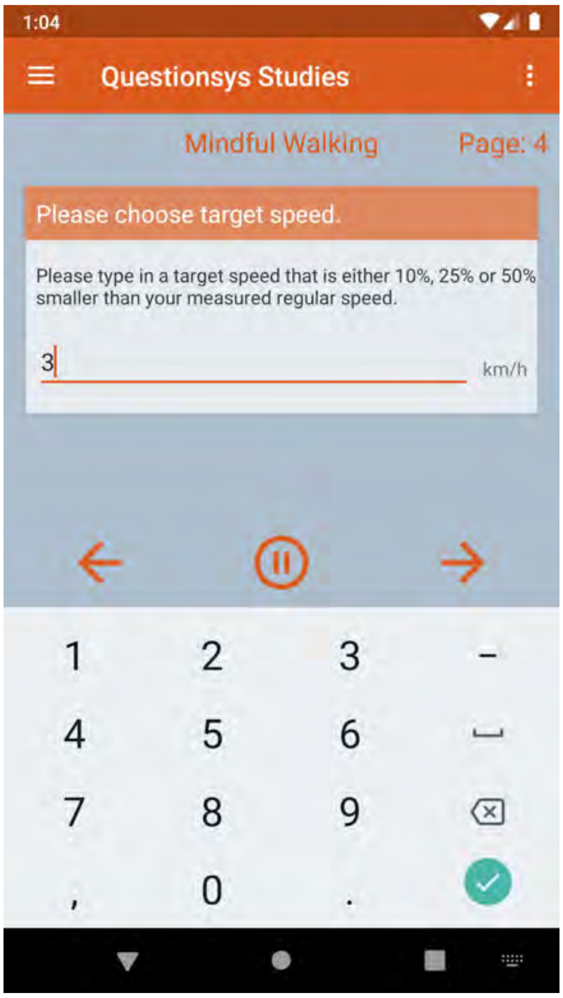
Second, an action element is shown (Figure 7.13), allowing the user to determine the regular walking speed. For that action view only the internal speed sensor is enabled and logging and notifications are disabled.



Please walk in your regular Walking Speed and determine the speed

Figure 7.13: Mindful Walking study: Regular Walking speed determination as an action view

In Figure 7.14 the user has to select the target speed for the Mindful Walk. That target speed is used as notification threshold for the followed Mindful Walk.



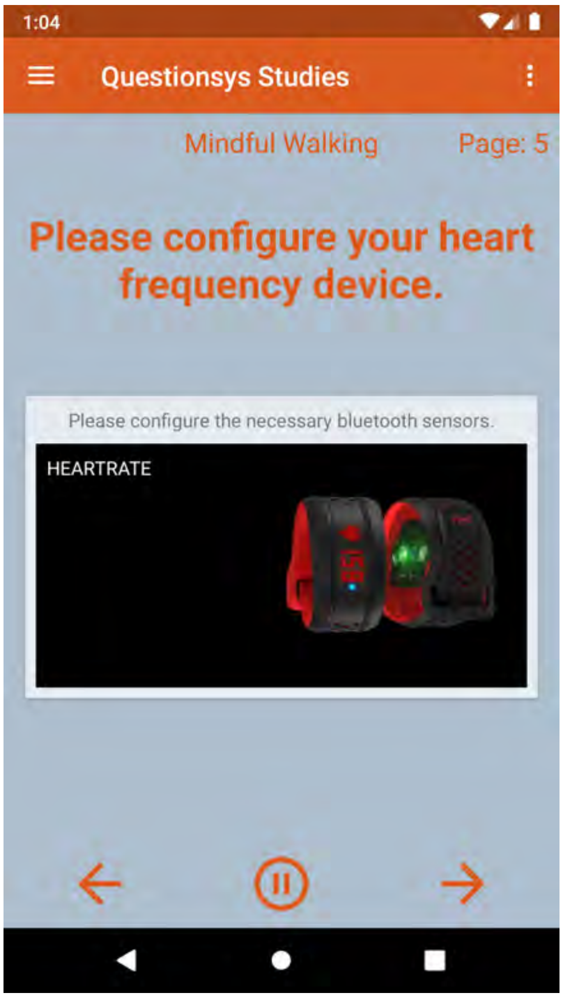
Please choose target speed.

Please type in a target speed that is either 10%, 25% or 50% smaller than your measured regular speed.

3 km/h

Figure 7.14: Mindful Walking study: Selecting the target speed for the Mindful Walk

Figure 7.15 shows the sensor configuration for the Mindful Walk. The page allows the user to reconfigure the bluetooth sensors that are required for the study. In this case only the heart rate sensor is required.



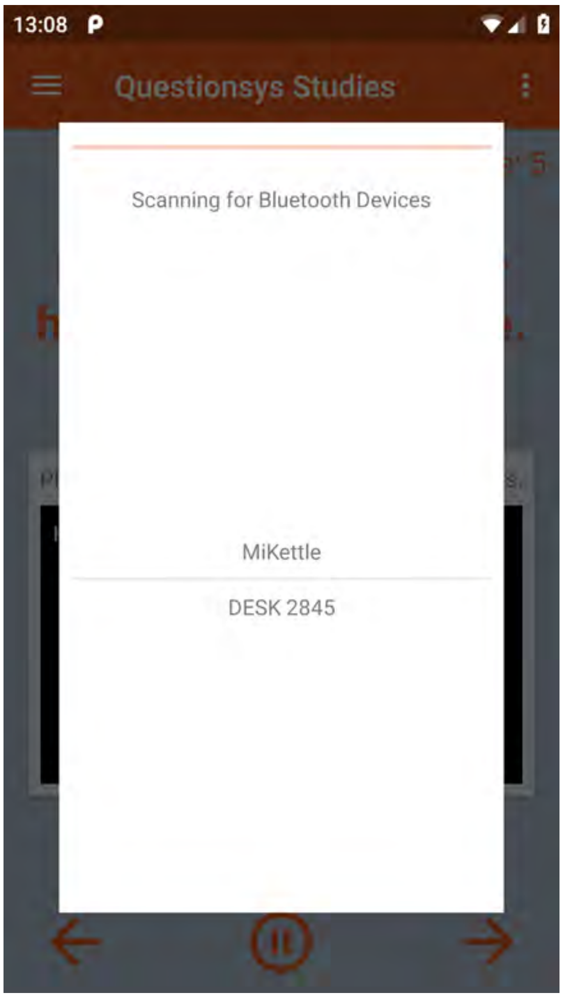
Please configure your heart frequency device.

Please configure the necessary bluetooth sensors.

HEARTRATE

Figure 7.15: Mindful Walking study: Sensor configuration during a study

Clicking on the heart rate sensor opens a dialog (Figure 7.16) in which the sensor can be selected.



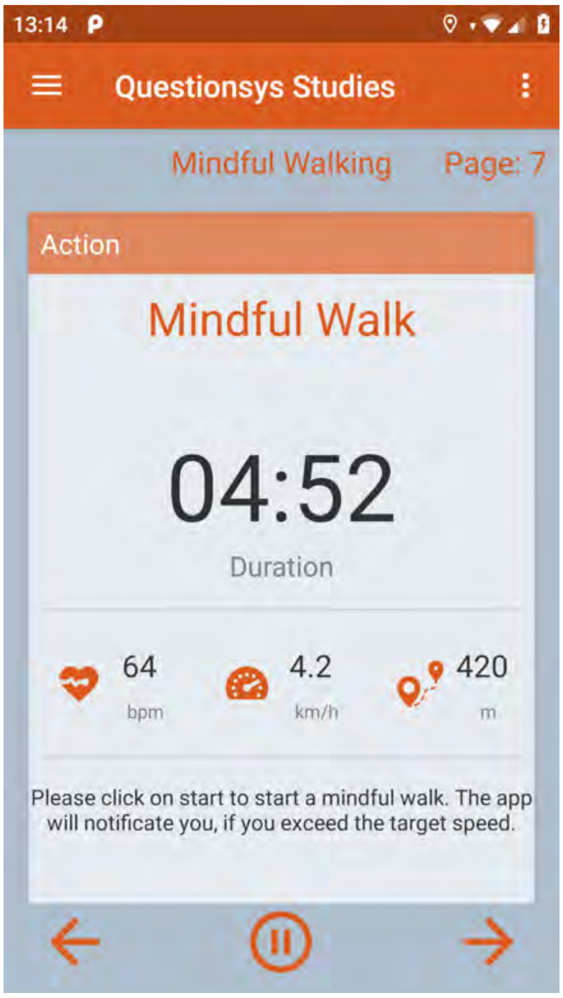
Scanning for Bluetooth Devices

MiKettle

DESK 2845

Figure 7.16: Mindful Walking study: Bluetooth device selection screen for connecting external sensors

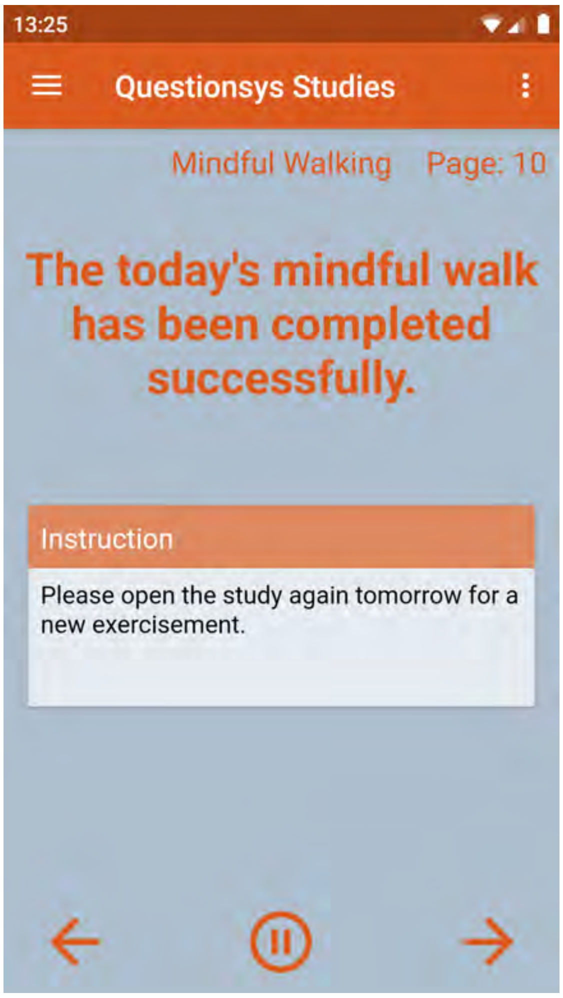
Figure 7.17 shows the actual Mindful Walk. The action element contains a headline, a countdown, a list of the activated sensors and the current sensor readings as well as an instruction. The Mindful Walk is configured to run for at least five minutes, therefor the countdown is set to five minutes and the page cannot be switched before the countdown runs out.



Please click on start to start a mindful walk. The app will notificate you, if you exceed the target speed.

Figure 7.17: Mindful Walking study: Performing a Mindful Walk

Figure 7.18 shows the last screen of a Mindful Walk session.


13:25

Questionsys Studies

Mindful Walking Page: 10

The today's mindful walk has been completed successfully.

Instruction

Please open the study again tomorrow for a new exercisement.

Figure 7.18: Mindful Walking study: Completion of a daily mindful walk

7.3 Additional question types

In this section additional question types that are supported by the framework are shown. These question types have not to be used in the Mindful Walking study.

Figure 7.19 shows demos for a drop down question, a multiple choice question and a single choice question.

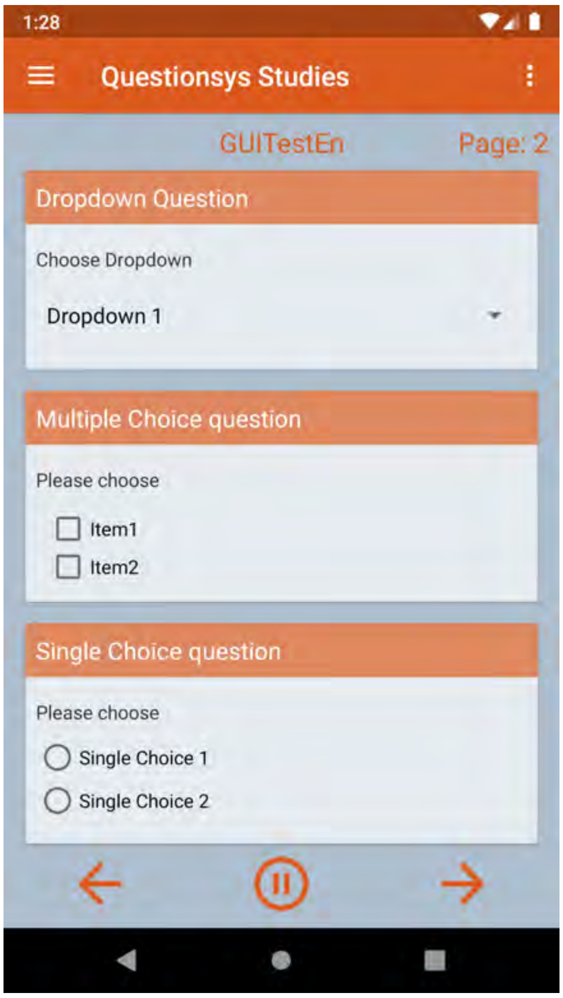


Figure 7.19: Dropdown, multiple choice and single choice question UI elements

In Figure 7.20 there is a Yes/No question, a date selection and a float number input.

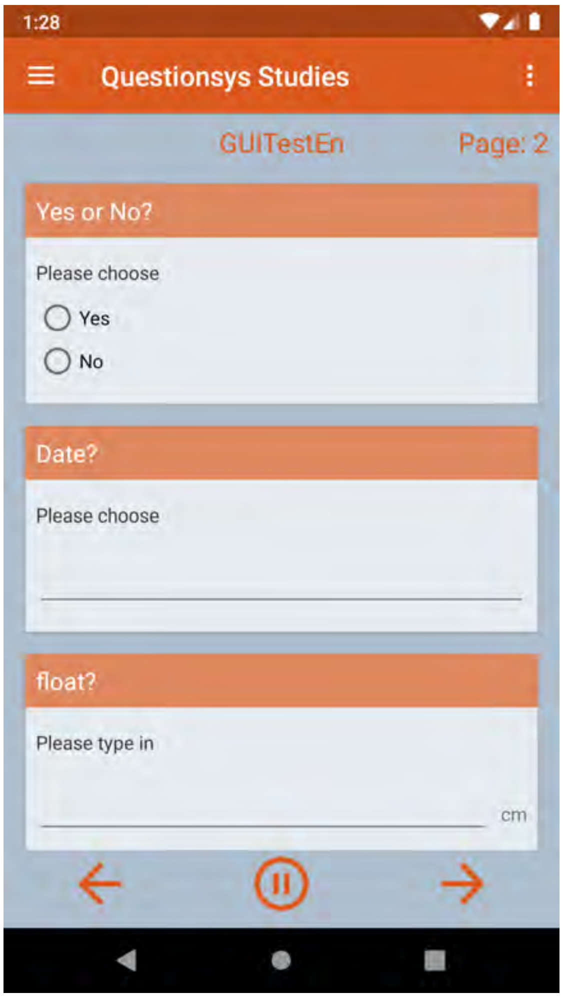


Figure 7.20: YesNo, Date and number input question UI elements

The framework also supports integer input, single line text input and multi line text input. The date selection can be seen in Figure 7.21.

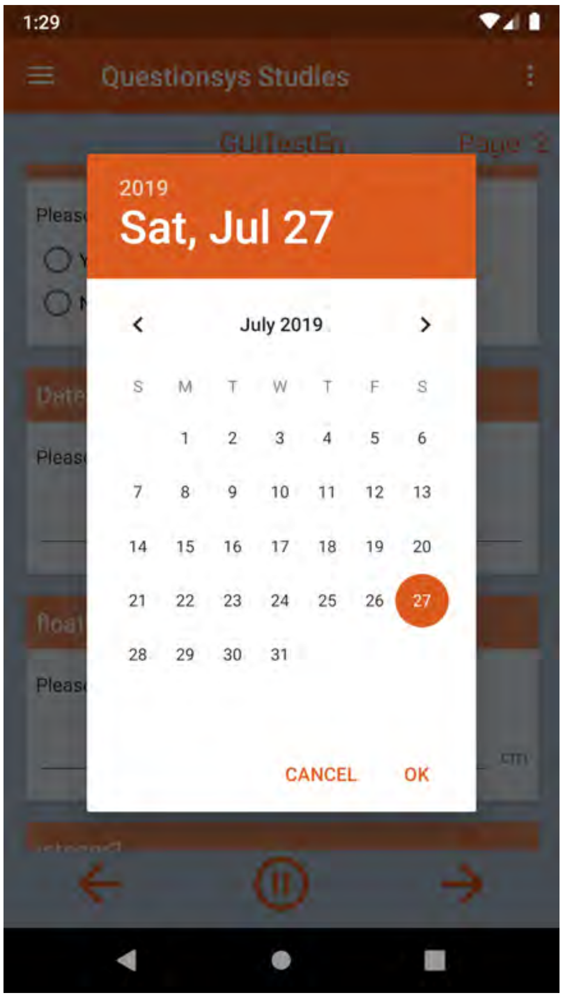


Figure 7.21: Date input popup window

Figure 7.22 shows a Slider range question and a single slider question.

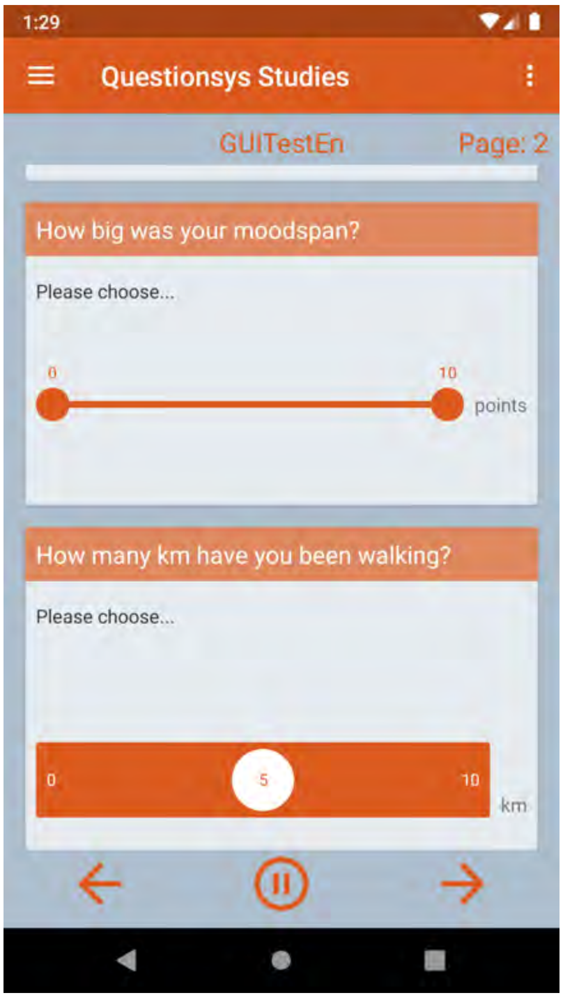


Figure 7.22: Single slider and range slider question UI elements

8 Requirements Check

In this chapter, it is shown whether the previously defined requirements were fulfilled.

8.1 Functional requirements

1. **QuestionSys as interface for generic studies:** The requirement has been fulfilled. The studies are created in the QuestionSys Configurator and additional features are covered by custom elements. The JSON output of the study exported from the QuestionSys Configurator is used as input for the framework without any additional changes.

2. **Multiple session studies:** The requirement has been fulfilled. The framework supports multiple cycles for each study. A study may have one or more cycles and data may be transported from cycles to cycles using meta data. Every *SaveData* Object has an attribute that assigns it to a specific session.

3. **Meta data:** The requirement has been fulfilled. The framework supports saving arbitrary meta values that can be used to save instance comprehensive data. For example the meta data object *finished* is used as a flag to save if the study has been fully completed and if there are no more cycles available.

4. **Multiple studies:** The requirement has been fulfilled. The framework supports multiple studies. Studies are transferred to the framework as a string in the JSON format and then cast to the internal models.

5. **Elements:** The requirement has been fulfilled. The framework supports all previously defined elements like headlines, text elements, media elements, question elements and custom elements.
6. **Questions:** The requirement has been fulfilled. The framework supports and provides UI elements for drop down questions, multiple choice questions, single choice questions, yes/no questions, date input, single line text input, multi line text input, likert scales, likert scales with images, single sliders and slider ranges.
7. **Logic elements:** The requirement has been fulfilled. The framework supports logic elements like XOR split and join elements. XOR split elements split the questionnaire into multiple paths based on conditions that access either node result data or meta data input.
8. **Custom elements:** The requirement has been fulfilled. The framework supports the custom elements *WRITE_META*, *READ_META SENSOR_CONFIG*, *SENSOR_START*, *SENSOR_STOP* and *ACTION*.
9. **Action elements:** The requirement has been fulfilled. The framework supports action elements. Action elements may contain a countdown timer, a chronometer, a view to show sensor values and an instruction for the user.
10. **Sensors :** The requirement has been fulfilled. The framework supports smartphone internal sensors as well as external bluetooth sensors that support the GATT protocol.
11. **Sensor Notifications:** The requirement has been fulfilled. The framework supports fluid notifications. That means, while the sensors are enabled the user constantly receives notifications if the sensor values are not within the configured limits.
12. **Pausable:** The requirement has been fulfilled. A study may be paused at any time and input data remains persisted.
13. **Notifications:** The requirement has not been fulfilled yet, since this thesis focuses on the architecture of the framework. User notifications may be added to a future

app that embeds the *QuestionSys studies* framework. However, the input data form of the notification settings has already been prepared.

14. **Results:** The requirement has partly been fulfilled. The framework supports exporting study results in JSON form after the study has been finished. However, since the *QuestionSys* API at this point has not been finished yet, the result data can not be uploaded yet.
15. **Statistics:** The requirement has not been fulfilled yet, since this thesis focuses on the architecture of the framework. Statistics and user feedback may be added to a future app version.
16. **Log-in:** The requirement has partly been fulfilled. The app sample app provides a log-in screen. However, since the *QuestionSys* API at this point has not been finished yet, the log-in function has to be added in the future.
17. **Multi-language support:** The requirement has been fulfilled. User Interfaces have been exemplary translated into English and German. Also, questionnaires can be translated into different languages using the *QuestionSys Configurator*. The framework fully supports loading different languages for study content.

8.2 Non-functional requirements

1. **Generic:** The requirement has been fulfilled. All kind of studies can be created in the QuestionSys Configurator and imported in the framework.
2. **Android compatible:** The requirement has been fulfilled. The framework has been implemented for Android.
3. **Well structured input data:** The requirement has been fulfilled. As input data for the framework, the export data from the *QuestionSys Configurator* is used, which is well formatted.
4. **Well structured output data:** The requirement has been fulfilled. The results data is of type *JSON* and well structured.

5. **Loose coupling:** The requirement has been fulfilled. The framework can easily be embedded into other applications.
6. **Modularity:** The requirement has been fulfilled. The framework is modular and parts of the framework like the UI can be exchanged easily.
7. **Extensibility:** The requirement has been fulfilled. The framework can be extended by new functionality using the custom element. New bluetooth sensors may be added by extending the *BluetoothSensor* class and defining the GATT attribute.
8. **Offline mode:** The requirement has been fulfilled. Studies are saved and loaded locally as well as media objects are cached in the local storage. Result data is locally saved in the realm database.
9. **App feeling:** The requirement has been fulfilled. The app is usable by all kind of users and provides feedback and instructions for the user. For instance, questions are marked red if the input data is incorrect or missing. Sensors may even be configured during the study if they have not been connected yet.

9 Conclusion & Prospects

The objective of this master's thesis was to build a generic framework, that connects to the *QuestionSys* backend and runs a process engine that is capable rendering a complete clinical study in form of a business process. Therefore the custom elements in the *QuestionSys Configurator* have been used to extend the functionality and to meet the requirements of an interventional clinical study.

Main requirements have been the integration of action elements as a template for clinical exercises. Also internal as well as external sensors had to be integrated. During the execution of action elements, the sensor values have to be logged and feedback has to be given to the user.

In scope of this thesis the above described framework has been developed for Android. The framework renders studies that consist of various study pages. Every page may have questions or action elements and can be rendered by the framework.

Sensor integration as well as sensor logging and user notifications have been implemented.

A layout for the questionnaire elements has been developed and the framework renders various UI elements depending on the node and question type.

A persistence manager has been implemented that is responsible to persist questionnaire answers and sensor logs.

To investigate if the designed concept and architecture work, a case study *Mindful Walking* has been implemented to show how a clinical study may be created for the framework. For the study *Mindful Walking* some challenges had to be overcome.

First, *Mindful Walking* is a study that is designed to be executed for a longer period of time. That's why there had to be a function to the framework added to save meta data in order to transport data from one study instance to another. Second, during each *Mindful Walking* instance, there are two exercises for the user. The first exercise is to measure usual walking speed and the second is the actual Mindful Walk. Since for those two exercises two different sets of sensors have to be used, functionality had to be added to overwrite the sensor configuration at any time of the study.

The above mentioned challenges have been solved by making various changes in the framework and adding functionality in the custom elements. The creation of the *Mindful Walking* study took around fifteen hours, so the effort creating a study with the *QuestionSys Configurator* and the framework can be significantly lower than creating an individual app for each new study.

9.1 Prospects

In order to deploy the application on Android devices and advertise it, some additional changes have to be made and some additional features may be added:

- In context of this thesis a demonstration application has been implemented that embeds the framework. However, since the *QuestionSys* backend system at the time of the thesis has not fully been finished, some adaptions have to be made for the app. First, the log-in page should be filled with functionality and the user and password should be sent to the backend to authenticate the user. Also a registration function for new users should be added. Second, at this time the studies used in the app are added as assets to the app. To publish the app, a feature needs to be added to download the studies that the user is assigned to. The feature to list the studies has already been prepared, but also needs to be adapted. There is already a function implemented to export study results and sensor logs to a JSON file and send as mail to the reviewer, but in best case the study results should be uploaded to the *QuestionSys* backend automatically, therefor the results export should be adapted.

- To motivate the user to regularly participate in the studies, a function should be added to give feedback and show statistics on the study progress. There are many ways to design statistics, one of them would be to add a statistic that compares sensor values of exercises of each study cycle. In the case study *Mindful Walking* the user would then see how the quality of the Mindful Walks improve each time and the heart rate may reduce in each walk. Another way would be to compare results of likert scales in each study cycle. In the case study *Mindful Walking* the user would then see how the stress level becomes lower each day of participation.
- To allow doctors and psychologists to design even more types of studies besides the support of bluetooth heart rate devices, support for more types of bluetooth sensors may be added. As described in section 5.5 sensors can be added fairly easy to the framework if they support the Bluetooth Low Energy protocol.
- Since notifications are an app specific function, when embedding the framework into an Android application, the support for user notification needs to be added. This allows to remind the user of new studies that can be downloaded or to remind the user if one should participate in another cycle of the study. The preparations in the framework to set notifications for a study have already been made and the study creator may add a custom element that sets the count of days until the notification should be sent to the user.
- The *QuestionSys* backend may be adapted to allow the upload of the study results. Especially for uploading the sensor log data, some changes may have to be made. In best case, the backend should add functionality to create different types of statistics using both study result data and sensor log data.
- Since for the framework some new types of custom elements have been added to the input data, the *QuestionSys Configurator* might be adapted to allow a more user friendly study creation.

Bibliography

[1] Joachim, J.: Entwicklung neuer Anwendungsszenarien für ein prozessorientiertes Fragebogensystem. (2017)

[2] Frank, J.: A personalized support tool for the training of mindful walking: The mobile "MindfulWalk" application. (2017)

[3] Martin, R.: Developing a Complex User Interface for Mobile Data Collection Applications. (2018)

[4] Malsam, F.: Konzeption, Implementierung und Evaluation eines Rahmenwerks zur Auslesung der Herzfrequenz durch Fitnesstracker in Android. (2018)

[5] Müller, D.: Conception and implementation of a mobile application to conduct a Mindful Walking Study regarding Clinical Psychology. (2019)

[6] Schobel, J., Ruf-Leuschner, M., Pryss, R., Reichert, M., Schickler, M., Schauer, M., Weierstall, R., Isele, D., Nandi, C., Elbert, T.: A generic questionnaire framework supporting psychological studies with smartphone technologies. In: XIII Congress of European Society of Traumatic Stress Studies (ESTSS) Conference. (2013) 69–69

[7] Schobel, J., Schickler, M., Pryss, R., Maier, F., Reichert, M.: Towards Process-Driven Mobile Data Collection Applications: Requirements, Challenges, Lessons Learned. In: 10th Int'l Conference on Web Information Systems and Technologies (WEBIST 2014), Special Session on Business Apps. (2014) 371–382

[8] Schobel, J., Schickler, M., Pryss, R., Reichert, M.: Process-Driven Data Collection with Smart Mobile Devices. In: 10th International Conference on Web Information Systems and Technologies (Revised Selected Papers). Number 226 in LNBIP. Springer (2015) 347–362

[9] Schobel, J., Pryss, R., Reichert, M.: Using Smart Mobile Devices for Collecting Structured Data in Clinical Trials: Results From a Large-Scale Case Study. In: 28th IEEE International Symposium on Computer-Based Medical Systems (CBMS 2015), IEEE Computer Society Press (2015) 13–18

[10] Schobel, J., Pryss, R., Schickler, M., Reichert, M.: Process-Driven Mobile Data Collection (Extended Abstract). In: 8th International Workshop on Enterprise Modeling and Information Systems Architectures (EMISA 2017). (2017)

[11] Schobel, J.: A Model-Driven Framework for Enabling Flexible and Robust Mobile Data Collection Applications. (2018)

[12] Pryss, R., Reichert, M., John, D., Frank, J., Schlee, W., Probst, T.: A Personalized Sensor Support Tool for the Training of Mindful Walking. In: IEEE 15th International Conference on Wearable and Implantable Body Sensor Networks (BSN 2018). (2018) 114–117

[13] Mehdi, M., Mühlmeier, G., Agrawal, K., Pryss, R., Reichert, M., Hauck, F.: Referenceable mobile crowdsensing architecture: A healthcare use case. In: 1st International Workshop On Services For Mobile Data Collection. (2018)

[14] Jimenez-Ramirez, A., Barba, I., Reichert, M., Weber, B., Valle, C.D.: Clinical Processes - The Killer Application for Constraint-Based Process Interactions? In: 30th Int'l Conference on Advanced Information Systems Engineering (CAiSE 2018). Number 10816 in LNCS, Springer (2018) 374–390

[15] Hoppenstedt, B., Pryss, R., Kammerer, K., Reichert, M.: CONSENSORS: A Neural Network Framework for Sensor Data Analysis. In: 26th International Conference on COOPERATIVE INFORMATION SYSTEMS (CoopIS 2018)). LNCS, Speinger (2018)

[16] Schobel, J., Probst, T., Reichert, M., Schickler, M., Pryss, R.: Enabling Sophisticated Lifecycle Support for Mobile Healthcare Data Collection Applications. IEEE Access **7** (2019) 61204–61217

[17] Schobel, J., Pryss, R., Probst, T., Schlee, W., Schickler, M., Reichert, M.: Learnability of a Configurator Empowering End Users to Create Mobile Data Collection Instruments: Usability Study. JMIR mHhealth and uHealth **6** (2018) e148

[18] Michalak, J., Zarbock, G., Drews, M., Otto, D., Mertens, D., Strähle, G., Schwinger, M., Dahme, B., Heidenreich, T.: Erfassung von Achtsamkeit mit der deutschen Version des Five Facet Mindfulness Questionnaires (FFMQ-D). Zeitschrift für Gesundheitspsychologie **24** (2016) 1–12

[19] Kinnamon, D., Ghanta, R., Lin, K.C., Muthukumar, S., Prasad, S.: Portable biosensor for monitoring cortisol in low-volume perspired human sweat. Nature (2017) 1–13

[20] Muaremi, A., Arnrich, B., Tröster, G.: Towards Measuring Stress with Smartphones and Wearable Devices During Workday and Sleep. BioNanoScience **3** (2013) 172–183

[21] Anastasova, S., Crewther, B., Bembnowicz, P., Curto, V., Ip, H.M., Rosa, B., Yang, G.Z.: A wearable multisensing patch for continuous sweat monitoring. Biosensors and Bioelectronics **93** (2017) 139 – 145 Special Issue Selected papers from the 26th Anniversary World Congress on Biosensors (Part II).

[22] Choi, S., Kim, S., Yang, J.S., Lee, J.H., Joo, C., Jung, H.I.: Real-time measurement of human salivary cortisol for the assessment of psychological stress using a smartphone. Sensing and Bio-Sensing Research **2** (2014) 8 – 11

[23] Pryss, R., Reichert, M., John, D., Frank, J., Schlee, W., Probst, T.: A personalized sensor support tool for the training of mindful walking. In: 2018 IEEE 15th International Conference on Wearable and Implantable Body Sensor Networks (BSN). (2018) 114–117

[24] Mani, M., Kavanagh, D.J., Hides, L., Stoyanov, S.R.: Review and Evaluation of Mindfulness-Based iPhone Apps. PubMed (2015)

[25] Kozlowski, D.: Konzeption und Realisierung einer mobilen Anwendung zur Unterstützung von gestressten Patienten mithilfe des "Mindful Walking Gedankens". (2018)

[26] Google: Sensors Overview. https://developer.android.com/guide/topics/sensors/sensors_overview (2019) Accessed: 2019-07-01.

[27] SIG, B.: GATT Specifications. https://www.bluetooth.com/specifications/gatt (2019) Accessed: 2019-07-01.

[28] Ulm University, I.o.D., Systems, I.: QuestionSys - A Generic and Flexible Questionnaire System Enabling Process-Driven Mobile Data Collection. https://www.uni-ulm.de/in/iui-dbis/forschung/laufende-projekte/questionsys/ (2019) Accessed: 2019-07-01.

[29] Schobel, J., Schickler, M., Pryss, R., Nienhaus, H., Reichert, M.: Using Vital Sensors in Mobile Healthcare Business Applications: Challenges, Examples, Lessons Learned. In: 9th Int'l Conference on Web Information Systems and Technologies (WEBIST 2013), Special Session on Business Apps. (2013) 509–518

[30] Runtastic: Runtastic. https://play.google.com/store/apps/details?id=com.runtastic.android (2019) Accessed: 2019-07-01.

[31] IO, R.: Realm Database. https://realm.io/products/realm-database (2019) Accessed: 2019-07-01.

List of Figures

Listings

Name: Robin Bird Matriculation number: 750747

Honesty disclaimer

I hereby affirm that I wrote this thesis independently and that I did not use any other sources or tools than the ones specified.

Ulm, ...

Robin Bird