**Software Requirements Specification**

Version X.X

for

GetARoom

Prepared by

|  |  |  |
| --- | --- | --- |
| Mihai Damsachin | 27177895 | mi\_damas@encs.concordia.ca |
| Jesus Imery | 27174276 | imeryjesus@gmail.com |
| Meetaz Alshbli | 25558840 |  |

|  |  |
| --- | --- |
| Instructor: | Dr. Constantinos Constantinides |
| Course: | SOEN 343: Software Architecture and Design I |
| Date: | <date of submission> |

**Document history**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 10/10/2016 | 0.1 | Part 1 | Meetaz Alshbli |
| 10/18/2016 |  | Part 2 | Samuel Dufresne |
| 11/7/2016 | ? | Added the Operations Contracts to the Analysis Models | Jesus Imery |
| 11/7/2016 | ? | Added state Diagram and Use Case Diagram to the Analysis Models | Samuel Dufresne |
| 11/6/2016 |  | Added external interfaces | Samuel Campbell |
| 11/7/2016 |  | Added functional requirements | Samuel Campbell |
| 11/7/2016 |  | Edited and reviewed Part #1 | Samuel Campbell |
| 11/7/2016 |  | Added System Sequence Diagram to the Analysis Models | Meetaz Alshbli |
|  |  | Modified operations contracts,  System Sequence Diagram, Domain Model | Meetaz Alshbli, Jesus Imery, Samuel Dufresne |
| 11/8/2016 |  | Added block diagram in part #2, Completed part #2 despite elections | Samuel Campbell |
| 11/9/2016 |  | Modified operation contracts and SSDs to eliminate superfluous contracts | Meetaz Alshbli, Jesus Imery |
| 11/14/2016 - 11/15/2016 |  | Worked on part 4  Lots of modification still to be done | Samuel Campbell |
| 11/14/2016 |  | Modifications and commentary in Chapter 1 and 2 | Mihai Damaschin |
| 11/19/2016 |  | Rearranged document, deleted non-critical use case specifications, added missing Make Reservation SSD | Meetaz Alshbli, Jesus Imery, Samuel Dufresne, Samuel Campbell |
| 11/19/2016 |  | Modified the use cases that include Reservation Sessions to reflect the discussed changes | Meetaz Alshbli, Jesus Imery, Samuel Dufresne, Samuel Campbell |
| 11/19/2016 |  | Review of SRS, modifications and commentary throughout documentation, precision to language and changes in description of functionality of the web application | Mihai Damaschin |

**Table of contents**

[1. Introduction](#_gjdgxs)

[Purpose](#_30j0zll)

[Scope](#_1fob9te)

[Definitions, acronyms, and abbreviations](#_3znysh7)

[References](#_2et92p0)

[2. Overall description](#_tyjcwt)

[Product perspective](#_3dy6vkm)

[Product functions](#_1t3h5sf)

[User characteristics](#_4d34og8)

[Constraints](#_2s8eyo1)

[Assumptions and dependencies](#_17dp8vu)

[3. Specific requirements](#_3rdcrjn)

[External interfaces](#_26in1rg)

[Functionality](#_lnxbz9)

[Actor goal list](#_35nkun2)

[Use case view](#_1ksv4uv)

[Reliability](#_z337ya)

[Usability](#_3j2qqm3)

[Efficiency](#_1y810tw)

[Maintainability](#_4i7ojhp)

[Portability](#_2xcytpi)

[Design constraints](#_1ci93xb)

[(On-line) user documentation and help](#_3whwml4)

[Purchased components](#_2bn6wsx)

[Licensing requirements](#_qsh70q)

[Legal, copyright and other notices](#_3as4poj)

[3. Analysis Models](#_1pxezwc)

**List of figures**

[Figure 1. Use case model.](#_2jxsxqh)

# Introduction:

This transcript is a Software Requirements Specification document which may enlighten stakeholders of Collegiate Institutions on the scope and purpose of the project. GetARoom is a system which enables users to make room reservations in a College Institution. The text provides the reader with graphical and textual representations of the GetARoom software through various critical and non-critical use cases. This document was prepared by following the IEEE conventions of Software Requirement Specification.

## 1.1 Purpose:

The contents of this SRS outlines the online conference room reservation system (GetARoom). Its purpose is to define all functionalities and expected behaviors represented in a way in which stakeholders may arrive to a common understanding. It does so by providing the reader with the system’s functionality, the design constraints, the external interfaces, performance, and attributes of the system to be developed. Furthermore, it shall be used by the project team who will design, develop and implement the code. This project team, shall consequently test all the features which define the behaviors of GetARoom. In sum, this document seeks for the development team and the end users to mutually agree on the expected features of this solution.

## 

## 1.2 Scope:

The GetARoom system is a web application targeted towards Collegial Facilities in order to reserve rooms to permitted individuals. In addition to allowing users to reserve a workspace, the user that created a specific reservation may also modify or cancel it.

A Collegiate institution may provide reservations to rooms through a web portal stored in a database. From this, any registered user may be able to view room availabilities as well as the reservations associated to them.

## 1.4 Definitions, acronyms, and abbreviations

|  |  |
| --- | --- |
| **Term** | **Definition** |
| User | A registered Individual who interacts with the web application |
| SRS | Software Requirement Specification |
| GetARoom | Name of the web application |
| AJAX | Asynchronous Javascript and XML |
| XML | Extensible Markup Language |
| HTTP | HyperText Transfer Protocol |
| MySQL | Relational database management system |
| UML | Unified Modeling Language |
| CO | Contract Operation |
| Stakeholder | Investors, employees and customers or any other individual that has an interest or can be affected by the business |
| e.g. | For example |

## References

[1] IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications (SRS), IEEE Computer Society.

# 2. Overall description

## This section will provide the reader with a global overview of the entire web based integrated system.

## 

## 2.1 Product perspective

This software product is intended for Collegial Educational Organizations where students may reserve rooms through a web server. The user may access the website from any browser indiscriminate of the operating system. The website will serve as the main user interface although it will only represent a fragment of the system.

## 

To utilize the product, a user is required to be registered within the system where their credentials are saved on the server ’s database (http://getaroom.lvain.com). A user may be able to login and logout at will. Due to the circumstances of a user’s personal information being stored on the server’s database, a user who logs in will find all their reservations previously requested under their user panel.

Furthermore, from the point of view of the user, they will have the choice to either add new reservations or modify and delete existing ones that are associated to their accounts.



## 2.2 Product functions

In this section, all critical product functions are mentioned and the following can be divided into three distinct categories which are the user management requirements, the room reservation requirements and the interface requirements.

2.2.1 User Management Requirements

This system requirement deals with user authentication where a user’s login information is stored in a database. Additionally, someone who successfully logs in will be assigned their previously registered reservations, will be able to view all registered reservations (put in place by the user and other users) of every room and will be able to create new reservations (which will either be registered or put on a waiting list depending on the availability of the targeted time slot).

2.2.2 Room Reservation Requirements

The room reservation requirements is one of the most important functionality expected of this software. It allows a user to add room reservations as well as to modify or cancel existing ones. Additionally, a user requesting a reservation which is already booked by someone else will get their request appended to a waiting list. Only one user may reserve a room at a time. Entering the “reservation mode” of a room on a specific day will block other users attempting to register a reservation for that room on that specific chosen day.

2.2.3 Interface requirements

The interface requirements is essential as it ties up every other requirements together and also connects the virtual application to the outside world. This specification will consist of a graphical representation of the system by which a user may use to interact with the software.

## 2.3 User characteristics

The system is meant to be used by college facilities to allow students to reserve rooms. The intended users are thus college student and professors. The experience and technical expertise required is comparable to managing an electronic calendar (e.g. Google Calendar).

## 2.4 Constraints

The system should effectively protect user identities from other users, e.g. when a user requests a room, his identity must remain unknown to other users who view the waiting list of a room. Only the current size of the waiting list should be given so the user may know his position on the waiting list before putting in place a reservation. Furthermore, the user may have access to their own reservations. The online system should provide an adequate interface for desktop users.

## 2.5 Assumptions and dependencies

A web browser that offers compatibility with HTML 5, CSS 3 and Javascript on a desktop. Recommended web browser are Google Chrome, Chromium, Firefox, and Safari. An internet connection is also necessary to interact with the system.

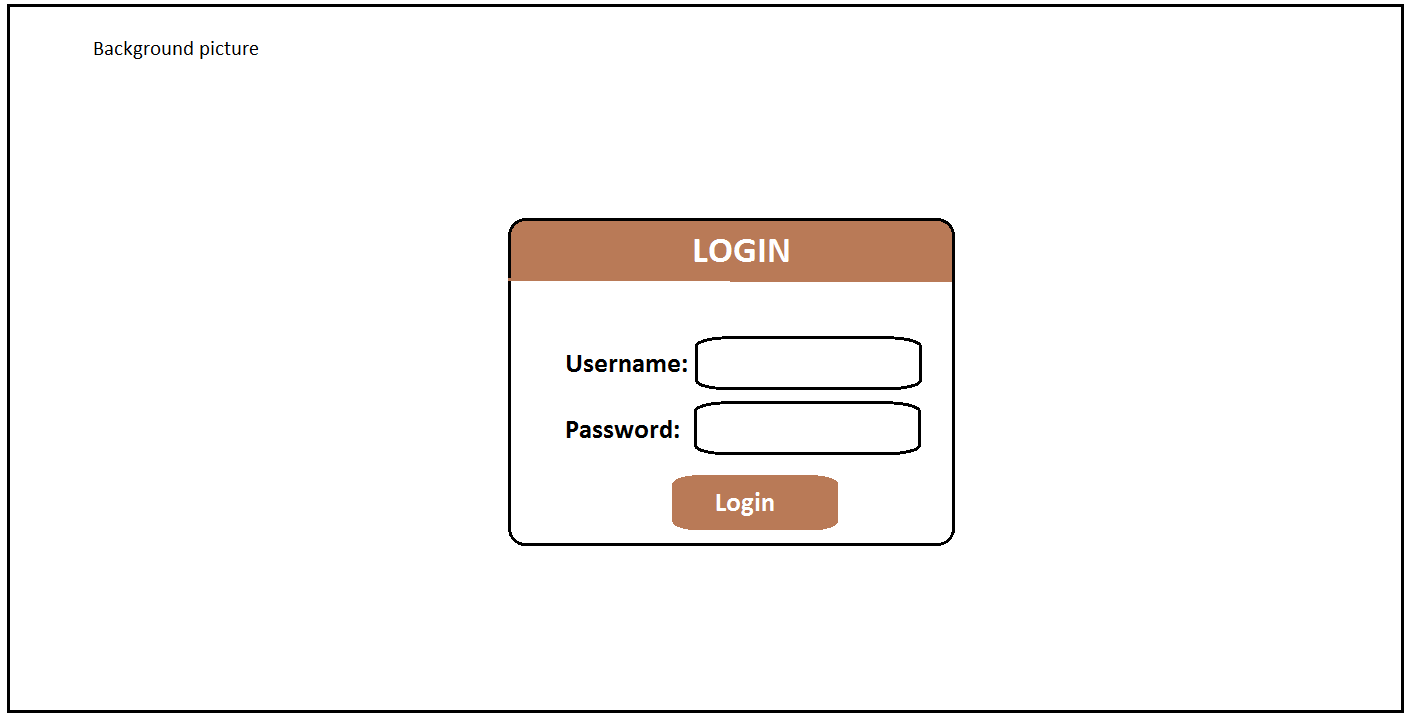
# Specific requirements

This section contains all functional and quality requirements of the system.

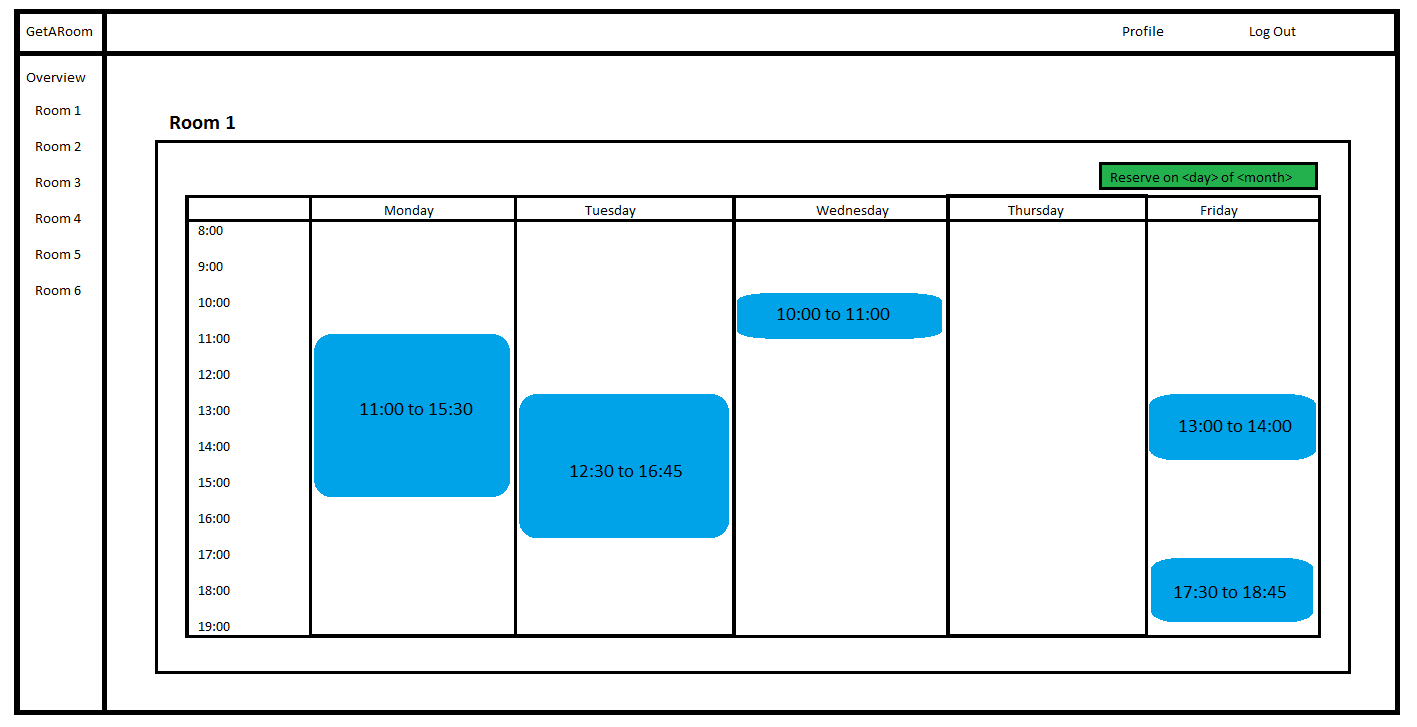
## 3.1 External interfaces

This section contains a series of mock-up user interfaces that the website will implement. The external interfaces allow the actor to communicate with the system.

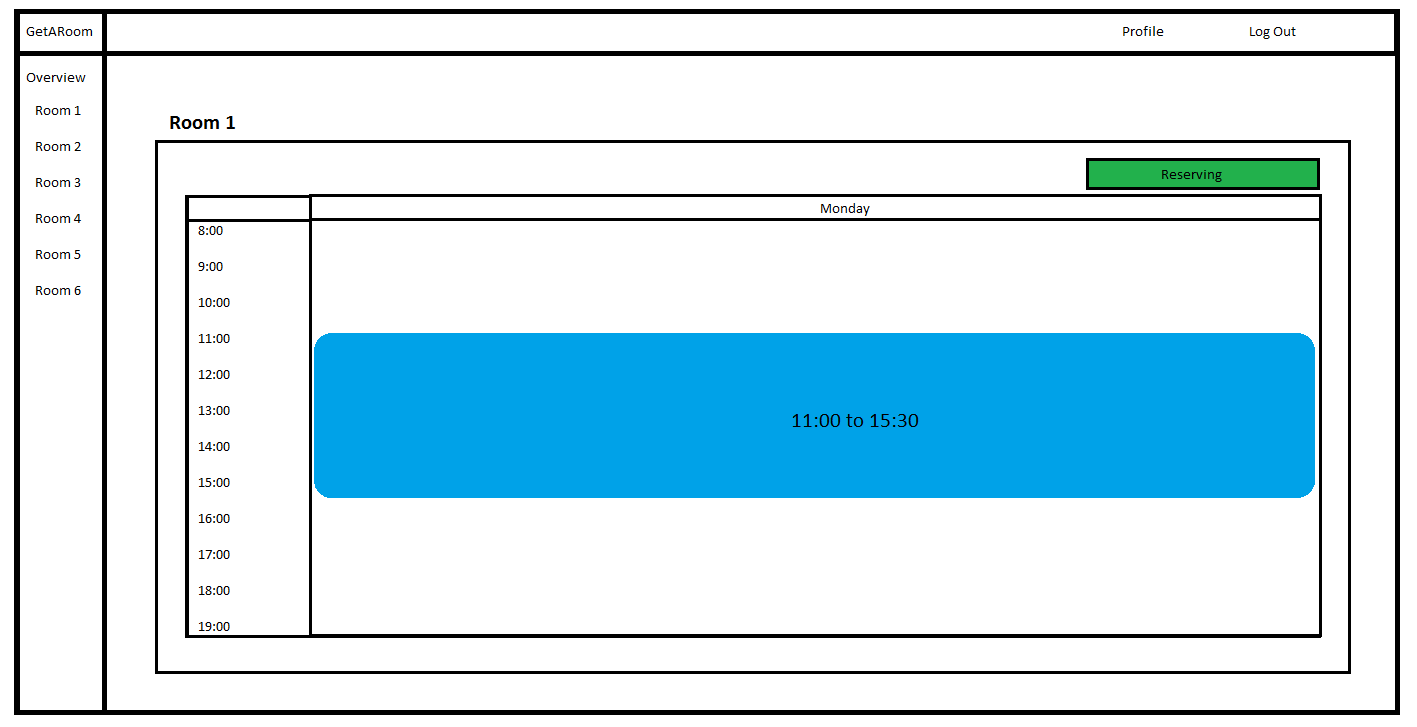
**3.1.1 Login page mock-up**



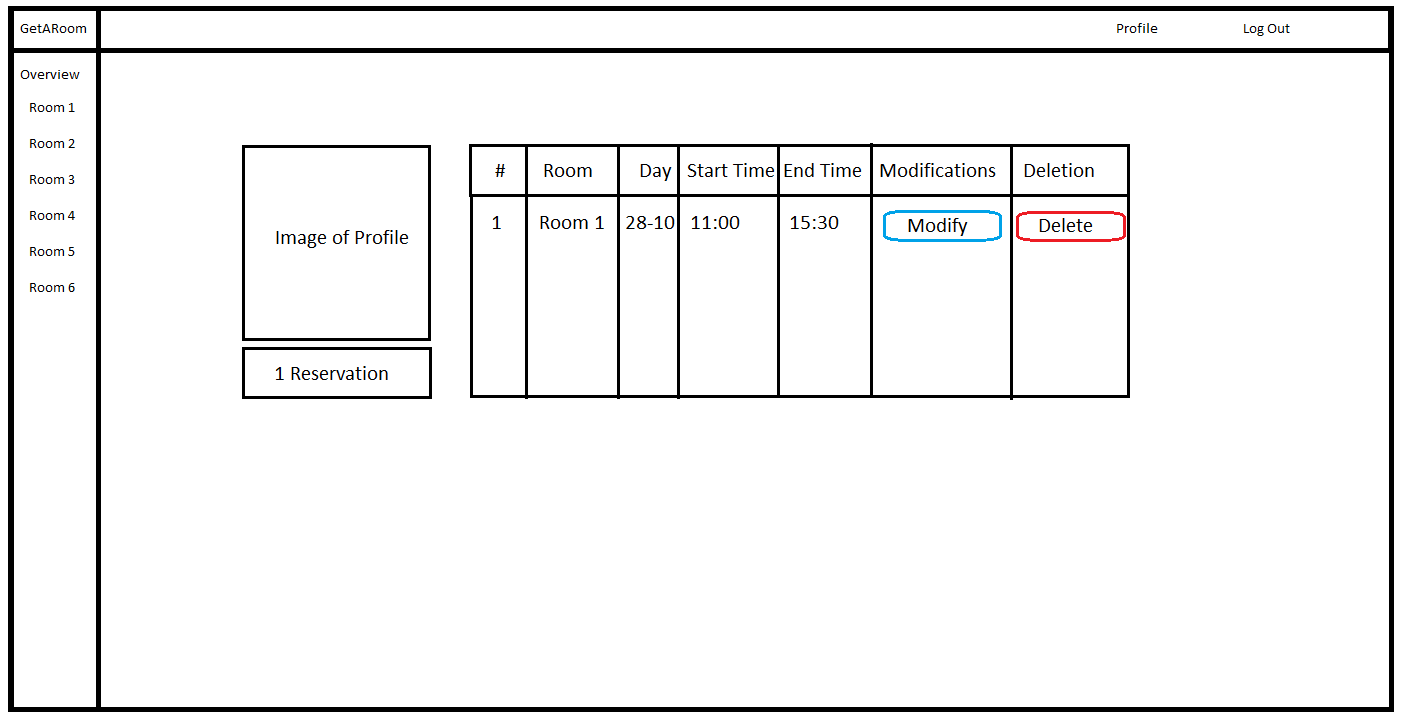
**3.1.2 Schedule view reservations**



**3.1.3 Registration view**



**3.1.4 User Panel View**



**3.1.1 Software Interfaces**

GetARoom shall communicate with the database in order to relay information to the AngularJS framework. The communication from the java application to the database consists of read, write and modify operations. Furthermore the java application shall transmit information to the framework through HTTP requests.

**3.1.2 User Interfaces**

screen mock-ups in the SRS

**3.1.3 Communication Interfaces**

The website may be accessed through the following link: http://www.getaroom.lvain.com

The network communications protocol shall be done through asynchronous Javascript and AJAX using HTTP between the client and the server.

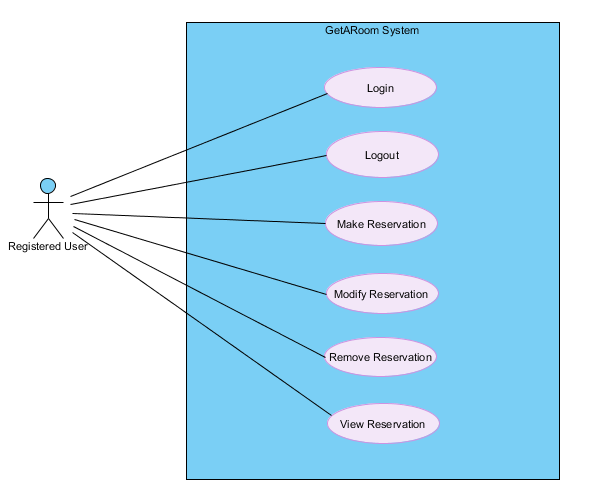
**3.1.4 Hardware Interfaces**

There are no direct hardware interfaces since the web portal doesn’t have any designated hardware. Access to the database from the client is done through the framework and communication between the backend and the database is undergone through the server’s operating system.

## 3.2 Functionality

3.2.1 Class User

## Use Case Model



3.2.1.1 **Log In**

|  |  |
| --- | --- |
| Use Case UC1 | Log In |
| Brief Description | A registered user may log into their account |
| Precondition | A user is already registered in which case all their credentials are stored in the database of the system |
| Triggering event | The user clicks on “login” |
| Main Flow | 1. A user accesses the login page 2. The user enters their username and password 3. The user clicks on “login” 4. The system authenticates the user for valid information 5. The user is assigned a token 6. The user is redirected to the main webpage |
| Post Conditions | The user may view, add, delete and modify their reservation(s) |

3.2.1.2 **Log Out**

|  |  |
| --- | --- |
| Use Case UC2 | Log Out |
| Brief Description | A registered user may log out of their account |
| Precondition | A user is logged into their account |
| Triggering event | A user clicks on “logout” |
| Main Flow | 1. The user presses “logout” 2. The user is redirected to the login page |
| Post Conditions | The user may log back into their account |

3.2.1.3 **Make Reservation**

|  |  |
| --- | --- |
| Use Case UC3 | Make Reservation |
| Brief Description | A User may create a new room reservation |
| Precondition | A User is logged into their account |
| Triggering event | A User clicks on “Make Reservation” |
| Main Flow | 1. The user selects a room to reserve 2. The User selects a time period for which he would like to have a room 3. The user clicks on the “Reserve” button 4. The system stores the reservation information in the database 5. The system returns a success message to the user 6. The user is redirected to the room page view |
| Alternate Flow | 1. The User selects a room to reserve 2. The user selects a time period for which he would like to have a room 3. The user clicks on the “Reserve” button 4. The system indicates to the user that the room is already booked for the time specified and asks the user if they would like to be put on a waiting list 5. The user clicks on “Add to Waiting List” 6. The System stores the reservation information in the database 7. The user is redirected to the room page view |
| Post Conditions | The User is either possess a registration for a room at a given time or is put on a waiting list |

3.2.1.4 **Remove Reservation**

|  |  |
| --- | --- |
| Use Case UC4 | Remove Reservation |
| Brief Description | A User cancels one of their room reservations at a given time |
| Precondition | 1. A user is logged into their account 2. A user is on the “User Panel” page 3. A user has at least 1 room reservation |
| Triggering event | A user clicks on “Confirm” |
| Main Flow | 1. The user pinpoints the reservation to be deleted in the user panel 2. The user clicks on “Remove Reservation” 3. The system displays a drop down menu showing “Confirm” and “Cancel” 4. The user clicks on “Confirm” 5. The system removes the reservation at a given time associated to the user in the room 6. The system retrieves the next reservation on the waiting list associated to a user who previously made that reservation in the room 7. The system displays to the user that their reservation has been removed. 8. The system redirects the user to their user panel |
| Post Conditions | The user’s reservation has been successfully removed |

3.2.1.5 View Reservation

|  |  |
| --- | --- |
| Use Case UC5 | View Reservation |
| Brief Description | A User may view a reservation |
| Precondition | A user is logged into their account |
| Triggering event | A User clicks on “View Room Availability” |
| Main Flow | 1. A user selects a room 2. The user clicks on “View Room Availability” 3. The system displays a calendar view of the room with their reservations |
| Post Conditions | The User is able to view room reservations |

3.2.1.6 Modify Reservation

|  |  |
| --- | --- |
| Use Case UC6 | Modify Reservation |
| Brief Description | A user modifies the time of their room reservation |
| Precondition | 1. The user is logged in 2. The user has a room reservation 3. The user is accessing the user panel |
| Triggering event | The user clicks on “Modify” |
| Main Flow | 1. The user clicks on the “Modify” button associated with the reservation targeted 2. The system asks the user for which time he would like to change their reservation 3. The user inputs a time slot and clicks “Accept” 4. The system returns a success message to the user 5. The system displays further possible actions for the user to take |
| Alternate Flow | 1. The user clicks on their room reservation 2. The system ask the user which action they would like to perform 3. The user clicks on “Modify Reservation” 4. The system asks the user for which time he would like to change their reservation 5. The user inputs a time slot and clicks “Accept” 6. The System informs the user that they will be put on a waitlist and asks the user if they would consent 7. The USer clicks on “Accept” 8. The system stores the reservation information in the database and associates it to the user which is placed in waiting list 9. The system displays further possible actions for the user to take |
| Post Conditions | The user has their reservation modified or is placed on the waiting list for the time chose of a room |

## Actor goal list

|  |  |
| --- | --- |
| Actor | Goal |
| Registered User | Make Room Reservations |
| Modify Own Room Reservations |
| Cancel Own Room Reservations |

## 3.3 Reliability

The database of the system shall operate on a RAID 10 configured server indicating that all information is backed up.

Access to this database shall be as reliable as the internet service provider of the host.

## 3.4 Usability

The system shall implement a calendar graphical interface to display room reservations to the user.

The system shall allow a user to highlight a selected time slot in order to book their .

The system shall conform to the three-click rule in order to navigate the website. This three-click rule means that the user may navigate to any webpage content in three mouse clicks or less.

## 

## 3.5 Efficiency

The system shall operate on a RAID 10 server where the memory and runtime is split between two hard drives.

## 

## 3.6 Maintainability

The system shall create a backup of the database. This is a service provided due to the use of our RAID 10 server.

The system shall keep a log file of all errors that occur with the timestamp at which they happen.

## 3.7 Portability

The system is implemented in Java and shall be compatible with any operating system such as (Windows, Linux, Mac, etc.) for as long as the user has access to a modern desktop web browser.

## 3.8 Design constraints

No Specific design constraints.

## 3.9 (On-line) user documentation and help

## 3.10 Purchased components

The server costs 10$/month for a server with 2 Terabytes bandwidth with 24GB of storage.

## 3.11 Licensing requirements

Not Applicable.

## 3.12 Legal, copyright and other notices

Not Applicable.

# 4. Analysis Models

In this section, all critical use cases will be briefly described textually by a narrative passage as well as graphically through the representation of state diagrams, system sequence diagrams and operation contracts.

## 4.0.1 State Diagrams

State diagrams are models used to describe different states in which a system may find itself. Those states are surrounded by guard conditions which must be met in order to transition from one state to another. Hence, these diagrams depict which system operations must be executed in order to meet these transitions. Furthermore, the state of a system differs from another by their specific attributes and roles they play in the sequence of operations.

## 4.0.2 System Sequence Diagrams

System sequence diagrams represent how a use case processes systematically following steps by showing the transmission of messages between the actor and the system. All elements within the diagram are formed of system operations accompanied with the appropriate return value. The return line is optional if nothing is to be returned. Additionally, arrows regulate the directions of the messages being sent to the entities with their corresponding system operation right above the pointer.

## 4.0.3 System Operations

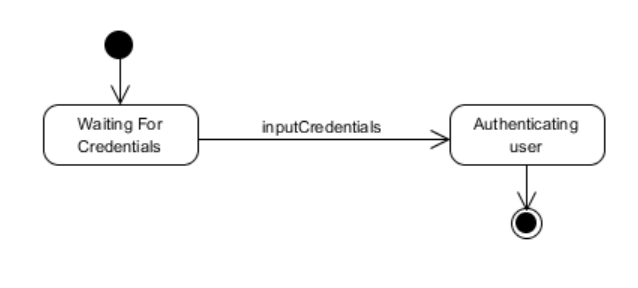
The system operations required to be written in a system sequence diagram encompasses the bulk of all the system events. The system operations are to be enumerated by the concrete method they represent.

## 4.0.4 Operation Contracts

Operation contracts breakdown system operations described from an applicative point of view. It achieves so by showing forth an operation’s name, the system operation they represent, the cross-referenced use case and a set of preconditions and postconditions.

## 4.1 Login

### 4.1.1 Use Case State diagram



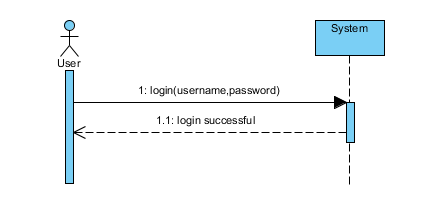
### 

### 4.1.2 System Sequence Diagram

1- A user visits the login page.

2- A user enters his username and password

3- The system authenticates the user and redirects them to the main page



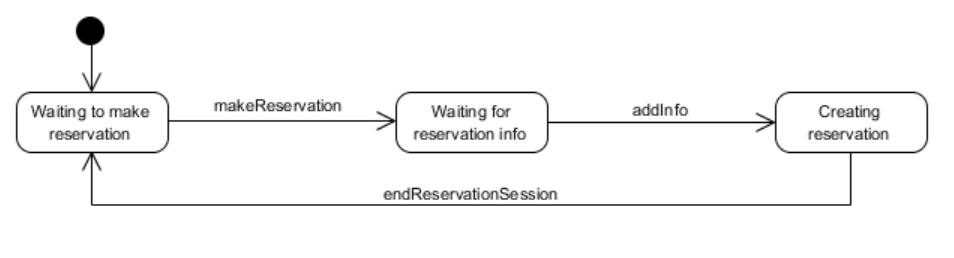
### 4.1.3 Operation Contract

|  |  |
| --- | --- |
| Contract CO1: | Login |
| Operation: | Login(username, password) |
| Cross Reference: | Use case User Management |
| Precondition: | - User account (username, password ) must exist in the database  - User must enter valid corresponding credentials (username, password)  - User must be logged out |
| Postcondition: | - An instance of User ***u*** was created. (instance creation)  - Instance ***u*** was associated with the terminal via UserCatalog. (formation of association)  - Instance ***u*** was associated with the ReservationSession. (formation of association) |

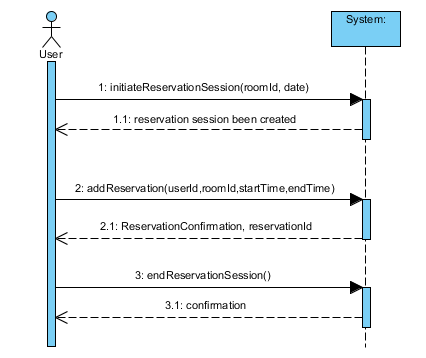
## 4.2 Make Reservation

### 4.2.1 Use Case System State Diagram

During the makeReservation use case, it is illegal to addInfo before executing the makeReservation operation. A user is asked to enter/modify their room reservation information until completion. Only then will the reservation session end.



### 4.2.2 System Sequence Diagram



### 4.2.3 System Operations

This use case consists of three system operations which allow a user to make a room reservation or to end their room reservation request session.

|  |
| --- |
| **System Operations** |
| initiateReservationSession(roomId, date)  addReservation(userId, roomId, startTime, endTime)  endReservationSession() |

### 

### 4.2.4 Operation Contracts

### 4.2.4.1 Initiate Reservation Session

|  |  |
| --- | --- |
| Contract CO3: | Initiate Reservation Session |
| Operation: | initiateReservationSession(roomId, date) |
| Cross Reference: | Use case Make Reservation |
| Precondition: | * User has been authenticated by the system |
| Postcondition: | * An instance of ReservationSession ***rs*** was created (instance creation) * ***rs*** was associated with the Terminal (association was formed) * ***rs*** was associated with User (association was formed) |

#### 4.2.4.2 Add Reservation

|  |  |
| --- | --- |
| Contract CO4: | Add Reservation |
| Operation: | addReservation(userId, roomId, startTime, endTime) |
| Cross Reference: | Use case Make Reservation |
| Precondition: | * ReservationSession is under way. |
| Postcondition: | * An instance of Reservation ***r*** was created (instance creation) * An instance ***r*** was associated to ReservationSession (association was formed) * Instance ***r*** was associated with the ReservationList (formation of an association) |

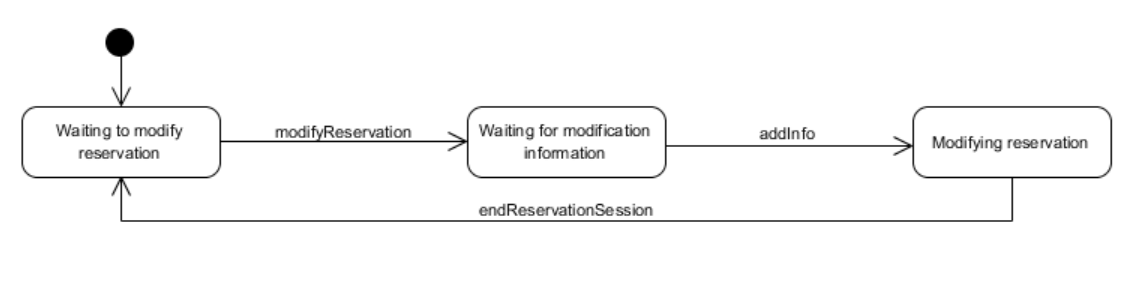
#### 4.2.4.3 End Reservation Session

|  |  |
| --- | --- |
| Contract CO5: | End Reservation Session |
| Operation: | endReservationSession() |
| Cross Reference: | Use case Make Reservation |
| Precondition: | * A ReservationSession is underway |
| Postcondition: | * ReservationSession.isComplete is set to true (attribute modification) |

## 4.3 Modify Reservation

### 4.3.1 System State Diagram

During the modify reservation use case, the system shall not enter the “Waiting for Reservation Selection” state before the modifyReservation operation has occurred. From this state a user may select a reservation through reservationSelected or end their reservation modification session with modifcationSessionEnded. Furthermore, a user shall only be permitted to modify their reservation with the infoAdded method once the system is in “Waiting for Modification Information” state which follows the reservationSelected method invocation.



### 4.3.2 System Sequence Diagram

### modifyReservation.PNG

### 4.3.3 System Operations

\*Cannot make proper desc yet because errors\*

|  |
| --- |
| **System Operations** |
| initiateReservationSession(roomId, date)  modifyReservation(reservationId, userId, startTime,endTime)  endReservationSession() |

### 

### 4.3.4 Operation Contracts

#### 4.3.4.1 Initiate Reservation Session

|  |  |
| --- | --- |
| Contract CO3: | Initiate Reservation Session |
| Operation: | initiateReservationSession(roomId, date) |
| Cross Reference: | Use case Modify Reservation |
| Precondition: | * User has been authenticated by the system * User has an active reservation in the system |
| Postcondition: | * An instance of ReservationSession ***rs*** was created (instance creation) * ***rs*** was associated with the Terminal (association was formed) * ***rs*** was associated with User (association was formed) |

#### 4.3.4.2 Modify Reservation

|  |  |
| --- | --- |
| Contract CO6: | Modify Reservation |
| Operation: | modifyReservation(reservationId, roomId, startTime, endTime) |
| Cross Reference: | Use case Modify Reservation |
| Precondition: | * ReservationSession is underway. |
| Postcondition: | * Reservation.roomId was modified (attribute modification) * Reservation.startTime was modified (attribute modification) * Reservation.endTime was modified (attribute modification) |

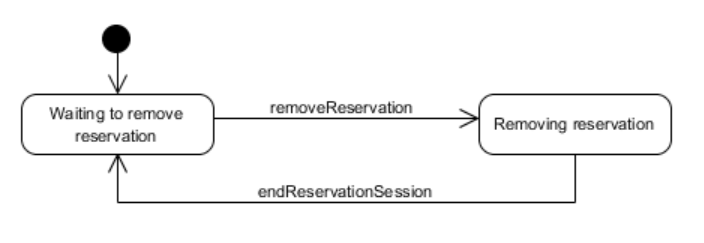
#### 4.3.4.3 End Reservation Session

|  |  |
| --- | --- |
| Contract CO5: | End Reservation Session |
| Operation: | endReservationSession() |
| Cross Reference: | Use case Modify Reservation |
| Precondition: | * ReservationSession is underway |
| Postcondition: | * ReservationSession.isComplete is set to true (attribute modification) |

## 4.4 Remove Reservation

### 4.4.1 System State Diagram

This use case shall not permit a user to select a reservation if the state of the system isn’t in “Waiting for Reservation Selection”. The system shall only enter this state when the removeReservation method has been invoked. Furthermore as long as the state corresponds to “Waiting for Reservation Selection”, a user may either select reservations with the reservationSelected method or end the removal session through the removeSessionEnded method. The former invocation sets the system back to it’s default Idle state.



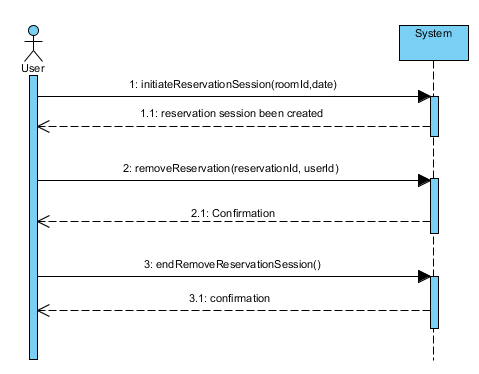
### 4.4.2 System Sequence diagram

1- Users starts a reservation removal session

2- The system returns a list of all reserved rooms associated to the user

3- The user selects a reservation and removes it

4- The system confirms reservation removal. Repeat steps 3-4 until user indicates done



### 4.4.3 System Operations

\*need so much fixing before desc…\*

|  |
| --- |
| System Operations |
| initiateReservationSession(roomId, date)  removeReservation(reservationId, userId)  endReservationSession() |

### 4.4.4 Operation Contracts

#### 4.4.4.1 Initiate Reservation Session

|  |  |
| --- | --- |
| Contract CO3: | Initiate Reservation Session |
| Operation: | initiateReservationSession(roomId, date) |
| Cross Reference: | Use case Remove Reservation |
| Precondition: | * User has been authenticated by the system * User has an active reservation in the system |
| Postcondition: | * An instance of ReservationSession ***rs*** was created (instance creation) * ***rs*** was associated with the Terminal (association was formed) * ***rs*** was associated with User (association was formed) |

4.4.4.2 Remove Reservation

|  |  |
| --- | --- |
| Contract CO7: | Remove Reservation |
| Operation: | removeReservation(reservationId, userId) |
| Cross Reference: | Use case Remove Reservation |
| Precondition: | * User must have an active reservation that is available for modification * initiateReservationSession is underway |
| Postcondition: | - The instance r of reservation is removed for the system (instance deletion)  - ***r*** is disassociated from ReservationList (broken association)  - User.activeReservation has been modified (attribute modification) |

#### 4.4.4.3 End Reservation Session

|  |  |
| --- | --- |
| Contract CO5: | End Reservation Session |
| Operation: | endReservationSession() |
| Cross Reference: | Use case Modify Reservation |
| Precondition: | * ReservationSession is underway |
| Postcondition: | * ReservationSession.isComplete is set to true (attribute modification) |

# 

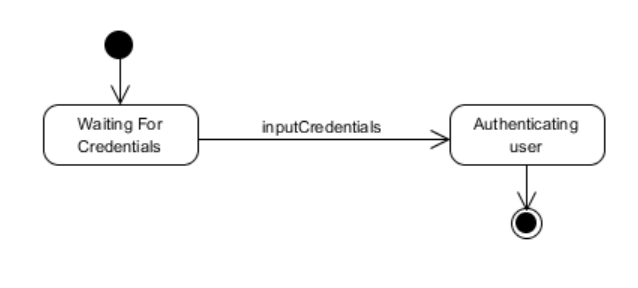
# 

# 

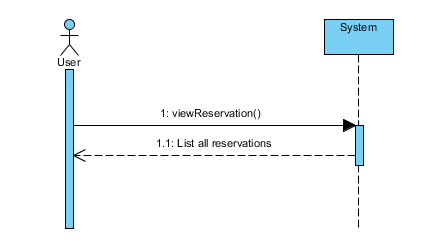
# 

# 4.6 View User Reservations

## 4.6.1 State Diagram



## 4.6.2 System Sequence Diagram



## 4.6.3 Operation Contracts

4.6.3.1 View User Reservations

|  |  |
| --- | --- |
| Contract CO10: | View User Reservations |
| Operation: | viewReservations() |
| Cross Reference: | Use case View Reservations |
| Precondition: | * User is logged in |
| Postcondition: | * None |

## 4.6 Domain Model:

