UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Faculty of Electrical Engineering,
Computer Science and Mathematics
Department of Computer Science
Database and Information Systems

Master Thesis

by

Jörg Amelunxen
Paulinenstrasse 9
33098 Paderborn
Student Registration Number: 650 44 65

Paderborn, December 28, 2014

DEVELOPMENT OF THE HIP-APPLICATION– NO SUBTITLE

Supervisor_____

Dr. Simon Oberthür

Examiners_____

Dr. Simon Oberthür
Prof. Dr. Gregor Engels

ABSTRACT

Lorem

ZUSAMMENFASSUNG

Ipsum

ACKNOWLEDGMENT

At this point I would like to thank all people who supported me and made this thesis possible in the first place.

Furthermore, I would like to thank all people who have participated at the survey, which was needed to evaluate the meaningfulness of the diagrams, and all people who proofread my thesis.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ABBREVIATIONS

AJAX   Asynchronous JavaScript and XML

API   Application Programming Interface

AR   Augmented Reality

CMMI   Capability Maturity Model Integration

CMS   Content Management System

CSS   Cascading Style Sheets

DOM   Document Object Model

HiP   History in Paderborn

HTML   Hypertext Markup Language

HTML5   Hypertext Markup Language V5

HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
JS	Javascript
JSON	JavaScript Object Notation
MVC	Model-View-Controller
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
SPA	Single Page Application
TDD	Test-Driven Development
UML	Unified Modelling Language
URI	Uniform Resource Identifier
WebGL	Web Graphics Library
WSDL	Web Services Description Language

INTRODUCTION IN THE THESIS

This first chapter will introduce the benefits of the system that will be developed within this thesis and will explain the current situation without the system.

1.1 WHAT IS THE CURRENT SITUATION WITHOUT THE TOOL/APP

At the present point in time, guests of the city Paderborn has to look up information about the city in a tedious way, for example using Wikipedia or other existing platforms. On the other hand, people, who want to provide information about the city (e.g., university employees or students), have to provide these information in a general accessible way, again like Wikipedia. Thus, they are limited to the features that are provided by these platforms. Since Wikipedia has been founded in 2004 by the Wikimedia foundation (Wikimedia-Foundation (2014)), most of the used technologies of the web application are nowadays outdated and very general. So, there is a rising need for a new technological updated approach, which is more focused on the specific topic of the city Paderborn and its history.

Especially the use of mobile devices has been risen in this time, which is easily recognizable at the number of sales of the Apple iPhone. The iPhone has been sold 0.27 million times in Q3 2007 and 51.03 million times in Q1 2014 (Statista (2014). Of course, this shift from the device side (i.e., hardware) includes a major shift in (software-) technology as well. Technologies like the nowadays well known AR would not be possible in 2004. Of course, this new technology includes a lot of new opportunities to transfer knowledge between people and cultures. AR is a great example to show how 'the real world' is blended more and more with artificial information; for example in the form of call-outs and

layers. AR is a technology that displays virtual (i.e., digital) information on top of a real object or location using the camera of a mobile device as input for the real objects. So, it ends up to be a combination of both worlds; the real and the digital one. Azuma et. al. has shown a lot of possible fields where the usage of AR would be a great improvement, which includes the field of annotation and visualization (Azuma (1997)). Furthermore, path finding and navigation are fields that could be revolutionized by using AR on mobile devices.

With a simple information website or app like Wikipedia, we include the tedious situation that the person that wants to get to the place he just read about, needs to input the address into another app to navigate him to the position. After he have arrived, he need to switch back to, for example, Wikipedia to manual compare the written information with the object or place he sees in front of him. If the person wants to change the shown information on his mobile device, he does this in general by using the touchscreen of the device. Nevertheless, he is comparing and looking at something that is placed in front of him. This leads to a break within the action and perception space (Hampel (2001)) and is a bad example with respect to the locality of the information (Bondo (2010)). As we will see, AR is a tool that we can use to remove this problem and join the action and perception space while keeping the locality of the information in mind. Furthermore, at the moment we have a lot of unnecessary overhead due to the needed app switching between the information app and the navigation app.

Now, even if somebody wants to publish information about Paderborn on Wikipedia to enable guests of the city to get knowledge about the environment, it is only possible to publish the information as static content (that includes text, graphics, audio and video). On top of that, it is not possible to review the information privately and in enough detail to create university courses that do not include a written paper as the final exam but an entry within such an information system. So, if we would have private annotations within a system that is owned by the university, it would enable the university employees to offer courses that fill the database about Paderborn with high quality content by students.

This leads us to the application that should be prototypical developed within this master thesis, which will be described in the next section.

1.2 WHAT WOULD THE SYSTEM LOOK LIKE (BRIEFLY)

As we will see in chapter **??**, the system will be divided in two big parts. One part is the web-backend, which is connected to an *MongoDB* to store and retrieve the needed information. This backend will provide a Representational State Transfer (REST) interface, which enables it to be connected two different kind of frontends.

These frontends create the second big part of the system. The first prototype of the system will include a web-frontend to access the backend for administrative purposes (e.g., including new data by students, creating groups, review data, etc.) and will be driven by the play framework in combination with AngularJS.

The second kind of frontend, which will be needed in the first prototype of the system, will be the smartphone frontend. With this frontend, the end-users (i.e., everybody who downloads the app from the app-store) are able access the information, which are included in the backend. Furthermore, the smartphone frontend will make use of AR features to show the information that is stored in the backend.

After we have now seen, how the system will look like, we will now take a short look at the question, who will actually benefit of such a system.

1.3 WHAT WOULD BE BETTER IF THE APP WOULD EXIT? WHO WOULD BENEFIT?

On the one hand, users would benefit from the app by having a neat tool to discover the history of the city paderborn. It will be a great experience to be guided trough the city and learn a lot of important and interesting facts about the environment. On the other hand, the system will be a nice variation for the students, which may be bored from the typical *send in a homework to pass the exam* cycle and can include the information directly into the backend and are able to see *their* work some time later via the app in the frontend. So, they are actually able to **do** something, which is used in the future.

1.4 OUTLINE

The second chapter will explain all needed fundamentals of this Master thesis in detail and will show the used frameworks and tools.

The third chapter will outline the application design and describe some general design decisions.

The fourth chapter will show important parts of the actual implementation, used tools and the final Unified Modelling Language (UML) diagrams of the application.

The fifth chapter will show unit tests and the results of a survey that was used to evaluate usability of the system.

The sixth and last chapter will deal with arisen problems and will discuss the development for future work.

TECHNICAL AND METHODOLOGICAL BACKGROUND

After we have seen the need for the planned application, we will now get an insight into the used technical frameworks respectively tools and methodological concepts, which are used during the development process.

2.1 AGILE DEVELOPMENT – SCRUM

The main idea of agile development is described within the agile manifesto[1]. Within agile development, the focus is set to frequent software delivery and close customer relationship. Because Scrum is such an agile development method, we find similar ideas in the Scrum development process.

A big problem within the software industry is that projects take longer than expected and the expectation of the customer differs more and more from the view of the development team because doing a 6 month part of requirements engineering, followed by an 18-month development period lets one finish a product that is already obsolete at the day it gets published.

Similarly, the HiP-application will be developed in a Scrum-like fashion to achieve a high efficiency in the small timeframe. However, a full Scrum approach is not possible because the development team is very small. Nonetheless, we will include the ideas and concepts of the Scrum process but will, for example, combine different roles in one person. But to get a first intuition about the Scrum process itself, we will now describe the main ideas of Scrum and agile development in general.

Using Scrum means, the application will be developed within autonomous short *sprints* with a length between 1 up to 4 weeks. However Berczuk (2007) points out that a four week sprint is in many cases problematic because a lot of customer requests have to be included into currently running sprints and so

---

1 The whole manifesto can be found here: http://agilemanifesto.org

the backlog becomes more and more useless; he suggests having about 2 weeks long sprints. In any case, after every sprint the product should be more refined (Schwaber and Beedle (2002)). However, it should be possible to execute the application at the end of any given sprint, which will result in a fast and stable development process. The general development process is also shown in Figure 1. The Product-Backlog is the foundation of every sprint-backlog because it contains every feature that will be needed in the product at some time. So, one derives the sprint-backlog, which includes every feature that should be added in a specific sprint, from the product-backlog for every new sprint. In addition to that, daily Scrum meetings should ensure that the whole team is up-to-date and as efficient as possible. In more detail, Scrum is known to reduce every category of work (i.e., defects, rework, total work required, and process overhead) within a Capability Maturity Model Integration (CMMI) compliant development process by almost 50% (Sutherland (2009)), which is also great for development in this short timeframe.

The development process also influences the order of the development because the chose of Scrum indicates that we will develop the front-end and back-end in parallel. We will use a Test-Driven Development (TDD)-like approach within every sprint (where possible). So, we will at first create needed test cases and afterwards implement against these test cases. This approach will prevent that testing of the application will be shifted into the last week(s) of the development process and done in a superficial way. In addition to that, a comprehensive test suite is a great basis for further development (Maximilien (2003)).

Now, after we have seen the general concept of Scrum, we will now take a look at methods for cost estimation.

2.2 METHODS FOR COST ESTIMATION

To get a feasible estimation of the workload of a given backlog, as it has been described in section 2.1, we need some methods to create a good work- respectively cost-estimation. This is important to be able to choose a fitting amount of work per sprint.

... add ....

A burn-down chart shows the amount of remaining story points (which correspond to a specific estimated time-frame, like one story point per 30 minutes) on the y-axis and the days of the sprint on the x-axis. With such a chart, one
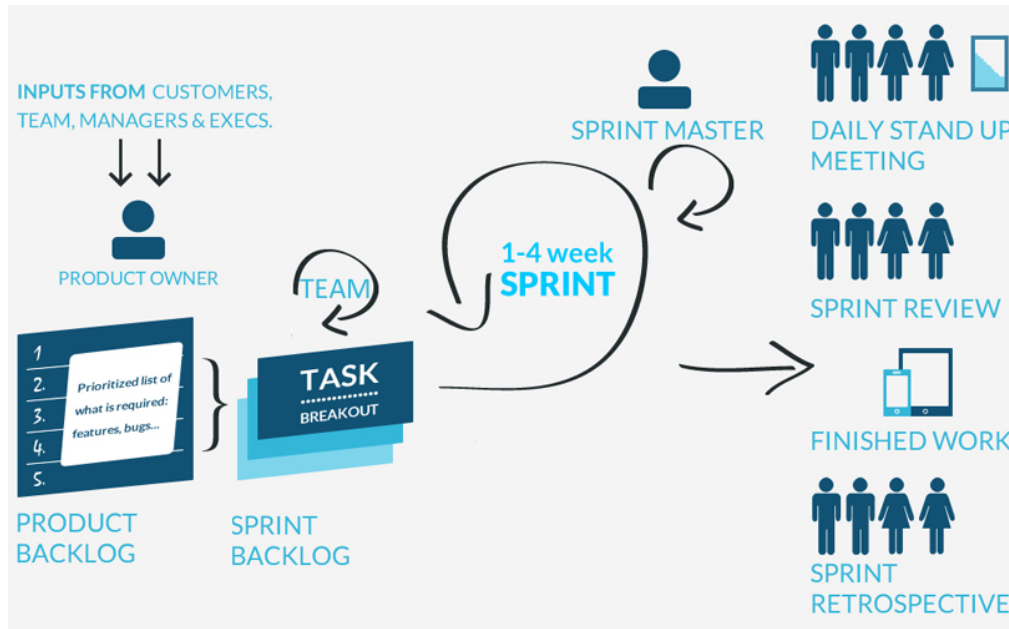
Figure 1: The diagram shows the Scrum development process. (Simplified, original work from **?**)

can estimate the remaining time, and can derive if the sprint can be finished in the given time-frame, by looking at the *slope* of the graph. Figure 2 shows an example burn down chart over 8 days. The OPT line shows the optimal slope that ends up on 0 remaining story points on the last day. If the sprint curve is above the OPT line, you are to slow in the current sprint and in the case the sprint curve is below the OPT line, you are ahead of the time.

2.3 USED FRAMEWORKS

Because the time frame for the project is quite small, it is not possible to create the whole application from scratch and, thus, we need a couple of frameworks to accelerate the process. We will use the Play-Framework in the backend, which offers a REST-interface and handles the routing from Hypertext Transfer Protocol (HTTP)-requests to application code. On the frontend-side, AngularJS will be used to create a fast and responsive web-interface and Junaio-will be used to include the AR-functionality on the smartphone application.
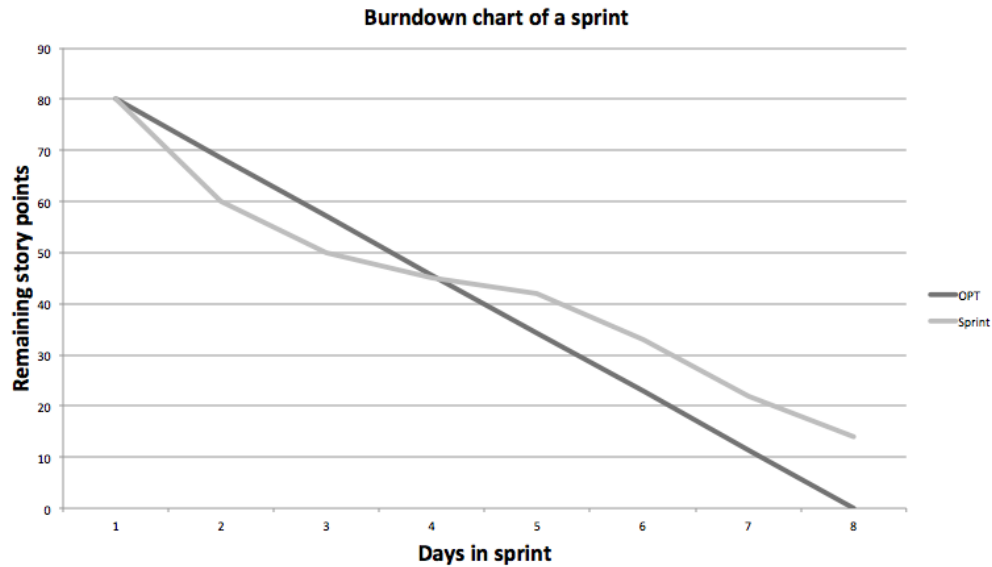
Figure 2: The burndown diagram shows an example sprint.

2.3.1 Play Framework – Backend

We will use the Play framework for the backend of the application because Play is an open source web application framework, which is written in Scala and Java, follows the Model-View-Controller (MVC) architectural pattern and handles the routing from HTTP-requests to application code.

A simple example for the routing configuration file is shown in Listing 2.1. In this file, each documented route consists of an HTTP method and Uniform Resource Identifier (URI) pattern that is linked to a call of a, so called, *action method* within the Java respectively Scala code. As one can easily see in line 9, it is quite easy to pass parameters to the application code. Furthermore, one can see that the parameters are type-safe; thus, for example, a String passed in as an Integer would result in a compilation error.

Listing 2.1: Simple routing configuration file within the Play Framework

```
1  # Routes
2  # This file defines all application routes (Higher priority
       routes first)
3  # ~~~~
4
5  # Home page
6  GET /                   controllers.Application.index()
7
```

```
8   # Usage of parameter
9   GET /thesis/:grade   controllers.Application.exp(grade: Integer)
10
11  # Map static resources from /public to the /assets URL path
12  GET /assets/*file    controllers.Assets.at(path="/public", file)
```

Very briefly, the Play framework includes three different parts:

1) Java Code that implements the controllers. The controllers are used to handle requests that get routed to them via HTTP. A simple controller is shown in Listing 2.2. As one can see within line 10 of Listing 2.2 the String "Your new application is ready" gets passed to the render function of the class index and returned as a parameter within the *ok* function, which creates a simple HTTP header with return-code 200. The index class is a Scala class that gets automatically creates from the Scala/Hypertext Markup Language (HTML) template called *index.scala.html*. We will see this in more detail within point 3 of this list.

Listing 2.2: Simple Java-controller within the Play Framework

```
1   package controllers;
2
3   import play.*;
4   import play.mvc.*;
5   import views.html.*;
6
7   public class Application extends Controller {
8
9       public static Result index() {
10          return ok(index.render("Your new application is ready."
                ));
11      }
12  }
```

2) Java Code that implements the model entities. The model is used to do the actual calculation and data handling. In essence, we should include as less application-logic as possible within the controllers and use these model classes instead.

3) Scala templates that are used as *views*. As a return value of the controllers, they pass data to a fitting template and return a corresponding HTML-view. However, we may also skip the template engine sometimes to directly return JavaScript Object Notation (JSON)-documents, which can be used to provide a Application Programming Interface (API). Listing 2.3 shows the used *index.scala.html* template used in point 1 of this list. In line 1, we declare the used parameters, in our case one String variable, which we have used to pass

the String "Your new application is ready". The @*main* command in line 3 calls another template, which includes everything beside the HTML body. The body of the file is now included in line 4 by calling another framework specific method, which includes a welcome and documentation message and renders our passed String variable.

Listing 2.3: Simple Scala template within the Play Framework

```
1  @( message : String )
2
3  @main( "Welcome to Play" ) {
4      @play20 . welcome( message , style = "Java" )
5  }
```

As another important fact, Play emphasizes the usage of the REST principle, as it can be seen within the routing configuration file. We can easily and directly make use of the different HTTP commands and use them to structure our API accordingly. In general, Representational State Transfer (REST) is a style of software architecture that is used to build distributed systems and has been introduced in the dissertation of Fielding (2000). As Rodriguez et. al. point out, REST based web services are easier to use than Simple Object Access Protocol (SOAP) and Web Services Description Language (WSDL)-based ones and getting more and more importance since mainstream web 2.0 service providers are taking up on REST(Rodriguez (2008)). Furthermore, the Play-framework comes with integrated unit testing and full support of asynchronous I/O. So, all in all, Play will noticeably enhance the development speed of the backend.

2.3.2 MongoDB

As Trelle (2014) describes, data entries within a MongoDB are called documents and are in essence ordered sets of key-value pairs. However, values can also be complete documents and arrays, so one can store complex hierarchical structures within a MongoDB. Similarly, the data gets stored in a **BSON!** (**BSON!**)-format, which is a binary JSON format with more datatypes and better traversability.

These documents are stored in so called collections, which are in general comparable to the tables in a relational database, like **MySQL! (MySQL!)**.

For every document within the database we need to include a field called *_id*, which contains the primary key of the document. Of course, this primary key has to be unique within the collection that contains the document. If an

document is stored within the database without a *_id* field the field gets automatically generated.

An insert into a MongoDB is easily done and shown in Listing 2.5.

Listing 2.4: Inserting into a MongoDB

```
1  var db = ... // contains the connection to the database
2
3  var object = {
4    firstname : 'John',
5    lastname : 'Doe'
6  };
7
8  db.hipUsers.insert(object);
```

Similarly one can read documents from the database by creating a document that is matched against the collection. For example, to retrieve the user *John Doe* that has been included within Listing 2.5 by matching against his lastname, one would need to do the steps shown in Listing **??**.

Listing 2.5: Reading documents from a MongoDB

```
1  var db = ... // contains the connection to the database
2
3  var object = {
4    lastname : 'Doe'
5  };
6
7  db.hipUsers.find(object);
```

Now, we will take a look at the frontend technologies.

2.3.3   AngularJS – Frontend

After we have now seen the basics of the Play framework and the MongoDB, which will handle the backend functionality, we will now take a look at *AngularJS*, which will provide needed features to create a fast and responsive frontend. The frontend will be designed as a Single Page Application (SPA). A SPA is in general an orthogonal approach to the common way of creating websites as a set of linked pages. A SPA is a composition of individual components which can be updated respectively replaced independently of the complete site and, thus, without any reload after the actions of the user. This results in a cou-

ple of benefits, like improved interactivity, responsiveness and user satisfaction (Mesbah (2007)).

Listing 2.6: Simple example that shows the use of an AJAX request that shows the reponse text within a specific div container

```
1  xmlhttp.onreadystatechange=function(){
2    if (xmlhttp.readyState==4 && xmlhttp.status==200){
3      document.getElementById("myDiv").innerHTML=xmlhttp.
           responseText;
4      }
5    }
6  xmlhttp.open("GET","ajax_info.txt",true);
7  xmlhttp.send();
```

Obviously, creating a SPA implicitly forces the usage of some kind of request mechanism to get the data that the user needs, without reloading the site. This could for example be done with an Asynchronous JavaScript and XML (AJAX) request like the one that is show in Listing 2.6. The listing shows how the request is being made in line 6 and 7. Line 3 shows the exchange of the content of the div container with the id *myDiv*. However, using AJAX is cumbersome and can nowadays easily be hidden in very sophisticated frameworks, like AngularJS.

The *AngularJS* framework will be explained briefly in the following. AngularJS makes heavy use of expressions and directives. While directives in AngularJS are functions that get run when the DOM is compiled by the compiler and are shown as simple tags or attributes, an expression is a term that is encapsulated by *{{ ... }}* and gets evaluated while the page gets loaded.

Listing 2.7: Simple example that shows the use of expressions

```
1  <div class="panel-heading">
2    {{lc.getTerm('system_group_navigation')}}
3  </div>
```

Listing 2.7 shows a simple example, where an expression is used to call a method of a controller object. As soon as the page gets rendered, the Document Object Model (DOM)-tree will be loaded with the result value of the given javascript function called *getTerm(String)*. Another major feature of AngularJS is the so called two-way data binding, which is closely coupled to expressions. The two-way data binding ensures that the rendered value of the function getTerm(String) gets automatically updated, as soon as the function returns a different value.This creates a source-code that includes less unnecessary lines of code for updating the values in the view.

Furthermore, AngularJS offers directives like *ng-class* which adds dynamically a specific class to a DOM-element if a given expression evaluates to true.

Listing 2.8: Simple example that shows the ng-class directive to change the style respectivly color of an alert depending on its type

```
1  <div ng−class="{'alert-warning' : alert.type == 'warning',
2                            'alert-danger' : alert.type == 'danger
                                ',
3                            'alert-info' : alert.type == 'info',
4                            'alert-success' : alert.type == '
                                success'}"
5       role="alert">
6
7    {{alert.msg}}
8  </div>
```

Listing 2.8 shows how the *ng-class* directive exchanges the used style of the alert depending on the boolean expression that is places behind the ':'. So the syntax of the attribute value is { class : expression }.

Of course one can also create own directives to get a much cleaner code. For example, it is possible to create a directive called *chat-box* which can directly included within the DOM-tree. Thus, the usage of the created chat element folds down to the code that is shown in Listing 2.9. The same can be achieved by using web components or Google polymer, which is in essence an extension of the web-components technology. However both technologies are, at the present point in time, only fully compatible to Google Chrome. Because of that we will use custom AngularJS directives to create clean and maintainable code.

Listing 2.9: Simple example that shows the usage of a custom directive

```
1  <!-- some code up here -->
2
3  <chat−box> </chat−box>
4
5  <!-- some code down here -->
```

Besides AngularJS, Twitter Bootstrap will also play an major role in the frontend development process.

2.3.4 Twitter Bootstrap

Twitter Bootstrap[2] is an open and freely available collection of tools on the basis of the HTML, Cascading Style Sheets (CSS) and Javascript (JS) and can be used to support and accelerate the building of web applications. We will use Twitter Bootstrap inside the user front-end of our web application because it works nicely together with AngularJS and can for example be used to create tabs and alerts. Furthermore, Twitter Bootstrap is nowadays used a lot by common web-applications and, thus, we enhance the external consistency of the History in Paderborn (HiP)-application in respect of other web-applications, which may be well known to the user.

ref external consistency

Twitter Bootstrap is licensed under the terms of the Apache License v2.0[3].

Furthermore, Bootstrap can be used with Bootstrap UI, which are Bootstrap components that have been written in AngularJS and can easily be reused. Examples for these components are tooltips, datepickers, timepickers, etc. So, this is also a great repository for components to accelerate the development process. [4]

2.3.5 Junaio – Frontend

The AR-functionality will be offered by the framework Junaio. The company Metaio, which runs Junaio, offers a developer program to develop own applications on the basis of the Junaio (eco-)system. Moreover, it is completely free of charge for the developers (Junaio (2014)). However, deployed apps will be shipped with a Metaio watermark inside as long as you do not buy a specific license.

---

2 Twitter Bootstrap is hosted on GitHub and can be downloaded here: `https://github.com/twitter/bootstrap`

3 The terms of the license can be found here: `http://www.apache.org/licenses/LICENSE-2.0`

4 Bootstrap UI can be downloaded here: http://angular-ui.github.io/bootstrap/ and is distributed under the MIT license https://github.com/angular-ui/bootstrap/blob/master/LICENSE

2.4 WEBGL

Furthermore, we will need Web Graphics Library (WebGL) to create a possibility to render and manipulate the 3D-point clouds, of the scanned objects, right within the browser.

2.5 TOOLING

A couple of frameworks and techniques is a good start for creating such a sophisticated system, however, we will also need fitting tooling to support the development. These tools will be described in the following section.

2.5.1 Git

We will use Git, which is a commonly used distributed revision control and source code management system, for the versioning of our source-code. Git is free software distributed under the terms of the GNU General Public License version 2.

The service GitHub offers his users the possibility to maintain public and private Git repositories. The usage of GitHub is free, if the user uses public repositories only. We will use GitHub to host our source-code.

2.5.2 Jira

Jira is a proprietary software for project tracking purposes, which has been developed by the company Atlassian. It has support for agile development methods like Scrum or Kanban and offers a couple of features for bug tracking and time respectively cost estimation, like burn-down charts, which has been explained in section 2.2.

We will use Jira to track the status of the HiP application.

2.5.3   IntelliJ IDEA

IntelliJ IDEA is a Java Integrated Development Environment (IDE) by the company JetBrains.

The current version offers support for Java 8, UI designer for Android development, Play 2.0 and Scala and is, thus, a good choice to work with because all of these features will be used in our development process.

The IDE is available as an Apache 2 Licensed community edition and a commercial edition. The commercial edition can also be downloaded for free for educational purposes.

2.6   TESTING TECHNIQUES

2.6.1   Why is testing within an agile development even more important

2.6.2   TDD

2.6.3   The fundamental test process

DRAFT OF THE APPLICATION

To handle all these problems that have been described above within the section about the current situation, we will create a smartphone application that gets supported by a backend web-server. The master thesis will handle the planning respectively cost estimation of the different parts/features of the system and will, after the needed technologies/frameworks are elaborated and evaluated, include a prototypical implementation of the needed components. But at first, we will start with the first step in a development process; the requirements engineering.

3.1 REQUIREMENTS ENGINEERING

Because the HiP-Application will be developed closely together with our *costumer*, other working-groups at the university of paderborn, the whole process starts with the requirements engineering phase.

First of all, a requirement is defined as "[...]A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents[...]" (IEEE (1990)).

We started the development with a requirements engineering meeting together with our *customer* and ended up with a couple of cards with written user stories. Afterwards, these stories got refined to concrete requirements, which are measurable and prioritized. A complete list of all requirements, which were derived from these user stories, can be found within the appendix in Table 5 and **??**. These requirements can directly be used to derive test cases from them, which is good because we will use a TDD driven development approach in every sprint.

In the following, we will explain the two parts of the application separately.

3.2 BACKEND (WEB–SERVER)

Another important part of the system is contained within the backend-web-server. The backend should contain the whole data handling and assessment. The students should be able to add data to the system (e.g., a textual article, graphics, AR-data, etc.) and to modify existing data via a Content Management System (CMS). These entries get reviewed, for example by the course supervisor, and unlocked for the frontend application. To do this, the backend needs features like annotations and highlighting, which should be private for a specific user. By using this, the supervisor can evaluate the given texts right within the CMS and give his final judgement. If the supervisor is not satisfied with the quality of the given text, he should be able to send the document back to the student, to get a revised and updated version of the document. If the supervisor is satisfied, he can unlock the information for showing in the frontend application.

The data should be stored in a way that it can be shown within an AR-environment in the smartphone application. Of course, we will need some mechanism to structure the data, for example tags or stored categories. This kind of information (especially tags) are also very important for the described filtering techniques on the client side.

Furthermore, the backend should include a way to modify the point-clouds of the objects that has been scanned with the smartphone application. It will need features to add annotations directly to these point-clouds to show them afterwards within the AR-environment. This editor will be created on the basis of Hypertext Markup Language V5 (HTML5) and WebGL. A mockup of this site is shown in Figure 3. These annotations should also be assessable and (un-)lockable for the supervisor.
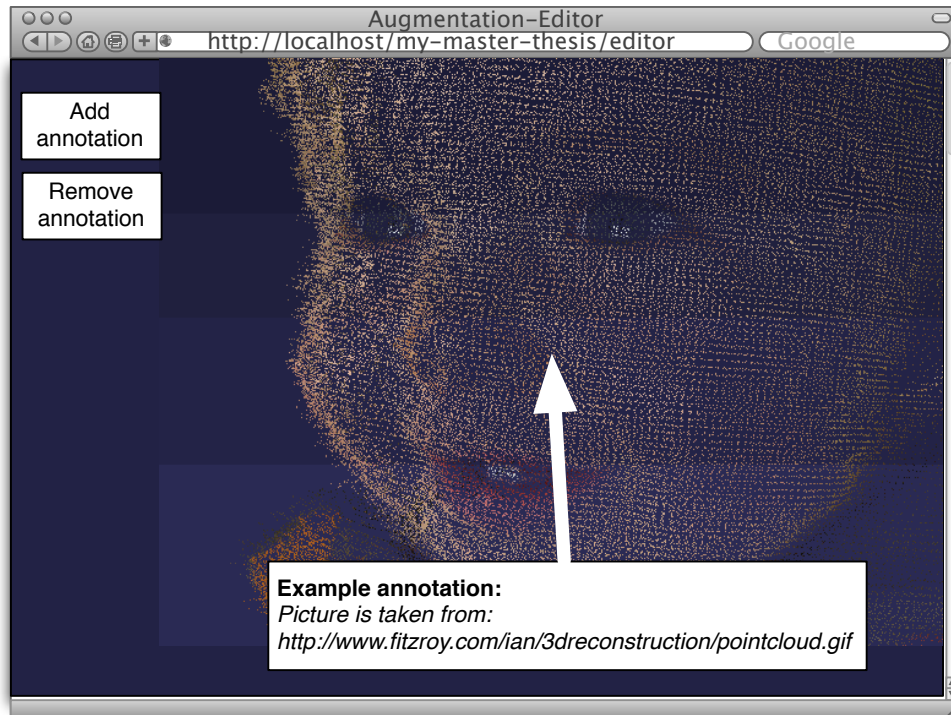
Figure 3: A mockup showing the augmentation editor that will be included in the web-application. The editor will be used to edit the point-clouds, which have been added with the help of the smartphone-application

3.2.1 Cost estimation of the backend

3.2.2 Input data/content via CMS in the system

3.2.3 Manage content as a reviewer

3.2.4 Including a 3D-Tooling system for point–clouds (WebGL)

3.3 FRONTEND (APP)

The smartphone application is the part of the system that gets shipped to the end-user (respectively downloaded via an App-Store like Google-Play). The user can use the app to find interesting places respectively objects in Paderborn and is able to start a navigation to the place/object easily. Furthermore, the user can get an overview about all places in Paderborn by activating a map
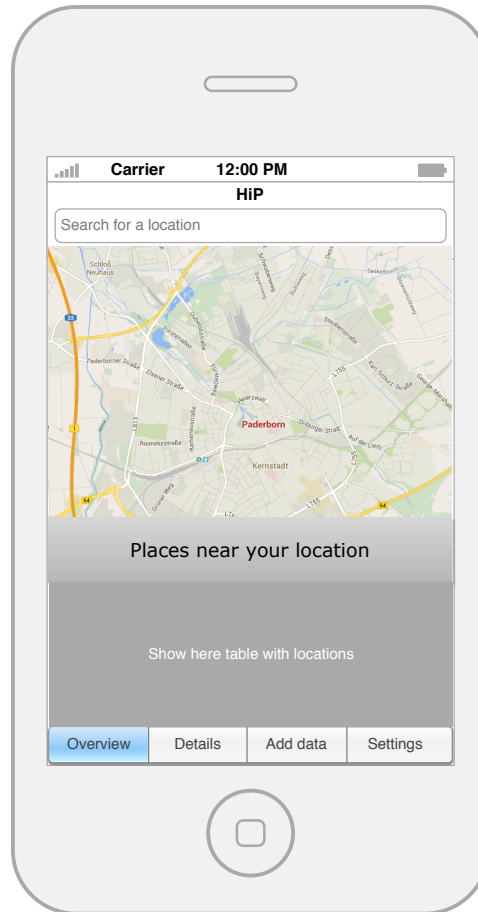
Figure 4: A mockup showing the main page of the frontend application showing a map of paderborn and a general overview about the UI-elements

that shows all entries within the system. A mockup of this view is shown in Figure 4. Of course, the user will be able to set up specific filters like 'show only art', 'show only historic buildings' or 'use simplified language' to adapt the system to his own experiences and educational qualifications. Moreover, if the university courses would add information over years, the system will need filtering features like this to handle the complexity of the data.

After an user has reached an interesting place, he can use the details tab to switch into the AR-mode. With this view, the user can use the smartphone-camera to embed information, which has been added via the backend, right into the picture of the object. An mockup of this view is shown in Figure 5. To create a feasible input for the AR system, the user should be able to scan objects in 3D right with his smartphone application and send the data (i.e., a

point-cloud of the scanned object), back to the web-server. Afterwards, the user can add annotations to the point-cloud via the web-backend of the system.
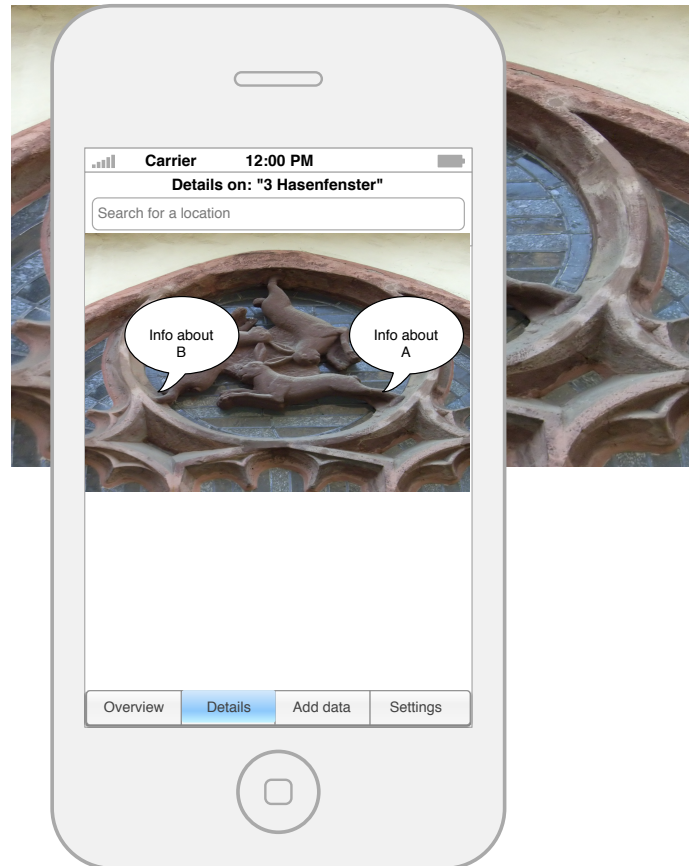

Figure 5: A mockup showing the details page of the "Dreihasenfenster" while the camera of the smartphone is pointing to the window itself

3.3.1   Cost estimation of the frontend

3.3.2   Input data into the system (scan objects and annotate them)

3.3.3   Show close "interesting places" within a map / via a overlay

3.3.4   Navigation to "interesting places"

3.4   INTERFACE

3.4.1   Data format for AR files

4

IMPLEMENTATION DETAILS

5

TESTING THE APPLICATION

5.1   TEST ENVIROMENT

5.2   TESTING RESULTS

5.3   ACCEPTANCE TEST OF THE PROTOTYPE

5.3.1   Small usablity study of the app

5.3.2   Small usability study of the backend / CMS

6

DISCUSSION AND FUTURE WORK

6.1 ARISEN PROBLEMS WITHIN THIS THESIS

6.2 DISCUSSION AND FUTURE WORK

6.2.1 Results / Conclusion

6.2.2 Future work

A

APPENDIX

The appendix contains some diagrams and tables that were to big to put them into the continuous text.

The IDs of the list, which are written in bold letters, will be done within the master-thesis itself. The remaining parts can be done via, for example, project-groups, etc.

Table 1: Showing the derived requirements of the Backend for the supervisor role, which are sorted by priority

ID	Description	Acceptance criteria
BS1	The supervisor should draft **guidelines** and assistance (e.g., Button with question-mark)	- At least one information resp. help function per functionality
BS2	The supervisor should be able to see **which data is missing**	- The feedback should be visual - The feedback should be transparent to upper layers of the UI
BS3	The supervisor should be able to see **data that is ready** for review	- Try without content that is ready for review - Try with content that is ready for review
BS4	The supervisor should be able to **assign exhibits** to students	- Try assigning an exhibit to one student - Try assigning an exhibit to more than one student
BS5	The supervisor should be able to **trace content** back to specific students	- Show visual connection between student and content
BS6	The supervisor should be able to **define topics** and exhibits	- Try defining a topic more than once
BS7	The supervisor should be able to **comment and discuss** the given content of the students	- Try commenting empty content - Try commenting a lot of content
BS8	The supervisor should be able to **mark errors** in the content	- Try marking an error twice
BS9	The supervisor should get **e-mail notifications** about new content handed in by students	- The message should leave the system in less than 2 minutes in 90% of the time
BS10	The supervisor should be able to **copy topics** and categories (e.g., usage of templates for different typical cases, duplication, etc.)	- Try copy an empty topic - The copied topic should be easily changeable to adapt it to the new usage
BS11	The supervisor should be able to define **validation-constraints** (e.g., character limitation)	- Try with error within the validation constraints
BS12	The supervisor is able to see the **amount** of **texts and pictures** in a hidden topic	- Try without any content included - Try with a lot of content included
BS13	The supervisor should be able to **work offline**	- Try disconnecting a running session

Table 2: Showing the derived requirements of the Backend for the student role, which are sorted by priority

ID	Description	Acceptance criteria	Pr
BSt1	The students are only able to **send in specific content** (field / topic)	- Try sending content to another topic	1
BSt2	The students should get an **e-mail noti-fiation** about new content in their topic (e.g., send in via fellow students)	- The e-mail should be received in less than 2 minutes in 90% of the time	1
BSt3	The students should be able to send in **metadata**	- Try with errors within the meta-data	1
BSt4	The students should be able to **overview** the **possible links** within their topic (e.g., GPS-information)	- Try without any links - Try with a lot of links	1
BSt5	The students should be able to **send in content**	- Try sending empty content - Try sending a lot of content	1
BSt6	The students should be able to **propose** topics and content	- Try proposing an existing topic	1
BSt7	The students should only have **access** to the backend **for a specific time**	- Try logging in after the temporary account has been deleted	1
BSt8	The students should have **access to all** temporary content (i.e., not reviewed content)	- Try accessing currently empty content	1
BSt9	The students should be able to create **interdisciplinary groups** and communicate within these	- Try creating a group without users - Try to send an empty message to the group - Try to send a very long message to the group	1
BSt10	The students should be able to see their content in a **preview mode** that simulates the frontend	- Try showing an empty topic - Try showing a huge topic	2
BSt11	The students should be able to see content of **other groups in a preview** mode that simulates the frontend	- Try showing an empty topic - Try showing a huge topic	2
BSt12	The students should be able to comment and **discuss** the content of their group or other groups	- Try to send an empty comment - Try to send a huge comment	2
BSt13	The students should be able to **hide** their unfinished work to the supervisor	- Try hiding without having any content	2

Table 3: Showing the derived requirements of the Backend for the master role, which are sorted by priority

ID	Description	Acceptance criteria	Prio
BM1	The master should be able to **recover data** by using a back-up system	- The recovery should not take longer than one hour	1
BM2	The master role can be assigned to a **couple of users** at the same time	- Try to assign the master role to nobody	2
BM3	The master is able to do the final **acceptance**	- Try to accept an empty topic - Try to accept a huge topic	2

Table 4: Showing the derived requirements of the Backend, which are sorted by priority

ID	Description	Acceptance criteria
BMi1	The data of the system is **stored** on IMT-Server	- The data should be easily transferable
BMi2	The system can be **updated** and maintained in the future (e.g., project-groups, SHK, etc.)	
BMi3	The content should not be limited to specific **layouts, views** (e.g., languages) and templates	
BMi4	The system should be **expandable** (e.g., new content, filters, etc.)	
BMi5	The system should be **safe** with respect to hackers resp. data manipulation	-The system should be safe with respect to the economic view/ definition of safety
BMi6	The system offers features to **manage** groups	- Try managing a group with an empty name

Table 5: Showing the derived requirements of the Frontend, which are sorted by priority

ID	Description	Acceptance criteria
F1	The user should be able to **navigate** to the different locations shown in the HiP-application	- The navigation should response fast - Try navigating to the current position
F1.A	The user should be able to navigate to the different locations and discover these locations on his own	See F1
F1.B	The user should be able to navigate to the different locations and use round tour information of the application	See F1
F1.B	The user should be able to navigate to the different locations while using filters (e.g., epochs)	See F1
F2	The user should be able to create **thematic routes**	- Try creating a route without assigning a theme
F3	The user should get a **list of locations/exhibits** in Paderborn	- Try opening an empty list
F4	The user should **see linkings** within an exhibit different exhibits (e.g., Liborischrein -> Hle -> Scriptorium)	- Try opening a topic without links - Try opening a topic with a lot of links
F5	The user should be able to **deselect** specific categories	- Try deselect only one - Try deselect many
F6	The user should be able to **filter** exhibits on the map (e.g., locations, historical figures, etc.)	- Try using multiple filters
F7	The user is able to **overlay** the current map of the city with historical maps	- Try overlay one map with a hist. one - Try overlay a couple of maps
F8	The user is able to **see himself and historical** places on the map	- Try in an area without hist. places - Try in an area with a lot of hist. places
F9	The user should not **exceed his storage** on the smartphone	- Clear cache should be possible
F10	The user should not **exceed his data-volume** on the smartphone	- Pictures and videos have to be small
F11	The user should be able to use the application easily (**good usability**)	- Interface should not include too many functions per view

Table 6: Showing the derived requirements of the Frontend, which are sorted by priority

ID	Description	Acceptance criteria
F12	The user should be able to switch between **different contents** (e.g., Video, 3D, etc.) fast	- At most two clicks/touches between the different contents
F13	The user should be able to see *invisible* objects within the details-tab (e.g., something placed inside an altar)	- Try with more than one invisible object at the same time
F14	The user should be able to use **tablets** and smartphones	- The UI should adapt to the screen size resp. resolution
F15	The user should only get details about an exhibit while he is **next to it or afterwards**	- Try to get details beforehand
F16	The user should be able to get **texts, graphics/ pictures** and links about an exhibit	- Try without any texts, etc. - Try with a lot of texts, etc.
F17	The user should be able to get **audio, video** and **3D-views/models** about an exhibit	- Try without any videos, etc. - Try with a lot of videos, etc.
F18	The user can create and join **treasure hunts** respectively geo-caching features	- Try join an treasure hunt without a name
F19	The user should get informed about **exhibits** and **locations that are next to him**	- The information should be send immediate as the user arrives at the position
F20	The user should be able to get navigated with **AR-rabbits**	See F1
F21	The user should be able to get navigated inside of a **building**	- The navigation should be accurate
F22	The user should be able to choose between different **starting possibilities** (i.e., tour, discovery and historical topics)	
F23	The user should be able to **hear the content** via an audio-guide	- The audio files should be small (see, F9, F10)
F24	The user should be able to get exhibits as **comparison by using AR**	- Try opening more than one exhibit as comparison
F25	The user should be able to **create own notes and comments**	- Try creating an empty note/comment - Try creating a huge note/comment
F26	The user should be able to **share content** via social media	- Sharing should not need more than two clicks
F27	The user should be able to **export content** as PDF and create book-marks	- The export should not take longer than 30 sec in 90% of the time
F28	The user should be able to get the content in **different languages** (i.e., english, french, turkish)	- Adding new languages should be easy

BIBLIOGRAPHY

Azuma, R. T. (1997). A survey of augmented reality a survey of augmented reality a survey of augmented reality. *In Presence: Teleoperators and Virtual Environments*, 6(4):355–385.

Berczuk, S. (2007). Back to basics: The role of agile principles in success with an distributed scrum team. *Agile Conference (AGILE), 2007*, pages 382 – 388.

Bondo, J.Barnard, D. B. D. (2010). *iPhone User Interface Design Projects*. Apress.

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, USA. AAI9980887.

Hampel, T. (2001). *Virtuelle Wissensräume*. PhD thesis, Universität Paderborn.

IEEE (1990). Ieee standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, pages 1–84.

Junaio (2014). Why to become a junaio developer. Slideshare presentation, http://www.slideshare.net/metaio_AR/why-to-become-a-junaio-developer, last checked 14.09.2014.

Maximilien, E. Michael. Williams, L. (2003). Assessing test-driven development at ibm. In *Software Engineering, 2003. Proceedings. 25th International Conference on*, pages 564–569. IEEE.

Mesbah, A., v. D. A. (2007). Migrating multi-page web applications to single-page ajax interfaces. *Software Maintenance and Reengineering, 2007. CSMR '07.*, pages 181 – 190.

Rodriguez, A. (2008). Restful web services: The basics. *Online article in IBM DeveloperWorks Technical Library*, 36.

Schwaber, K. and Beedle, M. (2002). *Agile Software Development with Scrum*. Pearson.

Statista (2014). Global apple iphone sales from 3rd quarter 2007 to 3rd quarter 2014 (in million units). Website, http://www.statista.com/statistics/263401/global-apple-iphone-sales-since-3rd-quarter-2007/, last checked 13.09.2014.

Sutherland, J. Jakobsen, C. (2009). Scrum and cmmi – going from good to great. *Agile Conference*.

Trelle, T. (2014). Mongodb. *JavaMagazine*, 5:56–62.

Wikimedia-Foundation (2014). About us. Website, `https://www.wikimedia.de/wiki/%C3%9Cber_uns`, last checked 12.09.2014.

Wikitude (2014). Augmented reality for multiple platforms. Website, `http://www.wikitude.com/products/wikitude-sdk/`, last checked 14.09.2014.

STATUTORY DECLARATION

I hereby declare that I have developed and written this thesis on my own, and no external sources were used except as acknowledged in the text and footnotes. The thesis in this form or in any other form has not been submitted to an examination body and has not been published.

Paderborn, December 28, 2014

Jörg Amelunxen