Escuela Universitaria de Ingeniería

Técnica de Telecomunicación

UNIVERSIDAD POLITÉCNICA DE MADRID

Proyecto de Fin de Carrera

Evaluation and implementation of a hybrid cross-platform application combining mobile web-technologies and native device functions

---

Autor:

Anel Mansilla, Clara

Tutor:

González Martínez, Carlos

Septiembre de 2013

PROYECTO DE FIN DE CARRERA

PLAN 2000

Tema:	Aplicaciones distribuidas multiplataforma
Título:	Evaluation and implementation of a hybrid cross-platform application combining mobile web-technologies and native device functions
Autor:	Clara Anel Mansilla
Titulación:	Telemática
Tutor:	Carlos González Martínez
Departamento:	DIATEL
Director:	René Pongratz
Tribunal	
Presidente:	Pedro Cobos Arribas
Secretario:	Ana Belén García Hernando
Fecha de lectura:	26 de Septiembre de 2013

Resumen

Este Proyecto de Fin de Carrera presenta un prototipo de aplicación móvil híbrida multi-plataforma para Android y iOS. Las aplicaciones móviles híbridas son una combinación de aplicaciones web móviles y aplicaciones móviles nativas. Se desarrollan parcialmente con tecnologías web y pueden acceder a la capa nativa y sensores del teléfono. Para el usuario se presentan como aplicaciones nativas, ya que se pueden descargar de las tiendas de aplicaciones y son instaladas en el dispositivo. El prototipo consiste en la migración del módulo de noticias financieras de las aplicaciones actuales para móviles de una compañía bancaria reimplementándolo como aplicación híbrida utilizando uno de los entornos de desarrollo disponibles en el mercado para este propósito.

El desarrollo de aplicaciones híbridas puede ahorrar tiempo y dinero cuando se pretende alcanzar más de una plataforma móvil. El objetivo es la evaluación de las ventajas e inconvenientes que ofrece el desarrollo de aplicaciones híbridas en términos de reducción de costes, tiempo de desarrollo y resultado final de la aplicación. El proyecto consta de varias fases.

Durante la primera fase se realiza un estudio sobre las aplicaciones híbridas que podemos encontrar hoy en día en el mercado utilizando los ejemplos de linkedIn, Facebook y Financial times. Se hace hincapié en las tecnologías utilizadas, uso de la red móvil y problemas encontrados. Posteriormente se realiza una comparación de distintos entornos de desarrollo multi-plataforma para aplicaciones híbridas en términos de la estrategia utilizada, plataformas soportadas, lenguajes de programación, acceso a capacidades nativas de los dispositivos y licencias de uso. Esta primera fase da como resultado la elección del entorno de desarrollo más adecuado a las exigencias del proyecto, que es PhoneGap, y continua con un análisis más detallado de dicho entorno en cuanto a su arquitectura, características y componentes.

La siguiente fase comienza con un estudio de las aplicaciones actuales de la compañía para extraer el código fuente necesario y adaptarlo a la arquitectura que tendrá la aplicación. Para la realización del prototipo se hace uso de la característica que ofrece PhoneGap para acceder a la capa nativa del dispositivo, esto es, el uso de plugins. Se diseña y desarrolla un plugin que permite acceder a la capa nativa para cada plataforma. Una vez desarrollado el prototipo para la plataforma Android, se migra y adapta para la plataforma iOS.

Por último se hace una evaluación de los prototipos en cuanto a su facilidad y tiempo de desarrollo, rendimiento, funcionalidad y apariencia de la interfaz de usuario.

Abstract

This bachelor's thesis presents a prototype of a hybrid cross-platform mobile application for Android and iOS. Hybrid mobile applications are a combination of mobile web and mobile native applications. They are built partially with web technologies and they can also access native features and sensors of the device. For a user, they look like native applications as they are downloaded from the application stores and installed on the device. This prototype consists of the migration of the financial news module of current mobile applications from a financial bank reimplementing them as a hybrid application using one of the frameworks available in the market for that purpose.

Development of applications on a hybrid way can help reducing costs and effort when targeting more than one platform. The target of the project is the evaluation of the advantages and disadvantages that hybrid development can offer in terms of reducing costs and efforts and the final result of the application.

The project starts with an analysis of successfully released hybrid applications using the examples of linkedIn, Facebook and Financial Times, emphasizing the different used technologies, the transmitted network data and the encountered problems during the development. This analysis is followed by a comparison of most popular hybrid cross-platform development frameworks in terms of the different approaches, supported platforms, programming languages, access to native features and license. This first stage has the outcome of finding the development framework that best fits to the requirements of the project, that is PhoneGap, and continues with a deeper analysis of its architecture, features and components.

Next stage analyzes current company's applications to extract the needed source code and adapt it to the architecture of the prototype. For the realization of the application, the feature that PhoneGap offers to access the native layer of the device is used. This feature is called plugin. A custom plugin is designed and developed to access the native layer of each targeted platform. Once the prototype is finished for Android, it is migrated and adapted to the iOS platform.

As a final conclusion the prototypes are evaluated in terms of ease and time of development, performance, functionality and look and feel.

Prologue

Since the increasing demand of IT systems in almost every domain of our modern life, companies are forced to stem tremendous financial costs in order to win the competition of bringing their solutions to the customer. This ongoing match dealing with a trade-off between software quality, time to market and development costs pushes more and more the search for alternatives to be most effective in all areas of the development process. Cortal Consors as one of the leading European broker in personal investment and online trading started about five years ago to offer mobile channels for customers for giving the ability to fulfill their daily financial business via smartphones and tablets.

Driven by the technological progress of devices, the changing user acceptance, fully connected intelligent backend systems and the constant increase of transmitted data almost in real-time lead us to development costs of mobile applications exceeding millions of euros. With the assignment of this bachelor thesis Cortal Consors wants to achieve a proof of concept in which extent the use of mobile hybrid frameworks can fulfill the need of a financial institute under the aspects of reducing time of development and costs, security, reliability, performance and need of special knowledge. To reach this goal the bachelor thesis had to address the following milestones:

- Research of available mobile hybrid cross platform applications
- Exposing of differences between mobile and native solutions
- Research on current state of the art of hybrid cross platform frameworks and research on related work
- Setup and configuration of the chosen hybrid framework and the corresponding development environment
- Design of a hybrid application in terms of the migration of an existing application module of our current available applications

- Proof of deployment of the developed application on different operating systems like iOS and Android
- Tests in terms of UI (look & feel), performance and reliability
- Documentation of the results

During the given time of 5 months all tasks were completed and the proof of concept for the development of a hybrid application for iOS and Android was established. Both applications demonstrate the quality of the new hybrid approach and show imposingly the strength of the selected hybrid framework. From the current state of view Cortal Consors will focus in future work on the proposed approach acquired by this thesis. A significant reduction of development costs and time can be estimated for sure.

Dipl.-Inf. (Univ.) R. Pongratz, Cortal Consors.

Table of Contents

Illustration Index

Drawing Index

1. Introduction

1.1. Motivation

Mobile technology become more important in our world day by day. We use our smartphone to check our e-mails, communicate to people, and also for work. We read the newspaper in our tablet and we use it for surfing the web, with the outcome that we make less and less use of our personal computers every day.

With the evolution of technologies, every day there are new different mobile platforms competing in the market, making it difficult to make presence in all of them. Each platform uses different programming languages and different development tools increasing the time to market of developing mobile applications for every platform.

In order to reduce costs of development and target the higher number of platforms in the minimum time, we can choose to use hybrid technologies for developing our mobile applications.

Cortal Consors is a leading online broker subsidiary of the french bank BNP Paribas. It was formed by the union of the french company Cortal and the german company Consors on 2003. Cortal Consors is currently present in Germany, France and Spain. They have several mobile applications for different platforms. Currently targeted platforms are iOS, Android and Blackberry. Future support can be also for Windows Phone. These applications are hard to maintain, as each one has its own source code written with

different programming languages and it took a lot of time and high costs until they were completely released to the market.

The hybrid cross-platform approach can open to the company a new concept of mobile development. It allows to be prepared for the future reducing the entry costs to new mobile platforms. With only one source code it can lower the risks on the possible defects that can appear on different implementations when maintaining a single code for every platform. The efforts for user acceptance tests can be reduced too as the plan is that the application has the same look and behaves the same in every targeted platforms. The maintenance of a single source code (or at least as much common code as possible) is much easier and costs less money than maintaining several source codes.

Another possibility that hybrid can bring is to maintain the current applications and prepare them to be hybrid ready, so it can be easy and fast to add new modules (by integrating them as web views) or redesign old modules in a hybrid way.

1.2. Goals

Cortal Consors mobile applications offer a wide variety of financial business functionalities like banking, trading, the use of watchlists, market search or financial news. Especially financial news pose a denotative feature for customers who are interested in understanding the progression of securities in the finance business. Imagine a worldwide concern busted, a natural catastrophe happened affecting the oil market or Apple releases a new world changing device. These are certain indicators which let Wallstreet be more busy than on usual days. Cortal Consors trading customers are being well informed at any time about any ongoing occurrences by supplying them with late-breaking news and giving them the ability to decide right in time what to do with their current affected cash items.

Cortal Consors news module is a decoupled software component that help the user to get an overview of the current state of the market in order to analyze it and make better trading decisions. The goal of this project is to prototype a cross platform hybrid application targeting iOS and Android platforms taking the financial news module of current application and migrate and rebuild it following the hybrid approach.

During the realization of the project, the strengths and weak points of hybrid technologies will be learnt as well as performance capabilities and development efforts. For this approach there is the need to develop some part of the application on a native way and the other with web technologies. With the use of web technologies for the user interface, the target is that the application has the same look and feel in both platforms. The user experience of the application should also be the same in every targeted platform as well as the complete functionality.

This prototype can be integrated in the future as a proof of concept in the company's current applications to evaluate the possibility of easy and fast add new modules to the applications and remove them if necessary.

1.3. Related work

Several publications, investigation studies and discussions about the different possibilities for developing mobile applications can be found while browsing expert literature or the world wide web.

With this project I want to give a proof of concept of the hybrid mobile development on a real life company project and how can it benefit and improve the development process. I want to point some articles that I found interesting and provide a good theoretical base for the realization of this project.

In this article from the research company Gartner, Inc. (www.gartner.com) "**Gartner Says by 2016, More Than 50 Percent of Mobile Apps Deployed Will be Hybrid**" we can find interesting predictions about mobile technology. They predict that due to the increasing number of different mobile platforms and customers demand, more than the half of deployed mobile applications will be hybrid by the year 2016. Web technologies and specially the great mobile browser support for HTML5 help the development of portable code reusable for several platforms. This, combined with the possibility of using a native container to have access to native device features make hybrid, as their point of view, probably the most wanted approach for new development.
http://www.gartner.com/newsroom/id/2324917

In the next article about hybrid mobile development is "**Hybrid Mobile App development using PhoneGap: A case study for beginners**", Shine Ravindra (http://shineravindra.wordpress.com/), a former designer and developer for Digital Media describes a case study for beginners in mobile hybrid development. Concretely development with the PhoneGap framework. He gives some hints and things that you should think about before starting developing, like trying not to mimic the native user interface of each platform, but instead of that choose a layout that fits well and looks the same for every chosen platform. He also gives the suggestion of creating HTML templates once the design is over, and the advice to handle the DOM elements carefully.
http://shineravindra.wordpress.com/2013/05/20/hybrid-mobile-app-development-using-phonegap-a-case-study-for-beginners/

Here we have an article published by Shane Church (http://s-church.net/) in www.effectiveui.com "**MOBILE WEB, HYBRID OR NATIVE MOBILE – HOW DO YOU CHOOSE?**", discussing how to choose the right strategy for your mobile applications: web, native or hybrid. He explains that there is no absolute answer for this question, but a requirements-dependent choice. You should evaluate the needs of your application, the target platforms, time to develop, need or no need of accessing device native features, etc. Only the answer to those questions will lead you to the proper decision on which direction to go on your development.

http://www.effectiveui.com/blog/2012/04/05/mobile-web-hybrid-or-native-mobile-how-do-you-choose/

In the Master Thesis "**THE STUDY OF WEB APPLICATION DEVELOPMENT VERSUS NATIVE APPLICATION DEVELOPMENT ON IPHONE**" done by Sunee Waleetorncheepsawat at the Faculty of San Diego State University 2010 we can find a study of web and native application development with the strengths and weeknesses of each approach. It has several interesting points like, for example, that for new devices with better capabilities, the launch time on native application compared to a web application is noticeably faster in slow devices but that the difference is not so high when it comes on devices with better CPU or more memory.
http://sdsu-dspace.calstate.edu/bitstream/handle/10211.10/628/Waleetorncheepsawat_Sunee.pdf

The last article I want to mention is "**Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options**", published by Mario Korf and Eugene Oksman on wiki.developerforce.com. Here they show a good overview of the different options we have for mobile applications development pointing the strengths and weaknesses of native, web and hybrid applications.

Some people are against hybrid cross platform mobile development and some other people are in favor of it. This approach has pros and contras but in some way it is changing the way new mobile applications are designed and developed. Lowering costs and time of development is always interesting for the companies. Anyway, every company has to evaluate all possibilities and decide which way to go with their mobile applications.

2. Basic concepts

In this chapter there is a brief explanation of the basic concepts needed to follow this documentation.

2.1. HTML

The HyperText Markup Language (HTML) refers to the markup language used to build web pages and it is the publishing language of the World Wide Web. It was created in 1991 by Tim Berners-Lee. The current W3C Recommendation for HTML is HTML 4.01, published in December 1999 [HTM12].

A HTML consists of HTML elements written in form of tags. Tags always begin and end with angle brackets (<>). Almost every element consist of one start tag (`<element>` and one end tag (`</element>`). However, there are some empty elements that do not need to be ended like the image element (`<img>`). The basic structure of a HTML document consist of a root element (`<html></html>`) that contains a head (`<head></head>`) and a body (`<body></body>`) element. Before the `<html>` start tag, and always as the first line in the document, there should be the `<!DOCTYPE>` declaration. This declaration contains the HTML version used in the document and the DTD (Document Type Definition) that the document should be written according to. An example of `doctype` declaration is:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
          "http://www.w3.org/TR/html4/strict.dtd">
```

Between the start and end head tags you can specify metadata of the web page as well as resources to be included (style sheets, scripts). The body contains the web page itself.

The following elements will be used in this document:

- `<ul></ul>`: defines an unordered (bulleted) list element
- `<li></li>`: defines one item inside a list element
- `<div></div>`: defines a division or a section in an HTML document

2.2. HTML5

HTML5 is the next version of HTML [HT513]. It is still not definitive and it is also not fully supported by all web browsers. HTML5 has better capabilities for multimedia content. An HTML5 document has the same structure as on previous versions of HTML (root `<html>` root element that contains a header and a body), but a different doc type declaration. In HTML5 it is no longer required to have a reference to a DTD:

```
<!DOCTYPE html>
```

The user interface of the prototype is written in HTML5.

2.3. CSS

Cascading Style Sheets (CSS) [W3o13] refers to a stylesheet language used for describing the presentation of Web pages including colors, layout and fonts. It allows handling the visual presentation of the content for different types of devices, such as large screens, small screens, or printers. CSS is independent of HTML and can be used with any XML-based markup language (HTML, XHTML, XML, SVG, etc). The separation of HTML from CSS helps maintaining sites, sharing style sheets across pages, and custom fitting pages to different environments. This concept is known as the separation of

structure (or content) from presentation. The CSS specifications are maintained by the World Wide Web Consortium (W3C).

The basic syntax consists of the name of the class that you will use to apply the styles to one or more elements followed by styles block defined inside a pair of curly braces ({ }). The style block consists of a series of `property:value` pairs. The following CSS properties will be used in this document:

- transition: CSS3 transitions are effects that let an element gradually change from one style to another. Two values must be specified:
  - The CSS property you want to add an effect to
  - The duration of the effect.

```
.newsElement{
            transition: height 0.2s
}
```

2.4. JavaScript

JavaScript is a (normally) client-side scripting language first implemented by Netscape Communications Corp. in Netscape Navigator 2 beta (1995) [jav11]. It is usually used for providing interactivity and dynamism to web pages. It can be directly integrated in a webpage (HTML document) in the body inside a script element (`<script>` tag) or loaded as an external file inside the header element of the HTML page:

```
<script type="text/javascript" src="js/index.js"></script>
```

2.5. JSON

JavaScript Object Notation (JSON) is a format for data interchange based on a subset of the JavaScript programming language [jso13]. It is used for serializing objects. JSON

objects have a structure that made them easy to understand by humans and also it is easier to parse than, for example, XML. JSON is built on two structures: a collection of name/value pairs (in various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array); and an ordered list of values (in most languages, this is realized as an array, vector, list, or sequence).

A simple example of the definition of a JSON object in JavaScript that is used in this prototype is the following:

```
this.options = {
      newsOpt: sourceNews,
      ntvOpt: sourceNtv,
      youtubeOpt: sourceYoutube,
      hopeeOpt: hopeeNews,
      twitterOpt: twitterNews,
      podcastOpt: podcastNews
   };
```

2.6. ORM

Object-relational mapping (ORM, O/RM, and O/R mapping) is a technique for dealing with data persistence converting data between the type systems used in object-oriented programming and the one used in a relational database, making use of an object-relational mapping software [ORM13]. With ORM you define your data model (as an object) and it is mapped to a relational database table. This helps the programmer to deal with data without the need of using database statements, like SQL statements.

2.7. RSS

Really Simple Syndication, Rich Site Summary or RDF Site Summary (RSS) are the different names used to refer to a specific format for providing changing web content to a subscribed user. This flow of continuos delivered content is known as **RSS feed**. The

content of the feed is provided in an XML format that conforms to the W3C's RDF specification [RSS12]. To receive and read this content, clients can subscribe to these RSS feeds using a **RSS reader** that will get and format the received information to periodically present it to the user in a friendly way so the user is automatically up to date without the need of visiting each site or providing any personal information (like your e-mail address) to be able to receive periodic news letters. RSS is usually offered by news-related sites, social networks and more online publishers [wh13].

2.8. XML

Extensible Markup Language (XML) is a metalanguage, subset of SGML (Standard Generalized Markup Language). XML can be similar as HTML in a way that they both use tags (<>) and properties (`name="value"`) but in HTML these tags are elements that will be interpreted by the browser and in XML they only define pieces of data. XML is a structured way of presenting data. The following example shows an XML document in response to a request to a RSS feed:

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<rss version="2.0" xmlns:content="http://purl.org/rss/1.0/modules/content/"
xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:media="http://search.yahoo.com/mrss/">
<channel>
    <title>Nachrichten von www.cortalconsors.de</title>
    <link>https://www.cortalconsors.de</link>
      <description>Nachrichten von Börse, Wirtschaft und Finanzen</description>
      <language>de-de</language>
      <copyright>copyright 2012 Cortal Consors S.A.</copyright>
      <image>
            <url>https://www.cortalconsors.de/euroWebDe/themes/euroPort/master/de/img/-
                              logo_cortalconsors.gif</url>
            <title>Cortal Consors</title>
            <link>https://www.cortalconsors.de</link>
      </image>
      <pubDate>Wed, 14 Aug 2013 05:04:17 +0200</pubDate>

      <item>
            <title><![CDATA[OkruNews08]]></title>
            <author><![CDATA[zu S&amp;P 500]]></author>
            <pubDate>Wed, 14 Aug 2013 05:04:17 +0200</pubDate>
            <description><![CDATA[#handelsblatt: Streit über Schuldenschnitt:...]]>
            </description>
            <link>http://t.co/4fuCwKqCdz</link>
            <content:encoded><![CDATA[<a href="http://twitter.com/" ]]></content:encoded>
            <category>twitter</category>
      </item>
</channel>
</rss>
```

2.9. Java

Java is a general-purpose, concurrent, class-based, object-oriented computer programming language that is specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that code that runs on one platform does not need to be recompiled to run on another. Java applications are typically compiled to bytecode (class file) that can run on any Java virtual machine (JVM) regardless of computer architecture. Java is, as of 2012, one of the most popular programming languages in use, particularly for client-server web applications, with a reported 10 million users [Wi13a].

2.10. dalvik

Dalvik is the process virtual machine (VM) in Google's Android operating system. It is the software that runs the apps on Android devices. Dalvik is thus an integral part of Android, which is typically used on mobile devices such as mobile phones and tablet computers as well as more recently on embedded devices such as smart TVs and media streamers. Programs are commonly written in Java and compiled to bytecode. They are then converted from Java Virtual Machine-compatible .class files to Dalvik-compatible .dex (Dalvik Executable) files before installation on a device. The compact Dalvik Executable format is designed to be suitable for systems that are constrained in terms of memory and processor speed [Dal13].

2.11. Objective-C

Objective-C is a general-purpose, object-oriented programming language that adds Smalltalk-style messaging to the C programming language. It is the main programming language used by Apple for the OS X and iOS operating systems [Wi13c].

3. Technical Background

3.1. Mobile applications

Mobile applications are designed and developed for mobile platforms, i.e. smartphones, tablets and PDAs. These applications can be included with the devices' operating system or can be downloaded by a user using the different stores (Apple App Store, Google Apps Marketplace) or directly from the internet. There are different types of mobile applications depending on which technologies they are built with: web mobile applications, native mobile applications and hybrid mobile applications.

3.1.1. Web mobile applications

Mobile web applications are similar to websites in that they are server/side applications built with server/side technologies (PHP, Java, ASP.NET) that render HTML locally. Mobile web applications look like mobile applications.
There are differences between web sites and web applications. All web applications are websites, but not all the websites are web applications. A web application is used as a tool, like any other application, while the purpose of websites is just to present information to the user.

3.1.2. Native mobile applications

Native applications run physically on the mobile device and are developed specifically for each different type of device using different programming languages and the development frameworks usually provided by the different vendors (Xcode and Objective-C for iOS, Eclipse and Java for Android, Visual Studio and C# for Windows Phone). You can find these type of applications embedded in the devices' operating system or you can download them from the different marketplaces.

3.1.3. Hybrid mobile applications

Hybrid applications are built with the same technologies as web applications (HTML, CSS, JavaScript) and are wrapped to look like and behave as native applications.
These applications use a native container (UwebView on iOS, WebView on Android) to render and present the HTML code and process JavaScript locally using the devices' browser engine.
One main advantage of hybrid applications is the possibility to access the devices' native features, what you can not achieve with web applications (at least not all the features).
This is made through an abstraction layer that exposes the native functionalities of the device to the web part of the application as if they were JavaScript (PhoneGap, Titanium Mobile, Appcelerator, Rhodes). By this feature, you can share the same code between different platforms, saving time and costs in development taking advantage of the native functionalities of the devices.

3.2. Hybrid mobile applications in the market

Nowadays, more and more companies take the decision to re-implement their native mobile applications as web applications or hybrid applications (web and native). This decision is often taken with the idea of saving costs in development and to target a higher number of different platforms in less time. Some examples of companies that

reimplemented their mobile applications are: linkedIn, Facebook (who reimplemented again their applications as native), Financial Times (who completely abandoned their native application in the Apple App Store to develop a 100% web application but keeps a hybrid application for the Android devices), Xing, Netflix, Foursquare, Twitter, Yelp and more.

In next paragraphs we will take a look into the cases for linkedIn, Facebook and Financial Times.

3.2.1. linkedIn

linkedIn (www.linkedin.com) is a web portal for professional networks. It offers also mobile applications for iPhone, iPad, Android, Blackberry and Windows Mobile. For the first three platforms, the applications are built in a hybrid way. In April 2012 they delivered the iPad application, made 95% with HTML5. For iPhone and Android they use from 40 to 60% HTML5.

For the iPad application they use *backbone.js* and *underscore.js*.

- *Backbone.js* [bac13] gives structure to web applications by providing models with key-value binding and custom events, collections with a rich API of enumerable functions, views with declarative event handling, and connects it all to your existing API over a RESTful JSON interface. The project is hosted on GitHub and Backbone is available for use under the MIT software license. USA Today, DocumentCloud and WordPress.com are three more examples of applications that are built making use of *backbone.js*.
- *Underscore.js* [und13] is a utility-belt library for JavaScript that provides a lot of the functional programming support that you would expect in Prototype.js [pro13] (a JavaScript framework that aims to ease development of dynamic web applications. It offers a familiar class-style object oriented framework, extensive Ajax support, higher-order programming constructs, and easy DOM

manipulation) or Ruby, but without extending any of the built-in JavaScript objects. The project is hosted on GitHub.

To improve the user experience the make use of HTML5 local storage, available in modern mobile web browsers. This way, they can present to the user the requested content shown in the last session while requesting the latest content to the server, and also, provides the functionality that the user can navigate easily through the different sections of the application without having to wait to network requests each time.

Illustration 1: Screenshots of linkedIn mobile applications

In the following chart we can see a comparison between the linkedIn's web site, the mobile web application and the iPad hybrid application in terms of number of network requests, payload and time to response.

This analysis is made using the Fiddler tool available for Windows OS. Fiddler is a tool to analyze the network traffic. By setting up the computer where Fiddler is running on as a proxy and setting that proxy on your mobile applications you can see the network requests, traffic and get some statistics.

	Web site (Desktop)	Web app	iPad app
Request Count	115	24	20
Unique hosts	13	5	2
Bytes Sent	57.672	9.968	5.664
Bytes Received	707.250	82.907	73.807
Requests started at	17:53:36.312	17:55:55.734	17:57:38.812
Responses completed at	17:54:04.437	17:56:22.843	17:58:02.796
Sequence duration	00:00:28.125	00:00:27.109	00:00:23.984

Illustration 2: Fiddler statistics for linkedIn

We can see that in the desktop application, the number of requests is much higher, and also the sent and received data, than for mobile applications, but the loading time is almost the same. In the hybrid iPad application the requested servers as well as the number of requests and interchanged data are less compared to the web site.

3.2.2. Facebook

Facebook (www.facebook.com) is a social network web portal founded by Mark Zuckerberg. They offer their mobile applications for iOS, Android and Windows Phone. The Facebook team decided to re-implement their native mobile applications as hybrid applications (native/web views). As long as the penetration increased and Facebook became more used on mobile devices, the result was an increasing number of complaints from the users regarding the speed and bad user experience of these applications.

Due to this reason, they decided to rollback to their native mobile applications (or almost 100% native), with the result that they noticeably increased the speed of the application and the users gave a good feedback to them.

Having that, the Facebook team announced that the chose for HTML5 for developing their applications was one of the big mistakes of the company. This statement made the Sencha team (Sencha provides *Sencha Touch*, a framework for developing mobile web applications using the standards of HTML5, CSS3 and JavaScript) to take the decision of

developing the most challenging parts of the Facebook application with HTML. This web application is called FastBook and it is available in http://fb.html5isready.com. We can find in the Fastbook application the same or even better performance as in the native Facebook application. When navigating between different sections of the application there is no need of reloading the whole content every time each section is accessed. This increase the speed and response of the application and gives the user a better experience. After demonstrating the great performance of Fastbook, the Sencha team stated that "the problem of the HTML5 Facebook application was not HTML5, but Facebook".
You can find more about this story in http://www.sencha.com/blog/the-making-of-fastbook-an-html5-love-story/.

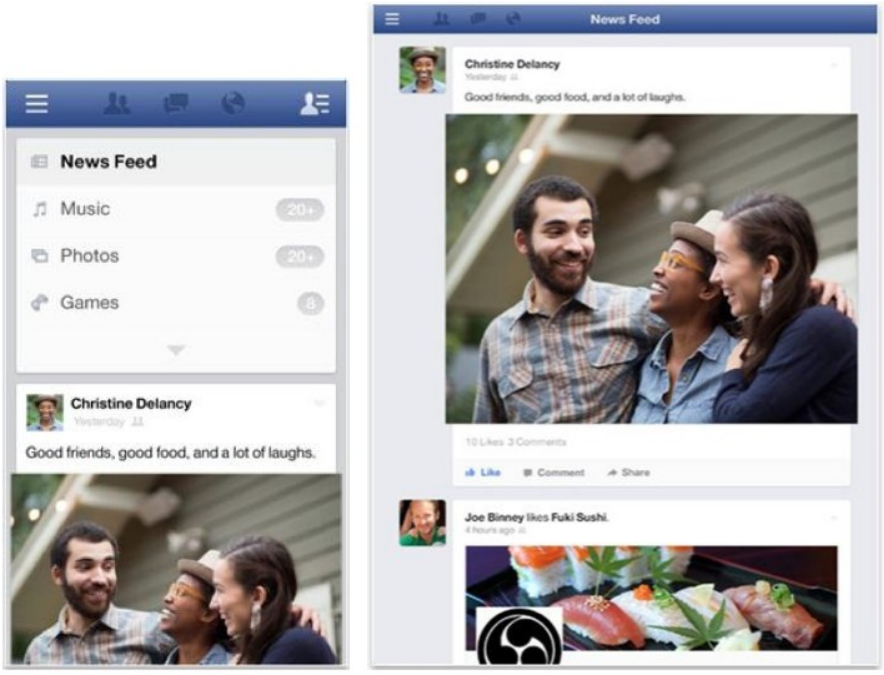


Illustration 3: Screenshots of Facebook mobile applications

In the following chart we can see a comparison between the Facebook's website, the iPhone web application, the iPhone native application and the FastBook web application developed by the Sencha team in terms of number of network requests, payload and time to response.

	Web site (Desktop)	Web app (iPhone)	iPhone app	FastBook (iPhone)
Request Count	70	76	59	54
Unique hosts	16	10	12	14
Bytes Sent	21.521	25.140	23.266	24.720
Bytes Received	12.146	199.188	12.800	180.174
Requests started at	17:47:28.140	17:32:39.218	17:30:11.546	17:04:48.929
Resp. completed at	17:48:03.843	17:33:12.484	17:30:26.828	17:05:03.773
Sequence duration	00:00:35.703	00:00:33.265	00:00:15.281	00:00:14.843

Illustration 4: Fiddler statistics for Facebook

3.2.3. Financial Times

Financial Times (www.ft.com) is an international business newspaper. They have mobile applications for iOS and Android. Financial Times completely abandoned their Apple App Store application to develop a 100% web application accessible by the web browser on an iOS device. With this change they could be present in a larger number of platforms and, in the other hand, they gained a 12% increment on the number of subscribed users, as they could offer a cheaper subscription since they don't have to pay Apple to publish their apps, and increasing 50% of access through smartphones and tablets.

This web application is the basis for the hybrid application that they offer for Android devices (smartphones and tablets).

Illustration 5: Screenshots for Financial Times applications

In the following chart we can see a comparison between the Financial Time's web site accessed on an iPad, the web application for iPad and the hybrid application for Android devices.

	Web site	Web app	Android app
Request Count	119	20	54
Unique hosts	39	4	3
Bytes Sent	60.106	11.833	34.099
Bytes Received	12.732.285	331.976	2.495.972
Requests started at	16:30:35.937	16:27:56.046	16:50:31.500
Responses completed at	16:30:54.093	16:28:06.875	16:51:42.328
Sequence duration	00:00:18.156	00:00:10.828	00:01:10.828

Illustration 6: Fiddler statistics for Financial Times

We can see that the number of network requests for the desktop web site is noticeably bigger than the number of requests for the mobile applications. The same we can find for the sent and received data. In consequence, the loading time is also longer. This behavior is acceptable, since in the desktop version is not so important to take care of the consumed bandwidth, and you always show a larger amount of information in the first loading of the web page.

If we now take a look on the statistics for the web application for iPad and the hybrid application for Android, we will see very important differences. The hybrid Android application makes more than twice the number of requests than the iPad web application, more than three times the sent data and more than seven times the received data. The loading time is one minute ten seconds in the hybrid application while for the iPad web application it is only ten. This is a very important difference and gives the user a bad experience.

3.2.4. Summary

Native applications typically perform faster than mobile web apps. The app stores and marketplaces help users to find native applications. Tools, support and standard development best practices provided by device manufacturers can help speed up development. In the other hand, they are typically more expensive to develop (when supporting multiple mobile devices). Supporting multiple platforms requires maintaining multiple code bases and concurrent different versions can make your app harder to maintain. App store approval processes can delay or prevent the launch of them.

Mobile web applications have a common code base across all platforms. Users don't have to go to a store or marketplace, download the app and install the app. Can be released in any form and any time as there isn't an app store that has to approve the app. If you already have a web app, you can retrofit it with a responsive web design. But web apps can't access all of the device's features (yet). Supporting multiple mobile web browsers can result in higher costs in development. Users can be on different mobile browsers and can make your app harder to maintain.

Hybrid Apps allow to pick out the strengths of both world depending on the requested needs but also the weaknesses.

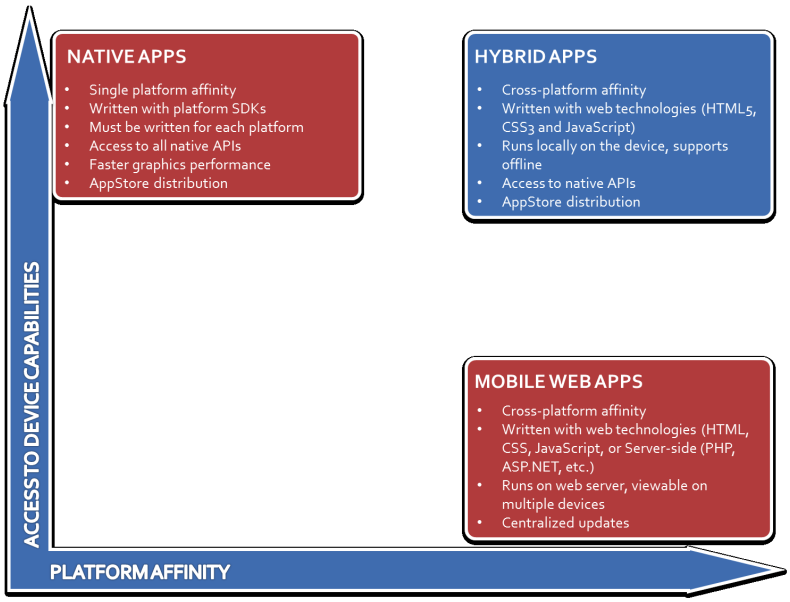


Illustration 7: Mobile applications.
Source: http://www.icenium.com/iceniumImages/blog-images/native-v-hybrid.png?sfvrsn=0

3.3. Hybrid development frameworks

Currently there are many frameworks for developing hybrid applications available for the programmers. Some frameworks use only web technologies, other use programming languages that are different from the native languages used in each platform. The way that they work is different for almost every of them. With some frameworks you develop applications as if they were web applications, while with other frameworks you use web programming languages (such as JavaScript) and that code is converted in human readable native code. We will take a look at the following six hybrid development frameworks as they cover a wide range of different ways of developing hybrid that we can find nowadays in the market: Apache Cordova, Coronalabs, Kony One, Appcelerator Titanium Mobile DE, MoSync and RhoMobile.

3.3.1. Phonegap (Apache Cordova)

PhoneGap is a free licensed, open source solution for building cross-platform mobile applications using standards-based Web technologies (HTML, JavaScript, CSS) [pho12]. In October 2011, PhoneGap was donated to the Apache Software Foundation (ASF) under the name Apache Cordova. PhoneGap is an open source distribution of Cordova. It supports a quite large number of platforms. These are: iOS, Android™, Windows® Phone, Blackberry®, webOS, Symbian™, Windows Phone 7, Windows Phone 8, Windows 8, Bada and Tizen.

For developing with PhoneGap, you use web technologies (HTML, CSS, JavaScript) and there is no interpretation of the code. The result is that the final application makes almost no difference independently from the different targeted platform.

PhoneGap loads your application into a chrome-less web browser and uses JavaScript as an abstraction layer between your mobile web application and the device to expose the native features of this. This framework does not emulate the device user interface. The supported native features by PhoneGap are: accelerometer, camera, capture, compass, contacts and geolocation. However, you can access any other native feature that you need and that is currently not supported by the framework by creating a Cordova Plugin. Cordova Plugins are developed using JavaScript in the web part of the application and platform-dependent programming language in the native part of the plugin, although there is a big number of plugins already created by other programmers and available to download. One SDK for each target platform is needed for developing applications with PhoneGap.

3.3.2. Coronalabs

Corona is a cross-platform development framework for mobile applications. The license of use is free for the Corona SDK Starter (from April 2013) and it supports native iPhone and Android user interface in the Enterprise product [cor13].
Ansca Mobile is the previous name of Corona Labs. The company changed its name in June 2012 to better align its brand with that of its flagship product, Corona SDK. The official name of the company is now "Corona Labs Inc." [cor13b]

Corona currently supports building apps for iOS, Android, Amazon Kindle Fire and Barnes & Noble NOOK. It is more focused on creating games and books (graphically rich applications). However, you can also use it to create other types of applications, such as business applications. When developing with Corona you will use only one SDK for all targeted platforms.

One main disadvantage of using Corona is that once you start using Corona, you won't be able to migrate to another framework, as you use a completely different programming language, and also you cannot combine your own native code with Corona's code.

The programming language used in Corona is Lua. Lua is a lightweight, embeddable scripting language [lua13]. It is designed, implemented, and maintained by a team at PUC-Rio, the Pontifical Catholic University of Rio de Janeiro in Brazil. Lua was born and raised in Tecgraf, the Computer Graphics Technology Group of PUC-Rio, and is now housed at Lablua.

Although you use different programming language than the native languages (you use Lua instead of Objective-C or Java), applications developed with Corona are considered as native applications and not hybrid, as the application is constructed of byte code

compiled from Lua source and that byte code is processed by a native hypervisor that is built in Objective C for iOS and Java for Android [cor13c].

Part of the build process in the Corona client is made on Corona servers, so it requires an Internet connection. The script written in Lua language is precompiled into byte code and then sent to the server. The server embeds this data into the Corona engine, but does not save or archive it. In the end, you get an .app bundle or .apk file ready to deploy as you would get if you used the iOS or Android SDK for the building process.

The supported native features are the following (iOS only): camera, activity indicator, orientation changes and openAL audio (limited support).

3.3.3. Appcelerator Titanium Mobile DE

Appcelerator Titanium Mobile is a free licensed, open source solution for building cross-platform mobile applications. Currently, Titanium Mobile supports iPhone and Android (with BlackBerry support in beta mode for paid subscribers). For developing with Titanium you will use one SDK for all targeted platforms. The programming language used in Titanium is JavaScript. This framework provides a bridge between the JavaScript and the SDK that reads your JavaScript code and uses it to build web views that have the same features as native applications [whi13]. In contrast of PhoneGap, Titanium offers native user interface controls and animations instead of replicating them using CSS or JavaScript. JavaScript written in Titanium framework is not cross-compiled into the respective native languages [top13]. The JavaScript code is evaluated at runtime. It gives access to almost every native UI element with support for customizing the look.

The supported device native features are: accelerometer, camera, capture, compass, contacts and geolocation.

Titanium also allows the programmer to use web technologies (JavaScript, HTML, and CSS).

3.3.4. MoSync

MoSync is a free and open source cross platform mobile application development SDK tool. Individuals, businesses, and organizations who wish to develop MoSync-based applications but who do not wish to publish their source code must purchase a commercial subscription. MoSync enables developers to build and compile applications for iOS, Android™, Windows® Phone, Java, Windows Mobile and MeeGo using only one project for all targeted platforms. The supported native features are graphics, communications, location, contacts, camera, sensors and more.

The SDK allows programming with web technologies (HTML, CSS, JavaScript) and also C and C++. With MoSync you are not restricted to only JavaScript frameworks to replicate native UI, you can truly create native UI elements that are more responsive using only JavaScript. [top13]

3.3.5. Motorola RhoMobile

RhoMobile offers Rhodes, which is an open-source framework based on Ruby, so there is a need knowledge of Ruby programming language. The supported platforms are iPhone, Android, BlackBerry, Windows Mobile, and Windows Phone 7.

A Rhodes application is a web application that runs locally on your mobile device [rho13]. It is implemented with the standard MVC architecture, allowing you to efficiently separate the content from the presentation.

Views are sets of ERB (embedded Ruby) templates: HTML, CSS, and JavaScript files executed by the WebView control available on the device and served by the local web server. This server is a very lightweight web server running on the device. See User Interface section for more details on Layouts, CSS framework, JavaScript frameworks, Menus, and Native UI elements that you may use in the View.
Controllers are sets of usually very simple Ruby scripts in controller.rb files. You have access to many native device capabilities from your controllers. Models are defined by a Ruby script in the model.rb file.

You may generate a Rhodes application using the rhodes utility.

The supported native device features are: accelerometer, camera, capture, compass, contacts and geolocation. However, you can expose a currently not supported device feature by creating a “native extension”.

3.3.6. Kony

KonyOne is a platform for developing mobile (native, web and hybrid) applications introduced by Kony Inc. Kony Native Code Generation lets you generate human-readable native source code in the appropriate language for the operating system [kon13]. Specifically, Objective C for iPhone, Java for Android and BlackBerry and C# for Windows Phone. The generated code is ready to compile with the OS SDK and any third-party libraries in use. The programming language that is used for developing with Kony is

JavaScript. The hybrid approach is achieved with web technologies (HTML5 embeded in a native container).

KonyOne [kpt13] supports the following platforms: iOS 3+, Android 2.0+, Blackberry 4.5+, Windows Mobile 6+, Windows Phone 7+, Symbian and JavaME. They provide access to device-specific features available in each platform.
The pricing for the license is per application, licensing of platform and subscription of hosted/managed services.

3.3.7. Summary

This chapter shows a collection of currently most used hybrid frameworks on the market. Even if all mentioned frameworks follow the same goal, differences can be found in their philosophy and approach.

If certain low level device features like sensors have to be addressed within a project a closer look at the frameworks low level device interfaces and plugins is recommended. There is no clear suggestion for naming a one and only solution because developers must compare pros and contras individually in reference to the needs of their projects.

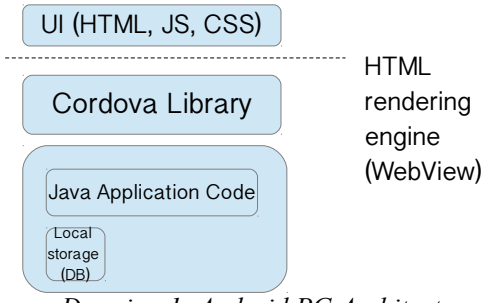
4. PhoneGap

For the realization of this project I decided to choose PhoneGap as the cross-platform hybrid mobile development framework for the prototype. PhoneGap is free licensed, what means no additional costs for the use of the framework, and open source. It is continuously evolving and offering better possibilities to the programmers. For the cross-platform part of the development it uses web technologies (HTML, JavaScript, CSS), so it is a good start point for web developers that want to target mobile applications. The platform support of PhoneGap is quite extensive. Currently supported platforms are iOS, Android™, Windows® Phone, Blackberry®, Symbian™ and many more. That offers the company the possibility to be present for many more customers with every new targeted platform. Connections to the servers and local data storage should be implemented as native in this project and this is possible by the use of the Cordova plugins offered by PhoneGap. This plugins are used to map some native functionalities from native to JavaScript. The last point to take on account for the decision is the look and feel of the application. Building the user interface with web technologies almost guarantees similar look in every platform. There will be no use of native (platform-dependent) user interface components. This can be a contra, since many users want the applications to use the native interface they are used to (the back button on Android, for example) but with a good design you can make a very intuitive and user friendly application without the need of use native components that looks and behaves the same (that is the goal) in every platform.

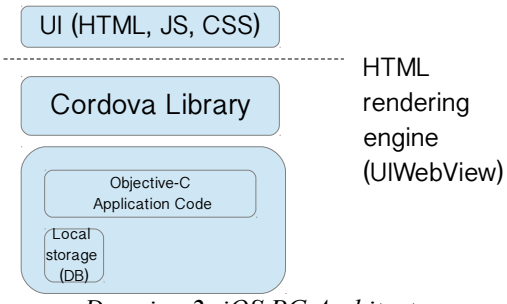
4.1. PhoneGap Architecture

As described in the overview of hybrid cross-platform development frameworks, PhoneGap loads your application into a web view and uses JavaScript as a bridge between your mobile web application and the device to access native capabilities, so the basic components that every application developed with PhoneGap has are the following:

- The web view: a web view is a controller for displaying web pages that uses the browser's rendering engine but has no user interface (no tool bar, no url field, etc). Everything that is shown is the webpage. The rendering engine on iOS and Android is WebKit. The controllers in each platform are UIWebView for iOS and WebView for Android.
- A JavaScript to Native bridge to allow the communication between the HTML application and the native platform.
- A native to JavaScript bridge to allow the native platform to communicate with the HTML application.



Drawing 1: Android PG Architecture



Drawing 2: iOS PG Architecture

In next paragraphs we will take a deeper look on how these bridges work for the two targeted platforms: Android and iOS.

4.2. Android PhoneGap Application

On Android, the JavaScript to native bridge is implemented using the *prompt* command. The JavaScript functions through which you access the native features (like camera, contacts, etc) are converted to Prompt commands by the Cordova JavaScript and intercepted by the WebView using the *onJsPrompt* method. This method, based on a specific signature, calls the respective native plugin (camera, contacts etc).

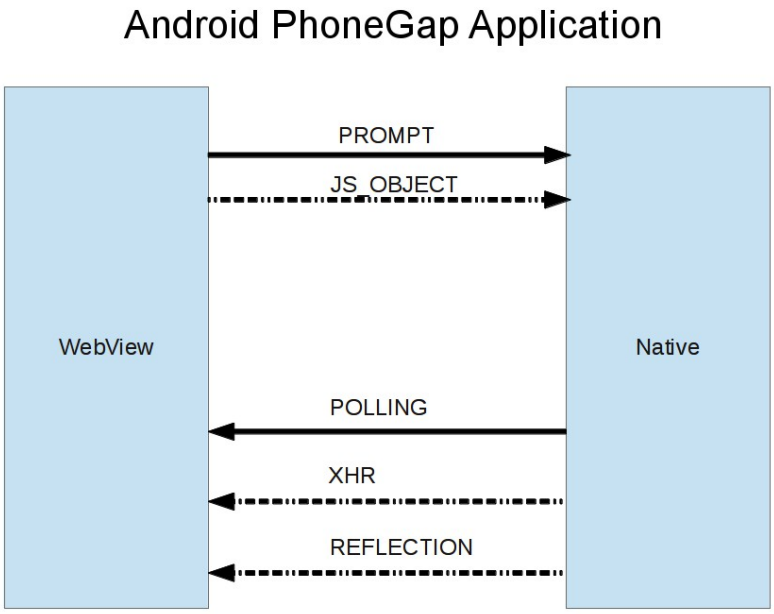


Illustration 8: Android PhoneGap application

The native to JavaScript default bridge (it is, the bridge that is used for the supported native capabilities included in PhoneGap) is made by polling. The JavaScript keeps polling the native side for a response every 50 milliseconds. For custom plugins (the mentioned Cordova plugins), the bridge is set to the XHR (XMLHttpRequest) bridge. This bridge runs a callback server locally and responds to the XHR requests.

4.3. IOS PhoneGap Application

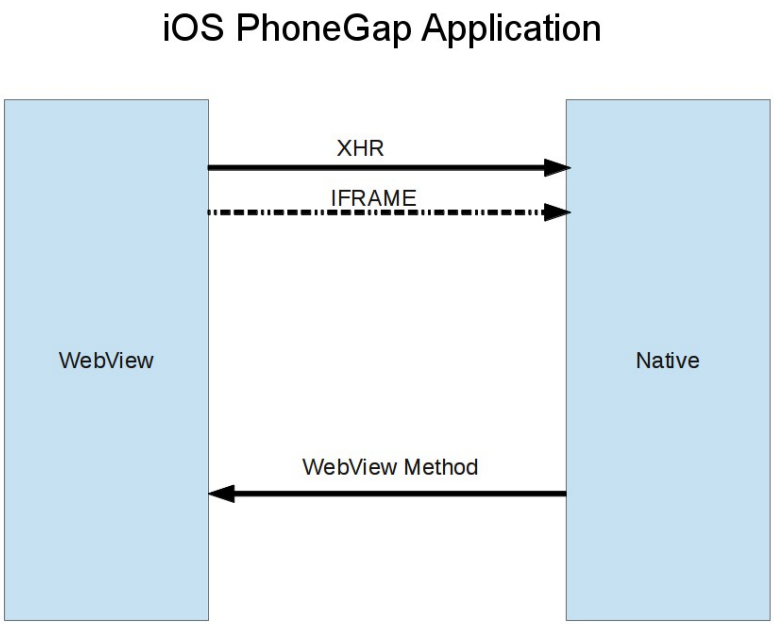


Illustration 9: iOS PhoneGap Application

On iOS 4.2 and below, the way of communication between the JavaScript and the native is using an *iframe*. The JavaScript calls are stores in a JavaScript queue. The native component reads from this queue and executes the calls.

Another way of communication is using a XHR (XHttpRequest) bridge. The bridge makes calls to a fake URL adding the commands in the header. These commands are intercepted, serialized and then executed.

The native to JavaScript bridge is made through a UIWebView. The communication happens using a method from this UIWebView called *stringByEvaluatingJavaScriptFromString*. This method is the one used to run JavaScript scripts.

4.4. Basic Components

A PhoneGap application itself is implemented as a web page. This web page references the CSS, JavaScript, images, media files, or other resources that are necessary for the application to run. The default name for that webpage is *index.html* but you can choose a different name editing the following line configuration file.

```
<content src="index.html" />
```

The configuration file, named *config.xml*, allows developers to easily specify metadata about the application. This file adheres to the W3C's Packaged Web App, or widget specification. You can specify in this file for example the device orientation (portrait, landscape or both), the targeted type of device (handset, tablet or both), if the application executes in fullscreen mode, if the application uses a splash screen, and many other options. Note that some of these options are platform-related, it is, some of them are only available for iOS, some other for Blackberry, etc. The location of this configuration file must be in the root directory (same level as the *index.html* file) for iOS projects and in the *res/xml/* directory for Android projects [pho13].

A WebView within the native application wrapper executes the PhoneGap application. It is necessary to reference also a *phonegap.js* file so the application is able to interact with the native device features in the same way as native applications do. This JavaScript file provides the necessary bindings for this communication. The PhoneGap-enabled WebView may provide the application with its entire user interface but it can also be a component within a larger, hybrid application that mixes the WebView with native application components. PhoneGap provides a plugin interface for these components to communicate with each other.

4.5. Cordova Plugin

Cordova plugin is the name for the bridge between the WebView that executes the PhoneGap application and the native platform the application is running on.
Plugins developed for PhoneGap consist of two main parts [plu13]: a single JavaScript interface used across all platforms and native implementations following platform-specific Plugin interfaces that the JavaScript calls into. The JavaScript interface of the plugin defines the prototype of your Cordova plugin, it is, the methods that the plugin consists of. All of those methods need to be implemented in the native part of the plugin. The way to link those methods defined in the JavaScript and implemented in the native part is to use the *cordova.exec* function:

```
cordova.exec(function(winParam) {}, function(error) {}, "service",
             "action", ["firstArgument", "secondArgument", 42,
             false]);
```

The parameters of this method are detailed below:

- `function(winParam) {}`: Success function callback. Assuming your exec call completes successfully, this function is invoked (optionally with any parameters you pass back to it).
- `function(error) {}`: Error function callback. If the operation does not complete successfully, this function is invoked (optionally with an error parameter).
- `"service"`: The service name to call into on the native side. This is mapped to a native class.
- `"action"`: The action name to call into. This is picked up by the native class receiving the exec call, and, depending on the platform, essentially maps to a class's method.
- `[/* arguments */]`: Arguments to pass into the native environment.

The following example is one of the defined functions of the custom plugin developed for this prototype:

```
getNewsDetails: function(idNew, success, error) {
                    this.options = {
                          newsId: idNew
                    };
                    console.log('getNews: '+idNew);
                    cordova.exec(success, error, 'CocoNews', 'getNewsDetails',
[this.options]);
              }
```

This JavaScript interface is the common part in every project that you will use to target different platforms, it will be the same code for every platform. The native part of the pluging is the platform-specific part of the development. It must be developed using the specific languages and tools available for every platform. We will take a look into the format and components of the native part of the plugin for both targeted platforms in this prototype.

4.5.1. Android plugin

The native part of an Android plugin consists of at least one Java class that extends the *CordovaPlugin* class. A plugin must override one of the `execute` methods from *CordovaPlugin*. As best practice, the plugin should handle pause and resume events, and any message passing between plugins.
The definition of the custom plugin is set in the *config.xml* file located in the *res/xml/* directory of the Android PhoneGap project [pla13].

```
<plugin name="<service_name>" value="<full_name_including_namespace>"/>
```

The service name should match the one used in the JavaScript exec call, and the value is the Java classes full name, including the namespace.

This is the actual declaration of the custom plugin in this prototype for Android (news plugin):

```
<plugin name="CocoNews" value="org.apache.cordova.plugin.CocoNews"/>
```

CocoNews is the name in the JavaScript interface. The name of the package and Java class is `org.apache.cordova.plugin.CocoNews`.

In the Java class of the plugin it is necessary to implement the `execute` method. Inside this implementation you handle the different calls from the JavaScript. The name of the call is passed in the `action` string and matches the name defined in the JavaScript file. The next picture is an example of a basic implementation of the `execute` method.

```
@Override
public boolean execute(String action, JSONArray args, CallbackContext callbackContext) throws
JSONException {
    if ("beep".equals(action)) {
        this.beep(args.getLong(0));
        callbackContext.success();
        return true;
    }
    return false;  // Returning false results in a "MethodNotFound" error.
}
```

The following is the basic structure of the implementation of the `execute` method in the *CocoNews* class:

```
public boolean execute(String action, final JSONArray args, final CallbackContext
callbackContext) throws JSONException {

        if (action.equals("getNews")) {
             ...
            return true;
        }else if (action.equals("getRecentNews")) {
             ...
            return true;

        }else if (action.equals("loadOlderNews")) {
             ...
            return true;

        }else if (action.equals("getNewsDetails")) {
             ...
            return true;

        }else if (action.equals("setNewsOptions")) {
             ...
            return true;
        }else if (action.equals("clearNewsList")) {
             ...
            return true;
        }
        return false;
    }
```

The result of the function in the native part is passed to the JavaScript in a JSON format.

4.5.2. iOS plugin

The native part of the plugin for iOS is an Objective-C class that extends the `CDVPlugin` class. The plugin must be added under the `<plugins>` tag of the *config.xml* file in the Cordova-iOS application's project folder.

```
<plugin name="service_name" value="PluginClassName" />
```

The key `service_name` should match the one used in the JavaScript exec call, and the `value` will be the name of the Objective-C class of the plugin [pli13].

The actual declaration of the custom plugin in this prototype for iOS (news plugin) is the following:

```
<plugin name="CocoNews" value="CocoNews" />
```

As in the Android project, the key name *CocoNews* is the name of the plugin defined in the JavaScript interface. The key value *CocoNews* represents the name of the Objective-C class.

In the Objective-C class of the plugin it is necessary to implement the different `action` methods. There will be one implemented method per defined function in the JavaScript interface and they must have the same name as defined in the JavaScript interface. The next picture is an example of implementation for one of the `action` methods.

```
- (void)myMethod:(CDVInvokedUrlCommand*)command
{
    CDVPluginResult* pluginResult = nil;
    NSString* myarg = [command.arguments objectAtIndex:0];

    if (myarg != nil) {
        pluginResult = [CDVPluginResult resultWithStatus:CDVCommandStatus_OK];
    } else {
        pluginResult = [CDVPluginResult resultWithStatus:CDVCommandStatus_ERROR messageAsString:@"Arg
was null"];
    }
    [self.commandDelegate sendPluginResult:pluginResult callbackId:command.callbackId];
}
```

The following is a basic example for the declaration of these methods in the custom plugin for this project:

```
- (void)getNews:(CDVInvokedUrlCommand*)command;
- (void)getLatestNews:(CDVInvokedUrlCommand*)command;
- (void)loadOlderNews:(CDVInvokedUrlCommand*)command;
- (void)getNewsDetails:(CDVInvokedUrlCommand*)command;
- (void)setNewsOptions:(CDVInvokedUrlCommand*)command;
- (void)clearNewsList:(CDVInvokedUrlCommand*)command;
```

The result of the function in the native part is passed to the JavaScript in a JSON format.

5. Description of the proposed solution

The financial news part of current application consists of a set of financial news that help the user to get an overview of the current state of the market in order to analyze it and make better trading decisions. The target of the project is to make a prototype of a hybrid application. The financial news module will be taken from the current application and migrated and rebuilt as a hybrid application.

5.1. Business requirements

The application must accomplish the following requirements:

- ➢ Retrieve news from the news server and present them to the user: the news feed is provided as XML content. The application must parse the XML content, format the news and present them in the user interface.
- ➢ The user should have the option to filter the different news sources: there are several news sources available for the user. The user can select from one to all of the sources.
- ➢ The user should have the possibility to load older news: by scrolling the list of news, older news should be presented to the user in a chronological order.
- ➢ The news should be refreshed every certain time to keep the user up to date: the application must request for recent news to the news server every certain time.

- ➢ The user should be able to read the content of the new when tapping on it: the whole content of the new will be shown when the user taps on a news item (teaser) from the list.
- ➢ The user should be able to increase or decrease the font size of the news content: once the content of the new is shown, the corresponding buttons for changing the font size should appear at the bottom of the section.
- ➢ The application should show the last update date and time: on the top of the screen there must be shown the last time of requesting news.
- ➢ The content of the new should show the media content when available: some of the news sources provide news with multimedia content, pictures, videos, podcast. The application must show this multimedia resources inside of the news content when the new is open.

Additionally, the following improvements will be made to the expected functionality with the hybrid approach:

- ➢ Automatically load older news when scrolling to the bottom of the list
- ➢ Insert news on the top (reference to the afterPubDate section)
- ➢ Adapt the UI to the new approach that will be taken in new version of current application (tap to see the content, etc)
- ➢ Help page to show the user how to use the application

5.2. System architecture

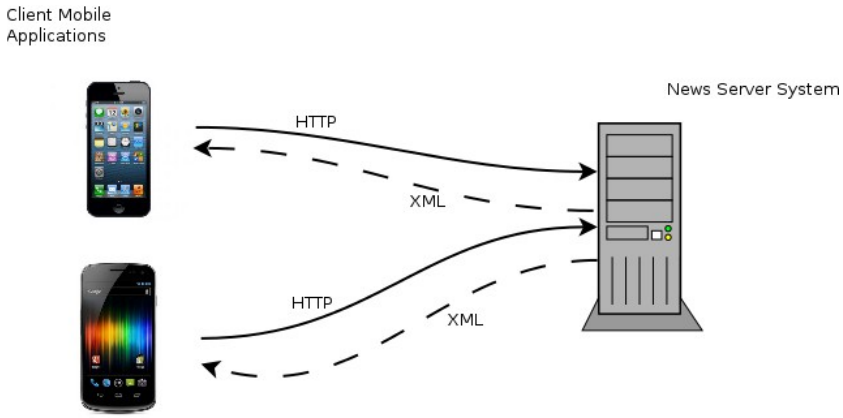


Illustration 10: System architecture

The architecture of the system for delivering RSS content to mobile applications consists of a news server system that provides content to the client mobile applications in an XML format. The client mobile applications request the content using a HTTP request.

5.2.1. The news server

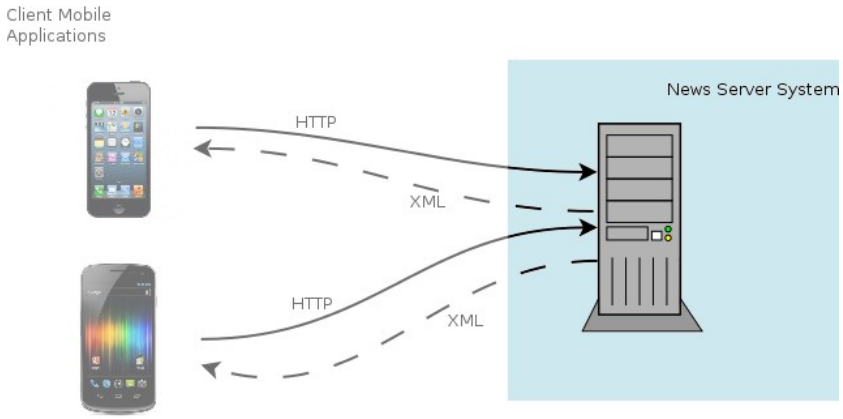


Illustration 11: System architecture: news server system

The Cortal Consors news server is a dedicated system component that is used by several clients and products like ActiveTrader, Cortal Consors Web or Mobile Channels. As shown in figure 12 the news server offers different end points to the clients to connect to. For the mobile side CatFront is used to send and get request from the server. To lower IO activity and to improve the performance to deliver the required data that was requested from callers, the news server internally uses different caching mechanisms.

The observer get news form the different sources by polling. Some of the sources are passed directly to the caller client without saving them in the database. Because of this, there is no ID field in the news object that can be provided with the news, making it difficult for the clients to request for single specific news or even for latest news. As result latest news can only be requested by the publishing date. The reason why some of the sources are not stored locally is because of sources with high data output like twitter. Twitter generates a lot of data and it can cause performance and storage issues to keep them in the local database.

The news server can deliver new news only every 5 minutes to the caller because of internal update policies.

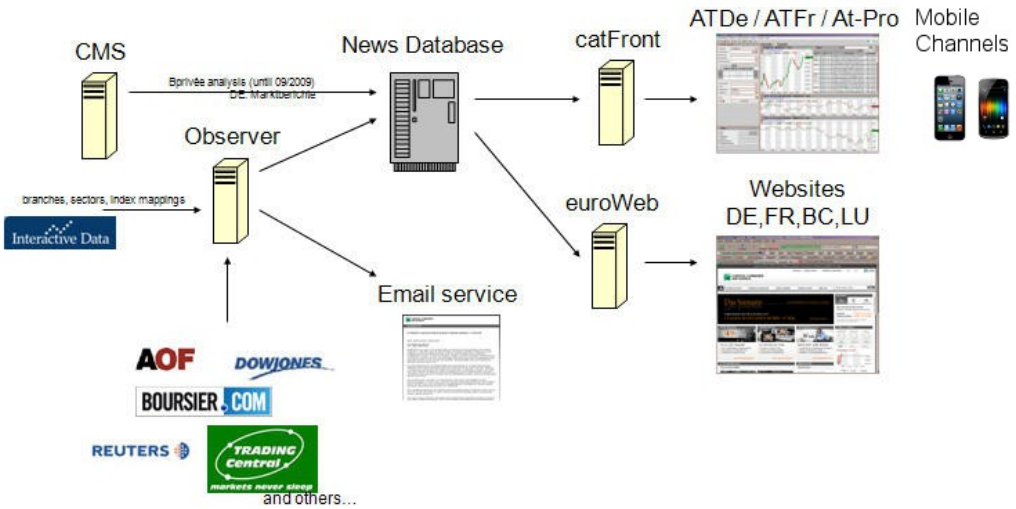


Illustration 12: The news server system

5.2.2. News request

Two news servers provide news in an XML format in response to HTTP requests:

- general news server
- n-tv videos news server

The **general news server** provides financial news from different sources. You can request for five different types of news:

- text news: news retrieved from different financial news sources like *Dow Jones*, *Spiegel Online*, *Manager Magazin*, *Deutsche Börse* and more
- *youtube* videos: news that include links to financial videos from this popular video sharing web site
- *twitter* news: financial related *tweets* from this well known social network
- podcasts: news that include links for multimedia (concretely audio) files related to financial news
- *hopee* news: *hopee* is the social network for investors created by Cortal Consors. The users can publish short opinions and recommendations related to finance.

The url for this server is:

```
http://int-acc.news.cortalconsors.de/CatFront/feed/de/mobile?channel=
```

This request accepts the following parameters:

- **channel:** defines filter-criteria for news. The possible filter-criteria are "twitter", "podcast", "youtube", "news" and "hopee". Format: *channel=filter1,filter2,...*
- **numFetch:** defines the number of news to get. Although *numFetch* is an optional parameter, there is a limit of 50 news for every request, i. e. *numFetch*=60 get the maximum of 50 news. Format: *int*.
- **category:** calls news for this ISIN. category is an attribute-value-pair (e. g. attribute: ISIN, value: DE0009652644). Format: *ISIN_XXXXX*

- **beforePubDate:** gets all news before this date. Format: *yyyyMMddHHmmss*

The different channels are directly requested by the news server to their original source except for the text news. The text news are requested every five minutes by the news server and cached in a local database. This has the drawback that if the client application makes two request for that channel during the time between to consecutive requests of the news server to the sources, the retrieved results for text news will be the same for both requests.

The **n-tv news server** provides links to videos. The url for this server is:

```
http://aktionen.cortalconsors.de/ntv-video/pub/n-tv/ipad/video.xml
```

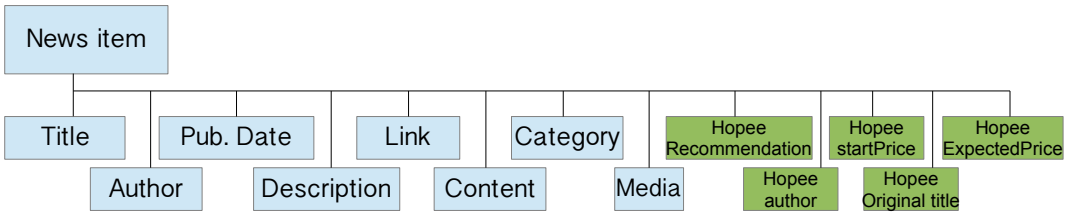
This request does not accept any parameters. The retrieved result will be the same in every request. The maximum number of results is five links to videos per day. This videos are added to the n-tv videos news server at certain times of the day, so the same request will give different results depending on what time the news are requested to the server.

5.3. News model

Now let's have a closer look at the model for financial news. News are requested to the news server using a HTTP request. The news are delivered in an XML format following the RSS item XML structure. The minimum common structure for every news feed is the following:

```
<item>
  <title><![CDATA[DJ TABELLE/Renditen und Spreads]]</title>
  <author><![CDATA[Dow Jones]]</author>
  <pubDate>Fri, 8 Jun 2012 11:13:00 +0200</pubDate>
  <description><![CDATA[aktuelle Spread in Rendite]]></description>
  <link>http://web-ics.consors.de/CatFront/JspNews.jsp?m_id=34425335</link>
</item>
```

The Elements `<title>`, `<author>`, `<pubDate>`, `<description>`, `<link>` are used by all feed-types. For the *hopee* source of news, five more elements are provided. These are: `hopeeRecommendation`, `hopeeStartPrice`, `hopeeExpectedPrice`, `hopeeAuthor` and `hopeeOriginalTitle`. The complete structure for the news will be as follows:



Drawing 3: News model tree

The following list contains all these elements and their description:

- `<title>`: title of the new
- `<author>`: author of the new
- `<pubDate>`: date of published
- `<description>`: brief description of the new
- `<link>`: url for the original source of the new
- `<content>`: body of the new
- `<category>`: channel of the new (hopee, youtube, podcast, etc)
- `<media>`: link for the thumbnail for the youtube videos
- `<hopeeRecommendation>`: the recommendation of the user that publish the content (buy, sell, hold)
- `<hopeeAuthor>`: user that publishes the content
- `<hopeeStartPrice>`:
- `<hopeeOriginalTitle>`: original title for the published content
- `<hopeeExpectedPrice>`:

For dealing with news in the application, the data model that is used is represented by the object GeneralNews.

5.4. Client application

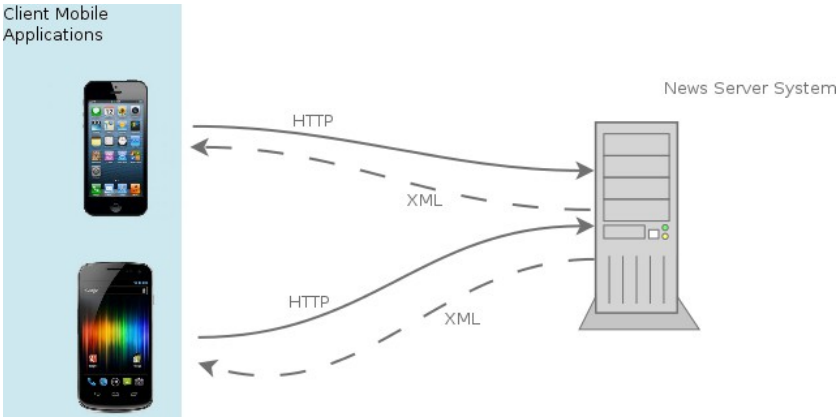


Illustration 13: System architecture: client applications

Before starting with the client application for this prototype, I will make a short review of the news functionality on current application. The financial news module is a section of the complete current mobile application from Cortal Consors. The news are presented to the user as a list.

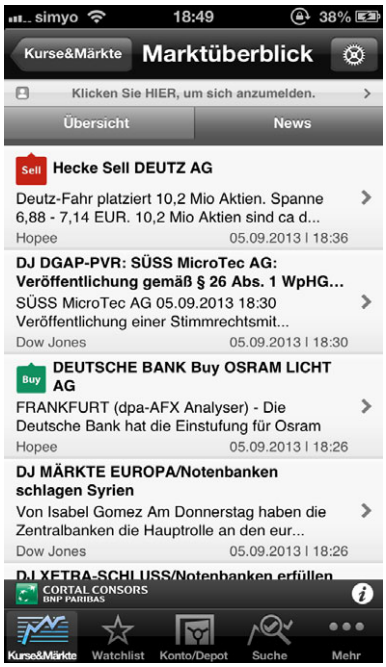


Illustration 15: Current application. News overview

Illustration 14: Current application: single new view

When the user clicks on a teaser, a new view is presented showing the whole content of the new. At the top of this new view, two buttons are placed on the left for increasing or decreasing the font size. On the top right there is one additional button for sharing the new via e-mail.

When studying the behavior of the current application I found several performance issues:

- Every five minutes the application sends a request to the news servers that provides the latest 20 news. The news are parsed, the database is cleared, the UI is cleared, the news are stored in the database and passed to the UI to show to the user. Even if the retrieved news are not different from the stored news (no recent news are found) the flow is exactly the same.
- If the user scrolls the list of news, the 5 minutes timer is paused and no more news are requested until the user closes the application or makes a change on the filtering settings. This is implemented on purpose to prevent the automatic clearing and update of the UI when the user is reading news which gives a bad user experience.

The following diagram illustrates the flow of current application and helps understanding the problems encountered before.

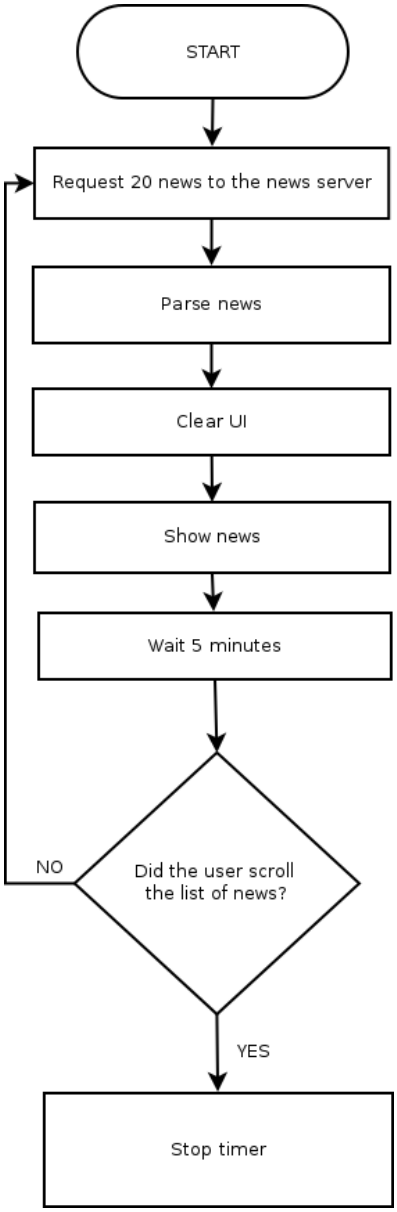


Illustration 16: Flow chart for current application

The current news server supports a parameter called `beforePubDate` that provides news older than the given date with a maximum of 50 news. To improve the performance of the application regarding I suggested a change to the news server. The change consists of

adding a new parameter called `afterPubDate` that provides news published after a given date. The addition of this new parameter solves two problems:

- If no recent news are found, there is no need to make any change in the user interface reducing the unnecessary logic on the client side.
- The recent news found can be inserted on the top of the list of news without the need of clearing the user interface every 5 minutes and eliminating the need of pausing the timer when the user scrolls the list of news.

With the addition of this new parameter, the flow of the prototype will be as shown in next page:

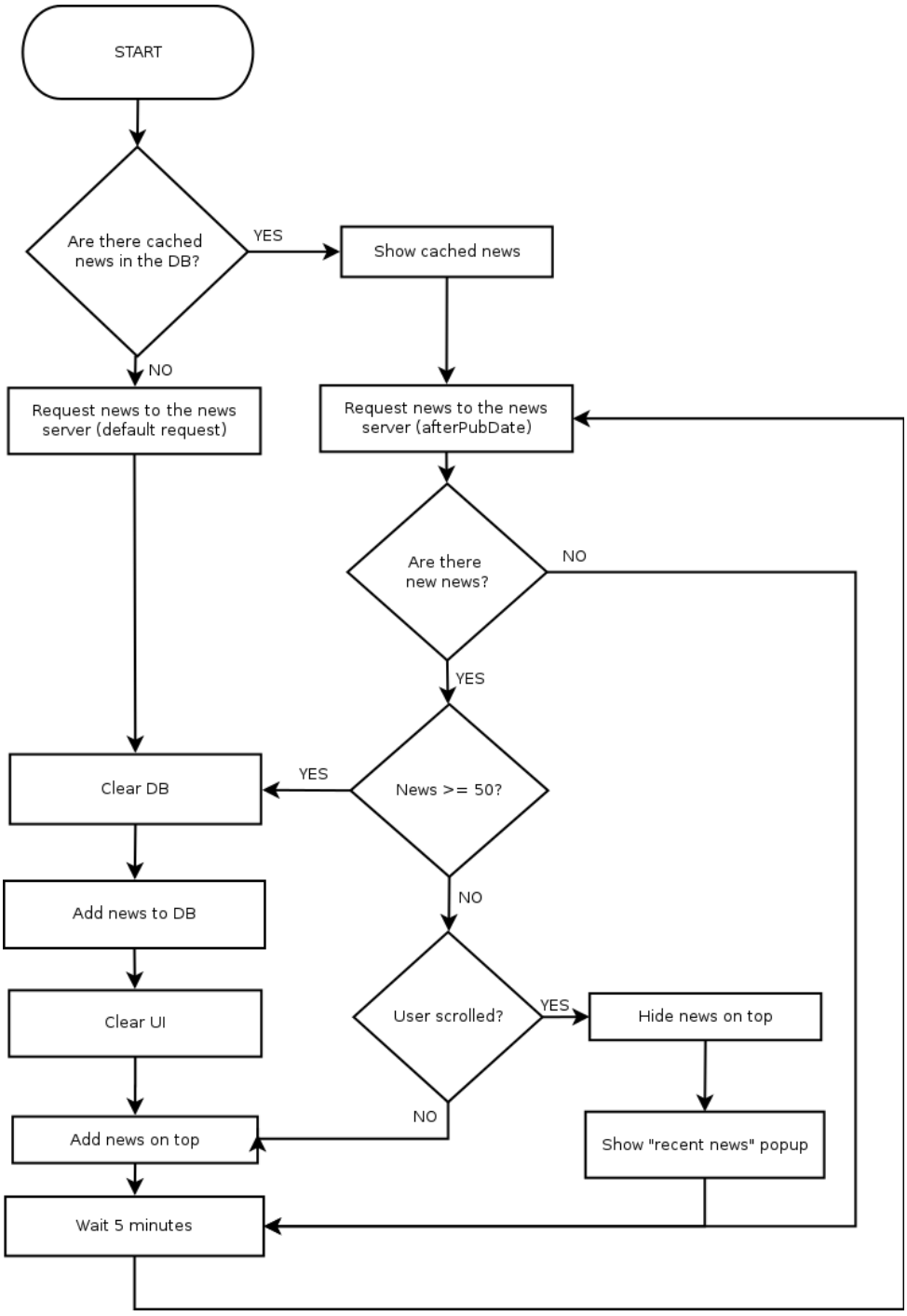


Illustration 17: Flow chart for the prototype

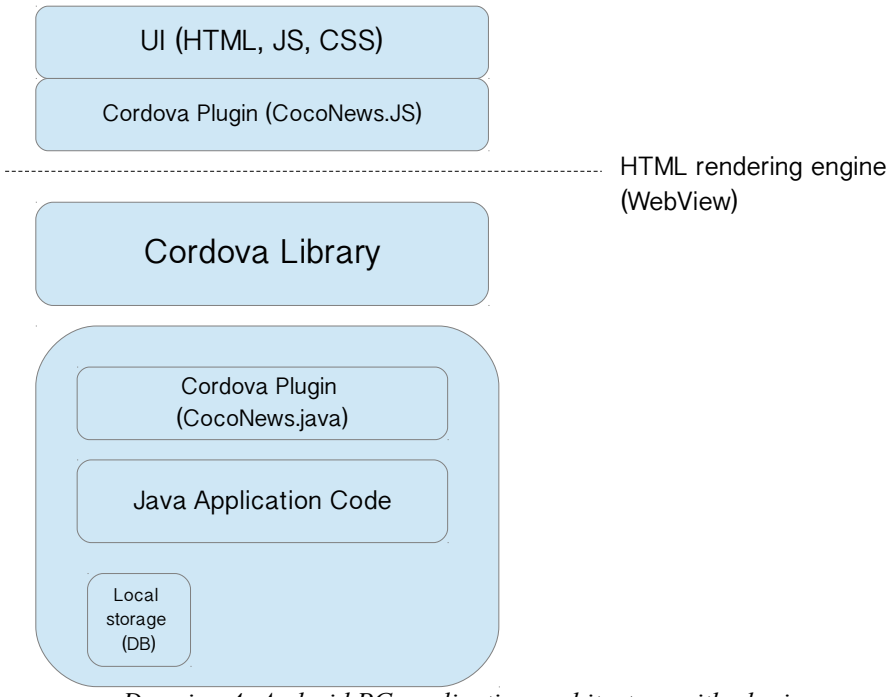
First, show news from the database (in case that there are cached news)
Request news to the news server with the parameter afterPubDate and the value: last publication date stored in the database in case there are news in the database. If there are no news in DB, the request is the default request.
If the number of retrieved news is 50 (the maximum number per response) it means that the cached news are deprecated news so next steps are: clear database, add the retrieved news to the database, clear the UI and show latest news in the UI.
If the number of retrieved news is less than 50, it means that the shown news in UI are up to date. Here there are two different situations:

- if the user did not scroll the list of news: place news on top
- if the user scrolled down the list of news: place news hidden in the top and show popup "news available". The "news available" popup shows a message to the user that more recent news are found. If the user taps on the popup, the screen is scrolled to top to show the recent news first.

5.4.1. Architecture of the application

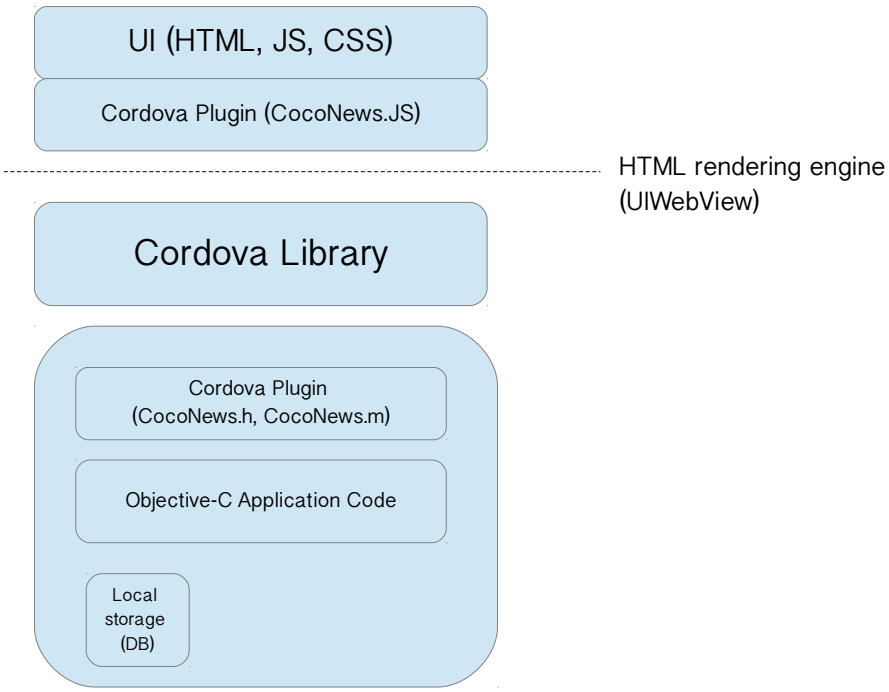
At this point of the document I have studied the structure of a PhoneGap application and the Cordova plugins. The model for the financial news has been set. Now lets see how the prototype will look like.

The following picture represents the architecture of the prototype for the Android client application.



Drawing 4: Android PG application architecture with plugins

The architecture for the iOS application will be the same. The web view in iOS is called UIWebview and the native code for is written in Objective-C.



Drawing 5: iOS PG application architecture with plugins

The application will consist in a user interface developed with web technologies embedded in a native web view container that is able to communicate with the device using the bridge provided by the PhoneGap framework. This part of the code will remain exactly the same for both Android and iOS projects.

Additionally, a custom Cordova plugin will be developed that will provide the communication with the news server as well as other functionalities like parsing and formatting the news or the local database storage of them. The implementation of this plugin has to be done with native code.

6. Implementation

In this chapter I will describe the implementation of each part of the prototype. I will start describing the functionality of the application and the logic in the HTML part of the application. Then, I will describe the custom plugin needed for the connections with the server and handling the news data describing first the JavaScript interface needed for the plugin and then going through the native implementation. Next part will be the user interface, implemented with HTML, CSS and JavaScript. And in the last point I will talk about the HTML local storage for the user preferences.

6.1. HTML application

The application consist of a html file (*index.html*) that is loaded in a web view controller. In the header of this HTML file all the needed resources (stylesheets, scripts, PhoneGap library) are included. This web page is a very simple web page that contains a toolbar on the top (implemented as a `<div>` element), a hidden menu that is shown when the user taps on the settings button from the toolbar, and a JavaScript piece of code that calls the `app.initialize()` method. This method is defined in the JavaScript file *index.js* inside a defined `app` object. This object is defined to bind the different events that can be fired in your application. All the necessary logic to build the news list is also implemented in the *index.js* file. But for understanding that, it is first necessary to understand the Cordova lifecycle events.

6.1.1. Events

A mobile application can receive a notification when something changes on the application or even on the device. For example, if the application changes from foreground to background or if the device has lost its network connectivity or the battery level is low. The PhoneGap framework can listen to those events allowing the programmer to establish the necessary logic to adapt the application to different situations. The events that are treated in this prototype are the following:

- **deviceready:** this event is fired when Cordova is fully loaded. This is a very important event, since the application should wait for the full load of the library before starting any call to a Cordova JavaScript function.
- **pause:** this event is fired when a Cordova application is put into the background
- **resume:** this event is fired when a Cordova application is retrieved from the background

This is the definition of the var object for binding different events. No handling logic is shown in this example:

```
var app = {
    // Application Constructor
    initialize: function() {
        this.bindEvents();
    },
        // Bind Event Listeners
        //
        // Bind any events that are required on startup. Common events are:
        // 'load', 'deviceready', 'offline', and 'online'.
    bindEvents: function() {
        document.addEventListener('deviceready', this.onDeviceReady, false);
        document.addEventListener('pause', this.onPause, false);
        document.addEventListener('resume', this.onResume, false);
    },

    onPause: function() {    },// pause Event Handler
    onResume: function() {    },// resume Event Handler
    onDeviceReady: function() {// deviceready Event Handler
            // Here starts all the logic of the application
    },
    receivedEvent: function(id) { // Update DOM on a Received Event
        var parentElement = document.getElementById(id);
        var listeningElement = parentElement.querySelector('.listening');
        var receivedElement = parentElement.querySelector('.received');

        listeningElement.setAttribute('style', 'display:none;');
```

```
        receivedElement.setAttribute('style', 'display:block;');

        console.log('Received Event: ' + id);
    }
};
```

When the `app.initialize()` method is called in the HTML file, all the defined events are bound. As explained before, the event `onDeviceReady` is fired when the Cordova libraries are fully loaded, and it is in that moment when the application itself can start. The following snippets shows the important parts of the `onDeviceReady` listener function implemented for this prototype (some parts that are not important for this section are omitted or shown as comments):

```
onDeviceReady: function() {

        //Get the user settings or set
            //the default settings if no user settings are found
            cocoNews.setNewsOptions(this.options, success, error);
            ...
            cocoNews.getNews(success, error);
            intervalID = setInterval(refreshNews, refreshInterval);

    },
```

The first line sets the user preferences for the filtering options. This settings are persisted using HTML5 local storage. That will be explained later in this chapter. Then, the application gets the cached news that can be in the database by calling a function from the custom plugin explained in next subchapter and finally it starts a timer for requesting news every certain time. Let's see how this timer works.

6.1.2. Timer

To keep the user up to date, the application makes periodic request to the news server for recent news. As a way to make the code as much portable as possible, I decided to implement the timer in JavaScript using the setInterval method. The setInterval method calls a function or evaluates an expression at specified intervals set in milliseconds. This method will continue calling the function until the method clearInterval() is called. The

call to setInterval returns an ID that can be used as a parameter for the clearInterval() method.

```
intervalID = setInterval(refreshNews, refreshInterval);
```

The time value for the interval is set as a constant in the beginning of the application:

```
var refreshInterval = 300000; //milliseconds
```

This means that every 300000 milliseconds (five minutes) the refreshNews function will be executed.

The application should not make any network requests or logic while paused (or put on the background). To prevent the application to work during the background the necessary logic should be placed in the handler of the onPause and onResume events. The following is the code of the onPause event handler in the application:

```
onPause: function() {
        window.clearInterval(intervalID);
        onPauseTime = new Date();
    }
```

During the pause time, the interval is cleared (using the ID that was stored when setting the interval the last time) so new network requests are made and there are no changes in the user interface, etc. In the moment that the application is paused, we store the actual time where it was paused to use it in the onResume event handling.

```
onResume: function() {
        var onResumeTime = new Date();
        var delayTime = onResumeTime - onPauseTime;
        if(delayTime < refreshInterval){
            setTimeout(function() {
                        refreshNews();
                        intervalID = setInterval(refreshNews, refreshInterval);
                        }, refreshInterval - delayTime);
        }else{
            refreshNews();
            intervalID = setInterval(refreshNews, refreshInterval);
        }
    }
```

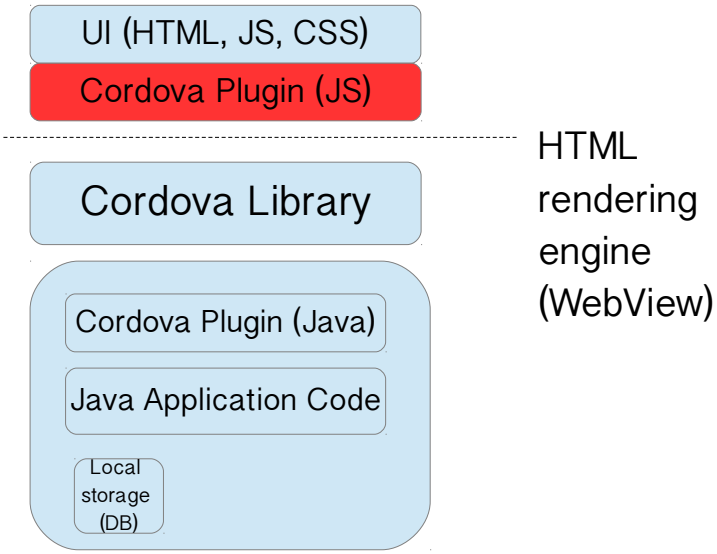
The objective for the `onResume` event in this application is to set again the timer for refreshing news (as it was cleared in the `onPause` event). The periodical news refresh should not be executed in less time than set in the `refreshInterval` variable. For this, the `delayTime` variable is defined. This variable stores the time that has passed between the time when the application was paused and the time when the application was resumed.
If there is some remaining time (the difference between the two stored times is positive) until the next time that the news should be refreshed, a timeout is set (`setTimeout` method) that will execute a new `setInterval` for refreshing news like it was set the first time, but only after the remaining time has passed.
If the remaining time is zero or less, the `refreshNews()` method is called without any delay and a new timer for refreshing news is set.

6.2. Custom plugin for news

To allow the HTML application to communicate with the native part for some custom features that are not supported directly by PhoneGap, a custom Cordova Plugin needs to be implemented. This plugin will be in charge of the communication with the news server, parsing the XML response, constructing the object model, storing the news in the database and pass the objects to the HTML application. As explained before, a Cordova plugin consists of two parts. A JavaScript interface that defines the plugin itself, and a native implementation dependent on the platform. This native implementation will be done in Java for the Android platform and in Objective-C for the iOS platform.

The first step for designing the prototype was to analyze current applications (Android and iPhone) and extract from them the classes that belonged to the financial news section. Once the classes were extracted, some changes in some classes were needed to adapt them for building the custom news plugin. This will be explained later in this chapter.

6.2.1. JavaScript plugin interface



Drawing 6: The Cordova Javascript plugin

The JavaScript interface of the plugin is defined in the *CocoNews.js* JavaScript file. The plugin itself is defined as an object so the application can call the methods just calling the `object.method`:

```
CocoNews.prototype = {
      //Here the methods that the plugin consists of
}
```

In this interface all the methods of the plugin must be defined and every defined method must call the `cordova.exec` function to communicate to the native environment. The following is an example of one of the methods included in the CocoNews plugin definition:

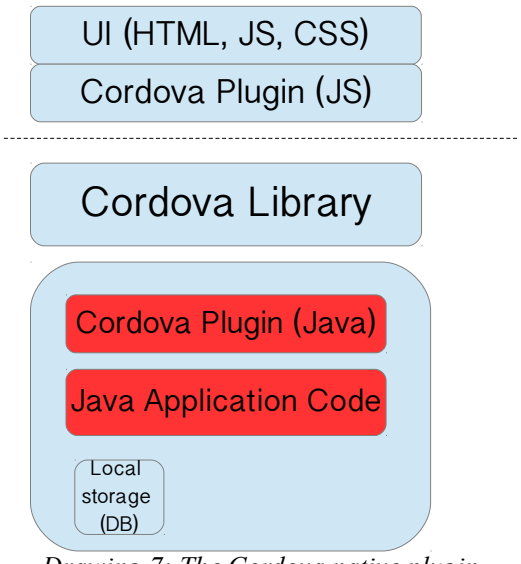
```
getNews: function(success, error){
                showSpinningWheel();
                cordovaRef.exec(success, error, 'CocoNews', 'getNews', []);
          },
```

This method calls a function defined in the *index.js* file to show a spinning wheel while requesting for news to the native part of the plugin. In case of success of the native called

method, the `success` callback function passed as an argument will be called after the execution of the method. In case of execution error, the call will go to the `error` function. The methods defined in the plugin are:

- `getNews: function(success, error)`: get the latest cached news
- `getRecentNews: function(success, error)`: request for recent news and retrieves them if found
- `loadOlderNews: function(lastIndex, success, error)`: loads older news from a given offset
- `getNewsDetails: function(idNew, success, error)`: gets the details of the new with id idNew
- `setNewsOptions: function(options, success, error)`: sets the user preferences for the filtering options
- `clearNewsList: function(success, error)`: clear the news database

6.2.2. Native implementation



Drawing 7: The Cordova native plugin

The native part of the plugin consists of a Java class where the plugin itself is defined in and more Java classes that the plugin uses to accomplish the functionalities. The module that is shown on the previous picture as “Java Application Code” refers to the extracted classes from current application once been changed and adapted to fit in this custom plugin. First let's take a look at the final architecture for the native part of the plugin and the changes made to the code of current application (from now on I will refer to it as “old code”) will follow in the explanation of this architecture.

The following picture illustrates the architecture of the news plugin:

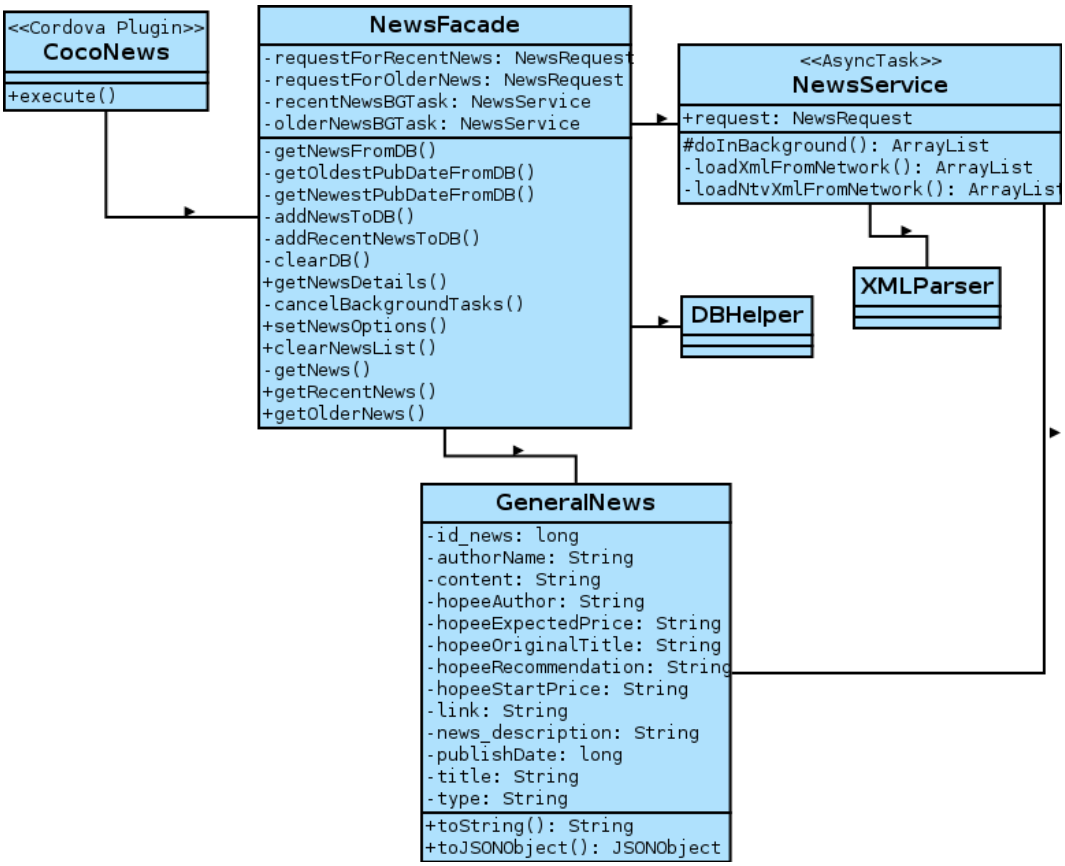


Illustration 18: Custom plugin architecture

CocoNews class

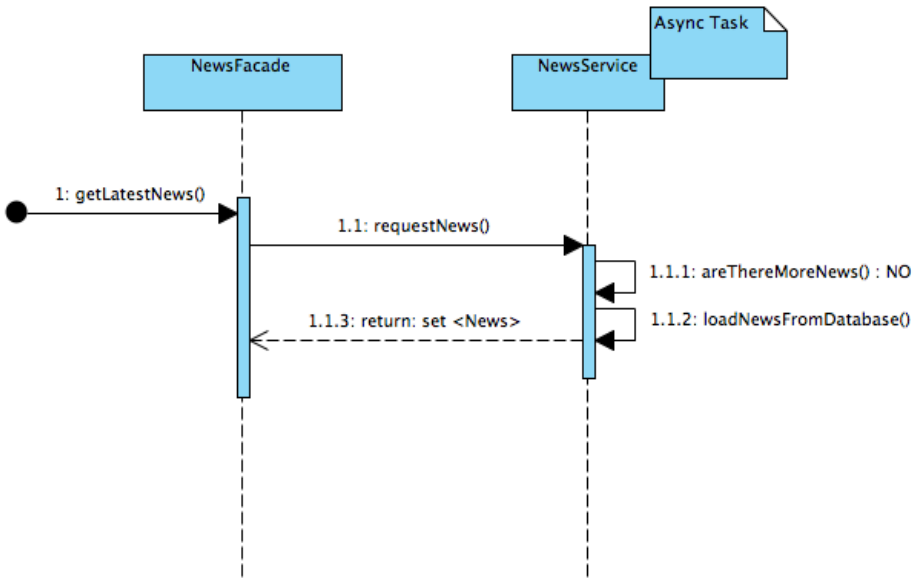
This class defines the plugin, for what it extends the `CordovaPlugin` class. The CocoNews class implements the `execute` method for the different actions that are expected. These actions match the names of the methods defined in the JavaScript interface of the plugin. The following is an example of implementation of one of the action inside the `execute` method in the CocoNews class:

```
if (action.equals("getNews")) {
            cordova.getThreadPool().execute(new Runnable() {
                public void run() {
                  getNews(callbackContext);
                }
            });
            return true;

        }
```

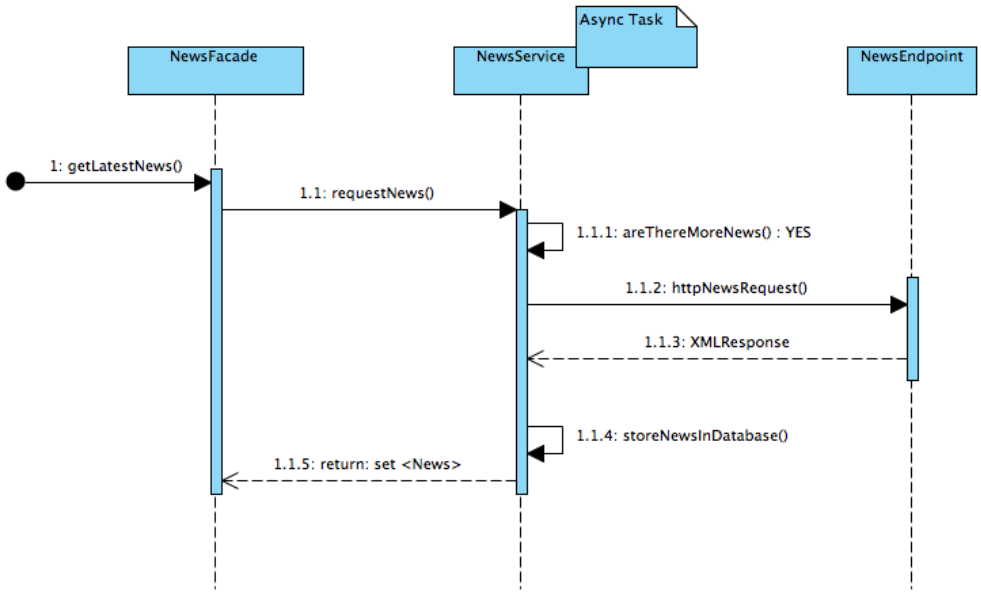
NewsFacade class

The news facade is in charge of handling the retrieval of the news in a GeneralNews format. Then it stores the news in the database and pass them to the CocoNews class that will pass them back to the HTML application using the corresponding callback function. The NewsFacade starts an asynchronous task (NewsService class) that requests for news and parses the response. The results are passed to the facade in an ArrayList format and the facade builds the object model (GeneralNews class). This news in a GeneralNews format are stored in the database making use of the DBHelper (data base helper) class. The following sequence diagrams illustrate the cases where the application starts and request for latest news. The first one shows the case that there are no more recent news, so the news shown are the cached news from the database.



Drawing 8: Sequence diagram: no recent news

The second diagram shows the case that there are more recent news. A new request is made to the server and the retrieved news are stored in the database. After that, the news are passed back to the CocoNews plugin.



Drawing 9: Sequence diagram: recent news

NewsService class

The news service class is in charge of the connections with the news server. This class sends the HTTP request to the server and gets the XML response. It extends the AsyncTask class that allows to perform background operations and publish the results to the UI thread without having to manipulate threads or handlers. For parsing the response this class makes use of the XMLParser class. In case of the need to request both servers (general news server and n-tv videos server) the NewsService waits until both requests are finished and parses and merges the results to pass them together back to the NewsFacade.

This class that I name NewsService, was formerly named BackgroundThread. This BackgroundThread class implements the interface runnable, so it can be executed by a thread. For passing the result to the corresponding class, it was made by interfaces, so one class need to implement the interface defined in the other class. For adapting this class, I changed the implementation from runnable to AsyncTask. AsyncTask allow to perform background operations and publish the results in the UI thread. The class that starts the async task (by calling the execute method) just need to call the method `get` to obtain the results. This method gives the result of the async task if it is finished or waits until it finishes and then passes the result.

XMLParser

This class parses the XML content and builds the corresponding DOM tree making use of the DocumentBuilderFactory class available in the Java framework. This class remains the same as in the old code.

DatabaseHelper

This class is the helper for dealing with the database. I use the ORMLite framework for easier dealing with the database. With the use of ORMLite, the helper is created with the class of the object that it will manipulate and it automatically provides the CRUD (create, remove, update and delete) operations. The way to define the properties of the object that will be persisted is by the use of annotations. For example, in the GeneralNews class:

```
@DatabaseField( generatedId = true, allowGeneratedIdInsert  = true)
private long id_news;
@DatabaseField
private String authorName;
@DatabaseField
private String content;
```

In this example three properties are marked as to be persisted. The first one has the options generated = true, to specify that the value for id_news in the table should be autogenerated, and allowGeneratedInsert = true. This last option specifies that even if the id_news will be autogenerated, the table should also accept manually generated id_news property. This is needed because the table will have new elements on the top (recent news) and also new elements at the bottom (older news). The insertion of new elements on the top of the table will always have a pre-established id_news value.

Data model

The data model is represented by the class GeneralNews. This class has one property for each field included in the news item retrieved from the RSS feed. Besides the getters and setters methods for every property, it also has two more methods. The method toString() returns a formatted string containing every field in the news object. And the method toJSONObject() returns the JSON object necessary to pass the data to the HTML application.

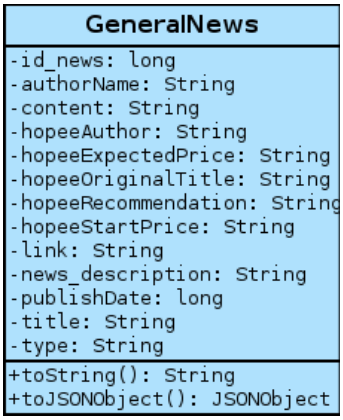
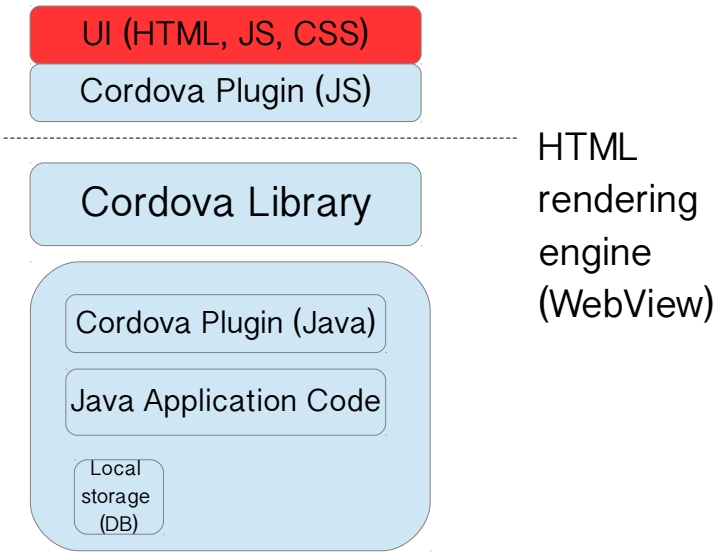


Illustration 19: General News Model

6.3. User Interface



Drawing 10: The user interface

The application consists of a single web view that contains two sections: the first section is a menu bar that remain always visible. The second section is a list of financial news.

Illustration 20: Prototype: list overview

Every item of the list shows the header (title) of the new, a short description, the source of the new and the date and time of publishing.

When a new is tapped, the corresponding section (presented as a rectangle) is expanded showing the whole content of the new. At the bottom of the expanded new, two buttons are placed to increase or decrease the font size of the news content. When the item is tapped again, the new is closed. If the user taps on a different news item when another news item is open, the previously opened news item is closed and the new one is expanded.

Illustration 21: Prototype: single new view

The user interface is built with HTML5, CSS3 and JavaScript. The content is presented as a list of news: HTML unordered list (`<ul>`) consisting of list items (`<li>`).
Both the `ul` and `li` elements are customized using CSS.
Every list item contains several section (`<div>`) elements to allocate the elements that show the needed information (author, description, date and time, source)

The first 20 news are loaded in the list and a 21st list item is added with the label “Mehr news” (More news). This list item has a tap event listener to load older news although the older news are also added when the user scrolls the list of news to the bottom.
When older news are loaded, this list element is removed, so the next 10 news can be added at the bottom of the list. Below the 10 new list items, an additional “Mehr news” list element is added as it was added before.

Every five minutes, the application requests again for recent news. This news are loaded on top of the list in chronological order (descending). If the user scrolled down the list of news, this news are loaded hidden on the top of the list of news. If the user did not scroll, so it is on the top of the list, the news are loaded on top where the latest new is the first item shown in the list of news.

I encountered a problem when dealing with the increase and decrease font size buttons.
At the beginning I placed the content and the buttons inside the same container and added the different tap events listener to each single element.
As the opened new should be closed when tapping again, and the buttons were placed in a container that was also listening to tap events (although the target action was different), when tapping in any of the buttons the tap event was propagated to the list element, causing the new to close automatically when tapping to change the font size. To solve this, I decided to add a section (`<div>` element) inside the closed new list element and another section (`<div>` element) to the opened new list element and add the tap event listener to both of them. Below the opened new container and still inside the list item I added a third container to allocate the two buttons with their own tap event listener so they don't interfere with the tap event to close the new. The result is shown in the following figures:

Closed new teaser:

Illustration 22: Closed new

Opened new item:

Illustration 24: Open new

Illustration 23: Open new

6.3.1. Tap vs. click event

I included the *tap.js* [tap13] library to deal with the tap events on touchscreen devices (smartphones, tablets). As the click event waits for approximately 300 milliseconds [dev11] before executing the target action to see if it is a single click or double click, the user experience is not the desired. For using this library you only have to include the JavaScript file in the header of your html file:

```
<script type="text/javascript" src="js/tap.js"></script>
```

For every element that you want to use the “tap” event you have to listen to the tap events instead of the onclick events. For this, you need to create a new Tap object with that element and then add a tap event listener to the element as you can see in the following example:

```
var newsPage = document.createElement('div');
myTap = new Tap(newsPage);
newsPage.addEventListener('tap', function(){showNewsDetails(this.parentNode.id)},
false);
```

6.4. Animations

There are several animations implemented in the user interface:

- open/close new
- show/hide news settings menu
- show/hide help

The animations are implemented with the CSS3 transition property.

When the user taps on a new to see the content, a CSS3 transition is applied during 0.2 seconds:

```
transition: height 0.2s
```

The first parameter is the property that the transition should be applied to and the second parameter is the duration time of the transition.

When the user taps on any of the buttons for increasing or decreasing the font size of the content, the height of the container where the new is shown is also increased or decreased following the same transition as the one shown before.

6.5. User settings – HTML5 Local storage

In the application, the user is able to choose a filtering for selecting the different news sources. This settings should persist between every use of the application. There are some options to implement this. For example:

- Store the settings in a database table
- “Shared preferences” for Android, NSUserDefaults for iOS
- HTML5 local storage in the web client

As the application must be as much portable as possible, and the cross-platform part of the application is built with web technologies, I decided to use the HTML5 local storage feature for storing the user settings. With HTML5, web pages can store data locally within the user's browser. In this way, no changes are needed to store the preferences in different platforms, as this feature is supported in every browser engine that supports HTML5. HTML5 local storage consists of a set of key-value pairs:

```
window.localStorage.setItem("newsOpt", value);
window.localStorage.getItem("newsOpt")
```

6.6. Porting to iOS

The first and main part of the project was to design and develop the prototype for the Android platform. After that, there was time to evaluate the prototype on the iOS platform reusing the whole HTML part of the application and developing the native part of the plugin for iOS using Objective-C. As with the Android project, the first step was to analyze the current iPhone application and extract the classes that belonged to the financial news section. There were less changes needed to the adapt the code for iOS than for Android.

As the HTML part of the application was already complete before starting this process and the architecture and functionality of the plugin was also defined, the effort to build the iOS application was much lower than for the Android one.

The process for porting to iOS was first, creating a new PhoneGap iOS project, using the command line tool [ios13]. Once the project was created, I copied the HTML, CSS and JavaScript files from the /assets folder of the Android project to the /assets folder of the iOS project keeping the same directory structure in both projects. Then, edit the config.xml file to map the plugin. At this step, even if the plugin was not implemented, the application was able to run on the emulator and the iOS device. It was showing only the toolbar with the buttons (that had their basic functionality) but no list of news.

For implementing the plugin I followed the same approach as on the Android project. A news facade that communicates with several components (news service, database helper).

As shown in the following pictures, the goal of having the same look and field on both applications was achieved. These screenshots are taken from an iPhone 5 running the hybrid iOS application.

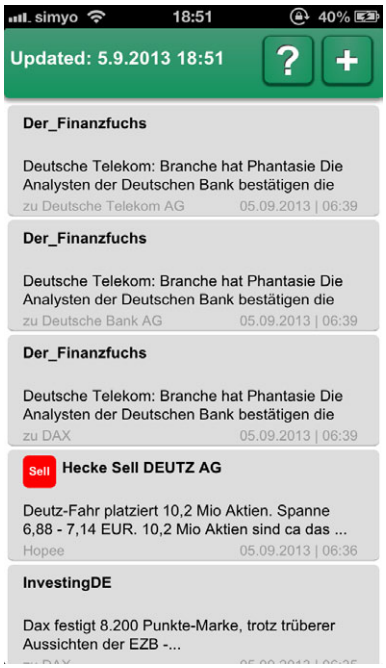


Illustration 25: iOS prototype: list overview

Illustration 26: iOS prototype: single new view

7. Conclusions

During the realization of the project I made a research of the current hybrid cross-platform frameworks available in the market, investigating the way they work, requirements, programming languages, and more to choose the one that fits best on my opinion to this project. The choice was PhoneGap. The chosen framework is abundant enough to fulfill the business requirements of the project. It was clear to understand the way it works and it was comfortable to get in touch and start programming. Another advantage is that the framework is well documented on the company's homepage and on user boards so that difficult issues during the development process could always be solved. PhoneGap is a well improved and well known framework for many developers and there are lots of resources, examples, proofs of concept, as well as already developed plugins by other programmers available on the internet.

One of the disadvantages of using this kind of frameworks is that if new devices come to the market with new native features, the framework needs to be updated before starting developing applications for those devices. For example if we think of NFC (near field communication) sensors a lot of similar frameworks compared to PhoneGap still do not provide interfaces to use these sensors. Luckily PhoneGap at current state supports most of all common used sensors but nobody knows which new sensors the device manufactors are going to provide to surprise us software developers. In general updating a hybrid framework is a process that takes time, so it will delay the future development of the application.

To evaluate the performance of the framework, before starting the prototype I made a proof of concept for Android and iOS. The proof of concept was an application consisting

of two screens, one native and one hybrid. In both screens there was a map that led to the current location of the device. It used Google Maps for the Android application and Apple Maps for the iOS. There was also the need of using a custom plugin for showing the maps on a hybrid way using PhoneGap. This plugin (called MapKit) was available on the internet, and even if it needed some changes to use it on the iOS application (the plugin was outdated) it was easy to make it work. The proof of concept was tested with some non project related users. Some of them did not detect any difference from the native to the hybrid screen, but some others detected a slightly difference in the performance. This difference is not important for this kind of business applications, but it can be a big problem on graphical rich applications.

In current prototype, the HTML, CSS and JavaScript are rendered locally. There is a possibility to make a server provide this content so the application gets automatic updates without the need of downloading a new version form the app store. But this has the disadvantage that your application may be rejected by Apple, since they do not allow applications that load and executes external code.

As a summary, I will say that the chosen framework was the right one for the prototype. The user interface developed with web technologies allows you to create 100% custom designs for your application and the bridge that PhoneGap provides for accessing native features as well as developing a custom plugin for your needs make the framework up to almost everything. Evaluating the time that took me to have the iOS application up and running once the Android application was finished, demonstrates that this cross-platform approach saves time (and consequently money) on developing for new platforms once you have at least one application finished.

This hybrid approach was good for this prototype. Next work should be trying to build a more complex hybrid cross-platform application following this process.

8. Bibliography

[HTM12] HTML
http://www.w3.org/wiki/Open_Web_Platform#HTML
Last access Aug. 2012

[HT513] What is HTML5?
http://www.w3.org/html/wiki/FAQs#What_is_HTML5.3F
Last access Aug. 2013

[W3o13] What is CSS?
http://www.w3.org/standards/webdesign/htmlcss#whatcss
Last access Aug. 2013

[jav11] What Is JavaScript?
http://www.javascripter.net/faq/whatisja.htm
Last access Aug. 2013

[jso13] Introducing JSON
http://www.json.org/
Last access Sep. 2013

[ORM13] Object-relational mapping
http://en.wikipedia.org/wiki/Object-relational_mapping
Last access Sep. 2013

[RSS12] RSS Specifications
http://www.rss-specifications.com/rss-specifications.htm
Last access Ago. 2013

[wh13] What Is RSS? RSS Explained
http://www.whatisrss.com/
Last access Aug. 2013

[Wi13a] Java
http://en.wikipedia.org/wiki/Java_%28programming_language%29
Last access Sep. 2013

[Dal13] Dalvik (software)
http://en.wikipedia.org/wiki/Dalvik_%28software%29
Last access Sep. 2013

[Wi13c] Objective-C
http://en.wikipedia.org/wiki/Objective-C
Last access Sep. 2013

[bac13] Backbone.js
http://backbonejs.org/
Last access Ago. 2013

[und13] Underscore.js
http://underscorejs.org/
Last access Ago. 2013

[pro13] Prototype.js
http://prototypejs.org/doc/latest/

[pho12] FAQs
http://phonegap.com/about/faq/
Last access Sep. 2013

[cor13] Native UI
http://developer.coronalabs.com/content/native-ui-features
Last access Ago. 2013

[cor13b] FAQ
http://www.coronalabs.com/resources/frequently-asked-questions/
Last access Aug. 2013

[lua13] What is Lua?
http://www.lua.org/about.html
Last access Aug. 2013

[cor13c] Are Corona SDK built apps - Native or Hybrid?
http://forums.coronalabs.com/topic/34257-are-corona-sdk-built-apps-native-or-hybrid/
Last access Aug. 2013

[whi13] Which Cross-Platform Framework is Right For Me?
http://floatlearning.com/2011/07/which-cross-platform-framework-is-right-for-me/
Last access Aug. 2013

[top13] Top 5 Tools for Multi-Platform Mobile App Development
http://mobiledevices.about.com/od/mobileappbasics/tp/Top-5-Tools-Multi-Platform-Mobile-App-Development.htm
Last Access May. 2013

[rho13] Rhodes
http://docs.rhomobile.com/rhodes/introduction
Last access May. 2013

[kon13] Kony
http://www.innovatech.com.tr/kony
Last access May. 2013

[kpt13] Kony Platform
http://info.konysolutions.com/rs/konysolutions/images/DS-Kony-Platform.pdf
Last access Jun. 2013

[pho13] Overview
http://docs.phonegap.com/en/2.5.0/guide_overview_index.md.html#Overview
Last access Aug. 2013

[plu13] Plugin Development Guide
http://docs.phonegap.com/en/2.5.0/guide_plugin-development_index.md.html#Plugin%20Development%20Guide
Last access Aug.2013

[pla13] Developing a Plugin on Android
http://docs.phonegap.com/en/2.5.0/guide_plugin-development_android_index.md.html#Developing%20a%20Plugin%20on%20Android
Last access Aug 2013

[pli13] Developing a Plugin on iOS
http://docs.phonegap.com/en/2.5.0/guide_plugin-development_ios_index.md.html#Developing%20a%20Plugin%20on%20iOS
Last access Aug. 2013

[tap13] Tap.js - A lightweight ‘tap’ event JavaScript plugin
http://alxgbsn.co.uk/2012/03/12/tap-js-a-lightweight-tap-event-javascript-plugin/
Last access Jul. 2013

[dev11] Creating Fast Buttons for Mobile Web Applications
https://developers.google.com/mobile/articles/fast_buttons?hl=de-DE.
Last access Jul. 2013

[ios13] Getting Started with iOS
http://docs.phonegap.com/en/2.5.0/guide_getting-started_ios_index.md.html#Getting%20Started%20with%20iOS
Last access Aug. 2013