Buzz Discussion Board
Requirements and Design Specifications
(version 0.1)

Fritz Solms, Vreda Pieterse, Stacey Omeleze & Lecton Ramasila
Dept Computer Science, University of Pretoria

March 16, 2015
Contents

1 Vision and scope .................................................. 2
  1.1 Project background ............................................. 2
  1.2 Project vision .................................................. 2
  1.3 Project scope .................................................. 3

2 Application requirements and design ....................... 3
  2.1 Modular System ............................................... 3
  2.2 The Buzz-Authorization module .............................. 4
    2.2.1 Scope .................................................... 4
    2.2.2 Use cases ................................................ 5
      2.2.2.1 addAuthorizationRestriction — priority: important .... 5
        2.2.2.1.1 Service contract ...................................... 5
      2.2.2.2 isAuthorized — priority: important .................... 6
        2.2.2.2.1 Service contract ...................................... 6
      2.2.2.3 removeAuthorizationRestriction — priority: important ... 7
        2.2.2.3.1 Service contract ...................................... 7
      2.2.2.4 getAuthorizationRestrictions — priority: important .... 8
        2.2.2.4.1 Service contract ...................................... 9
      2.2.2.5 updateAuthorizationRestriction — priority: important ... 9
        2.2.2.5.1 Service contract ...................................... 9
    2.2.3 Domain model .............................................. 10
  2.3 The Buzz-Spaces module ..................................... 11
    2.3.1 Scope .................................................... 11
    2.3.2 Use cases ................................................ 11
      2.3.2.1 Login and administrative user ......................... 11
        2.3.2.1.1 Service contract ...................................... 11
        2.3.2.1.2 Functional requirements ............................... 12
        2.3.2.1.3 Process specification ................................ 13
      2.3.2.2 buzzSpace.createBuzzSpace — priority:critical ...... 13
        2.3.2.2.1 Service contract ...................................... 13
        2.3.2.2.2 Functional requirements ............................... 14
        2.3.2.2.3 Process design ....................................... 15
      2.3.2.3 Close buzz space — priority:important ................ 16
        2.3.2.3.1 Service contract ...................................... 16
    2.3.4 buzzSpace.registerOnBuzzSpace — priority:critical ... 17
      2.3.4.1 Service contract ...................................... 17
    2.3.5 getProfileForUser — priority:critical .................. 18
      2.3.5.1 Service contract ...................................... 18
    2.3.3 Domain model .............................................. 18
  2.4 The Buzz-Data-Sources module ............................. 19
    2.4.1 Scope .................................................... 19
    2.4.2 Use cases ................................................ 19
      2.4.2.1 Login and administrative user ......................... 19
        2.4.2.1.1 Service contract ...................................... 19
        2.4.2.1.2 Functional requirements ............................... 20
2.4.2.1.3 Process specification ........................................... 21
2.4.2.2 getUsersRolesForModule ....................................... 21
2.4.2.2.1 Service contract .......................................... 21
2.4.2.3 getUsersWithRole .............................................. 22
2.4.2.3.1 Service contract .......................................... 22
2.4.3 External database structures ................................... 22
2.4.4 Domain model ..................................................... 24

2.5 The buzzResourcesModule ........................................ 24
2.5.1 Scope ..................................................................... 24
2.5.2 Use cases ............................................................. 24
2.5.2.1 uploadResource ................................................... 24
2.5.2.1.1 Services contract ........................................... 25
2.5.2.1.2 Functional requirements ..................................... 25
2.5.2.1.3 Process design ............................................... 25
2.5.3 Domain model ...................................................... 27

2.6 Threads ................................................................. 27
2.6.1 Scope ..................................................................... 27
2.6.2 Use cases ............................................................. 28
2.6.2.1 submitPost — priority:critical ................................ 28
2.6.2.1.1 Services contract ........................................... 28
2.6.2.2 markPostAsRead — priority:niceToHave .................. 28
2.6.2.2.1 Services contract ........................................... 29
2.6.2.3 closeThread — priority:important ........................... 30
2.6.2.3.1 Services contract ........................................... 30
2.6.2.3.2 Thread summarizers .......................................... 31
2.6.2.3.3 Functional requirements ..................................... 32
2.6.2.3.4 Process design ............................................... 33
2.6.2.4 moveThread — priority:important ........................... 34
2.6.2.5 hideThread — priority:important ............................ 34
2.6.2.6 Threads.queryThread — priority:medium .................. 34
2.6.2.6.1 Services contract ........................................... 34
2.6.2.6.2 Functional requirements ..................................... 35
2.6.3 Domain model ...................................................... 35

2.7 The Buzz-Status module ............................................. 36
2.7.1 Scope ..................................................................... 36
2.7.2 Use-cases ............................................................. 37
2.7.2.1 assessProfile ..................................................... 37
2.7.2.2 setStatusCalculator ............................................ 38
2.7.2.2.1 Services contract ........................................... 38
2.7.2.3 getStatusForProfile ............................................. 39
2.7.2.3.1 Services contract ........................................... 39
2.7.2.4 createAppraisalType ............................................ 40
2.7.2.5 activateAppraisalType ......................................... 41
2.7.2.6 assignAppraisalToPost ......................................... 42
2.7.2.6.1 Service contract ........................................... 43
2.7.3 Domain model ...................................................... 44
4.4.18 jsreport reporting framework ........................................ 64
4.5 Infrastructure ......................................................... 64
4.6 Development architecture ........................................... 66
  4.6.1 Version control ................................................. 66
  4.6.2 IDE .............................................................. 66
  4.6.3 Builds ............................................................ 66
  4.6.4 Unit testing ...................................................... 67
  4.6.5 Integration testing ............................................. 67
  4.6.6 Documentation ................................................ 68
  4.6.7 Bug tracking ..................................................... 68
1 Vision and scope

1.1 Project background

Students can provide a lot of knowledge to other students and can learn a lot from another. However, at the University where there are naturally large student numbers, it is often difficult to establish contact and get to know fellow students and to remain in contact with them. Often one student has a problem where other’s could and would help, but the communication channels for this are simply not there.

The University would like an application which can enrich the student experience, by providing an environment where students of a module can get to know one another, request assistance from the class, contribute to the class and appreciate each other’s contributions. It should also enable teaching staff to pose challenges and provide feedback to the students.

1.2 Project vision

Buzz space is aimed to provide an observable communication space for University modules where students can raise questions, share knowledge add comments and mark up each other’s contributions. It should also enable students to follow threads of conversation and be notified of any events occurring in Buzz space.

The proposed system is a discussion board which is to be integrated into the department’s web site and ultimately also with the Hamster marking system. It should enable lecturers to set up a discussion board for a module which provides an observable communication infrastructure for a course can be accessed by the students and teaching staff for that module.
### 1.3 Project scope

The core of the system is a discussion board which is enriched with functionality for tagging, appraisals and status building, notifications, post moderation and reporting. The high level modules and their responsibilities are shown in Figure 1.

![Figure 1: High level system modules and their responsibilities.](image)

The core of the system is a discussion board which is enriched with functionality for tagging, appraisals and status building, notifications, post moderation and reporting. The high level modules and their responsibilities are shown in Figure 1.

### 2 Application requirements and design

This section discusses for each module of the Buzz Discussion Board the functional requirements as well as the process designs for the use cases and the domain models which must be maintained within persistent storage (databases) by that module.

#### 2.1 Modular System

The system is to be a modular system which allows for

- only a subset of modules to be deployed – minimally the system will require the core modules to be deployed.
- further modules to be added at a later stage.

To this end there should be
• minimal dependencies between modules, and
• no dependencies of core modules on any add-on modules.

For example, there is no service on BuzzSpace which sets a ProfileAssessor as statusCalculator for that BuzzSpace as that would directly introduce a dependency between the core buzzSpaces module and the add-on buzzStatus module. Instead there is a statusCalculatorAssignment which maintains a reference to both the ProfileAssessor and the BuzzSpace for which the ProfileAssessor is assigned to be the statusCalculator.

2.2 The Buzz-Authorization module

All service requests in Buzz, irrespective of the channel through which they are requested, are to be intercepted by an AuthorizationInterceptor which checks whether the user is authorized to use the service. The interceptor raises a NotAuthorized exception if the user is not authorized to use a particular service.

Nearly all use case specifications have thus no separate authorization specifications and hence also no actor (user role). The only two exceptions are

• the createBuzzSpace addAdministrator, and removeAdministrator use cases which can be accessed only by a user of the CS systems who is assigned to be the lecturer for the module for the year for which a buzz space is created or for which an administrator is assigned or removed, and

• the assignAuthorizationInterceptor and removeAuthorizationInterceptor use cases which can be accessed only by users who have been assigned to be administrators for that space.

The access rules for all other services are fully configurable (by space administrators) from this authorization module. The access to any of the services except the ones discussed above is solely determined by authorization interceptors. If no authorization interceptor covers a particular service, then access is open, i.e. anyone can access the service.

An authorization interceptor can restrict access to one or more system services based on

• the role a user has within a particular module, and
• the status the user has earned on the buzz space for that module.

In addition to being able to configure access rules and to enforce these through interception, the module also enables access channels to query access rules in order to configure user interfaces such that they only show those services to a user to which a user has access.

2.2.1 Scope

The scope of the buzzAuthorization is shown in Figure 2.
Administrators are able to assign authorization interceptor to a buzz space which restrict access to one or more services to users who have a particular role and/or a minimum status.

If no authorization interceptor has been added to a particular service, the access to that service is not restricted.

2.2.2 Use cases

This section provides the details use case requirements for the use cases offered by the buzzAuthorization module.

2.2.2.1 addAuthorizationRestriction — priority: important  This use case adds an authorization restriction for a user role and a particular buzz space.

2.2.2.1.1 Service contract  The service contract for the addAuthorizationRestriction service is shown in Figure 3. The pre-conditions are enforced (raising the appropriate exception should they not be met) and a relevant authorization restriction entity is created and persisted through to database.
2.2.2.2 isAuthorized — priority: important  This use case is meant to be used by front-ends (web interface, android client, . . . ) to query the services a user may access in order to customize the user interface for the user. This could be used to, for example, to show to the user only those services which the current user may access.

2.2.2.2.1 Service contract  The service contract for the isAuthorized service is shown in Figure 3 This is a simple query service.
2.2.2.3 **removeAuthorizationRestriction** — *priority: important* This is a simple use case which removes an authorization restriction for a user role from a buzz space.

**2.2.2.3.1 Service contract** The service contract for the **removeAuthorizationRestriction** is shown in Figure 4.
2.2.2.4 getAuthorizationRestrictions — priority: important This use case is used typically by front-ends (web interfaces, Android clients and other UIs) to retrieve the authorization restrictions to enable users to select an authorization restriction they would like to update.
2.2.2.4.1 Service contract

2.2.2.5 updateAuthorizationRestriction — priority: important This use case facilitates editing of authorization restrictions.

2.2.2.5.1 Service contract The service contract for the updateAuthorizationRestriction is shown in Figure 7.
Figure 7: The service contract for updating an authorization restriction for a user role on a buzz space.

2.2.3 Domain model

The buzzAuthorization module maintains the access restriction information in the form of authorization restrictions. Each authorization restriction applies to one or more services identified by the fully qualified name of the interface representing the service contract and the method name.

Figure 8: The domain model of the buzzAuthorization module.

If there is no authorization restriction for a service, then access to that service is not restricted. If there are multiple authorization restrictions, then at least one of the restrictions need to grant
access to a user for a user to be able to make use of the restricted service.

2.3 The Buzz-Spaces module

BuzzSpaces is the core module which is responsible for managing the buzz spaces for the different modules.

2.3.1 Scope

The scope of the buzzSpaces module is shown in Figure 9. The scope is really restricted to space and user management.

![Figure 9: The scope of the buzzSpaces module.](Image)

2.3.2 Use-cases

The buzzSpaces module provides services to create and close spaces.

2.3.2.1 Login and administrative user

The system will authenticate against the CS data sources within which authentication credentials are currently stored. Currently this is an LDAP repository. This ensures that the same authentication credentials are used across the different system used within the department.

2.3.2.1.1 Service contract

The service request for the login use case contains the authentication credentials which are currently UsernamePasswordCredentials but which could change in future. The service will perform the authentication against the Computer Science LDAP Repository.
Figure 10: The service contract for the login use case.

If the login is successful, the userId is returned – this may be the same as the username used for authentication (if the username and password are used as authentication credentials).

Figure 11: The functional for the login use case.

2.3.2.1.2 Functional requirements
2.3.2.1.3 Process specification  The process for this is trivial. Upon receiving the login-request, the service tries to authenticate using the CS data sources adapter. Any exception raised by this lower level service is also raised by this service. In the success scenario the result object is populated and returned.

![Diagram: The process specification for the login use case.]

2.3.2.2 BuzzSpace.createBuzzSpace — priority:critical  The service enables lecturers to create a Buzz space for a particular module they present during a particular year. This creates a root thread for the buzz space with an associated welcome post and assigns the lecturer as administrator to the space.

2.3.2.2.1 Services contract  The CreateBuzzSpaceRequest identifies 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.
The service has 3 pre-conditions, i.e. 3 conditions under which the buzz space will not be created. For each of those pre-conditions an exception is introduced which is raised by the service to notify the caller that the service is not being provided because the pre-condition associated with that exception has not been met.

The post-conditions specify the conditions which must hold true when the service has been provided. The post-conditions for this service are that the requested buzz space has been created and that a root thread has been created for the buzz space.

### 2.3.2.2 Functional requirements

The use case realizes the service specified in the services contract for BuzzSpaces. The functional requirements for the use case specify the lower level functions required by the use case to realize the service as specified in the service contract. Each functional requirement is either checking for a pre-conditions, realizing a post-condition or both.
Figure 14: The functional requirements for the createBuzzSpace use case.

Each functional requirement is associated with a services contract for the responsibility domain within which that lower level service falls. Whichever object realizes that services contract needs to provide the service.

2.3.2.2.3 Process design The process is now assembled by “orchestrating” the service from the lower level services. The outer (context) activity is the activity of realizing the createBuzzSpace service. It receives the service request and requests in its process the lower level services from whichever object is deployed to realize the services contract.

1In the implementation dependency injection is used to fully decouple the implementation classes — i.e. the dependencies are purely on contracts (interfaces) and not on classes.
Figure 15: The process design for the createBuzzSpace use case.

Note that the first activity is to retrieve the buzz space for the module. If a buzz space is provided, our service raises an exception because a buzz space for that service and the current academic year exists already, i.e. we only continue with our service if the lower level getBuzzSpace service raises an exception signalling that no such buzz space exists.

Note also that if any of the three pre-conditions for our service is not met, that an exception is raised and the service is not provided. Only if all preconditions are met is the buzz space created and the result returned.

2.3.2.3 Close buzz space — priority:important The closeBuzzSpace use case is very simple. It is simply sets the buzz space as inactive.

2.3.2.3.1 Service contract The service contract for the closeBuzzSpace use case is shown in Figure [16]
2.3.2.4 **buzzSpace.registerOnBuzzSpace** — priority:critical  Registering a user with a buzz space effectively creates a profile on that buzz space for that user. The profile will be populated, for example, by various modules with activities done and status earned by the user, as well as any notifications or setup information persisted for that user.

**2.3.2.4.1 Service contract**  The service contract for the `registerOnBuzzSpace` use case is shown in Figure 17.
2.3.2.5 getProfileForUser — priority:critical

This is a simple query service which returns the profile the user has on the buzz-space.

2.3.3 Domain model

The domain model for this module is very simple. It only requires that buzz spaces are persisted and that each buzz space has a thread which plays the role of the root thread for that buzz space.

Figure 17: The service contract for the registerOnBuzzSpace use case.

Figure 18: The domain model for the buzzSpaces module.

Note that the relationship between a BuzzSpace and that Thread which is the root thread for that space is a composition relationship as the root thread

- is only accessible from that space, and
- if the space is deleted, so is its root thread.
2.4 The Buzz-Data-Sources module

This module is responsible for sourcing data from external CS databases, i.e. databases maintained by Computer Science but which are not part of this system. The access is purely read-access, i.e. no data will be modified in the external databases.

2.4.1 Scope

![Diagram](image)

Figure 19: The scope of the CsDataSources module.

2.4.2 Use cases

This module provides a number of services to Buzz which require information from data stored in external databases maintained by the Department of Computer Science.

2.4.2.1 Login and administrative user

The system will authenticate against the CS data sources within which authentication credentials are currently stored. Currently this is an LDAP repository. This ensures that the same authentication credentials are used across the different system used within the department.

2.4.2.1.1 Service contract
The service request for the login use case contains the authentication credentials which are currently UsernamePasswordCredentials but which could change in future. The service will perform the authentication against the Computer Science LDAP Repository.
If the login is successful, the userId is returned – this may be the same as the username used for authentication (if the username and password are used as authentication credentials).

2.4.2.1.2 Functional requirements
2.4.2.1.3 Process specification  The process for this is trivial. Upon receiving the login-request, the service tries to authenticate using the CS data sources adapter. Any exception raised by this lower level service is also raised by this service. In the success scenario the result object is populated and returned.

![diagram of process specification](image)

Figure 22: The process specification for the login use case.

2.4.2.2 getUsersRolesForModule  This use case queries the user roles for a particular user. This is basically a database lookup and hence only the service contract is specified. The design (functional requirements and process specification) are so simple that they are left to the for the implementation phase.

2.4.2.2.1 Service contract  Figure 23 depicts the services contract for the use case.

![diagram of service contract](image)

Figure 23: The service contract for the getUsersRolesForModule use case.

Note that there are two pre-conditions and hence that the service will possibly throw two notifiable exceptions. Not also the associations to User, Module and Role. These are not composition relationships and hence the request and result object do not contain the user, module and role information themselves, just references or identifiers for users, roles and modules.
2.4.2.3  getUsersWithRole  This use case retrieves all users which have a particular role (e.g. a teachingAssistant role) for a particular module. Since the use case is effectively just a database lookup with the appropriate result object creation, we only specify the services contract and leave the design to development.

2.4.2.3.1  Service contract  Figure 24 depicts the services contract for the use case.

![Service contract diagram](image)

Figure 24: The service contract for the getUsersForRole use case.

Add missing use cases

2.4.3  External database structures

The details for the external database structures can be obtained from Neels van Rooyen who is working at the tech team. Neels may also assist you with other technicalities around integrating with the CS databases.

The relational database structure containing the courses/module information has the following structure:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Key</th>
<th>Default</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>int(11)</td>
<td>NO</td>
<td>PRI</td>
<td>NULL</td>
<td>auto_increment</td>
</tr>
<tr>
<td>code</td>
<td>varchar(20)</td>
<td>NO</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>varchar(255)</td>
<td>NO</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>lecturer</td>
<td>varchar(255)</td>
<td>NO</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>description</td>
<td>text</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>semester</td>
<td>smallint(6)</td>
<td>NO</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>has_webct</td>
<td>tinyint(4)</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>year_group</td>
<td>int(11)</td>
<td>NO</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>hidden</td>
<td>tinyint(3) unsigned</td>
<td>NO</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>last_updated</td>
<td>datetime</td>
<td>NO</td>
<td></td>
<td>0000-00-00 00:00:00</td>
<td></td>
</tr>
<tr>
<td>discussion_board</td>
<td>tinyint(4)</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>tutors_allowed</td>
<td>tinyint(2)</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
</tbody>
</table>
The structure of the LDAP repository is shown in the following figure.

For example, an LDAP search for stud_COS301 returns:

dn: cn=stud_COS301,ou=Modules,ou=Groups,ou=Computer Science,o=University of Pretoria,c=ZA
objectClass: top
objectClass: posixGroup
cn: stud_COS301
gidNumber: 10093
memberUid: u11061015
memberUid: u12214834
memberUid: u11063612
memberUid: u11002566
memberUid: u12019837
memberUid: u11371910
memberUid: u29557373
memberUid: u11247143

and an LDAP search for u29052735 returns:

dn: uid=u29052735,ou=students,ou=Computer Science,o=University of Pretoria,c=ZA
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: top
uidNumber: 26526
gidNumber: 10001
loginShell: /bin/bash
title: Mr
initials: CJ
st: 8911085042083
sn: van Rooyen
homeDirectory: /home/cs/students/u29052735
employeeNumber: 29052735
uid: u29052735
mail: nexusdk@gmail.com
cn: Neels

and an LDAP search for memberuid=u29052735 returns:
2.4.4 Domain model

There is no domain model for this module as this is purely an adapter providing services to Buzz. This module does not persist any information into a database.

2.5 The buzzResources Module

The buzzResources module is used to upload and manage resources like media (e.g. images, video, ...) and documents (e.g. PDF documents, Open Document Format documents, ...). These resources can be either embedded or linked to in posts.

2.5.1 Scope

The scope of the buzzResources module is shown in Figure 25.

![Figure 25: The scope of the buzzResources module.](image)

2.5.2 Use cases

This section specifies the use case requirements for the use cases offered by the buzzResources module.

2.5.2.1 uploadResource Users can upload resources like media files or documents. Any uploaded resource is accessible by other users who can also specify links to that resource.
2.5.2.1.1 Services contract

2.5.2.1.2 Functional requirements  The functional requirements for the uploadResource use case are shown in Figure 27.

2.5.2.1.3 Process design  The process is now assembled by “orchestrating” the service from the lower level services. The outer (context) activity is the activity of realizing the createBuzzSpace service. It receives the service request and requests in its process the lower level services from
whichever object is deployed to realize the services contract.\(^2\)

Figure 28: The process design for the createBuzzSpace use case.

Note that the first activity is to retrieve the buzz space for the module. If a buzz space is provided, our service raises an exception because a buzz space for that service and the current academic year exists already, i.e. we only continue with our service if the lower level getBuzzSpace service raises an exception signalling that no such buzz space exists.

Note also that if any of the three pre-conditions for our service is not met, that an exception is raised and the service is not provided. Only if all preconditions are met is the buzz space created and the result returned.

\(^2\)In the implementation dependency injection is used to fully decouple the implementation classes — i.e. the dependencies are purely on contracts (interfaces) and not on classes.
2.5.3 Domain model

![Domain model diagram](image)

Figure 29: The domain model for the buzzResources module.

2.6 Threads

The *BuzzThreads* module is responsible for the functionality around threads and posts.

2.6.1 Scope

The scope of the *Treads* module is depicted in Figure 30. Users can CRUD posts. Depending on their privileges there may be restrictions on where users will be allowed to create posts.
2.6.2 Use cases

The Threads module provides services to CRUD posts. In addition it also provides functions to change properties of a thread that influences how a thread is displayed, like hiding and closing threads.

An important feature is to allow users with authority to move a branch to a new location. In essence it boils down to change the parent of a selected post.

Furthermore information regarding the posts in a thread can be gathered using the versatile functions called queryThread and getThreadStats.

2.6.2.1 submitPost — priority:critical  The service enables a user to submit a post to thread, creating a child thread. Top level threads for a buzz space are created by submitting a post against the rootThread of the buzzSpace.

2.6.2.1.1 Services contract  The SubmitPostRequest contains the post itself and identifies the thread the user wants to post on and through this the buzz space the user is posting on. The post itself has a content and a mime type. Initially only txt and html mime types will be supported. Within HTML posts one can embed uploaded images and provide links to uploaded documents.
The service has a number following pre-conditions. For each pre-conditions an exception is introduced which is raised by the service to notify the caller that the service is not being provided because the pre-condition associated with that exception has not been met.

2.6.2.2 markPostAsRead – priority:niceToHave

This use case simply stores a reading event which contains the information that a user has read a particular post at a particular time.

2.6.2.2.1 Services contract

The CloseRequest identifies the thread the user wants to close and may optionally specify a ThreadSumamrizer which should be used to summarize the thread. The thread summary can be text-based or hypertext (HTML) with embedded images and links to uploaded documents.
2.6.2.3 closeThread — priority:important This service closes a thread preventing any further changes to a thread. This includes preventing further posts, appraisals and all other activities on a thread by any normal users. Optionally a thread can be summarized either manually or via an automated thread summarizer.

2.6.2.3.1 Services contract The CloseRequest identifies the thread the user wants to close and may optionally specify a ThreadSummarizer which should be used to summarize the
thread. The thread summary can be text-based or hypertext (HTML) with embedded images and links to uploaded documents.

![Class closeThread diagram](image)

**Figure 33:** The service contract for the closeThread use case.

### 2.6.2.3.2 Thread summarizers

The design for automated and manual thread summarizers is shown in Figure ??

---

35
2.6.2.3.3 Functional requirements When a thread is closed one can optionally request thread summarization. If thread summarization is not requested, the thread continues to be readable, but no modifications can be made by any users other than administrators. Administrators will be
able to add appraisal types to closed thread for assessment purposes.

Figure 35: The functional requirements for the closeThread use case.

Threads can be summarized manually or using an automatic thread summarizer.

2.6.2.3.4 Process design Figure 36 is an activity diagram which specifies the process for the closeThread use case.
2.6.2.4 moveThread — priority:important  This use case is used to detach a sub-tree of thread nodes from one thread and add it to another thread. The accessibility of this use case is determined by the AuthorizationRestrictions set for the buzz space.

2.6.2.5 hideThread — priority:important  This use case is not meant to be used directly by users, but is a functionality required by the moderation module. The selected thread nodes and all its descendant nodes will be marked as hidden and user interfaces are meant not to render them.

2.6.2.6 Threads.queryThread — priority:medium  The functionality provided by the queryThread function is to provide a versatile way to get all the information of subsets of posts complying with specified restrictions.

2.6.2.6.1 Services contract  The queryThreadRequest requires no pre-conditions and does not change anything.

The data set returned by this function should contain the following fields for each post in the set:

ParentID  Post ID of parent post
Author  the ID of the user who created the post
TimeStamp  Date and time when the item was posted
Content  the text in the post
Status  hidden / deleted / closed / etc.
2.6.2.6.2 Functional requirements

The use case realizes the service specified in the services contract for Threads.

The getThread service returns a handle to the thread node and provides access to the thread tree. The default that is returned by queryThread is a dataset containing all posts in the thread, i.e. all posts in the tree of which the postID is specified. It should, however, be possible to specify each of the following as well as any combination of these:

startDate Time Restrict returned posts to be after this time stamp. Default is the time stamp of the root post in the Buzz space.

endDate Time Restrict returned posts to be before this time stamp. If unspecified all posts are returned.

maxLevel Restrict returned posts to be at most at the specified depth relative to the post. If this value is 0, minLevel will also be 0 only the specified post is returned.

minLevel Restrict return posts to be at least at the specified depth relative to the post. Obviously it has to be less or equal to maxLevel. If both minLevel and maxLevel is 1, only the immediate children is retrieved.

userGroup Restrict returned posts to be limited to a specified user group. Default is all users. It should be easy to limit it to a single user or a set of users who have a specified status or reputation.

phraseSet Restrict returned posts to be only posts that contains all the strings specified in the phrase set. The default is an empty set. If the set is empty all posts are returned.

2.6.3 Domain model

The domain model for this module requires that posts are persistent. Each post has a parent post and may have any number of children posts. All posts within a BuzzSpace are descendants of the root thread of the BuzzSpace.
2.7 The Buzz-Status module

Buzz-Status is the module which provides the functionality to assess a number of measures around individual buzz-space contributions, to use these to calculate a BuzzSpace status for a profile (i.e. for a user and a specific module), and to restrict access to system functionality based on user status and user role.

2.7.1 Scope

The scope of the buzzStatus module is shown in Figure 38.
2.7.2 Use-cases

The use cases of the status module provide functionality around appraisals of posts, profile assessment, and status calculation.

2.7.2.1 assessProfile  The system will enable lecturers to specify different ways in which the profiles of users can be assessed. This is done through a number of pluggable ProfileAssessors, each implementing the ProfileAssessor interface (see Figure ??).
2.7.2.2 setStatusCalculator  By default the NumPostsAssessor is assigned as statusCalculator for a buzz space, i.e. the status earned by a space member is directly proportional to the number of posts the member has made. This use case is used to assign any other ProfileAssessor to be used as the statusCalculator for a particular Buzz Space.

2.7.2.2.1 Services contract  The status calculator can only be set for a buzz space which is open. The SetStatusCalculatorRequest identifies the buzz space for which the status calculator is to be set and the profile assessor which should be used as status calculator.
Once the status calculator has been set for a buzz space, the status of all profiles of that space will have been recalculated.

2.7.2.3 getProfileStatus This is a simple query service which returns the status for a specific profile, i.e. the status a user has on a particular buzz space.

2.7.2.3.1 Services contract The service contract for the getProfileStatus use case is shown in Figure 41.
2.7.2.4 createAppraisalType  This use case creates a new appraisal type which can be reused across buzz spaces.

An appraisal type (e.g. FunnyAppraisalType) has different appraisal levels (Hilarious, Funny, Boring). These are in an ordered list (0, 1, 2) which correspond to the level values. Each appraisal level has optionally an icon (e.g. laughing, smiling and bored smiley type face) which can be used in the web interface to provide clickable icons to assign appraisals to posts. One can also specify an notRatedIcon which can be displayed in the UI when the current user has not yet rated the current post with this particular appraisal type.
2.7.2.5 activateAppraisalType  This use case activates an appraisal type for a specified period on a specified buzz space.

Figure 43: The service contract for creating an appraisal type.

Figure 44: The data structure of appraisal type activation.
2.7.2.6 assignAppraisalToPost  This use case enables a member to assign an appraisal to a post.
Figure 46: The service contract for assigning an appraisal to a post

### 2.7.2.6.1 Service contract
2.7.3 Domain model

Figure 47: The domain model for the buzzStatus module.

2.8 buzzNotification

The functionality provided by the buzzNotification module is focused on registering to receive notification messages for, for example, new posts submitted on a thread, posts of a particular user (tracking a user) and notification of any appraisals one receives for any of one’s posts. Notifications are at this stage via email and the email address maintained in the CS database will be used.

2.8.1 Scope

The scope of the buzzNotification module is shown in Figure 48.

Figure 48: The scope of the buzzNotification module.
2.8.2 Use-cases

The buzz notification module track threads from other modules and users of the buzz system. Its main functional requirements is to track activity threads via email. Below is the overview of the notification module.

2.8.2.1 registerForNotification — priority: niceToHave  This use case enables users to register to receive notifications on posts and appraisals. For example, a user may want to receive notification of any post made on a particular thread. If this is requested for the root thread of a buzz space, then the user will receive notification of any post submitted on that entire buzz space.

Users can restrict the notifications to posts made by certain users. If no users (profiles) are specified, then notification for posts made by all users on a thread is requested.

In addition to post notifications, users can also request notifications on any appraisals they receive on a specified thread. Once again, if the root thread for the space is selected, users will receive notification of all appraisals they receive on a buzz space.

2.8.2.1.1 Services contract  The service contract for the registerForNotification use case is shown in Figure 49.

Figure 49: The service contract for the registerForNotification use case.
2.8.3 The domain model for buzzNotification

![Diagram of the domain model for buzzNotification](image)

Figure 50: The domainModel for the buzzNotification module.

2.9 The web module

The Web module provides a portal based, web responsive front end that allows a user to get access to the application and its facets. The web module provides a web-based human integration channel. The purpose of this module is to make all user services from the business logic layer accessible through web browsers.

As such the purpose of the web module is to capture the information for the request objects in the business logic layer, submit the service requests to that layer and render the responses in a useful way back to the user.

To this end one needs to differentiate between business processes and user work-flows. user-work flows are managed in the presentation layer whilst business processes are encoded within the services of the business logic layer.

2.9.1 Scope

The scope of the web module is the that of making all the system’s user services accessible to humans through the web. Hence it needs to cover the user services from the various modules. Lecturers must be able to create buzz spaces, members must be able to submit posts and appraise posts of other users, moderators must be able to moderate posts and so on.

It is thus important to check that all user services from the different modules are included within the scope of the web module.
2.9.2 UI requirements

The UI development should be guided throughout by standards and usability guidelines. The following UI requirements need to be balanced:

- intuitiveness and usability,
- making the UI interesting and dynamic, and
- making the user interface configurable and personalizable.

2.9.2.1 Login/out

The system will allow users to log into or out of the CS systems. This means a user is either logged into the entire buzz space system, being able to see multiple buzz spaces potentially simultaneously). The system will also have a logout functionality where a user would like to reduce the risk that another person uses their computer or device to perform unauthorized actions (e.g. posts) on their behalf.

2.9.2.2 Spaces

Users should be able to see all spaces which they could currently be a member of, i.e. the spaces of all those modules which they are currently registered for. There needs to be a differentiation between the spaces for which they have registered and the spaces they have not yet registered for.

2.9.2.3 UI considerations around threads

Threads should be viewable in two different ways:

1. as an expandable/collapsible tree structure rendering nodes containing posts which have been read in a faint style.
2. as a flat list of posts showing the contents of each post inline.

The user interface needs to facilitate that a user can post on any thread node from either view. If a user posts on the root thread node of a buzz space, a new first-level thread is created on that space.

2.9.2.4 Posts

Each post should be rendered with a header area, a content area and a footer area.

- the title of the post,
- the username of the member who submitted the post,
- the date/time stamp for the post,
- and the current appraisals received by the post.

The content of a post can be encoded as either text or as hypertext markup (HTML). The latter may have embedded images (where images are sourced from uploaded content) and links to uploaded documents and should be rendered in a way which is analogous to a browser rendering the same HTML.

The footer area has the links or buttons to respond to the post and to appraise a post.
Each space can have different and multiple appraisal types. An appraisal type should typically be rendered as a group of icons with each icon representing an appraisal level for that appraisal type. For example, you might have a laughing, smiling and bored phase to represent three levels of funny in a funny appraisal type or thumb symbols for a usefulness appraisal type.

Similarly, in order to submit a new post, the post content must be captured. For that a rich editor supporting the capturing of marked up text (HTML) needs to be used.

2.9.2.5 Profile information A user should be able to view his or her profile information for a space including

- the user name on that space,
- the profile picture the user has on that space,
- the role the user has on that space,
- the status the user has on that space, and
- the number of posts and appraisals the user has made/received on that space.

2.9.2.6 Personalization and configurability A user should be able to configure a dashboard type of view for buzzSpaces where the user should be to assemble a page by positioning different view windows on that page. For example, a user might have on the dashboard a window with his/her profile information, and for each buzz space for which she/he is registered a window with a tree view of the threads.

2.9.2.7 Accessibility of functionality Authorization is enforced at the business logic (services) layer and not at the presentation layer. Access to a service may depend on a user’s role on a buzz space as well as on a user’s status on that space.

Nevertheless, the user interface should only enable the user to access functionality which he/she is authorized to use. This can be, for example, done by greying out buttons or menu items which are used to access functionality which the user is currently not authorized to use. The buzzAuthorization module provide a isAuthorized service which can be used by the web module to determine whether a user may currently use some functionality.

2.10 The Buzz Reporting module

The functionality provided by the buzzReporting module includes the following:

1. It provides statistical information that can be used in the interface to display facts about the average user and how the logged-in user compares with the average.

2. It provides ways to gather data that is in the persistent store and present it in a format that is usable by other modules that are external to Buzz.

3. It provides functionality to alter record sets in a Buzz space by uploading the relevant information that is stored in a csv file.

3Please read the section on the buzzAuthorization module to understand this section
2.10.1 Use-cases

The *reporting* module provides services to gather, import and export information about posts in specified subsets.

2.10.1.1 Treads.getThreadStats — priority:medium

The functionality provided by the `getThreadStats` function is to provide a versatile way to get statistical information of subsets of posts complying with specified restrictions.

**2.10.1.1.1 Functional requirements**

The parameters passed to the function is

- a set of posts returned by the `Threads.queryThread` function.
- action keyword

The set of posts returned by the `Threads.queryThread` function is analysed according to the specified action keyword. The following describes the result returned upon each of the action keywords:

- **Num** A count of the entries in the dataset that was created.
- **MemCount** A count of the number of members who are the creators of posts in the dataset.
- **MaxDepth** The maximum depth of a post in the queried thread tree.
- **AvgDepth** The average depth of a post in the queried thread tree.

2.10.1.2 Get Thread Appraisal — priority:medium

The functionality provided by the `getThreadAppraisal` function is to provide a versatile way to get detailed or statistical information of subsets of posts complying with specified restrictions and their associated appraisals assigned by specified members.

**2.10.1.2.1 Functional requirements**

The parameters passed to the function is

- a set of posts returned by the `Threads.queryThread` function.
- a set of members
- a set of appraisals
- action keyword

A data set is created containing entry for each valid member-appraisal-post combination. i.e. for each member in the set that may assign an appraisal in the set to a post in the set a separate entry is created.

Each entry has a field that should contain an ordinal number that represent a level of the specified appraisal. This field is empty for posts for which a member has not assigned an appraisal.

The following describes the result returned upon each of the action keywords:

- **All** A detailed dataset containing all post detail, memberID, appraisalID and appraisal value (ordinal number).
**Sum**  The sum of all appraisal values for the entries in the dataset that was created.

**Avg**  The average of all appraisal values for the entries in the dataset that was created.

**Max**  The maximum of all appraisal values for the entries in the dataset that was created.

**Min**  The minimum of all non-empty appraisal values for the entries in the dataset that was created.

**Num**  A count of all non-empty appraisal values for the entries in the dataset that was created.

### 2.10.1.3 Export Thread Appraisal — priority:medium
The functionality provided by the `exportThreadAppraisal` function is to realise an off-line facility to apply a manual appraisal. It creates the dataset to be used that can be edited off-line and allow updates to be inserted through the `importThreadAppraisal` function.

#### 2.10.1.3.1 Functional requirements
An external file is created containing data generated by the `getThreadAppraisal` function only if the function executed with the All action keyword.

### 2.10.1.4 Import Thread Appraisal — priority:medium
The functionality provided by the `importThreadAppraisal` function is to realise an offline-facility to apply a manual appraisal. It is dependant on the `exportThreadAppraisal` function.

#### 2.10.1.4.1 Functional requirements
Data in an external file that was created using the `exportThreadAppraisal` function is used. It is required that the data set is associated with only one member and only one specified appraisal to be eligible for import.

A record contains all detail about the post along with a field that should contain an ordinal number that represents the levels of the specified appraisal.

It is assumed that the detail of the posts are not edited. Edits to this data is ignored when importing a thread appraisal.

For each record the `assignAppraisalToPost` function is applied. The appraisal level as stored in the file for each post is updated as an appraisal assigned by the member associated with the data set. Although it can be assumed that the member and the appraisal is valid for each post in the set, their validity should be checked. If invalid an exception is raised and the service not delivered. If the appraisal level is outside the range specified for the appraisal an exception is raised and the service not delivered.

### 2.10.1.5 Export Thread — priority:low
The functionality provided by the `exportThread` function is to provide means to backup the content of a thread or subset of a thread in a serialised text file.

#### 2.10.1.5.1 Functional requirements
An external file is created containing data generated by the `Treads.queryThread` function.

### 2.10.1.6 Import Thread — priority:low
The functionality provided by the `importThread` function is to provide means to restore the content of a thread or subset of a thread that was stored using the `exportThread` function.
2.10.1.6.1 Functional requirements

An external file that was created by the export-Thread function is restored in the Buzz space. A post is added to the Buzz space only if it is not in the Buzz space i.e. it should be ensured that duplicates of posts are not created when using this function.

2.11 The Buzz-Android-Client module

This module is out of scope for the mini-project.

2.12 The Buzz-Hamster-Integration module

This module is out of scope for the mini-project.

3 Architectural requirements

This section discusses the software architecture requirements — that is the requirements around the software infrastructure within which the application functionality is to be developed. The purpose of this infrastructure is to address the non-functional requirements. In particular, the architecture requirements specify

- the architectural responsibilities which need to be addressed,
- the access and integration requirements for the system,
- the quality requirements, and
- the architecture constraints specified by the client.

3.1 Access and integration requirements

This section discusses

1. the requirements for the different channels through which the system can be accessed by people and systems, and
2. the integration channels which must be supported by this system.

This section specifies the different channels through which users will be able to access the system services.

3.1.1 Human access channels

The system will be accessible by human users through the following channels:

1. From a web browser through a rich web interface. The system must be accessible from any of the standards-compliant web browsers including all recent versions of Mozilla Firefox, Google Chrome, Apple Safari and Microsoft Internet Explorer.

2. From mobile Android device

The Android client is out of scope for the mini-project.
3.1.2 System access channels

Other systems should be able to access the services offered by the system through convenient access channels which are supported across different technologies and frameworks. This integration channel should be usable by a mobile client which is to be developed in the future.

3.1.3 Integration channels

This system will be able to access

- the CS LDAP server in order to retrieve person details and class lists.
- the CS MySQL database to access course/module information.

Details of the respective data structures are given in appendix 2.4.3. Further details regarding the integration to these databases and the use of the mock databases created for this project can be obtained from the TechTeam.

In addition, the system will allow manual integration through importing and exporting of CSV files. In particular, the system will support

- Importing of assessment entries from CSV files.
- Exporting of mark sheets to CSV files.

3.2 Quality requirements

The quality requirements are the requirements around the quality attributes of the systems and the services it provides. This includes requirements like maintainability/flexibility, extensibility, performance, scalability, security, auditability, usability, and testability requirements.

3.2.1 Maintainability

Amongst the most important quality requirements for the system is maintainability which includes flexibility and extensibility. It should be easy to maintain the system in the future. To this end

- future developers should be able to easily understand the system,
- the technologies chosen for the system an be reasonably expected to be available for a long time,
- and developers should be able to easily and relatively quickly
  - change aspects of the functionality the system provides, and
  - add new functionality to the system.
3.2.2 Scalability

1. Initially the system must support the ability to host buzz spaces for all Computer Science modules at a University of the size of the University of Pretoria.

2. It should be possible to, in the future, scale the architecture such that the system can be used
   • University-wide, servicing in the order of 50 000 students, and
   • for Massive Open Online Courses (MOOC) by porting the system onto clustered and cloud-computing based architectures.

3.2.3 Performance requirements

The system does not have particularly stringent performance requirements.

1. All non-reporting operations should respond within less than 0.2 seconds.

2. Report queries should be processed in no more than 5 seconds.

The above figures do not include the network round-trip which is outside the control of the system.

3.2.4 Reliability and Availability

The system should provide by default a reasonable level of availability and reliability and should be deployable within configurations which provide a high level availability, supporting

• fail-over safety of all components and
• a deployment without single points of failure.

Hot deployment of new/changing functionality is not required for this system.

3.2.5 Security

Initially the system needs to support only

• authentication against a chosen user repository (initially against the LDAP repository of the Computer Science Department of the University of Pretoria),
• and a flexible, configurable authorization framework allowing administrators to configure the access to any particular service based on the user role and other factors like the user status.

In future the system is expected to also enforce confidentiality through encrypted communication and protection against man-in-the-middle attacks through hashing.
3.2.6 Auditability

The system will log all requests and all responses (including exceptions) for all user services provided by the system.

For each request the log should contain an entry with

- an id for the log entry,
- the userId of the user requesting the service,
- the date/time stamp when the request was made,
- the user service requested, and
- the request object stringified as JSON with any sensitive information like passwords removed.

For each response the log should contain an entry with

- an id for the log entry,
- the id of the corresponding request entry,
- the date/time stamp when the response is provided, and
- the response object stringified as JSON with any sensitive information like passwords removed.

The system will provide only services to extract information from the audit log and will not allow the audit log to be modified.

Audit logs will be directly accessible to both, humans and systems.

3.2.7 Testability

All services offered by the system must be testable through

1. unit tests testing components in isolation using mock objects, and
2. integration tests where components are integrated within the actual environment, which test.

In either case, these functional tests should verify that

- that the service is provided if all pre-conditions are met (i.e. that no exception is raised except if one of the pre-conditions for the service is not met), and
- that all post-conditions hold true once the service has been provided.

In addition to functional testing, the quality requirements like scalability, usability, auditability, performance and so on should also be tested.
3.2.8 Usability

Usability is one of the most important quality attributes. The system should be intuitive and fun to use. Usability tests should be performed in order to assess whether

- users with a literacy computer rate of an average first year students are able to use the system without any guidance or documentation,
- whether users find any aspects of the system cumbersome, non-intuitive or frustrating.

In addition the system should be usable by users which have different language preferences. Initially the system needs to support the three official languages of the University of Pretoria, Afrikaans, Sepedi and English.

Innovation and novel approaches are encouraged for this system. Nevertheless, in cases where a UI design aspect deviates from accepted standards and norms, there should be a strong usability reason for having deviated from these norms.

3.2.9 Integrability

The system should be able to easily address future integration requirements by providing access to its services using widely adopted public standards.

3.2.10 Deployability

The system must be deployable

- on Linux servers,
- in environments using different databases for persistence of the Buzz data, and
- in environments where the user authentication credentials and roles are sourced from different repositories.

3.3 Architectural responsibilities

The architectural responsibilities include the responsibilities of providing an infrastructure for

1. a web access channel,
2. hosting and providing the execution environment for the services/business logic of the system,
3. persisting and providing access to domain objects,
4. specifying and executing reports,
5. sending emails,
6. logging,
7. integrating with an LDAP repository.
3.4 Architecture constraints

The choice of architecture is largely unconstrained and the development team has the freedom to choose the architecture and technologies best suited to fulfill the non-functional requirements for the system subject to

1. the architecture being deployable on Linux servers, and

2. the architecture using only open source frameworks and tools.

4 Architecture design

4.1 Overview

layered architecture: - presentation layer - access layer - services layer - domain objects layer - infrastructure layer (ORM, integration channels, ...)

presentation layer - django web app - android app - restful web services access for other systems
layers: presentation business logic domain objects infrastructure backend

4.2 Modularization

The software architecture of Buzz is a modular software architecture with a number of core modules and a number of pluggable add-on modules. Further add-on modules can be added at a later stage. Add-on modules may add additional functionality and may enrich existing functionality through interception.
4.3 Architectural tactics addressing quality requirements

This section discusses the architectural tactics which are used to concretely address the quality requirements for the application.

4.3.1 Minimize technology suite

In order to improve maintainability (requirement 3.2.1) and reduce the skills requirements for the team who needs to maintain the system, the software architecture will minimize the number and complexity of programming languages and frameworks used.

4.3.2 Dependency Injection (DI) — flexibility, deployability, testability

Dependency Injection will be used to ensure that the environment will provide the appropriate service providers for the software components (e.g. production and testing environments). It enables one to easily change the service providers which should be used by an application.

DI will improve

- flexibility/maintainability (requirement 3.2.1) by being able to replace service providers through configuration,
• *testability* (requirement 3.2.7) through being able to inject mocking objects in unit tests, and

• *deployability* (requirement 3.2.10) into different environments where different service providers for, for example, persistence, authentication, . . . , are used by configuring by having the environment inject the appropriate service providers.

### 4.3.3 Aspect-Oriented Programming (AOP)

AOP is used as flexible/maintainable interception mechanism to wire in additional functionality across domain of services. It can be used to manage interceptions across an application’s services effectively.

AOP based interception will be used to

• intercepting services with authorization functionality,

• enhance core services with additional functionality required by plugin modules,

• apply logging across all services, and to

• enrich services with any configuration dependent functionality.

AOP improves

• *extensibility* (requirement 3.2.1) enabling one to add additional functionality to existing services and, in particular, facilitating the plugging in of modules which enrich existing services with additional functionality,

• *auditability* (requirement 3.2.6) being able to weave in logging functionality across services, and

• *security* (requirement 3.2.5) being able to weave in authorization functionality across all services.

To explain the benefit which AOP provides to the ability of plugging in extension modules, consider the notification and status modules. The former weaves in a *statusUpdater* interceptor into the *submitPost* and *submitAppraisal* services which updates the member’s status upon these events.

Similarly, a *notificationInterceptor* will intercept the same two services to send notification messages to members who requested notification of certain events.

### 4.3.4 Contracts based development

Contracts will be enforced across

• services via services contracts with pre and post-conditions which are assessed in unit tests, and

• data structure constraints enforced through data structure validation.

A contracts based approach improves
- **flexibility/maintainability** (requirement 3.2.1) through being able to replace a component with another which realizes the same contract,

- **testability** (requirement 3.2.7) as the contract clearly specifies the requirements for the component and since the contracts for the lower level components allow plugging in of mocking service providers.

### 4.3.5 Support plugin framework

The application will provide a plugin framework in order to allow to facilitate *extensibility* and *flexibility* (requirement 3.2.1). This will allow for different configurations of the application to be installed and also for future development of pluggable modules providing additional functionality.

### 4.3.6 Database abstraction

The system will include a database abstraction layer improving

- **deployability** (requirement 3.2.10) in environments where different databases are used,

- **flexibility** (requirement 3.2.1) allowing the database to be replaced for one which better addresses the potentially evolving persistence requirements.

### 4.3.7 Automated persistence mapping

The architecture will provide automated persistence mapping in order to improve *flexibility/maintainability* (requirement 3.2.1) by

- simplifying the replacement of one persistence technology with another,

- reducing the code bulk which needs to be maintained, and

- not polluting application logic with persistence logic.

### 4.3.8 Caching

Caching is a form of resource reuse used to improve *scalability* (requirement 3.2.2) and *performance* (requirement 3.2.3). Caching should be used to cache any fetched data which is unlikely to change,

### 4.3.9 Connection pooling

The system will use *connection pooling* and in particular *database connection pooling* as a special form of resource reuse to improve *scalability* (requirement 3.2.2) and *performance* (3.2.3).

### 4.3.10 Indexing

*Indexing* of persisted objects on their attributes will be used to accelerate the speed with which they can be retrieved. This will improve performance (requirement 3.2.3) and scalability (requirement 3.2.2).
4.3.11 Templating

Templating must be used to

- enforce a consistent UI, improving usability (requirement 3.2.8), and
- to improve maintainability (requirement 3.2.1).

4.3.12 UI components framework

The system will use a rich, dynamic JavaScript component library in order to

- provide a rich, dynamic user interface (requirement 3.2.8),
- improve the maintainability of the system (requirement 3.2.1) through component reuse and aggregation,
- and to improve scalability (requirement 3.2.2) and performance (requirement 3.2.3).

4.3.13 Asynchronous processing

In order to reduce improve scalability and performance, the system will use asynchronous processing, deferring, as far as possible, tasks which have to wait for resources and tasks which have to perform lengthy processing into the background. This will avoid holding up request processing threads, make the system more responsive and reduce the need for excessive thread switching.

4.3.14 Internationalization

The application will use an internationalization framework to support multiple languages (initially the 3 official languages of the University of Pretoria) to address aspects of the usability requirements 3.2.8

4.3.15 Off-load rendering responsibilities to client

In order improve performance, scalability and usability (requirements 3.2.3 3.2.2 and 3.2.8) rendering which can be done on the client side (within the browser) is to be delegated to the browser.

4.3.16 Rest web-services

In order to address the integrability requirement (requirement 3.2.9), all services are published as RESTful web services with the request and result objects encoded in JSON. This will allow for the future development of application clients (e.g. mobile clients) and for other systems to use the services (e.g. to facilitate automated posting, appraisal assignment and report generation).

4.3.17 Logging

The system will employ logging to capture audit data (requirement 3.2.6). Logging functionality is to be applied via aspects as to not pollute the business logic resulting in more maintainable business logic (requirement 3.2.1) and also to make the logging itself fore flexible and maintainable (requirement 3.2.1). The system will have a web front-end to provide humans access to the system logs.
4.3.18 Clustering

The frameworks must be chosen such that they do allow for clustered deployments in the future. Clustering is, however, not required for the current version of the Buzz system.

4.4 Architectural components

This section discusses the architectural components, technologies and frameworks used to address the architectural responsibilities and the architectural tactics chosen to address the quality requirements as specified in section 3.2.

4.4.1 JavaScript

JavaScript is chosen as a single programming language used across the presentation and business logic layers of the system in order to implement the tactic of minimizing the technology suite as specified in section 4.3.1. This prototype based dynamic programming language which supports duck typing is aligned with realizing flexibility and rapid development. Using a single programming language across the client and the server reduces complexity and improves maintainability.

4.4.2 Node.js

The system will be deployed in a Node.js execution environment. Node.js implements the asynchronous-processing (tactic 4.3.13) by providing a framework for event-driven asynchronous callbacks in the form of asynchronous libraries.

These typically provide second order functions which receive two functions as arguments.

```javascript
1: provider.someFunction(task, callbackFunction);
```

The first is the function which may require waiting for resources or consume a significant amount of time and the second is the callback function which is to be called once the results from the first function are obtained.

4.4.3 Electrolyte

Electrolyte is to be used as a simple, light-weight dependency injection (tactic 4.3.2) framework which introduces minimal intrusion into the JavaScript coding style. The framework provides a minimal Inversion of Control (IoC) container.

This improves service provider flexibility, deployability into different environment and simplifies unit testing with mocking as well as integration testing.

4.4.4 node-aop

Aspect-Oriented Programming (tactic 4.3.3) will be addressed by incorporating the node-aop framework into the software architecture of Buzz. This framework will be used to apply a range of interceptors across a subset of the system’s services including

- authorization interceptors,

---

5For both of these it is common to use anonymous inner functions or λ-expressions
• notification interceptors, and
• status-update interceptors.

4.4.5 Broadway plugin framework
The application is going to be developed as a modular application which
• can be deployed with different modules, and
• and allowing for new modules to be plugged in at a later stage.

This will be facilitated by including the Broadway plugin framework to implement tactic 4.3.5, improving both flexibility and extensibility (requirement 3.2.1).

4.4.6 Express HTTP server
Express will be used as a minimalist, high-performance HTTP server which supports configurable routing (tactic ??) in order to improve maintainability and flexibility.

4.4.7 Handlebars
The Handlebars templating (tactic 4.3.11) framework provides a templating language and compiler which is easily integrated with the Express HTTP server. It can render templates on both the client side and the server side.

The system will use client-side rendering of templates.

4.4.8 react.js
The system will be built using the react.js JavaScript component library to implement tactic ??.

Performance in improved by react.js building an internal virtual DOM and performing a diff against the actual DOM in order to determine which components are to be updated. The react.js framework does facilitate the aggregation of basic components into higher level components.

The react.js framework also enables one to implement tactic ?? in order to improve performance, scalability and usability.

4.4.9 i18n-2 internationalization framework
i18n-2 provides a simple internationalization framework used to implement tactic 4.3.14. The framework is a simple, light-weight framework which is easy to use and the language mappings are easy to maintain, improving the maintainability requirement 3.2.1.

-----

6 You can have a look at http://scotch.io/tutorials/build-a-real-time-twitter-stream-with-node-and-react-js to get you started with node.js and react.js.
4.4.10 MongoDB

The software architecture will use as persistence provider the MongoDB cross-platform document store. The reasons for this are that the application largely stores data whose state does not change (posts, appraisals, markEntries, ...).

Using MongoDB as a NOSQL document store results in a highly cachable persistence environment which is very scalable (requirement 3.2.2) and can result in high levels of performance (requirement 3.2.3). The default locking mechanism is a readers-writer lock which allows concurrent read access but only allows a single write operation, i.e. while one transaction is writing a document no other transaction can read that document or write to that document.

Scalability and performance can be further improved by using

- effective indexing [4.3.10],
- clustering with load balancing, and
- sharding.

Of these only the first tactic is to be used for the mini-project.

Reliability and availability (requirement 3.2.4) can be improved by using MongoDB’s support for replication. This is also out-of-scope for the current version of the Buzz system.

4.4.11 Mongoose ODM

Mongoose is an open-source object data modeling (ODM) environment that wraps the native Node.js MongoDB driver. It supports

- automatic object to document mappings (tactic 4.3.7),
- enforces contracts and structure through validation (tactic 4.3.4), and
- provides connection pooling to improve scalability through connection pooling (tactic 4.3.9).

4.4.12 Resourceful database abstraction layer

It is not required for the mini-project to use Resourceful as a database abstraction layer in order to improve flexibility and deployability of the application. This is expected to be addressed in a later version of the system.

4.4.13 node-cache caching framework

The system will use the node-cache caching framework to realize tactic 4.3.8 in order to improve scalability and performance. All documents which are not expected to change frequently (which is most document within the Buzz system) are to be cached using node-cache.

4.4.14 restify REST framework

The system will publish the user services as REST-based web services using the restify REST framework (tactic 4.3.16) in order to address the integrability requirements and to facilitate the development of a mobile client (e.g. Android client) at a later stage. This should include the publication of WADL contracts.
4.4.15 **ldapjs LDAP adapter**

*Buzz* will integrate with the CS-LDAP database using the *ldap.js* JavaScript LDAP framework. The framework is developed in pure JavaScript and is easy to use.

4.4.16 **NodeMailer email client**

*Buzz* will make use of the *NodeMailer* JavaScript email client to send emails to a mail server. In the current deployment the CS mail server will be used.

4.4.17 **Scribe.js logging framework**

*Buzz* will use *Scribe.js* as a light-weight, simple logging framework to implement tactic 4.3.17. Logging functionality will be woven into the business logic using *node.aop*. *Scribe.js* does provide a web front-end to make the system monitorable and auditable by humans.

4.4.18 **jsreport reporting framework**

The *jsreport* reporting framework will be used to provide a simple, yet powerful and flexible reporting framework for the application.

4.5 **Infrastructure**

Figure 52 shows the infrastructure of the *Buzz* discussion board. The application is deployed across 4 layers including an access layer, a services or business logic layer, an infrastructure layer and a persistence layer.
Figure 52: The infrastructure of the Buzz software architecture.

The diagram shows the components used to address architectural responsibilities including those used to implement tactics through which the quality requirements for the application are addressed.
4.6 Development architecture

The development architecture is the architecture within which the development of the system will be done. The architecture has been designed to support qualities in the development process itself. This includes qualities like reliability, development velocity and integrability of separate development teams.

4.6.1 Version control

Each of the two projects will have its own git repository which is to be created by the top-level integration team and which will be used by all the teams for that project. Any new feature or bug fix should be developed in a new branch which is only merged into the trunk once the feature or fix passes its unit tests.

4.6.2 IDE

Each developer has the freedom to choose whichever IDE they prefer. All builds are driven by npm and should be IDE independent. Some IDE’s for node.js include

- Eclipse Node.js,
- WebStorm, WebStorm\(^7\),
- NetBeans, . . . ,
- as well as editors like Atom and SublimeText.

4.6.3 Builds

In this project the node.js package manager (npm) will be used to build the project artifacts. No builds should be IDE-specific, though npm builds can be integrated into an IDE. It must be easy to build the system from outside any IDE.

 npm projects have a package.json project descriptor\(^8\) which is similar in content to a Maven POM. It specifies

- an identifier for the artifact being built made up of an
  - an artifact name (e.g. buzzSpaces) and
  - a version
- the artifacts your project depends on,
- the project’s repository,
- the GitHub repository for your project,
- configuration information for your project, and
- metadata like a project description, developer information and so on.

\(^7\)You can register as a student at https://www.jetbrains.com/student/ to be able to use the IDE free of charge.

\(^8\)see http://docs.npmjs.com/files/package.json.
Each team including all development teams and all integration teams will have their own package.json project descriptor. Any new version of the code which compiles and passes the unit tests should be published in a npm registry which is accessible by public all teams of that project. This will facilitate the integration of the different teams and enable the teams to effectively collaborate on building the full system. It is the responsibility of the top-level integration team to put in place the npm repository for their project.

The standard 3-level semantic versioning convention should be used to improve maintainability and reliability, e.g. version 3.1.2. In this convention

- a 3’rd digit change represents a patch release containing a bug fix which should not require any changes to the code using the dependency.
- a 2’nd digit change represents a new version with additional features which do not break any existing code, and
- a 1’st digit change contains changes which break existing code, i.e. which are not backward compatible.

The dependency declaration should specify which changes you are prepared to accept. If you are prepared to

- accept patch releases only, use either 3.1.x,
- accept also new features, but nothing which breaks existing code, use either 3.x,

When building an artifact you can use

- npm version patch to increment patch number for package you are working on, followed by npm publish to publish the artifact in the repository,
- npm version minor to increment minor version number, and
- npm version major to increment major version number

4.6.4 Unit testing

Unit testing will be done using Unit.js which supports dependency injection.

4.6.5 Integration testing

The integration teams will do the integration tests across the artifacts they are integrating. At the top level, the integration testing will use no mocking. The mid-level integration teams will only use mocking for those components they are not integrating, i.e. they will use the actual components for all those sub-projects they are integrating.

---

9Sinopia can be used to host a public npm repository.
4.6.6 Documentation

API documentation should be generated and centrally hosted in order to improve development speed and maintainability (requirement 3.2.1) of the application. All code should be annotated with JSDoc documentation metadata. The integration teams should ensure that for each of the two projects the API documentation is available to all development teams.

4.6.7 Bug tracking

The projects will use GitHub’s issue tracker for bug tracking.