UNIVERSITY OF GOTHENBURG

Comparing HTML5 frameworks when creating multi-platform mobile views in an existing MVC4 application

Bachelor of Science Thesis in the Programme Software Engineering and Management

FREDRIK BJÖRK
NIEL MADLANI

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
Göteborg, Sweden, June 2013

Comparing HTML5 frameworks when creating multi-platform mobile views in an existing MVC4 application

Fredrik Björk
Niel Madlani

Examiner: Morgan Eriksson

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2013

Comparing HTML5 frameworks when creating multi-platform mobile views in an existing MVC4 application

Fredrik Björk
Software Engineering and Management
University of Gothenburg
tabbyboll@gmail.com

Niel Madlani
Software Engineering and Management
University of Gothenburg
niel.madlani@gmail.com

Abstract
There is an increasing number of programming languages and software development platforms on the contemporary mobile market. Developers choose to adjust to the needs for portability of their applications by using of multi-platform frameworks to reduce the need of rework when deploying to multiple platforms. The goal of this thesis is to identify the most suitable HTML5 framework for implementing cross-platform mobile views to existing MVC4 applications in a context of industrial applications at Volvo IT. The method is a case study with an implementation of a industry-representative sample solution and analysis of its performance, maintainability and simplicity. The results show that one of the frameworks – JQuery Mobile is better suited than others for implementing applications based on MVC4.

Keywords**:** *HTML5, multi-platform, mobile, MVC4, Framework*

1. INTRODUCTION
The mobile market has evolved a lot in the recent decade, with the number of apps increasing steadily for all mobile platforms [3]. Mobile technologies such as smart phones enable a new generation of consumer and business applications [16], but at the same time cause a need for developers to use an increasing number of different tools and programming languages if they want to support all platforms, such as BlackBerry, Windows Phone, Symbian, Palm OS and iOS [3]. Developers cannot make native application compatible with all mobile platforms at the same time since it would be prohibitively expensive. This claim is supported by studies by Google which states that they would not be able to afford such a diversity and suggests that web technology can solve the platform fragmentation we currently see in the markets [1].

A successful solution addressing the diversity of platforms would be an application that is compatible with iOS, Android and Windows Phone devices, providing the same functionality and not requiring expensive rework. In the case of the studied organization – Volvo IT – the solution has to be completely integrated into the MVC4 application and handle all of its functions. This solution would allow organizations that have stationary systems (alike Volvo IT), but need mobile solutions to improve the efficiency and flexibility of their workers to quickly receive mobile functionality in their existing systems.

Therefore, we address the following research question: *Which framework suits the organization best based on dedicated architectural principles?* In order to address the research question we conducted a quantitative research study at Volvo IT. In the study we developed two applications with different HTML5 frameworks, Kendo UI Mobile and JQuery Mobile. The frameworks were chosen based on requirements from Volvo IT and evaluated against the 10 architectural principles from the company (VGTA - Volvo Group Target Architecture, [6]).

The report is structured as follows: In section 2, we explain background for the technologies we have been using. In section 3, we present the method we used. In section 4, we present the results and in section 5, we discuss the results and their limitations. In Section 6 we present the conclusions.

Limitations
We will not consider the differences between native vs. web application since it is not the scope of the solicited industrial project. The article does not contain any information, which APIs from each mobile platform are supportable for each framework. We do not consider testing an application with JavaScript code since our main objective is to focus on the HTML5 frameworks.

2. THEORETICAL BACKGROUND
In this section we explain some important concepts and technologies used in this thesis.

- A framework is a reusable software platform used for development. Frameworks extend the capabilities of the base languages by adding functionality or compatibility.
- Each prototype in this article is the result of integrating code built using a framework into an existing MVC4 application in Visual Studio 2012.
- HTML5 was introduced 2012 and is a web development tool for making websites, application, videos and graphics [19]. The differences between HTML4 and HTML5 are that the old elements tags have been restructured [18].
- Visual Studio 2012 is an IDE (Integrated Development Environment [15]) developed by Microsoft primarily for use in programming C++, C# and F#.
- NuGet is a package manager in Visual Studio 2012 used for locating, downloading, installing and removing packages from a project.
- ASP.NET is a Web development model that includes necessary services for building enterprise-class Web application with a minimum of coding [14].

- C# is an object-oriented programming language that enables developers to build a variety of secure and robust applications that run on the .NET Framework [13].

- MVC4 is an alternative architectural framework for the ASP.NET web forms pattern for creating web applications, and is structured as a Model-View-Controller architecture (see Figure 1), where the views use HTML5 and the models and controllers use C# [12].

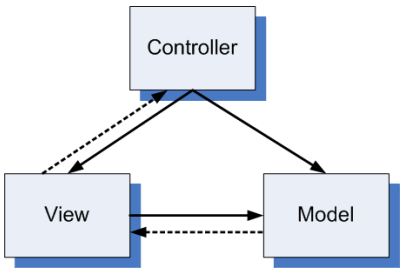

Figure 1: Model-View-Controller

2.1 The 10 architectural principles from Volvo IT

The studied company adopted 10 architectural solutions for their applications. The principles guide the design of their contemporary systems and were chosen as the evaluation criteria. In this section we describe all ten principles while later we only focus on 3 due to time limitations of the thesis.

Maintainable solutions - *Deliver maintainable solutions to Maintenance*
Maintenance is one of the most expensive and resource heavy phases and is getting more and more attention, especially in component-based projects [11]. One of the reasons maintenance is so demanding is that improving it involves so many factors, such as modifiability and testability [11].

Conformity to standards - *Drive usage of open and industry standards*
The code is easier to maintain, if it follows the established and sensible coding standards, both if it is still in-house and if the customer needs to perform the maintenance. Since everyone knows what to expect from the structure of the code.

Autonomous & loose coupling - *Flexible subsystem and granular component setup, avoiding monoliths*
"Loose-coupling applications are typically those that view the database as a data server, with knowledge-based processing used to interpret data obtained by issuing SQL queries to the database" [7].
Autonomous & loose coupling can improve the process a lot as it is important for achieving good maintainability to have components separate from each other, which is greatly beneficial to testability by allowing unit tests to be run for individual components without having to use the entire system. Furthermore, it allows for better modifiability by making the code more readable as well as allowing individual components to be modified and tested without risking errors in the rest of the system. Finally, there is also a big performance benefit in that this kind of structure allows for high response times in competitive situations [10].

Simplicity - *Clean solutions from technical, application and user perspective*
The goal of simplicity is to reduce the complexity of the system.

Usage of Agile work methods and design principles - *Use Agile system development and implementation principles*
Agile work methods are becoming more and more common and the benefits to using them are for most projects, so a framework with a structure that suits agile processes is preferable.

Strive for usage of existing services in the organization - *Whenever possible, avoid application specific infrastructure and instead use already existing services in the organization*
If the organization is already using a framework that can work for the current application, it might not be worth it to start using another framework instead even if that new framework is better, as the cost of retraining as well as the potential cost of a commercial license for the new framework may be too high.

Robust solutions - *Strive for robust solutions securing uptime*
Robustness relates to the capability of a system to handle internal and external negative situations and disturbances [4]. When performing testing of a framework we want to make sure that the framework meets the requirements, such as no hardware failures, no crashes and no subsystem malfunctions.

Performance focus from start - *Strive for good performance in solutions from the start*
A Good performance is an application or website have short response time, high throughput, and high availability among other things.

Secure solutions - *Strive for secure solutions from the start* Is the application secure enough so no one can hack into the system and wipe everything out, steal customer information or place fake orders without paying.

Good integration solution - *Follow the organization's integration policies and guidelines*
If the structure of the framework is more compatible with the integration policies of the organization, it will require less modification or retraining for successful integration solutions and reduce the frequency at which integration errors occur.

3. RESEARCH METHOD

In this thesis we used a quantitative research method [2] to test different frameworks and collect the data to measure how the different frameworks adhere to the VGTA principles.

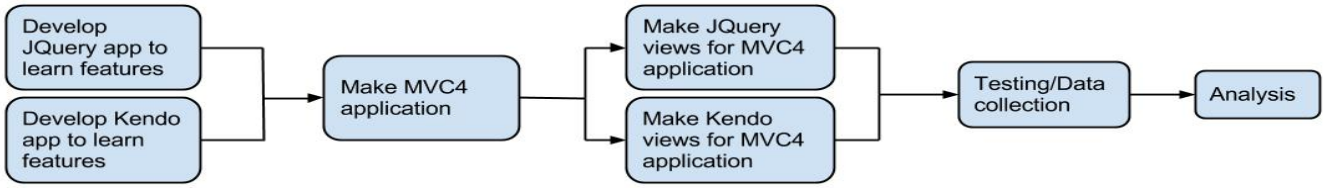

Figure 2: The Workflow

We collected quantitative data by performing an experimental multiple case study. In this study, we created a basic MVC4 application to serve as the model for the requirements of the prototypes, as well as the application the framework were integrated into. After the application was completed, we created the framework prototypes and treated each prototype as a case, including all tests performed on all specified platforms. We collected data from tests performed on the prototypes and analysis of official websites of Kendo UI Mobile, JQuery Mobile their communities.

When all data was collected and analyzed, we created an overview of the different prototypes, from which we drew a conclusions about how each prototype adhered to the VGTA principles. We have illustrated the comparison between the two prototypes using statistical diagrams, where we have measured the loading time of the website. Figure 2, shows the workflow of our research process and describing how we integrated each framework relating to the VGTA principles about maintainability and simplicity with Visual Studio 2012.

For the VGTA principles about maintainability and simplicity, we described the process of integrating each framework with Visual Studio 2012.

The frameworks need to be fully html5 based, as otherwise they cannot be integrated as MVC4 views. The frameworks also need to be flexible in what low-level languages can be used for the back-end, as C# will be used there due to MVC4. Furthermore, the frameworks need to support iOS, Android and Windows Phone. Finally, the frameworks need to have sufficient documentation for us to be able to learn how to use them well enough to implement the prototypes in the short time available to us. The frameworks that met our framework requirements were Kendo UI Mobile and JQuery Mobile.

Many other frameworks were considered, but due to our requirements they were excluded. Titanium, for example, is one of the most used mobile frameworks, but it is primarily JavaScript based and was therefore excluded.

3.1 Data collection

Due to the differences between the three VGTA principles of performance, maintainability, and simplicity, we collected our data in 7 different ways, which we have described in the following sections.

Performance

To collect our performance data, we hosted our prototypes on the same laptop with the same background processes running. A server in Dulles, Virginia, USA then ran automated load-time tests with two attached phones, a Nexus S phone running Android 2.3, and an iPhone 4 running iOS 5.1. The server in Dulles, was chosen because it provided the possibility to use the same server for simulating both Android and iOS devices. The other servers that were considered (from the website: http://www.webpagetest.org/) did not provide the possibility of using both operating systems. Choosing two different servers would make the result not comparable as there would be confounding factors for example different distances and hardware. We did not do any performance testing for Windows Phone due to lack of viable testing tools. To measure each framework for the different operating systems (OS), a website to measure the loading time [20].

For each framework with OS we did 100 test runs. Each test run resulted in one measurement, which was loading time. For each test run the server loaded the page and measured time from sending the request from until getting the full page. The page contains orders and is presented in Figure 15 and Figure 25.

From these tests we received 100 data points for each framework and platform measuring the seconds taken to load the page. Since the distance to the server was quite long we choose to collect 100 data points in order to minimize the risk of confounding factors like traffic congestion. These tests were performed between 10.00-14.00 CET on weekdays, meaning from 4.00-8.00 EST in Dulles. This meant that we avoided most traffic on the US side, but probably faced some bottlenecks on the EU to US connection due to European business traffic.

Maintainability

According to the ISO/IEC 9126 standard, maintainability includes quality attributes such as Analyzability and Changeability, which we were able to examine.

- For Analyzability there is no difference between the frameworks since both use HTML 5.
- For Changeability the code for both frameworks is equally changeable. But in some types of organization it is possible to influence the development of the actual framework. This is discussed further in the article.

However we focused on available documentation, and support community as additional attributes affecting maintainability. We chose these aspects to focus on since there were no noticeable differences between the frameworks with more important aspects of maintainability such as testability and readability. Furthermore,

they assist the task of maintaining the code by providing insight of how different functions work and allowing developers to seek help from other developers that had similar problems.

- For documentation, we compared the sizes of the official APIs for both frameworks, listing the number of functions in each framework.
- For the support community, we compared the number of posts on the official forums and what kind of organization handles the maintenance and development of the frameworks.

Simplicity

To measure the principle of simplicity, we compiled a list of steps taken to integrate the frameworks into Visual Studio 2012, and compared the complexity of the integration processes. For comparison, we used category definitions and ranked them from best to worst. This was the only aspect of simplicity that differed between the frameworks. As both frameworks are based on HTML 5 with different data types, they are both easy to read and work with. Furthermore, as the back-end for both prototypes is the exact same C# code it does not affect the results in any way.

These are the steps taken to integrate the frameworks into Visual Studio 2012:

1. Install from NuGet and start working
2. Manually put framework files in project folders
3. Install from NuGet and add additional files manually

3.2 Data analysis

The end result we aim to produce from this study is a list of how well the frameworks adhered to the VGTA principles and a conclusion that clearly shows which framework is the best in that context.

The data used will be from 4 data sets (one for each framework and platform) with 100 tests per dataset meaning 400 data points in total. Due to there being 4 different data sets, we have chosen to aim for analysis methods that analyse differences between data sets to reduce the risk of type I errors (a false positive error).

To analyse our data, we first ran a Kolmogorov-Smirnov test [21] for determining whether the data is normally distributed, as this determined what kind of tests we could use to further analyse the data.

Since the data was not normally distributed, we first used a Friedman's test, which is a non-parametric version of the ANOVA test [9]. This test determined if there were any significant differences between the values in the data sets. Then we performed non-parametric effect size tests for each combination of data sets [17].

4. RESULTS

The outcome from our analysis of the collected VGTA principles is shown in this section. In our work we limited ourselves to three principles due to time limitations. We choose performance in order to conduct quantitative analysis and objective measurements. We choose maintainability in order to capture how well documented the frameworks are. We choose simplicity in order to capture the subjective view on the difficulty in getting started with a framework. This subjective view is important for large organization, which usually deploy and have to support large number of clients.

We list the formal statistical analysis of the performance data gathered, as well as a table of data gathered related to maintainability. Furthermore, we list a comparison of the process of integrating the frameworks into Visual Studio 2012 and MVC4 to cover simplicity and compatibility with existing integration standards, from now on combined into simplicity due to test similarity.

Performance

As a first step in our data analysis we used SPSS [5] and RStudio to compile descriptive statistics of the data to get a general overview of the structure of the data and produce some diagrams.

Table 1: Shows the statistical data for both frameworks with the different platforms

Framework with platforms	Mean	Median	Max loading time	Minimum loading time
JQuery_iOS	11.42 sec	10,81 sec	22,59 sec	7,62 sec
Kendo_iOS	14,30 sec	13,45 sec	29,19 sec	12,33 sec
JQueryAndroid	8,43 sec	8,09 sec	16,95 sec	6,05 sec
KendoAndroid	17,41 sec	16,19 sec	44,94 sec	12,33 sec

Table 1 represents the statistical data for the 100 tests-runs for each framework in iOS and Android. It shows the mean and median for each data set and also shows the maximum and minimum time it took to load the same page.

Table 2: Kolmogorov-Smirnov Test of Normality

	Kolmogorov-Smirnov[a]		
	Statistic	df	Sig.
JQuery_iOS	,197	100	,000
Kendo_iOS	,228	100	,000
KendoAndroid	,234	100	,000
JQueryAndroid	,105	100	,009

a. Lilliefors Significance Correction

Table 2 presents the values that show whether data for the different OS are normally distributed. If the significant value is above 0.05 the data is normally distributed, while if it is below 0.05 the data is not normally distributed [8]. As shown in Table 2, all the data sets are not normally distributed because they are all below 0.05.

In Figure 3 we get a better understanding from the values that each framework for the different mobile platforms are not normally distributed. This is because the data points in the graphs are not following the line of normality.

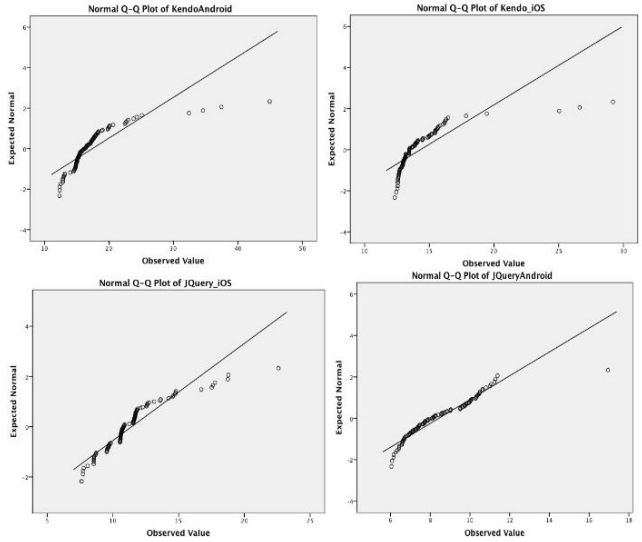

Figure 3: Q-Q plots of Kendo UI and JQuery for the 2 mobile platforms

Histograms in Figure 4 and Figure 5 show distribution curves for each framework for the different mobile platforms. The X-axis is the amount of time it took to load the page while the Y-axis is amount of data points in the same time. The line drawn in the histograms tells us that none of the histograms are normal. This is because the curve is not equally bent from each side; instead it falls down further to the end of the X-axis. The histograms are different because some histograms had more test runs in the same time interval.

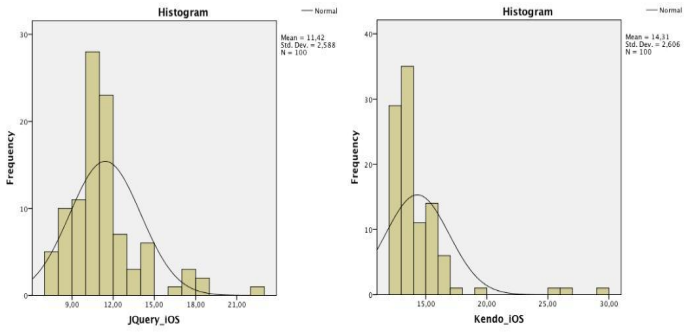

Figure 4: Difference between the frameworks in iOS

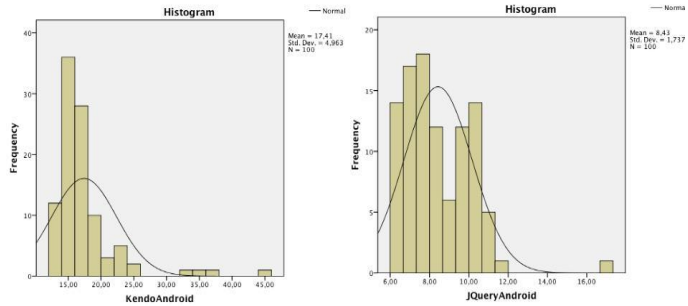

Figure 5: Difference between the frameworks in Android

The next phase was to determine if the differences between the data sets were statistically significant, and which data sets differed from each other. To measure this we performed a Post hoc analysis for Friedman's Test to see if there were any statistically significant differences between JQuery_iOS, Kendo_iOS, KendoAndroid and JQueryAndroid. The result showed that the test groups C, D and E were statistically significantly different between the data sets. There is a difference because each of the groups has a value below the p-value, which is 0,001.

Table 3: Post hoc Friedman's test statistic

Test group	Framework with platform: Sample1 – Sample2	Statistically significant difference
A	JQueryAndroid - JQuery_iOS	p < 0,001
B	Kendo_iOS - JQuery_iOS	p < 0,001
C	KendoAndroid - JQuery_iOS	0
D	Kendo_iOS - JQueryAndroid	0
E	KendoAndroid – JQueryAndroid	0
F	KendoAndroid – Kendo_iOS	p = 0,006

In addition to this a non-parametric effect size test was done to see how each test group's data sets differ from each other. Since test group C, D and E all has significant differences, each of the samples 1 from the three test groups has a higher loading time and large effect size. The result for this test is shown in Table 3.

From this data it is clear that Kendo UI Mobile on Android has much longer loading times than JQuery Mobile on both Android and iOS. Furthermore it is clear that Kendo UI Mobile on iOS has much longer loading times than JQuery Mobile on Android. Any other differences are not large enough to be statistically significant without excluding potential outliers.

Finally, we compared the two frameworks for the two OS to see how they measure against each other. In figure 6 diagram you can see four box plots and how they compare. The JQuery Mobile framework has better performance than the Kendo UI Mobile framework. The line in the middle of the boxes indicates the median of loading time. The top of the box represents the 75th percentile where 25% of the data points are above the 75th percentile. The bottom of the box represents the 25th percentile where 25% of the data points are below the 25th percentile. The T-bars, also called as inner fences, are extension from the boxes. The T-bars represents the minimum and maximum values of the box-plots. The points represent outliers while stars represent extreme outliers. These outliers are outside of the box plots because their values are three times time height of the boxes.

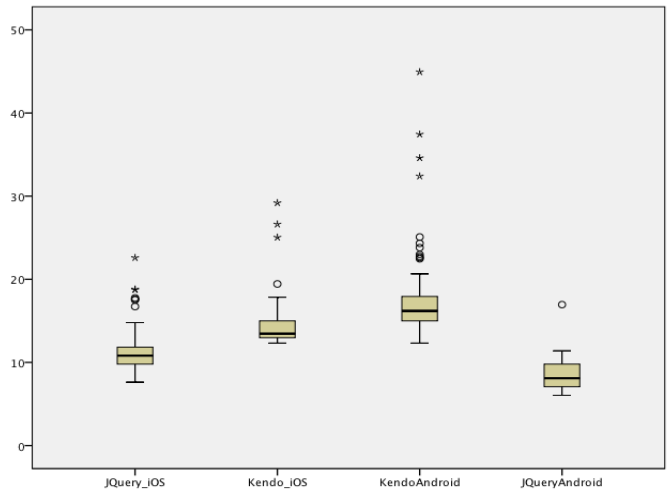

Figure 6: A box-plot diagram showing load-times in seconds.

Maintainability

In Table 4, we can see that JQuery Mobile has a much more active support community, with 3 times the number of threads on their forum.

Table 4: A table listing information gathered about maintainability

Maintainability	Kendo UI Mobile	JQuery Mobile
Documentation		
Functions covered in API	808	69
Support Community		
Type of organization	Commercial company	Open volunteer based community
Thread on official forum	3672 (premium + Stack overflow)	11351

Kendo UI Mobile has much better documentation, with an API covering more than 11 times more functions, thereby providing a much larger knowledge base.

For organization, JQuery Mobile is community-driven and relies on volunteers to keep maintaining it and provide support. This means it is vulnerable to differences of opinion between different groups within the community. Furthermore, there is the possibility of the community losing interest, either due to changing priorities or due to finding a "better" alternative and switching their support there. Kendo UI Mobile, however, is maintained and developed by a company that makes money from keeping it up to date. This motivates them to provide swift support and can make them less likely to discontinue the framework.

Simplicity

For JQuery Mobile, the only thing we needed to do in order to integrate it into Visual Studio 2012 was to install the JQuery.Mobile.MVC package through the NuGet package manager built into Visual Studio 2012 and then start creating mobile views.

For Kendo UI Mobile, we needed to put the correct files in the content and scripts folders in the Visual Studio 2012 project manually.

According to our defined categories for this principle this means that JQuery Mobile was easier to integrate, as everything could be done automatically through built-in functionality in Visual Studio 2012 without needing to manually sort files into their proper locations.

5. DISCUSSIONS

Performance

All analysis here points to JQuery Mobile being better, with the worst performing platform for Kendo UI Mobile being much worse than both platforms for JQuery Mobile and the best performing platform for Kendo UI Mobile being much worse than the best performing JQuery Mobile platform.

The performance figures are not infallible, as there are many factors that it sometimes took longer loading time for some test. These factors could be:

- **Code**

  Since Kendo UI Mobile and JQuery Mobile have different coding standard they might be implemented poorly which might cause Kendo UI Mobile to load slower than it otherwise would.

- **Laptop hardware**

  Since the server was hosted on an outdated laptop (2 GHz dual core processor, 3GB ram) on a wireless network. The mobile phones we used were outdated as well (iOS 5.1 and Android 2.3). If we would have had better hardware the result might have showed us differently.

- **Location of phones and server**

  The mobiles we used were located in Dulles, Virgina USA. The reason why we choose to test our web application on that server was because it was the only server that supported iOS and Android. There were other servers in EU as well, but some them only did support one of the mobile platforms. So it would not have been fair if one was tested in EU and the other one in USA. Furthermore the server was hosted in Sweden, so each load-time test had to first send a signal to the server in Sweden all the way from USA, and then returns the page the same distance. This is why the loading times are so long.

Maintainability

Both frameworks were quite even here, with JQuery Mobile having a more active community while Kendo UI Mobile has a much better API. The deciding factor here will most likely be whether an open volunteer based community or a framework driven by a profit generating company is preferable. In an open community, it is possible to contribute to steering the framework in the direction you want it to go, but it is also possible for the community to break down due to differences of opinion or due to lack of interest. In a profit driven company, however, the end users have less ability to impact the direction of the development, but it may be less likely for the framework to be discontinued.

5.1 Ethical Considerations

This thesis project was performed in cooperation with Volvo IT, and as such there may be possibility of bias in the results produced. We believe that this is not the case, as we were given complete freedom from the company as long as we integrated into MVC4 and used the principles to evaluate. How we went about implementing the prototypes and how we interpreted the principles and evaluated the prototypes against them was also left completely to us.

6. CONCLUSIONS

The goal of this thesis was to determine how suitable different HTML5 frameworks are for integration as MVC4 views when comparing to the VGTA principles. The comparison was done by implementing a prototype for each of the two frameworks and performing tests to gather data for statistical analysis.

The results show that JQuery Mobile is better suited to implementing mobile multi-platform MVC4 views. For the 4 measured and discussed in this article, JQuery Mobile was better for *performance*, *simplicity* and *compatibility with existing integration practices*, and only slightly worse for *maintainability*.

One of the main directions for continuing the research presented in this thesis is to construct a stand-alone prototype that connects to MVC4 through RESTful web API (which is easily implemented to the current base MVC4 application by auto-generating a web-api controller through visual studio to act as a RESTful API server) to compare the performance of a server-side integrated framework web page to a stand-alone app deployed on the phone itself. Other possibilities of research include comparison between MVC4 based prototypes and stand-alone apps complete with backend and database deployed to phones or comparing work processes for transitioning from existing applications to computer-mobile hybrids like our prototypes.

7. ACKNOWLEDGEMENTS

We want to thank our supervisor Miroslaw Staron, associate professor at the Department of Computer Science and Engineering Chalmers | University of Gothenburg, for guiding us through our project and supporting us. We would also thank Volvo IT and our supervisor, Johan Westlund, for providing us with this research topic and supporting us during the development of the thesis.

8. References

[1] Charland, D., & Leroux, B. 2011. Mobile application Development: Web vs . native, 0–4.

[2] Creswell, J. K. 2011. Research Design: Qualitative, Quantitative, and Mixed Methods Approaches - 3rd ed. *SAGE Publication Inc.*

[3] Drake, S.D. 2008. Embracing Next-Generation Mobile Platforms to Solve Business Problems. Available: http://www.eloquenza.de/fileadmin/content/Sybase/PDF/Sybase_WP_IDC_MobilePlatform.pdf. Last accessed 6/3/2013.

[4] Eldh, S. and Sundmark, D. 2012. Robustness Testing of Mobile Telecommunication Systems A Case Study on Industrial Practice and Challenges. *IEEE computer society*.

[5] IBM SPSS Data Collection 6.0.1 Information Center. 2013. *IBM SPSS Data Collection 6.0.1 Information Center*. [ONLINE] Available at:http://publib.boulder.ibm.com/infocenter/spssdc/v6r0m1/index.jsp?topic=%2Fcom.spss.ddl%2Ftukey_test.htm. [Accessed 26 March 2013].

[6] Järkeborn, J. 2012. AVS Architecture for Volvo Solutions.

[7] Kerschberg, L. 1989. The Role of Loose Coupling in Expert Database System Architecture. *IEEE*.

[8] Laerd Statistics. 2013. Testing for Normality using SPSS when you only have one independent variable. [ONLINE] Available at: https://statistics.laerd.com/spss-tutorials/testing-for-normality-using-spss-statistics.php. [Accessed 08 May 13].

[9] Jin, M., Li, Y., Wang, G., & Chen, J. 2012. Multiantenna based spectrum sensing via Friedman test for cognitive radio. 7th International Conference on Communications and Networking in China, 321–324.

[10] Leguizamo et al. 2003. Autonomous Decentralized Database System Reconstruction Technology through Mobile Agent Monitoring and Coordination. *IEEE computer society*.

[11] Mari, M. Eila, N. 2003. The Impact of Maintainability on Component-based Software Systems. *IEEE computer society*.

[12] Microsoft Corporation. 2012. *ASP.NET MVC Overview*. [ONLINE] Available at: http://msdn.microsoft.com/en-us/library/dd381412(v=vs.108).aspx. [Accessed 22 March 13]

[13] Microsoft Corporation. 2012. Introduction to the C# Language and the .NET Framework. Available at: http://msdn.microsoft.com/en-us/library/vstudio/z1zx9t92.aspx

[14] Microsoft Corporation. 2012. ASP.NET Overview. Available at: http://msdn.microsoft.com/library/4w3ex9c2.aspx

[15] Rouse, M. 2007. *What is integrated development environment (IDE)? - Definition from WhatIs.com*. [ONLINE] Available at: http://searchsoftwarequality.techtarget.com/definition/integrated-development-environment. [Accessed 17 May 13].

[16] Teng, Ch., & Helps, R. 2010. Mobile Application Development: Essential New Directions for IT. Information Technology: New Generations (ITNG) (pp. 471-475). Las Vegas, NV: IEEE.

[17] Vargha, A., & Delaney, H. D. (2000). A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics*, *25*(2), 101–132.

[18] W3C. HTML5 differences from HTML4. 2012. Available at: http://www.w3.org/TR/html5-diff/

[19] W3School. 2013. HTML5 Introduction. Available at: http://www.w3schools.com/html/html5_intro.asp

[20] WebpageTest. Available at: http://www.webpagetest.org/

[21] Zhang, G., Wang, X., Liang, Y. C., & Liu, J. 2010. Fast and Robust Spectrum Sensing via Kolmogorov-Smirnov Test. IEEE Transactions on Communications, 58(12), 3410–3416.

Appendix A

In this appendix we list all requirements and information about the prototypes needed to replicate this process.

Application requirements:

General:

- css file applicable to change colors and logos for different customers.
- display contents of database
- links to create/edit/view details/delete posts
- text field to enter search term
- search displays proper results

***Create order**:*

- input fields for name, date and part
- inputs data properly into database

Show details:

- displays name, date and part of chosen post

Edit:

- input fields for name, date and part
- properly modifies database post

Delete:

- display details about item being deleted (name, date, part)
- properly removes post from database

Appendix B

MVC4 application

1. In Visual Studio 2012, create a new project, select ASP.NET MVC4 Web Application in the Visual C# Web section and then select Internet Application on the next screen.
2. Remove all Views (except _Layout and _ViewStart), all controllers and the model and remove all links and printed text from the _Layout file.
3. Create a new model with the following variables: int id, string name, string part, string date. in this model also add a DbContext for use as a reference with entity framework. (make sure it is using System.Data.Entity)
4. Build the project so the model is recognized.
5. Create a controller, in the create window, in the template menu select "MVC controller with read/write actions and views, using entity framework". In the Model class dropdown menu select the model you created. In the data context class dropdown menu select the DbContext you created.
6. Implement the SearchIndex view as specified in Microsoft's "Getting Started with ASP.NET MVC 4" > "Examining the Edit Method and Edit View" substituting movies with your model and Title with Name.
7. Install Microsoft SQL Server Express 2012.
8. Set up IIS to receive remote connections to the port used by the project.

In the ConnectionString in the web.config file located OUTSIDE the views folder, change Data Source to .\SQLEXPRESS. Restart Visual Studio 2012 and build the project. While the debugger is running the page should now be accessible remotely.

Screenshots of the web application with Kendo UI Mobile and JQuery Mobile specification.

Figure 7: Index

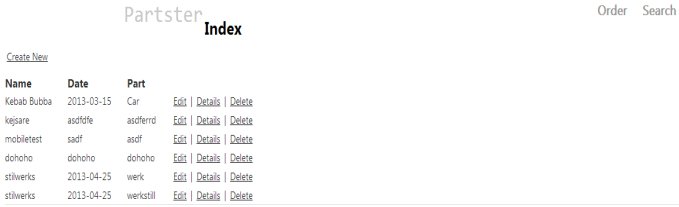

Figure 8: Orders

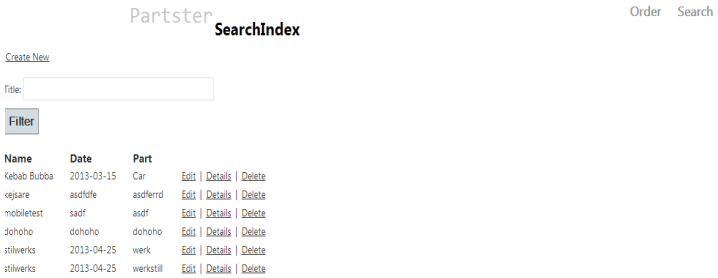

Figure 9: Search

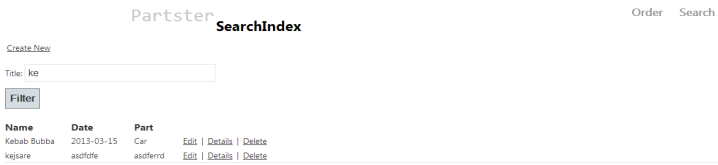

Figure 10: Search Result

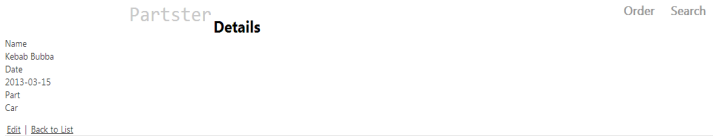

Figure 11: Detail

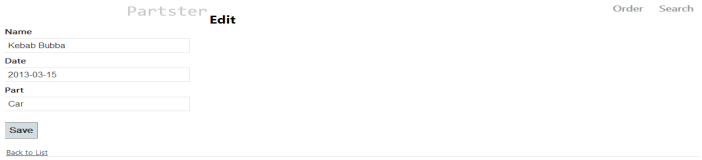

Figure 12: Edit

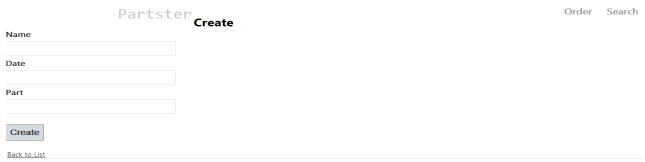

Figure 13: Create

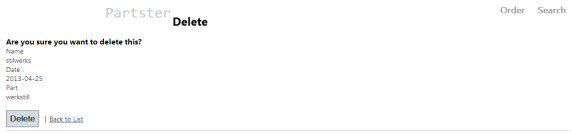

Figure 14: Delete

Kendo UI Mobile application

1. In a copy of the MVC4 application, add the Content folders for Kendo UI Mobile to the Content folder of the project. Add a Scripts folder to the root of the project and copy the Kendo UI Mobile scripts files and folder into that folder.
2. Create a _Layout.Mobile.cshtml file in the shared views folder and import the content and scripts folders for Kendo UI Mobile.
3. Create mobile versions of the views with the following naming convention: ViewName.Mobile.cshtml.
4. Where there are tables or lists in the regular views, replace those with listviews. use <li> sections with @html.actionlink to link to different views.
5. For the list of orders use this specific format: inside listview <li> Name: model.name Part: model.part Date: model.date

For the mobile searchindex view do not use a listview for the search form, instead nest a regular <ul> list inside the form. using a listview caused the textbox to be inaccessible.

Screenshot picture of Kendo UI Mobile in iPhone.

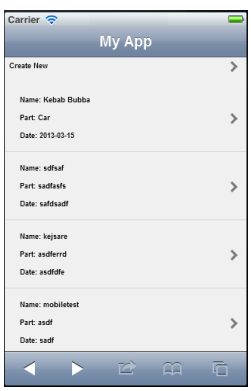

Figure 15: Order

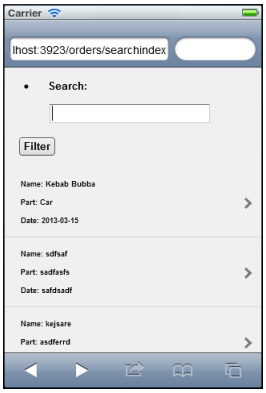

Figure 16:Search

Figure 17: Search Result

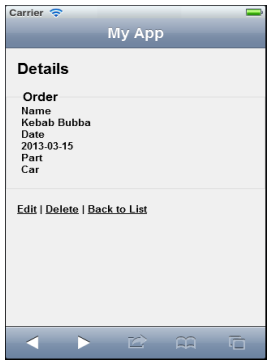

Figure 18: Detail

Figure 18: Create

Figure 19: Edit

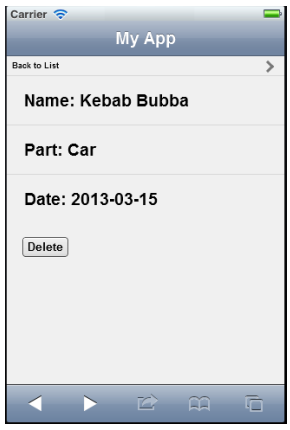

Figure 20: Delete

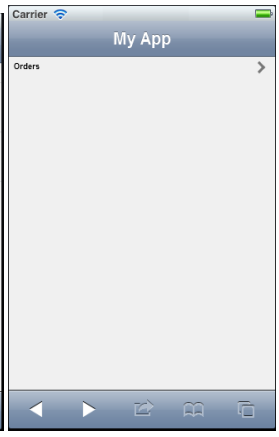

Figure 21: Index

JQuery Mobile application

1. In a copy of the MVC4 application in Visual Studio 2012, open the NuGet package manager, click the online tab and search for JQuery.mobile.mvc, add this package.
2. Remove the ViewSwitcher code from the _Layout.Mobile.cshtml file generated in the shared views folder.
3. Create mobile versions of the views with the following naming convention: ViewName.Mobile.cshtml.
4. Use listviews for every page, every separate section in the screenshots is a <li> section inside a listview.

Sometimes an empty <p> section inside a <li> section can be needed to avoid weird effects.

Do NOT implement a mobile Searchindex view, instead use data-filter=true when declaring the listview for the orders index view.

Screenshot picture of JQuery Mobile in iPhone 4.

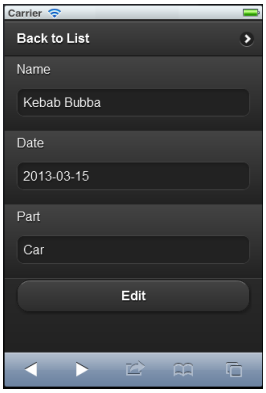

Figure 22: Edit

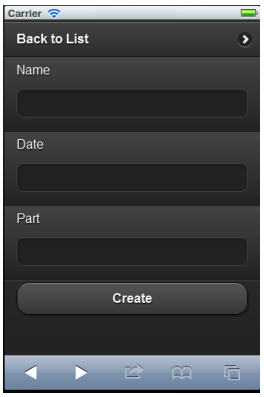

Figure 23: Create

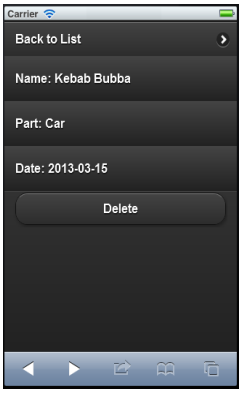

Figure 24: Delete

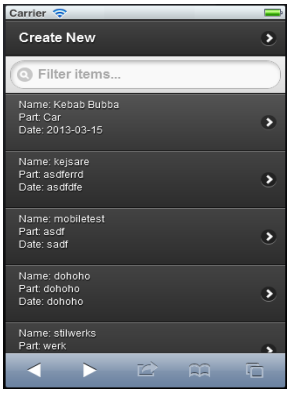

Figure 25: Order

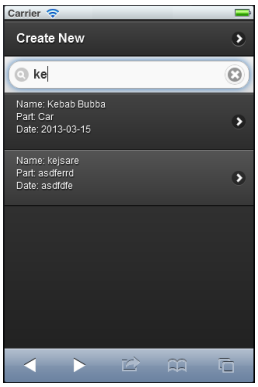

Figure 26: Search Result

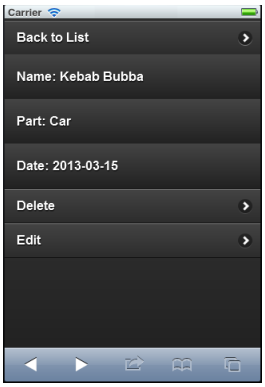

Figure 27: Detail

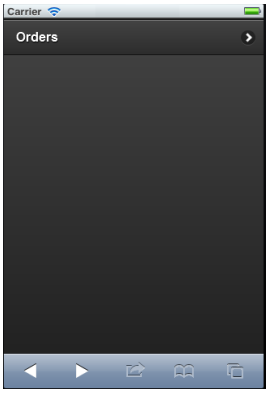

Figure 28: Index