Multimedia Report Forum Application

Multimedia modelleren en programmeren

Joris Schelfaut

Katholieke Universiteit Leuven

Contents

Chapter 1

Introduction

1.1 About the course

The course *Multimedia: modelleren & pogrammeren* is given by prof. E. Duval at KULeuven. The objective of this course to create a mobile application using different technologies, namely `iOS`, `Android` and `HTML5`, to eventually gain insight in the advantages and disadvantages of each technology.

The technologies that will be discussed on this blog are `iOS` and `Android`. All code for this project is open source and can be found on and downloaded from Github[1]. The progress of the development is communicated through a blog[2].

1.2 About the application

The application is an online forum where the main question of each tread is directed at one expert or a group of experts. Each member can enlist him/herself as an expert in certain areas and will get notified when a question is posted, related to his/her domain of expertise.

Although similar websites already exist, e.g. *Stackoverflow*[3] and *Yahoo! Answers*[4], one idea would be to direct the application rather at students than at the general public. The idea arose when looking at the rather limited use of the fora on *Toledo*[5].

1.3 Structure of this text

The next chapter looks at the idea behind the application in more detail. We use tools such as user stories, story boards, use case diagrams, and screen transition diagrams to get an understanding of the application from a user's perspective. From these functional requirements, a number of general requirements for the software can be derived.

These requirements serve as the input for chapters 3 and 4. In these chapter we will determine how the architecture of the application is constructed on three levels: a global scope, the data itself, and each app individually.

Chapter 5 looks at different aspects of each mobile technology and tries to compare them.

Finally chapter 6 looks back at the project, points presented in the discussion, and the course as a whole.

[1] `https://github.com/mumedev`
[2] `http://mumedev.wordpress.com/`
[3] `http://stackoverflow.com/`
[4] `http://answers.yahoo.com/`
[5] `https://toledo.kuleuven.be`

Chapter 2

Requirement analysis

2.1 User story and storyboard

The use of the application can be illustrated by the following user story. Figure 2.1 depicts the same story line.

Jake is a student at the department of computer science at KULeuven. For his project of Multimedia he has to develop an Android application. Unfortunately he has encountered a particular problem as he was working on the application. He decides to look for some help.

He starts the application and enters his question. He also attaches a number of specialities as tags to his post. Users that are listed as specialists in these technologies are then notified and can start working on a solution.

Several solutions are proposed. Jake finds one that works and is able to continue with his work.

2.2 Use case diagram

The use case diagram for the application is shown in figure 2.2. Each use case is elaborated in tables 2.1, 2.2 and 2.3.

2.3 Screen transition diagram

2.4 Summary

The following tries to summarize some of the requirements to support previously described functionality:

Figure 2.1: Story board depicting a potential scenario of use.

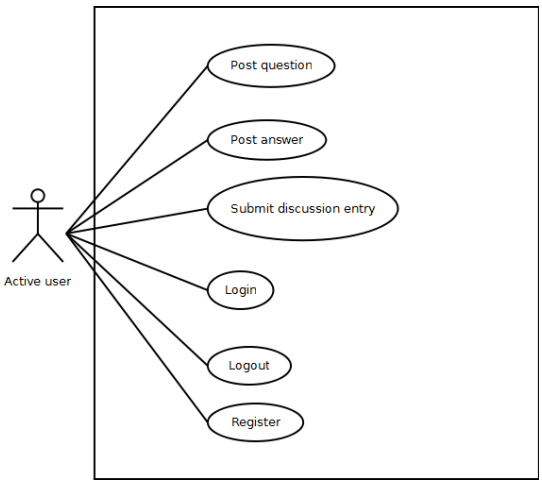

Figure 2.2: The use case diagram showing the functionality in the system from a user's perspective.

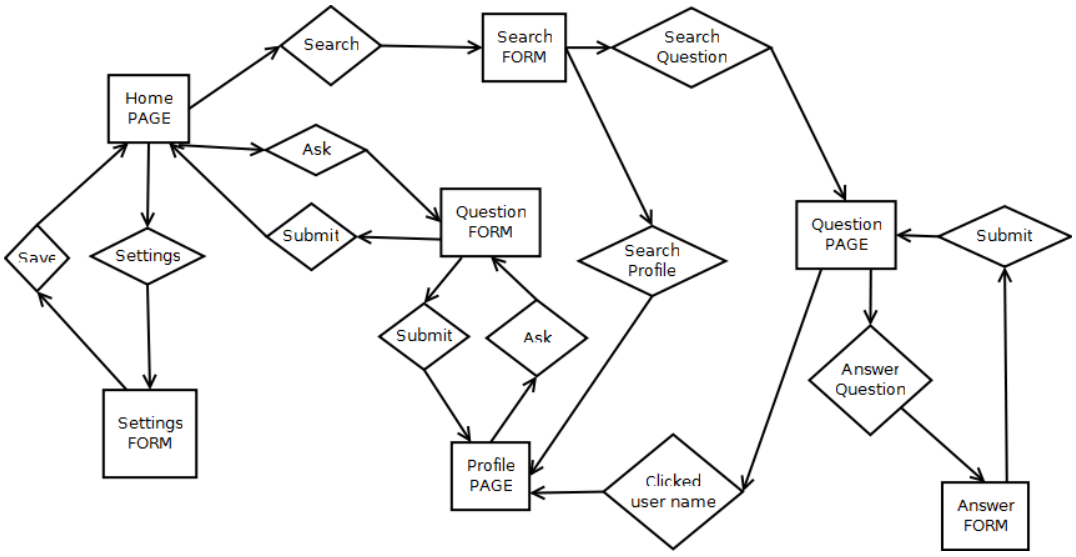

Figure 2.3: The transitions between the screens.

Table 2.1: Use case 1 *Post question*

Primary actor:	Active user
Preconditions:	User is logged in;
Basic flow:	(1) The question form is loaded.
	(2) The user enters his question.
	(3) The user selects recipients, including specific people and groups.
	(4) The user submits the form data.
	(5) The systems shows a confirmation message.

Table 2.2: Use case 2 *Post answer*

Primary actor:	Active user
Preconditions:	User is logged in;
	The user has selected a question to answer.
	User is allowed to participate in the discussion.
Basic flow:	(1) The user selects an option to answer the question.
	(2) The user enters his/her answer.
	(3) The user submits the answer.
	(4) The systems shows a confirmation message.

- Users have to be able to interact with each other via the app on their device;
- Users can be identified through a unique set of credentials;
- Users have to be able to post questions and answers in text format, or if possible, other types of multimedia;
- Data is persisted across sessions;
- Users receive updates when new data is available, or alternatively can request for updates to be downloaded, e.g. by refreshing the page;

Table 2.3: Use case 3 *Take part in discussion.*

Primary actor:	Active user
Preconditions:	The user is logged in; The user has found a question he/she would like to provide further comments on; The user is allowed to participate in the discussion.
Basic flow:	(1) The user selects an option to create a discussion entry. (2) The user types in the discussion entry. (3) The user submits the entry. (4) The entry is added to the discussion.

Chapter 3

Architecture

3.1 Architectural challenges

Before discussing the end result, we will try to give an overview of some of the challenges that need to be addressed when designing mobile applications. One of the central questions when building any application is how to organize information. Trade-offs in the design of data organization go beyond the level of the data itself. For example the physical location of data, i.e., whether data is kept locally or stored remotely. Availability, fault-tolerance and Quality of service (Qos), which covers reliability, security and performance, are other aspects that can be taken into account[].

To support interactions between users on different devices, some data will be sent over the network. Several communication paradigms exist, such as direct or indirect communication. The important difference here is that in the second case an additional layer of indirection exists, decoupling the producers and consumers of information streams[2]. These communicating entities can also take on different roles, from a *client-server* approach were clients are consumers and servers are producers, to a *peer-to-peer* (P2P) architecture where conceptually only one type of role exists[2].

In a distributed environment software engineers may have to deal with a degree of heterogeneity both in software and hardware. As a result layers of transparency can be introduced to reduce complexity of the underlying structures.

Another challenge of the application that we are building is inherent to *mobile computing*. Coulouris et al. define mobile computing as "the performance of computing tasks while the user is on the move, or visiting places other than their usual environment"[2]. This poses a difficulty, as the quality of the connection may fluctuate, devices may be disconnected and reconnected as the user moves, or temporary go offline for an undefined period of time[2]. Although this is an important aspect, it is not a major concern of the application as changes in the data may take several minutes to hours or days to take place. However, when creating data, it might be a good idea to give an option to store local drafts; for example when the connection is lost.

3.2 Architecture design

The roles of the communicating entities correspond to the client-server paradigm. The choice for this architecture is to provide a single point to access stored data. Storing all data locally and exchanging updates is possible, but would be a lot more complicated. Based on the characteristics of peer-to-peer systems, replication, CPU usage, and network traffic, introduce significant constraints on the mobile device's performance. Bandwidth, local storage and CPU are all much more limited than in home computers. In addition, due to their mobile nature, the physical layout of the network may change constantly. As a result, a client-server approach seems the logical choice. The global architecture is shown in figure 3.1.

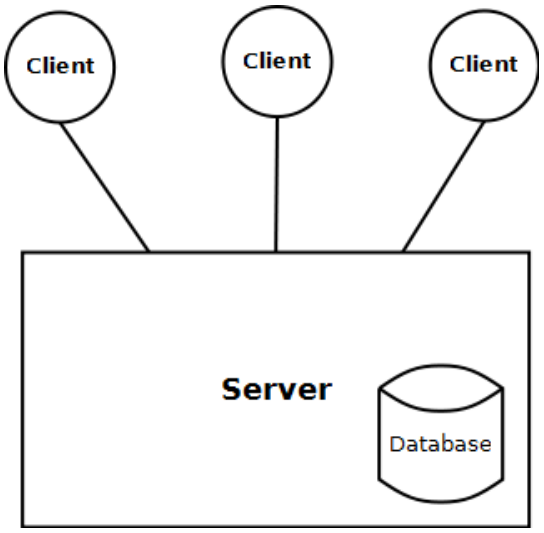

Figure 3.1: The global architecture of the application.

To allow access to the database, each application can talk to a public interface. For example, in listing 3.1 is shown how one could use PHP scripts to access the database, using URLs to pass arguments. This kind of approach is

Listing 3.1: Example of a PHP script to insert data into the database (simplification).

```
1  <?php
2  // http://example.com?method=insert&table=user&username=joris
3
4  if ($_GET['method'] == 'insert') {
5          $database->open();
6          $id = $database->insert($_GET['table'], array(
7                  'username' => $_GET['username']
8          ));
9          $database->close();
10         echo $id;
11 } else echo '-1';
```

In this case, REST (Representational State Transfer) was used to implement this point of public access. The REST approach uses HTTP operations GET, PUT, DELETE and POST to manipulate resources represented in XML[2]. The API shields the underlying data structures from the clients, organizes the application's data model into resources and can be accessed through a standardized protocol.

The REST API used here, consists out of a series of methods, as listed in table A.1. Five resources are provided: *answer*, *authentication*, *question*, *skill*, and *user*.

Chapter 4

Implementation

4.1 Data model

To structure the information on the level of the data, different alternatives can be conceived to represent the data made available through the public API. For example, conversations can be represented as `XML`, as shown in listing 4.1. Operations can then be implemented to append or remove elements.

Listing 4.1: Example of an `XML` data representation.

```
<?xml version="1.0"?>
<conversation channels="sql,php,codeigniter">
        <author>joris</author>
        <date>09-02-2013 18:52:00</date>
        <question>How do I ...</question>
        <answers>
                <answer>
                        <author>sander</author>
                        <date>09-02-2013 19:28:00</date>
                        <text>
                                Use this as the ...
                        </text>
                <answer/>
                <answer>
                        ...

        <answers/>
</conversation>
```

Alternatively, a NoSQL database can be designed, for example using the Datastore on Google App Engine (GAE).

For this application, a relational database was implemented using `MySQL`. Figure 4.1 shows the entity relationship diagram (ERD) of the entities in the database. As the backend of the application was created with `PHP`, *PHPMyAdmin*[1] was used to create the database.

4.1.1 Remarks

Although the relational database certainly will do the job in this case, NoSQL databases scale better when huge quantities of data are involved. As a result, if this app would be used by a great number of students, it is not unlikely that this design choice may form a bottleneck.

[1] `http://www.phpmyadmin.net/home_page/index.php`

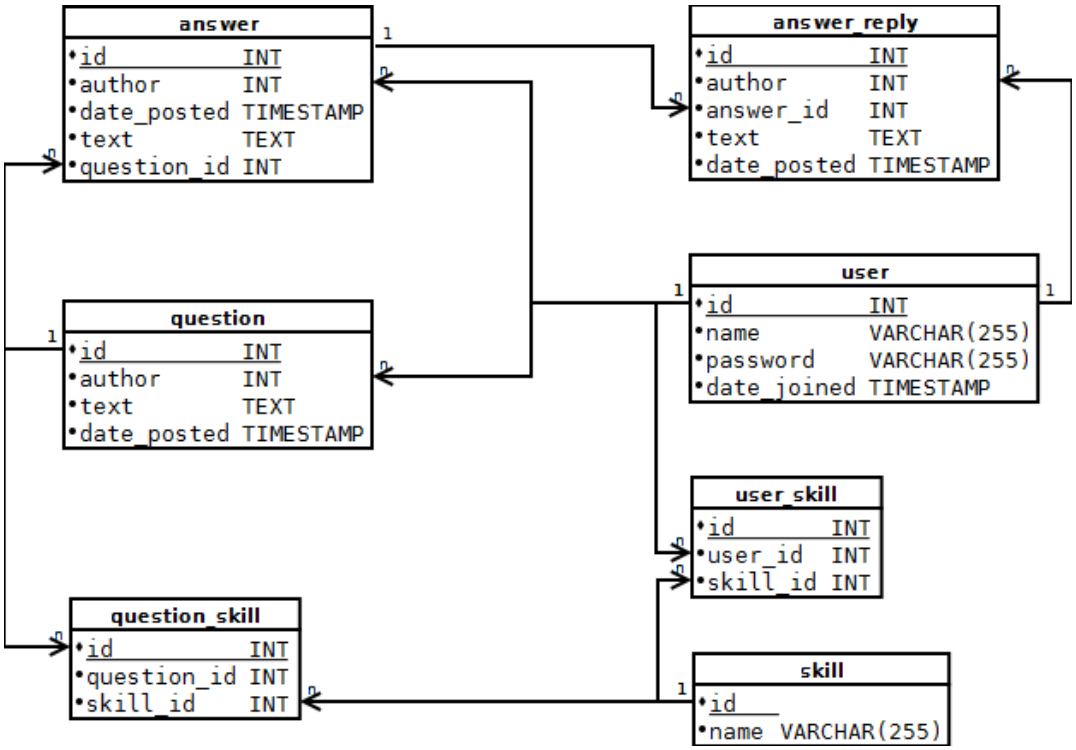

Figure 4.1: Entity relationship diagram of the database.

4.2 Server

The REST API is implemented using the codeigniter framework. Codeigniter[2] is a `PHP` framework that relies heavily on the model-view-controller design principles. Via model classes the database can be accessed. Controllers can load model classes to pass on the data to the views. A special controller, `Api`, which extends the `REST_Controller`[3] class, implements the REST service.

Each method listed in table A.1 is then implemented in the `Api` class. Each method signature has as a suffix an underscore followed by the HTTP method name, e.g. for the method *create* and HTTP method POST, this results in *create_post*. The class diagram of the backend of the application is shown in figure 4.2. Figure 4.3 summarize the application's internal structure as discussed so far.

Alternative libraries for REST and PHP exist, but also for other technologies, such as GAE. For example the Boomi Appengine REST Server[4] is a library for Google App Engine applications. Of course, this would also require a somewhat different underlying data model as mentioned earlier.

4.2.1 Remarks

When designing an API as described above, it is important to ensure that the data remains consistent. In this case, the login functionality forms a possible danger in that respect, as a user may log in from multiple apps simultaneously, and the server doesn't keep track of the number of logged in apps per user account. At the moment the session key is required to ensure that no one else log out any other user. Instead, a solution might be to create a separate table to keep track of these sessions, rather than keep a single session key entry in the user table.

[2] `http://ellislab.com/codeigniter`
[3] The source code can be found at `https://github.com/philsturgeon/codeigniter-restserver`
[4] `https://code.google.com/p/appengine-rest-server/`

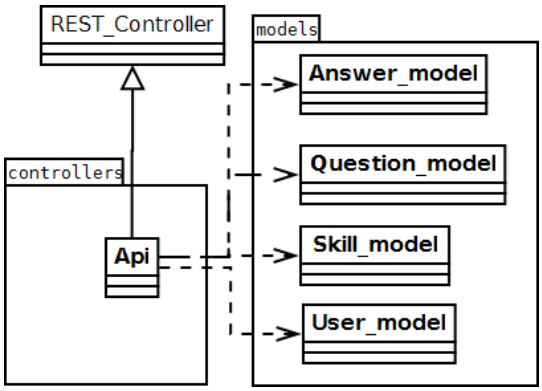

Figure 4.2: The class diagram of the codeigniter project.

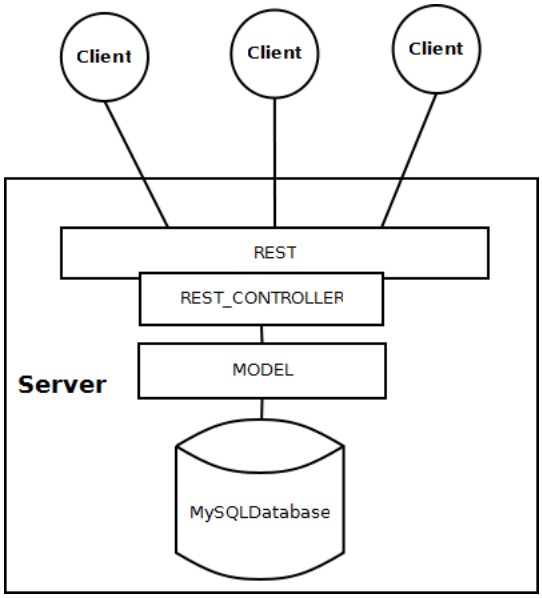

Figure 4.3: The global architecture of the application, including REST, the Codeigniter MVC design and the MySQL database.

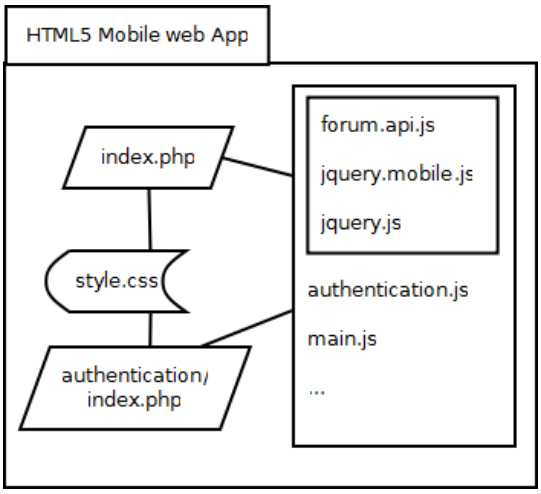

Figure 4.4: The internal structure of the HTML5 mobile web app.

If we were to create only the mobile web app, which depending on the type of application, is quite reasonable, implementing a REST API may seem like overkill. Instead a simpler design could be used to persist data. Nonetheless, when developing multiple native applications, the cost savings of creating a back-end to process parts of the data may be significant. For example, graphical operations that require a lot of computational power may be done on the device itself, while operations on the data may be done on the server.

Coming back to the point raised earlier on the performance bottleneck of a relational database. Suppose this is the case, having a well-defined API provides transparency, as the clients depend on the public interface, rather than how everything works under the hood.

4.3 Clients

Now that we have discussed the server side of the architecture, we will take a closer look at the different clients. For this project, very few code was actually written at the client side. Eventually it has become merely an experiment to test the client-server architecture, rather than an elaborate implementation of the application as described in chapter 2. Nonetheless, we will focus further on how the final elements of the architecture fall into place for each app.

4.3.1 HTML5 mobile web application

The `HTML5` mobile web application is basically implemented as an `HTML` web page enhanced with the jQuery mobile library[5]. The internal architecture consists out of JavaScript, HTML and CSS. The JavaScript code consists out of a number of libraries and source code making the connection to the back-end and controlling the view. The jQuery mobile library provides a layer on top of the original JavaScript to facilitate operations for modifying the view and the use of AJAX. A second library mirrors the REST API methods and helps to provide additional transparency when making asynchronous calls to the server. The resulting application structure is shown in figure 4.4.

Listing 4.2 shows how the connection is made to the server in the api library. Listing 4.3 shows how it is used in the application. An asynchronous call is made to the server. Once the result is available the anonymous function on line 5 to 13 in listing 4.3 is executed.

[5] `http://jquerymobile.com/`

Listing 4.2: A method from the JavaScript API library to make the connection to the server.

```
Forum = function () {
    var _API = 'http://localhost/forum-server-restapi/index.php/api/';
    this.authentication = {
        startsession : function (id, username, password, callback) {
           var data = {};
           if (id)       data['id']       = id;
           if (username) data['username'] = username;
           if (password) data['password'] = password;
           $.ajax({
               url      : _API + 'authentication_startsession/',
               dataType : 'json',
               type     : 'get',
               data     : data
           }).done(callback);
        },
```

Listing 4.3: Calling a method from the JavaScript API library.

```
var _FORUM = new Forum();
function login() {
    var username = $('#login-form input:text#username').val();
    var password = $('#login-form input:password#password').val();
    _FORUM.authentication.startsession(null, username, password,
                function(key) {
                        var obj = {
                                username : username,
                                key      : key
                        };
                        localStorage.setObject('forum_user', obj);
                        window.location.href = '../';
                });
};
```

4.3.2 Android application

The Android application follows a similar design. A seperate collection of classes provides transparent access to the REST methods of the server, as shown in figure 4.5. Instead of passing an anonymous function as parameter, a specific listener is implemented for each asynchronous operation. Once the data is available, the notify method of the listener is called. The custom implementation of this method deals with the result from the request. This is shown in listings 4.4, 4.5 and 4.6. The corresponding class diagram is shown in figure 4.6.

Listing 4.4: AnsyncTask to execute an asynchronous request to the REST server.

```
class StartSession extends AsyncTask<String, Void, String> {
        private StartSessionListener listener; ...
        StartSession(StartSessionListener listener, ...) {
                super ();
                this.listener = listener; ...
        }
        @Override
        protected String doInBackground(String... params) {
                String sessionkey = null;
                try {
```

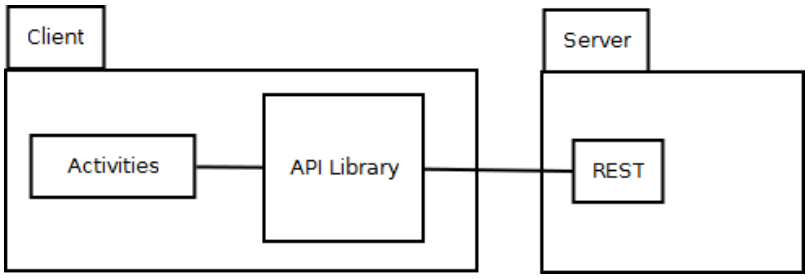

Figure 4.5: The internal structure of the Android native app.

```
11                          String httprequest = ForumAPI.BASE_URL
12                                  + "authentication_startsession/"
13                                  + ...
14
15                          HttpGet httpGET = new HttpGet(httprequest);
16                          HttpResponse response = getHttpClient()
17                                  .execute(httpGET, getHttpContext());
18                          sessionkey = EntityUtils
19                                  .toString(response.getEntity());
20                  } catch (IOException e) {
21                          ...
22                  }
23                  return sessionkey;
24          }
25          @Override
26          protected void onPostExecute(String key) {
27                  super.onPostExecute(key);
28                  if(this.listener != null)
29                          this.listener.handleStartSession (key);
30          }
31  }
```

Listing 4.5: Listener for the StartSession task.

```
1  public interface StartSessionListener {
2          void handleStartSession (String key);
3  }
```

Listing 4.6: Activity that implements the StartSession listener.

```
1  public class MainActivity extends Activity
2          implements StartSessionListener
3  {
4          @Override
5          protected void onCreate(Bundle savedInstanceState) {
6                  super.onCreate(savedInstanceState);
7                  setContentView(R.layout.activity_main);
8                  new ForumAPI().authentication()
9                          .startSession(this, -1, "joris", "joris");
10         }
11         @Override
12         public void handleStartSession(String key) {
```

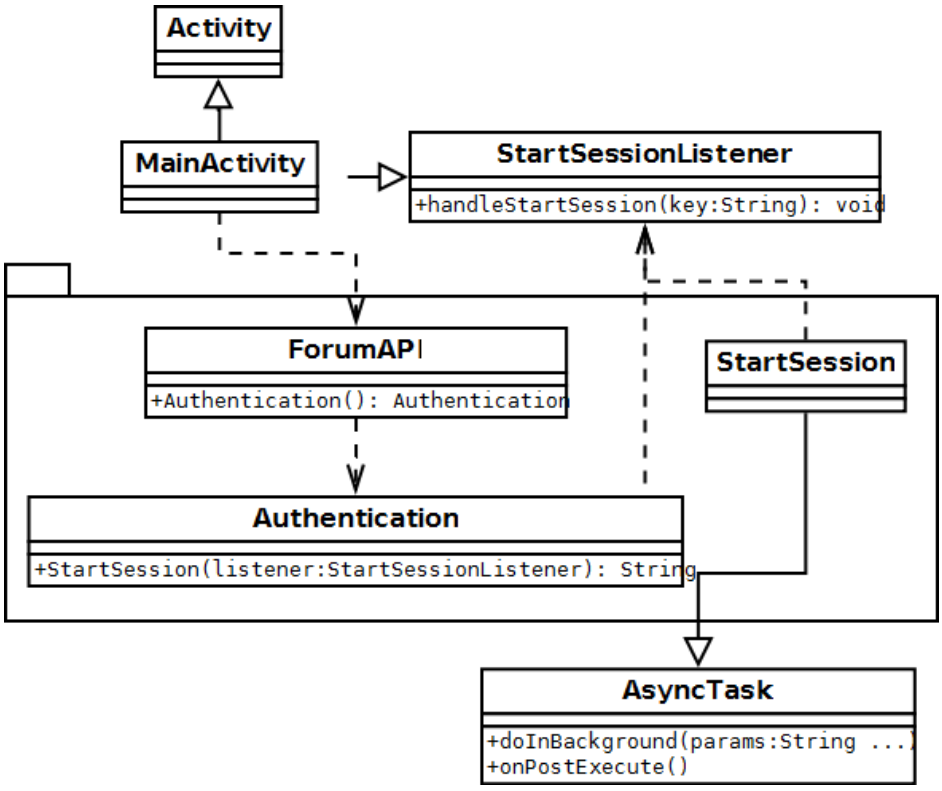

Figure 4.6: Class diagram for the Android app.

```
13                  // do something ...
14          }
15  }
```

4.3.3 iOS application

Unfortunately no implementation for iOS has been completed. The global structure of the class diagram for the application would have been relatively similar to the Android application, cf. figure 4.5, again with a separate library, which can be implemented in iOS through a static library[6], or model classes that are mapped onto the REST service's resources.

To work with REST on iOS, the RESTKit[7] library can be used for this purpose.

[6] `https://developer.apple.com/library/ios/technotes/iOSStaticLibraries/Introduction.html`

[7] `http://restkit.org/`; a recent tutorial can be found here: `http://www.raywenderlich.com/13097/intro-to-restkit-tutorial`

Chapter 5

Comparing mobile technologies

5.1 Web versus native mobile applications

In [1] a table is presented of the required skill set to be able to write mobile applications for each platform. From the nine mobile technologies that were included, the market shares for Android and iOS operating systems are 68.3% and 18.8% respectively, clearly dominating the market [3]. Numbers suggest that the Windows Phone will establish itself as another giant on the market, however still considerably less significant than Android or iOS. Even though this suggests development of native applications can be reduced to development for two to three technologies, the cost of developing and maintaining applications in these technologies would still create a considerable overhead. Charland et al. [1] look at mobile web technology to address this issue of heterogeneity.

The 'Phonegap hack' has grown from the fact that all mobile operating systems are equipped with a mobile browser, in which the native API can be called by using JavaScript. Unfortunately there are differences between the Webkit implementations of these browsers. In recent years many of these issues are being addressed through various libraries, e.g. jQuery Mobile, Sencha Touch, DaVinchi Studio, and Wink[1, 4]. Also, the `W3C` has a device API working group that is working to bridge the gap between lower level native APIs and web technology [1]. JavaScript virtual machine technology is quickly getting more powerful, driven by competition between browsers. In the end, in [1] is stated that 'if you want to add a native capability to a browser, then you can either bridge it or recompile the browser to achieve that capability'.

> *If a browser does not support a native capability, it's not because it can't or that it won't; it just means it hasn't been done yet (Charland et al., 2011).*

A number of trade-offs between web and native mobile applications have to be taken into account. First of all, the size of the code base certainly affects the required development time. The code base for a mobile web application will be significantly smaller than the code base for two or more native applications[1]. On the other hand, using a server to process transactions to the database as described in the previous chapters, alleviates the need to develop complicated data management systems for each app separately. Also, by providing a public API, third parties may be tempted to write their own clients, possibly increasing the value of the data and application as a whole.

Related to the size of the code base is the efficiency of maintenance, as the cost of maintenance may be directly related to the number of versions of an application that need to be updated[1].

On the other hand native applications still outperform mobile web applications. In this respect the application objectives should be taken into careful consideration, as nice looking visuals do not fulfill business requirements, but may require significantly more processing power. Although web applications are aimed at supporting multiple platforms, creating only one application for different platforms, may require a lot of conditional statements to provide user interface code that is conform with the guidelines for each seperate platform[1].

All in all, the main trade-off seems to be one in terms of development time and maintenance against performance and look & feel. Although, Charland et al. suggest that the performance gap is getting smaller over time, as of now utility applications with low graphic processing lend themselves well for mobile web apps, while demanding applications in terms of CPU would benefit from a native approach. However, to cover a maximum amount of potential users, it is obvious that in the case of native applications, more than one app will have to be built.

5.2 iOS versus Android

Chapter 6

Conclusion

Bibliography

[1] A. Charland and B. Leroux. Mobile application development: web vs. native. *Commun. ACM*, 54(5):49–53, May 2011.

[2] Coulouris, J. Dollimore, T. Kindberg, and G. Blair. *Distributed Systems: Concepts and Design (Fifth Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 5 edition, 2012.

[3] D. Graziano. Mobile market share 2012: Android continues its success, ios follows. URL: `http://bgr.com/2012/12/04/mobile-market-share-2012-android/`, 2012. Online; Accessed January 22, 2013.

[4] MobPartner. Top 5 html5 frameworks' for mobile web apps. URL: `http://blog.mobpartner.com/2013/06/24/top5-html5-frameworks-web-apps/`, 2013. Online; Accessed September 02, 2013.

List of Figures

List of Tables

Appendix A

REST API

Table A.1: REST API

Resource	**Method**	**Description**
`ANSWER`		
	create	Create a new answer.
	get_question	Get the question this answer was posted on.
`AUTHENTICATION`		
	endsession	Create a new answer.
	startsession	Get the question this answer was posted on.
`QUESTION`		
	create	Post a new question.
	get_answers	Get the answers for a question.
	get_info	Get information on a question.
	get_skills	Get the skills related to this question.
	remove	Delete a question.
`SKILL`		
	create	Create a new skill and add it to the database.
	get_questions	Get questions related to a given skill.
`USER`		
	get_answers	Get the answers submitted by a user.
	get_questions	Get questions submitted by a user.
	get_info	Retrieve profile information about a user.
	get_skills	Get the skills of a user.
	register	Register a new user account.
	search	Search a user given certain parameters.
	unregister	Delete a user account.
	update_info	Update profile information about a user.

Appendix B

Statistics