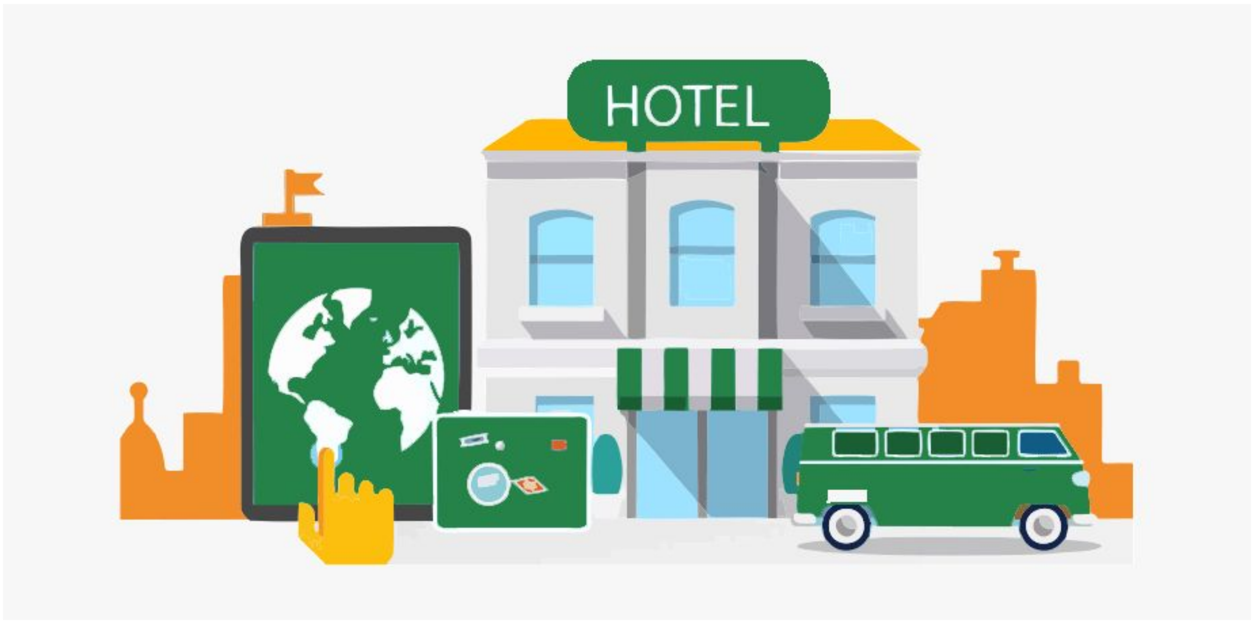
CS4125 - Systems Analysis and Design

EZ Hotel Management System



Team Cyan 1

John Long - 12132306
Naichuan Zhang - 18111521

Table of Contents

Business Scenario Overview

The EzHotel application is a website where the users can have the same functionality as having to ring a hotel and make a room or event booking. This reduces the need for staff in the hotel and brings the cost of running down as staff costs remain a huge part of any business's bottom line. Many hotels now have invested heavily in their online platforms as they see the value that they can provide for users and staff alike. The system can be run by a single person maintaining and adding features to the application.

The hotel system involves registering as a new user or logging in as an existing user. A user can make a booking online and either confirm or cancel once that booking is processed. The user can check in and check out once their stay is complete.

To facilitate the user experience we attempted to make a simple and intuitive GUI so that users can avail of the features of the platform without any confusion or hassle.

Users can receive discounts on bookings depending on loyalty levels. If a user is a returning guest then they can avail of the discounts provided to loyal guests.

Creating a system like this can ensure that the hotel can maximise profits by enhancing and making the repetitive tasks associated with a stay easier to complete and keep up with modern day trends where online applications are necessary.

Project Roles

	Role	Description	Team Member
1.	Project Manager	Set up meetings, agree on project plan and tracks progress	All
2.	Documentation Manager	Source relevant supporting documentation from each team member and composing it in report	All
3.	Business Analyst/Requirements Engineer	Responsible for Requirements	All
4.	Architect	Defines system architecture	John
5.	Systems Analyst	Creates conceptual class model	Naichuan
6.	Designer	Responsible for recovering design time blueprints from implementation	All
7.	Technical Lead	Leads the implementation effort	All
8.	Programmers	Actual implementation	All
9.	Tester	Coding of automated test cases	John
10.	Dev Ops	Ensure team is competent with development infrastructure	All

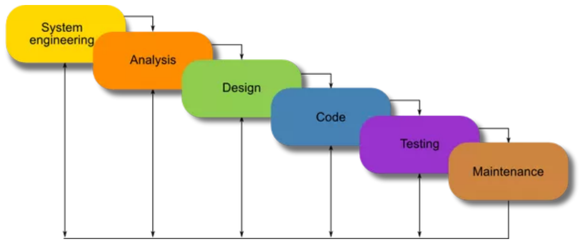
Project Plan

Deliverable	Description	Responsibility	Week #
Business Scenario	Narrative description of business scenario	John	4
Software Lifecycle	Discussion on software lifecycle adopted	Naichuan	5
Establish Roles	Specify roles of group members to complete project	Group	4
Requirements	Functional requirements Use case diagram Use case description Non-functional requirements Tactics to support quality attributes GUI prototypes	Naichuan John Naichuan John John John	5
System Architecture	Package diagrams Architectural decisions	John Naichuan	6
Analysis Sketches	Candidate objects Analysis class diagram Communication diagram	Naichuan Naichuan Naichuan	7
Code	Code implementation	ALL	7/8
Design	Architectural diagram Design-time class diagram State chart Sequence diagrams	John Naichuan John	11
Added Value	Spring Boot Github Google Docs Database - JPA/H2 Mockito Maven Builder DP	John, Naichuan	11
Critique	Compare analysis sketches vs	John	11

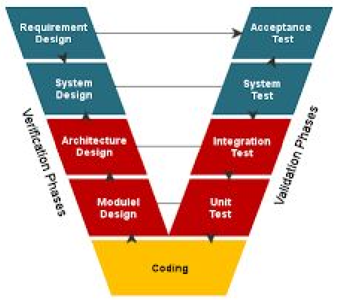
	blueprints		
References	List sources	Group	12

Software Lifecycle Model

The first lifecycle model we discussed was the **Waterfall Model**, which can be said is the earliest SDLC approach that was used in software development. This model divides the software lifecycle into six stages with a fixed order to transition the stages from the previous to the next, and finally get the developed software product. In this model, each stage proceeds in a linear approach and each stage must be completed before the next stage can begin. Although it is easy to be understood and adopted, some limitations of the model discourage us from using it. One problem is its inflexibility to adapt to changing requirements, it greatly increases the risk of development since we cannot be sure we will not change anything. In addition, errors that occur early in development may not be detected until the testing stage and we know it will be a huge risk for our group.



The second model we investigated was the **V-Model**. Different from the Waterfall Model, it specifies the different levels in the testing process throughout the whole development lifecycle. It greatly shortens the development lifecycle and improves the efficiency of development by developing and testing simultaneously. However, it is still a risky approach for our group to take, since it is unable to adapt to any necessary changes during the development, much like the Waterfall Model. So, we quickly crossed this model out as well.



Agile approach is what we finally decided to adopt. The biggest advantage of Agile is its flexibility and adaptability compared to the previous models analyzed, the benefit for us is that we are free to explore instead of sticking to some rigid plans. And it can be said that it is one of the most popular approaches when the development team is small-scaled, this is the main reason why we preferred to use it. But we know that agile focuses on communication and ignores the importance of documentation, which is something we should pay special attention to in the future.

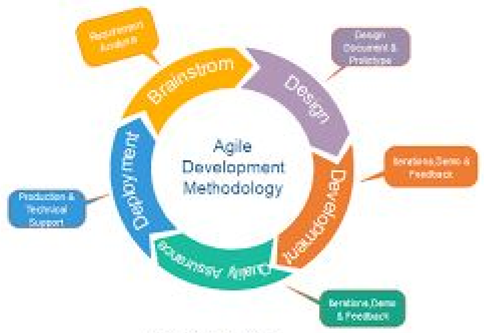

Requirement Analysis
Brainstrom
Design Document & Prototype
Design
Iterations,Demo & Feedback
Development
Iterations,Demo & Feedback
Quality Assurance
Production & Technical Support
Deployment
Agile Development Methodology

Fig. Agile Model

Requirements

Functional Requirements

A guest can:

1. register (with username, email address, password, phone number etc.)
2. login and logout
3. change password
4. view as a guest without reservation or payment
5. make a room reservation
    a. guest can reserve different number of rooms (max 8)
    b. guests can select date to check in and check out
    c. guest can choose different room types (double room, single room, suite, etc.)
6. cancel a reservation before checking in (with an explanation why)
7. search available rooms during a time period
8. request an event booking
    a. guest can choose event type
    b. guest can select date for event
9. cancel an event booking

10. make a payment
11. view booking history
12. check in with a valid reservation
13. check out after checking in
14. comment/rate after checking out
15. contact a member of the hotel staff.

A manager can:

1. login and logout
2. change password
3. manage guests (delete, ban etc.)
4. change guests rate and/or comment //Bad
5. process refund (validate)
6. add new rooms (price, availability, room type, etc.)
7. update rooms (price, availability, room type, etc.)
8. remove/make unavailable rooms
9. confirm event
10. cancel event

The system should:

1. record guest details
2. record reservation
3. record payment
4. provide 2.5% discount per guest added during booking, maximum 8 guests
5. display all available rooms during a period, ranked by rate, default rate is 5 (rate: 0-5)
6. require an explanation when a guest cancels a reservation
7. automatically cancel reservations if a guest doesn't make a payment within 24h after reservation
8. add loyalty points (+10) every time when a guest completes a whole room booking process
    a. when loyalty points <= 50, copper (level 1), no discount
    b. when loyalty points >50 and <= 100, silver (level 2), 10% discount

    c. when loyalty points >100 <= 150, gold (level 3), 20% discount
    d. when loyalty points >150, platinum (level 4), 25% discount

9. display guests rate and/or comment

Non-Functional Requirements

Reliability: System must be available 24/7

Portability: Java can work on any system that can support JVM

Security: A user must be logged in to use the system, sensitive data being sent to and from database must be encrypted and credit cards must be validated.

Performance: The system should connect to the database in a timely fashion.

Scalability: System should be able to handle being scaled, programming to interfaces not implementation.

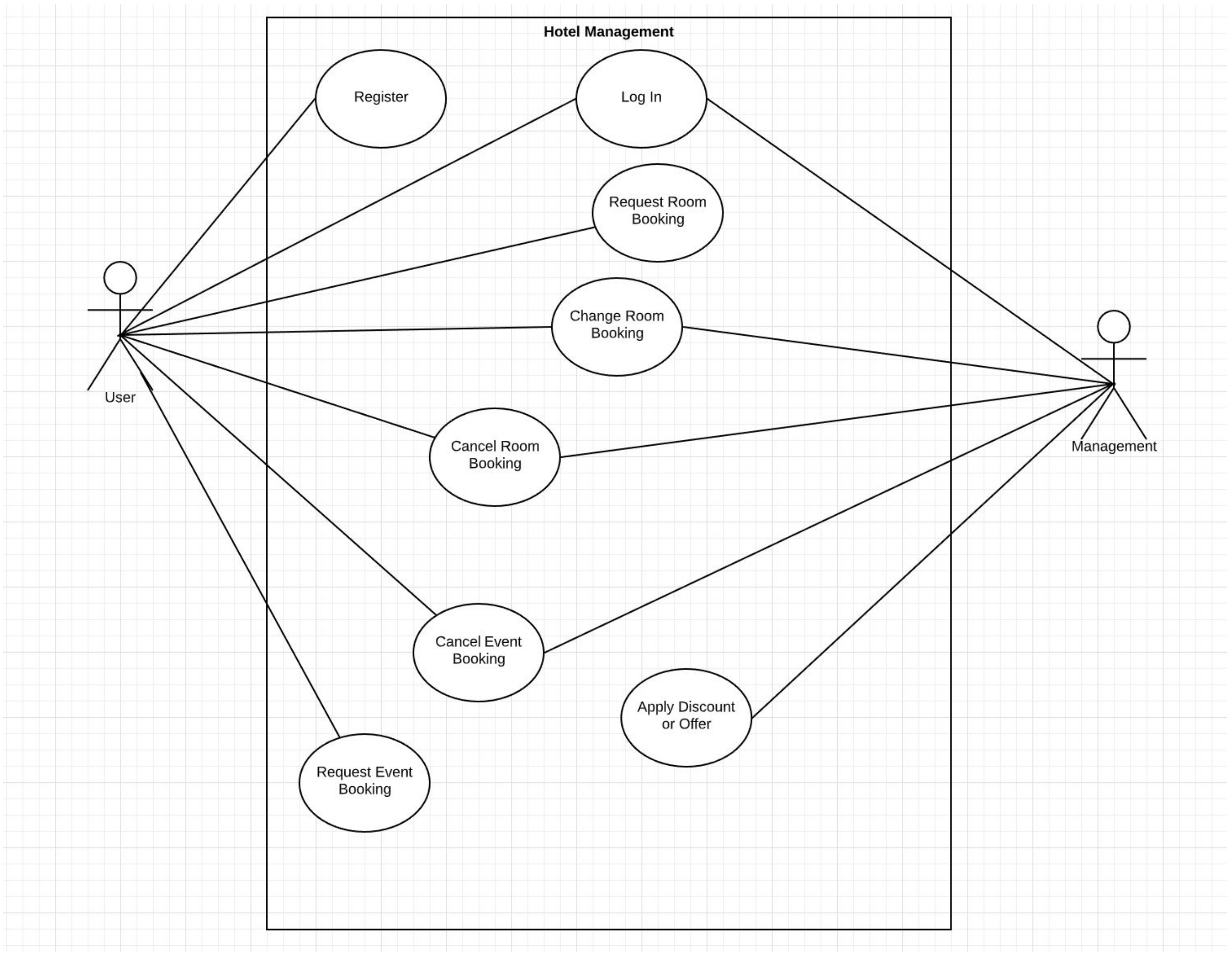
Usability: System/menus should be easy to use and interact with.

Maintainability: The software should be easily maintained; readability of code and connected documentation is key to this.

Quality Tactics

We would decide to constrain ourselves to use the model view controller pattern from the outset of the project in order to stick to good coding practices and standards.

Use Case Diagram



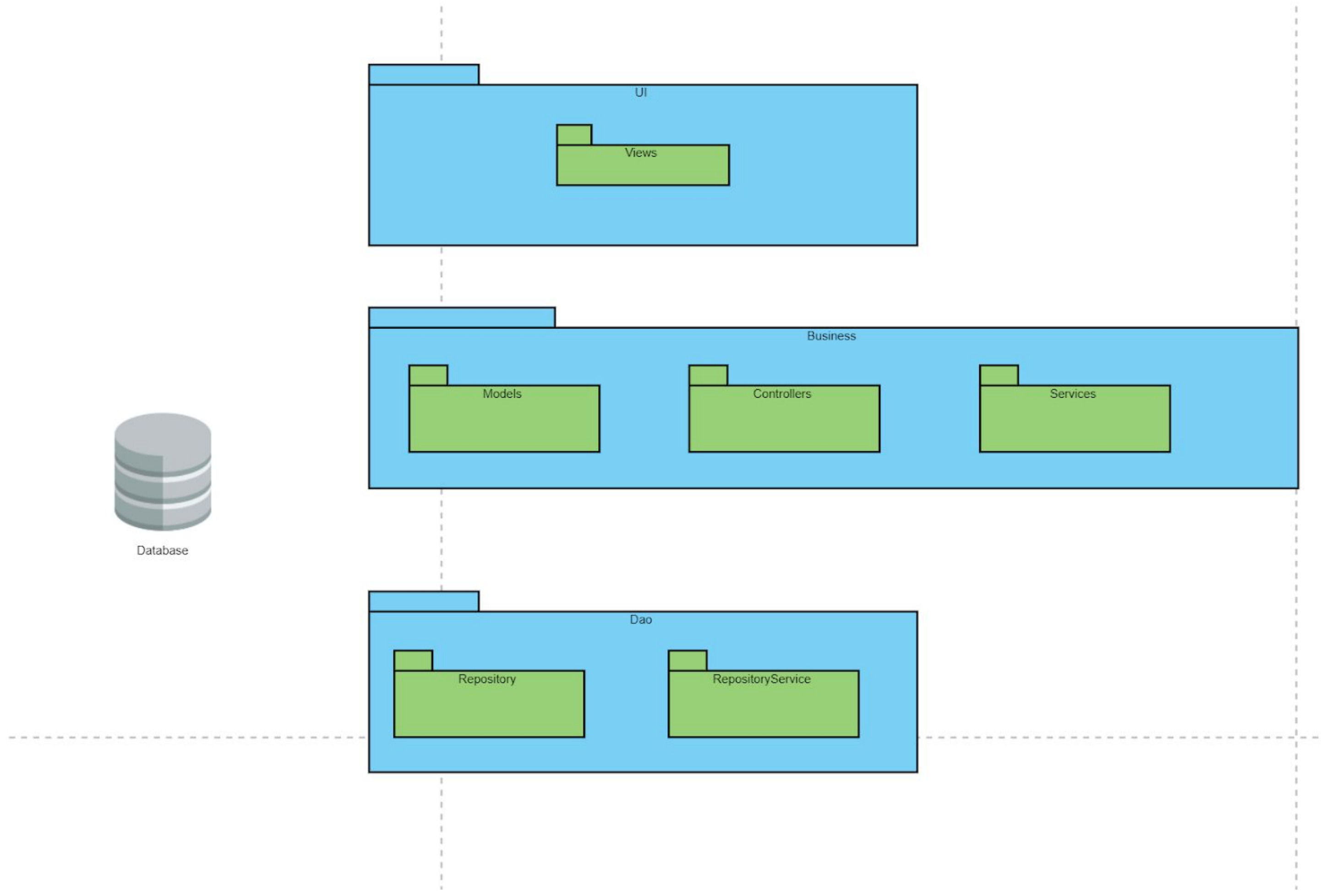
Use Case Descriptions

Use Case	Request Room Booking
Goal in Context	User selects a type room the wish to stay in and then all the Booking details are added to the database.
Scope & Level	Company
Preconditions	1. User is registered 2. User is logged in
Success End Conditions	A room is booked and added to database
Failed End Conditions	A room booking is not added to the database and error message detailing why is displayed
Primary, Secondary, Actors	User, system
Trigger	A booking request comes in
Description	1. User selects request room booking from the main menu. 2. System opens the room booking screen. 3. User enters what hotel, rome type and dates they will be staying and then selects "continue". 4. System verifies that there are rooms available at the selected hotel within the selected dates and calculates price. 5. User clicks "pay" 6. System writes to the database, marking the room as unavailable and adding the booking
Extensions	4a. If there are no rooms left for date the user is informed 5a. If the user declines to pay they are sent to the main menu
Variations	
Priority	Top
Due Date	Release 1.0

Use Case	Cancel Room Booking
Goal in Context	User selects a booking they made in the past that they wish to cancel
Scope & Level	Company
Preconditions	1. User is registered 2. User is logged in 3. User has made a booking
Success End Conditions	A room that was booked is removed from the database
Failed End Conditions	A room booking is not removed from the database and error message detailing why is displayed
Primary, Secondary, Actors	User, system
Trigger	A request to cancel a booking comes in
Description	1. User selects cancel room booking from the main menu. 2. System opens the cancel room booking screen. 3. User selects a booking the wish to cancel and then selects “continue”. 4. System removes the booking from the database and refunds the money spent
Extensions	
Variations	
Priority	Top
Due Date	Release 1.0

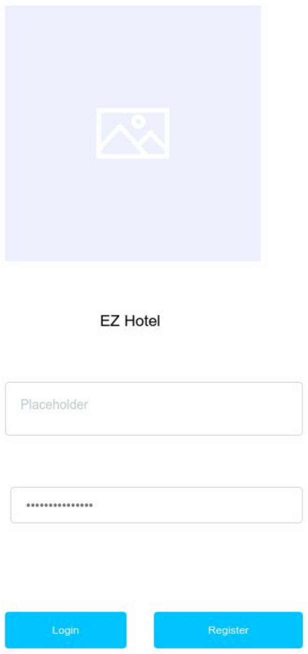
Use Case	Change Room Booking
Goal in Context	User selects a booking they made in the past that they wish to change
Scope & Level	Company
Preconditions	1. User is registered 2. User is logged in 3. User has made a booking
Success End Conditions	A room booking is modified in the database
Failed End Conditions	A room booking is not modified in the database and error message detailing why is displayed
Primary, Secondary, Actors	User, system
Trigger	A request to modify a booking comes in
Description	1. User selects modify room booking from the main menu. 2. System opens the modify room booking screen. 3. User selects a booking the wish to modify and then modifies hotel, room type or date and then selects "continue". 4. System verifies that there are rooms available at the selected hotel within the selected dates and calculates price. 5. User clicks "pay" 6. System modifies the booking in the database
Extensions	4a. If the the new room costs less them the room before the user is refunded the difference. 5a. If the user declines to pay they are sent to the main menu
Variations	
Priority	Top
Due Date	Release 1.0

Package Diagram

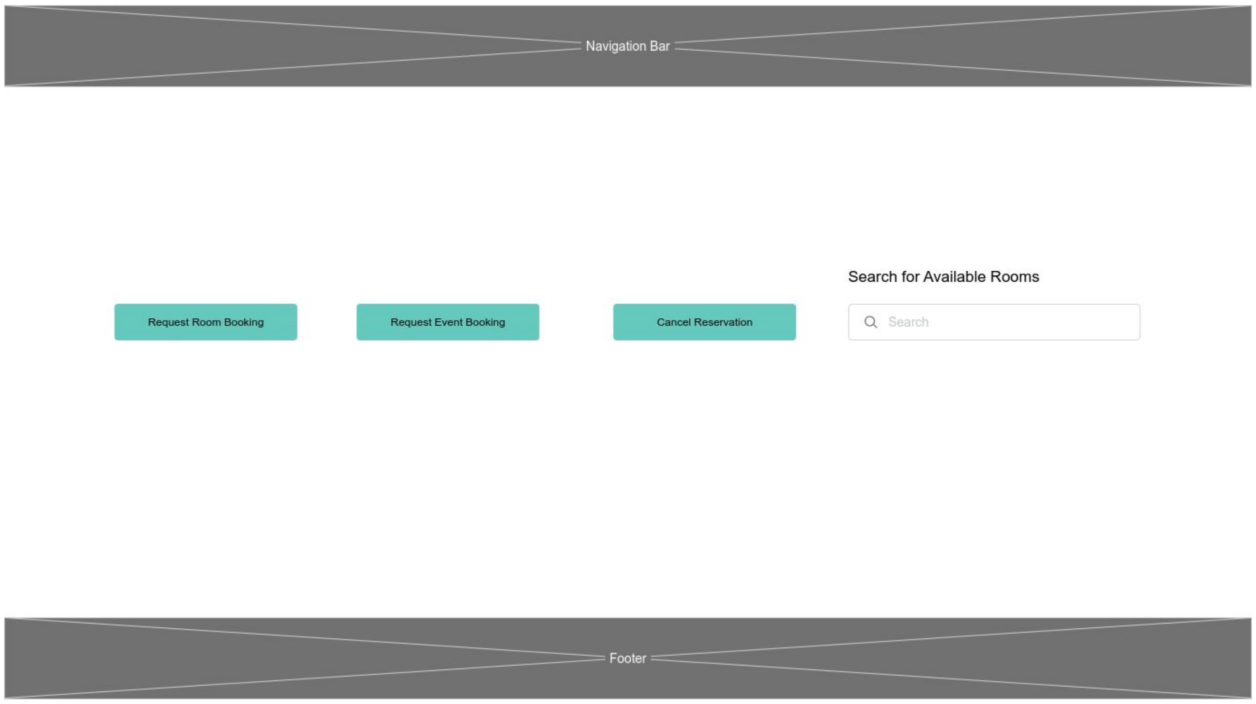


GUI Prototype

Login Screen



Guest Screen View



Analysis Sketches

Candidate Classes Identification

Data Driven Design is applied to identify all candidate classes. Here is an initial list of all possible candidate classes we found from functional requirements by using noun identification technique.

List of Possible Candidate Classes

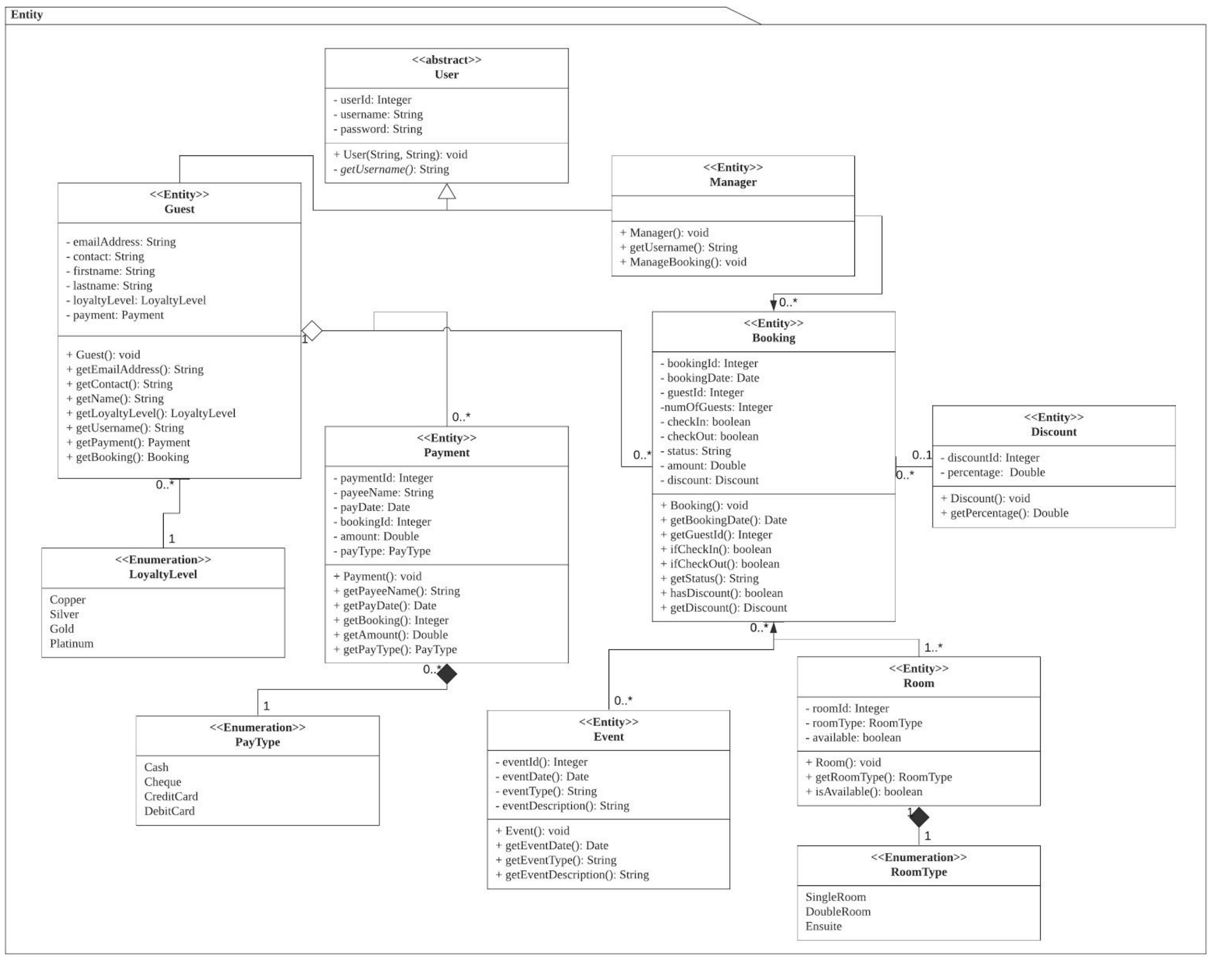
guest	username	password	phone number	reservation
payment	room	date	room type	event
event type	booking	booking history	manager	rate
comment	day	member of hotel staff	guest details	discount
loyalty points	level	loyalty level	system	

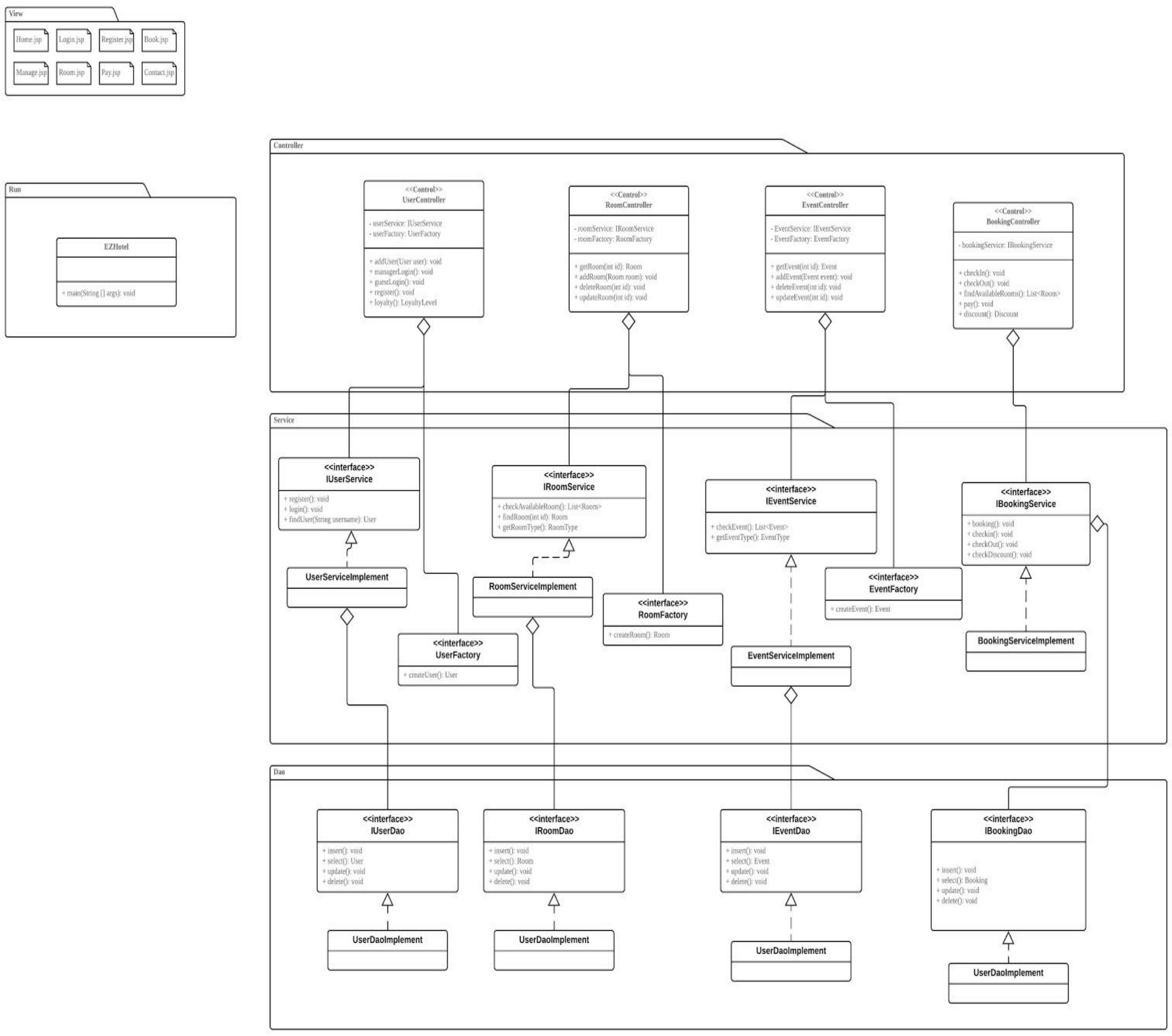
After careful consideration and filtering, we intend to eliminate/modify the following possible candidate classes for the following reasons.

Classes	Explanations
username	An attribute of guest
password	An attribute of guest
phone number	An attribute of guest

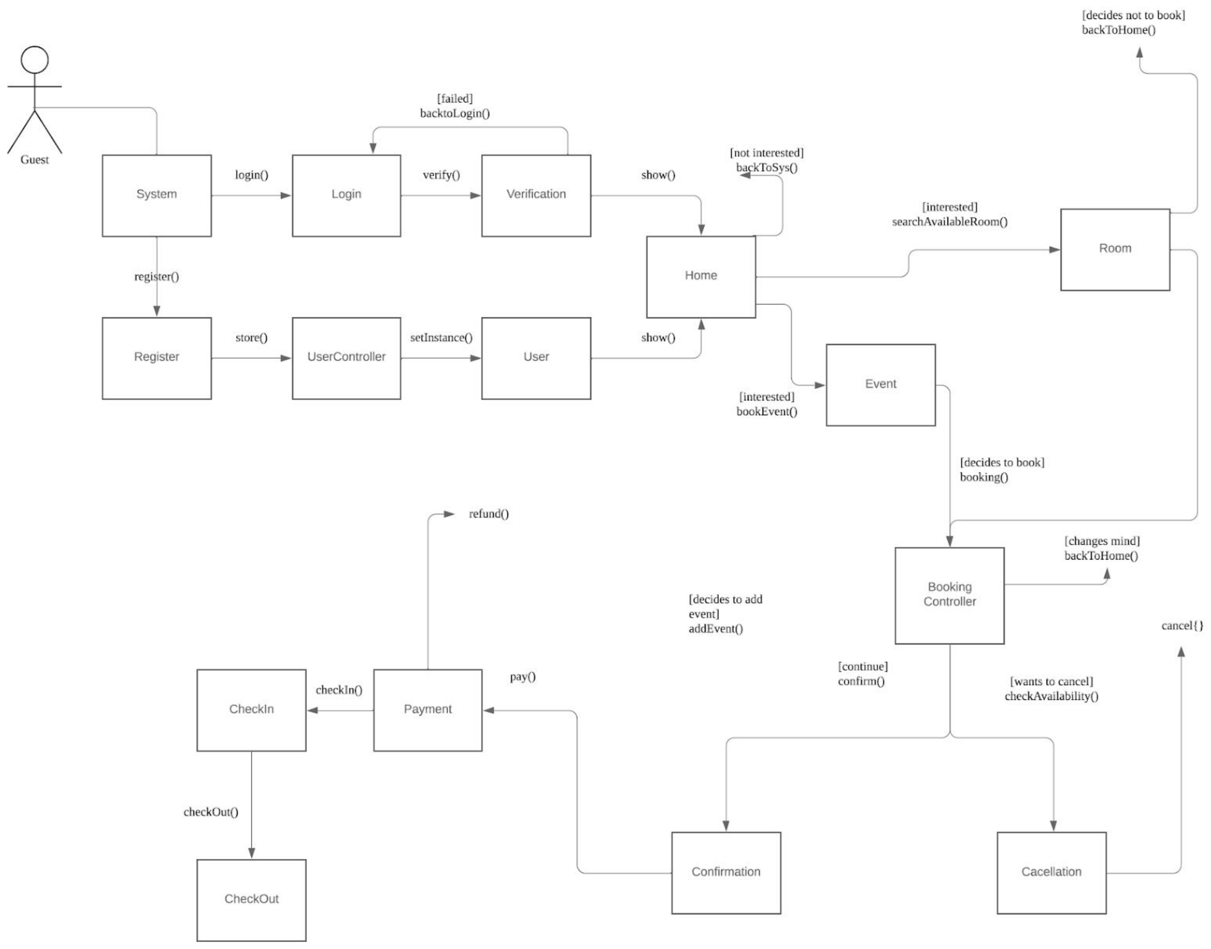
date	An attribute of booking
booking history	An attribute of booking
day	Measure of time, not a thing
member of hotel staff	Same as manager, use manager instead
guest details	Too vague, use guest instead
level	Similar to loyalty level, use loyalty level instead
system	Part of meta-language of functional requirements

Analysis-Time Class Diagram

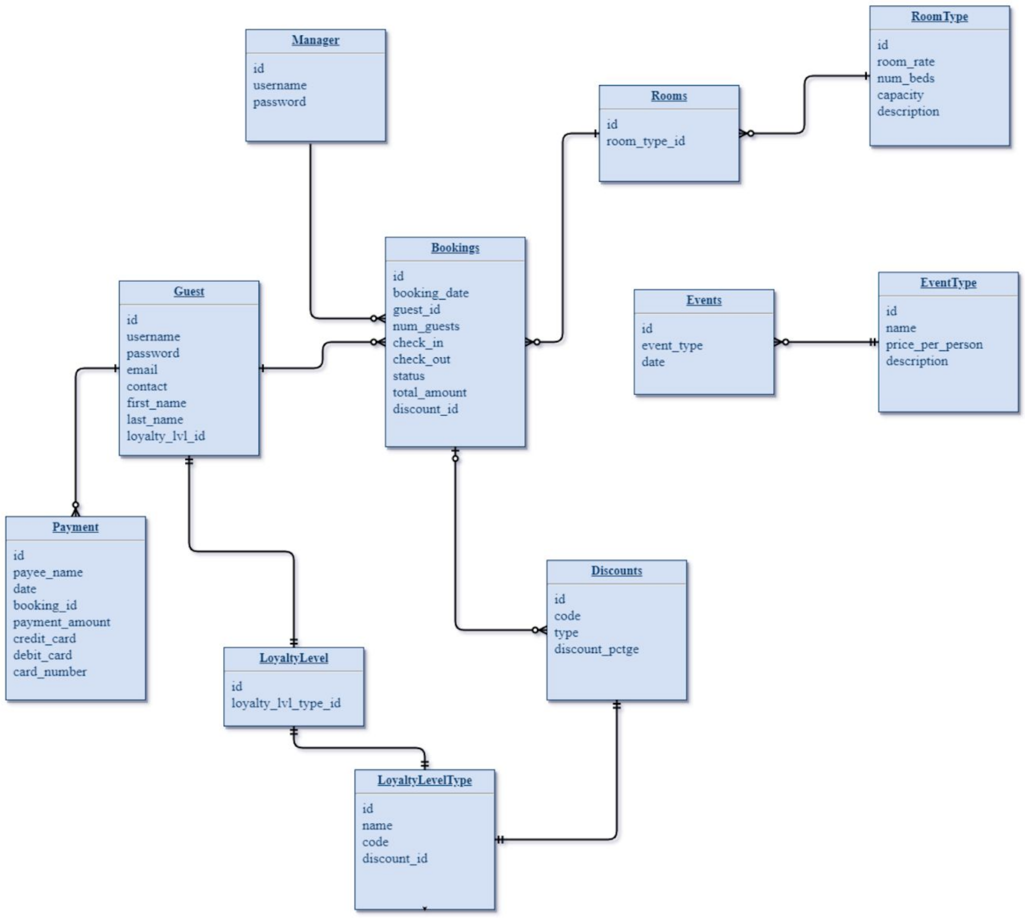



View
Home.jsp
Login.jsp
Register.jsp
Book.jsp
Manage.jsp
Room.jsp
Pay.jsp
Contact.jsp
Run
EZHotel
+ main(String [] args): void
Controller
<<Control>> UserController
- userService: IUserService
- userFactory: UserFactory
+ addUser(User user): void
+ managerLogin(): void
+ guestLogin(): void
+ register(): void
+ loyalty(): LoyaltyLevel
<<Control>> RoomController
- roomService: IRoomService
- roomFactory: RoomFactory
+ getRoom(int id): Room
+ addRoom(Room room): void
+ deleteRoom(int id): void
+ updateRoom(int id): void
<<Control>> EventController
- EventService: IEventService
- EventFactory: EventFactory
+ getEvent(int id): Event
+ addEvent(Event event): void
+ deleteEvent(int id): void
+ updateEvent(int id): void
<<Control>> BookingController
- bookingService: IBookingService
+ checkIn(): void
+ checkOut(): void
+ findAvailableRooms(): List<Room>
+ pay(): void
+ discount(): Discount
Service
<<interface>> IUserService
+ register(): void
+ login(): void
+ findUser(String username): User
UserServiceImplement
<<interface>> UserFactory
+ createUser(): User
<<interface>> IRoomService
+ checkAvailableRoom(): List<Room>
+ findRoom(int id): Room
+ getRoomType(): RoomType
RoomServiceImplement
<<interface>> RoomFactory
+ createRoom(): Room
<<interface>> IEventService
+ checkEvent(): List<Event>
+ getEventType(): EventType
<<interface>> EventFactory
+ createEvent(): Event
EventServiceImplement
<<interface>> IBookingService
+ booking(): void
+ checkin(): void
+ checkOut(): void
+ checkDiscount(): void
BookingServiceImplement
Dao
<<interface>> IUserDao
+ insert(): void
+ select(): User
+ update(): void
+ delete(): void
UserDaoImplement
<<interface>> IRoomDao
+ insert(): void
+ select(): Room
+ update(): void
+ delete(): void
UserDaoImplement
<<interface>> IEventDao
+ insert(): void
+ select(): Event
+ update(): void
+ delete(): void
UserDaoImplement
<<interface>> IBookingDao
+ insert(): void
+ select(): Booking
+ update(): void
+ delete(): void
UserDaoImplement

Communication Diagram



Entity Relationship Diagram with Cardinality



Code Breakdown

Package	**Class**	**Contributor**	**Lines of Code**
com.cyan.hotel	HotelApplication.java	Both	26
com.cyan.hotel.billingSystem	BillingSystem.java	John	34
com.cyan.hotel.controller	BookingController.java	Both	118
	EventController.java	Naichuan	18
	HomeController.java	Both	32
	RoomController.java	Naichuan	65
	UserController.java	Both	88
com.cyan.hotel.enumeration	LoyaltyLevel.java	Naichuan	8
	PayType.java	John	8
	RoomStyle.java	John	8
com.cyan.hotel.exception	ResourceNotFoundException.java	Both	14
	RoomNotFoundException.java	Both	12
com.cyan.hotel.model	Booking.java	John	99
	Discount.java	John	41
	DoubleRoom.java	Both	37
	Event.java	John	58
	ExecutiveRoom.java	Naichuan	38
	Guest.java	Naichuan	90
	JuniorSuiteRoom.java	Naichuan	41
	Manager.java	Naichuan	34
	Payment.java	John	79

	Room.java	Naichuan	74
	RoomDecorator.java	Naichuan	24
	RoomType.java	John	90
	SingleRoom.java	Naichuan	37
	User.java	Naichuan	166
	withAC.java	Naichuan	23
	withBottleOfWine.java	Naichuan	24
	withDinner.java	Naichuan	23
	withWifi.java	Naichuan	22
com.cyan.hotel.repository	BookingRepository.java	Both	16
	RoomRepository.java	Naichuan	31
	UserRepository.java	Both	31
	PaymentRepository.java	Both	9
com.cyan.hotel.repositoryService	BookingService.java	John	17
	BookingServiceImpl.java	John	43
	RegistrationService.java	John	5
	RegistrationServiceImpl.java	John	42
	RoomService.java	Naichuan	14
	RoomServiceImpl.java	Naichuan	53
	UserService.java	Both	17
	UserServiceImpl.java	Both	46
com.cyan.hotel.validator	UserValidator.java	Naichuan	60
webapp.WEB-INF.view	home.jsp	Both	35

	login.jsp	John	46
	register.jsp	Both	89
	error.jsp	Both	19
	event.jsp	Naichuan	32
	contact.jsp	Naichuan	32
	booking.jsp	Naichuan	86
	about.jsp	Both	54
	room.jsp	Naichuan	84
	welcome.jsp	John	19
	payment.jsp	John	75
webapp.resources	main.css	Both	11
	main.js	Both	-
	nav.jsp	Both	37
resources	application.properties	Both	34
	import.sql	Both	34
	userValidation.properties	Both	4

Total Lines of Code Developed:

Nearing the end of project deadline pair programming had to be implemented in order to get a working system and troubleshoot problems.

Team Member	Team Contribution (LoC)
John Long	1112
Naichuan Zhang	1731

Total Num Packages	Total Number of Classes and Files	Total Lines of Code
8	60	2843

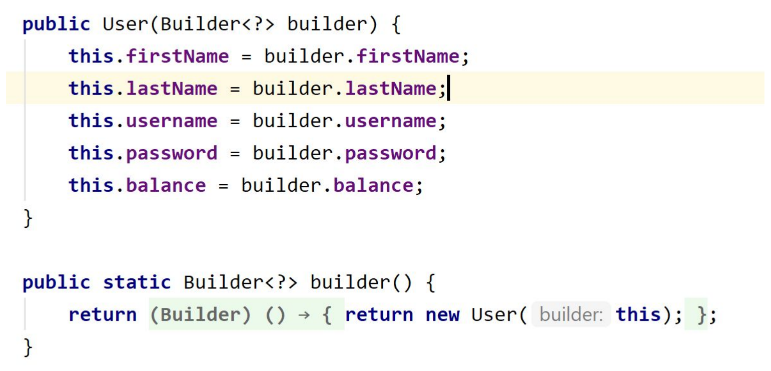
Code

Builder Pattern

To facilitate the different types of users that could potentially be part of our system like Guest and Manager we used the Builder design pattern to implement this. This pattern is a creational design pattern.

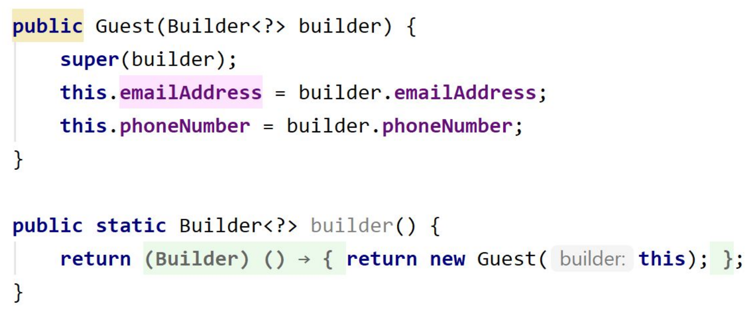
It is usually an inner class that provides a series of set methods as well as a build method that creates the object.

User



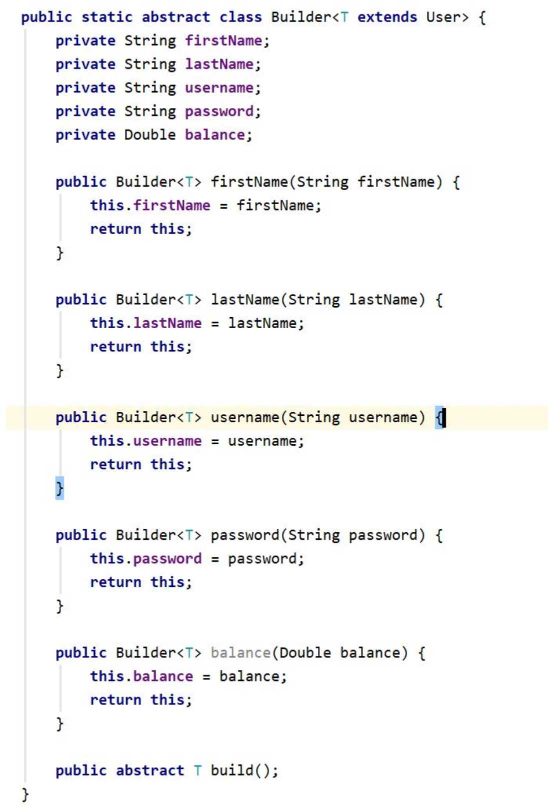
```
public User(Builder<?> builder) {
    this.firstName = builder.firstName;
    this.lastName = builder.lastName;
    this.username = builder.username;
    this.password = builder.password;
    this.balance = builder.balance;
}

public static Builder<?> builder() {
    return (Builder) () → { return new User( builder: this); };
}
```

Guest



```
public Guest(Builder<?> builder) {
    super(builder);
    this.emailAddress = builder.emailAddress;
    this.phoneNumber = builder.phoneNumber;
}

public static Builder<?> builder() {
    return (Builder) () → { return new Guest( builder: this); };
}
```

User

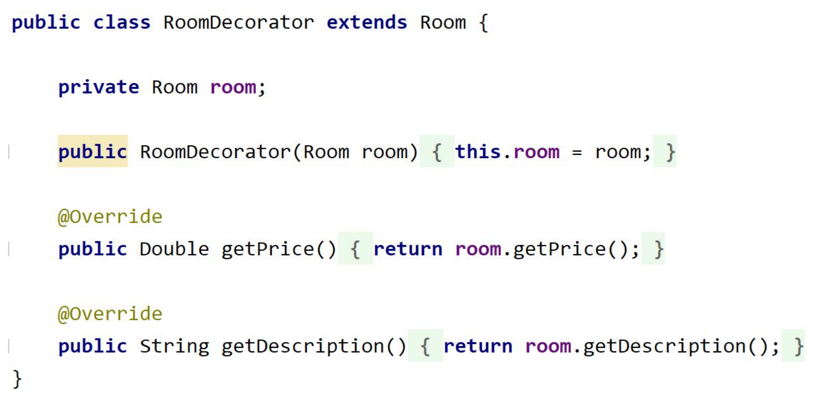

```java
public static abstract class Builder<T extends User> {
    private String firstName;
    private String lastName;
    private String username;
    private String password;
    private Double balance;

    public Builder<T> firstName(String firstName) {
        this.firstName = firstName;
        return this;
    }

    public Builder<T> lastName(String lastName) {
        this.lastName = lastName;
        return this;
    }

    public Builder<T> username(String username) {
        this.username = username;
        return this;
    }

    public Builder<T> password(String password) {
        this.password = password;
        return this;
    }

    public Builder<T> balance(Double balance) {
        this.balance = balance;
        return this;
    }

    public abstract T build();
}
```

By using the Builder pattern we benefit by avoiding a large number of parameters in the constructor by providing a constructor with required parameter and then setter methods to set the optional parameters.

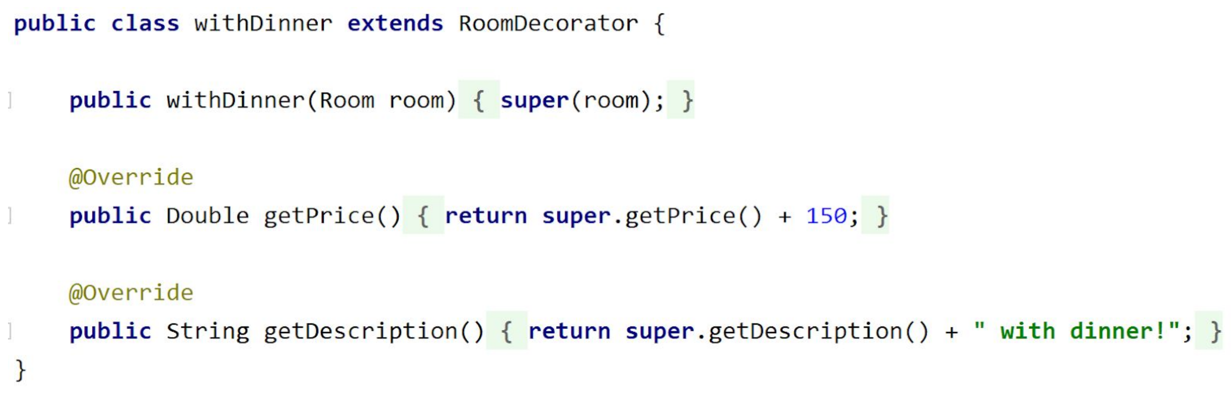
Decorator Pattern

In order to support additional room extras as part of a room booking we have utilised the decorator pattern.

The decorator design pattern is helpful in that it provides runtime modification abilities and more flexibility. It is a structural design pattern and can attach responsibilities to an object either statically or dynamically.



```java
public class RoomDecorator extends Room {

    private Room room;

    public RoomDecorator(Room room) { this.room = room; }

    @Override
    public Double getPrice() { return room.getPrice(); }

    @Override
    public String getDescription() { return room.getDescription(); }
}
```

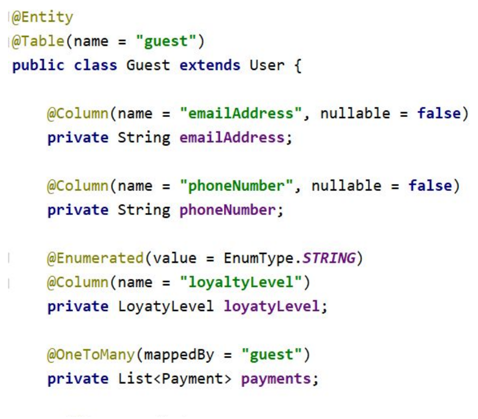


```java
public class withDinner extends RoomDecorator {

    public withDinner(Room room) { super(room); }

    @Override
    public Double getPrice() { return super.getPrice() + 150; }

    @Override
    public String getDescription() { return super.getDescription() + " with dinner!"; }
}
```

Model View Controller (MVC) Pattern

The models we used in the project are used to map to the database. In order to do this we used the Java Persistence API. In order for JPA to map a table to the database we must define an entity and the columns in that table.

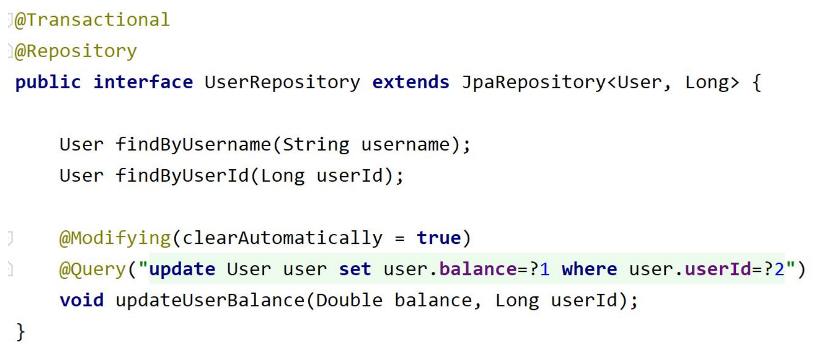
Guest Entity



```java
@Entity
@Table(name = "guest")
public class Guest extends User {

    @Column(name = "emailAddress", nullable = false)
    private String emailAddress;

    @Column(name = "phoneNumber", nullable = false)
    private String phoneNumber;

    @Enumerated(value = EnumType.STRING)
    @Column(name = "loyaltyLevel")
    private LoyatyLevel loyatyLevel;

    @OneToMany(mappedBy = "guest")
    private List<Payment> payments;
```

We then need an interface to access the data which are our repositories. Here we can use the CRUD operations that JPA provides or else we can define custom queries should the need arise.

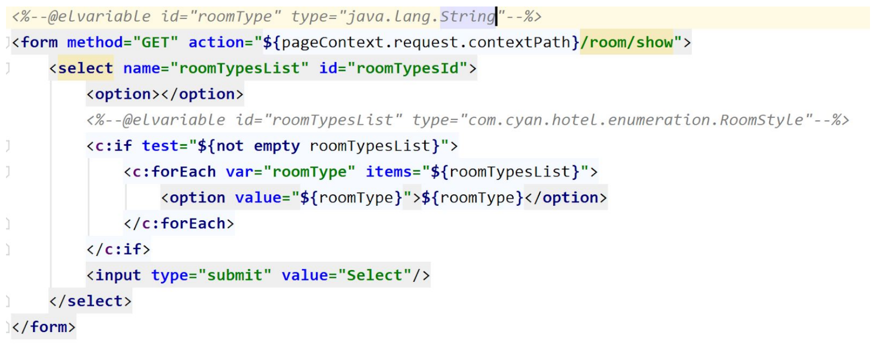
User Repository



```java
@Transactional
@Repository
public interface UserRepository extends JpaRepository<User, Long> {

    User findByUsername(String username);
    User findByUserId(Long userId);

    @Modifying(clearAutomatically = true)
    @Query("update User user set user.balance=?1 where user.userId=?2")
    void updateUserBalance(Double balance, Long userId);
}
```

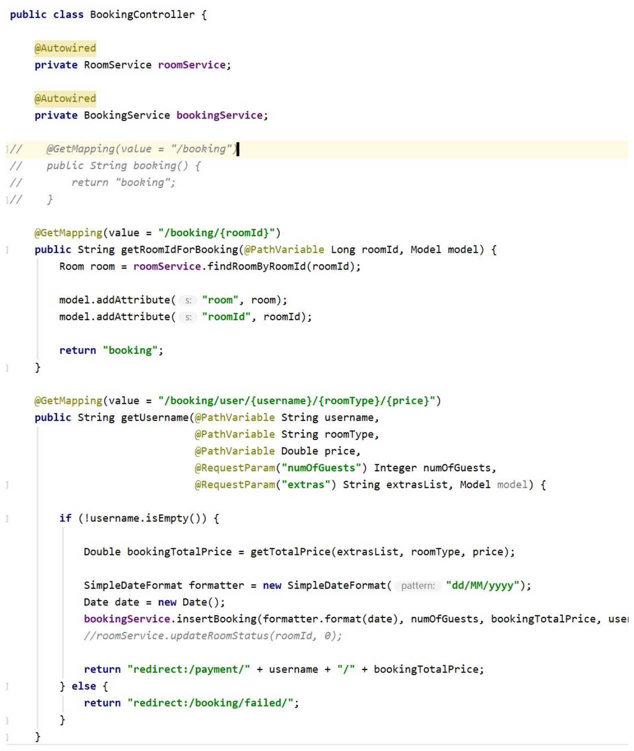
For our views in the project we used Jsp files and to pass data between the files we used some of Spring Boots built in capabilities.

Example code from room view



```
<%--@elvariable id="roomType" type="java.lang.String"--%>
<form method="GET" action="${pageContext.request.contextPath}/room/show">
    <select name="roomTypesList" id="roomTypesId">
        <option></option>
        <%--@elvariable id="roomTypesList" type="com.cyan.hotel.enumeration.RoomStyle"--%>
        <c:if test="${not empty roomTypesList}">
            <c:forEach var="roomType" items="${roomTypesList}">
                <option value="${roomType}">${roomType}</option>
            </c:forEach>
        </c:if>
        <input type="submit" value="Select"/>
    </select>
</form>
```

Finally we used controllers to act as an intermediary between our views and our models. Each controller was mapped to perform a specific operation on a page and is called when the endpoint is triggered. The controller classes use the models and the repository classes to perform operations that the application needs.

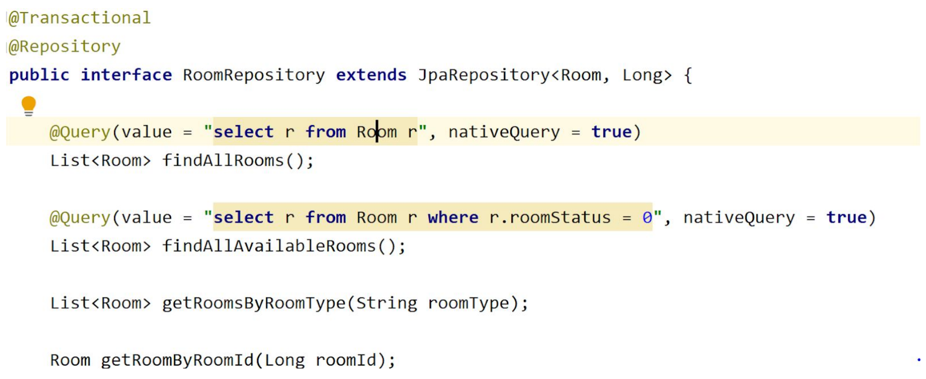


```
public class BookingController {

    @Autowired
    private RoomService roomService;

    @Autowired
    private BookingService bookingService;

//    @GetMapping(value = "/booking")
//    public String booking() {
//        return "booking";
//    }

    @GetMapping(value = "/booking/{roomId}")
    public String getRoomIdForBooking(@PathVariable Long roomId, Model model) {
        Room room = roomService.findRoomByRoomId(roomId);

        model.addAttribute( s: "room", room);
        model.addAttribute( s: "roomId", roomId);

        return "booking";
    }

    @GetMapping(value = "/booking/user/{username}/{roomType}/{price}")
    public String getUsername(@PathVariable String username,
                              @PathVariable String roomType,
                              @PathVariable Double price,
                              @RequestParam("numOfGuests") Integer numOfGuests,
                              @RequestParam("extras") String extrasList, Model model) {

        if (!username.isEmpty()) {

            Double bookingTotalPrice = getTotalPrice(extrasList, roomType, price);

            SimpleDateFormat formatter = new SimpleDateFormat( pattern: "dd/MM/yyyy");
            Date date = new Date();
            bookingService.insertBooking(formatter.format(date), numOfGuests, bookingTotalPrice, use
            //roomService.updateRoomStatus(roomId, 0);

            return "redirect:/payment/" + username + "/" + bookingTotalPrice;
        } else {
            return "redirect:/booking/failed/";
        }
    }
```

Repository Design Pattern

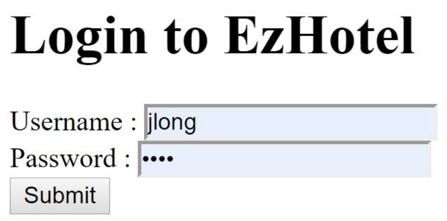
We used the repository design pattern to separate the storing of our data from the model implementation.

Repository

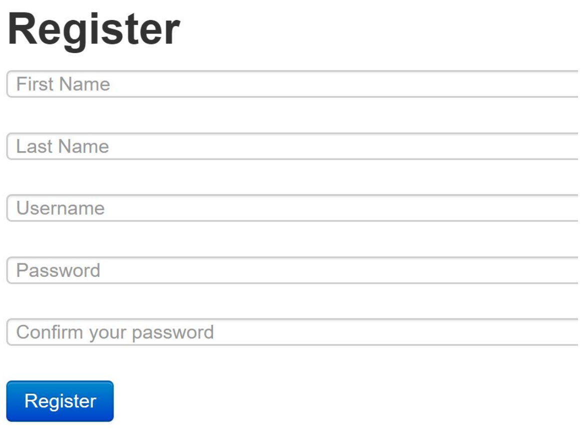


```
@Transactional
@Repository
public interface RoomRepository extends JpaRepository<Room, Long> {

    @Query(value = "select r from Room r", nativeQuery = true)
    List<Room> findAllRooms();

    @Query(value = "select r from Room r where r.roomStatus = 0", nativeQuery = true)
    List<Room> findAllAvailableRooms();

    List<Room> getRoomsByRoomType(String roomType);

    Room getRoomByRoomId(Long roomId);
```

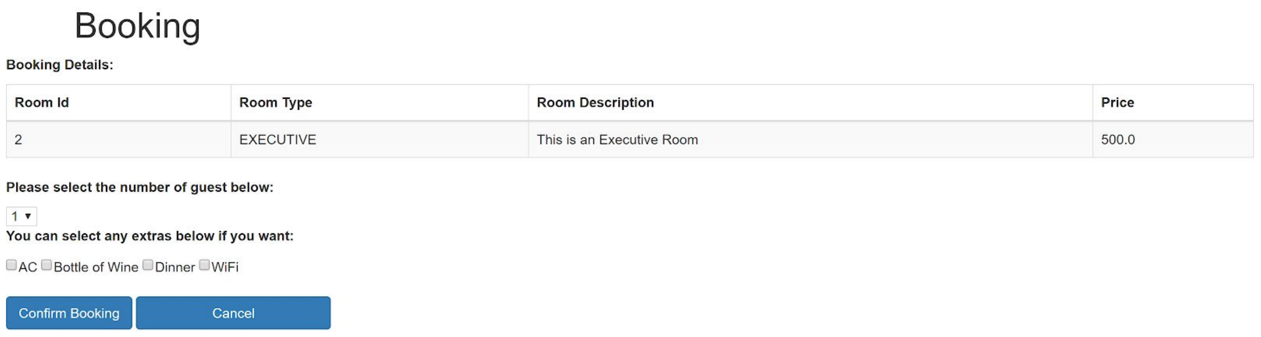
GUI Screenshots


Login to EzHotel

Username : jlong
Password : ••••


Register

First Name
Last Name
Username
Password
Confirm your password


Booking

Booking Details:

Room Id	Room Type	Room Description	Price
2	EXECUTIVE	This is an Executive Room	500.0

Please select the number of guest below:

1 ▾

You can select any extras below if you want:

☐AC ☐Bottle of Wine ☐Dinner ☐WiFi

Added Value

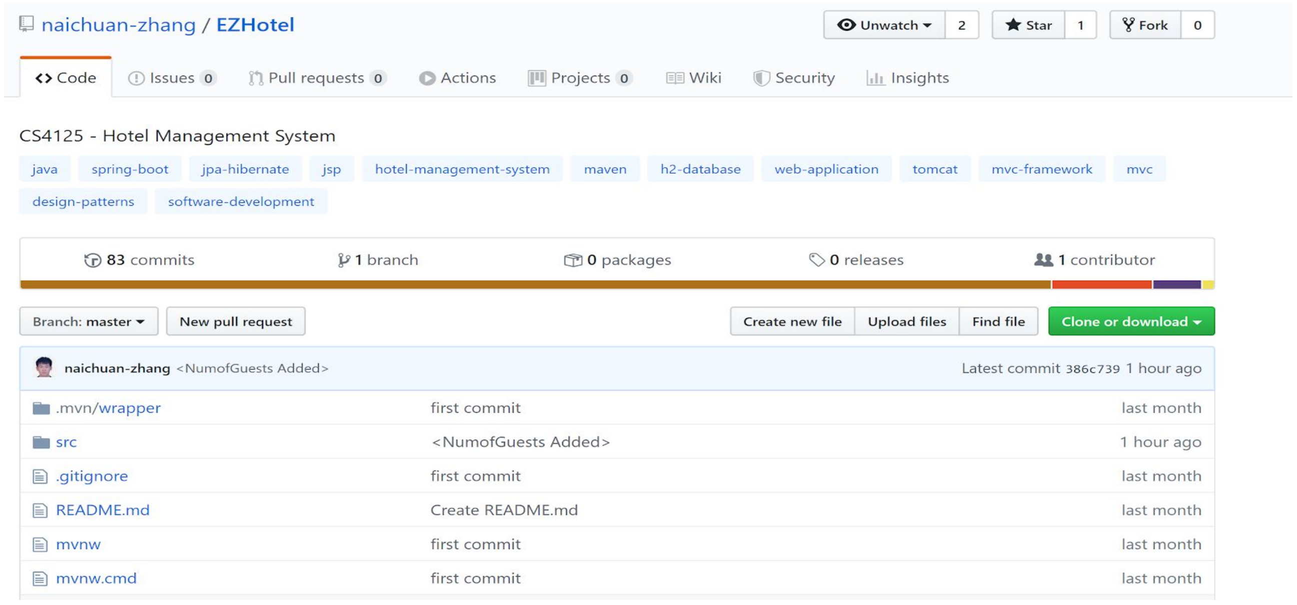
Spring Boot

For the hotel management we wanted to implement our system as a web service and to do that we used the Spring Boot Framework. Spring Boot makes it easy to create stand-alone, production grade Spring Applications and get a web application up and running easily. Spring Boot applications need a lot less Spring configuration than others and reduces the amount of boilerplate code that often needs to be implemented in projects. It allows easy deployment on HTTP servers like Tomcat as they are embedded in the project. Spring also enables you to autowire beans on the setter method, constructor or field name.

Version Control - Github

https://github.com/naichuan-zhang/EZHotel

Github was used as our version control system throughout the project implementation to maintain and collaborate on the developed code. Due to the small nature of our team we would not need to use branching as part of the development process. Instead we would commit small changes notifying the other person before doing so. This lead to less merge conflicts and enabled us to develop more effectively. Using IntelliJ as our IDE also worked well with GitHub as it provided a simple visual when merging a commit with the master repository.



CS4125 - Hotel Management System

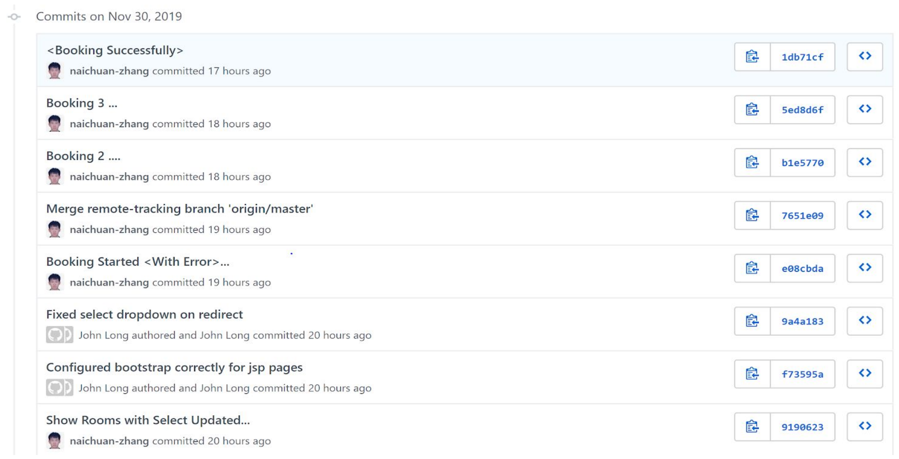
java spring-boot jpa-hibernate jsp hotel-management-system maven h2-database web-application tomcat mvc-framework mvc design-patterns software-development

83 commits | 1 branch | 0 packages | 0 releases | 1 contributor

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

naichuan-zhang <NumofGuests Added> Latest commit 386c739 1 hour ago

.mvn/wrapper	first commit	last month
src	<NumofGuests Added>	1 hour ago
.gitignore	first commit	last month
README.md	Create README.md	last month
mvnw	first commit	last month
mvnw.cmd	first commit	last month



Commits on Nov 30, 2019

<Booking Successfully>
naichuan-zhang committed 17 hours ago
1db71cf

Booking 3 ...
naichuan-zhang committed 18 hours ago
5ed8d6f

Booking 2 ....
naichuan-zhang committed 18 hours ago
b1e5770

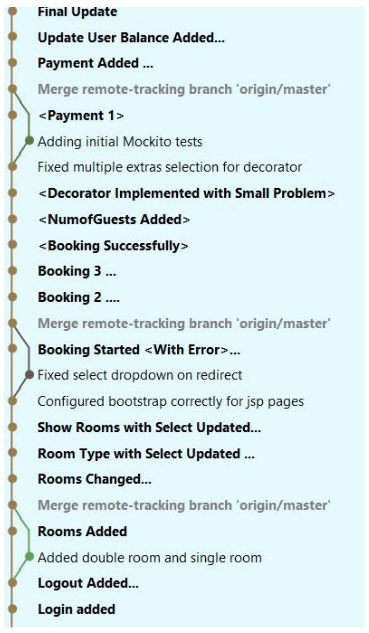
Merge remote-tracking branch 'origin/master'
naichuan-zhang committed 19 hours ago
7651e09

Booking Started <With Error>...
naichuan-zhang committed 19 hours ago
e08cbda

Fixed select dropdown on redirect
John Long authored and John Long committed 20 hours ago
9a4a183

Configured bootstrap correctly for jsp pages
John Long authored and John Long committed 20 hours ago
f73595a

Show Rooms with Select Updated...
naichuan-zhang committed 20 hours ago
9190623



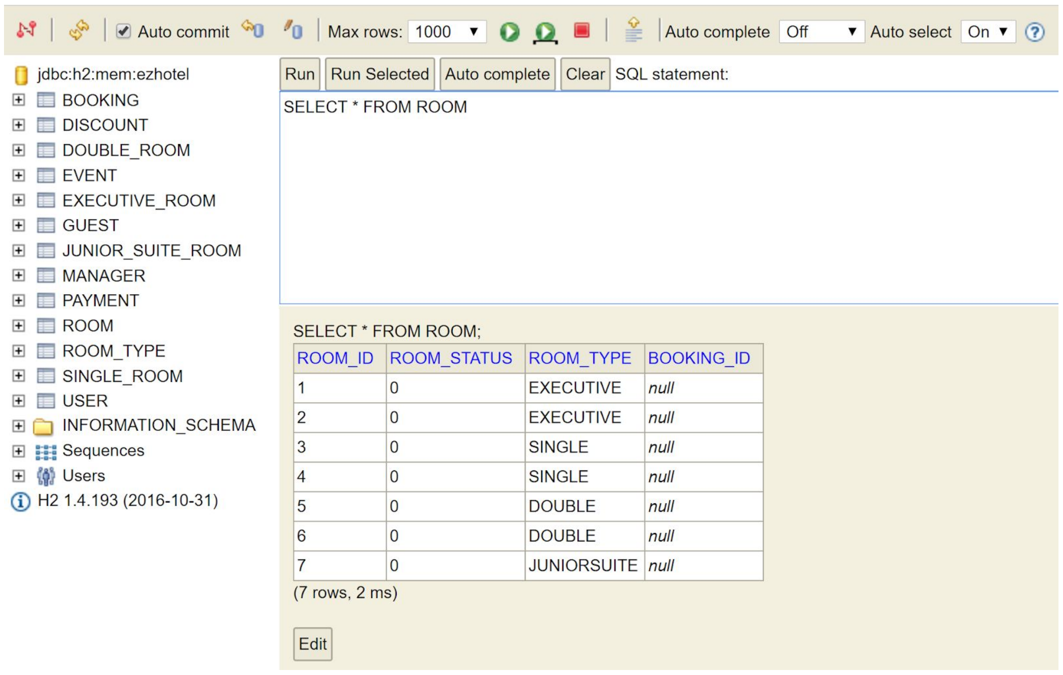
Final Update
Update User Balance Added...
Payment Added ...
Merge remote-tracking branch 'origin/master'
<Payment 1>
Adding initial Mockito tests
Fixed multiple extras selection for decorator
<Decorator Implemented with Small Problem>
<NumofGuests Added>
<Booking Successfully>
Booking 3 ...
Booking 2 ....
Merge remote-tracking branch 'origin/master'
Booking Started <With Error>...
Fixed select dropdown on redirect
Configured bootstrap correctly for jsp pages
Show Rooms with Select Updated...
Room Type with Select Updated ...
Rooms Changed...
Merge remote-tracking branch 'origin/master'
Rooms Added
Added double room and single room
Logout Added...
Login added

Google Docs

We also used Google Docs as a version control method for the reporting part of the project which allowed us to track the progress of the project and update the other team member once a change to the report was made.

Database Implementation

For our database we decided to use the h2 in-memory database. We coupled this with the Java Persistence API (JPA) which let us define which objects should be persisted and how those objects should be persisted in our Java application. Using JPA provided us with an ORM layer. This enabled us to avoid doing database and relational mapping manually. The ORM layer is responsible for managing the conversion of software objects to interact with the tables and columns in a relational database. By default, the name of the object being persisted becomes the name of the table, and fields become columns. Once the table is set up, each table row corresponds to an object in the application. Object mapping is configurable, but defaults tend to work well also. When you use JPA, you create a map from the datastore to the application's data model objects. Instead of defining how objects are saved and retrieved, you define the mapping between objects and your database, then invoke JPA to persist them. To use h2 as the in-memory database all that required was to add it to our application.properties file and the rest was taken care of by Spring and JPA.



jdbc:h2:mem:ezhotel
- BOOKING
- DISCOUNT
- DOUBLE_ROOM
- EVENT
- EXECUTIVE_ROOM
- GUEST
- JUNIOR_SUITE_ROOM
- MANAGER
- PAYMENT
- ROOM
- ROOM_TYPE
- SINGLE_ROOM
- USER
- INFORMATION_SCHEMA
- Sequences
- Users

H2 1.4.193 (2016-10-31)

```
SELECT * FROM ROOM
```

SELECT * FROM ROOM;

ROOM_ID	ROOM_STATUS	ROOM_TYPE	BOOKING_ID
1	0	EXECUTIVE	*null*
2	0	EXECUTIVE	*null*
3	0	SINGLE	*null*
4	0	SINGLE	*null*
5	0	DOUBLE	*null*
6	0	DOUBLE	*null*
7	0	JUNIORSUITE	*null*

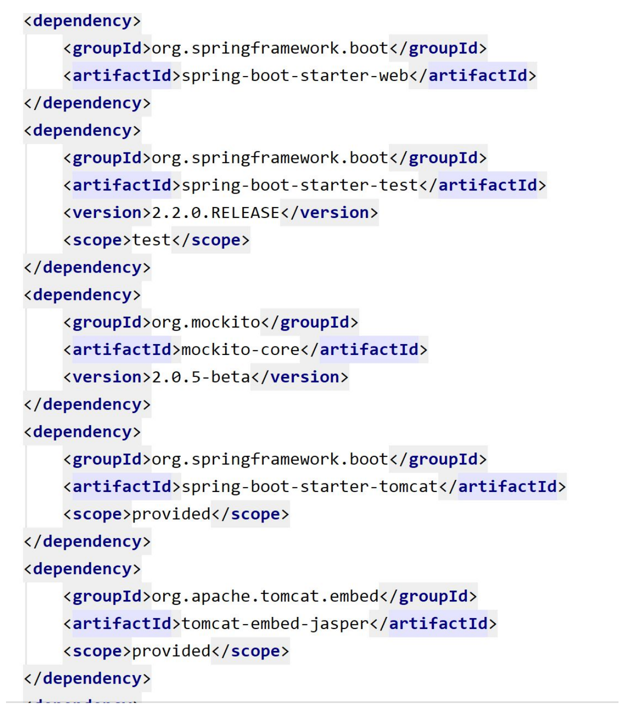
(7 rows, 2 ms)

Mockito

To acknowledge the role of test driven development we added tests for the booking controller class. We did this using Mockito a testing framework that can facilitate the writing of clean and simple tests. To add the dependency to our project we again used Maven to manage this.

Maven

Maven is a build automation tool primarily used for Java projects. We used Maven to manage our dependencies by making use of the POM or Project Object Model. It is an XML representation of a Maven project held in a file named pom.xml. This file is used to handle project relationships like dependencies. The main advantage of using the Maven POM is its dependency list. Maven downloads and links the dependencies on compilation and other goals that require them. As an added bonus, Maven brings in the dependencies of those dependencies (transitive dependencies), allowing your list to focus solely on the dependencies your project requires. This was what made using Spring Boot and JPA in our project easier as all we had to do was include their dependencies in the list.



```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <version>2.2.0.RELEASE</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
    <version>2.0.5-beta</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-tomcat</artifactId>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-jasper</artifactId>
    <scope>provided</scope>
</dependency>
```

Builder Design Pattern

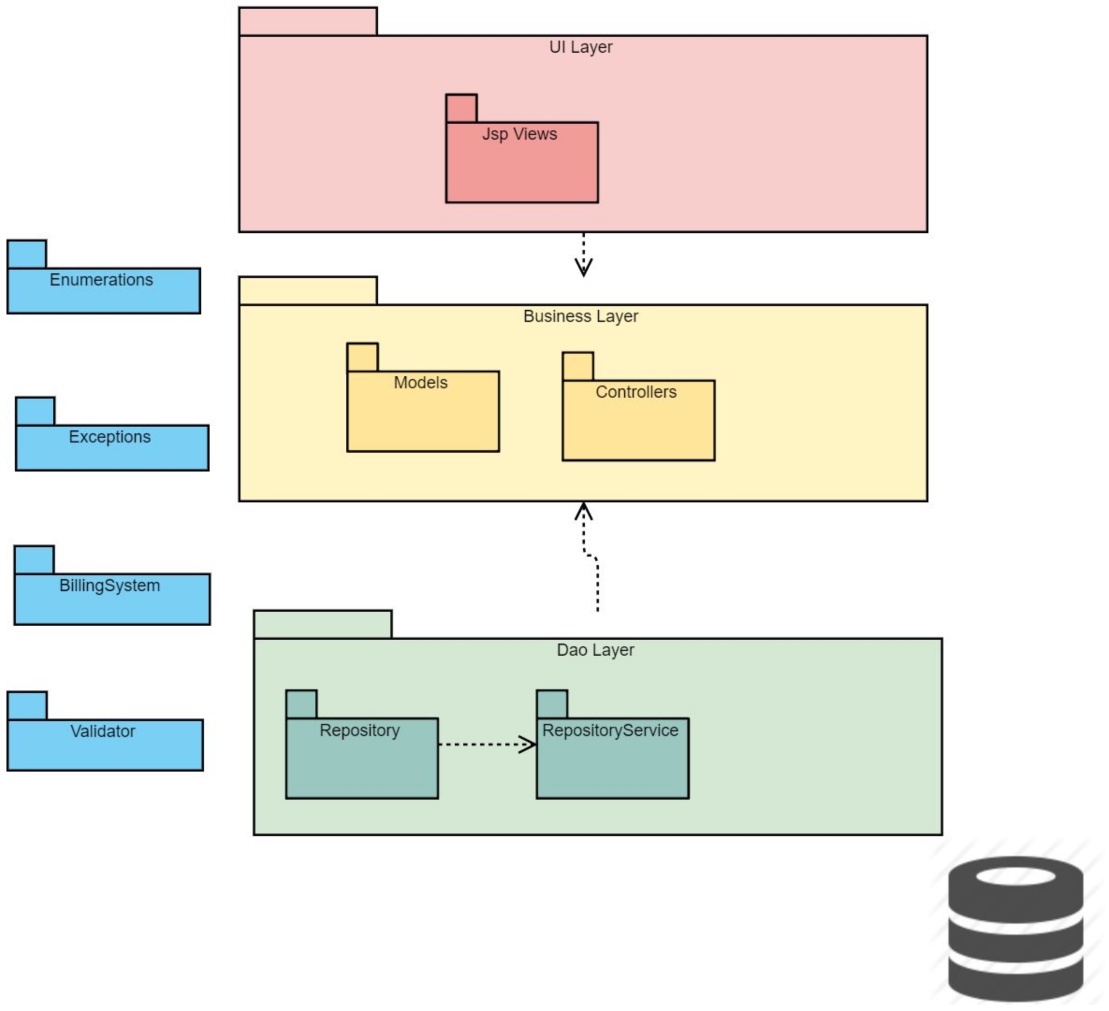
Refer to coding fragments for use of builder pattern in implementation

Repository Pattern

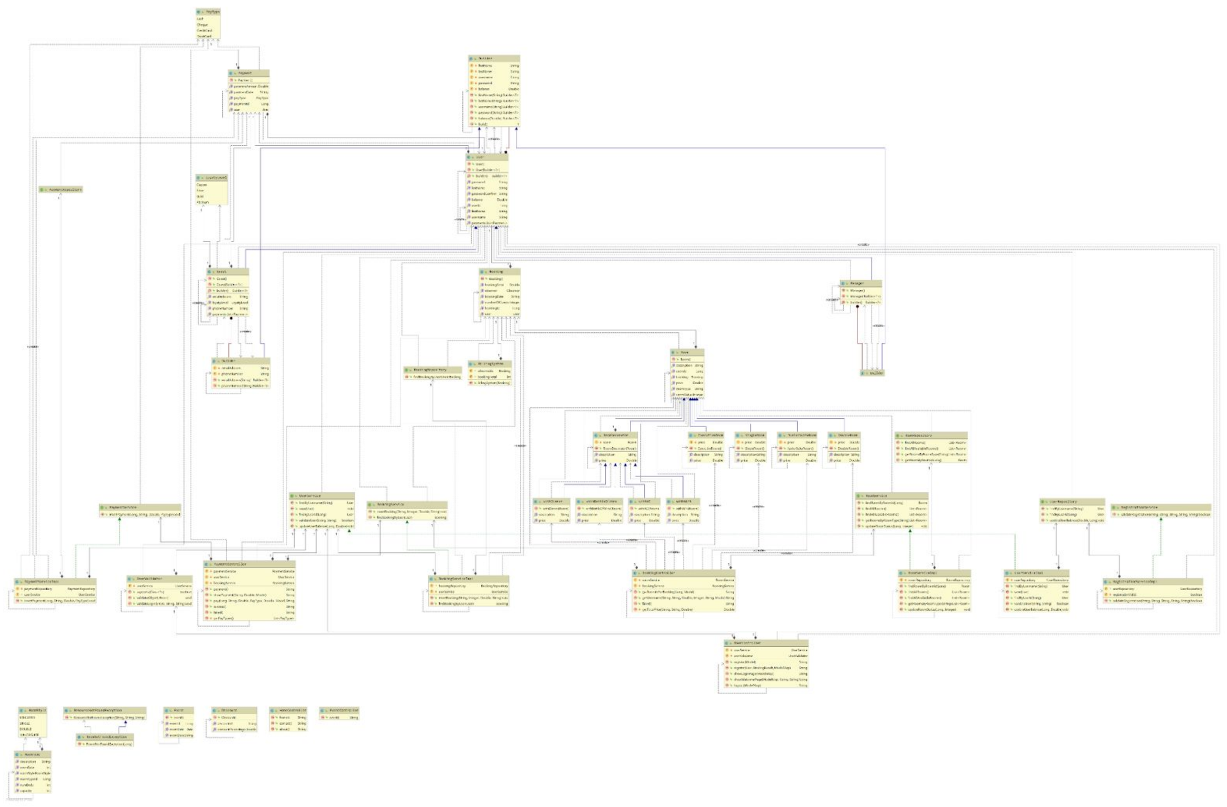
Refer to coding fragments for use of repository pattern in implementation

Recovered Architecture and Design Blueprints

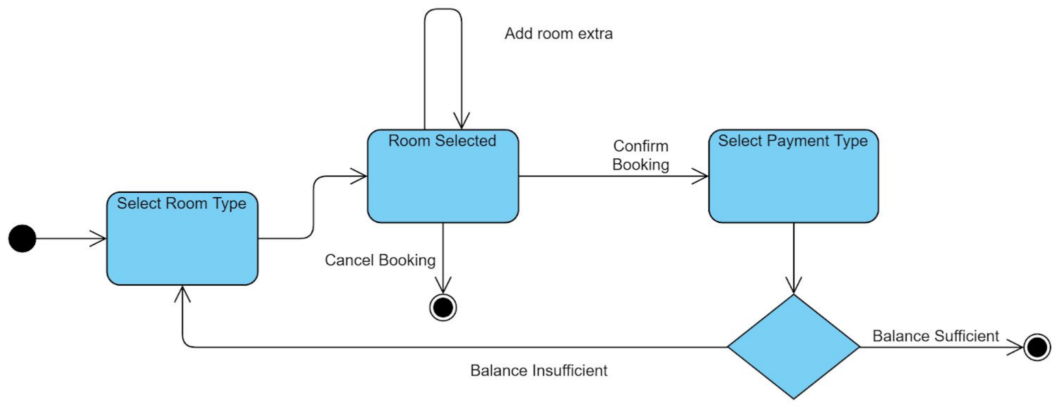
Architectural Diagram



Design-Time Analysis Class Diagram

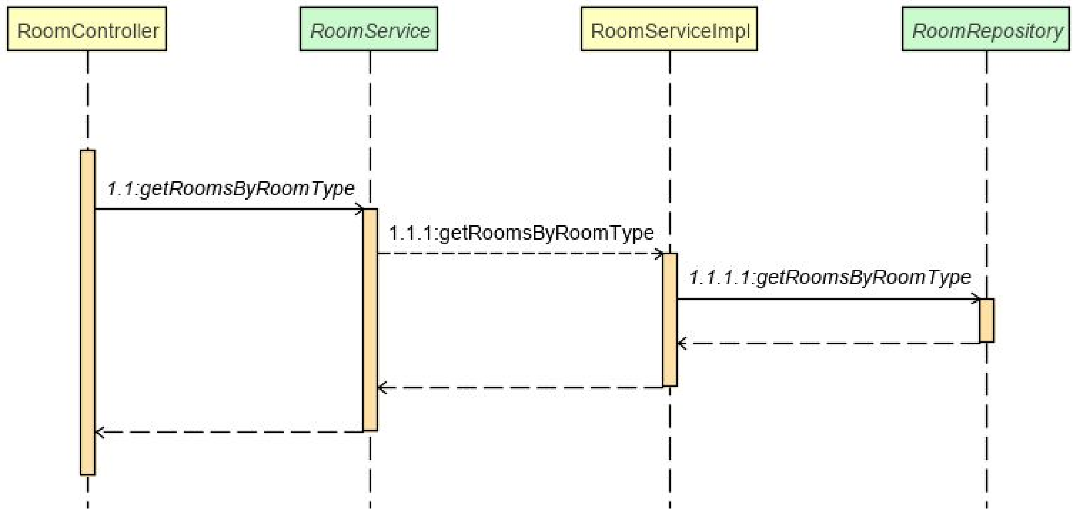


State Chart

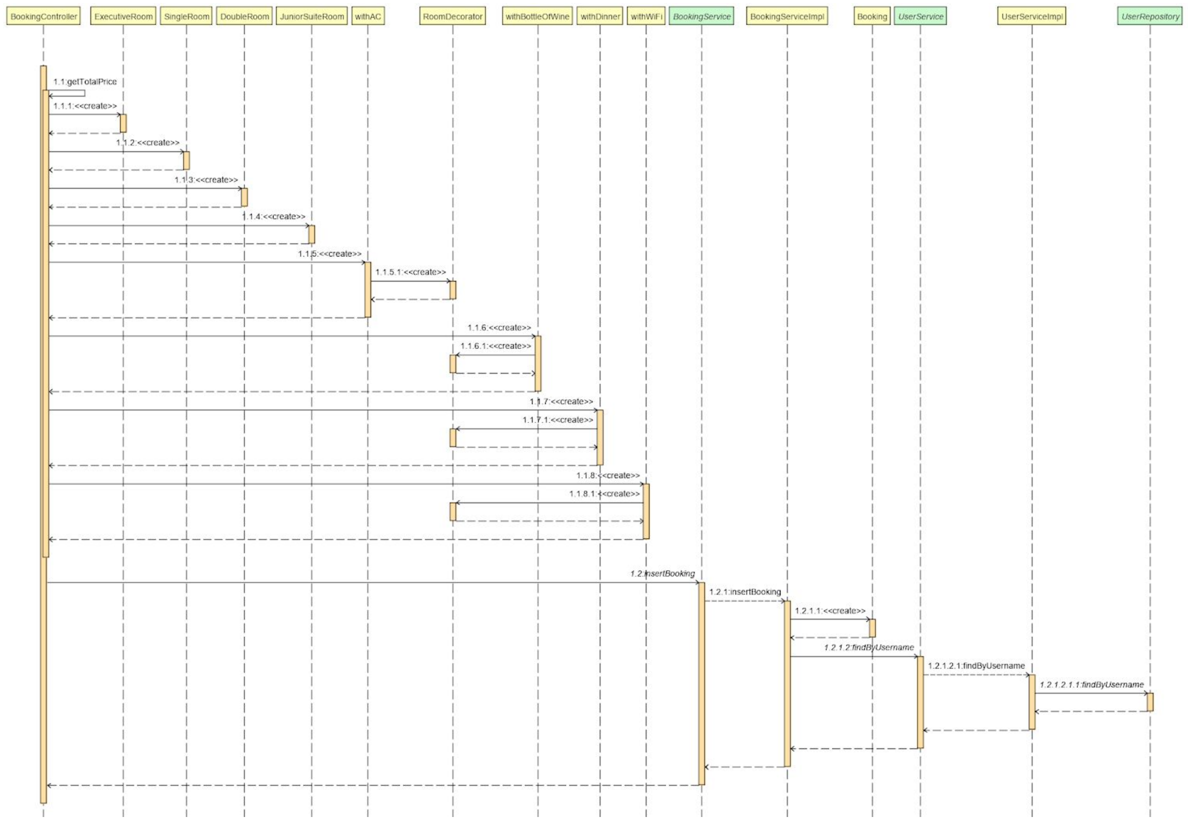


Sequence Diagrams

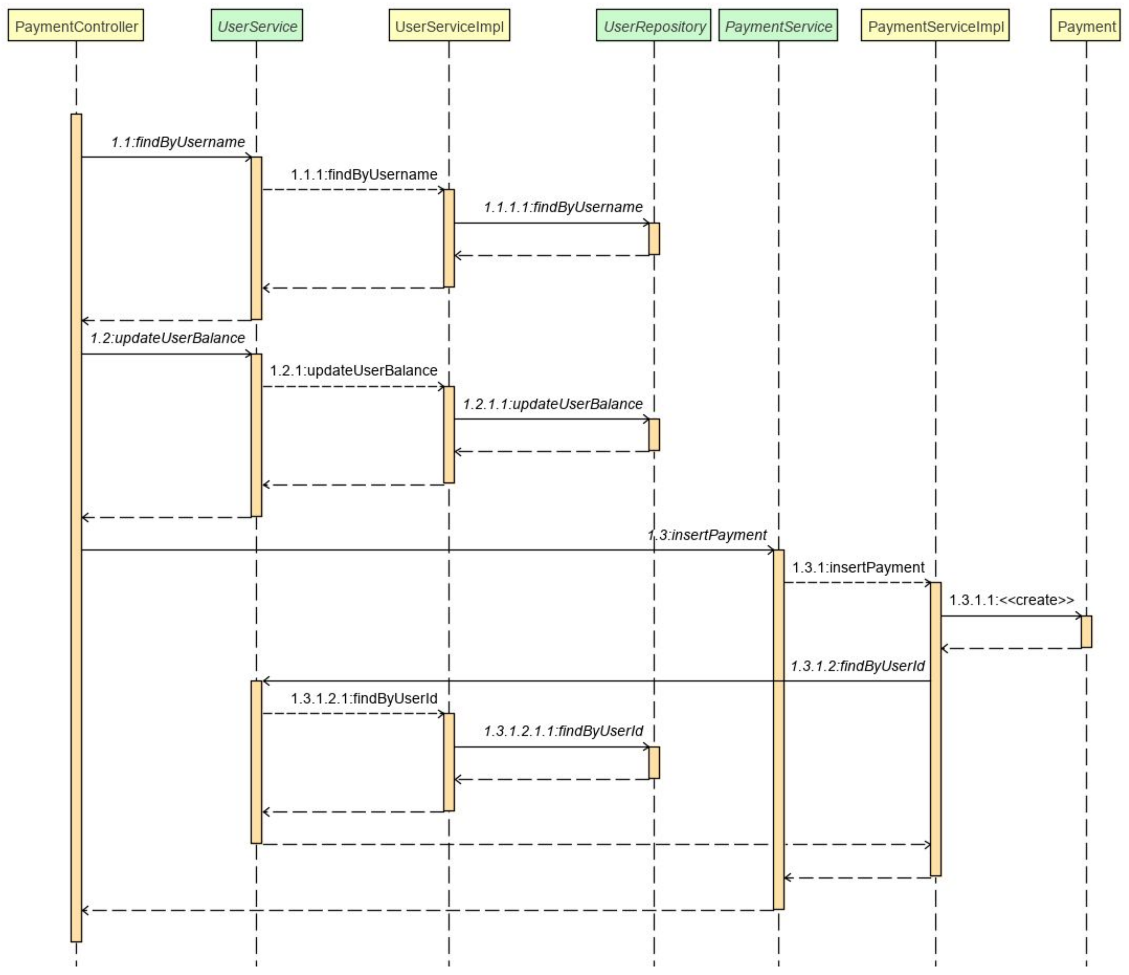
For Room



For Booking


BookingController
ExecutiveRoom
SingleRoom
DoubleRoom
JuniorSuiteRoom
withAC
RoomDecorator
withBottleOfWine
withDinner
withWiFi
BookingService
BookingServiceImpl
Booking
UserService
UserServiceImpl
UserRepository
1.1:getTotalPrice
1.1.1:<<create>>
1.1.2:<<create>>
1.1.3:<<create>>
1.1.4:<<create>>
1.1.5:<<create>>
1.1.5.1:<<create>>
1.1.6:<<create>>
1.1.6.1:<<create>>
1.1.7:<<create>>
1.1.7.1:<<create>>
1.1.8:<<create>>
1.1.8.1:<<create>>
1.2:insertBooking
1.2.1:insertBooking
1.2.1.1:<<create>>
1.2.1.2:findByUsername
1.2.1.2.1:findByUsername
1.2.1.2.1.1:findByUsername

For Payment


PaymentController
UserService
UserServiceImpl
UserRepository
PaymentService
PaymentServiceImpl
Payment
1.1:findByUsername
1.1.1:findByUsername
1.1.1.1:findByUsername
1.2:updateUserBalance
1.2.1:updateUserBalance
1.2.1.1:updateUserBalance
1.3:insertPayment
1.3.1:insertPayment
1.3.1.1:<<create>>
1.3.1.2:findByUserId
1.3.1.2.1:findByUserId
1.3.1.2.1.1:findByUserId

Critique

Language, Architecture and Framework Selection

The language and frameworks we used in the project were selected for a number of reasons. Java was chosen as our main language to program business logic as the members of the group were familiar with the language and had used it in their studies.

We chose Jpa and Spring Boot as they are well known and well documented if we ran into problems in the project. We felt that we would gain added value for the project by using these frameworks and showing implementation using them.

Having never really used these frameworks before we did encounter problems during the project when trying to implement the frontend of the application through Jsp pages.

The architecture has changed in that we have added some more packages to support the original architecture but overall the structure and idea is very similar.

Design Patterns

Upon starting this project our knowledge of design patterns was very basic having only been introduced to them in this module in a meaningful way. The main pattern we stuck to was the Model View Controller and as we proceeded in the project phases we added patterns when necessary to show competence.

Rather than set out to use design patterns we instead refactored the code to use the patterns which is something we would change should we have to do a project like this again. Knowing where and why we would use our patterns in our application would have aided in the development process and prevented needless refactoring.

Analysis Diagrams

Our diagrams are far more detailed now after the actual implementation and recovering the design time blueprint. The overall structure of our project did not change drastically which showed that we stuck to our original design and expanded upon it.

References

Images

Waterfall Model:

Powell-Morse, A. (2019). *Waterfall Model: What Is It and When Should You Use It?*. [online] Airbrake Blog. Available at: https://airbrake.io/blog/sdlc/waterfall-model [Accessed 1 Nov. 2019].

V-Model:

Google.com. (2019). *Image: V Model SDLC: Verification and Validation Model |Professionalqa.com*. [online] Available at: https://www.google.com/imgres?imgurl=http%3A%2F%2Fwww.professionalqa.com%2Fassets%2Fimages%2Fv-model.png&imgrefurl=http%3A%2F%2Fwww.professionalqa.com%2Fv-model&docid=v2mvtXbu6nyNTM&tbnid=PVaEM_V4IXZsaM%3A&vet=10ahUKEwiQ9qCC_rflAhUxQUEAHUMuD10QMwh4KAEwAQ..i&w=459&h=443&bih=840&biw=853&q=V%20model&ved=0ahUKEwiQ9qCC_rflAhUxQUEAHUMuD10QMwh4KAEwAQ&iact=mrc&uact=8 [Accessed 1 Dec. 2019].

Agile:

Google.com. (2019). *Redirect Notice*. [online] Available at: https://www.google.com/url?sa=i&source=images&cd=&ved=2ahUKEwj4lYaK_7flAhWuVBUIHUt8D50QjRx6BAgBEAQ&url=https%3A%2F%2Fwww.javatpoint.com%2Fsoftware-engineering-agile-model&psig=AOvVaw0wn7PGEvvA5ANP-L2A6Nzr&ust=1572112842343067 [Accessed 1 Nov. 2019].

Added Value

JPA:

Tyson, M. (2019). *What is JPA? Introduction to the Java Persistence API*. [online] JavaWorld. Available at: https://www.javaworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html [Accessed 1 Dec. 2019].

Spring Boot:

Spring.io. (2019). *Spring Projects*. [online] Available at: https://spring.io/projects/spring-boot [Accessed 1 Dec. 2019].

H2 Database:

H2database.com. (2019). *Features*. [online] Available at: https://www.h2database.com/html/features.html#in_memory_databases [Accessed 1 Dec. 2019].

CS4125: Systems Analysis and Design. Semester 1, 2019-2020

Guidance on the MARKING SCHEME for Team-Based Project: Version 2 (24th September 2019 Week 3)

Name:	ID:
Name:	ID:
Name:	ID:
Name	ID:

	Item	Detailed Description	Marks Allocated		Marks Awarded
			Sub-total	Total	
	Presentation	• General Presentation • Adherence to guidelines i.e front cover sheet, blank marking scheme, table of contents	1 1	**2**	
3	Narrative	Narrative description of business scenario		**1**	
4	SLC	Discuss and justify SLC & risk mgt strategy?		**1**	
5	Project Plan	Plan specifying timeline, deliverables, and roles.		**1**	
6	Requirement	• Listing of Func. Reqs. and use case diagram(s) • Structured use case descriptions(s) • Listing of NFRs and quality tactics • Screen shots / report formats	2 2 1 1	**6**	
7	System Architecture	System architecture diagram with interfaces		**2**	
8	Analysis Sketches	• Method used to identify candidate classes • Analysis Class diagram with generalisation, composition, multiplicity, dialog, control, entity, interfaces, pre and post conditions, etc. • Interaction diagram • Entity relationship diagram with cardinality	1 2 2 1	**6**	
10	Code	• Compiles and runs LoC • Key use cases implemented #Pkgs • MVC/other implemented #Classes • Design pattern(s) implemented • Git • Automated Testing	1 4 2 3 P/F P/F	**10**	
11	Added value			**5**	
12	Design and Architecture Recovery	• Architectural diagram based on implementation • Design time class diagram • State Chart	2 2 P/F	**4**	
14	Critique	Evaluate the analysis & design artefacts.		**1**	
15	References			**1**	
	Interview Week 13 (Pass/Fail basis)			**P/F**	
	Sub-total (A)			**40**	

PENALTIES					
	Description	**TM1**	**TM2**	**TM3**	**TM4**
1	Late Submission				
2	Failure to contribute to coding effort				
3	Failure to contribute to writing of report				
4	Failure to report problems with team dynamics				
5	Failure to contribute to demo week 13				
	Sub-total (B)				

FINAL MARKS AWARDED					
	(A-B)	**TM1**	**TM2**	**TM3**	**TM4**