AALBORG UNIVERSITY
STUDENT REPORT

GIRAF: CAT, Database and Oasis

SW6 Project
Group sw614f14
Software
Department of Computer Science
Aalborg University

28 of May 2014

AALBORG UNIVERSITY
STUDENT REPORT

Department of Computer Science
Selma Lagerlöfs Vej 300
Telephone: +45 9940 9940
Fax: +45 9940 9798
http://www.cs.aau.dk/en/

Title: Embedded Systems

Project period:
SW6 semester project, spring 2014
3rd of February 2014
28th of May 2014

Project Group:
SW614F14

Participants:
Martin Bøje Rasmussen
Simon Rasmussen
Simon Binderup Støvring
Kasper Lind Sørensen

Supervisor:
Thomas Pedersen

Circulation: 6

Amount of pages: 66

Amount of appendices: 8

Summary

This report describes parts of the development of GIRAF, an ecosystem of applications created to allow the use of mobile applications running on Android tablets to help people with autism. The project was divided into four sprints and each sprint will be documented in this report.
The first sprint documents the refactoring, maintenance and visual improvements made to CAT, the tool used to categorize pictograms. During the first sprint the program Pecs Importer, used to import pictograms into the database, was developed. The develpoment and the choices made regarding what pictograms to import and how they are imported is documented in the chapter on sprint one.
In sprints two to four the work done on the local database used in GIRAF and OasisLib is documented. The OasisLib is a library used to manipulate the data in the local database installed on the Android tablets. The chapters on sprints two to four will also document the work done on Oasis, the administrative tool for the Android tablets.
The project was part of a multi project in which sixteen groups worked together on multiple related projects. The collaboration with the other groups working on the multi project will be described and two activites made with other groups will be presented.

Contents

1 Introduction

This report describes the development of group sw614f14's bachelor project during the period from February the 3rd to May 28th.

Our development process is a small part of the large scale multiproject solution called GIRAF with the goal of fostering collaboration and emulating a large scale software development project. As such this report will focus on our additions to the mulitproject and the collaboration with other groups working on other parts of the muliproject during the same period.

GIRAF stands for *Graphical Interface Resources for Autistic Folk* and is the name of the digital ecosystem meant to replace the paper-based system and is developed at Cassiopeia, the computer science department of AAU. It was started in 2011 with the vision of providing a complete digital environment for autistic people and their guardians with by providing a complete application environment for Android tablets.

1.1 Customers

The customers consists of a large variety of institutions from the Aalborg area that are specialized in taking care and teaching citizens with autism. These citizens can be young children in preschool all the way up to almost self sufficient adults who live in dedicated areas with caregivers, so called guardians, nearby ready to assist day and night. Below is a list with the names of the institutions we worked with and their type:

- Taleinstituttet - Speech improvement
- Birken - Preschool
- Egebakken - School
- Mymo - School
- Bosted - Housing area

These were also our immediate stakeholders and were represented by 1-2 contact persons from each institution whom all were invited to each of the sprint review meetings that are further described in *1.3 Project Management p. 6*.

Since the citizens require very specialized care the parents' roles are diminished compared to regular parent-teacher relationships and as such parents are not given full insight into the day to day care and teaching of the citizens[12].

1.2 Existing System

The existing systems each of the institutions use are all based on pictograms and these are sorted in many different categories as seen in figure 1.1.

Figure 1.1: Picture of a whole wall with boxes that each represent one category of pictograms.

The pictograms are used for many different tasks in the teaching and day to day life of the citizens, one such usage is creating sequences to describe step by step guides on how to reach a goal such eating breakfast. In the case of eating breakfast the sequence could consist of finding a spoon, a bowl and cereal and last but not least eating the food. Each citizen has its own folder with common sequences, such that these sequences do not need to be recreated each day, an example of such a folder is shown in figure 1.2.

Figure 1.2: Picture of a folder that holds many sequences of pictograms for a citizen.

Another usage for the pictograms and sequences are to create life stories that serve as memories of events in the citizen's life and as such can be photos of the citizen shot by the guardians.

A third application of pictograms and sequences are to create sentences that expresses the desires of the citizens and therefore facilitates the communication with the guardians since a trait of autism is having inhibited vocal language.

1.3 Project Management

It was early in the project decided across all the project groups to use agile development methods for the multiproject. This meant that the project was divided into 4 scrum-like sprints with backlogs, sprint review meetings and weekly meetings with one representative from each group.

The use of a common integrated development environment (IDE) was agreed upon early on in one of the early meetings with all the groups in the multiproject, since it would make it easier to change between projects between sprints. Android Studio was chosen because many groups had bad experiences with Eclipse, and good experiences with IntelliJ on which Android Studio is based[1].

The overall project was divided into subprojects by the previous year's students and this division was adapted with groups being assigned 1-3 subprojects for 1 sprint.

At each sprint start these assignments were reevaluated but after sprint 2 almost all groups continued with the same assignments.

Sprint	Projects worked on
1	Category Administration Tool
2	Oasis, OasisLib and LocalDB
3	Oasis, OasisLib and LocalDB
4	Oasis, OasisLib and LocalDB

Project	Type of work
CAT	Refactoring
Oasis	Rewrite
LocalDB	Refactoring and shared code extraction
OasisLib	Refactoring, shared code extraction and new features

Figure 1.3: Overview of the type of work done on each project and which sprint which projects was worked on.

As seen in figure 1.3 we worked on a wide range of projects throughout the semester, that touched many parts of the system. This enabled us to get an overview of what could be improved between different parts of the system, especially between parts that were previously developed by different groups.

Figure 1.3 also shows the primary types of work that was done to each project throughout the semester. In this context refactoring means we cleaned up "bad smells" from the code in accordance to [6] which includes things like removing duplicated code, splitting up God classes and making sure each class has a clear set of responsibilities.

Rewriting a project means we deemed starting from scratch due to the amount of time it would take to refactor the project would exceed rewriting it, the reasoning behind this is specified in the relevant sections.

Shared code extraction means we found code duplicated across multiple projects and as such could be extracted from both and moved into a shared library. Last but not least new features meant implementation of features, in the logic and data tiers, as requested by other project groups.

Time management was primarily done through the multiproject's issue tracking software Redmine. This was done by creating issues and tasks for almost everything regarding code and especially feature requests and bug reports coming from other groups. Each of these tasks/issues were then given an estimated amount of time for completion and then logged for amount of time spent on it. All in all our approach was similar to the agile development method Scrum but without the daily meetings since we felt this was unnecessary when sitting directly next to each other with the ability to do pair programming and ask questions if specific trouble occurred with solving issues.

2 GIRAF Architecture

The GIRAF system we inherited from the previous year consisted of many different applications, libraries and systems which each has a different set of responsibilities and requirements.

The list of applications in *appendix A.1* shows that the system is segregated into three different groups, namely online or offline user accessible applications, libraries and data storage which resembles a three tier system architecture.

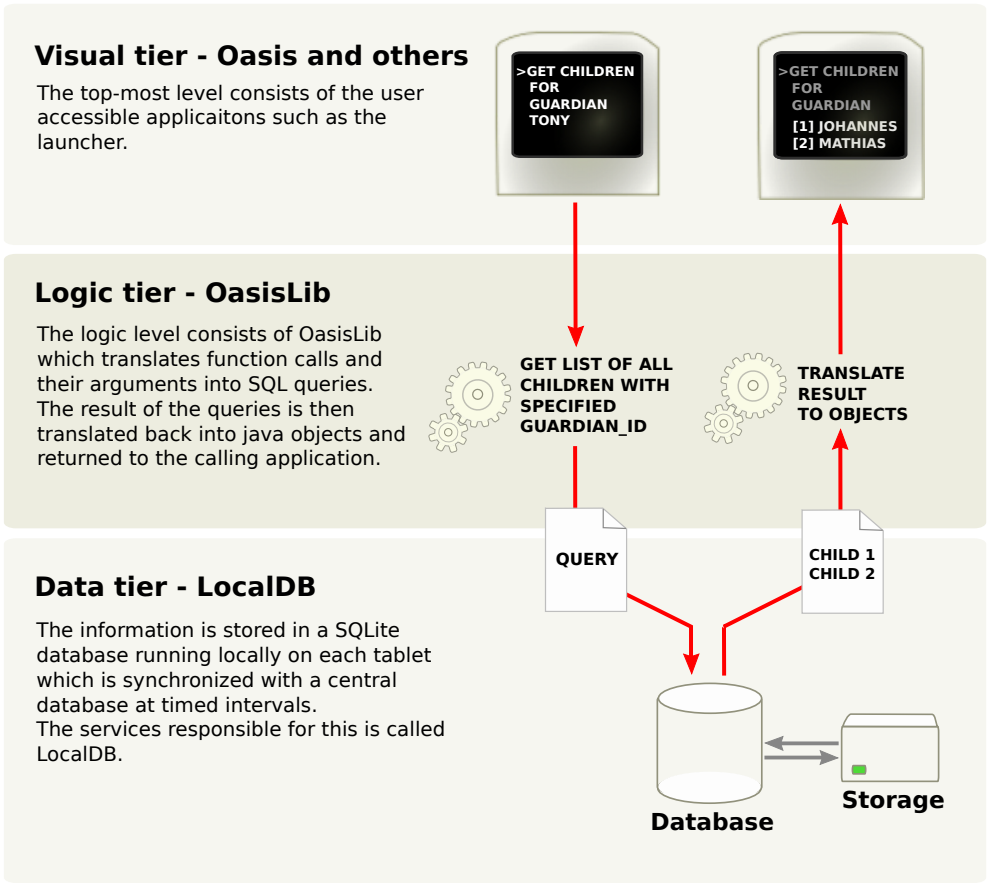


Figure 2.1: Overview of the three tier architecture of GIRAF

This three layer architecture is illustrated in figure 2.1 and shows how the the visual tier consists of user accessible applications such as the launcher, wombat, train, car and Oasis that communicate only with the layer below in the form of OasisLib which the forwards the data request to the storage layer.

It should be noted that the GUI library, also called GIRAF component, lives in the visual tier and is responsible for providing common GUI components that other visual tier applications use such as buttons, lists and color schemes.

The logic tier that consists solely of OasisLib which provides a object oriented interface to the presentation tier that translates function calls and their arguments to queries the data tier can execute. The returned data is also translated back into objects by the logic tier to ease the usage of the database.

The data tier consists of LocalDB and RemoteDB where LocalDB is the SQLite service running on each individual tablet providing an offline version of the central database while RemoteDB is a MySQL server which acts as central storage and enables the web admin interface to administrate the data on tablets.

2.1 OasisLib Architecture

As mentioned earlier, we inherited OasisLib from previous year which has a design reminiscent of Model-View-Controller (MVC), without the view part since the visual tier applications are responsible for this, which is a common design pattern amongst business applications.

The idea behind the MVC pattern is that presenting data to the user, representing the data and executing functions on the data should be separated cleanly from each other. In our case this results in models simply being containers of data that represents tables with data types from Java. Controllers are the parts of OasisLib that ends up talking to LocalDB through the `ContentResolver` interface described in *2.2 Android Database Sharing p. 10*. Whenever a function to query the database is called the arguments are transformed into selection statements on the correct table which are then transformed further by LocalDB. The result of the query is then returned to the calling controller which transform the result into a single instance of the model or multiple depending on the desired result.

If the user of the API wants to retrieve a `Pictogram` from the database, they have to create a `PictogramController` and use any of the `get-` methods created for this purpose.

2.2 Android Database Sharing

Due to the limited amount of resources on platforms that Android is intended for and security there is no Database Management System (DBMS) installed that can be accessed from all applications, instead developers have the ability to run a application local SQLite database which is more lightweight than a full DBMS like MySQL. This is problematic since data sharing and synchronization must happen over unstructured inter process communication (IPC) channels instead of being handled by the DBMS.

This is however not a new problem since Android stock applications need to share structured information such as contact persons and is therefore already solved with the concept of content providers and resolvers where one module provides content and many others asks for content.

The content provider publishes data to a Unique Resource Identifier (URI), which is similar to a website URL, and whenever a content resolver request data from this URI the operating system will do the heavy lifting of finding which process to contact and move data between the two processes to fulfil the request.

This means that sharing a database in Android boils down to creating a service that extends the abstract `ContentProvider` class supplied by the Android framework and choosing which URIs expose what data.

For example to fulfil a query request the `ContentProvider` implementation must translate the received URI into a table the query should be executed on and then return the result of this query. Translation of this URI is for the most part trivial since it's just a matter of giving the URIs and tables the same name but in the case of more advanced queries that requires joining of tables special precautions must be taken to translate this into a SQL join statement.

3 Sprint 1

In sprint number 1 we worked with the Category Administration Tool (CAT) from February the 12th to March the 19th.

The basic premise for the CAT application is to provide an administrative interface to create categories for individual citizen. Categories provide a way for guardians of a citizen to group and filter the pictograms that a citizen should have access to. For example, a guardian could create a category called "Taylor's Morning" with all pictograms that Taylor would need in the morning. Taylor may not be able to search pictograms using the PictoSearch application as she may not be able to spell. By creating separate categories each associated with a pictogram to distinquish the category, the guardian makes it easier for Taylor to navigate all the pictograms that she will need.

Categories are also a necessity for the Train application in which a guardian will create a set of stations, each station representing a category. The train will then be loaded with pictograms belonging to each of the stations. When the train arrives at a station the citizen will have to unload the pictograms belonging to the category the station represents from the train.

The application provides functionality to add new categories and subcategories, choose the color of categories and with the help of the picto search application add pictograms to these categories.

In this sprint a a script to import pictograms into the central database was developed. This is described in *3.1 Pecs Importer p. 13*.

Last but not least we did some refactoring of the CAT application as described in *3.2 Changes to CAT p. 15* which was partially done in regards to what was introduced in the Software Engineering course the previous semester and partially in regards to the analysis of the requirements and thing what was decided at the semester project status meetings.

3.1 Pecs Importer

The customers expressed that the current pictograms was too limited and wanted more.

Having looked at Picto-Selector, a desktop client to manage pictograms[18], it was clear that a database of pictograms could eventually grow to include thousands of pictograms and these all had to be imported somehow. This can be done by manually importing the images one by one and provide a text for them. Because this approach would end up taking a long time, especially if done more than once, it was instead decided to develop a piece of software that would import the pictograms into the database. Having a program that can perform the import will make the process faster. By ensuring that the program is reusable, the process of developing the soft-

ware will not have to be repeated if it is decided to import a new set of pictograms in the future.

3.1.1 Distribution of Pictograms

At first, we investigated how pictograms are usually distributed. This was done by visiting websites that provide pictograms. The pictograms were then downloaded and the files and format were compared.

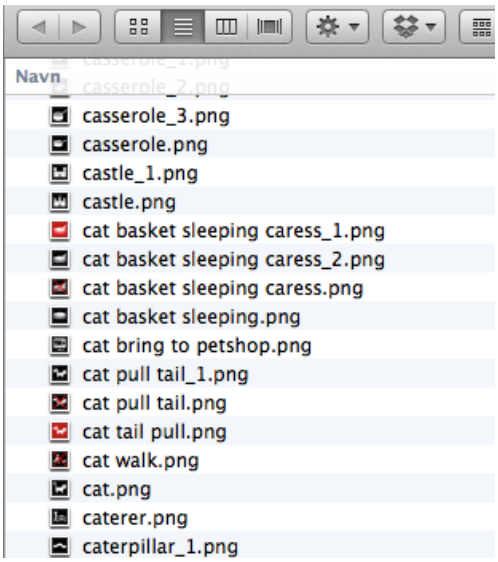
On the website of Picto-Selector, there are links to various websites that provide pictograms free of charge[19][20]. The symbols may be distributed with various license, however the license is not relevant for the purpose of investigating how the images are distributed.

The websites that are referenced on the Picto-Selector distributes pictograms in two different ways. On some websites the pictograms are downloaded one by one while other websites provide an archive of pictograms.

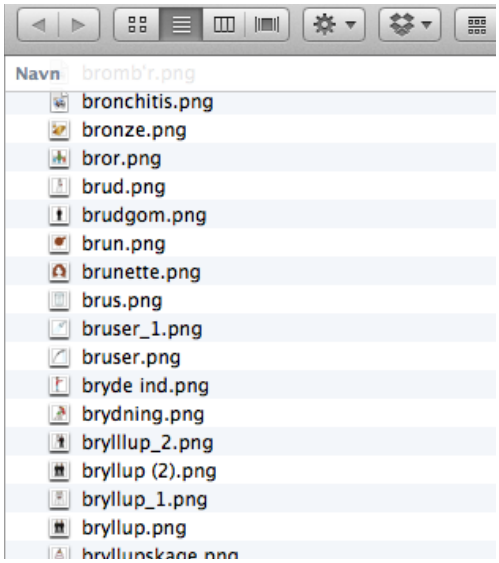
None of the pictograms that could be found through the Picto-Selector website were distributed in Danish. This was an issue as the pictograms that will be included in Giraf will be used by Danish citizens.

Kommunikationscenteret Hillerød distributes the ARASAAC archive of roughly 6,300 pictograms drawn by Sergio Palao and translated into Danish by Kommunikationscenteret Hillerød[9].

It was chosen not to work with the pictograms that are distributed one by one and instead focus on the archives as these are more easily downloaded. Some of archives are distributed with files that have meaningful names. Amongst others, this is the case for both the English Sclera package and the Danish ARASAAC package as depicted in figure 3.2. These file names can be stored in the database along with the image to easily identify the pictogram.



(a) Example of contents of the Sclera archive



(b) Example of contents of the ARASAAC archive

Figure 3.1: Examples of packages containing pictograms.

3.1.2 Usage

Pecs Importer is a small program that takes the path to a directory as argument and imports all images in that directory. The program is run as shown in *Code 3.1*. The `-p` argument specifies the password of the database user. More arguments can be set as seen in the information shown when running the program with the `-help` or `-h` argument.

Code 3.1: Example usage of the Pecs Importer

```
1 python pecs-importer.py -p mypassword ~/path/to/images
```

3.1.3 Implementation

The responsibility of the Pecs Importer is to load a batch of images and store them in the central database. Table 3.1 shows the schema of the table in which the pictograms are stored. The relevant fields for Pecs Importer is the `name`, `inline_text`, `image_data` and `public`. The `name` and the `inline_text` are both populated with the name of the file without the file extension. In the `image_data` we store the image. `public` is always set to true so all pictograms imported with Pecs Importer are available for everybody.

The help screen is generated by Pythons `argparse`, which provides a simple way of handling arguments and displaying the documentation for the arguments. `argparse` was chosen as the way it treats arguments and the help screen it generates should be immediately familiar to anyone comfortable with command line tools.

Field	Type
id	int(11)
name	varchar(64)
public	tinyint(4)
image_data	blob
sound_data	blob
inline_text	varchar(64)
author	int(11)

Table 3.1: `pictogram` table in which the pictograms are stored

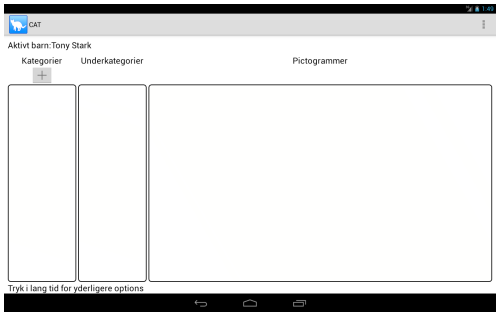
Using the official MySQL connector[13], Pecs Importer will open a connection to the central database in which pictograms are stored. Every file in the directory is then iterated and we check if the data in the file is a valid image. If it is, we insert the image into the database with the file name as the text. Finally we commit the changes and close the connection to the database.

3.2 Changes to CAT

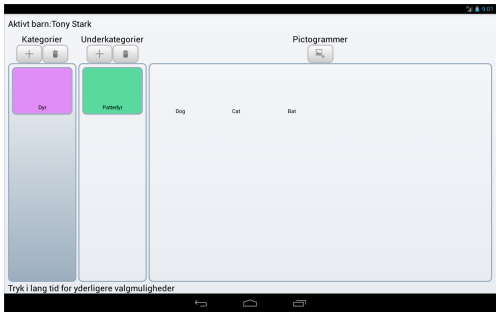
In this section we describe the changes made to the CAT application, including both changes to the interface, its underlying code and functionality.

Shown in figure 3.2 are two screenshots of CAT. Figure 3.2a shows the initial version of the application, which was created a previous year as described in *chapter 1*. Figure 3.2b shows the result of the changes made to the interface of CAT.

Our focus during this sprint was to update the interface to adhere to the standard that GIRAF applications must use GIRAF components, and refactor code for example by removing redundant code.



(a) The interface of CAT when we started at the beginning of sprint



(b) The interface of CAT at the end of sprint 1

Figure 3.2: Before and after screenshots of CAT.

3.2.1 Resource Strings

While reviewing the source code of CAT we noticed that the code included a lot hard coded strings. Resources such as strings and images should be separate from the code, allowing easier maintainability[4]. If an error is found in a string, or a string has to be changed, it is easy to find all the strings in the same place. In the future if localization has to be added, it is possible to add other locales using the resource files[3]. Therefore we extracted the strings from the source code and placed them in the string resource file.

An example of how a string is hard coded into the source code is shown in *Code 3.2*. This is a message which is shown if a certain button is pressed, without the application used to create pictograms installed. Once the application is finished, this could be changed to send the user to Google Play where they could download and install the application.

Code 3.2: An example of how the string is written into the source code.

```
1 message = new MessageDialogFragment("Croc is not installed");
```

Instead we used the method shown in *Code 3.3* where *R.string.croc_missing* refers to the string resource shown in *Code 3.4*.

Code 3.3: The use of resource files after extracting a string.

```
1 message = new MessageDialogFragment(R.string.croc_missing);
```

Code 3.4: The use of resource files after extracting a string.

```
1 <string name="croc_missing">Croc is not installed</string>
```

3.2.2 Code Cleanup

The main interface contains two lists, one for categories and one for subcategories. In the code there are several methods that are called based on the selected item. One of these changes the title of category and is shown in *Code 3.5*. This example is a direct

copy of the code before we started refactoring. *Code 3.6* is the code snippet after our refactorization. The first major problem with this piece of code is that the part in the else statement is a copy of the if statement, with the only difference being which category it works on. To solve this problem we took the parameter `isCategory` *(Code 3.5, l. 1)* which is used in the `if` statement to determine which list of categories should be used, and replaced it with a list of categories*(Code 3.6, l. 1)* . With this change we have moved the `if else` statement outside this method, and removed a chunk of copy pasted code.

The next change we did was to remove the boolean `legal`. This was done by replacing the `break` in the for loop*(Code 3.5, l. 9)* with a `return`*(Code 3.6, l. 7)* . Having a `return` at this position does the same as the `legal` boolean from *Code 3.5*, in that it skips the operations that change the name of a category and returns to the caller of the method.

Code 3.5: The original `updateTitle` function.

```
1  private void updateTitle(PARROTCategory tempCategory, int pos, boolean isCategory) {
2      boolean legal = true;
3      if(isCategory) {
4          for(PARROTCategory c : categoryList) {
5              if(c.getCategoryName().equals(tempCategory.getCategoryName())) {
6                  legal = false;
7                  message = new MessageDialogFragment(getString(R.string.title_used_short));
8                  message.show(getFragmentManager(), "invalidName");
9                  break;
10             }
11         }
12         if(legal) {
13             categoryList.get(pos).setCategoryName(tempCategory.getCategoryName());
14             categoryList.get(pos).setChanged(true);
15         }
16     }
17     else {
18         for(PARROTCategory sc : subcategoryList){
19             if(sc.getCategoryName().equals(tempCategory.getCategoryName())){
20                 legal = false;
21                 message = new MessageDialogFragment(getString(R.string.name_used));
22                 message.show(getFragmentManager(), "invalidName");
23                 break;
24             }
25         }
26         if(legal){
27             subcategoryList.get(pos).setCategoryName(tempCategory.getCategoryName());
28             selectedCategory.setChanged(true);
29         }
30     }
31 }
```

Code 3.6: The `updateTitle` after refactorization.

```
1  private void updateTitle(PARROTCategory changedCategory, int pos, ArrayList<PARROTCategory>
       list) {
2      String catName = changedCategory.getCategoryName();
3      for(PARROTCategory c : list) {
4          if(c.getCategoryName().equals(catName)) {
5              message = new MessageDialogFragment(R.string.name_used,this);
6              message.show(getFragmentManager(), "invalidName");
7              return;
8          }
9      }
10     list.get(pos).setCategoryName(catName);
11     list.get(pos).setChanged(true);
12 }
```

A similar change has been done to a snippet of code from the method `onActivityResult` which is called when the pictogram search application returns to CAT.

3.2.3 Backgrounds of Categories

When a category is created, it is assigned a color. The color aids the autistic citizens to identify the category and the color can be used by other applications in the GIRAF ecosystem to distinguish between sets of pictograms. The background of the button that is tapped to select the category was meant to have the color that the category was assigned. The previous group who worked with the category tool did not manage to implement this and instead they used the same gradient background color for all categories.

We added this color to the categories by creating a new layout resource file for the background. This layout is shown in *Code 3.7*. One of the things that differentiates this layout file from the one previously used is the addition of the ID which can be seen on line 3. By referring this ID, we can change the background color at runtime as seen in *Code 3.8*.

Code 3.7: Layout used for the background of categories

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <layer-list xmlns:android="http://schemas.android.com/apk/res/android" >
3      <item android:id="@+id/category_background">
4          <shape android:shape="rectangle">
5              <solid android:color="#ffff0000" />
6              <stroke android:width="3px" android:color="#FF9CAEBE" />
7              <corners android:radius="10dp" />
8              <padding android:left="20dp"
9                  android:top="10dp"
10                 android:right="20dp"
11                 android:bottom="10dp"/>
12         </shape>
13     </item>
14 </layer-list>
```

Code 3.8 shows an extract from the `PictoAdminCategoryAdapter` class. We first load the background resource into an instance of `LayerDrawable` which manages an array of drawables. We get the drawable that we are interested in, specifically the one with ID `category_background`. From this, we can get the shape and store it on an instance an instance of `GradientDrawable`. This class name is somewhat misleading as the `GradientDrawable` does not necessarily have to do with a gradient, it can simply refer to a shape which is the case here. We then change the background color and assign the drawable as background to `convertView` which is the entire view for the row.

Code 3.8: Changing the background color of categories at runtime

```
1 LayerDrawable background = (LayerDrawable)convertView.getResources().getDrawable(R.drawable
      .category);
2 final GradientDrawable shape = (GradientDrawable)background.findDrawableByLayerId(R.id.
      category_background);
3 shape.setColor(categoryColor);
4 convertView.setBackgroundDrawable(background);
```

Using the color the user selected as background caused the black text to be unreadable on certain colors. The solution we came up with for this issue was to change the color of the text depending on the color of the background which can be seen in *Code 3.9*. We take the red, green and blue components of the background color and find the average value between these. This gives us a grayscale of the background color. We

then check if this grayscale color is higher or lower than 127,5. If the grayscale color is higher, then the background is light and we use a black text color. If it is lower, it is a dark background color and we use a white text color.

Code 3.9: Changing the text color depending on the color of the background

```
1  int categoryColor = catList.get(position).getCategoryColor();
2  int red = Color.red(categoryColor);
3  int green = Color.green(categoryColor);
4  int blue = Color.blue(categoryColor);
5  float average = (red + green + blue) / 3;
6  int textColor = average > 255 / 2 ? Color.BLACK : Color.WHITE;
7
8  TextView textView = (TextView) convertView.findViewById(R.id.pictogramtext);
9  textView.setText(catList.get(position).getCategoryName());
10 textView.setTextColor(textColor);
```

3.2.4 Require Category Color

Using the color of a category as background color meant that the user had to select a color for the background, else the color would be `0`, an invalid color. We could use a default color instead of `0`, however, as other applications can rely on the color of a category, we chose to force the user to select a color.

To do this, we added the snippet of code shown in listing 3.10 to the `onCatCreateDialogPositiveClick` method in the `AdminCategory` class. If no color is selected, we display a dialog informing the user that he must select a color and we return from the method.

Code 3.10: Checking if a color has been selected before creating the category

```
1 if (newCategoryColor == 0) {
2     message = new MessageDialogFragment(R.string.category_color_missing, this);
3     message.show(getFragmentManager(), "categoryColorMissing");
4     return;
5 }
```

While adding the check for a selected color shown in listing 3.10 we found that the `onCatCreateDialogPositiveClick` method could easily be refactored to become much more readable.

The method handles addition of both categories and subcategories. This makes sense because the creation of these categories are very similar and the same requirements apply to both types of categories. However, this also means that it is easy to add redundant code. As the same requirements apply, we can use the same code for validating these requirements. For example, it is a requirement that no two categories with the same name can exist and no two subcategories with the same name can exist in the same category. Previously this check was implemented as seen in listing 3.11. Different loops were used and error messages were given in different places depending on whether the user was adding a category or a subcategory. We refactored this into the much more consise and easier to maintain implementation seen in listing 3.12. The main difference is that we run the same loop on different array lists depending on whether the user is adding a category or a subcategory. We also handle the error message as the same place.

In languages such as Java where there is garbage collection, there is no issue to return multiple places in the code. This is what we utilize here. If two categories with the same name exist, we give an error message and simply return. We managed to refactor the `onCatCreateDialogPositiveClick` method further by allowing

return multiple places. We can check all the requirements at the beginning of the method and if a requirement is not fulfilled, we will give an error message and return. Thereby it is known that all the requirements have been fulfilled when lines 10-14 in listing 3.12 are reached.

However, multiple return statements can be an issue in languages where the developer does most of the memory management himself. By restricting the use of multiple return statements, he can do all his clean up of the memory in one place. If the uses multiple return statements, he will have to clean up the memory in multiple places. The use of multiple return statements can be an advantage when used with care as it generally allows for fewer nestings and cleaner code.

Code 3.11: Previous check of existence of categories with the same name

```
1  if (isCategory) {
2      // User is trying to add a new category
3      if(!categoryList.isEmpty()){
4          for(PARROTCategory c : categoryList) {
5              if(c.getCategoryName().equals(title)){
6                  legal = false;
7              }
8          }
9      }
10     if(legal){
11         // Continue adding category
12     } else {
13         // Give error indicating that a category with the same name exists
14     }
15 } else {
16     // User is trying to add a new subcategory
17     for(PARROTCategory sc : subcategoryList) {
18         if(sc.getCategoryName().equals(title)){
19             legal = false;
20         }
21     }
22     if(legal) {
23         // Continue adding subcategory
24     } else {
25         // Give error indicating that a subcategory with the name exists
26     }
27 }
```

Code 3.12: Current check of existence of categories with the same name

```
1  ArrayList<PARROTCategory> relatedCategories = isCategory ? categoryList : subcategoryList;
2  boolean categoryWithNameExists = false;
3  for(PARROTCategory sc : relatedCategories) {
4      if (sc.getCategoryName().equalsIgnoreCase(title)) {
5          // Give error indicating that a category with the same name exists
6          return;
7      }
8  }
9  if (isCategory) {
10     // Continue adding category
11 } else {
12     // Give error indicating that a subcategory with the name exists
13 }
```

4 Sprint 2

In sprint two we worked with Oasis, OasisLib and LocalDB from March the 20th to April the 14th.

Oasis is the management application used on the tablets for offline administration of profiles. OasisLib is the interface that all of the applications in the GIRAF environment rely on to communicate with the local database in an object oriented fashion without the burden of writing SQL statements and serialization code. LocalDB is the background service that provides shared access to the local SQLite database and synchronize data to the remote database.

The part of the GIRAF environment we worked with is the area marked by the dotted line in figure 4.1. The development of OasisLib and LocalDB was picked up from the two groups who worked on it in sprint 1 who claimed the development of LocalDB was done which turned out to not be quite true as shown in *4.1 Implementing N-N Relations In LocalDB p. 21*.

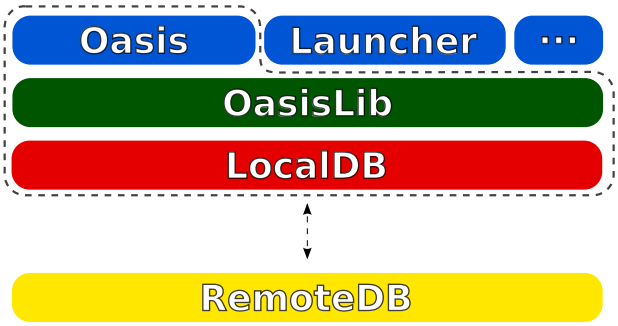


Figure 4.1: Overview of Oasis, OasisLib, LocalDB.

Furthermore the work that done in sprint 1 on OasisLib was mostly translating the existing API design to the new database schema which meant some of the things in the new schema was missed or done incorrectly as described in more detail in *4.2 Implementing N-N Relations in OasisLib p. 22*.

Last but not least the Oasis application was thrown out and rebuilt from scratch since the old version was tightly coupled to the previous database schema and the effort to fix everything would be greater than starting from scratch. This is described and explained in greater detail in *4.4 Oasis p. 25*.

4.1 Implementing N-N Relations In LocalDB

While understanding the code of OasisLib we discovered some weird and inefficient decisions, one of which was that many of the relations between tables was handled entirely by the library instead of the DBMS. This meant that for example getting a list of pictograms with a certain tag name could potentially result in hundreds of queries since first the `tag` table would be queried for the name, then the `pictogram_tag` relation table would be queried for the found id and lastly that list would be used to query the `pictogram` table once per relation to get the right pictograms.

We suspect this was originally done because the Android `ContentProvider` and `ContentResolver` abstraction does not provide facilities to make joined queries and

as such has to be exposed as a separate URI.

So what we did was to create additional URIs that expose these relations as a table that in reality is a double inner join of the three tables a N-N relation consists of such OasisLib could query these directly and efficiently as shown in *4.2 Implementing N-N Relations in OasisLib p. 22*.

Code 4.1: Condensed example of code used to expose joined a table.

```
1  @Override
2  public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs,
       String sortOrder) {
3      //get an instance of a readable database object
4      SQLiteDatabase db = dbHelper.getReadableDatabase();
5      SQLiteQueryBuilder builder = new SQLiteQueryBuilder();
6      //sets the table to be the joined tables of pictogram and tag through the pictogramtag
           table
7      builder.setTables(
8      String.format("%s INNER JOIN %s ON (%s.%s = %s.%s) INNER JOIN %s ON (%s.%s = %s.%s)",
9          PictogramMetaData.Table.TABLE_NAME,
10         PictogramTagMetaData.Table.TABLE_NAME,
11         PictogramMetaData.Table.TABLE_NAME,
12         PictogramMetaData.Table.COLUMN_ID,
13         PictogramTagMetaData.Table.TABLE_NAME,
14         PictogramTagMetaData.Table.COLUMN_IDPICTOGRAM,
15         TagMetaData.Table.TABLE_NAME,
16         PictogramTagMetaData.Table.TABLE_NAME,
17         PictogramTagMetaData.Table.COLUMN_IDTAG,
18         TagMetaData.Table.TABLE_NAME,
19         TagMetaData.Table.COLUMN_ID
20     ));
21     //set distinct to filter out duplicated tags on the same pictogram
22     builder.setDistinct(true);
23     //project OasisLib table and column names to the ones in localdb
24     builder.setProjectionMap(pictogramWithTagProjectionMap);
25
26     //build the query
27     Cursor queryCursor = builder.query(db, projection, selection, selectionArgs, null, null
           , null);
28     return queryCursor;
29 }
```

The primary functionality seen in *Code 4.1* is building the double inner join statement that enables efficient querying of which tags belongs to which pictograms.

Similar implementations were made for the other N-N relations along with the necessary boilerplate code to setup the export of the tables.

4.2 Implementing N-N Relations in OasisLib

OasisLib is the interface between the GIRAF applications and the database that abstracts the raw SQL queries into function calls that returns the information requested in Java objects that are easy to work with.

The primary workload in this process is translating from column names and raw data into the correct type and storing it in objects representing the tables.

Communicating with the exported URI described in *4.1 Implementing N-N Relations In LocalDB p. 21* is done through the Android class called `ContentResolver` which provides the exact same functions as the `ContentProvider`, namely `Query`, `Insert`, `Delete` and `Update`. The difference however is that this time no functions must be implemented as the `ContentResolver` merely works as a shim that enables the Android OS to redirect the function calls to the `ContentProvider` that is registered as the handler of the supplied URI.

Code 4.2: Code to query for pictograms that has been tagged with the supplied name.

```
1  public List<Pictogram> getPictogramsWithTagName(String tagName){
2      Cursor c = _context.getContentResolver().query(
3              PictogramMetaData.PICTOGRAMS_WITH_TAG_URI,
4              null,
5              "? like ?",
6              new String[]{
7                      TagMetaData.Table.COLUMN_NAME,
8                      "'" + tagName + "'"
9              },
10             null
11     );
12     ArrayList<Pictogram> list = new ArrayList<Pictogram>();
13     while (c.moveToNext())
14     {
15         list.add(cursorToPictogram(c));
16     }
17     return list;
18 }
19
20 private Pictogram cursorToPictogram(Cursor cursor) {
21     Pictogram pic = new Pictogram();
22     pic.setId(cursor.getInt(cursor.getColumnIndex(PictogramMetaData.Table.COLUMN_ID)));
23     pic.setName(cursor.getString(cursor.getColumnIndex(PictogramMetaData.Table.COLUMN_NAME)
           ));
24     // Some setters are omitted in this code snippet
25     return pic;
26 }
```

The code in *Code 4.2* is the corresponding function in OasisLib that queries the newly created URI that we defined in *Code 4.1* and translate the SQL data into an instance of the `Pictogram` class.

It can be seen that this is much more efficient than querying the DBMS multiple times as the DBMS does a lot of joined query optimization that is impossible to do across separate queries since the DBMS will have no knowledge about the queries being related.

4.2.1 Additional Changes to OasisLib

In this section we will list some of the changes made to OasisLib during sprint 2.

- We removed all the modify methods to controllers that act as a link in a many to many relationship. We did this because we wanted such a relation to belong to the two rows it refers to. Instead of modifying, a remove and insert can be used.
- All insertion method names have been made consistent to use `insert` as the first part of their names. For example `addUser` was renamed to `insertUser`.
- Images are saved in the database as binary large objects (BLOB). They are used in `pictogram`, `category` and `profile` as shown in *appendix C*. Previously the API exposed a BLOB to the outside. We changed it to expose a bitmap instead because it is easier for the users of the API just to get the image instead of a BLOB. We then make sure to compress the images when they are saved to the database, and decompress when retrieving it.

4.3 Test of OasisLib

OasisLib which we inherited from the previous year did not have any tests when we started working on it in sprint 2. During the development and maintenance of `OasisLib` we chose to set up integration tests. Integration tests were used because of

the need to test the interaction between the database and the library. We wanted to make sure the calls to the database work as intended, so that when other groups work with the library they can be sure that our methods work if they have a problem. In addition to make sure that it works, the tests can show other groups, including those who are going to work on it in the coming years, how to use the library. The test suite in Android Studio is used because it is convenient, since it is the development tool we use, as described in *1.3 Project Management p. 6*. The testing framework used by Android is based on JUnit[5], and normal JUnit tests can be created if desired. To get access to the database in the tests, we need access to the `contentResolver` through the `context` object belonging to an activity[11]. We use `AndroidTestCase` which gives access to the context of the activity running the tests[2].

Each test is run in three stages:

1. Call `setUp`
2. Call the test method
3. Call `tearDown`

The first stage of a test is running the setup method belonging to it. *Code 4.3* shows how we setup the test of the `pictogramController` by adding two pictograms to the database, and make sure they are inserted using asserts. If the pictogram is inserted correctly using `insertPictogram`, the method will return the index at which the pictogram has been inserted into the database, and if it failed the method will return -1. Shown in lines 7 and 8 we check whether the pictograms are inserted by checking the value returned.

Code 4.3: During the test of interaction between the `pictogramController` we insert pictograms into the database, and check if they are given a position.

```
1 @Override
2 protected void setUp() throws Exception{
3     super.setUp();
4     pictogramController = new PictogramController(this.getContext());
5     pictogram1 = new Pictogram(name, pub, defaultBlueBitmap, sound_data, inline_text,
          author);
6     pictogram2 = new Pictogram(name + "2", pub, defaultBlueBitmap, null, inline_text,
          author);
7     assertTrue(pictogramController.insertPictogram(pictogram1)>0);
8     assertTrue(pictogramController.insertPictogram(pictogram2)>0);
9 }
```

In the second stage we start testing the methods of our classes and their methods. *Code 4.4* is a code snippet of how we test `modifyPictogram`. When the goal of the method is to modify the data in the database so it consistent with the changes to the modified pictogram object, we want to test if the different aspects of the pictogram, like the name, are changed correctly. In lines 2 through 5 we set the different values of `pictogram1`. The update to the entry in the database happens in line 7, where `modifyPictogram` is called with the pictogram we want to update as parameter. The entry in the database will then be updated accordingly if the method runs correctly.

In line 8 we load the modified pictogram from the database, which we then compare to the object we updated before, in line 10 through 13. In line 14 throught 17 we make sure the values are different from the original values of `pictogram1`.

Code 4.4: We want to make sure that the modify method of `pictogramController` works correctly.

```
1  public void testPictogramModify() throws Exception{
2      pictogram1.setImage(defaultRedBitmap);
3      pictogram1.setName("test image changed");
4      pictogram1.setInlineText("Changed test inline text");
5      pictogram1.setIsPublic(false);
6
7      pictogramController.modifyPictogram(pictogram1);
8      Pictogram pictogramChanged = pictogramController.getPictogramById(pictogram1.getId());
9
10     assertEquals(pictogram1.getName(), pictogramChanged.getName());
11     assertEquals(pictogram1.getInlineText(), pictogramChanged.getInlineText());
12     assertEquals(pictogram1.getIsPublic(), pictogramChanged.getIsPublic());
13     assertTrue(equals(pictogram1.getImage(),pictogramChanged.getImage()));
14     assertFalse(pictogramChanged.getName().equals(name));
15     assertFalse(pictogramChanged.getInlineText().equals(inline_text));
16     assertFalse(pictogramChanged.getIsPublic() == (pub == 1));
17     assertFalse(equals(pictogramChanged.getImage(),defaultBlueBitmap));
18 }
```

The third stage is all about removing the footprint made in `setup` or during the test. *Code 4.5* shows how the `tearDown` method works. Since we added two pictograms to the database, we want to remove them when we are done, to make sure that other tests will start with a clean database.

Code 4.5: In `tearDown` we clean up after ourselves.

```
1 @Override
2 protected void tearDown() throws Exception{
3     super.tearDown();
4     pictogramController.removePictogram(pictogram1);
5     pictogramController.removePictogram(pictogram2);
6 }
```

The main goal of the tests created during sprint 2 were to ensure that the basic methods such as `insert`, `get` and `modify` work.

4.4 Oasis

The Oasis application lets guardians manage profiles, departments and applications. The application allows for creating new citizens, parents and guardians as well as creating new departments and modifying which department a profile belongs to.

The role of the Oasis application is much similar to the online GIRAF administration tool. However, the GIRAF system is meant to be fully working while offline as guardians may go on excursions with citizens and therefore the Oasis application is required.

Because the Oasis application may be installed on multiple devices and because there is an online administration tool which has functionality similar to the one of the Oasis application, it is important that any changes made to the data are synchronized to other devices as well as the online administration tool. There is a group working with the central database that is allocated to come up with a tool for synchronizing the data and it is therefore not within our scope.

4.4.1 Previous Application

Because the previous Oasis application was not updated in 2013, there were a lot of work to do to make it compatible with all the changes that have been made to the database schema and OasisLib. In belief that a native application provides the best user experience and performance it was decided to focus on developing a native application rather than looking into porting the web version. We chose to rebuild the

application from the ground to make it use the new updated APIs in OasisLib as well as rebuild the interface to make the primary functionality easily accessible.

It was decided to collaborate with the group who continued the development of the online administration tool so that the requirements for the applications as well as knowledge about the users and their needs could be shared between the groups.

4.4.2 Requirements

The requirements for the new Oasis application is created partly by taking the requirements presented the report on the development of the previous Oasis application[8] and partly by using the information provided in an interview with Birken conducted by group sw606f14. Please refer to section 7 for more information on how the requirements were found.

The requirements for the Oasis application are:

1. Guardians must be able to log in using the central log in provided by the Launcher application.
2. Guardians must be able to create, edit and delete profiles for citizens.
3. Guardians must be able to create, edit and delete profiles for parents and guardians.
4. Administrators must be able to create, edit and delete profiles for administrators.
5. Administrators must be able to create, edit and delete departments.
6. Administrators must be able to manage which department a profile belongs to.
7. Developers must be able to add test data.
8. Children should be called “Borger” and parents should be called “Værge”.
9. In the relation between a guardian and a citizen, the guardian should be called “Kontaktpædagog”.
10. No private information on profiles.
11. GIRAF components should be used to ensure that the application look similar to other applications in the GIRAF ecosystem.

If you look in the report written by group sw604f12 who did previous Oasis application, you will find requirements related to management of media. These requirements are not included as Birken did not mention this in the interview and it was obvious that we had to cut something from the original requirements due to time limitations as we started developing the new application in the second sprint. The interview with Birken also states that when a parent logs in they should be presented with a special screen showing just the information they would need. Birken was unsure what should be presented to the parents and therefore this was not included in this project.

The requirement that says developers must be able to add test data does not exist in the report on the development of the previous Oasis application but the previous application did have this functionality. We quickly found that other groups working on the GIRAF project as we ourselves relied on this test data so we added it as a requirement that when the application is in development the developers must be able to edit test data to the local database.

The list of requirements states that GIRAF components should be used. The components forms a library of graphical user interface elements that are used throughout all applications in the GIRAF ecosystem. This ensures that all applications look familiar and thus should be easier to use for the users. At the beginning of the project it was decided on a meeting between all the groups to include this requirement for all applications in the GIRAF ecosystem to ensure a uniformity.

When reading the requirements, it is important to note that there are four different roles in the GIRAF system. The roles can be considered being in a hierarchy meaning that the higher the role is in the hierarchy the more functionality you have access to. There hierarchy is as follows:

- Administrator
- Guardian
- Parent
- Citizen

The hierarchy must be read as follows: administrators can do everything that guardians, parents and citizens can do as well as things that are specific to the administrator role. Guardians can do everything parents and citizens can do as well as a few other things. Parents can do anything that citizens can do and a few other things. Citizens does not inherit functionality from any other roles and thus their access is limited. In fact, citizens do not have access to the Oasis application.

This means that when the requirements says that guardians must be able to do something, administrators must also be able to do the same thing.

Please note that the administrator role is not currently used in the GIRAF system because the Launcher application does not support this role. As later described in section 4.6.3, the Launcher application passes the ID of a guardian, a citizen or both to the applications that use the central log in. The Launcher application is thus not able to pass the ID of an administrator. This issue was discussed with the group developing the Launcher application who explained that they will not have time to add support for administrators this semester. Therefore, we will allow guardians to perform the actions of an administrator. When students are working on this project the next time, they should consider it crucial to add support for administrators and make the required changes to the Oasis application to only allow administrators to perform the actions that a guardian should not be allowed to perform.

4.5 Additional Requirements For the Oasis Application

The requirements listed in 4.4 were the basis for the development of the Oasis application. During the development of the application we found some requirements that should also be fulfilled. These requirements are treated equally with the requirements in 4.4 but where the those were primarily formulated as very short user stories the requirements found during the development are more technical.

The requirements were often found as a result of fulfilling the requirements in 4.4.

For each sprint the additional requirements found in that sprint will be presented before describing how the requirements were fulfilled. During the second sprint the following requirements for the Oasis application were found.

1. Nested fragments must be used for greater reusability of code and layouts
2. A menu that provides easy access to the functionality of the application

The requirements are further explained in the following section.

4.6 Requirement Fulfillment For the Oasis Application

The following sections describes how the requirements for the Oasis application were fulfilled. The requirements will be described one at a time and for each requirement the implementation and the design that was used to fulfill the requirement will be described. This section will only describe the requirements that were fulfilled during the second sprint. Please refer to chapters 5 and 6 for more information on the rest.

4.6.1 Nested Fragments

Nested fragments are used to create split views in the app, but instead of creating a split design in a single XML layout file we can have two seperate XML layout files that we can change independently. An example of the use of split view can be seen in figure 5.6a where one fragment displays a list of departments while the other displays fields with information about the selected department.

The way nested fragments are implemented is shown in listing 4.6. The first line simply creates a new `EditDepartmentFragment` which is the fragment that is to be nested, and the one shown on the right hand side in figure 5.6a. The second line is the interesting one. First off it uses the `getChildFragmentManager` method to get a `FragmentManager` that is used to manage fragments and perform transactions. Transactions are operations performed on fragments, such as add and remove. Nested fragments were not introduced to Android until API level 17 but unfortunately Oasis uses level 15. The way we got around this was by using the Android Support library which offers newer features for older APIs. This means that instead of using the native fragments the app uses support fragments.

Now that we have a `FragmentManager` a transaction is started that replaces the `FrameLayout` holding the id `edit_frame` with the fragment created in line one. A `FrameLayout` is a class designed to hold views and is used to store the nested fragments in Oasis.

Code 4.6: Implentation of a nested fragment

```
1 final EditDepartmentFragment editDepartmentFragment = new EditDepartmentFragment(
      departmentList.get(position));
2 getChildFragmentManager().beginTransaction().replace(R.id.edit_frame,
      editDepartmentFragment).commit();
```

4.6.2 Creation, Modification and Deletion of Profiles For Citizens

Registering new profiles for citizens is necessary for guardians. When a guardian gets a new citizen, that is, he or she becomes the guardian of the citizen, the citizen must be registered in the database. We also allow the guardians to edit profiles as changes may be needed. E.g. if a citizen changes department[1], name or simply because an error occurred when adding the citizen. Citizens can also be deleted as they may no longer be associated with Egebakken or Birken.

The follow requirement is treated in this section.

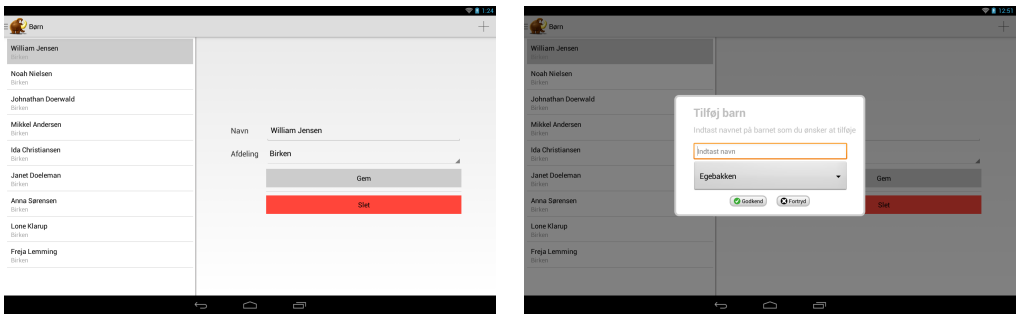
[1] Changing the department was not added until sprint three. Please see seciton 5.8.3.

Guardians must be able to create, edit and delete profiles for citizens.

Figure 4.4 shows the interfaces the user are presented with when editing profiles for citizens. Please note that the interface in the figures allows for managing the department a profile belongs to. This feature was not added until sprint three as described in 5.8.3.

The interface in figure 4.2a shows the list of citizens for the guardian currently logged in. Selecting a citizen in the left side will allow editing the name and department of the citizen in the right side of the screen. When selected, a citizen can also be deleted by pressing the red button.

Clicking the plus in the upper right corner will present the dialog shown in figure 4.2b. It was chosen to use this dialog for adding citizen because it removes focus from the background context in which citizens are edited and by dimming the background creates a new context that attracts the users attention.



(a) Interface presented when editing citizens

(b) Dialog presented when adding a citizen

Figure 4.2: Screen and dialog presented when editing citizens.

Dialog For Adding Citizens

The dialog is based on another dialog that is part of the GIRAF components maintained by group sw403f14.

The `GDialog` in GIRAF components is a dialog with no content but provides the `SetView` method which can be used to set the content of the view. A more common dialog is `GDialogMessage` which inherits from `GDialog` and displays a title, a description and a confirm and cancel button. This dialog, however, does not provide a way to add fields to the dialog. Calling `SetView` on an instance of `GDialogMessage` is equivalent to calling `SetView` on an instance of `GDialog` and therefore will replace all the contents of the dialog and thus remove the title, description and buttons.

When asked if there is a way to add custom views to instances of `GDialogMessage`, the advice from the graphical user interface groups was to replicate the interface of `GDialogMessage` and call the `SetView` with this interface. This solution is not satiesfiable for two reasons.

1. When the interface groups changes the layout of the title, description and buttons, the same changes will have to be made in the replicate of the layout.

2. When creating several dialogs that need different views added, the title, description and buttons will have to be added to all the different interfaces.

A solution to this issue can be found by digging in the `GDialogMessage` class. This class adds the `setContent` method which sets the title, description and if provided, an image in the custom layout that `GDialogMessage` uses. This method is called from the constructors of `GDialogMessage` which takes the title, description and optionally and image as parameters. `setContent` will finally call `SetView` with the layout it just prepared with title and description. We wish to add the custom views between the description and the buttons and as such we should modify this layout in `SetView`.

Our implementation of the `SetView` method must resemble the original implementation closely to ensure that the result has the same style as the other dialogs in the GIRAF components library. The original implementation is shown in *Code 4.7*. Four things happen in this implementation.

1. The `gdialog_layout` view is created. This is the base layout for the dialog.
2. The private method `SetStyle` is called. This applies a background color, corner radius and stroke to the layout.
3. The `content` view that was passed as input parameter is added to the layout.
4. `setContentView` that is implemented by `Dialog` that `GDialog` inherits from, is called. This sets the custom view in the dialog.

Our subclass of `GDialogMessage` is called `OasisContentDialogMessage`. It will have to do the four steps above. The implementation of `SetView` in `OasisContentDialogMessage` can be seen in *Code 4.8*. The mplementation can be divided into the following five steps where the first four are similar to the previous four steps. Please see the comments in *Code 4.7* for references to the below steps.

1. Exactly as in the original implementation, the `gdialog_layout` is created.
2. The `setSyle`[2] method is private but we have to call it with the layout as parameter to ensure the correct style is applied. Using Javas reflection API we can modify the protection of a method on runtime. We get the method using `getDeclaredMethod`, store it in `setStyleMethod` and set it to publicly accessible. Calling `invoke` on `setStyleMethod` with the layout as argument is equivalent to the `setStyle` call in the original implementation.
3. The content view is added to the layout.
4. The layout is set as the content view.
5. Adding the custom content view happens in step five. This is what differentiates our implementation from the original implementation. First, the layout parameters that places the custom content view correctly below the description and the buttons are created. Next the custom content view is created by calling `createContentView`, a method that superclasses must implement to create and return their custom view. Finally the custom content view is then added to the `content` view passed as input argument by `setContent` in `GDialogMessage`.

[2]Please note that the typo is intentional. The method is created by the interface group and there is a typo in the method name.

Code 4.7: Implementation of `SetView` in `GDialog`

```
1 public void SetView(View content) {
2     View layout = LayoutInflater.from(this.getContext()).inflate(R.layout.gdialog_layout,
          null);
3     SetSyle(layout);
4     RelativeLayout wrapper = (RelativeLayout) layout.findViewById(R.id.GDialog_ViewWrapper)
          ;
5     wrapper.addView(content);
6     this.setContentView(layout);
7 }
```

Code 4.8: Implementation of `SetView` in `OasisContentDialogMessage`

```
1  public void SetView(View content) {
2      // 1
3      View layout = LayoutInflater.from(getContext()).inflate(dk.aau.cs.giraf.gui.R.layout.
           gdialog_layout, null);
4
5      // 2
6      Method setStyleMethod = null;
7      try {
8          Class[] args = new Class[1];
9          args[0] = View.class;
10         setStyleMethod = GDialog.class.getDeclaredMethod("SetSyle", View.class);
11         setStyleMethod.setAccessible(true);
12     } catch (NoSuchMethodException e) {
13         e.printStackTrace();
14     }
15
16     try {
17         assert setStyleMethod != null;
18         setStyleMethod.invoke(this, layout);
19     } catch (IllegalAccessException e) {
20         e.printStackTrace();
21     } catch (InvocationTargetException e) {
22         e.printStackTrace();
23     }
24
25     // 5
26     RelativeLayout.LayoutParams params = new RelativeLayout.LayoutParams(RelativeLayout.
           LayoutParams.WRAP_CONTENT, RelativeLayout.LayoutParams.WRAP_CONTENT);
27     params.addRule(RelativeLayout.RIGHT_OF, R.id.GDialogMessage_image);
28     params.addRule(RelativeLayout.BELOW, R.id.GDialogMessage_description);
29     params.topMargin = 20;
30     params.width = 400;
31
32     if (contentView == null) {
33         contentView = createContentView();
34     }
35
36     if (contentView != null) {
37         RelativeLayout contentLayout = (RelativeLayout) content.findViewById(R.id.
               GDialogMessage_topWrapper);
38         contentLayout.addView(contentView, params);
39     }
40
41     // 3
42     RelativeLayout wrapper = (RelativeLayout) layout.findViewById(dk.aau.cs.giraf.gui.R.id.
           GDialog_ViewWrapper);
43     wrapper.addView(content);
44
45     // 4
46     setContentView(layout);
47 }
```

Using the Dialog For Creating A Citizen

The previously described class `OasisAddProfileDialogMessage` for displaying a dialog with a custom content view was subclassed to `OasisCreateUserDialogMessage`, the dialog presented when adding a new citizen.

Listing 4.9 shows the implementation of `presentAddChildDialog` which is called pressing the plus icon in the upper right corner in figure 4.2a. The implementation of `presentAddChildDialog` is quite simple. When the confirm button is clicked, the `onClick` method of the `OnClickListener` is called. We check if the user has entered a name, if not, we display an alert dialog. If the information is valid, we call `addChild` which stores the citizen in the database. The implementation of `addChild` is shown in listing 4.10.

Code 4.9: Implementation of `presentAddChildDialog`

```
1  private void presentAddChildDialog() {
2      final OasisCreateUserDialogMessage addDialog = new OasisCreateUserDialogMessage(
3              getActivity(),
4              getString(R.string.children_add_dialog_title),
5              getString(R.string.children_add_dialog_message));
6      addDialog.setCancelable(false);
7      addDialog.getRoleSpinner().setVisibility(View.GONE);
8
9      final EditText nameField = addDialog.getNameField();
10     addDialog.setOnClickListener(new View.OnClickListener() {
11         @Override
12         public void onClick(View view) {
13             String name = nameField.getText().toString();
14
15             String alertTitle = null;
16             String alertMessage = null;
17
18             if (name == null || name.length() == 0) {
19                 alertTitle = getString(R.string.children_add_dialog_missing_name_title);
20                 alertMessage = getString(R.string.children_add_dialog_missing_name_message)
                       ;
21             }
22
23             if (alertTitle != null || alertMessage != null) {
24                 GDialogAlert alertDialog = new GDialogAlert(
25                         getActivity(),
26                         alertTitle,
27                         alertMessage,
28                         new View.OnClickListener() {
29                             @Override
30                             public void onClick(View view) {
31
32                             }
33                         });
34                 alertDialog.show();
35             } else {
36                 addChild(name);
37                 addDialog.dismiss();
38             }
39         }
40     });
41     addDialog.show();
42 }
```

Storing Citizens In the Database

When storing the citizens in the database, it is only necessary to create a profile. No user is required for the citizen as they should not be able to log in. This does not only apply to the Oasis application. There are no applications which the citizens should be able to log into. According to the interview conducted by SW606 and shown in *appendix B* citizens will always be supervised by a guadian when using the tablet. The

guardians will log in, find the application the citizens should use and hand the tablet to the citizen.

The code that stores the profile in the database is shown in *Code 4.10*. Lines 6-18 that store the profile in the database are very straight forward. Notice on line eight we set the address of the profile to the empty string as the address cannot be null in the database but according to requirement number 10, no private information can be stored on profiles which includes the address.

We found that persisting changes to the database may block the UI for an unacceptable amount of time when performed on the UI thread. Therefore we do all persisting to the database in the background, hence the `OasisAsyncTask`.

The `OasisAsyncTask` is a simple subclass of `AsyncTask` that performs operations in the background and returns on the UI thread. The `OasisAsyncTask` abstracts away the generic types and the `onPreExecute` and `onProgressUpdate` callbacks of `AsyncTask`. This limits its applicability but makes it ideal for performing simple and short operations in the background.

Code 4.10: Storing citizens in the database

```
1  private void addChild(final String name) {
2      new OasisAsyncTask(new OasisAsyncTask.OasisAsyncTaskHandler() {
3          boolean success = true;
4
5          @Override
6          public void perform() {
7              Profile profile = new Profile();
8              profile.setRole(Profile.Roles.CHILD);
9              profile.setAddress(""); // Address cannot be null but we may not store
                   addresses
10             profile.setName(name);
11
12             if (profileController.insertProfile(profile) == -1) {
13                 Log.e("OasisApp", "Could not insert profile");
14                 success = false;
15             } else {
16                 profileController.attachChildToGuardian(ProfileManager.getProfile(),
                       profile);
17                 success = true;
18             }
19         }
20
21         @Override
22         public void didFinish() {
23             if (success) {
24                 reloadChildren();
25             } else {
26                 GDialogAlert alert = new GDialogAlert(
27                         getActivity(),
28                         getString(R.string.children_add_failed_title),
29                         getString(R.string.children_add_failed_message),
30                         new View.OnClickListener() {
31                             @Override
32                             public void onClick(View view) { }
33                         });
34                 alert.show();
35             }
36         }
37     }).execute();
38 }
```

4.6.3 Logging in through the Launcher application

Guardians and administrators must be able to log in using the central log in provided by the Launcher application.

The requirement is that admins and guardians should be able to log in through the Launcher instead of inside the Oasis application. This is similar to how all the other applications work as they log in through the Launcher as well.

To allow guardians to sign in using the Launcher application, the id of the guardian is sent in an intent from the Launcher to Oasis. This intent may contain either guardian id, citizen id, or both, but since citizens are not allowed to use this app we have to make sure that a guardian id is present. If there is no guardian id, then the profile is invalid. If the intent is null we check whether or not the application is a debug release by checking the BuildConfig.DEBUG variable. BuildConfig.DEBUG is a static final boolean which is normally set to true, but will change to false when a project is exported as a signed release build. In the case that it is a debug build the application signs in with a debug profile as no real profile was given from the Launcher application.

4.6.4 Adding test data

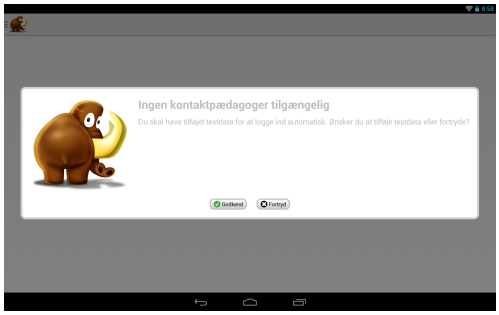
The requirement is that developers of the GIRAF project must be able to add test data to the local database by using the Oasis application. This is because all of the developers need similar data to work with.

Developers must be able to add test data.

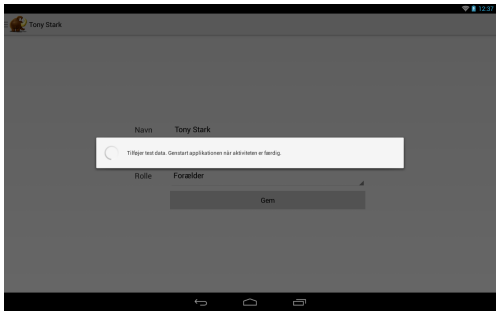
As mentioned in section 4.6.3, the app checks whether a profile is signed in or not, and if not signs in as a debug profile. The debug profile is only available if there is a profile in the local database, so if there is no such profile the app asks the user to add test data and restart the application. This dialog is shown in figure 4.3a.

When the test data is added the `OasisAsyncTask` described in section 4.6.2 is utilized so that all persisting to the database remains in the background. While the test data is being added the `ProgressDialog` in figure 4.3b is shown. This dialog is not cancellable as the user is not supposed to be able to do anything with the app until data has been added to the local database. The dialog asks the user to restart the app once the data is added but the app automatically shuts down once done.

The `ProgressDialog` can have two different styles, one displays a horizontal progress bar which indicates how many percent the task is done, and the other just displays a circular spinner. The horizontal progress bar was not an option because we have no way of telling how much work needs to be done and OasisLib gives no indication of its progress. One way of doing it would be to measure the average time it takes to add the test data and then have the progressbar use time as progress. This however is not a reliable way of doing it as the work time could vary between devices. Instead the circular spinner was chosen, as the animation still gives the user the impression that the app is working and that they need just wait. If there was no animation, one could get the idea that the app may have crashed but it would be hard to tell. This is another reason why we need to do all the database work in the background, because had it been done in the UI thread the spinner might not have been animated, and users could think the app had stopped.



(a) Dialog presented when no data is available



(b) Dialog presented while test data is being addded

Figure 4.3: Dialogs presented when launching the Oasis application with no data in the database and in debug mode.

4.6.5 Easy Access to Functionality Through a Menu

Early in the first sprint of working with the Oasis application we discovered that we needed to come up with easy way for the user to access the primary functionality. The requirement for this is shown below.

A menu that provides easy access to the functionality of the application

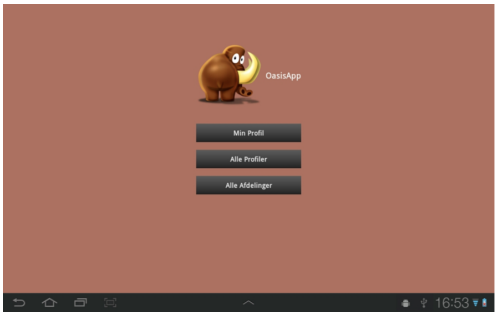
In the application that was developed in 2012 the user was presented with a menu when opening the app as seen in figure 4.4a. When pressing a menu item a screen similar to the one in figure 4.4a is shown. This particular screen is shown when the "Min Profil" menu item is selected. Notice that the menu changes location from being centered to being at the top of the screen. An extra menu item has also been added to the menu. Being convinced that changing the position of the menu and adding menu items without the user performing any action that justifies this behaviour, we chose another approach for the menu.

It was chosen to create the menu as a navigation drawer, a way of providing a way for the user to navigate the application. The navigation drawer was chosen because it is one of the navigation patterns suggested by Google[7] and can be used to provide a great overview of all of the functionality in the application. The result can be seen in figure 4.5.

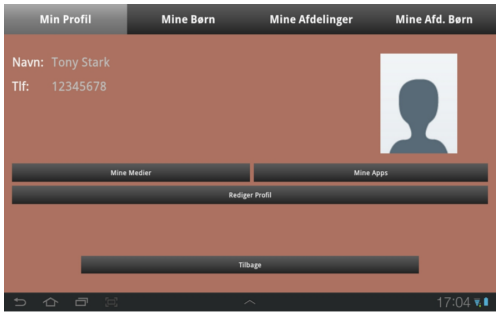
The menu items are grouped into sections to give a better overview. Each section has a colored headline while the menu items are black. There is one exception though, the profile name at the top serves as both a section header and menu item to the profile of the user currently logged in.

The menu shows the name of the user currently logged in. This ensures that the user always know which profile he is logged in as. Showing the profile name in the menu caused an issue. When a profile name was changed, this change would have to be reflected in the menu.

This issue can be solved by utilizing the `LocalBroadcastManager`. When a profile is saved we send out a broadcast as seen on line 12 in listing 5.5. Any object can subscribe to this broadcast and when it is sent out, they will be notified that the profile has changed. The `MainActivity`, which is responsible for providing the contents of the menu, registers for this broadcast and updates the profile name when it is received.



(a) Main screen in the previous Oasis application developed in 2012



(b) Profile screen in the previous Oasis application developed in 2012

Figure 4.4: Screenshots of the previous Oasis application developed in 2012.

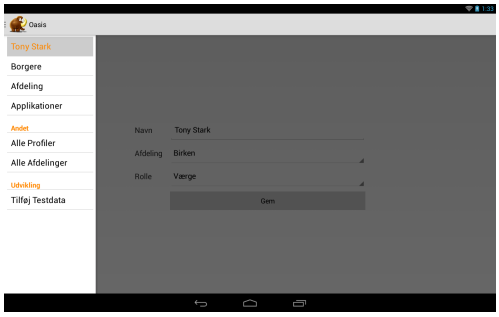


Figure 4.5: The menu in the new Oasis application

Navigation Diagram for Oasis

The navigation diagram in figure 4.6 illustrates how the Oasis application is navigated. The navigation diagram shows that all the functionality of the application is easily reached from the menu. This supports the previously established point that the primary functionality of the application must be aasily accessible. All arrows go both ways as any operation which changes the context of the application is easily cancelled and thus the user is brought back to the context he came from. The three views “Add citizen”, “Add profile” and “Add department” are dialogs that are presented modally when the user adds either a profile or a department. These dialoags are further edescribed in sections 4.6.2, 5.8.2 and 5.8.4.

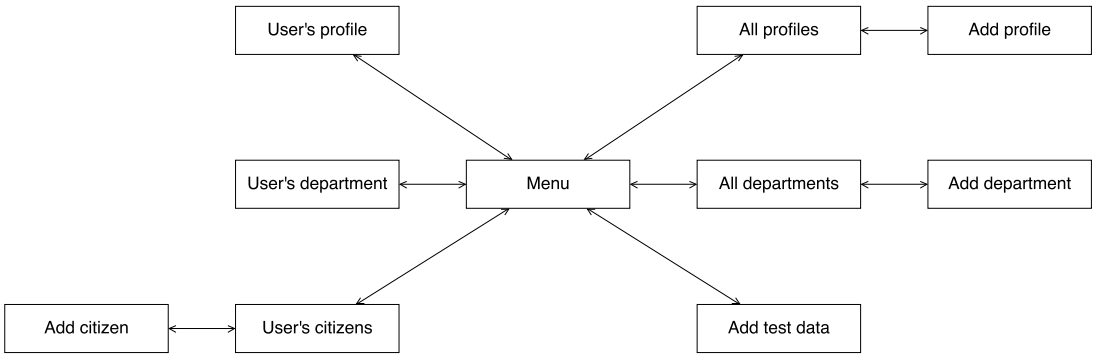


Figure 4.6: Navigation diagram for the Oasis application

5 Sprint 3

From April 15th to May 7th we were working on sprint 3 in which we focused our efforts once again on refactoring and improving OasisLib and LocalDB while continuing the rewrite of Oasis.

To summarize the work done in this sprint, we removed unneeded and redundant code from LocalDB, extended the database design to accommodate the sequence features requested by multiple groups, implemented transaction support in LocalDB, enabled SQLite foreign key constraints and triggers, changed the OasisLib API slightly and implemented profile and department handling in Oasis.

5.1 LocalDB Refactoring

While adding the n-n relations described in *4.1 Implementing N-N Relations In LocalDB p. 21* we quickly realized there was a lot of code duplication both within LocalDB but also between OasisLib and LocalDB.

LocalDB was all in all an unmaintainable mess of four switch statements each with 40 cases of which many only did slightly different things, a short example of this is shown in *appendix F* where it should be noted how the `setTables()` and `setProjectionMap()` calls are duplicated multiple times with very little variation in the arguments.

Furthermore there was a lot of duplicated string constants in both projects used to describe the structure of the database which is a bad idea because it is error prone having to change information about tables in many different places. This was solved fairly easily by extracting the string constants into a submodule and adding it as dependency.

The interesting part is that we looked at what both OasisLib and LocalDB did and realized the needs of both could be satisfied with a simple interface defining the three methods *getPath()*, *getTable()* and *getSelection()* and an object describing the table being accessed.

Thus we ended up designing a couple of classes and enums to describe how a SQL table looks as seen in figure 5.1. Since SQL statements are clearly defined this approach also had the added benefit of being able to automatically generate SQL create, drop and join statements from the objects representing a table.

Previously the create statements were manually crafted by concatenating together the needed SQL syntax with each column's name and type along with primary and foreign keys which yielded a very long and hard to read line of java code for each table. With the new design this could be done by simply calling `getSQLCreateString()` and telling the SQLite database to execute this whenever the database needed to be (re)created.

Of of the consequences of the new design is the removal of the switch cases since we could now put a reference to each `ExportDefinition` into an array and asso-

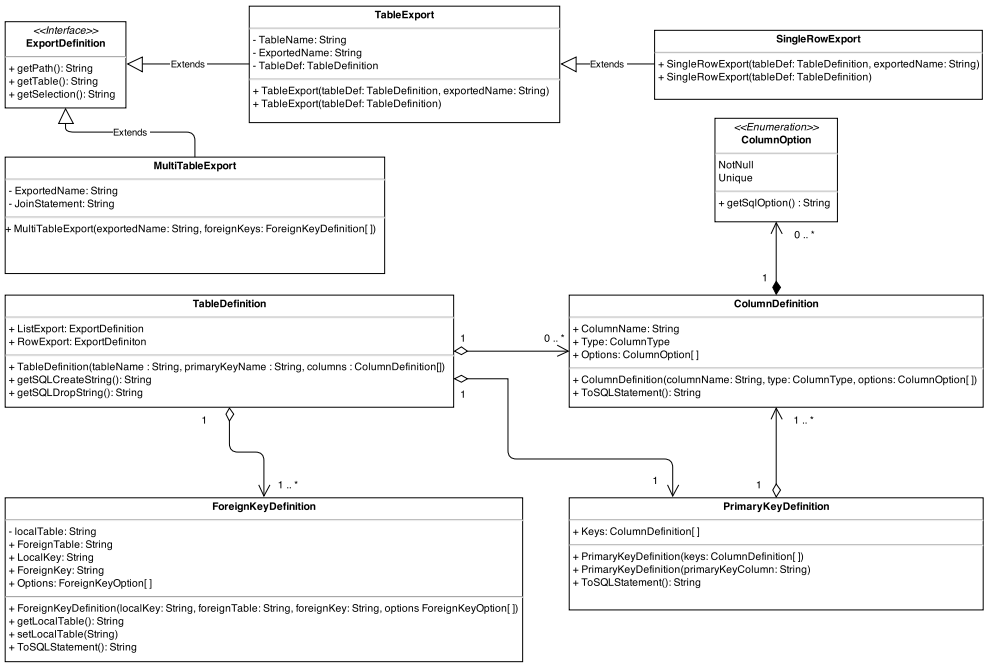


Figure 5.1: UML class diagram of the classes used to describe database tables.

ciate the index with the exported URI. This information could then, with the help of Android's `UriMatcher`, be used to translate the URI OasisLib queries back to an instance of the `ExportDefinition` interface in the form of one of the three implementations depending on what was specified to be exported for each table. The LocalDB ContentProvider can then use this specific `ExportDefinition` instance to determine whether to query one table or a set of joined tables which means that the whole process of translating URIs is a dynamic array lookup that removes the need for switches.

Code 5.1: Whole code for new and optimized query method.

```
1  @Override
2  public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs,
       String sortOrder) {
3      SQLiteDatabase db = helper.getReadableDatabase();
4      SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
5      ExportDefinition ed = matcher.matchQuery(uri);
6
7      if(ed == null)
8          throw new IllegalArgumentException("Unknown URI: " + uri);
9
10     qb.setTables(ed.getTable());
11
12     String select = ed.getSelection();
13     String limit = null;
14
15     if(select != null){
16         selection = select;
17         selectionArgs = new String[]{
18                 String.valueOf(ContentUris.parseId(uri))
19         };
20         limit = "1";
21     }
22
23     return qb.query(db, projection, selection, selectionArgs, null, null, sortOrder,limit);
24 }
```

The above design, usage and implementation is a simpler variation of the command design pattern that instead of executing actions just stores information.

All in all the whole code to implement the `ContentProvider` and related helper classes went from more than 1500 lines of code to less than 500, an example of this simplification can be seen in *Code 5.1* which contains code for the whole function.

It should be noted how stark a contrast this is to the code in *Code 4.1* that is only for a single exported table while the refactored version handles all cases in the same amount of code.

As for the metadata library and how tables are defined in Java code compared to the previous approach can be seen in *appendix G*, here it's also shown how multitable exports are defined in the library.

5.2 SQLite FOREIGN KEY Constraint

When foreign keys refer to a row in a table, it is expected that the referred row actually exists. We want the integrity of the database to be preserved when a row is inserted or removed. To keep the consistency of data we have enabled `FOREIGN KEY` constraint. `FOREIGN KEY` constraint is disabled by default in SQLite[17]. The constraint prevents invalid data from being inserted and removed[15]. For example if a `profile`, see overview of tables in *appendix D*, is inserted with the `deparment_id` pointing at a `department` that does not exist, it will fail due to `FOREIGN KEY` constraint.

When a row is deleted from a table in the database, it is important to make sure that the rows that refer to it through foreign keys are updated or deleted to keep the database consistent. To achieve this `ON DELETE` triggers are used. If a row is to be removed `ON DELETE CASCADE` is used, and if it is to be set to null, `ON DELETE SET NULL` is used. OasisLib emulated cascading before we enabled `FOREIGN KEY` constraints. This could easily cause consistency problems, if something went wrong in one of the steps that removed a row from the database. Instead of doing it ourselves we lay the responsibility on the database.

In all the tables shown in table 5.1 which are used as links in many to many relations, we use `ON DELETE CASCADE`. We do this because a link with one end missing would cause inconsistency in the database.

frame_pictogram	pictogram_tag	department_pictogram
pictogram_category	profile_application	guardian_of
admin_of	department_application	profile_pictogram
profile_category		

Table 5.1: The different tables used for N to N relations in the database as shown in *appendix E*.

Whenever a table has a foreign key pointing at the author we use `ON DELETE SET NULL` because the author is nullable, and the user that created a row can have been deleted, while we still want to keep everything the user has created.

A department from the `department` table, can have many sub departments. The column `super_department_id` which is a foreign key to a super department, is nullable because of this tree like structure. If the column is null, then the department is the root of the tree, and all the departments below it belong to it.

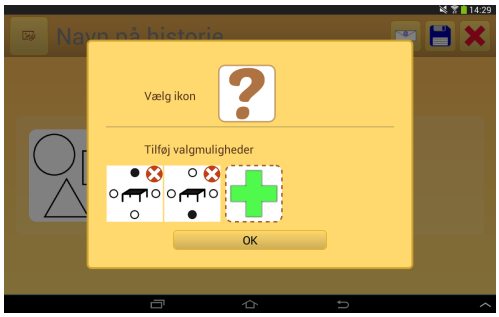
If `super_department_id` used `ON DELETE SET NULL`, and a root department is deleted, then the tree is split into many new trees with those just below the old root become their own new roots. To prevent this we use `ON DELETE CASCADE`, so that the entire tree of departments is deleted with the root.

5.3 Sequence Feature

Groups SW602, SW607, and SW615 required the ability to save sequences to the database. A sequence contains a list of frames which in turn contains a list of pictograms. They are used to show something a person does or has to do. For example if a person with autism has to use the toilet, a sequence would show what the person should do step by step.

Group SW602 has been working on the application Life Stories, shown in figure 5.2. A sequence is shown in figure 5.2a. It contains two frames, one with three geometric shapes and one with a question mark. A frame can be represented by a pictogram, which is the case in the first frame, and the second frame is represented as a question mark which represents a choice in their life story application. When there is a choice, the user can choose between the pictograms contained in the frame, shown in figure 5.2b. For example a choice could be between which toy a child has to play with.

(a) An example of how group SW602 uses sequences



(b) An example of how group SW602f14 uses frames in sequences

Figure 5.2: Screenshots of how group SW602 uses sequences in Life Stories.

5.4 LocalDB Sequence Implementation

The groups involved with the use of sequences had a meeting in which they came to an agreement of what they would need to save to a database, in order to save their different sequences as described in *5.3 Sequence Feature p. 40*. The results of the meeting were handed to us and we started implementing the sequence tables in the database.

Additions To the Database Schema

Figure 5.3 shows the entity relationship model for sequences. This model shows how the new entities *Sequence* and *Frame* relates to the existing entities *Profile* and *Pictogram*. This model only shows the entities which are related to the requirements of sequence. The model shows that each sequence has an id which is used to identify it, a sequence type which describes how the sequence is used and a name. A sequence belongs to a profile and a profile can have many sequences. A pictogram can be selected to represent a sequence.

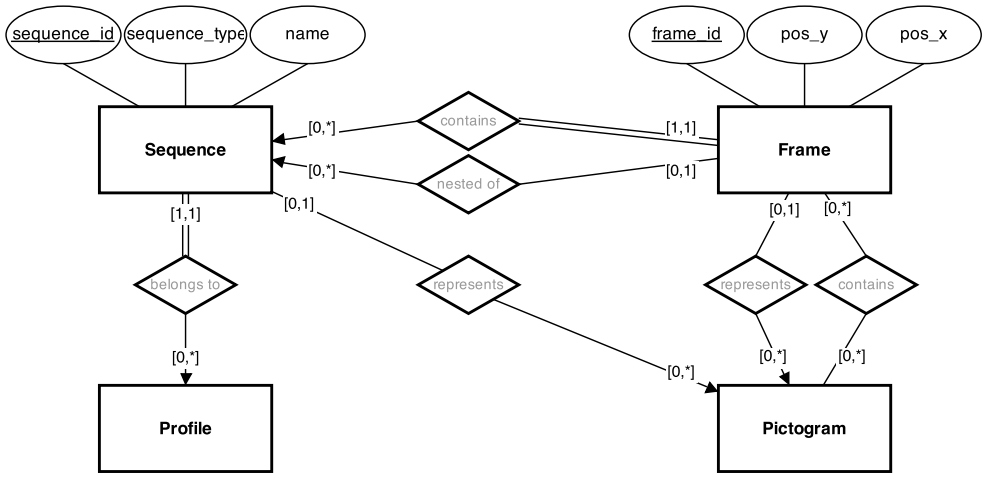


Figure 5.3: Entity relationship model for sequences and frames using Chen's and Min-Max notation.

A frame has an identifier, a position with x and y coordinate which is used to keep track of the position of a frame compared to other frames, and can contain many pictograms. Each frame belongs to a sequence, and a sequence can contain many frames. A pictogram can be used to identify a frame much the same as done in Sequences application. It is possible for a frame to contain a sequence, which allows sequences to be nested in another sequence. There is a many to many relation between *Frame* and *Pictogram* which allows a frame to contain many frames, and many different frames can refer to the same pictogram.

It should be noted that the implementation of this new table was at first done before the refactoring described in *5.1 LocalDB Refactoring p. 37* and then translated to the new table definition design which took much less time than the original implementation since a lot less files had to be changed.

The relations shown in figure 5.3 are translated into tables and relations as shown in *appendix D*.

5.5 OasisLib Sequence Implementation

With the new tables added to the database, some of the groups involved requested the feature to read and write to the database in batches. In other words, they wanted the sequence to contain a list of frames, and the frames to contain a list of pictograms. When they read or write a sequence they want all the nested lists to be applied to the database as well. To accomodate this request we have implemented some features using transactions, to allow the use of backtracking if something goes wrong. For example if a frame references to a pictogram which is not in the database, the transaction will fail because of a foreign key constraint error, and all changes made during the transaction will be reverted.

The `applyBatch` on the database has been overridden to implement transactions. It takes a list of `ContentProviderOperation` which can be the following:

- `newInsert`
- `newDelete`

- `newUpdate`
- `newAssertQuery`

Each operation in the list is applied to the database, and if one of them fails, it is shown in the return value. The order the operations are added to the list is important when using back referencing. Back referencing is used when an item inserted or updated has a reference to another item which is not inserted to the database, but is in the list of operations. For an item to reference to something in the list, it has to reference the index of the referred item in the list. When the operations then are applied to the database, the reference will be set to the item in the referred index.

The first method inserts sequences and its frames into the database. The first item to be inserted into the list of operations is always the sequence as shown in *Code 5.2*. When a sequence has been added, we can add its frames. A frame has a reference to the sequence it belongs to, therefore back referencing as described above is used. *Code 5.3* shows the implementation of how a frame is added to the operations list, and the back reference in line 4 being set to sequence which lies in index 0.

Code 5.2: The code snippet showing how a `newInsert` operation is created with with the values of the sequence to be inserted, and then added to the `ContentProviderOperation` list.

```
1 contentProviderOperations.add(
2   ContentProviderOperation.newInsert(SequenceMetaData.CONTENT_URI)
3     .withValues(getContentValues(sequence))
4     .build()
5 );
```

Code 5.3: An example of how a `newInsert` operation is created with back referencing to the sequence which is added in position 0 of the list.

```
1 contentProviderOperations.add(
2   ContentProviderOperation.newInsert(FrameMetaData.CONTENT_URI)
3     .withValues(FrameController.getContentValues(frame))
4     .withValueBackReference(FrameMetaData.Table.COLUMN_SEQUENCEID, 0)
5     .build()
6 );
```

Adding the pictograms belonging to a frame, requires the index of the inserted frame to be stored due to back referencing. This is done by saving the position in a `HashMap` with the frame as the key, and the index of the frame as the value. The actual insertion of a `frame_pictogram` which is the link between `frame` and a `pictogram`, is done the same way as when inserting a frame, just with the index of the frame as the reference id instead of 0.

The second method that was used to modify a sequence, its frames and their pictogram connections. This method uses transaction as well. Since the speed of insertions when using transactions in SQLite is limited by the speed at which the data is inserted to the secondary storage, because it waits for the data to be safely stored to the storage[16], we want to reduce the amount of insertions we do. Figure 5.4 shows the concept of what is done when a sequence is modified.

- With the black circle representing the set of frames before the modifications
- The white circle representing the frames in the modified set
- The grey set being the intersection of the old frames and the new frames

First we compare the set of frames saved on the database and the new modified set, and remove all frames lying in the black part of the figure. Next we go through the frames which lie in the grey set which are neither added nor deleted, but they may still have been modified. If a frame has been modified we add a `newUpdate` to the list of operations. To keep track of whether a frame has been changed, we keep a boolean value updated. It is set to true whenever an operation which changes the frame is performed, and set to false when it has either been saved to the database or read from the database.

All frames lying in the white set are inserted into the database.

As explained at the start of this section, a frame can have many pictograms. Almost the same concept of removing and adding, as described for a sequence, is used in frames for pictograms. We remove all the `frame_pictograms` in the old set, and insert all the `frame_pictograms` in the new set. There is no need to have an updating part of this method.

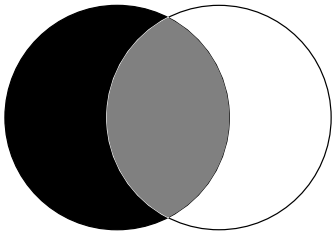


Figure 5.4: Two sets and their intersection.

5.6 Additional Tests To OasisLib

As described in *4.3 Test of OasisLib p. 23* we test the library using integration tests. The major focus of tests made during this sprint was the new features described in *5.5 OasisLib Sequence Implementation p. 41*. Many things can go wrong when working with batch insertions. First of all we created tests to make sure that all the frames belonging to a sequence are inserted when the sequence is inserted. And then that the pictograms belonging to each frame have a `frame_pictogram` inserted each. We had to make sure the right `frames` or `frame_pictograms` are removed during a modify, and that the cascading work when deleting a `sequence`.

5.7 Additional Requirements For the Oasis Application

Additional requirements were found during the development of the application in sprint 3. The following additional requirements were found.

1. Whenever a list is presented to the user it must clear to the user which item is selected
2. There must be a homogenous way of providing a menu in a fragment

The requirements are further explained in the following section.

5.8 Requirement Fulfillment For the Oasis Application

The following sections describe how the requirements for the Oasis application were fulfilled in sprint 3.

5.8.1 Show Selected Item In List View

When adding the lists of citizens as seen in figures 4.2a and 5.6a in which the user will select an item and have information for the selected item displayed on the right side, it was clear that it was a poor user experience that the list did not show which item was selected. Therefore the following requirement was created.

Whenever a list is presented to the user it must clear to the user which item is selected

It was decided to change the background color of selected rows in all lists in the app. The standard way of changing a background color based on the state a view is in (e.g. highlighted and selected) is to use drawable resources that provide different colors for different states and set this resource as the background of the row. After struggling with this for several days with little luck of getting the selection to work, we decided to create our own implementation of the selection.

The `OasisSelectionAdapterProxy` has the central role in the list selection. An instance of the proxy is added as an attribute to any adapter that should support selection. The proxy will keep track of which row is selected and when called, make sure that a row has the decided apperance for its state.

Amongst others, the `OasisSelectionAdapterProxy` implements the two methods `getView` and `setSelectedPosition` shown in *Code 5.4*. The `getView` method of `OasisSelectionAdapterProxy` should be called from the the `getView` of the `Adapter` class in the standard Android widget API. The method is called with the view to be configured and the current position and the proxy will check if the position matches the selected position and set the background accordingly.

Code 5.4: `getView` and `setSelectedPosition`

```
1  public View getView(int position, View view) {
2      if (position == selectedPosition) {
3          view.setBackgroundDrawable(selectedBackground);
4      } else {
5          view.setBackgroundDrawable(deselectedBackground);
6      }
7
8      return view;
9  }
10
11 public void setSelectedPosition(int position) {
12     selectedPosition = position;
13     adapter.notifyDataSetChanged();
14 }
```

The `setSelectedPosition` is called when a row should be selected. Note that the position is stored in the `selectedPosition` variable and thus the proxy only supports selection of a single row.

The constructor of the `OasisSelectionAdapterProxy` takes the adapter as input parameter so the adapter can be notified of changes in `setSelectedPosition`. This will invoke the `getView` method of the adapter for all visible rows and thus the `getView` method of the proxy will be invoked and the backgrounds of the rows will be updated.

As previously mentioned, the adapter is supposed to reference the proxy which stores a reference to the adapter. That is, the objects point to each other to form a cycle. Such references are considered dangerous in some languages as the objects will never be released from memory but this is not the case with Java. The garbage collector checks if there exists a path to the objects and if not consider them as garbage. This means that as long as there does not exist a path to the cycle, the objects are considered garbage.

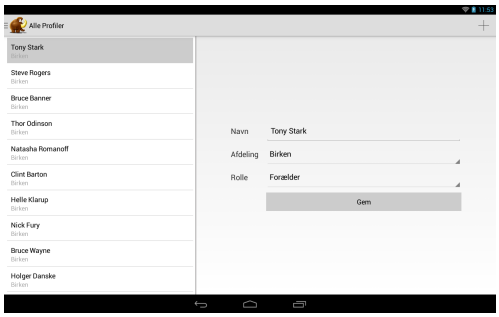
5.8.2 Creation, Modification and Deletion of Profiles

Guardians must be able to create, edit and delete profiles for parents and guardians.

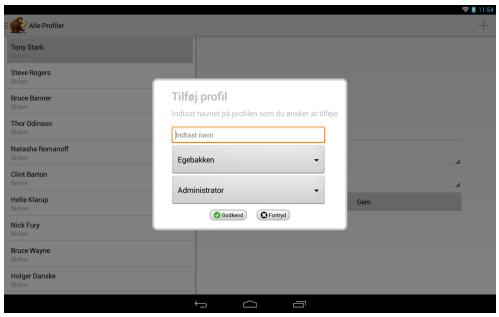
The creation, modification and deletion of profiles is much similar to the creation, modification and deletion of citizens described in *4.6.2 Creation, Modification and Deletion of Profiles For Citizens p. 28*.

The interface presented when editing profiles can be seen in figures 5.5a and 5.5b. When used to the interface for editing citizens, the interface for editing profiles will be very familiar. This is by choice as the user should spend as little time as possible doing a task, therefore a familiar interface is an advantage.

When the plus icon in the upper right corner is pressed, the `presentAddProfileDialog` method is called. The implementation of this method does not differ much from the one of `presentAddChildDialog`. When adding a profile for a parent, guardian or an administrator we also take the role into account. For citizens, the role is automatically set to child but when adding another profile the user will have to specify the role. Therefore we make sure that both the name and the role is present and if not, an alert dialog is displayed. If the validation passed, the `addProfile` method is called and the profile is added to the database.



(a) Interface presented when editing profiles



(b) Dialog presented when adding a profile

Figure 5.5: Screens presented when editing a profile.

The implementation of `addProfile` does not differ much from the one of `addChild` shown in *Code 4.10*. However, as parents, guardians and administrators should be able to log into the GIRAF ecosystem through the Launcher application, they must have a `user` associated with them. Therefore we associate the profile with an instance of `User` when creating a profile for any person who is not a citizen. When persisting multiple entities to the database, we must make sure that both entities are persisted. If the user was successfully added but the profile could not be added for some reason, we must delete the user again. We check for this and perform the ncessary actions on lines 25-30.

The requirement described in this section also covers the following requirement:

Administrators must be able to create, edit and delete profiles for administrators.

This is because the administrator role is not yet used in this project, as explained in *4.4 Oasis p. 25*. Instead of allowing the administrators to handle profiles when they can't use the app, the guardians have been granted the same power as administrators.

5.8.3 Managing Which Department a Profile Belongs To

In sprint two we added functionality for adding, editing and deleting citizens profiles but we did not have enough time to fulfill requirement number six that states that administrators should be able to change the department of a profile.

Administrators must be able to manage which profile a department belongs to.

The functionality for this was added to both the management of citizens and profiles at the same time. The interface can be seen in figures 4.2a and 4.2b for the citizens and in figures 5.5a and 5.5b for the profiles of parents, guardians and administrators.

To manage the department, a spinner was added to both the `OasisAddProfileDialogMessage` which is the dialog being displayed when adding a profile and to `EditProfileFragment`, the fragment displayed and responsible for editing a profile. When a profile is being added or edited the list of departments are loaded from the database and presented in the spinner.

`EditProfileFragment` is used whenever a profile is edited. It is shown on the right side in figures 4.2a and 5.5a. When the "Gem" button is pressed, the changes are saved by calling the `saveProfile` method in `EditProfileFragment`. The implementation of this method is shown in *Code 5.5*. This method was updated to store the selected department as seen on lines 3 and 7.

Code 5.5: Implementation of `saveProfile`

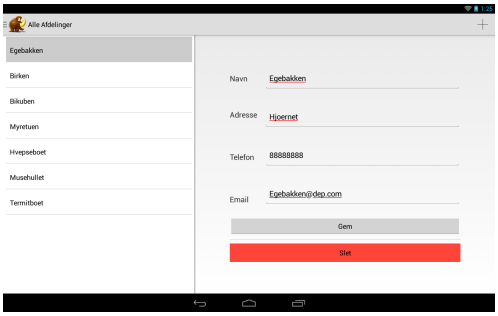
```
1  private void saveProfile() {
2      String name = nameTextView.getText().toString();
3      Department department = departments.get(departmentsSpinner.getSelectedItemPosition());
4      Profile.Roles role = Profile.Roles.values()[rolesSpinner.getSelectedItemPosition()];
5
6      profile.setName(name);
7      profile.setDepartmentId(department.getId());
8      profile.setRole(role);
9
10     profileController.modifyProfile(profile);
11
12     LocalBroadcastManager.getInstance(getActivity()).sendBroadcast(new Intent(Constants.
           PROFILE_UPDATED_NOTIFICATION));
13 }
```

5.8.4 Creation, Modification and Deletion Of Departments

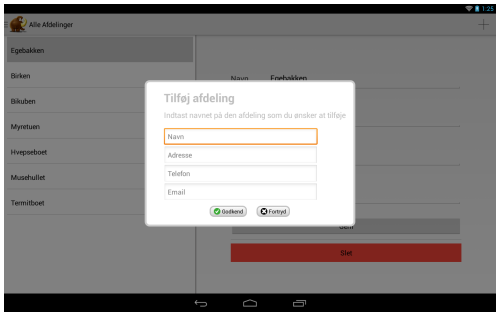
Creating new departments is necessary for admins when a new department becomes part of the ones using the GIRAF project. Modification of departments is allowed and might be needed in the case that some information was not entered correctly or in the case that a department moves.

Administrators must be able to create, edit and delete departments.

Handling of departments is done in a very similar way to handling of profiles. Figure 5.6a shows the interface which displays a list of all departments. The selected department can then have its information edited, for example in case the information was mistyped when the department was created or if the address changes. As with the profile view there is a plus sign in the upper right hand corner that displays the dialog shown in figure 5.6b. This dialog allows the user to enter all the necessary information about a department and create it.



(a) Interface presented when editing departments



(b) Dialog presented when adding a department

Figure 5.6: Screen and dialog presented when editing departments.

5.8.5 Menu Provider

Different parts of the Oasis application provide a button in the top right hand corner displayed as a plus sign. Such a button can be seen on figure 5.6a, and is in that case used to display a dialog that allows users to add a department. Such a button is called a menu and is provided through the class `MenuProviderFragment` and fragments such as the `DepartmentsFragment` extend this class. The implementation of the class is shown in *Code 5.6*. The class provides four methods, two of which the fragment extending it should override, `getMenu` and `onMenuItemSelected`. `getMenu` should be overriden to return the id of the resource file for the menu, which allows us to use different layouts for the different menu providers throughout the app. However, the two fragments that use the menu provider, use identical layouts, that contain a menu with a single item. A menu can of course contain multiple items but nowhere in the app is this done. `onMenuItemSelected` should be overriden to handle what happens when a user clicks a menu item. For the `DepartmentsFragment` it displays the dialog shown in figure 5.6b.

Code 5.6: Implentation of the MenuProviderFragment class

```
1  public class MenuProviderFragment extends Fragment {
2
3      private boolean providesMenu = false;
4
5      public int getMenu() {
6          return -1;
7      }
8
9      public boolean getProvidesMenu() {
10         return providesMenu;
11     }
12
13     protected void setProvidesMenu(boolean menuAvailable) {
14         if (menuAvailable != this.providesMenu) {
15             this.providesMenu = menuAvailable;
16             getActivity().invalidateOptionsMenu();
17         }
18     }
19
20     public boolean onMenuItemSelected(MenuItem item) {
21         return false;
22     }
23 }
```

6 Sprint 4

The last fourth sprint happened from the 7th to 20th of May and was deliberately shorter than the others to accommodate for the groups needing time at the end of the semester to write their project reports.

In *5.1 LocalDB Refactoring p. 37* we mentioned that we extracted the shared metadata about tables into a submodule, this was however only done for the LocalDB project so in this sprint we refactored OasisLib to also use this module and in the same process clean up the code to query tables and making sure we didn't miss any code that did not use the new N-N relations.

We also worked on fulfilling more of the requirements for the Oasis application as described in *6.2 Requirement Fulfillment For the Oasis Application p. 50*.

6.1 OasisLib Cleanup

Refactoring OasisLib to the new shared metadata library required changes that affected the code in all of the controller classes. However before these changes we decided to look at the kind of code duplication happening in each of the query, insert, update and delete methods and came to the conclusion that getting a `ContentResolver` from a `Context` and specifying the URI to query was the main offenders.

With this knowledge the class diagram shown in figure 6.1 was designed. It interfaces with the `ExportDefinition` classes shown in figure 5.1 which enables seamless propagation of changes to tables.

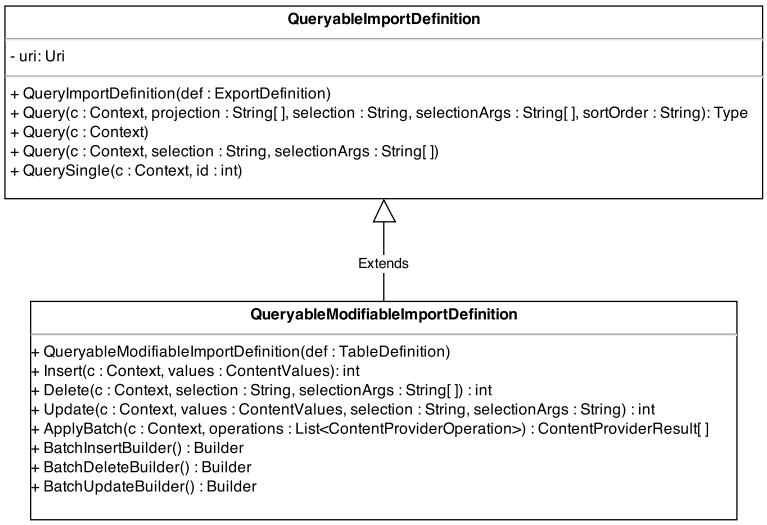


Figure 6.1: UML class diagram of the classes used to import tables.

Usage of the new design in shown in *Code 6.2* which is the cleaned up version of the

same code shown in *Code 6.1*. It should be noted how many of the requried nulls and null checking has been moved into the common function `cursorToTagList` which is used by multiple of the query functions for the tag table.

Code 6.1: Previous function to query for pictograms with a tag.

```
1  public List<Tag> getTagsByCaption(String name) {
2      List<Tag> tags = new ArrayList<Tag>();
3      if (name == null) {
4          return tags;
5      }
6      Cursor c = _context.getContentResolver().query(TagMetaData.CONTENT_URI, columns,
           TagMetaData.Table.COLUMN_NAME + " LIKE '%" + name + "%'", null, null);
7      if (c != null) {
8          tags = cursorToTagList(c);
9          c.close();
10     }
11     return tags;
12 }
```

Code 6.2: Cleaned up version of getTagbyCaption.

```
1  static final QueryableModifiableImportDefinition table =
2      new QueryableModifiableImportDefinition(TAG_TABLE);
3
4  public List<Tag> getTagsByCaption(String name) {
5      if (name == null) {
6          return null;
7      }
8
9      name = DatabaseUtils.sqlEscapeString(name);
10
11     Cursor c = table.Query(_context, "%s like %s", COLUMN_NAME, name);
12
13     return cursorToTagList(c);
14 }
```

All the other controllers received the same treatment which also made the API more consistent in regards to when null is returned and when an empty list is returned.

6.2 Requirement Fulfillment For the Oasis Application

The following sections describe how the requirements for the Oasis application were fulfilled in sprint 4.

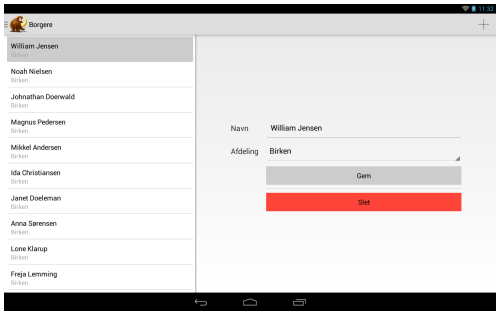
6.2.1 Naming Conventions

Throughout the course of this project the different parties involved have been referred to with different terms. For example, people with autism would be referred to as children, but as the customer has explained, some of them are adults and thusly should not be called children. Instead the following two requirements were established:

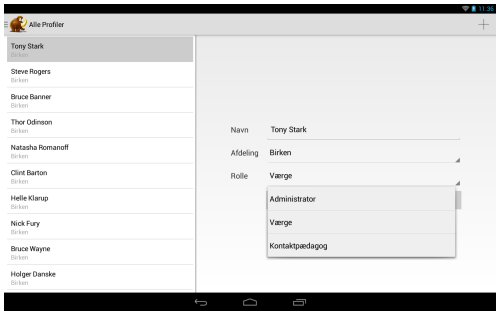
Children should be called "Borger" and parents should be called "Værge".

In the relation between a guardian and a child, the guardian should be called "Kontaktpædagog".

The term "Kontaktpædagog" was already used as seen in figure 4.3a, but the terms "Borger" and "Værge" were not used in the previous sprints, as seen in figures 4.2a, 4.2b, and 5.5a. In sprint four this was changed as seen in figures 6.2a and 6.2b.



(a) Children renamed from "Børn" to "Borger"



(b) Parents renamed from "Forælder" to "Værge", and guardians renamed from "Værge" to "Kontaktpædagog"

Figure 6.2: Examples of strings being renamed to fulfill requirement for naming conventions.

6.2.2 Using GIRAF Components

A requirement states that the GIRAF components should be used throughout the application.

> *GIRAF components should be used to ensure that the application look similar to other applications in the GIRAF ecosystem.*

At the beginning of the project it was decided on a meeting with all groups who are part of the GIRAF project that all applications in the GIRAF ecosystem should use the components. This ensures that all applications look familiar to the users and thus should be easier to use.

Throught sprints two and three the components in GIRAF GUI library changed a lot. In sprint four we began using a newer version of the library than we had previously used. This resulted in the interface having new colors. Not all user interface elements have exist in the GIRAF library, for example the fragments, so we needed to update this to match the new colors manually. Even though interface components that are specific for the Oasis application has been developed, including but not limited to the list selection descrbied in *5.8.1 Show Selected Item In List View p. 44* and the dialog described in *4.6.2 Creation, Modification and Deletion of Profiles For Citizens p. 28*, this was a simple matter of changing the colors in the Android resources as these custom components all take basis in the GIRAF equivalents.

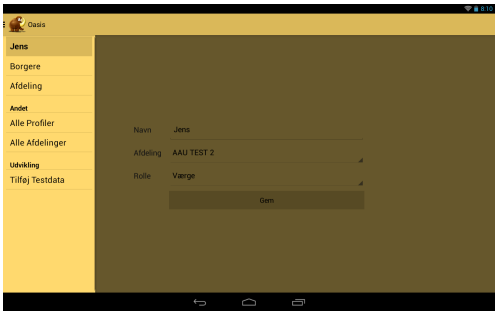
6.2.3 Edit Profile Picture

During the second and the third sprint creation, modification and deletion of citizens and guardians was added to the application but viewing and changing the profile pictures was not added until sprint four.

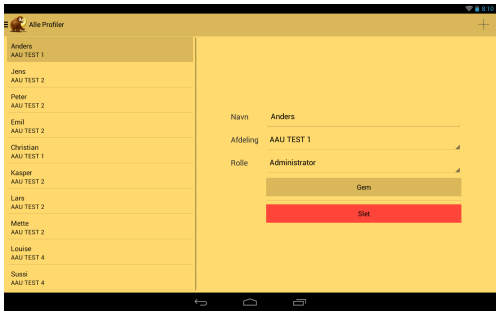
The profile picture provides an easy way for guardians to identify citizens and other guardians. and quickly see which profile is signed into an application.

The profile picture can be edited for both the users own profile (as seen in figure 6.4), for citizens and other profiles.

When the "Vælg piktogram" button is pressed the PictoSearch application wil be opened allowing the user to search for and select a pictogram to be used as the profile picture. PictoSearch is started by calling `startActivityForResult` on the frag-



(a) Menu when using the GIRAF components



(b) Profiles screen when using the GIRAF components

Figure 6.3: Screenshots of the previous Oasis application developed in 2012.

ment with the correct intent. When the user returns after selecting a pictogram, the `onActivityResult` method will be called with the picture as input parameter.

If the "Tag nyt billede" button is pressed the camera application is started and a photo can be taken and used as profile picture. This provides an easy way for the guardians to use a photo of the citizens as profile picture.

The use of nested fragments as described in *4.6.1 Nested Fragments p. 28* was cause to the profile pictures not being saved when the PictoSearch application was started from a child fragment. The reason for that is that `onActivityResult` is only called on activites and is propagated to fragments but not child fragments. To correct this issue, the implementation of `onActvitiyResult` was added to the parent fragments that present a `EditProfileFragment`, used to change the profile picture. Every child fragment of the parent fragment is iterated and the `onActivityResult` is manually called on the child fragment ensuring that it knows when a profile picture has been changed.

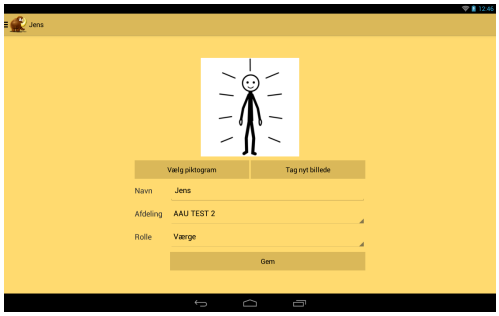


Figure 6.4: Profile picture added to the edit profile screen

7 Common Activity Between Oasis Application and Online Administration Tool

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

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

7.1 Motivation

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

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

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

7.2 Results

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

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

A project to implement an administration tool for the tablets was started in 2012 but the development was not continued in 2013. However, in 2013 the development of the online administration tool begun and there was a plan to port the website to Android using Pro Active Webfilter which, according to the report documenting the development of the administration tool, can be used to run applications developed in PHP on Android[10]. However, the group developing the online administration tool in 2013 was prevented from creating the port to Android and there was therefore not any development on the Android administration tool in 2013.

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

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

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

- Children should be called "Barn/Borger" and parents should be called "Værge/-Forældre".
- In the relation between a guardian and a child, the guardian should be called "Kontaktpædagog".
- No private information on profiles.
- Parents should only be able to access information specified by the guardians.
- The guardians would like to be able to add and remove profiles. It should not necessarily be done by administrators.

These requirements along with the report on the development of the Oasis application in 2012[8] was used to outline the requirements for the Oasis application in development by the group sw614f14.

8 Common Activity During Development Of OasisLib

Many things can go wrong when using OasisLib. To ensure that all of the applications in the multi project worked with the database and OasisLib, help was provided to those who needed it. When someone came to us with a problem, we often joined them at their development environment to give them one on one support. We went through their code together to see where the problem was and sometimes found that the library was used incorrectly and explained how to correctly use the library.

From time to time we also found bugs in OasisLib while helping another group. When this happened we started by creating tests to emulate what caused the problem for the other group. After that we started working on solving the problem until the tests passed. When the problem has been solved, we went back to the group which had the problem to make sure that the solution worked as intended.

During the development we kept track of changes made to the library in a changelog. This changelog was then shared with the other groups in the multi project on the wiki page when the changes were merged into the master branch of OasisLib. This was done as a result of the chaos that ensued after the library was updated to the new database schema right before we took over which happened because of missing documentation and lack of a changelog.

In terms of new features and small non-critical bugs we redirected groups to the issue tracker since they would come by at inconvenient times while we were working on other bugs or features. This was a challenge to deal with since we would often have to prioritize tasks depending on how time critical they were and this could lead to some groups getting their feature request pushed back multiple times which sometimes would make them annoyed at us. So it was difficult to judge which tasks to work on each day when sometimes critical bugs effecting one or two groups would crop up that meant they could not continue development until the bug was fixed.

9 Concluding

9.1 Conclusion

Through the project we have worked on multiple pieces of software. It makes the most sense to conclude on the projects separately and therefore we will conclude on the CAT, the database project and Oasis separetely. Finally we will conclude on the multi project.

9.1.1 CAT and Pecs Importer

The CAT application described in *3.2 Changes to CAT p. 15* was the first project we worked on. As stated in *3.2 Changes to CAT p. 15* the graphical interface was changed to use GIRAF components which helped make sure that the application had a similar look and feel compared to the other applications of the GIRAF project. In addition to this, the categories each have a color associated with it, and this was supposed to be shown in the list of categories but the people who worked on the app before us had not done this and thus it was done by us as described in *3.2.3 Backgrounds of Categories p. 18*. The code was refactored by removing duplicate code as described in *3.2.2 Code Cleanup p. 16*. This reduction in code makes it easier to read and understand the code which will aid in maintanence. There were also strings that were hardcoded as described in *3.2.1 Resource Strings p. 16*, which makes it difficult to change these strings. Instead we used string resource files which means we have all the strings in one place, and if we ever need to translate the application to a different language, it is easier to do using string resource files.

As described in *3.1 Pecs Importer p. 13* we made a small python program that allowed users to import pictograms from their local machine to the central database. The database scheme however, changed in later sprints and we did not maintain Pecs Importer so it has ceased to function. It would not require much effort to change it to be compatible with the new database scheme but we saw no reason to do this.

9.1.2 OasisLib and LocalDB

By working closely together with other groups developing GIRAF we were able to locate and implement missing features. Furthermore with the theory learned in the Database Systems course we were able to remove a lot of redundant code from the library by enabling foreign key constraints and creating triggers that cascades deletion in the appropriate cases. This is in our opinion also a crucial change since it moves the responsibility of maintaining consistency in the data to the DBMS which it is undoubtedly better at than OasisLib. Moving this responsibility also means that LocalDB can be accessed directly more easily in the future if this turns out to be a better approach than OasisLib.

As described we removed code duplication by gathering metadata about tables in a shared Java library which results in increased consistency since tables only have to be

changed in one place instead of two. This transition was however not entirely painless as many groups had trouble with getting the submodule to work which meant we had to provide emergency support to many groups.

The decision to create integration tests was very helpful in making sure the behavior of OasisLib didn't change in unexpected ways that could introduce subtle bugs in the applications depending on OasisLib. The tests also had the added benefit of serving as code examples to groups that would like to know how certain things could be accessed from the API.

We also found that the refactoring significantly improved the quality of the code and addressed many unimplemented and suboptimal implementations of features. LocalDB was as a result of this also made significantly easier to navigate simply due to there being a lot less code to understand.

9.1.3 Oasis Application

Most of the requirements that were laid down by the customers and that were found in the report for the Oasis application from 2012 were successfully fulfilled.

Because only three out of four sprints were used on the Oasis application and both the OasisLib ad the LocalDB project was developed simultaneously with the Oasis application the time that was used on the application was very limited and this has resulted in missing functionality and the application can tehrefore not be considered as finished.

The customers mentioned the requirement that the parents should see a special screen that only presented them with very little information when opening the application. Furthermore it should be possible to administrate the relationships between a guardian and its citizens for all guardians and not only the one currently logged in. Subdepartments are also missing from the Oasis application. However, feature pairity with the previous Oasis application from 2012 was achieved.

The limited time spent on the Oasis application also resulted in very little time being spent on the user interface design of the application. Rather the time was spent on the needed functionality. That said it is important to keep in mind that it was no where mentioned that an interesting interface design wsa one of the customers priorities but it is not unthinkable that the results of usability test will indicate that the design of the application can be improved.

9.1.4 The project as a whole

As described in *chapter 1* the goal of the multiproject was to foster collaboration between groups and emulate a large scale software development project. This resulted in some challenges but also worked out surprisingly well. We were for example worried about how the collaboration between groups would work out and especially that people would be afraid to ask for help with usage of OasisLib. But many groups were excellent at asking for help before trying their hand at workarounds in their own application. In terms of our side of the collaboration we tried our best to not introduce breaking changes and maintain backwards compatibility with the existing API such groups would not constantly be interrupted by their existing functionality breaking and in worst case stop compiling.

The status meetings every week between the groups were in our experience of little to no value since we did not depend on any of the other groups and any other business would usually have been better suited for condensed email summaries. While we can

see the value in these meetings for other groups we are certainly advocates for keeping the meetings short, at most 15 minutes long.

The sprint ends were very long since they presented every application in full each time since different customers attended each time. While the idea behind inviting customers to our sprint ends is solid, the sprint ends became more about the customers than us since many technical details were spared and the meetings dragged out. Our suggestion is to create a separate product presentation for the customers that only involve a few people in presenting all the applications. This would mean that customers would not be invited for sprint ends and only the new additions to each product would need to be presented.

An interesting challenge in the multiproject was providing the 'on site' troubleshooting of other group's applications, since it required us to quickly gather a very localized code overview without understanding what a lot of the other code in the project does.

Over all our experience with the mulitproject was very positive since we learned a lot about collaboration and how to conduct in a larger organization. We feel like we had a major positive impact on the project as a whole since we touched all three tiers of the architecture. This more than made up for the fact that we were to some degree forced to work on the limited set of applications in the GIRAF environment unlike previous semesters where we could work with almost anything.

9.2 Future Work

The following sections describe the work that is suggested to be done if these projects are to be continued by another group next year when the multi project is active again.

9.2.1 OasisLib

This section describes what could be worked on in OasisLib.

- The user should not have access to the database id of an object of the database. It should be something only the library has access to and manages. This results in the library not being as close to the database and more object oriented.
- Instead of using the model ids when inserting relations, as listed in table 5.1, the user should do it by passing the two objects, which needs to be connected.
- Add checks to make sure all the relations make sense. For example if a guardian adds a profile to themselves, they should not be able to add one from another department.
- Make the API user-aware and enforce privacy settings.
- Make the API consistent with what it returns from functions, currently it sometimes returns null for an empty set and other times an empty `ArrayList`.
- More tests should be created, to test all the different methods and their cases.
- Look into creating asynchronous API that helps the visual tier groups create fluid and seamless applications that does not freeze while loading data.
- Create a caching mechanism for images that stops applications from running out of memory, even if they decide to query 10000 pictograms.

- Remove functions that returns all data from a table since this gives performance and privacy concerns. This would also force the older applications to be cleaned up as they were written with the old OasisLib and database design in mind.
- Find a better way to implement the metadata library such other projects do not need to include it in their project root directory and gradle build files.

9.2.2 LocalDB

This section describes what could be worked on in LocalDB.

- Two additional columns have been requested in `profile_pictograms`. The first is a nullable name and the second is a nullable inline text, similar to those in the `pictogram` table. This would allow each profile to customize their pictogram names and inline text.
- In the `sequence` table, one of the groups requested a way to keep track of how far the user has progressed. This could be done by either keeping a number updated or having sequence refer to a frame.
- In the `frame` table, keep track of whether a frame has been completed or rejected. An example would be if the user should ignore one of the steps in a sequence, they would reject it.
- Tests should be created to make sure all functionality of the database works as intended.
- Look into storing image and sound data on sd card instead of inside the database.
- Simplify the multitable export design by using SQL views.
- Get a better requirement specification from the customer regarding exactly how much data needs to be accessible offline since there are a lot of privacy concerns with a 1-1 clone of the central database. It might be enough to simply cache pictograms already used by the citizen.

9.2.3 Oasis Application

For the Oasis application the following functionality and exercises were not done due to the limited time of the project. Instead the focus was on getting the primary functionality done. This includes management of profiles and departments. Two of the following points are requirements from the customers and therefore should have high priority.

- No usability test of the Oasis application was conducted during this semester as it was not until in the middle of the fourth sprint that there was enough functionality in the Oasis application and enough bugs had been fixed to conduct a usability test. There were plans to do an Instant Data Analysis immediately after the last sprint review where the customers were also present. However, the meeting dragged on and no test was conducted. The application has been deployed to the customers tablets and it is expected that they will use it throughout the following year. They will gather experience with the software and should this project continue next year, this experience should be utilized to improve the usability of the product.

- In the database there is a relationship between profiles and applications. Each relationship indicates that the profile has access to the application. The application should allow guardians to manage this relationship and thus restrict which applications a profile has access to. The relationship can currently be managed through the online administration tool and inspiration on how the relationships should be managed can be drawn from the online tool.
- Subdepartments provide a way of organizing departments. When a department has a main department it allows for displaying the apartments in a clear way. Subdepartments were not implemented in the application this year but the database can handle this so implementing it should be a simple task. Inspiration can be drawn from the online administration tool which supports subdepartments. Subdepartments has very little practical meaning as no properties will be shared between the subdepartment and its main department. Therefore the subdepartments were given a low priority.
- In the interview conducted with Birken they mention that when a parent logs into the administration tools, they should have access to very little information and functionality. Birken says that experience tells them that the parents with an autistic child already receive a lot of information about having an autistic child and thus they want to keep the information they give to parents to a minimum. There were no information on exactly what information should be visible to the parents so this should be researched if the project is to be continued.

Bibliography

[1] Android. Android studio. http://developer.android.com/sdk/installing/studio.html, May 2014.

[2] Android. Androidtestcase. http://developer.android.com/reference/android/test/AndroidTestCase.html, May 2014.

[3] Android. Localizing with resources. http://developer.android.com/guide/topics/resources/localization.html, March 2014.

[4] Android. Providing resources. http://developer.android.com/guide/topics/resources/providing-resources.html, March 2014.

[5] Android. Testing fundamentals. http://developer.android.com/tools/testing/testing_android.html, May 2014.

[6] Martin Fowler. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 1999.

[7] Google. Implementing effective navigation. http://developer.android.com/training/implementing-navigation/index.html.

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

[9] Kommunikationscentret Hillerød. Picto-selector with symbols for communication using images. http://www.kc-hil.dk/produkter/picto-selector-med-symboler-til-billedstottet-kommunikation.html.

[10] Lars Chr. Pedersen & Tommy Knudsen Jens M. Lauridsen, Johan Sørensen. Giraf - admin. http://projekter.aau.dk/projekter/da/studentthesis/giraf--admin(d4567010-eab8-4bfd-9a06-43dde4cef2ad).html, May 2014.

[11] Alex Lockwood. Content providers & content resolvers. http://www.androiddesignpatterns.com/2012/06/content-resolvers-and-content-providers.html, May 2014.

[12] N/A. Meeting summerary of sprint end 3. http://cs-cust06-int.cs.aau.dk/attachments/download/298/Sprintend-3-meeting.pdf, May 2014.

[13] Oracle. Mysql connectors. http://www.mysql.com/products/connector/, May 2014.

[14] Defuse Security. Salted password hashing - doing it right. https://crackstation.net/hashing-security.htm, May 2014.

[15] A. Silberschatz, H. Korth, and S. Sudarshan. *Database System Concepts*. Connect, learn, succeed. McGraw-Hill Education, 2010.

[16] SQLite. Frequently asked questions. http://www.sqlite.org/faq.html#q19, May 2014.

[17] SQLite. Sqlite foreign key support. http://www.sqlite.org/foreignkeys.html, May 2014.

[18] Martijn van der Kooij. Pictoselector. http://www.pictoselector.eu.

[19] Martijn van der Kooij. Pictoselector, free symbol resources. http://www.pictoselector.eu/index.php?option=com_weblinks&view=category&id=42&Itemid=244&lang=en.

[20] Martijn van der Kooij. Pictoselector, optional downloads. http://www.pictoselector.eu/index.php?option=com_content&view=article&id=88&Itemid=252&lang=en.

A List Of Inherited GIRAF Applications, Libraries and Services

Name	Type	Description
Tortoise	User application	Create stories based on pictograms
Timer	User application	Manage time and display countdowns in other applications
Sequence	User application	Create and show how to do tasks based on a sequence of pictograms
Croc	User application	Draw new pictograms
Pictosearch	User application	Seach for pictograms and categories
Parrot	User application	Get pictograms and sequences read out aloud
CAT	User application	Create categories for pictograms
Launcher	User application	Start other GIRAF applications from here
Oasis	User application	Offline profile managemenet
Giraf Admin	Website	Online management of profiles, departments and pictograms
Train	Game	Learn how to categorize differnt pictograms
Cars	Game	Learn how to control speech volume
GUI	Library	Common GUI components used by user applications
OasisLib	Library	Communication layer between user applicaitons and database
LocalDB	Service	Locally running SQLite database
RemoteDB	Service	External MySQL database used for synchronizing between tablets

Figure A.1: List of applications, libraries and services in the GIRAF environment.

B Customer Contact Questions

The following documents the questions used to talk to the customers.

B.1 Preliminary

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

B.2 Pictograms

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

B.3 Profiles, users and relations

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

B.4 Applications

- Assign applications to citizens.

B.5 Parent access

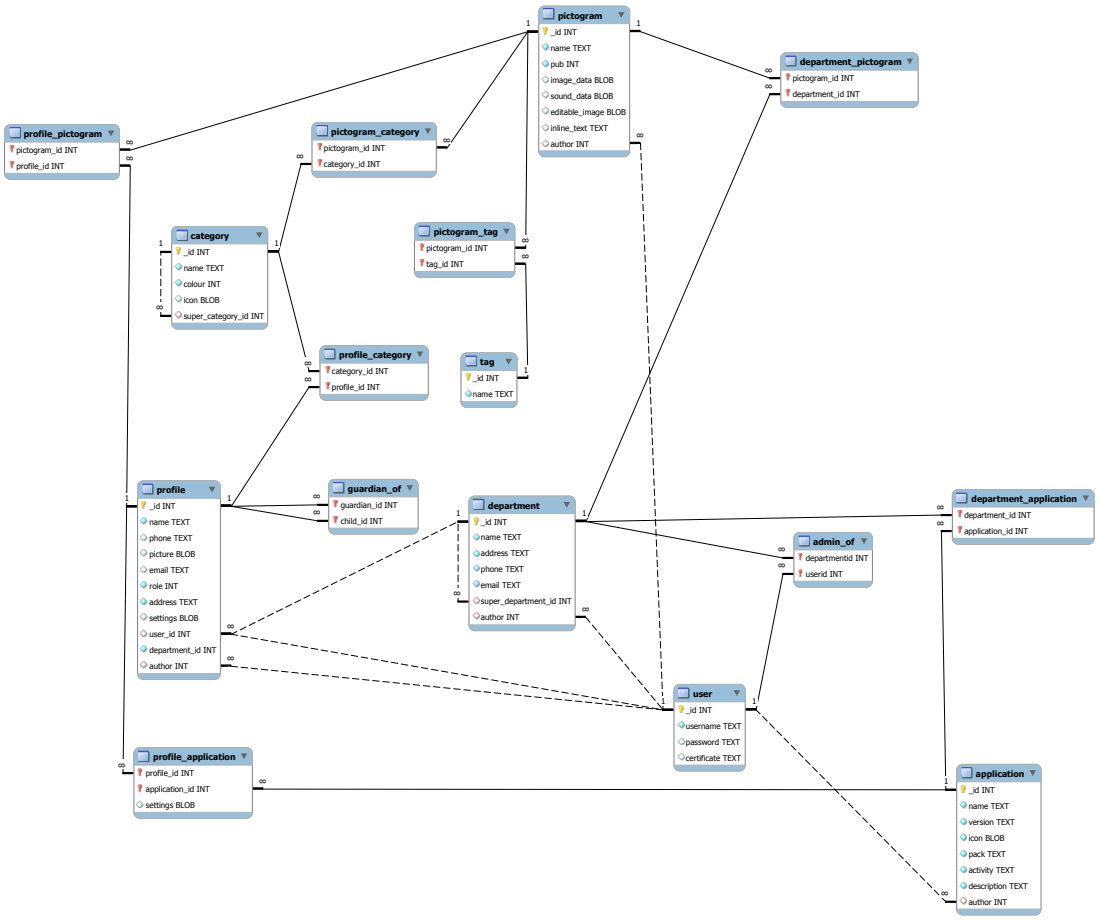
- Should the parents be able to log in to the homepage?
- What should they be able to do?

B.6 Show the site to the customer and talk loosely about it

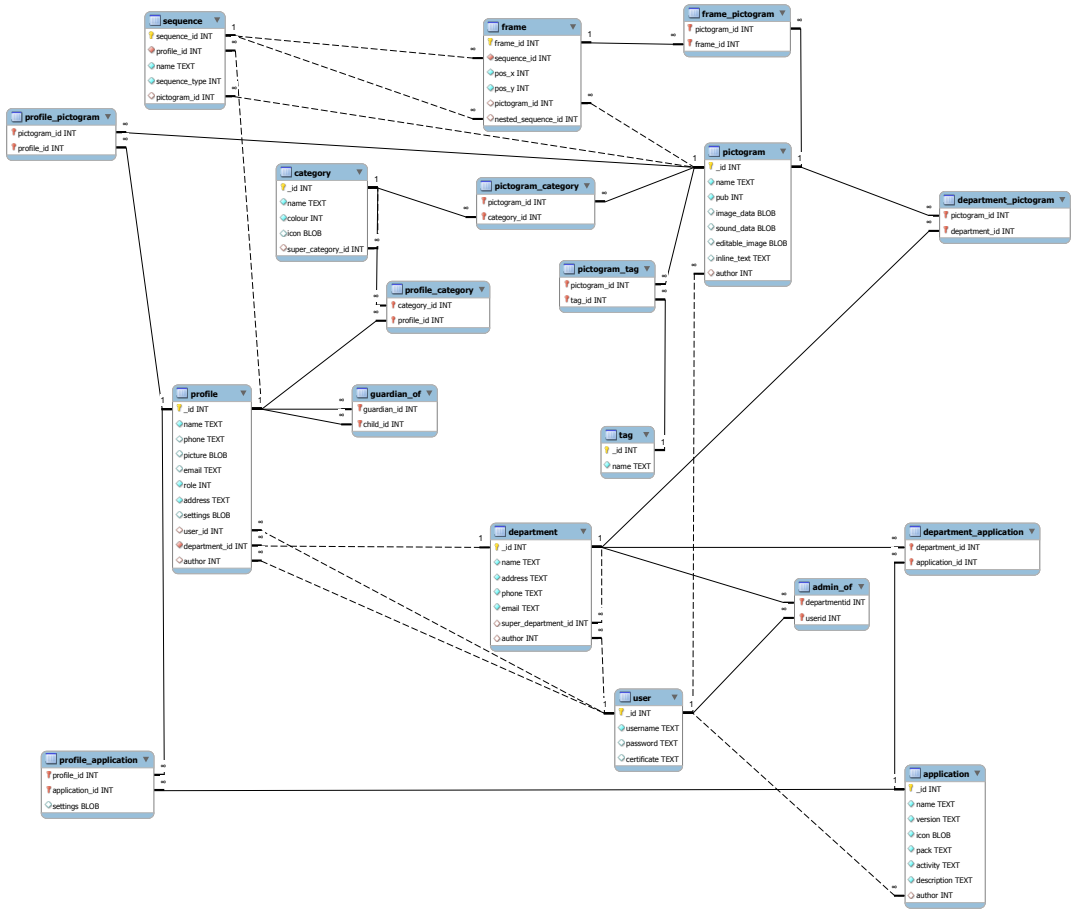
- Would drag and drop make it easier?

C Sprint 2 EER For LocalDB

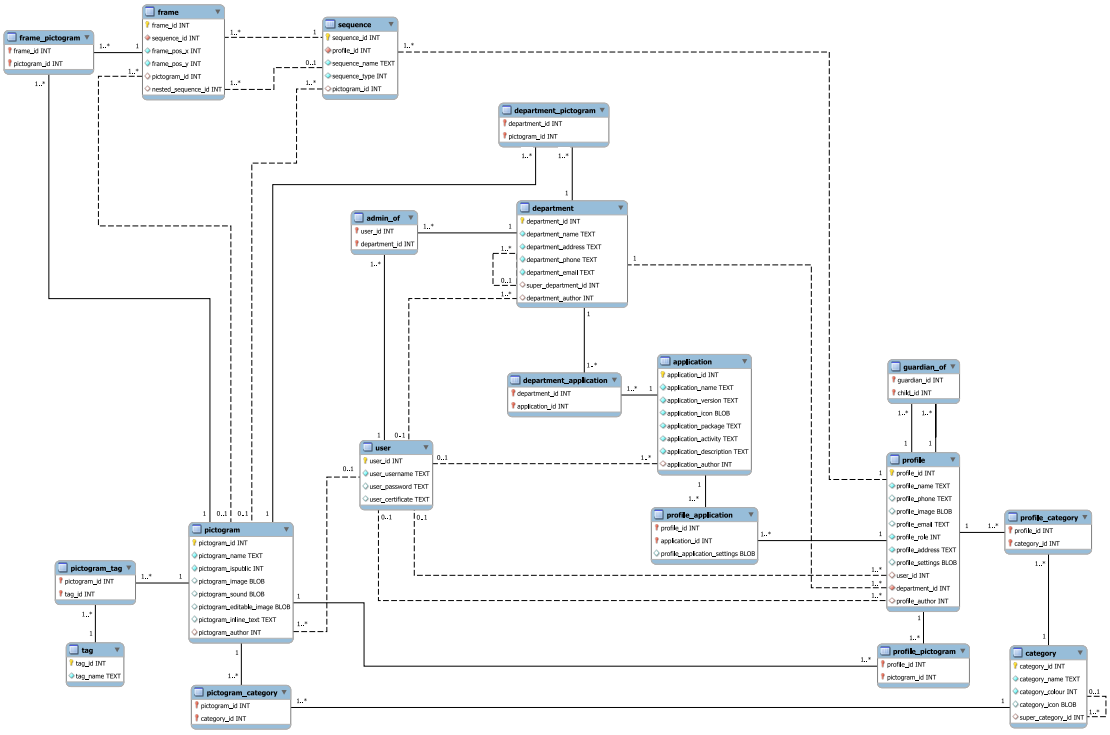
Legend
Primary key
Primary and foreign key
Column definition
Nullable column definition
Foreign key
Nullable foreign key



D Sprint 3 EER for LocalDB



E Sprint 4 EER for LocalDB



F Old LocalDB Code Examples

Code F.1: Snippet of query method in DbProvider.

```
@Override
public Cursor query(Uri uri, ...) {
switch (sUriMatcher.match(uri)) {
    .
    .
    .
    case TAGS_TYPE_LIST:
        builder.setTables(TagMetaData.Table.TABLE_NAME);
        builder.setProjectionMap(tagProjectionMap);
        break;
    case TAGS_TYPE_ONE:
        builder.setTables(TagMetaData.Table.TABLE_NAME);
        builder.setProjectionMap(tagProjectionMap);
        builder.appendWhere(TagMetaData.Table.COLUMN_ID + " = " + uri.getPathSegments().get
    (1));
        break;

    case USERS_TYPE_LIST:
        builder.setTables(UserMetaData.Table.TABLE_NAME);
        builder.setProjectionMap(userProjectionMap);
        break;
    case USERS_TYPE_ONE:
        builder.setTables(UserMetaData.Table.TABLE_NAME);
        builder.setProjectionMap(userProjectionMap);
        builder.appendWhere(UserMetaData.Table.COLUMN_ID + " = " + uri.getPathSegments().
    get(1));
        break;
    .
    .
    .
    }
}
```

Code F.2: Excerpt of the update method in DbProvider.

```
@Override
public int update(Uri uri, ContentValues values, String where, String[] whereArgs) {
SQLiteDatabase db = dbHelper.getWritableDatabase();

switch(sUriMatcher.match(uri)) {
    .
    .
    .
    case DEPARTMENTSAPPLICATIONS_TYPE_LIST:
        rowsUpdated = db.update(DepartmentApplicationMetaData.Table.TABLE_NAME, values,
    where, whereArgs);
        break;
    case DEPARTMENTSAPPLICATIONS_TYPE_ONE:
        throw new UnsupportedOperationException("Not yet implemented");
    case DEPARTMENTSPICTOGRAMS_TYPE_LIST:
        rowsUpdated = db.update(DepartmentPictogramMetaData.Table.TABLE_NAME, values, where
    , whereArgs);
        break;
    case DEPARTMENTPICTOGRAMS_TYPE_ONE:
        throw new UnsupportedOperationException("Not yet implemented");
    case GUARDIANSOF_TYPE_LIST:
        rowsUpdated = db.update(GuardianOfMetaData.Table.TABLE_NAME, values, where,
    whereArgs);
        break;
    case GUARDIANSOF_TYPE_ONE:
        throw new UnsupportedOperationException("Not yet implemented");
    case PICTOGRAMSCATEGORIES_TYPE_LIST:
        rowsUpdated = db.update(PictogramCategoryMetaData.Table.TABLE_NAME, values, where,
    whereArgs);
        break;
    case PICTOGRAMSCATEGORIES_TYPE_ONE:
        throw new UnsupportedOperationException("Not yet implemented");
    case PICTOGRAMS_TYPE_LIST:
        rowsUpdated = db.update(PictogramMetaData.Table.TABLE_NAME, values, where,
    whereArgs);
        break;
    .
    .
    .
}

getContext().getContentResolver().notifyChange(uri, null);
return rowsUpdated;
}
```

G Metadata Code Examples

Code G.1: Metadata describing the Category table with new design.

```
1  public class CategoryTable {
2
3      public static final String TABLE_NAME = "category";
4      public static final String COLUMN_ID = "category_id";
5      public static final String COLUMN_NAME = "category_name";
6      public static final String COLUMN_COLOUR = "category_colour";
7      public static final String COLUMN_ICON = "category_icon";
8      public static final String COLUMN_IDSUPERCATEGORY = "super_category_id";
9
10     public static final ForeignKeyDefinition ID_SUPER_CATEGORY_FOREIGN_KEY =
11             new ForeignKeyDefinition(
12                     COLUMN_IDSUPERCATEGORY,
13                     TABLE_NAME,
14                     CategoryTable.COLUMN_ID,
15                     OnDeleteCascade
16             );
17
18     public static final TableDefinition CATEGORY_TABLE = new TableDefinition(
19             TABLE_NAME,
20             new PrimaryKeyDefinition(COLUMN_ID),
21             new ForeignKeyDefinition[]{
22                     ID_SUPER_CATEGORY_FOREIGN_KEY
23             },
24             new ColumnDefinition(COLUMN_NAME, Text, NotNull),
25             new ColumnDefinition(COLUMN_COLOUR, Integer, NotNull),
26             new ColumnDefinition(COLUMN_ICON, Blob),
27             new ColumnDefinition(COLUMN_IDSUPERCATEGORY, Integer)
28     );
29 }
```

Code G.2: Old metadata describing the Category table.

```
1  public class CategoryMetaData {
2      public static final Uri CONTENT_URI = Uri.parse("content://dk.aau.cs.giraf.oasis.
       localdb.AutismProvider/category");
3
4      public static final String CONTENT_TYPE_CATEGORIES_LIST = "vnd.android.cursor.dir/vnd.
       dk.categories";
5      public static final String CONTENT_TYPE_CATEGORY_ONE = "vnd.android.cursor.item/vnd.dk.
       categories";
6
7      public class Table implements BaseColumns {
8          public static final String TABLE_NAME = "category";
9
10         public static final String COLUMN_ID = "_id";
11         public static final String COLUMN_NAME = "name";
12         public static final String COLUMN_COLOUR = "colour";
13         public static final String COLUMN_ICON = "icon";
14         public static final String COLUMN_IDSUPERCATEGORY = "super_category_id";
15     }
16 }
17
18 public class CategoryTable {
19
20     private static final String TABLE_CREATE = "CREATE TABLE "
21             + CategoryMetaData.Table.TABLE_NAME
22             + "("
23             + CategoryMetaData.Table.COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, "
24             + CategoryMetaData.Table.COLUMN_NAME + " TEXT NOT NULL, "
25             + CategoryMetaData.Table.COLUMN_COLOUR + " INTEGER NOT NULL, "
26             + CategoryMetaData.Table.COLUMN_ICON + " BLOB, "
27             + CategoryMetaData.Table.COLUMN_IDSUPERCATEGORY + " INTEGER, "
28             + "FOREIGN KEY(" + CategoryMetaData.Table.COLUMN_IDSUPERCATEGORY +
29                 ") REFERENCES "+ CategoryMetaData.Table.TABLE_NAME +
30                     "("+ CategoryMetaData.Table.COLUMN_ID +") ON DELETE CASCADE"
31             + ");";
32
33     private static final String TABLE_DROP= "DROP TABLE IF EXISTS " + CategoryMetaData.
       Table.TABLE_NAME + ";";
34
35     /**
36      * Executes a sql string for creating a new departments table
37      * @param db this is an instance of a sqlite database
38      */
39     public static void onCreate(SQLiteDatabase db) {
40         db.execSQL(TABLE_CREATE);
41     }
42
43     /**
44      * Executes a sql string which drops the current table and then the methods calls
       oncreate, which creates a new table
45      * @param db this is an instance of a sqlite database
46      * @param oldVersion integer referring to the old version number
47      * @param newVersion integer referring to the new version number
48      */
49     public static void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
50         db.execSQL(TABLE_DROP);
51         onCreate(db);
52     }
53 }
```

Code G.3: Example of a multitable export.

```
public class ProfileApplicationTable {

    public static final String TABLE_NAME = "profile_application";
    public static final String COLUMN_IDPROFILE = "profile_id";
    public static final String COLUMN_IDAPPLICATION = "application_id";
    public static final String COLUMN_SETTINGS = "profile_application_settings";

    public static final ForeignKeyDefinition ID_PROFILE_FOREIGN_KEY =
            new ForeignKeyDefinition(
                    COLUMN_IDPROFILE,
                    ProfileTable.TABLE_NAME,
                    ProfileTable.COLUMN_ID,
                    OnDeleteCascade
            );
    public static final ForeignKeyDefinition ID_APPLICATION_FOREIGN_KEY =
            new ForeignKeyDefinition(
                    COLUMN_IDAPPLICATION,
                    ApplicationTable.TABLE_NAME,
                    ApplicationTable.COLUMN_ID,
                    OnDeleteCascade
            );

    public static final TableDefinition PROFILE_APPLICATION_TABLE = new TableDefinition(
            TABLE_NAME,
            new PrimaryKeyDefinition(
                    new ColumnDefinition(COLUMN_IDPROFILE, Integer, NotNull),
                    new ColumnDefinition(COLUMN_IDAPPLICATION, Integer, NotNull)
            ),
            new ForeignKeyDefinition[]{
                    ID_PROFILE_FOREIGN_KEY,
                    ID_APPLICATION_FOREIGN_KEY
            },
            new ColumnDefinition(COLUMN_SETTINGS, Blob)
    );

    public static final MultiTableExport PROFILE_APPLICATION =
            new MultiTableExport(
                    "profile_with_application",
                    ID_PROFILE_FOREIGN_KEY,
                    ID_APPLICATION_FOREIGN_KEY
            );
}
```

H License For Password Hashing Algorithm

During the maintenance of OasisLib, we noticed that it saves passwords in the `user` table as plain text. To keep the user passwords safe, we implemented a salting and hashing class created by Taylor Hornby[14]. The use of the class requires the copyright notice as shown in *Code H.1*.

Code H.1: License for the password hashing algorithm created by Taylor Hornby

```
/*
 * Password Hashing With PBKDF2 (http://crackstation.net/hashing-security.htm).
 * Copyright (c) 2013, Taylor Hornby
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 * this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 * this list of conditions and the following disclaimer in the documentation
 * and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */
```