OOAD Project 6

Status Summary

Project Title: Meeting Room Scheduler

Team-mates:
1. Bhoomika Prasad
2. Shamanth Urs Jagadeesharaje Urs

Work Done:

Until now, we have mostly worked on designing and implementing the backend, by implementing all the classes and exposing the services in form of REST APIs. Currently, we have all the major class functionalities defined. All the basic services related to Users, Meetings, Feedbacks, Rooms are completed, and smoke tested. We are yet to integrate the same with front end in the next phase. Many tasks were coordinated/worked on together. Below is the rough work split up among us:

Bhoomika:
1. User Class and subclasses implementation.
2. Creation of Services, Controller and Repository associated with User model.
3. Implementing Factory Pattern for Rooms and Users.
4. Feedback Class implementation.
5. Creation of Services, Controller and Repository associated with Feedback model.
6. Implementing login, logout services.

Shamanth:

7. Room Class and subclasses implementation
8. Creation of Services, Controller and Repository associated with Room model.
9. Implementing Decorator Pattern for providing Add-ons after room is booked.
10. Connecting and implementing Scheduler Management Service to manage all other operations.
11. Meeting Class implementation.
12. Creation of Services, Controller and Repository associated with Meeting model.

Changes or Issues Encountered:

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.

We also had to change the database schema to match our requirements:

a. Some variable and table name changes to improve readability.

b. Adding columns to user table to implement login and staff-only rooms functionality.

Patterns:

We have presently implemented two design patterns:

a. Factory Design Pattern:
   This pattern is implemented to create rooms and users with a common interface without exposing the internal object creation details.
b. Decorator Design Pattern:
   This pattern is implemented to provide add-on facilities to the user after he books a room. Based on the add-ons he selects; he will be charged extra amount. Decorator Pattern has helped us achieve this functionality without making any changes to existing Room and its related Entities.

Class Diagram

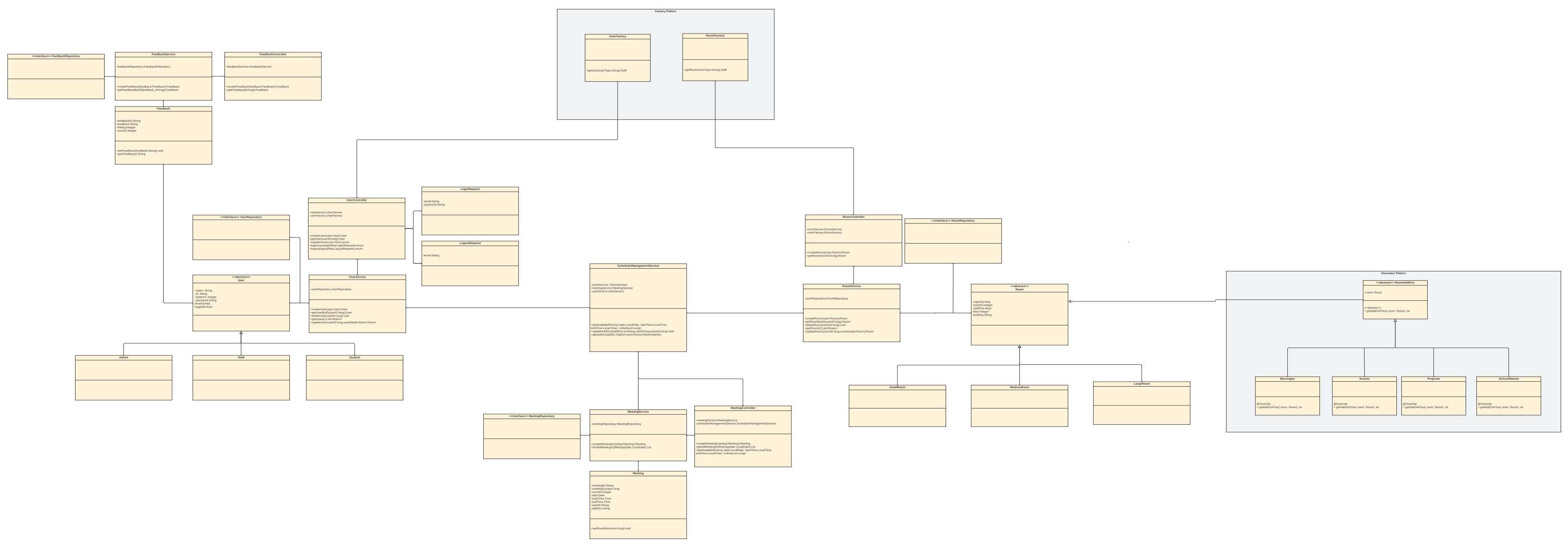
Attached in landscape mode in a new sheet. Please zoom in to see the text.

Plan for Next Iteration

Estimated Time for Completion: 7 – 10 days

Iteration TO-DO Plans:
1. Integrate the present back-end functionalities by creating front-end pages and integrating the same for the user to navigate.
2. Implement Observer and Singleton Pattern for notifying observers of event changes.
3. Create Testing Suites to check and fix bugs, if any.

Final Project by 5/3:
After our final iteration, we plan to have a complete functional web application which can handle complete real world room scheduling process for a meeting like adding/deleting rooms, displaying rooms to user, handle login logout, scheduling rooms, providing feedback, and many more.



Factory Pattern
UserFactory
+getUser(userType:string):Staff
RoomFactory
+getRoom(roomType:string):Staff
<<interface>> FeedbackRepository
FeedbackService
-feedbackRepository:FeedbackRepository
+createFeedback(feedback:Feedback):Feedback
+getFeedbackByID(feedback_id:long):Feedback
FeedbackController
-feedbackService:FeedbackService
+createFeedback(feedback:Feedback):Feedback
+getFeedback(id:long):Feedback
Feedback
-feedBackID:String
-feedBack:String
-Rating:Integer
-roomID:Integer
+setFeedBack(feedback:String):void
+getFeedBack():String
<<interface>> UserRepository
<<abstract>>
User
- name: String
- id: String
- balance: Integer
- password:String
-email:Email
-loggedIn:bool
Admin
Staff
Student
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
UserService
-userRepository:UserRepository
+createUser(user:User):User
+getUserByID(userID:long):User
+deleteUser(userID:long):void
+getUsers():List<Room>
+updateUser(userID:long,userDetails:Room):Room
LoginRequest
-email:String
-password:String
LogoutRequest
-email:String
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
MeetingController
-meetingService:MeetingService
-schedulerManagementService:SchedulerManagementService
+createMeeting(meeting:Meeting):Meeting
+getAllMeetingsOftheDay(date:LocalDate):List
+getAvailableRooms(date:LocalDate, startTime:LocalTime, endTime:LocalTime): LinkedList<Long>
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
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