Speed Performance Comparison of JavaScript MVC Frameworks

Alexander Svensson

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Contact information:

Author: Alexander Svensson
Email: alexander_svensson85@hotmail.com
Program: Web programming

Supervisor: Guohua Bai
Email: guohua.bai@bth,se
Department: Ph.D Professor, School of Computing

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

ABSTRACT

Context: Many websites today are very interactive and the users are getting used to sites that change hundreds of elements every second. Often a JavaScript framework is used to build the web site and with many changing elements on the site the need for a JavaScript framework that can handle the fast changes are needed. Each frameworks do it differently to achieving this but most of them do some manipulation with the Document Object Model (DOM).

Objectives: This research will show how fast the selected MCV like JavaScript frameworks (AngularJs, AngularJs 2.0, Aurelia, Backbone, Ember, Knockout, Mithril, Vue) can create, delete and update HTML elements on the screen.

Methods: This research have used Google Chromes TimeLine tool to measure the speed of the frameworks. The test involves creating a HTML table and fill it with a thousand rows of data with six columns. The tables content are tested to see how fast the frameworks can create, update and remove the elements.

Conclusions: Angular 2.0 almost achieved first place in all tests. Angular 1.5 did very good in the update tests and was good in the create elements test. Backbone and Ember did not do so well in the create and update tests but Backbone was the best framework in one of the delete tests. Aurelia got very good results and so did Vue which almost had the same values as Aurelia throughout the tests. Mithril and Knockout performed well in the create test which placed them in the middle among all the selected frameworks. When it came to the update tests Mithril and Knockout also found them self in the middle positions of all the frameworks.

Key Words: JavaScript, Framework, performance, Angular, Aurelia, Backbone, Ember, Knockout, Mithril, Vue.

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

CONTENTS

1. Introduction

Comparing JavaScript framework to each other are a hot topic in the web development communities[27][28] and this research will focus on the performance when working with the DOM. This research will put an extra light on how fast they creates, remove and update element from the DOM to the screen. Web pages with much content on each page like forums, blogs, social media, statistic sites are some examples of sites the may gain from a JavaScript framework that performs well.
Today web browsers are very fast compare to older versions and they are constantly making them faster[17] but not all the users have the newest version or maybe a web browser that doesn't perform so well. With a newer web browser that display a website that have fewer elements/content the speed performance values may not differ so much between the JavaScript frameworks but when the web page content gets really big the performance metrics have much more impact. Web sites like Facebook and Twitter that have a lot of content on each page are site that must have a framework that performs fast update to the screen. Other site that maybe are not so popular but also have much content to display on the screen could gain from selection a fast performing JavaScript framework.

There are already a few similar studies made on web sites but not to this reseach extent and with the rapid update changes the frameworks do each month the performance values could drastically change. Most of the studies that are made are also only focusing on a fewer amount of frameworks.

The selected JavaScript frameworks for this research are selected based on popularity or how new it is and for a more detailed elucidation read the "Selected JavaScript frameworks" section. This study have also focused more on MVC (Model View Controller) like frameworks that usually are the base for the application and can often perform many different tasks. There are a lot of frameworks and to test all of them would be impossible for this research but the once that will be examined are:

- AngularJs 1.5
- AngularJs 2.0
- Aurelia
- Backbone
- Ember
- Knockout
- Mithril
- Vue

There are some other libraries that would have been interesting to include like React and Incremental DOM but they are mostly just used as the View in MVC.
This research will only focus on the JavaScript frameworks rendering speed performance values from the web browser. How fast the framework renders things on the screen are the interesting part. Other aspects of the frameworks are not valuated in this study like usability, learnability,documentation, useful additions/functions that also are important to evaluate when choosing a JavaScript framework.

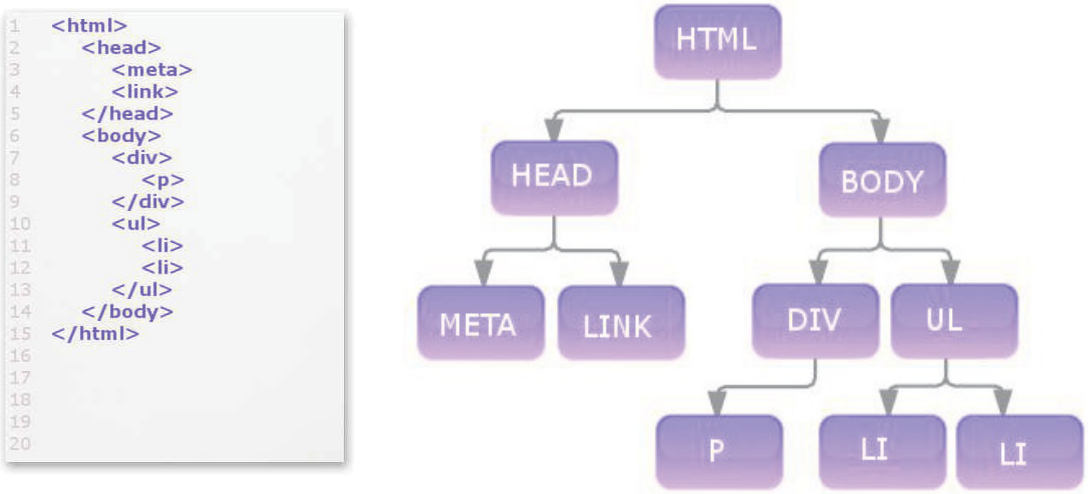
2. Background

In order for us to understand how the selected JavaScript frameworks manage their performance we first have to understand the basic of how a web browser work.
Web browser are built with some main components like browser engine, rendering engine, JavaScript Interpreter, User Interface, Data storage, Networking and UI backend[9]. We will focus on the rendering engine part of the web browser and show the basic how it works. To measure the JavaScript frameworks performance and compare them to each other we have to understand some concepts that are explained below.

2.1 Document Object Model

The Document Object Model (DOM) is a programming interface for HTML, XML and SVG documents [1][2]. The DOM provides a object-oriented structure of the document (HTML,XML or SVG) and each element is turned into a object that have its own methods, properties and values. It also creates a structured interface node tree representation of the document that can be manipulated in some way. When the web browser first receives the page's document text (HTML, XML or SVG) it parses it into the DOM and with a programming language you can access the DOM nodes.

Figure 1.0 - Image: HTML document converted to a Document Object Model tree



The top of the tree(in figure 1.0) is the HMTL object and in the image example above we can see that it have two child nodes which are HEAD and BODY. The image with the tree is just a simplified example of the DOM interface and the HTML element is a child of the document object which is the root of the DOM.
We can easily traverse the node tree and select the elements we want with the DOM interface. We can select a specific element by searching the tree for an elements id

attribute or select many elements by their class name attribute. The nodes that are on the same row and have the same parent node are called siblings and if an element have a parent element it's also called a child node to the parent node.
Each node in the tree represent an element, an attribute, content or some other object[3] and with the DOM API we can add, remove or update the nodes. Each change to the DOM creates a chain of actions the web browser must do before the user can see the new updated web page and it could be time consuming. Some change stakes more time then others and one of the more time expensive ones are when we change the layout which means if we change the geometrics of an element like its height or width.
Today's web pages often uses JavaScript to manipulate the DOM in some way like adding, removing or updating elements in the DOM tree. The DOM was not created to only work with one specific programming language like JavaScript but JavaScript have become one of the most popular language to manipulate the DOM.

2.2 CSS Object Model (CSSOM)

When the web browser is constructing the DOM it finds a link element that references to a external CSS stylesheet and fetches it. If there are no stylesheet link element in the HTML markup it uses the web browsers default stylesheet. After the stylesheet is downloaded the web browser converts the CSS styles into a tree structure like the DOM but this time it`s only for the CSS. When computing the final set of styles for any object on the page, the browser starts with the most general rule applicable to that node (e.g. if it is a child of body element, then all body styles apply) and then recursively refines the computed styles by applying more specific rules - i.e. the rules "cascade down"[4]. So a style that effect many object element down the tree will recursively add it to all of them. Let say we have a BODY node in the CSSOM and under it there is a P node which have a child node of a SPAN and now we have a style that only apply to that SPAN element, body > p > span. The CSSOM becomes like a tree map of all the CSS styles combined and tells what style a different element have depending on where in the tree it is.

2.3 Rendering tree

The Rendering tree is a combined tree of the DOM and the CSSOM which contains everything that is going to be displayed on the web page.When we combine the trees an element in the DOM could have the CSS style property from the CSSOM set to "display: none" and then that node will not be included in the Render-tree. The Render tree only include things that will be visible to the users in the web browser so for example the HEAD will not be included. The Render-tree tells the web browser how to layout the page and how to paint it. The name Rendering tree may have different names depending on web browser. The Rendering tree could also be seen as a set of objects being rendered [5] Webkit(Layout engine used in e.g Safari) and Blink(Layout engine used in e.g Chrome and Opera) calls each of those a "renderer" or "render object" while

Gecko(Layout engine for Mozilla used in e.g Firefox, SeaMonkey) calls it a “frame”[5]. If we do anything on a web page like scrolling the page, changing the DOM with JavaScript, CSS animations or basically anything we do on a web page the Rendering tree will be updated around sixty times each second[10]. The rendering of each frame differ for different web browsers and the tree are created a little bit different between the web browser.

2.4 Web browser rendering

Web browser rendering happens every time we interact with the website and each web browser has its own engine to calculate each frame. A web browser engine (sometimes called layout engine or rendering engine)[6] are very complex and the terminology used by the different web engines my differ but they often mean the same thing. Web browsers e.g Google Chrome, Apples Safari uses WebKit as there web engine and e.g Maozilla Firefox, SeaMonkey uses Gecko engine[6]. All web browsers strive to achieve sixty frames per second so that the flow of the page feels smooth for the users. The web browser rendering do different steps like layout and painting the content to the web page so it feels like the page load faster. An example of this is that we see the text on the web page before we see the images but in the old browsers we had to wait until all the images are fully loaded before we could see anything on the page. It’s important to understand that this is a gradual process and engine will not wait until all the HTML is parsed before starting to build and layout the render/frame tree[9].
Each web browser engine have there own way of creating a page and updating each frame but the main flow are basically the same.

Figure 1.1 - Image: The flow of Webkit engines, image inspired by [7][8][9]

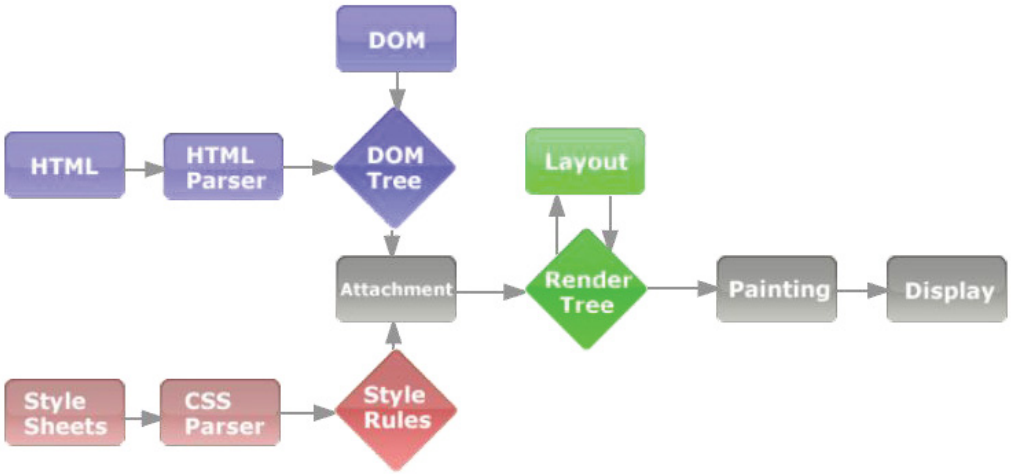
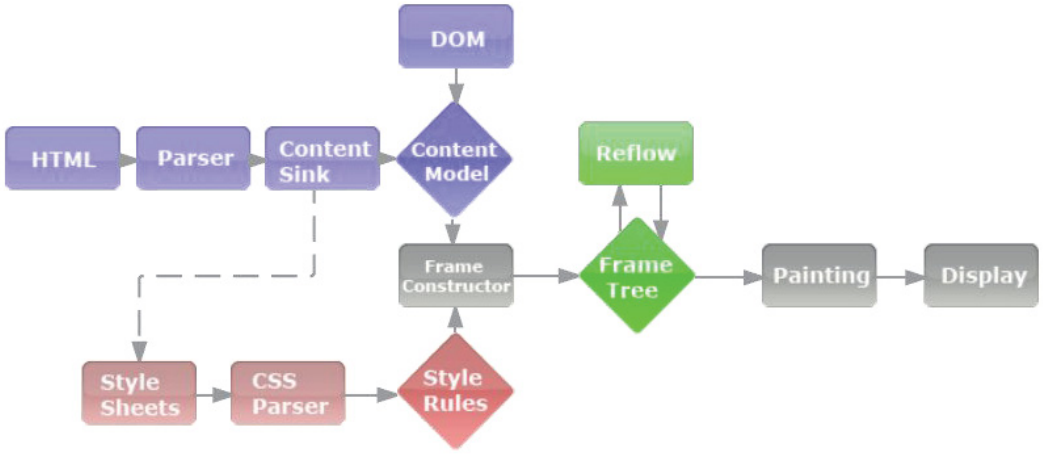


Figure 1.3 - Image: The flow of Mozilla's Gecko engines, image inspired by [7][8][9]



We can see from the images above(figure 1.2 and 1.3) that the DOM and the CSSOM(Styles rules) are combined to the render/frame tree and in the layout/reflow the content(HTML) gets the right position,width and height to the web browser. After that it paints the site after the given style(CSS) like background-color, color, show text, images and more that each node(HTML element) have in the render/frame tree.
So the basic flow of a rendering engine is to first to parse the HTML and CSS to construct the DOM tree and the CSSOM tree. Then the render/frame tree is created and after that the layout/reflow(the position,width and height of the elements on the page) are calculated. The last step is the painting.

2.4.1 Rendering performance

Most devices have a refresh rate of sixty frames per second which the browsers do when an animation, transition(like with CSS), or the page is scrolled[10]. Each frame must be done in 16ms(1 second / 60 = 16.66ms) but the browser needs around 6ms for housekeeping so you only have less then 10ms for each frame[10]. If we take more then 10ms the browser will start dropping frames which will result in juddering images on the screen.
The tests on the selected JavaScript frameworks are going to be done in the web browser Chrome so we are going to explain how they see a frame.The tests are also going to be executed in Chrome Developer Tools(DevTools) which can be used measure the time the web browser spend on different things like JavaScript, style and layout calculations.
Each time we do something with the screen like scrolling the page or some other JavaScript,CSS or Web Animations are being triggered the browser takes different calculations paths depending on the action. So if we do some kind of visual change to the screen a new frame is constructed.

Each framework have already optimized their rendering performance but in this research

the DevTool will be used to see how much time it takes to calculate the JavaScript, Style, Layout, Paint and Composite. It's also very good to use the DevTool to optimize your code and see if there is something that is slowing down your site.

This is how the Google Development team describe flow of a pixel to the screen(the pixel pipeline shown in figure 1.4):

Figure 1.4 - Image: Googles vision of pixel to screen flow. [10]

- **JavaScript**. Typically JavaScript is used to handle work that will result in visual changes, whether it's jQuery's `animate` function, sorting a data set, or adding DOM elements to the page. It doesn't have to be JavaScript that triggers a visual change, though: CSS Animations, Transitions, and the Web Animations API are also commonly used. [10]
- **Style calculations**. This is the process of figuring out which CSS rules apply to which elements based on matching selectors, e.g. `.headline` or `.nav > .nav__item`. From there, once rules are known, they are applied and the final styles for each element are calculated. [10]
- **Layout**. Once the browser knows which rules apply to an element it can begin to calculate how much space it takes up and where it is on screen. The web's layout model means that one element can affect others, e.g. the width of the `<body>` element typically affects its children's widths and so on all the way up and down the tree, so the process can be quite involved for the browser. [10]
- **Paint**. Painting is the process of filling in pixels. It involves drawing out text, colors, images, borders, and shadows, essentially every visual part of the elements. The drawing is typically done onto multiple surfaces, often called layers. [10]
- **Compositing**. Since the parts of the page were drawn into potentially multiple layers they need to be drawn to the screen in the correct order so that the page renders correctly. This is especially important for elements that overlap another, since a mistake could result in one element appearing over the top of another incorrectly. [10]

There are three different paths the web browser takes when a change is made on the screen.

Figure 1.5 - Image: First pixel pipeline flow.[10]

In figure 1.5 the flow of the web browser will do JavaScript, Style, Layout, Paint and Composite. When we for example change a HTML's elements width or height we have changed the Layout and therefor it effect all the other calculations in the pixel pipeline.

Figure 1.6 - Image: Second pixel pipeline flow.[10]

In figure 1.6 the flow of the browser must do calculations on the JavaScript, Style, Paint and last Composite but no Layout calculations is needed. This could happen if we for example change the color or background image on an element.

Figure 1.7 - *Image: Third pixel pipeline flow. [10]*

In figure 1.7 the last flow do no Layout or paint is made and this is the cheapest/fastest. If we scroll a page for example then the browser will take this pixel pipeline.

2.5 Selected JavaScript frameworks

There are many JavaScript frameworks and in this research only a few of them are selected. The criteria when selecting the frameworks was that it either have to be a new framework(from 2014) or in the top fifteen most popular JavaScript frameworks according to Google Trends [11][12][13][14][15][16]. The search term made in Google Trends for each framework are the one that have the best hit search rate(e.g ember js, ember.js ember framework). The selected frameworks must also have a MVC like architecture and not only focus on the View aspect of the framework. Some frameworks might also have been selected on my personal opinion that my be interesting to include in this research. For each selected framework there will be a short motivation why the framework was selected for this research. This research will only focusing on the create, update and remove elements speed performance of the frameworks so other aspect of the framework are not considered like usability or ease of use. The selected frameworks are only tested with there minimum included appendices that otherwise would probably have been included like routing for AngularJ though there are many different scenarios when one framework fits better then another and it doesn't have to be the performance of the framework which is the most important. Some of frameworks might have a really good documentation or other functionalities that might favor one over another. Some JavaScript frameworks are really light weight and focus mostly on performance while other are built to do a lot of different useful functions so its important to know that some frameworks might do a specific work really good. So it could be good to read more about each framework to find out there strengths beside the performance tests which is covered in this study.

What most of the JavaScript frameworks have done is to create a rendering engine to manipulate the DOM in some way to gain performance. Working directly with the DOM without using any JavaScript framework the browser must update the DOM for every

request like remove, add or change a element and that is costly for the browser. What most of the frameworks do is to combine some DOM API calls into one so that there is less updates needed.They each have different techniques to work with the DOM and some frameworks work in a similar way but some do better then others.

2.5.1 Angular 1.5

Version used in tests:

- 1.5.3
- Released 25th Mars 2016

AngularJS is by far the most popular JavaScript framework[13] so its obvious the it's included in this research. AngularJS in a Model View Controller(MVC) framework and it could also be used as a Model View ViewModel (MVVM). Angular first start by reading the HTML files and interpreters the custom Angular tag elements and binds the model data from JavaScript variables(two-way-binding). With angular you can separate small parts of the HTML page and only manipulate that small part without affecting the hole page.
Angular uses so called "Dirty checking" that checks for data changes in the template and creates a watcher on that particular value. If there is a change to the value the page renders the new value. One of the great benefit of this design is that it can watch any data value like an array or just an variable. The downside to this model is that the values are constantly being checked for updates in a JavaScript loop but it does not effect the DOM which makes it not so costly after all[19].

Some words how they self describe their framework:
The following quotes are made by the frameworks official site[26]:

- Angular simplifies application development by presenting a higher level of abstraction to the developer.

- Angular is not a single piece in the overall puzzle of building the client-side of a web application. It handles all of the DOM and AJAX glue code you once wrote by hand and puts it in a well-defined structure.

- AngularJS is a structural framework for dynamic web apps. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly. Angular's data binding and dependency injection eliminate much of the code you would otherwise have to write. And it all happens within the browser, making it an ideal partner with any server technology.

- Angular frees you from the following pains: Registering callbacks, Manipulating HTML DOM programmatically, Marshaling data to and from the UI,Writing tons of initialization code just to get started.

2.5.2 Angular 2.0

Version used in tests:
- 2.0 (Beta v9)
- Released 10th Mars 2016

Angular 2.0 is the new and improved version of Angular in performance but when version 2.0 fist has showed to the public it was meet with some skepticism from a large number of peoples in the community and mainly because of the big changes. Some of the changes are that the Scope object and Controllers are removed to be replaced with Components and Directives.

Some words how they self describe their framework:
The following quotes are made by the frameworks official site[26]:

- Angular simplifies application development by presenting a higher level of abstraction to the developer.

- Angular is not a single piece in the overall puzzle of building the client-side of a web application. It handles all of the DOM and AJAX glue code you once wrote by hand and puts it in a well-defined structure.

- AngularJS is a structural framework for dynamic web apps. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly. Angular's data binding and dependency injection eliminate much of the code you would otherwise have to write. And it all happens within the browser, making it an ideal partner with any server technology.

- Angular frees you from the following pains: Registering callbacks, Manipulating HTML DOM programmatically, Marshaling data to and from the UI,Writing tons of initialization code just to get started.

2.5.3 Aurelia

Version used in tests:
- 1.0.0 (Beta 1.2.2)
- Released 13th April 2016

Aurelia is one of the newer JavaScript frameworks and it have some similarities to Angular 2.0. It has a MVVM architecture and is meant to be highly modularized where you only include what you need. It uses Polyfills that enables older web browsers to

utilize newer modules/inventions to work even if the user have an older web browser.
It also utilizes two-way-binding like Angular do which makes it easy and flexible to work with the data.

Some words how they self describe their framework:
The following quotes are made by the frameworks official site [29]

- With its strong focus on developer experience, Aurelia can enable you to not only create amazing applications, but also enjoy the process. We've designed it with simple conventions in mind so you don't need to waste time with tons of configuration or write boilerplate code just to satisfy a stubborn or restrictive framework. You'll never hit a roadblock with Aurelia either. It's been carefully designed to be pluggable and customizable.

- Over the last 10 years we've labored in building a variety of front-end libraries and frameworks on several different platforms. These libraries have been used to develop thousands of applications for virtually every industry. We've harnessed this rich experience and used it to build Aurelia, the most advanced and developer friendly front-end framework today. We're sure you're going to love it.

2.5.4 Backbone.js

Version used in tests:
- 1.3.3
- Released 5th April 2016

Backbone are one of the fist JavaScript frameworks that became really popular but in the resent years the popularity seams drop for each year[12] but it’s still one of the most popular frameworks. Backbone uses the same approach as Ember to with object observation listeners for changes on the data.

Some words how they self describe their framework:
The following quotes are made by the frameworks official site[]

- Backbone.js gives structure to web applications by providing models with key-value binding and custom events, collections with a rich API of enumerable functions,views with declarative event handling, and connects it all to your existing API over a RESTful JSON interface.

2.5.5 Ember

Version used in tests:
- 2.5.0
- Released 11th April 2016

Ember is one of the more popular frameworks and it's an full-blown MVC framework with a lot of helpful functions. Some of the features they have are routing, custom components with handlebars, loading data from server helpers. With Ember you create Ember Objects that store data and then the object is observed for changes witch is updated to the screen if a change occur. In a way each data model have a listener attached to it and when that data change only that data model event occur.

Some words how they self describe their framework:
The following quotes are made by the frameworks official site[20]

- Ember.js is built for productivity. Designed with developer ergonomics in mind, its friendly APIs help you get your job done—fast.

- Ember's Handlebars integrated templates that update automatically when the underlying data changes.

- Don't waste time making trivial choices. Ember.js incorporates common idioms so you can focus on what makes your app special, not reinventing the wheel.

2.5.6 Knockout

Version used in tests:
- 3.4.0
- Released 17th November 2015

Knockout is one of the first JavaScript frameworks that became popular in the beginning of 2011 and is still quite popular according to Google trends[13]. It is categorized as a MVVM (Model View ViewModel) framework where the ViewModel is observing data changes either in the View or Model.

Some words how they self describe their framework:
The following quotes are made by the frameworks official site[24]

- Knockout is a JavaScript library that helps you to create rich, responsive display and editor user interfaces with a clean underlying data model. Any

time you have sections of UI that update dynamically (e.g., changing depending on the user's actions or when an external data source changes), KO can help you implement it more simply and maintainably.

- Easily associate DOM elements with model data using a concise, readable syntax.

- Automatic UI refreash, When your data model's state changes, your UI updates automatically.

- Dependency tracing, Implicitly set up chains of relationships between model data, to transform and combine it.

- With the templaeing you can quickly generate sophisticated, nested UIs as a function of your model data.

2.5.7 Mithril

Version used in tests:
- 0.2.3
- Released 13th Mars 2016

Even though Mithril is not the most popular framework right now its still quite new compare to some other famous frameworks like Knockout, Backbone and Ember. Mithril is a small MVC frame work that is meant to be as simple as possible and lightweight. Mithril leverages developer experience with server-side MVC frameworks, and in many ways, is very similar in scope to AngularJS[23]. The main difference is the way they handle their templates.

Some words how they self describe their framework:
The following quotes are made by the frameworks official site[23]

- Mithril is a client-side MVC framework - a tool to organize code in a way that is easy to think about and to maintain.

- Fast Virtual DOM diffing and compilable templates.

- Intelligent auto-redrawing system.

- The goal of the framework is to make application code discoverable, readable and maintainable, and hopefully help you become an even better developer.

2.5.8 Vue

Version used in tests:

- 1.0.21
- Released 8th April 2016

Vue is one of the newer JavaScript frameworks and is growing fast in popularity according to Google trends[12] and there are many reasons for that. Vue have much in common with Angular 2.0 which is one of the reason for its popularity but Vue is also easier to use according to Vue's official website[25]. Vue have a clear separation between directives and components which have been a confusing topic for many Angular users. To learn how to use Vue is very easy and the syntax they use is very similar to standard JavaScript code.

Some words how they self describe their framework[25]:

- Vue is fast! Precise and efficient async batch DOM updates.
- Vue is simpel! Write some HTML, grab some JSON, create a Vue instance, that's it.
- Reactive! Expressions & computed properties with transparent dependency tracking.
- Components! Compose your application with decoupled, reusable components.

3. Method

3.1 Research questions

Research questions:

RQ 1 How fast does the selected MVC JavaScript frameworks create new HTML elements on a web page.

RQ 2 How fast does the selected MVC JavaScript frameworks update HTML elements on a web page.

RQ 3 How fast does the selected MVC JavaScript frameworks remove HTML elements on a web page.

3.2 Literature Review Design

For this research most of the information are collected from the frameworks official web sites and from Google's development page. No relevant research studies where found that compare different MVC JavaScript frameworks from these scientific databases:

- Diva portal
- Google Scholar
- Summon BTH
- IEEE Xplore
- ACM Digital Library

There a few similar studies made on web bloggs and web sites but not in this size.
The information for the selected JavaScript frameworks was found on the official frameworks websites and how to setup the framework followed there guidelines. Most of the literature in the background chapter was found from reliable organizations and web communities. The literature for organizing and setting up the test that was performed in this research was mainly found on Googles development web page.
The selected literature must have been written from year 2011 and contain the author who wrote it or be a well known organization.

3.3 Empirical Study Design

This research have used Chromes Timeline tool[22] which is a part of the Chrome Development Tool to measure the time it took for each frameworks in the tests and it's a great tool for revealing the web page performance. With the Timeline tool we can measure the number of frames per second and the total time it took for each test. In the background chapter 2.4.1 there was some explanation of what the Chromes Timeline tool

can monitor. Chromes Timeline tool is the only web browser tool(of my knowledge) that can capture the time it took for a action and to capture the rendering speed of a web page. Chrome have a great documentation how to use the tool and its free to use. There are some other tools that this research could have used but most of them cost money and unfortunately that is not an option. There are also some tool made up by communities but often the documentation is poor and its hard to understand whats really going on under the hood.

This research only capture the frameworks speed when performing an action that changes the DOM. All frameworks are installed with their minimum requirements so for example Angular Routing will not be included. The setup of each framework are made from each frameworks official web sites and the code are following there guide lines. Each framework also follows there own guides how to proper setup it for a production version and not for a development version.

The frameworks will be put through a series of tests to show how well they performs when changing the DOM elements.
The tests consist of a HTML table with no data to start with and these actions are performed:

1. Create rows (1000 rows)
2. Delete all rows (1000 rows)
3. Update all rows (1000 rows)
4. Update part of rows (1/3 of the rows)
5. Delete part of rows (1/3 of the rows)

The table have 1000 rows and each row consist of six columns of random generated numbers from a JavaScript loop. The values are created locally with JavaScript so no values are fetched from a database.

Each create/delete/update/update part/delete part for every framework are done 10 times and then the worst score are removed. The worst score was removed it was sometimes so far from the other 9 results and that could have been caused by other factors then the framework. After that the 9 scores are calculated to get the average score.

The test are done with:

- Google Chrome Version 50.0.2661.94
    - Chromes Incognito mode(for a “clean” browser)
- Windows 8 64bit
- Intel Core i5-42000 CPU 1.60GHz 2.30Ghz
- 12 GB RAM

The computers CPU usage have been under 3% and Disk usage have been under 1% for the test to run. Otherwise a bad result could have been caused by other programs interfering with the test.
Google Chrome was reinstalled so that all plugins where removed that could have slowed down the browser and it was run in Incognito Mode.

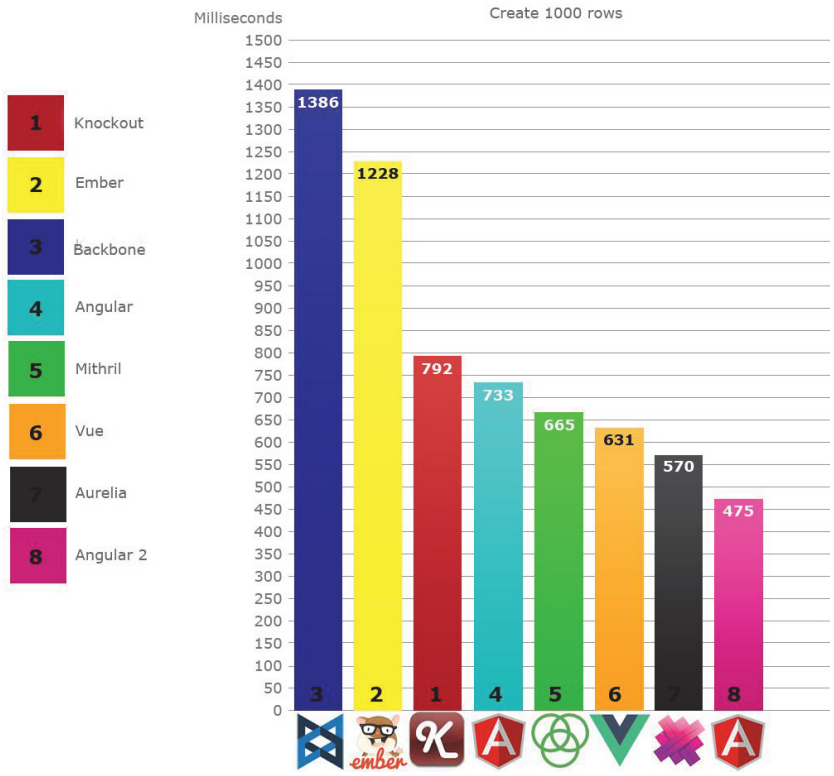
4. Results

4.1 Data results and metrics

4.1.1 Result from create 1000 rows

In the first test a HTML table was created with a thousand rows and the time it took was measured. Creating all the DOM elements takes the longest time of all the tests and some of the frameworks manage this better then others. The frameworks that monitor object model changes (Backbone, Ember) took a little bit longer time to create then the other frameworks.

Figure 1.8 - Image: Average score from ten test runs.



Notes from some of the frameworks:

Angular 2.0 Scored the best result of the selected frameworks and all the test had even results and were stable with no really bad result.

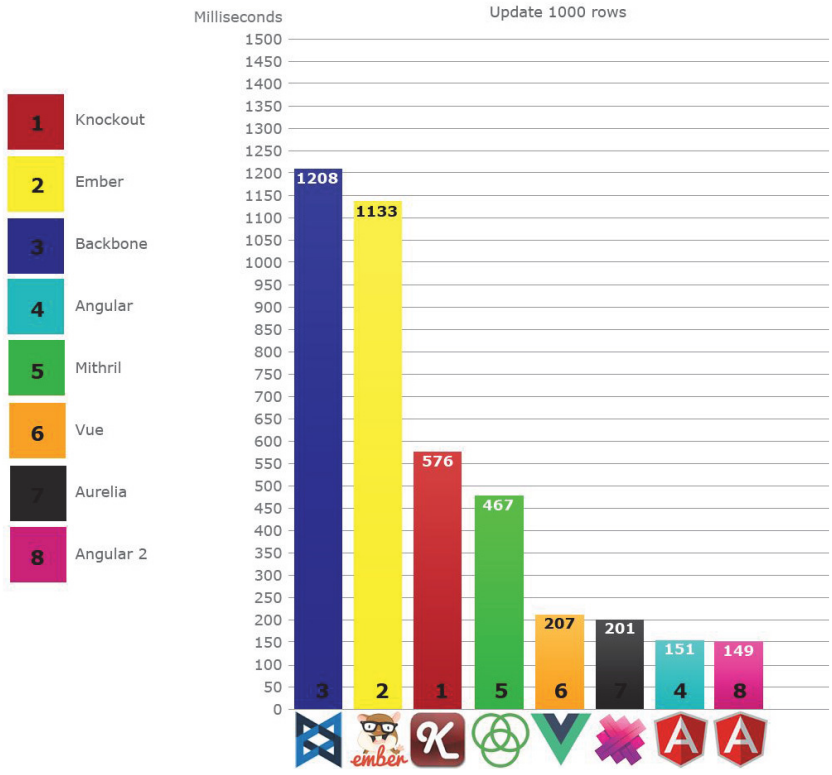
Angular Just lite Angular 2.0 the all the results were very stable and even with no really bad result. But compare to the new version it was 258 milliseconds slower.

Aurelia	Aurelia also did score very well here and it was a little bit expected because it's one of the newer frameworks.
Backbone	In Backbone it's easy to use JavaScript or JQurey to directly change a element and alter its View which could cause a problem. Many people append the Models data directly to the DOM for each Model and this will result in a reflow of the page for each Model. So here we had to create a separate DOM node tree and append all the Models data to it and when all is done we only append one DOM fragment to the real DOM. The result could have been even worse if we did not do this change but Backbone makes it so easy working directly with the DOM so sometime its easy to forget to optimize the performance.
Ember	Ember creates an object for each Models data and then observe it for changes but this also makes it a little bit slower. Although Ember did not score so well here they have a really nice feature to fetch data directly from a API and into the model.
Knockout	Knockout did score quite well here but the result would have been a lot worse if the "observerArray" was not set to extend({deferred:true}). This in short means that the Model should not update the View for each new Model created instead it should wait until all the elements are created which results in fewer reflows . On the official site this is not highlighted good enough in my opinion so it could easily happen when developing.
Vue	Vue's had some really fast results but it also had some that were not so good but the average result was very good.

4.1.2 Result from update all rows

This test shows how fast the frameworks updates all the thousand rows that was created and the result between the frameworks varied by some big margins.

Figure 1.9 - Image: Average score from ten test runs.



Notes from some of the frameworks:

Angular 2.0 Angular 2.0 manage to update the all the rows really fast and got the best result here as well.

Angular In this test Angular 1.5 had almost the same values compare to version 2.0 and that was a little surprise. Both Angular versions have very fast updates when updating all the data.

Aurelia Aurelia also made a really good result once again and so did Vue.

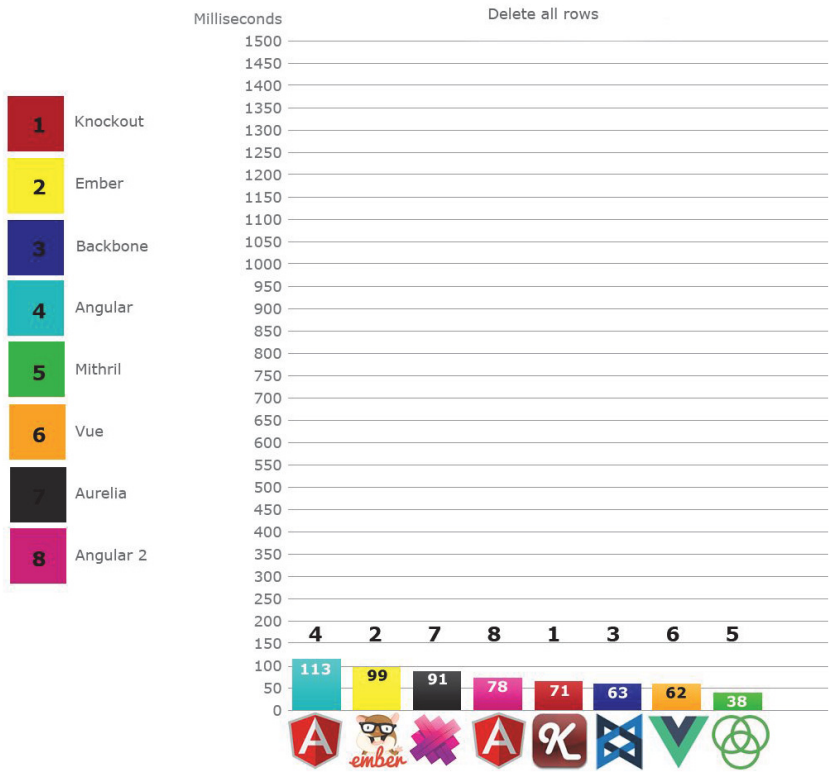
Backbone Backbone observe its Model object changes and each Model object contains a row of number but to update all numbers we have to replace the hole array of number in the object. This makes it just a little bit faster then creating all the rows.

Ember	Ember had the same problem as Backbone making it just slightly faster then create. Changing an arrays value that belongs to an Ember object is not optimal for Ember in this case but in another test scenario the result could have been a little bit better.

4.1.3 Result from delete all rows

In this test all the DOM elements were removed. To remove all the elements only took around eighty milliseconds in average for all the frameworks. The result from this test is probably not the most significant because it doesn't take much time to perform.

Figure 2.0 - Image: Average score from ten test runs.



Notes from some of the frameworks:

Angular — Angular did not score so well here but 113 milliseconds are not much time but in some cases it might matters.

Backbone — Deleting all row went better for Backbone who made the third best result.

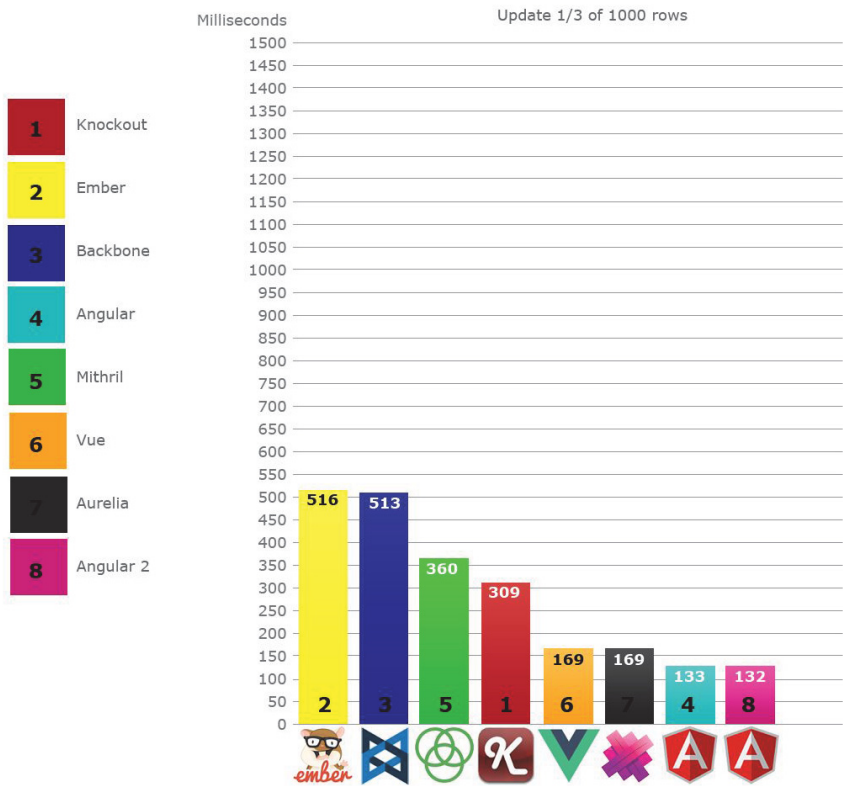
Mithril — Mithril is very fast here and was three times faster then Angular even tho we are talking about 38 milliseconds it can have some effect in some cases. Mithril was three time faster then Angular 1.5

Vue — Vue also made a good result here with an average of 62 milliseconds.

4.1.4 Result from update 1/3 of 1000 rows

In this test the frameworks updated 333 of the thousand rows created. Some frameworks only did small improvements compare to updating all the rows and for some the speed was significantly improved.

Figure 2.1 - Image: Average score from ten test runs.



Notes from some of the frameworks:

Angular — Here Angular 1.5 also almost had the same result as version 2.0. It almost took the same time to update all rows compare to just a third of the rows and it was the same for Vue and Aurelia.

Knockout — Knockout was almost four time slower when updating all the rows compare to Angular 2.0 but when just updating a third it was 2,3 times slower.

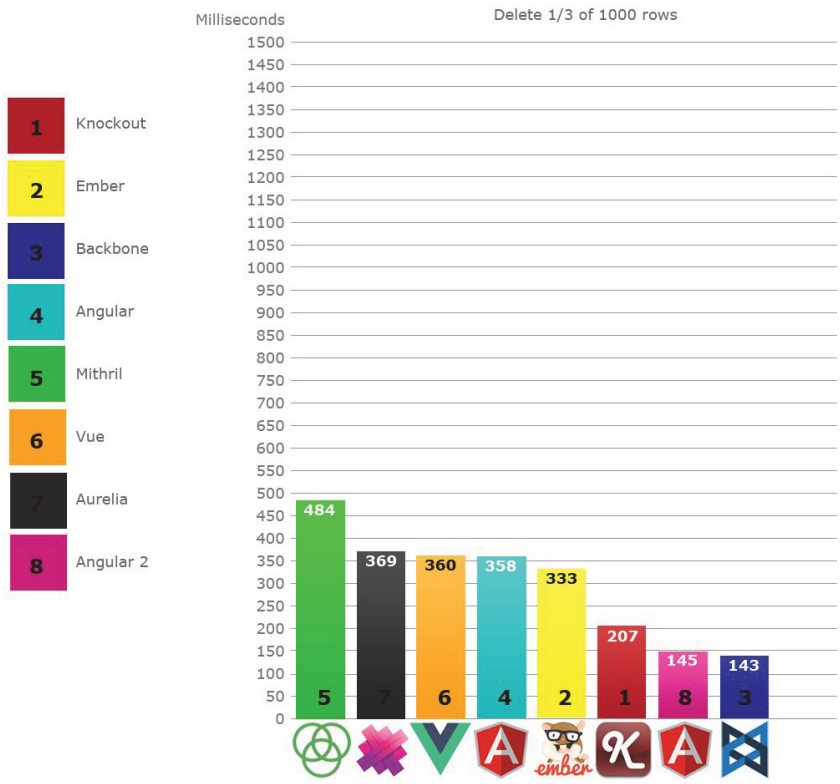
Mithril — Mithril only was 167 milliseconds faster when updating a third of the rows which is not much of a improvement.

Backbone — Made a huge improvement and was more then twice as fast (2.3) compare to updating all rows.

4.1.5 Result from delete 1/3 of 1000 rows

Here 333 rows will be removed of the thousand row that was created and there are some interesting results from the test runs.

Figure 2.2 - Image: Average score from ten test runs.



Notes from some of the frameworks:

Angular 2.0 Here we can see the the new Angular 2.0 have made some great improvements over the older version of Angular.

Backbone Here Backbone shines and show that it can be really fast when removing elements.

Ember In this test Ember made some improvements compare to some other tests made in this study.

Mithril Mithril was by far the fastest framework when deleting all the rows but in this test it scored the worst result. But is was not that much slower then some of the other popular or new frameworks.

5. Analysis

5.1 Summary

When we have analyzed the tests were elements were created, updated and deleted we now have a much clearer picture how well the selected JavaScript frameworks performs. There were some frameworks that performed better then others but we have to keep in mind that it was for these specific test cases. We also have understand that some of the selected frameworks are oriented to help the developers solve some specific problems or tasks before we evaluate the results. Some frameworks might focus on learnability so the users can learn very fast and others might focus on performance.

5.1.1 Research Question 1 - Create elements

Building up the DOM is the most costly action and the result showed that there are some frameworks that manage this much better then others.

Ember and Backbone creates an own object that contains the data and then it observes any changes made on that object before its rendered to the screen. They both had by far the slowest speed when creating the elements but they both have a very easy and fast way to initiate data from a database into their objects.

The Angular frameworks did a good job and it was no surprise that version 2.0 was the fastest but we thought that version 1.5 would have performed a little bit better although it was pretty even between some of the frameworks. Angular 2.0 did address some of the performance problems that version 1.5 had like the data observer checker loop which have made it faster.

Aurelia and Vue were two frameworks that we didn't have much knowledge about before the study that surprised us with there good performance. They are relatively new frameworks which might have indicated that they would get some good performance results in the tests. Vue is really easy to use and the learning curve is fast. Aurelia utilizes new technologies like ES6, TypeScript and is build in pluggable way which makes it easy to include routing, animations, virtualization and more.

We didn't expect Knockout to perform so good because its one of the oldest frameworks in this research and the result was not far from Anuglar 1.5 in some tests.
Mithril did manage to create all the element faster then Angular 1.5 and didn't get that good result that we expected before the tests but it was also not that bad compare to other frameworks.

5.1.2 Research Question 2 - Update elements

There were four frameworks that manage to update really fast and they were Vue, Aurelia and both versions of Angular. These four frameworks are build in a similar way and they are often comparing themselves to one another. All of the four frameworks updated all the rows very fast compare to the others and when updating 1/3 of the rows they also made the best result. We were surprised that Angular 1.5 was as fast as version 2.0.

Before the test run we expected that Mithril should do better although the result was not that bad but compare to the fastest once there was some margin between them. There was not so much speed difference between updating the hole table and updating 1/3 of the rows in Mithril's tests.

Knockout made some real improvement from updating all the rows to only update 1/3. It was only around twice as slow compare to Angular 2.0 in the update 1/3 test but almost four times slower when updating all the rows.

Both Backbone and Ember almost took the same time when creating all the elements to updating all. Updating 1/3 of the rows was obvious better but they were still far from the fastest frameworks.

5.1.3 Research Question 3 - Delete elements

When deleting all elements in the table one might argue that this has such a small effect and that is true in most cases but there might be some scenarios were it could be helpful. When deleting only parts of the DOM things might be more interesting performance wise.

Mithril was very fast and the fastest of the frameworks when deleting the whole table and it almost did it twice as fast as the second fastest framework Vue. The slowest framework was Angular 1.5 with an average of 113 milliseconds and that was a surprise according to us. Mithril was the slowest framework when deleting 1/3 of the table and that did I not expect.

When all the results was gathered from the delete 1/3 of the rows test we were surprised to see that Backbone was the fastest framework because its not one of the newest frameworks. It was more then twice as fast as Aurelia, Angular 1.5 and Vue. Angular 2.0 was only 2 milliseconds slower then Backbone and it was much faster then version 1.5 which is an improvement compare to the old version.

Knockout did well in both tests when deleting the whole table and when deleting 1/3 of the table.

5.2 Validity Threats

There are many ways to measure framework performance speed and this study only shows the result from the specified test cases.
Some of the frameworks were new to me so the test for them are setup according to the frameworks own guidelines and recommendations. This could potential have affected the outcome of the result in some way. There could also be some tweaks that improve the performance of the framework that is not mentioned in their guides that is not implemented in the code.
Though there might be things that could effect the outcome of the results made in this research we can see the result more as an indication of how well the frameworks performs.

6. Conclusion

6.1 Summary

This study have investigated how fast the selected JavaScript frameworks can create, delete and update HTML elements. The result from the test are an indication of how well the frameworks performs thought there are many other factors and tests to include to get a fair evaluation. Things like learnability, documentation, community, support, maintainability, type of project, framework size and ease of use are also important factors to evaluate when selecting a framework.

Angular 2.0 was the framework with the best overall result and performed very good on all test cases. It almost received first place in all tests out of the eight frameworks. The framework had made some improvements over its older version 1.5 in creating and deleting elements.

Angular 1.5 managed to create a thousand elements in 733 milliseconds which is 258 milliseconds slower then version 2.0 and it received fifth place for it. Angular 1.5 and version 2.0 almost had the same result when updating elements. When deleting elements it made an average result.

Aurelia made the second best result when creating elements and was almost as fast as the Angular frameworks when it came to updating elements. Four frameworks almost got the same result when deleting elements and Aurelia was one of them which is an average result event though it was the second slowest framework.

Backbone was the slowest framework when it came to creating and updating elements. It was almost three time slower then Angular 2.0 when creating elements and almost eight time slower when updating all elements. Deleting elements was better for Backbone and it got the best result when deleting only a portion of the elements.

Ember did not do so well when creating and updating elements but in the deleting test it made an average result. Ember and Backbone almost made the same result when creating and updating elements.

Knockout was not the fastest framework when it came to creating and updating elements but it was much faster then Ember and Backbone. In the creating elements test it also only were 59 milliseconds slower then Angular 1.5. Deleting elements it was one of the fastest framework.

Mithril had the fourth best result when creating elements and was faster then Angular 1.5. When it came to updating Mithril made an average result. It was the fastest framework when deleting all elements but it was also the slowest in the deleting 1/3 of rows.

Vue was one of the fastest framework when creating and updating elements. It also made

a good result when deleting elements.

Even though there were some big differences in the tests between the frameworks we have to keep in mind that the test were quite big with a thousand rows and six columns. Normally the changes on a website is much smaller so there will not be that many milliseconds between the frameworks but when the number of elements increase performance matters.

6.2 Future Work

Performing tests like this study have made is very time consuming and with the constant changes made by the frameworks the result might only be valid for a specific time frame. It would have been interesting to include a “to do list” test that creates, updates and removes elements in the list. The result of the test would probably be similar to this study but it could have been a lite bit different in that use case.
Another interesting use case would have been to see how fast the frameworks could fetch data from a data base and rendering it out on he screen.

7. REFERENCES

[1] Mozilla Developer Network - Document Object Model (DOM).
Written by: ...
Date: ...
Link: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

[2] w3.org - Document Object Model (DOM).
Written by: ...
Date: ...
Link: https://www.w3.org/TR/WD-DOM/introduction.html

[3] Introcuction to the Document Object Model
Written by: ...
Date: ...
Link: http://www.brainjar.com/dhtml/intro/

[4] Google Developer CSS Object Model
Written by: ...
Date: ...
Link: https://developers.google.com/web/fundamentals/performance/critical-rendering-path/constructing-the-object-model?hl=en

[5] What Every Frontend Developer Should Know About Webpage Rendering
Written by: Alexander Skutin
Date: 14 May 2014
Link: http://frontendbabel.info/articles/webpage-rendering-101/

[6] Wikipedia - Web browser engine
Written by: Wikipedia organization
Date: -
Link: https://en.wikipedia.org/wiki/Web_browser_engine

[7] How browsers work
Written by: -
Date: -
Link: http://taligarsiel.com/Projects/howbrowserswork1.htm

[8] Blink Rendering Engine
Written by: Hyungwook Lee
Date: 26 Jan 2014
Link: http://www.slideshare.net/HyungwookLee/mobilebrowserinternal-20140122

[9] How Browsers Work: Behind the scenes of modern web browsers
Written by: Taki Garsiel and paul Irish
Date: 5 Aug 2011
Link: http://www.html5rocks.com/en/tutorials/internals/howbrowserswork/

[10] Google Developers - Rendering performance
Written by: Paul Lewis
Date: -
Link: https://developers.google.com/web/fundamentals/performance/rendering/?hl=en

[11] Google Trends
Written by: -
Date: April 2016
Link: https://www.google.com/trends/explore#q=vue.js%2C%20aurelia%20js%2C%20polymer%20js%2C%20meteor%20js%2C%20mercury%20js&cmpt=q&tz=Etc%2FGMT-2

[12] Google Trends
Written by: -
Date: April 2016
Link: https://www.google.com/trends/explore#q=vue.js%2C%20ember%20js%2C%20mithril%20js%2C%20backbone%20js%2C%20knockout%20js&cmpt=q&tz=Etc%2FGMT-2

[13] Google Trends
Written by: -
Date: April 2016
Link: https://www.google.com/trends/explore#q=vue.js%2C%20angularjs%2C%20react%20js&cmpt=q&tz=Etc%2FGMT-2

[14] Google Trends
Written by: -
Date: April 2016
Link: https://www.google.com/trends/explore#q=vue.js%2C%20asana%20Luna%20js%2C%20Agility%20js%2C%20Choco%20js%2C%20ExtJS&cmpt=q&tz=Etc%2FGMT-2

[15] Google Trends
Written by: -
Date: April 2016
Link: https://www.google.com/trends/explore#q=vue.js%2C%20Jamal%20js%2C%20PureMVC%2C%20TrimJunction%2C%20CorMVC&cmpt=q&tz=Etc%2FGMT-2

[16] Google Trends
Written by: -
Date: April 2016
Link: https://www.google.com/trends/explore#q=vue.js%2C%20batman%20js%2C%20Sammy.js%2C%20Eyeballs%20js%2C%20ActiveJS&cmpt=q&tz=Etc%2FGMT-2

[17] pcmag.com - The fastest browser
Written by: Michael Muchmore
Date: January 2013
Link: http://www.pcmag.com/article2/0,2817,2413632,00.asp

[18] auth0.com - React/Ember/Incremental DOM
Written by: Sebastian Peyrott
Date: November 2015
Link: https://auth0.com/blog/2015/11/20/face-off-virtual-dom-vs-incremental-dom-vs-glimmer/

[19] teropa.info - Changes and its detection in JS frameworks
Written by: Tero Parvianinen
Date: Mars 2015
Link: http://teropa.info/blog/2015/03/02/change-and-its-detection-in-javascript-frameworks.html

[20] Ember official website
Written by: Ember team
Date: -
Link: emberjs.com

[21] Backbone.js official web site
Written by: Backbone.js team
Date: -
Link: http://backbonejs.org/

[22] Google Timeline tool
Written by: Goole Chrome team
Date: -
Link: https://developers.google.com/web/tools/chrome-devtools/profile/evaluate-performance/timeline-tool

[23] Mithril official website
Written by: Mithril team
Date: -
Link: http://mithril.js.org/

[24] Knockout official website
Written by: Knockout team
Date: -
Link: http://knockoutjs.com

[25] Vue official website
Written by: Vue team
Date: -
Link: http://vuejs.org

[26] Angular official website
Written by: Angular team
Date: -
Link: http://angularjs.org

[27] Airpair
Written by: Uri Shaked
Date: -
Link: https://www.airpair.com/js/javascript-framework-comparison

[28] Tutorialzine.com
Written by: Martin Angelov
Date: 11 Dec 2015
Link: http://tutorialzine.com/2015/12/the-languages-and-frameworks
-you-should-learn-in-2016/

[29] Aurelia official website
Written by: Aurelia team
Date: -
Link: http://aurelia.io/

Appendix A

Code for each selected framework:

Angular 1.5

app.js

```
var myApp = angular.module('myApp',[]);

myApp.factory('NumbersFactory', function(){
 var banners = [];
 var bannerObj = {
   image:"",
   name:"",
   description:"",
   rating: 0,
   logo:""
 }

 return {
   create: function(){
     for(var i=0; i < 100; i++){

     }
     return this.banners;
   },
   delete: function(){
     numbers = [];
     return numbers;
   },
   update: function(){
     for(var p=0; p < numbers.length; p++){
       for(var t=0; t < 6; t++){
         numbers[p][t] = "updated!";
       }
     }
     return numbers;
   },
   deletePart: function(){
     for(var p=0; p < numbers.length; p += 3){
       numbers.splice(p,1);
     }
     return numbers;
   },
   updatePart: function(){
     for(var p=0; p < numbers.length; p += 3){
       for(var t=0; t < 6; t++){
         numbers[p][t] = "updated!";
       }
     }
     return numbers;
   }
 }
});

myApp.controller('Test3Controller', function($scope,NumbersFactory) {
```

```
 $scope.create = function(){
   $scope.numbers = NumbersFactory.create();
 }

 $scope.delete = function(){
   $scope.numbers = NumbersFactory.delete();
 }

 $scope.update = function(){
   $scope.numbers = NumbersFactory.update();
 }

 $scope.updatePart = function(){
   $scope.numbers = NumbersFactory.updatePart();
 }

 $scope.deletePart = function(){
   $scope.numbers = NumbersFactory.deletePart();
 }
});
```

Angular 1.5 test.html

```
<!DOCTYPE html>
<html ng-app="myApp">
<head lang="en">
 <script type="text/javascript"
src="../../frameworks/angular1.5.3/angular.min.js"></script>
 <script type="text/javascript"
src="../../frameworks/angular1.5.3/test1/app.js"></script>
 <link rel="stylesheet" type="text/css" href="../../stylesheets/bootstrap.min.css">
</head>
 <meat charset="UTF-8">
 <title>Start page</title>
</head>
<body>
 <div class="container">
   <div class="row">
     <div class="col-xs-12">
       <h3>Angular Framwork v1.5.3 Performance</h3>
       <h4>Test nr 1</h4>
       <div ng-controller="Test1Controller">
         <p>
           <button class="btn btn-info" ng-click="create()">Create</button>
           <button class="btn btn-info" ng-click="delete()">Delete</button>
           <button class="btn btn-info" ng-click="update()">Update</button>
           <button class="btn btn-warning" ng-click="deletePart()">Delete 1/4</button>
           <button class="btn btn-warning" ng-click="updatePart()">Update 1/4</button>
         </p>
         <p class="alert alert-info"><b>Number of rows: {{numbers.length}}</b></p>

         <table>
           <tr ng-repeat="number in numbers track by $index">
             <td>{{number[0]}}</td>
             <td>{{number[1]}}</td>
```

```
            <td>{{number[2]}}</td>
            <td>{{number[3]}}</td>
            <td>{{number[4]}}</td>
            <td>{{number[5]}}</td>
          </tr>
        </table>
      </div>
    </div>
  </div>
 </div>
</body>
</html>
```

Angular 2.0

app.components.js

```
(function(app) {
 app.AppComponent =
   ng.core.Component({
     selector: 'my-app',
     templateUrl: '../../../views/angular2/test2.html'
   })
   .Class({
     constructor: function() {
       this.numbers = [];
     },
     create: function(){
       for(var i=0; i < 100; i++){
         this.numbers[i] = [];
         for(var p=0; p < 6; p++){
           this.numbers[i][p] = Math.random();
         }
       }
     },
     delete: function(){
       this.numbers = [];
     },
     update: function(){
       for(var i=0; i < this.numbers.length; i++){
         for(var p=0; p < 6; p++){
           this.numbers[i][p] = "updated!";
         }
       }
     },
     deletePart: function(){
       for(var p=0; p < this.numbers.length; p += 3){
           this.numbers.splice(p,1);
       }
     },
     updatePart: function(){
       for(var p=0; p < this.numbers.length; p += 3){
         for(var t=0; t < 6; t++){
           this.numbers[p][t] = "updated!";
         }
       }
     }
   });
```

```
})(window.app || (window.app = {}));
```

Angular 2.0 main.js

```
(function(app) {
 document.addEventListener('DOMContentLoaded', function() {
   ng.core.enableProdMode();
   ng.platform.browser.bootstrap(app.AppComponent);
 });
})(window.app || (window.app = {}));
```

test2.html

```
<div class="container">
 <div class="row">
   <div class="col-xs-12">
     <h3>Angular Framwork v2.0(Beta) Performance</h3>
     <h4>Test nr 2</h4>
     <div ng-controller="Test1Controller">
       <p>
         <button class="btn btn-info" on-click="create()">Create</button>
         <button class="btn btn-info" on-click="delete()">Delete</button>
         <button class="btn btn-info" on-click="update()">Update</button>
         <button class="btn btn-warning" on-click="deletePart()">Delete 1/4</button>
         <button class="btn btn-warning" on-click="updatePart()">Update 1/4</button>
       </p>
       <p class="alert alert-info"><b>Number of rows: {{numbers.length}}</b></p>

       <table>
         <tr *ngFor="#number of numbers">
           <td>{{number[0]}}</td>
           <td>{{number[1]}}</td>
           <td>{{number[2]}}</td>
           <td>{{number[3]}}</td>
           <td>{{number[4]}}</td>
           <td>{{number[5]}}</td>
         </tr>
       </table>
     </div>
   </div>
 </div>
</div>
```

Aurelia

app.js

```
export class App {

 constructor(){
     this.numbers = [];
 };

 create(){
   console.log("create");
```

```
    for(var i=0; i < 1000; i++){
      this.numbers.splice(i, 1, []);
      for(var p=0; p < 6; p++){
        this.numbers[i].push(Math.random());
      }
    }
  };
  delete(){
    console.log("delete");
    this.numbers = [];
  };
  update(){
    for(var i=0; i < this.numbers.length; i++){
      this.numbers.splice(i, 1, []);
      for(var p=0; p < 6; p++){
        this.numbers[i].splice(p,1,"updated!");
      }
    }
  };
  updatePart(){
    for(var i=0; i < this.numbers.length; i += 3){
      this.numbers.splice(i, 1, []);
      for(var t=0; t < 6; t++){
        this.numbers[i].splice(t,1,"updated!");
      }
    }
  };
  deletePart(){
    for(var p=0; p < this.numbers.length; p += 3){
      this.numbers.splice(p,1);
    }
  };
}
```

Aurelia app.html

```
<template>
 <require from="bootstrap/css/bootstrap.css"></require>
 <div class="container">
   <div class="row">
     <div class="col-xs-12">
       <h3>Aurelia Framwork v1.0.0 Performance</h3>
       <h4>Test nr 1</h4>
       <div ng-controller="Test1Controller">
         <p>
           <button class="btn btn-info" click.trigger="create()">Create</button>
           <button class="btn btn-info" click.trigger="delete()">Delete</button>
           <button class="btn btn-info" click.trigger="update()">Update</button>
           <button class="btn btn-warning" click.trigger="deletePart()">Delete
1/4</button>
           <button class="btn btn-warning" click.trigger="updatePart()">Update
1/4</button>
         </p>
         <p class="alert alert-info"><b>Number of rows: ${numbers.length}</b></p>

         <table>
           <tr repeat.for="rows of numbers">
             <td>${rows[0]}</td><td>${rows[1]}</td><td>${rows[2]}</td><td>${rows[3]}</t
```

```
d><td>${rows[4]}</td><td>${rows[5]}</td>
          </tr>
        </table>
      </div>
    </div>
  </div>
 </div>
</template>
```

Backbone

model.js

```
App = {};

// ================== DATA MODELS ==========================
// =========================================================
var ConterModel = Backbone.Model.extend({
 initialize: function(nrs){
     this.counter = nrs;
 },
 update: function(nr){
   this.counter = nr;
 }
});
var NumberRowModel = Backbone.Model.extend({
 initialize: function(nrs){
   this.numbers = nrs;
 },

});

var NumbersCollection = Backbone.Collection.extend({
 model: NumberRowModel,

 initialize: function(){
 },

 create: function(){
   console.log("create in modle");
 },
 render: function(){
   console.log("NumbersCollection render");
 }
});

App.NumbersCollection = new NumbersCollection;


// ======================= VIEWS ===========================
// =========================================================
var ViewCounter = Backbone.View.extend({
 el: '#counter',

 initialize: function(){
```

```
    this.render();
  },

  render: function(){
    console.log(this.model.counter);
    this.$el.html(this.model.counter);
  }
});
App.counter = new ConterModel(0);
App.ViewCounter = new ViewCounter({model:App.counter});

var ViewRow = Backbone.View.extend({

  tagName: 'tr',
  template: _.template($('#row-template').html()),

  initialize: function(){
      this.listenTo(this.model, "change:numbers", this.render);
  },

  render: function(){
    console.log("viewRow render");
    this.$el.html(this.template(this.model.attributes));
    return this;
  },
});

// Store row view so we can delete them
App.viewsRow =[];

var ViewNumbers = Backbone.View.extend({
  el: '#table-body',

  initialize: function(){

  },

  render: function(){
    var container = document.createDocumentFragment();

    this.collection.each(function(model,index){
      App.viewsRow[index]= new ViewRow({
        model:model
      });
      container.appendChild(App.viewsRow[index].render().el);
    });
    this.$el.append(container);
    return this;
  },
  create: function(){
    for(var i=0; i < 100; i++){
      var nrs = [];
      for(var p=0; p < 6; p++){
        nrs[p] = Math.random();
      }
      this.collection.add(new NumberRowModel({numbers: nrs}));
    }
```

```
    App.counter.update(this.collection.length);
    App.ViewCounter.render();
    this.render();
  },
 updatePart: function(){
    this.collection.each(function(model,index){

      if(index % 3 == 0){
        var ar = [];
        for(var i=0; i < 6; i++){
          ar[i] = "updated!";
        }
        model.set('numbers', ar);
      }
    });
  },
  deletePart: function(){

    for(var p=0; p < this.collection.length; p += 3){
      this.collection.remove(this.collection.at(p));
      App.viewsRow[p].remove();
    }
    App.counter.update(this.collection.length);
    App.ViewCounter.render();


  },
  delete: function(){
    this.collection.reset();
    this.$el.html("");
  },
  update: function(){
    this.collection.each(function(model){
      var ar = [];
      for(var i=0; i < 6; i++){
        ar[i] = "updated!";
      }
      model.set('numbers', ar);
    });
  }
});
App.ViewNumbers = new ViewNumbers({collection:App.NumbersCollection});

var ButtonView = Backbone.View.extend({
  el: '#buttons',
  initialize: function(){
    this.render();
  },
  events: {
    'click #btn-create': 'create',
    'click #btn-delete': 'delete',
    'click #btn-update': 'update',
    'click #btn-deletePart': 'deletePart',
    'click #btn-updatePart': 'updatePart',
  },
  create: function(){
    App.ViewNumbers.create();
  },
  delete: function(){
    App.ViewNumbers.delete();
  },
```

```
 update: function(){
   App.ViewNumbers.update();
 },
 deletePart: function(){
   App.ViewNumbers.deletePart();
 },
 updatePart: function(){
   App.ViewNumbers.updatePart();
 },
 render: function(){
   this.$el.append(' <button class="btn btn-info" type="button"
id="btn-create">Create</button>');
   this.$el.append(' <button class="btn btn-info" type="button"
id="btn-delete">Delete</button>');
   this.$el.append(' <button class="btn btn-info" type="button"
id="btn-update">Update</button>');
   this.$el.append(' <button class="btn btn-warning" type="button"
id="btn-deletePart">Delete 1/4</button>');
   this.$el.append(' <button class="btn btn-warning" type="button"
id="btn-updatePart">Update 1/4</button>');
 }
});

App.ButtonView = new ButtonView;
```

Backbone test1.html

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="UTF-8">
 <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
 <title>Backbone.js Test</title>
 <link rel="stylesheet" type="text/css" href="../../stylesheets/bootstrap.min.css">
</head>
<body>
 <!-- ========= -->
 <!-- Your HTML -->
 <!-- ========= -->
 <div class="container">
   <div class="row">
     <div class="col-xs-12">
       <h3>Backbone Framwork v1.3.3 Performance</h3>
       <h4>Test nr 1</h4>
       <p id="buttons">
       </p>
       <p class="alert alert-info"><b>Number of rows: <span id="counter"></span></b></p>
       <table id="table-body">
         <script type="text/template" id="row-template">
           <td><%= numbers[0]%></td><td><%= numbers[1]%></td><td><%=
numbers[2]%></td><td><%= numbers[3]%></td><td><%= numbers[4]%></td><td><%=
numbers[5]%></td>
         </script>
       </table>
     </div>
   </div>
 </div>
</body>
```

```
<!-- ========= -->
<!-- Libraries -->
<!-- ========= -->
<script type="text/javascript"
src="../../frameworks/backbone1.3.3/jquery-1.12.3.min.js"></script>
<script type="text/javascript"
src="../../frameworks/backbone1.3.3/underscore.js"></script>
<script type="text/javascript"
src="../../frameworks/backbone1.3.3/backbone.min.1.3.3.js"></script>
<script type="text/javascript"
src="../../frameworks/backbone1.3.3/test1/models.js"></script>
</html>
```

Ember

test1.js

```
import Ember from 'ember';

export default Ember.Controller.extend({
 numbers:[],
 init: function(){
 },
 actions: {
   create() {
     var nrs = [];

     for(var i=0; i < 1000; i++){
       nrs[i] = [];
       for(var p=0; p < 6; p++){
         nrs[i][p] = Math.random();
       }
     }
     this.set('numbers',nrs);
   },
   delete() {
     this.numbers.clear();
   },
   update() {
     for(var p=0; p < this.numbers.length; p++){
       for(var t=0; t < 6; t++){
         this.numbers.get(p).removeAt(0);
         this.numbers.get(p).pushObject('updated');
       }
     }
   },
   deletePart() {
     for(var p=0; p < this.numbers.length; p += 3){
       this.numbers.removeAt(p);
     }
   },
   updatePart() {
     for(var p=0; p < this.numbers.length; p += 3){
       for(var t=0; t < 6; t++){
         this.numbers.get(p).removeAt(0);
```

```
        this.numbers.get(p).pushObject('updated');
      }
    }
  }
 }
});
```

Ember test1.html

```
<!DOCTYPE html>
<html>
<head lang="en">
 <script type="text/javascript"
src="../../frameworks/ember2.5.0/ember2.5.0.min.js"></script>
 <script type="text/javascript" src="../../frameworks/ember2.5.0/test1/app.js"></script>
 <link rel="stylesheet" type="text/css" href="../../stylesheets/bootstrap.min.css">
</head>
 <meat charset="UTF-8">
 <title>Start page</title>
</head>
<body>
 <div class="container">
   <div class="row">
     <div class="col-xs-12">
       <h3>Ember Framwork v2.5.0 Performance</h3>
       <h4>Test nr 1</h4>
       <div>
         <p>
           <button class="btn btn-info" {{action "create"}}>Create</button>
           <button class="btn btn-info" {{action "delete"}}>Delete</button>
           <button class="btn btn-info" {{action "update"}}>Update</button>
           <button class="btn btn-warning" {{action "deletePart"}}>Delete 1/4</button>
           <button class="btn btn-warning" {{action "updatePart"}}>Update 1/4</button>
         </p>
         <p class="alert alert-info"><b>Number of rows: {{numbers.length}}</b></p>

         <table>
           {{#each model as |number|}}
             <tr>
               <td>{{number[0]}}</td>
               <td>{{number[1]}}</td>
               <td>{{number[2]}}</td>
               <td>{{number[3]}}</td>
               <td>{{number[4]}}</td>
               <td>{{number[5]}}</td>
             </tr>
           {{/each}}
         </table>
       </div>
     </div>
   </div>
 </div>
</body>
</html>
```

Knockout

main.js

```
require(['knockout', 'numbersViewModel','rowModel', 'domReady!'], function(ko,
numbersViewModel,rowModel) {
   ko.applyBindings(NumbersViewModel());
});
```

Knockout app.js

```
var App = {};
// Class to represent a row of numbers
App.RowModel = function(numbers) {
   var self = this;
   self.numbers = numbers;
}

// Overall viewmodel for this screen, along with initial state
App.NumbersViewModel = function() {
   var self = this;
       self.rows = ko.observableArray([]);
       self.rows.extend({ deferred: true });

   self.create = function() {
     console.log("create");
     var arr = [];
     for(var i=0; i < 1000; i++){
       var numbers = [];
       for(var p=0; p < 6; p++){
         numbers[p] = Math.random();
       }
       arr[i] = new App.RowModel(numbers);
     }
     self.rows(arr);
   };
   self.deleteAll = function() {
     self.rows.removeAll();
   };
   self.update = function() {
     for(var p=0; p < self.rows().length; p++){
       self.rows.splice(p,1,new
App.RowModel(['updated!','updated!','updated!','updated!','updated!','updated!']));
     }
   }
   self.updatePart = function() {
     for(var p=0; p < self.rows().length; p+=3){
       self.rows.splice(p,1,new
App.RowModel(['updated!','updated!','updated!','updated!','updated!','updated!']));
     }
   };
   self.deletePart = function() {
     for(var p=0; p < self.rows().length; p+=3){
       console.log("u");
       self.rows.splice(p,1);
     }
   };
}

ko.applyBindings(new App.NumbersViewModel());
```

Knockout test1.html

```
<!DOCTYPE html>
<html ng-app="myApp">
<head lang="en">
 <script type="text/javascript"
src="../../frameworks/knockout3.4.0/knockout3.4.0.js"></script>
 <link rel="stylesheet" type="text/css" href="../../stylesheets/bootstrap.min.css">
</head>
 <meat charset="UTF-8">
 <title>Start page</title>
</head>
<body>
 <div class="container">
   <div class="row">
     <div class="col-xs-12">
       <h3>Knockout Framwork v3.4.0 Performance</h3>
       <h4>Test nr 1</h4>
       <div>
         <p>
           <button class="btn btn-info" data-bind="click: create">Create</button>
           <button class="btn btn-info" data-bind="click: deleteAll">Delete</button>
           <button class="btn btn-info" data-bind="click: update">Update</button>
           <button class="btn btn-warning" data-bind="click: deletePart">Delete
1/4</button>
           <button class="btn btn-warning" data-bind="click: updatePart">Update
1/4</button>
         </p><!--<span data-bind="text: rows().length">-->
         <p class="alert alert-info"><b>Number of rows: <span data-bind="text:
rows().length"></span></b></p>
         <table>
           <tbody data-bind="foreach: rows">
             <tr></td><td data-bind="text: numbers[0]"></td><td data-bind="text:
numbers[1]"></td><td data-bind="text: numbers[2]"></td><td data-bind="text:
numbers[3]"></td><td data-bind="text: numbers[4]"></td><td data-bind="text:
numbers[5]"></td></tr>
         </tbody>
         </table>
       </div>
     </div>
   </div>
 </div>
</body>
 <script type="text/javascript"
src="../../frameworks/knockout3.4.0/test1/app.js"></script>
</html>
```

Mithril

app.js

```
var myComponent = {
 model: function(name){
   this.name = name;
   this.numbers = [];
```

```
  this.rows = function(){
    return this.numbers.length;
  }
},
controller: function(){
  var  ctrl = this;
  var create = false;

  ctrl.model = new myComponent.model('MVC');

  ctrl.create = function(){
    for(var i=0; i < 1000; i++){
      ctrl.model.numbers[i] = {};
      for(var p=0; p < 6; p++){
        ctrl.model.numbers[i][p] = Math.random();
      }
    }
    create = true;
    ctrl.getData();
  }

  ctrl.delete = function(){
    ctrl.model.numbers = [];
  }

  ctrl.update = function(){
    for(var p=0; p < ctrl.model.numbers.length; p++){
      for(var t=0; t < 6; t++){
        ctrl.model.numbers[p][t] = "updated!";
      }
    }
  }

  ctrl.deletePart = function(){
    for(var p=0; p < ctrl.model.numbers.length; p += 3){
      ctrl.model.numbers.splice(p,1);
    }
  }

  ctrl.updatePart = function(){
    for(var p=0; p < ctrl.model.numbers.length; p += 3){
      for(var t=0; t < 6; t++){
        ctrl.model.numbers[p][t] = "updated!";
      }
    }
  }

  ctrl.getData = function(){
    if(create){
      return m("table", ctrl.model.numbers.map(function(nr){
     return m("tr", [m("td", nr[0]),
                          m("td", nr[1]),
                          m("td", nr[2]),
                          m("td", nr[3]),
                          m("td", nr[4]),
                          m("td", nr[5])
          ]);
  }));
    }
```

```
   }

   return ctrl;
 },
 view: function(ctrl){
   return m(".container",[
            m(".row",[
              m(".col-xs-12",[
                m("h3","Mithril Framwork Performance"),
                m("h4","Test nr 1"),
                  m("p", [
                      m("button[type=button] .btn btn-info", {onclick: ctrl.create},
"Create"),
                      m("button[type=button] .btn btn-info", {onclick: ctrl.delete},
"Delete"),
                      m("button[type=button] .btn btn-info", {onclick: ctrl.update},
"Update"),
                      m("button[type=button] .btn btn-warning", {onclick:
ctrl.deletePart}, "Delete 1/4"),
                      m("button[type=button] .btn btn-warning", {onclick:
ctrl.updatePart}, "Update 1/4"),
                  ]),
                  m("p.alert alert-info",[
                    m("b","Numbers of rows: "+ctrl.model.rows())
                  ]),
                    ctrl.getData()
                ])
              ])
            ]);
 }
}

m.mount(document.body, myComponent);
```

Mithril test2.html

```
<!DOCTYPE html>
<html>
<head>
 <script type="text/javascript"
src="../../frameworks/mithril0.2.3/mithril.min.js"></script>
 <link rel="stylesheet" type="text/css" href="../../stylesheets/bootstrap.min.css">
 <title>Start page</title>
</head>
<body>
 <script type="text/javascript"
src="../../frameworks/mithril0.2.3/test2/app.js"></script>
</body>
</html>
```

Vue

app.js

```
new Vue({
 el: '#app',
```

```
 data: {
   numbers: []
 },
 methods: {
   create: function(){
     var nrs = [];
     for(var i=0; i < 1000; i++){
       nrs[i] = [];
       for(var p=0; p < 6; p++){
         nrs[i][p] = Math.random();
       }
     }
     this.numbers = nrs;
   },
   delete: function(){
     this.numbers = [];
   },
   update: function(){
     var nrs = [];
     for(var i=0; i < this.numbers.length; i++){
       nrs[i] = [];
       for(var p=0; p < 6; p++){
         nrs[i][p] = "updated!";
       }
     }
     this.numbers = nrs;
   },
   deletePart: function(){
     for(var p=0; p < this.numbers.length; p += 3){
         this.numbers.splice(p,1);
     }
   },
   updatePart: function(){
     for(var p=0; p < this.numbers.length; p += 3){
       for(var t=0; t < 6; t++){
         this.numbers[p].$set(t,["updated!"]);
       }
     }
   }
 }
})
```

Vue test2.html

```
<!DOCTYPE html>
<html>
<head lang="en">
 <script type="text/javascript" src="../../frameworks/vue1.0.21/vue.min.js"></script>
 <link rel="stylesheet" type="text/css" href="../../stylesheets/bootstrap.min.css">
</head>
 <meat charset="UTF-8">
 <title>Start page</title>
</head>
<body>
 <div class="container">
   <div class="row">
     <div class="col-xs-12">
       <h3>Vue Framwork v1.0.21 Performance</h3>
       <h4>Test nr 2</h4>
         <div id="app">
```

```
        <p>
          <button class="btn btn-info" v-on:click="create">Create</button>
          <button class="btn btn-info" v-on:click="delete">Delete</button>
          <button class="btn btn-info" v-on:click="update">Update</button>
          <button class="btn btn-warning" v-on:click="deletePart">Delete 1/4</button>
          <button class="btn btn-warning" v-on:click="updatePart">Update 1/4</button>
        </p>
        <p class="alert alert-info"><b>Number of rows: {{numbers.length}}</b></p>
        <table>
          <tr v-for="number in numbers" track-by="$index">
            <td>{{number[0]}}</td>
            <td>{{number[1]}}</td>
            <td>{{number[2]}}</td>
            <td>{{number[3]}}</td>
            <td>{{number[4]}}</td>
            <td>{{number[5]}}</td>
          </tr>
        </table>
        </div>
     </div>
   </div>
 </div>
</body>
 <script type="text/javascript" src="../../frameworks/vue1.0.21/test2/app.js"></script>
</html>
```