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

User Requirements	
ID	Requirement
UR-01	As a student, I want to be able to create an account using my school email address.
UR-02	As a student, I want to be able to log in to the system.
UR-03	As a student, I want to be able to view my bookings.
UR-04	As a student, I want to be able to receive email notifications on bookings made.
UR-05	As a student, I want to be able to make a booking.
UR-06	As a student, I want to be able to edit a booking.
UR-07	As a student, I want to be able to delete a booking.
UR-08	As a student, I want to be able to view locations' schedule.
UR-09	As an admin, I want to be able to add new locations that can be booked.
UR-10	As an admin, I want to be able to delete locations.
UR-11	As an admin, I want to be able to delete a student's account.
UR-12	As an admin, I want to be able to override a student's booking.
UR-13	As a student, I want to be able to log out of the system.
UR-14	As an admin, I want to be able to log out of the system.
UR-15	As an admin, I want to be able to log in to the system.
UR-16	As a student, I want to be able to receive email notification if my booking was overridden.
UR-17	As an admin, I want to able to view locations' schedule.

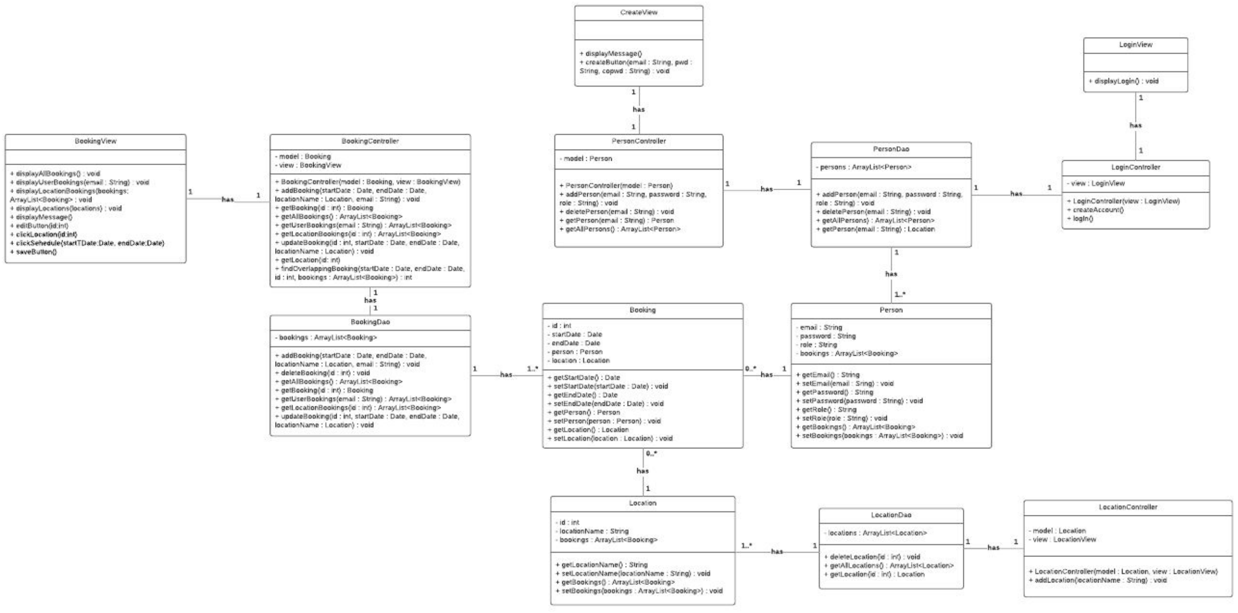
UR-13 to UR-17 were newly added user requirements and were completed within the span of project time.

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

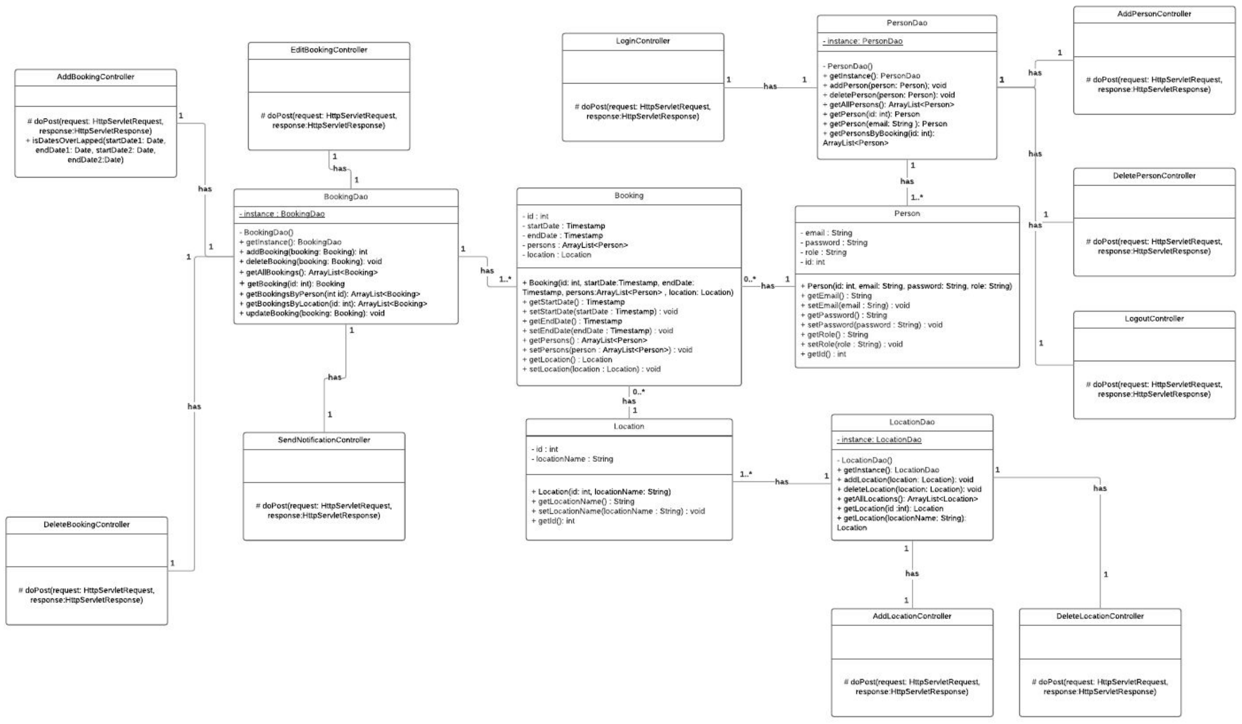
We managed to implement all features listed in Part 2. The above table in Question 1 illustrates the functions we had written in Part 2.

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.

Part 2 Class Diagram


CreateView
+ displayMessage()
+ createButton(email : String, pwd : String, copwd : String) : void
LoginView
+ displayLogin() : void
BookingView
+ displayAllBookings() : void
+ displayUserBookings(email : String) : void
+ displayLocationBookings(bookings: ArrayList<Booking> : void
+ displayLocations(locations) : void
+ displayMessage()
+ editButton(id:int)
+ clickLocation(id:int)
+ clickSehedule(startTDate:Date, endDate:Date)
+ saveButton()
BookingController
- model : Booking
- view : BookingView
+ BookingController(model : Booking, view : BookingView)
+ addBooking(startDate : Date, endDate : Date, locationName : Location, email : String) : void
+ getBooking(id : int) : Booking
+ getAllBookings() : ArrayList<Booking>
+ getUserBookings(email : String) : ArrayList<Booking>
+ getLocationBookings(id : int) : ArrayList<Booking>
+ updateBooking(id : int, startDate : Date, endDate : Date, locationName : Location) : void
+ getLocation(id: int)
+ findOverlappingBooking(startDate : Date, endDate : Date, id : int, bookings : ArrayList<Booking>) : int
PersonController
- model : Person
+ PersonController(model : Person)
+ addPerson(email : String, password : String, role : String) : void
+ deletePerson(email : String) : void
+ getPerson(email : String) : Person
+ getAllPersons() : ArrayList<Person>
PersonDao
- persons : ArrayList<Person>
+ addPerson(email : String, password : String, role : String) : void
+ deletePerson(email : String) : void
+ getAllPersons() : ArrayList<Person>
+ getPerson(email : String) : Location
LoginController
- view : LoginView
+ LoginController(view : LoginView)
+ createAccount()
+ login()
BookingDao
- bookings : ArrayList<Booking>
+ addBooking(startDate : Date, endDate : Date, locationName : Location, email : String) : void
+ deleteBooking(id : int) : void
+ getAllBookings() : ArrayList<Booking>
+ getBooking(id : int) : Booking
+ getUserBookings(email : String) : ArrayList<Booking>
+ getLocationBookings(id : int) : ArrayList<Booking>
+ updateBooking(id : int, startDate : Date, endDate : Date, locationName : Location) : void
Booking
- id : int
- startDate : Date
- endDate : Date
- person : Person
- location : Location
+ getStartDate() : Date
+ setStartDate(startDate : Date) : void
+ getEndDate() : Date
+ setEndDate(endDate : Date) : void
+ getPerson() : Person
+ setPerson(person : Person) : void
+ getLocation() : Location
+ setLocation(location : Location) : void
Person
- email : String
- password : String
- role : String
- bookings : ArrayList<Booking>
+ getEmail() : String
+ setEmail(email : Sring) : void
+ getPassword() : String
+ setPassword(password : String) : void
+ getRole() : String
+ setRole(role : String) : void
+ getBookings() : ArrayList<Booking>
+ setBookings(bookings : ArrayList<Booking>) : void
Location
- id : int
- locationName : String
- bookings : ArrayList<Booking>
+ getLocationName() : String
+ setLocationName(locationName : String) : void
+ getBookings() : ArrayList<Booking>
+ setBookings(bookings : ArrayList<Booking>) : void
LocationDao
- locations : ArrayList<Location>
+ deleteLocation(id : int) : void
+ getAllLocations() : ArrayList<Location>
+ getLocation(id : int) : Location
LocationController
- model : Location
- view : LocationView
+ LocationController(model : Location, view : LocationView)
+ addLocation(locationName : String) : void
has
1
1..*
0..*

Final Class Diagram

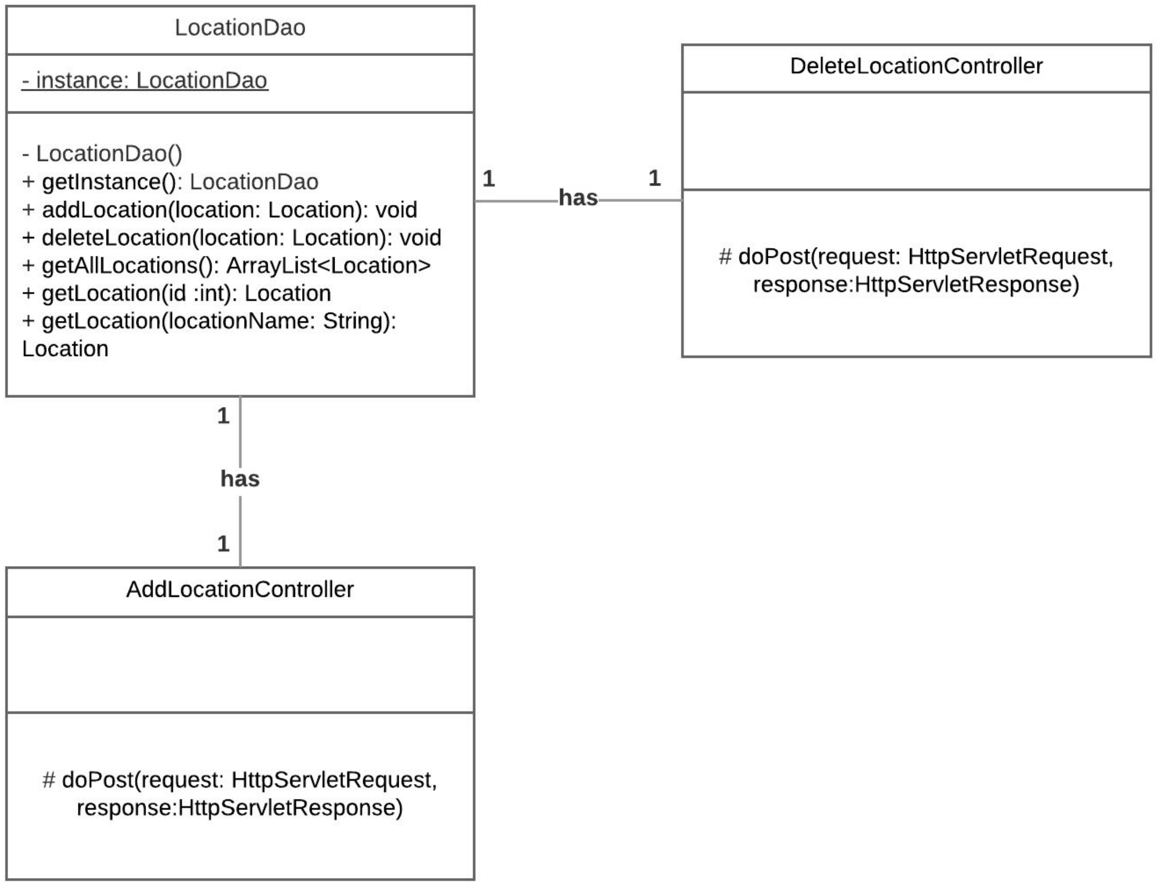

AddBookingController
# doPost(request: HttpServletRequest, response:HttpServletResponse)
+ isDatesOverLapped(startDate1: Date, endDate1: Date, startDate2: Date, endDate2:Date)
EditBookingController
# doPost(request: HttpServletRequest, response:HttpServletResponse)
LoginController
# doPost(request: HttpServletRequest, response:HttpServletResponse)
PersonDao
- instance: PersonDao
- PersonDao()
+ getInstance(): PersonDao
+ addPerson(person: Person); void
+ deletePerson(person: Person): void
+ getAllPersons(): ArrayList<Person>
+ getPerson(id: int): Person
+ getPerson(email: String): Person
+ getPersonsByBooking(id: int): ArrayList<Person>
AddPersonController
# doPost(request: HttpServletRequest, response:HttpServletResponse)
BookingDao
- instance : BookingDao
- BookingDao()
+ getInstance(): BookingDao
+ addBooking(booking: Booking): int
+ deleteBooking(booking: Booking): void
+ getAllBookings(): ArrayList<Booking>
+ getBooking(id: int): Booking
+ getBookingsByPerson(int id): ArrayList<Booking>
+ getBookingsByLocation(id: int): ArrayList<Booking>
+ updateBooking(booking: Booking): void
Booking
- id : int
- startDate : Timestamp
- endDate : Timestamp
- persons : ArrayList<Person>
- location : Location
+ Booking(id: int, startDate:Timestamp, endDate: Timestamp, persons:ArrayList<Person> , location: Location)
+ getStartDate() : Timestamp
+ setStartDate(startDate : Timestamp) : void
+ getEndDate() : Timestamp
+ setEndDate(endDate : Timestamp) : void
+ getPersons() : ArrayList<Person>
+ setPersons(person : ArrayList<Person>) : void
+ getLocation() : Location
+ setLocation(location : Location) : void
Person
- email : String
- password : String
- role : String
- id: int
+ Person(id: int, email: String, password: String, role: String)
+ getEmail() : String
+ setEmail(email : Sring) : void
+ getPassword() : String
+ setPassword(password : String) : void
+ getRole() : String
+ setRole(role : String) : void
+ getId() : int
DeletePersonController
# doPost(request: HttpServletRequest, response:HttpServletResponse)
LogoutController
# doPost(request: HttpServletRequest, response:HttpServletResponse)
SendNotificationController
# doPost(request: HttpServletRequest, response:HttpServletResponse)
Location
- id : int
- locationName : String
+ Location(id: int, locationName: String)
+ getLocationName() : String
+ setLocationName(locationName : String) : void
+ getId(): int
LocationDao
- instance: LocationDao
- LocationDao()
+ getInstance(): LocationDao
+ addLocation(location: Location): void
+ deleteLocation(location: Location): void
+ getAllLocations(): ArrayList<Location>
+ getLocation(id :int): Location
+ getLocation(locationName: String): Location
DeleteBookingController
# doPost(request: HttpServletRequest, response:HttpServletResponse)
AddLocationController
# doPost(request: HttpServletRequest, response:HttpServletResponse)
DeleteLocationController
# doPost(request: HttpServletRequest, response:HttpServletResponse)
has
1
1..*
0..*

Instead of having all respective methods in one controller, we split up the methods and had a separate controller for each of them. For example, BookingController was split into AddBookingController, EditBookingController and DeleteBookingController. In addition, we removed View Classes in the final diagram as Views are not needed in class diagrams. In our implementation, views are represented by JSP pages.

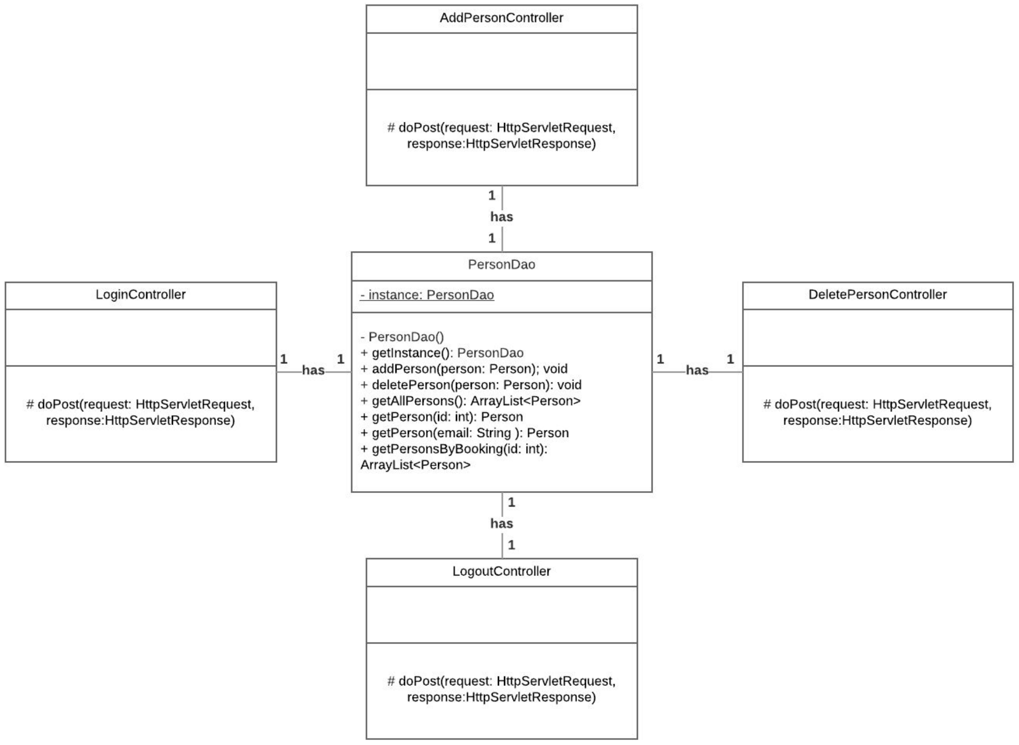
4. Did you make use of any design patterns in the implementation of your final prototype? If so, how? Show the classes from your class diagram that implement each design pattern (each design pattern as a separate image in the .PDF). If not, where could you make use of design patterns in your system?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

We implemented the Singleton design pattern. The Singleton design pattern was implemented on the Data Access Objects (DAO). DAO is an interface that communicates with the underlying database. We implemented Singleton on DAO because we wanted only one instance of each of the available DAO, namely BookingDao, LocationDao and PersonDao. Implementing the Singleton design pattern could save us some memory space and prevent currency issues.

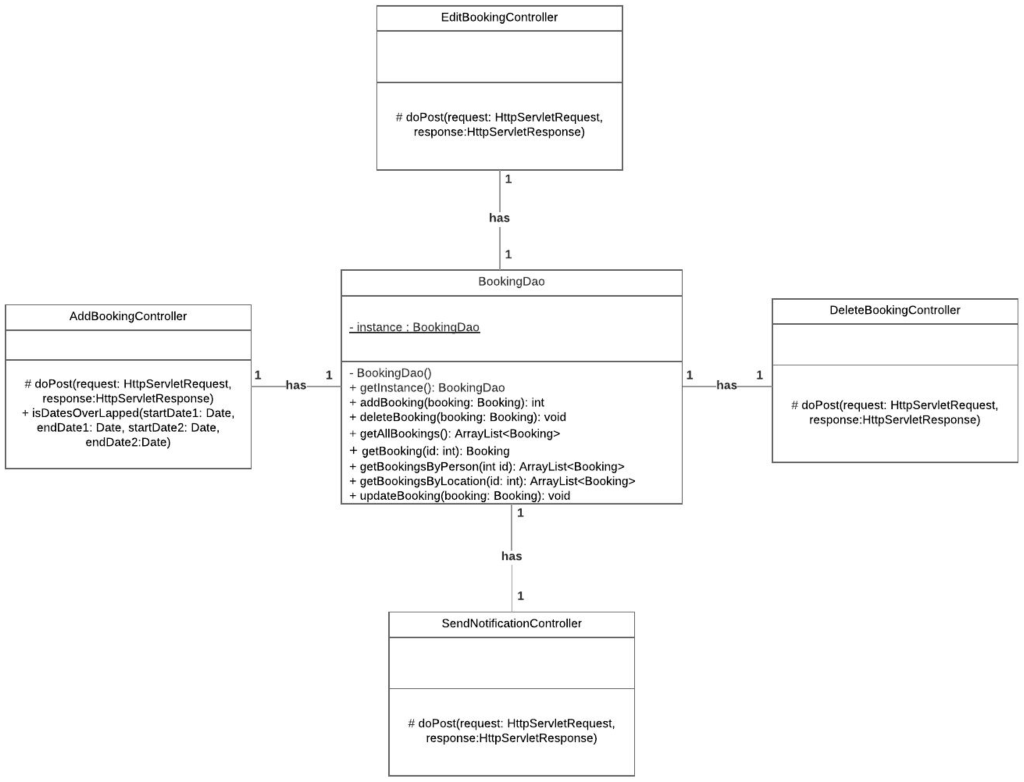
Singleton design pattern: LocationDao


LocationDao
- instance: LocationDao
- LocationDao()
+ getInstance(): LocationDao
+ addLocation(location: Location): void
+ deleteLocation(location: Location): void
+ getAllLocations(): ArrayList<Location>
+ getLocation(id :int): Location
+ getLocation(locationName: String): Location
DeleteLocationController
# doPost(request: HttpServletRequest, response:HttpServletResponse)
1
has
1
1
has
1
AddLocationController
# doPost(request: HttpServletRequest, response:HttpServletResponse)

Singleton design pattern: PersonDao

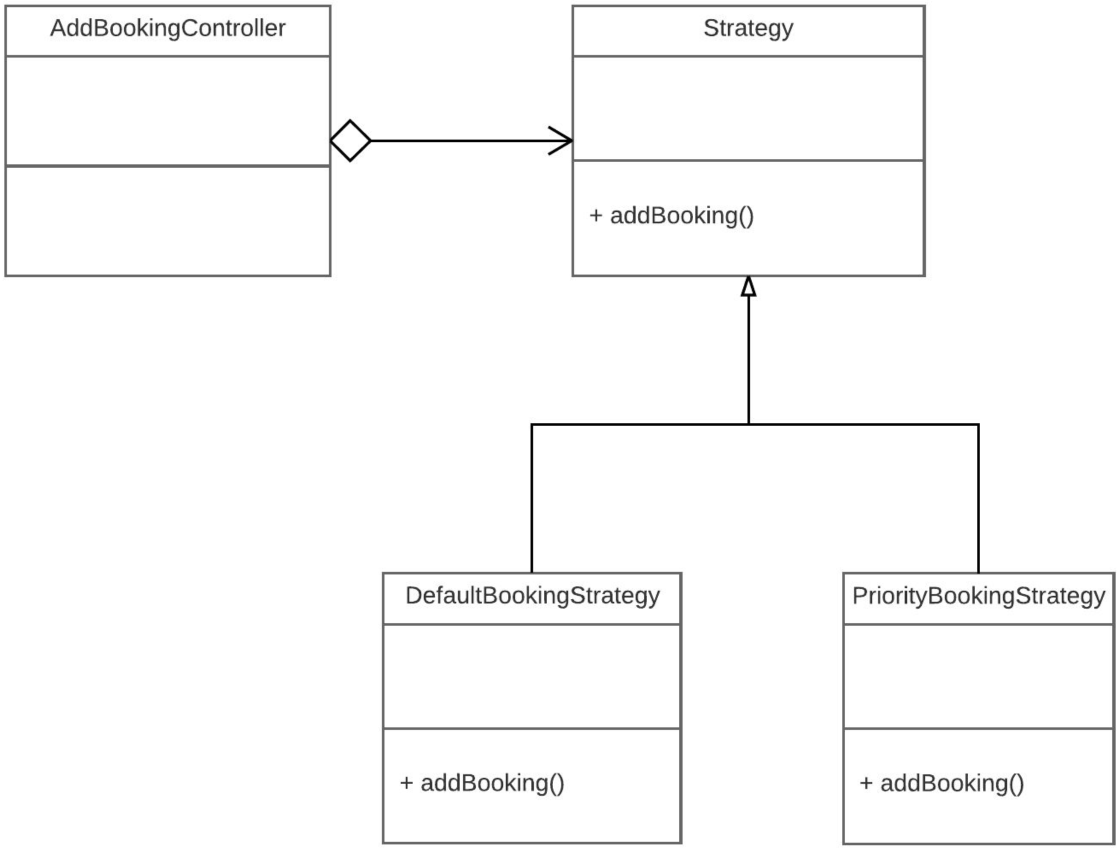

AddPersonController
# doPost(request: HttpServletRequest, response:HttpServletResponse)
1
has
1
PersonDao
- instance: PersonDao
- PersonDao()
+ getInstance(): PersonDao
+ addPerson(person: Person); void
+ deletePerson(person: Person): void
+ getAllPersons(): ArrayList<Person>
+ getPerson(id: int): Person
+ getPerson(email: String): Person
+ getPersonsByBooking(id: int): ArrayList<Person>
LoginController
# doPost(request: HttpServletRequest, response:HttpServletResponse)
1
has
1
DeletePersonController
# doPost(request: HttpServletRequest, response:HttpServletResponse)
1
has
1
1
has
1
LogoutController
# doPost(request: HttpServletRequest, response:HttpServletResponse)

Singleton design pattern: BookingDao


EditBookingController
# doPost(request: HttpServletRequest, response:HttpServletResponse)
1
has
1
BookingDao
- instance : BookingDao
- BookingDao()
+ getInstance(): BookingDao
+ addBooking(booking: Booking): int
+ deleteBooking(booking: Booking): void
+ getAllBookings(): ArrayList<Booking>
+ getBooking(id: int): Booking
+ getBookingsByPerson(int id): ArrayList<Booking>
+ getBookingsByLocation(id: int): ArrayList<Booking>
+ updateBooking(booking: Booking): void
AddBookingController
# doPost(request: HttpServletRequest, response:HttpServletResponse)
+ isDatesOverLapped(startDate1: Date, endDate1: Date, startDate2: Date, endDate2:Date)
1
has
1
DeleteBookingController
# doPost(request: HttpServletRequest, response:HttpServletResponse)
1
has
1
1
has
1
SendNotificationController
# doPost(request: HttpServletRequest, response:HttpServletResponse)

We could have implemented the Strategy design pattern on AddBookingController. While admin and student are able to make a booking, admin has "special" rights such that s/he is able to override a student's booking. The strategy design pattern would definitely be a smarter and more code-efficient way to implement the booking method. From the class diagram below, you would see that AddBookingController is the aggregate class while Strategy is the component class. DefaultBookingStrategy would implement student add booking method while PriorityBookingStrategy would implement admin booking method.

Strategy design pattern: AddBookingController



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?

We learned that it is important to be familiar and aware of the architecture design in order to implement a system more smoothly. While the application that we have implemented, Facility Booking Management System, is not an application as large as industry-standards and we did not experience the adverse effects of not planning out the process of analysis and design properly, we had a few hiccups in the process which resulted in productivity and efficiency level to go down. In the future, before implementing the application, we should be well-versed or at least familiar with the architecture design before jumping into coding.