Linnæus University
Sweden

Degree project

Comparing performance between plain JavaScript and popular JavaScript frameworks

Author: Zlatko Ladan
Supervisor: Johan Hagelbäck

Date: 2015-02-02

Course Code: 2DV00E, 15 credits
Level: Bachelor

Department of Computer Science

Abstract

JavaScript is used on the web together with HTML and CSS, in many cases using frameworks for JavaScript such as jQuery and Backbone.js. This project is comparing the speed and memory allocation of the programming language JavaScript and its two most used frameworks as well as the language on its own. Since JavaScript is not very fast and it has some missing features or features that differ from browser to browser and frameworks solve this problem but at the cost of speed and memory allocation, the aim is to find out how well JavaScript and the two frameworks jQuery and Backbone.js are doing this on Google Chrome Canary. The results varied (mostly) between the implementations and show that the to-do application is a good enough example to use when comparing the results of heap allocation and CPU time of methods. The results where compared with their mean values and using ANOVA. JavaScript was the fastest, but it might not be enough for a developer to completely stop using frameworks. With JavaScript a developer can choose to create a custom framework, or use an existing one based on the results of this project.

Keywords: comparison, programming, frameworks, JavaScript, jQuery, Backbone.js, heap allocation time-line, CPU profile time-line

Contents

1 Introduction

This project is comparing the speed and memory allocation of the programming language JavaScript and frameworks, jQuery and Backbone.js. Since JavaScript has some missing features or features that differ from browser to browser and frameworks solve this problem but at the cost of speed and memory allocation, the aim is to find out how well JavaScript and the two frameworks are doing this.

Heap allocation time-line and CPU Profile time-line are important tools to use with Google Chrome in order to make applications build up less memory and to run faster. Heap allocation time-line (and heap snapshot) is also used in order to check for problems such as memory not being released when it is not needed (known as a memory leak) [1][2]. In general a performance comparison between frameworks is important since it can help give a better picture of what framework is faster or what builds up the least memory. A developer might need to know if it is worth starting completely from scratch or to choose a framework by first knowing the difference between plain JavaScript and the frameworks jQuery and Backbone.js based on speed and memory allocation.

1.1 JavaScript

JavaScript is an important Programming Language since it is used on the web, applications like Google Mail and Google Maps use JavaScript to make the user experience highly interactive by giving their users rich desktop-like experience [3].

JavaScript is an interpreted programming language, with capabilities of an object-oriented (OO) programming language. An interpreted language is usually a programming language that executes most of its instructions directly as opposed to previously compiling a program into machine-language instructions [4]. It is mostly used in web browsers, but not limited to, for instance Node.js. Node.js is an runtime environment which is designed to build scalable server-side and networking applications [5].

In general a loosely typed programming language is a language that instead of generating an error produces unpredictable results or performs an implicit conversion when an argument gets passed to a function that does not closely match the expected type [6]. Type refers simply to a data type which is for instance a text string, integer or even a floating-point number. [7]. Implicitly converting means that the programming language on its own converts from one type to another without specifying it in the written code [8].

1.2 jQuery

jQuery is a small and fast library for JavaScript and it has plenty of features. Event handling, Ajax and animation are much simpler to use and works across a multitude of web browsers [9]. Companies like Microsoft and Nokia bundle jQuery on their platforms [10]. Ajax (asynchronous JavaScript and XML) are techniques used on the client-side (on users computer, not on server) to create asynchronous web applications [11]. In a study presented on 2014-11-26 showed that jQuery is used on many web sites, 90.67% to be exact [12].

1.3 Backbone.js

Backbone.js is a lightweight framework for JavaScript that uses the MV* application design which is similar to the MVC (model-view-controller) application design except it does not have a controller and instead adds the responsibility of the controller to the

view. LinkedIn, Walmart, SoundCloud, Disqus and Sony Entertainment Network's web shop are examples of companies that have used Backbone.js to build applications with [13][14]. Backbone.js requires the libraries underscore.js (version 1.5.0 or later versions) and jQuery in order to work according to its website [15].

1.4 Previous research

Some studies have been made that come close to the aim in this study. One study tested the performance differences between Angular.js and Backbone.js. It came to the conclusion that the differences were too small to be of any importance [16]. Another study tested the JavaScript memory leaks on some of the most used Google applications, and it came to the conclusion that there were some memory leaks that could cause problems if they were used in devices with limited resources as well as effect the users perceived latency [17].

A study researched if programmers should stick to following guidelines that are about writing efficient JavaScript code or to rely on the JavaScript execution engines to achieve good performance results. It validated across the browsers: Apple Safari (nightly build), Google Chrome, Mozilla Firefox (Beta), Opera (Beta), and Microsoft Internet Explorer. The web browsers at the time where the latest and the most commonly used. Guidelines that it followed were: not using the *eval* function, using local variables instead of using global ones, avoiding the *with* language construct, and more. It concluded that the question they initially had was timely, that using guidelines did make a difference in the performance of JavaScript and that the guidelines were valid to use. The guidelines usually helped even further those web browsers that had JIT-based (just in time compilation) JavaScript interpreters [18]. Just in time compilation is a technique where the intermediate representation is compiled to native code at runtime [4].

In general, most of the scientific performance tests in JavaScript have been made in plain JavaScript and in jQuery. Backbone.js has not been tested that much in comparison to plain JavaScript and jQuery.

1.5 Problem definition

Three implementations of a to-do application are made, every implementation identical in functionality for the user. Two of the implementations are made with frameworks and one is made in plain JavaScript (more details on framework versions in section 1.7). The three implementations are later compared in terms of heap allocation and time spent on each JavaScript method using mean value to compare and ANOVA test to check if the results are significant enough to differ (more details in section 2.1) [19].

1.6 Purpose and research questions / hypothesis

Since the purpose of the study is to compare the time and the memory allocation between plain JavaScript, Backbone.js and jQuery, the following three research questions have been made:

- RQ1: How fast in terms of execution time is jQuery and Backbone.js compared to plain JavaScript in Google Chrome?
- RQ2: How does memory usage of jQuery and Backbone.js compare to plain JavaScript in Google Chrome?

- RQ3: Is there a case where plain JavaScript is slower or builds up more memory than the frameworks: jQuery and Backbone.js?

Previous research has been done on jQuery compared to Angular.js and it has mentioned that the results did not differ very much for it to make a decision to choose the one framework over the other. One hypothesis is based on that research:

- H1: The CPU profile and the heap allocation will come quite close to one another and will not be of such big difference.

H1 is based on the previous research, referred to in section 1.5.

1.7 Limitation

In this report, the scope is limited to three types of applications in JavaScript and two frameworks. That is plain JavaScript, Backbone.js and jQuery. Specifically:

- Backbone.js (compressed version 1.1.2 with compressed jQuery version 2.1.1 and compressed underscore.js compressed 1.7.0 since they are required by the library).

- jQuery (compressed version 2.1.1).

Compressed means that the JavaScript file has been removed from all unnecessary characters from source code, though the functionality stays the same as the non-compressed version [20]. The compressed versions where chosen to save space and time when the web browser is fetching the file.

jQuery is the most used framework for JavaScript and Backbone.js is the second most used framework for developing JavaScript websites [12].

Other more common frameworks exist, such as Errorception, which is used for error finding and Modernizr which is "a small JavaScript library that detects the availability of native implementations for next-generation web technologies..." according to its own web site [21][22]. Besides jQuery, Backbone.js and Angular.JS are frameworks that are more dynamic and can accomplish more than the mentioned ones.

The functionality that will be tested in this report is very basic and it is reasonable to believe that if there are memory problems in the frameworks, they are not to be found there.

Angular.js was dismissed mainly because it was too time consuming to evaluate and it was a completely new framework to work with, and also because its functionality differs a bit from the others, an example is templates. Angular.js seemed to only allow templates to be stored in HTML-like files that had to be fetched separately using JavaScipt. Fetching the template-file separately would likely take more time to do and Google Chrome seems to not store a methods CPU time if it takes little time compared to the total time (mentioned later in section 2.4).

1.8 Target group

Web developers that focus on single page applications or just web applications that require JavaScript, jQuery or Backbone.js. The results might be interesting for developers in general and also for companies that need to know how much memory can be saved by using a different framework or no framework.

1.9 Overview of the report

Chapter two will explain which methods have been made, which tools and why and how the software is built up. Chapter three contains the study itself and presents the results in detail. Chapter four contains a discussion about the results and Chapter five, concludes the report by discussing some possible topics for future research.

2 Method

The report is based on experiments that provide quantitative data. It is completely based on the data that a closed system (in this case the web browser Google Chrome) is providing. All of the tests can be replicated by anyone who has the code and the same version of Google Chrome.

2.1 Experimental setup

Tools that were used when testing:

- Windows 7 Professional N (installed on a Virtualbox machine version: 4.3.20).
- Google Chrome (version: 40.0.2202.2 canary).
- Google Chrome's built-in developer tools.

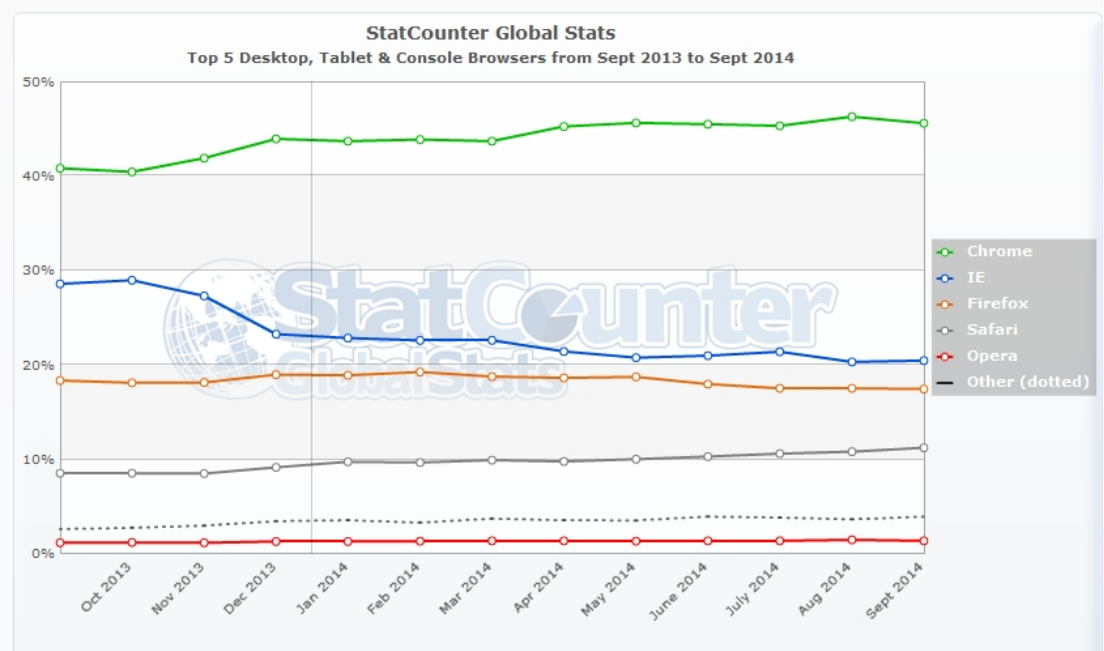


Figure 2.1: Browser usage statistics.

Since the most frequently used web browser for computers takes up around 46% is Google Chrome (Figure 2.1 for more details) it was chosen, another reason that Google Chrome was chosen was because it did not require any extensions/add-ons and had the developer tools built into the program [23]. The Canary version of Google Chrome on the other hand was used since it is aimed at web developers and from what was tested the ordinary version of Chrome could not save CPU time profiles which was necessary for testing [24].

Four experiments have been made using Chrome's built-in "Web developer tools". Each test is done five times, five for each "heap allocation time-line" and five for each "CPU profile time-line". The four tests are as follows:

- Loading the web page.

- Adding a to-do task.
- Removing a to-do task.
- Editing a to-do task.

"CPU profile time-line" and "heap allocation time-line" are used since they are simple to get in to and use, and thereby making comparing simple. The web browser lacks the features to compare data for "heap allocation time-line" and "CPU Profile time-line" which has to be done manually.

When doing the tests it was not possible to save the heap allocation time-line, instead it got saved as a heap snapshot which is not the same since it does not involve a time-line and tests were needed to be done again because of it. When recording "CPU profile time-line" it was noticed that it did not record every little detail if the method took too little time compared to the recording time, which resulted in tests being done again.

2.1.1 Loading the web page

Recording starts from a previous page and ends when all of the text and the graphical elements are displayed.

2.1.2 Adding a to-do task

Recording starts before writing a to-do task, text is written in the *textarea* and enter is pressed, and ends when task is added to the list of tasks.

2.1.3 Removing a to-do task

Recording starts before removing a to-do task, *confirm* dialog (mentioned later in 3.1) is displayed and closed with pressing *OK* button, ends with the item getting removed from the to-do list.

2.1.4 Editing a to-do task

Recording starts before pressing the edit button on to-do task, text from the to-do task gets automatically added to the *textarea*, text is changed, enter is pressed and ends when the text on the task has been changed.

2.1.5 Data to compare

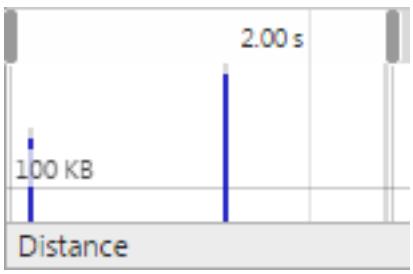


Figure 2.2: Example of Chrome heap allocation time-line selection.

When recording a heap allocation's data it is selected manually from the time-line chart which is found in the "Summary" view (Figure 2.2) since Google Chrome seems to have data stored from even before the recording. The horizontal axis is time in seconds, the vertical axis is memory (in this case kilobytes) that is being allocated, exact data in kilobytes cannot be found on the diagram, though it possible to see approximately how much data is getting allocated. A horizontal line represents 100 kb which is used as a scale, same thing applies to the vertical line that represents 2.00 seconds. In the same Figure two blue lines can be seen, the first takes a bit more than 100 kb and the second line even higher than that, the two represent two times where memory has been allocated (not added, but allocated at different times).

When recording a CPU profile time-line it is found in the "Tree(Top Down)" view, the data seems to not always be the same and that is why it has to be redone and have the most methods in the list. Garbage collector is ignored since it seems to not differ (sometimes it does not even show up) and is taken care by the browser. Only Methods from the implementations are compared (some methods are seen as anonymous in some cases).

2.2 Data Collection

The data that has been used in this study is based on the tools that Google Chrome provides for its users in its web development tools.

Self ▼		Total		Function
709.6 ms	44.74 %	709.6 ms	44.74 %	(idle)
109.2 ms	6.88 %	109.2 ms	6.88 %	(program)
3.0 ms	0.19 %	767.2 ms	48.37 %	▶ ZAPP.remove script.js:85:19

Figure 2.3: Example of Chrome CPU profile time-line.

Figure 2.3 shows a CPU profile time-line that was running for the test for removing a to-do task, specifically the JavaScript version. The "CPU profile time-line" can be found by first opening the menu which is indicated with three horizontal lines and is placed on the right side of the browser, then selecting "More tools" and in that sub-menu clicking "Developer tools". In that window, tabs are visible, click "Profiles", now it should look like Figure 2.4. Recording is started by selecting the "Collect JavaScript CPU Profile" (or "Record Heap Allocations" to record the heap allocation time-line) radio button and pressing the *Start* button, to stop, click the red button placed top left in developer tools.

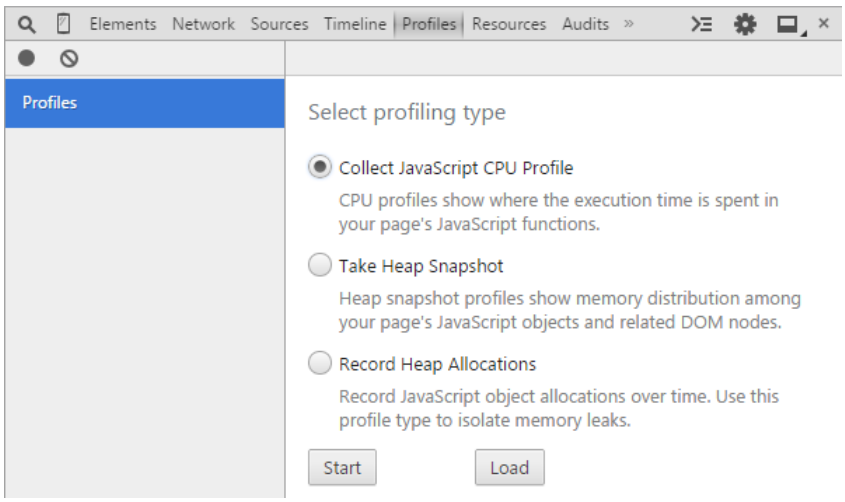


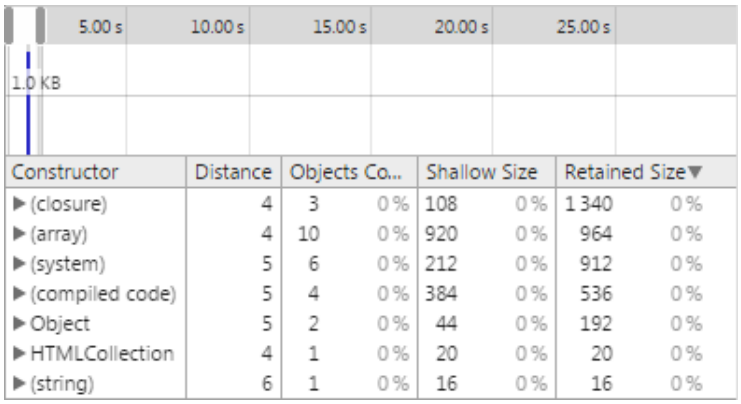
Figure 2.4: Developer tools.

Time spent in non-browser processing is "(program)" and "(idle)" is when the script is idle [25]. Difference between "Self" and "Total" is [1]:

- "Self time — How long it took to complete the current invocation of the function, including only the statements in the function itself, not including any functions that it called."
- "Total time — The time it took to complete the current invocation of this function and any functions that it called."

A test was executed for loading the page with the CPU profile time-line for Backbone.js. The recording was first started, then the site was opened, lastly the recording was stopped. The two functions that took the most time where "(idle)" and "(program)". Between 46.21 and 72.91% of the time was taken by the function "(idle)". The function"(program)" took between 8.7 and 31.05% of the time. Percentages did differ since "(program)" can differ from what the browser (Chrome) is currently doing. If enough time has passed the garbage collector starts working and it can be shown.

When running CPU profile for plain JavaScript "(program)" and "(idle)" were mostly only visible and occasionally "(garbage collector)", only when the recording time was lower than two seconds were the written JavaScript functions visible on the list.



Constructor	Distance	Objects Co...		Shallow Size		Retained Size▼	
▶ (closure)	4	3	0 %	108	0 %	1 340	0 %
▶ (array)	4	10	0 %	920	0 %	964	0 %
▶ (system)	5	6	0 %	212	0 %	912	0 %
▶ (compiled code)	5	4	0 %	384	0 %	536	0 %
▶ Object	5	2	0 %	44	0 %	192	0 %
▶ HTMLCollection	4	1	0 %	20	0 %	20	0 %
▶ (string)	6	1	0 %	16	0 %	16	0 %

Figure 2.5: Example of Chrome heap allocation time-line.

Figure 2.5 shows a "heap allocation time-line" that was running for the test for removing a to-do task, specifically the JavaScript version. It shows how much memory has been allocated by the application, when calculating the combined "retained size" are added together. This can be found in the same place as "CPU Profile time-line", as mentioned in section 2.4 it can only be saved as a heap snapshot.

2.2.1 Software Development

One application was developed in three different ways. One in plain JavaScript, one in Backbone.js and one in jQuery. Each of them is made for testing basic CRUD (create-read-update-delete) functionality on so called to-do tasks. These tasks are basically regular strings in a list that can be added, altered and removed, the tasks can be toggled to be done. A set of rules were applied for the applications, the rules were made up during the development:

- A list should be visible when there are to-do tasks and for each of them there should be options removal and edit visible for them.
- A text box for adding new to-do tasks, the enter key is used instead of an "OK" button when adding a new task.
- In order to edit there has to be a to-do task available and the edit button should start the editing, the task gets highlighted and the text from the textarea gets replaced with the to-do task which is being edited, and when editing the escape button cancels it, but keeps the text in the textarea.
- In order to remove there has to be a to-do task available and the remove button should first prompt if the user wants to remove the task (with the text if the task), or not. If the user cancels or chooses no it is kept, if not, it gets removed.
- If a task is was just added, removed, or edited the textbox is still focused on.
- Structure wise the HTML created by the different frameworks should be all the same in the application.

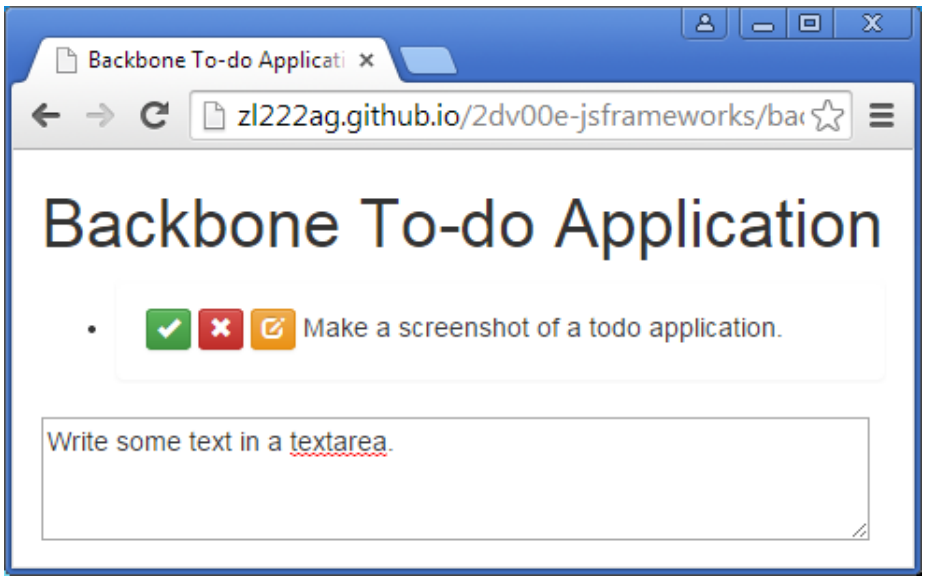


Figure 2.6: Example of the application, specifically the Backbone.js version.

Figure 2.6 is an example of how the implementations look, as mentioned previously they should look the same, both the HTML structure created by the JavaScript code and the visual colors and so on.

The source code to the applications can be found in the GitHub repository at `http://github.com/zl222ag/2dv00e-jsframeworks`.

2.3 Ethical considerations

This study does not consider any social or ethic consideration whatsoever. It only deals with the performance of plain JavaScript and the two frameworks jQuery and Backbone.js. The collected data comes from different system tools and standard libraries. The evaluation is based on numerical values and done as objective as possible.

2.4 Reliability

Many of the plain JavaScript CPU profile time-line tests were needed to be repeated several times since Chrome does not seem to record everything if a method takes little time compared to the total recording time. The same thing happened to the Backbone.js and jQuery tests, but not as frequent. The results which have missing data were not used because of that reason.

All data was collected from Google Chrome, but it was not running on a physical computer directly, instead it was running from a virtual machine (Virtualbox, with OS Windows 7). Because of this the data could differ if the same tests were done on a physical computer, but since all of the data was collected in the same environment they probably got affected the same amount. A virtual machine is a software implementation of a machine (for example a computer) that runs an operating system and applications like a physical machine [26]. A virtual machine was chosen since it is under an environment which can be done on a different computer and result in similar results (results in Google Chrome differ always a little) since hardware is emulated and can be chosen.

Another issue is that the recorded data could differ depending on web browser and even web browser version (if its even possible to record data the same way), for instance

the frameworks may handle its code differently depending on which web browser runs its code.

Google Chrome does exist for Linux, Mac OS and Windows computers, of which I chose Windows. The Google Chrome implementation when running on other operating systems could affect the data results, though the results would likely be reasonably similar.

3 Results and Analysis

CPU profile time-line tests and heap allocation tests were recorded, included mean values and they where checked for statistical significance using ANOVA with significance level 0.05.

Analysis of variance (ANOVA) is a collection of statistical models that are used in order to analyze the difference between group means and their associated procedures. It is useful for comparing three or more mean values for statistical significance [19].

3.1 CPU profile time-line

The Table 3.1 shows results of the recordings, row 1 shows which application it is, row two shows methods that were recorded and the rest of the Table shows the results of the tests except for the two last rows which show the mean values for each ones of the applications. First with each of the functions shown, then the total run time per application.

The Table has columns in the second row with text like "(s:115:42)". This refers to an anonymous method in a JavaScript file (in this case "script.js", with the position in the file (row:column). The different items are explained in Table 3.2.

	JavaScript	**jQuery**		**Backbone.js**				
	(s:115:42)	**(jQ:1)**	**I**	**(BB:1)**	**(jQ:1)**	**(_:1)**	**(s:1)**	**I**
	7.1	290.0	80.9	207.7	282.2	42.6	47.9	111.8
	5.8	384.9	60.4	39.4	171.5	6.9	8.6	18.9
	4.1	347.7	78.2	18.7	137.0	22.8	16.6	10.4
	3.9	220.8	37.8	31.0	155.2	29.0	20.7	22.8
	9.6	187.4	47.9	39.7	152.0	19.8	2.2	37.5
Mean	6.1	286.2	61.0	67.3	179.6	24.2	19.2	40.3
Total mean	6.1	347.2		330.6				

Table 3.1: Method running time (ms) with mean time (ms) when loading page.

In Table 3.1 there is not any significant difference between the jQuery and the Backbone.js implementations, the JavaScript implementation however seems to differ significantly between the other two according to an ANOVA test, the JavaScript application is the fastest.

Item	**Description**
s	The app starting file (script.js) which exists for all implementations.
jQ	The jQuery source file (jquery-min.js) which exists for the Backbone.js and jQuery implementations.
BB	The Backbone.js source file (backbone-min.js).
_	The underscore source file (underscore-min.js).
I	A help method in the jQuery framework (jquery-min.js) that runs automatically.

Table 3.2: Explanation of items in Table 3.1.

Table 3.3 shows time in milliseconds for adding a to-do task to the application. First

row show the framework, second row shows the methods that were running, last row shows the mean value of the results.

	JavaScript	jQuery	Backbone.js
	ZAPP.keyDown	**r.handle**	**r.handle**
	8.5	3.6	51.0
	5.8	8.5	34.1
	4.2	6.9	21.1
	12.6	11.2	13.5
	4.0	12.1	46.6
Mean	7.0	8.5	33.3

Table 3.3: Method running time (ms) with mean time (ms) when adding a to-do task.

In Table 3.3 there is a statistically significant difference between JavaScript and the two frameworks, but no significant difference between jQuery and Backbone.js. The different items are explained in Table 3.4.

Item	**Description**
ZAPP.edit ZAPP.keyDown	Methods in all of the implementations.
r.handle	A Method in the jQuery framework (jquery-min.js).

Table 3.4: Explanation of items in Table 3.3.

Table 3.5 shows time in milliseconds for removing a to-do task to the application. First row show the framework, second row shows the methods that were running, last row shows the mean value of the results.

	JavaScript	jQuery	Backbone.js
	ZAPP.remove	**r.handle**	**r.handle**
	3.0	6.8	10.4
	3.7	18.3	12.9
	3.4	10.9	13.1
	19.9	0.0	41.1
	4.2	10.9	3.4
Mean	6.8	9.4	16.2

Table 3.5: Method running time (ms) with mean time (ms) when removing a to-do task.

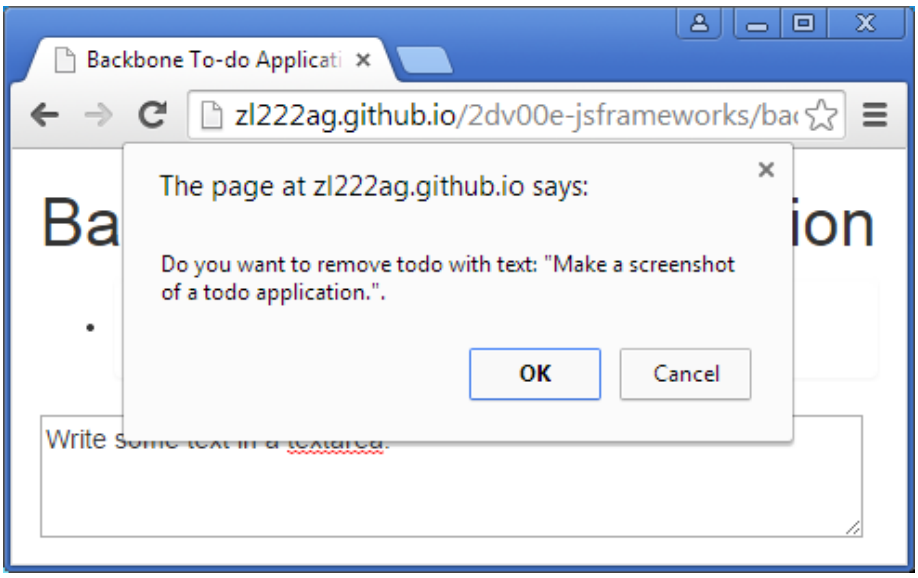


Figure 3.1: Example of a *confirm* dialog, specifically the Backbone.js version.

The Table 3.5 is using the built-in method *confirm* before removing which locks the code and prevents the user from accessing the rest of the application's interface, shows a dialog (Figure 3.1) and asks the user for input, after the users chooses an options the code continues and the user can access the application's interface. The *confirm* method does affect the applications equally and the time it takes to press "OK" or "Cancel" button is not taken into consideration when comparing the frameworks. Because of this its time is removed.

There is no statistically significant difference between the results of the implementations when comparing for removing a to-do task. The different items are explained in Table 3.6.

Item	**Description**
ZAPP.remove	A Method in all of the implementations.
r.handle	A Method in the jQuery framework (jquery-min.js).

Table 3.6: Explanation of items in Table 3.5.

Table 3.7 shows time in milliseconds for editing a to-do task. First row show the framework, second row shows the methods that were running, the two last rows shows the mean value of the results with the last one of them showing the total mean of all methods for each application.

	JavaScript		jQuery	Backbone.js
	ZAPP.edit	**ZAPP.keyDown**	**r.handle**	**r.handle**
	11.1	3.7	37.4	11.1
	4.2	8.4	44.3	39.0
	4.8	3.2	85.1	56.1
	6.8	1.7	23.4	64.7
	5.6	1.9	59.0	16.2
Mean	6.5	3.8	49.8	37.4
Total mean	10.3		49.8	37.4

Table 3.7: Method running time (ms) with mean time (ms) when editing a to-do task.

When editing a to-do task the time of the JavaScript and jQuery implementations seem to differ significantly which is seen in Table 3.7. At the same time we can state that the time does not differ significantly between JavaScript and Backbone.js. The implementation of Backbone.js and jQuery do not differ significantly in time either. The different items are explained in Table 3.8.

Item	**Description**
ZAPP.edit ZAPP.keyDown	Methods in all of the implementations.
r.handle	A Method in the jQuery framework (jquery-min.js).

Table 3.8: Explanation of items in Table 3.7.

r.handle (Figure 3.8) in the Backbone.js and jQuery applications the framework jQuery calls the *ZAPP.keyDown* and/or *ZAPP.edit* methods since those methods are used in events.

Test Subject	**Loading**	**Adding**	**Removing**	**Editing**
JavaScript	6.1	7.0	6.9	10.3
jQuery	347.2	8.5	9.4	49.8
Backbone.js	330.6	33.3	16.2	37.4

Table 3.9: Mean values for all CPU times in milliseconds.

Tables 3.1, 3.3, 3.5 and 3.7 are the results of the tests that were recorded with Google Chrome's JavaScript CPU Profile. Table 3.9 is only displays the mean value from all of the tests data that are on the mentioned Tables.

3.2 Heap allocation time-line

The Tables in this section show the heap allocation in kilobytes.

The Table 3.10 shows how much memory has been allocated when loading the applications. First row with the framework, the rest with the results and last one with the mean value.

	JavaScript	jQuery	Backbone.js
	271.0	528.0	633.0
	271.0	525.0	631.0
	272.0	525.0	631.0
	273.0	525.0	634.0
	273.0	528.0	631.0
Mean	272.0	526.2	632.0

Table 3.10: Heap allocation in kilobytes with mean value when loading the to-do page.

In Table 3.10 there is a significant difference between the results for the different implementations.

The Table 3.11 shows how much memory has been allocated when adding a to-do task for the applications. First row with the framework, the rest with the results and last one with the mean value.

	JavaScript	jQuery	Backbone.js
	17.5	68.6	59.4
	17.0	69.9	54.7
	17.0	68.5	56.9
	16.8	68.5	56.6
	17.0	68.3	56.9
Mean	17.1	68.8	56.9

Table 3.11: Heap allocation in kilobytes with mean value when adding a to-do task.

Like the previous Tables (in section 3.1) the Table 3.11 shows that there is a significant difference between the results of the three implementations when adding a to-do task.

The Table 3.12 shows how much memory has been allocated when removing a to-do task for the applications. First row with the framework, the rest with the results and last one with the mean value.

	JavaScript	jQuery	Backbone.js
	1.7	25.1	25.9
	1.7	24.5	24.4
	1.7	24.5	24.4
	1.7	24.5	25.6
	1.7	22.9	25.6
Mean	1.7	24.3	25.2

Table 3.12: Heap allocation in kilobytes with mean value when removing a to-do task.

Though when removing a to-do task the Table 3.12 has a significant difference between the results of the JavaScript and both jQuery and Backbone.js implementations. There seems to be no difference between the results of the jQuery and Backbone.js implementations.

The Table 3.13 shows how much memory has been allocated when editing a to-do task for the applications. First row with the framework, the rest with the results and last one with the mean value.

	JavaScript	jQuery	Backbone.js
	5.9	30.6	39.3
	5.9	30.6	39.1
	5.9	30.5	39.0
	5.9	30.6	39.0
	5.9	30.6	34.2
Mean	5.9	30.6	38.1

Table 3.13: Heap allocation in kilobytes with mean value when editing a to-do task.

Table 3.13 shows that in the case of editing to-do tasks that there is a significant difference between the results of the implementations.

It seems that on most occasions the implementations do significantly differ in memory usage, the only time it does not differ significantly is when comparing the implementations of jQuery and Backbone.js during removal of a to-do task.

Test Subject	Loading	Adding	Removing	Editing
JavaScript	272.0	17.1	1.7	5.9
jQuery	526.2	68.8	24.3	30.6
Backbone.js	632.0	56.9	25.2	38.1

Table 3.14: Mean values for heap allocations in kilobytes.

Tables 3.10, 3.11, 3.12 and 3.13 are the results of the tests that were recorded with Google Chrome's heap allocation time-line. Table 3.14 only displays the mean value from all of the tests data that are on the mentioned Tables.

4 Discussion

Research questions have been answered and are based on data found in Chapter 3.

4.1 Problem solving / Results

The results may be used to show that Backbone.js does not differ much from jQuery, that the results differ more between JavaScript and jQuery and between JavaScript and Backbone.js.

RQ1 How fast in terms of execution time is jQuery and Backbone.js compared to plain JavaScript in Google Chrome?

JavaScript is the fastest, followed by jQuery and then Backbone.js. In most cases JavaScript is much faster and jQuery and Backbone.js do not differ much.

From what was shown in Table 3.1 (loading the to-do page) and the ANOVA results there is no significant difference between the jQuery and Backbone.js implementations and JavaScript is much faster with 6.1 ms compared to jQuery's 347.2 and Backbone.js's 330.6 ms.

Adding a to-do task can be seen in Table 3.3, according to ANOVA results the JavaScript and jQuery results do not differ significantly and Backbone.js is the slowest with 33.3 ms compared to JavaScript's 7.0 and jQuery's 8.5 ms.

When removing a to-do task which is on Table 3.5, there is no significant difference between any of the implementations.

Editing a to-do task (Table 3.7) is again fastest on the JavaScript implementation, but according to the ANOVA results there also is no difference between the jQuery and Backbone.js implementations, 10.28 ms for JavaScript, 49.8 ms for the jQuery implementation and 37.4 ms.

RQ2 How does memory usage of jQuery and Backbone.js compare to plain JavaScript in Google Chrome?

JavaScript uses less memory than both framework, followed by jQuery and Backbone.js, though the frameworks do not differ much between them.

When Loading the to-do page (Table 3.10) there is a significant difference between the implementations, JavaScript takes less memory with 272.0 kb followed by jQuery with 526.2 kb and Backbone.js with 632.0 kb.

Adding a to-do task (Table 3.11) again differs significantly between the implementations JavaScript again takes the least memory with 17.1 kb, jQuery takes the most with 68.8 kb and Backbone.js is between the two with 56.9 kb.

Removing a to-do task (Table 3.12) does not differ between jQuery and Backbone.js. JavaScript takes the least memory with 1.7 kb compared to the memory of jQuery and Backbone.js, 24.3 kb and 25.2 kb respectively.

Editing a to-do task (Table 3.13) differs in implementations, Backbone.js takes the most memory with 38.1 kb, jQuery takes 30.6 kb and the least is JavasScript with 5.9 kb.

RQ3 Is there a case where plain JavaScript is slower or builds up more memory than the frameworks: jQuery and Backbone.js?

There is no such case, there is on the other hand a case with CPU time where all the implementations do not differ significantly (removing a to-do as mentioned in RQ1).

H1 The CPU profile and the heap allocation will come quite close to one another and will not be of such big difference.

The CPU profile and heap allocation did differ enough to be a difference, though in many cases it did not differ greatly, especially between the frameworks jQuery and Backbone.js as mentioned earlier in RQ1, RQ2, and RQ3.

4.2 Method reflection

Plain JavaScript was fastest and used the least memory, except for one case where memory did not differ significantly between all of the implementations.

Even though using plain JavaScript is the fastest and uses the least memory, it has a few drawbacks such as:

- Compatibility, e.g. element functions that were used like classList are not available for all web browsers, do not function or might introduce bugs.
- Missing features, e.g. templates: creating hierarchy of elements.
- Not always easy, e.g. Ajax, sending and fetching data involves many steps and checks in order to work properly.

jQuery has more features though it is much slower than plain JavaScript it does not have any of the mentioned drawbacks of plain JavaScript. A drawback which has occurred with jQuery are bugs, but they are uncommon.

Backbone.js is the slowest, but it has even more features than jQuery since it is based in jQuery (and underscore.js) and has its own features. Backbone.js is great for using in big applications which without it (or a similar framework or architectural pattern) might get its code tangled together, MV* is based on MVP and it improves the separation of concerns which solves problems by separating data and the visuals (and others) [27]. Separation of concerns is a design principle for separating a computer application into distinct sections, with each section addressing a separate concern. Its value is making development and maintenance of computer applications simpler [28].

Plain JavaScript is great for making a small, simpler applications. jQuery is great for big applications though it would as plain JavaScript get tangled the code does not follow the separation of concerns. Backbone.js has the MV* pattern which helps a lot.

5 Conclusion

This Chapter contains the conclusions that have been made from this study and some thoughts about future research.

5.1 Conclusions

The results of the experiments came close to what was expected, though it did not vary as much as expected.

It was expected that in every case JavaScript would be the fastest and would build the least memory. Backbone.js would always be slower and would build the most memory.

JavaScript did for the most case be the fastest, except for two cases, one where it was not significantly different from both jQuery and Backbone.js and one where it was not significantly different for jQuery (RQ1 in Section 4). JavaScript takes always the least memory compared to the other applications.

Backbone.js did in many cases not differ significantly from jQuery when comparing time. It did differ when comparing heap allocation, though it had one case where it built up less memory than jQuery, in many cases the differences were not big (between 4 and 20%).

The conclusion of the results is that it seems that the framework Backbone.js does not differ enough with jQuery in terms of speed and memory allocation for it to be not chosen over jQuery.

5.2 Further research

One thing that could be useful to really spot the differences in speed and heap allocation between JavaScript and its frameworks would be to dig deeper into the functionalities that the frameworks provide.

Of course, it would also be possible to run the code on different web browsers and operation systems to see if they are as good at handling the performance on them, the only problem possibly being difficult to compare as previously stated in 2.4.

Hopefully in the future Google Chrome might get better at recording the method calls, since it does not seem to record every function that was running (mentioned previously in Chapter 2.4).

References

[1] G. Inc. (2014, 11) Profiling javascript performance. Google Inc. [Online; accessed 21-November-2014]. Available: https://developer.chrome.com/devtools/docs/cpu-profiling

[2] Wikipedia, "Memory leak — wikipedia, the free encyclopedia," 2014, [Online; accessed 11-December-2014]. Available: http://en.wikipedia.org/w/index.php?title=Memory_leak&oldid=636969476

[3] C. Severance, "Javascript: Designing a language in 10 days," *Computer*, vol. 45, no. 2, pp. 7–8, 2012, [Online]. Available: http://www.computer.org/csdl/mags/co/2012/02/mco2012020007.pdf

[4] Wikipedia, "Interpreted language — wikipedia, the free encyclopedia," 2014, [Online; accessed 2-December-2014]. Available: http://en.wikipedia.org/w/index.php?title=Interpreted_language&oldid=635640648

[5] ——, "Node.js — wikipedia, the free encyclopedia," 2014, [Online; accessed 2-December-2014]. Available: http://en.wikipedia.org/w/index.php?title=Node.js&oldid=636217208

[6] ——, "Strong and weak typing — wikipedia, the free encyclopedia," 2014, [Online; accessed 12-December-2014]. Available: http://en.wikipedia.org/w/index.php?title=Strong_and_weak_typing&oldid=636506759

[7] ——, "Data type — wikipedia, the free encyclopedia," 2014, [Online; accessed 12-December-2014]. Available: http://en.wikipedia.org/w/index.php?title=Data_type&oldid=633654954

[8] ——, "Type conversion — wikipedia, the free encyclopedia," 2014, [Online; accessed 12-December-2014]. Available: http://en.wikipedia.org/w/index.php?title=Type_conversion&oldid=634691740

[9] T. jQuery Foundation. (2014, 11) jquery. The jQuery Foundation. [Online; accessed 29-November-2014]. Available: http://jquery.com/

[10] ——. (2014, 12) jquery, microsoft, and nokia. The jQuery Foundation. [Online; accessed 8-December-2014]. Available: http://blog.jquery.com/2008/09/28/jquery-microsoft-nokia/

[11] Wikipedia, "Ajax (programming) — wikipedia, the free encyclopedia," 2014, [Online; accessed 2-December-2014]. Available: http://en.wikipedia.org/w/index.php?title=Ajax_(programming)&oldid=629758201

[12] SimilarTech. (2014, 11) Top javascript technologies. SimilarTech. [Online; accessed 26-November-2014]. Available: https://www.similartech.com/categories/javascript

[13] A. Osmani, *Developing Backbone.js Applications*, 1st ed. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, Inc., 5 2013.

[14] Backbone.js. (2014, 12) Projects and companies using backbone. Backbone.js. [Online; accessed 15-December-2014]. Available: https://github.com/jashkenas/backbone/wiki/Projects-and-Companies-using-Backbone

[15] J. Ashkenas. (2014, 12) Backbone.js. backbonejs.org. [Online; accessed 11-December-2014]. Available: http://backbonejs.org/

[16] J. Runeberg, "To-do with javascript mv* : A study into the differences between backbone.js and angularjs." *Arcada - Nylands svenska yrkeshögskola*, 2013, [Online]. Available: http://publications.theseus.fi/bitstream/handle/10024/57918/Runeberg_Joakim.pdf?sequence=1

[17] J. Pienaar and R. Hundt, "Jswhiz: static analysis for javascript memory leaks." in *JSWhiz: static analysis for JavaScript memory leaks.*, 2013, [Online]. Available: http://ieeexplore.ieee.org.proxy.lnu.se/stamp/stamp.jsp?tp=&arnumber=6495007

[18] Z. Herczeg, G. Loki, T. Szirbucz, and A. Kiss, "Validating javascript guidelines across multiple web browsers." *Nordic Journal of Computing*, vol. 15, no. 1, pp. 18 – 31, 2013, [Online]. Available: http://www.inf.u-szeged.hu/~akiss/pub/pdf/herczeg_validating.pdf

[19] Wikipedia, "Analysis of variance — wikipedia, the free encyclopedia," 2015, [Online; accessed 20-January-2015]. Available: http://en.wikipedia.org/w/index.php?title=Analysis_of_variance&oldid=641119685

[20] ——, "Minification (programming) — wikipedia, the free encyclopedia," 2014, [Online; accessed 18-December-2014]. Available: http://en.wikipedia.org/w/index.php?title=Minification_ (programming)&oldid=623761715

[21] Errorception. (2014, 11) Errorception - painless javascript error tracking. Errorception. [Online; accessed 26-November-2014]. Available: https://errorception.com/

[22] Modernizr. (2014, 11) Modernizr documentation. Modernizr. [Online; accessed 26-November-2014]. Available: http://modernizr.com/docs/

[23] StatCounter. (2014, 12) Top 5 desktop, tablet & console browsers from sept 2013 to sept 2014 | statcounter global stats. StatCounter. [Online; accessed 6-December-2014]. Available: http://gs.statcounter.com/#browser-ww-monthly-201309-201409

[24] G. Inc. (2014, 11) Chrome devtools overview. Google Inc. [Online; accessed 26-November-2014]. Available: https://www.google.com/chrome/browser/canary.html

[25] ——. (2014, 11) Tips and tricks. Google Inc. [Online; accessed 21-November-2014]. Available: https://developer.chrome.com/devtools/docs/tips-and-tricks

[26] Wikipedia, "Virtual machine — wikipedia, the free encyclopedia," 2014, [Online; accessed 3-December-2014]. Available: http://en.wikipedia.org/w/index.php?title=Virtual_machine&oldid=634817528

[27] ——, "Model–view–presenter — wikipedia, the free encyclopedia," 2014, [Online; accessed 15-December-2014]. Available: http://en.wikipedia.org/w/index.php?title=Model%E2%80%93view%E2%80%93presenter&oldid=637558376

[28] ——, "Separation of concerns — wikipedia, the free encyclopedia," 2014, [Online; accessed 15-December-2014]. Available: http://en.wikipedia.org/w/index.php?title=Separation_of_concerns&oldid=627731535

Linnæus University
Sweden

Faculty of Technology
SE-391 82 Kalmar | SE-351 95 Växjö
Phone +46 (0)772-28 80 00
teknik@lnu.se
Lnu.se/faculty-of-technology?l=en