AALBORG UNIVERSITY

STUDENT REPORT

SW6 Multi Project

Developing Complex Software Systems:

GIRAF Web Admin and GIRAF Timer

P6 Project by sw606f14

3-2-2014 to 28-5-2014

Participants:
Dennis Jakobsen
Erik Sidelmann Jensen
Lasse Vang Gravesen
Jens Møller Kursch

AALBORG UNIVERSITY

STUDENT REPORT

Third study year
Software
Selma Lagerlöfsvej 300

Title: GIRAF Web Admin and GIRAF Timer

Theme: Developing Complex Software Systems

Project period: P6, spring semester 2014

Project group: sw606f14

Participants:

Dennis Jakobsen
Erik Sidelmann Jensen
Lasse Vang Gravesen
Jens Møller Kursch

Supervisor:

Hua Lu

Copies: 8

Content Pages: 46

Appendix: 47

Total Pages: 102

Completed: 28-5-2014

Abstract:

To improve the quality-of-life of people on the autism spectrum there is a very big need for special activities along with rigidly structured days. This is primarily not done using computers or other digital devices, instead it is done physically with for example folders of pictograms to help facilitate communication.

In collaboration with a larger group of people we wanted to continue work on an existing system that moves a lot of these activities, structure and administrative tasks to digital platforms, on Android, iOS and on webpages.

Our contribution to the larger project was at first to work on an existing Timer application that allows the 'Guardians' of autistic people to set up some time for an activity such that the person knows when to stop. We then moved to an administrative tool, called GIRAF Web Admin, that allowed administrating people, their relationships, pictograms and who they are assigned to, along with an overview of general information about people, their department and so on.

A fairly complete administrative tool was the result of the GIRAF Web Admin, though with a lot of room for improvement.

Resume

In this project we, sw606f14, worked on a multi project called GIRAF (**G**raphical **I**nterface **R**esources for **A**utistic **F**olk). GIRAF is a project for 6th semester bachelor students of Software Engineering at Aalborg University. The idea of the project is to create an autism friendly software package for communication and everyday activities to be used in departments for citizens with autism, though some of the package is not specifically for autistic people but rather for administrative purposes as well.

The challenge of this semester is to continue the development of a system, containing many different applications with the shared purpose of helping citizens with autism. The multi project is constructed of 16 groups of 2-4 members per group, they created applications for Android, iOS and browsers. We worked in a Scrum of Scrums-like environment, with weekly meetings between the groups. The project was divided into four sprints, with each sprint ending with a presentation to the rest of the groups and the costumers to show what had been done during the sprint.

In Sprint #1, we continued the work on a project called Timer which was started in the previous semester. The purpose of the Timer application was visualization of time, with various ways of showing the time like an hourglass, a clock, a bar and a digital watch. We used most of the sprint refactoring, correcting bugs, learning the structure of code, learning Android development and getting an overview of the overall system.

From Sprint #2 till Sprint #4, we started working on a project called GIRAF Web Admin. GIRAF Web Admin should be a website that guardians can use to administer the use of the GIRAF system for citizens in their department. This project was started in previous semesters, and each time the website have been remade from scratch. After getting an overview of the code we decided to start GIRAF Web Admin from scratch, because we thought it would take too much time to rearrange to code for what the overall GIRAF system would end up being. We decided to implement a Model-View-Controller architecture and use the visual design of the previous project for our reimplemention of GIRAF Web Admin. Throughout the sprints we developed functionality for the project which should be able to administer users, profiles, departments, pictograms, and applications, and handle relations between these entities.

We had two larger collaborations with other groups in the multi project which we documented. The first collaboration was with the group that worked with the application Oasis. They were creating an administration tool similar to GIRAF Web Admin but to use on the tablet. We had conducted an interview with a couple of costumers to find requirements for the administration tools of the GIRAF multi project. Together with Oasis we analysed the interview and established requirements that were relevant for both projects and for each individual project.

The second collaboration was together with the group that worked with the application called Piktotegner. We conducted a collaborative code review together in which we covered around 2500 lines of code and comments for each project. This was done towards the end of the project to further improve the general quality of the code for both applications. A few days was used to correct the errors and problems that the other group found.

We ended up with a website that could serve the basic needs of administrating users, profiles, departments, pictograms and applications within the GIRAF system. The website at its state in the end of the semester should be able to fulfill its role in the GIRAF system, though further development could enhance some of the functions and usability of GIRAF Web Admin.

Foreword

This report was made at Aalborg University in the sixth semester of the Software study by the group sw606f14. The report was made as a part of the P6 project in the period 3-2-2014 to 28-5-2014. The project was supervised by Hua Lu, whose supervision was much appreciated.

Dennis Jakobsen _____

Erik Sidelmann Jensen _____

Lasse Vang Gravesen _____

Jens Møller Kursch _____

Contents

1 Introduction **13**

2 Role in Multi Project **15**
2.1 Analysis of Organizational Context 15
2.2 Role of the Timer 17
2.3 Role of GIRAF Web Admin 17

3 Sprint #1 **19**
3.1 Analysis 19
3.2 Refactoring 22
3.3 Implementation 22
3.4 Test 25
3.5 Summary of Sprint #1 26

4 Sprint #2 **27**
4.1 Analysis 27
4.2 System Definition 27
4.3 Architecture 29
4.4 Implementation 32
4.5 Summary of Sprint #2 34

5 Sprint #3 **35**
5.1 Requirements 35
5.2 Implementation 37
5.3 Test 39
5.4 Summary of Sprint #3 40

6 Sprint #4 **43**
6.1 Refactoring 43
6.2 Implementation 43
6.3 Test 45
6.4 Summary of Sprint #4 45

7 Common Activity Between Oasis Application and Online Administration Tool **47**
7.1 Motivation 47
7.2 Results 47

8 Common Activity Between GIRAF Web Admin and Piktotegner **51**
8.1 Approach 51

9 Conclusion & Reflection **53**
9.1 Multi Group Cooperation . . . 53
9.2 Experience And Recommendations . . . 53

Bibliography **55**

A Final UI **57**

B Test **59**
B.1 Sprint #1 . . . 59
B.2 Sprint #2 . . . 64
B.3 Sprint #3 . . . 74
B.4 Sprint #4 . . . 82

C Changelog **87**
C.1 Sprint #1 . . . 87
C.2 Sprint #2 . . . 88
C.3 Sprint #3 . . . 88
C.4 Sprint #4 . . . 89

D Implementation **91**
D.1 Sprint #2 . . . 91
D.2 Sprint #3 . . . 93

E Customer Contact Questions **97**
E.1 Preliminary . . . 97
E.2 Pictograms . . . 97
E.3 Profiles, Users and Relations . . . 98
E.4 Applications . . . 98
E.5 Parent Access . . . 98
E.6 Show The Site to The Customer and Talk Loosely About It . . . 98

F To the Next Group **99**
F.1 Structure . . . 99
F.2 Current Problems . . . 100

1 Introduction

This chapter introduces the GIRAF multi project. It also introduces what it is about and what the focus is for this semester.

The GIRAF multi project is a large multi group project, where the entirety of SW6 works on different parts of an overall system with a lot of the importance being laid on cooperation between individual groups. The GIRAF multi project is primarily an Android project, with a few other platforms such as iOS and Websites. The Android and website part of the GIRAF multi project was already worked on in previous semesters, therefore most of the new work will be working on-top of an existing, albeit non-functional system, though it is to be expected that some parts of the project will have to be reimplemented because of a variety of reasons. The focus for this semester, rather than analysis and design, was put on implementing as much as possible and getting the overall project in a functional state.

Since the focus was to implement a working system, and because of the nature of the project, which is to cooperate with other groups on a system that have already been designed, there will not be any very technical theory in the report. The primary focus will be on the implementation and test of what was implemented.

To reiterate this means that this semester should result in a working overall system and that emphasis is on collaboration between groups, along with figuring out a way to construct a working organization between the groups that work on this multi project.

We use some specific terms in this report that have a specific meaning, these are as follows:

- **We** : This group, sw606f14.
- **GIRAF** : **G**raphical **I**nterface **R**esources for **A**utistic **F**olk is the acronym for the system.
- **GIRAF multi project** : Used to refer to the overall GIRAF system, as it is being developed by the SW6 semester.
- **GIRAF Web Admin** : Our product, the web administration tool for GIRAF.
- **Previous Group** : When we use this, we usually refer to the work the previous group did implementing the previous GIRAF Web Admin.
- **Previous Report** : When we use this, we refer to the previous report by the previous group.
- **Child(ren)** : A deprecated term for people with autism in the context of the multi project.
- **Citizen(s)** : The new term for people with autism in the context of the multi project.
- **Page** : A term used to describe a whole page on the website.
- **View** : A term used for the HTML/PHP in the view files, that generates the content of what the browser shows to the user.
- **Model** : Models represent some 'object' and has methods for communicating with the database.

- **Controller** : Controllers are groups of related actions, such as actions for a Profile.
- **Action** : Actions are functionality within a controller that usually represents some page within GIRAF Web Admin, such as ViewProfile.

Before we can start describing various development activities, we will document the overall organisational structure and the role of our group in the overall multi project.

2 Role in Multi Project

This chapter presents an analysis of the organizational context in which the GIRAF multi project is being developed, covering structure, resources and tools. Afterwards the roles of the two projects we worked on is described.

2.1 Analysis of Organizational Context

This section will contain an analysis of the organizational context of the GIRAF multi project. The organization consists of 16 groups of 2-4 students on their bachelor semester of Software Engineering at Aalborg University. The organization is ultimately under Aalborg University and the semester coordinator. Therefore every decision made by the students can be overruled by the university, this will however be ignored in the rest of the analysis as it is not something that is likely to happen.

2.1.1 The Organizations Purpose

The purpose of the organization is to create an autism friendly software package for Android tablets and websites along with starting up an iOS branch to help with communication, planning and learning tools. The software should be used in departments, schools and homes that helps people with autism. The goal is that the code base will eventually be released as open source, so that it is freely available and everyone interested can create additional features and functionality.

2.1.2 The Organizations Structure

The structure of the organization is likely to change next year when new students will take over the organization. A simplified current structure for the spring semester of 2014, can be seen in Figure 2.1. The structure was decided upon within the first week of the semester start at a meeting where every student at the semester was invited and decisions was made by voting, majority decides.

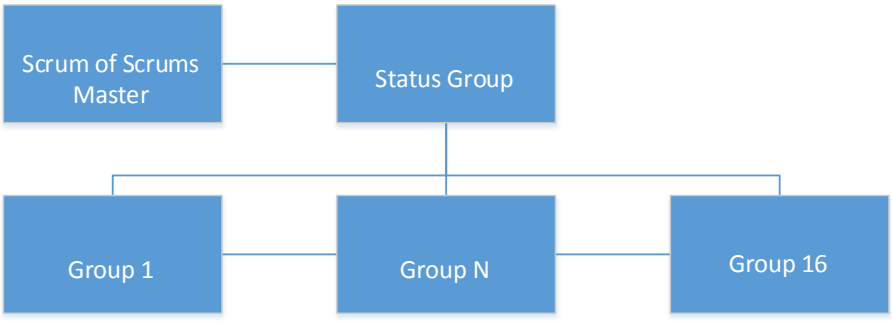

Figure 2.1: An organizational chart of the GIRAF multi project organization.

The scrum of scrum master is a student who plans and conducts meetings for the status group, the role itself has no additional power besides being part of the status group. The status group has the power to make decisions for the organization by voting and this voting is done by representatives of every group.

2.1.3 Leadership

The leadership of the organization is in the hands of the status group. The status group makes decisions by majority voting, in that the thing that the majority of the group votes for wins. The status group has several responsibilites:

- Leading entity
- Information flow
- Keeping the project on track

As the leading entity the status group have power to make general decisions for the groups that are working on the multi project. A status meeting is conducted every week, the meeting follow Scrum meeting protocols such that every group reports what they have done and what they are planning to do and if they have any issues. At the end of the meeting different topics are discussed to ensure that the projects is not derailed.

2.1.4 Tools

The software package is being developed for Android and iOS tablets, and web. The organization is using tools to ease parts of the development, these tools are as follows:

- Android Studio
- Redmine
- Git
- Jenkins

Different tools are of course used when developing for webpages. The tools used to develop for web are as follows:

- PHP
- PHPStorm
- XAMPP

Android studio is used as the IDE in which most of the application development is done. Redmine is a project management software with functionality such as burn down charts, wiki's, issue trackers, forums, calenders and much more. It is expected that every group uses Redmine as thoroughly as possible to track the entire GIRAF multi project as single entity. Git is a tool for revision control and source code management. The groups use git to commit changes and

push to a shared repository, from here every other group can download the newest version of the entire project. Jenkins is an automated build software, it finds the files in the git repository and attempts to build them. If the project does not compile the groups with the faulty code will get informed, so they can correct the error. PHPStorm is an IDE in which PHP development can be done, both PHPStorm and Android Studio are built on top of IntelliJ IDEA. XAMPP is a webserver solution package that allows easily developing locally.

2.1.5 Resources

Every group consists of 2-4 students, two groups share a single group room at the university in which they have access to a blackboard. The university also makes Android tablets available for each group room to make testing easier and faster. The project has one informant and six customers. The informant works for the software company Mymo, whom creates learning games, they can be consulted for knowledge about learning games. The customers work for departments which takes care of citizens and younger people with autism. These costumers can be contacted for additional requirements and ideas. They are also our experts on autism, and when further knowledge about autism needs to be obtained to develop a certain functionality, these costumers can be contacted. The resources coming from within the organization is knowledge of software development and the execution of this.

2.2 Role of the Timer

This section will explain the role of the timer application in the overall GIRAF multi project. We took part in the development of the timer in Sprint #1.

The Timer application is an application that the guardians use to help the citizens visualize the amount of time they have left on some activity, since without some sort of visualisation it can be difficult for some autistic people.

The reason we wanted to work with the Timer application, was to acquire an understanding of android development and to figure out how the multi project worked together. It was also communicated that it was a fairly complete application, that worked sufficiently well but had some problems that needed to be figured out, especially in the context of the larger system since it was working rather independently from them at the start of Sprint #1. It also allows guardians to time activities for citizens.

2.3 Role of GIRAF Web Admin

This section will explain the role of the GIRAF Web Admin in the overall GIRAF multi project. We have developed the GIRAF Web Admin from scratch from Sprint #2 till the end of Sprint #4, the last sprint.

The GIRAF Web Admin is a tool for the guardians who use the GIRAF multi project. It works as a user friendly interface to the database. Guardians can create and delete profiles, users, and pictograms. They can also create relations between profiles, pictograms, and applications.

Now that the functionality have been shortly covered we can delve into the actual role of the GIRAF Web Admin in the overall GIRAF multi project. Most application in the GIRAF multi project is developed to be installed on Android tablets. These applications are developed for communication and learning for autistic people under the supervision of guardians and parents. The system is being developed to be used in kindergarten, schools and group homes for

citizens, teenagers and young adults. Many of the citizens that would use this Android system are so impaired by their autism that they need supervision from a guardian to use the applications and the guardian will need a mode to control which functions the citizens have access to. The guardian will need to add pictograms to the system, new users, new profiles and much more. There is an application under development for this purpose, but much of this is easier to do from a computer, as there are existing tools to make pictograms for the platform, furthermore it is easy to download other pictograms from the internet. Therefore it makes sense to place the tool for uploading pictograms and administration of the system at the medium where the guardians normally do this, at their computer. The system is also intended to be released for private use and again the parents of the citizens to whom use the tablet should be able to administer their access in the system, this can be done without taking the tablet away from the citizen, via the GIRAF Web Admin.

3 Sprint #1

This chapter covers Sprint #1, primarily describing the analysis of the current Timer architecture to determine exactly what the previous group did and gathering it into a single design document that can be referred to in the future. The purpose of Sprint #1 was set down by the status group: It should be for fixing bugs, fixing functionality, and generally learning how to add features to the codebase along with learning how to work with Android.

3.1 Analysis

To continue the development on the Timer project of the GIRAF multi project, an analysis of the current work has to be done. An important part of understanding the current code base, is to understand the design of the application. Parts of the information comes from the report of the previous group who developed the Timer application [1].

3.1.1 Design Document From Previous Report

A design document was not presented in the previous report, but rather spread out through the report, from which the following information was gathered:

- Architectural Design
- Class Diagram
- Functionality

Architectural Design

The architecture consists of five layers each of which has a separate responsibility of the application, see Figure 3.1.

The Main layer handles the interactive display towards the user and will initialise the application with the appropriate method calls to the Tools layer. The Tools layer contains the modelling of the problem domain, such as a guardian and a citizen. Furthermore it interfaces with the local database through the CRUD layer which returns objects of the classes defined in the Tools layer. The Layout layer is responsible for updating the fragments in the main activity, these fragments can be seen in Figure 3.2. The layout layer depends on the Draw layer which generates the views for timers and pictograms that are displayed in the activities or fragments.

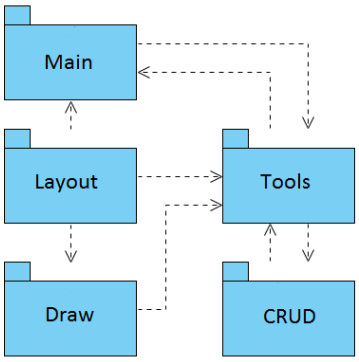

Figure 3.1: Five Layered Architecture [1].

The previous report stated, that the five-layered architecture makes further development possible because each layer can be replaced by new ones. Furthermore

they argue that this architecture allows for other GIRAF projects to continuously add new features which Timer is compatible with [1, p. 31].

We think that the chosen layered architecture resembles the Model-View-Controller(MVC) design pattern. Where the current Tools layer would be placed on the Model layer. The Main layer and partly the Layout layer would form the View layer in that the Layout layer also contains logics about the information to show, which is why it also belongs to the controller layer. The CRUD layer can be argued whether it should be contained in the controller layer or in a separate Database layer, although this layer should not be necessary since a common API should be available across all projects in the GIRAF multi project. The Draw layer would probably be placed in the controller layer since it contains methods to generate the timers and pictograms to be displayed in the View layer.

There exists no reasoning in the previous report on why the current layered architecture was chosen. Perhaps no other option was considered.

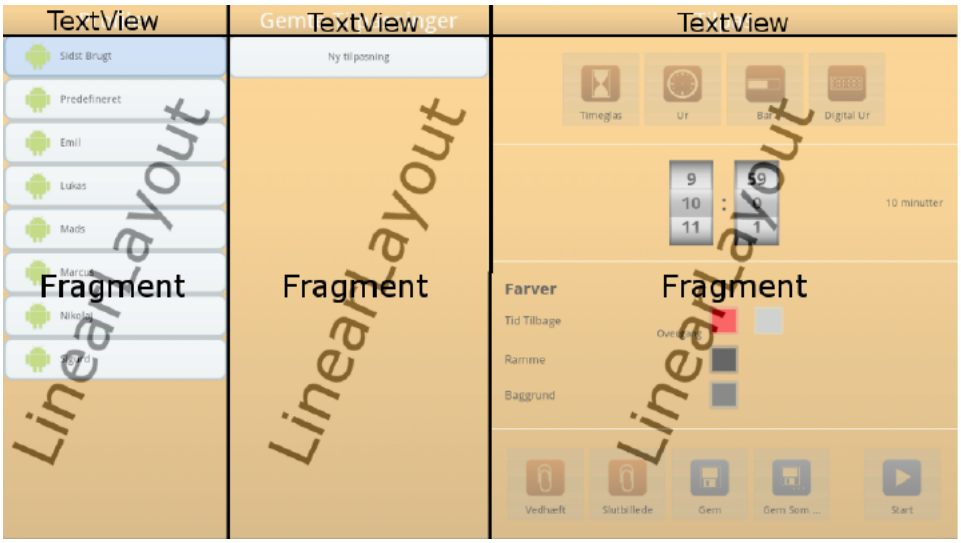

Figure 3.2: Overview of Fragments in Main Activity [1].

Class Diagram

The class digram from the previous report, seen in Figure 3.3, show that one `Guardian` has one or more `Child` objects, each of which can have zero or more `SubProfile` objects. A `SubProfile` consist of an `Attachment`, this being either a single image, split images, or a timer. Furthermore we see that each type of timer, `DigitalClock`, `ProgressBar`, `Hourglass`, `TimeTimer`, inherits from `SubProfile`. The class diagram covers the architectural layer, Tools.

Functionality

The functionality of the system is not described in the design document, but the way each feature is implemented is described in the implementation chapter. Therefore the functionality

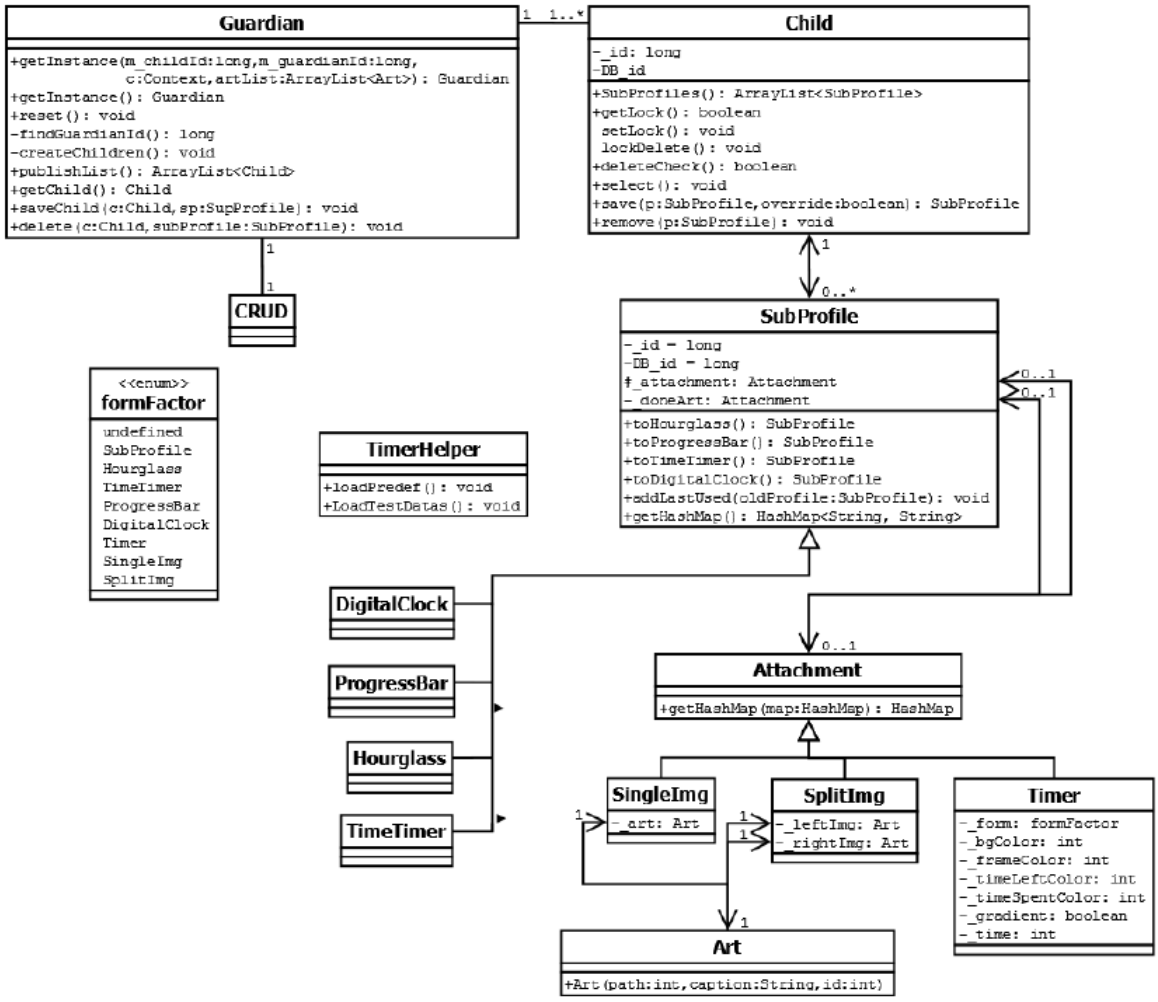

Figure 3.3: Class Diagram of Tools Layer [1].

of the system is derived in a high-level description from what was implemented by the previous group [1, p. 39-40].

"Change style of the timer"
Allow the user to select which type of timer to use, this being either an hourglass, progressbar, analog watch, or digital watch.

"Change the timespan of the timer"
The user should be able to set the time of the timer, to anything from 0 minutes and 0 seconds to 60 minutes and 0 seconds.

"Change the color of the timer and background"
Once a timer is selected the user should be able to set the colors of the timer and background. For example, when an hourglass is picked, the color of the sand and the background should be allowed to be changed.

"Change the color of the timer to be changing gradiently"
This feature allows the user to specify whether the color of time, e.g. the sand in an hourglass, changes gradiently between two specified colors.

"Attach one or two pictograms, or a timer"
There should be a feature that allows the user to attach pictograms or another timer to the timer configuration. This means that the user can split the screen in two, half of the screen containing the original timer, and the second half either contains one or two pictograms or another timer not necessarily of the same style.

"Change the default done pictogram to one or two pictograms"
A default pictogram is displayed when the timer has expired. The user should be able to customise the configuration to show one or two relevant pictograms.

"Save the timer"
The user should be able to save the timer configuration under each citizen.

"Start the timer"
Once a timer configuration is set up with a timer style, duration, and or different attachments, the user should be able to start the timer.

3.1.2 Auto generated Class Diagram

Since there was no class diagram for Timer, the class diagram was auto generated from the source code. This was done with the enterprise edition of Visual Paradigm for UML 11.0 [2]. The project was split up into three subprojects: Draw-lib, Timer-lib, Wombat(Timer). The layers of the architecture was split up between the subprojects, where the Draw-lib covered the Draw layer, Timer-lib covered the Tools layer and CRUD layer, and Wombat(Timer) covered the Main layer and Layout layer. The diagrams for each of the subprojects can be found on the attached medium (CD).

3.2 Refactoring

Refactoring is the process in which existing code is improved in some way without changing the behaviour. Due to the purpose for Sprint #1 refactoring was done to some extent. The changes includes:

- Removal of unused drawable and layout files.
- Removal of constants and replacing them with resource values.
- Cleaning up sources of Null Pointer Exceptions, and various other exceptions.

3.3 Implementation

This section will document usability problems identified by the previous group that were not resolved by them. Furthermore a description of how we resolved them during this sprint and a description of different development done in Sprint #1 is presented in this section.

3.3.1 Usability Problems

The previous group identified a few usability problems in the Timer application that they intended for the next development group to resolve, see [1, p. 12] and [1, p. 79]. The following documents what the usability problems were, and how we resolved them during this sprint.

There were two cosmetic issues identified in the previous report. The first issue was the fact that users were not sure when attachments had been attached. This was resolved by adding a toast, a type of temporary pop-up notification when the user adds attachments. The second issue was the fact that the users had difficulties identifying the button to start the selected timer. This was not fixed as it was deemed as an issue that would resolve itself over a couple of sessions of using the application.

There were three serious issues identified in the previous report. The first issue was the fact that the user had difficulties using the Android color palette. This issue was not resolved. The second issue was that long click was unintuitive. This along with the fact that not having long click interactions is now a general requirement meant that all long click interactions were removed from the applications and were replaced with ordinary buttons. The third issue was that the "Last Used" list was difficult to find. This was resolved by the removal of the profiles list entirely, as the current guardian and citizen are selected when the application is opened through the Launcher application.

There was one critical issue identified in the previous report. This issue was the fact that the check box for the gradient option could be invisible on some background colors. This was resolved by making the color of the check box complementary to the color of the background.

3.3.2 Launcher—Timer Interaction

A large part of the development focus was also put on ensuring that the interaction with the Launcher application worked properly. One aspect of doing this was ensuring the information, the extras, sent by the Launcher application was used properly. The most important part of this information is the id of the currently selected guardian and citizen. What this means is that the Profile fragment in the original layout was removed entirely, see leftmost fragment in Figure 3.2, as this overrode the information sent by the Launcher, which is not the desired behavior. It is not the desired behavior because we want the Launcher application to work as a hub that sends all necessary information regarding the current guardian and current citizen. Only using the extra information from the Launcher then means that opening the application outside of the Launcher should be practically impossible because no information is made available regarding which guardian is currently selected. How this extra information is used in the `MainActivity` class can be seen in List 3.1. The application can determine if it was opened outside the Launcher by checking whether or not the extras are sent. If the extras are not null it will set the `guardianId`, `childId` and `color` and then use this information to create the context around which timers are saved by creating a `Guardian` object using the extras. If however the extras are null, it will create a dialog box giving only the option to quit the program and use a different layout for the application.

```
1 Bundle extras = getIntent().getExtras();
2 if (extras != null) {
3         guardianId = extras.getLong("currentGuardianID");
4         childId = extras.getLong("currentChildID");
5         color = extras.getInt("appBackgroundColor");
6 } else {
7     // Alert Dialog where only option is to quit if extras are null.
8 }
9 [...]
10 /* Initialize the guardian object */
11 guard = Guardian.getInstance(childId, guardianId, getApplicationContext(),
      artList);
12 guard.backgroundColor = color;
13
14 // Set content view according to main, which implements two fragments
15 if(extras != null) {
16     setContentView(R.layout.main);
17     Drawable d = getResources().getDrawable(R.drawable.background);
18     d.setColorFilter(color, PorterDuff.Mode.OVERLAY);
19     findViewById(R.id.mainLayout).setBackgroundDrawable(d);
20 }
21 else {
22     setContentView(R.layout.blank);
23 }
```

List 3.1: Usage of the extras sent by the Launcher.

There are of course other changes made with regard to getting the Launcher and the application to work together correctly. As hinted above, a result of ensuring that the Launcher is the hub that determines which guardian and which citizen are loaded is that there should be no way to change the currently selected citizen or Guardian from the application itself. Making it work meant multiple aspects of the application had to be modified. Because it meant that some behavior that depended on changing or getting information about the currently selected citizen was no longer available because the ListView to select the citizen had been removed entirely. One example is that saving timers used the currently selected citizen and this had to be changed to use the `childId` sent over by the Launcher instead.

3.3.3 New Requirements and Other Changes

New requirements were set out by the customer throughout the sprint, and the ones received fairly early in the sprint are documented here. The following are the new requirements that were worked on:

- Change the layout of the buttons from horizontal to vertical.
- Removal of the color red.

The requirement of changing the layout was resolved using a new layout for the customise fragment and a button that allows the user to switch between the two layouts. The requirement to remove the color red from the default color scheme, however, the color is still available to be chosen, and the timers was fixed fairly easily, as the only thing that required changing was the fact that the default color of the timers was a bright red. Instead this was changed to grey and white. Other requirements came later in the sprint and were therefore not documented. A different change that was added because it felt necessary was the addition of a new type of

timer. This timer is very similar to the original TimeTimer but instead of slowly decreasing the size of the time left of a 60 minute analog clock, the whole clock face would be used regarding the duration of the timer.

3.4 Test

The report written in 2012 contained eight test cases testing for functionality to:

#1 Save customised timers in specific lists.

#2 Save customised or predefined timers in the highlighted citizen, without having a to choose name and location.

#3 Save timers into the "Last Used" list every time any timer have been run.

#4 Highlight list items when they are clicked.

#5 Highlight list items after a save procedure.

#6 Highlight list items according to the chosen citizen when the application is launched through GIRAF launcher.

#7 Draw and update the timer according to the time left and ensures the timer ends when the timer is up.

#8 Displaying the "Done" screen when the time has run out.

It made sense to reuse these tests to see if it, with our modifications, still performs as intended. A few of the test, specifically the first and the third test, could not be redone in this sprint as these two functionalities had been removed entirely. The second test also had to be modified since the predefined timers were removed. The fourth test was changed because the selection of a citizen was done before the timer application was started. Because of this the fifth test no longer makes much sense, since only the configurations for the selected citizen was shown.

All the tests from the previous report can be found in Appendix B.1.1. The list of test cases was also extended to address the new features added to the application, these include functionality to:

#9 Change the layout of buttons.

#10 Delete timer configurations from a citizen.

#11 Attach another timer and pictograms, and to change the end pictogram.

#12 Change the colour of the timer.

From Table 3.1 it can be seen that all except two tests passed. The first failed test, #4, failed because of the "Skift Layout" button. The button works by reloading the activity with another layout. A side effect from the reloading is that the currently selected timer, the time, and colours are deselected and reset. This could potentially be fixed by saving the current selections and colours and then restoring them again after the reload. This was not fixed, though, since it seems reasonable that the user will press the "Skift Layout" button as one of the first actions, since it

Test #	Test Result (pass/fail)
2	Pass
4	Fail
6	Pass
7	Pass
8	Pass
9	Pass
10	Pass
11	Fail
12	Pass

Table 3.1: Test results.

does not make much sense to change the layout just before they start the timer. Furthermore the selected layout is remembered between launching the application, so if it is primary the same citizen who is using the tablet they will properly use the same layout each time.

#11 failed when attaching a certain timer to another timer. Specifically when attaching the "Ur(Time Skive)" timer to one of the other timers. This was because of an error with the implementation of the timer. The error was corrected so that the tests passed.

3.5 Summary of Sprint #1

At the end of the sprint, we decided to switch to a different project in the overall system. For Sprint #2 the project chosen was a webpage for administrating the contents of the remote database like adding or removing guardians, citizens, or parents and administrating their connections. The reason for the switch to a new project, is that we felt that it was more important to have some means of administrating the contents of the remote database at least in comparison to working on an application where most of the work would be polishing and getting to work in the overall system.

4 Sprint #2

This chapter covers Sprint #2. During this sprint we left the Timer application in favour of a webpage for administrating different aspects of the overall project, GIRAF Web Admin. The chapter will cover a system definition, analyse the state of the existing GIRAF Web Admin project as it was left by the previous group that worked on it. Then it will document the important development steps in the sprint, such as a basic analysis of requirements and setting up a backlog, architecture, implementation and test. This is also the chapter where the architecture of GIRAF Web Admin is described along with a general implementation strategy.

4.1 Analysis

As mentioned in Section 2.3 the role of the GIRAF Web Admin is an interface for guardians, and possibly parents and administrators to the database. GIRAF Web Admin provides them with predefined functionality that lets them change the content of the database, without needing to interact directly with the database. Such as adding new profiles, modifying information or adding and deleting relations between themselves and the citizens in their care.

There have been several attempts of implementing an administration tool in the past, both as an Android application and as a webpage. We will draw inspiration from the visual design, the user stories and features from the previous GIRAF Web Admin group [3]. We will reimplement much of it using the Model-View-Controller architecture. We do this because the current site contains no clear architecture. Our time will be better spent reimplementing the functionality in a clear architecture, compared to spending the sprint understanding their implementation and then add code to a loose architecture.

We will proceed with something very similar to the existing visual design, since the previous GIRAF Web Admin group had been in contact with the customers and they had accepted it.

We started by creating a proper system definition to understand what the exact functionality should be. Even though we are using an Agile development method, the tools learned in Object Oriented Analysis and Design can still be useful for determining the scope of a project.

4.2 System Definition

The system definition of the GIRAF Web Admin is created by first considering the FACTOR criterion and then writing it into a short and precise text defining the system.

Below is the criterion for the tool considered.

Functionality:
Support for administrating access of pictograms and applications to a certain citizen. Creation, deletion and administration of citizens, guardians, parents and pictograms along with the associated relations.

Application domain:
Guardians, Departments, Home (possibly administrator and parents).

Conditions:
The system will be used through a website hosted at a server accessible for the application domain.

Technology:
An web server capable of serving PHP to the customer, along with use of JavaScript.

Objects:
Citizen, Guardian, Parent, Pictogram, Application, Department

Resposibility:
The responsibility of the system is to administrate the individual setting and information for each citizen regarding their profile, pictograms, and applications. To organize and administrate the organizational context regarding guardians and their associated citizen and their associated parents.

From that we can create a system definition:

A website system used to administrate the accessibility of media on each citizens tablet device, with focus on restricting access to and creating settings for applications and administrating access to pictograms. The system should secondarily serve as an administrative and overview tool for organizational purposes, keeping track of the users in the system and their relations. The system should run on an Apache server with PHP support, accessible from any device with a modern browser including the home and department of the citizen.

From this we get an understanding of what the system should be capable of doing. We can visualize this understanding using rich pictures, as can be seen in Figure 4.1, where the perception at this point is that the parents should be able to view basic information about their own citizen through the GIRAF Web Admin, while guardians should be able to modify and add new information and some administrator that can add, view and modify more general information such as departments.

Using that and the existing site and the previous report, we can create a set of user stories that we can add to our backlog and once the architecture has been set this can be used to start development.

The following requirements were identified and added to the sprint backlog:

- Log In
- Log Out
- Edit Profile
- Create Profile
- Remove Profile
- Create User
- Add relationship between profiles
- Remove relationship between profiles
- Create pictogram
- Assign pictogram to categories
- Remove pictogram from categories

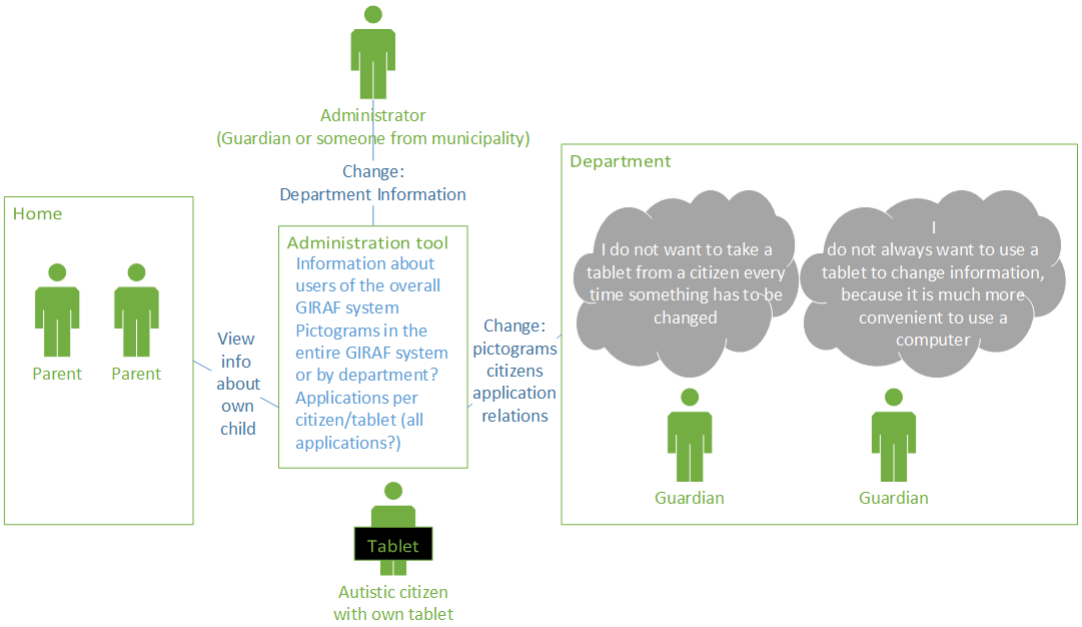

Figure 4.1: A rich picture of the system as it is understood at this time.

4.3 Architecture

The website is implemented using PHP. We chose to implement the website using the MVC pattern. The MVC pattern contains three layers which is the model, view and controller layers.

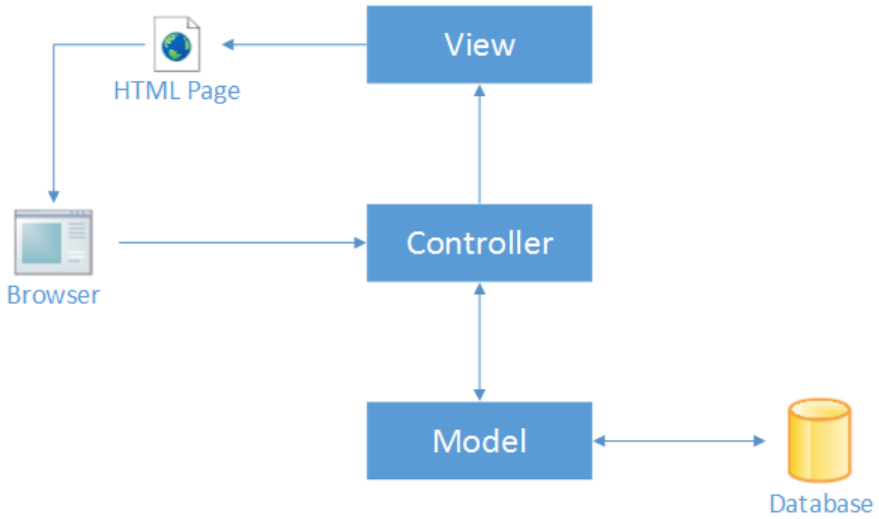

Figure 4.2: MVC Pattern

The following is based on [4][5]. Each layer have different responsibilities. Typically the model

is the layer which takes care of everything related to the database. That means that database access is restricted to the model layer. This has the advantage that if the database changes, then it should only be changed in the model layer and not in the other layers, except if the change introduces changes that needs to be presented to the user. In principal when designing the model you do not have to think about how exactly the information should be used but only making sure that you provide all the required information.

The view layer has the responsibility of showing information to the user. In our case it means that the view layer has the responsibility of defining the HTML that is shown to the user. The view layer contains a minimal amount of PHP, as it only contains PHP used to generate the needed HTML, and the information needed to generate it is provided by the controller layer which is the last of the three layers.

The controller layer is contacted by the browser when the user performs an action on the website. The controller is the link between the model and view layers, so when a user wants to load a certain page, the controller for that page is contacted by the browser. The controller will then, if needed, send and receive information to and from the model layer, and parse it to the view layer which is then sent back to the browser to be presented for the user.

Throughout the development of GIRAF Web Admin a naming convention of the controller actions will be to create an additional action for each of the pages containing HTML forms with "Form" appended to the action name. The form will submit to this page, and the "form"-action will, after performing validations, redirect to the actual page without the "Form" appended. For this reason a "form"-action does not have an associated view.

The browser then visits the page through: `http://cs-cust06-int.cs.aau.dk/webadmin/`. The way this address is structured is done like so:

```
http://cs-cust06-int.cs.aau.dk/webadmin/{controller}/{action}/{p1}/{p2}/{p3}/
```

Where the controller is any controller for example Image and the action is any method within that controller, for example jpg. And finally where the p's are the actual parameters of the controller action. This could be put together to call the jpg action in the Image controller to get an image with the id 1

```
http://cs-cust06-int.cs.aau.dk/webadmin/Image/jpg/1/
```

If, however, you want to call a form action in this architecture, the parameters have to be sent using either GET or POST. For GIRAF Web Admin we will use POST form actions primarily, these send the information to the server and they can be accessed through a super global array in PHP, called `$_POST`. Two things are needed for the `$_POST` array, a key and a value. This key becomes the name of the field used and the value becomes the value of the field. It should be kept in mind that different ways of calling the form action are possible, and it is not always the name and value of the field that is the relevant data.

A model of the MVC pattern can be seen in Figure 4.2.

We decided to use this layered architecture because of the minimal interface between layers making it easier to add or replace components of the website. We also think it helps making cleaner code by separating the different parts into the appropriate layers and files with different responsibilities.

The MVC skeleton we used is based on [6].

The file and folder structure can be seen in Figure 4.3. There the MVC structure is apparent. The models can be found in */application/model/*. The controllers can be found in */application/controller/*. The views can be found in */application/view/*. All CSS, JavaScript and images can be found in the */public/* folder. The files in the application folder can not be opened directly in the browser, which is also why CSS, JavaScript, and images have its own public folder.

```
./ ..... root directory
├── application ..... Containing MVC components
│   ├── config ..... General configuration
│   ├── controller ..... Containing all controllers
│   ├── libs ..... Base Controller and Application class
│   ├── model ..... Containing all models
│   └── view ..... Containing all views
├── public ..... All style, images and JavaScript
│   ├── css ..... All stylesheets
│   ├── js ..... All JavaScript files
│   ├── images ..... All images
│   └── imagecache ..... All cached pictograms and application icons
└── index.php ..... Entry point for the website
```

Figure 4.3: Structure of the folders

An example model and controller can be seen in respectively Figure 4.4 and Figure 4.5.

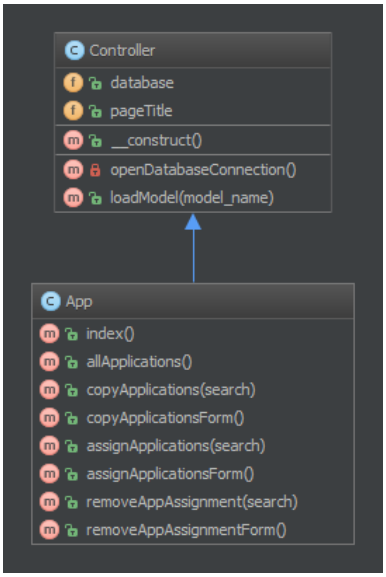

Figure 4.4: Example Controller.

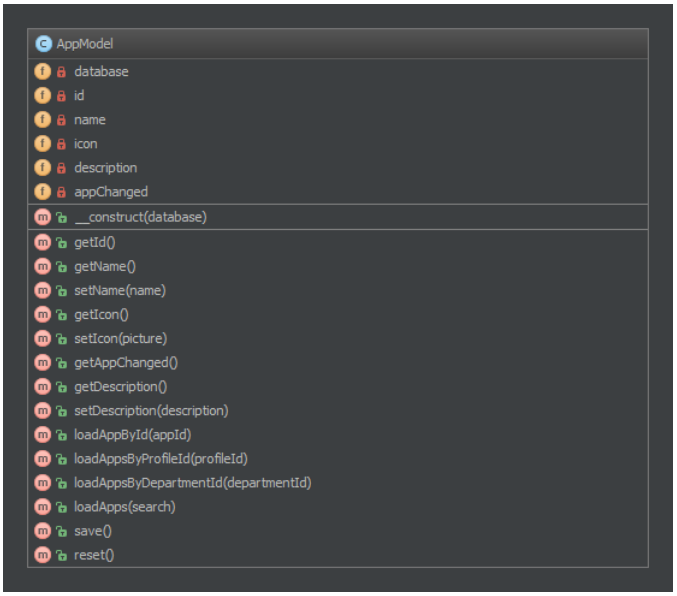

Figure 4.5: Example model.

4.4 Implementation

The most important features implemented throughout Sprint 2 will be described in this section. A description of less interesting features can be found in Appendix D.

4.4.1 Create Profile

During the project, a decision about not having any personal information about each user of the system was made at a status meeting for the multi project. This means that we cannot store any information such as addresses, phone numbers, e-mails etc. in the database for profiles, which leaves the creation of a profile to a type, being either guardian, citizen or parent and a profile name. The graphical user interface for creating profiles can be seen in Figure 4.6.

Figure 4.6: GUI of Create Profile.

The implementation uses two different actions in the `Profile` controller, one for displaying the corresponding view called `createProfile` and one for making the server side validation of the information in the action called `createProfileForm`. The `createProfileForm` action redirects to `createProfile` and as such has no view associated. It also validates the submitted information from the form and either creates a profile or displays an error massage. When validating the information submitted, sessions scoping the entire website is set containing either an error message or a success message. These messages are displayed to the user as either a red or green text. Examples of success- and error messages can be seen in Figure 4.7.

Figure 4.7: Success- and error messages.

The constraints for creating a profile are as follows:

> Profile type $\in \{2, 3, 4\}$ represented as list item {Parent, Guardian, Child}, where the profile type 1 is an Administrator.
>
> Name is not null.

4.4.2 Add Relation Between Profiles

A feature for relating guardians and parents to citizens is implemented for illustrating the relationship between guardians and the citizens they are guardians of, and to relate a citizen to its parents. The visual design can be seen in Figure 4.8, also corresponding the View of the Action `addRelation`.

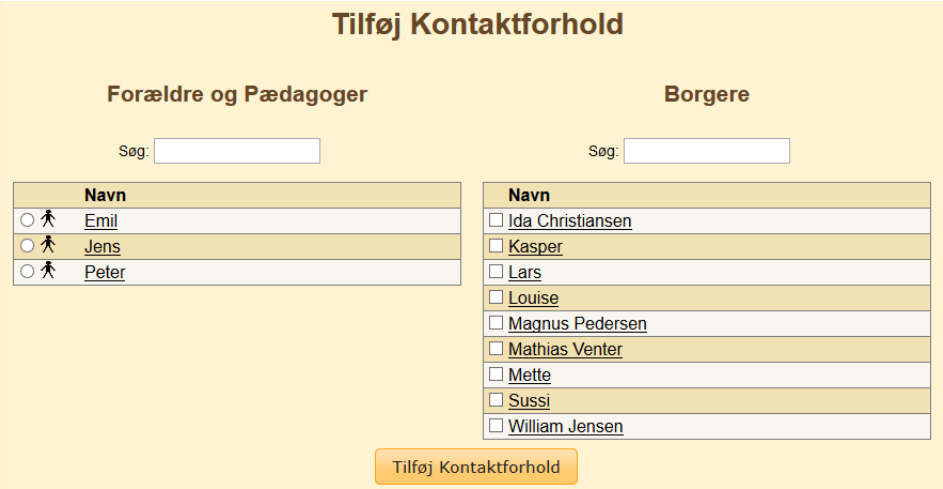

Figure 4.8: GUI of Add Relation Between Profiles

The feature is implemented such that a list of parents and guardians is present to the left, and a list of citizens is present to the right. The user can then via radio buttons select a parent or guardian and via check boxes select one or more citizens. We though it would be more practical to add more citizens to the same guardian, rather than adding more guardians to the same citizen. An error message will be shown to the user if the following criteria is not fulfilled:

Exactly one parent/guardians is chosen.

At least one citizen is chosen.

4.4.3 Profile Model

To handle the creation of profiles and creation and deletion of relations, `ProfileModel` is implemented. Through this model we can call a function to load a specific profile from the database. After this we can either edit the profile information and call the `save` function to update the database with the updated profile information. It is also possible, after loading a profile, to call `setChild` or `deleteChild` to add or remove citizens from the loaded profile. To create a profile an empty instance of the profile class is used. After adding the necessary information to the profile instance a call to the `create` function will add the profile to the database. Besides getters and setters for the private variables, these are the functions implemented:

- `loadProfileById(id)`
- `loadProfileByUserId(id)`

- `loadProfiles(type, search)`
- `loadProfilesByDepartmentId(id, type, search)`
- `getChildren()`
- `getGuardians()`
- `setChild(id)`
- `deleteChild(id)`
- `setGuardian(id)`
- `deleteGuardian(id)`
- `save()`
- `create()`
- `delete()`
- `reset()`

Where the id is the id that the profile or user has in the database. The type is an enum indicating the profiles to load where 0 = `All`, 1 = `Citizens`, 2 = `Guardians`, 3 = `Parents`, 4 = `GuardiansParents`, 5 = `GuardiansParentsCitizens`.

4.5 Summary of Sprint #2

At the end of the sprint it was decided we would continue on this project because a lot of functionality was still missing from the site itself. From the requirements identified we implemented the following completely or partially:

- Log In
- Log Out
- Edit Profile (partial)
- Create Profile
- Remove Profile
- Create User (partial)
- Add relationship between profiles
- Remove relationship between profiles

That leaves the pictogram and partially completed requirements for Sprint #3, where new user stories will be added based on discussions with the users as well. From the collective sprint end we also received comments and suggestions. One suggestion we received was that there should be added an overview page where you can see the relationships of different profiles. Another suggestion was that it should also be possible to add citizens and parents at the same time. Comments include that the language should be entirely Danish, tables should be sorted alphabetically, terms in the system should be clearer and consistent, and that we should consider what restrictions there should be for who could do what.

5 Sprint #3

This chapter covers the work done through Sprint #3. In this sprint further requirements analysis is performed with a meeting with the customers. Furthermore user stories are defined and various implemented features are described and tested.

5.1 Requirements

In this section we will present the user stories for the sprint and the requirements we got from visiting the customers in the kinder garden Birken.

5.1.1 User Stories

After Sprint #2, the backlog was almost empty, so user stories needed to be added. The following user stories were identified as necessary for the site from the initial requirements document created by another group in the GIRAF multi project and added to the backlog of Sprint #3.

- As a guardian I want to have a searchable list of all profiles within my department.
- As a guardian I want to edit my profile, such as my name, profile picture, QR code and password.
- As a guardian I want to edit others profiles.
- As a guardian I want to view the profile of all profiles within my department.
- As a guardian I want to create new users for my department.
- As a guardian I want to remove users from the system.
- As a guardian I want to upload pictograms, with titles and texts for the system.
- As a guardian I want to make a pictogram available for a citizen in the system.
- As a guardian I want to make a pictogram unavailable for a citizen.
- As a guardian I want to categorize pictograms.
- As a guardian I want to remove a pictogram from a category.
- As a guardian I want to view the profile of my department.
- As a guardian I want to edit the information of my department.
- As a guardian I want to make an application available for a citizen.
- As a guardian I want to make an application unavailable for a citizen.
- As a guardian I want to copy application assignments from one citizen to another.
- As an admin I want to have a searchable list of all departments.

- As an admin I want to create new departments.
- As an admin I want to remove departments.
- As a user I want all lists of relations to be searchable.

5.1.2 Requirements From the Kinder Garden Birken

During this sprint we visited the department Birken in Vodskov where we spoke to two of the employees. In preparation we wrote relevant questions for the parts of the problem domain we needed a clearer understanding of, such as the glossary, pictogram functionality and restrictions of users, for all the questions see Appendix E. Based on the interview we got the following requirements, some of these are very specific to Birken, these will be specified with an asterisk at the end.

- Children should be called "Barn/Borger" and parents should be called "Værge/Forældre".
- In the relation between a guardian and a citizen, the guardian should be called "Kontakt".
- A feature to edit the appearance of pictogram is not necessary on the website. Upload is enough.
- They would like to be able to delete pictograms, but they should not be able to delete pictograms used by other departments.
- The names, and perhaps the inline text, of the pictograms should be editable as local changes to a single or some citizen, and not be global changes.
- They would like to control pictogram categories on the website.
- No private information on profiles.
- It is not needed to be able to play the sound through the website.
- Parents should only be able to access information specified by the guardians.
- Parents should only see the pictograms assigned by the guardians.
- The guardians would like to be able to add and remove profiles. It should not necessarily be done by administrators.
- The guardians would like to be able to copy pictograms from one profile to another.
- QR codes should be shared between all employees in the department. *
- QR codes should be used to lock a citizen to an applications and unlock it again. *

That is enough requirements for the backlogs of Sprint #3 and #4, however with the limited time of Sprint #4 we will likely not be able to fulfil all of them.

5.2 Implementation

A more general approach to the implementation compared to the implementation section in Sprint #2 is given in this section. A description of how the upload of images, the search feature, the caching of pictograms and the department pages is implemented follows. More trivial implementation from this Sprint can be found in Appendix D.2.

5.2.1 Upload Images

To add new images to the database the user is able to upload existing images. The upload is implemented using the PHP superglobal `$_FILES` array which contains information about the uploaded files. The front-end of the upload is done using a standard HTML form. Both the HTML form as well as the PHP limits the file type of images to JPEG and PNG. The sound upload should be limited to only MP3, primarily because this is what Google text-to-speech provides. This is only enforced by PHP, though, as the file type can not be enforced by the HTML form. Because of this the HTML form accepts all audio formats.

To add the image and sound to the database the `PictogramModel` is used for pictograms and `ProfileModel` for profile picture which contains no sound. The upload is handled by creating a new instance of one of the models and then using the `setImage` function. The image and sound is stored in the database as binary data. When retrieving the data from the database the browser is told to parse the binary data as an image or sound, using `header("Content-Type: image/jpeg")` for images and `header("Content-Type: audio/mpeg")` for sound.

5.2.2 Search

Throughout the website several search fields are implemented in order to filter the list shown. This feature is implemented using Asynchronous Javascript and XML (AJAX). AJAX can be used to send HTTP requests asynchronously with Javascript, providing a more responsive website compared to synchronous requests. Using a library written on top of Javascript called jQuery eases the use of AJAX and can be simplified to the command `$.load()` taking a HTTP URL as input. Using jQuery to load a page using the input of a search field with AJAX every time a key is pressed and 500 milliseconds has passed, gives a user experience of a search interactively responding when they are typing. This search works by actually searching in the database, and replacing the table with the new information, a different way of doing it would have been to filter existent data. In order to provide the user with a response indicating that a search is processing, a loading animation is shown to the user.

List Ordering

The output from the database is a sorted list first sorted by department id, then by profile name in an ascending order, hence every profile in the department with id 1 would be listed first in an ascending order by profile name. For example if Anders is the only person starting with an A in the first department, he would be the top item of the list, provided that the search string either is empty or part of his name.

5.2.3 Caching

When showing images from the database in the website, the images are cached on the server to lower the data transferred between the website and the database. This is done in the `Image`

controller since this is the controller that is used to show images from the database. On the first request to an image, it is fetched from the database and stored locally on the web server. On subsequent requests to the same image, it will be loaded from the local copy instead of the database. Currently the local image is not changed or removed since there is no option to edit the images already uploaded. Only data like name and in-line text can be edited, but that does not change the image data.

5.2.4 Department

This subsection describes the various functionality implemented for the Department pages. This includes the controller and the model for it, and associated views. It is possible to view and edit the department a profile belongs to, view a list of all departments and search among them, create department, and remove departments. It is important to note that some of this functionality is restricted, the ability to create, remove, and view all departments lies with the administrators, not with the guardians that belong within the department.

We have created a `DepartmentModel`, a `Department` controller, and four view pages. `DepartmentModel` is a class that contains information for a department such as id, name and address. It also handles SQL calls to the database so the model can be loaded, changed and deleted.

Own Department

This view `ownDepartment` consists of a few elements. First is a table that contains basic information regarding the department which is the name, the address, the telephone number, the email, and if they have a superdepartment(e.g if they are a subdepartment) the name of the superdepartment. This information is subtracted from the `DepartmentModel` which is loaded from the database with the function `loadDepartmentById(currentId)`, the `currentId` is provided by the `ProfileModel` of the logged in profile. Then it contains two more tables, the first with all guardians and parents within the department and the second with all the citizens within the department, they show their name and an icon that indicate their role in the system. The content for these two tables are both loaded through the `ProfileModel` with the function `loadProfilesByDepartmentId(depId, profileType, search)`, the `depId` is the id of the department being viewed, the `profileType` is the type of profiles we want to load and the `search` is for the dynamic jQuery described in Section 5.2.2.

There is a View Department action that does essentially the same, but for a specified department.

Create Department

The `createDepartment` view contains a form that the user has to fill out. It contains name, phone number, email, address, city, zip code and superdepartment. If anything received from the form was incorrectly filled out or missing the `Department` controller sends an error message to the user. For example if the zip code and the city does not correspond. If there are no errors the `DepartmentModel` receives the information the user filled in and the function `create()`, which inserts the information for the new department into the database, is called.

Remove Department

The remove department page is implemented as a table that lists the available departments. This is done by the `DepartmentModel` with the function `loadDepartments(search)`, where

`search` is for the dynamic JQuery described in Section 5.2.2. The user have the ability to select one or more of those departments, which is sent to the `Department` controller which loads the associated `DepartmentModels` and calls the `delete()` function on them to remove them from the database.

5.2.5 User Actions

The following documents the actions related to administering users. These actions are incorporated into the `Profile` controller, because they are strictly related.

Create User

The `createUser` view is implemented as a form that a Guardian or Admin user has to fill out. It shows what department the new user's profile will be added under. It has a username, password, repeat password field and the option to choose an existing profile. It also allows for creating a new profile. The backend of how this is implemented is fairly simple. The form sends its content to the `createUserForm` action in the `Profile` controller and verify that the user is at least a guardian, and if not it will redirect. If the current user is at least a guardian however, it will create an empty profile, load the right department id, and the author and start verifying the data. If there are no errors, it will use the `UserModel` to create a new User and use the `ProfileModel` to create a new Profile if an existing profile was not selected. If the creation succeeds it will send a success message back, or if it does not succeed it will send an error message.

Remove User

The `removeUser` view is implemented as a table of existing users, that the current user has access to remove. It provides an icon to show what kind of profile they have, what their name is, a link to the associated profile and a checkbox to select the user for removal, along with a button to submit the information to the associated action in the `Profile` controller. The `removeUserForm` action in the controller is the action that processes the submitted form. It verifies that the user is at least a guardian, and if not it will redirect. If the user is at least a guardian it will load the user ids that it is supposed to remove from the database, load the User Model and get the currently logged in user. It will then check if the loaded user ids is not empty, because if it is it means that no users were selected for removal and that will result in an error. If not, it will loop through the ids, load the associated user, check that it is not the currently logged in user and that the current user has permission to delete the user, if it does the user is removed. If it is the currently logged in user or if the user does not have permission it will result in an error. Finally it will redirect back to the `removeUser` view.

5.3 Test

This section covers the usability test of the previous group and how they applied to our implementation of similar functionality. Tests for features implemented in Sprint #3 can be read in Appendix B.3.

5.3.1 Usability Test - Previous Group

The previous group identified a series of usability problems. We decided to use those, and compare them to our current system to get an idea of what was a problem in both, and what

was not a problem in ours, and what could be immediately fixed. There were of course problems identified on the previous site that did not apply to the current site, specifically the pictogram problems, and as such they were removed from the list of usability problems. See Table B.1 for the full list of usability problems.

Description	Severity	Corrected
Profile - Restrict QR editing to department manager	Critical	
Profile Pic - GIRAF logo as placeholder is misleading	Cosmetic	✓
Design - Standardise button names	Cosmetic	

Table 5.1: Bugs/Errors Found Under Usability Testing. Problems taken from [3, p. 79], edited to remove irrelevant problems.

The first usability problem common in both systems is the fact that every normal user could generate new QR code. In the future a way to avoid this would be to set up a configurable permissions systems about who is allowed to do that in the system.

The next is the default profile picture being the GIRAF logo. We created a new logo that looks like a person and use that as the default profile picture instead.

The next is the design problem where buttons have different text for similar functionality. We fixed this by ensuring every button that did some action similar to an existing action had the same text.

5.4 Summary of Sprint #3

This section will cover how Sprint #3 went, and what items are left and can be put into the backlog for Sprint #4. Much of the backlog was completed, as such it is simpler to identify which were not completed.

- Assign pictogram to categories – Assign pictograms to a category(A set of pictograms, with some kind of theme).
- Remove pictogram from categories – Remove the assignments from categories.
- They would like to be able to delete pictograms, but they should not be able to delete pictograms used by other departments.
- The names, and perhaps the inline text, of the pictograms should be editable as local changes to a single or some citizens, and not be global changes.
- They would like to control pictogram categories on the website.
- Parents should only be able to access information specified by the guardians.
- Parents should only see the pictograms assigned by the guardians.
- The guardians would like to be able to copy pictograms from one profile to another.
- QR codes should be shared between all employees in the department.

- QR codes should be used to lock a citizen to an applications and unlock it again.

The time remaining is limited, there are two weeks until the last sprint end. That leaves one and a half week for actual development, so it is important that the backlog is very short and only highly prioritized stories are chosen for development, the rest of the time is for testing and bug fixing. The backlog for Sprint #4 is constructed from the feedback of the customers at this sprint end.

- Search among the pictograms in the database.
- Remove pictograms from profiles
- Delete pictograms, in the database, if it is not used by any profiles.
- Parents access to the web page should be restricted and read only.

6 Sprint #4

This chapter covers the work done through Sprint #4. This sprint was primarily refactoring with implementation of a few new features. Mostly we polished existing features and got the codebase ready for deployment, and to make the code simpler and easier to understand for further development in the next year.

6.1 Refactoring

Refactoring was a big part of Sprint #4, we worked on documenting undocumented code, inconsistent variable names and not following a code standard. Since the project is to be handed over to another group that will continue to work with it, it is important to make it as consistent and readable as possible. We got input from another group through a code review. They went through our code and found flaws in the code, for more detail on this see Chapter 8. A standard that we have adapted with the model classes is placing all the fields at the very top of the class, followed by all the getters and setters, which gives a good overview of the class. Furthermore we wrote summary comments for each function, to give subsequent developers a better understanding of the meaning of the formal parameters. We adapts the naming convention of variables, that all variables should be in camelCase and that there should not be any shorthand letters unless it is used in a loop, and the letter is descriptive enough for the programmer to realize what it is. Furthermore we adopts a standard for HTML class names using a dash (-) between each word and using only lower-case letters, for example `body-content`.

Further refactoring of the code was not done because the end of the project was reached. A further refactoring of the functions in the model classes should be considered. The creation of factory classes for each of the model classes would make the purpose of the model classes clearer. The new objects would not be instantiated in the model classes but instead be produced with the matching factory class. E.g. the Profile factory would be responsible for returning a list of Profile model objects, rather than the Profile model class containing a function for returning a list of objects of its own class. Perhaps this could be a future consideration.

6.2 Implementation

This section covers the implementation done in Sprint #4. This covers cropping of profile pictures, removing applications from citizens, search in pictograms and assignment of pictograms.

6.2.1 Crop Profile Pictures to a 8:10 Ratio

The Launcher group set a requirement that profile pictures should be 8:10 in aspect ratio, that is the width should be 8 wide when the height is 10 high. This was implemented using a jQuery plugin called ImgAreaSelect, that allows selecting an area of an image, and then set the relevant values which can be submitted to the controller. The way we use it is we have hidden input fields that the plugin sets the value for, specifically we have the x and y position of where the top-left corner of the selection is, and the height and width that can be used to determine the area that was selected. We also have the shown height and the shown width of the image that is used to scale the selection up. This data is then sent to the `setProfilePicture` action, where

the cropping is done by first getting the relative location of the top-left corner of the selection by dividing x and y with the shown width and the shown height and we also get the relative height and width by dividing the selected height and the selected width with the shown height and the shown width. We do this because we were not working on the 'real' image in the sense that the dimensions of the image might have been changed and we need to figure out how to apply the crop to the real image, and with what we have now we can calculate the actual x and y along with the actual height and width. Cropping is done using a PHP function called `imagecrop`. If the image is still too big then the image will be resized using `imagecreatetruecolor` and `imagecopyresampled`, to a maximum of 160 pixels wide and 200 pixels high.

6.2.2 Remove Assignment of Applications from Citizens

The removal of application assignments from citizens feature is implemented with a view that lists all citizens, with their associated applications and a checkbox for each application. The list of citizens can be searched using a search feature. If any checkboxes are checked and the button is pushed, it will submit the citizen id(s) and the application id(s). These two are then split inside the `removeAppAssignmentForm` action and the citizen id is used to load the profile while the application id is used to unset the application. If more than one citizen application pair was sent over, the action will loop over all of them. It will then set a success message, if no pairs were sent, it will just set an error message and then finally redirect back to the main action.

6.2.3 Search Pictograms

Because of the huge amount of pictograms in the database we implemented a search feature. The feature is used when assigning pictograms to profiles. When the user visits the Assign Pictogram page and no search have been made, then the search page is loaded using AJAX. On the search page the user is presented with a few search options to narrow the search such as choosing to search in name or in tags and if the search should take place in department pictograms, or the pictograms the current user has created or just all public pictograms in the system. This information is sent to the `searchForm` action in the `Pictogram` controller, which uses the `PictogramModel` to do the search. The model returns an array of pictogram ids which is stored in a session, then the user is redirected back to the original page where the search result should be presented. Because of the potential for many ids they are split into pages, and the database is then queried for one page at a time to limit the amount of data that needs to be fetched at once. Since the search result is stored in a session, the result can be reused without performing a new search.

List Ordering

The returned list of pictogram ids is ordered by the database by the string distance from the search string in ascending order. The actual algorithm that is used to calculate this distance is called Levenshtein Distance algorithm[7], and gives the number of characters that needs to be edited in order to make the two string match. We used a pre-made SQL version of this algorithm [8]. By doing this, a search in the database on the string "hest" would return "heste" as the first item rather than when no ordering is done returning "flodhest" (search string is in Danish because of a Danish database). So it is a way to get more relevant results first.

6.2.4 Assign Pictograms To Citizens

The Assign Pictograms page consists of two steps, first the entire content area is loaded with a search form that contains different functionality depending on whether its is a guardian or an administrator that is logged in. Afterwards the content area is loaded with a list of pictograms to the left, the result from the search, and a list of citizens to the right. The difference in search functionality consists of different queries. It was decided that an administrator would have no use of searching in the sets of pictograms of the department or privately created pictograms, since the administrator is not able to upload and create new pictograms, and therefore would not have any pictograms to search for in these sets. A guardian on the other hand can search in all, department-wise and privately created pictograms, because a guardian is able to create pictograms under the department and privately.

The page takes one parameter namely the currently shown page of the search result, see Section 6.2.3 for further explanation of this implementation. The list of citizens will also depend on the type of user that is logged in, because a guardian can only see the citizens registered in the same department as the guardian, whereas an administrator is able to see all citizens registered in the system. When the user has selected a number of pictograms and a number of citizens and submitted the information, several server-side validations are performed before the database is updated. First of all, there is a check on the emptiness of the two list, they cannot be empty. In case they are not empty, there is a check on the validity of the supplied citizen ids and pictogram ids. A list of allowed citizens and a list of allowed pictograms is compared against to check if the supplied ids is valid. The check is to see whether setdiff in (6.1) is empty or not. If not, one or more of the supplied ids is invalid and a proper error message is given. In case no errors was found during the validity check, the database is updated with the relations between pictograms and citizens in the `profile_pictogram` table, and a proper success message is given.

$$\begin{aligned} \text{allowed} &= \{1, 2, 3, 4\} \\ \text{supplied} &= \{4, 34\} \\ \text{intersect} &= \text{allowed} \cap \text{supplied} = \{4\} \\ \text{setdiff} &= \text{supplied} \setminus \text{intersect} = \{34\} \end{aligned} \tag{6.1}$$

6.3 Test

Test of implemented functionality during this sprint can be found in Appendix B.4.

It was decided that usability tests would not be performed. This decision was made primarily due to time constraints. If we had done usability testing, it would have been done using Instant Data Analysis. We would have presented a series of tasks to the tester that they would have to accomplish. We would then observe which tasks they found difficult, and categorize those problems under either "Cosmetic", "Serious" or "Critical".

6.4 Summary of Sprint #4

This section will cover how Sprint #4 went.

During this sprint, much less actual development of new features for the site was done and instead refactoring, commenting, cleaning the code up was done. In general this sprint was spent making the project ready for subsequent developers to start working on it.

During the sprint the website became available on the web at:

```
http://cs-cust06-int.cs.aau.dk/webadmin/
```

This deployment to a different server resulted in one issue, specifically broken functionality. This broken functionality is caused by a last-minute change of database, which resulted in some attributes of the tables not being present, to be specific the `inline_text` attribute of the `profile_pictogram` table. The feature not working because of this is editing the inline text for pictograms for each individual citizen.

That ends the description of the sprints, what follows are common activity chapters.

7 Common Activity Between Oasis Application and Online Administration Tool

This chapter was written in a collaboration between the groups sw606f14 and sw614f14 and is in both reports.

This chapter documents a common activity between the groups sw606f14(This is our group) and sw614f14. The common activity has been done because both groups have been working on administration tools. To be clear, the group sw606f14 is making an administration tool to be accessed through a website called GIRAF Web Admin and the group sw614f14 is making an administration tool for tablets called Oasis.

7.1 Motivation

In general the primary focus of this chapter will be the requirement analysis of the shared functionality of these two projects.

The benefit of analysing the shared responsibility between these applications is, it will streamline the two applications and provide a guideline for both groups as to what they should be developing. The primary idea behind the tablet application, Oasis, is to enable the individual guardian to administrate profiles, such as the "citizens", e.g. change their names or add new "citizens" under their care as well as administering departments and relationships between guardians and citizens. Meanwhile the primary idea behind the website administration tool is to control more than just profiles and departments. For example printing the QR codes used to log into the overall GIRAF multi project and allowing guardians to upload and assign pictograms to individual citizens and control their access to applications on their tablets.

There exist a common requirements document created in the beginning of the semester by another group. The document covers requirements in depth for the applications that the citizens and guardians use for communication but only scratch the surface of the administration tools. Group sw606f14 will therefore meet with the customer from "Specialbørnehaven Birken", who takes care of citizens up to the age of 7 with autism. The purpose of the meeting will be to understand their current administrational work flow, what they are interested in from an administrational tool in the context of the larger GIRAF multi project, and to show and obtain feedback on the GIRAF Web Admin they have been working on during the last sprint. From the meeting it should be possible to create a more concrete requirements document for the administration tools. While those requirements apply directly to the GIRAF Web Admin, it is also possible to derive required functionality for the Oasis Application.

7.2 Results

The interview was performed, and the topics discussed can be viewed in Appendix E. From the interview the following requirements can be created for the online administration tool.

- Children should be called "Barn/Borger" and parents should be called "Værge/Forældre".

- In the relation between a guardian and a citizen, the guardian should be called “Kontakt”.
- A feature to edit the appearance of pictogram does not need to be editable on the website. Upload is enough.
- They would like to be able to delete pictograms, but they should not be able to delete pictograms used by other departments.
- The names, and perhaps the inline text, of the pictograms should be editable as local changes to a single or some citizen, and not be global changes.
- They would like to control pictogram categories on the website.
- No private information on profiles.
- It is not needed to be able to play the sound through the website.
- Parents should only be able to access information specified by the guardians.
- Parents should only see the pictograms assigned by the guardians.
- The guardians would like to be able to add and remove profiles. It should not necessarily be done by administrators.
- The guardians would like to be able to copy pictograms from one profile to another.
- QR codes should be shared between all employees in the department.
- QR codes should be used to lock a citizen to an applications and unlock it again.

A project to implement an administration tool for the tablets was started in 2012 but the development was not continued in 2013. However, in 2013 the development of the online administration tool begun and there was a plan to port the website to Android using Pro Active Webfilter which, according to the report documenting the development of the administration tool, can be used to run applications developed in PHP on Android[3]. However, the group developing the online administration tool in 2013 was prevented from creating the port to Android and there was therefore not any development on the Android administration tool in 2013.

Group sw606f14 who were working on the administration tool this year would focus on the desktop version of the application and would not look into porting the application to Android. Sw614f14 found that they had enough time to work on both the local database, OasisLib as well as the Oasis application so they overtook the development of the application.

As previously mentioned, the interview is primarily concerned about the online administration tool. This is a result of the website providing a much broader functionality than the Oasis application because there exists other applications for the tablet that focuses on some of the functionality. This includes the “Piktotegner” application that manages the creation of pictograms and “Kategori værktøj”, the category manager that organizes the pictograms in categories.

The requirements that apply to the Oasis applications are the following.

- Children should be called “Barn/Borger” and parents should be called “Værge/Forældre”.
- In the relation between a guardian and a citizen, the guardian should be called “Kontakt”.

- No private information on profiles.
- Parents should only be able to access information specified by the guardians.
- The guardians would like to be able to add and remove profiles. It should not necessarily be done by administrators.

These requirements along with the report on the development of the Oasis application in 2012[9] was used to outline the requirements for the Oasis application in development by the group sw614f14.

8 Common Activity Between GIRAF Web Admin and Piktotegner

This chapter was written in collaboration between the groups sw606f14 and sw608f14 and is in both reports.

A common activity was performed between the *GIRAF Web Admin* (sw606f14 - This is our group) and *Piktotegner* (sw608f14) groups, where a code review was done. It was done in order to secure a better code quality, to find bugs, and as a learning experience. By code quality meaning e.g. suitable comments, reducing redundancy, and following the code standards, camel casing for example.

The code review was constructed in an informal way, as it made us able to cover more code in the same time span as a formal code review. The formal code review may find more serious bugs, however, it would only be able to cover a limited amount of code with the time at hand, thus the prioritisation of an informal code review over a formal version.

8.1 Approach

In order to secure that the two groups understood each others applications' architecture, a presentation was performed before the code review. For *GIRAF Web Admin*, it was told how it is based on the model-view-controller pattern. For *Piktotegner*, a class diagram showing how entities are structured as well as their respective handlers were presented. This gave insight to how the code, to be reviewed, was structured, in order to avoid confusion that could help us get to review more code and to give a better review of the code.

The review was done in a pair of two, such that two groups from *GIRAF Web Admin* reviewed *Piktotegner* and vice versa. The presentation was the first time the groups were presented to each others code, and as of such we sat in the same room when reviewing. This ensured that if any questions arose, regarding the architecture and language specific operations, a group from the other application could then fast be consulted.

The corrections and suggestions found during code reviewing was documented with the file name, the line number(s), and a description.

In hindsight, it would be good if the two groups had more experience in the languages of the code and the problem domain of the application they reviewed, as *GIRAF Web Admin* is primarily programmed in *PHP* and *Piktotegner* is programmed in *Java*. If the groups had more experience with these different languages, more bugs would likely be discovered, and code optimisations might have been suggested.. However, despite this, it was found that the code reviewing was helpful and is a practice that should be continued in the future.

We find it a good practice to review code, however, if the comments given on ones code is not addressed, the code review does not give much value. Therefore both groups addressed the problems found during the code review session as the first activity the day after the code review.

8.1.1 Experience

The overall impression of the code review activity was positive, and is agreed upon by both groups. About 2500 lines per project group, including comments, was covered by the code review, which may have been too much for a four hours code review. It is believed that instead of having a singular lengthy code review activity, it would be better to perform several shorter ones, in order to not get tired and do inefficient code reviewing.

GIRAF Web Admin was lacking comments to document their code, it was recommended that all functions was given a comment to concisely describe them. *GIRAF Web Admin* had inconsistency in their naming conventions. *camelCase* and underscores for variable names, *PascalCase* and underscores for functions, and *camelCase*, *PascalCase*, underscores, and dashes for *CSS*, were used. Furthermore, redundant code was found in the *PHP* classes and it was suggested that the similar code could be moved to a new function or separate library. Lastly, suggestions for optimisation of *SQL* queries was given.

Piktotegner had a lot of comments explaining both complex functions as well as the simple ones, some comments were deemed unnecessary or being superfluous. The variable names followed a coding convention but a few were found to not follow this convention or were found to have non describing names. Most properties had no constraints which might cause an error in the application if the properties were to be used improperly.

In addition to this, an opinion of both groups is that having people not familiar with the code, try and read the code, gives the advantage that it checks which parts of the code is hard to read and what code may need more documentation or refactoring.

9 Conclusion & Reflection

For Sprint #1 we worked with the Timer project for Android. Much of the sprint was used to understand the code and Android as a development platform and very little in terms of new features was developed. Since we did not develop much for the application very little collaboration with other groups was needed. Sprint #2 to #4 was spent working on the GIRAF Web Admin project. Because the GIRAF Web Admin works as a interface to the database and is used to upload data that most of the applications in the overall GIRAF multi project synchronizes with, more communication with other groups was needed.

9.1 Multi Group Cooperation

We had to establish contact with the remote database group to get access to the database and get documentation for the database schema. The Oasis/Admin group was creating a similar project for Android and we worked with them to understand what features were needed only for GIRAF Web Admin, what features were needed only for Oasis and what features should be covered by both applications to make them feel similar. It was decided that GIRAF Web Admin should have a design similar to Android applications in the GIRAF multi project, so we worked together with the GUI group to achieve this. These were the main three groups we worked together with to develop GIRAF Web Admin, however a lot of problems were solving through informal or ad hoc collaboration with other groups.

When problems arose or anything else where we needed to communicate with another group, it was done by visiting them in their group rooms. When visiting other groups it almost always ended with three scenarios, either the group could answer our question and we could continue our work. At other times a group member of the other group would come to us and stay to solve the problem. Else if the other group needed to implement something we wrote as an issue on redmine and they would inform us when they had solved it. This worked the other way around as well when groups came to us for help.

9.2 Experience And Recommendations

The first two sprints of the multi project was chaotic and messy. We had no experience of working together with other groups, because in all previous semesters a group had been a stand-alone entity responsible for the entire project and we all needed time to adapt to the new style of project work. We soon learned how important good communication is, whenever a group is working on something that will affect another groups work. Our website crashed once because the database schema was changed and we only got informed as they happened to change it. The project worked on are continued from previous semesters, though that is not entirely true for the GIRAF Web Admin project that we worked on. This makes the start of the semester very slow, since we only have the source code and report from the last semester.

From our experience with the multi project on this semester we have the following ideas for the future. A mini multi project could be introduced at the third semester, with projects such as a mini facebook. We also recommend any group to take contact to the former group who worked on the project to get an in-person introduction to the project. Further we recommend all groups to use redmine, the wiki feature has helped a lot with this project, while the issue tracker has

been less effective than we expected, this however is our own fault for not using it enough. Plan all sprints in the beginning of the semester, plan the days for sprint ends and give the dates to the costumers as soon as they are known.

Bibliography

[1] Kristian Kolding Foged-Ladefoged, Rasmus Hoppe Nesgaard Aaen, and Simon Blaabjerg Frandsen. Wombat: Part of the giraf system. Aalborg University Student Report, June 2012.

[2] Visual Paradigm. Visual paradigm for uml 11.0 enterprise edition. Online, 03 2014. URL http://www.visual-paradigm.com/product/vpuml/editions/enterprise.jsp.

[3] Jens M. Lauridsen, Johan Sørensen, Lars Chr. Pedersen, and Tommy Knudsen. Giraf - admin. http://projekter.aau.dk/projekter/da/studentthesis/giraf--admin(d4567010-eab8-4bfd-9a06-43dde4cef2ad).html, June 2013.

[4] Jeremy Morgan. What is mvc? Online, 04 2013. URL http://www.jeremymorgan.com/blog/programming/what-is-mvc/.

[5] Yii Framework. Best mvc practices. Online, 04 2014. URL http://www.yiiframework.com/doc/guide/1.1/en/basics.best-practices.

[6] Panique. Mvc skeleton application. Web, April 2014. URL https://github.com/panique/php-mvc.

[7] Many. Levenshtein distance. Web, May 2014. URL http://en.wikipedia.org/wiki/Levenshtein_distance.

[8] Chella. 'how to add levenshtein function in mysql?' answer. Web, December 2012. URL http://stackoverflow.com/a/13928762/1308250.

[9] Henrik Klarup, Jens Mohr Mortensen, and Dan Stenholt Møller. Oasis: Part of the giraf system. http://projekter.aau.dk/projekter/da/studentthesis/oasis-del-af-giraf-systemet(40e68aca-89a7-4d9f-806b-b73dfb8ac673).html, May 2014.

A Final UI

This appendix shows the final user interface of GIRAF Web Admin shown to the user when they have just logged in.

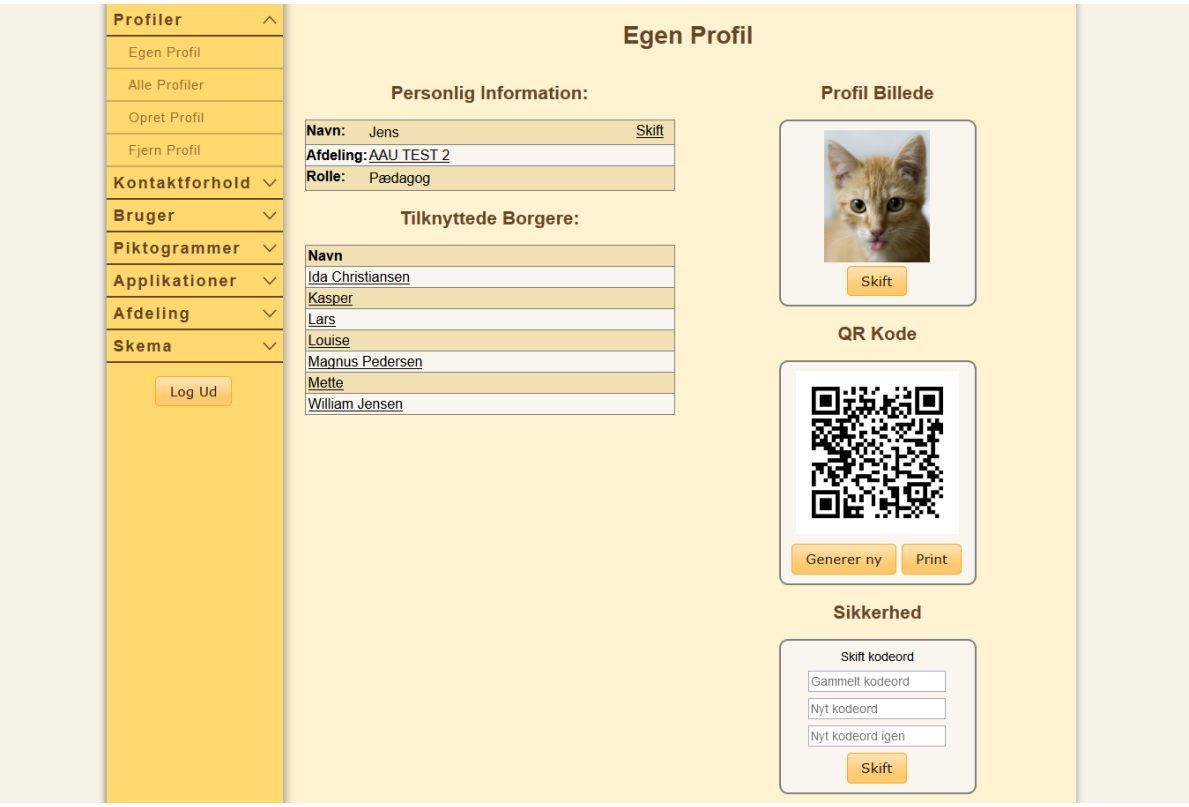

Figure A.1: Final GUI

B Test

This appendix chapter covers the tests done for Sprints #1-4.

B.1 Sprint #1

For Sprint #1 there was previously conducted a test on the product. These test cases will be listed below in the same notation as utilised in the previous report. This exact same notation will be adopted for the new extended test cases.

B.1.1 Test Cases From Previous Report

Test Case #1

Test Item
Functionality to save customised timers in specific lists.

Input Spec

1. Click "New Template", choose a timer, click "Save As", and choose name and location. Check if the profile was saved in the chosen location and with the chosen name.
2. Choose any configuration, edit the settings, click "Save As", and choose name and location. Check if the profile was saved in the chosen location and with the chosen name.
3. Create a new configuration with a random settings and use "Save As" to save it. Go to the saved configuration, and check if the settings has changed since the save.
4. Choose an existing configuration or create a new one, and do step 1 with "Predefined" and "Last Used" as save location.
5. Do step 1-3 again, but clear the tablet memory before the correctness is checked.

Output Spec

1. Configuration is saved in the chosen location, unless the chosen location is "Predefined" or "Last Used".
2. Configurations is saved with the chosen name.
3. Configurations is saved with the chosen settings.

Test Case #2

Test Item
Functionality to save customised or predefined timers in the highlighted citizen, without having to choose name and location.

Input Spec

1. Check if it is possible to save configurations in "Predefined" or "Last Used" by choosing one of the configurations in each list, edit some settings, and press "Save".

2. Select a citizen, edit the settings, and press "Save". Check if the chosen settings were saved.

3. Select a citizen, select a configuration among the citizen's configurations, edit the settings, and press "Save". Check if the chosen settings were saved in the same configuration.

4. Select a citizen, edit the settings, and press "Save" two times and see if two identical configurations are saved on the given citizen.

5. Highlight another configuration than the one you have just saved, then highlight the one you just saved and press "Save". Check if there is now saved a duplicate of the first saved configuration.

6. Do step 2-3 again and check if any other configuration was changed during the saving process.

Output Spec

1. Configurations is saved in the highlighed citizen.

2. When "Predefined" or "Last Used" is highlighted, nothing is saved when the "Save" is pressed.

3. New configurations are saved with the chosen settings.

4. When selecting and saving existing configurations, they are updated with the edited settings.

Test Case #3

Test Item
Functionality to save timers into the "Last Used" list every any timer has been run.

Input Spec

1. Run 3 different timers, and see if they were saved on top if the "Last Used" list.

2. Repeat step 1 but clear the tablet memory before "Last Used" is inspected.

Output Spec

1. Whenever a timer has been used, it is saved on top of the "Last Used" list.

Test Case #4

Test Item
Functionality to highlight list items when they are clicked.

Input Spec

1. Select three different list items in both the citizen list and the configuration list and see if they stay highlighted.

Output Spec

1. When a list item is selected, it is highlighted, and it stays highlighted until another list item is selected.

Test Case #5

Test Item
Functionality to highlight list items after a save procedure.

Input Spec

1. Select a citizen and a configuration, edit the settings for the configuration, and click "Save". See if the selected list items stay highlighted after it has been updated.

Output Spec

1. When a citizen and configuration is selected, and the settings for that configuration is changed and saved, the citizen list item and configuration list item is still highlighted.

Test Case #6

Test Item
Functionality to highlight list items according to the chosen citizen when the application is launched through the GIRAF launcher.

Input Spec

1. Select the GIRAF launcher and open the timer application.
2. Select a citizen and note the name of the citizen.

Output Spec

1. The citizen seledted in the GIRAF launcher is highlighted and the configurations belonging to this citizen is loaded.

Test Case #7

Test Item
Functionality which draws and updates the timer according to the time left and ensures the timer ends when the timer is up.

Input Spec

1. Run four different timer styles with a static timespan (fx 20 minutes).
2. Each time a timer is started, start a precise independent stopwatch.
3. When the timer reaches zero stop the independent stopwatch.

Output Spec

1. The stopwatch must deviate no more than two seconds from the time selected in input spec. step[1].

Test Case #8

Test Item
The "Done" screen appearing when the time has run out.

Input Spec

1. Start a timer at any timespan and let the time run out. See if the "Done" screen appears within two seconds after the time has run out.
2. Start a timer at any timespan and click the "back" button, and wait at least the amount of time the timer would have run, to verify that the "Done" screen do not show up anyways, if the timer has been interrupted.

Output Spec

1. When a timer has been run, and not interrupted, the "Done" screen appears about two seconds after the time has run out.
2. The "Done" screen is only shown if the timer is not interrupted, and the time has run out.

B.1.2 New Test Cases

Test Case #9

Test Item
Functionality to change the layout of buttons.

Input Spec

1. Press the "Skift Layout" button.
2. Press again.

Output Spec

1. Observe if the position of the buttons "Start", "Gem", "Vedhæft", "Slutbillede", and "Skift Layout" has changed.
2. Observe that the buttons again is in the original position.

Test Case #10

Test Item
Functionality to delete timer configurations from a citizen.

Input Spec

1. When a citizen is selected, select a timer configuration.
2. Press the bin logo in the right side of the selected timer configuration.
3. Press "Ja".

Output Spec

1. The selected timer configuration is deleted from the list and the database.

Test Case #11

Test Item
Functionality to attach another timer and pictograms, and to change the end pictogram.

Input Spec

1. Create a new timer configuration.
2. Select a timer type.
3. Set the "Vedhæft" to a:
    a) Timer
    b) Single Pictogram
    c) Double Pictogram
4. Set the "Slut Billede" to a:
    a) Single Pictogram
    b) Double Pictogram

Output Spec

1. Shows the timer configuration running along with the attachment chosen in the input spec.
2. In case of another timer is attached, the time should match the one of the original timer.

Test Case #12

Test Item
Functionality to change the colour of the timer.

Input Spec

1. Create a new timer configuration.
2. Select a timer type.
3. Change the colours of the "Tid Tilbage", "Ramme", and "Baggrund".
4. Run the timer configuration.
5. Repeat from step 2 until every timer style has been checked.

Output Spec

1. Observe that the colours of the running timer is changed accordingly.

B.2 Sprint #2

This sprint establishes a new project not previously documented, and thus a different notation of test is applied. Each test case is given an optional input specification, if appropriate, followed by a description of the expected output and the actual output for comparison.

B.2.1 Log In

It is fairly easy to test the log in functionality, there are three cases to test for:

#1 The user has valid username and password

Input: {Username: TestUser, Password: 123456}
Expected: The user is logged into the system.
Actual: The user is logged into the system.

#2 The user has valid username but not valid password

Input: {Username: TestUser, Password: 1234567}
Expected: The user is not logged into the system.
Actual: The user is not logged in. Message: "Forkert brugernavn eller kodeord.".

#3 The user does not have a valid username but a valid password

Input: {Username: TestUsers, Password: 123456}
Expected: The user not is logged into the system.
Actual: The user is not logged in. Message: "Forkert brugernavn eller kodeord.".

The system behaves as expected when it comes to log in.

B.2.2 Log Out

The user must be logged in to be able to log out and as such the only thing to check for is if the button "Log Ud" will behave as expected. The expected result of this action is of course that the user is logged out of the system. The system behaves as expected.

B.2.3 Own Profile

In case it should be needed a user is able to change his or her password for the current logged in user. The following tests was made to this feature while on the user John with the current password 123456:

#1 Test what happens if you enter the wrong current password

Input: {old : 12345, new : 123}

Expected: The password will not be changed.

Actual: The password was not changed.

#2 Test what happens if the old password is correct but the new passwords is different from each other

Input: {old : 123456, new : 1234, repeat new : 123}

Expected: The password will not be changed.

Actual: The password is changed to 1234.

#3 Test what happens if the new password is the same as the old

Input: {old : 123456, new : 123456, repeat new : 123456}

Expected: The password will be changed.

Actual: The password is changed to 123456.

There was an error in the second test where the "New password again" field was ignored. As a result the password would always be changed to what was in the "New password" field. The error was located and fixed.

B.2.4 Create Profile

For a profile to be valid, it has to be either a administrator, parent, guardian or citizen. When a profile is created a type and name is given as input, this is done with a drop down box for the selection of type and a text field to write the name. We do tests to check if we can create parents, guardians, citizens, that they can have the same name and that it is impossible to create administrators.

#1 Test if parents can be created

Input: {type = Parent, name = Tulle}

Expected: The profile is created and inserted into the database.

Actual: The profile is created and inserted into the database.

#2 Test if guardians can be created

Input: {type = Guardian, name = Tulle}
Expected: The profile is created and inserted into the database.
Actual: The profile is created and inserted into the database.

#3 Test if citizens can be created

Input: {Type = Child, name = Tulle}
Expected: The profile is created and inserted into the database.
Actual: The profile is created and inserted into the database.

#4 Test if administrators can be created

Input: {type = Admin, name = Tulle}
Expected: The profile is rejected and an error message is displayed.
Actual: The profile is rejected and an error message saying you can not create admin profiles.

#5 Test if more profiles of the same type can have the same name

Input: {type = Child, name = Tulle}
Expected: The profile is created and inserted into the database.
Actual: The profile is created and inserted into the database.

#6 Test if a profile is created without a name

Input: {type = Child, name = }
Expected: An error will appear.
Actual: An error message is shown saying you can not create a profile without a name.

B.2.5 Remove Profile

The remove profile functionality also needs to be tested. These are the test cases.

#1 Select one citizen and remove

Input: Any citizen.
Expected: The citizen is removed, success message is given.
Actual: The citizen was removed, success message was given.

#2 Select one guardian and remove

Input: Any guardian.
Expected: The guardian is removed, success message is given.
Actual: The guardian was removed, success message was given.

#3 Select one parent and remove

Input: Any parent.

Expected: The parent is removed, success message is given.

Actual: The parent was removed, success message was given.

#4 Select multiple profiles for removal

Input: More than one profile and submit.

Expected: The profiles are all removed, success message is given.

Actual: The profiles were all removed, success message was given.

#5 Select own profile for removal

Input: Own profile.

Expected: Nothing is removed, error message is given to specify that you can not delete your own profile.

Actual: Something was removed, no error message given.

There was a problem with test case #5 because it does not make a lot of sense that you can delete your own profile. As such this was fixed by removing own profile from the table of possible profiles you could delete and even if you bypass the table and send your own id anyway the action `removeProfileForm` will give you an error.

Note that a feature of HTML 5.0 is that it will do form validation automatically for example requiring a field be filled out, which is the case for #6. Browsers not supporting the HTML 5.0 standard will display an error saying that the name may not be empty. We find no bugs or unexpected behaviour in the create profile functionality.

B.2.6 Add Relation Between Profiles

Citizens have parents and guardians which have to be related correctly. We do tests to check if we can create these relations between citizens and their parents and their guardians, one relation at a time or more, and if citizens can have more than two parents.

#1 Test if a parent can have a relation to a citizen

Input: Parent Tulle $\rightarrow$ Child Tulle.

Expected: The relation is accepted and inserted into the database.

Actual: The relation is accepted and inserted into the database.

#2 Test if guardians can have a relation to a citizen and if a citizen can have a relation to multiple other profiles, both parents and guardians

Input: Guardian Tulle $\rightarrow$ Child Tulle.

Expected: The relation is accepted and inserted into the database.

Actual: The relation is accepted and inserted into the database.

#3 Test if already existing relation can be added again

Input: Guardian Tulle $\rightarrow$ Child Tulle.

Expected: The relation is rejected since it already exist.

Actual: The relation is accepted but not inserted into the database.

#4 Test if multiple relations can be added at the same time

Input: Guardian Tulle → Five different citizens

Expected: The relations is created and added in the database.

Actual: The relations is accepted and added in the database

We encounter an unexpected behavior in the add relation functionality. When an already existing relation is prompted to be inserted into the database it is not, which is correct behavior, however a message is displayed to the user stating that the relation was added. The message will be changed to tell the user that the relation already exist.

B.2.7 Remove Relation Between Profiles

When relations is in the database it should be possible to remove them again. A citizen might be assigned to a new guardian and the old relation needs to be removed. A list is displayed with parents and guardians, to the right of those the citizens which they have relations to is listed, and a check box is showed next to the citizens. To remove one or more relations the check boxes is marked. We create six test cases:

#1 Test if a relation between a parent and a citizen can be removed

Input: Parent Tulle → Child Tulle

Expected: The relation is deleted from the database.

Actual: The relation is deleted from the database.

#2 Test if a relation between guardian and a citizen can be removed

Input: Guardian Tulle → Child Tulle

Expected: The relation is deleted from the database.

Actual: The relation is deleted from the database.

#3 Test if multiple relations can be removed from multiple parents and guardians

Input: Guardian profile → Child profile, Guardian profile → Child profile2, Guardian profile2 → Child profile2

Expected: The relations are deleted from the database.

Actual: The relations are deleted from the database.

#4 Test if all relations can be removed entirely

Input: All guardians and parents and their associated citizens.

Expected: The relations are deleted from the database.

Actual: The relations are deleted from the database.

#5 Test if success messages are shown when relations are removed

Input: Guardian/Parent profile → Child profile.

Expected: Success message is given.

Actual: Success message is given.

#6 Test if error messages are shown when no relations are selected and removal is attempted

Input: None.

Expected: Error message is given.

Actual: Error message is given.

We find no unexpected behavior in the remove relation functionality.

B.2.8 User Model

The user model will be tested on a per function basis, because it is modeled as a user with associated functions. The functions to be tested are:

#1 authenticateUser(username, password)

#2 certificateExist()

#3 userExist()

#4 loadUserById(userId)

#5 save()

#6 create()

#7 delete()

What will be conducted is a black-box unit test on each of the above mentioned functions. Although with a limited test input set because these functions most often returns a boolean answer. The input, expected output and actual output will be listed below:

#1 authenticateUser(username, password)

Input: TestUser, 123456

Expected: {id : 1, username : "TestUser", success: true}

Actual: {id : 1, username : "TestUser", success: 1}

#2 certificateExist()

Input: none, but private variable certificate should be set.

Expected: true.

Actual: 1.

#3 userExist()

Input: none, but private variable `userId` should be set.

Expected: true.

Actual: 1.

#4 loadUserById(userId)

Input: 1

Expected: none, but private variables should be set {id : 1, username : TestUser, password : 123456, certificate : testcertificate }

Actual: {id : 1, username : TestUser, password : 8d969eef6ecad3c29a3a629280e686c..., certificate : testcertificate }

#5 save()

Input: {username : TestUsers, password : 1234567, certificate : TestCertificates}, note that this information has changes

Expected: The value of the private variables is updated to the database.

Actual: {username : TestUsers, password : 1234567, certificate : testcertificate}

#6 create()

Input: {username : NewTestUser, password : newpass, certificate : NewTestCertificate}

Expected: The value of the private variables is inserted into the database.

Actual: {username : NewTestUser, password : newpass, certificate : NewTestCertificate}

#7 delete()

Input: none.

Expected: The value of the private variables is deleted from the database.

Actual: Nothing happens in the database.

The user model does not behave as expected in three test cases, namely #5, #6 and #7. Test case #5 does not behave as expected because the output fails in two point, the certificate is not updated and the password loses its encryption. Test case #6 has similar problems with the password, because it is not encrypted before inserted into the database. Test case #7 fails completely and actually does nothing. These errors are now corrected and all test cases behave as expected.

B.2.9 Profile Model

The profile model will be tested on a per function or method basis, because it is modelled as a profile with associated functions. The class `ProfileModel` contains getters and setters for the private variables, but these will not be tested since they are considered so simple that they do not need testing. The functions with further functionality to be tested are:

#1 loadProfileById(profileId)

#2 loadProfileByUserId(userId)

#3 loadProfiles(profileType)

#4 loadProfilesByDepartmentId(departmentId, profileType)

#5 setGuardian(guardianId)

#6 setChild(childId)

#7 deleteGuardian(guardianId)

#8 deleteChild(childId)

#9 save()

#10 create()

#11 delete()

What will be conducted is a black-box unit test on each of the above mentioned functions. The input, expected output and actual output will be listed below:

#1 loadProfileById(profileId)

Input: 118
Expected: {id: 118, name : "Test", role : "Child", departmentId : 1, author : 9}
Actual: {id: 118, name : "Test", role : "Child", departmentId : 1, author : 9}

#2 loadProfileByUserId(userId)

Input: 9
Expected: {id: 73, name : "Guardian John", role : "Guardian", departmentId : 1, author : NULL}
Actual: {id : 73, name : "Guardian John", role : "Guardian", departmentId : 1, author : NULL}

#3 loadProfiles(profileType)

Input: All
Expected: A list of all profiles.
Actual: All profiles are loaded.
Input: Citizens
Expected: A list of all citizens.
Actual: All citizens are loaded.
Input: Guardians
Expected: A list of all guardians.
Actual: All guardians are loaded.
Input: Parents
Expected: A list of all parents.
Actual: All parents are loaded.
Input: GuardiansParents
Expected: A list of all parents and guardians.
Actual: All parents and guardians are loaded.

#4 loadProfilesByDepartmentId(departmentId, profileType)

Input: 1, All
Expected: A list of all profiles under department 1.
Actual: All profiles for department 1 are loaded.
Input: 1, Citizens
Expected: A list of all citizens under department 1.
Actual: All citizens for department 1 are loaded.
Input: 1, Guardians
Expected: A list of all guardians under department 1.
Actual: All guardians for department 1 are loaded.
Input: 1, Parents
Expected: A list of all parents under department 1.
Actual: All parents under department 1 are loaded.
Input: 1, GuardiansParents
Expected: A list of all parents and guardians under department 1.
Actual: All parents and guardians under department 1 are loaded.

#5 setGuardian(guardianId)

Input: 73, load Test(118).
Expected: Guardian John(73) becomes guardian of Test(118).
Actual: Guardian John(73) becomes guardian of Test(118).

#6 setChild(childId)

Input: 118, load Guardian John(73).
Expected: Guardian John(73) will have Test(118) as citizen.
Actual: Guardian John(73) have Test(118) as citizen.

#7 deleteGuardian(guardianId)

Input: 73, load Test(118).
Expected: Loaded profile Test(118) will not have Guardian John(73) as guardian any more.
Actual: The relation between Guardian John(73) and Test(118) does not exists any more.

#8 deleteChild(childId)

Input: 118, load Guardian John(73).
Expected: Loaded profile John(73) will no longer have Test(118) as citizen.
Actual: Guardian John(73) does no longer have Test as citizen.

#9 save()

Input: Load Test and change the following {name : Tests, role : Guardian, departmentId : 3, author : 37}.

Expected: The data for Test is updated to the values of the input.

Actual: The data for Test is updated to the input values, except the author which remains unchanged.

#10 create()

Input: Set the attributes as follows {name : Testing, role : Admin, departmentId : 3}.

Expected: A new profile is created with the attributes not mentioned above set to null, and the others set to what is specified in the input.

Actual: {id : 126, name : Testing, role : Admin, departmentId : 3}, with all other attributes set to NULL except the timestamp which is maintained by the database.

#11 delete()

Input: Load the profile Testing(126) created above.

Expected: The loaded profile will be deleted from the database.

Actual: The loaded profile is removed from the database.

The tests reveals unexpected behaviour for test case #9 where the author of the profile does not change in the database. This behaviour can be both correct and incorrect depending on the purpose of the author attribute. If the author attribute is there to be able to argue about the profiles original creator/author, the behaviour is correct. But if the purpose of author is something else, such as last edited by, the behaviour is incorrect. The attribute does not serve a purpose in this project during this sprint and we will therefore not change the behaviour of the save function.

B.3 Sprint #3

B.3.1 Last Years Usability Test

The following is usability problems found by the last years group which was listed in their report[1].

Description	Severity	Corrected
Pictograms - It is not possible to create new categories of pictograms	Critical	
Profile - Restrict QR editing to department manager	Critical	
Database Problem - Not possible for all pedagogues to fix pictograms for all department citizens	Critical	
Navigation - The site "Profiles" should link to each profile	Critical	
Missing - Unable to remove relations	Critical	
Profile Picture - Accept button is hard to find	Serious	
Profile Picture - Word "change" is misleading	Serious	
Navigation - "Add" and "Make" under pictogram manager is confusing	Serious	
Navigation - Language support on navigation did not change to Danish	Serious	X
Missing - No Danish language support on the site "Profiles"	Serious	X
Profile - Department should be a link, not editable	Serious	(X)
Profile - Links from "Own Profile" relations is missing	Serious	
Profile Picture - GIRAF logo as placeholder is misleading	Cosmetic	X
Profile Picture - Word "Edit" is not informative	Cosmetic	
Database Problem - "Own Profile" takes too long to load	Cosmetic	
Navigation - "Add Relation" is before "Create Profile"	Cosmetic	X
Design - Standardise button names	Cosmetic	
Logout - Can navigate in system without sesson	Cosmetic	

Table B.1: Usability problems found by last years group.

B.3.2 Own Department

This pages shows information about the department of the logged in user, including the name, address, phone number, e-mail and super department. Further more information about which guardians, parents and citizens is associated with that department. There are several links to profile pages, and functionality to change the given information about the department, that needs to be tested.

When Logged In As a Guardian (Eva Kvist)

The following cases is for when logged in as a parent, and it is divided into two parts, the content of the page, and editing the information on the page.

Content on the page

#7 Information

Expected: All contact information about the department should be shown.

Actual: All contact information about the department is shown.

#8 Værger(Guardians)

Expected: All guardians and parents of that department should be shown.

Actual: All guardians and parents of that department is shown.

#9 Borgere(Citizens)

Expected: All citizens of that department.

Actual: All citizens of that department.

Edit information

#10 Department name

Expected: The information should be editable and updated to the database.

Actual: The information was edited and updated in the database.

#11 Address

Expected: The address should be editable and updated to the database.

Actual The address was edited and updated to the database. (It is the responsibility of the user to enter a valid address)

#12 Phone number

Expected: The phone number should be editable and updated to the database.

Actual: The phone number was changed and updated in the database. (It is the responsibility of the user to enter a valid phone number)

#13 E-mail

Expected: The e-mail should be editable and updated to the database.

Actual: The e-mail was edited and updated to the database. (It is again the responsibility of the user to enter a valid e-mail)

#14 Super department

Expected: The super department should not be editable.

Actual: The super department was edited through a drop-down list to another department and updated in the database.

Guardians should not be able to edit the super department, only Administrators should be able to do that. Furthermore there is no validation checks on the edited information meaning that a string can be entered as phone number and an invalid e-mail can be entered. The errors found in the test was be corrected.

B.3.3 All Departments

This page is an administrator only page, and will show all departments of the system. Each listed department will have a link to the department. Furthermore a search function is implemented on the page. The search will not be tested here, but in the Section B.3.6. The expected content of the page will be a list of all departments in the whole system. The actual content of the page is exactly that and the page does therefore pass the test.

B.3.4 Create Department

This page provides functionality to create a new department, which is also an administrator only feature. There are several input fields including name, phone number, e-mail, address, city, zip code, and super department. A set of test cases with input, expected output and actual output, each addressing different possible errors of the feature is given.

#1 Create a valid department

Input: {name : depName, phone number : 98989898, e-mail : dep@department.dk, address : Selma Lagerlöfsvej 300, city : Aalborg Øst, zip code : 9220, super department : Aalborg Universitet}

Expected: Information gets inserted into the database.

Actual: A new row is inserted into the database containing the information above. Response message: "depName er oprettet.".

#2 Check for phone number validation(too long)

Input: Valid information from case #1 but with the phone number changed to 989898989898 instead.

Expected: Respond with a message saying that the phone number is invalid.

Actual: A new row is inserted into the database. Response message : "depName er oprettet.".

#3 Check for phone number validation(containing letters)

Input: Valid information from case #1 but with the phone number changed to FF88BB88 instead.

Expected: An error message will be displayed saying that it is not a valid phone number.

Actual: A new row is inserted into the database. Response message : "depName er oprettet.".

#4 Check for e-mail validation(missing part of domain)

Input: Valid information from case #1 but with the e-mail changed to dep@department instead.

Expected: An error message will be displayed saying that the e-mail is invalid.

Actual: An error message was shown : "Følgende fejl opstod: Email feltet er ugyldigt.". No row is inserted into the database.

#5 Check for e-mail validation(missing @)

Input: Valid information from case #1 but with the e-mail changed to depdepartment.dk instead.

Expected: An error message will be displayed saying that the e-mail is invalid.

Actual: An error message was shown : "Følgende fejl opstod: Email feltet er ugyldigt.". No row is inserted into the database.

#6 Check for e-mail validation(missing domain)

Input: Valid information from case #1 but with the e-mail changed to depdepartment.dk instead.

Expected: An error message will be displayed saying that the e-mail is invalid.

Actual: An error message was shown : "Følgende fejl opstod: Email feltet er ugyldigt.". No row is inserted into the database.

#7 Check for city validation(including numbers)

Input: Valid information from case #1 but with the city changed to Aalborg Øst12 instead.

Expected: An error message will be displayed saying that the city is invalid.

Actual: A new row is inserted into the database. Response message : "depName er oprettet.".

#8 Check for city validation(include special characters)

Input: Valid information from case #1 but with the city changed to !Aalborg@Øst$ instead.

Expected: An error message will be displayed saying that the city is invalid.

Actual: A new row is inserted into the database. Response message : "depName er oprettet.".

#9 Check for zip code validation(too many numbers)

Input: Valid information from case #1 but with the zip code changed to 92208 instead.

Expected: An error message will be displayed saying that the zip code is invalid.

Actual: A new row is inserted into the database. Response message : "depName er oprettet.".

#10 Check for zip code validation(including letters)

Input: Valid information from case #1 but with the zip code changed to abcd instead.

Expected: An error message will be displayed saying that the zip code is invalid.

Actual: A new row is inserted into the database. Response message : "depName er oprettet.".

#11 Check for matching zip code and city

Input: Valid information from case #1 but with the zip code changed to 9000 (not for Aalborg Øst) instead.

Expected: An error message will be displayed saying that the zip code and city does not match.

Actual: A new row is inserted into the database. Response message : "depName er oprettet.".

The following test cases does not output as expected: #2, #3, #7, #8, #9, #10, #11. The only thing there seems to be a validation check on is the e-mail. The errors discovered during test has been corrected.

B.3.5 Remove Department

This is an administrator only feature to remove a department from the system. Each department name links to its own overview page, and has a check box that can be checked indicating that the department is to be removed from the system. What can be tested for is the removal of a single department and multiple departments and to check that the database is updated accordingly. The test of both single and multiple departments behaved as expected and thus the feature works correctly.

B.3.6 Search

The search functionality is used to search for profiles, users, and departments on some of the pages. It searches in either profile names or department names depending on the page where it is used.

Search in "All Departments"

#1 Search for "Universitet"

Expected: Show "Aalborg Universitet" and "Aarhus Universitet".

Actual: Shows "Aalborg Universitet" and "Aarhus Universitet".

#2 Search for "Aa"

Expected: "Aalborg Kommune", "Aalborg Universitet" and "Aarhus Universitet".

Actual: "Aalborg Kommune", "Aalborg Universitet" and "Aarhus Universitet".

#3 Search for "b"

Expected: "Aalborg Kommune", "Aalborg Universitet" and "Birken".

Actual: "Aalborg Kommune", "Aalborg Universitet" and "Birken".

#4 Search for "aaa"

Expected: Empty table.

Actual: Empty table.

Search in "All Relations"

#5 Search for "Hansen"

Expected: Show "Anders Hansen", "Tom Hansen" and all of their relations.

Actual: Shows "Anders Hansen", "Tom Hansen" and all of their relations.

#6 Search for "im"

Expected: "Kim Brun", "Mette Neiman" and all of their relations.

Actual: "Kim Brun", "Mette Neiman" and all of their relations.

B.3.7 Applications

It is possible to assign applications to profiles as well as copying applications from one profile to another. These are the tests perform on the two pages.

Assign Application

#1 Add "Piktotegner" to Clara Verner

Expected: "Piktotegner" should be added to Clara Verner

Actual: "Piktotegner" is added to Clara Verner

#2 Add "Cars" to both Clara Verner and Emil Gajhede

Expected: "Cars" should be added to Emil Gajhede. Nothing should be added to Clara Verner since she already has that applications

Actual: "Cars" was added to Emil Gajhede and nothing to Clara Verner.

#3 Add "Cars" and "Piktotegner" both Emil Gajhede and Emma Højgård

Expected: Both applications should be added to Emma Højgård, but only "Piktotegner" should be added to Emil Gajhede, since he already has "Cars".

Actual: "Piktotegner" was added to Emil Gajhede and both applications was added to Emma Højgård.

Copy Application

#4 Copy from Freja Hansen to Frederik Nielsen causing Frederik Nielsen to have both applications

Expected: Frederik Nielsen should have both applications.

Actual: Frederik Nielsen has both applications.

#5 Copy from Frederik Nielsen to Hanim Abdulla causing Hanim Abdulla to have both applications

Expected: Hanim Abdulla should have both applications.

Actual: Hanim Abdulla has both applications.

#6 Copy from Hanim Abdulla to Frederik Nielsen. Both already have both applications

Expected: Nothing should happen.

Actual: Nothing happened.

B.3.8 Create Pictogram

The Create Pictogram page is where you can upload new pictogram and set relevant information for it.

Upload Features

#1 Try to upload nothing

Expected: Show a notice on the page, saying that the image field can not be empty.

Actual: Shows a notice on the page, saying that the field can not be empty. If you try to trick the server by sending the form with the rest of the required information it gives a filetype error.

#2 Try to upload a pictogram without all the required information

Expected: Show a notice on the page, saying that the name field can not be empty.

Actual: Shows a notice on the page, saying that the field can not be empty. If you try to trick the server by sending the form with the rest of the required information it gives a required fields error.

#3 Try to upload a pictogram setting all the required information

Expected: Show a notice on the page saying that the pictogram was uploaded. Add pictogram to database.

Actual: Show a notice on the page saying that the pictogram was uploaded. Added pictogram to database.

#4 Try to upload a pictogram setting all the required information, along with the inline text field and the sound field

Expected: Show a notice on the page saying that the pictogram was uploaded. Add pictogram and sound to the database.

Actual: Show a notice on the page saying that the pictogram was uploaded. Added pictogram and sound to database.

B.3.9 Create User

Creating a user to the system is both a task of a guardian and the administrator. The creation of a new user requires several input fields to be filled out, namely user name, password, whether to create a new profile along with the user, and in that case information such as profile name and type (Guardian or Parent). Several test cases can be set up checking for not repeating the password correctly, and various manipulations of the HTML selection list.

#1 Check for acceptance of assumed valid user with new profile

Input: { username : Test, password : 123456, repeat password : 123456, newProfile : new, profile type : guardian, profile name : Tester}.

Expected: A new user and a new profile is inserted into the database with the information inserted above.

Actual: The data is inserted into the database with response message : "Brugeren er oprettet.". The profile is created under the same department as the logged in user creating the profile.

#2 Check for acceptance of assumed valid user with existing profile

Input: { username : Test, password : 123456, repeat password : 123456, newProfile : existing(a)}.

Expected: A new user is inserted into the database with the information given above and with a relation to user 'a'.

Actual: The data is inserted into the database with response message : "Brugeren er oprettet.". The profile is created under the same department as the logged in user creating the profile and with relation to user 'a'. Noting that only profiles without a relation under that department is listed in the list.

#3 Check for different repeated password

Input: As case #1 but with repeat password set to 1234567.

Expected: An error message will be displayed saying the two passwords is not the same.

Actual: Response message : "De to kodeord er ikke identiske.". No data is inserted into the database.

#4 Omitting information about the profile

Input: As case #1 but omitting the information about the profile.

Expected: An error message will be displayed saying that the profile information have to be given.

Actual: Response message : "Navnet må ikke være tomt.". No data is inserted into the database.

#5 Manipulate the list of profiles to include a random selected id

Input: As case #2 but editing the id of af user to something else say 172.

Expected: An error message should be displayed saying that the profile is not in the list.

Actual: Response message : "Brugeren er oprettet.". The user with id 172 gets a relation to the user (accidentally a citizen).

#6 Manipulate the profiles type list to include a random selected number (0 for admin)

Input: As case #1 but editing profile type to 0.

Expected: An error message should be displayed saying that it has to be a guardian or parent.

Actual: Response message : "Profil typen er ikke gyldig.".

Test case #5 fails because a citizen could get a user, which should not be possible. This error has been corrected and the feature passes all test cases.

B.3.10 Remove User

The Remove User page is where you can remove users form the system.

#1 Try to remove 0 users

Expected: Show an error message on the page, saying that no users were selected.

Actual: Shows an error message on the page saying that no users were selected.

#2 Try to remove 1+ user(s)

Expected: Show a notice on the page saying the user(s) was removed, and remove the user(s) from the database.

Actual: Shows a notice on the page saying the user(s) was removed, and removed the user(s) from the database.

#3 Try to remove user in different department as a Guardian

Expected: Should be impossible, as you can not even see the user in the list.

Actual: If you manipulate the data, to a different user in a different department it gives an error saying that you can not remove users in different departments.

#4 Try to remove user in different department as an Admin

Expected: Show a notice on the page saying the user was removed, and remove the user from the database.

Actual: Shows a notice on the page saying the user was removed, and removed the user from the database.

#5 Try to remove your own user

Expected: Should be impossible, as you can not even see the user in the list.

Actual: If you manipulate the data, to point it to your user, it gives an error saying you can not remove your own user.

#6 Try to remove user that does not exist

Expected: Show an error message on the page saying that no valid users were selected.

Actual: Shows an error message on that page saying that no valid users were selected.

B.4 Sprint #4

B.4.1 Search through pictograms

The search through pictograms feature allows a user to search for some text string in the name or in tags. It can also restrict searching to the department, the pictograms the current user's profile has uploaded, and all.

General

#1 Searching for nothing

Expected: Show a message asking for the text search field to be filled out.

Actual: Shows a message asking for the text search field to be filled out.

#1 Searching for nothing, subverting the required attribute

Expected: Redirect to home.

Actual: Redirects to home.

Searching For Pictograms

#3 Search using name in own department or uploaded by users in own department

Expected: Show a list of pictograms with a name that matches search, from own department or uploaded by users in own department.

Actual: Shows a list of pictograms from own department or uploaded by users in own department matching search.

#4 Search using name in current user's uploaded pictograms

Expected: Show a list of pictograms with a name that matches search, from current user's uploaded pictograms.

Actual: Shows a list of pictograms uploaded by current profile matching search.

#5 Search using name in all pictograms

Expected: Show a list of pictograms with a name that matches search, from all pictograms.

Actual: Shows a list of all pictograms matching search.

#6 Search using tags in own department or uploaded by users in own department

Expected: Show a list of pictograms with tags that matches search, from own department.

Actual: Shows a list of pictograms that have matching tags, from own department or uploaded by users in own department.

#7 Search using tags in current user's uploaded pictograms

Expected: Show a list of pictograms with tags that matches search, from current profile's uploaded pictograms.

Actual: Is broken, does not search correctly.

#8 Search using tags in all pictograms

Expected: Show a list of pictograms with tags that matches search, from all pictograms.

Actual: Shows a list of pictograms that have matching tags, from all.

Searching using tags in current user's uploaded pictogram was broken and was fixed.

B.4.2 Crop profile pictures to meet a 8:10 ratio

The Launcher group set a requirement that all profile pictures should be in a 8:10 ratio, as such this was implemented. This documents the tests.

Cropping properly on both the home page and the view profile page

#1 Cropping a very big image(2891x2196) by selecting an image area and accept

Expected: Crop the image to the selected area and resize it to at maximum 160 pixels wide and 200 pixel high.

Actual: Crops the image to the selected area and resizes it to at maximum 160 pixels wide and 200 pixel high.

#2 Cropping a very narrow image(1x500) by selecting an image area and accept

Expected: Should be impossible because of the narrowness of the image, should give an error.

Actual: Is impossible, gives an error.

#3 Cropping a very short image(500x1) by selecting an image area and accept

Expected: Should be impossible because of the narrowness of the image, should give an error.

Actual: Is impossible, givesan error.

Not cropping on purpose on both the home page and the view profile page

#4 Uploading an image that can be cropped to 8:10 and not cropping it

Expected: Should give an error.

Actual: Gives an error.

#5 Uploading an image that can not be cropped to 8:10(such as a 500x1 image) and not cropping it

Expected: Should give an error.

Actual: Gives an error.

B.4.3 Assign Pictograms to Citizens

Assign pictogram is a page where guardians and parents can assign pictograms to citizens. The pictograms are shown on the left side of the screen and a list of citizens is shown on the right side of the screen. The user can set a check mark on the pictograms and the citizens which they want to assign them to, this is a many to many relationship. An important part of the feature is the distinguishing between the profile that is logged in, whether it is a guardian or an administrator. The first part of the feature consist of a search dialog, see Section B.4.1 for conducted test of that feature. The feature will be tested both when logged in as guardian and as administrator to see whether there is a different in the result shown.

As Guardian

#1 The pictogram list

Expected: All relevant pictograms will be shown.

Actual: All relevant pictograms was shown.

#2 The citizen list

Expected: Only citizens of the logged in users department.

Actual: Only the citizens of the department was shown.

#3 Assign a single pictogram to a single profile

Expected: The id of the profile and the id of the pictogram will be set into the relation profile_-pictogram in the database and a message confirming the input is displayed.

Actual: The id of the profile and the id of the pictogram will be set into the relation profile_-pictogram in the database and a message confirming the input is displayed.

#4 Assign a single pictogram to multiple profiles

Expected: The id of the profiles and the ids of the pictogram will be set into the relation profile_-pictogram in the database one row for each pictogram to profile and a message confirming the input is displayed.

Actual: The id of the profiles and the ids of the pictogram will be set into the relation profile_-pictogram in the database one row for each pictogram to profile and a message confirming the input is displayed.

#5 Assign multiple pictograms to a single profile

Expected: The id of the profiles and the ids of the pictogram will be set into the relation profile_-pictogram in the database one row for each pictogram to profile and a message confirming the input is displayed.

Actual: The id of the profiles and the ids of the pictogram will be set into the relation profile_-pictogram in the database one row for each pictogram to profile and a message confirming the input is displayed.

#6 Assign multiple pictogram to multiple profile

Expected: The id of the profiles and the ids of the pictogram will be set into the relation profile_-pictogram in the database one row for each pictogram to profile and a message confirming the input is displayed.

Actual: The id of the profiles and the ids of the pictogram will be set into the relation profile_-pictogram in the database one row for each pictogram to profile and a message confirming the input is displayed.

#7 Assign a single pictogram to no profiles

Expected: No input will be made in the database and an error message will be shown.

Actual: No input will be made in the database and an error message will be shown.

#8 Assign multiple pictograms to no profiles

Expected: No input will be made in the database and an error message will be shown.

Actual: No input will be made in the database and an error message will be shown.

#9 Assign no pictogram to a single profile

Expected: No input will be made in the database and an error message will be shown.

Actual: No input will be made in the database and an error message will be shown.

#10 Assign no pictogram to multiple profiles

Expected: No input will be made in the database and an error message will be shown.

Actual: No input will be made in the database and an error message will be shown.

#11 Assign a pictogram to a citizen outside department (edit HTML values)

Expected: Error message saying that only citizens in your department can be chosen

Actual: A reponse message saying: "Du valgte en eller flere borgere som ikke er lovlige.". Illegal citizen(s).

#12 Assign something non-existing either citizen or pictogram

Expected: Error message saying that either the pictogram did not exist or that the citizen was not in your department.

Actual: A reponse message saying: "Piktogram(mer) tildelt". The database is not updated. What if the id was a private pictogram to someone?

#13 Assign the id of a private pitogram to a citizen

Expected: Error message saying that pictogram cannot be assigned

Actual: A reponse message saying: "Piktogram(mer) tildelt". The database is updated with the private pictogram.

A lot of the test cases performed when logged in as a guardian can be copied and applied to the test conducted when logged in as administrator, but these will not be copied to keep the listing short.

As Administrator

#14 The pictogram list

Expected: All relevant pictograms will be shown.

Actual: All relevant pictograms was shown.

#15 The citizen list

Expected: All citizens plus their department name.

Actual: All citizens including their department was shown.

#16 Assign a single pictogram across departments

Expected: The pictogram gets assigned to the chosen citizens.

Actual: A reponse message saying: "Du valgte en eller flere borgere som ikke er lovlige." Illegal citizen(s).

#17 Assign multiple pictograms across departments

Expected: The pictograms get assigned to the chosen citizens.

Actual: A reponse message saying: "Du valgte en eller flere borgere som ikke er lovlige." Illegal citizen(s).

#18 Assign something non-existing either citizen or pictogram

Expected: Error message saying that the pictogram did not exist or the profile did not exist

Actual: A reponse message saying: "Piktogram(mer) tildelt". The database is not updated.

The errors discovered during test case #12, #13, #16, #17, and #18 was corrected.

C Changelog

This appendix chapter describes in non-technical terms what was implemented during Sprints #1-4.

C.1 Sprint #1

Worked on Timer application.

- Removed profile list.
- Removed long click gestures entirely from the application, such as deletion of timers is now done with a push of a button instead.
- Removed the "Gem Som" button. Saved using the "Gem" button instead.
- Changed the gradient button.
- Changed time representation from m:s to mm:ss.
- Changed so that a citizen can no longer attach another citizen's timer.
- Changed toasts from using constant, different, duration to a single toast timer variable.
- Changed so the application can only be started through the launcher.
- Changed the names of the timers to a more expressive names.
- Changed the default colours to black, grey, and white.
- Changed so that the user does not end at the login screen after a timer has finished.
- Added a button to change the button orientation from horizontal to vertical.
- Added the current citizen's name to the header of the application.
- Added some missing toasts.
- Fixed so the application only shows one icon in the application menu instead of three.
- Fixed so the timer can not be started with 0 minutes and 0 seconds.
- Optimised the layout to also fit a 7" tablet.

C.2 Sprint #2

Worked on GIRAF Web Admin.

- Remade visual design with heavy inspiration from previous GIRAF Web Admin project
- Added login functionality
- Added logout functionality
- Added own profile page where you can view your information
- Added functionality to change password for your user.
- Added create profile page where you can make new profiles.
- Added remove profile page where you can remove profiles.
- Added add relation page where you can create relations between guardians/parents and citizens.
- Added remove relation page where you can remove relations between guardians/parents and citizens.

C.3 Sprint #3

Worked on GIRAF Web Admin.

- Updated visual design.
- Added pages for pictogram.
- Added pages for applications.
- Added pages for departments.
- Added view profile pages.
- Made profiles more editable.
- Added Search functions.
- Added listing pages for profiles, relations, departments and applications.
- Fixed glossary of words.
- Restrict various edit and view operations.

C.4 Sprint #4

Worked on GIRAF Web Admin.

- Search through pictograms
- Assign pictograms to citizens
- Edit pictogram inline-text individually for each citizen
- Remove assignment of applications to citizens
- Crop profile pictures to meet a 8:10 ratio
- Show department for citizens on various pages when logged in as Administrator

D Implementation

This appendix will describe various features that were implemented during sprints, but were deemed not important enough to include in the main report. All of the features of Sprint #1 is places in the report.

D.1 Sprint #2

During this sprint a lot of features that were not particularly important to the end product, but still needed, were implemented. This includes features such as:

- Log In
- Log Out
- Change Password
- Remove Relation

D.1.1 Log In

The Log In page is just a feature that allows a specific user to log into the system with his username and password. As such it is not particularly important in that it does not provide any obvious functionality, other than giving access to the system.

The way the Log In page is implemented is first to instantiate a `UserModel`, as this will hold the behaviour for communicating with the database and thus the behaviour for authenticating a username-password combination. The method for authenticating a user is called `authenticateUser`.
What it actually does is as follows:

1. Hashes the password with SHA256.
2. Looks in the database for a match with the specified username and hashed password.
3. If there is one result, it fetches the id and username from the database query and puts it into an array along with setting a value called success to true in the array.
4. If there is more than one result or no result, it sets success to false in the array.
5. It then returns the array to be used elsewhere.

The next step was creating a controller for the login page, along with the needed views. This controller is called `Login`. The controller defines 3 separate behaviours for different actions. The important action for logging in has been called `authenticate`. The `authenticate` action receives data and validates it with a call to the `authenticateUser` method which will either return a successful or unsuccessful authentication. If the authentication was successful it will set various SESSION variables such as the `username`, the `userId`, the `userRole` and it will

redirect to the Home page. If the authentication was not successful it will set an error message in the SESSION and redirect back to the login page where that message will be shown.

To simplify whether or not the user is authenticated, we also define a static function called `isAuthenticated` that will check that the appropriate SESSION variables are set and are true. This is then used to redirect to the Log In page for anyone not authenticated if there is an attempt to visit any page but the Log In page, and to the Home page for anyone authenticated if there is an attempt to visit the Log In page.

D.1.2 Log Out

The Log Out is much simpler than the Log In page, as it simply just needs an action defined in the controller, which we will call `logout` that will check whether or not the user is authenticated and if the user is, it will destroy the current session and redirect to the Log In page.

D.1.3 Change Password

The Change Password functionality implements a form in the `index` view for the `Home` controller, that sends data about password modifications. The action this form uses is called `editUserSavePassword`. This action gets the data being the old password and two new passwords. The action then tests for if the two new passwords are identical and if the old password matches the one in the database for the currently logged in user. If all those checks pass, it uses a `UserModel` to set the new password and save it to the database and then it sets a success message. If any of the checks fail, it will set an error message that tells the user what they did incorrectly. In both instances it redirects and shows either the success message or the error message to the user, depending on what happened.

D.1.4 Remove Relation

In Figure D.1 the GUI for the Remove Relation page can be seen. In the left column is the parents and guardians listed with their related citizens in the center column. There is a check-box in the right column for each citizen. To remove relations the user simply marks the citizens to be removed from the guardian or parent. What this generates is an array with elements of the form "x|y" where x is the parent or guardian id and y is the citizen id. In the `removeRelationForm` action this array is iterated upon and for each guardian or parent their selected citizens is removed.

Figure D.1: GUI for Removing Relations page

D.2 Sprint #3

This appendix section covers various features implemented in Sprint #3 that were not included in the main report.

D.2.1 Restrictions

During this sprint restrictions was implemented. The restrictions is used to restrict certain pages to guardians and others to administrators. This was done using five static functions `requireAdmin`, `requireGuardian`, `requireGuardianAndNotAdmin`, `isAdmin` and `isGuardian`. The first three functions are only used in the controllers to restrict the access to the page. The last two are used in both the HTML, to hide or show elements that should or should not be shown, and in the controllers to give specific data to for example administrators, if the page is accessible to both guardians and administrators but with slightly different content. All five functions use the session `$_SESSION["userRole"]` which is set at login and contains the role of the current user. If a user attempts to access a site which they do not have access to, the `requireAdmin`, `requireGuardian` and `requireGuardianAndNotAdmin` functions redirect them to the front page.

D.2.2 Profile Pages

This subsection describes implementation of various profile pages, such as All Profiles, All Relations and View Profile.

All Profiles

The All Profiles page consists of a table that either show all profiles within a department or only specific profiles in the department. A logged in admin profile will be able to see all profiles within all departments while a parent and a guardian will see the citizens, parents

and guardians. The `Profile` controller creates a `ProfileModel` using the logged in profile's id and calls the `loadProfilesByDepartmentId(depId, profileType, search)` function on the `ProfileModel`. The `depId` is the department that the logged in profile is associated with, the `profileType` is a number from zero to five and choose the type of profiles which is loaded from the database, the `search` is for the jQuery search described in Section 5.2.2. The profiles loaded from the database are saved in an array which the All Profiles page uses to display the table of profiles.

All Relations

The All Relations page contains a table which show parents and guardians on the left and the citizens they are related to the right of them. This means that citizens can be shown multiple times in the table, e.g(to the right of both their parents and the guardian that is taking care of them in their department). The `Profile` controller creates a `ProfileModel` for the logged in user and calls the `loadProfilesByDepartmentId(depId, profileType, search)` function in the model. These profiles are returned to the All Relations page which iterate each guardian and parents and load the profile into the table, then for each of them it iterates their related citizens, all these profiles are then shown it the table.

View Profile

The View Profile page consists of two tables and a profile picture. The first table show the name, department and role of the profile, this is from the `ProfileModel` which is loaded from the database with the function `loadProfileById(currentId)`, where the `currentId` is the id of the profile which page the user is visiting. The second table show the name of other profiles related to the profile, it can be parents and guardians to a citizen or citizens to a guardian. The `Profile` controller checks the role of the profile, if it is a citizen it calls `getParents()` and `getGuardians()` which loads the profiles in these relations to the citizen profile from the database. The profile picture is an image loaded from the database related to the profile, it is also possible to change the profile picture by the use of a HTML form.

D.2.3 Application Pages

This subsection documents various application pages that were not documented in the main report, such as All Applications, Copy Applications, Assign Applications.

All Applications

On the All Applications page, all applications are shown. This also include applications that is not assigned to the current department. This is implemented using the `AppModel`'s `loadApps` function. This function is implemented like the equivalent function in the other models by fetching all the relevant ids, and then returning an array of `AppModel` objects, generated by `loadAppById`, with information about each application.

Copy Applications

The HTML for the application copying is made using two nested `foreach` loops for each table. The outer `foreach` loop is iterating though all the citizens with autism from the current department who have applications assigned to them. For each person that persons applications is loaded using `loadAppsByProfileId` which returns an array with all the applications. This array is iterated upon in the second `foreach` loop. The same is happening for the second table,

except for this one, all persons with autism is shown, even though they have no applications assigned.

The copying is done by first loading the "From" profile and fetching the applications for that profile. Then the "To" profile is loaded and all the applications from the first profile is added to the other profile.

Assign Applications

This page changes depending on the role of the current user. Administrators is able to see all the autistic profiles while guardians is only able to see the profiles in their own department. The tables generated on the website is a table for the relevant applications and a list of the relevant autistic profiles. It is possible to select multiple applications together with multiple profiles. This will add all the selected applications to all the selected profiles by iterating on both selections. The submitted information is an array of applications and an array of profiles. A nested `foreach` firstly iterates upon the profiles, and for each of the profiles a second `foreach` iterates through the applications and adding them to the profile.

D.2.4 Pictogram Pages

This subsection covers implementation of Assign Pictogram.

Assign Pictogram

This feature consists of two parts, on the left a selection of pictograms to be assigned to a citizens, and on the right a list of citizens to assign the pictogram to. The functionality of the feature is finished, but the selection of pictograms needs to optimised in terms of showing the pictograms to the user. The feature allows multiple pictograms to be assigned to multiple citizens and thus the implementation uses a nested loop assigning each selected pictogram to each of the selected citizens. The `Assign Pictogram` feature makes use of the `PictogramModel` calling the `assignToProfile` function.

E Customer Contact Questions

The following documents the questions used as guidelines during the interview with the customers.

E.1 Preliminary

- What should parents, social workers, and children be called?
- What do the customers call relations?
- Tell the customers the difference between users and profiles.

E.2 Pictograms

- What are the customers thoughts in regards to pictograms?
  - Do you need to be able to
    - Upload
      - Should you be able to 'edit' the pictograms after upload?
    - Delete
      - What should happen to others that use the pictogram?
    - Assign
    - Amend pictograms
      - Should the amendments happen to all who use the pictograms?
  - Sound
    - Should it be playable through the website?
    - How do the customers imagine it will work?
  - Categories
    - Create
    - Delete
    - Copy
    - Assign
    - Private and publicly available categories
      - Who should be able to see private categories?
      - Public, available only for the department or for all departments?
- Other (possible from the customers?)

E.3 Profiles, Users and Relations

- Who of the pedagogues, parents and so on, should be able to do what?
  - Create users
  - Delete users
  - Create profiles
  - Delete profiles
  - Copy from one profile to another
    - Should pictograms be copied?
    - Should applications be copied?
    - Should parents be copied?
    - and so on
  - Add or remove relations
  - QR Codes
    - Generate new
    - Print
  - Other (potentially from the customers?)

E.4 Applications

- Assign applications to citizens.

E.5 Parent Access

- Should the parents be able to log in to the homepage?
- What should they be able to do?

E.6 Show The Site to The Customer and Talk Loosely About It

- Would drag and drop make it easier?

F To the Next Group

This chapter is meant for the group who may be working on the website next. This should give them an introduction to the structure of the site as well as a list of current problems that could be a starting point for the next group to work on.

F.1 Structure

The design pattern used for the site is MVC(Model-View-Controller) which means that the site is split into three layers. The model, which can be found in */application/model/*, is the classes used for all database related communication, which means that these are the only files in which you will find any SQL. The controllers, which can be found in */application/controller/*, is the connection between the views, found in */application/view/*, and the model. The views contain all the HTML, and is also the only place where HTML can be found. All CSS, JavaScript and images can be found in the */public/* folder. Furthermore the files in the application folder can not be opened directly in the browser, which is also why CSS, JavaScript, and images have its own public folder.

F.1.1 Edit or Create Pages

First of all, the current address for the website(as of spring 2014) is `http://cs-cust06-int.cs.aau.dk/webadmin/`. The structure of the address is

```
http://cs-cust06-int.cs.aau.dk/webadmin/{controller}/{action}/{p1}/{p2}/{p3}/
```

The controller corresponds to the class name of each controller file. The action corresponds to the functions in each controller class. This has the side effect that creating a function in the controller, which is not supposed to be a site on its own, can lead to error messages since making private functions and then navigating to that page in the browser causes a fatal error. Because of this extra functions should be moved elsewhere like in a model. This structure does have its advantages though. It is very easy to find the function you are looking for, if changes needs to be made, and it is easy to create new pages, since it just requires a new function in the corresponding controller. The p's are function parameters.

For each of the controllers a folder in the view folder can be found. This folder contains all the HTML pages needed for the functions in the controller. Note however that not all functions in the controllers have an associated file in the view folder. This happens if the action does not require any HTML, like actions used only for receiving form data, where a redirect is used instead. For each new page containing HTML the `header.php` and `foopter.php` in the `_template` folder should be includes before and after the file in the view folder. These two files contains the navigation etc.

If new CSS or JavaScript is needed, and it is preferred to have in new files, these files can just be added to the css and js folders in the public folder. The files will automatically be loaded on the website.

The models are loaded in the controllers by using
`$someVar = $this->loadModel('ModelName');`. The current models use the same structure for loading, editing, deleting and creating new rows in the database. Loading is done by one of the load functions like
`$someVar->loadProfileById(someId)`. After this the getter and setter functions are used to read or modify the information loaded. If information is changed with one of the setter functions, then the save function can be used to update the database with the new information. In the same way the delete function can be used to delete the row. If a new row has to be inserted, instead of loading an existing row, just use the setter functions followed by the create function.

F.2 Current Problems

Since all code have been tested only few errors should be experienced. Because of this along with the choice of design pattern and because it took us more time than expected to start over again, we **highly recommend** that the page is not remade for the next semester.

Currently the page is able to assign different applications to different profiles but the Android applications do not implemented this currently. So for the next semester this feature could either be removed, or, better, the groups working with the Android applications could be convinced to comply with the information in the database. In particular the group working with Launcher/GIRAF should be convinced, since they seem to be the group who decides which profile is able to use which applications.

Another problem is with the database, which is missing a column in the profile_pictogram table namely an inline text column which should be used to rename a pictogram for a specific user. Again, the functionality is ready on the website, but out commented. The lines to be activated again is in the file *application/view/pictogram/getRemovePictograms.php* where line 7 should be removed and line 5-6 should be activated. The out commented HTML around line 20-22 should also be activated. Because of a new naming convention in the column name, the name of the column should maybe be changed in the pictogrammodel.php around line 267 and 286. A functionality that is missing is the ability to give another sound to the pictogram as well. This should be added to the profile_pictogram table as well. All the functionality to handle the new sound have not been made and this could therefore be a task for the next group.

Currently there is a very simple caching function on the website for application icons and pictograms. The function is located in the `Image` controller, but is not working very well. Proper caching is something that could be worked on for the next semester, since it could potentially save a lot of loads from the database and potentially save loading times for the page.

We were told at the end of the last sprint, that the icons used to show the difference between guardians, parents, and citizens should be bigger.

One of the features that the customers would like, was the ability to delete pictograms from the database. This was not implemented since it was suggested at the end of the last sprint, and such a feature requires a proper thought. They should not be able to delete pictograms that other people are using, so it has to be clear which pictograms they should be able to delete, and which pictograms they should not. To make this feature work properly a change to the database might be necessary.

It might be a good idea to go through all the form actions and check if the correct checks is being made, or if it is possible to make a local form with some specific data and by that perform actions to e.g. citizens that you should not be able to do.

Even though all the code have been tested, some of the code have been changed after the tests were performed. Therefore the test cases might need to be updated to fit with the new code. They should maybe also be more comprehensive than the current state.