UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

COS 301 MINI PROJECT
Top Level Integration Testing

Elana Kuun u12029522
Hlavutelo Maluleke u12318109
Estian Rosslee u12223426
David Breetzke u12056503
Sylvester Mpungane u11241617
Phethile Mkhabela u12097561
Renaldo van Dyk u12204359
Antonia Michael u13014171
Herman Keuris u13037618
Jaco-Louis Kruger u13025105

Github Repository

April 2015

Contents

1 Introduction

This document contains the findings of the various activities performed as per testing of the Buzz Space system top level integration.

2 Testing Results

2.1 Top Level A Testing Results

2.1.1 Functional Testing

1. Buzz-Spaces Module

- Use Cases
  - createBuzzSpace
    This is the service which enables a lecturer to create a Buzz space for a particular module they present.
    * Pre-conditions:
      -buzzSpaceExists (should not be able to create a duplicate buzz space) (implemented, was unable to create a space that already existed)
      -moduleNotActive (for the current year) (not implemented, was able to create a space for a non existent module)
      -notAuthorized (only an authorized user can create a buzz space) (not implemented, was able to create a space without signing in as a lecturer)

    * Post-conditions:
      -storeBuzzSpace (persist the new buzz space)(implemented, new buzz space was reflected in system once user is returned to home page)
      -lecturer registered on the buzz space (not implemented)
      -lecturer assigned as administrator of the buzz space (not implemented)
      -Create welcome message as root thread for the buzz space (not implemented)

2.1.2 Non-functional Testing

1. Usability:

   The system is usable because firstly the necessary actions that a user can take appear in a navigation bar at the top of the screen. Thus it is easy for a novice user to be able to know what to click on and navigate the website.

   The interface is not cluttered, and only basic functionality is displayed on the home screen, making the system more learnable. The buttons are labelled with text rather than with graphical icons, and the text on the button is quite explanatory, which makes their purpose more clear.

   Larger headings are used to label the different sections, for example under the Manage Constraints tab there are large headings to indicate the Existing constraints section and the Add new constraint section. This again contributes to ease of use for the novice first year user.

   Through these mechanisms the system is memborable hence it is also be understandable.

2. Integratibility:

   The system is able to address future integration requirements by providing access to its services using widely adopted public standards such as firstly having separate npm packages for all the different modules. The packages are stored on Synopia. Also, electrolyte is being used in the server to provide a dependency injection. The HandleBars server is the main server that needs to be used to test and integrate all the modules on. The routes/index.js, routes/infrastructure.js and routes/content.js files use express to route the different hbs files for the different modules in order to integrate the infrastructure and content subsystems into the main system.

   A separate file is used to establish the connection to the database to avoid having this done in all the separate files. Also, global variables are now used such as the global password and username for example.

   A document has been provided via email and a README file has been provided to specify important standards and regulations that must be followed.

The functional code in the separate packages must be placed in an exportable function taking parameters such as the database or settings, and this is done as part of the electrolyte dependency injection.

Exports are also used in the different files to make the code accessible to the other files.

3. Deployability:

   The system is deployable on Linux servers as we have run it using Ubuntu 14.04 Linux and the system was able to run. The following screenshot shows the system running on a Linux server:

   The system is deployable on an environment using different databases for persistence of the Buzz database because the Handlebars server contains a folder called node_modules, and it contains the buzz_database package. This package can easily be swapped out and an alternative database package can be plugged in, that the system can use due to the flexibility of this server. As long as the new package has the same name so that the files that require the database do not need to be changed, no major changes will need to be made. //screenshot

   The system is deployable in environments where the user authentication credentials and roles are sourced from different repositories. The following screenshot indicates that the Handlebars server contains a folder called node_modules, and it contains the buzz_csds package. //screenshot

   This package can easily be swapped out and an alternative data source package with different credentials and roles can be used, due to the pluggability of the system and dependency injection employed by the top level team. The new package must just be named the same as the old one to avoid errors where the file is required from the package in the code.

2.2 Top Level B Testing Results

2.2.1 Functional Testing

1. Buzz Resources Module

   Description

The system suppose to integrate a resource module that is used to upload and manage resources like media files and documents that can either linked or embedded to a post.

Preconditions

- Size constraints must be met
- The resource mimetype must be detected
- The resource to must be supported

Post-conditions

- Url for resource must be generated
- Resource must persist

Test Results

The Buzz resources module works when test in isolation through unit testing but fails on automated integration testing, it was not integrated as required by system specification.

2. Buzz Notification Module

Description This module is focused on registering to receive notification messages for submitted threads and post of a particular user and any notification one receive for their posts via email.

Preconditions

- User must have registered to receive notifications

Post-conditions

- User should receive all notification based on the domain (receive from all threads or a particular thread) the user registered to receive notifications.

Test Results

The Buzz notification was not integrated for this system, no email notifications are sent when a thread or post is submitted.

2.2.2 Non-functional Testing

1. Maintainability:

   The Buzz(B) system is maintainable in the following aspects:

   - Understandable by future developers:

     Every subsystem or 'module' of the system is implemented as a seperate functioning module. Standalone modules are then plugged into the system. New developers can change modules independantly.

   - Technologies used is current and expected to be available for a long time:

     Node.js makes up the core of the buzz system. It is a widely used open-source platform released in Q1 of 2009. Popularity has grown tremendously and is not expected to stop growing in the near future.

     MongoDB is used as a database structure of the Buzz system and is highly compatible with Node.js. Also released in 2009 and is now the fourth most popular type of database management system. MongoDB is expected to grow even more in the coming years which makes it a good choice for the Buzz system in terms of maintainability.

   - The system is easily updated

     New functionality can be added by adding modules to the system. Functionality can be changed by changing independat modules without having to rebuild the whole system.

2. Scalability

   The Buzz(B) system is partially scalable in the following aspects:

   - Able to host buzz spaces for all Computer Science modules:

     The Buzz(B) system failed this scalability test. It does not support functionality of adding a new buzz space thus cannot host buzz spaces fot all CS modules.

   - University-wide, servicing in the order of 50 000 students:

The Buzz(B) system failed this scalability test. It does not support functionality of adding a new user or student. Thus one default user exists as apposed to 50 000.

3. Auditability:

   The Buzz Space system is partially auditable, only the requests for the user serves of the system are logged. The logged request only contains:

   - User serves requested.
   - Request object stringified as JSON

   There are no log responses that are provided by the system, and the serves required to extract information from the audit logs is also not available not provided.

4. Testability:

   The Buzz system is divided into manageable components (modules) for each use case. Breaking the system into these components makes make the system testable through unit testing using mock objects. The Buzz system is a modular system, the system components are pluggable and makes automated integration testing simple as the component is integrated in the entire system.

5. Performance requirements:

   Requirements for the Buzz system were for non-reporting operations to preferably not last longer than 0.2 seconds and for report queries to be processed in no more than 5 seconds. To test this we used Firebug, an open source web browser extension of Firefox used for webpage monitoring. (Version used: Firebug 2.0.7). The Buzz system met these requirement:

   - Non-reporting operations such as returning to a previous page (i.e. operations that relied on cached copies of a webpage) completed relatively close to the target time limit (e.g. returning to the main Buzz Spaces page after viewing the user's profile took only 0.286 seconds, very close to the target time of 0.2 seconds, Figure 1).

- Report query operations (i.e. which had to communicate and receive some information from the system's database) also completed within their desired time limit of 5 seconds (e.g. accessing the test module "TEST443"'s discussion space took only 1.24 seconds, Figure 2). Even some of the report query operations which took comparatively long to complete stayed within the 5 second limit (e.g. accessing module "COS332" took 4,81 seconds and accessing the Admin tab took 4,96 seconds).

6. Reliability and availability:

   Availability: The Buzz system can be accessed at the following URL (http://buzz-codechat.rhcloud.com/) at any time of day on any of the widely used web browsers (Google Chrome, Mozilla Firefox, Microsoft Internet Explorer, Safari and Opera). The Buzz system even functions and displays correctly on the web browsers of mobile devices to ensure maximum availability. Reliability: The system performs various error checks to ensure that no user action could cause the Buzz system to fail/crash (e.g. returning error messages if the user tries to log on with incorrect information , Figure 3). Even in cases were the system's functionaity was not fully implemented procedures are in place to rather warn the user of incomplete operations instead of blindly carrying out a harmful request (e.g. an error message is diplayed when a user tries to upload a new profile picture, Figure4). The reason why the Buzz system can still function (to an extent) even though much of its functional procedures are incomplete/missing is due to the modular fashion in which the system was created. Even if one module of the system fails it is appropriatley seperated from the other modules to prevent a complete system failure.