Studienarbeit
Supporting Learning on Mobile Devices

Author: Yeray García Quintana
Matriculation Number: 2481753
Address: Arcisstr. 21
80333 München
Advisor: Andreas Möller
Begin: 18.10.2010
End: 31.03.2011
Abstract

In this thesis we will address the lack of an information system for lecturers to manage and share didactic methods. This is part of a bigger mission: The system will hopefully help lecturers learn about new ways to increase the quality of their teaching strategies and it will also help them make their courses more engaging for the students. On the other hand, we wanted it to adapt to the current life style where information is accessible from everywhere. So lecturers should be able to consult didactic content on the go, in the breaks between lectures, in the subway, etc. To achieve this goal we have come up with a combination of a cloud-based database and two interfaces, namely, a webapp\footnote{A webapp or web application is an application that is accessed over the Internet. It is therefore based solely on web technology and is, thus, platform independent} and a mobile app for Android devices. All this three components communicate with each other to build a complex system that behaves as one entity. What we have done, in the end, is not only create the necessary infrastructure to achieve our immediate goal but we have also gotten one step closer to our bigger mission.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1 Introduction</strong></td>
<td>5</td>
</tr>
<tr>
<td>1.1 Overview</td>
<td>6</td>
</tr>
<tr>
<td><strong>2 Related Work</strong></td>
<td>7</td>
</tr>
<tr>
<td>2.1 Overview</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Books</td>
<td>7</td>
</tr>
<tr>
<td>2.3 Courses</td>
<td>7</td>
</tr>
<tr>
<td>2.4 Multimedia</td>
<td>8</td>
</tr>
<tr>
<td>2.5 Online content</td>
<td>8</td>
</tr>
<tr>
<td>2.6 Mobile content</td>
<td>8</td>
</tr>
<tr>
<td>2.7 Conclusion</td>
<td>9</td>
</tr>
<tr>
<td>2.8 Other systems supporting learning</td>
<td>9</td>
</tr>
<tr>
<td><strong>3 Survey Of Demand</strong></td>
<td>10</td>
</tr>
<tr>
<td><strong>4 Technical Part</strong></td>
<td>13</td>
</tr>
<tr>
<td>4.1 Overview</td>
<td>13</td>
</tr>
<tr>
<td>4.2 Android Mobile App</td>
<td>13</td>
</tr>
<tr>
<td>4.2.1 ActionBar</td>
<td>14</td>
</tr>
<tr>
<td>4.2.2 Sign in screen</td>
<td>15</td>
</tr>
<tr>
<td>4.2.3 Sign Up Screen</td>
<td>15</td>
</tr>
<tr>
<td>4.2.4 Home screen</td>
<td>15</td>
</tr>
<tr>
<td>4.2.5 Favorites</td>
<td>17</td>
</tr>
<tr>
<td>4.2.6 Search</td>
<td>18</td>
</tr>
<tr>
<td>4.2.7 Detailed View</td>
<td>22</td>
</tr>
<tr>
<td>4.2.8 Image Gallery</td>
<td>24</td>
</tr>
<tr>
<td>4.2.9 Settings</td>
<td>25</td>
</tr>
<tr>
<td>4.3 Remote MySQL Database</td>
<td>26</td>
</tr>
<tr>
<td>4.3.1 Tables For Didactic Methods</td>
<td>27</td>
</tr>
<tr>
<td>4.3.2 Tables For The User Account System</td>
<td>27</td>
</tr>
<tr>
<td>4.4 Web Server</td>
<td>29</td>
</tr>
<tr>
<td>4.4.1 Header</td>
<td>30</td>
</tr>
<tr>
<td>4.4.2 Sign in</td>
<td>30</td>
</tr>
</tbody>
</table>
4.4.3 Forgot Password .................................................. 31
4.4.4 Sign up ............................................................. 32
4.4.5 List of Methods .................................................. 33
4.4.6 View single method ............................................ 37
4.4.7 Edit Method ....................................................... 39
4.4.8 Add new method ............................................... 39
4.4.9 View Account .................................................... 40
4.4.10 Edit account ...................................................... 40
4.4.11 Admin Center ................................................... 41

4.5 Communication Between Mobile App and Website .................. 42

5 Possible Improvements .................................................. 45
5.1 Recommendation system ........................................... 45
5.2 Sorting by popularity ................................................ 45
5.3 User Interaction ..................................................... 45
5.4 Optimization ......................................................... 46
5.5 User Base Expansion ............................................... 46

6 Conclusion ................................................................. 48

List of Figures ............................................................. 49
List of Tables .............................................................. 50
Bibliography ................................................................. 51
Chapter 1

Introduction

The use of smartphones is increasing everyday and with it a life style where every piece of information is available everywhere. Smartphones come today most of the time with Internet flat rates which give them the power of a nearly infinite source of information. So why not take advantage of this potential and use it to solve problems that exist nowadays? Well, this is exactly what we have done.

The Institute for Media Technology at Technische Universität München focuses on several research topics. Some of them are networked multimedia systems, mobile multimedia, human-X interaction, ubiquitous computing or multi-modal information processing. Combining several aspects of each of this research areas we came up with a practical project whose main goal was to build a system for professors, doctorate candidates or assistants to manage didactic methods right from their mobile phones. This way they could read about their favorite didactic methods, learn about new ones, share them with others, search for specific criteria and all of this from the palm of their hands.

To achieve this goal we not only had to make sure the system worked from a technical point of view, but we had to work together with didactic departments from several universities in Munich, such as the Sprachraum at the Ludwig-Maximilians-Universität München or the Carl von Linde-Akademie at the Technische Universität München, to make the final product usable.

No mobile app exists yet for this purpose and this made its design and implementation a little more difficult but also more engaging and even fun. We have encountered some difficulties along the way and there is still plenty of room for improvements but we think we have done a good job.

On a side note, I would like to mention that this project not only let me apply the knowledge I have been acquiring throughout my studies, but it also let me get in touch with personal interests and current trends, such as cloud computing and the emerging mobile operating system Android.

Without further ado, let me guide you through this thesis.
Chapter 1 Introduction

1.1 Overview

This thesis is divided in several parts. First we are going to see what the state of the art is in this field and what other ways there are for users to access didactic content.

After analyzing the related work and choose the goal of our project we are going to show the results of a survey that was conducted to verify if there really was a demand for the system we were about to build. This survey will also serve us to extract from potential users which are the most important features the system should have to be a high-quality product. This will give us a good starting point to begin with the design of the system and it will also give us the direction we should follow to achieve our goal.

After that we will continue with the technical part, where we will get into the details of every component of the system. We will see how each component was implemented, why it was implemented that way and how each component interacts with each other to form a complex system capable of solving the problem presented at the beginning.

After that we will discuss the possible directions in which the system can be expanded in the future.

At the end we will come to a conclusion where we will synthesize all chapters to leave the reader with a complete idea of what was achieved.
Chapter 2

Related Work

2.1 Overview

In this chapter we will see what the state of the art is on sources for didactic methods. We will discuss what the pros and cons are for each of them.

Nowadays there are different types of sources a teacher or professor can go to for information on teaching methods, for example:

1. Books
2. Courses
3. Multimedia (CD’s, DVD’s)
4. Online content (webpages, video clips, documents)
5. Mobile content (smartphone apps)

2.2 Books

Books are excellent information sources to read about didactic methods. They can go into much detail and their content is most of the time of great quality. However they have multiple disadvantages: Books are not updated frequently, they cannot be carried around in great amounts and sometimes only one chapter is of interest for the reader so they can be inefficient.

2.3 Courses

Courses are great and fast ways to learn about new didactic methods, since they allow experts in the field to guide teachers and to explain them in a practical way how to apply
those methods. However the problem with them is how their participants access the content that is taught. The annotations are usually all they have left at the end and sometimes they are not sufficient. Additionally, courses require their participants to adapt to the course’s timetable so the philosophy of “any time, any place” is not present. A couple of examples are mentioned in the literature [9, 17].

2.4 Multimedia

It is well known that a person can retain more information about a topic if it is taught using audio-visual material. Because of this, multimedia sources are of great importance to a learning teacher. Nowadays this material is offered in form of CD’s and DVD’s. This has several advantages as well as disadvantages. CD’s and DVD’s allow one to watch the didactic content in the television, computer or even in class using a projector. However, it is content that is only accessible offline. It requires the CD’s or DVD’s and some equipment to display it. If, on the other hand, the user wants to retrieve that information from somewhere else, he/she will not be able to do so. Some examples are mentioned in the literature [9, 17, 19].

2.5 Online content

One of the main sources for information retrieval is the Internet. It has several advantages over the other sources, namely, it offers access from any place with an Internet connection (this includes mobile phones), online content can be easily updated, user interaction is possible so users can learn from each other, it can offer multimedia content, it can be used to categorize information and to connect it with related or similar content, etc. However, despite all this advantages, no application or website has taken advantage of its full potential. The majority of them consists of a series of links to certain didactic methods updated normally by a single person or institution. Some examples are shown in the literature [4, 6, 19, 20].

2.6 Mobile content

As mentioned in the previous section, only online content provides ubiquitous access to the information. Websites can be accessed from mobile phones using the built-in browser. However, the minority of them has a mobile version, so viewing them in the small screen of mobile devices is not a pleasant experience for the user and navigating through the website can be difficult and cumbersome.
2.7 Conclusion

As we can see, there is a lot of ways for teachers and professors to read about didactic methods. However, there is not a good way to organize, manage, share or contribute to that information yet. The goal of this project was therefore to build such a system. One where ubiquitous access to didactic methods were possible, where users could share their knowledge with the community, where they could organize their own methods and all of this from the palm of their hands.

2.8 Other systems supporting learning

There exist many approaches supporting learning with mobile devices, as well as physical artefacts, e.g. tangible objects [15, 22, 25]. Such systems particularly can support explorative and cooperative learning at various locations. However, they focus on playful learning instead of targeted acquisition of knowledge.
Chapter 3

Survey Of Demand

As can be read in the previous section, few to no didactic apps exist for teachers and professors to manage or read about their didactic methods. Therefore a survey was made to see if developing an app to fill this gap would be used at all and, if so, what requirements should it fulfill.

The survey was made to a group of people composed of students, advisors, Ph.D. students, postgraduate students, professors and assistant professors (see figure 3.1) coming from universities like the Technische Universität München, the Ludwig-Maximilians-Universität München and the Universität der Bundeswehr München. They were attending a course on didactic methods and represented therefore perfect candidates for a statistical survey. A few questions were asked to them regarding how they used their phones and where did they go to read more about didactic methods.

![Figure 3.1: Heterogeneity of the Statistical Population](image)

From figure [3.1] we can see that the majority of the asked persons owned a smartphone, this was key to decide whether an Android mobile app made any sense. On the other hand, those who didn’t own a smartphone pointed out that they would like to access the
information in some other way. We can also see that the mobile app was of interest for the majority of the interviewed persons.

Figure 3.2: Own a Smartphone and Would Use the Mobile App

As we asked them how they used their smartphone and how they looked for information about didactic methods we could draw some conclusions. Since researching about information was the activity the users did the most right behind checking their email, this gave us a hint about what type of mobile app to develop. It should be for reading and not for editing, so user input was to be reduced to a bare minimum but should allow them to filter out the desired information quite easily. As we can see in figure 3.3 one of the primary sources to read about didactic methods is the Internet. To maintain that model, the information should be available in the cloud\(^1\) so that ubiquitous access could be possible. We can also see that the users also ask their colleagues about didactic methods, so the system should allow them to create and share their methods for other users to see.

Figure 3.3: Smartphone Features And Methods’ Information Sources

The next questions were regarding the type of features they would use from the mobile app the most. In figure 3.4 we can see that searching/filtering, examples and multimedia are the most popular. So special thought must be put into the design of this sections of the mobile app.

\(^1\)Cloud is used here in the context of cloud computing, where it represents a metaphor of the Internet.
Chapter 3 Survey Of Demand

Figure 3.4: What features would be used the most

After evaluating the results of the survey we had enough information to start the design of the system. For starters, it would involve a database, an Android mobile app and probably some sort of stand-alone PC program (or even a website) for those who don’t own a smartphone. Since filtering out the desired methods turned out to be very important for the potential users, special care should be put into the design of the search functionality of the mobile app. In the long run, user interaction should be supported, so a way for the users to create and share methods should be implemented. To let the application have enough information about user preferences and learn about the relevance of certain methods, some sort of user rating or favorites system should be put in place.
Chapter 4

Technical Part

4.1 Overview

Following the guidelines of the survey of demand (see chapter 3) a system was built that allowed users to manage didactic methods. The users would create an account and would have access to a database which contained a series of didactic methods created by experts or even by other users. This database could be visualized using a website or a smartphone with the Android Operating System installed. Each user has by default edit permissions, meaning that they can create new methods and modify or delete the methods they have created. They can keep them private or share them with the community.

In this chapter the technical aspects of this system will be explained in detail.

The system is composed of three components:

1. Android mobile app
2. Remote MySQL database
3. Web server (Apache server)

4.2 Android Mobile App

This section will describe how the mobile app was designed, why it was designed that way and what was made to achieve that goal.

The mobile app is composed of eight main screens:

1. Log in
2. Sign up
3. Home (List of all methods)
4. Favorites (List of all favorite methods)
Chapter 4 Technical Part

5. Search
6. Detailed view (for each method)
7. Image Gallery
8. Settings

4.2.1 ActionBar

The navigation between the main screens is made much more intuitive with the implementation of a so called ActionBar [18, 21]. The ActionBar stays at the top of the screen replacing the default application title bar. It contains the icon of the application, the title of the current screen, and a series of buttons which would allow the user to navigate to important sections of the application, e. g., the Home screen, the Favorites screen, the Search screen and to different sections of the Detailed View screen. The user name of the currently logged in user is displayed right below it.

Several alternatives have been tested, like a tabbed interface or an initial list view (see figure 4.1). The ActionBar (apart from being the current trend in Android mobile apps [18]) has been chosen as the better alternative due to many reasons: it is intuitive and self-explanatory, it allows the user to navigate quickly to different sections of the app, it allows the developer to have full control of a custom title bar, it does not take as much screen space as a tabbed interface, it allows branding, etc.

Figure 4.1: Design Alternatives (Tabs, List, ActionBar)
4.2.2 Sign in screen

The first screen that the user encounters is the Sign in screen. Here the user can enter his/her user name and password and mark a check box which would tell the application to remember the entered credentials (see figure 4.2). If the user does not have an account in the system, he can click on a button which would take him/her to the Sign up screen.

Under The Hood

The Sign in screen is implemented in the file Login.java and performs one of two actions:

1. It allows the user to enter his/her credentials and upon clicking the Sign in button a background task (implemented in LoginTask.java) is run which would contact the remote server and check whether the entered information is correct or not (see section 4.5 for more details).

2. If the user has marked the “remember” check box in a previous session a background task (implemented in CheckLoginTask.java) is automatically run which would contact the remote server and check if the stored information is still valid (since sessions can be invalidated remotely after a maximum of two weeks) (see section 4.5 for more details).

4.2.3 Sign Up Screen

The Sign up screen is similar to the Sign in screen. Only one more field for the email address is shown (see figure 4.2). After tapping on the Sign up button, and if the operation completes successfully in the remote server, the user is automatically logged in and redirected to the Home screen.

Under The Hood

After entering the information and tapping on the Sign Up button the background task SignUpTask is started. While the mobile app talks to the remote server a “Loading” dialog is shown to the user. For more details see section 4.5.

4.2.4 Home screen

After signing in the user sees the list of all the methods which are currently stored in the remote database (see figure 4.3). Each list item consists of the following parts:

1. An icon which would allow the user to quickly identify each method
2. The method’s title

3. A star which would allow the user to mark the method as favorite

If the method is a user-defined method, it is differentiated using an additional user icon (🔗).

The user can fling up and down to navigate through the list.

At the top of the screen the user can see the ActionBar where he/she can identify the icon and name of the app. There are also two buttons which would allow him/her to navigate to the Favorites and the Search screens.

On the Home screen the user can click on the menu button, where he/she can find different menu items to sign out, to update the list of methods with newer information from the remote server and to open the settings screen.

Under The Hood

The Home screen accepts a variable (INTENT_REFRESH) that can be attached to the Intent which starts the Activity. This variable indicates whether the data should be refreshed (that is, fetched from the remote server) right away. Based on this variable, the home screen performs the following actions:
1. If \textit{INTENT\_REFRESH} is passed with the \textit{Intent} and it is \textit{true}, the data is fetched from the remote server. If it is \textit{false}, the locally stored data is shown on the screen.

2. If \textit{INTENT\_REFRESH} is not passed, we assume the data has already been fetched not long ago by another process and therefore the screen is just filled with locally stored data.

The \textit{Home} screen, as well as the \textit{Favorites} and \textit{Search} screens, is a subclass of \textit{CustomList}. This allowed me to avoid redundancy in the code and to extend it if screen-specific functionality was required. The main functionality was therefore implemented in \textit{CustomList.java} and only the method \textit{fillData()} was overwitten in the child class \textit{Home.java}. The method \textit{fillData()} selects the wanted data from the database as a \textit{Cursor} and binds that \textit{Cursor} with the user interface with the help of a \textit{List Adapter}. In this process intervene two important classes: \textit{DbAdapter} and \textit{CustomCursorAdapter}.

\textit{DbAdapter} is a class which performs all database related operations, like creating tables, reading information out of the database and updating, deleting and inserting entries in the database. It is based on the \textit{NotesDbAdapter} class from the Android Notepad tutorial \cite{3}, but has been heavily enhanced.

\textit{CustomCursorAdapter} is an important class which, as mentioned above, helps bind data in a \textit{Cursor} with a user interface element (for example, a list). In order to make the app respond fast despite a large number of list items, various techniques have been applied which have been extracted from the Android Developer examples \cite{2} and from Android presentations on performance by Romain Guy \cite{12}. One of those techniques is to create as many list items as fit inside the screen. When the user scrolls up or down and a list item is about to get hidden, that same list item holder is used to hold the data of the list item which is about to get uncovered at the other end of the screen. So if the list contains hundred elements, but only ten can be shown at a time on the screen, only ten list item views are created and reused when the user scrolls up or down.

4.2.5 Favorites

The \textit{Favorites} screen can be accessed by clicking the star button in the \textit{ActionBar}. In this screen the user can navigate through all methods that he/she has marked as favorite. Otherwise it is similar to the \textit{Home} screen (see figure \ref{fig:4.3}).

Under The Hood

\textit{Favorites} is, as mentioned above, a subclass of \textit{CustomList}. Therefore everything that applies for the \textit{Home} screen, applies also here except for following differences:

1. The \textit{ActionBar} is updated to match the current screen with a corresponding title and with a \textit{Home} button to return to the \textit{Home} screen
2. The `filldata()` method would fetch in this case only the entries from the database that are marked as favorite.

3. This class uses a `FavoritesCursorAdapter` which is a child of `CustomCursorAdapter`. `FavoritesCursorAdapter` overwrites the method `toggleFavorite()` of his parent class. The new method would not only change the value of the favorite flag, but it would also hide that list item from the current screen, since only entries marked as favorite should appear on this screen.

### 4.2.6 Search

The `Search` screen allows the user to filter the list of methods. The goal of this class is to make complex queries in an easy manner. To fulfill this goal a custom `Search Activity` had to be implemented, since the default Android search mechanisms were not sufficient.

To interact with this screen the user would enter search terms in the upper text field. Each time a new character is entered, the methods’ list is updated to match the new search terms. To narrow the search results, the user can click on the “add” button (+) and a new search field would slide in from the right-hand side. This should have the effect of making the user think that the search terms that he/she has entered until now are still there and can be accessed by swiping left or right with the finger. In each search field the user can choose between a variety of categories to search by. If the user wants to remove a
search field to widen his search results, he/she just needs to click on the “remove” button (x).

**Approaches**

The search feature is probably the feature that will be used the most, as can be extracted from the statistics of the survey, and therefore much thought has been put into its design.

It is accessible from all other screens, either by pressing the magnifying glass icon from the **ActionBar** or by pressing the hardware magnifying glass button which comes with many of Android devices.

As opposed to the approach where the user sees an empty screen and search results are shown after typing what the user wants to retrieve, the filtering approach has been chosen, where the user is shown the full list of results from the beginning (much like the **Home** screen) and he/she begins to narrow it down to the results he/she wants to retrieve.

The **Search** screen was designed in a first stage in the following way: The user sees a search field at the top of the screen and the search results right below it. When the “add” button is clicked a new search field is created right below the first one. After adding two or three search fields, one below the other, the results get hidden by the soft keyboard, so the user can’t see them and, hence, from a usability point of view, this design failed (see figure 4.4).

To solve the space problem and to give the user a notion of how many search results got hidden by the soft keyboard, the following enhancements were made (see figure 4.4):

1. The **Search Activity** was made full screen so that the whole screen, including the status bar at the top, gets covered by the **Search Activity**
2. The **ActionBar** was removed
3. An information bar was added right below the input fields showing the number of results returned by the search query
4. Every search field was inserted one next to the other, rather than one below the other, sliding in from the right or left-hand side as the user added new fields or swiped with his/her finger in either direction. This way the user gets the feeling that the search terms extend beyond the screen, but remain there in spite of not seeing them
5. By making the search results update with each new character, the need of a search button disappeared, leaving more space for the actual results

---

1 In the definition of the **Activity** in the **AndroidManifest.xml** file two variables can be defined, namely **stateVisible** and **adjustResize**, that would allow the soft keyboard to pop up when the **Activity** is launched and would allow the **Activity** to resize in order to allow the user to scroll and see what is behind the keyboard. However this features do not work when the **Activity** is in full screen mode, as can be read here: [http://code.google.com/p/android/issues/detail?id=5497](http://code.google.com/p/android/issues/detail?id=5497)
Under The Hood

The main task of the Search Activity is to interpret the user input, build an SQLite query, send it to the database and show the query results on the screen.

Interpret User Input   For the user input two types of input fields are available:

1. A simple text field that allows the user to make general queries to the database, that is, to search within all columns of the database table where the methods are stored. This input field is the only one shown when the Activity is launched.

2. A combination of drop-down input fields and text fields that allow the user to make specific queries to the database, that is, to search within one single column of the database table. This input fields are shown after clicking the “add” button, since the user would click this button to narrow down the search results and thus make a specific query.

Build An SQLite Query   In order to build the SQLite query, the set of input fields is walked through programatically and an SQLite query is thereby created with the values chosen by the user in each drop-down input field and entered in each text input field.
Send Query To The Database  In order to build the SQLite query it was necessary to take into account which type of tables to use in the database. Two approaches have been taken:

1. **Standard tables**: The main advantage is that more complex and specific queries can be made. The main disadvantage is that for large databases the response can take much longer than with Full-Text Search tables.

2. **Full-Text Search (FTS) tables**: The main advantage is that the response is much faster in comparison to standard tables. The main disadvantage is that only text and prefix queries can be made.

At this point a decision was made in favor of the standard tables, due to the ability to make complex queries and because a performance difference would not be perceived by the user considering the estimated dimension of the database.

However both approaches have been coded and one can easily switch between them by changing the variable `mFtsIsEnabled` in `Search.java` to `true` or `false`.

The SQLite query that would be sent to the database for a standard table definition would have the following structure:

If entered in the general search input field:

```sql
SELECT * FROM METHODS_TABLE WHERE (column_1 LIKE '%MY GENERAL SEARCH TERMS%' OR column_2 LIKE '%MY GENERAL SEARCH TERMS%' OR ... OR column_n LIKE '%MY GENERAL SEARCH TERMS%')
```

**Listing 4.1: General Search SQLite Query**

If entered in the specific search input field:

```sql
SELECT * FROM METHODS_TABLE WHERE column_i LIKE '%MY SPECIFIC SEARCH TERMS%'
```

**Listing 4.2: Specific Search SQLite Query**

If entered in both types of fields:

```sql
SELECT * FROM METHODS_TABLE WHERE (column_1 LIKE '%MY GENERAL SEARCH TERMS%' OR column_2 LIKE '%MY GENERAL SEARCH TERMS%' OR ... OR column_n LIKE '%MY GENERAL SEARCH TERMS%') AND column_i LIKE '%MY SPECIFIC SEARCH TERMS%' AND column_j LIKE '%MY SPECIFIC SEARCH TERMS%' ...
```

**Listing 4.3: Combined Search SQLite Query**

Show The Query Results On The Screen  After retrieving the Cursor object from the database the `CustomCursorAdapter` is used to bind the data with the GUI. Thus the results are shown to the user as a filtered list of methods.

---

2 GUI: Graphical User Interface
4.2.7 Detailed View

If the user wants to get more detailed information about a didactic method listed in one of the previous screens (Home, Favorites or Search), he/she can click on it and the Detailed View screen is launched. On this screen relevant information is shown as a list of sections composed of a section header and a section body. The available sections are: Picture gallery, Social form, Phase, Subphase, Result, Course type, Participants, Seating, Material, Time, Rating, Our Rating, Proceeding, Phase-specific proceeding, Variation, Examples, Tips, Visualization (see figure 4.5).

Figure 4.5: Detailed View and Content Menu

To jump between all these sections the user can fling up or down to scroll through the detailed view or he/she can click on the content menu available as a button in the ActionBar (see figure 4.5). Tapping on it would show a floating menu containing the headers of all sections and tapping on each entry would scroll the detailed view to the desired position.

Next to that button there are others to go to the Home screen, to the Favorites screen and to the Search screen.

To navigate through the different methods, the user can swipe the screen left or right with his/her finger and a new item would slide in from the right or left-hand side. The order in which the methods are shown depend on the previous screen. E.g. if the user tapped on a method in the Favorites screen, swiping through the methods would loop through all favorite methods, in the same order that they were shown in the Favorites screen. The same applies for the Home and Search screens.
The same gestures are supported on the gallery section. The gallery section contains a set of thumbnails the user can navigate through by swiping left or right. Once the user has found a picture he/she is interested in, he/she can tap on it. This would open it in full-screen (see Image Gallery 4.2.8).

Since user-generated content is supported, a visual distinction is made to help the user recognize which methods are default ones and which were created by another user. This was achieved by adding a user icon next to the method’s title and showing a message below the title section indicating that the method was generated by a user (see figure 4.6).

![Figure 4.6: User-defined Method](image)

**Under The Hood**

This screen is implemented in the file `ViewItem.java`. When this `Activity` is created, the database helper `DbAdapter` is instantiated, since the database will be accessed several times to retrieve the needed information.

Another important class is instantiated, namely `ViewFlipper`, which is used throughout the application. This special `View` container acts as a `View` switcher. Only one of its children `Views` is shown at a time and by calling methods like `ViewFlipper.showNext()`, `ViewFlipper.showPrevious()` or `ViewFlipper.setDisplayedChild()`, the currently displayed `View` is changed. One can even define Animations to perform between each `View` change.

A `GestureDetector` object recognizes gestures made by the user and sends different commands to the `ViewFlipper`, depending on which gesture has been performed.

In addition to this infrastructure two custom classes have been defined: `Container` and `Item`. `Container` is a convenience class which holds references to all the `Views` on the GUI which can be populated with information. `Item` is a convenience class which holds the method’s information that has been extracted from the database. By calling `Container.populateFields(Item item)` the information from the database can be shown on the screen.

Each time the user makes a gesture to flip to the next method, the information of the next method is extracted from the database and saved into `Item`. The `Container` representing the hidden `View` is fetched and populated with the information saved in `Item` in the background and then the command `ViewFlipper.setDisplayedChild()` is called, triggering the `View` change with its corresponding Animation.
4.2.8 Image Gallery

The only task of the *Image Gallery* is to show a full-screen picture of a thumbnail that a user has tapped on in the *Detailed View* screen. Therefore, the only two things the user sees when launching this screen is the *ActionBar* and the tapped picture (see figure 4.7).

![Image Gallery](image)

**Figure 4.7: Image Gallery**

Various gestures are supported on this screen:

1. **Pinch**: Pinching on a picture would enter the “zoom” mode. In the zoom mode the user can zoom in and out by pinching the fingers together or away from each other.

2. **Drag**: Once a picture has been zoomed, the user can drag the picture around by tapping and dragging the finger in the direction he/she wants the picture to move.

3. **Double tap**: After zooming in or out on the picture, the user needs to double tap on the picture to return to the navigation mode. This would resize the picture to fit the screen and allow the user to make swipe gestures to navigate through the pictures.

4. **Swipe**: Once in navigation mode, the user can swipe left or right to navigate through the pictures in the same order they were shown in the thumbnail view on the *Detailed View* screen.
Chapter 4 Technical Part

Under The Hood

This screen is implemented in SingleImage.java. As in ViewItem.java, this Activity makes use of an ImageSwitcher object, which has a similar behavior to ViewFlipper, which in combination with a GestureDetector object, handles the images switching (see 4.2.7).

To handle zooming and panning a custom ImageView has been used which has been found in the literature [7, 11] and has been modified to fit the requirements.

The main mechanism to zoom and pan is the following:

The ImageView which holds the image fills the whole screen from the ActionBar to the bottom and only the Bitmap inside it is resized and moved. An OnTouchListener is attached to the ImageView and it listens to every touch event.

Drag  When the user touches the screen with one finger the position of the touch event is saved and the drag mode is enabled. If the user moves then the finger the image is moved by the distance difference between the new position and the start position. When the user lifts the finger the drag mode is disabled.

Zoom  When the user touches the screen again with one finger and a second finger is detected the distance between both fingers is calculated and saved. The middle point is also calculated and saved. The zoom mode is enabled. When the user moves one of his/her fingers the distance between them is recalculated. Then the ratio between the new distance and the previous distance is calculated and the image is scaled by this amount around the middle point saved in the previous step. When the user lifts the finger the zoom mode is disabled.

When in navigation mode, which is activated at the beginning and after double tapping a zoomed image, only fling gestures are recognized using the same GestureDetector object mentioned in 4.2.7.

4.2.9 Settings

The Settings screen allows the user to change different aspects of the mobile app. Here the user can change the type of network connection, whether to save remotely fetched information in the SD card or clear the cache (see figure 4.8).
Chapter 4 Technical Part

Figure 4.8: Settings Screen

Under The Hood

The Settings screen is implemented in the Preferences.java file. It extends PreferenceActivity available in the Android SDK\(^3\) and, thus, has the same look and feel as every other settings screen throughout the Android Operating System. Therefore the user is already familiar with it and knows instantly how it works.

The Preference class contains multiple static methods to set and retrieve the settings variables needed throughout the application. This methods can be called directly, passing a Context object as argument, without the need to create an instance of the class Preference.

4.3 Remote MySQL Database

All the information shown in the mobile app is retrieved from a remote MySQL database. The database is used for two main tasks: Storing information about the didactic methods and storing user credentials for a user account system.

The database contains the following tables:

1. For didactic methods:

\(^3\)SDK: Software Development Kit
2. For the user account system
   a) ddk_users
   b) ddk_sessions
   c) ddk_active_users
   d) ddk_active_guests
   e) ddk_banned_users

4.3.1 Tables For Didactic Methods

**ddk_methods**  This table contains all fields shown in the Detailed View section of the mobile app. In addition to the fields already listed in [4.2.7], there are four more fields: id (a unique number that identifies the method), folder (which is used to save the name of the folder the pictures are saved in), scope (to mark the method as global (0), user-defined private (1) or user-defined public (2)) and username (to save the user name of a user-defined method).

**ddk_favorites**  This table contains pairs of method id’s and user names. If an entry exists with method_id = 1 and username = johndoe it would mean that the user johndoe has marked the method with id = 1 as favorite. Every time a method is marked as favorite a new row is added to this table and every time a method is deleted or the favorite star is toggled off, the corresponding entry is deleted from this table.

4.3.2 Tables For The User Account System

The User Account System is a standard Hypertext Preprocessor [23] (PHP) User Account System that has been found in the literature [14] and has been heavily modified to fit the needs of the application.

**ddk_users**  This table holds the following fields: username, password, userlevel, email, timestamp, language, favoritesversion and dbversion. The fields username, password, email and language are set by the user when he/she signs up. userlevel can be any number between 0 and 9 (see table [4.1]). timestamp holds the last time the user was active. This is used in the Admin Center section of the website to delete users who haven’t been active for a certain time period. favoritesversion holds the current user-specific version
of the favorites table (`ddk_favorites`). It is increased by one each time a user marks or unmarks a method as favorite. `dbversion` holds the current version of the methods table (`ddk_methods`). When a method that is visible to everyone is added, edited or removed, the database version of every user is increased by one. When a method that is private, and thus visible only to one user, is added, edited or removed, the database version is increased by one only for that user. By doing this, we prevent the whole database to be sent to users who don’t see private methods. If a user updates a private method in the web page, this doesn’t trigger an update in the mobile app of other users, which is done to reduce network traffic.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Guest user (no permissions)</td>
</tr>
<tr>
<td>1</td>
<td>User with read-only permissions</td>
</tr>
<tr>
<td>2</td>
<td>User with edit permissions</td>
</tr>
<tr>
<td>3-8</td>
<td>For future use</td>
</tr>
<tr>
<td>9</td>
<td>Administrator (full control)</td>
</tr>
</tbody>
</table>

Table 4.1: User level

**ddk_sessions** This table holds information about user sessions. It contains the following fields: `username`, `userid`, `timestamp` and `time`. The field `username` contains the name of the users who have an open session, that is, that have signed in using the mobile app or the web page and haven’t signed out yet. Every time a user sends his/her user name and password to sign in, a unique `userid` is generated, saved in this table and sent back to the user. If the user signed in through the web page, it is saved locally as a cookie and sent back to the server with every page refresh. If the user signed in through the mobile app, it is saved in the local preferences and sent back to the server as a `POST` variable with every HTTP request. From this point on, the user only needs to send the `userid` and not the password anymore. This way the security risk of the password being stolen is reduced. The field `timestamp` holds the time the user was last active using this session. The field `time` is not used in the code and its only purpose is to display a human-readable timestamp for administrators who are reading the database directly.

**ddk_active_users** This table contains a pair of user names and timestamps. Its purpose is to hold a list of currently active users. A user is active if less than 10 minutes have passed since his/her last activity. With this information statistics can be shown on the web page, but this feature is not used at the date of this writing (March 2011).

---

[^1]: HTTP: Hypertext Transfer Protocol
**ddk_active_guests**  This table contains a pair of IP addresses and timestamps. Its purpose is to hold a list of currently active guest users. A guest user is active if less than 5 minutes have passed since his/her last activity. With this information statistics can be shown on the web page, but this feature is not used at the date of this writing (March 2011).

**ddk_banned_users**  This table contains a pair of user names and timestamps. Its purpose is to hold a list of currently banned users. A user can be banned by any administrator using the *Admin Center* section of the web page 4.4.11 after signing in. Banned users get an error when they try to sign in.

### 4.4 Web Server

The web server has an Apache HTTP server installation where a website is hosted. The website is programmed in **HTML**, **CSS**, **Javascript**, **Ajax**, and **XML** on the client side and in **PHP** on the server side.

It was implemented at the beginning for one reason: It allowed the administrators to add new methods to the database.

As the website evolved, it no longer served as an interface to add new methods, but became a complex system that had an integrated user account system, allowed not only administrators but also standard users to add new methods to the database and served as an alternative for those users who didn’t own a smartphone with the Android Operating System installed on it. The website provides also a means in form of an XML interface to access the data stored in the remote MySQL database from the mobile device.

The website is therefore the hub that makes the whole ecosystem work together. It was implemented using web technology mainly because doing so made it platform-independent and followed the cloud computing philosophy mentioned in the introduction (see ??).

The website is composed of following pages:

1. Log in
   a) Sign in (start page)
   b) Forgot password
   c) Sign up

---

5IP: Internet Protocol  
6HTML: HyperText Markup Language  
7CSS: Cascading Style Sheets  
8Ajax: Asynchronous JavaScript And XML  
9XML: Extensible Markup Language
2. Database viewing and handling
   a) List of methods (main page)
   b) View single method
   c) Edit single method
   d) Add new method

3. User Account
   a) View account
   b) Edit account

4. Administration
   a) Admin Center

4.4.1 Header

At the top of each page is the header and it allows the user to navigate to main parts of the website, e.g., to the Main page, to the Account page, to the Admin Center (for administrators). A Sign out link is also available to close the current session (see figure 4.9).

Figure 4.9: Website header

Under The Hood

Since the header is seen on every page it was implemented as a stand-alone module in header.inc.php. This allowed me to avoid redundancy in the code since it could be imported into any other file with a simple include command. The branding (Webapp name and logo) is done here. The navigation bar is implemented separately in topbar.inc.php, which is included inside header.inc.php.

4.4.2 Sign in

The first screen a user sees when accessing the website is the Sign in page (see figure 4.10). To sign in, the user would enter his/her user name and password in the corresponding text
Chapter 4 Technical Part

fields, check whether the session should be remembered for a maximum of 2 weeks and hit the Sign in button. If the user has forgotten his/her password, he/she can click on the link “Forgot password” and he/she would be redirected to the Forgot password page (see 4.4.3). If the user has not created an account yet, he/she can click on the link “Sign Up” and he/she would be redirected to the Sign up page (see 4.4.4).

Figure 4.10: Sign in Page

Under The Hood

The Sign in page is implemented in account/login.php. It starts including the file globals.inc.php where all needed constants are initiated and all important files are included, like MyDB.class.php (which handles all transactions with the MySQL database), account/session.php (which handles all user session related features) or functions.lib.php (which contains global functions used throughout the website).

After submitting the form, the information is sent to account/process.php. This file handles session-related actions like signing in, signing out, signing up, editing user accounts and sending new password to those who have forgotten it.

If signing in completed successfully, the user is redirected to the Main page, otherwise an error is shown to the user next to the field where the error occurred.

4.4.3 Forgot Password

The Forgot password page is a very simple one. It asks the user for his/her user name and upon submitting the form, it sends him/her a new generated password to the email address stored in his/her account (see figure 4.11).
Under The Hood

This page contains a form with a single input field for the user name. After submitting the form, the information is sent to account/process.php where the existence of the user name is checked. If the user name does not exist, an error is shown to the user. Otherwise, an email is sent using the convenience class implemented in account/include/mailer.php.

4.4.4 Sign up

The Sign up page is similar to the Sign in page. In addition to corresponding fields for the user name and password, two more fields are available to enter the email address and the preferred language (see figure 4.12). At the moment of this writing (March 2011) two languages are available: English and German, being English the default one. To add new languages to the interface, the variable $CONFIG['AVAILABLE_LANGUAGES'] in globals.inc.php needs to be updated accordingly to include a two letter identification for the new language. In addition to that, the file lang/<langId>.lang.php should be created containing a translation for all the words found in any of the other language files. E. g., if Spanish were to be included, “es” should be added to the variable $CONFIG['AVAILABLE_LANGUAGES'] and the file lang/es.lang.php should be created.
Under The Hood

The Sign up page is implemented in account/register.php. After submitting the form, all the information is sent to account/process.php as usual. Two new important classes intervene in this process: the Session class, which can be found in account/include/session.php, and the Database class, which can be found in account/include/database.php and handles all database-related operations only for the user account system, as opposed to MyDB.class.php, which is more of a general purpose class. Process calls the method Session.register(). Here the validity of the user input is checked, in order to return to the Sign up page in case any errors occurred. If no errors were found the method Database.addNewUser() is called. Here a new entry is added to the ddk_users table (see 4.3.2). If the operation completed successfully, a link is provided to return to the Sign in page and proceed to sign in.

4.4.5 List of Methods

The List of methods is the Main page. After the user signs in, this would be the next page he/she would see. This page is composed of three main parts (see figure 4.13): A toolbar on the left-hand side (which would allow the user to add, delete and export methods), a search box at the top (which can turn into an advanced search box to allow the user to make the same type of filtering enabled in the mobile app (see 4.2.6)) and a table containing the list of all methods (which has the same look-and-feel as the list of methods shown in the mobile app (see 4.2.4), but has more room for customizations).

Depending on the user permissions, different information is shown and different actions can be performed. E. g., standard users would only see global methods and user-defined methods that have been marked as public by the author. Users with read-only permissions would see the list of methods and, as opposed to users with edit permissions, they would not see the toolbar, since they were not allowed to perform actions like adding or deleting methods. Admin users on the other hand, have full permissions and would therefore be able to see all methods, as well as edit or delete any of them.

Under The Hood

Ajax. This page uses a lot of Javascript and Ajax technology. All necessary functions are implemented in functions.js, utils.js, ajax.js and ajax-new.js. The purpose of using Ajax was to make the page more responsive by reducing the number of page refreshes. The way Ajax works is the following: An XMLHttpRequest object is created, POST or GET variables are attached to it and then the request is sent to the server asynchronously in the background. In this case, the page that receives the request is request.php. Depending on which variables were passed, a different response is sent back to the client. The most
frequent action done then on the client side is to update a portion of the page with the received data using javascript. Although Javascript and Ajax is supported by all modern browsers, the website was programmed in a way that disabling Javascript would make the website return the same results at the expense of more page refreshes.

The actions that are performed using Ajax are those which require a refresh of the methods’ table:

1. Searching
Chapter 4 Technical Part

2. Adding new columns to the table
3. Sorting the table by column
4. Navigating through the pages of methods
5. Marking or unmarking a method as favorite

**Search**  There are two types of search: A basic search and an advanced search.

When a term is entered in the basic search field (see figure 4.14), the entered terms are searched in every column of the methods’ table in the MySQL database, following the same scheme as in the mobile app (see query 4.1). The query is updated with every entered character, it’s then sent to the server using Ajax and the result is displayed right below the search box without needing to refresh the page. This make the website feel fast from a usability point of view.

![Figure 4.14: Basic Search](image)

In the advanced search (see figure 4.15) two types of search fields are available, as in the mobile app: A general search box (which behaves exactly like the basic search) and a specific search box (which searches only within one column of the database’s table). Clicking on the green add icon (➕) would add a new search field to the advanced search box, and clicking on the red cross (❌) would delete the search field from the advanced search box and update the search results accordingly. The queries that are sent to the MySQL database are the same ones used in the mobile app (see queries 4.2 and 4.3)

![Figure 4.15: Advanced Search](image)

**Adding new columns to the table**  To add a new column to the methods’ table, the user would click on the right arrow next to the header of the last column (➡️). After clicking on the icon, a floating menu is displayed where the user can choose which columns to show. Clicking on a check box would trigger an Ajax call and the new column would be added on the fly behind the floating menu.
Figure 4.16: Add Column Menu

Under The Hood  The Ajax call would send the column names as **GET** variables (*columns[]*) to `request.php`. Here the column names would be inserted in the *columns* section of the **SELECT** query (see query 4.4).

```sql
```

Listing 4.4: Choose columns

Sorting the table by column  Clicking on the header of each column would sort the results alphabetically by clicked column. Clicking again would sort it in reverse order.

Under The Hood  The Ajax call would send the column name and the sort order as **GET** variables (*sortColumn* and *sortOrder*) to `request.php`. This information would then
be inserted in the ORDER BY section of the SELECT query (see query 4.5).

| SELECT [columns] FROM [table] WHERE [condition] ORDER BY <sortColumn> < sortOrder |

Listing 4.5: Sort by column

Choose columns

Navigating through the pages of methods The default number of methods listed per page is 20, as defined in globals.inc.php. If the number of methods stored in the database is larger than this default value, a bar is shown on top of the list to navigate between the different pages (see figure 4.17). Clicking on a number would take the user to the selected page, whereas clicking on “Previous” or “Next” would take the user to the page right before or after the current one.

Figure 4.17: Go to next or previous pages

Under The Hood The Ajax call would send the page offset and the number of methods to show on each page as GET variables (offset and rowCount) to request.php. This information would be inserted in the LIMIT section of the SELECT query (see query 4.6).


Listing 4.6: Sort by column

Choose columns

Marking or unmarking a method as favorite If the user wants to mark or unmark a method as favorite, he/she just needs to click on the star next to the method’s name. If the method is marked as favorite, the star changes from a gray one (_GRAY) to a yellow one (_YELLOW).

Under The Hood When a star is clicked, the id of the method is sent with the Ajax request to request.php. In request.php it is checked whether an entry with that id exists for the current user. In such a case, that entry is removed from the favorites table, otherwise a new entry is added. If the operation completed successfully the value “1” is sent back to the client, otherwise “0” is returned. On the client side, if a “1” was received, the star image is toggled.

4.4.6 View single method

The View method page shows all relevant information about a single method, similar to the Detailed view from the mobile app (see 4.2.7).
Under The Hood

The View method page needs a GET variable for the method’s id passed with the URI\textsuperscript{10}. If it’s not passed, the user is redirected to the Main page. If it’s passed, the corresponding entry is fetched from the database and stored in an array. The information is extracted from the array with a for-loop and printed as an HTML table.

As in the mobile app, this page has a gallery section (see figure 4.18). If a folder name is stored in the folder field of the database’s table, this page would retrieve all pictures saved under that folder and store them in an array. The pictures are then displayed in a thumbnail view using a for-loop. To show each picture in full-screen, the user can click on any of them and a slideshow view is activated (see figure 4.19). This slideshow is possible thanks to the Lightbox [10] script by Lokesh Dhakar, so all credit goes to him.

\textsuperscript{10}URL: Uniform Resource Locator

Figure 4.18: Web Gallery

Figure 4.19: Picture Slideshow

Keyboard shortcuts are activated for this page. Pressing the left or right arrow of the keyboard would allow the user to navigate through the list of methods without using the mouse. If the picture slideshow is active though, pressing the arrows would allow the user to navigate through the list of pictures instead.
4.4.7 Edit Method

The Edit method page is similar to the View method page. The only difference is that the information is not displayed as text, but populated inside the input fields of an HTML form. Assuming the user has edit permissions, he/she would be able to modify the values of each field. If the user has read-only permissions, he/she would be redirected to the View method page instead.

Under The Hood

Like the View method page, the Edit method page needs a GET variable for the method's id passed with the URL. If it's not passed, the user is redirected to the Main page. If the id is passed with the URL but the user has read-only permissions, he/she would be redirected to the View method page. Otherwise, the corresponding entry is fetched from the database and stored in an array. The information is extracted then from the array with a for-loop and printed as an HTML form, where each field is populated with the information from the database. After submitting the form the user would be redirected to the View method page where he/she could see the modifications just made.

For the gallery section an upload form is available (see figure 4.20). After clicking on the file input a dialog is shown to the user where he/she can choose a picture from his/her computer. To add more files the user can click on the “more” link and another file input would be added below the previous one using Javascript. Clicking on the “remove” link next to each file input would remove it from the upload form. If one of the pictures should be used as the method’s icon, the user has the possibility to mark the radio button next to the file input that says “Use as icon for this method”. After having chosen all pictures the user wants to upload, he/she can proceed to click on the “Upload” button. On the server side, the pictures would be saved under the path stored in the database. If no folder name is stored in the database yet, a new folder would be created and its name would be saved in the database for future use.

4.4.8 Add new method

The Add method page allows a user with edit-permissions to add a new method to the database. This page would therefore display an empty form, where the user can enter text or choose entries from predefined drop-down input fields. After submitting the form, the user is redirected to the View method page, where he/she can verify that everything was entered correctly and where he/she can proceed to add pictures.
Figure 4.20: Upload form

Under The Hood

The Add method page is a variation of the Edit method page. So please refer to section 4.4.7 to get more details on how it works. The main differences are that no id variable needs to be passed with the URL and that no upload form is available for the gallery section. For the picture gallery the existence of an entry in the database is required in order to update the folder field upon submitting the upload form. Therefore, the ability to add pictures is made possible in the next screen (the View method page).

4.4.9 View Account

The View account page is a simple one, where the user can see all information regarding his/her user account, e.g., user name, email address and language. To edit this information, a link to the Edit account page 4.4.10 is provided.

4.4.10 Edit account

The Edit account page contains an HTML form populated with the user account information. Here the user can change his/her password, email address or the language of the website (see figure 4.21).

Under The Hood

After submitting the form, the information is sent to account/process.php. Here the method Session.editAccount() from account/session.php is called, where the validity of the user input is checked. If the user input is not valid an error is shown to the user. If it is,
the method `Database.updateUserField()` from `account/database.php` is called where the information is updated in the database (see 4.3.2).

### 4.4.11 Admin Center

The *Admin Center* is a page that is only available to the administrators and can be accessed by clicking on the “Admin Center” link that appears in the upper right section of the header (see 4.4.1). Here an administrator can manage the user accounts and perform actions like updating user levels, deleting users, deleting inactive users, banning users or deleting banned users.

**Under The Hood**

This page is composed of a set of HTML forms. Every HTML form sends the entered information to `account/admin/adminprocess.php`. Depending on which form has been submitted, different methods are called.

**AdminProcess.procUpdateLevel()**  This method would update the `userlevel` field of table `ddk_users` (see 4.3.2).

**AdminProcess.procDeleteUser()**  This method would delete the entry for the selected user from the table `ddk_users`.

**AdminProcess.procDeleteInactive()**  This method would delete all entries from the table `ddk_users` where the `timestamp` field, that is, the last time the user was active, is older than a certain time period chosen in the *Admin Center* page.
AdminProcess.procBanUser()  
This method would move a user entry from the ddk_users table to the ddk_banned_users table (see 4.3.2).

AdminProcess.procDeleteBannedUser()  
This method would delete a certain user from the ddk_banned_users table.

4.5 Communication Between Mobile App and Website

The mobile app needs to connect to the server for certain tasks. Those tasks are:

1. Check if user is remotely signed in
2. Sign in
3. Sign out
4. Sign up
5. Retrieve data from the MySQL database
6. Send and receive favorites
7. Download remote pictures

The communication takes place thanks to an XML interface, which is implemented in the file interface.php. The general mechanism is the following: The mobile app sends an HTTP request to interface.php with certain GET and POST variables attached to it. Depending on the attached variables, the interface extracts the necessary data from the MySQL database and prints it as an XML file. When the mobile app receives the XML file, the information is extracted using an XML Parser and, depending on the type of request, the corresponding task is performed.

All this tasks are performed either using the AsyncTask class provided in the Android SDK or the Java Thread class. The main reason is that downloading information from a remote server can take a considerable amount of time and the GUI would freeze until the data is completely downloaded. This tasks are contained in their own package de.tum.lmt.supportinglearning.asynctasks

Check if user is signed in

The Sign in screen is the one that is started when the mobile app is launched. Therefore it is here where the user credentials are checked. If the “remember” check box was marked in a previous session, the CheckLoginTask class is instantiated and executed in the background. This would show a “Loading” dialog while the stored user name and user id are sent to the
server. If the server replies with a SUCCESS message, the user is redirected to the Home screen, otherwise a dialog is shown with the errors returned by the server for authentication.

**Sign in**

If the user credentials haven’t been stored yet, empty input fields are shown in the Sign in screen. The user would enter his/her user credentials and upon clicking the Sign in button, this information would be sent to the server. If the server replies with a SUCCESS message, the returned user id is stored locally together with the user name and password. Otherwise a dialog is shown with the errors returned by the server. From this point on, the user id, and not the password, is sent together with the user name to the server.

**Sign out**

When the user clicks the MENU button, a menu pops out with an entry to sign out. Tapping on this menu entry would send the user name and user id to the server, which would in turn delete that session from the MySQL database. If this completes successfully, a SUCCESS message is returned to the mobile app and there all open screens would be closed except for the Sign in screen. If the server responds with an ERROR message, a dialog with that message is shown.

**Sign up**

In the Sign up screen, when the user enters his/her information and clicks on the button to sign up, the user credentials are sent to the server. There the sign up task is performed. If the operation completes successfully, the server signs the user in automatically, then a welcome message is sent to the given email address and a SUCCESS message is returned to the mobile app. The mobile app would then redirect the user to the Home screen. If an error occurred or the user input was not valid, then an error is shown giving more information.

**Retrieve data from the MySQL database**

Each time a user signs in in the mobile app, it is checked whether changes have been made to the methods’ database. For that, the mobile app sends the version of the database that is stored in the mobile device. The server checks then whether that version is older than the version stored in the server. If that’s the case, the new database is sent to the mobile app, which would replace the old information with the new one and then display it.
**Send and receive favorites**

For the favorites, the same procedure as in the above paragraph is used. The mobile app sends the version of the favorites version with the HTTP request to the server. The server checks then whether that version is older than the version stored in the server. If that’s the case, the list of favorites is sent to the mobile app, which would update local database correspondingly.

Since the user is also able to mark methods as favorite being offline, that is, while no connection to the server is available, every time the user does so, the method id and its “favorite” flag are saved in a local SQLite database. The next time the database is refreshed, either by restarting the mobile app or by clicking on the *Sync* button of the options menu, the offline favorites are sent first to the server before retrieving the new data.

**Download remote pictures**

Each method can have an icon and a set of pictures assigned to it. There are two locations where the icons or the pictures are downloaded.

One of them is the *Detailed view* screen. Here the icons and pictures are downloaded with the class `ImageThreadLoader` which is a combination of various examples found in the literature [24, 26] and modified to meet the application requirements. This class downloads the images in a background thread so that the mobile app doesn’t freeze while the pictures are being downloaded.

The other location is the *Home* screen. In this screen the icons are downloaded with the same mechanism as in the *Detailed view* screen. The images however are downloaded in a different manner. As mentioned in the *Home* screen section (4.2.4), a menu is shown when the user clicks the menu hardware button. After clicking on the menu item to sync the database, a dialog pops up to let the user choose whether he/she wants to download the pictures of all methods or not. Tapping on “Yes” will start a *Service* implemented in `DownloadPicturesService.java`. This *Service* will show a notification in the *Status bar* where the user can see the progress of the download. Meanwhile the user can interact with the app without interfering in any way with the download task. When the download is complete, a new notification is displayed in the *Status bar* to let the user know.

In the *Settings* screen the user can choose whether to store the images in the SD card or not. If the user has marked this check box, the icons and images are downloaded only the first time. The following times, it is only checked whether new pictures are available. If that’s the case, only the new pictures are downloaded while the old ones are loaded directly from the SD card. If the user has chosen not to save the images in the SD card, all icons and images are loaded on the fly from the server every time the corresponding *Activity* is restarted.
Chapter 5

Possible Improvements

Although the system has evolved to become a robust and complex one, there’s still plenty of room for improvements. In this chapter, we are going to go through a list of different enhancements which can be subject for a future expansion.

5.1 Recommendation system

Using information about what methods have been marked as favorites, different algorithms can be applied to come up with a recommendation system. This recommendations could be added as a new section, both in the website and in the mobile app, where the user could navigate through a list of methods which could be of interest to that specific user based on what other methods he/she has marked as favorites.

5.2 Sorting by popularity

As of now the methods are sorted either alphabetically or with the order they were created in. This can be useful if the user wants to scroll manually to a method whose name is known to the him/her, but the user can be interested in some method he/she doesn’t even know they exist. It would be interesting then to show popular methods first and less popular methods at the end so that the user can see the interesting ones right away. To achieve this, information can also be extracted from the favorites or ratings other users have given to certain methods.

5.3 User Interaction

Currently only little user interaction is supported. A user can choose to share methods publicly for other users to see, but apart from that there is really no communication possible
between users. This could therefore be subject for a future enhancement of the system. A commenting system, e.g., could be added to allow users to review methods, so that others can have different views and opinions on them. The sharing capabilities could be improved to allow users to share methods only with a reduced number of users. Some sort of messaging or chat system could be added to allow users to talk to each other about different didactic methods.

5.4 Optimization

In addition to the enhancements mentioned above there is also room for optimizations in the way the current system works. Those optimizations have to do with the amount of network traffic that is generated, the amount of storage space that is used and the performance of the user interface.

To reduce the amount of network traffic that is generated different approaches can be taken. For example, a more complex database versioning system would allow to send to each mobile device only those methods which have been added or modified, instead of sending the whole database each time any modification is made. Another approach could be to combine multiple HTTP requests into one, so that multiple tasks can be performed after sending one single request.

To reduce the amount of storage space used by the mobile app the following could be considered. If the user has chosen not to store data in the SD card, the images are loaded every time directly from the server with every restart of the Activity that contains them. This way the images are only loaded in RAM\textsuperscript{1} while they are being shown but deleted while they are not. This uses the minimum amount of storage but generates the maximum amount of network traffic, so a middle point has to be met. The internal cache directory can be used, but since only using 1MB is the recommended practice, it is not enough to hold all pictures. For this reason, one approach could be reducing the size of the images before sending them. This would not only reduce storage space but also network traffic. In the gallery section (both in the website and the mobile app), this would make the loading of the thumbnails quicker contributing to an increase in performance.

5.5 User Base Expansion

To increase the number of users that have ubiquitous access to the didactic content, mobile apps for the rest of the main mobile platforms (iOS, Symbian, BlackBerry OS, Windows Phone) can be developed. To get an overview, just by developing an app for the first two

\textsuperscript{1}RAM: Random-Access Memory
(iOS and Symbian) the 79.5 % of the market share would be covered [8] as depicted in figure 5.1.

Figure 5.1: Smartphone Market Share (Q4 2010)
Chapter 6

Conclusion

The problem that we were trying to solve was the lack of a system to manage and share didactic content. For that we have built an information system that is complex on the inside, thanks to modern ways to access information and to current technologies in network communications, but familiar to the user on the outside, thanks to trendy designs for mobile app interfaces and websites.

Now we have achieved the necessary infrastructure to focus on the content, rather than on the tools, and we can begin to analyze the relationship between the didactic methods and the users. We can use the information gathered from the usage of the application, from the didactic methods that are clicked the most, from the favorites, from the ratings and from the similarities between each didactic method, to let the methods find their way to the user, rather than the other way around. This will hopefully open a new door where it’s not the users that find information, but it’s the information that finds the users.

So, as we can see, this is a first but very important step towards a bigger vision. This will hopefully help professors get better at what they do and help students get a better understanding of what is being taught to them.
List of Figures

3.1 Heterogeneity of the Statistical Population ........................................ 10
3.2 Own a Smartphone and Would Use the Mobile App ............................... 11
3.3 Smartphone Features And Methods' Information Sources ....................... 11
3.4 What features would be used the most ............................................ 12

4.1 Design Alternatives (Tabs, List, ActionBar) .......................................... 14
4.2 Sign In and Sign Up Screen .............................................................. 16
4.3 Home and Favorites Screen ............................................................... 18
4.4 Different Approaches for the Search Screen ......................................... 20
4.5 Detailed View and Content Menu ....................................................... 22
4.6 User-defined Method .......................................................... 23
4.7 Image Gallery ..................................................................... 24
4.8 Settings Screen .................................................................... 26
4.9 Website header ......................................................................... 30
4.10 Sign in Page .......................................................................... 31
4.11 Forgot Password Page .............................................................. 32
4.12 Sign Up Page ........................................................................ 32
4.13 List of Methods (Main Page) ....................................................... 34
4.14 Basic Search ........................................................................ 35
4.15 Advanced Search ..................................................................... 35
4.16 Add Column Menu .................................................................. 36
4.17 Go to next or previous pages ......................................................... 37
4.18 Web Gallery .......................................................................... 38
4.19 Picture Slideshow ................................................................... 38
4.20 Upload form .......................................................................... 40
4.21 Edit User Account ................................................................... 41

5.1 Smartphone Market Share (Q4 2010) .................................................... 47
List of Tables

4.1 User level ......................................................... 28
Bibliography


