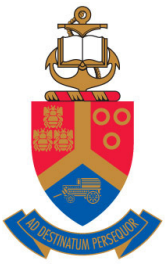

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

COS301 Mini Project Functional Testing - Space module

Group Members:

Xoliswa Ntshingila 13410378
Godfrey Mathe 13103394
Renette Ros 13007557
Byron Dinkelmann 11057638
Elzahn Botha 13033922
Esabel Nel 13070305
Lelethu Zazaza 13028023
Nicholas Robinson 12026442
Maria Qumayo 29461775
Jaco Bezuidenhout 11013878
Isabel Nel 13070305

Git repository link:

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

Final Version

April 24, 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.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 CreateBuzzSpace service contract.

Service Construct: To Identify the user who is requesting the creation and the module for which the buzz space is to be created. The system will assume that a buzz space for the current academic year is to be created.

1.1.1 Spaces A

Pre-condition 1: Module is active for current year.
Pre-condition 2: Upon cration, no buzz space should exist
Pre-condition 3: User must be a lecture for the spesific module
Result: Success on the first two pre-conditions. Failure on the last pre-condition.Exceptions are thrown if pre-condition is not met.

Post-condition 1: Buzz Space created for modle and Buzz Space is open.
Post-condition 2: Lecturer assigned as space administrator.
Post-condition 3: Profile for lecturer created.
Post-condition 4: root thread with welcome post created for Buzz Space.
Post-condition as per Spaces A: Buzz Space Created on success of pre-conditions.
Result: Success, on post-condition as per Spaces A - any authenticated user may create a Buzz Space.

Discussion: Users are not identified by their unique roles upon Buzz creation. Any user may create a Buzz Space as long as it doesn't exist .Upon a created Buzz space, threads may be created, active modules are retieved as well as user roles for modules.

1.1.2 Spaces B

Pre-condition 1: Module is active for current year.
Pre-condition 2: Upon cration, no buzz space should exist
Pre-condition 3: User must be a lecture for the spesific module
Result: Success on the first two pre-conditions. Failure on the last pre-condition.Exceptions are thrown if pre-condition is not met.

Post-condition 1: Buzz Space created for modle and Buzz Space is open.
Post-condition 2: Lecturer assigned as space administrator.
Post-condition 3: Profile for lecturer created.
Post-condition 4: root thread with welcome post created for Buzz Space.
Post-condition as per Spaces B: Buzz Space Created.
Result: Success on the post-condition as per Spaces B.

Discussion: A registered user can create a Buzz Space successfuley. Though after creation it lacks in most of the functionality, nameley thread creation, registering on the buzz space, getting active modules for the year, getting user roles for modules and assigning an administrative user.

1.2 getProfileForUser

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.getProfileForUser has a priority classified as critical. 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.

Service Construct: This is a simple query service which returns the pro
le the user has on the Buzz Space.

1.2.1 Spaces A

Pre-condition 1: BuzzSpace is active.
Pre-condition 2: User is associated to the relevant Buzz Space.
Result: Failure on the first pre-condition, sucess on the second pre-condition

Discussion: Pre-condition 1 - There is no apparent evidence present in the getProfile-ForUser 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 violation of the service contract pre-condition. Pre-condition 2 -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 1: Buzz Space is active.
Pre-condition 2: User is associated to the relevant Buzz Space.
Result: Failure on the first pre-condition, sucsess on the second pre-condition

Discussion: Pre-condition 1 - There is no apparent evidence present in the getProfile-ForUser 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 violation of the service contract pre-condition. Pre-condition 2 - 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, this is according to the classification of it's priority. It simply sets the Buzz Space to inactive.

Service Construct: The Service Contract of the coloseBuzzSpace is specified in the pre and post conditions of this module.

1.3.1 Spaces A

Pre-condition 1: BuzzSpace for current year and module exists.
Pre-condition 2: -User is authorised to create BuzzSpace .
Result: Success on pre-condition 1 and 2.

Discussion: Pre-condition 1 - There exists proof that shows that a Space is tested for being active or not in the Moongoose Database. Pre-condition 2 - 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.

Discussion: There exists proof that the isOpen field is set to false.

1.3.2 Spaces B

Pre-condition 1: BuzzSpace for current year and module exists.
Pre-condition 2: User is authorised to create BuzzSpace.
Result: Success on both pre-conditions.
Discussion: Pre- condition 1 - There exists proof that shows that a Space is tested for being active or not in the Database. Pre-condition 2 - 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.

Discussion: 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.

Service Construct: The Service Contract of the registerOnBuzzSpace is specified in the pre and post conditions of this module.

1.4.1 Spaces A

Pre-condition 1: User is registered for module to which the buzz space is associated.
Pre-condition 2: Buzz space is active.
Result: Failure on pre-condition 1 and 2.

Discussion: Pre-condition 1 - 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 2 - 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 1: User is registered for module to which buzz space is associated.
Pre-condition 2: Buzz space is active
Result: Failure on pre-condition 1, success on pre-condition 2.

Discussion: Pre- condition 1 - 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 2 - 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 comments on code or descriptions of functions were made.
  - 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 was 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 is being written to log files, and nothing is being sent to the database audit log.

2.1.4 Testability

Testability is made difficult because all the tests are commented out lines in the app. (eg. index.js: Line 97-107) Modules like mocha, supertest and should could hav 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 was provided 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:

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

- 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.

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

3.1 Conclusion for sapaces A

In conclusion, the spaces module that team A submitted was not a resounding success. Some of the functionality that they implemented can be used, but the module as a whole is not ready for deployment. The team didnt follow the standards that where requested of them meaning that the module cannot simply be plugged into the rest of the Buzz system. Some of the functionality was successful, however incomplete. For example, when Creating Buzz Space the Buzz Space is created, however there is little to no security that should monitor who can and cannot create the space, there is no trace of this action being kept, either in a log file or in the audit database and therefor no accountability. We see the same situation arise in the getProfileForUser and when closing a Buzz Space.

3.2 Conclusion for sapaces B

In conclusion, the spaces module that team B submitted was also not a resounding success. Again some of the functionality that they implemented can be used, but the module as a whole is not ready for deployment. All though the team did follow a lot of the standards that where requested of them meaning that the module can simply be plugged into the rest of the Buzz system, there was very little provided by the team to help with future maintenance of the system. Some of the functionality was successful, however it to was incomplete. Taking the same example, when Creating Buzz Space the Buzz Space is created, however there is little to no security that should monitor who can and cannot create the space, there is no trace of this action being kept, either in a log file or in the audit database and therefor no accountability. We see the same situation arise in the getProfileForUser. However, when it comes to closing a Buzz Space the functional contract was fully adhered to because everything that is required of the function is performed. However, the same issues with the auditability, security and maintainability we present.