Aalborg Universitet

SW6 Project

Developing Complex Software Systems

The Launcher for the GIRAF Software Suite

Group:
Jesper Byrdal Kjær
Frederik Bruhn Mikkelsen
Stefan Mailund Thilemann
Anders Vagner

Supervisor:
Hua Lu

AALBORG UNIVERSITY
STUDENT REPORT

AALBORG UNIVERSITY
STUDENT REPORT

Department of Computer Science
Software 5th semester
Address: Selma Lagerlöfs Vej 300
9220 Aalborg Øst
Phone no.: 99 40 99 40
Fax no.: 99 40 97 98
Homepage: http://www.cs.aau.dk

Project title:
The Launcher for the GIRAF Software Suite

Subject:
Developing Complex Software Systems

Project periode:
Spring 2014

Group name:
SW605F14

Supervisor:
Hua Lu

Group members:
Jesper Byrdal Kjær

Frederik Bruhn Mikkelsen

Stefan Mailund Thilemann

Anders Vagner

Copies: 6

Pages: 86

Appendices: 6 & 1 cd

Finished: 28th of May 2014

Abstract:

In cooperation between a number of institutions in Aalborg Municipality and Aalborg University, an Android system for autistic citizens and their guardians has been in development since 2011. In January 2014, the project was taken over by the new 6th semester software engineering students, whom this group was a part of.
This report documents the progress of the group SW605F14 collaborating with the 15 other groups working on the project. As the group mainly picked up the task of improving *Launcher*, the main application from which all other applications should be accessed, the report describes the continued analysis, client collaboration and development on this application. Furthermore, the challenge of working within a multi-project setting is described, as are the advantages and problems this setting brings.
The project was developed over four sprints, and thus this report is divided into chapters concerning the activities carried out during each sprint.

Preface

This report is created by Software Engineering students as a Bachelor project at Aalborg University, spring semester 2014. To read and understand the report, it is expected that the reader has a background in Computer Science in the light of the technical contents. The Android APIs are not described directly in this report and thus the reader is encouraged to explore the introduction to Android given by Google Inc. [7].

The project is organized in multiple groups, where 16 bachelor groups have collaborated to create an Android system for autistic citizens and their guardians. The multi-project has been in progress since 2011, and is a collaboration between Aalborg Municipality, Aalborg University and several institutions working with autistic citizens. The multi-project is further described in Chapter 1.

Since multiple project groups are collaborating towards developing a complete system, it is necessary to have a common work process. Therefore, it was decided by the semester coordinator to have the groups work in four sprints. The structure of this report is divided into chapters that logically follows from this work process. Each sprint chapter is then further subdivided into sections that are similar for each sprint. A particular section is only added if it describes activities carried out during the specific sprint.

It is important to note, that the applications in the multi-project were renamed during this semester. The new names better describe the essence of the applications. As a result, many applications have been given Danish names, which will not be translated to English in this report.

It should also be noted that some developments are not described by analysis nor design. This follows from the fact, that many developments are a consequence of having tasks in the backlog from a preceding sprint. Furthermore, these developments are often centered around the refactoring of existing code.

References and citations are given with number notation, for example as: Frandsen et al. [5], or without listing the authors as: [5].

We would like to thank our lecturers and especially our supervisor, Hua Lu, for excellent cooperation during the project work. Furthermore, we would like to thank the other project groups for the educational collaboration performed during the project.

Contents

Introduction **1**

1 GIRAF **3**
1.1 GIRAF Applications . . . 3
1.2 GIRAF Architecture . . . 3
1.3 Multi-project Management . . . 4

2 First Sprint **7**
2.1 Sprint Overview . . . 7
2.2 Analysis and Design . . . 8
2.3 Developments . . . 11
2.4 Functional Tests . . . 14
2.5 Sprint Summary . . . 16

3 Second Sprint **17**
3.1 Sprint Overview . . . 17
3.2 Analysis and Design . . . 17
3.3 First Client Meeting . . . 20
3.4 Second Client Meeting . . . 22
3.5 Developments . . . 23
3.6 Sprint Summary . . . 24

4 Third Sprint **27**
4.1 Sprint Overview . . . 27
4.2 Analysis and Design . . . 27
4.3 Developments . . . 31
4.4 Sprint Summary . . . 37

5 Fourth Sprint **39**
5.1 Sprint Overview . . . 39
5.2 Analysis and Design . . . 39
5.3 Developments . . . 40
5.4 Usability Test . . . 45
5.5 Sprint Summary . . . 48

6 Collaboration **49**
6.1 Google Analytics . . . 49
6.2 Integrating the Local Database in *Launcher* . . . 49
6.3 Integrating Database Synchronisation . . . 50
6.4 Sprint End Organization . . . 51
6.5 Informal Collaboration . . . 53

7 Evaluation **55**
7.1 Multi-project 55
7.2 Our Work Process 56
7.3 Discussion 56
7.4 A Summary of Future Works 57

A Specification Requirements and User Stories **59**

B Client Meeting 01-04-2014 **65**

C Client Meeting 03-04-2014 **67**

D Front End Applications **69**

E Further code examples **71**
E.1 Debug mode for development 71
E.2 Launching Google Play 72
E.3 Derived class of LoadApplicationTask 73
E.4 Marking GIRAF and Android applications 74

F Future Works **75**
F.1 Proper Icon Size 75
F.2 Ugeskema Calendarwidget 75
F.3 Login with Citizen and Administrator Profiles 75
F.4 Automatically Download GIRAF Applications 75
F.5 Adding Sound to Launcher 75
F.6 Profile Selection Dialog 76
F.7 Copying settings from one user to another 76
F.8 Authentication Activity 76
F.9 Download pictograms while tablet is being used 76
F.10 Use GridView for loading applications 76
F.11 Conduct Functional Tests 76

Bibliography **77**

Introduction

The desire for the latest technology, whether it be a new car that enhances the safety of driving it, a gesture enabled television set, or the latest mobile phones and tablets, often reflects a wish of improving certain areas of our lives using technology. This report focuses on improving the way-of-life of citizens diagnosed with different kinds of *Autism Spectrum Disorders* (ASD), including autism and Asperger syndrome. The goal is to continue the development of an existing Android software suite for tablets, consisting of tools that seek to replace and improve some of the citizen's current tools for easing their daily routines.

This suite is called *Graphical Interface Resources for Autistic Folk*, abbreviated GI-RAF.

Examples of GIRAF applications include *Livshistorier*, an interactive application that guides citizens in putting on their outdoor clothes in the best order, and *Stemme-spillet*, a game that focuses on training the voice of the speech-impaired citizens. These applications are described briefly in Appendix D.

GIRAF is developed in cooperation with Aalborg Municipality, which has several institutions specializing in children and adults with various kinds of ASDs. These institutions rely heavily on paper-based tools for their work with autistic citizens. The idea behind GIRAF is to organize and streamline these existing tools, by implementing them on Android tablets. The institutions are considered the clients of GIRAF, and therefore play a key role in guiding the developers, so GIRAF can become as useful to them as possible.

The ideas of the clients are formalized by a group of students as a requirements specification and user stories, which contains requirements for the further development of each application, as described in Appendix A.

CHAPTER 1

GIRAF

The term “GIRAF ” describes the entire multi-project as an entity. The suite of GIRAF applications is developed iteratively at Aalborg University’s Department of Computer Science, by software engineering students on the final semester of their bachelor. The project was initiated in 2011 by Ulrik Nyman, Associate Professor at Aalborg University. The project has subsequently evolved with each class of students, bringing new ideas to the project and improving on the existing applications. Furthermore, the requirements from the clients has changed over time.

Overall, the multi-project is designed for various kinds of people. The most important of which are the *citizens*, for whom the applications were designed. A *citizen* refers to either a child or an adult person diagnosed with ASD. Each citizen is supervised by another adult, referred to as being a *guardian*. This person is typically employed in one of the institutions affiliated with the GIRAF project. Thus, when referring to both, the term *user* is used.

This chapter gives a more thorough introduction to the GIRAF multi-project. For a description of the development methodology employed, see Section 1.3.

1.1 GIRAF Applications

The GIRAF suite consists of several components, not least a multitude of front end applications, providing various features to the users. All front end applications are described in Appendix D. The suite also includes a number of back end components, allowing sharing and controlling data across every application. The back end components are described below.

1.1.1 Back End Components

Central to most of the Android applications is *OasisLib*, which provides both models for the entire problem domain, and interfaces for retrieving data from the local database. Every device has its own local database, with the goal of regular synchronization with a single remote database. Finally, *GIRAF Components* contains common user interface components with a uniform look-and-feel.

1.2 GIRAF Architecture

The GIRAF system consists of several layers, as illustrated in Figure F1-1. At the top we have applications such as those described in Appendix D, which communicate with a local database on the device through the library *OasisLib*.

GIRAF Components consists of multiple graphical components and provide common services to other GIRAF applications. As an example, a service that applications need is a profile selector. This component allows a guardian to choose with which citizen they wish to work. As a result, the process of choosing a citizen will be the same throughout GIRAF.

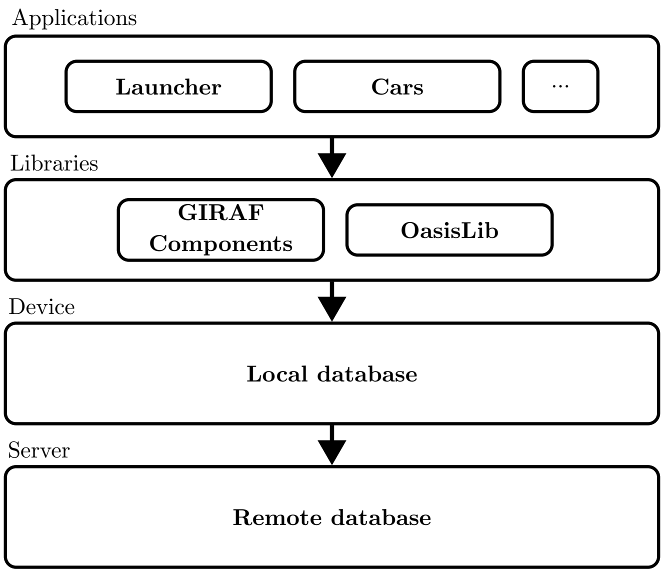


Figure F1-1: Illustration of the layers and dependencies in the GIRAF application suite.

1.3 Multi-project Management

The entire GIRAF multi-project consists of 12 Android applications, two Android libraries, two iOS applications, two web-based applications and a remote database server. During the period described in this report, around 60 developers divided into 16 teams worked on these components. As nearly all of these components either depend on others or are depended upon by others, it was naturally necessary to make the effort somewhat coordinated.

Allowing the groups to develop their individual components, with no communication between them, will obviously result in an incoherent and chaotic system, that probably will not run at all. It is therefore necessary to set up a framework for the cooperation between the groups, and for ensuring that the multi-project as a whole will move towards fulfilling the needs of the clients.

1.3.1 Scrum

By vote, the multi-project development team decided to use Scrum as the overall method of project management. We also decided to encourage the use of Scrum in the project groups, but to leave the final decision to the individual groups.

Larman [14] explains that a central part of Scrum is the Scrum Master, who facilitates development by solving problems and enforcing Scrum. We decided on a single Scrum Master for the entire project period, to avoid losing gained experience with each changing Scrum Master.

While the Scrum method recommends daily Scrum meetings, we decided to settle with a weekly meeting for the multi-project Scrum. The development tempo is relatively low as we also have to follow courses. Some of these courses are electives, so it will also be difficult to gather every group at the same time every day.

A short evaluation of the use of Scrum on the multi-project level can be read in Section 7.1.1.

1.3.2 Sprints

We decided to work in sprints. Changing requirements is the main argument against the traditional “single-sprint” model. The resulting system may not live up to the clients’ expectations, so it is important to regularly align the visions of the clients and the developers.

As a unit for measuring time in the sprints, we chose “half days”, which last 3-4 hours. This unit fits well into our schedule, with relation to course modules. The available half days were divided into four sprints:

1. First sprint starts 24-02-2014 and ends 19-03-2014.
2. Second sprint starts 20-03-2014 and ends 14-04-2014.
3. Third sprint starts 15-04-2014 and ends 07-05-2014.
4. Fourth sprint starts 08-05-2014 and ends 27-05-2014.

At the end of each sprint all groups should present and demonstrate their topic at a sprint end meeting with the clients.

1.3.3 Roles and Responsibilities

A large number managerial tasks were delegated to members of the different groups. The most notable were:

- The *Scrum Master* solved tasks mainly related to coordination. His chief task was to chair the weekly Scrum meetings, where the groups summed up their progress, and brought up problems and suggestions relevant to other groups. He also had some tasks related to project-wide initiatives and problems, e.g. readying the applications for release to Google Play.
- The *Client Contact* took care of all communication with the clients, giving the them a single contact point, and made sure that clients were not overburdened with individual requests and questions from the 16 groups.
- The *Sprint End Specialist* was appointed after the problematic first sprint review meeting, described in Section 6.4. His task was to plan and organise the sprint review meetings, where the groups’ progress is presented to the clients.

CHAPTER 2
First Sprint

2.1 Sprint Overview

The first sprint focused on getting to know the GIRAF code base, which we inherited from the students working on the project the previous years.

This section covers the layout of the first sprint of the project, which sets itself apart from the following sprints, since it focuses on getting to know GIRAF in details, setup of the development environment and getting the system up-and-running.

2.1.1 Planning

The agenda for the first sprint meeting is to get an overview of the development taken place during the preceding semesters, and to get development started. To begin with, each group has chosen a GIRAF application to work on, even though this selection is subject to change during the following sprints. We have chosen the to work with the application launcher, which is further described in Section 2.2.1.

It is also decided that the following tools are used to strengthen cooperation between all project groups:

- **Redmine** (project management web application)
- **Git** (distributed version control system)
- **Jenkins** (continuous integration)
- **Android Studio** (Android integrated development environment)
- **Gradle** (build automation control)

This selection differs from that of last year, since Git has replaced Subversion (SVN), Android Studio has replaced Eclipse and the Gradle buildsystem has replaced Ant.

2.1.2 Objectives

Since this is the first sprint and thus we have no prior knowledge of GIRAF, this sprint can not focus on any new developments. Therefore, attention is given to the work done by last years students working on the project, looking for improvements and fixing bugs. During the first sprint meeting, it was clear, based on advice from semester coordinator Ulrik Nyman, that we should strive to deliver a functioning suite of applications, especially at the end of the project period. It should not be understood as a version to be delivered to the clients, but merely as an installable version that can run without any crashes.

A separate project group was tasked with handling all negotiations with the clients during this first sprint in order to compile a backlog for the second sprint. They are also undertaking the act of creating a requirements specification for the entire project together with user stories explaining how common tasks are executed. It can be seen in its entirety in Appendix A.

The rest of this chapter focuses on the application launcher.

2.2 Analysis and Design

This section considers the analysis of application launcher based on its current state. First the launcher is shortly introduced, followed by our motivation for taking over its development. Next, the functionality of the launcher is described. Lastly, the section focuses on our desired improvements found by looking at the existing code-base and the report by Andersen et al. [1], which is the last project group working on the launcher, with emphasis on the drawer component.

2.2.1 Launcher

A launcher in Android terminology is an application that provides easy access to other applications present in the system. When the system is fully booted, the launcher is automatically started and thus is comparable to booting a desktop computer directly to the desktop.

From this point and onwards, when referring to *Launcher*, we are drawing attention to the launcher that is part of the GIRAF application suite.

The main purpose of *Launcher* is to provide a user-friendly means of accessing other applications in the GIRAF suite, as well as regulating access to these based on the active user profile.

Motivation for Working with Launcher

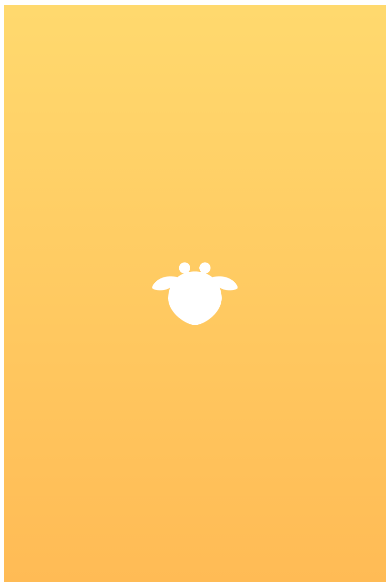
Launcher was originally developed by Frandsen et al. [5], and further refined by Andersen et al. [1]. At an introductory meeting with the clients however, they made us aware that *Launcher* is rarely used, as it often crashes. They preferred to access the GIRAF applications through the Android native launcher. As mentioned in Section 2.1.2, the focus in this sprint is on creating running applications, and since *Launcher* was relatively complete based on the development of Andersen et al. [1], but reported as being unreliable by the clients, an obvious task for this sprint is to make *Launcher* run reliably.

Launcher Functionality

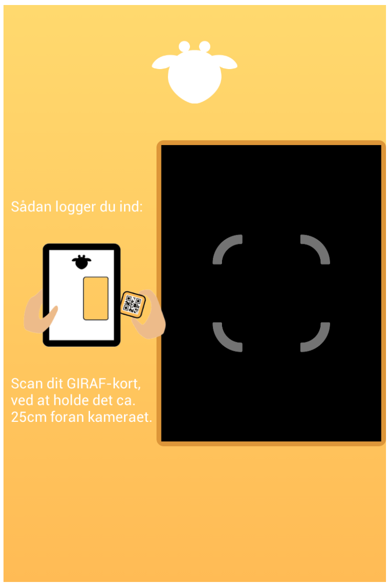
As *Launcher* covers multiple tasks in the GIRAF project, this section gives a detailed description of its most important activities. Screenshots of each activity is found in Figure F2-1.

Main Activity is only responsible for showing the GIRAF logo for a specified amount of time, as seen in Figure F2-1a, and then redirecting the users to the proper activity. It redirects to *Authentication Activity* if the user is not logged in, or to *Home Activity* if the user is already logged in.

Authentication Activity requires the user to authenticate him- or herself before proceeding. Andersen et al. [1] describes why user identification is necessary, as access rights to the different GIRAF applications must be able to vary between users. The



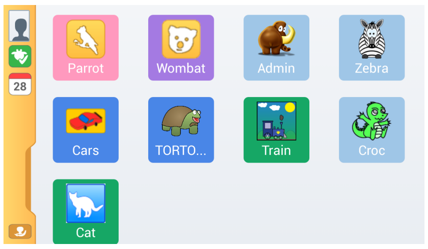
(a) *Main Activity.*



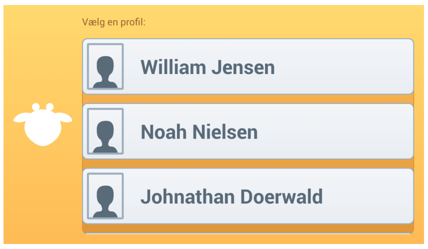
Sådan logger du ind:

Scan dit GIRAF-kort, ved at holde det ca. 25cm foran kameraet.

(b) *Authentication Activity.*



(c) *Home Activity.*



(d) *Profile Selection Activity.*

Figure F2-1: Activities of the GIRAF application.

report furthermore describes how autistic citizens might have problems using a traditional user name and password system. Therefore, *Authentication Activity* is based on a QR-scanner, as seen in Figure F2-1b. Each citizen and guardian then uses a small brick printed with their personal QR-code they scan to identify themselves. The activity loads the user information from the database, and proceeds to *Home Activity*.

Home Activity allows the user to launch GIRAF applications available to his or her user profile. The availability of applications depends on whether the application exists locally on that device, and on whether the user is marked in the database as having proper access rights to that application. *Home Activity* for a guardian user is seen in Figure F2-1c with the users installed GIRAF applications. Furthermore, a widget allows the user to see the synchronization status of the local database in relation to the remote database[1], and another widget shows the current date. There is also a widget that allows the user to log out, and return to *Authentication Activity*. These widgets are supplied by *GIRAF Components*. Finally, there is a colour palette hidden in a drawer component, where the user can change the base colour of each installed application. The idea is to make it easier for the citizens to differentiate between the various applications by being able to choose their own colour scheme. Ideally, the choice of colour should also reflect in the application started from *Launcher*, giving

[1]The connection indicator widget currently has no effect, since database synchronization is not yet implemented.

the citizen consistent visual associations throughout GIRAF. The latter is for example implemented in the *Tidstager* application.

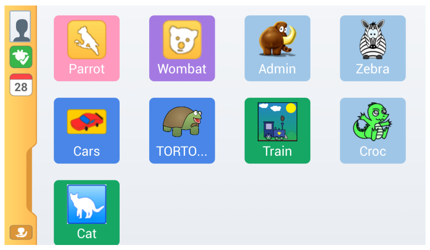
Profile Selection Activity is started when a guardian starts an application from *Launcher*. It displays a list of all citizens associated with this guardian, seen in Figure F2-1d, allowing him or her to choose which citizens profile to use, when starting the selected application. When a citizen is logged in, the application starts, omitting *Profile Selection Activity*.

2.2.2 Drawer

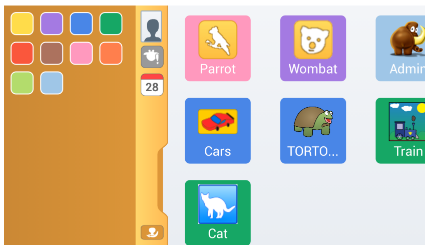
The drawer is a separate view that is situated on the left-hand side of *Home Activity*. Its main purpose is to allow *Launcher* users to apply a new colour scheme to the installed GIRAF applications.

To avoid cluttering the screen, the drawer can be shown and hidden at will. This means that the only part of the drawer that is visible at all times is the sidebar, which can be dragged towards the centre of the screen to reveal the content of the drawer. The sidebar also contains the informative widgets described in Section 2.2.1.

Illustrations of the drawer in closed and opened state can be seen in Figures F2-2a and F2-2b respectively.



(a) Drawer closed.



(b) Drawer fully opened.

Figure F2-2: States of the drawer component shown in *Home Activity*.

As indicated by Figure F2-2, the drawer already meet the requirement that a colour can be dragged onto an application. In spite of this, many deficiencies have been located, which are further enlightened by the following sections.

Behaviour of the Drawer

The drawer opens by dragging the sidebar to the right. While being opened, the drawer pushes the application icons along with it to the right, resulting in some icons disappearing out of view. Furthermore, the drawer stops sliding when pressure is released from the touch-screen, meaning it can be left in a half-open, half-closed state. To change the colour of an application, a colour can be dragged and dropped onto an application icon. The result is immediately apparent, as the colour framing the icon changes to the selected colour.

Desired Improvements

Since no formal requirements are available and we have not yet held a meeting with the clients, the suggested improvements below is based on what we find to be natural requirements of such an user interface component.

- The drawer should be either open or closed. A half-open or half-closed state should result in the drawer popping into the state closest to its current position.

- The drawer should close while a colour is being dragged, and open again when the colour is dropped.
- The drawer should not push the application icons out of the screen.
- The source code responsible for the animation of the drawer and the layout of *Home Activity* should be refactored in order to
  - take advantage of standard Android animations and layout features,
  - allow the activity to dynamically adjust according to display-size and
  - reduce the amount of clutter in and improve readability of the source code .

The implementation of the improvements is described in Section 2.3.2.

2.3 Developments

As previously mentioned in Section 2.1, time is dedicated to fixing minor bugs and getting to know the code base. Furthermore, as described in Section 2.2, the drawer is improved upon. As a result, the developments in this sprint focuses on four major points:

- Minor bugs are identified and corrected.
- The existing code is refactored to improve its readability.
- The layout is restructured to be more dynamic and compliant with Android guidelines.
- Various minor improvements are made to *Launcher*.

This section begins with a description of the minor bugs found and continues with a more detailed description of the improvements made to *Launcher* and its drawer component.

2.3.1 Minor Bugs

Several minor bugs, anomalies and things that did not work as expected were found in the beginning of the sprint. Considering their negligible significance, we will not discuss them in depth, but rather describe each briefly.

Citizen Mode

Regardless of whether the currently logged in profile is a citizen or a guardian, *Launcher* opens *Profile Selection Activity* before opening an application. It should only open that activity while a guardian profile is logged in. The issue is solved by checking on the `getRole()` parameter of a profile, and then appropriately override the `OnClickListener` of *Profile Selection Activity*, depending on the role.

Force Landscape Orientation

Reports from 2013 developments describes that landscape orientation should be forced in all GIRAF applications[15, p. 33]. Since *Launcher* is currently able to enter portrait mode, it is corrected by changing `AndroidManifest.xml` to only allow landscape orientation for all activities[6].

Issues related to other groups

Two issues caused by other projects are also discovered:

- The `GButton` widget from *GIRAF Components* library crashes any activity that implements it.
- Danish characters are not encoded correctly in the database.

The issues are reported to the groups responsible for these components through the issue tracker.

2.3.2 Improved Behaviour of the Drawer

As mentioned in Section 2.2.2, the drawer is opened by dragging the sidebar towards the right. The code working the animation of the drawer is based on an `OnTouchListener`, using a motion event which is fired only by movement. The event sets the position of the entire drawer, panel and sidebar to the exact point it has just been moved to, before redrawing the elements. Since the position is not limited, the user can leave it in any position between fully extended and closed state, as explained in Section 2.2.2.

To change this behaviour, an Android `TranslateAnimation` class is implemented to start as soon as the user touches the sidebar of the drawer. A `boolean` variable determines whether the drawer is open or closed to force the animation to either translate left or right, depending on its current state. By also adding an `OnDragListener` to the drawer, the animation will also play when dragging and dropping colours from the panel; closing the drawer when a drag is initiated and opening it again when the drag has ended.

2.3.3 Refactoring the Drawer Layout

The behaviour of the original drawer, described in Section 2.2.2, has the unfortunate effect of pushing the applications icons to the right, as the drawer is dragged open. This is caused by the view containing the application icons having the `android:layout_toRightOf` attribute set to be placed to the right of the sidebar in the layout XML files. This makes its position adjust automatically to the newest rightmost position of the sidebar each time the bar is moved, effectively pushing the icons off the screen.

The main issue with the current implementation of *Home Activity* layout lies in the fact that many parameters such as heights and widths are set at compile-time through constants. The result is that the layout does not adapt to other devices than the one it has been designed for, which is not viable if the clients choose to replace their current ones. The improved implementation includes restructuring all parts of the XML file defining the layout of *Home Activity*, making it adapt to devices with various screen sizes and pixel density.

Implementation

To avoid the icons being pushed outside the screen, the layout for the drawer is restructured so its elements are contained within a custom view called `DrawerLayout`. This ensures that the animation and layout hierarchy can be controlled more effectively, making the drawer being drawn on top of other layout elements.

Listing 2.1 shows the elements of the XML layout.

```
1  <dk.aau.cs.giraf.launcher.layoutcontroller.DrawerLayout
      android:id="@+id/Drawer" android:layout_marginLeft="-400dp">

3  <RelativeLayout android:id="@+id/DrawerContentView"
      android:layout_width="400dp">
4  <GridView android:id="@+id/AppColoursView" />
5  </RelativeLayout>

7  <RelativeLayout android:id="@+id/SidebarView">
8  <dk.aau.cs.giraf.gui.GWidgetProfileSelection />
9  <dk.aau.cs.giraf.gui.GButtonSettings />
10 <dk.aau.cs.giraf.gui.GWidgetLogout />
11 </RelativeLayout>

13 </dk.aau.cs.giraf.launcher.layoutcontroller.DrawerLayout>

15 <ScrollView android:id="@+id/AppContainerView"/>
```

Listing 2.1: Structure of the XML layout of the drawer. Note that some attributes are omitted and others are renamed to improve understanding.

Notice the `ScrollView` on the last line in Listing 2.1. The before mentioned `android:layout_toRightOf` attribute on this `ScrollView` has been removed, to keep it from being pushed out of the screen. The margin of $-400dp$ defined on Line 1 effectively hides the `SideBarLayout` when *Home Activity* has just started, revealing only the `SideBar`. Since the `SideBar` is placed next to the `GridView` and this view has the width of the margin defined on `SideBarLayout`, it is the only visible element. This gives the impression of a closed drawer, and is illustrated in Figure F2-3.

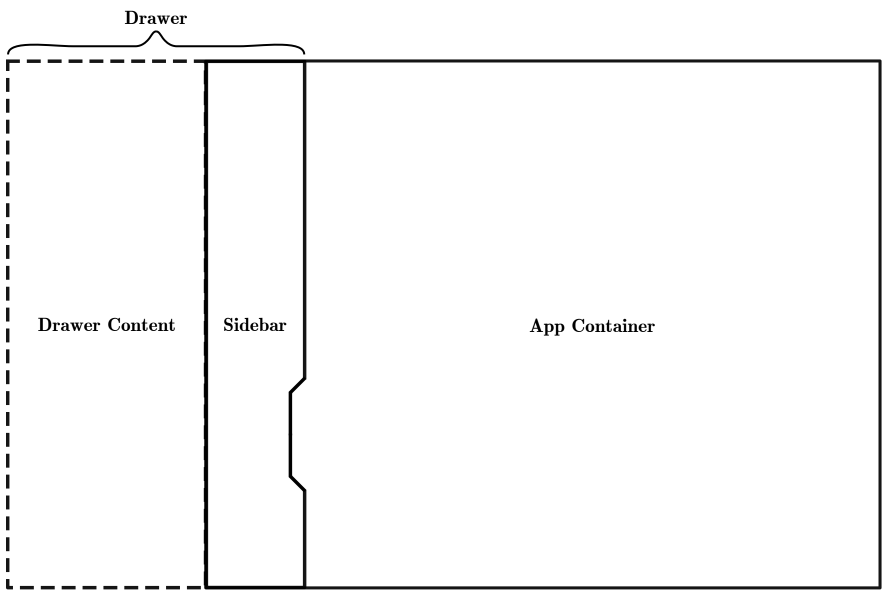


Figure F2-3: The layout of the drawer when it is closed. This dashed lines represent parts of the drawer that are not visible to the used.

2.3.4 The Error in Authentication Activity

When starting *Launcher* for the first time, the application crashed with an `OutOfMemoryError` upon reaching *Authentication Activity*. The problem is caused by an animation in the activity, implemented as an instance of the Android class `AnimationDrawable`, a class designed to create animations from a sequence of images. According to the Android documentation Google Inc. [8], the implementation is technically correct, but it is found that when an instance of `AnimationDrawable` is created, it loads the entire sequence of images into main memory of the device at once. The images are contained within the application in the compressed Portable Network Graphics (PNG) format. However, as they are loaded, Android converts each image to a 32-bit bitmap, which is 4 bytes per pixel, making the images take up about 20 times the space. The problem with the original implementation is, that the animation consists of 90 pictures, thus resulting in the animation using about 40 MB of memory when converted to bitmaps. We hypothesized that this was the source of the `OutOfMemoryError`.

Our solution is to create a new class which simulates the behaviour of `AnimationDrawable`, but only loads one frame at a time. This is achieved by a recursive method, which changes the shown image automatically. The image references are stored in a static array, which is iterated repeatedly using the method `postDelayed()`, that runs a given method after a set delay.

In order to further ensure the reliability of *Launcher*, functional tests of the application are also carried out. These are described in the following section.

2.4 Functional Tests

As one of our goals is to increase the reliability of *Launcher*, we decided to conduct functional testing of the application. We studied the specifications formulated by the earlier development teams[5, 1], and then studied how well *Launcher* correspond to these specifications by using it in a number of different ways.

Furthermore, a number of test cases are not directly derived from the old specifications, but from what we find reasonable to expect of the application, based on our own knowledge of the system.

The tests described here are performed by operating the running application, and are explicitly not based on our knowledge of the source code. This characterizes our tests as being dynamic black box testing[17].

A number of test cases that would have been relevant is not performed during this sprint, as we deem that they are not sufficiently interesting, considering the time needed to run them. This includes using *Launcher* with a varying set of applications and users. These test cases would require significant changes to the test data, which is hard coded into the GIRAF system, or require repeatedly installing and uninstalling applications from the device. As there is currently no central repository containing stable application builds, most of the time will be spent building, installing and resetting different applications.

We only describe the results we find the most interesting in this section; mainly tests that failed in a significant way.

2.4.1 Specifications

In the 2011 report by Frandsen et al. [5], the following specification for *Launcher* is presented:

- *List only applications that [are] part of the GIRAF system.*
- *List only applications that the user is able to use, according to the user's capabilities.*

- *List only applications if their usage is not limited by the current location or time.*
- *Only display application names if the user is able to read.*

Common to all these is that they are either difficult to test, or simply not implemented, though some may have been to some degree in 2011 and then removed by the 2012 team. The difficulty stems from the necessity of manipulating test data that is hard coded into the local database of each device. We decide it would be more reasonable to revisit these test cases when GIRAF's database system is ready for use.[2]

The 2012 report by Andersen et al. [1] contains no explicit specifications, but does contain a list of use cases. These form the basis of our tests:

- *New guardian log in.*
- *Configuring an application for a child.*
- *Launching an application for a child in guardian mode.*
- *Letting a child use an application for a limited time.*

Note that their report also contains an explanation of each use case. We leave these out for brevity. Also note that what the quote refers to as a *child*, this report refers to as a *citizen*.

2.4.2 Test Results

Below is shown a selection of our test cases and their results. As mentioned in the introduction of this section, we will only discuss the most interesting results, i.e. tests that led to the discovery of significant issues.

Changing the Colour of an Application Icon

Specification: Configuring an application for a citizen.

Test case: Use the drawer to change the colour of an application, and check if the change is applied to both the application's icon, and its own user interface.

Result: The colour of the icon changed immediately, but when starting the application, it still used the pre-change colour. A little experimenting showed that the change is not applied to the application until the user logs into *Launcher* again.

Resolution: The issue was resolved, and the change in colour is now reflected in the application immediately.

Applications that Crash Launcher

Specification: Starting an application for a citizen in guardian mode.

Test case: Open an application with a randomly selected citizen profile by using the profile selector.

[2]The database system is implemented in the fourth sprint, however, it is implemented at such a late stage, that it was not possible carry out functional tests before the deadline. More information can be found in Section 5.5

Result: While not directly related to the intent of the use case, during the test, we attempted to open what turned out to be a non-working version of a GIRAF application. The application suffered from an unhandled exception, causing both it and *Launcher* to crash. This is not satisfactory, as *Launcher* should always work on top of the device operating system, denying the citizen access to device settings when using the device without supervision.

Resolution: The issue was resolved by making *Launcher* catch all exceptions thrown by the GIRAF applications it hosts.

Use of Device Buttons

Specification: The device buttons (*Back*, *Home*, and *Multitasking*) should have no effect in *Main Activity* and *Authentication Activity*, as the user should not be able to "back out" of *Launcher* to the device OS.

Test case: Use the device buttons in different activities and situations.

Result: In both *Main Activity* and *Authentication Activity*, applying the *Home* and *Back* buttons made *Launcher* restart. The *Multitasking* button always activates the multitasking screen of the device OS.

Resolution: The *Back* button are suppressed by overriding a listener method. We were unable to find similar means for suppressing the other two buttons, as we unsuccessfully returned to this issue in the second sprint, as described in Section 3.6.2.

2.5 Sprint Summary

2.5.1 Evaluation

Overall, the first sprint was a success for us. All task were finished and *Launcher* is now ready for the sprint review together with the clients, receiving feedback regarding the next steps.

However, for various reasons, the meeting does not yield the desired results for planning the next sprint. Therefore, the groups decide to appoint a single person responsible for planning and executing this meeting in the future. The chosen person is from this group, so the details of this and later sprint reviews are discussed in depth in Section 6.4.

2.5.2 Backlog

As mentioned above we did not get any feedback during the sprint review to use for planning our next sprint. We still think there are elements to improve in *Launcher*, but we cannot immediately continue the development, before having explored whether the clients have further requirements for the application. These are explored in Section 3.1.

CHAPTER 3
Second Sprint

3.1 Sprint Overview

Having completed all remaining task from the first sprint, the second is about making a foundation for future work. The idea of creating a settings button in *Launcher* is explored by the means of prototypes and client meetings.

As the described in Section 2.5, the end of the first sprint left us unsure of whether there was more work to be done on *Launcher*, and if so, what this work could be. The necessity of arranging a meeting with some of the clients is decided, in part to hear their thoughts on the current version of *Launcher*, and in part to discover new features. During the first sprint we informally discussed ideas for new features. These features include an improved format for the profile selector, and a new *Settings Activity* and providing a uniform interface for changing settings across all applications. To provide a good foundation for presenting and discussing these ideas with the clients, a number of prototype drawings are designed to illustrate how our initial thoughts can be realized.

Another important issue in this sprint is adapting *Launcher* for new versions of its dependencies, among these *OasisLib*. The library underwent drastic changes at the end of the first sprint, which unfortunately resulted in several applications, including *Launcher*, being unable to run.

3.2 Analysis and Design

This section is centred around extending *Launcher* with new functionalities. The ideas originates from discussions at the weekly Scrum meetings[1], and are developed in form of graphical prototypes. The prototypes are in subsequent sections, Sections 3.3 and 3.4, presented to the clients for feedback.

3.2.1 Prototypes

At the end of first sprint, the existing features of *Launcher* were polished, and the project would only need to be kept updated with the implementation of the new *OasisLib* database. We thought it could be desirable for *Launcher* to have additional features:

- It might be useful to be able to change profiles at other occasions than when starting an application, possibly in *Launcher* or in the started application.
- Settings for the different applications could be set from an activity accessed from *Launcher*.

[1]The Scrum meetings are described in Section 1.3.1.

- Permissions for which applications users can access could likewise be set from *Launcher*.

It was decided to have a brainstorm to gather ideas regarding these features, create prototypes of the best ones[2] and present these to the clients for feedback.

The prototypes created were **low fidelity**, **horizontal** prototypes, as described in Benyon [2, p. 184-195], made partly with screenshots of existing functionality and partly with the modelling tool Gliffy (Kohlhardt and Dickson [13]). The purpose was to convey several different ways of including the above features, while not spending too much time creating them.

Profile Selection

At the end of sprint one, the profile selection would only appear when launching an application, as illustrated by the state chart diagram in Figure F3-1.

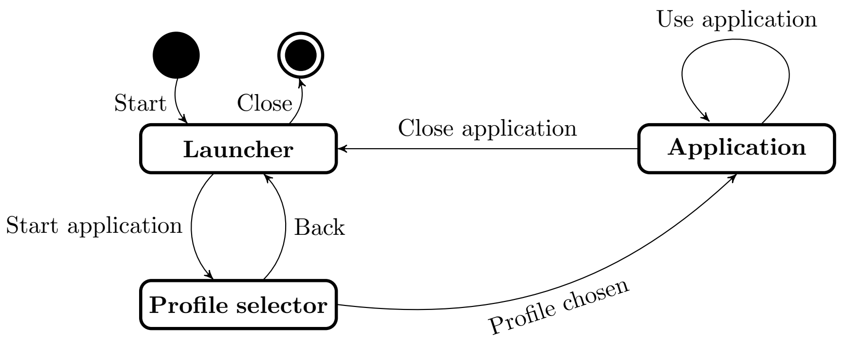


Figure F3-1: State chart diagram for the original process of choosing a profile.

Two alternatives are suggested:

- Selection of a different profile from within *Launcher*. This will work by pressing the profile picture, giving a list of profiles to choose from. Being relevant only for a guardian, the profiles selectable should only be the guardian itself and a related citizen.

- Selection of another active profile from within each individual application. While this solution does not require any modification of *Launcher*, it would mean that each application has its own profile selection button. As a result, a user does not have to exit an application simply to change profile.

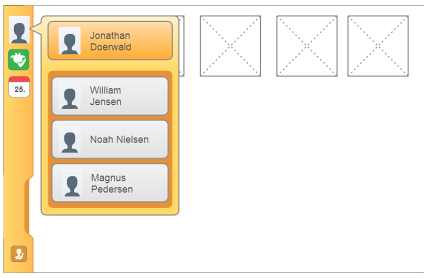
Common for both alternatives is to display the currently selected profile at the top of the sidebar. Illustrations can be seen in Figures F3-2a and F3-2b respectively. For an explanation of the suggested process of choosing a profile, see the state chart diagram in Figure F3-2c.

Settings

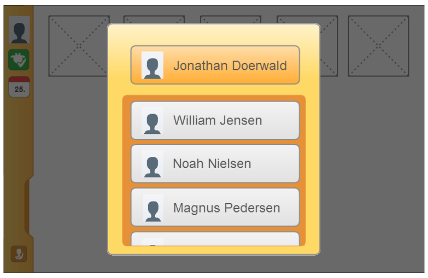
At the end of first sprint, settings for different applications where handled inside the applications exclusively. The idea of centralizing the settings for the different applications was formed both to streamline data for the database, but also to grant the user a better overview over the settings shared by applications.

Again, two alternatives are presented:

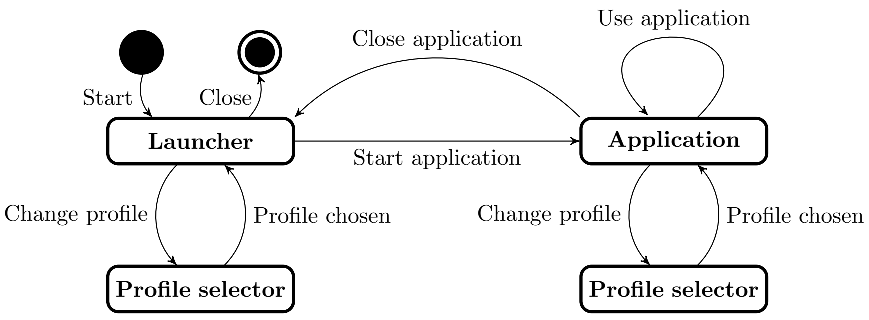
[2]The prototypes regarded as being the best, were deemed so by the project group.



(a) The dropdown profile selection option. This appears when tapping a profile picture as a guardian.



(b) The profile selection pop up. This only appears while pressing the appropriate button inside an application.



(c) State chart diagram for the suggested alternative process of choosing a profile.

Figure F3-2: Suggestions for the new profile selector.

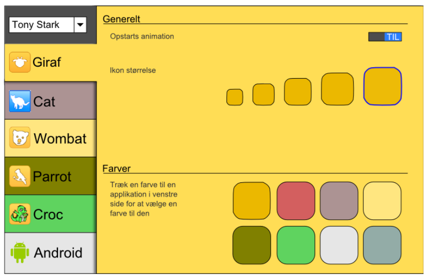
- A settings activity, accessible from *Launcher* only. This activity will contain settings for all applications enabled for the selected user, as well as selecting which applications should be enabled for that user. This also includes the native Android settings.
- Letting each application have a settings button, where settings native to each application can be set.

The first option is the most encompassing one, containing functionality for both manipulating settings for other applications and to decide which applications can be used by different users. It is furthermore discussed to add functionality allowing a user to add applications from Google Play to *Launcher*. The settings part of the activity can be seen in Figure F3-3a, while the application management part can be seen in Figure F3-3c.

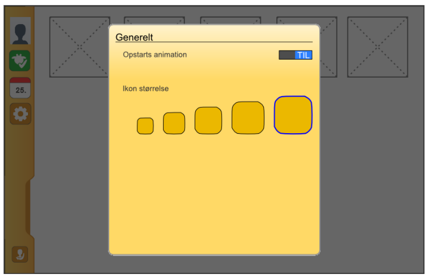
The second option is more centred on making the settings activity streamlined across applications and would thus be a cooperative project with the *GIRAF Components* group. An example of settings for *Launcher* can be seen in Figure F3-3b.

Profile Info

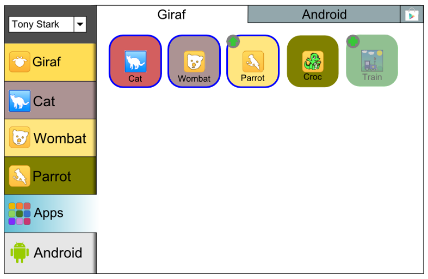
We have an idea of displaying information about the active user in the drawer section of *Launcher*. This is a relatively small feature that would be trivial to implement. An example can be seen in Figure F3-3d.



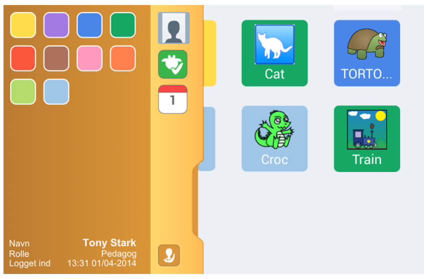
(a) The settings activity with the *Launcher* settings open. The current profile being edited can be seen in the top left corner.



(b) The settings for a single application as a dialogue. In this example, the settings for the home screen of *Launcher* is seen.



(c) The application management part of settings. GIRAF or Android applications can be chosen in the pane in the top.



(d) An example of information about a user displayed in the drawer component.

Figure F3-3: Suggestions for settings and application management.

3.3 First Client Meeting

This section focuses on the first client meeting, that is held to determine the viability of the thoughts described in Section 3.2.1.

It is conducted with one client and an external contact on the 1st of April, 2014. The client is Drazenko Banjak from the institution called Egebakken, which is a school for children with autism. The external contact is Pernille Hansen Krogh, the owner of the company Mymo, which works with design and décor optimized for learning. Together, they are referred to as the *clients*.

A full summary of the meeting can be found in Appendix B.

3.3.1 Feedback

Since the content of the meeting is mostly concerned with new developments, a slide show is made to support the presentation of the prototypes. Following that the clients not were staying to discuss GIRAF at the sprint review, a tablet with the most recent version of *Launcher* is also brought to get feedback on the changes made during the first sprint (Chapter 2).

Profile Selector

The profile selector prototypes (Figures F3-2a and F3-2b) are presented, and both clients are thrilled about the idea of being able to easily switch profiles from *Launcher* and each GIRAF application individually.

Settings

The idea of adding settings management to *Launcher*, both regarding settings for *Launcher* itself and for all GIRAF applications, is the main idea to be assessed at this meeting. Therefore the two possibilities, seen in Figures F3-3a and F3-3b are presented. The clients like the visual look of the prototypes, but have a big concern that the design would add to much complexity to *Launcher* and that Drazenko's colleagues at Egebakken would be scared of using it. They like the idea of changing settings from each application for the logged in user, even though we argue that it would impose higher workloads to the guardians if they need to change settings for many users. A hybrid consisting of both possibilities is suggested to overcome this.

The clients made the following suggestions based on the presentation of the settings management application:

- Add an external (Android) application to *Launcher*.
- Restrict access to use other applications when a guardian has assigned a citizen to work with a specific application for a specified amount of time, using *Tidstager*.

These suggestions are further considered in Chapter 4.

3.3.2 Clarification of Issues

Although Section 2.2.2 consider improvements to the drawer component in GIRAF, a major discussion at the meeting is about whether to keep the drawer or not. More specifically, Pernille make us rethink if users should be able to colour applications. If not, it will render the drawer unnecessary.

To overcome this decision, Pernille also suggested us to hold another meeting with Birken, since they are the originator of the requirement. This is discussed further in Section 3.4.

3.3.3 Communication

At this meeting the communication between the clients and us are quickly characterized as being problematic. The reason being that one client does not always quite understand what we are asking about. We establishes two possible reasons:

1. He (Drazenko) is lacking technical domain knowledge and does not know the correct terminology[3] or
2. he is of foreign extraction and is thus not a profound speaker of Danish.

Pernille, on the other hand, show great, technical understanding, even though she, as an external contact, has had no previous experience with the GIRAF project. Thus, she does not contribute regarding how GIRAF is used 'out in the field', but touches areas that make us rethink previous decisions.

The above facts result in a somewhat vague feedback, in the sense that the client is not always able to clearly formulate his requirements. Another problem is that when asking a question regarding one topic, feedback is given on another topic which at times are irrelevant in the context of *Launcher*.

Because of these defects in communication, the meeting is not considered to be a complete success, but useful feedback regarding the settings design has nevertheless been received.

[3]We quickly established this fact and thus made an effort to use as many non-technical terms as possible.

3.4 Second Client Meeting

The meeting is held the 3rd of April 2014 with two representatives Mette Als Andreasen and Kristine Niss Henriksen from the nursery for children with ASD, called Birken. A summary of the meeting is found in Appendix C. The presented prototypes are the same as those in the previous meeting (Figure F3-3).

3.4.1 Feedback

This meeting was conducted in the same manner as Section 3.3, therefore please refer to the Feedback section from this meeting in Section 3.3.1.

Profile Selector

The profile selection tool is preferred as a combination of the two options – as a selector inside each launched application and as a tool-tip from clicking the profile picture in *Launcher*.

Settings

The settings tool is preferred also as a combination. Accessing the settings related to each application from inside that particular application is the most important of the two. Accessing the settings is achieved by pressing a button with an image of a cogwheel. Furthermore, pressing the settings button while in *Launcher* should open an activity, where settings for both *Launcher* itself and all installed applications can be manipulated. An important aspect of the settings is the ability to copy settings from one user to another.

External Applications

Lastly, the idea of being able to add applications from outside the GIRAF suite through *Launcher*, including downloading new applications from Google Play, is well received. The clients especially like the idea of enabling applications on a ‘per user’-basis. This functionality is advised to be in the settings activity found in *Launcher*.

Android Buttons

Apart form the prototype, the solution of overriding the *Home*, *Back* and *Multitasking* buttons is accepted, but the clients prefer it to be disabled completely. It is suggested by the clients to utilize the Android equivalent of iOS’ “Restricted Access”, to achieve the preferred result. The research is described in Section 3.6.2.

3.4.2 Clarification of Issues

One major issue regarding conflicting requirements is clarified at this meeting. Previously, there was a requirement of a basic black-and-white colour scheme, with options to change scheme. It is then clarified that this requirement is related only to the pictograms and not to the program as a whole, meaning the pictogram should in general be black and white and those in colour should be able to be converted to black and white. The drawer component in *Launcher*, responsible of changing the colour of applications was not a requirement, and therefore is not particularly important to the clients.

3.4.3 Communication

Communication with these clients is more smooth than at the previous meeting, however, there are still comprehension barriers. Most barriers are related to the clients not understanding what features would be trivial to implement and which would be exceedingly difficult. Furthermore, questions asked by us often have to be reformulated for the clients to understand them completely. However, the clients often explain how each question is understood, and always point out elements they do not understand. This helps greatly in case of communication problems.

These clients furthermore express more concretely what is desired from the application. Additional questions are still required, but are answered to completion.

Overall, the meeting is regarded as a success, with constructive feedback.

3.5 Developments

While no developments have been planned for this sprint, a situation arises, forcing us to develop upon *Launcher*. The situation is related to the dependencies between the multi-project groups. Because of these dependencies, this situation will arise in every sprint, however, the developments concerning adjusting to new versions of dependencies will not be described in future sprints.

3.5.1 Migrating OasisLib

At the very end of sprint one, the group responsible for *OasisLib* published a complete rewrite of their API, upon which many groups (including us) are dependent. They did this, because the local database layout has been updated to match that of the remote database.

Migration is made difficult since the old API has been removed completely and no documentation exists on how to migrate to the new API. Because emphasis in this sprint is given to creating the prototypes and holding the client meetings, added with the fact that the new API comes without documentation, migration is postponed to late in the sprint.

Since other applications in the GIRAF suite rely on *Launcher* passing on information about the currently logged in user, their development effectively comes to a standstill until our implementation is fixed. Because of this, we soon start to get requests from the other groups on a new version of *Launcher*, as they have already migrated to the new *OasisLib* and local database, see Figure F3-4. This means that older versions of *Launcher* do not support the new local database installed on the devices, resulting in *Launcher* crashing.

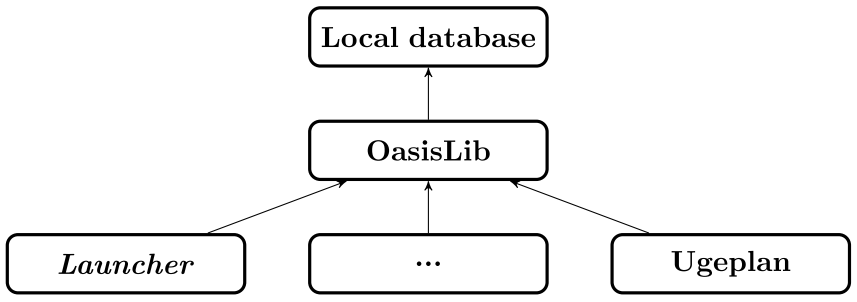


Figure F3-4: Illustrates the role of OasisLib in the GIRAF project.

This adds pressure on us to get the process of migrating to the new version started. Many of the methods in the API have been given new names and thus are not consistent with the existing API. Also, their naming scheme is using both full names and abbreviations, which makes it difficult to distinguish their behaviours. For example, a method to get the package name for an Android application is named `getPack()`, where another is named `getProfile()`. The latter being far more descriptive and easier to deduce its functionality from.

Updating an API

Since this project is developed in Java, one has the possibility to use the Java annotation `@deprecated`. This annotation is used in Java when you are updating an API and you want the developers who are using the API to migrate to the new one. By using this annotation, the compiler or developing environment warns the developer that a deprecated class, method or field is being used.

According to Oracle [16], a valid reason to deprecate the old API includes "it is going away in a future release". The deprecation comments helps the developers to chose when to use the new API and could tell which method should be used instead, see Listing 3.1.

```java
/**
* Get the ID of the currently logged in guardian.
* @return The guardian ID.
*
* @deprecated Should no longer be used.
* This method has been renamed to create consistence in method
   naming.
* One should use getGuardianID instead.
*/
@Deprecated public int getGuardID() {
return mGuardian.getId();
}
```

Listing 3.1: Example of a deprecated method could look like this.

The method of deprecating methods should have been the used method of migrating from the old *OasisLib* API to the new one.

3.6 Sprint Summary

3.6.1 Evaluation

The main task of this sprint has been to develop a prototype design of the *Settings Activity* in *Launcher*. Multiple versions of the prototypes are presented to the clients present at each meeting. They respond positively to the prototypes presented, even though problems are found in communicating our intentions with the designs.

As described in Section 3.5.1 problems arise when *OasisLib* is updated. The group is aware that the migration could have been handled with more grace, because other groups are dependant on *Launcher*. Despite of this fact, the we agree that the group in charge of *OasisLib* should have been more careful and use better software development practices when updating their API.

As the tasks of creating prototypes and holding meetings with the clients have been successfully completed, and the problems solved with new *OasisLib* API, the sprint is considered to be a success.

3.6.2 Backlog

As apparent from Sections 3.3 and 3.4, the clients are in general enthusiastic about the presented prototypes. While smaller adjustments are needed to fit their requirements, many of the suggested features are ready to be designed and implemented.

The requirements specified by the clients can be outlined as follows:

- A profile selection feature available both from *Launcher* and from within each individual application.
- A settings activity for all enabled applications accessible from *Launcher* and inside each individual application.
- Ability to access Google Play from *Launcher* settings activity.
- Ability to add applications from outside the GIRAF suite to *Launcher*.
- Ability to copy settings from one user profile to another within both versions of the settings activity.
- Ability to enable and disable applications on a ‘per user’-basis within *Launcher* settings activity.
- When attempting to exit an application while *Tidstager* runs as an overlay, only that application should be accessible until time runs out.

Note that requirements outside the scope of *Launcher* have been discovered as well. As an example can be mentioned the requirement of *Tidstager* as an overlay for each open application. Therefore these are not considered any further.

The meetings also uncover unnecessary functionality in form of the drawer component no longer being a requirement, along with the ability to change colours of an specific application on a ‘per user’-basis. This discovery also renders our prototype, presented in Figure F3-3d, showing profile information in the drawer, impossible to implement and will thus not be considered any further.

The issue of suppressing the Android buttons seems to be essentially unsolvable. In spite of several hours of research, we have been unable to find a means for suppressing the *Home* and *Multitask* buttons. This intuitively makes sense from a security standpoint, since suppressing these buttons would allow malicious applications to essentially “hijack” the device.

These requirements provide us with a solid backlog for the next sprint.

CHAPTER 4

Third Sprint

4.1 Sprint Overview

The clients' feedback to the prototypes in the second sprint provided vital information about the important parts of the settings, as presented in Section 3.2.1 and summarized in Section 3.6.2.

This sprint is focused on designing and developing these settings. Section 4.2 explains the design choices made. Section 4.3 then describes the implementation of the design, focusing upon the overall structure.

Although the design and the implementation of settings is the main focus in this sprint, we begin by clarifying the reason for disabling the drawer, as specified in Section 3.6.2.

4.2 Analysis and Design

This section is grounded in the analysis and design of how a settings activity can be integrated into *Launcher*, as to obtain valuable information to be considered before its implementation. It is founded in the results gathered from the client meetings, described in Chapter 3.

But first, it will elaborate on a misinterpretation found during the client meetings.

4.2.1 Disabling the Drawer

As described in Sections 3.3 and 3.4, the purpose of the drawer is founded in a misinterpretation. It was thought by the previous groups working with *Launcher* as being a requirement to be able to colour each of the GIRAF applications. However, at the client meetings, it was clarified that no such requirements for the applications exist. What the clients want is to be able to switch the colours of *pictograms*, being able to also show them in black and white. This is due to some citizens handling colours better than black and white and vice versa, as referred to in Section 3.4.2.

Some additional problems are found with the drawer. These are mostly centred upon other applications absorbing the colour given to them by *Launcher*, and using that as their main theme. However, this means that for each possible colour *Launcher* can give an application, it needs to implement a colour scheme that fits with the given colour. This is especially a problem for applications using a wide variety of colours.

Since the colouring functionality of the drawer is not a requirement per se and caused problems in other applications, it is decided to disable the drawer, as it now serves no particular purpose. However, the code for animating the drawer is preserved, in case the functionality at a later time will prove useful. This way, should a future group need to implement the drawer, they merely have to uncomment parts of the code.

4.2.2 Adding Settings to GIRAF

The idea behind providing a way to change settings from *Launcher* for both itself, but also other GIRAF applications, is to create a streamlined and uniform user interface for changing settings across the entire project. This section discusses how to design the architecture of what will be an activity in Android, implemented in the context of *Launcher*, to contain the settings.

Choosing Between Prototypes

As discussed in Section 3.4, the clients agreed that settings should be accessible through two methods:

- A dialogue box in each individual application, containing only settings relevant to that specific application, as seen in Figure F3-3b. The dialogue box should be displayed when the user presses a "settings button" identical throughout all GIRAF applications, making it easier for users to find.
- A unified settings activity, where the user has immediate access to settings for all GIRAF applications, which should be a part of *Launcher*. Besides settings of individual applications, it should also contain settings for *Launcher* itself, and the ability to indicate which applications a user should be allowed access to.

It is immediately apparent that for making both these approaches work in parallel, a single plan for how user settings are handled throughout the GIRAF project is needed. In the following paragraph this plan is described.

Concept It is decided that the settings system should consist of four components, to be implemented separately:

- The unified settings activity should be implemented as a part of *Launcher*, and therefore the implementation of this component naturally falls to ourselves. Excluded from this task is the settings of the individual applications.
- A dialogue box able to contain the settings for an arbitrary GIRAF application. Included in this is a standard button for launching the dialogue box. As this user interface component is useful to all applications, it should be implemented by the group responsible for the *GIRAF Components* library.
- A set of settings for each GIRAF application. As these sets may vary widely in content and complexity from application to application, and furthermore may change along with their associated applications over time, these should be defined and implemented by the individual project groups.
- A method for storing the settings in the database, allowing the users to easily use the same settings across several devices. This should be implemented by the group responsible for the remote database, however, as this group has other priorities, the settings will initially be saved locally on each device.

The challenge of implementing settings, using the above concepts, is how to allow applications other than *Launcher* to access the settings activity, running in the context of *Launcher*.

- The first option is to ask each project group to create a fragment within their own project. The settings activity in *Launcher* should be able to open this fragment. This approach will isolate the responsibilities of each group and keep the overall complexity to a minimum. Although there is an official way for loading a custom class[3], it is complex and it will make the coupling of *Launcher* and other applications very tight. Additionally it is difficult to make the loading of custom classes dynamic, since the name of the class to be loaded has to be known. Each Android application is running as an instance of the Dalvik Virtual Machine that sandboxes the data of the application, making it impossible for others to access it. This method of custom class loading is intended to be used when a project exceeds 64.000 method references, which is the limit of the Dalvik VM.

- To simply the above idea, *Launcher* could instead include all other projects as dependencies when compiling. However, this solution will make *Launcher* project unnecessary bloated and difficult to work with.

- Another possibility is to make the individual applications implement an interface, where a function would add the settings widgets of an application, to a given view. While Android provides facilities for inter-application communication through its *Broadcast* mechanism[9], this requires all the involved applications to be running. This is not an acceptable condition in this situation.

The Final Design

Because of the problems with the designs presented above, another is chosen. An Android feature called Intent Filters allows applications to request an action from another Android component; an activity, a broadcast receiver, a content provider or a service. It is implemented by adding an `<intent-filter>` XML element to the Android Manifest with its actions defined within[11]. With this method it is possible to check each installed application to see if it implements a certain action. This action will be the same for each GIRAF application implementing the intent filter. If an application implements this filter, it can be assumed that it provides an activity for showing settings, which in turn can be added safely to the list in the settings in *Launcher*.

With this design it is not possible to load settings from other applications as fragments from *Launcher*. Instead we start their application to access their settings. However, this simplifies the complexity about saving the settings. If *Launcher* was to load their settings as a fragment, it would load the settings in its own context, thereby saving them in this context. This problem is avoided by this solution.

Designing from Prototypes

Having investigated how to handle settings from other GIRAF applications in Section 4.2.2, this section elaborates on how the activity is designed. The design uses Figure F3-3a as the underlying basis of the decisions.

Settings Activity Analogous to opening a new window on a desktop computer, a “screen” on an Android device consists of an activity. Figure F4-1 can be compared

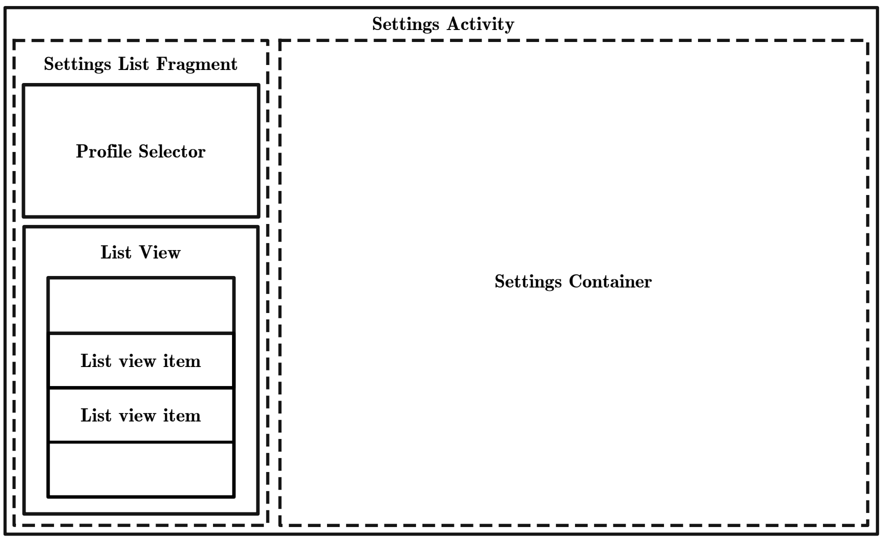


Figure F4-1: The architecture showing how the prototype, Figure F3-3a, will be implemented.

directly to Figure F3-3a, expanding on the user interface components that need be included.

To follow Android guidelines, we are using fragments[10] to implement the functionality this activity requires. They are ideal in this situation, as they are reusable modules which ensure decomposition of the application functionality and user interface. Managing these with a Fragment Manager, it is also possible to add multiple fragments to a screen without switching activity.

By following these guidelines for developing an activity, it is possible to use the same implementation to show the settings on a smaller screen. This will be an advantage, should the clients purchase tablets with smaller screens.

As illustrated in Figure F4-1, the design consists of one fragment, *Settings List Fragment*, and one container, *Settings Container*. The settings list fragment contains the profile selector and a list view as described below:

- **Profile Selector** is what makes it possible to edit settings for multiple users. The prototype previously mentioned shows a profile selector resembling a drop-down menu. To follow the multi-project guidelines, a component developed by the group responsible for *GIRAF Components* will be implemented instead. When the selected profile changes, the settings activity should be reloaded to reflect the settings of the newly selected user.

- **ListView** shows the categories of settings which are available to the user. The fragment attached to each category is presented in the settings container when the category is chosen.

4.2.3 Launcher Settings

Following the prototypes from Figure F3-3, it is clear that a decision has to be taken regarding which settings to include for *Launcher*. According to the client meetings, Sections 3.3 and 3.4, the presented design and its functionality received positive feedback. To build on this idea, the following settings should be included:

- ***Launcher* settings**

Since no critical settings are needed for *Launcher*, it will contain the settings presented in the prototypes, apart from the ability to change the colour of an application, as mentioned in Section 3.4.

- **Application management**
  This part of *Launcher* is more critical as it will be the only way of adding certain applications to a user, whether they be an Android or GIRAF application. Thus, this part has the responsibility of handling all tasks related to application management.

This concludes the design of the settings activity and we subsequently describe the implementation.

4.3 Developments

In the following sections, the architecture for settings illustrated in Figure F4-1, is further elaborated regarding its actual implementation. The container components presented with dashed lines in the figure are explained together with the components they contain.

4.3.1 The Settings Activity

The main responsibility of this class is responding to updates from the components it contains, and also inflating the layout seen in Listing 4.1.

```
<fragment
  android:id="@+id/settingsListFragment"
  android:layout_weight="2"
  class="dk.aau.cs.giraf.launcher.settings.SettingsListFragment"
</fragment>

<FrameLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
  android:id="@+id/settingsContainer"
  android:layout_weight="6">
</FrameLayout>
```

Listing 4.1: Excerpt of the layout defined for `SettingsActivity`.

The layout is fairly simple, since it is not responsible for rendering any dynamic content other than displaying what is to be contained within the activity. Instead it defines two containers. The first is the `<fragment>` element, which is the Android way of defining a static fragment that can not be changed after instantiation. As indicated by the `class` attribute, it is inflating the `SettingsListFragment`, which purpose is explained in Section 4.3.3. The second container is a `FrameLayout`, which contains the settings selected from `SettingsListFragment`. This layout is equivalent to the *Settings Container* illustrated in Figure F3-3a.

Since Android disapproves of inter-fragment communication, `SettingsActivity` will have to stand between and respond to requests send from its active fragments. This is done by implementing the `SettingsListFragmentListener` interface to define methods for what the `SettingsListFragment` wants to communicate, as seen in Listing 4.2. Defining the methods of the interface, no coupling between the fragment and activity exist because the fragment does not know of the activity.

```java
public interface SettingsListFragmentListener {
  public void setActiveFragment(Fragment fragment);
  public void reloadActivity();
  public ArrayList<SettingsListItem> getInstalledSettingsApps();
  public void setCurrentUser(Profile profile);
}
```

Listing 4.2: The interface implemented in `SettingsActivity` defined in `SettingsListFragment`.

4.3.2 Loading Applications

In the application management part of the new `SettingsActivity`, *Launcher* will load applications similar to the way it is done in `HomeActivity`. However, the methods that load applications into a given view are placed in `HomeActivity`. Therefore, it is needed to refactor much of the existing code, so *Launcher* can load applications from any given activity and code redundancy is reduced.

`HomeActivity` is refactored by moving the relevant methods, most prominently the `loadApplications()` method, into `LauncherUtility`. However, `loadApplications()` needs to be changed further to accommodate the different usages in `HomeActivity` and `SettingsActivity`. It is required to pass different `OnClickListners`, since the behaviour is not the same. The application management fragment requires the applications to be marked when pressed, while *Home Activity* requires the applications to start.

Observing New Applications

To increase usability when installing or removing applications on the device, an observer is implemented. It is responsible for keeping an eye on currently installed applications and compare these with those which are shown at that moment. If there e.g. are new applications on the device, which are not in the list of currently shown applications, they are reloaded, by calling the `loadApplications()` method described above.

This functionality is implemented creating a custom timer task by extending the Java class `TimerTask` and scheduling this custom task to run at a fixed interval. Thus, a check is made every fifth second to see if any new applications have been installed. The interval is decided after what we found acceptable by trial and error.

For every execution of the custom task, the list of installed applications is retrieved and compared with the list of all currently shown applications. If the two lists differ, the layout is refilled and the list of currently shown applications is updated.

4.3.3 The Settings Menu

This is the static fragment as defined in the `SettingsActivity` layout from Listing 4.1. The main task of `SettingsListFragment` is to update `SettingsActivity`, using the interface callback member `mCallback`. This is done in the appropriate listeners for the *Profile Selector* and *ListView*, as illustrated in Listing 4.1. In the implementation, these components corresponds to a `GProfileSelector` and a standard Android `ListView` respectively.

To ensure that the context of the underlying activity implements the interface just described, the an exception is thrown if the interface is not implemented.

To enable the list view to render its items, an adapter is instantiated with a list of installed applications and attached to the list view.

Before describing this adapter in Section 4.3.4, the meaning of `SettingsListItem` is explained.

Contents of the Settings Menu

Each item in the settings menu contains an instance of `SettingsListItem`, which is a simple class whose members are used to render the information shown in the list view. Thus, it is used as a placeholder for the fragments and intents that should be opened by clicking a list item. The members of this class are seen in Listing 4.3.

```
String mPackageName;
String mAppName;
String mSummary;
Drawable mAppIcon;
Fragment mAppFragment;
Intent mIntent;
```

Listing 4.3: Members of the `SettingsListItem` class.

By using constructor chaining, the class provides various public constructors that use one private constructor to set the values of its members. Depending on the constructor used, for each item in the list view in `SettingsListFragment`, the item is able to either change the fragment in *Settings Container* from Figure F4-1, or start an activity through an intent. The fragment is changed using the `FragmentManager`, which is used to manage fragment transactions. The list view should also be able to contain an item for settings in other GIRAF applications, therefore the necessity for launching activities through intents. Logic used in `SettingsActivity` checks if either the fragment or intent member is `null`, and carries out the according action, when an item is clicked.

4.3.4 Adapter for the Settings Menu

Considering the prototypes seen in Figures F3-3a and F3-3c, it is apparent that an adapter view for each list item is required. In Android, the easiest way of doing so, is by implementing an `Adapter` for the list view. An adapter serves the objective of bridging the underlying data for a view, and to render it appropriately. Various adapters exist for different purposes. It is e.g. possible to implement an extension of `Adapter`, called `ListAdapter`, which serves the purpose of easily adding items to a list view. Implementing an adapter also has performance related benefits. Most importantly being the concept of *recycling* a view, which makes it possible to handle large lists efficiently[19]. The performance is achieved in the `getView()` method, seen in Listing 4.4. In case of an already instantiated view given as parameter, we can skip its instantiation, thus saving both time and memory.

In the case of yielding a visual result as close as possible to the prototypes, our implementation requires a custom class extending the `BaseAdapter` class. The reason is that the list requires both a title and an icon, but also because of the shadows rendered around a selected item in the list. This would not be possible to handle with a standard adapter.

The most important method of an adapter is its `getView()` method, since it is solely responsible for returning a new `AdapterView` to the underlying `ListView`. This method is seen in Listing 4.4.

```java
public View getView(int position, View convertView, ViewGroup
    parent) {
  View vi = convertView;

  if(convertView == null)
    vi = mInflater.inflate(R.layout.settings_fragment_list_row,
       null);

  // Get current item in the list
  SettingsListItem item = mApplicationList.get(position);

  if (position == mLastSelectedItem) {
    setShadowRightCurrent(vi, false);
    ListView listView = (ListView)parent.findViewById(R.id.
       settingsListView);
    listView.setItemChecked(position, true);
  }
  else {
    setShadowRightCurrent(vi, true);
  }

  if (position == mLastSelectedItem - 1)
    setShadowAboveCurrent(vi, true);
  else
    setShadowAboveCurrent(vi, false);

  if (position == mLastSelectedItem + 1)
    setShadowBelowCurrent(vi, true);
  else
    setShadowBelowCurrent(vi, false);

  ImageView appIcon = (ImageView)vi.findViewById(R.id.
     settingsListAppLogo);
  TextView appNameText = (TextView)vi.findViewById(R.id.
     settingsListAppName);

  // Setting all values in ListView
  appIcon.setBackgroundDrawable(item.mAppIcon);
  appNameText.setText(item.mAppName);

  return vi;
}
```

Listing 4.4: `getView()` method in `SettingsListAdapter` class.

In the `getView()` method the most important parameters are `position` and `convertView`. `position` is the index of the item in the list view that is currently being inflated. `convertView` is the view obtained from the list view. As seen in the example, this view is only inflated if it is not `null`, which is one of the advantages of using an adapter.

To be able to add data to each view contained in `convertView`, the item related to the position is obtained, where a view can e.g. be an image showing an application icon.

The rest of the `getView()` method is spent on checking logic for which shadows to show, based on the last selected item. This logic utilizes three different methods looking much alike. One of these are found in Listing 4.5

```java
private void setShadowRightCurrent(View currentView, boolean
    visible) {
  View rightShadow = currentView.findViewById(R.id.
      settingsListRowShadowRight);
  if (visible)
    rightShadow.setVisibility(View.VISIBLE);
  else
    rightShadow.setVisibility(View.GONE);
}
```

Listing 4.5: One of three methods setting the visibility of shadows related to the selected list item.

This implementation is not considered the “most optimal”, since it brings some code duplication for methods doing the same kind of task. The reason for doing this is that we need a reference to a specific view, depending on the shadow needed to be shown.

The most recently selected item is a global member in `SettingsListAdapter`, made available through the public method `setSelected()`. The method is called from an `OnItemClickListener`, implemented by the underlying list view.

4.3.5 Adding to Settings

The interface implemented by `SettingsActivity` forces it to implement a method called `getInstalledSettingsApps()` that has to return a list consisting of *Launcher* settings, intents available from other GIRAF applications and intents related to the native Android settings. The method returns an `ArrayList<SettingsListItem>` by utilizing a range of methods that all adds an item to the list, whether it be a fragment or intent. This means that each GIRAF application that wants to be available through *Settings Activity* in *Launcher*, has to be added to `getInstalledSettingsApps()`.

To enable another GIRAF application to be shown in settings, it must provide the following intent filter declared for the activity containing its settings, as illustrated in Listing 4.6.

```xml
<intent-filter>
  <action android:name="dk.aau.cs.giraf.cars.SETTINGSACTIVITY"/>
  <category android:name="android.intent.category.DEFAULT"/>
</intent-filter>
```

Listing 4.6: The intent filter and action a GIRAF application has to provide to be shown in settings.

Important to notice is that the name of the provided action within the intent filter, is a key having the value `SETTINGSACTIVITY`, which is appended to the full package name of the application to show settings for.

4.3.6 Settings specific to Launcher

The implementation of the settings for *Launcher*, as described in Section 4.2.3, is elaborated in this section. Since these settings are exclusive to *Launcher* and thus are only possible to open within *Settings Activity* by implementing them as a fragment, they will be available in the *Settings Container* as shown in Figure F4-1.

Launcher Settings

The settings which are directly in connection with *Launcher* are presented in Figure F3-3.

The available settings are:

- Disabling the start animation, when starting *Launcher* the first time after a system restart.
- Changing the size of application icons in *Home Activity*.

These are added as suggestions of relevant *Launcher* settings as outlined in Section 3.2.1.

The most interesting of the two settings is the icon scaling, since "Show start up animation" is created with a standard Android preference component[1], providing the layout with a two-state switch.

The icon scaling is created by extending the standard `Preference` class in Android, thereby inheriting the native setup, i.e. title and summery fields, and constructors for such a component. We supplement it with a custom layout containing a slider and an application icon, as well as providing functionality to react when the slider changes its position.

Application Settings

This section follows up from the design described in Section 4.2.2. When entering the application management fragment, the `AppManagementFragment` is loaded into the settings container, seen in Figure F4-1. The application management fragment in turn contains another container, where the fragments showing the applications are loaded into. Furthermore, `AppManagementFragment` implements three buttons in the top of the layout: the *Giraf*, the *Android*, and the *Butik* buttons.[2]

The *Butik* button opens Google Play, so the user can download additional Android applications, if desired. We open the Play Store as a separate activity with certain flags, ensuring that the user returns to *Launcher* when exiting the Google Play. The exact code can be seen in Listing E.4.

The *Giraf* and the *Android* buttons load a new fragment into `AppsContainer`, by using a `FragmentManager` - the same way it is described in Section 4.3.3. The fragments loaded are derived from the same superclass, called `AppContainerFragment` and is described subsequently.

AppContainerFragment and its Derived Classes

Because the GIRAF and Android fragments contain many of the same variables, these inherit all shared information from a superclass, `AppContainerFragment`, consequently reducing redundancy and clarifying how the two fragments are different. The main responsibilities of the superclass are to initialize shared variables and implement shared methods that handle loading applications.

As a result, the two derived classes get their shared variables initialized by a call to `super.onCreate()` and reloading of applications is handled by calls to `AppContainerFragment`.

[1] "Preferences" in Android terminology is what we refer to as "settings".
[2] "Butik" is Danish for a "store".

However, which types of applications are loaded are different for `AndroidFragment` and `GirafFragment`. This is solved by initializing the shared variable `apps` differently in the two derived classes - `GirafFragment` loads GIRAF applications into `apps`, while `AndroidFragment` loads Android applications into it.

Since the automatic load methods mentioned above work on the `apps` variable, this solution solves the problem.

The other problem is related to the marking of applications. This is due to the fact that GIRAF applications are saved as the *OasisLib* type `Application`, while Android applications are saved as the type `ResolveInfo`.

The GIRAF applications connected to a user are saved or removed through a call to an *OasisLib* method, while Android applications are stored using the preferences system in Android. More about saving settings in Android can be found in Section 4.3.6 and the code for marking the applications can be seen in Appendix E.4

These two problems are the main reasons we need to derive the two subclasses.

Having described all the different types of settings *Launcher* treats, the next sections explains how to save the settings.

Saving Settings

This section explains the functionality that saves preferences when a change is made, by describing the features, which Android provides for managing preferences on a 'per application'-basis, as well as how we use it.

Android provides the class `SharedPreferences`, which is used to handle settings such as ours. This class provides functionality to save preferences in the context of an application by writing to a file on the device.

We must create the files through `SharedPreferences` manually. Creating a unique file name is achieved by combining the role and ID of the current user. This means that a *citizen* profile with the ID *1*, would get the file name c1, prepended with the package name of our application, i.e. `dk.aau.cs.giraf.launcher.c1`.

Preferences are identified by a key, and edited through the `Editor` class, which provide methods to edit preferences. An example of one of these methods is `putBoolean()`, which stores a boolean field in a preference, identified by a provided key. If the preference exists it is updated - if does not, it is created.

When a preference is to be read, it is retrieved by a *get* method, e.g. `getBoolean()`. This method is provided with the key for the preference and a default value in case it has not yet been set.

In our application, we are managing settings using the Android activity and fragment life cycles. In *Settings Activity* we are reading and storing settings when each fragment is shown and hidden, respectively. By doing this, the user will not have to do anything to load the preferences or store changes.

4.4 Sprint Summary

4.4.1 Evaluation

Implementing the settings was the primary task for the third sprint. The task consisted of several subtasks, which have been completed:

- It is possible to scale applications up and down and disable the start up animation.
- It is possible to add and remove both GIRAF and Android applications on a 'per user'-basis.
- It is possible to access Google Play and the native Android settings from *Settings Activity*.
- It is possible to access the settings of other GIRAF applications through *Settings Activity*.

This means the task has been solved and the sprint has been a success.

We decided not to present our progress at this sprint review meeting. While it technically goes against Scrum practice to not demonstrate the current state of the project to the clients at each sprint review, *Settings Activity* in its current state is essentially identical to the prototypes previously presented to the clients. Furthermore the clients do not have the same urge to monitor progress, as paying clients might have.

Sprint review is once again conducted by a member from this group - the details can be found in Section 6.4.4

4.4.2 Backlog

There are some general tasks to be carried over to the next sprint:

- The new profile selector from *GIRAF Components* needs to be implemented.
- The code base should in general be cleaned up and refactored to ensure structural integrity of the application.
- The loading animation when starting *Launcher* statically shows the logo for two seconds and should only be shown for the amount of time it takes to load the needed data.

The main task, however, is to refactor and improve the settings:

- The list view on the left in *Settings Activity* needs to display the chosen tab correctly and proper shadowing should be shown.
- Loading applications in both `GirafFragment` and `AndroidFragment` should display a loading animation to the user and load them in a background thread.
- Installing a new application should update the available ones in `GirafFragment` and `AndroidFragment`.

This concludes the third sprint.

CHAPTER 5

Fourth Sprint

5.1 Sprint Overview

The third sprint left a backlog with a number of issues. These are mostly related to either improving the existing code base or the user experience with settings and *Launcher* in general. Therefore, this fourth sprint is dedicated to continuing developments from the third sprint and making *Launcher* a pleasant program to use.

Since the fourth sprint is the last one of this years development of the project, it is different from the other sprints. First of all, the clients have put a significant emphasis on receiving working applications that do not crash which was also emphasised by semester coordinator Ulrik Nyman in the first sprint, as mentioned in Section 2.1.2. While we cannot guarantee that *Launcher* will not crash during the eights months between the end of this sprint and the beginning of the next, we can improve the code to make *Launcher* as reliable as possible.

After development of the application has been completed, a Usability Test is furthermore carried out to gain insight as to what improvements should be done to *Launcher* next year.

5.2 Analysis and Design

This sprint is dedicated to polishing the application, both in terms of visual effects and code structure. The three main elements requiring explanation are:

- Improve the indication of time consuming application processes to the user.
- Adjusting *Launcher* to the completed *GIRAF Components*.
- Integrating the local database into the *Launcher* project.
- Incorporation of the remote database.

The first element we describe is indicating time consuming processes in a favourable way to the user.

5.2.1 Background Loading

Launcher often needs to carry out time consuming application processes:

- When starting up, all data must be loaded.
- In *Home Activity* and *Settings Activity*, all applications a user has access to need to be displayed as click able views.

While the first case only happens when starting the application, the second case occurs several times while it is running, especially if the user is adjusting settings for many other users.

Because of this frequency, it is favourable to load data in a background task and give the user visual feedback that the device is, in fact, working on getting this data.

Implementation of the first case is described in Section 5.3.5 and the second case in Section 5.3.4.

5.2.2 GIRAF Component widgets

As described in Section 2.2.2, the sidebar of *Launcher* contains four widgets made by the group responsible for *GIRAF Components*. As this is the last sprint, the widgets that work should be enabled and those that are non-functional should be removed. Therefore, the calender and connectivity widgets are removed and the profile selector was replaced with a new version. The implementation of the selector is trivial and is therefore not described in Section 5.3.

5.2.3 Remote Database

In order to make the GIRAF system even more useful for the clients, it is a goal for the semester to implement synchronisation between the local database and the remote database. Synchronising with the remote database concerns *Launcher*, since the local database has been integrated into *Launcher* during this sprint. For clarification on this decision please refer to Section 6.2 and Section 5.3.1 for development details on this matter.

At the moment of writing the synchronisation is only one way. This means that it is only possible for the system to synchronise data from the remote to the local database, not the other way around. This was the priority on a multi-project level, since it meant that the online administration tool could be used to manage the system, and the group responsible the synchronisation did not have time to implement two-way synchronisation.

Activating the synchronisation with the remote database introduced some issues which is described in Section 6.3.

5.3 Developments

The developments for this sprint does not include any new features. Instead, focus is given to the task of restructuring select classes to improve their readability and make their functionality easier to understand. Developments from previous sprints are also refactored.

5.3.1 Integration of the Local Database into Launcher

For an explanation of why the integration of the local database into *Launcher* is conducted, please refer to Section 6.2.

The process of integrating it with *Launcher* is fairly simple. The local database project consists of a single provider. In order to start a provider/service from a dependency, the manifests of the projects have to be merged, moving the provider/service to the dependant manifest file.

5.3.2 Improving the Application Management Fragment

The application management fragment is improved in two ways:

- Choosing a new profile loads the settings of the chosen user. This is done by restarting *Settings Activity* with the new user.
- When having loaded applications in `GirafFragment` and `AndroidFragment` once, selecting the panes subsequently does not cause the fragment to reload applications. This is a huge optimization.

In order to achieve the last improvement of the fragments, we take advantage of the fragment life cycle. This means that when a fragment is left, the operating system marks the fragment as paused, and if it has not been used for some time, the operating system will run its garbage collector and destroy the fragment. However, if you return to the fragment before it is destroyed the views are still visible. Therefore, we simply check if the layout in which the applications are contained, is already instantiated. If this is the case, we use the existing layout, otherwise we instantiate it again.

5.3.3 View Holder Design Pattern

As mentioned in Section 4.3.4, the `getView()` method implemented in `SettingsListAdapter` class, is not considered to be optimal. In this sprint, it is chosen to refactor this method to overcome the deficiencies found, by implementing the *View Holder* design pattern, which is an Android best practice of handling multiple views in an `AdapterView`[19]. It is a way of improving performance and enhance the readability of the code, handling shadows for each list item.

The View Holder design pattern is implemented as a class that holds a reference to a view. The implementation of the class used in `getView()` is shown in Listing 5.1.

```java
private static class ViewHolder {
  public ImageView appIcon;
  public TextView appName;
  public View shadowTop;
  public View shadowBottom;
  public View shadowRight;
}
```

Listing 5.1: The intent filter and action a GIRAF application has to provide to be shown in settings.

The essence of the pattern is to use it when a `convertView` is being inflated. If the `convertView` is `null`, we inflate the view, instantiate a `ViewHolder` and obtain a reference to each of the views it contains. This can be seen in Listing 5.2. To be able to get `holder` back when `getView()` is called again for the same `convertView`, the view is tagged to save a reference to `holder`. If `convertView` is not `null`, it means that it has already been inflated, but has been recycled for performance reasons, and `holder` should be restored. It is restored by retrieving the tag of `convertView`.

```java
ViewHolder holder;

if(convertView == null) {
  convertView = mInflater.inflate(R.layout.
      settings_fragment_list_row, null);
```

```
    holder = new ViewHolder();
    holder.appIcon = (ImageView)convertView.findViewById(R.id.
        settingsListAppLogo);
    holder.appName = (TextView)convertView.findViewById(R.id.
        settingsListAppName);
    holder.summary = (TextView)convertView.findViewById(R.id.
        settingsListSummary);
    holder.shadowTop = convertView.findViewById(R.id.
        settingsListRowShadowBelow);
    holder.shadowBottom = convertView.findViewById(R.id.
        settingsListRowShadowAbove);
    holder.shadowRight = convertView.findViewById(R.id.
        settingsListRowShadowRight);
    convertView.setTag(holder);
}
else
    holder = (ViewHolder)convertView.getTag();
```

Listing 5.2: Excerpt of the refactored `getView()` method in `SettingsListAdapter`.

The use of the View Holder pattern is most noticeable in situations where many list items are handled. Nonetheless, it reduces the amount of calls made to `findViewById()`, which iteratively searches the view hierarchy for the specified ID. Since creation of views is expensive in Android, and considering the many views contained within each `convertView` and the amount of times `getView()` is called, the calls made to `findViewById()` is dramatically reduced[4].

As also mentioned in Section 4.3.4, the initial implementation had problems with duplicate code for setting the visibility of the shadows. This is resolved by using the View Holder pattern, since it makes it possible to always hold a reference to the views. Therefore, the three methods are merged into one, taking a view and whether to show or hide it as input.

5.3.4 Background Loading of Applications

There are three views that get populated by applications in the *Launcher* project:

- The container in `GirafFragment`, populated by GIRAF applications only.
- The container in `AndroidFragment`, populated by Android applications only.
- The main container in `HomeActivity`, populated by both Android and GIRAF applications.

Since all three views need some time to load the applications, it is deemed important to populate them off the UI thread. Furthermore, populating all three views in the same way would also simplify the code. Thus, the class `LoadApplicationTask` is created to load applications. It inherits from `ASyncTask` and must therefore implement three methods:

- `onPreExecute()` is called before the task is carried out and runs on the UI thread.
- `doInBackground()` should contain the task to be carried out and will carry it out in a background thread.

- `onPostExecute()` is called after the task has been carried out and runs on the UI thread.

We override `onPreExecute()` and `onPostExecute()` to start and stop showing a loading animation, respectively.

The method in `LauncherUtility` used to populate a view with applications, `loadApplications()` is deleted, and the contents are moved into `doinBackground()`. Since the views can only be manipulated in the UI thread, we add the applications as views to a list of views, which `onPostExecute()` iterates over and adds to the target layout. The constructor of `LoadApplicationTask` is also adjusted to take as parameter all the information needed to populate a given view.

The observer which is responsible for executing the `LoadApplicationTask`, and is described in Section 4.3.2, must be paused while loading applications and restarted afterwards. This happens in `onPreExecute()` and `onPostExecute()` respectively.

Because of the differences in the three classes which are loading applications, namely `HomeActivity`, `GirafFragment` and `AndroidFragment`, we need to derive a specialized class of `LoadApplicationTask` inside each class. The reasons relate to the responsibility of the three classes:

- *Home Activity* has the responsibility of opening applications when added.
- The two fragments in the application management fragment have the responsibility of marking applications as selected.

Even though the two fragments have the same responsibility, they differ by the fact that the GIRAF fragment handles applications from the GIRAF suite and thereby does changes in the local database. On the other hand, the Android fragment handles selection of other Android applications, and these selections are stored on the device. An example of one of these derived classes is shown in Listing E.5.

To start the task for loading applications, the observer is using a specialised version of the code in Listing 5.3.

```
LoadApplicationTask lat = new LoadApplicationTask(context,
    currentUser, guardianUser, targetView, iconsize,
    onClickListener);
loadApplicationTask.execute(applicationsToLoad);
```

Listing 5.3: The simplified code for loading applications into a view.

The next section handles the improved indication of loading data when starting *Launcher*.

5.3.5 Background Loading During Startup

Implementing the background loader for `MainActivity` is done the same way as when loading applications, described in the previous section, but with minor changes. The same choice of deriving from `ASyncTask` is made and the derived class is called `LoadDataTask`.

Upon starting *Launcher*, a one-second animation is shown, bringing the logo into view, before the activity starts loading data. As a result, in case there is no new data, *Main Activity* is shown for at least one second before launching the next activity.

An `AnimationListener` is attached to the starting animation, which implements `onAnimationEnd()`. Once the animation ends, this function initiates the `LoadDataTask` and calls `loadDataTask.execute()` to make it run.

A text view containing information about the current state is also updated to display a welcome message, a loading message during `doInBackground()`, and finally a ready message, once `onPostExecute()` is called.

When this background loader is implemented, the data to be loaded consists of some test data generated by *OasisLib*. However, just before the deadline of this sprint, the implementation of the communication between the local database on the tablet and the remote one on the server is finished. As a result, we are asked to add one line of code, starting the synchronization between the two databases, to the published version of *Launcher*, instead of only using the local database.

This one line of code introduced some issues, described further in Section 6.3.

5.3.6 Splitting up LauncherUtility

A quick analysis of the code reveals that `LauncherUtility` is now too big, spanning over 900 lines with static methods for many different purposes. By dividing the class into smaller classes, each with a certain purpose, the maintainability, readability, and understandability of the code is increased. Furthermore, many of the existing functions are either simplified or removed due to lack of use or redundancy.

`LauncherUtility` is split into three different classes:

- `LauncherUtility` remains as a class. It now hosts many of the minor functions, that do not warrant their own class, but is used in a variety of activities. Examples are methods getting helper functions from *OasisLib*, information about the current user, session expiration methods, and handling debug mode. The latter is described in Appendix E.1.

- `ApplicationControlUtility` is created. This class contains methods that are used to either return different lists of applications or check if a certain list lives up to a given criteria. These methods are used when sorting which applications should be marked in *Settings Activity* or shown in *Home Activity*.

- `AppViewCreationUtility` is created as well. This class is used for the creation of views that contain an application. The views are of the class `AppImageView`, which extends the image view class and has functionality used in *Settings Activity*, when selecting applications. This class has state field, which indicates if the application is selected or not. It also implements functionality to toggle this state, between selected or not selected, changing the background of the view to indicate that it has been selected.

The two new classes have their own specific purposes, while `LauncherUtility` handles a collection of smaller purposes.

A discussion on how this could have been improved even further, can be read in Section 7.3.

5.3.7 Additional Improvements

Apart from the improvements described in the above sections, many others are made. Some resolve bugs in the code and others improve the visual presentation of *Launcher*.

However, most of these do not warrant their own sections. Therefore, a non-exhaustive list of improvements can be found below:

- When loading applications into a view, the last row often contains fewer applications than the container can accommodate. This is alleviated by adding invisible, empty views next to the real ones, so the last row appears as the previous ones. A calculation error, that accidentally filled the last row with double the amount it could contain, causing all applications to be displayed as significantly smaller views, is corrected.

- Two issues arose when closing the application before a `ASyncTask` was done with `doInBackground()`. The progress bar, which is initiated in `onPreExecute()`, is never destroyed in `onPostExecute()` and persists until the activity is restarted. Furthermore, a text view is referenced in `onPostExecute()`, which no longer exists at that point, causing an exception. Complete removal of all progress bars on start up and adding a null pointer check resolved the two problems.

- The colour of the marking frame used in the application management fragment, along with several other instances of text throughout the application, is changed to fit the orange-coloured theme.

- A small animation is displayed when launching applications from *Home Activity*.

- The padding surrounding the loaded applications is adjusted to be easier on the eye.

This concludes the developments made in the fourth sprint. The following section will present the usability test done after development.

5.4 Usability Test

As the settings component starts coming together, we decided it is time to evaluate the user interface design, we originally presented to the clients in the second sprint, described in Sections 3.3, 3.4 and 3.6.2.

We abstained from testing usability of the original *Launcher* design earlier, as this had already been done by Andersen et al. [1] in 2012.

At this point, *Launcher* is still a relatively simple application, and since time is always an issue, we decided on an informal test and evaluation, along the lines of the *Instant Data Analysis* (IDA) method.

5.4.1 The IDA Method

IDA emphasizes one-day usability test and evaluation sessions, that are meant to be more cost-efficient than the traditional *Video Data Analysis* (VDA) method. In its defining article, Kjeldskov et al. [12] describe how the method should reveal most of the problems also revealed through VDA, but in much less time.

IDA prescribes three players:

- The *test monitor* instructs and observes the test subject during the usability test.

- The *data logger* observes the tests, and records events of interest.

- Following the actual test session, the *facilitator* runs a brainstorming session with the test monitor and the data logger, where the observed problems are discussed, listed, and rated.

IDA test sessions are based on the think-aloud technique, where the test monitor gives the test subject a number tasks to be carried out with the system. The test subject then tries to solve the tasks, while explaining his or her thoughts along the way. This makes it easier for the players to identify problematic areas. The data logger observes the session, and takes notes based on his observations. The facilitator may or may not be present during the test session.

After the test session, the test monitor and the data logger meet with the facilitator for a brainstorming session. The observations are presented to the facilitator, who starts compiling a list of usability problems on a white-board. Aside from controlling the discussion, it is the job of the facilitator to create an organised list of distinct usability problems, by categorising related observations, and eliminating redundancy. Kjeldskov et al. [12] sets the length of this session to one hour.

Finally the facilitator compiles a systematic list of problems, with a short description of each.

5.4.2 Test Sessions

We conducted test sessions with four different test subjects:

- The first subject has been a client to the GIRAF project since it was initiated in 2011, and has tried the various programs several times. She describes herself as "an okay computer user", with some experience from the iOS environment, but comparatively little experience with the Android environment.

- The second subject is also a client to the GIRAF project, but has little experience in using it, as she feels it has not yet been ready for use. She also describes herself as "an okay computer user", with a lot of experience using iOS devices, but little experience with Android.

- The third subject is a school teacher not associated with the GIRAF project, but related to one of the developers. She only has superficial knowledge of GIRAF, but is very used to Android.

- The fourth subject, an optician, is also a relation of one of the developers. As with the third subject, she only knows the GIRAF project superficially, but uses Android devices daily.

5.4.3 Results

The observations from the four tests were compiled into a list of concrete usability problems. We assigned each problem a seriousness rating on the following scale:

- A problem is *critical* if the user is unable to finish their task as a result of it.

- A problem is *serious* if the user is delayed a significant amount of time, before figuring out how to solve the task.

- A problem is *cosmetic* if the user is only delayed for a short amount of time, before figuring out how to solve the task. Cosmetic problems also include issues that did not give the user any problems, but were nonetheless identified as an area of possible improvement.

For most problems, we also present a suggested solution, which may be implemented by any future development team.

Doubt about how the application selection screen works

Problem: Some test subjects had difficulty figuring out how to add applications to a profile's home screen. Some tried dragging the icon. Some figured out how to mark the application as added, by clicking the icon, but were unsure of whether this solved the task.

Rating: Critical.

Suggested solution: It might be more intuitive to have two containers: one with applications not added to the profile, and one with applications added to the profile. Applications are then dragged between the two. This allows the user to go with their initial impulse to drag the icons instead of clicking them. If the containers are named with labels, it should also be easy to figure out the state of an application.

Difficulty finding Google Play

Problem: Some test subjects had difficulty finding the shortcut to Google Play, when asked to install a certain application. They often started with pressing the "Android Settings" tab. Some also correctly chose the "Apps" tab, but then went to the section labelled "Android".

Rating: Serious.

Suggested solution: We propose a renaming of various tabs. The "Android Settings" tab should be named "Device Settings", to lessen association with Android as a software platform. The "GIRAF" and "Android" tabs in the "Apps" screen, should be renamed "GIRAF Apps" and "Android Apps", to make it clearer that these tabs contain application lists, and no settings as such. Furthermore, the "Butik" tab should be resized to match the other two, to make it more visible.

Locating the Back button

Problem: The test subjects with little Android experience, initially had difficulty figuring out how to return to a previous screen. They had to be pointed to the hardware *Back* button. After this initial confusion, the clients had no problem finding the button again.

Rating: Serious.

Suggested solution: Throughout the project period, is has been discussed a number of times to add a software *Back* button to all GIRAF applications. This should completely eliminate any confusion. Based on our observations, however, we believe that it is only a question of getting used to the existence of a hardware button. When users have learned to use the hardware button, a software button will only clutter up the user interfaces. Furthermore, most other Android applications use the hardware button.

Unsure whether changes are saved

Problem: The first times a test subject changed a setting, they were unsure how to continue, because they could not figure out whether the change had been saved, or if they had to do so themselves.

Rating: Serious.

Suggested solutions: We discussed various possibilities for indicating that the changes had been saved, including an on-screen message, a check mark next to the setting, and even a dummy save button. However, it is an Android, and iOS, norm to save changes as they are made. We therefore decided that, as with the *Back* button issue, it is a question of the users getting used to the current system.

Misunderstanding the authentication screen

Problem: Some test subjects did not fully understand how *Authentication Activity* worked, as they tried to keep the camera focused on the QR code, after it had verified the code. This made it difficult for them to press the *Login* button, as they awkwardly tried to keep the tablet in position while doing so.

Rating: Cosmetic.

Suggested solution: The camera feed could be turned off as soon as a valid code has been scanned, maybe even turning into the *Login* button, instead of the button appearing under the feed.

5.5 Sprint Summary

5.5.1 Evaluation

As this is the last sprint of the project, *Launcher* should be completely done and all functionalities should either have been thoroughly tested or disabled. There is a large emphasis from the clients on receiving fully working applications that do not crash, as described in Section 2.1.2. We fulfil this requirement by the time the deadline for *Launcher* is reached.

However, one day before the official sprint review, the groups working with the remote and local databases declared that synchronisation between the two was ready to be implemented.

This is an example of where *Launcher* is ready and our responsibility towards *Launcher* is fulfilled. However, *Launcher* is dependant on the two database groups and as such, on the multi-project level, our responsibility towards having all components of *Launcher* working is not fulfilled, since synchronisation with the remote database was not implemented when we reached our deadline. The details are explained in Section 6.4.5.

5.5.2 Backlog

The criteria for a successful sprint review was to have fully working applications, that do not crash. This meant that some functionality is not added in this sprint, as it has not been tested and debugged and we can therefore not be certain that the application will not crash.

As a result, there are some remaining features to be implemented. Furthermore, there are several ways the existing code can be improved as well. A description of what can be done by the next group working with *Launcher* can be found in Appendix F.

CHAPTER 6

Collaboration

A central element in working with a multi-project such as GIRAF, is the collaboration between the several individual development teams. It is not enough to concentrate on developing individual components, but also necessary ensure that they actually function together. It is impossible to manage that many development teams without instilling a certain amount of formalism and discipline.

This chapter describes how this group has been involved in these aspects of the multi-project.

6.1 Google Analytics

Google Analytics is a tool for gathering statistics on the use of mobile applications and websites. It is developed to help marketers reach their target groups, by analysing how their applications are used.

Most of the possibilites of Google Analytics are not directly relevant to the GIRAF project, however it has a feature which could be of great use. Google Analytics can automatically record uncaught exceptions that occur on the users' devices. This is a useful feature for multiple reasons. The users may not report an application crash to the developers, which prevents the them from addressing the underlying issue. Furthermore, Google Analytics continuously monitors problems with the applications, even during the eight months each year, where no students are assigned to the project. When a new team of students resume the work, they can immediately start addressing issues that may have shown up during this period.

A member of this group initially came up with the idea of integrating Google Analytics into the GIRAF project. He was made responsible for instructing groups in setting up the tool and make sure that it was implemented correctly. He also wrote a tutorial, which was distributed to the other groups.

6.2 Integrating the Local Database in *Launcher*

Half way through the first sprint, semester coordinator Ulrik Nyman, requested that at the end of the semester, the applications should be available on Google Play. This request made all groups in the multi-project realise, that it would not be favourable to have the local database distributed as a stand-alone application, since the GIRAF system as a whole is dependant on the data stored in the local database. The reason for integrating the local database into the *Launcher* project, is that most of the other applications in the GIRAF suite depend on data from *Launcher*. In this way, they are required to be started from *Launcher*.

Section 5.3.1 clarifies the technical aspect of the integration.

6.3 Integrating Database Synchronisation

Late in the fourth sprint, after the formal deadlines described in Section 6.4.2 had passed, the group implementing the remote database declared that it was ready, and that we should attempt to set up synchronisation between the remote and local databases. While this undoubtedly would present new technical problems throughout the system, we chose to move forward for two reasons. First, it would allow us to distribute a huge number of pictograms in the remote database to the clients, making GIRAF significantly more useful to them. Previously they had to create all the needed pictograms locally in *Pictotegner*, the included graphics application. Second, the remote database group had put significant effort into making the database ready before the end of the semester, and we found it unfair to deny them the last step.

As described in Section 5.2.3, this concerned us, as the local database service had been merged into *Launcher*, to ease distribution of the system.

The expected problems did arise, the most major ones including:

- The remote and local database schemas did not fully match. The main issue was that the remote database was implemented with MySQL, while the local database was implemented using SQLite, an Android standard. Differences in the two SQL implementations, gave rise to some problems. Other problems were related to the two schemas being developed by two different groups. While the groups communicated about changes continually through the project period, some last minute redesigns of the remote database schema, had not been added to the local database schema. Identifying and fixing these problems were a collaborative effort between us, the remote database group, and a few other developers present at the time.

- After making necessary changes in the local database to resolve the above-mentioned issues, a series of new problems arose in the applications that saved information in the local database, as these save operations no longer conformed with the schema. Identifying and fixing these problems was mainly the responsibility of the affected groups, but members of our group moved around and helped, as *Launcher* did not require any significant effort at this point, and the final presentation to the costumers was scheduled for later that day. Furthermore, we had some insight into the root problems, from working with the database issues the previous day.

- *Pictosearch* is an application meant to provide an interface for searching through the pictograms in the database. As it provides this central functionality, a number of other applications use *Pictosearch*, rather than providing their own search interface. When we started testing the database synchronisation, we tried to transfer only 100 pictograms to the device, and then view them in *Pictosearch*. It turned out that the application had problems, even with this small number of pictograms, and became highly unstable. The resolution of these issues are described in the next section.

6.3.1 Reworking Pictosearch

As the remote database group and we attempted to add small subsets of pictograms, around 100 from a collection of several thousand, we discovered that *Pictosearch*, the main application for accessing pictograms, had serious issues. The main problem was that scrolling through the collection of pictograms caused an `OutOfMemoryError` and an application crash after a short while. Furthermore, the application showed some performance issues, with the scrolling animation lagging annoyingly.

At the point of these discoveries, it was late afternoon the day before the final presentation to the clients. We discussed the possibility of giving up database synchronisation, and releasing the project to the clients without this feature. But even without the database collection of pictograms, the clients would have the same issues with *Pictosearch* when creating pictogram collections of their own, so the system would be equally unusable. We then discussed to possibility of waiting till the following day, and then ask the group responsible for *Pictosearch* to resolve the problem. But the necessary changes would more then likely cause problems for the applications depending on *Pictosearch*, and there would be too little time to also resolve these issues. Furthermore the group in question had unfortunately earlier proved difficult to contact, and so might not even be present the following day. Therefore the developers present decided to attempt to resolve the issues themselves.

The memory error turned out to be caused by new views being inflated continually, rather then recycling the already inflated ones. The offending method was redesigned, and the error ceased occurring on most of the test devices. It persisted on a few low-memory devices, as *Pictosearch* nonetheless loads are large number of bitmaps into memory. This final problem was solved by increasing the heap size of *Pictosearch*.

The performance issues were not solved, but deemed not serious enough to warrant the same last-minute effort as the out of memory error.

6.4 Sprint End Organization

As we are using the Scrum method of development for the entire GIRAF project, a sprint review was planned for the end of each iteration. Larman [14, p. 71] describes this type of meeting as follows:

> *"At the end of each iteration, there is a review meeting (maximum of four hours) hosted by the Scrum Master. The team, Product Owner, and other stakeholders attend. There is a demo of the product.*
>
> *Feedback and brainstorming on future directions is encouraged, but no commitments are made during the meeting. Later at the next Sprint Planning meeting, stakeholders and the team make commitments."*

6.4.1 The First Sprint Review

At the meeting following the first sprint, the Scrum Master of the GIRAF project started with a short presentation, and then gave the floor to the groups. They spent five minutes each demonstrating their progress to the stakeholders. After the demonstrations, the clients left, and the groups that would not continue working on the same project for the next sprint, proceeded to distribute these projects among them, in preparation for the second sprint.

We do not consider this sprint review a success, due to the following reasons:

- It was a one-way meeting, where the clients were presented with the progress, but were not encouraged to give any feedback. As a result, the development teams had no gain from the meeting, and were ill prepared for the second sprint. This problem was exacerbated by the lack of the sprint planning meetings required in Scrum to start a new sprint. As Larman [14, p. 70] describes, during the sprint planning meetings, goals are re-prioritised and a new sprint backlog is created.

- Several of the demonstrations had technical problems, such as applications crashing in the emulator used for the demonstration. This surprised the presenting groups, and thus they were unable to demonstrate their actual progress. A related problem was that several groups did not direct their presentation towards the clients, and therefore focused on technical details, using a very technical vocabulary. The demonstration should be better prepared, with more focus on the actual goal of the meeting.

- Not all clients were able to attend, as the meeting was announced at a relative late date beforehand. As the main goal of the meeting is for the clients to review the progress of the project, this is not optimal.

These problems were discussed after the conclusion of the meeting. It was suggested that a possible root problem was the Scrum Master being overworked with planning this meeting along with his other responsibilities. It was decided to, at a later date, discuss the possibility of designating another person as responsible for planning and conducting the sprint reviews, and possibly the sprint planning meetings as well. Eventually this role was given to a member of this project group.

6.4.2 Sprint Review Planning

To counter the above-mentioned points, the new sprint end specialist wrote a directive on how sprint reviews should be performed. Main points in this document include:

- The clients should be encouraged to give feedback continually during the meeting, hopefully giving the groups an idea of where their projects are still lacking. Furthermore, the second half of the meeting will consist of a planning phase, where the clients have the opportunity to re-prioritise the goals of the upcoming tasks, or add goals if something important has come up during the meeting. Scrum suggests building the new sprint backlog from scratch in cooperation with the clients. We however, see no reason to keep nine clients and 60 developers in a room for several hours to discuss the sprint backlog, which has to be large enough to keep every developer working throughout the upcoming sprint. Instead, one developer will compile a preliminary suggestion from the backlog, based on what each group think is the next step for their project. We then give the clients the opportunity to influence this backlog.

- The groups are given deadlines for releasing their projects in the versions to be used for the demonstration. Dependency projects are to be released to the other groups at least three working days before the sprint review. This gives the other groups a chance to adapt their own projects to the new dependencies. Groups who wish to demonstrate their product to the clients, are to release their final versions at least one working day in advance. This gives the sprint end specialist a chance to collect the applications on a tablet, and let the groups test whether their application works as expected on this tablet.

- To keep the demonstrations on a level that the clients understand, the groups are asked to base their demonstrations on user stories along the lines of “You are now able to ... by doing ...”. This should prevent the groups from becoming technical in their presentations, and from discussing topics irrelevant to the clients, such as how many hours were spent debugging.

- The dates of the sprint reviews are advertised to clients well in advance, with a program for the upcoming meeting being sent out at least a week in advance, which should ensure that as many clients as possible are able to attend.

Install Parties

At each sprint review, the clients can bring their Android devices and have the installed GIRAF applications updated to the latest versions. This is not seen as being an efficient method of distributing updates. As mentioned in Section 6.2, the applications should be available through Google Play, since this is the optimal method to distribute applications to Android devices. If the applications are distributed through Google Play, the users will automatically get the updates installed on their devices.

6.4.3 The Second Sprint Review

It is difficult to evaluate on the second meeting, as the clients left during a break, due to a misunderstanding. Furthermore, the sprint end specialist was unable to participate in neither the preparations, nor the meeting itself, so no detailed evaluation of the meeting was conducted.

6.4.4 The Third Sprint Review

The third meeting was the first to be executed fully in accordance with the new meeting directive. In the final days before the meeting, the sprint end specialist spent significant amounts of time checking that the various groups finished their demonstration versions, and that the groups tested their applications, in what can be described as an informal integration and system test.

During the meeting, the sprint end specialist tried to involve the clients in discussions, by asking them open-ended questions. This achieved the affect of making the clients more open to present their thoughts on the various applications. The presentations were much more minded towards the clients, and at a later evaluation of the meeting, it was agreed that the only problem was the presentation skills of some groups, who had some issues with mumbling etc.

The planning phase could be characterised by limited success, as the clients for the most part accepted the suggested backlogs with no comments. A single project had more task suggestions than the group would be able to finish, and the clients were asked directly to opt one out, which they hesitantly did. The problem may have been that they felt awkward telling the groups what to do, and what not to do.

6.4.5 The Fourth Sprint Review Meeting

The fourth meeting was somewhat different, as it marked the end of the last sprint, and thus no further developments would be made afterwards. Therefore, the time was spent with presenting all the developed applications, while the sprint planning phase was left out.

The preparations for this meeting became somewhat more chaotic, as it was decided to try making synchronisation between the remote and local databases work. While all deadlines had passed, the day before the meeting, the remote database was declared ready for this step. The issues and decisions regarding this are described in detail in Section 6.3.

6.5 Informal Collaboration

The internal dependencies between the various components of GIRAF, necessitated several instances of ad hoc collaborations.

Launcher depends on other projects and vice versa, which caused some of unfortunate errors in both *Launcher* and other applications. In this section, the most interesting of these collaborations are described.

One of the most interesting and recurring causes of errors, which cascaded through the GIRAF project, was corrections or additions to the *OasisLib* project. The reasons for this, is that all applications use this library and the groups had no plan of when to update their version of *OasisLib*. This meant that the groups updated these dependencies independently of each other. Since either the other groups still had not updated their dependencies or we had not updated ours, several problems arose.

An example of a change cascading through the project caused by *OasisLib*, is a change where the type of an application ID was changed from a `long` to an `int`. When we updated our *OasisLib* we did not notice this change, as we store the `int` value into a variable of type `long`. Since we are passing this value to the other applications, it created errors in them. This caused a `ClassCastException` in the other applications, since it is not possible to cast a `long` to an `int` without loosing precision. We were notified when multiple groups suddenly started knocking our door.

Solving these problems required cooperation between the different groups.

CHAPTER 7

Evaluation

Overall, we believe *Launcher* fulfils the requirements of being a running and working application, when the work was concluded at the end of the last sprint. We also believe the collective GIRAF suite fulfils this requirement. There are still issues of reliability, as well as ample opportunity for extending the existing features and adding new ones. However, especially with the addition of the remote database and its huge collection of pictograms, GIRAF is very close to becoming a truly useful tool at the client institutions.

7.1 Multi-project

Reflecting on the multi-project work flow, a number of things could have been improved, and a few things should be retained by later teams. The most important ones are described here.

7.1.1 Use of Scrum

The use of Scrum on the multi-project level did not work out well. While Scrum, as described by Schwaber and Sutherland [18], requires that at least two roles are filled, the Scrum Master and the Product Owner, only a Scrum Master was appointed.

The Product Owner functions as a representative of the clients' interests, and most importantly manages the product backlog. The Product Owner is therefore vital to keeping development focused, according the needs of the clients.

The Scrum Master's task is to facilitate the work of the development team. He solves problems the team cannot solve itself, and handles communication with the rest of the organisation. He also coaches the developers in improved organisation and work flow. During this iteration of the GIRAF project, the Scrum Master functioned most of all as a coordinator, as described in Section 1.3.3. It may have given an overall better work flow, if the Scrum Master had focused on improving the processes, and enforcing these processes, as to prevent the groups from taking more disorganised approaches to development.

7.1.2 Intergroup Communication

Considering how used the involved groups are to working with single-team projects, the communication between the different groups worked very well.

While a number of formal communication channels were set up early in the project, most importantly the project management tool Redmine, the groups quickly learned to simply visit each other's group rooms, when the need to discuss a common issue arose. This form of communication is the reason why the informal collaboration, described in Section 6.5, worked so well. It furthermore helped preventing a "them and us"-mentality between the groups, and instead helped create a notion of shared responsibility and community.

7.2 Our Work Process

While it was decided to use Scrum at the multi-project level, the individual groups were free to use other methods. We chose to adopt Scrum as well.

The morning Scrums provided a great overview of the tasks to be carried out during the day. Furthermore, the sprint backlog compiled after each sprint, left us with a good overview of tasks to be completed during each sprint.

However, we deviated from Scrum in several ways. First of all, we did not use a Scrum Master. Since our group only consisted of four member, it was deemed to be unnecessary. Second, though we did have sprint backlogs, they were not as detailed as they should be. This also lead to many tasks appearing during sprints, which were not accounted for in the sprint backlog, and slowed us down. Lastly, the act of estimating the time each task would require, was mostly omitted. While Larman [14] points out that this is one of the more difficult parts of Scrum, we rarely even made an attempt to do so.

Despite these deviations, it is our general consensus, that the process worked well, probably because a group of this small size, generally needs less formal rituals.

The next section discusses further improvements that could be made to the code of *Launcher*.

7.3 Discussion

This section discusses the development of *Launcher*, focusing upon elements that could have been done better.

First of all, while a class called `Constants` does exist, there are still places in the code where "magic numbers" are used. A good example is seen in `HomeActivity` where instantiating a new `TranslateAnimation` to animate a view from on position to another. The constructor `TranslateAnimation(0, to, 0, 0)` does not give any indication to what the 0 mean for each parameter. Instead, constants defined as a members of the class should have been used: `new TranslateAnimation(FROM_X, to, FROM_Y, TO_Y)`, to make the code more readable.

Generally, this implies that a good code style should have been established before beginning any developments.

Currently, three utility classes exist, each containing between 100 and 300 lines of code. The common practice when programming in OO-languages, is to contain methods within classes that use them, and generally not create utility classes. An example of how this could have been done is to have an `AppLayout` class, containing all methods related to populating the layout with applications, which should then be used in `HomeActivity` and `SettingsActivity` classes.

Post development, it is concluded that better decisions regarding the general best practices for Android development should have been researched further. The reason is that the implementation of `AppViewCreationUtility` is considered inefficient in the way views are handled. An optimal implementation would extend `GridView` instead.

This would simplify the implementation drastically, since it handles spacing between the views which currently is half of the current implementation.

Reflecting on this, the existing code for handling loading applications into view should have been replaced by another implementation, utilizing an adapter as done with `SettingsListAdapter`, Section 4.3.4.

Furthermore, testing should have been done more thoroughly regarding developments from both the last group working with *Launcher*, but also our new additions. While functional testing were done during the first sprint, mentioned in Section 2.4, and informal integration testing before each sprint review, mentioned in Section 6.4.4, neither system- nor unit tests were explored. As described in Section 6.3, problems were found regarding to activating the remote database, which indicates that the informal integration testing should have been replaced by a fully formal integration test.

Lastly, a code review session with another group would have a positive effect on the code.

7.4 A Summary of Future Works

We found several candidates for improvements of *Launcher*. In this section, we only describe the three we deem to have the highest priority. An exhaustive list of improvements can be found in Appendix F.

Copying settings from one user to another has not been accomplished due to time constraints. The clients requested the possibility to copy settings from one user to another, this is described in Section 3.4. Since this is an actual requirement, we strongly recommend this to be implemented as soon as possible.

Downloading pictograms while the tablet is being used, is a feature which we find very important. Currently, when *Launcher* is started after a system restart, the synchronisation of data with the remote database is executed. This synchronisation can last up to 30 minutes, during which the user has to wait. It would be ideal to only download the most essential data, such as profiles and their relations, and thereafter give the user the possibility to login. *Launcher* could then continue downloading other data in the background while the tablet is being used.

The loading of applications into views is suggested to be reimplemented. The layout in which applications are being loaded, is currently a linear layout where another linear layout is added acting as a row. This results in multiple calculations of padding around each application, depending on the chosen icon size.

Instead, the mechanism should be implemented using a grid view. This makes it possible to add the applications using an adapter and the spacing around the applications would be set automatically.

This increases performance as described in Section 4.3.4, and would simplify the algorithm used to add the applications to a view significantly.

APPENDIX A

Specification Requirements and User Stories

This appendix includes the complete specification requirements and user stories developed during the beginning of the first sprint by one of the multi-project groups.

The original document is included such that four pages are inserted on a single page in the appendix and should be read in a horizontal top-down manner.

Specification requirements 1

1.0.1 Requirement Prioritization

The requirements in this section is split into the different application they belong to. Each individual requirement has a number in end, which shows the priority of this specific requirement.
(1) - Requirements that has the number (1) is a requirements that must be fulfilled.
(2) - Requirements that has the number (2) is a requirements that must be fulfilled, but which are not essential.
(3) - Requirements that has the number (3) is a requirements that must be fulfilled, but only if there is additional time available.

Profiles

- There is two types of profiles:
  They are divided into organizations, this means that a profile from one organization does not have access to another organization's profiles. (1) See user story number 6.
  - Guardian profile:
    - Has access to all functions in all applications. (1)
    - Can give authorization to which functions in the applications, a citizen profile has access to. (1)
    - Can add new guardian- and citizen profiles on both tablets and web. (1)
    - Can edit existing profiles, both guardian- and citizen profiles. (1)
    - Can remove existing profiles, both guardian- and citizen profiles. (1)
  - Citizen profile:
    - Has access to functions and applications, which are authorized by a guardian profile. (1)
    - As standard, citizen profiles does not have access to any functions. (1)
    - Cannot add new profiles. (1)
    - Citizen profiles can never use the three androids buttons (back, home and menu). (1)
- It must be possible to transfer a citizen profile to another organization, i.e. that a citizen profile's settings and private pictures transfers to another organization so that they can be used there. (3)
- Shifting between profiles requires approval, e.g. by the use of QR code. (3)

General

- No applications must close, stop or crash unexpectedly. (1)
- All applications must synchronize with the global database, once a day, if there is access to the Internet. (1)
  - It must be possible to synchronize manually with the global database. (2)
- All applications must be able to save settings for each individual profiles. (1)
  - It must be possible to copy settings from one profile to another. (2)
- All applications must run in landscape mode (horizontal). (1)
  - Sequences and Week Schedule must be able to use portrait mode. (1)
- All applications, except the Launcher, must have a 'close' button. (2)
  - When the button is pressed, it must close the application, if the user has permission to close applications. (1)
  - If the button is pressed and the user does not have access to close the application, the application must be able to close with the help of a guardian, e.g. by scanning QR code. (1)
- When and only when there is a permitted input, there must be some kind of response, which makes sense in context, but otherwise this is up to the individual developer. (2) see user story number 4.
  - Applications with functions targeted towards citizens must only use singleclick, drag and drop, and swipe input. (2)
  - If there is a function, which should only be usable by a guardian, then all Android gestures are permitted. (2)
  - All applications must be named with meaningful Danish names. (1)
- If a pictogram is not yet saved to the global database, then all applications must show a 'pending' icon, so the guardian can see that it is not yet saved to the database. (2) see user story number 8.
  - Pictograms must be able to be marked as private, so they are not uploaded to the global database. (2)
- All applications must be able to play a pictogram's associated sound. (2)

General GUI

- The general color scheme must be black and white (3)
  - It must be possible to choose a color theme for each application. (2) see user story number 3.
  - Each profiles must be able to have their own chosen color theme. (2) see user story number 3.
- All applications must use the same GUI components. (2)
- All applications must indicate if there is a sound attached to a pictogram. (2)

Database

- Pictograms, photos, sound clips, profiles and categories must be able to be saved in the database. (1)
  - Citizen profiles are split into groups. (1)
  - Profiles must be split into organizations. (1)
  - Pictograms and photos must be handled the same way in the database. (1)
  - Sound clips must be handled separately. (1)
  - Categories must be handled separately. (1)
- Global database:
  - It must be possible to synchronize the local database with the global database. (1)
  - Must contain all standard pictograms. (2)
  - Guardian profiles must be able to add new pictograms to the database. (1)
  - Guardian profiles must be able to remove and edit pictograms they have added to the database. (2)
  - Guardian profiles must be able to copy standard pictorgrams, edit them and then save the copy to the database. (2)
  - Guardian profiles must not be able to remove standard pictograms. (3)
- Local database:
  - All contents of the local database must be able to be saved in the global database, this includes settings. (1)
  - It must be possible to synchronize with the global database. (1)
    - It must be possible to synchronize manually. (1)
      - There must be a synchronize button for the guardian in all applications. (2)
    - It must be possible for the synchronization with the global database to happen automatically. (2)
      - As standard, automatic synchronization should happen once a day when there is wifi. (2)
  - All applications must use the same local database, e.g. different application does not have their own local database. (2)

Launcher

- The launcher must change the functionality of the three android buttons (back, home and menu), depending on the user's permissions. (1)
- The launcher must handle switch of color themes for each application. (3)
  - If you are logged in as a guardian, it must be possible to change launcher. (1)
  - If you are logged in as a citizen, it must not be possible to change the launcher. (1)
- The launcher must keep track of which applications should be visible. (2)

Pictocreator

- Must be able to use the camera to create a pictogram. (1) see user story 26.
- Must have a function to draw, save and edit pictograms. (2) see user story 8 and user story 6.
- Must have a function to record, save and edit sound clips for a pictogram. (3) see user story 8 and user story 6.
- It must be possible to mark a pictogram as private, so that it will not be saved to the global database. (2)
- It must be possible for a pictogram to have a title, which can be freely placed on the pictogram. (2)

Pictosearch

- Must be able to delete pictograms. (1)
- Must be able to show all pictograms, in their specific categories. (1)
- Must be able to search through pictograms. (1)

Categorizer

- Must be able to create new categories of pictograms. (1)
- Must be able to view, add and remove pictograms from existing categories. (1)
- Must be able to delete existing categories. (2)
  - When a category is deleted, a warning dialog box must be shown asking the guardian if they are sure they want the category to be deleted. (2)
- Guardian profiles must be able to connect a specific category to a specific citizen profile. (1)
  - It must be possible to copy a category from one profile to another. (2)

Communication Tool

- Must be able to view pictograms.
- Must be able to play the sound of a corresponding pictogram, when the pictogram is pressed. (1)

Sequences

- Citizen profiles must be able mark how far they are in a sequence. (1) see user story number 14
- The guardian can change whether sequences are shown horizontally or vertically for the citizens. (2)
- Guardian profiles must be able to view, create, edit and remove sequences. (1)
- There must be multiple ways to view sequences. (1)

  - It must be able to show one pictogram, three pictograms or a complete sequence. (1)
- It must be possible for a guardian profile to authorize a citizen profile to be able to make sequences. (2)

Life Stories/Week Schedule

- Citizen profiles must be able to mark how far in a sequence they are. (1) see user story number 14.
- Colors for the week schedule must follow the international color standard for days. (1)
  - These colors must always be viewable, even if you have scrolled down in a sequence. (3)
- It must be possible to turn on a feature, such that days are switched upon swiping to either side, when you are in the day tasks. (1)
- This application must function as a tool to plan daily activities for citizens. (1)
- This application must be able to help citizens to formulate their day in pictograms. (2)
- This application must be able to allocate periods where citizens can choose between preselected activities. (1) see user story number 11.
- Guardian profiles must be able to define how many pictograms are shown at a time. (2) see user story number 18.
  - It must be able show one pictogram, three pictograms or a complete day. (2)
- Sequences of pictograms must have the same size and be locked in a grid. (1)
- When a citizen has a choice to make, other functions must be locked, until a choice has been made. (2)

Timer

- This application must be able to reset, edit, start and stop the timer. (1)
- This application must be able to make the timer visible in all other applications that the citizen profile has access to. (2)
- Citizen profiles can only see the time. (2)
- The timer must always count down. (1)
- When the time has run out, there must be a notification and a sound, that gives the user two options (1)
  - It must be able to move on to the next planned activity (this function must be able to be turned off). (2)
  - A button that stops the timer, but which requires a guardian to continue. (1)

  - A notification must have a short iconic/recognizable sound, which indicates that the time has passed. (1)
    - There must be a possibilities for a guardian profile to choose which sound that is used as the notification. (1)
    - One of the possibilities for the sound must be the sound of a kitchen timer. (1)
  - A notification must contain a pictogram, that explains to the citizen that the time has passed. (1)
- The list of buttons must be able to be listed horizontally or vertically. (1)
- There must be a representation of the Product Owners' timers. (1)
  - The clock face must always represent 1 hour. (1)
  - The time indicator (that which shows how much time there is left) is pulled from 0 and is pulled clockwise. (1)
  - When the timer is counting down, it will go counterclockwise. (1)
- There must be three alternatives for their timers. (3)
  - Hourglass. (3)
  - Progress bar. (3)
  - Digital watch. (3)

Train

See user story number 30 og 32.

- A train leaves the main station after it have been loaded with pictograms and drives to one or more stations where some of the pictograms will need to be unloaded before it reaches the train depot at the end. (2)
  - A pictogram must be unloaded on a station with a category the pictogram belongs to. (2)
  - The train cannot leave a station before all the right pictograms are unloaded from the train. (2)
- It must vary how long a train takes to reach a station. (2)
- The picture of the driver must be the picture associated with the citizen profile. (2)
- There must be instructions for the game. (2)
  - In the beginning of the game, there must be instructions with arrows that indicate that a pictogram should be dragged down into a cart. (2)
- Carts must be named. (2)
- The train must be able to change colors. (2)

Cars

See user story number 31.

- You must be able to steer the car by the use of your voice volume. (1)
- While driving there must be obstacles that is avoided by changing the volume of your voice. (1)

New Entertaining Learning Tools (Not requirements)

- Barrels, a game where a barrel is shown, where possibly a sound comes from it, and the child will have to guess what is in the barrel. (3)
- PuzzleStory, a game where a collection of specific pictograms are shown, where the child then has to organize these into a coherent story. (3)
- RepeatStory, a story where repetition is important. A good example of this is the song "12 Days of Christmas", where each new day repeats the previous days. (3)

The Product Owner suggests that we can be creative and mix these ideas together with existing games. Such as when you have completed a game of Train, a barrel will appear as a fun surprise for the child.

User Stories 2

We have used the user stories from the earlier years as a foundation to create user stories for this semester. The user stories will be used to describe the application of the system in a common language which the stakeholders can understand and relate to. The stories can then be used as guidelines when developing the specification requirements, which is a more formal way the developers can base their design on. The user stories will also be used to check that all the necessary specification requirements are described to develop the solution the stakeholder seeks.

1. Adding profiles
   - A new citizen has started at the institute so as a guardian I want to able to create a new autistic profile for that citizen.
2. Deleting profiles
   - A citizen is no longer part of the institute so I as a guardian want to be able to remove his profile from the system. I want to be able to remove his personal pictograms from the system, but be able to save them to a local storage device if needed before they are deleted.
3. Changing color scheme
   - A new profile for citizen does not have the appropriate color scheme. As a guardian I want to change the color scheme to a color that suits the specific citizen. I select the citizens profile, selects the desired color for scheme of an application and drags it over the application. I expect that the new color scheme for the given application is store for the given citizen profile.
4. Application responding to the user
   - A citizen is playing one of the learning games in the system. As the citizen I expect the application to respond to the input I give the system so I see that something is happening. This should be general for the whole system.
5. Opening an application through the launcher
   - As a guardian or citizen I want to work with many of the applications from the system. I open the launcher and select the application I want to work with. When I am done, I go back to launcher and simply select the next application I want to work with.
6. Creating a pictogram

   - As a guardian I want to create a pictogram to describe an action. I open the pictogram creator, where I draw the pictogram as want it. I add a caption and record myself pronouncing the action. I then assign it into a suited category and saves the pictogram. I expect that the pictogram will be accessible for other guardians from the institute I work as well as myself.
7. Deleting a pictogram
   - A citizen has a personal pictogram that is no longer needed, so I as a guardian want to remove it from the system.
8. Editing an existing pictogram
   - A pictograms caption does not fit and it needs to be changed. The pictogram also have sound attached. As a guardian I select the specific pictrogram and open it in pictogram creator. I now replaces the caption with a new, record a suitable sound and stores the edited pictogram. I expect that the changes to the pictogram now are store and is changed for all application using this specific pictogram. I also expects an indication of whether a pictograms changes have been uploaded to the database or if it still to be done.
9. Adding a new pictogram to a specific category
   - As a guardian I am in the categorizer application and I notice that there is not a pictogram for the activity I need. So I press the button for adding a new pictogram to category and I am sent to the pictocreater so I can either take a photo or draw my own pictogram. When I am done I save it and I am returned to the categorizer application where I can choose the new pictogram. I expect that the newly created pictogram is not accessible to any other institutions except my own.
10. Removing a pictogram for a specific category
    - As a guardian I have accidentally added a pictogram to a category it did not belong to, so I want to be able to remove it from that category.
11. Planning activities
    - A citizen needs a schedule to tell him what he is doing a specific day. I as a guardian want an application that lets me put together a schedule that I can attach to the autistic profile. When making the schedule I make a sequence of pictograms one for each activity, though I can also make an activity free. A free activity is an activity where the citizen has a choice between multiple activities. Each activity can lead to a stored sequence describing how to complete that activity.
12. Keeping track of the schedule
    - As a citizen I look at the schedule to see the activities planned for the day. I check the color of each day to find which day I have to look at in the schedule. I then mark the activity that have been done and move on to the next activity.
13. Creating a new sequence

    - A new citizen is starting at the institute. As a guardian I want to create a new sequence for this citizen because the stored sequences does not fit this specific citizen. I can then use this new sequence to explain activities for the citizen. I expect that the newly created sequence is stored and accessible only to my institute.
14. Mark actions as complete
    - As a child I want to be able to mark actions as completed so I can keep track of my current progress in a sequence. There should be multiple options for how to mark actions as complete so I can choose one I find appropriate.
15. Different views of sequences
    - A child cannot always understand if there are to many sequences in a week schedule, and a child sometimes pick another sequences instead of the one that was meant to be. Therefore the child want to have as few as possible sequences, and it is possible as a citizen to create sequences for month, week, day or simply a half day schedule.
16. Edit a sequences title and images
    - A citizen would like a new brand of cereal instead of the old brand for breakfast. As a guardian I want to open the old breakfast sequence and replace a specific pictogram with the new pictogram. Then I want to save the sequence with a new title so I still have access to both. I can then help the new citizen with breakfast by using the new sequence while my colleagues can still access the old sequence.
17. Deleting a sequence
    - A specific sequence is no longer necessary in the system. As a guardian I select a sequence and delete it. I expect that the deleted sequence is removed from the system.
18. Opening a sequence
    - The entire institute is going for a walk. As a guardian I want to show a sequence for a citizen so I can help them with the correct order of actions for putting on outdoor clothes. I expect the application to provide the possible to view the sequences displaying all, 3 or 1 pictogram in sequence, so I can help the citizen.
19. Helping a citizen communicate
    - As a citizen I can have a difficult time speaking, so I want an application that can play the sound of a pictogram when I press it.
20. Using the timer application
    - A citizen needs to do an activity for a set amount of time. As a guardian I open the timer application and allocate the appropriate time. I select a suited visual representation of the timer and start it. I expect the timer application to show the representation for the allocated time without the screen turning off. When the time runs out I expect the timer application to notify with a sound that the time has passed and the citizen shall move on to the next activity.

21. One application available in another
    - A citizen has a free time slot and wish to play the Train application. As a guardian I want to able to set a timer so the citizen knows how long he can play. I want the timer to be visible in the Train application, so he always know exactly how long he can play. This should be general for the entertainment and learning applications.
22. Creating a category
    - A large collection of pictograms have been added to the system. As a guardian I want to create suitable categories so can I can add these pictograms to the category I deem they belong to. I expect that the created categories persist until they are deleted. I also want to be able to link the created category to specific profiles.
23. Deleting a category
    - As a guardian I have accidentally made a category that will not be used, so I want to able to delete it.
24. Viewing the collection of categories
    - When I as a guardian enter the categorizer application I expect to only see the categories associated with my profile or the autistic profile I am currently helping. I also expect to be able to press a button to see all the categories and pictograms that are available to me. When looking at all the categories I expect I am able to see what each category contains.
25. At the end of the day
    - When a citizen gets home they want to tell their parents about their day. As a guardian I want to able to help them make a life story of their day. I make a template for them where I access the categorizer and choose a set of pictograms for each activity they have done this day. When the citizen then use this template to create their life story they can choose between the pictograms and place them as they like. When they get home they can show off their life story on a tablet.
26. Take a photograph and use it as a pictogram
    - Some citizens need a real photo before they can understand the meaning of a pictogram and link it to the real world. As a guardian I want to be able to open the application for creating pictograms, take a photo and use it to create a new pictogram. I go to the sequence application and use the newly created pictogram in a sequence.
27. Changing to another citizens profile in a given application
    - As a guardian I want to change the settings of some citizen profiles in a given application. I open the list of profiles and select the profile which settings I want to change. I expect that all the settings appears as they are for a given profile and changes to these settings will be stored in place of the old settings. I adjust the settings for all the profiles that I wanted to change and then return to the given application.

28. Remember what I do
    - As a guardian I expect that anything I do or change will be stored so I can use it on any tablet I choose when I am logged in on my profile.
29. Safe navigation
    - As a citizen I want to be in an application mode where I cannot accidentally navigate to parts of the system I am not supposed to use, so I do not get lost or loose focus on the task I should do.
30. Training citizens sequences view
    - For helping citizens in a fun way they use the train application to understand how they can place something in a sequences. They are using the train and have to drop the things at the right place.
31. Training a citizens voice
    - Some citizens cannot control the level of their voice. As a guardian I open the training game to help the citizen control their voice and give them an understanding of how to use it. I expect the game to use the citizens voice to achieve some kind of goal so they find it entertaining while using it and thereby help motivate them to complete the training.
32. Training a citizens understanding of pictograms
    - As a guardian I want to train a citizens understanding of pictograms and their relation to each other. I open the learning game application for understanding pictograms, if this is not the first time then I expect that the selection of categories is the same as from last session. I can edit the list of categories that should be present in the game and start the game. I expect the game to arrange a selection of pictograms from each selected category and choose one category which some of the pictograms are associated with. The citizen then tries to find the pictogram associate with the chosen category.

APPENDIX B

Client Meeting 01-04-2014

Launcher: The point of *Launcher* is to foreclose the citizens from the rest of the tablets functionality. It should be as simple as possible. Pernille thinks it is great that one can change the profile both in *Launcher* and inside each individual app. It is nice that it is the same icon to change profiles everywhere. There are institutions where everyone have their own tablet and others where the citizens must share. Pernille thinks that *Launcher* has the nicest design demonstrated today and that the presentation of how it could work has been very good. Settings should be individual for each citizen. Pernille thinks it would be realistic to adjust settings inside each app. Drazenko just wants it to work. He believes it provides the best overview to adjust the settings inside each app as well. However, it is very different how technically adept a citizen is. "Generelt" is a bad choice of words under the settings for GIRAF. Pernille wants to know why there is an ability to change colours of apps. Drazenko thinks it is ok, but not an important feature. It is probably a good idea to discuss with Birken and bostedet what they meant with their colour requirements. Generally, people are leaning towards adjusting settings inside each individual app. Drazenko thinks it is a good idea that external apps can be added to GIRAF. It must be possible to lock which applications can be chosen, so a citizen can not just play everything. Use a different word than "Android", since most are used to iPads. Another idea is to make an "Import App" button like when you insert pictures in a Word document. Keep it clean and simple! The drawer becomes irrelevant if the colours are not going to be used. It is not clear what the purpose of the colours is. Double check that the requirements from the groups from previous years (If you have any problems with requirements, write me a mail!) The question is whether or not change of colours is still relevant. An idea could be to remove the possibility of using other applications for a certain period of time, so a citizen can not switch to a different game while working with another. Drazenko thinks that *Launcher* is useful and that it should be simple and grant a good overview. It should be intuitive how the drawer is opened for change of colour. Go out and test with the users! Do not say anything, but observe how they are using it. That is really good feedback. Take pictures so you get some great feedback.

APPENDIX C

Client Meeting 03-04-2014

Check if Android has a feature similar to Apples' "Restricted Access". They like the idea regarding blocking access to other applications while the "Timer" is running. They like the idea regarding profile selection in both *Launcher* and the individual applications. They would prefer accessing the settings inside the individual applications. They like the idea of a collection of settings in *Launcher*, but they would prefer to have it inside the applications. They like the feature of creating setting for the guardian and being able copy settings onto the citizen profiles. They would like both. They like the idea of being able to bring external applications into *Launcher*. They particularly like the idea of disabling and enabling applications for different citizens. Birken thinks it is okay to see who is logged in. To change the colours of applications is not a requirement for them. It is fine with different colours on the applications, but changing colours is not important.

APPENDIX D

Front End Applications

This appendix describes all front end applications in the GIRAF suite.

Launcher provides an interface for accessing the other tablet applications in a controlled environment, easy to use for both guardians and citizens. Through *Launcher*, the guardians should be able to control what applications the citizens should be allowed to use. In the application interface, *Launcher* is referred to as GIRAF.

Sekvens allows users to build sentences from pictograms, a central activity in the lives of the citizens GIRAF is made to service. Many citizens with autism, especially young children, have difficulty formulating sentences in speech. A central feature of *Sekvens* is also the possibility to save often used sequences of pictograms.

Pictooplæser is similar to *Sekvens*, but is focused on building more ad hoc sentences, which the application is then able to read aloud using either an existing recording, or an online text-to-speech tool.

Kategoriværktøjet allows guardians to manage the categories into which the pictograms are organised, including adding and deleting categories and subcategories.

Oasis App is an administration tool for manipulating the user information in the database.

Pictosearch is not used as a standalone application, but provides other applications with a common interface for searching through the device's collection of pictograms.

Pictotegner allows the user to create their own pictograms with basic graphics tools, such as a free-hand pen tool, a rectangle tool, a circle tool etc.

Livshistorier is also similar to *Sekvens*, but is created specifically for saving pictogram sequences that contain instructions for the citizen's everyday life, e.g. instructions on how to use the bathroom.

Ugeplan allows the users to build weekly activity schedules using pictograms. Autistic citizens thrive best in highly strutured environments with regularly scheduled days.

Tidstager provides the ability to time an activity, allowing a guardian to let a citizen use an application, e.g. a game, for a set amount of time. When the time runs out, the device locks, forcing the citizen to move on to the next scheduled activity.

Stemmespillet is a game, where the citizen controls a car by varying the volume of his or her voice.

Kategorispillet is a game, where the citizen unloads pictograms from a train. The pictograms must be unloaded at different stations, where each station accepts a certain category of pictograms.

Web Ugeplan is a web-version of *Ugeplan*, specifically design for large touchscreens. A few costumers had access to TV-size touchscreen, which they specifically wished to use for their week schedules.

Webadmin is a web-based administration tool for manipulating the user information in the database.

APPENDIX E

Further code examples

This appendix contains code fragments, that are too big to be included as part of the report and that can be omitted..

E.1 Debug mode for development

This section describes the implementation of a debug mode, making developers able to skip certain steps in *Launcher*, such as the animation screen and the authentication screen.

When opening *Launcher* one is presented *Main Activity*, which shows a loading animation, while loading data from the remote database[1]. *Launcher* also requires the users to authenticate themselves, before being given access to the *Home Activity*. While these activities hold meaning in the context of the intended users, a significant amount of debugging time is wasted, as the application is often reinstalled and restarted during development.

To overcome this problem, we decided to implement a debug mode to simplify and quicken the process of working with *Launcher*. The debug mode is controlled from the source code `MainActivity` by setting the local fields below (thus, it is not possible to control debug mode at runtime):

- Enable (true) or disable (false) debug mode entirely, overriding other settings:
  `private final boolean DEBUG_MODE = true;`
- Skip `AuthenticationActivity` activity:
  `private final boolean showAuthentication = false;`
- Skip the animation on `MainActivity` activity:
  `private final boolean showMainAnimation = false;`
- Login either as a guardian or child when skipping authentication:
  `private final boolean loginAsChild = false;`

When the above fields are set, debug mode is enabled globally in *Launcher* from the `onCreate()` method in `MainActivity` through the call showed in Listing E.1.

```
if(DEBUG_MODE)
  LauncherUtility.enableDebugging(DEBUG_MODE, loginAsChild, this)
     ;
```

[1]Please note, that because the remote database synchronisation was enabled so late in the fourth sprint, debug mode has not been tested.

Listing E.1: Enable debug mode from `MainActivity`.

The `enableDebugging()` method is seen in Listing E.2. It needs a reference of the calling activity to show debug information in the active activity.

```java
public static void enableDebugging(boolean debugging, boolean
    loginAsChild, Activity activity) {
  DEBUG_MODE = debugging;
  DEBUG_MODE_AS_CHILD = loginAsChild;

  ShowDebugInformation(activity);
}
```

Listing E.2: Enable debug mode by calling `enableDebugging()`.

The code in Listing E.3 is used to set the necessary views, as to inform the developer that debug mode is enabled.

```java
public static void ShowDebugInformation(Activity a) {
  if (DEBUG_MODE) {
    LinearLayout debug = (LinearLayout) a.findViewById(R.id.
       debug_mode);
    TextView textView = (TextView) a.findViewById(R.id.
       debug_mode_text);
    textView.setText(a.getText(R.string.giraf_debug_mode) + "␣" +
        (DEBUG_MODE_AS_CHILD ? a.getText(R.string.
       giraf_debug_as_child) : a.getText(R.string.
       giraf_debug_as_guardian)));
    debug.setVisibility(View.VISIBLE);
    try {
      Thread.sleep(200);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  }
}
```

Listing E.3: Show a debug information on activity if debug is enabled.

E.2 Launching Google Play

The code in Listing E.4 describes the `OnClickListener` for the **"Butik"** button in the "Apps" pane of settings. It attempts to open the Play Store app - If it is not installed on the device, it opens Play Store in the browser instead.

```java
googlePlayButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
                // Try to start the Google Play app
                try {
                        Intent intent = new Intent();
                        intent.setData(Uri.parse(
                          MARKET_SEARCH_APP_URI + PUBLISHER_NAME
                          ));
                        intent.setFlags(Intent.
                          FLAG_ACTIVITY_NEW_TASK | Intent.
                          FLAG_ACTIVITY_CLEAR_TOP);
                        startActivity(intent);
                        // If Google Play is not found, parse the
                           url for Google Play website
                }
                catch (android.content.ActivityNotFoundException
                  e) {
                        startActivity(new Intent(Intent.
                          ACTION_VIEW, Uri.parse(
                          MARKET_SEARCH_WEB_URI + PUBLISHER_NAME
                          )));
                }
        }
}
```

Listing E.4: The OnClickListener for the googlePlayButton, launching the Play Store correctly

E.3 Derived class of LoadApplicationTask

Listing E.5 shows the `LoadGirafApplicationTask` - a derived class of `LoadApplicationTask` . The methods of the class calls `super.fooBar()` to let the super class carry out most of the work, along with initiating the `AppUpdater` in that class and specifying which applications to load.

```
class LoadGirafApplicationTask extends LoadApplicationTask {

        @Override
        protected void onPreExecute() {
                if(appsUpdater != null)
                appsUpdater.cancel();

                super.onPreExecute();
        }

        @Override
        protected HashMap<String, AppInfo> doInBackground(
          Application... applications) {
                apps = ApplicationControlUtility.
                  getGirafAppsOnDeviceButLauncherAsApplicationList
                  (context);
                applications = apps.toArray(applications);
                appInfos = super.doInBackground(applications);

                return appInfos;
        }

        @Override
        protected void onPostExecute(HashMap<String, AppInfo>
          appInfos) {
```

```
                    super.onPostExecute(appInfos);
                    loadedApps = appInfos;
                    startObservingApps();
                    haveAppsBeenAdded = true;
          }
}
```

Listing E.5: The LoadGirafApplicationTask, derived from LoadApplicationTask. This is the derived class used by GirafFragment to load applications into view. Please note that all comments and the constructor have been removed to make the listing smaller

E.4 Marking GIRAF and Android applications

Listing E.6 contains the code for marking GIRAF applications in the "Apps" pane of settings, while Listing E.7 contains the code for marking Android applications in the same pane.

```
ProfileApplicationController pac = new
    ProfileApplicationController(context);
ProfileApplication pa = new ProfileApplication(currentUser.getId
    (), app.getApp().getId());
if(pa == null)
          pac.insertProfileApplication(pa);
else
          pac.removeProfileApplicationByProfileAndApplication(app.
              getApp(), currentUser);
```

Listing E.6: The methods used for adding or removing a Giraf application to a user

```
String activityName = app.getActivity();

if (selectedApps.contains(activityName))
     selectedApps.remove(activityName);
else
     selectedApps.add(activityName);
```

Listing E.7: The methods used for adding or removing an Android application to a user. Please note that the documentation has been removed.

APPENDIX F

Future Works

This appendix contains information about the things still remaining to be done in the *Launcher* project. As such, it acts as a backlog for the next year bachelor students working on the project.

F.1 Proper Icon Size

When scaling the application icons some of them gets pixellated. This indicates that there is still work to be done, when choosing the icons to show. This might be caused by low resolution icons supplied by the developers of the other applications.

F.2 Ugeskema Calendarwidget

The group working on the *GIRAF components* has a widget, showing the current date, in their backlog. It could be an idea to use to open the application *Ugeplan* through this widget. The application should then show the schema for the citizen chosen from the profile selector.

F.3 Login with Citizen and Administrator Profiles

Currently, it is possible to login with all types of profiles, but there is no handling of types other than guardian profiles in most of the application. It might be needed for citizen and administrator login as well. Citizen profiles should not be able to access certain activities, such as settings in *Launcher*. On the contrary, the administrator profiles would need additional access to system administration applications.

F.4 Automatically Download GIRAF Applications

It could be preferable to have *Launcher* start downloading all GIRAF applications as soon as it is started. This is most likely not possible in Android but it is possible to start Google Play with a search string. This way *Launcher* could begin with starting Google Play with the common part of the Java package name used for the GIRAF applications a search string.

F.5 Adding Sound to Launcher

Currently, none of the actions a user carries out when using *Launcher* provides audio feedback. Adding minor sounds for button presses, application launching, setting changes and the like, could be a nice addition to *Launcher*.

F.6 Profile Selection Dialog

It should be clearly indicated what role, the individual user is assigned, in the list when opening the profile selection dialog. Currently there are no indication of the role of a user. Please note, that the profile selection dialog is supplied by *GIRAF Components*, and is therefore not directly the responsibility of *Launcher*.

F.7 Copying settings from one user to another

For many citizens, the settings may be quite similar. For this reason, it would be preferable to be able to copy settings from one user to another.

F.8 Authentication Activity

There is still work to do in *Authentication Activity*.

- **Make it clear that the QR code was successfully scanned.**

  An idea is that the camera feed is hidden when the QR code is accepted and who buttons are show, one to scan again and one to login.

- **Rotation of the instruction animation.**

  Currently it is shown as the tablet should be held in portrait mode. For usability matters this should be rotated.

F.9 Download pictograms while tablet is being used

When *Launcher* is started after a reboot of the device the synchronisation i of data with the remote database is done before being able to login. If *Launcher* has never been installed on the device before, this means that it can take up to 30 minutes, before it is ready to login. It would be an idea to only load the necessary information such as profiles, and then continue downloading the pictograms in the background while the tablet is being used.

F.10 Use GridView for loading applications

The layout in which applications are being loaded is of an inconvenient type. They should instead be loaded into a `GridView`, using a custom adapter similar to the one mentioned in Section 4.3.4. This is better for memory management, follows Android Best Practices and would simplify the algorithm used for showing applications.

F.11 Conduct Functional Tests

Section 2.4 describes function tests carried out by this group, on the version of *Launcher* we initially took over. These tests provided valuable information and revealed several unhandled bugs. As the final version of *Launcher* was rushed to implement the remote database, it is highly recommended, that the group taking over the development of *Launcher*, carry out functional tests as well.

Bibliography

[1] Kasper Møller Andersen, Magnus Stubman Reichenauer, Rasmus Steiniche, and Thomas Kobber Panum. *Launcher - Part of the GIRAF Platform*. Technical report, Aalborg University, 2012.

[2] David Benyon. *Designing Interactive Systems*. Addison-Wesley Professional, 2010.

[3] Fred Chung. *Custom Class Loading in Dalvik*. http://android-developers.blogspot.pt/2011/07/custom-class-loading-in-dalvik.html, 2014. Visited 21-05-2014.

[4] edureka! *Understanding Adapters in Android*. http://www.edureka.in/blog/what-are-adapters-in-android/, 2014. Visited 28-04-2014.

[5] Anders Frandsen, Michael Lisby, Nikolaj Andersen, and Rune Jensen. *Android Application Management System for Children with Disabilities*. Technical report, Aalborg University, 2011.

[6] Google Inc. *Android Developers Reference App Manifest*. http://developer.android.com/guide/topics/manifest/manifest-intro.html, 2012. Android Developers. Visited 10/4/2014.

[7] Google Inc. *Introduction to Android*. https://developer.android.com/guide/index.html, 2014. Visited 10-04-2014.

[8] Google Inc. *Android Developers Reference*. http://developer.android.com/reference/packages.html, 2014. Android Developers. Visited 12/3/2014.

[9] Google Inc. *BroadcastReceiver*. http://developer.android.com/reference/android/content/BroadcastReceiver.html, 2014. Visited 21-05-2014.

[10] Google Inc. *Fragments*. http://developer.android.com/guide/components/fragments.html, 2014. Visited 21-05-2014.

[11] Google Inc. *Receiving an Implicit Intent*. http://developer.android.com/guide/components/intents-filters.html#Receiving, 2014. Visited 21-05-2014.

[12] Jesper Kjeldskov, Mikael B. Skov, and Jan Stage. *Instant Data Analysis: Conducting Usability Evaluations in a Day*. NordiCHI '04, 2004.

[13] Chris Kohlhardt and Clint Dickson. *Gliffy Modelling Website*. http://www.gliffy.com/, 2014. Gliffy Website. Visited 09/04/2014.

[14] Craig Larman. *Agile & Iterative Development: A Manager's Guide*. Addison-Wesley Professional, 2003.

[15] Jens M. Lauridsen, Johan Sørensen, Lars Chr. Pedersen, and Tommy Knudsen. *GIRAF - Admin*. Technical report, Aalborg University, 2013.

[16] Oracle. *How and When To Deprecate APIs*. http://docs.oracle.com/javase/1.5.0/docs/guide/javadoc/deprecation/deprecation.html, 2014. Java Documentation. Visited 07/04/2014.

[17] Ron Patton. *Software Testing*. Sams, second edition, 2005.

[18] Ken Schwaber and Jeff Sutherland. *The Scrum Guide*. https://www.scrum.org/Portals/0/Documents/Scrum%20Guides/2013/Scrum-Guide.pdf, 2013. Visited 25-05-2014.

[19] Lars Vogel. *ListViews and Performance*. http://www.vogella.com/tutorials/AndroidListView/article.html#adapterperformance, 2014. Visited 25-05-2014.