CHALMERS
UNIVERSITY OF TECHNOLOGY

iPhone Application Development

LiveScore Göteborg

Degree Project, Bachelor thesis in Computer Science

Timocin Zaynal

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2015

iPhone Application Development
Livescore Göteborg

Timocin Zaynal

Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Tel: +46-(0)31-772 1000
Fax: +46-(0)31-772 3663

Department of Computer Science and Engineering
Gothenburg, 2015

Preface

I would like to thank my brothers for helping me with different ideas, and also Uno Holmer for giving clear guidelines to build the thesis the best possible way and any other assistance I needed along the way.

Abstract

The demand for live information in the sports community is growing every day. Thanks to the Internet, new kinds of information delivery services are being implemented for the top football leagues. However there is one area that remains untouched, the lower leagues. Fans supporting their local teams do not have the means to receive live updates and information. Few problems prevent the development for this, the quantity of different leagues and obviously popularity. In terms of business, it is not lucrative. Crowd-Sourcing was used to overcome these problems. A low-maintenance server-client solution is implemented to handle the information which is populated by the “crowd”, in this case the board members of a football club.

Abstrakt

Efterfrågan för live information i idrottsrörelsen växer varje dag.
Nya typer av informations-leveranstjänster genomförs för de bästa fotbollsligorna tack vare Internet. Men det finns ett område som är orörd, de lägre ligorna. Supportrar som stöder sina lokala lag har inte möjlighet att ta emot live uppdateringar och information. Enstaka problem hindrar utvecklingen för detta, mängden olika ligor och naturligtvis popularitet. När det gäller affärer är det inte lönsamt. Crowd-Sourcing kan användas för att lösa dessa problem. En simpel serverklientlösning implementeras för att hantera den information som uppdateras av "publiken", i detta fall styrelsen i en fotbollsklubb.

Table of Contents

Definitions

API (Application Programming Interface) is a set of programming instructions and standards for accessing a Web-based software application or Web tool. There are many API that is released to the public so that developers can design products that are powered by its service.

Asynchronous means that the program permits other processing to continue before the transmission has finished.

Git is an open source repository with a full version-tracking capability.

GUI (Graphical User Interface) is an interface that allows users to interact with electronic devices through graphical icons.

IDE (Integrated development environment) is a programming environment that has been packaged as an application program, normally consisting of a compiler, code editor, a debugger and a graphical user interface

Livescore is a type of service offered by many sports-related websites and broadcasters as well as online sports betting operators. The idea of livescore is to provide real time information about sports results

1. Introduction

1.1 Background

Football, also known as soccer in United States of America, is the world's most popular sport played by an estimated 250 million players in over 200 countries [01]. It is a sport that has enchanted billions of people, and a market within Information Technology has grown to cater for the needs of these fans. Getting live news of football games has become increasingly more popular and addictive in contemporary times community. Thanks to the current technology supporters can follow their teams and be updated in real time on their smartphones or websites. However, this is only available for the top league teams such as Allsvenskan to Division 2 in Sweden.

The idea of making a livescore application for the lower leagues was brought up when talking to a friend after a soccer game since we wanted to know the score of another game that started around the same time.

1.2 Problem

The project will be focusing on building an iPhone application that will be able to provide real-time updates of soccer statistics to the clients. The mobile application was developed in Objective C for Apple iOS devices and takes advantage of web technologies such as JSON and HTTP-requests.

1.3 Purpose

In this project, I will introduce a mobile application in conjunction with an online website that will utilize crowdsourcing to provide the real-time information for lower league teams in Sweden.

1.4 Requirements

The goal is to make an application that gets the information that it needs automatically every day. The application will have five views that the user can navigate through (see annex A).

1.5 Boundaries

In this project, the application will only be displaying game information of one league. The focus is put mainly in the functionality instead of the GUI. Statistics such as team line up and substitutes will not be included.

2. Method

Initially, some research will be made to see if there is an API that can provide information about the games, dates and times for the leagues the app is going to show. From that research, a choice will be made to choose between an already existing API or to create something similar. A decision will be taken on how to save information about the games and how clients will be able to retrieve it. Since there are so many different methods and third party libraries that help with this, an assessment will be made as a basis for the choice of the best-suited technologies.

2.1 Planning and organization

An efficient way to work on mobile app development is to work agile, meaning breaking down the project into smaller pieces [02]. Planning a reasonable timeline is done next by estimating how much time each piece that was broken down would take without making the project to complex.
Using this method makes it easy to detect when problems occurs and then iteratively try new functionalities through simulation, experience faults and go back to code. It also helps to keep track of the time meaning the risk of falling behind in the time schedule is minimized.
Another benefit is when working in a group, since the whole project is now in many pieces, everybody can get their share of tasks to chew on. Figure 2.1 shows the gantt-scheme.

Aktivitet	Startdatum	Dagar att slutföra	Slutdatum
Bas Design/ Funktionalitet	2014-09-17	2	2014-09-19
Parsa från hemsida	2014-09-19	7	2014-09-26
Kommande/Aktuella matcher	2014-09-26	10	2014-10-06
Funktionalitet för att uppdatera ställning	2014-10-06	7	2014-10-13
Sätta upp databas för inloggning samt funktionalitet	2014-10-13	7	2014-10-20
Finslipa Design/buggar	2014-10-20	3	2014-10-23
Eventuellt Lagra lag i databas för sökning av serietabell	2014-10-23	7	2014-10-30
Eventuellt kontakta lagen i en division ochgeuppgifter	2014-10-30	3	2014-11-02
Skriva rapporten	2014-11-02	21-28	2014-11-28

Figure 2.1. *Time schedule.*

2.2 Developing the application

Most of the coding will take place in Xcode. Xcode is an IDE developed by Apple for developing software for OS X and iOS. The project was saved every time a new functionality was implemented using Git. Implementing new functions does not always work out, thus saving a working project is good since it is easy to revert if things go wrong

2.3 Testing and verification

The implementation order for the functions is set in a special way. Since some functions rely on other components to work, it is smart to first implement the root function, which does not rely on other components. After that, the path is made just like a tree's branch spreads out and connects to more branches. Same idea goes here. After making the root function and testing that it works, it is easier to move on to the next function that relies on the root function. Then test those two functions if they work together. Using this method is good because a developer can iteratively test the functions to see if everything is in its order. It is hard to test a method when it relies on a method that has not been done yet.

3. Technology Summary & background

This chapter describes the different techniques used in this project.

3.1 HTTP Request Methods POST/GET

HTTP is used for communication between client and server. The most commonly used requests are POST and GET. POST-requests are used when the client sends information to the server to be processed. GET-requests are used when the client needs information stored in the server. It is a request-response protocol. In this project, the website acts as a server and the users who are using the application are the clients. Visiting a page is a typical example of a GET-request [03]. Figure 3.1 below shows an example of how a request is made in Objective-C.

```
NSMutableURLRequest *request = [[NSMutableURLRequest alloc] init];
[request setHTTPMethod:@"GET"];
[request setURL:[NSURL URLWithString:url]];
```

Figure 3.1. *Example of how to initialize and create a request with a predefined URL variable.*

3.2 JSON

As the clients and servers want to communicate, they need to use a data-interchange format. A few known data interchange formats are JSON, XML, YAML and REBOL. An example of usage is web services responses. Before JSON appeared, XML was the preferred one but that changed, since JSON is much more lightweight and because it has much larger support. A web browser that supports JavaScript can process JSON [04].

3.3 HTML

HTML (HyperText Markup Language) is a markup language rather than a programming language since it describes the structure of a website semantically. It is used to make websites that can be viewed by anyone connected to the Internet. Hypertext is what allows a user to click on links. It is what makes it possible to jump from page to page. These special links are called hyperlinks. Markup is what properties a text will have inside the tags. Such as bold or underlined text.

There is plug-ins called HTML-helpers that help extending the standard functionality of a web browser. They are often used to display maps, verify bank id, etc.
A standard well known plug-in that is used are Java applets [05].

3.4 Python

Python is a high-level programming language. It is very simple and easy to learn. Since the language is so simple it makes it very easy to read. The code efficient syntax used in Python makes it easy to do something in fewer lines of code than what would be needed in Java or C++ [06].

3.5 Dictionary

Dictionaries are made of pairs also known as items. The items consist of a key and their corresponding values. Searching in a dictionary is faster since each item has a unique key. Each key is separated from its value by a colon and each item is separated by a comma [07]. Figure 3.2 show an example of a simple dictionary in Python.

```
dict = {'Alice': '2341', 'Beth': '9102', 'Cecil': '3258'}
```

Figure 3.2. *Example of a simple dictionary with the keys to the left and the values to the right.*

3.6 LXML & XPath

LXML is a library made for parsing XML-and HTML-documents very quickly [08].
XPath is a query language for selecting nodes from a document. A node can be an element node or an attribute node. XPath is often used to compute values from the contents of a document after parsing the document [09].

3.7 Objective - C

Objective-C is an object-oriented programming language. It is the main language used by Apple. It is a superset of the C programming language meaning the syntax is very alike. iPhone applications like OS X applications are written in Objective-C [10].

3.8 Apple SDK

The SDK (Software Development Kit) for IOS was released in March 06, 2008. The SDK makes it possible for developers to build applications for iPad and iPhone, as well as test them in a simulator. After simulation is a success, and the application seems to work fine it can be tested in a real device. An Apple Developer License is needed to do that. Paying a fee of 99$ / year gives the ability to test it in real devices as well as uploading the application to App Store. Xcode is the development environment for the IOS SDK [11].

3.9 Web Server

A web server is what processes HTTP requests. HTTP is the basic network protocol used to distribute information on the World Wide Web. There are many areas where the web server can be used. The most common areas are gaming, data storage, host a website and running enterprise applications [12].

4. Implementation

4.1 Base design

There are a few common templates that can be chosen from, such as: Master-Detail Application, Page-Based Application, Single View Application, Tabbed Application and Game. Each template gives a different advantage depending on what the developer are looking for. Meaning no pre-configuration is needed to be done.

4.1.1 Application template

No research was needed to choose a design template for the application since it all depends on the creator's taste. Tabbed Application was chosen since it has a very simple and easy navigation system. The functionality that it provides fits perfectly with what this application needs. The different views will be accessed by the tab bar icons representing the views as shown at the bottom of TabBarController. The icons will be changed, and a keyword will be added below the icon so the navigation will be easy for the user. Example of a Tabbed Application is shown in figure 4.1.

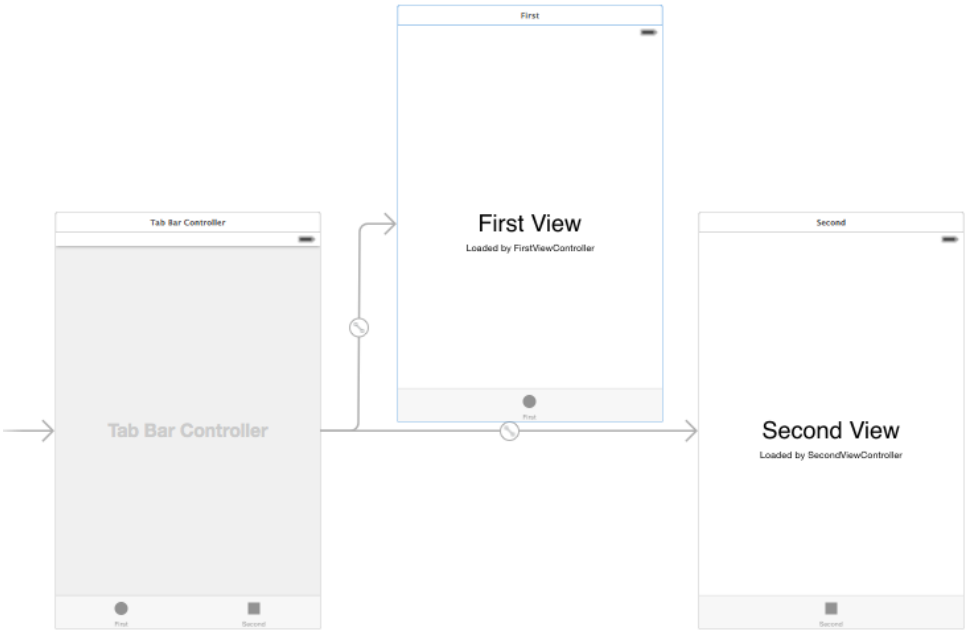

Figure 4.1. *How a basic Tabbed Application looks in its earliest stage.*

4.1.2 Integrating UITableView

Displaying information to a user can be done in many ways. One of them is by using tables. Integrating UITableView is done by deleting the views as seen in figure 4.1, then drag and drop a UITableViewController and set up the tab bar connections to the new Table views.

4.2 Information gathering

The only information the application needs from the web is which teams that are going to play on a certain date and time. The app will take that information once every day and store it in a text file as a dictionary which we will talk about later. The information the application needs is shown in figure 4.2.

Kommande matcher		
Tid	Match	Arena
2014-11-15 16:00	Göteborg Futsal Club - Skoftebyns IF	Frölunda Kulturhus Sporthall
2014-11-16 14:00	Malmö City FC - Avenyn United FC	Limhamns sporthall
2014-11-16 14:30	Gothia Futsal Förening - Borås AIK	Angereds Arena Sporthall
2014-11-16 19:00	IFK Uddevalla - Futsal Club Frölunda	Agnebergshallen
2014-11-20 20:30	Göteborg Futsal Club - Borås AIK	Lisebergshallen

Figure 4.2. *Example of a table to take relevant information from.*

4.2.1 Import.io

Import.io is a platform that allows anyone, regardless of technical ability, to get structured data from any website [13]. At the first glance Import.io seemed like the perfect solution for information gathering. Integrating it into the application and using it was very easy. But there is one problem, Import.io did not support schedule re-crawls, meaning that the application is supposed to run on its own and get new game information once per day so Import.io is not an option.

4.2.2 Kimonolabs

Kimonolabs offers the same service as Import.io does with the addition of scheduling crawling. It is easy and simple to use. The downside was telling Kimonolabs the information to retrieve. It selects the whole table when a table is clicked. Since the idea is to show the games with the current date, this did not seem appropriate since it gathered unrelated information that would not be used. That results in slowing the application down.

4.2.3 Python & LXML

After trying to use existing information gathering services without any luck, I decided to code it myself using Python. Choosing this method to get the information is much better than using existing services for several reasons:

- getting exactly what is needed for the application without any excess information,
- easy to tweak and change the output format if needed in the future when upgrading,
- does not depend on third parties for the application to work.

There are many libraries that help to parse HTML documents such as BeautifulSoup, LXML and Pythons built in HTML.parser. After some research LXML was chosen since it provides simplicity and good performance [13].
Accessing the specific information in a website is done by using LXML's built-in function XPath. Figure 4.3 illustrates usage of Xpath & LXML.

```
currentDate = (time.strftime("%Y-%m-%d"))

rows_xpath = XPath("//*[@id='content-primary']/table[3]/tbody/tr[td[1]/span/span//text()='%s']" % (currentDate))
time_xpath = XPath("td[1]/span/span//text()[2]")
team_xpath = XPath("td[2]/a/text()")

html = lxml.html.parse(url)

for i,row in enumerate(rows_xpath(html)):
    time = time_xpath(row)[0].strip()
    team = team_xpath(row)[0]
    homeTeam, awayTeam = team.split(" - ")
```

Figure 4.3. *Example of extracting game time and team names for the current date in Python using LXML & XPath*

4.3 Storing information

There are many different ways of storing data. Two data types that are common are lists and dictionaries. The datatype that was used to store the information mentioned above was lists. It started out great but when things started to get complicated a new data type was needed to ease the flow of the new information (dictionaries). Example of how a list can be used is shown in figure 4.4

Index	Value
0	January
1	Feb
2	Mar
3	Apr

Figure 4.4. Example of a list. Every index has a value that can be returned by giving the corresponding index number

4.3.1 Nested dictionary

A standard dictionary works fine in most cases. As the information starts to grow hierarchies are needed. Dictionaries that have hierarchies are called nested dictionaries. They are like dictionaries in dictionaries [15]. Figure 4.5 shows how a nested dictionary can look like.

```
Vastra gotalands lan{
  Div 4A Herrar{
0
     homeScore : 0
     awayTeam :  Floda BoIF
     homeTeam : SKIF Semberija
     awayScore : 0
     time : 14:00
1
     homeScore : 0
     awayTeam :  Kode IF
     homeTeam : Partille IF FK
     awayScore : 0
     time : 14:00
2
     homeScore : 0
     awayTeam :  Romelanda UF
     homeTeam : IK Kongahälla
     awayScore : 0
     time : 14:00
```

Figure 4.5. *Example of a nested dictionary structure. "Vastra gotalands lan" is an item, "Div 4a Herrar" is another item but belongs to "Vastra gotalands lan" and all the games with id's from 0 to 2 are items that belongs to "Div 4A Herrar".*

4.3.2 Checking game time

The time of all the soccer games must be updated every time new data is requested from a client. That is why the game time must be checked and calculated for every GET-request.
If the game has started, it calculates the time difference from the time the game was started with the current time the code was executed. The difference in time will be stored in the time variable. The time variable will always go through three states in order:

1. The time it initially receives from the parsed information.
2. When the game is active and the time variable changes according to the time difference.
3. When the game has ended and is given the time value of “FT” (Full Time).

Using this method requires us to have two time variables. One that always keeps the initial value when parsed that is used every time the difference is calculated and the other one that is used to present to the client.

4.3.3 Printing as JSON

Using Python’s built-in method JSON made it really easy to print the dictionary as JSON text.

4.4 Server

Now when the parsing and storing information side is done it is time to work on the server.
The idea is that the server will parse for new games once every day around 00:00. The server then stores that information in a text file as JSON to serve the clients. The server should be able to respond to GET-requests for now, but POST-requests is yet to be implemented. After some research, Heroku, AWS (Amazon Web Services) and Microsoft Azure were found that could act as the server. Heroku was chosen since it supported Python and had good tutorials on how to get started.

4.4.1 Flask

To make web development possible with Python, a web framework is needed. Python offers several great frameworks such as Django, Flask, TurboGears and some more. The best suited for this application was Flask.
The reason why Flask was chosen was because it is lightweight which makes it easy to get started [16]. Since our web servers job is only to receive POST-and GET-requests a lightweight framework seemed logical.
The other frameworks were a little heavier, and the documentation was not as elegant and simple as Flask.

4.4.2 Heroku

Heroku is a cloud platform as a service supporting several programming languages. Using services like Heroku allows developers to forget about the infrastructure such as managing servers, deployment and scaling. It grants the wishes of many developers only to have to focus on the application.
Heroku has a good “get started with Python” tutorial that explains step by step what needs to be done in order to host a web server that supports Python [17].

4.4.2.1 Set up Heroku

These are the following step that needs to be done to get started with Heroku:

Registration was easy and after that the tool belt was downloaded. The tool belt is used so all Heroku commands can be executed from a terminal.

Creating a Heroku environment was the next step and was done by the tool belt. The modules and LXML that were used in the Python code had to be installed in this environment.

Procfile is needed to declare what command to run to start the Heroku application. Example of a procfile:

web: gunicorn Work: app --log-file=-

Where web declares that the process type will be attached to the HTTP routing stacks of Heroku and receive web traffic when it is deployed. Work declares the name of the Python file to be executed.

Requirements file was created to specify which modules are needed to run the Python code. Heroku will read this file and install the appropriate dependencies. The creation was simple using the tool belt.

After the setup it is ready to be tested. With the tool belt came a tool called foreman that lets the user run the application locally to test before deploying it to the web.
Running foreman deploys the application locally at localhost:5000. Figure 4.6 shows the result.

Uploading the application to the web via Heroku's tool Git will be done when the project is done. Until then, the work environment will be held in localhost. GET request is now done since localhost:5000 displays the correct information.

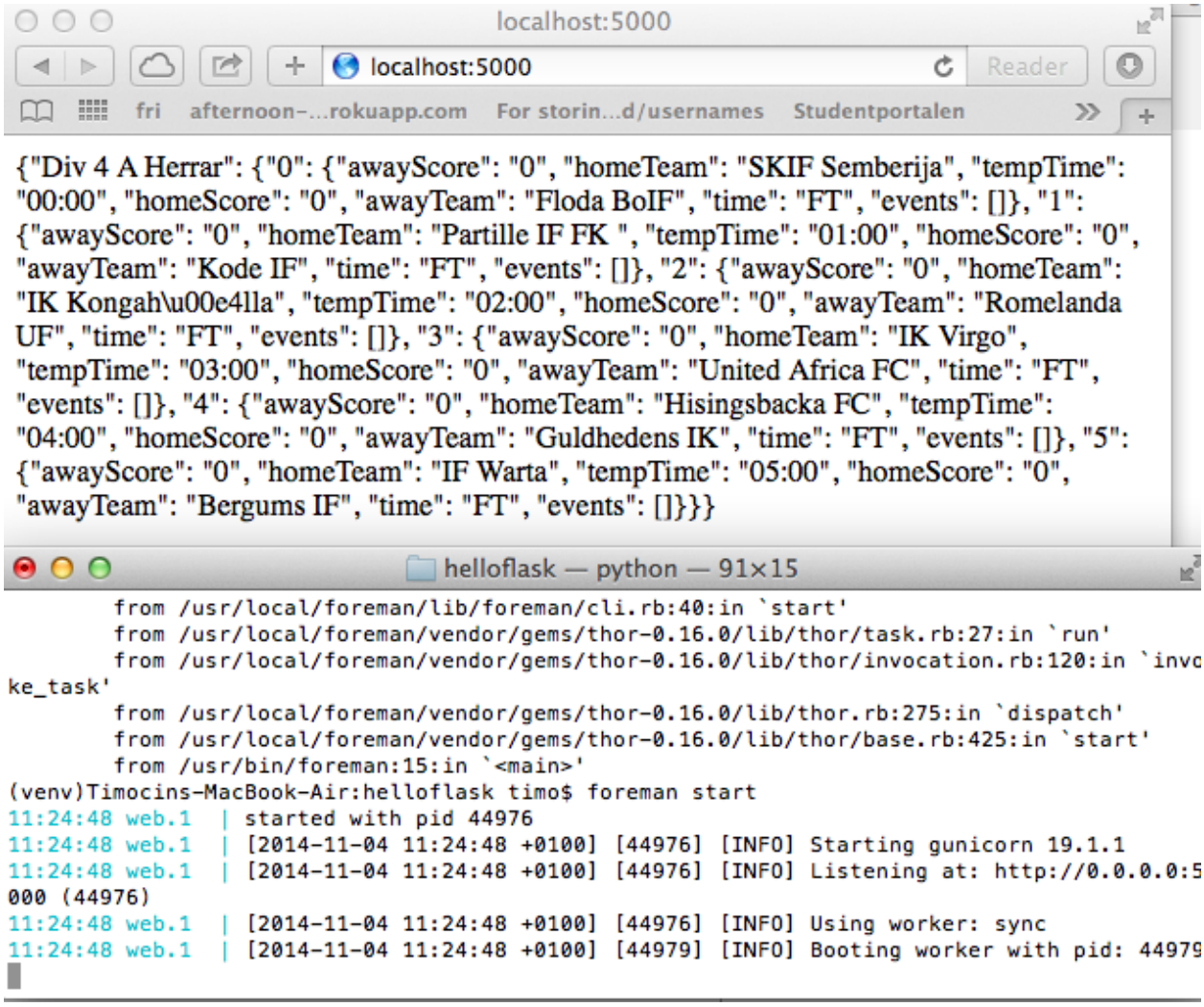

Figure 4.6. *Example of a JSON dictionary dump and Heroku toolbelt using foreman.*

4.5 Client

It is time to implement the client side. The server side is ready to process GET requests, but there is nobody reaching out to the server for now. There are built-in methods in Objective-C such as NSURLConnection and NSURLSession to load the contents of a URL.

4.5.1 Making a GET request method

We now have a web server that delivers the JSON information of the games that we need to display in the application. Every time the client opens the application a GET-request will be

made to get updated information. Making a GET-request can be accomplished with the following steps:

1. Make the request and embed the URL in it. The URL points to the source we want the information from.
2. Define the request type POST/GET.
3. Run the request and get the JSON as an NSData object.
4. Parse the received JSON to a dictionary using NSJSONSerialization.
5. Send the dictionary to the class that requested the information.

This method is fired every 60 seconds to update information on the view that the client is active in. The GET task is set to be done asynchronously. Since getting data can be time consuming, not doing it asynchronously, increases the risk that the user will not be able to navigate while it is processing the data.

4.5.2 Populating the data

Populating the table was done very easily using the built-in methods for a UITableView. Figure 4.7 shows how the information is being presented.

Figure 4.7. *Example of a populated table.*

All the rows use the same row template design called a cell. There are small differences in the cell's different time states. Instead of making two different templates for the different states it is more effective to hide information that is not needed. The arrow to the right means that the row is clickable to show additional information.

4.5.3 Authentication

Authentication is needed to prevent everybody from being able to send updates to the server. There will be no registration form since everybody is not allowed to update. Predefined username and passwords will be defined for each team to use.

4.5.3.1 Parse.com

Parse.com is a service that provides numerous features to enhance an application. It offers push notifications, data storage and analytics. Analytics gives valuable information of when the application is used the most and if it is getting popular. Using Parse.com SDK made it easy to check if the client is providing correct information to get authorized. Adding a new user can be done anytime using a curl command or by manually logging in to Parse.com and directly creating a new user. Figure 4.8 shows an error message when authentication is failed.

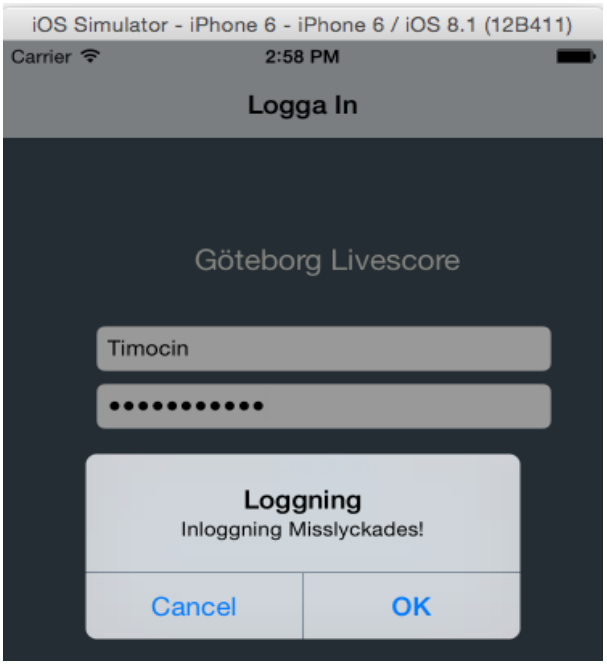

Figure 4.8. *Example of the login form. If login is a success the client gets directed to the Live Games view.*

4.5.4 Making a POST request method

Now that the basic information is presented it is time to implement the POST method to update statistics. A boolean variable is used to check whether the client is logged in. If the user is logged in an extra view will be visible to the client. The singleton class was used to implement the boolean variable since many classes will use the same variable. Using singleton means that only

one instance of the boolean variable is created which is crucial to maintaining the correct state which is either true or false.

4.5.4.1 Saving POST Information (Server)

We have to store the information to save the scores for the teams. As mentioned earlier, all the information is stored as JSON in a text file. In order to update the values given by the POST-request, these steps are taken:

1. Load the information in the text file to a dictionary variable in the Python code to add the changes
2. Disassemble the POST-request to get the gameID, this is done to know which game field to change in the dictionary.
3. Apply the changes to the corresponding team.
4. Dump the information back to the text file again so it can serve the new information to the clients.

5. Result

All the views that were mentioned in the Problem section is finished and fully functional. The code structure was built in a way making it easy to include other leagues to the application with a single row.

5.1 Server

The server is finished and prepared to handle the basic HTTP requests that are needed for the application to run. The server can do the following:

- handle GET requests to deliver information to the client,
- receive POST requests from the client and use that information to update stored information,
- parse website once per day to get the new upcoming games.

5.2 Client

The client is all set and ready to enjoy the implemented functionalities and can do the following:

- view live, upcoming and finished games,
- get authorized to update statistics on a game.

5.3 User Interface

The following images will show each view of the application with a simple description of what is being presented.

Figure 5.1. *View that shows all games.*

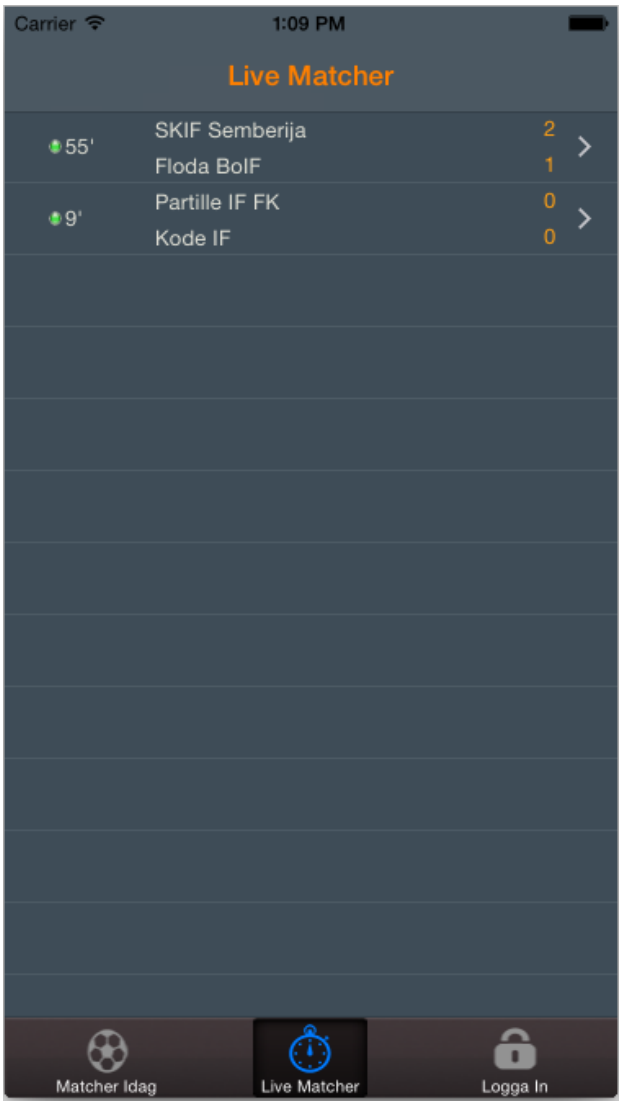

Figure 5.2. *View showing only live games*

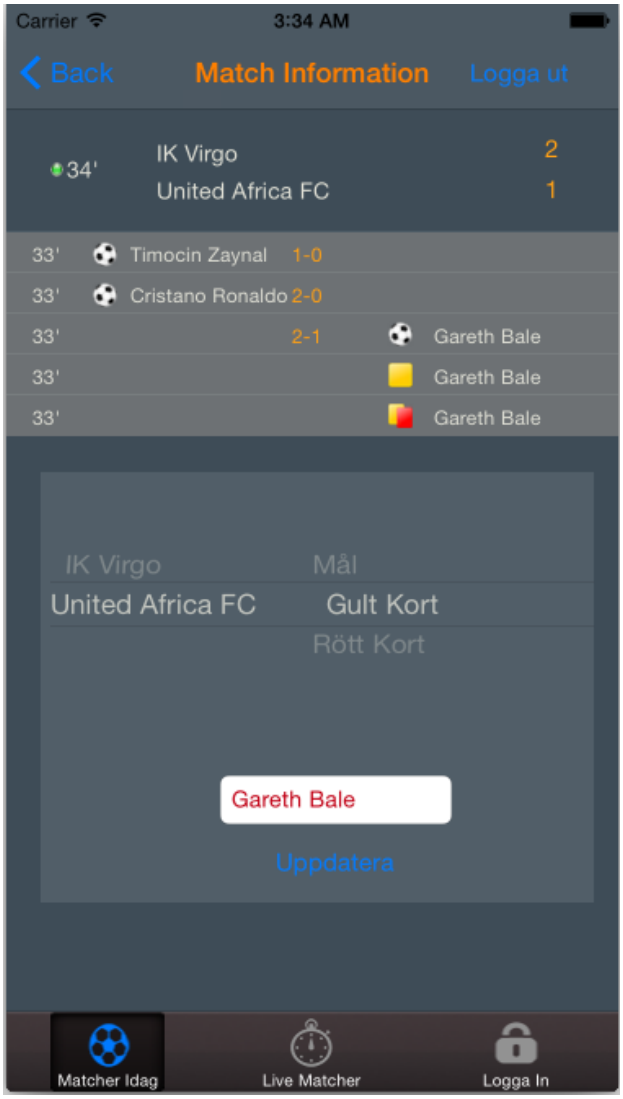

Figure 5.3. *Detailed view of a game showing statistics and the option to update*

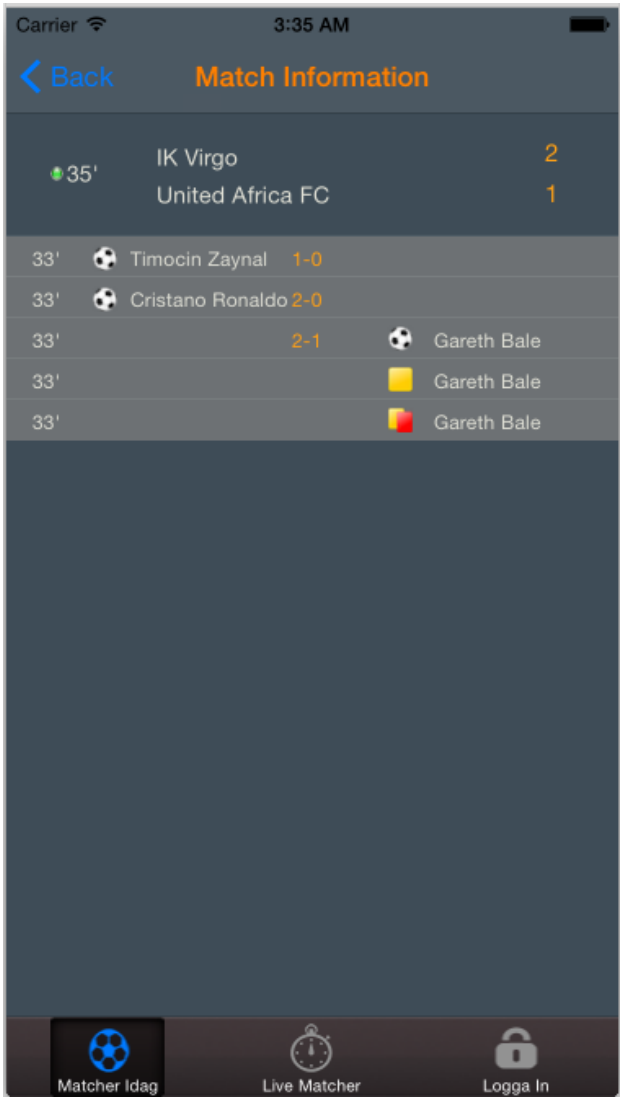

Figure 5.4. *Detailed view without a user logged in*

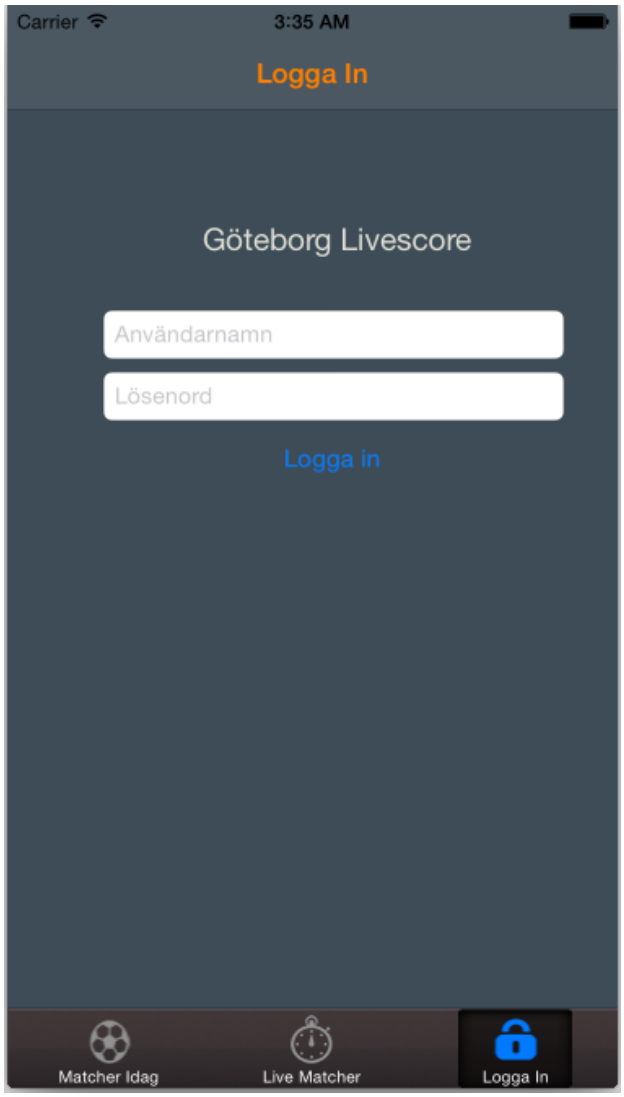

Figure 5.5. *Login view*

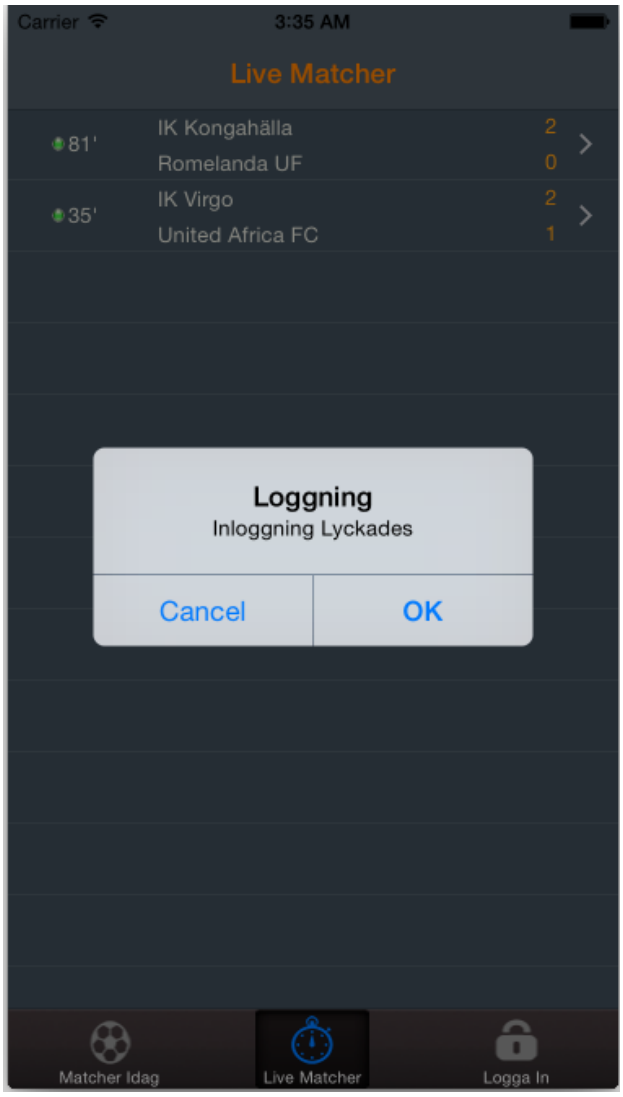

Figure 5.6. *Pop up after successful login*

6. Conclusion

6.1 Environmental Aspects

The application does not just make life easy for soccer fanatics to keep tabs on teams, but it also helps environmentally. Since it is possible to know the score without traveling to the game means that there would be less carbon dioxide released if the traveler's choice was something that releases carbon dioxide.

6.2 Security & Ethics

Since the project was about building the application not much weight was put on writing my own code for security. Instead, ready services such as Parse.com was used to handle the data communication and password storing in a secure way. The application will also be reviewed by Apple before it is ready to be published in Appstore meaning any security risks in the code that I may have missed will be noticed by them. This does not mean that I did not put an effort on writing the code as secure as possible.

To know what kind of information to store in a database can be an ethical question. Since the users are created for the clients and no personal information is being provided from the clients there is not so much to talk about.

6.3 Project Planning

The planning went better than I predicted it would. It is hard to anticipate how long it would take to implement a function when it is the first time making an iPhone application. Looking at the Gantt-diagram, the only thing that did not become finished was implementing a database. That will be on hold for future improvements. Much time was used doing research of what libraries to use that fits best to my situation. Both Python and Objective-C languages were new to me. Getting comfortable coding in those environments and learning the syntax takes some time.

6.4 Reaction from the teams

The response that I got from the teams I contacted was wonderful. After all hard work, it all came down to the point on how the teams would react to run an application that uses crowdsourcing. It was hard to find time that matched both my and the teams schedule. With the limited amount of time on my hands I managed to arrange three meetings with different teams. After a short presentation of the application, all three teams said they loved it and was looking

forward to working with it. The meetings were held with the team's trainers and board of directors of the following teams: Kungsladugård BK, IF Väster and SG 97.

6.5 Future Improvements

6.5.1 Adding league tables

The application is made to make it easier to see the results or experiencing live games without being there. Another great feature would be to view the table for each league. A table contains the rankings for each team, amount of points, goals, games played, etc. Implementing this means that the users would not just use the application when there are live games, but also on a daily basis to view rankings for all the teams.

6.5.2 GUI

There will be a lot of changes in the design. As pointed out in Boundaries in Chapter 1. Most of the time was spent making the application work as it should. There were no thoughts behind when choosing colors and making the design for the application.

6.5.3 Support for Other Platforms

A good way of getting more users is to support multiple platforms. Making a website will be easy since the backend server is already finished. The website will only be used to view the scores, upcoming games and results. There will be no POST method since a website is usually accessed by computers and it is not very often that fans brings a computer to a soccer game. Developing for Android will also be easier for the same reason. However, a POST method will be needed.

Works Cited

Works cited in order of appearance.

[01] C.Giulianotti. R. *Football*.

http://www.britannica.com/EBchecked/topic/550852/football. (2014-11-04)

[02] Pahuja.S. *Agile for Mobile Application Development*.

http://www.infoq.com/news/2014/10/agile-mobile-application. (2014-11-03)

[03] Refsnes Data. *HTTP Methods: GET vs. POST*.

http://www.w3schools.com/tags/ref_httpmethods.asp. (2014-10-08)

[04] JSON.

http://en.wikipedia.org/wiki/JSON. (2014-10-10)

[05] *HTML*.

http://en.wikipedia.org/wiki/HTML. (2014-10-09)

[06] *Python (programming language).*

http://en.wikipedia.org/wiki/Python_(programming_language). (2014-10-07)

[07] *Tuples, Lists, and Dictionaries.*

http://www.sthurlow.com/python/lesson06/. (2014-10-15)

[08] A.Kenneth Reitz. *HTML Scraping.*

http://docs.python-guide.org/en/latest/scenarios/scrape/. (2014-10-15)

[09] *XPath.*

http://en.wikipedia.org/wiki/XPath. (2014-10-17)

[10] *About Objective-C.*

https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html. (2014-11-20)

[11] *IOS SDK*.

http://en.wikipedia.org/wiki/IOS_SDK. (2014-11-20)

[12] *Web server*.

http://en.wikipedia.org/wiki/Web_server. (2014-10-13)

[13] *What is import·io?*.

http://support.import.io/knowledgebase/articles/251955-what-is-import-io. (2014-09-24)

[14] Behnel.S. *Benchmarks and Speed*.

http://lxml.de/performance.html. (2014-10-03)

[15] *Python Dictionary*.

http://www.tutorialspoint.com/python/python_dictionary.htm. (2014-10-12)

[16] *Flask (web framework)*.

http://en.wikipedia.org/wiki/Flask_(web_framework). (2014-10-23)

[17] *Getting Started with Python on Heroku*.

https://devcenter.heroku.com/articles/getting-started-with-python#define-a-procfile. (2014-10-11)

Annex A

Flow diagram of the application

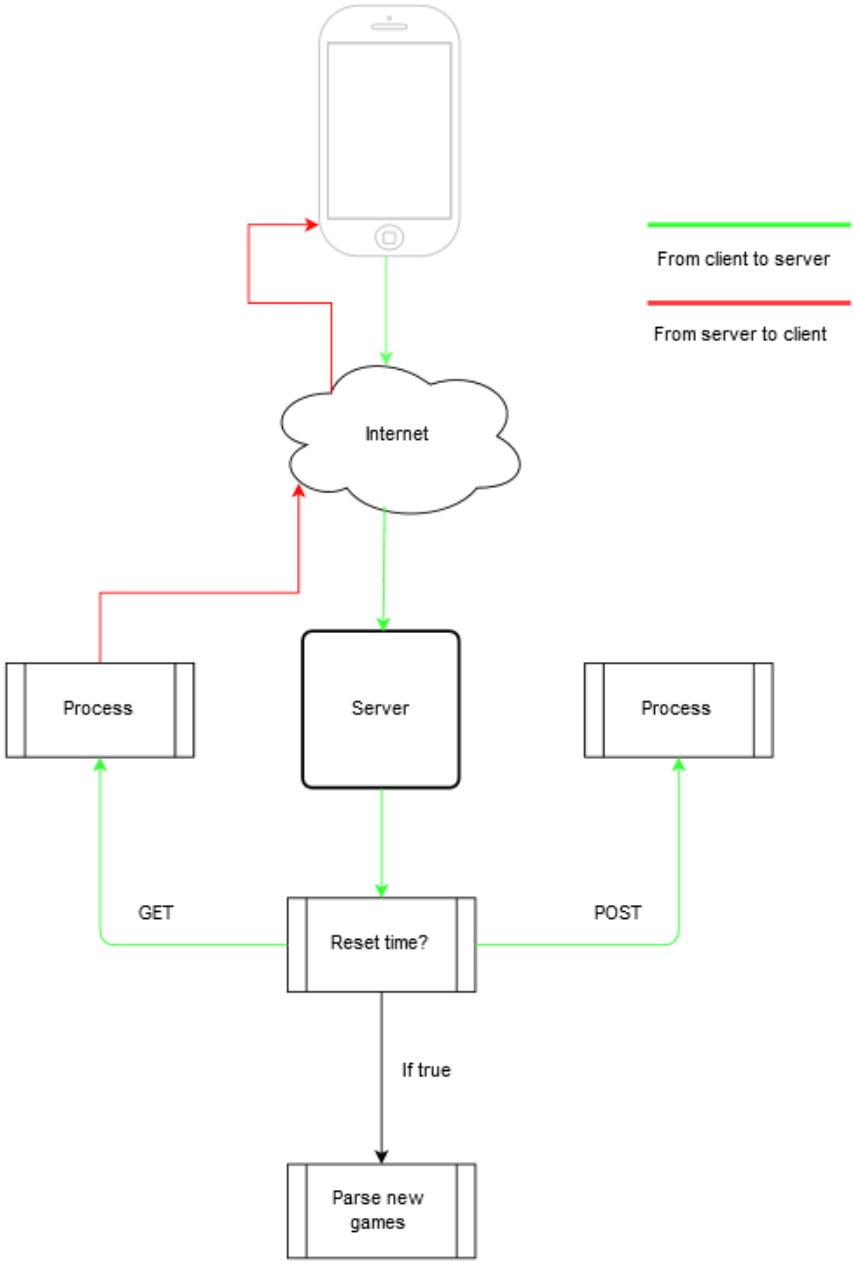

Figure A.1. *The flow of the application. If the reset time is true it continues to parse new games. If it is not true, checks if it is POST- or GET-request.*

Annex B

Requirements of the application

1. **Soccer Scores**
   This view will provide information about the soccer games that will be played for the day the user starts the application. All games will be present here. Games that are about to start, games that are live and the games that ended.
2. **Soccer Live**
   The amount of games per day will vary therefore it is suitable to make a view that will only show the games that are live. It saves the user some time not having to scroll down on finished or upcoming games.
3. **Detailed Information**
   If the user clicks on a game that has started or ended, then a view where only information about the clicked game is shown. The additional information that is shown here is details about events that might have occurred during the game such as goals, yellow or red cards.
4. **Detailed Information with rights**
   If the user has logged in and clicks on a live game, it enters the same view as Detailed Information that were mentioned above with extra functionality such as to update statistics.
5. **Login**
   This view will contain a simple login form to authorize users.