**Summary**

The Visual Layer is structured in such a way that the views and controllers serving the data models are interchangeable. One view can use data from many controllers/data models, one data model can serve data to many views.

**Detail**

Design Pattern Used: MVC – Module View Controller

Views:

The View components consist of html twig template files. These are essentially HTML files containing twig elements which are replaced with data when rendered. This functionality allows for reuse and the dynamic generation of content specific HTML files.

The application can be seen as a SPA (Single Page Application) as the Index page containing all the graph elements.

The Index and Graph pages both rely on controllers to supply the views with data.

Controller:

Data controller is responsible to serve a specific data model to a view thus all data controller methods have the exact same implementation with only the source data file name that differs.

The Index controller works in a similar way

Model:

Each model is pre-compiled and contained in a json file. This static data model is then served by the Controllers to the views.

Each view typically draws off one or more data source. Each data source can be represented by any number of views.

**Script:**

|  |
| --- |
| Role: |
| Hi, my name is Dave. As you might have heard from my colleges already, our team decided early on in the project, to divide the solution into 3 main component areas:   * Data Collection * Data Transformation and * Data Visualisation   I focused most of my effort in the design and development of the Data Visualisation Component.  Looking at the Approach:  In the early stages of the project, the team did a couple of collaborative high level designs sessions and then each developer resource set out to build POC to prove that our communication methods, tools and technologies selection will be feasible.  Tools & Tech:  For Visualisation we decided to use a HTML/CSS/JavaScript Web frontend, using the D3 Charting, Bootstrap component and NodeJS as a backend web/api component. The reason we chose NodeJS is it complements the Web/front-end technology stack in that other than HTML for mark-up and CSS for styling, you only have one programming language to learn. NodeJS will also run on any platform (Windows/Mac/Linux) which was important as part of the experiment was focused on the PI infrastructure which is Linux based however development was done using a Windows machine.  Design:  From a design perspective, we had to cater for different views of the same data in a dashboard form. I chose the MVC type implementation as the pattern address separation of concerns and there by portability and reusability or the underlining components. The MVC pattern was also applied in building the supporting dashboard components such as the index page and criteria view as both of these supporting components required a data model, View and controlling logic. |
| The Index / Dashboard View takes on a view/graph configuration defining the views and their corresponding controller methods. Each controller method returned a specific data model which was simply a static JSON file produced by the Data Transformation component. |
|  |
| Challenges:  Learning new programming languages and frameworks within a short amount of time pretty much sums it up.  Learning how to transform datasets into vector graphics using the D3 engine was by far the most challenging. |

Conclusion:

Learning Big Data and Big Data concepts need not be reserved for people who only have access to enterprise scale hardware and software licenses. We have proven that Big Data project can effectively be developed and executed on entry-level, affordable hardware using mostly open source/free-ware software and IDEs. All made possible by the FOS community.