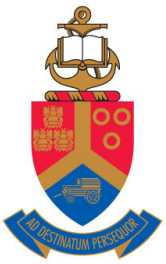

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

COS301 Mini Project Functional Testing - Space module

Group Members:

Xolieswa Ntshingila 13410378
Godfrey Mathe 13103394
Renette Ros 13007557
Byron Dinkelmann 11057638
Elzahn Botha 13033922
Name Surname 13070305
Lelethu Zazaza 13028023
Name Surname 12026442
Maria Qumayo 29461775
Jaco Bezuidenhout 11013878

Git repository link:

`https://github.com/u12026442/cos301_testing`

Version One

April 23, 2015

Contents

1 Functional Requirements

1.1 createBuzzSpace

The priority of creating a Buzz Space is a critical use case. Without it, lecturers may not create a Buzz space for a particular module they present during a particular year. It should create a root thread for the buzz space with an associated welcome post and assign the lecturer as administrator to that particular space.

1.1.1 Spaces A

Service Construct: To Identify the user
Pre-condition: BuzzSpace is active
Result: Failure
Discussion: There is no apparent evidence present in the getProfileForUser use case that indicates that a Buzz Space is tested for activness/inactivess. The use case is conducted under the assumption that the Buzz Space is active, this is in the violation of the service contract pre-condition.

Pre-condition: User is associated to the relevant Buzz Space
Result: Success
Discussion: The getProfileForUser use case conducts the query service by ensuring that the current user being searched is associated with the respective Buzz Space.

1.1.2 Spaces B

Pre-condition: Buzz Space is active
Result: Failure
Discussion: There is no apparent evidence present in the getProfileForUser use case that indicates that a Buzz Space is tested for activness/inactivess. The use case is conducted under the assumption that the Buzz Space is active, this is in the violation of the service contract pre-condition.

Pre-condition: User is associated to the relevant Buzz Space
Result: Success
Discussion: The getProfileForUser use case conducts the query service by ensuring that the current user being searched is associated with the respective Buzz Space.

1.2 getProfileForUser

The getProfileForUser is a critical use case in the Spaces module as it used to return the profile a user has on the Buzz Space.
The implementation of this use case is inclusive of a query service which returns the relevant profile.
Before the query is made there is a requirement that a Buzz Space exists and that a user is associated with a Buzz Space.
Thereafter the result from the query is returned with no other modifications to the Buzz Space.
Detailed below is an evaluation of Buzz Space A and Buzz Space B to determine whether or not they conform to the pre and post conditions required by the getProfileForUser service contract.

1.2.1 Spaces A

Pre-condition: BuzzSpace is active
Result: Failure
Discussion: There is no apparent evidence present in the getProfileForUser use case that indicates that a Buzz Space is tested for activness/inactivess. The use case is conducted under the assumption that the Buzz Space is active, this is in the violation of the service contract pre-condition.

Pre-condition: User is associated to the relevant Buzz Space
Result: Success
Discussion: The getProfileForUser use case conducts the query service by ensuring that the current user being searched is associated with the respective Buzz Space.

1.2.2 Spaces B

Pre-condition: Buzz Space is active
Result: Failure
Discussion: There is no apparent evidence present in the getProfileForUser use case that indicates that a Buzz Space is tested for activness/inactivess. The use case is conducted under the assumption that the Buzz Space is active, this is in the violation of the service contract pre-condition.

Pre-condition: User is associated to the relevant Buzz Space
Result: Success
Discussion: The getProfileForUser use case conducts the query service by ensuring that the current user being searched is associated with the respective Buzz Space.

1.3 closeBuzzSpace

Close BuzzSpace use case is important across the Buzz System. It simply sets the buzzSpace to inactive.

1.3.1 Spaces A

Pre-condition: BuzzSpace for current year and module exists
Result: Success
Comments: There exists proof that shows that a Space is tested for being active or not in the Moongoose Database.
Pre-condition: -User is authorised to create BuzzSpace
Result: Success
Comments: There exists proof that shows that a User is tested for being authorised to create a space.
Post-condition: BuzzSpace is not active
Result: Success
Comments: There exists proof that the isOpen field is set to false.

1.3.2 Spaces B

Pre-condition: BuzzSpace for current year and module exists
Result: Success
Comments: There exists proof that shows that a Space is tested for being active or not in the Database.
Pre-condition: User is authorised to create BuzzSpace
Result: Success
Comments: There exists proof that shows that a User is tested for being authorised to create a space.
Post-condition: BuzzSpace is not active
Result: Success
Comments: There exists proof that the isOpen field is set to false.

1.4 registerOnBuzzSpace

Creates a profile on the buzz-space for that specific user registering.

1.4.1 Spaces A

Pre-condition: User is registered for module to which the buzz space is associated
Result: Failure
Discussion: Even though the userID that is to be registered to the buzz-space is check to see if it is already registered for that specific space, it however does not check to ensure that the user has been registered for the module that the buzz-space is associated.

Pre-condition: Buzz space is active
Result: Failure
Discussion: While there is a condition that checks whether or not the the buzz space is active the condition always returns false even if the buzz space is open.

Post-condition: User profile is persisted to database
Result: Success
Discussion: The user profile appears to be persisted to the database as trying to register the same user profile to the database throws an exception in the form of "the space profile already exists". Indicating that the user profile was persisted in the first test case.

1.4.2 Spaces B

Pre-condition: User is registered for module to which buzz space is associated
Result: Failure
Discussion: Instead of checking whether or not the user is registered for the associated module it instead checks whether or not the user is an administrator to that buzz space. Only if the user is an administrator will the user be registered to that buzz space.

Pre-condition: Buzz space is active
Result: Success
Discussion: The function successfully checks whether or not the buzz space is active before continuing.

Post-condition: User profile is persisted to database
Result: Failure
Discussion: It appears to be the case that the function persists the user profile to the database, yet on further investigation of the database itself, there was no reflection that the user profile has been registered on the buzz-space. Instead, it seems that an administrator already in the database is attempting to be registered again back into the database.

2 Non-Functional Requirements

2.1 Spaces A

2.1.1 Maintainability

There is a lack of the use of development standards such as:

- structured programming
  - index.js: No good commented code for descriptions of functions.
  - index.js: Line 310-315: Commented out code used for debugging instead of checking if the development flag is set for the system as a whole.
  - routes/index.js: Line 302-304: Different indentations in code.
- recognizable nomenclature index.js: Line 111 vs 130: "submitNotify" is good recognizable nomenclature, where as in line 109 "submitCS" (refering to submit create space) are very confusing. Same in line 267,278,293,159

Programs have not been parameterized under necessary conditions to promote reusuability:

- index.js: Line 141-150: Mail Username and password are hard-coded. No settings file for the email settings.

2.1.2 Security

No trouble have been made to check if a user is logged in before adding a space, adding/removing users and admins. eg. routes/index.js: Lines 93-101

2.1.3 Auditability

Nothing are being written to log files. Also nothing are being sent to the database audit log.

2.1.4 Testability

Testability are made difficult because all the tests are commented lines in the app. (eg. index.js: Line 97-107) Modules like mocha, supertest and should could be used to test the unit without running the actual program.

2.1.5 Deployability

Very easy to deploy as an app. (Just use "npm install buzz-spaces"). No install script for Node and MongoDB so it should be installed beforehand. There could also be a setting.json file made to setup the app name, database path and DB name and mail server settings for easier deployment.

2.2 Spaces B

2.2.1 Maintainability

A maintainability issue we found in this module was related to the module's code documentation. The code was documented using the basic JSDoc structure, but it was incomplete:

1. Possible exceptions that could be thrown should have been documented for each function.
2. Data Types should have been specified for all function arguments using the correct JSDoc tags.

3. In the case of request objects the structure of these objects should have been documented using the correct JSDoc tags instead of just listing it in the description.

2.2.2 Testability

Some unit test existed for this module but they were incomplete and wrong.

1. Only the getProfileForUser unit test was actually executed, the other unit tests existed but was never called.

2. Some of the unit tests, test for an expected return value of null from asynchronous functions. This means the test will always succeed even if the function doesn't work.

2.2.3 Usability and Integrability

A usability and integrability problem identified in this module is that the functions throws strings when they have succeeded after an asynchronous call. This might cause the server to crash or show an error page. A better solution might have been to pass a callback argument to the function as is the normal JavaScript convention.

2.2.4 Deployability and Flexibility

This module was not written to use the Electrolyte dependency injection framework and it depends on another module, DatabaseStuff, for the database schemas and connection.

Due to the module directly requiring mongoose it will not work in an environment where a different persistence module is used.

2.3 Other issues

We did not identify issues regarding scalability, reliability and availability or performance in this module.

Security and audibility is not applicable to this module as it was the responsibility of the main integrated system.

3 Conclusion