ARCADA

TO-DO WITH JAVASCRIPT MV*

A study into the differences between Backbone.js and AngularJS

Joakim Runeberg

Degree Thesis

Mediekultur

2013

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Mediekultur / Digitala Multimedia
Identifikationsnummer:	9073
Författare:	Joakim Runeberg
Arbetets namn:	To-Do's with JavaScript MV*
Handledare (Arcada):	Owen Kelly
Uppdragsgivare:	

Sammandrag:

Arbetet undersöker skillnader mellan de två mest populära JavaScript ramverken: AngularJS och Backbone.js. Målet med undersökningen är att se ifall det ena gränssnittet är att föredra over det andra, och vilka arkitektoniska skillnader det finns emellan de två gränssnitten. Det här sker genom att använda det öppna projektet TodoMVC, med HTML5s localStorage för att lagra information. I samband med undersökningen utvecklades automatiserade test med hjälp av PhantomJS, ett JavaScript gränssnitt ämnat för automatisering av test. Med hjälp av dessa test jämfördes prestandan mellan de två, till synes identiska applikationerna i en omgivning baserad på WebKit. Undersökningen visade att Backbone.js är bättre än AngularJS på att hantera stora mängder av samtidig data. Jämförelser i hur mycket kod som krävs för att uträtta samma funktionalitet fick AngularJS att framstå som en segrare. De "heap profiling" tests som genomfördes visade sig vara otillräckliga för att kunna användas i undersökningen eftersom skillnaderna i resultaten var för små för att vara signifikanta. Undersökningen visar att skillnaderna i prestanda mellan den två gränssnitten i en vardaglig applikation är så små att de kan förbises, och att de arkitektoniska skillnaderna, som t.ex. dubbelriktad bindning, HTML mallar och tillägg, spelar en mer betydande roll i valet av gränssnitt.

Nyckelord:	AngularJS, Backbone.js, TodoMVC, JavaScript MVC, Model-View-Controller
Sidantal:	64
Språk:	Engelska
Datum för godkännande:	24.4.2013

DEGREE THESIS	
Arcada	
Degree Programme:	Media Culture / Digital Multimedia
Identification number:	9073
Author:	Joakim Runeberg
Title:	To-do's with JavaScript MV*
Supervisor (Arcada):	Owen Kelly
Commissioned by:	

Abstract:

The thesis investigates the differences between the two currently most popular JavaScript architectural frameworks: AngularJS and Backbone.js. The aim of the study is to see if one framework is to prefer over the other, and what the most notable architectural differences are. This is done through the use of the open-source application TodoMVC with HTML5 localStorage as a persistence layer. Tests were developed in PhantomJS to measure the DOM performance between two behaviourally identical applications in a WebKit environment. The tests showed that Backbone.js is better at handling big amounts of simultaneous data than AngularJS. Additional SLOC tests were performed to compare the amount of source-code required to build each application, with the clear winner being AngularJS. Heap Profiling tests to examine memory usage between the two frameworks proved to be an unsuccessful research method because of the minimal differences in the obtained data. The conclusion of the thesis is that the performance differences in a real world application are too small to be of much importance, and that the architectural differences such as two-way data binding, HTML templating and extensions play a more important role when deciding between the two frameworks.

Keywords:	AngularJS, Backbone.js, TodoMVC, JavaScript MVC, Model-View-Controller
Number of pages:	64
Language:	English
Date of acceptance:	April 24, 2013

CONTENTS

Figures

Tables

1 INTRODUCTION

1.1 Research Topic

The goal of this thesis is to examine the two JavaScript MV* frameworks Backbone.js and Angular.js and to highlight differences, advantages and disadvantages.

1.2 Motive for the research

I believe JavaScript MVCs are taking an increasingly important role in modern web development where more and more logic is being moved from the server to the client. JavaScript MVC gained a lot of traction only recently and can still be consider quite immature. Through this project I hope to be able to add both frameworks to my toolkit and to get a better understanding of when to use which framework, and that showing deep knowledge of modern high-end frontend technology will be prove to be a door opener in my career. I can also see myself working with both languages extensively in future personal- and client-projects.

1.3 Target group

This thesis targets and audience that is already familiar with JavaScript and the Model-View-Controller pattern, but still aims to provide enough background for novice JavaScript developers interested in exploring the possibilities and architecture of AngularJS and Backbone.js.

1.4 Background

We live in a world that is becoming increasingly computerized. Just in the last year products such as Pebble (a programmable watch) and Raspberry Pi (a credit-card-sized

single-board computer for consumer programming) have entered the market. The former collected $10,266,845 USD on popular crowd funding site Kickstarter and are backed by Y-Combinator, a start-up incubator notorious for investing in Dropbox, Scribd, Disqus and Airbnb – to name a few hugely popular services. Raspberry Pi is gaining popularity and was recently backed[1] by Google who are giving away 15,000 microcomputers to students around the UK. With Code.org recently making an online appearance, with backing from famous personalities such as President Bill Clinton, President Obama and Mark Zuckerberg, proclaiming that "every student in every school should have the opportunity to learn to code", coding has never been more popular.[2]

Back when personal computers were first introduced, interacting without writing a bit of code or domain-specific language was hardly possible. Over time computers have become user-friendlier and in the process the code is almost completely hidden away from the common consumer.[3] Still, at this day the vast majority of personal computers come equipped with a web-browser capable of processing JavaScript. JavaScript is not to be confused with the programming language Java – or as Azat Mardanov, author of *Rapid Prototyping With JavaScript* puts it: "[JavaScript] has the same relationship with Java as a hamster and a ham".[4] Contrary to languages like Java or C that use strong typing, JavaScript has a loose/weak typing which can make the learning curve less steep for a newcomer.

For many years JavaScript was relegated mostly to simple scripts that performed very basic functions on a page. It was slow and web developers had to resort to proprietary flash to get anything more complex done. Clicking tabs of a webpage such as pictured in fig. 4 required a full-page refresh, and JavaScript code in general was often repetitive and rarely object oriented.

---

[1] *15,000 Raspberry Pis for UK schools - thanks Google!*, Available at: http://www.raspberrypi.org/archives/3158 Accessed: 10.4.2013.
[2] *Anybody can learn*, Available at: http://www.code.org/ Accessed: 10.4.2013.
[3] Mardanov, Azat. 2013, *Rapid Prototyping with JS; Version 0.4*. Page 6.
[4] Mardanov, Azat. 2013, *Rapid Prototyping with JS; Version 0.4*. Page 15.

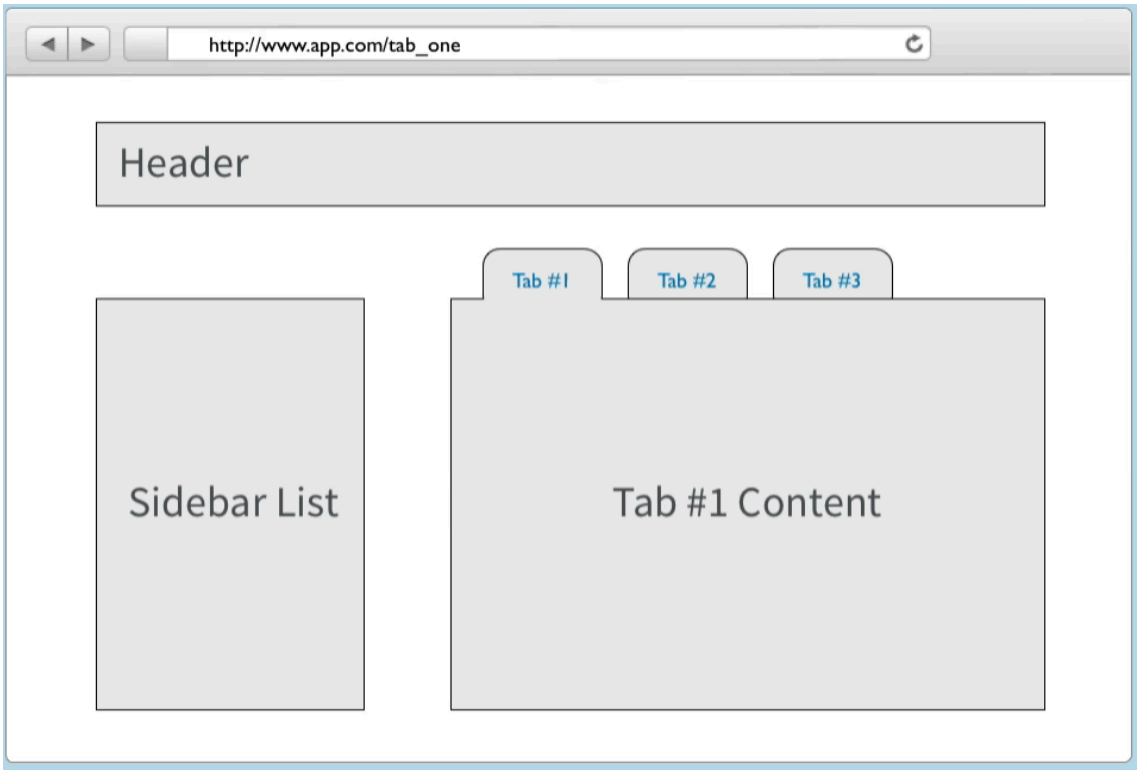


Figure 1. A tabbed window in a web application

One arguable turning point was the popularisation of AJAX that came with the release of Gmail in 2004. With Gmail Google showed that you could provide a rich user experience without page refreshes using just JavaScript. The release of the JavaScript framework jQuery in August 26, 2006 brought yet another renaissance to the language. Libraries such as jQuery are known to ["have done a great deal to help abstract inconsistencies across browsers and provide a high-level API for making AJAX requests and performing DOM manipulation"].[5] In the tabbed example from before, the popularization of jQuery meant that the tabs would work without a full-page refresh – in some cases by AJAX calls and in others by simply hiding and showing parts of the same page.

It is worth to note that in the early days of jQuery, web applications rarely updated the state of the active tab to the user (this is currently mostly done with hashes and push-

[5] Morrison, Jason; Pytel, Chad; Quaranto, Nick; Giménez, Harold; Clayton, Joshua; Berke-Williams, Gabe & Mazzola, Chad. 2012, *Backbone.js on Rails*. Page 6.

states) so the active tab was lost on a page refresh and the user wasn't able to navigate backward or forward between tabbed content with the standard buttons of the browser.

A good example is the shopping basket in an e-commerce application. Before the introduction of Ajax, adding items to the basked required a full page refresh, but with the use of jQuery and Ajax calls and call-backs developers were able to both persist the data and keep the view synchronised. The result of this was often a codebase that neither very well structured nor easy to maintain and debug.[6]

According to the official website of jQuery, ["jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript."][7]

With the help of jQuery can often get much more done with less code:

jQuery:

```
$('#container');
```

Vanilla JavaScript:

```
var container = document.getElementById('container');
```

A study conducted by BuiltWith in 2013 reports JQuery to be used by over 55% of the most visited websites in the world.[8]

[6] Osmani, Addy. 2012, *Developing Backbone.js Applications – Early Release*. Page 17.
[7] *jQuery*. Available at: http://jquery.com Accessed: 18.4.2013.
[8] *jQuery Usage Statistics*. Available at: http://trends.builtwith.com/javascript/JQuery Accessed: 18.4.2013.

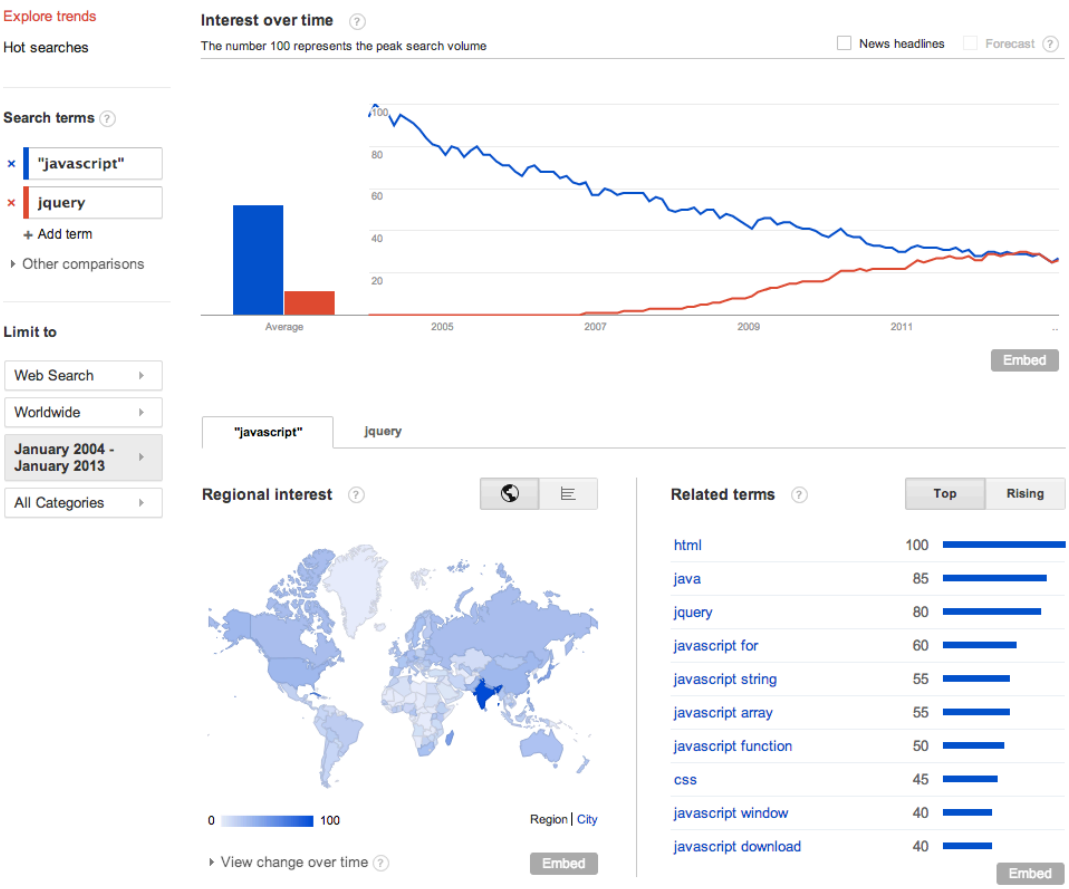


Figure 2. Google trends graph over the JavaScript and jQuery search terms

To further illustrate the importance of jQuery the graph above illustrates how the search term “jquery” has become nearly as popular as “javascript” in the last years.

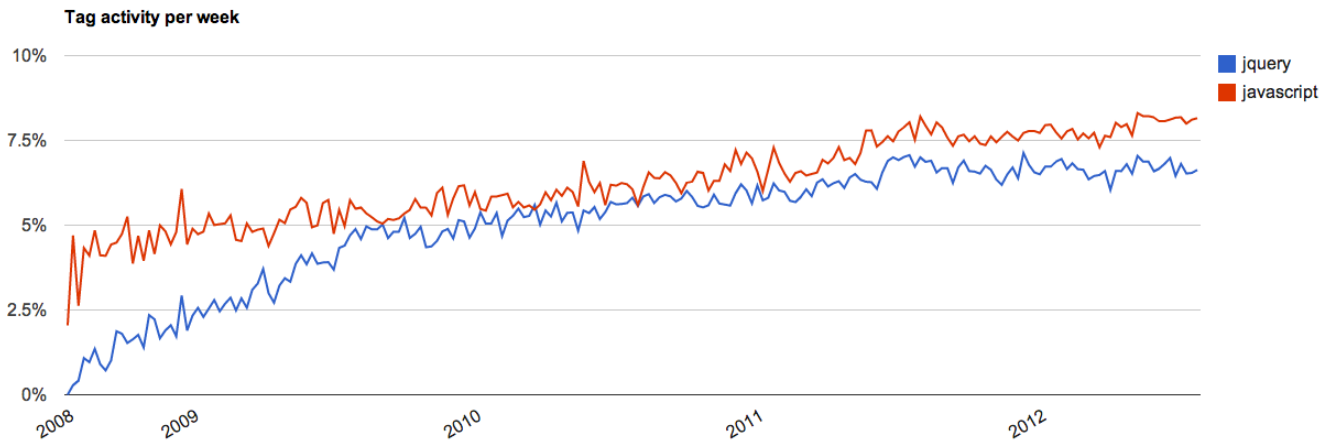


Figure 3. Tag activity for jQuery and JavaScript on StackOverflow.com

The popular developing Q&A site Stack Overflow also illustrates how the increased interest in jQuery and JavaScript has gone hand in hand.

JavaScript is also is also the by far most popular language on open-source git hosting service Github:

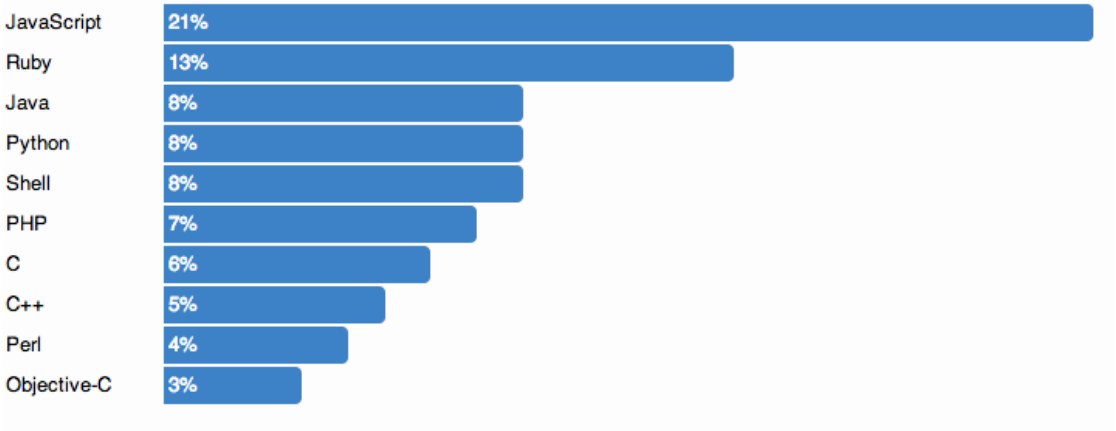


Figure 4. Github as of March 21, 2013

Modern NoSQL databases such as MongoDB also takes use of JavaScript for functions such as map/reduce, and MongoDB also has a shell that is based on JavaScript. Modern programming langue Clojure also has a compiler called ClojureScript that targets JavaScript. With the introduction of Node.js in 2009 that makes it easy to write scalable Internet applications and notably web servers in JavaScript, and with the backing of companies such as Microsoft and Linkedin, JavaScript is more popular than ever. As Backbone.js expert Brian Mann puts it, [“JavaScript is absolutely here to stay”] and [“JavaScript has become the cornerstone of modern development”].[9]

1.5 Client-side Applications

With the growing popularity of JavaScript and with the rise of client-side applications modern web applications have become increasingly complex in their use of JavaScript to the point where entire core logic is done entirely on the client-side and data fetching

[9] Mann, Brian. 2013, *Backbone Rails – Client Side Development*. Available at: http://vimeo.com/58787395 Accessed: 28.4.2013.

and storage is done in the background with Ajax. Pages such as Pandora.com or Rdio.com are popular examples of what is knows as a single-page application (SPA), as it fetches all data without reloading the page and processes all the logic on the client. The benefit in doing this is speed, as the client no longer has to query the server for huge amounts of data and expensive database interactions. Most data is loaded when the initial page loads, and the remaining data is transmitted asynchronously in a JSON (more about JSON later) format with less need for server processing and with fewer HTTP requests.

The process of a normal interaction between a client and web server consists of the following steps[10]:

1. The user types or clicks on a link in his or her client (in most cases a browser).
2. The browser makes a HTTP request to the server, containing a header and a body.
3. The server processes the request according to the query and potential parameters
4. The server manipulates or creates data in the database in the case of a dynamic page.
5. The server sends a HTTP response with a Header (e.g. 200 OK) and a Body (e.g. the changed model) containing the data in a format understandable by the client, such as HTML or JSON.
6. The browser receives the HTTP response.
7. The browser renders the response to the user.
8. The response can in many cases fire off more HTTP requests to the server.

If the user clicks another link, the cycle repeats.

In some cases the cycle from before can be entirely avoided because all the data already exists on the client. Instead of sending a HTTP request and loading a new page when the user clicks a link, the originally fetched data can just be represented in a different manner. Instead of the server delivering HTML pages containing images, style sheet-

[10] Mardanov, Azat. 2013, *Rapid Prototyping with JS.* Page 13.

and script references that initiates new HTTP requests, all interaction is done through JSON, and the only information being delivered is that of changed data on the model.

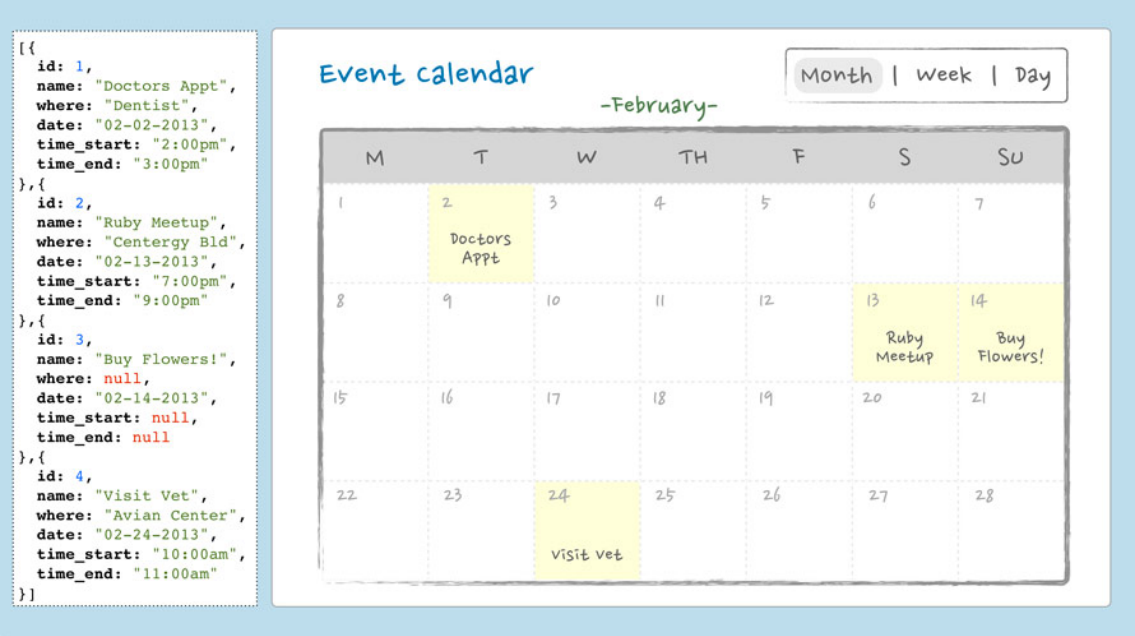


Figure 5. An event calendar displaying events by month

Take for instance a calendar application, where switching between a month and week view would traditionally require a re-rendering of the page or an AJAX request to the server that loads the new view for the region. As seen in fig2, a client-side application retains all the information needed to display the calendar already exists on the client (maybe even in HTLM5 local storage) so that all that has to change is the template that presents the information. If the data of a model changes, the view will automatically reflect this across templates.

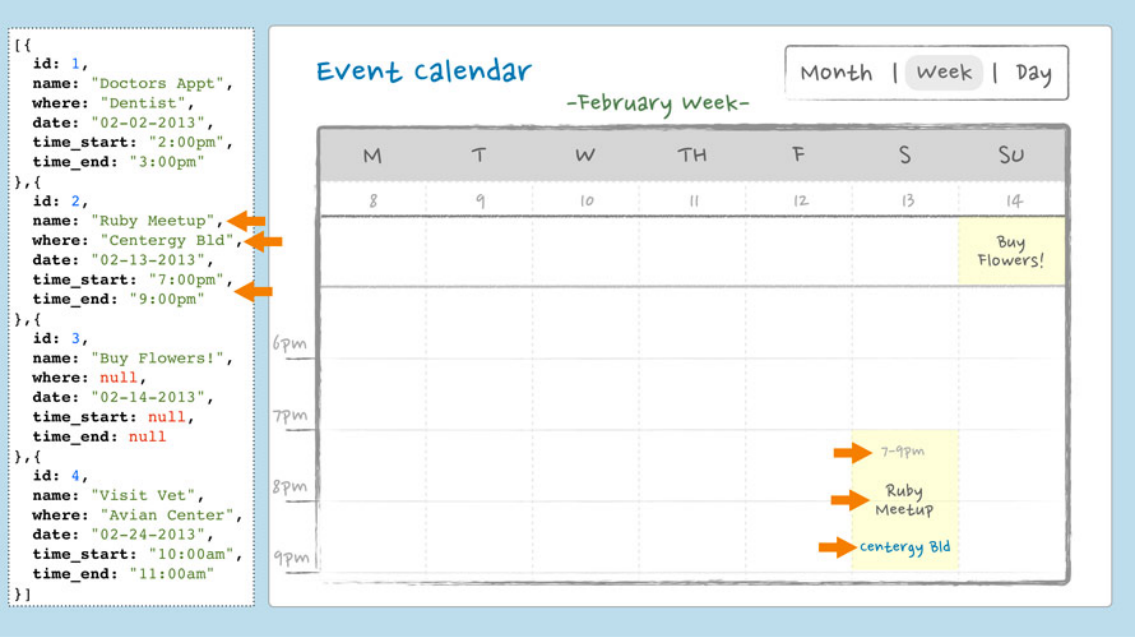


Figure 6. An event calendar displaying events by week

It is computationally expensive and a bad practice to create HTML with JavaScript through string concatenation, which is why JavaScript templating was introduced. Variables in templates are delimited using a specific syntax, such as {{ example }} in Handlebars.js, and defined by injecting JSON into the template.

Client-side applications also enables developers to easily implement offline versions of websites by using modern web technologies such as local storage – something that becomes increasingly important in our ever-increasing mobile information usage. Having a backend that serves JSON also opens up a lot of possibilities for easy integration with mobile applications. JSON requests require much less data to be transferred which can be seen as an extreme benefit, especially while operating over the often limited bandwidth of mobile networks.

JSON as described by json.org[11]:

> JavaScript Object Notation, or JSON, is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 19993. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

JSON has become increasingly popular with the popularization of public developer APIs (Application Programming Interfaces), and is the standard storage model for several NoSQL (not using sequel query language) databases, for example CouchDB and Riak. Contrary to SQL stored tabular data, JSON makes it easy to store data with a deeper nesting, such as with children of children.

```
{
    "id": 1,
    "name": "Foo",
    "price": 123,
    "tags": [ "Bar", "Eek" ],
    "stock": {
        "warehouse": 300,
        "retail": 20
    }
}
```

Figure 7. Example of JavaScript Object Notation (JSON) data

One of the major disadvantages in client-side and single-page applications is that they can be difficult for search engines to index, unless it degrades gracefully into a normal application. Traditionally in web development you wanted to degrade JavaScript functionality gracefully so that devices that had JavaScript disabled would not be at a disadvantage, but according to a 2010 study by Yahoo, the number of devices with disabled

[11] *JSON*. Available at: http://json.org Accessed: 20.4.2013.

JavaScript sits at around 1.3%. One might thereby question if graceful degradation persists as a requirement for accessibility.[12]

1.6 Model-View-Controller

Model-view-controller (MVC) is a software architecture pattern designed by Trygve Reenskaug while working on Smalltalk-80 in 1979, but only gained real popularity after being described in depth in Design Patterns: Elements of Reusable Object-Oriented Software in 1994. MVC divides parts of an application into three kinds of components in an attempt to make big projects more manageable through abstraction, and to create a unified structure between different projects. This helps a JavaScript application to stay manageable and scalable, and prevents having a codebase polluted by figuratively endless amounts of AJAX callbacks. Decoupling views and models also simplifies the writing of unit tests. [13]

MVC is not unique to JavaScript but can be found in many of the popular web development frameworks, such as Django and Ruby on Rails (technically Rails uses not traditional MVC but a pattern called Model2 which has a lot of similarities). The three components of traditional MVC and their interactions between each other are:

Controller

Processes and responds to events and manipulates the model and the view.

Model

Models contain the domain-specific representation of the information that the application is running. Models notify the Views when its state changes so that they can produce the updated model data.

[12] Zakas, Nicholas C. *How many users have JavaScript disabled?*, published 13.10.2013. Available at: http://developer.yahoo.com/blogs/ydn/posts/2010/10/how-many-users-have-javascript-disabled/ Accessed: 28.4.2013.
[13] Osmani, Addy. 2012, *Developing Backbone.js Applications – Early Release*. Page 13.

View

Views present the model in a form that is suitable for interaction by the user, such as the HTML in a web application. In MVC the logic of the application should be separated from the view as much as possible.

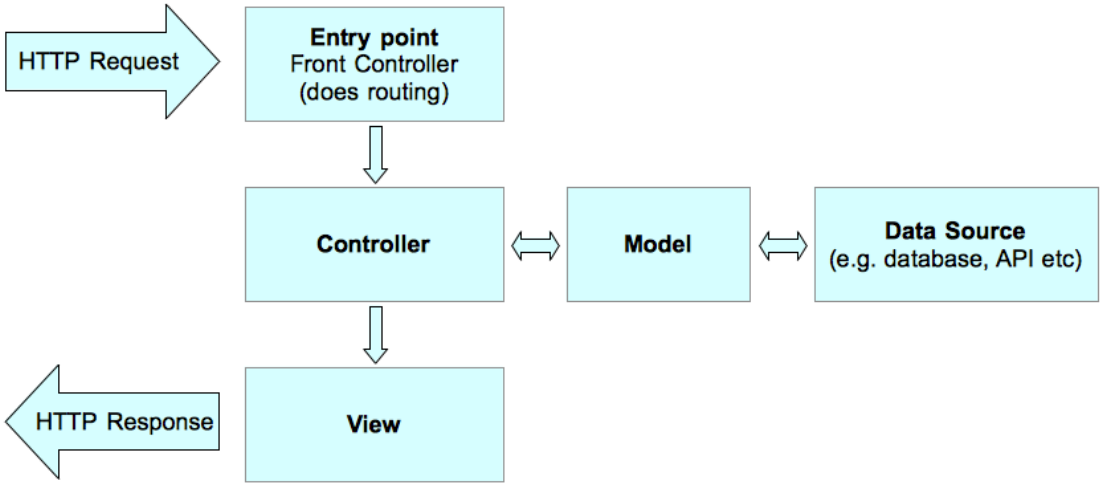


Figure 8. A typical HTTP request/response lifecycle for server-side MVC[14]

In JavaScript MVC frameworks the implementation is adjusted to address specific needs. While the following characteristics may not apply to every JavaScript MVC, they can be regarded as something of a standard:

Model

- Validates attributes
- Persists the model to a database or to the local storage of the browser
- Observed by views, to reflect model changes to the user
- Often grouped inside "collections" so that logic can be applied to several models at the same time

[14] Osmani, Addy. 2012, *Developing Backbone.js Applications – Early Release*. Page 16.

View

- Displays an interface to the user
- Renders the contents of a model (or collection)
- Updates when the model is changed
- Renders templates, by the use of a JavaScript template library such as Handlebars.js, Eco (Embedded CoffeeScript) or EJS (Embedded JavaScript)

EJS template:

```
<table>
  <tr>
    <th>Title</th>
    <th>Created</th>
  </tr>
  <% articles.each(function(model) { %>
    <tr>
      <td><%= model.escape('title') %></td>
      <td><%= model.escape('created_at') %></td>
    </tr>
  <% }); %>
</table>
```

Handlebars.js template:

```
<table>
  <tr>
    <th>Title</th>
    <th>Created</th>
  </tr>
  {{#each articles}}
      <tr>
        <td>{{title}}</td>
        <td>{{created_at}}</td>
      </tr>
  {{/each}}
</table>
```

Controller

- Handles changes in the view and updates the model
- Controllers are the components of MVC that varies most between JavaScript frameworks, to the point where they sometimes do not even technically exist, so it is extremely difficult to describe a common functionality between them

In a single-page JavaScript application, the views of the backend MVC do little to nothing but provide a container for population by JavaScript by the JavaScript MVC (fig.7). JavaScript models should store data and state and reflect the logic of the server-side, while views should update automatically to represent any data changes. Controllers should decide which models get used and which views get displayed. The server and the client usually interact with each other using JSON over a REST (Representational State Transfer) or SOAP (Simple Object Access Protocol) interface. The Backbone.js examples within this thesis will be communicating with a Ruby on Rails backend using REST.

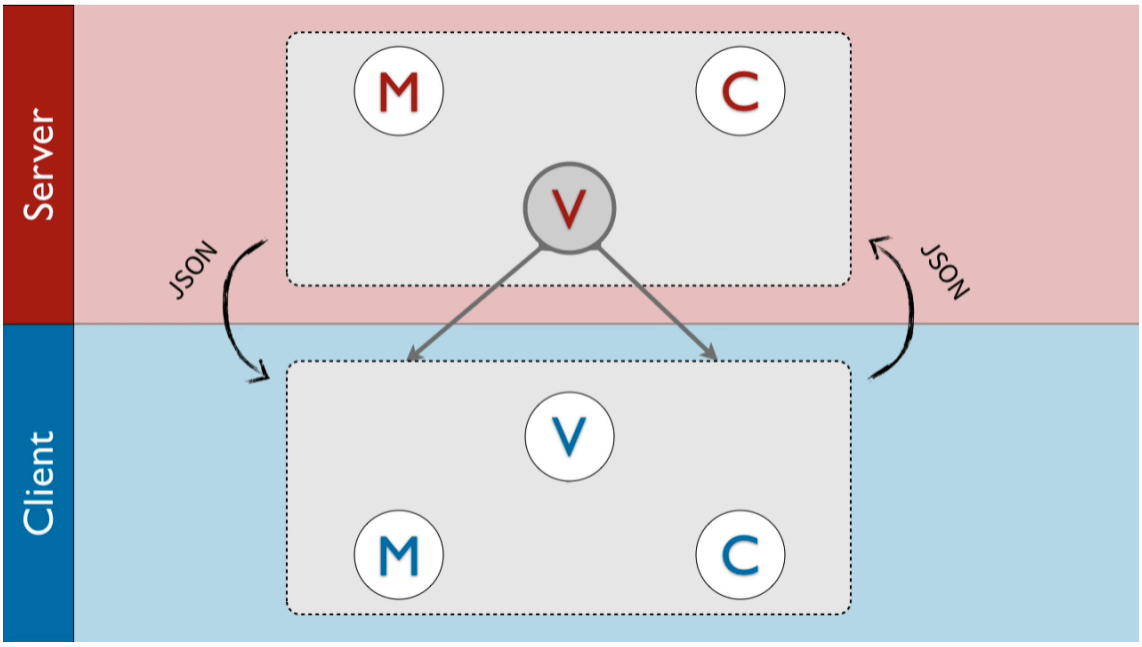


Figure 9. Model of a production setup of a single-page application[15]

[15] *Backbone Rails – Client Side Development*. Available at: http://vimeo.com/58787395 Accessed: 28.4.2013.

["RESTful (Representational State Transfer) API became popular due to the demand in distributed systems where each transaction needs to include enough information about the state of the client. In a sense this standard is stateless because no information about the clients' state is stored on the server, thus making it possible for each request to be served by a different system."] [16]

That REST replaced SOAP can be attributed to the simpler and more readable structure, and that REST utilizes standard HTTP methods such as GET, POST, DELETE and PUT. Fig. 7 shows an example of a CRUD (create-read-update-delete) implementation in a Ruby on Rails application.

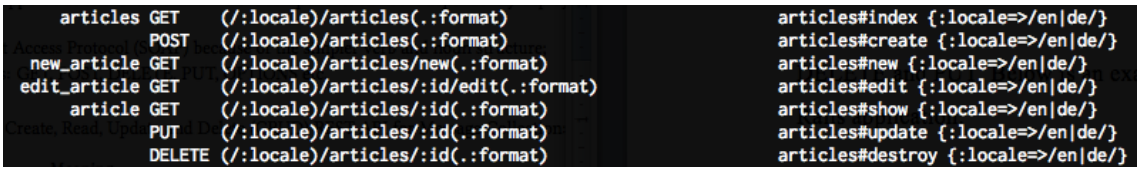


Figure 10.CRUD implementation in a Ruby on Rails application

Prior to the aforementioned JavaScript renaissance there existed many best practices, recommendations and frameworks for structuring server-side (code that executes on the server) code but little to nothing for organizing client-side (code that executes on the client, such as a browser) code.[17] While jQuery is extremely useful for modern web development, it still lacks a structure for organizing code. Larger client-side applications that lack decoupled and modular organizational structures often end up in what has become to be popularized under the term "spaghetti code":

> Spaghetti code is a pejorative term for source code that has a complex and tangled control structure, especially one using many GOTOs, exceptions, threads, or other "unstructured" branching constructs. It is named such because program flow tends to look like a bowl of spaghetti, i.e. twisted and tangled. Spaghetti code can be caused by several factors, including inexperienced programmers and a complex program which has been continuously modified over a long life cycle. Structured programming greatly decreased the incidence of spaghetti code.
>
> *– Wikipedia March 22, 2013*

[16] *Rapid Prototyping with JavaScript*. Page 6.
[17] *Backbone.js on Rails*. Page 6.

To address this, several JavaScript MVC frameworks sprung to life to bring some order and structure into bigger scale JavaScript applications. While many frameworks focus on the Model-View-Controller pattern there are those (for example KnockoutJS) that bring a different organization pattern such as Model-View-Presenter or Model-View-ViewModel. Some frameworks include the responsibility of the Controller in the View (e.g. Backbone.js) while others add their own opinionated components into the mix as they feel this is more effective. Backbone.js is for example technically speaking not even MVC, which it acknowledged by renaming its "Controllers" to "Routers" in version 0.5.0 (Backbone.js is as of writing in version 1.0). Because some JavaScript MVC frameworks differ in structure from what has traditionally been considered MVC, they are often referred to as using the MV* pattern, since they are all likely to have at least a Model and a View.[18] [19]

> "Modern JavaScript frameworks and libraries can bring structure and organization to your projects, establishing a maintainable foundation right from the start." – *Addy Osmani, author of Developing Backbone.js Applications*

As noteworthy additions to providing structure, frameworks such as Backbone.JS and AngularJS make it easier to build applications with asynchronous calls to the backend and to parse JSON data.

JavaScript MVCs can also aid in keeping the HTML from storing too much data in rel or data-* attributes, something that can be considered better because it lets the view remain a presentation layer without polluting it with application logic or data, and makes the codebase modular and more easily maintainable.

[18] Osmani, Addy. *Journey Through the JavaScript MVC Jungle,* published 27.6.2012. Available at: http://coding.smashingmagazine.com/2012/07/27/journey-through-the-javascript-mvc-jungle/ Accessed: 20.2.2013.
[19] *Backbone.js on rails.* Page 11.

This paper limits itself by focusing exclusively on Backbone.JS and AngularJS – the most popular JavaScript MV*'s being used as of writing this thesis according to Google Trends.

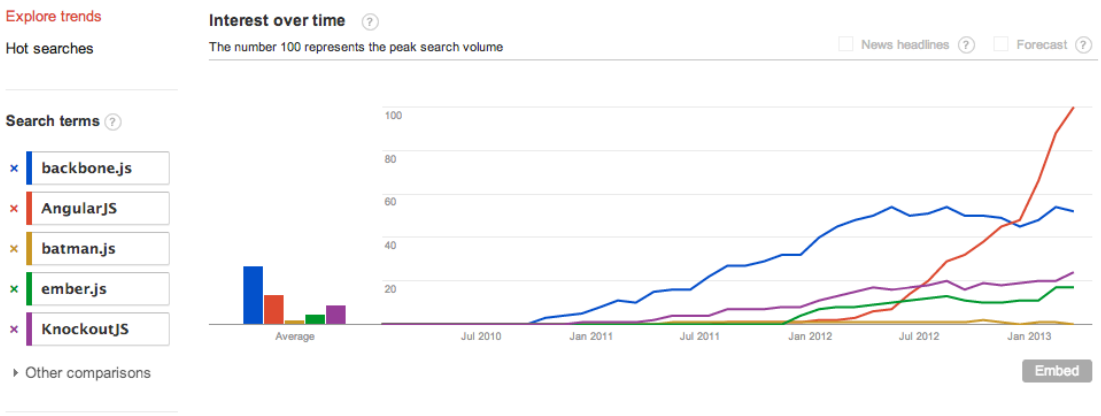


Figure 11. JavaScript MVC's trending on Google

It is also worth noting that both frameworks are in active use by some of the most visited websites of today. High-profile companies or products using Backbone.JS[20] **or** AngularJS[21] include but are not limited to:

- Del.icio.us
- Stripe
- Nokia
- LinkedIn
- Hulu
- Wordpress
- Foursquare
- Disqus
- Groupon
- Wallmart
- Basecamp
- Soundcloud
- Pandora
- Code School
- YouTube
- Netflix

[20] *Backbone.js | Examples*. Available at: http://backbonejs.org/#examples Accessed: 20.2.2013.
[21] *Built with AngularJS*. Available at: http://builtwith.angularjs.org/ Accessed: 20.2.2013.

1.7 Aim

The goal of this thesis is to investigate the difference between the highly popular JavaScript MVC frameworks Backbone.JS and AngularJS in terms of logic, performance, lines of code and boilerplate code. Through my research I hope to shine a light into the differences between the two frameworks, and to provide an extensive and reliable resource for others when it comes to selecting which JavaScript MVC framework to use.

1.8 Limitations

Out of the multitude of interesting modern JavaScript MVC frameworks available I've chosen to focus on Backbone.JS and AngularJS, based on their popularity. Despite the existence of popular extensions of the MVC frameworks mentioned that simplifies common tasks and/or reduces boilerplate code (for example Backbone Marionette), this project focuses solely on the core functionality of both frameworks in order to bring a just comparison. The projects will **not** take use of any Asynchronous Module Definition (AMD) services such as Require.js.

There is very little published about Backbone.js and especially AngularJS as of yet, because both frameworks are still very young. As a result, much of my source material will be limited to books being in various stages of progress, and online resources. Many of the eBooks being used still get updated on a weekly if not even daily basis, which makes writing about such a modern subject as this one very challenging.

1.9 Methodology

I will compare two identical To-Do list applications built using AngularJS and Backbone.JS in logic and performance. Chrome Development Tools will be used to measure DOM performance by CPU-profiling and heap snapshots. I will also be doing SLOC (Source Lines of Code) and disk-size comparisons for both the finished web application and the frameworks themselves. Both applications are fully available online on Github (a service for hosting source-code) as open-source applications. Because the project is open-source, it welcomes and accepts outside contributions that improve on the overall

code, to aid the applications in reaching their maximum potential and limit the effect of possible oversights by a single developer.

Automated tests will be developed by the use of PhantomJS, a headless testing framework. I will time the time it takes for the automated test to create, edit, and delete 1000 tasks in the To-Do-list over 1000 iterations in both frameworks within a Webkit environment. The results of these tests will be analysed and conclusions will be made.

1.10 Structure

I will begin with detailing the core architecture of Angular.js and Backbone.js and then highlight the differences between the respective frameworks. After that I will take a closer look at, and run performance comparisons on an example application through the use of functional tests. The results of the tests will be evaluated and a conclusion will be made about whether there is reason to prefer one framework to the other. I will also give an opinion on if the frameworks live up to their design goals.

2 ANGULARJS

2.1 Architecture

AngularJS is an opinionated JavaScript framework that is used to build and structure modern web applications, primarily single-page applications. AngularJS has its origins in a project called Google Feedback, developed by a team of Google employees in the beginnings of 2009. The original AngularJS released in 2009 heavily inherited from the MVC pattern, it gradually evolved into something closer to MVVM (Model-View-ViewModel) in the way in which it adapted the pattern for modern JavaScript use. The developers behind JavaScript have since humoristic ally started to call AngularJS a MVW (Model-View-Whatever) framework.[22] [23]

[22] Minar, Igor. *MVC vs MVVM vs MVP*, published 19.7.2012. Available at: https://plus.google.com/+AngularJS/posts/aZNVhj355G2 Accessed: 19.3.2013.

At the startup of the application, the browser loads the HTML and parses it into the DOM, and with it the angular.js script. Once the DOM has loaded, AngularJS looks for an ng-app directive that designates the application boundary. If a module is defined in the directive it is used to configure the $injector that creates the $compile service as well as the $rootScope. $compile then compiles the DOM and links it into the $rootScope after which remaining directives (if any) get executed.

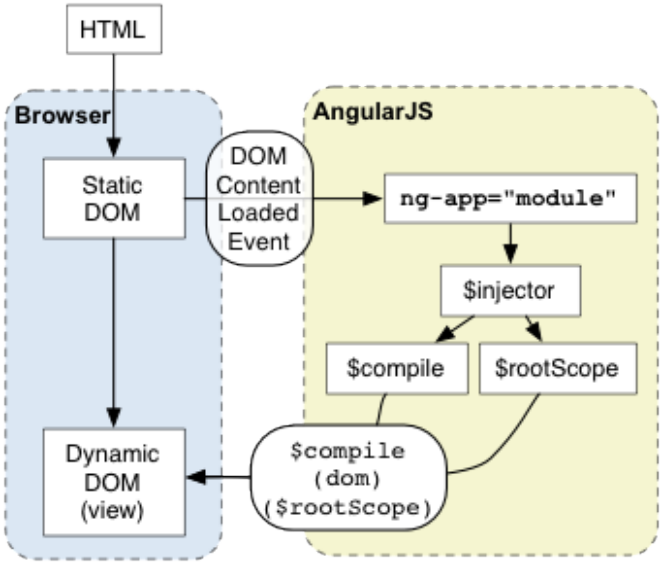


Figure 12. AngularJS startup cycle

Core features of AngularJS include:

- Two Way Data-binding
- HTML Templates (**not** a templating framework)
- Dependency Injection (high testability)
- Deep Linking
- Directives
- Model View Whatever
- $http service

[23] Green, Brad & Seshadri, Shyam. 2013, *AngularJS – Less Code, More Fun and Enhanced Productivity with Structured Web Apps AngularJS*. Page vii.

2.1.1 Two Way Data-Binding

Two way data-binding means that the view changes when the model changes, and the model changes when the view changes (usually through a form).

Two way data-binding is easier explained with an example:

```
<input type="text" ng-model="myName">
<h1>Hello {{myName}}!</h1>
```

If the user changes the value of model through an input in the view, it stores the value for the model in a variable. The h1 element then updates automatically to reflect the changes in the value of the model variable. The view also updates if the variable would be manually changed with JavaScript:

```
var myName ="Inigo Montoya"
```

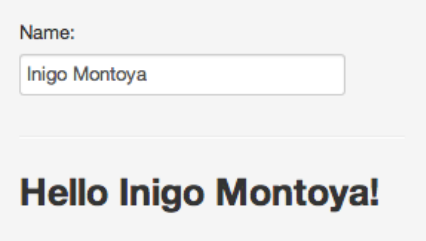


Figure 13. Shows example output produced by the previous code.

While AngularJS supports two-way data binding, Backbone.js relies on boilerplate code (repetitive sections of code) plugins or extensions to synchronize its models and views.[24]

[24] Gupta, Raj. *Backbone.js vs AngularJS : Demystifying the Myths*, published 27.12.2012. Available at: http://www.nebithi.com/2012/12/27/backbone-and-angular-demystifying-the-myths/ Accessed: 22.3.2013.

2.1.2 HTML Templates

Many other JavaScript MVCs use a templating system that is based on html (or for example HAML) with special markup that can be difficult to use for developers not familiar with the particular templating engine of choice. AngularJS doesn't rely on outside template engines such as Handlebars.js, but uses a templating system that is built on top of HTML through clever use of ng- attributes. The browser parses the HTML and looks for directives that, when executed, bind the view to the model.

Compare the following code to the Handlebars.js example in Figure 7:

AngularJS template:

```
<table>
  <tr>
    <th>Title</th>
    <th>Created</th>
  </tr>
  <tr ng-repeat="article in articles">
    <td>{{article.title}}</td>
    <td>{{article.created_at}}</td>
  </tr>
</table>
```

The template is displaying a table of articles with a column for title and creation date, and loops through a collection of articles to enter the row data. In contrast to the Handlebars.js template, the code of the AngularJS template contains only HTML code and does not require learning of new syntax to get into. There is no pre-processing involved. The downside compared to a Handlebars.js approach is that the HTML becomes polluted with ng-* attributes and exposes small parts of the application logic in the HTML code. This is still mostly just a matter of preference and one might certainly argue that the benefits outweigh the disadvantages.

2.1.3 Deep Linking

In a single-page application it is important to retain the state of the application in the url so that users are able to bookmark or share links to different states of the application, such as the index view and the show view. AngularJS uses the HTML5 history API together with a shebang (#!) fallback for older browsers. This functionality might not seem important at first, but it is extremely powerful in this modern age of social media and sharing.

2.1.4 Directives

Directives are something that is very unique to AngularJS and that enables you to extend the functionality of HTML. While AngularJS comes with a collection of predefined directives, it can be extended with custom functionality to the point where it allows the user to create his own DSL (Domain Specific Language). It allows you to create custom DOM elements, attributes and classes that you can attach functionality to, and thereby circumvent weird class hierarchy or boilerplate code. The following example from the official AngularJS documentation shows that you can for instance enable data-bindings for an html element if as certain attribute exists.[25]

HTML:

```
1. <div contentEditable="true" ng-model="content">Edit Me</div>
2. <pre>{{content}}</pre>
```

JavaScript:

```
1. angular.module('directive', []).directive('contenteditable', function()
   {
2.   return {
3.     require: 'ngModel',
4.     link: function(scope, elm, attrs, ctrl) {
5.       // view -> model
6.       elm.bind('blur', function() {
```

[25] *AngularJS: Conceptual Overview*. Available at: http://docs.angularjs.org/guide/concepts#directives Accessed: 20.4.2013.

```
7.          scope.$apply(function() {
8.            ctrl.$setViewValue(elm.html());
9.          });
10.       });
11.
12.       // model -> view
13.       ctrl.$render = function(value) {
14.         elm.html(value);
15.       };
16.
17.       // load init value from DOM
18.       ctrl.$setViewValue(elm.html());
19.     }
20.   };
21. });
```

2.1.5 Model-View-Whatever

Even though the developers of AngularJS have decided to call AngularJS a MVW framework, to many developers MVVM is a similar enough pattern to describe how AngularJS works. The following Wikipedia entry summarizes MVVM pretty well:

> ["MVVM facilitates a clear separation of the development of the graphical user interface (either as markup language or GUI code) from the development of the business logic or back end logic known as the model (also known as the data model to distinguish it from the view model). The view model of MVVM is a value converter meaning that the view model is responsible for exposing the data objects from the model in such a way that those objects are easily managed and consumed. In this respect, the view model is more model than view, and handles most if not all of the view's display logic (though the demarcation between what functions are handled by which layer is a subject of ongoing discussion and exploration). The view model may also implement a mediator pattern organizing access to the backend logic around the set of use cases supported by the view.
>
> MVVM was designed to make use of data binding functions in WPF to better facilitate the separation of view layer development from the rest of the pattern by removing virtually all GUI code ("code-behind") from the view layer. Instead of requiring user interface (UXi) developers to write GUI code, they can use the framework markup language (e.g., XAML) and create bindings to the view model, which is written and maintained by application developers.] […]
>
> - Model View ViewModel Wikipedia 29.3.2013[26]

As illustrated in fig. 12, as the application loads the **View** (HTML with AngularJS syntax) it binds to the **ViewModel** ($scope) and when the ViewModel changes it automatically updates the **View**. This makes up AngularJS' two-way data binding. This architec-

[26] *Model-View-ViewModel*. Available at: http://en.wikipedia.org/wiki/Model_View_ViewModel Accessed: 29.3.2013.

tural pattern separates the logic from the view and makes AngularJS applications very testable.

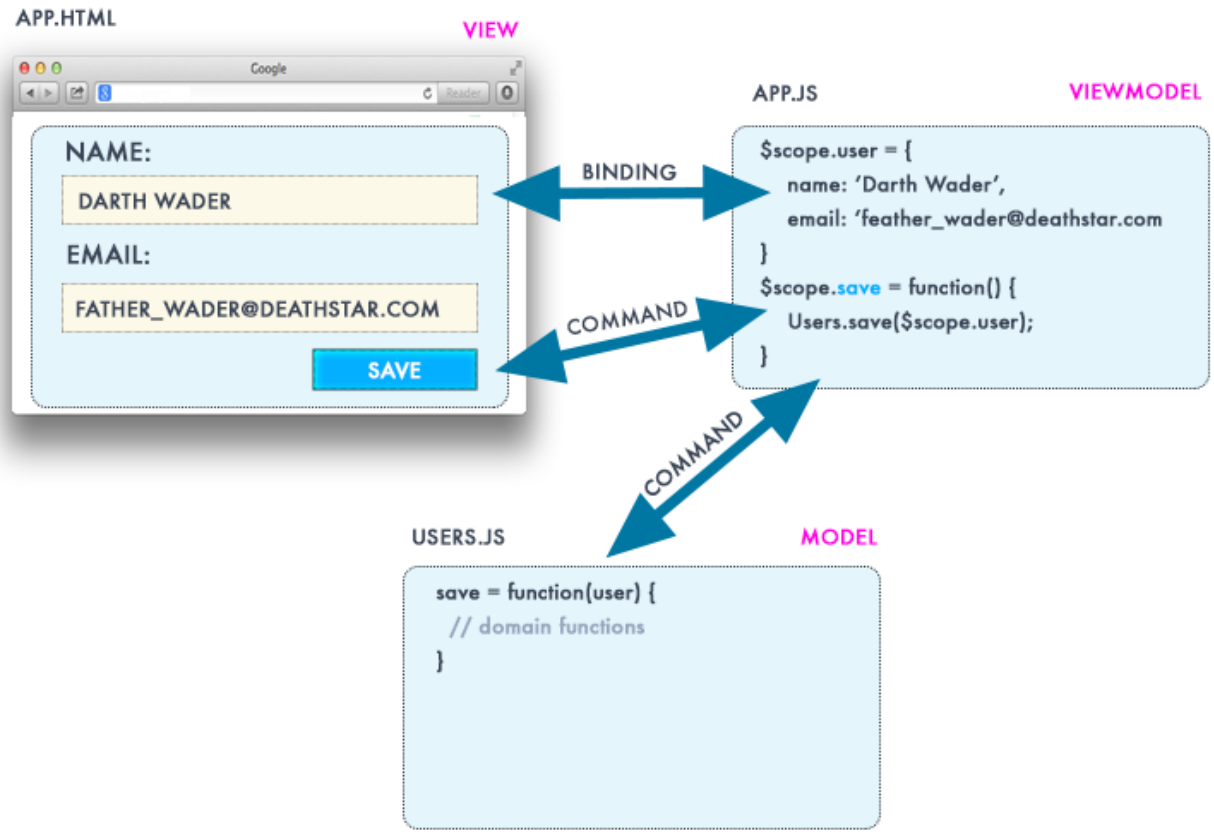


Figure 14. Two-way Data Binding in AngularJS

The two-way data binding of AngularJS forces views to always be a projection of the model. Worth noting is that the model has no restrictions when it comes to the type of to model, but can be a primite, an objet hash (such as in figure 14) or a function.

2.1.6 Dependency Injection (DI)

Dependency Injection is greatly encouraged by AngularJS and is another of its components that greatly improves on its testability. While perhaps sounding complex, depend-

ency injection is simply when you have a function that takes a parameter, such as the following:

```
addItem = function(item){
  $scope.add(item);
}
```

The example above is easy to test because the item that is introduced to the function can be either a real object or a mock object for testing purposes. Separation into smaller testable components can really aid in tracking down bugs.

2.1.7 $http service

In comparison to for example Backbone.js, AngularJS is highly flexible in how it communicates with different backends. Instead of relying solely on a REST interface it communicates freely through the browser's XMLHttpRequest object or via JSONP.[27]

```
$http({method: 'GET', url: '/someUrl'}).
  success(function(data, status, headers, config) {
    // this callback will be called asynchronously
    // when the response is available
  }).
  error(function(data, status, headers, config) {
    // called asynchronously if an error occurs
    // or server returns response with an error status.
  });
```

Even though $http service adds default http headers to requests, they are all easily configurable through the $httpProvider.defaults.headers object.

[27] *AngularJS: $http*. Available at: http://docs.angularjs.org/api/ng.$http Accessed: 6.4.2013.

2.2 Design Goals

AngularJS lists the following as their design goals, or as put on the official website, "The Zen of Angular":

- Decouple DOM manipulation from application logic
- Regard application testing as equal in importance to application writing
- Decouple the client side of the application from the server side
- Guide the developer into building a structured, testable application
- Make common tasks trivial and difficult tasks possible

2.3 Other benefits

AngularJS is highly embeddable and fairly lightweight, which means that it is very flexible when it comes to adding into an existing jQuery application. One can use AngularJS for just a single feature of the site, such as an event calendar, without having to worry about the impact on the rest of the application, nor design the rest of the application around AngularJS.[28]

AngularJS is also being maintained by Google.

3 BACKBONE.JS

3.1 Architecture

Backbone.js is a JavaScript MVC framework that leaves many decisions open for the developer. Backbone.js was released in October, 2010 by Jeremy Ashkenas, also famous for creating CoffeeScript and the Underscore.js JavaScript library – the later being the only hard dependency of Backbone.js. While most configurations of Backbone.js

[28] Ford, Brian & Ruebbelke, Lukas. 2012, *AngularJS in Action – Early Access Edition*. Retrieved: 28.3.2013. Page 13.

use jQuery (93kb), it can also be swapped out for the newer and more lightweight Zepto.js (27kb) or the highly modular Ender.js.

Backbone.js is derived from the MVC but (as many other JavaScript MVCs) has developed its own interpretation of the pattern. In version 0.5.0 Backbone.js renamed the controllers into routers, which arguably acknowledged its move from the traditional MVC pattern. The official documentation of Backbone.js declares that ["in Backbone the View class can also be thought of as a kind of controller, dispatching events that originate from the UI, with the HTML template serving as the true view"][29].

At the startup of the application, the browser loads the HTML and parses the JavaScript file into the DOM. Once the DOM has loaded, Backbone.js is commonly initialized through an App.initialize(); call. The initialize creates a router that in turn creates a view that is inserted into the DOM, mostly through the use of a template. Once the initialize is completed it runs the Backbone.history.start() method to activate route handlers.[30]

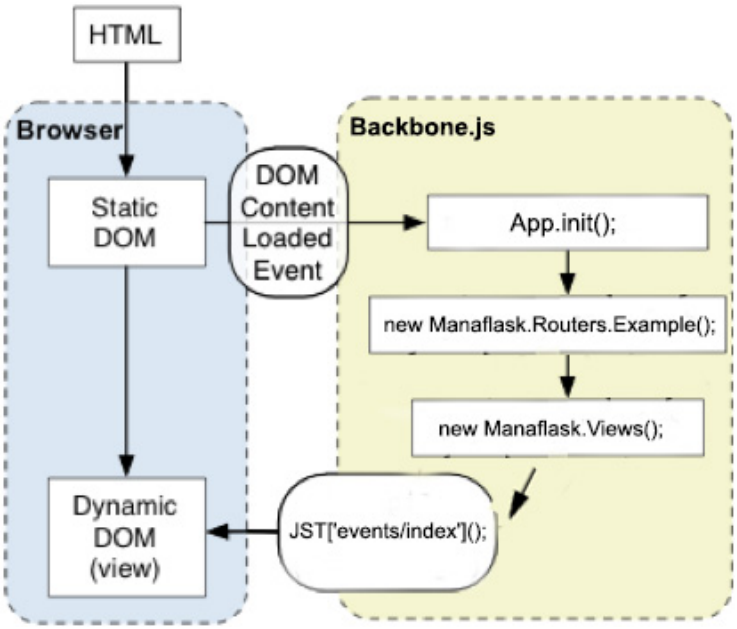


Figure 15. A typical Backbone.js startup sequence

[29] *Backbone.js | How does Backbone relate to "traditional MVC?*. Available at: http://backbonejs.org/#FAQ-mvc Accessed: 28.3.2013.
[30] Bailey, Derick. *3 Stages of a Backbone Application's* Startup, published 6.2.2012. Available at: http://lostechies.com/derickbailey/2012/02/06/3-stages-of-a-backbone-applications-startup/ Accessed: 28.3.2013.

Core features of Backbone.js include:

- Underscore.js
- MV*
- Agnostic Templating
- Clean HTML
- Synchronous events
- REST
- Backbone.sync
- Extensions
- Deep linking

3.1.1 Underscore.js

Underscore.js is a library created by the author of Backbone.js that provides additional functionality to JavaScript without extending the built-in JavaScript objects, and is heavily used by Backbone.js.

Other languages such as Python or Ruby come with constructs such as map, select and invoke that are not currently natively supported by JavaScript. Underscore.js addresses this problem, and adds a lot of new functionality to the language. In the presence of a modern browser, Underscore.js will use the browser-native implementation of the functionality, where it exists.[31]

Underscore.js also supports micro templating through the highly customizable _.template function and is the default templating framework being shipped with Backbone.js.[32]

[31] Gupta, Siddhartha, *Getting Cozy With Underscore.js*, published 31.3.2013. Available at: http://net.tutsplus.com/tutorials/javascript-ajax/getting-cozy-with-underscore-js/ Accessed: 5.4.2013.
[32] *Underscore*.js. Available at: http://underscorejs.org/ Accessed: 5.4.2013.

3.1.2 Agnostic Templating

Despite offering templating through Underscore.js, Backbone.js suggests[33] implementing a JavaScript templating library, and lists Mustache.js, Haml-js and Eco as "fine alternatives". Developers are still by no means limited to these three but can choose from many of the other templating libraries available, such as EJS or Handlebars.js. It is also possible to use several different templating libraries within the same application.

3.1.3 Model-View-*

Technically speaking, Backbone.js is not MVC. Backbone.js doesn't really have controllers. The view classes are responsible for not only presentation but also establishing and responding to UI event bindings[34], and Routers are used to help manage application state.[35]

Model-View-Presenter (MVP) is an architectural pattern that is derived from MVC and that is not very much unlike what Backbone.js uses. Because Backbone.js leaves very much up to the developer, it is hard to define as an MVP but some developers feel that it is closer in its structure to MVP than to MVC. In their opinion, the Presenter in MVP describes the Backbone.View better than a Controller does in MVC, and that Views in turn best represent templates. The truth is that both the V in MVC and P in MVP can be accomplished by Backbone.View, and it is up to the developer to decide how he or she wants to integrate Backbone.View in his or her application.[36]

According to Derick Bailey, author of the popular meta framework Marionette, Backbone.js developers should ["toss MVC/MVP/MVVM out the window and just call it part of the MV* family. Or better yet, let's just call it "The Backbone Way"

---

[33] *Backbone.js | Backbone.Events*. Available at: http://backbonejs.org/#Events Accessed: 6.4.2013.
[34] *Backbone.js on Rails*. Page 11.
[35] *Developing Backbone.js Applications*. Page 26.
[36] Bailey, Derick. *Backbone.js Is Not An MVC* Framework, published 23.12.2011. Available at: http://lostechies.com/derickbailey/2011/12/23/backbone-js-is-not-an-mvc-framework/ Accessed: 6.4.2013.

and forget about trying to fit some cookie cutter mold around a fluid and flexible library.“][37]

The structure of Backbone.js can be divided into the following components:

Backbone.Model

Models contain the business logic of the data in application, and provide functionality for managing changes and persistence through Backbone.sync. Backbone.js supports model validation via the Model.validate method. Models should be separated from the presentation layer in Backbone, as in any MVC application.

Backbone.Collection

In Backbone.js, a collection is an array of models. The collection supports for example sorting, filtering, aggregation and Underscore.js methods.

Backbone.Events

Synchronous events are one of the core concepts of Backbone.js. Backbone.Events is a module that can be attached to any object to enable the object to respond to and trigger custom events across the application. The event system is based on the Publisher-Subscriber Pattern, where Subscribers listen for defined events. When a model is changes it “publishes” the change to the rest of the application, and Subscribers react when Publishers trigger these events and update the view accordingly.

[37] *Developing Backbone.js Applications*. Page 297.

```
class Application.Views.ArticlesNew extends Backbone.View
  events:
    "submit #new-article": "save"

  constructor: (options) ->
    super(options)
    @model = new @collection.model()

  save: (e) ->
    e.preventDefault()
    e.stopPropagation()

    @model.unset("errors")

    @collection.create(@model.toJSON(),
      success: (article) =>
        @model = article
        window.location.hash = "/index"

      error: (article, jqXHR) =>
        alert("there was an error")
        @model.set({errors: jQuery.parseJSON(jqXHR.responseText)})
    )
```

Figure 16. The save function is triggered after the "save" event is triggered when the user submits the #new-article form

Backbone supports data bindings through both manual events and a seperate key-value observing library.

Backbone.Router

The router maps client-side URL fragments (with shebang) to functions that in turn render views, and thereby enables deep linking. Starting with Backbone 0.5, Backbone.js also includes support for HTML5 pushState, which enables the use of real full URLs instead of "shebanged" fragments. PushStates still degrade gracefully to shebangs (#!) for browsers that lack pushState support.

Backbone.View

The view is a logical reusable piece of the UI that is usually connected to a model or collection, and gives a visual presentation of the model data and its current state. Views can also bind to events that may cause the view to be re-rendered (you can even bind a

view's render() function to a model's change() event). Views in Backbone usually render HTML through the use of templates.

Client-side Templates

Templates render HTML for the view to be appended to the DOM and often depend on data from a model or a collection, as displayed in the following example:

```
#! CoffeeScript
render: ->
  @.$el.html(HandlebarsTemplates['articles/index'](collection: @collection))
```

Figure 17. The example renders HTML for a collection inside a view by the use of the Articles/Index Handlebars template

Note that the template exists on the client, and is just being injected with a JSON string. Nothing additional has to be fetched from the server after the initial data has been loaded, even if the template was to change.

3.1.4 Clean HTML

Backbone.js is very unobtrusive when it comes to its impact on the DOM. While other frameworks (or the lack of frameworks) might rely on invented HTML-tags, data- attributes or custom attributes such as ng-, Backbone.js strives to keep the HTML clean. There is no embedded JavaScript, template logic or bindings being defined in the HTML. In comparison to AngularJS this means that Backbone.js requires the added component of templates to render the HTML.

3.1.5 Backbone.sync

Sync uses the ajax functionality of jQuery or Zepto to make a RESTful JSON requests over the standard CRUD (CREATE, READ, UPDATE or DELETE) methods. Backbone.js calls sync whenever it attempts to read or persist a model to a server. The persistence strategy can be overridden to use WebSockets, XML transport or Local Storage.

The CRUD methods are by default mapped to REST like in figure 18 (it is modeled after the Ruby on Rails web framework) but can, like most things in Backbone.js, be completely customized or overridden.

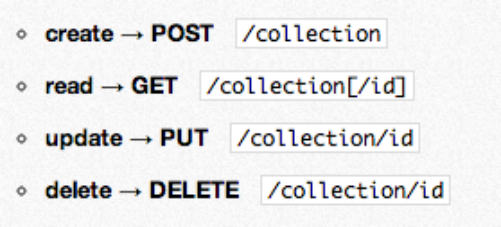


Figure 18. CRUD mapping in Backbone.js

3.1.6 Extensions

A core concept of Backbone.js is extending existing functionality, either yourself or by use of outside community-developed plugins, [38] and inherits the extend command from Underscore.js. Backbone.js does not for instance want to support two-way data binding by default, but enables that option through the use of extensions such as Rivet.js[39] or Backbone.stickit[40]. Notable extensions include:

- **Backbone.Validations** – Declarative per-attribute validations
- **Backbone-forms** – Provides form markup construction and serialization
- **Backbone.localStorage** – LocalStorage adapter that overrides Backbone.Sync
- **Backbone-relational** – Support for one-to-one, one-to-many and many-to-one relations for Backbone models
- **Backbone-pageable** – Replaces traditional Collections and extends it to support pagination
- **Backbone.DataBinding** – Adds bidirectional binding between views and models
- **Backbone.BabySitter** – Manage child views in a Backbone.View

[38] *Backbone.js | Extending Backbone*. Available at: http://backbonejs.org/#FAQ-extending Accessed: 7.4.2013.
[39] http://rivetsjs.com/
[40] http://nytimes.github.io/backbone.stickit/

Entire frameworks have also been made around extending Backbone and/or providing opinionated defaults that prevent repetition and boilerplate. Good examples of such frameworks are Chaplin[41] and Marionette[42].

Due to the vast amount of extensions available, many of which have extremely similar names (e.g Backbone.validation, Backbone.validations and Backbone.validator), it can easily become confusing for a new developer and the learning curve can be quite steep.

3.2 Design Goals

There is more than one way to accomplish things in Backbone.js, and Backbone is intended to be fairly agnostic when it comes to how you as a developer write your code.

Some of the design goals listed on the official website are:

- How one binds models to views, or defines events is largely up to one self[43]
- Backbone.js does not force you to use a single template engine
- Backbone.js has no logic inside the HTML
- Backbone.js is easy to scale
- Backbone.js is embeddable into existing applications
- Backbone.js does **not** support two way data-binding as a default
- Backbone.js is not very opinionated
- Backbone.js is easily extendable

3.3 Other Benefits

Backbone.js is a quite mature framework for its age, and has a large community following and many openly available extensions that add to its functionality. Backbone has also been used to create some extremely popular applications, such as Disqus, Four-

[41] http://chaplinjs.org/
[42] http://marionettejs.com/
[43] *Backbone.js | There's More Than One Way To Do it*. Available at: http://backbonejs.org/#FAQ-tim-toady Accessed: 8.4.2013.

square and SoundCloud. Because the framework is as mature as it is, there is a lot of solutions ready online to answer to the need of developers new to the framework. As of now, there are also many more published books available for Backbone.js than for AngularJS.

4 DIFFERENCES

Having taken a closer look at both AngularJS and Backbone.js, there are some notable differences that should be highlighted.

4.1 Dependencies & Size

While Backbone.js depends on both Underscore.js and jQuery or Zepto, AngularJS has no direct outside dependency.

Backbone.js (18KB)	Angular.js (77KB)
jQuery (93KB)	-
Underscore (13.6KB)	-
Total: **124.6KB**	Total: **77KB**

Table 1. A size comparison of backbone.js and agular.js

It is worth to note that most developers will still find themselves including jQuery or Zepto in their AngularJS environment, but unlike for Backbone it is not a requirement. As seen in fig. 15, this results in that AngularJS is vastly smaller in size. Backbone.js is also by its nature more dependent on outside extensions to extend its functionality whereas AngularJS comes bundled with opinionated defaults. One might also want to include another templating framework for Backbone.js, for example Handlebars.js that further adds to the size of the application.

4.2 Structure

As already explained in more detail in previous chapters, Backbone.js is closer to the MVC or MVP pattern while AngularJS is not structured so far from the MVVM pattern. Neither framework can strictly fit under either definition, and are best described under the term MV*.

Backbone includes the bare base for structuring your application and leaves the rest (e.g. memory management, layout management, structure, global event bus…) up to the developer to code or patch by the use of extensions. In theory you could even customize Backbone.js by the use of extensions (such as Knockback.js) to be very similar to AngularJS.[44]

It would be easy to list all the things that AngularJS supports but that Backbone.js doesn't support, but it is simply due to the extendable nature of Backbone that can be seen both as a blessing and a curse, and as a result only the most notable differences will be mentioned.

4.2.1 Two Way Data-Binding

Backbone.js does not support two way data-binding by default, and is surprisingly hostile towards the idea in the official documentation, yet mentions that it is possible through the use of extensions ("go for it" are links to extensions):

> "Two way data-binding" is avoided. While it certainly makes for a nifty demo, and works for the most basic CRUD, it doesn't tend to be terribly useful in your real-world app. Sometimes you want to update on every keypress, sometimes on blur, sometimes when the panel is closed, and sometimes when the "save" button is clicked. In almost all cases, simply serializing the form to JSON is faster and easier. All that aside, if your heart is set, go for it. – Backbonejs.org

Compare that to the documentation of AngularJS that states that its two-way data binding is […]["greatly simplifying the programming model for the developer."] [45]

---

[44] Gupta, Raj. *Backbone.js vs AngularJS: Demystifying the Myths*, published 27.12.2013. Available at: http://www.nebithi.com/2012/12/27/backbone-and-angular-demystifying-the-myths/ Accessed: 9.4.2013.

[45] *AngularJS: Data Binding in Angular*. Available at: http://docs.angularjs.org/guide/dev_guide.templates.databinding Accessed: 9.4.2013.

AngularJS takes use of the $digest() method to accomplish its two-way data binding. AngularJS remembers the value of a model and compares it to a previous value, and when it differs it fires the change event. This is knows as dirty checking. In contrast to the dirty checking of AngularJS, Backbone.js takes use of change listeners that according to some developers contain more programming gotchas[46].

Dirty checking is by definition inefficient but Backbone.js has its own issues especially when looping through a big array and adding models to a collection. In a typical Backbone.js application, each time an object is added it fires change events, which is rendering the UI. This is very bad for performance, and the optimal solution would be to only update the UI once at the end of the loop.

Miško Hevery, one of the original authors of AngularJS, argues on Stack Overflow that the differences in performance in a normal application are still to small to be noticeable by a human, and that the differences are for this reason negligible.[47]

[46] *Gotcha (programming)*. Availabe at:
http://en.wikipedia.org/wiki/Gotcha_(programming) Accessed: 9.4.2013.
[47] *Databinding in AngularJS*. Available at:
http://stackoverflow.com/questions/9682092/databinding-in-angularjs#answer-9693933
Accessed: 9.4.2013.

4.2.2 Templating

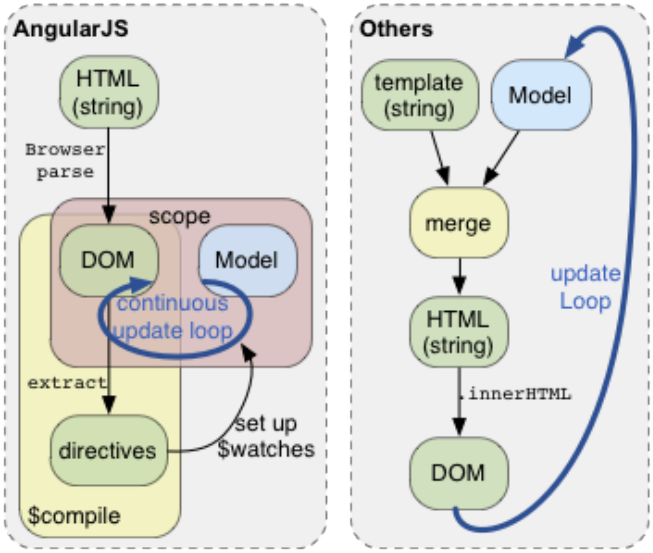


Figure 19. The update loop of AngularJS and "Other" JavaScript MVC[48]

AngularJS takes a static DOM containing HTML, CSS and AngularJS elements, expressions and directives (usually by the use of ng- attributes) and parses it to add behaviour that transforms the static template DOM into a dynamic view DOM with a continious update loop as seen in fig. 19.[49]

Backbone.js on the other hand takes the two components of a model (or a collection) and a template and, by the help of a templating engine, merges the two into a HTML string that is then inserted into the DOM. Backbone.js could be seen as very flexible as it is completely agnostic about which templates it uses (so seasoned developers can use templates they are already familiar with), while AngularJS has its own set standard that a developer needs to learn and follow.

Backbone.js is in favour of keeping the HTML free from custom attributes and elements:

[48] *AngularJS: Conceptual Overview*. Available at: http://docs.angularjs.org/guide/concepts Accessed: 10.4.2013.
[49] *AngularJS: Understanding Angular Templates*. Available at: http://docs.angularjs.org/guide/dev_guide.templates Accessed: 10.4.2013.

> "[Backbone] doesn't depend on stuffing application logic into your HTML. There's no embedded JavaScript, template logic, or binding hookup code in data- or ng- attributes, and no need to invent your own HTML tags." – Backbonejs.org[50]

AngularJS doesn't want to rely on a special templating markup:

> "Most templating systems begin as an HTML string with special templating markup. Often the template markup breaks the HTML syntax which means that the template can not be edited by an HTML editor. The template string is then parsed by the template engine, and merged with the data. The result of the merge is an HTML string. The HTML string is then written to the browser using the.innerHTML, which causes the browser to render the HTML. When the model changes the whole process needs to be repeated. The granularity of the template is the granularity of the DOM updates. The key here is that the templating system manipulates strings." – Angularjs.org[51]

Either solution has its own advantages and disadvantages, and in the end it comes up to personal preference and style.

4.3 Popularity & Maturity

As of 8.4.2013, a Google search for Backbone.js returns 5 690 000 results, while a search for AngularJS returns 1 130 000, less than one fifth of the amount of results. While this is not by any means a clear indicator of popularity, it speaks something of the maturity of Backbone.js and the amount of information available. As of now, not a single book has been published about AngularJS (*AngularJS* by O'Reilly Media is estimated to be published by the end of April, 2013) while many exist for Backbone.js, which can make learning Backbone significantly easier in comparison to Angular.

As of April 9, 2013, AngularJS has 4183[52] questions answered on Stack Overflow while Backbone.js has 8195[53]. One can thereby assume to easier find solutions to common developer-problems if one were to use Backbone.js.

---

[50] *Backbone | Why use Backbone, not [other framework]?*. Available at: http://backbonejs.org/#FAQ-why-backbone Accessed: 12.4.2013.

[51] *AngularJS: Conceptual Overview*. Available at: http://docs.angularjs.org/guide/concepts Accessed: 12.4.2013.

[52] http://stackoverflow.com/questions/tagged/angularjs Accessed: 9.4.2013.

[53] http://stackoverflow.com/questions/tagged/backbone.js Accessed: 9.4.2013.

4.4 REST

Backbone.js is highly targeted towards a certain kind of RESTful backends, such as Ruby on Rails, and requires overriding of the Backbone.sync function (or extensions) to configure it for something that varies a little bit from an extremely traditional REST backend. AngularJS comes with $http and is designed to be used for nearly any configuration, not only REST. For RESTful applications, AngularJS offers ngResource, an additional file that can be included to simplify the communication, but ngResource is by no means a requirement.

4.5 Extensions

While not a dependency, it is normal for Backbone.js applications to rely on outside plugins. Not all plugins come with tests and the quality of the code can vary a lot. If the plugin contains a bug it can be difficult for a developer that isn't familiar with the code to track down and address the issue. It can also be difficult to become aware of all the plugins that exist since there is no centralized hub as of now besides the Wiki page[54] of the project's Github repository, and the quality of documentation varies a lot between different extensions.

AngularJS also has many modules available, most of which are listed at ngmodules.org, but includes much functionality by default that Backbone.js developers have to look to extensions for. This is part of what makes AngularJS a little heavier than Backbone.js, but it also means that, in the case of Angular, the documentation is often better and available in a searchable centralized place.

4.6 Other differences

The learning curve for AngularJS can currently be considered higher, since you need a very good understanding of the DOM, directives and filters; many of which are com-

[54] *Extensions, Plugins, Resources*. Available at: https://github.com/documentcloud/backbone/wiki/Extensions,-Plugins,-Resources Accessed: 9.4.2013.

pletely new concepts to frontend developers. Additionally, the current lack of published books on AngularJS can make it less accessible than Backbone.js for novice developers.

5 PERFORMANCE

5.1 TodoMVC

TodoMVC is a tool designed to help developers select a JavaScript MV* framework. The premise is that ["developers these days are spoiled with choice when it comes to selecting an MV* framework for structuring and organizing their JavaScript web apps"][55] and TodoMVC steps in to solve this problem of choice.

The TodoMVC is an open source project that has implemented the same To-Do (task management) application using most of the popular JavaScript MV* frameworks of today. Because it is open source, the code has been scrutinized and improved by thousands of developers worldwide. As of April 9, 2013, TodoMVC has over 6000 stars and 1380 forks on its public Github repository.

[55] *TodoMVC*. Available at: http://todomvc.com/ Accessed: 9.4.2013.

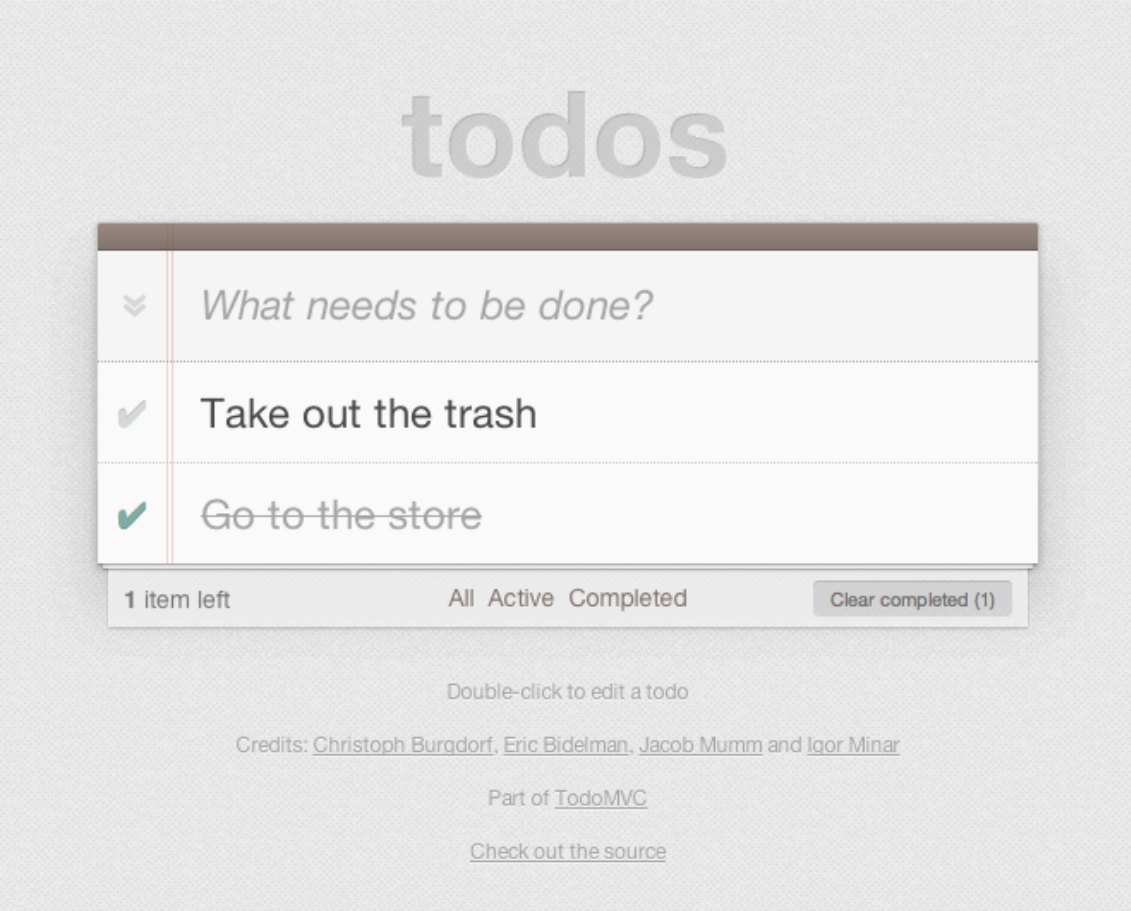


Figure 20. The TodoMVC application interface

Well-known developers such as Paul Irish (of Google Chrome) and Michael Mahemoff (author of Ajax Design Patterns[56]) have also commented on the importance of TodoMVC[57]:

> "TodoMVC is a godsend for helping developers find what well-developed frameworks match their mental model of application architecture." – Paul Irish

> "Modern JavaScript developers realise an MVC framework is essential for managing the complexity of their apps. TodoMVC is a fabulous community contribution that helps developers compare frameworks on the basis of actual project code, not just claims and anecdotes." – Michael Mahemoff

This thesis examines the default (**not** optimized or module loaded) Backbone.js and AngularJS applications available on April 9, 2013 at http://todomvc.com/architecture-examples/backbone/ respectively http://todomvc.com/architecture-examples/angularjs.

[56] http://shop.oreilly.com/product/9780596101800.do

5.1.1 Source Lines of Code

Source Lines of Code (SLOC) is a widely accepted way to measure code quality by comparing how many lines of code is required to accomplish a task. SLOC is typically used to estimate how maintainability and programming productivity in software development. Experiments have showed that effort can be highly correlated with SLOC and that applications with larger SLOC values take more time to develop.

In this thesis, the open source tool CLOC (Count Lines of Code) version 1.58 has been used to compare SLOC values between the AngularJS and Backbone.js version of the TodoMVC application. The first table for each JavaScript MV* reflect the whole application. The second table reflects only the JavaScript components that the developer had to write himself when building the application.

AngularJS

```
http://cloc.sourceforge.net v 1.58  T=1.0 s (9.0 files/s, 15387.0 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Javascript                       7           1480           6963           6460
CSS                              1             48              7            359
HTML                             1              0              0             70
-------------------------------------------------------------------------------
SUM:                             9           1528           6970           6889
-------------------------------------------------------------------------------
Components: AngularJS

http://cloc.sourceforge.net v 1.58  T=0.5 s (10.0 files/s, 264.0 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Javascript                       5             20             25             87
-------------------------------------------------------------------------------
SUM:                             5             20             25             87
-------------------------------------------------------------------------------
```

Backbone.js

```
http://cloc.sourceforge.net v 1.58  T=0.5 s (26.0 files/s, 26870.0 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Javascript                      11           1970           2265           8722
CSS                              1             48              7            359
HTML                             1              0              0             64
-------------------------------------------------------------------------------
SUM:                            13           2018           2272           9145
-------------------------------------------------------------------------------
Components: Backbone.js, Backbone.localStorage, Underscore.js, jQuery


http://cloc.sourceforge.net v 1.58  T=0.5 s (12.0 files/s, 666.0 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Javascript                       6             60             65            208
-------------------------------------------------------------------------------
SUM:                             6             60             65            208
-------------------------------------------------------------------------------
```

As illustrated in figure 19, the Backbone.js version of the application requires notably less Source Lines of Code than the equivalent application written in AngularJS, both when comparing the full applications and when comparing the business logic of the applications. It is also worth highlighting that Backbone.js depends on the external Backbone.localStorage extension while AngularJS performs local storage with built-in methods.

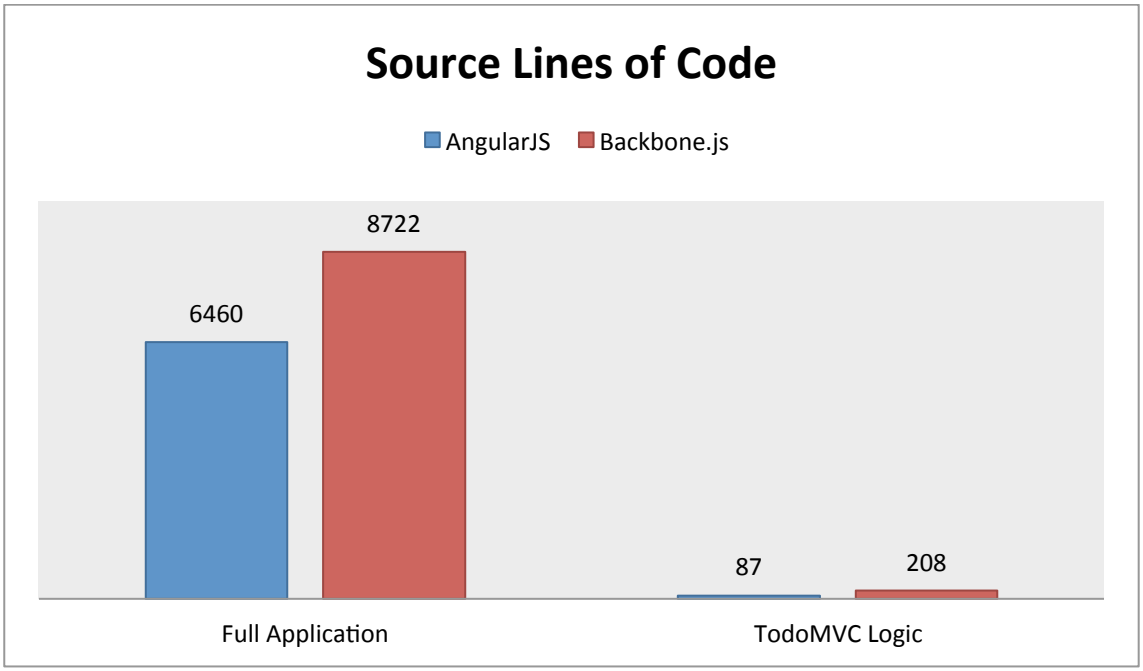


Figure 21. SLOC comparison between TodoMVC in AngularJS and Backbone.js

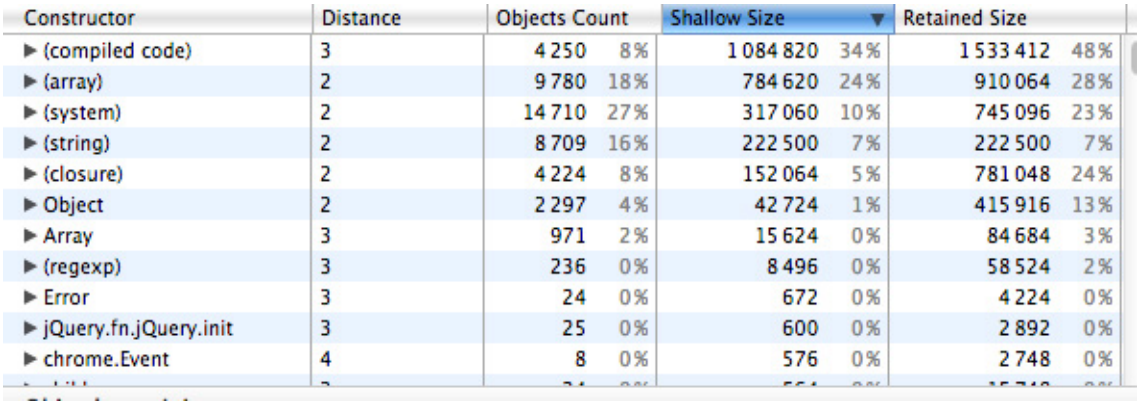
5.1.2 Heap Profile

Chrome Developer tools contain a Heap Profiler that is normally used to track down memoryleaks and show how the application is using its memory. The shallow size of an object it the amount of memory used to store the object without taking referenced objects into consideration. The retained size of an object additionally contains the shallow sizes of objects accessible directly or indirectly **only** from this object.[58]

Constructor	Distance	Objects Count		Shallow Size		Retained Size	
▶ (array)	2	9 466	18%	1 062 568	31%	1 149 376	34%
▶ (compiled code)	3	3 409	6%	876 044	26%	1 285 316	38%
▶ (system)	2	14 986	28%	319 528	9%	643 904	19%
▶ (string)	2	6 919	13%	187 684	6%	187 724	6%
▶ (closure)	2	4 895	9%	176 220	5%	718 640	21%
▶ Object	2	2 484	5%	43 448	1%	397 780	12%
▶ Array	3	1 396	3%	22 336	1%	120 384	4%
▶ Text	2	185	0%	3 692	0%	3 784	0%
▶ NodeList	2	176	0%	3 504	0%	3 664	0%
▶ (regexp)	3	96	0%	3 456	0%	24 524	1%
▶ JQLite	4	90	0%	1 440	0%	8 204	0%

Figure 22. AngularJS Heap Profile

[58] *Shallow and retained sizes*. Available at: http://www.yourkit.com/docs/80/help/sizes.jsp Accessed: 10.4.2013.



Constructor	Distance	Objects Count		Shallow Size		Retained Size	
▶ (compiled code)	3	4 250	8%	1 084 820	34%	1 533 412	48%
▶ (array)	2	9 780	18%	784 620	24%	910 064	28%
▶ (system)	2	14 710	27%	317 060	10%	745 096	23%
▶ (string)	2	8 709	16%	222 500	7%	222 500	7%
▶ (closure)	2	4 224	8%	152 064	5%	781 048	24%
▶ Object	2	2 297	4%	42 724	1%	415 916	13%
▶ Array	3	971	2%	15 624	0%	84 684	3%
▶ (regexp)	3	236	0%	8 496	0%	58 524	2%
▶ Error	3	24	0%	672	0%	4 224	0%
▶ jQuery.fn.jQuery.init	3	25	0%	600	0%	2 892	0%
▶ chrome.Event	4	8	0%	576	0%	2 748	0%

Figure 23. Backbone.js Heap Profile

Figure 22 and figure 23 shows the memory usage of AngularJS and Backbone.js after having added 10 tasks to the application. The differences between both applications can be considered too small to be of any real relevance, but the retained size of the compiled code is slightly higher in the Backbone.js application.

5.2 Functional Testing

Functional tests, unlike Unit tests are written to test the behavior of a system. The tests confirm that the system does what it is intended to do, without going into as much detail by examining the individual components as Unit testing does. Unit tests are written to ensure that a method produces the expected output when given a known input.[59]

The tests being run within this thesis are by their nature functional tests, but the tests themselves do not ensure any functionality but simply automates a create, update and destroy action.

5.2.1 PhantomJS

While PhantomJS is not a test framework in itself, it is used to launch tests with other frameworks. [“PhantomJS is a headless WebKit scriptable with JavaScript API. It has

[59] *Unit Testing versus Functional Tests.* Available at: http://www.softwaretestingtricks.com/2007/01/unit-testing-versus-functional-tests.html Accessed: 10.4.2013.

fast and native support for various web standards: DOM handling, CSS selector, JSON, Canvas, and SVG."][60]

In the setup of this thesis, for testing reasons, PhantomJS was set to include a minified jQuery from a server running on the local host, with an average response time of 8ms.

The JavaScript source code for the constructed PhantomJS test is publicly available at https://gist.github.com/eoy/5356255/53079f819054f4972f29fcb1bb10557b13eb3c99 and will be accepting contributions. Because Github retains every revision of "Gists" on their service, the original source will remain intact despite improvements. The full source follows:

```
const PHANTOM_FUNCTION_PREFIX = '/* PHANTOM_FUNCTION */';
var page = require('webpage').create(),
    system = require('system'),
    t,
    address,
    url = system.args[1],
    t = Date.now(),
    length;

page.onInitialized = function() {
  page.evaluate(function(domContentLoadedMsg) {
    document.addEventListener('DOMContentLoaded', function() {
      window.callPhantom('--------------- START ---------------');
    }, false);
  });
};

console.log('--------------- START ---------------');

page.onCallback = function() {
  console.log('DOMContentLoaded');
  page.onConsoleMessage = function(msg) {
    if (msg.indexOf(PHANTOM_FUNCTION_PREFIX) === 0) {
      eval('(' + msg + ')()');
    } else {
      console.log(msg);
    }
  };
  // Fetch jQuery for easier selectors
  page.includeJs("http://localhost:3000/assets/jquery.min.js", function() {
    var t = Date.now();
```

[60] *PhantomJS: Headless Webkit with JavaScript API*. Available at: http://phantomjs.org/ Accessed: 10.4.2013.

```
    // Print out the title of the page
    var title = page.evaluate(function() {
        return document.title
    });
    console.log(title);

    // Create 10000 todos
    console.log("Creating todos...");
    for (var i=0; i<1000; i++) {
      page.sendEvent('keypress', 'This is todo number: '+i+'\n');
    }
    // page.render('resluts.png');

    // Print the number of todos to confirm
    var length = page.evaluate(function() {
      return $('#todo-list li').length
    });
    console.log(length +" Created");

    // Mark the todos as complete
    console.log("Marking todos as complete...");
    page.evaluate(function() {
      $('#todo-list li').each(function(){
        $(this).find('input').click();
      });
      return true;
    });
    // page.render('resluts2.png');


    // Destroy all todos
    console.log("Destroying todos...");
    page.evaluate(function() {
      $('.destroy').click();
    });
    // page.render('resluts3.png');

    // Print out total time
    t = Date.now() - t;
    console.log('Loading time ' + t + ' msec');
    console.log('---------------- END ----------------');
    phantom.exit();
  });
};

page.open(url, function(status) {

});
```

The script creates 1000 To-Do entries and marks them as complete, after which it deletes each entry. While the numbers produced by this test will vary if repeated, depending on the hardware and performance of the environment where the test is run, the relation between the values should persist. The result of running this script is as follows:

Backbone.js

```
--------------- START ---------------
DOMContentLoaded
Backbone.js • TodoMVC
Creating todos...
1000 Created
Marking todos as complete...
Destroying todos...
Loading time 51831 msec
---------------- END ----------------
```

AngularJS

```
--------------- START ---------------
DOMContentLoaded
AngularJS • TodoMVC
Creating todos...
1000 Created
Marking todos as complete...
Destroying todos...
Loading time 229432 msec
---------------- END ----------------
```

Backbone.js comes out as the clear winner of this test, completing the script in 22% of the time it takes for AngularJS.

In an attempt to better understand the results, the script was adjusted to only output a single character instead of a title for each To-Do entry, which resulted in AngularJS reducing the execution time to **112672ms**. Performing the same single character test with Backbone.js resulted in an execution time of **44894ms** – still much faster than AngularJS. While reducing the character count made the AngularJS application almost double in performance, it had very little impact on the Backbone.js application. The same tests were also performed with only 50 entries, and there the difference was a mere 61ms. The combined results of the tests are illustrated in Figure 20.

The script was adjusted again to instead of performing create, update and destroy actions while the DOM contains a thousand entries, perform the actions a thousand times with one individual entry. In this case AngularJS came out as the winner with **10712ms** versus the **14730ms** execution time of Backbone.js. The adjusted script is available in the appendix.

6 DISCUSSION

As illustrated by figure 24, AngularJS has a hard time keeping up with Backbone.js in the TodoMVC application if the number of simultaneous entries reaches abnormal amounts. In the case of Angular, the difference can be somewhat reduced by decreasing the number of characters in each entry, while it hardly affects Backbone.js by a notable amount. This can be considered the result of the two-way data binding of AngularJS.

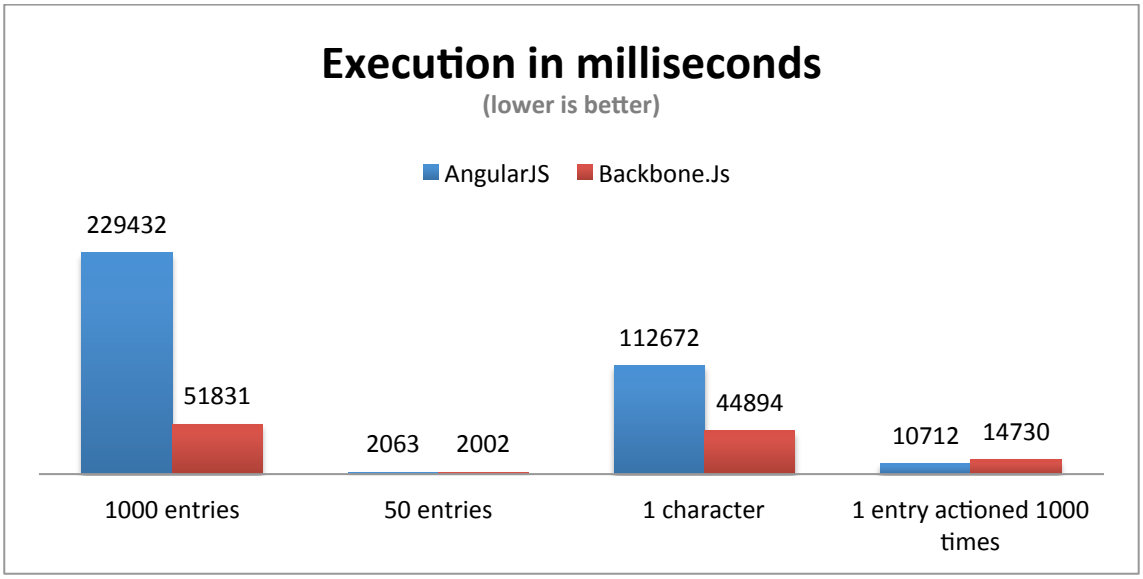


Figure 24. Comparison of execution times between AngularJS and Backbone.js on TodoMVC.

When the script is adjusted to only contain a single entry in the DOM at any given time, AngularJS outperforms Backbone.js, and it seems as if the biggest struggle for AngularJS comes out of handling a DOM containing multiple entries. It is worth to point out that having 1000 entries simultaneously inside a To-Do application is neither intended, nor a normal use case, and that under normal use the difference is too small to be detectable by a user. I do not consider this to be reason enough to pick one framework

over the other, unless one deals with thousands of models at the same time – something that should be avoided.

The heap snapshots sadly proved to be quite useless for this research, as the differences in numbers were too small to be of any real significance. For future research, I recommend comparing the difference in the profile for each application after adding and removing a big number of models. Through this method there is a chance to identify memory leaks in both applications.

I believe that an, in comparison, more important factor when deciding between the two frameworks is the amount of SLOC (Source Lines of Code) written to make both applications perform identically. The more you look at Backbone.js, the more you come to realize how much is left to the user to either develop himself or extend with the help of publicly available extensions. Being able to accomplish the logic of the TodoMVC in less than half the amount of code (87 SLOC versus 208 SLOC) certainly makes AngularJS seem the better choice, but where AngularJS aims to be a full framework Backbone.js takes a step back and remains just the "backbone"; the core scaffolding of the application. Backbone.js states that ["Backbone is a library, not a framework"][…][61] on its official homepage, and in the end that is not too far from the truth. For larger applications you should optimize for ease of maintenance rather than raw performance. Choosing Backbone, one **will** be forced to type more, and maintainability is subject to fluctuate depending on the developer and number of extensions used.

AngularJS is a relatively opinionated framework but that does not mean Backbone.js comes without opinions: Backbone is very clear about how it wants you to extend its views and models, and is highly targeted towards REST. That Backbone.js is so focused on REST might sound like a huge disadvantage, but it is quite easily amendable by overriding the Backbone.sync function. If you are still in need of a more opinionated framework, there are many extensions (such as Marionette) that will add opinions to Backbone, in many cases reducing the amount of boilerplate needed to perform simple tasks such as serializing a form and adding a new model to a collection.

[61] *Backbone.js | Why use Backbone, not [other framework X]?*. Available at: http://backbonejs.org/#FAQ-why-backbone Accessed: 11.4.2013.

Fragmented documentation is great downside of not including all the functionality within the default application and depending on outside. Published books rarely cover extensions and neither does the official documentation, and this sends new developers on a treasure hunt for usable extensions with good documentation. Not everyone desires or has the time to participate in such a treasure hunt. Another downside with Backbone.js (for me) is the separation of the application into so many separate files that it becomes difficult to maintain a mental image of how a big application is connected, in comparison to AngularJS where everything feels very intuitive.

In my opinion, AngularJS does have a steeper learning curve. Whereas Backbone.js follows a structure that is fast to learn and familiar to experienced JavaScript developers, AngularJS introduces a several patterns that were new, at least to me. Add to that the relative lack of literature and you have a framework that is, as of now, quite hard to access. The documentation of AngularJS is, however, absolutely excellent and is searchable and contains many practical examples.

I believe there is more than enough room for both frameworks and that one does not necessarily replace the other as they are targeted towards quite different ideologies. Both frameworks are more than capable to serve the needs of a modern web application, and in the end I believe it comes down to personal preference more than performance. If you enjoy an opinionated framework with two-way data bindings then AngularJS is a perfect choice, but if you want to pick and choose and construct your own framework according to your own needs then Backbone.js might hit closer to home. Because of AngularJS's bindings and directives it could sometimes feel quite magical and juvenile, contrary to Backbone.js that always felt quite dry and mature to me. The question of if you want your work-tool to be magical or dry is up to you, but I lean towards Backbone.js for its dully-logical structure.

REFERENCES

Crockford, Douglas. 2008, *JavaScript: The Good Parts*. O'Reilly Media, Inc. Retrieved 21.3.2013. 172 pages. ISBN 978-0-596-15873-6.

Dietz, Frederik. 2013, *Recipes with Angular.js; beta version*. Leanpub. Retrieved 28.3.213. 94 pages.

Ford, Brian & Ruebbelke, Lukas. 2012, *AngularJS in Action – Early Access Edition*. Manning Publications Co. Retrieved 28.3.2013. 38 pages. ISBN 978-1-6172-9133-3.

Green, Brad & Seshadri, Shyam. 2013. *AngularJS – Less Code, More Fun and Enhanced Productivity with Structured Web Apps*. O'Reilly Media, Inc. 196 pages. ISBN 978-1-449-34485-6.

Haverbeke, Marijn. 2011, *Eloquent JavaScript*. No Starch Press, Inc. Retrieved 28.3.2013. 199 pages. ISBN-10 1593272820.

MacCaw, Alex. 2011, *JavaScript Web Applications*. O'Reilly Media, Inc. Retrieved 1.9.2011. 282 pages. ISBN 978-1-4493-0380-8.

MacCaw, Alex. 2012, *The Little Book on CoffeeScript*. O'Reilly Media, Inc. Retrieved 3.2.2012. 62 pages. ISBN 978-1-4493-2105-5.

Mardanov, Azat. 2013, *Rapid Prototyping with JS; Version 0.4*. Leanpub. Retrieved 28.3.2013. 149 pages.

Morrison, Jason; Pytel, Chad; Quaranto, Nick; Giménez, Harold; Clayton, Joshua; Berke-Williams, Gabe & Mazzola, Chad. 2012, *Backbone.js on Rails*. Retrieved 28.3.2013. 139 pages.

Osmani, Addy. 2012, *Developing Backbone.js Applications – Early Release*. O'Reilly Media, Inc. Retrieved 28.3.2013. 308 pages. ISBN 978-1-4493-3603-5.

Reisig, John & Bibeault, Bear. 2012, *Secrets of the JavaScript Ninja*. Manning Publications Co. 392 pages. Retrieved 2.1.2013. ISBN 978-1-933-98869-6.

ONLINE REFERENCES

Appleton, Andy. *Diving into AngularJS*, published 21.4.2013. Available at: http://floatleft.com/notebook/diving-into-angularjs Accessed: 26.4.2013.

Takada, Mikito. *Single Page Apps in Depth*, published 28.6.2012. Available at: http://singlepageappbook.com/index.html Accessed: 20.4.2013.

Bailey, Derick. *3 Stages Of A Backbone Application's Startup*, published 6.2.2012. Available at: http://lostechies.com/derickbailey/2012/02/06/3-stages-of-a-backbone-applications-startup/ Accessed: 20.4.2013.

Bailey, Derick. *Backbone.js is Not An MVC Framework*, published 23.12.2011. Available at: http://lostechies.com/derickbailey/2011/12/23/backbone-js-is-not-an-mvc-framework/ Accessed: 20.4.2013.

Gupta, Siddhartha. *Getting Cozy with Underscore.js*, published 31.3.2012. Available at: http://net.tutsplus.com/tutorials/javascript-ajax/getting-cozy-with-underscore-js/ Accessed: 20.4.2013.

Raw, Craig. How JavaScript is creating a web development renaissance, published 6.9.2012. Available at: http://memeburn.com/2012/09/how-javascript-is-creating-a-web-development-renaissance/ Accessed: 19.4.2013.

15,000 Raspberry Pis for UK schools - thanks Google!, Available at: http://www.raspberrypi.org/archives/3158 Accessed: 10.4.2013.
AngularJS: $http. Available at: http://docs.angularjs.org/api/ng.$http Accessed: 6.4.2013.
AngularJS: Conceptual Overview. Available at: http://docs.angularjs.org/guide/concepts#directives Accessed: 20.4.2013.
AngularJS: Data Binding in Angular. Available at: http://docs.angularjs.org/guide/dev_guide.templates.databinding Accessed: 9.4.2013.
AngularJS: Understanding Angular Templates. Available at: http://docs.angularjs.org/guide/dev_guide.templates Accessed: 10.4.2013.
Anybody can learn, Available at: http://www.code.org/ Accessed: 10.4.2013.
Backbone.js | Backbone.Events. Available at: http://backbonejs.org/#Events Accessed: 6.4.2013.
Backbone.js | Examples. Available at: http://backbonejs.org/#examples Accessed: 20.2.2013.
Backbone.js | Extending Backbone. Available at: http://backbonejs.org/#FAQ-extending Accessed: 7.4.2013.
Backbone.js | How does Backbone relate to "traditional MVC?. Available at: http://backbonejs.org/#FAQ-mvc Accessed: 28.3.2013.
Backbone.js | There's More Than One Way To Do it. Available at: http://backbonejs.org/#FAQ-tim-toady Accessed: 8.4.2013.
Backbone.js | Why use Backbone, not [other framework]?. Available at: http://backbonejs.org/#FAQ-why-backbone Accessed: 12.4.2013.
Bailey, Derick. *3 Stages of a Backbone Application's* Startup, published 6.2.2012. Available at: http://lostechies.com/derickbailey/2012/02/06/3-stages-of-a-backbone-applications-startup/ Accessed: 28.3.2013.
Bailey, Derick. *Backbone.js Is Not An MVC* Framework, published 23.12.2011. Available at: http://lostechies.com/derickbailey/2011/12/23/backbone-js-is-not-an-mvc-framework/ Accessed: 6.4.2013.
Built with AngularJS. Available at: http://builtwith.angularjs.org/ Accessed: 20.2.2013.
Databinding in AngularJS. Available at: http://stackoverflow.com/questions/9682092/databinding-in-angularjs#answer-9693933 Accessed: 9.4.2013.
Extensions, Plugins, Resources. Available at: https://github.com/documentcloud/backbone/wiki/Extensions,-Plugins,-Resources Accessed: 9.4.2013.
Gupta, Raj. *Backbone.js vs AngularJS : Demystifying the Myths*, published 27.12.2012. Available at: http://www.nebithi.com/2012/12/27/backbone-and-angular-demystifying-the-myths/ Accessed: 22.3.2013.
Gupta, Siddhartha, *Getting Cozy With Underscore.js*, published 31.3.2013. Available at: http://net.tutsplus.com/tutorials/javascript-ajax/getting-cozy-with-underscore-js/ Accessed: 5.4.2013.
Gotcha (programming). Available at: http://en.wikipedia.org/wiki/Gotcha_(programming) Accessed: 9.4.2013.
JSON. Available at: http://json.org Accessed: 20.4.2013.
jQuery. Available at: http://jquery.com Accessed: 18.4.2013.
jQuery Usage Statistics. Available at: http://trends.builtwith.com/javascript/JQuery Accessed: 18.4.2013.

Mann, Brian. 2013, *Backbone Rails – Client Side Development*. Available at: http://vimeo.com/58787395 Accessed: 28.4.2013.

Minar, Igor. *MVC vs MVVM vs MVP*, published 19.7.2012. Available at: https://plus.google.com/+AngularJS/posts/aZNVhj355G2 Accessed: 19.3.2013.

Model-View-ViewModel. Available at: http://en.wikipedia.org/wiki/Model_View_ViewModel Accessed: 29.3.2013.

Osmani, Addy. *Journey Through the JavaScript MVC Jungle,* published 27.6.2012. Available at: http://coding.smashingmagazine.com/2012/07/27/journey-through-the-javascript-mvc-jungle/ Accessed: 20.2.2013.

PhantomJS: Headless Webkit with JavaScript API. Available at: http://phantomjs.org/ Accessed: 10.4.2013.

Shallow and retained sizes. Available at: http://www.yourkit.com/docs/80/help/sizes.jsp Accessed: 10.4.2013.

TodoMVC. Available at: http://todomvc.com/ Accessed: 9.4.2013.

Underscore.js. Available at: http://underscorejs.org/ Accessed: 5.4.2013.

Unit Testing versus Functional Tests. Available at: http://www.softwaretestingtricks.com/2007/01/unit-testing-versus-functional-tests.html Accessed: 10.4.2013.

Zakas, Nicholas C. *How many users have JavaScript disabled?*, published 13.10.2013. Available at: http://developer.yahoo.com/blogs/ydn/posts/2010/10/how-many-users-have-javascript-disabled/ Accessed: 28.4.2013.

APPENDICES

1. Modified version of the TodoMVC test

```
const PHANTOM_FUNCTION_PREFIX = '/* PHANTOM_FUNCTION */';
var page = require('webpage').create(),
    system = require('system'),
    t,
    address,
    url = system.args[1],
    t = Date.now(),
    length;

page.onInitialized = function() {
  page.evaluate(function(domContentLoadedMsg) {
    document.addEventListener('DOMContentLoaded', function() {
      window.callPhantom('--------------- START ---------------');
    }, false);
  });
};

console.log('--------------- START ---------------');

page.onCallback = function() {
  console.log('DOMContentLoaded');
  page.onConsoleMessage = function(msg) {
    if (msg.indexOf(PHANTOM_FUNCTION_PREFIX) === 0) {
      eval('(' + msg + ')()');
    } else {
      console.log(msg);
    }
  };
  // Fetch jQuery for easier selectors
  page.includeJs("http://localhost:3000/assets/jquery.min.js", function() {
    var t = Date.now();

    // Print out the title of the page
    var title = page.evaluate(function() {
        return document.title
    });
    console.log(title);

    // Create 10000 todos
    console.log("Creating todos...");
    for (var i=0; i<1000; i++) {
      page.sendEvent('keypress', 'This is todo number: '+i+'\n');
      page.evaluate(function() {
        $('#todo-list li').each(function(){
          $(this).find('input').click();
        });
        return true;
      });
      page.evaluate(function() {
        $('.destroy').click();
```

```
      });
    }

    // Print the number of todos to confirm
    var length = page.evaluate(function() {
      return $('#todo-list li').length
    });
    console.log(length +"  To-Dos remain");

    // Print out total time
    t = Date.now() - t;
    console.log('Loading time ' + t + ' msec');
    console.log('---------------- END ----------------');
    phantom.exit();
  });
};

page.open(url, function(status) {

});
```