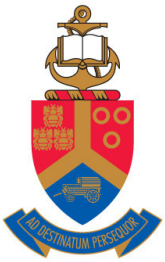

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

COS301 Mini Project

Round 4

Testing Report: Reporting module

Vivian Venter *u13238435*
Tienie Pritchard *u12056741*
Lindelo Mapumulo *u12002862*
Sphelele Malo *u12247040*
Martha Mohlala *u10353403*
Sean Hill *u12221458*
Goodness Adegbenro *u13046412*
Tsepo Ntsaba *u10668544*
Michael Nunes *u12104592*

Here's a link to GitHub.
`https://github.com/VivianLVenter/TestPhase-Reporting`

April 24, 2015

Contents

1 Testing TeamA

1.1 Functional Requirements Review

1.1.1 Treads.getThreadStats

The getThreadStats function has achieved what it needs to do but it has minimal error checking. The parameters that are required are all available. The set parameter contains a list of posts. The action parameter is tested for Num, MemCount, MaxDepth, AvgDepth. The action that is specified will carry out the appropriate action to be done with the threads such as calculate the number of threads. There is no error checking so if the action specified is not one of the ones in the functional requirements then the return value will be null which could cause a problem when the module is integrated with the other modules. There is no exception thrown so the function will exit without any knowledge of an error.

Full use case test rating

Partial Pass

The function will run but in some cases the lack of error checking will cause errors in integration.

1.1.2 Get Thread Appraisal

The GetThreadAppraisal function works and produces the required output and will calculate the appraisals. All the parameters and post conditions are present. The data is stored correctly into a dataset, JSON string, which contains all the entries for that thread and a action value. The action value which is specified as one of the parameters to be either Sum, Avg, Max, Min or Num. The entries contain all of the threads and the information about the thread including an ordinal value for the thread which is used in the calculation that is specified by the action. There is no error checking so if the action value is not one that is specified then the action value will not be set and this could cause problems in integration.

Full use case test rating

Partial Pass

The function will run but in some cases the lack of error checking will cause errors in integration.

1.1.3 Export Thread Appraisal

The ExportThreadAppraisal function could not be found. This function has a medium priority and is important to the system. No testing could be done on this use case. This functionality is important as it allows the user to export the thread appraisal and allows offline editing.

Full use case test rating

Failed

The function is missing so no testing.

1.1.4 Import Thread Appraisal

The ImportThreadAppraisal function could not be tested because it was not implemented.

Full use case test rating

Failed

The function is missing so no testing.

1.1.5 Export Thread

The ExportThread function was tested and runs successfully. It provides the functionality to backup the content of a thread or subset of a thread. It achieves this by getting the threads using the ThreadID parameter. It then converts the threads to a serialized object and writes the serialized thread object to a file for backup. However, this function does not throw exceptions when errors occur. It only sends an output to the console which will not be visible to the user. Therefore, there is no way of knowing if an error has occured.

Full use case test rating

Partial Pass

The function will run but in some cases the lack of error checking will cause errors in integration.

1.1.6 Import Thread

The ImportThread function was tested and runs successfully. It provides the functionality to restore the content of a thread or subset of a thread that was stored using the ExportThread function. It achieves this by retrieving the serialized thread that was exported to a backup file in the ExportThread function. It converts the serialized thread to an unserialized thread object and returns the imported thread. Therefore, this function meets all the pre and post condition requirements. This function however, does not throw appropriate exceptions if an error occurs to notify users of this error.

Full use case test rating

Partial Pass

The function will run but in some cases the lack of error checking will cause errors in integration.

1.2 Architectual Requirements Review

1.2.1 Intergration Channels

Manual Intergration The module allows for manual intergration by exporting JSON objects into csv files so that they can be used by other modules.

1.2.2 Software quality requirements

Reliability: Parameter values aren't checked before db access is made, which shouldn't be a problem, since these functions aren't directly called by a client, but garbage values could cause a fatal error. Errors aren't very well handled, and in some cases aren't getting handled at all. There was no package.json file so it is not easily deployable because you cannot see what dependencies the module needs. The function export thread appraisal could not be found, so the module cannot realise an offline facility to apply a manual appraisal. Its also the same case with import thread appraisal.

Maintainability: The code is well laid out, very easy for a programmer to understand and add upon. However though there are too many files and directories, most of the files could have been out together to create a single file. The technologies used also seem like they will be used for quite some time to come, making the module easily maintainable.

Test-ability: No unit test where found for this module, but the easy readability makes testing easier. The (mostly) lack of error handling makes the challenge a bit more daunting.

Audit-ability: There are no tables in the db for auditing, so tracking the origin and correctness of a transaction is impossible. The module provides a function called getThreadStats which provides a way to report user threads and posts, unfortunately thats only providing logs for another module not the reporting module.

Perfomance: There were no unit tests for this module so could not test how long it took for the reporting functions to complete.

1.2.3 Architectural components

Framework(s): To use this module as part of the system, NodeJS was used. Handlebars templating framework was used to render templates on the client side and server side.

Technologies: Javascript was obviously used for implementation, JSON was used to pass mongoDB documents as objects, and json2csv was used to export some of those JSON strings as csv files. There was no trace of mocha, unit.js or electrolyte being used as specified. HTML was also used to display the values returned. Mongoose was also used to wrap up the native Node.js MongoDb driver. Node-serialize was used to serialize objects into JSON objects.

2 Testing TeamB

2.1 Functional Requirements Review

2.1.1 Threads.getThreadStats

- Parameters
  - A set of posts returned by the Threads.queryThread function.
    - ***Passed***
    - The Parameter Exists
  - Action Keyword
    - Num
      - ***Passed***
      - Successfully retrieves count of entries
    - MemCount

· ***Passed***

· Successfully retrieves count of entries

* MaxDepth
  · ***Failed***

  · This function counts unique occurrences of parent IDs - it does not count the depth. Assumptions about the amount of unique parent IDs do not relate to the depth of the tree. For example, all threads on level 2 could have children, which will cause the children on level 3 to have unique parent IDs for every thread on level 2, but that amount of unique parent IDs do not relate to the level, as level 2 could have thousands of threads.
  · Currently, this function will return the largest amount of threads on a level, NOT the depth of the tree.
* AvgDepth
  · ***Failed***

  · The code used for this function is almost exactly the same as that of MaxDepth, and MaxDepth does not count the depth, thus AvgDepth will fail.

2.1.2 Get Thread Appraisal

- Parameters
  - A set of posts returned by Threads.queryThread function
    * ***Passed***
  - A set of members
    * ***Passed***
  - A set of appraisals
    * ***Passed***
  - Action Keyword
    * All
      · ***Passed***
    * Sum
      · ***Passed***
    * Avg

        - ***Passed***

    * Max
        - ***Passed***

    * Min
        - ***Partial Passed***

        - Assuming there is no appraisal value smaller than the assigned min value (100000)
        - ***Failed***

        - Because assumptions are made about the appraisal value the code does not cater for a situation where there is no appraisal value smaller than the min constant (bad code design)

    * Num
        - ***Failed***

        - Because in JavaScript, null is compared as null and not 0 (this isnt C++). Thus, no empty appraisals will ever be found, and then all records will be found as non-empty. Or, it will simply count all the appraisals whose value arent 0.

2.2 Export Thread Appraisal

- An external file is created
    - ***Passed***

- Containing data generated by the getThreadAppraisal function
    - ***Passed***

    - This data is passed as an argument (parameter threadObject)

- Only if the function executed with the All action keyword.
    - ***Failed***

    - This is never tested for, because they assume that the data is passed as an argument since its passed as an argument you will never be able to test if getThreadAppraisal is called with the All action keyword

2.2.1 Import Thread Appraisal

Data in an external file that was created using the exportThreadAppraisal function is used.

Partial Passed

The function do make use of a file. The file is sent as arguments in the importThreadAppraisal function (parameters are directory and fileName). So in this function they assume this is the file created by the exportThreadAppraisal function.

The data set is associated with only one member and only one specified appraisal.

Failed

The function never actually check if the data set is about only one member and only one specific appraisal. Therefore the data cannot be deemed as eligible for import.

A record contains all detail about the post along with a field that should contain an ordinal number that represents the levels of the specified appraisal.

Passed

The record does contain all detail about the post and a field that represents the leves of the specified appraisal which is the appraisalValue.

Edits to the data is ignored when importing a thread appraisal.

Failed

The function does not prevent edits to the data and there is not clear indicated of ignoring such edits.

For each record the assignAppraisalToPost function is applied.

Passed

The function does call assignAppraisalToPost for each record.

The appraisal level as stored in the file for each post is updated as an appraisal assigned by the member associated with the data set.

Failed

The function does not use the appraisal level to update the data set.

Check validity of member and appraisal.

Partial Passed

The function does check the validity of the member, however they have a dummy function that only returns true.
The function does check the validity of the appraisal, however no exception is thrown/raised it only returns true or false and there is also no means of catching an exception, that is the isValid function is not surrounded with try/catch blocks to catch exceptions and therefore the service delivery is not stopped when the appraisal is not valid.
The appraisal level is check for out of range, however no exception is thrown/raised when is it out of range and there the service delivery is not stopped if the appraisal level is out of range.

Full use case test rating

Partial Passed

There were 7 requirements listed for this use case.

- 2 out of 7 passed.
- 2 out of 7 partially passed.
- 3 out of 7 failed.

Hence, it is save to say the use case only partially passed due to the fact that most requirements was implemented, but there were a lot of requirements missing as well.

2.2.2 Export Thread

An external file is created.

Partial Passed

The function does create an external file, however :

- The data generated to by the queryThread function is assumed to be given as an argument to the function (parameter threadObject).
- The data is actually not correctly stored in the file since the threadObject is not parsed as a JSON-string. The object is directly stored in the file. So the actual data of the threadObject is not stored in the file.

Full use case test rating

Failed

This use case failed, since the contents of the file is not what is expected at all. (See point 2 above).

2.2.3 Import Thread

An external file that was created by the exportThread function is used to restore the content of a thread.

Passed

The function does use an external file. Again the file is passed as arguments to the function (parameters directory and fileName), so they are assuming this file was generated by the exportThread function before importThread is called with the file as arguments.

The content of the external file is used to restore content of a thread/post.

Failed

The function never tries to restore or even store the content of the external file on the Buzz Space.

A post is added if and only if it is not in the Buzz Space already.

Failed

The function never adds the content to the Buzz Space and therefore does not check if it is already in the Buzz Space.

Full use case test rating

Failed

There were 3 requirements listed for this use case.

- 1 out of 3 passed.
- 2 out of 3 failed.

This use case failed, since a thread was not restored or added to the Buzz Space which was the core functionality of this use case.

2.3 Architectual Requirements Review

2.3.1 Software quality requirements

Reliability: The switch statement's (getThreadAppraisal.js) default case was not specified. If an error occurred, it wouldn't be traceable since there is no error handling. If "garbage" values were passed through this switch statement, the error's result could be a system failure.

Maintainability: There is code duplication within the conditional statements. Many statistical operations exist and should someone wish to add them, it would be a daunting task. A function, instead of duplicated code, would simplify the modifications.
These two files (importThread.js and importThreadAppraisal.js) also have duplicate code, the effects are similar to the ones mentioned earlier.

Test-ability: Unit tests were included for each function. This reduces single-point-of-failure, because errors in smaller functions can be addressed without affecting the whole system.

Audit-ability: The file getThreadStats.js has a few entries for auditing (ParentID, Author, Timestamp, Content, Status). These are not adequate for auditing, especially since there's a single timestamp (which was ambiguous).

2.3.2 Architectural components

Framework(s): Node.js was used, as specified. Node-aop and Broadway plugin framework were included in "package.json" but they both appear unused.

Technologies: JSON (JavaScript Object Notation) was used to store "thread objects". HTML5 was used, however, this was for testing the low-level implementation. Electrolyte was included in "package.json" but there were no signs of it's use for the actual implementation.

3 Conclusion

3.1 Team A

The general consensus for this module was that there was poor error checking/handling. While most use cases work well and perform desired tasks, there are many instances where there might be an error that the user will not know about. This could result in the system failing or being non-responsive without the user having an explanation as to why this is the case. What little error checking did occur, they were console logged instead of thrown as an exception, again resulting in potential system failure without reporting the reason to the user.

Most software (non-functional) requirements were met, although reliability could be compromised because of the poor error handling/checking. Auditabilty isn't possible because of the absense of a tracking table in the database which was however beyond the scope of this module.

Technologies used were in line with the specification, with a few (mocha, unit.js electrolyte) not being used at all.

3.2 Team B

The biggest issue in this module were the assumptions made about data being used in operations. Simply using data and not testing validy could result in garbage values being used in operations and undesired output being produced. There were also some components of certain use cases that failed, making the use case not work as expected. This is obviously not ideal.

Maintainability is compromised by the un-readablity of the code, as well as code duplication in functions. Adding functionality would be difficult.

3.3 Overall

Overall, team A have created a better reporting module than team B. Team B have issues which could result in bigger errors than team A, and with Team B's module compromising the Maintainability requirement, it makes it the less desirable choice to fix for the bigger system.

Both modules need work however as neither is ready to be put in as part of a greater system. In a purely black and white sense they both failed to meet the full requirements of the specification. However it is recommended that Team A be the one chosen should there be an attempt to modify a module to use for the final system.