



**ZORGCAMPUS**

## Caregiving Time Registration System



DELFT UNIVERSITY OF TECHNOLOGY

FACULTY ELECTRICAL ENGINEERING, MATHEMATICS  
AND COMPUTER SCIENCE

IN3405 BSc PROJECT

*Authors:*

Erik van Paassen (1376624)  
Joep Talboom (1272969)

*Supervisors:*

Dr. Ertan Onur  
Yunus Durmuş MSc

*Company:*

ZorgCampus, Rotterdam

December 6, 2012

## Preface

At the Delft University of Technology, Computer Science students finish their bachelor's by doing a BSc project. For this project, students perform an internship at a company. This report is a result of the BSc project of Erik van Paassen and Joep Talboom. The project has been done at ZorgCampus in Rotterdam from September to November 2012.

We would like to thank our supervisors from the university, Dr. Ertan Onur and Yunus Durmuş MSc, for their expertise and very helpful feedback at our biweekly meeting. We would also like to thank Lionel Amstelveen for the opportunity to do our BSc project at ZorgCampus.

Erik van Paassen and Joep Talboom

Delft, December 6, 2012.

## Summary

Registration of working hours in home care can be very labor-intensive when paper declaration forms are filled in by hand. Currently, these forms also have to be entered in a computer system by hand. To maximize time spent on care giving, the time required for these registrations has to be minimal.

To accomplish this, an Android application has been developed which helps caregivers with time registration. The smartphone can be used to check in and out with a client's RFID-equipped card. Subsequently, the gathered information is transmitted securely to the backoffice system, where it is to be stored.

## Glossary

This section gives an overview of the terms and abbreviations commonly used in this document.

| Term              | Description                                                                                                           |
|-------------------|-----------------------------------------------------------------------------------------------------------------------|
| Check-in          | The action of registering a start time for giving care to a client.                                                   |
| Check-out         | The action of registering an end time for giving care to a client.                                                    |
| Transaction       | Information about check-in or -out send to the server.                                                                |
| Client            | The person who receives the home care.                                                                                |
| Caregiver         | The person who gives home care to a client.                                                                           |
| Coordinator       | The person of the home care company managing a group of caregivers.                                                   |
| Smartphone        | An Android 4.x based phone used to run the caregiver app.                                                             |
| Caregiver app     | The Android application created for this project used by the caregiver to check in or out.                            |
| NFC library       | The Android library created for this project with all NFC read and write functionally.                                |
| Backoffice module | The time registration management module for the existing backoffice created for this project used by the coordinator. |

Table 1: Terms

| Abbreviation | Description                                          |
|--------------|------------------------------------------------------|
| RFID         | Radio-Frequency IDentification.                      |
| NFC          | Near Field Communication.                            |
| REST         | REpresentational State Transfer.                     |
| JSON         | JavaScript Object Notation.                          |
| EEPROM       | Electrically Erasable Programmable Read-Only Memory. |
| API          | Application Programming Interface.                   |
| IMEI         | International Mobile Equipment Identity.             |
| IMSI         | International Mobile Subscriber Identity.            |
| DBAL         | DataBase Abstraction Layer.                          |
| ORM          | Object-Relational Mapper.                            |
| ADB          | Android Debug Bridge.                                |

Table 2: Abbreviations

# Contents

|                                                        |           |
|--------------------------------------------------------|-----------|
| Preface . . . . .                                      | i         |
| Summary . . . . .                                      | ii        |
| Glossary . . . . .                                     | iii       |
| <b>1 Introduction</b>                                  | <b>1</b>  |
| 1.1 Current situation . . . . .                        | 1         |
| 1.2 Proposed system . . . . .                          | 1         |
| <b>2 Final product</b>                                 | <b>3</b>  |
| 2.1 Android caregiver app . . . . .                    | 3         |
| 2.1.1 Known issues . . . . .                           | 3         |
| 2.2 Backoffice module . . . . .                        | 4         |
| 2.3 Recommendations . . . . .                          | 4         |
| 2.3.1 Android caregiver app . . . . .                  | 4         |
| 2.3.2 Backoffice module . . . . .                      | 5         |
| 2.3.3 Overall development process . . . . .            | 5         |
| <b>3 Security research</b>                             | <b>6</b>  |
| 3.1 Mifare Classic key recovery . . . . .              | 6         |
| 3.2 Duplication of a tag . . . . .                     | 6         |
| 3.3 ‘Rooting’ the smartphone . . . . .                 | 7         |
| 3.4 Decompilation of the Android application . . . . . | 7         |
| <b>4 Design decisions</b>                              | <b>8</b>  |
| 4.1 Caregiver application . . . . .                    | 8         |
| 4.2 Backoffice module . . . . .                        | 10        |
| 4.3 Design patterns . . . . .                          | 11        |
| 4.4 Language . . . . .                                 | 11        |
| 4.5 Standards . . . . .                                | 12        |
| <b>5 Reflection</b>                                    | <b>13</b> |
| 5.1 Planning . . . . .                                 | 13        |
| 5.1.1 Milestone 0 – Research phase . . . . .           | 13        |
| 5.1.2 Milestone 1 – Design phase . . . . .             | 14        |

|          |                                                       |           |
|----------|-------------------------------------------------------|-----------|
| 5.1.3    | Milestone 2 – Basic Android application . . . . .     | 14        |
| 5.1.4    | Milestone 3 – Basic backoffice module . . . . .       | 14        |
| 5.1.5    | Milestone 4 – Extending functionality . . . . .       | 15        |
| 5.1.6    | Extra time – Code quality and report . . . . .        | 15        |
| 5.1.7    | Conclusion . . . . .                                  | 15        |
| 5.2      | Design . . . . .                                      | 15        |
| 5.2.1    | Use cases . . . . .                                   | 16        |
| 5.2.2    | Application GUI flow . . . . .                        | 16        |
| 5.2.3    | Technical design . . . . .                            | 16        |
| 5.3      | Quality assurance and testing . . . . .               | 16        |
| <b>A</b> | <b>Assignment description</b>                         | <b>21</b> |
| <b>B</b> | <b>User manual of the caregiver application</b>       | <b>25</b> |
| <b>C</b> | <b>Planning</b>                                       | <b>31</b> |
| C.1      | Initial planning . . . . .                            | 31        |
| C.2      | Revised planning after milestone 3 . . . . .          | 31        |
| <b>D</b> | <b>Requirements analysis</b>                          | <b>34</b> |
| D.1      | Proposed system . . . . .                             | 34        |
| D.2      | Functional requirements . . . . .                     | 34        |
| D.3      | Non-functional requirements . . . . .                 | 36        |
| D.4      | Constraints (pseudo-requirements) . . . . .           | 36        |
| D.5      | Use cases . . . . .                                   | 36        |
| D.5.1    | Caregiver . . . . .                                   | 37        |
| D.5.2    | Coordinator . . . . .                                 | 38        |
| <b>E</b> | <b>Architectural design</b>                           | <b>43</b> |
| E.1      | Introduction . . . . .                                | 43        |
| E.1.1    | Purpose of the system . . . . .                       | 43        |
| E.2      | Recommended software architecture . . . . .           | 43        |
| E.2.1    | Overview and subsystems . . . . .                     | 43        |
| E.2.2    | Hardware/software mapping . . . . .                   | 45        |
| E.2.3    | Communication flow . . . . .                          | 45        |
| E.2.4    | Global resource handling and access control . . . . . | 47        |
| E.2.5    | Exceptional situations . . . . .                      | 47        |
| E.2.6    | Security threats . . . . .                            | 48        |
| <b>F</b> | <b>Technical design</b>                               | <b>52</b> |
| F.1      | Introduction . . . . .                                | 52        |
| F.2      | Mifare Classic data layout . . . . .                  | 52        |
| F.2.1    | Key features of the Mifare Classic tag . . . . .      | 52        |
| F.2.2    | Record system . . . . .                               | 52        |

|          |                                                         |           |
|----------|---------------------------------------------------------|-----------|
| F.3      | Care record . . . . .                                   | 53        |
| F.4      | Caregiver application . . . . .                         | 56        |
| F.4.1    | Class diagram . . . . .                                 | 56        |
| F.4.2    | Database diagram . . . . .                              | 56        |
| F.5      | Backoffice module . . . . .                             | 56        |
| F.5.1    | Class diagram . . . . .                                 | 58        |
| F.5.2    | Database diagram . . . . .                              | 58        |
| <b>G</b> | <b>Quality assurance and testing plan</b>               | <b>61</b> |
| G.1      | Development process . . . . .                           | 61        |
| G.2      | Continuous automated build and testing system . . . . . | 61        |
| G.3      | Code quality . . . . .                                  | 62        |
| G.4      | Unit testing . . . . .                                  | 62        |
| G.5      | Regression testing . . . . .                            | 62        |
| G.6      | GUI testing . . . . .                                   | 62        |
| G.7      | Functional testing . . . . .                            | 63        |
| G.8      | Acceptance testing . . . . .                            | 63        |

# Chapter 1

## Introduction

Home care working hours registration is a very labor-intensive job. It has to be done very accurately to make sure no mistakes are being made. When spending a lot of time on administration, less time is available for the actual care giving. Home care is becoming expensive more and more, so the little time a caregiver has to give care to a client, has to be used efficiently.

### 1.1 Current situation

In the current situation, working hours registration is done by filling out declaration forms by hand by the caregiver. These forms, in turn, have to be approved and entered in a computer system manually by the home care coordinator. These operations are labor-intensive, sensitive to human mistakes and very vulnerable to fraud. It is hard to check whether the working hours declarations match the actual hours the caregiver has been working. Incorrect administration can lead to significant financial implications for both the employee and health care company and it could negatively influence the quantity of care clients receive.

### 1.2 Proposed system

The main reason for automation of the working hours registration is to reduce the time needed for weekly or even daily administrative tasks. By taking away these tasks from the caregivers and the care coordinators, they will save more time to focus on their main tasks. The coordinator can spend more time on keeping the care records up to date and improving the health care plan.

The system should allow the caregivers to check in when they start working with a client and check out when stop working with that client. These check-ins and check-outs should be collected centrally, to allow automatic generation of declarations. By approving the registered working times automatically when the system can guarantee the correctness, much of the coordinators' time will be saved.

The assignment for this project, created by the company ZorgCampus, can be consulted in appendix A. The design documents that specify the proposed system in a more detailed way, are included in appendix D, E and F.

## Chapter 2

# Final product

This chapter covers the final product, its issues and the recommendations for further development.

### 2.1 Android caregiver app

The Android app is currently at a state of being ready for usage in a pilot program. Probably some of the configured parameters have to be tuned during such a test program. These options are mainly about the synchronization process and about how a location is acquired.

Consult the user manual for this application to get an impression of the exact workings. This manual is included in appendix B. The application is available both in English and Dutch, but the manual is available only in Dutch for now, though.

#### 2.1.1 Known issues

The application has some known issues, mainly due to the lack of time to fix these problems. These include:

- Keys for database access, NFC connectivity, client-side NFC data verification and HTTPS authentication are hard coded in the application. This makes the application vulnerable, because these keys can be obtained by decompiling the application package.
- Application is vulnerable to changes in system time.
- Time registration by using QR-codes is not yet supported, but could be implemented in the future.

## 2.2 Backoffice module

The backoffice module consists of a RESTful service to store time registrations and some views to show the smartphones, NFC tags and time registrations. Currently, there's no possibility to edit this data yet. It should provide a good basis, though, which can be extended by the company's developers.

## 2.3 Recommendations

This section will address the recommendations about the *caregiver app* and the *backoffice module* for the existing system.

### 2.3.1 Android caregiver app

#### NFC tag

In chapter 3 and in the Architectural design document E, it was already mentioned that the security of the Mifare Classic chip should be considered weak. It was decided that in this case, this is not yet a problem because of the amount of effort still needed to copy or manipulate a tag.

If, at any moment in the future, copying of a Mifare Classic tag would become easier, a different tag type might be chosen. This could be the case when, for example, smartphones would be going to support *tag emulation* and *one-click* Mifare Classic emulators would be released.

The Mifare Desfire EV1 chip uses the more sophisticated AES encryption method, and has not yet been cracked at the time of writing. Switching to this type of card would require development of a new specific class in the NFC library and making a minor adjustment to the Android application.

#### QR-codes

The functionality of scanning a QR-code with the camera of the phone has not yet been implemented. We think it would take about three days to add this to the Android app. The most time will be needed to create a system to generate QR-codes, assign these to an NFC tag and make sure each one is printed on the right card.

#### Extra functionality

The company has indicated that some extra functionality should be added to the app in the future. One thing is that the planning schedule could be sent to the phone. If the schedule changes, the caregiver could be notified. The caregiver can also be alarmed if he hasn't checked in yet with a client in time. As this is a very large feature, it has to be designed first to make an accurate estimation of development time.

### 2.3.2 Backoffice module

The *backoffice module* shows an overview of the smartphones, NFC tags and all saved time registrations. The next step is to add functionality to edit and create new smartphones and NFC tags. Time registrations should not be edited, as the original data must be saved at all times. This can probably be done in within a week or two.

More design and implementation time must go to the coupling between the time registration and planning.

### 2.3.3 Overall development process

To create a better development atmosphere within the IT department we have a few recommendations. It could be useful to use SCRUM as a development method, with sprints of two or three weeks. Trac could be very useful as management tool. Not only to track issues, but also to create a roadmap. This allows to keep track of the progress at all times.

Overall productivity could be boosted when new features would be planned and designed in more details, before implementing them. This could be done at the start of every sprint. Also, requirements should be clear before design and implementation.

At the end of each sprint, an evaluation can be made about what went wrong, what went well and how realistic the planning was. This way everyone learns to make a realistic planning so deadlines don't move all the time.

## Chapter 3

# Security research

During the design phase of the project, we have investigated some of the potential security issues which were identified in the architectural design document E. To get a realistic estimation of the effort needed to break the security of certain systems, we tried break it ourselves.

### 3.1 Mifare Classic key recovery

As already mentioned in this architectural design document, the Mifare Classic chip uses keys to regulate the access to the EEPROM sectors. Anyone with the correct key can read or write such a sector, depending on how the access conditions of the sector are configured.

In March 2008 the Mifare Classic was completely reverse engineered by the Digital Security research group of the Radboud University Nijmegen [30]. The keys configured on the Mifare Classic can be recovered within a few minutes with a tool called `mfoc`, which stands for *Mifare Classic Offline Cracker* [16]. All that is needed to recover the keys from the tag are an affordable NFC tag reader and this piece of software. We used the *ACR122U* [2] from Advanced Card Systems Ltd. [3], which is widely available in many online stores for about 30 euros. With `mfoc`, a dump of the entire EEPROM, including the blocks in which the keys are stored, can be made. Additionally, it's possible to write arbitrary content to the tag with the `nfc-mfclassic` [13] program, using the recovered keys from the dump file.

In conclusion, the security of the Mifare Classic chip should be considered relatively weak, as can be breached within minutes.

### 3.2 Duplication of a tag

All official Mifare Classic tags are fabricated with a read-only unique id in the first EEPROM block. Unfortunately, there exist some Chinese Mifare Classic compatible clones available, on which the first block is fully customizable. This means the UID can be overwritten, what makes it possible to create an exact duplicate of a tag.

When a tag is copied, this will mean that a caregiver could potentially use it to check in or out while not being at the client’s home.

### 3.3 ‘Rooting’ the smartphone

Most of the Android smartphones available on the market can be *rooted*. When a smartphone is rooted, the user has root access to the entire file system. This means, that access will also be granted to the SQLite database file of our application, in which time registrations are stored until these are synchronized with the backoffice system. It’s fairly easy to pull the database file from the smartphone using the ADB interface over a USB connection.

So, with an unencrypted database file on a rooted smartphone, time registrations read or even manipulated, as long as they are not synchronized yet. Section 4.1 covers implementing the protection against this vulnerability.

### 3.4 Decompilation of the Android application

Android application packages (so-called APK-files) can be decompiled with `apktool` [5]. We have successfully decompiled an early version of the caregiver application, concluding that the result was easily understandable. To make an app harder to reverse engineer, it can be obfuscated with a tool called *ProGuard*, which is included in the Android Software Development Kit.

“The ProGuard tool shrinks, optimizes, and obfuscates your code by removing unused code and renaming classes, fields, and methods with semantically obscure names. The result is a smaller sized .apk file that is more difficult to reverse engineer. Because ProGuard makes your application harder to reverse engineer, it is important that you use it when your application utilizes features that are sensitive to security like when you are Licensing Your Applications.”

Of course, obfuscation does not provide any real security, but it significantly raises the effort that is needed for reverse engineering. When even more security is needed, *DexGuard* [7] could be used: it can, for example, encrypt strings and even entire classes. The use of this piece of software requires purchasing a license, so for now, it won’t be used in this project.

## Chapter 4

# Design decisions

In this chapter, we will discuss the design decisions that were not yet covered by any of the design documents.

### 4.1 Caregiver application

**NFC library** We have decided to put all NFC related functionality in a separate library, so it can be used for other Android applications that could be developed in the future. The library defines a high-level interface to read and write data records on an NFC tag. It also significantly lowers the extra effort needed to implement a new record type, or to use a different kind of NFC tag.

**Mifare Classic vs. Mifare Desfire EV1** The Mifare Desfire EV1 has more hardware and software security features than Mifare Classic. The security of this card hasn't been breached yet. It supports a higher level of security by using 128bit AES keys and the hardware and operating system is Common Criteria certified at level EAL (Evaluation Assurance Level) 4+.

“To achieve a particular EAL, the computer system must meet specific assurance requirements. Most of these requirements involve design documentation, design analysis, functional testing, or penetration testing. The higher EALs involve more detailed documentation, analysis, and testing than the lower ones. Achieving a higher EAL certification generally costs more money and takes more time than achieving a lower one.”

**Manual input** If for some reason it's not possible to scan the client's card, working times have to be registered manually. The caregiver has to select the date and time and enter the name of the client and a reason why a manual input is performed. All these fields are mandatory. The app checks whether the date and time specified by the user are not in the future. It is also not possible to perform a check-out with a date and time that lay before the date and time of checking in.

**Acquiring the location of the device** On Android, an application can subscribe to location updates, after which the phone will try to acquire an accurate location from the GPS sensor. Every time the device gets a newer or more accurate location, a location update is sent to all applications that have subscribed to these updates. For this reason, it's not possible for an application to acquire the accurate location immediately. Accuracy is defined in the Android API as:

“We define accuracy as the radius of 68% confidence. In other words, if you draw a circle centered at this location's latitude and longitude, and with a radius equal to the accuracy, then there is a 68% probability that the true location is inside the circle.”

At the time of checking in or out, the application needs to register the current location of the device. In order to do this, it waits for at most 30 seconds before finalizing the time registration and transmitting it. If the desired accuracy of the location is achieved earlier, that is 50 meters or less, the time registration will also be finalized earlier. The above-mentioned values are fully configurable by the developer, only by adjusting the application's configuration file.

**Ensuring enabled GPS and NFC sensors** The caregiver app utilizes both the GPS and NFC sensor of the smartphone it's running on. As these sensors can be disabled in the settings panel of the Android OS, the application needs to enforce that these remain enabled. By design, disabled sensors cannot be enabled by an application programmatically. Android restricts this to prevent abuse, so the user will have to enable the sensors by hand. This is not an ideal situation, because doing so can be a hard task for some users.

When a sensor is disabled, the caregiver app notifies the user by means of an alert dialogue, gives instructions and opens up the right settings panel, so the user can enable it in the most easy way possible.

**Database encryption** The Android platform provides the ability to create and use *SQLite* databases on the smartphone. By default, such a database is unencrypted. It can be accessed by *rooting* the smartphone and then extracting the database file from the device through a USB connection.

The *caregiver app* uses an *SQLite* database to store the time registrations which haven't been transmitted successfully yet. To prevent unauthorized access to the database, we chose to encrypt it using the *SQLCipher for Android* [21] library. This library encrypts the database using *256-bit* AES encryption. At this moment, the key for encrypting and decrypting the database is hard coded in the application. This way, it is a little bit more secure than when placed in a separate configuration file, but the key could be obtained by decompiling the application.

## Application state

In order to keep the user interface as simple as possible, it has been decided that a caregiver cannot check out when he or she hasn't yet checked in, or the other way around. This introduces state into the application, but reduces the complexity of operation for the caregiver. The application should show clearly what current state (*checked in* or *checked out*) it's in. If one forgets to check in, the check-in time has to be entered manually before checking out will be possible.

## 4.2 Backoffice module

**RESTful service** For enabling the smartphone to communicate with the backoffice system, a *RESTful* [19] web service has been implemented. When a time registration has been completed, a POST request is sent to this service, which then stores the time registration. Using a REST architecture was strongly advised by Yunus Durmuş, one of our supervisors from TU Delft. Finally, we chose this solution, because it is lightweight and easy to implement.

The data that is sent is encoded in the JavaScript Object Notation, or *JSON* for short. We decided to go for this solution, as almost every platform has JSON support. PHP 5.2+ has methods for encoding and decoding JSON [8] and Android has some classes (such as *JSONObject* [11]) for it. This makes it, again, easy to implement.

**Input validation** As all time registrations that are received by the web service have to be validated, we had to make a decision about how this would be done. After some research, we chose the validator component from the *Symfony framework* to provide a framework for this. This component integrates neatly with the Doctrine ORM that was already in use: each entity can define its own validation rules per attribute.

**HTTPS connection** The communication between smartphone and server happens over an HTTPS connection. A self-signed certificate can be used to verify the authenticity of the server, which requires that this certificate is installed on the Android device or included in the application. Alternatively, the company could purchase a certificate from a certificated authority (CA), or act as a certificate authority itself. The latter still requires a certificate to be installed on the client side. During development, we have used self-signed certificates which were installed on the smartphone.

We disabled *SSLv2* in the configuration of the web server, as this protocol has known vulnerabilities. For security reasons, only *SSLv3* or any version of *TLS* should be allowed. Additionally some cipher suites can be disabled to strengthen the security. Because the request volumes will be low (so computational overhead is not a problem), the web server can be configured to allow only the *HIGH encryption* [6] cipher suites, with at least 128-bit keys. We have tested this configuration in combination with the Android application and it works just fine.

## 4.3 Design patterns

In both the application and the *backoffice module*, we have used several design patterns. This section discusses the most important ones.

**Model–View–Controller (MVC)** To achieve separation between data and logic on one side, and visual representation on the other side, both the Android application and the server side counterpart make use of the *Model–View–Controller* design pattern. Note that Google forces Android developers to use this architecture. For the backoffice system, the MVC structure was already present.

**Singleton** Some classes of the Android application, such as *StateManager*, use the Singleton-pattern, which ensures they can be instantiated only once. After instantiation, these objects can be called from anywhere within the code. The *StateManager* class uses this pattern because it keeps track of the global state of the application.

**Factory** The *NfcRecord* class, from the *NFC library* for Android that has been created, implements the *Factory* design pattern. It accepts a record identifier and a payload, on which it decides the right *NfcRecord* subclass and instantiates it.

**Observable** The *StateManager* also implements the *Observable* pattern. Android Activity-classes can attach themselves as an observer to the *StateManager* and observe the global state of the application. When the state changes, the observer classes are notified and the changes can be rendered to the user interface.

**Broadcast–Listener** The Android platform uses the *Broadcast–Listener* pattern for many purposes. This pattern is quite similar to the Observable pattern. In the *caregiver app*, it is used for receiving broadcast messages about screen state (on/off) and for broadcasting messages about synchronization being completed.

**Front Controller** The backoffice system uses the *Front Controller* pattern, which means all requests are routed through a single class, which decides how to handle the request. We have created such a *single point of access* to serve all REST requests.

## 4.4 Language

The *caregiver app* is in both English and Dutch available. When the language of the smartphone is set on Dutch, the app is also in Dutch. With every other phone language, the language of the app is in English.

## 4.5 Standards

We tried to use as much standards already available so we could focus on the new parts in our project.

- The NFC tag that is used in this project is the widely used Mifare Classic.
- The card with the NFC tag inside has a standard credit card format.
- The *caregiver app* can be installed on any Android phone with Android 4.0 and up.
- The *NFC library* is suitable for extension to support other NFC tags, e.g. the Mifare Desfire EV1.
- The interface of the app compliance most of the Android GUI guidelines.
- The code style of the app compliance the Android code style guidelines.
- The RESTful service to transmit the time registrations in JSON format to the central server.

# Chapter 5

## Reflection

In this chapter, we will reflect on our planning, design and testing documents.

### 5.1 Planning

In the first two weeks of our project, we were still looking for a supervisor from TU Delft. During this phase, the work that was done consisted mainly of performing research on Android development and on what would be possible with near field communication. After two weeks, the first meeting with our TU Delft supervisors took place. We agreed to use an agile development method with a milestone every two weeks.

In this section, we'll give a short overview of what was done in which milestone. Our initial planning is included in appendix C.

#### 5.1.1 Milestone 0 – Research phase

September 17 - 28 – Like said before, the first two weeks consisted of research that was needed to be done before we could design the system. We investigated the existing backoffice system and the possibilities to extend it. We experimented with Android development and set up a development environment with the Jenkins continuous integration system.

After we knew what we were dealing with, we started to write scenarios for system failures and security threats, including the possible solutions. These are included in the requirements analysis document, which is attached as appendix D.

At the end of week two, we created a small demo app, which could read and write some plain text to a Mifare Classic tag. At the end of week two we had our first meeting with our supervisors on the university. In this meeting we were advised to have development *sprints* of two weeks and also use a project management tool to keep track of all tasks. We chose Trac for that, which integrated neatly with *Mercurial* the version control system we were using.

### 5.1.2 Milestone 1 – Design phase

October 1 - 11 – A global planning was made in the form of a Gantt chart (see appendix C). This planning for the next eight weeks was divided in four sprints. At the end of every sprint, there was a milestone with certain goals. The eleventh and last week of our project was not included in this initial planning, as we decided to use this week as *extra time* to fix remaining issues and finish our final report. With every milestone we had a meeting with our supervisors from the university to discuss the progress we made. We also introduced Trac to the other developers of the company, and deployed it to the company’s server, so they were able to use it too.

During this second sprint several required documents were written: *Requirements analysis*, *Architectural design*, *Technical design* and *Quality assurance and testing*. These documents attached as appendices D, E, F and G. Construction of the Android GUI was also started, so we could demonstrate this during the meeting with our supervisors.

### 5.1.3 Milestone 2 – Basic Android application

October 12 - 25 – Implementation of the NFC library started at the beginning of the milestone and took about a week and a half. We created unit tests for some important library classes, and these tests successfully pointed out some critical bugs. Trac tickets were created for these bugs, even if they could be fixed easily. This way, we could keep track of the overall progress in a better way.

Furthermore, a basic version of the caregiver application was created, which allowed time registration by scanning an NFC tag and by manual input. As there was no backoffice connectivity yet, it just logged the data that would have to be sent.

During this sprint we also tried to hack our design by performing Mifare Classic key recovery and decompiling the Android application package. We also tested what could be done with a rooted smartphone. See chapter 3 for the results.

### 5.1.4 Milestone 3 – Basic backoffice module

October 26 - November 8 – With sprint 3, the project continued with extending the existing backoffice system. The first thing needed to be done was creating a RESTful service that would accept time registrations and store these. This put several requirements: model classes (Doctrine *entities*) had to be created and the backoffice system had to be altered on some points to allow a REST interface. We discovered that exception handling was not sufficiently robust, mainly due to a PHP bug in some builds. As this raised an issue with the HTTP status code not being set correctly, we had to fix it first. Such issues have cost us some time, but until then the initial planning was followed very strictly.

Most of the *backoffice module* was produced during this milestone. First we had to read about Doctrine ORM, because the existing backoffice system was using it. There was no validation framework in use yet, so we had to pick one. We chose Symfony Validation, but didn’t get it to work with docblock annotations. Finally, it worked when

we specified validator meta-data in a dedicated method per class, but it has been quite a setback to our planning.

Next thing to be done was connecting the application to the backoffice system. As synchronization could be implemented much less complex way when every time registration was first stored in the database, we decided to change the design and planning. Storage of registrations would have to be implemented during the next sprint anyway, so the time that would be lost now could be recovered right after milestone 3.

On the last day we created a new Gantt chart for the remaining two weeks of the project, plus the one week of extra time. We specified the tasks in more detail, because we wanted to avoid that new unforeseen situations would lead to missing the final deadline.

#### **5.1.5 Milestone 4 – Extending functionality**

November 9 - 23 – We cleaned our total code base and added extra comments to clarify our code. The codebase of the caregiver app, NFC library and backoffice module were sent to *SIG* for a maintainability and complexity check. The recommendations following from this evaluation are listed in section 5.3.

Some of the tasks that have been completed during this sprint are: implementing GPS functionality, implementing encryption of the database on the Android device and improving the user interface of the application.

The main focus was on improving the Android application, as the company’s developers don’t have any experience with Android development yet. So for them, it would be easier to extend the functionality of backoffice module than to do so for the app.

#### **5.1.6 Extra time – Code quality and report**

November 26 - 30 – This week was initially planned as extra time to fix potential remaining issues. On Monday, we were still making some improvements to the overall code quality and on Tuesday the codebase was sent to the Software Improvement Group for re-evaluation. The remaining days of this last week were all about finishing the final report.

#### **5.1.7 Conclusion**

The planning was quite accurate for the first half of our project and we followed it very strictly. After six weeks, we experienced some setbacks and a dependency flaw in our planning. For the end of the project, we made a new and more detailed planning, which helped us staying on track in the final phase. Overall, we have benefited greatly from creating these Gantt charts.

### **5.2 Design**

This section evaluates the design process. For more information, see the *Requirements analysis document*, which is included appendix D.

### 5.2.1 Use cases

One thing we didn't design on beforehand, is the possibility to view the time registrations which have not been synchronized yet. First we thought that the users should not see these *pending* time registrations, because they couldn't influence them. But when the registrations can't be send to the central server, they should be written down on paper so they can be entered manually in the backoffice system. For this, the caregiver must be able to see the time registrations which have failed synchronization.

### 5.2.2 Application GUI flow

In the use cases we described that the caregiver would first scan an NFC tag and then choose if it was a check-in or -out. When we started to implement the app, we changed this behavior. A caregiver should now always check in first before he can check out. When this flow is forced, the user interface will be less confusing and the caregiver can make less mistakes. This decision has been taken in consultation with the ZorgCampus management

### 5.2.3 Technical design

The technical design has been quite accurate. There have been some small adjustments to the class diagrams during the implementation. For example, a database helper has been added and some attributes turned out to be missing. Also, some parts have been refactored (*Location* and *DeviceReport* classes), but the overall structure of is still the same, though. Figure 5.1 shows the class diagram of the caregiver application in the current state. The original class diagram can be found in appendix F.

## 5.3 Quality assurance and testing

This section evaluates the quality assurance and testing document, which is included in appendix G.

**SCRUM** During the project, we adhered to some rules the SCRUM development method. Working with development sprints of two weeks was a good way to keep track of the global planning. Each sprint had it's own *theme* and corresponding milestone. Additionally, at the end of each day, we evaluated was done that day and what should be done the next day. Because our team consisted of only two people, we left out most of the other SCRUM rules.

**Continuous integration** The continuous integration system called *Jenkins* was set up in the very first week. It's nice to have an automated system that builds, installs, tests and saves the artefacts of every revision that that committed to the source code repository. Because it automatically saves all executables and log files of every build, we were able to install older versions of the app without the need of rebuilding it first.

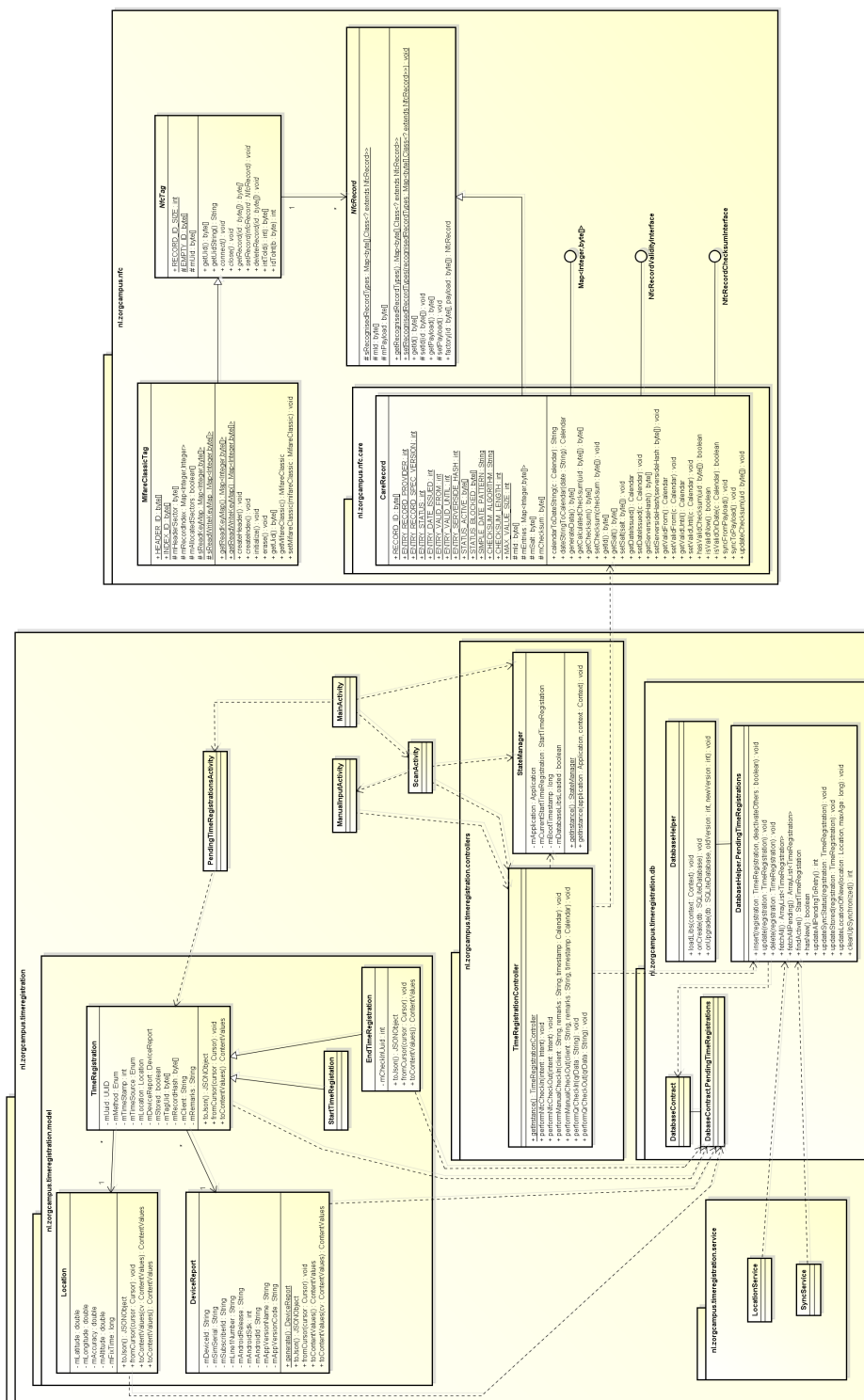


Figure 5.1: The revised class diagram of the caregiver app.

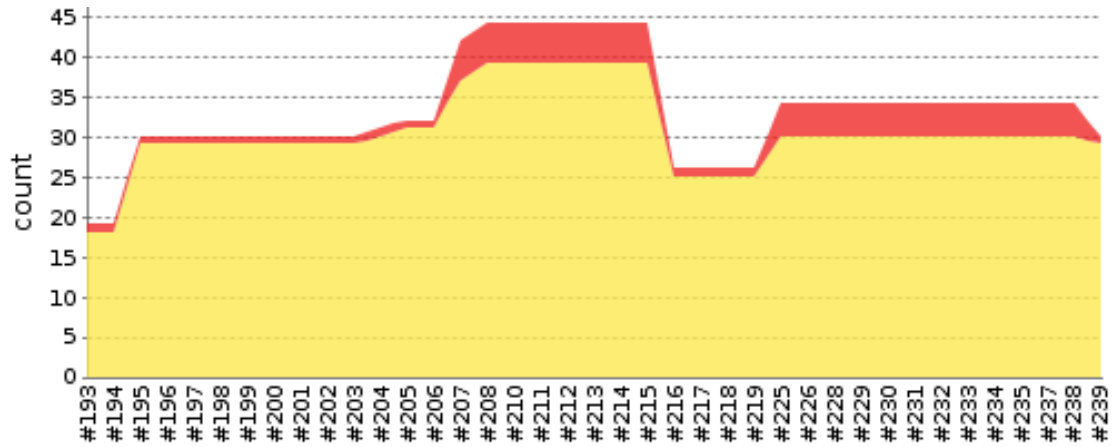


Figure 5.2: Lint trend as shown by Jenkins

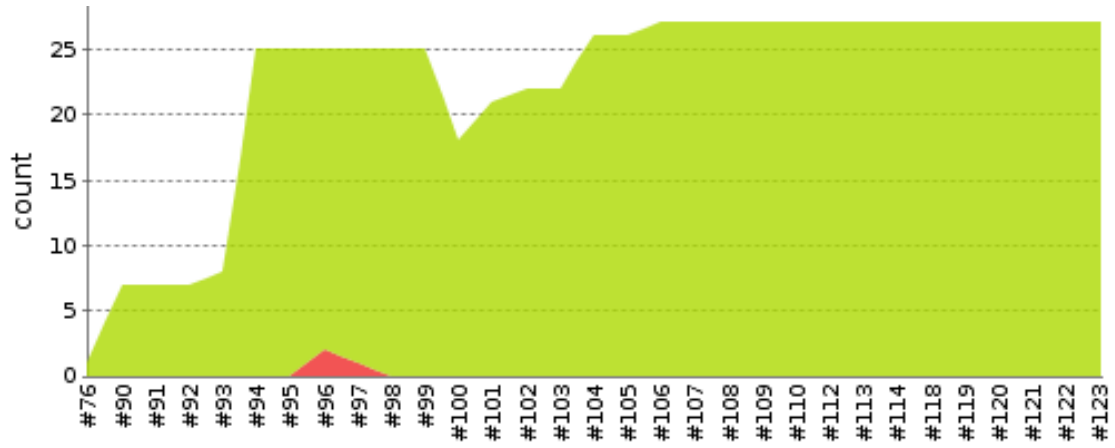


Figure 5.3: JUnit test results as shown by Jenkins

This makes it easy to check whether a bug was already present in a previous version or that it is caused by new functionality.

With the Android Lint tool, which is included in the Android SDK, the application source code was scanned for potential bugs. This tool examined the code for known pitfalls and creates a report and a graph with the errors and warnings per build. An example of the Lint trend of the *caregiver app* is shown in figure 5.2.

Jenkins also executed all unit tests after a successful build. A graph with the total number of *JUnit* test cases of every build of the *NFC library* can be viewed in figure 5.3 and the line coverage of these tests per build are shown in figure 5.4.

**Unit testing** Almost all methods in the model classes of the *caregiver app* and the *NFC library* have *JUnit* tests. Methods that perform NFC I/O operations don't have

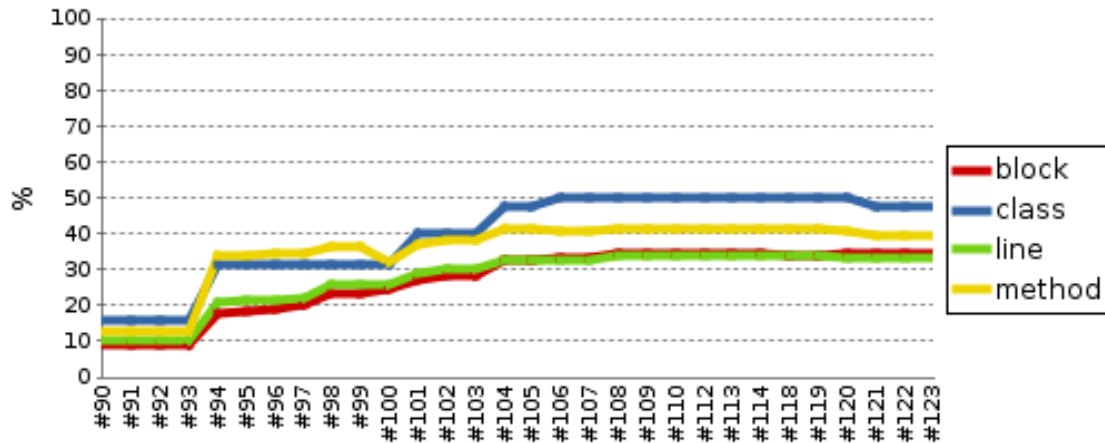


Figure 5.4: Code coverage as shown by Jenkins

unit tests, as these operations cannot be tested automatically on Android (yet). All tests regarding NFC operations were done by hand, and we even permanently crippled two tags by writing accidentally invalid access conditions to it. The GUI testing was done by hand due to the lack of priority to automate this, but this was already foreseen in the testing document.

The backoffice module has no unit tests yet, because all model classes mainly contain generated code with only trivial *getters* and *setters*. Database mapping is handled entirely by the Doctrine ORM and checking whether the model class has been configured correctly is as simple as comparing the generated database table against the original database diagram.

**Functional testing** In the *Requirement analysis document* all functional requirements are sorted by *MoSCoW* priority (see D.2). All tasks from the *Must have* section are completed. From the *Should have* section, everything except 4 requirements are done. The ability to edit NFC tags (D.2 S4), edit working hours (D.2 S9), approve/disapprove working hours (D.2 S10) and the resistance against malicious changes of the system time (D.2 S8) are not finished. The main reason for this is that we focused on finishing the Android part of the project, because this was more important to the company in the end. The developers of the company are able to complete the remaining tasks for the backoffice system. If the programmers had to finish the app, it would take them a lot of time to get familiar with the Android environment first. We have implemented tasks with the highest priority first, so none of the *Could have* and *Want to have* requirements are done.

**Acceptance testing** There has been an acceptance test with a coordinator from the company, acting as a caregiver for once. This coordinator is responsible for the time registrations of caregivers. When things go wrong, this coordinator is the one who

must take care of this. Only some small remarks were made during the acceptance test meeting, most about adding extra explanation in the user manual, one about the name of an input field. All found issues were resolved within an hour.

There was no acceptance test for the backoffice module, because it doesn't have enough functionality to be usable for coordinator already.

**Code quality** Between milestone 3 and 4, after eight weeks of our project, we have submitted our codebase to the *Software Improvement Group*. This company evaluated the source code on the aspects of quality of maintainability. Getting the source code evaluated by SIG was a requirement posed by TU Delft. At this first evaluation, the code received a 4 out of 5. According to SIG, there was room for improvement in the field of *Component Independence*, *Unit Size* and *Unit Complexity*.

The `MifareClassicTag` from the NFC library contained both generic and some company-specific functionality, which should have been separated. The dependency between the Android application and the NFC library was tighter than needed (Component Independence). Also, some methods were too long, which made them too complex too (Unit Size and Complexity).

In the last week of our project, we have improved the quality codebase on most of the above-mentioned weak points. The `MifareClassicTag` class however, remained rather large and still has some large methods. Partly, this was due to the fact that it translates a high-level API to low-level I/O operations.

After the corrections had been made, the codebase has been re-evaluated by SIG again, while focusing on the points identified earlier. The evaluation showed that the codebase had grown by about 20% and that the maintainability was about the same. Unit Size and Unit Complexity had improved, except within the `MifareClassicTag` of the NFC library, just like expected.

According to SIG, the caregiver app still depended on the `MifareClassicTag` class, and it assumed certain knowledge of this specific tag type within the application. This is certainly true, but when using a certain NFC tag type for an application, it just cannot be used though one comprehensive interface without losing specific functionality. Each piece of hardware has different functionality and specifications. For example, think of key authentication: this is not supported by just every type of NFC tag. But still, when choosing a different type of NFC tag for this application, this would require only a minimal amount of caregiver app source code (4 statements) to be changed. This seems very acceptable for now, though.

Furthermore, it was noticed that the amount of test code had grown like the total amount of code had, what obviously was positive.

## Appendix A

# Assignment description

The next pages show the description of the assignment we received from the ZorgCampus. Roughly translated it says:

ZorgCampus wants to develop a healthcare registration application which registers the start and end time of visiting a client. The registered times have to be stored in the web-based backoffice system. The current ideas are about an RFID-card or QR-code which is kept at the client's home and can be scanned.

The company expects a well-documented prototype, of which the maintenance should be easy. The prototype should be able to be integrated the existing system that have been developed by ZorgCampus.

# ZorgCampus – Zorgregistratiesysteem



## Hoe kan jij de zorg verbeteren?

(het d.m.v. QR code of RFID pas registreren van bezoek aan huis bij een zorgbehoevende)

### ***ZorgCampus, Versneld en Levenslang Leren***

#### **Wie is Zorgcampus?**

ZorgCampus biedt mensen de mogelijkheid aan om zich volgens het *levenslang leren* concept te ontwikkelen.

De organisatie is een MVO-initiatief (Maatschappelijk Verantwoord Ondernemen) waarin niet winstmaximalisatie centraal staat, maar ontwikkelingsmaximalisatie van mensen. De tijd waarin wij allen leven staat in het teken van snelle ontwikkelingen. Deze snelle ontwikkelingen zijn vooral te merken in hightech markten, maar deze snelle ontwikkelingen zijn ook te voelen in de zorg. Daardoor wordt het gat tussen onderwijs en werk steeds groter.

ZorgCampus zorgt, in samenwerking met zorginstellingen en werkgevers, door middel van competentiegericht onderwijs dat de aansluiting tussen studie en werk wordt versoepeld. Door de onderwijsvorm, verzorgd door ZorgCampus, worden studenten opgeleid aan de hand van de verwachting van de werkgever, waardoor studenten leren waarde toe te voegen aan de organisaties van werkgevers en gemotiveerd aan de slag kunnen.

“Onderwijs als onderdeel van extra diensten” is de Zorgcampus visie hoe zij onderwijs op een innovatieve manier een rol willen geven. Het draait niet om het onderwijs, maar om de mens die onderwijs volgt.

#### **Hoe gaat ZorgCampus te werk?**

ZorgCampus is een flexibele zorgopleider, waarbij instroomperiodes worden bepaald op basis van aanmeldingen in tegenstelling tot vaste instroomperiodes volgens schoolkalenderdagen.

Het onderwijs van ZorgCampus bestaat uit hoofdzakelijk twee aspecten: Competentiegericht onderwijs en een interactief e-learning systeem. Door het gebruik van het ZorgCampus e-learning systeem, kunnen studenten via het e-learning systeem huiswerk maken, opsturen en tevens feedback ontvangen. Tevens worden verworven competenties digitaal afgevolgd, totdat de student alle benodigde competenties heeft verworven.

ZorgCampus zorgt voor de opleiding van de studenten en levert baangaranties. Het werkleer-traject wordt op wijkniveau uitgevoerd, zodat studenten vlakbij huis kunnen leren en werken, met als doel om leefomstandigheden in de wijk op het gebied van zorg te verbeteren.

#### **Wat doet ZorgCampus?**

ZorgCampus biedt levenslang leren en ontwikkelen aan, waarbij studenten worden opgeleid tot zorgverleners en waarde kunnen toevoegen aan het leven van de zorgbehoevende.

### ***Beschrijving van de opdracht***

Zorgcampus wil een zorgregistratie applicatie ontwikkelen waarbij het tijdstip van bezoek en vertrek bij een zorgbehoevende wordt geregistreerd door een applicatie. De geklokte tijd wordt vervolgens weggeschreven in de webapplicatie van Zorgcampus. De ideeën gaan nu uit in de richting van een RFID-kaart die bij een zorgbehoevende cliënt thuis ligt en waarbij de zorgverlenende persoon op inscant, of bijv. via een QR code, of bijv. via een te installeren unieke certificaat zowel bij de zorgbehoevende bewoner als bij de zorgverlener. Het wegschrijven van de uren zou via een dockingstation kunnen dan wel via andere alternatieven.

### ***Wat wordt er verwacht?***

De inschatting is dat de opdracht door één student kan worden gedaan (max. 2 studenten bij uitbreiding opdracht en goede fasering, of taakverdeling).

Van jou wordt naast engineering skills ook verwacht dat je goed vanuit de eindgebruiker denkt en kan programmeren.

Er wordt verwacht dat tijdens de bouw van het prototype dat er op een duidelijke wijze gedocumenteerd wordt. Dit vragen wij in verband met het in praktijk brengen van de prototype. het prototype moet makkelijk in onderhoud zijn en we moeten het prototype kunnen integreren in bestaande systemen die eerder zijn ontworpen door ZorgCampus.

Goede communicatie wordt door ZorgCampus als een zeer belangrijk aspect tijdens het ontwerp van de prototype beschouwd. Het is belangrijk dat er op een regelmatige basis wordt gecommuniceerd met de doelgroepen van het systeem, zodat het systeem daadwerkelijk aansluit bij de behoefte en wensen van de betrokkenen (o.a. de programmeurs van ZorgCampus, de administratie-afdeling en de directie van ZorgCampus).

### ***Programmeervaardigheden***

|                                                                                                                                              |                                                                                                                                                                                |
|----------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"><li>- HTML</li><li>- CSS3</li><li>- Javascript</li><li>- PHP 4 en 5</li><li>- XML</li><li>- SOAP</li></ul> | <ul style="list-style-type: none"><li>- MySQL</li><li>- UML</li><li>- Object-Oriented programmeren</li><li>- FTP</li><li>- POP3/SMTP</li><li>- SSL</li><li>- Symfony</li></ul> |
|----------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### ***Systemen***

|                                                                                                                                                                                     |                                                                                                                            |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"><li>- Microsoft Windows Small Business Server 2003 met Terminal Server</li><li>- Active Directory</li><li>- Ubuntu</li><li>- Windows XP</li></ul> | <ul style="list-style-type: none"><li>- Firewall</li><li>- VPN</li><li>- Apache webserver</li><li>- Thin Clients</li></ul> |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------|

### ***Algemene vaardigheden***

- Humor

### ***Wat kunnen wij jou bieden?***

ZorgCampus heeft een eigen kleine IT-afdeling, die in samenwerking met de directeur, het ontwikkelproces van het project zal begeleiden. Er wordt tijdens het ontwerp vrijheid geboden om nieuwe functionaliteit en toevoegingen aan het systeem te bedenken. Hierbij is het wel vereist dat er een duidelijke communicatie is over het implementeren van nieuwe functionaliteit en toevoegingen aan het systeem.

ZorgCampus kan werkplaatsen aanbieden in een omgeving waarbij techniek en uitvoering plaatsvindt. Hierdoor is het gemakkelijk om het concept van het systeem aan te sluiten bij de verschillende wensen en behoeften van de verschillende afdelingen, die betrokken zijn bij het ontwerp van het prototype.

Stagevergoeding behoort tot de mogelijkheden.

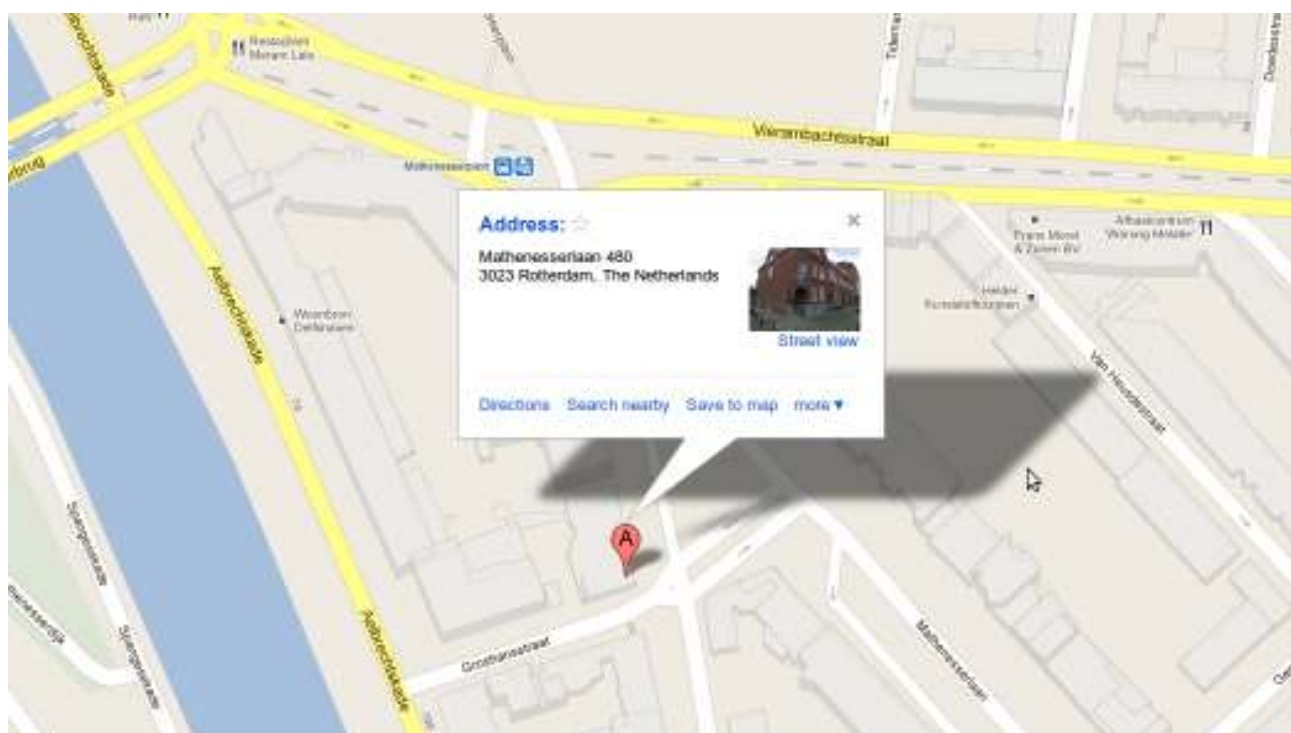
## Contactgegevens van ZorgCampus

### Algemeen

Contactpersoon: L. Amstelveen  
Telefoon: 088 881 23 33  
Mobiel: 06-17382865  
E-mailadres: info@zorgcampus.nl

### Bezoekadres

Straat: Mathenesserlaan 480  
Postcode en plaats: 3023HL te Rotterdam



## Appendix B

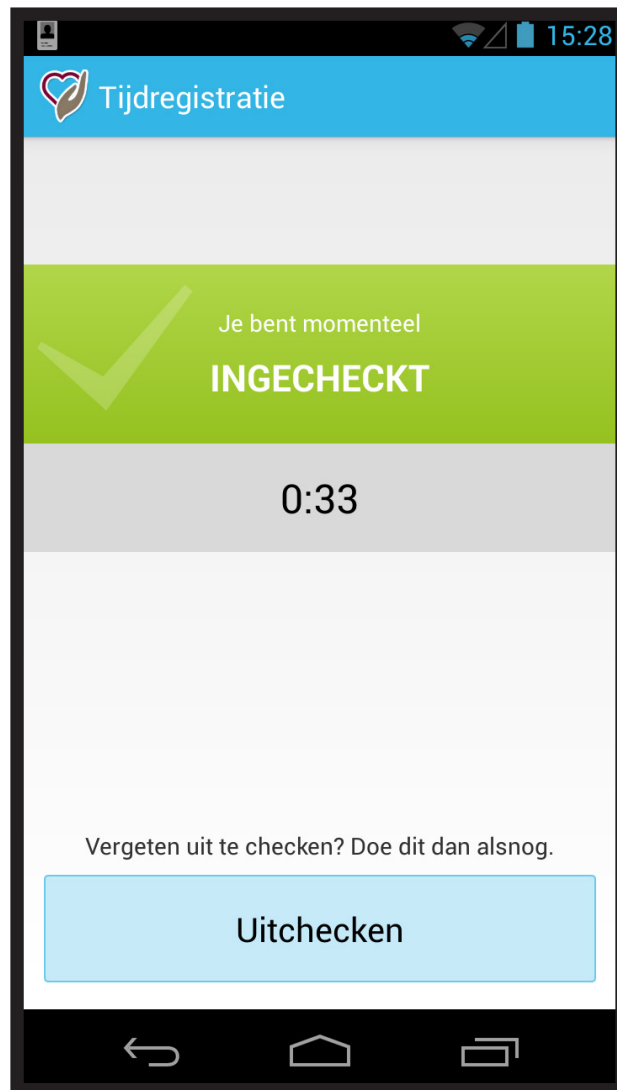
# User manual of the caregiver application



# ZORGCAMPUS

## Tijdregistratie met de smartphone

### Handleiding



*Voorbeeld*

# 1. Algemene werking van de telefoon

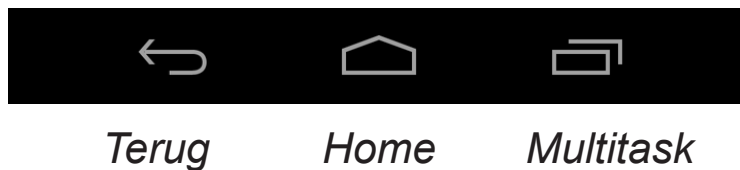
## 1.1 De telefoon inschakelen

De telefoon kan worden aangezet door op de knop aan de rechter zijkant te drukken.

## 1.2 Bediening

Onderin het scherm van de telefoon staan drie iconen:

Afbeelding 1



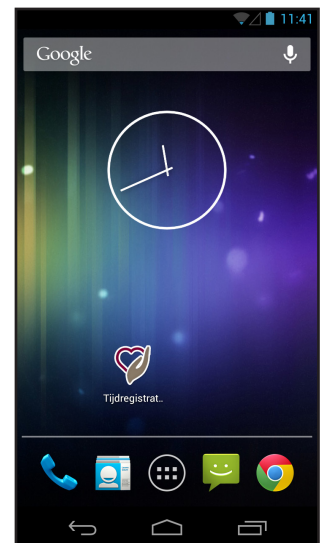
- **Terug-knop**  
Hiermee ga je één scherm terug.
- **Home-knop**  
Als je op deze knop drukt, gaat de telefoon altijd terug naar het beginscherm.
- **Multitask-knop**  
Met deze knop kun je wisselen tussen de actieve programma's (applicaties).

***Als je niet meer weet wat je moet doen omdat je een scherm niet herkent, kun je met de Home-knop altijd weer teruggaan naar het beginscherm.***

## 2. Werking van de Tijdregistratie-app

### 2.1 Het opstarten van de applicatie

In afbeelding 2 is het beginscherm van de telefoon weergegeven. Op het beginscherm staat een icoon van de applicatie, met daaronder de tekst 'Tijdregistratie'. Door op dit icoon te tikken, start de applicatie.



Afbeelding 2



Afbeelding 3

### 2.2 Het beginscherm van de applicatie

Zodra de applicatie is opgestart, wordt het scherm uit afbeelding 3 getoond. Dit scherm laat zien of u reeds bent ingecheckt of niet en zo ja, voor hoe lang.

Op het moment dat u start met werken bij cliënt, start u de applicatie en tikt u op de knop 'Inchecken'. Als u stopt met werken tikt u op de knop 'Uitchecken', die hier zal verschijnen nadat u bent ingecheckt.

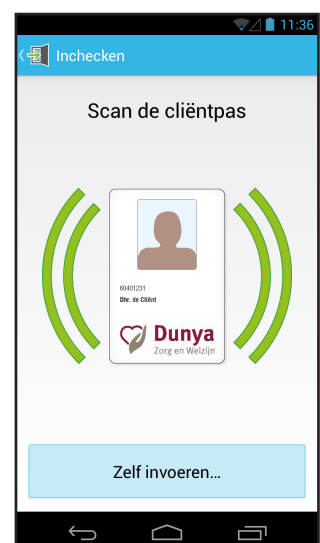
### 2.3 Inchecken met een cliëntpas

Om in te checken moet de telefoon eerst in het incheck-scherm staan. Dit scherm is te zien in afbeelding 4 en kan bereikt worden door in het beginscherm van de applicatie op 'Inchecken' te tikken.

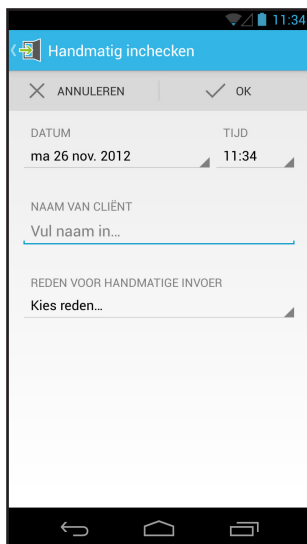
Houd vervolgens de cliëntpas op korte afstand achter de telefoon. Hierbij is het belangrijk dat u de pas stil houdt, totdat de telefoon teruggaat naar het beginscherm en aangeeft dat u ingecheckt bent.

#### **Niet gelukt?**

Als het inchecken mislukt, kunt u het nogmaals proberen. Werkt het andermaal niet, dan kunt u gebruikmaken van de handmatige invoermogelijkheid. Tik hiervoor op de knop 'Zelf invoeren...', onderaan het incheckscherm. Zie hiervoor ook 2.4.



Afbeelding 4



Afbeelding 5

## 2.4 Handmatig inchecken

Indien het inchecken met de cliëntpas niet mogelijk is (bijvoorbeeld: de pas werkt niet, of u bent vergeten in te checken), dan kunt u inchecktijd zelf invoeren. Hiervoor tikt u op de knop 'Zelf invoeren...', onderaan het incheckscherf zoals te zien in afbeelding 4.

In het volgende scherm (afbeelding 5) kunt u zelf aangeven wanneer u bent gestart met werken, bij welke cliënt en wat de reden is waarom u niet met de cliëntpas hebt ingecheckt.

## 2.5 Uitchecken

Wanneer u klaar bent met werken bij de cliënt, dient u weer uit te checken. Dit werkt op exact dezelfde manier als het inchecken.

## 2.6 Niet-verzonden tijdregistraties bekijken

De gegevens die vastgelegd worden bij het in- en uitchecken worden automatisch naar de zorginstelling verstuurd. Indien dit versturen om bepaalde redenen mislukt, zal op een later tijdstip opnieuw geprobeerd worden de gegevens te versturen.



Afbeelding 6

Als er nog gegevens op de telefoon staan die nog niet zijn verzonden, verschijnt er rechtsbovenin het beginscherm van de applicatie een knop (afbeelding 6). Daarin staat met een getal aangegeven hoeveel check-ins en check-outs er nog niet zijn verzonden. Door op deze knop te tikken, kunt u precies zien welke dit zijn.



Afbeelding 7



*Dit groene icoon betekent een check-in.*

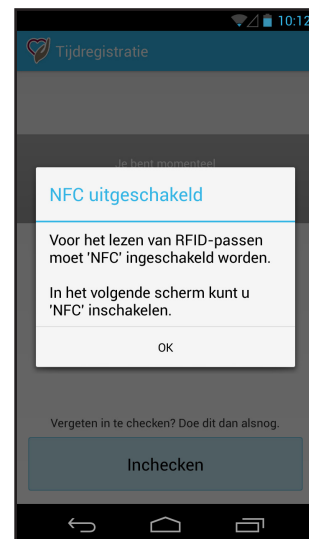


*Dit rode icoon betekent een check-uit.*

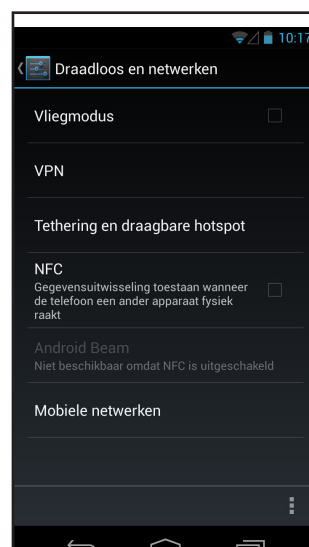
## 2.7 Instellingen van de telefoon

In sommige gevallen kan de applicatie vragen om een instelling van de telefoon aan te passen. Dit is te zien in afbeelding 8.

Verander in zo'n geval de instelling aan in het scherm dat volgt (zie afbeelding 9) en tik daarna op de Terug-knop. De melding zal dan verdwenen zijn.



Afbeelding 8



Afbeelding 9

## Appendix C

# Planning

### C.1 Initial planning

Figure C.1 shows the original planning as determined after roughly two weeks. The last week of this planning is week 10, so it does not include the week of *extra time*.

### C.2 Revised planning after milestone 3

Figure C.2 shows the planning for the last three weeks of the project. It includes week 11, the very last week of the project.

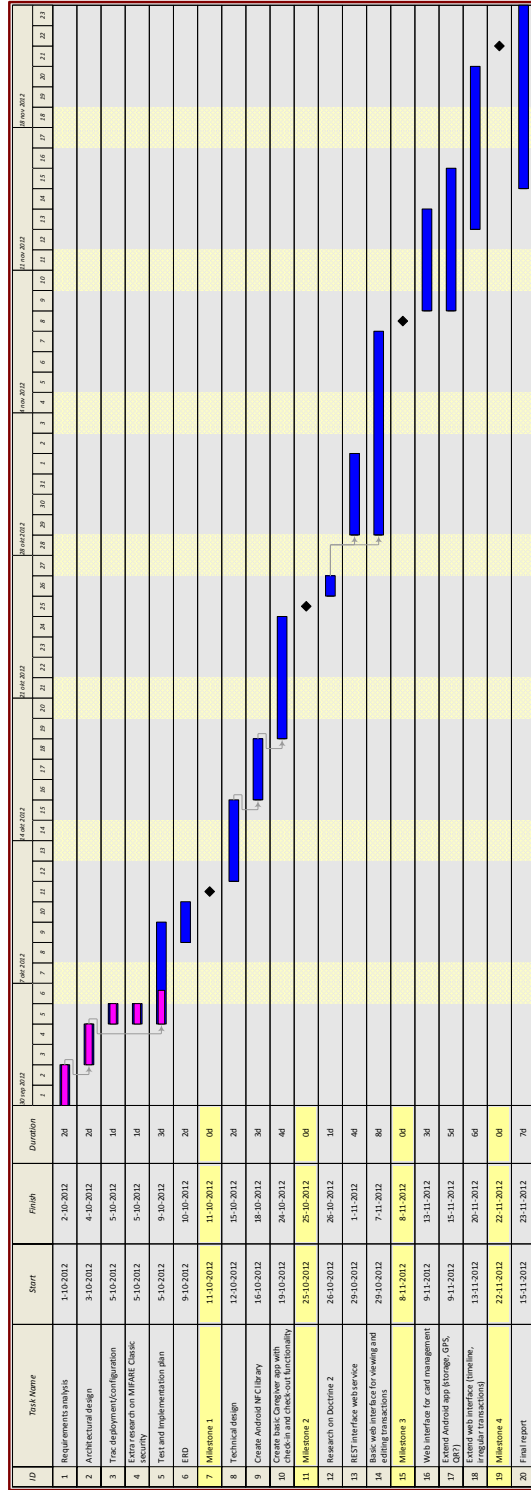


Figure C.1: The original planning.

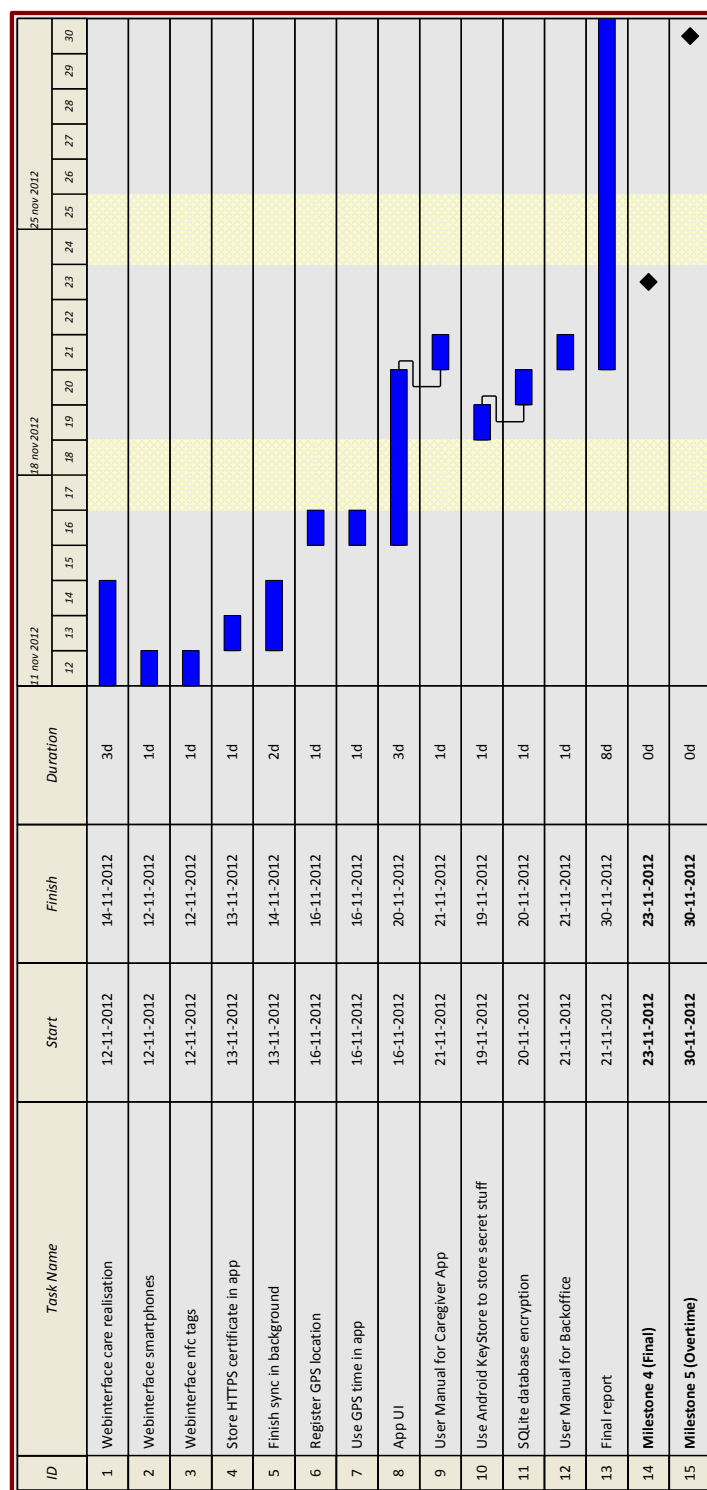


Figure C.2: The revised planning after milestone 3.

## Appendix D

# Requirements analysis

### D.1 Proposed system

The system, of which the requirements are proposed in this document, should allow the caregivers to check in when they start working with a client and check out when stop working with that client. These check-ins and check-outs should be collected centrally, to allow automatic generation of declarations. By approving the registered working times automatically when the system can guarantee the correctness, much of the coordinators' time will be saved.

### D.2 Functional requirements

#### Must have

- M1. The app registers working hours of the caregiver by means of checking in and out with the client's NFC tag.
- M2. The app registers the current time at every check-in and check-out.
- M3. The registered working hours of every caregiver are gathered centrally on a server.
- M4. A coordinator can view the working hours of all caregivers through a web interface.
- M5. Device IMEI numbers are registered centrally.
- M6. SIM card IMSI or serial numbers are registered centrally per device.
- M7. The caregiver is able to view the current status of the app (checked in or checked out, elapsed time since check-in, etc.).
- M8. When a caregiver checks in with a client without checking out with the previous client, the app should notify the caregiver and require to perform the needed checkout manually by entering the checkout time.

- M9. Manual input of check-ins and check-outs are possible, and requires time, name of the client and the reason for manual input to be entered.
- M10. A check-out should always use the exact same NFC tag as the corresponding check-in, if NFC is the check-out method.

#### **Should have**

- S1. The app registers the GPS position at every check-in and check-out.
- S2. Clients can have one or more NFC tags assigned to.
- S3. The coordinator is able to view NFC tags issued to clients.
- S4. The coordinator is able to edit NFC tags issued to clients.
- S5. The NFC tag has enough free space to store future information like the type of tag (client tag, tag belonging to client's family) or the client number. This could be needed for new applications of the tag.
- S6. The caregiver can view the internally stored transactions, which are waiting to be transferred.
- S7. The app can temporarily store transactions, if necessary, and upload them later (e.g. in case of an internet outage).
- S8. The app is resistant against malicious changes of the system time.
- S9. A coordinator can edit the working hours of all caregivers through a web interface.
- S10. A coordinator can approve/disapprove the working hours of all caregivers through a web interface.

#### **Could have**

- C1. NFC tags can be blocked and unblocked centrally. Transactions with blocked tags are allowed, but should be marked in the management interface.
- C2. The coordinator is able to view all irregular transactions on a certain day (e.g. transactions not matching the predefined schedule, transactions without GPS coordinates, manual transactions, etc.).
- C3. The app can be locked down if the registered IMEI or IMSI number does not match the device and SIM.
- C4. QR-codes/barcodes can be used when NFC is not available.
- C5. The coordinator can view a timeline/grid of a day, with caregivers on one axis and the time of day on the other axis.

### **Want to have**

- W1. The coordinator can view the current status of the app of a specific caregiver.
- W2. The app reports every 15 minutes to the server, even if no stored transactions are pending. (Because ‘not checked in’ is also a valid status.)
- W3. The coordinator can view failure rates of check-in methods, sensors, connections of a device.
- W4. The app can receive the work schedule, so the caregiver can be alarmed when not checked in on time.

## **D.3 Non-functional requirements**

- NF1. The caregiver app must have an intuitive interface, no training or manual should be needed to understand how the app works.
- NF2. The code must be usable for other programmers and documented in English.
- NF3. The apps and backoffice manual must be in English and in Dutch.
- NF4. The program code must be tested well.
- NF5. The app is built for the Android 4.x OS.
- NF6. All communication between app and server is encrypted securely.

## **D.4 Constraints (pseudo-requirements)**

- PR1. NFC tags can have only one client assigned to.
- PR2. The smartphone can be used by multiple caregivers in a week.
- PR3. The yet to be bought Android smartphones must be powerful enough for future purposes.
- PR4. The backoffice module must be written in PHP 5.3, using the Doctrine 2 DBAL and ORM.
- PR5. The backoffice module should store its information in a MySQL 5 database.

## **D.5 Use cases**

The system has two main actors: the caregiver and the care coordinator. For both actors, the most important use cases have been defined below.

### D.5.1 Caregiver

The caregiver should be able to check-in, -out and view the current status of the app. These use cases are shown in figure D.1.

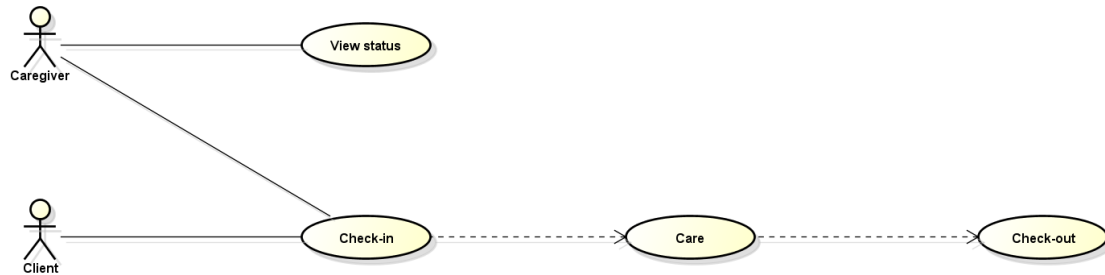


Figure D.1: Use cases care registration

---

#### CHECK-IN

---

**Actors:** Caregiver

**Goals:** Registering the starting time of working with a client

**Preconditions:** The current status is *Not checked in*

**Related use cases:** Check-out

| Actor actions           | System responses                                                  |
|-------------------------|-------------------------------------------------------------------|
| Unlocks smartphone      |                                                                   |
|                         | Phone displays home screen                                        |
| Starts caregiver app    |                                                                   |
|                         | Caregiver app starts and enters NFC read mode automatically       |
| Scans client's RFID tag |                                                                   |
|                         | App asks what to do with the tag (check in, check out, or cancel) |
| Touches check-in button |                                                                   |
|                         | Sends transaction to server and confirms check in                 |

---

---

---

## CHECK-OUT

**Actors:** Caregiver

**Goals:** Registering the finishing time of working with a client

**Preconditions:** The current status is *Checked in (with this client)*

**Related use cases:** Check-in

| Actor actions            | System responses                                                  |
|--------------------------|-------------------------------------------------------------------|
| Unlocks smartphone       |                                                                   |
|                          | Phone displays home screen                                        |
| Starts caregiver app     |                                                                   |
|                          | Caregiver app starts and enters NFC read mode automatically       |
| Scans client's RFID tag  |                                                                   |
|                          | App asks what to do with the tag (check in, check out, or cancel) |
| Touches check-out button |                                                                   |
|                          | Sends transaction to server and confirms check out                |

---

---

---

---

## VIEW STATUS

**Actors:** Caregiver

**Goals:** Viewing whether he has already checked in or checked out

**Related use cases:** Check-in, Check-out

| Actor actions        | System responses                                             |
|----------------------|--------------------------------------------------------------|
| Unlocks smartphone   |                                                              |
|                      | Phone displays home screen                                   |
| Starts caregiver app |                                                              |
|                      | Caregiver app starts and show the current status immediately |
| Views status         |                                                              |

---

---

### D.5.2 Coordinator

The coordinator should be able to view and edit all registered working times, spot irregularities, view the status of caregivers and manage the RFID cards issued to clients. Figure D.2 shows an overview of these use cases.

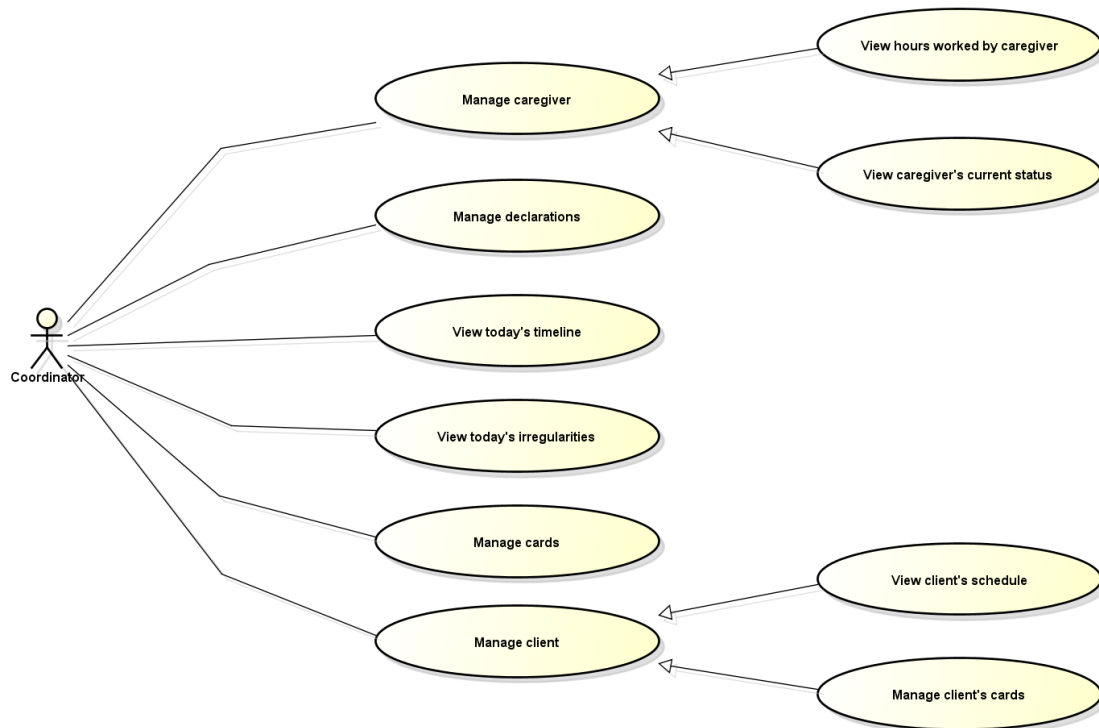


Figure D.2: Use cases Coordinator

---



---

#### VIEW HOURS WORKED BY CAREGIVER

---

**Actors:** Coordinator

**Goals:** View the hours worked by caregiver

**Related use cases:** View caregiver's current status

| Actor actions                                      | System responses                            |
|----------------------------------------------------|---------------------------------------------|
| Logs in to backoffice system                       | Shows backoffice front page                 |
| Clicks on <i>Caregivers</i> in the navigation menu | Shows a list of caregivers                  |
| Clicks on a specific caregiver                     | Shows an overview page about that caregiver |
| Clicks on <i>Working times</i>                     | Shows hours worked of this caregiver        |

---

---

---

## VIEW CAREGIVER'S CURRENT STATUS

**Actors:** Coordinator

**Goals:** Viewing the current status of a caregiver

**Related use cases:** View hours worked by caregiver

| Actor actions                                      | System responses                            |
|----------------------------------------------------|---------------------------------------------|
| Logs in to backoffice system                       |                                             |
|                                                    | Shows backoffice front page                 |
| Clicks on <i>Caregivers</i> in the navigation menu |                                             |
|                                                    | Shows a list of caregivers                  |
| Clicks on a specific caregiver                     |                                             |
|                                                    | Shows an overview page about that caregiver |
| Clicks on <i>View current status</i> button        |                                             |
|                                                    | Shows the current status of this caregiver  |

---

---

## VIEW TODAY'S TIMELINE

**Actors:** Coordinator

**Goals:** Spotting irregular transactions

**Summary:** The coordinator views the today's transactions of the caregivers combined in a timeline view. This timeline has an overlay with the original schedules of the caregiver.

**Related use cases:** View hours worked by caregiver

| Actor actions                                                | System responses               |
|--------------------------------------------------------------|--------------------------------|
| Logs in to backoffice system                                 |                                |
|                                                              | Shows backoffice front page    |
| Clicks on <i>Transaction timeline</i> in the navigation menu |                                |
|                                                              | Shows the transaction timeline |

---

---

## REVIEW TODAY'S IRREGULARITIES

**Actors:** Coordinator

**Goals:** Spotting irregular transactions, Approving an irregular transaction

**Related use cases:** Correct irregularities

| Actor actions                                                  | System responses                       |
|----------------------------------------------------------------|----------------------------------------|
| Logs in to backoffice system                                   |                                        |
|                                                                | Shows backoffice front page            |
| Clicks on <i>Irregular transactions</i> in the navigation menu |                                        |
|                                                                | Shows a list of today's irregularities |
| Clicks on the <i>Approve</i> button of a transaction           |                                        |
|                                                                | Saves and confirms the modification    |

---

---

## CORRECT IRREGULARITIES

**Actors:** Coordinator

**Goals:** Correcting an irregular transaction

**Related use cases:** Review irregularities

| Actor actions                                                  | System responses                         |
|----------------------------------------------------------------|------------------------------------------|
| Logs in to backoffice system                                   |                                          |
|                                                                | Shows backoffice front page              |
| Clicks on <i>Irregular transactions</i> in the navigation menu |                                          |
|                                                                | Shows a list of today's irregularities   |
| Clicks on the <i>Edit</i> button of a transaction              |                                          |
|                                                                | Shows a form the correct the transaction |
| Fills out the form and clicks <i>Save</i> button               |                                          |
|                                                                | Saves and confirms the modification      |

---

---

## MANAGE CLIENT'S CARDS

**Actors:** Coordinator

**Goals:** Managing the current RFID tags issued to a client

| Actor actions                                   | System responses                           |
|-------------------------------------------------|--------------------------------------------|
| Logs in to backoffice system                    |                                            |
|                                                 | Shows backoffice front page                |
| Clicks on <i>Clients</i> in the navigation menu |                                            |
|                                                 | Shows a list of clients                    |
| Clicks on a specific client                     |                                            |
|                                                 | Shows an overview page about that client   |
| Clicks on a card's <i>Edit</i> button           |                                            |
|                                                 | Shows a form to edit the card details page |
| Clicks on <i>Save</i>                           |                                            |
|                                                 | Saves and confirms the modification        |

---

# Appendix E

## Architectural design

### E.1 Introduction

This document describes the architectural design of the caregiving time registration system.

#### E.1.1 Purpose of the system

With the home care registration system, caregivers must be able to register their working times by checking in and out with a smartphone and the personal RFID tag of each client they visit. The check-in and check-out times must be collected by a central web server, so the care coordinators can manage these transactions.

The system should replace the current system of manually filling paper forms with working times. By doing this, it should save valuable time of the caregiver, and time of the coordinator as well.

For a more detailed description on the purpose of the system, please refer to the requirements analysis document.

### E.2 Recommended software architecture

This section describes the recommended software architecture of the caregiving time registration system. It starts with an overview of the components and then continues with the overall behavior of the system.

#### E.2.1 Overview and subsystems

The system mainly consists of four physical components: an RFID tag, an Android-based smartphone, a central server and a computer. These components are depicted in figure E.1.

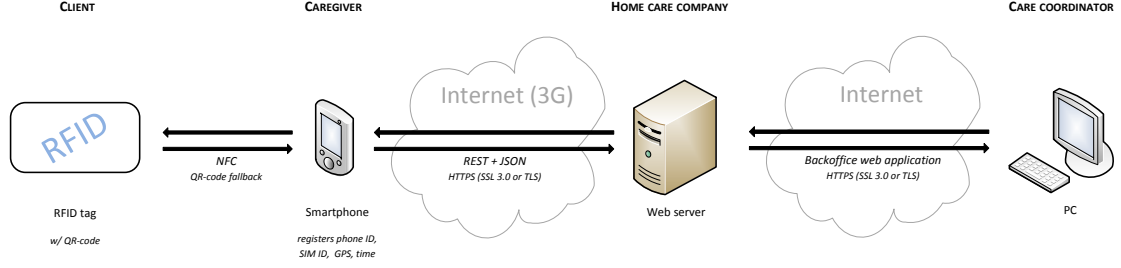


Figure E.1: Physical components of the system

## RFID tag

The RFID tag will be a plastic card with the credit card form factor. A QR-code on the card can serve as a fallback method when RFID fails to operate. It is equipped with the MIFARE Classic chip [15] with at least 1K of EEPROM. A 4K version to be used to ensure future extensibility.

By default, the MIFARE Classic contains a 4-byte UID. Since the 4-byte UID pools of manufacturers like NXP and Infineon are depleting soon, they are starting to reuse old 4-byte UIDs [29, 1]. This is an unwanted situation, as the reliability of the UIDs drops. However, a *next generation* version of the chip, with a 7-byte UID, exists [14]. These chip versions are identified by an *X* appended to the model name (*MF1SyyyyX* vs. *MF1Syyyy*). For this project, only next generation chips should be used, since the uniqueness of the UID is one of the most important aspects of the tag.

The information that will be stored on the tag is displayed in table E.1.

| Value           | Description                                                   |
|-----------------|---------------------------------------------------------------|
| UID             | 7-byte Unique ID of the tag, already available.               |
| Record provider | Name of the issuer of the record.                             |
| Spec version    | Data specification version number.                            |
| Status          | Status of the record (e.g. <i>active</i> or <i>blocked</i> ). |
| Date issued     | The date on which the record has been created.                |
| Valid from      | The date from which the record is valid.                      |
| Valid until     | The date until which the record is valid.                     |
| Hash #1         | Secret-salted hash to be validated by the smartphone.         |
| Hash #2         | Secret-salted hash to be validated by the server.             |

Table E.1: Information contained by the RFID tag.

## Smartphone

A caregiver will use a smartphone based on Android 4.x, with an dedicated app to register working times. Because the Android NFC API structure has changed dramatically with

the introduction of Android 4.0 (Ice Cream Sandwich, API level 14), the app won't be compatible with pre-ICS devices without a lot of compatibility code. This should not be a problem, since it hasn't been decided yet which phone model will be used. Most of the NFC enabled smartphones come with Android 4.x, nowadays.

## **Server**

The web service part that runs on the central server, will have plug into an existing, custom backoffice system. This system has been built with PHP 5.3, using the Doctrine 2 [25] DBAL and ORM and parts of the Symfony framework [26].

## **Coordinator PC**

A care coordinator has to be able to view and administer all check-in and -out transaction on a computer. Since there already is an existing backoffice environment, this will be made possible through a web interface.

### **E.2.2 Hardware/software mapping**

The mapping of the various software subsystems onto the hardware is depicted in figure E.2. Each smartphone runs the Android app and has a SQLite database offered by the Android OS. The app connects to the web server, which runs the backoffice system.

### **E.2.3 Communication flow**

There are three different communication paths: RFID tag – smartphone, smartphone – server and coordinator PC – server. The specification of the data sent over these connections is given below.

#### **RFID tag – smartphone**

The following data is sent from the RFID tag to the smartphone:

- Record creation date
- Record valid from
- Record valid until
- Record specification version
- Record status
- Record provider
- Hash #1
- Hash #2

By overhearing the communications between the tag and the smartphone, the RFID read-key or the transmitted data may be intercepted [20]. This is a known issue of the MIFARE Classic chip.

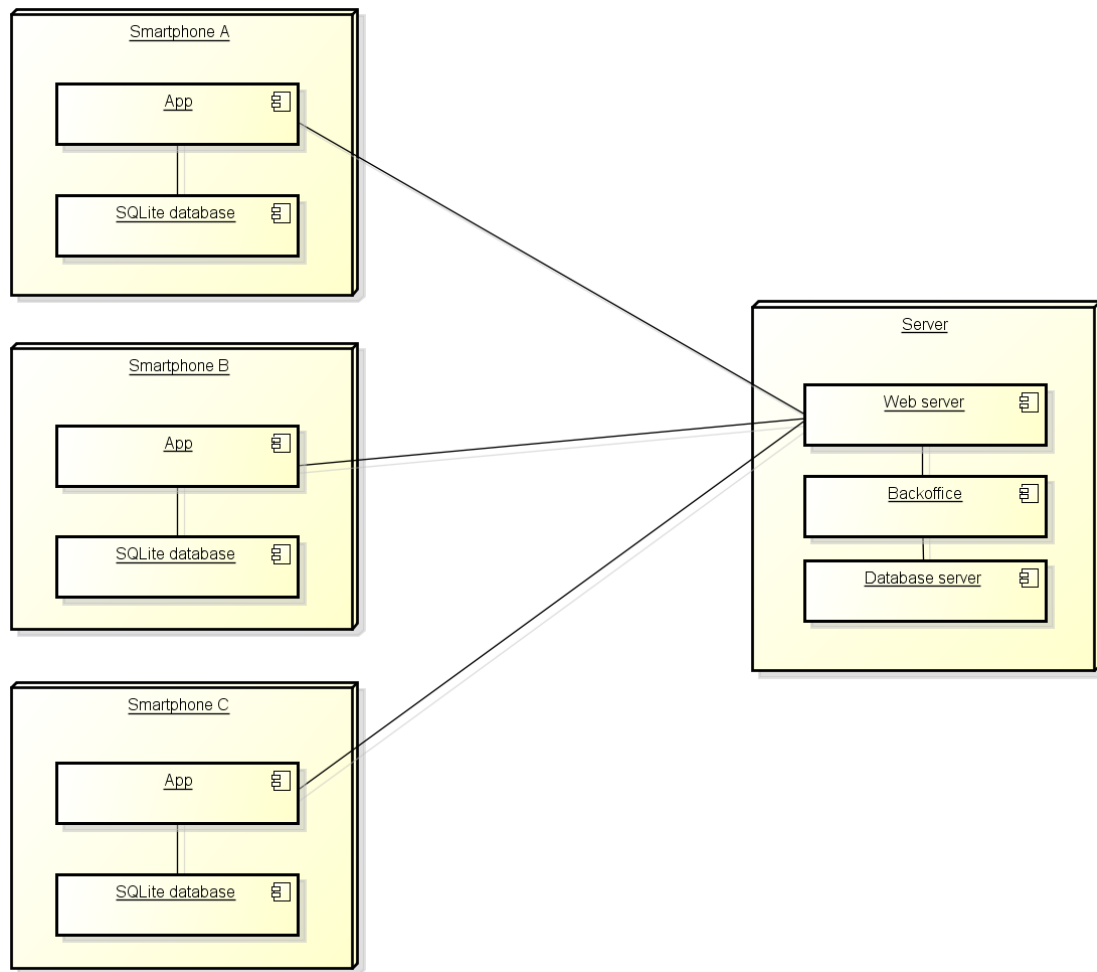


Figure E.2: Hardware/Software mapping

### Smartphone – server

The following data is sent from the smartphone to the web server when reporting a transaction. As long as the transaction is not acknowledged by the server, it must be retransmitted.

- RFID tag UID
- Hash #2
- Check-in or -out time
- GPS coordinates
- Input method (RFID/QR-code/manual)
- SIM ID
- Smartphone device ID
- Whether the transaction is sent directly or has been stored internally first

The server handles multiple types of requests, using the REST model. Data in REST-requests will be encoded as JSON. Examples of these requests could be:

- Post transaction
- Get server-side time
- Put smartphone status

The server sends back the requested data for a GET-request, or an acknowledgement or error message in case of a POST-, PUT- or DELETE-request.

### **Coordinator PC – server**

When the coordinator connects to the central web server, this server sends web application data to the coordinator's PC. This includes information about the transactions registered by the smartphones and should be considered sensitive. This data will be protected by HTTPS, just like the entire existing backoffice system.

#### **E.2.4 Global resource handling and access control**

The system has two different roles: the caregiver role and the coordinator role. These roles must have different permissions.

The caregiver must be able to:

- Check in with the smartphone app.
- Check out with the smartphone app.
- View the current status.

The coordinator must be able to:

- View and manage submitted transactions.
- View and manage RFID tags issued to clients.
- View current status of a caregiver.
- View irregularities of submitted transactions (e.g. manual input).

All permissions will be enforced by the web service. The existing backoffice already contains an ACL layer to do this.

#### **E.2.5 Exceptional situations**

There are several situations in which the regular behavior of the system is obstructed. The most important of these situations are mentioned below. For each of these is defined which action should follow in such a case.

### **Provider network (3G) is down / Server malfunction**

**Situation 1** At the time of a transaction (e.g. check-in), the internet connection is down. This means the actual server-side time cannot be requested and the transaction cannot be uploaded.

**Situation 2** At the time of a transaction (e.g. check-in), the server is not responding correctly. The server could be down, unreachable, or in any other kind of state in which it is not storing transactions correctly.

**Action** The transaction must be stored on the smartphone and when the connection restores, it should be retransmitted. The GPS time should be used if available, or the local time otherwise.

### **The internet connection disabled in OS**

**Situation** At the time of a transaction (e.g. check-in), the internet connection is disabled in the Android OS.

**Action** The application must provide a solution to let the user easily enable the internet connection. If the connection remains disabled, the caregiver should be told to use a paper declaration form.

### **NFC communication with the RFID tag is not working**

**Situation** At the time of a transaction (e.g. check-in), it is not possible to read the RFID tag with the phone.

**Action** The caregiver should be able to use the smartphone camera in combination with the QR-code on the tag instead. Another possibility is to use the manual input method in the application.

## **E.2.6 Security threats**

Each device and every communication path brings security threats to the system. These risks are assessed below and in most cases, a solution is proposed. In some situations, the risk could be acceptable when the impact of a security breach is small.

### **Unauthorized reading of the RFID tag**

**Issue** Information may be acquired by reading the RFID tag with hardware or software other than the own smartphone and app.

**Solution** In order to make it harder to perform unauthorized read operations, the data blocks on the RFID tag should be encrypted with the CRYPTO-1 encryption algorithm. This proprietary encryption algorithm is the only encryption algorithm supported natively by the MIFARE Classic chip. As this algorithm has already been breached [30], encrypting the data with it will by no means guarantee protection against unauthorized reads. This will not be a problem, because since there is no need to, no sensitive information should be stored on the tag itself.

### Unauthorized writing on the RFID tag

**Issue** Data may be written to the RFID tag by hardware or software other than the own smartphone and app. This could be a point of access for an attack, or it could just make the tag unusable.

**Solution** The data blocks of the MIFARE Classic chip should be made writeable only to those who possess the correct write key. This is a specific feature of the chip. Since, for the caregiver app, there is no need to write data to the tag, the app should not include the write key. As the security of the MIFARE Classic has already been breached [30], there will be no guarantee the data on the tag hasn't been overwritten. Therefore, the caregiver app should always make sure the secret-salted checksum on the card is correct.

### Eavesdropping on the NFC connection

**Issue** By overhearing the communications between the tag and the smartphone, the RFID read-key or the transmitted data may be intercepted [20]. This is a known issue of the MIFARE Classic chip.

**Considerations** As the attacker needs to be near the RFID tag to overhear the communications, this issue won't be problematic. The RFID tag will always be inside the home of the client, when it is being read. Besides, there exist other ways to retrieve the RFID read- or write-key (and with those, also the data) [20] and no highly sensitive information will be stored on the tag.

### Copying of the RFID tag

**Issue** Caregiver app has to notice the difference between a copied tag and a legitimate tag and deny the copied ones.

**Solution** This will be done by storing a hash (hash #1) of the UID of the card, prefixed with a secret salt.

## **Eavesdropping on the internet connection between smartphone and server**

**Issue** By eavesdropping on the internet connection, valuable information can be collected and man-in-the-middle or replay attacks can be performed.

**Solution** For this reason, all communication between a smartphone and the server should be securely encrypted using the HTTPS-protocol. This prevents eavesdropping, man-in-the-middle attacks and replay attacks between smartphone and server [28].

## **Reading of the smartphone storage**

**Issue** Since transactions can be stored locally when the internet connection is down, information about these transactions may be obtained by reading the storage of the smartphone.

**Considerations** All information that is stored locally by the application has to be encrypted. Since the stored information enters the smartphone unencrypted, encryption has to take place on the smartphone itself. As a result of this, the key needs to be known on the device itself to store the data encryptedly. Since entering a PIN/password at encryption time would be too obtrusive and probably weak against brute-force attacks, and hardcoding the key would make the application vulnerable to decompilation, there is no real solution for this issue.

The device's storage should be considered as a possible point of access for an attack on the system. Therefore, as little sensitive information as possible should be stored on the phone.

## **Manipulation of the smartphone storage**

**Issue** Since transactions can be stored locally when the internet connection is down and sent again when the connection turns live, the transaction store may be manipulated. When the storage encryption has been breached, like in E.2.6, a transaction could be altered or injected.

**Considerations** For successfully injecting a transaction into the storage, the attacker needs to have had access to an RFID tag of a client, because the transaction needs at least the UID and hash #2 to be validated on the server. For manipulating an existing transaction, this isn't necessary, as these two values are already stored. Scenarios like these seem highly unlikely, mainly because fake transactions will probably get noticed by coordinators within a week, for example, when clients will start to complain about not receiving care.

## **Reverse engineering the Android application**

**Issue** By transferring the application package to a computer, the app may be decompiled or reverse engineered. This way, several keys could be acquired. These keys include

the key for reading the data blocks of the RFID tag, the secret salt of hash #1, the key for decrypting stored transactions and the key for authenticating the app to the server.

**Considerations** Since it is hard or probably even impossible to make sure to application package doesn't leave the smartphone device and isn't decompiled, the application code should not contain any secret information that is vital to the correct and secure operation of the entire system.

However, the decompilation scenario is an unlikely scenario, because the attacker needs access to a trusted device. Therefore, a number of non-vital keys could be hard-coded into the application package. These are: the read-key of the RFID tag, the secret salt of hash #1 to validate an RFID tag on the smartphone and a key to authenticate the smartphone to the server. Because entering a password in the app would be too obtrusive, there is no point in using the Android key store.

The application package should *never* contain the write-key of the RFID tag and the secret salt of hash #2, as these are more critical for system security.

### **Manipulating transactions by changing the device time**

**Issue** By modifying the device time, one could influence a check-in or check-out time.

**Solution** When the internet connection is live, the server-side time should be used when registering a transaction. In times of a dead connection, the device could compare the device time to the GPS time. If GPS time is unavailable too, then the device time should be used, but the transaction should be marked when storing it on the server side.

# Appendix F

## Technical design

### F.1 Introduction

This document describes the technical design of the caregiving time registration system.

### F.2 Mifare Classic data layout

From the requirements analysis followed that the NFC tag that would be assigned to a client, should be usable for future purposes. In order to provide a flexible solution that allows multiple applications to read from and write to the tag, without interfering, we have designed a record system.

#### F.2.1 Key features of the Mifare Classic tag

These are some key features of the Mifare Classic, according to the datasheet provided by NXP [15].

- An ID per tag. We chose the version with a 7-byte Unique ID (*UID*).
- 16 sectors of 4 blocks (1K version) / 32 sectors of 4 blocks and 8 sectors of 16 blocks (4K version).
- 16 bytes per block.
- Two keys per sector, key A and B, of which the access conditions can be configured. This configuration is stored in so-called sector trailer: the last block of each sector.

#### F.2.2 Record system

The system allows each application to have its own record on the tag. Such a record consists of a *unique* 3-byte identifier and a payload.

**Header sector** The first sector (sector 0) will be called the *header sector*. Block 0, the first data block, contains manufacturer data by default and is not writeable. The next block can be used to store an arbitrary string or value. In the third block of this sector, information is stored about which sectors are in use. Every sector is represented by one bit.

**Index sector** The three data blocks of the second sector (sector 1) is reserved to store the index table. An entry (4 bytes) in this table consists of the unique record id (3 bytes) and the number (1 byte) of the sector in which the payload this record begins. As the index sector contains three blocks of 16 bytes each, the index can at most contain 12 references to records.

**Sector spanning** By implementing a linked list structure, the record payload can span multiple sectors. Byte 9 of the sector trailer is unused by default [15], so we use this byte to store the number of the sector in which the payload continues, if any.

**Access** Key A will be configured to allow reading of the data blocks and byte 9 of the sector trailer. Key B allows these operations too, but also allows writing to every block. Each record can have its own keys, so access control can happen on a per-application basis. The first two sectors will have predefined keys though, as they have to be accessible by all applications.

### F.3 Care record

The time registration application defines a record type for use the record system introduced in section F.2. From now on, we'll call this record type *care record*. A care record consists *entries*, which are key-value pairs. The entries, in turn, are composed of a 2-byte key, the length of the value and the value itself. Entries are stored successively, sorted ascendingly by their key.

**Integrity** The entries, preceded with a SHA-256 checksum, form the payload of the care record. This checksum ensures the integrity of the entries and is generated using the UID of the tag and the concatenation of all entries, just like they are stored on the tag. As the length of a SHA-256 hash is 256 bits (32 bytes), the checksum will cost exactly two blocks.

**Record end** The remaining bytes of the last sector that is in use by the record, should be set to 0x00. As a result, entry keys can **not** be empty (0x0000)!

Layout of the Mifare Classic EEPROM

| SECTOR                                                                                                                                                                                                                                                                                                                                           |                                 |   | SECTOR FUNCTION    |   |   | BLOCK |   |   | BLOCK FUNCTION |    |    | BYTE WITHIN BLOCK |                   |    |    |                                                                                                                                                                                                                            |  |              |  |  |  |  |                        |  |  |  |  |                       |  |  |  |  |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------|---|--------------------|---|---|-------|---|---|----------------|----|----|-------------------|-------------------|----|----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--------------|--|--|--|--|------------------------|--|--|--|--|-----------------------|--|--|--|--|
| 0                                                                                                                                                                                                                                                                                                                                                | 1                               | 2 | 3                  | 4 | 5 | 6     | 7 | 8 | 9              | 10 | 11 | 12                | 13                | 14 | 15 |                                                                                                                                                                                                                            |  |              |  |  |  |  |                        |  |  |  |  |                       |  |  |  |  |
| 0<br>0x00                                                                                                                                                                                                                                                                                                                                        | Header sector                   | 0 | Manufacturer block |   |   |       |   |   |                |    |    |                   | Manufacturer data |    |    |                                                                                                                                                                                                                            |  |              |  |  |  |  |                        |  |  |  |  |                       |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Header string      |   |   |       |   |   |                |    |    |                   | Header Key A      |    |    |                                                                                                                                                                                                                            |  |              |  |  |  |  |                        |  |  |  |  |                       |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Sector allocation  |   |   |       |   |   |                |    |    |                   | Header Key B      |    |    |                                                                                                                                                                                                                            |  |              |  |  |  |  |                        |  |  |  |  |                       |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Sector trailer     |   |   |       |   |   |                |    |    |                   | Header Key B      |    |    |                                                                                                                                                                                                                            |  |              |  |  |  |  |                        |  |  |  |  |                       |  |  |  |  |
| 1<br>0x01                                                                                                                                                                                                                                                                                                                                        | Index sector                    | 0 | Record #1 ID       |   |   |       |   |   |                |    |    |                   | #1 Sector         |    |    |                                                                                                                                                                                                                            |  | Record #2 ID |  |  |  |  | Access bits (selected) |  |  |  |  | Access bits (default) |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Index Key A        |   |   |       |   |   |                |    |    |                   | Index Key A       |    |    |                                                                                                                                                                                                                            |  | Index Key B  |  |  |  |  | Index Key B            |  |  |  |  |                       |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Record #1 ID       |   |   |       |   |   |                |    |    |                   | #1 Sector         |    |    |                                                                                                                                                                                                                            |  | Record #2 ID |  |  |  |  | Access bits (selected) |  |  |  |  | Access bits (default) |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Index Key A        |   |   |       |   |   |                |    |    |                   | Index Key A       |    |    |                                                                                                                                                                                                                            |  | Index Key B  |  |  |  |  | Index Key B            |  |  |  |  |                       |  |  |  |  |
| 2<br>0x02                                                                                                                                                                                                                                                                                                                                        | Example of an unallocated block | 0 | Record #1 ID       |   |   |       |   |   |                |    |    |                   | #1 Sector         |    |    |                                                                                                                                                                                                                            |  | Record #2 ID |  |  |  |  | Access bits (selected) |  |  |  |  | Access bits (default) |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Index Key A        |   |   |       |   |   |                |    |    |                   | Index Key A       |    |    |                                                                                                                                                                                                                            |  | Index Key B  |  |  |  |  | Index Key B            |  |  |  |  |                       |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Record #1 ID       |   |   |       |   |   |                |    |    |                   | #1 Sector         |    |    |                                                                                                                                                                                                                            |  | Record #2 ID |  |  |  |  | Access bits (selected) |  |  |  |  | Access bits (default) |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Index Key A        |   |   |       |   |   |                |    |    |                   | Index Key A       |    |    |                                                                                                                                                                                                                            |  | Index Key B  |  |  |  |  | Index Key B            |  |  |  |  |                       |  |  |  |  |
| 3<br>0x03                                                                                                                                                                                                                                                                                                                                        | Allocated for Record #1         | 0 | Record #1 ID       |   |   |       |   |   |                |    |    |                   | #1 Sector         |    |    |                                                                                                                                                                                                                            |  | Record #2 ID |  |  |  |  | Access bits (selected) |  |  |  |  | Access bits (default) |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Index Key A        |   |   |       |   |   |                |    |    |                   | Index Key A       |    |    |                                                                                                                                                                                                                            |  | Index Key B  |  |  |  |  | Index Key B            |  |  |  |  |                       |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Record #1 ID       |   |   |       |   |   |                |    |    |                   | #1 Sector         |    |    |                                                                                                                                                                                                                            |  | Record #2 ID |  |  |  |  | Access bits (selected) |  |  |  |  | Access bits (default) |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Index Key A        |   |   |       |   |   |                |    |    |                   | Index Key A       |    |    |                                                                                                                                                                                                                            |  | Index Key B  |  |  |  |  | Index Key B            |  |  |  |  |                       |  |  |  |  |
| 4<br>0x04                                                                                                                                                                                                                                                                                                                                        | Example of an unallocated block | 0 | Record #1 ID       |   |   |       |   |   |                |    |    |                   | #1 Sector         |    |    |                                                                                                                                                                                                                            |  | Record #2 ID |  |  |  |  | Access bits (selected) |  |  |  |  | Access bits (default) |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Index Key A        |   |   |       |   |   |                |    |    |                   | Index Key A       |    |    |                                                                                                                                                                                                                            |  | Index Key B  |  |  |  |  | Index Key B            |  |  |  |  |                       |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Record #1 ID       |   |   |       |   |   |                |    |    |                   | #1 Sector         |    |    |                                                                                                                                                                                                                            |  | Record #2 ID |  |  |  |  | Access bits (selected) |  |  |  |  | Access bits (default) |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Index Key A        |   |   |       |   |   |                |    |    |                   | Index Key A       |    |    |                                                                                                                                                                                                                            |  | Index Key B  |  |  |  |  | Index Key B            |  |  |  |  |                       |  |  |  |  |
| 5<br>0x05                                                                                                                                                                                                                                                                                                                                        | Allocated for Record #2         | 0 | Record #1 ID       |   |   |       |   |   |                |    |    |                   | #1 Sector         |    |    |                                                                                                                                                                                                                            |  | Record #2 ID |  |  |  |  | Access bits (selected) |  |  |  |  | Access bits (default) |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Index Key A        |   |   |       |   |   |                |    |    |                   | Index Key A       |    |    |                                                                                                                                                                                                                            |  | Index Key B  |  |  |  |  | Index Key B            |  |  |  |  |                       |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Record #1 ID       |   |   |       |   |   |                |    |    |                   | #1 Sector         |    |    |                                                                                                                                                                                                                            |  | Record #2 ID |  |  |  |  | Access bits (selected) |  |  |  |  | Access bits (default) |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Index Key A        |   |   |       |   |   |                |    |    |                   | Index Key A       |    |    |                                                                                                                                                                                                                            |  | Index Key B  |  |  |  |  | Index Key B            |  |  |  |  |                       |  |  |  |  |
| 6<br>0x06                                                                                                                                                                                                                                                                                                                                        | Allocated for Record #1         | 0 | Record #1 ID       |   |   |       |   |   |                |    |    |                   | #1 Sector         |    |    |                                                                                                                                                                                                                            |  | Record #2 ID |  |  |  |  | Access bits (selected) |  |  |  |  | Access bits (default) |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Index Key A        |   |   |       |   |   |                |    |    |                   | Index Key A       |    |    |                                                                                                                                                                                                                            |  | Index Key B  |  |  |  |  | Index Key B            |  |  |  |  |                       |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Record #1 ID       |   |   |       |   |   |                |    |    |                   | #1 Sector         |    |    |                                                                                                                                                                                                                            |  | Record #2 ID |  |  |  |  | Access bits (selected) |  |  |  |  | Access bits (default) |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Index Key A        |   |   |       |   |   |                |    |    |                   | Index Key A       |    |    |                                                                                                                                                                                                                            |  | Index Key B  |  |  |  |  | Index Key B            |  |  |  |  |                       |  |  |  |  |
| 7<br>0x07                                                                                                                                                                                                                                                                                                                                        | Example of an unallocated block | 0 | Record #1 ID       |   |   |       |   |   |                |    |    |                   | #1 Sector         |    |    |                                                                                                                                                                                                                            |  | Record #2 ID |  |  |  |  | Access bits (selected) |  |  |  |  | Access bits (default) |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Index Key A        |   |   |       |   |   |                |    |    |                   | Index Key A       |    |    |                                                                                                                                                                                                                            |  | Index Key B  |  |  |  |  | Index Key B            |  |  |  |  |                       |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Record #1 ID       |   |   |       |   |   |                |    |    |                   | #1 Sector         |    |    |                                                                                                                                                                                                                            |  | Record #2 ID |  |  |  |  | Access bits (selected) |  |  |  |  | Access bits (default) |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Index Key A        |   |   |       |   |   |                |    |    |                   | Index Key A       |    |    |                                                                                                                                                                                                                            |  | Index Key B  |  |  |  |  | Index Key B            |  |  |  |  |                       |  |  |  |  |
| :                                                                                                                                                                                                                                                                                                                                                |                                 | : | Record #1 ID       |   |   |       |   |   |                |    |    |                   | #1 Sector         |    |    |                                                                                                                                                                                                                            |  | Record #2 ID |  |  |  |  | Access bits (selected) |  |  |  |  | Access bits (default) |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Index Key A        |   |   |       |   |   |                |    |    |                   | Index Key A       |    |    |                                                                                                                                                                                                                            |  | Index Key B  |  |  |  |  | Index Key B            |  |  |  |  |                       |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Record #1 ID       |   |   |       |   |   |                |    |    |                   | #1 Sector         |    |    |                                                                                                                                                                                                                            |  | Record #2 ID |  |  |  |  | Access bits (selected) |  |  |  |  | Access bits (default) |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Index Key A        |   |   |       |   |   |                |    |    |                   | Index Key A       |    |    |                                                                                                                                                                                                                            |  | Index Key B  |  |  |  |  | Index Key B            |  |  |  |  |                       |  |  |  |  |
| n<br>(max. 128)                                                                                                                                                                                                                                                                                                                                  | Example of an unallocated block | 0 | Record #1 ID       |   |   |       |   |   |                |    |    |                   | #1 Sector         |    |    |                                                                                                                                                                                                                            |  | Record #2 ID |  |  |  |  | Access bits (selected) |  |  |  |  | Access bits (default) |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Index Key A        |   |   |       |   |   |                |    |    |                   | Index Key A       |    |    |                                                                                                                                                                                                                            |  | Index Key B  |  |  |  |  | Index Key B            |  |  |  |  |                       |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Record #1 ID       |   |   |       |   |   |                |    |    |                   | #1 Sector         |    |    |                                                                                                                                                                                                                            |  | Record #2 ID |  |  |  |  | Access bits (selected) |  |  |  |  | Access bits (default) |  |  |  |  |
|                                                                                                                                                                                                                                                                                                                                                  |                                 |   | Index Key A        |   |   |       |   |   |                |    |    |                   | Index Key A       |    |    |                                                                                                                                                                                                                            |  | Index Key B  |  |  |  |  | Index Key B            |  |  |  |  |                       |  |  |  |  |
| Data block access: Read, write, increment, transfer, restore with key A. Read with key B.                                                                                                                                                                                                                                                        |                                 |   |                    |   |   |       |   |   |                |    |    |                   |                   |    |    | Sector trailer access: Read on key A, read/write on key B and read/write on access bits plus byte 9 with key A. No access with key B.                                                                                      |  |              |  |  |  |  |                        |  |  |  |  |                       |  |  |  |  |
| Data block access: Read with key A. Read/write with key B.                                                                                                                                                                                                                                                                                       |                                 |   |                    |   |   |       |   |   |                |    |    |                   |                   |    |    | Sector trailer access: Read on access bits plus byte 9 with key A. Write on key A, key B with key B and read/write on access bits plus byte 9 with key B.                                                                  |  |              |  |  |  |  |                        |  |  |  |  |                       |  |  |  |  |
| Contains max. 12 references to records. Each record in the index takes 3 bytes for its ID and 1 byte for the starting sector number. The references must be stored successively from the start of the sector.                                                                                                                                    |                                 |   |                    |   |   |       |   |   |                |    |    |                   |                   |    |    | Records are implemented as a linked list. Byte 9 of every record sector trailer should contain the number of the sector in which the record payload continues. In every last sector of a record, this byte should be 0x00. |  |              |  |  |  |  |                        |  |  |  |  |                       |  |  |  |  |
| The sector allocation block contains information about which sector is in use. Every bit represents a sector: bit 0 (first) maps to sector 0, bit 1 maps to sector 1, bit 127 (max) maps to sector 127. 1 means 'in use', 0 means 'free'. Note that bit 0 and bit 1 should always be 1, as the header and index sector will always be allocated. |                                 |   |                    |   |   |       |   |   |                |    |    |                   |                   |    |    |                                                                                                                                                                                                                            |  |              |  |  |  |  |                        |  |  |  |  |                       |  |  |  |  |

CareRecord Layout

| BYTE WITHIN BLOCK        |                            |                            |                            |                            |                            |                            |                            |                             |                             |                             |                             |                             |                             |                             |                             |
|--------------------------|----------------------------|----------------------------|----------------------------|----------------------------|----------------------------|----------------------------|----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|
| DATA BLOCK WITHIN RECORD |                            |                            |                            |                            |                            |                            |                            |                             |                             |                             |                             |                             |                             |                             |                             |
| 0                        | 1                          | 2                          | 3                          | 4                          | 5                          | 6                          | 7                          | 8                           | 9                           | 10                          | 11                          | 12                          | 13                          | 14                          | 15                          |
| SHA-256 CHECKSUM         |                            |                            |                            |                            |                            |                            |                            |                             |                             |                             |                             |                             |                             |                             |                             |
| ENTRY #1 KEY             | ENTRY #1 LENGTH            | ENTRY #2 VALUE             | ENTRY #3 KEY               | ENTRY #3 LENGTH            | ENTRY #4 VALUE             | ENTRY #4 KEY               | ENTRY #4 LENGTH            | ENTRY #5 KEY                | ENTRY #5 LENGTH             | ENTRY #6 KEY                | ENTRY #6 LENGTH             | ENTRY #7 KEY                | ENTRY #7 LENGTH             | ENTRY #8 KEY                | ENTRY #8 LENGTH             |
| ENTRY #2 LENGTH          | ENTRY #3 VALUE (CONTINUED) | ENTRY #4 VALUE (CONTINUED) | ENTRY #5 VALUE (CONTINUED) | ENTRY #6 VALUE (CONTINUED) | ENTRY #7 VALUE (CONTINUED) | ENTRY #8 VALUE (CONTINUED) | ENTRY #9 VALUE (CONTINUED) | ENTRY #10 VALUE (CONTINUED) | ENTRY #11 VALUE (CONTINUED) | ENTRY #12 VALUE (CONTINUED) | ENTRY #13 VALUE (CONTINUED) | ENTRY #14 VALUE (CONTINUED) | ENTRY #15 VALUE (CONTINUED) | ENTRY #16 VALUE (CONTINUED) | ENTRY #17 VALUE (CONTINUED) |
| :                        | :                          | :                          | :                          | :                          | :                          | :                          | :                          | :                           | :                           | :                           | :                           | :                           | :                           | :                           | :                           |
| ...                      | ...                        | ...                        | ...                        | ...                        | ...                        | ...                        | ...                        | ...                         | ...                         | ...                         | ...                         | ...                         | ...                         | ...                         | ...                         |
| m-3                      | m-2                        | m-1                        | m                          |                            |                            |                            |                            |                             |                             |                             |                             |                             |                             |                             |                             |
| RECORD ID                |                            |                            |                            |                            |                            |                            |                            |                             |                             |                             |                             |                             |                             |                             |                             |
| CHECKSUM                 |                            |                            |                            |                            |                            |                            |                            |                             |                             |                             |                             |                             |                             |                             |                             |
| ENTRIES                  |                            |                            |                            |                            |                            |                            |                            |                             |                             |                             |                             |                             |                             |                             |                             |
| TRAILING ZEROS           |                            |                            |                            |                            |                            |                            |                            |                             |                             |                             |                             |                             |                             |                             |                             |

Figure F.2: The layout of a carerecord.

## F.4 Caregiver application

The class diagram and the database diagram of the *caregiver app* are included in this section.

### F.4.1 Class diagram

See figure F.3 for the class diagram of the caregiver app.

### F.4.2 Database diagram

The SQLite database of the caregiver app consists of only a single table. This table stores the not yet synchronized time registrations, plus the active check-in (if any).

| pending_time_registrations |         |
|----------------------------|---------|
| _ID                        | INTEGER |
| uuid                       | TEXT    |
| type                       | TEXT    |
| method                     | TEXT    |
| timestamp                  | INTEGER |
| timesource                 | TEXT    |
| location_latitude          | REAL    |
| location_longitude         | REAL    |
| location_accuracy          | REAL    |
| device_id                  | TEXT    |
| subscriber_id              | TEXT    |
| sim_serial_number          | TEXT    |
| line_1_number              | TEXT    |
| os_version                 | TEXT    |
| app_version_name           | TEXT    |
| app_version_code           | INTEGER |
| tag_uid                    | BLOB    |
| hash                       | BLOB    |
| client                     | TEXT    |
| remarks                    | TEXT    |
| stored                     | INTEGER |
| active                     | INTEGER |

## F.5 Backoffice module

The class diagram and the database diagram of the *backoffice module* are included in this section.



57

### **F.5.1 Class diagram**

See figure F.4 for the class diagram of the backoffice module.

### **F.5.2 Database diagram**

See figure F.5 for the database diagram of the backoffice module.

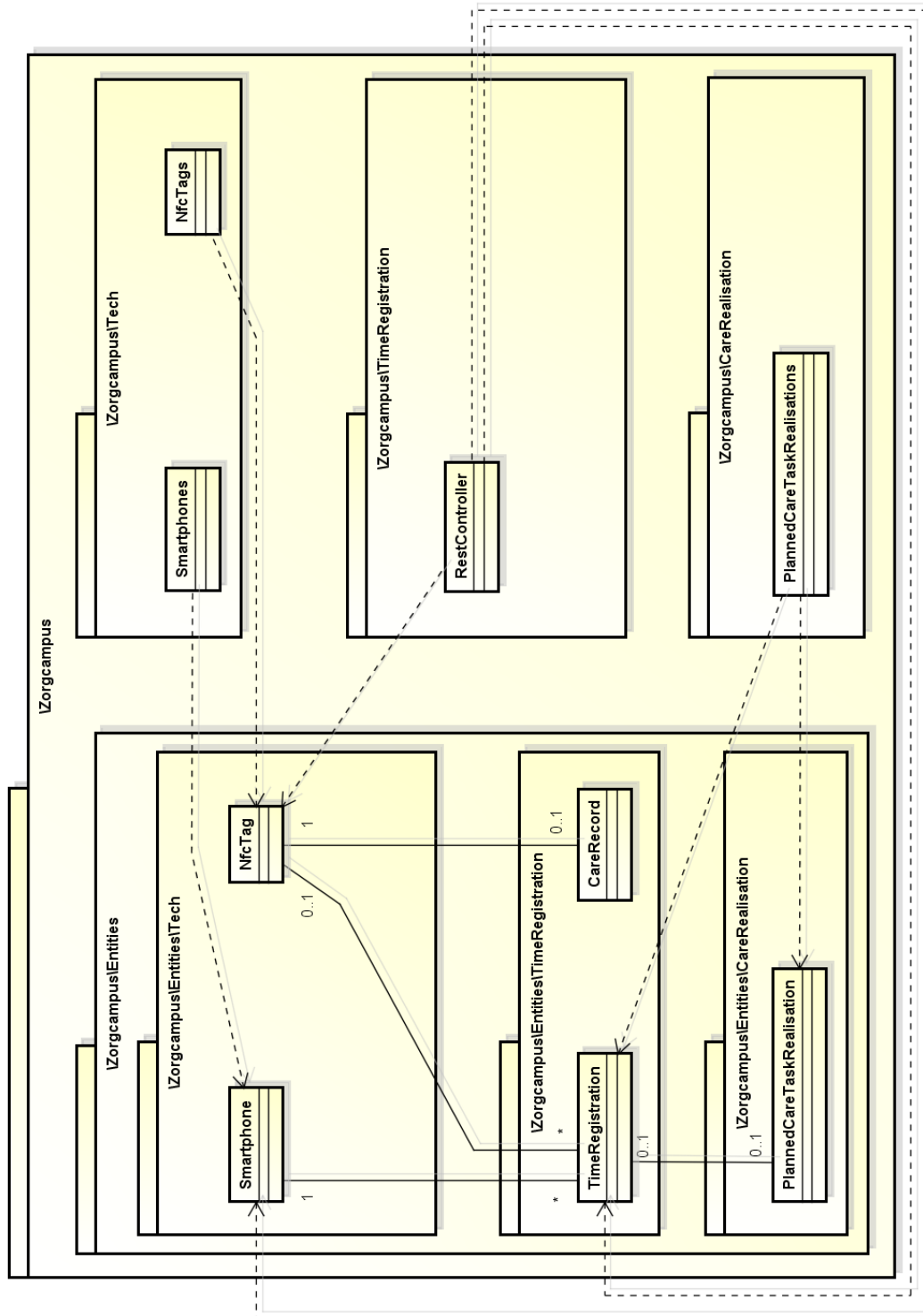


Figure F.4: Class diagram backoffice module.

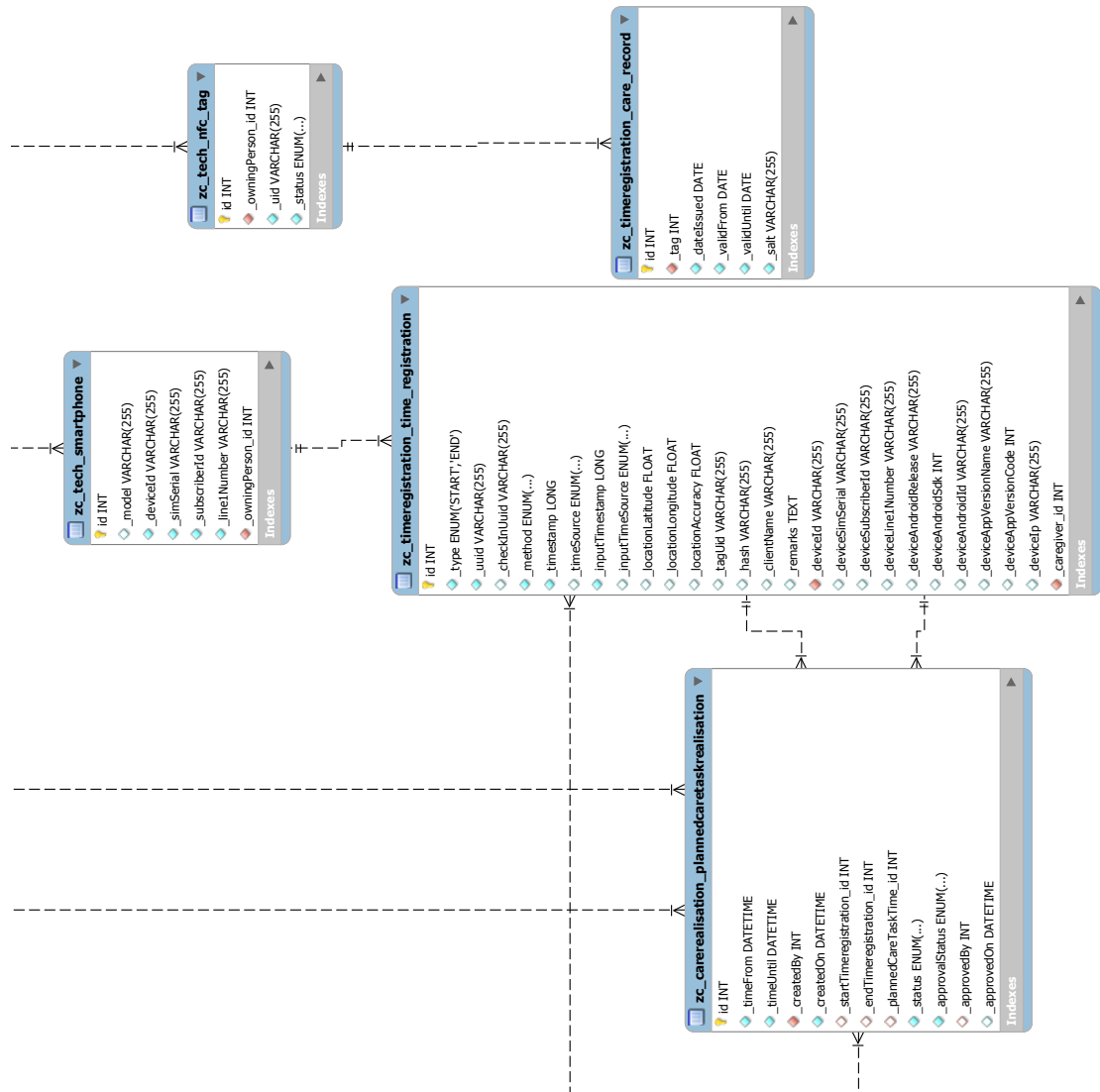


Figure F.5: Database diagram backoffice module.

## Appendix G

# Quality assurance and testing plan

### G.1 Development process

As development process, some parts of SCRUM [27] are used with sprints of 2 weeks. At the end of every sprint, there's a milestone with a working increment of the software and a small demonstration. A meeting with the university mentors will be planned to review the last sprint.

“Scrum is a simple framework for effective team collaboration on complex projects. Scrum provides a small set of rules that create just enough structure for teams to be able to focus their innovation on solving what might otherwise be an insurmountable challenge.”

For issue tracking, Trac is used to store all tasks and bugs. When a new sprint starts, tasks are assigned to this milestone.

### G.2 Continuous automated build and testing system

The continuous automated build and testing system Jenkins [9] will be used to ensure the continuous quality of the code.

The system checks in all code repositories every five minutes. If the code of the NFC library has changed, a new build is started and all test are executed. After a successful build of the library, a new build of the Caregiver app will be triggered too. When there are changes in the Caregiver app, the latest NFC library is used for building. It isn't necessary to rebuild the library if no code has been changed here. After a successful build the Caregiver app is installed on an Android emulator for automated GUI testing. Finally, JavaDoc is generated from source and the application packages are saved as artifact.

When the backoffice module will be developed, Jenkins shall also be used to execute the PHP unit tests.

### G.3 Code quality

The Android code style [4] shall be used for the Android app code. All functions and methods will have JavaDoc documentation and Android Lint will be used to secure the code quality. For the backoffice module, the PEAR coding standards [17] shall be respected, in the case of PHP code. PEAR’s *CodeSniffer* can be used to check the compliance of the application code to these standards. All HTML will be checked with W3C’s validator [24], CSS with W3’s Jigsaw validator [23] and for JavaScript, JSLint [10] is used. All code from person A will be reviewed and tested by person B and visa versa.

### G.4 Unit testing

All model classes should be covered by unit tests as much as possible. For Android Java classes, the JUnit unit testing framework [12] shall be used. For PHP classes, we’ll use the PHP equivalent to JUnit, which is called PHPUnit [18].

The usage of coverage reports should ensure that the model code is properly covered by the unit tests. These reports should be generated automatically by the continuous integration system Jenkins.

### G.5 Regression testing

When a bug is found a new ticket is made in Trac and if possible a JUnit test will be created to test for this bug. This assures that with every new build, all previously fixed bugs are tested again.

### G.6 GUI testing

The Android SDK has a built-in GUI test called *The Monkey* [22]. This is only a minor stability test to check whether the GUI stays responsive upon random input.

“The Monkey is a program that runs on your emulator or device and generates pseudo-random streams of user events such as clicks, touches, or gestures, as well as a number of system-level events. You can use the Monkey to stress-test applications that you are developing, in a random yet repeatable manner.”

All other kinds of GUI testing will be executed by hand, because of the limited amount of time available.

## **G.7 Functional testing**

At the end of the project, functional testing will be performed. With functional testing, the requirements analysis document is checked against the final product, to see whether the requirements defined previously were implemented. This will process will be performed by hand.

## **G.8 Acceptance testing**

The acceptance tests will be the final stage of this project. At this stage, the app and backoffice module will be tested by people who are going to use them with their daily work.

A functional prototype of the app will be tested by a caregiver, which will receive a short explanation about the app. After that, there will be a simulation of a work situation, in which the caregiver will need to check in and out with the smartphone.

The backoffice module will be tested by a care coordinator. A number of simulations will be done in which the coordinator should view and manage transactions, spot irregularities amongst those transactions and manage RFID tags issued to clients.

The various simulations will be observed closely, so behavior of the system that was not expected by the actor is spotted. After the simulations, there will be a short evaluation with the actors. Based on the observations and the evaluations, recommendations for improvement of the system can be made.

# Bibliography

- [1] 4 byte and 7 byte uid offering for mifare<sup>TM</sup> classic, mifare plus<sup>TM</sup>, smartmx<sup>TM</sup> and licensed products – questions and answers. [http://www.mifare.net/files/4713/0936/9004/4-7\\_B\\_ID\\_Questions\\_Answeres\\_V12.pdf](http://www.mifare.net/files/4713/0936/9004/4-7_B_ID_Questions_Answeres_V12.pdf). Retrieved on October 2nd, 2012.
- [2] Acr122u product page. <http://www.acs.com.hk/index.php?pid=product&id=ACR122U>. Retrieved on November 28th, 2012.
- [3] Advanced card systems ltd. website. <http://www.acs.com.hk>. Retrieved on November 28th, 2012.
- [4] Android code style guidelines. <http://source.android.com/source/code-style.html>. Retrieved on October 8th, 2012.
- [5] Apktool at google code. <http://code.google.com/p/android-apktool/>. Retrieved on October 17th, 2012.
- [6] Ciphers, openssl documentation. <http://www.openssl.org/docs/apps/ciphers.html>. Retrieved on November 29th, 2012.
- [7] Dexguard on saikoa website. <http://www.saikoa.com/dexguard>. Retrieved on November 28th, 2012.
- [8] Javascript object notation, php manual. <http://php.net/manual/en/book.json.php>. Retrieved on November 29th, 2012.
- [9] Jenkins, an extendable open source continuous integration server. <http://jenkins-ci.org/>. Retrieved on October 8th, 2012.
- [10] Jslint website. <http://www.jshint.com/>. Retrieved on October 15th, 2012.
- [11] JsonObject, android developers reference. <http://developer.android.com/reference/org/json/JSONObject.html>. Retrieved on November 29th, 2012.
- [12] Junit website. <http://www.junit.org/>. Retrieved on October 15th, 2012.
- [13] libnfc documentation: nfc-mfclassic example. <http://www.libnfc.org/documentation/examples/nfc-mfclassic>. Retrieved on October 17th, 2012.

- [14] Mifare and handling of uids. <http://www.mifare.net/files/6213/2453/8738/AN10927.pdf>. Retrieved on October 2nd, 2012.
- [15] Mifare classic - a pioneer and front runner in contactless smart card ics. [http://www.nxp.com/products/identification\\_and\\_security/smart\\_card\\_ics/mifare\\_smart\\_card\\_ics/mifare\\_classic/](http://www.nxp.com/products/identification_and_security/smart_card_ics/mifare_smart_card_ics/mifare_classic/). Retrieved on October 2nd, 2012.
- [16] nfc-tools at google code. <http://code.google.com/p/nfc-tools/wiki/mfoc>. Retrieved on October 17th, 2012.
- [17] Pear coding standards. <http://pear.php.net/manual/nl/standards.php>. Retrieved on October 8th, 2012.
- [18] Phpunit website. <http://phpunit.de>. Retrieved on October 15th, 2012.
- [19] Representational state transfer, wikipedia. [http://en.wikipedia.org/wiki/Representational\\_state\\_transfer](http://en.wikipedia.org/wiki/Representational_state_transfer). Retrieved on November 29th, 2012.
- [20] Security of mifare<sup>TM</sup> classic. <http://mifare.net/technology/security/mifare-classic>. Retrieved on October 3rd, 2012.
- [21] Sqlcipher for android. <http://sqlcipher.net/sqlcipher-for-android/>. Retrieved on November 28th, 2012.
- [22] Ui/application exerciser monkey on android developer website. <http://developer.android.com/tools/help/monkey.html>. Retrieved on October 5th, 2012.
- [23] W3c's css validator. <http://jigsaw.w3.org/css-validator/>. Retrieved on October 15th, 2012.
- [24] W3c's html validator. <http://validator.w3.org/>. Retrieved on October 15th, 2012.
- [25] Website doctrine project. <http://www.doctrine-project.org/>. Retrieved on October 3rd, 2012.
- [26] Website symfony framework. <http://www.symfony-project.org/>. Retrieved on October 3rd, 2012.
- [27] What is scrum? <http://www.scrum.org/Resources/What-is-Scrum>. Retrieved on October 8th, 2012.
- [28] The tls protocol version 1.2. <http://tools.ietf.org/rfc/rfc5246.txt>, 2008. Retrieved on October 2nd, 2012.
- [29] Important information regarding 4 byte & 7 byte uids of mifare<sup>TM</sup> products. <http://www.mifare.net/technology/4-7byte-uid/>, 2011. Retrieved on October 2nd, 2012.

- [30] Ronny Wichers Schreur, Peter van Rossum, Flavio Garcia, Wouter Teepe, Jaap-Henk Hoepman, Bart Jacobs, Gerhard de Koning Gans, Roel Verdult, Ruben Muijers, Ravindra Kali, and Vinesh Kali. Security flaw in mi-fare classic. [http://www.cs.ru.nl/~flaviog/publications/Security\\_Flaw\\_in\\_MIFARE\\_Classic.pdf](http://www.cs.ru.nl/~flaviog/publications/Security_Flaw_in_MIFARE_Classic.pdf), 2008. Retrieved on October 3rd, 2012.