# Majestic Applicant Tests

**Tom Goodman**

# 1 Test A - Template Toolkit and Catalyst

On the Majestic website, amongst many other technologies, we use a technology called Template::Toolkit. You can read more about Template::Toolkit here: http://template-toolkit.org/

Template Toolkit separates the web pages, which can be produced in a modified form of HTML, from the rest of the program code which is written in Perl and C#. We use a framework called Catalyst which enforces a separation of Model, View and Controller, often called Web MVC Template Toolkit performs a selection of mail merge type functions on html templates.

Given the following data specified in a catalyst action:

```
$c -> stash -> {'Party'} = 'World';
```

Some configuration, and the following template file:

```
<html>
    <head>
      <title>Greeting Page</title>
    </head>
<body>
    <h1>Hello [% Party %]</h1>
</body>
</html>
```

A web page with a level one heading containing the test "Hello World" will be produced.

## 1.1 Question 1

For the terms Model, View, and Controller, choose which of the following three phrases best applies:

a) Where all the business logic should reside
b) Handles the conversion of user input into calls to the business logic
c) Returns the webpages to the client.

## 1.2 Answer

a) **Model**
b) **Controller**
c) **View**

## 1.3 Question 2

Create a template file similar to the one used in the example above, which for the following data, uses Template Toolkit ( and the FOREACH block directive ) to create an html table displaying a list of shoe model, makes, and sizes.

```
$c -> stash -> {'Shoes'} = [
    {
        Model => 'Aggressor',
        Make => 'Shoe Co.',
        Size => '10'
    },
    {
        Model => 'LookinGood',
        Make => 'Shoe Co.',
        Size => '12'
    },
    {
        Model => 'PowerStrike',
        Make => 'Footware Inc.',
        Size => '7'
    },
    {
        Model => 'RunFast',
        Make => 'Footware Inc.',
        Size => '6'
    },
    {
        Model => 'RunFast',
        Make => 'Footware Inc.',
        Size => '8'
    },
    {
        Model => 'Lounger',
        Make => 'Comfy Shoe Ltd.',
        Size => '9'
    },
    {
        Model => 'Lounger',
        Make => 'Comfy Shoe Ltd.',
        Size => '7'
    },
];
```

### 1.4 Answer

```
<table style="width:100%">
%% FOREACH shoe IN [% Shoes %]
        <tr class="model"> [% shoe.Model %] </tr>
        <tr class="make"> [% shoe.Make %] </tr>
        <tr class="size"> [% shoe.Size %] </tr>
%% END
</table>
```

# 2 Test B - Data Visualisation

There are many ways of visualising data. At Majestic we have lots of data, and wish to visualise it better in the future.

## 2.1 Question 1

Create a program in one of the languages listed below to display the Top 10,000 domains in the Majestic Million (attached as majestic_10000.csv) in an interesting way. You dont have to use the whole file, but any adjustments should be justified.

Permissible languages:

Perl, Python, Ruby, C#, JavaScript, Java and any popular open source third party library.

Attach the program source code and the diagram produced in your answer.

## 2.2 Answer

Firstly, I opted to convert the original csv file into json format, as I personally find json easier to work with. I used an online tool for this (http://www.csvjson.com/csv2json). I originally planned to solely use JavaScript (and HTML/CSS), but I quickly found that using python would be a good idea too.

### 2.2.1 processdata.py

The purpose of this script is to take the json data of the 10000 top sites, and repurpose it into the format required by CanvasJS. I decided to use python for this, as I am somewhat familiar with it's json and collections libraries - having worked with python before. List comprehension provided me with a great way to 'filter' the data to attain what I needed.

The reason I opted to use a separate script for this is so that it would only have to be run once. If I had to run it every time I loaded the page, this would be quite resource-expensive.

Overall, the script reads in the data from the json file, using the built-in python json library. It then takes the data, and removes everything except for the TLD field. Following this, I use a Counter to compress this data into kev:value pairs of TLD:numberOfDomains. From here, I took the top 10 results (i.e. the 10 most common TLDs), and created a new dictionary, c, to hold solely these values.

Following this, I worked out how many of the 10000 domains weren't in the top 10, and accounted for these in the "Other" slice of the chart - because it would have been unrepresentative of the data had I not displayed them at all. I finally worked out the percentage values for each TLD, and dumped the resulting python list of dictionaries to a json file. This is later used by app.js to create the table.

```
from collections import Counter
import json

with open("majestic_10000.json") as f:
    data = f.read()
    json_data = json.loads(data)
    tld_data = [x["TLD"] for x in json_data]
    counter = Counter(tld_data)
    c = dict(counter.most_common(10))
```

```
domains_in_top_10 = sum(c.values())

other_slice_percent = round((100*(10000 - domains_in_top_10)/domains_in_top_10), 2)
percent_data = {}
for tld in c:
    percent_data[tld] = c[tld]/100
percent_data["Others"] = other_slice_percent
json_percent_data = json.dumps(percent_data)
with open("percent_data.json", "w") as f:
    f.write(json_percent_data)
data_points_list = []
for key, value in percent_data.items():
    data_points_list.append({ "y": value, "legendText": key, "label": key})
with open("data_points.json", "w") as f:
    json.dump(data_points_list, f)
```

#### 2.2.2 bigdata.js and datapoints.js

In order to use the json data in app.js, i copied the .json files, and wrapped the json data with: "var <name> = ... ;". This allowed me to directly load the variables bigdata and datapoints into the page later on.

#### 2.2.3 app.js

I had previous experience with JQuery, so I decided to use a little bit of JQuery with this project. I decided that a good way to display the data would be through an interactive (ish) pie chart; for which I found a great library called CanvasJS.

By studying some of their example code, and using it to fit my requirements, I created app.js to handle the creation and rendering of the pie chart itself.

The function rankdiff was part of a larger section of legacy code that I planned to implement if I had time. I was using it alongside another snippet of code to render the original (all 10000) data into a HTML table. After a while; however, I decided that this wasn't an "interesting way" of displaying the data that was provided.

Firstly, I checked to make sure that the HTML was fully loaded before running the script. Following this, I created a new CanvasJS chart using the data contained within datapoints.js. I finally rendered the chart using chart.render().

```
$(document).ready(function () {
    /*
     * Works out whether the current rank of a site is
     * higher, unchanged, or lower than the previous rank, and returns
     * html code containing an up-arrow, hyphen, or down-arrow
     * respectively.
     */
    var rankdiff = function(rank, oldRank) {
        if(rank < oldRank) {
```

```
            return "<b>(    ) </b>";
        }
        else if(rank == oldRank) {
            return "<b>(-) </b>";
        }
        else {
            return "<b>(    ) </b>";
        };
    };
    /*
     * Creates a new CanvasJS Chart called "piechart" and
     * styles it, then adds the data and finally renders
     * it to the page.
     */
    var chart = new CanvasJS.Chart("piechart", {
        title: {text: "Website Quantity by TLD" },
        animationEnabled: true,
        backgroundColor: "#add8e6",
        legend: {
            verticalAlign : "center",
            horizontalAlign : "left",
            fontSize : 18
        },
        theme : "theme2",
        data : [ {
            type : "pie",
            indexLabelFontFamily: "Quicksand",
            indexLabelFontSize: 20,
            indexLabel: "{label} {y}%",
            startAngle:-20,
            showInLegend: true,
            toolTipContent:"{legendText} {y}%",
            dataPoints: datapoints
        } ]
    }); chart.render();
});
```

#### 2.2.4 index.html

The HTML for displaying the pie chart was relatively simple. I made sure to load all of the JS scripts at the bottom of the body (such as to not impede load times), but other than that, the pie chart itself was just a div with the id "piechart", with the rest of the work done by app.js.

```
<! DOCTYPE HTML>
<html>

<head>

<meta charset="UTF-8">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js">
</script>
<link href="https://fonts.googleapis.com/css?family=Quicksand"
```

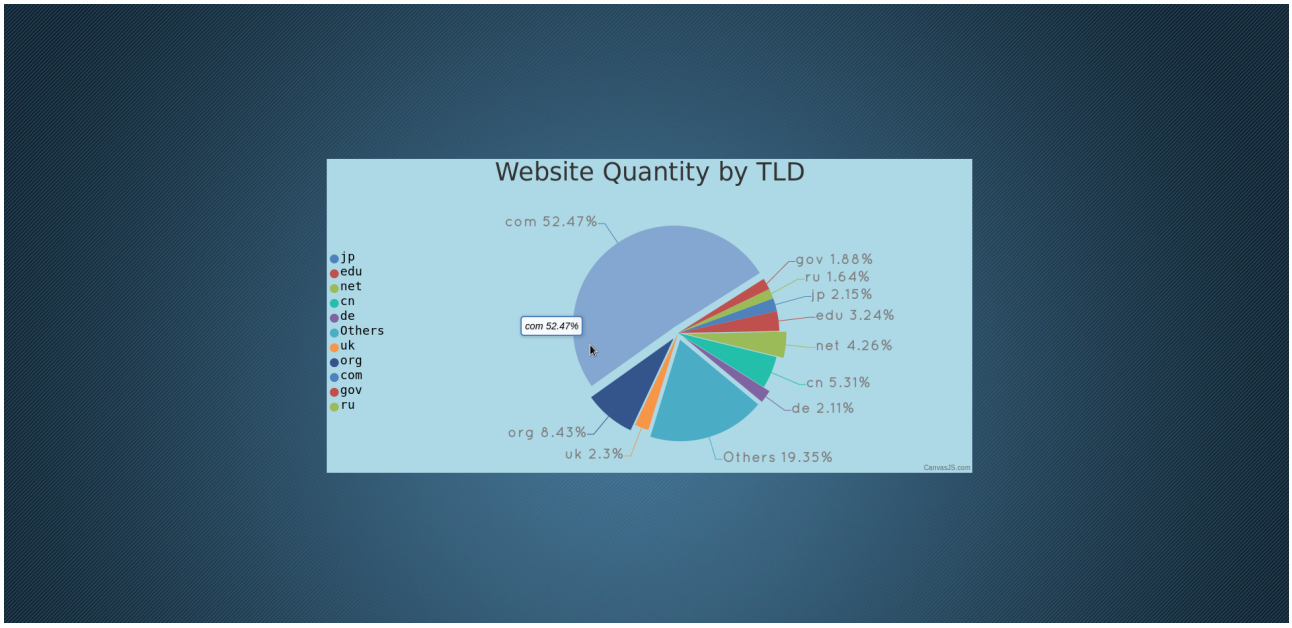
```
      rel="stylesheet"
      type="text/css"/>
<link href="stylesheet.css" rel="stylesheet" type="text/css"/>

</head>

<body>
    <div id="piechart">
    </div>

<script src="datapoints.js"></script>
<script src="bigdata.js"></script>
<script type="text/javascript"
        src="http://canvasjs.com/assets/script/canvasjs.min.js"></script>
<script src="app.js"></script>

</body>

</html>
```

#### 2.2.5 stylesheet.css

The final piece to the puzzle was my css file. I made the body use the font "Quicksand" (although I did redeclare this implicitly in app.js when I created the chart). I used a background image to make the page slightly more colourful and interesting, and I sized and centered the chart.

```
body {
        font-family : "Quicksand" , "sans-serif";
        weight : bold;
        background-image: url('http://i.imgur.com/wvbAhwy.jpg');
        background-repeat: no-repeat;
        background-size: cover;
}

#piechart  {
    height: 50%;
    width: 50%;

    position: absolute;

    top:0;
    bottom: 0;
    left: 0;
    right: 0;

    margin: auto;
}
```

#### 2.2.6 Final Product




# 3 Test C Adapting Code and overcoming fear

## 3.1 Question 1

In this post, we want you to produce a variation of a force-directed graph for some link data seeded from the domains from some academic institutions in the UK. Using the code available here:

http://bl.ocks.org/mbostock/4600693 ( Attached as the file CurvedLinks.html ) And the JSON file provided ( domainlinks.json ) containing the link data seeded from the domains from some academic domains in the UK.

Produce a graph using the data file we provide. Explain any modifications you make to the source code, and include an image of the graph produced.

## 3.2 Answer

After looking in-depth at the CurvedLinks.html, domainlinks.json and miserables.json files, I noticed that both json files appeared to be in a similar format. Because of this, I searched CurvedLinks.html to find out where miserables.json was loaded, and simply replaced it with domainlinks.json. I then ran the html document in my browser to see if it works (which it appeared to do).

### 3.2.1 Resulting Graph