

# Multimedia Report Forum Application

*Multimedia modelleren en programmeren*

JORIS SCHELFAUT

Katholieke Universiteit Leuven

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	About the course . . . . .	3
1.2	About the application . . . . .	3
1.3	Structure of this text . . . . .	3
<b>2</b>	<b>Requirement analysis</b>	<b>4</b>
2.1	User story and storyboard . . . . .	4
2.2	Use case diagram . . . . .	4
2.3	Screen transition diagram . . . . .	4
2.4	Summary . . . . .	4
<b>3</b>	<b>Architecture</b>	<b>8</b>
3.1	Architectural challenges . . . . .	8
3.2	Architecture design . . . . .	8
<b>4</b>	<b>Implementation</b>	<b>10</b>
4.1	Data model . . . . .	10
4.2	REST API and backend . . . . .	11
4.3	HTML5 mobile web application . . . . .	11
4.4	Android application . . . . .	11
4.5	iOS application . . . . .	11
<b>5</b>	<b>Comparison</b>	<b>13</b>
5.1	Web versus native mobile applications . . . . .	13
5.1.1	Phonegap hack . . . . .	13
5.1.2	Trade-offs . . . . .	13
<b>6</b>	<b>Conclusion</b>	<b>14</b>
	<b>References</b>	<b>14</b>
	<b>List of Figures</b>	<b>16</b>
	<b>List of Tables</b>	<b>17</b>
<b>A</b>	<b>REST API</b>	<b>18</b>
<b>B</b>	<b>Statistics</b>	<b>20</b>

# Chapter 1

## Introduction

### 1.1 About the course

The course *Multimedia: modelleren & programmeren* is given by prof. E. Duval at KULeuven. The objective of this course is 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<sup>1</sup>. The progress of the development is communicated through a blog<sup>2</sup>.

### 1.2 About the application

The application is an online forum where the main question of each thread 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*<sup>3</sup> and *Yahoo! Answers*<sup>4</sup>, 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*<sup>5</sup>.

### 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 chapters 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.

---

<sup>1</sup><https://github.com/mumedev>

<sup>2</sup><http://mumedev.wordpress.com/>

<sup>3</sup><http://stackoverflow.com/>

<sup>4</sup><http://answers.yahoo.com/>

<sup>5</sup><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.

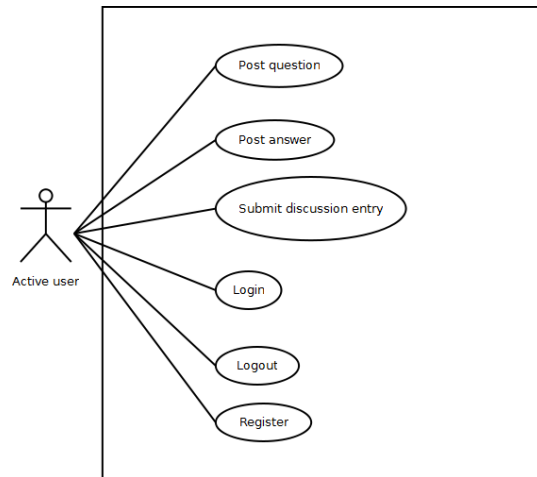


Figure 2.2: The use case diagram showing the functionality in the system from a user's perspective.

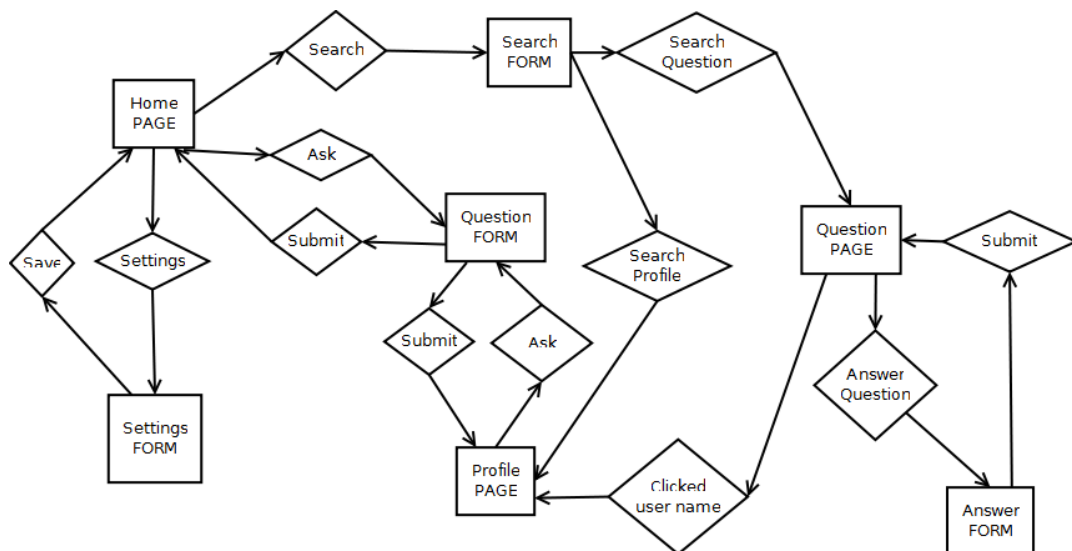


Figure 2.3: The transitions between the screens.

Table 2.1: Use case 1 *Post question*

<b>Primary actor:</b>	Active user
<b>Preconditions:</b>	User is logged in;
<b>Basic flow:</b>	(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*

<b>Primary actor:</b>	Active user
<b>Preconditions:</b>	User is logged in; The user has selected a question to answer. User is allowed to participate in the discussion.
<b>Basic flow:</b>	(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.*

<b>Primary actor:</b>	Active user
<b>Preconditions:</b>	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.
<b>Basic flow:</b>	(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 where 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 temporarily 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.



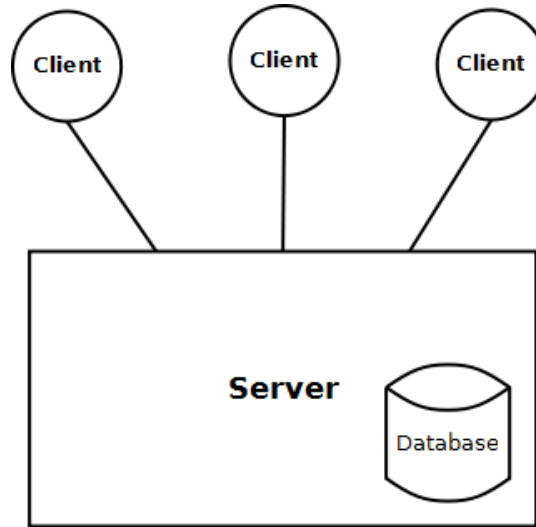


Figure 3.1: The global architecture of the application.

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).

```

<?php
// http://example.com?method=insert&table=user&username=joris

if ($_GET['method'] == 'insert') {
    $database->open();
    $id = $database->insert($_GET['table'], array(
        'username' => $_GET['username']
    ));
    $database->close();
    echo $id;
} 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 implement an SQL query ...
  </question>
  <answers>
    <answer>
      <author>sander</author>
      <date>09-02-2013 19:28:00</date>
      <text>
        Use this as the ...
      </text>
    </answer>
    <answer>
      <author>nik</author>
      <date>09-02-2013 19:30:00</date>
      <text>
        You might need ...
      </text>
    </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, *PHPMysqlAdmin*<sup>1</sup> was used to create the database.

<sup>1</sup>[http://www.phpmyadmin.net/home\\_page/index.php](http://www.phpmyadmin.net/home_page/index.php)

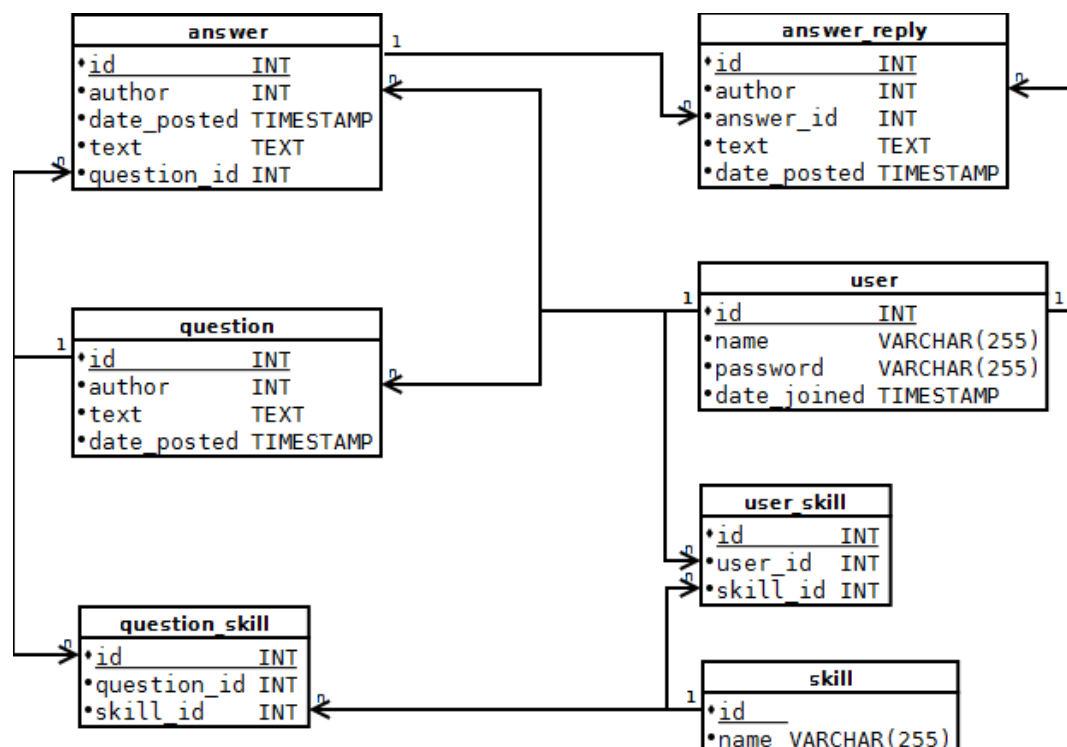


Figure 4.1: Entity relationship diagram of the database.

## 4.2 REST API and backend

The REST API is implemented using the codeigniter framework. Codeigniter<sup>2</sup> 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`<sup>3</sup> 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<sup>4</sup> is a library for Google App Engine applications. Of course, this would also require a somewhat different underlying data model as mentioned earlier.

## 4.3 HTML5 mobile web application

## 4.4 Android application

## 4.5 iOS application

<sup>2</sup><http://ellislab.com/codeigniter>

<sup>3</sup>The source code can be found at <https://github.com/philsturgeon/codeigniter-restserver>

<sup>4</sup><https://code.google.com/p/appengine-rest-server/>

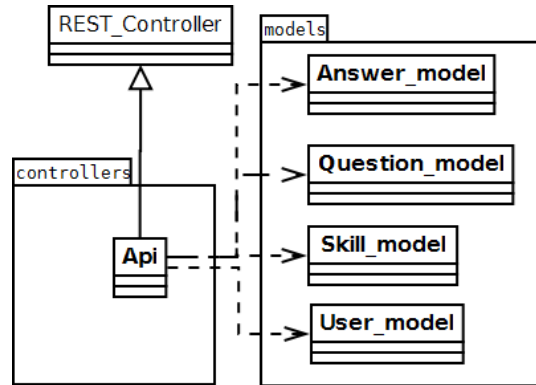


Figure 4.2: The class diagram of the codeigniter project.

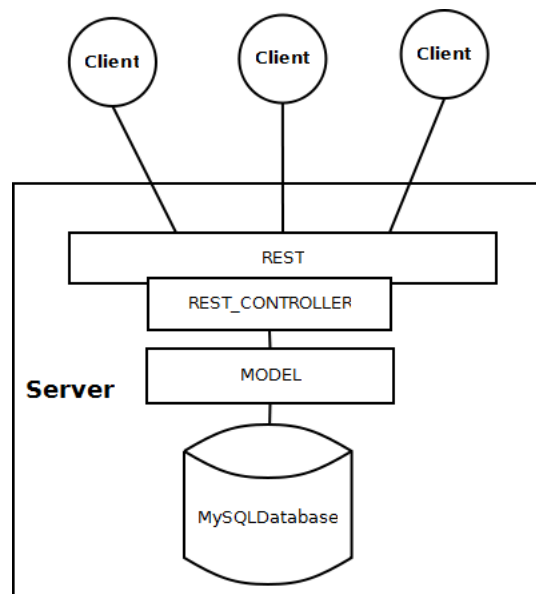


Figure 4.3: The global architecture of the application, including REST, the Codeigniter MVC design and the MySQL database.

# Chapter 5

## Comparison

### 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.

#### 5.1.1 Phonegap hack

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. 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.).

#### 5.1.2 Trade-offs

A brief list of trade-offs between web and native mobile applications: Size of the code base - the code base for a mobile web application will be significantly smaller than the code base for two or more native applications; Efficiency of maintenance - the cost of maintenance may be directly related to the number of versions of an application that need to be updated; Application performance - native applications still outperform mobile web applications; Development time; Application objectives - related to application performance, as nice looking visuals do not fulfill business requirements, but may require significantly more processing power; GUI guidelines: 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 separate platform;

## 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.

# List of Figures

2.1	Story board depicting a potential scenario of use. . . . .	4
2.2	The use case diagram showing the functionality in the system from a user's perspective. . . . .	5
2.3	The transitions between the screens. . . . .	5
3.1	The global architecture of the application. . . . .	9
4.1	Entity relationship diagram of the database. . . . .	11
4.2	The class diagram of the codeigniter project. . . . .	12
4.3	The global architecture of the application, including REST, the Codeigniter MVC design and the MySQL database. . . . .	12



# List of Tables

2.1	Use case 1 <i>Post question</i> . . . . .	6
2.2	Use case 2 <i>Post answer</i> . . . . .	6
2.3	Use case 3 <i>Take part in discussion.</i> . . . .	7
A.1	REST API . . . . .	19

# Appendix A

## REST API

Table A.1: REST API

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

# Appendix B

## Statistics