AALBORG UNIVERSITY

STUDENTS STUDYREPORT

GROUP SW604F12

BACHELOR PROJECT

Oasis Administration for GIRAF

Authors:
Henrik KLARUP
Jens Mohr MORTENSEN
Dan Stenholt MØLLER

Supervisor:
Ulrik NYMAN

May 31, 2012

Department of Computer Science
Aalborg University
Selma Lagerlöfs Vej 300
DK-9220 Aalborg Øst
Telephone +45 9940 9940
Telefax +45 9940 9798
http://cs.aau.dk

Title: Oasis Administration for GIRAF
Subject: Application Development
Semester: Spring Semester 2012
Project group: sw604f12

Participants:

Henrik Klarup

Jens Mohr Mortensen

Dan Stenholt Møller

Supervisor:
Ulrik Nyman

Number of copies: X

Number of pages: X

Number of appendices: X Pages

Completed: X

Synopsis:

This project is about the development of... This project is about the development of...

Preface

This project has been produced in the spring of 2012 in the sixth semester of the software engineering study at Aalborg University.

Contents

*

Part I

Common Introduction

CHAPTER 1

Introduction

In order to describe the context of the system, we – as a multi project group – will in the following state the motivation of the project, the group of people we are aiming at helping, the technological platform chosen, the used development method, followed by a problem definition, a system description and architecture, and the conducted usability test.

1.1 Motivation

As this is a student report written as part of a learning project, we are required to comply with the study regulation. The main areas of focus, according to the study regulation, are: multi-project management and quality assurance in the form of requirements analysis, requirements management, and testing. The goal is to create a comprehensive software system, across multiple project groups, in order to enhance our competences in analysis, design, implementation, and evaluation of software applications in regards to the system requirements[**?**].

This project builds on top of a previous project, and is further developed, with the aim of having other students continue the development. The goal of the project, we are building on top of, is to create a touch based tablet system to support children and their guardians in everyday scenarios.

1.2 Target Group

Our target group is both children and their guardians. These guardians have certain needs for special tools and gadgets that help to ease the communication between them and the children.

Five teachers and educators, who work with children, act as customers. They will provide requirements and information about the institutions' way of working to give us an insight into their daily struggles.

1.2.1 Working with Children with ASD

This section is based upon the statements of a woman with ASD [**?**], explaining what it is like to live with ASD, and an interview with an educator at Birken, a special kindergarten for children (see appendix **??** for interview notes).

People with ASD are often more visual in their way of thinking. Rather than visualizing thoughts in language and text, they do it in pictures or visual demonstrations. Pictures and symbols are therefore an essential part of the daily tools used by children and the people interacting with them. Also, children can have difficulties expressing themselves by writing or talking, and can often more easily use electronic devices to either type a sentence or show pictures, to communicate with people around them. Another characteristic of children is their perception of time. Some of them simply do not understand phrases like "in a moment" or "soon", they will need some kind of visual indicator that shows how long time they will have to wait.

Different communication tools for children with autism already exist, but many of them rely on a static database of pictures, and often these has to be printed on paper in order to use them as intended. Other tools, such as hourglasses of different sizes and colors, are also essential when working with children, and these tools are either brought around with the child, or a set is kept every place the child might go, e.g. being at an institution or at home.

There exists tools today which helps the guardians in their daily life, although – as stated in Drazenko's quote – none of them are cost-effective enough to be used throughout the institutions. From the quote, it is clear that there is a need for a more cost-effective solution.

> The price of the existing solutions are not sufficiently low such that we can afford to buy and use them throughout the institution.
>
> - Drazenko Banjak, educator at Egebakken.

1.3 Target Platform

Since we build upon last year's project, we are bound to use the platform they used, which is tablets running the Android operating system.

In this project we have been provided with five Samsung Galaxy Tab 10.1 devices[**?**]. The firmware on the tablets is version 3.2. This version, as of project start, is the latest stable version available for these specific tablets. [**?**]

1.4 Development Method

As a part of the study regulation we have been required to use the same development method in each individual group. Two methods have been considered, *XP* (eXtreme Programming) [**?**], and Scrum [**?**].

With the knowledge of both XP and Scrum, we decided in the multi project to use Scrum of Scrums, which is the use of Scrum nested in a larger Scrum project [**?**].

The reason for choosing Scrum of Scrums is that everyone, at all times, will be able to know what the vision of the project is, and how close every group is to achieving their individual goals of the vision.

Another element of the Scrum method is that a close contact with the customers is maintained. This helps keep the product backlog up to date and correctly prioritized. The customers are presented with the vision of the project, as well as showing the latest release when we have meetings with our customers.

We customized Scrum to fit our project. The changes are as follows:

- The sprint length have been shortened to approximately 7 - 14 half days.
- Some degree of pair programming have been introduced.
- There is no project owner because this is a learning project.
- Everyone is attending the Scrum of Scrums meetings.
- The Scrum of Scrums meetings are only held once at sprint planning.

1.5 Problem Definition

The problem statement is as follows:

> How can we ease the daily life for children with ASD and their guardians, while complying with the study regulation?

This problem statement is necessarily vague to allow the individual groups some freedom in their projects, while we maintain the overall structure of the multi project, however there are limiting factors. We are limited by resources and time available, as we are only working on this project for a single semester. However, all work done in this multi project will be passed on to the next line of students, which means we can make a full system design and pass on anything we do not have the time or resources for. This also requires that our work need to be of such quality that it is understandable by students of the same educational level as ourselves.

1.6 System Description

GIRAF is a collection of applications, either fully or partially interdependent, for the Android platform, designed to be used by guardians and children. GIRAF consists of five projects with various degree of interaction. These projects are named Launcher, PARROT, WOMBAT, Oasis, and Savannah. Each of the groups have produced individual products, which are parts of a greater project, GIRAF.

Launcher handles execution of GIRAF apps, and at the same time it provides safety features to ensure that a user that is not authorized to interact with the rest of the system will not be able to do so. When the launcher executes an app, it will provide it with profile information, specifying which child is currently using the app, as well as which guardian is signed in.

PARROT is an app which provides access to pictograms – pictures with associated information such as sound and text – which can be used for communication. PARROT also gives guardians functionality for adding additional pictograms, as well as organizing the pictograms into categories for ease of access, based on the needs of the individual child.

WOMBAT is an app which purpose is to help the children to understand the aspect of time, by visualizing it. WOMBAT provides different ways of displaying time, as well as the possibility to configure the app for the needs of individual children.

Oasis locally stores the data and configuration of the GIRAF platform, and provides an API to access it. The stored data and configurations are synchronized to the Savannah server, if available. In addition, an app is provided for the guardian to access the stored data and configurations.

Savannah provides Oasis with a way to synchronize tablets running GIRAF. Furthermore, a website is provided to ease administration of the synchronized data.

1.7 Architecture

Our System architecture – shown in Figure 1.1 has been designed with simplicity in mind and was greatly inspired by the MVC pattern. This means that the architecture is divided into three layers. The lowest layer is the database where the information is stored. Above this layer is the controller layer which, in the GIRAF platform, is known as Oasis. The controller is responsible for querying the database for information needed in an app and the controller is also responsible for storing information in the database. The last layer is the apps. This division of layers give the GIRAF platform a low cohesion which makes it easier to work with individual parts of the platform independently.

We have chosen to redesign last year's architecture [**?**] to make it easier to work with. We have simplified the architecture because we feel it is unnecessarily complex.

1.8 Usability Test

As stated in the motivation, quality assurance through testing of the system is required. Therefore a usability test was conducted in order to measure the current usability of the GIRAF platform as a whole, as well as of the individual parts of the platform. Furthermore, the next wave of developers will immediately be able to start correcting the found usability issues.

1.8.1 Approach

The test group consists of the five contact persons. We assess that they, as a test group, are representative. We base this on them being a mix of educators and teachers, with varying computer skills.

They have prior knowledge about the overall idea of the GIRAF platform, and although some of the contact persons had previously informally used some aspects or parts of the system, they had not been exposed to the platform as a whole, and therefore still are of value.

The invitation sent to the test persons can be found in **??**.

The Instant Data Analysis (IDA) method for usability is chosen. A traditional video analysis method could be used, but since IDA is designed for small test groups, this approach is used. [**?**]

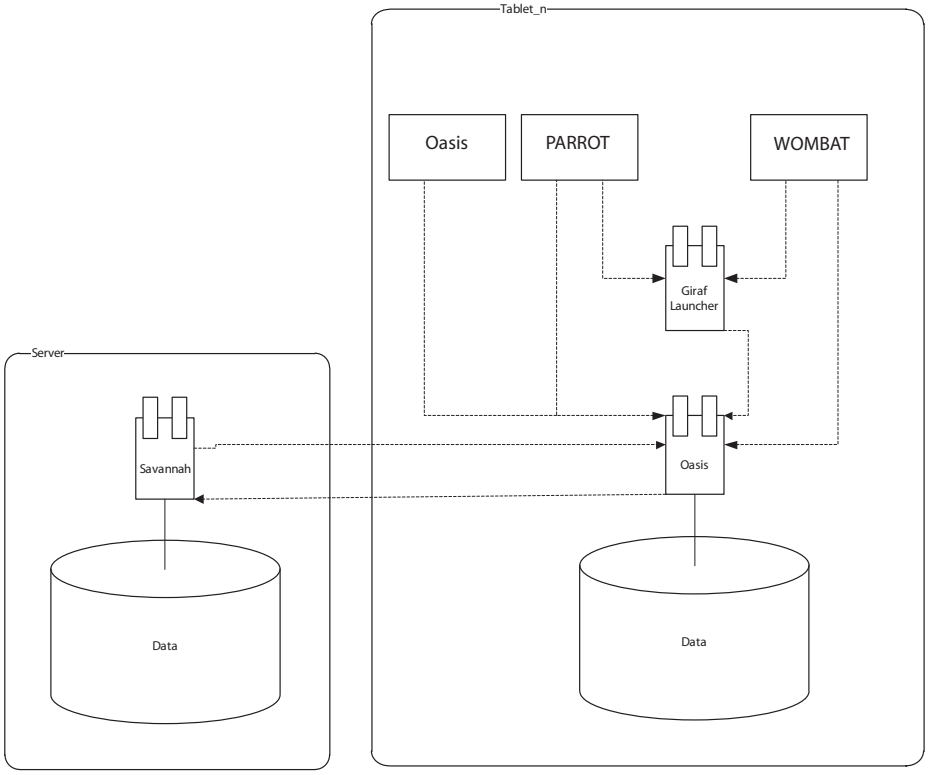

Figure 1.1: The GIRAF architecture

Setup

The usability test is divided into two tests: A test of three user applications, and a test of two administrative applications. The user applications are: The launcher, PARROT, and WOMBAT. The administrative applications are: The Oasis app and the Savannah web application. Each test is assigned a team to accommodate the need to run two tests simultaneously. The teams are made with respect to the criteria of the Instant Data Analysis process.
Each team consisted of:

- 1 x Test Coordinator
- 1 x Test Monitor
- 1 x Data Logger
- 2 x Observers

The usability lab at Aalborg University is designed with two rooms for usability testing and a control room to observe and record the tests. The two test chambers are assigned a test each and the control room is used to observe both tests as seen in figure 1.2.

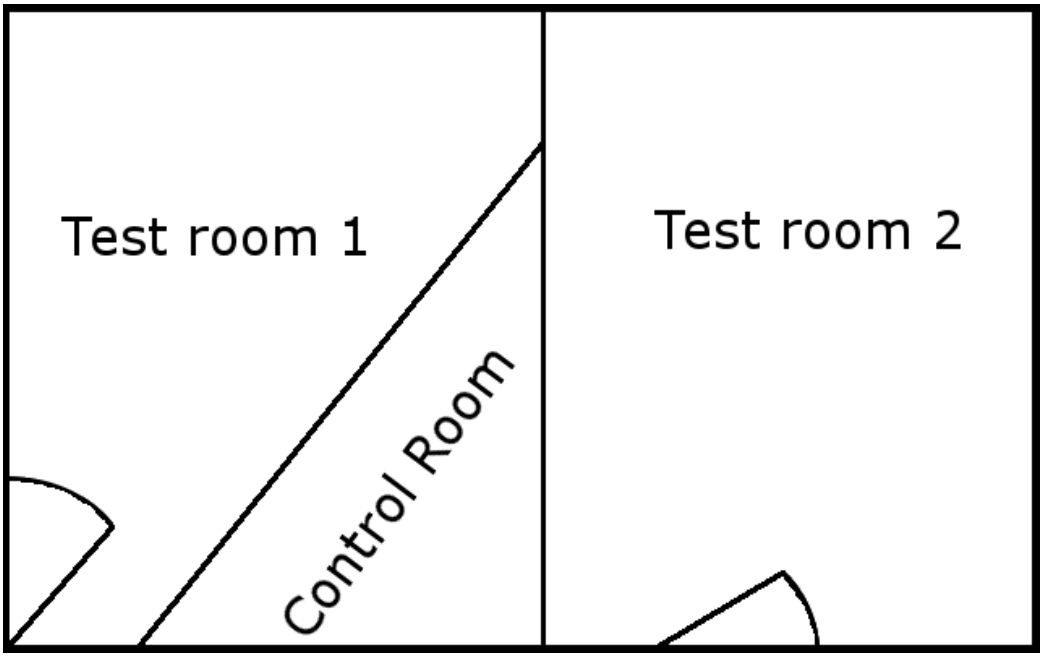

Figure 1.2: An overview of the usability lab at Cassiopeia, Department of Computer Science, Aalborg University.

Execution

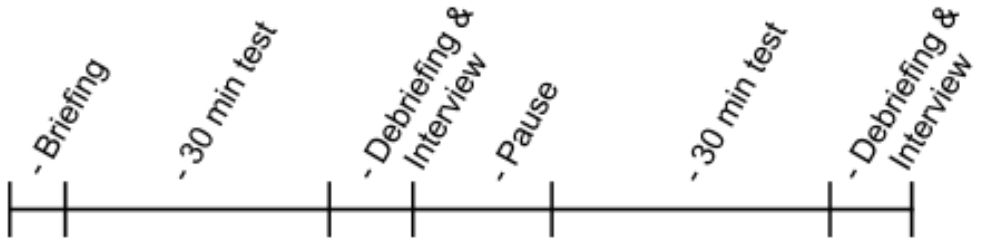

Figure 1.3: The schedule of the usability test.

The tests are conducted according to the schedule in Figure 1.3.

Briefing, debriefing, and questionnaire documents can be found in **??**, and the results of the test can be found in subsection 6.4.1.

Part II

Introduction

CHAPTER 2

Analysis

As a part of the multi project, we are not directly solving the problem ourselves, but providing a part such that the other project groups can perform that task easier. As we did not solve the problem directly we have made our own problem definition: *How can we provide a set of tools which can help develop application for the GIRAF-system?* As a way to solve this we have chosen to make 3 projects, a library providing methods and classes, a database to save information and an application to control the content of the database.

2.1 Requirements

When we where to develop our library we asked the other groups to supply requirements. We received the following requirements: Save data on the device Various classes for: Profiles Media Apps Departments From this we derived some features which will be shown in appendix FeatureList.

2.2 System architecture

2.2.1 In the multi project

The way Oasis fit into the multi project, is by being a middle layer between the Apps and the server, as seen in figure **??** on page ??. Oasis will handle the communication from the apps to save in the local database, as well as synchronizing the local database, with the server.

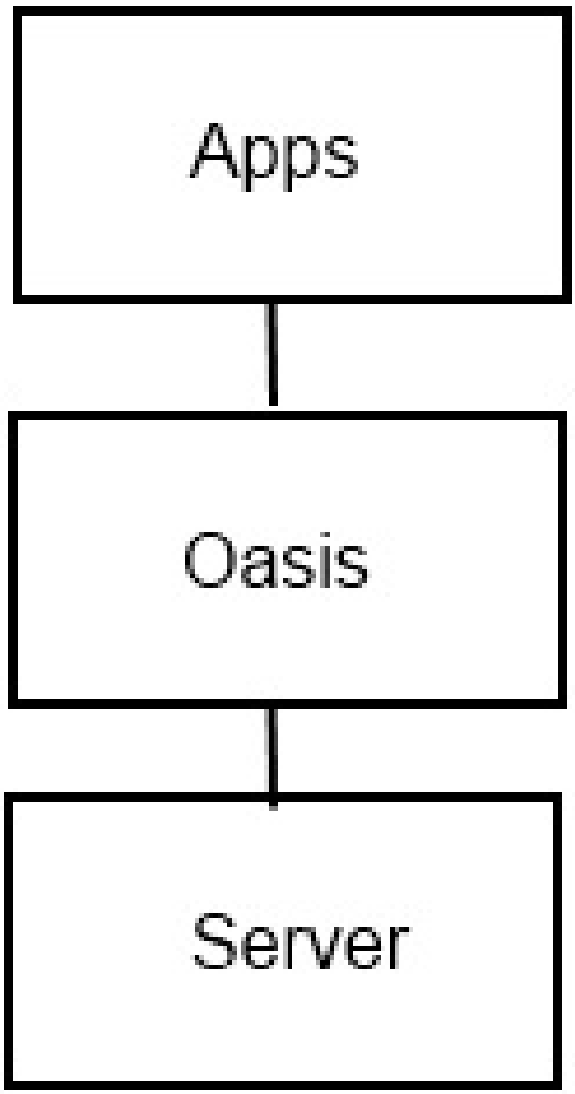

Figure 2.1: The multi project architecture

2.2.2 Each part

Oasis consists of 3 parts, the oasis library which is the core of the project, the administration app, and the local database, as shown in figure **??** on page ??. The Oasis library will handle applications interaction with the local database, and every giraf app should be utilizing this library. The Oasis library will also make sure that the local database is synchronized with the server's database. The administration application is an app from which the user can interact with the database directly, creating or deleting users and departments, and making sure these are connected.

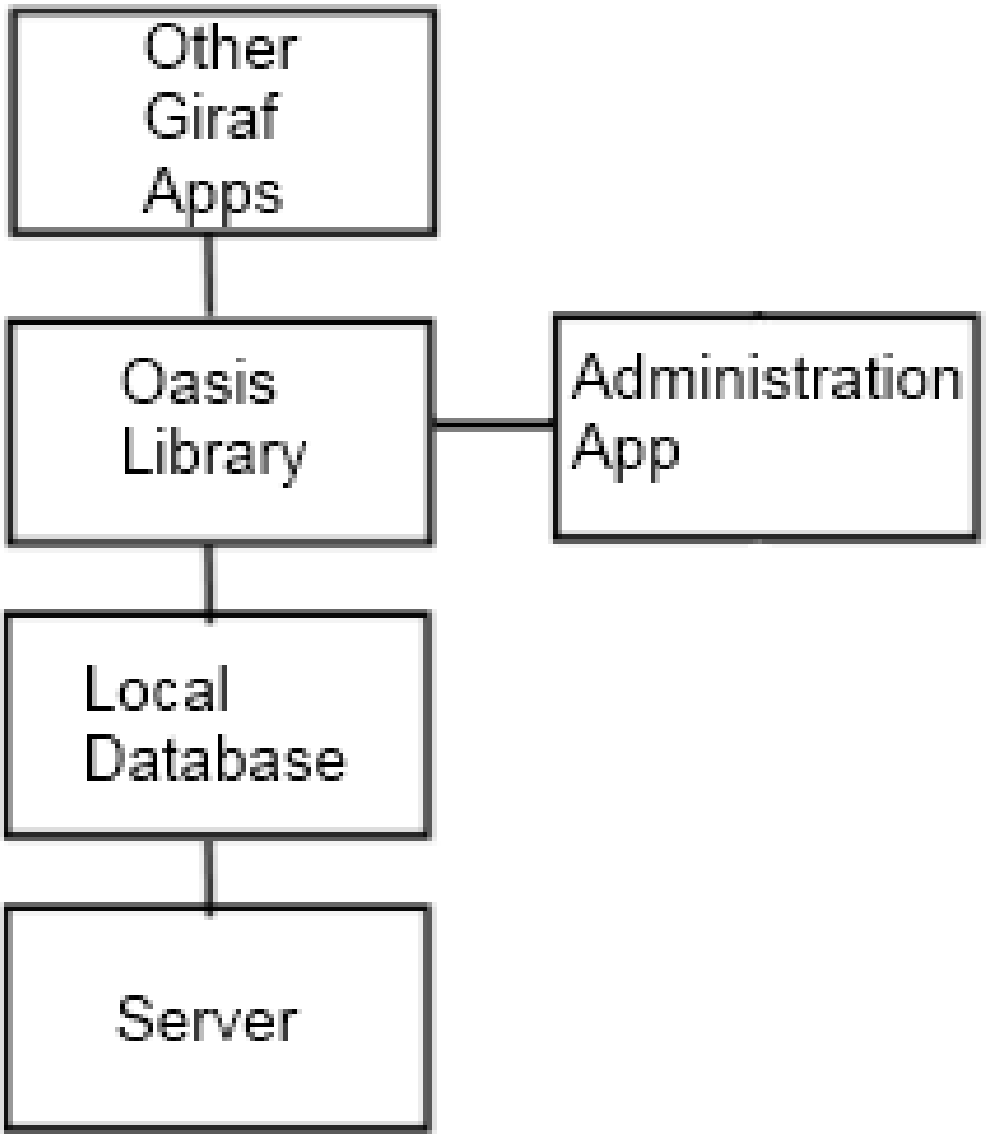

Figure 2.2: The project part architecture

CHAPTER 3

Sprint Process

In this chapter we describe the sprint process of our project. In the project period we underwent HVOR MANGE?? KAN KUN FINDE 4 BURNDOWNS? sprints. The length of the sprints was decided at every sprint meeting, but they were usually no longer than 10 half days long. At the end of every sprint period, we held a meeting to discuss what we learned through the sprint, and if all the tasks was finished. In the following a description of the different sprints is presented. The Burndown charts and sprint backlogs can be seen in the Appendix.

3.1 Sprints

3.1.1 Sprint 1

Sprint 1 started on date ?? and lasted 7 half days. In this sprint period we started making the local database, the model objects, that is supposed to wrap the data from the database, and some methods to the library, we got as requirements from the other groups.

what we learned

In this sprint we learned that we should be a bit better at estimating the time of each task and that we needed to break the task into bigger tasks, so each person would not need to take a new task every minute (bedre forklaring!.

3.1.2 Sprint 2

Sprint 2 started on date ?? and lasted 8 half days. In this sprint period we updated the local database and the library with the new requirements we got from the other groups. Besides that we used a little time on setting a counter-strike server up, so we had something to do socially between the groups.

what we learned

In this sprint we learned that we still should be a bit better at estimating the time of each task, because there was some tasks we did not even start on.

3.1.3 Sprint 3

Sprint 3 started on date ?? and lasted ?? half days. In this sprint period we continued on updating the local database and library with the new requirements gathered from the other groups. Besides that we started thinking on our "'BMI App"', which is not necessary a BMI application, but it is an application, which shows the functionality of the library and local database.

what we learned

In this sprint we learned that we should be a bit better at not taking new tasks in to the sprint backlog when the sprint has started, because this will prevent us from making any progress. Besides that we overestimated the size of the tasks, which lead to tasks not being completed before the end of the sprint, because they are connected to each other.

3.1.4 Sprint 4

Sprint 4 started on date ?? and lasted ?? half days. In this sprint period we continued on updating the local database and library with the new requirements gathered from the other groups. Besides that we started writing javadoc to the library, re factored the code in the local database, started making a method, which created dummy data, and wrote a section of the common report (Target Platform and Developing method).

what we learned

In this sprint we learned that we still needs to be better at saying no to other groups, which come with new tasks in the middle of the sprint period.

Tail

Part III

Development

CHAPTER 4

Oasis Local Db

In this chapter we describe the local database, called Oasis Local Db. Oasis Local Db is used for saving the data, which is used by the GIRAF applications. First the structure of Oasis Local Db is described in section 4.1. After that the implementation of Oasis Local Db on the android system is described in section 4.2 on the next page.

4.1 Structure

The structure of Oasis Local Db is developed in cooperation with the savannah server group. The reason for that is to alleviate the complexities that could occur during a synchronization of the tablet and the savannah server. The database schema can be seen in figure INSERT FIGURE REF. There are a few differences between our database schema and the savannah server's. The reason for that is that there only is SQLite, which is embedded into the Android system. SQLite is an open source database. It supports three kinds of data types; TEXT, which is similar to String type in Java, INTEGER, which is similar to long type in java, and REAL, which is similar to double type in java. SQLite does not validate if the types written to the columns actually are of the defined type. This means that it for instance is possible to write an integer into a TEXT column. On the positive site SQLite only requires a little portion of memory at runtime (250 KByte).

When making a SQLite Database on an Android device it is only needed to define the SQL statements for creating and updating the database. After that

the Android system will manage it. An example of how we implemented a table in the database can be seen in section **??** on page ??.

4.2 Implementation

As said above, we implemented the local database using SQLite. First we made metadata files for each table in the database. The one for the AuthUsers table can be seen in listing 4.1

```
1  package dk.aau.cs.giraf.oasis.localdb;
2
3  import android.net.Uri;
4  import android.provider.BaseColumns;
5
6  public class AuthUsersMetaData {
7
8      public static final Uri CONTENT_URI =
           Uri.parse("content://dk.aau.cs.giraf.oasis.localdb.AutismProvider/authusers");
9
10     public static final String CONTENT_TYPE_AUTHUSERS_LIST =
           "vnd.android.cursor.dir/vnd.dk.authusers";
11     public static final String CONTENT_TYPE_AUTHUSER_ONE =
           "vnd.android.cursor.item/vnd.dk.authusers";
12
13     public class Table implements BaseColumns {
14        public static final String TABLE_NAME = "tbl_authusers";
15
16        public static final String COLUMN_ID = "_id";
17        public static final String COLUMN_CERTIFICATE = "authusers_certificate";
18        public static final String COLUMN_ROLE = "authusers_role";
19     }
20 }
```

Listing 4.1: The AuthUsers MetaData

The uri, CONTENT_URI and the two strings, CONTENT_TYPE_AUTHUSERS_LIST and CONTENT_TYPE_AUTHUSER_ONE, is not used by the SQLite Database and will be explained later in the content provider section INSERT FIGURE REF. The inner class Table defines the Strings, which are used as names of the table and the columns.

When the metadata file is created we make a table class file, which defines the SQL statements for creating, updating and deleting the table. The table class file for AuthUsers can be seen in listing 4.2.

```
1  package dk.aau.cs.giraf.oasis.localdb;
2
3  import android.database.sqlite.SQLiteDatabase;
4
5  public class AuthUsersTable {
6
7      private static final String TABLE_CREATE = "CREATE TABLE "
8             + AuthUsersMetaData.Table.TABLE_NAME
9             + "("
10            + AuthUsersMetaData.Table.COLUMN_ID + " INTEGER NOT NULL, "
11            + AuthUsersMetaData.Table.COLUMN_CERTIFICATE + " TEXT NOT NULL, "
12            + AuthUsersMetaData.Table.COLUMN_ROLE + " INTEGER NOT NULL, "
13            + "PRIMARY KEY (" + AuthUsersMetaData.Table.COLUMN_ID + ", " +
                  AuthUsersMetaData.Table.COLUMN_ID + ")"
14            + ");";
15
```

```
16      private static final String TABLE_DROP= "DROP TABLE IF EXISTS " +
            AuthUsersMetaData.Table.TABLE_NAME + ";";
17
18      /**
19       * Executes sql string for creating certificate table
20       * @param db this is an instance of a sqlite database
21       */
22      public static void onCreate(SQLiteDatabase db) {
23         db.execSQL(TABLE_CREATE);
24      }
25
26      /**
27       * executes a sql string which drops the old table and then the method calls oncreate, which
                create a new certificate table
28       * @param db this is a instance of a sqlite database
29       * @param oldVersion integer referring to the old version number
30       * @param newVersion integer referring to the new version number
31       */
32      public static void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
33         db.execSQL(TABLE_DROP);
34         onCreate(db);
35      }
36  }
```

Listing 4.2: The AuthUsers Table

Now is the Oasis Local Db created, but to be able to access the Oasis Local Db from other applications, it needs to implement a content provider. A content provider allows other applications to fetch data from the application, which implement the content provider. The access to a content provider is done by using a URI, which is defined in the AndroidManifest file. The content provider must implement several methods. These methods are:

- onCreate() - called at startup to initialize the content provider.
- getType() - called when an application needs to know the type of the data. This is not used in Oasis Local Db.
- query() - called when an application wants to query in the Oasis Local Db.
- insert() - called when an application wants to insert data into the Oasis Local Db.
- update() - called when an application wants to update existing data in the Oasis Local Db.
- delete() - called when an application wants to delete existing data in the Oasis Local Db.

An example of how we have implemented our content provider, with focus on the authausers table, can be seen in listing 4.3

```
1  package dk.aau.cs.giraf.oasis.localdb;
2  .
3  .
```

```
4  .
5  public class DbProvider extends ContentProvider {
6
7     private DbHelper dbHelper;
8     private static final UriMatcher sUriMatcher;
9     .
10    .
11    .
12    private static final int AUTHUSERS_TYPE_LIST = 3;
13    private static final int AUTHUSERS_TYPE_ONE = 4;
14    .
15    .
16    .
17    static {
18       sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
19       .
20       .
21       .
22       sUriMatcher.addURI(DbHelper.AUTHORITY, "authusers", AUTHUSERS_TYPE_LIST);
23       sUriMatcher.addURI(DbHelper.AUTHORITY, "authusers/#", AUTHUSERS_TYPE_ONE);
24       .
25       .
26       .
27    }
28    .
29    .
30    .
31    private static final HashMap<String, String> authusersProjectionMap;
32    static {
33       authusersProjectionMap = new HashMap<String, String>();
34       authusersProjectionMap.put(AuthUsersMetaData.Table.COLUMN_ID,
               AuthUsersMetaData.Table.COLUMN_ID);
35       authusersProjectionMap.put(AuthUsersMetaData.Table.COLUMN_CERTIFICATE,
               AuthUsersMetaData.Table.COLUMN_CERTIFICATE);
36       authusersProjectionMap.put(AuthUsersMetaData.Table.COLUMN_ROLE,
               AuthUsersMetaData.Table.COLUMN_ROLE);
37    }
38    .
39    .
40    .
41    @Override
42    public boolean onCreate() {
43       dbHelper = new DbHelper(getContext());
44       return false;
45    }
46
47    @Override
48    public int delete(Uri uri, String where, String[] whereArgs) {
49       SQLiteDatabase db = dbHelper.getWritableDatabase();
50       int rowsDeleted = 0;
51       String rowId;
52       .
53       .
54       .
55       case AUTHUSERS_TYPE_LIST:
56          rowsDeleted = db.delete(AuthUsersMetaData.Table.TABLE_NAME, where, whereArgs);
57          break;
58       case AUTHUSERS_TYPE_ONE:
59          rowId = uri.getPathSegments().get(1);
60          rowsDeleted = db.delete(AuthUsersMetaData.Table.TABLE_NAME,
61                AuthUsersMetaData.Table.COLUMN_ID + " = " + rowId + (!TextUtils.isEmpty(where) ?
                      " AND (" + where + ")" : ""),
62                whereArgs);
63          break;
64       .
65       .
66       .
67       default:
68          throw new IllegalArgumentException("Unknown URI: " + uri);
69       }
70
71       getContext().getContentResolver().notifyChange(uri, null);
72       return rowsDeleted;
73    }
74
75    @Override
76    public String getType(Uri uri) {
77       switch(sUriMatcher.match(uri)) {
78       .
79       .
80       .
81       case AUTHUSERS_TYPE_LIST:
82          return AuthUsersMetaData.CONTENT_TYPE_AUTHUSERS_LIST;
```

```
83          case AUTHUSERS_TYPE_ONE:
84             return AuthUsersMetaData.CONTENT_TYPE_AUTHUSER_ONE;
85          .
86          .
87          .
88          default:
89             throw new IllegalArgumentException("Unknown URI: " + uri);
90          }
91       }
92       @Override
93       public Uri insert(Uri uri, ContentValues values) {
94          SQLiteDatabase db = dbHelper.getWritableDatabase();
95          long rowId;
96          Uri _uri;
97
98          switch(sUriMatcher.match(uri)) {
99          .
100         .
101         .
102         case AUTHUSERS_TYPE_LIST:
103            try {
104               rowId = db.insertOrThrow(AuthUsersMetaData.Table.TABLE_NAME, null, values);
105               _uri = ContentUris.withAppendedId(AuthUsersMetaData.CONTENT_URI, rowId);
106               getContext().getContentResolver().notifyChange(_uri, null);
107            } catch (SQLiteConstraintException e) {
108               _uri = ContentUris.withAppendedId(AuthUsersMetaData.CONTENT_URI, -1);
109            }
110            return _uri;
111         .
112         .
113         .
114         default:
115            throw new IllegalArgumentException("Unknown URI: " + uri);
116         }
117      }
118
119      @Override
120      public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs,
             String sortOrder) {
121         SQLiteQueryBuilder builder = new SQLiteQueryBuilder();
122
123         switch(sUriMatcher.match(uri)) {
124         .
125         .
126         .
127         case AUTHUSERS_TYPE_LIST:
128            builder.setTables(AuthUsersMetaData.Table.TABLE_NAME);
129            builder.setProjectionMap(authusersProjectionMap);
130            break;
131         case AUTHUSERS_TYPE_ONE:
132            builder.setTables(AuthUsersMetaData.Table.TABLE_NAME);
133            builder.setProjectionMap(authusersProjectionMap);
134            builder.appendWhere(AuthUsersMetaData.Table.COLUMN_ID + " = " +
                   uri.getPathSegments().get(1));
135            break;
136         .
137         .
138         .
139         default:
140            throw new IllegalArgumentException("Unknown URI: " + uri);
141         }
142
143         SQLiteDatabase db = dbHelper.getReadableDatabase();
144         Cursor queryCursor = builder.query(db, projection, selection, selectionArgs, null, null,
                null);
145         queryCursor.setNotificationUri(getContext().getContentResolver(), uri);
146         return queryCursor;
147      }
148      @Override
149      public int update(Uri uri, ContentValues values, String where, String[] whereArgs) {
150         SQLiteDatabase db = dbHelper.getWritableDatabase();
151         int rowsUpdated = 0;
152         String rowId;
153
154         switch(sUriMatcher.match(uri)) {
155         .
156         .
157         .
158         case AUTHUSERS_TYPE_LIST:
159            rowsUpdated = db.update(AuthUsersMetaData.Table.TABLE_NAME, values, where, whereArgs);
160            break;
161         case AUTHUSERS_TYPE_ONE:
162            rowId = uri.getPathSegments().get(1);
```

```
163             rowsUpdated = db.update(AuthUsersMetaData.Table.TABLE_NAME,
164                     values,
165                     AuthUsersMetaData.Table.COLUMN_ID + " = " + rowId + (!TextUtils.isEmpty(where) ?
                            " AND (" + where + ")" : ""),
166                     whereArgs);
167             break;
168         .
169         .
170         .
171         default:
172             throw new IllegalArgumentException("Unknown URI: " + uri);
173         }
174 
175         getContext().getContentResolver().notifyChange(uri, null);
176         return rowsUpdated;
177     }
178 }
```

Listing 4.3: A small part of the content provider

CHAPTER 5

Oasis Lib

In this chapter we describe the library of the administration module, called Oasis Lib. The Oasis Lib is the library, which works as a connection between the Oasis Local Db (4 on page 23 and the applications, which runs on the GIRAF system, by providing an API, which the GIRAF applications can use. First the structure of Oasis Lib is described in section 5.1. Then how the Oasis Lib is implemented is described in section 5.2 on the following page.

5.1 Structure

In the structure of Oasis Lib, we have taken inspiration of the MVC pattern, where the system is divided into three parts; Model, View, and Controller. In Oasis Lib the Model part is a package containing model objects. Each model object represent a table in the local db. The model objects is used to encapsulate the data, which is supposed to be saved and loaded from the Oasis Local Db. The reason that we use model objects is to ease it for the users and to make a uniform way of saving and loading data. The controller part is the package containing all the methods, which the developers can use to interact with Oasis Local Db. We do not have any view part. This is missing because we see it as a part of the GIRAF applications.

5.2 Implementation

The Oasis Library's controllers is what make up the main part of the library. It is devided into a number of different sub controllers, the public ones seen in figure **??** on page ??.

There are a number of other sub controllers as well, but they are only for internal use within the Oasis library. For futher information on how the Oasis library's methods work, see JavaDoc REF!.

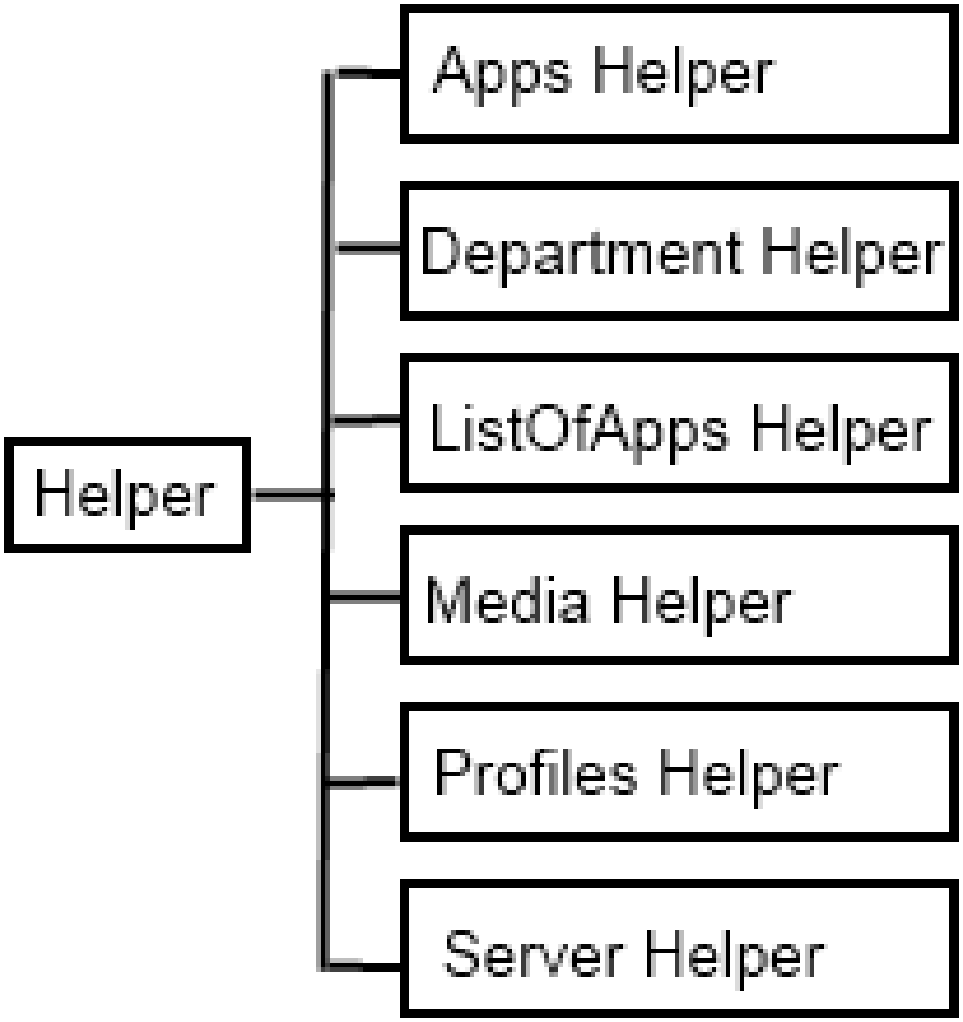

Figure 5.1: The different controlers in Oasis library

CHAPTER 6

Oasis App

In this chapter we describe the application of the administration module, called Oasis App. The Oasis App is an application, which demonstrates some of the utilities the Oasis Lib offers. The idea behind the Oasis App is that we want to make a tool, for the guardians, to manage data of the profiles, by giving them CRUD (Create, Read, Update, and Delete) options. First the structure of the Oasis App is described in section 6.1. After that the implementation of the Oasis Lib is described in section 6.2 on page 35.

6.1 Structure

The Oasis Application is build upon two activities; MainActivity and FragParentTab.

6.1.1 MainActivity

The MainActivity is the activity that starts on application startup. It uses the main.xml as its layout file, which is a layout file containing three buttons, which can be seen on figure INDSÆT SCREEN CAP AF MAINACTIVITY. A snap of the MainActivity's code can be seen in listing 6.1.

```
1 .
2 .
3 .
4 public class MainActivity extends Activity implements OnClickListener {
5
6     private Button bMyProfile, bAllProfiles, bAllDepartments, bAddDummyData;
```

```
7     private Intent direct;
8     private long guardianId;
9     public Helper helper;
10    public static Profile guardian;
11    public static Profile child;
12    public static int color;
13
14    @Override
15    public void onCreate(Bundle savedInstanceState){
16       super.onCreate(savedInstanceState);
17
18       .
19       .
20       .
21
22       helper = new Helper(this);
23
24       Bundle extras = getIntent().getExtras();
25       if (extras != null) {
26          guardianId = extras.getLong("currentGuardianID");
27          color = extras.getInt("appBackgroundColor");
28          guardian = helper.profilesHelper.getProfileById(guardianId);
29       }
30
31       setContentView(R.layout.main);
32
33       initializeViews();
34    }
35
36    private void initializeViews() {
37       findViewById(R.id.UpperLayout).setBackgroundColor(color);
38
39       bMyProfile = (Button) findViewById(R.id.bMyProfile);
40       bMyProfile.setOnClickListener(this);
41       bAllProfiles = (Button) findViewById(R.id.bAllProfiles);
42       bAllProfiles.setOnClickListener(this);
43       bAllDepartments = (Button) findViewById(R.id.bAllDepartments);
44       bAllDepartments.setOnClickListener(this);
45       bAddDummyData = (Button) findViewById(R.id.bAddDummyData);
46       if (guardian == null) {
47          bAddDummyData.setOnClickListener(this);
48       } else {
49          bAddDummyData.setVisibility(View.GONE);
50       }
51    }
52
53    @Override
54    public void onClick(View v) {
55       direct = new Intent(this, FragParentTab.class);
56
57       switch (v.getId()) {
58       case R.id.bMyProfile:
59          if (guardian != null) {
60             direct.putExtra("tabView", FragParentTab.TABPROFILE);
61             startActivity(direct);
62          } else {
63             Toast.makeText(this, R.string.noprofile, Toast.LENGTH_SHORT).show();
64          }
65          break;
66       case R.id.bAllProfiles:
67          direct.putExtra("tabView", FragParentTab.TABALLPROFILES);
68          startActivity(direct);
69          break;
70       case R.id.bAllDepartments:
71          direct.putExtra("tabView", FragParentTab.TABALLDEPARTMENTS);
72          startActivity(direct);
73          break;
74       .
75       .
76       .
77       }
78    }
79 }
```

Listing 6.1: The MainActivity class

When the activity starts it gets the information of which guardian that is currently logged in to the GIRAF system and what background color the Oasis

App is currently set to by the Launcher application. When one of the buttons is clicked the mainactivity will start the FragParentTab activity, but depending on which button is clicked, the mainactivity will put an integer in the itent's extra data.

6.1.2 FragParentTab

The FragParentTab is the activity, which, as stated above, is started by the MainActivity activity. The activity has the responsibillity of managing what view to show, by using fragments. The reason for choosing fragments instead of making new activities is that the layout we want is a tab layout. The layout can be seen in figure Indsæt FIGUR REF. The FragParentTab activity can be seen in listing **??** on page ??.

```
1  .
2  .
3  .
4  public class FragParentTab extends Activity {
5
6     private int tabView;
7     public final static int TABPROFILE = 0;
8     public final static int TABAPP = 1;
9     public final static int TABMEDIA = 2;
10    public final static int TABALLPROFILES = 3;
11    public final static int TABALLDEPARTMENTS = 4;
12    public final static int TABCHILD = 5;
13    public final static int TABCHILDAPP = 6;
14    public final static int TABCHILDMEDIA = 7;
15    static FragmentManager t;
16
17    @Override
18    protected void onCreate(Bundle savedInstanceState) {
19       super.onCreate(savedInstanceState);
20
21       .
22       .
23       .
24
25       Bundle extras = getIntent().getExtras();
26       if (extras != null) {
27          tabView = extras.getInt("tabView");
28       } else {
29          tabView = -1;
30       }
31
32       setContentView(R.layout.fragments_view);
33
34       findViewById(R.id.fragUpperLayout).setBackgroundColor(MainActivity.color);
35
36       t = getFragmentManager();
37
38       switch(tabView) {
39       case TABPROFILE:
40          t.beginTransaction().add(R.id.fDetails, new TabManagerProfile()).commit();
41          break;
42       case TABALLPROFILES:
43          t.beginTransaction().add(R.id.fDetails, new TabManagerAllProfiles()).commit();
44          break;
45       case TABALLDEPARTMENTS:
46          t.beginTransaction().add(R.id.fDetails, new TabManagerAllDepartments()).commit();
47          break;
48       case TABCHILD:
49          t.beginTransaction().add(R.id.fDetails, new TabManagerChild()).commit();
50       }
51    }
52
53    public static void switchTab(int tabViewId) {
54
55       switch(tabViewId) {
```

```
56          case TABPROFILE:
57              t.beginTransaction().replace(R.id.fDetails, new TabManagerProfile()).commit();
58              break;
59          case TABMEDIA:
60              t.beginTransaction().replace(R.id.fDetails, new TabManagerMedia()).commit();
61              break;
62          case TABAPP:
63              t.beginTransaction().replace(R.id.fDetails, new TabManagerApp()).commit();
64              break;
65          case TABCHILD:
66              t.beginTransaction().replace(R.id.fDetails, new TabManagerChild()).commit();
67              break;
68          case TABCHILDMEDIA:
69              t.beginTransaction().replace(R.id.fDetails, new TabManagerChildMedia()).commit();
70              break;
71          case TABCHILDAPP:
72              t.beginTransaction().replace(R.id.fDetails, new TabManagerChildApp()).commit();
73              break;
74          }
75      }
76
77      @Override
78      protected void onResume() {
79          super.onResume();
80          t = getFragmentManager();
81      }
82  }
```

Listing 6.2: The FragParentTab class

The way in which the activity controls which fragment it must show is done in two ways. First, when created, the activity decides which fragment to show, by using the integer it gets from the MainActivity. This integer represents a fragment class of every view, called TabManager[keyword]. This fragment is then added to the fragment stack. The other way is when a fragment wants to replace itself with another fragment. Then the fragment calls the switchTab method, in the FragParentTab activity, with the replacing view integer as parameter.

6.2 Implementation

When using the Oasis Lib it is necessary to do a few things. First it is needed to import the Oasis Lib as a library project. This can be done by either including a jar file we made for the purpose or by telling Eclipse that the Oasis Lib project is a library project. Then Eclipse will automatically include the needed files at compile time. When the Oasis Lib is included to be able to call the methods inside the library it is necessary to initialize a helper object. When initializing the object it is needed to put the current activity's context as a parameter. This is needed to give the Oasis Lib the information about where it is called from. An example of how to initialize the helper object can be seen in listing 6.3.

```
1  Helper helper = new Helper(getActivity().getApplicationContext());
```

Listing 6.3: Example of Initializing a Helper

Now it is possible to call all the methods in the library. An example of calling a method can be seen in listing 6.4

```
guardian = helper.profilesHelper.getProfileById(guardianId);
```

Listing 6.4: Call method from Oasis Lib

6.3 Dynamic White Box Testing

To ensure the correctness of the Oasis Library we enforce dynamic white box testing (side 106 - 107, Software Testing - af Ron Patton) through unit tests. kilde: The library is used by all GIRAF applications this means that if a bug exists in the library there is a potential bug in all GIRAF application. Therefore the library must be thoroughly tested to make sure that few or no bugs exists.

As this project have been developed using the agile development methods Scrum we have not devised a full test plan (side 263 - 275, Software Testing - af Ron Patton) as this is not needed. As a part of Scrum we have been testing the code snippets before they have been marked as completed. A decision has been made **??** to make a full functioning library for this semester. This decision means that the development time has to be prioritized and therefore the focus will be on test-to-pass tests (side 66, Software Testing - af Ron Patton). This ensures that the library will function as intended, though there is no guarantee that the library will work if bad parameters are used.

6.3.1 The Test Design

A test design (side 281 - 282, Software testing - af Ron Patton) have been elaborated for each method in the helper classes of the Oasis Library. This way the library methods will be tested along with the database thus saving some time. This means that the tests will be more efficient as more code will be tested in every test, but it has the drawback that if a bug is found more code will have to be investigated in order to locate the bug.

The test design in Table 6.1 is for the `authenticateProfile()` method. This test design tests if a profile can be authenticated. This is an essential method for the whole GIRAF platform, and therefore it is tested both using test-to-pass and test-to-fail tests to ensure that this method is particular robust.

The test design in Table 6.2 is for the `getProfileById` method. This is also a very important method for the whole GIRAF platform therefore it is tested with the same mix of test-to-pass and test-to-fail tests. The test make sure that a

Identifier:	TD00001
Feature to be tested:	Authenticate profile.
Approach:	An automated test will be made to authenticate a profile by its certificate. 1. Enter a profile with a specific certificate in the database. 2. Authenticate the profile by its certificate.
Test case identification:	Check valid certificate Test Case ID# 00001 Check too long certificates Test Case ID# 00002 Check too short certificates Test Case ID# 00003 Check invalid certificates Test Case ID# 00004
Pass/fail criteria:	All valid profile certificates that matches the certificate in the database must be accepted as well as all invalid certificates must be rejected.

Table 6.1: Test Design for `authenticatingProfile()`.

profile can be found by its id and that negative id's and id's not in the database will not make it crash.

The test design in Table 6.3 is for the `getChildrenByGuardian` method. This test ensures that the method performs as intended under normal operation. This is done with a single test-to-pass test, which tests if children associated to a guardian can be retrieved from the database.

More test designs have been elaborated but have not be entered in the report due to the similarities and the large amount. Those test designs look similar to the test design for the `getChildrenByGuardian` method.

6.3.2 The Test Cases

One or more test case are created for each test design (side 283 - 285, Software testing - af Ron Patton). Every test case is created in order to test a part of a method to ensure that the method performs as intended in the tested situation.

The test cases for the test designs in the prior Section can be seen in Table 6.4, Table 6.5, Table 6.6, Table 6.7, Table 6.8, Table 6.9, Table 6.10, and Table 6.11.

Identifier:	TD00002
Feature to be tested:	Get profile by id.
Approach:	An automated test will be made in order to ensure that the Oasis Library supports getting a profile by its id from the database. 1. Add Profiles to the database. 2. Get profile by id and verify the output.
Test case identification:	Check valid id present in the database Test Case ID# 00005 Check id not in the database Test Case ID# 00006 Check negative id Test Case ID# 00007
Pass/fail criteria:	The profile matching the id should be returned else null should be returned.

Table 6.2: Test Design for `getProfileById()`.

Identifier:	TD00003
Feature to be tested:	Get children by guardian.
Approach:	An automated test will be made to ensure that the Oasis Library supports getting all children associated with one guardian. 1. Children and guardians should be added to the database. 2. Associations between some children and guardians should be made. 3. Get children by guardian should be called and the output verified.
Test case identification:	Check valid guardian with children associated Test Case ID# 00008
Pass/fail criteria:	The list of children should match the children associated with the guardian.

Table 6.3: Test Design for get children by guardian.

Identifier:	TC00001
Test item:	Valid Certificate handling of the `authenticateProfile()` method.
Input specification:	A valid certificate.
Output specification:	The model of the authenticated profile.
Environmental needs:	A database is needed and the profile model.
Special procedural requirements	None.
Intercase dependencies:	None.

Table 6.4: Test Case for valid certificate handling of the `authenticateProfile()` method.

Identifier:	TC00002
Test item:	Certificate lenght too short handling of the `authenticateProfile()` method.
Input specification:	A certificate shorther than 200 chars.
Output specification:	Null.
Environmental needs:	A database is needed and the profile model.
Special procedural requirements	None.
Intercase dependencies:	None.

Table 6.5: Test Case for certificate lenght too long handling of the `authenticateProfile()` method.

Identifier:	TC00003
Test item:	Certificate lenght too long handling of the `authenticateProfile()` method.
Input specification:	A certificate longer than 200 chars.
Output specification:	Null.
Environmental needs:	A database is needed and the profile model.
Special procedural requirements	None.
Intercase dependencies:	None.

Table 6.6: Test Case for certificate lenght too long handling of the `authenticateProfile()` method.

Identifier:	TC00004
Test item:	Invalid Certificate handling of the `authenticateProfile()` method.
Input specification:	An invalid certificate.
Output specification:	Null.
Environmental needs:	A database is needed and the profile model.
Special procedural requirements	None.
Intercase dependencies:	None.

Table 6.7: Test Case for invalid certificate handling of the `authenticateProfile()` method.

Identifier:	TC00005
Test item:	Valid id present in the database handling of the `getProfileById()` method.
Input specification:	A valid id.
Output specification:	The profile matching the id.
Environmental needs:	A database is needed and the profile model.
Special procedural requirements	None.
Intercase dependencies:	None.

Table 6.8: Test Case for valid id handling of the `getProfileById()` method.

Identifier:	TC00006
Test item:	Invalid id present not present in the database handling of the `getProfileById()` method.
Input specification:	An invalid id.
Output specification:	Null.
Environmental needs:	A database is needed and the profile model.
Special procedural requirements	None.
Intercase dependencies:	None.

Table 6.9: Test Case for invalid id handling of the `getProfileById()` method.

Identifier:	TC00007
Test item:	Negative id handling of the `getProfileById()` method.
Input specification:	A negative id.
Output specification:	Null.
Environmental needs:	A database is needed and the profile model.
Special procedural requirements	None.
Intercase dependencies:	None.

Table 6.10: Test Case for negative id handling of the `getProfileById()` method.

Identifier:	TC00008
Test item:	Valid guardian with children associated handling of the `getChildrenByGuardian()` method.
Input specification:	A valid guardian.
Output specification:	A list of associated children.
Environmental needs:	A database is needed and the profile model.
Special procedural requirements	None.
Intercase dependencies:	None.

Table 6.11: Test Case for valid guardian with children associated handling of the `getChildrenByGuardian()` method.

6.3.3 The Unit Tests

All the tests cases have been used to construct the 89 unit tests, which have helped in the development of the library. The tests have been split up into three parts: Initialization, execution, and assertion.

The unit test for the `testAuthenticateProfileWithValidCertificate()` method can be seen in Listing 6.5. The method starts by initializing the environment. This means that a random valid certificate is created and a profile – the expected profile – is entered in the database and the certificate for the profile is set to the newly created certificate.

After the initiliazation the method is executed and the retrieved profile is stored as the actual profile in order to have some data for the assertion. At the end an `assertEquals` is called to ensure that the expected profile is the same as the actual profile. If this is the case the test passes, otherwise the test fails the two profiles are printed in the test log.

```
1  public void testAuthenticateProfileWithValidCertificate() {
2     Random rnd = new Random();
3     StringBuilder cert = new StringBuilder();
4     for (int i = 0; i < 200; i++)
5     {
6        cert.append((char)(rnd.nextInt(26) + 97));
7     }
8     String certificate = cert.toString();
9     Profile expectedProfile = new Profile("Test", "Profile", null,
            Profile.pRoles.GUARDIAN.ordinal(), 12345678, null, null);
10
11    long id = mActivity.helper.profilesHelper.insertProfile(expectedProfile);
12    expectedProfile.setId(id);
13
14    mActivity.helper.profilesHelper.setCertificate(certificate, expectedProfile);
15
16    Profile actualProfile = mActivity.helper.profilesHelper.authenticateProfile(certificate);
17
18    assertEquals("Should return profile; Test Profile", expectedProfile, actualProfile);
19 }
```

Listing 6.5: The `testAuthenticateProfileWithValidCertificate()` method.

The unit test for the `testAuthenticateProfileWithInvalidCertificate()` method can be seen in Listing 6.6. This method also starts by initializing its environment but in this method the created certificate is different. The created certificate consists of chars in the range from 65 - 91 while a valid certificate hahas a range from 97 - 123. The rest of the initialization is the same as in the valid test and the execution is the same as well. But in this test `assertNull` is used to confirm that there is not retrived a profile from the database due to an invalid certificate.

```
1  public void testAuthenticateProfileWithInvalidCertificate() {
2     Random rnd = new Random();
3     StringBuilder cert = new StringBuilder();
4     for (int i = 0; i < 200; i++)
5     {
6        cert.append((char)(rnd.nextInt(26) + 65));
```

```
7       }
8       String certificate = cert.toString();
9       Profile expectedProfile = new Profile("Test", "Profile", null,
              Profile.pRoles.GUARDIAN.ordinal(), 12345678, null, null);
10
11      long id = mActivity.helper.profilesHelper.insertProfile(expectedProfile);
12      expectedProfile.setId(id);
13
14      mActivity.helper.profilesHelper.setCertificate(certificate, expectedProfile);
15
16      Profile actualProfile = mActivity.helper.profilesHelper.authenticateProfile(certificate);
17
18      assertNull("Should return null", actualProfile);
19  }
```

Listing 6.6: The `testAuthenticateProfileWithInvalidCertificate()` method.

The remaining tests from the test suite have been created in the same manor with an initialization face, an execution face and one or more assertions and the can be seen in the source code of the Oasis App.

6.3.4 The Test Results

The overall result for the unit tests are 88 out of 89 tests passed and it can be seen in Figure 6.2.

The test that failed is `testGetChildrenByDepartmentAndSubDepartments()` and the result can be seen in Figure 6.1.

Thinks that last pictures could be put into the appendix

6.4 Usability Test

Setup - hvordan blev det udfoert
Opgaver - hvilke opgaver fik testpersonerne

6.4.1 Results and Observations

Test resultater og observationer
Tabel over hvilke funktionaliteter blev der fundet fejl ved og var det kritisk eller kosmetisk

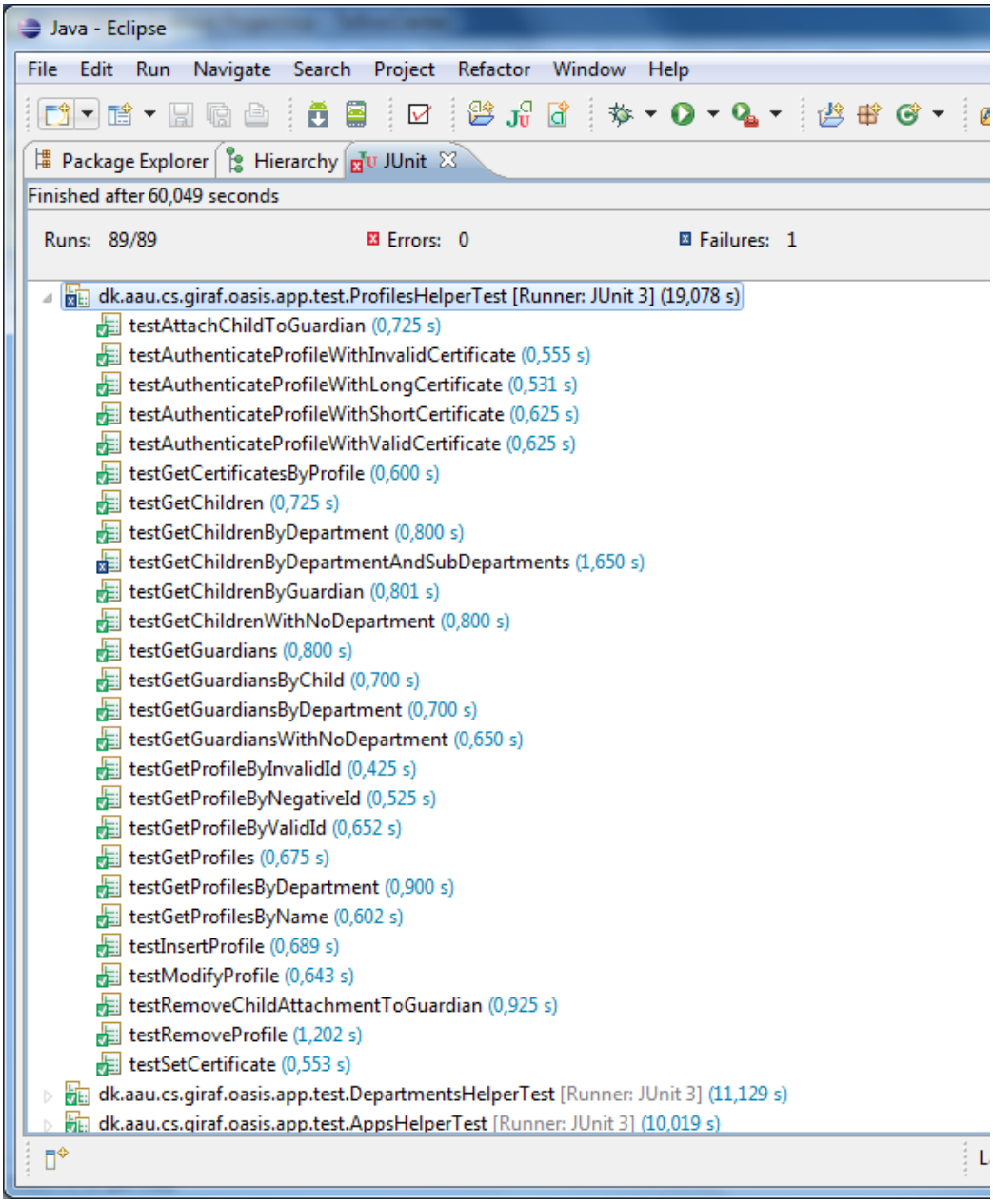

Figure 6.1: The result from the `profilesHelper` tests.

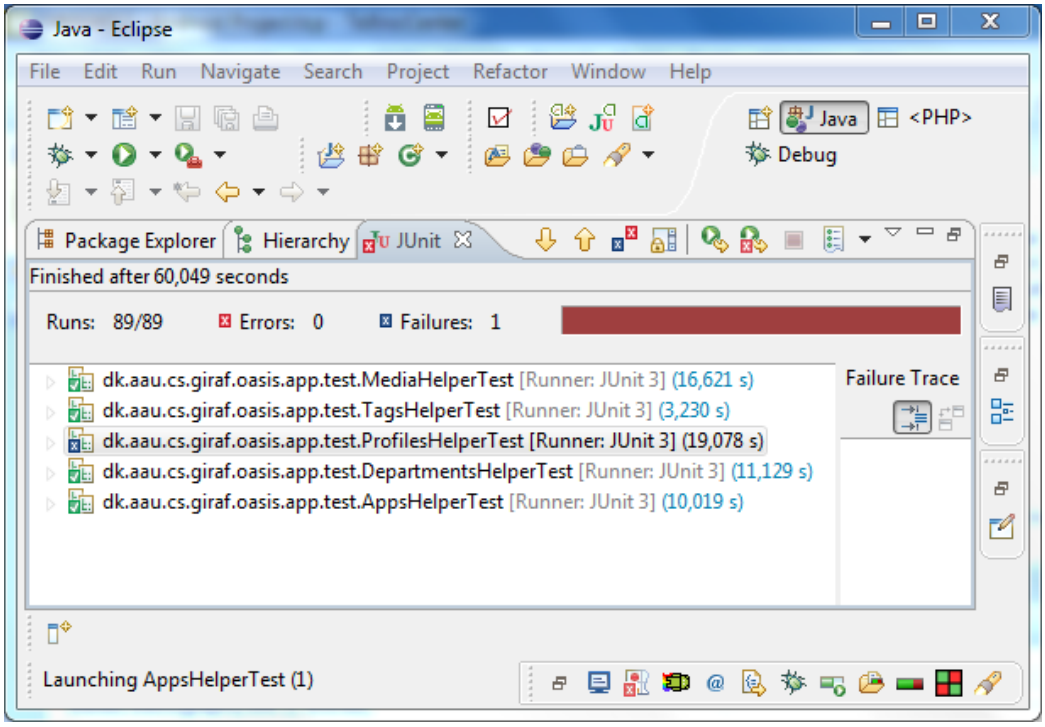

Figure 6.2: The result from all the performed unit tests.

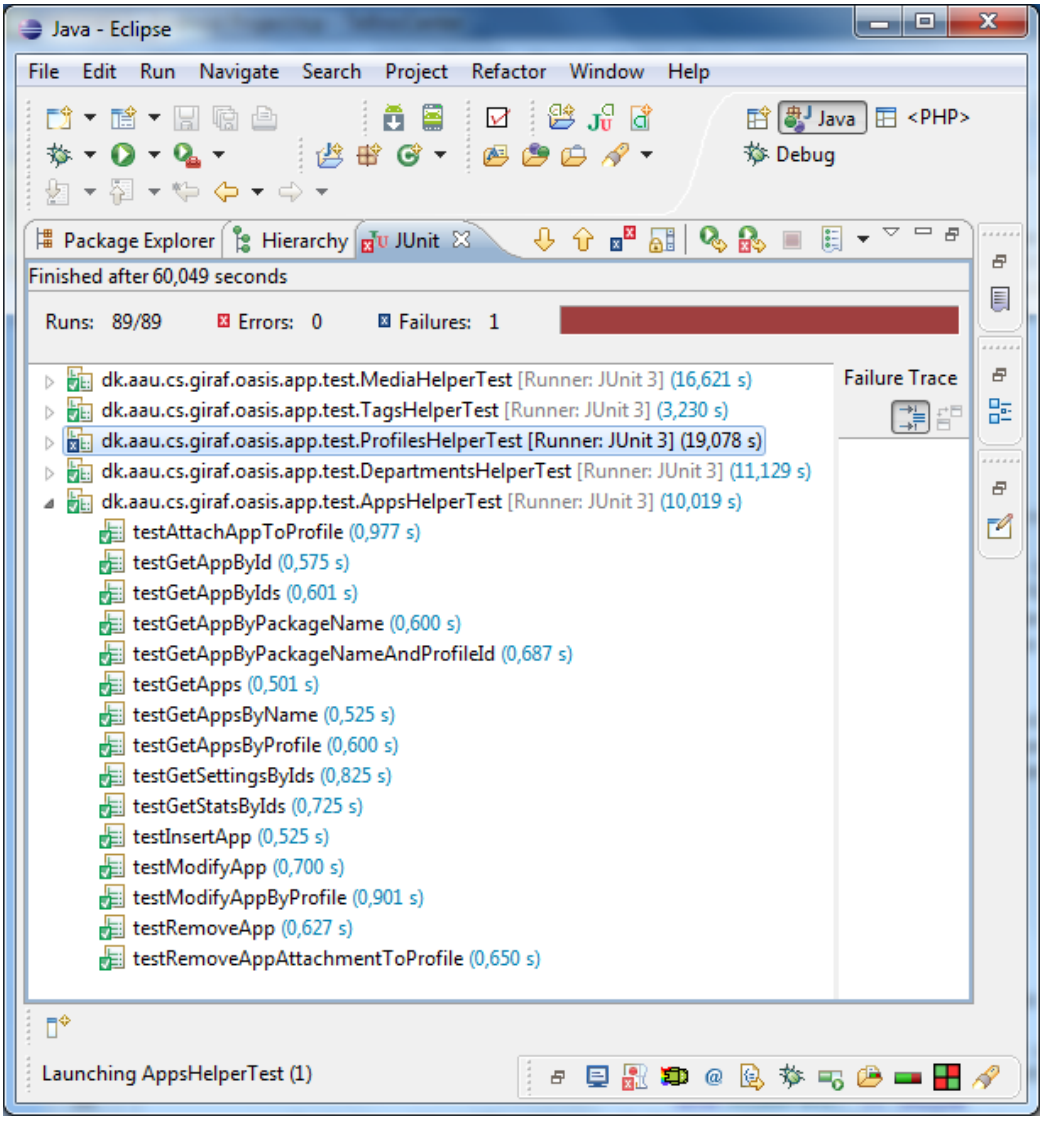

Figure 6.3: The result from the `appsHelper` tests.

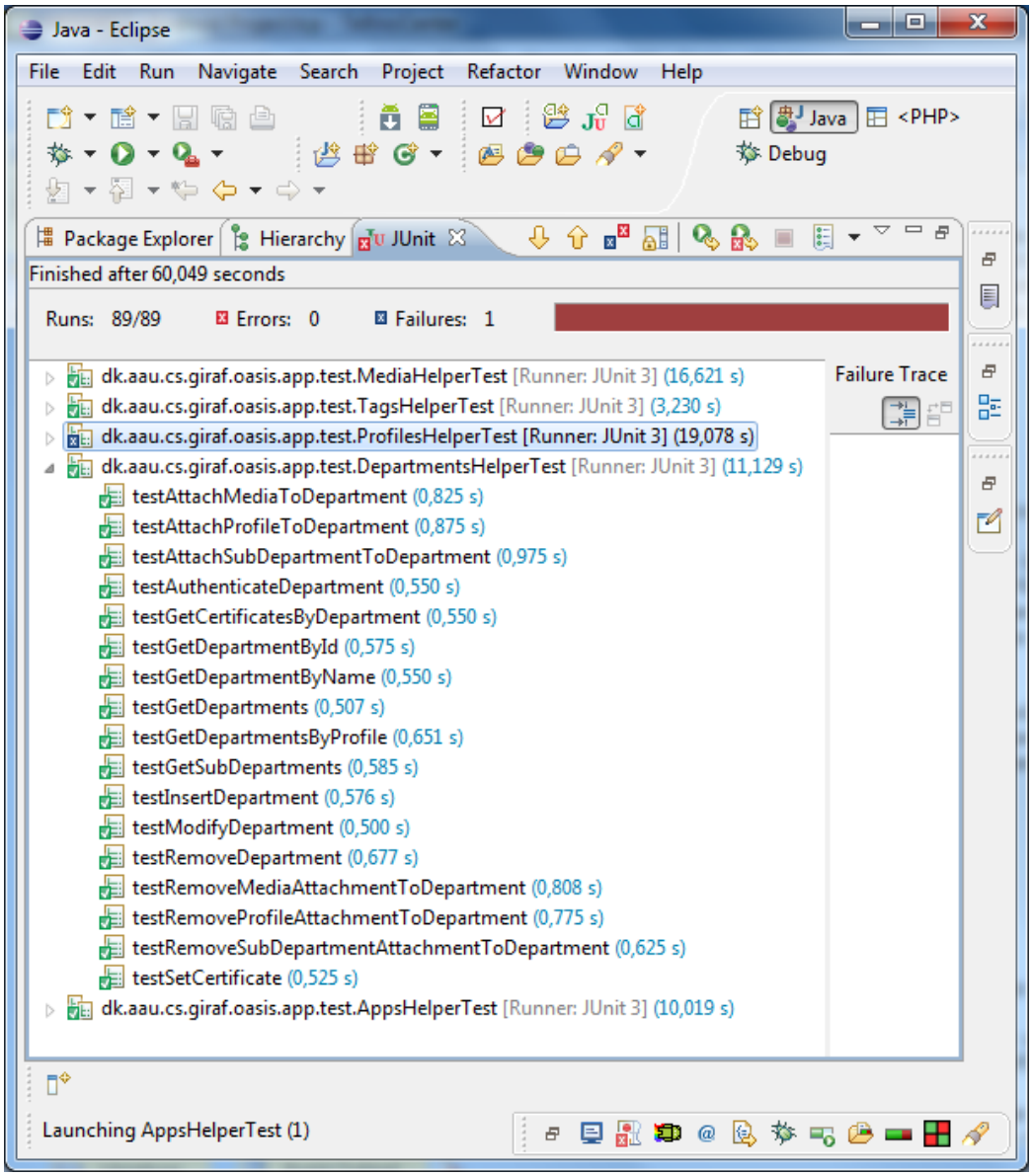

Figure 6.4: The result from the `departmentHelper` tests.

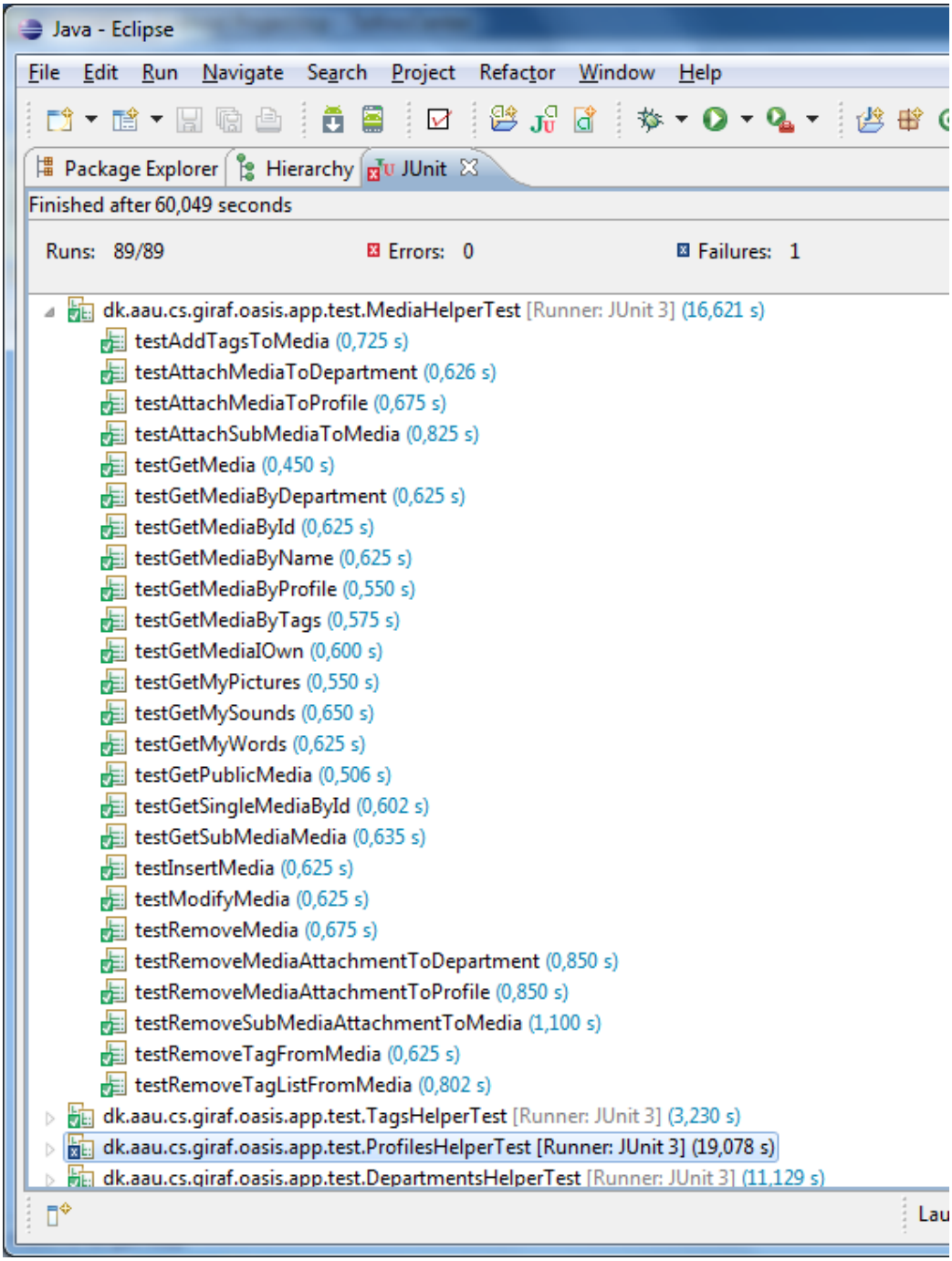

Figure 6.5: The result from the `mediaHelper` tests.

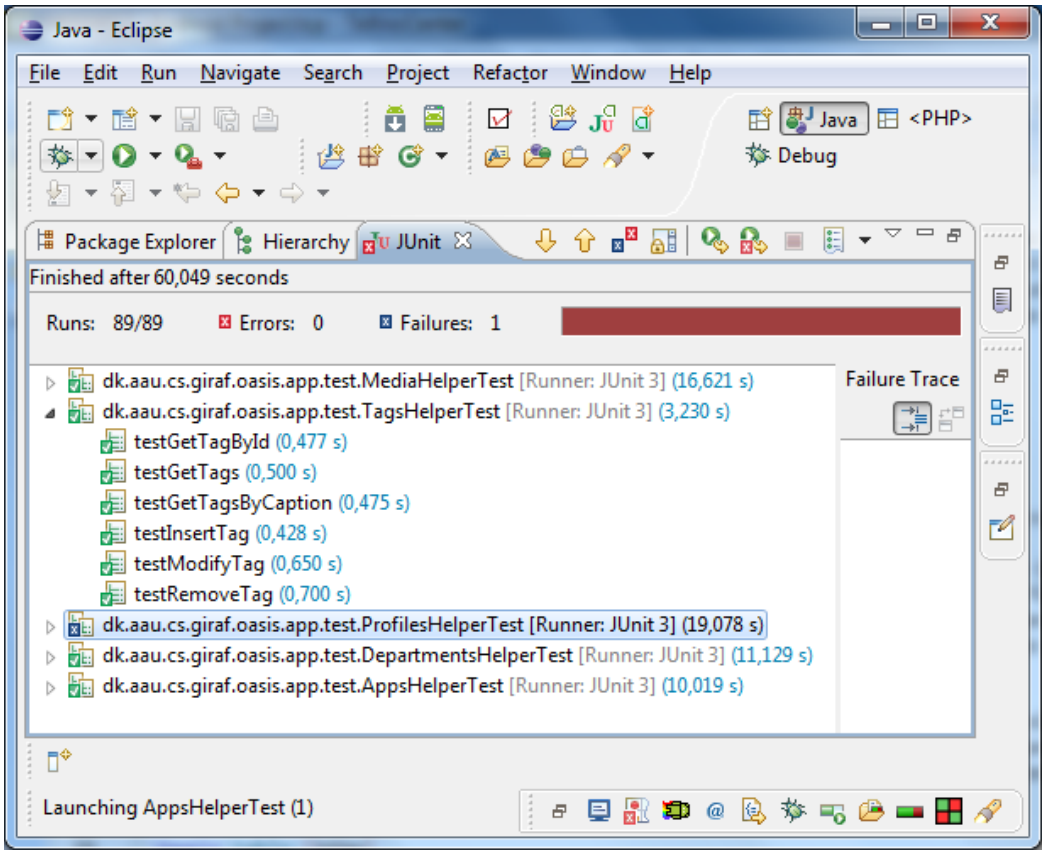

Figure 6.6: The result from the `tagsHelper` tests.

Tail

Part IV

Discussion

Head

CHAPTER 7

Reflections and Evaluations

7.1 Conclusion

7.2 Future Work

7.3 Conclusion

7.4 Future Work

A number of tasks did not get completed in this semester. As this project is properly going to be continued by others students, it is not that big of a problem.

7.4.1 Server syncronization

One of the main things which did not get completet, due to when the component we needed was available, we did not have more time to implement it, was the syncronization with the server. This can be implemented by using the components which the server group made. This would also make the sync status component in the launcher work.

7.4.2 Unit tests

Unit testing is an essential part of the project. We did manage to do some unit tests, but for future work more unit tests could be made, and it should be an ongoing process instead of doing them at the end of the semester. Test to fail, to insure the lib robust

Mere på app, refactor Certificates, timelimit, multiple certificates per user
Test suite til db provider

Tail

Part V

Appendix

Appendix

7.5 Sprint Burndown Charts and Backlogs

This page is left blank for the purpose of containing the attached CD-ROM.