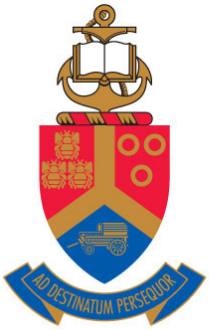

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science

COS 301 - Software Engineering

COS 301 - Mini Project

Author:	*Student number:*
Sean Hill	u12221458
Liz Joseph	u10075268
Jessica Lessev	u13049136
Kale-ab Tessera	u13048423
Sphelele Malo	u12247040
Diana Obo	u13134885
Armand Piterse	u12167844

March 10, 2015

Architectural Requirements

Buzz Space link

For further references see gitHub. March 10, 2015

Contents

1 Change Log . . . 3
2 Introduction . . . 3
2.1 Purpose . . . 3
2.2 Project Scope . . . 3
3 Access Channel Requirements . . . 3
3.1 Human Access Channel . . . 4
3.2 Access Channels . . . 4
3.3 System Access Channel . . . 5
3.4 Architecture Patterns . . . 6
3.5 Intergration Access Channel . . . 7
4 Quality Requirements . . . 7
4.1 Critical Quality Requirements . . . 7
4.2 Important Quality Requirements . . . 10
4.3 Nice-To-Have Quality Requirements . . . 12
5 Integration Requirements . . . 15
5.1 Integration Channel . . . 15
5.2 Protocols . . . 15
5.3 API Specifications . . . 15
6 Architecture Constraints . . . 16
6.1 System . . . 16
6.2 User . . . 16
6.3 Time . . . 16
6.4 Technologies . . . 16
6.5 Architectural Patterns/Frameworks . . . 17
7 References . . . 17
8 Glossary . . . 17

1 Change Log

2 Introduction

Designing and implementing a system as the one in question is a vast amount of work and requires a lot of planning and thinking. All the functional requirements needed for the system were covered in the previous phase. In this phase all the non-functional requirements will be covered. These are all categorized as architectural requirements.

2.1 Purpose

The main purpose of this document is to decompose the system at hand and fully describe the decomposition in terms of subsystem responsibilities, the mapping of the subsystem to hardware and the dependencies among the subsystems. Policy decisions that need to be made is also described in this document. This included decisions such as access control, data storage and control flow.

The document will cover all the non-functional aspects of the requirements including:
the architectural scope comprising the responsibilities,
the quality requirements for the software system,
the access and integration requirements, and
the architectural constrains.

2.2 Project Scope

The scope of this phase is understanding the functional requirements covered in the previous phase and using them to build and derive the architectural requirements needed for the implementation of the system.

3 Access Channel Requirements

Logging in to the system is done over HTTPS POST, and we send the user's password as an MD5-encrypted string.

The user's login details are kept in an HTTP session so the user does not have to keep logging in every time he/she makes a request to the server.

HTTP sessions are invalidated when the user logs out.

Communication is done over HTTPS, the login credentials should be relatively safe from malicious sniffing of data between the client and server applications.

Common address redundancy protocol will be used to provide load-balancing as well as fail-safe redundancy.

The servers will share a virtual IP address, which will be floated between them if the main server fails or if there are load problems.

All CRUD actions that will make changes to the database contents will be intercepted at the services layer and logged automatically along with the id of the user making the changes, the status and the date. This will ensure auditability of the system.

Any actions that would delete data from the database will not destroy any data, but instead set a 'deleted' flag.

Users are assigned a status level and permissions are set by their status. These permissions will determine the features covered by the interface. This will ensure the users only access features for which they are authorized. For example a guest cannot comment on the posts.

Users are allowed to CRUD posts but not all features are available to all users. For example, privileged users can delete their posts whilst ordinary users cannot.

3.1 Human Access Channel

End users interact with the web client to submit and interact with application jobs and to avail themselves of scalable system tools.

Thin Web client:

We need a thin web client as it allows the client to be relieved of computation duties and offers more security. The data will be verified by the server.

3.2 Access Channels

Android client -which is the mobile application, is out of our project scope. It runs on android software architecture.

The Web Client -communicates to services layer through RESTful web services in the access layer. It provides a web-based human intergration channel.

Purpose:

- Make all user services from the business logic layer accessible through the web browsers.
- Capture information for the request of objects in the business logic layer and submit the service requests to that layer and render the responses back to the user.
- JSF-2 Java Server Facelets.
- The managed beans who manage the state for the user and issue the service requests to the services layer will do so.

- Access to the stateless session beans from the services layer is obtained through dependency injected.

Services:

- Web services
- RESTful web services
- RESTful web services induce desirable properties, such as performance, scalability, and modifiability and this enables services to work best on the web.
- Note: These services have access to services layer

Database:

- Relational Database.
- Theres a need for multiple database connections, with connection pooling.
- No need for explicit implementation.

3.3 System Access Channel

Systems administrators interact with the project to configure and build software installations on individual nodes, schedule, manage, and account for application jobs and to continuously monitor the status of the system, repairing it as needed.

The web-based component of the system will be implemented in PHP, AJAX and HTML.

Connections to the system will be made to the shared virtual IP of the web servers.

Floating the same virtual IP address between multiple servers in this manner will be managed by Ucarp. The databases will run in a Relational Database.

The technologies selected for the system are already in use within the client company.

These technologies include JavaEE, which we will use to run the web services, JPA, to manage the relational data in JavaEE, JPQL, make queries for the database, JSF to build component user interfaces using COBALT and AJAX.

Reasons to use COBALT:

- It is simple and flexible. Systems administrators can easly customize or replace individual components independently of others.
- Deployment is easier, since the required software and technologies are already up and running.
- It also provides maintenance benefits, since the system administrators will already be familiar with the technologies.

The system needs to offer support for up to a thousand to two thousand simultaneous users.

This will be achieved by means of multiple web servers running PHP, which is known to scale well and is used by many large websites.

MySQL can be configured to be highly scalable and available as the traffic flow increases.

PHP has many security modules available which can be used to provide presentation-layer authentication.

CakePHP is one of the PHP frameworks that we will use. It is a rapid development framework that makes coding in PHP much easier and efficient. It is also based on the MVC architecture which will fit well with our current architectural design.

Reasons to use CakePHP:

- Application scaffolding
- Code generation
- MVC architecture
- Supports both version 4 and 5 of php
- Flexible caching

3.4 Architecture Patterns

The Model View Control (MVC) architecture will be used in the system. This architecture allows change to the system's state (The Model) is completed through a common interface (The Control). (The View) is where developments are made to user interfaces and these changes are independent to the system providing extensibility at a low cost. PHPs implicit MVC features will be used to implement this architecture.

We chose MVC because:

- It encapsulates the interaction from the user and transforms those interactions (in the form of requests) into business logic that interacts with the system model to produce responses.
- A (multi-)layered approach encapsulated by MVC which enables the separation of business logic proving the need for sub-systems and internal and external APIs.

This architecture will be based on Service Oriented Architecture (SOA) reference architecture. It is a loosely-coupled architecture where services exist independently from each other. They communicate with each other by means of protocols and standards-based interfaces such as SOAP and XML Schema to define messages passed between objects. In this case SOAP would not work.

Services are deemed as individual objects so new services and functionality can easily be added to the existing system during progression.

Reasons to use SOA:

- It is flexible and agile.
- System is open to change and easy alteration
- Simplicity of implementation.
- Reuse of objects

3.5 Intergration Access Channel

4 Quality Requirements

4.1 Critical Quality Requirements

Scalability

Stakeholders:

Programmers involved in System maintenance.

Motivation:

This is critical for the specific BuzzSpace system because without scalability, the system would have to be changed every year to account for increasing numbers of students, and this is unacceptable. This also ties into maintainability, intregratability and performance.

Description:

Firstly, the system should be able to handle all registered students, lecturers, tutors and teaching assistants of the COS department. It should be possible to add more students as required, by the addition of system resources, but without making changes to the deployment architecture.

Stratagy:

- Make reasonable predictions about projections of expected growth based on the BuzzSpace requirements and using data from previous years.
- The design should also be flexible enough to handle increased loads until a system can be upgraded with additional resources, this can be done using Caching and other code reuse techniques.

- We could also slightly reduce the systems performance to ensure the code is scalable i.e. add users to system without actually making a change to the architecture.

Pattern:

- Flyweight can also be used minimizes memory use by sharing as much data as possible with other similar objects.
- Service Oriented Architecture (SOA) reference architecture should also be used because services are deemed as individual objects so new services and functionality can easily be added to the existing system during progression, allowing the system to be flexible and agile.

Reliability And Availability

Stakeholders:

Users of the website i.e. students, lecturers, tutors, teaching assistants and system administrator.

Description:

This entails the system having as minimal as possible down time and there being no bugs or hardware failures that will cause the system to be unavailable

Motivation:

Should the system be unavailable it means that stakeholders have no way of retrieving any information on the system. This could result in important announcements being missed and/or students being unable to get the help they need on a specific topic.

Stratagy:

- Redundancy in the system. This eliminates a single point of failure. So that if a component in the system fails, it doesnt cripple the entire system, because there other components that can carry the work.
- Failure detection as it occurs. Should the system run in to a fault the system maintainer should be notified immediately so he can rectify it.

Pattern:

Run-time Reconfiguration Pattern ensures the system is designed in such a way that it can be reconfigured without requiring redeployment or restarting the application. Hosting infrastructure sends notifications for any configuration changes. Application will

examine the change and apply them to relevant components. From this point onwards, the system will use the new configuration value in the application. This will minimize system downtime, thus maximizing availability.

Usability

Stakeholders:

The users such as students, clients, lecturers, staff, tutors etc as well as the system maintainer.

Motivation:

This requirement is classified as critical as it is critical for a system to be easy to use by the client. How the user interacts and communicates with the system is vital to the survival and continuation of the systems use. If the system is not usable, then users will not achieve what they wish to on the system and will refrain from using the system. This will lead to the system no longer being needed.

Description:

This quality is used in the context of the system being operational and usable by its users. It forms the foundation of the system. If this quality does not exist then the system its self will not exist.

Any user who is registered to a module that makes use of the system and is capable of using a basic discussion forum should be able to make use of the system without a user manual. Thus any user who has any background knowledge of social media and discussion forums should be able to navigate and understand the system with little to no effort. The system has to be designed and implemented in such a way that it has to be functional and the users find is usable. The same applies for the system maintainers. The system has to be implemented and designed in such a way that it is easy and simple to use by the system maintainers in order to maintain the system.

Stratagy:

In order to make a system usable, the system has to be user friendly and the user should be able to interact with the system with little effort. This can be achieved by designing the system in such a way that the activities the user can perform are limited and easy to complete. The interface its self should be easy to follow and understand. User manuals and instructions can also be provided.

Monitoring And Auditability

Stakeholders:

System administrators and maintainers

Description:

The activity of users being monitored and logged somewhere to be reviewed at some point. Action can then be taken by the administrator based on the logged actions.

Motivation:

This is an important requirement of the system as one of the core responsibilities of this system is to award user participation with points to increase their rank on the system. The system, therefore, needs to be able to log user activity so adjustments can be made to their ranking. /likewise with checking for netiquette and plagiarism infringements. Without the system being monitorable, none of this is possible.

Strategy:

For plagiarism and netiquette the system implements components that review all posts made by users. Should these posts need flagging the system does so and notifies the user and the buzz space administrator.

4.2 Important Quality Requirements

Security

Stakeholders:

The users such as students, clients, lecturers, staff, tutors etc, as well as the system maintainer.

Motivation:

Security is classified as impotant as it is an essential component of a system. If a system is not secure, it will be susceptible to unauthorized alterations, malicious activites and loss and corruption of information. Thus security is needed to project the system in order for it to continue running smoothly and assure clients of their information safety. Security plays a major role in this specific system in terms of privileges. Users' profiles have to be kept secure because each user has a different status on the Buzz space and with each status comes certain privileges. If someone gains access to someone else's

profile with higher privileges then they can alter things on the buzz space that should not be changed.

Description:

Security is the capability of the system to prevent accidental and malicious activities and actions external to the designed usage, and to prevent revelation or loss of information/ It aims to protect resources and prevent unauthorized modification of information.

This quality is used in the context of the system hiding necessary information from certain users and keeping information such identities private from the public. The system has to maintain accessibility rights and proper authorization. The system has to provide access to authorized users and deny access to intruders. Information on the system has to be kept safe and the system has to have a high level of integrity.

Stratagy:

Security can be achieved through processes such as identification authentication, certain users having more privileges than others. Information can further be encrypted where hash codes can not be retrieved. This aspect can be further used to improve other parts of the system such as encrypting all the information regarding logged activities of users.

Pattern:

Patterns such as the MVC (Model-View-Controller) can be used to implement security as all the different aspects of the system are handled separately, hence security is built into the system.

Integratability

Stakeholders:

Programmers involved in System creation.

Motivation:

This was selected as being important because the subsystems need to be integrated to have a stable Buzz System as a whole. It wasnt selected as critical because, this specific Buzz System's purpose is to allow users to comment and the department to get feedback from their comments.

Description:

Integration of the different subsystems (BuzzSpace, Authorization, Services etc) should be seamless because sophisticated integration channels and protocols will be used. This will ensure that the system as a whole can will be composed effectively via the different subsystems.

Stratagy:

- The Model View Control (MVC) can be used to achieve this. A (multi-)layered approach encapsulated by MVC which enables the separation of business logic, which will allow various sub systems to be integrated into one larger system.
- Loose coupling and Parallelism will also allow seamless connection between the different subsystems.

4.3 Nice-To-Have Quality Requirements

Performance

Stakeholders:

The users such as students, clients, lecturers, staff, tutors etc, as well as the system maintainer.

Motivation:

This requirement is classified as nice-to-have as performance plays an important role when it comes to a system such as the one at hand but does not play a vital role. If a discussion forum is slow or lacks in performance in certain areas, then the system itself will still function and continue to work just at a slower pace. This however does not affect the system in a major way.

Description:

This quality is used in the context of the system running smoothing and upholding a high level of performance.

The system should not lack in performance. It should flow smoothly and quickly between users activities and actions. The system should display appropriate and correct information at all times. The system should not have lag and should have a high speed of performance. When feedback and statistics are requested, the system should generate them in a timely fashion. The system should not hog essential computer resources such as RAM, memory or CPU time.

Strategy:

There are many ways in which good performance can be achieved. The system should be implemented in such a way, that unnecessary memory space is not used up. The system should guarantee that no infinite loops and glitches will be encountered where by the CPU time will be consumed and the system will struggle with performance. Mechanisms such as threading should be used in order to make processing faster and thus better the performance. Concurrency is also vital to improve performance. To name but a few.

Pattern:

Several patterns can be explored when it comes to improving performance. Flyweight design pattern reduces memory consumption, The Proxy pattern can be used for speed optimization and the Bridge pattern can change the implementation of an abstraction on the fly by always picking the most efficient one.

Maintainability

Stakeholders:

The development team and system maintainers

Description:

This requirement enforces that the system be easy to maintain and/or update/modify without compromising the security and integrity of the site. It should be possible to modify/maintain the parts of the system without modifying the entire system.

Motivation:

Any software will eventually need to be updated or fixed at some point. Since this is an issue we will definitely run into at some point, it would be convenient if we prepared for it. Having a maintainable system means this task will be done much faster and with much less effort. Also we avoid the risk of compromising other quality requirements like security or system integrity.

Strategy:

The system should be separated into different components that make use of each other to operate. Components, however, should not depend on the internal details of other components. It should simply request a service and take the response. This ensures that modifying components internal information wont mean having to change all components that make use of it.

Pattern:

The Pipe and Filter pattern is ideal for this. The various components of the system will act as the filters. Since filters deal with input they were given (They are not concerned with where the data comes from) and produce output without being concerned about any of the other filters, it becomes easy to modify other filters without affecting any of the others. This makes maintenance easier, especially if, for example, it is only parts of the entire system that need modification.

Testability

Stakeholders:

Programmers involved in System creation.

Motivation:

For this system in particular, testability is a nice-to-have, because it is not a critical quality requirement or directly related to any critical quality requirements. It is simply an add-on which is not necessary but would help if the system made use of it.

Description:

The system should create software that is easy to test in various test contexts. Futhermore, the system must also produce faults that are easy to find and eventually solve. This will ensure

Strategy:

A combination between Static and Dynamic Testing needs to be used. Static testing (manual checking of the code and documentation) is more of a preventative method and this will ensure at the initial state the system is in a good condition. From then, Dynamic testing needs to be used to ensure the system has no bugs that eluded Static testing and to ensure the system remains in a good state.

Pattern:

Loose coupling and Parallelism can be used so that different sections of the system, can be tested individually and this will make it easier to locate bugs and other errors.

5 Integration Requirements

5.1 Integration Channel

CS Website - Will be integrated to work with the BuzzSpaces, each CS module having its own BuzzSpace.

LDAP - This the Computer Science Data Source Adapter, which is used to connect to and retrieve necessary information from the external Computer science database.

Buzz Moderation - Is a plug-in that moderates the use of language and spam detection on the Buzz system.

Buzz Notification - An add-on that will be used to notify users via email, when there are status changes, moderation failures and appraisals which have been received.

Buzz Reporting - An add-on that a user or lecturer can use to define and generate their own reports.

RDMS - A relational database management system will be used to store and retrieve all data/information that is directly connected to the Buzz system.

Both client- and server-side scripting to facilitate database access and algorithms.

File creation and storage for the communication of information between modules.

5.2 Protocols

HTTPS - Is the transfer protocol for the web interface. Main reason for using this rather than just HTTP is that HTTPS is more secure.

SMTP - Is an emailing protocol that will be used for sending email notifications to users with the Buzz Notification.

TCP/IP-is the transport layer used to provide devices with a connection to a router and thus also the internet.

5.3 API Specifications

Buzz uses a REST API. This means that the API consists of a set of resouces which you can request or manipulate using the standard HTTP methods GET and POST.

This request system might be used to link user graphs to other websites, perhaps work profile sites like linkedIn.

6 Architecture Constraints

6.1 System

The BuzzSpace has to be integrated into the CS departments website.

Reason: The users credentials from the CS LDAP will be used for authentication purposes in the BuzzSpace e.g. to register on the BuzzSpace to login.

6.2 User

- Only users registered for a module from the CS department will be allowed to participate in discussions on the discussion forums.
- Users who are registered as guests on the BuzzSpace will only be allowed to read and observe the discussion forums.

  Reason: To manage who has access to the BuzzSpace.

6.3 Time

If a user is logged in and remains inactive for more than 30mins the user will have to login again to post on the forum.

Reason: To ensure security for user Authentication.

6.4 Technologies

- HTML

  To create the skeleton of the BuzzSpace's front-end to enable users to login/logout create forums and participate in discussions.
- Javascript and AJAX

  Used for the BuzzSpace's front-end to verify the login details of each user and will keep track of user login and particiaptions on discussion forums.
- PHP

  Allows the front end to communicate with the CS LDAP database to access user credentials for authentication purposes for logging in to the BuzzSpace.
- JavaEE
- JPA and JPQL

6.5 Architectural Patterns/Frameworks

- Layer (object-oriented design)
- MVC (Model View Controller)
- Peer-to-Peer Network
- Services Oriented Architectures

7 References

PIETERSE, V. 2015. COS301: Mini project Buzz. In: Lecture notes issued online. University of Pretoria. Pretoria, South Africa

8 Glossary