University of West Bohemia

Faculty of Applied Sciences

Department of Computer Science and Engineering

Bachelor Thesis

Mobile client for collecting sport activity statistics

Pilsen, 2015

Pavel Fidranský

Statement

I hereby declare that this bachelor thesis is completely my own work and that I used only the cited sources.

Pilsen, May 3 2015

Pavel Fidranský

Abstract

Mobile applications development can be rather demanding in cases where it is required to users on multiple platforms. The goal of this thesis was to investigate methods of mobile multiplatform development, get acquainted with and utilize Apache Cordova to implement a mobile client application for Jäsenverkko.fi portal users. The application allows its users to manage their data stored on a remote server. It automatically synchronizes them using portal's RESTful API when connected to the internet but is able to work even when offline as it contains a local storage facility. The application makes it possible to collect custom sport statistics specified remotely with a module that is easily extendable. A result of the work is a portable application tested on Android and Windows Phone platforms.

Abstrakt

Vývoj mobilních aplikací může být poměrně náročný v případech, kdy je vyžadována dostupnost aplikace na více platformách. Cílem této práce bylo prozkoumat možnosti pro vývoj multiplatformních mobilních aplikací, seznámit se s technologií Apache Cordova a použít ji k vývoji mobilního klienta pro uživatele portálu Jäsenverkko.fi. Aplikace umožňuje uživatelům spravovat jejich data uložená na vzdáleném serveru. Pokud je zařízení připojeno k internetu, aplikace data automaticky synchronizuje za použití aplikačního rozhraní portálu, je však schopna pracovat i bez připojení díky implementaci lokálního úložiště. Aplikace umožňuje ukládání vzdáleně definovaných tréninkových statistik pomocí snadno rozšiřitelných modulů. Výsledkem práce je přenositelná aplikace otestovaná na platformách Android a Windows Phone.

Contents

1 Introduction

With more and more people using smartphones nowadays, mobile applications are becoming an important part of our day-to-day lives. A lot of people shift their life online from fully-fledged computers (PCs and notebooks) to lightweight and easy to carry mobile devices.

The thing is that these devices do not often use the same operating system. It results in the need to develop and maintain an application for each platform separately or to take advantage of some multiplatform development solution.

The target of this work is to introduce tools for cross-platform mobile development and use Apache Cordova to create a prototype of a client application of the Jäsenverkko portal. The application should implement basic functionalities of the portal and allow collecting custom attributes for the sportsman's diary. The statistics module is supposed to be modular and easily extendable and the application as a whole must be usable even when a device is offline.

The first part of the thesis focuses on methods of mobile multiplatform development, front-end JavaScript and CSS frameworks, approaches that are used within them and other techniques used in the work. It outlines various options, their advantages and weaknesses, it evaluates them and after that it introduces every tool in detail.

The second part describes the implementation of the application. More challenging pieces of the work are presented in this section, the portal's programming API is introduced and finally a process of testing is outlined. The part also mentions some inconveniences of the mobile multiplatform development and possible improvements.

1.1 Jäsenverkko portal

Jäsenverkko is a Finnish web portal for complete management of various interests groups developed by a small company Yoso Oy. The name of the portal can be translated into English as *member* (jäsen) *network* (verkko) thus the portal will be referred to as Membernet in the text below.

The main purpose of the site is to gather members of a particular group and to be able to handle their day-to-day agenda such as reading notifications of societies, displaying a calendar of joined events, managing members and messaging between them or payments for services of a society.

Structure

The portal structure is quite specific but it is crucial to understand it well in order to design the application properly.

Every user is represented by an account containing his personal information linked with a unique username and a password. Users can have multiple members created. Their information may differ from the parental user; a member can be applied to societies too.

The societies can be nested thus they can form a hierarchy. They may create notifications and events for their members and the portal also integrates a payment gate to collect fees for society's services.

The society's events can be divided into two types: activities and mini-activities. The main difference is that a member must be invited to mini-activities while he can apply for activities without limitations. The societies also define custom attributes they want to collect about their members.

1.2 Implemented features

The application should make it possible for regular Membernet users to manage their content on mobile devices. The users should be able to display a calendar of joined events, read notifications from societies, search and apply for societies, edit their members' information and collect data for their sportsman's diaries. The application will not make it possible to sign up for new users, pay for services and will not include a ward control.

2 Mobile applications

The mobile devices and applications running on them require totally different point of view than it used to be in the past when programs ran mainly in personal computers. The devices are commonly equipped with a touch screen and have relatively long battery endurance. They are not very suitable for work but can be really powerful and enjoyable to consume content. The thing is that these devices lack some features that we consider as standard and almost necessary for computers.

The most important difference is the screen. It is rather small in order to keep the device compact and easy to take with. The user interface is controlled by touches of our fingers and usually there is no precise input (like mouse or hardware keyboard) available. It results in a device that has a small screen with controllers that have to be bigger so users can control them easily. These attributes lead into lack of space which may be used for the application views.

Another (equally important) issue is lower performance of a device. Obviously there are smartphones that dispose of performance comparable with some notebooks, however it is important not only to think about these high-end devices but also mid- and low-end. The performance of these devices is significantly lower but an amount of their users is great and the application must be still usable to these people.

2.1 Multiple platforms development problem

Major issue of the development for mobile devices are different platforms and operating systems operated on them. An objective is usually to engage as many users as possible – thinking of mobile applications, it means to support multiple platforms. There are basically three different ways how to achieve this:

1. native applications
2. web applications
3. hybrid applications

Every possibility listed has own positives and negatives and there is not a definitive answer to which method is the best. The choice always has to be driven by a specific task assignment and required result.

Native applications

The applications developed for each platform separately are called native. The source code is usually unique for every operating system supported. However it is not only the code and the programming language – every platform defines totally different methodology of development. Ordinarily a programmer needs to use integrated development environment (IDE) specified by the author of the system, the desktop operating system where this IDE is able to run and where the application can be compiled, linked and tested. Sometimes even the operating system requires a specific computer for its operation. Most of the mobile platforms have their applications' stores so the programmer is required to own a (usually paid for) developer account and so on.

The biggest advantage of the native applications is the fact that they allow the developer to optimize the code for a particular platform. Whether it is graphics and perfectly smooth animation, customizing the application and its connection with the operating system or full access to device sensors and programmatic interfaces, native applications are the best performing solution.

The disadvantages follow from the need of writing the code for each platform separately. The code is more or less unique for every operating system and the application must be also separately maintained. A programmer has to handle more technologies and processes and unfortunately this knowledge is not portable.

Web applications

Web applications are in fact regular web pages which require much greater interaction with their users. Programmers use the same techniques and languages that are used for the development of websites such as HTML, CSS and JavaScript. The application is saved onto a web server from which it is downloaded and launched within an internet browser.

The web applications are relatively easy to develop. Thanks to the use of the web technologies, there are plenty of materials, tutorials and articles which can be useful while creating an application. Another advantage is that a software created as a web application is naturally multiplatform. It runs within a web browser and it does not matter which browser specifically is used.

However the web applications have their drawbacks too. The most severe disadvantage is their lower performance – an application has to be operated within a web browser which becomes an extra layer needed. This necessity results in some performance loss.

The web application also does not have access to all the capabilities which an operating system provides to a native application. This problem is slowly being solved by the JavaScript part of HTML 5 where the sensors interface is mapped straight to the JavaScript. However it is still a thing of the browser and its vendor whether the API is supported or not.

At last, there is another issue that cannot be solved even in HTML 5: the user interface and source code optimization. The application is written for more platforms at once without any reliable way to customize and to optimize it for each one.

Hybrid applications

The last option considered in this thesis is to develop a hybrid application. This method makes effort to combine the best approaches of the native and web applications development. The source code can be written using different technologies and programming languages since it is wrapped into a special layer during the build which enables the application to be launched on various platforms. This layer adds a possibility to access native API of a system so there are almost no limits of use.

There are more options for a development of a hybrid application. The majority involves using the WebView component of the operating system to display a web application underneath. This web application is written in common web languages where native programming interfaces are mapped to JavaScript functions. Other solutions use various, not native programming languages for the source code and a special layer to run it on different platforms.

The hybrid applications have some disadvantages too. To be multiplatform, they use an extra layer thus they are less efficient. They usually do not have access to the native components of the user interface so their visual aspect will never fully correspond with established standards of a platform. Unlike web applications, it is possible to target the hybrid application for a particular platform so the application can be customized and may at least imitate the native user interface.

2.1.1 Summary

All the possibilities have their advantages and deficiencies. Native applications allow the programmer to optimize the code, to customize the user interface and they allow the best cooperation with the operating system. On the other hand, it is necessary to write the code and to maintain a version of the application for each platform separately.

Web applications are multiplatform but they are not very powerful when it comes to their connection with an operating system.

Hybrid applications are portable and they also offer techniques to make use of the native programming interfaces of the operating systems. However they usually do not fully support the native look and feel of a user interface.

The task of this thesis is to create a multiplatform application which will take advantage of sensors of modern mobile devices. The application should be easy to maintain and it does not require high performing graphics. Considering all the requirements, the best solution is to create a hybrid application.

2.2 Possibilities for hybrid applications development

There are several ways how to achieve the portability of a hybrid application. Every possibility solves the problem in a different manner, however the target is always the same: to share as much code as possible across various platforms. In the past, developers could use Adobe AIR but there are more technologies available nowadays.

Technological solution for this project should be truly multiplatform, i.e. support at least the three major mobile operating systems: Apple iOS, Google Android and Windows Phone. The code needs to be easily maintainable therefore as much as possible should be shared across the platforms. The technology has to enable access to devices’ sensors such as GPS, gyroscope or compass and because the resulting application will be a prototype, the technology should be free of charge.

Apache Cordova

The first of the options is Apache Cordova (also known as PhoneGap) and other tools built on top of it. Cordova is an open-source framework by Apache Software

Foundation. For the development, it uses the three major web programming languages: HTML, CSS and JavaScript.

A Cordova application is inserted into a small and lightweight web browser called WebView which differs from platform to platform. Cordova also adds another JavaScript library which enables access to native programming interfaces. It does not make it possible to utilize native user interface components.

Appcelerator Titanium

Another solution for hybrid applications development has been created by Appcelerator Inc. In this case, the code is written mostly in JSON and JavaScript and it is interpreted by a JavaScript engine of a device’s internet browser. This way, the application is enabled to use native components of the user interface.

Appcelerator Titanium is a comprehensive platform offering its own JavaScript model-view-controller (MVC) framework Alloy, integrated development environment based on Eclipse and Appcelerator also provides cloud services for the back-end.

Xamarin Platform

Completely different approach is proposed by Xamarin Inc. For the code of the application, Xamarin uses C# programming language and adds an extra layer which maps the code straight to the platform-specific programming interface. This way it is possible to reach 100 percent of shared code, high performance and a good support of features specific for a particular operating system.

Xamarin allows calling existing native code (Java, Obj-C) and it also provides solution for the native application UI called Xamarin.Forms. Its greatest disadvantage is that you must pay monthly fees unless you settle with the very limited basic edition.

2.2.1 Summary

To conclude, Apache Cordova makes use of well-known and extensively documented web technologies and it is free of charge. On the other hand, it is not possible to build the UI using native elements of an operating system with Cordova.

Appcelerator offers a fully equipped development kit and permits to create the UI using native controllers. However it is impossible to share 100 percent of the code across platforms, it is a proprietary solution and it forces the use of the Alloy framework.

Xamarin Platform is a superb and well usable solution. However it is nearly obligatory to pay for it in order to take major advantages of the whole system.

Considering all the qualities and shortages, the application will make use of the Cordova platform since it is free and it comes out of proven web technologies.

2.3 Apache Cordova

2.3.1 Cordova vs. PhoneGap

Framework until recently called PhoneGap is being developed since 2009. It was originally created by Nitobi but two years later, the whole company including the framework was acquired by Adobe and the framework itself was contributed to Apache Software Foundation (ASF) afterwards. By then, the framework was released as open-source under the name Cordova while Adobe kept the PhoneGap trademark. This allowed emergence of many other frameworks build on top of Cordova such as Intel XDK, Ionic Framework or RhoMobile Suite. Currently Adobe uses the name PhoneGap for its own distribution of Apache Cordova.

2.3.2 Principle

As said earlier, the source code of a Cordova application is written using the web programming languages HTML, CSS and JavaScript. During deployment, it is inserted into a native UI browser component commonly called WebView. Basically it is a browser window without any controllers that renders the application using a rendering engine dependent on the operating system. It is very important to know which engine is used on a particular system as it defines features available [1] to the application. The rendering engines information is presented in the Table 2.3.1.

Cordova also adds an extra JavaScript library which permits the use of plugins. These add-ons are used to access parts of an operating system (contacts, camera, GPS and other) that cannot be gained by plain JavaScript functions.

This way, the application can be developed just as any other web application and tested in regular desktop browsers. However as soon as the plugins are utilized, it is necessary to test the functionality using some Cordova-specific method.

Operating system	**WebView (engine)**	**default browser (engine)**
Android 4.0+	Android default (WebKit)	Chromium (Blink)
Android 4.4+	Chromium (Blink)	Chromium (Blink)
iOS 4+	Safari (WebKit)	Safari (WebKit + Nitro)
iOS 8+	Safari (WebKit + Nitro)	Safari (WebKit + Nitro)
Windows (Phone) 8	Internet Explorer 10 (Trident)	IE 10 (Trident)
Windows (Phone) 8.1	IE 11 (Trident)	IE 11 (Trident)

Table 2.3.1: WebView rendering engines on various operating systems

Even as the base framework, Apache Cordova makes it possible to build fully-equipped applications. Nevertheless its extended derivatives frequently have some useful value added. These services are usually not free, companies usually offer some kind of basic rate and monthly fees are charged for further functionality.

The variations often provide their own IDE, JavaScript frameworks for the application's architecture or libraries that make it easy to build the user interface. They also supply servers for the application's back-end or services that help to build the application for every platform possible in cloud. This can be useful especially for building iOS applications while using a Windows computer.

For example Adobe provides PhoneGap Build, a paid service for building the application using their servers. There is also PhoneGap Developer App available for free – it is an application that facilitates the initial testing of the application.

3 Application frameworks

Apache Cordova does not force the developer to use any particular framework therefore the source code can be written in plain JavaScript and CSS. This can be an advantage in case of developing a small application where the costs of using a framework would exceed its benefits. However as soon as the application grows bigger and the code becomes widespread, it is a good idea to use some of existing frameworks. They facilitate some basic tasks, they are better performing, maintainable and testable so a programmer can focus on the main purpose of the application.

3.1 Single Page Application

Single Page Application (SPA) is a design pattern for all applications running on the client side. As the title indicates, the full page content is loaded just once during the first run of the application. Every other step through the application provokes just a partial modification of the page while the rest remains unchanged. This is extremely important in respect to the application performance. The page's basic source files are not loaded on every view change what would be slowing the application down otherwise.

Once the application uses an extensive amount of Cordova plugins, SPA has another hidden advantage. Since the plugins may be used only after the page is fully downloaded, it is undesirable to reload the page frequently.

AJAX

Each content modification is done using an AJAX call. AJAX is a JavaScript technology which performs an HTTP request in the background and returns the data at the moment they get back from the endpoint URL. It is also possible to append custom HTTP headers and data to the request. In the result, the data can be transmitted without user even noticing it.

3.1.1 Frameworks

There are plenty of frameworks fitting the definition of SPA and each of them has its own approach and qualities. The measures will be its maturity, the size of a community of users and simplicity. Also it is unwanted to use neither a framework whose new version is released every other week nor the one that is not used by anyone but its developers. The framework should also define its own structuring of source code and follow some well-known development pattern such as MVC/P.

Backbone.js

The first out of the three compared in this thesis, Backbone.js, is the most liberate one. It is very minimalistic but flexible in its basic form – entire further functionality is supplied by third-party plugins. However with every plugin added grows complexity and predisposition to faults. Backbone.js forces the use of the MVP pattern and has a built-in support of a RESTful JSON interface.

Ember.js

Another framework, Ember.js, is more complex in its base. It is very strict to hold on to its naming conventions and architectural pattern MVC. Ember.js depends on jQuery so all of a sudden the framework grows on its size. Comparing to AngularJS, it also takes more lines of code [2].

AngularJS

The framework AngularJS is developed, maintained, supported and used by Google Inc. It follows the MVC pattern, has very large community of users and does not require any external libraries. However it can be very difficult to understand some minor parts (e.g. directives, promises) of the framework correctly.

Other

Apart from the trio mentioned above, there are a lot of other frameworks that meet the definition of the SPA structure and provide tools for building the user interface in addition. It is for example Kendo UI, a framework that is shipped together with another derivative of Cordova by Telerik company. Sencha Touch and jQuery Mobile also should not be forgotten.

A common drawback of these frameworks is the look of the user interface. The frameworks are made for building web applications where it does not matter so much if the look is unified or not absolutely precise.

3.1.2 Summary

Backbone.js is very liberate but grows on its size and complexity easily. Ember.js is quite practical but requires jQuery which is not suitable for the application's needs. AngularJS is used by a large amount of developers and supported by Google, yet it can get complicated sometimes.

Other frameworks considered in this thesis provide a possibility to build the user interface as well. However it is wanted to follow the design of a particular operating system precisely in this project which is a requirement that none of these frameworks can satisfy.

Keeping in mind the measures set earlier, AngularJS will be used for the architecture of the application since it has the largest community and is well-established.

3.2 AngularJS

AngularJS is an open-source framework intended for developing single page web applications. It is a complete client-side framework for building all parts of a CRUD application following the model-view-controller architectural pattern. AngularJS also eases the testability of an application.

3.2.1 Model-view-controller

MVC is a software architectural pattern for implementing client applications. It divides the code into three interconnected but independent and reusable parts so that any of them can be changed or replaced without the need of extensive modifications of the rest.

Model is a part that takes care of the data and information the user works with. In most cases it is connected with another module that actually stores the data (e.g. SQL database) but thanks to its independency, it is possible to switch the storage without any need to reflect the change elsewhere in the code. In the context of client-side application, the model is also the part that communicates with a server.

View is a layer that request data from the model and shows them to an end user in an understandable and interactive way.

Controller is a part that communicates the changes in the view to the model or straight back to the view.

3.2.2 Conceptual overview

AngularJS applications are composed of many independent and reusable modules. This approach allows developers to mock and test the application by parts. It is possible to load modules in any order or even in parallel. By using dependency injection, the modules are created only when required [3].

Each module has the same life cycle:

1. `module.config()` function is executed
2. `module.run()` is executed
3. directives are compiled
4. controller is appended
5. directives are linked to the controller

AngularJS is a real-world application of the MVC architecture and features multiple module types that match various parts of the pattern.

Directive

AngularJS-specific HTML tags and attributes are called directives. All DOM manipulation should be performed inside directives that are compiled and processed when AngularJS bootstraps a module. Built-in directives are easily recognizable since they are usually prefixed with the `ng` namespace [3].

Filter

Filters are used for trivial and repeated manipulations with expressions (variables and functions) inside templates and controllers.

Template

Template is an HTML file extended by directives.

View

The result of template compilation is called a view and it matches the MVC definition above.

Controller

Views in AngularJS are usually connected with a controller. The controller takes care of all the business logic connected with the view.

Service

To define reusable parts of controllers so they can be accessed from other parts of the application, there are services and factories in AngularJS. The difference between them is that service just *is* a constructor function while factory returns an object of expressions and besides, it *can* run some code when constructed.

Scope

In AngularJS, a scope commonly means something slightly different than it is understood in other programming languages. Whereas it usually stands for a context a function is executed in, in AngularJS it is a data model object that binds expressions between a controller and an associated view [3]. Every controller is originally another directive wired with the template and serves as a constructor of the scope. In the terminology of MVC, the scope is used to expose model data to views.

In the background, AngularJS uses a concept called two-way data binding. It is a mechanism that watches for data changes in the view and automatically reflects the change in the associated scope and vice versa [3].

Scopes in AngularJS can be nested – child scopes always inherit expressions from their parental scopes. AngularJS also defines global `$rootScope` that is available wherever needed using dependency injection.

Promise

For asynchronous functions, AngularJS provides a concept of promises. The goal of the service is to allow an application to run long-lasting or blocking tasks asynchronously and to be notified about their progress [4]. The task returns a promise of the data right after it is called. Once the task is done processing, it either resolves the promise while appending the gained data, or rejects the request. The task can also send notifications of a progress to the promise.

Internally, AngularJS uses promises for services such as `$http` (service for handling asynchronous HTTP requests) and `$interval` (service for repeatedly performed actions). It is also possible to define own promises and even build a promise that waits for an array of promises to be resolved [5].

3.2.3 Code structuring

Using AngularJS in the way it was designed requires strict observance of its habits and guidelines. By following them, the code becomes easier to test, modules are portable between various projects and the application can be developed by multiple programmers concurrently.

No official guideline is written by the AngularJS team but there are at least three community made style guides available on GitHub [6] [7] [8]. To split the application into parts correctly, it is useful to follow another online guide [9].

The texts agree on most things, there are some differences though. To mention some basics used in the project, a developer should avoid using the prefix `$` for any additional modules as it is a reserved AngularJS namespace. Module names should contain information about the type of the module (controller, filter etc.) and should keep the CamelCase naming convention. All modules, configurations and routes should be taken out to separate files.

Despite recommendations of the guidelines, it was agreed to sort the modules into folders not by the feature they serve for but their type.

3.3 Graphical user interface

The user interface (UI) is a crucial part of the application so it is very important to be careful while building it. It is even more difficult in this project as the application will be launched on multiple platforms. Requirements for the UI of the application are that it should be easily controllable, strictly follow the look and feel of a particular operating system and behave just the way a user expects.

3.3.1 Frameworks

Onsen UI

The framework called OnsenUI cooperates well with both Cordova and AngularJS. Unfortunately it provides a single theme that imitates the look of iOS and is not very widely used.

Ionic

Not only a user interface framework but also another derivative of Cordova is Ionic framework. It is a full-stack hybrid mobile platform dependent on AngularJS with many developers involved in it and very active development. However it does not provide separated themes for each operating system – there is always the same look no matter on a device. It also is not very stable yet.

Mobile Angular UI

Mobile Angular UI is another library build on top of AngularJS. In addition, it uses parts of Twitter Bootstrap, a well-known and proven framework often used for the development of responsive websites. It is also good-looking but still very new therefore its documentation is not very comprehensive at the moment.

Ratchet

There is also a framework originated from the authors of Twitter Bootstrap, Ratchet. It is a JavaScript and CSS library that is being developed for more than two years offering well-made themes for Android and iOS with a lot of useful components. It also has a wide community of users.

ChocolateChip-UI

The last framework compared in the thesis is ChocolateChip-UI – a library imitating the native look of the three major mobile platforms: Android, iOS and Windows Phone. Its shortages are dependency on jQuery and not very active development.

3.3.2 Summary

To conclude, there are many libraries that take care of the graphical user interface. However most of them do not provide multiple themes for various platforms which is inappropriate for the project. This shortage excludes most of the frameworks including jQuery Mobile mentioned earlier. Even though Sencha Touch includes themes for many operating systems, they are not very precise so Sencha is discarded too.

From the options left, Ratchet is a library created by the authors of Twitter Bootstrap and has a number of developers using it. It comes with themes for Android and iOS.

ChocolateChip-UI is not as used as Ratchet and depends on jQuery but it adds Windows Phone theme in exchange.

In the end, Ratchet will serve as the user interface framework of the application as it has more users and it is more trustworthy.

4 Storage

Since the application needs to be usable even without the internet connection, it is necessary to store data locally. Mobile operating systems usually have their native methods of storing data but since the application is actually launched in a web browser, none of these possibilities is available.

The JavaScript part of HTML 5 has fortunately added few options for the client-side storage. During the specification phase of HTML 5, two concurrent storage methods were set, neither of them is widely implemented though. Because the application is supposed to be portable between multiple mobile platforms, the storage must be also available on each of them which is often a problem.

WebSQL

The first client-side storage is WebSQL. It is an old-fashioned storage that uses SQL language to manipulate the data. It was meant to be a standardized wrapper around an actual storage that could had been implemented in various ways depending on a browser's vendor choice. However the implementation was always the same, with a SQLite database in the background. The intent was unfulfilled, W3C decided to terminate further development and the specification was labelled deprecated on November 18, 2010 [10].

Although WebSQL is not maintained for more than four years, it is still implemented [11] in many (even mobile) browsers except for Mozilla Firefox and Internet Explorer. However as the specification is no longer maintained, it can be in fact removed in any future versions of a browser. Also there is not much documentation available.

With regard to the project, WebSQL is available on all target mobile platforms using a Cordova plugin [12] that maps the WebSQL API to a platform's native SQLite storage. Microsoft's platforms are supported as well since the MSOpenTech team created a polyfill [13] for it.

IndexedDB

Another local storage option is IndexedDB. It is a NoSQL database so it does not make use of SQL language. It works rather with objects just as they are defined in JavaScript.

Therefore, it is not needed to convert the objects to a flat structure like it is with any SQL database. This approach is very practical for JavaScript applications.

Nonetheless the support of IndexedDB has been added just recently [14] into Android (version 4.4+) and iOS (8.0+), moreover Safari implementation is still seriously faulty [15]. For older versions of these operating systems, it would be necessary to use a polyfill such as the one developed by the MSOpenTech team [16]. However, it is not proven to be fully functional yet.

Web Storage

Last possibility that has been considered is WebStorage (sometimes localStorage). It is a method of storing data locally in form of key-value pairs. It works like regular cookies with the difference that the storage size is not limited and the data are not contained in HTTP headers. Web Storage is fully supported by most (including mobile) browsers.

4.1 Summary

It is really not easy to conclude as every storage listed is kind of compromise. The project requires a robust multiplatform solution which does not exist at the moment.

Web Storage has been rejected right away because it provides only simple key-value storage which is insufficient for the project.

WebSQL is a regular SQL database supported on both iOS and Android but there are some widely used plugins available that make it usable almost everywhere. Still, WebSQL is not being developed for years, it is not supported natively on Windows (Phone) platform and it may be possibly dropped from other platforms in the future.

IndexedDB is a NoSQL database so it can store JavaScript objects as they are. Unfortunately it is not supported in older versions of the most used mobile operating systems so the application would have to use a walkaround.

In the end, WebSQL has been selected to serve as the local data storage because it is currently available at most of mobile devices. However it is not an ideal solution and as soon as the older devices die out, IndexedDB is a better choice.

5 Other tools

While developing using Cordova, not only JavaScript and CSS frameworks are good to use. Some other tools are required in order to build the application, the rest are utilities making the work easier and more productive.

5.1 Cordova CLI

The abbreviation CLI stands for command line interface and basically it is the main method of managing a Cordova project. It is used not only for the final deployment of an application but the whole workflow connected with it: adding and removing supported platforms, managing plugins and running the application on a real device or within emulators.

The CLI depends on Node.js and it is recommended to use node package manager (npm) for its installation. Then, Git is required for Cordova to download its platforms and plugins from remote repositories.

Alternatively Adobe developed PhoneGap Desktop App. It is an application that does not rely on the command line and works within a window with a graphical user interface. However it is still an early version so its possibilities cannot compare to the Cordova CLI at the moment.

5.2 Bower

Another not required but very practical tool is called Bower. It is a package manager of front-end frameworks that makes it easy to download them from a public catalogue and maintain their versions and dependencies.

The greatest advantage of Bower is that it stores a list of packages in a manifest text file. While saving the source codes onto some shared storage (typically Git), it is necessary to only append the manifest file instead of all the assets. This is extremely useful while working in a team because Bower allows keeping the dependencies up-to-date using a single command.

5.3 Google Chrome

Although most of its users probably know Chrome just as the internet browser created by Google, it is a great tool for the Cordova development as well. Developer tools similar to the ones in Chrome are contained in almost every other browser but there are some features that are unique and very useful for the mobile development.

The most important feature is that it allows the developer to view the application in the same way it will be displayed on a device’s screen. Chrome gives the option to pick from a list of more than twenty devices or to define a new one. It also solves the problem of high resolution displays and it is able to emulate the touch events. Then, the developer tools can limit the network speed so the application loads like in some village where only EDGE is available. Finally, there is a possibility to emulate a GPS signal.

Google Chrome also integrates JavaScript debugger and profiler, DOM viewer and storage browser. It is also used as the browser for a Cordova browser platform.

6 Data model

The model comes out of the structure of the portal and JSON objects returned from its API. The model is a bit complicated due to the use of a relational database – this way, the objects must be cut up to pieces that fit a single level structure of database tables.

The data model is defined in a separated `db.js` file as an AngularJS constant so it is possible to inject it where the tables' specification is required. Nonetheless it is usually used only when the application is launched for the very first time.

6.1 Specification

The specification of the data model can be easily represented by an ER diagram. All the entities of the application have their database tables defined. Datatype of their columns is either integer or string due to limitations [17] of SQLite database that is used underneath. This means that even dates, timestamps or boolean values must be saved as `TEXT` or `INT`.

The general part of the diagram [Image 6.1.1] contains all the entities except the statistics. In addition, there are tables for storing member's applications for societies and activities. The reason why the applications are separated is to ease the synchronization process.

The second part of the ER diagram [Image 6.1.2] introduces tables for collecting the statistics. The statistics are specified by their type and their input fields. There is also an extra table for saving attributes collected by a particular user.

The same attributes can be observed by more societies of a particular user while the user is still the same and he should not be forced to fill in the same values multiple times. Therefore the values are saved not with a member's but user's identifier and not appending a society's ID but rather the statistics key (e.g. `weight`, `running_tracker`).

7 Application architecture

The source code of the application is divided into files according to the functionality they supply. The project follows the guidelines outlined earlier in order to keep the code readable and expandable for other developers.

7.1 Partials

Services

In AngularJS, services are used for repetitive operations that can be called from various parts of the application.

Database factory

A separated factory was created to manipulate the data stored in the database. It also includes functions to manage the database in its entirety.

The `database.init()` function starts up the database and it is invoked on every application launch. It uses various Cordova plugins for different operating systems to define a `database.db` handler. Except for this, the function may also create all the database tables according to the `databaseConfig` constant if they do not already exist. The creation is performed in a single database transaction.

Then the database factory includes the `query` function. It accepts an SQL query and additional bindings as arguments, executes the query inside a transaction and returns a promise. To manipulate the data, the factory features additional functions that return either a single row or all rows returned from the query.

Entities factories

Every entity that is specified in the database has its factory too. These factories are very similar in fact: they always define several functions to return the data by an identifier, within a date period or since it was created or last updated.

After that, functions for handling remote data that are downloaded from a remote server during synchronization are included in order to convert and to save it to the local database. Finally the factories include functions that insert or update the data when a user performs some action within a view.

Synchronization factory and custom HTTP service

Last two factories that were implemented are used to synchronize the data between the local database and a remote server. The synchronization process is performed using AJAX calls and HTTP requests.

AngularJS is shipped with a built-in `$http` service that is used for this purpose, however it showed up to be a good idea to create own wrapper upon the service. Both services are further described in the chapter 7.4 dedicated to the synchronization.

Filters

AngularJS filters are used to adjust the data either in a view or in a controller. Angular comes with many useful filters (e.g. number, currency and date formatting) by default but it was useful to add some custom filters as well.

birthdayToAge

In views that display detailed information of a user or a member, `birthdayToAge` filter is helpful. It accepts a JavaScript Date object that indicates the date of birth of the persona and transforms it to its age in years.

dateFormat

The application shows lists of events in calendars. Apart from other information, a single event is always described by dates and times of its beginning and end. In order to define the format on one place only, there is the `dateFormat` filter that extends the filter shipped with AngularJS. Additionally the custom filter returns only the time in case where the dates of the event's beginning and the end are the same.

msToString

Lastly the project includes `msToString` filter that is used to convert a duration entered in milliseconds to a human readable string in hours, minutes and seconds.

Global expressions

AngularJS provides multiple options to store global expressions that are used in diverse parts of the application such as a user identifier and a token or the logout function.

Intuitively, the root scope is a place where the global variables and functions would fit the best. Nonetheless every guide line prevents developers to use the root scope this way as it could easily become chaotic. It is recommended to only store a few variables in the root scope.

Another possibility is to create an extra service and inject it wherever it is needed. However in this project, the service would have to be inserted in most of the controllers. Due to this characteristic, the global expressions are kept in a main controller whose expressions are inherited in all nested controllers. The main controller contains functions for navigating between views, searching and it operates a user logging out.

7.2 Bootstrapping the application

Index

The application is bootstrapped just like any other website from the `index.html` file. All frameworks files, JavaScript libraries and stylesheets are loaded here. Except for loading the external assets, the index also defines a wrapper for a `ngView` directive whose content will change according to routing rules. In addition, the wrapper has its own controller defined that serves as a global scope thus its expressions are accessible from every nested controller.

The index also fulfils another crucial task: whole AngularJS magic is started from here. There is a JavaScript listener that waits for the `deviceready` event to be triggered. As soon as it is fired, a callback function starts up the whole AngularJS application. Launching it this way is extremely important because Cordova plugins cannot be used before the device states it is ready. If the application was started up immediately after the page is loaded, it would not be possible to use any of the Cordova plugins including the database.

After the application is bootstrapped, AngularJS modules are instantiated. In the main module of the application, the `run` function is used to initialize the database using the `database` factory.

Router

To switch between individual views of an application, AngularJS uses a router to wire together controllers, view templates and URL locations. The router is used together with the `ngView` wrapper and it is responsible for displaying the view template matching the current URL. AngularJS automatically handles the history of visited locations so it makes it possible to navigate back and forward without any effort.

The router also allows passing variables in the URL – they are then accessible from the associated controller and can be used to select appropriate data. This is used extensively in this project for identifying the entities and also for carrying the searching query.

Views and controllers

Every view must have a controller assigned. The task of a controller is to feed convenient data to the view and watch for changes and events being triggered in the view. The controller is an appropriate place to manipulate the data before they get to the view, however the controller should never be used to manipulate DOM.

In this project, controllers usually depend on several services and use them to actually gain the data. After that, they construct an object of the data that are used in the associated view and pass it to the view using the `$scope` object. Furthermore, some controllers define functions that may be called from within the view to submit a form or perform changes to the data.

It is usual to set one view per controller but it is definitely possible to add multiple views to a single controller. The application uses this option to show and edit information of a user or a member profile.

7.3 Statistics

One of the tasks of the application is that it enables a user to collect the attributes required by applied societies. The statistics are specified on the server so the application must be able to construct views for their collection dynamically. Also it should be easy to add support for new types of the statistics. It means that the code must be modular but simple to extend.

7.3.1 Design

The statistics are actually independent plugins that are defined by inputs they require a user to fill in. The fields are always the same, it only depends on the statistics how the data will be processed later. It means that the inputs should be implemented so that they may be reused according to the statistics specifications. Nonetheless the statistics still need to have a view created that will display the data in a suitable form.

7.3.2 Implementation

Collecting attributes

In this project, several field types were implemented. AngularJS provides a great possibility to create reusable components of the user interface using modules called directives. They allow extending any HTML element with a custom template and even a controller while still permitting access to the scope of a parental view. This is extremely useful for some more complicated inputs such as the GPS tracker.

When constructing the statistics collecting view, AngularJS simply goes through the statistics specification and adds the directives according to their field type to the template. Each directive must be then compiled so it actually gains access to the scope.

In a controller of the view, there is a function `$scope.stat_save()` that can be called anytime from within the view. Its task is to collect the data from the fields in the view and save them to the local database. It is very simple for inputs that return just a single value but for example the GPS tracker may return much more of them. Due to this complexity, the function broadcasts the `statSave` event. Once the controllers of the directives catch up this event, they append their data to a global array. The array is then saved to the database with a current timestamp.

Displaying records

The timestamp is then used to synchronize the data to a server but also to display them correctly. Firstly, the view responsible for displaying the records selects the data from the local database by the statistics specification key. Then it groups them according to the timestamp so the values recorded at the same time are displayed together. Finally the view includes a template defined by the specification key which displays the data in a nice way.

7.3.3 Result

The statistics architecture allows societies to specify the attributes they want to collect remotely. The specification can either use the keys that are supported by the application but it may also define completely new statistics. In this case, the application can collect the data and synchronize them to the server but it is not able to display them until the AngularJS template is added to the source code.

7.4 Synchronization

The Membernet client application is supposed to work without an internet connection and synchronize its data against a remote server when online. In fact, it is really challenging to design the synchronization process so it is fast, reliable and not consuming all user's data rate.

The synchronization should be performed when a user is logged in and a device is online. It should be executed repeatedly in some interval and keep track of the last synchronization so only new or changed items are transmitted. Once the device goes offline, the synchronization interval should be resumed and as soon as the device is online again, it needs to be restarted immediately.

It is also important to choose amount of data that will be synchronized from the server to the application. In production, the server may contain data of many societies and a large number of users. It is useless for the application to store all the data since most of them never get displayed. However all the data of the current user should be available even when offline.

7.4.1 Portal's programming interface

The member network provides a public application programming interface (API) which will be used for the synchronization between the application and the server. The interface implements RESTful architecture using HTTP protocol to access and modify the data stored on a remote server. The data from the server are returned in JSON format.

The API may be accessed only with a valid user token which is returned after successful authentication using user's credentials. The token is unique for each user and it is regenerated on every password change.

Most of the API's endpoints support an optional parameter `since` that serves to return only items that were added, updated or deleted after the time defined by the parameter.

Representational state transfer

Software architecture style known mostly by its acronym REST is one of the methods of manipulation with remote data within a local application. It is used as an easy and unified way to access the server's resources that are represented by unique URLs [18]. In most cases, REST uses HTTP protocol – especially four of its methods that match specification of CRUD as presented in the Table 7.4.1.

The letter C of the CRUD acronym stands for data creation and is performed using the HTTP POST method in RESTful APIs. The R means data retrieval which is done by the GET method. When the data needs to be updated, REST uses the PUT method of the HTTP protocol for this purpose. At last, there is method DELETE that serves for data removal.

CRUD operation	HTTP request method	WebSQL query
Create	POST or PUT	`INSERT`
Retrieve	GET	`SELECT`
Update	PUT	`INSERT OR REPLACE`
Delete	DELETE	`DELETE`

Table 7.4.1: CRUD operations and their mapping to HTTP methods and WebSQL queries

JavaScript Object Notation

JSON is an open source format of saving data objects in a form of a human readable string. It is a language independent file format and is often used as an alternative to XML.

Since JSON was derived from JavaScript, it can be easily converted to native objects or arrays that can be manipulated immediately.

Prototype

At the time of programming the synchronization part of the application, the portal's API was not finished yet, only its prototype was available using the Apiary service[1].

Apiary is an online tool that makes it possible to create a prototype of the RESTful programming interface without actually writing any code. It is useful to find the needs of developers before implementing the API on the top of an application.

[1] Membernet API prototype; http://docs.mnapi.apiary.io

The documentation is written using intuitive Markdown syntax and is automatically transformed into a server mock with correct HTTP methods and addresses as you write. Apiary allows accessing the prototype on a public URL hosted on their servers but does not let you to make any changes to the data since it is still just a static prototype.

7.4.2 Implementation

The first time the synchronization is performed is when the user logs in. By that time, the application downloads all the data that are somehow connected to the user. After that, the sync is executed repeatedly with a timestamp of the last synchronization and a user's token appended to each request. To respond properly to the connection changes, there are JavaScript event listeners in the main controller that are triggered when the internet connection is either lost or gets available. Callback functions of these listeners then stop or restart the 30 seconds interval of the synchronization process.

The application downloads data of all societies so it is possible to search through them even when offline. Then, it gains member profiles, (mini)activities, notifications and statistics specifications and records of the societies that the current user is applied to. For the activities, it also gets its participants.

Synchronization factory

All the synchronization process is separated into a factory. Firstly, there are functions `sync.start()` and `sync.stop()`. The first is used to start the interval in which the data are synchronized periodically. The second is used to cancel the interval. These functions are used as the callbacks of the online/offline events listeners.

Except for that, functions `sync.toServer()` and `sync.fromServer()`are defined there. Both functions return promises so they can be chained and give a notification once finished. It is useful to split the process into parts so only one way transmission can be executed. The transmission from the server only is executed when a user logs in, the two-way synchronization is then performed in the *to-from* order in the interval.

HTTP requests wrapper factory

The data transmission is performed using AJAX HTTP requests. AngularJS contains a service for that but because the API requires the user's token appended to each request, it is a good idea to extend the default service.

Besides adding the token to each request, the wrapper is also used to react adequately to the server rejecting a request. This situation can be caused by a user changing his password using the web version of the portal. The change causes regeneration of the token while the old one still remains stored in the application.

The URL of the API is defined there too therefore it needs to be changed only at one place as soon as the official API is released.

The factory defines functions that copy the methods of the HTTP protocol and that are used in the application: `http.get()`, `http.post()` and `http.put()`. All the functions return promises just like the original AngularJS `$http` service does.

Timing

The synchronization part of the application makes use of the API's possibility to enter the time since when the updated items should be returned. This time is defined as an UNIX timestamp and it is always the time of the last synchronization *from* the server.

The timestamp is stored in Web Storage and because all the dates and times on the server are based in UTC, the timestamp is as well. This results in the need to convert the times in the application so they fit the local time zone. This is the place where the custom `dateFormat` filter is helpful.

8 User interface implementation

A user interface is the only part of the application that is really visible to a user. Since the application is multiplatform and it is wanted to follow the look of a particular operating system, it is useless to design the user interface for an exact use case as it needs to vary from platform to platform.

Various operating system use different components of the user interface so it is not simple to design it so it fits in and behaves expectably on any device. The main target is to find an intersection of features that are available on every platform so the interface may be written and maintained at only one place and customized by appending various stylesheets during the build.

It was decided earlier that the application will take advantage of the Ratchet framework. It eases most of the tasks that would otherwise have to be solved separately. These are for example actual styling of elements so that they imitate the native controllers accurately and can be combined in various ways, or designing icons that are easily understandable and good looking.

Nonetheless there are some tasks left to the programmer. Ratchet provides UI elements but they need to be composed together so the views of the application make sense. Also some functions have to be defined to react to user interactions.

8.1 Navigation

The application that is being developed in this thesis has many views among which a user has to go through. Navigation is the most important user interaction and some problems had to be solved while working on this part.

8.1.1 Views map

First of all, a map of the views and their relations was created as seen in the Image 8.1.3. The diagram introduces relations between all the views of the application.

Arrows between single items signify that a view refers to a following view through an actual hypertext link. The backwards move can be always performed using a device’s back button but in some cases, the arrow is bidirectional which means there exists a way to go back within the view. Cycles may be discovered in some parts of the map.

Several views are grouped into boxes titled Home and Society as can be seen on a simplified map in the Image 8.1.1. Source codes of the views are still separated but the views are connected using a secondary header bar with tabs. A user can navigate between these views by swiping his finger from one side of the display to the other.

Finally the map contains information about views' levels. It defines hierarchical relations between the views which are necessary for an Up button.

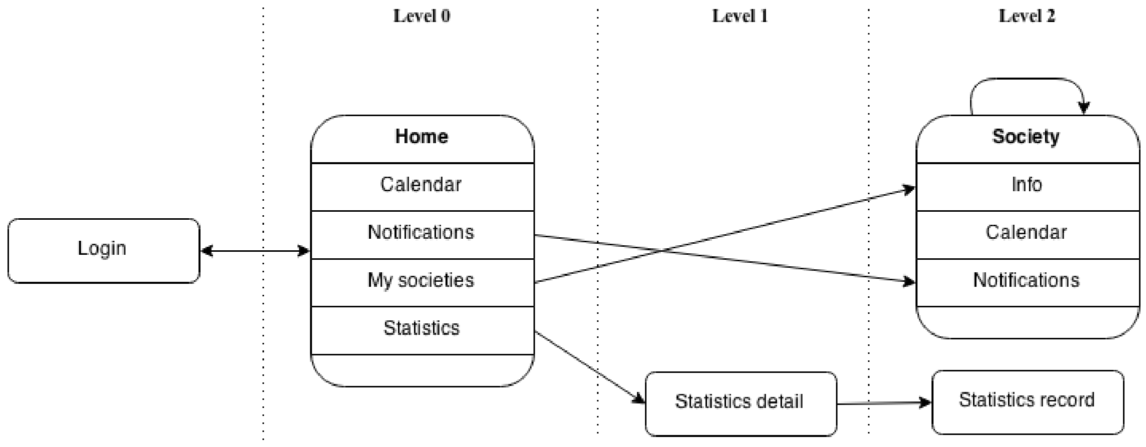


Image 8.1.1: a simplified schema of views, their relations and levels

Back and up button

On most of the devices and their platforms, there is only one method of going back from one view to another. The change can be triggered in multiple ways (e.g. swiping with finger, tapping a UI element), a button can be placed in the header bar of a view or in the bottom but it always takes the user one step back in the history of passed views. This method will be referred to as the *Back* button. However since Android version 3.0 [19], there is another controller called *Up* button thus it is necessary to distinguish among the behaviour of both these buttons.

The Up button is always shown on the top of the view within the header bar. It is used to navigate rather between hierarchical **levels** of the application than the history. It may sometimes lead to the same location as the back button but it is not a rule.

In the map of views in the Image 8.1.3, each view has a level assigned. On pressing the Up button, a view with a preceding level is chosen to be displayed. In the source code, the level is set in a controller of the view. The Up function is defined in the main controller and there is also an array of pages visited on each level. When the function is called, it decrements the level of a controller as long as values of the array are empty. Then it programmatically changes the view and deletes values of the array with indexes higher than the current level.

Bars

Navigation bars are used in the whole application to show titles and display links to related views. The views grouped under the Home title include a search icon and a hamburger menu icon in the header bar. In the views that serve to show and edit information of a persona and to view a single statistics record, there is another bar added in the bottom. Its only purpose is to allow a user to quickly return to the main Calendar view.

The views that are clustered in the map [Image 8.1.3] have also a secondary header bar which shows icons and captions of the other views in the group. This secondary bar is displayed at the bottom on iOS and at the top on Android like it is recommended by the documentation. However Ratchet contained a bug which resulted to the bar overlapping the top of the content on Android. Therefore I created a pull request [20] with a fix which was later merged to the master repository.

Tabs swiping

Besides tapping, the grouped views can be also accessed by a user swiping a finger. The implementation is similar to the Up button. Controllers of the views set locations of their neighbouring views while a callback function that performs the change of a view is defined in the main controller and can be called from within any view.

AngularJS has a built-in support of swiping events – an event is invoked at the moment a user swipes his finger. After that it calls the callback function which changes the view according to the preset locations.

The resulting interface can be seen in screenshots Image 8.1.4 and Image 8.1.5 that capture the views where the tabs and swiping gestures are used.

Search

The application also includes a search input that is used to look for new societies to apply for. The input is accessible from the views of the Home group.

It is currently displayed according to customs of Android to overlay the header bar. After tapping the search icon in the header bar, the bar is replaced by a search input. When a user submits the form, the location is changed, a separated view and a controller are loaded while the query is passed in the `$routeParams` parameters of the router.

8.2 Notifications

The operating systems provide many possibilities to inform a user about a result of some action. Fortunately the options are very similar on most of the operating systems so it is not needed to reinvent a wheel.

With Cordova, the notifications are invoked using plugins so they are native on each platform. However the use of Cordova plugins always brings worries about their support on various operating systems. The good thing is that they are usually functional on Android and iOS at least.

Alert

The first method is to display an alert. It is a basic way of notifying a user in JavaScript and it can be called in a Cordova application as well. However it is not customized by an operating system so an additional plugin is used in this project.

Alerts are used when it is required to confirm a message or input a value by a user. The plugin adds possibilities to show an alert with a cancel button or to play a beep sound and it does it in the style of a particular platform. The application uses alerts to warn of experimental GPS tracking features.

Spinner

Sometimes the application performs a long-lasting operation where it is good to inform a user so he knows that something is going on. For this purpose, operating systems use spinners.

In this project, a spinner is shown after a user successfully logs in for the first time. At that moment, the application needs to download one's personal data from the server which takes a while.

Toast

Toasts are basically non-blocking messages that disappear by themselves after some time. They are usually used to inform a user about some action that does not demand any interaction.

The application displays a toast when a user saves a persona details or applies for a society or an activity. Toasts are also used in the login view to notify about wrong entered credentials and other possible results.

9 Testing

The result of the development process is a multiplatform mobile application programmed in technologies usually being used for development of web applications. Therefore, it can be tested like a web application but there are some Cordova-specific methods too.

9.1 Possibilities

Since the application is a web application in fact, in can be tested just like any other of that type in a desktop browser. However it is also a mobile application thus it needs to be tested on a device which supports all the attributes of mobile devices. After that, it is a Cordova application where some functionalities added by plugins cannot be tested otherwise than on a real mobile operating system. Finally the application is multiplatform so it should be preferably checked on all the platforms that are supported and as many devices as possible.

Testing on a real device is just a tip of an iceberg and there are plenty of other options that are more suitable for various parts of the process during the development.

Apache server and a desktop browser

The first method is to use an Apache server to perform HTTP requests in the background and load the application in a generic desktop browser. This way, the testing is very fast because the application does not have to be rebuilt on every source code change. However it does not allow using any Cordova features (including plugins) because the Cordova specific JavaScript objects get appended during the build.

PhoneGap Desktop and Developer Apps

One of the value added by Adobe's variant of Cordova are development applications both for desktop and mobile devices. The PhoneGap Desktop App is used to start a local web server to which the Developer App installed on a mobile device can connect. Then it loads all the source files and displays it on the device's screen in a WebView just like Cordova does it in production.

Unfortunately this solution does not support external Cordova plugins yet and there are no developer tools available so it is not very powerful at the moment.

Cordova browser platform

In mid-October 2014, the Cordova team released a major version of the CLI [21] introducing (except for other features) a new browser platform. It is used like all other platforms so it has to be managed by the CLI but it uses Google Chrome to launch an application. It is intended purely for the testing process and it does a great service. The application must be built every time the code is changed but it only takes a few seconds.

Unlike the first possibility, it does not require an Apache server and supports the Cordova plugins in addition. By using the browser platform, it is not necessary to invent all the walkarounds because it is already done. Not all the plugins already support the platform but it is totally better than none.

Emulator and real devices

Cordova CLI also provides methods of deployment straight to an emulator or to a real device. There is not much of a reason to utilize emulators if you own a suitable device since the emulators are usually very slow and do not add any advantages in exchange.

A real device is good to test the application by actually using it in a real life. Additionally Android of version 4.4 and higher supports the so-called Remote Debugging [22]. This feature allows live debugging of the WebView content remotely from the desktop Chrome browser.

9.2 Process

At the beginning of this project, the application could be tested like a pure web application. At that time, before the Cordova plugins came into play, the Apache server option was used a lot. By opening the application in a Webkit/Blink-based browser [Table 2.3.1], it looks the same as on any Android device. In addition, there are the very useful developer tools in Google Chrome. The PhoneGap supportive applications were also used several times to see how the application feels on a real device.

As soon as the application started to require using of the plugins, the browser platform was used extensively. It behaves just like any Cordova platform but it keeps all advantages of the Chrome developer tools and the build is still reasonably fast.

Real devices were used to test the application at the very end of the project mostly to check up the synchronization process and the GPS tracker. However it showed up some differences between versions of Android as well as drawbacks of the Windows platform.

Server implementation

In order to test the application properly and help to design the API so it was actually usable, a sample implementation of the portal's server was programmed as a part of the thesis. The server is supposed to follow the programming interface exactly as it is specified on Apiary so it will be necessary to only replace the address of the endpoint as soon as the Membernet's official API is available.

The server part is hosted on a server accessible publicly[2] at the moment. It is written and it can be launched using PHP 5.2.0 or higher. The script uses a NoSQL MongoDB database as the storage thus the server must have a MongoDB PHP driver installed. Besides, it uses the `mod_rewrite` module of an Apache server to pass all incoming requests to the main `mongo-driver.php` script. These requests are then switched according to the input URL and parameters, data are manipulated and finally the script returns a well-formatted response and a matching HTTP status code.

The data are stored on an external hosting called MongoLab. It is a service that provides MongoDB hosting with on-line management tools, monitoring, backups etc. MongoLab offers a limited basic plan with 500 MB of the storage capacity and a server hosted in the USA for free which is enough for the needs of the prototype.

Android WebView changes

Since Android version 4.4, the WebView component of the operating system is backed by Chromium instead of own rendering engine as seen in the Table 2.3.1. This leads to huge performance increase [23] and better support of modern web technologies, however the older versions of Android must be still taken into account. In this project, some legacy CSS code had to be added.

[2] Membernet API sample implementation; http://devtest.yoso.fi/mnstub/

Windows platform issues

Even though it was not a task of this thesis, the application was tested on Windows Phone too to see how multiplatform it is in reality. Cordova's Windows Universal platform is still very new [24] therefore some obstacles had to be faced obviously.

First of all, AngularJS was not loaded due to Windows Runtime security restrictions so an extra library [25] had to be added. The platform is not supported by the chosen database plugin [12] thus another one [13] had to be used. The back button trigger function had to be added manually[3] because it is not tied with the WinJS framework yet. The in-application links did not work until they were whitelisted[4]. Since Ratchet lacks styles for Windows Phone, the look of Android was kept but still the menu cannot be opened and the swiping does not work. Finally, the GPS tracker does not show a map and the application crashes on resuming while the tracker is enabled.

9.3 Summary

To sum up, the application was tested on Android (versions 4.3 and 4.4) where it works without problems. It is also able to run on Windows Phone but it needs to be considered as nothing but a prototype.

[3] Cordova problems with windows phone 8; http://stackoverflow.com/a/28820910
[4] AngularJS and Windows 8 route error; http://stackoverflow.com/a/19683841

10 Results

10.1 DevStack evaluation

Since the beginning of this project, a lot of changes have happened: new versions of the tools were released, some frameworks came into play while others lost their drive.

Cordova has gained the brand-new and very useful browser platform. Adobe released the testing applications that are also getting to be useful. New versions of plugins have brought new possibilities but more importantly they have fixed some bugs. However most of the plugins are maintained by third-party developers who often do not manage to react fast enough thus only Android and iOS are often fully supported.

Facebook released duo of new JavaScript front-end frameworks, Flux and React, which provide a different and perhaps better approach than AngularJS. In response, the AngularJS team announced that the new major version of the framework will be completely different from the current. It is not a problem now but it is definitely not good news to the future.

From version 5.0, Android contains a redesigned user interface styling called Material. Because no Cordova application can rely on native components, it is a thing of Ratchet to integrate changes in this work. Unfortunately Ratchet seems to stop its development as there was not a single bigger update since the beginning of the project. On the other hand, Ionic has become a well-usable solution as its community has been growing up.

10.1.1 Summary

Due to enormously fast development and frequent changes, developing a mobile application using Cordova does not seem like a very good idea for professional purposes. There are many dependencies where each of them must support the target platforms and this is often a problem. In addition, there are not many quality sources of information.

10.2 Possible improvements

Even though the application is usable in its present form, some changes could be made to make it better.

The synchronization process might be improved to reflect a current type of the internet connection – the application could download the complete data more often on Wi-Fi while it could just update the most important information once in a while on a mobile network. Also it would be nice to permit a user to manually launch the process. There is also a known issue of not synchronizing a participant's state due to lack of an appropriate server endpoint.

After that, the statistics needs to be completed. It would be useful to display the collected data and compare them mutually within graphs to monitor user's performance and progress in time.

Last but not least, the user interface might be redesigned. It does not fit the last versions of the operating systems and it would also have to be tested in order to match real users' needs. The application completely lacks some branding too.

11 Conclusion

The aim of the thesis was to get acquainted with tools for multiplatform development of mobile applications with a focus on Apache Cordova and use it to implement a client application of the Jäsenverkko portal. The application should allow its users to manage their Membernet content and collect data for their sportsman's diaries. It was supposed to work even without the internet connection.

The first part of the work focuses on the tools that are used in the application and each of them was introduced in detail. After that, the practical use of these tools had taken a turn. The data model of the application's storage was designed, the architecture was described and the portal's API was presented. Next part was dedicated to the user interface with a focus on navigation and notifications. Finally the application was ready for testing. Several methods of testing of a Cordova application were mentioned and used during the testing process. Finally some problems of the Cordova multiplatform development were described as well as possible future improvements.

For me, the work has brought a lot of new knowledge of technologies I was not aware of before and which I can use again in my future projects. I had already known the web technologies used by Cordova but it was really exciting to use them for a totally different purpose. I have also started to focus more on the design than real programming, adjusted my workflow and contributed to several open source projects. Sometimes it was annoying to reveal and fix bugs of the Cordova plugins but I believe that these issues will be less frequent in the future.

The resulting application fulfils the assignment completely but it is still a prototype which only works properly on Android. Even though the tools often have their problems, it has been proven that development of a non-trivial multiplatform application using Apache Cordova is possible and it definitely has its advantages.

List of abbreviations

Adobe AIR	Adobe Integrated Runtime
ASF	Apache Software Foundation
HTML	hypertext markup language
XML	extensible markup language
CSS	cascading style sheets
IDE	integrated development environment
CLI	command line interface
UI	user interface
API	application programming interface
MVC/P	model-view-controller/presenter
JSON	JavaScript Object Notation
REST	representational state transfer
CRUD	create-retrieve-update-delete
SQL	structured query language
NoSQL	not only SQL
ER diagram	entity-relationship diagram
DOM	document object model
HTTP	hypertext transfer protocol
AJAX	asynchronous JavaScript and XML
SPA	single page application
W3C	World Wide Web Consortium

npm	node package manager
IE	Internet Explorer
GPS	global positioning system
EDGE	Enhanced Data rates for GSM Evolution
URL	uniform resource locator
UTC	Coordinated Universal Time

References

[1] **Firtman, Maximiliano.** HTML5 compatibility on mobile and tablet browsers with testing on real devices. *MobileHTML5.* [Online] http://mobilehtml5.org.

[2] **Sofer, Dean.** What Should Beginners Choose: AngularJS, Ember.js, or Backbone.js? *Codementor.* [Online] September 10, 2014. https://www.codementor.io/angularjs/tutorial/beginners-angular-ember-backbone.

[3] **Green, Brad and Seshadri, Shyam.** *AngularJS.* s.l. : O'Reilly Media, 2013. p. 196. ISBN 9781449344849.

[4] **Rimple, Ken.** AngularJS Corner – Using promises and $q to handle asynchronous calls. [Online] June 7, 2014. http://chariotsolutions.com/blog/post/angularjs-corner-using-promises-q-handle-asynchronous-calls/.

[5] **AngularJS.** API: $q. [Online] https://docs.angularjs.org/api/ng/service/$q.

[6] **Papa, John.** Angular Style Guide. [Online] https://github.com/johnpapa/angular-styleguide.

[7] **Gechev, Minko.** AngularJS Style Guide. [Online] https://github.com/mgechev/angularjs-style-guide.

[8] **Motto, Todd.** AngularJS styleguide for teams. [Online] https://github.com/toddmotto/angularjs-styleguide.

[9] **Moore, Dmitri.** AngularJS 1.x WGW (What Goes Where) guide. [Online] September 14, 2014. http://demisx.github.io/angularjs/2014/09/14/angular-what-goes-where.html.

[10] **Hickson, Ian.** Web SQL database. *W3C.* [Online] November 18, 2010. [Cited: April 7, 2015.] http://www.w3.org/TR/webdatabase/.

[11] **Can I Use.** WebSQL. *Can I Use...* [Online] http://caniuse.com/#feat=sql-storage.

[12] **Brody, Chris.** SQLite plugin. *GitHub.* [Online] [Cited: April 7, 2015.] https://github.com/litehelpers/Cordova-sqlite-storage.

[13] **Bloch, Olivier.** WebSQL polyfill. *Microsoft Open Technologies.* [Online] May 5, 2014. [Cited: April 7, 2015.] https://msopentech.com/blog/2014/05/05/websql-plugin-for-apache-cordova/.

[14] **Can I Use.** IndexedDB. *Can I Use...* [Online] http://caniuse.com/#feat=indexeddb.

[15] **Camden, Raymond.** IndexedDB on iOS 8 – Broken Bad. *Raymond Camden's Blog.* [Online] September 25, 2014. http://www.raymondcamden.com/2014/9/25/IndexedDB-on-iOS-8--Broken-Bad.

[16] **MSOpenTech.** IndexedDB polyfill. *GitHub.* [Online] https://github.com/MSOpenTech/cordova-plugin-indexedDB.

[17] **SQLite.** Datatypes In SQLite Version 3. [Online] https://www.sqlite.org/datatype3.html.

[18] **Malý, Martin.** REST: architektura pro webové API. *Zdroják.* [Online] August 3, 2009. http://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/.

[19] **Android Developers Docs.** Navigation with Back and Up. *Android Developers Docs.* [Online] http://developer.android.com/design/patterns/navigation.html.

[20] **Fidranský, Pavel.** Pull Request #727 - Bar header secondary and content. *Ratchet repository.* [Online] December 16, 2014. https://github.com/twbs/ratchet/pull/727.

[21] **Gill, Steve.** Apache Cordova CLI 4.0 Release. [Online] October 16, 2014. https://cordova.apache.org/announcements/2014/10/16/cordova-4.html.

[22] **Google Inc.** Remote Debugging on Android with Chrome. [Online] https://developer.chrome.com/devtools/docs/remote-debugging.

[23] **Roes, Tim.** Old WebView vs. Chromium backed WebView Benchmark. [Online] November 23, 2013. https://www.timroes.de/2013/11/23/old-webview-vs-chromium-webview/.

[24] **Mitt, Eric.** Apache Cordova gains Windows 8.1 and Windows Phone 8.1 support. [Online] September 25, 2014. https://msopentech.com/blog/2014/09/25/apache-cordova-gains-windows-8-1-and-windows-phone-8-1-support-2-2/.

[25] **MSOpenTech.** JavaScript Dynamic Content shim for Windows Store apps. *MSOpenTech GitHub.* [Online] https://github.com/MSOpenTech/winstore-jscompat.

Attachments

List of images

List of tables

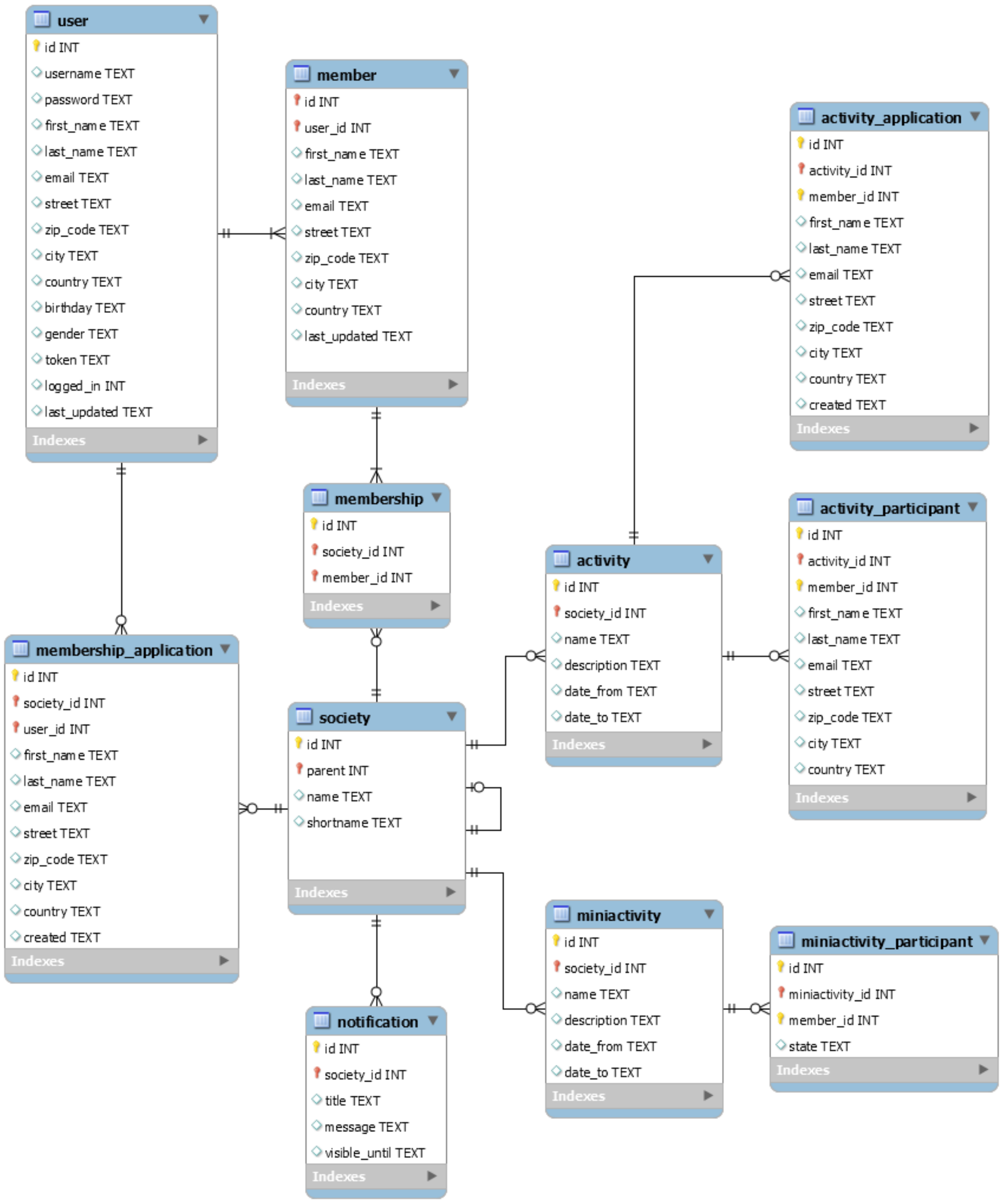


Image 6.1.1: an ER diagram in general

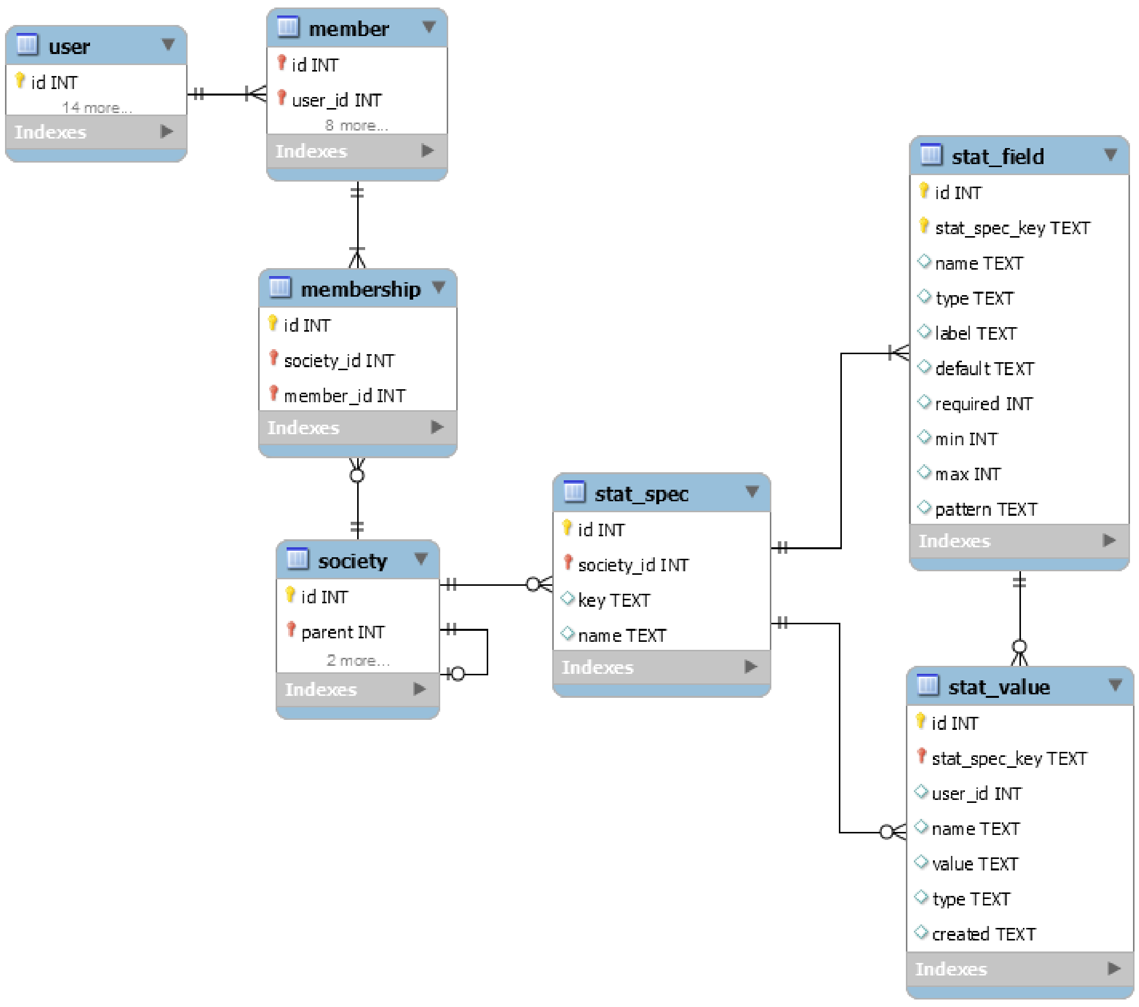


Image 6.1.2: a simplified ER diagram including statistics

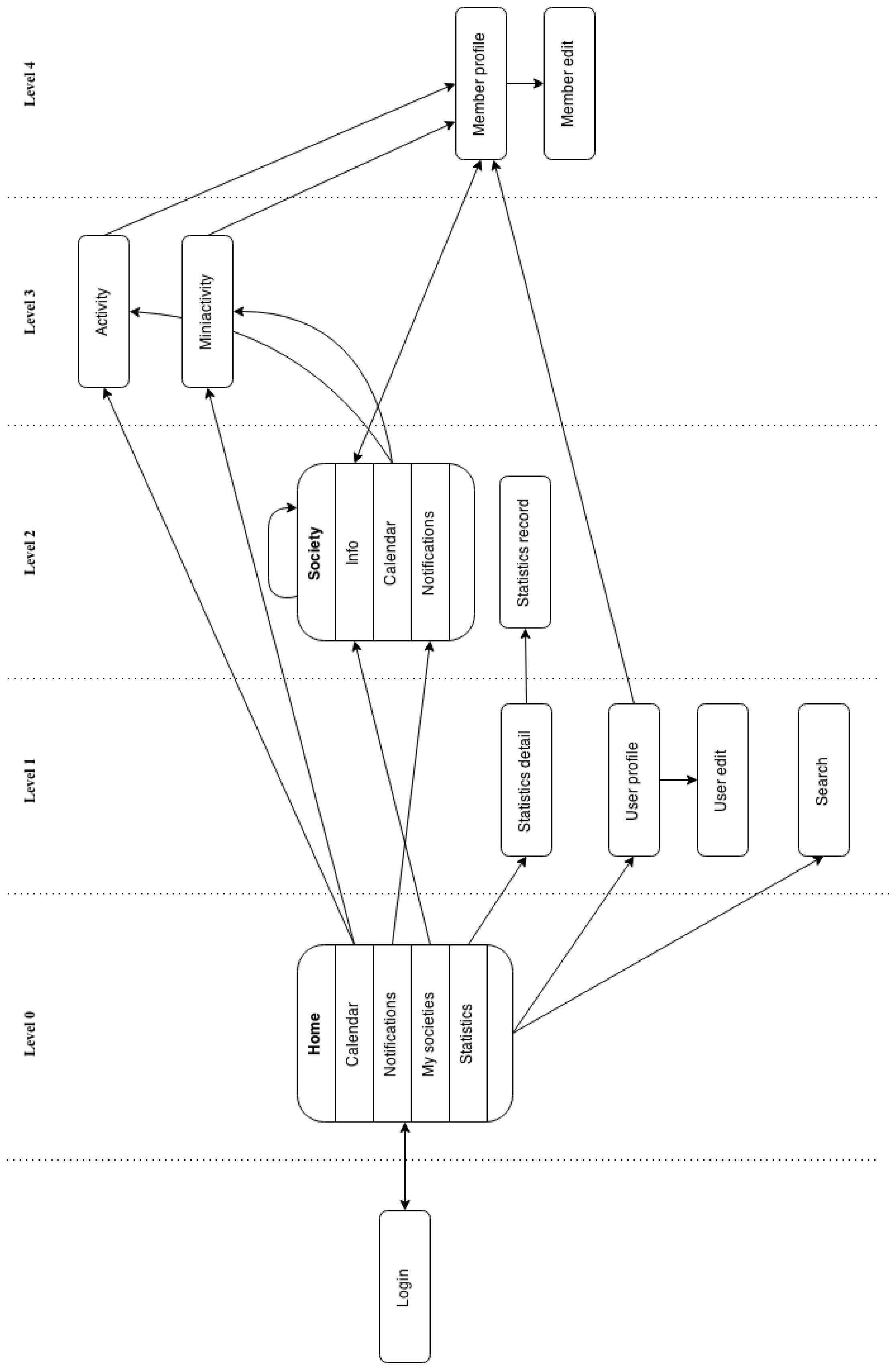
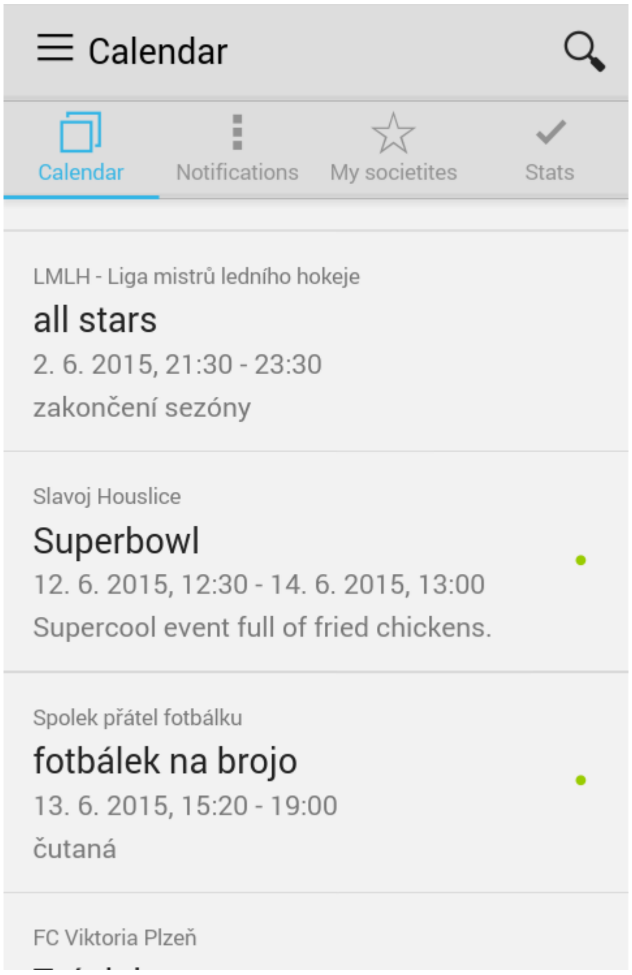


Image 8.1.3: a diagram of views, their relations and levels



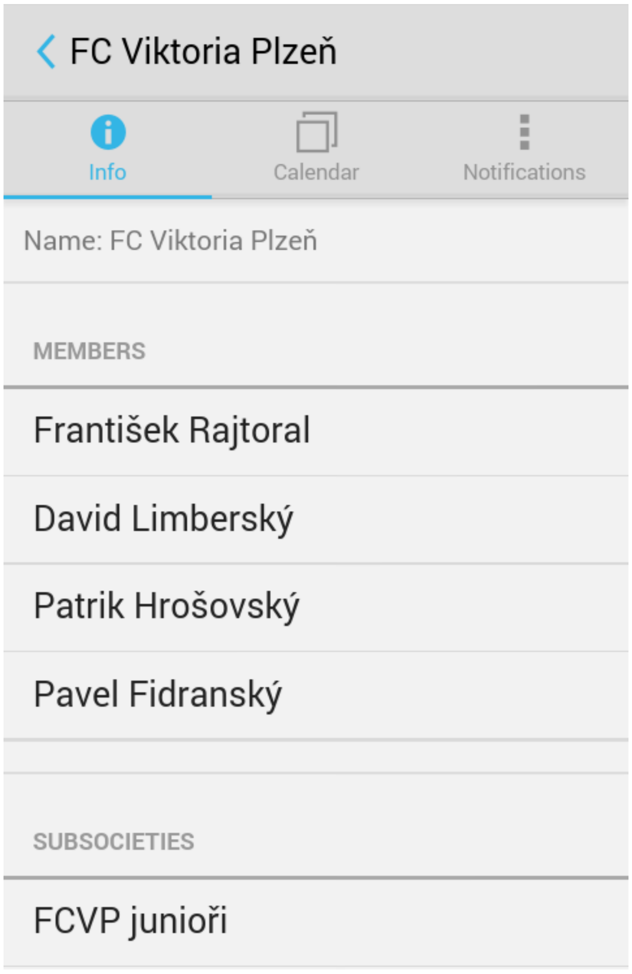
LMLH - Liga mistrů ledního hokeje
all stars
2. 6. 2015, 21:30 - 23:30
zakončení sezóny

Slavoj Houslice
Superbowl
12. 6. 2015, 12:30 - 14. 6. 2015, 13:00
Supercool event full of fried chickens.

Spolek přátel fotbálku
fotbálek na brojo
13. 6. 2015, 15:20 - 19:00
čutaná

FC Viktoria Plzeň

Image 8.1.4: header bars and tabs in a calendar of events view



Name: FC Viktoria Plzeň

MEMBERS

František Rajtoral

David Limberský

Patrik Hrošovský

Pavel Fidranský

SUBSOCIETIES

FCVP junioři

Image 8.1.5: tabs in a society detail view