# insert title here

os

May 23, 2012

# CONTENTS

# CHAPTER 1

# COMMON REPORT STUFF

# CHAPTER 2

# AGILE DEVELOPMENT/DEVELOPMENT METHOD

## 2.1 Theory

## 2.2 Scrum implementation

# CHAPTER 3

# SAVANNAH

## 3.1 Requirements

### Initial Requirement gathering

Requirement gathering was done in the multi project group in the very early stages of the project, but has also been updated as we have been progressing and showing our work to our customers. In the initial stages we did semi-structured interviews of our customers, where the primary focus was understanding our target group and exploring any tools they currently have access to. As mentioned the interviews were conducted in a semi-structured manner, we had a few questionsbut tried to let the interview run as an informal conversation where the customers could present their own ideas and visions of the project.

From these interviews we created an initial list of requirements for the multi project. three interviews were done, with Mette and Christina from (ehhh) , Drazenko from (ehh), and Tove from Tale Instituttet(the speech center). From the individual interviews we gathered a list of their individual ideas and visions and made a requirements list which groups later could refer back to.

- Mette And Christina

  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

From the list of requirements of the individual customers, we identified the requirements which were common for all of them, and made a list of requirements which should be common for all groups in the multi project.

- Customizable software

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

  Customization should trump feature bloat.

- Emphasize visual stimuli

  Autistic people are visually over stimulated.

- Authentic/proper feedback or behavior.

These are all requirements set by our customers, however, since this system is dealing with potentially sensitive personal data, it is also a requirement that before it can be deployed in a real world situation, that all transmissions are done via a secure connection.

## 3.2 Database

To be able to have profile control in the system, a way to keep track of the various profiles are required. A MySQL database has been designed and implemented to handle this. This section describes the process of the database creation.

### 3.2.1 Design

The database is designed in MySQL 5.1.61, and resembles the local database created by the "Admin-group" very closely – the only difference is, that the Savanah database has two extra fields in the "AuthUsers" table: username and password. The reason for this is, that while all apps running on the android platform uses QR-codes for user authentication, this is not a fitting choice for the webinterface; it would be more complicated to scan the QR-code form a webcam to login, than to simply use a username and password combination.

#### Requirements

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

- All users must be able to login with a QR-code
- Each department must also 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 have access all the 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 different tags

- Pictures must be able to be linked with audio and vice versa
- A department must be able to have a subdepartment[1]

### Diagram

A database diagram has been made to get a overview of the different tables and their relations. This diagram is designed in DIA, which is a GTK+ based diagram creation program[4], and can be seen in Figure 3.1.

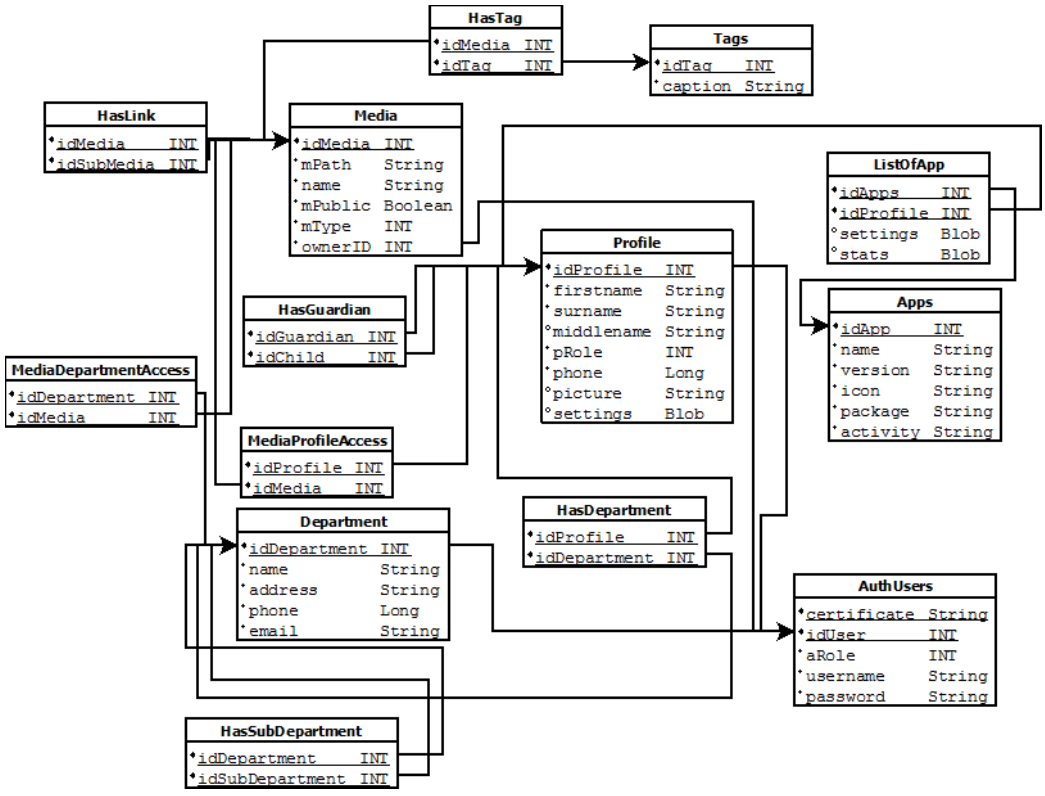



Figure 3.1: Dia diagram of the database

Although the diagram in Figure 3.1 does not meet the standard of database diagrams, it does provide a good view of the different tables, and the relations: Foreign keys points to where the value is fetched form. This crude

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

diagram shows that be having the "idUser" from the "AuthUsers" one is able to access all other information for that specific user.

**Rules**

To provide the security needed in the system, a few rules need to apply for the databse:

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

These rules will be applied in a mix between SQL script and software level in **REF TIL SECTION**

### 3.2.2 Implementation

The implementation of the MySQL database is done in MySQL Workbench 5.2.40 which is a GUI tool that provides a large set of features for various purposes.

Due to the dependencies of the various tables, the order of how the need to be created is quite strict. The code to create the tables is found in the appendix section A.1. To make sure the "username" and "userID" follows the previously stated rules, both fields has been made `UNIQUE NOT NULL` and the "userID" furthermore has `AUTO_INCREMENT` as seen in Listing A.1. This makes sure there can be no one with the same "username" or "userID" and the "userID" will automatic increase when new data is inserted. This also

guarantees a one-to-one relation between both "Profile" and "AuthUsers" and "Departments" and "AuthUsers". To be able to distinguish between "Department" and "Profiles" a filed called `aRole` is used. This is a int and will hold a number, which will be used at software level to do the distinguishing. The same applies for the distinguishing in "Profile" (code shown in Listing A.4), here a field called `pRole` holds an int, which again will be used at software level.

The MySQL Workbench provides the functionality to create a ERR diagram form an existing database, this diagram is shown in Figure 3.2. The diagram is con completely as MySQL workbench creates it, the real is shown in the appendix section A.2. But this is a error in the software; as seen in the diagram, the tool generates the "Profiles"→"AuthUsers" as a one-to-many relation. This however, is not possible since the "idUser" field in "AuthUsers" is unique (as seen in Listing A.1), the same goes for both "Departments" and "Medias".

To make the deletion of data easy, many of the filed in the tables has the constraint `ON DELETE CASCADE`. This can be "dangerous", as side-effects can result in unintended data will be deleted. To makes sure no unintended data will be deleted, the software level implementation needs to warn the user when deleting data. Furthermore an analysis has been made to argue for which fileds can have this constraint. The following tables and fields has the cascade constraint:

```
‘‘Profiles’’.’’idProfile’’,
‘‘ListOfApps’’.’’idProfile’’,
‘‘HasDepartment’’.’’idProfile’’,
‘‘HasGuardian’’.’’idGuardian’’,
‘‘HasGuardian’’.’’idChild’’,
‘‘Media’’.’’OwnerID’’,
‘‘HasTag’’.’’idMedia’’,
‘‘HasLink’’.’’idMedia’’,
‘‘HasLink’’.’’idSubMedia’’,
’’MediaProfileAccess’’.’’idProfile’’.
```

### Use case

An use case of the constraint is:

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

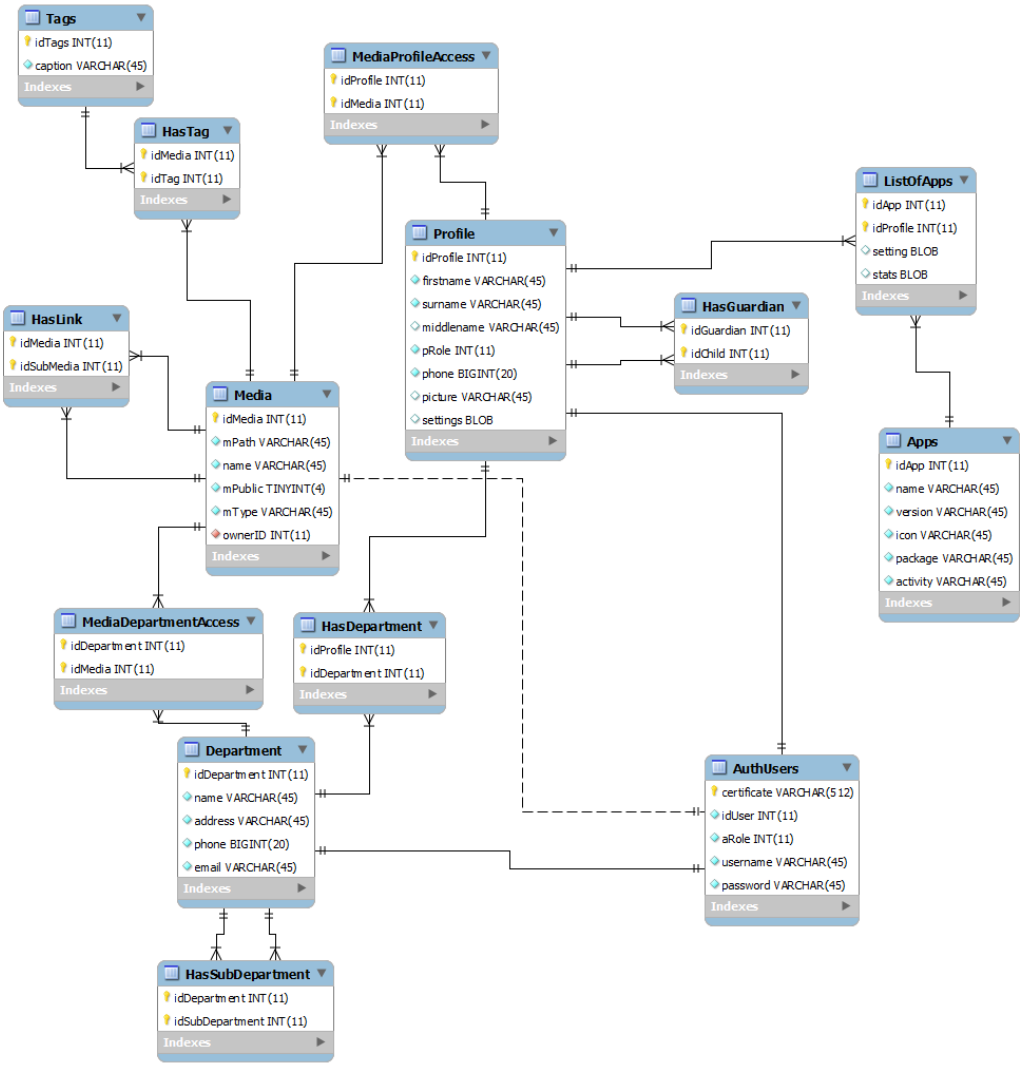



Figure 3.2: The ERR diagram

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"

### 3.2.3 Test

## 3.3 Webinterface

### 3.3.1 Programming language (Jesper)

To be able to provide the desired functionality form 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. Due to the fact that ASP.net is not open source [8], and the project is, this is not a feasible choice. This section will do a deeper analysis of PHP and Java 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) **Dette skal forklares** 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 form "Personal Home Page Tools" to the recursive acronym "PHP: Hypertext Preprocessor" as which it is known today. This is the first version that colsely resembles PHP as it exists today [6].

### Key functionality

The following list is a segment of functionality found in PHP 5.4.3. This is based on [5].

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

**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 no limited to MySQL, SQLite and PostgresSQL. 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, this needs to support PHP. Php.net recommends using Apache web server, and and for database control MySQL [7].

## Java Servlets

The first version of Java servlets was created by Sun Microsystems in mid 1997, and in December 2009 version 3.0 was released[11]. It was developed to use the advantages of Java to solve the problems of performance, scalability and security in CGI[1].

### Key functionality

The following list is a segment of functionality found in servlets. This is based on [2].

**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, multithreating, 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 Java servlet. Apache has made a service to allow the execution of servlets, called Tomcat.

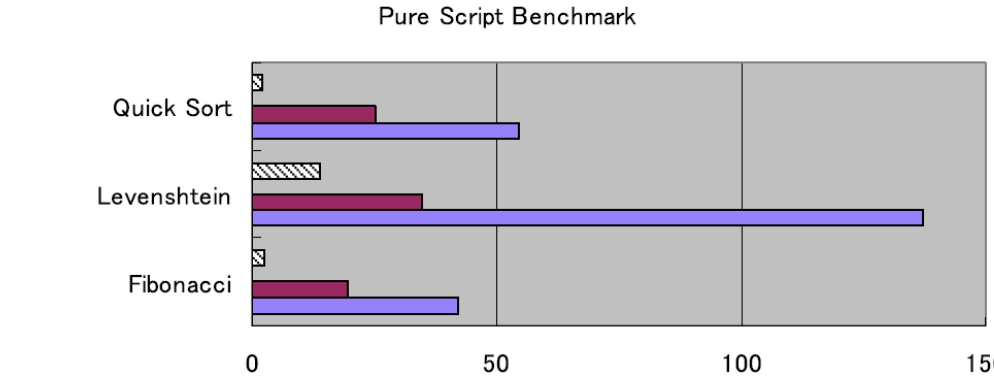
## Comparison between PHP and servlets

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 requiremtns for the web interface. The following is based on [10] - a research paper by "IBM Tokyo Research Laboratory".

### Efficiency

The web interface must be able to run fast and smooth, even on high load. A pure benchmark test is found in Figure 3.3. The test calculates a quicksort which sorts 100 integers, A Levenshtein alogithm to measure the similarity between two strings of 56 characters and a Fibonacci execution which calculates the 15th values, 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. [10, p. 167]




| | 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 3.3: A pure script benchmark calculating the average time of 1000 [10]

As seen from Figure 3.3 servlets are in all test faster, both with and without the 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 found in [10, p. 173], is shown in Figure 3.4.

> [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 [10, p. 173].

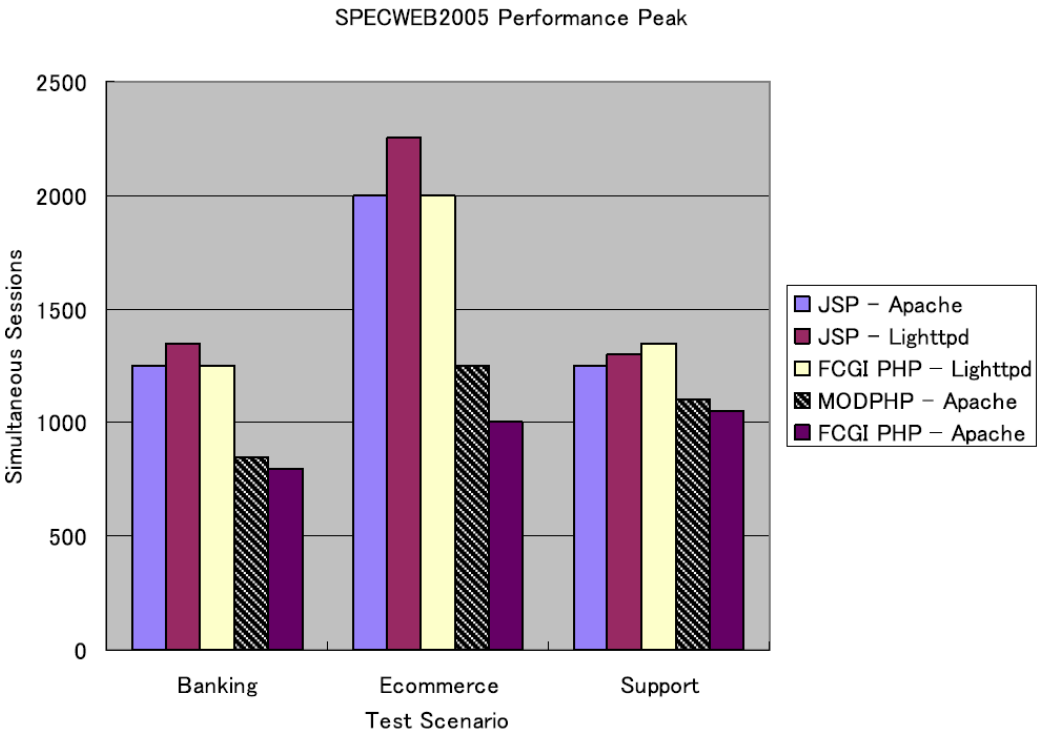



Figure 3.4: TROLOLOLO

Figure 3.4 shows that in 2 out of 3 test cases JSP is able to handle more active users than PHP.

Even though the servlets genually is more powerfull than PHP, this only comes to show, when the servler load is high (around 750 active sessions at the same time). Eventhough the efficiency of JSP is better than PHP [10] sums up the descussion quite well:

> When implementing a web server system which will never experience high load, or in which performance, throughput, and 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 [10, p. 181].

We have chosen to implement the web interface in servlets, due to the facts that we have a lot of experience in Java already and the Andoid apps are developed in Java, so choosing this seems fitting.

### 3.3.2 Implementation

### 3.3.3 Test

## 3.4 serverside

The following chapter concerns the design and implementation of the server side software for savannah and the sw6ml language.

### 3.4.1 Design

An overall design and architecture was designed in the early sprints of the project. The server side software is different from the rest of the software made in the project group, as the customers will never actually see it in action, it works perfect if they never notice it is there. Almost all requirements from our customers concern the user interface and general feel of the apps for the mobile devices. 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 rest of the world to us.

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 the Savannahs database. Rather than having Oasis query the global database directly, it was decided to implement access to both the databases through a software layer. The pros and cons of this seemingly redundant software layer was considered, see Table 3.1

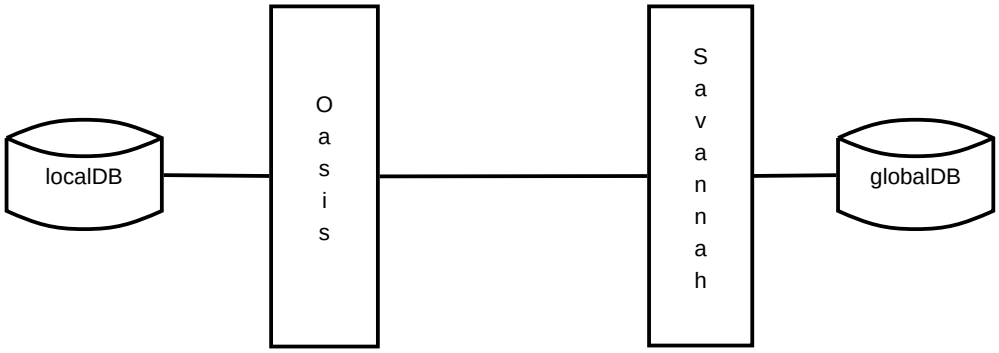



Figure 3.5: Software layers

| Pros | Cons |
|---|---|
| More flexibilty | higher complexity |
| | Lower performance |

Table 3.1: 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 worry about Savannahs 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.

### Architechture

#### sw6ml

In order to facilitate consistent data transfer between the global database and localDB, we have designed an XML language which mimics 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[12]. JDOM is a light weight implementation of SAX and the Data Object Model(DOM) for java, which allows seamless integration of XML, with support for XPath and

XSLT. In section 3.4.1 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 tell the server what action is should do with the incomming data. in Listing 3.1 a short example of legit 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 identify the table on which we want to make changes, the following `<Entry>..content..</Entry>` elements define what 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 3.1: Example of sw6ml syntax

The `action="foo"` xml attribute has four legal types corresponding to the CRUD profile actions, create, read, update, and delete, shown in Listing 3.2. 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 3.2: sw6ml crud simple type

### Documentation

To use the current version of sw6ml and savannah together it is essential to know the elements required from the different crud types. while XML schema provides advanced features for dynamic languages, sw6ml int 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 3.3 show the formal structure of a sw6ml document.

```
1  <sw6ml>
2    <table_element_1>
3      <Entry action="crud_type">
4        <table_element_1_attribute_1/>
5          ...
6        <table_element_1_attribute_n/>
7      </Entry>
8    </table_element_n>
9    ...
10   <table_element_n>
11     <Entry action="crud_type">
12       <table_element_n_attribute_1/>
13         ...
14       <table_element_n_attribute_n/>
15     </Entry>
16   </table_element>
17 </sw6ml>
```

Listing 3.3: 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 3.1 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 3.4 for an example.

**delete** Deletes a row in the able: Only the unique identier is required. See Listing 3.1 for an example.

**read** Read is only used in xml which is send back on a request to the server, and thus require no special formatting.

Notice the `<row_attribute type="bar">..</..>` xml attribute, legal types are `string` or `int`, if the data type of the row attribute is a string or any other type requiring apostrophes in a 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 3.4: ”sw6ml Update syntax example

The sw6ml schema can be found on the project repository together with an 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`

### 3.4.2 Implementation

#### Input handling

When the server starts it will execute a method called `listen()` that listens for any connections, see listing 3.5. Whenever any `Socket` connects to the server, the `listen()` method will create a new `CommunicationThread`. This thread will then read the information in the `Socket`'s `InputStream` and depending on the connection type (commit event, request event or ping) it will deal with it appropriately.

```
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                         //                              this.
                               connections.add(new ConnectionIO(con, new
```

```
                        DataOutputStream(con.getOutputStream()))
                    );
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 3.5: Method: `listen()`

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

### Ping

Diagram ledger: (see figure 3.6)
The uppermost port is used to indicate input and the lowermost port is used to indicate output. The arrows indicate which way the information is forwarded in the system. Any arrows leading to and fro a component, and not from an input or output port, are to be executed first.

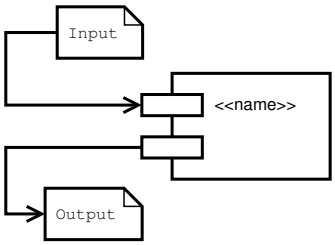



Figure 3.6: Ledger for diagrams

Figure 3.7 is a model of how a ping is process 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 `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 just directly respond to the caller (connection).

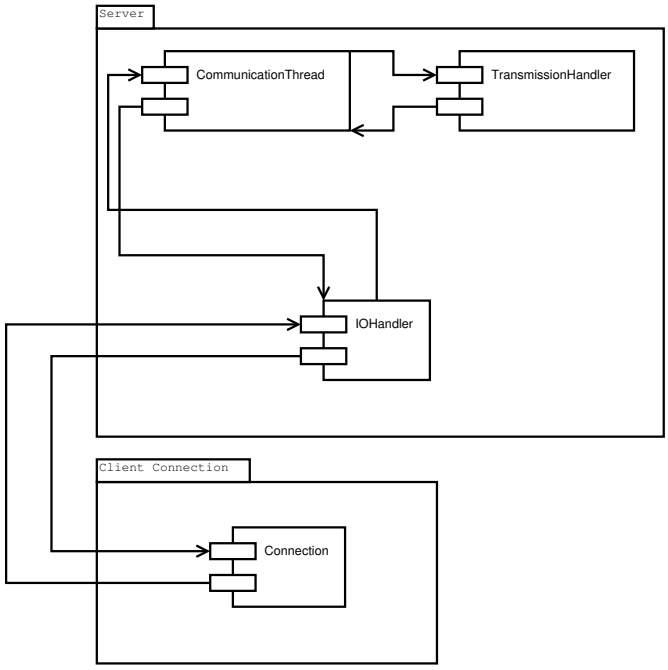



Figure 3.7: A diagram illustrating how a ping is processed.

The reason for this implementation is that Java only supports two types of sockets: stream based (TCP — `java.net.Socket` and `java.net.ServerSocket`) and datagram based ones (UDP — `java.net.DatagramSocket` and `java.net.MulticastSocket`)[9][3]. However an implementation of ping would require the use of ICMP(Internet Control Message Protocol) ping, and since this is not possible we have chose to implement a ping at a software level, rather than as a 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 determine the type of the connection, it will opposite for the ping, create a new event corresponding to the type and send it to the `EventHandler`, see figure 3.8. First when the `EventHandler` is done processing the event, will the server respond to the connector. To see how the `EventHandler` processes the given queries see section 3.4.2.

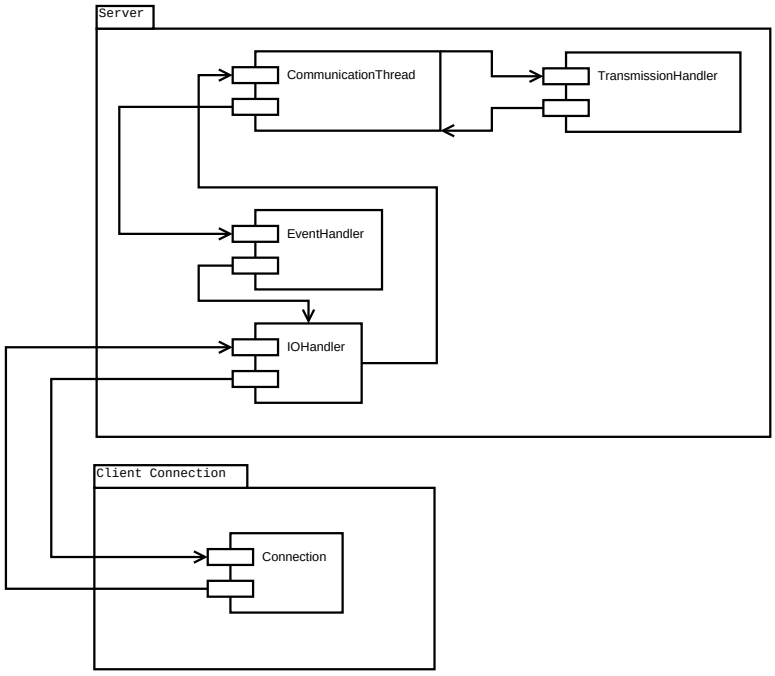



Figure 3.8: A diagram illustrating how a request or a commit event is processed.

#### Queue and Query handling

### 3.4.3 Test

#### JUnit

#### Other test

# CHAPTER 4

# RECAPITULATION

## 4.1 Conclusion

## 4.2 Future work

## 4.3 Multi project

# APPENDIX A

# APPENDIX

## A.1 MySQL code

```
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` (
```

```
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

```
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

```
CREATE TABLE `04`.`Department` (

  `idDepartment` INT NOT NULL ,

  `name` VARCHAR(45) NOT NULL ,

  `address` VARCHAR(45) NOT NULL ,

  `phone` BIGINT NOT NULL ,

  `email` VARCHAR(45) NOT NULL ,

  FOREIGN KEY (`idDepartment` )

  REFERENCES `04`.`AuthUsers` (`idUser` ),

  PRIMARY KEY (`idDepartment`) );
```

Listing A.6: Create Department

```
CREATE TABLE `04`.`HasDepartment` (

  `idProfile` INT NOT NULL ,

  `idDepartment` INT NOT NULL ,

  FOREIGN KEY (`idProfile` )

  REFERENCES `04`.`Profile` (`idProfile` ) ON DELETE CASCADE,

  FOREIGN KEY (`idDepartment` )

  REFERENCES `04`.`Department` (`idDepartment` ),


  PRIMARY KEY (`idProfile`, `idDepartment`) );
```

Listing A.7: Create HasDepartment

```
CREATE TABLE `04`.`HasGuardian` (

  `idGuardian` INT NOT NULL ,

  `idChild` INT NOT NULL ,

  FOREIGN KEY (`idGuardian` )

  REFERENCES `04`.`Profile` (`idProfile` ) on delete cascade,

  FOREIGN KEY (`idChild` )

  REFERENCES `04`.`Profile` (`idProfile` ) on delete cascade,

  PRIMARY KEY (`idGuardian`,`idChild`) );
```

Listing A.8: Create HasGuardian

```
CREATE TABLE `04`.`HasSubDepartment` (

  `idDepartment` INT NOT NULL ,

  `idSubDepartment` INT NOT NULL ,

  FOREIGN KEY (`idDepartment` )

  REFERENCES `04`.`Department` (`idDepartment` ),

  FOREIGN KEY (`idSubDepartment` )

  REFERENCES `04`.`Department` (`idDepartment` ),

  PRIMARY KEY (`idDepartment`, `idSubDepartment`) );
```

Listing A.9: Create HasSubDepartment

```
CREATE TABLE `04`.`Media` (

```

```
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

```
CREATE TABLE `04`.`HasLink` (

  `idMedia` INT NOT NULL ,

  `idSubMedia` INT NOT NULL ,

  FOREIGN KEY (`idMedia` )

  REFERENCES `04`.`Media` (`idMedia` ) on delete cascade,

  FOREIGN KEY (`idSubMedia` )

  REFERENCES `04`.`Media` (`idMedia` ) on delete cascade,

  PRIMARY KEY (`idMedia`, `idSubMedia`) );
```

Listing A.12: Create HasLink

```
CREATE TABLE `04`.`MediaProfileAccess` (

  `idProfile` INT NOT NULL ,

  `idMedia` INT NOT NULL ,

  FOREIGN KEY (`idProfile` )

  REFERENCES `04`.`Profile` (`idProfile` ) ON DELETE CASCADE,

  FOREIGN KEY (`idMedia` )

  REFERENCES `04`.`Media` (`idMedia` ) ON DELETE CASCADE,

  PRIMARY KEY (`idProfile`, `idMedia`) );
```

Listing A.13: Create MediaProfileAccess

```
CREATE TABLE `04`.`MediaDepartmentAccess` (

  `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 ERR diagram

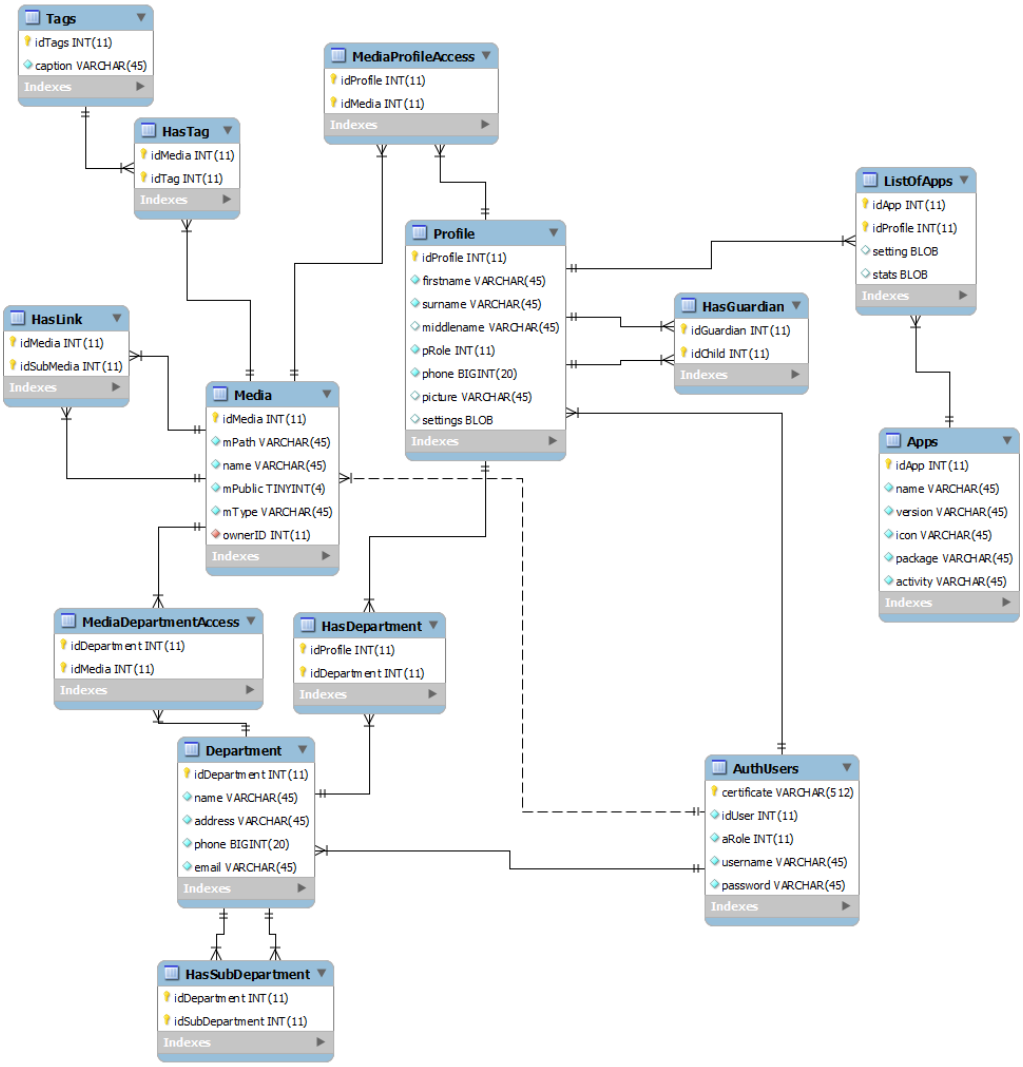



Figure A.1: The ERR diagram as MySQL Workbench generates it

# BIBLIOGRAPHY


[1] 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].

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

[3] 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].

[4] live.gnome.org. Dia homepage. https://live.gnome.org/Dia. [Online; accessed 23-May-2012].

[5] 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].

[6] 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].

[7] 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].

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

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

[10] 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].

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

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