



NTNU – Trondheim
Norwegian University of
Science and Technology

Using open vs. proprietary standards when developing applications for mobile devices

Jon Freberg

Master of Science in Computer Science

Submission date: May 2012

Supervisor: Terje Rydland, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

This thesis discusses the possibilities and limitations when developing mobile applications natively vs. HTML5. Research has been carried out to understand how mobile devices can be utilized to its fullest when running applications, and a native iPad application was developed to help in discovering unforeseen challenges. It was developed with a client-server architecture to decrease the amount of work it would take to implement the application natively as a client for all the different mobile platforms.

The native approach combined with the client-server architecture is not cross platform by nature, but the study shows that its benefits outweighs the limitations in many cases. It is also shown that time- and cost of the development of an application favours the HTML5 approach. HTML5 was concluded to be a solution that solves the cross platform problem, but it lacks both performance and API access. However, the type of application developed should be the deciding factor of what solution to choose.

Keywords: Cross platform mobile applications, HTML5 applications, Native applications, Objective-C, iPad

Sammendrag

Denne oppgaven drøfter muligheter og begrensninger ved utvikling av native mobile applikasjoner kontra HTML5. Videre har det blitt undersøkt hvordan mobile enheter kan utnyttes best mulig når disse kjører. En native iPad applikasjon er utviklet for å være bedre rustet til å oppdage uforutsette utfordringer. iPad applikasjonen ble utviklet med en klient-server arkitektur for å redusere implementeringsmengden nødvendig for å utvikle applikasjonen til flere mobile plattformer.

Den native fremgangsmåten kombinert med klient-server arkitektur vil ikke være kryss-plattform av natur, men i mange tilfeller er mulighetene og fordelene av hva løsningen gir tyngre vektet enn begrensningene. Med tanke på tid og kostnader ved utvikling av en applikasjon, er det nesten alltid billigere å utvikle i HTML5. HTML5 ble konkludert med å være en fremgangsmåte som løser kryss-plattform problemet på en god måte, men den har redusert ytelse og API tilgang. I et hvert utviklingsprosjekt av mobile applikasjoner er det applikasjonens oppgave og funksjonalitet som bør være den avgjørende faktor for hvilken av de to utviklingsmetodene som burde brukes.

Stikkord: Kryss-plattform mobile applikasjoner, HTML5 applikasjoner, Native applikasjoner, Objective-C, iPad

Preface

This work is carried out as the master thesis in my Master of Technology degree at the Norwegian University of Science and Technology. Personal interest in the field of mobile application development brought my advisor Terje Rydland from the Department of Computer and Information Science and me together. I would like to thank him for the guidance throughout the project.

I also owe my sister Hanne a great thanks for proofreading the report so that the final result became even better. A big thanks also goes out to my friends, in particular my roommates.

Last but not least, I want to thank my family, especially mom and dad, for all the support and help you have given me throughout 19 years of school. Your encouragement and contributions are very much appreciated.

Trondheim, May 31st, 2012 - Jon Freberg

Contents

1	Introduction	1
1.1	Motivation	2
1.2	Research Question	2
1.3	Project Goal	2
2	Background Research	5
2.1	HTML5 applications	5
2.2	Native iPad applications	7
2.3	Native vs. HTML5	8
2.3.1	Multithreading vs. web workers	8
2.3.2	Application runtime environments	9
2.4	Application design	9
	Challenges with the client-server model	11
2.5	Interviewing software developers	12
3	From Research to Prototype	15
3.1	Wanted functionality	16
3.2	Technical Research	17
3.2.1	The iPad application	17
3.2.2	The web server	19
3.2.3	Web services	19
	RESTful architecture	20
	JSON- JavaScript Object Notation	21
4	Education+ Overview	23
4.1	Education+	23
4.1.1	Education+ - The web server	24
4.1.2	Education+ - The administrator panel	24
4.1.3	Education+ - The iPad application	25
5	Implementation - The web server	27
5.1	The Spring Framework	27
5.1.1	Model view controller	28
5.1.2	Dependency Injection	31

5.1.3	Java Persistence API	31
5.1.4	Security	31
5.2	Glassfish vs. Tomcat	32
5.3	Architecture	33
5.3.1	Model classes	36
5.3.2	Data Access Object classes	39
5.3.3	Service classes	41
5.3.4	Controller classes	42
5.3.5	Spring configuration	44
	Web.xml	44
	servlet-context.xml	46
	Security-context.xml	47
	Persistence.xml	49
5.3.6	WEB-INF/views	49
6	The Concepts of Programming in Objective-C	53
6.1	The four layers in iOS	53
6.1.1	Core OS	54
6.1.2	Core Services	54
6.1.3	Media	54
6.1.4	Cocoa Touch	55
6.2	Design Strategies	55
6.3	Syntax	59
6.3.1	Header and Message files	59
6.3.2	Protocols and delegates	63
7	Implementation - The iPad application	65
7.1	Architecture	65
7.1.1	The Views	67
	1 - The initial view controller	67
	2 - A view	70
	3 - Setting the views class	70
	4 - The object palette	71
	5 - Segues	71
	6 - The view controller scenes	72
7.1.2	The Controllers	72
7.1.3	The Models	73
7.2	Handling threads	76
7.2.1	Multithreading and blocks	76
7.2.2	Core Data	78
8	Results	81
8.1	Testing	81
8.1.1	Timing of web service calls	81
8.2	Challenges	84
8.2.1	The web server	84

Configuring the Spring framework	84
Design bugs	85
8.2.2 The iPad application	86
Getting started	86
Going outside Apples design guidelines	87
8.3 New features	88
8.3.1 Messaging system	88
8.3.2 Quiz system	89
8.3.3 Sharing of documents	89
8.4 Improvements	89
8.4.1 Push Notification Center	90
8.4.2 Asynchronous web service calls	90
8.4.3 iPhone compatible	91
8.4.4 General improvements	91
9 Conclusion	93
9.1 Further work	94
APPENDIXES	100
A Education+ user manual	101
A.1 Adminstrator panel	101
A.2 iPad application	102
B Interview results	105
B.1 Interviewee 1	105
B.2 Interviewee 2	108
B.3 Interviewee 3	111
B.4 Interviewee 4	114
C General information	117
C.1 Tools and IDE's used	117
C.2 Creating admin and school objects	118

List of Figures

2.1	Coverage ratio of the 3G and EDGE networks in Norway[1]	6
2.2	The Model View Controller Design Pattern	10
2.3	Client-Server architectural pattern	11
3.1	iTunes U - CS193P course overview	18
3.2	iTunes U - Screenshot from a lecture video	18
3.3	XML vs. JSON - Parsed twitter search results[2]	22
4.1	The Education+ administrator panel	24
4.2	The Education+ iPad application	25
4.3	Education+ course and news overview	26
5.1	Spring Framework Overview	28
5.2	The Model View Controller Architectural Pattern	29
5.3	Request flow in the Spring MVC Framework - Administrator Panel	29
5.4	Request flow in the Spring MVC Framework - Web Services	30
5.5	Education+ web server architecture	33
5.6	Education+ structure	34
5.7	Education+ Web Server Class Diagram	35
6.1	The four layers of iOS	54
6.2	MVC - iOS specific part 1	56
6.3	An Objective-C outlet	57
6.4	MVC - iOS specific part 2	57
6.5	Example delegation between objects	58
6.6	Some methods implemented by the UISearchBar protocol	64
7.1	The Xcode storyboard	67
7.2	Education+ with swipe-enabled menu	70
7.3	Different objects from the object palette	71
7.4	A cutout from the storyboard where a segue is created	72
7.5	Core Data data model user interface	74
7.6	Graph view of the Education+ entities	74
7.7	Persistence Store Stack	78

8.1	Chrome Inspector showing the web service call <code>/getUsersInCours/1</code>	82
8.2	All dependencies needed for the web server	84
8.3	The .NET Framework Stack	85
8.4	News overview problem	87
8.5	Increase communication with the Jabber chat client	88
8.6	Notification badge on the application icon	90
A.1	The Education+ icon	102
C.1	HTTP client web service call	119

List of Tables

2.1	3G coverage worldwide	7
2.2	The MVC design pattern[3]	10
3.1	Web server functionality	16
3.2	iPad application functionality	17
3.3	Guiding principles of the REST uniform interface constraint[4]	20
3.4	CRUD functions	21
4.1	Overview of Education+ administrator panel	25
5.1	Model objects located in no.ntnu.jf.model package	36
5.2	Database entity creation annotations	38
5.3	Model objects located in no.ntnu.jf.dao package	39
5.4	Data access annotations	40
5.5	Model objects located in no.ntnu.jf.service package	41
5.6	Service layer annotations	42
5.7	Model objects located in no.ntnu.jf.controller package	42
5.8	Service layer annotations	44
6.1	MVC Architectural Pattern - iOS specific	55
6.2	Property declaration attributes - setter semantics	60
6.3	Property declaration attributes - writability	61
6.4	Property declaration attributes - atomicity	61
7.1	Folder contents overview	66
7.2	The different view controller options [5]	69
8.1	Timing of web service calls in milliseconds	83
8.2	General improvements	91
9.1	Possibilities and limitations when developing mobile applications	93
A.1	Education+ Administrator Panel information	101
A.2	List of available administrators	102
A.3	List of available student	103

C.1 Web server tools 117
C.2 iPad tools 117

Listings

3.1	Security annotated Java method	19
5.1	Snippet of no.ntnu.jf.model.Course.java object implementation . . .	37
5.2	Snippet of no.ntnu.jf.dao.CourseDAOImpl.java object implementation	39
5.3	Snippet of no.ntnu.jf.service.CourseServiceImpl.java object imple- mentation	41
5.4	Snippet of no.ntnu.jf.controller.CourseController.java object imple- mentation	43
5.5	web.xml file	45
5.6	servlet-context.xml file	46
5.7	security-context.xml	47
5.8	method from no.ntnu.jf.service.SpringLoginService.java class	48
5.9	persistence.xml	49
5.10	ModelAndView method used to delegate data to the views	49
5.11	Example HTML code to access model objects	50
5.12	Snippet of getJSON JQuery method	50
6.1	Dummy implementation of an Objective-C header file	60
6.2	Dummy implementation of an Objective-C message file	62
6.3	Example creation of Java object	62
6.4	Example creation of Objective-C object	62
6.5	Example Java method	63
6.6	Example Objective-C method	63
6.7	SearchViewController.h source file	63
6.8	Example implementation of an optional protocol method	64
7.1	LoginViewController.h - Header file	72
7.2	Example IBAction method implementation	73
7.3	User+Create.m category implementation	75
7.4	Stripped down version of LoginButtonAction method	76
7.5	LoginButtonAction method with thread	77
7.6	LoginButtonAction method with main thread	77
8.1	Entity description of the text instance variable	85
8.2	Entity description annotations used for the MySQL text data type .	86
C.1	Example JSON objects	118

Abbreviations

3G	3rd Generation mobile network
AOP	Aspect Oriented Programming
API	Application Programming Interface
ARC	Automatic Reference Counting
DAO	Data Access Objects
DoS	Denial of Service attacks
EDGE	Enhanced Data rates for GSM Evolution
GCD	Grand Central Dispatcher
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
IDE	Integrated Development Environment
IRC	Internet Relay Chat
Java EE	Java Enterprise Edition
JDBC	Java DataBase Connectivity
JPQL	Java Persistence Query Language
JSON	JavaScript Object Notation
MB	Megabyte
MVC	Model View Controller
NTNU	Norwegian University of Science and Technology
ORM	Object Relation Mapper

OS	Operating System
POJO	Plain Old Java Object
REST	Representational State Transfer
SHA-1	Secure Hash Algorithm 1
SQL	Structured Query Language
UI	User Interface
W3C	World-Wide Web Consortium
WAR	Web application ARchive
XML	eXtensible Markup Language

Chapter 1

Introduction

Being able to develop cross platform software applications has always been an important undertaking for developers. Given the frameworks that has been available, the task hasn't always been that easy. When Java was introduced in 1995 a solution to the problem had finally arrived. Java solved this problem by compiling the source code to an intermediate representation called Java byte code instead of directly to platform-specific machine code, which made any operating system being able to interpret a Java application. Software development companies in particular, were very satisfied with the Java solution due to a drastic decrease in both implementation cost and time in cross platform development projects. In recent years, smart phones and tablets has become very popular. With these "post-pc" devices, almost any application or data can be accessed at any given time or place, which makes them very useful for both private use, as well as in businesses. These new devices are used interchangeably together with both desktop and laptop computers, which makes it desirable that any application should work on any of these devices. To achieve this, Java does not do the job anymore. The different smart phones and tablets use different operating systems that are customized specifically to run on these devices so that the limited resources available can be utilized to its fullest. The different mobile operating systems that are used today are Google's Android, Apple's iOS, Microsoft's Windows Phone, Nokia's Symbian and RIM's BlackBerry. As of February 2012, there were approximately 300 million activated Android devices[6] and approximately 318 million activated iOS devices[7]. The user mass of the post-pc devices are enormous, which makes it important that any application that is being developed can be deployed to as many platforms as possible. All these operating systems are based off of different kernels, which means that there are no cross platform development solution available that makes an application run natively on every device, like Java does for PC applications. A solution to make an application run on both a PC or Mac, as well as on any smart phone or tablet, is to implement it in HTML5. Any application developed in HTML5 will be cross platform, since it is ran from the web browser in stead of natively. However, the issue with a HTML5 application is the performance. Since it is not run natively, it can not utilize its device resources to its fullest, and these

applications tend to be slower than native applications. This chapter will introduce the motivation behind the thesis and present the research question together with what goals are trying to be achieved.

1.1 Motivation

As a computer engineering student for almost five years, I have grown to be a big fan of software development. The two recent years, my focus has become more and more on work related software development problems. The transition from software development as a student, to a work situation is huge, which is something I got to experience myself at an internship the summer of 2011. When you develop an application as a student, the main focus is often to "get it to work", while developing as an employee for an IT company has a much broader scope, such as architecture and following specific design and architectural patterns. To make an application that other software developers can continue to develop without your expertise is something that is much more emphasized when developing an application in a work situation. Personally, I value this way of building applications, and I try to incorporate as much good programming practice into every project I embark upon.

Together with my genuine interest for programming, I am also very interested in development for mobile devices, and the huge progress that has happened the last couple of years. The app stores for the different platforms makes it easy to download any kind of application, which is one of the reasons why these mobile devices has become so popular. Stunning and innovative new applications are released into the app stores every day, and these applications has changed the way we communicate with each other, and they make any mobile device inspiring to use.

To see how far the technology in terms of both hardware and software has come makes me eager to be a part of this innovative development process, and is the main factor why I choose a master thesis that will dive into problems on how to make cross platform mobile development easier, and to get a peak of what possibilities that exist.

1.2 Research Question

The following research question was defined

1. What are the possibilities and limitations when developing a mobile application native vs. HTML5?

1.3 Project Goal

The main goal with this thesis is to find possibilities and limitations when developing mobile applications natively vs. HTML5. To be better fitted to answer the research question, a prototype of a learning portal implemented natively for an

iPad will be developed. Moreover, the implementation phase will be used to find unforeseen pitfalls to the native approach to better define what kind of technology or functionality to exclude to make cross platform mobile applications as good as possible.

Chapter 2

Background Research

Developing cross platform mobile applications in a good way is difficult, and at the same time to choose the best development approach for an application is critical to be able to achieve a seamless experience for the end user. The two choices available is to implement applications with HTML5 or to build it natively, however the best approach to use will be different for every application. This chapter will introduce the topics that has been researched to be better fitted to carry out the implementation of the iPad application as well as answering the research questions stated in 1.2 - Research Question.

2.1 HTML5 applications

Developing HTML5 applications is very popular today. They are instantaneously accessible from any platform or mobile device because the application itself is run through the web browser, in stead of as a native application. To sell HTML5 applications in any app store is not possible unless a framework like PhoneGap¹ is used. PhoneGap automatically wraps a HTML5 application into each platforms native web view, which results in what is called a hybrid application. By using the PhoneGap framework, the application would appear as an application icon, just as any other mobile application, instead of having to access the application through the web browser. Even though a hybrid application might appear to be a native application externally, it is actually just showing the application in a web browser with a stripped down UI Where the toolbars are removed.

A downside with any HTML5 application is that they are dependent on network connection to function. Since its runtime environment is a web view, and it basically is a web site, the application would need to communicate with a remote server every time a user performs some operation. Early in 2010, W3C released the HTML5 offline mode feature that made it possible for the web browser to download all the files needed to be able to work without any Internet connection. This is done

¹Framework that lets users develop cross platform applications with HTML5, CSS3 and JavaScript instead of lower-level languages such as Objective-C. <http://phonegap.com/>

through what is called a manifest file, and all the offline dependent files, such as background images, would be listed here. These files would be stored in the web browser itself, and it would also be the web browsers job to distribute the saved files when the application asked for them.[8]

However, for a HTML5 application to function in offline mode, it would need to be connected to the Internet to download the different files needed at least ones. Smart phones or tablets are used a lot outside of the users home or workplace, which makes it important to have some kind of Internet connection available when using a HTML5 application. Besides from WiFi, the possibilities for connecting a device to the Internet is through 3G or EDGE, which has a fairly good coverage ratio in Norway as shown in figure 2.1.

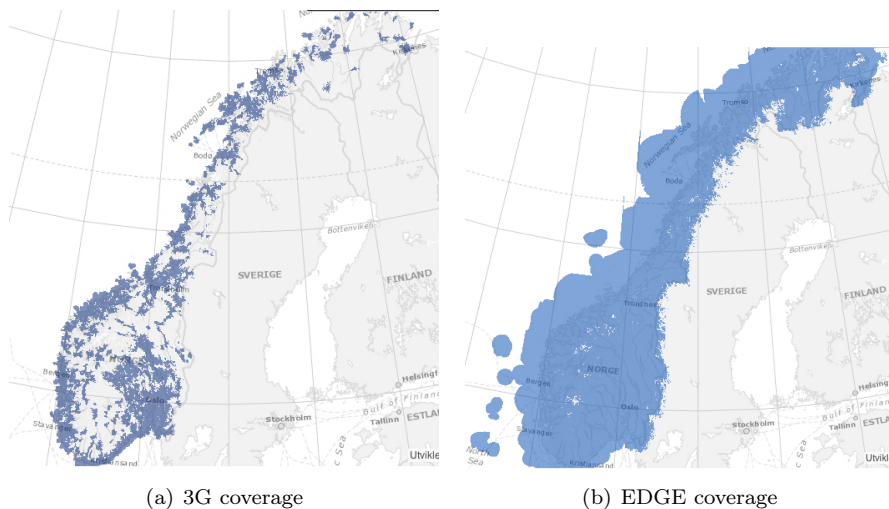


Figure 2.1: Coverage ratio of the 3G and EDGE networks in Norway[1]

As shown in figure 2.1(a) and 2.1(b), about 70% of Norway has 3G coverage, and almost 100% has EDGE coverage. However, the average download speed is 0,5 - 1,5 Mbit/sec for 3G networks, and 0,1 - 0,2 Mbit/sec for EDGE. To put that in perspective, in a best case scenario, 1 MB of data will be downloaded in 5 seconds with 3G, and in 40 seconds with EDGE. Even though there is network coverage almost everywhere in Norway, the speed is too slow to handle large data files. Outside of Norway, which actually has some of the best 3G and EDGE coverage in the world, you see that in general the mobile network coverage is not very good, as shown in table 2.1.

Good coverage	Belgium, Denmark, Ireland, Italy, Luxembourg, Montenegro, Netherlands, Norway, Portugal, Slovenia, Great Britain, Switzerland, Sweden, Germany, Austria, Bahrain, Brunei, Israel, Japan, Malaysia, South-Korea, Taiwan, Puerto Rico
Medium coverage	France, Greece, Croatia, Lithuania, Poland, Romania, Serbia, Spain, Hungary
Low coverage	Bulgary, Georgia, Finland, Cyprus, Latvia, Czech Republic, Australia, Philippines, China, Sri Lanka, Tajikistan, South-Africa, USA
Very low coverage	Estonia, Russia, Slovakia
No coverage	Other countries

Table 2.1: 3G coverage worldwide

This shows that even though the 3G and EDGE network coverage ratio seems to be fairly acceptable in Norway, it is not good at all in general. There is a long way to go when it comes to creating applications that are reliable on network connection to function. Even with the offline feature available, a HTML5 application is still not completely network independent, which some people might see as a design flaw for an application.

2.2 Native iPad applications

The biggest advantage when it comes to making a native application is how every feature from UI to functionality would be customized specifically for the iPad. It would follow the design paradigm intended for the iPad, which makes the user be familiar with how multitouch gestures works, as well as the layout of an application will be somewhat similar to others. This makes the interaction with an iPad much more personal than what it does with a PC or a Mac, which is one of the reasons why the tablet is now seen as the post-pc device, and people prefer to use an iPad instead of a PC or Mac to check email, surf the web, or read articles.²

From a more technical point of view, the biggest advantage of implementing an application natively for an iPad would be to have access to the full API. With a pixel resolution of 2048-by-1536, dual core CPU's and quad core graphics, the new iPad can handle advanced 3D graphics. To achieve this, external libraries like Open GL ES³ has to be used as well. Native development makes it possible to

²This was stated by Apple CEO Tim Cook in a keynote speech, and is not possible to source.

³A 3D graphics API designed for embedded systems such as iOS. <http://www.khronos.org/opengles/>

create applications that utilizes the devices full potential, which will be reflected in the applications performance in terms of being fast and responsive when a user performs different tasks

Native applications are made to run without any Internet connection, and because of this, making an application rely on data to be fetched from remote servers are an option, and not mandatory. Even though many native iPad applications chooses to fetch application specific data from a remote server, it can still run at any time without Internet connectivity. To fetch data remotely in any type of application is important that it is done as efficient as possible. Since any UI files, such as background images is fetched from the application itself, these often large files doesn't need to be fetched remotely. This decreases the applications need of generating lots of data traffic to function, which is very resource- and time consuming tasks to perform.

By using the Core Data API, data can be stored locally on the iPad, and can be worked with in form of objects, which is desirable in terms of both efficiency and keeping the object oriented programming style in tact. The Core Data API uses SQLite to keep track of any stored objects locally. SQLite is a small C-library that implements a stand-alone SQL database engine, and is a very popular choice for local storage on mobile devices.[9]

Building a native application also has a its drawback. It won't have the possibility to be deployed to other platforms than the one it was intended for. From a business perspective, this would mean multiple code bases to reach out to all the different platforms, which is both costly as well as time-consuming.

2.3 Native vs. HTML5

The two different approaches brings its own possibilities and limitations, and different general- and technological features are solved in different ways. Some features are better applied to native applications, while others to HTML5 applications. To get a better understanding of how both specific and well known problems are solved in the two approaches, some of them was compared to each other.

2.3.1 Multithreading vs. web workers

There are certain parts of the different devices API that HTML5 applications does not have access to. One in particular, which is used a lot when developing for mobile devices, is multithreading. Any touch from an user will result in some kind of work for the device. The work can be fast and easy, like updating some predefined text, or it can be more complex tasks like fetching data from a remote server when a user tries to log in to an application. For the latter case, any network based work needs to be performed in a separate thread, so that the applications UI will be responsive to the user, while the work is performed in the background.

When developing HTML5 applications, it is not possible to use threads as it is when developing with Objective-C. HTML5 applications use web workers instead, which are javascripts that are executed in the background, independently from

other scripts. This makes it possible for a web worker to continuously perform its job while the applications UI stays responsive. In theory, these two different approaches seems to be equally good, but there are very different design principles behind them. Web workers are relatively heavy-weight, they are expected to be long-lived, have a high start-up performance cost, and a high per-instance memory cost[10]. Because of their design, they are not intended to be used in large numbers as they very likely will end up hogging system resources. This last part is especially concerning when developing for mobile devices since these devices has limited resources available, and requires extra attention.

To execute certain operations in separate threads is built in to the very core of how iPad applications are built in Objective-C. Any code that needs their own thread to execute in, are put inside a block. Since the native development environment for the iPad has a full overview of the devices resources, and are built to take care of memory and resource allocations, using threads in Objective-C can be used without as much precautions as HTML5 would need.

Developers might think that it is easy to develop an application without having to use multiple threads or web workers, but it is not. Handling operations in the background is done in almost every application, since it should never be unresponsive for the user no matter how much work that needs to be done. Developing applications for mobile devices are very different from developing desktop applications, and it needs a much larger focus on efficiency and resource use. Due to this, to be able to use multithreading over web workers in an mobile application is desirable.

2.3.2 Application runtime environments

When an applications starts, it is launched in its own runtime environment. A HTML5 application is run inside the devices web browser, which means that all layout rendering is done by the web browser instead of iOS' native UI framework. The same is done when a HTML5 application is created with PhoneGap so it becomes a hybrid application. Layout rendering will still be done by the web view, but since the application is wrapped in Objective-C source code it will gain access to parts of the devices native API.[11] When a HTML5 application needs a lot of resources, it might not get all that it needs since it is the web browser that gets allocated memory and resources by the operating system, and not the application itself.

A native iPad applications layout rendering will be done by the iOS UI framework. A native application will also be given its own space in memory, which makes allocating resources to it happens on demand. This makes the native iPad applications run smoothly, and will act more responsive to the user.

2.4 Application design

When developing any software application, it is seen as good programming practice to follow specific architectural- and design patterns so that the application can

easily be further developed by others in the future.

A design pattern are descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context[12]. These patterns are followed to provide good delegation of tasks and to reuse already existing implementation specific application designs. Architectural patterns has a bit broader scope than a design pattern, and its function is to divide an application into subsystems to delegate tasks and doesn't provide any implementation specific details.

There are a lot of design patterns to choose from, but when developing an application that revolves so much around its user interface, the Model View Controller (MVC) design pattern⁴ was a clear choice to investigate further. This pattern is used in almost any mobile application, and will help generalize the structure of the classes used in an implementation. The MVC pattern divides an application into three areas of responsibility, and defines how they should communicate with each other as shown in figure 2.2.

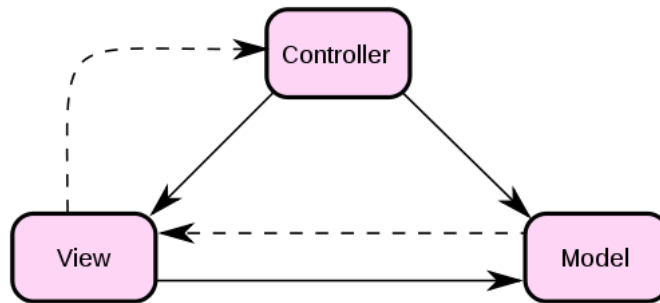


Figure 2.2: The Model View Controller Design Pattern

The three areas of responsibility shown can further be explained as following:

The Model	The domain objects or data structures that represent the application's state
The View	Observes the state and generates output to the users
The Controller	Translates user input into operations on the model

Table 2.2: The MVC design pattern[3]

(Further explanation about the different domains of the MVC pattern will be

⁴The MVC pattern has actually divided opinions about if it is an architectural- or a design pattern. Different programming languages also defines their own version of this pattern. In this chapters context, it is seen as a design pattern.

given in chapter 5 and 7.)

The well known architectural patterns has been developed to solve very specific application design problems, which doesn't present with that many different options. Keeping in mind that a prototype of a learning portal was going to be developed, to presume that users and professors would communicate with each other could be done at an early stage. The client-server architectural pattern proved to be a reasonable choice. The idea behind the pattern is that an applications logic should be located at a server, and that all users of the application would fetch any data needed from here, as shown in figure 2.3.

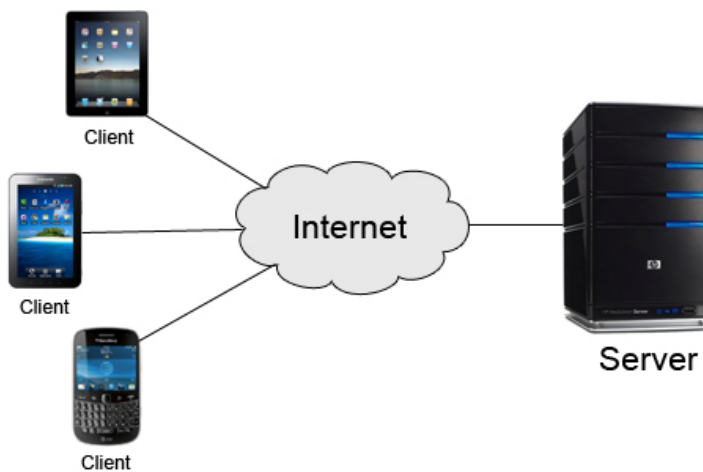


Figure 2.3: Client-Server architectural pattern

To decide to use the client-server architectural pattern would mean to create an application that could be completely network dependent, but it could be a good alternative depending on what features the learning portal application would contain.

Challenges with the client-server model

The client-server architectural pattern can experience a bottleneck as the server can be challenged beyond its capabilities. Due to this widely used application architecture, the option to store data with cloud service providers like Amazon and Google, has become more and more common. To use the cloud is a great way of solving server capability problems because the resources needed by a server is given by demand. The cloud solution makes the amount of traffic to the server be the factor of how much resources it will have allocated. This means that the cost of renting the web server also will be affected by the traffic it generates.

Another aspect to pay attention to when distributing an applications logic at a server, is the threat of being attacked by i.e. DoS attacks⁵ that is one of the most common ways of attacking a server. Since the applications functionality is actually stored at the server, it is important to be able to resist attacks like this. When renting a server with one of the big cloud service providers, the security regarding any server are in most cases taken care of in a very good way. The way to protect a server would then be in terms of building secure web services that the clients call upon. This to make sure that human- or application sensitive data doesn't get in the wrong hands, or that attackers can access your server through the web service API.

2.5 Interviewing software developers

A handful of software developers from Statoil ASA⁶ and Visma Consulting AS⁷ were interviewed to get an opinion from people with hands on experience in the field. They were all developing applications in HTML5 that made it possible to have only one code base, that results in both less work as well as lower project costs. A lot of the same responses came up when the interviewees were asked to give three positive and negative opinions about developing applications with HTML5- it is cross platform, but the lack of browser support and access to APIs sets a restriction to what can be achieved.

A senior consultant at Visma Consulting Denmark presented the native vs. HTML5 issue in a good way, which will be quoted below:

"I like the idea of an open standard language that is not financially beholden to any interest group. This is how the web was intended, and indeed what has made the web so ubiquitous. I am firmly opposed to the "walled garden" approach that seems to be prevailing at the moment. As I have personally stated on numerous occasions, I would like to have only one program on my phone, a browser.

Developing HTML5 apps is at the cutting edge right now. It's neither mature nor planned. It is an approach that has arisen from a fundamental need and desire that is currently lacking in most mobile OS vendors approach to app development. Many customers "bet on one horse" (usually Apple), but more and more are not accepting the fleeting dominance of a single OS vendor, and possible customer alienation that is inherent when snubbing other OS systems.

With many projects in all camps (Google, Apple, Microsoft..) being stunted due to the ongoing patent wars, I see no better approach than

⁵Short for denial-of-service attack, a type of attack on a network that is designed to bring the network to its knees by flooding it with useless traffic.

⁶A norwegian oil company. <http://www.statoil.com/en/Pages/default.aspx>

⁷A norwegian IT company. <http://www.visma.no/Prosjekt-og-konsulenttjenester/>

using a common, open, free and standardized programming approach to push forward application development.

Content should be free from technology. An app should not only be available on a single OS. My dvd should not only play on a Sony machine, my CD only on a Philips stereo, my book only on a Kindle, my App only on an iPhone. The web showed us that there was an alternative method for distribution of content and I can think of no better way to serve content on mobile devices than using the same language that powers the web.

From a technological angle, I hope that the OS vendors see HTML5 as an opportunity and not a threat. I hope that the use of exclusive content to power device popularity is removed from the equation and that a true cross platform programming interface is standardized and promoted."

From a business perspective, what is said here, are makes sense. Imagine developing an application with a single code base, and yet manage to utilize every different mobile device to its fullest. That would solve all the problems as they are today, but there is still a very long way to go before Garys, and so many others thoughts will come through. The one place where actually everything should be able to run, no matter what platform, is the web browser, but not even here does every HTML5 app or website run the same way, or even run at all. The web browser is the very backbone itself, where HTML5 and CSS3 was made to run in the first place, and there isn't even a common standard of how the browsers interpret the code. Many people argue that this is the heart of the problem, and to be able to step forward in this field, the foundation of it all needs to work flawless, so that it can be a stepping stone for others to see how it can work. Apple does contribute with the focus on further developing the HTML5 standard with its iBooks Author⁸ application that was launched January 12th, 2012. The iBooks Author application is mostly drag and drop based, but it is possible to incorporate HTML5 widgets inside a book, which makes it possible to have animations, videos and all the other effects that can be created with HTML5 and CSS3. The initiative that Apple has taken here is a very good one. It shows that even though they are holding on to making iOS applications natively for the iPad, iPhone and iPod Touch, they do see the opportunities that HTML5 offers, and it is a true call out to everyone that this technology needs to be used for what it is worth.

⁸A framework for creating digital books for the iPad.

Chapter 3

From Research to Prototype

Taking the research done in chapter 2 into consideration, the structure and architecture of the prototype iPad application could be done. To be able to develop a mobile application natively has so many benefits, which makes it the desirable choice. The native approach brings many upsides, but it also brings some problems that needs to be justified. To decrease the workload of having to implement the application specifically for each mobile platform, the client-server model was chosen. With this application architecture, the amount of code needed for each native implementation would be drastically reduced since the business logic of the application would be kept at a remote server. This makes it only necessary to implement a GUI, with a public interface that will handle communication and fetching the remotely stored data. The application would fall under the category of being network dependent, but in a much less nature than a HTML5 application. Natively, large files such as background images will be implemented into the application, which reduces the amount of data needed to be fetched remotely. The only data that would need to be fetched would be in the form of small text represented objects, which hopefully would make the amount of data needed to be fetched acceptable to be downloaded from both 3G or EDGE network connections. With access to the iPad's API, any fetched data would be stored and handled locally, which makes the application function even without a network connection. When new data is stored at the web server, only the recent updates would be downloaded to the application when it is online. This approach wouldn't be a true cross platform application, but the upsides of being able to implement natively would justify the increase in workload to deploy the application at multiple platforms, based on the decisions made.

Since the prototype application is a learning portal, the name of the application was chosen to be Education+, and this chapter will go further in detail on what technologies and choices that has to be made, to make it as easy as possible to implement different native versions of it.

3.1 Wanted functionality

To be able to test some of the theories stated in this thesis, some basic functionality will be added to the Education+ application. The web server will contain all the business logic of the learning portal, and the iPad application will mainly be front-end implementation together with an interface to handle communication with the web server. Another emphasized focus area for the web server is that it should be easy to add new features to it, as well as it should be scalable so that the work with this application can continue later on. The wanted features and functionality of the web server and the iPad application is shown in table 3.1 and 3.2.

	Web Server
Modifiable	The web server should be dynamic and modifiable in a way that there should be easy to add new features, as well as update existing features
Creating objects	The web server should be able to create user, admin, school, course and news objects
Communication	An admin should be able to communicate with its users/students by posting news to the iPad application
Functionality	An admin is going to be able to create new courses and add existing students to a course. An admin should also be able to post news to the different courses he or she is administrating

Table 3.1: Web server functionality

iPad application	
Modifiable	The iPad application should be dynamic and modifiable in a way that there should be easy to add new features, as well as update existing features
Creating objects	Users of the iPad application should be able to register and create new user objects
Functionality	A user should be able to login, and also get an overview over what courses he or she is attending, and see the courses corresponding news entries
Design principles	The iPad application should follow the iOS development design principles by using built in view controllers

Table 3.2: iPad application functionality

3.2 Technical Research

The technical research required was divided in two parts; web server- and iOS development related. The first part was going to be written in Java and could be approached from different angles, while the latter involved learning Objective-C, which seemed to be an extensive job. Much narrower research had to be done regarding the choices made, so that the implementation specific choices could be utilized to its fullest.

3.2.1 The iPad application

Since there are no courses at NTNU that teaches iPhone and iPad development, knowledge and competence had to be acquired on our own initiative.

To find good literature about how to learn Objective-C was important to be able to be introduced to the new programming language in a satisfying way. A much used resource regarding Objective-C specific problems, were IRC. A chat room made specifically for implementing native applications for the iOS platform called #iphonedev¹ was found to be an educational resource, with a genuine interest in the topic.

After discussions in the #iphonedev chat room, there were unanimously answers that the best way to gain knowledge about iPad and iPhone development was through iTunes U² and the CS193P: iPad and iPhone App Development course from Stanford University, California.

¹Any chat room at an IRC server are prefixed with a # sign. The #iphonedev channel is located at the irc.freenode.net server. It is a free and open source software-focused IRC network, encompassing more than 70,000 users and 40,000 channels.

²iTunes University - <http://www.apple.com/education/itunes-u/>

Every lecture from that course is filmed, and lecture notes and assignments are posted at the iTunes U application that you can download to your iPhone, iPad or Mac as shown in figure 3.1 and 3.2.

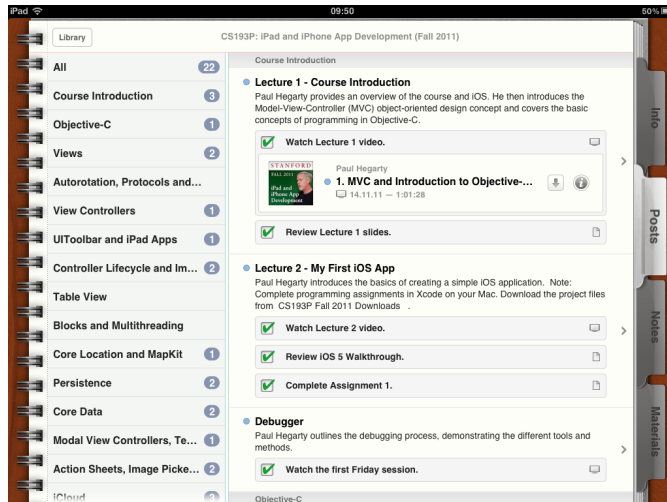


Figure 3.1: iTunes U - CS193P course overview

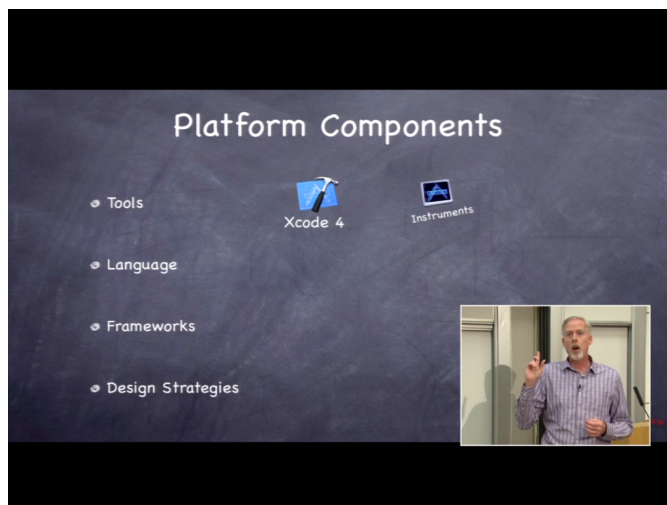


Figure 3.2: iTunes U - Screenshot from a lecture video

This course is taught to students that has never developed iOS applications before, so the lectures starts out with an introduction to the language with corresponding assignments that can be worked with to get a basic understanding of how

things work in Objective-C. The first lectures covered the basic concepts of programming in Objective-C, as well as a short introduction to the Objective-C IDE Xcode. The introduction to Objective-C and a more comprehensive walkthrough of how things work will be given in chapter 6 - The Concepts of Programming in Objective-C.

3.2.2 The web server

With the structure chosen for the learning portal, there were a couple of crucial aspects of the web server that needed to be thoroughly thought through, and some of these aspects are presented in this section.

The web server needs to be modifiable with regards to easily adding new functionality to it, and since the iPad application somewhat relies on fetching data from it, the server needs to be both efficient and available. To be able to make the web server modifiable, the underlying architecture needs to allow it, which is why the web server will follow the MVC architectural pattern³. In any software development project, to identify any pitfall as early as possible will help writing as little redundant code as possible, and also help avoid discovering too many unplanned items along the way.

The Java Spring framework⁴ is a well known and continuously updated framework, which makes it reliable for developing a fast and available web server. It has a very good API when it comes to making different web technologies work together, as well as trusted security features. The security part is especially important since all of the data for the whole application is seemingly accessible to the public. With the Spring framework, annotations can be made at method-level, as shown in listing 3.1.

```
@PreAuthorize("hasRole('ROLE_ADMIN')")
@RequestMapping(value="/user/{email:.+}", method = RequestMethod.GET)
@ResponseBody
public User getUser(@PathVariable String email) {
    User u = (User)userService.getUserByEmail(email);
    return (u != null) ? u : null;
}
```

Listing 3.1: Security annotated Java method

Security annotation makes sure that only people with the right privileges can access the web services, and it prevents that sensitive data becomes accessible to the public.

3.2.3 Web services

A web service is a method of communication between two electronic devices over the web. W3C defines a web service as "a software system designed to support interoperable machine-to-machine interaction over a network". It has an interface described in a machine-processable format, such as XML or JSON[13].

³The Spring framework has created their own version of the MVC pattern. In terms of the web server, MVC will be referred to as an architectural pattern.

⁴<http://www.springsource.com/>

When building an application that is dependent on communicating with a remote server, and created to target mobile platforms, it is very important to be careful with both what amount of data is sent, as well as what format it is sent in. Any data that is sent needs to be shrunk as much as possible, both to be efficient, and to allow a good user experience for users with limited network connections. The web service architecture and the data format the web services will use will be presented in the following two subchapters.

RESTful architecture

REST is a style of software architecture that was defined by Roy Fielding in 2000. The REST architecture is based on the client-server model, which makes it fit very well into the Education+ application. The key goals of REST are:

- Scalability of component interactions
- Generality of interfaces
- Independent deployment of components
- Intermediary components to reduce latency, enforce security and encapsulate legacy systems.

The REST architecture has six constraints[4]. The most important part is the restriction of a uniform interface. The three constraints most applicable to the Education+ application are stated in table 3.3.

Identification of resources	Individual resources are identified in requests, for example using URIs in web-based REST systems. The resources themselves are conceptually separate from the representations that are returned to the client. For example, the server does not send its database, but rather, perhaps, some HTML, XML or JSON that represents some database records expressed.
Manipulation of resources through these representations	When a client holds a representation of a resource, including any metadata attached, it has enough information to modify or delete the resource on the server, provided it has permission to do so.
Self-descriptive messages	Each message includes enough information to describe how to process the message. For example, which parser to invoke may be specified by an Internet media type (previously known as a MIME type). Responses also explicitly indicate their cacheability.

Table 3.3: Guiding principles of the REST uniform interface constraint[4]

REST is a complicated architecture, and it is really all about using the true potential of HTTP. The HTTP protocol is oriented around verbs and resources, like the two most common- GET and POST. But HTTP defines more verbs, like PUT and DELETE, and with these four verbs it is possible to achieve all the basic functions when it comes to persistent storage. In general these functions are together called CRUD- Create, Read, Update and Delete. A comparison of these functions can be seen in table 3.4.

Operation	SQL	HTTP
Create	INSERT	PUT
Read	SELECT	GET
Update	UPDATE	POST
Delete	DELETE	DELETE

Table 3.4: CRUD functions

A good example of "Identification of resources" would be that the Education+ application uses a database that holds all the users, and that these users are going to be created, fetched, updated and deleted through web service calls. With a non-restful approach the call to the web services could look something like this:

```
/user_create
/user?id=xxx
/user_edit?id=xxx
/user_delete?id=xxx
```

while a RESTful approach would expose the public API with a single base resource:

```
/user
/user/xxx
```

With the RESTful approach a POST or PUT request is sent to a URL with the data it wants to create or modify in the HTTP body. To retrieve a user, a GET request is sent to /user/xxx, and to delete a user, a DELETE request is sent. An example is that a GET request would never modify any data, this is what PUT, POST and DELETE are for. This shows that the individual resource is identified by its request, and not by the way the URL is built up.

JSON- JavaScript Object Notation

JSON was developed in 1999 and has recently been more and more used due to the transition to the post-pc era. With the massive use of mobile devices and the need of being online everywhere JSON has an advantage over XML- it is lightweight and easy for devices to both parse and generate. For the human eye, JSON differs from XML mainly in notation. A key/value pair in JSON would look like:

```
{"key" : "value"}
```

where the equivalent XML would be:

```
<node><key>value</key></node>
```

This shows that XML needs to name the node for each instance, as well as each key/value pair can be defined as either an attribute, or an object. This is almost always not worth the extra notation, and it will end up costing a lot of bandwidth and processing time when the files get bigger, which is demonstrated in figure 3.3[14].

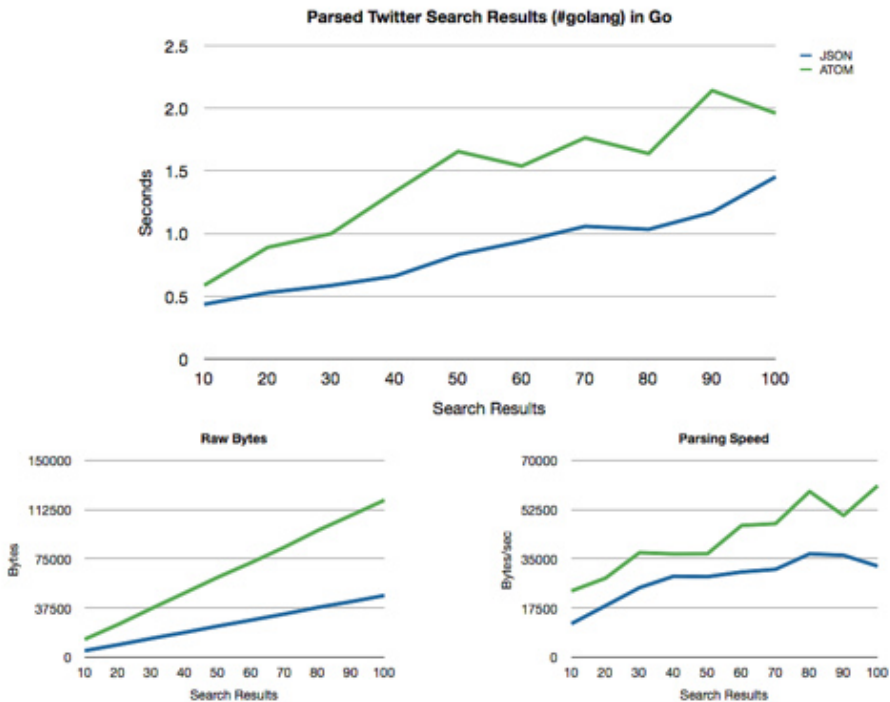


Figure 3.3: XML vs. JSON - Parsed twitter search results[2]

In the raw bytes diagram in the lower left corner, the two graphs increase linearly in bytes as the number of search results increase. After only 100 returned results, the XML file is almost three times as big as the corresponding JSON file. XML also needs to have a schema supplied so that it will know whether the values are objects or attributes, as well an XML parser. With JSON, no additional schema or external parser is needed. Almost every language has either a built-in or 3rd party JSON library which makes it easy to use at any platform[2], and is why JSON will be used to communicate data between the server and its clients in the Education+ application.

Chapter 4

Education+ Overview

The majority of students in Norway uses It's Learning¹ which has split opinions when it comes to both user experience and usability. With today's technology, a web portal like It's Learning should be available at several platforms and handheld devices. The user experience and benefits of using a web portal would be much better when it is more reachable.

This chapter will give a brief introduction to the Education+ application and its features to give a better overview before specific implementation details are introduced in chapter 5, 6 and 7. General information about the application such as URL and login credentials are given in appendix A - Education+ user manual.

4.1 Education+

The Education+ portal is made up by three parts, a web server with the applications logic, an administrator panel, and an iPad application. The administrator panel is implemented as a web site so the professors can interact with the Education+ users and system. The iPad application will be used by students to get information about courses they attend and to receive news and updates from their professors. The two front-end parts of the system, both the administrator panel, and the iPad application, communicates with the same web server, and with the same public API that the web server exposes. The web server is responsible for keeping track of all the users, and also to provide its clients with the data they request. There is no problem to communicate with the web server to request or receive data no matter what platform you are on. The only criteria to communicate with the web server is the need to send its requests in form of JSON objects, which is possible in almost every programming language used today.

¹<https://www.itslearning.com>

4.1.1 Education+ - The web server

The web server is what handles all the data requested to be created or fetched, and all other business logic. The Education+ applications logic is exposed through a public API, so that any device or any platform can communicate with it. The prototype implements web services for creating and fetching information about administrators, users, courses and news entries are created.

4.1.2 Education+ - The administrator panel

Professors using the application will have an administrator account where they can perform tasks such as creating courses, add students to a course, and create news entries. The layout of the administrator panel is shown in figure 4.1.



Figure 4.1: The Education+ administrator panel

The menu at the left hand side incorporates the following functionality:

Home	The welcome screen where you are taken after a successful login.
Admin	Displays the other administrators at the school.
User	Displays all the users/students at the school. All these users are created with the iPad application.
Course	Gives an overview of the courses an administrator teaches, as well as the possibility to see what students are taking the different courses, create new courses, and add new students to a course.
News	Gives an overview of the different news that belongs to the different courses. An administrator can create news entries for the courses he or she teaches.

Table 4.1: Overview of Education+ administrator panel

4.1.3 Education+ - The iPad application

Students will use the Education+ on an iPad. This is where students register their accounts and where they will receive news and course information. Both the administrator panel and the iPad application has the same GUI, which is shown in figure 4.2 and 4.3.

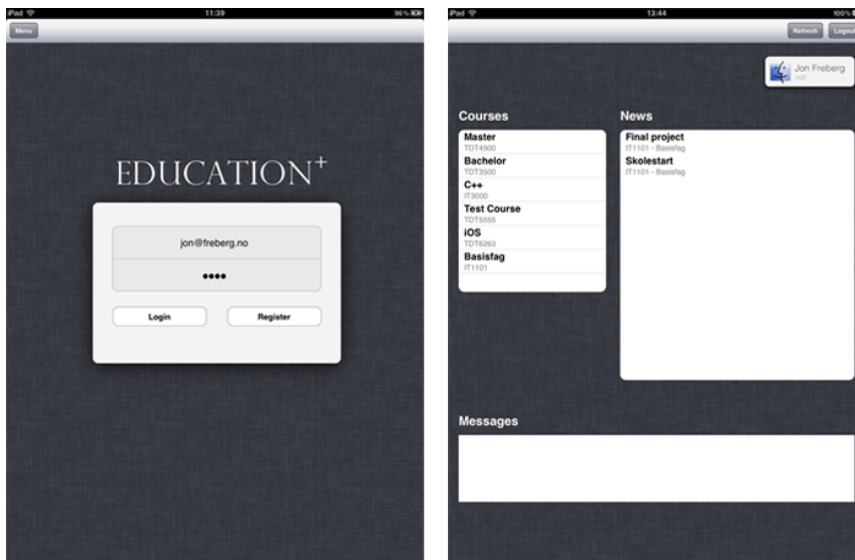


Figure 4.2: The Education+ iPad application

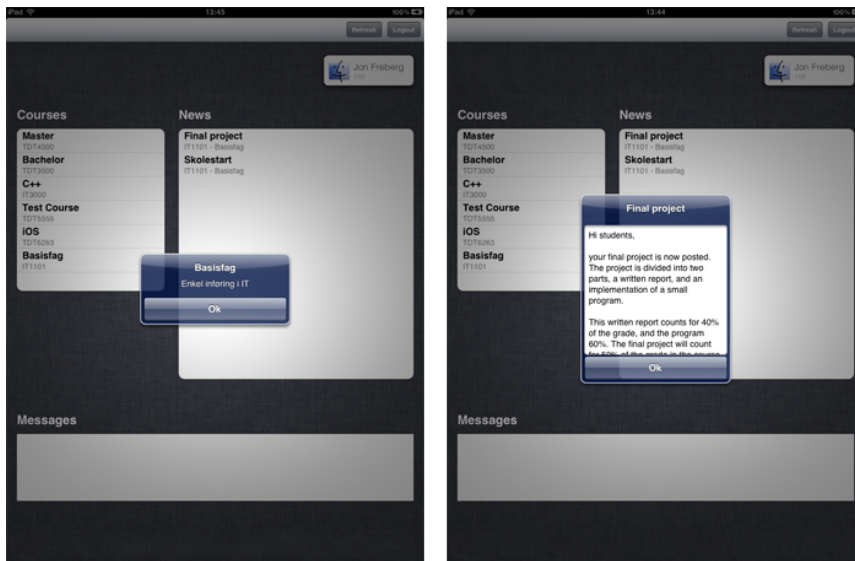


Figure 4.3: Education+ course and news overview

A student's course list will be updated when a professor has added a student to its course in the administrator panel. If a professor adds a student to its course, or creates a new news entry for a course when a student is using the iPad application, the student will have to press the refresh button in the top right corner of the application to fetch the latest updates from the web server. If not, an updated list of both courses and news are presented to the student after he or she has logged in.

Chapter 5

Implementation - The web server

The web server will have two main functions; exposing the web services through a public API, and host the administrator panel where administrators can update and create data and information for the users of the iPad application. All communication with the Education+ application happens here, no matter what type of client.

This chapter will give a technical summary of how the web server was implemented by using the Spring framework, and it will describe how the MVC design pattern was used in the Spring context.

5.1 The Spring Framework

Spring is a very powerful framework that solves many common problems when it comes to developing web applications. Spring's goal is to provide infrastructural support to developers so that they can have their main focus on the business logic of the application. To separate configuration and code is something Spring does very well, which helps the developer to achieve a well structured application. The Spring framework works solely with POJO's. The idea is that these classes sewed together with the configuration instructions creates an application system ready to use.

The Spring framework consists of powerful modules, and it is very modular so that it is easy to add new Spring features that is needed along the way of any project. The different features are organized into about 20 modules, which again are grouped into the categories core container, data access/integration, web, AOP, Instrumentation and Test. Figure 5.1 illustrates how these features are organized.

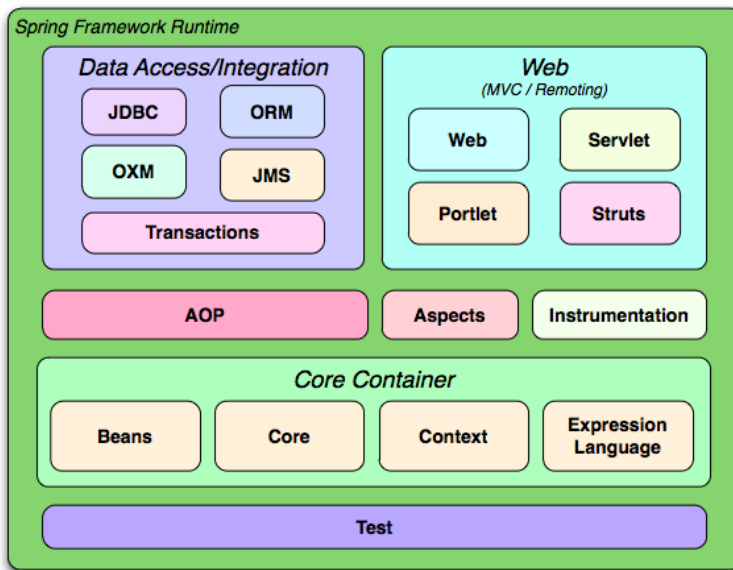


Figure 5.1: Spring Framework Overview

For the purpose of the Education+ web server, the following Spring features are emphasized:

- Model View Controller (Web)
- Beans and Dependency Injection (Core Container)
- ORM and Java Persistence API (Data Access/Integration)
- Security

5.1.1 Model view controller

Spring MVC helps in building flexible and loosely coupled web applications. This architectural pattern helps in separating the model (business logic), view (user interface) and the controller (user input)[3], as shown in figure 5.2¹.

¹The Spring frameworks version of the MVC pattern is by Spring referred to as an architectural pattern. MVC will be therefor be referred to as this in the context of the web server

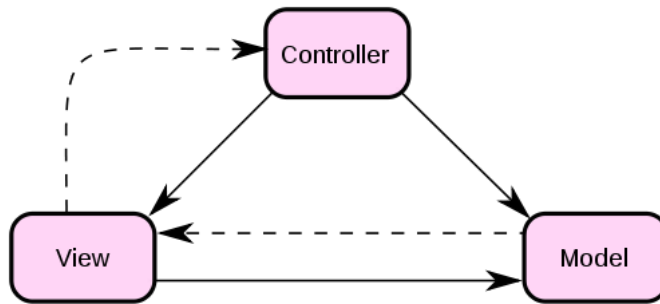


Figure 5.2: The Model View Controller Architectural Pattern

In the heart of Spring MVC, is the dispatcher servlet as displayed in figure 5.3[15][16] [17]. The dispatcher servlet is what manages the entire request-handling process. Since the web server serves two different purposes; exposing web services through a public API, and also implementing a web interface that acts as the administrator panel, the view part of this pattern is not used in the same way in both situations. The original purpose of the MVC pattern is retained for the administrator panel, while it is slightly modified for the part that exposes the web services. When a request is sent to the dispatcher servlet, it delegates the job by invoking the appropriate controllers to process the request.

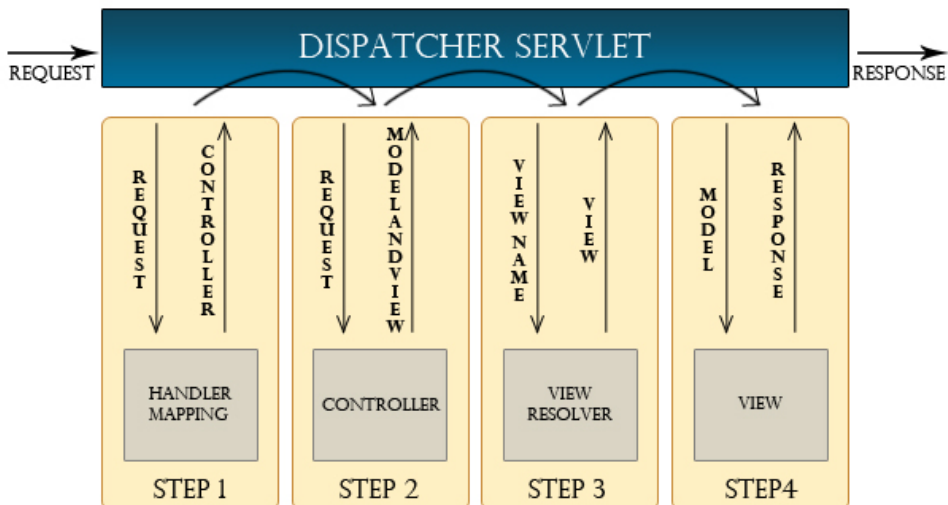


Figure 5.3: Request flow in the Spring MVC Framework - Administrator Panel

The Spring MVC framework handles a request in the following order:

1. The DispatcherServlet first receives the request
2. The DispatcherServlet consults the HandlerMapping and invokes the Controller associated with the request
3. The Controller process the request by calling the appropriate service methods and returns a ModelAndView object to the DispatcherServlet. The ModelAndView object contains the model data and the view name
4. The DispatcherServlet sends the view name to a ViewResolver to find the actual View to invoke
5. Now the DispatcherServlet will pass the model object to the View to render the result
6. The View with the help of the model data will render the result back to the user.

Further explanation of the segments mentioned will be done in section 5.3.

As mentioned earlier, there are fewer sequences to the dispatcher servlets action when a web service is requested. After the dispatcher servlet invokes the controller associated with the request, the actual model gets returned as the response. This model is turned into a JSON object so that it can communicate with whatever client that is requesting it, as shown in figure 5.4.

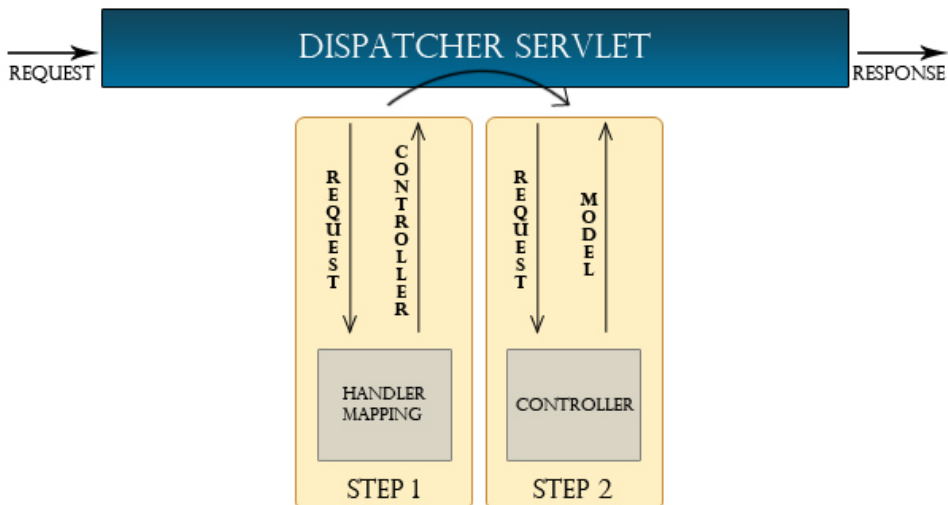


Figure 5.4: Request flow in the Spring MVC Framework - Web Services

5.1.2 Dependency Injection

Dependency Injection is a design pattern that is used to define the object dependencies. It is sometimes called Inversion of Control, and widely known as the Hollywood Principle- "Don't call us, we'll call you". With respect to the architecture, the idea is that the Service layer is given instances of DAO objects, instead of being responsible for looking them up. An example is provided to give a better understanding of it.

Imagine two friends, and one of them asks the other to drive him to the supermarket. Not a very hard task, the driver gets his car, and drives his friend to the supermarket. But what if suddenly 5 people were coming along, then a bigger car would be needed. In Java terms, the good news is that most cars implement the same interface (steering wheel, accelerator, barker etc.), and any person can drive any car. At the root of it, that's what dependency injection is all about. Instead of having to ask for a car with more than five seats, it is automatically given, without having to be responsible for getting the right kind of car.[18]

Dependency injection comes primarily in two different forms- Constructor or setter-based. Spring recommends to use the setter-based variant, and that is what is used at the Education+ web server. This means that there are no-argument constructors, and the setters are used to wire up dependencies between the different objects.

5.1.3 Java Persistence API

JPA is a standard for Java object-relational mapping. It specifies a set of annotations and an interface to perform persistence operations with any mapped objects. The Spring ORM enables the communication between the database and the Java objects. Due to this, a programmer won't need to think about MySQL and database configuration. A persistence entity is a lightweight Java class whose state is typically persisted to a table in a relational database, and the persistence entities corresponds to rows in a database table. Now java objects can be saved as a row in a database table with just a few lines of code, and just as easy create new Java objects from a table row.

5.1.4 Security

Security is an important feature of the Spring framework. The Spring security makes it possible to secure a complete web application with just a couple of lines of code in a XML file. Spring security works with filter chains, which tells what permissions a user should have to access certain URLs. With the use of wild cards, applications are easily secured, as the example below shows.

```
<intercept-url pattern="/navigation/**" access="hasRole('ROLE_ADMIN')"/>
```

With this single line of code, every URL with this structure can only be accessed by admins. If someone that does have an admin role assigned to them, the user gets redirected to a login page.

Security can also be applied to methods directly in Java. This makes it possible to set restrictions to who can call these methods, which covers the problem of unwanted users that tries to access the web server without using the web interface.

5.2 Glassfish vs. Tomcat

When making the decision of what kind of web server to use, there was a couple of different options presented, mainly with respect to how broad support they have to different features. The two obvious options to deploy the Education+ Java server was GlassFish or Tomcat. The main difference between these two servers are that GlassFish is an application server², while Tomcat is a web server with a servlet container, which makes it feasible to use with a Spring web application. Both servers are written in Java, made specifically for the Java Runtime Environment and deployment of Java applications. When it comes to administration tools and options, GlassFish has a broader specter of choices made available to the administrators, which is a feature that is always good to have, especially since the web server is made in a way so that further development should be easy. Performance wise, the two servers are pretty much the same. The only difference worth mentioning is that Tomcat is a little bit faster to load, while GlassFish is a little bit faster to reload. Since both servers have integrated support of deploying WAR files directly from Eclipse, to be able to reload the application as fast as possible is desirable to decrease deployment time as much as possible. The actual deployment of the WAR file with GlassFish takes about a second, while with Tomcat it takes about 3 seconds. In large software development projects, an application gets deployed and tested hundreds, maybe thousands of times, which will sum up to quite the difference.

With these results in mind, the GlassFish server was chosen to decrease deployment time, as well as achieve a wide specter of configuration possibilities for further development.

²http://en.wikipedia.org/wiki/Application_server

5.3 Architecture

One of the great guidelines to follow when using the Spring framework is how it encourages to use Java interfaces through out the layers of the architecture, so that communication happens through these. The benefit of Java Interfaces is that it allows multiple inheritance of an interface, but not of implementation. The implementation that includes instance variables and method implementations is always singly inherited and because of this, confusion will never arise over what inherited instance variable or method implementation to use[19]. The MVC architectural pattern made the classes communicate with each other in the following way:

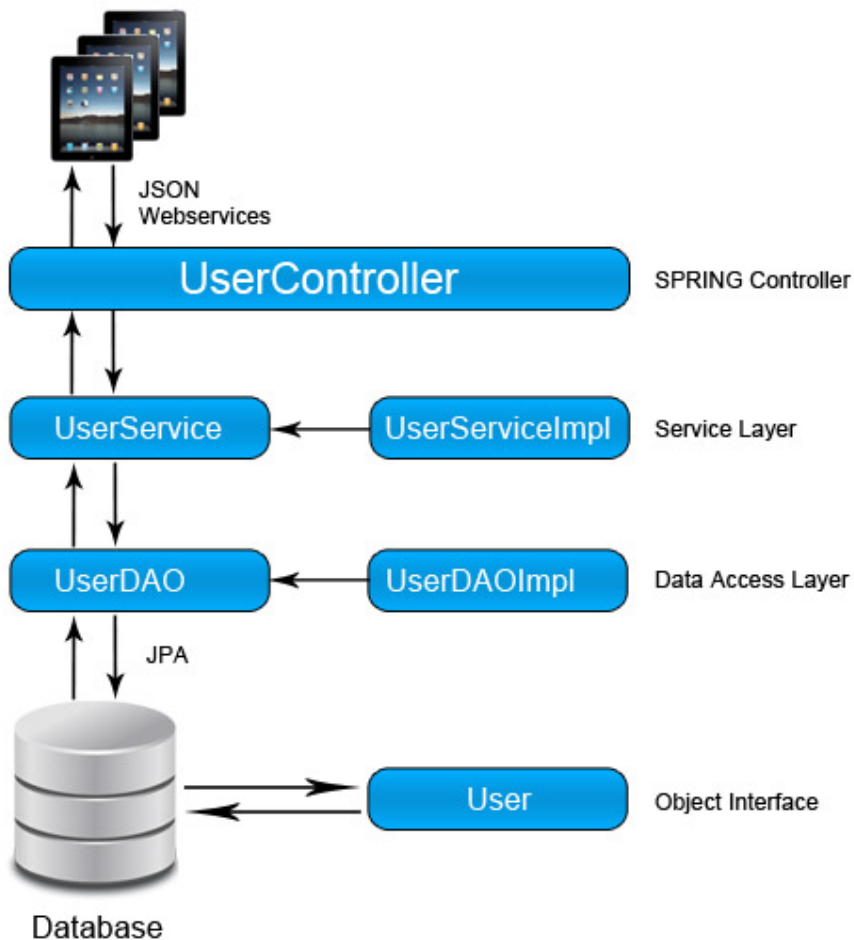


Figure 5.5: Education+ web server architecture

The figure also shows how the layered architecture of the web server hides

the functionality of the public API deeper into the application and makes it less available for the public. A better look at the project structure is given in figure 5.7:

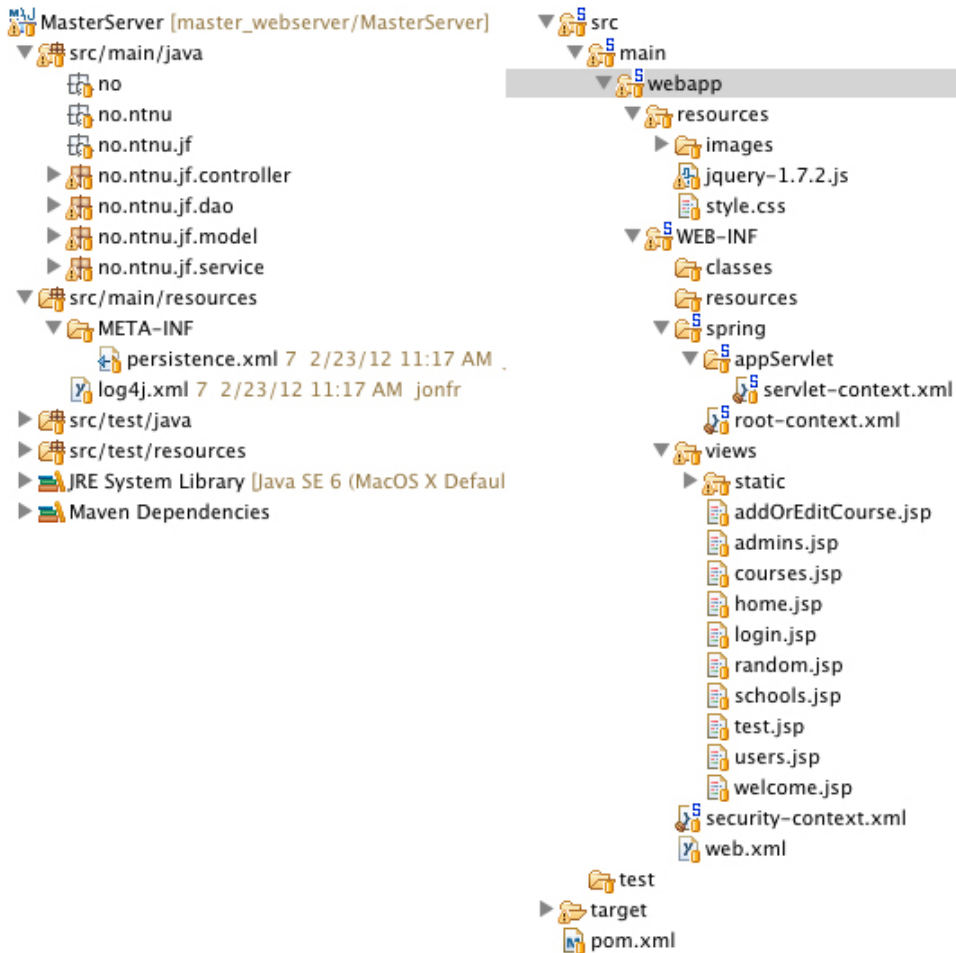


Figure 5.6: Education+ structure

There are a total of five types of objects that together creates the functionality necessary for this prototype implementation. These five objects are Admin, User, School, Course and News, as the class diagram in figure 5.7 explains.

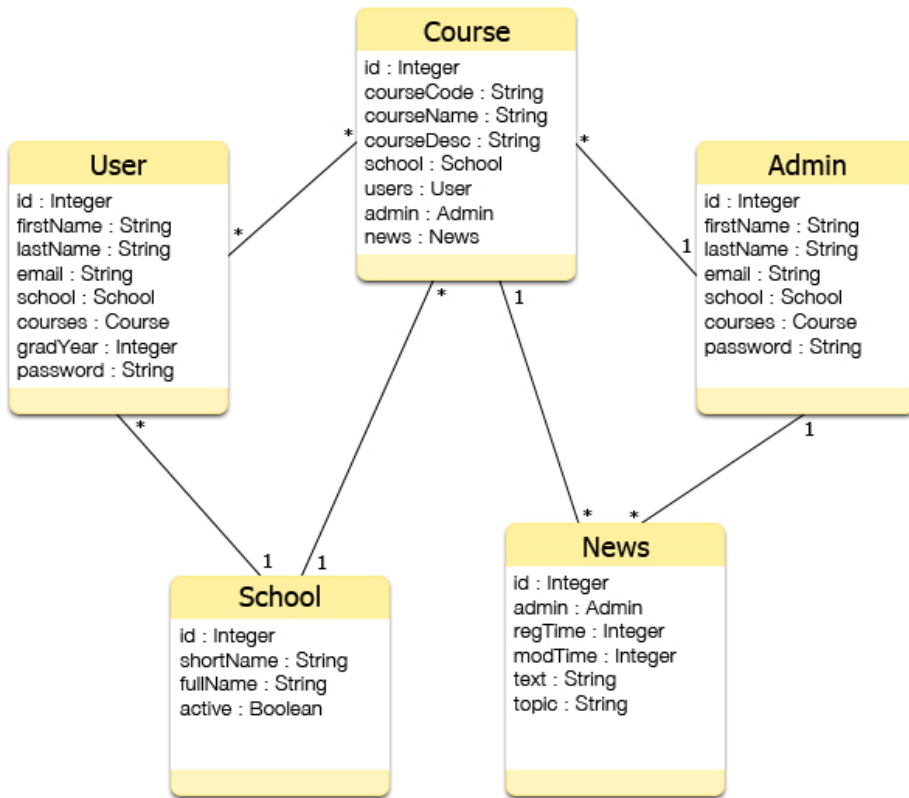


Figure 5.7: Education+ Web Server Class Diagram

The following parts of this chapter will go in detail about where the different objects are located, and what their tasks are.

5.3.1 Model classes

The different model classes implemented for the Education+ application is shown in table 5.1.

Contains	
Admin	Admin.java AdminInterface.java
User	User.java UserInterface.java
School	School.java SchoolInterface.java
Course	Course.java CourseInterface.java
News	News.java NewsInterface.java

Table 5.1: Model objects located in no.ntnu.jf.model package

The src/main/java package contains the sub package no.ntnu.jf.model which is where all the object interfaces and implementations are located. This is where the different objects gets their instance variables assigned to them, and it is also the class that creates the database entities, and the relationships between them. To achieve this, Spring plays an important role by using what is called annotations. Annotations are used by annotating certain parts of the java code so that Spring can collect information about the object and its relationships, and from this turn the representation of the java object into a database entity. To create a new entity in the database, a model class is annotated as follows in listing 5.1.

```

@Entity
@Table(name = "t_course")
@NamedQueries({
    @NamedQuery(name = QueryNames.FIND_ALL_COURSES, query = Queries.
        FIND_ALL_COURSES_QUERY),
    @NamedQuery(name = QueryNames.FIND_ALL_COURSES_FOR_SCHOOL, query = Queries.
        FIND_ALL_COURSES_FOR_SCHOOL_QUERY),
    @NamedQuery(name = QueryNames.FIND_ALL_USERS_IN_COURSE, query = Queries.
        FIND_ALL_USERS_IN_COURSE_QUERY)
})
public class Course implements CourseInterface, Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "course_id")
    public long courseId;

    @Column(name = "course_code")
    public String courseCode;

    @Column(name = "course_name")
    public String courseName;

    @Column(name = "course_description")
    public String courseDescription;

    @ManyToOne(cascade = {CascadeType.MERGE})
    @JoinColumn(name = "school_id", nullable = false)
    private School school;

    @ManyToOne(cascade = {CascadeType.MERGE})
    @JoinColumn(name = "admin_id", nullable = false)
    private Admin admin;

    @ManyToMany(cascade = {CascadeType.MERGE}, fetch = FetchType.EAGER)
    @JoinTable(
        name = "t_user_course",
        joinColumns = {@JoinColumn(name = "course_id", referencedColumnName = "
            course_id")},
        inverseJoinColumns = {@JoinColumn(name = "user_id", referencedColumnName = "
            user_id")})
    private List<User> users;

    public Course(){
        ...
    }

    /* Getter and Setters */
}

```

Listing 5.1: Snippet of no.ntnu.jf.model.Course.java object implementation

This is just a normal java class that defines an object. The different annotations here tells Spring what kind of type and relationship the entity will have in the database. An overview over the function of the different annotations are given in table 5.2 below.

@Entity	Creates a database entity of a Java class
@Table	Gives a specific name to a database table. Default name will be classname unless this annotation is given
@NamedQuery-ies	Defines the queries that will fetch data from the database. All queries are written in JPQL. All queries defined will return java objects of the specific class they are annotated in.
@Id	Assigns primary key
@GeneratedValue	With parameter GenerationType.IDENTITY set, this primary key will be an auto incremental BigInt, starting with the value 1
@Column	Gives a specific name to the database column. Default name will be instance variable name unless this annotation is given
@ManyToOne	Defines a many to one relationship where the source object references another object. This will create a foreign key relationship in the MySQL table. CascadeType tells how the relationship is handled. CascadeType.MERGE tells in this case that if a course is deleted, do not delete its referenced objects.
@JoinColumn	Describes what row name that the foreign key should reference.
@ManyToMany	Defines a many to many relationship between two objects. To oblige 1st normal form of database normalization, this annotation creates an additional table so that repeated values are not present in the same database table.
@JoinTable	Defines what database rows the relationship will be joined with.

Table 5.2: Database entity creation annotations

5.3.2 Data Access Object classes

The different data access object classes implemented for the Education+ application is shown in table 5.3.

Contains	
Admin	AdminDAO.java AdminDAOImpl.java
User	UserDAO.java UserDAOImpl.java
School	SchoolDAO.java SchoolDAOImpl.java
Course	CourseDAO.java CourseDAOImpl.java
News	NewsDAO.java NewsDAOImpl.java

Table 5.3: Model objects located in no.ntnu.jf.dao package

The no.ntnu.jf.dao package is where the classes that are used for data access are stored. These are the classes that handles the actual persisting of java objects to the database. A partial implementation of the CourseDAO object is shown in listing 5.2 as an example.

```

@Repository
public class CourseDAOImpl implements CourseDAO {

    @PersistenceContext
    EntityManager entityManager;
    @Autowired
    private SchoolDAOImpl schoolRepository;
    @Autowired
    private UserDAOImpl userRepository;
    @Autowired
    private AdminDAOImpl adminRepository;

    ...

    public Course saveCourse(Course c) {
        List<User> tempUserList = new ArrayList<User>();
        for(User user : c.getUsers()) {
            User tempUser = (User)userRepository.getUserByEmail(user.getEmail());
            tempUserList.add(tempUser);
        }

        c.setUser(tempUserList);
        try {
            if(c.getCourseId() > 0) {
                return entityManager.merge(c);
            } else {
                if(c.getSchool() == null) {

```

```

        School school = schoolRepository.getSchoolByShortName(c.getSchool().
            getSchoolShortName());
        c.setSchool(school);
    }
    if(c.getAdmin() == null) {
        Admin admin = (Admin)adminRepository.getAdminByEmail(c.getAdmin().
            getEmail());
        c.setAdmin(admin);
    }
    entityManager.persist(c);
    return c;
}
} catch (Exception e) {
    System.err.println("Something went wrong when trying to save course.");
    return null;
}
}

public List<Course> findAllCoursesForSchool(String shortName) {
    TypedQuery<Course> query = entityManager.createNamedQuery(QueryNames.
        FIND_ALL_COURSES_FOR_SCHOOL, Course.class);
    query.setParameter(1, shortName);
    return query.getResultList();
}
}

```

Listing 5.2: Snippet of no.ntnu.jf.dao.CourseDAOImpl.java object implementation

There are three types of annotations used; `@Repository`, `@PersistenceContext` and `@Autowired`. Also worth mentioning is the `EntityManager`, and how this object works. The `EntityManager` is the object that handles the actual persisting of objects to the database. It is created by an `EntityManagerFactory` which is declared in a configuration xml file. As presented in the source code above, both the `EntityManager` object, and the objects that are annotated with `@Autowired`, are never actually instantiated. They are all objects that are created by `Dependency Injection`, so they are created and handled by `Spring` when they are needed. An overview over the function of the different annotations used are given in table 5.4 below.

@Repository	Class used as data access class
@PersistenceContext	Creates the <code>EntityManager</code> object when needed. Created by the <code>EntityManagerFactory</code>
@Autowired	<code>Spring</code> automatically tries to "wire" all objects that are annotated like this. The <code>Spring</code> container handles creation of the object itself

Table 5.4: Data access annotations

5.3.3 Service classes

The different service classes implemented for the Education+ application is shown in table 5.5.

Contains	
Admin	AdminService.java AdminServiceImpl.java
User	UserService.java UserServiceImpl.java
School	SchoolService.java SchoolServiceImpl.java
Course	CourseService.java CourseServiceImpl.java
News	NewsService.java NewsServiceImpl.java

Table 5.5: Model objects located in no.ntnu.jf.service package

The no.ntnu.jf.service package acts as the service layer in the application. The service layers job is to invoke the DAO objects and to perform the task wanted. This layer works as a "middle-layer" and is helping abstracting away the DAO classes. The CourseService class implemented is given in listing 5.3 as an example below.

```

@Service
public class CourseServiceImpl implements CourseService {

    @Autowired
    private CourseDAO courseDAO;

    @Transactional
    public Course saveCourse(Course c) {
        return courseDAO.saveCourse(c);
    }

    @Transactional
    public boolean removeCourse(Course c) {
        return courseDAO.removeCourse(c);
    }

    public List<Course> findAllCourses() {
        return courseDAO.findAllCourses();
    }

    public List<Course> findAllCoursesForSchool(String shortName) {
        return courseDAO.findAllCoursesForSchool(shortName);
    }

    public List<User> findAllUsersInCourse(String courseId) {
        return courseDAO.findAllUsersInCourse(courseId);
    }
}

```

```

    }
}

```

Listing 5.3: Snippet of `no.ntnu.jf.service.CourseServiceImpl.java` object implementation

An overview over the function of the different annotations used are given in table 5.6.

@Service	Class used as a service class. A service class is seen as a facade for the business logic
@Transactional	Any method annotated with <code>@Transactional</code> means that this method will perform persisting of a Java object to the database. A common pitfall is to forget this annotation which will result in that no new database entities will appear in the actual database.

Table 5.6: Service layer annotations

5.3.4 Controller classes

The different controller classes implemented for the Education+ application is shown in table 5.7.

Contains	
Admin	<code>AdminController.java</code>
User	<code>UserController.java</code>
School	<code>SchoolController.java</code>
Course	<code>CourseController.java</code>
News	<code>NewsController.java</code>

Table 5.7: Model objects located in `no.ntnu.jf.controller` package

The `no.ntnu.jf.controller` sub package contains the files that actually exposes all the web services through a public API. How this works is that the Java class itself is annotated with `@Controller`, which tells the dispatcher servlet that this class is not implementing a specific controller interface, but that it is rather using different annotations to express request mappings for specific handler methods. The `CourseController` class is shown in listing 5.4 below.

```

@Controller
public class CourseController {
    @Autowired
    private CourseService courseService;
    @Autowired
    private AdminService adminService;
    @Autowired
    private SchoolService schoolService;
    private SpringLoginService loginService = new SpringLoginService();

    @RequestMapping(value = "/navigation/course", method = RequestMethod.GET)
    public ModelAndView listCourses(HttpServletRequest request) {
        ModelAndView mv = new ModelAndView();
        mv.setViewName("courses");
        mv.addObject("courseList", courseService.findAllCourses());
        return mv;
    }

    @RequestMapping(value = "/course", method = RequestMethod.POST)
    @ResponseBody
    public Course saveCourse(@RequestBody Course course) {
        courseService.saveCourse(course);
        return course;
    }

    @RequestMapping(value = "/course", method = RequestMethod.DELETE)
    @ResponseBody
    public Course deleteCourse(@RequestBody Course course) {
        courseService.removeCourse(course);
        return course;
    }

    @RequestMapping(value = "/navigation/addOrEditCourse", method = RequestMethod.
        GET)
    public ModelAndView addOrEditCourse(HttpServletRequest request) {
        ModelAndView mv = new ModelAndView();
        mv.setViewName("addOrEditCourse");

        UserDetails user = loginService.currentUserDetails();
        Admin admin = (Admin)adminService.getAdminByEmail(user.getUsername());
        School school = schoolService.getSchoolByShortName(admin.getSchool().
            getSchoolShortName());

        mv.addObject("admin", admin);
        mv.addObject("school", school);

        return mv;
    }
}

```

Listing 5.4: Snippet of no.ntnu.jf.controller.CourseController.java object implementation

A few new annotations are introduced in the controller classes that tells the dispatcher servlet what kind of controller that will be used, and what kind of datatype that is expected from them. These are explained further in table 5.8.

@Controller	Tells the dispatcher servlet that this class acts as a controller. Indicates that the classes methods should be scanned for request mappings
@RequestMapping	Given at method level. Each mapping binds to a specific HTTP path within the containing dispatcher servlet. This is where the type of HTTP request also is set
@ResponseBody	Instructs Spring MVC to serialize the return type object to the client. In our case, the course object gets automatically serialized to JSON because the client accepts that content type
@RequestBody	Instructs Spring MVC to map the body of the HTTP request to the specified object. In our case, the course object gets mapped to JSON because the client set the request Content-Type to application/json

Table 5.8: Service layer annotations

It is in the controller classes where most of the measures are taken to make the web services comply to a RESTful standard. The `@RequestMapping` will bind the same HTTP path to several web services, and the HTTP request type is what decides if an object should be fetched or saved from the web server. The Education+ web server has two different RESTful interfaces. One is for all the web services used for views and are prefixed `/navigation`, which will show actual data returned by a web service in the web browser, and the other is all the web services that either persists, deletes or updates entities in the database. The web server is implemented with two RESTful interfaces to make it easier to apply security to the application.

5.3.5 Spring configuration

Spring configuration is seen as a separate task. The five different XML files contains information about how the dispatcher servlet, security, database connectivity and third party dependencies are configured. Most of the xml configuration files are located inside the WEB-INF folder. The WEB-INF folder works as a private area for the web application so that any files under the WEB-INF directory cannot be accessed directly from the browser. The WEB-INF directory is accessible by the classes within the application so that files and classes can communicate.

Web.xml

The web.xml file implemented is shown in listing 5.5.

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app>
  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</
      listener-class>
    </listener>

    <servlet>
      <servlet-name>appServlet</servlet-name>
      <servlet-class>org.springframework.web.servlet.DispatcherServlet</
        servlet-class>
      <init-param>
        <param-name>contextConfigLocation</param-name>
      </init-param>
      <load-on-startup>1</load-on-startup>
    </servlet>

    <context-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>
        /WEB-INF/spring/appServlet/servlet-context.xml
        /WEB-INF/spring/root-context.xml
        /WEB-INF/security-context.xml
      </param-value>
    </context-param>

    <filter>
      <filter-name>springSecurityFilterChain</filter-name>
      <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-
        class>
    </filter>

    <filter-mapping>
      <filter-name>springSecurityFilterChain</filter-name>
      <url-pattern>/*</url-pattern>
    </filter-mapping>

    <servlet-mapping>
      <servlet-name>appServlet</servlet-name>
      <url-pattern>/</url-pattern>
      <url-pattern>*.html</url-pattern>
      <url-pattern>*.json</url-pattern>
      <url-pattern>*.xml</url-pattern>
    </servlet-mapping>

    <persistence-unit-ref>
      <persistence-unit-ref-name>persistence/application</persistence-unit-ref-
        name>
      <persistence-unit-name>application</persistence-unit-name>
    </persistence-unit-ref>
  </web-app>

```

Listing 5.5: web.xml file

All incoming requests flow through a `DispatcherServlet`. The Java EE container is told to load this servlet at the web applications startup in the `web.xml` file. The `DispatcherServlet` is defined at the top, inside a `servlet` element. It is the `DispatcherServlet`'s responsibility to load the Spring Application Context that is used to perform wiring and dependency injection. To do this, a set of initialization parameters are specified to the `DispatcherServlet` that configures the Application Context. Some bullet points will help to clarify the job of the `web.xml` file[20]:

- Register the `DispatcherServlet` as a Servlet called `appServlet`
- Map this servlet to handle incoming requests (relative to the app path) starting with `"/`
- Use the `ContextConfigLocation` initialization parameter to customize the location for the base configuration XML files that is loaded by the `DispatcherServlet`

- A filter and its FilterMapping is declared to be able to use the Spring Security framework. Again to handle different paths, just like the servlet mapping
- A persistence unit reference is added to be able to persist objects to a database

servlet-context.xml

The Spring Application Context is in the project called servlet-context.xml. This files job is mainly to wire up objects, and configuration with regards to dependency injection. To be able to use the @Autowire annotation in the Controller classes, these classes needs to be defined as Java Beans, so that the EntityManagerFactory can handle them, as shown in listing 5.6.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/mvc">
  <annotation-driven/>
  <mvc:annotation-driven />
  <context:component-scan base-package="no.ntnu.jf.controller"/>
  <mvc:resources location="/resources/" mapping="/resources/**"/>
  <beans:bean class="org.springframework.beans.factory.annotation.
    AutowiredAnnotationBeanPostProcessor"/>
  <beans:bean class="org.springframework.web.servlet.view.
    InternalResourceViewResolver">
    <beans:property name="prefix" value="/WEB-INF/views/" />
    <beans:property name="suffix" value=".jsp" />
  </beans:bean>
  <beans:bean id="transactionManager" class="org.springframework.orm.jpa.
    JpaTransactionManager">
    <beans:property name="entityManagerFactory" ref="entityManagerFactory" />
  </beans:bean>
  <beans:bean id="datasource" class="org.springframework.jdbc.datasource.
    DriverManagerDataSource">
    <beans:property name="driverClassName" value="com.mysql.jdbc.Driver"/>
    <beans:property name="url" value="jdbc:mysql://mysql.stud.ntnu.no"/>
    <beans:property name="username" value="jonfr_master"/>
    <beans:property name="password" value="lokeper2"/>
  </beans:bean>
  <jee:jndi-lookup id="entityManagerFactory" jndi-name="persistence/application"
    />
  <tx:jta-transaction-manager/>
  <context:annotation-config />
  <tx:annotation-driven />
  <context:load-time-weaver />
  <beans:bean id="userDAO" class="no.ntnu.jf.dao.UserDAOImpl"/>
  <beans:bean id="userService" class="no.ntnu.jf.service.UserServiceImpl"/>
  <beans:bean id="schoolDAO" class="no.ntnu.jf.dao.SchoolDAOImpl"/>
  <beans:bean id="schoolService" class="no.ntnu.jf.service.SchoolServiceImpl"/>
  <beans:bean id="adminDAO" class="no.ntnu.jf.dao.AdminDAOImpl" />
  <beans:bean id="adminService" class="no.ntnu.jf.service.AdminServiceImpl" />
  <beans:bean id="courseDAO" class="no.ntnu.jf.dao.CourseDAOImpl" />
  <beans:bean id="courseService" class="no.ntnu.jf.service.CourseServiceImpl" />
  <beans:bean id="customUserDetailsService" class="no.ntnu.jf.service.
    SpringLoginService" />
</beans:beans>
```



```
</beans:beans>
```

Listing 5.6: servlet-context.xml file

To clarify what is done in this xml file, some bullet points are presented[20]:

- The `<annotation-driven>` element ensures that Spring MVC is setup with support for routing requests done to the `@Controller` classes. It also handles conversion, formatting and validation between objects
- The `<component-scan>` element ensures that the Spring container performs component scanning, so that any class with `@Controller` annotations will be automatically discovered
- The `<mvc:resources>` element makes sure that files that are located inside the `/resources` folder in our structure can be referenced directly in the jsp and html files. This needs to be done to refer to stylesheets, images and other resources that are located outside the `WEB-INF` directory
- To be able to use the `@Autowired` annotation, a spring autowire bean needs to be declared
- The `InternalResourceViewResolver` handles mapping to the view files that are located inside the `WEB-INF` directory
- The `TransactionManager` bean is the bean that handles all the transactions towards the database. Used by the `EntityManager` and `JPA` in general
- The `datasource` bean handles the database configuration
- All the classes that use the `@Autowired` annotation when declaring instance variables needs to be referred to as a Java Bean so that the `EntityManager` can work with them.

Security-context.xml

The `security.context.xml` is where all the security configuration are done. Spring handles all the technical tasks concerning security, and the only thing needed to do is to tell Spring what URLs that needs to be secured, and what kind of authentication is needed. The `security-context.xml` file is shown in listing 5.7 below:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/security">
  <global-method-security pre-post-annotations="enabled" />
  <http use-expressions="true">
    <intercept-url pattern="/resources/**" access="permitAll"/>
    <intercept-url pattern="/navigation/**" access="hasRole('ROLE_ADMIN')"/>
    <intercept-url pattern="/welcome" access="hasRole('ROLE_ADMIN')"/>
    <intercept-url pattern="/static/**" access="permitAll"/>
    <intercept-url pattern="/course" access="permitAll"/>
    <form-login login-page="/login" default-target-url="/welcome" authentication
      -failure-url="/" />
    <logout />
  </http>
  <authentication-manager>
    <authentication-provider user-service-ref="customUserDetailsService"
      >
      <password-encoder hash="sha" />
    </authentication-provider>
  </authentication-manager>
</beans:beans>
```

```
</beans:beans>
```

Listing 5.7: security-context.xml

- First the `<global-method-security>` elements pre-post-annotation variable is enabled. This is done so that methods can be secured at Java level with annotations.
- The `<html>` element is considered self explanatory, a specific URL path and access level is given. Wildcards can be used, so `/navigation/**` means every URL that starts with `/navigation` is secured. Spring Security also handles logging in and out of a web application by just declaring the `<form-login>` and `<logout>` elements.
- The `<authentication-manager>` element refers to a Java implementation that sets the actual roles of the objects that are logging into the web application. It also tells Spring Security that log in credentials should be compared with entities in a database, and that passwords are hashed with the SHA-1 algorithm.

For the authentication manager to be able to use a database for a user lookup, a login service class needs to be created that implements Springs `UserDetailsService` class. This way, authentication roles can be set to each object that is trying to login, and can easily check if users or administrators exists as entities in the database. The `UserDetail` method in this project looks like this:

```
public UserDetails loadUserByUsername(String email) throws
    UsernameNotFoundException, DataAccessException {
    UserDetails userDetails = null;
    List<GrantedAuthority> authList = new ArrayList<GrantedAuthority>(1);
    authList.add(new GrantedAuthorityImpl("ROLE_ADMIN"));
    try {
        List<AdminInterface> adminList = checkIfAdminsExist();
        if(adminList.isEmpty()) {
            AdminInterface standardAdmin = (AdminInterface)adminService.
                getAdminByEmail(email);
            if(standardAdmin != null) {
                userDetails = new User(standardAdmin.getEmail(), standardAdmin.
                    getPassword(), true, true, true, true, authList);
            }
        } else {
            AdminInterface admin = (AdminInterface)adminService.getAdminByEmail(email)
                ;
            userDetails = new User(admin.getEmail(), admin.getPassword(), true, true,
                true, true, authList);
        }
    } catch(Exception e) {
        e.printStackTrace();
    }
    return userDetails;
}
```

Listing 5.8: method from `no.ntnu.jf.service.SpringLoginService.java` class

The method first creates a `GrantedAuthority` list which contains `ROLE_ADMIN`. This is the role that can referred to in the `security-context.xml`. The method returns a `UserDetail` object that is an admin object with a `GrantedAuthority` list linked to it.

Persistence.xml

For JPA to work, a persistence unit needs to be defined. This is defined in META-INF/persistence.xml as shown in listing 5.9. This makes Spring search java classes for the @Entity annotation so that these can be made into database entities.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence" version="2.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://
    java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
  >

  <!-- Glassfish server -->
  <persistence-unit name="application" transaction-type="JTA">
    <jta-data-source>jdbc/mysql</jta-data-source>

    <!-- To autcreate tables at deployment - drop-and-create-tables can also be
      used -->
    <properties>
      <property name="eclipseLink.ddl-generation" value="create-tables" />
    </properties>
  </persistence-unit>
</persistence>
```

Listing 5.9: persistence.xml

The datasource gets set to a JDBC mysql version which complies to the JDBC pool that is configured at the glassfish server. A property is also set that automatically tries to create all tables in the database every time the application is deployed to the server. This is done so that it is certain that all Java classes annotated with @Entity will have a corresponding database table linked to it when the application is deployed.

5.3.6 WEB-INF/views

Every jsp file are located in the view folder, which is a sub folder of the WEB-INF directory. Every file that is a child of the WEB-INF directory will be hidden from any clients browser. It is the DispatcherServlet job to invoke the InternalResourceViewResolver so that these view files are dispatched correctly so they become visible for the user. A JSP file is a Java Server Page file, and is a server side language made for dynamic webpages written in Java. When using JSP, Java code can be used directly in a HTML file. But to comply the MVC pattern to its fullest, this is not good programming practice. Every line of code that is inside the <%@ tags, are considered JSP code, and the only time these tags are used is to include other JSP files. Since the views job in a MVC web application is only to show data to the end user, it is the controllers job to delegate what data the views will need, as shown in listing 5.10.

```
@RequestMapping(value = "/navigation/addOrEditCourse", method = RequestMethod.
  GET)
public ModelAndView addOrEditCourse(HttpServletRequest request) {
  ModelAndView mv = new ModelAndView();
  mv.setViewName("addOrEditCourse");

  UserDetails user = loginService.currentUserDetails();
  Admin admin = (Admin)adminService.getAdminByEmail(user.getUsername());
```

```

    School school = schoolService.getSchoolByShortName(admin.getSchool().
        getSchoolShortName());

    mv.addObject("admin", admin);
    mv.addObject("school", school);

    return mv;
}

```

Listing 5.10: ModelAndView method used to delegate data to the views

Here both an admin and a school object are created from data fetched from a UserDetails object. The user that calls this method will now actually be an administrator with the ROLE_ADMIN authority. The ModelAndView object is created, and given a view name. This view name corresponds to a JSP file located in the view directory. As the name of the methods reveal, both models and views can be added to this object, and this is where the views gets the data needed to display the correct information. The addObject method gets called, and both an admin, and a school object are wired up to the ModelAndView object. The string that is given as the first parameter, is the name that will be used in the JSP files so that information about the model that are passed through to, can be fetched.

```

<h2>Welcome ${admin.email}</h2>
<p>Your a professor at ${school.schoolShortName}</p>

```

Listing 5.11: Example HTML code to access model objects

All public instance variables can be accessed through the familiar dot notation inside the JSP files, without mixing pure Java code into any of the view files. This way of separating the different parts of your web application is crucial to follow the principals of the MVC architectural pattern.

The administrator panel is built up by these views. Since the content at the different parts of the administrator panel is dynamic, the views needs to communicate with the database so that it can show additional information. To do this, JQuery's getJSON and postJSON are used, as shown in listing 5.12.

```

$.getJSON("/abc/getUsersAvailableForCourse/" + courseId, function(
    returnedList) {
    usersAvailableArray = returnedList;

    if(usersAvailableArray.length == 0) {
        $("#addUsersButton").hide();
        $("#notAvailable").html("No users are available for adding.");
    } else {
        $("#addUsersButton").show();
        $("#notAvailable").html("");
        var userTable = document.getElementById("apTable");

        /* Create table Headers */

        $(usersAvailableArray).each(function(){
            var userTable = document.getElementById("apTable");

            /* Append rows to table */
        });
    }
});

```

```
    }  
});
```

Listing 5.12: Snippet of getJSON JQuery method

These JQuery methods use something called a javascript callback method. A callback method is a reference to executable code that is passed as an argument to other code. This happens in the first line, where a function is passed as an argument to the getJSON method. This function has its own parameter, and this parameter is where the return value of the web service call gets stored. This way it is possible to check what was returned from the web service, and perform the necessary operations given the contents of the return value.

Chapter 6

The Concepts of Programming in Objective-C

Objective-C is the primary language used to implement Mac OS X, iPhone, iPad and iPod Touch applications. It was originally created as the main language for the NeXT¹ operating system, but didn't become a big success until Apple used the language for their first Mac OS X operating system in 1996. Objective-C is as the name reveals an object oriented language. It is a thin layer on top of the C language, and moreover a strict superset of C. Because of this it is possible to compile and easily include both C and C++ source code in to Objective-C projects that is often done when building graphical applications using external frameworks as OpenGL ES² or Cocos2D³.

This chapter will give an introduction to the basic concepts of how to program in Objective-C for iOS devices. There are a lot of different elements that needs to be considered when developing for mobile devices, and Objective-C works a bit different from other programming languages. This will be an introduction to chapter 7 where a detailed overview of how the Education+ iPad application has been implemented will be given.

6.1 The four layers in iOS

The iOS operating system can be divided into four layers as illustrated in figure 6.1. The bottom layer is closest to the hardware, while the top is the layer that the end user interact with.

¹<http://en.wikipedia.org/wiki/NeXT>

²http://en.wikipedia.org/wiki/OpenGL_ES

³<http://en.wikipedia.org/wiki/Cocos2d>

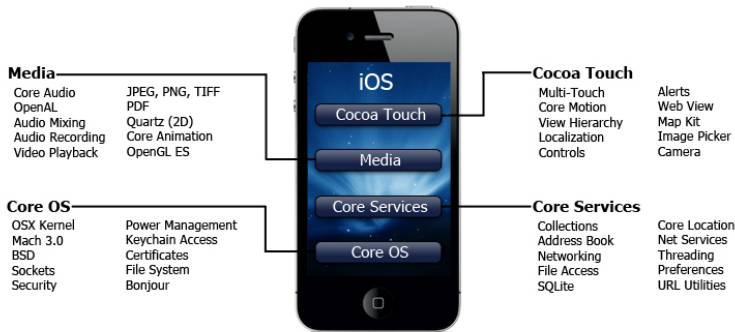


Figure 6.1: The four layers of iOS

6.1.1 Core OS

The iOS operating system is at the very bottom made from a Mach 3.0 UNIX kernel, and it derives from the Mac OS X implementation. Most of the public API to this layer is actually a C API because of the UNIX kernel, while some of these features can also be reached from the Core Service layer that will give access through an Objective-C API. The Education+ application uses the Core OS layers Keychain Access interface. This part is a widely used part of the layer, and is used for storing login credentials locally at an iOS device, and is therefore available with an Objective-C style API.

6.1.2 Core Services

The Core Services abstraction layer is a lot more object oriented than the Core OS layer. It provides a lot of the same functionality as the Core OS layer, but with an Objective-C API. A lot of the functionality from this layer is used in the Education+ application. To communicate with the Education+ web server, the network interface is used, and to store data fetched from the server locally at the iPad SQLite together with Core Data is used. Collection is where data structures such as arrays and dictionaries are located, which is widely used in any kind of iOS application. It is also the Core Services layer that provides you with the ability to program with the use of threads.

6.1.3 Media

The media layer is somewhat more vague than the other layers since any iOS device such as an iPhone or an iPad is fundamentally built for running multimedia applications. This makes it hard to pin down exactly where this belongs since multimedia code runs everywhere in iOS. Any application like iTunes that plays music, audio sent from a phone call or FaceTime uses this layer to interact with the specific part needed in the operating system.

6.1.4 Cocoa Touch

Cocoa Touch is the UI framework that is used in all of the iOS applications. Cocoa is the name of the application development environment for Mac OS 10, while it is called Cocoa Touch in the iOS environment. Everything at the screen of an iOS device is from this layer; buttons, sliders, views, alerts, navigation mechanisms alerts and so on. The cocoa touch layer is probably "the most" object oriented part of all the four layers. Not in terms of actually calling methods to access objects instance variables, but because this layer is made with a design paradigm that fundamentally uses object oriented mechanisms to make the different parts of the application able to talk to each other. The way the MVC architectural pattern is implemented into the heart of any iOS application is crucial to understand, and will be further explained in the next chapter, Design Strategies. [21]

6.2 Design Strategies

For the web server, the MVC pattern is followed because the Spring framework encourages us to do so. The way objects communicate with each other in iOS are restricted, and this is why it is crucial to get a good understanding of how this design paradigm is implemented into the platform.

This paradigm is the heart of any software development project done for the iOS platform. As mentioned before, the MVC pattern often comes in different versions in different environments. The basic idea behind the pattern is still the same as shown in table 6.1, but an introduction to how it is used specifically for the iOS environment is given to stress the importance of the role the MVC pattern plays.

Model	The model part are all the objects that defines what your application is. For example a user object in the Education+ application
Controller	The controllers job is to tell the views <i>how</i> the model should be presented. Ie. User Interface logic
View	The view is the controllers minions, and its job is solely to display User Interface components and whatever data the controller tells it to show. The views doesn't have any application specific knowledge.

Table 6.1: MVC Architectural Pattern - iOS specific

Every object in an iOS application will belong to one of the three "camps"

presented above. By designing an application this way, good object oriented programming practice will be applied. Besides from what is written above, there are a few other design principles Apple follows when implementing applications in Objective-C. Both the model and the view objects should be generic.

In the Education+ application it means that an user object, should be created in a way that this object can be used in any other application that also needs a user object, without having to rewrite any code at all. The same applies for the corresponding user view. This view should also be able to be used in any other application. The only objects that are completely application specific is the objects that belong in the "controller camp". It is the controllers job to take the model's data, and present it in a view.

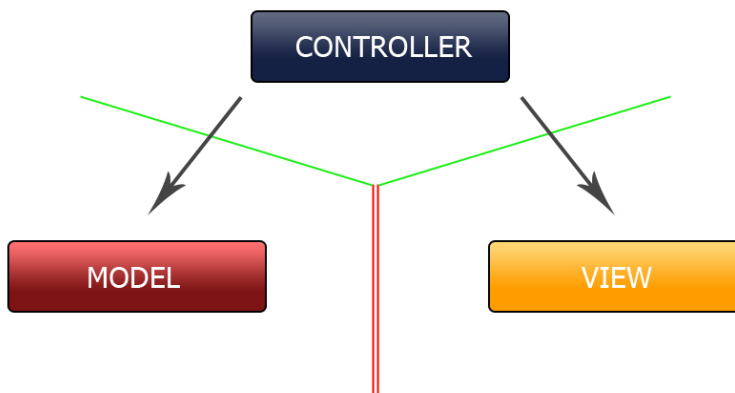


Figure 6.2: MVC - iOS specific part 1

To supplement figure 6.2 above, the controller works as the boss. The two red lines indicate that the model and the view **never** talk to each other. The key is to manage the communication between the objects in these three camps. A controller object can always talk to its model, and it knows everything there is to know about it. However, the model cannot directly communicate with its controller, and this is not the purpose either. The model objects should be reusable, and since the controller object is specifically made for each application, the model object will not be reusable if it has some kind of direct link to its controller. Because of this, the controller will know what kind of data that should be given to the view, so the controller can tell the view what to display on the screen.

Communication between the controller and the view is done through outlets as figure 6.3 outlines.

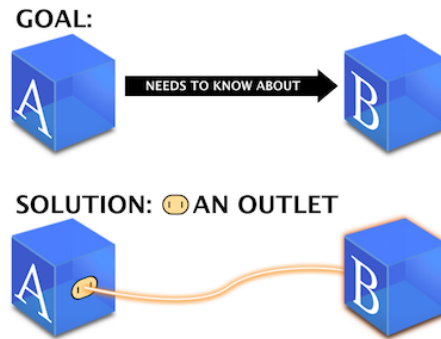


Figure 6.3: An Objective-C outlet

An example is provided of a label that should have some text updated. For the controller to know where to update the text, it needs to be connected to all the different elements that appears on the screen, and update the correct one, and this is done through outlets that are hooked up to each other in Xcode.

How the controller and the view communicates with each other is very important, and is illustrated graphically in figure 6.4.

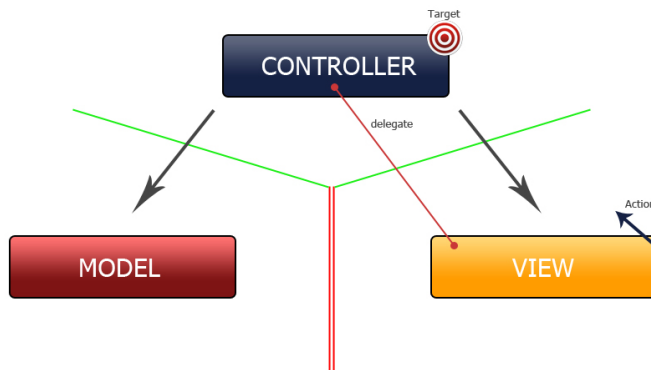


Figure 6.4: MVC - iOS specific part 2

The way this happens is a fundamental key of how objects communicates in Objective-C. The controller can only talk to the model, but the model can never directly talk to its controller. With the controller and the view its a bit different, because often the view needs to tell the controller that something happened on the screen that the controller needs to know about. Someone drags on a slider or touches a button, so the view has to be able to "talk" to the controller.

As figure 6.4 shows, there are two different ways this communication can happen, either through what is called a target/action, or through a delegate. The

first form of communication is target/action. What happens here is that from the controller object, an action is hooked up to an element that is present in a view. Lets say this element is a button, so when a button is touched in the view, the action for that button that is located in the view object, will have a corresponding target in the controller object that will tell the controller that someone touched my button- *"please handle this for me"*. The view doesn't know much about the controller so the view doesn't know what kind of action that should be performed when the button is pressed, it just sends the message, with no further concerns. This way of communicating is in Objective-C called blind communication and are used in any iOS application.

The other form for the communication from the view to the controller is called delegation, which is also a very important term to understand when developing for the iOS platform. Delegation is used in other contexts than just this, but this is a good place to explain how it works. What happens is that the controller sets itself as the views delegate through protocols, which will be explained further in the Syntax chapter. When a controller object is set to be some views delegate, it declares, without implementing, one or more methods from the given protocol, kind of like subclassing, but there is no need to override all the methods. These methods are often referred to as will, should or did methods, which means that these methods are run when the controller object wants an answer if something will happen, should happen or did happen in a view. Figure 6.5 makes it easier to understand.

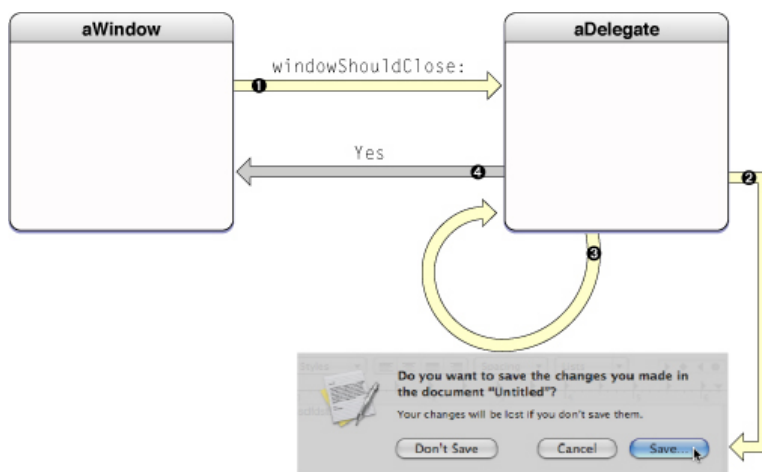


Figure 6.5: Example delegation between objects

The delegating object sends a message only if the delegate implements the method, which in this example is the `windowShouldClose:` method. From this method name, the delegating object waits for a response that might be if the close button to a new window has been pushed, should I really close the window? The

view figures out if the user wants to save or discard any recent updates from that view, and tells its controller, which is its delegate, either yes or no. In this example it tells its controller, yes, close this view, which is me, but first be sure to save all the data inside me to the correct model object.[22]

One last thing to also mentioned earlier is that the model cannot be directly linked to its controller due to the idea of all the model objects should be completely reusable. Most of the time, a model objects data will be updated because some user performed some action in a view. When this is the case, how the MVC pattern works so that the model objects data will be updated should be clear. But lets think of a scenario of an application where multiple users access and use the same model objects, so that another user might have updated or changed some data in a model object. Since the model object can't communicate directly to its controller, it has no way of telling the controller that something has changed, and the controller wont know that it actually should update some view. This is solved with notifications. Notifications can be thought of as a radio station, every model is a radio station that broadcasts if something is changed. It doesn't say what has changed, it just says that something has changed. A controller object can be "tuned into" their models radio station, and this way when the controller object receives a broadcasted message, it will go and ask its model what has been changed. Keep in mind that this notification feature must not be confused with the push notification center that handles all those red bubbles that appears at an applications icon at the iOS device home screen, these are two completely different things. This is extremely important to understand completely to be able to develop applications for the iOS platform. When learning Objective-C, a lot of new things is introduced and can easily be very overwhelming. It takes some time to understand all the different concepts and to get over the first "bump", but as they are, things start to flow more naturally, and the learning curve starts to get steeper.

6.3 Syntax

People that are familiar with programming in C or C++ will draw a lot of similarities in terms of the Objective-C syntax. There are some differences that are worth mentioning to both people that are new to the C family language, and those who aren't, and these will be highlighted in the next sections.

6.3.1 Header and Message files

A main difference from languages that most software developers know about, like Java, is that an interface of a class and the actual implementation has to be declared in two separate code blocks. By convention, the interface and implementation are always separated into two different files. The interface is declared in a header file, and are suffixed with `.h`. The implementation of the interface is implemented in a message file, suffixed with `.m`. Below is an example of a dummy header file implemented in Objective-C.

```

@interface classname : superclassname {
    // instance variables
}

@property (nonatomic) double someValue;
@property (strong, nonatomic) someOtherClass *someClass;
@property (weak, nonatomic) IBOutlet UILabel *someLabel;

+ classMethod1;
+ (return_type)classMethod2;
+ (return_type)classMethod3:(param1_type)param1_varName;

- (return_type)instanceMethod1 : (param1_type)param1_varName : (param2_type)param2_varName;
- (return_type)instanceMethod2 : (param1_type)param1_varName andOtherParameter:(param2_type)
    param2_varName;
@end

```

Listing 6.1: Dummy implementation of an Objective-C header file

Worth mentioning here is the difference between the methods that are prefixed with + and -, and also the @property declarations. The methods prefixed with a + sign are class methods, and can be called on the class itself, while the methods prefixed with the - sign are instance methods, which has to be called on a particular instance of the class.

The @property declarations, can be thought of as being an instance variable that also automatically declares the two accessor methods get and set. For most properties some property declaration attributes are given, which tells the compiler something about the property. The different property declaration attributes are as follows in table 6.2, 6.3 and 6.4[23]:

	Setter Semantics
strong	Specifies that there is a strong (owning) relationship to the destination object.
weak	Specifies that there is a weak (non-owning) relationship to the destination object. If the destination object is deallocated, the property value is automatically set to nil.
copy	Specifies that a copy of the object should be used for assignment.
assign	Specifies that the setter uses simple assignment. This attribute is the default. You use this attribute for scalar types such as NSInteger and CGRect.
retain	Specifies that retain should be invoked on the object upon assignment.

Table 6.2: Property declaration attributes - setter semantics

Writability	
readwrite	Indicates that the property should be treated as read/write. This attribute is the default. Both a getter and setter method are required in the @implementation block. If you use the @synthesize directive in the implementation block, the getter and setter methods are synthesized.
readonly	Indicates that the property is read-only. If you specify readonly, only a getter method is required in the @implementation block. If you use the @synthesize directive in the implementation block, only the getter method is synthesized. Moreover, if you attempt to assign a value using the dot syntax, you get a compiler error.

Table 6.3: Property declaration attributes - writability

Atomicity	
nonatomic	Specifies that accessors are nonatomic, this means that its setter and getter are not thread-safe. <i>By default, accessors are atomic.</i>

Table 6.4: Property declaration attributes - atomicity

Most of the time the setter semantic strong or weak are used. However, sometimes it is important to think about the different options available, so they are important to mention. Strong means that the memory used for the object will stay around for as long as needed, while weak means that as soon as there is no pointer pointing to the object memory location in the heap, it will be released. These types of measures are done because developing for a mobile device requires the programmer to use the limited resources available in a good way, so objects will be kept for as little time as possible. Garbage collection and the actual job of releasing the different objects from memory are done automatically by ARC⁴, which was an added feature with iOS 5.

The last thing to mention about the header file is that every method or variable that is declared will be publicly available through that classes public API. If methods or instance variables needs to be private to the class, the correct convention is to declare them inside an interface at the top of the .m file. The .m⁵ file where the

⁴Automatic Reference Counting: - <https://developer.apple.com/technologies/ios5/>

⁵The .m suffix stands for messages

interface is implemented looks like this:

```
#import "classname.h"

@interface classname()
- (void)foo:(int)bar;
@end

@implementation classname

@synthesize someValue = _someValue;
@synthesize someLabel = _someLabel;
@synthesize someClass = _someClass;

+ (return_type)classMethod {
    // implementation
}
- (return_type)instanceMethod {
    // implementation
}

// This method will be private
- (void)foo:(int) {
    // implementation
}

@end
```

Listing 6.2: Dummy implementation of an Objective-C message file

From listing 6.2 the `@synthesize` variables is worth mentioning. `@synthesize` does all the work of creating setters and getters in correspondence to the properties that is declared in the header file. The synthesized variable is declared to be equal to the same name as the instance variable, just with an underscore prefix. This is done to give the memory storage location of the property a name, which is a common naming convention when synthesizing variables. An important thing to remember, which is a very common pitfall, is that when synthesizing an instance variable, storage is not allocated for the object that the pointer points to, it just allocates room for the pointer. All objects created in Objective-C are always allocated on the heap, and because of this they are always accessed through a pointer. This might sound complex, but the only thing to remember is to actually allocate and initialize the objects themselves at some other time before you use them. In Java you would create a new object like listed below:

```
Foo someObject = new Foo();
```

Listing 6.3: Example creation of Java object

While in Objective-C you have to allocate and initialize an object like in listing 6.4:

```
[[someClass alloc] init];
```

Listing 6.4: Example creation of Objective-C object

As we are used to in languages like Java or C++, methods and its parameters are created with the following syntax:


```
public int squareRoot(int i) {
    return Math.SquareRoot(i);
}
```

Listing 6.5: Example Java method

Parameters are passed inside parentheses, and dot notations are used to perform method or function calls. In Objective-C it is done in a different way, as listing 6.6 shows:

```
- (int)calculateSquareRoot : (int)i {
    return [self squareRoot:i];
}
```

Listing 6.6: Example Objective-C method

The type of the parameter is annotated inside a set of parentheses while the parameter name is written after it. Objective-C uses dot notation as well, but are only used when calling getters or setters. For every other call to some method or function, it is called inside a set of square brackets. The return statement in the calculateSquareRoot instance method tells us to execute the method squareRoot, which is another instance method in the same class, since the method is called with the *self* keyword, which is equivalent to *this* in Java[24].

6.3.2 Protocols and delegates

A protocol is exactly similar to in implementation as an @interface, which is what is defined in the header files, except someone else does the implementing. The primary use of protocols in iOS is through delegates, so these two are naturally combined. Even though specific iOS classes hasn't been talked about yet, a shallow implementation of something with an example of how a protocol works will be introduced. When a view is created in Xcode, that view will automatically be a subclass of some preexisting Objective-C class. A view that has a search field in it where some arbitrary data can be searched for, can have a header file implemented as follows:

```
@interface SearchViewController : UIViewController<UISearchBarDelegate>
@property (nonatomic, retain) NSMutableArray *searchResults;
@property (strong, nonatomic) IBOutlet UITableView *tableView;
@property (weak, nonatomic) IBOutlet UISearchBar *searchDisplayController;
@end
```

Listing 6.7: SearchViewController.h source file

Here it is shown that the view that is called SearchViewController is a subclass of a UIViewController class. This makes the class inherit different necessary features from the view controller class. Whats inside the angle brackets is where it is stated that the SearchViewController class, will be the UISearchBars delegate. The UISearchBar object is just a textfield where the user can enter some text. If there is going to be searched for entries stored in a database, it would be nice for the controller class to know when a user starts and stops typing in the text

field. The methods for achieving this are what's implemented in a protocol. Since this protocol is already implemented in the standard iOS `UISearchBar` class, it can easily be seen what methods the protocol are able to provide us through the documentation:

Tasks

Editing Text

```
- searchBar:textDidChange:
- searchBar:shouldChangeTextInRange:replacementText:
- searchBarShouldBeginEditing:
- searchBarTextDidBeginEditing:
- searchBarShouldEndEditing:
- searchBarTextDidEndEditing:
```

Figure 6.6: Some methods implemented by the `UISearchBar` protocol

In the `SearchViewController.m` implementation file, these methods can be overridden as shown in listing 6.8:

```
- (void)searchBarTextDidBeginEditing:(UISearchBar *)searchBar {
    self.tableView.hidden = NO;
}

- (void)searchBarTextDidEndEditing:(UISearchBar *)searchBar {
    self.tableView.hidden = YES;
}
```

Listing 6.8: Example implementation of an optional protocol method

These two methods say that when someone starts typing in the search bar, set the `tableView` property `hidden` to `NO` so that the results that corresponds to what is typed will be shown in this table view. The other method does the opposite, it hides the `tableView` when a user stops typing, and the search bar becomes inactive. Since it is the controller's job to search for what's typed into a search field and then display it back to the view, this is a great example of how the controller and the view works together by using the MVC pattern, and how protocols and delegation helps to separate the code into the file where it belongs.

Chapter 7

Implementation - The iPad application

Implementing applications natively for the iOS platform has become more and more popular over the last couple of years. Developing natively gives complete access to the devices full API, and there are no limitations. The way Apple has made it so easy to distribute an application through the App Store has made the whole iOS application industry to take off. Since the App Store opened on July 10th 2008, there has been downloaded over 25 billion applications, which describes how big the scope of the application industry has become. When an application has been released into the App Store, the application is available for download to any iOS device user world wide. The market that is reached with little to none advertising or marketing costs are outstanding compared to other solutions of software distribution.

This chapter will go into detail about how to develop an iOS application with Objective-C. The different features of the Xcode IDE¹ will be presented as well as what should be thought how when developing mobile applications for the iOS platform.

7.1 Architecture

The architecture of the application is following the MVC architectural pattern, which separates all classes into the three categories model, view and controller. The structure of the Education+ workspace is given in table 7.1.

¹A suite of software development tools developed by Apple for developing software for OS X and iOS <https://developer.apple.com/technologies/tools/>

Classes	
MainStoryboard	This is where all the views are created. The storyboard is a new function in the Xcode IDE that makes it possible to create and connect all views in a drag and drop visual environment
Images	Images used in the application, such as background images and application icons are stored here
Webservices	These classes are the ones that communicates with the web server through web service calls
View Controllers	All the views created in the storyboard has a corresponding controller class.
Core Data	This folder contains both the SQLite data model as well as an implementation of the database entities that makes them the model objects
Supporting Files	Auto generated files created by Xcode when creating a new project. Contains the main method as well as some configuration files.
Frameworks	All additional frameworks needed for things like security, core data etc. are located here.
Rest of classes	The two classes that are not located inside a folder are the KeychainItemWrapper.h/m and EducationPlusLib.h/m. The KeychainItemWrapper class is copied from Apples documentation to be able to store login credentials locally in the keychain. The EducationPlusLib is a helper class that contains class methods that are used several places in the project

Table 7.1: Folder contents overview

When developing in Xcode, there are a lot of great features available that are important for developers to utilize. The following sections will give an introduction to how to use these features, and how they were used in the Education+ application.

7.1.1 The Views

The first thing a developer is presented with when creating a new project in Xcode, are the views. All the applications views are located inside a storyboard, as illustrated in figure 7.1.

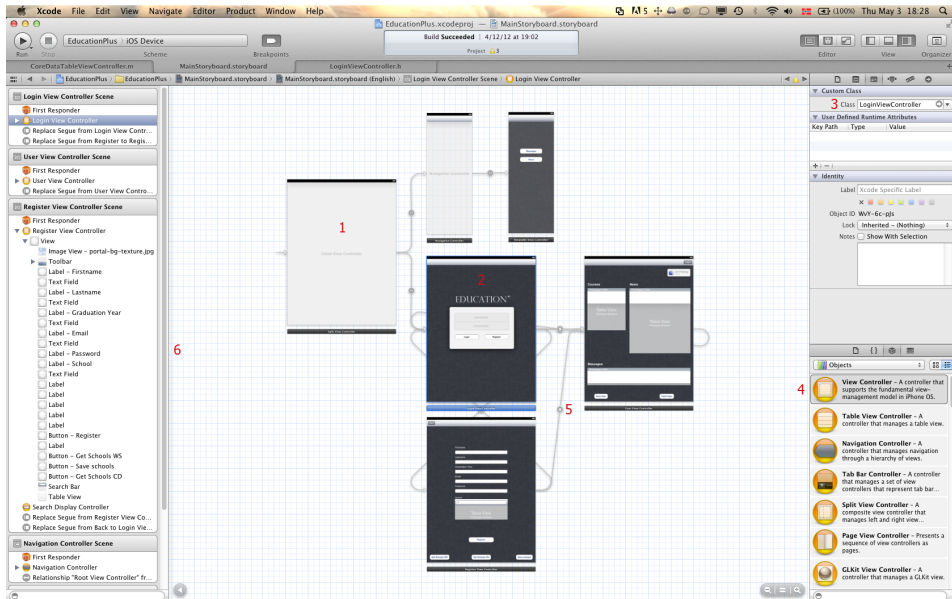


Figure 7.1: The Xcode storyboard

The storyboard is a great feature that makes it easy to create all the different views needed for an application. All the views are created without any code, and they are connected to each other by the lines that goes between the views- these are called segues. By creating all the views and connections first, it will automatically give knowledge about the application, and it helps any developer to think about what classes should be created for the different views.

The red numbers shown in figure 7.1 will be presented in the next subsections to show how to utilize the storyboard to its fullest.

1 - The initial view controller

In most cases, one of the view controllers will be set to be the initial view controller. This view controller will be the root view of the application, and will both be the view that decides what should be the start view, as well as it decides what type of view controller the application will use. The correct choice of view controller depends on what kind of application is created, and there are a number of different choices, as shown in table 7.2. However, the choice of root view controller isn't

always easy to change during implementation, so it is important to decide what type to use at an early stage.

View Controllers	
Split View Controller	A composite view controller that manages left and right view controllers. The Split View Controller splits the view into two pieces, the master and the detail view, where the master view is used as the menu, and the detail view are used as the presentation for a view. The split view controller's view should always be installed as the root view of your application window. You should never present a split view inside of a navigation or tab bar interface, which will make your application crash
Navigation Controller	Manages a stack of view controllers, each of which represents information about a view, such as its title and the navigation item associated with the view. When view controllers are pushed onto and popped off the stack, the navigation controller updates the navigation bar and view appropriately. The Navigation Controller automatically creates the navigation bar at the top and keeps track of going back to the previous views
Tab Bar Controller	Manages a set of view controllers, each of which represents a tab bar item. Each view controller provides information about its tab bar item and supplies the view to be displayed when the item is selected. Applications that use the Tab Bar Controller as their root view will have the menu of the application always presented to you as different tabs at the bottom of the screen
Table View Controller	Manages a UITableView, automatically creating an instance with the correct dimensions and resizing mask, and acting as the table view's delegate and data source. The UITableViewController class also provides toggling of editing modes. Applications like the Mail or Music application uses only table views to navigate back and forth
Page View Controller	Presents a sequence of view controllers as pages, via coordination with a data source and delegate. Swipe navigation between the pages is automatically handled with a page curl transition that tracks the user's finger. The navigation orientation can be horizontal, like pages in a book, or vertical, like pages in a wall calendar.

Table 7.2: The different view controller options [5]

These are all different types of view controllers that will give different choices for how to navigate through the different views, as well as automatically adding different menu and/or navigation features. For the Education+ application, the Split View Controller are chosen. For the purpose of the prototype, there is no need to actually use a menu, since the information presented in the application can be done in one single view. However, since it is mandatory to have a Split View Controller as a root view, and use this view controller style throughout the application, it is used so that when someone wants to add additional features to the application, the possibility of using a menu will be built in to the core of the application, as illustrated in figure 7.2.

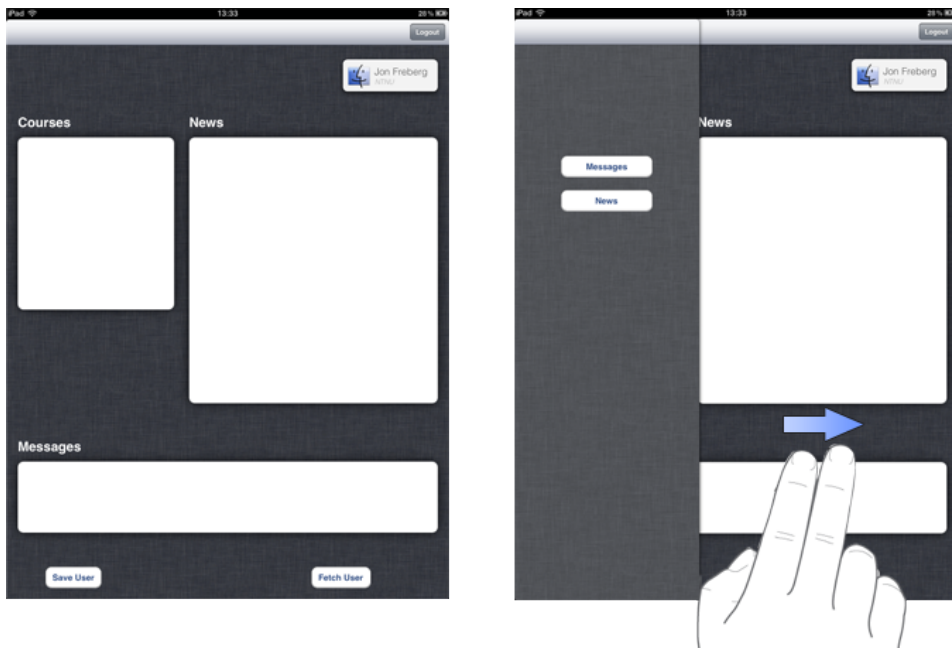


Figure 7.2: Education+ with swipe-enabled menu

2 - A view

To create a new view, drag a view controller from the object palette, and onto the storyboard canvas. If a normal view controller is dragged out, it view will automatically become a generic view controller. Any view needs to be a subclass of some implemented controller class, it is important to remember to set its views class to the controller implemented for it.

3 - Setting the views class

In figure 7.1 the class of the view is a LoginViewController class. Any class created that is a subclass of the UIView class will appear here so it can find any custom

controller implementation. It is very important to remember to go back to the storyboard and set all the views classes when they are created, since this is how controllers are connected to its views.

4 - The object palette

The object palette is where all the different objects to use in a storyboard is found. As seen in figure 7.1, some different view controllers are shown. All the different elements used in an application will be found in the object palette, as shown in figure 7.3:

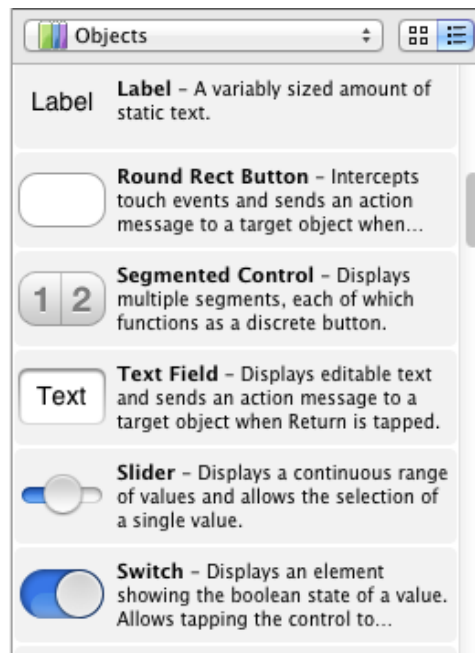


Figure 7.3: Different objects from the object palette

5 - Segues

A segue represents a triggered transition that brings a new view controller into the applications user interface, and it also instantiates the new view. A segue contains a lot of information, that are not going to go be discussed in detail now. However, a segue is created in the storyboard as follows:



Figure 7.4: A cutout from the storyboard where a segue is created

If a button should take the user to another view, control-drag with the mouse from the button, and to the view that should appear for the user. There are a couple of different segueing options, that are used for the different types of view controllers an application can use. The Education+ segues needs to be a replace segue since its root view controller is Split View Controller. This makes it possible to handle the master (menu) and the detail (main view) views separately.

6 - The view controller scenes

The View Controller Scene is an overview where all the objects and segues that are inside the different views are listed in an hierarchical view. Sometimes some elements on the screen needs to be hidden until some event occurs, and these elements will actually be hidden in the storyboard as well, so this is a great place to keep track of every object in a view.

7.1.2 The Controllers

In the controllers folder are all the controller classes that are implemented to handle the different views. As shown in the figure 7.1 in the last chapter, there are three views; the login view, the register view, and the welcome view. This means that there will be three view controllers as well, the LoginViewController.h/m, the RegisterViewController.h/m and the UserViewController.h/m. The controllers communicates with their views through outlets, which are created as shown in listing 7.1

```
// LoginViewController.h
// EducationPlus
//
// Created by Jon Freberg on 3/7/12.
// Copyright (c) 2012 FrebergWeb. All rights reserved.

#import <UIKit/UIKit.h>
#import "SplitViewBarButtonItemPresenter.h"
#import "CoreDataSingleton.h"

@interface LoginViewController : UIViewController <SplitViewBarButtonItemPresenter> {
    NSManagedObjectContext *managedObjectContext;
}

@property (strong, nonatomic) IBOutlet UIActivityIndicatorView *activityIndicator;
@property (weak, nonatomic) IBOutlet UITextField *UserNameTextField;
@property (weak, nonatomic) IBOutlet UITextField *PasswordTextField;
@property (weak, nonatomic) IBOutlet UILabel *ResponseTextLabel;
@property (weak, nonatomic) IBOutlet UIButton *LoginButton;
@property (nonatomic, retain) NSManagedObjectContext *managedObjectContext;
```

```
- (IBAction)LogInButtonAction:(id)sender;
@end
```

Listing 7.1: LoginViewController.h - Header file

The LoginViewController is a subclass of the UIViewController, which is what was dragged out from the object palette to the storyboard to create the view. When a custom subclass of the UIViewController is created, any objects that are in the actual view can be connected to the controller so it can handle any events that should occur when objects are touched.

After the custom controller class is created, it is important to remember to go back to the storyboard to set the views class as mentioned in subsection 7.1.1, and a custom class will appear in the drop-down menu once it is created.

The outlets declared in listing 7.1 are what connects the different objects in the view, programmatically to the controller file. For every object that needs to be handled, an IBOutlet is needed. For demonstration purposes, there is implemented a target/action method that programmatically fires off a segue when someone tries to log in to the application, and it is shown how this IBAction is declared as an instance method in listing 7.2.

```
- (IBAction)LogInButtonAction:(id)sender {
    NSString *username = UserNameTextField.text;
    NSString *pass = PasswordTextField.text;
    NSArray *user = [UserWS tryLogin ...]

    if(!user) {
        ResponseTextLabel.text = @"Wrong username and/or password.";
    } else {
        [self performSegueWithIdentifier:@"LoginSegue" sender:self];
    }
}
```

Listing 7.2: Example IBAction method implementation

Now the LoginViewController object has a target at itself so that the Login View can tell the controller that someone wants to try to login. Since the UITextField objects UserNameTextField and PasswordTextField are declared as IBOutlet, the controller class will use the content of these two textfields to check with its corresponding model class if this user exists or not when someone presses the login button, and then pass the user to the correct view.

7.1.3 The Models

The application is going to store all information about the user, the courses a user are attending, as well as the news for the different courses. The models are somewhat more complicated than just to create a regular user object, because they are also going to be stored locally once they are fetched from the server. To do this, the iOS Core Data API is used, which is a part of the Core Service layer. This is a great feature for storing data locally, and it is very well integrated with making Objective-C objects into SQLite entities. To store objects locally, a Core Data data model file is created. This data model file has a UI where all the entities are created, as shown in figure 7.5.

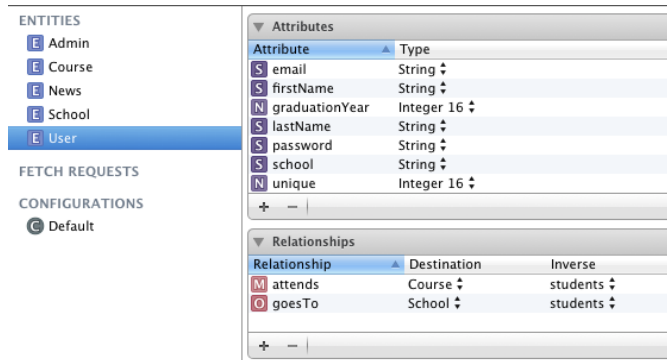


Figure 7.5: Core Data data model user interface

After the creation of an entity, attributes and relationships with other entities are given to it, if it has some. The relationships between the entities are created in a UI where all entities gets represented in a class diagram as shown in figure 7.6.

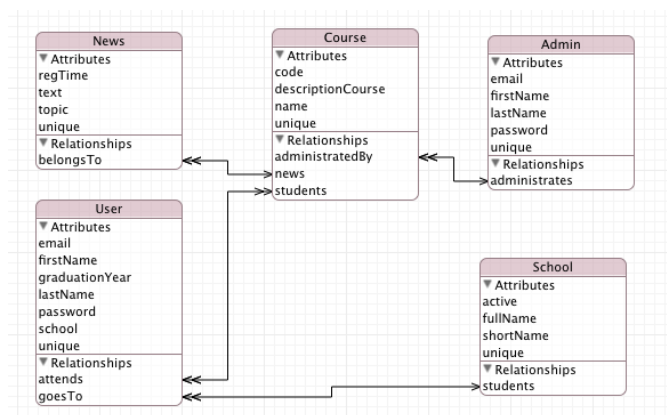


Figure 7.6: Graph view of the Education+ entities

Control-drag from one entity to another to create the relationship between them. A nice Xcode feature is how easy it is to create Objective-C objects from SQLite entities. Simply mark the entities to create objects of, go to editor, and choose "Create NSObject subclass". To be able to store something locally using core data, the objects need to be a subclass of the NSObject class, and this is done automatically by Xcode.

As good programming practice, auto generated files shouldn't be tampered with, but some additional functionality needs to be added to the different objects so that they can be created and stored at the iPad. This is where the Objective-C category comes into play.

There is very little difference between a category file, and a regular class. The only difference is that a category gets an extended name to show that this is not a complete implementation of a class. The user category is called `User+Create.h/m`, since the only functionality that needs to be added to the user class is being able to create objects. Below is the implementation of the category created on the user class:

```
//
// User+Create.m
// EducationPlus
//
// Created by Jon Freberg on 3/22/12.
// Copyright (c) 2012 FrebergWeb. All rights reserved.
//

#import "User+Create.h"
#import "UserWS.h"
#import "School+Create.h"

@implementation User (Create)

+ (User *)userWithEmail:(NSArray *)userArray
    inManagedObjectContext:(NSManagedObjectContext *)context {

    User *user = nil;

    NSFetchRequest *request = [NSFetchRequest fetchRequestWithEntityName:@"User"];
    request.predicate = [NSPredicate predicateWithFormat:@"email = %@", [userArray valueForKey:@"email"]];
    NSSortDescriptor *sortDescriptor = [NSSortDescriptor sortDescriptorWithKey:@"email"
        ascending:YES];
    request.sortDescriptors = [NSArray arrayWithObject:sortDescriptor];

    NSError *error = nil;
    NSArray *users = [context executeFetchRequest:request error:&error];

    if([users count] > 1) {
        NSLog(@"Same email registered multiple times. Should not be possible.");
    } else if(!users) {
        NSLog(@"Users are nil");
    } else if (![users count]) {
        user = (User *)[NSEntityDescription insertNewObjectForEntityForName:@"User"
            inManagedObjectContext:context];
        user.firstName = [userArray valueForKey:@"firstName"];
        user.lastName = [userArray valueForKey:@"lastName"];
        user.graduationYear = [[NSNumber alloc] initWithInt:[userArray valueForKey:@"graduationYear"] integerValue];
        user.email = [userArray valueForKey:@"email"];
        School *school = [School schoolsWithWSInfo:userArray inManagedObjectContext:context];
        user.goesTo = school;
        user.school = school.shortName;
        user.password = [userArray valueForKey:@"password"];
        user.unique = [[NSNumber alloc] initWithInt:[userArray valueForKey:@"id"] integerValue];
    } else {
        user = [users lastObject];
    }
    return user;
}

@end
```

Listing 7.3: User+Create.m category implementation

Listing 7.3 shows how a new object is created. It is checked to see if that same object is already stored in the SQLite database, because multiple occurrences of any objects shouldn't be stored. To query the SQLite database, a `NSFetchRequest` is used. If the request returns something, it is stored in an array, and checked if this array is nil or not. If it is not nil, then it means that this user doesn't need

to be stored, because it is already there. If the array is nil, the user object needs to be created, and its relationships needs to be set. Since the array is an array of user objects, and a user is saved in a NSDictionary, all the valueForKeys that are fetched are key/value pairs inside a NSDictionary. As shown in figure 7.6, a relationship between a user and a school called goesTo is created. This relationship is also set inside the user object so that every user belongs to a school.

The NSManagedObjectContext is the object that works with the Core Data to store and fetch objects, and is actually a pretty complicated object where thread programming is needed. It is important to be careful with how certain problems regarding saving objects locally are solved. The next chapter will discuss how this is done in a good way.

7.2 Handling threads

Handling threads is something that is done quite often when developing iOS applications. For the Education+ application, threads and objects that are not thread safe has to be handled with care. This is because the UI needs to be responsive while some other task is running in the background. The following subsections will first show how to create and run a thread, and then introduce Core Data.

7.2.1 Multithreading and blocks

To perform a task in its own thread, it needs to run inside something called a block. Blocks aren't just used when threads are created, they are used in enumeration, view animations, sorting, notifications and error- or completion handlers. However, the most important use of a block is when threads are created. Threads are created with the Grand Central Dispatch API, or the GCD API which it is commonly referred to as. The GCD API is a C API, and the basic idea behind it is that it has queues of operations. An operation is specified with a block, and every operation from a queue will run in a separate thread. This comes in handy when blocking activity like networking needs to be performed outside of a UI thread, which is always the main thread of an application. The following example will show how a new thread is created to perform a network task:

```
- (IBAction)LogInButtonAction:(id)sender {
    NSString *username = UserNameTextField.text;
    NSString *pass = PasswordTextField.text;
    // Tries to login by calling a web service
    NSArray *user = [UserWS tryLogin ...]

    if(!user) {
        ResponseTextLabel.text = @"Wrong username and/or password.";
    } else {
        [CoreDataSingleton saveUserToCoreData:username inManagedObjectContext:
managedObjectContext];
        [self performSegueWithIdentifier:@"LoginSegue" sender:self];
    }
}
```

Listing 7.4: Stripped down version of LoginButtonAction method

This is a stripped down version of the method that gets run when someone tries to login to the Eduaction+ application, but it serves its purpose to demonstrate how to create threads. A class method with the name `tryLogin` gets called, and this is the method that asks the web server if a user exists so that it can log in or not. It returns a user object if the login was successful, or `nil` if the user didn't exist. When a user is connected through high speed wifi, this method could be executed without making the UI become unresponsive, but an iPad user can be connected to the Internet with either 3G or EDGE, so its always good programming practice in the iOS environment to not execute any networking task in the main thread. To run the `tryLogin` method in a separate thread a block is created as shown in listing 7.5:

```
- (IBAction)LogInButtonAction:(id)sender {
    NSString *username = UserNameTextField.text;
    NSString *pass = PasswordTextField.text;

    // Creates a queue which is called "check login"
    dispatch_queue_t checkLoginQueue = dispatch_queue_create("check login", NULL);
    // Executes the queue in a block who's operations will be executed in a separate thread
    dispatch_async(checkLoginQueue, ^{
        // Tries to login by calling a web service
        NSArray *user = [UserWS tryLogin ...]

        if(!user) {
            ResponseTextLabel.text = @"Wrong username and/or password.";
        } else {
            [CoreDataSingleton saveUserToCoreData:username inManagedObjectContext:
            managedObjectContext];
            [self performSegueWithIdentifier:@"LoginSegue" sender:self];
        }
    });
    dispatch_release(checkLoginQueue);
}
```

Listing 7.5: LoginButtonAction method with thread

Everything that is inside the `dispatch_async` block will be performed in its own thread, which means that no matter how much time this networking task will take, the user can still interact with the applications UI. After a queue has been executed, it is important to remember to release it, so that it doesn't leak memory in the queue by "hanging around" too long.

There is one more problem with this method, which would make the application crash if someone tried to login. The reason for that is that calls to the UIKit, which is every operation that updates or involves some UI component, can *only* happen in the main thread. The `ResponseTextLabel.text` call is telling a label that is located at the login screen to be updated to "Wrong username and/or password" if the login operation was not successful, so this is something that needs to be executed in the main thread, and not in the new thread was just created.

```
- (IBAction)LogInButtonAction:(id)sender {
    NSString *username = UserNameTextField.text;
    NSString *pass = PasswordTextField.text;

    // Creates a queue which is called "check login"
    dispatch_queue_t checkLoginQueue = dispatch_queue_create("check login", NULL);
    // Executes the queue in a block who's operations will be executed in a separate thread
    dispatch_async(checkLoginQueue, ^{
        // Tries to login by calling a web service
        NSArray *user = [UserWS tryLogin ...]
```

```

// Get the main thread to perform UIKit operations
dispatch_async(dispatch_get_main_queue(), ~{
    if(!user) {
        ResponseTextLabel.text = @"Wrong username and/or password.";
    } else {
        [CoreDataSingleton saveUserToCoreData:username inManagedObjectContext:
managedObjectContext];
        [self performSegueWithIdentifier:@"LoginSegue" sender:self];
    }
});
};
dispatch_release(checkLoginQueue);
}

```

Listing 7.6: LoginButtonAction method with main thread

This is a very common problem, so the GCD has a method to get the main queue right away, so UIKit updates can be performed as wanted. The code example given in listing 7.6 is how most threads are created and executed, and is crucial to understand when developing iOS applications.

7.2.2 Core Data

Core Data is what makes it possible to store objects locally, and its `NSManagedObjectContext` is the hub around which all Core Data activity turns, as shown in figure 7.7.

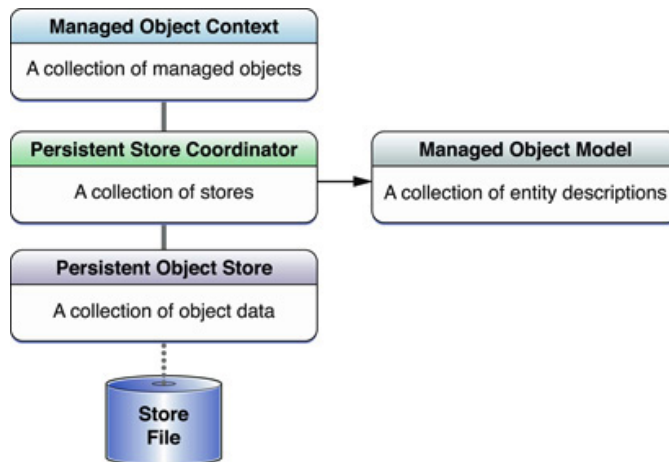


Figure 7.7: Persistence Store Stack

Whats important to know about the `NSManagedObjectContext` is that it is not thread safe. That means that all operations that revolves around the `NSManagedObjectContext`, like storing or retrieving data, has to happen in the same thread that the `NSManagedObjectContext` was created.

However, it is not possible at any time to ask for what thread a certain object is a part of. There are a couple of ways to implement the `NSManagedObjectContext` so that every interaction with it always happens in the same thread. The way it is

solved in the Education+ application is with a Singleton class that instantiates the `NSManagedObjectContext`. The name Singleton comes from the Singleton design pattern, and is a pattern that restricts the instantiation of a class to exactly one object. This singleton object is shared throughout all classes of the application so that communication with it will always happen in the same thread, hence no sudden crash due to trying to access a `NSManagedObjectContext` in the wrong thread will occur.

Chapter 8

Results

This chapter will describe the results of this thesis, with regards to the assumptions made after the initial background research phase. Results from specific tests concerning the effectiveness of the application architecture, as well as challenges that has been met along the way will be stated together with proposals to how they can be avoided in similar projects. Some possible new features for the application for further work with this thesis and application, will also be presented.

8.1 Testing

The usual way of testing a prototype of an application, is to give it to a certain number of people, and let them use it. This way, cumbersome design choices, lack of functionality and bugs are easily discovered. Due to time constraint, this form for testing was not possible to perform. However, to see if the choices made concerning efficiency and speed when communicating with the server was acceptable could be tested by timing the web service calls.

8.1.1 Timing of web service calls

One of the main reasons for developing the iPad application the way it has been done, was to be able to develop the application natively. The only data needed to be fetched over network would then be small JSON objects, since every large file such as a background image would be handled locally on the iPad. To time the different web service calls, the Google Chrome Inspector was used to get a nice overview of all the different elements that gets loaded in a web page. The timing was done from the administrator panel, where features like fetching courses, news and users are done, as shown in figure 8.1.

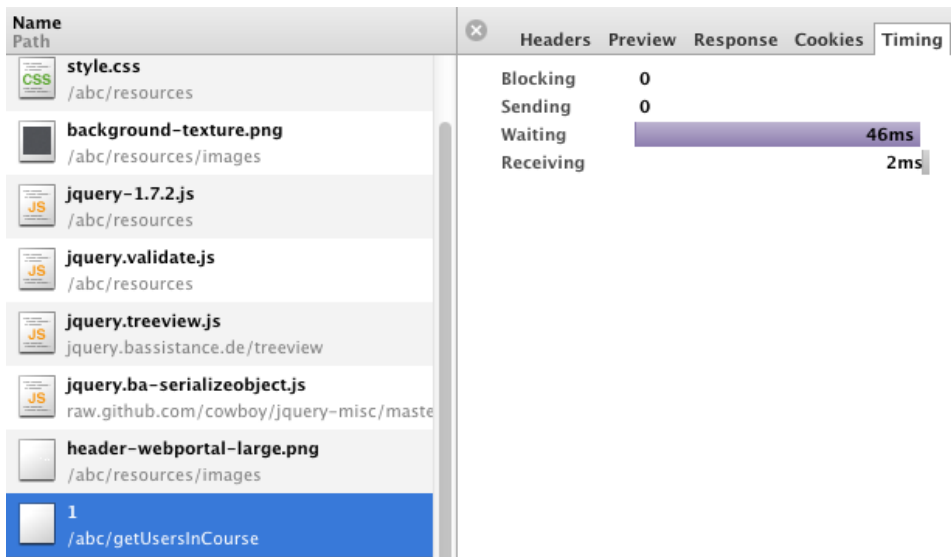


Figure 8.1: Chrome Inspector showing the web service call `/getUsersInCours/1`

To perform some test cases of how much time it would take when connected to either 3G or EDGE, simulations was done with a desktop application called Network Link Conditioner, which is available through Mac OS X Lion, and is used a lot by developers for testing their applications that rely on network connectivity.

Seven different timing tests were performed. Six of them were GET requests where a JSON object is returned of varied sizes, and the last was a POST request. Some of the same web services were called with different parameters that knowingly would give a return response from the web server of different lengths, so the returned JSON objects could be evaluated together with its execution time. This made it possible to see the difference in time when different sized JSON objects was returned from the web server.

Web service	Type	Wait	Response	Total
Get all admins	WiFi	41	1	42
	3G	279	12	291
	EDGE	973	40	1013
Get all users	WiFi	51	3	54
	3G	288	35	323
	EDGE	1007	113	1120
Get all courses	WiFi	46	1	47
	3G	273	17	290
	EDGE	986	37	1023
Get 2 users in course	WiFi	69	1	70
	3G	287	41	328
	EDGE	969	137	1106
Get 4 users in course	WiFi	60	6	66
	3G	291	75	366
	EDGE	974	249	1223
Get 6 users in course	WiFi	53	3	56
	3G	295	92	387
	EDGE	976	251	1227
Create course	WiFi	53	1	54
	3G	295	1	296
	EDGE	976	2	978

Table 8.1: Timing of web service calls in milliseconds

The tests referred to in table 8.1 confirms that the Education+ application has a reasonable response time when it communicates with its web server. The "get all users" web service calls return a JSON object that contains 12 user objects, which gave approximately the same total time as fetching JSON objects that only contained 2, 4 or 6 user objects. This shows that the JSON object data type is as lightweight as originally stated, as well as it is the best choice of format when it comes to client-server based applications for mobile devices.

When users of an application knows that he or she is connected with either 3G or EDGE, a response time of about one second is acceptable. Moreover, as long as the user gets notified with an activity indicator of some sort, to wait up to three seconds are seen as reasonable within the community for developing for mobile devices.

8.2 Challenges

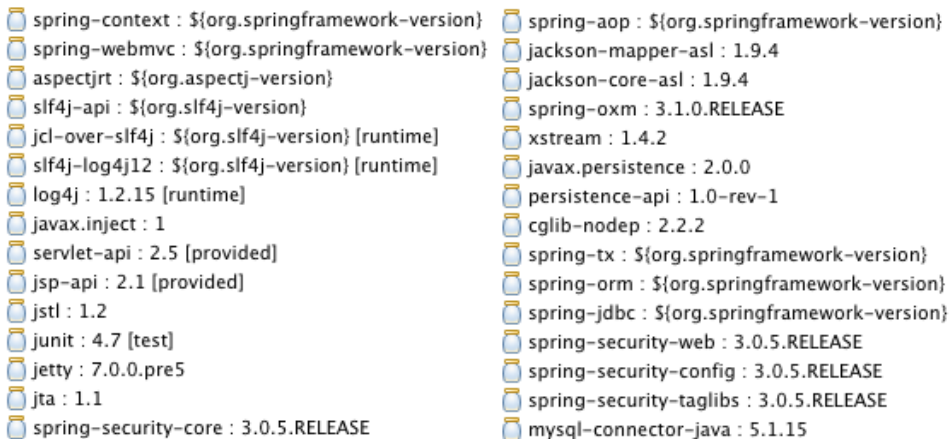
As the implementation of the web server and iPad application started, unforeseen challenges were discovered. The ones that was the most time consuming will be presented in subsection 8.2.1 and 8.2.2.

8.2.1 The web server

One of the best features of the Spring framework is how it helps separating configuration and code. The configuration part was however what brought the most challenges. The frameworks open source nature makes it rely on many different third party libraries, that quickly can cause problems.

Configuring the Spring framework

The Spring framework seemed to be the best framework to use for building the Education+ web server, with regards to the research done. It proved to contain extensive documentation as well as it is continuously maintained and updated due to its large user base. However, the framework showed that it was dependent on a large amount of third party libraries to function. For the Education+ server, 30 different dependencies was needed, as shown in figure 8.2.



```

spring-context : ${org.springframework-version}
spring-webmvc : ${org.springframework-version}
aspectjrt : ${org.aspectj-version}
slf4j-api : ${org.slf4j-version}
jcl-over-slf4j : ${org.slf4j-version} [runtime]
slf4j-log4j12 : ${org.slf4j-version} [runtime]
log4j : 1.2.15 [runtime]
javax.inject : 1
servlet-api : 2.5 [provided]
jsp-api : 2.1 [provided]
jstl : 1.2
junit : 4.7 [test]
jetty : 7.0.0.pre5
jta : 1.1
spring-security-core : 3.0.5.RELEASE
spring-aop : ${org.springframework-version}
jackson-mapper-asl : 1.9.4
jackson-core-asl : 1.9.4
spring-oxm : 3.1.0.RELEASE
xstream : 1.4.2
javax.persistence : 2.0.0
persistence-api : 1.0-rev-1
cglib-nodep : 2.2.2
spring-tx : ${org.springframework-version}
spring-orm : ${org.springframework-version}
spring-jdbc : ${org.springframework-version}
spring-security-web : 3.0.5.RELEASE
spring-security-config : 3.0.5.RELEASE
spring-security-taglibs : 3.0.5.RELEASE
mysql-connector-java : 5.1.15

```

Figure 8.2: All dependencies needed for the web server

The challenge that presented itself was to find the correct version for all these dependencies so that they were compatible with each other. Even though the Spring framework had an extensive documentation, a lot of the external libraries did not. This resulted in having to try and fail until the correct version was found. However, the incompatible version errors didn't occur when the dependencies were added to the project's class path. The application had to be deployed, and certain

aspects of the web servers functionality had to be used to force the error. The result of this was a cumbersome path to making all the pieces fit together.

As long as a Java driven web server is chosen, compatibility issues are most likely to appear due to its open source nature. A solution to achieve a more streamlined configuration phase would be to use Microsoft's ASP.NET framework. ASP.NET is a web application framework and is an integrated part of Microsoft's .NET framework stack as illustrated in figure 8.3.

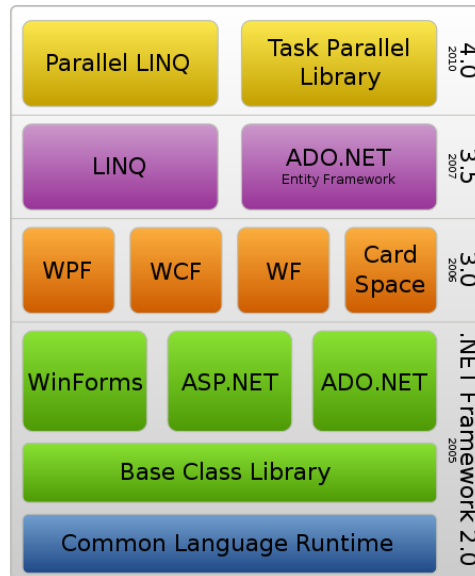


Figure 8.3: The .NET Framework Stack

Since the whole .NET framework is developed by Microsoft, the need of using third party libraries are rarely needed. This leaves out the challenge of keeping external dependencies compatible with each other, since Microsoft will always make sure that compatibility issues never occurs in its own framework stack.

Design bugs

An implementation specific Spring JPA problem was in particular demanding to try to solve. JPA is what handles the transformation of taking a Java object, and storing it as an entity in the database, where every Java instance variable will be represented as a row in the database. The news object has an instance variable called text, which job is to hold the String representation of the news entry created by the administrators. The text instance variable was created as shown in listing 8.1.

```
@Column(name = "newsText", nullable = false, unique = true)
```

```
private String text;
```

Listing 8.1: Entity description of the text instance variable

In Java, text is always represented with a String datatype¹, while a MySQL database has several different string datatype options. The most common is varchar, which is what JPA by default transforms a Java String object into. The problem with the varchar datatype is that it can only hold 255 characters, which is not a sufficient length for a news entry. The correct MySQL datatype to use is *text*, that can store news entries up to 32 768 characters². To achieve this, it had to be specified as a parameter to the @Column annotation as shown in listing 8.2.

```
@Column(name = "newsText", nullable = false, columnDefinition = "TEXT")
private String text;
```

Listing 8.2: Entity description annotations used for the MySQL text data type

The problem with specifying the column definition to be text was that it was no longer possible to set this column as unique. After a discussion at the #spring chat room at IRC, it showed that this was a Spring bug, which they were working on to fix. Due to this bug, the topic of a news entity in the database is set to be unique instead of the news entry itself, which isn't a very good way of implementing the design of the news object, but had to be done this way as no other solution currently exist.

8.2.2 The iPad application

The implementation of the iPad application had mainly two obstacles that took some time to overcome. These obstacles didn't occur because of bugs or implementation errors, but they arose due to lack of experience with Objective-C.

Getting started

When implementing for the first time in Objective-C, there is a lot of information and many design strategies that needs to be understood. A lot of time was spent on researching the design strategies and ideas of how to make iPad applications, but the only way to see if something was understood correctly was to start implementing the application, and learning along the way. In particular, the way the different views and controller classes talk to each other through delegates was challenging to understand completely from the start. After thoroughly working with some of the most basic assignments given in the Stanford course to get a better understanding of how to implement in Objective-C, the Objective-C programming style came more natural.

¹The char datatype is seen as a single character, where a text is defined as one or more words, so the String datatype must be used

²The MySQL TEXT datatypes maximum storage is $L + 2$, $L < 2^{16}$ where L is in bytes.[25] A single Java character is 2 bytes[26], hence $\frac{2^{16}+1}{2} = 32768,5$ number of characters can be stored.

Going outside Apples design guidelines

As many iOS developers advises, to start off any iOS development project by making its GUI is a good thing to do. This makes it easier to understand what data and information that should be given in what views, and it gives a better understanding about what needs to be implemented. Since the experience in developing iPad applications were slim to none, there was especially one design choice that was made that gave some troubles later in the implementation process.

The design choice of having only one view, and from here give information about the students courses and news proved to be a bad choice of design. If this was to be done after "Apples book", the application would have two different views, one for the students courses, and one for its news. Since this application has two tableviews that are going to be populated with data from two different objects that are stored locally with Core Data, two separate controller classes for the two tableviews would have to be created. Each of these then handles the fetching of the correct objects and the population in the correct table. To figure out how to create the course and news controller classes, tutorials written by iOS developers had to be followed instead of figuring out how to solve such a problem by reading Apples documentation. The population of the two tables seemed to work just fine, but after some testing it was noticed that only the news of the last course a student was enrolled in, was the news that got populated into the news table, as shown in figure 8.4.

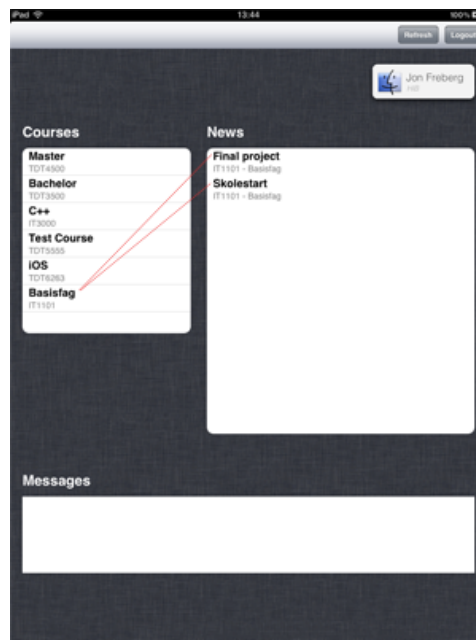


Figure 8.4: News overview problem

The reason behind this problem was that there was no way of passing the data from these controller classes back to the view so that it could work with the objects inside the two tables as wanted, and this was because of the tutorial that was followed had some design flaws in it, which wasn't caught until it was too late.

The solution to this problem would be to make the two controller classes that handles the course and news tableviews be a subclass of NSObject, instead of UITableViewController that they are today. To do this, some refactoring has to be done, and has been chosen to be left as a bug in the prototype due to the time that the error was caught at, and the workload it would take to fix it.

8.3 New features

For further work with the application, some ideas for new features will be presented. These new features could make the Educaiton+ application a good resource for both professors and students.

8.3.1 Messaging system

To ask questions to a fellow student, or to ask a more complex question to a professor about a topic from class is something that a lot of students are hesitant of doing because they are afraid of sounding stupid. But very often, there are a lot of students that are wondering about the exact same thing. By incorporating a messaging system to the Education+ application, it would help making this process a little bit less formal, and to lower the bar for asking questions. A chat that could be divided into different class chat rooms, as well as the opportunity to send private messages to both other students and professors could be used.

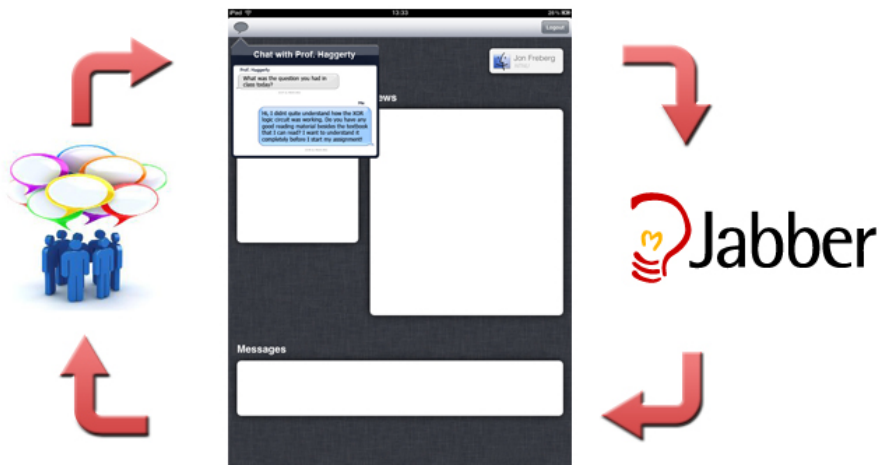


Figure 8.5: Increase communication with the Jabber chat client

This could be implemented by using the open source messaging client called Jabber. Jabber can communicate with the chat server through JSON objects that minimizes the amount of data that needs to be sent, as well as it can be implemented into any kind of platform. To have a messaging system isn't new or groundbreaking, in fact it exists in most modern learning portals like Its Learning³. However, Its Learning only has the opportunity to send private messages, or to communicate with classmates through a forum. What makes this feature so much better at a mobile device, such as the iPad or the iPhone is that this is a device that people carry with them everywhere, and can get notified right away when something happens through the the new iOS5 Push Notification Center feature. Not having to log onto your Mac or PC to check for updates lowers the bar of generating quick responses which makes the message feature more tempting to use.

8.3.2 Quiz system

To find good learning resources besides from the text book, isn't always that easy. Many students wants to gain more knowledge about certain topics in their curriculum to get a better understanding of them.

With a quiz system incorporated into the Education+ application, the professor can create relevant multiple choice quizzes about the topics of the curriculum. This will make the students feel confident that their knowledge about topics are correct, and will help create a better learning experience for the students.

8.3.3 Sharing of documents

A possibility that can only be achieved if implementing natively for iOS is the ability to easily share documents among the users of the application through the iOS iCloud sharing API. This makes it easy for users of the application to share notes and thoughts on different topics from a course so that it is available for others. To make the students collaborate, and to make them discuss different topics with fellow students can help students learn and to get a more fun and work-like experience of learning, rather than sitting one by one and memorizing the curriculum.

8.4 Improvements

Implementation specific improvements can be done with the prototype as it is today. As the implementation phase was ongoing, a lot of new ideas and design improvements came to mind, but where not possible to pursue due to time constraints. These improvements concern both new features to the application, as well as some proposals of how to enhance some of the functionality that is present today. The improvements are covered in the following subsections, and are recommended to be included in a new version of the Education+ application.

³A norwegian Internet-based virtual learning environment - http://en.wikipedia.org/wiki/It's_learning

8.4.1 Push Notification Center

To handle course or news updates in the prototype, the user would have to login to the iPad application, or press the refresh button if the user is already logged in. This is acceptable for a prototype, but for a working application, it should be implemented in a different way. By using the built-in iOS Push Notification Center⁴, updates can be handled by keeping users informed whether the Education+ application is running in the background or is inactive.



Figure 8.6: Notification badge on the application icon

Every time there is an update at the web server, a notification will be given to all its users in form of either a sound, an update badge on the application icon, or both.

8.4.2 Asynchronous web service calls

All web service calls made from the iPad are synchronous calls. The synchronous calls are made in a separate thread, so that the UI will be responsive no matter how much time the web service will use to return with a response. In most cases, this would work just fine, and the UI will be updated with the new data fetched from the server. But there are a couple of cases where the synchronous web service call does not work as wanted. Imagine a scenario where a user uses the Education+ application when traveling, and it is connected to the Internet through 3G or EDGE. If the user presses the refresh button to fetch updates from the server, and the user loses its network connection, the web service call will be aborted. To fetch the intended updates, the user would have to press the refresh button again, once he or she has restored Internet connection. If the web service is implemented asynchronously, the user wouldn't have to press the refresh button again, because the response part of the web service call will automatically wait for the device to get back online, and continue the web service call where it left off, and present the user with its data once it is fetched. After researching the topic, Apple encourages everyone to only send asynchronous HTTP URL requests, so that an application always will handle scenarios like this the appropriate way.

⁴Introduction to the Push Notification Center: <https://developer.apple.com/appstore/push-notifications/index.html>

8.4.3 iPhone compatible

The classes created are all reusable, so that they can be reused in other applications in the future. As long as the MVC pattern is followed, so that all model and controller classes are implemented correctly, it should be able to easily create an iPhone UI for the application as well. Simply create a new iPhone storyboard, create its views, and set the views controller classes. The Education+ doesn't use any elements that are only possible to use in an iPad application, so that internally in the Education+ application, the controller classes should be reusable as well.

8.4.4 General improvements

Some general improvements to apply to the Education+ application are given below in table 8.2.

Deleting objects	The ability to delete objects stored locally at the iPad is not possible. Information about locally deleted objects should be sent to the server regularly to keep the remote database in sync with the iPad.
Validation	When a user registers, no validation of what the user types in are done. To achieve this, the RegisterViewController needs to be the TextFields delegate, and the method textFieldDidEndEditing should perform validation of the different fields.
Searching for schools	When a user registers, the table view that appears when a user search for a school, behaves somewhat strange. Handle these bugs in the UISearchBar delegate methods.
Refreshing tables	When a user touches the refresh button, the news and course tables needs to be touched with a finger for the newly fetched objects to appear. This bug appears due to the nature of how the two table views are implemented. The setNeedsDisplay property set to YES should fix the problem if the tableviews are implemented differently.
Messages	Not implemented.

Table 8.2: General improvements

Chapter 9

Conclusion

The final chapter will answer the research question stated in section 1.2 - Research Question. A conclusion based on the work done will be presented together with contributions and prospects of further work.

The possibilities and limitations when developing mobile applications natively vs. HTML5 can be distinctively concluded with, as shown in table 9.1 below.

	Native	HTML5
Possibilities	Performance Full access to APIs More control Good debugging possibilities Device specific UI Richer functionality Sellable	Cross platform Single codebase/Write once Open coding standards Easy to implement Cost- and Time efficient Easy to update versions JavaScript libraries
Limitations	Not cross platform Complicated Time-consuming Hard to update versions	Bad performance Poor browser support Lack of general support Unfinished standards Limited device APIs Hard to debug at mobile device

Table 9.1: Possibilities and limitations when developing mobile applications

With the client-server architecture, the applications business logic will be kept at the server so the workload of implementing the clients natively for any platform would be significantly reduced. The application would be dependent on communicating with a web server to be able to receive updates, however the amount of data

to be sent back and forth are minimized. The data sent are JSON objects, and they are extremely light-weight. This data format makes it possible to slim down the data size to its minimum, which makes it adequate to make an application rely on network connectivity.

What differentiates a network reliable native application from a HTML5 application, is that together with the client-server architecture, the native application has access to the devices full API. Because of this, the Education+ prototype is able to efficiently store any updates from the server locally to the iPad through the Core Data API. This feature makes it possible to use the application and interact with its locally stored objects even though it is not connected to any network.

To create applications that are dynamic and with an architecture that sets as few limitations to modifiability as possible, is very important. With concern to time- and cost of the development of an application, it is almost always cheaper to choose the HTML5 solution. However, the type of application that will be made should be the deciding factor of what solution to choose. Proper planning always benefits any software development project both professional and private, and is the key to make any good application.

The key findings in the thesis emphasizes that the native approach combined with the client-server based architecture is a worthy candidate when it comes to creating mobile applications that needs to be deployed at multiple platforms. The application wouldn't be cross platform by nature, but the possibilities and benefits of what the native and client-server architecture brings together are heavier weighted than the limitations in many cases. What is important to understand is that the possibilities and limitations of developing applications natively vs. HTML5 will be different in every case. There are no blueprint of what is the correct way to go, and it shows that every mobile application developer are prone to analyze their own needs to be able to make the best decision for their type of application. The importance of planning and identifying an applications features are enormous, and will in many cases be the deciding factor if the outcome will be successful or not.

9.1 Further work

There are some key points to improve to make cross platform mobile application development an easier task. First of all, the need of having both desktop- and mobile browsers that interpret HTML5 and CSS3 code in the same way is the one of the key problems today. With this fundamental problem still in place, it is hard to focus the work on a specific topic like making HTML5 better fitted for mobile application development. Many people argue that the HTML5 compatibility in general will be better in time, and it most certainly will in many aspects.

By taking a look into the past, compatibility issues with HTML4 still exists. This is a "warning sign" that says that it needs to be a thorough review of how the web should behave, and how HTML5 should be interpreted.

From a native development point of view, the different mobile device companies should strive to let as much as their operating systems API be accessible for external use so today's proprietary standards can be let go of. The reason for why not everything is accessible, is due to both security and technological reasons. Android phones has a much more open API than iOS devices, which is reflected in the amount of attacks made on the two different operating systems[27]. There is almost always a valid reason for why these companies choose to make some features only available for native developers, and this is because for every person that will use a part of an API for its original purpose, there is probably a thousand people that will use it to cause harm. With the HTML5 standard being in its unfinished state as it is today, the different mobile device companies are probably reluctant to open up too much in fear of being a popular platform to attack and exploit.

More and more new features and possibilities appears in the field of mobile application development, and both developers as well as mobile device companies are continuously working to make progress in this revolutionary new way of incorporating technology into our every day lives.

Bibliography

- [1] Telenor ASA. 3G and EDGE coverage in Norway. <http://www.telenor.no/privat/dekning/>, 2012. [Online; accessed 16-March-2012].
- [2] Scott OConner. JSON and SOAP in Web Application Services. <http://igoesolutions.com/blog/2010/07/07/json-and-soap-xml-in-web-application-services/>, July 7th, 2010. [Online; accessed 04-February-2012].
- [3] Martin Fowler. Model View Controller. <http://en.wikipedia.org/wiki/Model-view-controller> - <http://martinfowler.com/eaCatalog/modelViewController.html>, March, 2012. [Online; accessed 20-May-2012].
- [4] Restful constraints. http://en.wikipedia.org/wiki/Representational_state_transfer#Constraints, March, 2012.
- [5] Apple Inc. View Controller Catalog for iOS. <http://developer.apple.com/library/ios/#documentation/WindowsViews/Conceptual/ViewControllerCatalog/Chapters/SplitViewControllers.html>, February 16th, 2012. [Online; accessed 10-May-2012].
- [6] Andy Rubin. Google plus public post. <https://plus.google.com/u/0/112599748506977857728/posts/Btey7rJBaLF>, February 27th, 2012. [Online; accessed 26-May-2012].
- [7] Steffen Itterheim. ios sales statistics including q1 2012: Split by model and opengl es 2.0 support. <http://www.learn-cocos2d.com/2012/03/ios-sales-statistics-q1-2012-split-by-model-opengl-es-2-0-support/>, March 22nd, 2012. [Online; accessed 04-May-2012].
- [8] W3Schools. HTML5 Aplacation Cache. http://www.w3schools.com/html5/html5_app_cache.asp, April, 2012. [Online; accessed 02-May-2012].
- [9] The SQLite Consortium. SQLite. <http://www.sqlite.org/>, May, 2012. [Online; accesses 09-March-2012].
- [10] Web workers. http://en.wikipedia.org/wiki/Web_worker, March, 2012.

-
- [11] Jose Fermoso. PhoneGap Seeks to Bridge the Gap Between Mobile App Platforms. <http://gigaom.com/2009/04/05/phonegap-seeks-to-bridge-the-gap-between-mobile-app-platforms/>, April 5th, 2009. [Online; accessed 04-February-2012].
- [12] Jongsoo Park. Design Patterns Elements of Reusable Object-Oriented Software. http://www.stanford.edu/~jongsoo/cgi-bin/moin.cgi/Design_Patterns_Elements_of_Reusable_Object-Oriented_Software#head-339c5a351e5a45acbcdd8ac2fd553301b8b7c6c0. [Online; accessed 26-May-2012].
- [13] Web services in general. http://en.wikipedia.org/wiki/Web_service, May, 2012.
- [14] Stackoverflow forum entry. What is RESTful programming. <http://stackoverflow.com/questions/671118/what-exactly-is-restful-programming>, March 22nd, 2009. [Online; accessed 11-April-2012].
- [15] Java servlet. http://en.wikipedia.org/wiki/Java_Servlet, January, 2012.
- [16] Vaan Nile. Spring MVC. <http://www.vaannila.com/spring/spring-mvc-tutorial-1.html>, January, 2009. [Online; accessed 19-April-2012].
- [17] SpringSource. Chapter:13.2 - The DispatcherServlet. <http://static.springsource.org/spring/docs/2.0.x/reference/mvc.html>, April, 2012. [Online; accessed 02-April-2012].
- [18] Kevin William Pang. Dependency Injection For Dummies. <http://www.kevinwilliampang.com/2009/11/07/dependency-injection-for-dummies/>, November 7th, 2009. [Online; accessed 14-April-2012].
- [19] Bill Venners. Designing with Interfaces. <http://www.artima.com/designtechniques/interfaces.html>, November, 1998. [Online; accessed 27-April-2012].
- [20] Colin Sampaleanu. Green Beans: Getting Started with Spring MVC. <http://blog.springsource.org/2011/01/04/green-beans-getting-started-with-spring-mvc/>, January 4th, 2011. [Online; accessed 27-April-2012].
- [21] ios overview. <http://deimos3.apple.com/WebObjects/Core.woa/Feed/itunes.stanford.edu-dz.11153667080.011153667082>, November, 2011.
- [22] Apple Inc. Delegates and Data Sources. <http://developer.apple.com/library/ios/#DOCUMENTATION/Cocoa/Conceptual/CocoaFundamentals/CommunicatingWithObjects/CommunicateWithObjects.html>, May, 2012. [Online; accessed 04-May-2012].

-
- [23] Apple Inc. Property Declaration and Implementation. <http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ObjectiveC/Chapters/ocProperties.html>, May, 2012. [Online; accessed 04-May-2012].
- [24] Objective c wiki. <http://en.wikipedia.org/wiki/Objective-C#Syntax>, May, 2012.
- [25] Oracle. Data Type Storage Requirements. <http://dev.mysql.com/doc/refman/5.5/en/storage-requirements.html>, May 29th, 2012. [Online; accessed 29-May-2012].
- [26] Oracle. Primitive Data Types. <http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>, May 29th, 2012. [Online; accessed 29-May-2012].
- [27] Juniper Networks Inc. Malicious Mobile Threats Report 2010/2011. <http://www.juniper.net/us/en/local/pdf/whitepapers/2000415-en.pdf>, March 2012. [Online; accessed 29-May-2012] – Pages 6–10.

Appendix A

Education+ user manual

The Education+ application is set up with an administrator user- Ola Nordmann, as well as a student user- Jon Freberg. These users are meant to be used for demonstration purposes.

A.1 Administrator panel

Table C.2 gives information about how to access and use the Education+ administrator panel.

URL	<code>http://master.freberg.org</code>
Login Credentials	Username: sensor@ntnu.no Password: 1234 Name: Ola Nordmann
General information	The original location to the web server is <code>http://vm-6115.idi.ntnu.no:8080/abc/login</code> . If the URL web forward given above doesn't work, or a complete URL is desirable to see, use this URL instead.

Table A.1: Education+ Administrator Panel information

There is not implemented any functionality for creating new administrators, due to the design of the application. The application is meant to be published with the administrators needed. The administrators listed in table A.2 are added to the system so the administrator panel can be tested with administrators from different schools.

Name	Credentials	School
Ola Nordmann	Username: sensor@ntnu.no Password: 1234	NTNU
Terje Rydland	Username: terje@ntnu.no Password: 1234	NTNU
Hans Hansen	Username: hans@hib.no Password: 1234	HiB
Nils Nilsen	Username: nils@uio.no Password: 1234	UiO

Table A.2: List of available administrators

A.2 iPad application

The Education+ application will be found at the iPad's springboard, and looks like this:

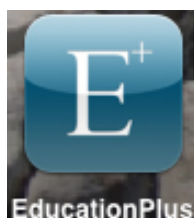


Figure A.1: The Education+ icon

The student with the name Jon Freberg is setup to attend courses that Ola Nordmann administrates. This is meant to be the test account for the iPad. The users available to login with is listed in table A.3 below.

Name	Credentials	School
Jon Freberg	Username: jon@stud.ntnu.no Password: 1234	NTNU
Peter Fredriksen	Username: peter@stud.stud.no Password: 1234	NTNU
Sjur Hjelle	Username: sjur@stud.ntnu.no Password: 1234	NTNU
Tim Ramsey	Username: tim@stud.ntnu.no Password: 1234	NTNU
Sigurd Alstad	Username: sigurd@stud.stud.no Password: 1234	NTNU
Mads Gjerding	Username: mads@stud.ntnu.no Password: 1234	NTNU
Hanne Freberg	Username: hanne@stud.ntnu.no Password: 1234	NTNU
Kristine Jensen	Username: kristine@stud.stud.no Password: 1234	NTNU
Rita Nilsen	Username: rita@stud.ntnu.no Password: 1234	NTNU
Magnus Sandvik	Username: nils@stud.uio.no Password: 1234	UiO
Tor Alfsen	Username: tor@stud.uio.no Password: 1234	UiO
Nick Tollefsen	Username: nick@stud.uio.no Password: 1234	UiO
Jonas Langmo	Username: jonas@stud.hib.no Password: 1234	HiB
Nicolai Prytz	Username: nicolai@stud.hib.no Password: 1234	HiB
Henrik Tenfjord	Username: henrik@stud.hib.no Password: 1234	HiB

Table A.3: List of available students

To test the iPad application in depth, new users can be created in the registration view.

Appendix B

Interview results

The interviewees referred to in section 2.5 - Interviewing software developers will be present as they were answered here.

B.1 Interviewee 1

Age: 40

Company: Visma Consulting AS

Work-title: Senior Consultant

Have you ever developed mobile applications native, with HTML5 or both?

I have developed mobile applications with HTML5. I have not developed native apps.

Which one do you prefer, and have you considered trying the "other"?

My primary function is that of an Interface (front-end) developer, using the open standards of the web. I have no plans on developing in the native languages of each phone's OS.

What is the reason why you chose the development type you did?

It is my primary focus area and makes sense for the project types involved due to the smaller budgets and cross-platform requirements.

Do you see any big advantages by implementing applications native? HTML5?

HTML5, using products such as PhoneGap, allows for a cross platform solution

with a single code base. The code is quite easy to understand and there is also an API that allows utilization of the device's capabilities. Web apps can easily be developed from the existing code base when using HTML5.

Do you see any possibilities that are only possible to achieve when an app is developed native? HTML5?

HTML 5 affords a single code base. This is not possible with native apps.

In earlier projects where you have developed either a HTML5 or a native application, is there anything you missed during the development process?

HTML5 apps are generally frowned upon by the device OS makers, as they would prefer a "closed garden" approach. This leads to a lack of browser support for many emerging areas of HTML5 and an almost complete lack of comprehensive documentation.

Do you see any specific things that can be improved when you develop HTML5 applications? (If you have developed HTML5 applications)

Browser support for the emerging standards (HTML5 CSS3).

A standard solution for making cross platform apps (ala phonegap), supported by all the major OS makers (pure Utopia, IÖm sure)

Have you ever used a HTML application? If so, what do you think about the user experience?

I have made and used many HTML applications and there are, as with any application, good and bad ones. The user experience is a combination of design and implementation, neither of which can be blamed nor attributed to the use of HTML in the programming of the application.

As an end-user of an application, what do you think is most important when it comes to the user experience?

Efficiency and speed.

With keywords, can you give three positive and three negative things when developing an application i HTML5 and native?

HTML5

Positive:

- Cross platform
- Open coding standards

- Code reuse

Negative:

- Poor browser support
- "Walled gardens"
- Lack of support

If you have anything else on your heart about developing applications native vs. HTML5, please elaborate here.

I like the idea of an open standard language that is not financially beholden to any interest group. This is how the web was intended, and indeed what has made the web so ubiquitous. I am firmly opposed to the "walled garden" approach that seems to be prevailing at the moment. As I have personally stated on numerous occasions, I would like to have only one program on my phone, a browser.

Developing HTML5 apps is at the cutting edge right now. It is neither mature nor planned. It is an approach that has arisen from a fundamental need and desire that is currently lacking in most mobile OS vendors approach to app development. Many customers "bet on one horse" (usually Apple), but more and more are not accepting the fleeting dominance of a single OS vendor, and possible customer alienation that is inherent when snubbing other OS systems.

With many projects in all camps (Google, Apple, Microsoft..) being stunted due to the ongoing patent wars, I see no better approach than using a common, open, free and standardized programming approach to push forward application development.

Content should be free from technology. An app should not only be available on a single OS. My dvd should not only play on a Sony machine, my CD only on a Philips stereo, my book only on a Kindle, my App only on an iPhone. The web showed us that there was an alternative method for distribution of content and I can think of no better way to serve content on mobile devices than using the same language that powers the web.

From a technological angle, I hope that the OS vendors see HTML5 as an opportunity and not a threat. I hope that the use of exclusive content to power device popularity is removed from the equation and that a true cross platform programming interface is standardized and promoted.

B.2 Interviewee 2

Age: 27

Company: Statoil ASA

Work-title: Software developer

Have you ever developed mobile applications native, with HTML5 or both?

Ive developed one native, and one PhoneGap application.

Which one do you prefer, and have you considered trying the "other"?
As of today, I prefer native application development. In time, I think HTML5 will take over, but the lack of API to utilize built-in hardware specific things are to big for it to happen now. Today, you can use built-in camera, GPS, accelerometer as well as some other stuff, but there are still a lot of things missing here.

Last but not least, I prefer the performance. Native apps are many times faster then a HTML5 application on a normal device. With normal device, I mean some of the older HTC/Samsung devices that does not run the Android 2.3.X or similar operating systems. CSS3 animations are extremely slow due to the lack of hardware acceleration. This is said to have become better in Android ICS, but I have not tested it.

What is the reason why you chose the development type you did?
We started with this at work. We wanted to explore the domain, as well as see what it could give of values to our organization. Personally, Ive done some mobile app development based on my curiosity for "the new stuff".

Do you see any big advantages by implementing applications native? HTML5?

The biggest benefit of native applications are the performance, as well as the possibility to use all the features that the device has (NFC f.ex.). The big advantage of HTML5 is that it is a lot easier to write code for, as well as you don't need to develop several different apps to make it work on the iPhone, Android, Symbian, Windows Phone, BlackBerry etc. Personally, I see the web browser as the futures platform, both for mobile and desktop applications.

Do you see any possibilites that are only possible to achieve when an app is developed native? HTML5?

With native applications it is much easier to take advantage of all the hardware possibilities the mobile device comes with, f.ex. NFC, gestures and swiping. You

can achieve swiping/gestures with HTML5 apps as well, but it is a lot slower than it is in a native app, thus the user experience would be better for a native app.

In earlier projects where you have developed either a HTML5 or a native application, is there anything you missed during the development process?

Better documentation for what is supported, and not. Like a table with a crossing list over what works, and what doesn't. It is big differences with what works, and what doesn't work for HTML5 apps at different devices.

Do you see any specific things that can be improved when you develop HTML5 applications? (If you have developed HTML5 applications)

For HTML5 apps, I think the performance is the biggest challenge. After that comes the possibility to, in a larger extent to use the built-in hardware features, and to easier be able to communicate with other apps (both native and HTML5).

Have you ever used a HTML application? If so, what do you think about the user experience?

Yes, my impression has been that a HTML5 application is significantly slower than a native app.

As an end-user of an application, what do you think is most important when it comes to the user experience?

It needs to be fast and responsive, and you should be able to use the screens possibilities as much as possible. With that I mean swiping/gestures.

With keywords, can you give three positive and three negative things when developing an application i HTML5 and native?

HTML5

Positive:

- Fast development phase
- Many good javascript libraries
- Easy to test

Negative:

- Not as easy to debug from the mobile devices
- Slow
- Cannot utilize the devices hardware features

NATIVE

Positive:

- Fast
- More controll
- Very good debugging opportunities

Negative:

- More complicated
- Slower development phase
- A little bit harder to test

If you have anything else on your heart about developing applications native vs. HTML5, please elaborate here.

Not answered.

B.3 Interviewee 3

Age: 28

Company: Statoil ASA

Work-title: Software developer

Have you ever developed mobile applications native, with HTML5 or both?

HTML5

Which one do you prefer, and have you considered trying the "other"?
Yes, but until now, I haven't had a "reason" for it.

What is the reason why you chose the development type you did?
My employer (Statoil) wants to focus on HTML5 because it results in less coding, as well as a lot of the material we already have can be reused. An existing web-app can easily be fitted for a mobile device (reactive design seems really interesting as well). If you need to use the devices API, we use PhoneGap.

Do you see any big advantages by implementing applications native? HTML5?

HTML5:

It works everywhere, and you only need one codebase. However, you don't have access to the devices API. (Unless using PhoneGap or similar hybrid solutions)

Native:

Access to native features, such as camera etc. It is also faster and more stable.

Do you see any possibilities that are only possible to achieve when an app is developed native? HTML5?

Since HTML5 is still a bit immature, it is a lot easier to create stable applications native. With a native approach you can also perform background processes such as push notifications.

In earlier projects where you have developed either a HTML5 or a native application, is there anything you missed during the development process?

HTML5:

I missed better javascript/html5 libraries, as well as a more complete standard. I achieved what I wanted, but the applications are a lot slower on older devices. An-

other limitation is that the web browsers still interprets the same code in different ways. Something that works perfect on an iPhone, might not work as perfect on an Android device.

Do you see any specific things that can be improved when you develop HTML5 applications? (If you have developed HTML5 applications)

Better tools like an IDE with full intellisense for javascript, as well as better support for meta tags in the different mobile systems like icons, enable/disable tap zoom etc. More access to mobile features through javascript would also be wanted.

Have you ever used a HTML application? If so, what do you think about the user experience?

More lag than a native application. The UI is also often much better at a native app.

As an end-user of an application, what do you think is most important when it comes to the user experience?

Responsetime, layout, intuitively, accessability.

With keywords, can you give three positive and three negative things when developing an application i HTML5 and native?

HTML5

Positive:

- A single code base
- Updates can be done without the need of deploying the update to every device
- No need for OS specific versions
- Works everywhere, PCs, tables etc.
- Can use the same backend with different view to customize the application to any device

Negative:

- Response time
- Unfinished standards
- To few libraries/widgets available

NATIVE

Positive:

- Apps can be sold through app stores
- Device specific UI
- Richer functionality

Negative:

- Multiple code bases if you want to support every device
- Objective-C/native languages
- More work to release new versions

If you have anything else on your heart about developing applications native vs. HTML5, please elaborate here.

I think that HTML5 is the way to go for companies (enterprise apps that doesn't need to much access to a devices API), especially since the "bring your own device" has become more and more popular. Games and other apps like facebook/twitter will still have better performance as native apps, but this might change when mobile browsers gets better as well as the HTML5 standard is finished.

B.4 Interviewee 4

Age: 36

Company: Statoil ASA

Work-title: Senior System Analyst Developer

Have you ever developed mobile applications native, with HTML5 or both?

Ive developed native apps for Android(Fifa 11 Tracker, Fifa 12 Tracker, VG debate) and for iPhone(FifaTracker 12). I haven't developed any HTML5 mobile applications yet, but Ive been thinking about it.

Which one do you prefer, and have you considered trying the "other"?
I choose native as long as there are no strong reasons against it.

What is the reason why you chose the development type you did?
My first app was natively for Android devices. I liked it, so that became my preferred platform.

Do you see any big advantages by implementing applications native? HTML5?

Native:

- Very good performance
- Access to low-level OS
- Possible to sell the app
- Possible with data updates in the background

HTML5:

- Cross platform
- Code once, run everywhere

Do you see any possibilites that are only possible to achieve when an app is developed native? HTML5?

Native:

- Push notifications for updates
- Low level OS API. (Android)

In earlier projects where you have developed either a HTML5 or a native application, is there anything you missed during the development process?

Doesn't miss anything, its like developing an desktop application.

Do you see any specific things that can be improved when you develop HTML5 applications? (If you have developed HTML5 applications)
Not answered.

Have you ever used a HTML application? If so, what do you think about the user experience?

HTML5 applications seems like they are not finished, like an early alpha version. Both JQuery and PhoneGap tries to copy the native feeling, but doesn't achieve it due to bad performance and the lack of functionality.

As an end-user of an application, what do you think is most important when it comes to the user experience?

Good performance. Don't want to get the feeling of using a website.

With keywords, can you give three positive and three negative things when developing an application i HTML5 and native?

HTML5

Positive:

- Write once

Negative:

- Bad APIs

- Bad mobile web browsers

- Bad performance

NATIVE

Positive:

- Performance

- API-access

- Sellable

Negative:

- Not cross platform

If you have anything else on your heart about developing applications native vs. HTML5, please elaborate here.

I feel that mobile web browsers need to get a fully accelerated support for HTML5 before this form of application will be reasonable to use. Third party APIs also

has to become more stable. HTML5 based websites (not applications) has recently become more mature, so if I was going to develop a website, the tools/language would be good enough.

Appendix C

General information

C.1 Tools and IDE's used

	Web Server
IDE	SpringSource Tool Suite (STS) version 2.8.1.RELEASE - http://www.springsource.com/developer/sts
Version control	Subclipse. An STS extension plug-in providing support for subversion (SVN) within the STS IDE

Table C.1: Web server tools

	iPad application
IDE	Xcode version 4.3.2 - https://developer.apple.com/xcode/
Version control	Built in GIT support within Xcode

Table C.2: iPad tools

C.2 Creating admin and school objects

There is no feature to create admin or school objects. This is left out because no administrator, which is a professor should be able to create these objects. The school and admin objects present today are created by calling their creation web services directly from a Mac OS X application called HTTP client. HTTP client is a tool for debugging, creating and inspecting HTTP messages. HTTP Client can be downloaded from the Mac OS X app store, and its web site is <http://ditchnet.org/httpclient/>. A similar client for the Windows operating system is Fiddler - <http://www.fiddler2.com/fiddler2/>.

The web service URL to create a school object is:
`http://vm-6115.idi.ntnu.no:8080/abc/school`

and for an admin object:
`http://vm-6115.idi.ntnu.no:8080/abc/admin`

The school object needs to be created before the admin object, since an admin object belongs to a school. It is important to remember to set the Content-Type to application/json, and to pass the JSON object to be created as the body of the HTTP message. Below is an example of how a school and admin JSON object looks like, and a screenshot of how the call to the create school web service from the HTTP client application:

```
/* School object in JSON format: */
{"schoolShortName": "UiO", "schoolFullName": "University of Oslo",
  active": true, "schoolID":0}

/* Admin object in JSON format: */
{
  "id": "0",
  "email": "sensor@ntnu.no",
  "firstName": "Ola",
  "lastName": "Nordmann",
  "password": "7110eda4d09e062aa5e4a390b0a572ac0d2c0220",
  "school": {
    "schoolID": "1",
    "schoolShortName": "NTNU",
    "active": "true",
    "schoolFullName": "Norwegian University of Science and
      Technology"
  }
}
```

Listing C.1: Example JSON objects

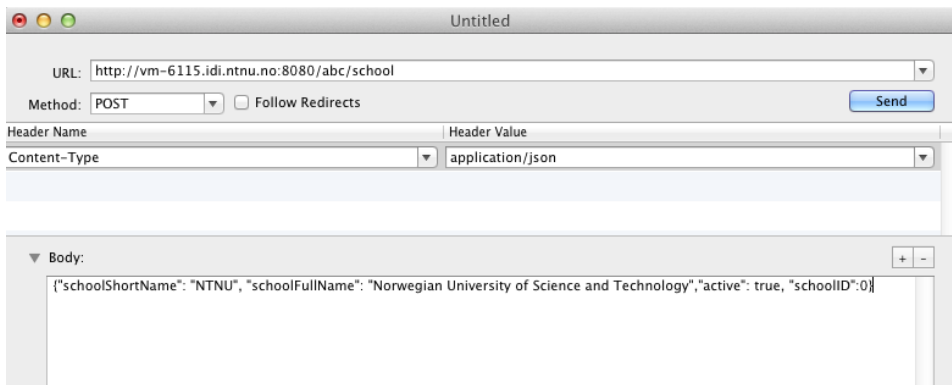


Figure C.1: HTTP client web service call