

Aalto University  
School of Science  
Master's Programme in ICT Innovation

Taha Kachwala

# Managing Data Visualization Pipeline with Backbone.js and D3.js

Improving overall software efficiency using  
Automated Build Pipeline with Gulp

Master's Thesis  
Espoo, 10.10.2016

Supervisor: Prof. Petri Vuorimaa  
Instructor: Pertti Lounamaa



<b>Author:</b>	Taha Kachwala		
<b>Title:</b>	Managing Data Visualization Pipeline with Backbone.js and D3.js. Improving overall software efficiency using Automated Build Pipeline with Gulp		
<b>Date:</b>	10.10.2016	<b>Pages:</b>	87
<b>Professorship:</b>	Digital Media Technology	<b>Code:</b>	T-111
<b>Supervisor:</b>	Prof. Petri Vuorimaa		
<b>Instructor:</b>	Petri Lounamaa Ph.D. (Economic Systems)		
<p>This thesis studies how a Model-View-Controller (MV*) framework can be integrated into a Data Visualization Pipeline. Specifically, this thesis aims to cover the challenges related in integrating an MV* framework like Backbone.js with D3.js, which is a popular JavaScript based visualization library. Additionally, it also evaluates another sub-topic regarding task runners, which are tools claiming to automate manual tasks as well as streamlining the build process.</p> <p>Data Visualization has become an important aspect for many web applications. Developers need to employ sophisticated mechanisms to provide interactive visualizations. This requires separation of concerns within the visualization pipeline which is achieved with the combination of Backbone.js and D3.js. Evaluation of this methodology indicates that such a combination enables the application to be more versatile and robust while also improving performance.</p> <p>In recent years, client-side web applications have become increasingly complex. A typical web application on average requires between 10-60 external open-source JavaScript libraries. Application development also requires the use of several other tools and performing manual tasks. Managing all these libraries and tools can create a bottleneck, and task-runners like Gulp aim to address these issues. This thesis implements an Automated Build Pipeline that can streamline the build process and automates all the manual tasks. The results prove significant performance and overall efficiency benefits. However, it also highlights a few serious drawbacks. Moreover, this thesis also covers some of the best practices employed by skilled front-end web developers.</p>			
<b>Keywords:</b>	Data Visualization, JavaScript, MVC, Web Application, Gulp		
<b>Language:</b>	English		

# Acknowledgements

First, I would like to thank my supervisor, professor Petri Vuorimaa, to provide me guidance throughout my internship and thesis.

Furthermore, I would also like to thank my employer Pajat Solutions to provide me with a challenging internship and an interesting thesis topic. I would especially like to thank the CEO Pertti Lounamaa, for providing me with the flexibility and freedom to make decisions regarding the product development of Poimapper.

Espoo, October 10, 2016

Taha Kachwala

# Abbreviations & Acronyms

API	Application Programming Interface
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
InfoVis	Information Visualization
JS	JavaScript or JavaScript file
JSON	JavaScript Object Notation
JSP	Java Server Pages
MVC, MV*	Model View Controller Framework
NPM	Node Package Manager
POI	Point of Interest
REST	Representational State Transfer
SLOC	<i>Source Lines of Code</i>
UI	User Interface

# Contents

<b>1. Introduction</b>	<b>7</b>
1.1 Motive for the Research	7
1.2 Research Objectives	9
1.3 Target Group	9
1.4 Poimapper	9
1.5 Structure of Thesis	10
<b>2. Product Description - Poimapper</b>	<b>11</b>
2.1 Form Builder	12
2.2 Data Viewer	13
2.3 Hierarchy Editor	15
2.4 Admin and Scheduler	15
<b>3. Data Visualization and D3.js</b>	<b>16</b>
3.1 Definition	16
3.2 Branches of Visualization	17
3.3 InfoVis	18
3.3.1 Attributes of Visualization	18
3.3.2 Principles of Visualization	18
3.3.3 The Visualization Pipeline	27
3.4 Creating Visualizations using D3.js	27
3.4.1 Chart Object	27
3.4.2 Scalable Vector Graphics (SVG)	28
3.4.3 D3-Scales	29
3.4.4 D3-Axes	30
3.4.5 Legends and Tooltips	31
3.4.6 Bar Charts	31
3.4.7 Pie Charts	34

3.4.8	Line Charts	35
3.4.9	Area Charts	35
<b>4.</b>	<b>Backbone.js - An MV* Framework</b>	<b>37</b>
4.1	Need for MV* Frameworks	37
4.2	Model-View-Controller	38
4.2.1	Model	39
4.2.2	Controller	39
4.2.3	Views	39
4.2.4	Routers	40
4.3	More about Backbone.js	40
4.3.1	Underscore.js	41
4.3.2	Agnostic Templating	42
4.3.3	MV*	42
4.3.4	Clean HTML	46
4.3.5	Extensions	46
<b>5.</b>	<b>Implementing the Data Visualization Pipeline</b>	<b>47</b>
5.1	Specifications of a Web Report	47
5.2	Architecture of Web Reports Module	48
5.2.1	Models	49
5.2.2	Collections	50
5.2.3	Views	51
5.3	Adding New Features	54
<b>6.</b>	<b>Automated Build Pipeline using Gulp</b>	<b>55</b>
6.1	Miscellaneous Tools	56
6.1.1	CSS Preprocessors	56
6.1.2	CoffeeScript	56
6.1.3	Bower	59
6.2	Gulp	60
6.2.1	Installation and Setup	60
6.2.2	Gulp Workflow	60
6.2.3	Helper Plugins	61
6.2.4	Build Pipeline	64
6.2.6	Gulp Tasks	66
<b>7.</b>	<b>Evaluation</b>	<b>74</b>
7.1	Evaluation of Data Visualization Pipeline with Backbone.js and D3.js	74
7.1.1	Data-Binding	74
7.1.2	Performance Analysis	75
7.1.3	Loose Coupling and High Cohesion	76
7.2	Evaluation of Automated Build Pipeline using Gulp	77

7.2.1 Improved Developer Efficiency	77
7.2.2 Performance Gains using Gulp	77
7.2.3 Page Speed Comparisons	79
7.2.4 Disadvantages of Gulp	79
<b>8. Discussion and Conclusion</b>	<b>81</b>
8.1 Future Work	82
<b>Appendix A</b>	<b>86</b>

# Chapter 1

## Introduction

### 1.1 Motive for the Research

The client-side applications using JavaScript are becoming increasingly complex. One factor contributing to this fact is that there are thousands of libraries available through open-source contributions that help in solving a variety of issues and few even introducing new ones. These libraries depend on other libraries, and in no time a web application might become an explosive cocktail that is impossible to contain. In recent years, Node.js has enabled JavaScript to be used as a server-side language as well, and some libraries like Browserify<sup>1</sup> propose to write even the front-end applications with server-side Node.js logic. Furthermore, there are different families of open-source libraries and tools that claim to solve various problems in various stages of software development cycle. One such family is of JavaScript Task Runners. This family consists of several libraries to choose from, like GRUNT<sup>2</sup>, Gulp<sup>3</sup>, Cake<sup>4</sup> and Broccoli.js<sup>5</sup>. They all claim superiority over others in certain aspects, however the leaders in this race are Grunt and Gulp. These task runners aim to automate various manual tasks like compiling scripts, minification, creating build pipelines, testing and several more and help tackle various issues surrounding JavaScript. Also, a different set of files needs to be served for development and production purposes. They claim to handle all these tasks seamlessly and ease the software development cycle, but the question is, is it really required?

In addition, JavaScript front-end MVCs (Model-View-Controller) is another family of open-source libraries and tools. They aim to address the logical complexities of the Web applications. MVCs is a very powerful software architecture concept and have

---

<sup>1</sup> "Browserify." 2013. 8 Sep. 2016 <<http://browserify.org/>>

<sup>2</sup> "Grunt: The JavaScript Task Runner." 2012. 8 Sep. 2016 <<http://gruntjs.com/>>

<sup>3</sup> "gulp.js - the streaming build system." 2013. 8 Sep. 2016 <<http://gulpjs.com/>>

<sup>4</sup> "cake.coffee - CoffeeScript." 2010. 8 Sep. 2016 <<http://coffeescript.org/documentation/docs/cake.html>>

<sup>5</sup> "Broccoli.js - The asset pipeline for ambitious applications." 2014. 8 Sep. 2016 <<http://broccolijs.com/>>



effectively helped solving a variety of issues on server-side programming languages. Some of the most powerful ones being Spring<sup>6</sup> for Java, Django<sup>7</sup> for Python and Express.js<sup>8</sup> for Node. In the past few years, increasing amounts of application logic is being implemented on the client-side rather than on the server-side<sup>9</sup>. This has led to the development of many front-end JavaScript based MVC frameworks like Backbone.js, Knockout.js, Ember.js, Angular.js and React.js. They too claim to be superior over others in some or the other aspect, and there have been several studies comparing the benefits and downsides of using these frameworks. However, I would like to find out if a MVC can help in tackling the challenges of Data Visualization, that have become an important aspect for front-end web applications. I planned to test this idea while I was working as an intern at Poimapper (see chapter 2). I was given the responsibility to create interactive Web Reports containing visualizations for different types of data. In the past, I have created visualizations and dashboards (figure 1.1) using purely JavaScript with D3.js- A JavaScript based visualization library. However, I soon realised that handling the entire Visualization Pipeline (see section 3.3.3) with such a complex set of data is extremely difficult if not impossible by using pure JavaScript.



Figure 1.1: A simple Dashboard

<sup>6</sup> "22. Web MVC framework - Spring." 2013. 8 Sep. 2016 <<http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html>>

<sup>7</sup> "Django: The Web framework for perfectionists with deadlines." 2011. 8 Sep. 2016 <<https://www.djangoproject.com/>>

<sup>8</sup> "Express - Node.js web application framework." 2015. 8 Sep. 2016 <<https://expressjs.com/>>

<sup>9</sup> Runeberg, Joakim. "To-Do with JavaScript MV\*: A study into the differences between Backbone. js and AngularJS." (2013).

## 1.2 Research Objectives

The main purpose of creating Web Reports for Poimapper is to provide our users with the answer of a simple question, “What is happening?”. The user can then dive into more details by interacting with the charts. I knew from my past experience that the process can become extremely complicated when the logic of creating charts (Chart Logic) gets intertwined with the actual application logic. Previously, I have ended in making several flavors of the same type of charts for different types of interaction instead of reusing the code. It would be interesting to test if an MV\* framework can effectively handle the challenge of separation of concerns within the Data Visualization Pipeline as described in section 3.3.3.

Furthermore, there are several important practices that also need to be followed to develop and manage a web application. JavaScript task runners claim to be an easy to use tool to conveniently implement these practices. Since Poimapper was missing a task runner, I planned to experiment with Gulp and test if it really helps.

Therefore, to test these hypotheses, I would like to experiment with two most important families of the JavaScript open-source libraries, namely, Task Runners and MV\* frameworks:

**RQ1:** Does a JavaScript task runner help improve overall software efficiency and productivity?

**RQ2:** Can a MV\* framework like Backbone.js help in separation of concerns within the Data Visualization Pipeline?

## 1.3 Target Group

The target audiences of this thesis are developers and researchers already familiar to some concepts of Data Visualization, Model-View-Controller frameworks and JavaScript. However, I have provided sufficient background for novice JavaScript developers interested in developing web applications and visualizations with minimal boilerplate.

## 1.4 Poimapper

During my internship at Pajat Solutions, I was working on their core product, *Poimapper*<sup>10</sup>. Poimapper is a fairly sophisticated product, but to put it simply it is a Mobile data collection platform to cater the needs of on-field teams such as industrial inspection, facility management, market research, inspections, audit, and sales reporting. It allows mobile users to collect, share, and visualize geotagged data in real-time.

---

<sup>10</sup> "Poimapper." 2010. 12 May. 2016 <<http://www.poimapper.com/>>

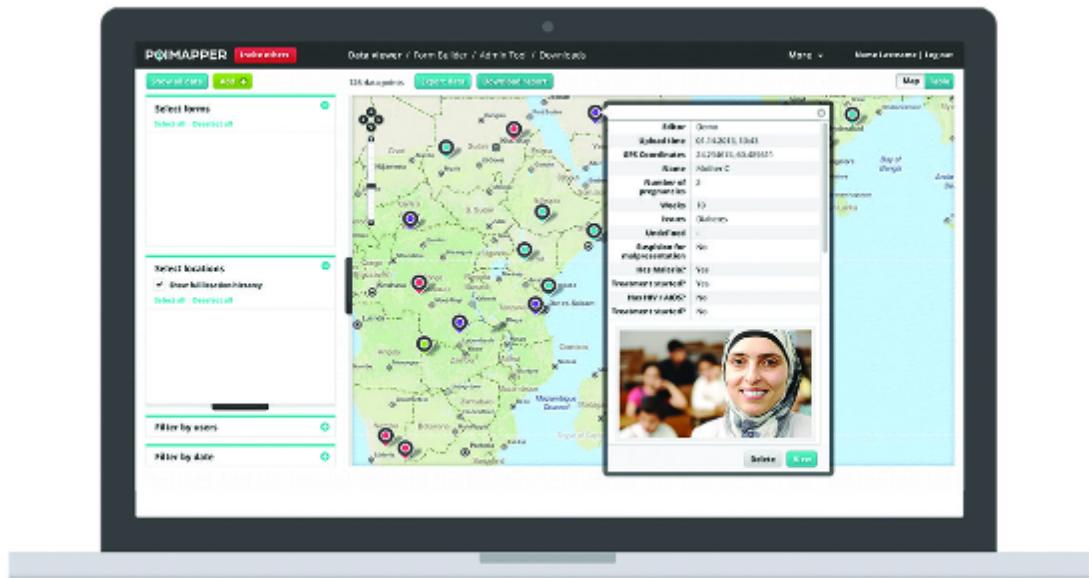


Figure 1.2. Poimapper

## 1.5 Structure of Thesis

Chapter 1 (Introduction) describes the motivation and the problem statement for this thesis. Chapter 2 (Product Description) briefly introduces the product (Poimapper.com) to the audience to have a better understanding of subsequent chapters and decisions taken. Chapter 3 (Data Visualization and D3.js) consists of a detailed discussion regarding the need for data visualization and crucial concepts. In chapter 4, I elaborate the need for MV\* frameworks and introduce the concepts of Backbone.js.

Chapter 5 and chapter 6 have detailed discussion and implementations of Data Visualization Pipeline and Automated Build Pipeline respectively. Chapter 7 (Evaluation) consists of analysis and performance gains using both the pipelines. Finally, conclusions and implications of the thesis are discussed in chapter 8.

## Chapter 2

# Product Description - Poimapper

As briefly discussed in section 1.4, Poimapper<sup>11</sup> allows its users to collect data through forms. For example, a project manager can create multiple forms and then assign specific forms to various teams. These teams collect data by filling out these forms and the data is reported back to the concerned authorities/users.

The main feature of Poimapper is geotagging. This makes Poimapper extremely efficient tool to be used for on-field studies/teams. It is being currently used by a chocolate manufacturing company with an aim to provide 100% slave-free chocolates. To ensure that the chocolates are slave-free, they have to monitor and inspect these farms from where they buy Cocoa. This requires them to have multiple inspection teams spread over a few African nations, and are using Poimapper to manage and process the data collected by them.

Poimapper consists of a Web Application and an iOS and Android app. Currently mobile apps are only used to fill in form data that are mostly used by teams for on-field surveys. All other functionalities are provided by the Web app. The prime aim of Poimapper is simple, users can create forms/surveys and other users can fill them up. However, in principle the product is fairly sophisticated and complex. It can be divided into the following main modules.

1. **Form Builder:** To create and edit forms.
2. **Data Viewer:** Users can view the reports of form data collected.
3. **Hierarchy Editor:** To manage locations and hierarchies of geographical locations. Users can create custom location levels upto four levels such as Continent -> Country -> State -> District.
4. **Admin:** Users with higher access rights can allocate roles to other users. This helps in managing access controls to different types of users.

---

<sup>11</sup> "Poimapper." 2010. 3 Jul. 2016 <<http://www.poimapper.com/home/>>

5. **Scheduler:** Users with higher access rights can allocate particular tasks for a particular user to a certain form on a specific date. It helps in scheduling tasks to on field teams.
6. **Profile Settings:** Generic settings for users like changing passwords etc.

Each of these core modules are developed using Backbone.js, which will be discussed in detail in chapter 4.

## 2.1 Form Builder

Form Builder is used to create new forms and edit existing form templates.

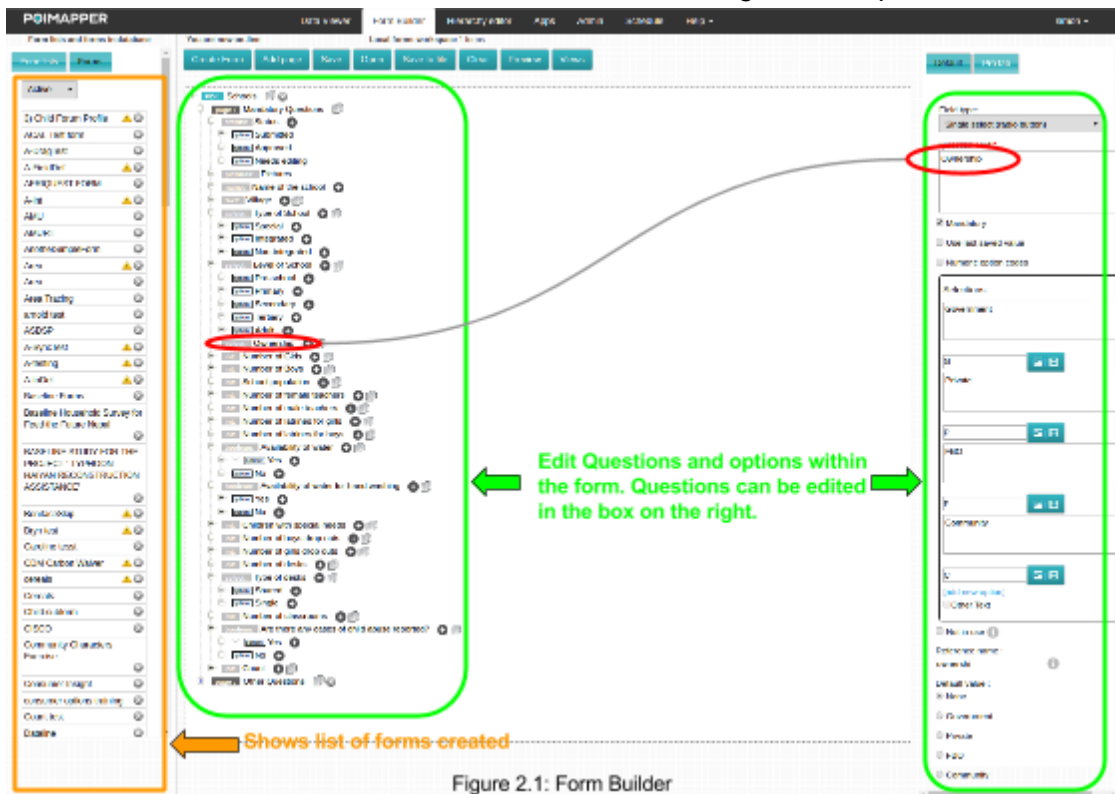


Figure 2.1: Form Builder

As seen in figure 2.1, on the left a list of existing forms are shown. A user can create a form by giving a name. Then by clicking +, Fields/questions can be added. Poimapper provides plenty of field types which can satisfy most of the survey needs. Every question can have the following types.

1. **Text:** To enter textual values. eg. Your name, favorite sport.
2. **Numerical:** Questions that require numeric values to be entered. Can be an Integer or Floating type. eg. Number of people living in a house
3. **Boolean:** Questions that can be answered in Yes or No.
4. **Single-Select:** Select one option from a list of options (e.g., Select your age group).
5. **Multi-Select:** Provides checkboxes to select multiple values from a list of values.
6. **Time of day:** Enter a specific time.
7. **Date:** Enter a date.

8. **Label:** Just provides a label. Can be used to elaborate specific instructions or provide some additional information to users while filling up the forms.
9. **Dynamic Table:** This provides a table where every column can be a specific question of any other basic type. The user answering the form can then add multiple instances of these values. For example, while interviewing workers of a factory, you need to have their Name, Age and allocated team. So for every new worker interviewed, a new row can be added.
10. **Static Table:** Similar to Dynamic Table with the only difference that a predetermined number of rows are available and new rows cannot be added.
11. **Skip Logic:** Provides a way to skip questions if a particular condition is satisfied. e.g., You might need to avoid asking different set of questions for male and female.
12. **Calculated Values:** It is used to derive values from previously answered questions. Suppose there are two questions that are needed; *What is the respondent's birthdate?* and *How old will the respondent be in 10 years.* The second question can be derived from previously answered question.
13. **Sub-Form:** Another form can be added as a sub-form. This helps in creating a generic forms for specific purposes like demographic data, and then use them as a part in other forms.
14. **Lookup Value:** This question type provides a way of having extra attributes within a single select like question and being able to access the attributes from Calculated Value Questions. The option attributes will not be visible. This question also enables accessing of values from data collected in another form.
15. **Miscellaneous Fields:** Some commonly used fields like email, phone number, and Digital Signature.
16. **Validation Rule:** To add custom form validation rules.

## 2.2 Data Viewer

The Data Viewer tab of Poimapper enables users to view the data collected through forms. It provides users with a set of filters that can be used to filter out data collected for specific forms, users, locations and date times. The data collected can be viewed in a Map, Table and through Charts. The *Image* tab shows a gallery of images that have been uploaded through forms.

The Web Reports module consisting of interactive visualizations has been developed by me through the course of my internship and is the main subject for this thesis.

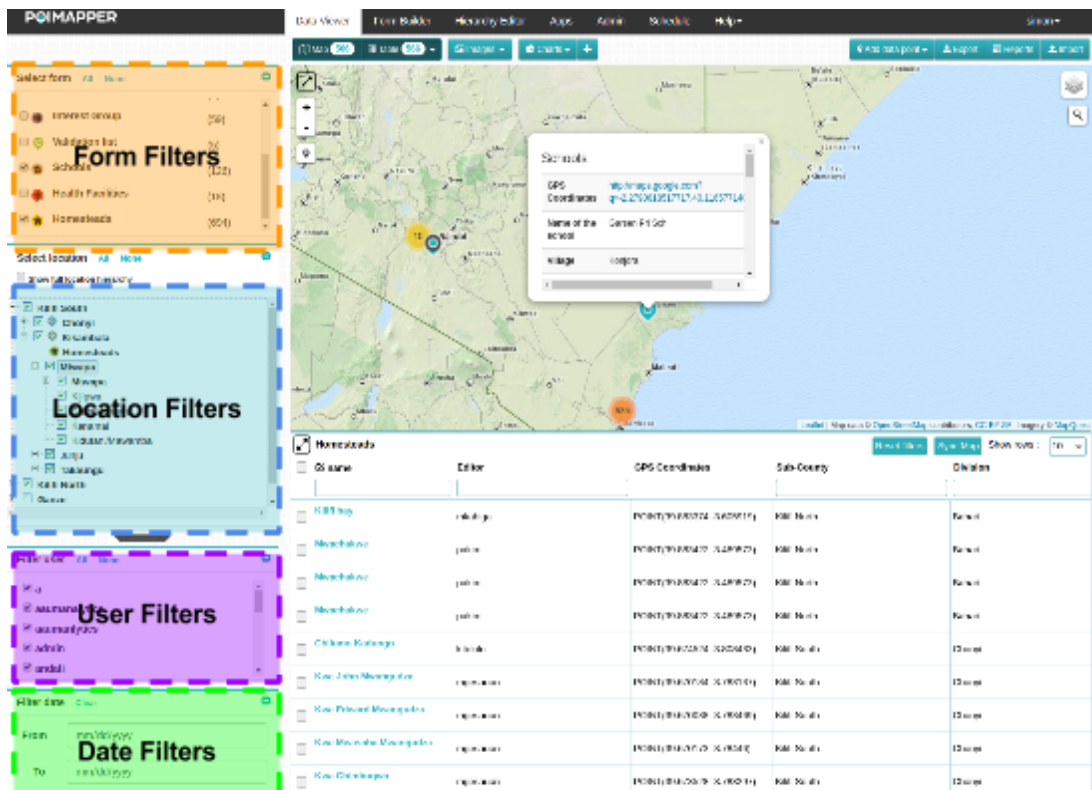


Figure 2.2: Data Viewer

Figure 2.2 illustrates the four filters provided. The figure currently has Map and Table modules active. Whenever a user fills up a form, a POI (Point of Interest) is created. As seen above the Map represents the geographical location where a particular POI was collected. The Table view below the Map displays POI's selected using filters in a table.



Figure 2.3: Web Reports module

The entire page is subdivided and managed by different Backbone Views (explained in section 4.2.3) that handle everything within their DOM components.

FormFilterView handles the filter components provided for Form. MapView and TableView handle events and interactions for the Map and Table respectively. There are also few other Views that handle smaller components and are children of other views. In this thesis, the main discussions will be on ChartView, which creates Web Reports containing data visualizations. Figure 2.3 illustrates the Web Reports module created.

## 2.3 Hierarchy Editor

The Hierarchy Editor is used to set hierarchies among geographical locations. This helps in knowing where the forms were filled and filter data based on specific location hierarchies.



Figure 2.3: Hierarchy Editor

As seen from the figure 2.3, upto 4 levels of location hierarchies can be defined by the user. Their particular locations can be set on the map. In this particular example location hierarchies are defined as Sub-County → Division → Location → Sub-Location.

## 2.4 Admin and Scheduler

The admin tools allow users with roles set as 'Admin' to manage privileges of other users. It also enables them to create user groups and assigning different form lists and locations accordingly.

Scheduler is just an helper tool, that enables the managers and admins to assign data collection tasks to other users.



## Chapter 3

# Data Visualization and D3.js

In recent years, the data generated every day is exceeding the total data generated from the dawn of civilization up until 2003 (Schmidt, 2010, in press <sup>12</sup>). To convert this data into decisions has become an urgent need for many corporations and governments around the world, and this has led to many researchers depend on the field of Data Visualization. The concept of Data Visualization is not new, and has been around for centuries like using maps for navigation for long sea voyages. Human eyes and brains have evolved to easily detect patterns (David McCandless, The beauty of Data Visualization, 2010<sup>13</sup>) and this makes visualization a very effective tools to summarize hundreds or even thousands of pages of data into a few graphs and plots. The use of correct visualization techniques can enable decision makers to observe patterns and connections that matter.

### 3.1 Definition

Data Visualization is the presentation of data in a pictorial or graphical format. It enables decision makers to see analytics presented visually, so they can grasp difficult concepts or identify new patterns.<sup>14</sup>

The minimalistic set of requirements for a successful visualizations according to Kosara (2007, chap. 3), are

1. It is based on (non-visual) data
2. It produces an image
3. The results are readable and recognizable

---

<sup>12</sup> "Eric Schmidt: Every 2 Days We Create As Much ... - TechCrunch." 2010. 15 May. 2016 <<http://techcrunch.com/2010/08/04/schmidt-data/>>

<sup>13</sup> "The beauty of data visualization - David McCandless | TED-Ed." 2012. 15 May. 2016 <<http://ed.ted.com/lessons/david-mccandless-the-beauty-of-data-visualization>>

<sup>14</sup> "Data Visualization: What it is and why matters | SAS." 2014. 15 May. 2016 <[http://www.sas.com/en\\_us/insights/big-data/data-visualization.html](http://www.sas.com/en_us/insights/big-data/data-visualization.html)>

### 3.2 Branches of Visualization

Data Visualization can further be classified as InfoVis (Information Visualization) and SciVis (Volume Visualization-VoVis / Medical Visualization-MedVis / Flow Visualization - FlowVis). InfoVis produces interactive visual representations of abstract data to make it easier for humans to interpret and enabling them to gain knowledge about the internal structure of the data and causal relationships within it. On the other hand, SciVis focusses more on spatial data such as volume visualization (Wolff et al., 1993), Medical Visualization and Flow Visualization . The main differences between the InfoVis and SciVis has been summarized in Table 3.1.

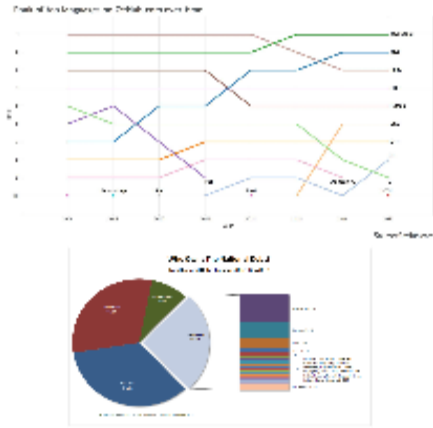
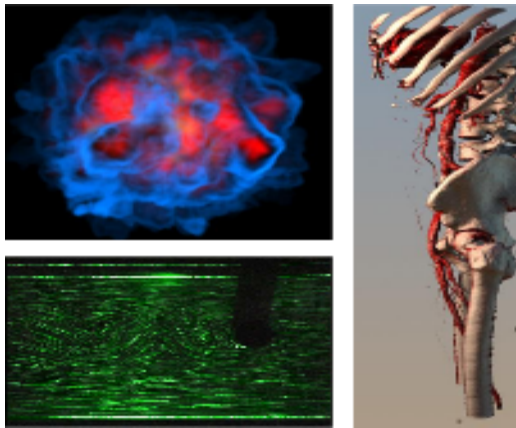
InfoVis	SciVis (VoVis/MedVis/FlowVis..)
Abstract Data	Spatial Data
N-Dimensional and heterogeneous	Mostly 2 or 3-dimensional
Data-Types: Numerical, text, images, multimedia	Data-Types: Scientific, engineering, biomedical
 <p>Source <sup>15</sup></p>	 <p>Source <sup>16</sup></p>

Table 3.1

This thesis will be focussing more on InfoVis.

### 3.3 InfoVis

In the figure 3.1, all the 4 datasets have similar mathematical statistics. If only given such a dataset without visualization, it will be impossible to figure out the

<sup>15</sup> "Discussion on Modal Survey Word Press Poll, Survey & Quiz Plugin ..." 2015. 15 May. 2016 <<http://codecanyon.net/item/modal-survey-wordpress-poll-survey-quiz-plugin/6533863/comments?page=4>>

<sup>16</sup> "Lyceum." 2006. 15 May. 2016 <[http://lyceum-jl.blogspot.com/2006/05/blog-post\\_17.html](http://lyceum-jl.blogspot.com/2006/05/blog-post_17.html)>

differences. This is why it is very crucial to have a visualization of such datasets to easily understand the differences between them.

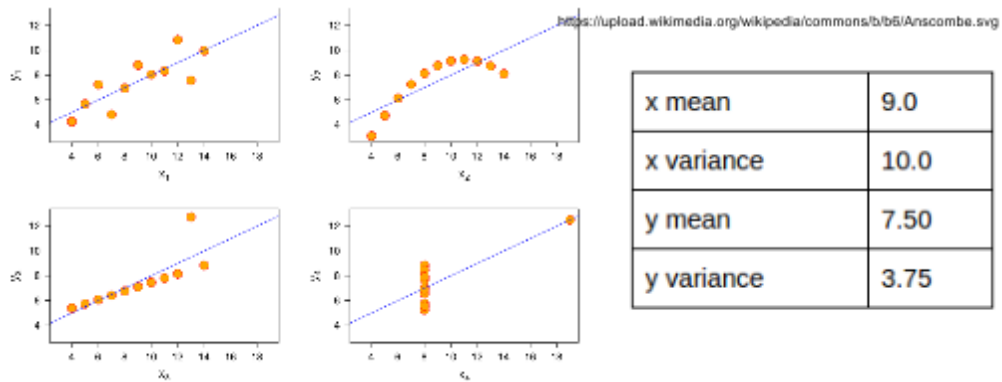


Figure 3.1: Need for Data Visualization

### 3.3.1 Attributes of Visualization

Computer-based visualization systems need to provide *visual representations* of datasets intended to help people carry out some *task* more *effectively*.

The workshop on Visualizing Biological Data (VIZBI 2011) specifies that an effective visualization should have the following attributes.

1. A human that requires details from the visualization. There is no need for a visualization if no one is going to see it.
2. Representation of Data: Perception versus Cognition. Perception deals with how easily a visualization can be organized and processed by our sensory organs, especially eyes. Cognition is more about how the perceived data can generate a thought process in the human brain. This attribute is explained in further detail by Borkin et al., (2016).
3. An intended task. Visualization should avoid catering to multiple tasks at once.
4. Measurable definitions of effectiveness. For a particular type of dataset and a need, some categories of visualization would perform better than the others. There will also be some tradeoffs.

### 3.3.2 Principles of Visualization

To create an effective visualization, certain aspects also needs to give special attention like,

- Visual channel types and ranks
- Categorical color constraints
- power of the plane
- danger of depth
- resolution beats immersion
- eyes beats memory
- validate against the right threat

### 3.3.2.1 Data Types

According to Marschner, S., & Shirley, P. (2015, chap. 27.2), the core aspect of visualization design is driven by the type of data it needs to address. Is it a table of numbers, or a set of relations or is it a collection of data points from different geographical locations?

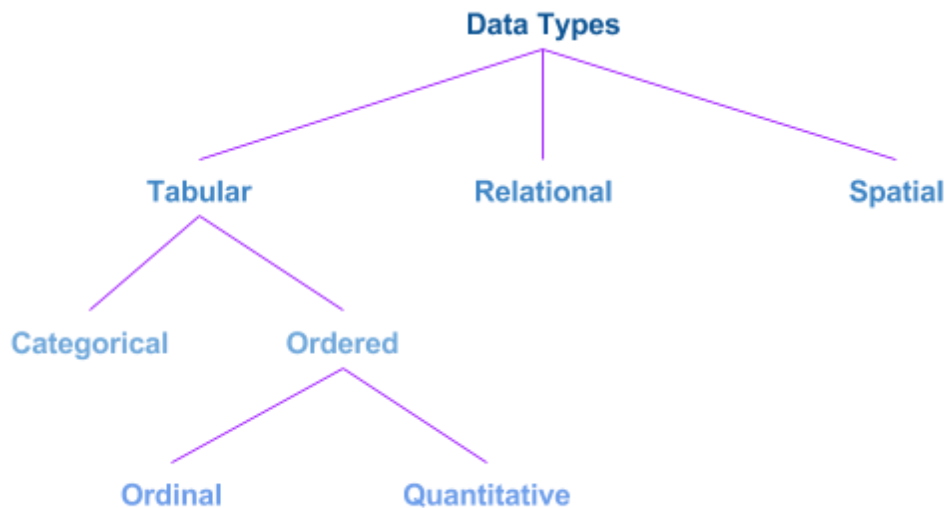


Figure 3.2 : Data Types

Let us start by considering a table representing a class of students, where the rows (items) represent different students and columns (dimensions) contains attributes such as name, age, student number, height, gender and shirt size. This particular table describes three different kinds of data types. *Quantitative* data type such as age and height, that are numerical and on which arithmetic calculations can be done. Secondly, an *Ordinal* data type field like shirt sizes like Small, Medium and Large. Arithmetic calculations cannot be performed on this type of data, but surely they have a well defined ordering. *Quantitative* and *Ordinal* data types can be generalised into *Ordered* data types. Finally the third type of data is *Categorical* data like gender. They do not have an implicit ordering, but they can be distinguished between types, such as *Male* and *Female*. Aforementioned *Categorical* and *Ordered* data types can be generalized into one super category which is *Tabular* data types.

Another super category of data type is *Relational* data. This type of data consists of a *graphs* or *networks*. A graph contains several *nodes* that are connected to each other by *links* (*edges*). A link between two nodes usually represent a relation between them. Both nodes and edges can have their own set of attributes. *Tree* is a specific kind of graph, it is typically used to represent hierarchical data.

Finally we have *Spatial* data type. It contains geographical locations and fields of measurements whose positions are in three-dimensional space such as MRI or CT scans used for medical purposes. Spatial data-type is one of the most difficult

problems of visualization design and falls into the category of *Volume Visualization* (VolVis).

The above mentioned data types can be summarised into a chart illustrated in figure 3.2.

### Dimension and Item Count

One of the most fundamental aspect of data visualization is the number of dimensions that need to be visually encoded. Visualization techniques that work for a low-dimensional dataset with fewer columns will often fail with very high-dimensional datasets with hundreds of columns. Some data dimensions might even have hierarchical patterns within them such as a time-series dataset which might contain a year, month, day, hours, minutes, seconds and even milliseconds and microseconds.

The number of items also needs to be taken into consideration. A bar chart can work effectively for a few hundred items, but will fail to deliver any results for millions of items. There are several reasons for it to happen such as computation and rendering takes too long; in others it is even a deeper perception problem where the visual clutter makes the representation impossible to cognitively analyse, making the visualization useless as explained by Borkin et al., (2013).

### Derived Dimensions

Data can also be derived from one type into another as a part of solving the domain problem. A dataset consisting of temperatures in Celsius, which is a quantitative data type, can be derived into an ordinal data type like cold, warm or hot.

#### 3.3.2.2 Visual Encoding

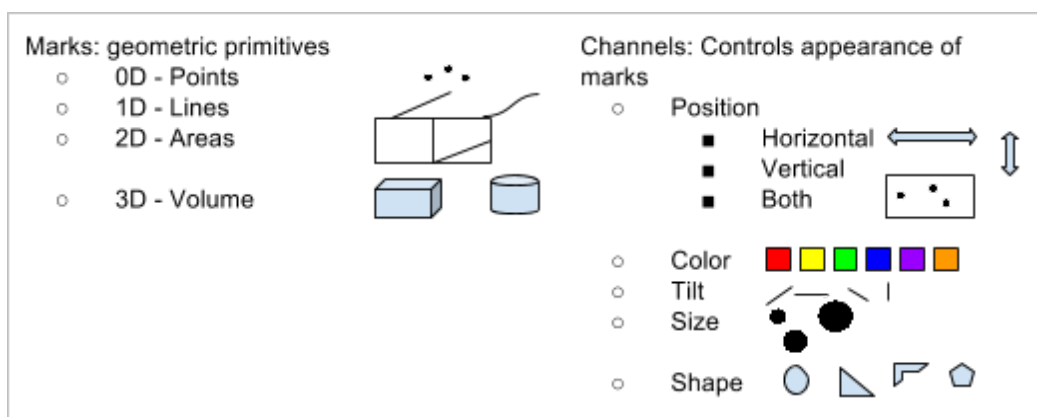


Figure 3.3: Marks and Channels

Visual Encoding consists of graphical elements such as *marks* and *channels* as described by Jacques Bertin, a French cartographer in his paper *Semiology of*

Graphics (1974). Marks use visual channels to convey information. Figure 3.3 indicates different types of marks and channels.

A zero-dimensional mark is a point, a one dimensional mark is a line, a two-dimensional mark is an area and finally a three-dimensional mark is a volume. Visual channels such as spatial position, color, texture, size, shape, orientation and direction encode information. Multiple visual channels can be used on mark to simultaneously encode different data dimensions.

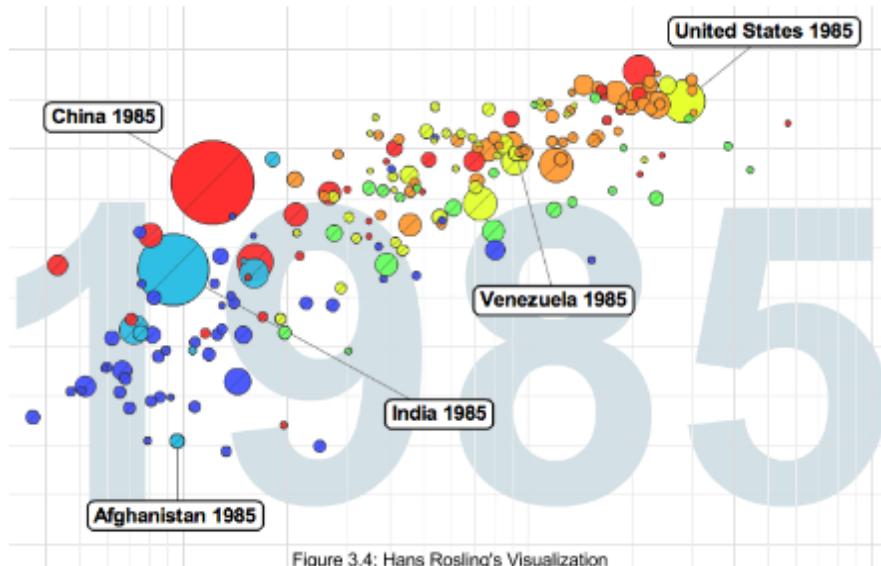


Figure 3.4 illustrates Hans Rosling's Visualization which became very popular after his Ted Talk - The best stats you have ever seen.<sup>17</sup> It uses a single mark which is a point with a combination of several channels like horizontal and spatial position, color, and size to display GDP per capita, CO2 emission levels, geographical regions, and population respectively. He also used to background as an additional channel to convey the year.

### Visual Channel Characteristics

Whenever a design decision needs to be taken in selecting marks and channels, the most important characteristics that needs to be considered are distinguishability, separability and pop out. All channels are not equally discernable, so many psychological studies have been carried out to measure the ability of people to make precise distinctions about information encoded by different visual channels. These abilities depend on whether the data type is quantitative, ordered, or categorical. Mackinlay (1986) ranked the priority of these visual channels for the three data types and is shown in the figure 3.5. Theses rankings are based on parameters like *Accuracy*, *discriminability*, *separability* and *popout*.

<sup>17</sup> "Hans Rosling: Global population growth, box by box | TED Talk | TED ..." 2014. 18 Aug. 2016 <[https://www.ted.com/talks/hans\\_rosling\\_on\\_global\\_population\\_growth](https://www.ted.com/talks/hans_rosling_on_global_population_growth)>

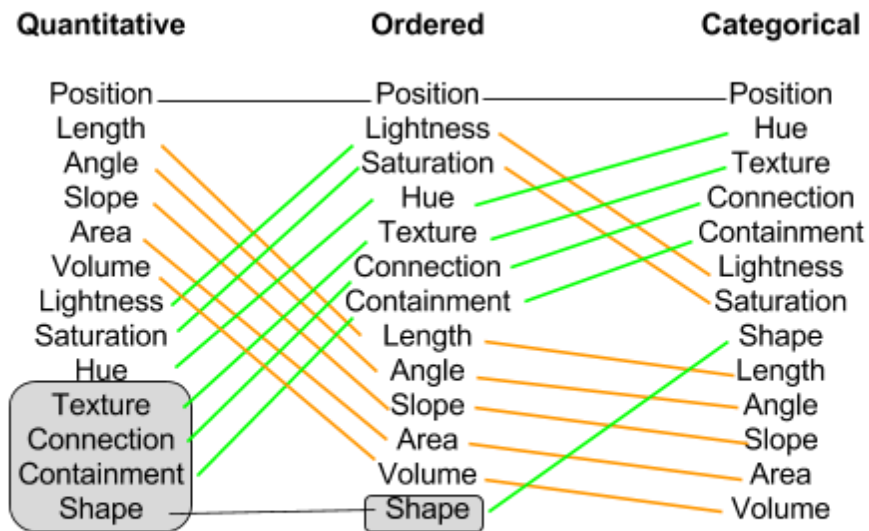


Figure 3.5: Our abilities to perceive information encoded by a visual channel depends on the type of data used, from the most accurate at the top to the least at the bottom. The ones in gray boxes are not relevant to that type of data. Redrawn and adapted from (Mackinlay, 1986)

As seen from the figure 3.5, spatial position is the most precise visual channel for all the three types of data and it dominates the human perception of visual encoding. Therefore, the most important data dimensions should be mapped to horizontal and vertical spatial positions. However, perception of other channels differ strongly between types. Length and Angle provide a better cognition over quantitative data, but provides poor cognition for ordered and categorical data types. An effective rule to follow is to encode the most important attributes with the highest ranked channels.

### Visual channel rankings

#### Accuracy

Steven's Power law (2014) proposes the relationship between the intensity of a particular channel against the perceived sensation by humans. As seen from the figure 3.6, Electric Current is the strongest. A small change in the intensity of Electric Current can be easily perceived by a human. Though electric current is not used for the purposes of visualization, the next contender which is Color Saturation is the strongest.

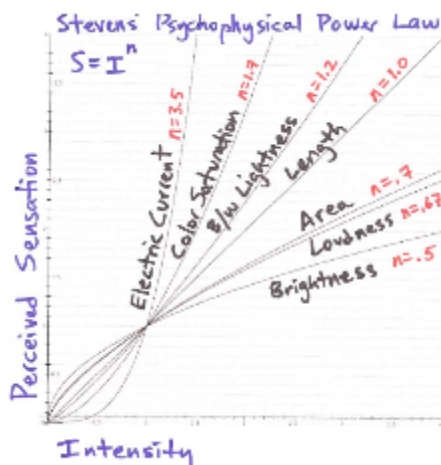


Figure 3.6: Steven's Power Law

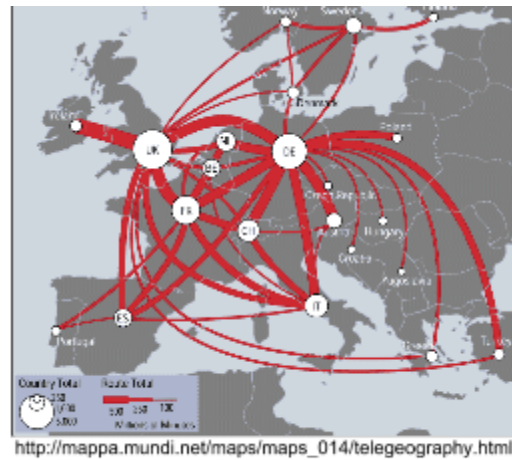


Figure 3.7: Discriminability

### Discriminability

It is crucial that the data dimensions can be easily discriminated within a channel. For example, when encoding with line width, there are a limited number of width steps that can be used. Beyond a particular width, the line will be perceived to be more of a polygon. Lines in figure 3.7 are easy to discriminate between because there are only three levels of thickness used.

### Color: Luminance, Saturation and Hue



Figure 3.8: Hue, Saturation, Luminance

Color can be divided into three sub channels, i.e. *Luminance, Saturation and hue* as shown in figure 3.8. Hue is a very strong channel to encode categorical data types, but should be completely avoided for ordered data types. Luminance or saturation should be preferred for ordered data types because they have an implicit perceptual ordering.

Quantitative data should be shown using a colormap. An example of colormap is shown in the Figure 3.9.a. Unfortunately, many software packages contain rainbow colormaps which uses hue to indicate order. That should be completely avoided and better alternative would be to use luminance. Figure 3.9.b shows the Bernice Rogowitz (1995) diagram. Both the images illustrate the elevation from the sea-level, the only difference is that the one on the left uses rainbow colormap while the one on the right uses luminance. Zero-crossings can easily be spotted from the map on the right in figure 3.9.b, rainbow colormap on the left renders it impossible to spot the difference. The rainbow colormap also creates a false impression about the structure of the surface topography and ocean depth.



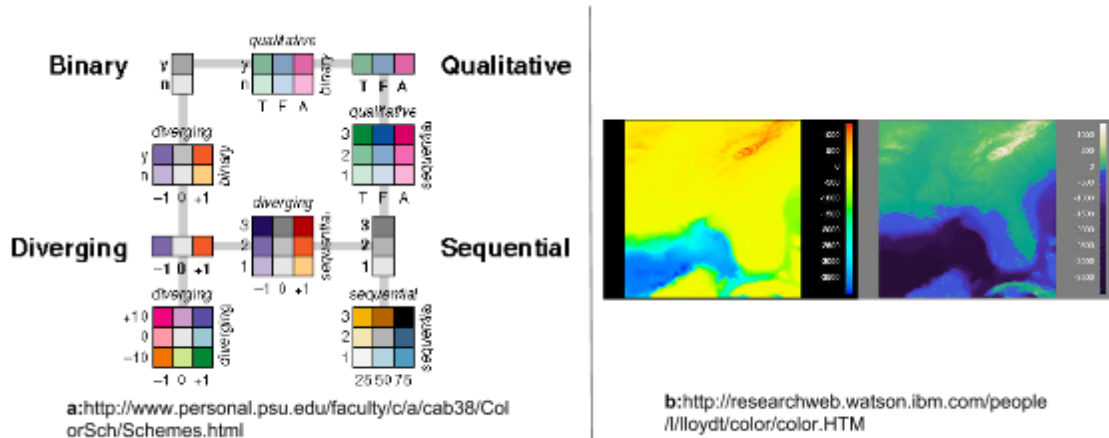


Figure 3.9: Colormaps

### Dangers of Depth: 2D vs 3D

When computer-based visualizations began in 1980s, there was a lot of enthusiasm for 3D representations. However, researchers soon began to understand the costs of 3D approaches when using abstract datasets (Ware, chp 5, 2008). Human visual system does not really see in 3D, but 2.05D.

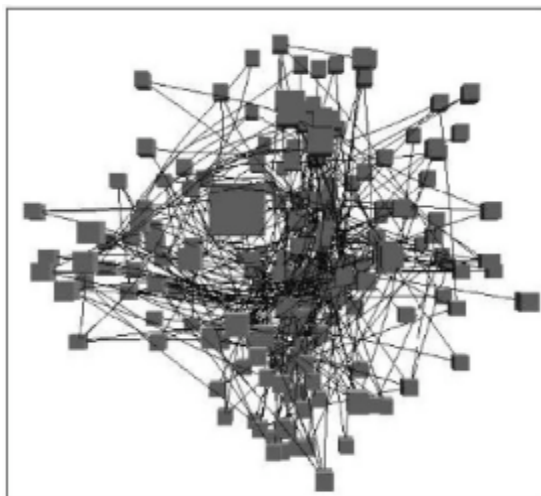


Figure 3.10: Distortion Viewing Techniques for 3D Data.  
Carpendale et al., InfoVis 1996

The prime issues involved with using 3D are *Occlusion*, *Interaction complexity* and *Perspective distortion*. As seen in figure 3.10, because of occlusion, the objects that are far behind cannot be determined. Even if interaction is provided, it will be fairly difficult to interact with such a visualization (Carpendale et al., 1996).

Real-world objects do appear smaller when they are far away, however foreshortening makes direct comparison of object difficult (Tory et al., 2006). This effect is called perspective distortion. It is easier to judge the heights of familiar objects in real world based on past experiences, however the same is not true for visual encoding of abstract data. This makes it difficult to interpret the bar heights of a 3D bar charts as opposed to 2D bar charts. Figure 3.11 shows Steve Jobs

using perspective distortion to his advantage. Apple's share of 19.5% appears to be much larger than Other's share of 21.2%.

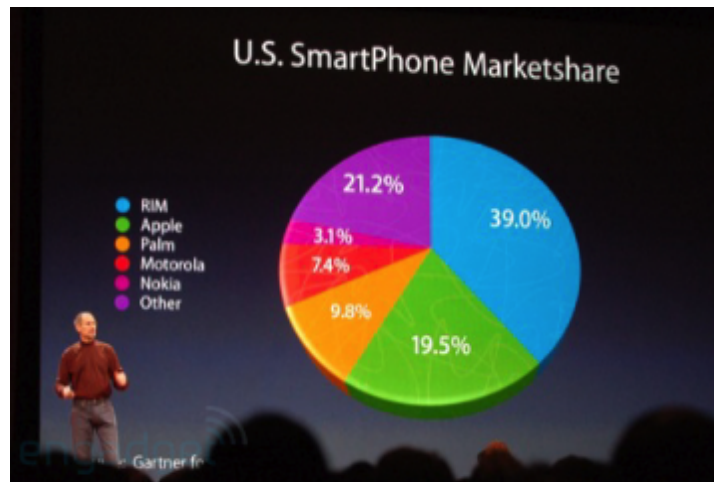


Figure 3.11:  
<https://www.engadget.com/2008/01/15/live-from-macworld-2008-steve-jobs-keynote/>

### Text Labels and Tooltips

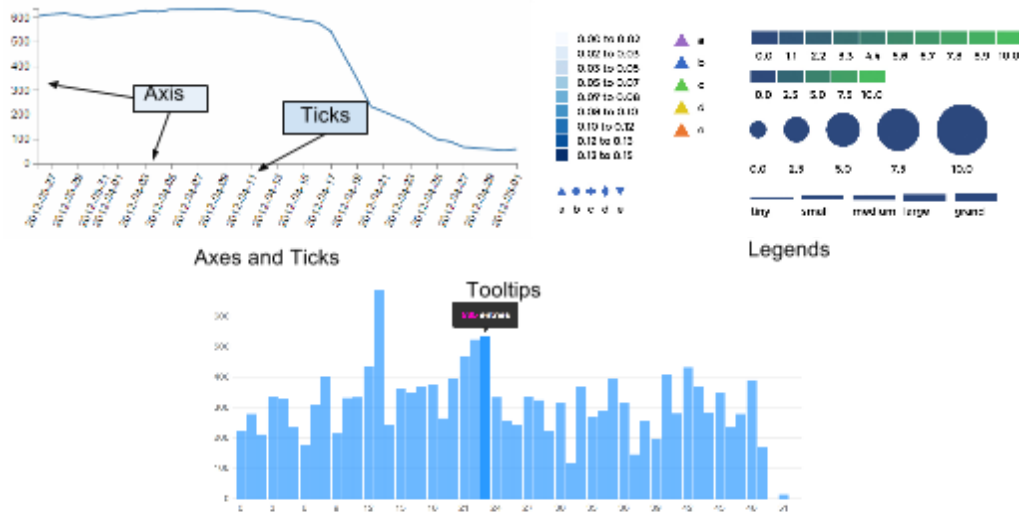


Figure 3.12: Axes, ticks, legends and tooltips

Text in the form of labels and legends is very important factor in creating visualizations that are useful rather than just being visually appealing. Axes and tick marks should be labelled and are also helpful tools. Legends should be used to indicate the meanings of colors whether it is used as discrete or in a continuous color ramps. In many cases showing all labels at all times would produce a visual clutter. In such cases interaction techniques like tooltip on hover can be used to avoid overlap (Luboschik et al., 2008).

### 3.3.3 The Visualization Pipeline

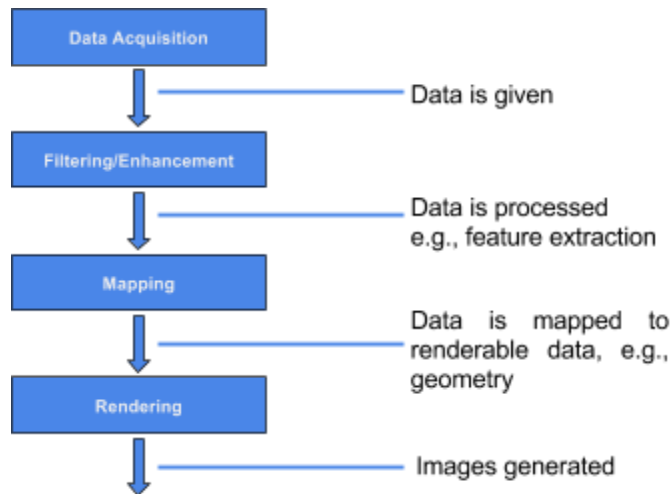


Figure 3.13: Data Visualization pipeline. Redrawn and Adapted from Nielson et al. (1990)

The *Visualization pipeline* describes the stepwise approach for creating visual representations of the data. Figure 3.13 shows the workflow of the visualization pipeline (Nielson et al., 1990).

#### 3.3.3.1 Data Acquisition

Data is prepared for visualization. e.g Measurements retrieved from CT/MRI, flow simulation (Card et al., 1999).

#### 3.3.3.2 Filtering/Enhancement

Data is filtered to select only the portions of the data that need to be visualized. Steps like applying a smoothing filter (noise suppression), interpolating missing values, or correcting erroneous measurements should be done during this step.

#### 3.3.3.3 Mapping

Filtered and focussed data is mapped to geometric primitives like points, lines, or polygons. They are also given attributes like color, position and size.

#### 3.3.3.4 Rendering

Finally the geometric data that is retrieved from the previous step of the pipeline is transformed into an image data. This may also include other sub-steps like visibility calculation, illumination, compositing and animation.

For the application described in this thesis, Backbone.js (chapter 4) will be used for implementing the first two steps of the visualization pipeline i.e. Data Acquisition and Filtering/Enhancement referred as *Application Logic*. D3.js (section 3.4) which is a JavaScript based data visualization library will handle the Mapping and Rendering steps to the visualization pipeline referred as *Chart Logic*. In my

previous projects, I have handled the entire visualization pipeline using only D3 and JavaScript. I have noticed that the application becomes quite complex after some implementations. That is why, I decided to test a different methodology by using an MV\* framework like Backbone.js along with D3.js.

### 3.4 Creating Visualizations using D3.js

For building visualizations on the Web, developers and designers have to use multiple tools simultaneously. HTML is used to create page content, CSS for design aesthetics, JavaScript for interaction and processing and SVG for vector graphics. One of the greatest success of Web as a platform is the seamless cooperation of these technologies together by shared representation of the page called the document object model. DOM exposes the hierarchical structure of the page and enables referencing and manipulations (Amr, Tarek el al., 2016).

D3.js is a JavaScript library that exploits all the benefits provided by the DOM to bring data to life using HTML, CSS and SVG. D3 manages complexities of Web standards and provides full capabilities to modern browsers by combining powerful visualization components along with a data-driven approach <sup>18</sup>.

There are several other visualization libraries that provide ready-made solutions to many of the commonly used charts and graphs. Some of popular libraries are Highcharts, C3, nvd3, and Plotly.js. All these libraries are amazing and create beautiful visualizations. They are comparatively easier to implement than D3, because all that is required is to call their API, specify the type of chart and provide data in the required format. However, they have limited customization and interactivity. Moreover, implementing customized interactions between different visualizations cannot be accomplished. Libraries like C3 and nvd3 are themselves developed using D3 at its core. Therefore, I chose D3 as it imposes relatively lower restrictions in terms of customizations and provides more freedom.

This thesis will focus more about some chart implementations of D3. To learn more about working of D3 visit <http://d3js.org> and refer to Maclean, 2014.

#### 3.4.1 Chart Object

Every JavaScript object has a prototype which is also a JavaScript object. All JavaScript objects inherit their methods and properties from their prototypes. Therefore, I created a Chart object that defines all the attributes and methods required to create different types of Chart Visualization using D3. For every chart that needs to be created, the *new* keyword is used to create a new instance of the Chart Object. The following lines of codes create the Chart Object prototype. This also creates the constructor method that is invoked whenever a new instance of the Chart Object is initialized like `var obj = new Chart();`

```
var Chart = function (e1Id) {
```

---

<sup>18</sup> "D3.js - Data-Driven Documents." 2015. 24 Aug. 2016 <<https://d3js.org/>>

```

    this.margin = {top: 10, right: 10, bottom: 30, left: 30};
    this.width = this.height = 0;
    this.svg = null;
    this.elId = elId; // HTML element Id where chart needs to be drawn
};

```

Then the JavaScript's prototype property is used to define new methods to the Chart Object to extend its functionality. The Chart Object defines of the following methods.

```

Chart.prototype.createSVG = function(width, height, margin){
    // Defines a new SVG element for charts to be drawn into
};

Chart.prototype.appendAxis = function(options){
    // Generic method used to draw Axis for Bar, Area and Line charts
};

Chart.prototype.renderLegend = function(options){
    //Generic method for creating legends for charts
};

Chart.prototype.bar = function (data, xLabel, yLabel, numeric) {
    // Draws a Bar chart
};

Chart.prototype.groupedBar = function (data, xLabel, yLabel) {
    // Draws a Grouped Bar chart
};

Chart.prototype.pie = function (data) {
    // Draws a Pie chart
};

Chart.prototype.line = function (data, xLabel, yLabel, time_param) {
    // Draws a Line chart
    // The time_param specifies type of time-series used
};

Chart.prototype.area = function (data, xLabel, yLabel, time_param) {
    // Draws an Area chart
    // The time_param specifies type of time-series used
};

```

### 3.4.2 Scalable Vector Graphics (SVG)

SVG is a language for describing 2D-graphics and graphical applications in XML which is then rendered using the SVG viewer. Most modern browsers like Google Chrome, Mozilla Firefox, Microsoft Edge, Safari and Opera support SVG and can display them as an images just like PNG, GIF, and JPG.

SVG is required by D3, and the following code snippet appends a <svg> tag into the DOM.

```
Chart.prototype.createSVG = function(width, height, margin){
  return this.svg = d3.select(this.elId).append("svg")
    .attr({
      width: width + margin.left + margin.right, // Width of SVG
      x: "0px", // horizontal Position
      y: "0px", // vertical Position
      height: height + margin.top + margin.bottom, // Height of SVG
      xmlns: "http://www.w3.org/2000/svg", // XML namespace
      version: "1.1"
    })
    .style("font-size", "8px");
};
```

### 3.4.3 D3-Scales

Every visualization dataset has values within a domain and this can vary dramatically between different datasets. A dataset specifying number of students studying in a school might be within thousands, while another containing population of a nation might be in millions or even billions. Though the domains can vary drastically, one thing remains constant; the number of pixels available on the screen. These different domains needs to be mapped onto this output range. If the dataset always remains consistent, then it can be hard-coded, but for most practical cases the data changes constantly. This issue is effectively handled by D3-Scales property that maps the input domain to the output range. Once D3 scale function is defined by providing it with input domain and output range of pixels, the scale function can be called by passing the input value and it returns a scaled output value (Murray, chp 7, 2013).

D3 provides different types of scales such as linear, ordinal, logarithmic, square root and so on for different datasets. I will be discussing a simple example of linear scale.

Let's consider a dataset of number of pizzas sold in a store in 6 days that needs to be mapped into an SVG of maximum height of 300px.

```
dataset = [170, 30, 245, 173, 486, 395];
```

A scale's input domain is a range of possible input data values, in this case it might be from 0 to 500. A scale's output range is the range of possible output values in pixels. Out of 300px available, let's keep 100px for margin and labelling, so now only 200px are available. The output range will therefore be from 0 to 200 pixels. This is illustrated in figure 3.14.

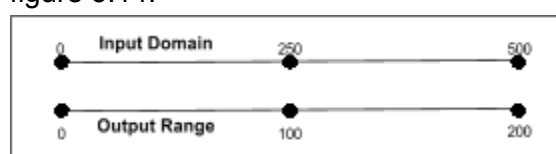


Figure 3.14: D3-Scales

This scale can now be defined in D3 using the following code snippet.

```
var scale = d3.scale.linear() // Specify type of scale
    .domain([0, 500]) // Input domain
    .range([0, 200]); // Output Range

scale(100); // Returns 40
scale(345); // Returns 138
scale(422); // Returns 168.799999
```

### 3.4.4 D3-Axes

It is similar to scales, the only difference being that D3-Axes generates visual elements like lines, labels and ticks. Axis also needs to be provided with a scale to operate on (Murray, chp 8, 2013).

The following code snippet defines an axis using the scale defined in section 5.3.

```
var yAxis = d3.svg.axis()
    .scale(scale)
    .orient('left')
    .ticks(5);
```

The orient attribute specifies where the labels needs to appear relative to the axis itself. Possible orientations for the labels are top and bottom for horizontal axis, and left and right for vertical axis. The orient attribute needs to be defined especially when axis needs to be vertical, else the default value is bottom and the axis will be drawn horizontally. Another attribute is *ticks*, which specifies to d3 exactly how many ticks should be shown on the axis. However, the default value is determined by D3 by making informed judgements based on the defined scale which in most cases is ideal.

The above code snippet will not generate a visual element on itself. The axis needs to be called.

```
svg.append("g")
    .attr("class", "axis")
    .attr("transform", "translate(40,20)")
    .call(yAxis);
```

The transform attribute specifies the coordinates of the axis where it should be placed within the SVG. This will generate an axis shown in figure 3.15.

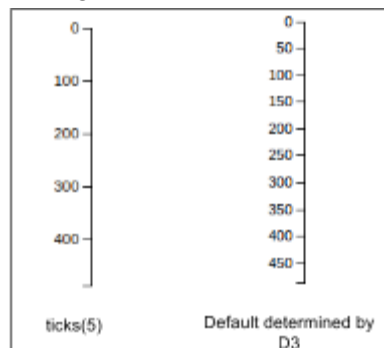


Figure 3.15

### 3.4.5 Legends and Tooltips

As described in section 3.3.2.2, legends are good indicators to summarize the use of different visual channels used in the visualization. One issue while creating the legends was that for several cases where the text labels had longer strings of characters, the legend box interfered with the visualization. Therefore a hide and show functionality was added for the legend as shown in the figure 3.16.

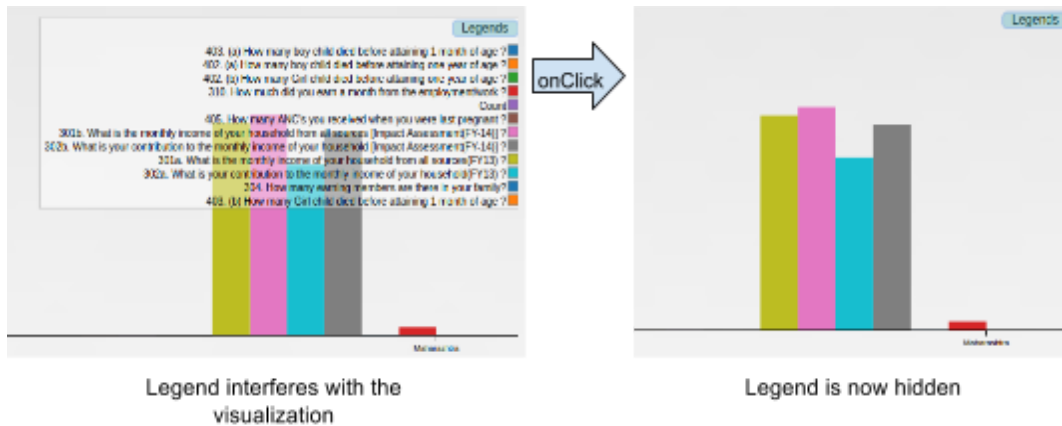


Figure 3.16: Legends

For now all charts use the same type of legend, and therefore I have generalized the creation of legends in a single method which is called by different charts. The code used to create the legend box is mentioned in Appendix A.

### 3.4.6 Bar Charts

Bar charts are good indicators for direct comparison of categorical as well as ordered data types. To use appropriate visual channel rankings as described in section 3.3.2.2, categorical datasets should use different hues, while ordered datasets like ordinal and quantitative should use color saturation and luminance respectively. Bar charts should be used when a categorical or ordered datasets needs to be mapped against a quantitative dataset. The following figure 3.17 shows an example of a Bar chart using categorical data. This chart as well as all the corresponding charts are built using data from Poimapper application.

As seen in the following figure 3.17.a, color coding is done using different hues for each category. Well on the other hand figure 3.17.b, is using quantitative data and therefore the difference can be easily compared using color saturation. Also it is a good practice to sort bar charts of ordered datasets. These bar charts are drawn using a combination of d3 modules like d3-scales, d3-axes, d3-colors, d3-selections, d3-Collections and d3-shapes<sup>19</sup>. Code for creating bar chart is available in Appendix A.

<sup>19</sup> "d3/API.md at master · d3/d3 · GitHub." 2016. 30 Aug. 2016  
<<https://github.com/d3/d3/blob/master/API.md>>



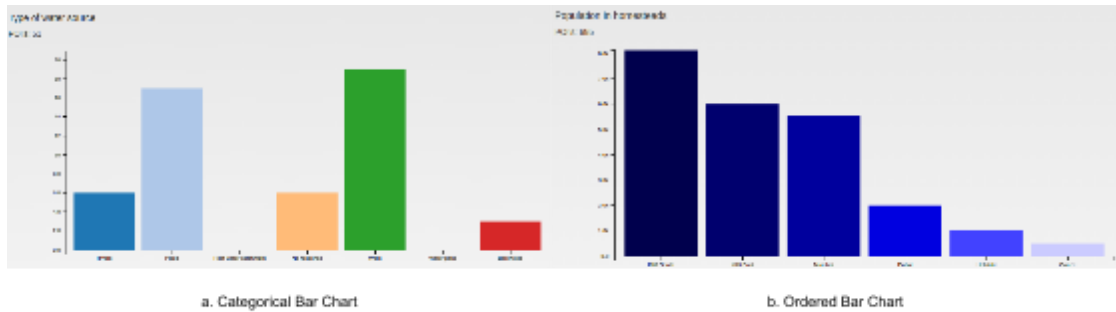


Figure 3.17: Bar Charts

### 3.4.6.1 Grouped Bar Charts

Grouped bar charts are used to show information about different sub-groups of the main categories. Figure 3.18 shows an example of grouped bar chart where all the numerical questions of a Poimapper form are grouped based on regions. Grouped bar charts require an additional legend as shown in the top right box. This legend provides information about the colors used by different questions while the X-axis displays the regions. Refer to Appendix A for the code to create Grouped Bar chart.

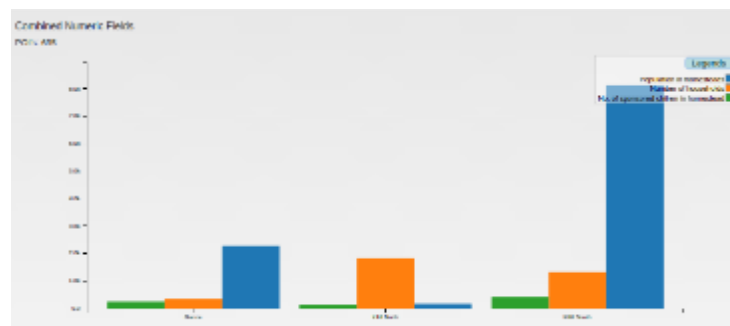


Figure 3.18: Grouped Bar Charts

### 3.4.6.2 Limitations of Bar Charts

Bar charts have several limitations listed as follows.

1. If there are too many domain values to be rendered as bars, the text in x-axis becomes difficult to read due to smudging.
2. Smudging of text labels can also occur if they have more characters as seen in figure 3.19.
3. Colors in categorical datasets loses their perceptive value once there are more than 7-8 different categories in a domain which are rendered with different colors.
4. Bars with extremely high values will make bars with much smaller values look invisible. This happens when one value is in millions while another just in hundreds. The bar showing a value in hundred will be invisible in the chart. D3 does provide log scales which can solve the problem, but then the bar itself will lose its cognitive strength. In figure 3.19, the second bar cannot be seen as it has a value in hundreds while the tallest bar has a value of 240,000.

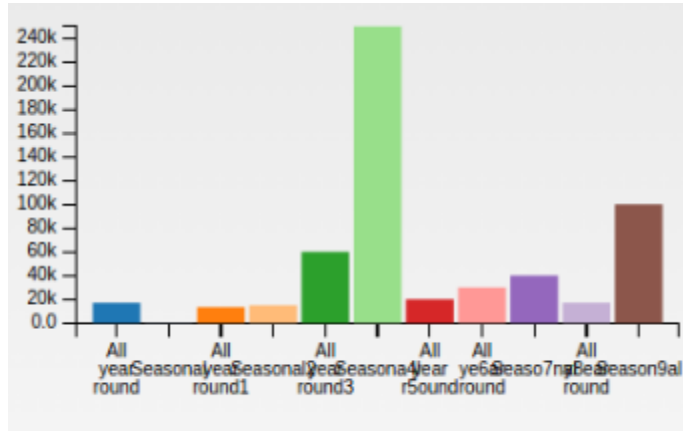


Figure 3.19: Limitations

### 3.4.6.3 Overcoming Limitations

One way to overcome these limitations is to make sure data is appropriately filtered so that such conditions do not happen. Such type of data filtering can be handled by Backbone and that will be discussed in later chapters. Another way to handle these limitations is by using a very efficient module provided by D3. D3-Brush is a module that can be used by the user to filter out appropriate sections of the data after the chart has been rendered. Figure 3.20 shows a grouped bar chart.

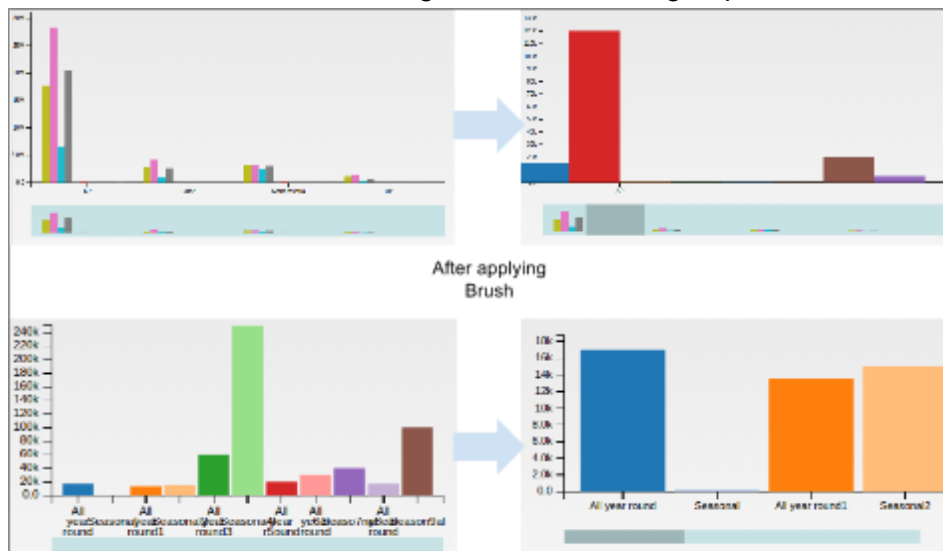


Figure 3.20: Brushing and Zooming

There are several bars that cannot be seen in between every group because the pink bar has a domain value in 30 million. There is another rectangle below the chart in bluish color. That is a brush box. After applying the brush, the values that could not be seen before can now be zoomed in and seen. The scaling of X and Y axis should change as well reflecting the brush selection.

### 3.4.7 Pie Charts

Pie charts is a circular statistical graphic, which is divided into slices to illustrate numerical proportion. The arc length or area or central angle of each slice in a pie chart is proportional to the quantity it represents (Spence, 2005). It is good choice only for categorical datasets where there are fewer than 7-8 categories. It provides an efficient cognitive value if only a couple of these categories occupy large area in the circle. Figure 3.21 shows an example of a pie chart.

There are a few variants of pie chart such as 3D pie chart, Doughnut chart (Harris, Robert L, 1999), exploded pie chart, Polar diagram and Ring Chart. They all rely on the concept of central angle value being proportional to the quantity. Refer to Appendix A for the code to generate a Pie chart.

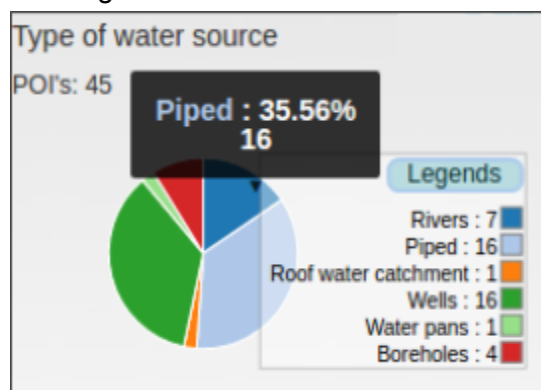


Figure 3.21: Pie Chart

#### 3.4.7.1 Limitations of Pie Chart

One obvious limitation of Pie chart is that they cannot show more than a few values without having a clutter. Also when there are a lot of values, it becomes difficult to differentiate colors as well as labelling them becomes a problem. Furthermore, if few values are exponentially smaller than others, they cannot be represented.

There are not many ways to overcome the limitations of Pie charts. Filtering of the data is quintessential to provide good cognitive value. To avoid cluttering of labels, tooltips can be used as explained in section 3.3.2.2.

### 3.4.8 Line Charts

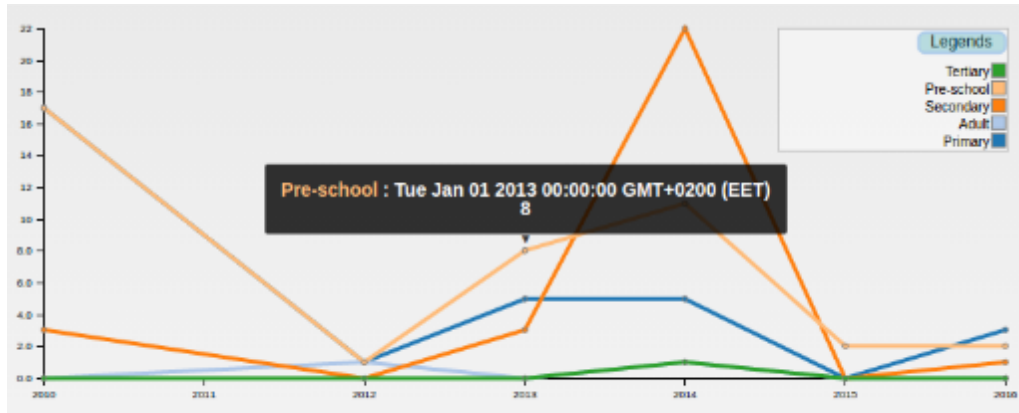


Figure 3.22: Line Chart

The primary use of Line charts is to show trends over a period of time such as stock prices over a period of 2 years. The time units such as years, quarters, months, weeks, hours and so on is distributed evenly across the horizontal axis. The magnitude of each data element in the series is represented by its position on the vertical axis. Multiple series can be represented in a single line chart and this makes it easy to compare trends. Refer to Appendix A for the code to generate a Line chart.

### 3.4.9 Area Charts

Area charts are very similar to line charts. They both require the same type of dataset i.e. quantitative vs quantitative (continuous and ordered like a time-series). The only difference is that the area below the plotted line is filled with color as shown in figure 3.23. Refer to Appendix A for the code to generate an Area chart.

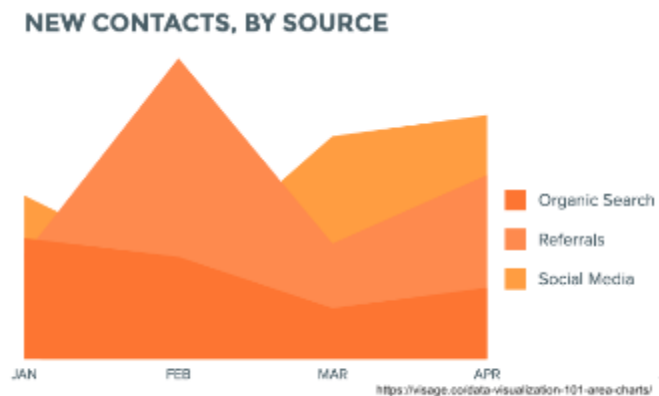


Figure 3.23: Area Chart

#### 3.4.9.1 Limitations of Area Charts

Area charts suffers from problem of occlusion. Figure 3.24.a shows one such example that suffers from significant interference in visual channel encoding. The

problem of occlusion can be minimized using transparency in different layers as shown in figure 3.24.b, but it still persists and interferes in visual channel encoding.

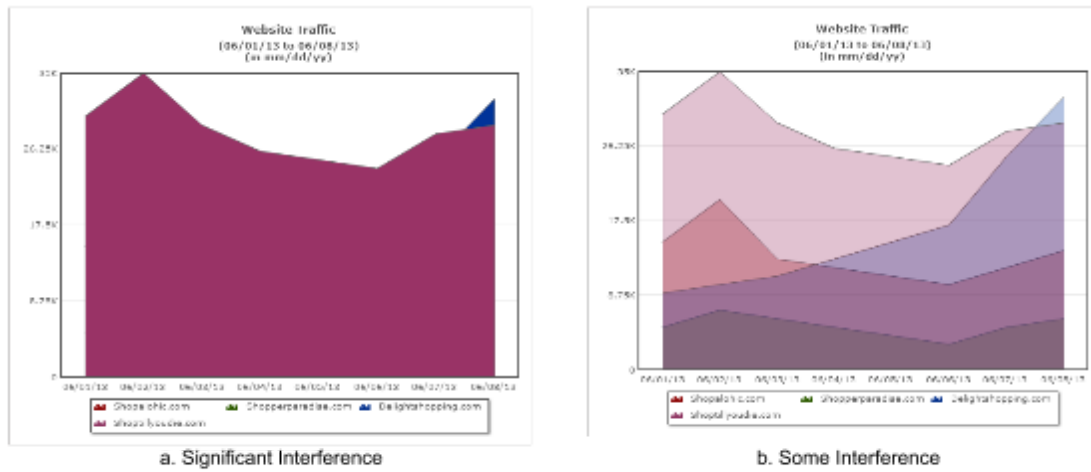


Figure 3.24: Limitations of Area Chart

One solution to overcome this limitation is to represent area plotted under the line as a summation of all the values as shown in the figure 3.25. This eliminates the occlusion problem.

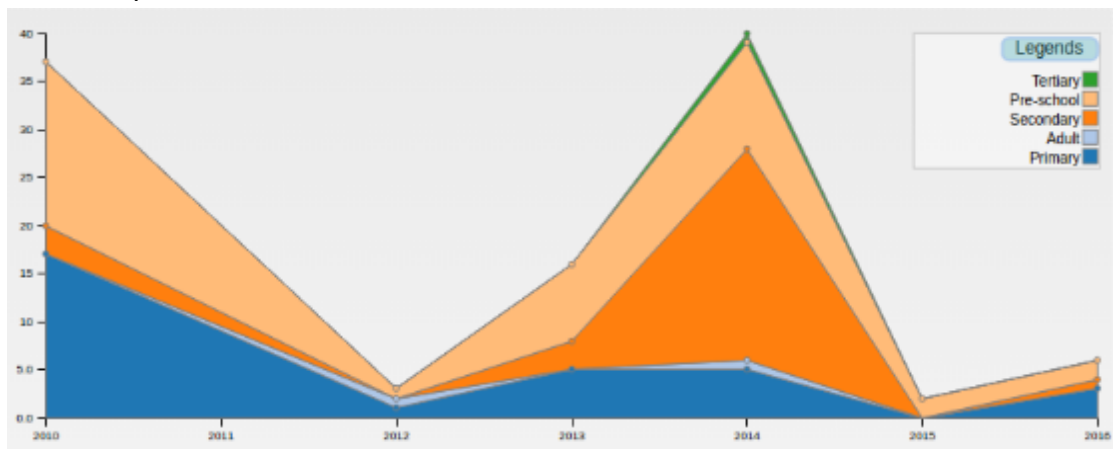


Figure 3.25: Stacked Bar Chart

## Chapter 4

# Backbone.js - An MV\* Framework

Backbone.js is a MV\* framework, and the next section will elaborate this framework.

### 4.1 Need for MV\* Frameworks

In recent years JavaScript has become one of the widely used language especially in context to Web programming. It is the most popular language on Github (see Figure 4.1). JavaScript enables programmers to implement logic to dynamically manipulate HTML DOM elements efficiently. Developing simple Web applications by using JavaScript is easy and along with jQuery<sup>20</sup>, HTML document traversal, handling events, animation, DOM manipulation and Ajax applications become even more versatile.

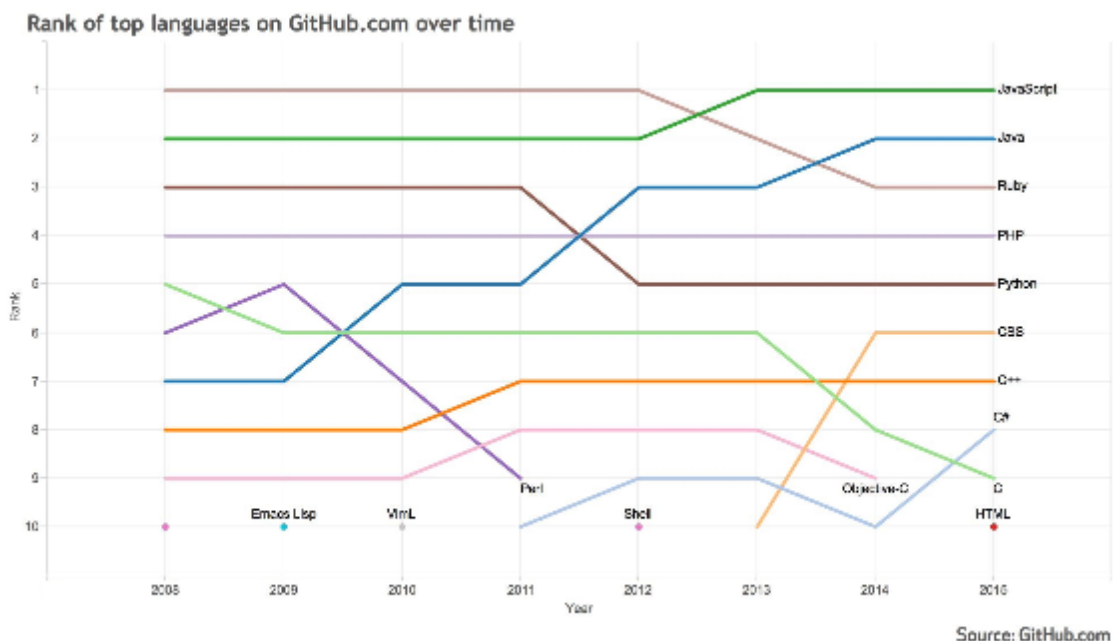


Figure 4.1: Top Languages on Github

However, managing code becomes trickier as the complexity of the Web application increases. Most JavaScript applications end up becoming entangled

<sup>20</sup> "jQuery." 2006. 12 May. 2016 <<https://jquery.com/>>

piles of jQuery selectors and callbacks as everything tries to keep data in sync between the HTML DOM, client-side logic and server-side data. Therefore, it is crucial to adapt to a more structural approach to create rich client-side applications and make the code manageable and easily adaptable. This is where MV\* frameworks such as Backbone.js,<sup>21</sup> Angular.js,<sup>22</sup> Ember.js,<sup>23</sup> and Knockout.js play an important role.<sup>24</sup> All of these frameworks have some advantages and disadvantages over each other. All of these frameworks can get the job done, although choosing the most suitable framework depending on the project requirements would be helpful in the long run. Shaked (Uri. "Angular vs. Backbone vs. Ember." URL: <https://www.airpair.com/js/javascript-framework-comparison>), provides a detailed comparison between some of these frameworks and is a recommended read.

## 4.2 Model-View-Controller

Model-View-Controller (MVC/MV\*) is a software architectural pattern for implementing user interfaces on computers (Coplien, 2014). The software application is divided into three (or in some cases it can be fewer or more) interconnected parts, to make projects manageable through different layers of abstraction and separation of concerns.<sup>25, 26</sup> This also helps to create a unified structure between different projects.

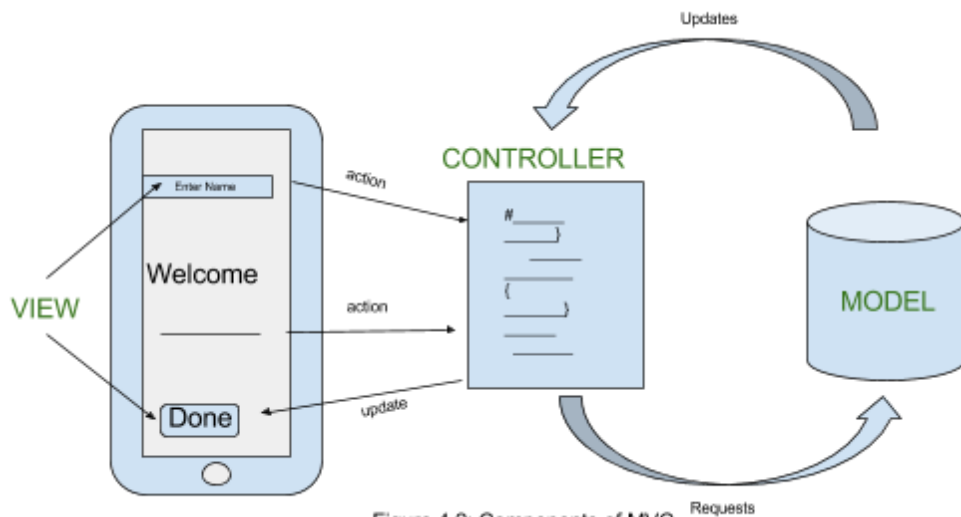


Figure 4.2: Components of MVC

<sup>21</sup> "Backbone.js." 2011. 12 May. 2016 <<http://backbonejs.org/>>

<sup>22</sup> "AngularJS — Superheroic JavaScript MVW Framework." 2014. 12 May. 2016 <<https://angularjs.org/>>

<sup>23</sup> "Ember.js - A framework for creating ambitious web applications." 2011. 12 May. 2016 <<http://emberjs.com/>>

<sup>24</sup> "Knockout : Home." 2010. 12 May. 2016 <<http://knockoutjs.com/>>

<sup>25</sup> "The DCI Architecture: A New Vision of Object-Oriented Programming." 2009. 14 May. 2016 <[http://www.artima.com/articles/dci\\_vision.html](http://www.artima.com/articles/dci_vision.html)>

<sup>26</sup> "08-MVC.pptx." 2014. 14 May. 2016 <<http://www.cs.arizona.edu/~mercero/Presentations/OOPD/08-MVC.pptx>>

### 4.2.1 Model

Model stores the data required by the application to run and defines the application. It can be perceived as the main actors/classes in the application such as person, or an invoice, or a student.<sup>27</sup> It is a similar concept to an SQL table row or a JAVA class object. A Model contains the data of the application and depending on the framework being used, it can notify the View or the Controller when their state changes.

The roles played by Model are as follows,

1. It validates the attributes of the Data.
2. If certain attributes are changed during the process of the application, it notifies the backend or database or the local storage to persist the model to the same state.
3. View or controller can subscribe to certain Model or its attributes to reflect changes.
4. Models can be grouped into a *Collection* in Backbone. Collection can be perceived to be synonymous with an SQL table or a JAVA ArrayList of objects. Collections can replicate *JOINS* like in SQL between different models.

### 4.2.2 Controller

Controller acts as a mediator between the Model and the View components. It receives the user input from the view and notifies the model, on the contrary, it receives an update from the Model, and notifies the corresponding View. Many frameworks like Django ( a Python framework) or Backbone.js have included the concept of a Controller in the View itself, and therefore it is not a completely separated module. It contains the controlling logic of the application, such as deciding which Views need to be updated when the state of the Model changes.

The roles played by Controller are as follows,

1. Notify the Model about View change and vice versa.
2. Controls the data flow between the View and the Models.
3. Implement restrictions and access controls if needed
4. Can perform operations of a Router (explained in the following sections).

### 4.2.3 Views

View module is responsible to present the data contained in the model to the user and manage user interactions.

The roles played by Views are as follows,

---

<sup>27</sup> Patterns, D. "lecture 10 handout." 2010. <<http://s290179663.websitehome.co.uk/blog/wp-content/uploads/2010/12/lecture-10.pdf>>



1. Displays an interface to the user.
2. Is responsible for a specific functionality of the application. For example, a LoginView manages login and logout interactions, MainView displays the central component of the app, FilterView can provide interactions for managing filters.
3. It renders the contents of the Model.
4. Updates when it notices a change in the Model.
5. Renders templates, by using JavaScript templating libraries like Underscore.js or Handlebars.js.

#### 4.2.4 Routers

This is an additional module prominently available in both frontend and backend frameworks for developing Web applications. In frameworks like SpringMVC (a JavaEE framework), it is a part of the Controller module servlet. For backend frameworks, routers are used to decide which View to call when a particular URL request is made. It also takes care of the types of request such as GET, POST, PUT, DELETE. In recent years, single page Web applications have become quite prominent. In such applications, the base url always remains the same and content updates depending on the user interactions, but users might require linkable, shareable or bookmarkable URLs for important locations of the app. Until recently, hash fragments (#page) were used to provide these permalinks. Now, by using front-end frameworks like Backbone.js, it is possible to use standard URLs (/page) for the same purpose. Backbone.Router provides methods for routing client-side pages, and connecting them to actions and events.

Also, in any typical JavaScript application, the communication between the client and the server is handled using JSON (JavaScript Object Notation) strings. These JSON strings are exchanged with the help of REST (Representational State Transfer) or SOAP (Simple Object Access Protocol) based services.

#### 4.3 More about Backbone.js

The reason Backbone.js is used for this thesis is that it is a lightweight MV\* framework and has very little boilerplate to start with. It only relies on one JavaScript library to run (i.e. Underscore.js). Backbone.js's code requires only 6KB to load compared to other available frameworks like AngularJS (46KB + many more third party libraries) and Ember (40KB). On the downside, Backbone does not provide structure but instead provides the means for the developer to create his own structure and therefore leaves many blanks for the developer to fill in, but this might be advantageous in some cases. Backbone.js is not strict MVC, but MVC-inspired or MV\*.

As mentioned before, Backbone does not have a separate Controller component, but its functionality has been dispersed between the View and Router components. The Controller functionality of bridging between the Model and the View has been

allocated into other components. Backbone.js has six components namely *Model*, *View*, *Collections*, *Events*, *Routers* and *Sync* that are built using JavaScript objects. These components give developers the possibility for customizations.

The core features of Backbone include:

- Underscore.js
- Agnostic Templating
- MV\*
- Clean HTML
- Synchronous events
- REST
- Backbone.Sync
- Extensions
- Deep Linking

The following sections will take a deeper view of Backbone.js features.

### 4.3.1 Underscore.js

Underscore.js is the only dependency required by Backbone.js and it provides additional functionality to built-in JavaScript objects without extending them. It has many useful methods that can help dealing with Collections, Arrays, Objects, Functions, and a few other utility functions. The main function of Underscore.js for Backbone is to provide templating framework. This is achieved by Underscore's `_.template` function.

`_.template(templateString, [settings])` compiles the JavaScript templates into functions that can be evaluated for rendering <sup>28</sup>. The following is a simple example of using `_.template`

```
_.templateSettings = {
  interpolate: /\{\{(.+?)\}\}/g
};
var template = _.template("<b>{{ value }}</b>");
template({value: 'My Backbone is Strong'});
Output => "<b>My Backbone is Strong</b>"
```

`_.templateSettings` changes the default delimiter from `<%= ...%>` to `{{ ...}}`, this is done because the server-side scripting language is JSP that uses the same delimiter and can create errors. Then using a the `_.template` a new template is designed, which when called will replace the field 'value' defined within `{{ }}` tags with the corresponding parameters passed.

---

<sup>28</sup> "Underscore.js." 2008. 15 Sep. 2016 <<http://underscorejs.org/>>

### 4.3.2 Agnostic Templating

Though Underscore.js is the default templating engine, it can be switched with other recommended templating libraries like Mustache.js, Haml.js and Eco. However, developers have the freedom to choose any other templating library they prefer. Backbone also allows the use of multiple templating libraries simultaneously.

### 4.3.3 MV\*

For front-end frameworks, there are several variants of the Model-View-Controller available. It depends on the concepts used by the framework and its strengths. For frameworks like Backbone, the *Controller* module is integrated in the *View* itself. This is an architectural pattern derived from MVC called Model-View-Presenter (MVP). Therefore, these front-end JavaScript frameworks are generally referred within the developer community as MV\*, where the asterisk\* is the REGEX representation of zero or more.

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

#### 4.3.3.1 Backbone.Model

Models store all the data required by a JavaScript application and also a large part of the application logic. Models take care of conversions, validations, computed properties and access control.<sup>29</sup>

For the application this thesis is targeting, the following example code is creating a Backbone Model for storing chart data.

```
ChartModel = Backbone.Model.extend {
  defaults: {
    title: null, # chart Title
    no_of_pois: 0, # Number of Data Points
    chart_type: "bar", # bar/pie/area/line etc
    time_param: "", # area/line etc
    location: null, # if charts needs to be grouped by
    locations
    chart_variables:[{"key":"All", "value":16}, {"key":"No", "value":6}, {"key":"Seasonal", "value":29}], # Default for blank
    data
    numeric: false,
  }
}
unless window.poimapper? #export the following globals
  window.poimapper = {}
```

<sup>29</sup> "Backbone.js." 2011. 3 Jun. 2016 <<http://backbonejs.org/>>

```

unless window.poimapper.models?
  window.poimapper.models = {}

window.poimapper.models.ChartModel = ChartModel

```

The `Backbone.Model.extend(properties, [classProperties])` method provides the functionality to create a customized Model class. By using *extend* instance properties (those provided by Backbone to Models) are attached to the model. Users can optionally specify classProperties of their own to be attached.

The *defaults* method is used to provide default values to attributes in case they are not specified while initializing the Model.

#### 4.3.3.2 Backbone.Collection

This stores an array of models. It provides functionality to sort, filter, aggregate and use Underscore.js methods over the models. It can also bind “change” events to notify views when any model changes. Backbone.Views can also listen for “add” and “remove” events on a Collection.

The following code snippet defines a Collection over the ChartModel defined previously.

```

ChartCollection = Backbone.Collection.extend {
  model: poimapper.models.ChartModel
}

```

#### 4.3.3.3 Backbone.Events

The backbone of Backbone.js are its events. This makes Backbone one of the most powerful framework available. Backbone supports *Synchronous* events that can be mixed with any of the other components of Backbone or custom defined objects and gives them the ability to bind and trigger custom named events. Following is an example of binding an event to a custom named object.

```

object = {}
_.extend object, Backbone.Events
object.on 'alert', (msg) ->
  alert 'Triggered ' + msg
  return
object.trigger 'alert', 'an event'

```

In the above code snippet, the custom object first extends the Backbone.Events functionality using Underscore’s `_.extend` method. Then an event name ‘alert’ is created on the object. Finally the event is triggered using the `trigger()` method extended from Backbone.Events.

There are several useful methods provided by Backbone.Events and some of them required by this thesis are as follows.

1. **on:** `object.on(event, callback, [context])`

- Binds a callback function to an object which is called when an event is fired.
2. **Trigger:** `object.trigger(event, [*args])`  
Triggers callback for a given event.
  3. **listenTo:** `object.listenTo(other, event, callback)`  
Tells an object to listen to a particular event on *other* object and fire the *callback*.
  4. **add:** `collection.bind('add', callChartTab)`  
Binds the *add* event to *collection*. Whenever, a model is added to the collection, *callChartTab* method is invoked.
  5. **remove:** `collection.bind('remove', removeChartTab)`  
Binds a *remove* event to *collection*. Whenever, a model is removed from the collection, *removeChartTab* method is invoked.

Events enable Backbone to be used also as a Publish-Subscribe Model. This gives Backbone a loosely-coupled nature that can effortlessly be scaled into larger applications (Eugster et al., 2003).

#### 4.3.3.4 Backbone.Router

The router is used for routing the applications URL's specified after the hash tags (#). This calls specific functions that in turn initiate specific views. This also enable deep linking.

#### 4.3.3.5 Backbone.View

View is the visual representation of the model or collections it has been bound to. Views give the visual representation to the data and its state. Backbone's Views can also bind to events, which then might re-render or update some other UI components depending on the type of events triggered. Views render HTML through the use of templating libraries discussed before. This thesis will be using Underscore.js.

```

ChartPanel = Backbone.View.extend {
  tagName: "div" # Div used for this view

  events: {
    "click button.close_chartPanel": "closeChartPanel"
    "click button.delete_chartPanel": "deleteTemplate",
    ....
  }
  initialize: (options) ->
    _._bindAll this, 'render', 'closeChartPanel',
    'deleteTemplate', ..
    @chartSelectionModel = options.chartSelectionModel
    @model.bind('remove', this.unrender)
    @chartCollection = new poimapper.ChartCollection()
    @listenTo(@chartSelectionModel, 'change:locations
change:fromDate change:toDate change:users', _._debounce(@selectionChange,
1))
    @chartCollection.bind 'add', @appendChart

```

```

        @model.on("change:report_title",@changeReport_Title)
render: () ->
    @$el.append @panel_template({
        "report_title": @model.get("report_title") #Assume
Report-1
        "report_id": @model.get("template_data").report_id
#Assume id-1
        "template_class": "delete_chartPanel"
    })

```

In the above code snippet, we create a View named *ChartPanel* by extending Backbone.View. Backbone's view has a set of predefined methods that can be initialized by passing options.

- 1. tagName/el:** Every view can either have a *tagName* or an *el*. *tagName* specifies the HTML tag in which all the components of the View will be rendered in. In this case, a new `<div>` tag will be created within the DOM. In our case it is the `<div>` tag. *el* can be used to specify an already existing tag in the HTML DOM. Assuming one wants to render all these elements into a `<p id="pViewId">` tag. Then the id can be specified as *el: "#pViewId"*.
- 2. events:** Events can be triggered like click on a button with class *close\_chartPanel* will call the *closeChartPanel()*.
- 3. Initialize:** This method is called upon initialization of the View. It can contain code for assigning models/collections and listening to their events.
- 4. render:** This method renders the View onto the DOM with the help of templates. In the code snippet above, when *ChartPanel.render()* is called, it will append the *@panel\_template* inside a `<div>` tag and render the whole onto the DOM. Let's say the *@panel\_template* is defined as below

```

panel_template:_.template      '<h2>{{report_title}}</h2>:
{{report_id}}' +
    '<div class="{{template_class}}"></div>'

```

The HTML rendered on the DOM will be

```

<div>
    <h2>Report-1</h2>:id-1
    <div class="delete_chartPanel"></div>
</div>

```

However, all these are generic principles, but as mentioned before, developers have all the freedom to do whatever they please. They can render templates in the *initialize()* method itself, or not use templating library at all. Backbone leaves all the choices with the developer.

#### 4.3.3.6 Backbone.sync

It is called every time Backbone attempts to read or save a model onto the server. It uses `jQuery.ajax` to make RESTful JSON request and returns a `jqXHR`. It can be

overridden to use a different persistence strategy like WebSockets, XML or Local Storage.

#### 4.3.4 Clean HTML

Backbone.js does not add its own invented tags onto the DOM. Other frameworks like Angular append custom ng-tags on several elements in the DOM, but Backbone keeps it clean. To make it even better, React.js, which is created and used by Facebook, can be used within Backbone.View. It makes rendering even more faster and manageable.

#### 4.3.5 Extensions

Backbone.js is build using the core concept of extending existing functionalities. It can easily extend and inherit community built or custom built plugins. Conversely, it is also possible to extend some of the Backbone.js functionalities into other frameworks.

## Chapter 5

# Implementing the Data Visualization Pipeline

In this chapter, an architecture will be created with Backbone.js to generate a Web Report. During our discussion about the Data Visualization pipeline (section 3.3.3), it was decided that we will try to implement the first two steps of the pipeline (i.e. Data Acquisition and Filtering) using Backbone.js, while the latter steps of Mapping and Rendering shall be accomplished using D3.js. The final structure of the required application is shown in figure 5.1.

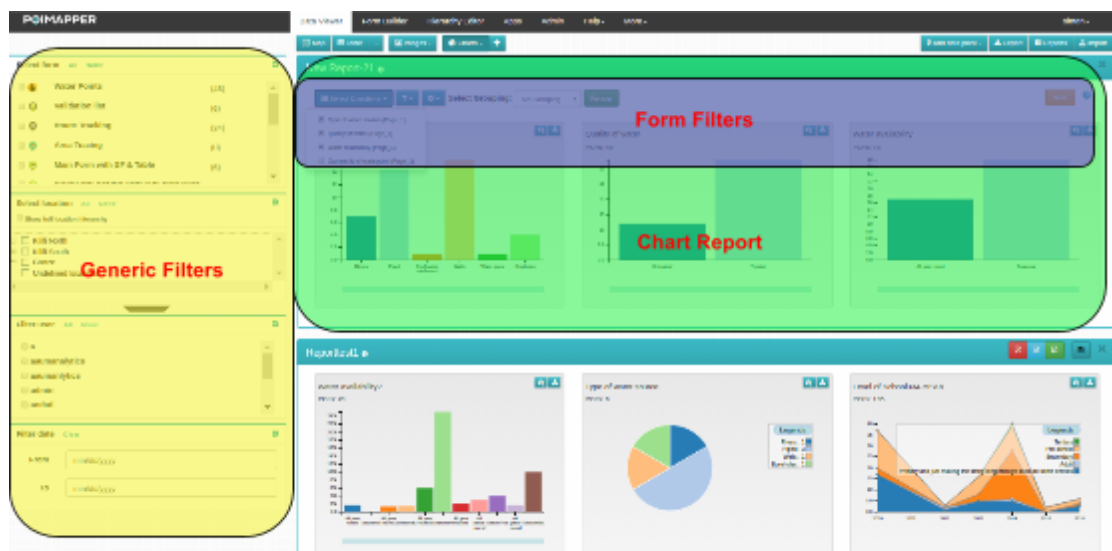


Figure 5.1: Structure of the application

### 5.1 Specifications of a Web Report

The Web Reports module of Poimapper requires to have the following functionality.

1. A Web Report should be defined on a form (refer 2.1) of Poimapper. Users should be able to select a form and the corresponding type of chart type (Bar/Pie/Area/Line) required.



2. User should then have the option to select a particular set of questions. Questions should be filtered to show only those questions whose data types create a perceptive value on the type of charts selected, e.g., Quantitative vs Time series type questions for Area and Line charts.
3. Appropriate filters are available to the user to tweak based on the type of the chart and the form selected along with some other fine tuning options.
4. When user clicks a 'Render' button, data is fetched from the server and web report is generated.
5. If external filters are changed by the user, the report should show an option to update/refresh the changes applied by the user.
6. Web Reports can be edited and saved for later use. A previously saved report can be deleted permanently
7. Report can be downloaded as a pdf, excel or word file formats. This will be achieved by connecting the Web Reports module to an already existing *Reporting* system that uses Jasper Reports to create downloadable files on the Java based backend.<sup>30</sup>
8. Multiple Web Reports should be able to be generated and loaded at once.

## 5.2 Architecture of Web Reports Module

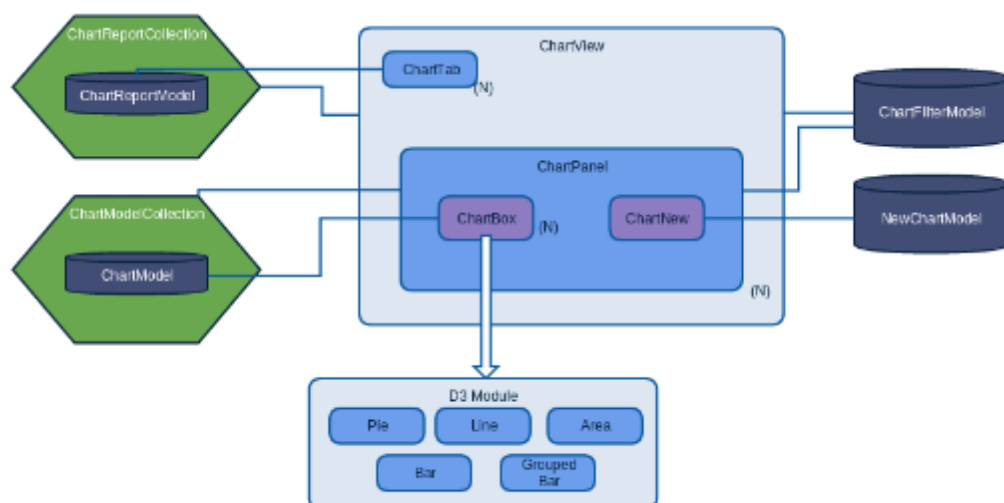


Figure 5.2: Architecture of Web Reports Module

The architecture of Web Reports Module contains five Backbone Views namely ChartView, ChartTab, ChartPanel, ChartNew and ChartBox. ChartView is the main parent and there should be only one instance of this View. There can be multiple instances of the remaining views. The architecture also consists of four models and different views listen to changes from different models. There are also two collections that hold multiple instances of their respective models. The architecture is illustrated in figure 5.2.

<sup>30</sup> "JasperReports® Library | Jaspersoft Community." 2012. 31 Aug. 2016  
 <<http://community.jaspersoft.com/project/jasperreports-library>>

## 5.2.1 Models

As explained in section 4.2.1, the models will contain all the information the application requires. For creating Web Chart reports, I used four models that hold necessary data for their respective Views.

### 5.2.1.1 ChartReportModel

```
ChartReportModel = Backbone.Model.extend {
  defaults: {
    report_title: "Report Title", # Title
    report_id: null,
    checked: false, # Whether collection is displayed or not
    template: false, # if template then true
    form_id: null, # for new Charts
    loaded: false, # true if data is fetched from the server
    chartData: null, # Stores data fetched from the server
    template_data: null, # Parameters needed for generating the
    Report
    # If it is a new Report
    newChart: false, # if chart is new
  }
}
```

This model defines all the necessary attributes that are required to fetch data from the server for the entire report containing data for several charts.

### 5.2.1.2 ChartModel

```
ChartModel = Backbone.Model.extend {
  defaults: {
    title: "Question", # Form Question
    no_of_pois: "53", # Number of Data Points
    chart_type: "bar", # bar/pie/area/line etc
    time_param: "", # For area/line etc Day/Month/Year/Week
    location: null, # if charts needs to be grouped by
    locations
    chart_variables:[ # Domain data for the chart
      {"key":"All year round","value":16},
      {"key":"No response","value":6},
      {"key":"Seasonal","value":29}
    ],
    numeric: false, # True for bar charts with X-axis a
    quantitative value
    xlabel: 'key',
    ylabel: 'value'
  }
}
```

Derived from ChartReportModel's *chartData* attribute. This model will update itself whenever the chartData attribute of ChartReportModel will be modified. It holds

attributes that define each chart within a report. The `chart_variables` attribute is the data needed by d3's module to create the necessary charts.

### 5.2.1.3 NewChartModel

```
NewChartModel = Backbone.Model.extend {
  defaults: {
    report_title: null, # Name of the report
    questions: null, # Questions selected from the form
    filters: null, # Selected filters
    settings: null, # Application specific settings
  }
}
```

This model hold attributes when a new report is being defined or edited by a user. It is primarily used by ChartNew View.

### 5.2.1.4 ChartFilterModel

```
ChartFilterModel = Backbone.Model.extend {
  defaults: {
    formIds: [], # Selected form ids
    locations: [], # Selected locations
    users: [], # Selected users
    fromDate: "",
    toDate: "",
    levels: [] # Levels of locations selected
  }
}
```

This is a requirement of the Poimapper app, as there are global filters specified for the application. Users can then have an option to use these filters to tweak the required data for the reports.

## 5.2.2 Collections

This application requires two collections that hold several instances of their respective models.

### 5.2.2.1 ChartReportCollection

```
ChartReportCollection = Backbone.Collection.extend {
  model: poimapper.models.ChartReportModel
}
```

This collection holds several instances of ChartReportModel. This is used by the parent View i.e. ChartView to keep track of the available reports.

### 5.2.2.2 ChartModelCollection

```
ChartModelCollection = Backbone.Collection.extend {
  model: poimapper.models.ChartModel
}
```

Holds several instances of ChartModel. Primarily used by ChartPanel View to create instances of ChartBox for every ChartModel defined within the collection.

### 5.2.3 Views

As described in section 4.2.3, Views are the Backbone of Backbone.js. They hold the responsibilities of Controllers as well as Views in an MVC architecture. For creating Web reports I decided to use a total of five Views (1 parent and 4 child Views) that specialize in a specific functionality of the Web Reports module. This makes it very effortless to tweak functionalities wherever required by modifying a particular View. New features can be efficiently added by just creating a new child View or editing the existing ones. The hierarchy of the View structure is shown in the figure 5.3. A bottom-up approach will be used to explain the View structure.

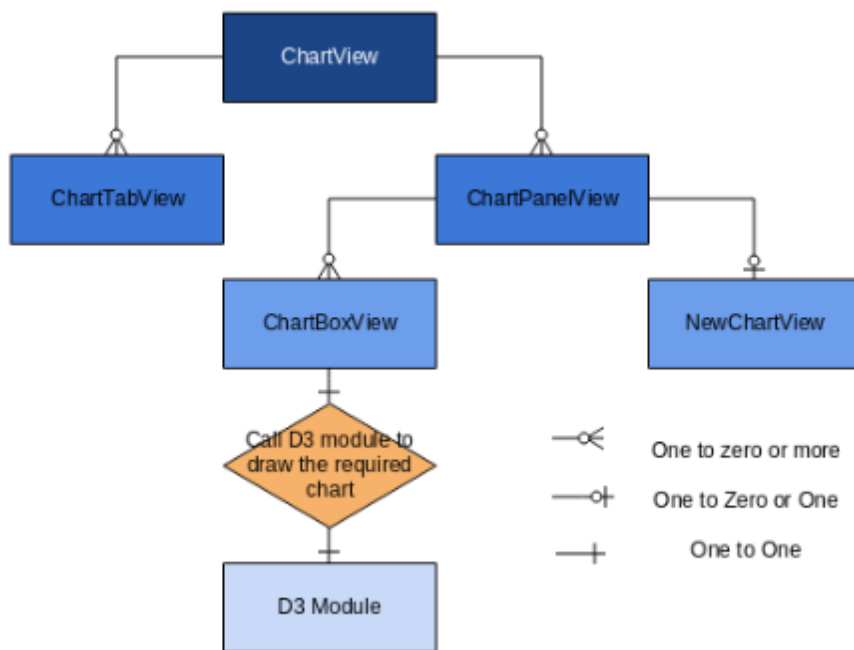


Figure 5.3: View Hierarchy

#### 5.2.3.1 ChartTabView

An instance of this View is created whenever a new ChartReportModel is added to its collection. Every `<li>` tag shown in figure 5.4 is an instance of ChartTabView. It has a very simple functionality. When the checkbox is checked, the report is shown, else it is hidden. When a user checks or unchecks an item in the list, the *checked* attribute of the respective ChartReportModel is toggled. Therefore, any other Views listening to changes for this attribute in the model will update themselves, in this case, the ChartPanelView is listening for this attribute change.

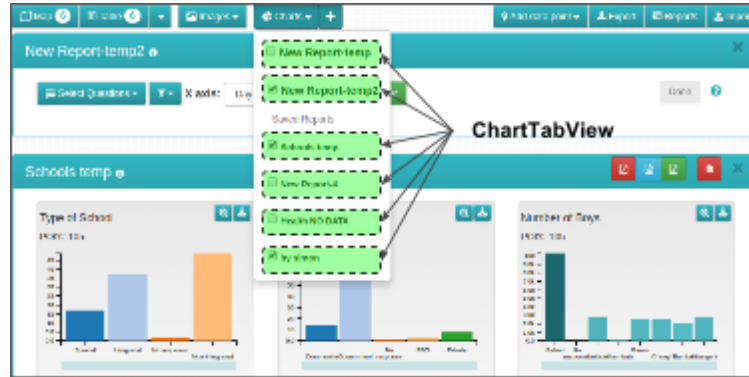


Figure 5.4: ChartTabView

### 5.2.3.2 ChartBoxView

For every ChartModel created by ChartPanelView after the data has been fetched by the server, an instance of ChartBoxView is created. This View then creates the necessary HTML tags required and calls the Chart Object, described in section 3.3.1, with the necessary data that creates the required chart. It listens to the changes in the model and updates the chart when data changes. This view completely isolates the workings of Chart Logic from the Application Logic provided by Backbone.js. This makes chart module reusable within the application.

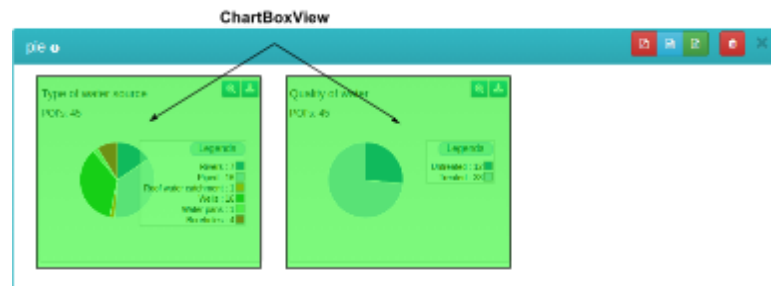


Figure 5.5: ChartBoxView

### 5.2.3.3 ChartNewView

This View is called whenever a user wants to define a new web report. Defining a new Web report is a 2 step process, a user first requires to select a particular form of Poimapper. Then in the next step user can select the required questions, filters and formats. The first step of this process is handled by the parent view (i.e. ChartView), since it requires data and interaction with external modules. ChartView then creates a NewChartModel and stores the required data needed by this view. ChartView also creates a new instance of ChartReportModel with newChart attribute set to true. This attribute is used by ChartPanelView to decide whether ChartNewView needs to be called. This View provides the user with all the questions, filters and tuning options required to define a new report.

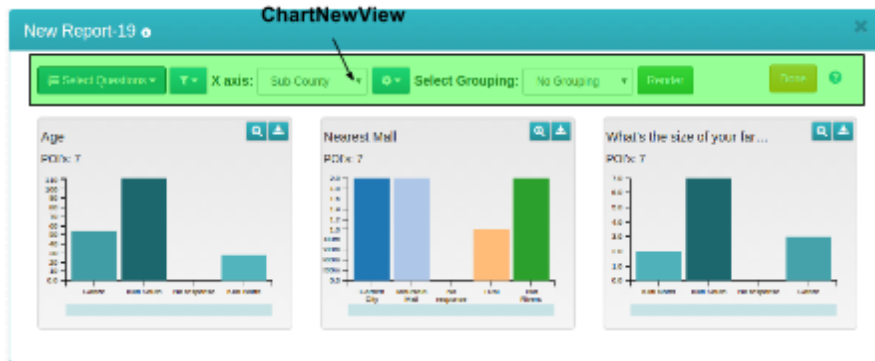


Figure 5.6: ChartNewView

### 5.2.3.4 ChartPanelView

This is the View that contains all elements of Web Report. It calls an instance of ChartBoxView for every ChartReportModel stored in ChartReportCollection. It also listens to changes in the ChartFilterModel for changes in external filters. This View handles other functionalities like downloading the report in pdf, excel and word formats as well as saving, editing and deleting previously stored reports. It calls NewChartView if user is defining a new report.

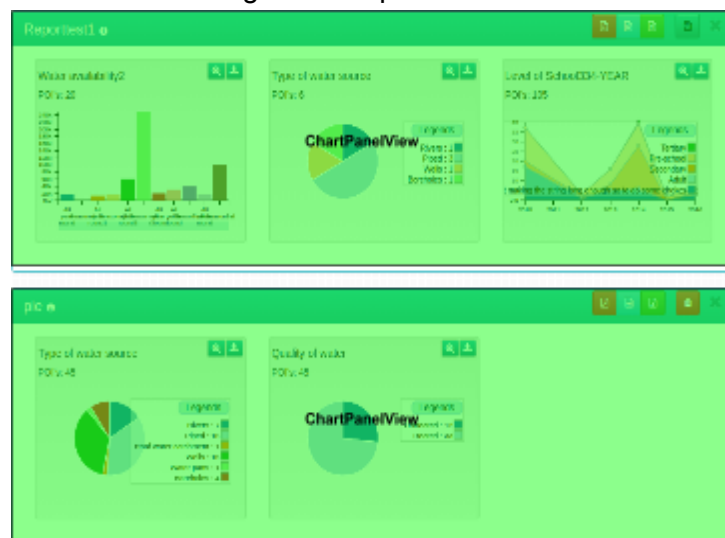


Figure 5.7: ChartPanelView

### 5.2.3.5 ChartView

This is the main parent of the Web Reports module. It initializes the ChartTabView and ChartPanelView and is the only View in the web Reports module that interacts with Views and Models of other core modules. It can also handle various other functionalities like disabling the Web Reports module from users that do not have access rights or for accounts that do not have the feature enabled.

ChartView handles the following functionalities,

1. Initializes ChartFilterModel with filters required for Web Reports. Also initializes ChartReportCollection.

2. Fetches all the previously saved reports by the user. For every saved report fetched or created, ChartView creates a new instance of ChartReportModel. It then adds the instance of ChartReportModel to ChartReportCollection.
3. A listener is attached to ChartReportCollection, whenever a new model is added to this collection, a new instance of ChartTab View is created.

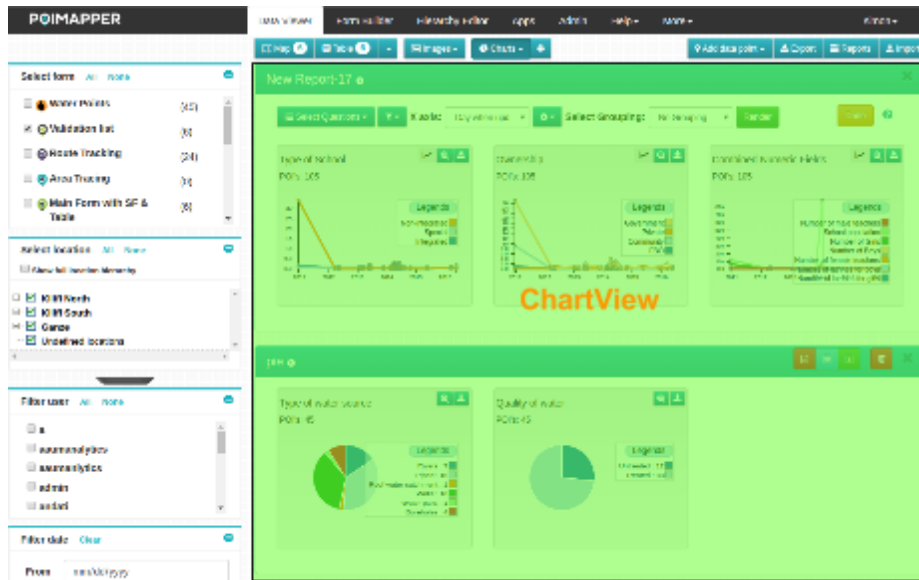


Figure 5.8: ChartView

### 5.3 Adding New Features

Adding new features to the Web Reports module is completely scalable. Since each view is completely independent from the workings of other Views, they are loosely-coupled components (Kaye, 2003). This makes the application versatile to changes. Alternative implementations and improving features becomes an effortless task.

## Chapter 6

# Automated Build Pipeline using Gulp

Modern Web applications generally tend to have two builds of the application, namely *Production* and *Development*. As their names suggests, the Production build is used when the application is to be catered to users, while the development build is used during the process of development. Production build requires to be much more optimized for faster executions and lower loading times. To achieve this, several optimization tasks needs to be performed.

Gulp is a JavaScript task runner that claims to automate these tasks and streamline the process of development and production builds. In this thesis, I will discuss and implement the following tasks using Gulp.

1. Minification of CSS and JS files
2. Concatenation of multiple files
3. Vendor prefixing for cross-browser compatibility
4. Less/SASS/Stylus to CSS compilation
5. CoffeeScript to JavaScript compilation
6. Optimizing 3rd party and custom code
7. Injecting files into HTML
8. File revisions and Version bumping
9. Code Analysis and linting

Before I introduced Gulp into the Poimapper project, our team was manually compiling the entire project using NPM's scripts, which is a node package manager. The package.json file used by NPM consisted of a few defined scripts, that needed to be invoked manually to perform the corresponding task. I found it inconvenient and time consuming.

Before starting with creating a build automation pipeline using Gulp, I would like to introduce to some miscellaneous tools that are being increasingly used for



developing web applications. These tools are not a requirement, but they are certainly helpful during the development cycle.

## 6.1 Miscellaneous Tools

The following are some helpful tools made available by the open-source community. These tools can reduce development complexity and improve the software's quality.

### 6.1.1 CSS Preprocessors

With the evolution of HTML5 and CSS3, many new features like gradients, transitions, and animations were added. These new features increased the complexity of CSS code and made it difficult to maintain. Furthermore, as the CSS code grows, writing and managing styles for various elements becomes an arduous job. Just to change a color used for an element, might require a programmer to find and replace it on hundreds of lines. Even small changes might add up to quite a bit of inefficiency. CSS Pre-processors is a solution to these inefficiencies.

CSS preprocessors are a must have tool for CSS development. Pre-processors like Stylus<sup>31</sup>, LESS<sup>32</sup> and SASS<sup>33</sup> extend CSS functionalities by providing variables, operators, interpolations, functions, mixins and many other usable assets (Davood M et al., 2016). For this thesis, I have used Stylus and the styling code for the project can be found in Appendix A. To learn more about Stylus, visit <http://stylus-lang.com>.

### 6.1.2 CoffeeScript

CoffeeScript is basically JavaScript, just cleaner. Code written in CoffeeScript eventually compiles into JavaScript. It exposes the good parts of JavaScript and makes the code easy to read and understand. For core Python, Haskell and Ruby developers, it is a boon as CoffeeScript takes many elements from these languages. Another great fact about CoffeeScript is that it is created by Jeremy Ashkenas, who is also the creator of Backbone.js and Underscore.js.<sup>34</sup> With CoffeeScript, tools like JSHint and JSLint are no more a necessity as JavaScript's code validation becomes redundant.

Though some main elements of the CoffeeScript will be explained, but to have a better understanding visit <http://coffeescript.org/>

---

<sup>31</sup> (2015). Expressive, dynamic, robust CSS — expressive, robust, feature-rich ... Retrieved July 23, 2016, from <http://stylus-lang.com/>.

<sup>32</sup> (2009). Getting started | Less.js. Retrieved July 23, 2016, from <http://lesscss.org/>.

<sup>33</sup> (2009). Sass: Syntactically Awesome Style Sheets. Retrieved July 23, 2016, from <http://sass-lang.com/>.

<sup>34</sup> (2010). CoffeeScript. Retrieved July 23, 2016, from <http://coffeescript.org/>.

To quickly start using with CoffeeScript

1. Go to <http://coffeescript.org/> and click **Try CoffeeScript** at the top.
2. Include CoffeeScript in an HTML page with `<script src="https://rawgithub.com/jashkenas/coffee-script/master/extras/coffee-script.js"></script>` in the `<head>` then wrap code in `<script type="text/coffeescript"></script>` tags.
3. Install CoffeeScript using Node.js and npm with `npm -g install coffee-script` and then run it with `coffee`.

### 6.1.2.1 Functions

CoffeeScript	JavaScript
<pre>compare = (a, b) -&gt;   if a.x &lt; b.x     -1   else if a.x &gt; b.x     1   else     0</pre>	<pre>var compare;  compare = function(a, b) {   if (a.x &lt; b.x) {     return -1;   } else if (a.x &gt; b.x) {     return 1;   } else {     return 0;   } };</pre>

Table 6.1

The CoffeeScript in the left column of Table 6.1 will be compiled into JavaScript on the right. It is easily noticeable that not only the CoffeeScript looks much cleaner, but one can also get rid of all the useless parentheses and curly braces. Variables do not need to be declared, as CoffeeScript will automatically determine where the variables need to be declared with its scope. CoffeeScript uses Haskell's function description *Integer -> Integer* and Ruby's implicit return of the last statement (Mark B, 2014).

### 6.1.2.2 Whitespace

CoffeeScript uses **syntactically significant whitespace** like Python to group blocks of code. Good developers always indent their code to give a visual clue, regardless of the language used for development. CoffeeScript takes this convention and makes it part of the language to tidy up all the curly braces. Indentation works for any kind of block, if, for, while, functions and classes.

CoffeeScript	JavaScript
<pre>for num in [1..10]   if num % 2 == 0     console.log "#{num} is even"   else     console.log "#{num} is odd"</pre>	<pre>var i, num; for (num = i = 1; i &lt;= 10; num = ++i) {   if (num % 2 === 0) {     console.log(num + " is even");   } else {     console.log(num + " is odd");   } }</pre>

Table 6.2

### 6.1.2.3 Class Definitions

Defining classes in JavaScript is confusing, as there are many ways an OOP-ish type class can be defined. CoffeeScript solves this problem by adding the **class** keyword.

CoffeeScript	JavaScript
<pre>class Person   constructor: (options) -&gt;     {@name, @age, @height = 'average'} = options   getName: () -&gt;     @name  tim = new Person name: 'Tim', age: 4</pre>	<pre>var Person, tim; Person = (function() {   function Person(options) {     var ref;     this.name = options.name, this.age = options.age, this.height = (ref = options.height) != null ? ref : 'average';   }    Person.prototype.getName = function() {   return this.name; };    return Person; })();  tim = new Person({   name: 'Tim',   age: 4 });</pre>

Table 6.3

Notice the number of lines that has been reduced by CoffeeScript compared to the corresponding JavaScript.

There are loads of other features available in CoffeeScript. CoffeeScript removes the ugliest parts of JavaScript and shows its real power. Most of the code for this thesis has been written in CoffeeScript.

## 6.1.3 Bower

Bower is a javascript package manager similar to NPM. A Web application may depend on different libraries, frameworks, assets, and utilities and Bower manages all of them and provides methods to easily manage these dependencies. Bower works by fetching and installing packages and takes care of searching, downloading and saving those libraries<sup>35</sup>.

### 6.1.3.1 Prerequisites

To use bower, the following tools are required

1. Node and NPM
2. Git

To install<sup>36</sup> bower globally (recommended):

```
$ npm install -g bower
```

To install bower only for a specific project (Not recommended):

```
$ npm install bower
```

### 6.1.3.2 Getting Started

Though packages can be installed directly now using the following command,

```
$ bower install <package>
```

it is always a better practice, to initialize bower.json file. Here all the dependencies regarding bower can be saved the same way, as shown for packages.json before for node packages.

To initialize bower.json automatically, type the following command, and then enter the requested fields.

```
$ bower init
```

### Install Packages

Install packages with *bower install*. By default Bower installs packages to bower\_components/ folder. However the default configurations can be changed by creating a .bowerrc file as described<sup>37</sup>.

```
$ bower install <package>
```

To save runtime dependencies in bower.json, use the following command that is similar to npm.

```
$ bower install <package> --save
```

To save devDependencies in bower.json, like testing libraries and other used only during development, use

```
$ bower install <package> --save-dev
```

To install a package of a specific version, use

```
$ bower install <package>#<version>
```

---

<sup>35</sup> "What's So Great About Bower? | CSS-Tricks." 2015. 22 Jul. 2016 <<https://css-tricks.com/whats-great-bower/>>

<sup>36</sup> "Bower — a package manager for the web." 2015. 2 Jul. 2016 <<https://bower.io/>>

<sup>37</sup> "Configuration · Bower." 2015. 22 Jul. 2016 <<https://bower.io/docs/config/>>

## Use packages

To use a particular package in the web application, it needs to be included in the web page like this:

```
<script src="bower_components/jquery/dist/jquery.min.js"></script>
```

However, this can be done using Gulp plugins like `gulp-wiredep` and `gulp-inject`, which will be discussed in a later section. The benefit of using Gulp to inject these CSS and JS filepaths is that Gulp will inject the package and its dependencies in the required order eliminating human errors and also update file paths when a library is updated.

## 6.2 Gulp

### 6.2.1 Installation and Setup

Gulp requires NodeJS and npm; which is a NodeJS package manager similar to Bower. A detailed installation instructions for Node can be found at [npmjs](#)<sup>38</sup>. After the installation is complete, a `package.json` file will be created that will later save the dependencies required by the project.

First a global installation needs to be done using

```
$ npm install --global gulp
```

Then, to add it to our projects devDependencies, type

```
$ npm install --save-dev gulp
```

Finally create a file `gulpfile.js`. Gulp will automatically look for this file and run its commands. This file will contain all of our code and configurations for Gulp. Also, create another file named `config.js`. This file will store all our file paths.

### 6.2.2 Gulp Workflow

Files are processed in Gulp with the help of a pipeline. Files enter into the pipeline, then different tasks are performed on these files within the pipeline. Finally, modified files exit the pipeline through the `gulp.dest('destpath')`. Gulp's pipeline is illustrated in figure 6.1 and Table 6.4

1. The source of the files is specified using `gulp.src('filepath')`.
2. On these src files, modifications such as minifications, compiling, injecting and so on are performed. These tasks are performed within the gulp pipeline using `.pipe(<gulp-plugin>)`, where `gulp-plugin` is one of the helper plugins used by Gulp.
3. Finally the output directory/file is specified using `gulp.dest(<'outputFilePath'>)` where the files exit the pipeline.

---

<sup>38</sup> "02 - Installing Node.js and updating npm | npm Documentation." 2014. 2 Jul. 2016  
<<https://docs.npmjs.com/getting-started/installing-node>>

# Gulp Passes Files Through a Stream

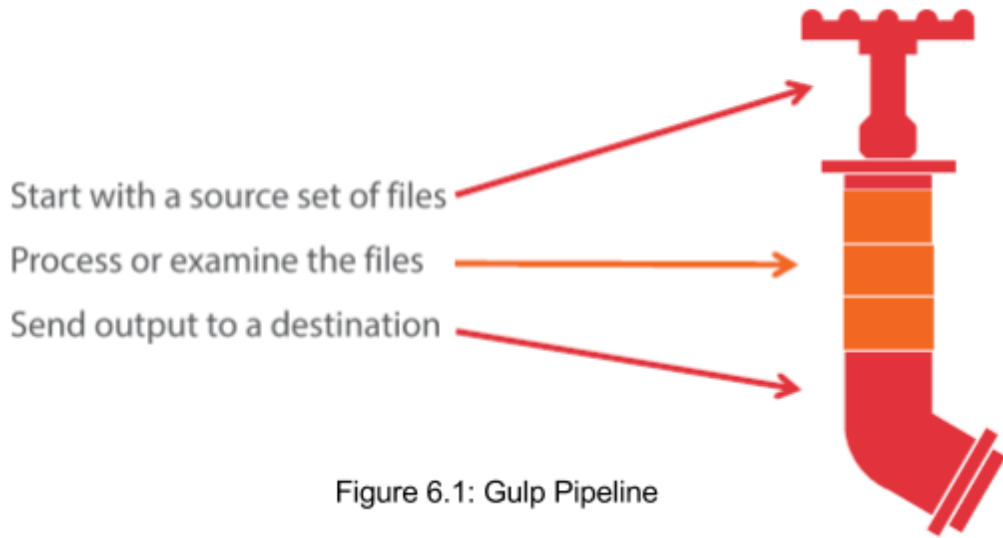


Figure 6.1: Gulp Pipeline


	<pre>return gulp.src(config.backbone)   .pipe(\$.changed(config.backbone_dest,     {extension: '.js'}))   .pipe(\$.print())   .pipe(\$.sourcemaps.init())   .pipe(\$.plumber()) // error logging   .pipe(\$.coffee({bare: true}))   .pipe(\$.sourcemaps.write('./maps'))   .pipe(gulp.dest(config.backbone_dest));</pre>
--	--

Table 6.4: Gulp Pipeline

## 6.2.3 Helper Plugins

There are hundreds of plugins available to use with Gulp that perform a host of tasks within the Gulp pipeline. Few NPM plugins can be used. Some of the useful plugins are mentioned in this section. To install a plugin type

```
npm install --save-dev <plugin-name>
```

1. **browser-sync:** This plugin is very helpful during development cycle. It can test features in synchronization with several browsers at once. Actions taken in one browser are imitated in other browsers. Also if a source file is modified, it automatically injects the changes into the browser saving time by avoiding unnecessary page refreshes.<sup>39</sup>

<sup>39</sup> "Browsersync + Gulp.js." 2015. 7 Sep. 2016 <<https://www.browsersync.io/docs/gulp>>

2. **del:** A node plugin to delete files.<sup>40</sup>
3. **gulp-stylus:** Compiles Stylus files into css. Similar plugins are available for Less and SASS as well.<sup>41</sup>
4. **gulp-autoprefixer:** Different browsers require specific vendor prefixes for using non-standard CSS properties. These vendor prefixes needs to specified within the styling for every browser. To relieve the developer from this ordeal, gulp-autoprefixer adds vendor prefixes so the developer just needs to write plain css and it will interpret and add the vendor prefixes wherever necessary.<sup>42</sup>
5. **gulp-coffee:** Compiles CoffeeScript files to JavaScript.<sup>43</sup>
6. **gulp-sourcemaps:** Sourcemaps creates a mapping between the source and compiled files so that the source can be recreated from the compiled version. Sourcemaps are generally helpful for debugging during the development process. If the development is done in CoffeeScript, it needs to be compiled into JavaScript to run in the browser. If sourcemaps are available, the browser can recreate the CoffeeScript file making it easier for debugging. Sourcemaps works in similar ways for css as well as minifications.<sup>44</sup>
7. **gulp-changed:** Web applications usually have several styling and coffee files. Whenever a certain file is modified, it will be inefficient to run the task on all the files specified at the source of the pipeline. This plugin takes care of it and only allows the modified file to enter the pipeline.<sup>45</sup>
8. **gulp-filter:** Filters out particular files within the pipeline to apply certain specific actions. For eg. only css files needs to be filtered out for gulp-autoprefixer though the pipeline might contain js files as well.<sup>46</sup>
9. **gulp-if:** Provides conditional 'if' logic within the pipeline.<sup>47</sup>

---

<sup>40</sup> "del - npm." 2014. 7 Sep. 2016 <<https://www.npmjs.com/package/del>>

<sup>41</sup> "gulp-stylus - npm." 2014. 7 Sep. 2016 <<https://www.npmjs.com/package/gulp-stylus>>

<sup>42</sup> "gulp-autoprefixer - npm." 2014. 7 Sep. 2016 <<https://www.npmjs.com/package/gulp-autoprefixer>>

<sup>43</sup> "GitHub - contra/gulp-coffee: Coffeescript plugin for gulp." 2015. 7 Sep. 2016 <<https://github.com/contra/gulp-coffee>>

<sup>44</sup> "gulp-sourcemaps - npm." 2014. 7 Sep. 2016 <<https://www.npmjs.com/package/gulp-sourcemaps>>

<sup>45</sup> "gulp-changed - npm." 2014. 7 Sep. 2016 <<https://www.npmjs.com/package/gulp-changed>>

<sup>46</sup> "gulp-filter - npm." 2014. 7 Sep. 2016 <<https://www.npmjs.com/package/gulp-filter>>

<sup>47</sup> "GitHub - robrich/gulp-if: Conditionally run a task." 2014. 7 Sep. 2016 <<https://github.com/robrich/gulp-if>>

10. **gulp-imagemin:** Optimizes image files for production <sup>48</sup>. This reduces filesize and helps conserve bandwidth.
11. **wiredep** and **gulp-inject:** Web applications tend to depend on several external libraries that might be installed through bower (Though there are manual methods but Bower is recommended as it makes these libraries much more manageable). The css and js of these external libraries needs to be included within html using <link> and <script> tags respectively. These libraries might also have dependencies on other libraries, like Bootstrap depends on jQuery and Backbone depends on Underscore. Wiredep helps to automatically inject these file paths into the html (and other variants like jsp, php) files. So developers no longer need to manually add file paths into their files. Similarly, application specific css and js files can also be injected using gulp-inject. <sup>49, 50</sup>
12. **gulp-useref:** During the production build, number of HTTP file requests needs to be reduced. This helps in reducing the overall load time of the page. gulp-useref reads the paths injected in a file through wiredep and gulp-inject, and concatenates many such files into one. It also injects the new filepaths replacing the previous older ones in the html file. <sup>51</sup>
13. **gulp-cssso** and **gulp-uglify:** Minification is a bandwidth optimization technique that reduces the size of code transmitted over the web<sup>52</sup>. During production, considerable performance gain can be achieved through minifying of css and js files. gulp-cssso and gulp-uglify helps to minify css and js respectively. These plugins can also be used within the same pipeline with gulp-useref. <sup>53, 54</sup>
14. **gulp-rev** and **gulp-rev-replace:** Browsers generally cache files locally to reduce requests made to the server. This considerably helps in reducing the initial page load-time. However, it might happen that the remote files have been updated, but the browser continues to use an older cache. This might lead to errors. To resolve this, gulp-rev renames the files with appending revision numbers to their filenames. gulp-rev-replace is another plugin that renames the occurrences of these filenames in the html files. These plugins should be used within the same pipeline as gulp-useref.<sup>55</sup>

---

<sup>48</sup> "gulp-imagemin-npm." 2014. 7 Sep. 2016 <<https://www.npmjs.com/package/gulp-imagemin>>

<sup>49</sup> "wiredep - npm." 2014. 7 Sep. 2016 <<https://www.npmjs.com/package/wiredep>>

<sup>50</sup> "gulp-inject - npm." 2014. 7 Sep. 2016 <<https://www.npmjs.com/package/gulp-inject>>

<sup>51</sup> "gulp-useref - npm." 2014. 7 Sep. 2016 <<https://www.npmjs.com/package/gulp-useref>>

<sup>52</sup> "What Is Minification? - MaxCDN." 2015. 2 Jul. 2016 <<https://www.maxcdn.com/one/visual-glossary/minification/>>

<sup>53</sup> "gulp-cssso - npm." 2014. 7 Sep. 2016 <<https://www.npmjs.com/package/gulp-cssso>>

<sup>54</sup> "gulp-uglify - npm." 2014. 7 Sep. 2016 <<https://www.npmjs.com/package/gulp-uglify>>

<sup>55</sup> "gulp-rev-replace - npm." 2014. 7 Sep. 2016 <<https://www.npmjs.com/package/gulp-rev-replace>>



**15. gulp-load-plugins:** Whenever we need to use a plugin in node, we need to import them using require like

```
var gulp = require('gulp'),
    gulpCoffee = require('gulp-coffee'),
    gulpStylus = require('gulp-stylus'),
    ..;
```

With gulp we use many such plugins and it becomes cumbersome to repeat this process for every plugin. By requiring gulp-load-plugins, all these other gulp plugins no longer needs to be defined. So now all that is needed is

```
var gulp = require('gulp'),
    $ = require('gulp-load-plugins')({lazy:true});
```

Any gulp-<plugin-name> can now be referenced using \$.<pluginName> in the gulpfile. So gulp-coffee becomes \$.coffee and gulp-task-listing can be called as \$.taskListing. The *lazy:true* option specifies to load a plugin only when needed.<sup>56</sup>

**16. gulp-plumber:** Error handling and logging within the pipeline.

**17. gulp-print:** Prints names of files that are being touched.

**18. gulp-rename:** Renames a file.

**19. run-sequence:** Used to run a series of tasks or function in a particular order.

**20. gulp-strip-debug:** JavaScript sourcecode might have a lot of console and debug messages added during the development process. One doesn't need them during production. This plugin gets rid of these messages.

**21. gulp-task-listing:** Prints a list of tasks created in gulp.

**22. gulp-util:** Provides utility functions to gulp plugins such as logging and colors.

**23. yargs:** Arguments can be passed to gulp tasks through command line. This is not a gulp plugin.

## 6.2.4 Build Pipeline

The flowchart of the desired automated build pipeline is shown in the figure 6.3. Parallelograms specifies the file types and rectangles specifies the tasks. The Debug property specifies if the project needs to be built in Production or Development mode

---

<sup>56</sup> "gulp-load-plugins - npm." 2014. 7 Sep. 2016 <<https://www.npmjs.com/package/gulp-load-plugins>>

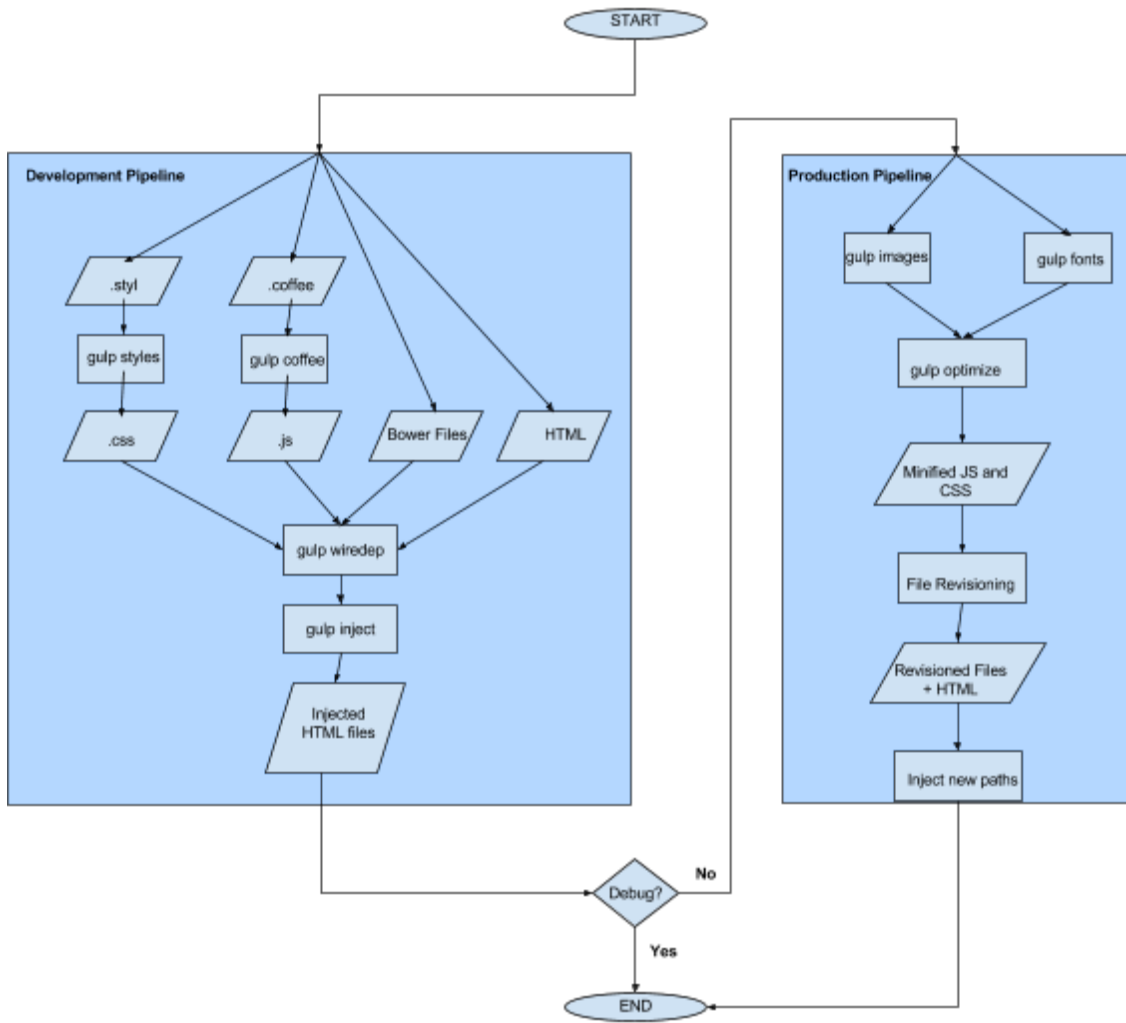


Figure 6.3: Automated Build Pipeline

### 6.2.5 Setting up File Paths

I find it a good idea to set all the file paths required by a project in one single JSON formatted file. This helps in keeping a track of all the files in one place. Gulp can require this file and use its properties. File paths of Bower components, CoffeeScript, JavaScript, CSS, HTML, images, fonts and any other paths that are required can be specified here. Specific configurations for some Gulp and NPM plugins like wiredep and gulp-userref can also be stored here. The code for setting up this file is provided in the following snippet.

```

// gulp.config.js
module.exports = function(){
  var src = './WebContent-src/';
  var prefix = './WebContent/';
  var bower = prefix + "js/vendor/";
  root = './';
  var config = {
    src: src,
    prefix: prefix,
    build: prefix + 'build/',
    packages: [
  
```

```

        './package.json',
        './bower.json'
    ],
    // Paths for .styl files
    styl: [
        'css/style.styl','css/charts.styl',
        '!'+ 'css/test.styl' // Do not include
    ],
    styl_dest: prefix + 'css/',

    // Paths for Backbone's .coffee files
    backbone:[
        src + 'js/**/*.coffee',
        src + 'js/**/*.coffee',
        src + 'js/custom/**/*.coffee'
    ],
    backbone_dest: prefix + 'js/backbone_files/',
    ...
}
return config;
};

```

## 6.2.6 Gulp Tasks

This section will describe a few of the Gulp tasks. Tasks in Gulp are similar to functions or methods. A config object is used to set all the paths and other variables required by Gulp. These variable are defined in gulp.config.js file in the repository as described in the section 6.2.5.

### 6.2.6.1 Stylus/Less/SASS -> CSS compilation

The following code describes compilation of Stylus-> CSS. This process is similar for Less and SASS. Explanation is provided in the comments.

```

var gulp = require('gulp'),
    $ = require('gulp-load-plugins')({lazy:true});
var config = require('./gulp.config')(); // Specifies Filepaths and
configurations
/*
 * 'clean-styles' function is called before running this task.
 * Other functions that needs to be run before the process can be specified
within
 * the array
 */
gulp.task('styles', ['clean-styles'], function(){
    log('Compiling Stylus --> CSS'); // logs the output
    return gulp
        .src(config.styl) // specifies the location of stylus files
in directory
        .pipe($.changed(config.styl_dest, {extension: '.css'})) //
Filters out only the files where there are changes and needs to be
compiled

```

```

        .pipe($.plumber()) // for error logging
        .pipe($.stylus()) // Compiles stylus to CSS
        /*Adds support for different browser platforms that have
more than 5% market share */
        .pipe($.autoprefixer({browsers: ['last 2 version',
'>5%']})))
        .pipe(gulp.dest(config.styl_dest)) // destination folder
    });

```

### 6.2.6.2 CoffeeScript -> JavaScript Compilation

A browser cannot interpret CoffeeScript directly. CoffeeScript therefore needs to be compiled into JavaScript.

```

gulp.task('coffee', function() {
  log('Compiling COFFEE files --> JS');
  return gulp.src(config.coffee_src) // Source .coffee files
    .pipe($.changed(config.coffee_dest, {extension: '.js'}))
    //Recompiles only the updated files
    .pipe($.sourcemaps.init()) // Initialization for generating
    Sourcemaps
    .pipe($.plumber()) // error logging
    .pipe($.coffee({ bare: true }))) // Compiles to Coffee
    .pipe($.sourcemaps.write('./maps')) // Writes source maps
    .pipe(gulp.dest(config.coffee_dest)); // Destination of coffee
files
});

```

gulp-sourcemaps generates a .coffee.map file for every .coffee file compiled. This is helpful during debugging in the browser as a .coffee file instead of a .js file.

### 6.2.6.3 Injecting File Paths

This is one of the most useful features through Gulp that I came across. A Web page usually needs several CSS and JS files to load. Many of these files are external open-source libraries such as jQuery, Bootstrap, Backbone and Underscore that should be preferably installed using Bower. Many of the files are project specific files that are needed for the application to perform the intended tasks. Therefore our .html or other variants like .jsp, .php, .jade would look similar to following snippet.

```

<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
    <title>D3 with Backbone</title>
    <!-- CSS from external Libraries -->
    <link href="/res/js/vendor/bootstrap/dist/bootstrap.min.css"
type="text/css" rel="stylesheet">
    ....
    <!-- Custom CSS -->
    <link href="/res/css/style.css" type="text/css"
rel="stylesheet">
    <link href="/res/css/charts.css" type="text/css"
rel="stylesheet">

```

```

.....
</head>
<body>
  <h1>Hello World!</h1>

  <!-- JS from External Libraries -->
    <script src='/res/js/vendor/jquery/dist/jquery.js'
type='text/javascript'></script>
    <script src='/res/js/vendor/underscore/underscore.js'
type='text/javascript'></script>
    <script src='/res/js/vendor/backbone/backbone.js'
type='text/javascript'></script>
    <script src='/res/js/vendor/d3/d3.js' charset='utf-8'
type='text/javascript'></script>
    <script src='/res/js/vendor/bootstrap/dist/js/bootstrap.js'
type='text/javascript'></script>
    ....
  <!-- Custom JS -->
    <script src="/res/js/charts.js"
type="text/javascript"></script>
    <script src="/res/js/backbone_files/models/ChartModel.js"
type="text/javascript"></script>
    ....
</body>
</html>

```

In the above example, the file paths have been manually inserted. Special care needs to be taken when paths are inserted manually. Many libraries have dependencies on other libraries that need to be called before. If bootstrap is called before jQuery, it will throw an error, as jQuery is a dependency for Bootstrap. Similarly, Underscore is a dependency for Backbone. Application specific JS and CSS also might have similar dependencies. For Backbone files, Models and Collections needs to be called before Views. During the software's life cycle, new libraries might be added, updated or removed, and this will require the developers to manually search, edit and remove file paths, which might lead to cases where errors go undetected, there are orphan file paths as well as dependencies that are no longer required. This can have a significant impact on the performance and load times of a web application.

This problem can be resolved automatically using two of Gulp's plugins, wiredep and gulp-inject. Wiredep primarily deals with injecting all the Bower libraries and its dependencies, while gulp-inject can inject custom .css and .js files. The good part is that all this can be done in a single gulp pipeline with just a few lines of code. First, some tags needs to be specified in the .html files that tells Gulp where and which files needs to be injected.

```

<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
    <title>D3 with Backbone</title>
    <!-- bower:css -->

```

```

        <!-- CSS files from bower goes here -->
    <!-- endbower -->
    <!-- inject:custom:css -->
        <!-- Custom CSS -->
    <!-- endinject -->
</head>
<body>
    <h1>Hello World!</h1>
    <!-- bower:js -->
        <!-- JS files from bower goes here -->
    <!-- endbower -->

    <!-- inject:custom:js -->
    <!-- endinject -->
</body>
</html>

```

As seen all the filepaths have been replaced with just a few tags. Then some options are set for wiredep<sup>57</sup>.

```

config.getWiredepDefaultOptions = function() {
    var options = {
        directory: 'js/vendor/', // directory where bower files are
        stored
        bowerJson: require('./bower.json'), // bower file
    }
    return options;
}

```

This task is named *inject*. Before this task runs, the previous tasks of *coffee* and *styles* needs to be called so that all the files are compiled and available.

```

gulp.task('inject', ['coffee', 'styles'], function(){
    log('Wiring up and injecting Bower\'s css & js and custom css and js files);
    // logs to the terminal
    var options = config.getWiredepDefaultOptions(); // Wiredep options
    var wiredep = require('wiredep').stream;
    return gulp
        .src(config.html_files) // Files where paths are injected
        .pipe($.plumber()) // logs if errors
        .pipe(wiredep(options)) // Injects bower css and js files
        .pipe($.inject(gulp.src(config.coffee_dest,{read: false})), { // compiled
        .coffee files
            starttag:'<!-- inject:backbone:js -->', // tag after which paths will
            be injected
            relative:true
        })
        .pipe($.inject(gulp.src(config.styl_dest, {read: false})),{ // compiled
        ,css files
            starttag:'<!-- inject:login:css -->', // tag where css paths will be
            injected
            relative:true
        })
        .pipe(gulp.dest(config.prefix)); // Output file path

```

<sup>57</sup> "wiredep - npm." 2014. 7 Sep. 2016 <<https://www.npmjs.com/package/wiredep>>

```
});
```

Several customizable options are available for both the helper plugins <sup>48,58</sup>. This is the task that should be run during the development phase of the project.

#### 6.2.6.4 Watch

During development, if we are using tools like Stylus and CoffeeScript, it would not be convenient to manually run *gulp styles* and *gulp coffee* after files are edited. Gulp therefore has an API method *gulp.watch()* that can run certain sets of tasks whenever there are changes in specific sets of files. Several watches can be created that listens to changes from different sets of files and calls the corresponding tasks. The watch task runs continuously on the terminal once initiated. In the following example, two watches are initiated, one that listens to changes in .coffee files and another to listen to Stylus files.

```
gulp.task('watch',['inject'], function(done){
  log(' Watching stylus and coffee files...');
  browserSync = require('browser-sync');
  var coffee_watcher = gulp.watch([config.coffee_src], ['coffee']);
  var stylus_watcher = gulp.watch([config.styl], ['styles']);
  browserSync({<options>});
});
```

An additional function that is mentioned is browserSync(). This function is used to initiate the Browser-Sync and it is quite useful during development as can inject changes into browser and avoids unnecessary reloads. To read more about the available options, check <sup>59</sup>. It can also synchronously simulate events across multiple browsers and has several other useful utilities.

#### 6.2.6.5 Optimize

This task is used to create the production version of the project. It has several subtasks that needs to be performed like concatenation, minification of css and js files, and static file revisions. All these steps are performed within a single gulp pipeline during the production build.

#### **Minification, Compression and Concatenation CSS and JS**

Whenever a web application loads on a browser, the browser requests all the CSS and JS files specified in the document. When all of these files are received and processed by the browser, the page finally completes loading. Minification and compression of these files reduces their overall file size by 30-80%. This leads to a reduction in load times of the webpage. Furthermore, this also leads to a reduction in bandwidth consumption and server load, allowing more visitors to access server

---

<sup>58</sup> "gulp-inject - npm." 2014. 7 Sep. 2016 <<https://www.npmjs.com/package/gulp-inject>>

<sup>59</sup> "Browsersync - Time-saving synchronised browser testing." 2015. 7 Sep. 2016 <<https://www.browsersync.io/>>

resources. Minification removes Comments and whitespaces from the files which are not required for execution during production, speeding up script execution times. Another important aspect is concatenation of a number of files into fewer files. This is required so that the browser makes fewer HTTP requests to the server (Souders, 2008).

All of the above can be achieved using gulp-userref in combination with gulp-cssso and gulp-uglify. gulp-userref requires similar html tag setup like gulp-inject or wiredep.

```
<!-- build:css build/app.css -->
  <!-- list of CSS filepaths -->
<!-- endbuild -->
<!-- build:js build/app.js -->
  <!-- list of js filepaths -->
<!-- endbuild -->
```

The tags specifies the path and name of the final output file. All the files mentioned within those tags will be concatenated into a single file with the name mentioned in the tag i.e. build/app.css and build/app.js. So, the HTML file defined in section 6.2.6.3 will be modified into

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
    charset=UTF-8">
    <title>D3 with Backbone</title>
    <!-- build:css build/app.css -->
      <!-- bower:css -->
        <!-- CSS files from bower goes here -->
      <!-- endbower -->
      <!-- inject:custom:css -->
        <!-- Custom CSS -->
      <!-- endinject -->
    <!-- endbuild -->
  </head>
  <body>
    <h1>Hello World!</h1>
    <!-- build:js build/app.js -->
      <!-- bower:js -->
        <!-- JS files from bower goes here -->
      <!-- endbower -->

      <!-- inject:custom:js -->
      <!-- endinject -->
    <!-- endbuild -->
  </body>
</html>
```

After concatenation, the files will be minified. gulp-cssso minifies the css file, while gulp-uglify minifies the js files. gulp-uglify performs an additional task of mangling the filenames. Mangling replaces the local variables and function names to usually single letters which further improves optimization. This can be an issue in some



cases especially when using AngularJS templates, and it can be disabled through options.

## Static File Revisions

Modern browsers like Google's Chrome, Mozilla's Firefox, Microsoft's Edge, Safari and Opera manage a local cache where css and js files are stored. When a web page requests a file, the browser first checks if it already has a local copy of the file before requesting the file from the server. This improves performance and reduces unnecessary requests to the server. Decision is made by the browser by comparing the url and name of the file requested to its locally cached version. If they are similar, the browser loads the cached file, else it clears old cache and requests the new one from the server. Therefore, file revisioning is crucial whenever the static files like css and js are updated on the server otherwise the browser will continue loading the older caches, which might lead to errors or confusion.

Gulp achieves this task by using gulp-rev; gulp-rev renames the file using content hashing. Whenever the content in a file is updated, so does its hash, giving files automatic revision numbers.

`app.js` → `gulp-rev()` → `app-098f6bcd.js`

Finally the file path of the updated file name needs to be replaced in the ,html file, this is accomplished by gulp-rev-replace. The new filepath is finally injected into the source html file by gulp-userref

The following code snippet describes the optimize task.

```
gulp.task('optimize', ['inject'], function() {
  log('Optimizing the static assets for Production build');
  var optimize = function(){
    var assets = $.userref({
      searchPath: config.prefix, // directory where static files
are stored
    });
    var htmlFilter = $.filter(['**/*', '!**/*.html'], { restore: true
}); // Filters out html files
    return gulp
      .src(config.html_files) // source html files
      .pipe($.plumber()) // error logging
      .pipe(assets) // calling gulp-userref
      .pipe($.sourcemaps.init()) // initialize sourcemaps
      .pipe($.if('*.css', $.csso())) //minify css files
      .pipe($.if('*.js', $.uglify())) //uglify js files
      .pipe(htmlFilter) // filters out html files from the
pipeline

      .pipe($.rev()) // File revisioning on JS and CSS files
```

```

        .pipe(indexJspFilter.restore) // Restores html files back
into the pipeline
        .pipe($.revReplace()) // replaces revisioned file paths in
the html files
        .pipe($.sourcemaps.write('./build/maps')) // generates
sourcemaps
        .pipe(gulp.dest(config.prefix)) // Outputs all the
processed files
        .pipe($.rev.manifest()) // Generates manifests
        .pipe(gulp.dest(config.build)); // outputs the
manifest.json file to the location
    });
    return optimize(); // call optimize() function
});

```

There are several other smaller tasks that are performed by Gulp in this project, such as version bumping, cleaning files, options for logging output, generating fonts and minifying images for production build. The code for these can be found the Appendix A.

## Chapter 7

# Evaluation

In this chapter, I will first discuss the benefits and performance analysis of using an MV\* framework within the Data Visualization Pipeline. It will be then followed by an evaluation of our second sub-topic, regarding the use of JavaScript task runners.

### **7.1 Evaluation of Data Visualization Pipeline with Backbone.js and D3.js**

The discussion regarding the use of Backbone.js with D3.js is subjective and based on my experience while working on it. Many studies have proven the superiority of using MVC frameworks for software development, and therefore using any framework having a substantive open-source community supporting it is always a safe choice. Frameworks are also crucial when working with teams as it sets some rules that needs to be followed and leads to a better architecture and understanding of a software project. Therefore, using any popular frameworks such as AngularJS, Backbone, Ember, Knockout and React for creating data visualizations or any other task would always be helpful. As I have an experience working with AngularJS and Backbone, my basis of comparison will be between them.

Before I started with my internship at Poimapper, I already had an extensive working experience with AngularJS. It takes a fair amount of time and practice initially to become productive with AngularJS. However, after I started working on my internship, I had to switch to Backbone.js as the Poimapper application was already developed using it. I had my own set of qualms of going through the same learning process with Backbone before I can become productive. Nevertheless, I found Backbone.js to be relatively simpler to understand and work with in contrast to AngularJS.

#### **7.1.1 Data-Binding**

As we are concerned with Data Visualizations which highly relies on data, this is the most important property for choosing the framework. Data binding is a

technique that establishes a connection between the application's UI and application's logic <sup>60</sup>. Data binding is established either as a two-way data binding or a newer concept of unidirectional data flow. In two-way data binding, when the properties of the model gets updated, so does the UI and when the UI elements are updated, the changes gets propagated to the model. AngularJS strictly follows two-way data binding, while Backbone gives the flexibility of choosing one of the flavours. In Backbone, it can be achieved with the use of change listeners on the Model.

Recently, unidirectional data flow has gained traction especially due to Facebook's introduction of React.js with Flux which implements unidirectional data flow. Cory (2015) has listed down several benefits of implementing unidirectional data flow over two-way data binding. It increases the lines of code required compared to two-way data binding, but the logical flow of data becomes easier to understand and interpret.

As Backbone leaves the decision of data binding upon the user, it is possible to implement unidirectional data flows using Backbone as opposed to Angular.

### 7.1.2 Performance Analysis

TodoMVC is a very useful tool that can help developers decide on the framework to be used to develop a Web application <sup>61</sup>. It is developed as an open-source project that is an implementation of the same To-Do task management application using different JavaScript frameworks and libraries.

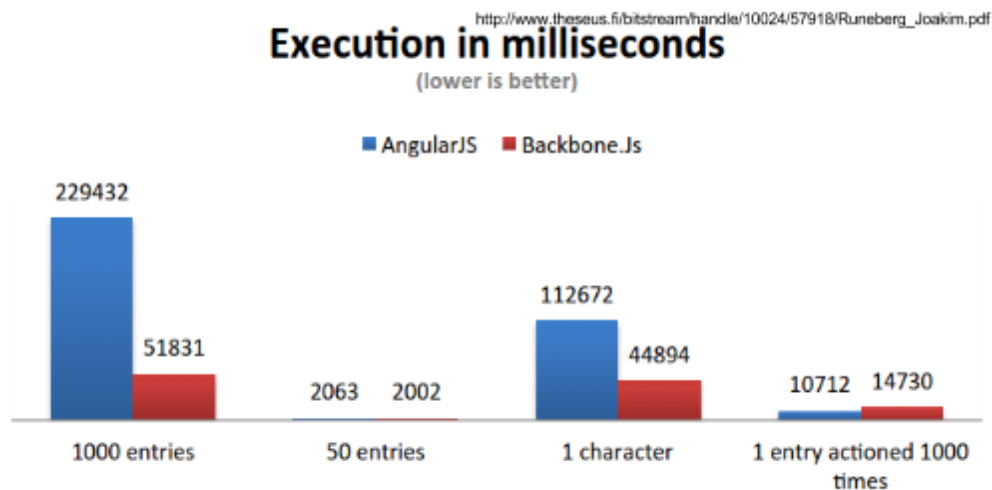


Figure 7.1: Execution in milliseconds

Runeberg (2013) has done several basic performance tests comparing Angular with Backbone using TodoMVC. His study proves many similar performance aspects between Angular and Backbone like *Source Lines of Code (SLOC)* and *Heap Profile*. The difference between their heap profiles is not significant to make

<sup>60</sup> "What is Data Binding? - Definition from Techopedia." 2012. 15 Sep. 2016 <<https://www.techopedia.com/definition/15652/data-binding>>

<sup>61</sup> "TodoMVC." 2012. 11 Sep. 2016 <<http://todomvc.com/>>

any noticeable difference. However, the amount of SLOC required by Angular are far lesser than required for Backbone. SLOC is highly correlated to programming productivity and manageability, and therefore Backbone will require some additional time to develop.

Moreover, when stress tests were performed on both of these frameworks, findings showed that Backbone outperformed Angular in most of the scenarios. Figure 7.1 shows the execution times in milliseconds for four different cases between Angular and Backbone on TodoMVC. Execution times is a crucial factor to provide smoother interactions with visualizations. A slower execution can render the visualization to be clunky. Backbone.js has a faster execution speed compared to Angular.js, and is therefore a better choice for creating visualizations.

### 7.1.3 Loose Coupling and High Cohesion

*Cohesion* is a measure used in software engineering to determine the degree to which elements of a certain module belong together. On the other hand, *Coupling* determines how much one component depends on the workings of other components.

According to Beyer (2001), software applications having reduced coupling and higher degrees of cohesion, tend to be increasingly versatile and robust. A software having high cohesion, will keep similar and related modules together. This helps in keeping modules sharing similar goals or functionality together, such as in our case of Web Reports Module. Backbone provides with a robust implementation through which several modules possessing high cohesion such as Table, Maps, Web Reports, Image Gallery, and Filters could be developed independently accomplishing different goals. Finally, loose coupling between these module helped to bring them easily together to create the Poimapper application.

## 7.2 Evaluation of Automated Build Pipeline using Gulp

Before I introduced Gulp into the project, all tasks were handled using NPM's scripts. Gulp did introduce several new features, but the basic tasks of compiling CoffeeScript and Stylus files, minification of custom JS files remains the same. However, Gulp did outshine when it comes to compiling and minifying all the external library files, Gulp watch and several other minor tasks like images and fonts optimizations, cleaning files, error logging and linting.

### 7.2.1 Improved Developer Efficiency

The efficiency gained from this parameter is hard to quantize, but can be explained. Prior to using Gulp, all developers were required to run a script through npm, that compiled all the CoffeeScript and Stylus files and concatenated them regardless of the environment (Development or Production). Usually, this script used to take somewhere between 30-40 seconds to run. The script required to be manually called on the terminal whenever a developer made changes. This whole

process would take about a minute before the developer can finally analyse the changes on the browser.

Gulp on the other hand, only compiles the the files that have changed, and doesn't require concatenation into a single file during the development mode as it injects file paths automatically. Because of Gulp's watch command, the necessary tasks are called automatically when a particular type of file is changed. The whole process is usually completed within a second, and the developer does not even require to refresh the browser as even that is handled using Browser-sync. NPM can also accomplish the same results, just that it was not implemented in our project, and it is not as versatile and easy to implement compared to Gulp. Gulp took care of the entire build process, and developers only need to concentrate on writing code.

### 7.2.2 Performance Gains using Gulp

These performance analysis are done using Chrome Developer Tools <sup>62</sup>. The performance tests are done on production builds of two versions of the project, one using Gulp and the other using NPM scripts. The following are the Common timeline event properties on which the comparisons are based on, <sup>63</sup>

1. **Loading Events:** Includes events like parsing HTML, loading network resources, receiving data, receiving response and sending network requests.
2. **Scripting Events:** Includes events such as Animation Frame Fired, Garbage Collection, DOMContentLoaded, Script evaluations, JavaScript events, Function calls and XHR events.
3. **Rendering Events:** Includes events like Invalidatelayout, Recalculate Styles and Scroll.
4. **Painting Events:** Contains events that belong to the painting category such as Composite Layers, Image Decoding, Image Resize and Paint.
5. **Other and Idle:** These contain some other events that do not belong to the previous categories. The rest is Idle time where browser is usually waiting for resources and no processing is being done.

As seen from Table 7.1, the build using Gulp takes an average of 3.96 seconds for all the elements of the page to load as opposed to 7.44 seconds of NPM scripts. Furthermore, Gulp outperforms in every category. The overall performance gained using Gulp is 46.77% which is quite appreciative.

---

<sup>62</sup> "Chrome Developer Tools - Google Developers." 2013. 8 Sep. 2016 <<https://developers.google.com/chrome-developer-tools/?PHPSESSID=ebd55af1156a549f3503274c51dd2b8f>>

<sup>63</sup> "Timeline Event Reference | Web Tools - Google Developers." 2015. 8 Sep. 2016 <<https://developers.google.com/web/tools/chrome-devtools/profile/evaluate-performance/performance-reference>>

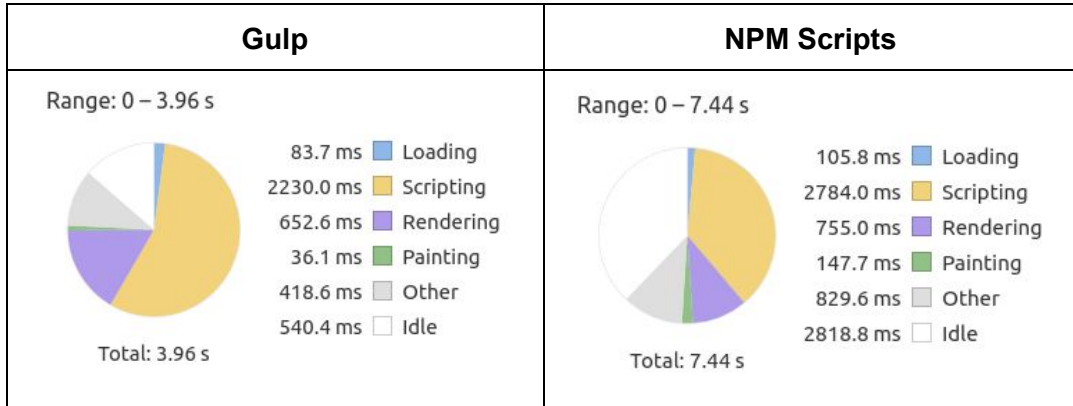
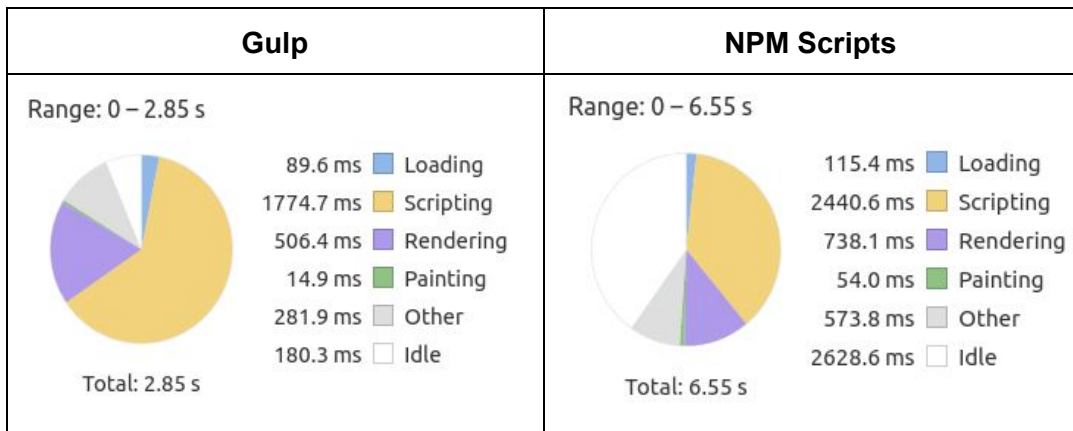


Table 7.1

In the previous case browser-cache was disabled and even better results are obtained when browser-cache is enabled. The following are the results when browser cache is enabled.



If you compare the results to the previous ones, Gulp has comparatively much higher performance gain. The performance gained with cache enabled is 56.49%.

The average performance gained using the automated build pipeline with Gulp is approximately 52%. This is very high and can significantly improve the page response and page speed.

**NOTE:** This comparison is not really between the performance of Gulp and NPM scripts, it is more about the practices followed. Similar results can be achieved with proper configuration of NPM, or even manually building the project by running a certain set of commands on the terminal. It is more about the convenience and versatility provided by Gulp that matters.

### 7.2.3 Page Speed Comparisons

Another analysis is done using a chrome developer plugin, PageSpeed <sup>64</sup>. It uses several heuristics to compare page speeds of web pages to give a score and provides necessary suggestions to improve them. The scores are based on parameters such as optimizing images, leveraging browser caches, file compression and minifications, cache validators and several others. The Gulp version received an overall score of 76/100 while the non-Gulp version received a score of 65/100.

### 7.2.4 Disadvantages of Gulp

Gulp is exceptionally faster if compared to task runners like Grunt or NPM. The reason for this is that Gulp is based on Node and its asynchronous as opposed to synchronous processing of Grunt. It also has in-memory streams and relies on parallel processing which this makes it much faster. This does not sound like a disadvantage, until resources are taken into consideration. I would not have discovered this disadvantage if the company's server was not limited on CPU resources. Gulp is much more resource intensive and the build started failing several times on the server. On my local machine, Gulp was building the application much faster than the non-Gulp versions. However, on the remote server where our application is hosted, Gulp took a much longer time to build. I had to make several changes and optimizations, and avoid tasks that required heavier processing to finally make it work. So make sure to have sufficient resources before using Gulp.

I have also come across several other disadvantages of using Gulp or other similar task runners that are discussed in the following sections.

#### 7.2.4.1 High dependency on Helper plugin authors

Gulp and Grunt depend completely on helper plugins like gulp-inject, gulp-userref and gulp-coffee to actually get a tasks done. These plugins are developed and managed by other open-source contributors, and there is no guarantee that they will always be supported in the long run. Many plugins are also deprecated and no longer supported in newer versions of Gulp. Recently, there was a famous story about the a programmer who almost broke the internet <sup>65</sup>. All he had to do was to delete one of his library that was downloaded more than half a million times from the NPM's repository. This led to an avalanche effect as all the projects and libraries having dependencies on this deleted library started failing, which in turn were dependencies in other projects and so on.

---

<sup>64</sup> "PageSpeed Insights - Google Developers." 2012. 8 Sep. 2016  
<<https://developers.google.com/speed/pagespeed/insights/>>

<sup>65</sup> "NPM left-pad controversy explained - Business Insider." 2016. 11 Sep. 2016  
<<http://www.businessinsider.com/npm-left-pad-controversy-explained-2016-3>>



#### 7.2.4.2 Frustrating Debugging

This one I realised again through experience. Poimapper had multiple language support and the character set specified in <head> tag was utf-8 that stands for unicode. This tells the browser about the character encodings to be used for the document. The project had some Finnish and German characters that were not written in the unicode format, but anyways popular browsers like Chrome and Firefox handle them without issues. However, Gulp processes files in unicode, and now since html files were parsed through Gulp, it messed up the character encodings. This took quite an effort to figure out where the bug was and why it was happening. Debugging might therefore become frustrating as an additional layer of abstraction is introduced.

#### 7.2.4.3 Disjointed Documentation

Many of Gulp/Grunt plugins are actually extensions of core NPM plugins. For example, gulp-wiredep, gulp-coffee, gulp-eslint are extensions of NPM tools like wiredep, coffee and eslint respectively. The documentation of these helper-plugins is never complete and one needs to refer to the core plugin's documentation and fill in the gaps themselves. Furthermore, when a newer version of the core tool is available, one might have to wait until their extended plugins are updated to support them as well.

## Chapter 8

# Discussion and Conclusion

There are many frameworks and libraries available, and all of them have their own sets of strengths and weaknesses. It is therefore very crucial to select the right tools for your project. In this thesis, I studied effects of using a build automation tool like Gulp and resulting implications on an ongoing project. Another important focus of the study was to manage the complexities of interactive visualization using a MV\* framework.

Gulp helped in automating several build tasks, improved efficiency and increased productivity. However, after I introduced Gulp into the project, I received a lot of friction from other developers. It took time for everyone in the team to get used to it. Furthermore, external resources in our project was poorly organized and I had to take a great number of precautions while moving them around without breaking the build. It also sets certain rules that needs to be followed by all developers which helps maintaining consistent architecture and principles for the project. The evaluation results indicate that by using Gulp along with the build pipeline, improved performance of every page event. It also improved Page Speeds and reduced overall load time. However, there are also some disadvantages.

**RQ1** Does a JavaScript task runner help improve overall software efficiency and productivity?

**A1** According to the evaluations discussed in sections 11.1, this depends greatly on the requirements and available resources. It has several benefits, but in many cases the weaknesses might outweigh them. A careful evaluation is crucial. For Poimapper, the benefits marginally outweighs the disadvantages. If our remote server did have enough resources, then Gulp would had been a major advantage. These resources do come with an additional cost and paying extra just to run Gulp might not be always feasible.

Furthermore, Backbone.js had a seamless integration within the Data Visualization Pipeline. This helped enable separation of concerns within the pipeline, and tasks were performed by libraries that are efficient in performing the relevant functionalities.

**RQ2** Can a MV\* framework like Backbone.js help in separation of concerns within the Data Visualization Pipeline?

**A2** The evaluations discussed in section 11.2 proved that Backbone.js has some significant advantages against Angular.js, which is another popular framework. The faster execution speed against Angular.js makes it a better choice to be used within the Visualization Pipeline. Moreover, using any MV\* framework within the visualization pipeline will provide significant benefits such as data binding, loose coupling, higher cohesion and seamless integration.

## 8.1 Future Work

Webpack is a new contender for task runners. In reality, it has created a new family of its own called *module bundlers*. It addresses many of the backdrops of Gulp and Grunt. Furthermore, it does not require custom helper modules, and can directly make use of npm plugin's. Webpack can also create an automated build pipeline with relatively fewer lines of code. It is a strong contender and might dethrone Gulp in near future.

On the other hand, React.js is a recent framework introduced by Facebook. It is not an MV\* framework, but it claims to be the most reactive View module in an MVC. Facebook has introduced Flux, a framework that enforces uni-directional data flow and provides the rest of the skeleton of MVC to work with React. However, React can be plugged in easily with Backbone.js replacing the default View component of Backbone. I believe, React can provide exceptional performance for data visualization due to the use of Virtual DOM technique. This can render data intensive visualizations and its transitions appear to be much more smoother.

# Bibliography

- Amr, Tarek, and Rayna Stamboliyska. "Getting Started with D3." *Practical D3.js* (2016): 75-90.
- Bergman, L.d., B.e. Rogowitz, and L.a. Treinish. "A Rule-based Tool for Assisting Colormap Selection." *Proceedings Visualization '95*.
- Bertin, Jacques. "Graphische Semiologie." (1974).
- Beyer, Dirk, Claus Lewerentz, and Frank Simon. "Impact of Inheritance on Metrics for Size, Coupling, and Cohesion in Object-Oriented Systems." *New Approaches in Software Measurement Lecture Notes in Computer Science* (2001): 1-17.
- Borkin, Michelle A., Azalea A. Vo, Zoya Bylinskii, Phillip Isola, Shashank Sunkavalli, Aude Oliva, and Hanspeter Pfister. "What Makes a Visualization Memorable?" *IEEE Trans. Visual. Comput. Graphics IEEE Transactions on Visualization and Computer Graphics* 19.12 (2013): 2306-315.
- Card, Stuart K., Jock D. Mackinlay, and Ben Shneiderman. *Readings in Information Visualization: Using Vision to Think*. San Francisco, CA: Morgan Kaufmann, 1999.
- Coplien, James O., and Trygve Reenskaug. "The DCI Paradigm." *Agile Software Architecture* (2014): 25-62.
- Eugster, Patrick Th., Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. "The Many Faces of Publish/subscribe." *CSUR ACM Comput. Surv. ACM Computing Surveys* 35.2 (2003): 114-31.
- Finkbine, Ronald. "Programming in CoffeeScript by Mark Bates." *SIGSOFT Softw. Eng. Notes ACM SIGSOFT Software Engineering Notes* 39.3 (2014): 25.
- Gackenheimer, Cory. "Introducing Flux: An Application Architecture for React."

- Introduction to React* (2015): 87-106.
- Kaye, Doug. *Loosely Coupled: The Missing Pieces of Web Services*. Marin County, CA: RDS, 2003.
- Luboschik, M., H. Schumann, and H. Cords. "Particle-based Labeling: Fast Point-feature Labeling without Obscuring Other Visual Features." *IEEE Trans. Visual. Comput. Graphics IEEE Transactions on Visualization and Computer Graphics* 14.6 (2008): 1237-244.
- Mackinlay, Jock. "Automating the Design of Graphical Presentations of Relational Information." *TOG ACM Trans. Graph. ACM Transactions on Graphics* 5.2 (1986): 110-41.
- Macleon, Malcolm. *D3 Tips & Tricks Interactive Data Visualization in a Web Browser*. Vancouver: Lean, 2014.
- Marschner, Steve, and Peter Shirley. *Fundamentals of Computer Graphics*. Print.
- Mazinanian, Davood, and Nikolaos Tsantalis. "An Empirical Study on the Use of CSS Preprocessors." *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)* (2016).
- Meng, X. "Information Graphics: A Comprehensive Illustrated Reference Robert L. Harris, Oxford University Press, New York & Oxford, 1999, 448 Pp. ISBN 0-19-5135326, £32.50." *Quaternary Science Reviews* 19.17-18 (2000): 1837.
- Murray, Scott. *Interactive Data Visualization for the Web*. Sebastopol, CA: O'Reilly Media, 2013.
- Nielson, Gregory M., Bruce D. Shriver, and Lawrence J. Rosenblum. *Visualization in Scientific Computing*. Los Alamitos, CA: IEEE Computer Society, 1990.
- Pfister, Hanspeter, Alyssa A. Goodman, and Eric Mazur. "Perception, Cognition, and Effectiveness of Visualizations with Applications in Science and

- Engineering." *Borkin, Michelle A.* Web. 15 Sept. 2016.
- Rigdon, Steven E. "Power Law Process." *Wiley StatsRef: Statistics Reference Online* (2014).
- Sheelagh, M., T. Carpendale, D.j. Cowperthwaite, and F.d. Fracchia. "Distortion Viewing Techniques for 3-dimensional Data." *Proceedings IEEE Symposium on Information Visualization '96*.
- Souders, Steve. "High-performance Web Sites." *Communications of the ACM Commun. ACM* 51.12 (2008): 36.
- Spence, I. "No Humble Pie: The Origins and Usage of a Statistical Chart." *Journal of Educational and Behavioral Statistics* 30.4 (2005): 353-68.
- "Timeline Event Reference | Web Tools - Google Developers." *Timeline Event Reference | Web Tools - Google Developers*. 12 May 2015. Web. 08 Sept. 2016.
- Tory, M., A.e. Kirkpatrick, M.s. Atkins, and T. Moller. "Visualization Task Performance with 2D, 3D, and Combination Displays." *IEEE Trans. Visual. Comput. Graphics IEEE Transactions on Visualization and Computer Graphics* 12.1 (2006): 2-13.
- Ware, Colin. *Visual Thinking for Design*. Burlington, MA: Morgan Kaufmann, 2008.
- Wolff, Robert S., and Larry Yaeger. "Volume Visualization." *Visualization of Natural Phenomena* (1993): 121-44.
- "The Best Stats You've Ever Seen." *Hans Rosling*:. Web. 18 Aug. 2016.

# Appendix A

## Project Code and Repository

I have created a public git repository for this thesis. I extracted the modules relevant to the thesis from Poimapper, and have created a new project. The repository contains the code for creating the Data Visualization Pipeline using Backbone.js and D3.js. It also contains Gulp's implementation of Automated Build Pipeline.

**Github Repository:** <https://github.com/CodeTaha/backbone-dashboard>

Instructions to build and run the project are as follows.

1. Make sure Git, Node, npm, Gulp and Bower are installed on the machine globally.
2. Either download a zip or clone the git repository.
3. cd to the root of the repository.
4. Run *npm install*
5. Run *bower install*
6. Run *gulp inject*
7. Run *node server.js* in the root directory of the project
8. Open <http://localhost:3000/> in the browser.