Post-Doctoral Application Management System

Architecture specification Document

September 13, 2014
Version 1.0
Iteration 4

Prepared for Ms. Cathy Sandis (UP Research Office)
by SoftServe Group

Group members

Kgothatso Phatedi Alfred Ngako (12236731)

Tokologo "Carlo" Machaba (12078027)

Mathys Ellis (12019837)

Change log			
Date	Version	Description	Person
18/05/2014	v 0.0	Document created.	Alfred Ngako
19/05/2014	v 0.1	Added more detail to information in subsections.	Alfred Ngako
20/05/2014	v 0.2	Added references and citations.	Alfred Ngako
22/05/2014	v 0.3	Transferred architectural scope to architectural requirements specification document. Began editing.	Mathys Ellis
23/05/2014	v 0.4	Added to and edited architectural patterns and style.	Mathys Ellis
23/05/2014	v 1.0	Finalised and edited document for first iteration.	Mathys Ellis
18/08/2014	v 2.1	Added to technologies as well as updated the section.	Mathys Ellis

Contents

List of Figures

1 Project Repository

https://github.com/mox1990/Project-Postdoc.git

2 Document description:

2.1 Document purpose

This document follows on the Architecture requirements specification document. The Architecture specification document provides the architectural descriptions of various architectural factors that the project will need to follow when developing the system. Architecture in this document's context refer to the technological basis and software development styles of the project. Thus this document serves as a contract between SoftServe and the client, Mrs Cathy Sandis of the DRIS of the University of Pretoria in terms of what technologies the project should incorporate as-well the software development infrastructure that the system will be based on.

2.2 Documentation methodology

The documentation and software development methodology used by the project adhere to the guidelines set out by the scum agile methodology. Thus this document has undergone and will undergo various iterations that may extend or reduce the contents of the document.
The document was compiled using a software architecture specification document template provided by Dr Fritz Solms as an alternative to the Kruchten 4 + 1 approach to documenting software.
This document was created using the requirement elicitation techniques and requirement definitions as specified by Klaus Pohl's book Requirements Engineering: Fundamentals, Principles, and Techniques [Dr.Phol, K., 2010]. The requirements, vision and scope were elicited from the following sources:

- Numerous interviews with the client.
- On-line research into UP Post doctoral applications.
- Correspondence with the UP IT department.
- Collecting and analysing various documents such as:
  - The initial project request document
  - Application forms
  - Renewal forms
  - CV templates
  - Approval and recommendation forms

2.3 Document conventions:

- Documentation formulation tool: LaTeX

2.4 References:

- Kang, A. August 9, 2002, *Enterprise application integration using J2EE*, Available from: http://www.javaworld.com/article/2074488/enterprise-java/enterprise-application-integration-using-j2ee.html , [Accessed on: 17 May 2014]
- Ali Babar, M., *Architectural Patterns and Frameworks, Week 3, Lecture 3*,Available from: https://blog.itu.dk/MSAR-E2013/files/2013/09/wk3_lect3_patternsframeworktactics.pdf, [Accessed on: 21 May 2014]
- Dr.Phol, K., 2010, *Requirements Engineering: Fundamentals, Principles, and Techniques*, Springer, Heidelberg.
- Kayal, D., 2008, *Pro Java EE Spring Patterns: Best Practices and Design Strategies Implementing Java EE patterns with Spring Framework*, Apress, New York.
- Jendrock E, Cervera-Navarro R, Evans I, Haase K, Markito W, *The Java EE 7 Tutorial*, Available from: http://docs.oracle.com/javaee/7/tutorial/doc/home.htm , [Accessed on: 20 May 2014]
- Oracle. April 2014, *The Java EE 7 Tutorial*, Available from: http://docs.oracle.com/javaee/7/tuto , [Accessed on: 20 May 2014]

3 Architecture Requirements

This section can be found in the architecture requirements specification document in which it discusses the following:

- Architectural scope
- Access channel requirements
- Quality Requirements
- Integration requirements
- Architectural Constraints

4 Architectural Patterns and Styles

The system will employ the Model-View-Controller, MVC, architectural pattern combined with a multi-tier/layered architectural pattern to form what is known as the Java Enterprise Edition system architecture. This will allow the client(s) to be decoupled from the server. Further this allows the view, the controller and the model each to have its own set of layers. Both of these patterns provide various benefits and are widely used. Some benefits include modularity, encapsulation, re-usability of components, decoupling and system maintainability [Ali Babar, M.] [Oracle, 2014][Kayal, D., 2008]

The Java EE system architecture is designed to support highly scalable, distributed, transactional, and portable applications that use the speed, security, and reliability of a Java EE server to provide powerful enterprise applications. [Oracle, 2014] For this reason SoftServe believes this system architecture is well suited for the development of the Post-Doctoral application management system as it will satisfy the requirements in terms of quality and functional as stipulated by the client. The following diagram provides the system architectural that will be employed by the system:

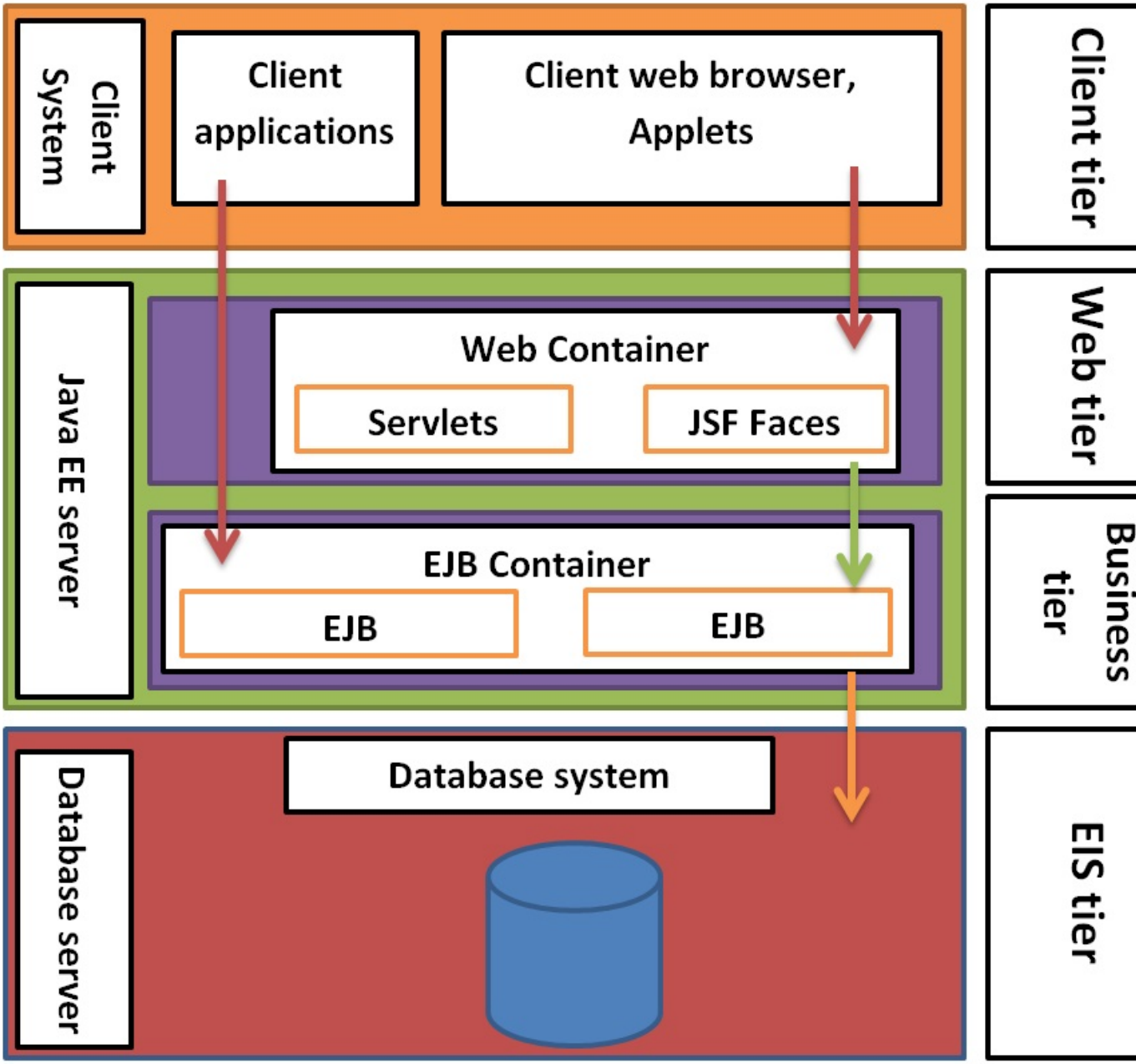

Figure 1: Java EE system architecture

4.1 View

The View component of the system architecture is used to represent and encapsulate the front-end of the system. It contains the client tier of the Java EE system architecture. Thus this allows the system to decouple the front-end and the back-end components. Allowing various clients to access the back-end without having to duplicate certain back-end tiers for each client. [Kayal, D., 2008]

4.1.1 Client Tier

This tier runs on the client system and encapsulates the various components that a client system may use to access the Java EE server-side tiers. These components include dynamic web pages, Java applications and Java applets. In order to make the Post-doctoral application management system accessible to any stakeholder over the internet and provide a uniform user experience the system will only make use of the dynamic web pages component provided by the Java EE client tier. [Oracle, 2014]

4.2 Controller

The controller component of the system architecture is used to provide the business logic and manage and manipulate the view's request in order to provide communication between the Model component and the View. This component therefore hosts the web tier and the business tier of the Java EE application. This component is located on the Java EE server which is a multi-threaded application server. [Kayal, D., 2008] [Oracle, 2014]

4.2.1 Web Tier

This Tier runs on the Java EE server and hosts the Web container. It provides the management and web page generation support for the web pages that the system has to provide to the view through the user of servlets, Java ServerPages and Java ServerFaces Facelets. Facilitates the communication between the business tier and client tier for web browser clients and applets. Client applications do not have to make use of the Web Tier and can directly skip to the business tier. But as stipulated above the other types of client components will not be considered for the project. [Oracle, 2014]

4.2.2 Business Tier

This Tier also runs on the Java EE server and hosts the Enterprise Java Bean container. It provides the business logic section for the Java EE application in the form of Enterprise Java Beans which are simply classes that represent various persistence entities of the data base, system messages, sessions, etc. This tier communicates with EIS tier in order to get access the database and various other lower level infrastructures that the Java EE application requires. [Oracle, 2014]

4.3 Model

The Model component of the system architecture contains the various persistence storage infrastructure and lower level system management features such as transaction processing. It hosts the Enterprise Information System tier of the Java EE application. [Kayal, D., 2008]

4.3.1 Enterprise Information Tier

The EIS tier provides mainly the support for database systems that is used by the Java-EE application. This tier can run on the Java-EE server as a virtual server or on a physically different database server. Due to the project budget and technical constraints the former will used by the system. [Oracle, 2014]

5 Architectural Tactics and Strategies

This section describes the architectural techniques which will be used in order for the system to satisfy the quality requirements.

5.1 Concurrency

The Java Platform has always offered support for concurrent programming, which is the basis for implementing many of the services offered by Java EE containers. This is realized through the two main concepts of having a multiple threads execute under a single process, in the case of Java EE mulitple beans execute under the JVM. The number of threads that can execute under the JVM can go well beyond thousands depending on factors such as the machine the JVM is running on and how it has been configured.

Even though the concurrent threads will mean better performance and a scalable implementation for the system they may lead to issues that effect the reliability and integrity of the system such as:

- Deadlocks,
- Thread Starvation,
- Concurrent accessing of shared resources, and
- Situations where the program generates incorrect data.

To deal with these issues Java EE provides concurrent utilities that access concurrent resources via JNDI lookup or resource injection. The components in the utilities ensure that the issues mentioned above are nullified. The primary components of interest in the concurrent utilities for our system are:

- managed executor service,
- managed scheduled executor service,
- managed thread factory, and
- context service.

5.1.1 Concurrency Utilities for Java EE

5.1.2 Managed Executor Service

A managed executor service is used by applications to execute submitted tasks asynchronously. Tasks are executed on threads that are started and managed by the container. The context of the container is propagated to the thread executing the task.

For example, by using an ManagedExecutorService.submit() call, a task, such as the GenerateReportTask, could be submitted to execute at a later time and then, by using the Future object callback, retrieve the result when it becomes available ().

5.1.3 Managed Scheduled Executor Service

A managed scheduled executor service is used by applications to execute submitted tasks asynchronously at specific times. Tasks are executed on threads that are started and managed by the container. The context of the container is propagated to the thread executing the task. The API provides the scheduling functionality that allows users to set a specific date/time for the Task execution programmatically in the application.

5.1.4 Managed Thread Factory

A managed thread factory is used by applications to create managed threads. The threads are started and managed by the container. The context of the container is propagated to the thread executing the task. This object can also be used to provide custom factories for specific use cases (with custom Threads) and, for example, set specific/proprietary properties to these objects (Cervera-Navarro, Evans, Jendrock, Haase and Markito 2014).

5.2 Design Patterns

5.2.1 Builder Pattern

The Builder Design Pattern allows for the construction of complex structures in small steps. An example of how it is used is in the process of making an application, the different elements of an application are separated. The process of creating objects speeds up as well. This construction of objects in small steps decreasing coupling and makes the code more testable. The design helps improve the flexibility as well as the maintainability of the code.

5.2.2 Factory Design Pattern

The Factory Design Pattern provides centralised to create our Database entities and entries in an orderly fashion. The pattern allows the developers to define an interface for creating an object, but let the classes that implement the interface decide which class to instantiate. By doing this the code is now flexible and creation of new objects is much simpler. It also helps redundancy

5.2.3 State Design Pattern

The State Design Pattern is best used in situations where the actions take place in a pre-defined order as with this project. The State pattern changes the state of the objects depending on how far in the pipeline. It provides a simple and clean way to change the state of an object during run time. The design pattern improves the usability of the system as users will know exactly when some change has taken place.

5.2.4 Dependency Injection

This is used to implement, inversion control. The client is only allowed to use a service rather than creating their own services. 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

5.2.5 Database Access Objects

The Database Access Object is used by all the application calls to provide specific data operations without exposing details of the database to external objects. It forms part of the Core J2EE Patterns. This design pattern acts an intermediary between the application and database, by moving back and forth between objects and database records. It also contains the effects of any changes to the persistence mechanism to a confined area and not the whole application. This pattern improves the audit-ability as well maintaining the integrity of the data contained in the system.

The use of DAO will help reduce the existance of object-relational impedance mismatch present in the system. It will also contribute to the flexibilty of the system. In the case where the systems underlying persistance mechanism has to change only the DAO will have to be updated and all the places in the system where the DAO was used will then remain constant.

6 Use of Reference Architecture and Framework

The core design philosophy of the Java-EE platform is to provide a Java-EE application developer with a set of test and well maintained and reusable APIs as well as frameworks that allow them to focus rather on implementing the actual business logic and UI than focusing on the underlying system technical and management services such as authentication, session management, etc. Thus the Java EE platform provides a runtime environment for developing and running large-scale, multi-tiered, scalable, reliable, and secure network applications. As seen above it also provides a system architecture and various frameworks, discussed below, for implementing services for multi-tier applications that deliver the scalability, accessibility,

and manageability needed by a system. Taking all the above into consideration this makes it ideal for the development of the project.

6.1 JavaServer Faces (JSF)

JSF is a web application GUI framework that is based on the JSP, EL and servlet technology that Java-EE provides. It allows the generation of various mark-up languages, such as HTML 4.0.1 and HTML 5, directly from objects and ORM model objects used by the Java-EE application. Thus it is ideal for system as the system needs to provide support for both HTML 5 and 4.0.1 web content.

It will help achieve the usability quality requirement as it will implement all aspects of the user interface.

6.2 Java Persistence API

Java EE is based on Java which is an object-oriented language. Whereas most modern day database management systems, DBMSs, provide relation databases. Thus to bridge this gap the Java Persistence API is used. It provides a Object Relational Mapping solution which allows the relation database to be viewed as a object-oriented database. Thus this critical for the system SoftServe wishes to develop as the system will make use of MySQL which is a relation database management system. The Java persistence API contains the following components:

- Persistence API
- The query language
- ORM

6.3 Java API for RESTful Web Services (JAX-RS)

The JAX-RS API provides a way for the Java-EE application to provide web services or data transfer via the HTTP or HTTPS protocol using the Representational State Transfer, REST, architectural style. This accomplished by the user of various JAX-RS runtime annotations. This will allow the system to provide a set of lightweight web services to various clients across the internet. This will allow the system to easily be accesses by mobile and computer platforms alike and also be accessible over most companies firewalls as it will make use of the HTTPS protocol. Further this will allow for future expansion if client wishes it. To insure security a the POST command will be preferred above GET.[Cervera-Navarro, Evans, Jendrock, Haase and Markito, 2014]

6.4 JUnit

JUnit is a simple unit testing framework used to write repeatable tests. Test methods must be annotated by the @Test annotation. It is also possible to define a method to execute before (or after) each (or all) of the test methods with the @Before (or @After) and @BeforeClass (or @AfterClass) annotations.

It will be used to achieve the testbility of the system.

7 Access and Integration channels

This section discusses the requirements for the access channels through which the system can be accessed by humans and other systems. Also it discusses the integration channels which need to be used by the system.

7.1 Access Channels

To provide a system that is as accessible as possible the system provide its services through this essentially makes the system OS independent and accessible over firewalls since it will make use of the HTTPS protocol that is usually not blocked by firewalls. This also improves the usability the system as most computer user or mobile users are aware of how to use web browsers. The system will support the following versions of modern web browsers for both their computer and mobile counterparts:

1. Mozilla Firefox 20+
2. Google Chrome 30+
3. Microsoft Internet Explorer 9+
4. Apple Safari
5. Opera

7.2 Integration Channels

As mentioned in the architecture requirements specification document the IT department of the University of Pretoria will only be willing to provide the SoftServe group with the knowledge of the Peoplesoft system in order to integrate it. Thus the SoftServe group will attempt to accommodate the PeopleSoft system as much as possible by conforming to same user name styling, export of data to formats that are used by the Peoplesoft system to import data. Also the since Peoplesoft is an Oracle enterprise system it would have been developed using Java-EE and the Java programming language. This is therefore a strong motivation

for Java-EE to be used by the SoftServe group as a development platform for the system. So to allow integration at a more technical level. Though if the integration was to be done this would be done via the EIS teir using the Java EE Connector Architecture API and the JAX-WS API.

8 Technologies

This section discusses and elaborates on the technologies that the system will use and should also be seen as an extension of the architectural constraints, specified in the Architecture requirements specification document, in terms of elaboration.

8.1 Integrated Development Environment

The system should be buildable independent of an IDE but it will be developed on Netbeans 8.0 to allow for uniformity amongst the development team, with regards to coding style, and provide easy integration with the tools that will be used such as Javadoc, to generate API documentation in HTML format.

8.2 Development technologies

8.2.1 Java Servlet Technology

Description
A Java servlet is used to extend the Java-EE application server to support HTTP or HTTPS requests and responses. It allows the server to provide RESTful based web services to connecting clients. Thus it acts as a middle man between any HTTP or HTTPS client and the business tier. This will not be used directly but will be used the JSF framework.

Reasons for use
The JSF framework uses servlets to render JSF pages to HTML and provide them to clients connecting via HTTP or HTTPS. Secondly this will allow the solution to provide lightweight RESTful based web services that will help improve accessibility and availability of the solution. Further it will allow the solution to satisfy the access channel requirements.

8.2.2 JavaServer Pages (JSP)

Description
JSP is a technology that is used by the Java-EE platform to provide a native language approach to creating web pages. It uses HTML or XML to specify static content on a web page and Expression Language (EL) to provide dynamic content. This will not be used directly but will be used by the JSF framework.

Reasons for use
JSF pages is the a specialised version of JSP pages thus it is used in the JSF framework. Secondly it will help with the maintainability of the code as pages based on JSP technology are easily understandable and readable due to the simplicity of the HTML and XML mark-up languages.

8.2.3 Expression Language (EL)

Description
It is a language used by JSP and JSF pages to write servlet based code snippets which allow the usage of the data available to the servlet to do calculations, call functions, get or set data. The language closely resembles the Java Language syntax.

Reasons for use
It is used by the JSP technology and JSF framework for dynamic content specification and communication with the backing servlet.

8.2.4 JavaMail API

Description
The JavaMail API provides a robust and well tested email communication infrastructure for any Java based applications.

Reasons for use
This technology will be employed by the system in order to provide the functional email notification infrastructure requirement needed by the solution.

8.2.5 JavaBeans Activation Framework (JAF)

Description
The JavaBeans Activation Framework allows the Java-EE application to determine the data type of some section of data and thus allow the application to provide access to it by encapsulating it and determining the operations that the application may perform on it.

Reasons for use
This technology is used by the JavaMail API thus it will be used by the system.

8.2.6 Java Database Connectivity API (JDBC)

Description
The Java Database Connectivity API provides the a Java-EE application with the means to access data from various datasources including databases, spreadsheet, etc via the Java

programming language.

Reasons for use
The system will use this to communicate with the databases located in the EIS tier of the system in order to retrieve and store data in the databases.

8.2.7 Java Transaction API (JTA)

Description
The Java Transaction API provides the Java-EE Application with the means to handle and demarcate data transactions to the database. The API allows the the manual or automatic demarcation of database transactions and ensures that the database and ORM entities are synchronised after commits.

Reasons for use
This will be used in the system to improve the centralisation and accuracy of the data accessed by users. Further it will be used to allow access to multiple databases located in the EIS tier and the controlling of such resources.

8.2.8 Enterprise JavaBeans (EJB)

Description
The Enterprise JavaBeans is a component used by all Java-EE applications to encapsulate the various business logic of the application into reusable modules. Thus it contains the attributes and methods associated with the business logic module and hence can be treated as an stand alone unit that can be reused. The two primary EJBs are Session beans that represent a clients session and the data associated with it and a Message-driven bean that can a allow the component to receive messages asynchronously via event listeners. The system will make use of Stateless session EJBs in order to capture the back end services of the solution.

Reasons for use
The use of EJBs will ensure the modularity of the system and the re-usability of its components. Also within the Java-EE framework it is considered the core of the business tier and thus will be required.

8.2.9 Java Naming and Directory Interface (JNDI)

Description
The Java Naming and Directory Interface provides Java-EE applications with the ability to search for data across LDAP, DNS, etc, directory services. Above this it also allows the application to search for objects that exist within the application and provides access to

them so that they can be used by the application. It is also used by Java-EE applications to locate object instances of EJBs and managed beans for usage by the application.

Reasons for use
This technology is a core service required by Java-EE applications to function and thus is needed by the solution.

8.2.10 Java Architecture for XML Binding (JAXB)

Description
JAXB is used to parse any XML data into a set of XML usable Java object instances that represent the content of the XML data. Likewise it is used to convert such objects to XML formatted data. This is accomplished by using an XML schema to define the structure or annotated classes.

Reasons for use
This will be used by the system to handle any XML data extraction or creation from or for storage.

8.2.11 Jasper Reports

Description
The Jasper reporting library provides a flexible, well-tested and maintained reporting framework to create reports based on data that is located in various data sources.

Reasons for use
This will be used to provide a reporting infrastructure as need by the system on a functional requirement level.

8.2.12 JSF Managed Beans

Description
This is a standard POJO (Plain Old Java Object), which is used to provide encapsulated services to front-end JSF pages. JSF access them via Expression Lanaguage. These beans can be used in conjunction with EJBs and CDI to communicate with the database or call back-end services of hosted by the server.

Reasons for use
This technology is part of the JSF framework and is therefore required. It further also allows improved modularity and re-usability of components.

8.2.13 Primefaces

Description
This is component library which is used to expand and provide an improved range of easy to use components for JSF pages. It incorporates Javascript, jQuery and AJAX in order to provide the various components.

Reasons for use
It is an easy to use, versatile library that allows the solution provide a sophisticated and clean user interface. Further is highly compatible with mobile platform HTTP web browsers thus allowing the solution to provide better accessibility without extensive modification.

8.2.14 Mockito

Description
This is a mock framework that allows the mocking of dependencies during unit testing phases.

Reasons for use
It is easy to use and provides a great number of effective features that will help enhance unit testing during the development of the solution. Thus it helps improve the testability of the solution.

8.2.15 MySQL DBMS

Description
This is a well-proven open-source relational database management system that provides extensive set of features for maintaining the data and database.

Reasons for use
It a is well-proven DBMS. Further since the data captured by the solution will fit well in a relation approach the DBMS will be efficient enough to meet the performance and scalability requirements of the solution.

Build Tools

- **Apache Maven** - This build tool was chosen due to the flexibility of in terms of configurations, its ability to automate tests and the fact that Netbeans IDE provides support for it.

Operating System

The system will be deployed on a single OS but will have clients that will use a variety of OSs. This is one of the main reasons the why the application's UI must be web browser

based so to allow OS independent support. Thus the only prerequisite of the client's OS is that it needs to support a HTML 4 or 5 web browser and be internet accessible.

- **Server OS:**
    1. Linux: Kubuntu 13.10
- **Client OSs**
    1. Windows: All
    2. Linux: All
    3. iOS: All
    4. Android: All

9 Glossary:

- **API** - Application Programming Interface
- **Application** -Both renewal applications or new fellowship applications are seen as applications by this project.
- **CV** - Curriculum Vita
- **EAI** - Enterprise Application Integration
- **HTML** - Hyper Text Mark-up Language
- **Java EE** - Java Enterprise Edition
- **JDBC** - Java Database Connection
- **MVC** - Model View Controller
- **UI** - User Interface
- **UP** - University of Pretoria