Comparison between Native and Cross-Platform Apps
Abstract
The primary purpose of this study is to determine in which technology we have to develop an application depending on the features that we would like to include, in order to deliver the best value for a good price to the customers. Consequently, in this research we have described the capabilities, performance and limitations that we have found while using the different technologies. The empirical part of this study was conducted in the first semester of 2014/2015 at the Linnaeus University in Växjö (Sweden), supported by Softwerk Company.
In conclusion, the thesis shows that the user experience with native apps is always better than using the web-based technologies, especially using maps, although the time and effort spent to develop them is higher. Cross-platform solutions can be very useful for simple apps, and also if the developer does not have a lot of time to develop them. The problem with this last kind of applications is that the performance is less than the native ones.

Keywords
Hybrid apps, Native Apps, Mobile, HTML5, iOS, Android, PhoneGap, Xamarin, Device Features, App Performance, User experience

Thanks
The bachelor degree project was performed at the department of Computer Science at the Linnaeus University in Växjö. We would like to thank Mathias Hedenborg for his outstanding guidance and support while we were working in the project and also big greetings to our supervisor Dr. Rüdiger Lincke from the Softwerk Company for his tips, meetings, supervision, and helpful comments of the project. Thanks a lot also to our examiner Johan Hagelbäck for all the comments, revisions and feedback for this thesis.
Finally, thanks to the Autonomous University of Barcelona and Linnaeus University allowing us to do this Erasmus Exchange and this great opportunity.
Contents

1 Introduction ................................................................. 7
  1.1 Problem and Motivation ........................................ 7
  1.2 Related Work ...................................................... 7
  1.3 Goal and Criteria ............................................... 8
  1.4 Research Questions ............................................. 9
  1.5 Method ............................................................... 9
  1.6 Selecting Evaluation Criteria .................................. 9
    1.6.1 Tool Evaluation ........................................... 9
    1.6.2 Evaluation of Implementation ............................ 10
  1.7 Implementation .................................................. 10
  1.8 App Candidates ................................................ 10
    1.8.1 Gemla .......................................................... 11
    1.8.2 Strama LVN ................................................ 11
    1.8.3 Quick Votes ................................................. 11
  1.9 Outline ............................................................. 11

2 Background .............................................................. 12
  2.1 Mobile Platforms ............................................... 12
  2.2 Used Technologies .............................................. 12
  2.3 Development Methodology ...................................... 13

3 Description of the Frameworks ....................................... 14
  3.1 Xamarin ........................................................... 14
  3.2 Titanium .......................................................... 14
  3.3 PhoneGap .......................................................... 14
  3.4 Conclusion ....................................................... 15

4 Implementation .......................................................... 16
  4.1 Xamarin ........................................................... 16
    4.1.1 Installing Xamarin ......................................... 16
    4.1.2 Working in Xamarin Studio ............................... 17
    4.1.3 Problems and limitations ................................ 18
    4.1.4 Conclusions ............................................... 19
  4.2 PhoneGap .......................................................... 20
    4.2.1 Installing PhoneGap ....................................... 20
    4.2.2 Versions of PhoneGap ..................................... 20
    4.2.3 Working in PhoneGap ...................................... 22
    4.2.4 Problems with PhoneGap .................................. 23
    4.2.5 Conclusions ............................................... 24

5 Comparison between the different Apps developed ................. 25
  5.1 Gemla ............................................................... 25
    5.1.1 Gemla in Android: PhoneGap .............................. 25
    5.1.2 Gemla in iOS: PhoneGap ................................... 31
5.1.3 Gemla in Android: Xamarin ............................................. 37
5.1.4 Gemla in iOS: Xamarin ................................................. 37
5.2 Strama LVN ................................................................. 38
  5.2.1 Strama LVN in Android: PhoneGap .............................. 38
  5.2.2 Strama LVN in iOS: PhoneGap .................................... 41
5.3 QuickVotes ............................................................... 43
  5.3.1 QuickVotes in Android: PhoneGap .............................. 43
  5.3.2 QuickVotes in iOS: PhoneGap ..................................... 46

6 Evaluation ................................................................. 48
  6.1 Survey ................................................................. 48
  6.2 Questions ............................................................ 48
  6.3 Results ............................................................... 49
  6.4 Conclusion .......................................................... 52

7 Conclusion and Future Work ......................................... 53
  7.1 Conclusion ............................................................ 53
    7.1.1 Learning Curve ................................................ 56
    7.1.2 Pros and Cons ................................................ 56
  7.2 Future Work ........................................................ 57

References ................................................................. 58
  Other Useful Links for our Project ................................ 61
List of Figures

Figure 2.1: Spiral Development Methodology. Source: [9] .................................................. 13
Figure 3.1: PhoneGap Build. Source: [13] ........................................................................ 14
Figure 3.2: Google Trends Statics Frameworks. ................................................................. 15
Figure 4.1: Xamarin Studio ............................................................................................... 16
Figure 4.2: Xamarin Shared Code. Source: [19] ............................................................... 18
Figure 4.3: PhoneGap developing. Source: [25] ................................................................. 20
Figure 4.4: Versions of PhoneGap – Plugins. Source: [32] ............................................... 20
Figure 4.5: Nipple Simulator ............................................................................................. 21
Figure 4.6: Checking Internet Connexion. ......................................................................... 23
Figure 5.1: Gemla Android - Welcome screen. ................................................................. 25
Figure 5.2: Gemla Android - List of events ....................................................................... 26
Figure 5.3: Gemla Android - Map and tracks .................................................................... 27
Figure 5.4: Gemla Android - Event's information ............................................................. 28
Figure 5.5: Gemla Android: Event's location. ................................................................. 29
Figure 5.6: Gemla Android - Resizing bug ......................................................................... 30
Figure 5.7: Gemla iOS: SplashScreen. ................................................................................ 31
Figure 5.8: Gemla iOS: List of events ................................................................................ 32
Figure 5.9: Gemla iOS - Map and tracks .......................................................................... 33
Figure 5.10: Gemla iOS: Markers in the map. ................................................................. 34
Figure 5.11: Gemla iOS - Event's information. ................................................................. 35
Figure 5.12: Gemla iOS - Event's location. ...................................................................... 36
Figure 5.13: Gemla Android Xamarin - Map and markers. ........................................... 37
Figure 5.14: Strama LVN Android - Information screen. ................................................ 39
Figure 5.15: Strama LVN Android - Dropdown menu. .................................................... 40
Figure 5.16: Strama LVN iOS - Information screen. ......................................................... 41
Figure 5.17: Strama LVN iOS - Dropdown menu. ............................................................. 42
Figure 5.18: QuickVotes Android - Login page .................................................................. 43
Figure 5.19: QuickVotes Android - Login page keyboard ............................................... 44
Figure 5.20: QuickVotes Android - Barcode scanner. ...................................................... 45
Figure 5.21: QuickVotes iOS - Login screen. ................................................................. 46
Figure 5.22: QuickVotes iOS: barcode scanner. ............................................................... 47
Figure 6.1: Results from question 1 ................................................................................ 49
Figure 6.2: Results from question 2 ................................................................................ 50
Figure 6.3: Results from question 2 – split by age ........................................................... 50
Figure 6.4: Results from question 3 ................................................................................. 51
Figure 6.5: Results from question 3 split by age ............................................................. 51
Figure 6.6: Results from question 4 ................................................................................ 52

List of Tables

Table 2.1: Mobile platforms comparative table .............................................................. 12
Table 4.1: PhoneGap features. Source: [39] ................................................................. 24
Table 7.1: Final Recommendations. .............................................................................. 55
List of Abbreviations

HTML: Hypertext Markup Language
CSS: Cascading Style Sheets
UI: User Interface
API: Application Programming Interface
QR: Quick Response code
JS: JavaScript
JSON: JavaScript Object Notation
XML: eXtensible Markup Language
1 Introduction

Softwerk AB is an IT company in Växjö, among other things they work with app development. Since 2010, they created over 50 apps mainly on iPhone, Android and Windows Phone. Often the same app has to be available on at least two platforms (Android and iOS are basically sharing the market).

Their customers have demanding projects with regard to requirements but also costs. To stay competitive it is important to be able to deliver the best value for a good price.

1.1 Problem and Motivation

Developing the same app for different platforms natively (iOS, Android, Windows Phone, etc.) costs a lot of time and resources. One has to develop not only the UI but also logic and data layer on each platform resulting in “double work”. Therefore, to stay competitive, reducing development and maintenance effort as well as the code specific to each platform, and in the best case write once run everywhere seems important.

There are a number of multi-platform app creation tools available (web based technologies or similar) which promise to allow writing an app and run it on different platforms, commonly known ones include PhoneGap, Titanium, Xamarin, etc.

But the existing tools may have limitations and thus might be not be suitable for solving all possible app projects, since they may not support all device specific features. Furthermore, user experience may be important and should be avoided to have, e.g., Android look and feeling on iOS or vice versa, violating the user interfaces guidelines for each platform.

Therefore the problem addressed by this thesis is to compare a selection of available tools to answer the question if and when they are a good choice for developing apps for multiple platforms.

Choosing a wrong approach can be costly, e.g., starting off with a web based approach (PhoneGap) and then needing to change to a native approach due to some requirement, low performance, or unsatisfied usability needs, and in consequence wasting money and time. But also in the contrary, it might be that one develops two or more native versions of the same app, while a simple web based app running on different platforms might have been sufficient and more cost and time efficient.

1.2 Related Work

Several publications study and investigate different ways for developing mobile applications can be found when the literature. Some of this publications are the base for this research and have been very useful for a deeper understanding of the possibilities that can be used to develop cross-platform applications.

In the article from Gartner company “Gartner Says by 2016, More Than 50 Percent of Mobile Apps Deployed Will be Hybrid”, the authors outline predictions about the use of web technologies in 2016 [1]. They predict that HTML5 development is going to be used for more than the half of the deployed mobile applications which will allow to reuse the code for the different platforms.

In the article “Mobile Application Development: Web vs. Native”, by Andre Charland and Brian Leroux, a good discussion about the differences between developing a mobile applications natively or using web-based frameworks is found [2]. The required skills, the performance or the user experience are some of the points that are discussed and described.
In the thesis “Evaluation and implementation of a hybrid crossplatform application combining mobile web technologies and native device functions”, done by Clara Anel in Sept. 2013, a good review of cross-platforms applications is done [3]. The article describes the process of developing a PhoneGap application and evaluates it in comparison with native apps. The prototypes are both evaluated from a technical perspective and their look and feel.

In the article “Comparison of Cross-Platform Mobile Development Tools” done by Manuel Palmieri, Inderjeet Singh and Antonio Cicchetti, a pragmatic comparison among four very popular cross platform tools were presented, which are Rhodes, PhoneGap, DragonRad and MoSync [4]. Their main focus is to provide an overview of the available application programming interfaces, programming languages, supported mobile operating systems, licences, and integrated development environment in order to support developers to choose correctly a good framework with respect to their needs and constraints by comparing the main supported native features.

The above articles have been very useful as guidelines to follow while doing our research, but none of them present clear guidelines to help a company decide between native development or using a framework. For this reason, our main motivation is to provide these guidelines in order to help developers and companies in choosing a technology to use.

1.3 Goal and Criteria

For answering the problem addressed by this thesis we want to reach the following goals:

1. Create an overview and describe existing multi-platform app creation tools to be able to evaluate them. This goal is met if we discussed at least 2 different platforms or tools, which can generate apps for more than one mobile platform.
2. Select a number of criteria for evaluating differences of these discussed tools or platforms. The goal is met if we have suitable criteria based on real world projects, which are suitable for testing the selected platforms/tools against.
3. We want to evaluate how easy/good it is to create common applications using different tools. In particular we want to know:
   a. How good is the development process (limitations/advantages).
   b. How good is the result (usability, performance, following guidelines). In particular when looking at how the app follows the UI guidelines on different platforms.
   This goal is met if we implement at least two different apps on two different platforms demonstrating a variety of generally used features. We need to specify criteria for evaluating the development process as well as for comparing the resulting apps with their native counterparts, and then perform an evaluation using the selected criteria.
4. The final goal is to document the experiences (good and bad) in developing the hybrid apps and to provide a list of recommendations allowing one to select based on the apps requirements, which approach to choose. This goal is met if a matrix or other decision tree can be formulated allowing one to choose based on the project requirements the appropriate technology.
1.4 Research Questions

The main purpose of the thesis is to explore and check the different cross-platform frameworks in the market nowadays and how their performance is compared to the native development. To help in our research Softwerk’s apps will be used, but the results can be applied to a more general purpose. Considering this general purpose, the following research questions will be investigated:

RQ1: *How does the performance of the apps change when developed in a cross-platform framework compared to native development?*
With RQ1, we want to figure out how different is the performance depending on the framework used and in comparison with native development.

RQ2: *How does the look and feel of the apps change when developed in a cross-platform framework compared to native development?*
With RQ2, we want to analyse if the user interface can be similar or equal using frameworks or natively.

RQ3: *Does developing natively save time or resources to companies?*
With RQ3, we want to have a more business approach when having to choose between using frameworks or not.

1.5 Method

The list below contains the steps that will be studied and answered along the report:

1. Select an app already available in a native version (reasonable complex), at least on iOS, Android, and maybe Windows Phone.
2. Select suitable frameworks that can run on multiple platforms using the same source code.
3. Select the core important functional requirements being supported.
4. Select other constraints, e.g., time to develop using tool, how to debug, how to run on different devices, upload to apps store, organize projects/code, documentation, compatibility, guidelines, community help, …
5. Integrate with 3rd party components.
6. Combination with native components.
7. Support for tablets.
8. Compare resulting apps, look and feel, performance, load time, stability, etc. benchmark
9. Is the tool free? Documentation on the website? Good marketing? What types of apps do they usually produce? What does the people say? Specific for small/big apps?

1.6 Selecting Evaluation Criteria

During the project, the next list was a guideline to follow for evaluating the used tools and the implemented features.

1.6.1 Tool Evaluation

When evaluating and selecting tools to be used in the project we have used the following criteria:

1.6.1.1 Features / API Support

The next items are some of the features and tools that are evaluated:

- Location: Try to geo-locate the user using GPS
• Maps: Possibility to insert a map into the app or use Google Maps, and markers with data.
• Importing online data: Downloading data from a server

1.6.1.2 Development

It would also be interesting to know something about the development and the available features in all technologies:
• Build and Deploy
• Testing and debugging the application in simulators/real devices
• Documentation that can be found online
• Support for latest API levels and devices
• Price for the IDE’s used or external libraries

1.6.1.3 UI

The following details about the user interface is also taken into consideration:
• Native look and Feel of the different screens and features
• Interaction with the apps
• Responsive (different screen size)
• Check if the App is Efficient and it doesn’t use a lot of memory
• Response time of downloading information from the server or starting the App.

1.6.2 Evaluation of Implementation

The possibility of implementing the following features are also of interest:
• Network connection to make sure the data is up to date.
• Local database/offline features.
• Map: Possibility to insert a map into the app or linking to an external app.
• GPS: How to know the exact location of the device. (Accuracy)
• Take image with the camera and evaluate QR codes
• Bluetooth and wireless communication
• Search/DB queries in SQLite or external databases

1.7 Implementation

This are the selected frameworks that will be discussed and used for the implementation of the Apps related with this thesis. They are explained in more details in Section 3. There are many more frameworks available, but these seem to be the most popular and mature ones. Also the authors of the thesis are best familiar with the used technologies (HTML/CSS/JavaScript and C#).

1. PhoneGap: Very commonly used framework, it is implemented using HTML, CSS and JS.
2. Xamarin: Is more close to be native, but knowledge in C# is required.

1.8 App Candidates

The chosen applications are from the company Softwerk. They have developed over 50 apps using native technologies and as they already exist, it will be easier to compare with the web-based apps that we will develop. The selected features that will be developed for each application are marked with an * below:
1.8.1 Gemla
This App shows the user information on accommodation, shops, cultural sites, etc, in Gemla

- Welcome screen*
  This is the SplashScreen that appears when the app is started.
- Menu in the top*
  The menu has buttons for changing which section is visible.
- Kalender - List of activities*
  In this screen there is list of all the events with description and date.
- Karta - Map - Interaction with the markers - Link to Google Maps*
- Kategorier
  It allows us to choose between the different categories of events.
- Slingor (Tracks)*
  It shows some track that the user could walk/run/bike around Gemla.
- Resmål
  Here we can find a list of places to visit in Gemla
- Information page
  A page with basic Information about the App

1.8.2 Strama LVN
Guidelines for antimicrobial treatment in Västernorrland County Council intended to be a support for doctors.

- Lateral menu*: Where the type of patient can be chosen.
- List: List with the different kinds of patients
- Dropdown menu: List divided by sections depending on the type of illness.
- Select items: Selecting an illness will show us its description

1.8.3 Quick Votes

- Log in: Log in into the voting system.
- Scan QR code*: Scanning the QR code and be redirected to the website.

1.9 Outline
Section 2 will describe the background of the project, the state of art, the technologies used and the methodology that will be used to work with this project. In Section 3, all the frameworks will be described and detailed and in Section 4 show our experiences of working with this frameworks and the limitations that we found. In Section 5, the three apps developed will be compared with their respective native apps. After the comparisons, we will evaluate the results of the survey done and the conclusions drawn from it in Section 6. Finally, in Section 7 we will argue the final conclusion after all the work done, we will answer the research questions and we will list other work that has to be done in the future.
2 Background

In this section we will discuss the actual state of art of the mobile platforms used around the world, and also how we have worked during the developing process along the thesis.

2.1 Mobile Platforms

Today the two most used platforms are Android and iOS, although Windows Phone is also gaining market share. Is difficult to find the exact percentages of usage (depends often on countries/regions) from searching the Internet, but as we can see in the next table, extracted from IDC, Android wins but the other platforms are also important. While developing an App, all of them would have to be in the point of view (if a large audience are the target, and not a closed user group, e.g., a company just having BlackBerry).

<table>
<thead>
<tr>
<th>Period</th>
<th>Android</th>
<th>iOS</th>
<th>Windows Phone</th>
<th>BlackBerry OS</th>
<th>Others</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q3 2014</td>
<td>84.4%</td>
<td>11.7%</td>
<td>2.9%</td>
<td>0.5%</td>
<td>0.6%</td>
</tr>
<tr>
<td>Q3 2013</td>
<td>81.2%</td>
<td>12.8%</td>
<td>3.6%</td>
<td>1.7%</td>
<td>0.5%</td>
</tr>
<tr>
<td>Q3 2012</td>
<td>74.9%</td>
<td>14.4%</td>
<td>2.0%</td>
<td>4.1%</td>
<td>4.5%</td>
</tr>
<tr>
<td>Q3 2011</td>
<td>57.4%</td>
<td>13.8%</td>
<td>1.2%</td>
<td>9.6%</td>
<td>18.0%</td>
</tr>
</tbody>
</table>

Source: IDC, 2014 Q3

Table 2.1: Mobile platforms comparative table.

The information in Table 2.1 was taken from [5], [6] and [7]. Developing separate native apps for all platforms costs a lot of time, resources and knowledge. Hybrid Applications, also called Cross-Platform applications, solves this. This kind of applications based on new technologies like HTML5 allows the developer to write less code, because most of this code would be reused and it will reduce the development time which saves time and money.

2.2 Used Technologies

For developing the Apps, the following resources have been used:
- MacBook Air: OS X Yosemite v10.10.2
- Eclipse Jade v8.0.2
- ADT plugin v23.0.0,
- Xcode v6.1.1
- PhoneGap v2.9, PhoneGap v3.6.3
- Xamarin Studio: v5.5.4

Devices for testing:
- Nexus 4: Android v5.0.1
- iPhone 5C: iOS v8.1.3
- Samsung S4 Android v4.4.4
- Tablet BQ Edison: Android 2.3

Programming languages:
- Xamarin: C#
- PhoneGap: HTML, CSS, JavaScript
2.3 Development Methodology

The methodology used while developing this project has been a Spiral Methodology that let us to do it in an iterative, incremental and evolutionary way.

Working with this methodology allowed us to change the requirements of the project if it was needed for satisfying the conditions and also identifying and resolving the problems we found. We also had efficient face-to-face communication, that helped us to resolve doubts and issues. As discussed in [8], this methodology is a combination of the features of both the waterfall model and the prototype model.

Figure 2.1: Spiral Development Methodology. Source: [9]

Figure 2.1 shows that the used method is divided in four parts and also that the main shape is a spiral. Each loop in this spiral represents a development phase and it has four sections that will be described now.

In the Analysis section the objectives are determined with the supervisor and the alternatives and constraints are discussed as well.

If there are risks, they have to be identified and solved in the Evaluation phase.

In the Development phase, the drafts and requirements are specified and the code is designed, integrated and tested.

After the Development, we have a new release and we can start thinking and planning the next iteration of the cycle to progress with the project.
3 Description of the Frameworks

On the Internet there is long list of frameworks that offer us to develop Mobile Hybrid Apps for more than one platform at the same time, as for example: RhoMobile, Appcelerator Titanium, Xamarin, NeoMAD, PhoneGap, MoSync, Whoop, AppInventor, Corona, Flash Builder, Sencha Touch, JQuery Mobile or IONIC.

Most of them are using HTML5, CSS and JavaScript, but others will need knowledge of C#, JQuery, Angular JS or Bootstrap.

Trying all of them and then having a useful matrix with the features that are better in one framework or another would be very useful, but only the most relevant nowadays are going to be discussed.

3.1 Xamarin

Xamarin is based in C# or .NET and it allows to develop apps for iOS, Android, Mac, Windows and also for Google Glass.

The Xamarin website states that it will run as a native app in all the platforms and also that they have native API access [10].

Xamarin also have a free development software called Xamarin Studio, an IDE similar to “Visual Studio” to develop your apps.

3.2 Titanium

Titanium in an open-source software development kit for cross-platform mobile development. Developing in Titanium means developing in JavaScript, for simplify the development with a single base code [11]. The developers say that about 70-80% of the code can be reused.

3.3 PhoneGap

Phonegap is most suitable for developing simple apps [12] and especially if a web version of the app already exists.

PhoneGap offers us a tool called PhoneGap Build. It is a cloud service where the developer can upload the www folder of the app done in one platform, and PhoneGap build will transform it to the other platforms. The cloud services compiles the code for different platforms.

To use PhoneGap Build it’s necessary to have an Adobe ID and the code must also be uploaded and public in Github. The free version has some limitations and for iOS it’s necessary to also upload the certificate (p12) file and the provisioning profile, as in Xcode. In Figure 3.1 a sketch of how it works is represented.

![Figure 3.1: PhoneGap Build. Source: [13]](image-url)
3.4 Conclusion

We have realized that Titanium and PhoneGap are really similar and also we have found the main differences between PhoneGap and Xamarin [14]. Xamarin is supposed to be faster and more robust to handle heavy applications, because it is more closer to be native and it does not have a built-in browser, although PhoneGap is recommended for more simple apps. As we could find more documentation about PhoneGap and also it allows us to develop for more technologies, we decided to start developing in Xamarin and PhoneGap (instead of Titanium), because we preferred to compare two different technologies. After developing in Xamarin and PhoneGap, we will do in using Titanium if the time left in our research is not a restriction.

Also, as we can see in Figure 3.2 extracted from Google Trends, Xamarin and PhoneGap are the leaders comparing with Titanium.

Figure 3.2: Google Trends Statics Frameworks.
4 Implementation

In this section we will discuss about Xamarin and PhoneGap, the two frameworks used to implement the different applications that were described in Section 1.6.

4.1 Xamarin

Xamarin is a platform to create cross-platform mobile applications. It uses the Common Language Infrastructure (CLI) and Common Language Specifications (.NET). Xamarin uses C# for sharing the code between iOS, Android and Windows. The application is developed natively for each platform but it shares in the ideal case around 75% of the code among them (mainly data and logic layer, as stated on their website.

The developer or company can save money and time developing for different platforms while allowing to develop native user interfaces, native API access and native performance.

4.1.1 Installing Xamarin

In the project we used Xamarin Studio for both Mac and Windows and while installing it, the product that will be developed (Xamarin.Android, Xamarin.iOS or MacOS) can be chosen [15]. Once Xamarin Studio is opened and the project created, it shows the solution pad in the left panel, where we can manage our solutions, as shown in Figure 4.1. Every solution is a project and in this project we should create the Shared folder, the Droid folder and the iOS folder after create a Blank solution. We have free access to Xamarin.Android, but for iOS we only have a 30 days trial. We will start using the Android Application template because it’s free and we are more used to it.

Figure 4.1: Xamarin Studio
4.1.2 Working in Xamarin Studio

4.1.2.1 Graphical interface

In Xamarin, creating and editing the graphical interface of the application it is very similar to the native way in Android Studio or Xcode. All this IDEs allow the developer to modify the user interface in a graphical way through dragging and dropping the elements into the xml (or a xml extension in Xamarin). In Xamarin, as well as in Android, the developer has to specify in the manifest the permissions needed, but in Xamarin this activities don’t have to be declared and because of this, some functionalities and customizations of the screens and the application are limited.

4.1.2.2 Slide menu with fragments

In the Gemla application, there is a menu at the top of the screen, in which you can navigate across the app, and you can also move through it with the swipe gesture. When you try to make a menu in Android, you must use fragments to do it. For doing a menu with swipe gesture in Xamarin the only existing solution that we have found is using one example in the official page of Xamarin [16]. This example use two classes to emulate the swipe menu:

- The ActionBar, work with fragments and provides the tabs.
- The ViewPager, provide a “swiping view”

In reality we can not implement these classes and after debugging the errors, we can not implement it properly. We found that this example is under development and still has errors. When you move the phone portrait to landscape mode, suddenly an error appears and the app crashes. And also, sometimes moving across the different tabs, a random error appears and makes the app crash again. These are just a few of the examples that will make Xamarin crash.

4.1.2.3 Integrating Google Maps

When you try to integrate Google Maps in Xamarin, you find a lot of examples in the official web page of Xamarin or in another pages. When we tried to implement them most of the examples would not work.

To integrate Google Maps in our project a lot of problems were encountered, but after a long search across the Internet we managed to solve the problem. The official examples provided by Xamarin were completely wrong and the experience implementing the maps caused a lot of frustration.

4.1.2.4 Downloading XML file from the server

As in many other applications, Gemla needs to download some information from the server and store it on the device as an XML file.

We used XmlTextReader class to download and parse the XML files [17].

4.1.2.5 Shared Code

The cross-platform capabilities in Xamarin are marketed as “By choosing Xamarin and keeping a few things in mind when you design and develop your mobile applications, you can realize tremendous code sharing across mobile platforms, reduce your time to market, leverage existing talent, meet customer demand for mobile access, and reduce cross-platform complexity”. More details can be found in [18].
There are two ways to share code in Xamarin:

- Portable Code Libraries
- Shared Project (Figure 4.2)

4.1.3 Problems and limitations

Disconnected from layout renderer:

- **Solution:** [20] When Xamarin Studio is installed, by default it installs the latest Android SDK, but there are compatibility problems when trying to watch the content of a Layout resource. The solution we found was to downgrade the SDK to 23.0.2v

Minimum Android version not supported by device:

- **Solution:** The device in which we are trying to test the App has an older API. To change the minimum Android version of the App we have to go to the Project options -> Build -> General -> Target

Starter free version limitations:
The free version limits us the app size to 64KB and we can not use 3rd party plugins.

- **Solution:** We used the 30 days trial version. The account was shared since every trial version can be used on two computers. There is also a free student version that can be ordered.

Amount of memory for Java:

This error appears when you add the component Google Play Services in the project, which we had to use to include Google Maps. In the Java compilation process the default memory was not enough.

- **Solution:** You need to increase the Java heap size adding this line in the .sprog file:
  
  `<JavaMaximumHeapSize>1G</JavaMaximumHeapSize>`

Default theme with a title bar:

- **Solution:** Remove the ActionBar: ActionBar.Hide(); [21]

Missing libraries for JSON Arrays:

- **Solution:** Adding JSON DLLs [22], [23].

Consuming REST Web services:

- **Solution:** Calling and parsing web services [24]
4.1.4 Conclusions

We started developing in Xamarin because it was a different language that we didn’t know and it was a challenge for us. We were expecting to learn and develop this new technology without a lot of difficulties, but the reality was that the learning curve and the complexity was bigger than expected.

In every step we encountered many obstacles. The first problems appeared when we installed and setup the environment. When the whole environment was installed, we started developing the application and we found a lot of problems in every functionality that we were trying to implement.

When we searched for solutions across the web, trying examples and following tutorials, we could not solve all the different errors that we found.

On Xamarin’s website many examples can be downloaded, but almost all of this examples contained errors in the code.

Xamarin is not exactly native. It has its own programming language and because of this, there is not a lot of online documentation and forums. Similarly to the Xamarin examples, the online documentation of the official website has also errors, or they say that is still under developing. In some cases we finally found a solution, but it took a lot of time.

Moreover, the main point of Xamarin is that it is a cross-platform, and they promise to share a lot of code in an easy way. In Xamarin developer’s website there is a separate section for the native way for iOS and the Cross Platform way.

As a summary we could say that we spent a lot of time trying to do the project in Xamarin but it would have been easier natively. In many cases we did not find any solution for the features we needed because the documentation and the online solutions are wrong. The main point of Xamarin was sharing the code with the Cross Platform but it does not work as they promise.

For all this reasons we would like to conclude that Xamarin is not a good option to develop applications, and that is why we left working with it and started with PhoneGap.
PhoneGap is an open source framework for quickly building cross-platform mobile apps using HTML5, JavaScript and CSS. In Figure 4.3 a diagram of how it works is shown.

![Diagram of PhoneGap](image)

Figure 4.3: PhoneGap developing. Source: [25]

4.2.1 Installing PhoneGap

The installation of the PhoneGap 2.x is not very difficult. For Eclipse you must install the Android SDK from the official page [26]. After this, typically for Eclipse, install the PhoneGap, Help -> Install new software, and there, put the URL of the source [27]. This will add PhoneGap to Eclipse. In the menu a new icon for PhoneGap appears. When pressing a button a popup window of the configuration of your new project will appear. Here you must specify the path of the version of your PhoneGap and your JavaScript. In JavaScript you can choose JQuery or Sencha Touch.

The last version of PhoneGap [28], JQuery [29] and Sencha Touch [30] can be downloaded from their official pages.

After these steps, the project is correctly created and you can start developing your app.

4.2.2 Versions of PhoneGap

There is not a lot of changes between the different versions of PhoneGap and installing and working with them is not very similar. The main differences are that in versions 3.x all the features that were API’s in 2.x, now are versioned platform plugins, as shown in Figure 4.4.

Until version 3, using PhoneGap is just copying the libraries into the Eclipse project, but versions later than 3 you have to use the command line and it is more complicated.

Although versions 2.x are easier to install and work perfectly, when you upload the APK to “Play Store”, a message is shown saying that there are security issues and it is recommended to upgrade it to the last version [31].

![Diagram of PhoneGap Versions](image)

Figure 4.4: Versions of PhoneGap – Plugins. Source: [32].

In versions 2.x, all the plugins are already pre-installed, but in the newest versions you have to install them though the Command Line Interface (CLI).

Upgrading PhoneGap 2.9 to 3.x is also a task that must be done with the CLI [33].
When our project was carried out, the latest version of PhoneGap was 3.6.3 which was very difficult to install. We reached it at the end and after following a lot of different tutorials. This guide was very useful but not possible to follow it without problems [34]. A summary of our solution to install Cordova in MAC, create a platform and create the app would be:

1. install node.js (nodejs.org)
2. npm -g install npm@2.0.0
3. sudo npm install -g cordova@3.6.-0.2.13
4. cordova create hello com.charlism.hello HelloWorld
5. in user$: sudo chmod -R 777 '.cordova'
6. in /usr/local/bin: sudo chmod -R 777 ‘npm’ and ‘node’ folders
7. modify the $PATH as indicated in the last link.
8. in /hello/: cordova platforms add android (and ios)
9. in Eclipse: import -> Android -> existing projects in the workspace
10. project -> properties -> resource filters -> delete exceptions

One of the worst problems we encountered was discovering which files had problems with the owner and the privileges.

The following command is used to test the app in a local server in Google Chrome:
Google Chrome: install NIPPLE extension and press ‘Enable’, as we can see in Figure 4.5.

Terminal: python -m simpleHTTPServer 8000
Google Chrome: localhost:8000

It is also possible to test the app directly in your Android device or in the Android simulator.

Figure 4.5: Nipple Simulator
4.2.3 Working in PhoneGap

PhoneGap works like a web page emulated in the device. For doing this in PhoneGap, there is a main Java Activity file that calls the index.html file located in the assets/www folder. This file is the first web page of the application and from here you can build all you application as a web page creating other HTML, JS or CSS files and also the images and extra libraries that you could need.

4.2.3.1 Graphical interface

The graphical interface in PhoneGap does not look like a native interface. The project is developed as if it was a website and the only option is to imitate the design of the different platforms using CSS. It is sometimes difficult to imitate this styles and functionalities perfectly.

The good point is that it is easy to develop and test, because the developer can test the pages in the navigators and use their tools, like “Inspect Element” in Google Chrome. Because it is not native and the objects are not created, it took more effort to mimic the design of the objects and putting them in the correct position.

4.2.3.2 Integrating Google Maps

Integrating Google Maps in PhoneGap was not difficult. We only had to follow the steps in the official page of the Google Maps API V3 [35]. In this web page we found tutorials for installing it and many examples [36]. All the examples and tutorials worked perfectly and it was very useful. Some examples would be: how to add markers or how to customize them.

4.2.3.3 Downloading a XML file from server

Downloading a XML file in PhoneGap works similarly as for webpages. To do this in JavaScript, we used the XMLHttpRequest object [37] and it worked perfectly. We could download the information and parse it without problems.

4.2.3.4 Tracks

In the implementation of the tracks we used a combination of the last two mentioned techniques. We download the file using the XMLHttpRequest object and integrated them in the existing map.
4.2.3.5 Checking Internet Connection

The plugin Network-information (org.apache.cordova.network-information) allows us to check the network state and returns us the type: Wi-Fi, 3G, 4G, None, etc. In Figure 4.6 we can see an alert message example that appears when we don’t have Internet of any type in our device [38].

<table>
<thead>
<tr>
<th>Android PhoneGap 2.9</th>
<th>iOS PhoneGap 2.9</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Android Screen" /></td>
<td><img src="image2.png" alt="iOS Screen" /></td>
</tr>
</tbody>
</table>

**Figure 4.6: Checking Internet Connexion.**

This is just checking if the device has any network connection. Note that this does not mean that it has access to Internet. It could be for example that the user has to do login first.

4.2.4 Problems with PhoneGap

When developing one application for Android, each screen has their own activity, and in the Android Manifest you can personalize things in it. For example, if you want the action bar to appear, the actions when the user interacts, the aspect, etc. In PhoneGap there is only one activity and that is why if we use some native options that will be the same in all screens. When the application starts, the first activity calls the index.html file in the folder www, and this is the only activity existing in the project.

For the Quick Votes application, we needed to use the camera and we saw on Internet that for PhoneGap 3.x there was an existing Barcode Scanner Plugin that seemed to be useful for us. At this moment we were using PhoneGap 2.9, and this is why we could not make it work as a barcode scanner. Instead we used the Camera Plugin to show that at least it is possible to use it. Also we installed PhoneGap 3.6.3 and we could use the barcode scanner plugin because this version supports this plugin.
4.2.5 Conclusions

PhoneGap is generally easy to learn and implement, and more importantly, easy to deploy in different platforms. However it also has some limitations that make it less perfect for our apps.

PhoneGap is not native and we have seen that for using some functionality it is necessary to use third party libraries and plugins.

For this reason, PhoneGap is only a good choice for developing simple applications, that are not graphically intense or where performance is not an issue. If the development team lacks human resources or time or it is a small application, it would be a good option because the app can be deployed on many mobile operating systems.

<table>
<thead>
<tr>
<th>Feature</th>
<th>iPhone 4/5</th>
<th>iPhone 5S and newer</th>
<th>Android</th>
<th>BlackBerry OS 5+</th>
<th>BlackBerry OS 6+</th>
<th>Windows Phone 8</th>
<th>Ubuntu</th>
<th>Firefox OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerometer</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Camera</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Compass</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Contacts</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>File</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Geolocation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Media</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>Network</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Notification (Alert)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Notification (Sound)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Notification (Vibration)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Storage</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 4.1: PhoneGap features. Source: [39].

Table 4.1 shows features being part of the PhoneGap, the platform supports almost all the native features of the devices.
5 Comparison between the different Apps developed

This section discusses a comparison between the different Apps using the different frameworks.

5.1 Gemla

Gemla is a town in southern Sweden. This application called also Gemla serves to know the town events and know where they are located.

5.1.1 Gemla in Android: PhoneGap

There are some difference in how each app is developed. Most importantly the native app downloads all the information and saves it local, and then every time we open the app, it checks if there is a new version of it. Every time we change screen in our app, it downloads the latest information. With PhoneGap it was also possible to implement it as natively, saving all the files in local and checking the version, but this was not done in our project due to time restrictions.

First of all, in Gemla there is a welcome screen (Splash Screen) that is shown until the app is loaded, and as we can see in Figure 5.1, there are no visible differences:

<table>
<thead>
<tr>
<th>Android Native</th>
<th>PhoneGap 2.9</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Android Native" /></td>
<td><img src="image2.png" alt="PhoneGap 2.9" /></td>
</tr>
</tbody>
</table>

Figure 5.1: Gemla Android - Welcome screen.
After the information is downloaded, the main screen shows us the calendar with the next activities and a menu in the top, as we can see in Figure 5.2. This menu in the native App is based in “Tabs”, but we were not able to do this in PhoneGap. We have simulated it with buttons but the swipe movement to move between the screens is missing. We can also see that we have created a similar design although we did not have the same URL to get the information from the API, which is why they show different events.

<table>
<thead>
<tr>
<th>Android Native</th>
<th>PhoneGap 2.9</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Gemla Android - List of events.</strong></td>
<td></td>
</tr>
<tr>
<td><img src="image1.png" alt="Android Native Screen" /></td>
<td><img src="image2.png" alt="PhoneGap 2.9 Screen" /></td>
</tr>
</tbody>
</table>

![Figure 5.2: Gemla Android - List of events.](image3.png)
In Figure 5.3 we can see how in the map section we were able to create a similar design, import all the markers and tracks and being able to interact with the markers [40].

Moving the view in the screen, we can realize that the feeling of the movement is much better in native than in PhoneGap, although it was possible to do this and also zoom in PhoneGap.

Every time the map is loaded, the application downloads a XML file from the server, with the list of all the events and placed on the map. Each event has its own information, and when selected by a user, the app shows the information of the event downloading it from the server.

It works similarly for the markers, but each track downloads a file with all the information.

<table>
<thead>
<tr>
<th>Android Native</th>
<th>PhoneGap 2.9</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Map and tracks" /></td>
<td><img src="image2.png" alt="Map and tracks" /></td>
</tr>
</tbody>
</table>

Figure 5.3: Gemla Android - Map and tracks.
Once we click the markers, the screen with all the details is opened. We can see that it almost has the same design, except for the status bar. We did not implement the status bar because we wanted to show that if we put a Status Bar in our application, it will be shown in all the screens, and using PhoneGap all the screens are inside the same activity. If we do it with native Android, we can choose if we want the status bar or not for each activity.

Note that in Figure 5.4 the transparency is missing in the PhoneGap original title bar.

If we click in Show the way (“Visa vägen”), the App Google Maps will open and will show us the best way to go to that place. This occurs in the native app. In PhoneGap all that you can do is only is call the sentence "geo:<lat>,<lon>", and the device will open it with the default maps application. In the native way it is opened with the best route, which is not possible to do in PhoneGap.
When we click in Show on map (“Visa på karta”), we include a view of Google Maps inside the application and we can see in Figure 5.5 that the result is nearly similar. The differences are just details that are easy to solve with CSS.

<table>
<thead>
<tr>
<th>Android Native</th>
<th>PhoneGap 2.9</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Android Native" /></td>
<td><img src="image" alt="PhoneGap 2.9" /></td>
</tr>
</tbody>
</table>

Figure 5.5: Gemla Android: Event's location.
The main differences that we have found are in the map, and apart from the feeling in the movement, when we zoom the markers does not resize until you take your fingers out of the screen. You can see this issue in Figure 5.6. This could be slightly uncomfortable for the user. This also happens with PhoneGap 3.6.3.

Figure 5.6: Gemla Android - Resizing bug.
5.1.2 Gemla in iOS: PhoneGap

The iOS app is very similar except the design part. We can see the main differences in Figure 5.7. We used the same code for Android as for iOS. Almost all the code could be reused and we only had to change some details.

The logic of the system is the same. We download the information in every screen, and this is why it is a little slower than the native one. An important change from Android to iPhone is the position of the menu. In Android it was fixed in the top and in iOS at the bottom. Also in the top of the iOS App we have to add a fake title header to each screen.

We also had to change the DisallowOverscroll preference in the config.xml to solve a problem with the vertical scrolling; the whole application was moving [41].

<table>
<thead>
<tr>
<th>iOS Native</th>
<th>PhoneGap 2.9</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="iOS Native" /></td>
<td><img src="image2.png" alt="PhoneGap 2.9" /></td>
</tr>
</tbody>
</table>

Figure 5.7: Gemla iOS: SplashScreen.
We had to change the colour of the status bar (service, time, battery) to the same green as the native app to make it similar, although we have not found a way to change the colour of the letters in PhoneGap. In the main screen we can see the list of the activities. It is the same than in the Android app in our case, but in the native app they are using different URL’s than in Android and iPhone. The iPhone URL returned a 403 Forbidden Access, as shown in Figure 5.8, and in the official application it was also not working. At the time Google had just recently disabled API v2 and were deploying API v3, which had a bit different URL. Meanwhile we used the same URL as in Android.

<table>
<thead>
<tr>
<th>iOS Native</th>
<th>PhoneGap 2.9</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="iOS Native" /></td>
<td><img src="image" alt="PhoneGap 2.9" /></td>
</tr>
</tbody>
</table>

Figure 5.8: Gemla iOS: List of events.
As can be seen in Figure 5.9, we also had to change the position of the menu. It was not possible to do the transparent status bar.

The feeling when moving the map with PhoneGap 3.6 was similar. It is slower and the icons does not resize as in the native app.

<table>
<thead>
<tr>
<th>iOS Native</th>
<th>PhoneGap 2.9</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="iOS Native Map" /></td>
<td><img src="image2.png" alt="PhoneGap 2.9 Map" /></td>
</tr>
</tbody>
</table>

Figure 5.9: Gemla iOS - Map and tracks.
When we click on a marker in iOS, the appearing box is different than in Android. The differences are shown in Figure 5.10. For this reason we had to do some changes in the code.

Figure 5.10: Gemla iOS: Markers in the map.
Figure 5.11 shows the screen with most differences between Android and iOS. The design and some parts of the code had to be changed because the displayed information was not the same.

<table>
<thead>
<tr>
<th>iOS Native</th>
<th>PhoneGap 2.9</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="iOS Native Screen" /></td>
<td><img src="image2" alt="PhoneGap 2.9 Screen" /></td>
</tr>
</tbody>
</table>

Fullskalig sporthall lämplig för de allra flesta inomhusporter.

Bokas hos Växjö kommun Pia Carlsson 0470 41077
e-post: pia.carlsson@vaxjo.se

Fullskalig sporthall lämplig för de allra flesta inomhusporter.

Bokas hos Växjö kommun Pia Carlsson 0470 41077
In Figure 5.12 we can see the Google Maps screen included in the application, and a box in the marker with all the information. We also had to change the code here because it is different than in Android.

<table>
<thead>
<tr>
<th>iOS Native</th>
<th>PhoneGap 2.9</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="iOS Native" /></td>
<td><img src="image2" alt="PhoneGap 2.9" /></td>
</tr>
</tbody>
</table>

Figure 5.12: Gemla iOS - Event's location.
5.1.3 Gemla in Android: Xamarin

It was impossible for us to develop the app in Xamarin (Xamarin Studio 5.5), because all the online documentation and examples are wrong. Because of this, we spend lots of hours trying to solve problems and some of them were finally impossible to solve.

We integrated Google Maps into the Xamarin app and as we can see in Figure 5.13 this part is nearly identical. Since the official examples and online documentation had lots of errors developing this part was very time consuming. We were finally able to solve the problems but it took a lot of extra time. It was impossible to do the action when the users press in the title of the event. Because of this, the interaction of the map is limited to move around on it, but the user cannot go to the page of each event.

Furthermore, in Figure 5.13 we can also see the menu in the top of the application. We could add icons, resize them, but we cannot change the colours of the background and the blue line.

In the native app the user can navigate through the screens making a swipe gesture. We only found one example on the Internet for do this, and this example does not works very well because it is under development and had several bugs.

<table>
<thead>
<tr>
<th>Android Native</th>
<th>Xamarin</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Android Native" /></td>
<td><img src="image2.png" alt="Xamarin" /></td>
</tr>
</tbody>
</table>

Figure 5.13: Gemla Android Xamarin - Map and markers.

5.1.4 Gemla in iOS: Xamarin

After the experience gained developing Gemla for Android and seeing the problems we had, we decided to leave Xamarin and start with PhoneGap.
5.2 Strama LVN

The Strama LVN app is for helping doctors by providing guidelines for antimicrobial treatment.

5.2.1 Strama LVN in Android: PhoneGap

In the Strama LVN application for Android we had to use a plugin to develop the lateral menu, but sometimes it does not behave as the native app. Also we cannot use the native status bar with it so we had to implement a fake one as we can see in Figure 5.13. With this plugin the swipe gesture is not very real, but it works fine and it was fast to develop.

<table>
<thead>
<tr>
<th>Android Native</th>
<th>PhoneGap 2.9</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Android Native" /></td>
<td><img src="image2.png" alt="PhoneGap 2.9" /></td>
</tr>
</tbody>
</table>

Figure 5.13. Strama LVN Android - Lateral menu.
The main screen of the app does not have any problems apart from the status bar. It works fluently and it is very similar to the native one. Figure 5.14 shows the main screen.

<table>
<thead>
<tr>
<th>Android Native</th>
<th>PhoneGap 2.9</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Android Native Screen" /></td>
<td><img src="image2" alt="PhoneGap 2.9 Screen" /></td>
</tr>
</tbody>
</table>

Figure 5.14: Strama LVN Android - Information screen.
The dropdown menu was implemented with JQuery Mobile, but we could also have implemented it with for example bootstrap. This is easy to do and it works almost as the native one. The differences are only in design and can easily be solved with CSS as we can see in Figure 5.15.

![Figure 5.15: Strama LVN Android - Dropdown menu.](image-url)
5.2.2 Strama LVN in iOS: PhoneGap

The Strama LVN application in iOS differs a lot from Android. As we can see in Figure 5.16 in iOS the lateral menu does not exist. Instead there is a menu in the bottom and it is also implemented in a different way. In iOS, everything is written in the same file and the buttons of the bottom just hide and show div elements, and this makes the app faster. All the information is written in the app and then it does not have to do the request to the server, download it and parse it, as is done in the native app. This also makes the app faster.

<table>
<thead>
<tr>
<th>iOS Native</th>
<th>PhoneGap 2.9</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Strama LVN" /></td>
<td><img src="image" alt="Strama LVN" /></td>
</tr>
<tr>
<td>Strama LVN</td>
<td>Strama LVN</td>
</tr>
<tr>
<td>Synpunkter kan skickas till <a href="mailto:smittskyddsenheten@lvn.se">smittskyddsenheten@lvn.se</a></td>
<td>Synpunkter kan skickas till <a href="mailto:smittskyddsenheten@lvn.se">smittskyddsenheten@lvn.se</a> Strama LVN</td>
</tr>
</tbody>
</table>

Figure 5.16: Strama LVN iOS - Information screen.
The dropdown menu in PhoneGap is exactly the same as in Android except for the design, as is shown in Figure 5.17. These changes are easily made with CSS.

<table>
<thead>
<tr>
<th>iOS Native</th>
<th>PhoneGap 2.9</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Öppenvård</strong></td>
<td><strong>Strama LVN</strong></td>
</tr>
<tr>
<td>Hud- och mjukdelsinfektioner</td>
<td>Hud- och mjukdelsinfektioner</td>
</tr>
<tr>
<td>Luftvägsinfektioner</td>
<td>Luftvägsinfektioner</td>
</tr>
<tr>
<td><strong>Urinvägsinfektioner</strong></td>
<td><strong>Urinvägsinfektioner</strong></td>
</tr>
<tr>
<td>Asymptomatisk bakteriuri</td>
<td>Asymptomatisk bakteriuri</td>
</tr>
<tr>
<td>Urinvägsinfektioner hos barn</td>
<td>Urinvägsinfektioner hos barn</td>
</tr>
<tr>
<td>Urinvägsinfektioner hos kvinnor</td>
<td>Urinvägsinfektioner hos kvinnor</td>
</tr>
<tr>
<td>Urinvägsinfektioner hos gravidas</td>
<td>Urinvägsinfektioner hos gravidas</td>
</tr>
<tr>
<td>Urinvägsinfektioner hos män</td>
<td>Urinvägsinfektioner hos män</td>
</tr>
<tr>
<td>KAD-byte - kvinnor och män</td>
<td>KAD-byte - kvinnor och män</td>
</tr>
</tbody>
</table>

Figure 5.17: Strama LVN iOS - Dropdown menu.
5.3 QuickVotes

With QuickVotes the user can easily create polls for smartphones. The application use the QR codes to make it easy for users to find the polls.

5.3.1 QuickVotes in Android: PhoneGap

The barcode scanner is only available for PhoneGap versions 3.x [42]. Since we used an earlier version we cannot use it, and instead we used just the camera to show that it was possible to add it.

![Android Native](image1.png)  ![PhoneGap 2.9](image2.png)

Figure 5.18: QuickVotes Android - Login page.
In Figure 5.18 and Figure 5.19 we can see the main screen with the login form, that is not functional and also the button to scan the code. At the bottom there is a link to their website to open it in an external browser. As in the native app, when the keyboard is out, the styles are different just to fit all the information.

<table>
<thead>
<tr>
<th>Android Native</th>
<th>PhoneGap 2.9</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Android Native Screen" /></td>
<td><img src="image2.png" alt="PhoneGap 2.9 Screen" /></td>
</tr>
</tbody>
</table>

Figure 5.19: QuickVotes Android - Login page keyboard.
We included the native camera, but PhoneGap did not allow us to add an image over the camera as in the native app. On the other hand we can use all the options of the camera app like it was the native camera app. We can see how the camera looks like in Figure 5.20.

<table>
<thead>
<tr>
<th>Android Native</th>
<th>PhoneGap 2.9</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Android Native" /></td>
<td><img src="image2.png" alt="PhoneGap 2.9" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PhoneGap 3.6.3</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image3.png" alt="PhoneGap 3.6.3" /></td>
</tr>
</tbody>
</table>

Figure 5.20: QuickVotes Android - Barcode scanner.
5.3.2 QuickVotes in iOS: PhoneGap

Figure 5.21 shows how the user interface in the app QuickVotes in iOS. Note that it is very different than in Android. This means lots of changes had to be made when using the Android code for iOS. The main advantage of using PhoneGap is being able to reuse as much code as possible, but for this section we had to do lots of changes. The changes are not important since they are mostly absolute style, but we preferred to focus on improving and testing other parts.

![QuickVotes iOS and PhoneGap 2.9](image)

Figure 5.21: QuickVotes iOS - Login screen.
The functionalities are the same in both apps and they only differ in the styles as shown in Figure 5.22. The Scan QR code button will open the camera and allows the user to capture photos.

The camera is the same as the native and has the same options.

![iOS Native](image1.png) ![PhoneGap 2.9](image2.png)  
![PhoneGap 3.6.3](image3.png)

Figure 5.22: QuickVotes iOS: barcode scanner.
6 Evaluation

To have an opinion on the look and feel of the apps we have conducted a research. We only had our own opinions and they were not enough or maybe we were forgetting or not focusing in everything. That is why we decided to do a survey with the goal of receiving more feedback and having different opinions from people outside the project. The test group consisted of international students and both the Android and iOS apps were tested.

6.1 Survey

Ten people from IT and no IT circles conducted the survey. Time restrictions prevented us from including more participants in the study. This is something that could be improved in future work.

The biggest difference is that IT people understood and knew which apps that were hybrid and which were native. The IT people gave us their opinions with more details and own experiences. Along the evaluation of the survey we thought that the difference of the age (older or younger than 30 years old) was more important because of their incomes and different life style. Usually, older people have more resources and money than the younger. This means that in some cases the older people are more willing to pay.

6.2 Questions

In order to know if people realize the differences between both applications, the native and the PhoneGap, we have done the following questionnaire that was handed out to the participants in the study. In addition, one of the main objectives of our project is to save time and resources for companies. One the other hand, sometimes this could as a side effect lead to decrease software quality. For this reason, we want to know if people will be able to pay more or not for the native than the PhoneGap application.

Below the questions used in the questionnaire are shown. The purpose of the questionnaire was to obtain more information for our research.

1. Do you feel comfortable in both applications?
   A: Yes.
   B: No, only in the native.
   C: No, only in the PhoneGap.
   D: In neither.

2. How annoying is the resize of the markers?
   A: Don't care.
   B: A little bit.
   C: Quite.
   D: Very uncomfortable.

3. If you had to pay for these applications, would you pay more for the native one?
   A: Yes.
   B: No.

4. Would you feel uncomfortable, if you had paid for the PhoneGap version?
   A: Yes.
   B: No.
6.3 Results
This section shows the different questions and a graph from the participants. An short opinion can also be found under the graphs.

Do you feel comfortable in both applications?

- A: Yes
- B: No, only in the native.
- C: No, only in the PhoneGap.
- D: In neither.

Figure 6.1: Results from question 1.

Figure 6.1 shows how comfortable the participants felt in both applications and some of them did not realise the differences between them. IT people said the native app felt a bit faster, but the difference was very small. They also understood that it is easier and faster to develop using these kinds of frameworks.
How annoying is the resize of the markers?

- A: Don’t care.
- B: A little bit.
- C: Quite.
- D: Very uncomfortable.

Figure 6.2: Results from question 2.

Nobody noticed the problems with the resize until they were asked about it. Then they noticed this and said that they do not like it very much, as you can see in Figure 6.2.

If we compare the results of the same question but, dividing it into the two groups of older people and younger than 30 years, we have different results as seen in Figure 6.3. We put this age limit because this is where we noticed the biggest difference. People who were under 30 years old did not focus a lot on this problem and they also seemed not worried about it when we told them. People who were older found this problem more important and described it as ugly or uncomfortable.
About paying apps, most of them are not used to it and they never pay for applications if it is not necessary. As seen in Figure 6.4, most of the people do not want to pay for the native one.

When comparing the results above (see in Figure 6.5), we can see that the people over 30 are less willing to pay. They also commented that they usually do not pay for any app and as they do not feel that there are enough differences compared to the native app they are less willing to pay.
Figure 6.6: Results from question 4.

The comments from the participants were that it does not matter how has it been developed the app to pay or not. This it is shown in Figure 6.6, only 20% of the participants would pay.

6.4 Conclusion

From the result of the survey we can conclude that everybody feels comfortable with both native and PhoneGap applications and some of them did not realize the difference. Also almost nobody noticed the resize problem until we asked about it, and once they recognized it they did not like it. Most of the participants are not used to pay for apps. To pay or not depends on if the app is good enough, and not on how it has been developed.
7 Conclusion and Future Work

In this section we summarize the results and present conclusions of the research.

7.1 Conclusion

Today, 4.5 billion people already have one or more mobile devices and this trend is forecasted to continue growing and even increasing its speed [43]. All of these devices get smarter and faster everyday and the number of apps for them is also rapidly growing. Although there are two main platforms (Android and iOS), for a complete coverage or for targeting niche markets other platforms may need to be considered. On top of this fragmentation among operating systems, there is an even more severe fragmentation regarding the OS versions.

As discussed in the introduction (Section 1.1) choosing the right approach for developing mobile apps can allow companies to save development costs, increase profits, and possible get more satisfied customers.

The problem addressed by this thesis was to compare a selection of available tools to answer the question if and when they are a good choice for developing apps for multiple platforms.

We were able to answer the research questions (see Section 1.4) using the collected data, but only comparing PhoneGap with native apps. We were not able to compare with Xamarin because we could not develop the apps with this framework, as it is explained in Section 4.1.

The collected data enabled us to answer the research questions:

**RQ1:** How does the performance of the apps change when developed in a cross-platform framework compared to native development?

In general, the performance using PhoneGap is very good, and as it is detailed in section 4.2, this framework has some limitations while looking at the high performance. Apart from this limitations, PhoneGap works perfectly and simple apps developed can be very close to native apps.

**RQ2:** How does the look and feel of the apps change when developed in a cross-platform framework compared to native development?

Using PhoneGap, the developed apps look very similar compared to the native apps. This is because PhoneGap is web based development and it uses CSS, providing to the developer a very high customization of the app. The user experience in PhoneGap differs a lot compared to native development. For basic features the user experience is almost the same, but if the developer wants to implement advanced features (like maps or menus), the app loses fluency.

**RQ3:** Does developing natively save time or resources to companies?

As explained in Section 4.2, PhoneGap is a good choice for developing simple apps. If the company wants to develop a simple app, PhoneGap can save a lot of time. The company can develop one app that works in all the platforms. This can reduce the time of development and resources needed for development, saving costs for the company.

The specified goals (see Section 1.3) were also reached while doing the research.

**Goal 1:** Create an overview and describe existing multi-platform app creation tools to be able to evaluate them. This goal is met if we discussed at least 2 different platforms or tools, which can generate apps for more than one mobile platform.

During the execution of this thesis, PhoneGap and Xamarin were discussed and evaluated. See Section 3 and in 4 for further details.

**Goal 2:** Select a number of criteria for evaluating differences of these discussed tools or platforms. The goal is met if we have suitable criteria based on real world projects, which are suitable for testing the selected platforms/tools against.
The criteria used for evaluating these frameworks were discussed in Section 1.6. On one hand we evaluate the features, the API support, the development and the user interface about the tool used, and in the other hand we also evaluate and describe our experience of implementing the features that the devices have nowadays.

**Goal 3:** We want to evaluate how easy/good it is to create common applications using different tools. In particular we want to know:

- How good is the development process (limitations/advantages).
- How good is the result (usability, performance, following guidelines). In particular when looking at how the app follows the UI guidelines on different platforms.

This goal is met if we implement at least two different apps on two different platforms demonstrating a variety of generally used features. We need to specify criteria for evaluating the development process as well as for comparing the resulting apps with their native counterparts, and then perform an evaluation using the selected criteria.

A summary of the two frameworks (Xamarin and PhoneGap) that we have used can be found below:

Actually, both of the frameworks promised the developer to ‘code once and run on at least two different platforms’, but we were only able to achieve this in PhoneGap.

The development of applications using Xamarin was not working that well for the reasons discussed in Section 4.1.3. As documented in Section 4.1, Xamarin has many problems in the online documentation and official examples. The main goal of Xamarin is to be cross-platform tool and to be able to share code between the different platforms, but we were disappointed because it did not work very well.

Xamarin also recommends working with the framework MvvmCross to be allowed to use the MVVM pattern. Xamarin Studio is an IDE similar to Eclipse which made it easy to use. We used the student version because it was free, but the commercial version is expensive to use. To debug in Xamarin we tried the app in mobile phone devices and used the console to check for errors in the logs. Since there were many problems with Xamarin and we had limited time for the project, the thesis was mainly focused on PhoneGap.

PhoneGap is a very popular open-source framework and it has been around longer than Xamarin. PhoneGap also has a lot of documentation and good support from the community. It can become a useful tool to easily create applications for all devices. PhoneGap is web-based, which makes it impossible to access all features of the mobile device, or have absolute control over all the elements. Today PhoneGap supports seven different platforms.

A good feature with PhoneGap was the debugging, where we could use the features that our browser has and other plugins to test the application. This is possible because all files are in HTML, which can be opened from the computer, although we could use our own device to debug the whole application.

The same apps that we were tested in our devices worked perfectly in the tablet. The main difference are the styles, but if the page is done in a responsive way it can be displayed perfectly too. Different CSS files can be used for different screen sizes.

PhoneGap was a good solution in the applications that did not require high performance or native and very fluid user interface. For these types of more simple apps PhoneGap can save time of development.

In this thesis we have not tried to upload the apps developed with PhoneGap to the App Store and Google Play, but based on our personal experience and the community online, there are many difficulties when doing this.

The report shows, especially in Section 5, the comparison between the developed apps (mainly in PhoneGap) and the native ones existing in Google Play and App Store.
The three developed applications are working in iOS and Android almost without any problems and, as discussed in Section 5, the different features and screens are evaluated and discussed using the selected criteria. The main differences between the PhoneGap Android Apps and PhoneGap iOS apps are mostly about the design and structure of the User Interface.

**Goal 4:** The final goal is to document the experiences (good and bad) in developing the hybrid apps and to provide a list of recommendations allowing one to select based on the apps requirements, which approach to choose. This goal is met if a matrix or other decision tree can be formulated allowing one to choose based on the project requirements the appropriate technology.

The final goal of this thesis was to provide recommendations depending on the features that the app will need. As shown in the previous sections of the report, we have tried to provide a clear and objective view of when it is useful to develop using cross-platform technologies or natively. In our project, we compared three existing applications (Gemla, Strama LVN and QuickVotes), that belong to Softwerk, because they had most of the features described at the beginning in Section 1.6.

The recommendations that is provided are formed based on subjective criteria, after we have searched and compared with the community and also our own experience in developing apps using this kind of frameworks. Table 7.1 provides a summary of our recommendations.

<table>
<thead>
<tr>
<th>Features</th>
<th>PhoneGap</th>
<th>Native</th>
</tr>
</thead>
<tbody>
<tr>
<td>High Performance</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Hardware Access</td>
<td>Limitations</td>
<td>✓</td>
</tr>
<tr>
<td>Games</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Map / GPS</td>
<td>Bad Performance</td>
<td>✓</td>
</tr>
<tr>
<td>Camera / QR scanner</td>
<td>Limitations</td>
<td>✓</td>
</tr>
<tr>
<td>Few Developing Time/Money</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Native Design</td>
<td>Good CSS</td>
<td>✓</td>
</tr>
<tr>
<td>Time Limitations (fast development)</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Budgetary Limitations</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Best user experience</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>No native interaction</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Fast maintainability</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Widgets</td>
<td>Done in Native</td>
<td>✓</td>
</tr>
<tr>
<td>Accessing device information</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Download data from server</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Upload data to a server</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Static application with local data</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

Table 7.1: Final Recommendations.

After this summary we could conclude that if the developer needs a complex and high efficiency app, the best option would be the native development since the frameworks analysed cannot reach the potential that the native way offers. The main problem with this is that a separate app needs to be developed for each platform. This is very time consuming for the developers.
7.1.1 Learning Curve

The learning curve for Xamarin is very huge and which is reported on many websites [44]. Based on our experience we cannot recommended it as a good and useful framework to use for developing mobile hybrid applications. The reason for this would be apart from the poor online documentation, the limited sharing of code across Android and iOS.

Xamarin is almost a native development using C#. The developer needs to know how to develop in C# and how Xamarin works. Besides this the developer needs to know how to develop in iOS and Android natively. As a result from this, development in Xamarin it is like a three learning curves in one.

In PhoneGap the learning curve, is much lower because PhoneGap is a web app and this requires only learning web developing, which reduces the time needed for learning the framework.

Most of the developers have a good knowledge of HTML, JS and CSS, and most of them also know Java for programming in native Android. They are very popular languages today. While discussing about mobile applications for more than one platform the biggest problem for most of developers is iOS, because they have to learn a new language (Objective C or the recently Swift) and sometimes developers are afraid of or not have enough time to learn new languages.

Learning Objective C or Swift would be a good idea if the better performance of them is needed, but in most cases it is easier to use languages that already is known and use a framework, for example PhoneGap [45][46][47].

7.1.2 Pros and Cons

Pros PhoneGap Apps:
- Faster development
- Same programming language for all the platforms
- Offers some native capabilities
- Big community of supporters and developers

Cons PhoneGap Apps:
- Worse user experience and general feeling
- Designing user interfaces not similar to native
- Scripting security issues
- Limited hardware access

Pros Native Apps:
- Best user experience (high performance and stability)
- Native design
- Fully access to the underlying mobile platform and capabilities.

Cons Native Apps:
- Different programming languages knowledge
- More time and money to develop
7.2 Future Work

As explained during the thesis, we did not have time enough to compare all frameworks, we have left out of this project the Titanium framework. Also there are a lots possible directions for future work.

On one hand we would like to test new versions of the frameworks because they are most likely more optimized and have more features than the versions used in this thesis. Additionally, a deeper understanding of each of the frameworks would be useful to be able to use all their features and to optimize the code.

Other features that we would like to investigate, try and test are:

- Cloud storage
- Calendar / address book
- Notifications
- Widgets
- Bluetooth, NFC
- Sensors information (luminosity, movement, …)
- Include Ads
- Multithreading / Memory handling
- Third party libraries (Google calendar).

Additionally the frameworks could be applied to this framework to other platforms and evaluate and compare how is the performance. We believe that they work but we do not know the limitations that these frameworks have in other platforms such as BlackBerry or Windows Phone.

Regarding the survey, it would be interesting to repeat it using more than ten participants to get more results. One of the main tasks could be to investigate in more detail what people thinks about the performance of this kind of apps. Also we got indications that people usually expect free apps and they do not seem to have very high expectations on quality or user experience. Future studies could investigate if there are differences between users of different platforms, based on age, income, etc.
References


58


Other Useful Links for our Project


