

Architectural Requirements Document
Specification
Project:
Cafeteria Management System:
Resolve

T-RISE

Rendani Dau (13381467)

Elana Kuun (12029522)

Semaka Malapane (13081129)

Antonia Michael (13014171)

Isabel Nel (13070305)

July 31, 2015

Contents

1	Introduction	5
2	Vision	5
3	Background	5
4	Architecture Requirementss	6
5	Architectural Responsibilities	6
6	Quality Requirements	7
6.1	Security	7
6.1.1	Reasons for security as a quality requirement	7
6.1.2	Strategies to achieve this quality requirement	7
6.1.3	Pattern to achieve these strategies	7
6.2	Usability	8
6.2.1	Reasons for usability as a quality requirement	8
6.2.2	Strategies to achieve this quality requirement	8
6.2.3	Patterns to achieve these strategies	9
6.3	Reliability	9
6.3.1	Reasons for reliability as a quality requirement	9
6.3.2	Strategies to achieve this quality requirement	10
6.3.3	Patterns to achieve these strategies	10
6.4	Auditability	10
6.4.1	Reasons for auditability as a quality requirement	10
6.4.2	Strategies to achieve this quality requirement	11
6.4.3	Patterns to achieve these strategies	11
6.5	Scalability	12
6.5.1	Reasons for scalability as a quality requirement	12
6.5.2	Strategies to achieve this quality requirement	12
6.5.3	Patterns to achieve these strategies	12
6.6	Integrability	13
6.6.1	Reasons for integrability as a quality requirement	13
6.7	Testability	13
6.7.1	Reasons for testability as a quality requirement	13
6.7.2	Strategies to achieve this quality requirement	13
6.8	Nice-To-Have	13
6.8.1	Performance	14
6.8.2	Flexibility	14
6.8.3	Maintainability	15

7	Access and Integration Requirements	15
7.1	Access Channel Requirements	15
7.1.1	Human Access Channels	15
7.1.2	System Access Channels	16
7.2	Protocols	17
7.2.1	HTTP - Hypertext Transfer Protocol	17
7.2.2	TCP - Transmission Control Protocol	17
7.2.3	SMTP - Simple Mail Transfer Protocol	17
7.3	Architecture Constraints	17
7.3.1	Technologies	17
7.3.2	Operating Systems	18
8	Architectural Patterns	19
9	Other Supporting Documentation	20
10	Bibliography	21

Document Title	Architectural Specification Document
Document Identification	Document 0.0.2
Author	Rendani Dau, Isabel Nel, Elana Kuun, Semaka Malapane, Antonia Michael
Version	0.0.2
Document Status	Second Version - contains edited architectural requirements

Version	Date	Summary	Authors
0.0.1	29 May 2015	First draft contains architectural requirements	Rendani Dau, Isabel Nel, Elana Kuun, Semaka Malapane, Antonia Michael
0.0.2	30 July 2015	Second draft contains edited architectural requirements	Rendani Dau, Isabel Nel, Elana Kuun, Semaka Malapane, Antonia Michael

1 Introduction

This document contains the architectural requirements for the Resolve Cafeteria Management System that will be created for Software Engineering (COS 301) at the University of Pretoria 2015, by the group T-RISE. In this document we will thoroughly discuss and layout the project's architectural requirements to provide a clear view of the system as a whole.

2 Vision

The vision of this project is to implement a flexible, pluggable, fully functional software application that will be maintainable, with detailed supporting documentation and an instruction manual for the Cafeteria Management System. This system will assist in managing the cafeteria's inventory/stock, executing orders from the cafeteria, generating bills and sending these to the appropriate parties and facilitating payments for access cards (or the use of unique access card numbers).

3 Background

As specified in the project proposal document from Resolve - the cafeteria is currently cash only and does not accept bank cards or electronic payments. This makes it inconvenient for employees as they have to carry around cash if they want to purchase anything from the cafeteria. Hence, this is equivalent to purchasing from an external food outlet where they can also pay with their preferred method of payment. The employees have to hence use up fuel and time and lastly this does not bring in the maximum amount of income to the cafeteria, hindering its growth and improvement.

Resolve is therefore looking for a means to accept payments from employees for the canteen using their employee access cards or access card numbers, with an amount being deducted from their salary at the end of the month.

Resolve proposed the Cafeteria Management System to assist with this problem. After our first meeting with the client, they brought to our attention that at times the cafeteria does not even have enough stock to provide some of the menu items, thus the managing of inventory or stock will also be part of the system. The system will also predict what inventory/stock needs to be bought for the next week in order to avoid such a problem. At the end of each month, the bill for the month will be sent to either payroll

or to the employee. This option is configurable from the user's profile. The employee can also set a spending limit for each month for control purposes. The system will have its own maximum, such that users cannot set a limit that exceeds this.

4 Architecture Requirementss

The software architecture requirements include the access and integration requirements, quality requirements, and architectural constraints. These points will be thouroughly discussed below.

5 Architectural Responsibilities

- The system must allow a user to view the menu without being logged in
- The system must allow a user to select items from the menu and proceed to place the order
- The system should not process an order unless a user is logged in
- On placement of an order the system should send the order details to the cafeteria
- The system should notify the user when their order is ready
- The system should send a bill to payroll or the user once a month or both (depending on preferance)
- The system should allow a user to manage their profile by editing their password, email and spending limit
- The system should allow configiration and branding by a super user or an admin user
- The system should allow management of inventory and management of the cafeteria
- The system should provide casheir functionality
- The system should provide for multiple deployment
- The system must be usable, scalable, reliable and auditable

6 Quality Requirements

6.1 Security

6.1.1 Reasons for security as a quality requirement

- Security is one of the most important requirement for the Cafeteria Management System because using the system involves storing confidential details such as personal e-mail addresses, employee ID's and financial information - the amount the user spend will be deducted from his/her salary. The user must hence have full control over the amount spent.
- There are also Super user and Admin user roles who have certain functionality assigned to them, dealing with the confidential details of the users and of the cafeteria, hence ordinary users must not be able to access any functionality besides the functionality assigned to their respective roles. Hence, security is of utmost importance.

6.1.2 Strategies to achieve this quality requirement

- Detecting attacks by determining message integrity, event logging and analysis, scanning for attack signatures and auditing sensitive events (Solms, 2014).
- Resisting attacks by limiting access and exposure. This can be done by minimizing access channels, minimizing access domains, authentication, confidentiality, assuming external resources as untrusted, minimizing hosting of external resources, additional security layers for valuables.
- Recovering from attacks by dropping connections and requests. In addition, updating access rules and restoring states.

6.1.3 Pattern to achieve these strategies

- Layering

Layering can enforce security due to separation into layers, in which different users have different levels of access to the layers, depending on their roles in the system. Layering can be used within the subsections of MVC, i.e. the Control and Model layer can be layered to further divide concerns and restrict different users' access on those layers, providing tighter security.

6.2 Usability

6.2.1 Reasons for usability as a quality requirement

- Usability involves measuring user's performance with regard to the use of a software system, since the users of the cafeteria system will be in different divisions within the company and have different technological skill levels, it should not be hard for new users to become familiarized with the system, thus the system has to be user friendly in all aspects.
- Due to the fact that Resolve has not had such a system in place before, the novice user would not have had exposure to such a system therefore the system should also be memorable as the first of its kind at Resolve.

6.2.2 Strategies to achieve this quality requirement

- Various goals of usability requirements are firstly, that the interface is intuitive, i.e. easy to navigate and understand, that the buttons and icons are self explanatory for the primary users.
- The interface must also not be a cluttered, frustrating and overwhelming one.
- Ease of learning is also an important goal, such that users who have never used such a system can catch on easily, for example users who have used applications which work on the same principle like an online shopping application (which most users would have some experience with in today's age) will find it easy to work with.
- The system must also be task efficient, i.e. if users access this space regularly, long tedious processes and other admin must be avoided.
- Also, the colour schemes, functionality and interactiveness of the interface and system must contribute to this task efficiency.
- Different usability tests can be conducted such as handing out paper prototypes of different interface designs, and questionnaires getting feedback from the sample of people that were consulted in the survey. Problems with the different interfaces can be picked up during the usability testing phase, as indicated by the sample of users consulted, such that the final product will be much more user friendly. (<http://www.usability.gov/what-and-why/usability-evaluation.html>)

6.2.3 Patterns to achieve these strategies

- MVC
- Layering

MVC is a suitable pattern because the user will only need to interact with the front end interface, rather than dealing with the technical aspects of the back-end system. Another reason for this is that the developers allocated to working on the View will have the sole focus of making it usable. Layering can be used within the subsections of MVC, i.e. the Control and Model layer can be layered to further divide concerns and allow different people to work on those layers.

6.3 Reliability

6.3.1 Reasons for reliability as a quality requirement

- The reason we have placed great importance on reliability is because a large scale of users (Resolve employees) will be using the system concurrently in the one hour lunch break thus the system should be able to notify each user separately when their orders are ready and a bill should be generated for each user every month and it will store the credit made by a user that he/she owes the canteen and reliability of this system of keeping the information save and correctly is thus very important.
- The system should ensure that the correct amount is deducted from the user's account and that the user's set limits are adhered to.
- Whilst it may not be possible to design software that is failure and defect-free, software needs to be tested and debugged until a satisfactory level of reliability can be achieved, reliability plays a key role, ensuring that all functions work as the user expects them, when the user requires to use the system, thus things like proper unit testing goes hand in hand with reliability.
- It must hence have a maximum of 2 or 3 hours down time a week (The ideal will be no down time at all, but unfortunately we live in a non-ideal world so one needs to be realistic).
- The system should be reliable in terms of ensuring that the different functionality is assigned to users with different roles and no user can

access functionality outside their role, thus reliability also have a close correlation with security of the system.

- Maintenance of a system is also important for reliability, a system must be maintained and kept up to date at all times to ensure that the system stays reliable and any faults found while the system is live should be fixed and the system should be updated thereafter.

6.3.2 Strategies to achieve this quality requirement

- Firstly, the prevention of faults. This is done by testing the system thoroughly, using resource locking as well as removing single points of failure (Solms, 2014). We will do this with thorough unit testing on both the client and server side and later on also with proper integration testing between all modules.
- Secondly, detection of faults, which is achieved through deadlock detection, logging, checkpoint evaluation and error communication to name but a few. Recovering from faults also falls under reliability. This is done by passive redundancy, maintaining backups and checkpoint rollbacks. (Solms, 2014)

6.3.3 Patterns to achieve these strategies

- MVC

The MVC pattern can be used for reliability, because since the different layers are clearly separated, hence particular teams are focused on working on each layer, making the system more reliable.

6.4 Auditability

6.4.1 Reasons for auditability as a quality requirement

- Any action performed on the system should be traceable back to the person who made these changes and when these changes were made.
- In the event of a system crash, it should be possible to roll the system back to a previous working state.
- The super user should be able to view every other user's activities. This is a part of the monitorability aspect of the system.

6.4.2 Strategies to achieve this quality requirement

- System should have log files running at all times to track all transactions made by users.
- Time stamps should be added to document time and date information of the activities done so that the system can trace through the information when needed, such as the events that precede a system crash or unauthorized access that alters the system in any way.
- System backup should allow rollback when needed.
- ACID test can be carried out. Acid is an acronym that describes the properties of a database or system. The properties are:
 - **Atomicity:** Defined as all or none situation referring to the processes that take place on the system. If something where to go wrong with a process such as posting on the system, then the entire process has to be repeated or not at all.
 - **Consistency:** All processes must be completed. No process can be left in a half-finished state, if a failure is detected in a process then the entire process has to be rolled back.
 - **Isolation:** Keeps process/transactions separate from one another until they are finished.
 - **Durability:** The system must keep a backup of its current state so as to roll back to it if the system where to experience a system failure, crash or corruption of data due to a security breach.

6.4.3 Patterns to achieve these strategies

- MVC
- Layering

MVC is a suitable pattern because it provides auditability through logging all filter inputs and outputs (off queues). Layering is a suitable pattern because each separate layer can be audited and monitored individually, rather than auditing the system as a whole.

6.5 Scalability

6.5.1 Reasons for scalability as a quality requirement

- Scalability refers to a software's ability to handle increased workloads, thus scalability is an important requirement due to the fact that a large volume of Resolve employees will be using the system possibly on the same time each each day (during lunch hour) and hence the system needs to support all these users concurrently.
- In saying this, the system must allow each user to order multiple items and process orders per user concurrently and efficiently.
- With this, we can assume that there will be in excess of 500 users meaning that the system has to have the ability to handle at least 500 concurrent users at peak times and in a case where the amount of users increase drastically, small server side changes needs to be made with ease to handle more users if needed.

6.5.2 Strategies to achieve this quality requirement

- We will need to firstly ensure that existing resources are managed efficiently, i.e. reducing the load using efficient storage, processing, and persistence (thus the server hosting the system should be efficient enough to handle the current amount of users and more). In addition, we will need to ensure that the load is spread across resources and time, using methods of load balancing to spread load across resources as well as using scheduling and queueing to spread load across time.
- Secondly, the resources can be scaled up by increasing storage, increasing processing power and increasing the capacity of communication channels (this can easily be done by increasing processing power, and storage space on the server that is hosting the system).
- Lastly, resources can be scaled out by means of using external resources, using commoditized resources and distributing tasks across specialized resources.

6.5.3 Patterns to achieve these strategies

- Concurrency Master-Slave

We chose this pattern here due to the concurrency of the system, meaning that a large number of users must be able to access the system at a time.

6.6 Integrability

6.6.1 Reasons for integrability as a quality requirement

- Integrability refers to how separate components of the system work together. Different components of the cafeteria should work together correctly.

6.7 Testability

6.7.1 Reasons for testability as a quality requirement

- It is important to ensure that a software system is adequately tested at various levels. In other words, testing is based on the concept of incremental development - as you are busy building the system each aspect needs to be thoroughly tested to ensure the functionality works correctly, before you move on to build the next step - this will make sure that at all times the software is functional with very little flaws (those flaws can refer to logical mistakes and possible untested sections still being built).
- Testing a system goes hand in hand with other quality requirements such as maintainability, reliability and security. All the quality requirements work together to ensure a product of high quality gets produced at the end of the project.

6.7.2 Strategies to achieve this quality requirement

- Unit tests will be conducted, each section of the software needs to be thoroughly tested using unit tests with mock objects ensuring that it works as expected, this includes unit testing on both the client side and the server side.
- Integration testing will also be conducted on both the client side and server side to ensure that all modules work together as expected.

6.8 Nice-To-Have

1. Performance. Performance refers to the behaviour of the system, meaning its response time and throughput - the number of operations performed per second. The cafeteria system should behave as expected and should not take long to respond. Some changes are important and should be reflected in a timely manner, for example, when an email

is sent to a user who forget his/her password, when an employee id is changed, when an order is ready the user should be notified soon.

2. Flexibility. Flexibility refers to the ability of a system to respond in a time and cost efficient manner to internal or external changes that affects the quality of its service.
3. Maintainability. Maintainability refers to how easily the software can be modified to adapt to a new environment, fix bugs, and improve performance. Other developers should be able to add new features, bug fixes, and make changes without investing a significant amount of time and effort. Users of the system should also be able to maintain it by making certain changes to the system. Some aspects of the cafeteria management system such as the branding, the colours, and the menu section needs to be configurable by a user. This is a desirable quality requirement to have as it means that the system can be changed, adapted, and improved, and thus be used for a longer time.

Listed below are the nice to have quality requirements related to the Cafeteria Management System, these requirements are not considered as critical but as desirable.

6.8.1 Performance

- Performance refers to the behaviour of the system, meaning its response time and throughput - the number of operations performed per second. The cafeteria system should behave as expected and should not take long to respond. Some changes are important and should be reflected in a timely manner, for example, when an email is sent to a user who forget his/her password, when an employee id is changed, when an order is ready the user should be notified soon.

6.8.2 Flexibility

- Flexibility refers to the ability of a system to respond in a time and cost efficient manner to internal or external changes that affects the quality of its service.

6.8.3 Maintainability

- Maintainability refers to how easily the software can be modified to adapt to a new environment, fix bugs, and improve performance. Other developers should be able to add new features, bug fixes, and make changes without investing a significant amount of time and effort. Users of the system should also be able to maintain it by making certain changes to the system. Some aspects of the cafeteria management system such as the branding, the colours, and the menu section needs to be configurable by a user. This is a desirable quality requirement to have as it means that the system can be changed, adapted, and improved, and thus be used for a longer time.

7 Access and Integration Requirements

7.1 Access Channel Requirements

In this section we will discuss the requirements for the different channels through which the system can be accessed by firstly, people (users - client side) and systems (server-side).

7.1.1 Human Access Channels

The Cafeteria Management System will be accessed by the different users via the online web page (web interface) or through the mobile application (if one is developed). The web interface will be accessible through all the standard web browsers such as Mozilla Firefox, Google Chrome and Microsoft Internet Explorer. The mobile application will be accessible on multiple platforms including the standard IOS/Android platforms. Different services will be available to different users. There are five types of users: Super User, Cafeteria Manager, Cashier, Normal User, and Resolve Admin. These will be discussed below.

Super User

The super user will be one of 2 administrative users that will have global access to all the functionality of the Cafeteria Management System, in particular the super user will have access to the branding of the Cafeteria Management system (changing the logo and so forth) and Administrative settings

such as the ability to assign roles to users . The super user will hence have access to all the functionality of all the other users listed below.

Cafeteria Manager

The cafeteria manager will have the ability to view his/her own profile, edit his/her profile, and place orders. This user will also be able to add and edit menu items, view the orders placed by the users of the system, view the inventory, and add or remove inventory.

Cashier

The cashier will be able to view his/her profile, edit his/her profile, view the orders placed, and mark off finished orders that are finished and have been collected. The cashier will also be able to make purchases, check inventory and add or remove inventory. Removal of inventory will be done in situations where stock has expired or depleted.

Normal User

The normal user will be a Resolve employee registered on the Cafeteria Management System. A normal user will only be able to view his/her profile, edit his/her profile, place orders, check if their order is ready, and view/print their balance reports and account history.

Admin User

The resolve admin user will be able to view all the registered users, their account history and their outstanding balances. This is for administrative and financial purposes . This role has been requested by the Resolve team.

7.1.2 System Access Channels

The different technologies selected will be used to support the access channels effectively. We will be using NodeJs running on an Express server and the server needs to be connected to the Mongo database on which various data will be stored and retrieved. This data will be transferred from the server to the respective node modules and so forth. The integration channels will also be accessible by the mobile applications, such as Phone Gap, which is the program we will be using to help us convert our web interface into a mobile application.

- The system will have to integrate with the Mongo database, retrieving information of the employees - such as contact information to notify the user that an order is ready, get inventory or stock and so forth.

- The system will also have to integrate with the server to pass information to and from the database.

7.2 Protocols

7.2.1 HTTP - Hypertext Transfer Protocol

Integration with this protocol will occur at a high level and typically be handled by libraries or browser-clients etc. **To be used for:**

- All data transferred between users and the server on which the system is hosted.
- Transfer of miscellaneous data such as HTTP error codes to ensure both servers and clients are aware of the state of data transfers and its results

7.2.2 TCP - Transmission Control Protocol

For establishing network connections between the user computers and the system server. Streams of data can then be exchanged between the connected hosts. Error detection, faulty transmission of data, resending of data etc. will all be done using TCP (Davids). Integration with this protocol will occur at a high level and will typically be handled by libraries or operating system functions.

7.2.3 SMTP - Simple Mail Transfer Protocol

This protocol will be used to handle e-mail communication, specifically notifications to users when their orders are ready, as well as the sending of bills to both users and payroll. It addresses Security as a quality requirement since it incorporates SMTP-Authentication defined by RFC 2554 (Meyers, 1999) which enhances the security of the protocol.

7.3 Architecture Constraints

7.3.1 Technologies

Technologies we will be using in the creation of the Cafeteria Management System includes the following:

- **HTML:** The Software system will be mainly web-based.

- **JavaScript together with AngularJS and NodeJS:** this will enable us to add extra functionality to our web page and modularise the system thus also helping us to implement dependency injection. For creating reports we can also import ReportingJS to create visually pleasing reports that is logically structured
- **CSS together with BootStrap:** which will allow us to style our page and also make it interactive.
- **Mongo DB:** MongoDB for our database which goes extremely well with NodeJS.
- **Express server:** will be set up as our server that will host the system.
- **Phone gap:** Phone gap will be used to convert our web page into a usable application which will then look like the online webpage that will run like a web interface in the background but will seem like a mobile application to the user that will be accessible from multiple platforms.
- **Git and GitHub:** A version control software and a repository website that will be used to host the source code of the Cafeteria Management System. Reasons for the use of Git is its ease of use and because it is free and open source. (<http://git-scm.com/>).

The above mentioned technologies will be our basis we will create our system on, but as we are busy building the Cafeteria Management System we will add other technologies as needed.

7.3.2 Operating Systems

Since server side of the Cafeteria Management System will be an Express server which is based on JavaScript, the CMS server will be able to run on both Windows and Unix platforms, given that they both support JavaScript and NodeJS.

The client side of the Cafeteria Management System will be able to run on almost all operating systems that can run a modern web browser such as Mozilla Firefox, Google Chrome or Internet Explorer (for example, Microsoft Windows, Apple Mac OSX or any distribution of Linux or BSD).

8 Architectural Patterns

For the design of The Cafeteria Management System, two patterns are considered: the **MVC (Model-View-Controller) pattern** and the **Layering architectural pattern**. For concurrency, the **Master-Slave pattern** is used.

The top layers of the Model and the Controller are interfaces for the MVC architecture and the layers below provide the functionality.

MVC is considered because:

- It provides modularity (i.e. the system's concerns are separated, thus easier to implement). (Solms, 2014)
- It allows for better maintainability (one can maintain the Model, View and Controller separately). (Solms, 2014)
- Testability (it is easier to test because of separated concerns, so the source of any problems are easy to identify). (Solms, 2014)
- Reuse (it is possible to take any component and reuse it where necessary). (Solms, 2014)

Layering allows the Cafeteria Management System to have pluggable layers, which will allow the developers to replace layers as needed. This pattern allows for:

- Improved cohesion. (Solms, 2014)
- Reduced complexity of the system. (Solms, 2014)
- Improved testability (which will allow for easier debugging). (Solms, 2014)
- Improved reuse and maintainability of the source code, because all the layers are individual and separate from one another. (Solms, 2014)

However, it should be mentioned that Layering has a performance overhead associated with it, as well as higher maintenance costs associated with the lower layers, because they impact the higher levels. (Solms, 2014). Given the benefits, however, the authors feel that the reduced performance and maintenance costs are a good compromise for reduced complexity and testability.

For concurrency, the Master-Slave architectural pattern (Solms, 2014) is considered, because the system needs to accommodate a large number of users at a time.

9 Other Supporting Documentation

On the GitHub repository the rest of the supporting documentation for the Cafeteria Management System can be found together with the actual implementation of the work.

<https://github.com/toniamichael94/MainProjectCOS301>

10 Bibliography

- *Basic MVC Architecture*. [Online]. Available: http://www.tutorialspoint.com/struts_2/basic_mvc_architecture.htm [Accessed 5 March 2015].
- Beal, V. n.d. *Linux Server*. [Online]. Available: http://www.webopedia.com/TERM/L/linux_server.html [Accessed 10 March 2015].
- Bien, A. 2009. *9 Reasons Why Java EE 6 Will Save Real Money - Or How To Convince Your Management*. [Online]. Available: <http://www.adam-bien.com/roller/abien/entry/8-reasons-why-java-ee> [Accessed 10 March 2015].
- Borges, B. 2013. *Reasons to why I'm reconsidering JSF*. [Online]. Available: <http://blog.brunoborges.com.br/2013/01/reasons-to-why-im-reconsidering-jsf.html> [Accessed 10 March 2015].
- Davids, N. *The Limitations of the Ethernet CRC and TCP/IP checksums for error detection* [Online]. Available: <http://noahdavids.org/self-published/CRC-and-checksum.html> [Accessed 8 March 2015].
- *Git - loval-branding-on-the-cheap*. [Online]. Available: <http://git-scm.com/> [Accessed 10 March 2015].
- *JavaServer Faces Technology*. [Online]. Available: <http://www.oracle.com/technetwork/java/javasee/javaserverfaces-139869.html> [Accessed 10 March 2015].
- Kabanov, J. 2011. *Ed Burns on Why JSF is the Most Popular Framework*. [Online]. Available: <http://zeroturnaround.com/rebellabs/ed-burns-on-why-jsf-is-the-most-popular-framework/> [Accessed 10 March 2015].
- Meyers, J. 1999. *SMTP Service Extension for Authentication* [Online]. Available: <http://tools.ietf.org/html/rfc2554> [Accessed 9 March 2015]
- *Securing FTP using SSH* [Online]. Available: <http://nurdletech.com/linux-notes/ftp/ssh.html> [Accessed 9 March 2015]
- *PostgreSQL: About*. [Online]. Available: <http://www.postgresql.org/about/> [Accessed 10 March 2015].
- *Software Engineering*. [Online]. Available: <http://sesolution.blogspot.com/p/software-engineering-layered-technology.html> [Accessed 5 March 2015].

- Solms, F. 2014. *Software Architecture Desgin*. [Online]. University of Pretoria: Pretoria. Available: <http://www.cs.up.ac.za/modules/courses/download.php?id=8565> [Accessed 9 March 2015].
- Solms, F. 2015. *Java Persistence API (JPA)*. [Online]. University of Pretoria: Pretoria. Available: <http://www.cs.up.ac.za/modules/courses/download.php?id=8640> [Accessed 7 March 2015].
- *SQL Server 2008 R2 Express Database Size Limit Increased to 10GB*. [Online]. Available: <http://blogs.msdn.com/b/sqlexpress/archive/2010/04/21/database-size-limit-increased-to-10gb-in-sql-server-2008-r2-express.aspx> [Accessed 9 March 2015].
- *Usability Evaluation Basics*. [Online]. Available: <http://www.usability.gov/what-and-why/usability-evaluation.html> [Accessed 5 March 2015].
- Whner, K. n.d. *Why I will use Java EE (JEE, and not J2EE) instead of Spring in new Enterprise Java Projects in 2012*. [Online]. Available: www.kai-waehner.de/blog/2011/11/21/why-i-will-use-java-ee-jee-and-not-j2ee-instead-of-spring-in-new-enterprise-java-projects-in-2012/ [Accessed 10 March 2015].
- *Welcome to Apache Maven*. [Online]. Available: <http://maven.apache.org/> [Accessed 10 March 2015].