Project Implementation
COS 301 Buzz Project
Group: Reporting B
Version 1.0

Carla de Beer 95151835
Phethile Mkhabela 12097561
Ephiphania Munava 10624610
Joseph Murray 12030733
Ryno Pierce 12003922

https://github.com/BuzzSpaceB/Reporting

University of Pretoria

17 April 2015

Contents

1 Introduction

The functionality provided by the buzzReporting module includes the following:

- It provides statistical information that can be used in the interface to display facts about the average user and how the logged-in user compares with the average.
- It provides ways to gather data that is in the persistent store and present it in a format that is usable by other modules that are external to Buzz.
- It provides functionality to alter record sets in a Buzz space by uploading the relevant information that is stored in a csv file.

IMPORTANT: The GitHub repository for this project was accidentally deleted on ??/??/????. Ryno Pierce managed to restore the repository from a backup copy. It is requested that marking scheme of the git hub contributions takes into consideration the fact that the repository had to be restored and that the apparent skew in contributions is not as a result of the group not evenly dividing up the workload, but as a result of emergency repository restoration.

2 Use cases

2.1 Treads.getThreadStats

The functionality provided by the getThreadStats function is to provide a versatile way to get statistical information of subsets of posts complying with specified restrictions.

2.1.1 Functional Requirements

The parameters passed to the function is

- a set of posts returned by the Threads.queryThread function.
- action keyword

The set of posts returned by the *Threads.queryThread* function is analysed according to the specified action keyword.The following describes the result returned upon each of the action keywords:

- **Num** A count of the entries in the dataset that was created.
- **MemCount** A count of the number of members who are the creators of posts in the dataset.
- **MaxDepth** The maximum depth of a post in the queried thread tree.
- **AvgDepth** The average depth of a post in the queried thread tree.

2.2 Get Thread Appraisal

2.2.1 Functional Requirements

2.3 Export Thread Appraisal

2.3.1 Functional Requirements

2.4 Import Thread Appraisal

2.4.1 Functional Requirements

2.5 Export Thread

2.5.1 Functional Requirements

2.6 Import Thread

2.6.1 Functional Requirements

3 Architectural components

The project made use of the following technologies architectural components, technologies and frameworks used to address the architectural responsibilities and the architectural tactics chosen to address the quality requirements as specified in section 3.2 of the Master Specification, version 0.1 (16 March 2015):

3.1 JavaScript

The project was implemented using JavaScript, as indicated as a requirement in the Master Specification, version 0.1.

3.2 Node.js

Node.js was employed as the execution environment within which the system was deployed. Node.js implements the asynchronous-processing by providing a framework for event-driven asynchronous callbacks in the form of asynchronous libraries. As a result, each of the function implemented inside the folder *ReportingB*, makes use of callbacks, and the functions themselves are called via Reporting.js, making use of `module.exports.<function name>`.

For example, the functionality `getThreadStats`, which is located in a file called `getThreadStats.js`, returns a callback:

```
   module.exports = function(posts, action, callback)
...
callback(getAveDepth(array, arrayParentID));
```

This callback is then sent through to the calling function in Reporting.js:

```
module.exports.getThreadStats = require('./ReportingB/getThreadStats.js');
```

3.3 ???? Electrolyte ?????

3.4 ???? Anything else ?????

4 Development architecture

4.1 Version Control

The project was undertaken by means of the GitHub repository, as created by the top-level integration team. The URL of this repository is indicated on the front page of this report.

4.2 IDE

The WebStorm IDE was used for the generation of the code for this project, which facilitated the generation of JSDoc comments. All builds were driven by npm and are IDE independent.

4.3 Builds

The node.js package manager (npm) was used to build the project artefacts. To this purpose, a *package.json* project descriptor has been included into the repository. New versions of the code that passed the unit testing were published in an npm registry, made accessible to all teams in the B-group.

4.4 Unit testing

???? The unit testing was undertaken by means of npm. The file `UnitTests.js` contains the unit testing information. The initial unit tests were undertaken manuallly

4.5 Documentation

JSDoc documentation metadata was used throughout the project to annotate each function. The use of WebStorm as an IDE facilitated the generation of these annotations.

4.6 Bug tracking

Bug tracking was undertaken by means of GitHub's issue tracker.