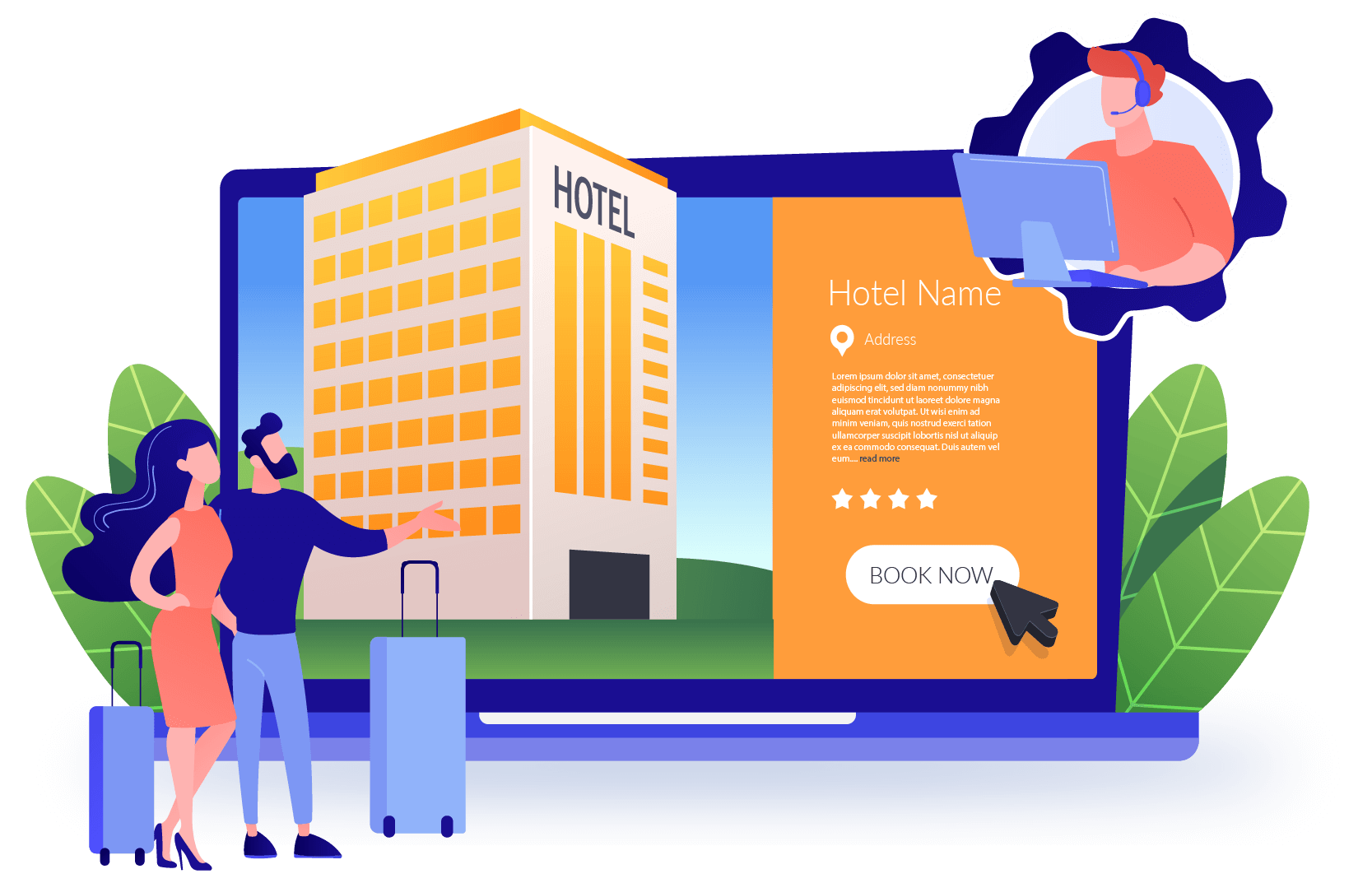
Hotel Management System

**LOW LEVEL DESIGN(LLD)**



Date: 19-04-2022

Current Document Version: 1.0

DOCUMENT APPROVAL

Approvers of this document

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Department** | **Role** | **Signature** | **Date** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Document change history

|  |  |  |  |
| --- | --- | --- | --- |
| **Document Version #** | **Author** | **Date** | **Description** |
| 1.0 | Guvvani Subhash Chandra | 19-04-2022 | Hotel Management System |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Table of Contents**

[DOCUMENT APPROVAL 2](#_Toc101285101)

[Approvers of this document 2](#_Toc101285102)

[Document change history 2](#_Toc101285103)

[Document Purpose 5](#_Toc101285104)

[Intended Audience 5](#_Toc101285105)

[Project Background, Objective 5](#_Toc101285106)

[Project Background 5](#_Toc101285107)

[Project Objective 5](#_Toc101285108)

[Design Pattern 5](#_Toc101285109)

[UML Diagram 6](#_Toc101285110)

[Solution Steps 9](#_Toc101285111)

[Manager 9](#_Toc101285112)

[Receptionist 10](#_Toc101285113)

[Owner 10](#_Toc101285114)

[Class/Functions 11](#_Toc101285115)

[Data Model 15](#_Toc101285116)

[API Canvas 18](#_Toc101285117)

[ENV Variables 21](#_Toc101285118)

[Integration 23](#_Toc101285119)

[HTTP Status Code 24](#_Toc101285120)

[Validations 24](#_Toc101285121)

[Unit testing 24](#_Toc101285122)

[Reservation Microservice 24](#_Toc101285123)

[Guest Microservice 25](#_Toc101285124)

[Bill Microservice 26](#_Toc101285125)

[Rooms Microservice 26](#_Toc101285126)

[Rates Microservice 27](#_Toc101285127)

[Staff Microservice 28](#_Toc101285128)

[Inventory Microservice 28](#_Toc101285129)

[Department microservice 29](#_Toc101285130)

[Request & Response 30](#_Toc101285131)

[Reservation Microservice 30](#_Toc101285132)

[Guest Microservice 31](#_Toc101285133)

[Bill Microservice 33](#_Toc101285134)

[Rooms Microservice 33](#_Toc101285135)

[Rates Microservice 34](#_Toc101285136)

[Staff Microservice 35](#_Toc101285137)

[Inventory Microservice 37](#_Toc101285138)

[Reports Microservice 38](#_Toc101285139)

[Department Microservice 39](#_Toc101285140)

Document Purpose

The purpose of the document is to provide the detailed information of all the microservices used to make a complete Hotel Management System application.

Intended Audience

This document is intended to provide an online application to the hotels, where hotel management staff can reserve and update all their hotel services.

Project Background, Objective

Project Background

Hotel Management system is an application where the end users are Owner, Manager and Receptionist. Owner can access all system functionalities without any restrictions. Managers can access all system functionalities with limited restrictions. Receptionists can only access the Reservation management section.

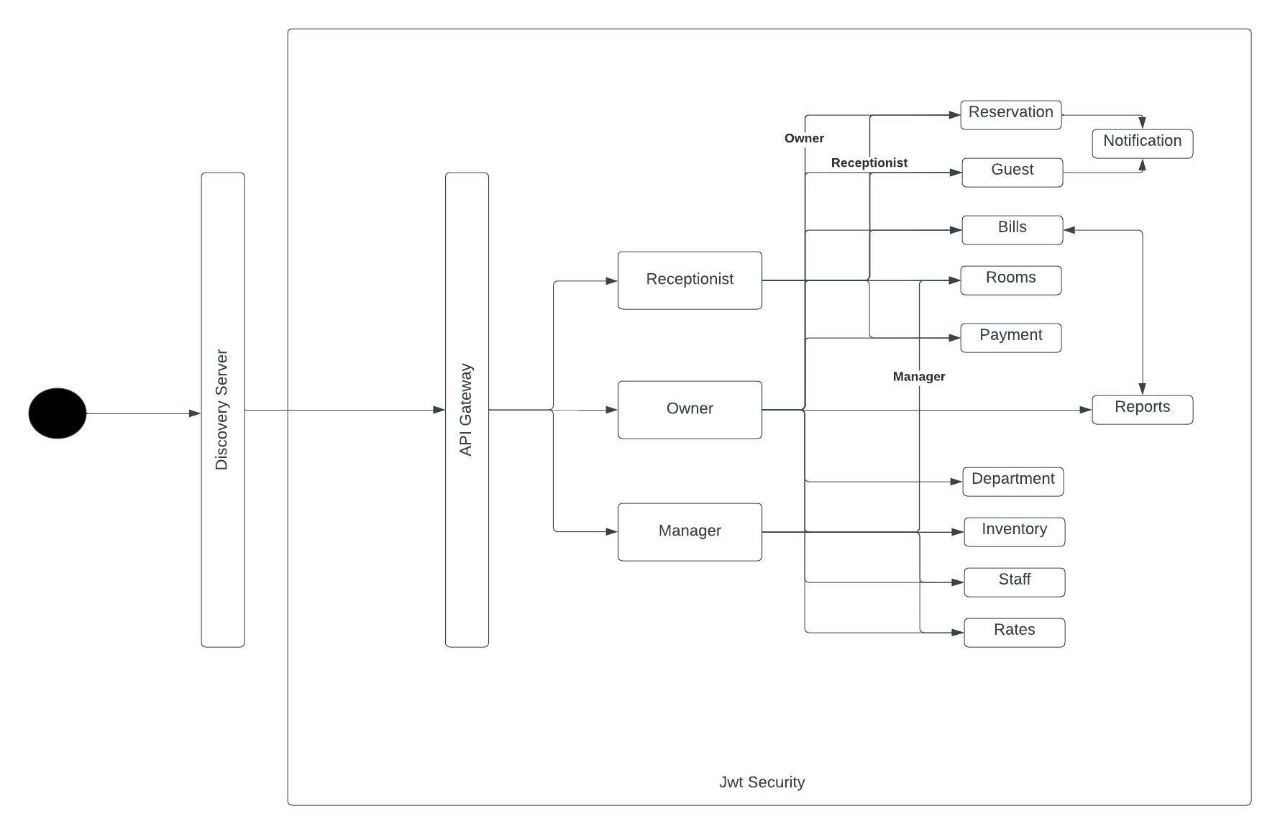
Project Objective

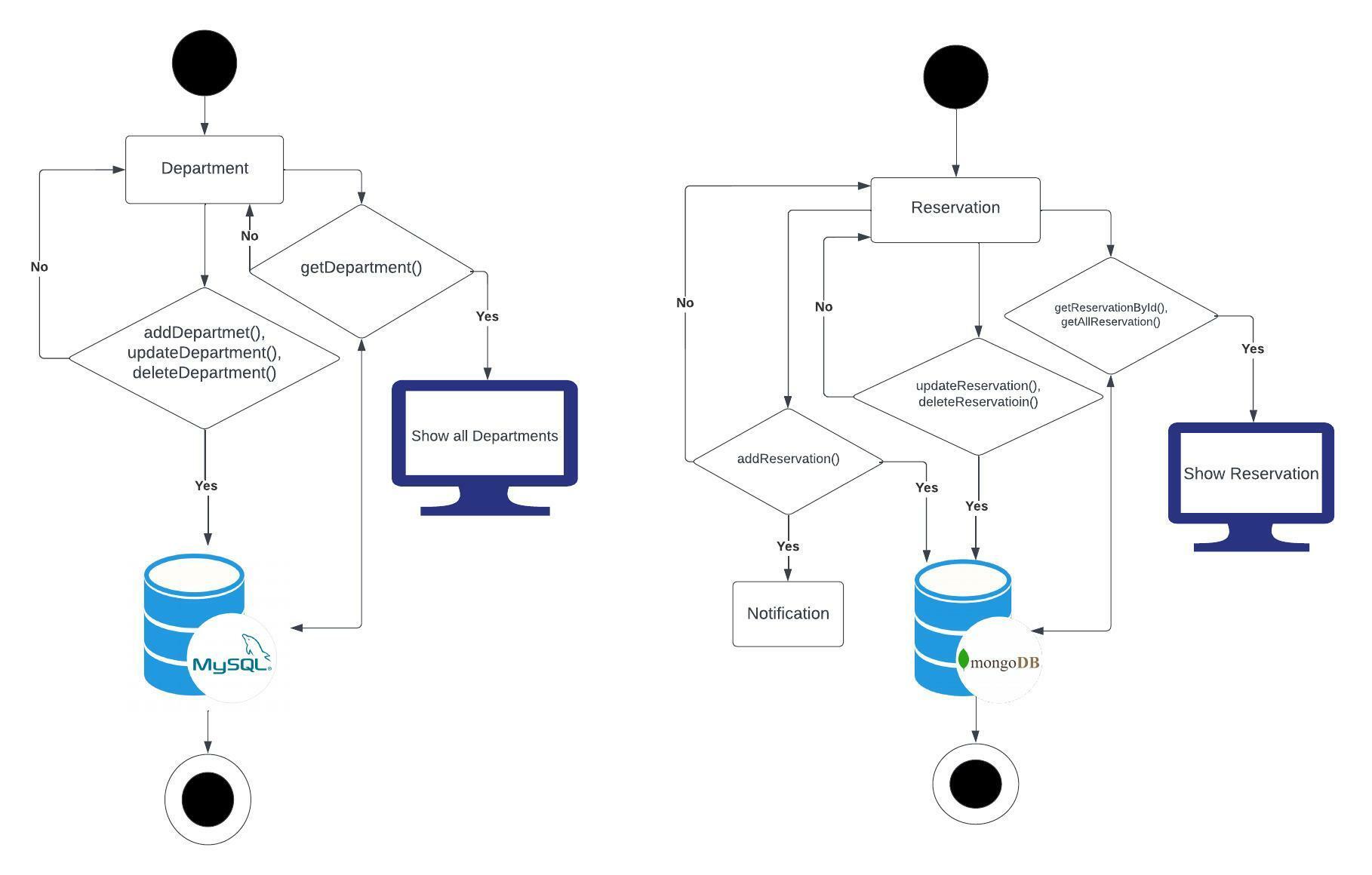
The objective of the Hotel Management System is to simplify the day-to-day processes of the hotel. The system will be able to handle many services to take care of all customers in a quick manner. As a solution to the large amount of file handling happening at the hotel, this software will be used to overcome those drawbacks.

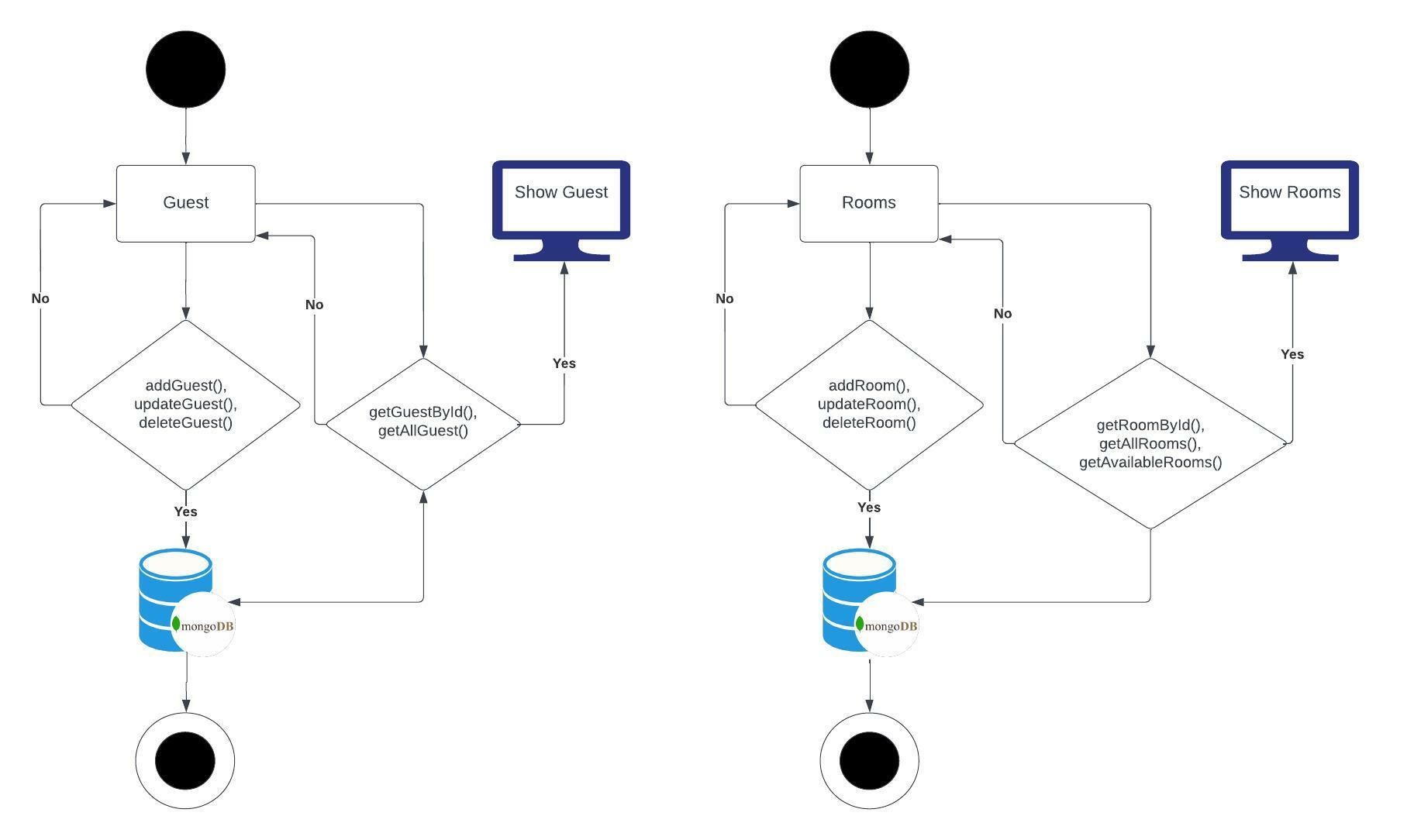
Design Pattern

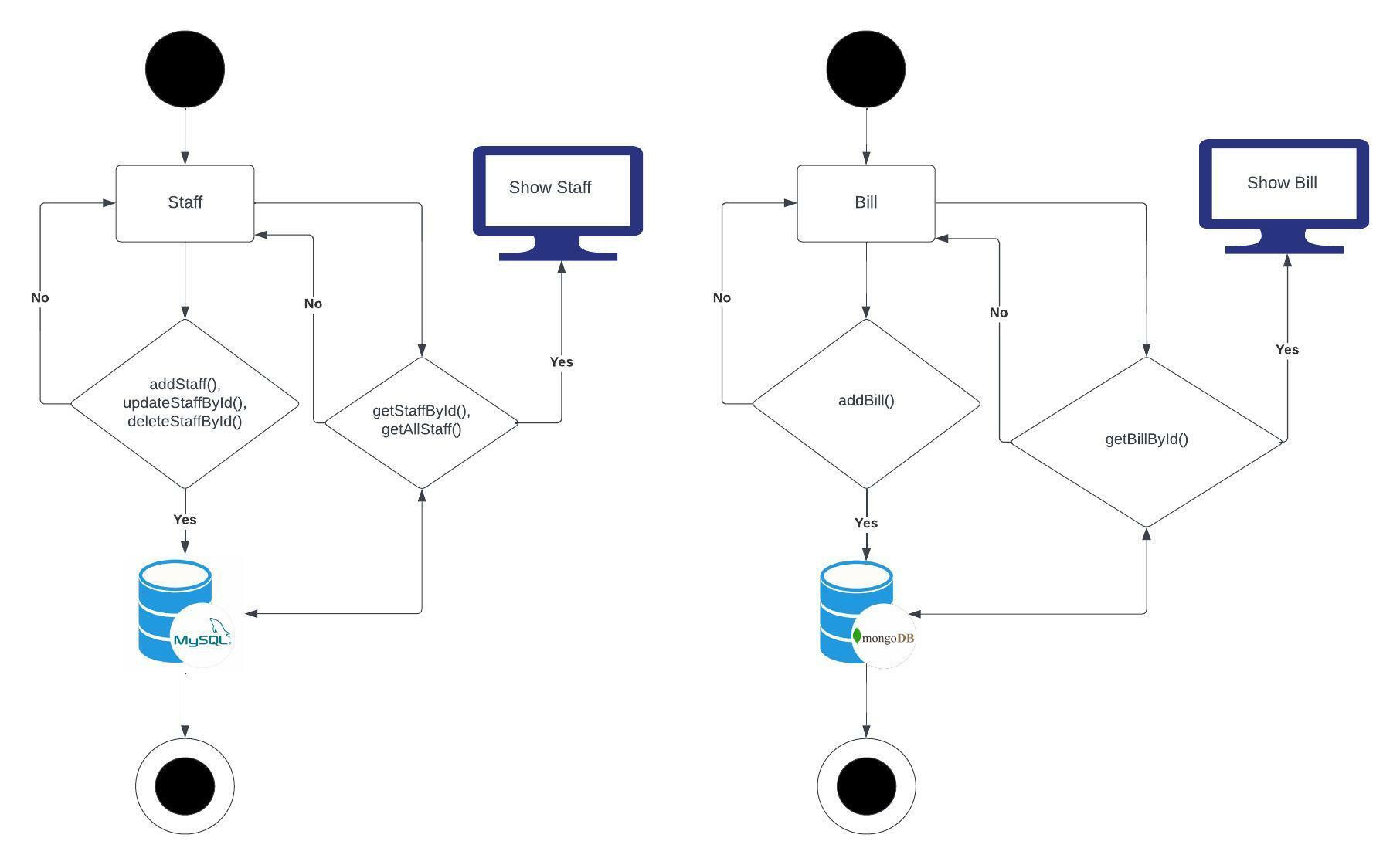
|  |  |  |
| --- | --- | --- |
| S/No. | Name | Description |
|  | Owner API | Using HTTP requests, users can access all the APIs in each microservices and can perform every functionality included in it. |
|  | Manager API | Using HTTP requests, users can access and perform functionalities of Inventory, Staff, Rates, Salary reports and Rooms API’s. |
|  | Receptionist API | Using HTTP requests, users can access and perform functionalities of Reservations, rooms, guest, payment and Bills API’s. |
|  | Reservations API | Using HTTP requests, end users can add, update, delete and view reservations. |
|  | Guest API | Using HTTP requests, end users can add, update, delete and view guest details in the hotel. |
|  | Bill API | Using HTTP requests, the receptionist and owner can Generate bill for customers using their id’s. |
|  | Rooms API | Using HTTP requests, the receptionist can view available rooms. The owner, manager can add, update and delete the rooms. |
|  | Rates API | Using HTTP requests, the owner and manager can add, update and view rates. |
|  | Staff API | Using HTTP requests, the owner and manager can add, update, delete and view Staff. |
|  | Inventory API | Using HTTP requests, the owner and manager can add, update, view and delete Inventory. |
|  | Department API | Using HTTP requests, the owner can add, update, delete and view department details. |
|  | Payment API | Using HTTP requests, the receptionist and owner can access the payment interface to pay the bill. |
|  | Reports API | Using HTTP request manager can generate the reports. |
|  | Notifications API | Using this API the reservation API sends the SMS and Email notification. |

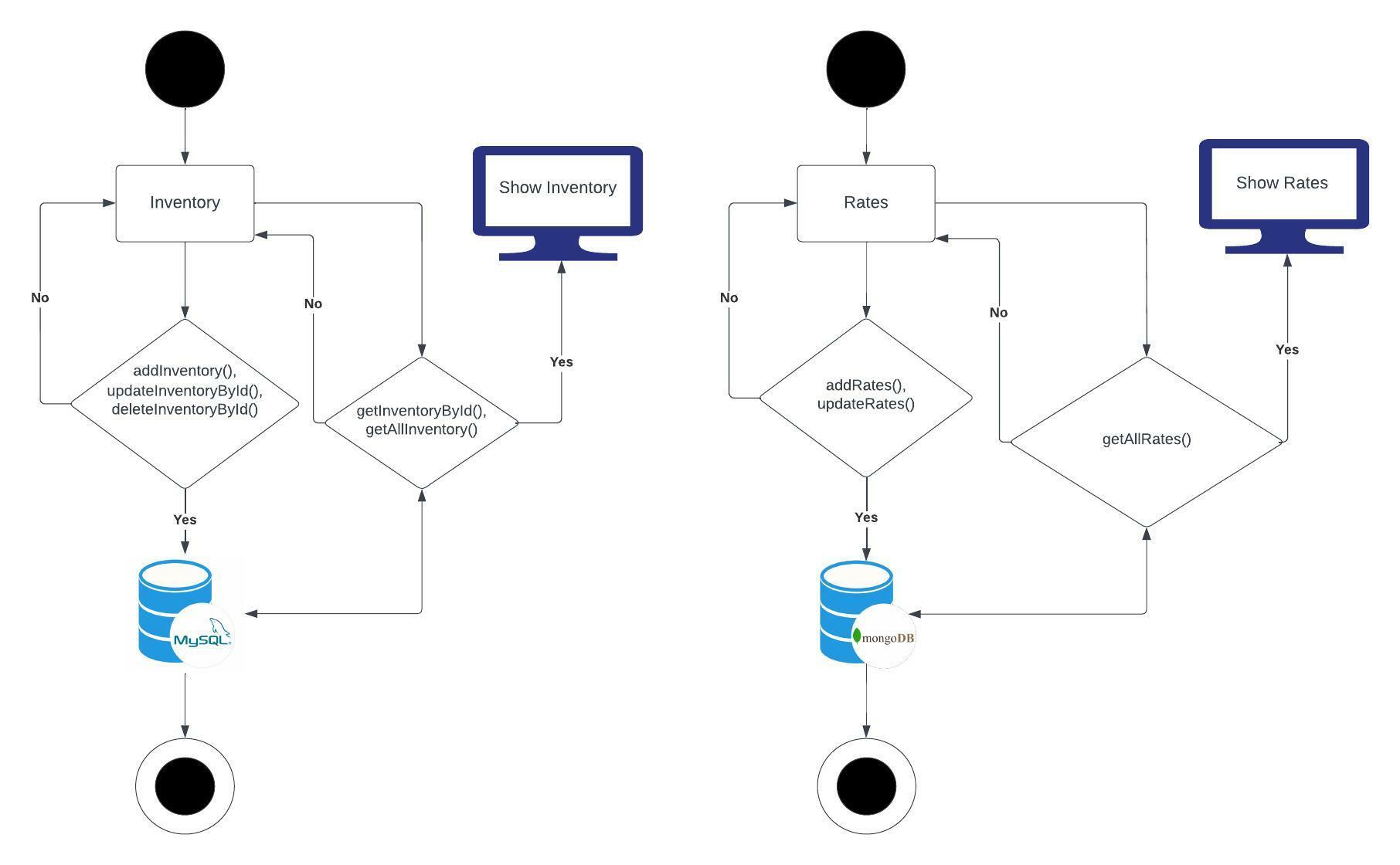
UML Diagram











Solution Steps

Manager

Staff API:

* Using the Staff API, managers can add, update and delete an employee.
* Using the department attribute in the staff API, the manager sorts the staff according to the department.
* Managers can add staff by providing the staffCode, employeeName, employeeAddress, salary, age, occupation, email.
* After entering all the details, the manager clicks on the submit button, which calls the post method and all the provided details will be saved in the staff database.
* Whenever a manager wants to update any staff details, the manager can provide respective staff id and make the required changes, on submit put method is executed and all the details will be updated.

Inventory API:

* All Inventory types were managed using add, update and delete requests inside the Inventory API.
* Managers can add inventory by providing the inventoryCode, inventoryType and inventoryQuantity.
* Using the GET request manager calls all the inventory details.
* Whenever the manager wants to update any inventory details, the manager can provide respective inventory id and make the required changes, on submit put method is executed and all the details will be updated.
* Using DELETE requests, managers can delete specific inventory types, by providing inventory id, on submit delete method is called and details will be deleted.

Rooms API:

* Manager can add rooms to the hotel using addNoofRooms() method.
* If the rooms were under maintenance, the manager can update the rooms as unavailable using updateTotalRooms() method.
* To know all the current existing rooms in a hotel, the manager can search using getAllRoomsData() method.

*Rates*:

* Manager uses setRates() method to add new prices by providing Id, firstNightPrice, dayPrice and nightPrice.
* Using updateRates() method, prices of firstNightPrice or dayPrice or nightPrice can be updated by giving specific rate id.
* To get all prices listed, the manager can access the getRates() method.

Manager API:

* Manager Authentication details are managed here using ManagerSecurityConfigurer(class)
* All the above-mentioned API’s which are managed by manager are called here.

Receptionist

Reservation API:

* Receptionist adds the reservation using the addReservations() method, when the customer visits the hotel for the room.
* Using getAllReservations() method the receptionist can get all the reservation information registered in the reservation repository

Guest API:

* Receptionist adds the guest information reserved using Reservation API to the repository and gives the certain room available in the hotel.
* Any information that needs to be updated after the guest reservation can be done using updateGuest() method.
* To get a list of guests information in the hotel, the receptionist uses the getGuestList() method.

Rooms API:

* To search the number of rooms available in the hotel for the guests, the receptionist uses the getAvailableRooms() method.

Bill API:

* Using room price and services received by the guest and the other taxes were combined and calculated to produce the total bill using setBillByGuestId() method.
* To check specific bills issued to the guest, the receptionist uses the getBiilByCode() method by providing the unique bill code.

Receptionist API:

* Receptionist Authentication details are managed here using ReceptionistSecurityConfigurer(class)
* All the above-mentioned API’s which are managed by Receptionist are called here.

Owner

Owner has access to all the API which are mentioned in the manager and receptionist API.

Department API:

* Owner can Create a new department in the hotel using createDepartment() method by providing the department details.
* Also owners can update and delete the departments using updateDepartment() and deleteDepartment() methods.

Staff API:

* All the functionalities mentioned in the manager API can be accessed by the owner
* Owner has extra functionality to manage the manager staff.

Class/Functions

|  |  |  |  |
| --- | --- | --- | --- |
| S/No. | Micro Services | Class | Description |
| 1. | Reservation | Main.java | It contains the main method of the reservation microservice and this class runs the microservice. |
| Reservation.java | It holds the Reservation schema details. |
| ReservationRepository.java | Mongo Repository is connected through this class for reservation microservice. |
| ReservationService.java | CRUD service methods of reservation microservice were included here. |
| ReservationController.java | HTTP request of Rest API methods (rest controller) were called here by using reservationService (class). |
| 2. | Notification | Main.java | It contains the main method of the notification microservice and this class runs the microservice. |
| NotificationController.java | HTTP request of Rest API methods (rest controller) were called here by using mailService (class) and smsService(class). |
| MailService.java | Mail subject, body and the attachment were pre-defined. |
| SmsService.java | Using twilio predefined messages were set here. |
| 3. | Guest | Main.java | It contains the main method of the guest microservice and this class runs the microservice. |
| Guest.java | It holds the guest schema details. |
| GuestRepository.java | MongoRepository is connected through this class for guest microservice. |
| GuestService.java | CRUD service methods of guest microservice were included here. |
| GuestController.java | HTTP request of Rest API methods (rest controller) were called here by using guestService (class). |
| 4. | Bill | Main.java | It contains the main method of the bill microservice and this class runs the microservice. |
| Bill.java | It holds the bill schema details. |
| BillRepository.java | MongoRepository is connected through this class for bill microservice. |
| BillService.java | CRUD service methods of bill microservice were included here. |
| BillController.java | HTTP request of Rest API methods (rest controller) were called here by using billService (class). |
| 5. | Inventory | Main.java | It contains the main method of the Inventory microservice and this class runs the microservice. |
| Inventory.java | It holds the Inventory schema details. |
| InventoryRepository.java | MySQL is connected through this class for Inventory microservice. |
| InventoryService.java | CRUD service methods of Inventory microservice were included here. |
| InventoryController.java | HTTP request of Rest API methods (rest controller) were called here by using InventoryService (class). |
| 6. | Rooms | Main.java | It contains the main method of the Rooms microservice and this class runs the microservice. |
| Rooms.java | It holds the room's schema details. |
| RoomsRepository.java | MongoRepository is connected through this class for Rooms microservice. |
| RoomsService.java | CRUD service methods of Rooms microservice were included here along with search rooms operations. |
| RoomsController.java | HTTP request of Rest API methods (rest controller) were called here by using roomsService (class). |
| 7. | Rates | Main.java | It contains the main method of the Rates microservice and this class runs the microservice. |
| Rates.java | It holds the rates schema details. |
| RatesRepository.java | MongoRepository is connected through this class for Rates microservice. |
| RatesService.java | CRUD service methods of rates microservice were included here. |
| RatesController.java | HTTP request of Rest API methods (rest controller) were called here by using ratesService (class). |
| 8. | Staff | Main.java | It contains the main method of the Staff microservice and this class runs the microservice. |
| Staff.java | It holds the staff schema details. |
| StaffRepository.java | MySQL is connected through this class for Staff microservice. |
| StaffService.java | CRUD service methods of Staff microservice were included here. |
| StaffController.java | HTTP request of Rest API methods (rest controller) were called here by using staffService (class). |
| 9. | Department | Main.java | It contains the main method of the Department microservice and this class runs the microservice. |
| Department.java | It holds the department schema details. |
| DepartmentRepository.java | This Class Extends MySQL from which the required methods are taken to perform the CRUD Operations. |
| DepartmentService.java | CRUD service methods are called from departmentRepository(class) to get required data from the database to perform functionalities. |
| DepartmentController.java | HTTP request of Rest API methods (rest controller) were called here by using departmentService (class). |
| 10. | Reports | Main.java | It contains the main method of the Reports microservice and this class runs the microservice. |
| Report.java | It holds the report schema details. |
| ReportRepository.java | This Class Extends MongoDB from which the required methods are taken to perform the CRUD Operations. |
| ReportService.java | CRUD service methods are called from ReportRepository(class) to get required data from the database to perform functionalities. |
| ReportController.java | HTTP request of Rest API methods (rest controller) were called here by using ReportService (class). |
| 11. | Payment | Main.java | It contains the main method of the Payment microservice and this class runs the microservice. |
| Payment.java | It holds the report schema details. |
| PaymentService.java | The payment gateway for bill payment methods were included here |
| PaymentController.java | HTTP request of Rest API methods (rest controller) were called here by using PaymentService (class). |
| 12. | OwnerEndUser | Main.java | It contains the main method of the OwnerEndUser microservice and this class runs the microservice. |
| OwnerSecurityConfigurer.java | It extends WebSecurityConfigurerAdapter to override the configure method for authentication and authorization for OwnerEndUser microservice. |
| OwnerRepository.java | This Class Extends MongoRepository from which the required methods are taken to perform the CRUD Operations. |
| OwnerInformation.java | It holds the owner information schema details. |
| Reservation.java, Guest.java, Bill.java, Inventory.java, Rooms.java, Staff.java, Department.java, Rates.java, reports, payments, notification | Model holders of microservices have access to the owner. |
| OwnerService.java | CRUD service methods are called from departmentRepository(class) to get required data from the database to perform functionalities. |
| OwnerController.java | HTTP request of Rest API methods (rest controller) were called here by using departmentService (class). |
| 13. | ManagerEndUser | Main.java | It contains the main method of the ManagerEndUser microservice and this class runs the microservice. |
| ManagerSecurityConfigurer.java | It extends WebSecurityConfigurerAdapter to override the configure method for authentication and authorization for ManagerEndUser microservice. |
| ManagerRepository.java | This Class Extends MongoRepository from which the required methods are taken to perform the CRUD Operations. |
| ManagerInformation.java | It holds the manager information schema details. |
| Inventory.java, Rooms.java, Staff.java, Rates.java, reports | Model holders of microservices have access to managers. |
| ManagerService.java | CRUD service methods are called from departmentRepository(class) to get required data from the database to perform functionalities. |
| ManagerController.java | HTTP request of Rest API methods (rest controller) were called here by using departmentService (class). |
| 14. | ReceptionistEndUser | Main.java | It contains the main method of the ReceptionistEndUser microservice and this class runs the microservice. |
| ReceptionistSecurityConfigurer.java | It extends WebSecurityConfigurerAdapter to override the configure method for authentication and authorization for ReceptionistEndUser microservice. |
| ReceptionistRepository.java | This Class Extends MongoRepository from which the required methods are taken to perform the CRUD Operations. |
| ReceptionistInformation.java | It holds the receptionist information schema details. |
| Reservation.java, Guest.java, Bill.java, Rooms.java, payments~~,~~ notification | Model holders of microservices have access to receptionists. |
| ReceptionistService.java | CRUD service methods are called from departmentRepository(class) to get required data from the database to perform functionalities. |
| ReceptionistController.java | HTTP request of Rest API methods (rest controller) were called here by using departmentService (class). |
| 15. | DiscoveryServer | Main.java | It contains the main method of the DiscoveryServer microservice and it will be annotated with @EnableEurekaServer to start the eureka server. |
| 16. | SwaggerAndAPIgateway | Main.java | It contains the main method of swagger where all the microservices API are connected and it routes between the API. |

Data Model

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S/No. | Microservice | Model | Attributes | Data type |
| 1. 1. | 1. Reservation | 1. reservation.java | reservationCode\_(Id) | STRING |
| numberOfAdult | INT |
| numberOfChildren | INT |
| checkIn | DATE |
| checkOut | DATE |
| numberOfNights | INT |
| name | STRING |
| emailId | STRING |
| phoneNumber | STRING |
| gender | STRING |
| address | STRING |
| company | STRING |
| status\_ | STRING |
| 2. | Guest | guest.java | guestCode\_(Id) | STRING |
| reservationCode | STRING |
| roomNumber | STRING |
| todayDate | DATE |
| name\_ | STRING |
| emailId\_ | STRING |
| phoneNumber\_ | STRING |
| gender\_ | STRING |
| address\_ | STRING |
| company\_ | STRING |
| guestStatus\_ | STRING |
| 3. | Rooms | room.java | roomNumber(Id) | STRING |
| roomStatus\_ | STRING |
| totalRooms\_ | LONG |
| 4. | Rates | rates.java | rateId(Id) | STRING |
| firstNightPrice | DOUBLE |
| nightPrice | DOUBLE |
| dayPrice | DOUBLE |
| 5. | Bill | Bill.java | billNumber\_(Id) | STRING |
| guestCode | STRING |
| quantity | DOUBLE |
| price | DOUBLE |
| taxes | DOUBLE |
| date | DATE |
| services | DOUBLE |
| unit | DOUBLE |
| 6. | Inventory | inventory.java | inventoryCode(PK) | VARCHAR (255) |
| inventoryType | VARCHAR (255) |
| inventoryQuantity | BigInt |
| 7. | Staff | staff.java | employeeId(PK) | VARCHAR (255) |
| departmentId | VARCHAR (255) |
| employeeName | VARCHAR (255) |
| phoneNumber | BigInt |
| emailId | VARCHAR (255) |
| age | INTEGER |
| employeeAddress | VARCHAR (255) |
| salary | BigDecimal |
| 8. | Reports | reports.java | reportId | STRING |
| employeeId | STRING |
| employeeSalary | DOUBLE |
| taxes | DOUBLE |
| date | DATE |
| 9. | Department | department.java | departmentId(PK) | VARCHAR (255) |
| departmentName | VARCHAR (255) |
| numberOfEmployees\_ | Int |
| 10. | Receptionist | receptionistInformation.java | receptionistId(Id) | STRING |
| emailId | STRING |
| password | STRING |
| 11. | Manager | managerInformation.java | managerId(Id) | STRING |
| emailId | STRING |
| password | STRING |
| 12. | Owner | ownerInformation.java | ownerId(Id) | STRING |
| emailId | STRING |
| password | STRING |

API Canvas

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| S/No. | Micro-services | Path | Verb | API Description | Role | Auth |
| 1. | Reservation | /(Role)/addreservation | POST | Add reservation to the database | Owner, Receptionist | False |
| /(Role)/getallreservation | GET | Retrieves list of reservations in the database | Owner, Receptionist | False |
| /(Role)/updatereservation | PUT | Updates the reservation | Owner, Receptionist | False |
| /(Role)/deletereservation | DELETE | Delete the reservation | Owner, Receptionist | False |
| /(Role)/getreservationbyid | GET | Retrieves a reservation by ID | Owner, Receptionist | False |
| 2. | Guest | /(Role)/addreserveguest | POST | Add guest to the database | Owner, Receptionist | False |
| /(Role)/getallguests | GET | Retrieves list of guests in the database | Owner, Receptionist | False |
| /(Role)/getguestbyid | GET | Retrieves a guest by ID | Owner, Receptionist | False |
| /(Role)/updateguest | PUT | Updates the guest details | Owner, Receptionist | False |
| /()/checkout | PUT |  |  |  |
| /(Role)/deleteguestbyid | DELETE | Delete the guest details | Owner, Receptionist | False |
| 3. | Bill | /(Role)/generatebillwithguestid | POST | Add bill to the database using guest id | Owner, Receptionist | False |
| /(Role)/getbillwithguestid | GET | Retrieves a bill by guest ID | Owner, Receptionist | False |
| 4. | Rooms | /(Role)/getavailablerooms | GET | Retrieves list of in-active rooms in the database | Owner, Receptionist | False |
| /(Role)/addrooms | POST | Add new rooms to the database | Owner, Manager | False |
| /(Role)/getallrooms | GET | Retrieves list of rooms in the database | Owner, Manager | False |
| /(Role)/updaterooms | PUT | Updates the room details | Owner, Manager | False |
| /(Role)/deleteroombyid | DELETE | Delete the room details | Owner, Manager | False |
| 5. | Rates | /(Role)/setrates | POST | set new rates to the database | Owner, Manager | False |
| /(Role)/getallrates | GET | Retrieves list of rates available in the database | Owner, Manager | False |
| /(Role)/updaterateswithid | PUT | Updates the rate details | Owner, Manager | False |
| 6. | Staff | /(Role)/addstaff | POST | Add staff to the database | Owner, Manager | False |
| /(Role)/getallstaff | GET | Retrieves list of staff members in the database | Owner, Manager | False |
| /(Role)/getstaffbyid | GET | Retrieves staff by id | Owner, Manager | False |
| /(Role)/updatestaff | PUT | Updates the staff details | Owner, Manager | False |
| /(Role)/deletestaff | DELETE | Delete the staff details | Owner, Manager | False |
| 7. | Inventory | /(Role)/addinventory | POST | Add Inventory to the database | Owner, Manager | False |
| /(Role)/getinventorybyid | GET | Retrieves inventory by id | Owner, Manager | False |
| /(Role)/getallinventory | GET | Retrieves list of inventory in the database | Owner, Manager | False |
| /(Role)/updateinventory | PUT | Updates the inventory details | Owner, Manager | False |
| /(Role)/deleteinventory | DELETE | Delete the inventory details | Owner, Manager | False |
| 8. | Reports | /(Role)/addreport | POST | By collection information from the database , it generates the report with a unique id | Owner, Manager | False |
| /(Role)/getreportbyid | GET | Retrieves the report using unique id | Owner, Manager | False |
| /(Role)/deletereport | DELETE | Deletes the report using unique id | Owner, Manager | False |
| 9. | Payment | /(Role)/makepayment | POST | Redirects to payment gateway to pay the hotel bill | Owner,  Receptionist | False |
| /(Role)/getpayment | GET | Retrieves all the previous payments | Owner,  Receptionist | False |
| 10. | Department | /owner/adddepartment | POST | Owner adds the new department to the database | Owner | False |
| /owner/getdepartment | GET | Gets all the department details from the database | Owner | False |
| /owner/updatedepartment | PUT | Updates the department details in the database | Owner | False |
| /owner/deletedepartment | DELETE | Deletes the department details in the database | Owner | False |
| 11. | Receptionist | /receptionist | POST | Log in to the Receptionist portal where other APIs are available for Receptionist | Receptionist | True |
| 12. | Manager | /manager | POST | Log in to the Manager portal where other APIs are available for Manager | Manager | True |
| 13. | Owner | /owner | POST | Log in to the Owner portal where other APIs are available for Owner | Owner | True |

ENV Variables

|  |  |  |  |
| --- | --- | --- | --- |
| S/No. | Micro Service | Configuration | Use case |
| 1. | Reservation | spring.data.mongodb.database | The name of the database to be created in the database is given here. |
| server.port | The 8081 port is provided and the localhost always runs on this port. |
| spring.data.mongodb.uri | To link provided here accesses the mongoDB cluster |
| spring.mvc.pathmatch.matching-strategy | ant-match-maker provided here makes the microservice available in the swaggerAPI |
| 2. | Guest | server.port | The 8082 port is provided and the localhost always runs on this port. |
| spring.data.mongodb.uri | To link provided here accesses the mongoDB cluster |
| spring.mvc.pathmatch.matching-strategy | ant-match-maker provided here makes the microservice available in the swaggerAPI |
| spring.data.mongodb.database | The name of the database to be created in the database is given here. |
| 3. | Bill | server.port | The 8083 port is provided and the localhost always runs on this port. |
| spring.data.mongodb.uri | To link provided here accesses the mongoDB cluster |
| spring.mvc.pathmatch.matching-strategy | ant-match-maker provided here makes the microservice available in the swaggerAPI |
| spring.data.mongodb.database | The name of the database to be created in the database is given here. |
| 4. | Room | server.port | The 8084 port is provided and the localhost always runs on this port. |
| spring.data.mongodb.uri | To link provided here accesses the mongoDB cluster |
| spring.mvc.pathmatch.matching-strategy | ant-match-maker provided here makes the microservice available in the swaggerAPI |
| spring.data.mongodb.database | The name of the database to be created in the database is given here. |
| 5. | Rates | server.port | The 8085 port is provided and the localhost always runs on this port. |
| spring.data.mongodb.uri | To link provided here accesses the mongoDB cluster |
| spring.mvc.pathmatch.matching-strategy | ant-match-maker provided here makes the microservice available in the swaggerAPI |
| spring.data.mongodb.database | The name of the database to be created in the database is given here. |
| 6. | Staff | server.port | The 8086 port is provided and the localhost always runs on this port. |
| spring.datasource.url | MySQL URI connection link is given here. |
| spring.datasource.username | MySQL username is given here to access the port. |
| spring.datasource.password | MySQL password is given here to access the port. |
| spring.jpa.hibernate.ddl-auto | It is given as update to implement the JPA methods |
| 7. | Inventory | server.port | The 8087 port is provided and the localhost always runs on this port. |
| spring.datasource.url | MySQL URI connection link is given here. |
| spring.datasource.username | MySQL username is given here to access the port. |
| spring.datasource.password | MySQL password is given here to access the port. |
| spring.jpa.hibernate.ddl-auto | It is given as update to implement the JPA methods |
| 8. | Reports | server.port | The 8090 port is provided and the localhost always runs on this port. |
| spring.data.mongodb.uri | To link provided here accesses the mongoDB cluster |
| spring.mvc.pathmatch.matching-strategy | ant-match-maker provided here makes the microservice available in the swaggerAPI |
| spring.data.mongodb.database | The name of the database to be created in the database is given here. |
| 9. | Department | server.port | The 8088 port is provided and the localhost always runs on this port. |
| spring.datasource.url | MySQL URI connection link is given here. |
| spring.datasource.username | MySQL username is given here to access the port. |
| spring.datasource.password | MySQL password is given here to access the port. |
| spring.jpa.hibernate.ddl-auto | It is given as update to implement the JPA methods |

Integration

* Swagger

It is used to connect all the API in a common swagger webpage for the documentation.

* Twilio

It is a third-party API used to send the SMS to the client.

* Eureka

It is a client-side service discovery used to register and manage the instances of the microservice API and also used for load balancing.

HTTP Status Code

* 201 – Customer Registered
* 200 - Request succeeded
* 400 – Inputs are invalid
* 404 – Customer Not found
* 502 – Bad gateway

Validations



Unit testing

Reservation Microservice

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Case ID | Test Case Scenario | Test Case | Pre-Condition | Test Steps | Test Data | Expected Results | Actual Result |
| RTC01 | Mocking (get info from reservation database) | End user need to enter all the required fields | Enter the size of the mocked repository | 1.Mocking the data to be called when the findAll() method is called.  2. compare the expected size with the actual size of mock data using assert equals. | <expected size is equal to the actual mock data size> | Size of the mock data | Size of the mock data using getAllReservation().size() method. |
| RTC02 | Mocking (Data to be inserted into reservation database) | End user need to enter all the required fields | Enter the inserted mock data | 1. Mocking the data to be called when addReservation() method is called.  2. compare the expected reservation details with actual details. | <expected data is equal to actual mock data> | Mocked reservation details | Mocked reservation details using addReservation() method from service class. |
| RTC03 | Mocking (Data to be updated into reservation database) | End user need to enter all the updated fields along with the required fields | Enter the updated mock data | 1. Mocking the data to be called when updateReservation() method is called.  2. compare the expected reservation details with actual details. | <expected data is equal to actual mock data> | Mocked reservation details | Mocked reservation details using updateReservation() method from service class. |

Guest Microservice

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Case ID | Test Case Scenario | Test Case | Pre-Condition | Test Steps | Test Data | Expected Results | Actual Result |
| GTC01 | Mocking (get info from guest database) | End user need to enter all the required fields | Enter the size of the mocked repository | 1.Mocking the data to be called when the findAll() method is called.  2. compare the expected size with the actual size of mock data using assert equals. | <expected size is equal to the actual mock data size> | Size of the mock data | Size of the mock data using getAllGuest().size() method. |
| GTC02 | Mocking (Data to be inserted into guest database) | End user need to enter all the required fields | Enter the inserted mock data | 1. Mocking the data to be called when addGuest() method is called.  2. compare the expected guest details with actual details. | <expected data is equal to actual mock data> | Mocked guest details | Mocked guest details using addGuest() method from service class. |
| GTC03 | Mocking (Data to be updated into guest database) | End user need to enter all the updated fields along with the required fields | Enter the updated mock data | 1. Mocking the data to be called when updateGuest() method is called.  2. compare the expected guest details with actual details. | <expected data is equal to actual mock data> | Mocked guest details | Mocked guest details using updateGuest() method from service class. |

Bill Microservice

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Case ID | Test Case Scenario | Test Case | Pre-Condition | Test Steps | Test Data | Expected Results | Actual Result |
| BTC01 | Mocking (get info from bill database) | End user need to enter bill ID | Enter the mocked bill data | 1.Mocking the data to be called when the findAll() method is called.  2. compare the expected bill data with the actual bill data of mock data using assert equals. | <expected bill data is equal to the actual bill mock data > | Bill details of the mock data | Bill details of the mock data |
| BTC02 | Mocking (Data to be inserted into bill database) | End user need to enter all the required fields | Enter the inserted mock data | 1. Mocking the data to be called when addBill() method is called.  2. compare the expected bill details with actual details. | <expected data is equal to actual mock data> | Mocked bill details | Mocked bill details using addBill() method from service class. |

Rooms Microservice

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Case ID | Test Case Scenario | Test Case | Pre-Condition | Test Steps | Test Data | Expected Results | Actual Result |
| RMTC01 | Mocking (get info from rooms database) | End user need to enter all the required fields | Enter the size of the mocked repository | 1.Mocking the data to be called when the findAll() method is called.  2. compare the expected size with the actual size of mock data using assert equals. | <expected size is equal to the actual mock data size> | Size of the mock data | Size of the mock data |
| RMTC02 | Mocking (Data to be inserted into rooms database) | End user need to enter all the required fields | Enter the inserted mock data | 1. Mocking the data to be called when addRoom() method is called.  2. compare the expected room details with actual details. | <expected data is equal to actual mock data> | Mocked room details | Mocked room details using addRoom() method from service class. |
| RMTC03 | Mocking (Data to be updated into rooms database) | End user need to enter all the updated fields along with the required fields | Enter the updated mock data | 1. Mocking the data to be called when updateRoom() method is called.  2. compare the expected room details with actual details. | <expected data is equal to actual mock data> | Mocked room details | Mocked room details using updateRoom() method from service class. |

Rates Microservice

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Case ID | Test Case Scenario | Test Case | Pre-Condition | Test Steps | Test Data | Expected Results | Actual Result |
| RTTC01 | Mocking (get info from rates database) | End user need to enter all the required fields | Enter the size of the mocked repository | 1.Mocking the data to be called when the findAll() method is called.  2. compare the expected size with the actual size of mock data using assert equals. | <expected size is equal to the actual mock data size> | Size of the mock data | Size of the mock data using getAllRates().size() method. |
| RTTC02 | Mocking (Data to be inserted into rates database) | End user need to enter all the required fields | Enter the inserted mock data | 1. Mocking the data to be called when addRate() method is called.  2. compare the expected rate details with actual details. | <expected data is equal to actual mock data> | Mocked rate details | Mocked rate details using addRate() method from service class. |
| RTTC03 | Mocking (Data to be updated into rates database) | End user need to enter all the updated fields along with the required fields | Enter the updated mock data | 1. Mocking the data to be called when updateRate() method is called.  2. compare the expected rate details with actual details. | <expected data is equal to actual mock data> | Mocked rate details | Mocked rate details using updateRate() method from service class. |

Staff Microservice

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Case ID | Test Case Scenario | Test Case | Pre-Condition | Test Steps | Test Data | Expected Results | Actual Result |
| STC01 | Mocking (get info from staff database) | End user need to enter staff ID | Enter the mocked staff data | 1.Mocking the data to be called when the findAll() method is called.  2. compare the expected staff data with the actual staff data of mock data using assert equals. | <expected staff data is equal to the actual staff mock data > | Staff details of the mock data | Staff details of the mock data using getStaffById() method. |
| STC02 | Mocking (Data to be inserted into staff database) | End user need to enter all the required fields | Enter the inserted mock data | 1. Mocking the data to be called when addStaff() method is called.  2. compare the expected staff details with actual details. | <expected data is equal to actual mock data> | Mocked staff details | Mocked staff details using addStaff() method from service class. |
| STC03 | Mocking (Data to be updated into staff database) | End user need to enter all the updated fields along with the required fields | Enter the updated mock data | 1. Mocking the data to be called when updateStaff() method is called.  2. compare the expected staff details with actual details. | <expected data is equal to actual mock data> | Mocked staff details | Mocked staff details using updateStaff() method from service class. |

Inventory Microservice

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Case ID | Test Case Scenario | Test Case | Pre-Condition | Test Steps | Test Data | Expected Results | Actual Result |
| ITC01 | Mocking (get info from inventory database) | End user need to enter all the required fields | Enter the size of the mocked repository | 1.Mocking the data to be called when the findAll() method is called.  2. compare the expected size with the actual size of mock data using assert equals. | <expected size is equal to the actual mock data size> | Size of the mock data | Size of the mock data using getAllInventory().size() method. |
| ITC02 | Mocking (Data to be inserted into inventory database) | End user need to enter all the required fields | Enter the inserted mock data | 1. Mocking the data to be called when addInventory() method is called.  2. compare the expected inventory details with actual details. | <expected data is equal to actual mock data> | Mocked Inventory details | Mocked inventory details using addInventory() method from service class. |
| ITC03 | Mocking (Data to be updated into inventory database) | End user need to enter all the updated fields along with the required fields | Enter the updated mock data | 1. Mocking the data to be called when updateInvnetory() method is called.  2. compare the expected inventory details with actual details. | <expected data is equal to actual mock data> | Mocked inventory details | Mocked inventory details using updateInventory() method from service class. |

Department microservice

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Case ID | Test Case Scenario | Test Case | Pre-Condition | Test Steps | Test Data | Expected Results | Actual Result |
| DTC01 | Mocking (get info from department database) | End user need to enter all the required fields | Enter the size of the mocked repository | 1.Mocking the data to be called when the findAll() method is called.  2. compare the expected size with the actual size of mock data using assert equals. | <expected size is equal to the actual mock data size> | Size of the mock data | Size of the mock data using getAllDepartment().size() method. |
| DTC02 | Mocking (Data to be inserted into department database) | End user need to enter all the required fields | Enter the inserted mock data | 1. Mocking the data to be called when addDepartment() method is called.  2. compare the expected department details with actual details. | <expected data is equal to actual mock data> | Mocked department details | Mocked department details using addDepartment() method from service class. |
| DTC03 | Mocking (Data to be updated into department database) | End user need to enter all the updated fields along with the required fields | Enter the updated mock data | 1. Mocking the data to be called when updateDepartment() method is called.  2. compare the expected department details with actual details. | <expected data is equal to actual mock data> | Mocked department details | Mocked department details using updateDepartment() method from service class. |

Request & Response

Reservation Microservice

Reservation:

|  |  |
| --- | --- |
| Request | Response |
| {"checkin": "2022/04/16", "checkout": "2022/04/19", "ReservationCode": "RS02", "noofadults": 3, "noofchildren": 3, "noofnights": 2, "status\_": "string"} | **If valid details**  {“message": "Reservation successfully"}  status code: 201  **If invalid details**  {“message”: “Inputs are not valid”}  status code: 400  **If server encountered unexpected error**  {“message”:” Internal server error”}  status code: 500 |

Update reservation:

|  |  |
| --- | --- |
| Request | Response |
| {"checkin":"2022/04/18", "checkout":"2022/04/22", "ReservationCode":"RS02", "noofadults": 2, "noofchildren": 5, "noofnights": 5, "status\_": "string"} | **If valid details**  {“message": "Reservation updated successfully"}  status code: 201  **If invalid details**  {“message”: “Inputs are not valid”}  status code: 400  **If server encountered unexpected error**  {“message”:” Internal server error”}  status code: 500 |

Get Reservation by ID:

|  |  |
| --- | --- |
| Request | Response |
| {  "ReservationCode": "RS02",  } | **If valid ID**  {“checkin": "2022/04/18", "checkout":"2022/04/22", "ReservationCode":"RS02", "noofadults": 2, "noofchildren": 5, "noofnights": 5, "Status\_":"string"}  status code: 200  **If invalid ID**  {“message”: “Customer not found”}  status code:404  **If server encountered unexpected error**  {“message”:” Internal server error”}  status code: 500 |

Guest Microservice

Add guest:

|  |  |
| --- | --- |
| Request | Response |
| {"address": "Rajendra nagar 4th line", "company": "Capgemini",  "email": "rajesh09@gmail.com",  "gender": "M",  "guestStatus\_": "string", "guestCode": "G01", "name": "Rajesh", "phoneNumber": 9876543217, "reservevationCode\_": "RS02", "roomNo\_":"RM01", "todayDate\_":"2022/04/16"  } | **If valid details**  {“message": "Guest added successfully"}  status code: 201  **If invalid details**  {“message”: “Inputs are not valid”}  status code: 400  **If server encountered unexpected error**  {“message”:” Internal server error”}  status code: 500 |

Update guest by ID:

|  |  |
| --- | --- |
| Request | Response |
| {“address": "Rajendra nagar 3rd line", "company": "Capgemini", "email":"rajesh09@gmail.com",  "gender": "M", "guestStatus\_": "string", "guestCode": "G01", "name": "Rohith", "phoneNumber": 98765765417, "reservevationCode\_": "RS01", "roomNo\_":"RM02", "todayDate\_":"2022/04/18"  } | **If valid details**  {“message": "Guest updated successfully"}  status code: 201  **If invalid details**  {“message”: “Inputs are not valid”}  status code: 400  **If server encountered unexpected error**  {“message”:” Internal server error”}  status code: 500 |

Get guest by ID:

|  |  |
| --- | --- |
| Request | Response |
| {  "guestCode": "G01",  } | **If valid ID**  {"address": "Rajendra nagar 3rd line", "company": "Capgemini", "email": "rajesh09@gmail.com", "gender": "M", "guestStatus\_": "string", "guestCode": "G01", "name": "Rohith", "phoneNumber": 98765765417, "reservevationCode\_": "RS01", "roomNo\_":"RM02", "todayDate\_": "2022/04/18"}  status code: 200  **If invalid ID**  {“message”: “Customer not found”}  status code:404  **If server encountered unexpected error**  {“message”:” Internal server error”}  status code: 500 |

Bill Microservice

Generate Bill by guest ID:

|  |  |
| --- | --- |
| Request | Response |
| {  "billingNo": "B01",  "date": "2022/04/16",  "guestCode\_": "G01",  "price\_": 0,  "quantity": 56,  "services": 89,  "taxes\_": 0,  "unit": 5  } | **If valid details**  {"billingNo": "B01","guestMemberCode\_": "G01","quantity": 20,"price\_": 7000,"taxes\_": 700,"date": "2022-04-14","services": 24,"unit": 2}  status code: 201  **If invalid details**  {“message”: “Inputs are not valid”}  status code: 400  **If server encountered unexpected error**  {“message”:” Internal server error”}  status code: 500 |

Get bill with bill ID:

|  |  |
| --- | --- |
| Request | Response |
| {  "billingNo": "B01",  } | **If valid details**  {"billingNo": "B01","guestMemberCode\_": "G01","quantity": 20,"price\_": 7000,"taxes\_": 700,"date": "2022-04-14","services": 24,"unit": 2}  status code: 201  **If invalid details**  {“message”: “Inputs are not valid”}  status code: 400  **If server encountered unexpected error**  {“message”:” Internal server error”}  status code: 500 |

Rooms Microservice

Add rooms:

|  |  |
| --- | --- |
| Request | Response |
| {  "roomNo": "RM01",  "roomStatus\_": "string",  "totalRooms": 10  } | **If valid details**  {“message": "Room added successfully"}  status code: 201  **If invalid details**  {“message”: “Inputs are not valid”}  status code: 400  **If server encountered unexpected error**  {“message”:” Internal server error”}  status code: 500 |

Update rooms:

|  |  |
| --- | --- |
| Request | Response |
| {  "roomNo": "RM01",  "roomStatus\_": "string",  "totalRooms": 20  } | **If valid details**  {“message": "Room updated successfully"}  status code: 201  **If invalid details**  {“message”: “Inputs are not valid”}  status code: 400  **If server encountered unexpected error**  {“message”:” Internal server error”}  status code: 500 |

Rates Microservice

Set rates:

|  |  |
| --- | --- |
| Request | Response |
| {  "rateId": "Rate01",  "firstNightPrice": 3000,  "nightprice": 2500,  "dayprice": 2000  } | **If valid details**  {“message": "Rates set successfully"}  status code: 201  **If invalid details**  {“message”: “Inputs are not valid”}  status code: 400  **If server encountered unexpected error**  {“message”:” Internal server error”}  status code: 500 |

Update rates with rate ID:

|  |  |
| --- | --- |
| Request | Response |
| {  "rateId": "Rate01",  "firstNightPrice": 4000,  "nightprice": 3000,  "dayprice": 2500  } | **If valid details**  {“message": "Rates updated successfully"}  status code: 201  **If invalid details**  {“message”: “Inputs are not valid”}  status code: 400  **If server encountered unexpected error**  {“message”:” Internal server error”}  status code: 500 |

Staff Microservice

Add staff:

|  |  |
| --- | --- |
| Request | Response |
| {  "age": 20,  "email": "rajesh07@gmail.com",  "phoneNumber":  "employeeAddress": "Rajendra nagar 4th line",  "employeeName": "Nishant",  "employeeId": "SF01",  "departmentId": "Receptionist",  "salary": 70000,  } | **If valid details**  {“message": "Staff added successfully"}  status code: 201  **If invalid details**  {“message”: “Inputs are not valid”}  status code: 400  **If server encountered unexpected error**  {“message”:” Internal server error”}  status code: 500 |

Update staff:

|  |  |
| --- | --- |
| Request | Response |
| {  "age": 20,  "email": "rajesh07@gmail.com",  "phoneNumber":  "employeeAddress": "Rajendra nagar 4th line",  "employeeName": "Nishant",  "employeeId": "SF01",  "departmentId": "Receptionist",  "salary": 70000,  } | **If valid details**  {“message": "Staff added successfully"}  status code: 201  **If invalid details**  {“message”: “Inputs are not valid”}  status code: 400  **If server encountered unexpected error**  {“message”:” Internal server error”}  status code: 500 |

Get staff by staff ID:

|  |  |
| --- | --- |
| Request | Response |
| {  "employeeId": "SF01",  } | **If valid details**  {"age": 20,"email": "rajesh07@gmail.com","phoneNumber":"employeeAddress": "Rajendra nagar 4th line","employeeName": "Nishant", "employeeId": "SF01", "departmentId": "Receptionist","salary": 70000, }  status code: 201  **If invalid details**  {“message”: “Inputs are not valid”}  status code: 400  **If server encountered unexpected error**  {“message”:” Internal server error”}  status code: 500 |

Delete staff by staff ID:

|  |  |
| --- | --- |
| Request | Response |
| {  "employeeId": "SF01",  } | **If valid details**  {“message": "Staff member deleted successfully"}  status code: 201  **If invalid details**  {“message”: “Inputs are not valid”}  status code: 400  **If server encountered unexpected error**  {“message”:” Internal server error”}  status code: 500 |

Inventory Microservice

Add inventory:

|  |  |
| --- | --- |
| Request | Response |
| {  "InventoryId": "I01",  "InventoryType": "Kitchen",  "quantity": "50",  } | **If valid details**  {“message": "Inventory added successfully"}  status code: 201  **If invalid details**  {“message”: “Inputs are not valid”}  status code: 400  **If server encountered unexpected error**  {“message”:” Internal server error”}  status code: 500 |

Update inventory by ID:

|  |  |
| --- | --- |
| Request | Response |
| {  "InventoryId": "I01",  "InventoryType": "Kitchen",  "quantity": "165",  } | **If valid details**  {“message": "Inventory updated successfully"}  status code: 201  **If invalid details**  {“message”: “Inputs are not valid”}  status code: 400  **If server encountered unexpected error**  {“message”:” Internal server error”}  status code: 500 |

Delete inventory by ID:

|  |  |
| --- | --- |
| Request | Response |
| {  "InventoryId": "I01",  } | **If valid details**  {“message": "Inventory deleted successfully"}  status code: 201  **If invalid details**  {“message”: “Inputs are not valid”}  status code: 400  **If server encountered unexpected error**  {“message”:” Internal server error”}  status code: 500 |

Reports Microservice

Add report:

|  |  |
| --- | --- |
| Request | Response |
| {  "reportId": "R01",  "employeeId": "S01",  "departmentId": "D01",  "employeeSalary": "70000",  "taxes": "500",  "date": "2022/04/16"  } | **If valid details**  {“message": "Report added successfully"}  status code: 201  **If invalid details**  {“message”: “Inputs are not valid”}  status code: 400  **If server encountered unexpected error**  {“message”:” Internal server error”}  status code: 500 |

Delete report by ID:

|  |  |
| --- | --- |
| Request | Response |
| {  "reportId": "R01",  } | **If valid details**  {“message": "Report deleted successfully"}  status code: 201  **If invalid details**  {“message”: “Inputs are not valid”}  status code: 400  **If server encountered unexpected error**  {“message”:” Internal server error”}  status code: 500 |

Department Microservice

Add department:

|  |  |
| --- | --- |
| Request | Response |
| {  "’departmentId": "D01",  "’departmentname": "kitchen",  "numberOfEmployees": "10",  } | **If valid details**  {“message": "Department added successfully"}  status code: 201  **If invalid details**  {“message”: “Inputs are not valid”}  status code: 400  **If server encountered unexpected error**  {“message”:” Internal server error”}  status code: 500 |

Update department by ID:

|  |  |
| --- | --- |
| Request | Response |
| {  "’departmentId": "D01",  "’departmentname": "kitchen",  "numberOfEmployees": "50",  } | **If valid details**  {“message": "Department updated successfully"}  status code: 201  **If invalid details**  {“message”: “Inputs are not valid”}  status code: 400  **If server encountered unexpected error**  {“message”:” Internal server error”}  status code: 500 |

Delete department by ID:

|  |  |
| --- | --- |
| Request | Response |
| {  "’departmentId": "D01",  } | **If valid details**  {“message": "Department deleted successfully"}  status code: 201  **If invalid details**  {“message”: “Inputs are not valid”}  status code: 400  **If server encountered unexpected error**  {“message”:” Internal server error”}  status code: 500 |