FAK EKI

Dennis Eklind, Zeena Yalda

Flexible Time Reporting with an iPhone Application

Computer Science

C-level thesis

Date/Term:	2010-06-11
Supervisor:	Simone Fischer-Hübner
Examiner:	Martin Blom
Serial Number:	C2010:13

Karlstads universitet 651 88 Karlstad
Tfn 054-700 10 00 Fax 054-700 14 60
Information@kau.se www.kau.se

This report is submitted in partial fulfillment of the requirements for the Bachelor's degree in Computer Science. All material in this report which is not my own work has been identified and no material is included for which a degree has previously been conferred.

Dennis Eklind

Zeena Yalda

Approved, 2010-06-11

Advisor: Simone Fischer-Hübner

Examiner: Martin Blom

Abstract

A company that works for many customers and charges their customers per worked hour needs to keep track of how many hours each employee has worked for each customer and on each project. To keep track of this, the employees have to report all their worked hours into the company's business database. In Ninetech's case they are using a web-based environment, which was not regarded as flexible and user-friendly. For allowing employees to report working hours more flexibly from various locations and at any time, Ninetech therefore felt the need to find out if the time reporting could be done using an iPhone application.

In this thesis, we have after studying the developing environment for iPhones, created an application for time reporting which was then evaluated by Ninetech employees according to its perceived usability and functionality and which received positive feedback. We find it therefore very likely that in the near future, Ninetech employees will be able to start reporting time on their iPhones.

We have also interviewed some of Ninetech's personnel to find out what other applications should be useful for the daily work at Ninetech if they were implemented. The result of these interviews has been presented as a prioritized list of applications to implement in the future.

Acknowledgements

We want to give a special thanks to our supervisor, Simone-Fischer Hübner for helping us with writing this report. We also want to thank Mattias Berglund, Janolof Elander and Jonas Rozenich at Ninetech, who have been our contacts at the company and helped us in the planning of the application. Also Irina Persson at the Karlstad University Library deserves a thank you for helping us with the reference list.

1 Chapter one: Introduction 1
1.1 Project task: 1
1.2 Thesis outline 2
2 Chapter two: Project Background 5
2.1 iPhone 5
2.1.1 iPhone application vs iPhone web-application 5
2.1.2 License, Enterprise vs Standard 6
2.2 Project Discussion 7
2.2.1 Study 7
2.2.2 Existing system 7
2.2.3 Prototype 7
2.2.4 Security 9
2.3 Chapter summary 11
3 Chapter three: Technical Background 13
3.1 Introduction 13
3.2 Project developing environments 14
3.2.1 iPhone SDK 14
3.2.2 Xcode 16
3.2.3 Interface builder 19
3.2.4 Objective C 22
3.2.5 XML 25
3.2.6 C#.NET 26
3.2.7 IIS 28
3.3 Summary 29
4 Chapter four: Security aspects 31
4.1 Cryptography 31
4.2 HTTPS 33
4.3 Basic Access Authentication 35
4.4 iPhone security 36
4.5 Security decisions in our prototype 37
5 Chapter five: Prototype development 39
5.1 Work method 39
5.2 Prototype description 41

5.3 SUMMARY: 48
6 Chapter six: Interviews 49
6.1 Introduction 49
6.2 Application ideas 51
6.3 Prioritized list 54
6.4 Chapter summary 56
7 Chapter seven: Evaluation 57
7.1 Introduction 57
7.2 Prototype 58
7.3 Chapter summary 63
8 Chapter eight: Conclusions 65
8.1 General summary 65
8.2 Future features 66
8.3 General conclusions-What has been achieved 67
9 REFERENCES 69
10 Appendix 1: Evaluation form 71

List of Figures

Figure 3.1 The four layers of the iPhone Operating System. 16
Figure 3.2 The Xcode text editor. 18
Figure 3.3 Window for choosing a Template. 21
Figure 3.4 A document window. 21
Figure 3.5 A view window. 22
Figure 3.6 A simple application in the iPhone simulator. 25
Figure 4.1 Shows the TLS handshake. 33
Figure 5.1 A model of the project. 39
Figure 5.2 A model of the project, using straws. 40
Figure 5.3 Overview of our prototype. 41
Figure 5.4 The messages sent between the application and the server. 42
Figure 5.5 The first view in our first suggestion for a GUI. 44
Figure 5.6 The second view in our first suggestion for a GUI. 44
Figure 5.7 The first view of our application. 45
Figure 5.8 The second view of our application. 46
Figure 5.9 The HoursAndDate view of our application. 47
Figure 5.10 The third view of our application. 47
Figure 7.1 Answers to the first question. 58
Figure 7.2 Answer to the second question 59
Figure 7.3 Answer to the third question. 59

1 Chapter one: Introduction

Ninetech is an expanding knowledge company within IT, located in Karlstad that has a need for their employees to have the possibility to be mobile. As a result of this, all employees have been equipped with a laptop and a mobile phone for a few years. The company has an interest in finding out whether an iPhone would give any added value for consultants and employees.

1.1 Project task:

The task consists of two parts:

- The first part is to design, develop, implement and test an iPhone application for time reporting directly into the company´s business system. The iPhone application needs to be simple, fast and intuitive for the employees to report their worked time. Ninetech already has a web-based time reporting system but the new application aims to complement the existing system and to make it more accessible.
- The second part is to analyze the employees need for additional applications to make their daily work more effective and to represent the result of our analysis as a priority-ordered list of applications that would be useful if they were developed in the future.

1.2 Thesis outline

Chapter one: Introduction

The first chapter gives a short introduction to our project, introducing our constituent and the two tasks of our project.

Chapter two: Project Background

In the second chapter we give a deeper background of the project. We explain the concept of an iPhone and discuss some of the first questions we had to deal with before starting to implement any code in our application.

Chapter three: Technical Background

In this chapter we give background information about the tools and languages used in this project. Both on the server side and on the client side that is the iPhone.

Chapter four: Security Aspects

The fourth chapter is about the security aspects of this project. We explain the need for security and cryptography. Details are also given about the specific security topics in our project, like HTTPS and iPhone security.

Chapter five: Prototype development

Here we discuss the work method used in this project and the process of implementing the prototype itself.

Chapter six: Interviews

In this chapter we interviewed seven volunteers to find out what other applications would improve Ninetech's daily work. We presented the result as a prioritized list of applications to be implemented in the future.

Chapter seven: Evaluation

We evaluated the project by letting some of Ninetech's employees try our application and answer a few questions about how they experienced the application. We also had an evaluation meeting with our supervisors at Ninetech.

Chapter eight: Conclusion

In the last chapter we summarize the thesis and list some more features that should be added to our application in the future. And finally we reveal our conclusions of the project.

2 Chapter two: Project Background

In this chapter we give a background to this project. We give a brief introduction to what an iPhone is and we discuss some of the first questions we had to deal with in this project. We go on to describe the part of the project where we interviewed a group of Ninetech employees about their need for different iPhone applications. Finally we discuss our prototype and the security aspects in this project.

2.1 iPhone

" The iPhone is a line of Internet and multimedia-enabled smartphones designed and marketed by Apple Inc., and released in 2007. An iPhone functions as a camera phone (also including text messaging and visual voicemail), a portable media player (equivalent to a video iPod), and an Internet client (with e-mail, web browsing, and Wi-Fi connectivity). The user interface is built around the device's multi-touch screen, including a virtual keyboard rather than a physical one." (Wikipedia [19].)

One of the revolutionary things about the iPhone is the possibility for everyone to develop their own applications. The environment used for developing iPhone applications is called the iPhone Software Development Kit (SDK)[1]. It is a part of Xcode [23], which is the standard development environment for Macintosh computers. Therefore iPhone applications can only be developed on a Macintosh. Anyone who wants to download this environment from Apple´s homepage first needs to register an Apple ID. There is a free version of the registration, however if you want to be able to use your application on an iPhone or release the application for sale or for free, you need to pay a fee for this. Apple then has the right to stop the application if they don't want this application available on an iPhone. The application can, if permitted, only be distributed through App Store. App Store is where Apple sells and distributes software and media online.

2.1.1 iPhone application vs iPhone web-application

An alternative solution to our problem could be to develop an iPhone web-application instead of a regular iPhone application. Apple has released a standard for how to write so called web-applications for iPhone. They are practically web pages designed to look like an iPhone application, when opened on an iPhone. This way, we would not need any license (see section 2.1.2). And since there is already a working web-interface for reporting time from a computers web browser, we would only have to make a new version of the existing web-interface

that was suitable for iPhone. However, if we choose the web-application solution, it will be impossible to enable the user to fill in his time report while being offline, to have the iPhone send the information as it regains Internet connection. This feature might not be implemented at a first stage anyhow, but it was requested to be a possible future addition to the application. Besides that, both we and Ninetech find a great value in learning how to develop an iPhone application. Therefore this was our choice.

2.1.2 License, Enterprise vs Standard

When the developed iPhone application is ready to be released, it needs to go through App Store. To be allowed to release applications through App Store, we need to buy a license. There are two types of licenses. The iPhone developer standard program is the most common one and is used for anyone who wants to release their application to the public. It is possible to set a price for purchasing the application, in which case the developer receives a check with 70% of the monthly revenue.

The iPhone Developer Enterprise Program on the other hand, is designed for company in-house applications. This would be the type of license suitable for us, but unfortunately it is only available for companies with 500 or more employees. Ninetech has less than 100 employees, so the iPhone Developer Enterprise Program is out of the question.

There is however another solution. Both licenses have the possibility of releasing Ad Hoc distributions. This means that one can chose 100 iPhones to distribute the application to at a much lower cost.

The Ad Hoc distribution is tempting, since the 100 copies of the application would be more than sufficient for the companies needs today. On the downside we have the future aspects. The company is expanding and does not want a limit for how many employees can be equipped with their iPhone application. And if the application turns out really well, they might be able to sell it to other companies. The conclusion Ninetech reached was to acquire an iPhone developer standard license and distribute the application as a free application. If anything should be sold, it should be the server side application, needed to use the iPhone application.

2.2 Project Discussion

2.2.1 Study

One part of our project was to conduct a study amongst the employees of the company to find out what other iPhone-applications could be useful for the everyday work. Seven volunteering employees were chosen to be interviewed. We encouraged them to be very creative and think "out of the box", possible technical limitations would not be taken under consideration at this stage. Our goal was to get a large amount of ideas, both realistic and not so realistic. We did not want to limit the interviewees, so we asked them fairly few and general questions to make sure we got their ideas and not ideas influenced by our questions. We would later evaluate all the ideas and make a priority list of which applications we would recommend Ninetech to develop if they decided to give all employees their own iPhones.
This list of possible future applications might also be possible for the company to produce for customers at a later state.

2.2.2 Existing system

Today Ninetech uses a web-based system for reporting their worked time. The timelines are stored in a database. Complaints have been made that the web page is not user friendly and a bug in the page creates a risk for time to be accidentally marked as non-chargeable resulting in the consultants working for free and Ninetech not getting paid.

2.2.3 Prototype

Our prototype is a solution for reporting time from an iPhone to the JEEVES database. It includes both the iPhone Application and the web service needed for the iPhone to communicate with the server. We will unfortunately have limited access to the server-side, so we will have to be assisted by Ninetech personnel in the development of the connection between the server interface and the actual database. We need to connect the full chain from the user´s finger-typing on the iPhone, via some background-layer inside the iPhone, resulting in the transmission of some data to the server. The server authenticates the user and accesses the database and sends the correct information back to the iPhone and this data is displayed in a lucid way. Then the user makes new decisions based on the new information and new information is sent all the way via the server and the database and back to the user.

User interface

There are many possible ways of creating a user interface. It was obvious that the existing web-based interface was not a good role model. Still we had to follow the same basic principal as the web-based interface did. Most of the information provided by the user had to be given in a specific order to function with the database. So the question was: How could we create a more user-friendly interface on a display that is much smaller than a computer´s and still lets the users provide the same amount of information?

After discussing how the time reporting is done today, with some of Ninetech's staff, we found that often an employee uses the same combination of options at multiple occasions. Therefore a good idea would be to store and enable reuse of an earlier selection.

The idea is to have a first page that lets the user choose from previously reported timelines and edit only the parts that needs editing before the timeline is reported. There is of course also an option to create a completely new timeline.

Another improvement will be the possibility to report time worked for a new, unregistered customer or project. This way, the employee does not need to keep track of how many hours he has worked while waiting for the administration to add the new customer or project.

Database connection

To reach the database we had to go through a server-based service, which we did not know anything about. The first thing we did was to setup a local virtual web server using Visual Studio, for test purposes. At first we setup the web server as a mockup. That is, it always gave the same response regardless of the request sent. This was done to be able to test the iPhone's ability to send and receive messages. At that stage we did not have any database connection to work with, so we had to work with the local mockup server. During the test period we did not work against the real database at all. We did however eventually get access to a testing database on the same server system as the actual database. This was necessary because that way we got an environment that acted the same as the real database would, but we did not risk doing any unwanted changes to the actual database. And there was also some sensitive information in the actual database. That information could be, for instance, what customers Ninetech has, how much they work for each customer or what price each customer is charged for their services.

When we started with the first part of our application which was the login, we hard coded all the information we needed and put it on the mockup using Visual Studio and C sharp(C#)[16]. The information we needed in the login application was username and employee number. This was a temporary solution until we had got the test database ready and we did not have to hard code the information any more. When the project ends and we get our application ready for delivery, Ninetech will connect it to the actual database in order to access the real information.

2.2.4 Security

One main issue is how to keep the user´s password safe. We can find three different possible risk zones, where information could be compromised. In the iPhone itself, on the web while being sent between the iPhone and the server and on the server-side in connection to the database. The last risk zone, on the server-side is not our job to secure. It is already a fully functional system and it is secured by Ninetech. The second risk zone, on the web while being sent, is a very critical step. But we use HTTPS[4] to send the information encrypted. This method is widely used and considered to be secure enough. The first risk zone is the iPhone itself. Since it is likely that every employee will be using only (or at least mostly,) their own iPhone for the time reporting, it would be very convenient if the username and password could be stored inside the keychain of the iPhone. But is it safe to store passwords in an iPhone? What happens if a user loses his iPhone and someone ill-intended finds it? Will the attacker be able to use the password to log into the system? Will he even be able to get the password out of the iPhone in clear text?

In July 7 2007, Independent Security Evaluators (ISE) came out with a report on security breaches in the iPhone. [3] They tell us that: *"The iPhone runs a stripped down and customized version of Mac OS X on an ARM processor. Much of the device's claimed security is reliant on its restrictions against running third party applications. Only Javascipt code can be executed in the Safari web browser, ensuring that all such code executes in a "sandbox" environment. Many of the features of Safari have also been removed, such as the ability to use plug-ins such as Flash. Likewise, many filetypes cannot be downloaded. These actions serve to reduce the attack surface of the device."*
Then they go on saying:

”However, there are serious problems with the design and implementation of security on the iPhone. The most glaring is that all processes of interest run with administrative privileges. This implies that a compromise of any application gives an attacker full access to the device. Like the desktop versions of Mac OS X on which its operating system is based, the iPhone also does not utilize widely accepted practices, such as using address randomization or non-executable heaps, to make exploitation more difficult. These weaknesses allow for the easy development of stable exploit code once a vulnerability is discovered."

The team working with this evaluation demonstrated the weaknesses by creating a malicious HTML document that forced the iPhone to make an outbound connection to a server they controlled and send personal data like SMS text messages. They later discovered that it would be possible to also have the iPhone send passwords and basically any information. ISE informed Apple of all security breaches they had found, and hopefully Apple upgraded the iPhone to be more secure.

This report from ISE is obviously worrying. But it is normal for new systems and devices to have problems. It is therefore likely that Apple took care of the problem before it was used for ill-intended purposes. We contacted one of the authors, Charlie Miller and he confirmed that improvements had been made.

2.3 Chapter summary

The application we are developing is going to be used for reporting time to Ninetech's business database, JEEVES. We will transfer data between the iPhone and the server, using the XML-format[22]. Communication security will be achieved by the use of HTTPS. There will be a server based service between the iPhone and the database, where authorization will be performed. An interesting aspect of this will be the interaction between a server from Microsoft and an iPhone from Apple. The use of XML for communication between the two should bridge that gap. We are using objective C[10][11] as the programming language on the iPhone and visual C#[21] on the server-side.

In addition to the iPhone application implementation, we also conduct a study of what other iPhone applications would be the first choices to implement for use within the company. We interview seven volunteering employees and ask them questions about what they would like to use an iPhone for in their work. The result is presented in a list of possible future applications to be implemented.

3 Chapter three: Technical Background

3.1 Introduction

This chapter is about giving an introduction to the technologies used for prototype development. Usually when we need to implement an application we first choose the programming language and second the developing environment, suitable to the chosen language. But sometimes we do not have the choice, because some applications must be done in a specific language which must be implemented in a specific development environment and sometimes we choose them, because we have more experience in running an environment than others and have more experience in implementing code in one language than others. The company might have resources such as platforms and staff that already are educated in a certain language.

The advantage of choosing a new programming language and developing environment is that it gives the company and the employees the opportunity to improve their knowledge. The disadvantage is that it costs time and money to get this knowledge. In our project we did not have this option. We were forced to learn Objective C, since it is the programming language used for developing iPhone applications, Xcode was the only suitable development environment and iPhone SDK was the only development tool. In order to test the application during the construction we used an iPhone simulator until we got our license from Apple which allowed us to test our application on an iPod from Ninetech. We worked with XML which is one of the most established and well known tools to encoding documents electronically.

The implementation of all iPhone applications must be made on a Macintosh and to test the results there is an iPhone simulator in Xcode which we could use. We could download the application into an iPhone or into an iPod to see the results, but to do that Ninetech needed to buy a license. The web server was one of the tools we needed to get information like authentication information such as username and password for the employees and C# was the language we used to implement the web-methods on the server interface.

3.2 Project developing environments

To develop iPhone applications, we use Xcode, Apple's Integrated Development Environment. Xcode provides tools to design the application's user interface and write the code that makes it work. Interface builder is another Apple graphical editor for designing user interface components for Cocoa and Carbon applications. In this section we will present the tools used for iPhone developing, in more details. To understand the functionality of these tools we need to explain the iPhone Operating System (iPhone OS)[14].

3.2.1 iPhone SDK

The iPhone SDK allows the user to create and develop applications which can be run directly on iPhone or iPod touch or it can be run on an iPhone simulator in MAC. Loading the application on to devices is possible after paying a fee and registering. The application can also be distributed through App Store, Apples own website. Apple has the right to remove the application from App Store.

Apple describes the iPhone technology as layers, the iPhone OS consists of four abstraction layers [1]. The programmer has the choice to use the low-level frameworks or the higher-level framework. The higher-level frameworks provide object-oriented abstractions for lower-level constructs. It helps to reduce the number of lines of the written code. The lowest layer is the Core OS layer and on top of that layer is the Core Services layer. On top of the Core Services layer is the Media layer and the highest layer is the Cocoa touch layer as shown in Figure 3.1. We can simplify the iPhone OS and think of it as two layers which are the Cocoa layer and the C layer. Cocoa layer comprises the operating system's layers as shown in Figure 3.1. The three lower layers which compose the C layer consists of low-level file Input/Output, network sockets, POSIX threads and SQLite and we use C language functions to manipulate this layer [1].

The first layer is the Touch layer [1]. In iPhone, Cocoa is called Cocoa touch rather than just Cocoa as it is in the MAC OS, because iPhone OS include touch events. Touch events allow the programmer to implement responses to the user's touching on the screen with his fingers. Cocoa touch layer contains: multi touch events, multi touch controls, accelerometer, view hierarchy, localization, alert, web views, people picker, image picker and controllers. There are two frameworks which are mostly used in this layer, they are used almost in every single

program and they are: the UIKit framework and foundation framework. When we developing an application, we should always start with these frameworks and drop down to lower-level frameworks only as needed.

The UIKit framework is dedicated to the iPhone user Interface layer and contains classes such as UIView. UIKit is the UIframework in the iPhone runtime, the equivalent of AppKit for traditional OS X applications. The second framework is the foundation framework and it is dedicated for the standard programming topics such as I/O files and the collections strings and it is the cocoa foundation layer.

The Media layer [14] is responsible to the graphics, audio, and video technologies to create the best multimedia experience on a mobile device. These technologies were designed to make it easy to build applications that deal with pictures and videos. The high-level frameworks make it easy to create advanced graphics and animations quickly, and the low-level frameworks provide the needed tools to give the programmer all the possible solutions he might want. The Media layer contains: Core Audio JPG, PNG and TIFF, OpenAL PDF, Audio Mixing Quartz (2D), Audio Recording Core Animation and Video Playback OpenGL ES.

The third layer is the Core Service layer and it contains: Collections Core Location, Address Book Net Services, Networking Threading, File Access Preferences and SQLite URL utilities.

The fourth and last layer is the Core OS layer and it manages the virtual memory system, threads, the file system, the power management and security. The drivers in this layer provide the interface between the available hardware and the system frameworks that vend hardware features.

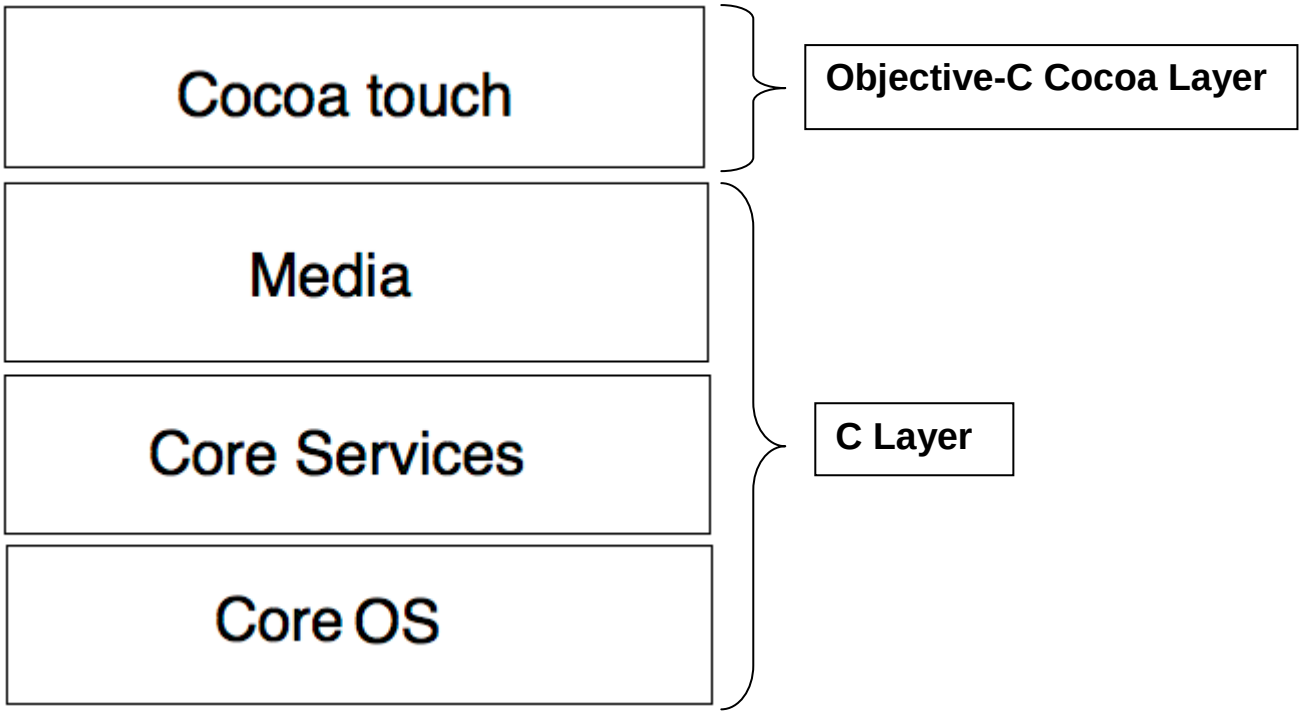

Figure 3.1 The four layers of the iPhone Operating System.

3.2.2 Xcode

The Xcode suit is a software development tool on MAC OS X, developed by Apple. The history of Xcode suit developing is synonymous with MAC OS X developing and it has its roots from NeXT (NeXT was an American computer company headquartered in RedWood City, California, that developed and manufactured a series of computer workstations intended for the higher education's and business markets). The Xcode suit was released on 24 of October 2003 of MAC OS X v10.3 and referred to as a developer tool. The latest version is Xcode 3.2 and it is bundled free with MAC OS X 10.6, but it is not installed by default because it is not supported on older version of MAC OS, but there are older versions available free to install. Figure 3.2 illustrates the Xcode window. Xcode contains a text editor for editing project's text files. There are many options for using this editor to view the text files in a certain project. The programmer can choose to have a single editor for all the text files or to have multiple editor windows open at once. Xcode text editor has a navigation bar that provides a number of menus for navigating within and between files to find information in these files. We can view the text editor in two ways:

-A text editor window: it is a window for editing a file. In order to have the text editor window open all you have to do is to double click on the desired file.

-A text editor pane: known as attached editor. The text editor pane is a part of other windows, such as the project window, debugger window and build results window.

The project window is one of the most important windows in Xcode because it is where we can do the most work and because it allows the user to display and organize the source files. It allows accessing and editing all the files of the project that the user is working on, because it includes the text editor pane. The text editor pane is one of the most important parts of the project window because we use it to write our code, but there are two other parts which are very important as well and these parts are the Groups & Files list and the Detail view.

The Groups & Files list provides an outline view of your project´s content. You can move files and folders around and organize your project´s content in this list. The current selection in the Groups & Files list controls the content displayed in the detail view. The Groups & Files list contains two types of groups: static groups and smart groups see Figure 3.2.

Static groups include the project group which is named after the project and represented by the blue project icon and the static groups themselves are grouped under the project group. A static group is identified by a yellow folder and it includes all the header files, frameworks files, implementation files.

Smart groups are subdivided into two types: built-in smart groups and custom smart groups.

Built-in smart groups: There are several built-in smart groups:

-Targets: Contains the targets in a project. A target contains the instructions for creating a software component or product.

-Executables: Contains all the executables defined in a project.

-Errors and Warnings: Lists the errors and warnings generated when the program is built.

-Find Results: Contains the results of any searches that has been performed in a certain project. Each search creates an entry in this group.

-**Bookmarks:** Lists locations files or specific locations within a file to which the programmer can return easily.

-**SCM:** Lists all the files that have source control information.

-**Project Symbols:** Lists the symbols defined in your project.

Custom smart groups: collect files that match a certain rule or pattern. These groups have purple folder icons. Xcode provides two predefined custom smart groups:

-**Implementation Files:** Contains the implementation files in a certain project.

-**NIB Files:** Contains the nib files used to create the product.

Hide Detail View button. Double-clicking this button hides and shows the detail view.

The Detail view shows the item or items selected in the Groups & Files list. You can browse your project's content in the detail view, search them using the search field, or sort them according to column. The detail view helps you rapidly find and access your project's content.

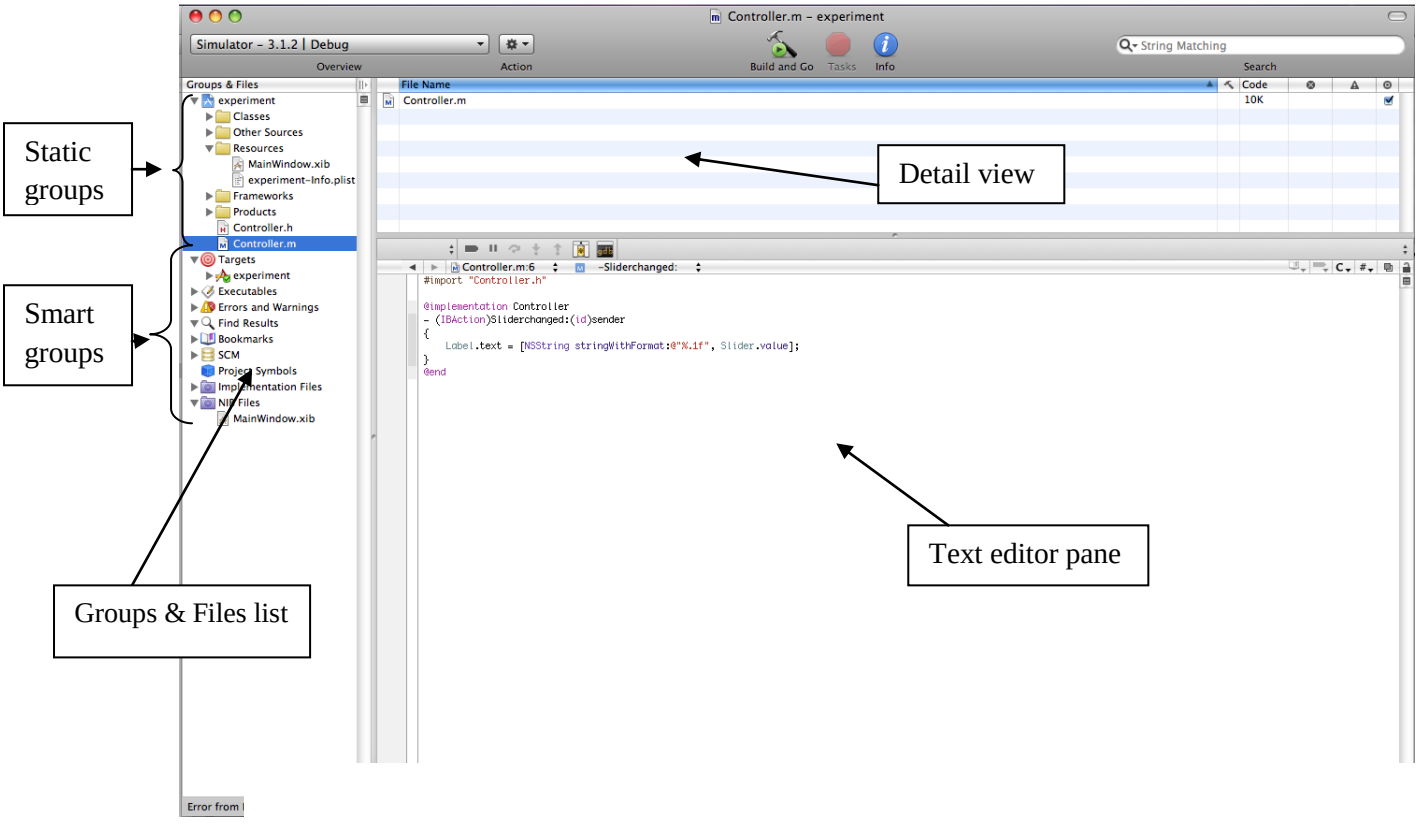

Figure 3.2 The Xcode text editor.

3.2.3 Interface builder

The interface builder is a visual design tool used to design the user interface of iPhone and MAC OS X applications [10]. The graphical environment of Interface Builder is used to assemble windows, views, controls, menus and other elements from a library of configurable objects. We can set attributes to these objects and establish connection between them with drag and drop functions and finally save the files in a special type of resource file called nib or the latest version, which we used in our application and it is called xib file. The extension xib is a short for the NextSTEP Interface Builder and all files with nib extension is Interface builder files. Now the question is how does Interface builder work with Xcode in order to make an iPhone or a MAC application?

Xcode and Interface builder are not the same thing. Xcode is a coding environment while the Interface builder is a visual design tool used to create new Objective-C classes and add outlets and action to the existing classes. Even though there is difference, Xcode and Interface builder are integrated tightly together and Interface builder can get back information about all objects (classes) in an application and make this information available to the programmer, who is working on projects associated with nib files. To make this integration possible, keep both Xcode and Interface builder running at the same time. It is easy to open Interface builder, because it is included in the toolset of Xcode. But we usually start by creating a new Xcode project by choosing File->new project from the Xcode toolbar and a window with different templates will appear. There are different templates for different purposes, see Figure 3.3. The user chooses the appropriate template by clicking on it and presses the bottom “choose”, and then another menu will appear to give the template a name and save it in a specified location. After creating the project, we will have many files in the detail view part in the Xcode window, among them we can find a xib file.

Interface builder template initially sets objects and action on these objects, so we can modify the template by add, remove, modify and display objects. In order to do that the Interface builder introduce four document windows to make these modifications possible and to open these document windows all we need to do is to double click on the MainWindow.xib file and the Interface builder document window and a window of the chosen template will appear. For

example if we choose the view template then a view window will appear when we double click on the MainWindow.xib file, see Figure 3.3 and Figure 3.5

The document window includes one or more objects that the user will need to create in run time in the user's application. Figure 3.4 shows a document window in the bottom bar. We can see the name of the created project, prov.xcodeproj. We can modify, add and remove objects in the templates and we do all this with the help of the tools menu in the menu bar of Interface builder. When we click on the tool menu a sub menu appears, there we can choose Library field and Inspector field. The Library field gives us the options of all possible windows we might need for our applications, for example we used many views to iPhone application and all these views can be chosen from Library field plus all objects that we might need to implement in our views, all kinds of buttons and all kinds of text fields. The mechanism of putting the object in a window or view is very easy, it is a drag and drop function. In the Inspector menu, we have four fields to give our objects or classes their identity, size, connections and attributes. First of all we have to choose an object by clicking on it and then click on the specific field in the Inspector menu. Now if we click on the button in our application window and then choose identity field, it will get the title button identity which means give button object a type and its action, which is a method to connect an action or event to the object. Then we have size field to edit the size of the current object. The connection field gives us the opportunity to connect the objects with its actions and this can also be done by using a drag and drop function. With the last field is the attributes field, which gives us the opportunity to edit the object for example, we can choose the background color and many other options.

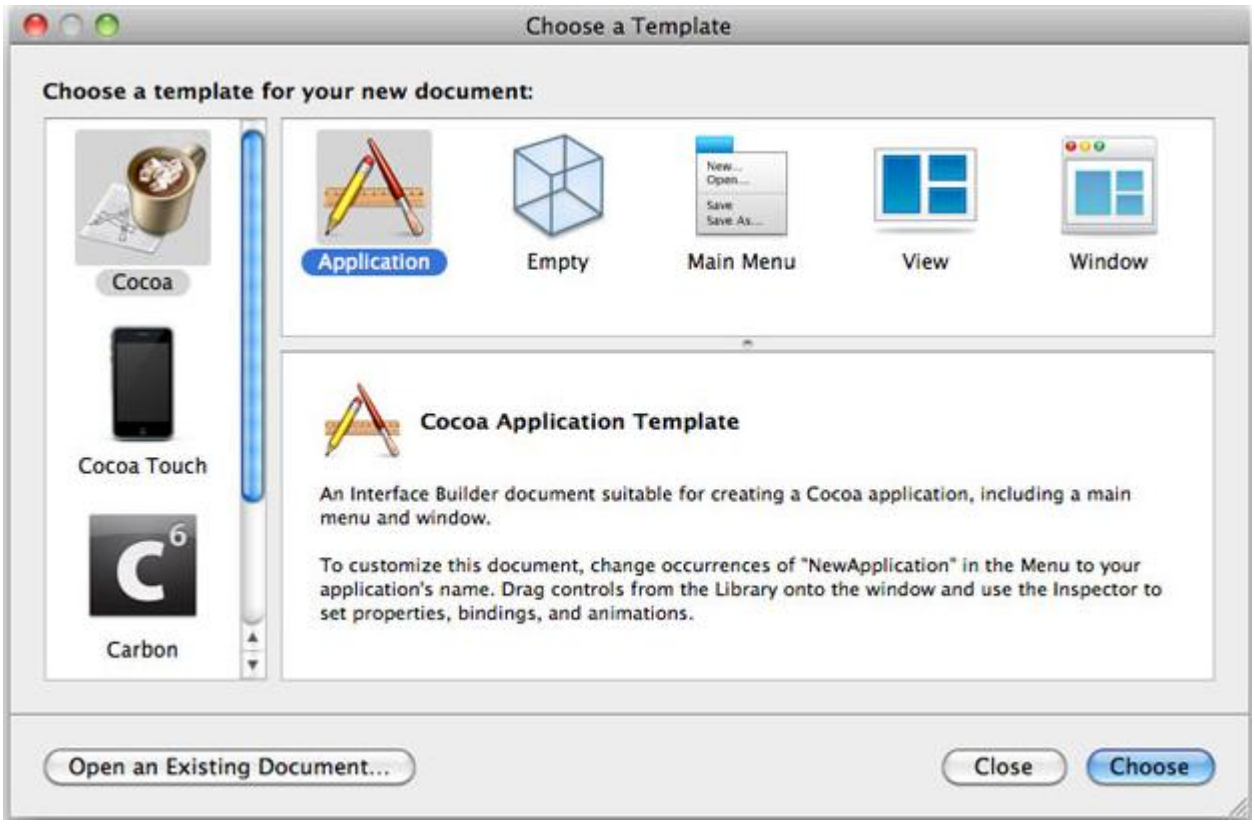

Figure 3.3 Window for choosing a Template.

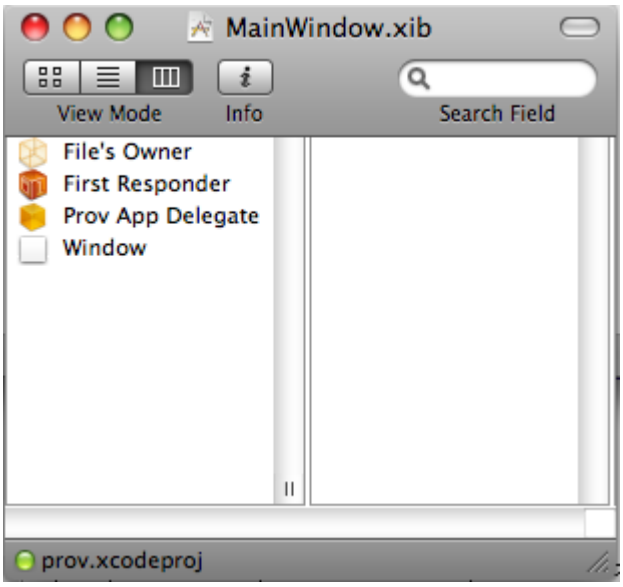

Figure 3.4 A document window.

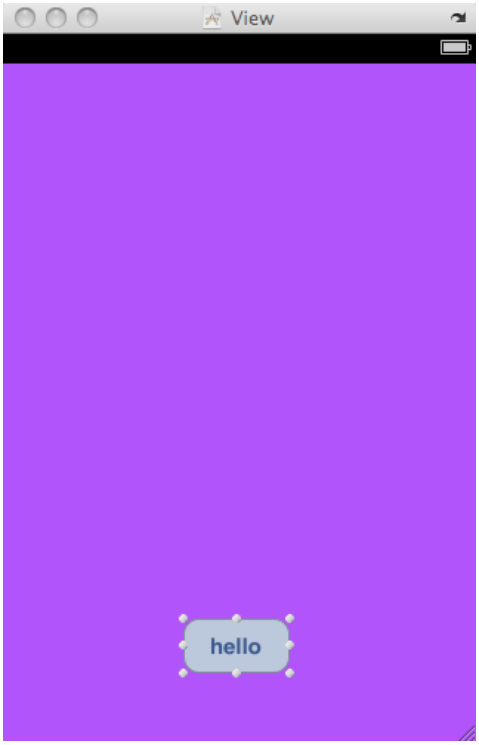

Figure 3.5 A view window.

3.2.4 Objective C

Objective C is a reflective, object oriented programming language which adds Smalltalk-style messaging to the C programming language. [11] Smalltalk is an object oriented dynamically typed programming language. Objective C is used today on apple's MAC OS X and iPhone OS. Objective C is also the primary language used for Apple's Cocoa API [12]. Objective C is a series of object oriented added to C programming. Objective C family includes the following members:

Objects: associate data and operations. The object is the root in the Objective family which means everything in general is object and in Objective C the root of all classes is defined as NSObject in the inheritance system.

Methods are the operations that apply to the data in Objective C.

Messages are the sending of a message from one method to another to perform an operation.

Classes are how objects are defined. Classes contain prototypes of all variables and methods implemented in a certain class. As we mentioned before in object description we have a root

called NSObject. All classes inherit from NSObject which makes it the super class and all other classes that inherit from NSObject are called the sub classes of NSObject.

Syntax in Objective C:

Objective C is a thin layer on the top of C, and it is a strict superset [13] of C. It is possible to compile any C program with an Objective C compiler. Objective C object syntax is from Smalltalk and all syntax of non object oriented operations including all primitive variables, expressions, function declarations and function calls are identical to that in C, but the syntax for object oriented features is an implementation of Smalltalk style messaging. In order to implement a class we have to create two files, one with the extension .h which we recognize as a header file from the C implementation, but in Objective C it is called Interface file and this file includes all the prototype of variables and methods which we implement. The syntax of Interface file begins with the prefix @interface, name of the class and semi colon then followed by the super class name. The super class here is NSObject and finally the Interface file will be ended by writing the postfix @end. Here follows an example to demonstrate a complete executable Interface file that we implemented while learning the language:

```
@interface Controller : NSObject
{
    IBOutlet UILabel *label;
    IBOutlet UISlider *slider;
}
- (IBAction)sliderchanged:(id)sender;
@end
```

Where label is an IBOutlet UILabel * and slider is an IBOutlet UISlider * and sliderchanged:(id)sender; is a method, where IBAction between two parentheses is the type declaration of the method and sliderchanged is the name of the method itself which is followed by colon and last thing is the parameters. The parameter begins with the type between two parentheses and then the name of the parameter. Note that the name of the parameter also is part of the method name. The full method name is therefore sliderchanged:sender.

The other file is called Implementation file with the extension .m. It includes all the implemented methods and in order to make the Implementation file work we have to import the corresponding Interface file. The Implementation file begins with prefix @implementation and ends with the same postfix as Interface file @end. We are used to have two kinds of methods in object oriented languages, the class method and the instance method, but in Objective C we have two different mathematical signs to represent these methods. The plus sign represents the class method and the minus sign represents the instance method as we can observe in the example bellow. The Implementation file in this example is the corresponding of the Interface file above and it is a complete executable file.

```
#import "Controller.h"

@implementation Controller
- (IBAction)sliderchanged:(id)sender
{
    label.text = [NSString stringWithFormat:@"%.1f", slider.value];
}
@end
```

These two files, combined with some settings in the interface builder, results in the application shown in Figure 3.6, where the user can move the slider to the left and right, which makes the value written to the right of the slider change.

Figure 3.6 A simple application in the iPhone simulator.

3.2.5 XML

Extensible Markup Language (XML)[2], describes a class of data called XML documents. XML is an application profile or restricted form of, the Standard Generalized Markup Language (SGML)[24]. By construction, XML documents are conforming SGML documents [2]. XML became a W3C-recommendation 10th of February 1998 [22].

Each XML document contains one or more elements. The boundaries of the elements are either delimited by start-tags and end-tags, or, for empty elements, by an empty-element tag. Each element has a type, identified by name and may have a set of attribute specifications. Each attribute specification has a name and a value.

A simple description of the XML-grammar:

XML ::= element (XML || NULL)

element ::= EmptyElemTag || StartTag content EndTag

StartTag ::= '<' Name Attribute '>'

EndTag ::= '</' Name '>'

EmptyElemTag ::= '<' Name Attribute ' />'

Attribute ::= NULL || Name '=' AttributeValue Attribute

Note that it is necessary for the Name of the start-tag and the end-tag to be identical. Name of a start-tag, end-tag or empty-element tag, specifies the element-type. The Name of an Attribute specifies the attribute-type. The same attribute-name may only occur once within an element. If an XML-document fulfills these requirements it is considered well-formed.

A specific XML type or structure can be defined using an XML Schema. The predecessor of the XML Schema is the Document Type Definition, (DTD)[22]. The DTD is still widely used since it is included in the XML definition, which gives ubiquity.

If a well-formed XML-document fulfills the requirements of the corresponding XML Schema or DTD, it is considered valid. These requirements typically include such constraints as:

- Elements and attributes that must/may be included, and their permitted structure.
- The structure as specified by a regular expression syntax.
- How characters data is to be interpreted.

3.2.6 C#.NET

To understand C#, we have to start with the .NET framework[20]. .NET framework is a system-component which is part of the operating system Microsoft Windows. .NET includes components for program execution and it also includes class libraries which contains solutions coded for many programming tasks like database management, web services and networks. .NET framework is standardized as CLI which stands for Common Language Infrastructure. .NET is simply a runtime environment and a base common class library. The runtime layer in .NET is referred to as Common Language Runtime (CLR), it is Microsoft's implementation for CLI and it is the runtime environment that runs all code and makes the development process easier. The main task of the CLR is to locate, load and manage .NET types on the programmer's behalf. CLR take care of a numbers of low level details such as automatic memory management, language integration and ensuring type safety and it has two fea-

tures, the first is Just In Time compiler (JIT), it manage to run all the code in the native machine language of the system on which it is executing. The second one is the garbage collector which allocates and deallocates the memory automatically so the programmer does not need to take care of memory allocation manually like in C and C++. Another block of the .NET framework is Common Type System (CTS). The CTS describe all the possible data types that are common to .NET and it support the type converting of data and communicating between two languages in the .NET family which are the same in basics but different in syntax[20].

When we develop a program or an application using C# we have to compile the code in order to make it understandable to the operating system to translate it to the machine language and execute it. That means we have to deal with complex operating system details. But .NET framework helps the programmers to implement and develop applications easily. For example if we write a function that generate a random number. If we compile this function using .NET framework the only thing we need to do is to create a class, write the random function and include .NET framework and execute the program. The .NET framework will take care of both translating the program to machine code and will also take care of all mathematical operations that the program needs to calculate the random function, which means that will save time and effort for the programmer [21].

Many classes have similar functionality and not just in C# but all languages that support .NET these classes could be grouped together under what is called namespaces. SDK documentation includes the whole list of the namespaces for the .NET framework. If we know the namespace which the class belongs to we do not have to specify it with each method call. For example if we want to write a line from the keyboard to the console window we write it simply by including the Using.System namespace and then implement the Console.WriteLine(); because the Console class is grouped in the System namespace and by including the namespace once we avoid to write every single time we need to implement the Console class in our code [16].

ASP.NET

ASP.NET is a web application framework developed and marketed by Microsoft. It is used to build dynamic websites, web applications and web services and it is used to support a variety of other languages including Visual Basic and C#. The main purpose of ASP.NET is to create webpages and linking them to a database. In order to program webpages in ASP.NET, the

developer has to know the basics of HTML, XML and XHTML. ASP.NET has some features that make it a good web applications technology and these are:

- **Use of Controls:** ASP.NET controls to provide basic and advanced operations which are server controls. Server controls are tags that are understood by a server and there are three kinds of server controls.
    - **HTML controls:** Traditional HTML tags. These controls are executed by the client and include textbox, label, image etc.
    - **HTML server controls:** are the same as the HTML control but they are executed on the server side rather than the client side.
    - **Web server controls:** ASP.NET tags. Web server controls are executed on the server side.
    - **Validation server controls**: For input validation.
- **Tools:** ASP.NET pages can be made in a variety of tools. The most basic editors are WordPad or Notepad, but Microsoft Visual Studio provides the developer with tools which makes it a lot faster and easier to create the desired ASP.NET pages.

3.2.7 IIS

Internet Information Services (IIS)[8], formerly called Internet Information Server, is a web server application and set of feature extension modules created by Microsoft for use with Microsoft Windows. IIS is built-in in the Windows Operating system, but it is not turned on by default and can be selected from the list of optional features. IIS is not available in Windows XP Home Edition, but it is available in all later versions of Windows and in Windows XP Professional. As of March 2010, 17.94% of all active servers were Microsoft IIS servers according to Netcraft [9]. This makes it the world's second most popular web server in terms of overall websites, behind the industry leader Apache HTTP Server.

IIS makes it easy to set up a web server to run in the background on your PC.

3.3 Summary

In this chapter we have reviewed the tools and environments used in this project. SDK is a software tool from Apple, used to implement and develop an application for iPhone. The SDK consist of four layers. Each layer includes a certain number of frameworks. These layer's frameworks correspond to libraries that we can include in our code. The purpose of these frameworks is to reduce the amount of code lines and make the coding more effective. Interface builder is also one of the effective tools that can be used to develop a GUI for an iPhone application. It is similar to Visual Studio in the sense that both uses drag and drop functionality to create the GUI design. But in Visual Studio we do not have to manually generate the code for GUI components and as soon as we make any changes on any of the GUI's components, the code will be automatically be updated. When we double click on a component, the code for clicking on that component is generated instantly. Interface builder does not work as easy as Visual Studio, when we create a GUI we need to first declare the names and types for all the objects we created in the GUI and we have to declare all events that are connected to these objects and then we have to use the mouse to drag a connection between these components in the GUI and their declarations names otherwise no code will be generated.

4 Chapter four: Security aspects

Always when transmitting information over a network it is important to consider the security aspects. Who might be able see what we are sending? Can the information be sensitive in any way? In reality almost all information can be sensitive in some way. Therefore it is always important to communicate in a secure way. It is reasonable to spend money and effort on security in proportion to the sensitiveness and value of the information being sent and the likeliness that someone, unauthorized would try to steal the information.

4.1 Cryptography

The basic idea of cryptography is to transform a message into a form, unreadable to anyone who doesn't have the correct key to decrypt the message. This way we do not need to worry about eavesdroppers or man-in-the-middle attacks, since no one except the intended receiver will be able to understand the message even if they can intercept it.
Since we are transmitting sensitive information (e.g. information about the customers and how much they are charged for different services), it is important for us to ensure that the information is securely transmitted between the iPhone and the server. Since we cannot guarantee that the iPhone will be used on a secure network, we will need to use cryptography.

There are several known and commonly used algorithms for encrypting messages. Some are more advanced, which means they will take longer to encrypt and decrypt, but they will be more secure. And others are simpler, they will be faster to encrypt and decrypt, but offers less security. We would have to choose an algorithm that corresponds to our need for security.

An encryption scheme is said to be computationally secure if the cost of breaking the cipher exceeds the value of the encrypted information or if the time required breaking the cipher exceeds the useful lifetime of the information.

Encryption algorithms are divided into two categories, Symmetric encryption, also called conventional encryption or single-key encryption, and Asymmetric encryption, also called public key encryption. [15]

Symmetric encryption uses the same secret key to encrypt plaintext into ciphertext and to decrypt ciphertext into plaintext. Encryption algorithms of this category are often much faster to encrypt and decrypt, but can still offer good security. A problem with single-key encryption is

the distribution of keys. Encrypted messages can only be sent when the sender and the recipient share the same secret key. In our case, all users will be within the same company, so it would be possible to distribute the keys manually, for example on a piece of paper. But it would be much better if the user didn't have to handle the key manually. Usually key exchange protocols are used to distribute the keys in a secure way.

With asymmetric encryption, one key is used to encrypt a message and another key is used to decrypt the message. These two keys make a pair, where the first key is needed to decrypt a text, encrypted with the second key and the second key is needed to decrypt a text, encrypted with the first key. Each user has a pair of keys, a public key and a private key. The private key is kept secret and the public key is known to everyone.

Public key encryption solves the problem with key distribution. If Alice wants to send a message to Bob, she can encrypt the message using her own private key to prove that she is the sender and then encrypt the result with Bob's public key to ensure that only Bob can read the message. The only problem is that it may be difficult for Alice to verify that the public key really belongs to Bob.

Asymmetric encryption is generally much slower and their key sizes must be much larger than those used with symmetric encryption.

4.2 HTTPS

Hypertext Transfer Protocol Secure (HTTPS)[4], is a combination of regular HTTP and Secure Socket Layer (SSL)[6] or Transport Layer Security (TLS)[5], which both are protocols for providing encryption and secure identification of the server. HTTPS creates a secure channel over an insecure network and is often used for payment transactions on the Internet.

SSL is the predecessor of TLS and was developed by Netscape Corporation and first released in 1995. TLS is an IETF standards track protocol, last updated in RFC 5246 (August 2008). [4][5][6] TLS is based on SSL. TLS/SSL allows clients and servers to communicate over the Internet using encryption. Typically only the server is authenticated, but there is also support for bilateral authentication. Authentication is done using a certificate, issued by a Certificate Authority (CA). Security relies on the clients trust in the CA.

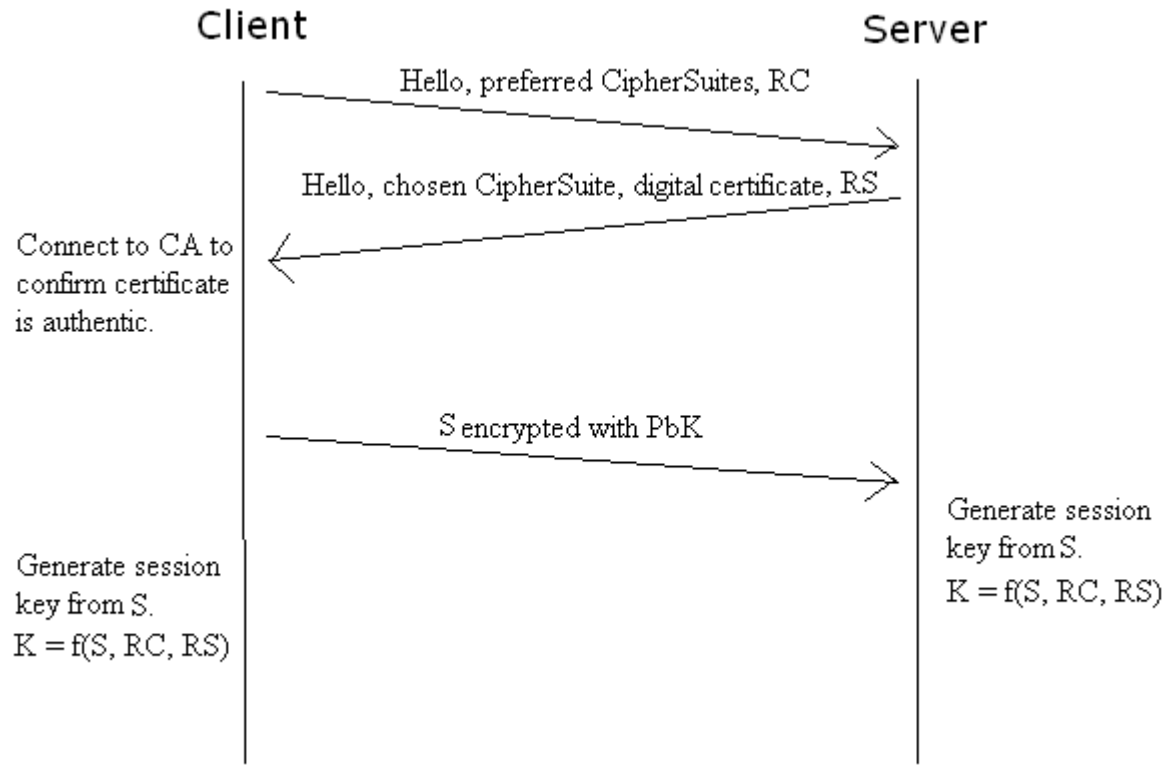

Figure 4.1 Shows the TLS handshake.

- First the client sends a **ClientHello**, which specifies the Cipher Suites (ciphers and hash functions) that are supported and a random number RC.
- The server responds with a **ServerHello**, which contains a digital certificate. The certificate specifies the server name, the Certificate Authority (CA), the server's public key (PbK) and a random number RS.
- The client connects to the CA to verify the authenticity of the certificate.
- The client generates a random number (S) and encrypts it with the server's public key and sends it to the server.
- Both the client and the server generate the session key from the random number. This is done using the function K = f(S, RC, RS). The function f depends on the chosen Cipher Suite.

In our case the use of HTTPS would provide secure communication between the iPhone and the server without the need of implementing some advanced encryption algorithm. HTTPS is already implemented and ready to be used. Note that TLS authenticates the server, not the client. Therefore an authentication method is required.

4.3 Basic Access Authentication

Basic Access Authentication [7] offers a way to pass username and password in the header field of an HTTP-message. These credentials can be use to authenticate the client on the server side. The credentials are written in a string with the format: “username:password”. This string is encrypted using a base-64 encryption and added to the header field of the message. The base-64 encryption does not offer security. It only converts the string with the username and the password into a new string with only characters that are supported by HTTP/HTTPS, so that the credentials can be transmitted correctly. Since the basic access authentication sends the credentials over the network in “clear text”, it is necessary to use it together with some encryption. Our solution is to combine basic access authentication with the use of HTTPS.

4.4 iPhone security

The need for a secure device increases as Apple intends to take over the business phone market. The founding idea of iPhone security is to keep each application running in its own sandbox environment. Only one application at a time is allowed to run. This way, a malicious application cannot reach the data stored with other applications and can therefore not do much harm.

Charlie Miller, Jake Honoroff and Joshua Mason describe [3], how they found a way around this in 2007. They used a harmful website to force the iPhone to send any desired information, including passwords, e-mails and text-messages, to the attacker. They delivered all details about the weaknesses they had found to Apple, before they published their evaluation article.

As it is usual with security, there is a race between Apple who increases their security and some hackers, who try to find new weaknesses in the security. Even though Apple made the proper adjustment to secure the specific weaknesses found by Miller, Honoroff and Mason, others have found other weaknesses, which later have been taken care of by Apple. This race will probably go on for a very long time.

In June 2009, Apple released a number of increased security features, intended for corporate use [17]. One of the features is Remote Wipe, which enables the owner of the device to remove all data from the device if it is stolen. However, a thief that wanted to steal information about the company could just remove the Sim-card to disable this feature.

It is also possible to set the iPhone to remove all data after several failed password attempts. But an advanced enough intruder will be able to also retrieve removed data from a locked device. Because of the large amount of disk space, it takes a long time before removed data is overwritten.

When used in a company, the administrator can setup device restrictions and configurations over the air. It is possible to set passcode policies such as:

- minimum length
- maximum failed attempts
- require both numbers and letters
- inactivity time in minutes

The iPhone OS uses a keychain to store passwords, keys, certificates and other secrets in a secure way, using encryption. Every application has access to its own keychain items.

4.5 Security decisions in our prototype

Our application is dealing with a lot of sensitive information, for example what customers Ninetech has and how much each of them is charged. And the usernames and passwords used to login are the same as the ones used to login at the rest of the company's computer system. Therefore the security is very important in this application.

Since we decided to use basic access authentication, the username and password is sent in clear text, which means that we need to have some additional encryption. We choose to use HTTPS/SSL to encrypt the communication between the server and the client. On the server side, the security aspects have already been dealt with, since we are using an already existing server at Ninetech. On the client side, we have decided to trust the encryption of the iPhone keychain to be secure enough to store the username and password of the user. We do realize that this is a compromise of the security, but the benefit of not having to reenter the password every time outweighs the slightly decreased security.

This decrease in security consists in the fact that the username and password is stored in the device. Even though they are encrypted, there is a way to retrieve them. Obviously the iPhone itself decrypts the username and password when they should be used. That must mean that the information required the decrypt this information is also stored somewhere inside the device. An attacker, that stole or found the device could try to find a way to force the iPhone to decrypt the username and password itself or he could try to do the decryption himself, possible with help of a key from the iPhone. Or the attacker could simply use the application and pretend to be the user that has stored his login information.

To prevent anyone to report time if they find a lost iPhone, we have planned to add a PIN code to the application. Our constituent wanted us to apply the password storing already before the PIN code functionality has been completed. This obviously means that the security of the application is further decreased. But until the PIN code has been implemented, Ninetech has decided to trust the device's PIN code function.

5 Chapter five: Prototype development

5.1 Work method

It is always important to have a good strategy before starting to work on a project. Planning the work well will decrease the risk of finding out halfway through the project that one has been working with the wrong things. Or realizing at the end of the project that the most important part of the project was never finished.

Our first plan was to view the project in layers, with the GUI (Graphical User Interface) at the top, the business logic as a second layer and the database as the third (See Figure 5.1). We planned to start at the top and work our way down. It made sense to start with the GUI, since we had not yet received the required information about the underlying structure at that point. We wanted to finish the GUI while we were waiting for the information about how we could reach the database through the business logic.

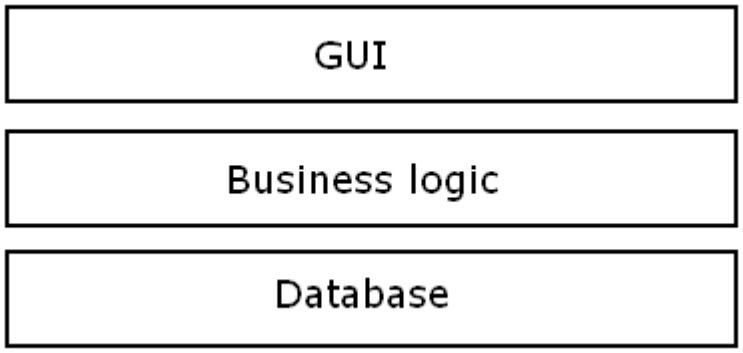

Figure 5.1 A model of the project.

Our mentors at Ninetech then proposed another approach. We were to divide the project into "straws", where each straw went all the way from the top to the bottom, through all three layers. (See Figure 5.2) The first straw would be the login procedure.

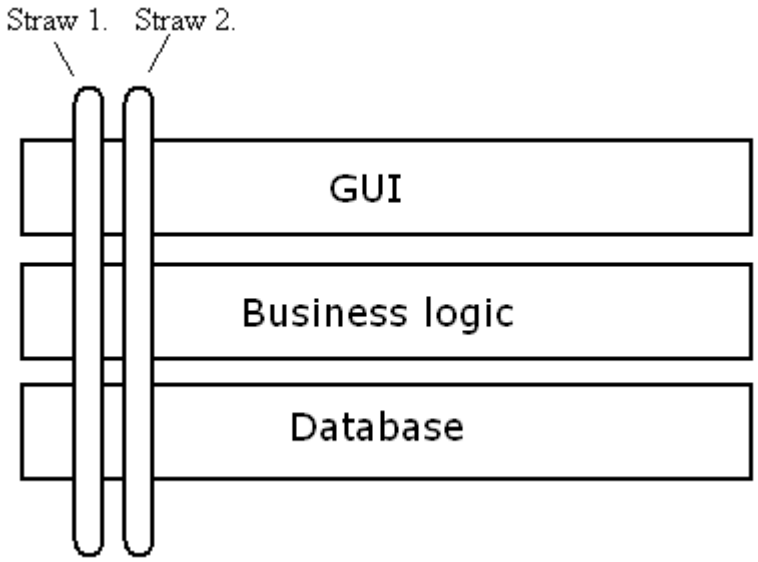

Figure 5.2 A model of the project, using straws.

We liked the idea and decided to use it. This new approach meant that we were in urgent need of the underlying structure of the system. So we got Ninetech to speed up the work of getting the needed information.

As a first step they helped us to setup a mockup server, using Microsoft Visual Studio. This server worked like the real server would do, but it always returned the same answer regardless the input. This gave us a server to connect to using our iPhone simulator. When we had managed to connect to the server we added the need for login authentication using the actual Ninetech user accounts. When the login had succeeded we had finished the first straw and could start the next one, which would be to retrieve a list of customers, projects and activities and let the user choose from them. The third straw would be to retrieve time-types and price-types for the project and let the user choose from them.

This way of working meant that the first straw would require a large effort, since we needed to find out how to go through all layers to get the connection we needed. The second straw was also a challenge, because we needed to go further than we had done in the first straw. The third straw however did not require as big an effort, since it was very similar to the second straw, therefore we could use the experience from the second straw to complete also the third straw.

5.2 Prototype description

Our prototype consists of two parts. The iPhone application itself and the server interface. The server also has a backend, but it is controlled by Ninetech and kept secret. That part is therefore not part of our prototype. The backend is where the actual connection to the database is made. Our only way to access the database is to use stored procedures, which are basically hidden functions in the backend, which makes the database queries for us and returns the corresponding result. This way the database is kept safe and unauthorized users or developers, such as our selves are unable to make illegal database queries.

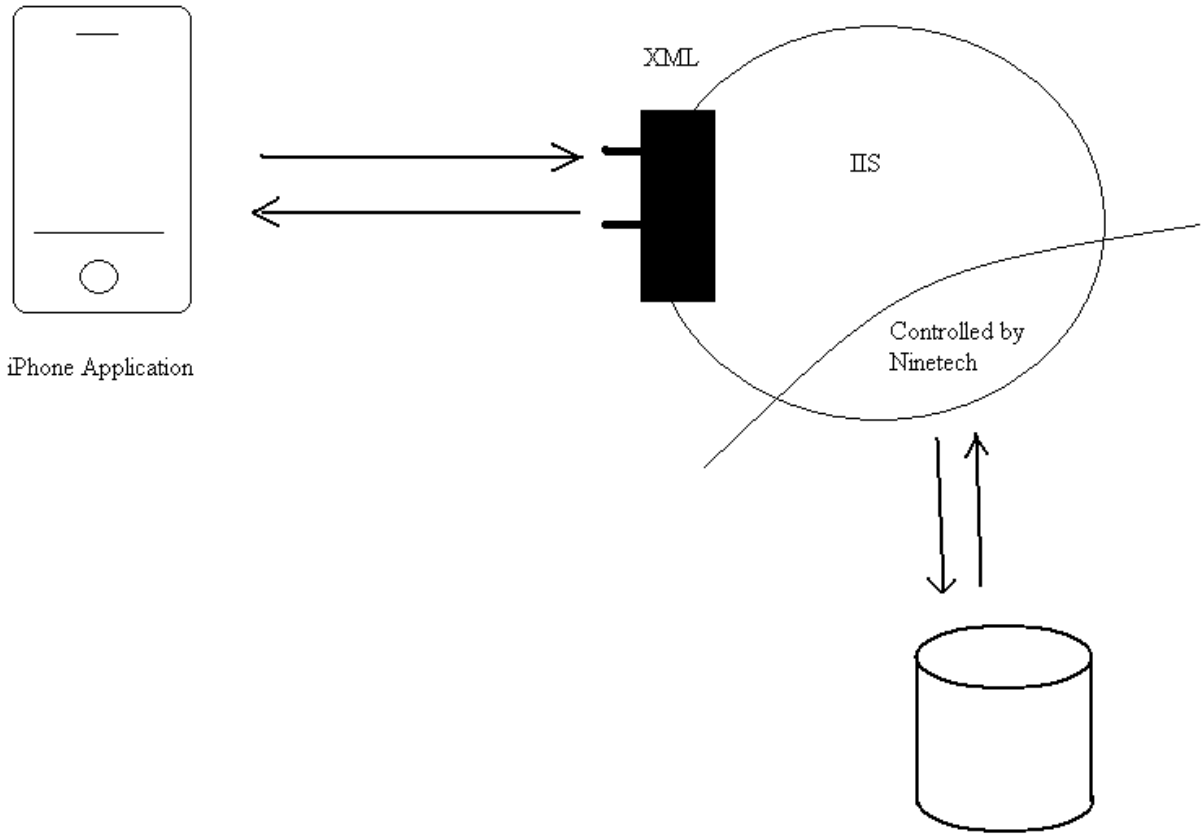

Figure 5.3 Overview of our prototype.

The purpose of using the application is to report a new timeline into the JEEVES database from the iPhone. The first thing we need to do is to identify the user. Luckily there is already a login function on the server system. We prompt the user for username and password and use basic authentication to access the login function on the server system (step 1 in Figure 5.4). If the user is authenticated, he receives permanent access rights (step 2 in Figure 5.4). These access rights are deleted when the user choose to log out. Now that we have both authenticated and identified the user, we need to get a list of all customers the user works for and all projects the user is involved in (step 3 in Figure 5.4). This can be done on the server side with

a stored procedure. The result from this stored procedure is saved in four arrays. One for the customers identification numbers, one for the customers names, one for the projects identification numbers and one for the projects names. These four arrays are sent back to the iPhone application in XML format as shown in step 4 in Figure 5.4.

The iPhone application has built in functions to receive and parse the XML. The data in the XML is first stored in four arrays, similarly to the way they were stored on the server side. Then the arrays are used to create a dictionary to map the names to the IDs and another dictionary to map each customer to an array of the projects this user is involved in with this customer.

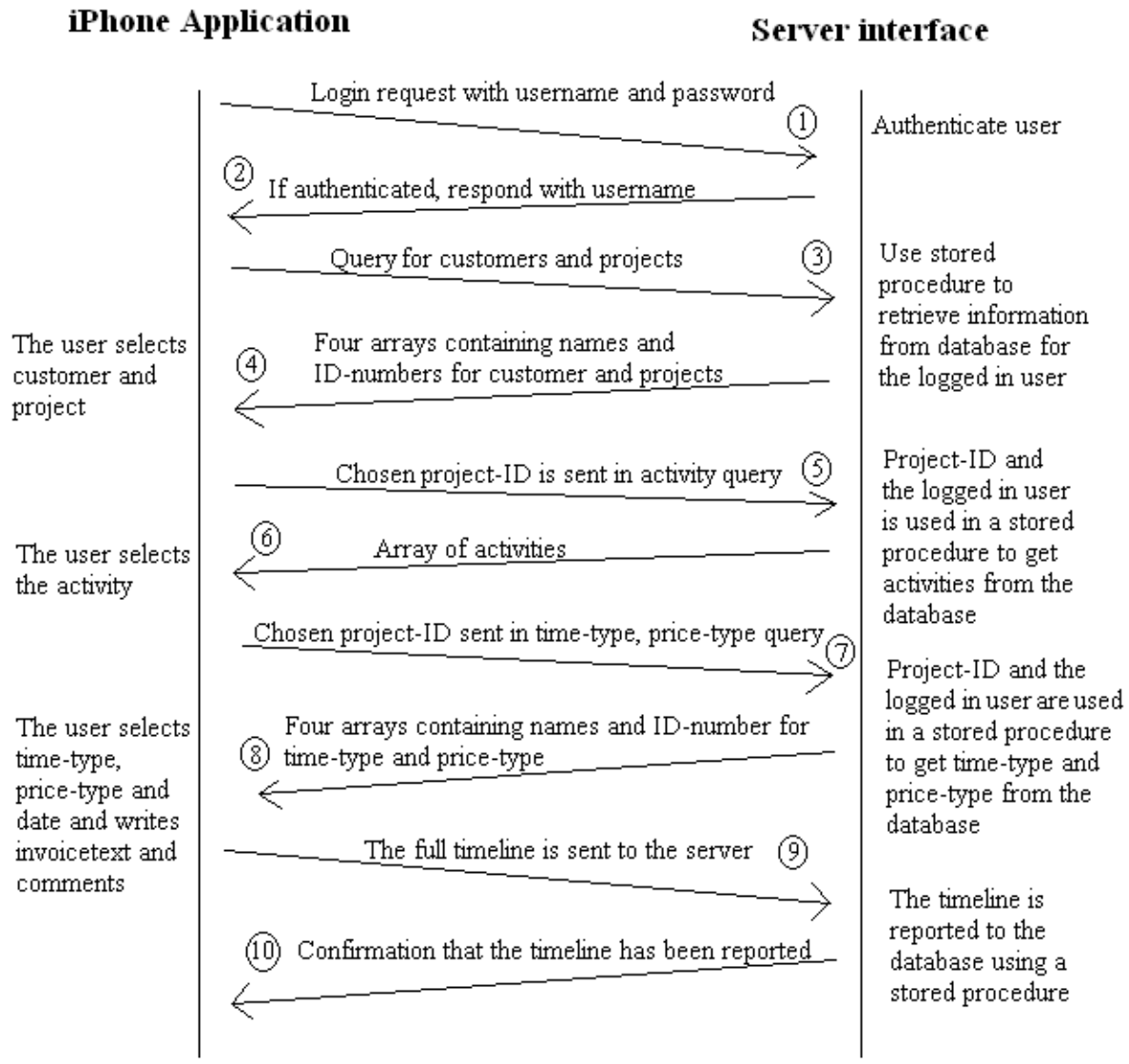

Figure 5.4 The messages sent between the application and the server.

When this information is stored properly, a new view is pushed in the iPhone application. This new view displays a list of all customers for the user. When a customer is chosen, another view is pushed, displaying a list of all projects for that user and customer.
When the project has been picked, the application needs to make a new call to the server to retrieve a list of all activities the user can perform in this project. There is a stored procedure also for this and that can be called from the application, with the project identification number as a parameter (step 5 in Figure 5.4). The stored procedure also takes the username as a parameter that can be retrieved on the server side, since we know which user has logged in. The call to the server is done by sending an XML-file containing the function name and the parameter. The result of this stored procedure is saved as two arrays: One for the activities identification numbers and one for the activities names. These arrays are serialized using XML-format and sent back to the iPhone application (step 6 in Figure 5.4).

Once the new data is received, a new view is pushed in the iPhone application. Here we present a list of all activities for the current user and project. When the activity has been selected, we make a new call to the server (step 7 in Figure 5.4). This time we want to retrieve a list of all available time types and prices types for the selected project and user. This is done the same way as with the previous calls, with stored procedures, sent over XML. Also the time types and price types has an identification number and a name. There is also a mapping between the price type and the time type. Every price type has a given number of possible time types. When this data has returned to the iPhone application (step 8 in Figure 5.4), a view with the price type list is displayed and after selecting the price type, the time types are displayed the same way in a new view. After selecting the price type, another view is pushed, where the user can specify the date to report a timeline for, the date of the actual day is the default. On the same view the user sets the number of worked hours and chargeable hours, using two sliders. When the user hit next, he reaches the last view, where he can write invoice text and comments and hit send to make the final call to the server (step 9 in Figure 5.4). This final call is another stored procedure that takes all ten selected attributes as arguments and writes a new timeline to the database.

Design

In every GUI (Graphical User Interface) the design is a very important aspect to deal with prior to any code writing, because the GUI gives the user his first impression of the application. When developing a GUI, the programmer should always aim to make it both simple and easy to use. The programmer should always keep in mind that the user may not have comput-

er skills and solutions that may seem obvious to the programmer might not be obvious to the user. Also users with good computer skills like to have a simple and comfortable GUI to use. This was the first demand from our customer in our first demonstration.

Figure 5.5 The first view in our first suggestion for a GUI.

Figure 5.6 The second view in our first suggestion for a GUI.

Our first layout of the GUI was not as simple as our constituent wanted it to be, see Figure 5.5. In our first design for the GUI, we had three buttons to represent three lists of information the user needed to provide to report his time to JEEVES database. The first button, named "Kund", is used to open the customer picker. This picker list includes all companies that exist for the current user in the database, so the user can pick a name from the list. The second button was for all the projects that exist in the database for the chosen company and the user can choose a project in the same way that the company name was chosen. The last button was for activity. That button opens the activity list for the chosen project in the picker. There was also a textbox to fill in the name of a new customer, project or activity in case one of these properties was not available in the database. We had a similar solution for the date, time type and price type choices, with three buttons for the date, time type and price type connected to a picker. See Figure 5.6. Our mentors at Ninetech preferred to have a larger number of views, with less information in each view, since the iPhone has a very small display compared to a computer screen. At this stage of the project we decided to make the GUI design simpler and in order to achieve this, we used a higher number of views in our GUI for the application.

Figure 5.7 The first view of our application.

Figure 5.8 The second view of our application.

The new design of the application includes ten views and most of the views have similar design. The first view is titled login, and has a very simple layout, it consists of two UITextLabels and a button as shown in Figure 5.7. The UITextLabels are for writing the username and password. As soon as the user has filled in the authentication information and touch the login button, the application sends an authentication attempt to the server. If the authentication succeeds, the second view will appear. If not, an alert window will appear to warn the user that either username or password is wrong and ask the user to try again. The second view, seen in Figure 5.8, includes a UITextLabel to display the username returned from the server to ensure the user that he has logged in. There are two buttons in the second view. The ″Ny Tidrad″ is to move to the third view, which displays all customers the user is registered at. The ″Logga ut″ button in the second view is to logout the user. The third view (See Figure 5.10,) displays all customers the user is registered at. Here the user has to pick the desired customer and the fourth view appears. That is called ProjectView. In ProjectView the user select the desired project from a list of all projects this user is registered at for the chosen customer. The fifth view is called ActivityView and includes all activities the user is registered at for the chosen project and customer. The sixth view is called PriceTypeView and here the user can choose the right price category for the worked time. The next view is called TimeType and enables

the user to select the time category to report (e.g. Normal time or Over time). The third through sixth view are all UITableViews. The next view is called HoursAndDate and here the user selects the number of worked hours and the date. The design of HoursAndDate, shown in Figure 5.9, includes two UISliders which are used to choose the worked time and chargeable time. When the user moves the worked hours slider the charged time slider will follow it, but if the user changes the chargeable time slider the worked time slider will not follow this change. The last view in the application is Comments and includes two UITextLabels and one button, one of the UITextLabels is for the comments and the second one is for the invoice text. The button is for sending in the completed timeline to the database through the web server by using a HTTPS request. A timeline includes the following information: Customer, Project, Activity, PriceType, TimeType, worked hours, chargeable hours, date, comments and invoice text.

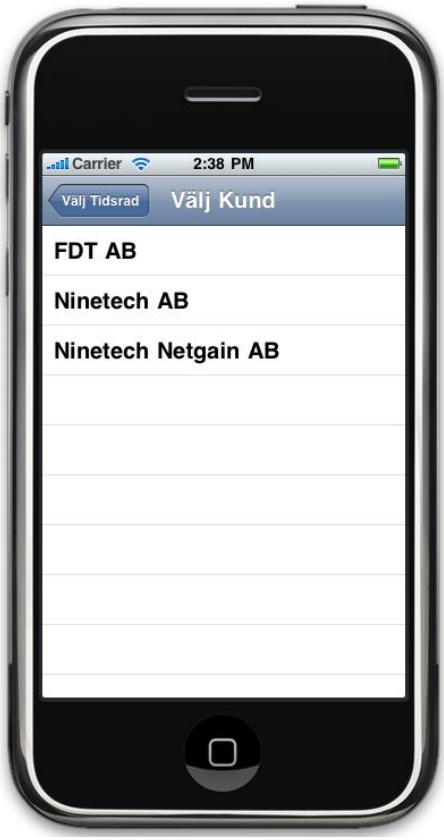

Figure 5.10 The third view of our application.

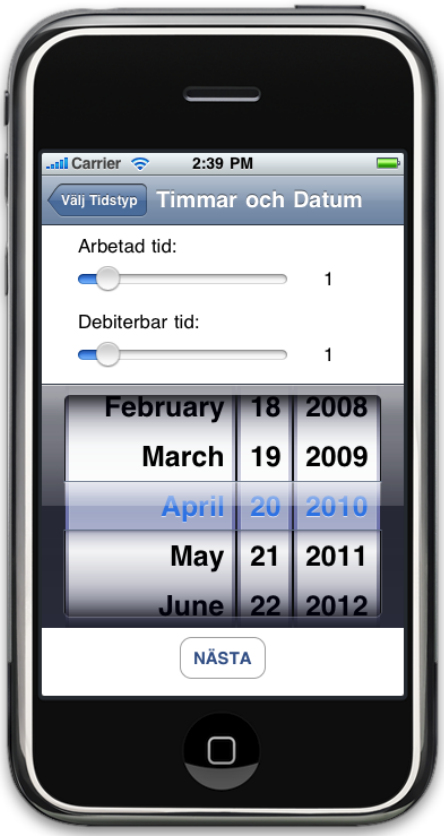

Figure 5.9 The HoursAndDate view of our application.

5.3 SUMMARY:

In this project we have gone through several steps on the way to get an iPhone application that communicates with a web server that reads and writes to a database (See Figure 5.3). Our first thought was to divide the project into layers and work with one layer at a time (See Figure 5.1). However, our mentors at Ninetech introduced us to the idea of working with "straws" like in Figure 5.2. This idea is based on finishing the entire first step, in our case the login, through all layers. The advantage with this way of working is that we can start testing the first part of the finished product at a very early stage. And if we should run out of time before we can finish the product, we will at least have something fully functioning even if some part of the product is incomplete.

One issue was how we could get information from the database without having direct access to the database itself. We decided to use stored procedures to retrieve information from the database. Authorized personnel at Ninetech wrote the stored procedures we needed. This way we could query the database without knowing the actual structure of the database. We are therefore unable to make illegal queries, intentionally and unintentionally.

The XML format has been used to send data between the client and the server. The client parses all data and stores it in dictionaries. All necessary data on the server side has been stored in arrays and dictionaries on the client side. The dictionaries are used to map the names with the IDs.

The GUI design is an important issue in every application, because the GUI is all the user sees of the application. We designed our application so that the user will stay logged in until he clicks the logout button. The aim of this procedure was to increase the accessibility of the application. Users with long passwords or usernames do not have to login every time, to report their worked time. They can stay logged in until they click the logout button. There is an obvious decrease of security involved, when storing login information inside the device, even when the information is encrypted. The device can be lost or stolen. But the small size of the touch keyboard of the iPhone makes it difficult to write longer words and especially if they involve a mixture of upper and lower case letters. This procedure was implemented on demand of our constituent because they knew how long their passwords are and it would be inconvenient to have to login every single time they need to report a timeline. Our constituent decided to trust the built in security tools in the iPhone, namely encryption in the keychain and a PIN-code to unlock the device.

6 Chapter six: Interviews

6.1 Introduction

A part of our project was to interview a number of employees about iPhone usage and what they would like to use an iPhone for at their work. From this we should create a prioritized list of future applications to implement for use at Ninetech.

Rather than selecting interviewees by random selection, we chose to only use volunteers. This was simply because we were not interested in finding out how many of the employees wanted an iPhone or if they thought an iPhone would be useful for them by using statistics. We were only interested in getting as many ideas for applications as possible. Therefore we assumed that those that did not volunteer for the interview were less likely to have good and creative ideas for applications.

Our idea was to primarily ask questions that would give much room for the interviewees to think and to do the talking. For each of the interviews we had four basic questions and depending on the answers we asked attendant questions. Our goal with every question was to get as many ideas for applications as possible. Our basic questions were:

1. Are you using an iPhone today?
   a) If yes, What are you mostly using it for?
   b) If no, Why not?
2. How would you like to use an iPhone in your work?
3. What Internet based services do you use in your work? Could they be used with an iPhone application?
4. Can you think of any additional application that would be useful for the company?

The result differed from case to case, but some ideas were more frequent then others. Some examples of the ideas are:

- Alert system to help the service desk to know about the problem before the customer does.
- Booking of conference rooms and other resources.
- Instant messaging within the company.
- Possibility to see information about Ninetech employees including pictures.
- Report travel expenses and other expenses in the work.
- See everything media is reporting about Ninetech.
- See statistics. Including totally worked hours on a project.

- Synchronization between iPhone calendar and Microsoft Exchange.

In the following two subsections we will first go through the different suggested applications to be implemented in the future and give some thought to how they could be designed. Thereafter we will sort them in prioritized order.

6.2 Application ideas

Alert System

The help desk could have great use of an application that would give them an alert call when there is a problem with a system. Today they are using a pager to warn them if some system goes down. The problem with the pager is that it doesn't give any information about the problem. The person that receives the warning has to get to a computer to be able to see what the problem is. With an iPhone application it would be possible to see what is wrong instantly, even if it is far to the nearest computer. It might also be possible to check the status of the system even when no warning has been given. Maybe it could all be shown in some clever graphical display. Such an application could enable the help desk to know about a problem before the customers do.

The question is however whether an iPhone application is able to receive alerts even if another application is running on the iPhone. The basic rule with the iPhone OS is that only one application at a time is allowed to run. There are no background processes. This offers security in the sense that a malicious application is unable to reach the other applications. But it also limits what is possible to do on an iPhone. However, if the help desk personnel keep using their pagers, an iPhone application could still be a great compliment. The pager would warn them and let them know that it is time to open the iPhone application to see what is wrong. It should also be possible to see system status and performance when no alert has been given.

Booking of conference rooms and other resources

Many of the interviewees expressed the need to be able to see what conference rooms are available without having to go to the computer and log into outlook. It would be a lot easier to pull up the iPhone from the pocket to see the bookings and even book the conference room or resource you need. This could be solved with an iPhone application. We investigated the possibility to solve the problem without developing a new application. One could simply synchronize the iPhone calendar with the outlook calendar. The problem is however that it is only possible to synchronize your own outlook calendar with the iPhone. You can therefore not see the calendar of the conference rooms. A new application is required to fill this need.

Expenses reporting

Employees and consultants that travel a lot find it difficult to remember all their expenses so they can report them when they get back to the office. Therefore they could make great use of an iPhone application that let them report their expenses at once when they get them. It could also be possible to scan receipts with the iPhone's built in camera, to send in along with the report. This application could be used for reporting all kinds of expenses.

Information about Ninetech employees

When the company is growing it becomes harder to keep track of all your co-workers. This application would provide information about all Ninetech's employees, including a picture, role at the company, personal information and contact information. It would also be useful if this application enabled the user to search for specific competence and certificates within the company. The application could also show who is in the office at the moment.

Maybe this application could be combined with the application for sending instant messages within the company, which was proposed by some of the interviewees. There are however many existing applications for instant messaging that could be used instead of implementing a new one.

Ninetech in Media

This application would search the web sites of all major and local media corporations for content about Ninetech. And display the result, preferably as downloaded articles or alternatively as url-links to the different webpages. This should be done automatically, but with the possibility to make manually changes to the result of the searches.

Project statistics

Primarily the team leaders expressed the need to view statistics of projects, for instance the total number of worked hours on a project.

Synchronization between iPhone calendar and Exchange

Some of the interviewees wanted the possibility to synchronize the iPhone calendar with Exchange and Outlook. There is already a way to do this, but it is not very user friendly. To do this the user needs to have Apple iTunes installed on the computer and connect the iPhone to the computer via a USB-cable each time the synchronization is to be performed. It would be better if the synchronization could be made automatically over the Internet. However the work needed to create an application that could solve this is rather big compared to the benefits it would give. Therefore this application will be put lower in the priority list.

6.3 Prioritized list

After careful consideration, this is how we arrange the priority of the different applications to be implemented:

1. **Expenses reporting.**

   This application was proposed by many of the interviewees, in slightly different versions. The implementation should resemble our application in the sense that the user gives some input that should be reported to a JEEVES database.

2. **Booking of conference rooms and other resources.**

   This was also one of the more popular suggestions and it is easy to see the use of this application to many in the staff. Before implementing this application it might be worth investigating if the task could be solved with some existing application.

3. **Alert system.**

   This application would be of great use for the helpdesk personnel. Unfortunately it is unlikely that it could completely take the place of the beeper, due to the iPhone's limitation to one active application at a time.

4. **Information about Ninetech employees.**

   A fairly simple application. The easiest way implement this application would be to store all the information in the application, but a better solution would be to connect the application to a server and download the desired information from there. This would provide scalability and make it easier to update the information through some simple interface.

5. **Project statistics.**

   Similar to "Information about Ninetech employees", but here we have to use the server solution, since the information needs to be updated every hour. Maybe these two applications could be combined?

6. **Ninetech in Media**

   This application would need to be connected to a server that did all the searching. It might also be possible to perform the searches directly in the application, without the use of a server, but that would make it impossible to make manual changes to the result of the searches, at least changes that should apply to all users.

7. **Synchronization between iPhone calendar and Exchange.**

   This could already be done by using iTunes and connecting the iPhone to the computer with a USB-cable. The iPhone calendar does not have all the features of Outlook,

but that is another issue. It would be good if it was possible to synchronize the calendars over the internet.

6.4 Chapter summary

In this chapter we have described the part of the project that was to interview personnel about how they would like to use an iPhone in their work. From the result of these interviews we have produced some ideas for future applications to be implemented for use on iPhones at Ninetech. We have also arranged these ideas in a prioritized list:

1. **Expenses reporting.**
2. **Booking of conference rooms and other resources.**
3. **Alert system.**
4. **Information about Ninetech employees.**
5. **Project statistics.**
6. **Ninetech in Media**
7. **Synchronization between iPhone calendar and Exchange.**

7 Chapter seven: Evaluation

7.1 Introduction

At the end of the project we sat down with our supervisors at Ninetech and evaluated the result of our project, to see if they acknowledge that we had accomplished our goals with this project.

We also had some of the employees trying the application on an iPod touch and fill out an evaluation form about the perceived usability of the application. This evaluation form can be found in the appendix.

We chose our test persons randomly at Ninetech's office. Some were very familiar with using iPhone / iPod touch. Others had never used anything like that before. And some preferred Android[25]. We managed to achieve a vast diversity of testers, even though we only had a group of eleven persons. Despite the diversity in the group of test persons, they showed a remarkable degree of unity in their answers. This makes us confident that the result of the evaluation form is reliable. This result also matched the opinion of our supervisors at Ninetech, which further improved the credibility of the result.

With such reliable feedback we can be encouraged by everything positive in the answers and we can learn from everything negative in the answers.

7.2 Prototype

We conducted an evaluation among eleven of Ninetech's employees. First the test person got to try to report a timeline with our application on an iPod touch. Then they got to fill out an evaluation form. The evaluation form had four questions. In the first three, the test person had to grade the answers from 1-4. We chose an even number of options to force them to make a decision and not just choose the middle option on all questions.

The form started with a small introduction with instructions of how to fill out the form, as follows:

"This test is designed to measure your experience with the application you've tested today. Your answers will be used to evaluate the application so please answer the questions as truthfully as you can. Please use the scale below to indicate to what extent you disagree or agree to the statements that follow.

1 - Strongly disagree
2 - Disagree
3 - Agree
4 - Strongly agree"

The first question was **I find the application interface easy to use.** The result of this question is shown in Figure 7.1.

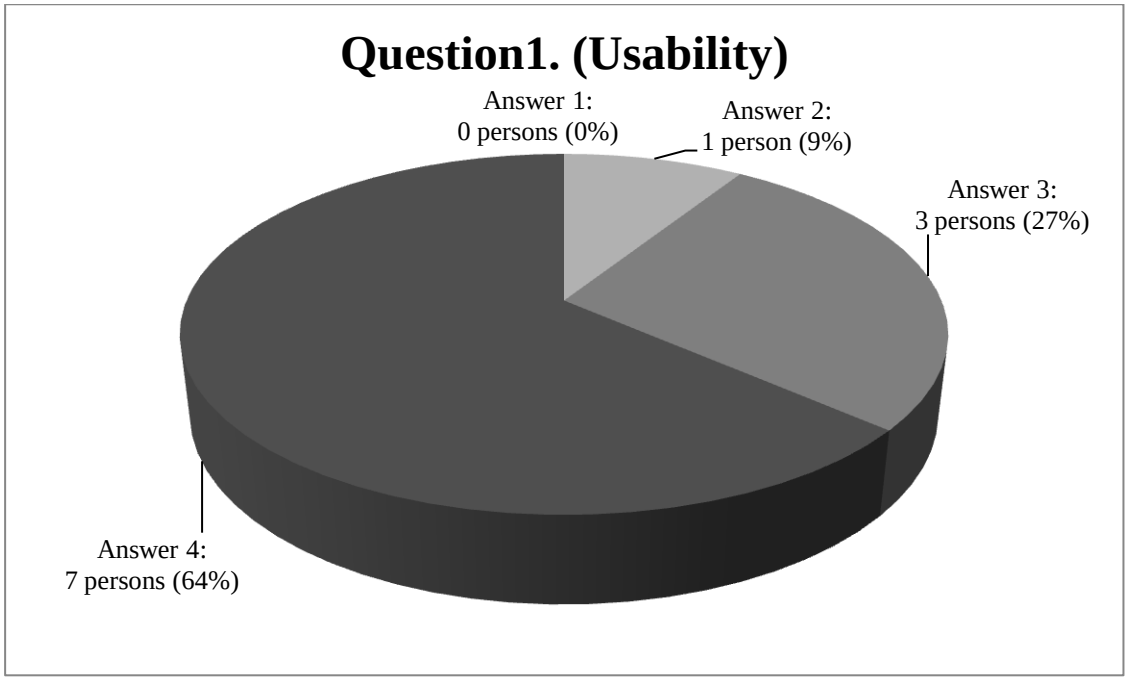

Figure 7.1 Answers to the first question.

The average answer was 3.55 and the median answer was 4, which is a very good feedback on the usability of our application. It is also worth mentioning that the only test person that answered 2, also found the need for an iPhone application very small, since he answered 1 on question number 2.

The second question was **There is a big need for a time reporting iPhone application.** The result of this question is found in Figure 7.2.

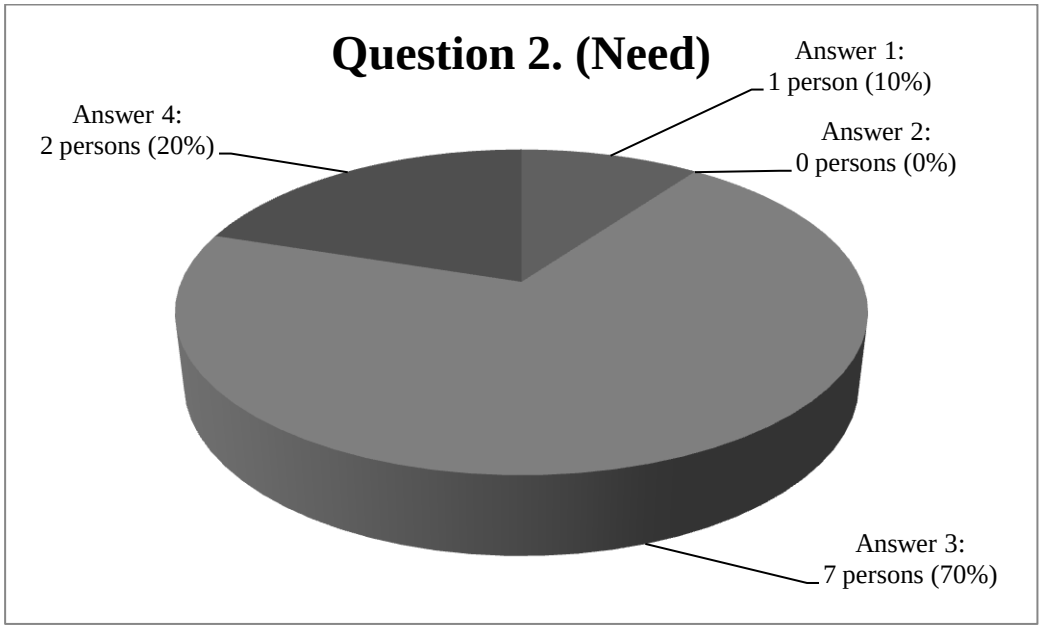

Figure 7.2 Answer to the second question.

The average answer and the median answer here are both 3. Our conclusion is that most of the test persons find a fairly big need for an iPhone application for time reporting. We heard comments about how difficult the current web-based application is and therefore there is a big need for an easier way of reporting times. An iPhone application is one way of solving this, but not necessarily the only way.

The third question was **I would like to use this application regularly (if I had an iPhone).** The result is shown in Figure 7.3.

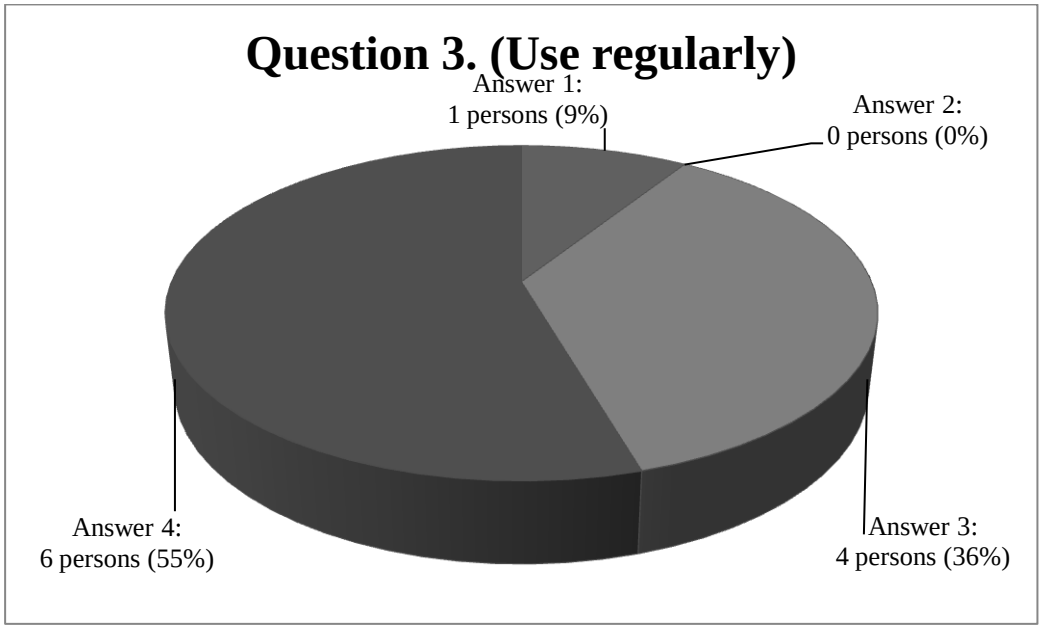

Figure 7.3 Answer to the third question.

The average answer here was 3.37 and the median answer was 4. Also this is a very good feedback for our application. We can also mention here, that the test person that answered 1, also answered 1 at the second question, about the need for an iPhone application. It is natural that someone who does not think there is a need for an iPhone application does not want to use the iPhone application when it is ready. 91% of the test persons answered that they agree or strongly agree that they would use the application regularly (if they had an iPhone).

The last question let them write down a few sentences about what was good about the application and what could be better. Here are some examples of their answers to the last question:

- "Easy to use!"
- "Fast"
- "Easy to report time, anytime!"
- "Easy to use, but I miss an overview of reported time. Possibly, the option to modify a timeline would also be good."
- "Easy to understand the flow. Not complete support for changing between vertical and horizontal position of the device. Not possible to report half-hours."
- "View reported time."
- "It feels like many next-steps. Some kind of overview had been good. Otherwise good!"
- "Some sort of wait-indicator while waiting for server communication. Possibility to send multiple requests."
- " [1] There is a big need for a better time reporting application, iPhone or not. [2]If the app was feature complete and easier to use than the current web app I would (if I had an iPhone).
  Navigation is difficult. The steps are to many so you'll loose yourself and forget where you are. Need overview.
  Would like to have the option to start with the time and add project details later."
- "Simple flow, easy to understand. Many possibilities to further development. Should be easier to find the desired customer."

As you can see, most of the test persons found the GUI easy to use. Most of the things that could be better are things that we had already planned to mention in the Conclusion as future features. For instance many wanted to have an overview of reported time, which is one of the things we would do if we had more time. It also requires some new stored procedure to be implemented for retrieving previously reported timelines.

Some of test persons pointed out that there were too many views. This was a decision we made because it would have been difficult to give all the information in the same view and we had to ask the user for all this information in order to make the application work with the logic of the database.

[1] Question 2.
[2] Question 3.

Evaluation with our supervisors

At the end of the course we sat down with our supervisors at Ninetech to evaluate the project. Just as in the evaluation form, they were generally very pleased with the result of the project. They admitted that they had not given us as much supervision as they had originally planned. They were impressed how far we had reached considering the limited amount of time and resource we had at our disposal. They did point out a number of things that could be complemented in the application. Unfortunately we did not have enough time to fulfill these additional requirements. The most important points that would need to be corrected before the application can be distributed were:

- The user needs to get feedback that a server request has been sent.
- Error handling. For example when a list of projects is empty, a popup message should appear and ask the user to contact the project leader that something is wrong in the database. The user should also always be informed if there is a problem with the server.

Other things that could be improved were:

- Catching information during a session, in order to prevent that identical information is requested from the server multiple times.
- If the user goes back and forth and therefore pushing the "new timeline" button multiple times, the project list is doubled. That means that one of the arrays or dictionaries are not reset in a proper way.
- The datepicker is a poor solution for selecting the day to report time for, since it does not tell the user what day of the week it is. Here in Sweden, most things are planned by the week. In USA, where the iPhone and Xcode has been developed, they plan most things by dates. Therefore there is no good implementation for selecting weekdays. One option would be to implement some sort of calendar view ourselves.
- Support for rotating the device in every view.
- The possibility to report half hours and to increase the maximum number of hours to 12. (The need for half hours is new; our supervisors only wanted full hours at the start of the project.)
- Invoice text should be mandatory.
- Comments should be put in a field with multiple lines.

After the evaluation meeting we made some minor changes to the application to allow rotating of the device in every view and make sure no buttons end up outside the screen when the device is rotated. We also made the invoice text mandatory and increased the maximum number of worked hours to 12.

7.3 Chapter summary

We conducted an evaluation of our iPhone application. First we let eleven random employees test the application and fill in an evaluation form which with four questions. The questions review the usability, the need for the application and if the test person would like to use the application regularly. The fourth question was a free-text question, where the test person was asked what was good about the application and what could be better. In the three first question, the test person was asked give a graded answer on a scale. The scale was from one for "strongly disagree" to four which is "strongly agree". The results of the first three questions we analysed in form of three diagrams one diagram for each question. The diagrams shows the percentage of each grade from one to four. We also calculated the average and median of the results for these three questions. We had another evaluation with our supervisors in Ninetech as well. Our supervisors evaluation was as positive as the other evaluation, except a few things that could be complemented in the application, for which we however had not enough time to implement them. Some of these requirements by our supervisors were later still implemented in the application because we had time to fulfill them.

8 Chapter eight: Conclusions

8.1 General summary

The main part of our project was to design, develop and implement an iPhone application for time reporting. Ninetech already has a web-based time reporting system, but the purpose of making it an iPhone application is to make it more flexible for consultants and employees to report their worked time regardless whether they are; in the office, at a customer's office, or somewhere else. They are going to be able to report their worked time directly to Ninetech from their iPhone.

The project has three major aspects: design, security and data communication between the application and the server. The first aspect we started with was the design. The first GUI design was similar to the web-based system design which had more than one function in a window. The design consisted of three views, one for selecting the customer, project and activity in a picker view, with three buttons to choose the desired list to be displayed in the picker view. The second view looked similar. Here the user selected time-type, price-type and date. In the third view, the user could enter invoice-text and comments. This design was modified to meet our constituents demand and to make the design simpler by dividing the lists in the first two views into multiple views, one view for each list, and a total of nine views. The communication security aspect is important since the information sent over the network is sensitive because it includes the credentials and their worked time for each customer. We chose to implement a security protocol which encrypts the information over the network, namely HTTPS. Data communication between the application and the server is done using XML.

The other part of our project was to investigate what other applications would be useful for the daily work at Ninetech, if they were implemented in the future. We solved this by interviewing seven volunteers from the Ninetech staff and evaluate the result. We then presented the result as a prioritized list of applications to implement in the future.

8.2 Future features

We have discussed with our mentors at Ninetech to find out what additional features the application should have, if we had enough time to develop them. If not, they might be implemented in the future.

New customer or project

One of these features is the ability to report a timeline for a customer or project that the user is currently not registered for. The idea was to write the customer or project name as free text and later have the administration change it to the correct ID number and register the user at that customer or project, if this is correct. We started implementing this feature, but had to stop because Ninetech still has to sort out whether this should be allowed or not and how it should be dealt with.

Reuse timelines

Another feature should be to display the latest reported timelines in a list when the user has logged in. The user should be able to pick one of these timelines to reuse, since users often report many similar timelines and this would then be a lot faster than providing all the information one more time.

Overview

It should also be possible to get an overview of the timelines reported during a week. In this overview it should be possible to both report new timelines and reuse the old timelines for new timelines.

PIN code

Since the user stays logged in even if the application is closed, a PIN code on the application is necessary to maintain security. We have started implementing this feature and we have finished the GUI part of the feature, but we have not had enough time to solve the problem of storing the PIN code in the iPhone's keychain.

8.3 General conclusions-What has been achieved

Mr. Andersson was sitting on the train headed for Stockholm, where he would represent Nine-tech at a big conference. While sitting on the train he decided to get some work done on his laptop. He was so focused on his work, that he was surprised at how fast the trip had gone, when the conductor shouted out: “Next stop Stockholm!” Mr. Andersson just had time to close his computer and get his coat on to leave the train. Out on the platform he realized he had not reported his time and he would not have internet access until he reached the hotel room that night. I better do not forget to report these hours, he thought. Then he remembered that he had just gotten the new iPhone application for time reporting, so he got it out of his pocket and reported his worked hours as he was walking down the street.

An iPhone application allows for time reporting with multiple views. Nine simple views to go through for a daily time report. All the views had the left and right orientation to use the key-board more flexible, because it becomes wider and the size of the buttons will be much big-ger. It does not take long time to fill in the daily time report using the application, because we have to choose one option at a time and just one touch will be enough. No multiple text labels or check boxes to fill in at the same window as it is in the web-based system. The credential procedures are flexible. Once the user has logged in, he will remain logged in. This way the user does not need to login every time. After the user has finished the reporting all he or she needs to do is to log out from the application.

9 REFERENCES

[1] James A. Brannan *iPhone SDK programming: A beginner's guide.*New York: McGrow-Hills, 2009

[2] Tim Bray, Jean Paoli, C.M. Sperberg-McQueen, Eve Maler, Francois Yergeau. *Extensible Markup Language (XML) 1.0 (Fifth Edithion)* W3C Recommendation 26 November 2008 http://www.w3.org/TR/REC-xml ,2010-05-12.

[3] Charlie Miller, Jake Honoroff, Joshua Mason. *Security Evaluation of Apple´s iPhone,* Independent Security Evaluators 2007-07-19. http://securityevaluators.com/files/papers/exploitingiphone.pdf ,2010-05-12.

[4] Eric Rescorla, *RFC 2818: HTTP Over TLS*, The Internet Society May 2000, http://tools.ietf.org/html/rfc2818 ,2010-05-12.

[5] Eric Rescorla, Tim Dierks. *RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2.* August 2008. http://tools.ietf.org/html/rfc5246 ,2010-05-12.

[6] Charlie Kaufman, Radia Perlman, Mike Speciner. *Network Security, PRIVATE Communication in a PUBLIC World Second Edition.* New Jersey, Prentice Hall 2002.

[7] John Franks, Phillip M. Hallam-Baker, Jeffery L. Hostetler, Scott D. Lawrence, Paul J. Leach, Ari Luotonen, Lawrence C. Stewart. *RFC 2617:HTTP Authentication: Basic and Digest Access Authentication.* The Internet Society 1999. http://tools.ietf.org/html/rfc2617 ,2010-05-12.

[8] Wikipedia. *Internet Information Services.* http://en.wikipedia.org/wiki/Internet_Information_Services ,2010-05-12.

[9] Netcraft. *Web Server Survey*. Netcraft, March 2010 http://news.netcraft.com/archives/2010/03/17/march_2010_web_server_survey.html ,2010-05-12.

[10] Tenon Intersystem. *Objective-C, Overview*. Tenon Intersystem. 2010. http://www.tenon.com/products/codebuilder/Objective-C.shtml ,2010-05-12

.

[11] Apple. *The Objective-C Programming Language: Introduction to The Objective-C Programming Language*. Mac OS X Reference Library, 2009-10-19, http://developer.apple.com/Mac/library/documentation/Cocoa/Conceptual/ObjectiveC/Introduction/introObjectiveC.html ,2010-05-12.

[12] Wikipedia. *Application programming interface.* http://en.wikipedia.org/wiki/API ,2010-05-12.

[13] Wikipedia. *Subset*.
http://en.wikipedia.org/wiki/Superset ,2010-05-12.

[14] Apple. *iPhone OS Technology Overview: iPhone OS Technologies*.
iPhone OS Reference Library, 2009-10-19.
http://developer.apple.com/iphone/library/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/iPhoneOSTechnologies/iPhoneOSTechnologies.html , 2010-05-12.

[15] William Stallings. *Cryptography and Network Security, Principles and Practice Fourth Edition.* New Jersey, Prentice Hall 2006.

[16] David Bishop. *A Complete Guide to C#.*
Jones and Bartlett Publishers, Inc, 2004.

[17] Apple Inc. *iPhone in Business, Security Overview*
http://images.apple.com/iphone/business/docs/iPhone_Security_Overview.pdf ,
2010-05-10.

[18] Apple. *Security Architecture.* iPhone OS Reference Library, 2008-10-15
http://developer.apple.com/iPhone/library/documentation/Security/Conceptual/Security_Overview/Architecture/Architecture.html , 2010-05-12.

[19] Wikipedia. *iPhone.*
http://en.wikipedia.org/wiki/IPhone ,2010-05-11.

[20] Andrew Troelsen. *C# and the .NET Platform, Second Edition.* New York, Springer-Verlag, 2003.

[21] Michael Yuossef. *Visual C#.NET, Introduction to Programming Languages*
http://www.devarticles.com/c/a/C-Sharp/Visual-C-Sharp.NET-Part-1-Introduction-to-Programming-Languages/ ,2010-05-28.

[22] Wikipedia. *XML.*
http://en.wikipedia.org/wiki/XML ,2010-06-15.

[23] Apple *Mac OS X – Developers*
http://www.apple.com/macosx/developers/#xcode ,2010-06-14.

[24] Mary Feeney. *The Standard Generalized Markup Language (SGML).* British Library Research and Development Dept. and Library & Information Technology Centre, London, UK, 1988.

[25] Android *Android.com*
http://www.android.com/ ,2010-06-15.

10 Appendix 1: Evaluation form

iPhone Application Evaluation

Instructions

This test is designed to measure your experience with the application you've tested today.
Your answers will be used to evaluate the application so please answer the questions as truthfully as you can. Please use the scale below to indicate to what extent you disagree or agree to the statements that follow.

1 - Strongly disagree
2 - Disagree
3 - Agree
4 - Strongly agree

I find the application interface easy to use.

☐1 ☐2 ☐3 ☐4

There is a big need for a time reporting iPhone application.

☐1 ☐2 ☐3 ☐4

I would like to use this application regularly (if I had an iPhone).

☐1 ☐2 ☐3 ☐4

What is good about the application? What could be better?