OOAD Project 7

Project Title: Meeting Room Scheduler

Team-mates:

1. Bhoomika Prasad
2. Shamanth Urs Jagadeesharaje Urs

Final State of the System:

We have successfully implemented the user authentication where The system should allow users to create accounts, log in, and log out securely. Based on the user selected date and time, the system displays all the available rooms. Then the system prompts the user to select one of the room and book the book under their name with any addons selected by the user. We also implemented the feedback submission system to capture and store the feedback on meeting rooms from user. Admin has the access to add/ delete the rooms and can see all the feedbacks given by the user. We have successfully implemented four design patterns – Factory, Decorator, Singleton and Observer. Observer pattern tracks the number of rooms booked and amount collected by the addons. All the relevant frontend and backend are completed successfully.

Final Class diagram and comparison statement:

(Final PDF copies are attached at the end of the file)

Final Class Diagram:
https://lucid.app/lucidchart/9b5f9d7b-5120-4163-8713-a22f4ef6ebf5/edit?viewport_loc=-8575%2C-4820%2C19211%2C11135%2C0_0&invitationId=inv_3a2a43a0-7e35-4e92-846f-abd28f6a36ed

Project 5 Class Diagram:
https://lucid.app/lucidchart/027f999c-3b25-45c8-9246-e565fa045a1f/edit?viewport_loc=43%2C-4%2C11176%2C4734%2CHWEp-vi-RSFO&invitationId=inv_da182ec7-dec7-4374-89d3-c26fdbbf7511

What changed between Project 5 and Project 7 (Final Submission):

1. As we have used Spring Boot 2.0, we modified our existing designs according to the rules of Spring Boot. Hence, each model (Room, Meeting, Feedback, User) has its own repository service and controller. Each repository interacts with MySQL Database separately. A Service encapsulates repository and controller is where we have all the REST APIs.
2. We added few more API's to facilitate integration with the Front-end like getAllMeetingsByUserID and getAllFeedbacks.
3. We added 2 separate models – LoginRequest and LogoutRequest to facilitate login and logout of users.

Third-Party code vs. Original code Statement:

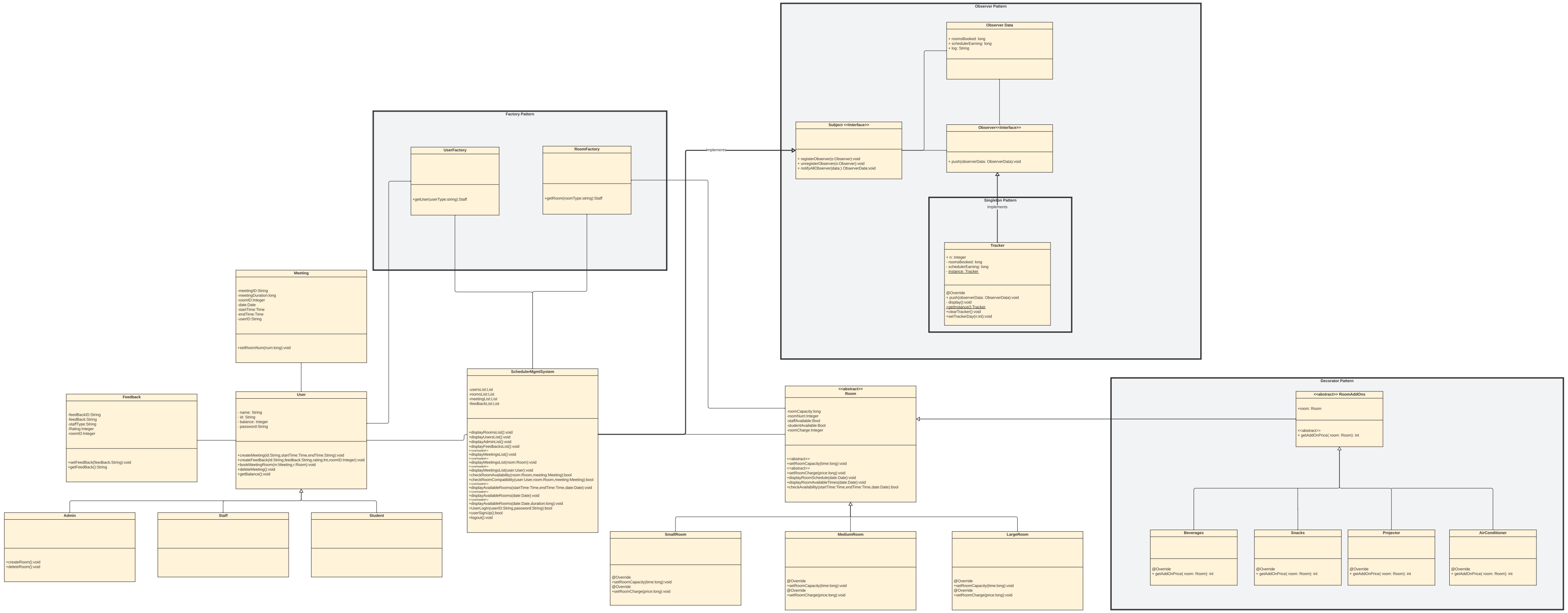
- We have used spring boot to build our application. Hence, while developing the backend the main reference used was the official website of spring boot - https://spring.io/guides/gs/spring-boot/
- For front-end UI design login/logout pages, the form container UI basic structure was based on the code example presented in - https://hirukarunathilaka.medium.com/signup-form-with-real-time-validation-using-react-typescript-6a7dfb3122b5

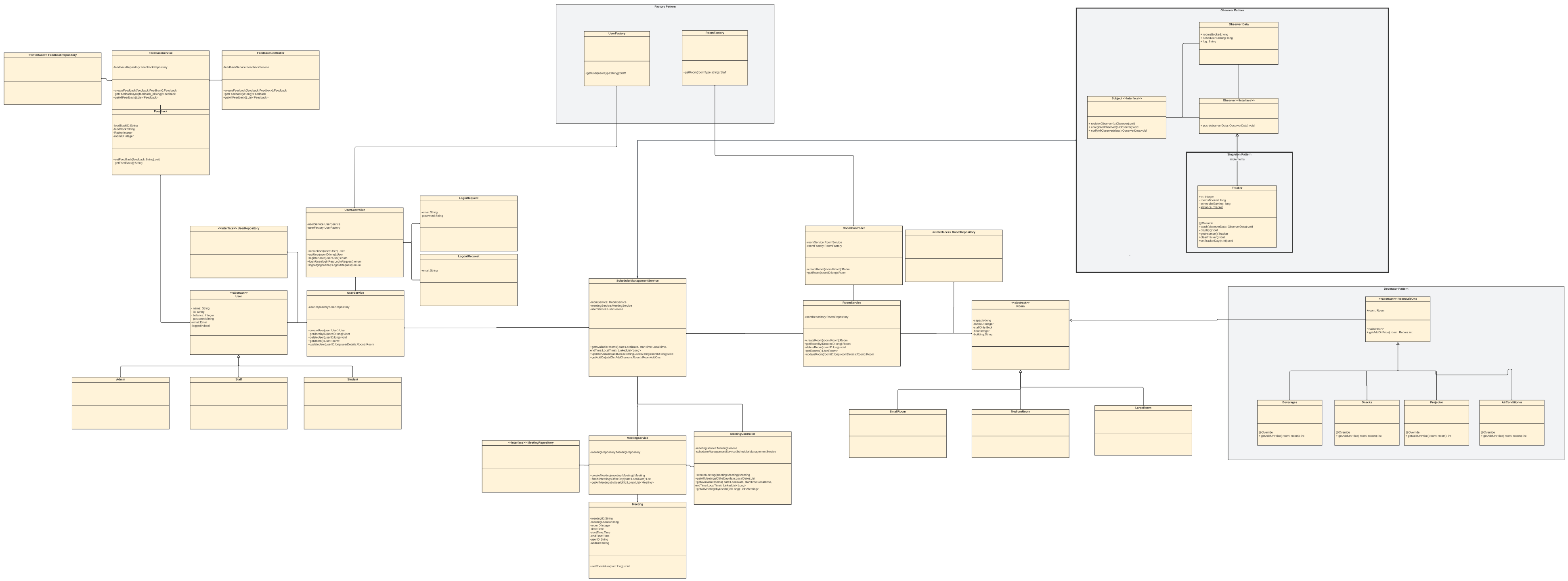
Statement on the OOAD process for your overall Semester Project:

Class Diagram: Class diagram played a very pivotal role in building the blueprint for the application. Identifying all the required elements of the application like classes, design patterns, models etc, helped us properly structure and come up with all required entities.

Sequence Diagram: Sequence diagram helped us think through all the interaction between different entities and systems and in what sequence we have to go through. For example, for room booking, we could easily identify the required steps when implementing as we had all the required steps through sequence diagram which we had created initially.

API Contracts: One major miss at our end was API contract. As we didn't define the API contract for the front end consumption initially, it became a lot of to and fro between front end and back end where we had to make change based on the need. This lead to more effort in making changes to accompany the front end than building the whole system.


Factory Pattern
UserFactory
+getUser(userType:string):Staff
RoomFactory
+getRoom(roomType:string):Staff
Implements
Observer Pattern
Observer Data
+ roomsBooked: long
+ schedulerEarning: long
+ log: String
Subject <<Interface>>
+ registerObserver(o:Observer):void
+ unregisterObserver(o:Observer):void
+ notifyAllObserver(data:) ObserverData:void
Observer<<Interface>>
+ push(observerData: ObserverData):void
Singleton Pattern
Implements
Tracker
+ n: Integer
- roomsBooked: long
- schedulerEarning: long
- instance: Tracker
@Override
+ push(observerData: ObserverData):void
- display():void
+getInstance():Tracker
+clearTracker():void
+setTrackerDay(n:int):void
Meeting
-meetingID:String
-meetingDuration:long
-roomID:Integer
-date:Date
-startTime:Time
-endTime:Time
-userID:String
+setRoomNum(num:long):void
Feedback
-feedBackID:String
-feedBack:String
-staffType:String
-Rating:Integer
-roomID:Integer
+setFeedBack(feedback:String):void
+getFeedBack():String
User
- name: String
- id: String
- balance: Integer
- password:String
+createMeeting(id:String,startTime:Time,endTime:String):void
+createFeedback(id:String,feedback:String,rating:Int,roomID:Integer):void
+bookMeetingRoom(m:Meeting,r:Room):void
+deleteMeeting():void
+getBalance():void
Admin
+createRoom():void
+deleteRoom():void
Staff
Student
SchedulerMgmtSystem
-usersList:List
-roomsList:List
-meetingList:List
-feedbackList:List
+displayRoomsList():void
+displayUsersList():void
+displayAdminList():void
+displayFeedbacksList():void
<<overloaded>>
+displayMeetingsList():void
<<overloaded>>
+displayMeetingsList(room:Room):void
<<overloaded>>
+displayMeetingsList(user:User):void
+checkRoomAvailability(room:Room,meeting:Meeting):bool
+checkRoomCompatibility(user:User,room:Room,meeting:Meeting):bool
<<overloaded>>
+displayAvailableRooms(startTime:Time,endTime:Time,date:Date):void
<<overloaded>>
+displayAvailableRooms(date:Date):void
<<overloaded>>
+displayAvailableRooms(date:Date,duration:long):void
+UserLogIn(userID:String,password:String):bool
+userSignUp():bool
+logout():void
<<abstract>>
Room
-roomCapacity:long
-roomNum:Integer
-staffAvailable:Bool
-studentAvailable:Bool
-roomCharge:Integer
<<abstract>>
+setRoomCapacity(time:long):void
<<abstract>>
+setRoomCharge(price:long):void
+displayRoomSchedule(date:Date):void
+displayRoomAvailableTimes(date:Date):void
+checkAvailability(startTime:Time,endTime:Time,date:Date):bool
SmallRoom
@Override
+setRoomCapacity(time:long):void
@Override
+setRoomCharge(price:long):void
MediumRoom
@Override
+setRoomCapacity(time:long):void
@Override
+setRoomCharge(price:long):void
LargeRoom
@Override
+setRoomCapacity(time:long):void
@Override
+setRoomCharge(price:long):void
Decorator Pattern
<<abstract>> RoomAddOns
+room: Room
<<abstract>>
+ getAddOnPrice(room: Room): int
Beverages
@Override
+ getAddOnPrice(room: Room): int
Snacks
@Override
+ getAddOnPrice(room: Room): int
Projector
@Override
+ getAddOnPrice(room: Room): int
AirConditioner
@Override
+ getAddOnPrice(room: Room): int


<<interface>> FeedbackRepository
FeedbackService
-feedbackRepository:FeedbackRepository
+createFeedback(feedback:Feedback):Feedback
+getFeedbackByID(feedback_id:long):Feedback
+getAllFeedback():List<Feedback>
FeedbackController
-feedbackService:FeedbackService
+createFeedback(feedback:Feedback):Feedback
+getFeedback(id:long):Feedback
+getAllFeedback():List<Feedback>
Feedback
-feedBackID:String
-feedBack:String
-Rating:Integer
-roomID:Integer
+setFeedBack(feedback:String):void
+getFeedBack():String
<<interface>> UserRepository
UserController
-userService:UserService
-userFactory:UserFactory
+createUser(user:User):User
+getUser(userID:long):User
+registerUser(user:User):enum
+loginUser(loginReq:LoginRequest):enum
+logout(logoutReq:LogoutRequest):enum
LoginRequest
-email:String
-password:String
LogoutRequest
-email:String
<<abstract>>
User
- name: String
- id: String
- balance: Integer
- password:String
-email:Email
-loggedIn:bool
UserService
-userRepository:UserRepository
+createUser(user:User):User
+getUserByID(userID:long):User
+deleteUser(userID:long):void
+getUsers():List<Room>
+updateUser(userID:long,userDetails:Room):Room
Admin
Staff
Student
Factory Pattern
UserFactory
+getUser(userType:string):Staff
RoomFactory
+getRoom(roomType:string):Staff
SchedulerManagementService
-roomService: RoomService
-meetingService:MeetingService
-userService:UserService
+getAvailableRooms(date:LocalDate, startTime:LocalTime, endTime:LocalTime): LinkedList<Long>
+updateAddOns(addOnList:String,userID:long,roomID:long):void
+getAddOn(addOn:AddOn,room:Room):RoomAddOns
<<interface>> MeetingRepository
MeetingService
-meetingRepository:MeetingRepository
+createMeeting(meeting:Meeting):Meeting
+findAllMeetingsOftheDay(date:LocalDate):List
+getAllMeetingsbyUserId(Id:Long):List<Meeting>
MeetingController
-meetingService:MeetingService
-schedulerManagementService:SchedulerManagementService
+createMeeting(meeting:Meeting):Meeting
+getAllMeetingsOftheDay(date:LocalDate):List
+getAvailableRooms(date:LocalDate, startTime:LocalTime, endTime:LocalTime): LinkedList<Long>
+getAllMeetingsbyUserId(Id:Long):List<Meeting>
Meeting
-meetingID:String
-meetingDuration:long
-roomID:Integer
-date:Date
-startTime:Time
-endTime:Time
-userID:String
-addOns:string
+setRoomNum(num:long):void
RoomController
-roomService:RoomService
-roomFactory:RoomFactory
+createRoom(room:Room):Room
+getRoom(roomID:long):Room
<<interface>> RoomRepository
RoomService
-roomRepository:RoomRepository
+createRoom(room:Room):Room
+getRoomByID(roomID:long):Room
+deleteRoom(roomID:long):void
+getRooms():List<Room>
+updateRoom(roomID:long,roomDetails:Room):Room
<<abstract>>
Room
-capacity:long
-roomID:Integer
-staffOnly:Bool
-floor:Integer
-building:String
SmallRoom
MediumRoom
LargeRoom
Observer Pattern
Observer Data
+ roomsBooked: long
+ schedulerEarning: long
+ log: String
Subject <<Interface>>
+ registerObserver(o:Observer):void
+ unregisterObserver(o:Observer):void
+ notifyAllObserver(data:) ObserverData:void
Observer<<Interface>>
+ push(observerData: ObserverData):void
Singleton Pattern
Implements
Tracker
+ n: Integer
- roomsBooked: long
- schedulerEarning: long
- instance: Tracker
@Override
+ push(observerData: ObserverData):void
- display():void
+getInstance():Tracker
+clearTracker():void
+setTrackerDay(n:int):void
Decorator Pattern
<<abstract>> RoomAddOns
+room: Room
<<abstract>>
+ getAddOnPrice(room: Room): int
Beverages
@Override
+ getAddOnPrice(room: Room): int
Snacks
@Override
+ getAddOnPrice(room: Room): int
Projector
@Override
+ getAddOnPrice(room: Room): int
AirConditioner
@Override
+ getAddOnPrice(room: Room): int