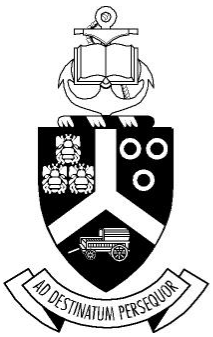

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Information Technology

COS 301 - Software Engineering

COS 301 - Mini Project

Author:	*Student number:*
Jason Richard Evans	u13032608
Abrie van Aardt	u13178840
Anrich van Schalkwyk	u13024427
Baruch Molefe	uXXXXXXXX
Jimmy Peleha	u12230830
Johannes Coetzee	u10693077
Liz Joseph	u10075268
Maret Stoffberg	u11071762
Sebastian Gerber	u12213749
Hugo Greyvenstein	u13019989

April 23, 2015

Threads Testing Report

Buzz Space Mini Project

Version: Version 0.1 Alpha For further references see gitHub
https://github.com/thepickpocket/ThreadsTesting
April 23, 2015

Contents

1 Company A

1.1 Functional Requirements.

SubmitPost

Test Status: *Partial Pass.*
Details: The purpose of SubmitPost is to allow a user to submit a post to a thread, creating a new child thread.
The implemented solution passes only partially, because while the new post is submitted, it does not successfully create it's own child thread.
It also has a mimeType, post content and dateTime fileds, which were required.
As this function is classified as critical and only partially works, the system does not meet all the requirements it should.

MarkPostAsRead

Test Status: *Complete Failure.*
Details: The purpose of MarkPostAsRead is to store a reading event which contains the information that a user has read a particular post at a particular time.
The implemented solution fails completely, posts are not marked as read and as this is the only requirement it fails.
As this function is classified as nice to have, it's failure does not prevent the rest of the system from working.

MoveThread

Test Status: *Partial Pass.*
Details: The main objective of the MoveThread function is to detach a sub-tree of thread nodes from one thread and add it to another thread.
The services contract MoveThreadRequest requires only one pre-condition, that the user is a administrator. The services contract was successfully implemented. This function only managed to pass partially. The MoveThread function depends on the functionality of the HideThread and UnHideThread functions. If these functions was not required during testing the thread could successfully be moved to another thread. One of the main requirements for all functions is that the function can be used on threads in the database. This is not true for this function as it only works locally. There is no functionality to retrieve the required threads from the database.

HideThread

Test Status: *Complete Failure.*
Details: The main objective of the HideThread function is that selected thread nodes and all its descendant nodes will be marked as hidden and user interfaces are meant not to render them.

The HideThread function was partially implemented. The services contract HideThreadRequest requires only one pre-condition, that the user is a administrator. The services contract was successfully implemented. The functionality of the HideThread function itself was not implemented. The priority of the HideThread function is stated as important thus the system will still function as it is not a critical function within the Thread module.

1.2 Architectural Requirements

The Threads module has the role of managing the threads of a buzz space in the Buzz system. This module has been made to be non-blocking so that multiple calls for services to this module will result in independent responses. This module is apart of the core modules used in the Buzz system.

The Threads module is implemented in Node(as in Node.js) in conjunction with the Node Package Manager(NPM). Node is by nature a non-blocking run-time environment using Java-Script. The installation of various packages from NPM allow for the different architectural requirements to be fulfilled.

Scope of responsibility

The Threads module encompasses a persistence infrastructure and a reporting infrastructure.

The persistent infrastructure that is used by this module is MongoDB. MongoDB is an open-source document database. The MongoDB database is accessed by

The reporting infrastructure is in the form functions or services that return/report the status of a request whether it be successful or not.

Quality requirements

The quality requirements addressed by this module are the following:

- Performance requirements
- Maintainability
- Testability
- Usability

The module is maintainable as the functional requirements for Threads are implemented with separation of concerns in mind and the code is well documented internally and well structured. The module is very testable and has unit tests performed on each of the exported functions. Because of the require function provided in node, it is easily usable, because the function prototypes correspond to the requirements specified in the specifications. The ones that are not addressed are:

- Scalability

The system module is not very scalable mainly because of the database management system, MongoDB. MongoDB is a light-weight database system and would not be suitable on a large scale. Otherwise the module would not have scalability issues. The missing callback functions prevent the functions from being scalable, or even cause them to not work correctly at all, because of the asynchronous nature of Javascript and evidently node.js.

Integration and access channels

Those who have access to this module are those that make use of the Buzz system. For example, a registered user, registered to a Buzz space, will have CRUD operations available to them.

Architectural constraints

The architectural constraints on this module would be the use of the Node run-time environment. There is no restriction on the database or the node modules used.

Technologies

- Node
- NPM
- Mongoose
- MongoDB

2 Company B

2.1 Functional Requirements

SubmitPost

Test Status: *Partial Pass.*
Details: The submit post use-case was divided into seperate smaller functions. One function *create(mUser, mParent, mPostType, mHeading, mContent, mMimeType)* is used to initialize the process of creating a new thread as well as its embedded post object. This function calls *createNewThread(mUser, mParent, mLevel, mPostType, mHeading, mContent, mMimeType, mSubject)* which parses the current thread and post object to JSON strings and then persists them in a remote database.
The problem however is that when child threads are created using the *createThread(...)* function call, the child thread is never persisted to the remote datase.

The functional requirements also state clearly that a thread (with its related post) should have been allocated a certain space, yet this functionality is never provided.

MarkPostAsRead

Test Status: *Complete Failure.*
Details: The teams approach to the reading events are not that well planned and clearly thought through. It simply sets a flag for the thread and persists that flag to the database. This however will then not be user based, as everyone who then accesses that specific thread from the database will have "read" it.
This function is hence not working as required and thus fails completely. It is however a nice-to-have function and does not break or affect any other part of the thread module.

Overview of Tests Results for Threads B- Functions MoveThread and HideThread

Tests log: The BuzzSpace software was tested on the nodejs test platform using the nodeunit environment located in the respective github repository for group ThreadB, from the 2015/04/20 to the 2015/04/24. The tests of the test phase (ref. software test plan) where executed.

Functions Tested: *moveThread(...) hideThread(...) unHideThread(...)*

Rationale for decision: After executing a test, the decision is defined according to the following rules:

- OK: The test result is compliant to the expected result.
- NOK: The test result differs from the expected result.
- Partial OK: There is partially compliant to the expected result.
- NOT RUN: The default state of a test sheet cannot be executed.
- NOT COMPLETED: The test sheet is set to "Not Completed" state when at least one step of the test is set "Not Run" state. ...

Overall assessment of tests: Overall assessment of tests:

- All tests did not provide an interface.
- All tests passed but with the use of different parameters were not supposed to pass.
- The tests respond to all kinds of input but they respond incorrectly.
- Tests achieve the desired input in the database.

Techniques used to test:

1. All tests did not provide an interface.
2. Code average- a check to see if all statement executed only once
3. Fault injection I placed invalid expectations of the result and an error was not thrown.
4. Software
5. Excercising inputs and checking the output for a differ in behavior

Statistics about tests:

- 0 percent of tests OK
- 0 percent of tests NOK
- 100 percent of tests POK
- 0 percent of tests NR
- 0 percent of tests NC

Test cases:

- Test cases planned = 4
- Test cases executed = 4
- Test cases passed = 0
- Test cases failed = 4

Give also statistics about bugs and enhancements:

- Total number = 4
- Number of Critical = 4
- Number of Major = 0
- Number of minor = 0
- Number of enhancements = 0

Conclusion:
The bugs specified were Mongoose related. (Error: Cast Error:Cast to ObjectId failed for value [object Object] at path_id)
In HideThread we changed the parameter in test.equal to a different expected result and it still passed so we dont think the function actually checks if the value field is hidden or not.
In MoveThread there are no error checks or bounds checks as we inserted an overbound value and it still passed the test.

CloseThread

Test Status: *Pass.*
Details:

The CloseThread function is designed to flag a thread as closed and then call a function to summarise the thread.
The function does pass unit testing but the structure and logic behind the function is somehwat flawed. Inside the function the mClosed flag is set to close to indicate the thread is now closed. A few lines later an if statement checks if the mClosed flag is set and then calls the createThreadSummary function. That if statement will always be true. Another flaw is that the createThreadSummary function returns a string containing the thread summary, but this string is never used within the closeThread function.

Conclusion: The function performs what it has to, it's just poorly coded.

queryThread

Test Status: *Partial Pass.*
Details:

The purpose of this function is to query the database for a specific thread according to various filters that has to supplied as parameters to the function. Although the function does pass unit tests it does not satisfy its functional requirements. The function does not use the database to query the threads schema. It uses recursion to traverse a tree of thread object, which will only be in memory as long as the Express application is running.

Conclusion:

Although the function does perform its functionality to a degree, it is not technically usable since the database is not being used. This means not results will ever be returned.

2.2 Architectural Requirements

The threads module is one of the most crucial modules in the Buzz system as it is the one through which users will actually interact with the system itself. It is, as outlined in the master specification responsible for functionality around threads and posts.

Since this system, and consiquently this module, will be used by potentially a huge amount of users, it is key that the Threads module be as scalable and as efficient as possible. With that said, here is an analysis of how the module performs as far as Architectural Requirements are concerned.

Access channels

Users are able to create, read, update and delete threads in accordance to their acquired status points. Users are responsible for their respective posts on each thread and this is the channel through which they will interact with the system. Other modules within the system use the same class functions to get certain information about the threads. For example, the system will *postPostToDatabase()* to carry information to the database.

Scope of Responsibilities

The module is responsible for providing an infrastucture that stores persistant domain data such as the posts within the threads. It is not responsible for the creation of an environment for process execution, however, it is responible for the guarantee that appropriate information is available for the module which is.

Quality requirements

The implementation of this module has a few flaws. This takes a toll on Testability and Reliability. However, the following quality requirements seem to be in place:

- Availability
- Maintainability
- Usability

Monitoribility and Auditability can also be achieved in this module but have the potential to be quite tedious since there are a lot of threads that will be created.

Archictectural constraints

This module succeeded in using the pre-specified technologies needed to implement the system. This is advantageous as it will make integrating the system as a whole a much simpler task without any tedious cross-compiling and coupling. The technologies used are:

Technologies

- Node
- NPM
- Mongoose
- MongoDB

3 Conclusion