Software Requirements Specification

caproomster

Zhipeng Cai 21346482
Adrianna Diaz 27184778
Lance Lafontaine 26349188
Lenz Petion 26775837
Arek Manoukian 21710389
Taimoor Rana 26436110

Instructor: Constantinos Constantinides

April 4, 2017

Original Credits

Version 0.13

for

BookMe

Prepared by

Emir Bozer	26424724	emir.bozer@gmail.com
Nikolas De Vigne Blanchet	27189877	nikdvb@gmail.com
Ahmad Hyjaz Loudin	27179294	hyjaz.loudin@gmail.com
Mary Psaroudis	27209193	marypsaroudis@gmail.com
Leo Yu	27036736	yuleo@outlook.con

Instructor: Constantinos Constantinides

Course: SOEN343: Software Architecture and Design I

Date: 23/11/2016

Document history

Date	Version	Description	Author
15/10/2016	0.01	Added Intro, Description and DM	Leo Yu
10/11/2016	0.02	Finalization of document	Leo Yu
13/11/2016	0.03	Final Formatting	Leo Yu
18/03/2017	1.00	Reformatting	Adrianna Diaz
04/04/2017	1.01	Finalization of document	Zhipeng Cai Adrianna Diaz Lance Lafontaine Arek Manoukian Taimoor Rana

Table of contents

List of figures

1. Introduction

Purpose

The purpose of this document is to clearly specify the software requirement for the system designated as **caproomster,** which is a web application of an online conference room reservation system that will be used by Concordia University students. The Software Requirement Specification (SRS) Document is divided into three sections: Overall Description, Specific Requirements and Analysis Models. The Overall Description section provides contexts to the requirements of the online conference room reservation system and how different factors can affect the product.

The Specific Requirements section lists in detail all functional requirement through use cases and non-functional requirement through the ISO/IEC 25010 standard. The Analysis Models section lists all UML models that will help visualize the structure and the interaction of the system based on the requirements. This document is intended for both developers and end- users to use as it provides a clear understanding of all the requirements of the system. For the end-users, they can use the document to review if all the requirements are met. For the developers, they can use the document to start designing the software architecture of system that will be used for implementation.

Scope

The SRS Document describes the software system of the conference room reservation system that will be developed. It provides the functional and nonfunctional requirements to describe what the **caproomster** system is expected to do. Additionally, it provides some scenarios that will showcase how the system is supposed to interact with a user based on the requirements. Hence, this documentation is heavily influenced by the requirements of the software. If there is a change of requirements by the stakeholders, the document must be modified in order to reflect on the new changes.

Definitions, acronyms, and abbreviations

CO	Contract Operation
ISO	International Organization for Standardization.
IEC	International Electronic Commission.
SRS	Software Requirements Specifications.
UML	Unified Modeling Language.
PostgreSQL	An object-relational database system.
Python	An interpreted, object oriented, high-level programming language with dynamic semantics.
Flask	“[It] is a microframework for Python based on Werkzeug, Jinja2 and good intentions.” [1]
UC	Use case
SSD	System Sequence Diagram

References

Provide a list of all documents referenced in the SRS.

[1] Flask. (2016) [Online]. Available: http://flask.pocoo.org/

2. Overall description

Product perspective

caproomster is a web application that allows the reservation of conference rooms in college facilities for registered users. However, caproomster is not a self-contained product as it requires access to a PostgreSQL database in order to fetch information for the system to execute. Thus, caproomster is responsible for authenticating the user, managing reservations, displaying reservations and requesting information from the PostgreSQL database. The PostgreSQL database is responsible for storing and providing both user data and reservation information to caproomster. The overall process is shown on Figure 1.

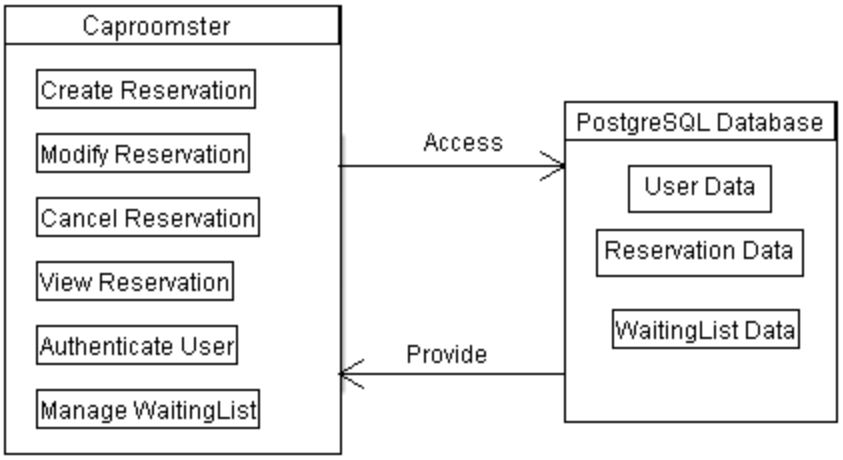

Figure 1. Block Diagram of Caproomster

Product functions

Caproomster contains five major system functions:

Creating Reservation

This major system function entails the reservation of a room through caproomster. If the user is authenticated and the selected room is not currently occupied in the system, the user can reserve a room by providing the time-range and a description of the reservation. caproomster verifies if the user inputs are valid and if the room is available at the specified time, the system creates the reservation and adds it to the ReservationBook.

Creating Repeat Reservations

The system allows the user to repeat their reservation for up to 3 weeks. The system will validate if the same room is available (at the same time) in the upcoming weeks before creating these reservations.

Viewing All Reservations

This major system function describes the viewing all the reservations made in caproomster. An authenticated user can request the system to view all reservations. In return, caproomster displays all the reservations, which includes their timeslots and room number, back to the user.

Modifying Reservation

This major system function describes the modification of a reservation in caproomster. If the user is the reservation holder and the selected room is not currently occupied in the system, the user can modify their own reservations. caproomster verifies if the user inputs are valid and if the room is available at the new specified time. If it satisfies the conditions, the system modifies the reservation and updates it to the registry. Afterward, the waiting list will get updated.

Cancelling Reservation

This major system function describes the cancellation of a reservation in caproomster. If the user is the reservation holder and the selected room is not currently occupied in the system, the user can

cancel his own reservations. In return, caproomster will delete the reservation and updates the ~~registry~~ reservation book. Afterward, the waiting list will get updated.

Read/Write Sessions: Safety, Liveness and Fairness

In the original SOEN 343 requirements, it is stated that *read* and *write* sessions for a determinate amount of time on a room's timeslot must exist in order to provide *safety, liveness and fairness* to our system. In addition, the SOEN 344 requirements asks us to elaborate on what happens on the server side when one opens a write session on his browser, then closes the browser before the session is completed or timed-out.

In both project submissions, for SOEN 343 and SOEN 343, an explicit server-side lock for room or timeslots was not provided. This was performed *purposefully*, as it allowed for a system that is still just as *safe, live, and fair* while being significantly more *concurrent*.

In this submission, our software system uses PostgreSQL, an open source Relational DataBase Management System (RDBMS) that is ACID-compliant. When a database is ACID-compliant, it is guaranteed to be Atomic (all-or-one), Consistent (ensures meaning state transitions), Isolated (executed sequentially) and Durable (persistent).

In addition, we are using Python 2 as a programming language with the Flask web framework. The use of these technologies ensures that our server is running *synchronously,* in a *single process*, with a *single thread*.

Therefore, the architecture of this system effectively ensures that all requests to our server are processed in a queue-like fashion and update the state of the application as is intended. Given this information, we are able to view the *all requests* as individual read and write sessions that happen to occur in a very short time. In addition, this solves the stated requirement that write sessions that are closed prematurely must be dealt with, given that all requests are processed in order, and cannot extend for a longer period of time than the length of the request. This solution effectively implies that our system is still *safe, live and fair*, while providing the added benefit of being significantly more *concurrent*.

User characteristics

The intended users of Caproomster are the students of Concordia University who have basic computer skills and knowledge to use the system.

Constraints

- ~~The inexperience with the programming language Python and the web framework Flask could dampen the development process of the project as time must be allocated to learn them.~~
- There is a time constraint for this project. Thus, there is a possibility not all requirements can be met on time.
- ~~The different programming environment could cause an issue later on as half the team is programming in a Mac environment and the other half is programming in a Windows environment.~~
- The original project BookMe was not tested in any way, and was written in Python, a weakly-typed language. As such, there is no guarantee that the expected functionality is present in the initial system. Thus a lot of time has to be invested just to make sure the original specifications are met on the base system, before the caproomster attempts to extend BookMe.

Assumptions and dependencies

- The caproomster web application is multi-platform, and its runtime environment takes place in a Mac, Linux or Windows operating systems.
- The caproomster web application shall be run on a local machine.
- The application shall run in an environment which includes the Python programming language version 2.7 binaries.
- Constant and reliable connection to the Internet is required in order to access the PostgreSQL database. Otherwise, a local PostgreSQL database must be used.

- ~~Users are assumed to be able to navigate and use the BookMe system perfectly.~~ User input is validated and the system is substantially more robust.
- There is an underlying assumption that all rooms in the system are equal in terms of ability to be reserved, as well as ability to accommodate reservable equipment.
- There is an underlying assumption that all reservable rooms and equipment are included and tracked within the system.
- ~~Assuming that the user would only need to modify the timeslot of their reservations.~~ Input is validated and permissions checked.

3. Specific requirements

External interfaces

An internal PostgreSQL database is used to store all the reservations and users. The database stores the usernames, user passwords as well as all the reservations that have been made. When a User logs in to the caproomster application, it connects to the external interface to receive the login information of the user. When a reservation is made, it sends information about the reservation to the database. When a user views their reservations in the caproomster application, it connects to the external database in order to receive, change and modify the reservation information.

- Hardware Interface
    - The Hardware Interface of the system is handled by Linux Centos 7 Operating System.
- Software Interface
    - The software is platform-agnostic.
    - The software uses PostgreSQL database to access its features. All the application features will be accessible through the frontend which is running on Node.JS

Functionality

Actor goal list

Actor	**Goal**
User	*1. Login* *2. Logout* *3. Reserve a room* *4. Cancel an existing reservation* *5. Modify a reservation* *6. View my reservations* *7. View schedule* *8. Add to waiting list* *9. Reserve room on a weekly occurrences*

Use case view

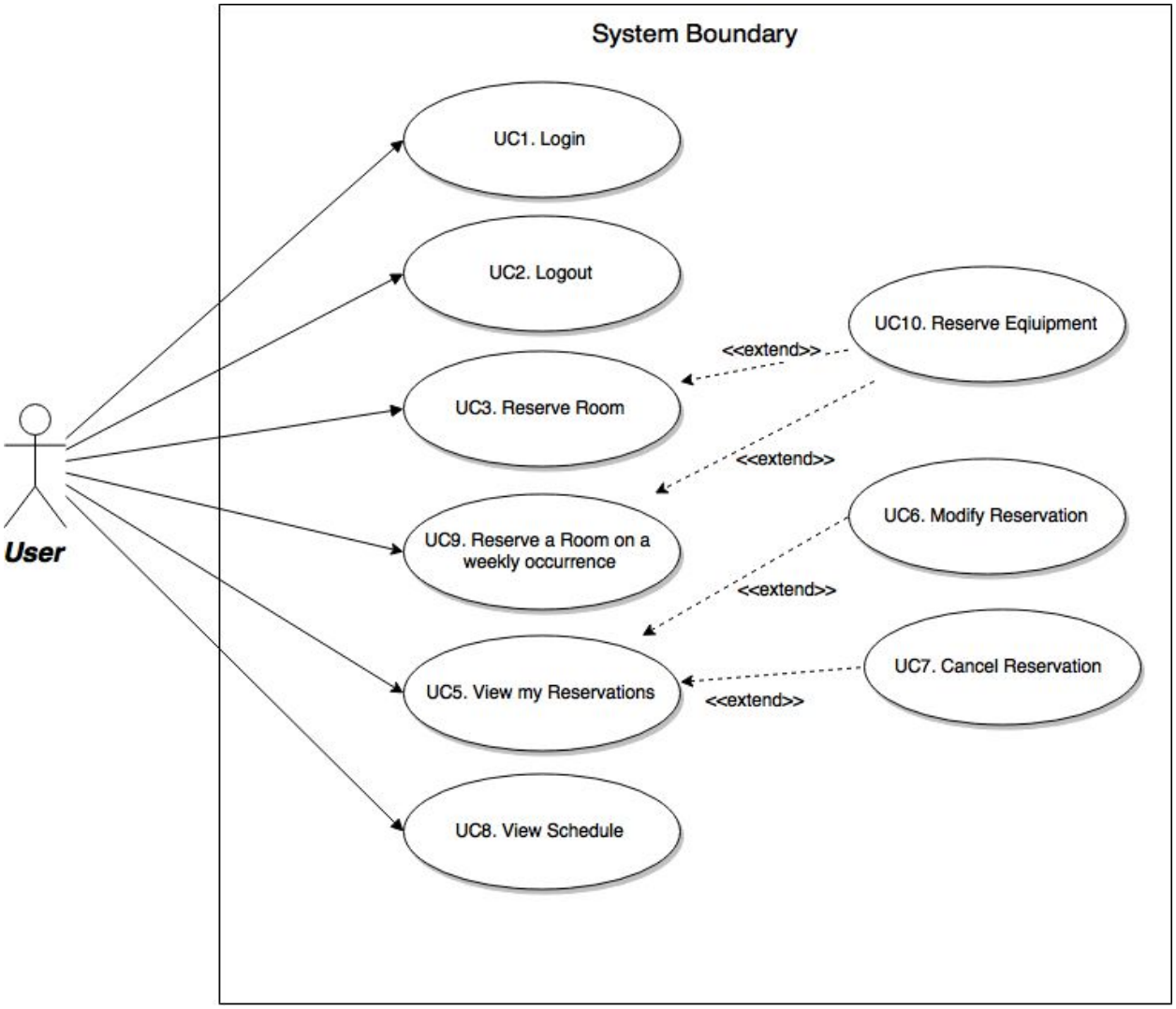

Figure 2. Use Case Model

Use Case UC1:	**Login**

Primary Actor	User
Stakeholders and Interests:	User logs into the system
Preconditions:	● The user is currently not logged in
Postconditions:	● The user is logged in ● The user has access to the functions of the system
Main success scenario:	1. The user provides a valid username and password 2. The system indicates that the user has successfully logged in.

Use Case UC2:	**Logout**
Primary Actor:	User
Stakeholders and Interests	User logs out of the system
Preconditions	● The user is logged in
Postconditions	● The user has been logged out
Main success scenario	1. The user requests to log out from the system 2. The system indicates that the user has successfully logged out.

Use Case UC3:	**Reserve Room**
Primary Actor:	User
Stakeholders and interests:	User reserves a room at specific date and time
Preconditions:	● User has been authenticated
Postconditions:	● The user has successfully reserved a room for a specific date and time
Main success scenario	1. The user selects the desired room to be reserved 2. The system confirms the vacancy of the room selected and lock it 3. The user inputs a time, a date, a description 4. The system confirms the reservation of the room to the user 5. The user indicates that they are done with the reservation of the room process 6. The system responds with a confirmation and unlock the room

	7. The user may repeat steps 1-6 until they are done with making reservation

~~**Use Case UC4:**~~	~~**Add to waiting list**~~
~~Primary Actor:~~	~~User~~
~~Stakeholders and interests:~~	~~User gets added to the appropriate waiting list~~
~~Preconditions:~~	~~• The user was unsuccessful reserving a room.~~
~~Postconditions:~~	~~• The user is added to the appropriate waiting list, depending on their capstone status~~
~~Main success scenario:~~	~~1. The user selects the desired room to be reserved~~ ~~2. The system confirms the vacancy of the room selected and lock it~~ ~~3. The user inputs a time, a date and a description to reserve the room~~ ~~4. The system responds with the unavailability of the room at the specified time and date~~ ~~5. The user requests to be added to the waiting list of that room with the same inputs when making the reservation~~ ~~6. The system confirms the user is queued on the waiting list~~ ~~7. The user indicates that they are done with being added to a waiting list process~~ ~~8. The system responds with a confirmation and unlock the room~~ ~~9. The user may repeat steps 1-8 until they are done~~

Use Case UC5:	**View my reservations**
Primary actor:	User
Stakeholders and Interests:	User views their reservations
Preconditions:	• User has been authenticated
Postconditions:	• The user is able to view all their reservations
Main success scenario	1. The user requests to view their reservations 2. The system displays all of the user's reservations

Use Case UC6:	**Modify Reservation**

Primary Actor:	User
Stakeholders and Interests:	User modifies an existing reservation
Preconditions:	● User has been authenticated ● User is viewing their reservations
Postconditions:	● User was able to successfully modify date and time
Main Success Scenario:	1. The user selects their reservation to be modified 2. The system confirms the vacancy of the selected room and lock it 3. The user modifies the time of the reservation 4. The system confirms the modification of the reservation to the user 5. The user indicates that they are done with the modification of the selected reservation 6. The system updates the waiting list and responds with a confirmation and unlock the room 7. The user may repeat steps 1-6 until they are done

Use Case UC7:	**Cancel Reservation**
Primary Actor:	User
Stakeholders and interests:	User cancels a specific reservation that they have made
Preconditions:	● A reservation has already been made.
Postconditions:	● The user has successfully cancelled the selected reservation
Main success scenario:	1. The user selects one of their reservation to be cancelled 2. The user requests the cancellation of the selected reservation 3. The system confirms the cancellation of the reservation to the user 4. The user indicates that they are done with the cancellation process 5. The system responds with a confirmation and unoccupied the room

Use Case UC8:	**View schedule**
Primary actor:	User
Stakeholders and Interests:	User views all the reservations in the system

Preconditions:	● User has been authenticated
Postconditions:	● The user is able to view the available and occupied time slots and rooms
Main success scenario	1. The user requests to view all reservations 2. The system displays all of the reservations

Use Case UC9:	**Reserve a Room on a weekly occurrence**
Primary Actor:	User
Stakeholders and interests:	User reserves a room at a specific time on a weekly occurrence
Preconditions:	● User has been authenticated
Postconditions:	● The user has successfully reserved a room for a specific date and time
Main success scenario	1. The user selects the desired room to be reserved 2. The user inputs a time, a date, a description and the repetitions (number of weekly occurrences) 3. The system confirms the multiple reservation of the room to the user

Use Case UC10:	**Reserve equipment**
Primary Actor:	User
Stakeholders and interests:	User reserves equipments for their reservation
Preconditions:	● User has been authenticated
Postconditions:	● The user has successfully reserved a room for a specific date and time and added equipments
Main success scenario	1. The user selects the desired room to be reserved 2. The user inputs a time, a date, a description and selects equipments needed 3. The system confirms the reservation of the room to the user

Reliability

The system will be available 24 hours a day, 7 days a week. User credentials will be stored and protected from potential hack threats and malicious software. Information input into the system will be saved except in the case of a power failure or unexpected crash.

Usability

Any user with basic computer skills will be able to use the software application to perform the basic operations outlined in the use cases. The user interface will be clean and easy to use. Documentation will be provided for users who require assistance.

Efficiency

The system should respond to user requests in no longer than 5 seconds. The system will be able to handle up to 50 users at a time.

Maintainability

The software applications language (Python) is universal, therefore in terms of reusability there will be no issues for using the code in different systems. Since the software application does not control any physical hardware, there will be no issues in terms of modularity. Also the software application can be easily modified without any defects because of the effectiveness and efficiency of the code.

Portability

The software applications language (Python) is portable on Windows OS, Linux OS, Windows Server 2012 and Max OSX.

Design constraints

In terms of programming languages, the project is required to be written in an object-oriented approach. Thus, Python is selected to be the main programming language of the project. For

databases, queries to the relational database postgreSQL can be done through the SQLAlchemy library provided by Flask itself. However based on the business needs for this project it is required to manually implement object-relational structural and behavioral patterns. The system must be designed with restrictions such as: A room can be booked on a repeated time-slot, but only for some time period and a user may create only up to some maximum allowable number of reservations per week. Also, the design of the system must provide safety, liveness and fairness.

Additionally, the implementation of a component of the system must be re-architected to include elements of the aspect-oriented programming paradigm. The chosen component must function as it did initially, while eliminating code duplication.

4. Analysis Models

Domain Model

The domain model is an abstract object model that showcases a static conceptual structure of the problem domain. Also known as the conceptual class diagram, the model illustrates concepts, associations between the concepts and attributes of concepts that are based on the functional requirement recorded in the use cases. Despite describing the overall project, the domain model doesn't describe the software design of the project. In order words, the diagram doesn't illustrate any system operations (methods or functions). In spite of that fact, all further analysis model can be traced back to the domain model, to which it can be traced back to the requirements. In the Caproomster system, the domain model is composed of several conceptual objects: `User`, `Registry`, `ReservationBook`, `Waiting`, `Reservation`, ~~`Directory`~~, Equipment, `Room` and `Timeslot`.

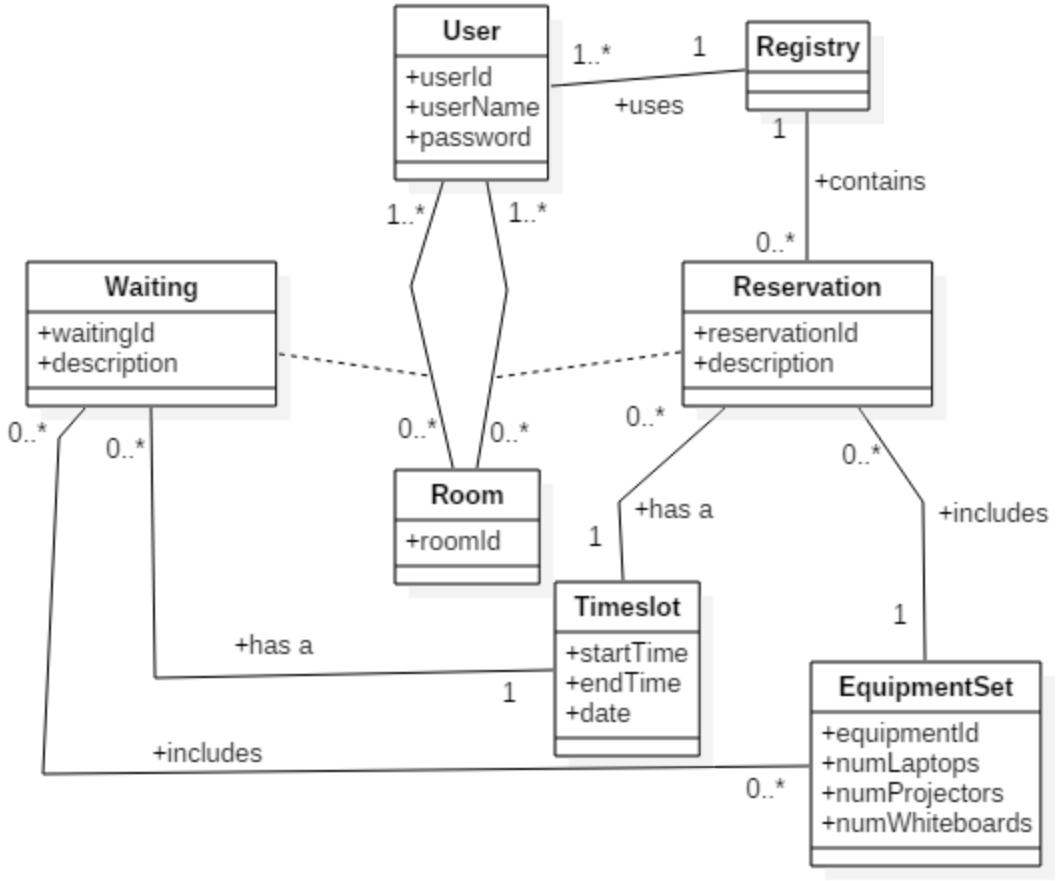

Figure 3. Caproomster Domain Model

User

The `User` is an actor of the system that represents a Concordia student. The `User` can use the `Registry` to start a reservation session. Also, the `User` can view their `Profile` to see their reservations and the waiting list. Each `User` is characterized by a name, an id number, an academic program, an email and a password.

Registry

The `Registry` is an object that represents the controller of the system where it receives requests from the `User` and assigns the appropriate responsibility to the system. The `Registry` holds the `ReservationBook` to start a reservation session and ~~utilizes~~ ~~the~~ ~~`Directory`~~ ~~to~~ ~~get~~ ~~relevant~~ ~~information~~ ~~about~~ ~~the~~ ~~rooms~~. ~~In~~ ~~addition,~~ ~~the~~ ~~`Registry`~~ ~~will~~ ~~update~~ ~~the~~ ~~`Profile`~~ ~~of~~ ~~the~~ ~~user~~ ~~with~~

new information.

ReservationBook

The `ReservationBook` is an object that represent a reservation book that keeps a list of reservations in the system. It contains multiple instances of both `Reservation` and the `Waiting` objects. Reservation

Reservation

The `Reservation` is an object that represents the reservation of a room at a specific timeslot. It is contained in the `ReservationBook` object. The `Reservation` object is characterized by a `Room` object, a description, a `Timeslot` object and the reservation holder's name.

Waiting

The `Waiting` is an object that represents a user being put on a waiting list when a room at a specific timeslot is unavailable for reservation. It is contained in the `ReservationBook` object. The `Waiting` is characterized by a `Room` object, a `Timeslot` object, a description and the reservation holder's name.

Equipment

The Equipment is an object that is referenced by a Reservation or a Waiting object to signify the equipment needed for a reservation. It contains a unique key, and the number of laptops, projectors and whiteboards.

~~Directory~~

~~The `Directory` is an object that represents the directory of rooms for the reservation system. It contains multiple instances of a `Room` object. Thus, it is used by the `Registry` object to get information about the Room object.~~

Room

The `Room` is an object that represents the room that are going to be reserved in the system. It is contained in the `Directory` object. The `Room` object is characterized by room id number ~~and its availability.~~

Timeslot

The `Timeslot` is an object that represents the time that the room is reserved for. It is contained in

the Room object. It is characterized by the start time, the end time the date of the reservation and the block which represents the amount of one hour interval during the reservation.

System Sequence Diagram

System Sequence Diagram is a subtype of the sequence diagram that visually describes a use case, events of a use case, in the order of the execution. Use cases describe the user's interaction with the system. System Sequence Diagram also shows how the tasks that are related to uses cases are completed. They are represented with the notation SSD. In an SSD, there are external actors, messages, system responses and any loops that are needed.

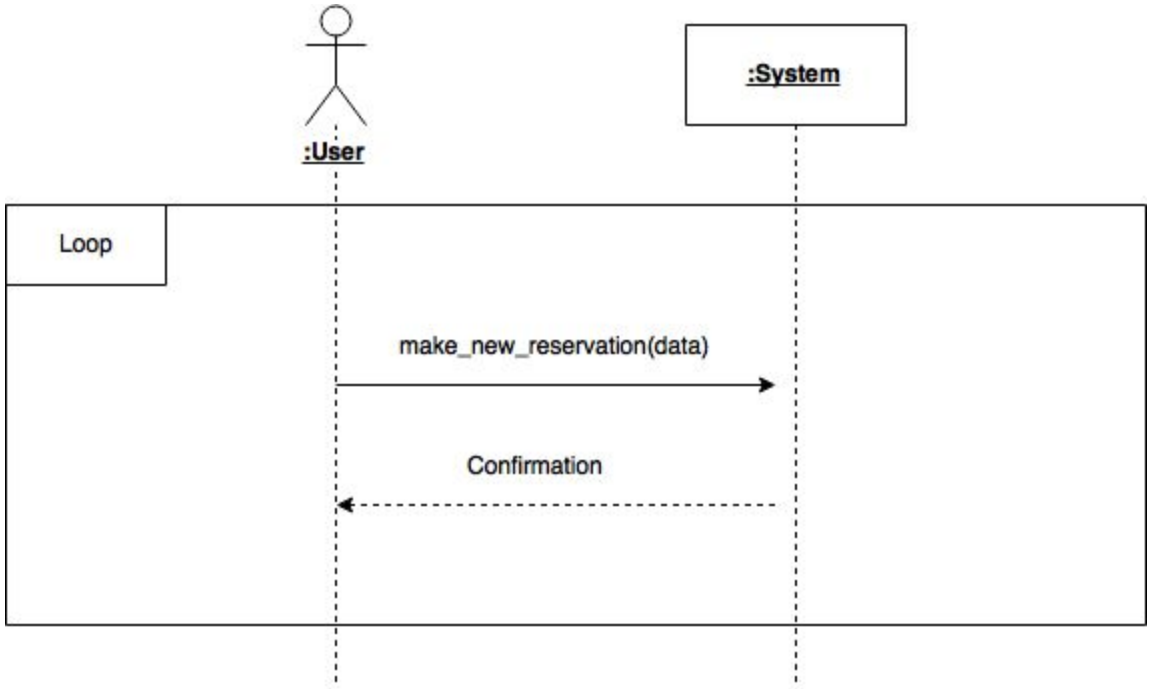

Figure 4. Reserve Room SSD

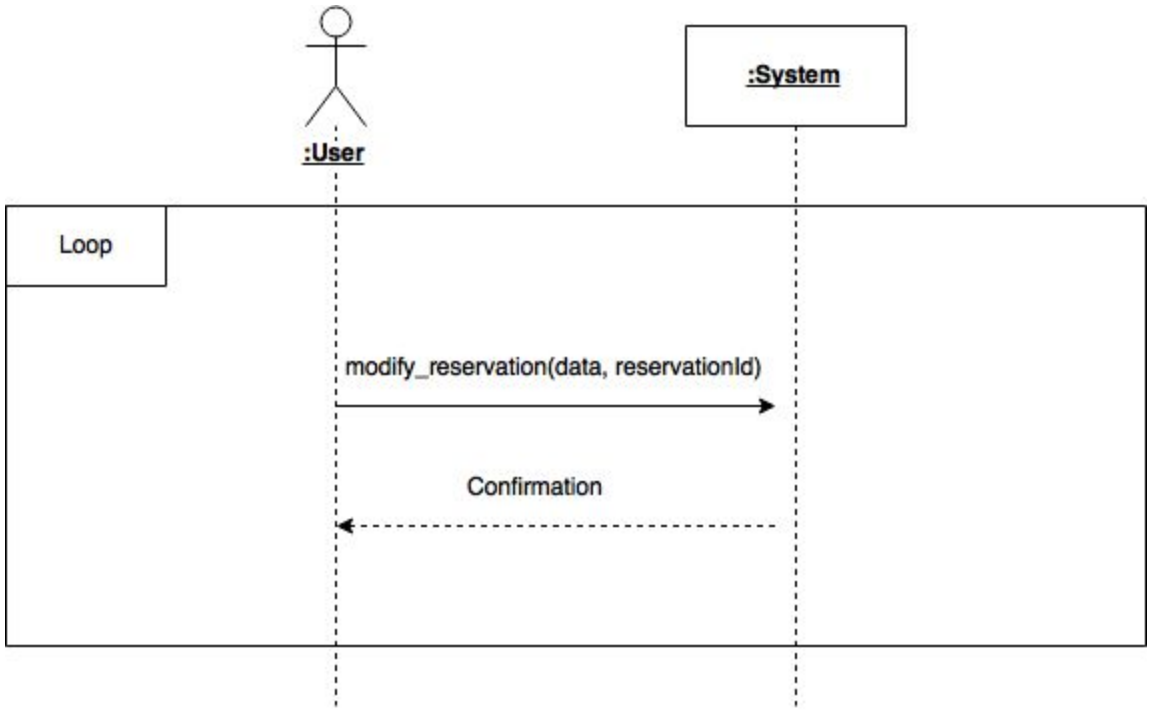

Figure 6. Modify Reservation SSD

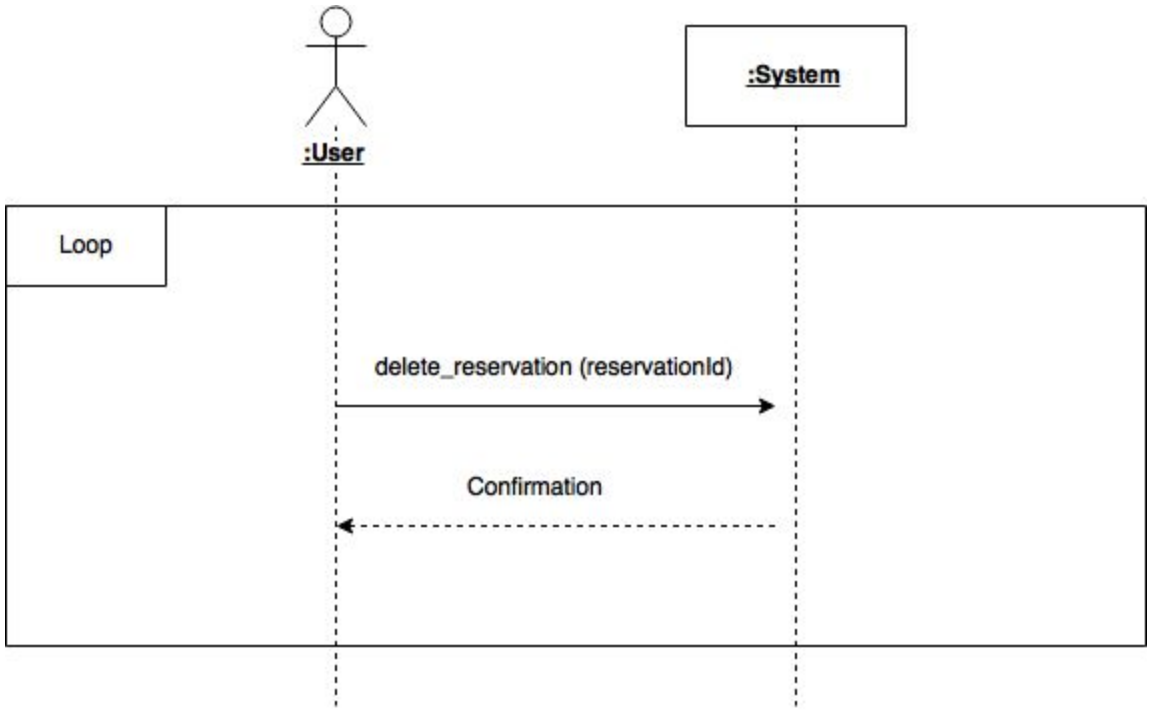

Figure 7. Cancel Reservation SSD

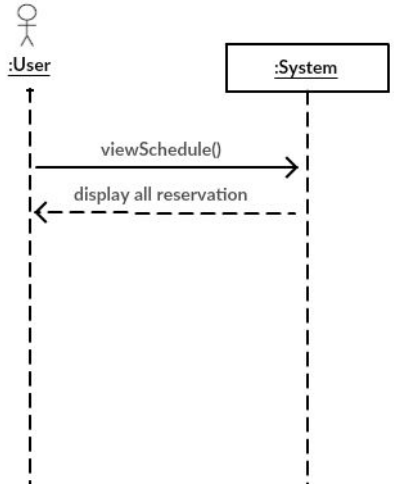

Figure 8. View Schedule SSD

System Operations

System Operations
~~initiateAction(roomId)~~ make_new_reservation(data) make_new_repeated_reservation(data,repeats) ~~addToWaitingList(roomId, holder, time, description)~~ commit_new_waiting(room,user,time,description,equipement,message) ~~endAction(roomId)~~ modify_reservation(data, reservationId) delete_reservation(reservationId) viewSchedule() updateWaiting(roomId)

Contract Operations

Contract CO1: Make New Reservation
Operation: make_new_reservation(data)
Cross References: Use Case Reserve Room
Preconditons:
- ~~A ReservationBook session is underway~~
- User is logged in

Postconditions:
- Reservation instance r was created
- Attributes of r were initialized
- r is added to ReservationBook

Contract CO2: View Schedule
Operation: viewSchedule()
Cross References: Use Case View Schedule
Preconditons:
- User is logged in.
- A ReservationBook session is underway.

Postconditions:
- A schedule is created from information contained in the ReservationBook

Contract CO3: Modify Reservation
Operation: modify_reservation(data, reservationId)
Cross References: Use Case Modify Reservation
Preconditons:
- User is viewing their reservations
- The user has reservations

Postconditions:
- Reservation entry attributes are updated

Contract CO4: Cancel Reservation
Operation: delete_reservation(reservationId)
Cross References: Use Case Modify Reservation
Preconditons:
- User is viewing their reservations
- The user has reservations

Postconditions:
- Reservation entry removed from ReservationBook

~~**Contract CO5**: Add to Waiting List~~
~~Operation: commit_new_waiting(room, user, time, description, equipement, message)~~
~~Cross References: Use Case Add to Waiting List~~
~~Preconditons:~~
- ~~The user has initiated a Reservation Session~~
- ~~User is logged in~~
- ~~The user has tried to make a new reservation, but the room is unavailable~~

~~Postconditions:~~
- ~~A WaitingList Waiting instance w was created and placed into the appropriate waiting list depending on the user's capstone status~~
- ~~Attributes of w were initialized~~
- ~~Instance w was added to user's myWaitingList~~

~~**Contract CO6**: Initiate Action~~
~~Operation: initiateAction(roomId)~~
~~Cross References: Use Case Reserve Room, Add to Waiting List, Modify Reservation, Cancel Reservation~~
~~Preconditons:~~

- The user has initiated a Reservation Session

Postconditions:
- The room's lock is set to true

Contract CO7: End Action
Operation: endAction(roomId)
Cross References: Use Case Reserve Room, Add to Waiting List, Modify Reservation, Cancel Reservation
Preconditions:
- The initiateAction(roomId) operation has been called

Postconditions:
- The room's lock is set to false

Contract CO8: Update Waiting

Operation: updateWaiting(roomId)

Cross References: Modify Reservation, Cancel Reservation

Preconditions:

- The modifyReservation(reservationId) or cancelReservation(reservationId) operation has been called

Postconditions:

- If the room is available at the same time as an instance of a Waiting item that is for that same room, an instance of Reservation r is created
- Reservation r's attributes are initialized with the w's attributes
- Reservation r is added to reservationBook and w is removed from the appropriate waiting list.

Contract CO9: Reserve a Room on a weekly occurrence
Operation: make_new_repeated_reservation(data, repeats)
Cross References: Use Case Reserve Room, Add to Waiting List, Modify Reservation, Cancel Reservation
Preconditions:
- A ReservationBook session is underway

Postconditions:
- For every week, if the room is available at the given time and day of the week, a reservation r is made.
- r is added to the ReservationBook

State Diagrams

State diagrams are used to depict the behavior of objects in response to a series of external events. Each event triggers an object to move from one state to another. A state is the representation of a condition of an object moving through its behavior pattern. At any given moment, an object can only be in one state at a time. State diagrams are often used to help understand the life cycle of an object based on its preceding states and are used to model the dynamic flow of an object within a system. State diagrams can be applied to UML components such as classes and use cases. Moreover, state diagrams are effective in specifying the legal sequence of events. The following is a state diagrams for the most critical use cases in **caproomster**.

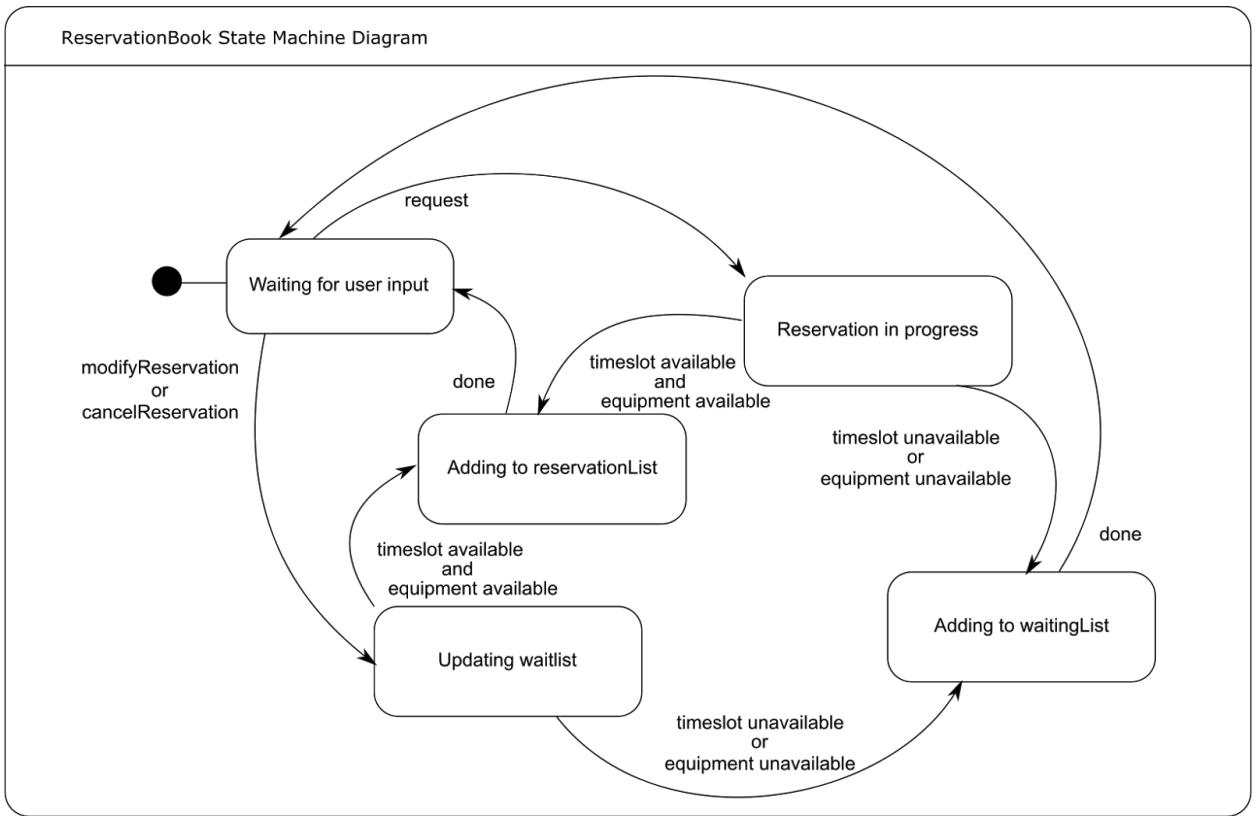

Figure 9. ReservationBook State Diagram