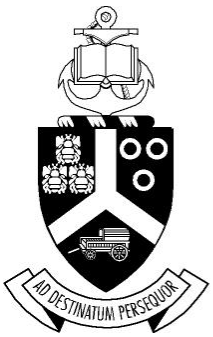

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science

COS 301 - Software Engineering

COS 301 - Mini Project

Author:	*Student number:*
Hanrich Potgieter	u12287343
Tshepiso Magagula	u12274195
Elana Kuun	u12029522
Jaco-Louis Kruger	u13025105
Sifiso Shabangu	u12081622
Kgomotso Sito	u12243273
Neo Thobejane	u11215918
Michael Nunes	u12104592

March 11, 2015

Architecture requirements

Buzz Space Discussions/Mini Project

Version: Version 0.2 Alpha March 11, 2015

Contents

For further references see gitHub or got to the link https://github.com/MichaelNunes/Cos301-phase-2-group-4-a

1 Architecture requirements

1.1 Quality Requirements

Scalability

There is a anticipated increase in entry volume. We are expecting an exponential increase in students using the system. It is predicted that this year 750 students are registered for COS132 by next year there will be close to a thousand. We expect a 25 percent growth rate over the next two years. [1] The system must scale and incorporate with the standard Computer Science website.

- Tactics or Strategies
  - Data Partitioning: Spread data into multiple databases.

  - Cache Engine (Dynamic Cache): instead of redo the same execution for same input parameters, we can remember the previous execution's result.
  - Resources Pool: DBSession and TCP connection are expensive to create, so reuse them across multiple requests.

- Patterns or Styles
  - Flyweight Design Pattern.

- Integration
  - The data source will contain a large volume of records pertaining to the students registered for the module making use of the Buzz system and therefore has to have the capability to handle a large data set and multiple concurrent users.

Performance

We will require a response time of no longer than 1 second. We expect about a maximum 100 searches of forum requests per second.With an average of about 5 requests per second. We however do not expect unindexed searches. To achieve both moderate scalability and performance the following will be incoporated into the system.

- Tactics or Strategies
  - Real time access: combining system with Cloud computing.
  - Data Partitioning: Spread data into multiple databases.
  - Cache Engine (Dynamic Cache): instead of redoing the same execution for same input parameters, we can remember the previous execution's result.
  - Resources Pool: DBSession and TCP connection are expensive to create, so reuse them across multiple requests.
  - Asynchronous Processing: continue with other processes, whilst waiting on the response of the other.

- Patterns or Styles
  - Flyweight Design Pattern.

- Integration
  - Document-based Integration allows for the document to be provided by the data source, therefore the performance of the external data source will not impact heavily on the performance of the Buzz system. Data required can be retrieved by multiple users which will not affect the systems performance by slowing down the retrieval process.

Maintanability

The system will be made up from various modules and using a Singleton design pattern to handle communication. Thus a centralised node that all communication will pass through between modules. This will allow us the easily edit/add/remove modules from the system.

- Tactics or Strategies
  - Design for maintainability from the outset.
  - Iterative development and regular reviews improve quality, e.g. Using the agile approach.
  - Code readable that is easy to understand.
  - Provide relevant documentation helps developers understand the software for further maintenance.
  - Make use of automated builds make the code easy to compile.
  - Make use of automated tests make it easy to validate changes.
  - Application Architecture Standards: Multilayer design compliance.
- Patterns or Styles
  - Dependency Injection and MVC Design Pattern.

Reliability and Availability

Reliability will definitely be a priority. We will use a server that guarantees us at least 99 percent uptime. This server will be provided by the client.

- Tactics or Strategies
  - Apply java OO and structured programming practices.
  - Use good architectural infrastructure.
  - Build management information into the application.
  - Use redundancy for reliability, that is have multiple copies or services of the same type running at the same time.
  - Use quality development tools.
  - Use consistent error handling, and provide meaningful error messages.
  - Elimination of single points of failure, by having multiple copies or services of the same type running at the same time, as specified before. So that when one service cannot be provided, then others or possibly the same services can be provided.
  - Provide reliable crossover, enabling uptime to remain as high as possible even during maintenance.
  - Detection of failures as they occur.

- Patterns or Styles
  - Dependency Injection.
- Integration
  - When the Buzz system requires certain records in an efficient timely manner, reliability is also a major concern. To allow proper accessibility, functionality and productivity of the system, reliable data has to be delivered at all times or else the whole system will not operate as desired.

Security

Clients must not have file access to the server and the root assess of the system must only be allowed to authorised personnel. Designing strong password policies and using **security methods**.

- Tactics or Strategies
  - Input Validation
  - Authentication
  - Authorization
  - Sensitive Data (Confidential information disclosure and data tampering)
  - Session Management
  - Cryptography
  - Parameter Manipulation
  - Exception Management
  - Auditing and Logging
  - Access control
  - Auditing
- Patterns or Styles
  - Authentication Enforcer.
- Integration
  - The data provided will come from a reliable data source that has sensitive information regarding students who are registered and the details of the system itself. During the integration process, the Buzz system cannot compromise the data source by feeding malicious data to intentionally or unintentionally extract valuable information. The Buzz system can only access records that the data source deems as essential for the use of the Buzz system.

Usability

The web pages must be usable and follow a responsive design approach. This means the pages will display correctly on each screen size. We will follow google material design standards to ensure responsiveness.

- Tactics or Strategies
  - Look-and-feel: includes making navigation easy, useful interface cues, good color choice, for easy reading and scanning.
  - Navigation: to tell the user where they are, and enable the user to go somewhere else.
  - Interface design: inform users of the task the interface can be used to complete and provide feedback to let users now what has been done.
  - Information architecture: organize or structure contact pleasant to read manner, and make use of short phrases as much as possible.
- Patterns or Styles
  - Dependency Injection.

Testablity and Integrability

Each function will be tested. But mathematical proofs for correctness is not necessary. Each part of the system must be tested. The system must be able to integrate into an existing computer science website. We must meet their standards when it comes to obtaining user information. We will conform to all communication standards to the system. The system must also provide its own integration library which will gain access throughout the singleton node. Lastly the following will enable the system to be both testable and integratation friendly.

- Tactics or Strategies
  - Isolate the Ugly Stuff: "ugly stuff" is any kind of code or infrastructure that is complicated or laborious or just plain inconvenient to get into a test harness, or that makes tests run very slowly.
  - Using Fakes to Establish Boundary Conditions: For instance instead of doing the data directly delegate to some other services.
  - Separate Deciding from Doing: an action and deciding to take an action treated as two separate responsibilities.
  - Small Tests before Big Tests: Small test often point you direct to point of failure or error, where else Big test have a lot of factors to consider, which makes debugging hard, so more small test and leading to easy debugging for Big test.
- Patterns or Styles
  - Dependency Injection and Mock Object.

1.2 Integration and access channel requirements

Access channel requirements

The system will be accessed through:

- Mobile devices, desktops and tablets by using web browsers (this includes all leading web

browsers such as Google chrome, Mozilla Firefox, Opera, Safari and Internet explorer).

- RESTful systems
- HTTP requests
- HTTPS

The system must also be able to access a database on a server to retrieve student information. This will be achieved by using LDAP. The buzz space should not only be accessible via desktop/laptop based web browsers, but also mobile devices.

Rendering of a mobile version of the website should thus be required in order to maximise ease of use for mobile users. Since a mobile app will not be developed, mobile sites should be of high importance when designing the website. The mobile websites does not need to be an entire different site, such as a .mobi website, where the main website would be for example .co.za. The need for a responsive website therefore arises.

Channels

- REST: Representational State Transfer
  - The REST architecture design is a good option to use as it is a simpler than SOAP and is a dynamic design.
  - A RESTul system can integrate well with HTTP as RESTful systems are optimized for the web.
  - Restful systems needs to follow a client-server model,so this means that there needs to be communication between the client and server which is vital in the buzz system.
  - There is support for a lot of components to interact with each other and to be interchangeable.

Protocols

- HTTP(Hypertext Transfer Protocol) is the main protocol for all websites in the modern internet usage.It allows linking of nodes which allows the users to easily navigate through web pages.

- HTTPS is a more secure version of HTTP. HTTPS is a combination of HTTP and TSL and SSH.This protocol will ensure that data is safely transported.
- TCP/IP(Transfer Communication Protocol/Internet Protocal) Used to as communication over the internet.TCP is reliable and is able to check for errors in the transfer of the page over the IP.
- SMTP(Simple Mail Transfer Protocol) is used to send emails. This will be easier then mailing manually and is prominently used in the web space.
- IPv6(Internet Protocol version 6) This will redirected and to allow them to be redirect users or routed to the correct space on the internet. This provides access to buzz through the use of http and the TCP/IP protocols.
- IPsec(Internet Protocol Security) will allow for a secure IP and to ensure no harmful data is ever transmitted to the servers of buzz.

API Specifications

- Web Service Definition Language(WSDL) - WSDL will be used to describe the functionality and the operations provided by the web-based service (Buzz system).
- Common Object Request Broker Architecture(CORBA) - CORBA will mediate the communication between the diverse systems that will be integrated to the Buzz system to provide added functionality and data for the operation of the system.
- Interactive Data Language(IDL) - This will be used for data analysis purposes for the data passed from the data source to the Buzz system and any other form of data required by the system.

1.3 Architecture constraints

The following architecture constraints have been selected by our client as being suitable for the system.

1. **Development environment:** Linux
2. **Development platform:** Java-Enterprise Edition (Java-EE)
3. **Architectural frameworks:** Java Server Faces (JSF)
4. **Version control management:** Git
5. **Development technologies:**
    - HTML
    - Java Persistence API (JPA)
    - Java Persistence query language (JPQL)
    - Asynchronous JavaScript and XML (AJAX)

2 Architectural patterns or styles

2.1 Authentication Enforcer

The Authentication Enforcer pattern handles the authentication logic across all of the actions within the Web tier. It assumes responsibility for authentication and verification of user identity and delegates direct interaction with the security provider to a helper class. This applies not only to password-based authentication, but also to client certificate-based authentication and other authentication schemes that provide a user's identity, such as Kerberos. This pattern will help us improve security of the system while keeping the scalability and performance well maintained since the Authentication Enforcer pattern provides a consistent and structured way to handle authentication and verification of requests across actions within Web-tier components and also supports MVC architecture without duplicating the code.

2.2 Dependency Injection

Dependency injection implements inversion of control for software libraries.
The design pattern allows a client to remove all knowledge of a concrete implementation that it needs to use. This helps isolate the client from the impact of design changes and defects. It promotes re-usability, testability and maintainability. The benefits of using Dependency Injection in our system, is that it will ensure loose coupling of code between the different modules that need to be implemented. Decoupling of code will ensure that the code in the system are cleaner, easier to modify when required and easier to adapt and implement for reuse.

2.3 Flyweight Design Pattern

A flyweight is an object that minimizes memory use by sharing as much data as possible with other similar objects; it is a way to use objects in large numbers when a simple repeated representation would use an unacceptable amount of memory. This is how we will improve on performance and the scalability of the system as resources are used efficiently and effectively.

2.4 Mock Object

Mock objects are simulated objects that mimic the behavior of real objects in controlled ways. We have to create a mock object to test the behavior of some other object and see how it reacts to certain conditions.

2.5 Model-View-Controller (MVC)

Adhering to the MVC design pattern provides us with numerous benefits:

1. **Separation of design concerns:** Because of the decoupling of presentation, control, and data persistence and behavior, the application becomes more flexible; modifications to one component have minimal impact on other components.
2. **More easily maintainable and extensible:** Good structure can reduce code complexity. As such, code duplication is minimized.
3. **Promotes division of labour:**Developers with different skill sets are able to focus on their core skills and collaborate through clearly defined interfaces.

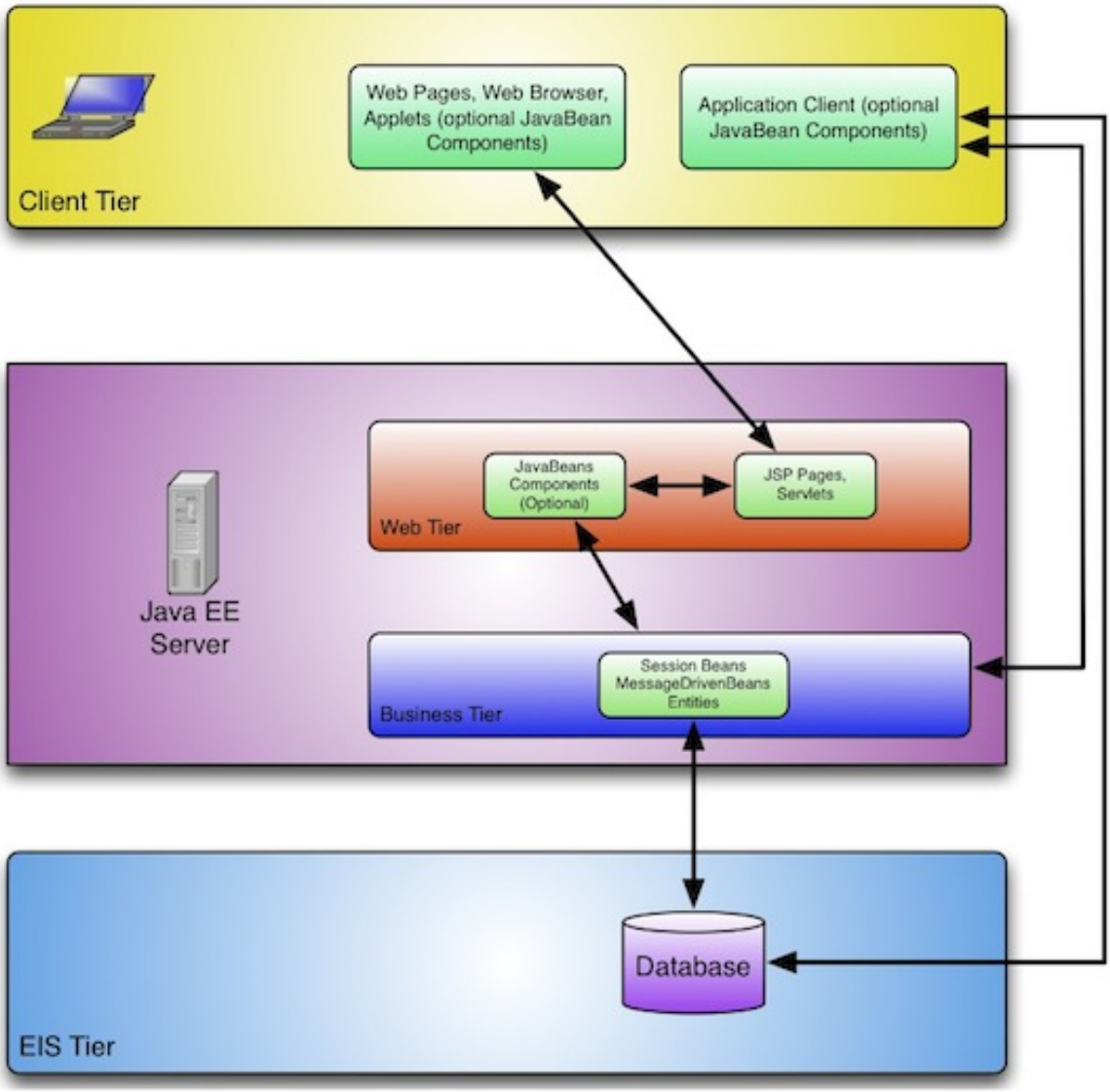

Figure 1: Java-EE system architecture

View (Client Tier)
This tier runs on the user system and encapsulates the various components that a user system may use to access the Java EE server-side tiers. These components include dynamic web pages, Java applications and Java applets.

Controller (Java-EE server)

The middle tier's business functions handle client requests and process application data, storing it in a permanent data store in the data tier.

- **Web Tier**
  The web tier consists of components that handle the interaction between clients and the business tier.
- **Business Tier**
  The business tier consists of components that provide the business logic for an application. Business logic is code that provides functionality to a particular business domain.

Model (Enterprise Information Systems (EIS) Tier)
The EIS tier consists of database servers, enterprise resource planning systems, and other legacy data sources. These resources typically are located on a separate machine than the Java EE server, and are accessed by components on the business tier. This will be the university's server for the mini project.

3 Architectural tactics or strategies

Performance is one of the most important quality requirements that must be fully incorporated into the system. This includes real time access; students will be reluctant to use a system with frequent delays. Asynchronous processing will increase the systems performance, so will data partitioning, resources pooling and others specified above. Performance is closely related to scalability, so these tactics will also enhance scalability, while ensuring good performance from a user point of view. Scalability will ensure that performance on the client side is up to standard.

Maintenance becomes easy if you design for maintainability from the outset, and be agile. Coding readable code that is easy to understand reduces documentation, of which is ideal, but unfortunately documentation is still required.

Systems that are available, reliable and secure encourage frequent usage. Students study or work all day, and mostly at night, so access will be available at all times. As much as security is a top priority, access to the system should be short and one-time sign-ins should be made possible, but in the background intense security measures, per single login attempt, can take place, e.g. input validation, authentication, and encryption.

It is also ideal to have small tests, to test each individual component or functionality, so as to make it easy to trace errors. If no errors were found, the big tests can be done. Testing is for developers but usability is strictly about the user point of view, so the systems must be easy to use, easy to remember how to use, as these mentioned practises

will encourage users to use the system again.

4 Use of reference architectures and frameworks

Reference Architecture

As discussed in a previous section, the type of reference architecture that will be used for our system, is a Layered reference architecture. Java-EE is the Layered reference architecture best suited for the Buzz system. Java-EE is the best choice because it encompasses most of the quality requirements that we want to exploit which are scalability, security, reliability and integrability.

- Apache CXF is a web service framework which could be used as it offers good performance with little computational overhead.It can be used with Java and Maven.
- Apache Axis2 is essential for any web service.it is compatible with RESTful systems.It does not use a lot of memory and allows for deploying while the system is up and running.Allows the programmers to change and insert anything they wish.
- Spring Framework is a application framework that can be used in conjunction with Java EE.It provides all the features we need such as working with a database,messaging and authentication.Objects can be managed through the use of the Inversion control container.
- Spark is a web framework which will allow for easy routing of buzz.
- Apache Tapestry is a view framework made up of components.it uses post and get when submitting the form and separates the html code from the Java code.

5 Technologies

We have chose the following technologies carefully as they incorporate into one another to form a sound basis for buzz space to operate on.

- **Java-EE** As mentioned in previous sections, Java EE will be best suited for the layered reference architecture that we want to use in our system. Java EE ensures efficient separation of layers when required in developing a system. Different layers is however what we want to achieve in our system.
- **Apache Maven** A great build tool that will be used for software project management.
- **HTTPS** We will use this technology to help reach our security quality requirement. It will add encryption to the website as a security method.

- **JPA** We will use this API to help with the database integration. Thus we will gain scalability as required.
- **mySQL** As mySQL fits nicely into the java persistence API we will us this to amplify the benefits of using the JPA.
- **Bootstrap** Used in the HTML and interface part of the system for a competitive and easy presentation across all browsers and mobile browsers .
- **SMTP** The standardized mail protocol used to send mail and this is vital, as one of the features of buzz is to send emails to the users.
- **IPSec** This is a security technology which secures IP addresses which will be useful to help secure buzz.
- **JavaScript/JQuery** Used in the client side of validation of registration and login details.
- **WSDL** We will use this technology to determine the function requirements and operations that will be required.
- **CORBA** This API will be used in the integration process independent from other technology's implementation, location, networking technologies and protocols.
- **IDL** This is a programming language that will be used for data analysis.
- **JSF** This is a Java specification for building component-based user interfaces for web applications.
- **JDBC** Defines how a client may access a database. It provides methods for querying and updating data in a database. JDBC is oriented towards relational databases.
- **EJB** An architecture for setting up program components, written in the Java programming language, that run in the server parts of a computer network.
- **JAX-RS** An API that provides support in creating web services according to the REST architectural pattern. JAX-RS uses annotations to simplify the development and deployment of web service clients and endpoints.
- **Glassfish Server** Reference implementation of Java EE and as such supports EJB, JPA, JavaServer Faces, JMS, RMI, JavaServer Pages, servlets, etc. This allows developers to create enterprise applications that are portable and scalable and that integrate with legacy technologies. Optional components can also be installed for additional services.
- **JUnit Testing** A unit testing framework for the Java programming language. JUnit has been important in the development of test-driven development.

Bibliography

[1] Prof Andries Engelbrecht, *Class Represetative Meeting*. University Of Pretoria, Wednesday, 4 March 2015, IT Building, room 4-66, 2015.