Project Part - 6: Final Report

- **Team :**
  Ashwin Sankaralingam
  Harivignesh Rajaram
  Ilamvazhuthy Subbiah
- **Title: BookIT**
- **Project Summary:**

  BookIT is a one stop destination for all your entertainment ticketing needs from concert ticket to NFL game tickets. You can also signup as an event Manager to host your own events.

Features that were implemented:

User Requirements

ID	Requirements	Topic Area	User	Priority
UR-001	User should be able to create an account	Authentication	User	Medium
UR-002	User should be able to login into his account	Authentication	User	High
UR-003	As a user I should be able to update my profile	Profile	Customer, Event Manager	High
UR-004	As a user I should be able to search for events based on some criteria	Search	Customer	High
UR-005	As a user I should be able to view the details of the event I am interested in	View Event	Customer	High
UR-006	As a user I should be able add events to my wishlist	Wishlist	Customer	Medium
UR-007	As a user I should be able to book tickets for the events I am interested in	Booking	Customer	High
UR-008	As a user I should be able to view my booking history	View History	Customer	Medium
UR-009	As a user I should be able to cancel my tickets	Cancellation	Customer	High

UR-010	As an Event Manager I should be able to signup and login	Authentication	Event Manager	Medium
UR-011	As an Event Manager I should be able to add events along with the event information	Add Events	Event Manager	High
UR-012	As an Event Manager I should be able to update details of an event	Update Events	Event Manager	Medium
UR-013	As an Event Manager I should be able to view the bookings so far	View Bookings	Event Manager	Medium
UR-014	As an Admin I should be able to login	Authentication	Admin	Medium
UR-015	As an Admin I should be able to approve the event manager	Approval	Admin	High

Functional Requirements

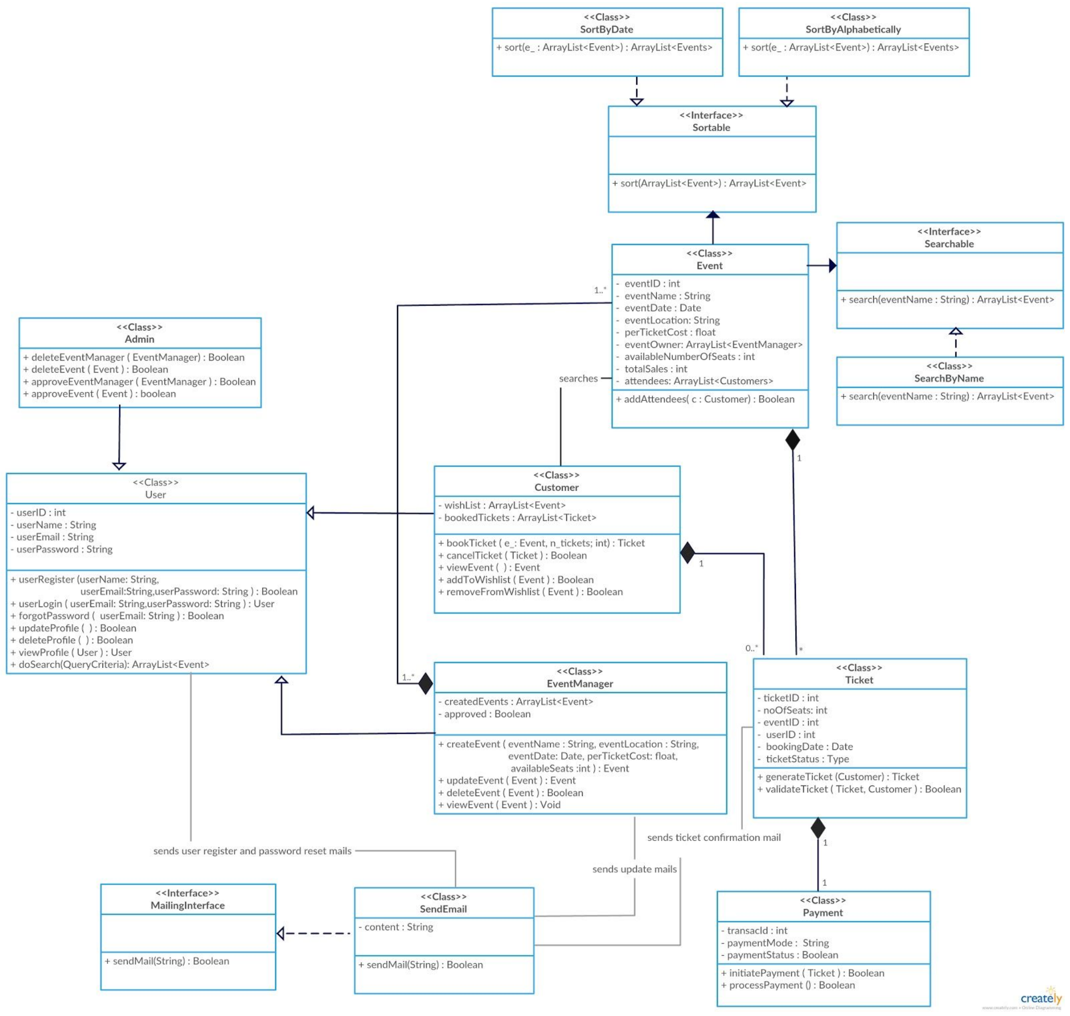
ID	Requirements	Topic Area	User	Priority
FR-001	A verification email should be sent to the user when they sign up	Login	User, Event Manager	High
FR-002	A link to reset password should be mailed registered email ID if the user forgets his password	Authentication	All	Medium
FR-003	A notification email should be sent to the user when a booking is successful	Booking	User	High
FR-004	A notification email should be sent to the user when there is a change in the event information	Event Update	User	High
FR-005	A notification email should be sent to the user upon successful cancellation	Cancellation	User	High
FR-006	A notification email should be sent to the user if any event in the user's wish list is approaching or is going to run out of tickets	Wishlist	User	Medium

Features that were not implemented

Non- Functional Requirements

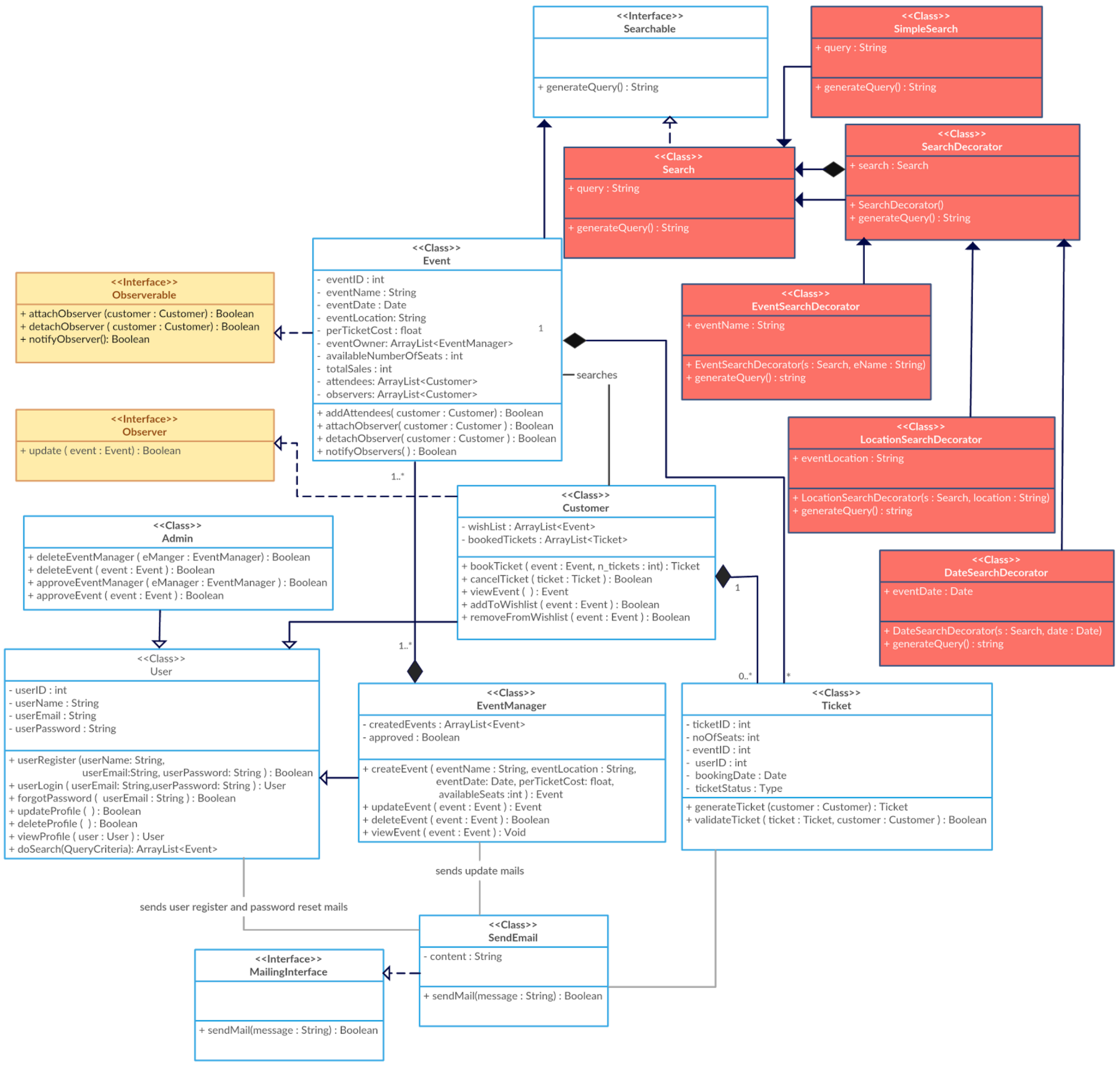
ID	Requirements	Topic Area	User	Priority
NFR-001	Double booking should be handled in an appropriate manner	Reliability	User	High
NFR-002	User account information must be stored in a secure manner	Security	All	High
NFR-003	All user searches must be logged	Search	User	Low

Part 2 Class Diagram(OLD)



<<Class>> SortByDate
+ sort(e_ : ArrayList<Event>) : ArrayList<Events>
<<Class>> SortByAlphabetically
+ sort(e_ : ArrayList<Event>) : ArrayList<Events>
<<Interface>> Sortable
+ sort(ArrayList<Event>) : ArrayList<Event>
<<Interface>> Searchable
+ search(eventName : String) : ArrayList<Event>
<<Class>> Event
- eventID : int
- eventName : String
- eventDate : Date
- eventLocation: String
- perTicketCost : float
- eventOwner: ArrayList<EventManager>
- availableNumberOfSeats : int
- totalSales : int
- attendees: ArrayList<Customers>
+ addAttendees(c : Customer) : Boolean
<<Class>> SearchByName
+ search(eventName : String) : ArrayList<Event>
1..*
searches
1
<<Class>> Admin
+ deleteEventManager (EventManager) : Boolean
+ deleteEvent (Event) : Boolean
+ approveEventManager (EventManager) : Boolean
+ approveEvent (Event) : boolean
<<Class>> User
- userID : int
- userName : String
- userEmail : String
- userPassword : String
+ userRegister (userName: String, userEmail:String,userPassword: String) : Boolean
+ userLogin (userEmail: String,userPassword: String) : User
+ forgotPassword (userEmail: String) : Boolean
+ updateProfile ( ) : Boolean
+ deleteProfile ( ) : Boolean
+ viewProfile (User) : User
+ doSearch(QueryCriteria): ArrayList<Event>
<<Class>> Customer
- wishList : ArrayList<Event>
- bookedTickets : ArrayList<Ticket>
+ bookTicket (e_: Event, n_tickets; int) : Ticket
+ cancelTicket (Ticket) : Boolean
+ viewEvent ( ) : Event
+ addToWishlist (Event) : Boolean
+ removeFromWishlist (Event) : Boolean
1
0..*
*
1..*
<<Class>> EventManager
- createdEvents : ArrayList<Event>
- approved : Boolean
+ createEvent (eventName : String, eventLocation : String, eventDate: Date, perTicketCost: float, availableSeats :int) : Event
+ updateEvent (Event) : Event
+ deleteEvent (Event) : Boolean
+ viewEvent (Event) : Void
<<Class>> Ticket
- ticketID : int
- noOfSeats: int
- eventID : int
- userID : int
- bookingDate : Date
- ticketStatus : Type
+ generateTicket (Customer) : Ticket
+ validateTicket (Ticket, Customer) : Boolean
sends ticket confirmation mail
sends user register and password reset mails
sends update mails
1
1
<<Interface>> MailingInterface
+ sendMail(String) : Boolean
<<Class>> SendEmail
- content : String
+ sendMail(String) : Boolean
<<Class>> Payment
- transacId : int
- paymentMode : String
- paymentStatus : Boolean
+ initiatePayment (Ticket) : Boolean
+ processPayment () : Boolean
creately

After Refactoring the Class diagram:



<<Interface>>
Searchable
+ generateQuery() : String
<<Class>>
SimpleSearch
+ query : String
+ generateQuery() : String
<<Class>>
SearchDecorator
+ search : Search
+ SearchDecorator()
+ generateQuery() : String
<<Class>>
Search
+ query : String
+ generateQuery() : String
<<Class>>
Event
- eventID : int
- eventName : String
- eventDate : Date
- eventLocation: String
- perTicketCost : float
- eventOwner: ArrayList<EventManager>
- availableNumberOfSeats : int
- totalSales : int
- attendees: ArrayList<Customer>
- observers: ArrayList<Customer>
+ addAttendees(customer : Customer) : Boolean
+ attachObserver(customer : Customer) : Boolean
+ detachObserver(customer : Customer) : Boolean
+ notifyObservers( ) : Boolean
1
searches
1..*
<<Interface>>
Observerable
+ attachObserver (customer : Customer) : Boolean
+ detachObserver (customer : Customer) : Boolean
+ notifyObserver(): Boolean
<<Interface>>
Observer
+ update (event : Event) : Boolean
<<Class>>
EventSearchDecorator
+ eventName : String
+ EventSearchDecorator(s : Search, eName : String)
+ generateQuery() : string
<<Class>>
LocationSearchDecorator
+ eventLocation : String
+ LocationSearchDecorator(s : Search, location : String)
+ generateQuery() : string
<<Class>>
DateSearchDecorator
+ eventDate : Date
+ DateSearchDecorator(s : Search, date : Date)
+ generateQuery() : string
<<Class>>
Customer
- wishList : ArrayList<Event>
- bookedTickets : ArrayList<Ticket>
+ bookTicket (event : Event, n_tickets : int) : Ticket
+ cancelTicket (ticket : Ticket) : Boolean
+ viewEvent ( ) : Event
+ addToWishlist (event : Event) : Boolean
+ removeFromWishlist (event : Event) : Boolean
1
<<Class>>
Admin
+ deleteEventManager (eManger : EventManager) : Boolean
+ deleteEvent (event : Event) : Boolean
+ approveEventManager (eManager : EventManager) : Boolean
+ approveEvent (event : Event) : Boolean
1..*
0..*
*
<<Class>>
User
- userID : int
- userName : String
- userEmail : String
- userPassword : String
+ userRegister (userName: String, userEmail:String, userPassword: String) : Boolean
+ userLogin (userEmail: String,userPassword: String) : User
+ forgotPassword (userEmail : String) : Boolean
+ updateProfile ( ) : Boolean
+ deleteProfile ( ) : Boolean
+ viewProfile (user : User) : User
+ doSearch(QueryCriteria): ArrayList<Event>
<<Class>>
EventManager
- createdEvents : ArrayList<Event>
- approved : Boolean
+ createEvent (eventName : String, eventLocation : String, eventDate: Date, perTicketCost: float, availableSeats :int) : Event
+ updateEvent (event : Event) : Event
+ deleteEvent (event : Event) : Boolean
+ viewEvent (event : Event) : Void
<<Class>>
Ticket
- ticketID : int
- noOfSeats: int
- eventID : int
- userID : int
- bookingDate : Date
- ticketStatus : Type
+ generateTicket (customer : Customer) : Ticket
+ validateTicket (ticket : Ticket, customer : Customer) : Boolean
sends update mails
sends user register and password reset mails
<<Class>>
SendEmail
- content : String
+ sendMail(message : String) : Boolean
<<Interface>>
MailingInterface
+ sendMail(message : String) : Boolean

Major Changes:

1. During the refactoring , we were able to add classes that facilitated the use of Design Patterns.
2. The major functionality of the class did not change. We added extra functions to extend the functionality.
3. Taking a lot of time discussing amongst the team helped us a lot to have a good start with the design aspect that reduce the constant refactoring. During the implementation we were

able to work in different phase of code simultaneously due the perfect planning in the design phase and understand the dependency in the project.

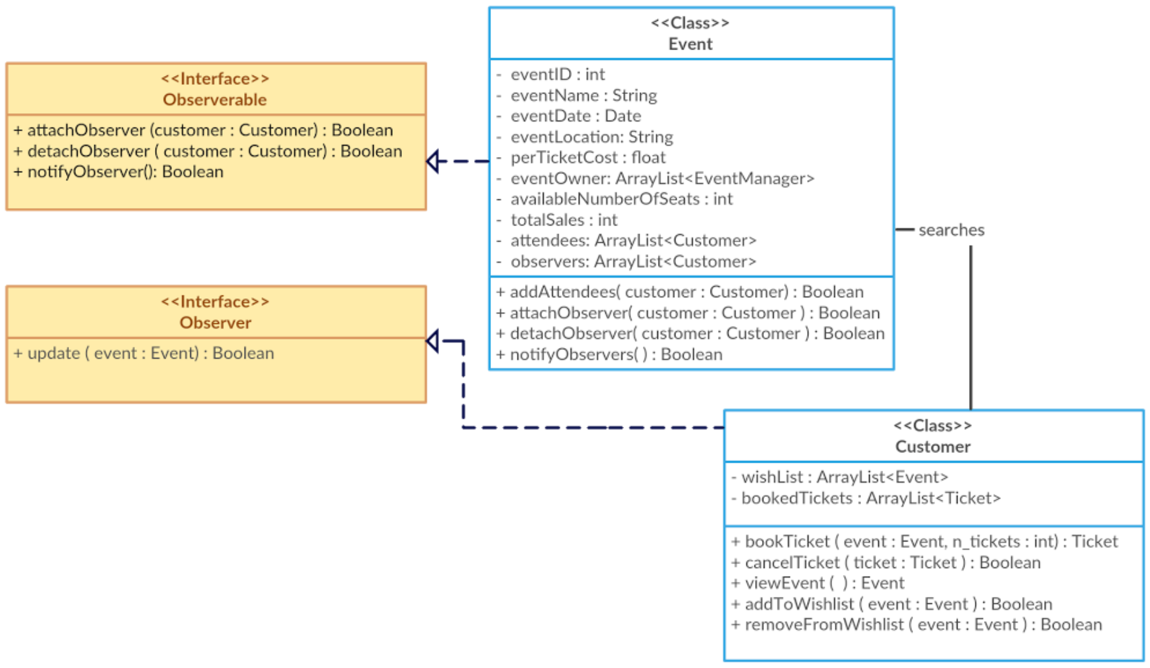
Classes that were not implemented:

1. *Sortable , SortBydate, SortAlphabetically :* As play framework provided us with an ORM inbuilt sorting functionality, we did not see the need to reimplement it.

Design Pattern used:

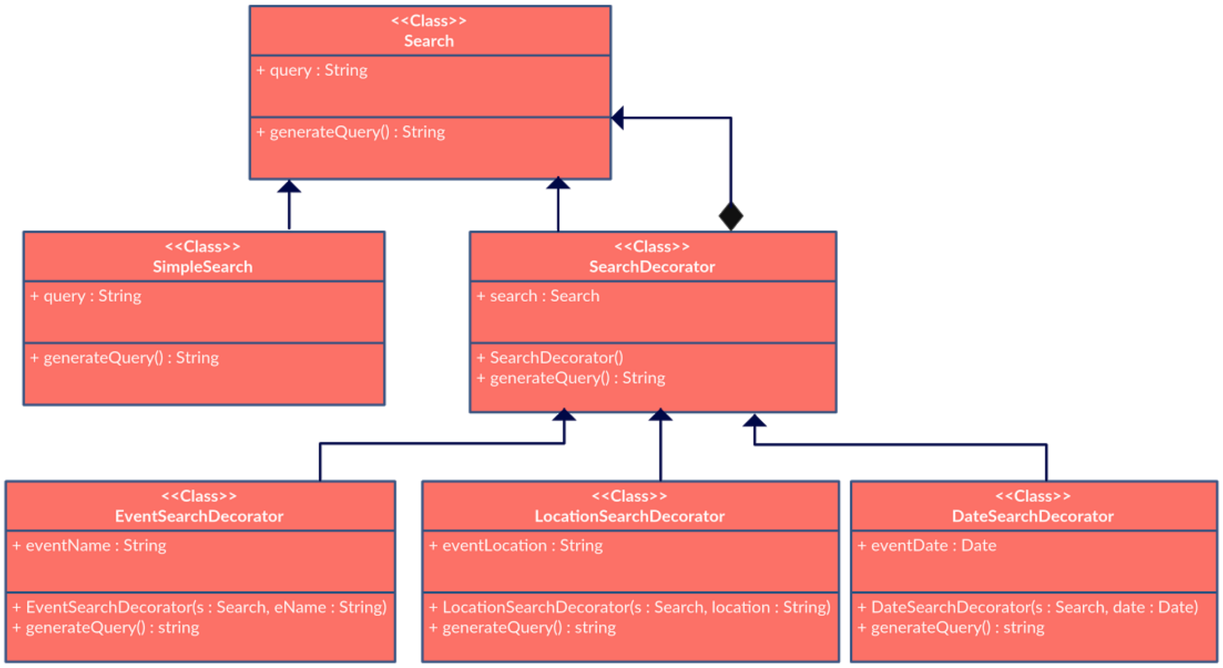
Observer Pattern for notifying the change in the event:

Implementing the Observer Pattern lets the interested customer be notified whenever there is any change in the event details like change in location or time, etc.
The Customer can either add an event to his wishlist or book tickets for the event. In either of the cases, he is an interested party regarding the event and hence should be notified of any changes. The Event class is implemented as the Subject class which knows its list of observers (The customers) and provides functionalities for attaching or detaching an observer. The Event class then notifies all the observers if there is any change in the event details.


<<Interface>>
Observerable
+ attachObserver (customer : Customer) : Boolean
+ detachObserver (customer : Customer) : Boolean
+ notifyObserver(): Boolean
<<Interface>>
Observer
+ update (event : Event) : Boolean
<<Class>>
Event
- eventID : int
- eventName : String
- eventDate : Date
- eventLocation: String
- perTicketCost : float
- eventOwner: ArrayList<EventManager>
- availableNumberOfSeats : int
- totalSales : int
- attendees: ArrayList<Customer>
- observers: ArrayList<Customer>
+ addAttendees(customer : Customer) : Boolean
+ attachObserver(customer : Customer) : Boolean
+ detachObserver(customer : Customer) : Boolean
+ notifyObservers( ) : Boolean
searches
<<Class>>
Customer
- wishList : ArrayList<Event>
- bookedTickets : ArrayList<Ticket>
+ bookTicket (event : Event, n_tickets : int) : Ticket
+ cancelTicket (ticket : Ticket) : Boolean
+ viewEvent ( ) : Event
+ addToWishlist (event : Event) : Boolean
+ removeFromWishlist (event : Event) : Boolean

Decorator Pattern for applying multiple filters:

The decorator pattern can be used when a user searches for events based on a search criteria. The user can couple many search criteria like location, date or the event name when he/she performs the search. The decorator pattern can be employed to handle these couplings in an effective manner.



<<Class>>
Search
+ query : String
+ generateQuery() : String
<<Class>>
SimpleSearch
+ query : String
+ generateQuery() : String
<<Class>>
SearchDecorator
+ search : Search
+ SearchDecorator()
+ generateQuery() : String
<<Class>>
EventSearchDecorator
+ eventName : String
+ EventSearchDecorator(s : Search, eName : String)
+ generateQuery() : string
<<Class>>
LocationSearchDecorator
+ eventLocation : String
+ LocationSearchDecorator(s : Search, location : String)
+ generateQuery() : string
<<Class>>
DateSearchDecorator
+ eventDate : Date
+ DateSearchDecorator(s : Search, date : Date)
+ generateQuery() : string

What problem did the design pattern solve?

1. **Observer:** Instead of querying the db every time the event gets updated, we now have a List of all the users associated with that event. We just have to call the notify event for the List of Users.
2. **Decorator:** Instead of having separate sub-class for every search combination, we have one decorator for every parameter. It is easier to add new search parameters in future.

What we learnt from the process?

- Working as a team and coming together discussing the design even before thinking about implementation.
- Understanding the trade-offs involved in feasibility while designing and the relationship between various diagrams.
- Leveraging patterns to make the implementation process smoother and having a long-term mindset while coding.
- Code collaboration and merging conflicts while simultaneous coding.
- Experimenting pair programming technique during the start of the project.
- Learning and relearning new frameworks like play in a short period of time.