Architecture Requirements for the Buzz System

Git: `https://github.com/FrikkieSnyman/Phase2_Group3B`

COS301 Group 3b
Andreas du Preez 12207871
Jason Evans 13032608
Sebastian Gerber 12213749
Baruch Molefe 12260429
Kyhle Ohlinger 11131952
Renette Ros 13007557
Michelle Swanepoel 13066294
Frikkie Snyman 13028741

March 2015

Contents

1 Architectural Requirements

1.1 Scope of Architectural Responsibilities

Database as mode of persistence:
In the scope of the BUZZ system and taking into consideration the type of data the system should store, we have concluded to make use of a Relational Database Management System (RDBMS). The BUZZ system will store only structured data with strong relations between content (e.g. Threads and Social Tags). Persistence of data is also closely related to the auditibility of the BUZZ system, hence deleted threads will not be removed completely from the database, but instead be archived (marked as hidden) for possible later retrieval.

Communication:
The BUZZ sytem will primarily be a web-based application accessable through any web browser, hence our focus on communication between the servers and the application should take place over **HTTP**(Hyper Text Transfer Protocol) requests and responses. This will also ensure that our system is more adaptable when there is a possibility of expanding the application system to support an Android based mobile application.
Any request that the server receives and processes should then be replied to the application (processed information should be reported back) using **JSON**(Javascript Object Notation). Our team has chosen this technology as our response communication based of the possibility of expanding the system to support other technologies such as mobile applications. When the response is sent as JSON, there is more effective **SoC** (Seperation of Concern) and all access channels that requires access to the server will effectively have the same information passed back to them.

1.2 Architectural Responsibilities

1.3 Quality Requirements

- **Scalability:** The scalability of the system is not such an important requirement.

  The system will mostly be run on large servers capable of servicing an entire university and as such will not need to be extremely scalable.

- **Performance Requirements:** Performance is a very important requirement as the system must be able to accomodate thousands of users and threads.

  The system will mostly be run on large servers capable of servicing an entire university, however performance must still be considered.

- **Maintainability:** Maintainability is very important to the system as any failure of a key component could result in loss of data or the inability to use the system.

- **Reliability and Availability:** The reliability and availability of the system is important for all users, i.e. The students, tutors, lecturers, and administrative staff.

  If the system is unavailable, then users would not be able to access the required information on the system. This could have serious implications on all users. It is thus imperative to minimise system downtime, thus maximising system availability.

- **Security:** The security of the system is important for all users, i.e. The students, tutors, lecturers, and administrative staff.

  The purpose of security is to protect the information stored in the system, whether it be the systems information or user data, and prevent unauthorised access to and/or modification of the information.

- **Monitorability and Auditability:** The system will be monitored by the administrative staff and users that are specifically assigned the role of maintenance.

  This will help ensure that users abide by the netiquette and plagiarism policies.

- **Testability:**
- **Usability:**
- **Integrability:**

1.4 Architecture Constraints

2 Architectural Patterns or Styles

3 Architectural Tatctics or Strategies

3.1 Scalability

if scalability is desired at some point it could be achieved by producing a light version of the system, which could be used on smaller systems which could be achieved by:

- Removing features not required by private users such as simplifying administration of users.
- Limiting the number of users of threads created to reduce overhead on the system.
- Remove some security or authentication features which would likely not be need on small, private servers.

3.2 Performance Requirements

Performance can be enhanced in the following ways:

- Reducing overhead by having unimportant processes such as profile editing suspended in times of high usage.
- Archiving old posts while using faster access storage for newer or more active threads.
- Removing very old posts to reduce the storage requirements of the system.
- Prioritize the requests of users with higher priveleges such as lecturers and administrators.

3.3 Maintainability

Maintainability can be achieved by:

- Having up to date backups of the data in the system to be used in the event of data loss due to hardware failure.
- Having the system modularised in such a way that one component failing will not affect other components.

3.4 Reliability and Availability

Reliability and Availability of a system is essential, this could be achieved by:

- Identifying ways to prevent system failure, and if the system does fail, have measures in place to start a failover, so that the system is still accessible.
- Detecting if there are problems with the system, in order to do maintenance on the system before the system fails.
- Identifying ways to recover from system failures, e.g. have backups and rollback functionality so that no data is lost.
- Identifying ways to handle the system when external systems, to which the system is connected, i.e communication networks, external databases,etc. are unavailable. This could be done by having some sort of offline system functionality, or having ways to switch between various external systems.

3.5 Security

In order to enforce security:

- The system should enforce authentication and authorization or users to prevent spoofing of users identities.
- Input validation is important in preventing damage caused by malicious input.
- Sensitive data should be encrypted and user activity, i.e. Guest and Authorised users, should be monitored to prevent loss or damage of data.
- The system should log all user interaction with the system, this would be beneficial when auditing the system.
- The system should have multiple safe guards in order to protect access to data.
- System timeouts could be considered, in the unlikely event of DOS or DDOS attacks.

3.6 Monitorability and Auditability

To help with the Monitorability and Auditability of the system:

- Track all changes made by all users.
- Any infringement of these policies should be captured/logged for later use by the administrative staff.
- The audit logs would be made accessible to the administrative staff through specific requests to the system.

3.7 Testability

3.8 Usability

3.9 Integrability

4 Use of Reference Architectures and Frameworks

5 Access and Integration Channels

To facilitate and simplify the communication between the BuzzSystem and various access channels all of these access channels should send requests using the http protocol. A specific module in the Buzz System should handle all http requests and return JSON objects containing the needed data from other modules. The structuring and responses of these http requests should be clearly documented enabling the easy addition of more access channels.

5.1 Human Access Channel - Website

The main human access channel for the Buzz System will be an web-based front end. It will be referred to as BuzzWeb in the rest of this document for easy distinction between it and the BuzzSystem.

5.1.1 Requirements

- BuzzWeb should be cross-browser compatible.
- BuzzWeb should be viewable on devices of different size using responsive web design.
- It should conform to the newest HTML5 and CSS3 standards.
- Techniques like Ajax should be used to submit content and periodically fetch new content from the server without refreshing the whole page.

5.1.2 Technologies

BuzzWeb will use several popular web technologies:

- HTML5
- CSS3
- JSON
- JavaScript:
  - Jquery
  - JqueryUI
  - Google Code Prettify
  - Ajax
- Bootstrap for responsive design

5.1.3 Protocols

The main protocol used to communicate with the BuzzSystem will be http, preferably over an encrypted connection (so https).

This will happen in two ways:

- The user's browser will send and http GET request to a specific page. The BuzzSystem will respond with static html content.
- BuzzWeb will send an asynchronous http POST request to the server which will respond with an JSON object. The client-side JavaScript will parse these JSON objects and create and return applicable html where necessary.

5.2 Human Access Channel - Smartphone Apps

Creating an Smartphone Apps for the Buzz System is not part of this project's scope, but these apps will also be able to communicate with the BuzzSystem using http requests with the same structure as those BuzzWeb uses.

5.3 System Access Channels

There are no system access channels that form part of the scope of this project at the moment, but any system access channels should also use the http protocol to communicate with the BuzzSystem. An example might be a later integration of the BuzzSystem with the Department of Computer Science's marking system.

5.4 Integration Channel - Authentication

The BuzzSystem will integrate with the Computer Science ldap repository to authenticate users and obtain user roles and module information.

5.4.1 Technologies and Protocols

The LDAP (Lightweight Directory Access Protocol) Protocol will be used to obtain user and module information from the Computer Science Department's ldap repository.

5.5 Integration Channel - Database

The Buzz System will integrate with an Relational database to store its content.

5.5.1 Technologies and protocols

The system will use MySQL and the MySQL JDBC driver or a siliar databse system.

6 Technologies