|  |
| --- |
| **Disclaimer**  This is a **template** for the Software Requirements Specification (SRS) that students may use. It povides a **starting point** for the preparation of SRS.  **Note to authors**  If you add any new sections to the document please make sure that you maintain the header and text styles.  Before submission of the first draft of this document please make sure to update the Table of Contents and to delete this page.  **Author**:  Dr. C. Constantinides <cc@cse.concordia.ca> |

**Software Requirements Specification**

Version 1.0

for

<Project name>

Prepared by

|  |  |  |  |
| --- | --- | --- | --- |
| Gabriele Bavaro | 27399103 | gabriele.bavaro@bell.net | |
|  |  |  | |
| Instructor: | | | Dr. C. Constantinides |
| Course: | | | SOEN 343 |
| Date: | | | November 23, 2016 |

**Document history**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| October 1, 2016 | 1.0 | Functionalities and Risks added | Gabriele Bavaro |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Table of contents**

1.      Introduction. 5

Purpose. 5

Scope. 5

Definitions, acronyms, and abbreviations. 5

References. 5

2.      Overall description. 6

Product perspective. 6

Product functions. 6

User characteristics. 6

Constraints. 6

Assumptions and dependencies. 6

3.      Specific requirements. 7

External interfaces. 7

Functionality. 7

Actor goal list. 7

Use case view.. 8

Reliability. 8

Usability. 8

Efficiency. 8

Maintainability. 9

Portability. 9

Design constraints. 9

(On-line) user documentation and help. 9

Purchased components. 9

Licensing requirements. 9

Legal, copyright and other notices. 9

3.      Analysis Models. 10

**List of figures**

Figure 1. Use case model. 7

# 1.      Introduction

The introduction of the Software Requirements Specifications Document provides an overview of the entire document.

## Purpose

This section defines the role or purpose of the Software Requirements Specifications Document and briefly describes the structure of the document. Identify the intended audience for the document is identified, with an indication of how they are expected to use the document.

## Scope

A brief description of what the Software Requirements Specifications Document applies to; what is affected or influenced by this document.

## Definitions, acronyms, and abbreviations

Provides the definitions of all terms, acronyms, and abbreviations required to properly interpret the Software Requirements Specifications Document.  This information may be provided by reference to the project’s Glossary.

## References

Provide a list of all documents referenced in the SRS.

# 2.      Overall description

This section describes a background to the requirements: The general factors that affect the product, such as constraints, assumptions and dependencies.

## Product perspective

Is the product self-contained? If not, then put the product into perspective with other related products. Use a block diagram to show the big picture.

## Product functions

Provide a summary of the major system functions.

## User characteristics

Who are the intended users of the system what is their expected educational level, experience and technical expertise?

## Constraints

Describe any items that will limit the options of the developers (such as regulations, hardware limitations, safety and security etc.)

There are a few potential risks and constraints associated with the system. There could be compatibility risks between different software, such as a user who utilizes an outdated computer system that can’t handle the software and subsequently fails to initialize it. In addition there could be communication issues between the different software components that make up the system such as the selected database not meshing well with the chosen Object Oriented Programming Language which could result in corrupted data.  Other risks include running out of time to implement critical or non-critical aspects of the system, team members leaving the project in the middle of the work period and writing the system code very poorly which can result in unforeseen consequences. In addition there could be use case risks in which the use cases themselves are poorly done or understood as well as the possibility of there being no scalable code in the end.

## Assumptions and dependencies

What assumptions are there? For example, a specific operating system should be present on a given hardware platform. If not, this document would have to be changed.

# 3.      Specific requirements

This section contains all requirements in detail: Functional as well as non-functional requirements (quality attributes and constraints). The quality attributes are listed according to the *ISO/IEC 25010* standard that classifies software quality in a structured set of characteristics and sub-characteristics.

**3.1  Functional Requirements**

The following are the functional requirements of the system which describe what the system will need to do in order to meet the demands of the stakeholder:

·         The system shall maintain a directory of rooms and their availabilities at different timeslot.

. A user shall be able to login and logout of the system.

. A user shall be able to see room availabilities.

. A user shall be able to reserve up to 3 timeslots per week.

. A user shall be able to able to put themselves on a waiting list if a room at a specific timeslot is not available.

. A user shall be able to cancel or replace their reservation by another one.

·         Only one user at a time on the system can have access to a specific room when they are reserving it or are updating or cancelling their reservation.

·         The system shall notify a user on the waiting list that was moved to a confirmed reservation.

·         A waitlist algorithm must be implemented so that users can wait to see if a room is available and then the room given to the next user on the list should the original user un-reserve a room.

·         The system needs to have an algorithm in place to help resolve the conflict of two users arriving at the exact same time to reserve a room.

**3.2  Non-Functional Requirements**

## Non-functional requirements define constraints on the way the software-to-be should satisfy its functional requirements or on the way it should be developed. Functional Suitability

To be completed once the coding is done, it says what was actually implemented vs what was supposed to be implemented.

This characteristic represents the degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions. This characteristic is composed of the following subcharacteristics:

* **Functional completeness.** Degree to which the set of functions covers all the specified tasks and user objectives.
* **Functional correctness.** Degree to which a product or system provides the correct results with the needed degree of precision.
* **Functional appropriateness.** Degree to which the functions facilitate the accomplishment of specified tasks and objectives.

## Performance efficiency

This characteristic represents the performance relative to the amount of resources used under stated conditions. This characteristic is composed of the following subcharacteristics:

* **Time behaviour.** Once
* **Resource utilization.** Degree to which the amounts and types of resources used by a product or system, when performing its functions, meet requirements.
* **Capacity.** Degree to which the maximum limits of a product or system parameter meet requirements.

The following are the non-functional requirements of the system:

·         The system must handle a large number of users at any given time.

· The system must be constructed using an Object Oriented Language (OO). Examples include Java, C++, C# etc.

·         The system must be safe and fair to use for every user.

·         The system must have be fair to every user.

·         The waiting list needs to be implemented as some sort of data structure such as an array or a queue.

·         A room can be booked multiple times up to a certain limit.

·         Each user needs to have a weekly allowance of time that they can use to reserve a room.

·         A room can be reserved for consecutive or non-consecutive time slots.

·         When reserving a room, users should have a certain view page that allows them to select which room they can reserve and at what times they can do so.

·         The wait list should have a limit for each room.

·         Users have a specific allowance of time that they can use to reserve rooms each week that will be renewed at the start of a new week.

·         The system could have extra features implemented such as a favourites list which contains a list of rooms that they prefer to reserve.

## External interfaces

A detailed description of all inputs into the system and all outputs from it (in terms of content and form).

## Functionality

Functional requirements capture the intended behaviour of the system. This section contains the *Actor Goal List* and the *Use Case view*.

## Actor goal list

|  |  |
| --- | --- |
| Actor | Goal |
|  |  |

## Use case view

The use case model is shown in Figure 1.

Figure 1. Use case model.

**Functional Suitability**

* **Functional completeness:** the system shall offer all functional requirements that are deemed critical (login, create and cancel reservations and add to waitlist) which represent 87% of the functional requirements and as many of the functional requirements mentioned in the above section as possible.
* **Functional correctness:** The system should present the correct information to the user in 90% of cases.
* **Functional appropriateness:** Each user activity (make, replace or cancel reservation) shall not take more than 3 steps for the user to accomplish.

**Performance efficiency**

* **Time behaviour:** The response to each user click or touch screen tap should take less than 2 seconds.
* **Resource utilization:** On the front-end, the user shall use an android mobile device with version 4.0 or 4.1 installed on it to be able to run the application. The system back-end will use a Wamp Server which consists of an Apache web server, MySQL database which will be connected to the Spring framework (Java language).
* **Capacity**: The system shall accommodate 100 concurrent users.

**Usability**

* **Appropriateness recognizability:** 80% of users should find that the system satisfies their needs.
* **Learnability:** It should take less than 30 minutes for a new users to figures out how to add, change and cancel reservations.
* **Operability:**  It should take less than 3 clicks (taps on touch screen) for the user to accomplish any of the main activities (add reservation, add to waitlist, change reservation or cancel reservation).
* **User error protection:** The system shall send reminder messages 1h before reservation time starts so student can cancel them if they can’t make it. CRUD operations shall require confirmation at the end for the changes performed to be saved in the database.
* **User interface aesthetics:** UI should implement many interface patterns found in similar applications to reduce confusion. A small sample survey shall be used to test it.
* **Accessibility:** User experience should feel familiar in its implementation and UI to 90% of users . Color blindness shall be taken into consideration when using colors to indicate results of operations. Front-end design shall follow Android best practices to allow augmentation of font size by user if necessary.

**Compatibility**

* **Co-existence.** The Android front-end mobile application shall co-exist with the Spring backend framework.
* **Interoperability.** The Android front-end mobile application shall exchange and communicate and receive information from the backend Spring framework through GET and POST HTTP requests.

**Reliability**

* **Availability**: System shall not make a Room instance available to other users if its WRITE status is set to ‘true’ as it is being reserved/canceled/updated by another user. A Room instance shall only be accessed by one user at a time for those operations.

**Security**

* **Confidentiality:** the system shall not disclose the identity of the room holders who have confirmed reservations to other users nor the identity of the people on the waiting list.
* **Integrity:** The system shall be safe and fair to every user.
* **Non-repudiation:** Database transactions shall be logged and saved.
* **Accountability:** Logs shall not be modifiable by Administrator.
* **Authenticity:** Long password (minimum 8 characters) shall be required from users.

**Maintainability**

* **Modularity:** A multi-layered system shall be designed to separate responsibilities and lower coupling. An object-oriented architectural style shall be used.
* **Reusability:** Multi-layer architecture shall allow main domain classes to be reused if need be as they do not directly communicate with low level layers.
* **Analysability:**  Logiscope shall be used to analyse the code. The report produced for this characteristic will include analysis of weighted methods per class , class comment rate, number of base classes and direct classes associated to each class. The resulting grade shall not be below fair.
* **Testability:** Logiscope shall be used to analyse the code. The report produced for this characteristic will include analysis of weighted methods per class, the total number of methods per class and the number of classes used directly by each current class. The resulting grade shall not be below fair.

**Portability**

* **Adaptability**: Android app shall work for devices with an Android OS version of 4.0 or 4.1.

## Design constraints

Decisions that must be followed, such as languages, processes, prescribed use of tools, architectural and design constraints, purchased components, class libraries, etc.

## (On-line) user documentation and help

Description.

## Purchased components

Description.

## Licensing requirements

Description.

## Legal, copyright and other notices

Description.

# 3.      Analysis Models

List all analysis models used in developing specific requirements previously given in this SRS.  Each model should include an introduction and a narrative description.  Furthermore, each model should be traceable the SRS’s requirements.

Illustrate (system) ***UML sequence diagrams*** (one for each critical scenario), identify system operations and describe operation contracts, one per critical system operation. You may also use ***UML state diagrams*** to describe critical use cases. Additionally, create a **domain model** for the system. Make sure that each model is traceable to the requirements.