COS301 Mini Project
Testing

Notification

Johan Marx 12105202
Tim Kirker 11152402
Edwin Fullard 12048675
Matthew Russell 12047822
Jessica Lessev 13049136
Hanrich Potgieter 12287343
Roger Tavares 10167324
Thinus Naude 13019602
Paul Engelke 13093500
Trevor Austin 11310856

Git repository link:

`https://github.com/DianMarx/notificationTesting`

Version April 23, 2015

Contents

1 Introduction

This document was compiled by our group and was produced as a whole by the team.

We were tasked with testing the two Buzz Notification modules. The functionality provided by the buzzNotification module should be focused on a user registering to receive a notification messages.

2 What we tested

Our testing involved looking at the two notification modules and comparing it to the specification. The use cases we will be evaluated can be seen in figure 3

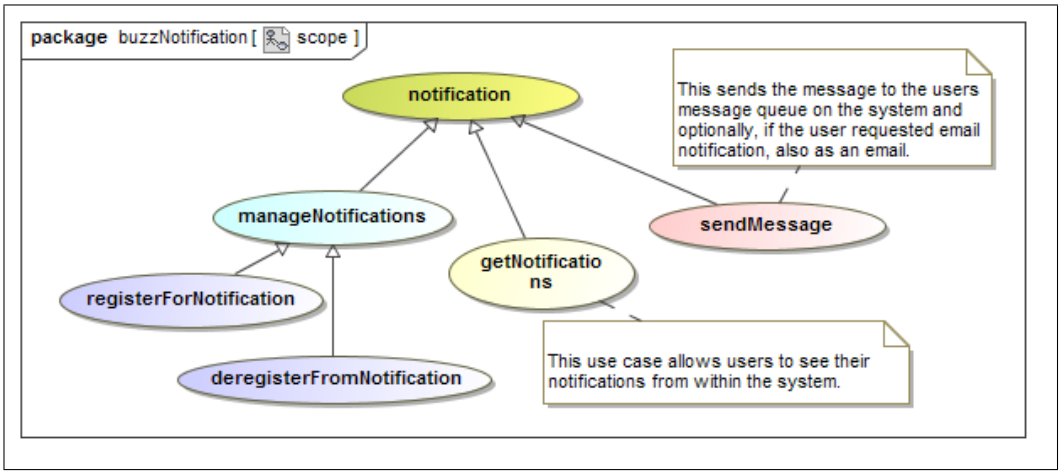


Figure 1: The scope of the buzzNotification module.

3 Functional Testing

We tested each use case and made conclusion based on what we found. For all use cases of Buzz Notification A and B we will either that the use case a success or we will provide a list of violations of the contract requirements (e.g. pre- and post-condition violations or data structure requirement violations)

3.1 Register for Notification Use Case

We found the following problems with regards to the "Register for Notification" use case of the module.

Notification A

The notification module for team A had the following violations

1. Pre condition violations:
    - Team A provided no checks for pre-conditions.
2. Data structure requirement violations:
    - None found.

Notification B

The notification module for team B had the following violations

1. Pre condition violations:
    - None found.
2. Data structure requirement violations:
    - None found.

Remarks

Team A provided no checks for pre-conditions before writing data to the database, in every case the function returns with a value of true, making it appear to succeed even when there was no data written to the database at all, and thus no registration actually happened.
Team B had no pre-condition violations. Any data that violated pre-conditions was correctly checked for and handled, registration of the user is only made if all the pre-conditions are successfully met.

3.2 Deregister for Notification Use Case

We found the following problems with regards to the deregisterFromNotification use case of the module.

Notification A

The notification module for team A had the following violations

1. Pre conditions:

    - Check that the user is registered.

      The function attempts to remove a record from the database and an error is thrown from the database. If error is null then it shows that such a record exists thus the user was registered. If the error is not null then the record did not exist and thus the user is not registered to notification. Thus the pre-condition is checked for but not in the best way possible.

      Thus no pre-condition violation.

2. Post conditions:

    - The function returns based on success or failure.

      This post-condition is violated as the function returns true as long as the function terminates. Thus the function return is not an indication as to whether the execution was a success or failure.

    - The user is de-registered from notification. By making changes to database.

      The function removes the record corresponding to the input parameters. Thus the post-condition is not violated.

3. Data structure requirement violations:

    - The data structure is violated in the sense that there is no return object. Only a set boolean value is returned.

Notification B

The notification module for team B had the following violations

1. Pre condition violations:

    - One

- Two
- Three
- Four

2. Data structure requirement violations:
    - One
    - Two
    - Three
    - Four

Remarks

Neither of the teams did what was required in the functional requirements. Some functionality reflected what was implied ... bla bla blas write something here if you deem it necessary

3.3 Send message

4 Non-functional testing/assessment

Problems regarding performance, scalability, maintainability, reliability, usability as well as evidence/proof of said problems.

4.1 Performance

4.2 Scalability

In regards to scalability of the notification module for both A and B the system should have had the ability to work for all Computer Science modules at a University of the size of the University of Pretoria. Each teams module is discussed below

Notification A

The notification A module will be scalable for any size as it is a proper NodeJS module, meaning one could launch multiple instances over a cluster of servers without any conflict or problem.

A problem could however arise i.t.o. scalability as the module does not cater for rapid switching of the database it uses. This could lead to problems down the line where you would need to manually intervene if you would want to support more then once database connection/location.

Notification B

The notification B module will not be scalable as it is an express application and as such wont be able to be included into another NodeJS application. This is because the functionality of the notification system is intertwined with a Express web server code. In their main file at the top they have the following code:

```
var express = require('express'),
app = express();
```

This will result in each inclusion of the notification module having a full blown express application. It would be a extreme waste of system resources to scale this module.

Remarks

Neither of the teams did what was required fully by the architectural requirements but Notification A does allow for scalability. There were no ma-

jor problems or concerns in regard to Notification A unlike notifications B which would not be scalable.

4.3 Plug-ability (Maintainability)

As defined in the Architectural Requirements Document all modules should be designed and developed in such a way that they are modular allowing for sub modules to be added or even removed during development.

This allows for a more maintainable system as Diagnostics of potential flaws and function clashes can now be eliminated as the system can be tested as individual parts or even as a whole allowing for a more stable module.

Team A and B's modules will each be discussed below with regards to this non functional requirement:

Notification A

- The initial problem with Notification A is that all the code is part of the same Java Script file suggesting that the whole module is dependent on itself and if one function is providing the wrong functionality the whole module will fail or produce incorrect output.
- Opening the Java Script file one can immediately see that this module is not made of sub modules and testing is done by commenting out of code. This can produce many errors and is hard to test since you don't know which line of code is producing errors or is providing incorrect results.

Notification B

- Notification B has sub-dived the whole notification module into smaller more manageable sub modules allowing for individual testing.
- Each sub module can then be altered and tested before integrating into the final module allowing the system to be maintained at a constant stable version. New functionality can now be added via new submodules allowing for plug ability.

Remarks

Notification A did not meet the non functional requirement maintainability as their code is not modular while on the other hand Notification B made an excellent job to ensure that their code remains as modular as possible.

4.4 Reliability

Notification A

Below are some of the issues that could be identified in the module provided by Notifications A, in terms of reliability.

- **Function Return Values:**
  All functions available in the module's API should return some indication of success. However, in the case of this module, all the functions that return a value, only return true (proof of which can be found on lines 137, 169, 215, 260, 305). This means that regardless of what the outcome of the function's execution is, the caller will only perceive it as having succeeded — which is an issue of reliability: if an error occurs and needs to be handled by the system, nothing can be done because the function does not notify the system of erroneous execution.

- **Callback Usage:**
  A follow-on of the previous point is the usage of callbacks. No provision has been made for the use of callback functions and therefore any information provided by asynchronous execution cannot be conveyed to the caller. So any return value, regardless of whether it may be true or false, cannot be trusted as the asynchronous part of the function will most likely not have been completed before it returns to the caller, i.e. the return value is unreliable.
  Below is a listing of the function signatures that require callbacks, but do not make provision for them:

```
line 94  : notification.notifyRegistration(jsonObject)
line 140 : notification.notifyDeregistration(jsonObject)
line 172 : notification.notifyNewPost(jsonObject)
line 218 : notification.notifyDeletedThread(jsonObject)
line 263 : notification.notifyMovedThread(jsonObject)
line 308 : notification.appraisalRegister(jsonObject)
line 321 : notification.appraisalDeregister(jsonObject)
line 336 : notification.appraisalNotify(jsonObject)
line 390 : notification.sendNotification(jsonObject)
```

- **Logging to Console:**
  In the cases where errors occur and in some cases where successful execution occurs, information about the success or failure is lost to the system as it is only logged to the console(not reliable as a persistence tool) instead of being returned in some form that the caller can interpret. This is another issue of reliability as the system cannot determine whether or not a user will now receive notifications or if a user has been deregistered from receiving notifications, etc. An example of the above can be found starting on line 321:

```
notification.appraisalDeregister = function (jsonObj) {
    appraisals.remove({
        notification_StudentID: jsonObj.studentID,
        notification_AppraisalType: jsonObj.appraisalType
    }, function (err) {
       if (err != null)
         console.log(err);
       else
         console.log("user has been removed from table");
    });
};
```

Notification B

Below are discussed some of the issues identified, in terms of reliability, for the Notifications B module.

- **Function Return Values:**
  In many cases, no values are returned at all, which forces the system to assume that the call executed correctly. This causes the system to be unreliable as there is no guarantee that the function call succeeded or not. An example of this is shown below and is taken from Appraisal-NotifyMe.js (lines 37 - 58):

```
function addAppraisalToDB(details)
{
  Subscription.findOne(
  {
    'user_id':details.post_user_id

  },function(err,docs){...});
}
```

  Some code has been removed from the above, as not to clutter the document with redundant code, and has been replaced with '...'.

- **Callback Usage**
  In most cases, like Notifications A, no provision has been made for callback functions to asynchronous function calls. This results in no means of obtaining reliable result information from the functions. Below is a list of the files containing the functions that fail to make provision for callback functions where they are needed:

```
AppraisalNotifyMe.js
DailyNotif.JS
DeleteNotif.js
StandardNotification.js
```

- **Logging to Console:**
  As with Notifications A, information that should be returned to the caller is logged to the console where it will be lost and not serve any useful purpose. An example of this is as follows, taken from DeleteNotif.js (lines 159 - 170):

```
newNotif.save(function(err,newNotif)
{
  if (err)
  {
    success = false;
    console.log("Error Adding Notification ");
  }
  else
  {
    success = true;
  }
});
```

  Reliability might be improved by incorporating logged information in a result object (as specified in the specification) so that the system can be informed if any issues arise and thus provide a reliable service.

Remarks

A number of problems pertaining to reliability have been identified across both modules. And in comparison, these issues are the same in both cases. It would be highly recommended that these issues be addressed accordingly as to improve the reliability of the module and thus the system as a whole.

4.5 Integrability

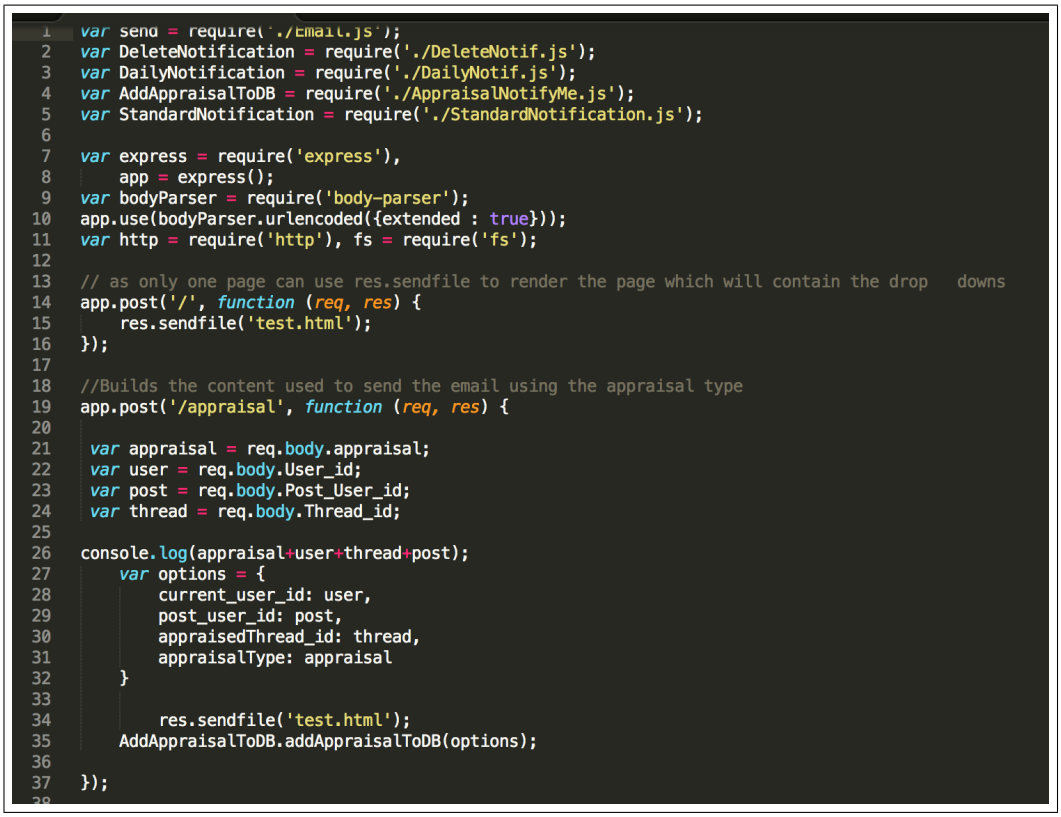
subsubsection*Integratability A We tested the Integratability of the following fucntions. These functions has been supplied by group A

- Installing required Packages There is no way of installing the required packages.
- Dependancy Injection There is no dependacy injection.
- Unit tests There was no supplied unit test. Therefore we cannot test if the functions is working.

Integratability B

The integratability of notification B is not adequite. There are numerbour challanges that are missing. There is no dependancy injection and all packages needs to be installed manually.

- File Structure Each function is placed in a seperate file. There is no common module to integrate that will allow access to all the capabilities of notifications.
- Installing required packages One cannot easily install the dependancys that is required by Notifications.
- Dependancy Injection There is no dependacy injection.
- Inappropriate Unit testing. Therefore we cannot test if the functions is working. There is a file called test.js that is an attempt at unit testing but no proper unit testing was applied. They should have used someting similar to Unit.js with mocha. **??**



```
var send = require('./email.js');
var DeleteNotification = require('./DeleteNotif.js');
var DailyNotification = require('./DailyNotif.js');
var AddAppraisalToDB = require('./AppraisalNotifyMe.js');
var StandardNotification = require('./StandardNotification.js');

var express = require('express'),
    app = express();
var bodyParser = require('body-parser');
app.use(bodyParser.urlencoded({extended : true}));
var http = require('http'), fs = require('fs');

// as only one page can use res.sendfile to render the page which will contain the drop   downs
app.post('/', function (req, res) {
    res.sendfile('test.html');
});

//Builds the content used to send the email using the appraisal type
app.post('/appraisal', function (req, res) {

 var appraisal = req.body.appraisal;
 var user = req.body.User_id;
 var post = req.body.Post_User_id;
 var thread = req.body.Thread_id;

console.log(appraisal+user+thread+post);
    var options = {
        current_user_id: user,
        post_user_id: post,
        appraisedThread_id: thread,
        appraisalType: appraisal
    }

        res.sendfile('test.html');
    AddAppraisalToDB.addAppraisalToDB(options);

});
```

Figure 2: Unit Tests

- database issues There is no way to access the specified database and this also contributes to the integratability. They sould supply a way to specifiy the database to be used. **??**

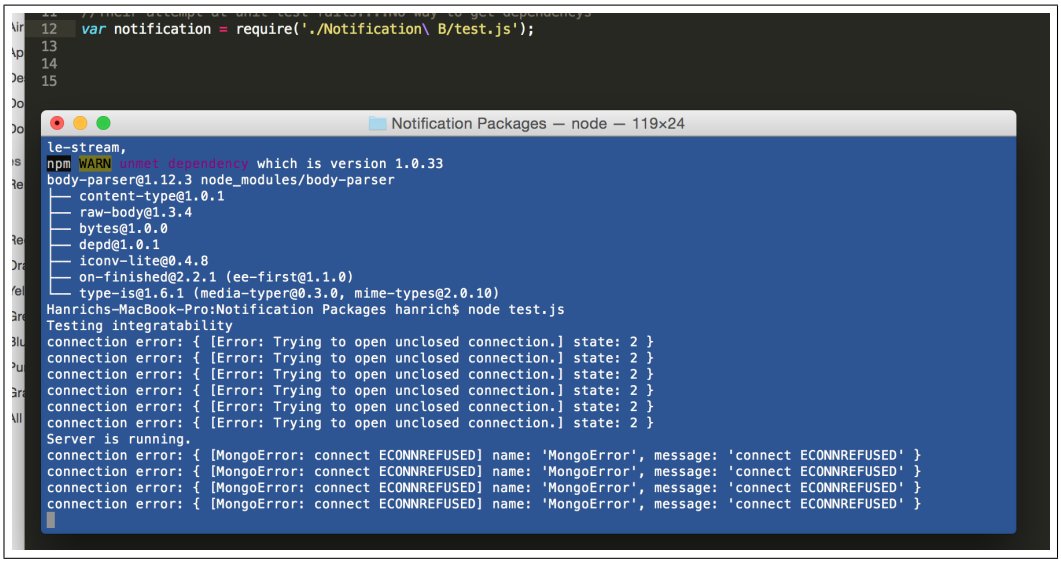


Figure 3: Database connection issues

Remarks

Notification B is not Integratable at alll. There is no provision for dependacny injection.

4.6 Usability