Final Report

Scansistant

an android application on behalf of the TU Delft to test the feasibility of Near Field Communication in mobile learning

Version 1.2 **August 6, 2013**

Delft University of Technology
TU Delft Library

Authors:
Dino K.L. Hensen 1367412
Tim M. Ypeij 1386174

Company Supervisor:
Karin Clavel

Supervisors:
Dr.ir. Koen Bertels
ir. A.T. Nelson

Coordinators:
Dr. Martha A. Larson
Dr. Hans-Gerdhard Gross

Summary

Nowadays a lot of students have smart-phones that can be put to use for mobile learning. The TU Delft Library has come up with an idea for this which also incorporates the use of a modern technology called Near Field Communication. It is already available on the market but immature so experience has to be gained in combination with mobile learning to see if it is worth to pursue further developments on this area.

The TU Delft Library has the idea of using a mobile application to scan machinery on the campus as a way to get instructional videos and information to perform a feasibility study and to gain experience on the aforementioned area of interest. From the wishes of the client and our own acquired computer science skills the requirements came forth and as a start of the software development.

For the requirements of the final delivered prototype application we carefully looked if the features could help to perform the feasibility study and are able to answer the questions for the this study. From these requirements we conclude that more than a simple Android application has to be built: a central server to house all video and data was also required and a way to provide communication between this server and android client application. For this we defined an API so that many clients can get information from the server.

The software subsystems consist of a back-end, front-end and client application. They are developed in Java, building upon the Android SDK and in PHP, building upon the Zend Framework 2. For the NFC technology libraries are available so that tags could be successfully scanned after which the machine data could be downloaded. For development of all subsystems we used Eclipse and associated plugins to enable for Android and ZF2 development.

The design of the subsystems was done in parallel and on-demand so that features required by the android application that needed an API method that was not yet there was immediately implemented when needed. It started with a global database design after which we could both focus individually on the platform specific details. The user interface design was important because that is the level at which the feasibility study will be performed when the end-user gets to use the application. The design focus was on smart-phones, but because tablets are widely used also we made it compatible making use of its bigger screen.

Because of an agile and feature driven development methodology we used exploratory testing for the largest part of the project. After we had a stable system we started writing tests because we now had gained enough knowledge of the development platform to efficiently write tests. At the end we managed to have tested the Android application by unit testing the models and the API was tested by a controller testcase to test the responses of critical API methods.

For the scope of our project we did not need to perform the feasibility study ourselves, but we did ran some user evaluation trials to get a first reaction on the application before delivering a final prototype. The feedback that we got from these user evaluations allowed us to increase the chances at success because the first issues we got from users could solved before delivering a prototype.

The final prototype meets all requirements we planned to have and executes its task to scan a machine and show an instructional video very well. The application works on Android phones with the 4.0 version or higher and we think is a good platform to further develop. The Software Improvement Group evaluated our code and rated it at 4 out of 5 stars on their maintainability model, which means the maintainability of our total software system is above average.

Preface

This report describes the development of a mobile application made for the TU Delft Library to perform a feasibility study and gain experience with mobile learning. It is the final report for the *TI3800 - Bachelor project* which is part of the bachelors curriculum of Computer Science at Delft University of Technology. The project lasted 14 weeks starting from April 22, 2013 until August 7, 2012.

We would like to thank the following people for their time, support and contribution throughout the project:

- Karin Clavel (TU Delft Library), for giving us the opportunity to do this project and helping us to set everything up and finish the project with a good feeling.
- Koen Bertels (Computer Engineering), for taking on our project .
- Martha A. Larson and Hans-Gerdhard Gross (Software Engineering), for coordinating the bachelor project.
- Andrew Nelson (Computer Engineering), for guiding us through much of our report writing and giving us lots of tips about what we should do and should not do.
- Eric Bouwers (SIG), for reviewing our source code and giving us feedback on how to further improve the quality and maintainability of our software.
- Gerard van Vliet (IWS 3ME), for helping to record an instruction video for the laser cutter at IWS.
- Don and Roel (PMB IO), for helping to record an instruction video for the disk sander at PMB.

Dino Hensen
Tim Ypeij

Delft, August 1, 2013

Contents

Chapter 1

Introduction

In this chapter we will explain the problem we want to solve during this bachelor project, as well as some background information. We also state our proposed solution here. Finally, a global outline of our report is given.

1.1 TU Delft Library

The TU Delft has got a department called the ICTO (ICT in Onderwijs) that comes up with all kind of ideas on how to improve mobile learning for students [1]. They would like to gain more experience with using smart-phones and they have come up with an idea of an application that uses the Near Field Communication technology.

They have also noticed that there are a lot machines on the campus that are difficult to operate and require an instruction before usage. At TU Delft many courses have a laboratory component or require the use of machine workshops in which we can find those machines that are hard to operate. So the ICTO wants to run a feasibility study with a mobile application on a smart-phone that has NFC capabilities to see what happens when students can just scan a machine with their phone and get an instructional video. The TU Delft Library facilitates our project by assigning our company client (Karin Clavel) to us as well as giving us a working place.

1.2 Problem Description

This brings us to the problem description of our project, which is to design and implement an application suitable for the TU Delft to perform a feasibility study. The main questions that need to be solved in the feasibility study performed by the ICTO when the application is done are:

- What if students could use their smart-phones to *scan the equipment*, *watch* an instructional video, click through the *steps* one by one and learn from tips & tricks by other students?
- What if we could *upload* and share *pictures or a video* from our experiment or the prototype we made with the machine?

- What elements need to be added in order to make mobile learning a success?
- Is mobile learning an interesting area to further explore, or is it not worth the time and resources?

Our task is develop a prototype of a mobile application. With developing a mobile application for a specific purpose we hope to create the platform that is necessary to have in order to answer the main questions. Answers to these questions can be gained by releasing our mobile application at the TU Delft campus for students of faculties like Industrial Design or Mechanical Engineering. Especially these student groups are required to use machines that are at first difficult to operate. These machines require instructions that can be provided by our application. After releasing a simple survey can be held to actually get feedback from the students as to whether a mobile learning solution like ours is feasible or not.

For our project we will not be releasing and surveying to such a large audience, because this is not in the scope of our bachelor project. Instead, we will evaluate it on a smaller scale at the TU Library office and instruction labs around the campus such as IWS (InloopWerkplaats Studenten), which is a workshop where students have access to machines for metal and plastics processing. By doing this on a small scale, we can still get some early feedback to filter out the some issues on the road in order to increase chances of success in the future.

1.3 Outline

We will discuss some methodologies we used during our project in chapter 3. After that, we will give our reader a closer look into the design and implementation of our prototype in chapter 4. An overview of our testing methods is given in chapter 5, while the process of our project is described in chapter 6. Several parts of our project are evaluated by others and that is described in chapter 7. Finally, in chapter 8 we will recap on the problem statement and conclude if our prototype will fulfill the questions, as well as give our client some recommendations.

Chapter 2

Requirements Analysis

In the following sections we describe the requirements of our project. We composed these requirements out of many sources. First of all, our client adviser had drawn up a proposal for us. We had weekly meetings with her were she gave us more requirements or advices. Our project supervisors gave us some useful tips as well. Also, we brainstormed a lot about the requirements ourselves about many additional functions. We also conducted a field study around the campus were we asked lab instructors and students about what they would expect out of a mobile learning application. We documented those findings in the preliminary report (Appendix A).

2.1 Domain Analysis

The domain on which the analysis is performed is mobile learning. In this domain, the end-user or customer will be the students that are required to learn how to operate specific machines on the campus. The environment for this analysis is the Delft University of Technology Campus, because that is where the machines are located. Because the focus of this domain is all about mobile learning, which means that the end-user will not be sitting at a desk, but instead will be in the context of the subject under study, it is required to look at devices that are mobile phones. Because it was made specific in the project proposal, the use of NFC gets a central role here. The only currently available platform with NFC capabilities is Android, so we will target Android devices. We will focus on users with smart-phones, but our application will run on Android tablets as well. In the current situation when students want to know how to operate a machine they have a few options to gain knowledge:

- Consult the manual if there is one available
- Follow the special lab instruction lectures
- Ask the workshop instructor for guidelines or instructions
- Search for themselves on the Internet: usually through a popular search engine or using YouTube for a video instruction.

There are already many software solutions for watching a video to gain knowledge about some subject, but have not yet seen one where you can retrieve it while you are in the context where you would like to use it.

2.2 Functional Requirements

Each requirement is assigned a priority according to the MoSCoW-model. The *M* indicates a must have requirement, *S* indicates *C* a could have and *W* indicates that the requirement will not be fulfilled.

#	**Client application Requirements**	**Priority**
F1	Capability to scan a machine via NFC technology.	M
F2	Capability to manually enter the machine code.	C
F3	Capability to scan a QR code attached to a machine	C
F4	Capability to authenticate through OAuth provided by TU Delft API	W
F5	Capability to register a user account.	S
F6	Capability to login.	M
F7	Capability to see an instructional video.	M
F8	Capability to see information on a machine if available.	M
F9	Capability to view and leave comments.	M
F10	Capability to add a machine to a favorite list.	C
F11	Capability to upload an image.	M
F12	Capability to upload a video.	M
F13	Capability to have chapters that guide the user to a point in a video.	S

From the requirements above we can immediately see that a client-server model is required to create the application. (See chapter 4 for more information). The server will provide the videos and machine information data to the client.

At the time of developing, the TU Delft API for OAuth authentication was unfortunately not ready yet. Due to this we decided to implement our own registration model.

#	**Server application Requirements**	**Priority**
F14	Capability to serve information for a specific machine.	M
F15	Capability to add comments on a machine.	M
F16	Capability to serve comments made for a specific machine.	M
F17	Capability to serve video data for specific machine.	M
F18	Capability to serve chapter data for specific machine.	M
F19	Capability to serve a video.	M
F20	Capability to provide user registration.	M
F21	Capability to authenticate a user.	M
F22	Capability to manage machines.	S
F23	Capability to add/edit/delete videos.	S
F24	Capability to add/edit/delete locations.	C
F25	Capability to add/edit/delete departments.	C
F26	Capability to add/edit/delete chapters.	C
F27	Capability to serve favorites machines for a user.	C
F28	Capability to upload media files for a comment.	M

2.3 Quality Requirements

These quality requirements should ensure that the user experience of our application is of decent quality.

#	**Requirement**
Q1	The server should be able to provide information data and video to the user in a few seconds to have the best user experience
Q2	The server should be able to serve video's to multiple clients at once. (Because the deliverable is a prototype we won't focus on scalability too much.)

2.4 Platform Requirements

The server application is supplying the client with information and must always run in order for users to use the client application. We have chosen to use a LAMP stack and therefore the server software requires the following system to run on: see table.

#	**Requirement**
Client	
P1	The platform to run the client software is an Android device(So either a smartphone or a tablet that runs Android).
P2	The minimal Android version is 4.0 (API 14), because the most NFC phones are equipped with at least this version and this was the last fundamental Android update.
P3	The phone must be connected to the Internet.
Server	
P4	Server application depends on: Apache 2.2.22, PHP 5.3.10, MySQL 5.5.31 on Ubuntu 12.04.2 LTS.

2.5 Security Requirements

These requirements should prevent any abuse of our application, server and database.

#	**Requirement**
S1	The user must be authenticated before adding data such as comments, media uploads and favorites
S2	The password the user provides when registering and logging in should be encrypted.
S3	When a user is not logged in, he should not be able to post comments.
S4	A tutorial video could be prevented from streaming when not accessed from the application (but i.e. from a browser)

2.6 Process Requirements

At the start of this project we were determined to use Test Driven Development, but in the first weeks we came to the conclusion that it did not fit this project. The reasons were that we did not have that much knowledge of the frameworks

we were building upon. These frameworks (described in chapter 3) have a lot of classes and patterns ready to use out of the box, but in order to do a test-driven approach, we should have had a deeper understanding of the frameworks we used in order to write tests up front.

#	**Requirement**
PR1	As development process or methodology we will use a feature driven development in an agile manner. You can read more about this in chapter 6.

Chapter 3

Methodology

In this chapter we describe the technologies, libraries and tools we have used during our bachelor project.

3.1 Scan technologies

Currently there are two different methods of scan-identifying objects. One method, called Near Field Communication, is relatively new. The other one, QR-code, is widely used already. We will discuss them both and explain why and which one we chose to focus on. Also we included here how to use both technologies to make them compatible to our mobile application.

3.1.1 Near Field Communication

There are many NFC-chips standards. We chose the NTAG203 NFC standard, because it is suitable for the latest Android versions, the versions that we are developing for. At some webshops (i.e. 123NFC.nl [2]), there are special NTAG203 sold that are suitable for placing on metal. When normal NFC tags are placed on a metal surface, the conduction causes errors in reading the tags. These special on-metal NTAG203 tags, have a special coating that prevents the conduction. Because of the fact that many machines at the TU Delft are made of metal, these special on-metal tags come in handy to prevent the communication between the phone and tag from malfunctioning.

To make a NTAG203 (or other NDEF format) ready for our application, we use an Android NFC capable phone with the NFC Developers application [3]. With this application, you can scan a QR-code and the content of the QR code can then be burned onto a NFC tag. These QR-codes can be generated on a website called NDEF Editor [4]. A short instruction of how you make your tag ready for your machine follows:

1. Go to `http://www.ndefeditor.com`
2. Add a text record with your machine code (in the format `UUU-DD-###`, where UUU is a three letter abbreviation for the University, DD a two letter abbreviation for the Department and ### a three digits code).
3. Add a Android Application Record with package name: `com.scansistant`.

4. Open or download the NFC Developers application, and scan the generated QR-code with this application.

5. Scan the tag you want to store your machine code on (You can activate Burn modus to prevent other people overwriting your tag).

When a tag is prepared for use, it can be stuck onto a machine. It is recommended to write or print the machine code on the tag for users with Android phones without NFC capabilities. When the machine with corresponding code exists in the database, the tag can now be scanned and will force the Scansistant application to open and loads the machine data from the server. Also with the application already opened, a tag can be scanned and the user will be directed to this certain machine.

One of the benefits of using NFC technologies is the possibility to protect the data that is stored on a chip. Anoher one is that the material is robust so it is sufficient for lab environment. A final advantage is that it is a new technology that has not been widely used by now, so our application will contribute to the study of a relatively new field. On the contrary, NFC chips are relatively expensive, and not every (Android) phone is capable of reading the NFC tags.

3.1.2 QR-code

A QR-code is a two dimensional bar-code that is capable of carrying data with itself. It is mostly used for containing URLs with it. We already use it to burn our NFC-tags as described above. We also built in the functionality to scan a QR-code, containing mostly the same information as our NFC-tag, with an external QR-code scanner and launch our application and load the machine that corresponds with the machine-code that is contained in the QR-code. It is unfortunately not possible at this moment to use the same QR-code as we generated above, because this QR-code contains meta-data for the NFC-developer application. You can generate a QR-code as follows:

1. Go to `http://goqr.me/`

2. Click on the tab URL

3. Give the URL: `scansistantQR://<machine-code>` where `<machine-code>` is in the format `UUU-DD-###`.

4. Download or open the QR-code.

When a QR-code is prepared for use, it can be stuck onto a machine. Now, aside from NFC-scanning, a phone can scan this code and this launches Scansistant with the machine data. This is useful for older phones which are not equipped with NFC capabilities yet.

The advantages of using QR-codes are that they are relatively cheap to make. Also every Android phone with a camera on the market is capable of reading QR-codes. A big disadvantage is that it is susceptible to abuse because everyone can print out a malicious QR-code and stick it onto the original QR-code. Another disadvantage is that a QR-sticker in lab environments could easily wear off and become too dirty or damaged to be read.

3.1.3 Our choice

We chose to focus on the NFC technology because we want to discover the opportunities of a relatively unexplored market. We built the NFC scan technologies into our application. We also think that it is important that NFC-tags are more reliable in terms of susceptibility to damage and misuse, because we are developing an application in the field of mobile learning. To reach owners of a phone without NFC capabilities, we also will support QR-code data that has to be read with an external QR-code scanner.

3.2 Libraries

The libraries that we have used are the Android framework for the client side of our project, and Zend Framework 2 for the server side of our project. Those two important libraries are discussed here.

3.2.1 Android Framework

The Android framework is a framework for developing mobile applications, based on the Java programming language and XML markup language. XML is used for defining constants and the graphical user interface elements and Java is used for every dynamic element. There are many different classes that are specifically for Android and not in Java that a programmer must understand before developing an Android application. Some important examples:

- Activity: an Activity is the main process that is running in an application and it interacts with the user. Usually the GUI elements are loaded in the Activity
- Intent: the description of an action the application is intended to do. For example, it can contain an Activity as well as data. Different applications can send each other Intents back and forth.
- Context: An abstract class that is mostly extended as an Activity and it is containing the global information about the application's environment.
- Asynctask: an abstract class that represents a task that is running on a separate thread. Mostly this class is extended for networking issues.

While these are a few important examples, it takes worth a while for an experienced Java programmer to understand the whole Android framework. During this final report, we will further explain some of the Android framework features especially in chapter 4.

3.2.2 Zend Framework 2

Zend Framework 2 (ZF2) is the industry standard development framework for enterprise PHP development. The first stable release (2.0.0) was on 5 September 2012 and year later the framework has matured and evolved to version 2.2.x which we are using for the development of the back-end. The ZF2 is completely object oriented and uses a lot of different design patterns to increase the reusability and maintainability. The framework offers a huge amount of reusable

components from which we will only use a few. The reason we chose to use the ZF2 framework is because it is a modern industry standard and it provides us with a lot of useful components. We took the components we thought we required for this project and took them as criteria for deciding whether use ZF2 or to write our own small framework:

- Modular design
- MVC layer
- Form filtering and validation
- Database object pattern
- Authentication mechanism

Because ZF2 has all the above components we chose to use it as it would save us time to write it all ourselves. The reason that we did not choose another frameworks is that it was important for the maintainability. It is easy to find certified Zend engineers, also there is a large community that offers supports and all code and manuals are documented very well.

3.3 Tools

We have used different essential tools during the development of our project.

3.3.1 Eclipse

Eclipse is a open-source IDE originally designed for developing Java code. When equipped with the ADT (Android Development Tools) plugin, it is ready for developing Java code on the Android framework. We also use Eclipse to program the server application subsystems in PHP for which a plugin is also available.

Eclipse is our coding environment of choice because we have much experience with it throughout our study and work, and it makes the life of a programmer so much easier. It has so many convenient hot-keys as well as code highlighting. It also compiles your code while typing and alerts you of syntax or other errors. Another important motivation for using Eclipse is that we don't need several editors: because of all the plugins we can simultaneously edit PHP and Java in the same instance of Eclipse.

Android Development Tools

The ADT plugin for Eclipse is a very useful plugin for developing Android apps. First of all, when starting a new Android project with Eclipse, it automatically adds all the Android framework classes (from the target API of Android you are developing for) to the classpath.

Aside of that, the graphical user interfaces (GUIs), predefined data such as strings and color codes and the important Android Manifest files in Android applications are defined with XML. The ADT plugin makes Eclipse capable of developing this XML code. Also regarding the GUI, the ADT Plugin can gives a actual preview of XML defined user interfaces. The other way round is also possible, when dragging and dropping several different graphical items into

the preview windows the XML code is automatically generated. When XML items are defined, you sometimes want to access these items in the Java code. Id names can be defined in the XML elements and then the ADT plugin puts them as a constant into a special R (resources) file. From then on, these items can be referred from the Java code.

Another great functionality of the plugin is that it adds an emulator to your environment. For example, emulators can be used to test applications with older versions of Android than the target version or to check how it looks on a tablet when there is no tablet to test on. There are more functionalities for testing more automatically in this plugin but these are discussed in chapter 5.

A final useful functionality of the ADT plugin that we want to highlight here is the LogCat. The LogCat is capable of getting the console output out of a phone or emulator. It is possible to let the LogCat print out any output (much like the `system.out.println()` function in standard Java), but supply them with a tag and filter on these tags. This comes in handy when debugging a specific issue. Also, different priority levels on the output and also filter on these levels. When the highest level occurs, the application closes immediately. These are often implemented in 'dangerous' errors in try/catch statements.

PHP Development Tools

For the subsystems that are programmed using PHP we installed another plugin called PDT (PHP Development Tools). This plugin ensures that Eclipse is capable of working with PHP code as well. PDT offers php syntax highlighting, code hyperlinking, autocomplete and also supports namespaces and automatically adds a `use namespace` line whenever using a new class in the source-code.

3.3.2 Pivotal Tracker

Our agile development requires a project tool that allows for an agile work-flow. We did not wanted to use a full Scrum methodology, but our own flavor of it. That means that we would not have strict roles and daily scrum meetings. Doing this would yield too much overhead since we were with two persons only. Therefore we chose to use Pivotal Tracker(PT) to put our project tasks into without being constrained in any way. Pivotal Tracker allows a task to have an point estimate assigned to it. A point estimate is a relative measure of complexity and risk, so it does not per se say anything about the hours spent on a task. PT works with iterations in which each iteration is defined a one week. It also estimated the projects point velocity, which is the rate at burning points. When starting an initial velocity can be set , but it soon finds out how much points per iteration you are able to burn. There is the ability to have a backlog full of tasks with points assigned to it. Because the backlog is prioritized Pivotal Tracker will automatically move a task from the backlog to the current iteration when there is room left to fit a task in the estimated available points according to the velocity. This way each week the velocity is averaged according to the points are burned that week.

3.3.3 Development Server

The back-end of the application will be developed in PHP, so it requires some webserver to run on. For data storage we also had to pick a database server. We simply chose the LAMP stack: A linux, apache, MySQL and PHP installation on a single server computer. It was possible to request a virtual private server (VPS) within the ICT department of the TU Delft Library, which we made use of.

Subversion

To manage our code base, we used Subversion (SVN) as our version control system. Both of us were already familiar with SVN and the installation on the server was also simple and not very time consuming. We did not use the TU Delft SVN repository, because we wanted more flexibility. The project management tool Pivotal Tracker had an option of using post-commit-hooks to automatically process commit messages and update story statuses. Unfortunately the Eclipse plugin that was necessary to accomplish this contained a bug that rendered it useless. So we might as well have used the TU Delft SVN repository after all.

Video Streaming Plugin

An requirement of our application is that it must be able to serve a video(requirement F19). A way to stream a video had to be found, which at first seemed easier than it was. Experiments with FFmpeg lead to nothing, a lot of confusion and a waste of time was the result of a fighting FFmpeg team leaving us with two versions of FFmpeg: the real FFmpeg and the Libav fork. The decision to skip these all together lead us to a successful solution. We installed an apache module called: `apache_mod_h264_streaming-2.2.7`. This allowed us to simply upload a video in the right format (.mp4 h264 encoded) to the webserver and play the video by visiting the URL.

Chapter 4

Design & Implementation

This chapter describes the design of the subsystems of Scansistant. The requirements from chapter 2 will be referenced to in this chapter to show how they have influenced design choices. It will consist of design and implementation details of all subsystems that are developed. In the first section we will explain what are the actual subsystems of our bachelor project.

4.1 Global Design

From section 2.2 : Functional Requirements, it is clear that several subsystems are to be developed. This situation is depicted by Figure 4.1.

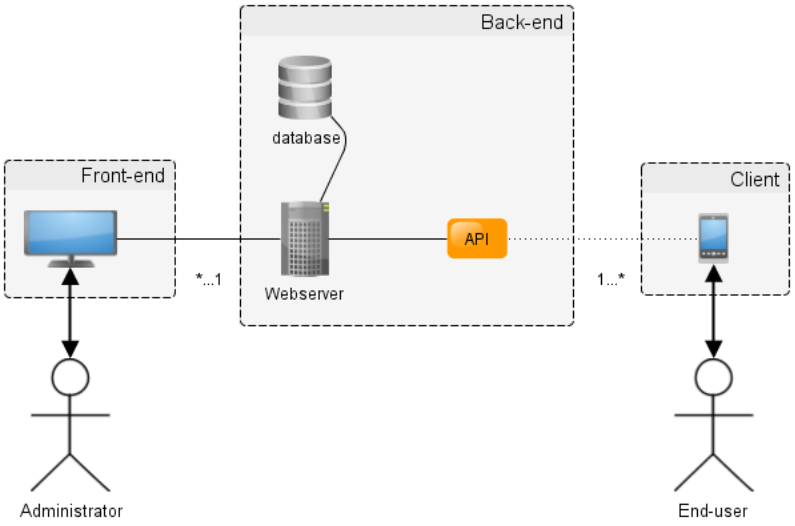


Figure 4.1: Diagram of the back-end, front-end and client and the interaction with administrator and end-user.

The subsystems that will be developed are:

- **Client** The client subsystem depends on the API provided by the back-end, without it it can not function. The client is the only subsystem that the end-user is going to use. The client will be an Android application developed in Java.

- **Back-end** The back-end subsystem provides an API (application programming interface) for the Client subsystem. It also has the models and controllers to be used by the front-end subsystem. It connects to a database to persist the model data. The back-end will be a ZF2 application developed in PHP.

- **Front-end** The front-end provides management functionality for administrators to manage the contents of the software. Every content change in the front-end results in a change in content in the client, through mediation of the back-end that provides the API to the client. The front-end will be a ZF2 application developed in PHP and will live next to the back-end.

The design and implementation will be explained from a user interface perspective in the next sections.

4.2 Android Client Application

For the user it is important to have a user interface that is intuitive and easy to understand. We designed the user interface with the use-case-diagram in mind. Figure 4.2 depicts the use-case-diagram for the end-user of the client application.

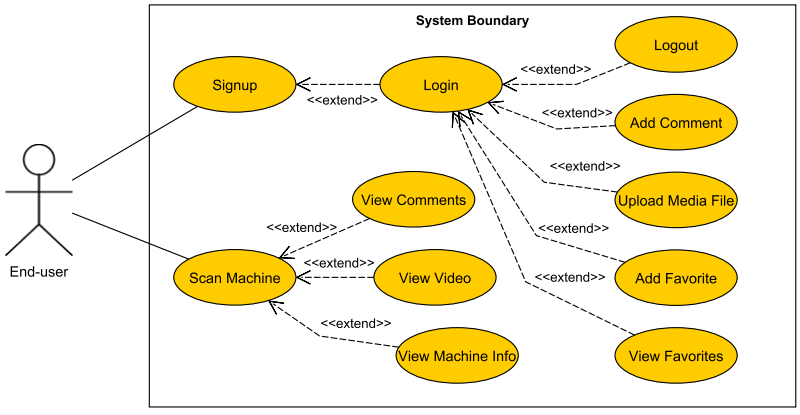


Figure 4.2: Use-case-diagram showing the end-user and actions

4.2.1 Graphical User Interface

The user interface is depicted by figure 4.3.

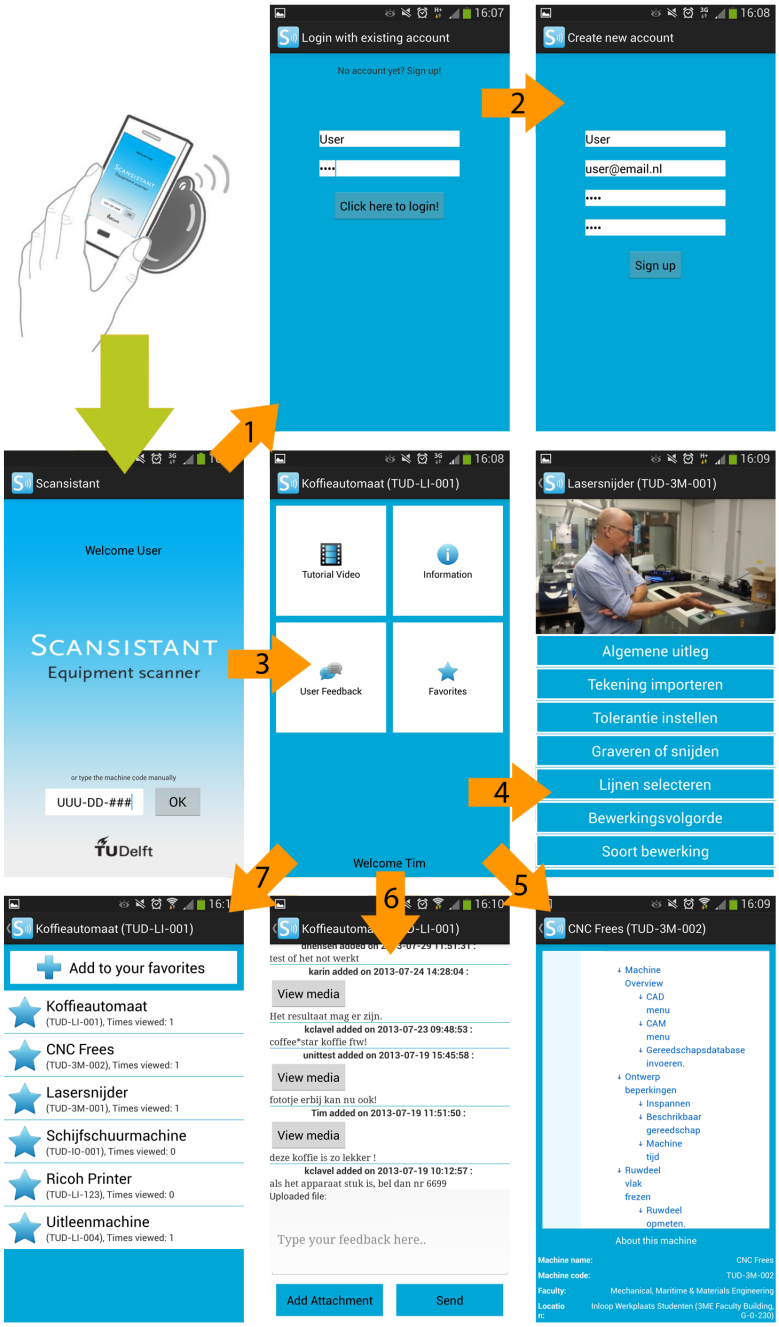


Figure 4.3: User interface flow

We chose a minimalistic design so that it is easy to use for everyone, even on phones with a small screen. The TU Delft corporate colors are used [5].

If we take a look at figure 4.3 we see the big green arrow. This represents the action that a user has either scanned an NFC tag, scanned a QR code or launched the application from the Android launcher home screen. In the first

two cases, the application automatically extract the Tag Code from the NFC tag or from the QR code. In the third case the user can manually enter the machine code and press OK when done. When OK is pressed or automated through the scanning action the Tag Code is identified on the server and all required data is being pre-fetched. This process is shown as in the following activity diagram in figure 4.4 (without the QR code action).

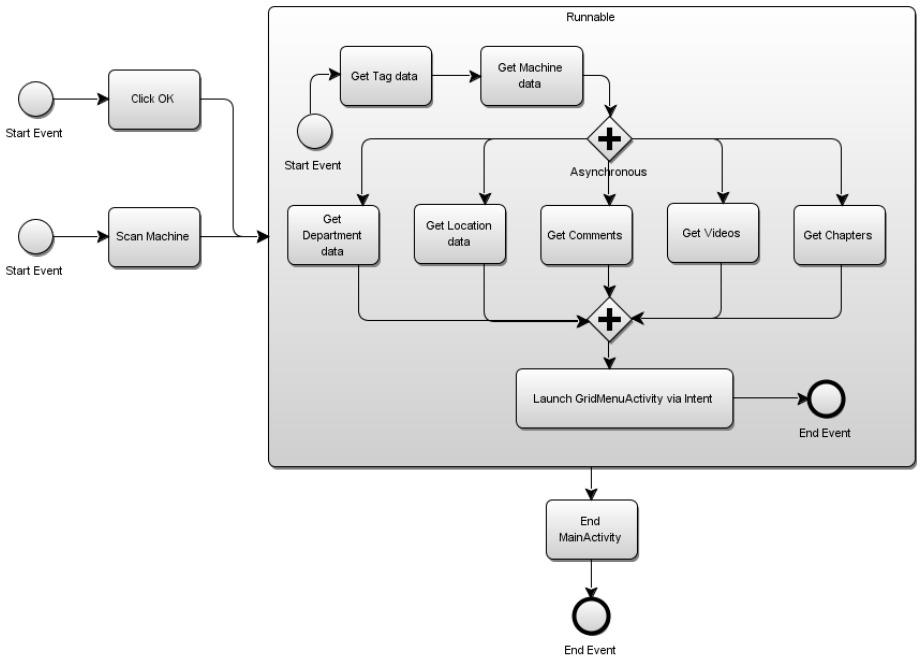


Figure 4.4: Activity diagram showing

From the activity diagram you can see that the Tag Data is fetched first. This contains the machine identifier after which the machine data can be asked. When the machine data is know, all the other model data can be fetched through the API. So here we already used 7 API methods:

API URL	**Action**	**Meets Req.**
/api/scan	get Tag data	F14
/api/machine/<machine_id>	get Machine data	F14
/api/department/<department_id>	get Department data	F14
/api/location/<location_id>	get Location data	F14
/api/machine/<machine_id>/video	get Videos	F17
/api/machine/<machine_id>/chapter	get Chapters	F18
/api/machine/<machine_id>/comment	get Comments	F16

For a more detailed description of the API see section 4.3.4 and appendix F. At the end of section 4.2.3 the classes from the `com.scansistant.network` that perform these API calls are described in detail.

4.2.2 Activities and Fragments

From figure 4.3 we see transitioning arrows that represent user interface flow. The first class diagram (figure 4.5) shows the relationship between the MainActivitiy, the GUI package, the detail package and the account package. Here also the orange arrows indicate that it is possible to navigate between the activity classes that each hold a user interface. describe that there is a button or function in the application that navigates back and forth between two activities.

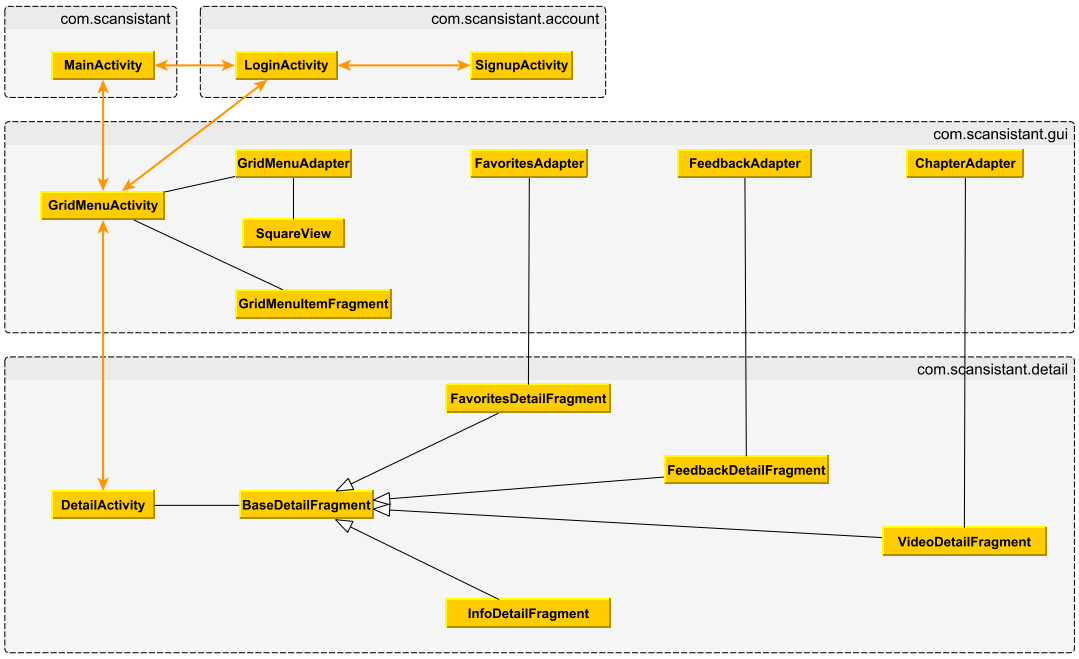


Figure 4.5: Flow between Activities is indicated by orange arrows

User interface flow between Activities is not the only flow that is present. An activity can be designed to hold fragments, so there is also flow between Fragments. This fragment structure is important because on devices with bigger screens, such as tablets or in the future phones with bigger screens, several fragments can be displayed at the same time. The GridMenuActivity is designed to permanently show us the GridMenuFragment on the left while showing a BaseDetailFragment on the right side when displayed on a device with a larger screen. This important fact makes our application ready for the future. On a normal sized Android device, only one fragment is displayed at a time. The next four paragraphs describe all extensions of BaseDetailFragment. Next we will explain each transition from figure 4.3 in terms of user interface flow.

MainActivity The MainActivity class is the core Activity of the Scansistant application. When the application is launched from the Android Launcher this is the starting screen from where the user can login, check his favorite machines or manually enter a machine code. MainActivity is also the activity that handles

the intents from the phone's NFC communicator as well as intents from QR-barcode scanners. An intent is a description of an action performed by an Android application. It can contain activities to launch or data to be processed. In the `AndroidManifest.xml` file, the core XML file where essential information such as version numbers and permissions are described, are intent-filters defined. These filters send QR-code intents which contain `scansistantQR://` or NFC intents which contain a combination of plain text and the `com.scansistant` data directly to the MainActivity, where the data of the intent is processed. The machine code in the MainActivity is extracted using a regular expression which must be met in the data of the intent. In the MainActivity, requirements F1, F2 and F3 are met.

The Login and Signup Activities For example, there is a button in the MainActivity that directs the user to the LoginActivity, corresponding with arrow 1. When the user successfully logs in he will be directed back to the MainActivity. This meets requirement F6. When the user does not have an account, he can register through the SignupActivity, which will be started when clicked on the button in the LoginActivity (arrow 2). In the SignupActivity the requirement F5 is fulfilled.

The Menu and Fragment structure When the GridMenuActivity is launched, corresponding with arrow 3, a menu appears where each button guides the user to one of the four main features (Video, Info, Feedback and Favorites) of our application. When pressing a one of the menu buttons the associated DetailFragment is loaded into the GridMenuActivity in case of a big tablet screen or by launching a new DetailActivity in case of a normal size screen. These buttons correspond with the arrows 4, 5, 6 and 7 in figure 4.3. These DetailsFragments each have their own special purpose user interface defined in the android XML format. Some of these DetailFragments contain a customized ArrayAdapter, which manages the data that an Android ListView should display. The ArrayAdapter is used because of the smart design of a ListView which reuses item views, so that the viewport is covered by an amount of views and a few extra are just outside of the viewport. The item views that are outside of the viewport are then recycled so that no new item views should ever be made when scrolling a ListView. This improves memory usage and performance. So the customized ArrayAdapter delivers the right list view with the right data when scrolling the list.

VideoDetailFragment The VideoDetailFragment launches a tutorial video uploaded by one of the administrators, instructors or teachers. If available, a list of chapters is loaded in a ListView using a ChapterAdapter and shown to the user. The user can navigate through the chapters by clicking on them. The chapters can be several points in one video, or can represent different videos. This decision is up to the one that uploads the videos. In this fragment, requirements F7 and F13 are met.

InfoDetailFragment The InfoDetailFragment shows the user the most essential information of a machine, for example the location and faculty. Also, the administrator should include a wiki page or manual which can be included

here in a WebView. A WebView is a View within an Application that acts like a browser. The user can hold the WebView for a few seconds to load the page fullscreen in his system browser. In this fragment, requirement F8 is met.

FeedbackDetailFragment This fragment is the feedback section of the application. Here, the users can discuss about a certain machine. When the FeedbackDetailFragment is launched, the CommentAdapter will display all comments belonging to the certain machine in a scrollable list view. When logged in, a user can leave a comment and/or upload his own video or photo material. This section of the application meets requirements F9, F11 and F12.

FavoritesDetailFragment This fragment is only visible when a user has logged in. The favorites of this user are loaded here in a FavoritesAdapter. When the machine that is scanned most recently is not already in this list, the user can add this machine to his favorites list. This is stored both local and on the server. Requirement F10 is met here.

4.2.3 Modeling and network

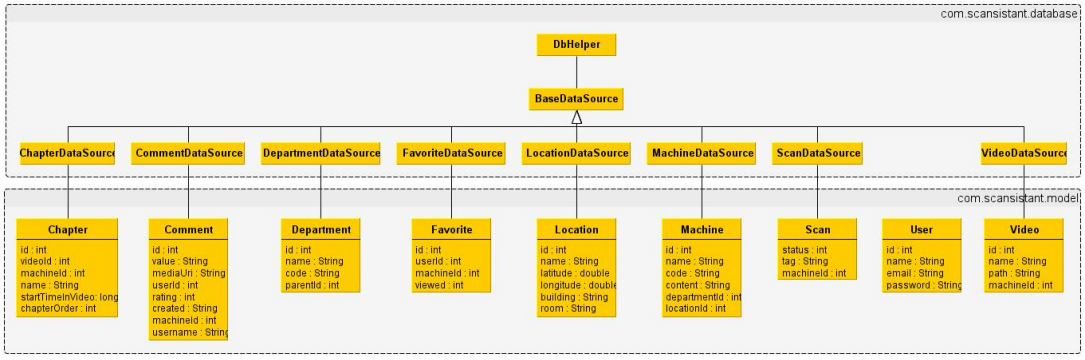


Figure 4.6: The model and database packages with their relations

In the second class diagram (figure 4.6) the local database is defined, and how it communicates with the application. The design of this database model is also implemented in the back-end more extensively. In the client application the database is less extensive because not all data from the back-end has to be saved in the client application. For the modeling of the database see section 4.3.1. The local database functions as a buffer between the application and the central database, which is called only a few times. The connection to the central database is described further on in this section.

DbHelper and the DataSources In the DbHelper class, the SQLiteHelper class from the Android framework is extended and the tables are defined. For every table defined in the DbHelper, there is a DataSource class (generalized for the same functions of each DataSource class in the BaseDataSource class) defined. These DataSource classes can put a given object into the corresponding

table. The model objects that the DataSource classes can take and return are defined in the model package.

Models The model package contains model objects such as Machine, Video, User, Chapter and so on. All these model classes contain the fields that are needed for such a type of object and getters and setters to access those fields. For example, a Machine object contains its database id, a name, a code (part of the Scan code), a content website address, a Department (id) and a Location (id). When such an object is constructed, it can be passed to the MachineDataSource. This class will insert the object into the associated table as an single row/entry. Also, the MachineDataSource can be called with a specified machine id and then returns or deletes this specific machine.

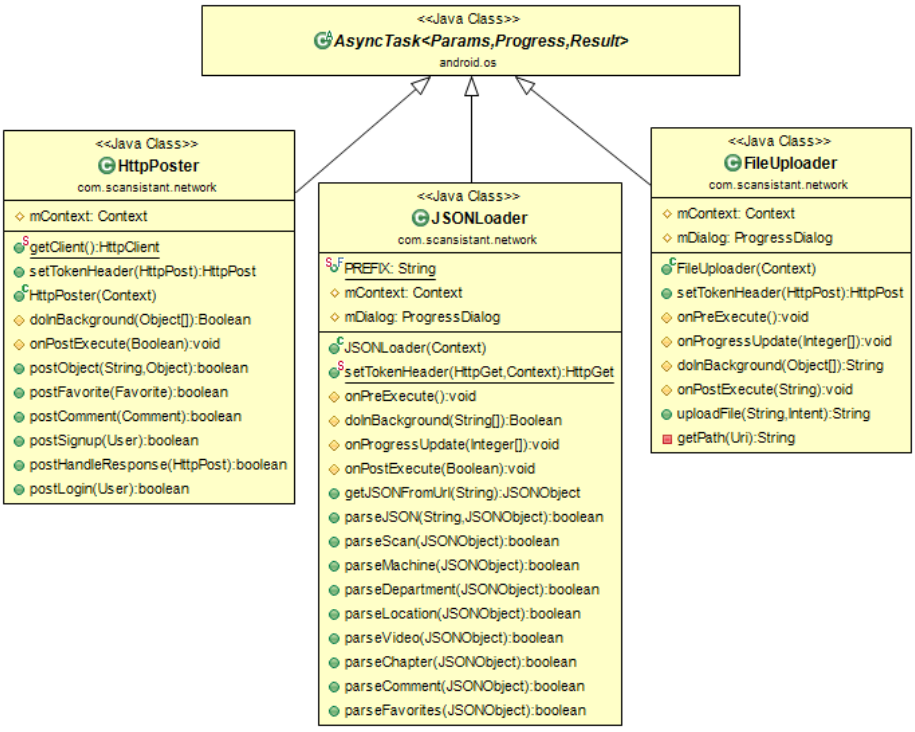


Figure 4.7: Extending AsyncTask to execute networking operations without blocking the UI thread

The network classes A detailed class diagram of the network package is shown in figure 4.7. The classes in this package are extended classes of the Android framework's AsyncTask. These AsyncTasks have the ability to run actions on a separate thread. This separate thread is called the network thread, because this thread is only used in Scansistant to perform actions to connect with the server. To prevent the application continuing on the UI thread without data that is essential for the application to continue, a new object is defined and an extra function, called `onPostExecute(result)` is defined at that place in the code. The most common example is that an object has to be loaded on

the network thread and inserted into the database and the application cannot continue without it. In this method, which runs on the UI thread, the following actions of the application after the data is successfully loaded or posted are described here. Also, there is room to inform the user when a load or post operation has failed. Finally, some of the tasks implemented in the network classes will provide a loading screen on the UI thread for the user, when the data is essential for continuing.

4.3 Back-end Application

This section describes how the server application back-end is designed. The API and front-end are built upon the back-end, that is why the latter two are explained in subsections. (If the difference between front-and-back-end is unclear, take another look at section 4.1) The server application will be developed on top of the Zend Framework 2 written in PHP. ZF2 has a modular design philosophy in which you can define an *application* that uses *modules*. It should then be able to quickly turn modules on or off, but also to add existing modules providing more functionality created by other developers on the web. This works because ZF2 conventions have a strict module definition convention that must be followed in order to have a valid ZF2-module.

For the back-end, front-end and API only two modules have been defined:

- **Scansistant** This module provides the back-end and front-end and can be used as standalone module.
- **ScansistantApi** This module provides the API and is dependent of the Scansistant module.

4.3.1 Modeling

We began the modeling process by thinking about logical objects and the data that each object should hold. The following entities were discovered:

User An end-user should be able to register for a user account. This is modeled by a user object, that should hold a user name, e-mail and password. This information will be used to authenticate an end-user when he/she want to login after registering.

Machine In the client application the machine is going to be scanned or manually entered, so this is modeled as a machine. A machine has a name, code and belongs to one department and has got a location. It must also have a multipurpose content field that can be used to hold the instruction data in HTML format or a HTTP URL that points to a web page giving the instructions of the machine.

Video A video is going to be displayed to the end-user. A video should have a name and a path. The path is the location on the file-system where the video-file itself is stored. This way the actual video-file can be retrieved and replaced without creating a new video object.

Chapter Each machine can have chapters that will link to a point in a video. A chapter is meant to explain a single aspect of an instructional video. For instance: one chapter could explain how a machine is turned on and another chapter could explain how it is turned off again. Both of these chapters can reference the same or a different video allowing for flexibility in video editing. So videos can be shot separately but one can also choose to use one big video. In both cases the chapter model also has a start time in a video. Of course a chapter must have a name, and chapter order is also important, because users need to follow the instructions step by step in the right order.

Department A department model is defined by a name and a code. A department can also have a parent department. This parent department would then be called an institute. We chose not to create a separate model for institute because this was solvable by reusing department. An institute can be seen as a large department. This also came forth out of the full machine code. For instance: `TUD-LI-001` is a code consisting of three parts: an institute, a department and a machine code. Now the institute is the parent department of the second part.

Figure 4.8: Code format depicts: department - department - machine

Location A location model stores information on the location of another model. The only model that uses a location is a machine. A location consists of a name, building and room. For future improvements we also included latitude and longitude. This would allow for GPS coordinate usage in statistical reports or integration with Google Maps.

Comment A user can make a comment on a machine, so the comment model consists of a reference to a machine and the user that made the comment. It also has a value containing the words spoken by the user. A comment can also hold a media content, which will be stored in the form of a HTTP URL pointing to the media content.

Rating Users could rate a comment by increasing or decreasing a point. A user must be able to only do this once per comment.

User has favorite Machine A user can store a machine in a favorite list. A favorite model has a reference to a user and a machine. These two references ensure the uniqueness of this model.

Machine has Video A machine can have zero or more videos. Several videos can be attached to a machine in a user defined order and with a video start time.

By designing this relationship, it is possible to have multiple videos attached to a machine and make it play as a single long video in the client application.

Because we knew beforehand that we wanted to use an object relationship mapping technique, we skipped the simple UML class diagram and went straight to the database design. From the modeling process we came to the database model as depicted in figure 4.9

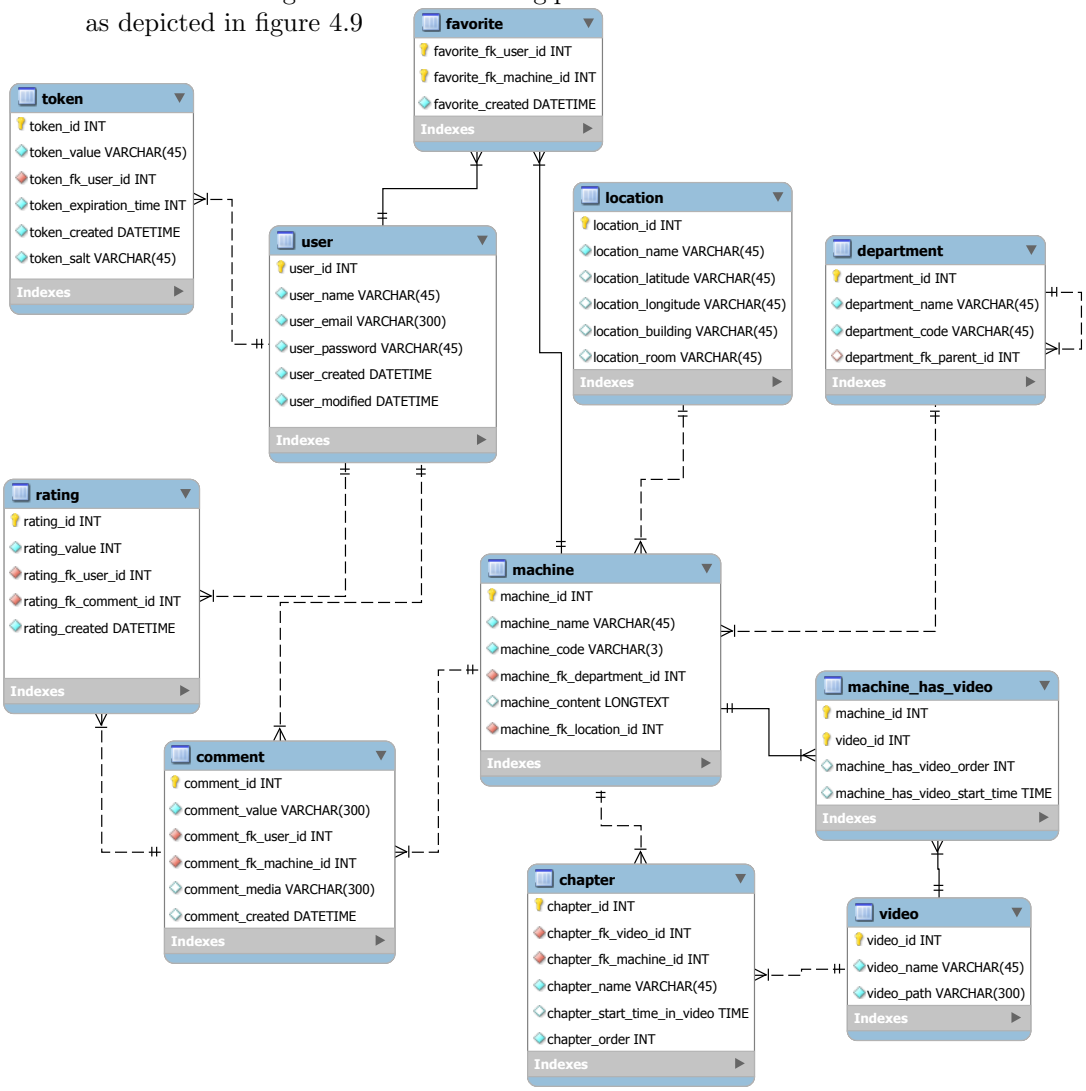


Figure 4.9: Database model

The implementation based on the database model is described in the next section.

4.3.2 Implementation

The back-end implementation begins by creating the model classes according to the database model. This can be done in a very straightforward way because we chose to use object relationship mapping technique. This technique means that

an object corresponds to one row in a database table. Thus several different instances of a model class represent several rows in the table corresponding to the model class. This is depicted by figure 4.10 from which you can see that there is an abstract class Model that all other classes in the diagram are extending from. To keep the size of figure 4.10 small the model classes' methods are omitted.

Models The *Model* abstract class shows the two methods that are obligated to be implemented by each class that extends *Model*:

- The method `getId` is to get an identifier corresponding to the primary key in the table that the object is mapped to.
- The method `getArrayForJson` is to create an array describing the model data that is ready to be converted to Json. Json is a data a lightweight data format. In comparison to XML it is lightweight, because all data can be defined more compactly, but the disadvantage is that Json lacks schema and namespace support. We don't need those features, so we choose to use Json because of the compact data representation. Because we are going to use Json the `getArrayForJson` method is important for defining the API which will be described in section 4.3.4 in greater detail.

Also the *Model* implements *ArraySerialiableInterface* which will require all model classes to implement the `exchangeArray` and `getArrayCopy` methods. The `exchangeArray` takes an array and exchanges the internal values from the provided array. This simply means that an object implementing this method can be populated by a simple array which in PHP is an ordered map with values to keys. The `getArrayCopy` on the other hand returns an array representation of the object. So basically when we have a model object and we would call `getArrayCopy` and store the returned array data and after that call `exchangeArray` with the stored array data from the previous call, we would end up having the same object if implemented properly. With this pattern implemented we can now easily map data from any datasource to the model and from any model to any datasource(database, textfile, etc.).

Mapping a table data to a model The implementation of *ArraySerialiableInterface* allows for easy mapping to the model and can now be connected to any datasource. To persist a model we are going to make use of the ZF2 *TableGateway* which in the end implements *TableGateWayInterface* that defines table manipulation methods: `select`, `insert`, `update`, `delete`. As you might suspect these methods query the database model. So these methods serve as a high level wrapper for executing SQL statements, because we are using MySQL as a database. To access a database table with a *TableGateWay* an instance of it needs to be constructed for each different table. It also needs a database adapter to access a database and finally a *ResultSetPrototype* must be given to the instance. This *ResultSetPrototype* can hold an *ArrayObject*. This *ArrayObject* is essentially our Model, because the ZF2 engineers made the ArraySerializableInterface which ensures that the `exchangeArray` and `getArrayCopy` methods are implemented. The *ArrayObject* also requires one to implement those two methods. So as a prototype we can now insert any of our models in a ResultSet for

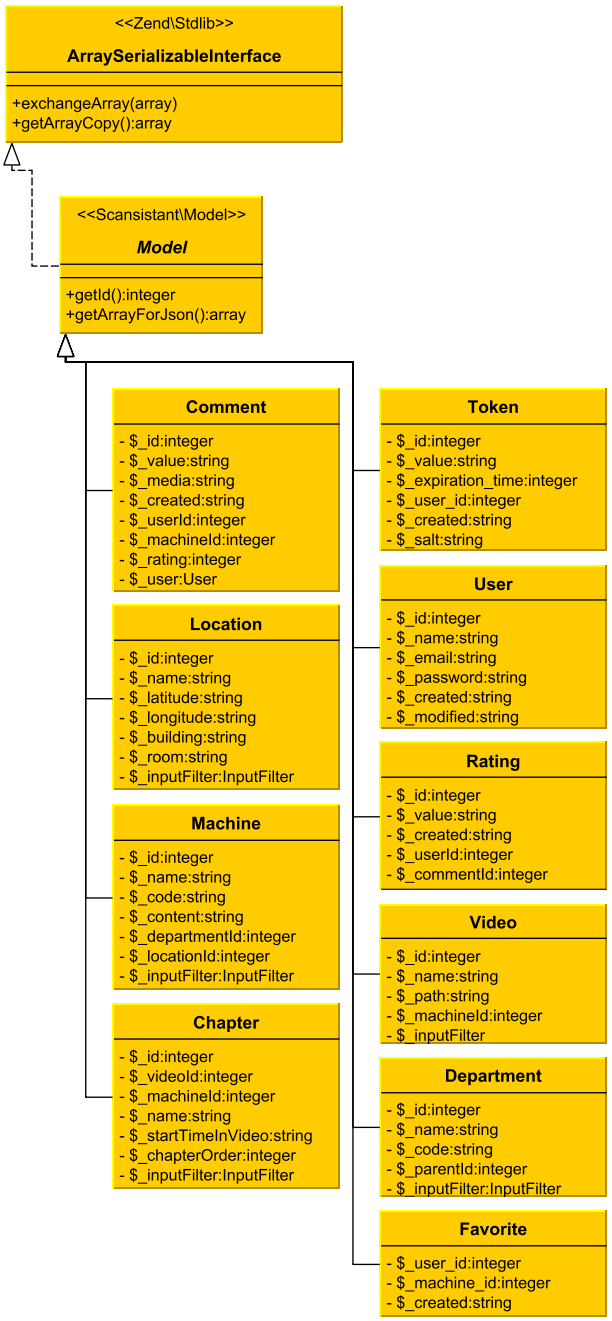


Figure 4.10: Model class diagram

a *TableGateway*. The code for creating one table specific *TableGateway* looks like this:

Listing 4.1: Constructing a *TableGateway* for the *MachineTable*

```
1 $dbAdapter = $sm->get('Zend\Db\Adapter\Adapter');
2 $resultSetPrototype = new ResultSet();
3 $resultSetPrototype->setArrayObjectPrototype(new Machine());
4 $tableGateway = new TableGateway('machine', $dbAdapter, null,
5     $resultSetPrototype);
```

So now that we have a *TableGateway* we can use this to access the underlying database model and execute queries on it, but we do not want to manually use `select`, `insert`, `update`, `delete` each time we need to access model data. Instead we design an abstract *BaseTable* that holds a *TableGateway*. This way the *TableGateway* can get a model object by its primary key. Figure 4.11 depicts the implementation of a *MachineTable* extending the *BaseTable* to illustrate how to use the it.

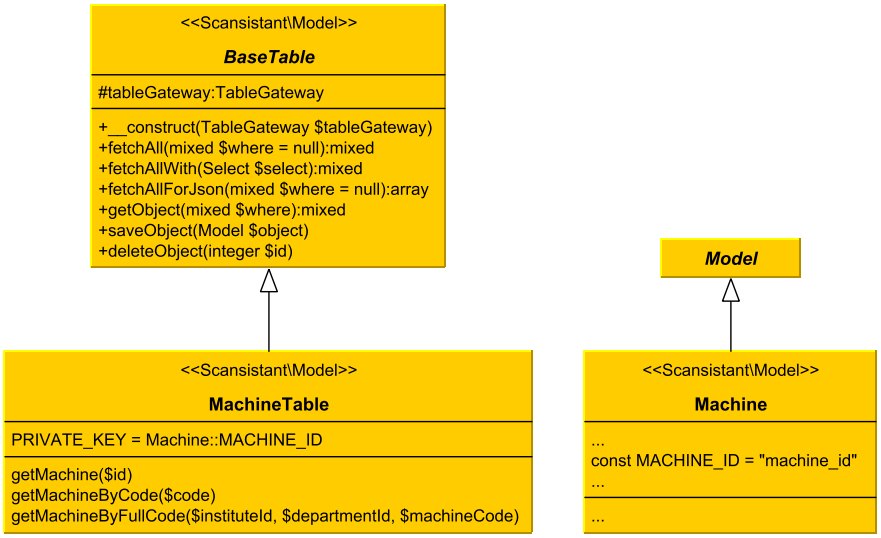


Figure 4.11: abstract class *BaseTable* holds a *TableGateway* instance

Figure 4.11 shows that *MachineTable* has a method called `getMachine`. This method calls the parent method `getObject` with an appropriate where-clause which uses the *TableGateway* to execute a select statement via the *DbAdapter*. The *ResultSet* which was injected during construction of the machine-specific TableGateway has a Machine instance as its `arrayObjectPrototype`. A *ResultSet* can contain multiple rows of a queried database table and can be iterated so that when the `current` is called on the *ResultSet*, the `arrayObjectPrototype` object is cloned(in this case the *Machine* instance) and then `exchangeArray` is called on it with the data from the row that the iterator is currently pointing to. The resulting object that is returned is a populated *Machine* object ready to use by the MVC-layer.

This same pattern can be applied to all model classes, which we have done by creating extending *BaseTable* for all model classes. Of course some of these extended classes have more advanced methods than a simple `getObject` wrapper for getting the model class. You should think of joining tables and getting model

objects by other fields than the primary key but also preparing for conversion to Json.

A new problem now arises which will be more clear if we take another look at listing 4.1. This shows how to construct a *TableGateway* for only one database table for one specific model, but now we need to do it for each extended class of *BaseTable*. Because of the dependencies between all the classes needed to form one *TableGateway* we want to use Dependency Injection(DI). The ZF2 Module class comes with a factory pattern supported by PHP closures which allows us to use DI. A closure in PHP is an anonymous function that is defined in-line and *can be used as the value of a variable*. In the *Module* class a method `getServiceConfig` is defined which will be called when the Module is loaded and the service configuration is requested. This method must return an array containing amongst others a key-value-pair in which the key is 'factories' and the value is again a key-value-pair. Listing 4.2 shows the code so the sketched situation is made more clear:

Listing 4.2: The Scansistant Module service configuration showing the factories definition

```
1  public function getServiceConfig()
2   {
3    return array(
4     'factories' => array(
5      'MachineTableGateway' => function ($sm) {
6        $dbAdapter = $sm->get('Zend\Db\Adapter\Adapter');
7        $resultSetPrototype = new ResultSet();
8        $resultSetPrototype
9        ->setArrayObjectPrototype(new Machine());
10       return new TableGateway('machine', $dbAdapter,
11       null, $resultSetPrototype);
12     },
13     'Scansistant\Model\MachineTable' => function($sm) {
14      $tableGateway = $sm->get('MachineTableGateway');
15      $table = new MachineTable($tableGateway);
16      return $table;
17     },
18     'DepartmentTableGateway' => function ($sm) {
19      $dbAdapter = $sm->get('Zend\Db\Adapter\Adapter');
20      $resultSetPrototype = new ResultSet();
21      $resultSetPrototype
22      ->setArrayObjectPrototype(new Department());
23      return new TableGateway('department', $dbAdapter,
24      null, $resultSetPrototype);
25     },
26     'Scansistant\Model\DepartmentTable' => function($sm) {
27      $tableGateway = $sm->get('DepartmentTableGateway');
28      $table = new DepartmentTable($tableGateway);
29      return $table;
30      },
31     /* other tables and gateways are left out... */
32    )
33   );
34  }
```

The factories definition allows the designer to construct the object one time in a factory closure method. When the ZF2 application loads its Module all the factory definitions are registered with the ServiceManager so that when for instance a DepartmentTableGateway is needed, it can be requested like shown in line 27 of listing 4.2. The ServiceManager is passed as an argument to the closure as variable `$sm`.

4.3.3 Front-end

The front-end provides a user interface for administrators in which they can manage aspects of the content provided to the android client application via the API. The administrator can login and do some of the following:

- Manage machines (meets req. F22)
- Manage locations (meets req. F24)
- Manage departments (meets req. F25)
- Manage videos (meets req. F23)
- Manage chapters for machines (meets req. F26)

The word *manage* means the administrator can execute some actions: add, edit, delete and view each of model specific content.

Controllers & Routing Because we are using the ZF2 MVC-layer we can let our custom controllers extend from the *AbstractActionController*. The advantage of doing this is that we use the ZF2 Routing system to dispatch the right controller for a certain URL. The URL can be defined by using segments so that we can easily extract the controller, action and id and dispatch the controller by invoking the right method with the right parameters. An example of a route definition is as follows:
/[:controller][:action][/:id]
This means that the default path `/` is accessible, but also an optional controller segment can be recognized. The action and id are also optional. Now we can define our controller classes as follows:

- MachineController
- LocationController
- DepartmentController
- VideoController
- ChapterController

All of the above defined controllers implement the `indexAction` method, so the default action of the route can be set to 'index'. The *AbstractActionController* takes care of checking if a controller has a public method `<action-value>Action` and if it does, it calls the function and eventually gives the output as response to the request. So it is now possible to call **http://scansistant.nl/machine**, because the Routing system will recognize the controller segment to be 'machine' which maps to the MachineController and the default index action is

implemented. When we implemented more action methods we automatically increase the amount of URLs that are possible to call. An overview of the controllers for the front-end is depicted in figure 4.12

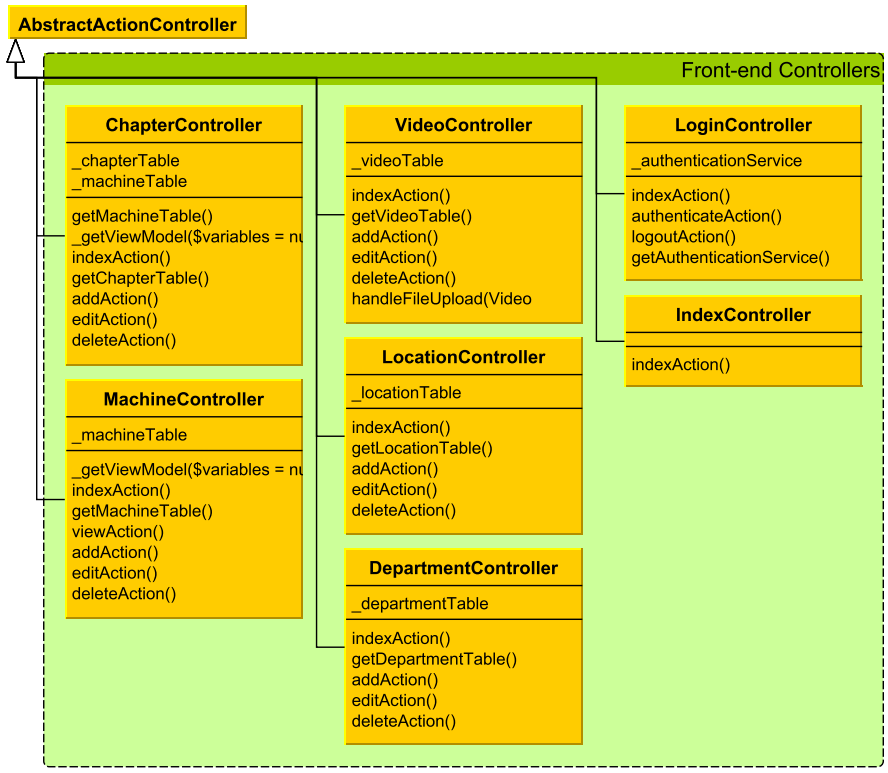


Figure 4.12: Front-end controllers extending *AbstractActionController*

The figure shows that a all controllers except for the *LoginController* and *IndexController* have `addAction`, `viewAction`, `editAction` and `deleteAction` methods. In the *MachineController* case, this makes the following URL routes available:

- http://scansistant.nl/machine/add
- http://scansistant.nl/machine/view
- http://scansistant.nl/machine/edit
- http://scansistant.nl/machine/delete

For the view, edit and delete actions the controller also requires an id, because without one the controller will not know on what machine to operate on. This is handled in the *MachineController* logic, but could have also been handled by the Routing System. We chose to handle this in the *MachineController* to avoid overly complex route definitions. Other controllers follow the same structure, but to avoid repetition we use the as a leading example *MachineController*.

View With these front-end controllers and the model introduced the **M**odel and **C**ontroller of MVC are explained, but what about the **V**iew? ZF2 uses *view scripts*, which are individual files with a .phtml extension instead of .php that contains code in a procedural style. So this file does not contain class definition, but only executes existing methods of existing classes. This works because the ZF2 *PhpRenderer*[6] includes the view script and executes it inside of the scope of the *PhpRenderer* giving it access to all its member variables through the use of `$this`.

All of the front-end controllers' action methods will return a *ViewModel* class. This class can have variables assigned to it that will be parsed in the view-script/template that is being fed with this *ViewModel*. By default the *AbstractActionController* will know where to find the view-script for a controller action by searching in the module folder for a folder with the controller name in which the file with the actionname and the extension .phtml prepended to it are found. For instance the machine overview template is: 'scansistant/machine/view', which resolves to a view.phtml file to be found in the ../scansistant/machine folder.

The result is shown in figure 4.13, 4.14 and 4.15 where some of the views are shown.

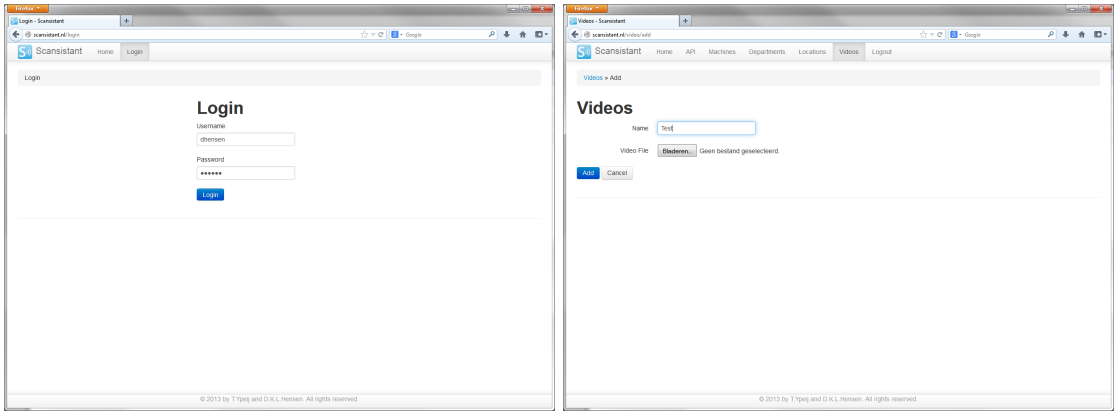


Figure 4.13: Figure showing front-end login and adding of a video

4.3.4 API

In the previous section we have seen that the MVC layer was used to create output generated by view-scripts that return HTML and are perceived as a web application by the administrator. For the API we do not want a HTML output, but instead we want to form Json. ZF2 allows for this very easily by exchanging the *ViewModel* with a *JsonModel*. Also the `view_strategy` has to be changed to *ViewJsonStrategy* so that the instead of the *PhpRenderer* a *JsonRenderer* is now used to create the final output for a response to a controller. The *JsonModel* extends the *ViewModel* and has some methods to encode array data to Json. It is now clear why we needed a `getArrayForJson` method in the abstract Model as shown in paragraph 4.3.2. The array produces by `getArrayForJson` will be fed to the *JsonModel*, that can take an array and encodes it to Json and returns

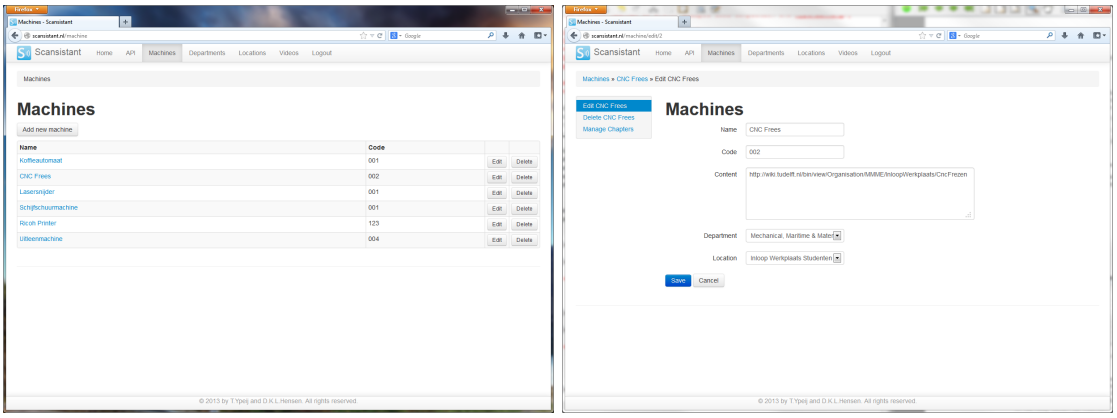


Figure 4.14: Figure showing a machine overview and editing of a machine

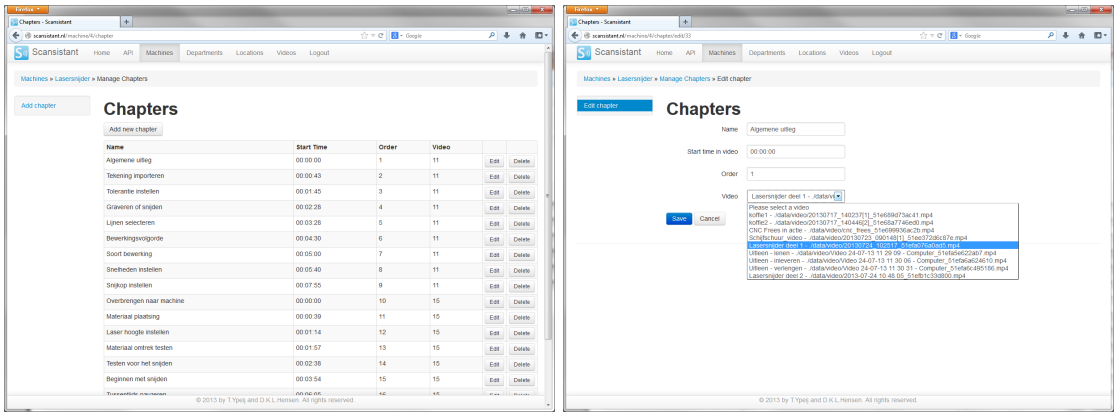


Figure 4.15: Figure showing a chapter overview for a machine and the editing of one of them

it as a response to a HTTP request with the right headers so a HTTP-client can detect that Json data is returned. This is done all automatically done by the *JsonModel*, which puts `Content-Type:application/json; charset=utf-8` in the HTTP response headers.

Like we did before with the front-end we can now define API controllers that allows us to build all of the URLs and make up the API definition. The following figure shows a class diagram of all API controllers:

The API provides URLs that can be called by the client to retrieve information. The API has publicly accessible methods, but also some methods that require a user to authenticate and retrieve a token that must be stored and used in the HTTP headers to access the protected API methods.

Public access

The following API actions can be issued without being an authenticated user.

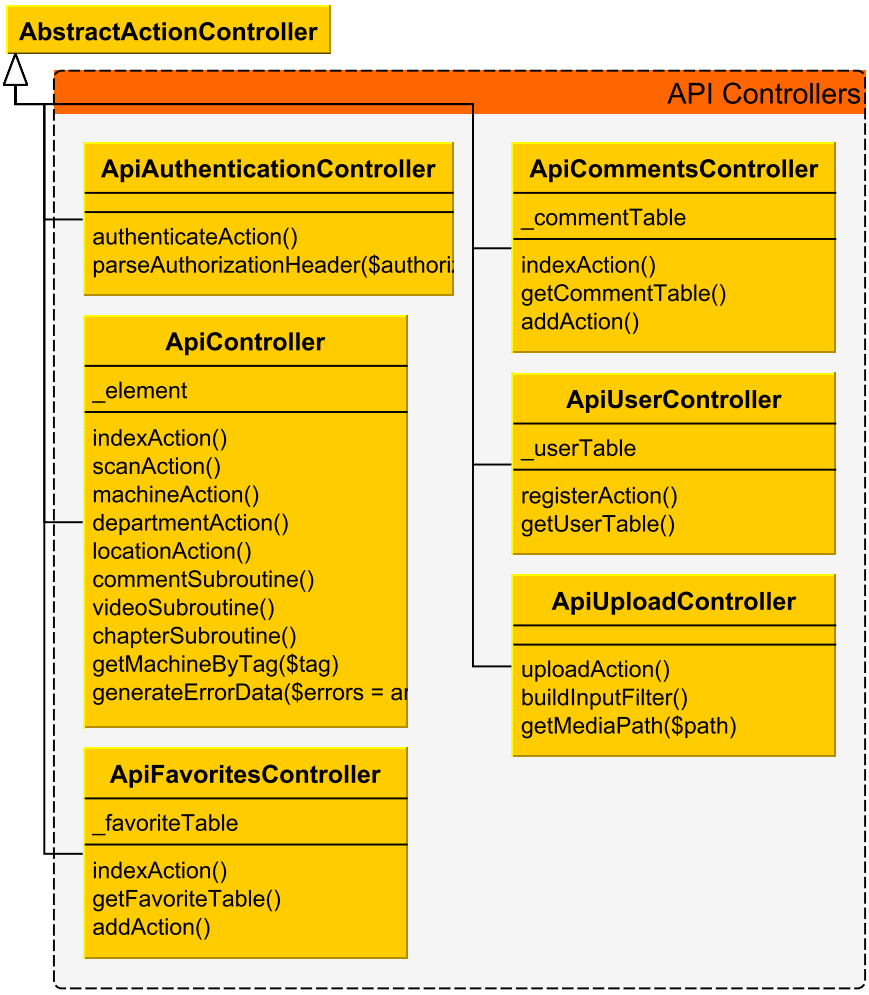


Figure 4.16: API controllers extending *AbstractActionController*

◦ **Registering a user**

URL	**/api/user/register**
Type	POST
Parameters	username, password, email

Response	**success:**
	`{"code":1,"message":"succesfully saved test"}`
	user already exists:
	`{"code":-1,"message":"This user already exists: test"}`
	missing parameter data:
	`{"code":-2,"message":"username, password and email may not be empty!"}`

Requirement	F20

◦ **Authenticating a user**

URL	**/api/authenticate**
Type	POST
Parameters	username, password

Response	**success:**
	`{"code":1,"message":"You are now logged in as dhensen","token":"42e66ac2e5ec52f9caa56b6b066c55e4aaaf6838"}`
	identity not found:
	`{"code":-1,"message":"You have entered an invalid password or username, please try again."}`
	invalid credentials:
	`{"code":-3,"message":"You have entered an invalid password or username, please try again."}`

Comments	The response codes correspond to the Zend\Authentication\Result response codes.
Requirement	F21

◦ **Scanning a Tag**

URL	**/api/scan/<tag>**
Example	/api/scan/TUD-LI-001
Type	GET
Parameters	tag as segment of the URL
Response	**success:** `{"status":"0","tag":"TUD-LI-001","machine_id":"1"}`
Requirement	F1, F14

◦ **Retrieving machine data**

URL	**/api/machine/<machine_id>**
Type	GET
Parameters	machine_id as segment of the URL

Response	**success:**
	`{"machine":{"id":"1","name":"Coffeemachine", "code":"001","content":"Insert a cup and press the button to get coffee","departmentId":"2","locationId":"3"}}`
	error:
	`{"errors":["please provide a machine id to request data for"]}`

Requirement	F14

◦ **Retrieving location data**

URL	**/api/location/<location_id>**
Type	GET
Parameters	location_id as segment of the URL
Response	**success:** {"location":{"id":"1","name":"Koffiehok 2de verdieping","latitude":null,"longitude":null, "building":"library","room":"342343"}} **error:** {"errors":["please provide a location id to request data for"]}
Requirement	F14

◦ **Retrieving a department data**

URL	**/api/department/<department_id>**
Type	GET
Parameters	department_id as segment of the URL
Response	**success:** {"department":{"id":"1","name":"Technische Universiteit Delft","code":"TUD","parentId":null}} **error:** {"errors":["please provide a department id to request data for"]}
Requirement	F14

◦ **Retrieving chapters data for a machine**

URL	**/api/machine/<machine_id>/chapter**
Type	GET
Parameters	machine_id as segment of the URL
Response	**success:** {"chapters":[{"id":"1", "videoId":"1", "machineId":"1", "name":"Koffie zetten", "startTimeInVideo":"00:00:00", "chapterOrder":"0"},{"id":"2", "videoId":"1", "machineId":"1", "name":"tweede hoofdstukjea", "startTimeInVideo":"00:00:00", "chapterOrder":"2"},...more chapters...]}
Requirement	F18

◦ **Retrieving comments data for a machine**

URL	**/api/machine/<machine_id>/comment**
Type	GET
Parameters	machine_id as segment of the URL
Response	**success:** `{"comments":[{"id":"1", "value":"praatjesmaker", "media":null, "created":"2013-07-29 10:24:20", "userId":"1", "username":"dhensen", "rating":null, "machineId":"1"},{"id":"2", "value":"praatjesmaker", "media":null, "created":"2013-07-29 10:26:37", "userId":"1", "username":"dhensen", "rating":null, "machineId":"1"},...more comments...]}`
Requirement	F16

◦ **Retrieving video data for a machine**

URL	**/api/machine/<machine_id>/video**
Type	GET
Parameters	machine_id as segment of the URL
Response	**success:** `{"videos":[{"id":"1", "name":"installing the material in the machine", "path":"http:\/\/www.scansistant.nl\ /videos\/install_material.mp4", "machineId":"1"},...more videos...]}`
Requirement	F17

Protected access

The following API actions can only be issued when being an authenticated user. The token that is served by the authentication API method must be stored and sent in the HTTP Authorization header of each request that is done to the protected API methods defined below. The format of the Authorization header is as follows:
`Authorization:SCANSISTANT-TOKEN token=2eb0d808317bad2b16590ea6aeb03512c855854a`
Where the token is a hash of a secret combination of login credentials and a generated salt.

• **Retrieving favorites for a user**

URL	**/api/favorites**
Type	GET
Parameters	-
Response	**success:** `{"favorites":[{"machine_id":"1", "created":"2013-07-08 17:22:41"},{"machine_id":"2", "created":"2013-07-29 10:23:10"},...more favorites...]}`
Requirement	F27

• **Adding comment for a machine**

URL	**/api/comments/<machine_id>/add**
Type	POST
Parameters	comment_value, comment_media
Response	**success:** {"code":1, "message":"comment successfully added"} **error:** {"code":0, "message":"Statement could not be executed"}
Requirement	F27

• **Adding a favorite for a machine**

URL	**/api/favorites/add/<machine_id>**
Type	GET
Parameters	-
Response	**success:** {"code":1, "message":"favorite successfully added"} **missing parameter data:** {"code":0, "message":"favorite already added"}
Requirement	F27

• **Uploading a media file**

URL	**/api/upload**
Type	POST
Parameters	upload_type = {image, video}, upload_file
Response	**success:** {"code":1, "message":"successfully saved test"} **user already exists:** {"code":-1, "message":"This user already exists: test"} **missing parameter data:** {"code":-2, "message":"username, password and email may not be empty!"}
Requirement	F28

Chapter 5

Testing

Because we made the choice to develop our application with the Feature-Driven Development (explained in section 6.1) methodology, it was easy for us to do the most testing by hand (exploratory testing) when we finished the implementation of one feature. We admit that we could have used more automated testing using Unit Tests and Instrumentation testing, but the implementation of these tests would have too much time overhead. So most of the time we decided to test the feature by hand and move on to the design and implementation of the next feature. In some of the cases we had the time to write Unit tests and therefore we will explain some of the technologies behind it.

5.1 JUnit 3

Android supports JUnit 3 for Unit and Instrumentation testing. The developers of the ADT-plugin tuned the JUnit 3 functionality for use with the Android Framework, to handle and eventually mock objects such as Contexts, Activities and generating test databases. To run JUnit for an Android project, you'll have to run the test packages on an Android device or emulator. For the remainder, JUnit testing on Android projects goes pretty much the same as on other Java projects.

Currently, we use JUnit testing for our MainActivity. Several functions in the MainActiviy that have nothing to do with the GUI are tested here. For example, the regular expression and login functions are tested here. This also applies to the Login and Signup activities where for example the encryption and the input fields are checked.

The model package is tested with the normal JUnit 3 test cases without making the use of the Android framework, as the model classes are not directly related to Android. All the getters and setters are tested here on normal and extraordinary input just to make sure that is free of strange errors.

The last package that is tested is the datasource package. Here the database is tested and how the datasource classes communicates with the database.

The reason that we did not test the GUI and the detail packages is that we found out that writing tests for the proper working of it is time consuming and is actually easy to test by hand. We also did not test the network classes because it would ask a lot of our time while we were far towards the last deadline

of the project. Another argument to not test these classes is that they act like a broker between the application's DataSource classes and the API and both of these sections are already unit tested.

5.2 PHPUnit

For the server side part of the application we only tested the API. This was the most logical choice because the API is a critical part of the whole system: if provides the client application with data. So in order to know if the API still functions well we tested all API methods and aimed at full code coverage for the API Controllers.

To test the PHP code we used PHPUnit 3.7.22 on our local development computer. After that some configuration files were created to start the first tests. We noticed that the API has methods that require to be authenticated but also some that do not. Also every API method returns Json response, so we made use of these facts to design abstract test cases that provided easy Json extraction and Authentication.

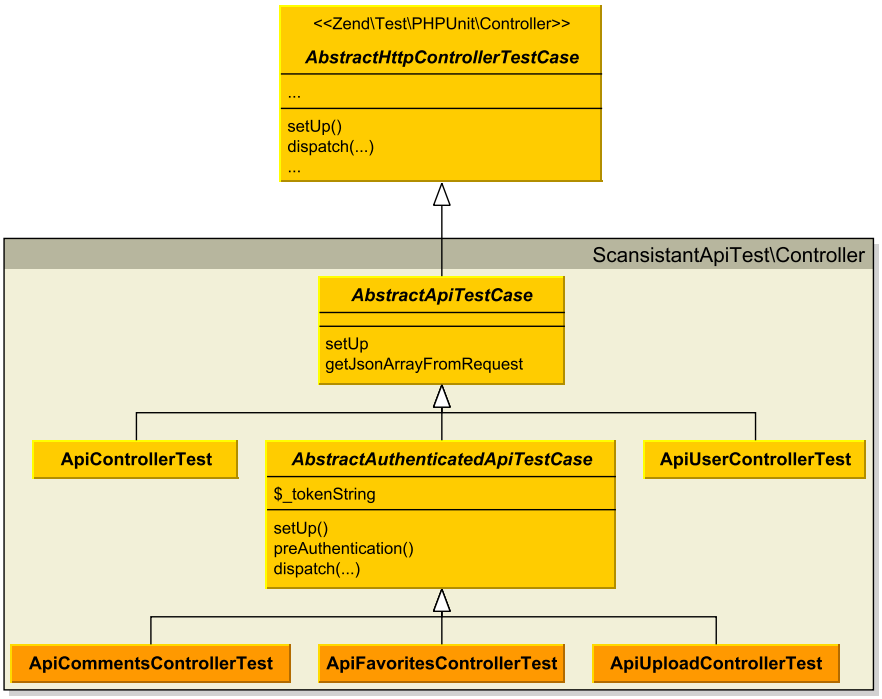
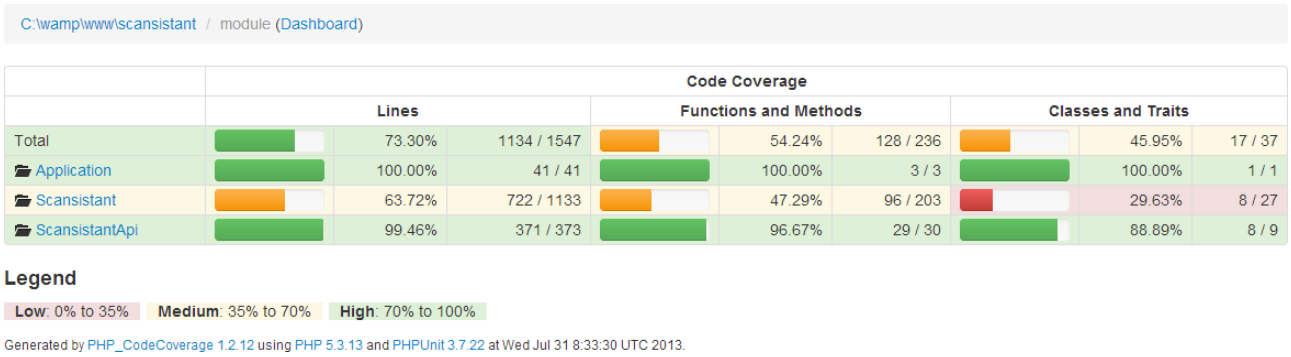


Figure 5.1: API Controller inheritance from special purpose abstract testcases. Controller testcases that require authentication are colored orange.

One major drawback of the API controller testing is that we did not define a test dataset. So the tests need at least three machines in the database and a user register. Also a first department needs to be made. If the project will be continued it will definitely be recommended that a test dataset it made or that

everything is mocked.

Coverage PHPUnit can also generate a coverage report. Our aim was to reach a 100% coverage and that is what we have accomplished. In the figure below you can see the results:


C:\wamp\www\scansistant / module (Dashboard)

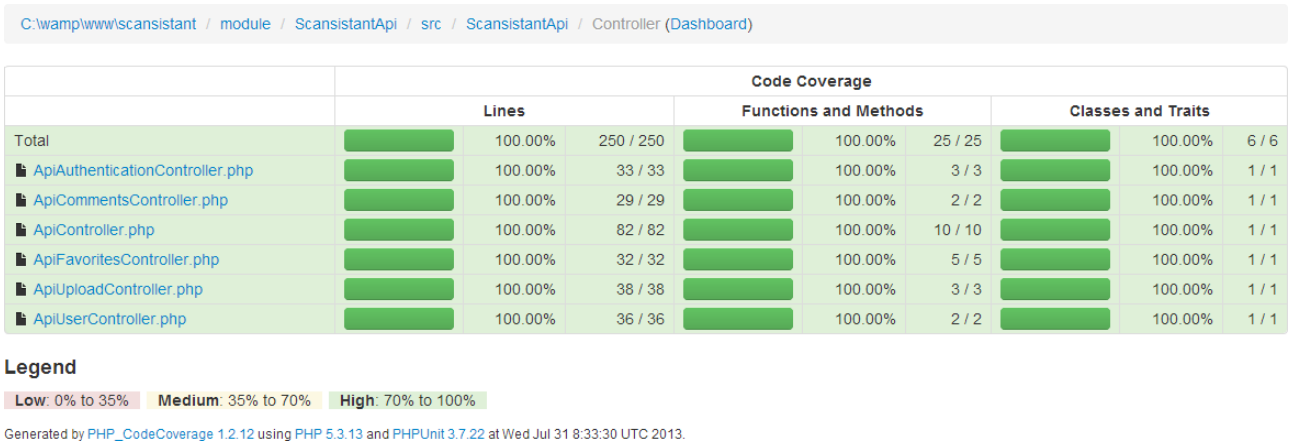
	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total		73.30%	1134 / 1547		54.24%	128 / 236		45.95%	17 / 37
Application		100.00%	41 / 41		100.00%	3 / 3		100.00%	1 / 1
Scansistant		63.72%	722 / 1133		47.29%	96 / 203		29.63%	8 / 27
ScansistantApi		99.46%	371 / 373		96.67%	29 / 30		88.89%	8 / 9

Legend

Low: 0% to 35% Medium: 35% to 70% High: 70% to 100%

Generated by PHP_CodeCoverage 1.2.12 using PHP 5.3.13 and PHPUnit 3.7.22 at Wed Jul 31 8:33:30 UTC 2013.

Figure 5.2: This coverage overview that the ScansistantApi module containing all API controllers is totally covered by tests.


C:\wamp\www\scansistant / module / ScansistantApi / src / ScansistantApi / Controller (Dashboard)

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total		100.00%	250 / 250		100.00%	25 / 25		100.00%	6 / 6
ApiAuthenticationController.php		100.00%	33 / 33		100.00%	3 / 3		100.00%	1 / 1
ApiCommentsController.php		100.00%	29 / 29		100.00%	2 / 2		100.00%	1 / 1
ApiController.php		100.00%	82 / 82		100.00%	10 / 10		100.00%	1 / 1
ApiFavoritesController.php		100.00%	32 / 32		100.00%	5 / 5		100.00%	1 / 1
ApiUploadController.php		100.00%	38 / 38		100.00%	3 / 3		100.00%	1 / 1
ApiUserController.php		100.00%	36 / 36		100.00%	2 / 2		100.00%	1 / 1

Legend

Low: 0% to 35% Medium: 35% to 70% High: 70% to 100%

Generated by PHP_CodeCoverage 1.2.12 using PHP 5.3.13 and PHPUnit 3.7.22 at Wed Jul 31 8:33:30 UTC 2013.

Figure 5.3: This coverage overview shows that all API controllers are 100% covered

If in the future the API is extended or changed and the coverage report is ran again, a developer will be able to instantly see which methods or lines are not covered anymore and an appropriate action can be taken to write more tests or leave it alone. Of course coverage says nothing about the quality of tests but it at least shows that all parts of the controllers have been reached and that there is no dead code.

Chapter 6

Process

This chapter is about our development method and the reason why we chose it. Also we describe our planning and weekly realization. Finally our workplace and accompaniment is described.

6.1 Feature Driven Development

As stated before in the requirements section, as a definitive choice during the first week of the implementation phase, we decided to switch from Test-Driven Development to Feature-Driven Development. We found out that that we had insufficient knowledge of the frameworks we were using at that time to plan ahead what features were testable and what features were not. So we decided to take our requirements as a guideline and try to complete as many features stated in those requirements each week rather than take each feature and write extensive use-cases and associated tests before actually starting to implement the feature itself. We changed our work flow in the following way [7]:

1. *Develop an overall model* We designed a prototype workflow of our application with some essential (Activity) classes on paper.

2. *Build a features list* Our features more or less follow from our list of requirements.

3. *Plan by Feature* For this we used Pivotal Tracker

4. The following steps are repeated for each planned feature:

   (a) *Design by feature* We designed each planned feature more detailed with use case diagrams and class diagrams when needed.

   (b) *Build by feature* When a design was clear enough, or sometimes the feature was trivial enough to build immediately, the feature was implemented. After a working (exploratory or unit tested) feature, we re-factored the code to make it more understandable and thus maintainable.

6.2 Planning

In our Project Plan we composed a week to week planning in terms of working hours and deadlines. Because we did not have the time to work full-time on this bachelor project, we decided to work 32 hours a week and spread the required 420 hours (15 ECTS) over more weeks. After all we think that working 32 hours a week on this final project was enough to put decent time in it, while still following remaining subjects and/or have a part-time job aside of this project. In section 6.3 we describe what we did week by week. We did not plan this beforehand because of our agile way of implementing. We had assimilated a list of requirements, but there was not a fixed order of implementing these features, we started to design one feature and implement it when we felt that we needed it at that certain point.

6.3 Weekly progress

In this section we describe from week to week which milestones we reached and which requirements have been met.

Week 1, 22-4-2013

- Started and finished the Project Plan Document

Week 2, 29-04-2013

- Started with Preliminary report.
- Conducted a field study in several labs.
- Decided to do an agile development methodology (requirement PR1).

Week 3, 06-05-2013

- Finished first draft of the preliminary report.
- Settled with the definitive name of our project : Scansistant.
- Prototyped some example applications and decided the phone's platform requirements (P1, P2, P3).

Week 4, 13-05-2013

- First menu structure finished.
- Determined code conventions.
- Requirements F2, F7 and F13 are met.

Week 5, 20-05-2013

- Decided to split up into an Android client application, API and back-end.
- Requirement F8 and F9 are met.
- Server has been set up (requirement Q1, Q2 and P4).

Week 6, 27-05-2013

- Requirements F1, F10 and F27 are met.
- Finished final version of the Preliminary report.

Week 7, 03-06-2013

- Requirements F16, F17, F18 and F19 are met.

Week 8, 10-06-2013

- Requirements F23, F24, F25 and F26 are met.

Week 9, 17-06-2013

- Requirements F5, F6 and F22 are met.

Week 10, 24-06-2013

- All work on Scansistant was paused because of the exam period.

Week 11, 01-07-2013

- Started with final report.

Week 12, 08-07-2013

- First code submittal to the Software Improvement Group.
- Requirements F15, F20, F21 are met.
- All completed client's functional requirements are made compatible with all completed server's functional requirements.

Week 13, 15-07-2013

- Planned deadline of second code submittal and final report.
- Planned final presentation date.
- Started on writing unit tests.
- Requirements F11, F12 and F28 are met.

Week 14, 22-07-2013

- Processed SIG's first feedback.
- Finished Unit tests.
- Delivered first draft of Final Report to our advisers.
- Requirement F3 is met.

Week 15, 29-07-2013

- Processed feedback on first draft.
- Finished and delivered Final Report.
- Submitted code for the second time to the SIG.

Week 16, 05-08-2013

- Included SIG's second feedback in the Final Report.
- Final presentation and evaluation.

6.4 Workplace

Our daily workplace was at the office section of the Library building at the TU Delft campus. At the second floor, there almost always was a office for the two of us available. The facilities were excellent and available for us each working day between 8.00-24.00, but usually we were at work between 9.30 - 17.30. Also, our company mentor approved that we sometimes worked at home or elsewhere at the TU Delft campus, so we were able to plan our working hours in a flexible way.

6.5 Accompaniment

Both our faculty and company coaches have given us the freedom to take full responsibility to work out our own plan. They were willing to adjust our process when we asked them to or when they saw us going in the wrong direction at any given time. We met them weekly and the conversations were always agreeable. Our company coach composed a professional usability report for us to help us improving our application. Also the project coordinators always answered in a timely manner when we communicated with them about any information. Finally, the SIG gave us very useful feedback twice and on schedule.

Chapter 7

Evaluation

This chapter describes all evaluation moments during the project. First we describe the user evaluation. The professional usability report by our client advisor is also described. Finally we discuss the code evaluation by the Software Improvement Group (SIG).

7.1 User evaluation

We have performed a small scale user evaluation at the TU Library offices, Indrustrial Design PMB workshop and Mechanical Engineering IWS workshop. This subsection describes the machines that we made a movie for at the previously mentioned places around the TU campus.

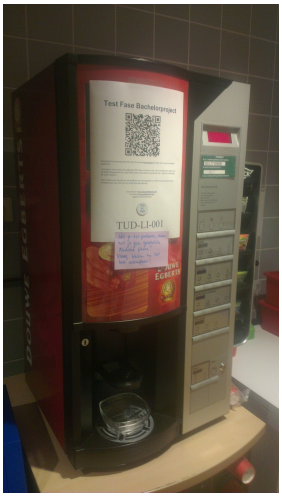


Figure 7.1: The coffee machine equiped with NFC tag and instructions for demonstrating purposes

TU Library offices At the TU Library offices we used the coffee-machine to demonstrate Scansistant for the first time. Co-workers at the second floor

tested the application for the first time and gave us some useful feedback.

Feedback for improvement given by the co-workers are:

- Multi language support
- Show in the application which chapter is currently viewed
- Choice for extra text with instructions
- Sound volume not hard enough

We incorporated some of these features in the recommendations in chapter 8.2.

Figure 7.2: The NFC tag with instructions placed in front of the laser cutter at IWS

3ME IWS Workshop After we processed the feedback and improved Scansistant we went to the IWS workshop at the faculty of Mechanical, Maritime and Materials Engineering. Together with Gerard van Vliet we recorded an instructional video for the control of the laser cutter. After uploading the movie we demonstrated the application to Gerard van Vliet to show the proof of concept. He was positive about it and will encourage students to take note of our application. Unfortunately while writing this report we did not yet receive feedback from users at the IWS.

IO PMB Workshop To increase our chances, we also went to PMB workshop, because the industrial design students think different and are more open to innovation than anybody else. We recorded an instruction video with Roel about the disk sander. We also left an instruction note with a NFC tag there, but unfortunately while writing this report we did not yet receive feedback from users at the PMB either. This instructor however gave us a very useful tip to make our application compatible with QR code as well, so we added this feature the next day. We also learned that we need to be more explicit to note what kind of platform the application uses, because the iPhone owners wanted to download the application, but of course the application only works on Android.

Figure 7.3: The NFC tag with instructions placed aside of the disk sander at PMB

7.2 Usability report

Our client advisor wrote a professional usability report for us. This is in appendix E. From this usability report only some minor improvements will be done in the software, others will be stated in the recommendation section of the next chapter. It was very useful to get a third party to look at the usability of both the back-end and the client application from a professional perspective.

7.3 Code evaluation through SIG

During our bachelor project we had the opportunity to send our code twice to the Software Improvement Group (SIG). The SIG measures the code in terms of maintainability and rates it from 1 to 5 stars. They gave us also some useful feedback and recommendations to further improve our code. The exact feedback of the two submittals (in Dutch) is included in Appendix D.

The first submittal got rated 4 out of 5 stars. That means that our code maintainability is above average. They told us that the highest possible score is not reached yet because of a lower score in the Unit Size and Duplication sections of their measurements. Concerning the Unit Size they look at the size of our methods and functions. After we received this feedback, we had split up the larger methods/functions into smaller more understandable pieces. With regard to the duplication in our code, we tried to join our most redundant code and to prevent this appearance in the future. They also told us that there were many 'TODO' comments in our code. At the time of the second submittal, we fixed all code that was marked by this type of comment and removed all these comments as well. Finally, they gave us a strong recommendation to include (more) test-code during our development phase. In the meantime we developed tests for the most important parts of our implementation.

Because we improved our code on all aspects the SIG recommended us, we expect that our rating will be improved after the second submittal. The feedback of the second submittal is included in Appendix D.

Chapter 8

Conclusion

This is the final chapter of this report and concludes the work that has been done to get to a prototype application ready for a feasibility study. The first section explains the results after which a recommendation section describes how to improve the software system in the future.

8.1 Result

This section will revisit the problem description defined in chapter 1, which was to create an application for the TU Delft to perform a feasibility study. Looking at the main questions for the feasibility study we will summarize how each question can be studies by using our application:

- *What if students could use their smart-phones to scan the equipment, watch an instructional video, click through the steps one by one and learn from tips & tricks by other students?*

  With Scansistant it is possible to give each end-user, the students of the TU Delft, the possibility to scan equipment and watch an instructional video. They can browse the steps by using the chapters and in that way get the information they need at the right moment. This allows them also to view the instruction step by step. If one step was not clear or hard to understand they can simple press on the chapter again and immediately review the step. Users are also able to leave comments or start a discussion on a machine from within the application.

- *What if we could upload and share pictures or a video from our experiment or the prototype we made with the machine?*

  Students can also make a photo or video of their experiment or prototype and share them in the comment section. This allows other students to learn from or get inspired by the ideas of other students.

- *What elements need to be added in order to make mobile learning a success?*

  Because we created a front-end administration system, a teacher could easily add a machine and notify students so that they can use Scansistant

to see the instruction. This makes it possible for teachers and instructors to add machines without notifying the developers or administrators of the application. This makes the process of adding more machines to the database more dynamic and therefore increases the chances that our system will be widely spread around the campus and maybe other institutes.

Also, we included the ability for the users of adding a machine to your favorites, so the user could access the instructions and comments of his favorite machine at any place at any given time. This is important for making mobile learning a success, because users use their phone all the time. The chances that they will use our application and therefore perform mobile learning when not in the lab increases with this functionality.

- *Is mobile learning an interesting area to further explore, or is it not worth the time and resources?*

  This last question is beyond the scope of our bachelor project, but we mostly experienced positive feedback from potential users. Even most instructors were positive about the application, except for a few where usage of the application would form a dangerous situation for the students, for example during welding instructions.

8.2 Recommendations

This last section contains three elements. First of all, there are some existing features in Scansistant that need some improvement in the future. There is also room for new features we thought about during the development and that is secondly described. Finally we give our opinion about how the project should be continued from now on.

8.2.1 Adjustments of existing features

Concerning the tutorial video subsection, in the future the application should cache the video. Also, the chapter list should show which of the chapters are playing at that time. The final recommendation for this subsection is to optimize the layout (the chapter/video ratio), especially in landscape mode.

The information subsection should load mobile phone friendly websites. with written instructions. Because of the small applications concerning mobile learning at the time of writing, none of the lab instructors had mobile phone friendly websites ready for us yet. Another recommendation is to add more useful information or maybe hide them when not requested by the user.

In the feedback subsection the uploaded media should be shown inside the application instead of a button that opens the browser. Also a user should be able to edit or delete his own comments.

In the favorites subsection, the user should be able to delete one of his machines out of this list.

An adjustment of the application in general would be to optimize the layout for landscape mode and for tablets. Also the application could be made compatible with older versions of Android (lower than API 14 / Android 4.0).

Adjustments for the front-end administration would be to have a chapter editor where the video is previewed within the browser, so that administrators can immediately create the chapters on the go without leaving the front-end for a second. This increases the workflow and gives a better user experience. In the machine overview the full machine code should be display and a QR code could be shown. This saves a lot of time, because the administrator would then be able to instantly use the NFC Developer application [3] to write the tags.

8.2.2 Missing features

However these features were nor in our requirements nor our formal proposal, we we recommend adding the following features in the future:

Android Application

- Usage statistics
- Social media integration
- Google maps integration
- An internal QR-code scanner
- An internal administrator function for burning NFC-tags
- The ability to rate comments
- OAuth authentication via TU Delft NetID

Back-end/Front-end/API

- Usage statistics
- Independent QR-code generator(s)
- OAuth authentication integration
- Logging functionality
- Map showing all machine locations
- Manage comments: remove (explicit) comments
- Manage users: banish abusive users
- Automatic spam detection system
- Video transcoding queue: update any video format and it will automatically be converted to the right format
- Support for multiple video quality: low, medium, high, etc.
- Caching mechanism in the API to let the Android client application know if data is changed or not. This prevents reloading of non-changed data.

8.2.3 How to make our product a success

We recommend our client to promote our application to all of the lab instructors inside the TU Delft. The application should also be promoted to the potential user. Also we recommend that our client publishes the application to the Google Play Store to make it easier for users to install it. The (read-only!) NFC-tags should be placed on a prominent place onto the machine, and we advice to place a sticker on top of it with the alphanumeric machine-code and/or the corresponding QR-code to make it backwards-compatible with older (non-NFC) phones. The existing tutorial videos should be replaced with more professional recorded videos. Finally, we advice our client to assign students or co-workers to make similar applications for other phone platforms than Android to reach even more potential users.

Bibliography

[1] Karin Clavel. Mobiel leren. URL: http://www.icto.tudelft.nl/projecten/mobiel-leren/.

[2] 123nfc.nl. 123nfc.nl snel nfc tags kopen. URL: http://www.123nfc.nl/.

[3] Thomas Rorvik Skjolberg. Nfc developer app. URL: https://play.google.com/store/apps/details?id=com.antares.nfc.

[4] Adrian Stabiszewski. Ndef editor. URL: http://www.ndefeditor.com.

[5] TU Delft. Corporate colors. URL: https://intranet.tudelft.nl/en/services/communication/communication-mc/manuals/tu-delft-corporate-design/toepassing-huisstijl-2/colour/.

[6] Zend Framework 2. Phprenderer view scripts. URL: http://framework.zend.com/manual/2.2/en/modules/zend.view.php-renderer.scripts.html.

[7] Scott W. Ambler. Feature driven development and agile modeling. URL: http://www.agilemodeling.com/essays/fdd.htm.

Appendix A

Preliminary Report

Preliminary Report
Version 1.0

Dino Hensen Tim Ypeij

May 2013

Contents

Chapter 1

Introduction

The subject of this project is to develop an application for TU Delft's ICT in Education group (ICTO). ICTO wants to gain experience with using smartphones for mobile learning by creating an application which students can use in context, that is: accessing information when and where they need it. An example of where this could come in handy is for many courses that have a laboratory component or require the use of machine workshops. At the moment all instructions are given to large groups of students in a lecture hall without the presence of the machine that is discussed. A solution could be to give more lectures to smaller groups, but then the costs would increase because the instructor has to show up several times instead of just one time. Also more planning needs to be done in order to schedule all groups to get an instruction but this yields more overhead. So an application where students can get their information "in context" might be a good idea, but has yet to be proven in terms of effectiveness and user satisfaction.

ICTO's research objective is to find out if an application as shortly described above can be done effectively and with high user satisfaction within one or more study programs at the TU Delft. In other words, a feasibility study needs to be done to evaluate the potential for success of an application. This brings forth our project: to create mobile software for ICTO to use in a feasibility study.

We will therefore be developing an application that provides a way to gain experience on this topic. This report is about the preliminary research on the application development for the educational application for ICTO and will describe some of the background research we have done on the topic of mobile software development.

Chapter 2

Project Description

For ICTO to be able to perform a feasibility study they need to have mobile software to perform a study on. So our project is narrowed down to create mobile software to get a video instruction of equipment and machines used in the lab courses followed by students that need to know how those machines are operated.

One requirement, that is imposed by our client, is the use of NFC (Near Field Communication), which is a short-range communication technology useful for quick scanning purposes (see chapter 4.4 for more information). Another immediate requirement is that the mobile software is being developed for the Android platform, because it is the largest platform with the most NFC phone's available. BlackBerry and Windows platform also have some NFC phone's, but the market and phones are less popular than Android giving it a disadvantage to be developing an application for at the moment. When the market for these other platforms grow and more phones within their own platform come equipped with NFC, then it will be worth reconsidering to also develop the same application for these platforms, but this is beyond the scope of this project. Also the iOS (iPhone and iPad) platform don't come equipped with NFC at all yet, so this is leaves Android as the best platform to be developing on.

ICTO does not require the Android application to be published to the Google Play Market. Instead a *proof of concept* application suffices for the scope of this project.

2.1 Problem Identification

We identified some problems that occur when an instruction on a particular device is given at the moment:

- Instructions given in one lecture hall to many students at once can cause unintelligibility
- Instructions given without the device nearby can be unclear
- Time-cost (instruction usually are always the same)
- Absent students miss the instruction

In the evaluation we will test to see if the above mentioned problems are solved by the application that will be made.

Chapter 3

Development Methodology

3.1 Project Management

To manage the software design and implementation we will apply our own flavor of agile development. We will use Pivotal Tracker [1] as our project management tool. This provides us with a user interface where we can create a backlog containing tasks that need to be done. we can then create an ordering in iterations, where each iteration equals one week and every task is assigned a number of points.

This way we can see how much points we "burn" every iteration and this gives us an indication of our total progress and might also indicate when we have to work a little harder, or even tell us when we have time left to implement a should/could have feature. This software allows us to work in an agile way prioritizing tasks the way we like it to and shift things around when necessary and still being able to work towards a goal defined at the beginning of an iteration.

We are planning to integrate SVN with Pivotal Tracker by using a post-commit-hook so that we can feed the Task-ID in the SVN commit message which automatically reached Pivotal Tracker and adds data to the task when the commit is being done. This also allows us to change the state of the task through the commit message enabling us to work more efficient.

3.2 Test Driven Development

The requirements document contains use cases that are perfect to implement as test-cases. These test cases can be implemented before and in parallel to the application implementation providing early automated testing to verify the functional behavior of our application [2].

We will use two testing methods: Unit testing and Instrumentation [3]. Both methods are briefly explained in the following sections.

3.2.1 Unit testing

The Android testing API supports JUnit3, so we will be using this feature to test our unit classes. We will write some use cases and translate them into test cases. Then we will write our code until it passes all the unit tests, not looking after efficiency or style. After that, there is time for optimization.

3.2.2 Instrumentation

Android instrumentation allows the control of components independently of it's normal life cycle. Normally the system controls the life cycle of an activity(an Android component representing a single focused thing a user can do) but with the instrumentation you can gain control of the application yourself. You can for instance start the application, then stop it and then start it again to check if some changes persisted. This also allows you to check application variables before the application is even started. It also allows us to mock system objects which are used by the activity under test. Furthermore we can send keystrokes and touch events to the user interface of the activity under test.

Chapter 4

Technologies and Tools

4.1 Development Setup

To develop our android application we will need a development setup which allows us to write code, test it on devices manually (real phones or emulators) and run automated test-cases (JUnit for example). We will develop using Windows as our operating system with Eclipse as our IDE (Integrated Development Environment). Because Eclipse is a Java cross-platform application we could easily switch to another operating system for which Java is available. Because we will be developing an Android App using Java, we need to have the Android SDK in order to build upon the Android framework. Google's Android Developers have created a plug-in for Eclipse that simplifies android programming. The plug-in is logically called the Android Development Tools (ADT) plug-in, which of course requires the Android SDK to be installed.

4.2 Code versioning using SVN

As our code versioning system we choose to use SVN above Git because we only only work in a team of two persons and will be working together much of the time. We both already know our way around with SVN and this will give us a head start. If we would have chosen Git we would have a lot of overhead in the start and in the end maybe not even use all exotic features that Git offers, which makes using it a waste of time for this specific setting.

The Subclipse Eclipse plug-in provides an SVN connector so that we can perform all SVN operations from within Eclipse. This is good because this way we never have to leave Eclipse in order to commit/update which increases our productivity.

4.3 Android devices

As we ourselves are Android users we already own a lot of phones to test on. (See section 4.6 for more information about the devices we use) We will also use the Android Emulator to test the functionality that does not include NFC scanning, because this is not supported by the emulator. For the NFC functionality testing we will use our Android phones.

4.4 Near Field Communication

Near Field Communication is the ability to communicate between devices on a short distance. It is a subset in the High Frequency (HF) band of the RFID standard [4]. The RFID standard is a widely used standard in the world and was originally intended to communicate in one way. NFC however is designed to communicate both ways, at a maximum speed of 424 kb/s and a maximum distance of 10cm.

There are some useful things you can do with the NFC technologies. For example, money transactions with your mobile phone using NFC will be a widely accepted method in the future. Another application is the use of NFC to gain access to certain buildings, for example the office you work at [5].

In our application, NFC is going to be used by two kinds of users. At end-user level, the application will *read* NFC-chips called tags to identify the target machine. At the administrator level, the administrator of the system will *write* information on the tags to be identified by the smart-phones owned by end-users. It is possible to protect the tag from overwriting with certain software, such as NFC Tag Writer [6].

The use of Near Field Communication technologies in Android Application Development is supported from API level 9. This means that only Android 2.3 or higher will support these features [7].

4.4.1 NFC Support

Even though NFC is supported since Android 2.3 (API 10), we will only support Android 4.0 (API 14) or higher because of software maturity. The newer API's are implemented with lot of fundamental changes in comparison to older versions. Because we are developing a prototype in a relatively short amount of time, we are not able to provide support for such a wide range of API's to make the application compatible with devices running older versions of Android. Aside of that, most NFC capable phones already run Android 4.0 or higher, for example all our phones we are going to test our application on (

mentioned in 4.6) do run Android 4.x.x. so developing for lower API versions would also require us to get hold of older development phone's. For a proof of concept application, this is just not feasible to do. New mobile phone's are released with increasing speed which is another reason not to focus on older phone's but instead focus on modern implementation's of NFC that will last a little longer.

4.5 Device Requirements

In this section we provide a small list of device requirements for our Android application being able to run:

- NFC capable Android phone or tablet
- At least API 14, Android 4.0 [8] (see 4.4.1)
- Fast Internet connection for high quality video
- Slower Internet connection for low quality video

For an optimal experience the phone has to be NFC capable. However, we will provide a way for phones that don't have NFC so that they can still benefit from our application by manually starting the application and typing in the machine code themselves.

Because of the widely known instability of the TU Delft Wifi-network called Eduroam network we will provide both high and low quality videos for fast and slow Internet connections. In the event that the wifi network is not working that well, users don't have to stream large high quality videos that affect their phone charges that much.

4.6 Development Devices

We will test our applications on different emulators and phones. The NFC-capable phones that we have access to are:

- Samsung Nexus S, API 16, Android 4.1.x
- Samsung Galaxy Nexus, API 17, Android 4.2.x
- Samsung Galaxy S4, API 17, Android 4.2.x
- HTC One X, API 16, Android 4.1.x

We will also use an emulator in the API range from 14(see 4.4.1) to 17 to test parts of our application that do not require the use of NFC.

Chapter 5

Hosting & Administration

We will be creating an Android application that is able to retrieve data in the form of text and video to instruct the end-user on how to operate a machine. This data has to stored somewhere and needs to be accessible by the Android application. One option would be to store all information about all machines in the Android application that is distributed to the end-user's phones. This would be a very bad solution, because the video files take a lot of storage space which would make the size of the distributable application so big, that it would be a problem to download and store it on a phone. Also it would require us, the developers, to update the Android application each time a new machine is added or info or videos are updated. This is not a feasible solutions and therefore we require the use of a central server following the Client-Server model. We will call this server the *application server*.

In a nutshell the application server needs to be able to provide data about a machine that is scanned by the android application. This data can a video stream or a textual representation information that can help the user to understand the machine that he wants to operate.

5.1 Backend & Web Service

To present data(except for the video stream) to our Android application we will implement a web service. Implementing a web service on a server adds another piece of software to be made during our project. Together with this web service we also need a backend for an administrator to manage the application data. We are using the term *backend* because it is invisible to the end-user, but we could have also called it a *frontend for the administrator*.

5.2 LAMP Environment

Because this assignment is commissioned by ICTO we try to use as much of the available facilities that ICTO has. SSS ICT offers preconfigured LAMP(Linux, Apache, MySQL, PHP) environment, but they can also provide a server with a lot more freedom. We have chosen to use an Ubuntu Linux server that SSS ICT provides for us.

5.2.1 Linux

An Ubuntu Linux server is what we are going to develop on. Ubuntu provides an easy way to install open source software without too much configuration in comparison with other Linux distributions. It is also very stable.

5.2.2 Apache

Apache will be used as our web server. It can be extended by modules to add additional functionality. PHP(see 5.2.4) is one example of such an extended module. Another one is the mod_ssl module, which adds SSL and TLS(see 5.2.5) support for security.

5.2.3 MySQL

MySQL is a database implementation of the Standard Query Language. We will use this database for our model data storage.

5.2.4 PHP

PHP is a scripting language that is especially used for web development. In this language we will create the backend web service.

We need a Linux server because of the many available open source libraries and it's ease of installing new applications. The back-end software will be developed in PHP in combination with a MySQL database to host the data. Video files can be uploaded to a folder and encoded in the right format.

5.2.5 Security

For security HTTPS can be used with a server and client certificate.

For authentication between the android application and backend service we will use Oauth2 authentication to check if a user has the right to access

data provided by the back-end service. The user can log in on the app with his TU Delft Net-ID to post feedback or media.

Chapter 6

Our observations around the campus

We decided to interview students around the campus, on different faculties. Until now, we went to the Industrial Design faculty and the 3ME (Mechanical, Maritime and Material Engineering) faculty. On our trip, we also came in conversation with some instructors and lab coordinators. We included our interview form in Appendix A (although it is Dutch). When we constructed this form, it was meant to be a guide to a conversation with students in particular. At our second location (the PMB lab of IO) we started conversations with instructors and thought these interviews were useful too, but did not have a special interview form for them. So we decided to have a conversation while taking some notes.

6.1 Industrial Design

We first went to a lab in the basement of the Industrial Design faculty. That lab had two simple machines: a drill and a jig saw. We spoke with two second year students there. They told us that in the case of these simple machines you get a live instruction at the beginning of your study. If you were not present at that time, you could try asking one of the lab coordinators to explain how to use these machines, but they usually are not happy to do that. These two students thought our application idea was very useful for this kind of machines. We asked about some more complicated lab equipment and they directed us to the PMB lab of the same faculty.

Arrived at this lab we first spoke to two fifth year students. They supported our idea, however they had very little time for us because they were in a hurry. They seem to skip a lot of instruction sessions, so our application

offers a quite handy solution for them. These students thought the cutter was the hardest to use here and the 3D-printer the easiest. They also gave us a tip about dividing the video in chapters and the ability to check the videos at home. They only would use the feedback system to show their end product, not to give tips to other students.

After this conversation we started a dialog with one of the instructors. He seemed a little bit afraid that he would become superfluous, but he would like the fact that not everybody would ask him about basic knowledge stuff all the time. He promised us that he would corporate with us to make a video for our application. However, he told us that for some lab equipment, a live instruction is a prerequisite before students are allowed to work with this equipment. This measure is for safety reasons, so a video cannot replace this instruction for this reason. We also spoke to the lab coordinator and he confirmed this. He only likes our idea in the case of equipment that don't need safety instructions, like the drill, jig saw and 3D printer.

As we stepped outside the faculty, we concluded that we need to focus on the so called safer to use machines. The potentially dangerous equipment need a live instruction anyway for safety reasons, so our application will be a bit superfluous in that case.

6.2 3ME

When we arrived at the 3ME faculty, we first stepped inside the IWS (Inloop Werkplaats Studenten, walk in workshop for students in Dutch). We decided to talk to the coordinator here as well, as the students were very busy here. He was very happy, as he claimed that he came up with our idea (with QR instead of NFC) as well a few years ago. He already administers a wiki/pdf page with instructions and pictures, but not specified for mobile application. He also had made some instruction videos, but they aren't posted on-line yet. We exchanged our contact information and went further to the next lab.

The next lab we visited was the welding hall. Here we spoke to two lab instructors. They rejected our idea because of safety reasons. During the practical sessions, you always have to wear special goggles, and through that goggles you cannot check your phone at all. To unequip these goggles during the welding sessions is also not a good idea because other people can be welding at the same time in the same hall and this will damage your eyes. Your phone would not be safe either, with the sparks flying around sometimes. They also told that many welding skills are much more a feeling thing rather than a theoretical thing, and this is hard to show in a video. However they were not interested in our idea, they invited us to observe a

practical session when we want to, so we can see for ourselves our application will be quite useless in this setting.

We concluded that regarding this faculty, we will limit ourselves to the IWS lab. We will not use our application for the welding hall, because of the obvious security reasons. We will contact the IWS coordinator soon, because he has already some videos for us.

Chapter 7

Conclusion

We already feel that there is a lot involved in our bachelor project. We have chosen to work with software and programming languages we are familiar with. Furthermore we want to apply all the relevant knowledge we gained during our study, especially the software engineering and testing part. It is interesting to involve NFC technologies in our mobile application because they have a promising future but nowadays we almost never use them. We did some research about the requirements of both the Android application and the server for hosting the back-end software. We are going deeper into this in our following requirements deliverable. Finally we did some research around the campus of how people involved with lab equipment think about our idea in practice. Most of them were positive about our idea and therefore we are very motivated to continue with our project.

Bibliography

[1] Pivotal Labs. Pivotal tracker features. URL: http://www.pivotaltracker.com/features.

[2] M.Young M.Pezze. *Software testing and analysis*. Wiley, 2008.

[3] Google. Android testing fundamentals. URL: http://developer.android.com/tools/testing/testing_android.html.

[4] Roy Want. Near field communication. *Pervasive Computing, IEEE*, 10(3):4 – 7, 2011.

[5] Kevin Curran et al. Near field communication. *International Journal of Electrical and Computer Engineering (IJECE)*, 2(3):371 – 382, 2012.

[6] NXP Semiconductors. Nfc tagwriter by nxp. URL: https://play.google.com/store/apps/details?id=com.nxp.nfc.tagwriter&hl=nl.

[7] Google. Api android.nfc. URL: http://developer.android.com/reference/android/nfc/package-summary.html.

[8] Google. Requesting nfc access in the android manifest. URL: http://developer.android.com/guide/topics/connectivity/nfc/nfc.html#manifest.

Appendix A: Interview form

Studierichting en studiejaar:

1. Heb je weleens een apparaat gebruikt?

   Ja

   Ga door naar vraag 2

   Nee

   Ga door naar vraag 5

2. Hoeveel verschillende apparaten heb je gebruikt?

3. Welk apparaat was het moeilijkst te bedienen?

   Apparaatnaam

   Hoe beoordeel je de begeleiding/instructies/handleiding?

   Welke manier werden de instructies gegeven?

4. Welk apparaat was het makkelijkst te bedienen?

   Apparaatnaam

   Hoe beoordeel je de begeleiding/instructies/handleiding?

   Welke manier werden de instructies gegeven?

5. Wat zou je ervan vinden om via een app ter plekke een instructievideo te kunnen zien?

6. Zou je dit prefereren boven een instructie in college?

7. Zou je via de app feedback achterlaten voor anderen in de vorm van video, foto of geschreven tekst?

8. Wat kwam er nog meer te sprake?

Appendix B

Project plan

Project plan
version 1.0

Dino Hensen Tim Ypeij

April 2013

Preface

This is the first step in our last phase of the Bachelor study in Computer Science. We are Dino Hensen and Tim Ypeij and we are going to develop an mobile application for the TU Delft. In the following months we are doing this traineeship in supervision of our company supervisor Karin Clavel and faculty supervisors Koen Bertels and Andrew Nelson (PhD student). Therefore we want to thank them for giving us this opportunity.

Chapter 1

Introduction

Applications for mobile phones, or in short apps, are a booming business nowadays. The Android platform is one of the biggest platforms available on the newer generation smartphones. Android is based on the programming language Java, a language we have great experience in. While we both have an Android phone ourselves, we already were interested in developing an app before this project.

As our last phase of our Bachelor study in Computer Science we had to choose a project from a list or approach a company ourselves. The project we chose was already on that list and the company is the ICT department of the TU Delft Library. It got our attention because it was about developing an application for Android smartphones.

The assignment is to develop an application for students so they can use their smartphone for mobile learning. The application has to support lab courses by showing tutorials at the right place and time for devices that one has to get instructions about before being able to operate the device. We want to use Near Field Communication to determine the right place. The student scans a chip on some machine to use in his or her lab cours even when he feels he could need some help (the right time). Then the student can view an instruction video, user pictures, user videos and user feedback. He also has to be able to leave some feedback as well as upload pictures or videos. We included the exact proposal in this document (Appendix A).

In this project plan, we describe our project assignment and project establishment. We also include how we are going to protect the quality.

Chapter 2

Project assignment

2.1 Project environment

The TU Delft ICT in Education Group (ICTO) are constantly doing research in how to support the education role of the TU Delft with ICT solutions. They think that the current way of giving instructions of lab classes in front of hundreds of students during lectures is a bit outdated. They want to investigate if the current trend of smartphones comes in handdy.

2.2 Objectives of the project

The ICTO want us to develop a prototype of an application for the TU Delft where students can scan a chip on lab equipment with their mobile phone equiped with NFC technology. Then the student can choose to see an instruction video or read/leave feedback for this piece of lab equipment.

2.3 Project description

The way we want to accomplish the objectives of our assignment is to apply many of the knowledge we gained during our Bachelor study(See 4.2 for more information about the knowledge domains we plan to use). We will be working in a flexible work environment which means that we can sit in a different room everyday meeting new people. This makes more knowledge and information from coworkers available for us. We plan to be at the office of the ICTO during 3 days of the week. The rest of the time we work at home or at other TU Delft facilities. We will meet with our company supervisor every Wednesday afternoon. Our faculty supervisor is available for us at Mondays, Wednesdays and Thursdays.

2.4 Deliverables

The delivered end product will be an mobile (Android) application for students who are the end-user. With this application the student can quickly get information about the operation of a complicated device in one of the many TU Delft practical rooms or labs. Aside of the implementation, we have to deliver a number of reports. First of all, this is the first report and it is named the project plan. The next report will be a research report of our findings in the orientation phase. Parallel to the implementation we will work on the final report. One important section in the final report will be about the requirements. Because a Requirement Document can be seen as a contract between the developer and client, therefore we also deliver a requirements document. We will offer this section to our company for review soon after the orientation phase. After all the deliverables are finished, we will give an end presentation.

2.5 Requirements and constraints

The requirements are a working prototype of the application on an Android phone. The application has to be stable and able to play video. Also the end-user has to be able to leave comments and feedback and add their own videos or pictures. One of our big constraints are that only new Android phones are capable of NFC communication. Maybe in the future if our prototype is a success, the TU Delft wants to develop this application for other phones (iPhone, Windows, Blackberry, older Android phones), for example with QR-tags.

Chapter 3

Planning

Table 3.1: Planning

Weeknr.	Date	Hours a week	Hours total	Deliverables
Week 1	22-04-2013	32	32	Project Plan
Week 2	29-04-2013	32	64	
Week 3	6-05-2013	32	96	Research report
Week 4	13-05-2013	32	128	
Week 5	20-05-2013	32	160	
Week 6	27-05-2013	32	192	
Week 7	3-06-2013	32	224	
Week 8	10-06-2013	32	256	
Week 9	17-06-2013	20	276	
Week 10	24-06-2013	0	276	
Week 11	1-07-2013	16	292	First SIG code submit
Week 12	8-07-2013	32	324	
Week 13	15-07-2013	32	356	
Week 14	22-07-2013	32	388	
Week 15	29-07-2013	32	420	Final Report & Code submit
Week 16	5-08-2013	additional	420	Presentation
Week 17	12-08-2013	additional	420	

In table 3.1 we outlined a global planning with soft deadlines of various deliverables. We chose to plan 2 additional weeks because we still have subjects to pass this quarter. Our planning is to finish this project in 420 hours each

person according to 15 ECTS. We will be present at the TU Delft Library office for at least 3 working days a week. The rest of the time we will work at home or at other facilities of the TU Delft.

Chapter 4

Project establishment

4.1 Organization

The work that is needed to complete the project is equally devided between us. We will both design and implement and write the reports. We will meet weekly with our supervisors, just as specified in section 2.3.

4.2 Staff

The requirements on the staff (on us) are that we are proficient to develop an app. Not only the implementation has to be stable, also the idea behind the design has to be good. We both think that we have gained enough experience to complete our job during the previous projects in the university, various subjects we followed and experience gained during jobs. In this project we are planning to use knowledge gained about the following domains:

- Software Engineering
- Test Driver Development
- Database Design (SQL)
- Java programming

Also it is required to spend enough time. We plan to spend 32 hours a week (average) except during our exams. For our planning see the previous chapter.

4.3 Administrative procedures

To monitor the project we will use Pivotal tracker, which is project management software providing the tools to collaborate and communicate with all parties involved in the project. This software is designed for agile development, a

method of software designing we want to apply in this project. Pivotal tracker also provides a file sharing feature which can be integrated with Google Drive. We will use this feature for easy file sharing with team members. For the implementation we will use SVN as our version control system.

4.4 Reporting

We will write the reports using the LaTeX format, so the reports will be in an acedemic report style. We will use the BibTeX format to manage our references. We will send all the reports on time to our direct supervisors so they can review our work.

4.5 Resources

The hardware resources we will use are our laptops and NFC capable phones to test our application on. Also we will have to buy some NFC tags to test and place on the lab equipment when the prototype is ready. For the software part we will use Java as our implementation language with some SDKs (Software Development Kits) of which the Android SDK is the most important. As our IDE (Integrated Development Environment) we will use Eclipse with the ADT plugin(Android Development Tools), which provides an easy way to create virtual android devices and connects to external android devices for debugging and testing purposes. Finally if we need a database we will use the SQL format.

Chapter 5

Quality

We want to secure the quality of our end project by making strict agreements with eachother and to the supervisors from week to week. We will control our versions and collaboration with SVN and Pivotal tracker, so we can work together separately on the same project without corruption or loss of data. Also we want to apply the knowledge and experiences we learned in past projects and subjects, especially Software Engineering and Software Testing. We want to develop by a methodology called Test Driven Development (TDD) as much as possible, because we can base our tests on the use cases and requirements which we will gather in the requirements elicitation phase. This means that we first will write scenarios and implement complementary testcases, before we start implementing the functional code. Then we will adapt our code just as good until our code passes all our tests. Finally, we are going to test our prototype on various real phones and emulators to explore several bugs and inefficiencies. Next to TDD we would like to use an agile software development method which is supported in Pivotal tracker. We want to take some elements of a method called Scrum, essentially creating our own flavour of Scrum so that is suites our needs without having too much overhead because Scrum has roles for more then two persons.

Appendix C

Project Proposal

Project description

Smart phones and tablets are more and more part of our daily life. More than 80% of TU Delft's students use a smart phone [1]. TU Delft's ICT in Education group (ICTO) wants to gain experience with using smart phones for mobile learning. In some cases mobile learning can be more effective than traditional learning, for example when learning in context. This means that you can access information when and where you need it.

At TU Delft, many courses have a laboratory component or require the use of machine workshops. Instructions for use and safety are often given in a lecture instead of in close proximity (place & time) to the equipment and machines. What if we could use our smart phones to scan the equipment, watch an instructional video, click through the steps one by one and learn from tips & tricks by others students? What if we could upload and share pictures or a video from our experiment or the prototype we made?

This project is about finding out if this can be done effectively and with high user satisfaction within one or more study programmes at TU Delft. The end product should be a demonstrator Android app (it doesn't have to be deployed in an app store) that can be used as a proof of concept. It should at least contain the following functionality: scan, launch machine/equipment information (including an instructional video), sign in (with an existing account) and add a comment, picture or user video. The demonstrator should work for at least one machine and one laboratory setting of choice, preferably from different faculties or study programmes.

Auxiliary Information

This project will be carried out within the context of the project 'Mobile' within the University Corporate Office Education & Student Affairs (O&S). The aim of this project is to gain, combine and share expertise in mobile learning. This project is coordinated by Karin Clavel of TU Delft Library. Students are eligible for a placement fee.

[1] http://www.e-learn.nl/2011/09/09/results-of-tu-delft-mobile-survey-2011

Project team

Supervision

Karin Clavel, c.l.clavel@tudelft.nl , 015-2787631
Koen Bertels, k.l.m.bertels@tudelft.nl, 015-2781632
Andrew Nelson, a.t.nelson@tudelft.nl, 015-2783644

Student team

Tim Ypeij, 1386174, tim.ypeij@gmail.com, 06-23063275
Dino Hensen, 1367412, dino.hensen@gmail.com, 06-24730860

Appendix D

Code Evalutation SIG

First review

18-07-2013

De code van het systeem scoort 4 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere score voor Unit Size en Duplication.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Binnen de langere methodes in dit systeem, zoals bijvoorbeeld de 'attachOnTouchListener'-methode in de 'FeedbackAdapter'-class (of the functie 'authenticateAction' in de 'ApiAuthenticationController'), zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes. Commentaarregels zoals bijvoorbeeld '//change color' en '//set color back and change rating' zijn een goede indicatie dat er een autonoom stuk functionaliteit te ontdekken is. Het is aan te raden kritisch te kijken naar de langere methodes binnen dit systeem en deze waar mogelijk op te splitsen.

Voor Duplicatie wordt er gekeken naar het percentage van de code welke redundant is, oftewel de code die meerdere keren in het systeem voorkomt en in principe verwijderd zou kunnen worden. Vanuit het oogpunt van onderhoudbaarheid is het wenselijk om een laag percentage redundantie te hebben omdat aanpassingen aan deze stukken code doorgaans op meerdere plaatsen moet gebeuren. In dit systeem is er duplicatie te vinden in bijvoorbeeld de verschillende 'post*' methoden binnen de 'HttpPoster'. Hier zou bijvoorbeeld het afhandelen van de response-code in een aparte functie gestopt kunnen worden zodat deze functionaliteit hergebruikt kan worden. Het is aan te raden dit type duplicatie goed in de gaten te houden en dit waar mogelijk op te ruimen.

Wat verder opvalt is dat de code vrij veel 'TODO'-commentaren bevat, 47 commentaren verspreid over 23 bestanden. Door issues in de code te documenteren is het lastiger om het overzicht op het aantal op te lossen issues te behouden. Het is aan te raden deze issues te documenteren in (bijvoorbeeld) een issue-tracker om dit overzicht te bewaken.

Over het algemeen scoort de code bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase. Als laatste nog de opmerking dat er geen (unit)test-code is gevonden in de code-upload. Het is sterk aan te raden om in ieder geval voor de belangrijkste delen van de functionaliteit automatische tests gedefinieerd te hebben om ervoor te zorgen dat eventuele aanpassingen niet voor ongewenst gedrag zorgen.

Second review

In de tweede upload zien we dat het codevolume in omvang is verdubbeld. Dit komt voornamelijk door uitbreiding van de PHP code, de Android app is slechts licht gegroeid. De score voor onderhoudbaarheid is ongeveer gelijk gebleven.

Bij Unit Size zien we dat een aantal langere methodes inmiddels is opgeknipt. De meeste moeilijk leesbare methodes uit de eerste upload, zoals FeedbackAdapter.attachOnTouchListener met een switch-in-een-anonymous-inner-class, zijn hierdoor een stuk overzichtelijker geworden. Bij duplicatie zien we dat de suggesties uit de eerste analyse zijn opgevolgd, maar omdat er op andere plaatsen weer wat duplicatie is toegevoegd is de deelscore er niet op vooruit gegaan.

Het is positief dat er inmiddels testcode is geschreven, al geldt dit vooralsnog alleen voor de Java-code en niet voor PHP. Ondanks dat unit testen voor PHP (nog) niet zo ingeburgerd is, heeft het alsnog toegevoegde waarde om ook die code te testen.

Tot slot is het goed om te zien dat het aantal TODOs in de code sterk is gedaald, van 47 in de vorige upload tot 2 nu.

Uit deze observaties kunnen we concluderen dat de aanbevelingen van de vorige evaluatie zijn meegenomen in het ontwikkeltraject.

PHP Test code not detected

SIG's analysis tool did not detect our PHP test cases:

> "Onze analyse-tooling heeft de PHP-testcode niet automatisch herkend"

This fact did not influence the code maintainability score of 4 out of 5 stars.

Appendix E

Usability report

Usability Review Scansistant

24 July 2013, Karin Clavel

Heuristic evaluation

The review was done based on the ten heuristics recommended by Nielsen (1995). Ideally, 3-5 different evaluators carry out this kind of review. In this case, only one evaluator was available.

Back-end

The back-end is used to add machines and videos to the app. It is available from http://scansistant.nl. An administrator login was provided to carry out the test.

1. Show system status

The system status was visible and clear in most screens of the back-end. The menu always shows in which section the user is and a breadcrumb path supports this for underlying pages/screens. A progress bar was visible when uploading videos.
Problems / opportunities for improvement found:

- Disable the Add button when uploading a video.
- The progress bar during upload is somewhat hidden.

2. Familiar metaphors and language

This can be improved, as sometimes the terms used are not explained or self-explanatory.

- Content: when adding a machine, the field content is too abstract. It is not clear what should be entered here. In the examples a URL was used. A better name for this field would be "URL" or "URL or Extra Information" if it is also possible to type in information.
- Building: this field is used in both Location and Department. It is in fact possible for a machine to belong to a department, which has a different location than the machine, but the field seems to be used in a different way, in one case with a code (LI, TUD) and in the other case as a name (Library office). For TU Delft, buildings are most often referred to by Building number. This would be more familiar to users.

3. User freedom & control

The back-end offers enough freedom to explore, without users having to worry about getting lost or damaging something. For all options, cancel buttons are available.

4. Consistency & standards

The back-end seems consistent. Links are blue, except in the menu, which makes it easy to identify the clickable text. The different forms all work the same.
Consistency issue:

- When choosing to edit a video, the file field says: "No file chosen". This seems strange, because in the file was uploaded earlier and the path is visible in the table where all the videos are listed.

5. Error prevention

Fields are checked when saved. When deleting something, a warning is issued with a clear dialogue to prevent data loss from clicking the wrong button.
Improvements:

- All fields in the various forms seem to be mandatory; this could be made visible somehow.

- It is not clear which video format is expected. The open file dialogue could specify which file types are allowed.
- Maybe the system could also specify other criteria for the video's, such as maximum size or minimum resolution.
- The machine code does not appear to be unique, only in combination with Department and Location. This does not seem to be enforced by the system.
- The format for starting time of a chapter is not specified.

6. Recognition over recall

Overall, users don't have to remember data from one screen to the next. Earlier data entries are visible and selectable in dropdown boxes. Some exceptions:

- Add machine -> Code: there is no way to see which codes are already in use, except for remembering the code.
- When adding chapters to a video, the number is presented. This should be the (user-entered) name instead. The number does not trigger any recognition as the name would.
- Building: see also Consistency & Standards. When you want to enter the same Building in both Department and Location, you have to remember exactly how you entered it the first time (or copy/paste).

7. Flexibility & efficiency.

The flow of the back-end could be designed better, as some forms depend on other forms to be filled in first.

- When entering a machine, it would be nice to have "Add location" and "Add department" and "Upload video" as side steps.
- In a next version, it would be nice to add a feature in to add machines by uploading videos in batches and information by xml or CVS files for expert users.

8. Aesthetic & minimalist design

No complaints here! The design is plain and functional. No superfluous elements or decorations.

9. Help users recognize, diagnose, and recover from errors

See also section 5.

- When adding chapters to a video, a start time is required. The format for this is not specified. The error message could suggest a solution.
- A video preview would be nice; to be able to check whether the video works and the chapters are as intended.
- When deleting a machine that is used in a favourites list, a system error occurs.
- When deleting a Department or Location that is used in Machine, the system updates the fields in Machine without notifying the user.

10. Help, documentation

Not present. Could be added in a future version.

Scansistant App

1. Show system status

The system status is generally visible. After scanning and logging in, a progress bar & counter is visible. A welcome message is present when signed in and an invitation to login is present when not signed in. When in a subsection, the top bar has a link to the machines with a "back"-arrow.

Problems with system status:

- The Tutorial Video section does not show which video is currently active or playing.
- Also use the top bar with a link back and "back"-arrow in the Login and Sign up screens.
- When making a mistake logging in, the error message is hidden in the keyboard.

2. Familiar metaphors and language

Familiar icons are used. The style of these icons is well adapted to the graphical design, but the meaning is still clear. There is very little text in the app. The terms used are easy to understand for the intended audience.

3. User freedom & control

Inherent to the app's functionality, the user is limited in exploring the app by encountering the machines or machine codes. Users do not have to log in to use the app for the instruction videos. Logging in only provides – useful – extra functionality for which the user's identity is needed.

Improvement:

- The Log out is only available from the home screen.
- In a future version, it should be possible to delete favourites and comments.
- The landscape view works, but should be optimised.

4. Consistency & standards

The app works according to Android standards. For example: links to outside content open in a browser when using tap-hold and video controls appear when tapping the video. The intention was to use the NetIQ to sign in. This would make it consistent with other TU Delft applications. Unfortunately, TU Delft's ICT department did not have the necessary client software up and running at the time of implementation of the scansistant app. In a future version it should be possible to use a NetID login.

- More consistent use of terminology: Sign in / Sign up / Sign out
  or Login / Register / Log out

5. Error prevention

When trying to add a duplicate to favourites, the app responds with a clear message. In a future version, it should be possible to remove items from favourites and remove comments.

6. Recognition over recall

The user does not have to remember anything. The last used code is remembered on the home screen. The login information is stored for future use. The favourites list also supports recall. Adding a thumbnail image to the favourite list would be nice. Icons also have a text.

7. Flexibility & efficiency.

The app is flexible and efficient for the intended use.

8. Aesthetic & minimalist design

The app is visually pleasing in TU Delft blue. The design is minimalist with no unnecessary clutter.

9. Help users recognize, diagnose, and recover from errors

No problems or issues found.

10. Help, documentation

No help or documentation is currently available.

References

Nielsen, J. (1995, January 1). *10 heuristics for user interface design*. Retrieved from http://www.nngroup.com/articles/ten-usability-heuristics/