COS 301

Architectual Requirements for Buzz System

Group 4B: Phase 2

Authors:
u11013878: Jaco Bezuidenhout
u12204359: Renaldo van Dyk
u13008219: Ivan Henning
u10126229: Matthew Nel
u13310896: Shaun Meintjes
u12035671: Zander Boshoff
u13019989: Hugo Greyvenstein
u11210754: Mpedi Mello

Supervisor:
Ms. Stacey Omeleze

https://github.com/RenaldoV/COS301_Group4_B
March 11, 2015

Contents

Chapter 1

Architecture requirements

1.1 Architectural scope

1.1.1 Architecture requirements

Persistence infrastructure

To all data will be saved in either a database or on a file server:

- Database:
  - All plaintext data (inc Code, Tex, etc...)
  - All links between modules
- File Server:
  - Images
  - Documents (PDF, docx, etc...)
  - Compressed Files (zip, tar, tgz, etc...)
  - Executables (exe, jar, etc...)

1.2 Quality requirements

1.2.1 Quality requirements

Critical

- Scalability:
  The system is expected to have at least a 1000 users and can be expanded as specified by the client. Buzz Space is meant to encourage discussions on multiple topics at the same time by a large amount of users so the system needs to be able to handle many concurrent users too.
- Maintenance and Audit-ability
  The system should be monitor-able and audit-able, to enable lecturers to monitor the discussion boards and to see who posted what on the Buzz Space. Lecturers should be able to evaluate and monitor each user to be able to give the user a participation mark for the discussion board. Lecturers should be able to prevent irrelevant posts and discussions and should be able to see who were part of the discussion and notify them that it has been removed and should then be able to give them a warning.

Important

- Usability
  The system must be as usable as possible so that users are encouraged to use it as much as possible. The extent to which the system can be made usable is affected by the priority of scalability and audit-ability because if the system is made scalable so that it can be used by all students at the same time, it is impossible to cater for 2000 students' needs and make it 100% usable for the mass. Usability is marked important because the participation of students can be greatly affected if the system is unusable and students will then lose marks.
- Integrability
  One of the functional requirements is that the system be integrate-able into other systems as a discussion board. The MVC pattern can be used to achieve this. The system will be split up from the view and the data so the data can be changed easily to suit the systems that is integrating this system into it.

- Security
  Security is important because the evaluation of users' participation is involved. Not all of the users should be able to access these sensitive information and therefore certain users will be authorized to do certain actions, where other users will not be allowed to. Only users who are registered to the Buzz Space gave access and will be able to partake in the discussion boards. If the system is very secure, it will help to ensure Reliability and Availability.

1.3 Integration and access channel requirements

1.3.1 Integration and access channel requirements

Integration requirements

The Buzz System should integrate seamlessly with the following existing systems:

- LDAP as an user authentication service
- CSWEB as a route to the Buzz System and as a summary of certain Buzz Spaces
- CS Department mailing server
  - Creating threads/replying to threads by email
  - Notifying users on changes/updates/replies to threads/spaces

Access Channels

- Human Access
  - Thin Client
    A client will access the system using a modern browser that supports HTML5, CSS3 and Javascript
    * All necessary resources need to be downloaded from the server.
    * Will use a RESTful API
    * User Interface can be scaled according to device/size of the window to ensure readability and usability. E.g. Bootstrap.
- System Access
  Other systems will access the system via the API provided as this system will act as a plugin that can be used on any other system that wishes to add a discussion board. RESTful will be the main approach to the API.

1.4 Architectural constraints

1.4.1 Reference architecture

JavaEE

JavaEE should be used as the system architecture.

RESTful

A RESTful approach will be used in creating the API to allow a lightweight platform for Thin Clients to access the system.

1.4.2 Particular technologies

Programming Languages

Java, HTML5 and CSS3

Framworks

JavaEE

Protocols

LDAPS, SMTP, IMAP, HTTPs, WebSockets

1.4.3 Operating systems

Server

Linux will be used on the server

Client

Any operating system with a modern browser that supports HTML5, CSS3, Websockets and Javascript

Chapter 2

Architectural patterns or styles

2.1 Architectural patterns

2.1.1 Loose Coupling

Description

Loose coupled systems only allow accessing of resources through a proxy or accessing a pool of resources instead of a single resource. Loose coupling refers to the indirect access to external resources.

Quality Requirements it enforces

- Scalability
  This loose coupling of systems via API's makes your application more scalable because systems can off-load part of your processing to other systems that are independently scalable.
- Maintainability
  This loose coupling of systems via API's makes your application more maintainable because bugs and problems are enclosed in the module and not in the system as a whole.
- Integration
  Because loose coupled modules should work independent, it's very easy to integrate a loose coupled system into another system by using an adapter module.
- Security
  Because loose coupled modules are only accessed through the API, all databases/servers are only accessed by the module and not directly by the user.

2.1.2 Blackboard Pattern

Description

By allowing multiple processes such as creating a root thread or rating a user to work closer together on different threads, polling and reacting if needed so that less time is wasted by running processes in parallel.

Quality Requirements it enforces

- Scalability
  It is easy to achieve scalability with the blackboard pattern accross a grid of processors subject to having a scalable implementation of the blackboard itself.

2.1.3 Prototype pattern

Description

By deep cloning objects to store their data in the audit log if any changes occurred, without keeping a reference to the original data and exposing it.

Quality Requirements it enforces

- Maintainability and Auditability
  The prototype pattern makes clones of the old values(deep cloning) and then extracts the data needed in the audit trail.

- Security
  Because the prototype pattern works on a skeleton based design in some sense it helps with security by limiting manipulation possibilities.

Chapter 3

Architectural tactics or strategies

3.1 Strategies

3.1.1 Scalability

- Addition of system resources.
- Store data on client side for less traffic on server.
- Loose-coupling sub systems tied together by API's.

3.1.2 Maintainability and Audit-ability

- Model-View-Controller Strategy when approaching the user interface for easy maintenance.
- Use separate table for the audit log.
- Log all user Activity.

3.1.3 Usability

- Model-View-Controller Strategy when approaching the user interface.
- Use as many as possible known icons and symbols to bridge the language barrier.
- Usability testing by asking non tech-savvy individuals to use the system.

3.1.4 Integrability

- Using a separate adapter class/model to make the system.
- Loose-coupling sub systems tied together by API's to allow easy access from other systems through the API's.
- Using the prototype pattern to create the default database layout with regards to your configuration. specifications

3.1.5 Security

- Model-View-Controller Strategy when approaching the user interface.
- Loose-coupling sub systems tied together by API's to limit access only to the allowed API calls.
- Security testing by running a hacking challenge on the system.

Chapter 4

Use of reference architectures and frameworks

4.1 Reference Architectures

4.1.1 Java-EE

Java-EE is based on a layered architectural pattern:

- Access Layer
- Business processes layer
- Persistence Layer

Effect on quality requirements

- Scalability
  Positive effect when reusing resources, caching and clustering. It also allows for load-balancing.
- Maintainability and Audit-ability
- Usability
- Integrability
- Security

4.2 Frameworks

4.2.1 Apache Geronimo

- By using clustering technology to provide a scalable and highly available platform for the system.
- Geronimo only provides LDAP viewing capabilities. This enforces security because editing of the LDAP database isn't available.
- Allows for RESTful Web Services.

Chapter 5

Access and integration channels

5.1 Access Channels

5.1.1 Human access channels

Thin client

All necessary resources need to be downloaded from the server.

Responsive design

User Interface can be scaled according to device/size of the window to ensure readability and usability. E.g. Bootstrap.

Software needed

-Any modern browser that supports HTML & JavaScript E.g. Google Chrome, Mozilla Firefox.

5.1.2 System access channels

Other systems will access the system via the API provided as this system will act as a plugin that can be used on any other system that wishes to add a discussion board. RESTful will be the main API

5.2 Integration Channels

5.2.1 Authentication and user list

- LDAP
- MySQL
- PAM Auth
- MongoDB
- SQLite
- Plaintext (text file, csv, etc...)

5.2.2 System Access and Integration into existing systems

- iframe adapter
- div adapter
- Plugin by using Buzz API

Chapter 6

Technologies

6.1 Database

6.1.1 MongoDB

6.2 Servers

6.2.1 Linux Virtual Servers

For scalability

Application Server

- NginX
- JavaEE
- sshd
- failtoban
- ldap-client

Distributed Filesystem Server

- AFS (Andrew File System)
- sshd
- failtoban
- ldap-client

Authentication Server

- Apache Geronimo (Java-EE Framework)
- slapd (LDAP Server)
- sshd
- failtoban
- ldap-client

Backup Server

- amanda (works over ssh)
- sshd
- failtoban

Database Server

- MongoDB
- sshd
- failtoban
- ldap-client

Email Server

- Exim4 (SMTP)
- Courier (IMAP/POP3)
- failtoban
- SpamAssassin
- ldap-client

6.3 asdf

6.3.1 asdf