**Software Development Report – Jordan McDonald**

**1. System Specification**

- introduce the problem to be solved by the system

-constraints (time to get data, if api changes software may suffer)

- target user base

- name - darwin

**2. System Design**

- Storyboards

-UML diagrams

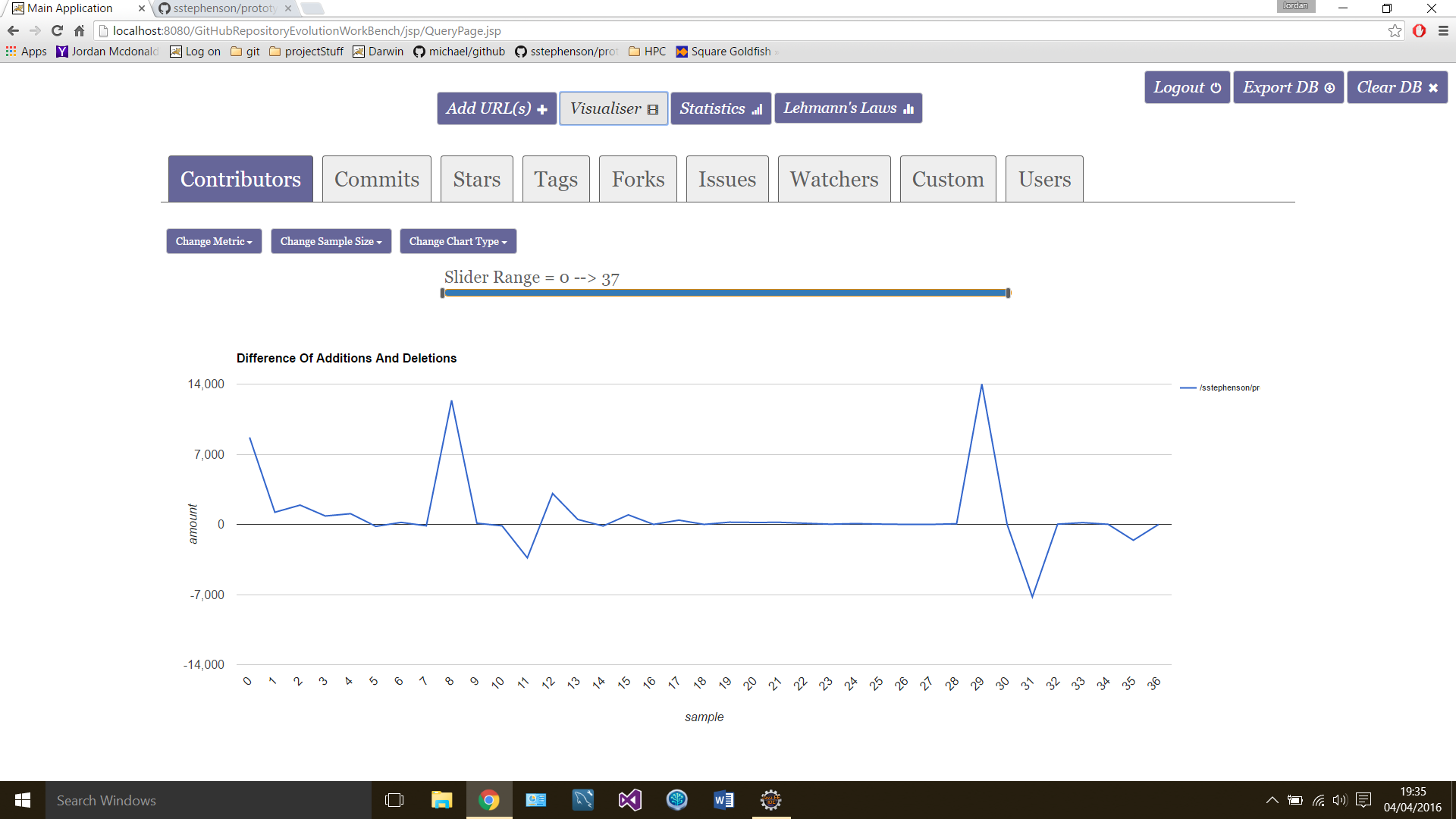
**3. Implementation & Testing**

This section will examine in detail the tools, libraries and environments that have been leveraged in order to realize the system that has been planned in the previous sections of this report. A key facet of the implementation involved extensive research and planning in order to select an approach which meets the requirements of the project and enable the research that would facilitate the dissertation. Each of the key decisions made will now be examined in sequence initially discussing the programming languages selected and concluding with a description of the testing process and verifying the validity of the software system.

**3.1 Programming languages**

In order to meet the requirements of the project the context had to be considered, to form a dataset from the GitHub API it was crucial to select a programming language which enabled direct access to API via HTTP requests. To account for this JavaScript was selected and in tandem leveraged the JQuery library which simplifies HTML document traversal, DOM (document object model – a cross-platform and language-independent convention for representing and interacting with objects in HTML) manipulation, event handling and animation. However the main reason that drives this choice of library is the AJAX functionality which allows the webpage to dynamically send various requests to the API to account for user input or as part of automated process using callbacks. Additional reasons that support the selection of JQuery include JSON parsing and manipulation – the API will return data in the JSON data format (key value-pairs) and therefore it was crucial to have a system that could perform robust processing of this data, something JQuery enables. It should be noted that other options for interacting with the API were available, examples include Octokit (Ruby and C# versions) which is a GitHub endorsed alternative, however the project was not mature enough for this particular context and neglected the ability to make certain requests whereas through AJAX any functionality/data the API provides can be utilized.

Based on the advantages stated for selecting JavaScript a decision was made to frame the project as a web application, therefore consideration for libraries and tools that would bring the storyboard designs of the system to fruition. To fulfill this the Twitter Bootstrap JavaScript language was identified which provides a large selection of components to develop appealing user interfaces, in particular the ‘tab’ navigation would prove to be the focal point for all user interaction with the system. In addition to this Bootstrap provides a ‘mobile first’ approach which liquid displays that adjust to become aesthetically pleasing on different devices via pre compiled styling that would reduce the amount of micro-managing required by the developer. Visualization of the data extracted from the API is a key requirement that was partially facilitated by the use of the Google Chart library which can render various graphs in appealing ways, a dependency for this library was JQuery was solidifies that as a prudent initial decision. In order to provide additional variety to the presentation of data another component of JQuery was integrated, JQuery UI which provides additional options and was key in proving further ways to engage a user. In order to further accessibility and reduce load on the database the Facebook login SDK was utilized to externalize the user management process and offer and additional way to access the system. To see an example of JavaScript and these libraries working in tandem to create a vivid UI, see figure \* below.



In order to Flesh out the capabilities of the web application to enable the storage and analysis of the mined data from the GitHub API a centralized server would be required, in this case the server side language is Java. This language provides an ideal method to interface with the web page via the Servlet technology which allows a server to send and receive HTTP requests in formats such as JSON so fits the overall architecture and work flow of the system being developed. In addition to this it was crucial that the server side language can communicate with the statistical analysis environment and the database technology (to be covered in the next section) which reinforces the choice of Java, as connectors and interfaces are provided which enable this process. To assist the default Java functionality a series of additional libraries were leveraged, GSON which performs serialization/deserialization of JSON to and from Java Objects, since all data instances on the server are modelled as Java beans this was a crucial library to standardize this process. In addition to this Reserve was utilized (a library to allow java to communicate with an R server instance) as well as Junit (for unit testing the system) and the Mongo Java driver in order to facilitate communication with the database, each of these will be examined in detail in the following section.

**3.2 System Environment**

Initially the development environment has to be considered, for this project eclipse was selected as the IDE. Advantages of this choice include integrated configuration with the project files and the Apache Tomcat web server which will be utilized to host the web application and support the use of several JEE specifications to enable the servlets to send and receive requests. In addition to this Apache Maven can also be easily included into the work flow using eclipse and provides the option to automate the build process of the project, however at this stage has not been pursued but could become useful in future versions of the software, as well at this maven provides simplistic management of dependencies which was a key driver of using this functionality. Eclipse also provides different project configurations, in our scenario the ‘dynamic web project’ was selected which ideally prepares the system by generating a POM, Servlet dependencies and a consistent folder structure.

Now to discuss the various facets of the systems components that interact in order to provide the functionality in an efficient and cohesive manner that meets the requirements. Figure \* shows a diagram that shows an overall view of each discrete part and the manner in which they communicate with each other. Two parts of the system are pending discussion, initially MongoDB will be considered which serves as the database for this application storing parsed API data and user details. MongoDB is a ‘no-SQL’ database which forgoes the typical relational model in place of documents that have dynamic schemas structured in a JSON format, each document is typically a member of a collection which contains a large number of similar documents – for example in a ‘commits’ collection will be a series of document each representing the commit data for one project. In this application context a database with no relations makes logical sense as each metric extracted from the API can be contained within a collection of that type and then composed of documents containing the actual data for a repository. Mongo also provides a driver to interface with Java, form this server side dynamic querying is possible in addition to exporting, resetting and dropping the database collections which add useful utility functionality to the project. In order to manage the database a GUI tool was utilized ‘Robomongo’ which connects to the mongo database and supports creating collections, removing documents and importing JSON documents and in general makes testing and managing a large dataset of documents more convenient.

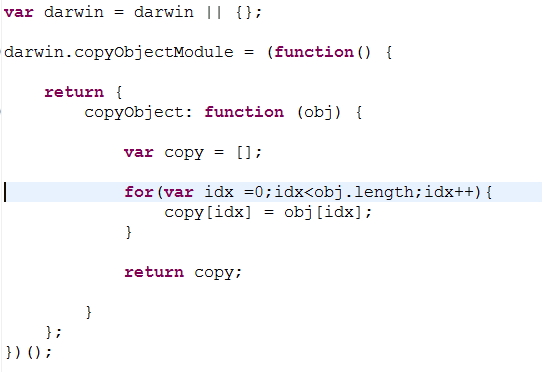
In order to enable reliable and robust statistical analysis of the data the R software environment was chosen as the platform. Java can interface with R via Rserve which is a TCP/IP server which allows other programs to use facilities of R and provides a library of operations and datatypes to enable manipulation of the returned data. This statistical analysis is key in allowing the workbench to directly show results dynamically based on user input, whether the entire the database is evaluated to fulfill the research objectives or a statistical measure is performed on a select data from the UI.



**3.3 Design Patterns/Implementation techniques of note**

In order to devise a system that meets the requirements and also follows best practices in regards to software engineering a series of design patterns where considered that would shape the low level approach of the coding process in order to enable ease of maintenance (e.g. fix bugs without impacting separate functionality) and adaptability (e.g. simplify adding new features).

The web specific portion of the application will be evaluated initially – the module pattern [1] was leveraged in order to encapsulate a discrete aspect of the systems functionality into a generic reusable ‘module’. Each module was contained within a global namespace (Darwin), benefits of this include minimizing the impact of external JavaScript libraries interfering with the custom written code (if variable names clash etc.) by designating a project scope, therefore to access any modules the namespace would have to be directly invoked which mimics the approach utilized by JS libraries. While JavaScript does not provide an ‘out of the box’ concept of privacy the module pattern can alleviate this by declaring variables and functions inside a module which prevents code from directly accessing the functionality without referencing the module itself which overcomes a key JavaScript shortcoming. However if it is desirable that a method if public the ‘return’ keyword can be utilized to give external module code access to the function/variable which provides flexibility in choosing what components have privacy enforced. In order to help visualize this process an example of a JavaScript module that is leveraged in the project is provided in figure \* which highlights the concepts which have been examined.



While the use of the module pattern has numerous advantages, it raises the question, how do the modules communicate with each other? A behavioral design pattern known as mediator was introduced. The mediator itself is represented as another JavaScript module with the purpose of coordinating data to and from other modules in the system which introduces low coupling as each module is an independent functional portion of code that do not rely on dependencies. The mediator was also given license to perform basic preprocessing to the data in some cases to ensure the module could operate as expected or to account for variation in structure of the different datasets extracted from the API. As the project increases in size beyond the current scope the Mediator may as a result become bloated, therefore it could become prudent to split the mediator into partitions to account for each facet of the system, its key to be aware of limitations to a design pattern as well as the strengths. A final consideration is even handling, each ‘tab’ of the system is bound to an ‘input manager’ which handles various user input and routes the selection (and data) to the Mediator for further processing. A general overview of the JavaScript program flow can be seen in figure \*.

User Input (UI)

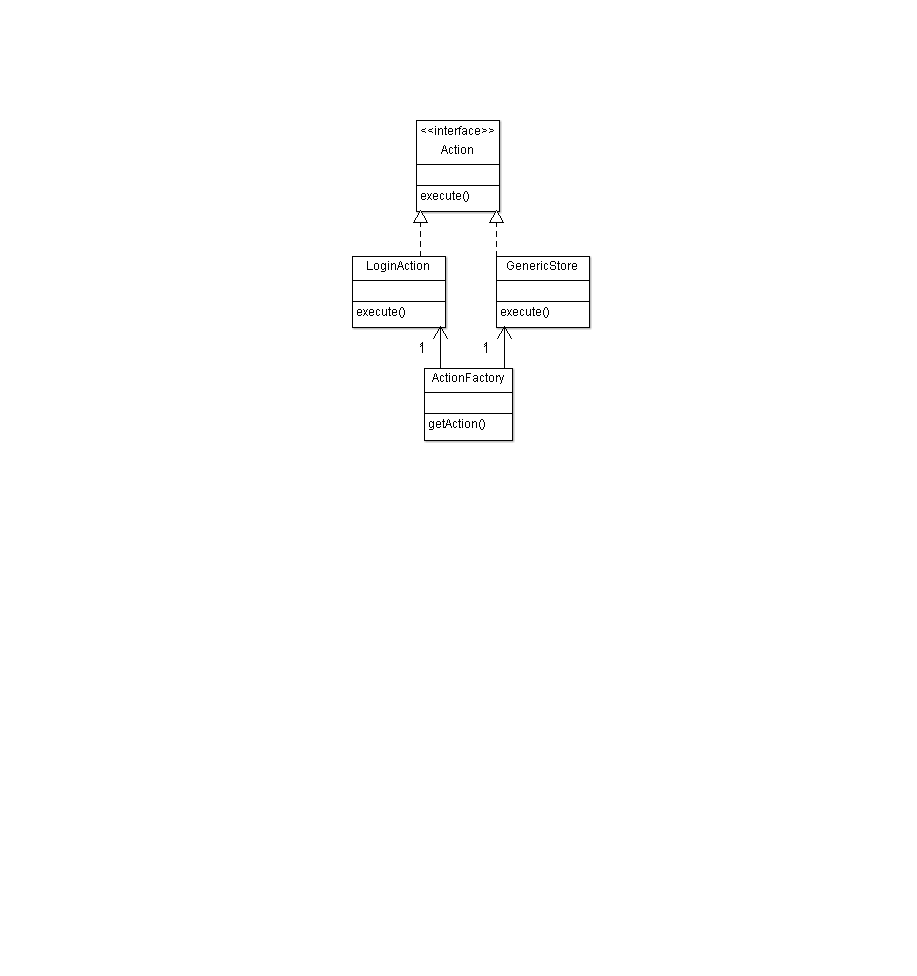
Input Manager

Mediator

Module A

Module B

Design patterns implemented on the server include the creational factory pattern which handles the various types of ‘actions’ that need to be performed on the server ranging from handling the login process to storing data in MongoDB. Advantages of applying this pattern are include the ability to create a new object without exposing the creation logic to the client while executing using a common interface, in this case an ‘Action’ Java interface which is implemented by each specific action type to use an ‘execute’ method. In addition to this code adaptability and reuse is simplified as introducing a new class/modifying a class or action involves adding code to the factory rather than managing instantiation instance. The factory method has the responsibility of returning the correct action based on the input received in the HTTP request, the execute method is then called and the specific operation then proceeds as desired. See figure \* for UML showing and example of this process.



An additional best practice that was implemented was the use of data access objects (DAO). This technique enforces the single responsibility principle by designating all persistence (a different DAO for each Mongo collection) to the DAO which separates the application from the database to enable each to evolve independently without affecting each other, this heightens loose coupling and ensures all persistence logic for a collection can be accessed in one location. Unit test also benefits from this design pattern as the database operations can be tested in isolation effectively on a cloned test database without relying on application specific code (known as actions in this project).

**3.4 Test Strategy/process (unit + integration)**

**3.5 Important Components/Functions**

-test suites

-integration tests

-algorithms/functions of note? Parser?

-describe import components?

**4. References**

[1] https://toddmotto.com/mastering-the-module-pattern/