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

1.1.1 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.

1.1.2 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 Quality Requirements

1.2.1 Scalability:

Scalability is of extreme importance.

No assumptions can be made on the strength and size of the servers that the BUZZ system will be hosted on and hence need to be scalable at any point in time. We also have to take into account that there will be a small margin of growth in users as the years go by since the University allows for this growth (an estimate of 10%) in their student admission numbers.

1.2.2 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.

1.2.3 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.

1.2.4 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.

1.2.5 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.

1.2.6 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.

1.2.7 Testability:

Testability is of medium importance in the system. The system needs to be tested before it goes online.

1.2.8 Usability:

User experience is an important aspect in the system. The users should be able to easily identify all the elements of the system as well as be able to use them with ease.

1.2.9 Integrability:

Integrability is of medium importance. It will be used for security reasons and portability reasons.

1.3 Architecture Constraints

- Constraints regarding technologies are discussed in section **??**
- Constraints regarding access channels (e. Web front-end, Android App) is discussed in section 5

2 Architectural Patterns or Styles

Three-tier Model: The architectural responsibilities that need to be addressed for the BUZZ system to be fully operational can be implemented in a multi-tiered fashion. This can be referred to as the model-view-controller approach or the three-tier approach which consists of persistence, logic/processing/communication and presentation. This approach is ideal for the BUZZ system as it is an on-line forum web application that will make use of each tier for related aspects of the system. This approach allows various users to make use of BUZZ through various clients such as a smartphone and a desktop computer. An example to highlight an advantage of using the three-tier system approach can be a user being able to access BUZZ on various devices. The model and the control layer will remain unchanged while the view will have to be modified to accommodate these clients. This setup will ultimately help with integration with various systems and it will help with portability between various clients.

REST: Since the BUZZ system will be in the form of an online discussion forum, a constant connection between server and client is not needed. Typically a user will request an action to be performed or simply request a page for viewing. A user may spend time only reading a thread and not actually utilize the network in which case the server should only wait for a request, deliver it, and then wait again. In this way we can us the REST technique to lighten the burden on the server side and free up unnecessary network traffic.

3 Architectural Tactics or Strategies

3.1 Scalability

When 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.
- Implementation of load balancing strategies.

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

What could be tested for:

- User profiles should be successfully created and have the correct information in them.
- Users privileges correspond to their user rank.
- Security breaches.
- All services provide the correct output if given certain input.

3.8 Usability

Participation will increase if the interface is easy to understand and easy to use.

A navigation bar needs to be implemented so that the users can easily navigate to where they want to be.

Readability is also a very important aspect in the system. Using a font that is easily readable can greatly increase users experience. Choosing the correct colour scheme is also important.

Users should be able to:

- Know how to use the system without any serious assistance.
- Learn how to use new concepts if the system was to be upgraded.

- Remember how to use the system after the first use.
- Like the system.
- Give feedback.

3.9 Integrability

The system will need to be easily integrable with other database software. If the current database software fails then the system will need to be able to switch database software in order to continue logging all incoming and outgoing data.

If the web service fails, then the system must be easily portable to another web service (different versions of software).

The abstract factory pattern can be used to easily port the system to another environment.

4 Use of Reference Architectures and Frameworks

Some appropriate API's already exist which suitably complements the implementation of the software.

4.1 Syntax highlighting

Seeing as ons of the priority features include posting code snippets in the BUZZthreads, the google-code-prettify (`https://code.google.com/p/google-code-prettify/`) is to be used in order to automatically implement syntax highlighting of the aforementioned code snippets when they are posted. The API supports a wide variety of languages, including C and friends, Java, Python, HTML, CSS, JavaScript and many more applicable languages.

4.2 Translation for pluggable requirements

The application must be pluggable. Using the google-translate-api (`https://cloud.google.com/translate/docs`) automatically translates the content that is found on the application. This means that the application is not restricted to certain language speakers, and enhances its pluggability.

4.3 Plagiarism check

PlagScan API (`https://api.plagscan.com/guide`) is an open-source API that handles plagiarism checks by sending the data to be checked as a POST request. The response sent is dependant on the configuration, and can be set to be received as XML data, and some methods can be received as either plain binary data or HTML. In order to use this API, one must first register for a PlagScan Pro organization account, after which an API key must be generated. The API can also be set up to define from which IP ranges requests can be sent using the generated API key.

4.4 Fonts

The front end of the application must be user friendly. This can be achieved by using the Google Fonts API (`https://developers.google.com/fonts/`)

4.5 Netiquette

An auto-moderator bot can be implemented using already existing open-source API's. Specifically the AutoModerator programmed for Reddit (`https://github.com/Deimos/AutoModerator`). This implementation is needed to be able to identify whether or not users have violated any netiquette rules in their posts. Note, that it would not be sensible to fork from the Reddit AutoModerator but to use it as a reference in what the appropriately implemented auto-moderator's functionality should stretch to.

5 Access 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 BuzzWeb

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

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 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.

5.1.3 Technologies

The technologies that will be used for BuzzWeb is discussed in Section **??**

5.2 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.

6 Integration Channels

6.1 Computer Science Website

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

6.1.1 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. The system will also connect to the CS-websites MySQL database.

6.2 Database

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

6.2.1 Technologies and protocols

The system will use MySQL and the MySQL JDBC driver or a similar database system.

6.3 Buzz System Front End

The Buzz System needs to integrate seamlessly with BuzzWeb as this will initially be the only human access channel to the system. It is discussed in more details in subsection 5.1.

7 Technologies

7.1 Technology Constraints

The following points explain certain architectural constraints specified by the client.

- **JavaEE**
  Java Enterprise Edition is a computing platform that provides an API and runtime environment for developing and running secure network applications which are scalable and reliable. This platform incorporates a design based largely on modular components running on an application server. The software running on JavaEE is developed in Java (primarily). JavaEE is the reference architecture to be used.

  The software of the system should be using JavaEE. This will be used on the server side.
- **JPA**
  Java Persistence API will be used to describe the management of relational data in the server of the system.

  This will be used on the server side.
- **JPQL**
  Java Persistence Query Language is part of the JPA specification. This will be used in the server to make a query for information stored in a relational database.

  This will be used on the server side.
- **JSF** JavaServer Faces is part of Java EE and is used to build component-based user interfaces for web applications. It was specified by the client that this should be used.

  This will be used on the server side.
- **HTML**
  HyperText Markup Language will be used to specify the structure of the web documents which will be used to present the application to the user.

  This will be used on the client side.
- **Ajax**
  JavaScript and XML will be used together to send and receive data to and from the server, asynchronously.

  This will be used on the client side.

7.2 Other technologies

The following points explain other technologies that will be used but are not constraints.

- **CSS3**
  Cascading Style Sheets will be used to describe the look and formatting of the HTML documents on the client side.
- **JSON**
  JavaScript Object Notation could be used to transmit data between the server and the actual web application.
- **JavaScript:**
  - **Jquery**
    jQuery is a JavaScript library which will be used to simplify the client-side scripting of HTML.

  - **jQueryUI**
    jQuery UI is a set of user interface interactions, themes, etc. which is built n top of the jQuery JavaScript Library.
  - **Google Code Prettify**
    Google Code Prettify will be used to highlight snippets of source code in HTML pages.

- **Bootstrap**
  This framework built in CSS and JavaScript can be used to lay out a fully responsive website for devices of various different sizes, easily.

- **ObjectDB**
  This an object database specifically for Java. It does not provide its own API, thus JPA (mentioned earlier in "Technology Constraints") will be used as the API. Object databases are generally faster and more efficient and should be used when high performance is needed and complex data is presented.