Contents

[1. MVP Hotel Booking 1](#_Toc157618394)

[1.1 Overview 1](#_Toc157618395)

[2. Explore REST APIs 1](#_Toc157618396)

[2.1 Hotel 1](#_Toc157618397)

[2.1.1 Create hotel 1](#_Toc157618398)

[2.1.2 Update existing hotel 2](#_Toc157618399)

[2.1.3 Get all existing hotels 2](#_Toc157618400)

[2.1.4 Get all available hotels 2](#_Toc157618401)

[2.1.5 Get a user specified hotel 2](#_Toc157618402)

[2.1.6 Delete hotel 3](#_Toc157618403)

[2.2 Booking 3](#_Toc157618404)

[2.2.1 Create Booking 3](#_Toc157618405)

[2.2.2 Get all Bookings 3](#_Toc157618406)

[2.2.3 Get Booking 3](#_Toc157618407)

[2.2.4 Delete Booking 4](#_Toc157618408)

[2.3 Database 4](#_Toc157618409)

[2.3.1 Hotel 4](#_Toc157618410)

[2.3.2 Reservation 4](#_Toc157618411)

[2.4 Testing 4](#_Toc157618412)

[2.5 Tech Stack 5](#_Toc157618413)

# MVP Hotel Booking

## 1.1 Overview

* This project is a Hotel Booking RESTful API designed to create hotels and associated bookings.
* The user can create a hotel, search for available hotels, search for all hotels within the database, search all bookings within the database, make bookings to a specific hotel, delete bookings as well as hotels.
* Necessary validation is incorporated within the API that prevents illogical operations from occurring such as making an overlapping Booking to a hotel or deleting a hotel that contains bookings.
* The Hotel-Booking-API follows a standard "User <-> Controller <-> Validator (Only interacts with Controller) <-> Service <-> Repository <-> Database" API schema.

# 2. Explore REST APIs

## 2.1 Hotel

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Endpoint | Description | Valid API Calls |
| POST | /api/v1/hotel | Create a hotel object | Create hotel |
| PATCH | /api/v1/hotel | Updates an existing hotel | Update existing hotel |
| GET | /api/v1/hotels | Get all existing hotels | Get all existing hotels |
| GET | /api/v1/hotels/availabilitySearch?dateFrom={from}&dateTo={to} | Get all available hotels between specified dates. | Get all available hotels |
| GET | /api/v1/hotel/{id} | Get a user specified hotel | Get a user specified hotel |
| DELETE | /api/v1/hotel/{id} | Delete a user specified Hotel | Delete hotel |

### 2.1.1 Create hotel

* Description: Create a hotel object.
* Validation (Throws an InvalidRequestException when):
  + Type is not DELUXE, LUXURY, or SUITE
  + Date is not YYYY-MM-DD
  + Only one date is recieved
  + availableTo date comes before availableFrom date
  + Name is null
* Request Body:

{

"name" : "String",

"type" : "String: (LUXURY, DELUXE, SUITE)",

"description" : "String",

"availableFrom" : "YYYY-MM-DD",

"availableTo" : "YYYY-MM-DD",

"status": true

}

### 2.1.2 Update existing hotel

* Description: Updates an existing hotel object.
* Validation (Throws an InvalidRequestException when):
  + ID does not exist
  + Type is not DELUXE, LUXURY, or SUITE
  + Date is not YYYY-MM-DD
  + Only one date is recieved
  + availableTo date comes before availableFrom date
  + Name is null
* Request Body:

{

"id" : 0,

"name" : "String",

"type" : "String: (LUXURY, DELUXE, SUITE)",

"description" : "String",

"availableFrom" : "YYYY-MM-DD",

"availableTo" : "YYYY-MM-DD",

"status": true

}

### 2.1.3 Get all existing hotels

* Description: Get all existing hotels
* Validation (Throws an InvalidRequestException when):
  + N/A
* Request Body: N/A

### 2.1.4 Get all available hotels

* Description: Gets all available hotel inventories between specified dates. This endpoint takes into account pre-exisiting Bookings and hotel availability dates and only returns hotels that do not have overlapping Bookings and do not have availibility dates that start or end between the user specified dates.
* Validation (Throws an InvalidRequestException when):
  + Dates are not in YYYY-MM-DD format
  + dateFrom comes after dateTo
  + One or no dates are inputted
* Request Body:
  + Path Variables dateTo and dateFrom, in YYYY-MM-DD format

### 2.1.5 Get a user specified hotel

* Description: Get a user specified hotel object.
* Validation (Throws an InvalidRequestException when):
  + ID does not exist
* Request Body: N/A

### 2.1.6 Delete hotel

* Description: Delete a user specified Hotel
* Validation (Throws an InvalidRequestException when):
  + ID does not exist
  + Bookings for this hotel object exist (A Bookings object that contains a foreign key associated with the user inputted ID exists)
* Request Body:
  + Path Variable Integer "id"

## 2.2 Booking

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Endpoint | Description | Valid API Calls |
| POST | /api/v1/Booking | Creates a hotel Booking with an existing hotel id | Create Booking |
| GET | /api/v1/Booking/{id} | Get all exsiting hotel Bookings | Get all Bookings |
| GET | /api/v1/Bookings | Get an exsiting user specified Hotel Booking | Get Booking |
| DELETE | /api/v1/Booking/{id} | Delete an existing user specified Booking | Delete Booking |

### 2.2.1 Create Booking

* Description: Creates a hotel Booking with an existing hotel id.
* Validation (Throws an InvalidRequestException when):
  + The hotel object ID does not exist (The foreign key does not exist)
  + Check in and check out dates are not in YYYY-MM-DD format
  + Check in and check out dates do not exist
  + Check in date comes after check out date
  + Number of guests is not an Integer
  + A Booking object with overlapping dates already exists for the hotel object
* Request Body:

{

"hotelId": 0,

"checkIn": "YYYY-MM-DD",

"checkOut": "YYYY-MM-DD",

"guests": 0,

"status": true

}

### 2.2.2 Get all Bookings

* Description: Get all exsiting hotel Bookings.

### 2.2.3 Get Booking

* Description: Get an exsiting user specified Hotel Booking
* Validation (Throws an InvalidRequestException when):
  + ID does not exist
* Request Body:
  + Path variable Integer "id"

### 2.2.4 Delete Booking

* Description: Delete an existing user specified Booking.
* Validation (Throws an InvalidRequestException when):
  + ID does not exist
* Request Body:
  + Path variable Integer "id"

## 2.3 Database

* The database structure contains two tables, a hotel table that contains the hotel object.
* And a reservation table that contains reservations objects and is associated with the hotel table through the hotel table's Id by storing it as a foreign key in its "hotel Id" value.
* The table structures and values are as follows

### 2.3.1 Hotel

* Id: Integer (Primary Key)
* Name: VarChar
* Type: VarChar [ DELUXE, LUXURY, SUITE ]
* Description: VarChar
* Availiable From: Date YYYY-MM-DD
* Available To: Date YYYY-MM-DD
* Status: Boolean

### 2.3.2 Reservation

* Id: Integer (Primary Key)
* Inventory Id: Integer (Foreign Key)
* Check in Date: Date YYYY-MM-DD
* Check out Date: Date YYYY-MM-DD
* Number of Guests: Integer
* Status: Boolean

## 2.4 Testing

* Tests for this project incorporated both JUNit and Integration tests.
* I used Mockito is several unit tests and H2 in memory databases for the integration tests.
* Coverage testing for this project covered 93% of classes, 94% of methods and 85% of lines.
* More elaborate descriptions of the tests and their functionalities can be found in the test folder within this project.

## 2.5 Tech Stack

* API Creation:
  + Java
  + MySQL
  + SpringBoot
  + Hibernate
  + JPA
  + Lombok
* Testing:
  + JUnit 5
  + Mockito
  + H2
* User Input Testing:
  + Swagger UI
  + Postman