Software Architecture Document

Version 1.5

for

Quickbook Conference Room Reservation System

Prepared by

Piratheeban Annamalai	27755708	piratheeban95@hotmail.com
Andrew Laramee	27050925	zephix109@gmail.com
Jacqueline Luo	26938949	luo.jackie@outlook.com
Michael Mescheder	27202202	meschedermichael@gmail.com
Hoang Khang Nguyen	27079427	alphakennyn@gmail.com
Eric Payette	27008058	eric.payette24@gmail.com

Instructor:	Dr. Constantinos Constantinides
Course:	SOEN 344
Date:	04/04/2017

Document history

Date	Version	Description	Author
10/12/2016	1.0	- Addition of UML Class Diagram with respective associations during team meeting	- All team members
		- Design of communication diagram for critical use cases	- Ideawin, Philip, Hannah
10/25/2016	1.1	- Modification of UML class Diagram	- All team members
		- Addition of Mappers, TDGs, unit of work diagram	-Ideawin, Philip, Hannah
11/21/2016	1.2	- Modification of visibilities for subsystems	-Hannah, Ideawin
		- Addition of methods in Class Diagram for respective restrictions	-Armine, Ideawin, Hannah, Aline
29/03/2017	1.3	-Addition of new requirements including updated diagrams	Eric, Michael, Jacqueline
03/04/2017	1.4	-Modified descriptions in SAD to include new entity	Kenny
04/04/2017	1.5	Updated Class diagram, domain model, sequence diagram and communication diagram	Eric

Table of contents

List of figures

1. Introduction

This document contains a high-level design overview and provides an overall architecture of QuickBook, a web application tool that helps ENCS faculty members reserve a conference room. With high-level descriptions of the goals, the SAD explains the underlying architecture behind some of the use cases such as when a user tries to create, modify and cancel a reservation. This document provides the goals of the architecture, a view of the use cases supported by the system and architectural styles and components that have been selected to best achieve the use cases.

1.1 Purpose

This document provides a comprehensive architectural overview of the QuickBook. To describe different aspects of the system, we have presented three different architectural views such as: Logic view, Data Model view and Use-Case view. The purpose of this documentation is to record and express our architectural decisions which have been made on the system.

1.2 Scope

SAD describes the architecturally significant design aspects of QuickBook. This document can be used to achieve a good understanding of the fundamentals of the system as well as a good guiding tool for duplicating or building the system. Any stakeholder who wants to have a good technical knowledge of QuickBook are encouraged to read this document in order to be able to follow up with the source code.

1.3 Definitions, acronyms, and abbreviations

UML: Unified Modeling Language

SAD: Software Architecture Document

TDG: Table Data Gateway

UoW: Unit of Work

ER: Entity Relationship

2. Architectural representation

QuickBook is implemented as a web application; it is developed using multi-layered architecture which is a client-server architecture. The multilayered architecture is composed of a presentation layer, application (logic) layer, data source (storage) layer.

A description for the QuickBook system will be provided through the representation of different views. The 4+1 view model is composed of Logical View, Process View, Development View, Physical View and Use Case View. This document will only be describing two of the five architectural views: Use Case View and Logical View. An extra view, Data View, will also be described.

In the **Logical View**, a class diagram shows the relationship between classes with their specific associations and dependencies. Also, the logical view includes the communication diagram, which illustrates the interaction of objects, for a given system operation, in a network format.

In the **Use Case View**, a use case model illustrates the functionality the system must provide; its behavior. Furthermore, the use case model displays the relationship between the system's intended functions and the actors (the user).

In the **Data View**, an Entity Relationship (ER) diagram is used to show a visual representation of the logical relationship between the data entities (or objects) of the system in order to build a database.

2.1 Scenarios (Use Case View)

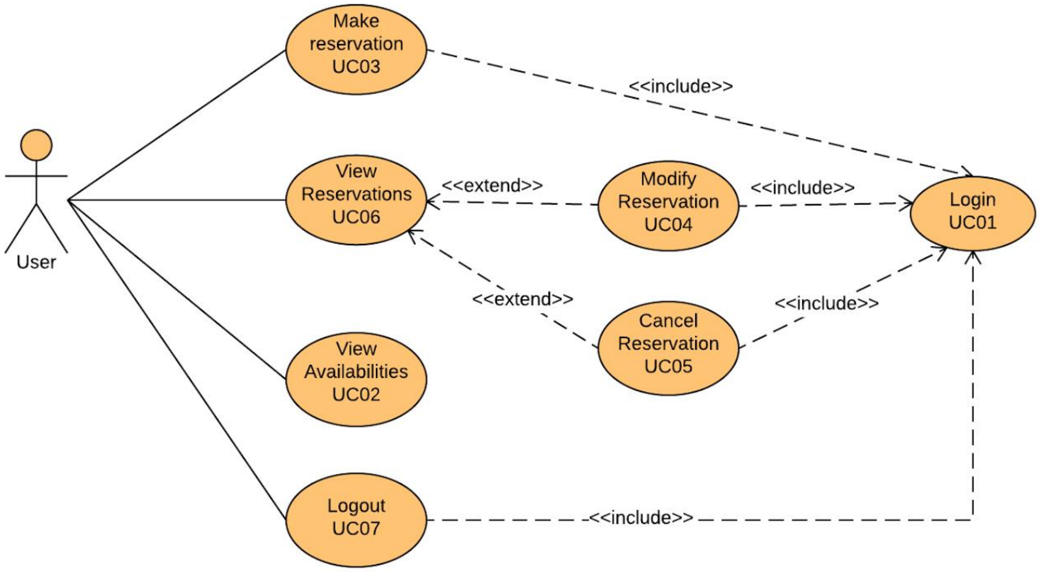


Figure 1: UML Use Case Diagram

Use case 3 **(UC03)** is a *critical use case* of the system because the system depends on its functionality. If users are not able to make a reservation, then the system is entirely non-functional and therefore it doesn't meet its requirements.

The communication diagram (fig. 8) describes the flow of the critical use case operation *makeNewReservation*. The interaction between domain objects is presented, showing how to successfully make a reservation.

2.2 Logical View

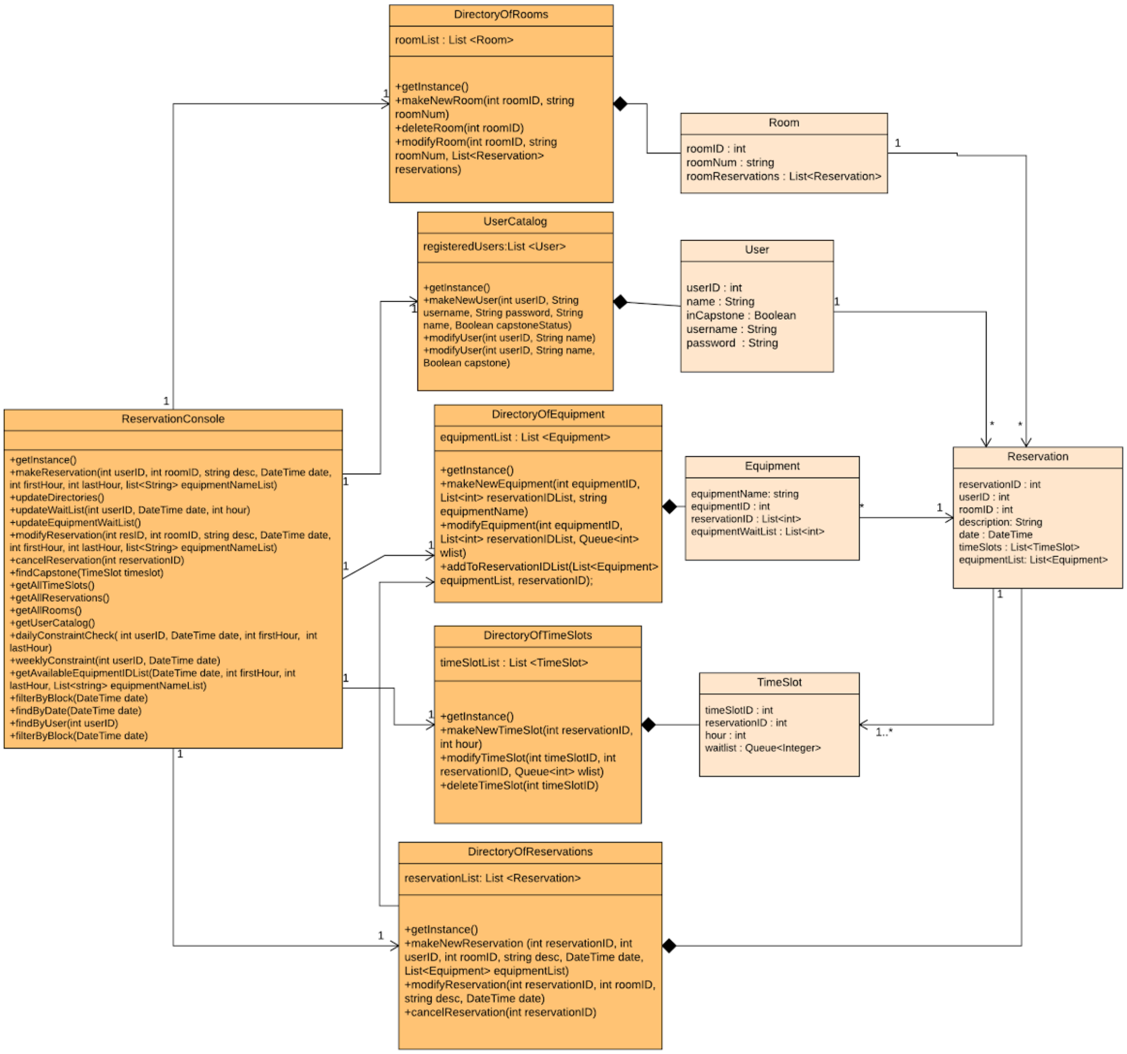

DirectoryOfRooms
roomList : List <Room>
+getInstance()
+makeNewRoom(int roomID, string roomNum)
+deleteRoom(int roomID)
+modifyRoom(int roomID, string roomNum, List<Reservation> reservations)
Room
roomID : int
roomNum : string
roomReservations : List<Reservation>
UserCatalog
registeredUsers:List <User>
+getInstance()
+makeNewUser(int userID, String username, String password, String name, Boolean capstoneStatus)
+modifyUser(int userID, String name)
+modifyUser(int userID, String name, Boolean capstone)
User
userID : int
name : String
inCapstone : Boolean
username : String
password : String
ReservationConsole
+getInstance()
+makeReservation(int userID, int roomID, string desc, DateTime date, int firstHour, int lastHour, list<String> equipmentNameList)
+updateDirectories()
+updateWaitList(int userID, DateTime date, int hour)
+updateEquipmentWaitList()
+modifyReservation(int resID, int roomID, string desc, DateTime date, int firstHour, int lastHour, list<String> equipmentNameList)
+cancelReservation(int reservationID)
+findCapstone(TimeSlot timeslot)
+getAllTimeSlots()
+getAllReservations()
+getAllRooms()
+getUserCatalog()
+dailyConstraintCheck(int userID, DateTime date, int firstHour, int lastHour)
+weeklyConstraint(int userID, DateTime date)
+getAvailableEquipmentIDList(DateTime date, int firstHour, int lastHour, List<string> equipmentNameList)
+filterByBlock(DateTime date)
+findByDate(DateTime date)
+findByUser(int userID)
+filterByBlock(DateTime date)
DirectoryOfEquipment
equipmentList : List <Equipment>
+getInstance()
+makeNewEquipment(int equipmentID, List<int> reservationIDList, string equipmentName)
+modifyEquipment(int equipmentID, List<int> reservationIDList, Queue<int> wlist)
+addToReservationIDList(List<Equipment> equipmentList, reservationID);
Equipment
equipmentName: string
equipmentID : int
reservationID : List<int>
equipmentWaitList : List<int>
Reservation
reservationID : int
userID : int
roomID : int
description: String
date : DateTime
timeSlots : List<TimeSlot>
equipmentList: List<Equipment>
DirectoryOfTimeSlots
timeSlotList : List <TimeSlot>
+getInstance()
+makeNewTimeSlot(int reservationID, int hour)
+modifyTimeSlot(int timeSlotID, int reservationID, Queue<int> wlist)
+deleteTimeSlot(int timeSlotID)
TimeSlot
timeSlotID : int
reservationID : int
hour : int
waitlist : Queue<Integer>
DirectoryOfReservations
reservationList: List <Reservation>
+getInstance()
+makeNewReservation (int reservationID, int userID, int roomID, string desc, DateTime date, List<Equipment> equipmentList)
+modifyReservation(int reservationID, int roomID, string desc, DateTime date)
+cancelReservation(int reservationID)
1
*
1..*

Figure 1: UML Class Diagram

2.2.1 Layers

The system's design uses a layered architectural style. There are three layered views of the system.

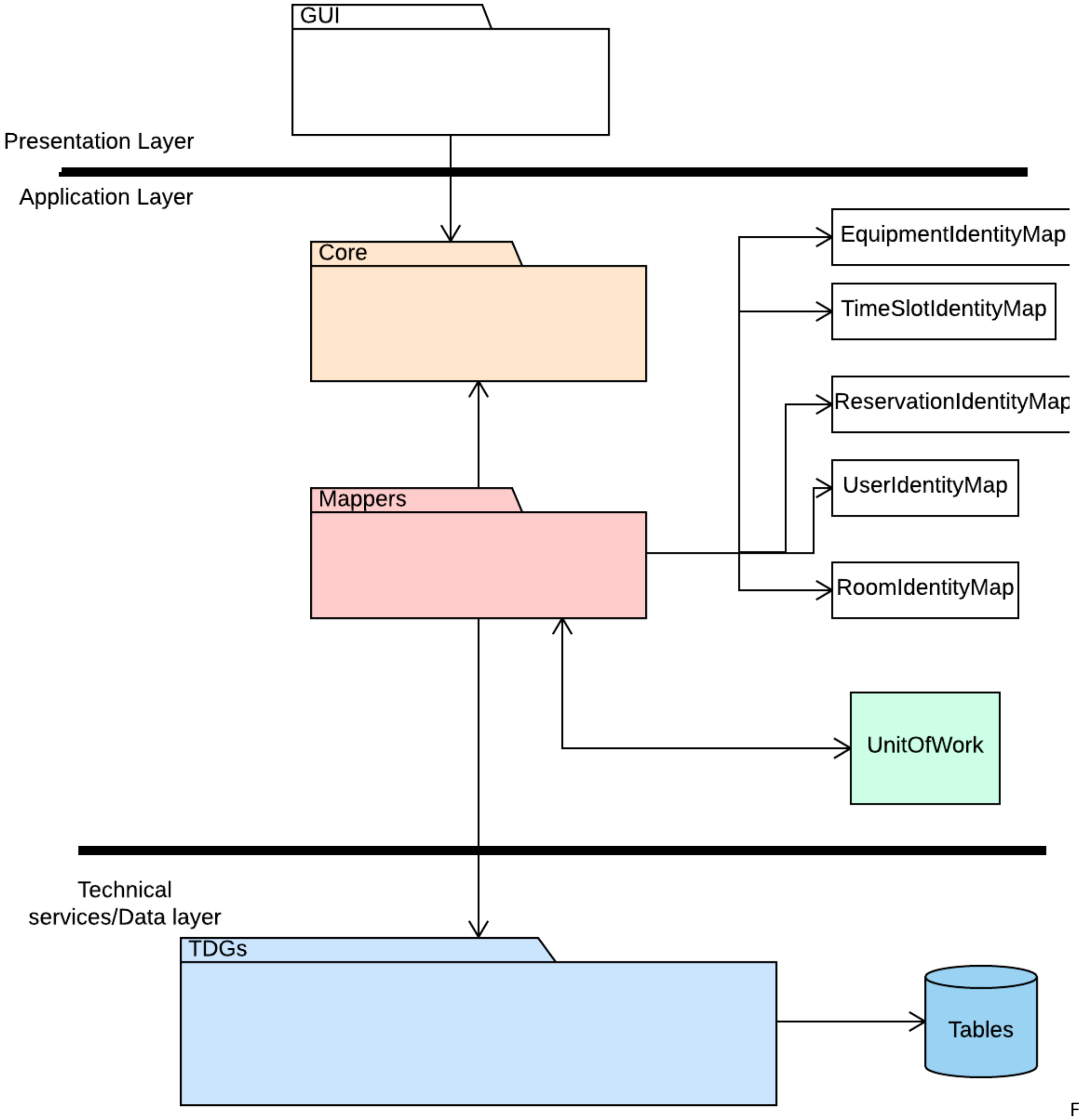


Figure 3: Server-side architecture: logical view: layered architectural style

2.2.2 Subsystems

Decomposition of the system in subsystems and their relation.

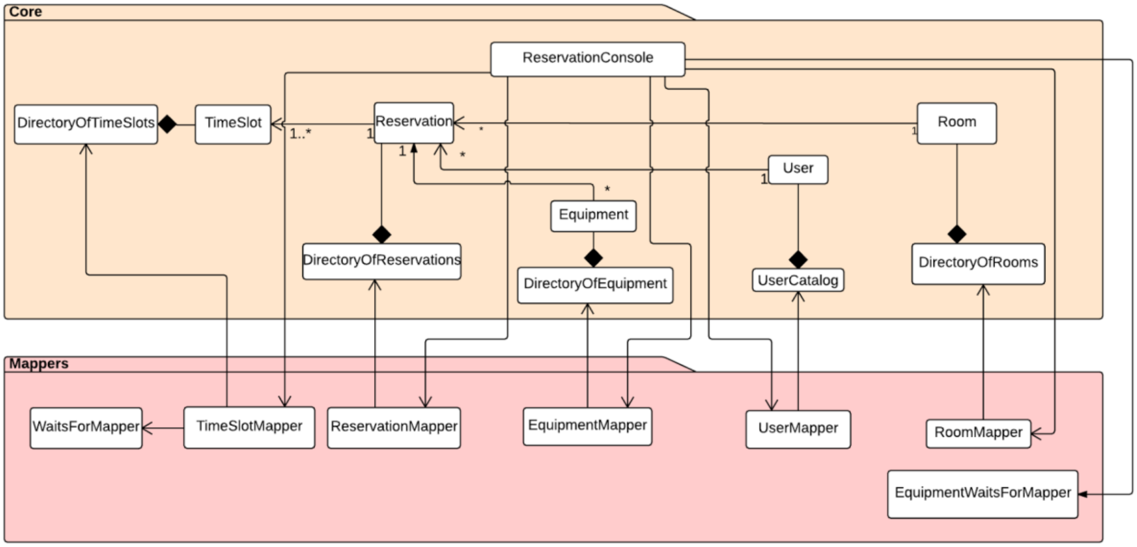
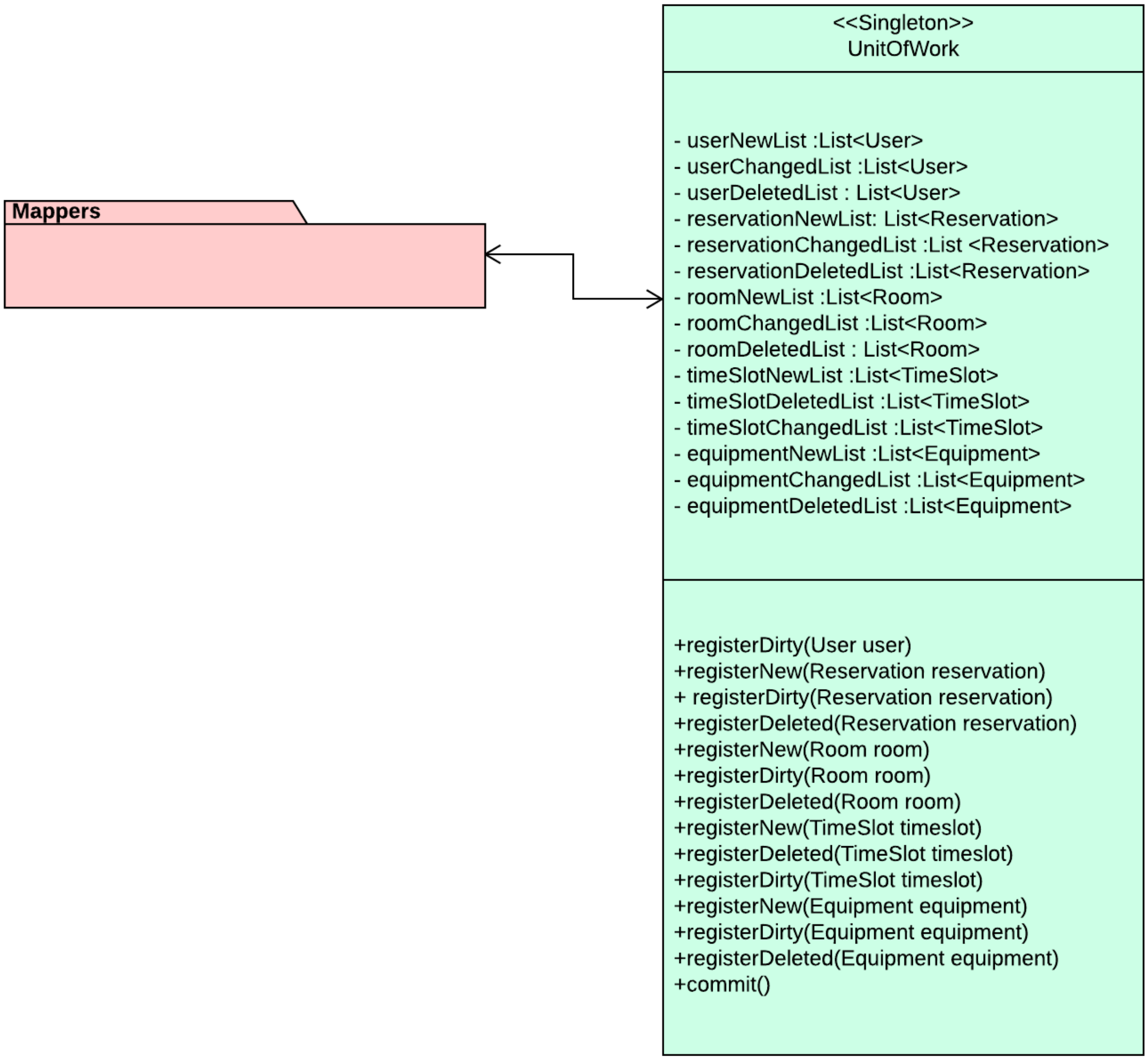
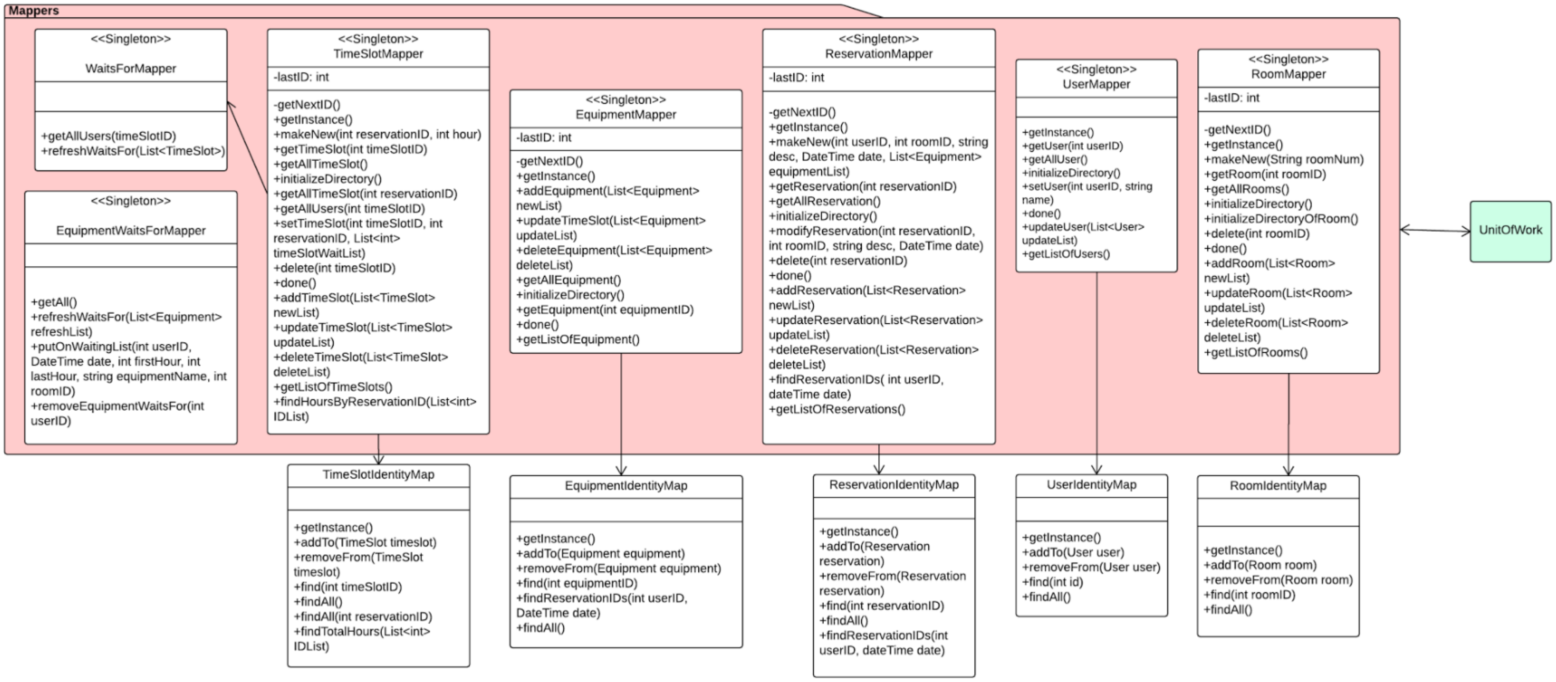


Figure 4: Relationship between Core and Mappers Packages



Mappers
<<Singleton>>
UnitOfWork
- userNewList :List<User>
- userChangedList :List<User>
- userDeletedList : List<User>
- reservationNewList: List<Reservation>
- reservationChangedList :List <Reservation>
- reservationDeletedList :List<Reservation>
- roomNewList :List<Room>
- roomChangedList :List<Room>
- roomDeletedList : List<Room>
- timeSlotNewList :List<TimeSlot>
- timeSlotDeletedList :List<TimeSlot>
- timeSlotChangedList :List<TimeSlot>
- equipmentNewList :List<Equipment>
- equipmentChangedList :List<Equipment>
- equipmentDeletedList :List<Equipment>
+registerDirty(User user)
+registerNew(Reservation reservation)
+ registerDirty(Reservation reservation)
+registerDeleted(Reservation reservation)
+registerNew(Room room)
+registerDirty(Room room)
+registerDeleted(Room room)
+registerNew(TimeSlot timeslot)
+registerDeleted(TimeSlot timeslot)
+registerDirty(TimeSlot timeslot)
+registerNew(Equipment equipment)
+registerDirty(Equipment equipment)
+registerDeleted(Equipment equipment)
+commit()

Figure 5: Relationship between Mappers Package and Unit of Work



Mappers
<<Singleton>>
WaitsForMapper
+getAllUsers(timeSlotID)
+refreshWaitsFor(List<TimeSlot>)
<<Singleton>>
EquipmentWaitsForMapper
+getAll()
+refreshWaitsFor(List<Equipment> refreshList)
+putOnWaitingList(int userID, DateTime date, int firstHour, int lastHour, string equipmentName, int roomID)
+removeEquipmentWaitsFor(int userID)
<<Singleton>>
TimeSlotMapper
-lastID: int
-getNextID()
+getInstance()
+makeNew(int reservationID, int hour)
+getTimeSlot(int timeSlotID)
+getAllTimeSlot()
+initializeDirectory()
+getAllTimeSlot(int reservationID)
+getAllUsers(int timeSlotID)
+setTimeSlot(int timeSlotID, int reservationID, List<int> timeSlotWaitList)
+delete(int timeSlotID)
+done()
+addTimeSlot(List<TimeSlot> newList)
+updateTimeSlot(List<TimeSlot> updateList)
+deleteTimeSlot(List<TimeSlot> deleteList)
+getListOfTimeSlots()
+findHoursByReservationID(List<int> IDList)
<<Singleton>>
EquipmentMapper
-lastID: int
-getNextID()
+getInstance()
+addEquipment(List<Equipment> newList)
+updateTimeSlot(List<Equipment> updateList)
+deleteEquipment(List<Equipment> deleteList)
+getAllEquipment()
+initializeDirectory()
+getEquipment(int equipmentID)
+done()
+getListOfEquipment()
<<Singleton>>
ReservationMapper
-lastID: int
-getNextID()
+getInstance()
+makeNew(int userID, int roomID, string desc, DateTime date, List<Equipment> equipmentList)
+getReservation(int reservationID)
+getAllReservation()
+initializeDirectory()
+modifyReservation(int reservationID, int roomID, string desc, DateTime date)
+delete(int reservationID)
+done()
+addReservation(List<Reservation> newList)
+updateReservation(List<Reservation> updateList)
+deleteReservation(List<Reservation> deleteList)
+findReservationIDs(int userID, dateTime date)
+getListOfReservations()
<<Singleton>>
UserMapper
+getInstance()
+getUser(int userID)
+getAllUser()
+initializeDirectory()
+setUser(int userID, string name)
+done()
+updateUser(List<User> updateList)
+getListOfUsers()
<<Singleton>>
RoomMapper
-lastID: int
-getNextID()
+getInstance()
+makeNew(String roomNum)
+getRoom(int roomID)
+getAllRooms()
+initializeDirectory()
+initializeDirectoryOfRoom()
+delete(int roomID)
+done()
+addRoom(List<Room> newList)
+updateRoom(List<Room> updateList)
+deleteRoom(List<Room> deleteList)
+getListOfRooms()
UnitOfWork
TimeSlotIdentityMap
+getInstance()
+addTo(TimeSlot timeslot)
+removeFrom(TimeSlot timeslot)
+find(int timeSlotID)
+findAll()
+findAll(int reservationID)
+findTotalHours(List<int> IDList)
EquipmentIdentityMap
+getInstance()
+addTo(Equipment equipment)
+removeFrom(Equipment equipment)
+find(int equipmentID)
+findReservationIDs(int userID, DateTime date)
+findAll()
ReservationIdentityMap
+getInstance()
+addTo(Reservation reservation)
+removeFrom(Reservation reservation)
+find(int reservationID)
+findAll()
+findReservationIDs(int userID, dateTime date)
UserIdentityMap
+getInstance()
+addTo(User user)
+removeFrom(User user)
+find(int id)
+findAll()
RoomIdentityMap
+getInstance()
+addTo(Room room)
+removeFrom(Room room)
+find(int roomID)
+findAll()

Figure 6: Relationship between Mappers and Identity Maps

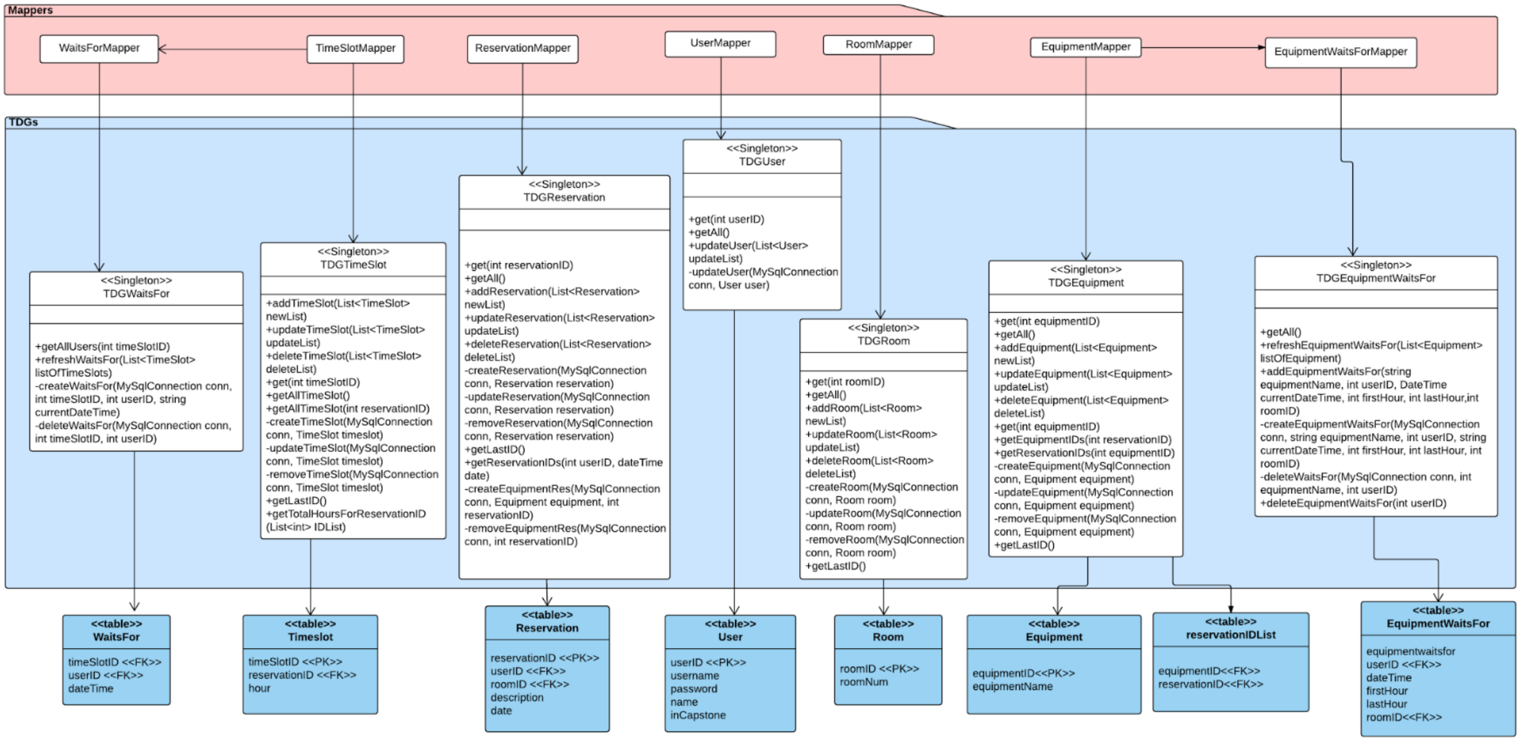

Mappers
WaitsForMapper
TimeSlotMapper
ReservationMapper
UserMapper
RoomMapper
EquipmentMapper
EquipmentWaitsForMapper
TDGs
<<Singleton>>
TDGWaitsFor
+getAllUsers(int timeSlotID)
+refreshWaitsFor(List<TimeSlot> listOfTimeSlots)
-createWaitsFor(MySqlConnection conn, int timeSlotID, int userID, string currentDateTime)
-deleteWaitsFor(MySqlConnection conn, int timeSlotID, int userID)
<<Singleton>>
TDGTimeSlot
+addTimeSlot(List<TimeSlot> newList)
+updateTimeSlot(List<TimeSlot> updateList)
+deleteTimeSlot(List<TimeSlot> deleteList)
+get(int timeSlotID)
+getAllTimeSlot()
+getAllTimeSlot(int reservationID)
-createTimeSlot(MySqlConnection conn, TimeSlot timeslot)
-updateTimeSlot(MySqlConnection conn, TimeSlot timeslot)
-removeTimeSlot(MySqlConnection conn, TimeSlot timeslot)
+getLastID()
+getTotalHoursForReservationID (List<int> IDList)
<<Singleton>>
TDGReservation
+get(int reservationID)
+getAll()
+addReservation(List<Reservation> newList)
+updateReservation(List<Reservation> updateList)
+deleteReservation(List<Reservation> deleteList)
-createReservation(MySqlConnection conn, Reservation reservation)
-updateReservation(MySqlConnection conn, Reservation reservation)
-removeReservation(MySqlConnection conn, Reservation reservation)
+getLastID()
+getReservationIDs(int userID, dateTime date)
-createEquipmentRes(MySqlConnection conn, Equipment equipment, int reservationID)
-removeEquipmentRes(MySqlConnection conn, int reservationID)
<<Singleton>>
TDGUser
+get(int userID)
+getAll()
+updateUser(List<User> updateList)
-updateUser(MySqlConnection conn, User user)
<<Singleton>>
TDGRoom
+get(int roomID)
+getAll()
+addRoom(List<Room> newList)
+updateRoom(List<Room> updateList)
+deleteRoom(List<Room> deleteList)
-createRoom(MySqlConnection conn, Room room)
-updateRoom(MySqlConnection conn, Room room)
-removeRoom(MySqlConnection conn, Room room)
+getLastID()
<<Singleton>>
TDGEquipment
+get(int equipmentID)
+getAll()
+addEquipment(List<Equipment> newList)
+updateEquipment(List<Equipment> updateList)
+deleteEquipment(List<Equipment> deleteList)
+get(int equipmentID)
+getEquipmentIDs(int reservationID)
+getReservationIDs(int equipmentID)
-createEquipment(MySqlConnection conn, Equipment equipment)
-updateEquipment(MySqlConnection conn, Equipment equipment)
-removeEquipment(MySqlConnection conn, Equipment equipment)
+getLastID()
<<Singleton>>
TDGEquipmentWaitsFor
+getAll()
+refreshEquipmentWaitsFor(List<Equipment> listOfEquipment)
+addEquipmentWaitsFor(string equipmentName, int userID, DateTime currentDateTime, int firstHour, int lastHour,int roomID)
-createEquipmentWaitsFor(MySqlConnection conn, string equipmentName, int userID, string currentDateTime, int firstHour, int lastHour, int roomID)
-deleteWaitsFor(MySqlConnection conn, int equipmentName, int userID)
+deleteEquipmentWaitsFor(int userID)
<<table>>
WaitsFor
timeSlotID <<FK>>
userID <<FK>>
dateTime
<<table>>
Timeslot
timeSlotID <<PK>>
reservationID <<FK>>
hour
<<table>>
Reservation
reservationID <<PK>>
userID <<FK>>
roomID <<FK>>
description
date
<<table>>
User
userID <<PK>>
username
password
name
inCapstone
<<table>>
Room
roomID <<PK>>
roomNum
<<table>>
Equipment
equipmentID<<PK>>
equipmentName
<<table>>
reservationIDList
equipmentID<<FK>>
reservationID<<FK>>
<<table>>
EquipmentWaitsFor
equipmentwaitsfor
userID <<FK>>
dateTime
firstHour
lastHour
roomID<<FK>>

Figure 7: Relationship between Mappers, TDGs and Tables

2.2.3 Use Case Realizations

To clearly describe the important architectural elements of *QuickBook,* interaction diagrams are provided for the critical use case Make Reservation: **UC03.**

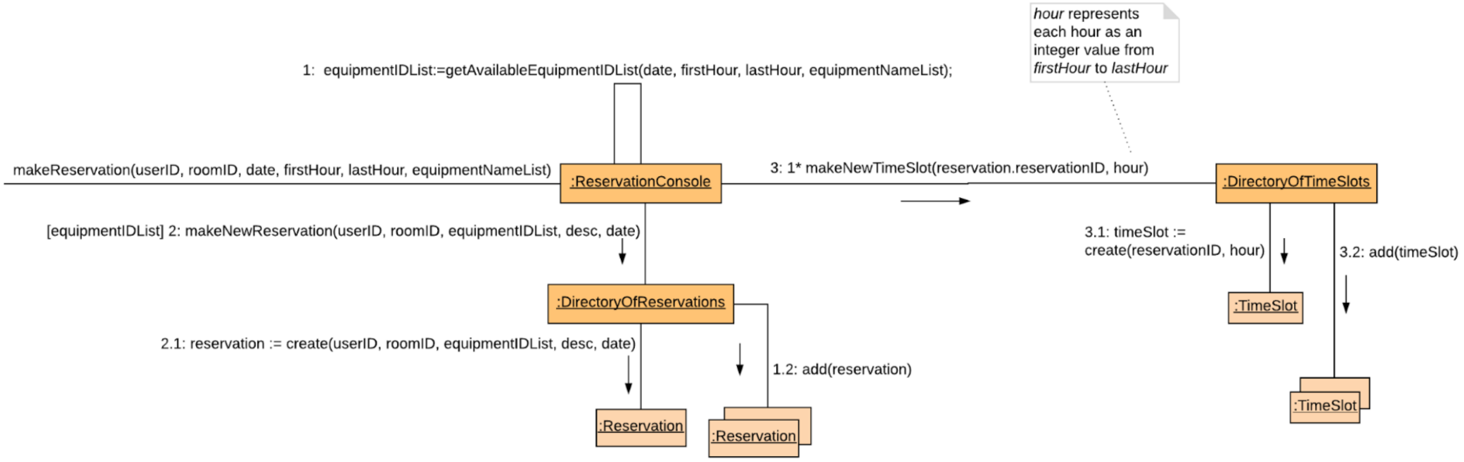

hour represents each hour as an integer value from *firstHour* to *lastHour*
1: equipmentIDList:=getAvailableEquipmentIDList(date, firstHour, lastHour, equipmentNameList);
makeReservation(userID, roomID, date, firstHour, lastHour, equipmentNameList)
:ReservationConsole
3: 1* makeNewTimeSlot(reservation.reservationID, hour)
:DirectoryOfTimeSlots
[equipmentIDList] 2: makeNewReservation(userID, roomID, equipmentIDList, desc, date)
3.1: timeSlot := create(reservationID, hour)
3.2: add(timeSlot)
:DirectoryOfReservations
:TimeSlot
2.1: reservation := create(userID, roomID, equipmentIDList, desc, date)
1.2: add(reservation)
:TimeSlot
:Reservation
:Reservation

Figure 8: Communication Diagram

The following Sequence diagrams illustrate the UMl interactions between the Domain Object, Mappers, Identity Maps, Unit of Work (UoW), Table Data Getaways (TDG) and the database table.

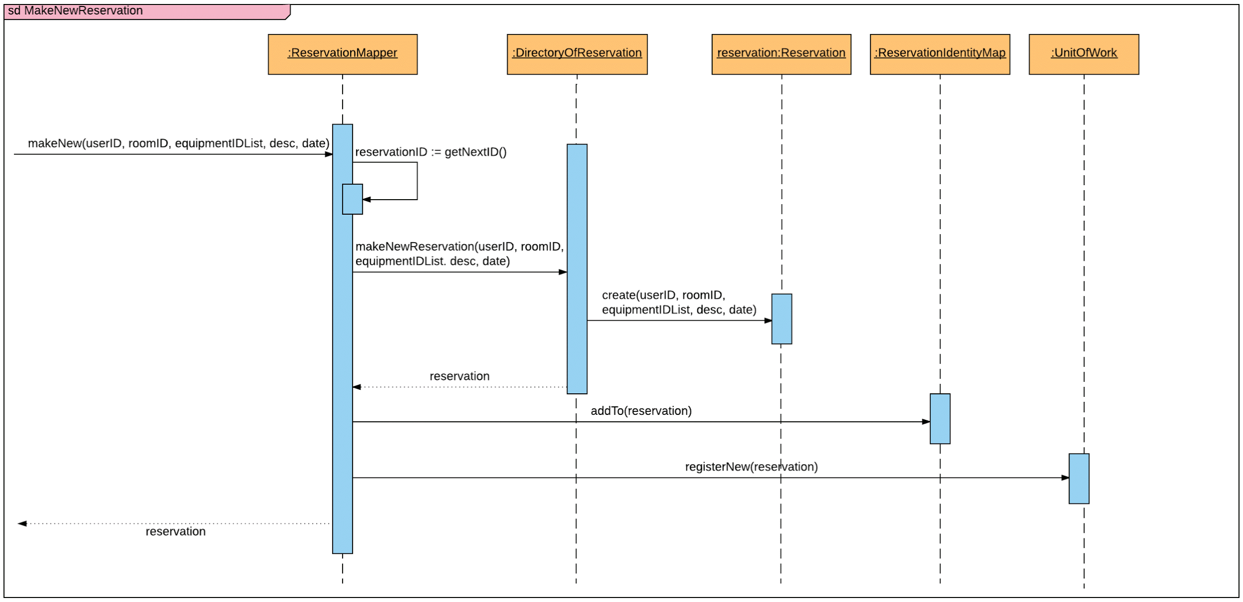

sd MakeNewReservation
:ReservationMapper
:DirectoryOfReservation
reservation:Reservation
:ReservationIdentityMap
:UnitOfWork
makeNew(userID, roomID, equipmentIDList, desc, date)
reservationID := getNextID()
makeNewReservation(userID, roomID, equipmentIDList. desc, date)
create(userID, roomID, equipmentIDList, desc, date)
reservation
addTo(reservation)
registerNew(reservation)
reservation

Figure 9: Sequence Diagram to Make a New Reservation

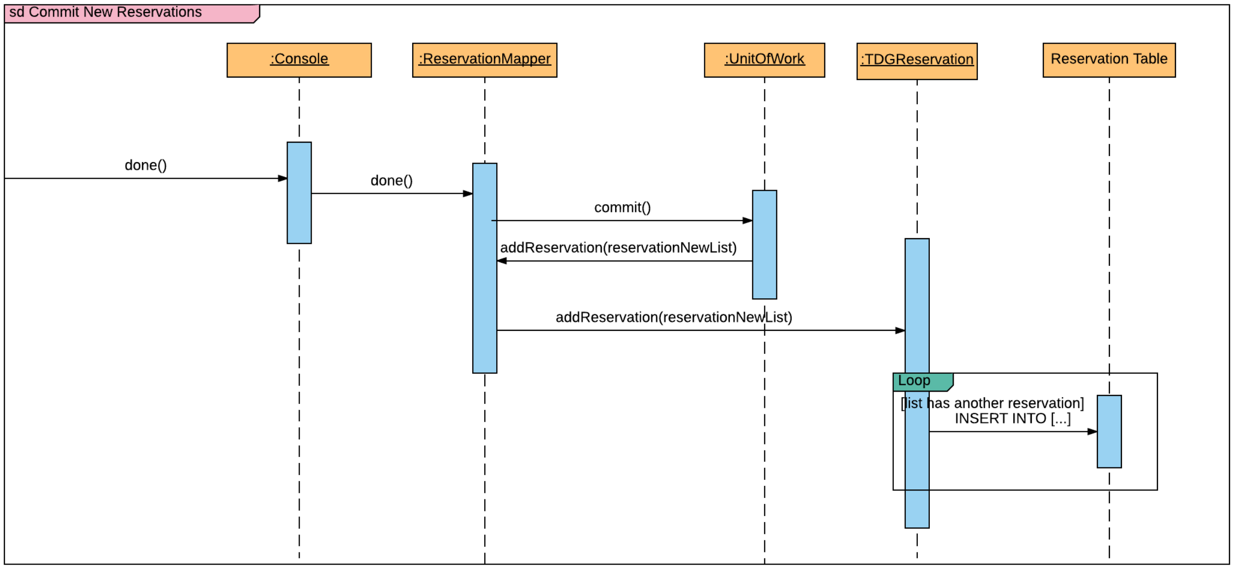

sd Commit New Reservations
:Console
:ReservationMapper
:UnitOfWork
:TDGReservation
Reservation Table
done()
done()
commit()
addReservation(reservationNewList)
addReservation(reservationNewList)
Loop
[list has another reservation]
INSERT INTO [...]

Figure 10: Sequence Diagram When a User is Done and Mapper are Ready to Commit to Unit of Work

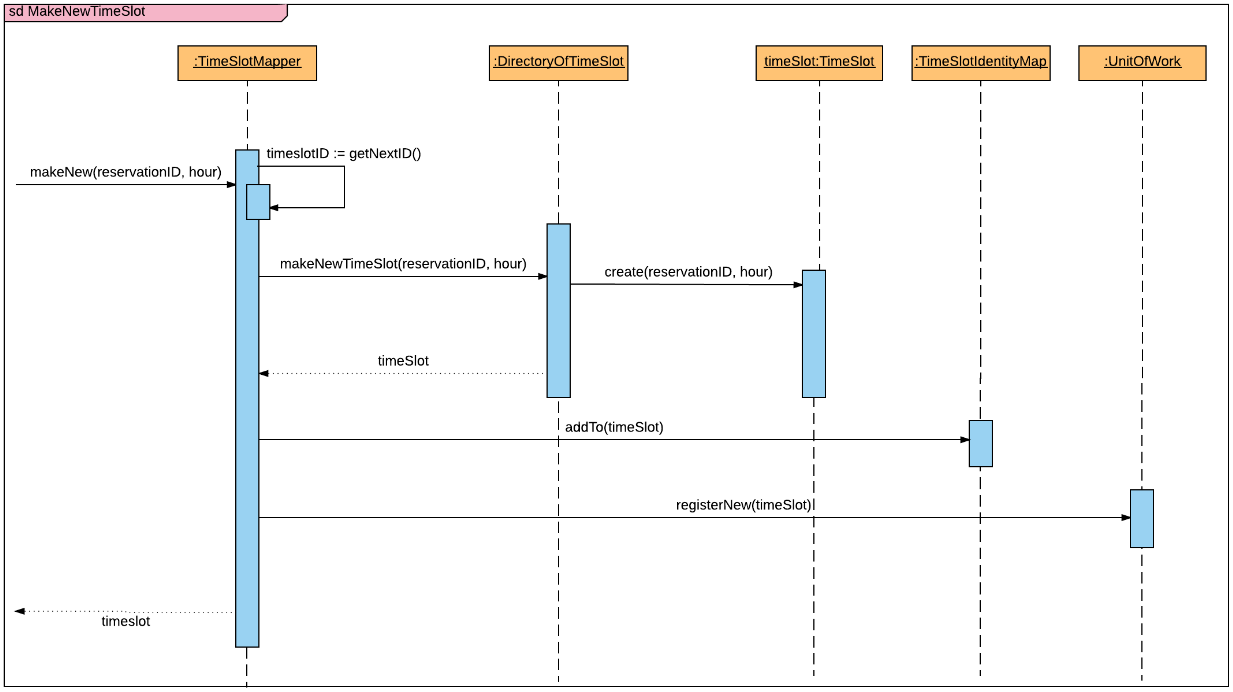


Figure 11: Sequence Diagram to make a new TimeSlot

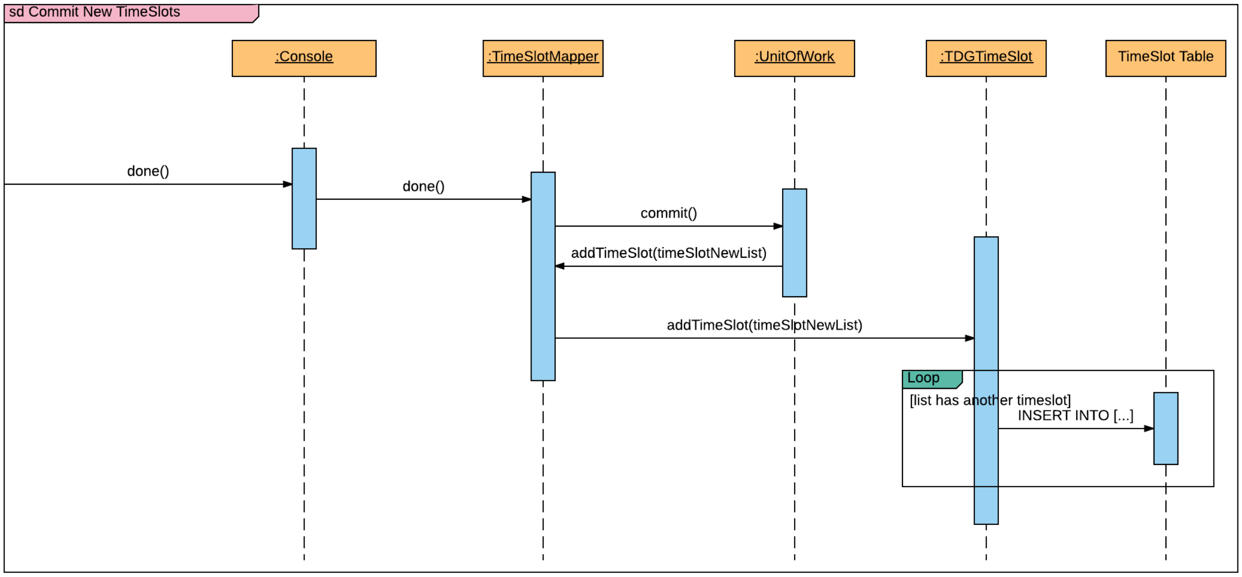
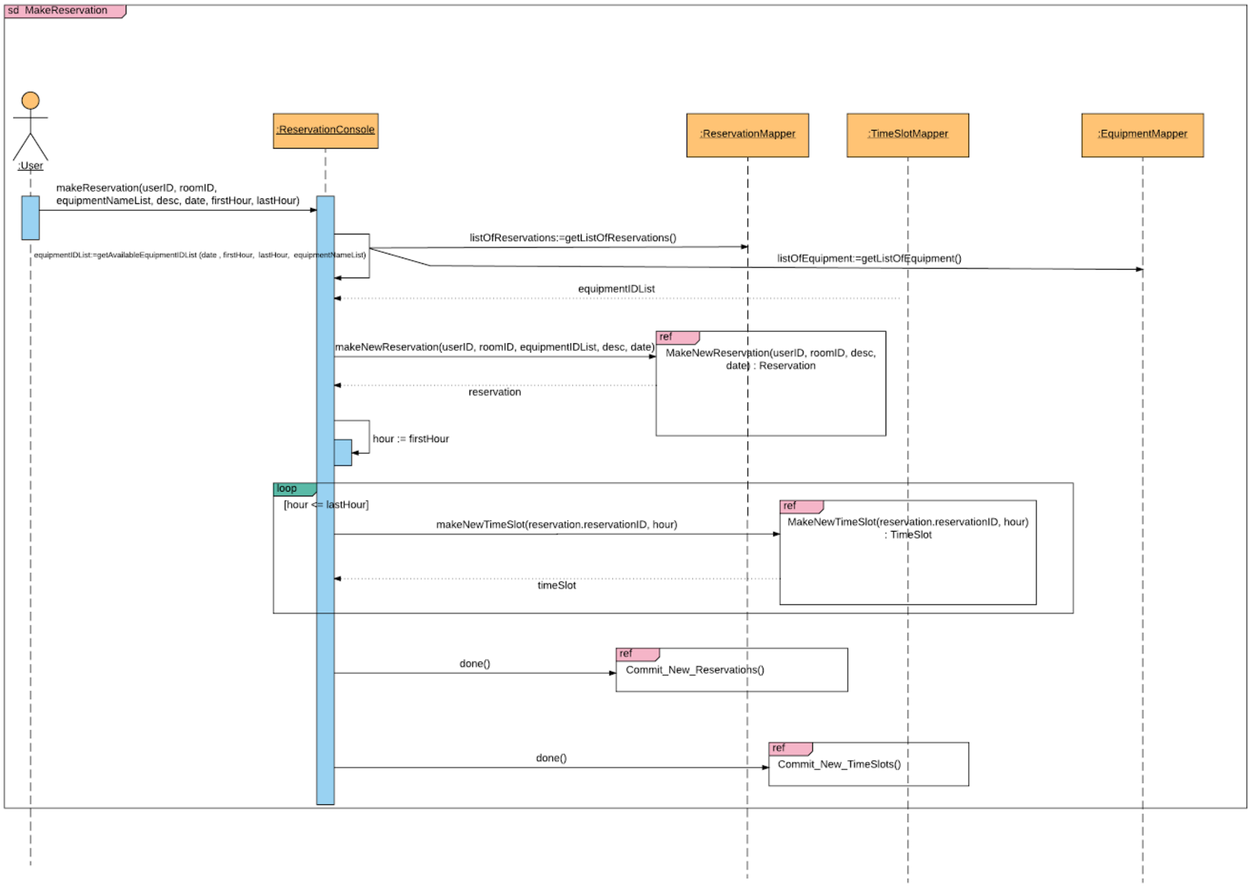


Figure 12: Sequence Diagram When User is Ready to Commit to Unit of Work



sd MakeReservation
:User
:ReservationConsole
:ReservationMapper
:TimeSlotMapper
:EquipmentMapper
makeReservation(userID, roomID, equipmentNameList, desc, date, firstHour, lastHour)
equipmentIDList:=getAvailableEquipmentIDList (date , firstHour, lastHour, equipmentNameList)
listOfReservations:=getListOfReservations()
listOfEquipment:=getListOfEquipment()
equipmentIDList
makeNewReservation(userID, roomID, equipmentIDList, desc, date)
ref
MakeNewReservation(userID, roomID, desc, date) : Reservation
reservation
hour := firstHour
loop
[hour <= lastHour]
makeNewTimeSlot(reservation.reservationID, hour)
ref
MakeNewTimeSlot(reservation.reservationID, hour) : TimeSlot
timeSlot
done()
ref
Commit_New_Reservations()
done()
ref
Commit_New_TimeSlots()

Figure 13: Complete Sequence Diagram for Make Reservation

2.3 Data View

Five main entities are present in the system: The User, the Reservation, the Room, the Equipment and the TimeSlot. The below Entity-Relationship (ER) model shows the attributes of each entity and the relationships between them.

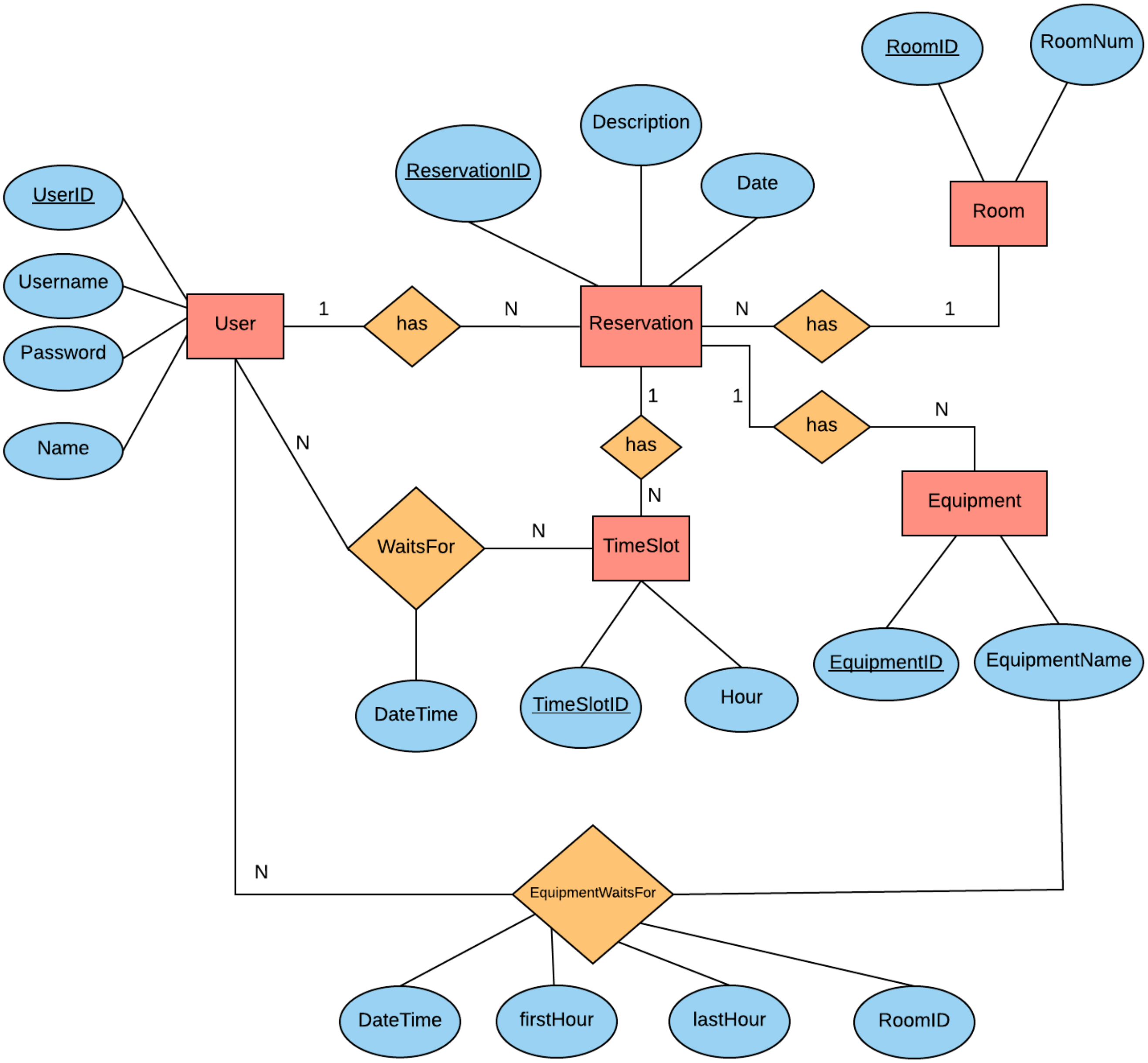


Figure 14: Entity-Relationship Model

3. Architectural requirements: goals and constraints

Functional Requirements and Non-Functional requirements can affect the architecture of a software system.

3.1 Functional requirements (Use case view)

This section describes which Use Cases from the Use Case Model are relevant to the software architecture.

Source	**Name**	**Architectural relevance**	**Addressed in:**
SRS	*Use Case Login*	*-A user is needed for most methods*	*Section 2.1*
SRS	*Use Case Make Reservation*	*-Making a reservation must be functional in the system for most methods to function.*	*Section 2.1*

3.2 Non-functional requirements (NFRs)

This section describes the non-functional requirements that are relevant to the architecture of the *Quickbook* Software. The two most important type of technical NFRs are Usability and Maintainability.

Source	**Name**	**Architectural relevance**	**Addressed in:**
SRS	*Usability*	*-Consistency of both the user interface and*	*Section 3.3.2*

		the functionality of the system. *-Natural Mapping : ease of navigation*	
SRS	*Maintainability*	*-When adding or changing of the functionality and meeting new requirements, the system endures these changes with a degree of ease.*	*Section 3.3.4*