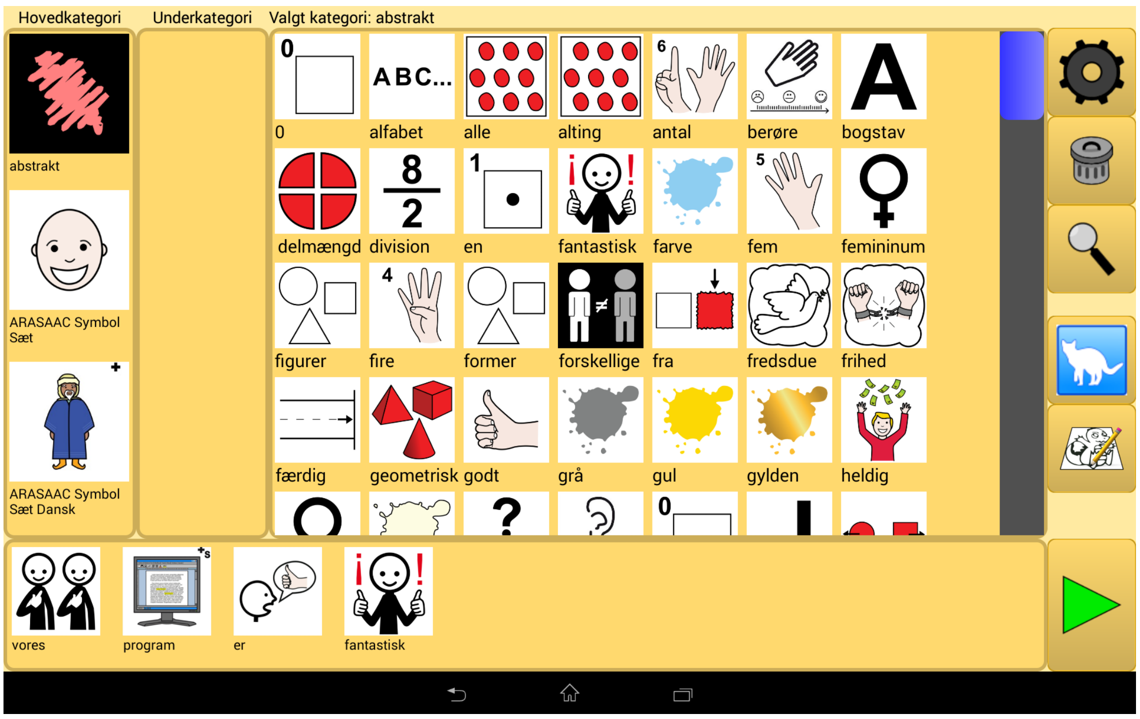

Multi-Project: Oasis Library & Parrot

P6 Project

Group SW615F14

Software

Department of Computer Science

Aalborg University

Spring 2014

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Multi-Project: Oasis Library & Parrot

Project period:
P6, spring 2014

Project group:
SW615F14

Group members:
Alexander Drægert
Rasmus Fischer Gadensgaard
Anders Christian Kaysen
Christian Jødal O'Keeffe

Supervisor:
Ulrik Nyman

Total number of pages:
79

Department of Computer Science
Selma Lagerlöfs Vej 300
DK-9220 Aalborg East
http://www.cs.aau.dk/en

Synopsis:

This report contains chapters about the progress made in each sprint in the development of the *Oasis Library* and the application *Piktooplæser*. Further it contains two collaborative chapters written in collaboration with other groups. The main subject of this semester project is complex software systems, and in this case it is a system to help autistic citizens in their everyday life.
The first sprint was used to refactor and update the *Oasis Library* to match the new database schema. The last three sprints was used to improve the application *Piktooplæser* formerly known as *Parrot*.
The report also contains sections about why we chose Scrum as management framework and what responsibilities that imposed on the group. Further it contains our recommendations for the next students to work with this GIRAF multi-project.

Preface

This is a 6th semester multi-project in Software Engineering at Aalborg University. This report documents the ongoing progress of the project throughout the semester.

References to sources are done using the name of the author in brackets, e.g. [Rubin, 2012]. More information about the source can be found on page 65.

The CD as seen in Appendix B of this report contains the source code of the application, the compiled application (.apk file), and a digital version of the report.

Aalborg, May 26, 2014

Alexander Drægert

Rasmus Fischer Gadensgaard

Anders Christian Kaysen

Christian Jødal O'Keeffe

Summary

This project is a multi-project consisting of 16 groups and a total of 60 members. We decided to use Scrum as a management framework and some of our group members were appointed special responsibilities including Scrum-of-Scrums Master. Before delegating projects to each group, we migrated the entire multi-project from Eclipse to Android Studio.

We then chose to work on the *Oasis Library* project, which is a library to communicate between the local database and the applications in this multi-project. After updating the library to communicate with the updated database layout, we switched to the project *Parrot*.

Parrot, now named *Piktooplæser*, is an application for pronouncing pictograms to create sentences. We started by changing the data storage from XML-files to the local database. Then we hosted an interview with a couple of customer to get a better understanding of their needs and requirements. From these requirements we changed the GUI and added small changes that would ease the workflow throughout the process of adding, categorizing, and pronouncing a pictogram.

In collaboration with another group, we then wrote the sound player and Text-To-Speech methods in the Pictogram Library.

After receiving feedback from our costumers at a demonstration event, we began implementing the newly requested features. We made a mode where the user should select the possible pictograms where after the citizen could only choose from these.

At the end of the multi-project the database was filled with approximately 14,000 pictograms. We spent the remaining time fixing bugs and crashes, so the application in the end would be stable no matter the amount of pictograms.

Contents

Introduction 1

This project is a multi-project called GIRAF which consists of many applications and libraries. The project is aimed at autistic citizens with the purpose of easing their everyday life. Given that some autistic citizens have a limited language and can only communicate with the help of pictograms, the main focus of our part of the multi-project is communication. A pictogram is a picture of an object, an action, a feeling etc. Autistic citizens can then form sentences by arranging multiple pictograms next to each other.

The Android applications and libraries in this multi-project are:

- *LocalDB*
- *Oasis Library*
- *Oasis App*
- *Kategoriværktøjet*
- *Giraf Components*
- *GIRAF Launcher*
- *Piktooplæser*
- *Piktotegner*
- *Pictosearch*
- *Ugeskema*
- *Sekvens*
- *Livshistorier*
- *Stemmespillet*
- *Tidstager*
- *Kategorispillet*

The tools developed for web use in this multi-project are:

- *Ugeskema*
- *GIRAF Admin*

The iOS applications in this multi-project are:

- *Tal For Mig*
- *Visual Scheduler*

The GIRAF project began in the spring of 2011, and has been worked on by the 6^{th} semester students every year since. Until this year, every application in the GIRAF project was made for Android, but this year two groups decided to work on applications for the iOS platform. Futhermore there were developed tools for the web also.

Multi Project - Preparation 2

This chapter describes how we created a working environment to manage the multi-project and the tasks we performed before releasing the source code to the other groups.

2.1 Scrum as Management Framework

The objective of this project is to make a series of software solutions which can help autistic people communicate and more. To achieve this objective we have 60 people divided into 16 groups and roughly four months.

In order to manage this many people and avoid chaos we needed a management framework. We looked at a series of frameworks but decided on using Scrum as described in "Essential Scrum" by [Rubin, 2012].

This decision was made because:

- Scrum allows for each individual group to write code as they prefer.
- We have actual product owners.
- Scrum assume the needs of the product owner changes or develops throughout the development phase.
- Product owners will see a demonstration and give feedback on our progress at each sprint review.
- The number of groups is acceptable with a Scrum-of-Scrums structure.
- Scrum-of-Scrums meetings are good for knowledge sharing.

2.1.1 Responsibility of the Scrum-of-Scrums Master

One of the responsibilities of a Scrum Master and by extension a Scrum-of-Scrums Master is to shield the development team from outside interferences. In this particular project that means to:

- Write and distribute agendas for each status meeting.
- Manage the status meetings and sprint planning.
- Interact with the semester coordinator.

- Ensure the needed equipment and facilities are provided.
- Execute the sprint planning meeting.

An additional responsibility of this Scrum-of-Scrums Master is to serve the groups when needed. This means he should solve inconvenient problems when they emerge or is made aware of by members of the groups. Examples of problems that occurred in this project could be to:

- Update the projects website.
- Create and use a Google Play account.
- Investigate how to sign Android applications to a Google Play account.
- Extend dedicated server space.
- Figure out how to project the screen of a tablet onto a projector.
- Book rooms for status meetings, sprint reviews, and sprint planning at times all group members can participate.

The Scrum-of-Scrums Master role was unanimously appointed to a member of our group. Given the size of the responsibilities entrusted to the Scrum-of-Scrums Master, some of these were delegated to other group members.

2.1.2 Scrum-of-Scrums Meetings

The Scrum-of-Scrums meetings were an extended version of the daily Scrum meetings. At these meetings the Scrum Master from each group gathered and presented their progress. First each Scrum Master presented the progress since last meetings, then shared whether they had encountered any problems or was waiting on the work of another group, and lastly what they expected to complete till next meeting. This would ensure all groups had knowledge of the overall status of the project and whether or not somebody was waiting for them to finish what they were working on.

These Scrum-of-Scrums meetings were also used as a sort of boardroom were the big decisions were discussed and decided. By doing this at these meetings we ensured everybody had a say in the particular matter and everybody knew the outcome of the decision. When deciding less important matters a committee was selected from the Scrum Masters of the affected groups.

2.1.3 Sprint Reviews

At the end of each sprint we hosted a sprint review event where the product owners, supervisors, and all developers participated. Before presenting at such an event, each group needed confirmation that the work they wanted to present is done and ready for presentation. This responsibility was given to one person who should keep track of the backlogs for each specific group and the entire project. He was given the title Backlog Manager.

Given the limited number of opportunities to get feedback on our plans we added a presentation about our plans for the future in the end of each sprint review. It was here we discussed ideas and the direction we should head. This presentation was managed and moderated by the Backlog Manager. To do this he should be aware of where the entire project as well as the individual subprojects are headed.

2.2 Project Preparations

The following sections describes the process of exporting the Eclipse projects from last year, and import it in Android Studio.

2.2.1 Eclipse to Android Studio

This section describes the work of migrating the code from Eclipse [The Eclipse Foundation, 2014] to Android Studio [Google, 2014]. This caused a great deal of trouble, at a time where the other groups depended on our group to complete the migration. In the beginning of the multi-project, we decided to work in Android Studio. Because the previous subprojects of the GIRAF project was written in Eclipse , which used a different build tool called ANT, we needed to convert the ANT build files to Gradle build files, which are used by Android Studio.

Our group exported the old project via Eclipse to an Android Studio project. The Eclipse export function had some issues and generated some errors when exporting the old project. We corrected the errors by:

1. Updating the projects build.gradle file to use the newest version of Gradle at the time as seen in Listing 2.1.

```
buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        //Updated from version 0.5.+ to 0.7.+
        classpath 'com.android.tools.build:gradle:0.7.+'
    }
}
```

Listing 2.1: Build.gradle file after error correction.

2. Updating the compileSdkVersion and buildToolsVersion of all AndroidManifest.xml and build.gradle files as seen in Listing 2.2 and Listing 2.3 to fix “manifest merging failed” error.

```
    <uses-sdk
        android:minSdkVersion="15"
        android:targetSdkVersion="19"/>
```

Listing 2.2: Uses-sdk in AndroidManifest.xml after error correction.

```
    compileSdkVersion 15
    buildToolsVersion "19.0.1"
```

Listing 2.3: SDK and build tools version in build.gradle after error correction.

3. Adding the code shown in Listing 2.4 to the AndroidManifest.xml files missing the code shown in Listing 2.4 to correct the "Missing 'Application' Error".

```
<application
</application>
```

Listing 2.4: Empty <application> tag.

4. Comparing and mirror contents of the <activity> tag for dependent subprojects to fix the remainder of "manifest merging failed" errors. An example of an <activity> tag is shown in Listing 2.5.

```
<activity
            android:name="dk.aau.cs.giraf.wombat.drawlib.
                DoneScreenActivity"
            android:screenOrientation="landscape"
            android:theme="@android:style/Theme.Holo.NoActionBar.
                Fullscreen">
</activity>
```

Listing 2.5: An example of an <activity> tag.

2.2.2 Android Studio to Git

At this point in time we had the project and all subprojects up-and-running in Android Studio. We then had to push the workspace to Git [Git] so the other groups could work on their respective assignments.

Because every group had different assignments and therefore different working areas, we needed to create a Git structure which would allow to pull only the needed subproject, that in turn should be buildable. For a subproject to be buildable, all its dependencies of other subprojects should be included in the original subproject. This should be done recursively so that all dependent subprojects in a given subproject are buildable.

For this structure to function correctly the newest revision of a subproject must be the revision which is used when other subprojects depend on it. To ensure this it is not possible to store a local version of all dependent subprojects in each subproject, as this would result in multiple different versions of the same subproject being used.

We therefore created all subprojects as modules in Git with submodules corresponding to their dependencies no matter at which recursion level it appears. The choice of using submodules is made because all dependencies, no matter at which recursion level it appears, must be stored in the root of the subproject.

In order to correctly create the complete Git structure, we made an overview of all subprojects dependencies which is seen in Figure 2.1.

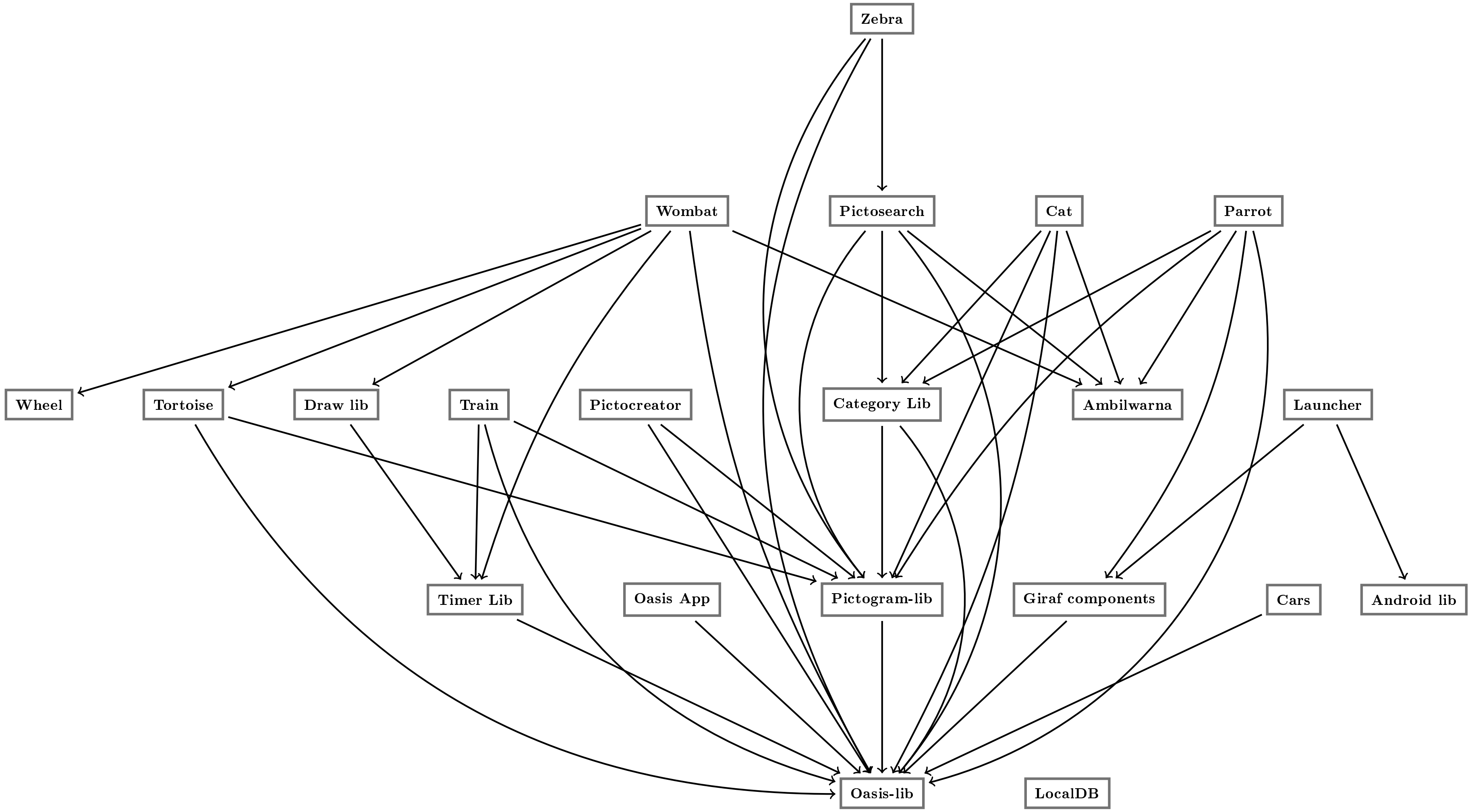

Figure 2.1: Overview of project dependencies as they were at the start of sprint 1.

2.2.3 Android Studio & Gradle Complications

As previously mentioned, we use the Android Studio IDE which is currently not released in a final edition. This proved to pose a series of problems with version consistency and Gradle versioning. When updating Android Studio to a version higher than 0.4.6 it no longer supports Gradle 1.09 and lower. This posed a problem because our entire project is using Gradle 1.09. We found that when individual members of the project updated their Android Studio and found the project was no longer buildable, they changed the code and pushed it, which would pose problems for other members. To remedy this problem, we agreed not to update Android Studio. We chose not to update it, because Android Studio was in development and new updates were available every couple of days. This would force us to update Gradle versions for all subprojects every couple of days, which we decided was too much work compared to the outcome. Futher, Jenkins was only set up to work with Gradle 1.09.

3 1st Sprint - Oasis and Databases

This chapter describes the first sprint in the multi project. At the beginning of this first sprint, the main task was identifying the objectives and shortcomings of the *Oasis Library* project. This involved reading the report from last year, and reading the source code. It was determined that the main problem with *Oasis Library* was that it did not work with the current database schema. As soon as these problems were identified, the work began on adapting the library to the new database schema. The structure of the database system is explained in Chapter 4.

3.1 Refactoring “Department” in OasisLib

We have refactored all classes from the existing *Oasis Library*. The `department` class, from the database schema seen in Appendix A, is here used to illustrate the most important and common changes.

3.1.1 Meta Data

We changed the URI and the relations to match the new database schema as seen in Listing 3.1 and Listing 3.2.

```java
public static final Uri CONTENT_URI = Uri.parse("content://dk.aau.
    cs.giraf.oasis.localdb.AutismProvider/department");
```

Listing 3.1: Content URI for `Department`.

```java
public static final String TABLE_NAME = "department";
public static final String COLUMN_ID = "id";
public static final String COLUMN_NAME = "name";
public static final String COLUMN_ADDRESS = "address";
public static final String COLUMN_PHONE = "phone";
public static final String COLUMN_EMAIL = "email";
public static final String COLUMN_SUPERDEPID = "super_department_id
    ";
public static final String COLUMN_AUTHOR = "author";
```

Listing 3.2: Table layout for `Department`.

3.1.2 Model

In the model of the department we updated the accessors to match the new table layout seen in Listing 3.2. Additionally we extended the constructor to input name, address, phone, email, super department id, and author, as seen in Listing 3.3. Finally we updated the `toString` and `equals` methods to handle the updated table layout. An example of the `equals` can be seen in Listing 3.4.

```java
public Department(String name, String address, long phone, String
    email, int superDepartmentID, int author)
{
  this.name = name;
  this.address = address;
  this.phone = phone;
  this.email = email;
  this.superDepartmentID = superDepartmentID;
  this.author = author;
}
```

Listing 3.3: Constructor for `Department`.

```java
@Override public boolean equals(Object aDepartment)
{
  if ( this == aDepartment ) return true;

  if ( !(aDepartment instanceof Department) ) return false;

  Department department = (Department)aDepartment;

  return
    EqualsUtil.areEqual(this.getId(), department.getId()) &&
    EqualsUtil.areEqual(this.getName(), department.getName()) &&
    EqualsUtil.areEqual(this.getAddress(), department.getAddress())
        &&
    EqualsUtil.areEqual(this.getPhone(), department.getPhone()) &&
    EqualsUtil.areEqual(this.getEmail(), department.getEmail()) &&
    EqualsUtil.areEqual(this.getSuperDepartmentId(), department.
       getSuperDepartmentId()) &&
    EqualsUtil.areEqual(this.getAuthor(), department.getAuthor());
  }
```

Listing 3.4: `Equal` method for `Department`.

3.1.3 Controller

We modified the controller for the `department` class by editing the existing methods to match the new table layout and wrote new methods when needed. An update of an existing method, `removeDepartment`, is partly seen in Listing 3.5. For the new `getDepartmentByName` method and other new methods to work, we use a cursor to get a copy of the desired `Department` object. The cursor can be seen in Listing 3.6. The new method we wrote, `getDepartmentByName`, is seen in Listing 3.7.

```java
public int removeDepartment(Department department)
{

    DepartmentPictogramController departmentPictogramController = new DepartmentPictogramController(_context);
    List<DepartmentPictogram> departmentPictograms = departmentPictogramController.getDepartmentPictograms();

    for (int i = 0; i < departmentPictograms.size(); i++)
    {
        if (departmentPictograms.get(i).getDepartmentID() == department.getId())
        {
            departmentPictogramController.removeDepartmentPictogram(departmentPictograms.get(i));
        }
    }
[...]

[...]
  return _context.getContentResolver().delete(DepartmentMetaData.CONTENT_URI, DepartmentMetaData.Table.COLUMN_ID + " = '" + department.getId() + "'", null);
}
```

Listing 3.5: Part of `removeDepartment`.

```java
private Department cursorToDepartment(Cursor cursor) {
  Department department = new Department();
  department.setId(cursor.getInt(cursor.getColumnIndex(DepartmentMetaData.Table.COLUMN_ID)));
  department.setName(cursor.getString(cursor.getColumnIndex(DepartmentMetaData.Table.COLUMN_NAME)));
  department.setAddress(cursor.getString(cursor.getColumnIndex(DepartmentMetaData.Table.COLUMN_ADDRESS)));
  department.setEmail(cursor.getString(cursor.getColumnIndex(DepartmentMetaData.Table.COLUMN_EMAIL)));
  department.setPhone(cursor.getLong(cursor.getColumnIndex(DepartmentMetaData.Table.COLUMN_PHONE)));
    department.setSuperDepartmentID(cursor.getInt(cursor.getColumnIndex(DepartmentMetaData.Table.COLUMN_SUPERDEPID)));
    department.setAuthor((cursor.getInt((cursor.getColumnIndex((DepartmentMetaData.Table.COLUMN_AUTHOR))))));

  return department;
}
```

Listing 3.6: Cursor for `department`.

```
public List<Department> getDepartmentByName(String name) {
  if (name == null) {
    return null;
  }

    List<Department> departments = new ArrayList<Department>();

    Cursor cursor = _context.getContentResolver().query(
        DepartmentMetaData.CONTENT_URI, columns, DepartmentMetaData.
        Table.COLUMN_NAME + " = '" + name + "'", null, null);

  if (cursor != null) {
    if (cursor.moveToFirst()) {
      while (!cursor.isAfterLast()) {
        departments.add(cursorToDepartment(cursor));
        cursor.moveToNext();
      }
    }
    cursor.close();
  }

  return departments;
}
```

Listing 3.7: `getDepartmentByName` from `DepartmentController`.

3.1.4 Communication With the LocalDB group

Because we worked on the interface for the *LocalDB*, we had to work closely together with the group working on *LocalDB* to ensure that their schema was consistent with our conception of the schema. The communication with the *LocalDB* happened by either one from our group walking across the hall to their group room or one of them comming to our group room. In that way the communication with the *LocalDB* group was very informal, yet effective, since we could help each other fix problems quickly.

3.1.5 Refactoring The Oasis App

The *Oasis App* had to be refactored because the old version did not match the new database schema. In order for the *Oasis App* to be able to display the data from the local database, *Oasis App* had to use some of the modified and new methods from the new *Oasis Library*, which also was updated to the new database schema. Most of the changes needed to make *Oasis App* work, was in the *Oasis Library*'s method to insert dummy data into the database. *Oasis App* initiated the insertion of dummy data, and then we observed and fixed the errors through debugging until the insertion into *LocalDB* succeeded with the new database schema. The errors was observable using debug mode because it was mostly type cast and null pointers.

3.2 Evaluation of Sprint 1

In this sprint we managed to adapt the *Oasis Library* to the new database schema. There were minor details, which was not prioritized, such as the convertion of BLOB objects from the database to more useable objects such as Bitmap objects etc. Now the *Oasis Library* and *LocalDB* can interact with each other, and both can save and retrieve information to and from the database accordingly. The API have been completely updated, and from the start of sprint 2 the groups working on other parts of the multi-project can update and use the updated API. *Oasis App* has been updated, so it uses the updated API.

The future work on *Oasis Library* should be to provide a conversion of the BLOB type to relevant types. The *Oasis App* could be refactored and debugged, as it crashed several times during this sprint.

1st Sprint - Collaborative Writing: Database 4

This chapter is written in collaboration with SW609F14, SW612F14, and SW615F14. The chapter was written with the state of the database at the end of sprint 1 as basis. The structure of the database can therefore have changed.

4.1 The Structure of the Databases

The databases of the GIRAF environment is structured as shown in Figure 4.1. The main database is Remote DB, which each device synchronizes against. This enables users to log in at any device and have up-to-date information about themselves. The local database on each device is *LocalDB*, which applications use to store and retrieve information from. It is the task of Sync-lib to facilitate synchronization between the Remote DB and the Local DB, which can be initiated from a specific App, illustrated as App$_{Sync}$. This construction has been created to ensure that the GIRAF environment works even if the device is offline. The synchronization should be automatic.

To communicate with the local database, the applications in the GIRAF environment use the *Oasis Library*, which is an API to the local database. This has been created to ease the programming for the applications of GIRAF.

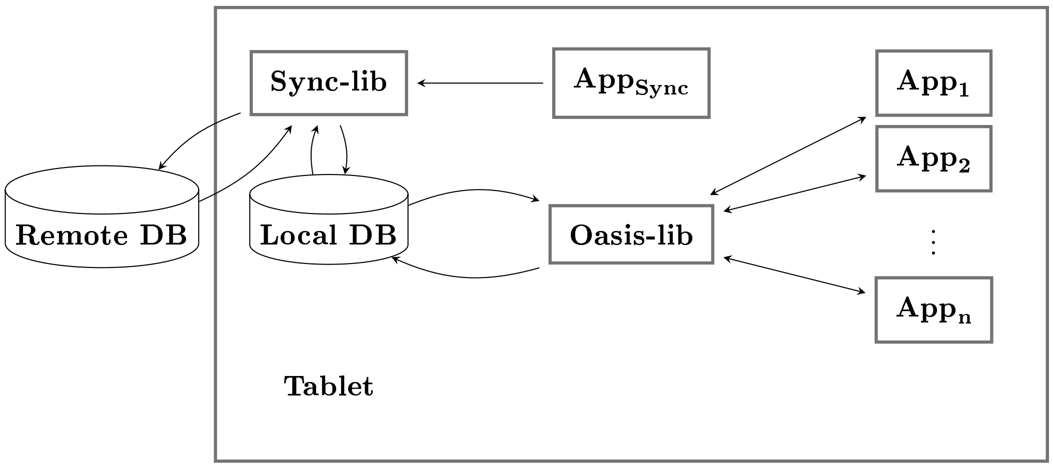

Figure 4.1: Overview of the Database Structure.

4.2 Remote database

The purpose of the remote database is to act as a data-central from where clients can synchronize their local database by sending and receiving updated content.

4.2.1 Database Management Systems

The system is currently running on MySQL, which is adequate for our purposes at the moment. However, if it is found that MySQL does not support needed features, another DBMS, such as PostgreSQL or Oracle, could be considered. It should be noted that Oracle is not free to use.

4.2.2 The Schema

The existing schema was created by students from previous semesters at Aalborg University [Flindt et al., 2013]. The schema is in Boyce-Codd normal form, and consists of a total of 16 tables, where nine of these are handling the relations between the seven remaining tables. These seven primary tables are, in no particular order, Pictogram, Category, Department, Profile, User, Application and Tag. The different relations is explained in the following section.

4.2.2.1 Pictogram

The *Pictogram* table stores image data with associated sounds and descriptions.

- Every *Pictogram* can have associated with it an author, which is the administrative *User* who created it.
- A many-to-many relation to *Tag* exists through the relation-table **pictogram_tag**.

4.2.2.2 Category

The *Category* table is used to group pictograms.

- A *Category* can be a child of another *Category*, specified by the foreign key **super_category_id**.
- A many-to-many relation to *Pictogram* exists through the relation-table **pictogram_category**.

4.2.2.3 Department

The *Department* table is used to group profiles by affiliation.

- A *Department* can be a child of another *Department*, specified by the foreign key **super_department_id**.
- Every *Department* can have associated with it an author, which is the administrative *User* who created it.
- A many-to-many relation to *Pictogram* exists through the relation-table **department_pictogram**.
- A many-to-many relation to *Application* exists through the relation-table **department_application**.

4.2.2.4 Profile

The *Profile* table stores relevant information about persons, such as contact information and roles.

- A *Profile* is connected to a specific *Department* through the foreign key **department_id**.
- A *Profile* can be associated with a *User* through the foreign key **user_id**.
- Every *Profile* can have associated with it an author, which is the administrative *User* who created it.
- A many-to-many relation to *Pictogram* exists through the relation-table **profile_pictogram**.
- A many-to-many relation to *Category* exists through the relation-table **profile_category**.
- A many-to-many relation to itself exists through the relation-table **guardian_of**.

4.2.2.5 User

The *User* table stores login information and is used to gain access to the system.

- A many-to-many relation to *Department* exists through the relation-table **admin_of**.

4.2.2.6 Application

The *Application* table stores meta information about applications and is used to control access to applications.

- Every *Application* can have associated with it an author, which is the administrative *User* who created it.

4.2.2.7 Tag

Tag is used to ease searching for pictograms.

4.3 Structure of the Oasis Library

The structure of the *Oasis Library* consists of models, controllers and metadata. The responsibility of the models and controllers is determined by following the Model-View-Controller pattern.

4.3.1 Model

The model describes the structure and handles the logic of the system. Change of data is handled in the model, and in case of major changes of data, the view is updated to display the current data.

4.3.2 View

The view displays the data, and it is used to interact with the system. The view interacts with the controller to request existing data and to modify or create new data.

4.3.3 Controller

The controller is the communication medium between the model and the view. It is only possible to access and modify data through the controller.

4.3.4 Implementation

In the *Oasis Library* there are only models and controllers. *Oasis Library* is used to communicate with the database, and therefore it has no graphical interface and hence no view. The views are represented by each application in the GIRAF environment. Metadata in *Oasis Library* is used to specify the information about the database, and it is not a part of the Model-View-Controller pattern.

Figure 4.2 shows the design of our implementation.

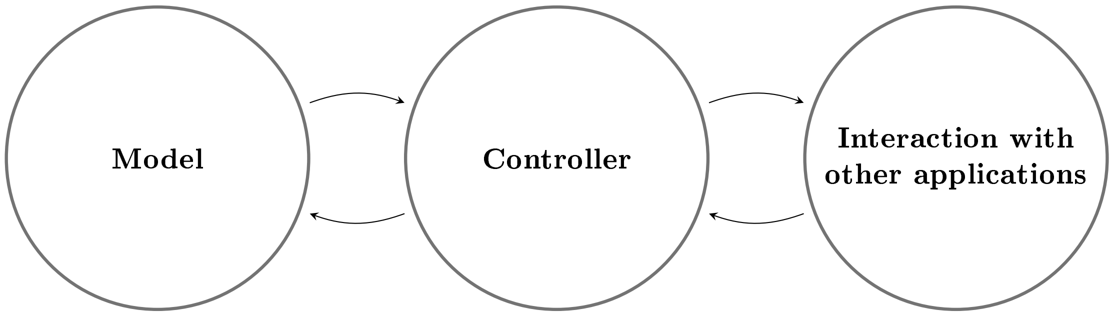

Figure 4.2: The structure of the *Oasis Library*.

4.4 Local database

As the applications run on the Android system, an obvious choice of database is SQLite, as it is fully supported on Android [Android Developers, b]. SQLite was also used by the previous database groups. An excellent tutorial for using SQlite is written by Vogel [Vogel, 2013], describing the use of both SQLite and `ContentProviders`.

4.4.1 SQLite

SQLite is a public domain, light-weight, embedded SQL database engine [SQLite, a]. It implements most of SQL-92 [Digital Equipment Corporation], but omits a few, for this project irrelevant, features [SQLite, b].

4.4.2 Implementation

The implementation of LocalDB follows the recommended approach for creating an SQLite database [Android Developers, b]. For *Oasis Library* to use the database, a class extending `DbProvider` [Android Developers, a] was implemented, giving access to the standard CRUD operations.

2nd Sprint - Parrot 5

In sprint 2 we worked with the application *Parrot*. Before we started refactoring and changing the project, we needed to know what the project was missing and where the bugs and shortcomings are located. Therefore we reviewed the previous code before we started the refactoring. Then we arranged a meeting with our costumers to specify further development of the application. After resolving issues with subprojects and the old data representation method, we went on to fulfill the specified requirements.

5.1 The Parrot Application

The *Parrot* application has been a part of the GIRAF project since its beginning in 2011. It is a communication tool, which should replace the current solution, which is a binder with laminated pictograms with velcro on the back. An example of this binder can be seen on Figure 5.1.

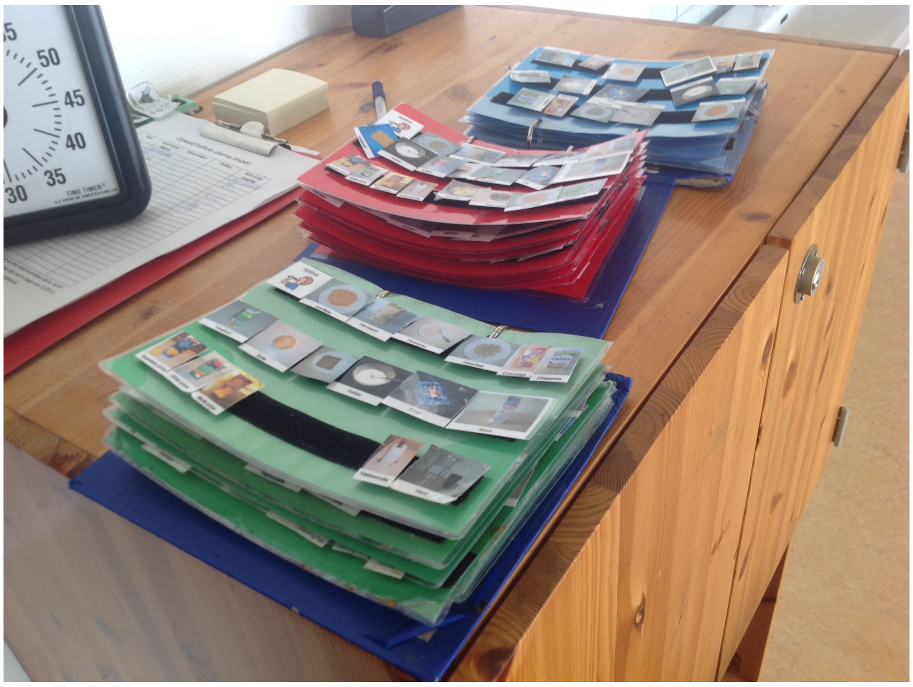

Figure 5.1: An example of the binder used now by people with autism to communicate.

The idea of *Parrot* is to have an overview of pictograms, sorted by categories, which can be dragged down to a bar at the bottom. This makes it possible for the user to construct a sentence with the use of pictograms, and by doing so communicate his/her message to the guardians. It should therefore be very similar to the way it works today, but just in a digital version. This gives a lot of possibilities, like the opportunity of using sound.

5.2 Startup Problems with Parrot

Since we were the first group working on *Parrot* this semester, we had some startup problems with the application. When we first built the application and installed it on the tablet, it would not start. Through debugging with step-by-step evaluation of the code, we found that the link between users and pictograms was nor stored or read from the database, but in a custom made XML-file. As we did not have this XML-file on our tablet, it closed unexpectedly. To solve this, we found a copy of the XML-file in the report for *Parrot* from last year, in which we corrected the typos, syntax errors and other errors. We then copied the XML-file to the right location on the tablet and downloaded some pictograms which paths were hardcoded in *Parrot*. When all this was done, the application could finally open as intended.

A screenshot of *Parrot* in its initial state can be seen in Figure 5.2.

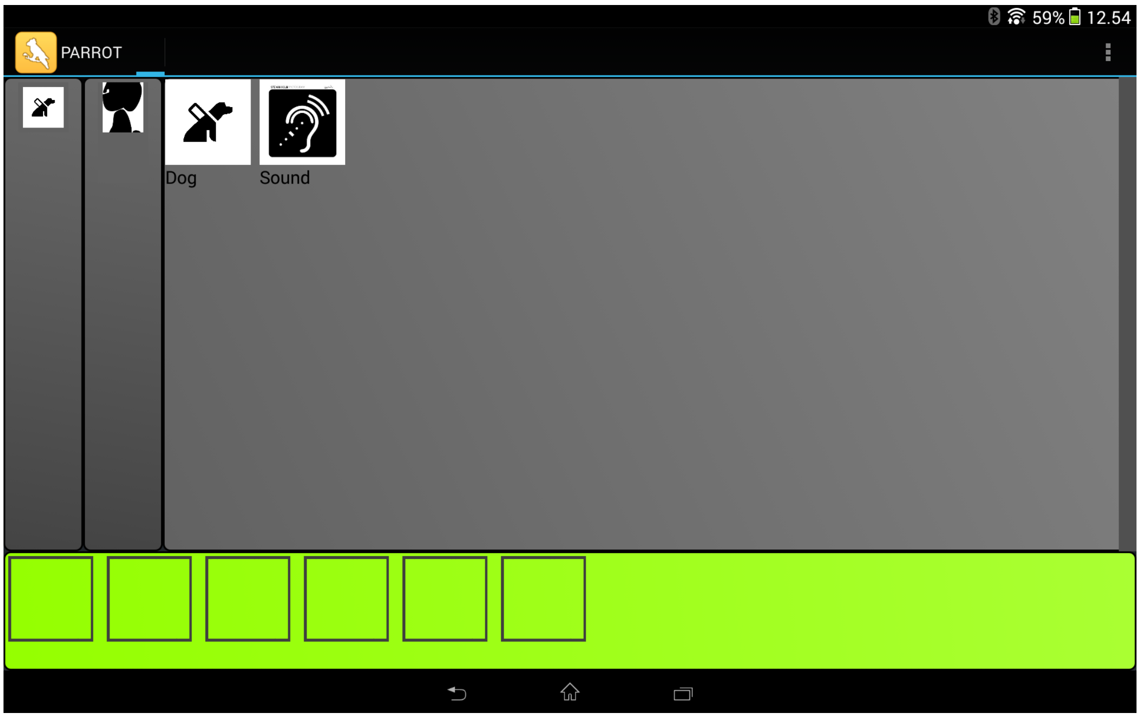

Figure 5.2: This is the old design of *Parrot* developed in the spring of 2013.

5.3 Migration to the new Database Structure

As mentioned in Section 5.2, the version of *Parrot* we inherited from the former semester used XML files for storing information, and not the database. Some information about the profile among other things was however read from the old version of the database, so we should therefore migrate *Parrot* to use the new database schema (see Chapter 3).

Our first problem was to fix *Parrot*'s code, so it matched the new database schema. Nearly all of the errors, were because the methods had been renamed, or moved to another controller due to the new structure. These errors were easy and quickly to fix. There were also more complicated errors fixes. Some methods did not exist in the new *Oasis Library*, such as `getApplicationByPackageName`. For these, we had to identify the missing methods and then talk with the new group in charge of the *Oasis Library*. With these methods fixed, there were no more unknown method calls in the code from *Parrot*.

Next problem was the submodules *Parrot* depended on. The submodules had not yet been updated to the new database schema, which conflicted with our updated code. This was a problem throughout the whole sprint, since some groups was very slow to update their code to the new schema due to migration problems. This complicated the development on *Parrot* a lot this sprint, but as the submodules were corrected, we could continue on *Parrot*. As we debugged our code, we found some migration errors in some of the submodules, from which we opened issues with the related groups. An example is that we found a null pointer exception in *Oasis Library* when calling one of their methods.

The migration problems in this sprint was a lot about working together with the other groups, figuring out where the problem was. There was a lot of waiting time, where we had to work on other parts of the project, since the problem was out of our hands.

5.4 Costumer Meeting

To specify the requirements for the Parrot project we met with Drazenko Banjak from Egebakken, which is a school for children with autism, and Pernille Hansen Krogh from MyMo. We conducted a semi-structured interview because we had some specific topics we wanted to address, but at the same time we wanted to hear suggestions from the costumers.

Due to the customers limited knowledge about software development, we focused on specifying the front-end experience. We ended up with the list of requirements as seen below.

- It should be possible to add a pictogram to the sequence in two ways. A single click on the pictogram should send it to the next place in the sequence, and it should also be possible to drag a pictogram to an arbitrary place in the sequence.
- While a pictogram is dragged it should be enlarged so it is possible to see around ones finger. When releasing the pictogram it should return to its original size.
- A sequence should not, by default, be forced to be n pictograms long. It should however be possible to restrict the sequence to be less than m pictograms long. If not restricted, the sequence should just expand every time a pictogram is added.
- It should be possible to remove a pictogram from the sequence in two ways. A long click on one of the pictogram in the sequence should display little remove icons above all pictograms in the sequence. The remove icon should only remove the pictogram which it is connected.
  Swiping a pictogram off the sequence should also remove it from the sequence. When removing a single pictogram from the sequence the previously occupied area should be left blank, so one can drag a pictogram to that area. Further all other pictograms in the sequence should remain where they were placed.

- It should be possible for a guardian to select eligible pictograms. This is to avoid a child wanting a cereal that is not in stock or equivalent situations.
  When a guardian have selected eligible pictograms, the category selection should slide out the screen so the children cannot access or see it. The child should therefore only have the opportunity to select the pictograms the guardian have just selected.
- It should be possible to adjust the display size of pictograms. Effects should affect all pictograms and should not be possible for individual pictograms.

5.5 Design of Parrot

The design of *Parrot* is based on the meeting with the customers, as described earlier, and on the existing design of *Parrot*, which was developed in the spring of 2013.

Figure 5.2 shows the initial design of *Parrot*. Our options were to continue the development of the old *Parrot*, or start from the beginning on a new version of *Parrot*. Not having to start from scratch was a big advantage, especially considering the time it would take to develop a new *Parrot*, and due to the work already made in *Parrot* we chose to work with the old *Parrot*. It was stated that it was more important with working code, compared to pretty code, which also supports the choice of working with the old base of *Parrot*.

5.5.1 Design Choices

After interviewing the customers, we deduced some issues with the old *Parrot* design that should be changed. Some major changes are listed below.

- It was decided to make three large buttons on the right hand side of the screen. One button for settings, one that empties the sentence board and one that plays the sequence of pictograms. Playing a sequence of pictograms should refer to playing the sound associated with the pictograms in order.
- When pictograms have been placed in a sequence in the bottom of the screen, pressing the play button should highlight the selected pictograms and move them to the center of the screen. This makes the child focus only on the sequence of pictograms while it is being read.
- In the old *Parrot* application, selected categories was not highlighted, and therefore it was not possible to see which category one was working in. Our *Parrot* design includes highlighting of the chosen category.
- The new *Parrot* will offer a guardian to select an amount of pictograms which the child can choose from.

There are many more changes than the above mentioned, but they will not be described in this section.

5.6 Evaluation of Sprint 2

After the migration to the new database schema, we corrected some bugs and simplified the source code, so maintenance and future development would become easier. There were also made some minor changes to the options fragment.

5.7 Tasks for 3. Sprint

The tasks for the third sprint was first and foremost the tasks which where originally scheduled for sprint 2, but which was not completed before the deadline. The main reason for the delays was errors and delays in the support modules of parrot. A few of the tasks which where moved from sprint 2 to sprint 3 were:

- Use G-components for layout
- Play audio
- The application crashes when filling the sentence board too quickly

We were unable to test the application with pictograms because the CategoryLibrary had not yet been updated to use the new schema of the *LocalDB*.

The focus of *Parrot* in sprint 2 was to make parrot run, and to migrate to the new *Oasis Library*.

3rd Sprint - Parrot 6

In this sprint our focus was to solve the remaining tasks described in Section 5.7 and to implement the requirements given by the costumers. It was further decided at a Scrum-of-Scrums meeting that we should start to investigate and implement a method for loading data into the remote database. Further it was decided that all front-end titles should be changed to a Danish explaning title. The *Parrot* application was named *Piktooplæser*.

6.1 Graphical User Interface

This section explains the changes made to the interface of *Piktooplæser*. An overview of the differen components in the interface can be seen in Figure 6.1

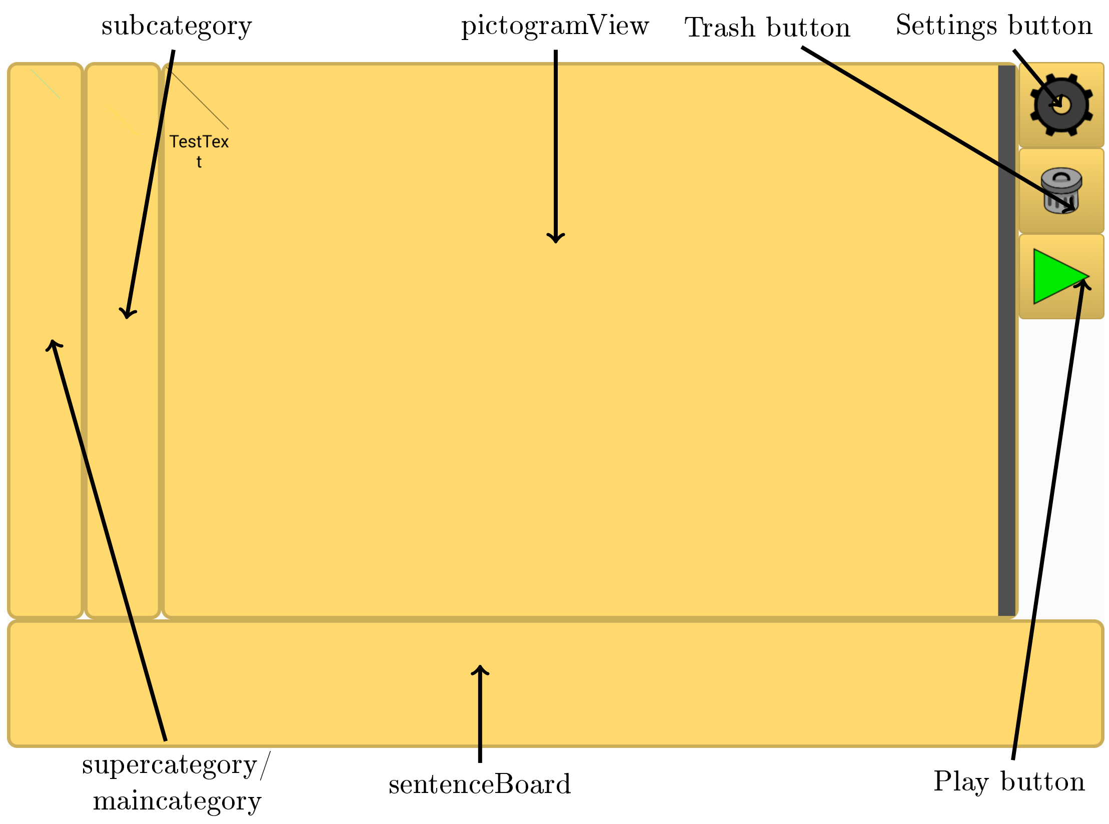

Figure 6.1: Picture of *Piktooplæser* at the end of sprint 3, with explanation of the different components

Figure 6.2 shows the design of *Piktooplæser*, before we began our work. Figure 6.3 shows our design of *Piktooplæser* after third sprint in the multi project.

The main difference is the change of color and the change from regular components to Giraf components (GComponents), which is a selection of common graphical components for this project [Bender et al., 2014]. The rectangular box in the bottom is called the sentence board, and the large box in the middle is called the pictogram grid. The surrounding boxes of slots for the pictograms in the sentence board have been removed. When pictograms are dragged from the pictogram grid, they are enlarged to make it easier to see the pictogram while holding it with a finger. When pictograms are dragged to the sentence board, they are placed in the slot they are dropped in, which makes it possible to select pictograms in a random order. On the right, three buttons have been created. The first one is a settings button, the second is a trash bin which empties the sentence board, and the third button is a play button that removes empty slots between pictograms in the sentence boards and moves the pictograms to the left. Also, the play button plays the associated sound of the pictograms in the sentence board. If sound is recorded for the pictogram, that sound is played, else we have implemented text to speech that reads the name of the pictogram.

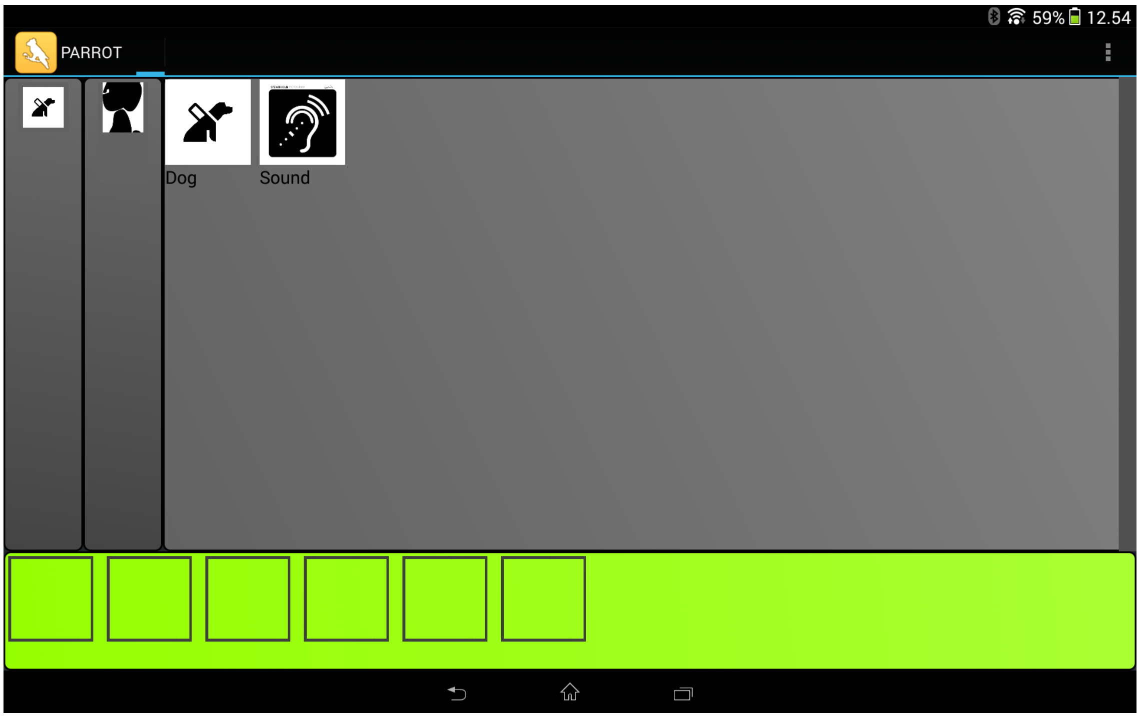

Figure 6.2: Old Parrot

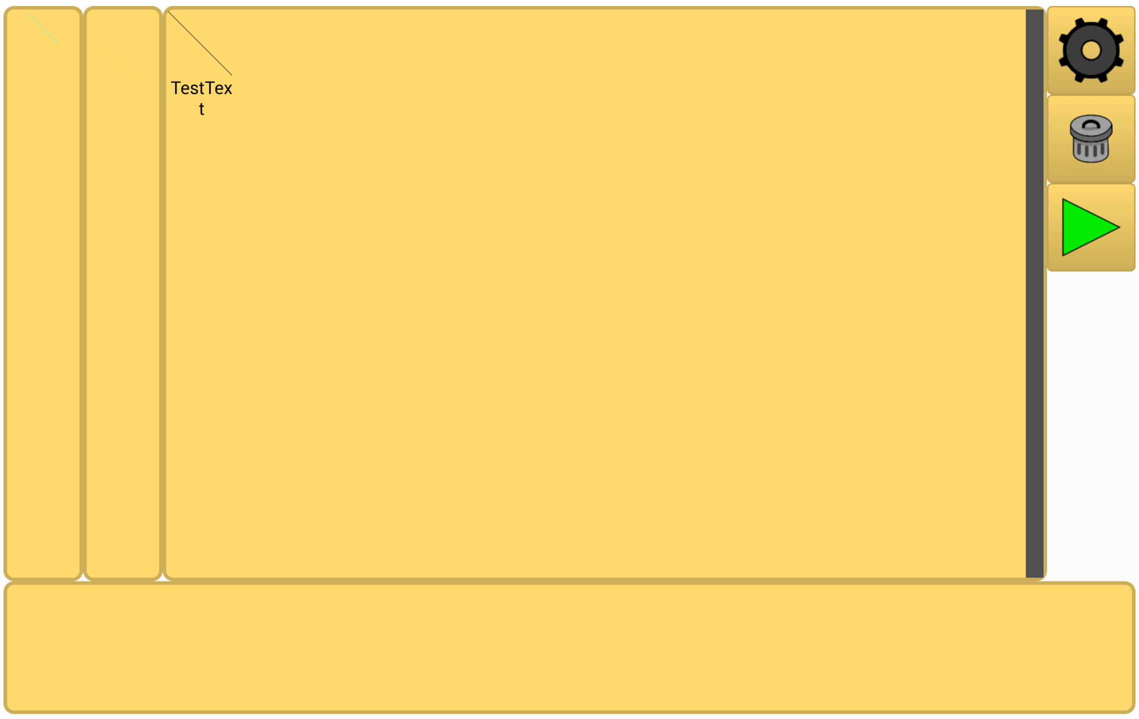

Figure 6.3: New Parrot

6.2 Drag Pictograms to Senteceboard

The user of the application should be able to drag pictograms from the gridview with the current showed pictograms, down to the senteceboard. The sentenceboard is a gridview with a custom adapter, which accepts a list of pictograms. To update the sentenceboard, the gridview is set with a new instance of the adapter with the updated list of pictograms as input.

To enable the drag, an onTouchListener and an onDragListener is set to the gridview with the possible pictograms. The onTouchListener stores the index of the pictogram, when it is touched. When the drag is started, the dragged pictogram is stored as seen on Listing 6.1

```java
@Override
public boolean onDrag(View self, DragEvent event) {
  if (event.getAction() == DragEvent.ACTION_DRAG_STARTED){
    //When pictogram is dragged from sentenceboard
    if(self.getId() == R.id.sentenceboard && SpeechBoardFragment.
        dragOwnerID == R.id.sentenceboard)
    {
        draggedPictogram = SpeechBoardFragment.pictogramList.get(
            SpeechBoardFragment.draggedPictogramIndex);
```

Listing 6.1: The dragged pictogram is stored when the drag is started.

When the drag event "ACTION_DROP" occurs, it is first checked from which gridview the pictogram is dragged. Hereafter it is calculated in which location on the sentenceboard the pictogram is dropped. This can be seen on Listing 6.2.

```java
GridView speech = (GridView) parrent.findViewById(R.id.
    sentenceboard);
int x = (int)event.getX();
int y = (int)event.getY();
int index = speech.pointToPosition(x, y);
```

Listing 6.2: Calculates the position on the sentenceboard where the pictogram is dropped.

Hereafter the pictogram on the given place is replaced with the dragged pictogram, and the gridview is updated.

To enable dragging a category down to the sentenceboard, a catagory must be converted to a pictogram, as the adapter to the gridview only accepts a list of pictograms. If the pictogram is dragged from one of the category gridviews, the category is converted, and added to the list as seen on Listing 6.3.

```
Category cat = SpeechBoardFragment.displayedCategory;
draggedPictogram = new Pictogram();
draggedPictogram.setName(cat.getName());
draggedPictogram.setInlineText(cat.getName());
draggedPictogram.setImage(cat.getImage());
draggedPictogram.setId(cat.getId()*-1);
```

Listing 6.3: The category is converted to a temporary pictogram

6.3 Play Button

When pressing the play button in *Piktooplæser*, the pictograms in the sentence board are moved to the left, eliminating all empty spaces between the pictograms. Listing 6.4 shows the implementation of eliminating all the empty spaces. The play button also plays the sound of the pictograms in the pictogram list, and this is described in Section 7.1. In the first line `change` is used to indicate whether a pictogram has been moved or not. If `change` is set to false, it means that there are only `null` value pictograms that can be moved, which have no visual effect on the pictogram list. On line 9-16, if the end of the pictogram list has not been reached, the pictogram at index `j` is moved to index `j-1`. On line 19-21, the changes made to the sentence board list are assigned to the grid view and shown in the application.

```
 1 boolean change;
 2 for(int i = 0; i < pictogramList.size(); i++)
 3 {
 4     change = true;
 5     while(change && pictogramList.get(i) == null)
 6     {
 7         change = false;
 8         for (int j = i + 1; j < pictogramList.size(); j++)
 9         {
10             if(pictogramList.get(j) != null)
11             {
12                 pictogramList.set(j-1,pictogramList.get(j));
13                 pictogramList.set(j,null);
14                 change = true;
15             }
16         }
17     }
18 }
19 GridView sentence = (GridView) parrent.findViewById(R.id.
       sentenceboard);
20 sentence.setAdapter(new SentenceboardAdapter(pictogramList, parrent
       ));
21 sentence.invalidate();
```

Listing 6.4: Code for eliminating empty spaces in the sentence board list when the play button is pressed.

6.4 Data Insertion

In order to test the applications in the GIRAF framework properly, there needed to be inserted some data into the remote database. This data should then be synchronized with the local database. We contacted Martijn van der Kooij from the Netherlands, who maintains the program Pictoselector [van der Kooij], and therefore has a lot of pictograms which are already divided into categories. From him we received a list of categories and associated pictograms ID's, a translation list from pictogram ID's to dutch filenames, and a translation list from dutch filenames to danish filenames. In total there were 27,772 pictograms. Of these about 14,000 pictograms was categorized. We made a small C# program to insert these pictograms into the database. The program does the following:

1. Goes through the categories and maps which pictograms belongs to which categories.
2. Translates the pictograms from Dutch to Danish via the provided list with translations.
3. Moves the translated pictogram to a folder with the name of the category.
4. Inserts the categories and pictograms into the database.
5. Create relations between pictograms and categories in the `pictogram_category` table.

The program also creates and inserts sample data into each table for the testing purposes.

6.5 Summary of Sprint 3

In sprint 3 we focused on making the look and feel of *Piktooplæser* better. We made it possible to drag pictograms to a specific locations on the sentence board. We created the *Data Inserter* for inserting data into the remote database. Lastly we collaborated with the group of *Pictocreator* to write the Text-To-Speech, so that pictograms with no recorded sound can be played as seen in Chapter 7. Further we collaborated to create a media player to play the sound of pictograms with sound recorded.

6.6 Tasks for 4. Sprint

There are a few tasks which should be the main focus for sprint 4, these are:

- Select specific pictograms, and allow selection from these and hide categories in this mode.
- Marking of selected categories.
- Shortcuts to *Piktotegner* and *Kategoriværktøj*

3rd Sprint - Collaborative Writing: Pictogram Library 7

This chapter is written in collaboration by SW608F14 and SW615F14.

Sound for pictograms was deemed important by the customers, and therefore, *Piktotegner* and *Piktooplæser* chose to cooperate in developing libraries for this. The libraries are mainly split into two parts, a media player and a Text-To-Speech library. The cooperation for developing the libraries was done by way of pair programming, with one person from each group per library. Thus, a mutual agreement was found for how the libraries should be developed. A description of these as well as motivation and a description of their implementation will be given hereafter.

7.1 PictoMediaPlayer

Associating sound to pictograms was stated as of great importance from the customers. For that reason a library has been developed that provides an easy to use sound playing service.

Both *Piktotegner* and *Piktooplæser* need the sound playing feature. *Piktotegner* needs it to get a preview of the sound recorded, to be saved when creating a pictogram. *Piktooplæser* needs the feature as the key feature of *Piktooplæser* is to get the sound of the pictograms played sequentially.

When initialising the library it needs a context and can additionally take a path for the sound file, a pictogram, or a byte array. The context is needed for the `PictoMediaPlayer` to know where to play the sound. In Listing 7.1 we see a constructor for the `PictoMediaPlayer`.

```java
public PictoMediaPlayer (Context activity, String path)
{
    this.activity = activity;
    assignMediaPlayer();
    setDataSource(path);
}
```

Listing 7.1: Constructor for `PictoMediaPlayer`.

When assigning a datasource to the `PictoMediaPlayer` the *setDataSource* method, seen in Listing 7.2, is called. The method releases an old `MediaPlayer` and assigns a new one if sound has already been attached to the `MediaPlayer`. Then the `MediaPlayer` gets a file descriptor assigned as source.

```java
public void setDataSource(String path)
{
    if(hasSound)
    {
        mediaPlayer.release();
        assignMediaPlayer();
    }

    try
    {
        FileInputStream fileInputStream = new FileInputStream(path)
            ;

        mediaPlayer.setDataSource(fileInputStream.getFD());
        hasSound = true;
    }
    catch (IOException e)
    {
        e.getStackTrace();
    }
}
```

Listing 7.2: `SetDataSource` method of `PictoMediaPlayer`.

The `OnCompletionListener`, seen in Listing 7.3, is necessary for external sources to detect when a sound is finished playing. The listener allows external sources to respond to a sound finished playing, e.g. a button icon change.

```java
private final MediaPlayer.OnCompletionListener onCompletionListener
    = new MediaPlayer.OnCompletionListener()
{
    @Override
    public void onCompletion(MediaPlayer mp)
    {
        isPlaying = false;
        if(customListener != null)
        {
            customListener.soundDonePlaying();
        }
    }
};

public interface CompleteListener
{
    public void soundDonePlaying();
}
```

Listing 7.3: `onCompleteListener` method of `PictoMediaPlayer`.

For *Piktooplæser* to play the sound of multiple pictograms in succession we made the `playListOfPictograms` method as seen in Listing 7.4. We first store the old listener in a temporary variable and stop playing sounds. Then we overwrite the method which is run after a sound is complete, so that it plays the sound of the next pictogram in the list. When all pictograms in the list have been read the old listener is restored.

```java
public void playListOfPictograms(ArrayList<Pictogram> pictogramList
    )
{
    pictogramListIndex = 0;
    this.pictogramList = pictogramList;
    TempCompleteListener = customListener;

    if(isPlaying)
    {
        stopSound();
    }
    this.setCustomListener(this);
    this.setDataSource(pictogramList.get(pictogramListIndex));
    this.playSound();
}

@Override
public void soundDonePlaying()
{
    pictogramListIndex ++;
    if (pictogramListIndex < pictogramList.size() && pictogramList.
        get(pictogramListIndex) != null)
    {
        setDataSource(pictogramList.get(pictogramListIndex));
        playSound();
    }
    else
    {
        this.setCustomListener(TempCompleteListener);
        pictogramListIndex = 0;
    }
}
```

Listing 7.4: `playListOfPictograms` method of `PictoMediaPlayer`.

To illustrate the ease of use of the `PictoMediaPlayer` we created the example seen in Listing 7.5. We here see how a `PictoMediaPlayer` is used to play a sound and writing "Sound is finished playing" when the sound is finished playing.

```java
public class example implements CompleteListener
{
    private void examplemethod(Context context, String path)
    {
        PictoMediaPlayer pictoMediaPlayer = new PictoMediaPlayer(
            context, path);

  pictoMediaPlayer.setCustomListener(this);
        pictoMediaPlayer.playSound();
    }

    @Override
    public void soundDonePlaying()
    {
        System.out.print("Sound is finished playing");
    }
}
```

Listing 7.5: Example of `PictoMediaPlayer`.

7.2 Implementing Text-To-Speech

The motivation for creating a Text-To-Speech (TTS) functionality is that the customers should be able to play the sound of all pictograms, even if a pictogram does not have a sound associated with it. The reason why the customers need to play a sound of a pictogram is to teach the autistic people to associate a pictogram with a sound. The *Piktooplæser* application should use the TTS functionality whenever a pictogram without a sound is added to the sequence to be read. Furthermore, *Piktooplæser* should store the newly created TTS sound, so the pictogram have the specific sound for later usage. The newly generated TTS sound should therefore be stored in the local database, which will synchronize the updated pictograms with the other users, such that the sound is only generated once. For *Piktotegner*, with pictograms created by the customer where no sound is recorded, a sound should be created by reading the inline text of the pictogram. This should of course also be stored to the database for later usage.

7.2.1 Available Text-To-Speech Tools

To enable the TTS function, the different available TTS tools should be compared, so the best suitable tool for this project can be chosen. There are a lot of different TTS tools, but there are some requirements:

- The tool must be free to use.
- The tool must be able to speak the given text in danish.
- The tool must be usable with an android application.

From these requirements, the build-in Android TTS can be excluded, because it does not support the danish language. Tools which fulfil these requirements have been found and the ones we chose to compare are listed below:

- Voice RSS
- eSpeak
- JeSpeak
- Google Translate TTS

7.2.1.1 Voice RSS

Voice RSS [Voice RSS] is an online TTS tool with an API. Voice RSS has a Danish voice for its TTS, but has a limited uses. It is only the first 350 TTS requests per day that are free, thereafter there must be paid a fee for more requests. The sound of the TTS is relatively good, and the Danish voice is understandable. To make a request to Voice RSS, the device must be connected to the internet.

7.2.1.2 eSpeak

eSpeak is a free open source TTS build for Linux and Windows with many supported languages, including Danish [Duddington]. The advantage of eSpeak is that it is free to use, and do not need internet access to work. The disadvantage is that it is not made directly for use with Android.

7.2.1.3 JeSpeak

JeSpeak is a Java Library built from eSpeak [Belanger]. The advantage of this TTS tool is that it is a Java Library which makes it compatible with the Java written GIRAF project. The downside is that it is not very well documented and requires not only Android SDK, but also Android NDK [Android Developers, c].

7.2.1.4 Google Translate TTS

Google Translate TTS is a possibility to use their API to make web based request and receive a sound file. It is very similar to Voice RSS, but has higher limit on the number of requests. It has a wide range of languages, including a quite good Danish TTS.

7.2.1.5 Comparison and Selection of TTS Tool

As seen, there are four examined solutions for enabling TTS in the GIRAF project. A comparison of the different TTS tools can be seen in Table 7.1.

TTS Tool	**Advantage**	**Disadvantage**	**Requires Internet Access**	**Price**
Voice RSS	Easy, well documented API, under-standable voice	Requires internet access, limit in requests	Yes	Free up to 350 requests per day, hereafter a fee starting from 5$
eSpeak	Free, does not require internet access	Not built for use with Android	No	Free, Open Source
JeSpeak	Free, made for Java	Not made directly for use with Android.	No	Free
Google Translate TTS	Free, supports many languages, easy to implement	Requires internet access	Yes	Free

Table 7.1: Comparison table for different TTS tools.

Due to this comparison, Google Translate TTS has been chosen. It is very easy to implement, compared to e.g. JeSpeak, it is free to use, and it has a good Danish TTS.

7.2.2 Handling TTS

As previous mentioned, Google Translate TTS is chosen as the TTS tool for this project. As it is a web-based tool, it must be considered how it should be used, and how to handle the situation where there is no internet connection. There is also the possibility that the creator of the pictogram has recorded his own reading of the pictogram. In this case the TTS should not be used. From these statements, a flowchart of the behavior of playing a pictograms sound can be seen on Figure 7.1.

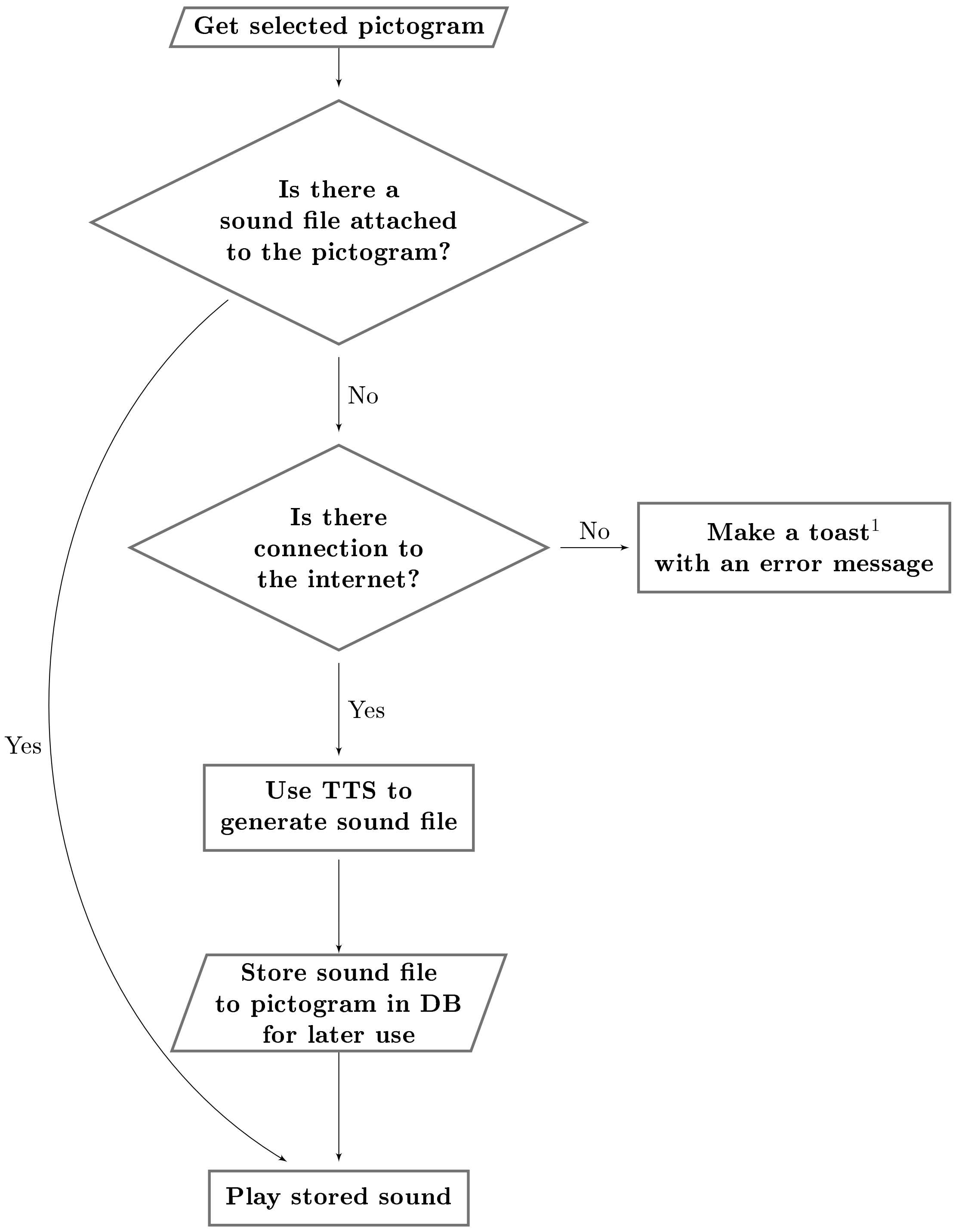
Get selected pictogram
Is there a sound file attached to the pictogram?
No
Is there connection to the internet?
No
Make a toast[1] with an error message
Yes
Yes
Use TTS to generate sound file
Store sound file to pictogram in DB for later use
Play stored sound

Figure 7.1: Flowchart for the use of sound on pictograms.

With the behavior in place, the TTS and soundplaying method can now be implemented. Note that once the sound has been generated, it is stored to the pictogram in the database, so it should only be generated once. The updated pictogram is synchronized with the remote database, so every other tablet also gets the sound file.

[1]A toast is a small message on the screen [Android Developers, d]. The GToast, which is a customization of the Android toast, is used.

7.2.3 Implementation

The implementation of the TTS functionality is created by opening an connection to Google Translate's TTS method. When the connection is established, a stream starts to return bytes. These bytes are then saved in an array so they can be translated into sound afterwards. How to get the bytes and save them can be seen in Listing 7.6.

```java
private void DownloadFile() {
    try{
        URL url = new URL(soundURL);

        URLConnection urlConnection = url.openConnection();
        InputStream inputStream = urlConnection.getInputStream();
        BufferedInputStream bis = new BufferedInputStream(
            inputStream);
        ByteArrayBuffer byteArrayBuffer = new ByteArrayBuffer(50);
        int current = 0;
        while ((current = bis.read()) != -1)
            byteArrayBuffer.append((byte) current);

        SoundData = byteArrayBuffer.toByteArray();
    }[...]
}
```

Listing 7.6: Download Sound method.

Before calling this method, it must be ensured that the device has a valid internet connection, because if there is no internet connection you are not able to connect to Google Translate. To check if there is an internet connection, a connection manager checks if it can get network activity. If the connection manager does not find an internet connection the `DownloadFile` method cannot be called.

4th Sprint - Parrot 8

In the beginning of sprint 4, we had almost fulfilled all requirements from the costumer meeting, see Section 5.4. In this sprint we focussed on fulfilling the rest of the requirements from the costumer meeting where after we turned to fulfilling requirements expressed at the 3rd sprint end.

8.1 The Components in Piktooplæser

The final design of *Piktooplæser* can be seen on Figure 8.1. Throughout this chapter, the different elements in the design of the application will be explained, and the different elements can be seen pointed out in the figure.

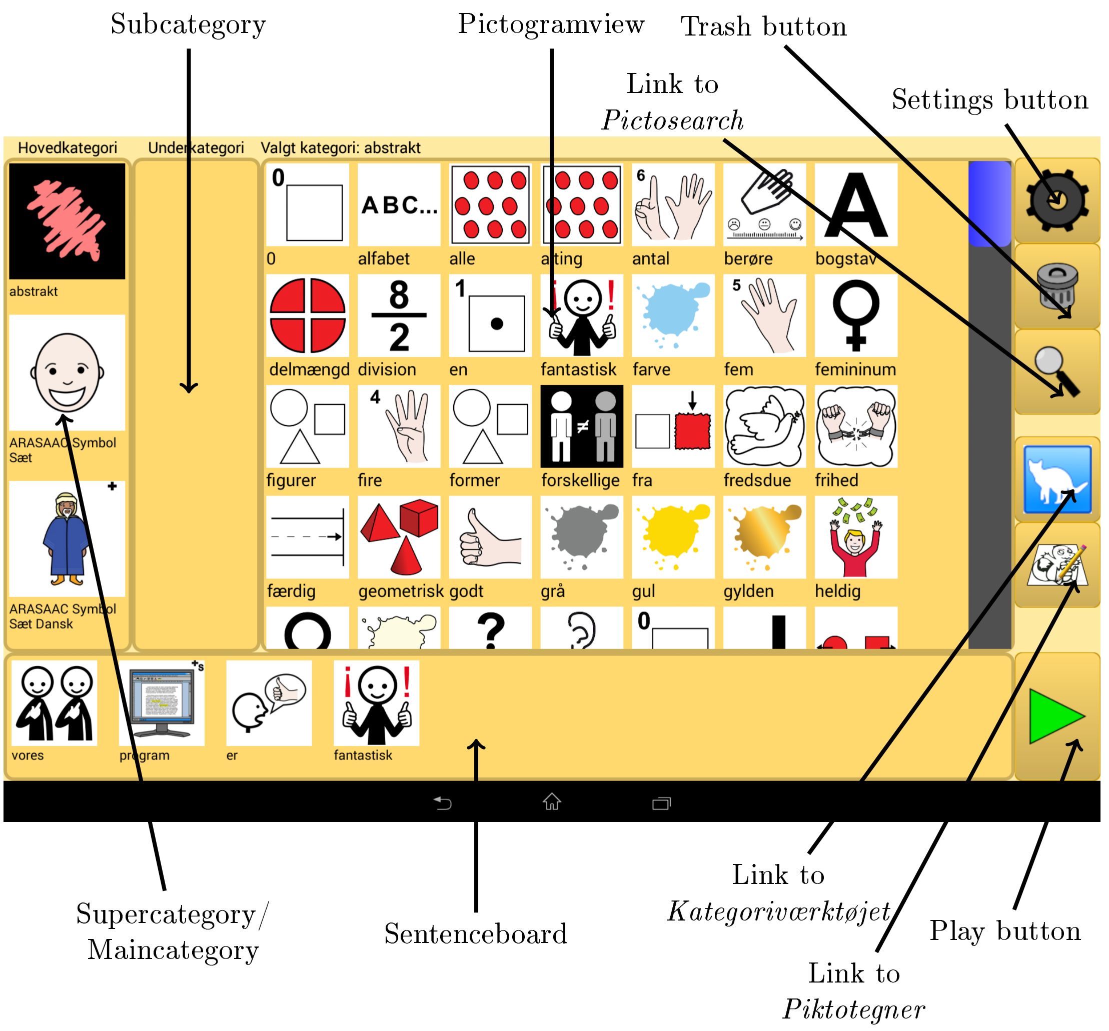

Figure 8.1: Picture of the final version of *Piktooplæser*, with explanation of the different components.

8.2 Simple Pictogram Mode

One of the requirements from the customers was that a guardian should be able to choose some pictograms, which the citizen can then choose from. This mode is called the simple mode. In this mode there are no categories available to choose from - these are removed to make space for a larger grid for the pictograms. Other than the categories being hidden, the application works as normal: A citizen can choose one or more pictograms from the grid and drag it to the sentence board and press play. Figure 8.2 shows the design of simple mode.

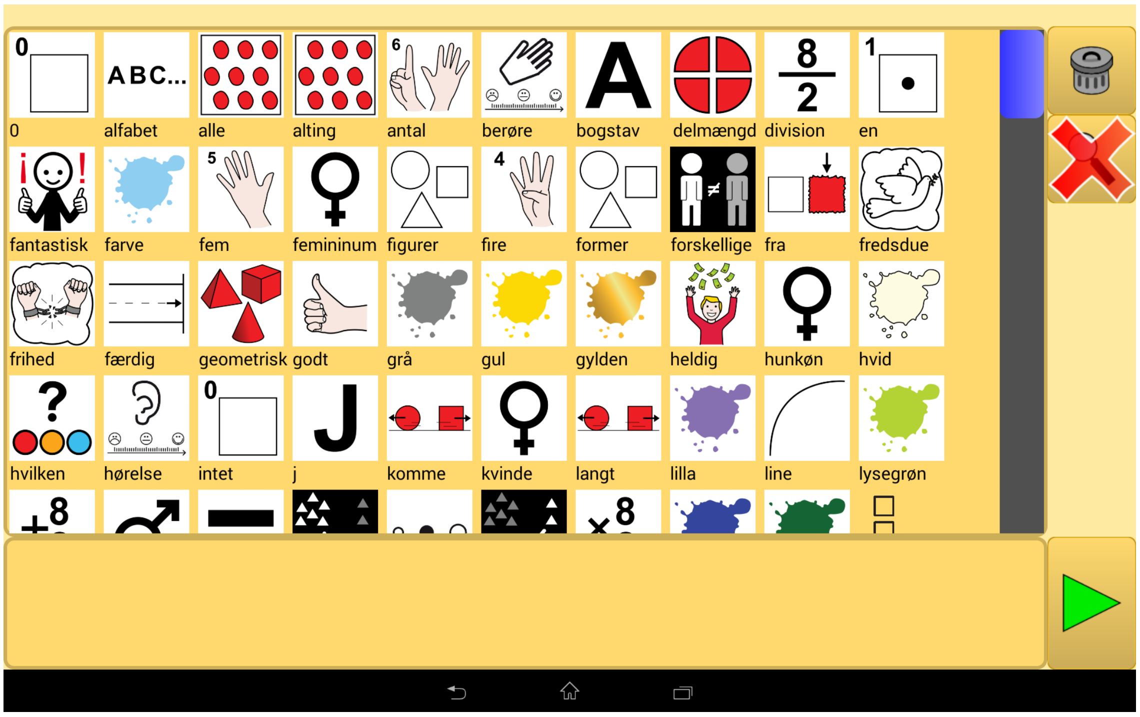

Figure 8.2: This is the design of simple mode.

The functionality is implemented very simply: The idea is that on the press of a button, *Pictosearch* is opened. *Pictosearch* then handles the selection of pictograms and returns a list of pictogram ID's. These ID's are then translated into a list of actual pictograms which are inserted into the pictogram grid. Then the super- and sub-category grids are hidden, and the width of the pictogram grid is expanded, so that it fills the entire width of the screen except the space for the buttons. Futhermore, the buttons with links to *Piktotegner* and *Kategoriværktøjet* is hidden in simple mode.

Listing 8.1 shows the implementation that shows how the array `pictogramIDs` is populated with the id's returned by *Pictosearch*. This list is used to populate the pictogram grid as outlined above. *Pictosearch* is called for the guardian to create a list of pictograms that the citizen can select from. The `loadPictogram` method is called after the list of pictograms is created.

```java
private void loadPictogram(Intent data){
    int[] pictogramIDs = {};
    try{
        pictogramIDs = data.getExtras().getIntArray("checkoutIds");
    }
    catch (Exception e){
        e.printStackTrace();
    }

    List<dk.aau.cs.giraf.oasis.lib.models.Pictogram>
        selectedPictograms = new ArrayList<dk.aau.cs.giraf.oasis.lib
        .models.Pictogram>();
    for (int i = 0; i < pictogramIDs.length; i++)
    {
        selectedPictograms.add(pictogramController.getPictogramById
            (pictogramIDs[i]));
    }
    displayPictogramList = selectedPictograms;
}
```

Listing 8.1: The loadPictogram method

The button which initiates *Pictosearch* has two functionalities: If you are in the normal mode, the button starts *Pictosearch* and changes the mode to the simple mode. If you are in the simple mode, the *Pictosearch* icon is shown with a red cross which indicates that *Pictosearch* is closed, and the press of the button will return you to the normal view.

8.3 Overview of the Flow in the Application

The diagram seen on Figure 8.3 is an overview of the flow in the application. When the application starts, it is in the center state of the diagram. From here the user can for example press play, clear the sentence board and go to options. If a user presses the options button, the application will go to the options state, and the user can change the options and go back to the main state again. From the main state the user can also press the *Piktotegner* button, which opens the *Piktotegner* application.

When the user is done in *Piktotegner*, he will be taken back to the main state. He can also press the *Kategoriværktøjet* button to open the *Kategoriværktøjet* application. From the main state it is also possible to use the *Pictosearch* application. When the button is pressed, *Pictosearch* opens and the user can select some pictograms. If he presses confirm, he is taken to the simple mode in *Piktooplæser*, where the selected pictograms from *Pictosearch* appears in the pictogram grid. From the simple view, the user cannot access *Piktotegner* or *Kategoriværktøjet*. However, he can exit simple mode and return to the normal state.

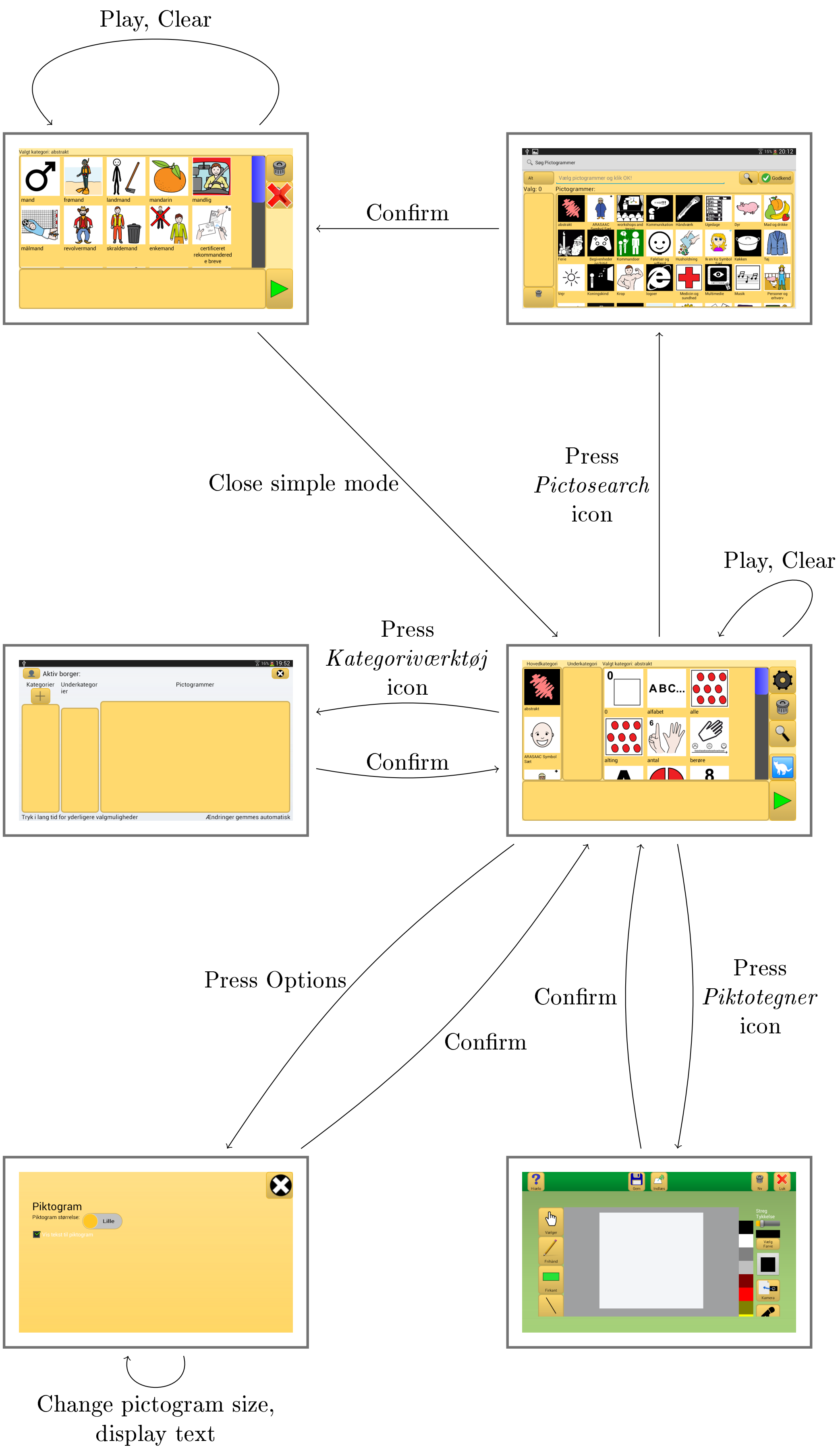

Figure 8.3: Flowchart for the use of *Piktooplæser*.

8.4 Setting the Number of Columns in Gridviews

Through the tests on the different tablets available, a problem with the number of columns on the two gridviews, the senteceboard and the pictogram grid, occurred. The number of columns was hardcoded to 7 columns. This worked well on a big screen, but on a smaller screen, this would be to many columns, and some of the pictograms would then be outside the view of the screen. To solve this, it was decided the number of columns should be calculated given the current screen size. To calculate the number of columns in the sentenceboard, the width of the screen was substracted with the width of the play button and then divided by the width of a column. This can be seen in Listing 8.2.

```java
Display display = getActivity().getWindowManager().
    getDefaultDisplay();
Point size = new Point();
display.getSize(size);
int width = size.x;

int buttonsWidth = 100;
int colWidth = GComponent.DpToPixel(125, parrent.
    getApplicationContext());
sentenceBoardGrid.setColumnWidth(colWidth);
noInSentence = (width-GComponent.DpToPixel(buttonsWidth, parrent))
    /(colWidth);
sentenceBoardGrid.setNumColumns(noInSentence);
```

Listing 8.2: The calculation of the number of columns in the sentenceboard.

A similar approach is used on the pictogram grid, but there had to be some modifications. The size of the pictograms can be set to two different sizes, so the column width is not a fixed size. Futhermore, the pictogram grid is showed in two different views - one with the category selection available, and one where the guardian have selected a number of possible pictograms, in which case the category selection has been hidden. There are therefore two task before the number of columns can be calculated: Calculate the width of the gridview and calculate the width of the columns. The calculation can be seen in Listing 8.3.

```
int categoryWidth = 2*150;
int scrollbarWidth = 10;
if(backToNormalView)
{
    categoryWidth = 0;
}
int pictogramgridWidth = width-GComponent.DpToPixel(categoryWidth+
    buttonsWidth+scrollbarWidth,parrent.getApplicationContext());

int pictogramWidth = 200;
if(PARROTProfile.PictogramSize.MEDIUM == user.getPictogramSize())
{
    pictogramWidth = 160;
}
pictogramGrid.setColumnWidth(pictogramWidth);
int piccolnumb = pictogramgridWidth/pictogramWidth;
pictogramGrid.setNumColumns(piccolnumb);
```

Listing 8.3: The calculation of the number of columns in the pictogramview.

8.5 Kategoriværktøjet and Piktotegner Buttons

Our customers requested quick access to *Kategoriværktøjet* and *Piktotegner*, so we determined to create buttons for the applications.

The buttons are placed on the right hand side of the application, as shown in Figure 8.1. The buttons are visible only if the respective application is installed. To find out if the applications are installed, a package manager provided by android is used. We search the package manager for the applications, and set the buttons to visible if the applications are found. If an application is found, we create an intent of that application, and set the button to be visible. When the buttons are pressed, their respective applications are opened in a new activity using the intent of the application. The implementation is shown in Listing 8.4.

```java
String catName = "dk.aau.cs.giraf.pictoadmin";
String crocName = "dk.aau.cs.giraf.pictocreator";
Intent catIntent = null;
Intent crocIntent = null;
GButton catButton = (GButton) parrent.findViewById(R.id.catButton);
GButton crocButton = (GButton) parrent.findViewById(R.id.crocButton
    );


final PackageManager packMan = parrent.getPackageManager();
List<ApplicationInfo> apps = packMan.getInstalledApplications(
    PackageManager.GET_META_DATA);

for (ApplicationInfo appInfo : apps)
{
  if (appInfo.packageName.toString().equalsIgnoreCase(catName))
  {
    catIntent = packMan.getLaunchIntentForPackage(catName);

    catIntent.putExtra("currentGuardianID", guadianID);
    catIntent.putExtra("currentChildID", childID);

    if (catIntent != null)
    {
      catButton.setVisibility(this.getView().VISIBLE);
      catButton.SetImage(appInfo.loadIcon(packMan));
    }
    createOnClickListener(catButton, catIntent);
  }
  else if (appInfo.packageName.toString().equalsIgnoreCase(crocName
      ))
  {
    crocIntent = packMan.getLaunchIntentForPackage(crocName);

    crocIntent.putExtra("currentGuardianID", guadianID);
    crocIntent.putExtra("currentChildID", childID);
    if (crocIntent != null)
    {
      crocButton.setVisibility(this.getView().VISIBLE);
      crocButton.SetImage(appInfo.loadIcon(packMan));
    }
    createOnClickListener(crocButton, crocIntent);
  }
}
```

Listing 8.4: This is the implementation of creating *Kategoriværktøjet* and *Piktotegner* buttons.

8.6 Category Selection

For the users to easily compose a sentence with pictograms from different categories, we kept and enhanced the supercategory and subcategory grids. These grids makes it possible to change category with only a press on the desired category. When a category is selected, its subcategories and pictograms are loaded.

8.7 Performance Test

This section is about a performance test of the application to test how it would handle the many pictograms from the database.

8.7.1 Planning of Performance Test

In the multi-project there is a large amount of pictograms, approximately 14, 000, and the largest category contains approximately 7, 900 pictograms. In *Piktooplæser* it is possible to show the content of a category, and this means showing up to 7, 900 pictograms in the pictogram grid. To test whether *Piktooplæser* is able to handle that amount of data, we want to test what happens when that specific category is loaded and its content is shown in the application.

This should be used as a guideline when creating new categories for *Piktooplæser*.

8.7.2 Results of Performance Test

We tested *Piktooplæser* by running it following the description in Section 8.7. At the end of sprint 4, the synchronization between the remote database and the local database began working. This caused a lot of problems because it turned out that *Piktooplæser* was not geared to handle 14,000 pictograms. *Piktooplæser* reacted slow to input and scrolling made the application lag or crash. A log from Android Studio showed that the applications ran out of heap space and then crash.

The reason the heap is filled so quickly is that we use a lot of Bitmap files which take up much space. To counter this, we caught the exception and made it wait for the garbage collector to clear up more heap space. This solution prevents the application from crashing, by waiting which is very time consuming. Another solution which in the future might solve this problem is to load only the visible pictograms, thus reducing the needed amount of heap space. This means that we can load a mere 100 pictograms instead of all 14, 000 pictograms. One could then navigate through the pictograms with a ViewPager. However it is only in very rare cases that the number of pictograms in a category will exceed a few hundred pictograms in user created categories.

Futhermore, it turned out that the category gridview did not work properly when a selection of a category occurred. When you select a category, that category should be marked. But this marking happens in the adapter. This means that when a marking needs to happen, there needs to be set a new adapter, which causes the entire category grid to be refreshed. This means that if you are halfway down in the category list and select a category, the list will jump back up to the top. It was decided that this behaviour was unacceptable, and instead we added a label above the pictogram grid, which tells which category is currently selected. The pictogram grid now stays in place when a selection occur, and the label is updated to reflect this. It should be noted that this is not the optimal solution, but it was prioritized to have a working solution.

8.8 Further Tests

In the future, there were a number of test we would like to perform, which we suggest to the next group working on *Piktooplæser*. These are described in the following sections.

8.8.1 Usability Test

A usability test would be usefull, especially because we work with autistic children who do not act as regular children. This means that we have to be aware that the product has to be user friendly and very easy to use, else the autists might become frustrated and unable to use the product.

To test whether the product lives up to the requirements, a usability test would be used. The test would consists of one or more observers, a number of test subjects and a task to be performed. The observers watch the test subjects perform their task, and note their reactions and use of the product. The test subjects should be taken from the target group, else the test results would be useless. The task would be something that they would use the product for when it is released.

This test would tell us whether the product is usable for the autists, and what we have to improve. In case there are made changes due to previous tests, the usability test might have to be performed multiple times.

8.8.2 Unit Test

To be certain that the implemented code meets the requirements, we have to test it. This can be done using unit tests.

Every part of our implemented code should undergo a unit test developed to test whether it performs as required. For example, this can be done with methods by providing some input for that method. Then the return value of that function is compared to the expected result.

If this was conducted, we expect to reduce the number of logic errors to a minimum.

8.9 Problems With Large Dataset in Pictosearch

After the import of the many thousand pictograms, the application to search for pictograms, *Pictosearch*, did not work. The synchronization of the pictograms was made at a very late state in the project, and the group maintaining *Pictosearch* did not intend to fix the problems in *Pictosearch* in time of the release of the applications at the end of the sprint. Because a lot of applications (including *Piktooplæser*) was depending on *Pictosearch*, our group started fixing the problems in the last minute before the sprint end. It was discovered that the problem with *Pictosearch* was that it loaded all the pictograms into an arraylist in the beginning of the activity. This worked fine with the 5 test pictograms which were before the synchronization, but with around 14,000 pictograms the application got severe memory and performance problems.

To solve this problem, it was changed, so the list was cleared at each search and added only with the pictograms relevant for the search. Furthermore, the result of a search was limited to 100 pictograms. This prevented the application from crash at startup. The design of *Pictosearch* was also improved to use the common `GComponents`, so the application looked like the other applications.

This improvement to the *Pictosearch* application made it possible for both *Piktooplæser* and many of the other applications in the GIRAF project to work and fulfil their purpose, where they otherwise would have been useless.

8.10 Feedback From Sprint End 4

After the presentation for the fellow students and the customers, there were some feedback for the *Piktooplæser*. The fix we made for *Pictosearch* worked perfectly at the demonstration. The feedback was, that the application in general was fine and useful, but there could be a faster way to take a picture than using *Piktotegner*, which would include saving it as a permanent pictogram. The feature should be a quick way to add a temporary picture to the sentenceboard.

8.11 Evaluation of Sprint 4

In this the last sprint, the focus was mainly on getting various minor bugs and problems with the application fixed. For example the pictogram grid now has an appropriate number of columns, which is calculated based on the screen resolution of the device. The category and sub-category selector are now annotated with a title so it is clear which is which. Apart from these minor details, the main focus in sprint 4 was getting the report done.

Multi-Project - Handover 9

This chapter will describe some of the suggestions for the students, who will work on the GIRAF project next year.

9.1 General Collaboration

We strongly advise for the groups next year to remember that this is a common project. They should not only prioritize their own part of the project, but focus on improving the overall project. This means that a group should not consider it a waste of time to help another group with their problem, whether the problem is a part of their own problem or not. It does not necessarily improve the groups part of the project, but it will improve the progress of the overall project and can be documented as such. It has worked very well this semester just to knock on the door of the other groups when there is a question or a problem. The next years students should remember they are located in the same building - email correspondences, forum posts etc. are usually a lot slower than just knocking on the door. Though, if it is a big question which others could benefit from, it can be preferable to write it on the forum, so others can see the solution.

9.2 Tools

This semester, a couple of different tools have been used to improve the workflow.

9.2.1 Redmine

Redmine was used for several purposes:

- Forum
- Issue tracker
- Wiki
- File sharing

Redmine worked fine for the purpose. There are similar tools on the market, but we recommended using Redmine or a similar tool to enable the listed features. The wiki was used to share the knowledge of how to use their applications, e.g. the group making the

GComponents have their own wiki containing information about their different components, and how to use them. It is advised for the next years students to use the written wiki's to get an overview of how to use the different components, and use the wiki's as a template for their own wiki. There were some minor problems with Redmine however. For example in the forum there is an option to follow a thread, where you will get an email every time someone adds a post to the thread. But the server did not have a service to send out emails enabled, so this feature did not work.

9.2.2 Jenkins

Jenkins is used an a automatic build tool. This also includes automatic signing, so the generated APKs can be uploaded directly to Google Play.

9.2.3 Git

Git is used for code distribution. It is highly recommended to use the Git setup from this year, as all the repositories are already set up and working.

9.3 Structure of Multi-Project

The structure of the multi-project worked fine this year, so it is a possible solution for the next years students. As seen in Section 2.1 the project is structured with each group working as a small Scrum team, and once a week the Scrum Master from each group met at a Scrum-of-Scrums meeting. This meeting was lead by the Scrum-of-Scrums Master.

This year we had some initiation problems with these meetings. In the beginning of the multi-project, we had many topics to discuss, which took a long time. We advise to organize meetings dedicated to discussing these topics. It is important that all Scrum Masters attends these meetings.

Further we advice to have Scrum-of-Scrums meetings. It was a very good way to have an overview of the other groups progress, and help them if needed.

9.4 Summary

There has, as previously mentioned, been used a number of different tools to improve the workflow of the multi-project, which we advise the next years students to use alongside with something similar to our structure. This being said, there is one main piece of advice from our experiences to the next years students: Think of this as one big project. It is important not to be afraid to knock on others groups doors, and be ready to help the other group, and not to think the only important thing is improving your own part of the project. Time used on other groups is not wasted. It can be documented in the report, and from this get benefit at the exam. The goal is to improve the whole GIRAF project, and not only the projects individually.

Discussion 10

This chapter contains a section about how *Piktooplæser* has progressed throughout this project. It also contains a section about the experience gained in this project.

10.1 The Development Progress of Piktooplæser

During the work on *Piktooplæser* in this semester, the look of *Piktooplæser* has changed a lot, see Figure 10.1 for a picture of how *Piktooplæser* looked like in the beginning. One difference that is very noticeable is the implementation of *GComponents*. All the visual elements of *Piktooplæser* have been changed so they conform to the standards of *GComponents*. Futhermore TTS and the usage of simplemode through *Pictosearch* have been implemented. Some minor changes are the shortcuts to *Kategorivælger* and *Piktotegner* and the settings, search, clear and play buttons. The play button will play the sound of the pictograms in the sequence. The search button opens the *Pictosearch* application where you can search for pictograms in the database. Here you select some pictograms and confirm. The selected pictograms are returned to *Piktooplæser*, which displays the chosen pictograms in the simple mode, see Figure 8.2. In this simple mode only the clear button, and the exit simple mode button is available. See Section 8.2 for further description of simple mode.

In settings you can switch the size of the pictograms between small and large and you can show or hide the pictogram text. A scroll bar has also been added to *Piktooplæser* to give an overview of the pictograms in the pictogram grid.

Both applications can choose category and subcategory, and also choose pictograms for the sentence board.

Figure 10.1 and Figure 10.2 shows *Piktooplæser* before and after our work.

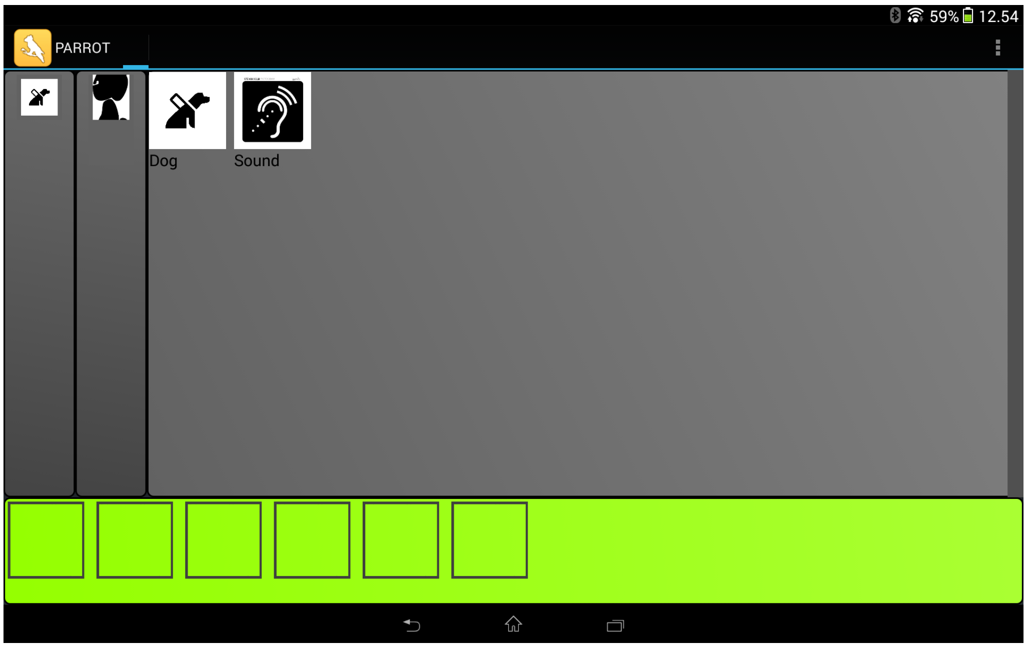

Figure 10.1: Design of *Piktooplæser* before our work.

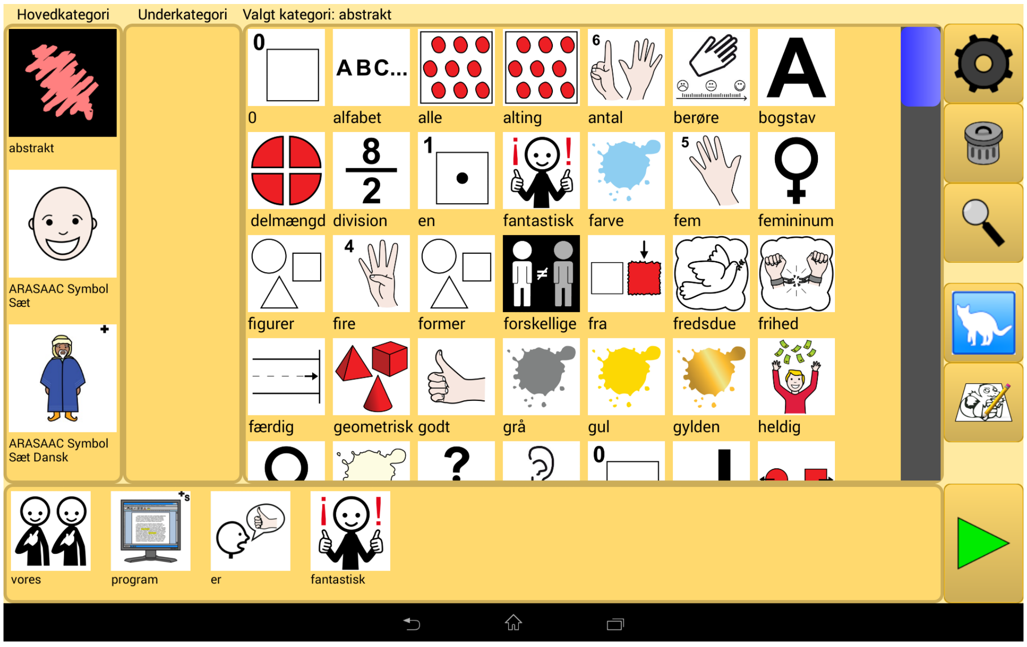

Figure 10.2: Design of *Piktooplæser*.

10.2 Experience Gained

This project is the first time we have worked in a multi-project. One of the advantages of working in such a large project is that each group contributes more or less to the multi-project. This makes it possible to have a large project, and not have to take care of everything. For example in this multi-project there were two groups dedicated for the development of the databases: One group for the remote database, and one group for the local database. This meant that all the other groups did not have to think about anything database related other than what they need to insert and retrieve from the database. Another example is that one group worked exclusively with GUI development, which first and foremost ensured that all applications in the multi-project had a similar look. On the other hand, having a group dedicated for one part of the project can also be a disadvantage if for example a lazy group is assigned a vital part of the project. When multiple projects depend on one sub-module which is developed by a lazy group, the other groups can risk sitting for weeks waiting for something to be fixed.

We found that when there are so many groups working on the same project, it is possible to collaborate with another group on some programming tasks. For example we developed a Text-To-Speech module with the *Piktotegner* group which also could be used by another application.

10.3 Future Work

The tasks which we would like to have implemented in *Piktooplæser* are listed below. We did not implement the listed tasks because some modules were not yet ready to be implemented or we prioritized other tasks as more important.

10.3.1 Future Work for Piktooplæser

- Save settings to profiles.
- Special view-mode when playing sentence.
- Play button should change into a stop button when sentence is playing.
- There should be an option to save and use template sentences.
- Improve scaling of pictograms on different screen resolutions.
- Quick add of temporary picture through the camera.
- When dragging a pictogram to the sentenceboard, the pictograms in the sentence-board should move to indicate what would happen if the pictogram was dropped in that position.

10.3.2 Future work for Pictogram-Lib

- Sound for categories should be saved automatically if there is no sound already, just like it is done for pictograms.

10.3.3 Further Explanation of Future Work for Piktooplæser

The settings should be saved to the profiles. It is the *GIRAF Launcher* which handles settings for profiles. For the *GIRAF Launcher* to handle settings it is required that the settings reside in a special settings activity.

The special view-mode when playing a sentence should be so that when you press play, the sentence is moved up to the center of the screen, and everything else is blacked out or hidden. In short it should be so that only the sentence is visible on the screen. Perhaps with highlighting of the currently playing pictogram. This feature was in development by *Sekvenser* but not ready for use.

The customers asked for the option to save templates. This could be a template like "I would like". This is useful when e.g. eating breakfast, a citizen can quickly construct the sentence "I would like oatmeal".

The customers also asked for a feature, where each pictogram in the sentence board would move, to show what would happen, if you release the hovering pictogram dragged from the pictogram grid.

Conclusion 11

The highest prioritized requirement from the customers was working code. When we took over the *Oasis Library*, the library did not match the new database schema. To make *Oasis Library* work with the database schema, the models of the library was updated. In collaboration with the *LocalDB* group, the schema and URL's were updated, so they match both libraries. All in all, when the first sprint ended, *Oasis Library* was functional with the new database schema, and the other applications in the GIRAF multi-project could use it to communicate with the database.

Piktooplæser (previously called *Parrot*) was not communicating with the database at all, and used local .xml files for storage. Most of *Piktooplæser* has been rewritten in order to communicate with the database, and to give *Piktooplæser* the same look and feel as the other applications in the GIRAF multi-project. The overall idea of design of *Piktooplæser* has been kept, see Section 10.1. The application still has a separate grid for categories, one for sub-categories and a grid for the sentence. Per request from the customers, it is now possible to choose a set of pictograms which can be selected from. In this mode the category selector is hidden, so that only the available pictograms are visible on the screen.

Piktooplæser also has Text-To-Speech (TTS) capabilities to read pictograms aloud, even though no sound has been recorded for the pictogram. The TTS sound is then stored in the database for future use.

By this we conclude that the most important requirements and suggestions from the customers has been implemented, and both *Oasis Library* and *Piktooplæser* work together with the database in order to provide a uniform experience across all GIRAF applications.

Bibliography

Android Developers, **a**. Android Developers. *Content Providers.* `http://developer.android.com/guide/topics/providers/content-providers.html`[Last seen: 2014/05/23].

Android Developers, **b**. Android Developers. *Storage Options.* `http://developer.android.com/guide/topics/data/data-storage.html#db`[Last seen: 2014/03/28].

Android Developers, **c**. Android Developers. *Android NDK.* `https://developer.android.com/tools/sdk/ndk/index.html`[Last seen: 2014/05/23].

Android Developers, **d**. Android Developers. *Toasts.* `http://developer.android.com/guide/topics/ui/notifiers/toasts.html`[Last seen: 2014/05/14].

Belanger. Pierre-David Belanger. *JeSpeak.* `http://jespeak.sourceforge.net/`[Last seen: 2014/05/06].

Bender et al., **May 2014**. Anders Bender, Benjamin Hubert, Dennis Bækgaard Nielsen and Michael Lausdahl Fuglsang. *Developing Software for Autism Instituions*, Aalborg University, May 2014.

Digital Equipment Corporation. Digital Equipment Corporation. *Information Technology - Database Language SQL.* `http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt`[Last seen: 2014/05/23].

Duddington. Jonathan Duddington. *eSpeak text to speech.* `http://espeak.sourceforge.net/`[Last seen: 2014/05/06].

Flindt et al., **June 2013**. Barbara Flindt, Hilmar Laksá Magnussen, Jeppe Blicher Tarp and Simon Jensen. *WASTELAND - A GIRAF Database*, Aalborg University, June 2013.

Git. Git. *distributed-even-if-your-workflow-isn.* `http://git-scm.com/`[Last seen: 2014/02/14].

Google, **2014**. Google, 2014. *Getting Started with Android Studio.* `http://developer.android.com/sdk/installing/studio.html`[Last seen: 2014/02/14].

Rubin, **2012**. Kenneth S. Rubin. *Essential Scrum.* Addison Wesley, 2012.

SQLite, **a**. SQLite. *About SQLite.* http://www.sqlite.org/about.html[Last seen: 2014/05/23].

SQLite, **b**. SQLite. *SQL Features That SQLite Does Not Implement.* https://sqlite.org/omitted.html[Last seen: 2014/05/23].

The Eclipse Foundation, **2014**. The Eclipse Foundation, 2014. *The Eclipse Foundation open source community website.* https://www.eclipse.org/[Last seen: 2014/02/14].

Kooij. Martijn van der Kooij. *Picto-Selector.* http://pictoselector.eu/[Last seen: 2016/05/26].

Vogel, **2013**. Lars Vogel, 2013. *Android SQLite database and content provider - Tutorial.* http://www.vogella.com/tutorials/AndroidSQLite/article.html.

Voice RSS. Voice RSS. *Voice RSS.* http://voicerss.org/[Last seen: 2014/05/06].

Appendix

Database Class Model A

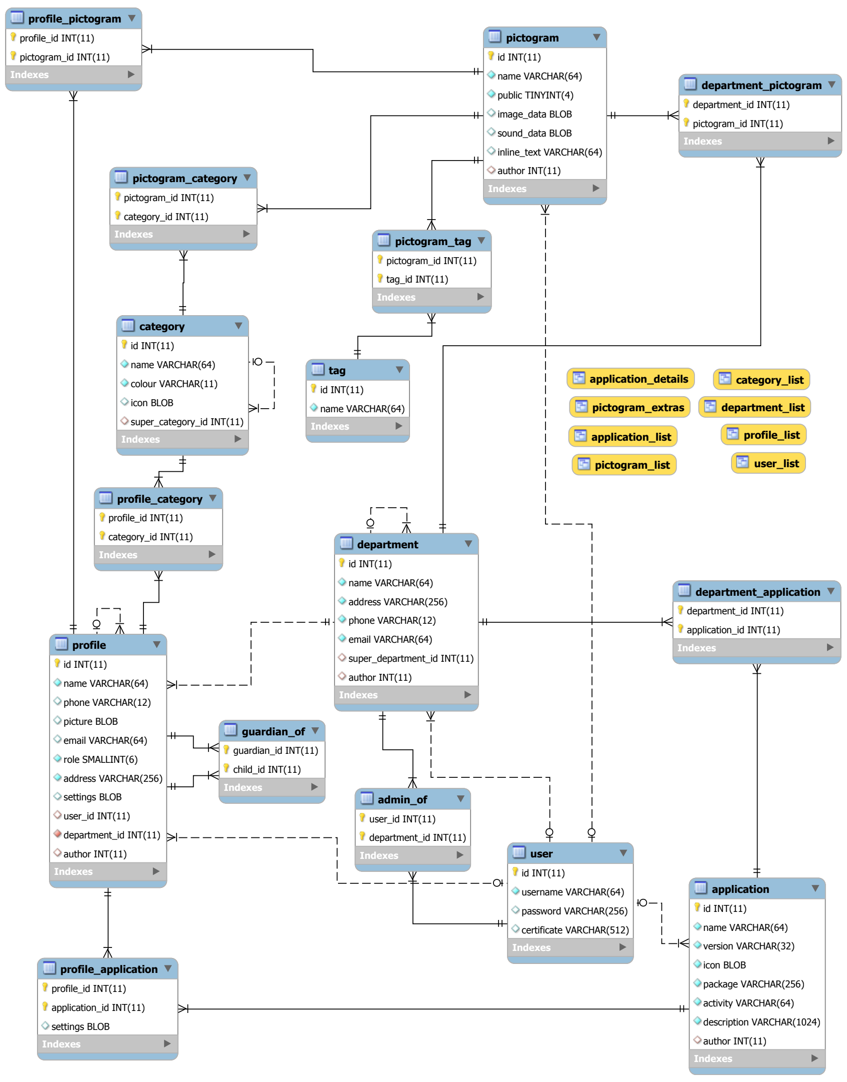

Figure A.1: Class diagram of the local database.

Project CD B

The CD found on this page contains the following:

- The source code for *Piktooplæser*, *Oasis Library* and *Oasis App* at the time of handover.
- A compiled version of *Piktooplæser* and *Oasis App*.
- A digital version of the report in PDF format.