Project Phase 1 And Phase 2 Software Design and Architecture

Name	ID
Ziad Medhat Elsaid Atia	20226044
Marwan Ahmad Moawad	20226099
Fady Nagy	20226074
Yousef Sherif Mohamad	20226117

Introduction:

1.1 Identifying information

- Architecture name: The travel agency booking application architecture.
- System of interests: The system is a software product for a travel agency in Cairo which handles booking of tickets to local events and hotel reservations by the user. The system also sends notifications to the user including booking, creating account, updating account, etc... to the user's mail if needed or user phone number via SMS. The system will include a notification module to send notifications, manage different templates of messages and provide some statistics of the notifications.

1.2 Supplementary Information

Scope:

The architecture covers booking hotel reservations provided by hotel providers or database, and booking of local events, and sending notifications to the Users through various means like mail about booking information and recommendations to local events.

Context:

An online application for a travel agency to allow users to easily search, view and book hotel reservations and tickets for local events through the applications and get notifications about their actions.

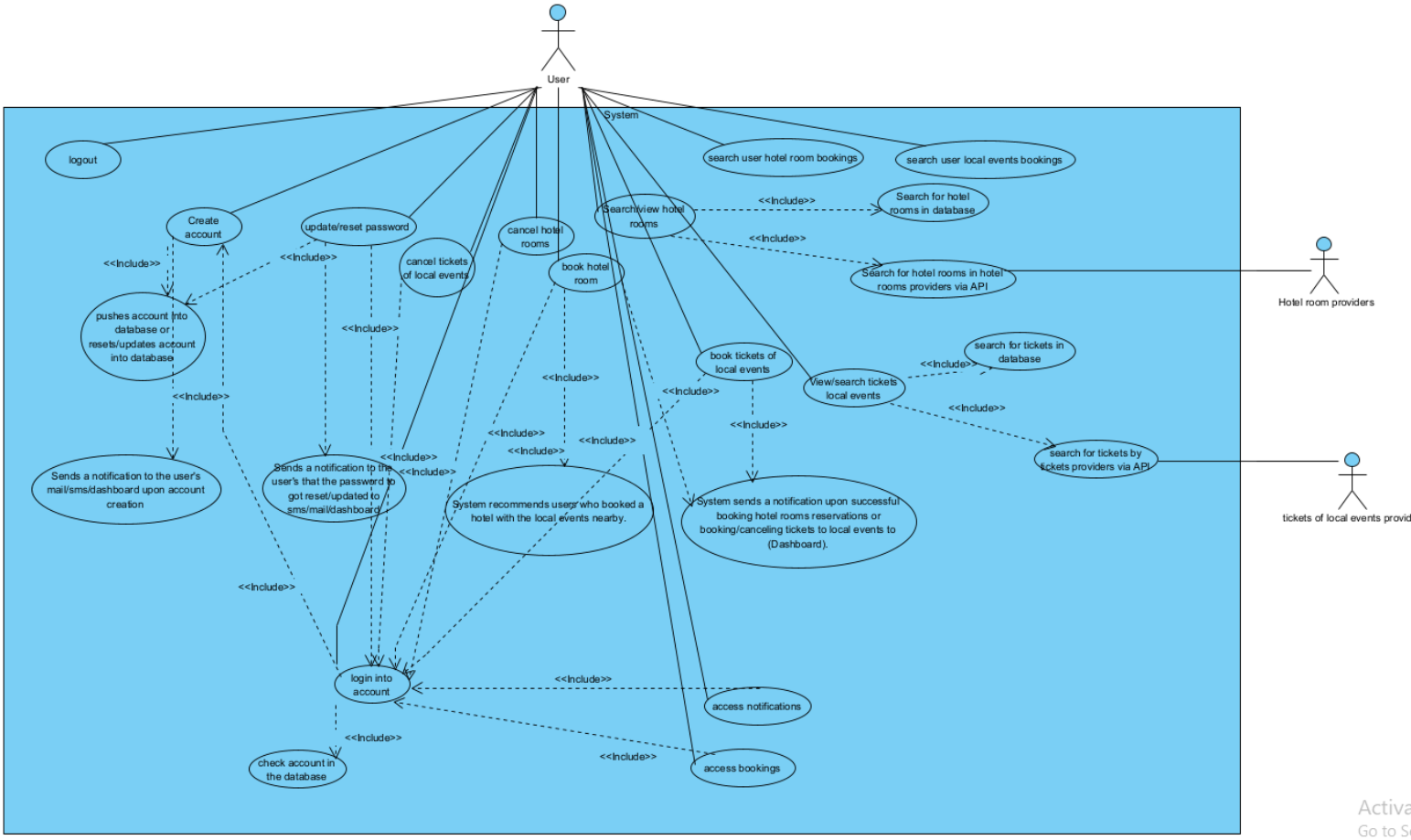
Glossary:

- **Notification:** A notification is a message sent to a user to inform them of something relevant, such as updates, reminders, or confirmations
- **Notifications Module:** A notifications module is a component of a software system responsible for managing the creation, queuing, and delivery of notifications to users
- **API:** An API is a set of protocols, tools, and definitions that allow software components to communicate with each other, also used to fetch information from external services
- **Database:** A part in the system where relevant data is stored

- **Dashboard:** A part of the system that shows the user some relevant information that concerns him such as a list of his current bookings
- **SOLID principles:** SOLID is a set of five object-oriented design principles aimed at making software designs more understandable, flexible, and maintainable
- **OOD:** Object-Oriented Design is a way of designing systems by organizing them around objects, which contain attributes and methods instead of the traditional procedural programming way that focuses only on 'actions'

1.3 Other information

1.3.1 Overview (use case diagram):



1.3.2 Architecture Evalution:

2 Stakeholders and concerns

2.1 Stakeholders

- Client (The Travel Agency)
    - The travel agency that requested and will pay for the development of the software product
- End-users
    - The end users that will use the online application's services.
- Developers
    - The Individuals who will design the system architecture and build the system
- Testers
    - The individuals who will create test cases for different scenarios and test the system after the implementation, and will test how well the architecture achieve the requirements.
- Hotel providers
    - The suppliers of the hotel rooms data and availability.

2.2 Concerns

Functional requirements:

User Account:

- Users can create/login account in the application.
- Users can update their password.
- System sends a notification upon account creation (Mail/SMS). (Using Notification Module)
- System sends notification upon user updating password (Mail/SMS) (Using Notification Module)
- Users can logout.
- User can access their notifications through their dashboard.

Hotel Management:

- Users can search/view/book hotel rooms.
- Users can cancel booked hotel rooms.
- System MUST check if a user has an account and is logged in before booking hotel rooms.
- System may search for hotel rooms information from a database and using an API of the specific hotel providers.

- System will recommend users who booked a hotel with the local events nearby using a dashboard notification (on the dashboard).
- System will send a dashboard notification and an Email notification to the user when the user books a hotel room.

Event Management:

- Users can search/view/book/cancel tickets of local events.
- System MUST check if a user has an account and is logged in before booking local events.
- System may search for local events nearby using an API of local events provider and from database
- System will send a dashboard notification and an Email notification to the user when the user books a local event ticket.

Non-functional requirements

Quality:

- **Scalability:** The system must be able to handle up to 100,000 concurrent users and scale horizontally to accommodate increased traffic. In case of a large surge in user numbers, the system should be capable of doubling server capacity, ensuring that it can meet growing demands without service degradation. the number of users of the application is very large.
- **Availability:** The application should be highly available and incase of server failures, at least one server should stay available to provide critical functionalities, (System should be available at all times)
- **Reliability:** System should be fault tolerant ensuring that the fault of one part of the system doesn't bring the whole system down
- **Modifiability:** The system should follow SOLID principles and object-oriented design (OOD) best practices to maintain flexibility and ease of modification. By adhering to these principles, the system can evolve over time with minimal impact on existing functionalities, and future updates could be delivered in less time frame
- **Reusability:** System should be designed for reusability so we can utilize its services in other future projects with ease

3-Architecture and Architecture Views:

Components:

- **SpringController**: provided interface: SpringController Interface. Required Interfaces:

ID Searcher Interface, Notification Interface, Dashboard Interface, Booking Interface, HotelManagment Interface, UserManagment Interface, LocalEventManagment Interface.

- **Dashboard:** provided interface: Dashboard Interface. Required interfaces: Model Interface, Notification Interface.
- **ID Searcher:** provided interface: ID Searcher Interface. Required interfaces: Model Interface
- **UserManagment:** provided interface: UserManagment Interface. Required interfaces: Model Interface, Notification Interface.
- **Booking:** provided interface: Booking Interface. Required Interfaces: Model Interface, Notification Interface.
- **HotelManagment:** provided interface: HotelManagment Interface. Required interfaces: Model Interface, Notification Interface.
- **LocalEventManagment:** provided interface: LocalEventManagment Interface. Required interfaces: Model Interface, Notification Interface.
- **NotificationModule:** provided Interface: Notification Interface. Required Interfaces: Model Interface.
- **Model:** provided Interface: Model Interface. Required Interface: Database Interface,Hotel Provider Interface, LocalEvent Provider Interface.
- **Database:** provided Interface: Database Interface
- **Hotel Provider (external):** provided Interface: Hotel Provider Interface.
- **LocalEvent Provider (external):** provided Interface: LocalEvent Provider Interface.

Connecter:

The connecters are the interfaces which each layer communicates with the layer below it plus the communication between the components inside the same layer.

Constraints:

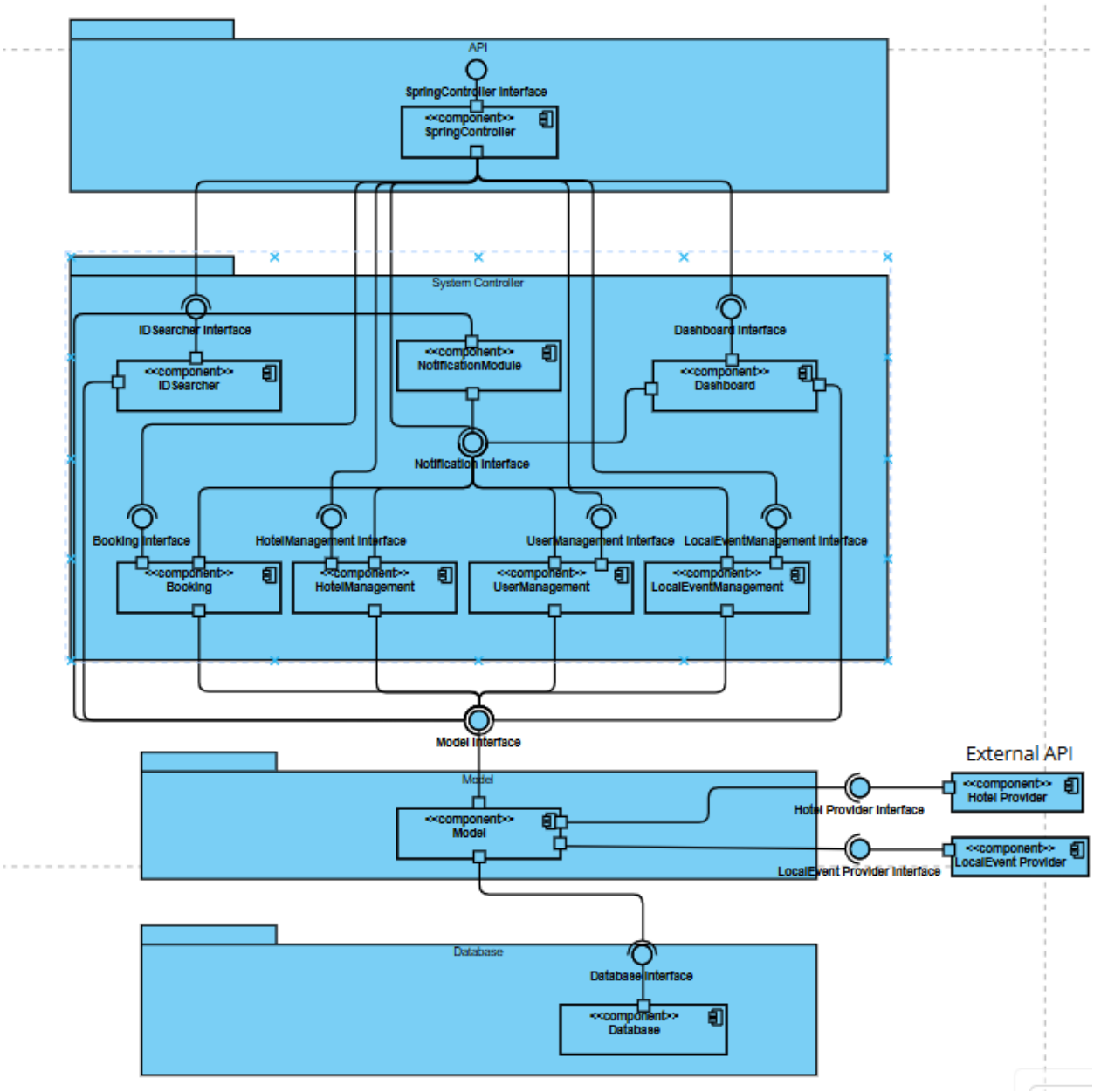
- System has to be implemented using java spring boot
- The notification module should be designed to be able to be reused for multiple applications
- A notification module must be implemented as part of the system
- Users must have an account to book event tickets or hotels.
- The waterfall software development model should be followed in this application.

Architecture Style:

The main Architecture style is the Layered Architecture where there are four layers: view, system controllers, Model, Database.

Architecture Views:

Conceptual View:

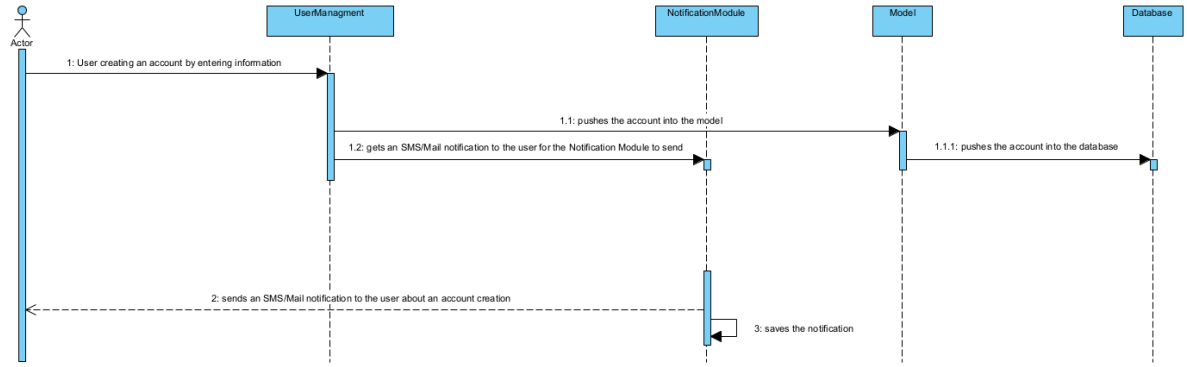


Execution view:

The view where components are execution units and the connecters are the flow information (it can be represented by sequence diagram) where this view focuses more on the runtime behavior of the architecture like when the user creates an account how will the components will interact with each other to achieve this goal. we assume that the user or the actor here is the view that the user will communicate through to execute commands.

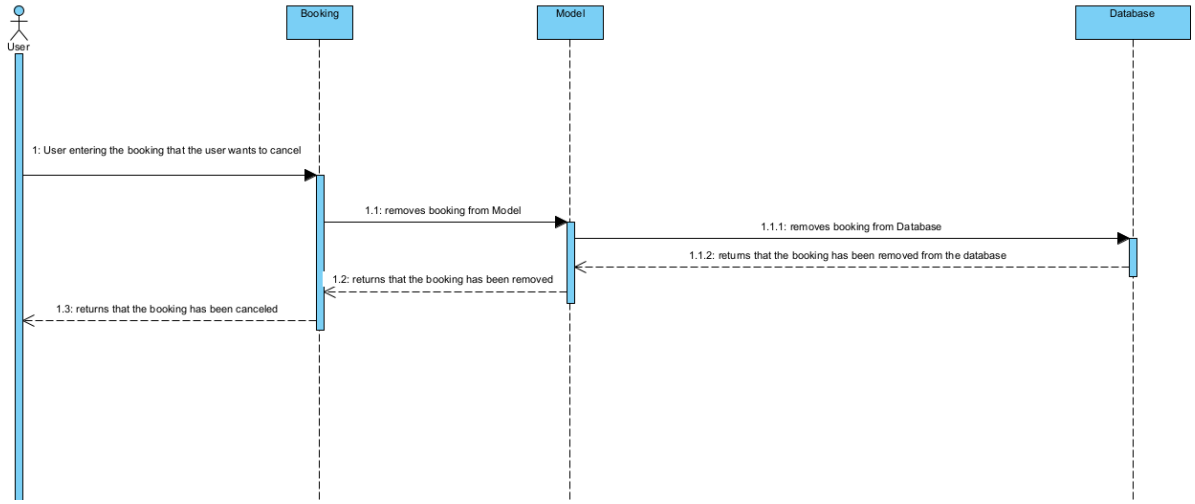
First sequence:

User management creates an account and pushes it into the model and then the model pushes the account into the database than user management sends a notification for the user that the account has been created with Notification Module



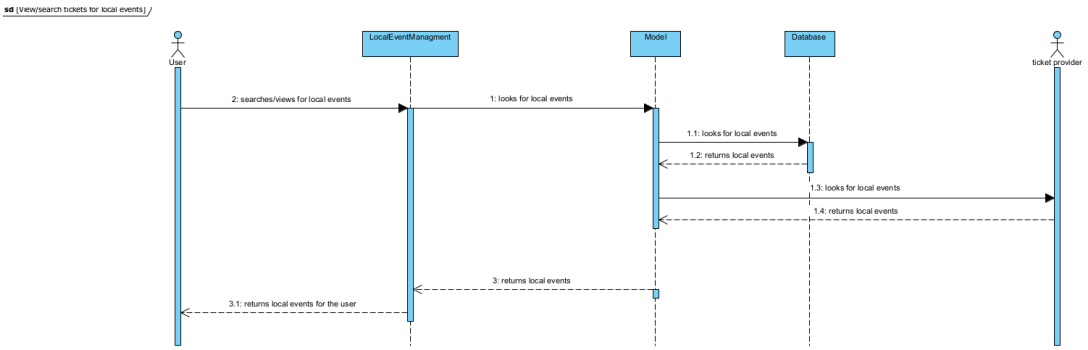
Second sequence:

In this scenario where user cancel the booking of hotel rooms or tickets of local events, Booking removes the booking of the user from the model than model removes it from the database



Third sequence:

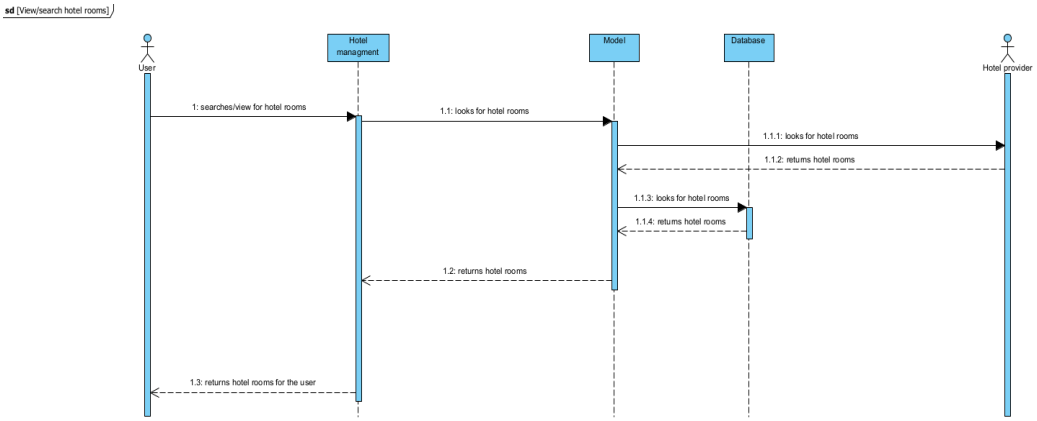
Here the user searches for tickets of local events then local event management gets them from the model and the model gets them from both the database and ticket provider or local event provider



Fourth sequence:

Same as third but instead of tickets of local events

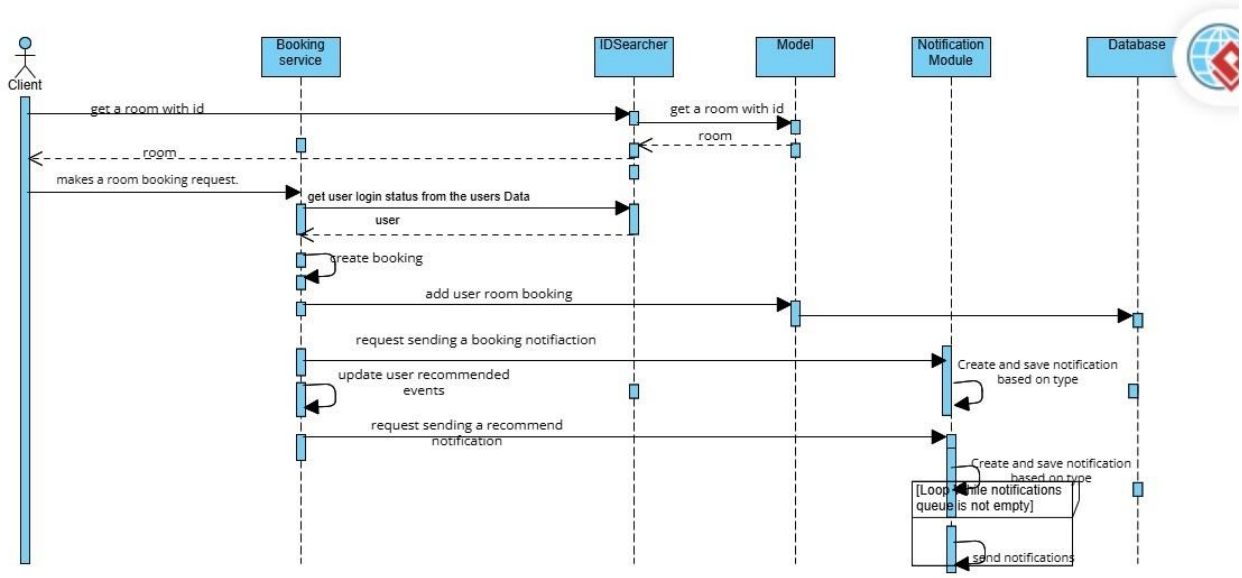
it's for hotel rooms and model gets from both database and hotel provider.



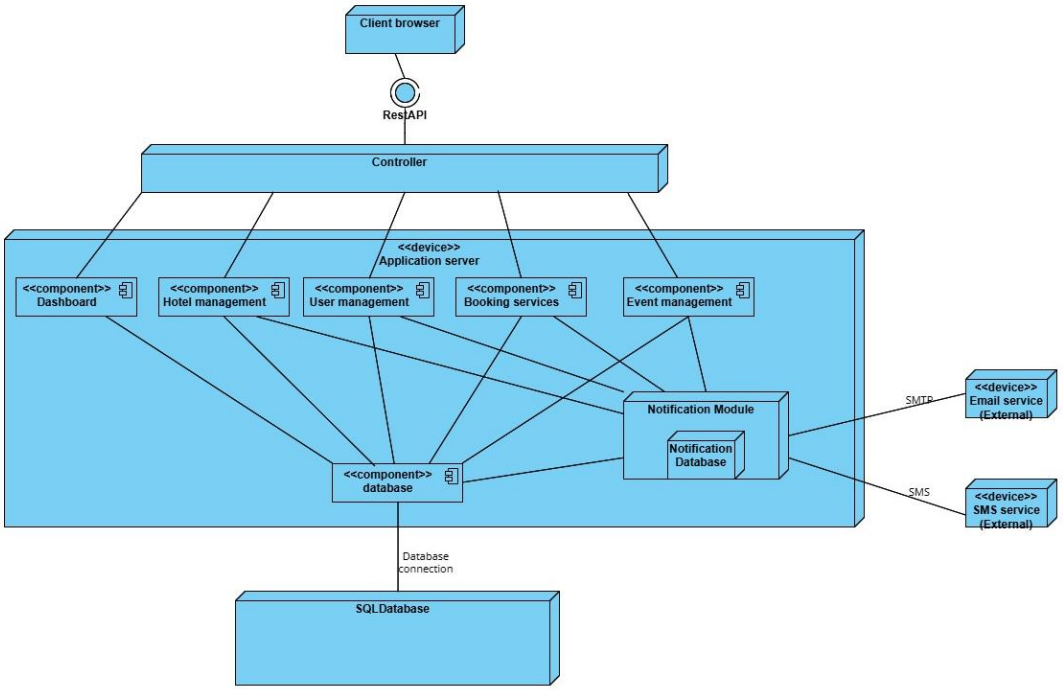
Fifth sequence:

The user sends a booking request for a room, with the room, user id, and other booking details to book a room which will be handled by the booking service which checks the user login status of the user from the Model, and creates a booking for the user and saves it in the database. Sends a request to the notification module to send a notification to the user via a channel chosen by the booking service.

The notification module creates the notification and saves it in its database and sends it via the chosen channel.



3- Deployment view



Clients send requests and the Api routes requests to services; the services then send the response.

The SQL database contain the tables needed for the services like users, notifications, event tickets, hotel rooms, bookings.
The email service and the SMS service route the notifications to the user or users if bulk notification.
The application logic is in the application server.
Components like event manager and hotel provided request data from the external providers using http messaging.

4-Design process and rationale

Step one:
during the Requirement gathering and analysis phase We began by gathering the system's functional and nonfunctional requirements. we refined the requirements and created a more detailed version of them and we identified the non-functional requirements, we also evaluated their importance and the trade-offs within our system.
Step two:
we started thinking about which architecture style to build our system with, the benefits of this style, and does it align with our goals or not, we settled with the Layered Architecture.
Rationale for Layered Architecture:
The Layered Architecture style was selected because of its ability to manage the system as a set of layers with each layer being responsible for a part of the system. it ensures flexibility, maintainability, and reliability, which is good for systems that rely on separation of concerns.

Key benefits:

- Separation of concerns: Since the system is divided to multiple layers, each layer now has its own responsibility which allows maintaining each layer without affecting the others.
- Scalability: Since the system is divided to multiple layers, each layer can scale on its own.
- Reusability: Different layers of the layered architecture can be used in more projects.
- Maintainability: Maintainability enhanced since each layer could be updated on its own.

Step 3:
Component Grouping and Interfaces With the architecture style selected, we focused on grouping related functional requirements into appropriate system components. This step helped us understand which requirements interact with each other and decide how components would communicate. We also defined the interfaces for each component.
Step 4:
Diagramming and Validation To visually represent the system, we created several diagrams. For example, the conceptual (static) view allowed us to understand how components would be arranged and interact. The execution (dynamic) view showed how the system would behave during runtime and helped validate our design decisions by illustrating how specific scenarios would be handled.
Reflection:

The Layered Architecture proved effective in terms of scalability, maintainability and reusability, since the architecture style introduces separation of concerns by adding layers.

Evaluating Design through Scenarios:

Scenario#1: Modifiability: if we found a more suitable model for the system, we can easily replace the model layer with another without affecting the other layers.

Scenario#2: Availability: if the network to send a mail/SMS has failed the Notifications Module has a queue to not hold the linker which links between the system and the user mail/SMS which have all the notifications to be sent.

Scenario #3: Availability: if one of the application servers crashes due to hardware failure, the system automatically switches traffic to a backup server.

Scenario #4: Scalability: if there is increased load of data that has to be stored, system can scale the database layer to match the increased load.

Scenario#5: Security: if the user tries to log in over an unencrypted channel, the system immediately rejects the connection request and notifies the user to use a secure channel.

Scenario#6: Security: If a user starts a transaction session, the system locks the item being booked or purchased to prevent any other user from purchasing or booking it until the transaction is complete.

Assumptions:

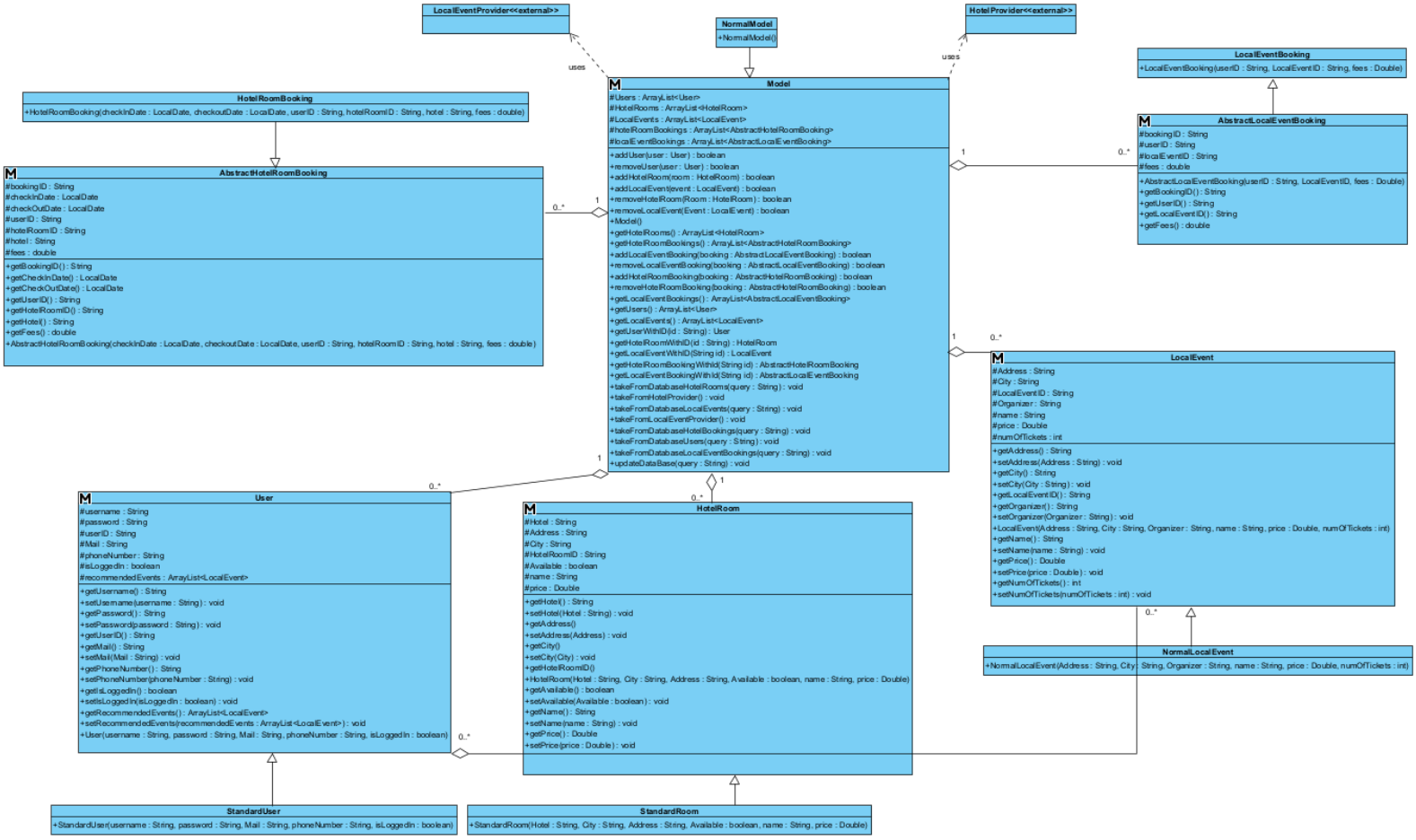
- That the notification itself will have a component that sends SMS/mail to the user.
- We assume that the database will be a dummy database and won't be implemented.
- We assume that the tickets are non-refundable and when a ticket gets canceled the number of tickets will not increase.
- We assume that the Notification Module will have its own database to track notifications.
- Users can not be removed from the system.
- The hotel room price is the hotel room price per night

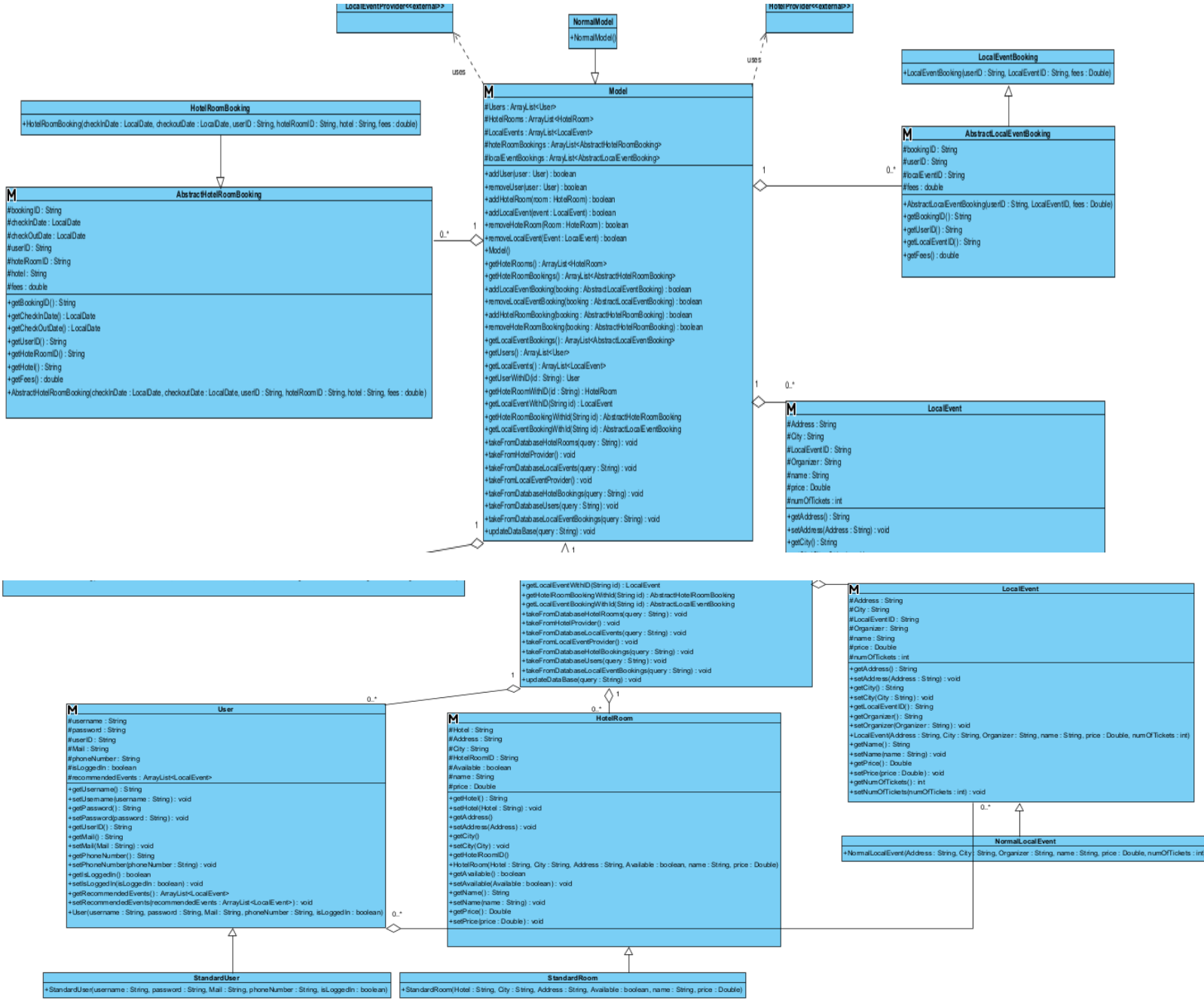
Phase 2:

Note: since Database is a dummy, we will not include it in class diagram or sequence diagram And each time the user needed to get an entity from the model the user uses IDsearcher we will not include it in the sequence diagram since its needed when the user or view has an id and want to get the entity to execute a command so we will just assume it got the necessary information to execute the command immediately.

Class Diagrams:

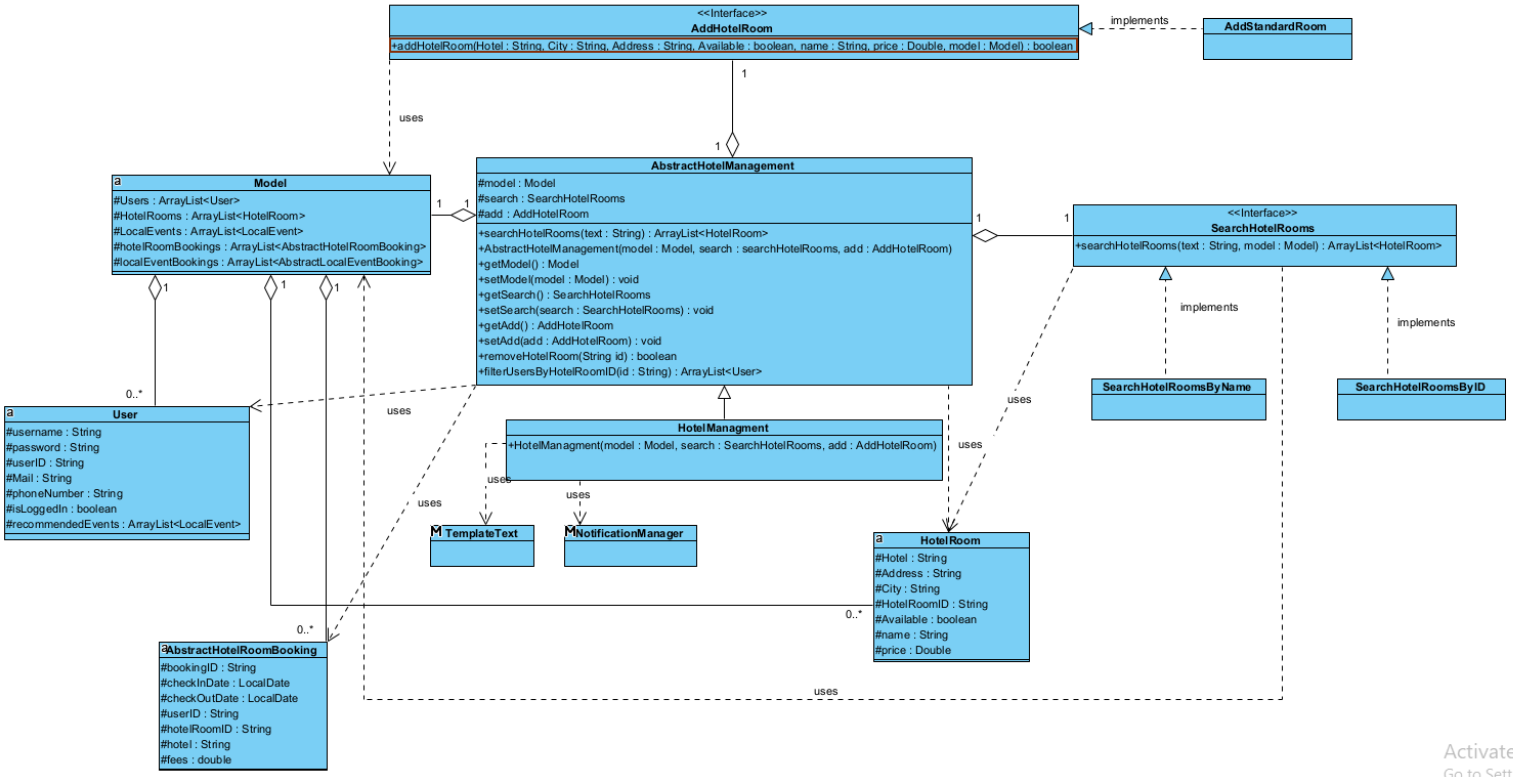
Model:




LocalEventProvider<<external>>
NormalModel
+NormalModel()
HotelProvider<<external>>
uses
uses
LocalEventBooking
+LocalEventBooking(userID : String, LocalEventID : String, fees : Double)
HotelRoomBooking
+HotelRoomBooking(checkInDate : LocalDate, checkoutDate : LocalDate, userID : String, hotelRoomID : String, hotel : String, fees : double)
Model
#Users : ArrayList<User>
#HotelRooms : ArrayList<HotelRoom>
#LocalEvents : ArrayList<LocalEvent>
#hotelRoomBookings : ArrayList<AbstractHotelRoomBooking>
#localEventBookings : ArrayList<AbstractLocalEventBooking>
+addUser(user : User) : boolean
+removeUser(user : User) : boolean
+addHotelRoom(room : HotelRoom) : boolean
+addLocalEvent(event : LocalEvent) : boolean
+removeHotelRoom(Room : HotelRoom) : boolean
+removeLocalEvent(Event : LocalEvent) : boolean
+Model()
+getHotelRooms() : ArrayList<HotelRoom>
+getHotelRoomBookings() : ArrayList<AbstractHotelRoomBooking>
+addLocalEventBooking(booking : AbstractLocalEventBooking) : boolean
+removeLocalEventBooking(booking : AbstractLocalEventBooking) : boolean
+addHotelRoomBooking(booking : AbstractHotelRoomBooking) : boolean
+removeHotelRoomBooking(booking : AbstractHotelRoomBooking) : boolean
+getLocalEventBookings() : ArrayList<AbstractLocalEventBooking>
+getUsers() : ArrayList<User>
+getLocalEvents() : ArrayList<LocalEvent>
+getUserWithID(id : String) : User
+getHotelRoomWithID(id : String) : HotelRoom
+getLocalEventWithID(String id) : LocalEvent
+getHotelRoomBookingWithId(String id) : AbstractHotelRoomBooking
+getLocalEventBookingWithId(String id) : AbstractLocalEventBooking
+takeFromDatabaseHotelRooms(query : String) : void
+takeFromHotelProvider() : void
+takeFromDatabaseLocalEvents(query : String) : void
+takeFromLocalEventProvider() : void
+takeFromDatabaseHotelBookings(query : String) : void
+takeFromDatabaseUsers(query : String) : void
+takeFromDatabaseLocalEventBookings(query : String) : void
+updateDataBase(query : String) : void
AbstractLocalEventBooking
#bookingID : String
#userID : String
#localEventID : String
#fees : double
+AbstractLocalEventBooking(userID : String, LocalEventID, fees : Double)
+getBookingID() : String
+getUserID() : String
+getLocalEventID() : String
+getFees() : double
AbstractHotelRoomBooking
#bookingID : String
#checkInDate : LocalDate
#checkOutDate : LocalDate
#userID : String
#hotelRoomID : String
#hotel : String
#fees : double
+getBookingID() : String
+getCheckInDate() : LocalDate
+getCheckOutDate() : LocalDate
+getUserID() : String
+getHotelRoomID() : String
+getHotel() : String
+getFees() : double
+AbstractHotelRoomBooking(checkInDate : LocalDate, checkoutDate : LocalDate, userID : String, hotelRoomID : String, hotel : String, fees : double)
LocalEvent
#Address : String
#City : String
#LocalEventID : String
#Organizer : String
#name : String
#price : Double
#numOfTickets : int
+getAddress() : String
+setAddress(Address : String) : void
+getCity() : String
User
#username : String
#password : String
#userID : String
#Mail : String
#phoneNumber : String
#isLoggedIn : boolean
#recommendedEvents : ArrayList<LocalEvent>
+getUsername() : String
+setUsername(username : String) : void
+getPassword() : String
+setPassword(password : String) : void
+getUserID() : String
+getMail() : String
+setMail(Mail : String) : void
+getPhoneNumber() : String
+setPhoneNumber(phoneNumber : String) : void
+getIsLoggedIn() : boolean
+setIsLoggedIn(isLoggedIn : boolean) : void
+getRecommendedEvents() : ArrayList<LocalEvent>
+setRecommendedEvents(recommendedEvents : ArrayList<LocalEvent>) : void
+User(username : String, password : String, Mail : String, phoneNumber : String, isLoggedIn : boolean)
HotelRoom
#Hotel : String
#Address : String
#City : String
#HotelRoomID : String
#Available : boolean
#name : String
#price : Double
+getHotel() : String
+setHotel(Hotel : String) : void
+getAddress()
+setAddress(Address) : void
+getCity()
+setCity(City) : void
+getHoteRoomID()
+HotelRoom(Hotel : String, City : String, Address : String, Available : boolean, name : String, price : Double)
+getAvailable() : boolean
+setAvailable(Available : boolean) : void
+getName() : String
+setName(name : String) : void
+getPrice() : Double
+setPrice(price : Double) : void
NormalLocalEvent
+NormalLocalEvent(Address : String, City : String, Organizer : String, name : String, price : Double, numOfTickets : int)
StandardUser
+StandardUser(username : String, password : String, Mail : String, phoneNumber : String, isLoggedIn : boolean)
StandardRoom
+StandardRoom(Hotel : String, City : String, Address : String, Available : boolean, name : String, price : Double)

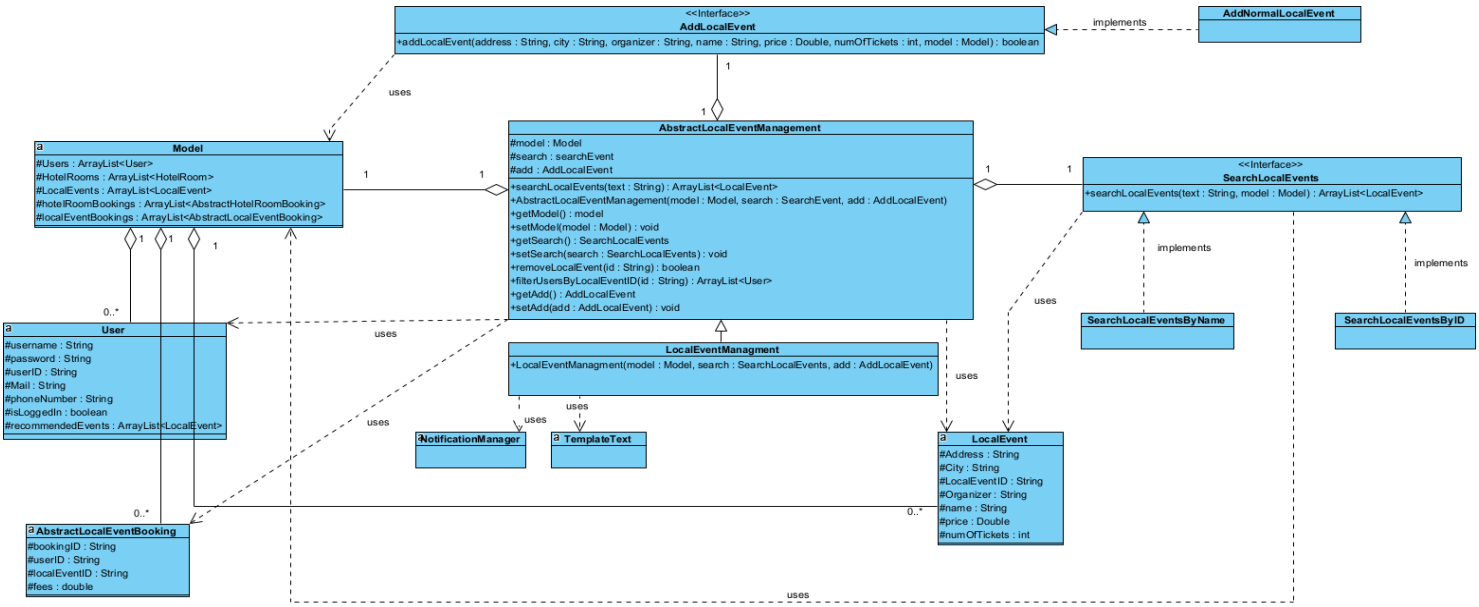
HotelManagment:

Design Patterns used: Strategy

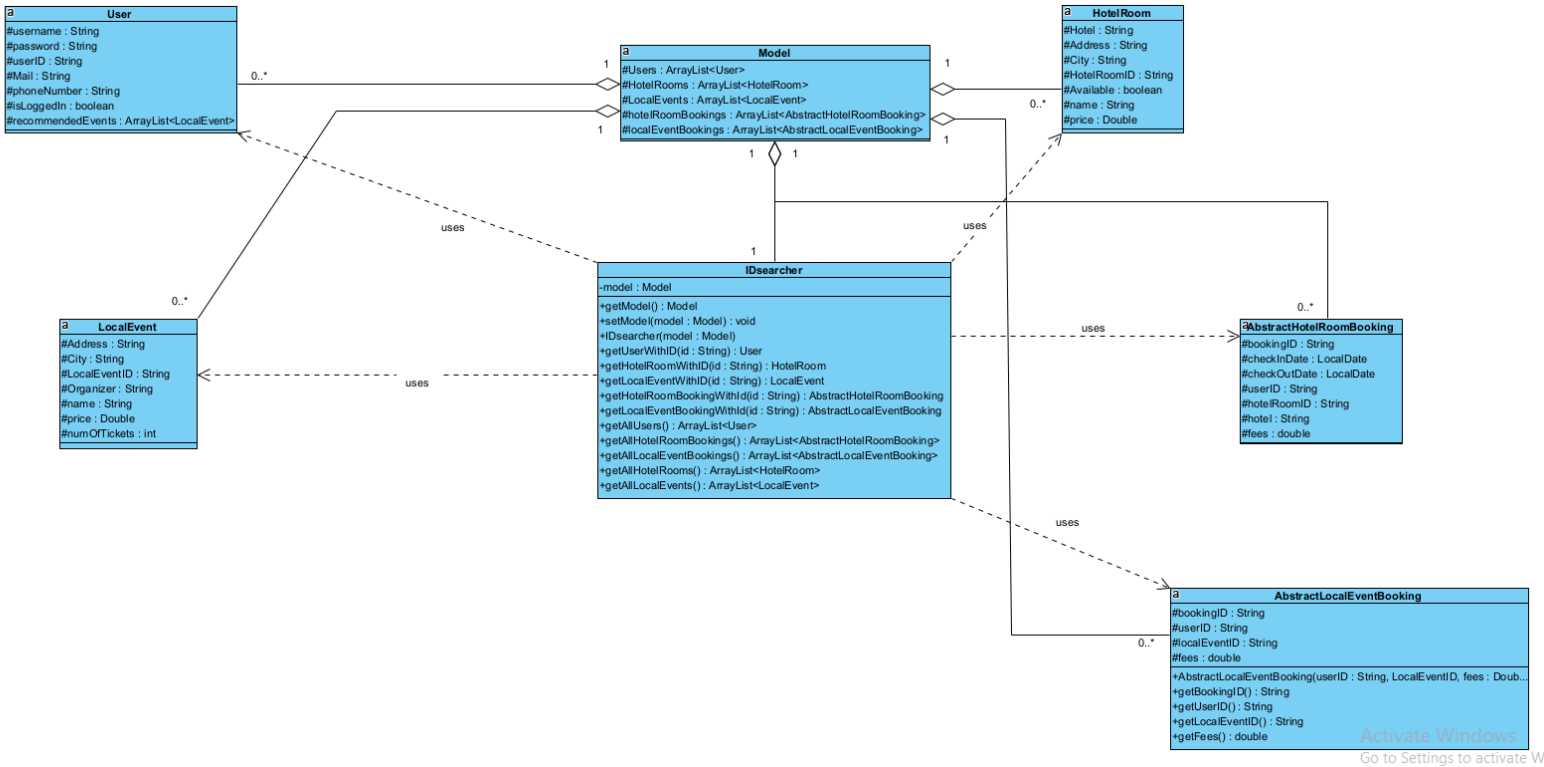


LocalEventManagment:

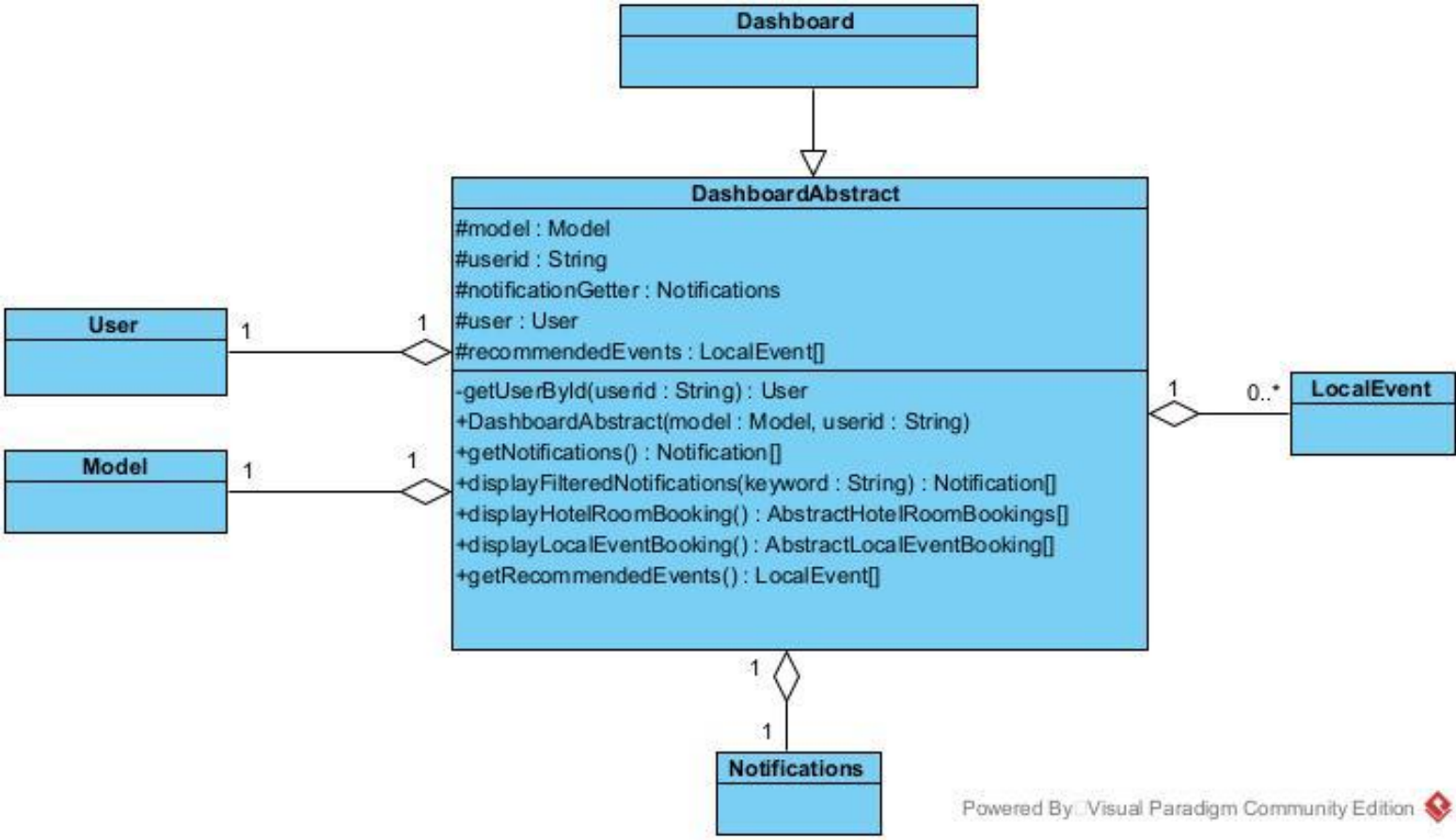
Design Patterns used: Strategy



IDsearcher:

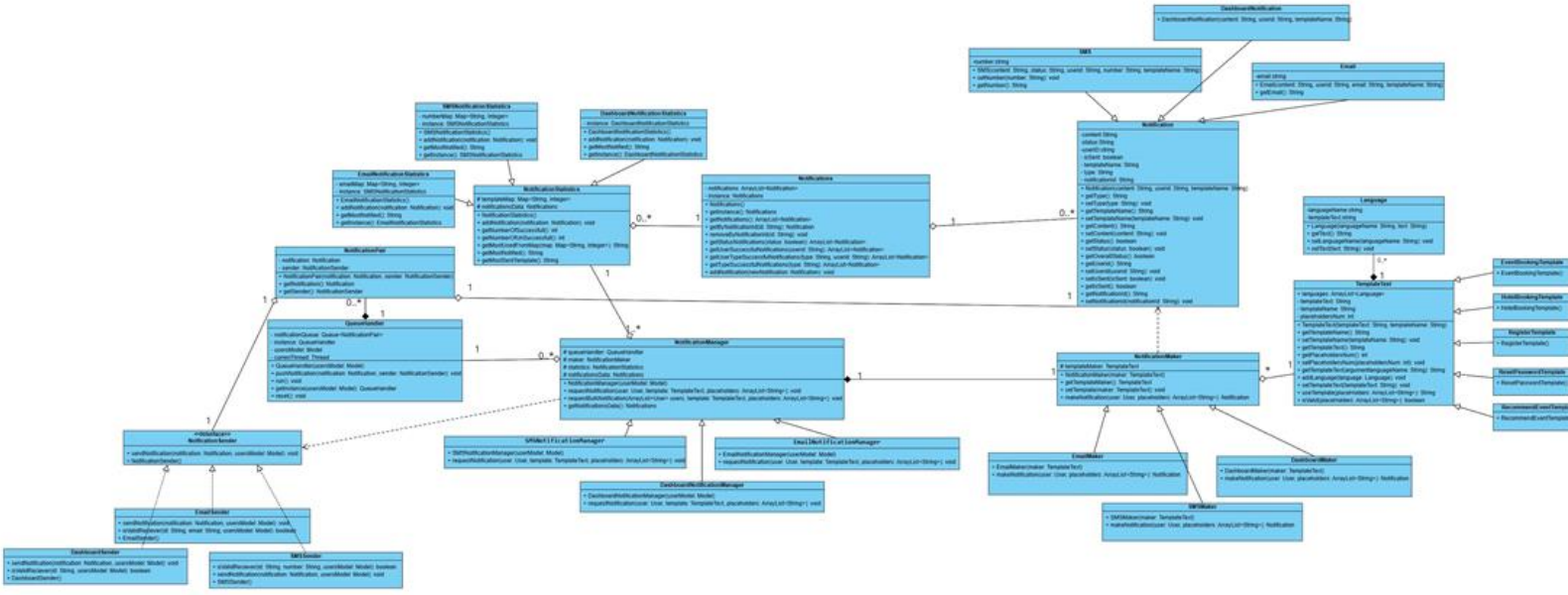


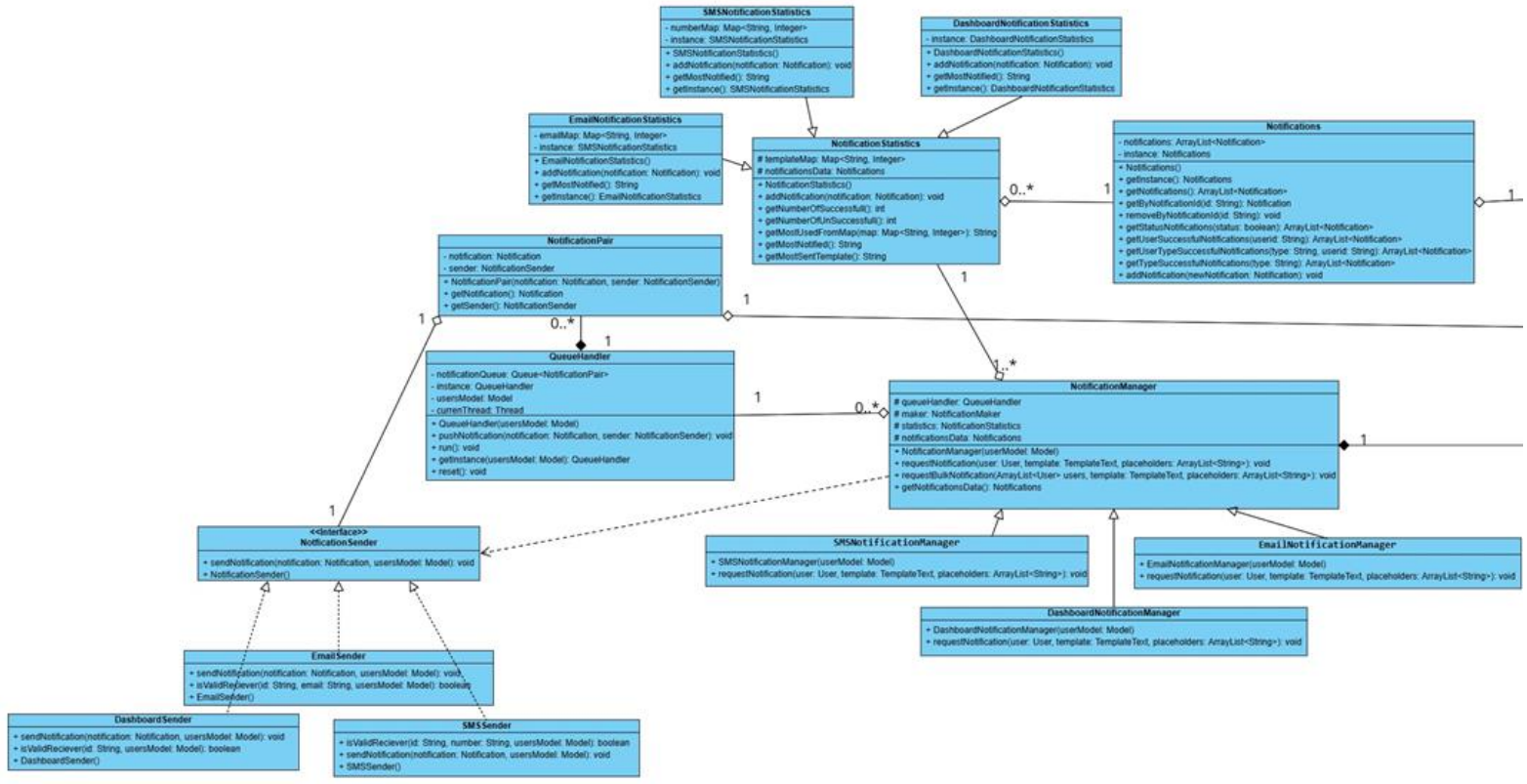
Dashboard:

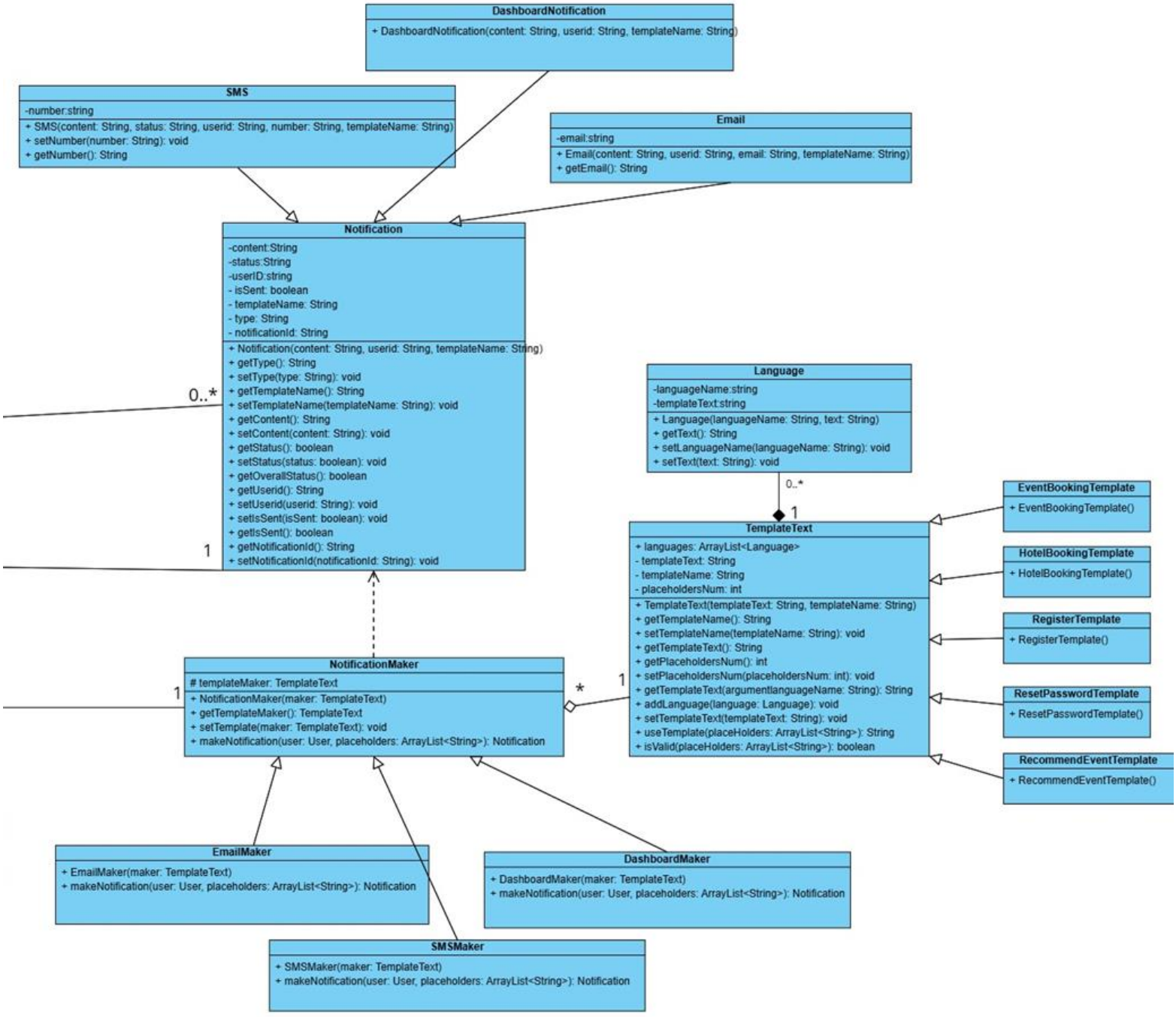

Dashboard
DashboardAbstract
#model : Model
#userid : String
#notificationGetter : Notifications
#user : User
#recommendedEvents : LocalEvent[]
-getUserById(userid : String) : User
+DashboardAbstract(model : Model, userid : String)
+getNotifications() : Notification[]
+displayFilteredNotifications(keyword : String) : Notification[]
+displayHotelRoomBooking() : AbstractHotelRoomBookings[]
+displayLocalEventBooking() : AbstractLocalEventBooking[]
+getRecommendedEvents() : LocalEvent[]
User
Model
LocalEvent
Notifications
1
1
1
1
1
0..*
1
1
Powered By Visual Paradigm Community Edition

Notification Module:

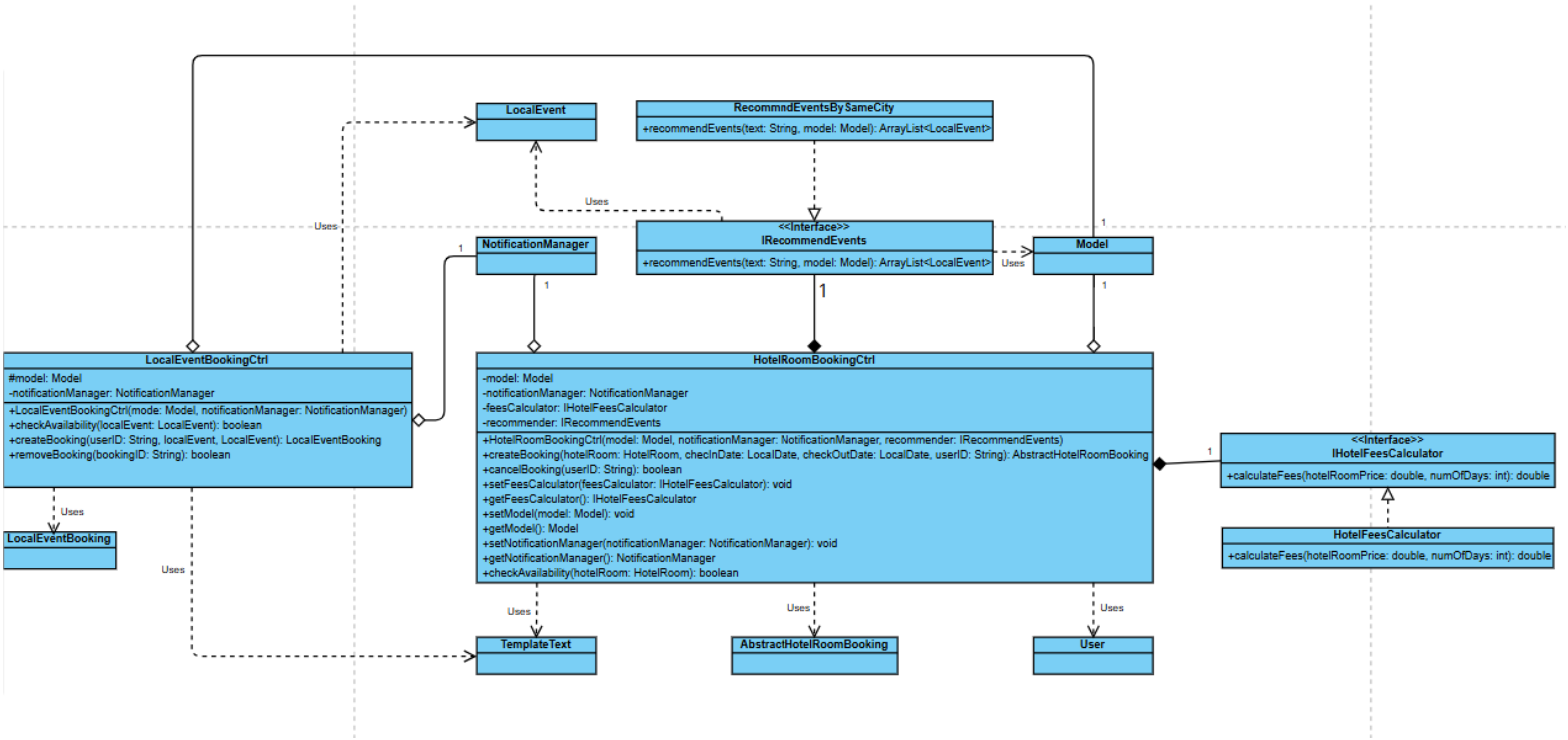
Desing patterns used: strategy pattern, factory pattern, bridge pattern.



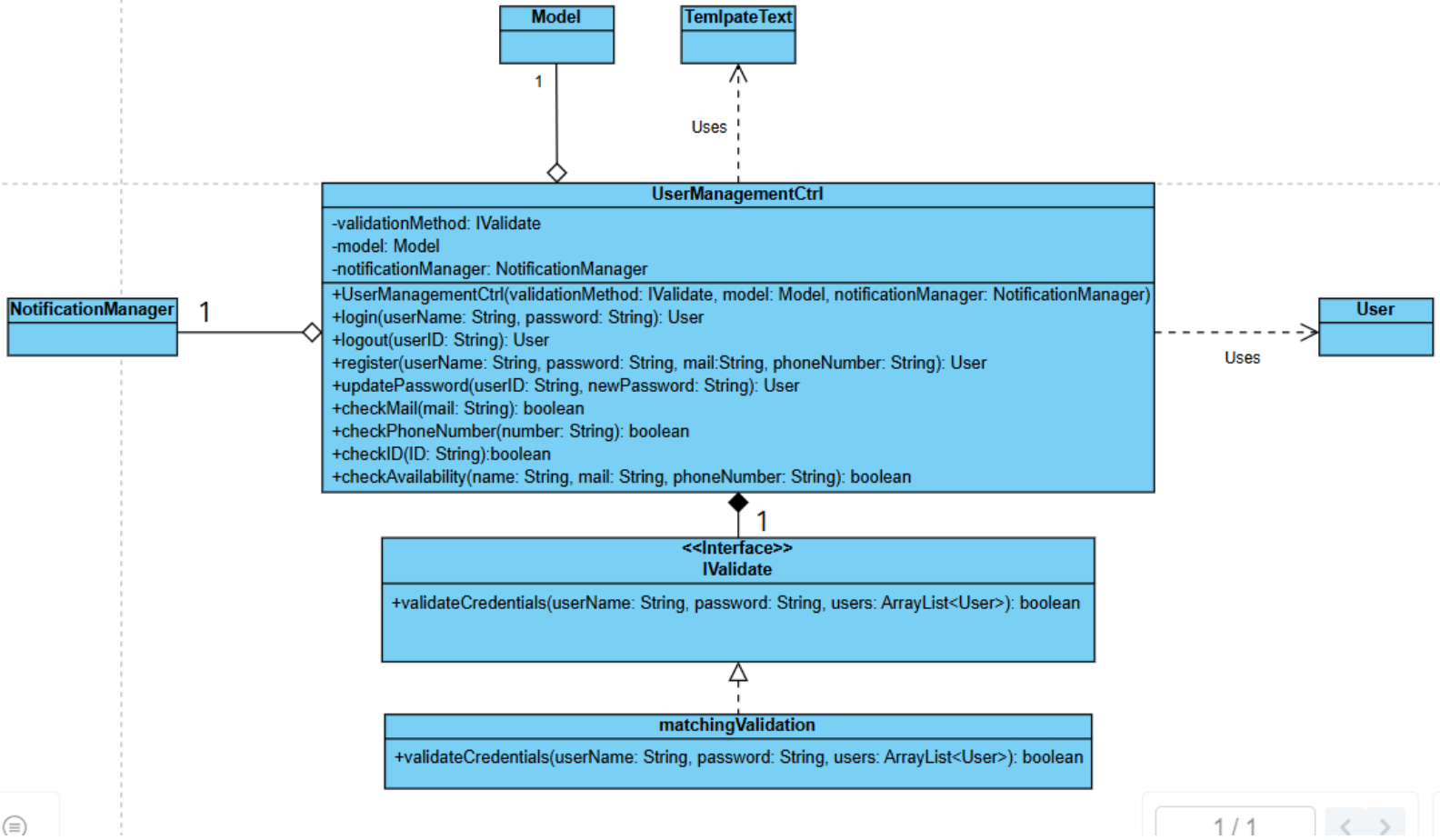



DashboardNotification
+ DashboardNotification(content: String, userid: String, templateName: String)
SMS
-number:string
+ SMS(content: String, status: String, userid: String, number: String, templateName: String)
+ setNumber(number: String): void
+ getNumber(): String
Email
-email:string
+ Email(content: String, userid: String, email: String, templateName: String)
+ getEmail(): String
Notification
-content:String
-status:String
-userID:string
- isSent: boolean
- templateName: String
- type: String
- notificationId: String
+ Notification(content: String, userid: String, templateName: String)
+ getType(): String
+ setType(type: String): void
+ getTemplateName(): String
+ setTemplateName(templateName: String): void
+ getContent(): String
+ setContent(content: String): void
+ getStatus(): boolean
+ setStatus(status: boolean): void
+ getOverallStatus(): boolean
+ getUserid(): String
+ setUserid(userid: String): void
+ setIsSent(isSent: boolean): void
+ getIsSent(): boolean
+ getNotificationId(): String
+ setNotificationId(notificationId: String): void
0..*
1
Language
-languageName:string
-templateText:string
+ Language(languageName: String, text: String)
+ getText(): String
+ setLanguageName(languageName: String): void
+ setText(text: String): void
0..*
1
TemplateText
+ languages: ArrayList<Language>
- templateText: String
- templateName: String
- placeholdersNum: int
+ TemplateText(templateText: String, templateName: String)
+ getTemplateName(): String
+ setTemplateName(templateName: String): void
+ getTemplateText(): String
+ getPlaceholdersNum(): int
+ setPlaceholdersNum(placeholdersNum: int): void
+ getTemplateText(argumentlanguageName: String): String
+ addLanguage(language: Language): void
+ setTemplateText(templateText: String): void
+ useTemplate(placeHolders: ArrayList<String>): String
+ isValid(placeHolders: ArrayList<String>): boolean
EventBookingTemplate
+ EventBookingTemplate()
HotelBookingTemplate
+ HotelBookingTemplate()
RegisterTemplate
+ RegisterTemplate()
ResetPasswordTemplate
+ ResetPasswordTemplate()
RecommendEventTemplate
+ RecommendEventTemplate()
NotificationMaker
templateMaker: TemplateText
+ NotificationMaker(maker: TemplateText)
+ getTemplateMaker(): TemplateText
+ setTemplate(maker: TemplateText): void
+ makeNotification(user: User, placeholders: ArrayList<String>): Notification
1
*
1
EmailMaker
+ EmailMaker(maker: TemplateText)
+ makeNotification(user: User, placeholders: ArrayList<String>): Notification
DashboardMaker
+ DashboardMaker(maker: TemplateText)
+ makeNotification(user: User, placeholders: ArrayList<String>): Notification
SMSMaker
+ SMSMaker(maker: TemplateText)
+ makeNotification(user: User, placeholders: ArrayList<String>): Notification

Booking:

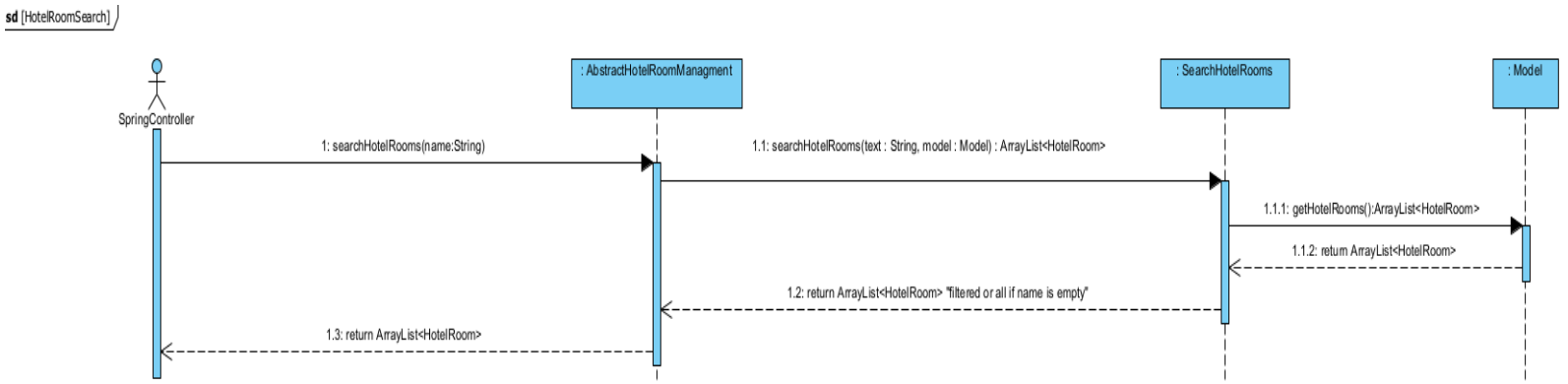

LocalEvent
RecommndEventsBySameCity
+recommendEvents(text: String, model: Model): ArrayList<LocalEvent>
Uses
Uses
<<Interface>>
IRecommendEvents
+recommendEvents(text: String, model: Model): ArrayList<LocalEvent>
Uses
1
NotificationManager
Model
1
1
1
1
LocalEventBookingCtrl
#model: Model
-notificationManager: NotificationManager
+LocalEventBookingCtrl(mode: Model, notificationManager: NotificationManager)
+checkAvailability(localEvent: LocalEvent): boolean
+createBooking(userID: String, localEvent, LocalEvent): LocalEventBooking
+removeBooking(bookingID: String): boolean
HotelRoomBookingCtrl
-model: Model
-notificationManager: NotificationManager
-feesCalculator: IHotelFeesCalculator
-recommender: IRecommendEvents
+HotelRoomBookingCtrl(model: Model, notificationManager: NotificationManager, recommender: IRecommendEvents)
+createBooking(hotelRoom: HotelRoom, checInDate: LocalDate, checkOutDate: LocalDate, userID: String): AbstractHotelRoomBooking
+cancelBooking(userID: String): boolean
+setFeesCalculator(feesCalculator: IHotelFeesCalculator): void
+getFeesCalculator(): IHotelFeesCalculator
+setModel(model: Model): void
+getModel(): Model
+setNotificationManager(notificationManager: NotificationManager): void
+getNotificationManager(): NotificationManager
+checkAvailability(hotelRoom: HotelRoom): boolean
1
<<Interface>>
IHotelFeesCalculator
+calculateFees(hotelRoomPrice: double, numOfDays: int): double
HotelFeesCalculator
+calculateFees(hotelRoomPrice: double, numOfDays: int): double
Uses
LocalEventBooking
Uses
Uses
Uses
Uses
TemplateText
AbstractHotelRoomBooking
User

UserManagement:

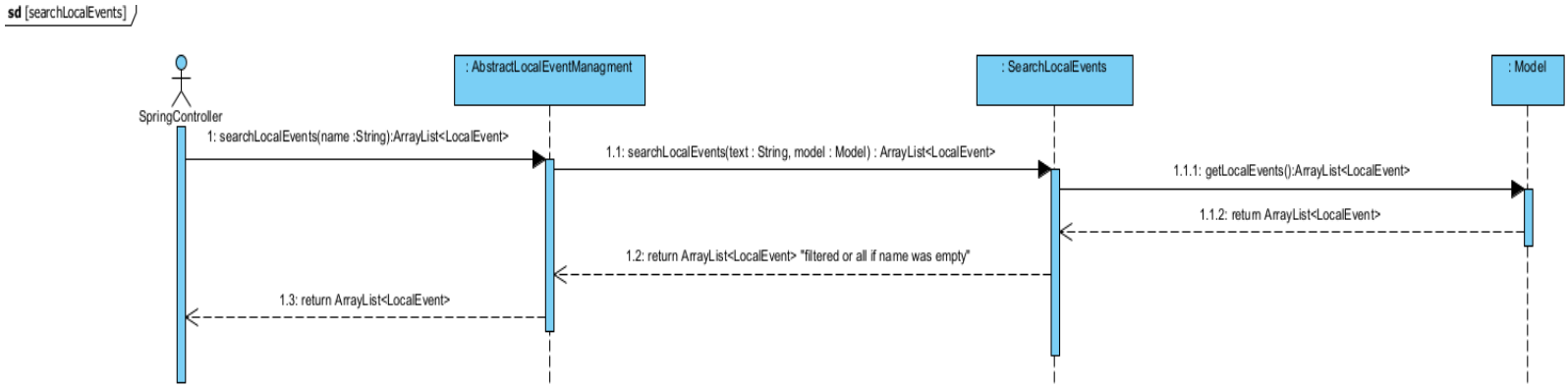


Sequence Diagrams:

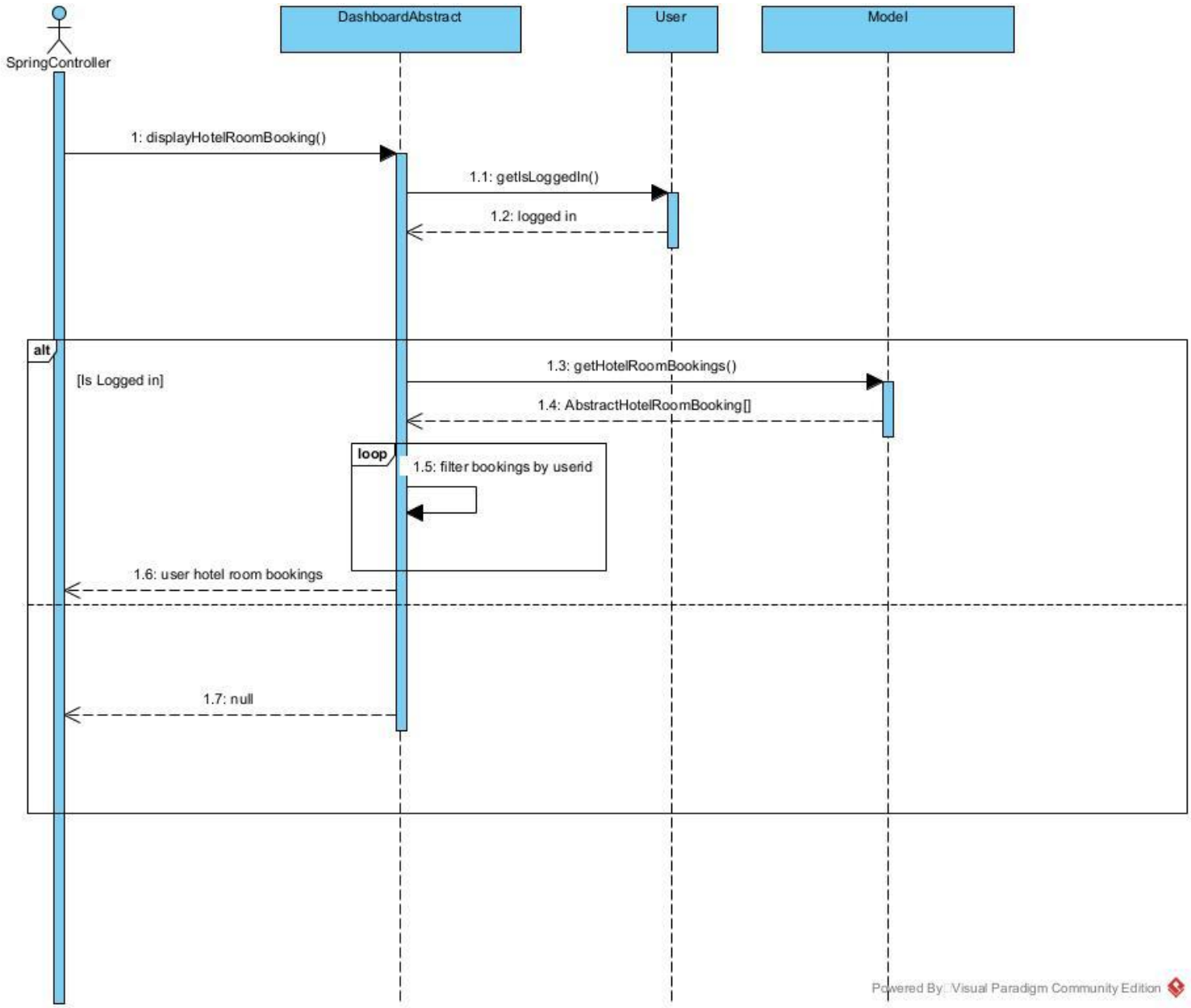
Hotel Room Search:



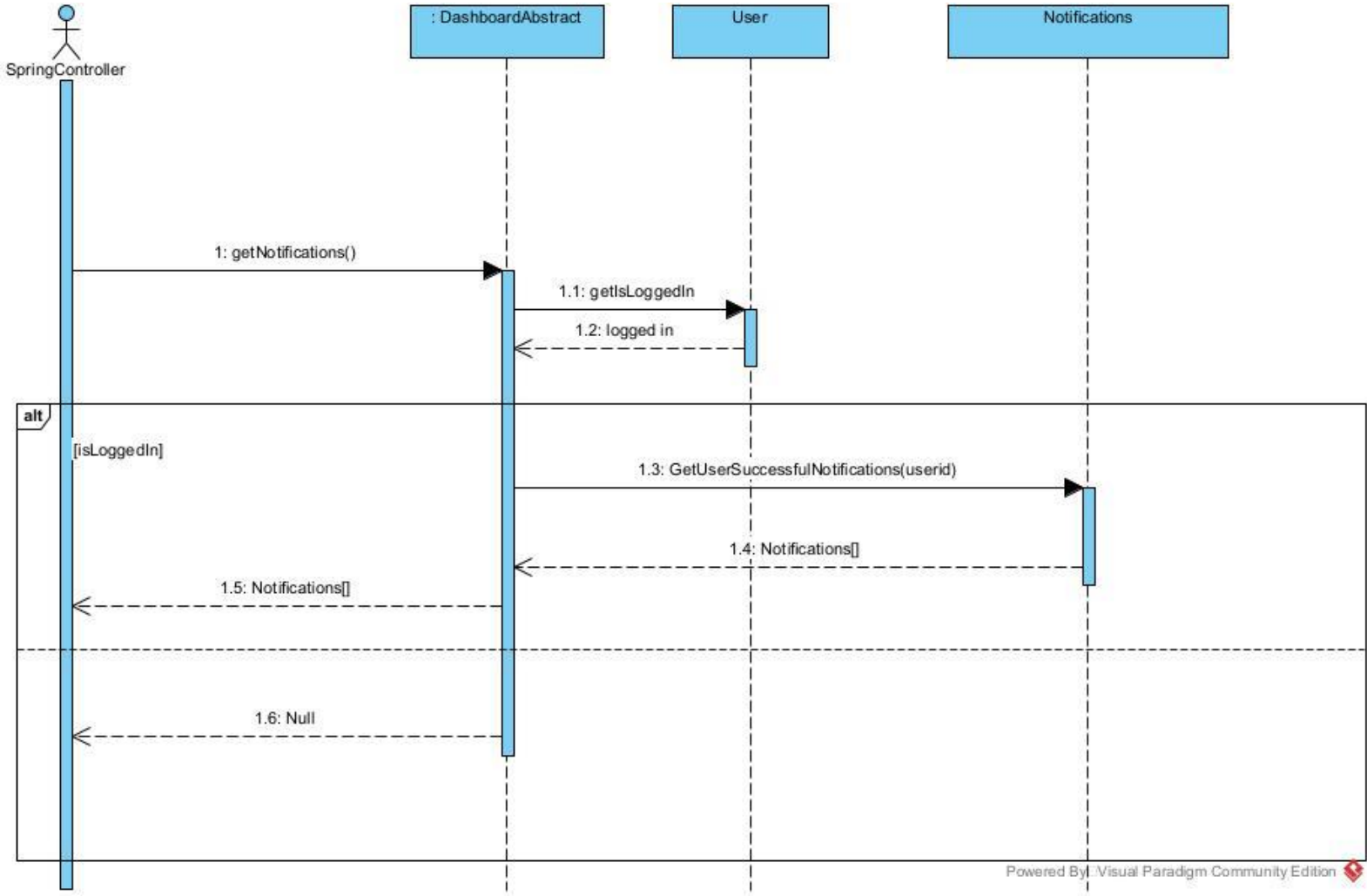
Search Local Events:



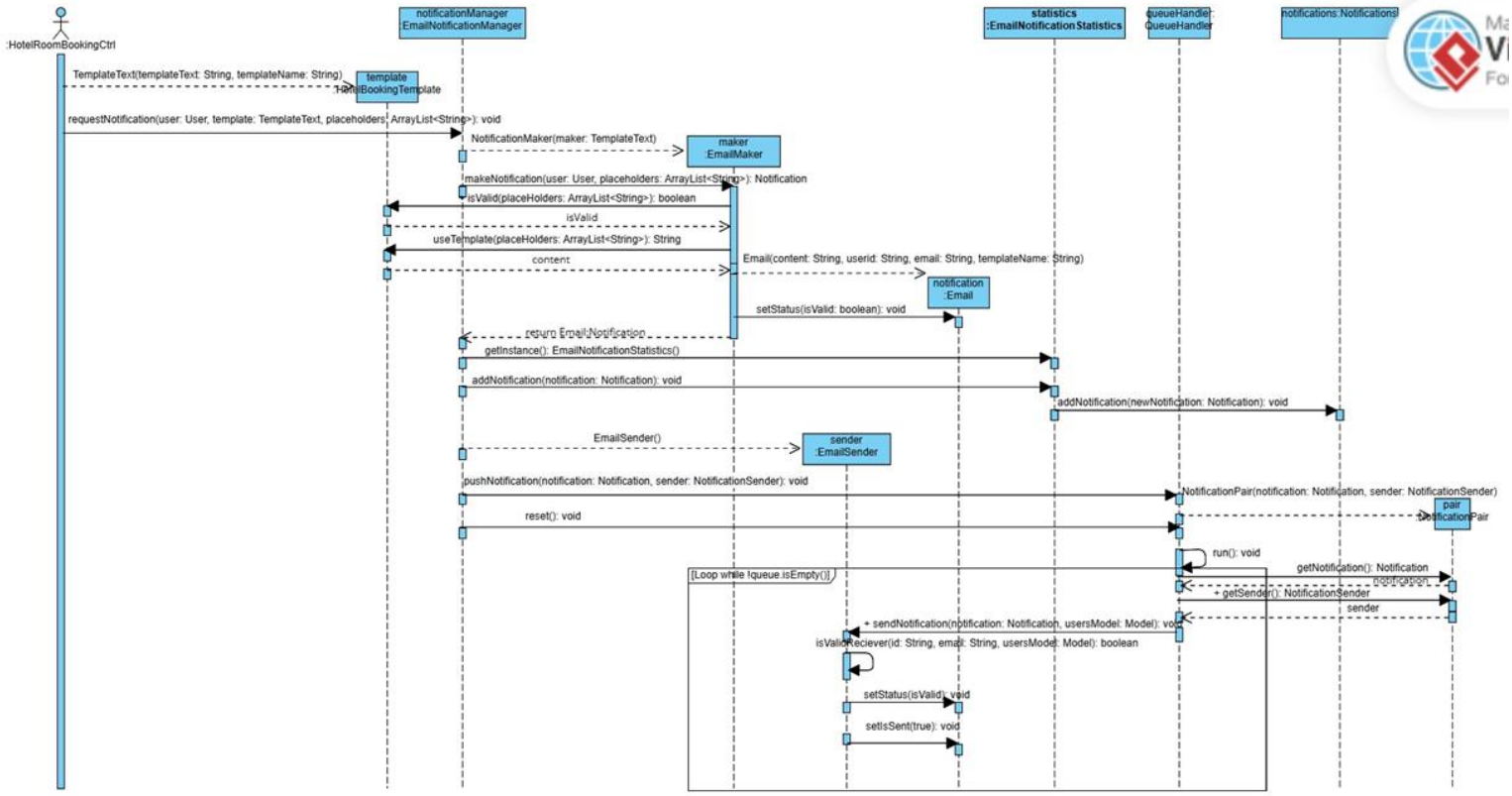
Retrieving Hotel Room Bookings:


SpringController
DashboardAbstract
User
Model
1: displayHotelRoomBooking()
1.1: getIsLoggedIn()
1.2: logged in
alt
[Is Logged in]
1.3: getHotelRoomBookings()
1.4: AbstractHotelRoomBooking[]
loop
1.5: filter bookings by userid
1.6: user hotel room bookings
1.7: null
Powered By: Visual Paradigm Community Edition

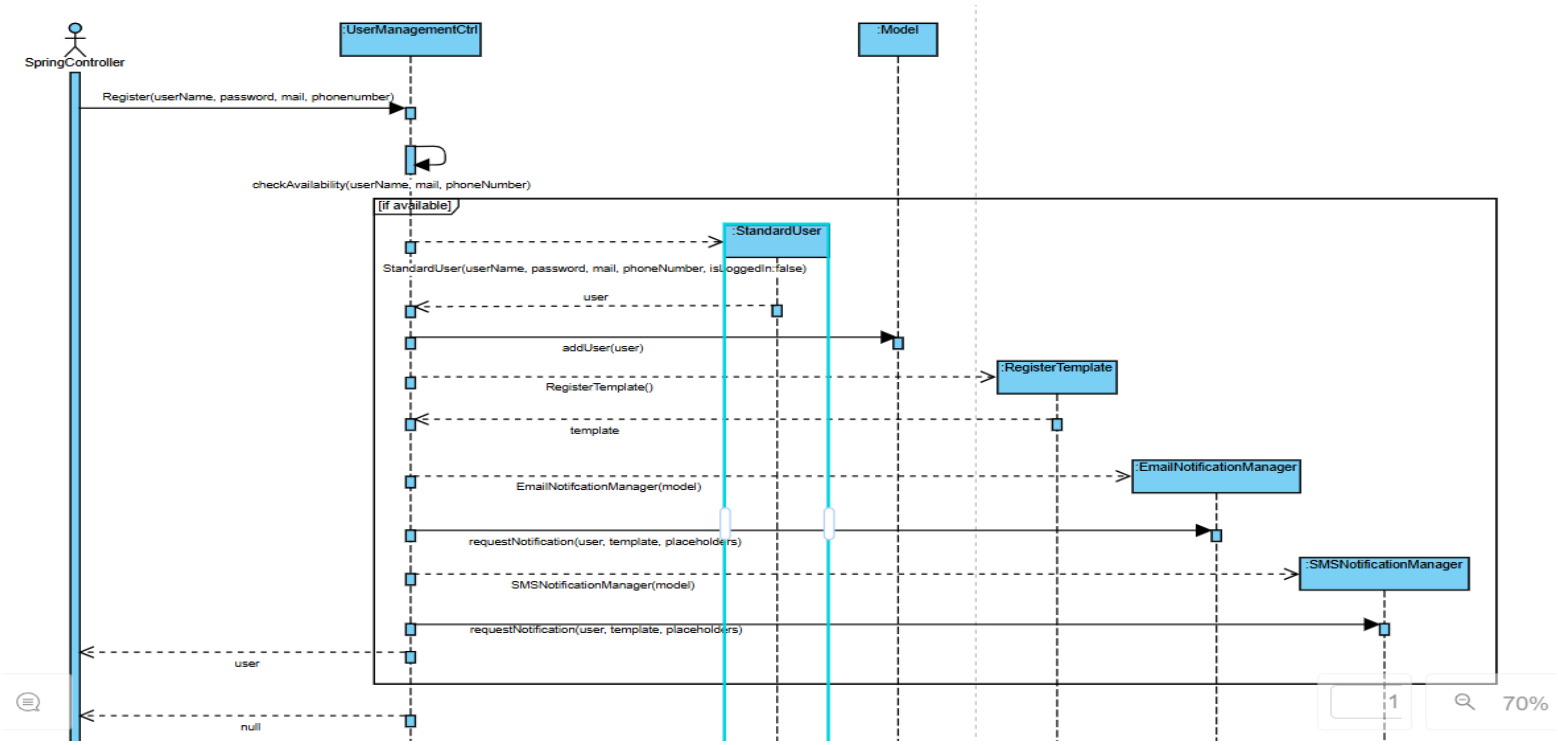
Retrieving User Notifications:



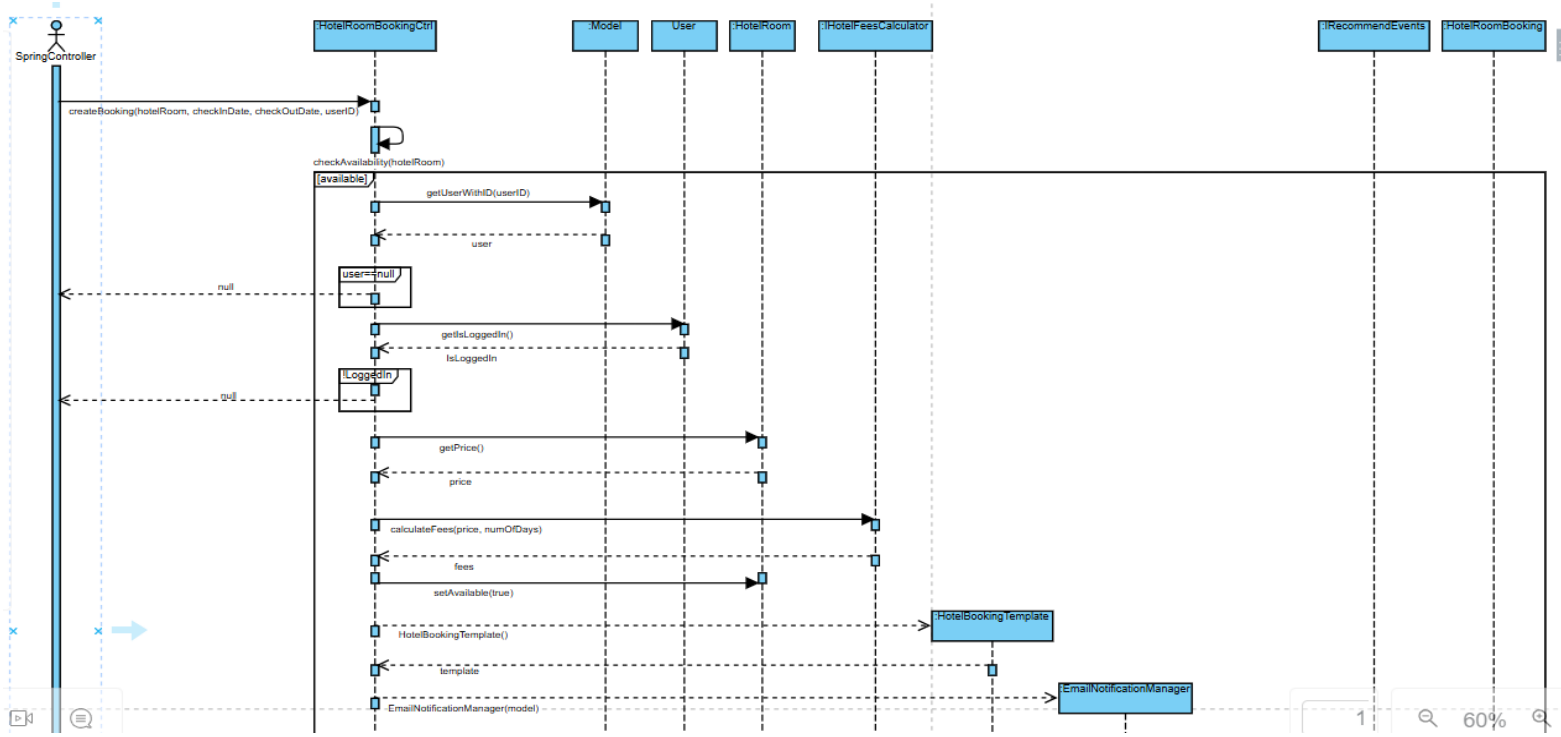
Send room booking email notification:

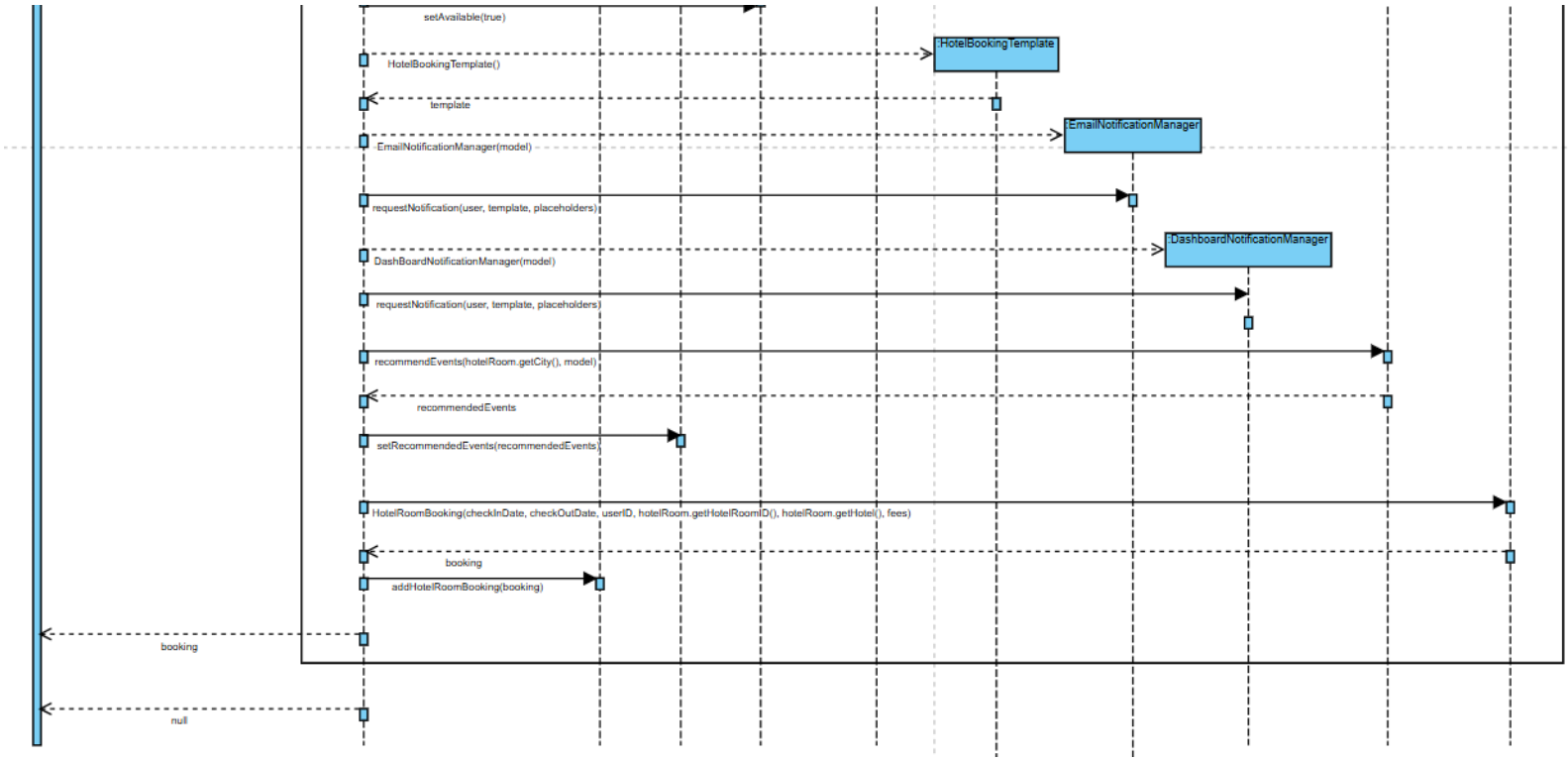


Register:



Create Booking:





Repository Code Link:
https://github.com/ZiadMedhat33/TravelAgencySDA

Phase 1 Updates:

- Switching architecture style from Event Based to layered architecture and thus we had to update every diagram to suit the new architecture style and every view.
- Update on assumptions.
- Update on scenarios.
- Update on functional requirements.
- Update on Design process and rationale.