COS 301 Final year project

Post-Doctoral management System

Software architecture document

October 17, 2014
Version 2.0 (Final)
Iteration 6

Prepared for
Client: Ms. Cathy Sandis (UP DRIS)
Supervisor: Prof. Stefan Gruner (SSFM Group)
by
SoftServe Group

Group members

Kgothatso Phatedi Alfred Ngako (12236731)

Tokologo "Carlo" Machaba (12078027)

Mathys Ellis (12019837)

Change log			
Date	Version	Description	Person
03/10/2014	v 0.0	Combined architectural specification and architectural requirements specification	Mathys Ellis
03/10/2014	v 1.0	Complete restructuring and most of the editing and additions	Mathys Ellis
17/10/2014	v 2.0	Finalised and revised document	Mathys Ellis

Contents

List of Figures

1 Project Repository

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

2 Document description:

2.1 Document purpose:

The Software architecture document provides the architectural requirements that the project will need to follow as well as a set of quality requirements that must be met by the system. The quality requirements have been quantified in order to provide the SoftServe group and the client with a means to test the system for non-functional compliance. This allows the SoftServe group to set up tests in the non-functional testing document that can be used at the end of the development cycle to verify that the system complies with the architectural and quality requirements specified by this document. Further the document provides the architecture specification which covers 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 and organisational styles and strategies 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 as the software development infrastructure that the system will be based on and also the quality the system should be of once the project is complete.

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.
This document was created using various insights gained throughout the COS 301 course and the requirement elicitation techniques and definitions as specified by Klaus Pohl's book Requirements Engineering: Fundamentals, Principles, and Techniques [Dr.Phol, K., 2010]. The quality requirements and aspects of the architectural requirements, 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

3.1 Architectural Scope

This section discusses the architectural scope that the software architecture of the system must cover in order to produce a compliant system:

- A persistence infrastructure using a DBMS to facilitate the storage of the various domain objects (e.g CVs, DRIS information, and Applications). So to allow the system to store and centralise the various data that the system will be handling as outlined in the functional requirements and application document.
- A multi-user session infrastructure to assist in realizing the security requirements of authenticating users and their actions.
- A work-flow infrastructure for processing applications that can be modified and extended.
- A RESTful web service infrastructure, to provide a means for the system to be integrated in other systems and also to allow lightweight data communication between the users and the server and support for mobile platforms.
- A report generation infrastructure in order to support the data retrieval requirements of the client.
- An email infrastructure for the system to use in order to provide notification support.
- A action logging infrastructure to track user activity on the system, in order to provide audit-ability.
- A gateway infrastructure to provide access control to services provided by the system.
- A pre post condition infrastructure to dynamically change per and post conditions for certain operations in the system.

3.2 Access channel requirements

The only two access channel requirements specified by the client are as follows:

- **All stakeholders:** Users of the system will access it through the HTTP and HTTPS protocols either via a web browser client or some software, which makes use of the Restful API the system provides. Both will be locally installed on a user's desktop or mobile platform. Web browsers are expected to be HTML 5 compliment. This access channel will provide excellent availability and usability of the system due to the RESTFul nature of most internet enabled systems and the ease of using web browsers. The access channel will allow different stakeholders access to different sections of the system based on the roles assigned to their accounts.
- **System Administrator:** The system administrator will require the same as the above but will in addition need to be able to conduct any external administration duties on the system which can not be done via the the above access channel. Such as updates, server maintenance, etc. Thus such operations will be done via the server by some terminal emulation protocol, such as Telnet or SSH, or by physical access.

3.3 Quality requirements

This section discuses the quality requirements as specified by the client and are also in the order of precedence. Where the first takes the highest precedence.

3.3.1 Usability requirements

Requirement
This is first and foremost quality requirement stipulated by the client. The primary language of the system will be South African English. Any other language support is not considered part of the requirement but the system will be designed to allow for such development in the future via internationalisation.

The system's UI will only consider 2 types of user categories with regard to usability:

- **Trained user:**

  This type of user will need to have training in understanding how the system functions and how to use it. Their computer skills will be assumed to be in the range of basic to intermediate. Thus the user interface can allow for certain complexities but these complexities must be kept at a minimal. These users will be regarded as

system administrators or specialised users of the system. The stakeholders who fall under this category is the DRIS staff members overseeing the application process and any technical or maintenance support users.

- **Normal user:**

  This type of user will have no training with regard to the system. Their computer skills will be assumed to be none or minimal. Therefore the UI that they will have access to will be simplistic and will be as user friendly as possible. The stakeholders that fall under this category will be Prospective fellows, Research fellows, Referees, Grant holders, HODs, Dean's office members and Post-doctoral committee members.

The above is quantified as follows: The SoftServe group will need to provide the system with a uniform user experience that is as simplistic and as visually oriented as possible without negating the below requirements severely. In addition to this any form of help text or instruction should be written in as natural, uncomplicated and unambiguous manner as possible. A further quantification of this requirement can be seen as finding the least number of clicks a user has to perform in order to perform any operation. Thus to meet this requirement local tasks should take a maximum of five clicks and distant tasks should take up to a maximum of 8 clicks. Where local refers to tasks that are related to the current section being accessed by a user within the system. These clicks should not include the clicks need to enter various lengthy form data. Distant task being those task outside the current section a user finds themselves within a system.

Trade Offs
A prominent trade off of this requirement is the security of the system as the more usable the system becomes the less authentication factors can be used. Also the more simplified the UI becomes the more abstraction is require thus supporting back becomes more complicated thus reducing maintainability and increasing possible chances for errors and security vulnerabilities. Also usability requires efficient performance thus reducing the ability audit everything.

3.3.2 Security requirements

Requirement
The system will need to be fully secured since it deals with confidential information such as personal information, application statuses, financial data and meeting information. Also since the systems main goal is to provide stable and audit-able environment for prospective and renewal application processes the system may not be vulnerable to data tampering or any tampering whatsoever.

The system will have to provide different security roles to the registered users on the system. Any number of roles should be assignable to any user by a system administrator to allow for flexibility.

There will also need to be predefined roles so to allow a stakeholder from list of stakeholders in the vision and scope document, to only have the ability and access to perform the actions as defined by the vision and scope document.
The system administrator should be able to access all the sections in the system and should be able to modify them except where only the system itself has that right.
Another security requirement is that the audit trail must not be allowed to be modified by any human user and only allow the system to be able to insert data into the audit log. Further the audit table should only be query-able by a user with the correct security role.
To further quantify this requirement the system must provide some means of a centralised user authentication that acts as a gateway for all users of the system. Also every time some action is made by the user the user needs to be re-authenticated to insure that the user does have the correct privileges and if any user account changes where made it takes immediate affect. When a user attempts to make an unauthorized action, the execution of that action will be halted and a message will be shown to the user, through the UI, that they are unauthorised to make that action.
Failed user actions will not be logged in the audit trail. SQL injection must be prevented. Also the services should be mutually suspecting of each other and assume the philosophy of least privilege.

Trade Offs
This requirement will influence the usability requirement negativity as the more secure a system becomes the less user friendly it is, this is so because there will be need for more user forms and interactions. Since Usability is of higher precedence then security the impact of this requirement must be minimized as much as possible. Therefore the SoftServe group will try to balance the two requirements in such a way that neither of the two is negated while maintaining the sort after usability and security requirement. Further this requirement will also negatively impact performance as it will result on more code execution per action that needs to be secured.

3.3.3 Audit-ability:

Requirement
This is seen as the third most important requirement according to the client. The system needs to provide some form of an audit trail of all critical actions that occur in the system. Critical actions are considered: user account management operations, login action, logout action as well as any operation by a user that leads to a change in the data stored by the system. The Audit trail will be in the form of a read-only table stored in the database as stipulated in the security requirements above.
To quantify this requirement, the system will need to have an active audit log that can capture all the actions, that meet the list of critical actions, of each user that uses the

system. With regards to usability, the feature needs to run in the background and not be noticeable by any user while they are using the system except if it is deliberately queried.

Trade Offs
This requirement will have a negative impact on performance as it will be adding more overhead the executions of audit-able actions. But since not all the actions are audit-able it should not have a to significant impact. To combat this overhead the audit logging action should be made as light weight as possible. Also adequate storage will be needed since such entries can easily grow exponentially which can lead to a degradation of availability and other performance issues.

3.3.4 Scalability requirements

Requirement
As research is still viewed as small field that is slowly growing by the client this is only the fourth most important requirement for the system. The current aim is to create a scalable system that can support 50 to 500 applicants per year with possible growth to a 1000 or more. This is in line with the figures given by the client and the growth, over the next ten to twenty years, in the research sector of the University of Pretoria.

The system needs to be scalable with regards to the following factors:

- **Performance:** This aspect of the system regards the speed and responsiveness of the system. The system needs to handle large report queries in less than 10 seconds. It should be able to handle any application approval service processing in less than 3 seconds. It should be able to send emails instantly from the system side. Note this may be difficult to measure due internet bandwidth and receiving mail server constraints. Login, logout and user authentication should be handled in less then a second by the system this does not considering the transfer of data to and from the server which is depended on the network connection speed.

- **Storage:** This aspect of the system regards the growth and shrinking of the data that is stored by the system. The system will need to be able to handle a database that is in the range of 1 GB to 15 GB that has the potential to grow even larger. The reason for this stems from the fact that the system will need to store a large amount of active data over a period of about 3 to 8 years before it can be archived. Another requirement of the system is that it will need to support archival functionality and archived data that will eventual store all the old data from the active database for as long a time period as possible. Since the archival database should only grow it will need to be about 15 times larger than the active database. Due to these reasons the calculation of how much space should be required will only be possible once the system is implemented and practically tested.

The suggested linear formula to the above analysis is as follows:

$$A \times ((Xna \times Nna) + (Xra \times Nra)) = B \tag{1}$$

A = Number of years for which archival or active support is intended
B = Number of bytes needed on average for A years
Xna = Average number of bytes per new application
Xra = Average number of bytes per renewal application
Nna = Number of new applications a year
Nra = Number of renewal applications a year

- **Concurrency:** This regarded as the amount of active users on the system at the same time. The system will need to support at least 100+/- concurrent users efficiently and effectively since the system requires multiple stakeholders to part take in the application process while there can be multiple applications occurring at the same time.

Trade Offs
This requirement will require that the code undergo optimization and as well as certain functionality which can reduce the maintain-ability of the code itself. It can also lead to reduction of system robustness.

3.3.5 Availability:

Requirement
The system's availability on designated networks will depend on the availability of the servers of the hosting service's that will be used to host the system. None the less, the system should theoritical be avaible for use any hour of day or night.
In order to quantify this it should be seen as follows: If the hosting service's servers hosting the system are active and provide access over a designated network then the system must be available over that designated network. Where the designated networks are defined as the internet and/or the local network where in the server of the hosting service is located.

Trade Offs
Due to the independences of the requirement it does not directly impact other requirements, but the availability as described above will expose the system to the outside world. Thus increasing the risk of attacks. Also if the system is not scalable then a sudden influx of information exchange due to its availability will cause the system to degrade or even fail.

3.3.6 Robustness:

Requirement
This requirement was identified by the SoftServe group as important since the system will be receiving and handling large amounts of sensitive data. Thus the integrity of the data must be insured. To do this the system needs to be robust in terms of its data validation routines, data storage routines, security routines, and system integrity checks.
These four aspects of robustness are quantified as follow:

- **Data validation**: This aspects refer to the high level data validation that occurs at a user interface level. All data that is manually imputed by a user, such as text, numbers, emails, passwords, etc. Must be checked to see if they meet constraints imposed on them through the use of data type checking and regular expression analysis. Any section in the system that requires certain data to completed by the user before performing a action must be checked for completeness before allowing the user to perform the action. Where completeness means the data is not empty and validate according to constraints imposed on it. Data is only sent to lower levels of the system after the data has been validated.

- **Data storage**: When the data is actually stored the data needs to be validated against a set of low-level storage rules and constraints. Many DBMS packages allow this type of validation through data typing and allowing the database administrator to specify constraints. Note that this validation is independent from the validation that occurs at the user interface level.

- **Security**: As mentioned above in the security requirements, the re-authentication of users is seen as a robustness check. Another robustness check for security is that a particular user may only have one of either an account retrieval token or on-demand user account creation token active at a time else it is regarded as fault.

- **System Integrity**: If any malfunctioning modules are detected within the system any user action will not be completed and the user will be notified immediately with the appropriate error message. So prevent any data corruption due to system faults.

Trade Offs
This requirement might have an effect on the scalability requirement of the system. As specified above, request should be handled in a timely manner and this requirement will increase the responses time of commands. It will also complicate the code hence reducing maintainability and performance of the system.

3.3.7 Testability:

Requirement
The system must be testable in order to show that the system meets the clients expectations. This will be done using user acceptance testing, integration testing and unit testing. The test plan will be laid out in the Non-Functional and Functional testing documents of this project. Each of which will provide a set of tests to be performed in order to verify the systems compliance. Therefore each non-functional and functional requirement needs to be quantified in a manner that allows the SoftServe group to test for it in the system. This requirement will help ensure the other requirements are met and all in all improve the system as a whole. Below are the suggested methods quantifying the tests for each type of test:

Unit testing will test each module and its unit in regard to:

- **Pre-conditions** of each unit contained in a module.
- **Post-conditions** of each unit contained in a module.
- **Result and request objects** of each unit contained in a module.

The integration tests will test the system with regard to:

- **Module communication** between units and if their interfaces match the expected interfaces.
- **Functionality** of the system in contrast to the required functionality.

User acceptance testing will test the whole system in regard to:

- **Non-functional requirements**: If the system meets the non functional requirements as stipulated by the architectural requirements documents.
- **Functional requirements**: If each of the expected system functional requirements are met as stipulated in the functional requirement and application design document.
- **Client acceptance**: If the system is at a level that client accepts the system and wishes to deploy it.

3.3.8 Flexibility:

Requirement
The system should be designed in such a way that adding new features can be done without extensive code restructuring or reverse engineering. The process of adding new features should require the programmer working on the system to easily create a new module for the new functionality. For this to be realised the system will should use appropriate design patterns that allow this. For example the strategy, dependency injection and command design patterns.

Trade Offs
This requirement will lead to an increase in complexity as implementing some of the design patterns will still require satisfying the large set of requirements.

3.3.9 Re-usability:

Requirement
The system's components should be designed in such away that they can be reused in different sections in the system or even in external systems. For example the user access control module should be portable to other systems that can be developed.

3.3.10 Maintainability and extensibility:

Requirement
The system should be maintainable in regards to 3 aspects:

- **Work-flow and persistence maintainability**: This aspect deals with structure of the application work-flow with regards to it adapting to change. Also it deals with the underlying persistence layer and how adaptable and flexible it is. The work-flow should allow for customizable pre and post conditions for application stages that can be changed with out recompiling or redeploying the application. The system will not employ a work-flow engine as it expects that the application processes will have to be manually reprogrammed and redeployed in the form of patches since the client has stipulated that the application process changes seldom and if it does it does not change greatly. As with regards persistence storage the data store needs to support excellent schema evolution and provide support for complex real world items and relations.

- **System housekeeping**: This aspect revolves around the actually administration of the system once it is deployed. The system should prevent the system administrator from unnecessary worrying about automated tasks. But the system should allow the system administrator to be able to manage user data, perform backups manually, archival routines and system configuration manually.

- **System development**: This aspect revolves around the developers of the software. The system must be developed in such a way that the source code is easy to maintain and highly readable such that debugging and patching difficulty can be reduced. Also this ensures that the code can easily be given to a third party if need be. The coding documentation standard that must be adhered is that stipulated by the JavaDoc standard. The coding style convention must be defined in such away that the code is easy to read and structured in a logical manner. The identifiers used must be named appropriately and must follow the camelCase standard except where the identifiers are constants in which case the entire name will be in capital letters. The style suggested is a variant of the Allman style:

Coding style sample:

```
public void run()
{
        /*This is a multi line commment*/
        //This is a single line comment

        for (int i = 0; i < x; i++)
        {
                int x = 0;
                int y = 0;
                while (x == y)
                {
                        if (x == y)
                        {
                            something();
                        }
                        else
                        {
                                somethingElse();
                        }
                }
        }
}
```

3.4 Integration requirements

The client's primary integration requirement is to completely integrate with UP's PeopleSoft management system. With the regards the vision and scope document it should be noted that the knowledge the SoftServe group has of the Peoplesoft system is very limited due to UP's IT department's prerequisite for the project. The system has to integrate with the PeopleSoft as follows:

- Must be able to create exportable data packages containing the information of prospective and current research fellows that can easily be loaded into the PeopleSoft system or used by other parties. In terms of quality requirements:

    1. **Performance:** The system should be able to handle the export and importing of application information at a reasonable rate of about at least 10 full application data packages in a second. The system should be able to import and generate on-demand user accounts at a rate of 50 per second. For exporting it should be able to do so at 60 accounts per second.
    2. **Scalability:** The system should be able to export or import as much data as specified by the user as long as it does not exceed the size of the database or the size of users secondary storage.
    3. **Audibility:** Any export and import should be recorded by the system.

- Must be able to query PeopleSoft system to retrieve the account information of internal individual or group Stakeholders to create a uniform login experience. As specified by the vision and scope document this will not be possible at the time of writing since no API can be given to the SoftServe group.

- To standardise information the following is to be done:

    1. Must assign Prospective fellows with a id number in such a way that is integrable with UP Emplid. SoftServe suggests: "f" + 8 digit code.
    2. Stakeholders that have a Peoplesoft account must use their UP emplid id as a user name. The password may vary.
    3. Must be able to provide access to the NRF researcher ratings list but not utilize it as the client has stipulated human error is possible in maintaining of those ratings.

SoftServe has also decided to add two further integration requirements in an attempt to improve future integration with other systems and to improve the application process. The two requirements are as follows:

- **RESTful API/Web services** - The system will need to be able to provide a set of RESTful web services that is able to make use of all the back-end services provided by the system. This will allow future systems to easily integrate with other systems who wish to make use of the system. The interchange formats for the data should be limited to XML and JSON as these are the most universal and at the time of writing the de facto data formats. This will also help solve any future integration with the PeopleSoft system.
- **Google Scholar** - The system will need to be able to integrate with the Google Scholar services. The service needs to allow the system to be able to create full fledged queries for google scholar and be able to retrieve their results. This will be used to allow applicants to quickly populate their CV references by having the system quick search google scholar. There is unfortunately at the time of writing no such API. Therefore the SoftServe group will need to create our own API. This proposed by employing a HTML scrapping technique.

3.5 Architecture constraints

The following architecture constraints have been selected by SoftServe group as being suitable for the system. Please note due to the client's inexperience with software development and the technologies that exist she has allowed the SoftServe group to use their expertises, to decide on the appropriate set of architectural constraints that can meet the above requirements.

1. Database technology: MySQL
2. Programming paradigm: Object-Oriented
3. Programming language: Java
4. Development platform: Java-EE 7
5. System architecture: Java-EE
6. Architectural frameworks: JSF, Java Persistence API, JAX-RS
7. Architectural pattern: Layered (Multi-tiered)
8. Development technologies: EJB, JSP, Expression Language, JDBC, JTA, Java EE Connector Architecture, JAF, JavaMail, JNDI, Concurrency Utilities for Java EE, JAXB, PrimeFaces

9. IDE: Netbeans 8.0
10. Build environment: Apache Maven 3
11. Web server software: GlassFish 4.0 server
12. Web interface protocol: HTTPS
13. Client web browser: Microsoft Internet explorer 9+, Google Chrome 30+, Mozilla FireFox 20+, Opera, Safari
14. Client device operating systems: Windows, iOS, Android, Linux.
15. Proprietary software: Cannot make use of this due to financial constraints.

4 Architectural Patterns and Styles

This section describes the various architectural patterns that will be employed by the project. It should be noted some of these patterns are implicitly employed by the system due to the development approach.

4.1 Layered pattern

The system will explicitly employ 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. This pattern is widely used and provides various benefits. Some benefits include modularity, encapsulation, re-usability of components, decoupling, system maintainability, pluggable-ness of layers and complexity reduction [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 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 architecture that will be employed by the system:

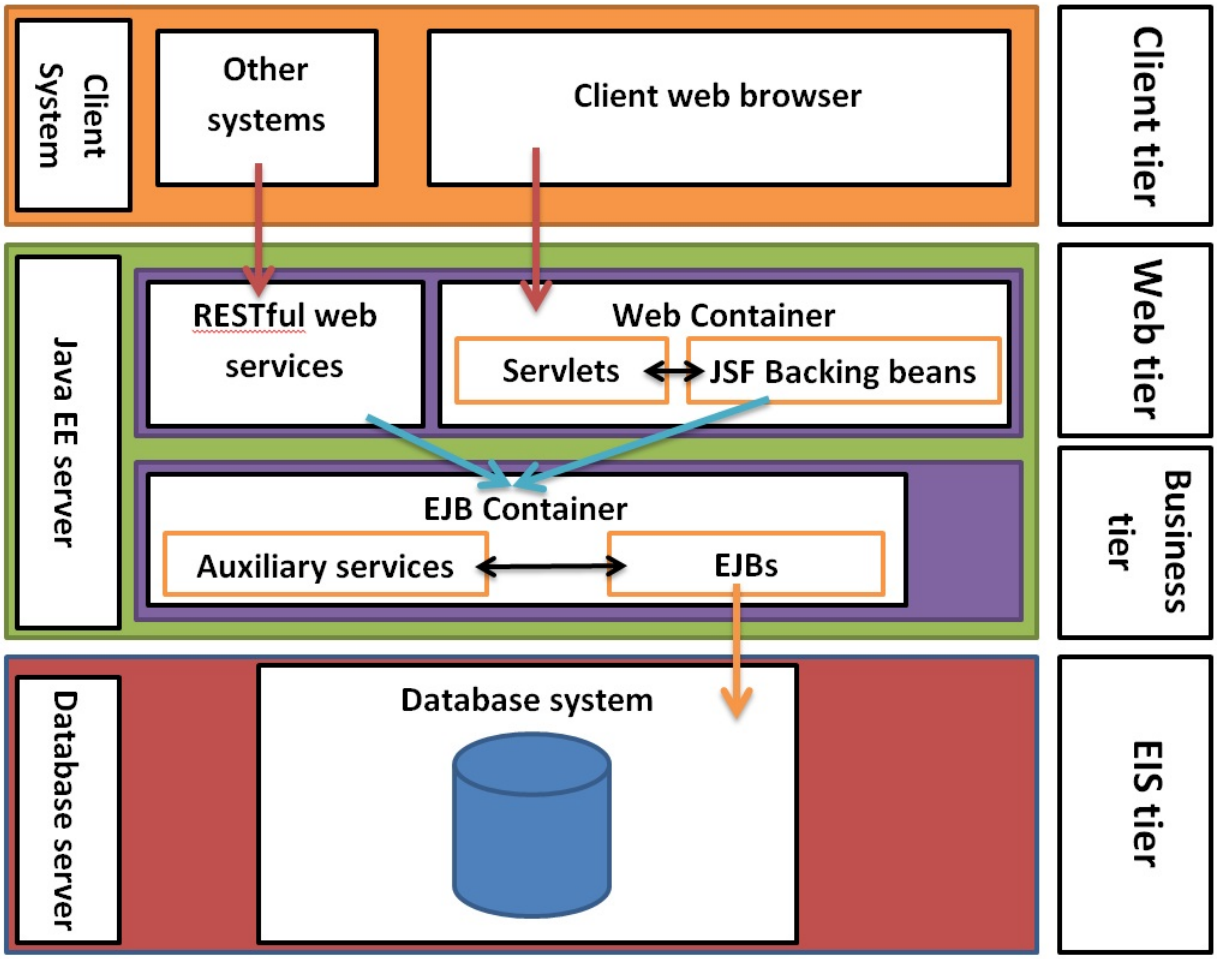

Figure 1: Java EE system architecture with respect to the Post-Doctoral management system

4.1.1 Client layer

This layer runs on the client system and encapsulates the various components that a client system may use to access the Java EE server-side layers. 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 make use of the JSF frame work to provide web pages via a web browsers and RESTful web-services to provide access to the system via other client side applications. [Oracle, 2014]

4.1.2 Web layer

This layer runs on the Java EE server and hosts the Web container and RESTful web services. It provides the management and web page generation support for the web pages that the system has to provide to the client Tier through the use of of the Java ServerFaces

Facelets and backing beans. It also facilitates the communication between the business layer and client layer and thus handles the data validation and transformation required by such communication.[Oracle, 2014]

4.1.3 Business layer

This layer 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 of POJOs that represent various persistence entities of the persistence layer, system messages, sessions, services. It also hosts any auxiliary services that the application wishes to implement in order to help the EJBs. This layer communicates with EIS layer in order provide access to the database and various other lower level infrastructures that the Java EE application requires. [Oracle, 2014]

4.1.4 Enterprise Information/Persistence layer

The EIS layer provides mainly the support for database systems that are used by the Java-EE application. This layer 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]

4.2 MVC Pattern

This pattern is implicit due to the design of the JSF framework and the Java-EE layers. The JSF framework is built according to the MVC pattern. And for case regarding Java-EE, the layers can be grouped as into the 3 different components of the MVC pattern as follows:

4.2.1 View

The view is what is presented to the user and is the front-end of the system. Thus this is the Client layer in Java-EE and is the face-lets in JSF. This component makes use of the controller in order to provide information to the user and send information to the Model.

4.2.2 Controller

The controller is the controlling mechanism of the system and provides the functionality and business logic of the system. Thus this encapsulates the Web layer and EJB layer of Java-EE as they provide the data conversion and business logic of the system. In JSF this is the backing beans.

4.2.3 Model

The Model is the data storage component of the system and provides the mechanisms necessary to store data and use it for use by a system. Thus this represents the EIS/Persistence layer in Java EE as it deals with the data stores. In JSF this is the persistence layer.

4.3 Pipes and Filters

This pattern is a consequence of the layered pattern employed by the system. Since the data that passes from layer to layer is encapsulated by a logical pipe which is the path of the data flow in the system. In this pipe the layers and the operations within the layers which the data flows through acts as filters as they may add or remove data from the objects passing in or add or remove inputs to the next layer or operation.
The application work flow can also be seen as a pipe and filter which is bidirectional. Where the pipe is the process flow and the filters the stages in the work flow. It is bidirectional since the applications can move forward and backwards through the work flow.

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/Spreading work load

The Java Platform offers support for concurrent programming, which is the basis for implementing many of the services offered by Java EE containers via concurrency utilities. This is realized through the use of thread pooling, threads, queuing and scheduling strategies.(Cervera-Navarro, Evans, Jendrock, Haase and Markito 2014).

5.2 Exception handling

To improve the robustness and reliability of the system the system employees various usages of try catch control structures and exception classes. This helps prevent system failure and improves availability. Also user friendly messages will be displayed to users using a messing framework to notify users about system failure or exceptions.

5.3 Logging

To improve security, audibility, reliability and maintainability the system logs exceptions and various other information such as queries and system status on a server log for later review. Thus it can be used to pinpoint bugs or attacks.

5.4 Testing framework

In order to satisfy reliability a testing framework will be used by the system in which unit and integration tests are designed and automatically ran at build time. To complement this a mock frame work will also be used to mock out dependencies for unit tests.

5.5 Checkpoints and roll-backs

To prevent the data integrity from being compromised when storing various items to the database in single operation the system will make use of two-phase commits and a entity manager that will only commit in a single transaction once all the data has been created/removed/modified in the temporary entity graph. Thus if an error occur all changes are rolled back to the previous checkpoint.

5.6 Maintaining backups

To improve reliability of the system the system will allow for full backups of the entire working database to be created.

5.7 Interceptors

To improve security and audibility interceptors will be employed by the system to detect the execution of methods that are demarcated by appropriate annotations. These interceptors will process the inputs to see if the user can use the service or if it needs to be logged by system in the working database. Further they will be used to improve maintainability by checking pre and post conditions of demarcated functions.

5.8 Minimize access channels

Another step taken to improve the security of the system was to minimize the access channels to only that of the web browser interface and RESTful API. Thus allowing the focus of security implementations on those provides.

5.9 Query optimisation

This tactic helps to improve the performance of the system as the system mainly uses database queries to retrieve data from the persistence layer. Therefore optimizing such queries will increase the retrieval speeds. Such optimisations are handled by the JPA provider used.

5.10 Resource locking

In order to improve fault tolerance of the system it will lock resources which are in use such that other processes cannot use it and possibly corrupt it the data. The system will accomplish this by using locking mechanisms provided by the DBMS and Java EE.

5.11 Minimize complexity

In order to improve maintainability of the system as well as security the system will be implemented in such a fashion that it is easy to read and understand and uncomplicated as possible. This is already a implication from using the 4-layers as per Java EE. To aid this approach already matured and well proven software libraries will be used as well as most of the default Java EE technology implementations in order to provide solid abstractions of certain services needed.

5.12 Authentication and authorisation

To provide security for the system an authentication mechanism, which makes uses of role based security and ownership verification in order to provide authorisation, will be used. The default JAAS framework for Java EE will not be used as it does not provide ownership verification of objects.

5.13 Default to deny and least privilege

To ensure confidentiality and prevent attacks the system will be implemented in such away that if an operation cannot verify the session it will deny the action. Also each role will be assigned in such a way that the user will only have access to bear minimum services needed in order to complete their tasks.

5.14 Mutual suspicion

In order to improve fault tolerance and prevent attacks, operations implemented by the system will attempt to verify and filter input data in such a manner as if expecting it to be malformed or malicious.

5.15 Self-describing data structures

To improve flexibility certain data will be stored using self describing data formats such as XML and JSON. Also due to the object orientated paradigm and the Java language reflection mechanisms are provided in order to analysis objects which will be employed by the report generation infrastructure.

5.16 Automation of builds and testing

To improve testability and maintainability as well as extensibility aspects of the system the project will make use a build automation tool in order to manage libraries, dependencies and automate unit testing.

5.17 Support for standard communication protocols and channels

To fulfil the integrability and accessibility requirements the system will make use of the HTTP/HTTPS protocols to provide RESTful web services and as well web interface.

5.18 Abstraction of services via interfaces

This helps improve the extensibility and maintainability of the whole system by allowing the services implemented by a layer to be hidden by an abstract interface whose concrete implementation can be changed without affecting the dependences on the service.

5.19 Design Patterns used

To help improve maintainability and design of the system various design patterns will be employed.

5.19.1 Builder Pattern

The Builder Design Pattern allows for the construction of complex structures in small steps. An example of how this is used explicitly is the criteria builder for DBMS queries. Implicitly the entire application work-flow is also a builder as it adds data to the applications in small quantified steps. This construction of objects in small steps decreases coupling and makes the code more testable and maintainable. The design helps improve the flexibility as well as the maintainability of the code.

5.19.2 Factory Design Pattern

The Factory Design Pattern provides centralised means to create similar objects. For example database entities and database access objects. 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 the creation of new objects is much simpler. It also helps avoiding code redundancy.

5.19.3 State Design Pattern

The State Design Pattern is best used in situations where the actions take place in a predefined 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.

5.19.4 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. An example for this is the injection of EJBs in one another or in other managed beans.

5.19.5 Database Access Objects

The Database Access Object is used by the EJBs to provide specific data operations without exposing implementations of the of such operation to external objects. It forms part of the Core J2EE Patterns. This design pattern acts an intermediary between the system 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 system. This pattern improves the audit-ability as well maintaining the integrity of the data contained in the system. Further it reduces code redundancies therefore improving maintainability.

6 Reference Architecture

The system will employee the Java-EE reference framework. The core design philosophy of Java-EE is to provide a 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's 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 this project.

7 Frameworks

7.1 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.

7.2 Java Persistence API

Java EE is based on Java which is an object-oriented language. Whereas most modern day database management systems, DBMSs, provide a range of different database approaches. Thus to bridge this gap the Java Persistence API is used. This is ideal for our project as it provides an Object Relational Mapping solution which allows the relation database which our solution will use to be viewed as a object-oriented database. Also it will allow the system to easily move to other DBMS if ever needed. The Java persistence API contains the following components:

- Persistence API
- The query language
- ORM

7.3 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. Allowing the system to easily be accesses by mobile and computer platforms alike and also be accessible over most company firewalls as it will make use of the HTTPS protocol. This will allow for future integration and 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]

7.4 JUnit

JUnit is a simple unit testing framework used to write repeatable tests. This will be employed by the project in order to manage the unit tests that will be implemented by the system.

Thus allowing the system to achieve the testability requirement of the system.

8 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.

8.1 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

8.2 Integration Channels

As mentioned earlier and in the vision and scope 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 either the persistence layer using the Java EE Connector Architecture API or the web layer using the JAX-RS API.

9 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.

9.1 Integrated Development Environment

The system should be build-able 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.

9.2 Build Tools

- **Apache Maven** - This build tool was chosen due to the flexibility and power of it in terms of configurations, dependencies management, its ability to automate tests and the fact the project can easily be ported over to other IDEs such as the Netbeans IDE or even no IDE.

9.3 Server 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

9.4 Development technologies

9.4.1 Annotations

This technology provides a means for developers to provide meta data for various data structures and operations in consistent and un-interfering manner within source code. This meta

data can be propagate to various levels of *-time stages in program lifestyles.

Reasons for use
Various features of the Java EE framework makes use of this technology to carry out its functions. Also this will be used by interceptors to extract information at run-time and annotation processors at compile time to configure the system accordingly.

9.4.2 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 layer. 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.

9.4.3 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.

9.4.4 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.

9.4.5 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.

9.4.6 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.

9.4.7 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 layer of the system in order to retrieve and store data in the databases.

9.4.8 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 layer and the controlling of such resources.

9.4.9 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 layer and thus will be required.

9.4.10 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.

9.4.11 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.

9.4.12 Dynamic Jasper Reports

Description
The Dynamic 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 needed by the system on a functional requirement level.

9.4.13 Jsoup

Description
Java library for working with real-world HTML. It provides a very convenient API for extracting and manipulating data, using the best of DOM, CSS, and jquery-like methods.

Reasons for use
Used to provide a means to integrate with the Google Scholar results via HTML scrapping.

9.4.14 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 Language. 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.

9.4.15 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.

9.4.16 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.

9.4.17 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. It should be noted due to to financial constraints the system can not make use of ObjectDB which is an OODBMS which has various advantages over MySQL and is easily integrate-able with the JPA API. Thus at some point in time when financial backing provided this technology can be replaced by ObjectDB.

9.4.18 Jackson XML and JSON processor

This technology provides a means to map POJOs without the need for annotations or other meta information to or from JSON or XML.

Reasons for use
It is well documented, matured and fast JSON and XML processor which does not need meta data and is excellent for converting dynamic data to and from text. Thus it is ideal for use in the RESTful web services and annotation processors.

9.4.19 Jersey

This technology is the default implementation of the JAX-RS API.

Reasons for use
It is used by Glassfish server and is also very mature implementation. Thus it is required by the project to provide RESTful web services.

9.4.20 Eclipse Link

This technology is the default implementation of the JPA API.

Reasons for use
It is used as the default implementation for the JPA API provided by Java EE. Therefore it is well documented and well tested.

9.4.21 Context and Dependency Injection (CDI)

This technology is used to provide a dependency injection framework for which to provide Inversion of control support to various components of the Java EE framework.

Reasons for use
This technology helps provide maintainable code and forms part of the core of Java EE thus it is required.

9.4.22 Javax Script library

This technology is used to parse and execute scripts from various supported scripting languages within the run-time of the software.

Reasons for use
This technology will help provide a means to allow for the creation of custom pre and post conditions that will used by the system.

10 Glossary:

- **Activity diagram** - A UML diagram that depicts the flow of actions or activities in the process.
- **API** - Application Programming Interface
- **Application** -Both renewal applications or new fellowship applications are seen as applications by this project.
- **CV** - Curriculum Vita
- **Domain objects** - Are the objects that are present in the system being modelled.
- **EAI** - Enterprise Application Integration
- **NRF** - National Research Foundation
- **Spreadsheet** - A special type of computer document that is used to represent data in rows and columns.
- **GlassFish** - GlassFish is a web server software package that is very flexible and compatible with Java EE applications.
- **HTML** - Hyper Text Mark-up Language
- **HTML Scrpping** - technique of extracting information from websites
- **HTTPS** - Hyper Text Transfer Protocol Secure is a higher level network oriented communication rule set that is highly secure and is used by all web browsers.
- **Java EE** - Java Enterprise Edition
- **MySQL** - Is a relational persistence database package that provides all the necessary management tools to run and manage a database server.
- **Object-Oriented** - A programming language style that encapsulates everything as an object instance of a particular class of attributes and methods.
- **JDBC** - Java Database Connection
- **MVC** - Model View Controller
- **PDF** - Portable Document Format file
- **Peoplesoft** - A management system designed by oracle.

- **Spreadsheet** - A special type of digital document that is used to represent data in rows and columns
- **UI** - User Interface
- **Use case diagram** - A UML diagram that gives a visual depiction of a service or group of services.
- **UML** - Unified modelling language. A commonly used model standard to provide technology neutral models of different aspects of software.
- **UP** - University of Pretoria
- **Work Flow** - Describes the tasks, procedural steps and tools needed for each step in a business process