TEAM LOTUS

Concordia University
Department of Computer Science
& Software Engineering

Software Architecture and Design II
SOEN 344 H --- 2016

CHANGELOG DOCUMENT

Team members	
Name	**Student ID**
An Ran Chen	27277385
Christiano Bianchet	27039573
Philippe Kuret	27392680
Alexander Rosser	27543069
Saif Mahabub	27392974
Amr Mourad	26795331
Youness Tawil	27013353
Adriel Fabella	27466005

1. PROJECT CHRONOS

The team spent three sprints, each consisting of two weeks, completing tasks for this project. Meetings were held every Tuesday afternoon.

Sprint 1	February 19 - March 6
Sprint 2	March 7 - March 20
Sprint 3	March 21 - April 3

Table 1 Project Chronos Schedule

Changelog table

The following table outlines the changes made to project *Chronos* sorted by completion date, and the issue number they refer to. Each issue links to a more detailed explanation.

Sprint	Date Completed	Change Type	Change Log Entry	Reference	Author
Sprint 1	February 20,2017	Fix	Update Use Cases	Issue #35	Alexander Rosser
Sprint 1	February 20, 2017	Fix	Use Case View	Issue #29	Alexander Rosser
Sprint 1	February 20, 2017	Fix	System Sequence Diagram & Operation Contracts	Issue #32	Saif Mahabub
Sprint 1	February 20, 2017	Fix	Architectural Style	Issue #39	Christiano Bianchet

Sprint 1	February 21, 2017	Feature	5 rooms at 24-hour basis	Issue#18	Amr Mourad
Sprint 1	February 21, 2017	Feature	Max reservation of 3h a week	Issue #19	Amr Mourad
Sprint 1	February 21, 2017	Feature	Recur max of 3 weeks	Issue #37	Amr Mourad
Sprint 1	February 23, 2017	Fix	Domain Model	Issue #31	Adriel Fabella
Sprint 1	February 26, 2017	Fix	Interaction Diagram	Issue #28	Philippe Kuret, Youness Tawil
Sprint 1	March 4, 2017	Fix	Reservation in past 24hrs are allowed	Issue #46	Amr Mourad
Sprint 1	March 4, 2017	Fix	Functional Requirements	Issue #45	Saif Mahabub
Sprint 1	March 4, 2017	Fix	Data Class in Class Diagram	Issue #27	Christiano Bianchet
Sprint 1	March 4, 2017	Feature	Request equipments in reservation rooms	Issue #41	Amr Mourad

Sprint 2	March 9, 2017	Feature	Capstone student priority	Issue #42	Amr Mourad
Sprint 2	March 9, 2017	Improvement	Visual cue showing whether or not a user is capstone	Issue #53	Amr Mourad
Sprint 2	March 13, 2017	Fix	Object-relational mapping	Issue #30	Christiano Bianchet
Sprint 2	March 13, 2017	Fix	Update the interaction diagrams for the new requirements	Issue #63	Christiano Bianchet
Sprint 2	March 14, 2017	Testing	Document and execute user acceptance test	Issue #60	Saif Mahabub
Sprint 2	March 15, 2017	Fix & Feature	Fixed class diagram & added new components	Issue #64	Philippe Kuret
Sprint 2	March 15, 2017	Fix & Feature	Added missing classes & added new requirement classes	Issue #43	Philippe Kuret
Sprint 2	March 18, 2017	Fix	Refactored the unit of work	Issue #54	Adriel Fabella
Sprint 3	April 1, 2017	Testing	Document the unit tests	Issue #56	Youness Tawil, Adriel Fabella

Table 2: Chronos Changelog table

Issue #35 - [Fix] Update Use Cases

Author: Alexander Rosser

Small changes were performed on use cases 1-6 and 8. The original use cases were completed correctly, however they were poorly phrased. Sentences were restructured to clarify the use cases. Furthermore dates were added to their existing notes and issues to avoid confusion with the new changes.

Issue #29 - [Fix] Use Case View

Author: Alexander Rosser

The use case model was updated to better reflect the relation between the use cases. As seen below, the old use case displayed no interaction between uses cases. The view was updated to display that use cases Log Out, View Calendar, Request Reservation, Modify Reservation, View Reservation, View Reservation List and Cancel Reservation extend the use case Log In as they all require that there be an active session. Furthermore, Cancel Reservation and Modify Reservation were linked to View Reservation as they both require that the user be viewing a reservation.

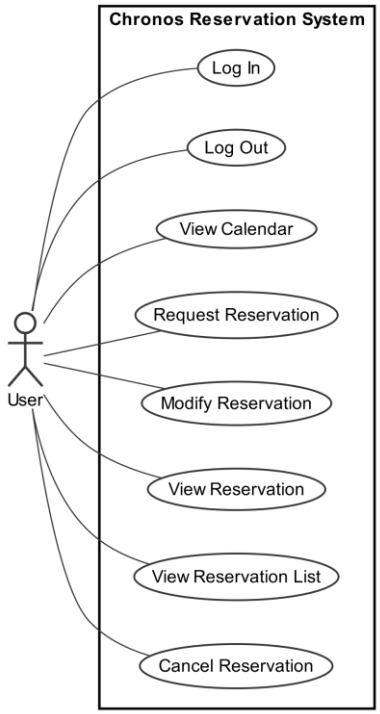

Figure 1: Old Use Case View

Issue #32 - [Fix] System Sequence Diagram & Operation Contracts

Author: Saif Mahabub

For the **Request Reservation** system sequence diagram, inconsistencies in the method's parameters were modified to match the code (with the new requirements) and a missing loop was added in order to perform multiple reservations until the user decides to stop or reached the maximum possible reservations.

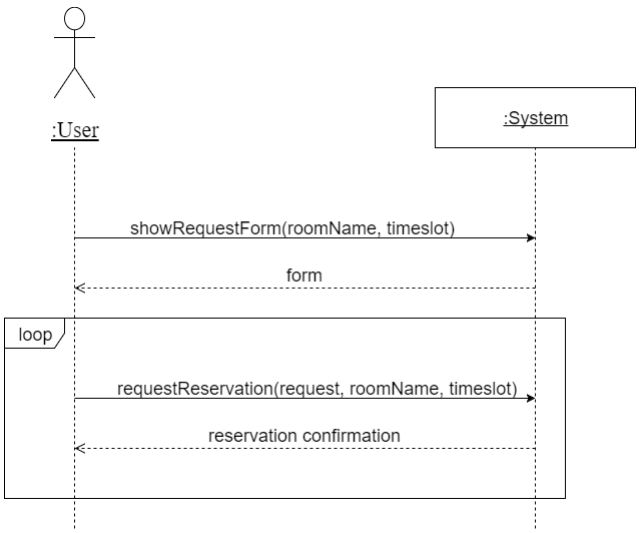

Figure 2: Request Reservation SSD

For the **Modify Reservation** system sequence diagram, inconsistencies in the method's parameters were modified to match the code and a missing loop was added to modify multiple reservations until the user is satisfied with the modifications.

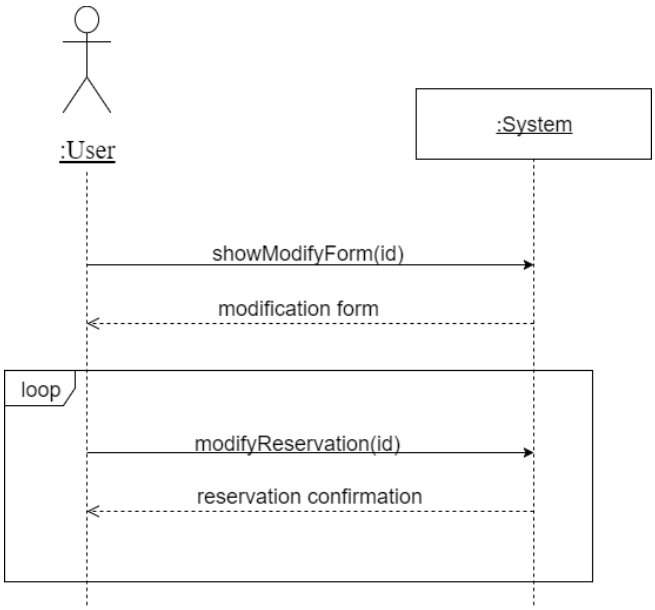

Figure 3: Modify Reservation SSD

The **View Reservation and Cancel Reservation** system sequence diagram had an inconsistent parameter in the method, thus was modified corresponding to the code (with the new requirements). Furthermore, the Cancel Reservation SSD was missing a loop and hence added to cancel multiple reservations until the user stops or there is no more reservations left to cancel.

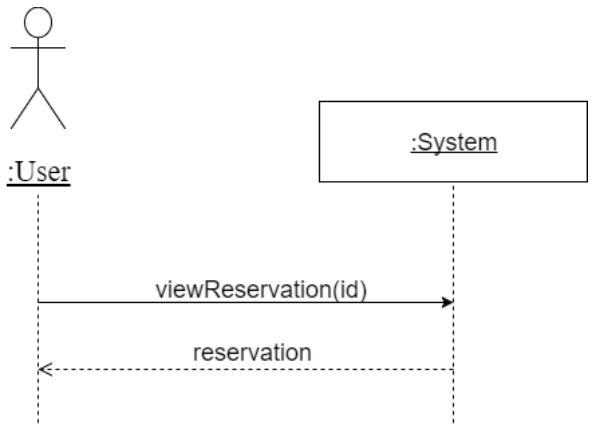

Figure 4: View Reservation SSD

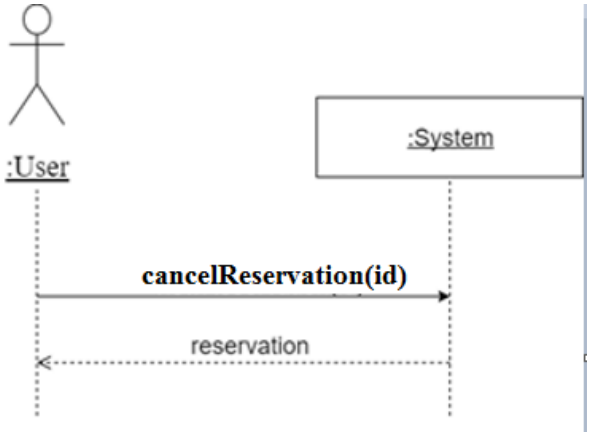

Figure 5: Cancel Reservation SSD

Issue #39 - [Fix] Architectural Style

Author: Christiano Bianchet

Documentation on the architectural styles present in the system was very brief and without a real explanation of what a layered style is or what it meant. To remedy this, an overview section was added which goes further in-depth in describing not only what is meant by a layered architectural style but also which other styles are present in the system (Client-Server, Tier). The descriptions of each layer, sections 2.1.2, 2.1.3 and 2.1.4, generally explained how each layer functioned as seen in the Logical View diagram. An example was added to each section to further clarify this, as well as more specific descriptions of each individual object type (Mappers, Maps, TDGs and Controllers).

Issue #18 - [Feature] 5 rooms at 24-hour basis

Author: Amr Mourad

This new feature implements the first requirement of having five rooms available on a 24-hour basis. The system was straightforward to extend, by reducing the number of available rooms from ten to five which involves changing the initialization/seeding of the database, as well as relaxing the front-end constraint, which only allowed reservations to be made between seven in the morning to eleven at night, in order to allow for 24-hour reservations.

Pull request #34 includes this requirement: https://github.com/anrchen/soen344/pull/34.

Issue #37 & #19 - [Feature] Recur max of 3 weeks & max reservation of 3h a week

Author: Amr Mourad

These new functionalities follow requirement number 2.

Firstly, the maximum number of reservation a user can do on a weekly basis (Monday to Sunday) is three, for a total of 3 hours. This is implemented upon a user requesting a reservation. A verification is made to the database to ensure that the there is space for that requested reservation.

Secondly, the requirement specifies that a maximum of 3 recurring reservations can be done. This means that a reservation cannot exist for the same user in three consecutive weeks where the room and timeslot are the same in a given week day. To implement this, when the user attempts to request a reservation, a future check is done recursively to find whether recurring reservations exist. For example, if a user wants to reserve a recurring reservation for two weeks, a verification is made to ensure that future reservations don't extend this reservation and exceed the limit of 3. If the future check does not exceed the limit, the verification is done in the previous weeks, using the same logic. As long as the total recurrence (future & past) does not exceed the limit of 3, the reservation will be added successfully, otherwise it will not and the user will be notified accordingly. On the front-end, the limit for a recurring selection at once is set to three in the drop-down, to prevent the user from mistakenly reserving an unacceptable amount.

Pull request #40 includes this requirement: https://github.com/anrchen/soen344/pull/40.

Issue #31 - [Fix] Domain Model

Author: Adriel Fabella

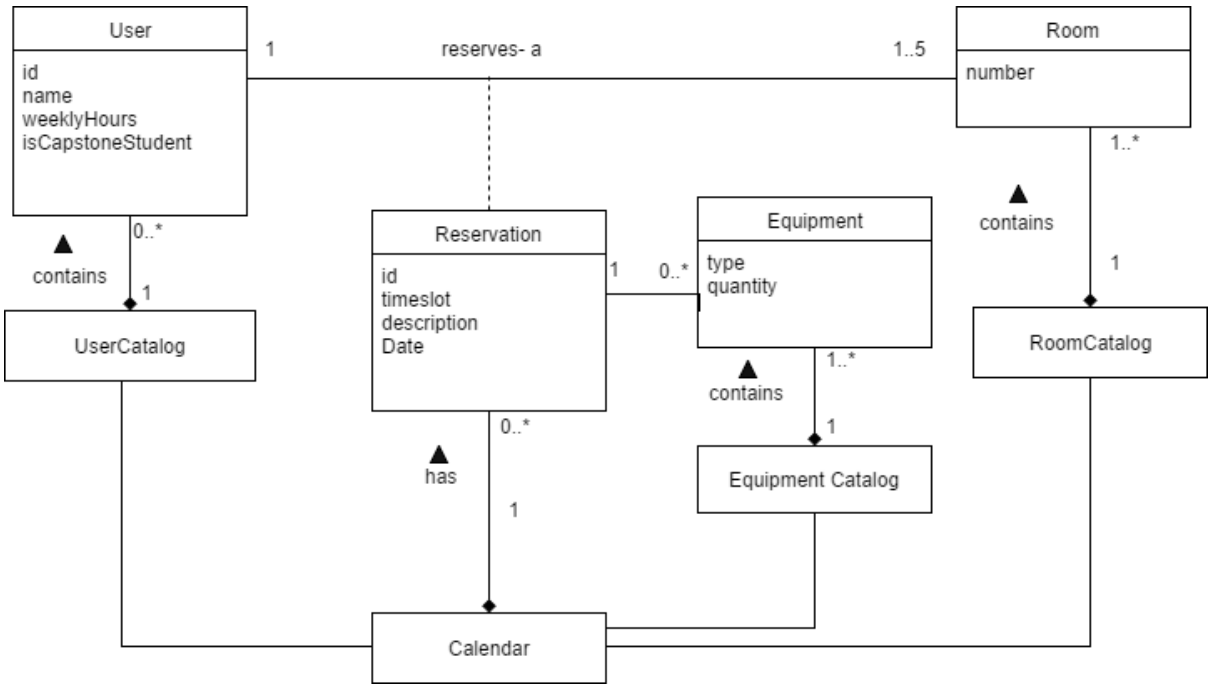

Figure 6: Domain Model

The Domain Model has been updated to accommodate for the new requirements. The main update from the previous domain model is the “equipment” and “equipment catalog” concepts. The calendar has access to an equipment catalog which contains equipment. Each equipment is given a type (projector, adapters, etc.) and the quantity. An association is created between reservation and equipment. This indicates that a reservation, which consists of a room and a user, has the possibility of renting an equipment. This association is created since the requirements specify that a reservation cannot be made if the required equipment is not available.

Issue # 28 - [Fix] Interaction Diagram

Author: Philippe Kuret, Youness Tawil

Changes were made to the interaction diagrams because some classes were labeled incorrectly such as UnitOfWork. In addition, a lot of the methods used in the diagrams are missing parameters. Some of the return operations are not represented properly. For example, return statements were added when nothing should have been returned. Overall, the diagrams did not fully reflect the interaction between classes and the flow of data in the system. The changes made are fully representative of the system we are using.

Show Request Form (2.3.2.1)

For this interaction diagram, we have added

- parameter “userID, start, end” for countInRange(),
- Added a return for the call to the database.

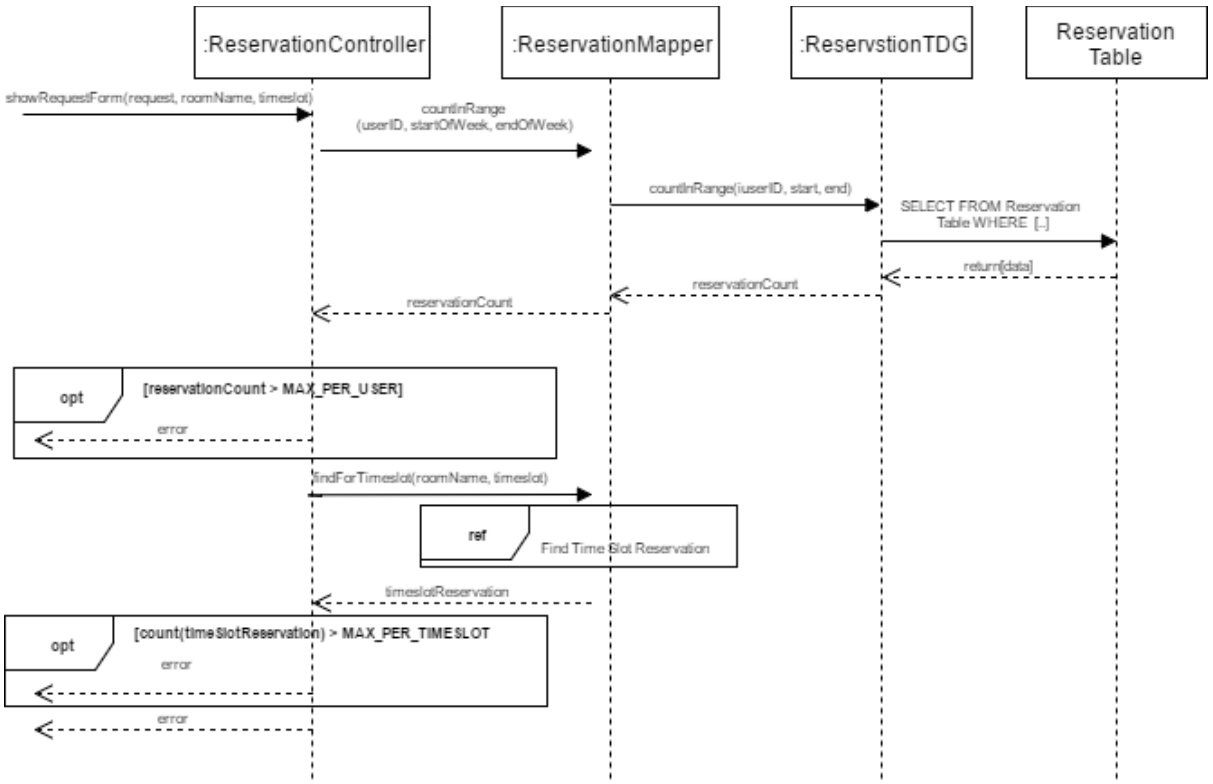

Figure 7: Show Request Form Interaction Diagram

Show Modify Form (2.3.3.1)

For this interaction diagram, we have added

- parameter “id” for find(),
- parameter “request” for showModifyForm()

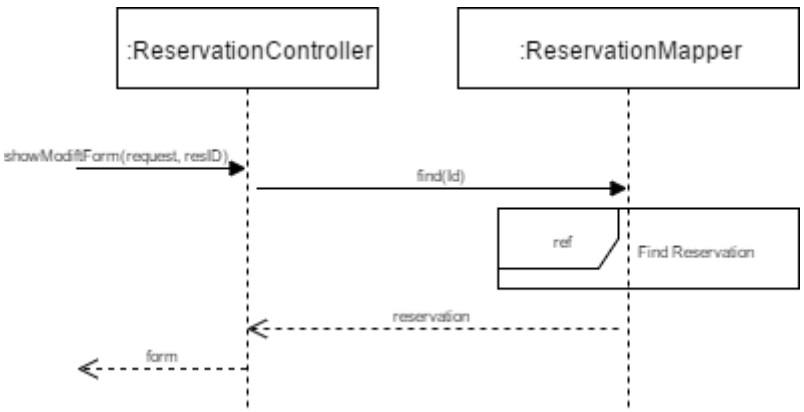

Figure 8: Show Modify Form Interaction Diagram

Modify Reservation (2.3.3.2)

For this interaction diagram, we have added

- parameter “ id, description” for set()

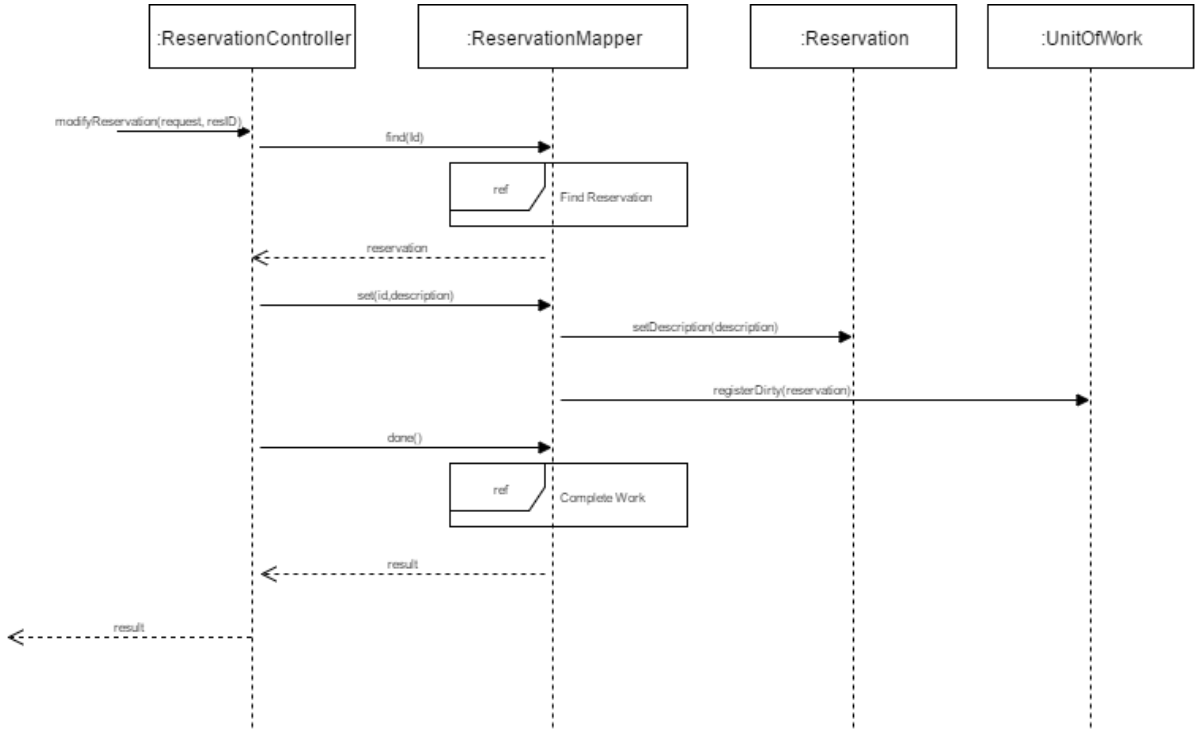

Figure 9: Modify Reservation Interaction Diagram

View Reservation (2.3.4)

For this interaction diagram, we have added

- Parameter “request, id” for viewReservation(),
- Added parameter “id” for find(),

For a better understanding of what is being sent to each class:

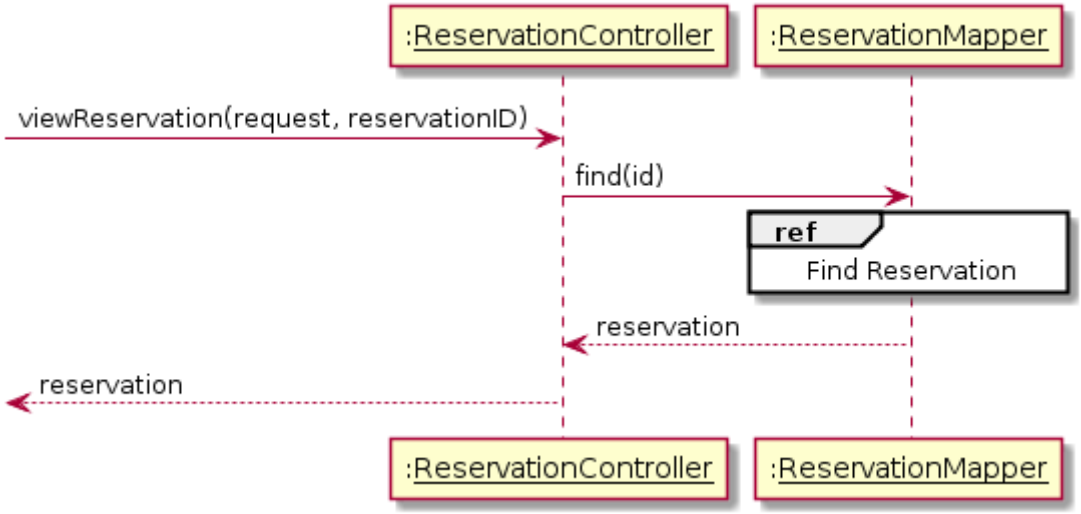

Figure 10: View Reservation Interaction Diagram

View Reservation List (2.3.5)

For this interaction diagram, we have added

- Parameter “request” for viewReservationList()

For a better understanding of the interactions:

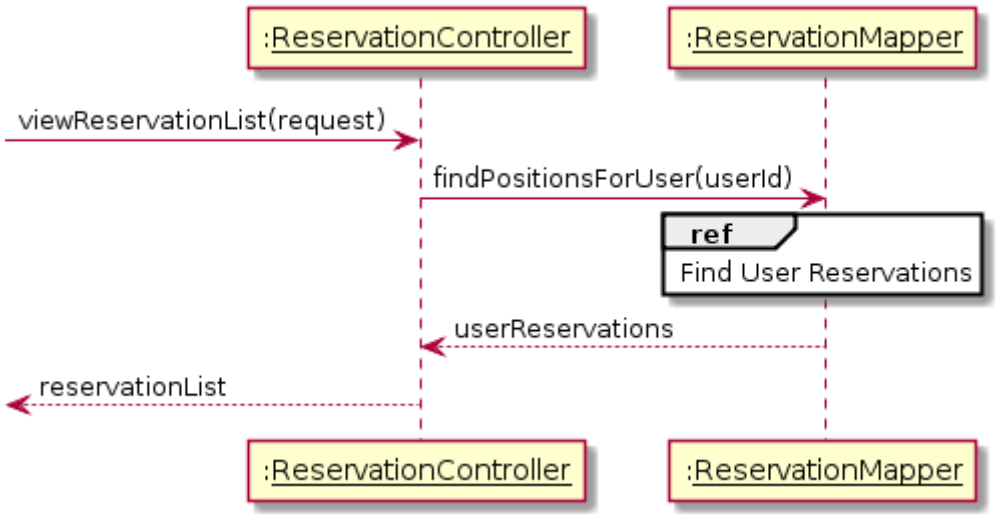

Figure 11: View Reservation List Interaction Diagram

Cancel Reservation (2.3.6)

For this interaction diagram, we have added

- parameter “request” for cancelReservation(),
- parameter “id” for delete(),
- parameter “ id” for find(),

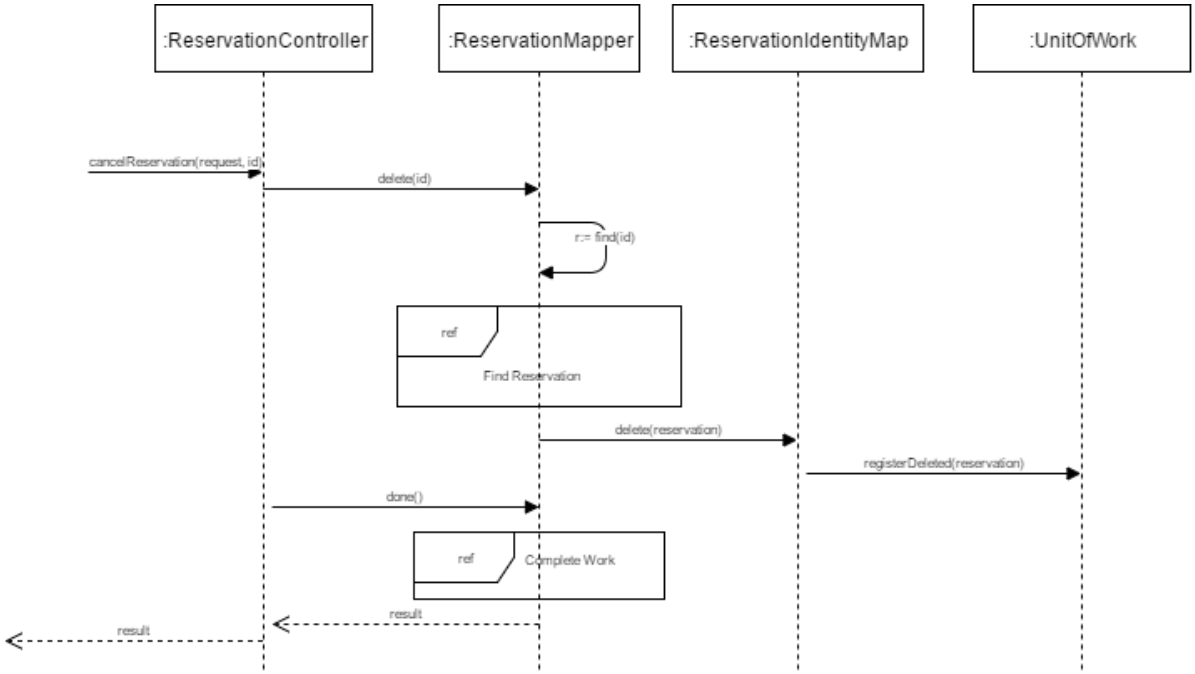

Figure 12: Cancel Reservation Interaction Diagram

Find Active Reservations (2.3.7.1)

For this interaction diagram, we have added

- WHERE DATE(timeslot) = DATE ORDER BY (timeslot, date) for ReservationTDG -> Reservation Table

For a better understanding of the interactions:

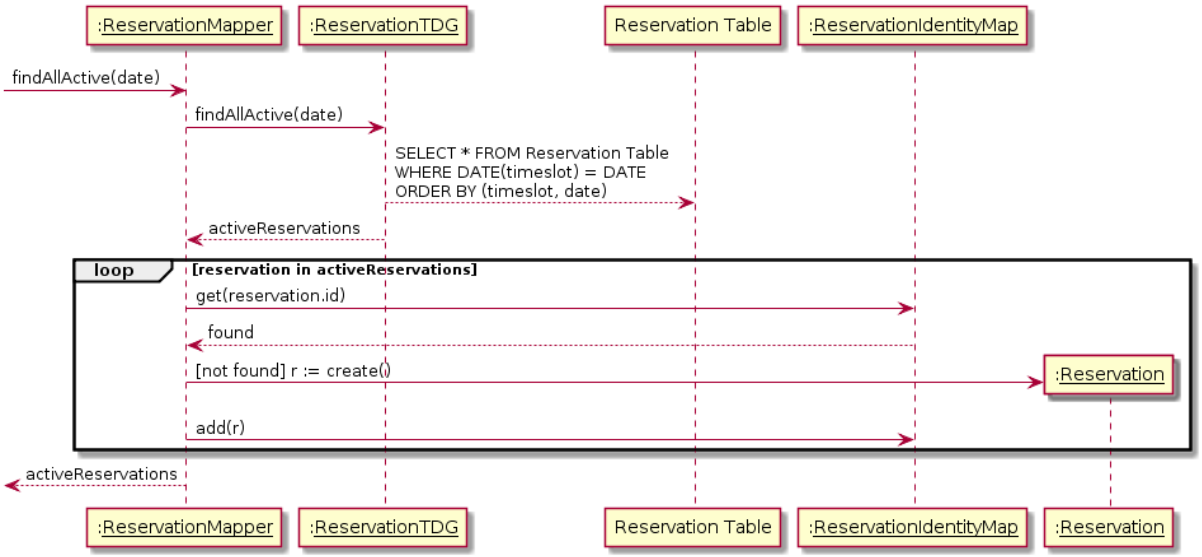

Figure 13: Find Active Reservation Interaction Diagram

Find Reservation (2.3.7.2)

For this interaction diagram, we have added

- parameter “id” for get(...),
- Parameter “ID” for find(...).
- WHERE id = id for ReservationTDG -> Reservation Table

For a better understanding of the interactions:

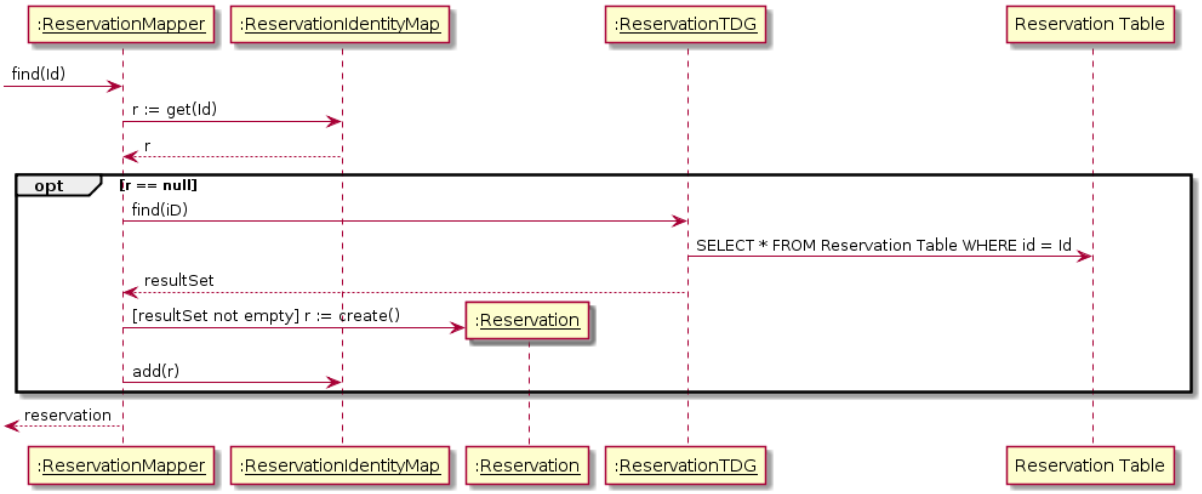

Figure 14: Find Reservation Interaction Diagram

Find Time Slot Reservations (2.3.7.3)

For this interaction diagram, we have added

- parameter “roomName” for findRoomTimesSlot(room, timeslot),
- parameter “roomName, timeslot” for findRoomTimesSlot(...),
- WHERE timeslot = :timeslot & room_name = :roomName for ReservationTDG -> ReservationTable
- ORDER BY id, timeslot, roomName for ReservationTDG -> ReservationTable

For a better understanding of the interactions:

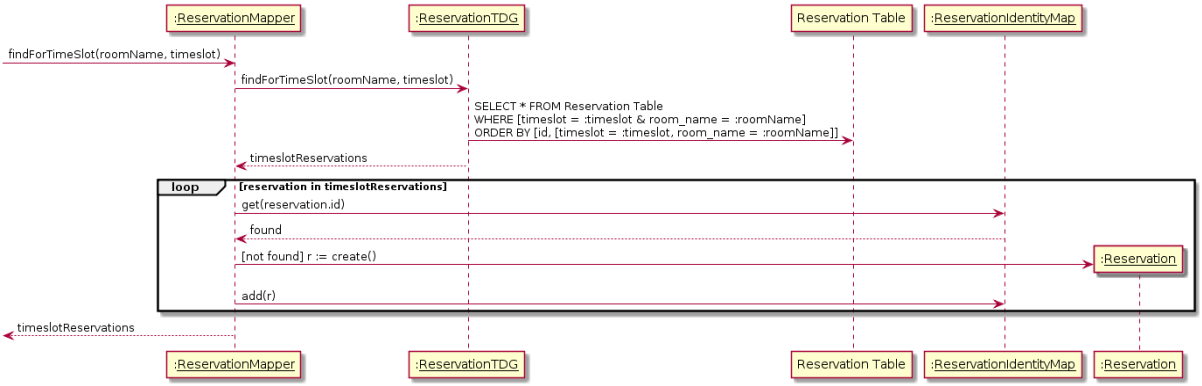

Figure 15: Find Reservations for Time Slot Interaction Diagram

Find User Reservations (2.3.7.4)

For this interaction diagram, we have added

- parameter “userID” for findPositionsForUser(...),
- WHERE user_id = ? AND timeslot >= CURDATE() for ReservationTDG -> ReservationTable,
- ORDER BY timeslot, userIdfor ReservationTDG -> ReservationTable

For a better understanding of the interactions:

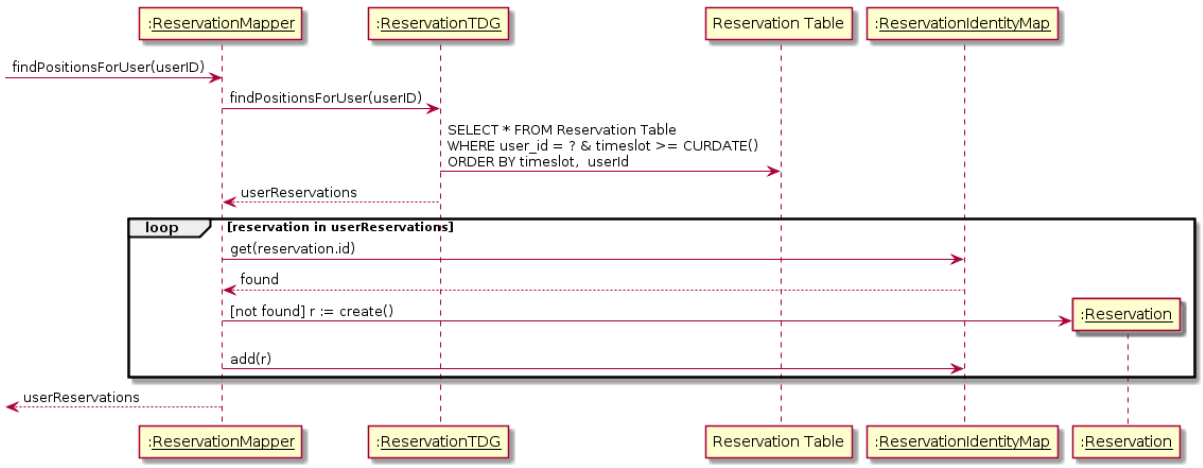

Figure 16: Find User Reservations Interaction Diagram

Commit Mapper Data (2.3.7.5)

For this interaction diagram, we have:

- Added parameter newList to addMany()
- Added VALUES [userId, roomName, timeslot, description, recurId] for ReservationTDG -> Reservation Table
- ReservationTDG returns Id
- Added parameter changedList to updateMany()
- Added SET [description = :description] WHERE [id = :id] for ReservationTDG -> Reservation Table
- Removed updated return from ReservationTDG
- Added parameter deleteList to deleteMany()
- Added DELETE FROM Reservation Table WHERE [id = :id OR (recur_id = :recur_id AND timeslot >= CURDATE())] for ReservationTDG -> Reservation Table
- Removed delete return from ReservationTDG
- Removed the return statement from ReservationMapper

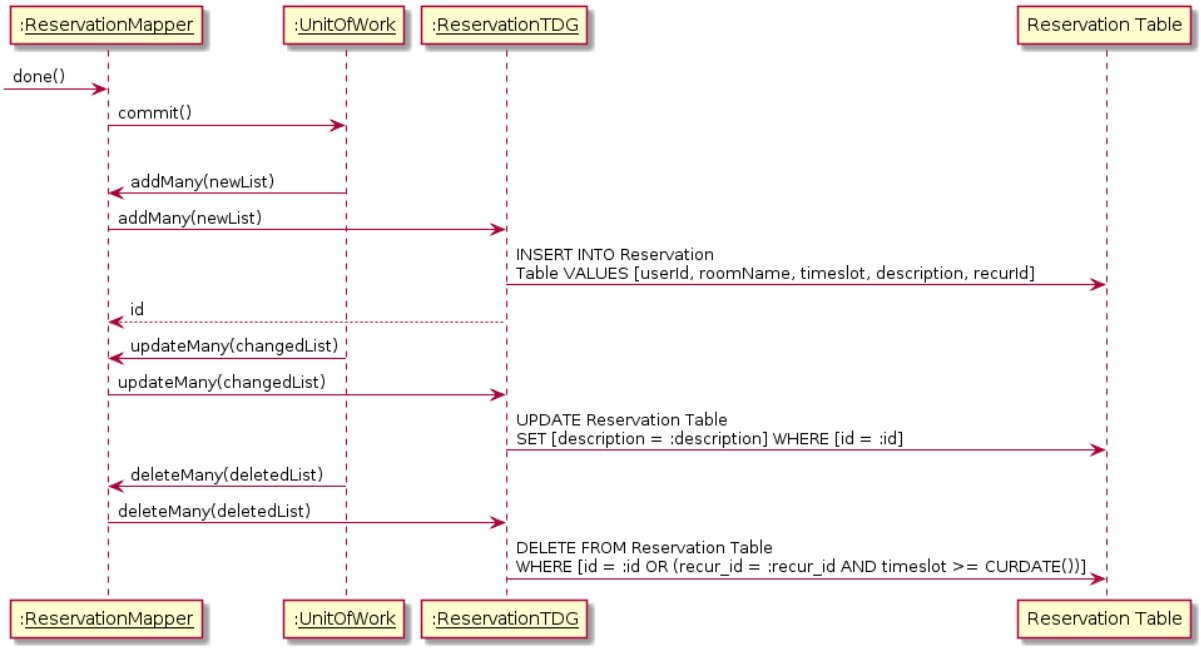

Figure 17: Commit Mapper Data Interaction Diagram

Issue #46 - [Fix] Reservation in past 24hrs are allowed

Author: Amr Mourad

A bug was found in the previous implementation of the code which allowed reservations to be done in the past, as long as they were within the past 24 hours. This was fixed simply by extending the check to be from the current time rather than the previous day.

Issue #45 - [Fix] Functional Requirements

Author: Saif Mahabub

The functional requirements were not explicitly mentioned, just showed the use cases as example of functional requirement. The following table provides concrete description for each functional requirements.

Functional Requirements	Relevant Use Cases
The system will allow the user to securely log in.	UC01
The system will allow the user to log out.	UC02

The system will allow the user to view the status in all the rooms.	UC03
The system will allow the user to reserve a maximum of 10 time slots a week.	UC04
If a user on a wait list obtains the reservation, the system removes the user from all other wait lists with identical time slots.	UC04, UC08
The system will allow only one user at a time to add, modify or cancel a reservation on a specific room.	UC04, UC05, UC08
The system will allow the user to view information regarding an existing reservation.	UC06
The system will allow the user to view a list of reservations associated with their account.	UC07
The system will allow the user to reserve a maximum of 3 hours a week.	UC04
The system will allow the user to request equipment while reserving a room.	UC04
The system will give priority for reservations to users that are capstone student.	UC04, UC05, UC08

Table 3: Functional Requirements Table

Issue #27 - [Fix] Data Class in Class Diagram

Author: Christiano Bianchet

Changes were made to both the **Data package diagram** as well as the **main class diagram** to better represent the system. Some Associations were missing or incorrect, some classes were mislabeled or missing and many unnecessary methods were included in the diagram such as constructors. A brief description on the Singleton and Controller design patterns used was also included.

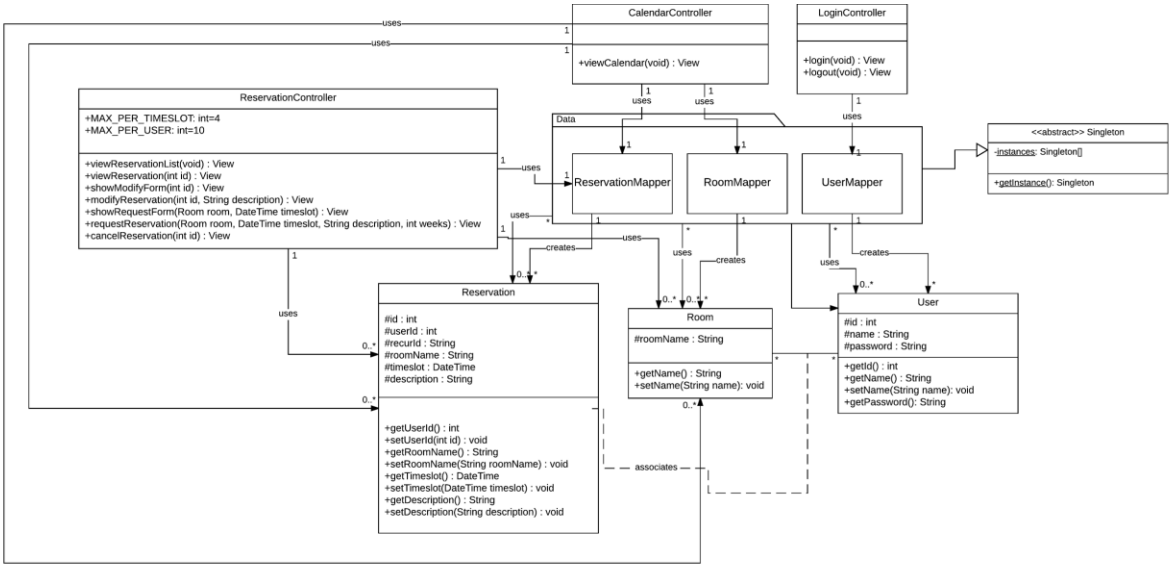

Figure 18: Main Class Diagram

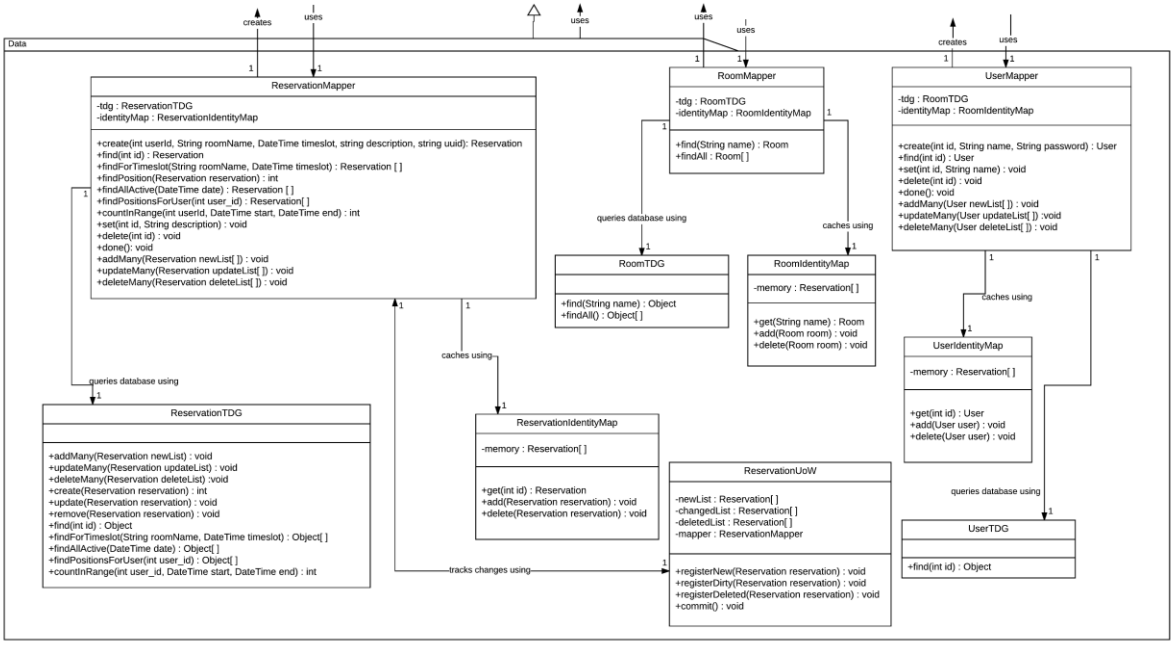

Figure 19: Data Package Diagram

Issue #41 - [Feature] Request equipment in reservation rooms

Author: Amr Mourad

This new functionality follows requirement number 3: making a limited list of equipment available at any given point in time. So even if the room is available for this timeslot, if the

equipment is not available, then the user is put on the waiting list until that equipment becomes available. If the room is already reserved, no change occurs.

The implementation was made to extend the system as simply as possible. First, an equipment table has been added, which holds the list of equipment with their name as well as the amount of available pieces for this equipment. In the reservations table, a foreign key representing the equipment was added. This foreign key defaults to null if no equipment is chosen. The availability of an equipment is done by comparing the number of active reservations for a given timeslot that use this equipment, versus the total amount of this equipment that is available. When a user cancels their reservation, the next in line for the room/timeslot becomes the active entry. This is determined by the position in the waitlist as well as the availability of the equipment. Now if the equipment becomes available, other open rooms with waitlists are verified and the first possible reservation to be promoted will be moved to active.

On the front-end, in the request reservation form, an extra dropdown exists to list all the equipment from which one can be selected.

Pull request #47 includes this requirement: https://github.com/anrchen/soen344/pull/47.

Issue #42 - [Feature] Capstone student priority

Author: Amr Mourad

This new functionality follows requirement number 4, which involves capstone students and their priority on a waiting list for a reservation. If a capstone student attempts to reserve a booked room, they will be put at the top of the waitlist. Any subsequent capstone students to request this room will follow the previous capstone students in the waitlist.

To implement this, an attribute called "*isCapstone*" was added to the user table to identify whether or not a user is a capstone student. When the waiting list is retrieved, the query prioritizes first by whether or not the reservation is waitlisted, and second by whether or not the student is capstone, and then by the date of the reservation. This means if two capstone students are in the waitlist, the priority goes to the one who made the reservation first.

Pull request #48 includes this requirement: https://github.com/anrchen/soen344/pull/48.

Issue #53 - [Improvement] Visual cue showing whether or not a user is capstone

Author: Amr Mourad

An improvement was implemented to the system in order to have a visual cue in the user interface indicating whether or not a user is a capstone student or not. This improved the visibility, and was implemented by adding "*(Capstone)*" next to a username at the top of the screen if they are indeed a capstone student.

Issue #30 - [Fix] Data Model

Author: Christiano Bianchet

An ORM was created to give a visual representation of the one-to-one concept being described in the Data Model section of the SAD. A brief description was also included to re-iterate what was already said in that section.

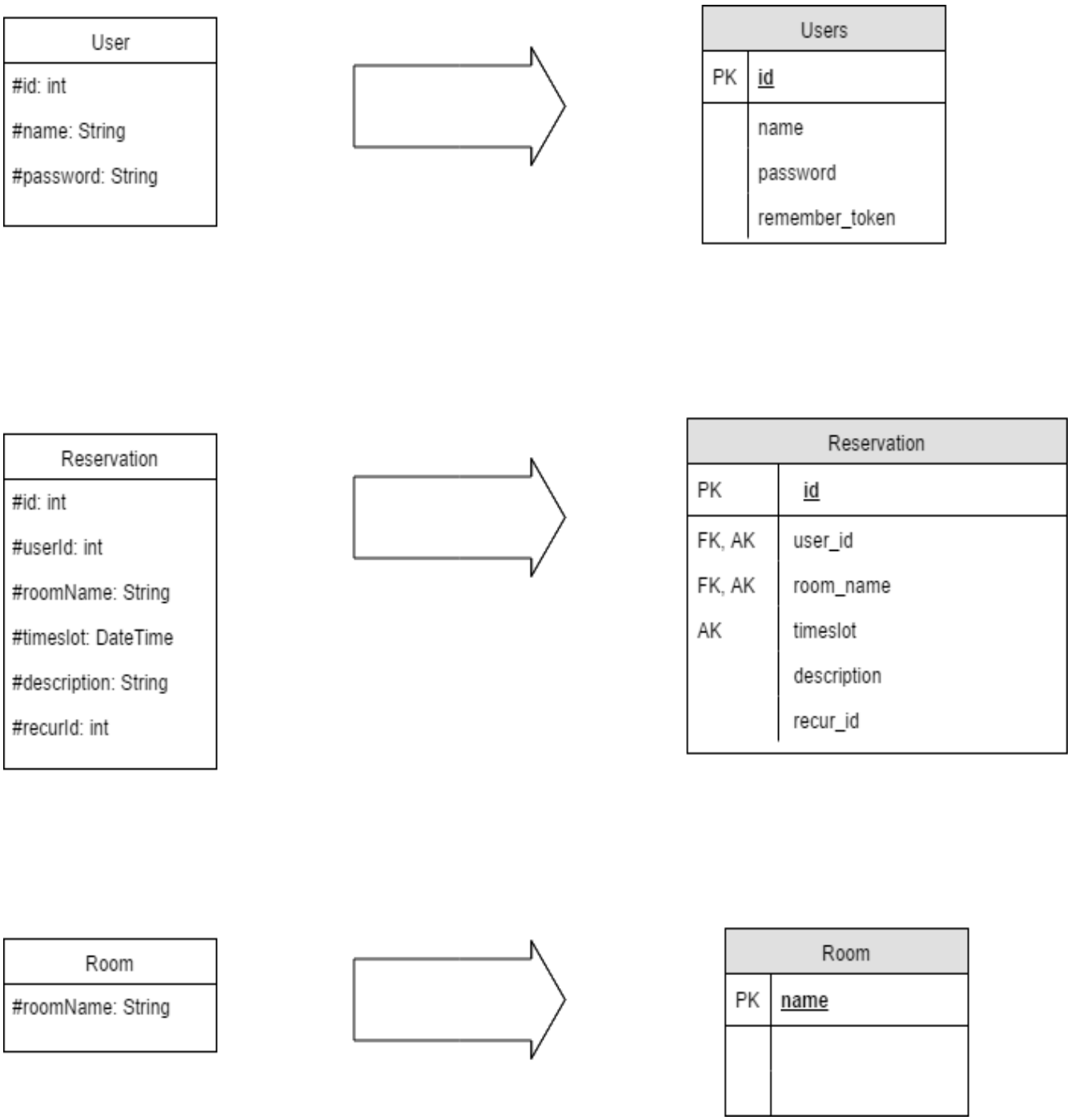

Figure 20: First Iteration Data Model

Once the additional features were added, the appropriate adjustments were made to the ORM.

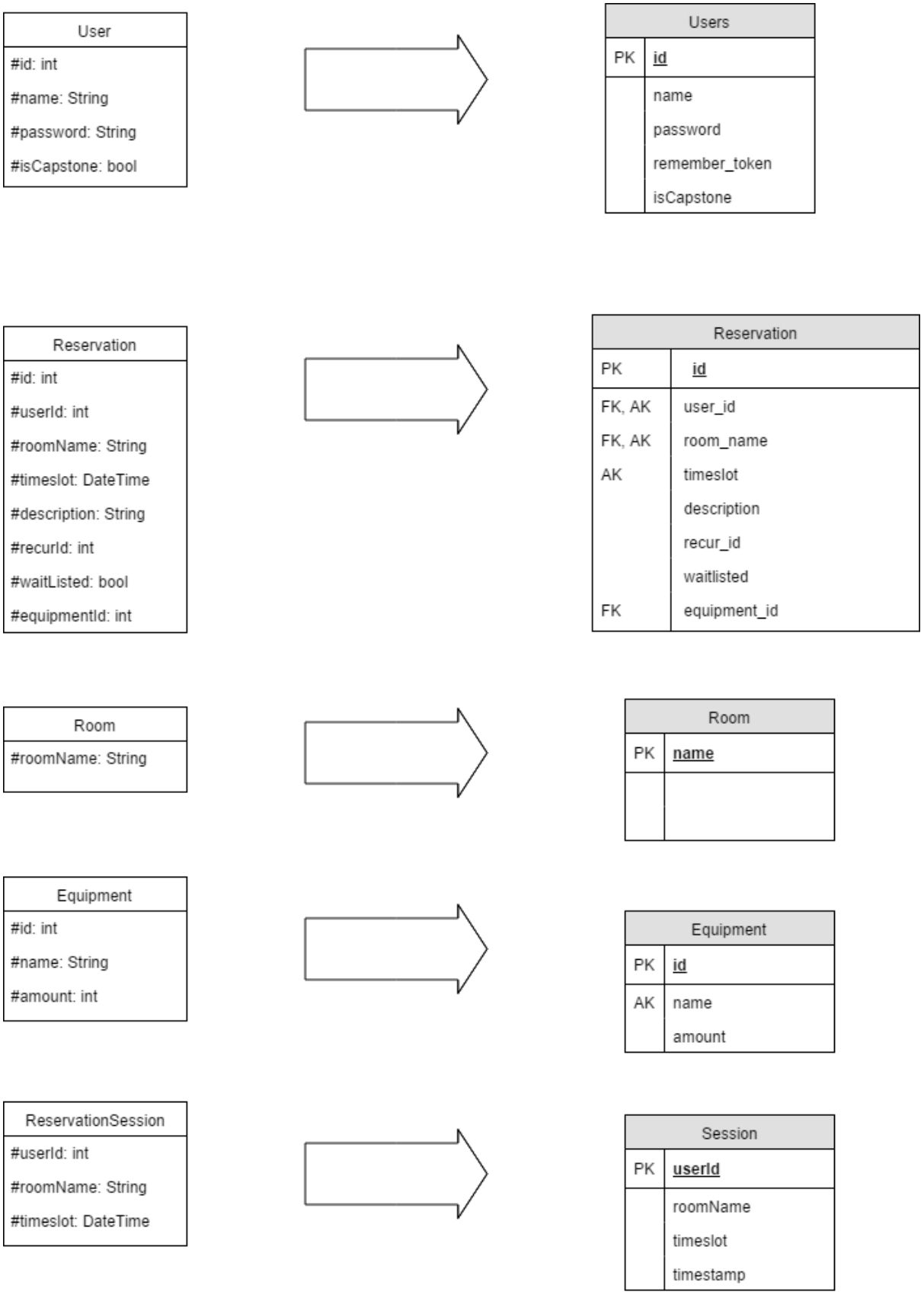

Figure 21: Second Iteration Data Model

Issue #63 - [Fix] Update Interaction Diagrams for New Requirements

Author: Christiano Bianchet

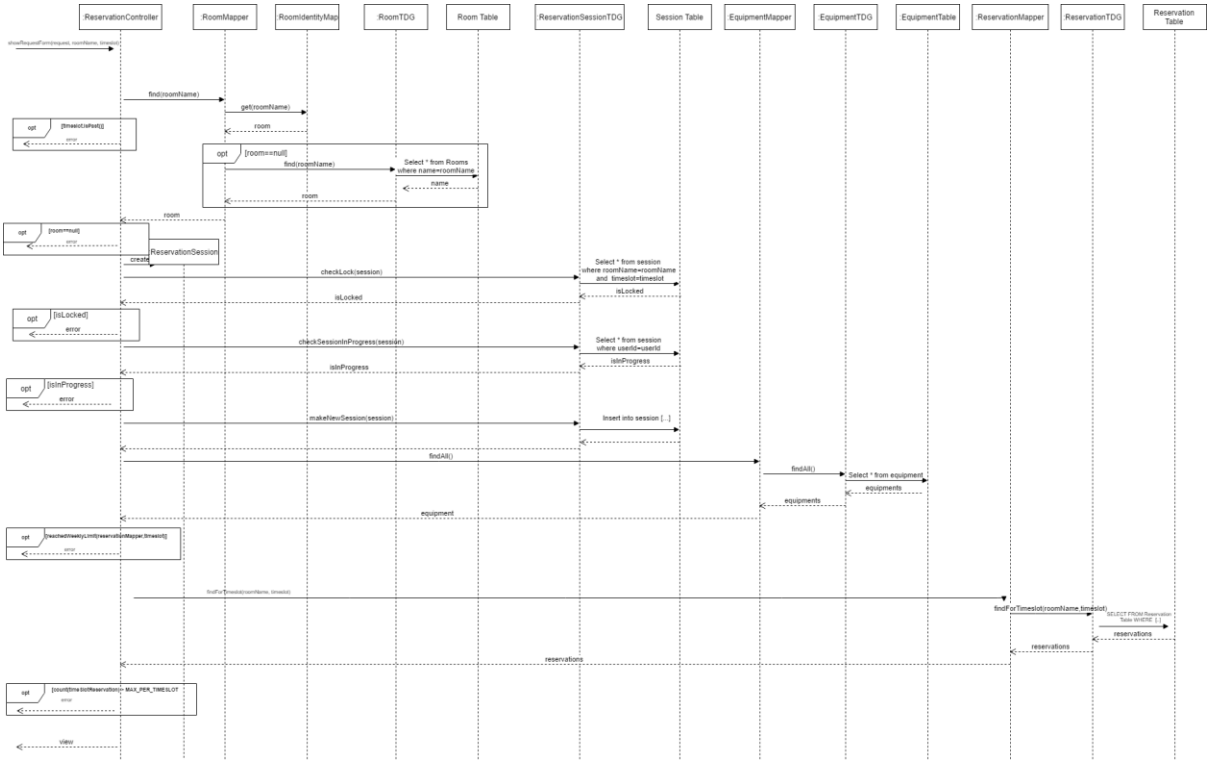

Figure 22: Show Request Form Interaction Diagram

There were a few issues with missing interactions between the ReservationController and The RoomMapper which were added. Also, the new functionality with the EquipmentMapper, EquipmentTDG, ReservationSession and ReservationSessioTDG classes was added to the diagram.

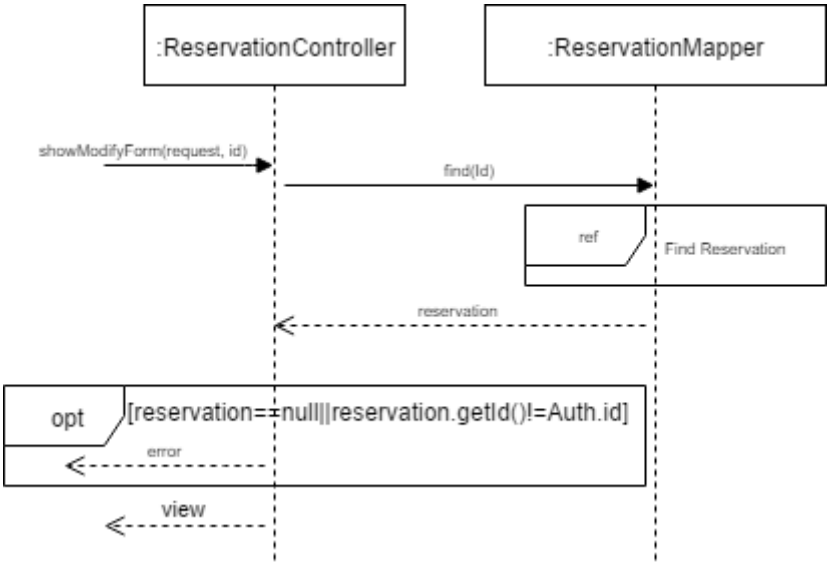

Figure 23: Show Modify Form Interaction Diagram

The addition of the potential error case to the diagram.

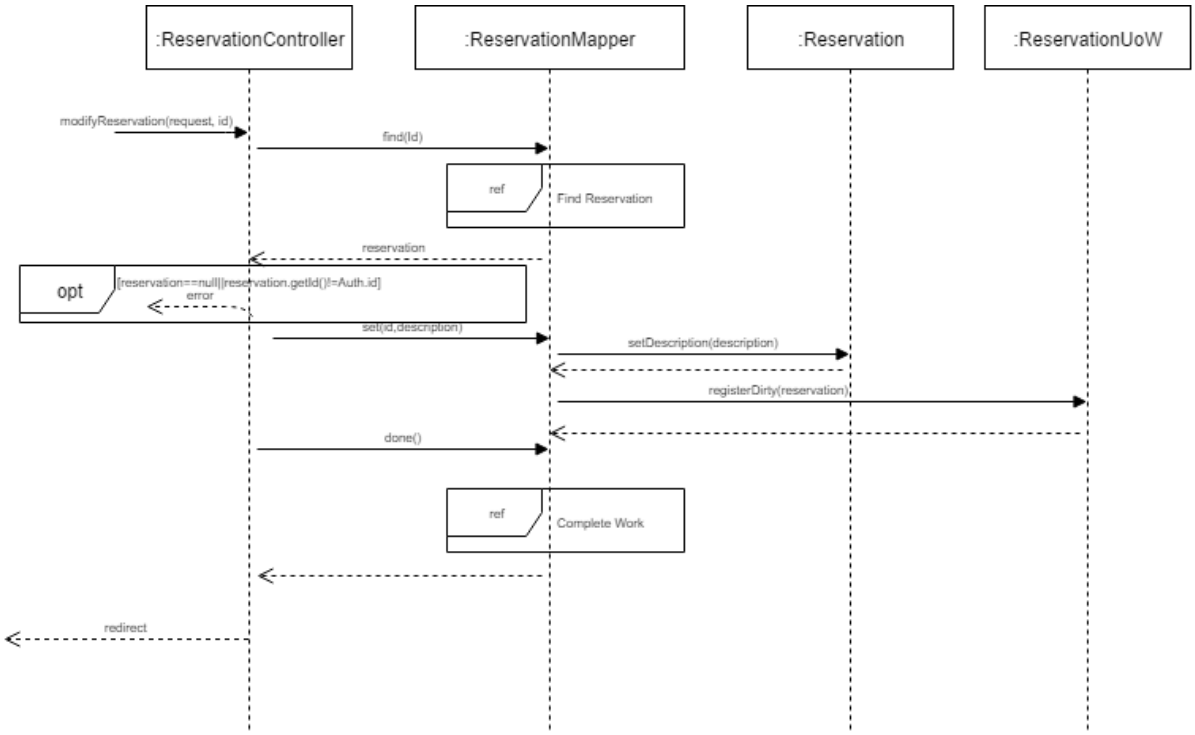

Figure 24: Modify Reservation Interaction Diagram

Addition of the potential error case, as well as using proper naming for ReservationUoW.

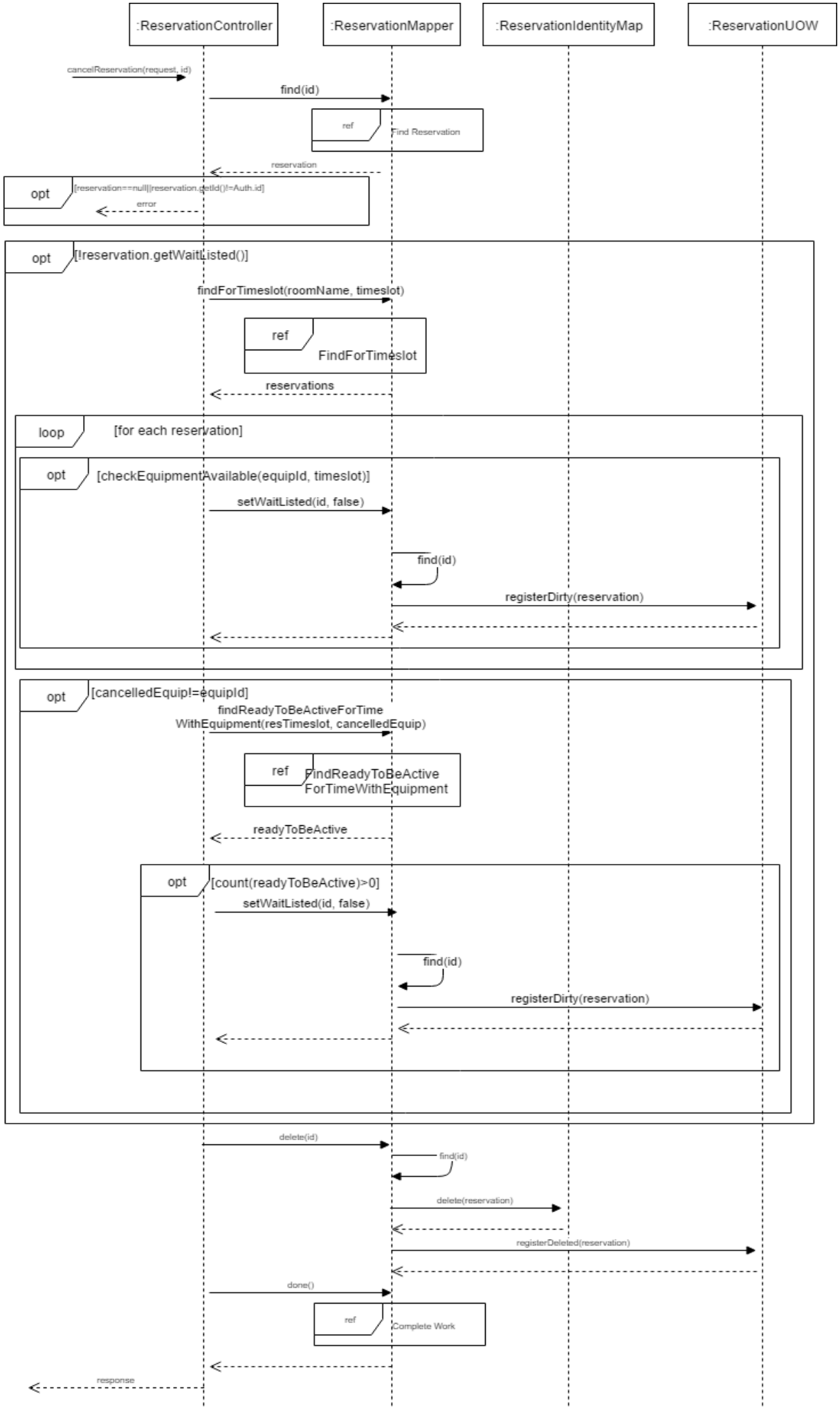
:ReservationController
:ReservationMapper
:ReservationIdentityMap
:ReservationUOW
cancelReservation(request, id)
find(id)
ref
Find Reservation
reservation
opt
[reservation==null||reservation.getId()!=Auth.id]
error
opt
[!reservation.getWaitListed()]
findForTimeslot(roomName, timeslot)
ref
FindForTimeslot
reservations
loop
[for each reservation]
opt
[checkEquipmentAvailable(equipId, timeslot)]
setWaitListed(id, false)
find(id)
registerDirty(reservation)
opt
[cancelledEquip!=equipId]
findReadyToBeActiveForTime
WithEquipment(resTimeslot, cancelledEquip)
ref
FindReadyToBeActive
ForTimeWithEquipment
readyToBeActive
opt
[count(readyToBeActive)>0]
setWaitListed(id, false)
find(id)
registerDirty(reservation)
delete(id)
find(id)
delete(reservation)
registerDeleted(reservation)
done()
ref
Complete Work
response

Figure 25: Cancel Reservation Interaction Diagram

Modified flow to include conditionals dealing with equipment and went more in depth with sequences than previous version.

Issue #60 - [Testing] Documenting and executing acceptance test

Author: Saif Mahabub

The following table provides a detailed acceptance test report with both previous requirements and new requirements:

Step #	Execution Procedure/Input	Expected Result/Output	Passed/Failed
1. Log in	**Log in to the Chronos website with a valid user ID and password.**	**Logged in to Chronos and Calendar page opens.**	**PASS**
2. Request reservation	**Click on an empty timeslot/room.**	**Request a reservation page opens.**	**PASS**
3. Reservation information	**Complete required fields and click Request.**	**Reservation data entered/selected is saved, calendar page opens and a message displays reservation created successfully.**	**PASS**
4. Reservation list	**Click My reservations link.**	**Reservation list page opens.**	**PASS**
5. Reservation status	**Select a reservation and click View.**	**Reservation page opens.**	**PASS**
6. Modify Reservation	**Click Modify.**	**Modify reservation page opens.**	**PASS**
7. Reservation modifications	**Complete required field and click Modify.**	**Reservation data entered is saved and a message displays reservation modified successfully.**	**PASS**
8. Calendar	**Click Calendar link.**	**Calendar page opens.**	**PASS**

9. Calendar information	**Input date and click Jump to date.**	**Calendar page opens on desired date.**	**PASS**
10. Reservation status	**Click on a taken timeslot/room (in green).**	**Reservation page opens.**	**PASS**
11. Reservation cancellation	**Click Cancel this and all recurring.**	**Reservation(s) data is deleted and a message displays reservation cancelled successfully.**	**PASS**

Table 4: Requirements Acceptance Test Report

Alternative flows:

waitlisted, max reservation/week, max recurrence, waitlist full, equipment unavailable, session expired, closing browser before session completed

Step #	Execution Procedure/Input	Expected Result/Output	Passed/Failed
1. Log in	**Log in to the Chronos website with a valid user ID and password.**	**Logged in to Chronos and Calendar page opens.**	**PASS**
2. Request reservation	**Click on a reserved timeslot/room.**	**Request a reservation page opens.**	**PASS**
2.1 Reservation information	**Complete required fields and click Request.**	**Reservation data entered/selected is saved, calendar page opens and a message indicating user has been put into the waitlist with the position number is displayed.**	**PASS**
3. Request reservation (reservation reached for the week)	**Click on an empty or reserved timeslot/room.**	**Request a reservation page opens.**	**PASS**
3.1 Reservation information	**Complete required fields and click Request.**	**Error message displays reservation request limit for the week has been exceeded.**	**PASS**

4. Request reservation (max recurrence reached)	**Click on an empty similar timeslot/room from recurring reservation after 3 weeks.**	**Request a reservation page opens.**	**PASS**
4.1 Reservation information	**Complete required fields and click Request.**	**Error message displays recurring reservation limit has been exceeded.**	**PASS**
5. Request reservation (waitlist full)	**Click on a reserved timeslot/room.**	**Request a reservation page opens.**	**PASS**
5.1 Reservation information	**Complete required fields and click Request.**	**Error message displays waiting list is full.**	**PASS**
6. Request reservation (equipment unavailable)	**Click on an empty timeslot/room.**	**Request a reservation page opens.**	**PASS**
6.1 Reservation information	**Complete required fields and click Request.**	**Error message displays requested equipment is not available and user is put into a waitlist pending availability of the equipment.**	**PASS**
7. Request reservation (session expired)	**Click on an empty timeslot/room.**	**Request a reservation page opens.**	**PASS**
7.1 Reservation session expires	**Wait more than 60 seconds.**	**Message displays session has expired.**	**PASS**
7.2 Reservation information (reservation taken by other user)	**Complete required fields and click Request.**	**Reservation data entered/selected is saved, calendar page opens and a message indicating user has been put into the waitlist with the position number is displayed.**	**PASS**

8. Request Reservation (close browser)	**Click on an empty timeslot/room.**	**Request a reservation page opens.**	**PASS**
8.1 Leaving reservation session	**Close browser.**	**Browser closed.**	**PASS**
8.2 Log in	**Log in to the Chronos website with a valid user ID and password.**	**Logged in to Chronos and Calendar page opens.**	**PASS**
8.3 Request Reservation	**Click on the previous desired timeslot/room**	**Request a reservation page opens.**	**PASS**
8.4 Reservation information	**Complete required fields and click Request.**	**Reservation data entered/selected is saved, calendar page opens and a message displays reservation created successfully.**	**PASS**

Table 5: Alternative Flows Acceptance Test Report

Priority User

Step #	Execution Procedure/Input	Expected Result/Output	Passed/Failed
1. Log in	**Log in to the Chronos website as a capstone student with a valid user ID and password.**	**Logged in to Chronos and Calendar page opens.**	**PASS**
2. Request reservation	**Click on a reserved timeslot/room.**	**Request a reservation page opens.**	**PASS**
3. Reservation information	**Complete required fields and click Request.**	**Reservation data entered/selected is saved. A message indicating user has been put into waitlist position 1 unless another capstone student is in position 1 then the user is put into the following position of the waitlist.**	**PASS**

Table 6: Priority User Acceptance Test Report

Issue #64 - [Fix & Feature] Fixed class diagram & added new components

Author: Philippe Kuret

The class diagram was missing connection, method and variables which did not represent the actual code. A connection between ReservationController and RoomMapper was added. Also, RoomCatalog, UserCatalog were added. The class diagram was also updated to represent the new features added to the system. EquipmentMapper, Equipment & ReservationSession and EquipmentCatalog were added. A new association was made between Equipment, Room and User.

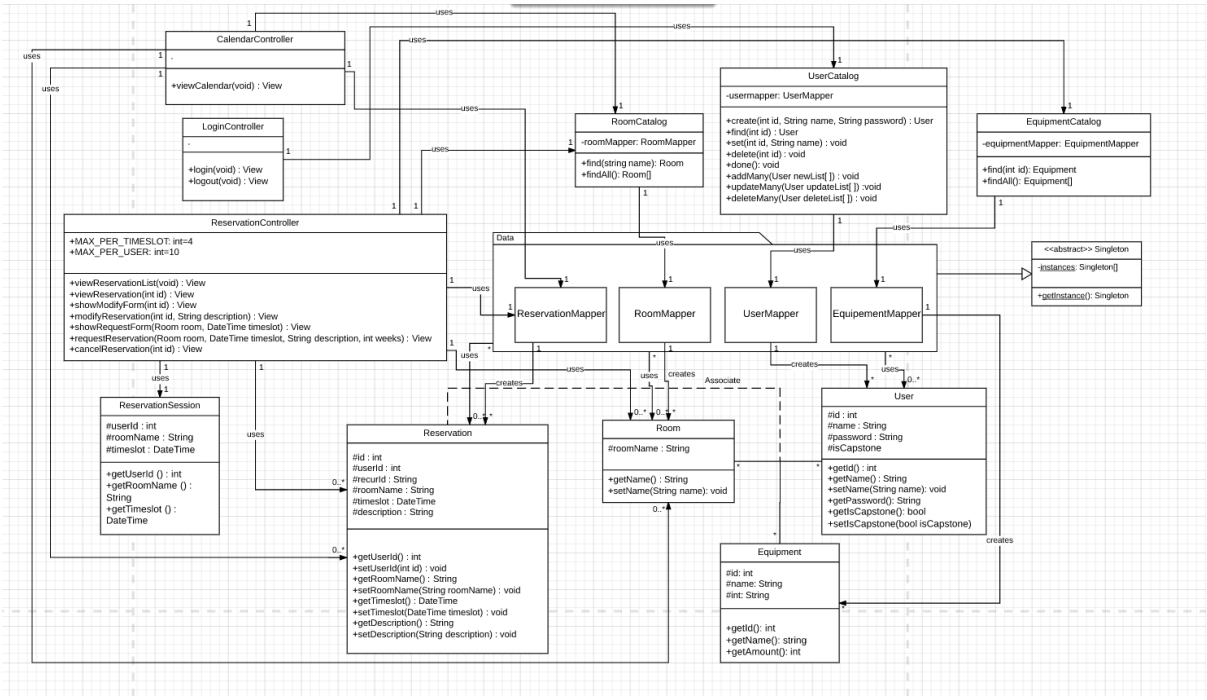

Figure 26: Class Diagram (Main)

For this class diagram, an Equipment Mapper, EquipmentTDG and EquipmentIdentity map was added and their respective association were added.

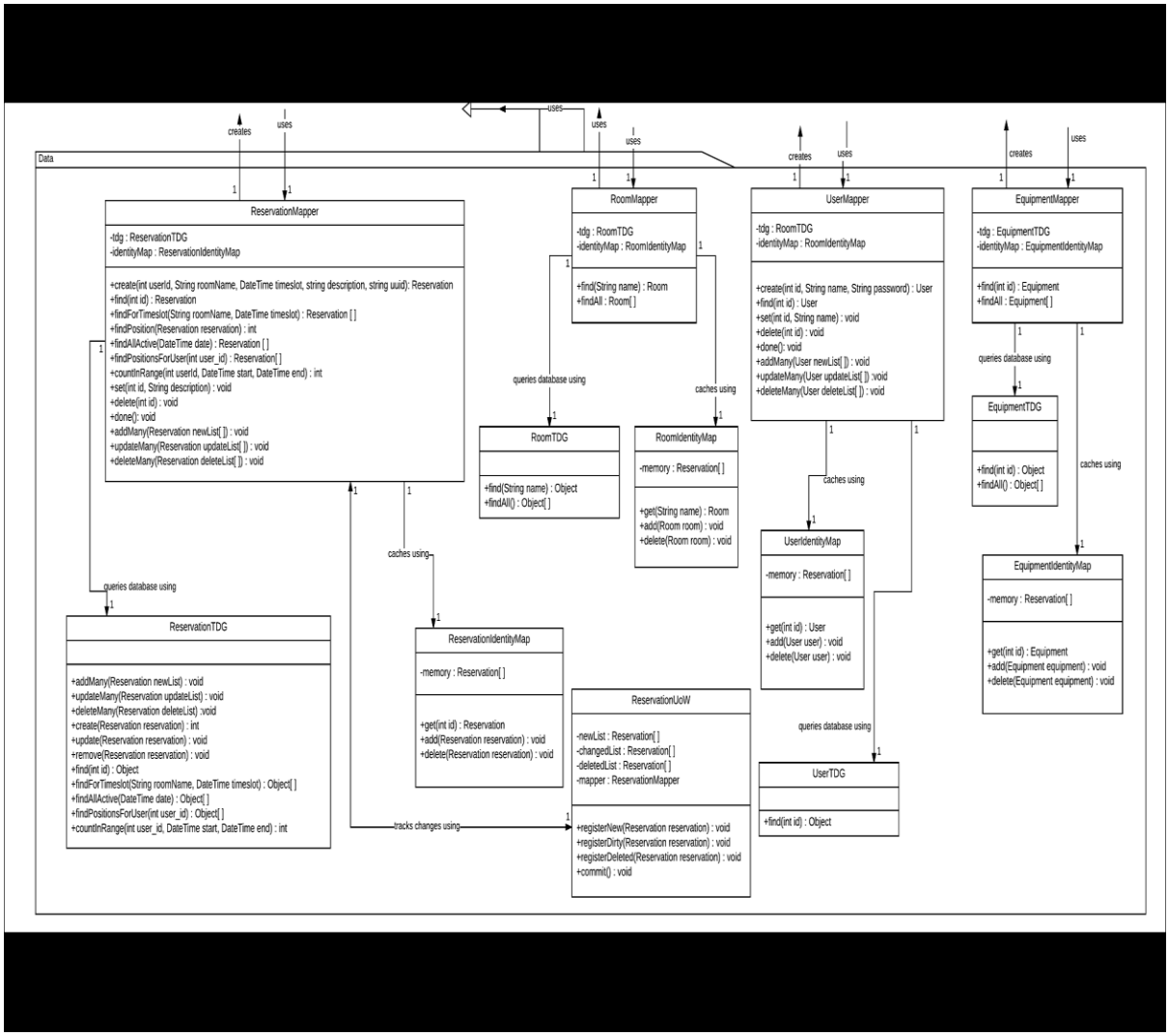

Figure 27: Class Diagram (Data package)

Issue #43 - [Fix & Feature] Added missing classes & new requirement classes

Author: Philippe Kuret

The new classes UserCatalog and RoomCatalog were added to the code to better represent the Domain model. Their purpose is to act as a communicator between the mapper and the controller. Additonnaly, the classes Equipment and EquipmentCatalog were added as part of the new requirement.

Issue #54 - [Fix] Refactored Unit of Work

Author: Adriel Fabella

A unit of work was added for the User and reservation Domain and functionality. Both UoW classes have the functionality of keeping track of in-memory updates and sending these updates to the database in one call. The UoW classes implement arraylists to store and hold data. The data consists of any update, creation or manipulation towards the database. Once the database call has been, each UoW is to empty out the arraylists to keep track of all the future updates to be made. The **User UoW** has the functionality of keeping track of database call towards the User table or domain, and the **Reservation UoW** has the role of database calls towards any manipulation in the Reservation table.

Issue #56 - [Testing] Document the unit tests

Author: Youness Tawil, Adriel Fabella

The testing report covers testing on the Table Data Gateway files of the application. Unit tests were made to ensure that there is a proper communication between the third party systems, which is the MySQL database. Also, one test was made for the Identity Map to ensure that the system stores the data that is received from the database.

Method Coverage

Class Name	Method Coverage (%)
UserTDG.php	29% (2/7)
RoomTDG.php	50% (½)
ReservationSessionTDG.php	75% (¾)
EquipmentTDG.php	50% (½)
EquipmentIdentityMap	100% (3/3)

Table 7: Unit Tests Method Coverage

Class **RoomTDG.php**

TEST CASE 1: Method**: find (string room)**		
Tests that a room is successfully retrieved upon the input of a valid room number		
Input	Expected Output	Result
H-901	An object with the room number	Pass
TEST CASE 2: Method**: find (string room)**		
Tests that a room is successfully retrieved upon the input of an invalid room number		
Input	Expected Output	Result
H11111	Nothing is to be returned (null)	Pass
TEST CASE 3: Method**: find (string room)**		
Tests that a room is successfully retrieved upon the input of an empty string		
Input	Expected Output	Result
Null	Nothing is to be returned (null)	Pass

Table 8: RoomTDG Test Cases

Class **ReservationSessionTDG.php**

TEST CASE 1: Method**: makeNewSession (ReservationSession $session)**		
Tests that a session has been successfully added from the sessions table		
Input	Expected Output	Result
('10000009', 'H-903', '2017-03-26 19:00:00')	The corresponding session has been added to the session table	Pass
TEST CASE 2: Method**: endSession (ReservationSession $session)**		
Tests that a session has been successfully deleted from the sessions table		
Input	Expected Output	Result
('10000009', 'H-903', '2017-03-26 19:00:00')	The corresponding session has been removed from the sessions table	Pass
TEST CASE 3: Method**: checkSessionInProgress (ReservationSession $session)**		
Tests that the session that was added to the table is still active and still exists in the table		
Input	Expected Output	Result
('10000009', 'H-903', '2017-03-26 19:00:00')	Returns **true** indicating that the **session** still exists in the table	Pass

Table 9: ReservationSessionTDG Test Cases

Class **EquipmentTDG.php**

TEST CASE 1: Method**: find (int $equipment_id)**		
Tests that an equipment object (name, amount) is successfully retrieved based on fetching its id		
Input	Expected Output	Result
1	An equipment object will be returned	Pass
TEST CASE 2: Method**: find (int $equipment_id)**		
Tests that upon inserting an invalid equipment id, no object is returned		
Input	Expected Output	Result
null	Nothing is to be returned (null)	Pass

Table 10: EquipmentTDG Test Cases

Class **UserTDG.php**

TEST CASE 1: Method**: create(User $user)**		
The test ensured that the user was successfully inserted in the User table. Once the user is created we retrieve it by ID and compare it to the original user.		
Input	Expected Output	Result
new user(123,"test","password", false)	A user is added to the User table	
TEST CASE 2: Method**: add(Equipment $equipment)**		

The Test is successful when a user's name is updated in the User table. Create a new user then update the user name by ID. Retrieve the user by ID from the database and compare to the data passed to the update method.		
Input	Expected Output	Result
new user(1234,"testUpdate","password", false)	The user is Updated in the User Table	

Table 11: UserTDG Test Cases

Class **EquipmentIdentityMap.php**

TEST CASE 1: Method**: get(int $id))**		
The test is successful when the added data is retrieved correctly or null is returned if no data exists. . First add an equipment to memory then use the get method to retrieve it by ID and compare. Second use the get method to retrieve an equipment that is not in memory and compare with null.		
Input	Expected Output	Result
654321	An equipment Object	
888888	null	
TEST CASE 2: Method**: add(Equipment $equipment)**		
The Test is successful when an equipment is added to memory. Add an equipment to memory then retrieve it by Id and compare.		
Input	Expected Output	Result
new Equipment($id, "test",123456)	The equipment object is added to memory	
TEST CASE 2: Method**: delete(Equipment $equipment**		
The Test is successful when an equipment is deleted from memory. Add an equipment to memory then remove it using the delete method. Try getting the removed equipment by using the get method and compare the result with null.		

Input	Expected Output	Result
654321	The equipment object is deleted from memory	

Table 12: EquipmentIdentityMap Test Cases

2. PROJECT FORCE AWAKENS

The team spent two sprints, each consisting of two weeks, completing tasks for this project. The project was dropped in the middle of the second sprint (after a week) due to the selection of a different project and thus the tasks were dropped. Meetings were held every Tuesday afternoon.

Sprint 1	January 25 - February 8
Sprint 2	February 9 - February 15

Table 13: Project Force Awakens Schedule

Changelog table

The following table outlines the changes made to project *Star Wars* sorted by completion date, and the issue number they refer to. Each issue links to a more detailed explanation.

Sprint	Date completed	Change Type	Change Log Entry	Reference	Author
Sprint 1	January 31	Fix	Fix Log-in "*session_start"* call when session has already started	Issue #12	Amr Mourad
Sprint 1	February 2	Improvement	Incorrect explanations on the architectural style	Issue #3	Christiano Bianchet
Sprint 1	February 2	Fix	Use cases should not include functionality	Issue #9	Alexander Rosser
Sprint 1	February 2	Fix	Inexistence of database field "busy"	Issue #14	Amr Mourad
Sprint 1	February 3	Fix	Fix Home page "required_once" file loading warnings	Issue #13	Amr Mourad
Sprint 1	February 4	Improvement	Missing Object-Relational	Issue #2	Amr Mourad

			Mapping		
Sprint 1	February 4	Fix	Inconsistencies in SSD, incorrect system self-returning operations	Issue #7	Saif Mahabub
Sprint 1	February 5	Fix	Inconsistencies in Interaction Diagrams	Issue #5	Youness Tawil
Sprint 1	February 7	Fix	Inconsistencies in Domain Model	Issue #6	Philippe Kuret
Sprint 1	February 7	Fix	Inconsistencies in Use Case Model	Issue #1	Alexander Rosser

Table 14: Force Awakens Changelog Table

Issue #12, #13, #14 - [Fix] Project Setup Fixes

Author: Amr Mourad

Code fixes were applied in order to have the splash page, home page, and reservation page work smoothly after finding many errors. Multiple errors were found across the project related to redeclared classes as well as files loaded multiple times. To avoid these issues and limit file loading to a single instance, files were referred to using *require_once*.

There were also warnings/errors across the project relative to PHP's *session_start()* being called even when the session is already started. This was avoided by conditionally starting the session only if it hasn't already started.

There were a few constructors called with the wrong number of parameters or no parameters at all when they did indeed require parameters. To avoid facing errors, default parameter values were set within these constructors' signatures.

Some undefined session values were assigned to variables and used which resulted in warnings and errors. To work around this, a null value was assigned to the variable whenever the session value is undefined, and any code using the undefined values are to be called conditionally.

Parts of the room reservation code executed queries that update and retrieve a non-existent database field called “busy” in the “room” table. This field is supposed to ensure that when a user is attempting to reserve a given room, other users are locked out of reserving this same room in order to avoid reservation conflicts. Its omission caused the reservation logic to fail. Adding the busy field to the room table with a default value of 0 (false) eliminated the issue. These maintenance fixes refer to issues #12, #13, and #14 in the github repository and corresponding commits can be found on the “fixes” branch.

Issue #3 - [Improvement] Incorrect explanations on the architectural style

Author: Christiano Bianchet

This issue involved reviewing the entirety of the software architecture document. While reviewing, it became evident that there were inconsistencies between the diagrams included in the document and the explanations of the underlying architecture of the system. One such example is their description of their layered architecture as open (transparent), such that any layer can access any other layer. However, according to their diagrams, the system was actually closed (opaque) as only adjacent layers could communicate with one another. This can be found in the Architectural Representation section of their SAD.

The document goes on to define and explain the purpose of the Unit of Work in their architecture. The explanation required some rewriting as it was written in an unclear manner with many redundant statements.

Likewise, the Logical View section of the document contained much of the same information as the Architectural Representation section. It re-describes the layered architecture once again and each of the subsystems. It also incorrectly refers to layers as tiers under the assumption that they are synonymous. The system could have potentially been described as a two tier system and a form of the client server architecture, but this is never described nor mentioned. Rewriting of this section was required.

Note: These modifications were made under the assumption that the class diagram and other diagrams were valid and representative of the actual system.

Issue #9 - [Fix] Use cases should not include functionality

Author: Alexander Rosser

Overall the displayed use cases in the SRS document were correct, with the exception that use cases 2 to 6 referenced functional aspects of the application. An example can be seen in UC3 with the steps referencing the “My Profile” button:

UC3	**Update Password**
Actor	Student
Goal	Change current password for authentication
Summary	Student accesses their profile and enters current and desired password for change
Preconditions	• Student has been authenticated
Steps	• Student selects “My Profile” • System displays profile information • Student enters current password • Student enters desired password • Student submits form • System verifies if old and new passwords are valid • System changes student password • System displays confirmation
Postconditions	• Student password changed

Table 15: Update Password Use Case

To solve the issue, all references to functional aspects were removed and replaced with new steps describing the action without functional aspects. Another issue with the Use cases was discrepancies between the Use Case Model and the Use Cases. Some uses cases presented in the model were not found elsewhere in the SRS and some use cases presented were not found in the Use Case Model. Further details are presented in issue #1.

Issue #2 - [Improvement] Missing Object-Relational Mapping

Author: Amr Mourad

An object relational mapping model was created to reflect the code and the database tables.

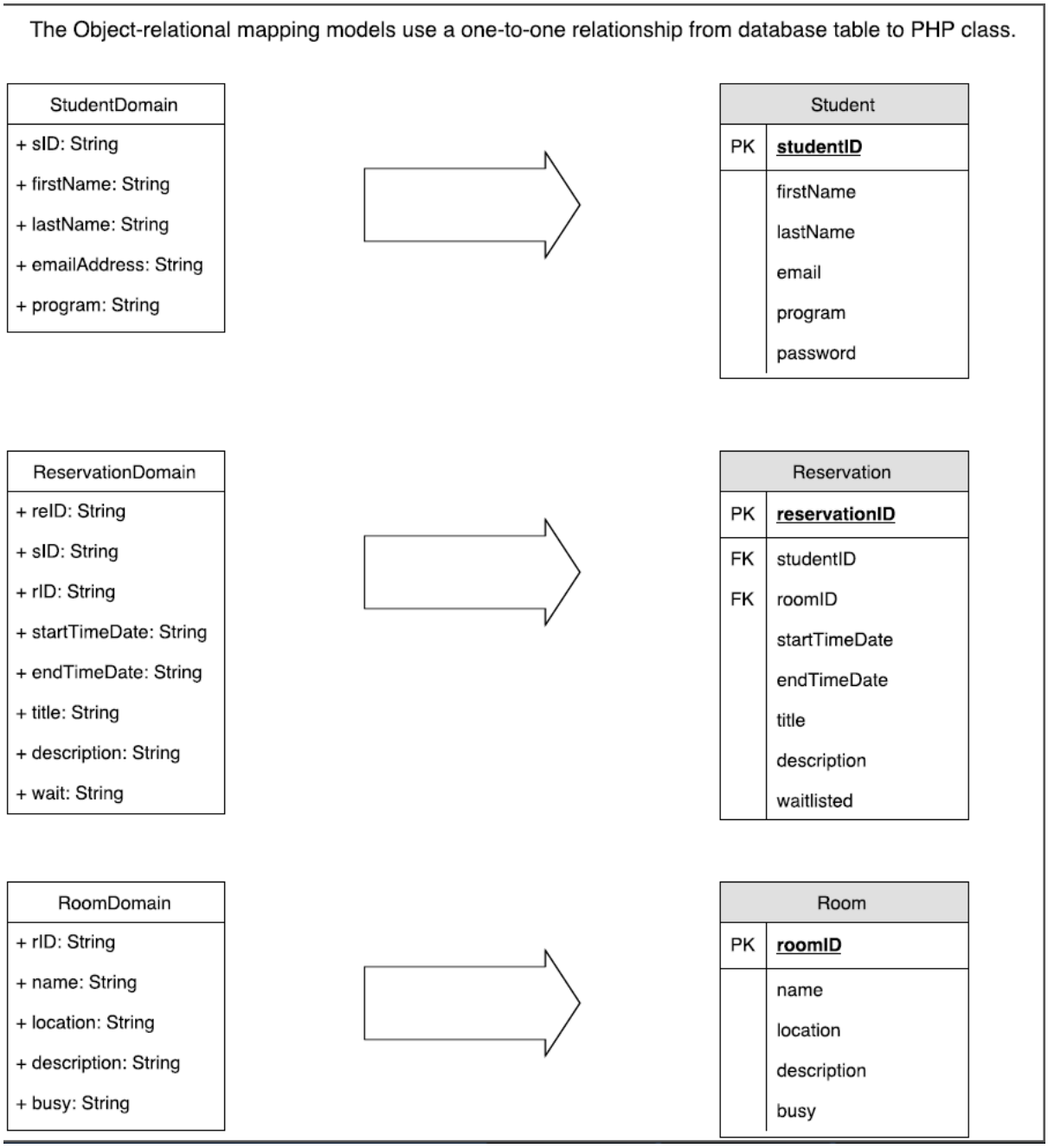

Figure 28: ORM Diagram

Issue #7 - [Fix] Inconsistencies in SSD, incorrect system self-returning operations

Author: Saif Mahabub

In general, the system sequence diagrams had various problems such as the system returning operations to itself, naming convention of the system operations were inconsistent

with the code as well as the operation contracts and two SSD's (Authentication and Change Profile Details) were not following a critical scenario therefore it was needed to be removed.

For the SSD Create Reservation (Booking Available), the naming was changed to Create Reservation Success Scenario as the previous title created confusion and implied verification of booking which is not required for a success scenario. A loop was required in order to add more than one reservation in one session and the verification for booking availability was removed as it is not a requirement. The SSD's below demonstrate the previous and modified version.

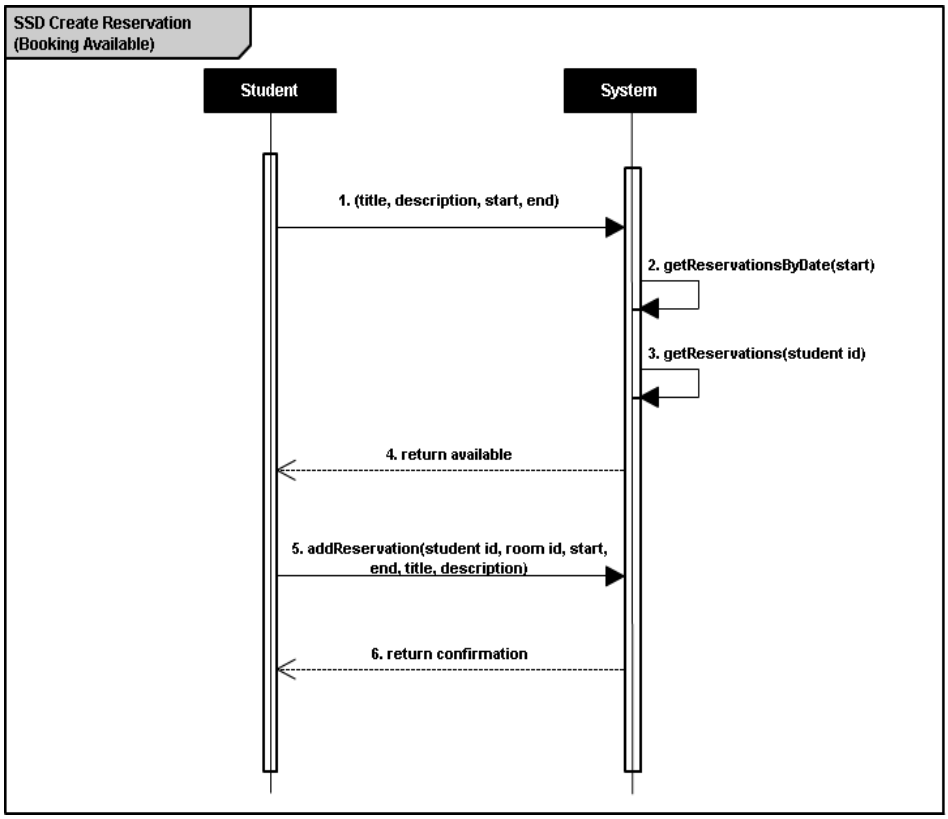

Figure 29: Create Reservation SSD (PREVIOUS)

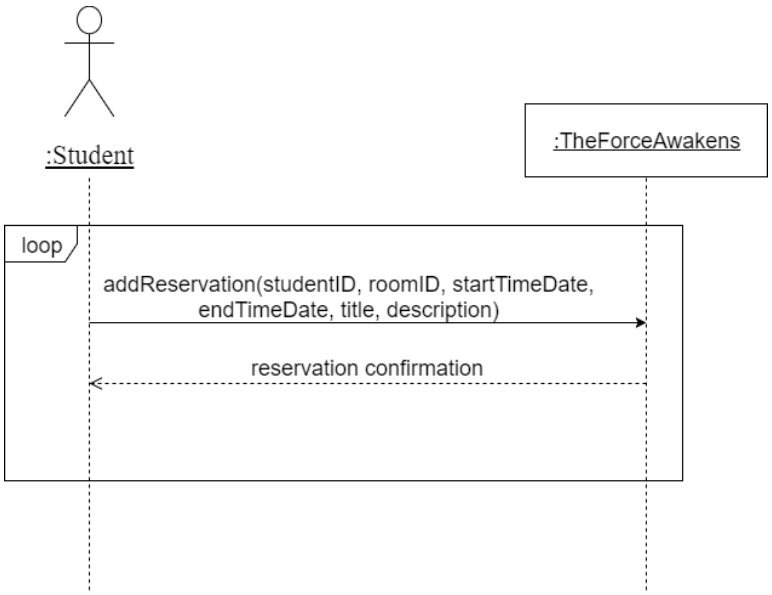

Figure 30: Create Reservation SSD (NEW)

For the SSD Cancel Reservation, the first operation is wrong as it should be from user to system and not the opposite way. The first operation asking to display reservations was removed as it is assumed that the student is already in the page showing all reservations with the defined SSD and a loop was needed for dropping a reservation more than once in one session. The figures below show the previous/new SSD's.

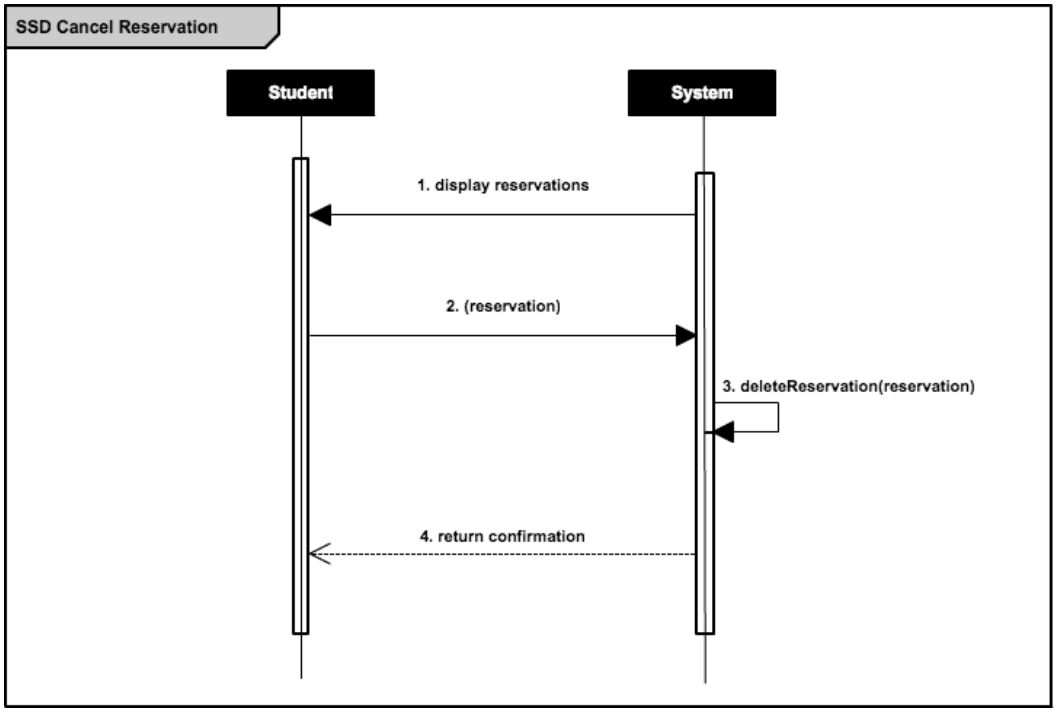

Figure 31: Cancel Reservation SSD (PREVIOUS)

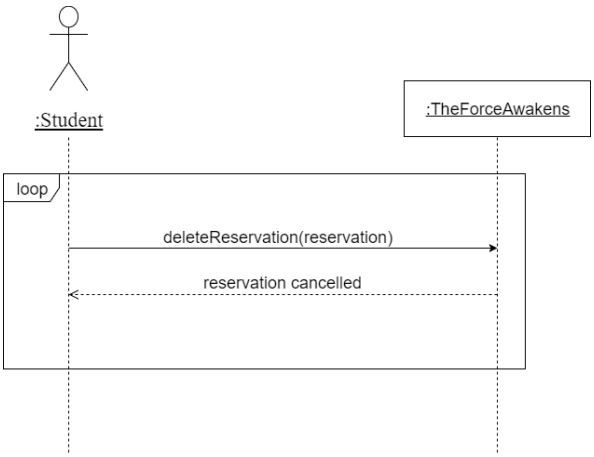

Figure 32: Cancel Reservation SSD (NEW)

For the SSD Modify Reservation, the initial call to display reservations was removed as it is assumed that the user is already in the modification page and a loop was added so that multiple modifications can be done in one session. The following figure 5 and figure 6 displays the previous and modified versions of the SSD's.

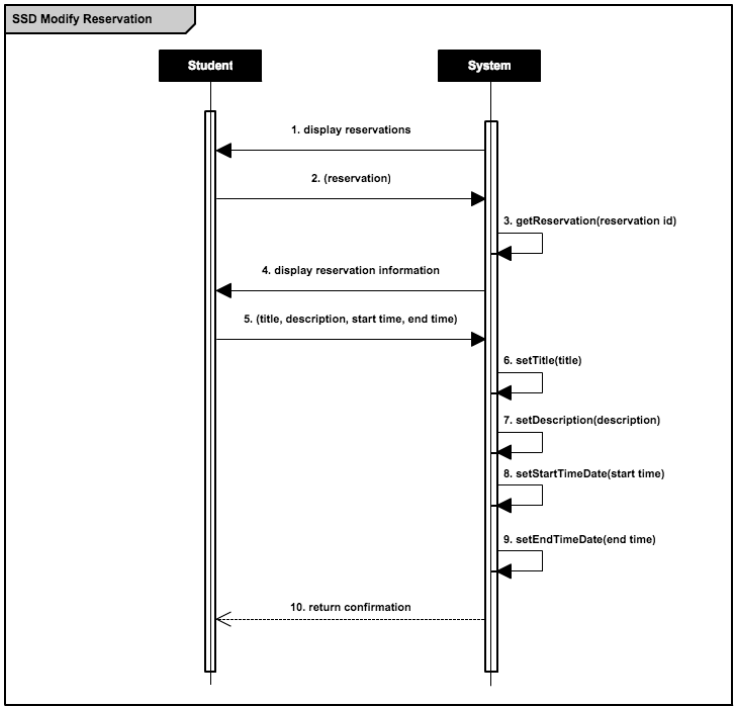

Figure 33: Modify Reservation SSD (PREVIOUS)

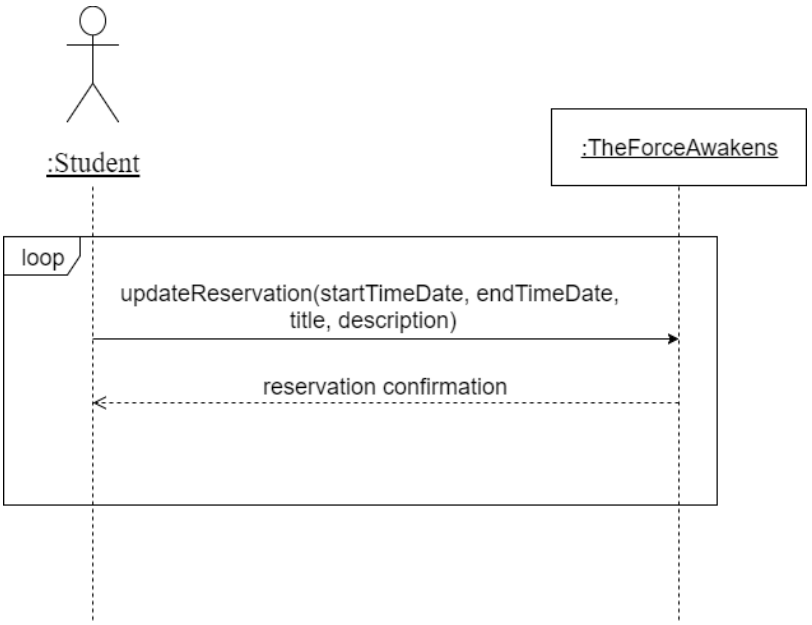

Figure 34: Modify Reservation SSD (NEW)

Issue #5 - [Fix] Inconsistencies in Interaction Diagrams

Author: Youness Tawil

Issues were found in all the interaction diagrams in the SAD documentation. The messages calls in the charts are not reflected in the code. Some classes such as UnitofWork is never used but yet referenced in every diagram. The messages names used in the charts do not match the ones in the code. A list of issues for every diagram is added below.

Interaction Diagrams	Issues
Figure 7: Adding a Reservation Sequence Diagram	• ReservationDomain and UnitOfWork classes are never initialized • The following messages (registerNewReservation(res) , commit()) are never sent to UnitOfWork • ResevationMapper never sends a confirmation back.
Figure 8: Modifying a Reservation	• Message set(title,description,startTime,endTime,roomId,studentID) never sent to ReservationDomain • The following messages (registerDirtyReservation(res) , commit()) are never sent to UnitOfWork

	• ResevationMapper never sends a confirmation back. • ModifyReservation is not a message in the class ReservationMapper
Figure 9: Deleting a Reservation	• Message delete(res) never sent to ReservationDomain • The following messages (registerDeletedReservation(res) , commit()) are never sent to UnitOfWork • ResevationMapper never sends a confirmation back. • Delete() is not a message in the class ReservationMapper
Figure 10: Editing Profile Information	• Message modifyStudent(oldEmail, newEmail, newPass, oldPass) never sent to studentMapper • The following messages (registerDirtyReservation(stu) , commit()) are never sent to UnitOfWork • StudentMapper never sends a confirmation back.

Table 16: Interaction Diagrams Updates

Issue #6 - [Fix] Inconsistencies in Domain Model

Author: Philippe Kuret

Looking at their Domain Model, we can clearly see that it does not accomplish the objective that a Domain Model is supposed to show. There is a clear mismatch between the code and the Domain Model since it is showing the interaction between the database and the presentation and it does not show in any way the conceptual classes. A complete makeover would have been necessary to clearly represent what is written in the source code. Their Domain Model is presenting the TDG, Mapper and Unit Of Work and how they are associated with the classes instead of showing how the conceptual classes are interacting with each other. These classes should have been shown in the interaction diagram instead of the Domain Model. We would've had to check the code for all interaction between classes and if there is any, translate it back to the SRS. This is our Domain Model based on what is written in their code:

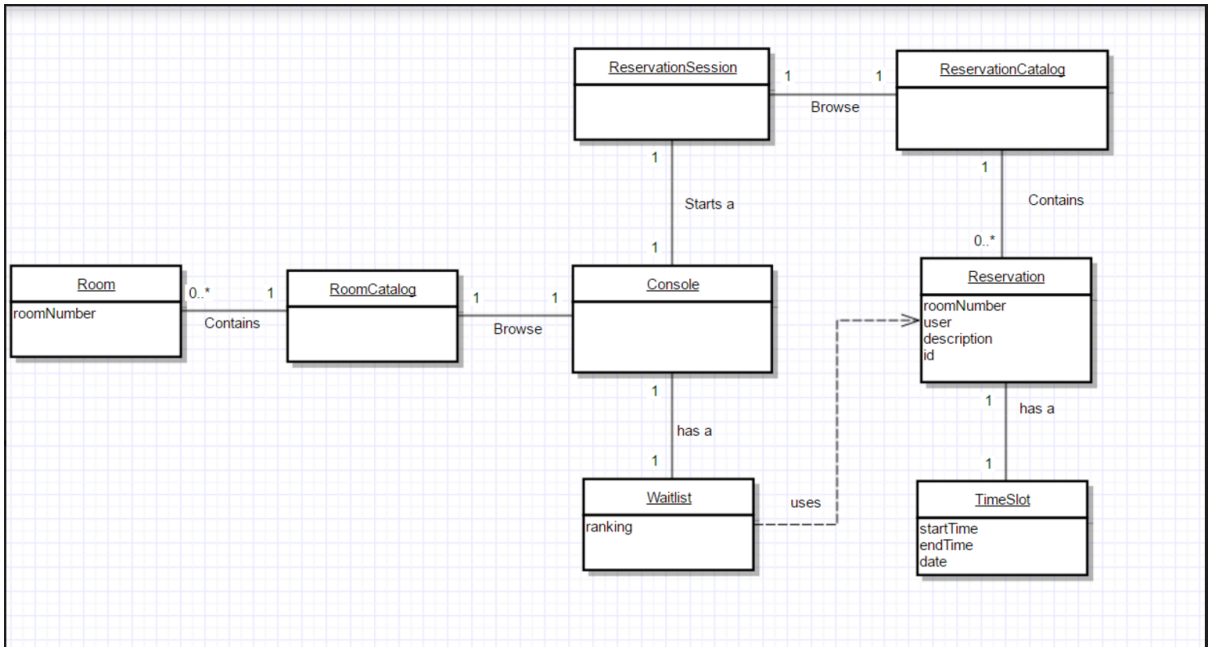

Figure 35: Domain Model Update

Since we had to redo their Domain Model, our workload has dramatically increased for our limited time constraint. This is the reason why we have decided to use a different project to refactor and add the new requirements.

Issue #1 - [Fix] Inconsistencies in Use Case Model

Author: Alexander Rosser

Issues were found in the Use Case model. The first problem with the model is that it contains use cases which are not found in the SRS and some use cases listed in the SRS are not displayed on the model. From the original model, the following are presented: authentication, create reservation, change profile details, modify time, modify room, log out and add to waitlist however none of these use cases are presented elsewhere in the SRS. Furthermore, use cases 1, 2, 3 and 4 are not presented in the use case model. Use cases 1 and 2 appear to simply inconsistent naming between the model and the use case.

Use Case Name	Use Case Model Name
UC1 Check Username & Password	Authentication
UC2 Add Reservation	Create Reservation

Table 17: Modified Use Cases

The second issue with the Use Case Model is that there are missing labels on relational links between use cases. Labels should have been added to the links connected to authentication

as well as conditional gates should have been added on some links. The following is the original use case model.

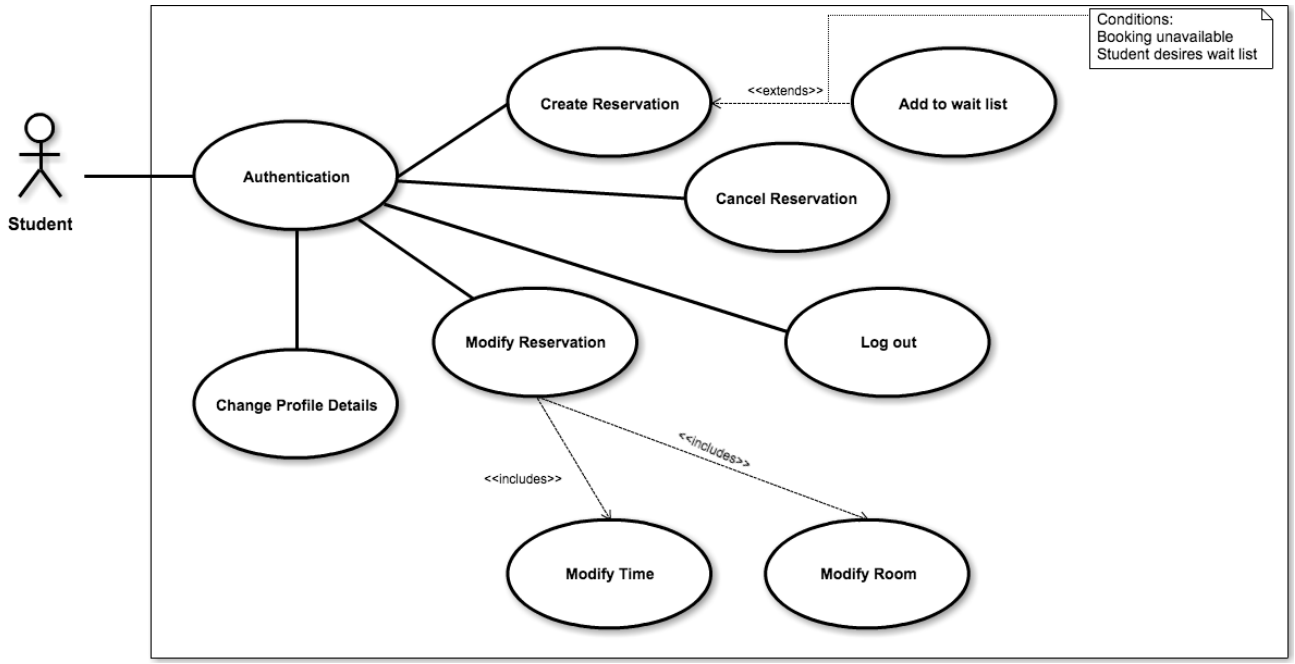

Figure 36: Original Use Case Model