VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

WAN/LAN MANAGER FOR IOS

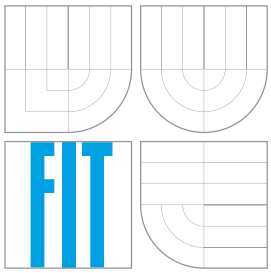
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ JAKUBIS

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

WAN/LAN MANAŽÉR PRO IOS

WAN/LAN MANAGER FOR IOS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ JAKUBIS

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. LUKÁŠ MARŠÍK

BRNO 2013

Abstrakt

Cílem tohoto projektu je vytvořit aplikaci pro systémy iOS pro správu síťových zařízení, klienty a konfigurace zařízení. To zahrnuje projektovou analýzu, návrh aplikace, její implementaci a testování. Aplikace se skládá z klientské a serverové strany, databáze a komunikačního protokolu. Výslední aplikace je pak porovnána se stávajícími řešeními s podobnými výsledky. Výsledky testů ukázaly určité nedostatky a nevýhody řešení. Možná zlepšení a budoucí vývoj byly popsány. Na závěr, po dalším vývoji má aplikace potenciál stát se používanou v praxi.

Abstract

The goal of this project is to create an application for iOS to manage network devices, clients and device configurations. This includes project analysis, application design, implementation and testing. Application consists of client's side, server's side, database and communication protocol. The resulted application is then compared with existing solutions with similar results. Test results have revealed several weaknesses and drawbacks of the solution. Possible improvements and future development have been described. As conclusion, the application may have potential to be used in practice after more development.

Klíčová slova

síťová konfigurace, uživatelská rozhraní, mobilní zařízení, iOS, PHP, MySQL, Objective-c, Xcode

Keywords

network configuration, user interface, mobile devices, iOS, PHP, MySQL, Objective-C, Xcode

Citace

Tomáš Jakubis: WAN/LAN Manager for iOS, bakalářská práce, Brno, FIT VUT v Brně, 2013

WAN/LAN Manager for iOS

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Lukáše Maršíka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Jakubis
May 14, 2013

Poděkování

Chcel by som sa poďakovať vedúcemu práce Ing. Lukášovi Maršíkovi za vedenie práce a Michalovi Wernerovi za rady a poskytnutie technického vybavenia.

Contents

Chapter 1

Introduction

Information technologies in super yachts and travelling industry in general face many challenges. Since super yachts are very expensive privately owned yacht, owners require the best service and the best solutions for these challenges. One of them is providing stable internet connectivity during the whole trip. In consideration of these network devices are constantly moving and may pass through different coverages of different providers, it is not an easy task.

This document describes the analysis and the design for LAN/WAN Manager for iOS project. This project started on impulse of the company Bond Technical Management. BondTM is technical management company for on-board Audio Visual, IT, Communications, Security and Navigation systems for the most advanced super yacht builds and projects (example of super yacht is on picture 1.1).

Figure 1.1: Example of super yacht.

This document starts with analysis in chapter 2. Here is this challenge described in more detail, existing solutions and available tools are presented. After that in chapter 3 behavioural design will be introduced, entities and their relationships will be described, graphical design of application and test preparation are presented. Implementation of this design is described in next chapter 4. Test definitions and results are located in chapter 5, future plans in chapter 6 and finally, the project is concluded in chapter 7.

Chapter 2

Analysis

Formulation of Sections Problem description and Requirements is based on several consultations with the Bond TM developer.

2.1 Problem description

The goal of this project is to create an application for iOS for managing network devices in a way, that is easily managed by non certified personnel. This is useful in the traveling industry. Every internet service provider provides only some coverage. For example, MTN Satellite Communications (MTN) is a company to offer a stabilized Very Small Aperture Terminal (VSAT) satellite solution for ships at sea. Figure 2.1 found in MTN's websites [8] shows C-band global coverage of the MTN company. MTN company there also claims, that C-band is the most-effective solution. Compared to this their ku-band coverage in figure 2.2 is smaller, but offers more redundant and stable conectivity. As shown in the figures, as the ships travel around the world they cross areas of coverage of several providers and every time they need to reconfigure their devices accordingly. To accomplish this, every ship would must have networking certified expert onboard. Because of the cost of this solution, an easy-to-use application for managing networks is very desirable.

2.1.1 Requirements

The application needs to store data and keep them consistent, we need a server onboard for storing them. Because of that, the application must have a client-server model. In case of traveling industry, especially yachts and super yachts, servers are already built to fulfill the certain needs, that may vary and often implement several various protocols. For the purpose of this project the server part of the application can be simplyfied.

The server does not necessarily need all the functionality of the commercial servers. It does not need to be connected to any networking devices as this is not goal of this project. Although, it needs database to store the data and must simulate the rest required functionality like saving current running networking configuration. Both client and server must implement the same communication protocol, although it can be simplyfied as different servers may implement different protocols. Because of that, the development of this application will not be focusing on communication protocol.

On the other hand, the client's part of the application requires user-friendly presentation of received data from the server. User must be able to select profiles with configuration of

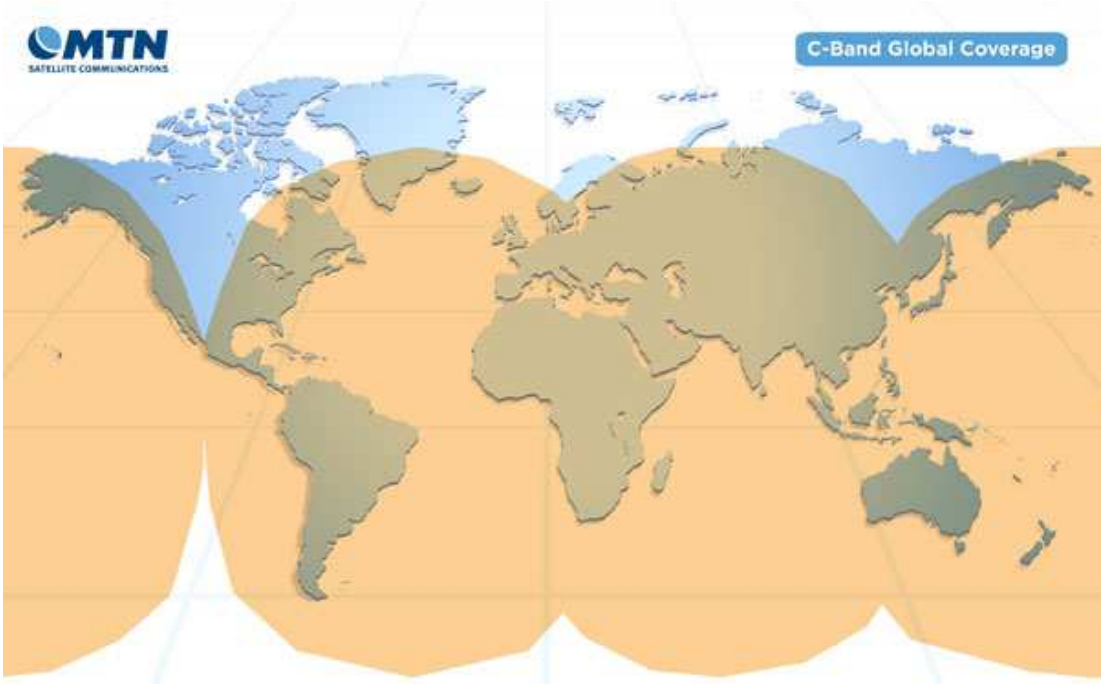


Figure 2.1: C-band global coverage

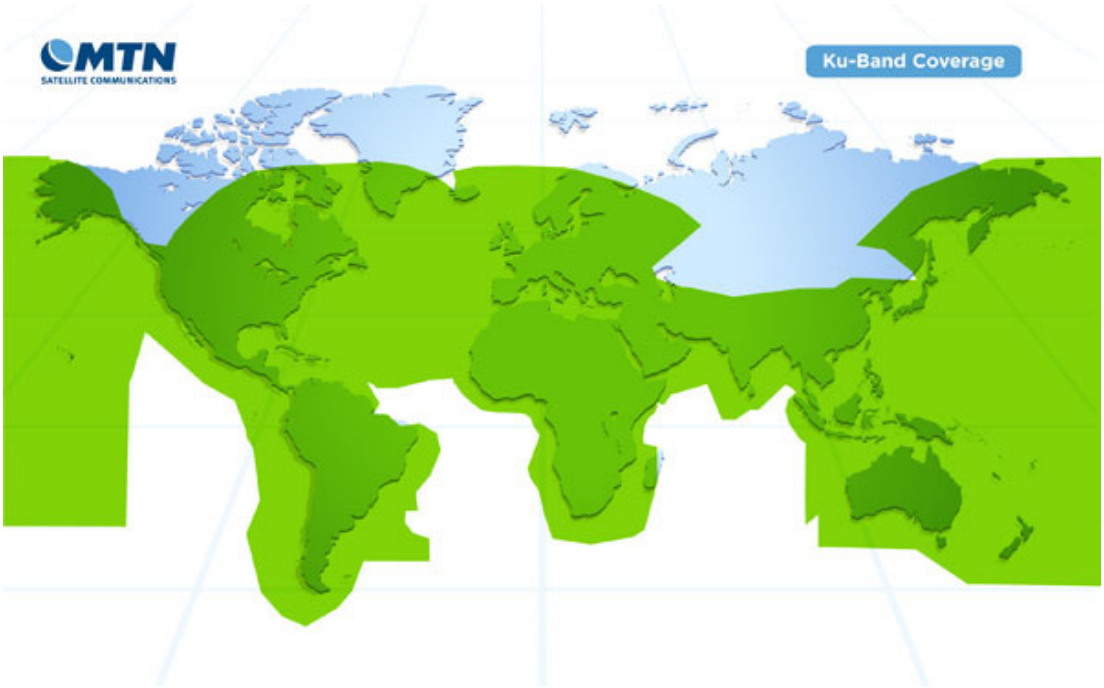


Figure 2.2: Ku-band global coverage

networking devices easily, change user priviledges and observe statistics in understandable way.

Interface requirements are:

- authentication for different user types,

- creating, storing, editing, deleting and selecting profiles with WAN configurations,
- presenting data gathered by various management protocols (Netflow, SNMP) in understandable way,
- managing authorized and unauthorized clients and devices in the network,
- network time managing,
- changing settings and
- keeping log of events.

According to the above requirements, the development of the project will be mainly focusing on the clients part of the application. Another requirement, that is specific for this project is compatibility with iOS devices.

2.2 Existing solutions

The Bond TM company owns project The Jetstream with product Jetselect. This product meets all requirements stated above. Application is divided in several parts: WAN Manager, LAN Manager, NTP Manager, Settings and Statistics. However the Jetselect is not compatible with the iOS, what is the goal for this project. Figure 2.3 shows profile selection in Jetselect.

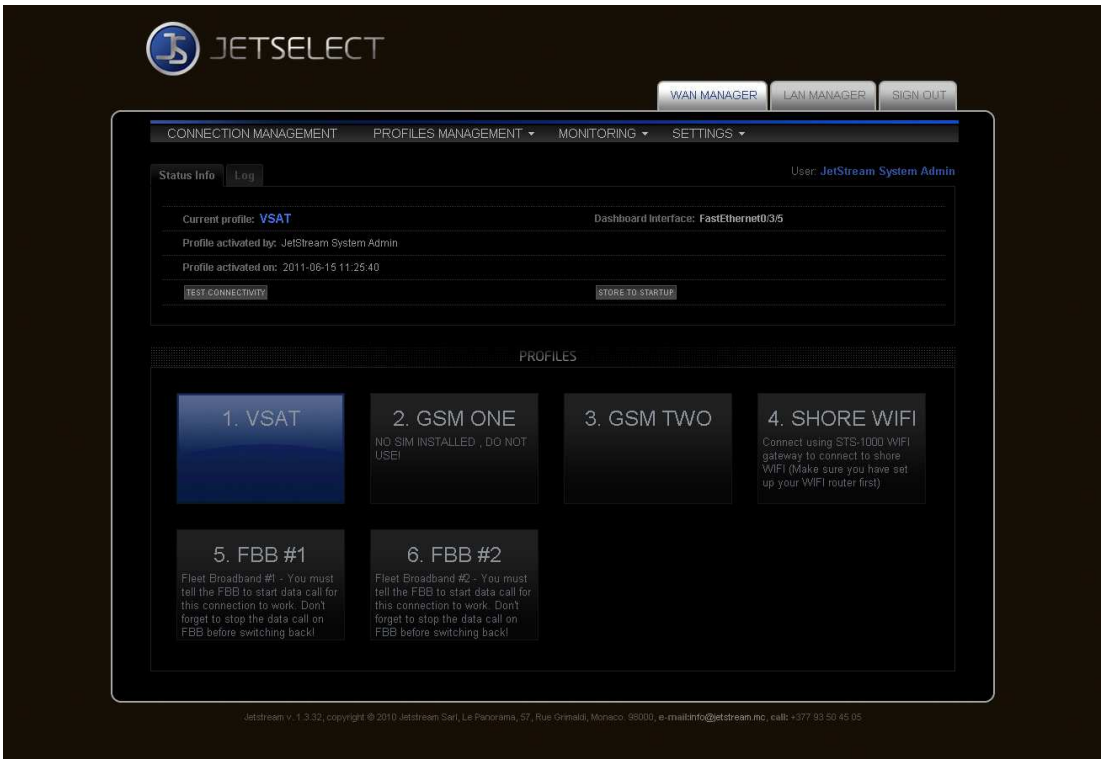


Figure 2.3: WAN profile selection in Jetselect

Other managers do not meet the key requirements. They are not easy to use, they require certified personnel or they are for visualisation and statistical purposes only and

can not change configurations. Why they are not suitable is described in the previous chapters.

2.3 iOS

iOS Technology Overview[2] describes iOS as following:

iOS is the operating system that runs on iPhone, iPod touch, and iPad devices. The operating system manages the device hardware and provides the technologies required to implement native apps. The operating system also ships with various system apps, such as Phone, Mail, and Safari, that provide standard system services to the user.

The iOS Software Development Kit (SDK) contains the tools and interfaces needed to develop, install, run, and test native apps that appear on an iOS device's Home screen. Native apps are built using the iOS system frameworks and Objective-C language and run directly on iOS.

As with the computers, the operating system acts at the intermediary level between the hardware and software. Apps communicate with the hardware through the system interfaces and the app is protected from any hardware change. Thanks to this, one app can run on both iPad and iPhone and all other compatible iOS devices that may come in the future.

2.3.1 Xcode

In the same article is Xcode described as following:

Xcode is the development environment you use to create, test, debug, and tune your apps. Xcode comprises the Xcode app, which is a wrapper for all of the other tools you need to build your apps, including Instruments and iOS Simulator. You use Xcode to write your code and then run your apps in iOS Simulator or directly on an attached iOS device. After debugging your app, you can use Xcode to launch Instruments and profile your app.s performance.

Development on an actual device requires signing up for Apple's paid iOS Developer Program and configuring a device for development purposes.

When creating new application, Xcode starts a new project. User can select from several different built-in templates, that start a project with prepared code for a basic ready to deploy application. These templates ensure, that all applications have the standard layout of graphical user interface. After the project creation, user is able to simulate his application on iPhone or iPad simulator or he can deploy the application to a connected device.

2.3.2 Objective-C

Objective-C is described in Write Objective-C Code article[5] as following:

The Objective-C language specifies a syntax for defining classes and methods, for calling methods of objects, and for dynamically extending classes and creating programming interfaces adapted to address specific problems. As a superset of the C programming language, Objective-C supports the same basic syntax as C. You get all of the familiar elements, such as primitive types (`int`*,* `float`*, and so on), structures, functions, pointers, and control-flow constructs such as* `if...else` *and* `for` *statements. You also have access to the standard C library routines, such as those declared in* `stdlib.h` *and* `stdio.h`*.*

Objective-C adds the following syntax and features to ANSI C:

- *Definition of new classes*
- *Class and instance methods*
- *Method invocation (called messaging)*
- *Declaration of properties (and automatic synthesizing of accessor methods from them)*
- *Static and dynamic typing*
- *Blocks-encapsulated segments of code that can be executed at any time*
- *Extensions to the base language such as protocols and categories*

Apart of that, every application uses Model-View-Controller design pattern, classes can be defined as delegates so objects can communicate with each other and they can delegate specific tasks, new version of iOS brings Automatic Reference Counting, a graphical user interface can be constructed through Storyboards and transitions between different views are done by using segues. This sets out a very complex programming language. Since Objective-c is not that common language, several key features that are not commonly used in other languages will be presented.

Delegates

Cocoa Core Competencies [3] describes delegating design pattern as following:

Delegation is a simple and powerful pattern in which one object in a program acts on behalf of, or in coordination with, another object. The delegating object keeps a reference to the other object - the delegate - and at the appropriate time sends a message to it. The message informs the delegate of an event that the delegating object is about to handle or has just handled. The delegate may respond to the message by updating the appearance or state of itself or other objects in the application, and in some cases it can return a value that affects how an impending event is handled. The main value of delegation is that it allows you to easily customize the behavior of several objects in one central object.

It is almost impossible to build a user interface for iOS in Objective-c without using delegates. For example, when tapping a text field, keyboard will be shown to enable user to fill that text field. However, the keyboard will not hide after the task is done. The keyboard does not know, to which object it should return the first responder status. By assigning a delegate to text field, keyboard is functioning as intended.

Storyboards

Cocoa Application Competencies for iOS [1] describes storyboards as following:

A storyboard is a visual representation of the user interface of an iOS application, showing screens of content and the connections between those screens. A storyboard is composed of a sequence of scenes, each of which represents a view controller and its views; scenes are connected by segue objects, which represent a transition between two view controllers.

Xcode provides a visual editor for storyboards, where you can lay out and design the user interface of your application by adding views such as buttons, table views, and text views onto scenes. In addition, a storyboard enables you to connect a view to its controller object, and to manage the transfer of data between view controllers. Using storyboards is

the recommended way to design the user interface of your application because they enable you to visualize the appearance and flow of your user interface on one canvas.

Storyboards allow to user make the graphical user interface completely without writing any code. If needed, all views can be linked to code and then any changes to interface can be done through the linked references. Similar functions has QT api, however, in Xcode there are no needs for signals and slots.

Model-View-Controller

In Xcode, new applications and projects use the Model-View-Controller design pattern by default. Article Streamline Your App with Design Patters[4] describes this design pattern as following:

The Model-View-Controller design pattern (commonly known as MVC) assigns objects in an app one of three roles: model, view, or controller. The pattern defines not only the roles objects play in the app, it defines the way objects communicate with each other. Each of the three types of objects is separated from the others by abstract boundaries and communicates with objects of the other types across those boundaries. The collection of objects of a certain MVC type in an app is sometimes referred to as a layer - for example, a model layer.

The relationship among the layers is shown in figure 2.4

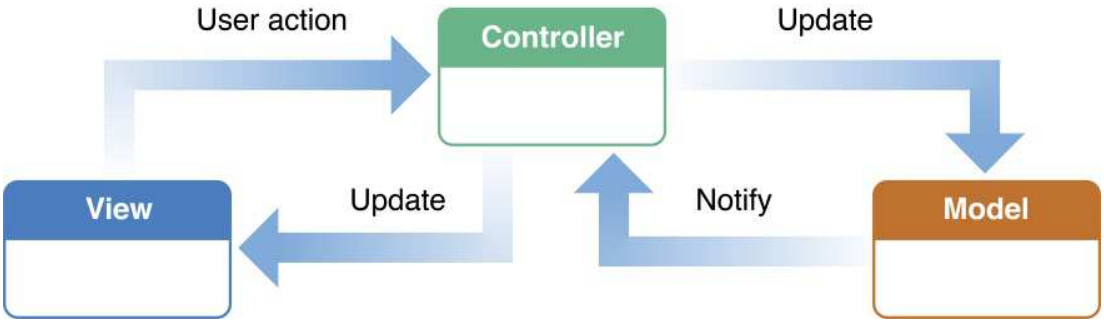


Figure 2.4: The relationships and actions among the layers. The model never communicates with the view directly.

Chapter 3

Design

Previous chapter analyzed requirements and available tools. In this chapter application design is presented.

3.1 Behaviour

The behaviour of application is defined by roles and actions. Desired roles or actors will be:

- Administrator,
- Technician,
- Captain,
- Time.

Administrator has access to all information and to all parts of the application. He has the authority to make any changes to settings of the application, manage users, manage network profiles, manage clients and devices and manage NTP. Technician is subset to administrator. He has the authority to make all the decisions as the administrator, but does not have any access to application settings or user management. Technician nor administrator does require to be physically present on the ship. Captain is crew member of the ship and has access to profiles and clients, but only in read-only mode.

Time is a special role of this design, that does not involve human interaction. Server manages user sessions, monitors network and tracks history of performed actions.

Figure 3.1 shows all stated roles and actions in one use-case diagram.

3.2 Model-View-Controller

In case of this project, the model of application will be a communication protocol with server and received data. LAN Manager, WAN Manager, Statistics and Settings panels will all be views. Every view requires a controller to display requested data, switch to different views, handle user's input and send request to database controller. This database controller will manage database model and will implement communication protocol.

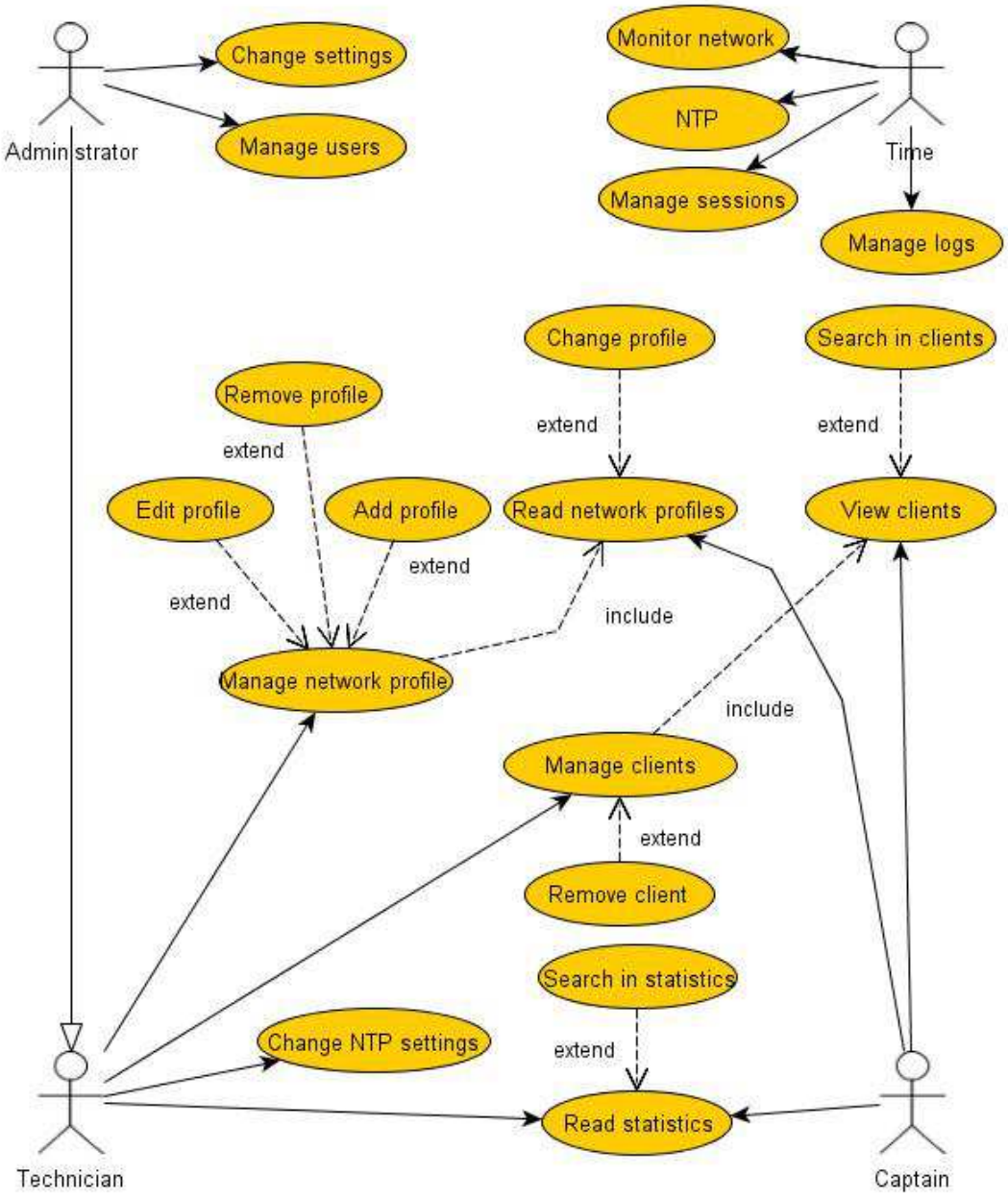


Figure 3.1: The behaviour of application in roles and their available actions.

3.3 Database server

Relationships between entities are shown in diagram 3.2.

3.3.1 Tracking history

Almost every entity has a compound key that consists of the `id` and `time` attributes. `Activity` attribute set to zero represents that an object has been deleted. Example of this is shown on the `Host` entity in following table:

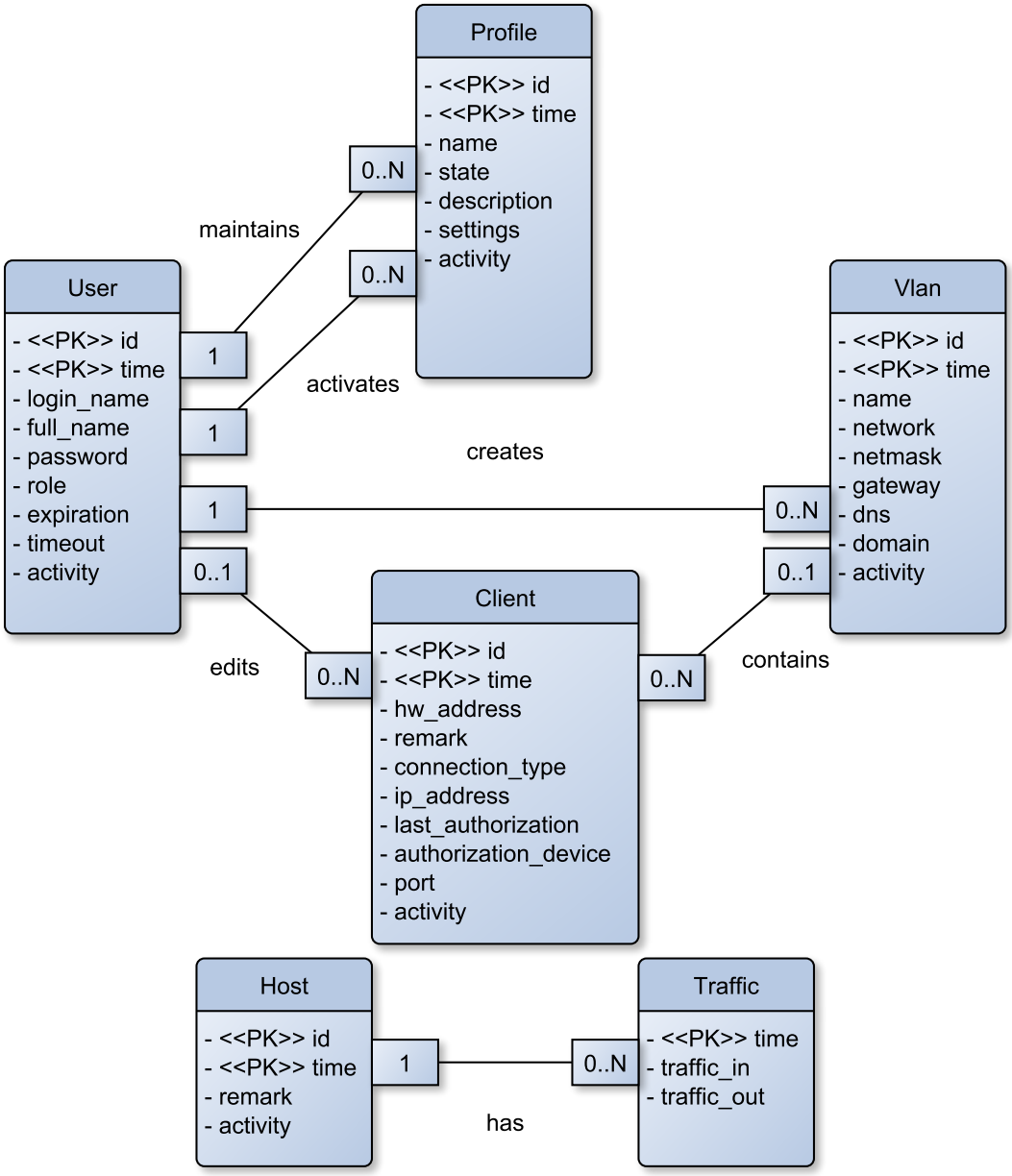


Figure 3.2: Entity relationship diagram.

id	time	remark	activity
1	2013-01-01	hostCreated	1
1	2013-01-02	hostChanged	1
1	2013-01-03	hostChanged	0

This table shows only one host with constant `id`. On the first day, this host had been created with `activity` attribute set to non-zero. On the second day a new row was added to the database with the same `id`, but with a new `time`, which means that this was a new version of the host. On the last day, this host has been deleted, what is represented by `activity` attribute set to zero. This allows to track a history and all changes made to every object, that is represented by any entity.

3.3.2 Roles of entities

Entities `Client`, `Host` and `Traffic` contain data generated by network. Clients represent devices that are trying to access the network, Hosts are network devices or ports that are being monitored and Traffic represtents an amount of data went in and out through selected host.

On the other hand, entities `Profile`, `User` and `Vlan` contain data generated by user input in the application. User is a person, that can log in to the server through the application, Profile represents a WAN profile to be configured on the network devices and Vlan contains information about dfferent vlans.

3.4 Communication protocol

Following data transmissions are required:

- authorization and authentication of users,
- session data,
- database actions such as selecting, adding, editing and deleting

Because of that application is going to connect to different servers, different communication protocols may be used. This can be achieved by dynamic typing with database abstraction layer. [1] To simplify this protocol, HTTP GET request may be used. In that case, the server needs to handle these requests and some kind of scripting language is required. In that matter PHP with MySQL may be used. To transmit requested data from server, it is acceptable to use JSON.

3.5 Graphical design

Since the goal of this application is to make work easier to some people, the main focus of user interface is on the speed of manipulation. Firstly, user should fast and easily navigate between views to perform desired action. This can be done by consistency with other apps. Consistency is described in iOS Human Readable GuideLines [6] as following:

Consistency in the interface allows people to transfer their knowledge and skills from one app to another. A consistent app is not a slavish copy of other apps. Rather, it is an app that takes advantage of the standards and paradigms people are comfortable with.

Consistency is achievable by using standard iOS controls, for example, text fields, buttons or even a keyboard.

Secondly, a large amount of data will be shown to the user. Views must be in logical hierarchy so the user can navigate and find the required piece of information, that he is looking for easily. With consistency in mind, this is achievable with iOS providing a navigation bar and a tab bar. With navigation bar, user can go through content from general to more particular and then get back. On the other hand, with tab bar, content can be divided in several groups and then, user can switch between them. With both bars, a user interface with a powerful navigation hierarchy can be created.

Lastly, different user types have different rights. Controllers have to ensure, that every user can access only specific views and perform only a specific actions. This is achievable by disabling or hiding specific controls of user interface.

[1] The database abstraction layer is described in [7].

3.6 Testing plan

Tests will be based on what was stated in previous sections, mainly in section Graphical design 3.5. Testing subjects will be divided in two categories. People, that work with iOS regularly and people who don't. Tasks will be given to both groups and time will be measured. After finisng all tasks, subjects will wait some amount of time and then the similar tasks will be given to both groups again.

The time difference will reveal the quality of design and user interface. The group, that works with iOS regularly should have rather small difference, since user interface is consistent with other apps. On the other hand, the aim of the other group is to have a bigger difference, which would mean, that navigation in application is easy to learn and even for new users after several tries the application is easy to use.

Chapter 4

Implementation

The implementation of application have been based on previous analysis and design from previous chapters. Almost all validation of user data is done in client side of application. Valid data is then sent to database server that handles the request and responds accordingly. Client waits for response and displays the received data to the user. Client part of application requires iOS version 6.0 or higher.

4.1 Client

Based on what was said earlier, the client's part of the application have been implemented in Objective-c. Database entities are mapped to classes in application model. Views and view controllers in storyboard are mapped to controller classes in code.

4.1.1 User interface

User interface have been created through the Storyboard. User interaction starts with login screen with the text fields and a button. In this state, no navigation through the application is available. When user taps on any text field or text area in this application, the keyboard will be shown.

WAN manager

After that user is successfully logged in the WAN manager tab is active as shown in the figure 4.1. On the bottom is tab bar with tabs for a LAN manager, where clients are managed, a WAN manager, where profiles are managed, a settings, where administrator can manage users and vlans and a logout button. The WAN manager contains the basic information about current active connection and an additional menu where user can go to a connection management, a profile management and monitoring.

Connection management as shown in figure 4.2 brings collection view controller with buttons to change current active connection. Every button represents one configured profile. Profile name and description should guide user to select correct profile. Tapping on a profile button activates the selected profile.

In profile management user can browse and reconfigure profiles as shown in picture 4.3. This section contains a navigation bar at the top of the screen and table with available profiles and drafts. Tapping on profile or draft brings detail view with more detail information about that profile. In the views, that contain text areas or text fields at the bottom

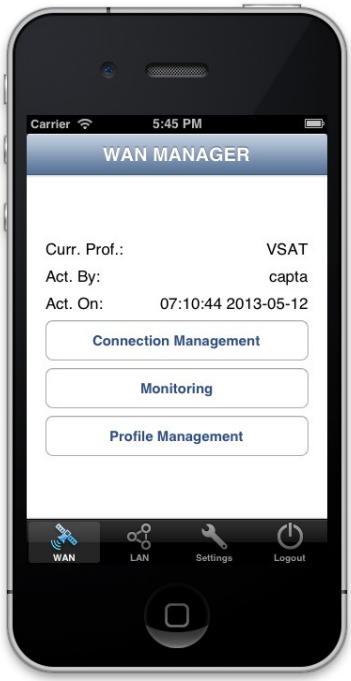


Figure 4.1: WAN manager tab

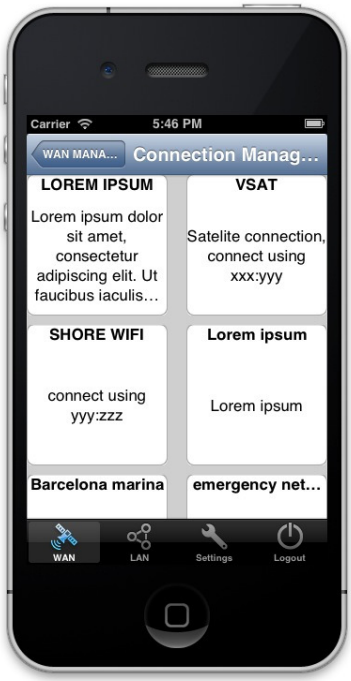


Figure 4.2: Profile selection

of the view, there is a scroll view as well. Then, when user taps the bottom text area, the view scrolls so keyboard will not hide the text area behind. Thanks to this, the text area is visible all the time user is writing to it. Apart of text areas for description and settings,

there is present a state, with which user can change profile to draft and back. Drafts are not visible in connection management when changing the active connection. When creating a new profile, user can start with one of the drafts and then save it as profile. Profile management is only available to administrator and technician. Captain is not authorized to make changes to profile settings.

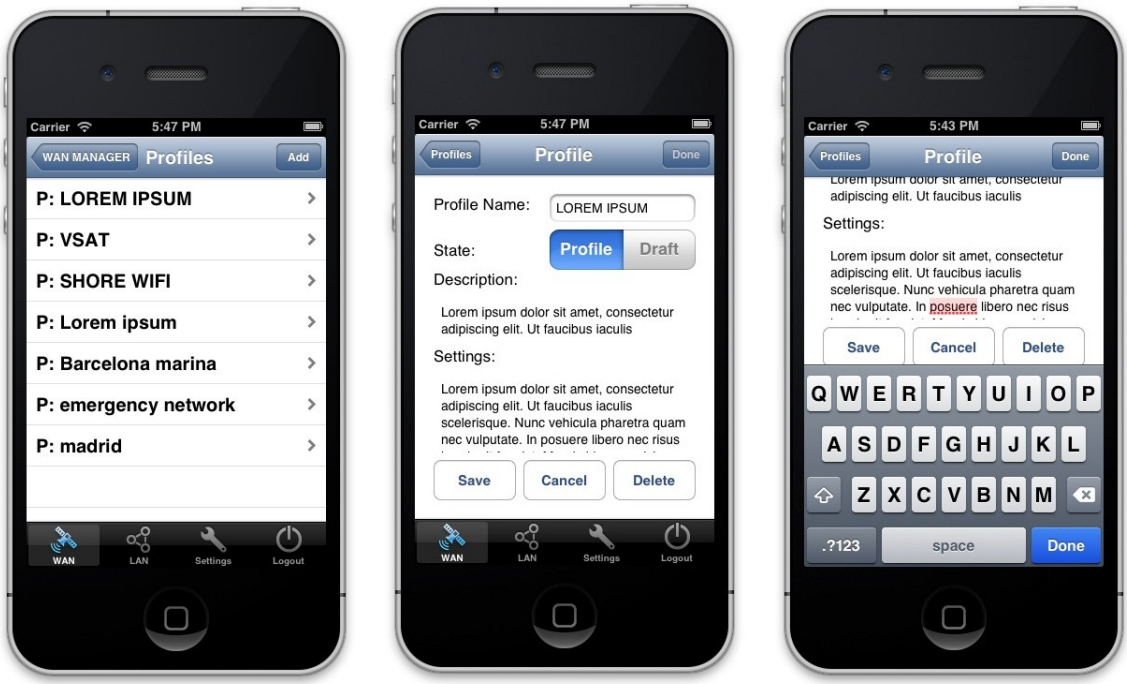


Figure 4.3: Example of profile selection with table, profile management and scrolling the view with keyboard.

In monitoring shown in picture 4.4, user can browse between the hosts with navigation bar and hosts table. Detail view shows a chart with traffic of selected host for last 8 hours. Chart is made with CorePlot framework. With arrows user can observe more previous data. This could be replaced with tap gestures, but the framework does not bring the desired behaviour sufficiently enough. Because of that, the chart is locked and is controlled only via arrow buttons. Thanks to that, controlling the displayed data is simpler and more accurate. The client asks data from the database only when button is pressed. Displayed maximum and average data is based only on data shown on current time interval.

LAN manager

The LAN manager contains a navigation bar with table of authorized and unauthorized clients as shown in picture 4.5. The difference between them is that unauthorized clients are not assigned to any vlan. A switch between authorized and unauthorized clients can be done by tapping a button in the navigation bar. Tapping on a client in the table brings a view with details about this client. Administrator and technician can both change these details. To assign or change a vlan, user can tap a vlan button. This will bring the table of vlans that user can choose from. Even though clients represent devices in the network and should be added to the database by connecting to the network, creation and deleting is available for the users. This is because administrator may configure a device even before this

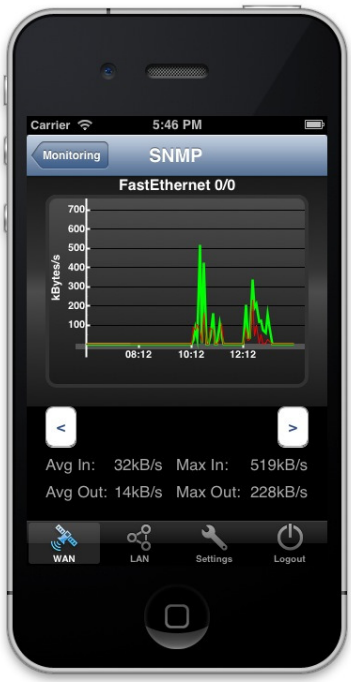


Figure 4.4: Example of SNMP chart with local maximum extreme and average.

device is plugged to the network for the very first time or technician may want to delete old devices that are no longer in the network. Captain can browse clients, but control buttons are hidden from him and even though data fields are still visible, they are disabled.

Figure 4.5: Example of authorized and unauthorized clients and clients data manipulation.

Settings

Settings are only available to administrator. Here is available user management and vlan management. In user management expiration time button brings a rollable calendar, with which administrator can pick a date for expiration as shown in picture 4.6. Apart of that, both works similarly like client or profile management. Picture 4.7 shows data manipulation with vlan management.

Figure 4.6: User data manipulation and picking expiration date.

4.1.2 Communication protocol

Since client may in the future communicate with different servers, the communication protocol creation has been mainly focused to be easy to replace. This was not an easy task, because every view controller needs to communicate with database. To make all view controllers consistent with communication protocol, delegate protocol have been created. Every view controller, that needs to communicate with the database must conform to this protocol. After that, view controller must have implemented methods specified by this protocol. In these methods, controllers receive and handle data from server and mostly perform some kind of action based on received data.

There is a class `DatabaseController` which is called when a controller needs to send a request to database. Every time, view controller is in a role of delegate in this action. Database controller then constructs a GET request and sends it to the server. Database controller then handles receiving of response and then, the response is sent back to the delegate.

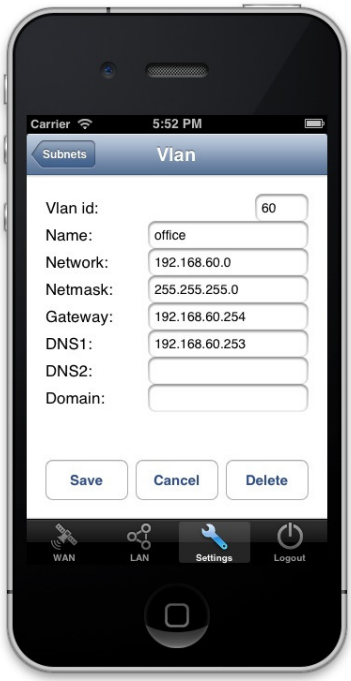


Figure 4.7: Vlan data manipulation.

4.2 Server

Server has been implemented in PHP. It is a simple script that creates a session for user and handles GET requests. All requests follows the same pattern of required `action` parameter and optionally more following parameters. For example, request for getting all unauthorized clients will look like this:

?action=getClientList&authorized=NO

Based on this action parameter is then selected the corresponding function. In this function is then constructed MySQL request and then it is executed. Every response follows the same pattern of required `success` boolean object and objects `data` or `errorMessage`. If action is successful success object is set to true and data array is constructed with requested data or success set to false and errorMessage is returned otherwise. These two objects (success and data or success and errorMessage) are then encoded to JSON and sent back to client. From previous example, result would be:

{ „success“:true, „data“:[{ „clientId“: „24“, „clientRemark“: „Room 41 - unknown device“},{ „clientId“: „25“, „clientRemark“: „Room 41 - broken switch“}]}

4.2.1 Database

Database creation have been based on entity relationship diagram from previous chapter. Every table tracks back history of every object. To be able to track the history, no rows are ever deleted or updated, only new rows with new timestamps are inserted. To make browsing and selecting from these complex tables easier, views have been created. These views contain only active and up to date objects.

Chapter 5

Testing

In the design, the testing phase was planed with for two groups in two rounds. The group, that is used to iOS applications could not be found in reasonable time and because of that tests were applied only on the other group. However, testing have been extended to comparison with the Jetselect. Testing subjects were all students in 21-22 age with at least minimal knowledge of the network fundamentals. Measured data consists of time of completing given task, number of taps and lost count. Lost count means have many times subject went other direction in navigation hierarchy that he was supposed to.

Testing have been done on iPhone 6.0 Simulator in Xcode, server's side of application have had MySQL version 5.5 and PHP version 5.3.24.

5.1 Prepared tests

In the first round following tasks were assigned for both Jetselect and iOS application:

Task number	Task description
1	Connect as captain and activate VSAT wan profile
2	Connect as technician and create wired client Room 210 - printer 1 with vlan office, wireless Room 210 - printer 2 with wifi vlan and Room 211 - printer
3	Connect as administrator and create vlan Printers
4	Change password of captain to „captain“
5	Connect as technician and move all printers to vlan Printers
6	Find out, which host had in the last 16 hours the most average traffic in
7	Create an empty draft for profile
8	Create a profile from an empty draft that has the name Barcelona marina and description that warns a user to use 7824623 as password
9	Find out, which vlan has most clients

Second round have been measured after several hours on iOS application with following tasks:

Task number	Task description
1	Connect as captain and activate Shore wifi wan profile
2	Connect as technician and change Room all printers to IP phones and assign them to office vlan
3	Connect as administrator and delete vlan Printers
4	Move expiration date of technician to 2014-07-01
5	Connect as technician and delete all unauthorized clients
6	Find out, which host had in the last 16 hours the less average traffic out
7	Delete all drafts
8	Create a profile that has the name Emergency network and description that warns a user to use the profile only in emergencies
9	Find out how many wireless clients are in use

5.2 Test results

5.2.1 Time

Required time per task is shown in figure 5.1. From this figure, it is clearly visible that first round of iOS application was far more time-consuming then a round in Jetselect. This may be because of Jetselect is more straight-forward thanks to wider screen of computer and HTML. Thanks to that Jetselect has easier navigation hierarchy. However, the second round was in most cases much faster than the first round and is comparable to Jetselect round. This shows that after several tries, application gets easier and work with it is much faster. If subjects had more rounds, speed of work would probably went even higher and maximum speeds of work with Jetselect and iOS application could finnish with the same values.

5.2.2 Tap count

Taps for each round with iOS application have been counted too, but the resulting values are not that clear like with time. That is because user input have been counted as well and since rounds were not identical, gathered data does not provides much of information value. However, an overall slight improvement in second round can be seen and chart with gathered data is shown in figure 5.2.

5.2.3 Lost count

Lost count in figure 5.3 has with time the most information value. Both charts have brought the same results. Here as well is major improvement in the second round compared to first round and the results with Jetselect and second round are similar. However, second round have not been perfect and several lost counts have been still present. That means that navigation hierarchy is not that intuitive and on the second use, subjects occasionally had lost in it.

From previous chart it is easy to find which sections of applications were the least intuitive and most confusing. This can be done by categorising performed tasks and adding average lost counts into these categories. The result of this action is shown in the figure 5.4. Subjects found confusing to find client management in LAN tab. This may be caused by

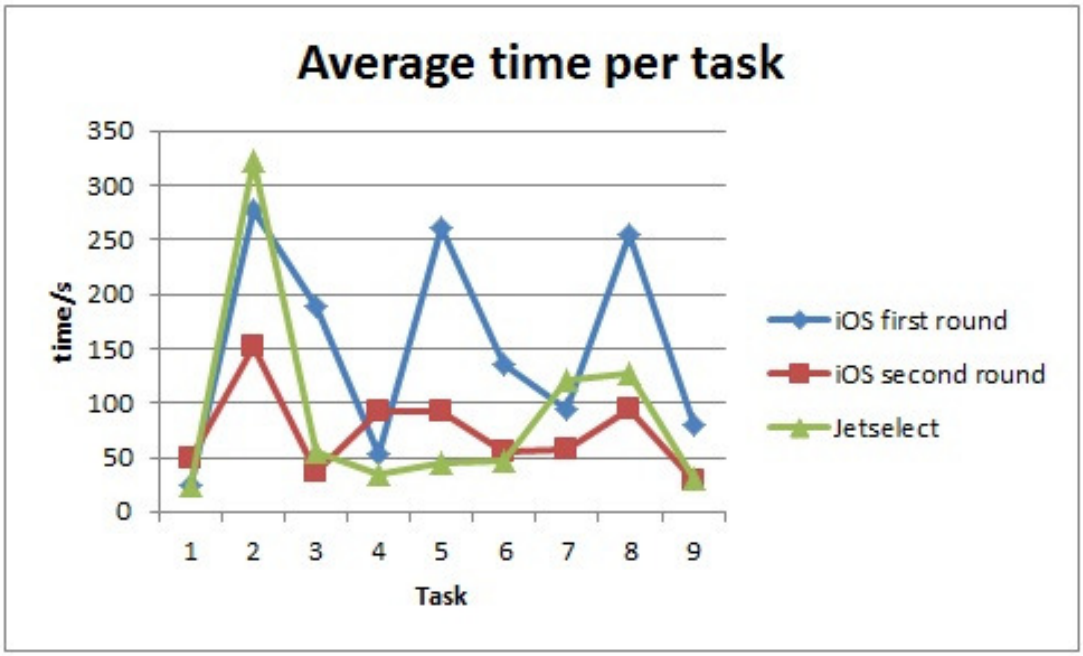


Figure 5.1: Chart that represents how much time was required to finnish given task.

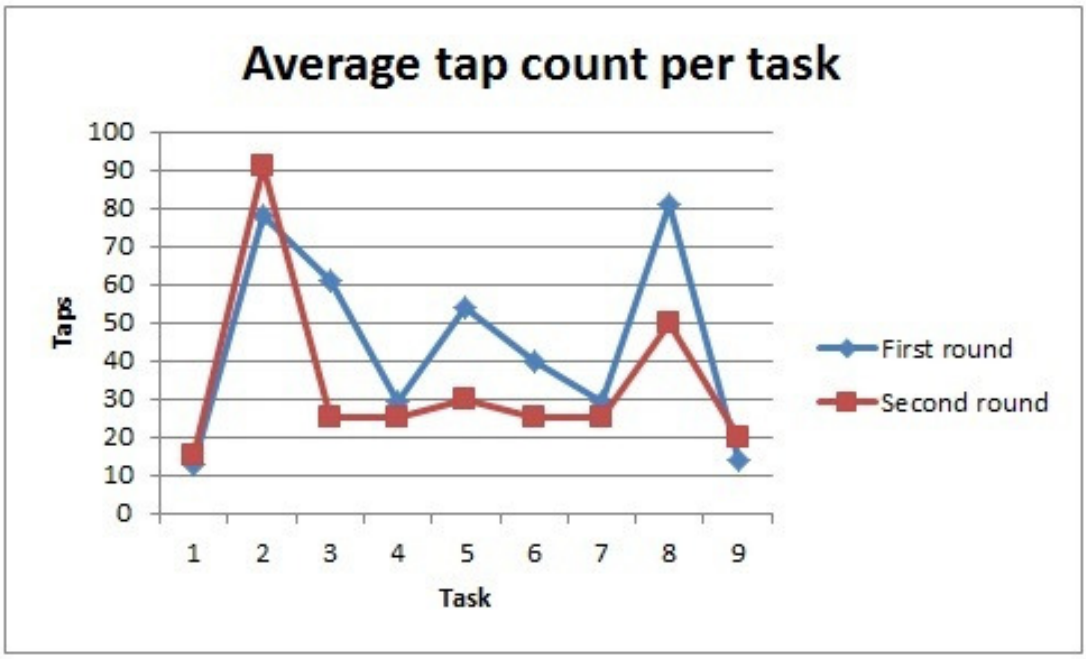


Figure 5.2: Chart that represents how many taps were required to finnish given task.

that the subjects were not used to consistent application user interface with other applications. Because of that navigation hierarchy was for them confusing. Profile and monitoring sections were with the least lost count. This may also be inaccurate because of the same reason. After user is logged in, first tab is WAN with profile management and monitoring and because of this, subjects were getting lost more if they were supposed to switch tabs.

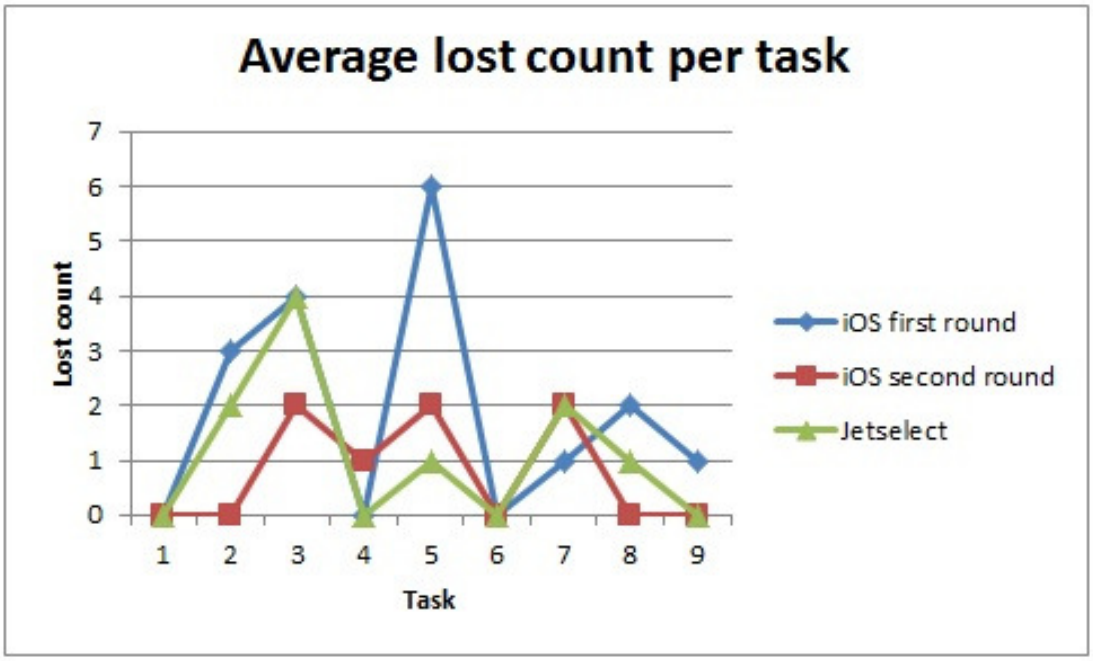


Figure 5.3: Chart that represents how many times subjects have gone the wrong direction in navigation hierarchy.

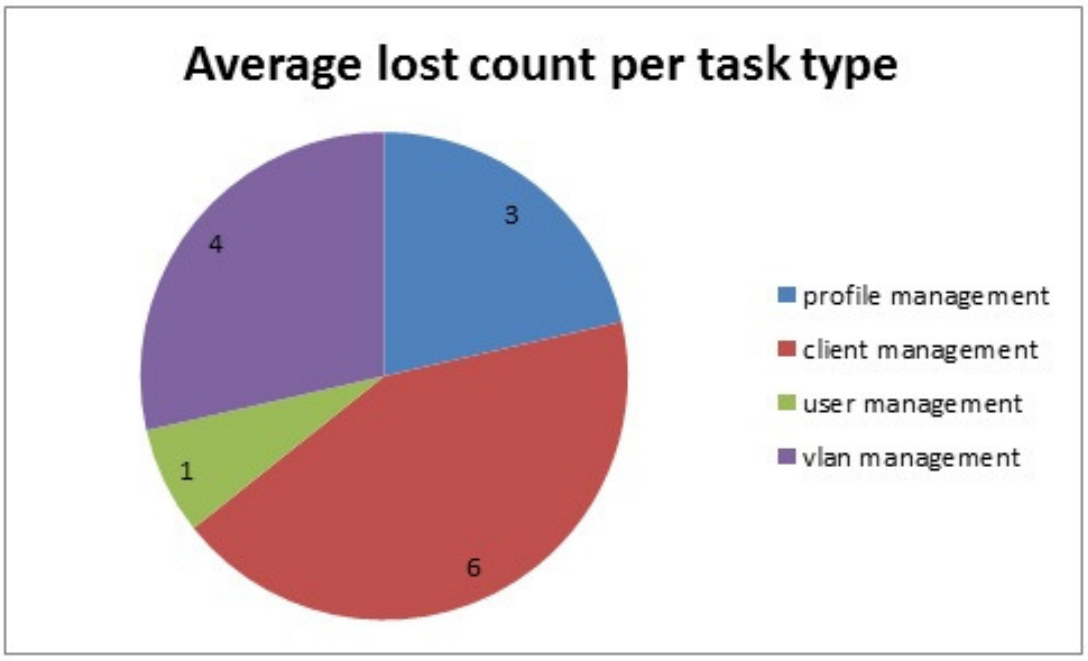


Figure 5.4: Chart that shows the lost count per task type.

5.2.4 Overal quality comparison

Every subject was asked to compare both products. Result of that is shown in figure 5.5. Navigation seemed easier in Jetselect, main reasons were explained in previous sections.

Apart of that, another reason why navigation seemed complicated is, that subjects had hard time to distinguish between profiles, drafts, users, vlans and clients. Several times happened that subject had got to profile management and was trying to create a new client. This is a serious issue that will be addressed in next chapter.

On the other hand, data manipulation was much easier in iOS application. That means that table views with detail views were a good choice.

The rest of quality comparison compares different sections of application, but these were heavily influenced by the navigation and manipulation results. There rest comparisons resulted in comparable values for both Jetselect and iOS application.

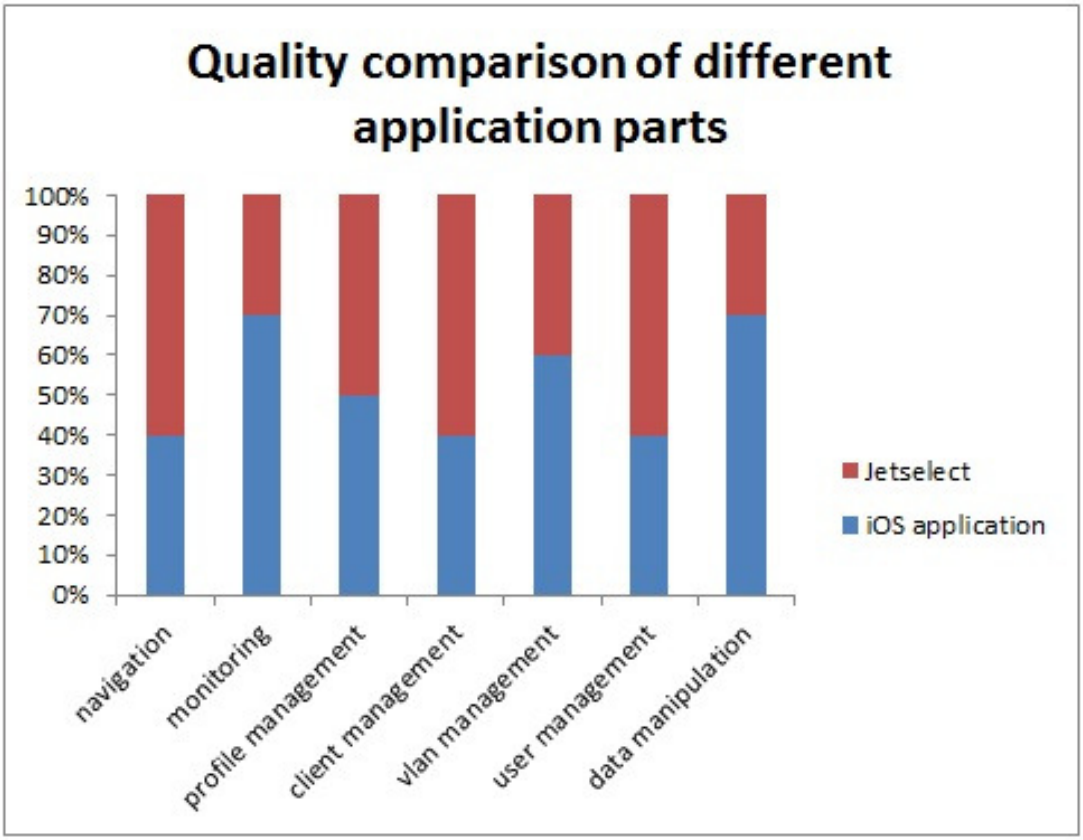


Figure 5.5: Subjects quality comparison between Jetselect and iOS application for different sections.

Chapter 6

Future plans

In previous chapter several drawbacks of current design were addressed. The biggest issue was navigation hierarchy, that was confusing for testing subjects. To improve this, every section should have labels to describe the goal of that particular section. Even views should have a short informational label about what can be done in that particular view. By doing that, lost count and time to perform tasks can be considerably reduced.

Apart of that, several additional things need to be done, to bring this application to a whole new level. Following things will be added in the future:

- making landscape orientation for the application available. This is currently not implemented, because of large amount of data (mostly in forms) needs to be displayed in several views and landscape orientation is not suitable for that. However, if suitable user interface design was found, it could improve user experience with this application.
- Making this application iPad compatible. As was previously stated, iOS runs not only on iPhones and iPods, but on iPads too. No functionality would be necessary to change. However, in user interface a different approach should be used. Because of that a complete new design should be made for iPads. Doing this would made this application more accessible.
- Introducing tap gestures to make work even faster. Tap gestures are powerful tool, that this application did not take advantage of yet.
- The most important part and most challenging part however is testing this application on live servers. This would need to change the communication protocol of application to adapt to server. And only after that, it can be determined if this project was fully successful on all levels, not only on the user interface part.
- Bringing more functionality to the application to adapt capabilities of the servers. Current server, even the one in this project has full tracking of history and database changes available. Browsing this history, reverting changes and making statistics based on gathered data should be introduced to this application.

After implementing these stated items, this application may have a potential to be used in travelling industry. In more distant future, more things can be introduced, but that will be determined by the fact, if the project will be successful. If so, and all previous items were implemented, NTP and SNMP protocols may be introduced. However those are beyond the scope of this document.

Chapter 7

Conclusion

Previous text described one of the challenges that information technologies face in travelling industry. Network devices may need to be reconfigured several times to stay connected to the internet during the travels. To help, this project made analysis of this challenge and currently available tools. HTML application Jetselect already exists, but is not compatible with the operating system iOS. Because of that a new application was designed and implemented in this project in chapter 4.

Testing phase and results are described in chapter 5 and they showed what drawbacks this design has and several issues were addressed. However, in overall testing showed that this application has comparable results with Jetselect.

Future tasks for this application will be improving the navigation hierarchy, making landscape orientation available and iPad compatible, tap gestures will be introduced and application will be tested on live servers. Possible extensions are NTP protocol and SNMP protocol implementations. After items in previous chapter will be implemented, the application has potential to be used in practice. Therefor, the result of the project was successful.

Bibliography

[1] Apple Inc. Cocoa application competencies for ios. `developer.apple.com`, 2012-09-19.

[2] Apple Inc. ios technology overview. `developer.apple.com`, 2012-09-19.

[3] Apple Inc. Cocoa core competencies. `developer.apple.com`, 2013-04-05.

[4] Apple Inc. Streamline your app with design patterns. `developer.apple.com`, 2013-04-23.

[5] Apple Inc. Write objective-c code. `developer.apple.com`, 2013-04-23.

[6] Apple Inc. ios human interface guidelines. `developer.apple.com`, 2013-05-01.

[7] Hruška Tomáš and Radek Burgeti. Internetové aplikace (wap) iv. část programování serveru (php) [online]. `https://www.fit.vutbr.cz/`, 2012 [cit. 2013-1-17].

[8] WWW pages. `http://www.mtnsat.com/`.