Group SW603F12

Bachelor Project

Savannah - part of the GIRAF system

Authors:
Jesper Bromose
Martin Fjordvald
Sebastian Lybæk
Thorbjørn Nielsen

Supervisor:
Ulrik Nyman

June 3, 2012

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

Title: Savannah - part of the GIRAF system
Subject: Application Development
Semester: Spring Semester 2012
Project group: sw603f12

Participants:

Jesper Bromose

Martin Fjordvald

Sebastian Lybæk

Thorbjørn Kvist Nielsen

Supervisor:
Ulrik Nyman

Number of copies: X

Number of pages: X

Number of appendices: X Pages

Completed: X

Synopsis:

Children with ASD have special needs in their daily life. This can be help to interpret time and how to communicate. The costumers from Birken, Egebakken and Taleinstituttet each have requests for the product, these have been implemented in the GIRAF multi project. Here, the server part is explained: Both the database, the web interface and the synchronization of databases. The database is designed and implemented in MySQL, the web interface is implemented in Java Servlets and the synchronization of the database is implemented in Java.

WORD DEFINITION

- Pictogram - A graphic symbol that conveys its meaning through its similarity to an object.
- ASD - Autism Spectrum Disorder.
- Children - Children refers to "Children with ASD".
- Guardians - Parents, teachers, caretakers, and educators of children with ASD.
- MVC - **M**odel **V**iew **C**ontroller [18].
- GIRAF - The name of the project from last year, which we are developing further.
- WOMBAT - Name of the timer application, **W**ay **O**f **M**easuring **Ba**sic **T**ime.
- PARROT - Name of the pictogram application, **P**ictogram **A**ssisting with **R**hetoric **R**easoning **O**r **T**alking
- We - In the introduction, it refers to the entire multi project group. After the common report, it refers to the individual project group.

CONTENTS

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[25].

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 [5], explaining what it is like to live with ASD, and an interview with an educator at Birken, a special kindergarten for children (see appendix A.8 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 institu-

tion.

- 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[21]. 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. [24]

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) [26], and Scrum [1].

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 [2].

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 addi-

tional 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 [8] 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

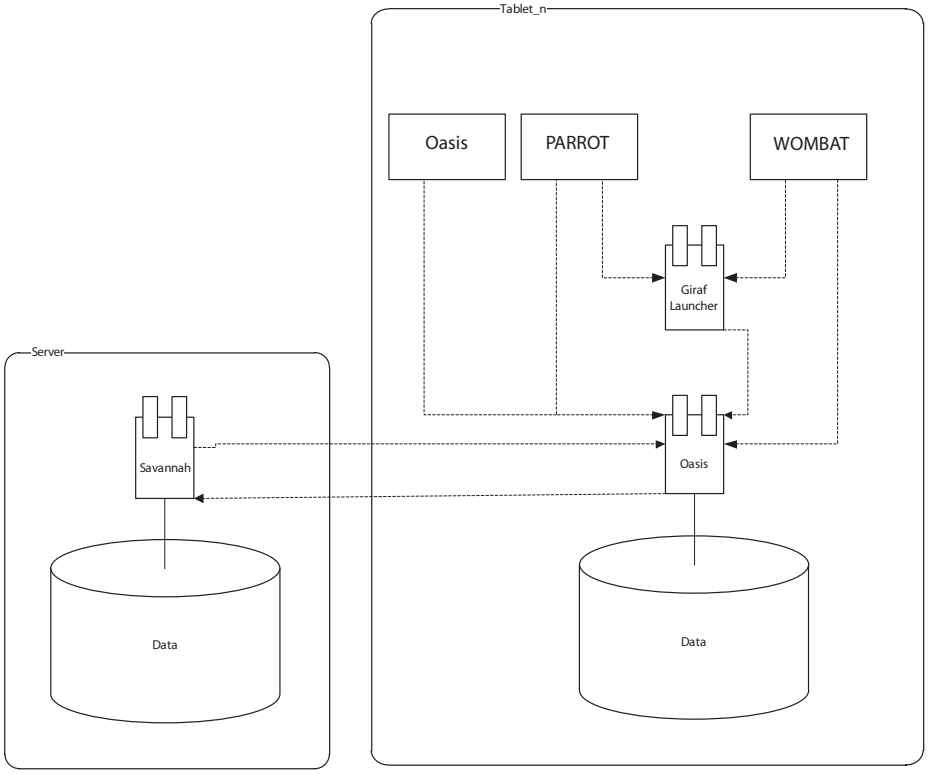


Figure 1.1: The GIRAF architecture

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 Figure A.32.

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. [10]

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.

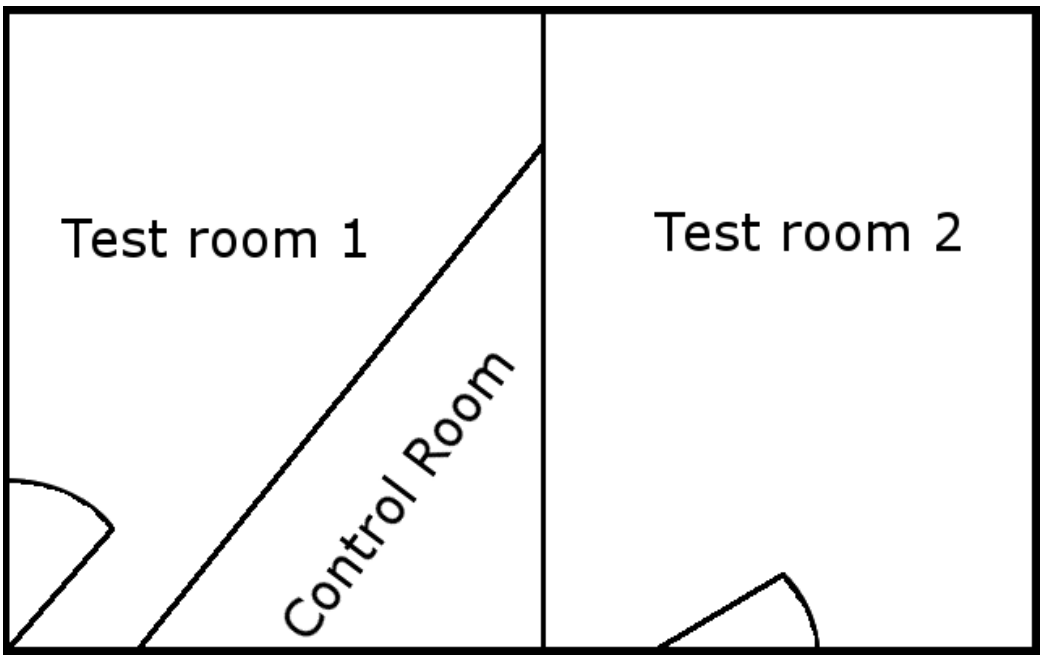


Figure 1.2: An overview of the usability lab at Cassiopeia, Department of Computer Science, Aalborg University.

Execution

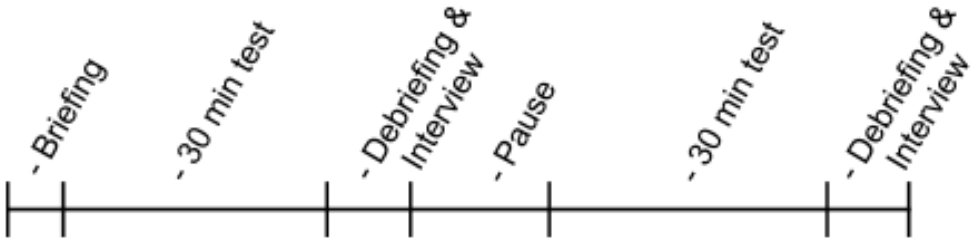


Figure 1.3: The schedule of the usability test.

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

Briefing, debriefing, and questionnaire documents can be found in Figure A.9, and the results of the test can be found in Table 2.3.

CHAPTER 2

SAVANNAH

The previous chapter, is the common part of the report for the entire multi project. In the following chapter, we will develop a system to handle the assignments from our project definition:

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

2.1 Problem Statement

The problem definition has been used to create our problem statement:

> "How can we implement a system to handle the synchronization of data from the various GIRAF apps, and administrate this?"

To be able to answer our problem statement, it is necessary to consider the following:

- Which requirements will there be to the system?
- How will the control of the data be handled?
- Which expectations does the end users have to our system?

- Which programming languages should be used?

This chapter will focus on answering the problem statement. Chapter 3 concludes on the project, gives suggestions for future development, and finally describes our experience with the multi project.

2.2 Scrum Implementation

In our project it was required that all groups used the same development method to keep things simple, hence we used Scrum as development method.

While we have chosen Scrum, some adjustments have been made to better match our personal preferences. The key points of the adjustments are as follows:

Daily Scrum Meetings The daily Scrum meeting, a tool for communicating our daily progress, is a central concept in Scrum. We have chosen not to use this tool, since we work in close quarters and have running dialogue throughout the day, therefore the meeting became redundant.

Customer Involvement Customer involvement has been a big focus in the multi project. However, our group has been in a different situation than the other project groups, as our biggest requirement providers have been the other groups. This meant that we did not have a release after each sprint to show to the customers. However, a mockup of the web interface, seen in Figure A.5, was presented to the customers at an early stage in the development, and as mentioned in section 1.8, a usability test was carried out to get feedback.

Sprint Length The sprint length has been modified because we have some days with lectures and others without. Instead we have defined the term "half days" which covers either from 8.00 to 12.00 or from 12.00 to 16.00. This interval makes it possible to use days, with lectures covering up half the day, as working days. The sprint length has been dynamic and has been decided at each sprint start. A typical sprint length could be 10 half days. This could be a week without any lectures, or two weeks with lectures.

Pair Programming Pair programming has been adopted from the XP development method[12]. The reason for doing so, is that we find it well suited for programming while learning.

2.3 Project Backlog

This section will describe how we used the project backlog, as well as how we handled each sprint.

The project backlog can be seen in Figure 2.1.

Our project backlog consists of:

Column 1: An ID for the assignment

Column 2: A name

Column 3: A priority from 1-10, 1 being the highest

Column 4: An estimate of the time taken

Column 5: A "how to demo" description

Column 6: A possible note

Column 7: A status

Note that time taken is not actual hours but a relative value. This means that an assignment with the value 4 should take twice as long as an assignment with the value 2, but not necessarily take 4 and 2 hours respectively.

The project backlog is divided into colors. The colors represent which sprint the assignment was added, starting with before sprint 1, during sprint 1, before sprint 2, during sprint 2 etc. The status attribute has four values: Not started, Completed, Waiting for external source, and In progress.

The project backlog was a new tool for us this semester, therefore we had some issues in the start of the project, which we gradually managed to solve. The main issues were the estimation of time and vaguely defined assignments.

Our first version of the backlog was short and unspecific, with large assignments, which made it hard to estimate the implementation time. We felt like no progress was made, as we did not finish most of the planned assignments during the sprint. To counteract this, we started splitting our assignments into smaller more concrete assignments. This made it easier for us to estimate the time, and we started to finish more assignments during a sprint giving a feeling of progress.

An example of this is the web interface: It was defined as two assigments called "Web Interface: GUI Design" and "Web Interface: CRUD Management". These two assignments cover the whole web interface of our project. Even though we gave them a big time estimate of 20 and 10 respectivly, this was nowhere near enough time to finish it. These two assignments kept being in the "In progress" state for quite some time. After that, we decided to split the two assignments into more concrete assigments. This resulted in 20 simpler assignments like "Profile: add" or "Tags: delete".

Our sprints reflects this progress of experience as well

Sprint 1, from 19/03/2012 to 23/03/2012 During this sprint we focused on the database and the sw6ml[1] schema. We wanted to start out light because we had no experience with how much we were capable of doing, during one sprint. We decided that it was better to start with fewer assignment on the sprint backlog, and then add assignments if necessary, rather than struggling to finish everything in the sprint. We found that we could easily finish what we had added to the sprint backlog, but this was mainly because we had done a lot of work prior to the sprint, so much of the work was already done.

Sprint 2, from 26/03/2012 to 04/04/2012 During this sprint we started working on the design of the web interface. We also had to update the sw6ml schema and the database to new requirements. Server data I/O and transmission packages was also worked on in this sprint. In the end of the sprint we had a mock up of our web interface for our contact to see, therefore we contacted her to schedule a meeting and receive feedback before the beginning of the next sprint. The meeting went well and we got some feedback that was added on to our requirements list.

Sprint 3, from 10/04/2012 to 19/04/2012 In this sprint we worked on the CRUD capabilities for the server to communicate with the database. We also had to update the database design and sw6ml schema again to meet new requirements. The server controller,the main class of the server, was also planed as well as to create the Savannah data API. For the web interface we had two assignments: Update the design to meet the new requirements, and implement the web interface on the Tomcat[2]

[1]XML language for Savannah subsection 2.7.3

[2]A web server with Servlet support

server. We realized that the complexity of the web interface was higher than estimated, and we did not finish anything web interface related.

Sprint 4, from 23/04/2012 to 02/05/2012 In this sprint we added a lot of assignments to the project backlog. This sprint's backlog was unlike the previous, in that it had almost trice as many assignments. This was because of the assignment reassessment: All the assignments were smaller and more manageable. Even though there were almost trice as many assignments, we also finished more assignments during the sprint.

Remaining sprints, 5 to 7 Sprint 4 became our last organized sprint. The reason for this was that no more assignments were added to the project backlog so we had a clear overview of what needed to be done. We started to focus on finishing current work, instead of adding new features. This resulted in us not utilizing the sprint backlog. We started working together with the other groups to make a release, and to plan the usability test, which was carried out in sprint 6. In this period we also started working on an assignment, which had been on our project backlog through the entire project: Writing the report.

Working in sprints has yielded positive results for the multi project, as all groups have had great knowledge of what the other groups were doing, and when they could expect certain features to be finished.

2.4 Requirements

The following section concerns requirements gathering done in the multi project group, as well as the continuous requirements gathering done in the project group.

Initial Requirements Gathering

Requirements gathering was done in the multi project group in the early stages of the project, with continuous requirments gathering throughout the project. In the initial stages we did semi-structured interviews with our customers, where the primary focus was understanding our target group and exploring any tools they currently have access to, while allowing the customers to present their own ideas and visions of the project.

1	sw6ml schema	3	6	Use cases	Version 0.2,missing validation	*Completed*
2	Database Design	1	0,5	ER-diagram	Version 0.2 implemented	*Completed*
3	Database Installation	5	0,5	No errors		*Completed*
4	Database: Dokumenter Design	-	4	ER-diagram + text		*Completed*
5	Tomcat installation på server	5	2	No errors	Waiting for IST	*Completed*
6	Server Data I/O	2	10	ping -> pong	Must be secure (SSL suggested)	*Completed*
7	Crud db Connections	3	meh	Data in database		*Completed*
8	Crud: Create	3	5	Data in database		*Completed*
9	Crud: Read	3	5	Data in database		*Completed*
10	Crud: Update	3	5	Data in database		*Completed*
11	Crud: Delete	3	5	Data in database		*Completed*
12	Web Interface: Gui Design	4	20	Visual	Focus on performance	*Completed*
13	Web Interface: Crud Management	4	10	Visual + Data in database		**In progress**
14	Test app: Gui Design	6	20	Visual	No need for eye candy	*Not started*
15	Test app: Crud mng. via Admin layer	6	8	Visual + Data in database		*Not started*
16	Testing	-	60			*Completed*
17	Report skrivning	-				**In progress**
18	Setup Server	1	???			*Completed*
19	Xml parser	3	10	Show it works!	Check for automated tools, alternatively, implement as visitor	*Completed*
20	Define transmission package	1	4	Show the definition	Package standard	*Completed*
21	Make the IOHandler SSL	3	4	Show it authenticates	IOHandler is currently normal and it needs to be SSL	*Not started*
22	Document the design	5	8	Present the design	Diagrams and charts	*Completed*
23	Document the architecture	5	8	Present the architecture	Diagrams and charts	*Completed*
24	DB: auto pk + auto inc	4	2	DB consisitency		*Completed*
25	Update DB to requirements	1	5	Present the diagram	Update the DB to the new requirements	*Completed*
26	Update sw6ml to match DB	2	3	Present the document	Update the sw6ml to match the newest DB version	*Completed*
27	Create Savannah data api	3	8	Show to admin group	Implement an api for creating well formed transmission packets to the server.	*Completed**
28	Server Controller	3	2	Present impl. modules	A server "main" class of sorts	*Completed*
29	Uber eyecandy web-interface design	10	???	Show it	A beautiful interface for the web-interface	*Not started*
30	Architecture for Web-interface ServerSide	5	6	Show it	UML/Class diagram, possibly flow diagram.	*Not started*
31	App plugin system til web interface	8	12		Snak med wombat	*Not started*
32	Nice and clean database design	1	5			*Completed*
33	Server: Folder system	5	12		How files are going to be organized	*Not started*
34	Synchronize server DB & device DB	7	25	Show data consistency	How can we achieve this ?	*Not started*
35	Update web interface to requirements from contact person	1	5	Show the new design		*Not started*
36	Implement the web interface on tomcat	2	20	Present the interface		*Completed**
37	Server: logfile module	9	5	null	Logging commits, requests, and pings ?	*Completed*
38	Profile: add	3	8	Add profile		*Completed*
39	Profile: edit	3	8	Edit profile		*Completed*
40	Profile: delete	3	8	Delete a profile		*Completed*
41	QR code generation	3	8			*Completed*
42	Picture: add	3	8	Add a picture		*Completed*
43	Picture: delete	3	8	Delete a picture		*Completed*
44	Picture: permissions	4	3			*Completed*
45	Picture: links	5	8	link a sound to the picture		*Completed*
46	Picture: tags	4	8	Add a tag to a picture		*Completed*
47	Tags: add	3	8	Add a tag to a picture		*Completed*
48	Tags: search	4	8	Search for a tag		*Completed*
49	Tags: delete	6	8	Delete a tag		*Completed*
50	Audio: add	3	8	Add an audio		*Completed*
51	Savannah XML outputter (Requests)	2	8	show it?	XML outputter for requests	*Completed*
52	Audio: search	4	8	Search for audios		*Completed*
53	Audio: delete	3	8	Delete an audio		*Completed*
54	Audio: tags	4	4	Add tags for an audio		*Completed*
55	Settings: parse xml	7	10			*Not started*
56	Settings: edit	7	10			*Not started*
57	Settings: save	7	10			*Not started*
58	Stats	8	12			*Not started*
59	Make the server work with admin					*Not started*

Figure 2.1: Project backlog.

From these interviews we created an initial list of requirements for the multi project.

Three interviews were done: Mette and Kristine from Birken[3], Drazenko from Egebakken[4], and Tove from Taleinstituttet[5]. From the interviews we gathered a list of their ideas and visions, and made a requirements list that the project groups can consult.

- Mette And Kristine

  Customizable software

  Ease of use, compared to current physical tools

  Emphasize visual stimuli

  Continuous stimuli

- Drazenko

  Customizable software

  Visual abstract concept

  Emphasize visual stimuli

  Unambiguous

  Consistency/structure

- Tove

  Customizable software

  Engaging/entertaining software

  Authentic/proper feedback or behavior

A list of common requirements have been created:

- Customizable software

  Some way to distinguish unique users on the same tablet is required, since people with ASD have different needs and have very different perceptions of the world.

  Customization is more important than many features.

[3]A kindergarten for children with special needs
[4]A school for children with special needs
[5]The speech center

- Emphasize visual stimuli

  People with ASD are visually over-stimulated.

- Authentic/proper feedback or behavior

  If a dog is depicted in an application, it should act, sound, and feel like a dog.

Continuous Requirements Gathering

As an integral part of Scrum, we have had continuous requirements gathering. For Savannah, the continuous requirements gathering comes from discussing the needs of Savannah with the other project groups, as well as talks with the customers during the project.

During the project we have gathered these requirements:

- PARROT

  Linking audio and images in the database.

- Launcher

  QR-code support and storage in the database.

  A live test of the connection to Savannah (ping functionality).

- Oasis

  Retrieval styles: Full profile or children associated with a specific guardian.

- Customers

  Changes to the web interface.

- Multi project group

  Secure connection

The majority of requirements for Savannah has come through cooperation with the Oasis group during the design phase of the database, which was a running process that ended at the beginning of the 4^{th} sprint, where the database was frozen. Any requirements that may have arrived beyond this point was added to the project backlog for future development. The requirement listed as being set by the multi project group regarding a secure

connection, was a descission made in the multi project group. The reasoning is that the system could be handling potentially sensitive personal data, and as such secure communication is a requirement when the server is ready for deployment.

2.5 Database

To facilitate profile management, a database has been designed and implemented. This section describes the process of the database creation.

2.5.1 Design

The database is designed in MySQL 5.1.61, and resembles the local database created by the Oasis group – the only difference is, that the Savannah database has two extra attributes in the `AuthUsers` table: `username` and `password`. The reason for this is that while all GIRAF apps running on the Android platform uses QR-codes for user authentication, this is not a feasible solution for the web interface, as it can not be assumed that every user has a webcam conneceted to their computer.

Requirements

The requirements for the database has been provided by the other project groups, and are as follows:

- All users must be able to login with a QR-code.
- All departments must be able to login.
- All users must be linked to at least one department.
- All guardians must be able to be linked to at least one child.
- All parents must be able to be linked to at least one child.
- All users must be able to access their selected apps.
- All users must be able to access their own pictures, and all public pictures.

- All departments must be able to access all public pictures.
- All pictures needs to be able to be linked to tags.
- All Pictures must be able to be linked with audio and vice versa.
- A department must be able to have a subdepartment[6].

Early Diagrams

The database has had several redesigns, due to new requirements and new ideas.

First draft (0.2) The first draft of the database can be seen in Figure A.1.

Second draft (0.3) The second draft of the database can be seen in Figure A.2. Adds department access to medias.

Third draft (0.4.1) Several tables have been renamed to ease understanding.

Final draft The final database can be seen in Figure 2.2. Adds `username` and `password` to `AuthUsers`.

Current Database

A schema diagram has been made to get an overview of the different tables and their relations. This diagram is designed in DIA[13], and can be seen in Figure 2.2.

The diagram in Figure 2.2 provides an overview of the different tables and their relations: Foreign keys points to where the value is fetched from.

Normal Form

To prevent modification anomalies in the database, it can be normalized. This becomes important when a table is dealing with functional dependencies. The solution is to move the dependencies from the table to another, e.g. split a table into two new tables. An example of this is shown in Table 2.1 and Table 2.2.

[6]Example: Birken has two departments, at two different addresses, both these subdepartments needs to be linked with Birken as a superdepartment

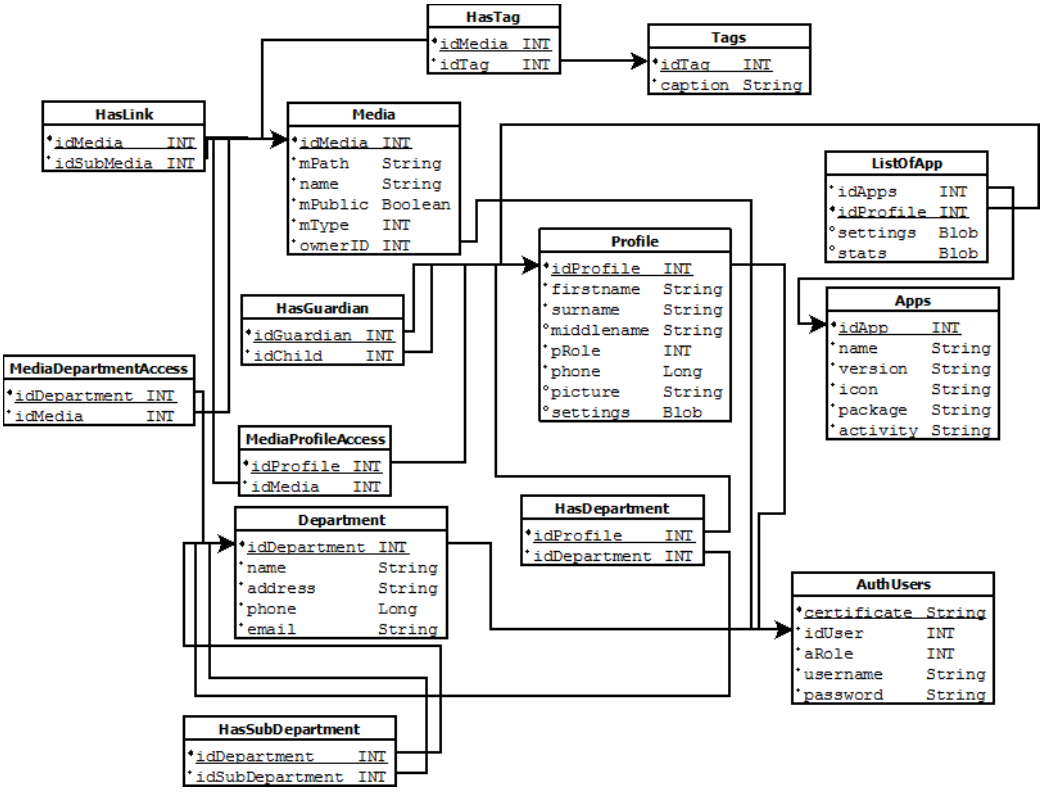


Figure 2.2: Schema diagram of the database, primary keys are underlined

SALES		
Customer_ID	Product	Price
1001	Laundry detergent	12
1007	Toothpaste	3
1010	Chlorine bleach	4
1024	Toothpaste	3

Table 2.1: Non-normalized database[23, p. 114]

The problem in Table 2.1 is that if customer 1001 is removed from the table, not only is his product removed, but the price of laundry detergent is also lost. A way to prevent this, is to split the table into two tables, as seen in Table 2.2

CUST_PURCH	
Customer_ID	Product
1001	Laundry detergent
1007	Toothpaste
1010	Chlorine bleach
1024	Toothpaste

PROD_PRICE	
Product	Price
Laundry detergent	12
Toothpaste	3
Chlorine bleach	4

Table 2.2: Normalized database example[23, p 115]

As seen in Table 2.2 the table `CUST_PURCH` now only deals with the customer, and thus it is now possible to remove a customer from the table, without loosing the product's price.

There are several degrees of normal form for a database, here the focus is on the first three: first, second, and third normal form.

First Normal Form (1NF)

The example in Table 2.2, is an example of 1NF. To achieve this the following qualities must apply[23, p. 116]:

- Each cell (intersection of a row and a column) of the table must have only a single value.
- Each column must have a unique name.
- No two rows may be identical (that is, each row must be unique).
- Each column contains data for a single attribute of the thing it's describing.

As seen in Figure 2.2 the Savannah database fulfills the qualities of 1NF, and thus is 1NF.

Second Normal Form (2NF)

For a database to be in 2NF, it is required that if a table contains a composite key, all other attributes in the table must be depended on the entire key. Furthermore [23, p. 117] states:

> ...every relation that is in 1NF with a single attribute key is automatically in second normal form.

As this is the case in the Savannah database, it is in 2NF.

Third Normal Form (3NF)

For a database to be in 3NF, it is required that there are no transitive dependency[7]. As seen in Figure 2.2 Savannah's database lives up to this quality, as there are no transitive dependencies within any of the tables. A good example of this is the handling of a profile's access to apps: When the app is removed from the profile it is done in the `ListOfApps` table, and thus only the relation is removed, and the specific app is still available in the system. This means the Savannah database is in 3NF.

Constraints

To provide the security needed in the system, several constraints need to apply for the database:

- The `Profile`→`AuthUsers` relation must be one-to-one, as one user from the `AuthUsers` can only have one profile in the system.
- The `Department`→`AuthUsers` relation must be one-to-one, as one department from the `AuthUsers` can only be one department in the system.
- The `idUser` attribute in `AuthUsers` must be unique.
- The `username` attribute in `AuthUsers` must be unique.
- It must be possible to distinguish between users and departments in the `AuthUsers` table.

[7] A transitive dependency occurs when one attribute depends on a second attribute, which depends on a third attribute. Deletions in a table with such a dependency can cause unwanted information loss. [23, p. 118]

- It must be possible to distinguish between children, parents and guardians in the `Profile` table.

These constraints will be implemented through the use of SQL scripts and Java.

2.5.2 Implementation

The implementation of the MySQL database is done in MySQL Workbench 5.2.40.

Due to the dependencies of the various tables, the order of how they need to be created is quite strict, see section A.1. To make sure the `username` and `userID` attributes follows the previously stated constraints, they have been made `UNIQUE NOT NULL` and `userID` has `AUTO_INCREMENT` as seen in Listing A.1. This ensures there can be no profiles with the same user-name or user ID, and `userID` will automatically increment when new data is inserted. This also guarantees a one-to-one relation between both `Profile` and `AuthUsers`, as well as `Departments` and `AuthUsers`.

To distinguish departments and profiles an attribute called `aRole` is used. This is an integer, which will be used at software level. The same applies for `Profile`'s attribute `pRole`, see Listing A.4.

The MySQL Workbench provides the functionality to create an ER diagram from an existing database, shown in Figure 2.3. The diagram is not completely as MySQL workbench creates it, the original is shown in the appendix section A.4. This is an error in the software, as seen in Figure A.9, the tool generates the `Profiles`→`AuthUsers` as a one-to-many relation. However, this is not possible since the `idUser` attribute in `AuthUsers` is unique, as seen in Listing A.1, the same goes for both `Department` and `Media`.

To make the deletion of data easy, many of the attributes in the tables has the constraint `ON DELETE CASCADE`. This can be dangerous, as side-effects can result in unintended data being deleted. To avoid this, the software level implementation needs to warn the user when deleting data. The following tables and attributes have the cascade constraint:

```
Profiles.idProfile
ListOfApps.idProfile
HasDepartment.idProfile
HasGuardian.idGuardian
HasGuardian.idChild
```

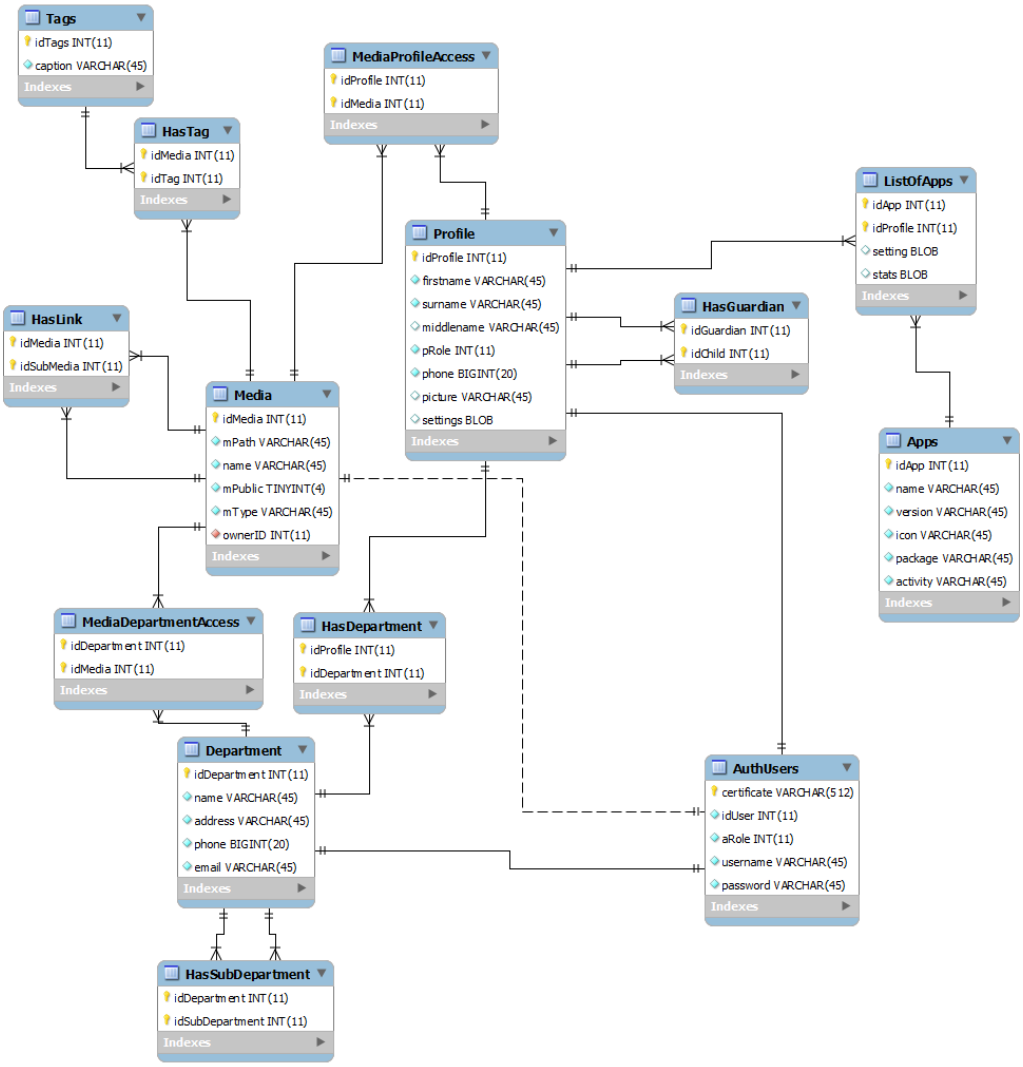


Figure 2.3: The ER diagram

```
Media.OwnerID
HasTag.idMedia
HasLink.idMedia
HasLink.idSubMedia
MediaProfileAccess.idProfile
```

Use case

A use case of the constraint is:

> "A user "Jesper" wishes to delete his entire profile"

What will happen is:

1. A deletion of the `userID` in `AuthUsers` is executed
2. The relation between `AuthUsers` and `Profiles` will delete the profile
3. The relation between `Profiles` and `HasGuardian` will delete all fields where the `userID` is either `idGuardian` or `idChild`
4. The relation between `AuthUsers` and `Media` will delete all fields where `idUser` is the owner
5. The relation between `Media` and `HasTag` will delete all fields where `HasTags.idMedia` equals `idMedia`
6. The relation between `Media` and `HasLink` will delete all fields where `HasLink.idMedia` or `HasLink.idSubMedia` equals `idMedia`

2.5.3 Test

Database testing is performed in a dynamic white-box manner. The tests are white-box as it is not the MySQL server we are testing, but rathers that the database design corresponds to the functionality which is expected. A total of 16 test cases have been designed, Figure 2.4 shows the test design for test cases #0000 to #0011. The identifier `TD0001` points to the table in Figure 2.5, which explains each step to be performed for the test cases in Figure 2.4.

Identifier	**TD0001**
Features to be tested	Database, create
Approach	Insert 2 “full” entries
Test case identification	• Insert into AuthUsers - Test Case #0000 • Insert into Department - Test Case #0001 • Insert into Subdepartment - Test Case #0002 • Insert into HasSubdepartment - Test Case #0003 • Insert into Profile - Test Case #0004 • Insert into ListOfApps - Test Case #0005 • Insert into Apps - Test Case #0006 • Insert into Certificate - Test Case #0007 • insert into HasGuardian - Test Case #0008 • Insert into HasMedia - Test Case #0009 • Insert into Media - Test Case #0010 • Insert into Has - Test Case #0011
Pass/fail criteria	Pass: A successful insertion of the entries. An error if a constraint fails. Fail: Unable to insert a correct “full” profile. Able to insert a wrong “full” profile.

Figure 2.4: Testcase design of tests #0000 to #0011.

Identifiers	Test Case #0000-0011, for each of the steps in TD0001
Test item	Insertion of a correct entry into * Insertion of a wrong entry into *
Input specification	Insertion of a int to int Insertion of string to string Insertion of null Insertion of string to int Insertion of string that is too long Insertion of int to string (Foreign key constraint holds)
Output specification	One entry inserted No entry inserted
Environmental needs	A MySQL database
Special procedural requirements	--
Intercase dependencies	--

Figure 2.5: Testcases #0000 to #0011

All the database tests are done manually and require human interpretation. Listing 2.1 shows an example of the test output, in this particular case attempting to insert a user-name longer than the allowed 45 characters.

```
1 ----------------
2 INSERT INTO AuthUsers
3 values('tolongstring',null,1,
4       'username00username00username00username00username00',
5       'hansen')
6 ----------------
7
8 ERROR 1406 (22001): Data too long for column 'username' at row 1
```

Listing 2.1: Test case output

Remaining test designs and cases can be found in A.3.

2.6 Web Interface

In this chapter the process of developing the web interface, which purpose is profile management, is explained. We start with a discussion about the best suited programming language, then an explanation of the general structure of the code, important code snippets and a navigation diagram, and finally the test carried out.

2.6.1 Programming language

To be able to provide the desired functionality from the web interface, the web pages must be dynamic and able to interact with the database. There are several different programming languages to choose from. Some of the most common are ASP.NET, PHP, and Java Servlets(Servlets). Due to the fact that ASP.NET is not open source [17], and the project is, this is not a feasible choice. This section will do a deeper analysis of PHP and Servlets, to argue for the best choice of programming language.

PHP

PHP originally emerged in 1994 when creator Rasmus Lerdorf wrote a simple set of Common Gateway Interfaces (CGI)[8] to track visits on his online resume, and decided to name it "Personal Home Page Tools" (PHP Tools). As more functionality was desired, he rewrote PHP Tools to fit his demands, and in mid 1995 he released the source code to the public.
In 1997 version 3.0 of PHP was released, and the name changed from "Personal Home Page Tools" to the recursive acronym "PHP: Hypertext Preprocessor" as it is known today. This is the first version that closely resembles PHP as it is today [15].

Key functionality

The following list is a segment of functionality found in PHP 5.4.3. This is based on [14]:

Portability PHP is supported by all major operating systems, including Linux, many UNIX variants, Microsoft Windows, MAC OSX etc.

[8]The Common Gateway Interface (CGI) allows an HTTP server and a CGI script to share responsibility for responding to client requests [4].

Output PHP provides the functionality to output not only HTML, but also images, PDF files, and Flash movies, all generated on the fly.

Databases PHP supports a wide range of different databases, including but not limited to MySQL, SQLite, and PostgreSQL. The entire list of supported databases can be found at `http://www.php.net/manual/en/refs.database.php`.

Protocols PHP can use other service protocols than HTTP, and it is possible to open raw network sockets.

Requirements

To be able to run PHP on a web server, it needs to support PHP. php.net recommends using Apache web server, and for database control MySQL [16].

Java Servlets

The first version of Servlets was created by Sun Microsystems in mid 1997, and in December 2009 version 3.0 was released[27]. It was developed to use the advantages of Java to solve the problems of performance, scalability, and security in CGI[6].

Key functionality

The following list is a segment of functionality found in Servlets[7]:

Portability Because Servlets are written in Java, they are platform independent.

Power Servlets can use the full functionality of core Java APIs, this includes URL access, multithreading, image manipulation, database connectivity etc.

Efficiency When a Servlet is loaded, in remains in the servers memory as a single object instance, this makes it able to handle requests almost immediately.

Safety Due to Servlets being written in Java, they inherit the strong type safety.

Requirements

To be able to run Servlets on a web server, this needs to support Servlets. The Apache Foundation has made a service to allow the execution of Servlets, called Tomcat[9].

Comparing PHP and Servlets

When comparing PHP and Servlets, the focus is put on efficiency and number of active sessions in both languages, as this will be the main requirements for the web interface. The following is based on [22] - a research paper by "IBM Tokyo Research Laboratory". published at Middleware 2008[10].

Efficiency

The web interface must be able to run fast and smooth, even on high load. A benchmark test is found in Figure 2.6. The test calculates a quicksort which sorts 100 integers, a Levenshtein algorithm to measure the similarity between two strings of 56 characters, and a Fibonacci execution which calculates the 15th value, with two random starting values. The setup is:

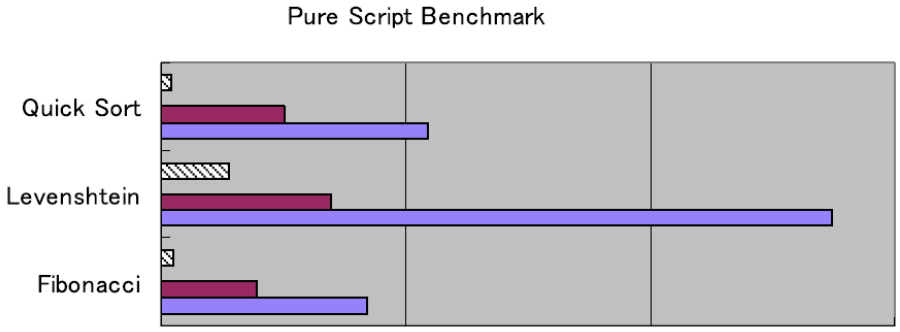
> We compared the total run time of executing each test 10,000 times with each engine. We also executed each benchmark an additional 10,000 times as a warm-up, before the measured test. This prevents Java just-in-time compilation overhead from impacting the score in the Java tests. We ran the experiment on an Intel Pentium 4 CPU at 3.40 GHz with 3GB RAM Memory, with the Linux 2.6.17 kernel. [22, p. 167]

As seen from Figure 2.6 Servlets are faster in all tests, with and without Just In Time (JIT) compilation.

Sessions

The web interface must be able to handle many active sessions at the same time. A test of this is shown in Figure 2.7[22, p. 173].

[9] `http://tomcat.apache.org/`
[10] `http://www.middleware-conference.org`



	Fibonacci	Levenshtein	Quick Sort
Java 5 without JIT	2.6	13.7	2.2
Java 5 with JIT	0.1	0.5	0.1
PHP 5.2.3	19.6	34.7	25.1
PHP 4.4.7	42.1	137.2	54.5

Run Time in Seconds

Figure 2.6: A script benchmark calculating the average time of 10000 executions [22].

> [The figure]... shows the maximum performance for each configuration and scenario, as determined by the maximum number of simultaneous sessions (e.g., users) which can be supported with acceptable Quality Of Service as defined by SPEC [22, p. 173].

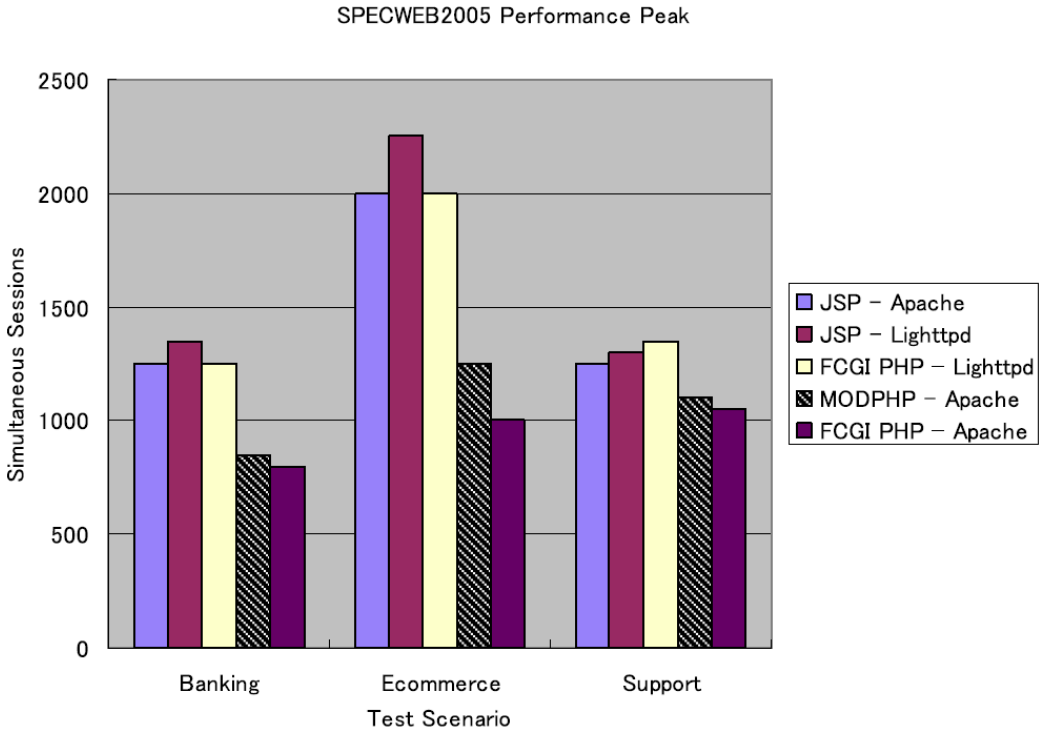


Figure 2.7: Simultaneous active sessions.

Figure 2.7 shows that in 2 of 3 test cases Java Server Pages(JSP)[11] is able to handle more active users than PHP.

The Servlets in general are more powerful than PHP, but this only comes to show when the server load is high, around 750 active sessions at the same time. Even though the efficiency of JSP is better than PHP, the following quote sums up the discussion quite well:

> When implementing a web server system which will never experience high load, or in which performance, throughput, and

[11] The elements in a Java Server Page will generally be compiled by the JSP engine into a Servlet[20].

> reliability under high load is not an issue, then the use of any of the analyzed languages or web servers will achieve similar performance results. If outstanding performance and throughput is the primary goal, then the use of JSP over PHP is advisable. However, if a 5-10% difference in throughput and performance is acceptable, then the implementer of a web system can achieve similar results using either PHP or JSP. In which case, other requirements such as developer language familiarity and programming efficiency, maintainability, security, reliability, middleware compatibility, etc. would be the deciding factors [22, p. 181].

We have chosen to implement the web interface in Servlets, due to the fact that we have a lot of experience in Java.

2.6.2 Implementation

This section describes the work of implementing the web interface in Servlets.

Requirements

The requirements for the web interface have been gathered from the other project groups, and are as follows:

Profile management It must be possible to add, edit and remove users.

Media management It must be possible to add, edit and remove media, add tags, and link pictures and sound.

Settings It must be possible to add and change the settings for a specific user.

Stats It must be possible to generate stats of the use of various apps.

Mockups

To get started, a series of mockups were created to get an idea of the general look and feel of the website. An example of this is shown in Figure 2.8, the rest can be found in Figure A.5.

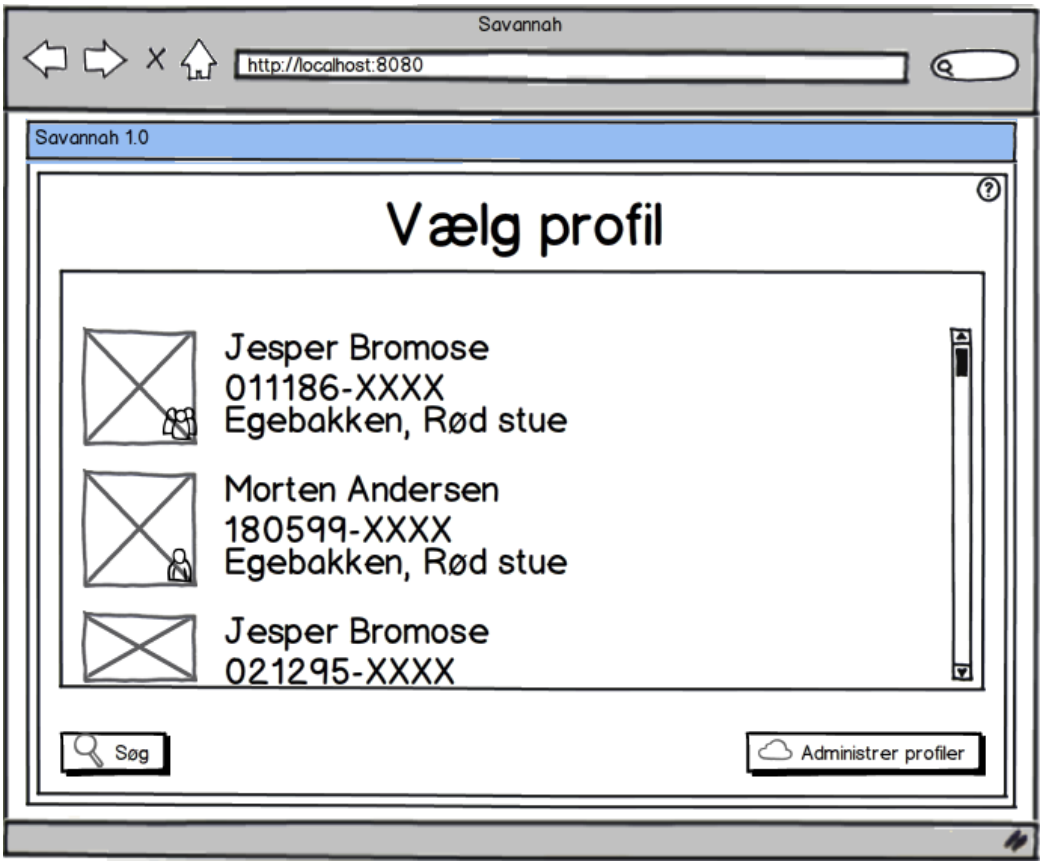


Figure 2.8: A mockup of profile selection

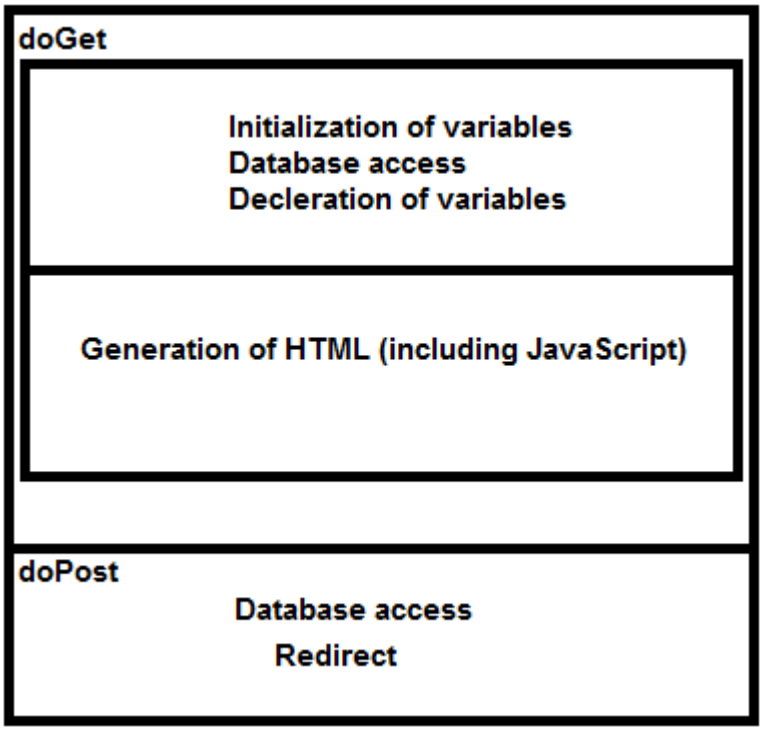


Figure 2.9: General structure of Servlets

Programming languages

The web interface uses four different languages: Servlets, HTML, JavaScript, and Cascading Style Sheets (CSS). The Servlets are responsible for database access, and generation of both the HTML code and JavaScript, and the CSS is used for the layout on each page. The HTML is used for making the elements on the webpages, and the JavaScript is used to add functionality to a generated HTML page.

Structure

The structure of each Servlet is seen in Figure 2.9

When a webpage is first loaded, it execute the `doGet(...)` method. After this there are three scenarios:

Scenario 1 The user clicks a link, and is sent to a new page.

Scenario 2 The user submits a `form` with the `doGet(...)` method.

Scenario 3 The user submits a `form` with the `doPost(...)` method.

In scenario 1, the current page does not do anything else but send the user to the new page, which executes the code in its `doGet(...)` method. In scenario 2, the `form` structure dictates what happens: It can either redirect to itself, and execute the `doGet(...)` method again, see Listing 2.2, or redirect to a new page and execute the `doGet(...)` method on that page, see Listing 2.3.

```
1  @WebServlet("/DeleteTags")
2
3  /**
4   * @see HttpServlet#doGet(HttpServletRequest request,
        HttpServletResponse response)
5   */
6  protected void doGet(HttpServletRequest request,
       HttpServletResponse response) throws ServletException,
       IOException {
7
8  out.println("<center><form method='GET' name='formName'
       action='DeleteTags'>");
9  }
```

Listing 2.2: A `form` which redirect to its own get method

```
1        @WebServlet("/DeleteTags")
2
3        /**
4         * @see HttpServlet#doGet(HttpServletRequest request,
              HttpServletResponse response)
5         */
6        protected void doGet(HttpServletRequest request,
             HttpServletResponse response) throws ServletException,
             IOException {
7
8        out.println("<center><form method='GET' name='formName'
             action='NewPage'>");
9        }
```

Listing 2.3: A `form` which redirect to another page

Scenario 3, has almost the same code as Listing 2.2 and Listing 2.3, with the exception that the `method='POST'`.

In general the `form` is build as `<form method='POST/GET' name='NameOfForm' action='WhichPageToExecuteCodeFrom>`.

Database Access

Reading data from the database is handled by Java code in each of the Servlets, shown in Listing 2.4. To update attributes in the database, the `stmt.executeQuery("SQL query")` method is used, with the appropriate SQL script. To delete data, instead of `stmt.executeQuery("SQL query")` a `PreparedStatement` is created, this is executed with by calling `int i = ps.executeUpdate()` which will return a response code.

```
1  public void readData() {
2  String dataVar;
3  Connection con = null;
4  Statement stmt = null;
5  ResultSet rs = null;
6  try {
7     Class.forName("com.mysql.jdbc.Driver"); //Use this driver to access
          the database
```

```
8     con = DriverManager.getConnection("jdbc:mysql://
          DatabaseAddress", "Username", "Password"); //Instansiate the
          connection
9     stmt = con.createStatement(); //Instansiate stmt
10    rs = stmt.executeQuery("SQL query"); //Executes the SQL query
          and get the result store in RS
11    //As long rs contains data, read it!
12    while (rs.next()) {
13       String name = rs.GetString("Name"); //Get a string attribute
14       int number = rs.GetString("Number"); //Get a int attribute
15 }
16 //Error handeling
17 catch (SQLException e) {
18    throw new ServletException("Servlet Could not display records
          .", e);
19 }
20 catch (ClassNotFoundException e) {
21    throw new ServletException("JDBC Driver not found.", e);
22 }
23 //Make sure the connection is closed, no matter what
24 finally {
25    try {
26       if (rs != null) {
27          rs.close();
28          rs = null;
29       }
30       if (stmt != null) {
31          stmt.close();
32          stmt = null;
33       }
34       if (con != null) {
35          con.close();
36          con = null;
37       }
38    }
39 }
```

Listing 2.4: Code to read data from the database

Generation of a Web Page

The generation of a web page, is done within the Servlet, using a `PrintWriter`. This is used to write the HTML code by calling `out.println("HtmlCode")` which appends the argument to the page. Furthermore, `out` can be used to write both JavaScript and CSS.

The CSS is not written within any of the Servlets, instead it is located in a separate file and is included in the HTML by `out.println ("<link rel='stylesheet' type='text/css' href='CSS/SavannahStyle.css' />")`.

Exchange of Data

When a Servlet submits a form, the data is sent differently depending on the `<form method='...'/>`. A `GET` will send the data as a part of the URL, thus making it visible for the user, as seen in Listing 2.5, whereas the `POST` method does not make it visible.

Form Data

To access the data from the `form`, the elements in the `form` needs a name. The name is set by the `name` attribute in the input element. The type of data sent from the `form` depends on the type of element.

Text/Password seen in Listing A.15. The value of these fields are of the type string, and contains the value entered into the text field on the page.

RadioButton seen in Listing A.16. All `RadioButton`s with the same name, sends `some_value` of the selected `RadioButton`.

CheckBox seen in Listing A.17. All `CheckBox`es with the same name sends `some_value` of each checked box.

DropDown menu seen in Listing A.18. Sends `some_value` of the selected item.

Multiple Select seen in Listing A.19. Sends `some_value` of all the selected items as a string array.

```
1 http://www.xoxoxoxox.com/servlet/ServletName?var1=value&var2=value
    &var3=value
```

Listing 2.5: URL with visible parameters

To be able to retrieve the data sent by the `form`, the `request .getParameter()` method is used. An example of this is shown in Listing 2.6.

```
1         @WebServlet("/PseudoClass")
2         /**
3          * @see HttpServlet#doPost(HttpServletRequest request,
               HttpServletResponse response)
4          */
5         protected void doPost(HttpServletRequest request,
              HttpServletResponse response) throws ServletException,
              IOException {
6                 String myVar = request.getParameter("FieldName"); //
                      Get single Value
7                 String[] myVarArray = request.getParameterValues("
                      FieldName") //Get string array
8         }
```

Listing 2.6: How to read parameters

Current Limitations

When uploading a picture, a `<input type=file .../>` is used to select the local file on the hard disk. The `file` tag has a security restriction, which makes it impossible to set a default value for the field. This is due to the fact, that if it was possible, it could be used to send a file from the visitor of a web page's hard disk, without the visitor having any knowledge of it[11]. Futhermore it was not straight forward to get the path of a selected file, again due to security restrictions, but this turned out to be possible by the use of JavaScript, see Listing 2.7, and adding the attribute `onChange="readURL(this);"` to the `file` tag.

```
1  var reader = new FileReader();+
2  reader.onload = function(e)
3     {
4        document.billedet.src = e.target.result;
5     };
6
7  function readURL(input)
8     {
9        if(input.files && input.files[0])
10       {
11          reader.readAsDataURL(input.files[0]);
12       }
13       else
14       {
15          document.billedet.src = "";
16       }
17    }
```

Listing 2.7: Hack to load file from `file` box

A navigation diagram of the web site is shown in Figure 2.10, and shows that all pages and information can be accessed within three clicks.

2.6.3 Test

In this section we have described the tests we have made for our web interface. First we present our results from the usability test, then we present the test cases and the results.

Usability Test

Our usability test was carried out using the IDA method, described in section 1.8. The results are as shown in Table 2.3

In summation: 11 errors were defined as cosmetic, 7 as serious, and 4 as critical.

Error	Frequency	Category
Create a profile was difficult and counter intuitive	3	Serious
The home button was only on some sites and confuses the user	5	Serious
/imagenull would appear for no reason when a user was created	3	Cosmetic
HTTP errors during the test	5	Critical
Access to apps caused confusion, the user could not figure out what it was used for	5	Cosmetic
Choose file was not further specified and caused confusion	3	Cosmetic
GUI size was not filling up the whole screen which some of the test subjects found inconvenient	2	Cosmetic
QR-code page misunderstanding home link	3	Cosmetic
Add guardian to child was counter intuitive	3	Critical
Edit profile made the users believe that they had to change password every time	2	Serious
When logging in on child's profile as guardian the dropdown menu was confusing	4	Serious
The user was confused of what to do after login	1	Cosmetic
Edit/add picture was confusing	3	Serious
The phone number was required but this is not the situation in reality	5	Critical
Missing caption for choosing role causes confusion	1	Cosmetic
The test subject was unsure which tags corresponded to which check-box	2	Critical
Confusing, too many options	1	Serious
The test subject tried to login as one of the predefined users because they thought it was their profile	2	Cosmetic
The user lost the overview	5	Serious
The user was not sure which pictures were public when asked to delete a public picture	1	Cosmetic
When deleting pictures, the tags in brackets caused confusion and the text fields implied that you could write in them which is not the case	1	Cosmetic
The test subject was unsure how to get back from the audio page	1	Cosmetic

Table 2.3: Results of the usability test

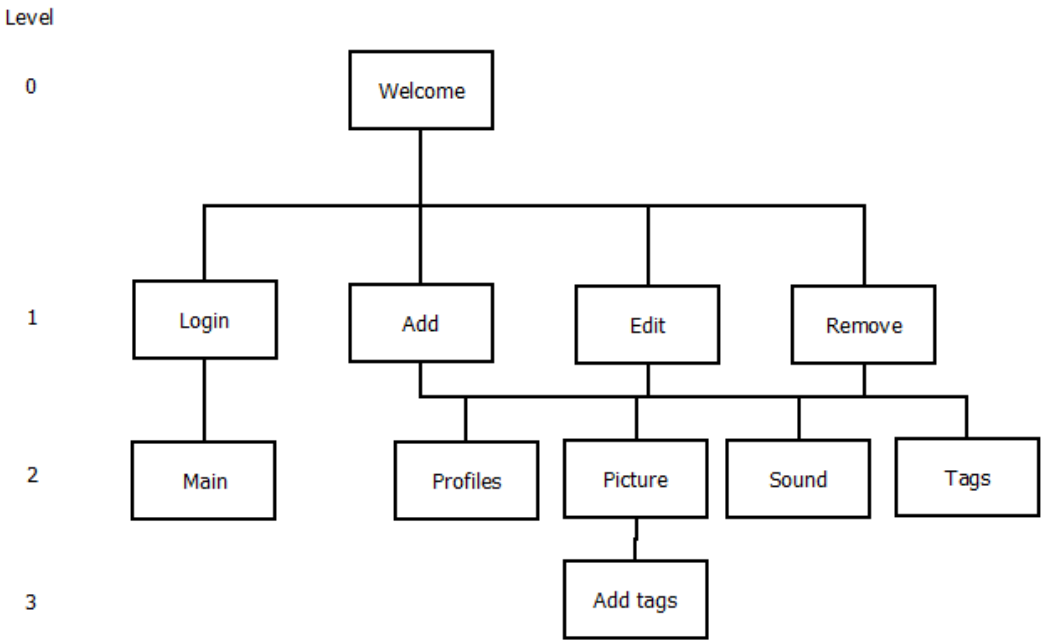


Figure 2.10: Navigation diagram for the web site

The critical errors were that the page broke down with an error. another critical error was that a phone number was required for a child, while it should be optional. This is a requirement that has not been implemented, because we have not received the request before the test. When prompted to add a guardian to a child there was confusion on how to do that. The reason is mainly because it is not placed in an intuitive place, and the test subjects had to get help from the test monitor. The last critical error was the tags selection. The problem was that the user did not know, if a checkbox was assigned to the caption above or below. While this seems as a cosmetic or serious problem at worst, while the user was experimenting with the tags, the system crashed for unknown reasons. We have not been able to recreate this crash so we have rated the problem as critical although the system might not crash.

The rest of the problems found in the test was primarily things missing, bad layout, or misplaced functionality. Missing things include “back” or “home” options from certain pages or captions that tells the user what certain fields require as input when filling out formulas. Bad layout is the main reason for confusion when using the system, this could be too much information

presented to the user at once. Misplaced functionality includes the placement of the profile list on the main page, which a lot of users thought related to editing and adding profiles – leading to confusion.

Test cases

To test the web interface we have made some use cases, which reflects how we assume the system will be used, and we have tested if it works as intended.

Case 1:

(a) Karen Tailor Smith is an educator at Egebakken. She wants to create her own profile in the system. She wants her user name to be KTS and her password to be oliver (the name of her son).

(b) She then wants to create a profile for one of the children at Egebakken. The child's name is Eric Carter, she chooses his user name to be "Eric C" with the password "EC". He should have access to TestAppe1, which is an app on the GIRAF system. She then sets herself as his guardian.

(c) Karen realizes that she wrote the wrong phone number for Eric and wants to change it.

(d) When the summer comes, Eric is done at Egebakken, so Karen wants to delete his profile.

Case 2:

(a) John Glenn is an educator at Birken, and wants to add a picture to his profile. The picture is of dog and he wants the picture to have following tags: Dog, Animal, and Brown. The tags, Animal and Brown, are not amongst the existing tags. He wants to add these to the tags and add them to his picture.

(b) John Glenn if as guardian for a child, and he wants the child be able to see the picture.

(c) He also wants to add a new tag, Mammal, to his picture but this is not in tags yet.

(d) He realize that he spelled Mammal wrong so he has to edit it

(e) The next day he finds a better picture of a dog and want to delete the old one.

Case 3:

(a) Gabriel Ryder is a parent to Mike who attends Birken. Gabriel wants to add a picture of a cat for her son. She wants the picture to have the tags Cat and Small. Small is not in the tags so she has to add it

(b) She also want to add the sound of a cat to the picture. She want this sound to have the tags Loud and Cat. Loud is not in the tags so she has to add it.

(c) She learns that cats scare Mike, so she want to remove it.

The results of the test cases are shown in Table 2.4

Case assignment	Result	Note
1.a	Success	
1.b	Success	
1.c	Success	
1.d	Success	
2.a	Success	
2.b	Success	
2.c	Failed	You are not able to add a tag to an existing picture, a workaround is to delete the picture, then add it again with the desired tags.
2.d	Success	
2.e	Success	
3.a	Success	
3.b	Success	The link is made, but there is no way to use it in the current version of the web interface.
3.c	Success	

Table 2.4: Results of our web interface test cases

There were only one of the test cases that did not succeed. The reason is that we have not implemented the feature.

2.7 Serverside

The following section concerns the design and implementation of the server side software for Savannah and the sw6ml language, which is an XML language designed for data transfers.

2.7.1 Architecture

An overall design and architecture was created in the early sprints of the project. The server side software is different from the rest of the software made in the multi project group, as the customers will never actually see it in action. It works as intended if they never notice it is there. Almost all requirements from our customers concern the user interface and general feel of the GIRAF apps. In regards to the server software, it is the requirements of the multi project that are of interest, in particular the Oasis group, as their responsibility is to link the GIRAF apps to Savannah.

The GIRAF system is designed in such a way that it deploys with two databases: Savannah, our project, a global database for a full deployment unit, and Oasis, or localDB, a local database which exists only on the mobile devices, which has an almost identical schema to Savannah's database. Rather than having Oasis query the global database directly, it was decided to implement access to both the databases through a software layer, as shown in Figure 2.11. The pros and cons of this extra software layer was considered, see Table 2.5

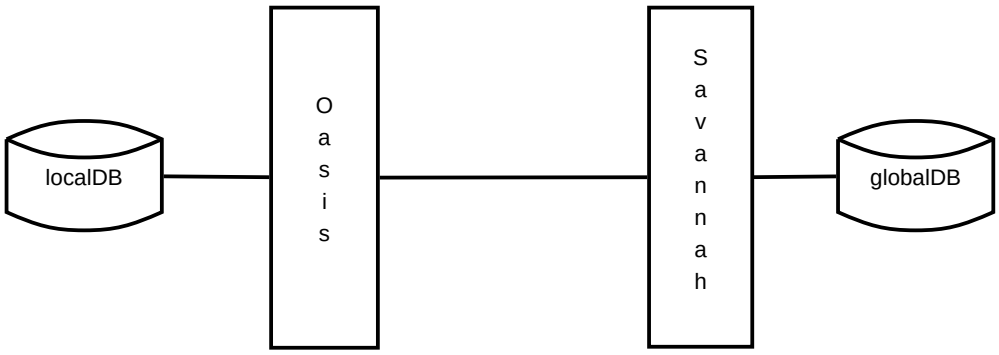


Figure 2.11: Software layers

Pros	Cons
More flexibilty	Higher complexity
Independent DB updates	Lower performance

Table 2.5: Pros and cons of an extra software layer between the databases

Having an extra software layer between the databases, means we can alter the global and local databases independently of each other. By providing software with methods for extracting specific details from Savannah, like full profiles, Oasis does not have to worry about Savannah's internal database schema. The downside of this is that the complexity inevitably will be higher, and performance will be lower. The performance issue is not critical though, since the perceived performance of the system will be dominated by the bandwidth of the mobile devices. The communication between the software layers will be facilitated by the sw6ml XML language, presented in subsection 2.7.3.

Savannah has a three layered internal architecture, shown in Figure 2.12.

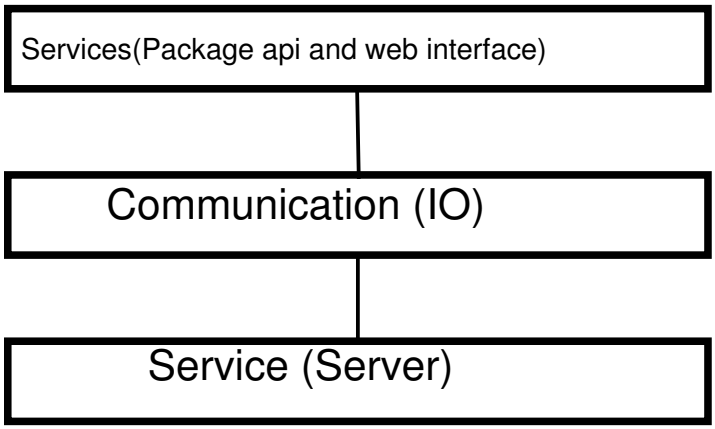


Figure 2.12: Savannah's architecture

A short description of each layer's responsibilities follows:

Services The services layer consists of the services that we provide to external users. We have considered two services to include, which are an API for creating transmissions packages that Savannah understands, and the web interface described in section 2.6.

Communication The communication layer consists of the `io` package of the server side software, which handles retrieval and responding.

Service The service layer consists of event handling, query handling and the building of sw6ml documents.

2.7.2 Design

The overall server software is designed around a producer-consumer pattern, where Oasis acts as the producer through Savannah's communication layer. Request or commit packages sent to Savannah will be processed by the `IOHandler`, which will add an event to the `EventQueue`. The consumer is the server itself, which will remove events from the queue and process its content, being it a request or a commit. Using a queue based design was chosen for the sake of simplicity. The project is a part of a learning process, and building a server with concurrent event handling was down prioritized versus a simpler server, which would allow us to gain experience and still meet the requirements of the study regulations. A draft of the design for the server can be seen in Figure 2.13. This diagram represents the general idea and flow of the server, and should not be understood as formal technical specification.

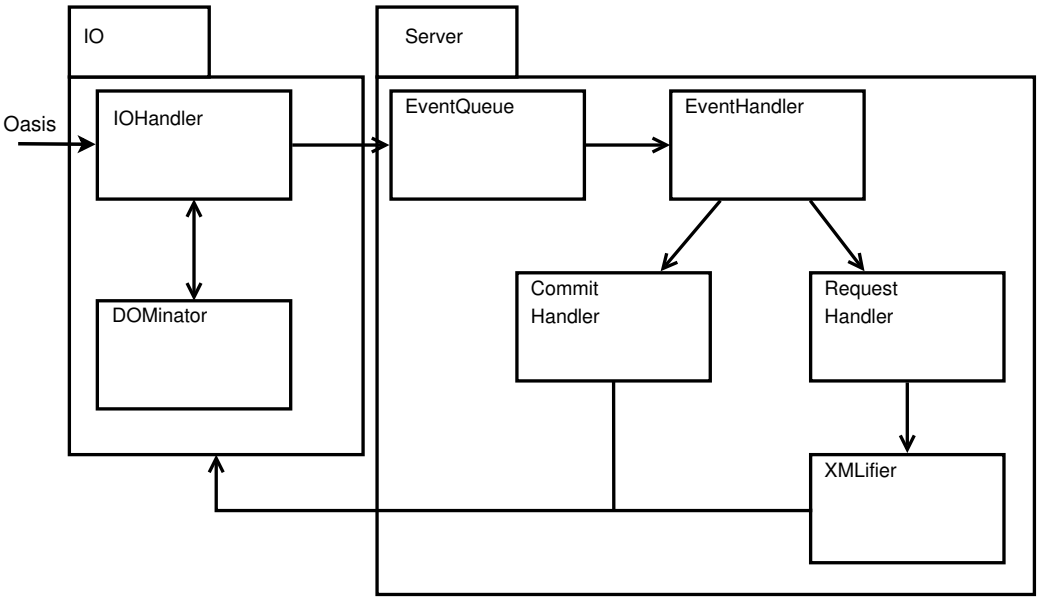


Figure 2.13: Savannah server draft

2.7.3 sw6ml

In order to facilitate consistent data transfers between the global database and localDB, we have designed an XML language which resembles the schema of the database. We have chosen to use XML, as it is a recognized standard with a wide array of tools available, in particular JDOM[28]. JDOM is a light weight implementation of SAX[12] and DOM[13] for Java, which allows seamless integration of XML, with support for XPath[14] and XSLT[15]. In section 2.7.3 a short usage documentation for sw6ml is provided. sw6ml is defined with XML schema.

Language Design

The sw6ml language consists of a number of primary elements, which reflect the tables in the database. Each of these elements accepts any number of `Entry` elements, that tells the server which action it should take with the

[12]Simple API for XML[30]
[13]Data Object Model
[14]Language for navigating XML documents[31]
[15]extensible Stylesheet Language Transformations[32]

incoming data. In Listing 2.8 a short example of legitimate sw6ml syntax for adding a row to the `AuthUsers` table, and deleting a row with the `idUser` table attribute equal to 2, is shown.

The `<AuthUsers>..content.. </AuthUsers>` element identifies the table on which we want to make changes, the following `<Entry>..content.. </Entry>` elements define which row and what kind of action, through the `action="foo"` XML attribute, should be done on the row.

```
1  <AuthUsers>
2     <Entry action="create">
3       <certificate  type="string">This is a certificate</certificate>
4       <idUser type="int">1</idUser>
5       <arole type="int">1</arole>
6       <username type="string">mette</username>
7       <password type="string">obfuscated</password>
8     </Entry>
9     <Entry action="delete">
10      <idUser type="int">2</idUser>
11    </Entry>
12  </AuthUsers>
```

Listing 2.8: Example of sw6ml syntax

The `action="foo"` XML attribute has four legal types corresponding to the CRUD profile actions: Create, Read, Update, and Delete, this is shown in Listing 2.9. Contained in the `<Entry action="crud_type">..content.. </Entry>` element is a series of 0 or more elements corresponding to the table schema of, in this case, the `AuthUsers` table. It takes zero or more of the table attributes in a row, since not all table attributes are needed for all actions. As an example delete only requires the unique identifier of the table, and updates will only need the unique identifier and the table attribute being updated.

```
1  <xs:simpleType name="crud">
2    <xs:restriction base="xs:string">
3      <xs:enumeration value="create"/>
4      <xs:enumeration value="read"/>
5      <xs:enumeration value="update"/>
6      <xs:enumeration value="delete"/>
7    </xs:restriction>
8  </xs:simpleType>
```

Listing 2.9: sw6ml crud simple type

Documentation

To use the current version of sw6ml, it is essential to know the elements required from the different crud types. While XML schema provides advanced features for dynamic languages, sw6ml in its current version, is a primitive language consisting of simple types and sequences. This is however all that is needed, with a few assertions on the format server side.

Listing 2.10 shows the formal structure of a sw6ml document.

```
<sw6ml>
  <table_element_1>
    <Entry action="crud_type">
      <table_element_1_attribute_1/>
        ...
      <table_element_1_attribute_n/>
    </Entry>
  </table_element_n>
  ...
  <table_element_n>
    <Entry action="crud_type">
      <table_element_n_attribute_1/>
        ...
      <table_element_n_attribute_n/>
    </Entry>
  </table_element>
</sw6ml>
```

Listing 2.10: Root and table elements

Following is a short description of the setup of the `<Entry>` element for each CRUD type:

create creates a new row in the database: All attributes from the database schema is required, if no value exists, use null. See Listing 2.8 for an example.

update Updates a field in a row, required in this order: Unique identifier of the row, attribute to be updated. See Listing 2.11 for an example.

delete Deletes a row in the table: Only the unique identifier is required. See Listing 2.8 for an example.

read Read is only used in XML documents which are sent back from a request to the server, and thus requires no special formatting.

Notice the `<row_attribute type="bar">..</..>` XML attribute: Legitimate types are `string` or `int`, if the data type of the row attribute is a

`string` or any other type requiring apostrophes in an SQL query, use `string`, for anything else use `int`.

```
1  <Entry action="update">
2    <unique_identifier type="bar">foo</unique_identifier>
3    <attribute_to_be_updated type="bar">newValue</
         attribute_to_be_updated>
4  </Entry>
```

Listing 2.11: sw6ml update syntax example

The sw6ml schema can be found in the project repository together with a valid sw6ml document, Full path: `http://code.google.com/p/sw6-2012/source/browse/random_group_stuff/Group_server/xml/sw6_schema.xsd` and `http://code.google.com/p/sw6-2012/source/browse/random_group_stuff/Group_server/xml/sw6_example.xml`

2.7.4 Implementation

In the following section, Savannah's server implementation will be presented.

Input handling

When the server starts, it will execute a method called `listen()` that listens for any connections, see Listing 2.12. Whenever a connection is made to the server, the `listen()` method will create a new `CommunicationThread`. This thread will read the information in the `Socket`'s `InputStream` and depending on the connection type[16] it will deal with it appropriately.

In the following sections we will explain how the different types of connections are processed by the system.

Ping

Figure 2.14 is a model of how a ping is processed by the server. The `Connection` component sends output, in this case a ping event, to the server. In the server this is received by the `IOHandler`, which creates a new

[16] Commit event, request event, or ping

```
1  private void listen() {
2    try {
3      //Initiates a ServerSocket
4      System.out.println("Initiating serversocket !");
5      this.serverSocket = new ServerSocket(Configuration.PORT);
6      System.out.println("Initiation complete");
7
8    }  catch (IOException io) {
9       System.err.println("Could not create ServerSocket -_-");
10   }
11   System.out.println("Starting to listen !");
12   //Loop that continues to look & accept connections
13   //on the ServerSocket
14   while (true) {
15     try {
16       //Making a connections
17       Socket con = this.serverSocket.accept();
18       System.out.println("Connection accepted - Socket: " + con);
19
20       //Saving the connections and a corresponding
21       //DataOutputStream for later use
22       this.connections.put(con, new DataOutputStream(con.getOutputStream()
             ));
23
24       //Starts a new Thread used for communication
25       Thread comThread = new CommunicationThread(con, this.folder);
26       comThread.start();
27
28       }  catch (IOException e) {
29          System.out.println("Could not accept connection !");
30       }
31   }
32 }
```

Listing 2.12: The `listen()` method

`CommunicationThread`. The `CommunicationThread` will then use the `TransmissionHandler` to determine if the input is of the type commit, request, or ping. In this case the input is of the type ping, and it will directly respond to the connection. More information on these classes can be found in Figure A.29, Figure A.30, and Figure A.31.

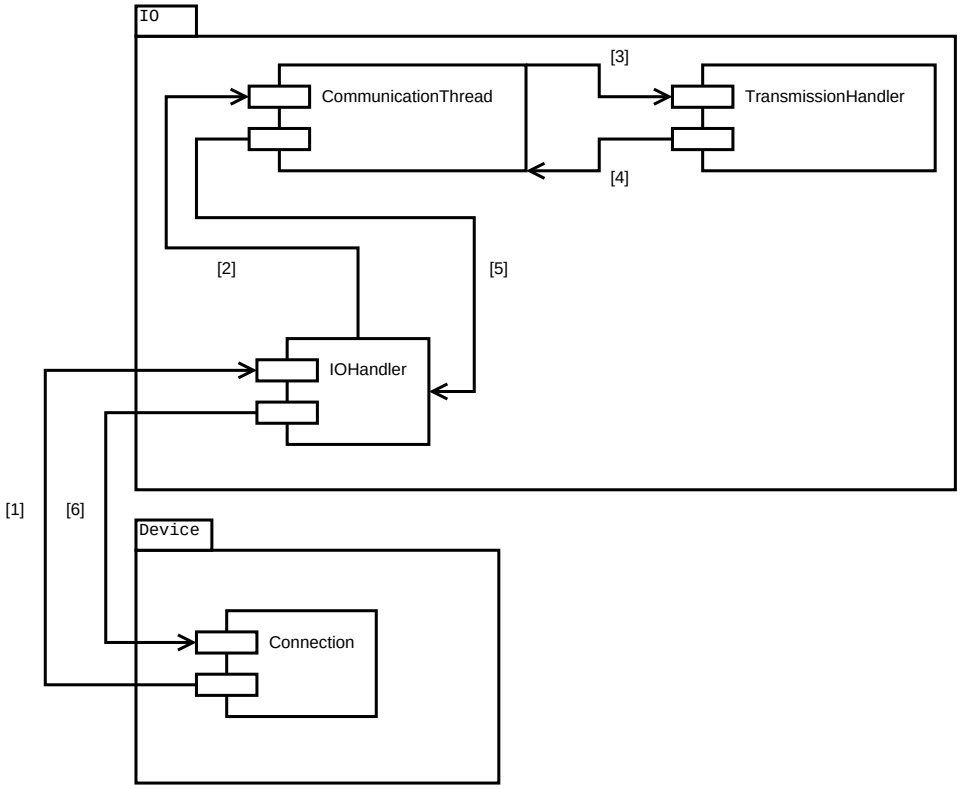


Figure 2.14: A diagram illustrating how a ping is processed.

The reason for this implementation is that Java only supports two types of sockets: stream based[17] and datagram based[18][9]. However, an implementation of ping would require the use of ICMP[19] ping, and since this is not possible we have implemented our own ping function[19].

[17] TCP – `java.net.Socket` and `java.net.ServerSocket`
[18] UDP – `java.net.DatagramSocket` and `java.net.MulticastSocket`
[19] Internet Control Message Protocol

Commit and Request

For a connection of type commit or request, the procedure is almost the same. However, when the `CommunicationThread` has been created and its `TransmissionHandler` has determined the type of the connection, it will as opposed to the ping, create a new event corresponding to the type and send it to the `EventQueue`, see Figure 2.15. When the `EventHandler` is done processing the event, the server responds to the connector. To see how the `EventHandler` processes the given queries see section 2.7.4.

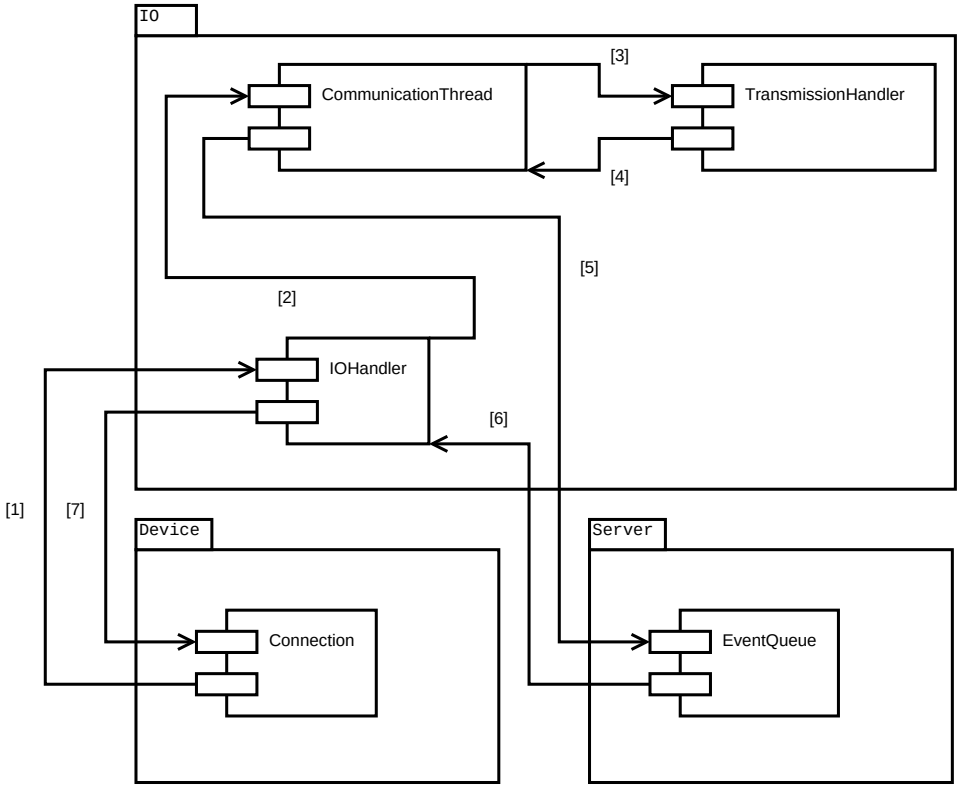


Figure 2.15: A diagram illustrating how a request or a commit event is processed.

Queue and Query handling

The following sections presents an overview of the queue and query implementation of Savannah's server software.

Queue

As mentioned in section 2.7 Savannah is designed around a producer-consumer pattern centered on a FIFO queue, implemented in the `EventQueue` class. The UML diagram in Figure 2.16 shows the event queue implementation and closely related classes. The event queue is a `LinkedList<Event>` object instantiated as a `Queue<Event>`, this instantiation defaults to a FIFO queue through the `queue.add(event)` and `queue.remove()` methods. `EventQueue` is implemented as a singleton to ensure that Savannah never has access to more than one queue. Events are implemented through the `Event` interface that `RequestEvent` and `CommitEvent` implements. Finally the `EventHandler` class is responsible for removing events from the queue and make sure they are processed by the server. It is started as a thread and continuously probes the queue for content. All queue access is synchronized, through the use of the Java `synchronized` keyword, to avoid race conditions.

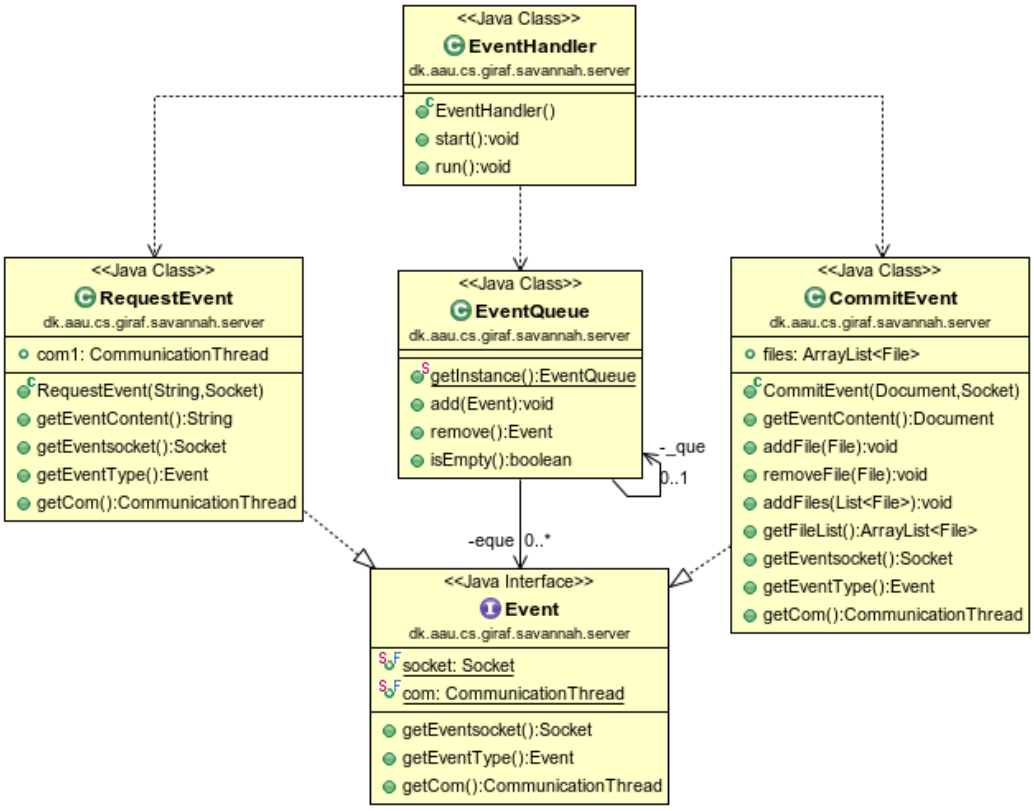


Figure 2.16: EventQueue UML diagram.

An event contains a reference to the socket on which the connection was made and the event content, which is either an XML document or a `String` containing certificates needed to query profiles from the database.

Query creation and handling

The UML diagram in Figure 2.17 shows the `EventHandler` and related classes for building SQL queries, querying the database, and building XML documents. The `Eventhandler` instantiates a `RequestHandler` and a `CommitHandler`, which are responsible for handling request- or commit events. Each of the handlers instatiate a `QueryBuilder` and a `QueryHandler`. The `QueryBuilder` creates SQL queries based on the event content. The queries are forwarded to the `QueryHandler` that interacts with the database and, if the event is a `RequestEvent`, returns a `ResultSet` object. Finally the `Resultset` object

is forwarded to the `XMLBuilder` that builds an sw6ml document, in `String` form, that can be returned to the requesting party.

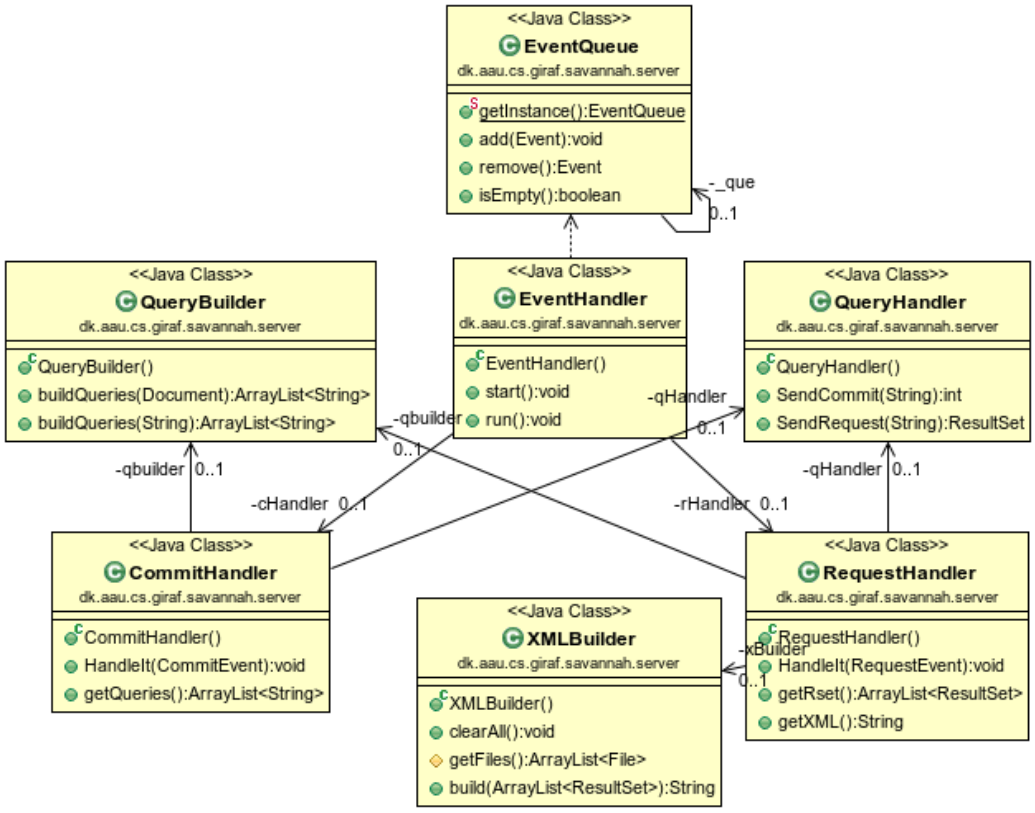


Figure 2.17: Query handling UML diagram.

2.7.5 Test

Testing on the server software has been done through JUnit test cases and big bang system tests, as well as ad hoc testing during implementation. JUnit is a unit testing framework for Java[29]. All tests described in this section are dynamic white-box, and have been done by the authors of the implemented software.

IOHandler Test

JUnit tests for the `IOHandler` have been done as a series of 18 tests.

Tests $1-9$ tests the different methods used for calculations and conversions.

Tests $10-14$ tests the different methods used for generating transmissions.

Tests $15-18$ tests the different methods used for sending generated transmissions.

Testing these methods is not a trivial matter since they all have the access modifier `private`. However, thanks to Ross Burton's article[20], a method for testing these methods has been devised. The way these methods will be tested is by the use of reflection in Java, which gives us the ability to manipulate a methods accessibility, thereby exposing it and allowing us to test it.

There is only one problem with this method, this is that it cannot handle overloaded methods. However, each time the test is run it does somehow, seemingly per random, choose one of the overloaded methods. This means that if you have to test an overloaded method, you may have to run the test several times to get the actual "correct" result.

For tests $1-9$ and tests $11-18$ the yielded results were as expected. However, for test number 10 there was one of the three results that were not as expected, the results can be seen in Table 2.6. Below is a description of how the method tested in test number 10 should work.

makePing(builder, pingSize)

This method generates a ping with the specified size of $1 \leq pingSize \leq 4096$, and adds it to the builder.

Case	Input	Expected result	Actual result
1	32	32	32
2	12	12	12
3	4096	4096	12

Table 2.6: Test10 – Method: `makePing(builder, pingSize)`

This error was not hard to isolate and correct, since it was an error in the logic determining the size of the ping, more specifically – there was a missing

[20] *Subverting Java Access Protection for Unit Testing*[3]

'=' operator. So instead of actually checking for '≤' it was actually checking for '<'. Therefore in case 3, it evaluated $4096 < 4096$ to a case which yielded the value 12, instead of the value 4096.

Queue and Query Test

In the `server` package there is a total of 10 classes, which have been subjected to testing. An overview is shown in Table 2.7. The column labeled "JUnit test cases" shows how many JUnit unit tests have been written for the class.

Class	JUnit test cases	System test
Event	-	Yes
CommitEvent	5	Yes
RequestEvent	-	Yes
EventHandler	-	Yes
EventQueue	5	Yes
QueryBuilder	20	Yes
QueryHandler	-	Yes
RequestHandler	-	Yes
CommitHandler	-	Yes
XMLBuilder	-	Yes

Table 2.7: Queue and query test overview

Unit testing was not found suitable for all classes, due to the interdependency of other classes in the project. As en example testing the `XMLBuilder` with a unit test would require a `ResultSet` object. The easiest way to build this is to get it from the `QueryHandler`, which in turn require well formed SQL queries from the `QueryBuilder`. It was decided that it would be most efficient to test many of the classes during a system test. JUnit has been used to test the building of SQL queries, singleton implementation, and that the queue is in fact FIFO.

The code in Listing 2.13 shows a JUnit test case for the `QueryBuilder`. It verifies that the `QueryBuilder` properly creates an SQL query for inserting an integer value into the `Media` table. Similar test cases exists for `String` values and for SQL queries that update or delete rows. Due to the implementation of the `Querybuilder` we can cover all values and SQL query types with 9 unit tests. The remaining 11 test cases are for request queries.

```
1  public void testCreateIntValue() throws Exception
2  {
3    String xml = "<sw6ml><Media><Entry action=\"create\">" +
4                 "<idMedia type=\"int\">1</idMedia>" +
5                  "</Entry>" +
6                  "</Media></sw6ml>";
7    doc = dom.Dominate(xml);
8    ArrayList<String> one = qbuilder.buildQueries(doc);
9    assertEquals("INSERT INTO Media values(1);", one.get(0) );
10 }
```

Listing 2.13: A JUnit test case

All classes in the `server` and `io` package has one or more corresponding ad hoc testing classes as well. Many of these classes are similiar in nature to the unit tests. They are however not automated and mostly rely on `System.out.println()` debugging output, requiring a human reader. They also include trial and error implementation of unfamiliar technology. In total these ad hoc testing classes totals 764 lines of code.

The big bang system test performed was not documented or organized, but rather a test of the server software at the end of development. It was primarily for verifying our implementation and polishing the software.

CHAPTER 3

RECAPITULATION

3.1 Discussion

In this section we discuss what we have implemented, and what we did not manage in time.

Agile Development

Implementation of Scrum in our project has been successful. We have modified the method to better suit our needs for the project. We skipped the daily scrum meeting because we did not feel that we got enough benefit out of it, we already had sufficient communication. In the beginning, we had sprint backlogs every sprint, but slowly started to phase it out, as we stopped adding assignments to our project backlog, and began to wrap up our work. Our project has been requirement driven, so we have had meetings with our contacts to find out what they need. In the beginning we tried to gather information on how they worked and what it was like working with children with ASD, later the meetings became more focused on the system.

The database

Our database was designed in close cooperation with the Oasis group, which was a great success. It is functional and tested. We have made a lot of small adjustments throughout the project, based on requirements from the

other groups, but after the third sprint we froze the database design, as this was necessary to be able to start implementing the synchronization and web interface.

The Web Interface

The web interface is not completely implemented. We have had to change the priority of the features to implement. The following parts are implemented, tested, and functional:

Add profile This is the most important part of the system, as the capability of adding new users is essential.

Add pictures and sound This was a high priority requirement from the PARROT group.

Add tags Important for sorting pictures and sound.

Profile management Allow assigning of guardians and parents to children. A high priority requirement from the Launcher group.

Edit profile The users need to be able to edit their profile, which is an important feature for a functional system.

Delete profile Allowing the deletion of profiles from the system.

All this is handled without any user authentication, and can be accessed directly from the welcome screen. This is a security issue, but we decided on to focus on features, and postpone security until we are ready for deployment.

We have implemented the possibility to login, but after this, the user is met by a main screen with limited capabilities, and the few available are for testing purposes.

The Servlet code for the different pages could benefit a great deal from refactoring. This is a result of working with a technology in which we had no prior knowledge, and thus learning while doing.

The usability test of the web interface, showed a few critical, and several serious and cosmetic errors. Unfortunately we did this test very late in the process, and we did not manage to correct this.

The Server Software

All development was stopped at the end of the 5^{th} sprint, no matter what condition the software was in. While we have generally attempted to polish the implementation, some errors still exists. Following is a list of known issues and shortcomings of Savannah at the end of the project.

Server file structure Files retrieved by the server will be stored in a single folder, meaning that new files name identical names to existing files, will overwrite the existing files.

Known error in query building Database testing showed us that there had been a misunderstanding in the project group, which means that query building is erroneous. The problem lies in that rows in the `Department` and `Profile` cannot have identical unique identifiers, as they both have foreign key constraints to the `AuthUsers` table. The query builder however, builds a query that extracts information where `Department` and `Profile` are joined, ultimately resulting in the query always returning the empty set.

No synchronization done One of the goals of the project was to synchronize data with the local database, and a .jar file[1] has been created, and tested on a laptop computer, for this purpose. However, Oasis was, due to a still unknown error, unable to include the .jar file in the Oasis library.

SSL communication Communication to and from the server is not in SSL, which at this point is not an issue. SSL communication is only required upon a product release, and Savannah is not ready for release.

3.2 Conclusion

A system to handle synchronization of data from the various GIRAF apps, and administrate this, has been implemented using a combination of Java, Java Servlets, MySQL, HTML, JavaScript, and CSS. Although a full implementation has not been achieved, the key functionality has been implemented.

[1] a Java archive

The GIRAF apps are not able to retrieve any data from the global database, but the only thing needed is to implement the code for this in Oasis.

The web interface features full profile management, but the security issues of being able to have full control without authentication, needs to be addressed. Furthermore the usability test showed various problems in the web interface, shown in Table 2.3. The settings and stats are not implemented, as we chose to lower the priority of this part, and focus on the profile management.

The database is fully implemented, and meets all requirements stated in section 2.5.1

The system is partially implemented, but very open for future work.

3.3 Future work

In this section our suggestions for future development is listed.

Serverside

Of the issues discussed in section 3.1, the most important thing to implement, is the communication between Savannah and Oasis. To do this, it is required to import the ServerConnection.jar file into Oasis. This should, in theory, be enough to enable the communication. Furthermore, the communication protocol, must be changed from TCP/IP to SSL, when the system goes live, as this is a major security issue at the moment. As a product Savannah's server is still in a primitive state, and many hours of continued development could be put into it.

Web Interface

The web interface is the part that has the most future work to be done:

Main screen The major focus should be on implementing a fully featured main screen after login.

Security Only authenticated users should be able to access specific data.

Usability The results of the usability test should be incorporated, to ease the use of the web interface.

Eye candy Color schemes, button layout etc. should be changed to match the theme and layout of the GIRAF system.

Implementing the main screen after login, will make it quite easy to increase the security, this will also enable the implementation to retrieve and edit settings, and generation of stats. Furthermore most of the functionality needed beyond login (edit user, add pictures to the users, assign children to guardians, and vice verse, etc.) are already present, they merely need to be relocated.

3.4 Multi project

Overall the multi project went well. Our cooperation with the other groups has been good. When the other groups had new requirements they contacted the Oasis group who would then tell us.

The multi project meetings have run without too much complication. In the beginning the meetings tended to get too long, but once we got a little more structure on the meetings, they also got shorter and easier to manage. We quickly found a routine for the meetings that made sure that there would always be progress, instead of us just getting stuck on the same topic without taking any decisions.

During the usability test the groups had great cooperation, the only complications were technical issues. Every group participated and helped with both the preparation and execution of the test.

There has not been a lot of complications for our group in regards to the multi project.

APPENDIX A

APPENDIX

A.1 MySQL Code

```
1  CREATE TABLE `04`.`AuthUsers` (
2
3  `certificate` VARCHAR(512) NOT NULL ,
4
5  `idUser` INT NOT NULL AUTO_INCREMENT ,
6
7  `aRole` INT NOT NULL,
8
9  `username` VARCHAR(45) UNIQUE NOT NULL,
10
11 `password` VARCHAR(45) NOT NULL,
12
13 PRIMARY KEY (`certificate`) ,
14
15 UNIQUE INDEX `idUser_UNIQUE` (`idUser` ASC) );
```

Listing A.1: Create AuthUsers

```
1
2   CREATE TABLE `04`.`Apps` (
3
4   `idApp` INT NOT NULL AUTO_INCREMENT,
5
6   `name` VARCHAR(45) NOT NULL ,
7
8   `version` VARCHAR(45) NOT NULL ,
9
10  `icon` VARCHAR(45) NOT NULL,
11
12  `package` VARCHAR(45) NOT NULL,
13
14  `activity` VARCHAR(45) NOT NULL,
15
16  PRIMARY KEY (`idApp`) );
```

Listing A.2: Create Apps

```
1  CREATE TABLE `04`.`Tags` (
2
3    `idTags` INT NOT NULL AUTO_INCREMENT,
4
5    `caption` VARCHAR(45) NOT NULL ,
6
7    PRIMARY KEY (`idTags`) ,
8
9    UNIQUE INDEX `caption_UNIQUE` (`caption` ASC) );
```

Listing A.3: Create Tags

```
1  CREATE TABLE `04`.`Profile` (
2
3    `idProfile` INT NOT NULL ,
4
5    `firstname` VARCHAR(45) NOT NULL ,
6
7    `surname` VARCHAR(45) NOT NULL ,
8
9    `middlename` VARCHAR(45) NULL ,
10
11   `pRole` INT NOT NULL ,
12
13   `phone` BIGINT NOT NULL ,
14
15   `picture` VARCHAR(45) NULL ,
16
17   `settings` BLOB NULL ,
18
19   FOREIGN KEY (`idProfile` )
20
21   REFERENCES `04`.`AuthUsers` (`idUser` ) on delete cascade,
22
23   PRIMARY KEY (`idProfile`) );
```

Listing A.4: Create Profile

```
1  CREATE TABLE `04`.`ListOfApps` (
2
3    `idApp` INT NOT NULL ,
4
5    `idProfile` INT NOT NULL ,
6
7    `setting` BLOB,
8
9    `stats` BLOB,
10
11   FOREIGN KEY (`idApp` )
12
13   REFERENCES `04`.`Apps` (`idApp` ),
14
15   FOREIGN KEY (`idProfile` )
16
17   REFERENCES `04`.`Profile` (`idProfile` ) ON DELETE CASCADE,
18
19   PRIMARY KEY (`idApp`,`idProfile`) );
```

Listing A.5: Create ListOfApps

```
1  CREATE TABLE `04`.`Department` (
2
3    `idDepartment` INT NOT NULL ,
4
5    `name` VARCHAR(45) NOT NULL ,
6
7    `address` VARCHAR(45) NOT NULL ,
8
9    `phone` BIGINT NOT NULL ,
10
11   `email` VARCHAR(45) NOT NULL ,
12
13   FOREIGN KEY (`idDepartment` )
14
15   REFERENCES `04`.`AuthUsers` (`idUser` ),
16
17   PRIMARY KEY (`idDepartment`) );
```

Listing A.6: Create Department

```
1  CREATE TABLE `04`.`HasDepartment` (
2
3    `idProfile` INT NOT NULL ,
4
5    `idDepartment` INT NOT NULL ,
6
7    FOREIGN KEY (`idProfile` )
8
9    REFERENCES `04`.`Profile` (`idProfile` ) ON DELETE CASCADE,
10
11   FOREIGN KEY (`idDepartment` )
12
13   REFERENCES `04`.`Department` (`idDepartment` ),
14
15
16   PRIMARY KEY (`idProfile`, `idDepartment`) );
```

Listing A.7: Create HasDepartment

```
1  CREATE TABLE `04`.`HasGuardian` (
2
3    `idGuardian` INT NOT NULL ,
4
5    `idChild` INT NOT NULL ,
6
7    FOREIGN KEY (`idGuardian` )
8
9    REFERENCES `04`.`Profile` (`idProfile` ) on delete cascade,
10
11   FOREIGN KEY (`idChild` )
12
13   REFERENCES `04`.`Profile` (`idProfile` ) on delete cascade,
14
15   PRIMARY KEY (`idGuardian`,`idChild`) );
```

Listing A.8: Create HasGuardian

```
1  CREATE TABLE `04`.`HasSubDepartment` (
2
3    `idDepartment` INT NOT NULL ,
4
5    `idSubDepartment` INT NOT NULL ,
6
7    FOREIGN KEY (`idDepartment` )
8
9    REFERENCES `04`.`Department` (`idDepartment` ),
10
11   FOREIGN KEY (`idSubDepartment` )
12
13   REFERENCES `04`.`Department` (`idDepartment` ),
14
15   PRIMARY KEY (`idDepartment`, `idSubDepartment`) );
```

Listing A.9: Create HasSubDepartment

```
1  CREATE TABLE `04`.`Media` (
2
3    `idMedia` INT NOT NULL AUTO_INCREMENT,
4
5    `mPath` VARCHAR(45) NOT NULL ,
6
7    `name` VARCHAR(45) NOT NULL ,
8
9    `mPublic` TINYINT NOT NULL ,
10
11   `mType` VARCHAR(45) NOT NULL ,
12
13   `ownerID` INT NOT NULL ,
14
15   FOREIGN KEY (`OwnerID` )
16
17   REFERENCES `04`.`AuthUsers` (`idUser` ) ON DELETE CASCADE,
18
19   PRIMARY KEY (`idMedia`) );
```

Listing A.10: Create Media

```
1  CREATE TABLE `04`.`HasTag` (
2
3    `idMedia` INT NOT NULL ,
4
5    `idTag` INT NOT NULL ,
6
7    FOREIGN KEY (`idMedia` )
8
9    REFERENCES `04`.`Media` (`idMedia` ) on delete cascade,
10
11   FOREIGN KEY (`idTag` )
12
13   REFERENCES `04`.`Tags` (`idTags` ),
14
15   PRIMARY KEY (`idMedia`, `idTag`) );
```

Listing A.11: Create HasTag

```
1  CREATE TABLE `04`.`HasLink` (
2
3    `idMedia` INT NOT NULL ,
4
5    `idSubMedia` INT NOT NULL ,
6
7    FOREIGN KEY (`idMedia` )
8
9    REFERENCES `04`.`Media` (`idMedia` ) on delete cascade,
10
11   FOREIGN KEY (`idSubMedia` )
12
13   REFERENCES `04`.`Media` (`idMedia` ) on delete cascade,
14
15   PRIMARY KEY (`idMedia`, `idSubMedia`) );
```

Listing A.12: Create HasLink

```
1  CREATE TABLE `04`.`MediaProfileAccess` (
2
3    `idProfile` INT NOT NULL ,
4
5    `idMedia` INT NOT NULL ,
6
7    FOREIGN KEY (`idProfile` )
8
9    REFERENCES `04`.`Profile` (`idProfile` ) ON DELETE CASCADE,
10
11   FOREIGN KEY (`idMedia` )
12
13   REFERENCES `04`.`Media` (`idMedia` ) ON DELETE CASCADE,
14
15   PRIMARY KEY (`idProfile`, `idMedia`) );
```

Listing A.13: Create MediaProfileAccess

```
1  CREATE TABLE `04`.`MediaDepartmentAccess` (
2
3    `idDepartment` INT NOT NULL ,
4
5    `idMedia` INT NOT NULL ,
6
7    FOREIGN KEY (`idDepartment` )
8
9    REFERENCES `04`.`Department` (`idDepartment` ),
10
11   FOREIGN KEY (`idMedia` )
12
13   REFERENCES `04`.`Media` (`idMedia` ),
14
15   PRIMARY KEY (`idDepartment`, `idMedia`) );
```

Listing A.14: Create MediaDepartmentAccess

A.2 Database Draft

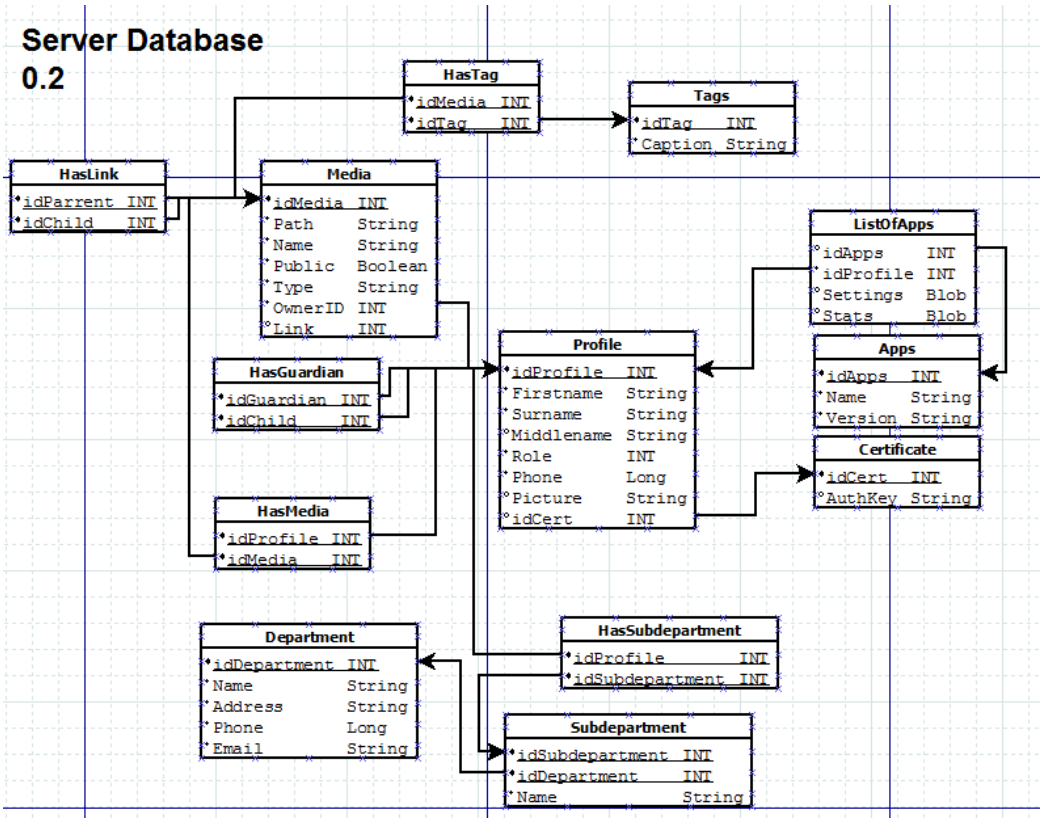


Figure A.1: First draft

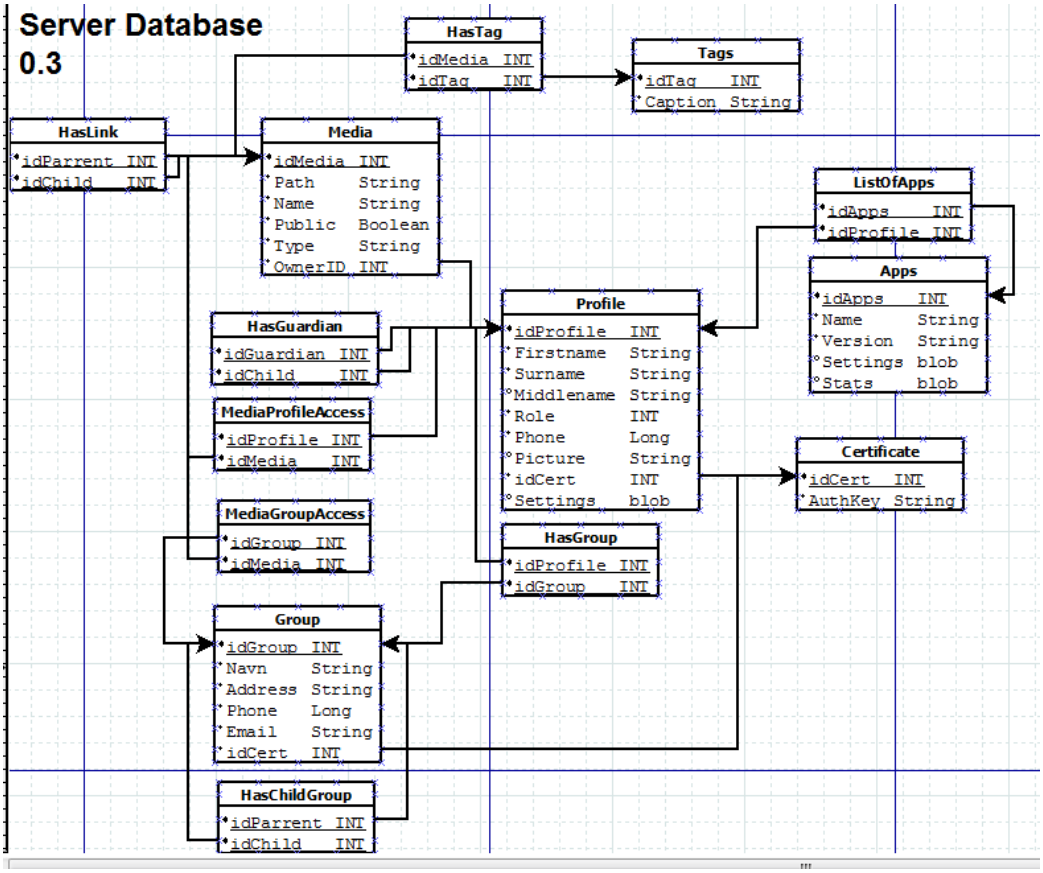


Figure A.2: Second draft

Identifier	**TD0002**
Features to be tested	Database, read data
Approach	Insert a few “correct” entries Fill database with dummy entries Retrieve a “correct” entry
Test case identification	• Insert “correct” entry - Test Case ID #0012 • Retrieve “correct” entry - Test Case ID #0013
Pass/fail criteria	Pass: Retrieve the expected entry Fail: Retrieve any other entry

Figure A.3: TD0002

Identifier	**TD0003**
Features to be tested	Database, update data
Approach	Edit an entry's Firstname
Test case identification	• Retrieve a entry - Test Case ID #0013 • Edit the Firstname - Test Case ID #0014 • Retrieve the entry again, to check if change is stored - Test Case ID #0013
Pass/fail criteria	Pass: The Firstname of the entry has to be stored Fail: The new Firstname is not stored

Figure A.4: TD0002

Identifier	**TD0004**
Features to be tested	Database, delete entry
Approach	Delete an entry
Test case identification	• Retrieve a entry - Test Case ID #0013 • Delete the entry - Test Case ID #0015 • Try to retrieve the entry again - Test Case ID #0013
Pass/fail criteria	Pass: The entry is able to be retrieved, after the deletion, it should not be Fail: The entry is able to be retrieved after deletion

Figure A.5: TD0002

Identifiers	**Test Case ID #0013**
Test item	Retrieve “correct” entry
Input specification	Retrieval of entry
Output specification	Correct entry retrieved Wrong entry retireved No entry retrieved
Environmental needs	A MySQL database, with entries
Special procedural requirements	–
Intercase dependencies	Test Case #0001-0012

Figure A.6: Testcase #13

Identifiers	**Test Case ID #0014**
Test item	Edit the Firstname
Input specification	Change name Change string to int NULL
Output specification	Correct update Error
Environmental needs	A MySQL database, with entries
Special procedural requirements	–
Intercase dependencies	Test Case #0001-0012 Test Case ID #0013

Figure A.7: Testcase #14

Identifiers	**Test Case ID #0015**
Test item	Delete an entry
Input specification	Delete an entry in database Delete an entry NOT in the database Deletion of a entry which a foreign key point to
Output specification	Correct deletion Error Foreign key constraint fails
Environmental needs	A MySQL database, with entries
Special procedural requirements	–
Intercase dependencies	Test Case #0001-0012 Test Case ID #0013

Figure A.8: Testcase #15

A.3 Database Test

A.4 ER Diagram

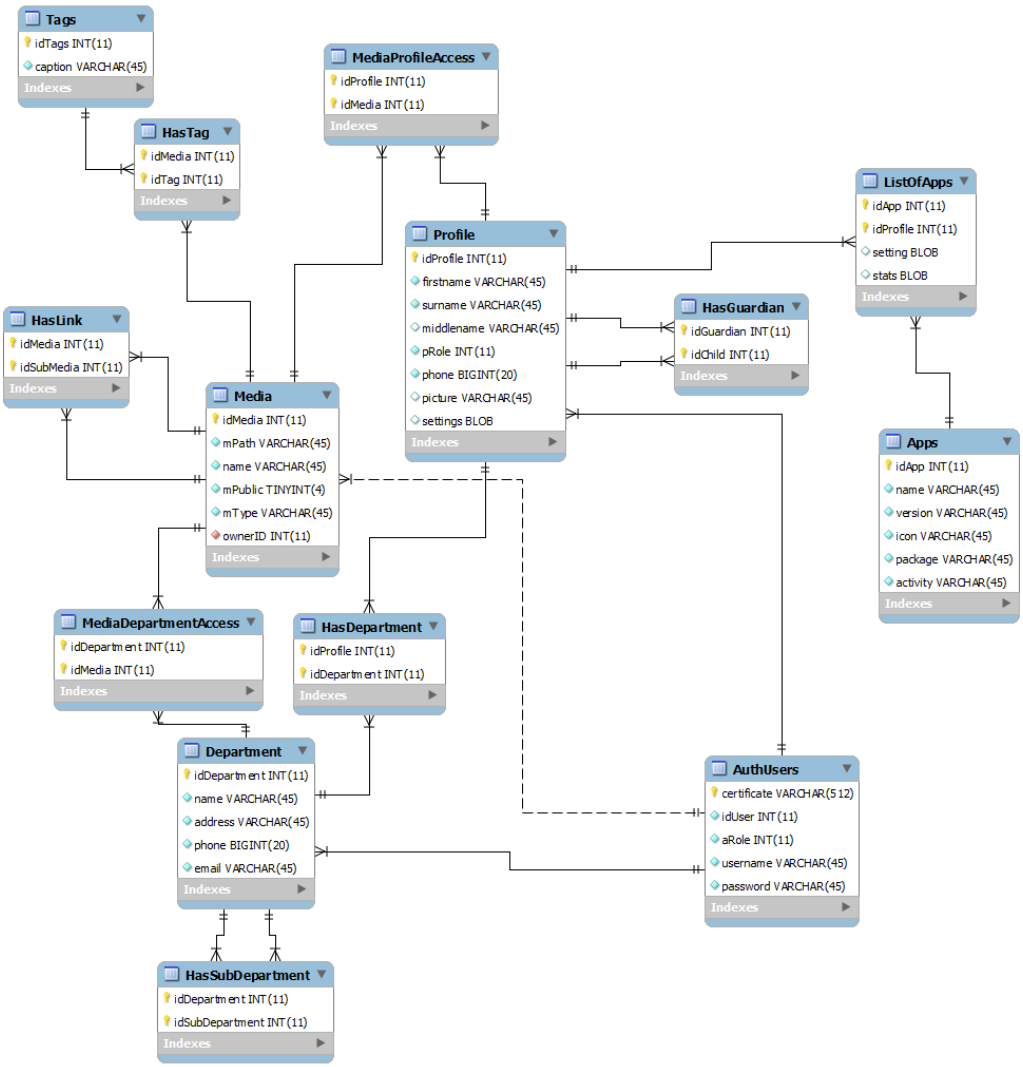


Figure A.9: The ER diagram as MySQL Workbench generates it

A.5 Web Interface Mockups

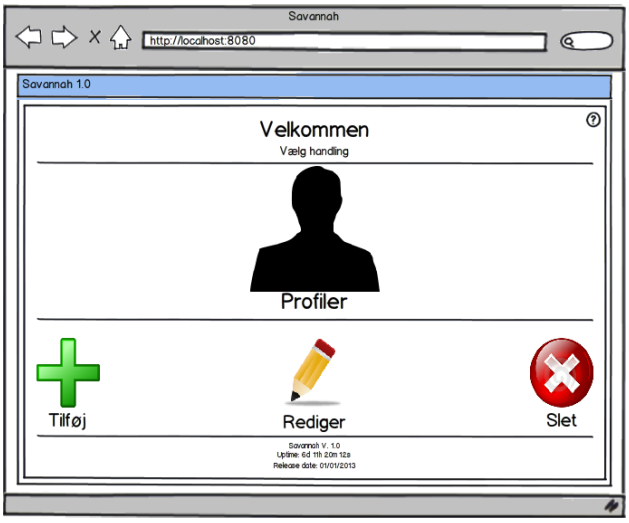


Figure A.10: Welcome screen

Figure A.11: Add something, not logged in

Figure A.12: Delete something, not logged in

Figure A.13: Edit something, not logged in

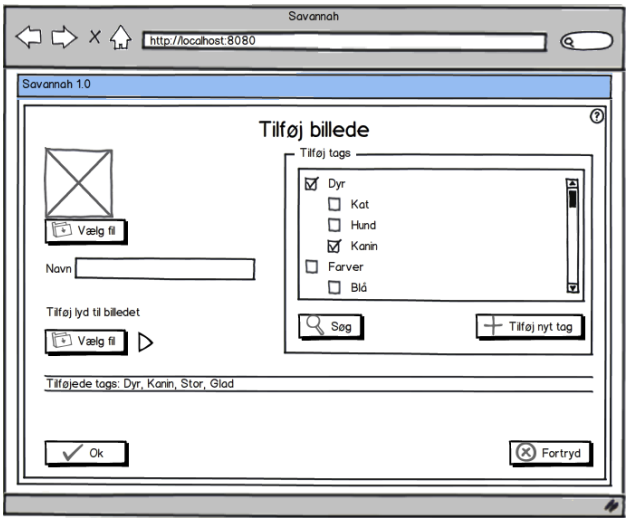


Figure A.14: Add picture, not logged in

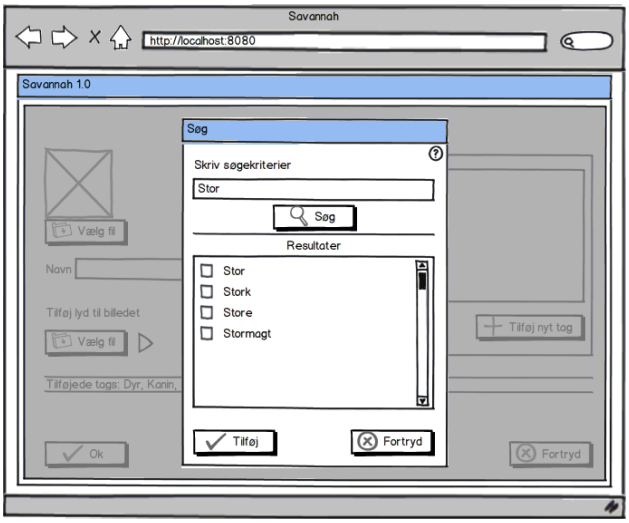


Figure A.15: Add picture tags

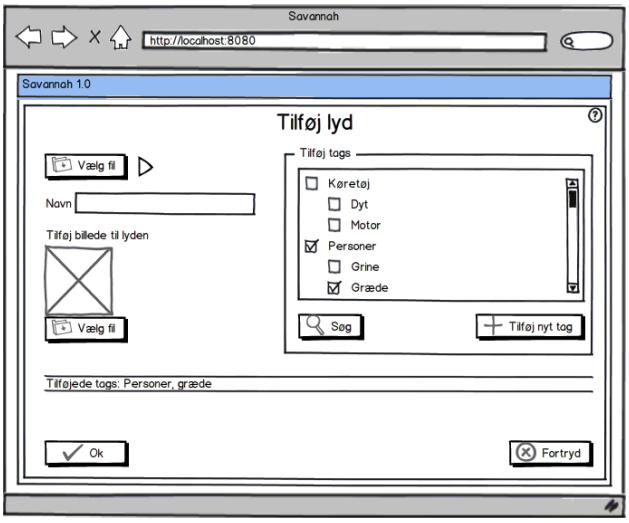


Figure A.16: Add sound, not logged in

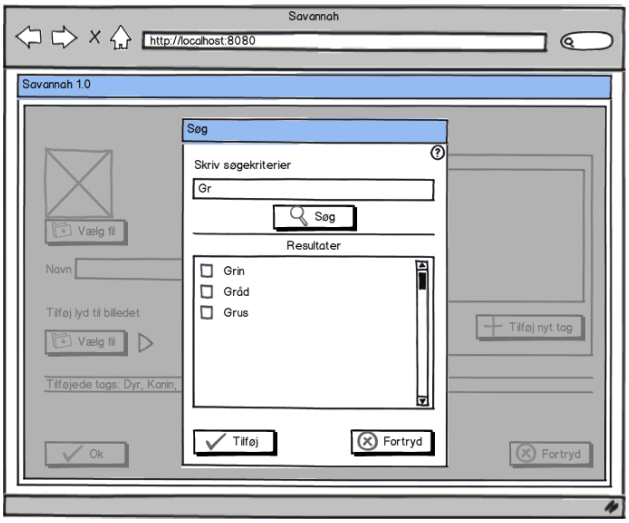


Figure A.17: Add sound tags

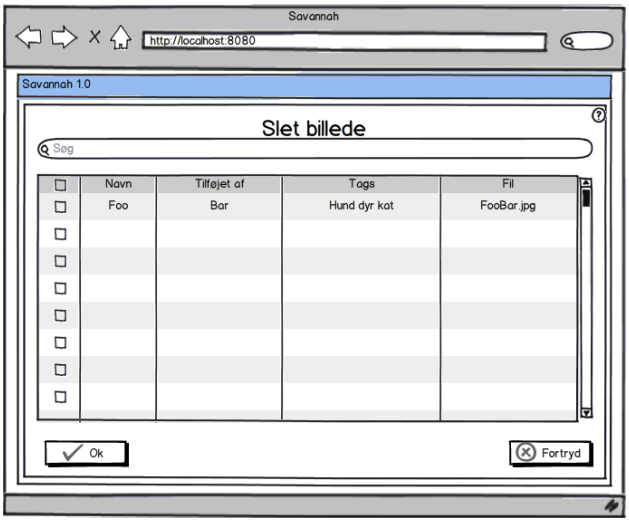


Figure A.18: Delete something, not logged in

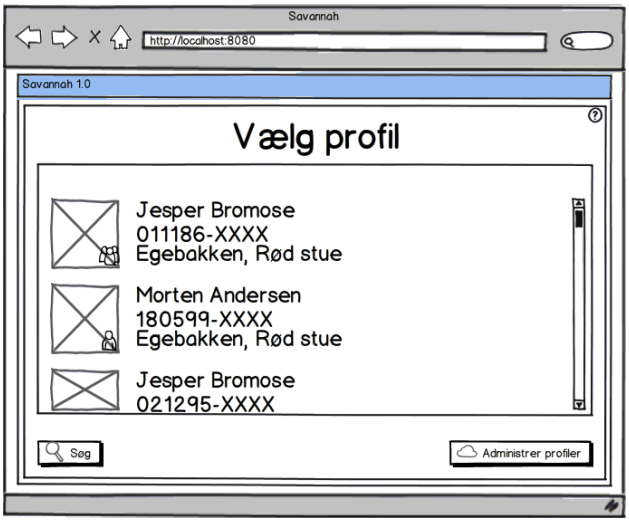


Figure A.19: Select profile to login

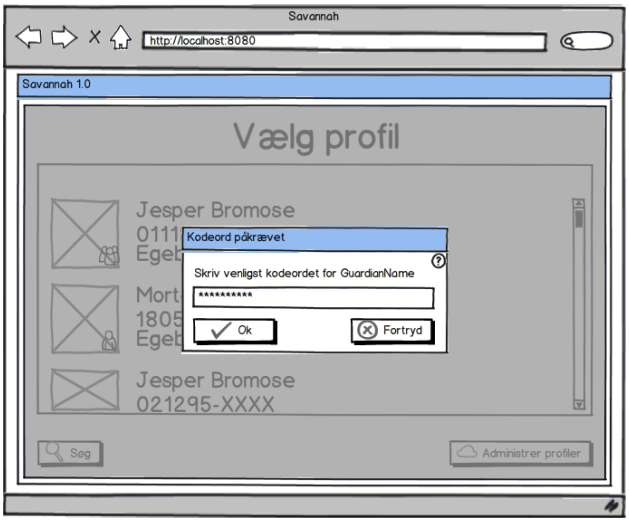


Figure A.20: Login screen

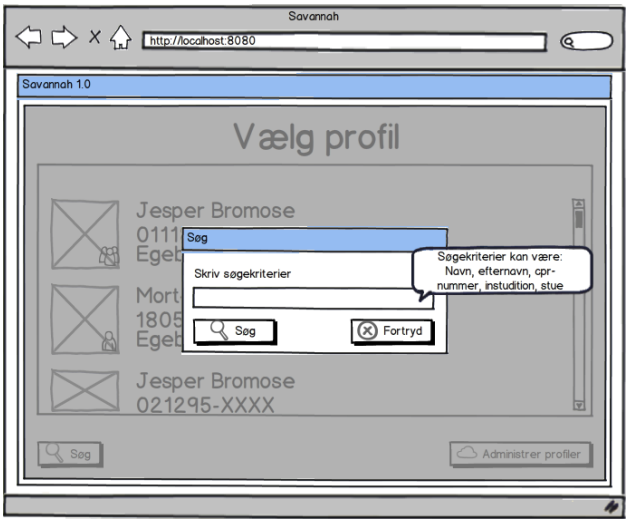


Figure A.21: Search for profile

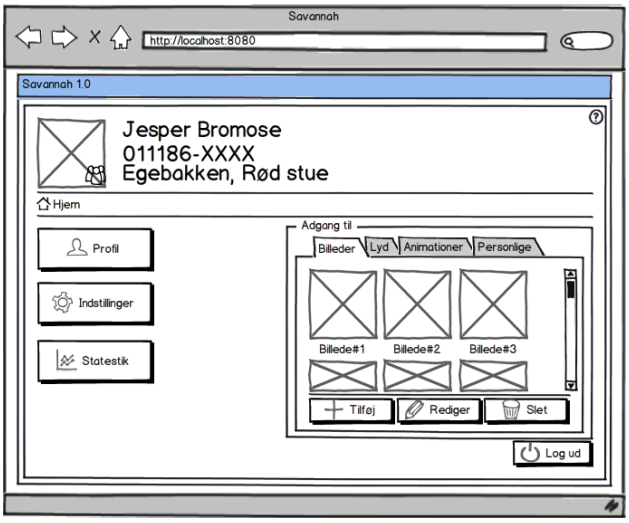


Figure A.22: Main screen

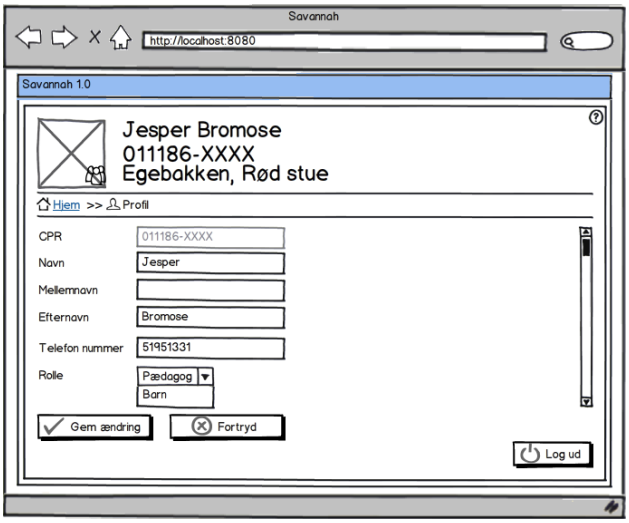


Figure A.23: Edit profile

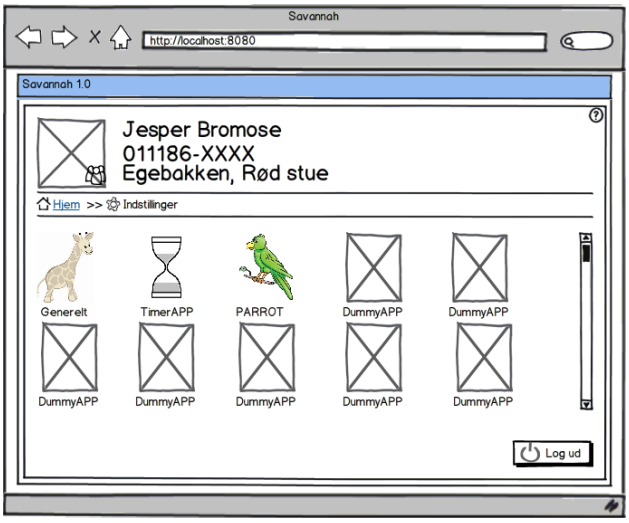


Figure A.24: Settings

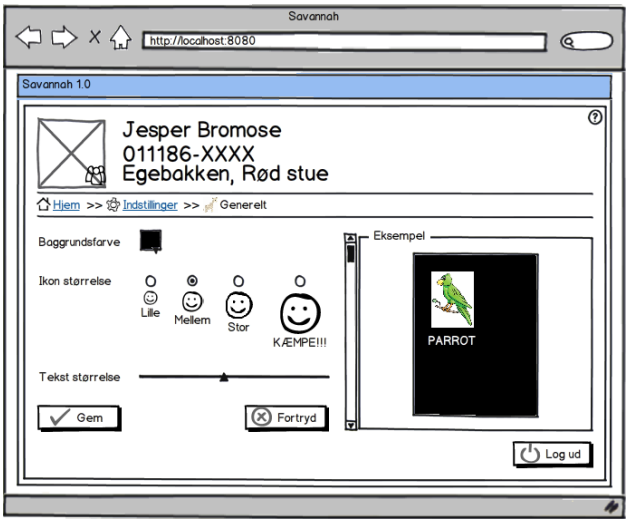


Figure A.25: General settings

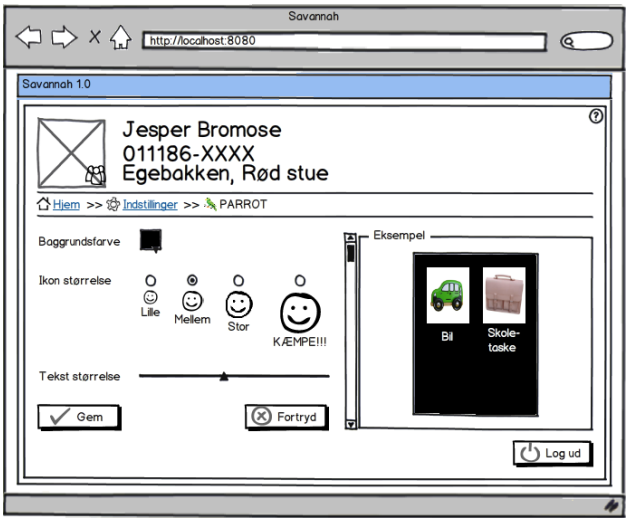


Figure A.26: PARROT settings

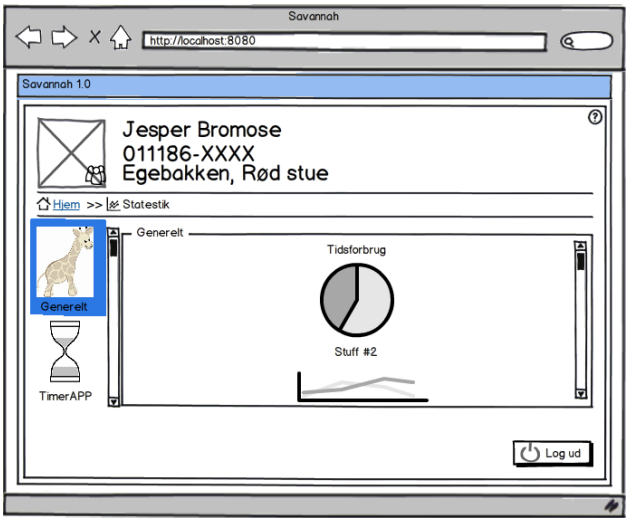


Figure A.27: Stats

Figure A.28: Administrate profiles

A.6 Web Interface Code

```
1 <input type=Text/Password name="some_name" />
```

Listing A.15: HTML tag for Text/Password

```
1 <input type=RadioButton name="some_name" value= name="
    some_value">Caption
```

Listing A.16: HTML tag for RadioButton

```
1 <input type=CheckBox name="some_name0" value="some_value0"/>
    Some_Caption0
2 <input type=CheckBox name="some_name1" value="some_value1"/>
    Some_Caption1
```

Listing A.17: HTML tag for RadioButton

```
<select name="some_name">
<option value="some_value0">Some_Caption0</option>
<option value="some_value1">Some_Caption1</option>
<option value="some_value2">Some_Caption2</option>
<option value="some_value3">Some_Caption3</option>
</select>
```

Listing A.18: HTML tag for DropDown

```
<select name="some_name" size="4" multiple="multiple">
<option value="some_value0">Some_Caption0</option>
<option value="some_value1">Some_Caption1</option>
<option value="some_value2">Some_Caption2</option>
<option value="some_value3">Some_Caption3</option>
</select>
```

Listing A.19: HTML tag for MultipleSelect

A.7 Class Diagrams

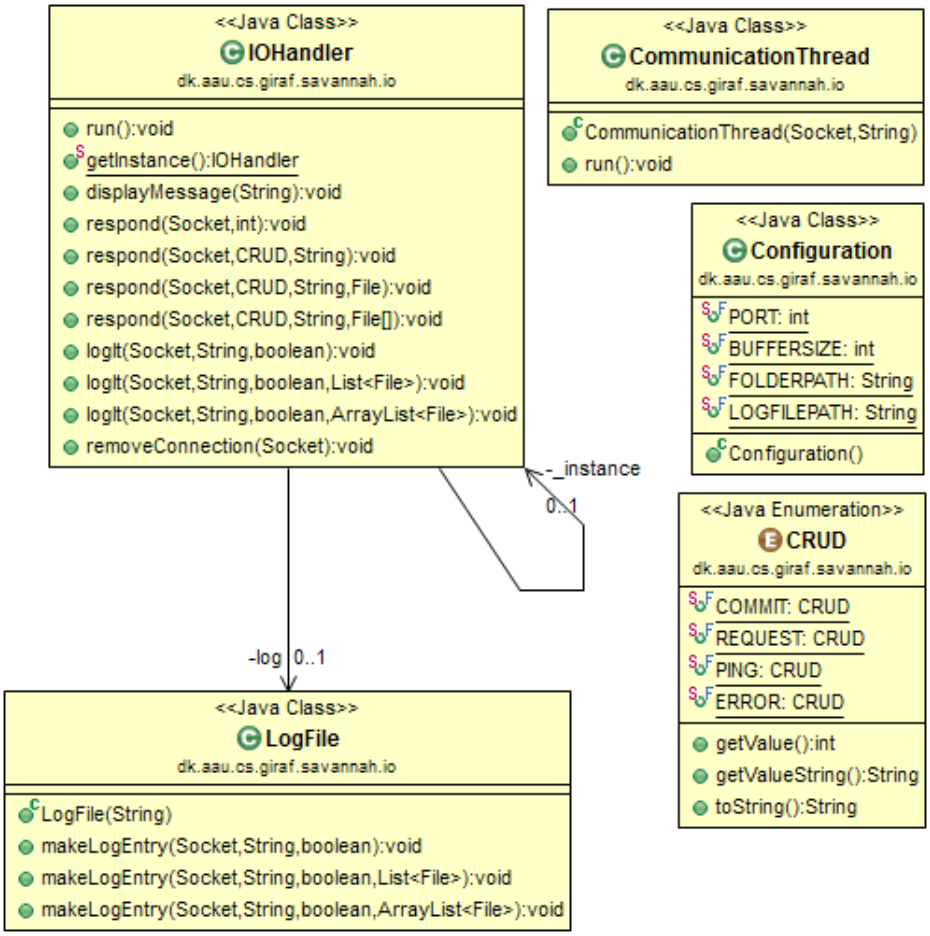


Figure A.29: Class-diagram of the `IOHandler`

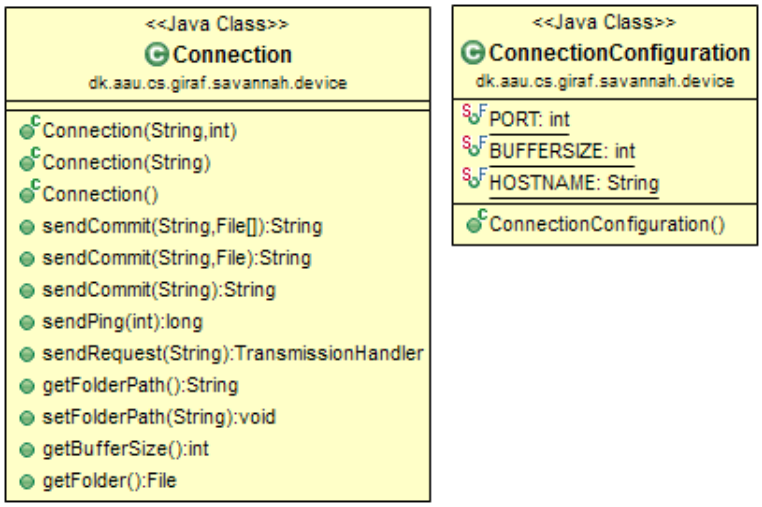


Figure A.30: Class-diagram of the `Connection`

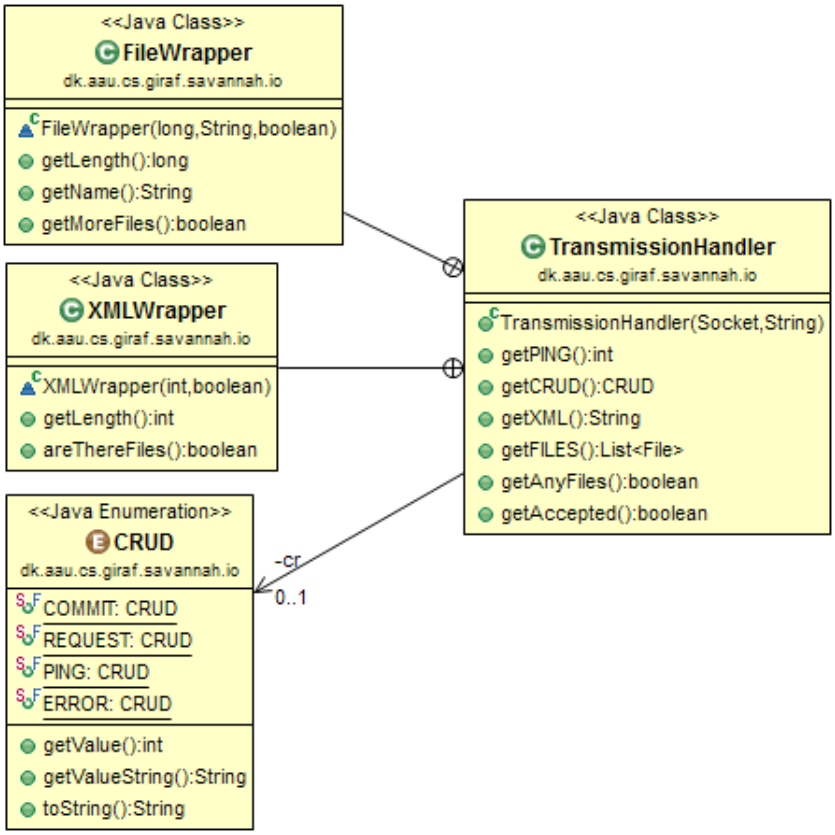


Figure A.31: Class-diagram of the `TransmissionHandler`

A.8 Notes from Interview with Contact Person

Interview 1

This is notes from an interview with Mette Als Andreasen, an educator at Birken in Langholt, Denmark.

Naar tiden loeber ud (kristian har tage et billede):
Faerdig - symbol
Gaa til skema - symbol
Taget fra boardmaker

Kunne vaere godt hvis man kunne saette egne billeder ind som start/stop symboler.

Roed farve = nej, stop, aflyst.

De har saadan et ur paa 60 minutter hvor tid tilbage er markeret med roed, og saa bipper den lige kort naar den er faerdig.
Det ville vaere fint hvis de kunne bruge sort/hvid til dem der ikke kan haandtere farver, men ogsaa kan vaelge farver.

Stop-ur:
en fast timer paa 60 minutter + en customizable som ikke ser helt magen til ud, som f.eks, kan vaere paa 5, 10 eller 15 minutter for en hel cirkel.

timeglas:
skift farve paa timeglassene, men ikke noedvendigvis goere dem stoerre. Kombinere med mere/mindre sand. Eventuelt kombinere med et lille digitalt ur, til dem der har brug for det, skal kunne slaaes til og fra.

Dags-plan:
ikke saerlig relevant til de helt smaa og ikke saerligt velfungerende boern. Men kunne vaere rigtig godt til de lidt aeldre.
En plan gaar oppefra og ned, og hvis der saa skal specificeres noget ud til aktiviteterne, saa er det fra venstre mod hoejre ud fra det nedadgaaende skema.

Til parrot:
Godt med rigtige billeder af tingene, som paedagogerne selv kan tage, eventuelt ogsaa af aktiviteter, saa pedagogerne kan have billeder af aktiviter som de kan liste efter skeamet.

Der var mange skemaer rundt omkring, og der henviser det sidste billede i raekken til naeste skema, som haenger f.eks. paa badevaerelset eller i garderoben.

Interview 2

SUPER RELEVANT: Et billede paa skaermen af den aktivitet de er i gang med, mens uret koerer. Slaaes til og fra.

SUPER RELEVANT: Baade sand-ur og time timer paa samme tid.

Faerdig-skaermen: Mulighed for to billeder: faerdig + gaa-til-skema Kunne

vaere fint hvis der var mulighed for at indspille en lyd/tale naar faerdig-billedet vises. Defineres nok i vores settings-app.

Farverne paa de pre-definerede timere er maaske ikke saa vigtige. 10 min - roed 3 min - gul 1 min - blaa 5 min - blaa 30 sek - sort 30 min - sort (Timere helt ned paa sekund-basis)

Maaske mulighed for forskellige stoerrelser af ure, men det er ikke kritisk.

Progress-bar er ikke saa relevant, saa vi behoever ikke have den med.

A.9 Usability Test

Invitation

Invitation til deltagelse i test af GIRAF

- En Android applikation bygget til børn med autisme

Kære ekspert

Vi vil gerne inviterer dig til at deltage i den første brugervenligheds test af GIRAF, en Android applikation bygget til børn med autisme. Formålet med denne test vil være at undersøge hvor brugervenlig applikationen er og hvor nemt eller svært det er at bruge den. Derfor er det helt fint hvis du aldrig har set eller hørt om denne applikation før nu, da vi gerne vil observerer, hvordan første gangs brugere så vel som brugere med kendskab til applikationen, har det med denne applikationen.

Bemærk venligst at vi er ikke tester din kendskab til applikationen eller evner med en tablet, men derimod om GIRAF applikationen er nem at bruge, vi har kun interesse i at kende til de svagheder der ville være i applikationen. Dette betyder også at du ikke kan give nogle forkerte svar, da du er eksperten.

Derfor vil vi gerne inviterer dig ud i vores brugervenligheds laboratorie, hvor vi kan studere din brug af applikationen. Under brugervenligheds testen vil du blive givet en række opgaver, som skal udføres. Yderligere vil du blive bedt om at tænke højt og fortælle alle tanker, indtryk og valg du tager ved brug af applikationen under testen. Under testen af applikationen vil der blive optaget både video og lyd, til at studere testen senere.

Dagen kommer til at bestå af:

Vi vil meget gerne høre fra dig hvis du har lyst og tid til at deltage i denne brugervenligheds test, den 22/5 - 2012, på Aalborg Universitet.
For at vide hvornår på dagen du kan komme vil vi gerne, at du går ind på denne side (http://www.doodle.com/d2h6swgbtsdf6z2b) skriver dit navn og vælger det tidspunkt på dagen du helst vil komme, dette er svar nok for at vi ved du gerne vil komme.

Kommentarer og spørgsmål kan sendes retur til den mail invitationen kom fra.

På forhånd tak,
Android projektet
Software 6. semester
Aalborg Universitet
Selma Lagerlöfs vej 300, 9220 Aalborg

AALBORG UNIVERSITY

Figure A.32: Invitation sent to the test persons of the usability test.

Briefing

Briefing

Goddag og velkommen til denne brugervenlighedsundersøgelse.

Vi vil gerne starte med at takke dig for, at du vil hjælpe os med at gennemføre denne brugervenlighedsundersøgelse. Vi læser op fra dette dokument for at sikre os, at alle personer som deltager i vores studie for samme introduktion. Hvis du har spørgsmål undervejs, er du naturligvis meget velkommen til at stille disse spørgsmål.

Vi har i dette semester bygget et system til Android til at hjælpe børn med autisme og deres pædagoger og forældre, og det er nu nået til et stadie hvor vi gerne vil teste systemet. Denne test handler udelukkende om at finde problemer og mangler i systemet, og ikke om at teste jeres viden af systemet, så alle tanker I må have om produktet vil vi meget gerne høre.

Før vi starter første del af testen, vil jeg bede dig om at underskrive denne samtykkeerklæring for at sikre, at du er indforstået med rammerne for studiet. Derudover skal du også svare på et demografisk spørgeskema inden testen går i gang.

Testen består af fire dele:

- Test af applikationer (20 min)
- De-briefing og spørgeskema (5 min)
- Test af Administrations applikation og web applikation (20 min)
- De-briefing og spørgeskema (5 min)

Undervejs vil der være en pause.

I de to tests vil du blive stillet en række opgaver som du skal løse. Læs opgaveformuleringen grundigt og fortæl så test hjælperen hvad du mener opgaven går ud på. Derefter skal du forsøge at løse opgaven så godt som muligt. Opgaverne skal løses i den rækkefølge de står således at du starter med opgave 1 og arbejder dig ned af.

Det er meningen at du skal tænke højt mens du løser opgaverne. Dvs. at du siger hvad du har tænkt dig at gøre for at løse opgaven, hvilke ting du synes virker uklare eller komplicerede og hvordan du tror systemet virker. For eksempel vil det være godt hvis du nævner hvad du forventer en knap gør inden du trykker på den.

Når testen er færdig vil der være nogle afsluttende spørgsmål som du skal besvare omkring hvordan du synes testen er forløbet og hvad din opfattelse af systemet er.

Questionnaires

Test Peron 1 - Questionnaire

Usabilitytest - Spørgeskema

1. Hvilket køn er du?

__ Kvinde X Mand

2. Hvor erfaren vurderer du at du selv er med computerer? (vælg en)

__ Meget begrænset erfaring
__ Lettere erfaren
X Forholdsvis erfaren
__ Meget erfaren

3. Har du brugt en tablet før (f.eks. en iPad)?

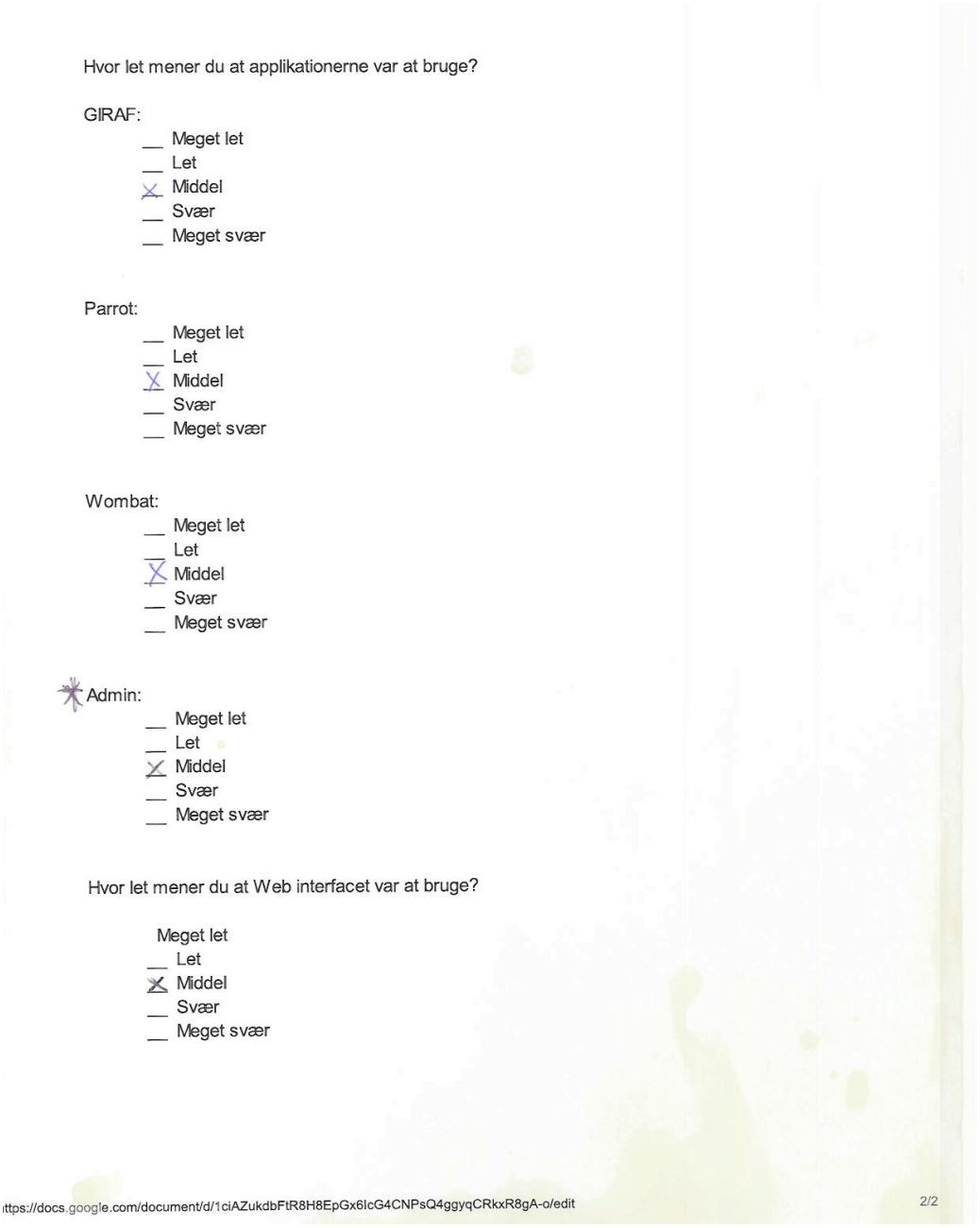
X Ja __ Nej

4. Tror du at en tablet med de rigtige programmer vil kunne forbedre din arbejdsgang?

X Ja __ Nej

5. Tror du at en tablet med de rigtige programmer vil kunne forbedre børnenes hverdag?

X Ja __ Nej



Hvor let mener du at applikationerne var at bruge?

GIRAF:
- __ Meget let
- __ Let
- X Middel
- __ Svær
- __ Meget svær

Parrot:
- __ Meget let
- __ Let
- X Middel
- __ Svær
- __ Meget svær

Wombat:
- __ Meget let
- __ Let
- X Middel
- __ Svær
- __ Meget svær

Admin:
- __ Meget let
- __ Let
- X Middel
- __ Svær
- __ Meget svær

Hvor let mener du at Web interfacet var at bruge?

Meget let
- __ Let
- X Middel
- __ Svær
- __ Meget svær

https://docs.google.com/document/d/1ciAZukdbFtR8H8EpGx6lcG4CNPsQ4ggyqCRkxR8gA-o/edit 2/2

Test Peron 2 - Questionnaire

Usabilitytest - Spørgeskema

1. Hvilket køn er du?

X Kvinde __ Mand

2. Hvor erfaren vurderer du at du selv er med computerer? (vælg en)

__ Meget begrænset erfaring
X Lettere erfaren
__ Forholdsvis erfaren
__ Meget erfaren

3. Har du brugt en tablet før (f.eks. en iPad)?

X Ja __ Nej

4. Tror du at en tablet med de rigtige programmer vil kunne forbedre din arbejdsgang?

X Ja __ Nej

5. Tror du at en tablet med de rigtige programmer vil kunne forbedre børnenes hverdag?

X Ja __ Nej

Hvor let mener du at applikationerne var at bruge?

GIRAF:

__ Meget let
__ Let
X Middel
__ Svær
__ Meget svær

Parrot:

__ Meget let
__ Let
X Middel
__ Svær
__ Meget svær

Wombat:

__ Meget let
__ Let
X Middel
__ Svær
__ Meget svær

Admin:

__ Meget let
__ Let
X Middel
__ Svær
__ Meget svær

Hvor let mener du at Web interfacet var at bruge?

Meget let
__ Let
X Middel
__ Svær
__ Meget svær

Test Peron 3 - Questionnaire

Usabilitytest - Spørgeskema

1. Hvilket køn er du?

X Kvinde __ Mand

2. Hvor erfaren vurderer du at du selv er med computerer? (vælg en)

__ Meget begrænset erfaring
__ Lettere erfaren
X Forholdsvis erfaren
__ Meget erfaren

3. Har du brugt en tablet før (f.eks. en iPad)?

__ Ja X Nej

4. Tror du at en tablet med de rigtige programmer vil kunne forbedre din arbejdsgang?

X Ja __ Nej

5. Tror du at en tablet med de rigtige programmer vil kunne forbedre børnenes hverdag?

X Ja __ Nej

Hvor let mener du at applikationerne var at bruge?

GIRAF:
- __ Meget let
- __ Let
- __ Middel
- X Svær
- __ Meget svær

Parrot:
- __ Meget let
- __ Let
- __ Middel
- X Svær
- __ Meget svær

Wombat:
- __ Meget let
- X Let
- __ Middel
- __ Svær
- __ Meget svær

Admin:
- __ Meget let
- __ Let
- X Middel
- __ Svær
- __ Meget svær

Hvor let mener du at Web interfacet var at bruge?

Meget let
- __ Let
- X Middel
- __ Svær
- __ Meget svær

Test Peron 4 - Questionnaire

Usabilitytest - Spørgeskema

1. Hvilket køn er du?

X Kvinde __ Mand

2. Hvor erfaren vurderer du at du selv er med computerer? (vælg en)

__ Meget begrænset erfaring
__ Lettere erfaren
X Forholdsvis erfaren
__ Meget erfaren

3. Har du brugt en tablet før (f.eks. en iPad)?

X Ja __ Nej

4. Tror du at en tablet med de rigtige programmer vil kunne forbedre din arbejdsgang?

X Ja __ Nej

5. Tror du at en tablet med de rigtige programmer vil kunne forbedre børnenes hverdag?

X Ja __ Nej

Hvor let mener du at applikationerne var at bruge?

GIRAF:

__ Meget let
X Let
__ Middel
__ Svær
__ Meget svær

Parrot:

__ Meget let
__ Let
X Middel
__ Svær
__ Meget svær

Wombat:

__ Meget let
__ Let
X Middel
__ Svær
__ Meget svær

Admin:

__ Meget let
__ Let
X Middel
__ Svær
__ Meget svær

Hvor let mener du at Web interfacet var at bruge?

Meget let
__ Let
X Middel
__ Svær
__ Meget svær

Test Peron 5 - Questionnaire

Usabilitytest - Spørgeskema

1. Hvilket køn er du?

X Kvinde __ Mand

2. Hvor erfaren vurderer du at du selv er med computerer? (vælg en)

__ Meget begrænset erfaring
__ Lettere erfaren
X Forholdsvis erfaren
__ Meget erfaren

3. Har du brugt en tablet før (f.eks. en iPad)?

X Ja __ Nej

4. Tror du at en tablet med de rigtige programmer vil kunne forbedre din arbejdsgang?

X Ja __ Nej

5. Tror du at en tablet med de rigtige programmer vil kunne forbedre børnenes hverdag?

X Ja __ Nej

Hvor let mener du at applikationerne var at bruge?

GIRAF:

__ Meget let
__ Let
X Middel
__ Svær
__ Meget svær

Parrot:

__ Meget let
X Let
__ Middel
__ Svær
__ Meget svær

Wombat:

X Meget let
__ Let
__ Middel
__ Svær
__ Meget svær

Admin:

__ Meget let
X Let
X Middel
__ Svær
__ Meget svær

Hvor let mener du at Web interfacet var at bruge?

Meget let
__ Let
__ Middel
X Svær
__ Meget svær

https://docs.google.com/document/d/1ciAZukdbFtR8H8EpGx6IcG4CNPsQ4ggyqCRkxR8gA-o/edit 2/2

BIBLIOGRAPHY

[1] Scrum Alliance. Scrum alliance, 2011. URL http://www.scrumalliance.org/. [Online; accessed at 01-june-2012].

[2] Scrum Alliance. Advice on conducting the scrum of scrums meeting, 2011. URL http://www.scrumalliance.org/articles/46-advice-on-conducting-the-scrum-of-scrums-meeting. [Online; accessed at 01-june-2012].

[3] Ross Burton. Subverting Java Access Protection for Unit Testing, 2003. URL http://onjava.com/lpt/a/4346. [Online; accessed at 01-June-2012].

[4] D. Robinson, K. Coar. The Common Gateway Interface (CGI) Version 1.1. http://tools.ietf.org/html/rfc3875#section-1, October 2004. [Online; accessed 29-May-2012].

[5] Temple Grandin. Teaching tips for children and adults with autism, December 2002. URL http://www.autism.com/ind_teaching_tips.asp. [Online; accessed at 01-june-2012].

[6] Hans Bergsten. An Introduction to Java Servlets. http://www.developer.com/java/an-introduction-to-java-servlets-.html, 18. May 2000. [Online; accessed 17-May-2012].

[7] Jason Hunter. *Java Servlet Programming*. O'Reilly Media, 1998. ISBN 156592391X.

[8] Steffan Bo Pallesen Jacob Bang, Lasse Linnerup Christiansen. Administration module for giraf, May 2011. URL `http://people.cs.aau.dk/~ulrik/Giraf/Admin.pdf`. [Online; accessed at 01-june-2012].

[9] Java™Platform, Standard Edition 6. Overview (Java Platform SE 6. `http://docs.oracle.com/javase/6/docs/api/overview-summary.html`, 09. January 2012. [Online; accessed 20-May-2012].

[10] Jesper Kjeldskov, Mikael B. Skov, Jan Stage. Instant Data Analysis: Conducting Usability Evaluations in a Day. *NordiCHI, Nordic Conference on Human-Computer Interaction*, 2004.

[11] Jukka Korpela. File input (or upload) in HTML forms, 2012. URL `http://www.cs.tut.fi/~jkorpela/forms/file.html#value`. [Online; accessed at 31-May-2012].

[12] Craig Larman. *Agile and Iterative Development: A Manager's Guide*. Addison-Wesley Professional, 2003. ISBN 0131111558. URL `http://www.amazon.com/Agile-Iterative-Development-Managers-Guide/dp/0131111558%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0131111558`.

[13] live.gnome.org. Dia homepage. `https://live.gnome.org/Dia`, 2011. [Online; accessed 23-May-2012].

[14] Mehdi Achour, Friedhelm Betz, Antony Dovgal,Nuno Lopes, et. al. What can PHP do? `http://www.php.net/manual/en/intro-whatcando.php`, 11. May 2012. [Online; accessed 17-May-2012].

[15] Mehdi Achour, Friedhelm Betz, Antony Dovgal,Nuno Lopes, et. al. History of PHP - PHP Tools, FI, Construction Kit, and PHP/FI. `http://www.php.net/manual/en/history.php.php`, 11. May 2012. [Online; accessed 17-May-2012].

[16] Mehdi Achour, Friedhelm Betz, Antony Dovgal,Nuno Lopes, et. al. What can PHP do? `http://www.php.net/manual/en/tutorial.requirements.php`, 11. May 2012. [Online; accessed 17-May-2012].

[17] Microsoft. Terms of use. `http://www.asp.net/terms-of-use`, 2. November 2010. [Online; accessed 17-May-2012].

[18] Microsoft. Model-view-controller, May 2012. URL http://msdn.microsoft.com/en-us/library/ff649643.aspx. [Online; accessed at 01-june-2012].

[19] Network Sorcery, Inc. ICMP, Internet Control Message Protocol. http://www.networksorcery.com/enp/protocol/icmp.htm, 20. April 2010. [Online; accessed 20-May-2012].

[20] DevX Pro. Servlets vs JSP, 1999. URL http://www.devx.com/tips/Tip/25217. [Online; accessed at 30-May-2012].

[21] Samsung. Samsung tablet, 2012. URL http://www.samsung.com/global/microsite/galaxytab/10.1/index.html. [Online; accessed at 01-june-2012].

[22] Scott Trent, Michiaki Tatsubori, Toyotaro Suzumura, Akihiko Tozawa and Tamiya Onodera. Performance Comparison of PHP and JSP as Server-Side Scripting Languages. http://www.research.ibm.com/trl/people/mich/pub/200812_middleware2008specweb.pdf, 2008. [Online; accessed 17-May-2012].

[23] Allen G. Taylor. *SQL For Dummies (For Dummies (Computers))*. For Dummies, 2007. ISBN 978-0-470-04652-4. URL http://www.amazon.com/SQL-For-Dummies-Computers-ebook/dp/B003S9VTFU%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3DB003S9VTFU.

[24] Android Development Team. Android 4.0.3 platform, 2012. URL http://developer.android.com/sdk/android-4.0.3.html#relnotes. [Online; accessed at 01-june-2012].

[25] Aalborg University. Studieordning for bacheloruddannelsen i software. *URL: http://www.sict.aau.dk/digitalAssets/3/3331_softwbach_sept2009.pdf*, 2009. Last viewed: 2012-03-15.

[26] Don Wells. extreme programming, 2009. URL http://www.extremeprogramming.org/rules.html. [Online; accessed at 01-june-2012].

[27] Wikipedia - the free encyclopedia. Java Servlet. http://en.wikipedia.org/wiki/Java_Servlet, 11. May 2012. [Online; accessed 17-May-2012].

[28] www.jdom.org. Jdom hompage. `http://www.jdom.org`, 2012. [Online; accessed 10-February-2012].

[29] www.junit.org/. Welcome to junit.org! | junit.org, 2012. URL `http://www.junit.org/`. [Online; accessed at 01-june-2012].

[30] www.saxproject.org. Sax, 2004. URL `http://www.saxproject.org`. [Online; accessed at 01-June-2012].

[31] www.w3school.com. Xpath tutorial, 2012. URL `http://www.w3school.com/xpath`. [Online; accessed at 02-june-2012].

[32] www.w3school.com. Xslt tutorial, 2012. URL `http://www.w3school.com/xsl`. [Online; accessed at 02-june-2012].

LIST OF FIGURES

LIST OF TABLES

LISTINGS