OOAD PROJECT FINAL REPORT (BUFF BEDS)

Team:
Gaurav Gandhi
Sarang Kulkarni
Akshit Shah

Title: Buff Beds
Description: A hotel/accommodation booking website, where End Users can have access to large number of houses and hotels for accommodation. Also, house owners can post their houses for renting.

1. List the features that were implemented (table with ID and title).

ID	Requirement	Topic Area	Actor	Priority
UR-01	Customer can sign up.	Login	Customer, Host	Critical
UR-02	Host can Apply for hotel/SignUp	Login	Host, Admin	Critical
UR - 03	End Users can log in.	Login	Customer, Hosts,Admin	Critical
UR – 04	Admin can block both hosts and Customers	Login	Admin	High
UR – 05	Customer can search and sort hotels or rooms	Database	Customer	High
UR – 06	Customer can checkout the hotel/room	Orders	Customer	High

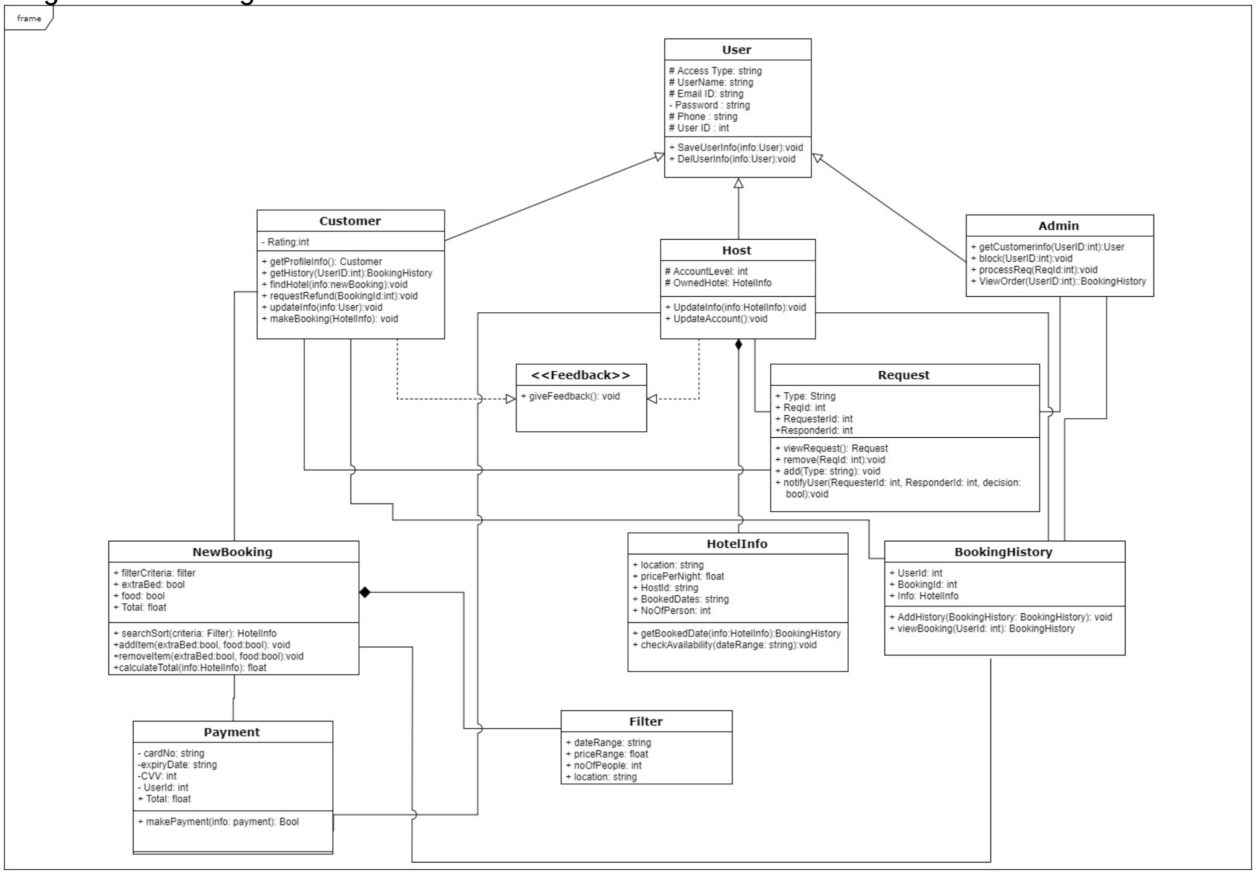
UR – 07	Customer cancel the booking	Orders	Customer	High
UR – 09	Hosts can upgrade their account	Database	Hosts	High
UR – 12	Admin can approve request for new hosts/request for update information	Database	Admin	High

2. List the features were not implemented from Part 2 (table with ID and title).

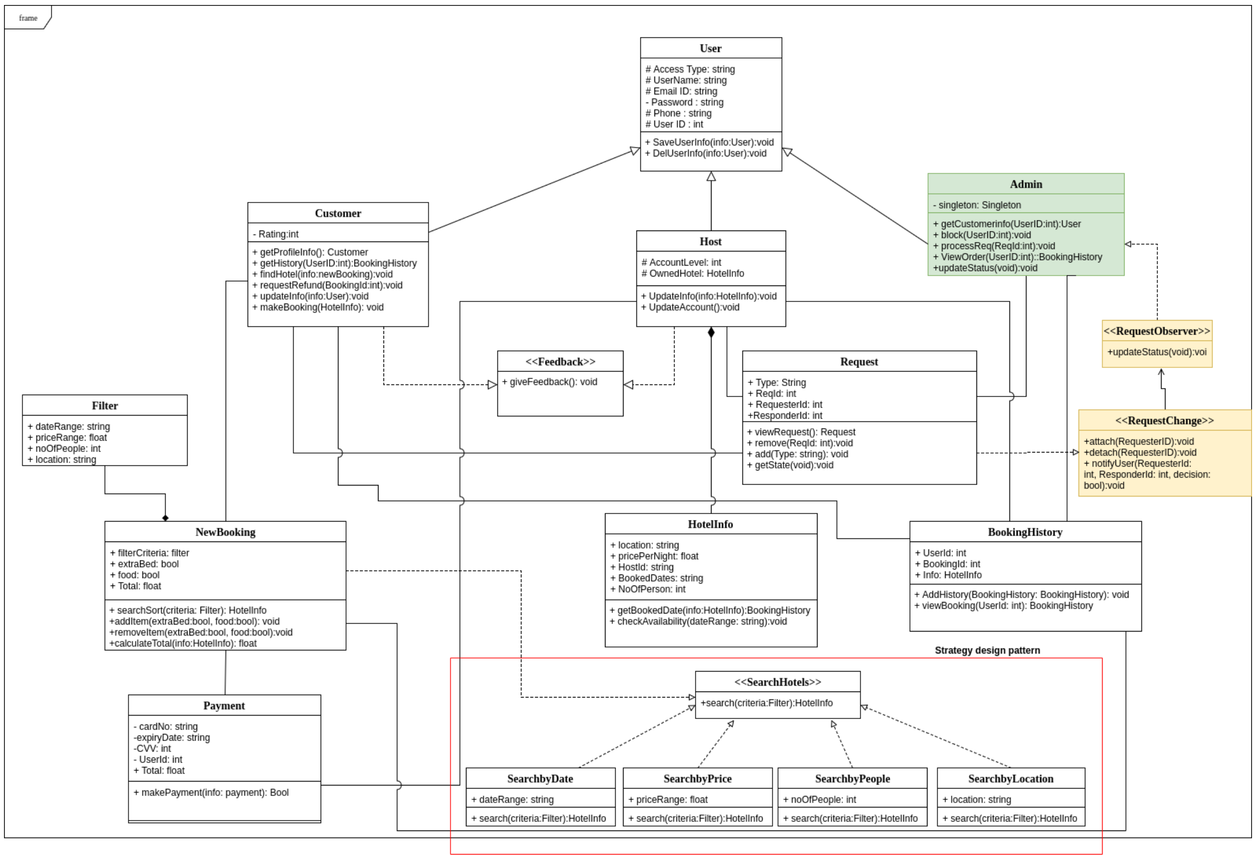
ID	Requirement	Topic Area	Actor	Priority
UR – 08	Customer can request addons to their orders	Orders	Customer, Host	Low
UR - 10	Host can update the info	Profile	Host	Medium
UR -11	End User can rate and give feedbacks	Database	Customer, Host	Medium
UR - 13	Admin can approve a refund	Orders	Admin	Medium

3. Show your Part 2 class diagram and your final class diagram. What changed? Why? If it did not change much, then discuss how doing the design up front helped in the development.

Original Class Diagram


frame
User
# Access Type: string
# UserName: string
# Email ID: string
- Password : string
# Phone : string
# User ID : int
+ SaveUserInfo(info:User):void
+ DelUserInfo(info:User):void
Customer
- Rating:int
+ getProfileInfo(): Customer
+ getHistory(UserID:int):BookingHistory
+ findHotel(info:newBooking):void
+ requestRefund(BookingId:int):void
+ updateInfo(info:User):void
+ makeBooking(HotelInfo): void
Host
# AccountLevel: int
# OwnedHotel: HotelInfo
+ UpdateInfo(info:HotelInfo):void
+ UpdateAccount():void
Admin
+ getCustomerinfo(UserID:int):User
+ block(UserID:int):void
+ processReq(ReqId:int):void
+ ViewOrder(UserID:int)::BookingHistory
<<Feedback>>
+ giveFeedback(): void
Request
+ Type: String
+ ReqId: int
+ RequesterId: int
+ResponderId: int
+ viewRequest(): Request
+ remove(ReqId: int):void
+ add(Type: string): void
+ notifyUser(RequesterId: int, ResponderId: int, decision: bool):void
NewBooking
+ filterCriteria: filter
+ extraBed: bool
+ food: bool
+ Total: float
+ searchSort(criteria: Filter): HotelInfo
+addItem(extraBed:bool, food:bool): void
+removeItem(extraBed:bool, food:bool):void
+calculateTotal(info:HotelInfo): float
HotelInfo
+ location: string
+ pricePerNight: float
+ HostId: string
+ BookedDates: string
+ NoOfPerson: int
+ getBookedDate(info:HotelInfo):BookingHistory
+ checkAvailability(dateRange: string):void
BookingHistory
+ UserId: int
+ BookingId: int
+ Info: HotelInfo
+ AddHistory(BookingHistory: BookingHistory): void
+ viewBooking(UserId: int): BookingHistory
Payment
- cardNo: string
-expiryDate: string
-CVV: int
- UserId: int
+ Total: float
+ makePayment(info: payment): Bool
Filter
+ dateRange: string
+ priceRange: float
+ noOfPeople: int
+ location: string

Final Class Diagram

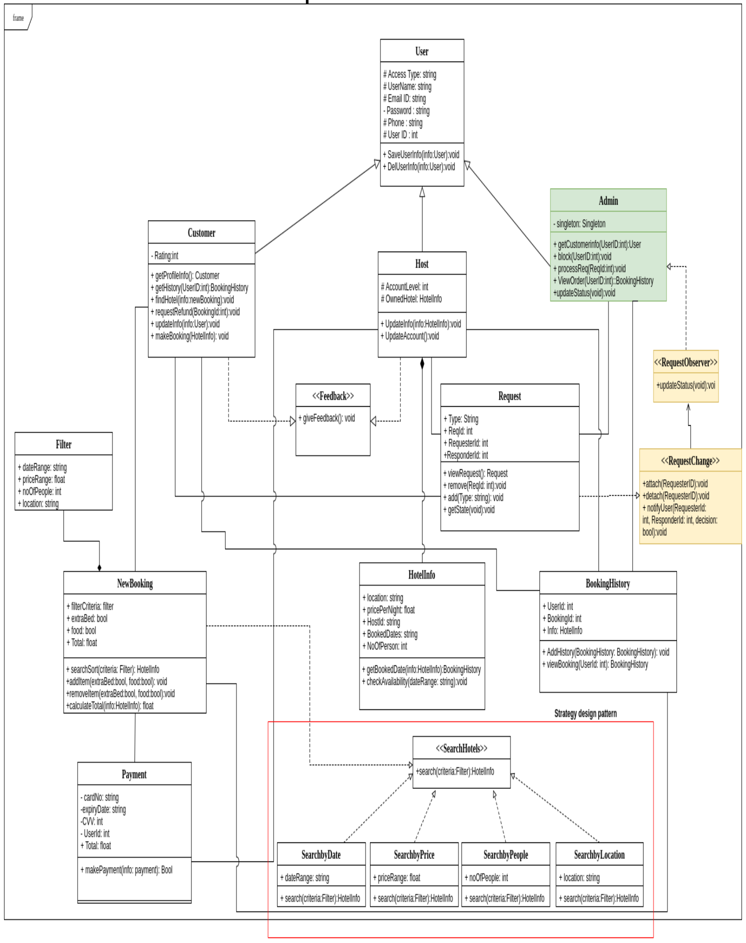

frame
User
# Access Type: string
# UserName: string
# Email ID: string
- Password : string
# Phone : string
# User ID : int
+ SaveUserInfo(info:User):void
+ DelUserInfo(info:User):void
Admin
- singleton: Singleton
+ getCustomerinfo(UserID:int):User
+ block(UserID:int):void
+ processReq(ReqId:int):void
+ ViewOrder(UserID:int)::BookingHistory
+updateStatus(void):void
Customer
- Rating:int
+ getProfileInfo(): Customer
+ getHistory(UserID:int):BookingHistory
+ findHotel(info:newBooking):void
+ requestRefund(BookingId:int):void
+ updateInfo(info:User):void
+ makeBooking(HotelInfo): void
Host
# AccountLevel: int
# OwnedHotel: HotelInfo
+ UpdateInfo(info:HotelInfo):void
+ UpdateAccount():void
<<RequestObserver>>
+updateStatus(void):voi
<<Feedback>>
+ giveFeedback(): void
Request
+ Type: String
+ ReqId: int
+ RequesterId: int
+ResponderId: int
+ viewRequest(): Request
+ remove(ReqId: int):void
+ add(Type: string): void
+ getState(void):void
<<RequestChange>>
+attach(RequesterID):void
+detach(RequesterID):void
+ notifyUser(RequesterId: int, ResponderId: int, decision: bool):void
Filter
+ dateRange: string
+ priceRange: float
+ noOfPeople: int
+ location: string
NewBooking
+ filterCriteria: filter
+ extraBed: bool
+ food: bool
+ Total: float
+ searchSort(criteria: Filter): HotelInfo
+addItem(extraBed:bool, food:bool): void
+removeItem(extraBed:bool, food:bool):void
+calculateTotal(info:HotelInfo): float
HotelInfo
+ location: string
+ pricePerNight: float
+ HostId: string
+ BookedDates: string
+ NoOfPerson: int
+ getBookedDate(info:HotelInfo):BookingHistory
+ checkAvailability(dateRange: string):void
BookingHistory
+ UserId: int
+ BookingId: int
+ Info: HotelInfo
+ AddHistory(BookingHistory: BookingHistory): void
+ viewBooking(UserId: int): BookingHistory
Strategy design pattern
<<SearchHotels>>
+search(criteria:Filter):HotelInfo
Payment
- cardNo: string
-expiryDate: string
-CVV: int
- UserId: int
+ Total: float
+ makePayment(info: payment): Bool
SearchbyDate
+ dateRange: string
+ search(criteria:Filter):HotelInfo
SearchbyPrice
+ priceRange: float
+ search(criteria:Filter):HotelInfo
SearchbyPeople
+ noOfPeople: int
+ search(criteria:Filter):HotelInfo
SearchbyLocation
+ location: string
+ search(criteria:Filter):HotelInfo

Peach Color - Observer Pattern
Green Color – Singleton Pattern

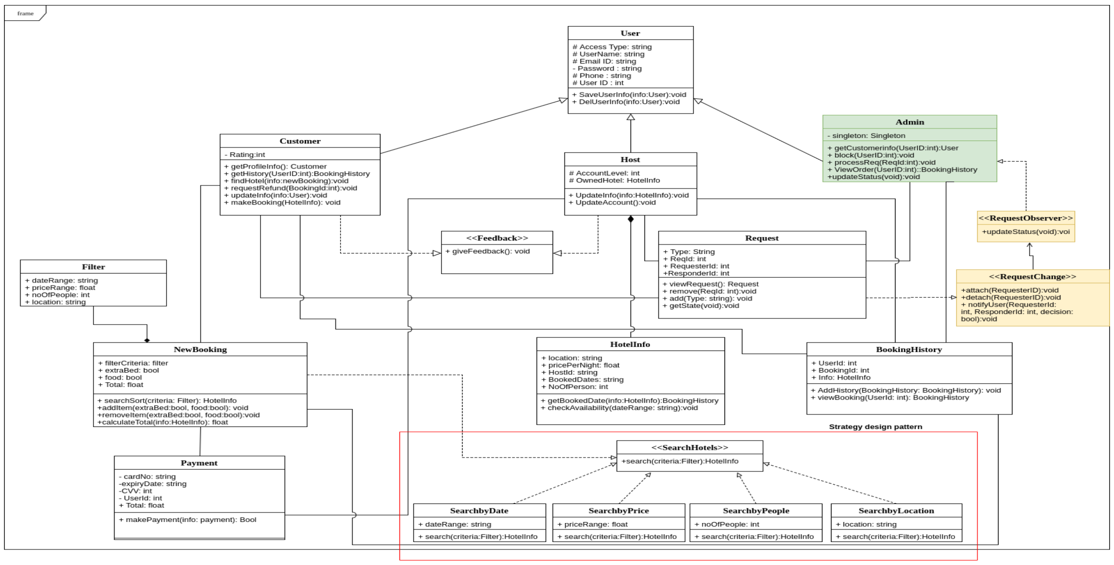
There were few changes in the design. Most important change was the implementation of the design patterns such as observer, singleton and strategy. We gained knowledge about the actual form of Design Patterns through the lectures and applied them during refactoring. Adding these design patterns made the implementation of some use cases simpler and maintainable. We used strategy pattern to divide the search criteria into submodules.
Understanding the design upfront helped us in organizing the software architecture. Developing use case, activity and sequence diagrams solidified our design and helped us in covering all the bases of software flow. We understood how the user requirements will be implemented in software beforehand and this saved us from the constant struggle of changing our implementation again and again when new user requirements came up.

4. Did you make use of any design patterns in the implementation of your final prototype? If so, how? Show the classes from your class diagram that implement each design pattern (each design pattern as a separate image in the .PDF). If not, where could you make use of design patterns in your system? Show a class diagram of how you could implement each design pattern and compare how it would change from your current class diagram.

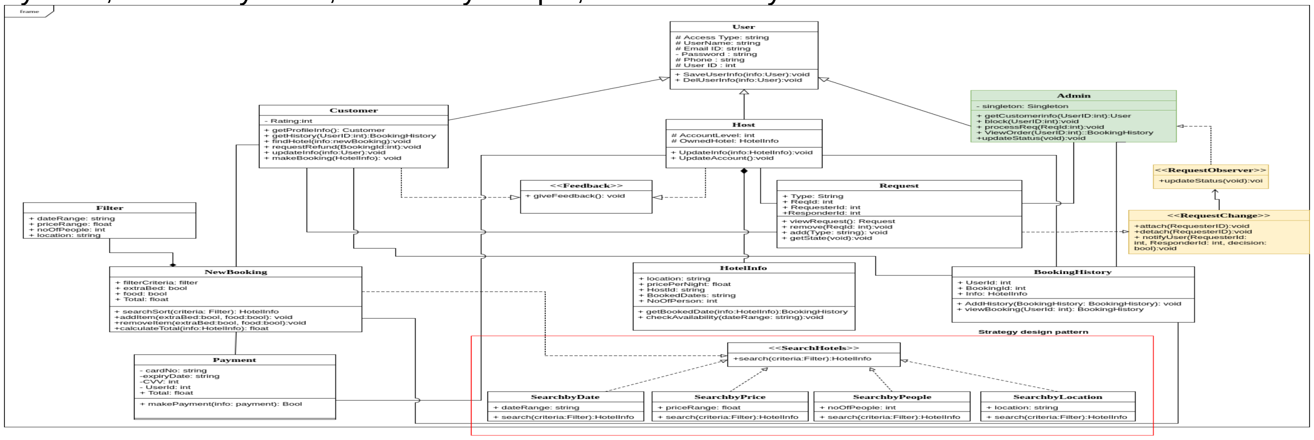
The observer design pattern: It is applied to the request use case which is used by the host to request changes to his/her profile. This is done because before implementing this the host had no way to get to know the status of his request. Through this change the host will be able to get the status of his/her request.


frame
User
# Access Type: string
# UserName: string
# Email ID: string
- Password : string
# Phone : string
# User ID : int
+ SaveUserInfo(info:User):void
+ DelUserInfo(info:User):void
Admin
- singleton: Singleton
+ getCustomerinfo(UserID:int):User
+ block(UserID:int):void
+ processReq(ReqId:int):void
+ ViewOrder(UserID:int)::BookingHistory
+updateStatus(void):void
Customer
- Rating:int
+ getProfileInfo(): Customer
+ getHistory(UserID:int):BookingHistory
+ findHotel(info:newBooking):void
+ requestRefund(BookingId:int):void
+ updateInfo(info:User):void
+ makeBooking(HotelInfo): void
Host
# AccountLevel: int
# OwnedHotel: HotelInfo
+ UpdateInfo(info:HotelInfo):void
+ UpdateAccount():void
<<RequestObserver>>
+updateStatus(void):voi
<<Feedback>>
+ giveFeedback(): void
Request
+ Type: String
+ ReqId: int
+ RequesterId: int
+ResponderId: int
+ viewRequest(): Request
+ remove(ReqId: int):void
+ add(Type: string): void
+ getState(void):void
<<RequestChange>>
+attach(RequesterID):void
+detach(RequesterID):void
+ notifyUser(RequesterId: int, ResponderId: int, decision: bool):void
Filter
+ dateRange: string
+ priceRange: float
+ noOfPeople: int
+ location: string
NewBooking
+ filterCriteria: filter
+ extraBed: bool
+ food: bool
+ Total: float
+ searchSort(criteria: Filter): HotelInfo
+addItem(extraBed:bool, food:bool): void
+removeItem(extraBed:bool, food:bool):void
+calculateTotal(info:HotelInfo): float
HotelInfo
+ location: string
+ pricePerNight: float
+ HostId: string
+ BookedDates: string
+ NoOfPerson: int
+ getBookedDate(info:HotelInfo):BookingHistory
+ checkAvailability(dateRange: string):void
BookingHistory
+ UserId: int
+ BookingId: int
+ Info: HotelInfo
+ AddHistory(BookingHistory: BookingHistory): void
+ viewBooking(UserId: int): BookingHistory
Strategy design pattern
Payment
- cardNo: string
-expiryDate: string
-CVV: int
- UserId: int
+ Total: float
+ makePayment(info: payment): Bool
<<SearchHotels>>
+search(criteria:Filter):HotelInfo
SearchbyDate
+ dateRange: string
+ search(criteria:Filter):HotelInfo
SearchbyPrice
+ priceRange: float
+ search(criteria:Filter):HotelInfo
SearchbyPeople
+ noOfPeople: int
+ search(criteria:Filter):HotelInfo
SearchbyLocation
+ location: string
+ search(criteria:Filter):HotelInfo

Singleton pattern: The admin class is made Singleton as for security reasons there should only be a single administrator account. Before this change it would have been possible to create many admin accounts.


frame
User
# Access Type: string
# UserName: string
# Email ID: string
- Password : string
# Phone : string
# User ID : int
+ SaveUserInfo(info:User):void
+ DelUserInfo(info:User):void
Admin
- singleton: Singleton
+ getCustomerinfo(UserID:int):User
+ block(UserID:int):void
+ processReq(ReqId:int):void
+ ViewOrder(UserID:int)::BookingHistory
+updateStatus(void):void
Customer
- Rating:int
+ getProfileInfo(): Customer
+ getHistory(UserID:int):BookingHistory
+ findHotel(info:newBooking):void
+ requestRefund(BookingId:int):void
+ updateInfo(info:User):void
+ makeBooking(HotelInfo): void
Host
# AccountLevel: int
# OwnedHotel: HotelInfo
+ UpdateInfo(info:HotelInfo):void
+ UpdateAccount():void
<<RequestObserver>>
+updateStatus(void):voi
<<Feedback>>
+ giveFeedback(): void
Request
+ Type: String
+ ReqId: int
+ RequesterId: int
+RespondersId: int
+ viewRequest(): Request
+ remove(ReqId: int):void
+ add(Type: string): void
+ getState(void):void
<<RequestChange>>
+attach(RequesterID):void
+detach(RequesterID):void
+ notifyUser(RequesterId: int, ResponderId: int, decision: bool):void
Filter
+ dateRange: string
+ priceRange: float
+ noOfPeople: int
+ location: string
NewBooking
+ filterCriteria: filter
+ extraBed: bool
+ food: bool
+ Total: float
+ searchSort(criteria: Filter): HotelInfo
+addItem(extraBed:bool, food:bool): void
+removeItem(extraBed:bool, food:bool):void
+calculateTotal(info:HotelInfo): float
HotelInfo
+ location: string
+ pricePerNight: float
+ HostId: string
+ BookedDates: string
+ NoOfPerson: int
+ getBookedDate(info:HotelInfo):BookingHistory
+ checkAvailability(dateRange: string):void
BookingHistory
+ UserId: int
+ BookingId: int
+ Info: HotelInfo
+ AddHistory(BookingHistory: BookingHistory): void
+ viewBooking(UserId: int): BookingHistory
Strategy design pattern
Payment
- cardNo: string
-expiryDate: string
-CVV: int
- UserId: int
+ Total: float
+ makePayment(info: payment): Bool
<<SearchHotels>>
+search(criteria:Filter):HotelInfo
SearchbyDate
+ dateRange: string
+ search(criteria:Filter):HotelInfo
SearchbyPrice
+ priceRange: float
+ search(criteria:Filter):HotelInfo
SearchbyPeople
+ noOfPeople: int
+ search(criteria:Filter):HotelInfo
SearchbyLocation
+ location: string
+ search(criteria:Filter):HotelInfo

Strategy pattern: While searching for a hotel, a user can search based on Date, Price, Number of people allowed or Location. Depending on the type of search, different algorithm would be used. For this purpose, the strategy design pattern has been used. When searching for a hotel, the interface Search is used, which has four realizations depending on the type of search, Search by Date, Search by Price, Search by People, and Search by Location.


frame
User
# Access Type: string
# UserName: string
# Email ID: string
- Password : string
# Phone : string
# User ID : int
+ SaveUserInfo(info:User):void
+ DelUserInfo(info:User):void
Admin
- singleton: Singleton
+ getCustomerinfo(UserID:int):User
+ block(UserID:int):void
+ processReq(ReqId:int):void
+ ViewOrder(UserID:int)::BookingHistory
+updateStatus(void):void
Customer
- Rating:int
+ getProfileInfo(): Customer
+ getHistory(UserID:int):BookingHistory
+ findHotel(info:newBooking):void
+ requestRefund(BookingId:int):void
+ updateInfo(info:User):void
+ makeBooking(HotelInfo): void
Host
# AccountLevel: int
# OwnedHotel: HotelInfo
+ UpdateInfo(info:HotelInfo):void
+ UpdateAccount():void
<<RequestObserver>>
+updateStatus(void):voi
<<Feedback>>
+ giveFeedback(): void
Request
+ Type: String
+ ReqId: int
+ RequesterId: int
+RespondersId: int
+ viewRequest(): Request
+ remove(ReqId: int):void
+ add(Type: string): void
+ getState(void):void
<<RequestChange>>
+attach(RequesterID):void
+detach(RequesterID):void
+ notifyUser(RequesterId: int, ResponderId: int, decision: bool):void
Filter
+ dateRange: string
+ priceRange: float
+ noOfPeople: int
+ location: string
NewBooking
+ filterCriteria: filter
+ extraBed: bool
+ food: bool
+ Total: float
+ searchSort(criteria: Filter): HotelInfo
+addItem(extraBed:bool, food:bool): void
+removeItem(extraBed:bool, food:bool):void
+calculateTotal(info:HotelInfo): float
HotelInfo
+ location: string
+ pricePerNight: float
+ HostId: string
+ BookedDates: string
+ NoOfPerson: int
+ getBookedDate(info:HotelInfo):BookingHistory
+ checkAvailability(dateRange: string):void
BookingHistory
+ UserId: int
+ BookingId: int
+ Info: HotelInfo
+ AddHistory(BookingHistory: BookingHistory): void
+ viewBooking(UserId: int): BookingHistory
Strategy design pattern
Payment
- cardNo: string
-expiryDate: string
-CVV: int
- UserId: int
+ Total: float
+ makePayment(info: payment): Bool
<<SearchHotels>>
+search(criteria:Filter):HotelInfo
SearchbyDate
+ dateRange: string
+ search(criteria:Filter):HotelInfo
SearchbyPrice
+ priceRange: float
+ search(criteria:Filter):HotelInfo
SearchbyPeople
+ noOfPeople: int
+ search(criteria:Filter):HotelInfo
SearchbyLocation
+ location: string
+ search(criteria:Filter):HotelInfo

5. What have you learned about the process of analysis and design now that you have stepped through the process to create, design and implement a system?

Before the course, engineers like us tend to concentrate more on technical aspects and directly jump on developing the software without proper planning. In this class, we have learned that it is extremely important to spend a good amount of time on analyzing and designing the system before implementing it. Software is naturally intangible, it is not easy to make the code portable and readable across the team. Sketching out the diagrams that enabled us to pen our ideas and make sure everyone understands the role and task to be completed. During the development, we observed that usually the first design is not perfect and has flaws in it. Refactoring of the design helped us to remove our flaws and make our design simpler. It shows that many iterations are necessary to ensure a good design. We found the class diagram to be most useful as it clearly iterates software modules needed to be implemented along with attributes and methods. So, most of the implementation part was creating those same exact classes. The design patterns helped us making our code more readable and easy to implement. This class has changed our approach in design a software and we would obviously implement these learning in our future designs.