Project: Hostel Management Application

Back End Development

|  |  |  |
| --- | --- | --- |
|  | **FUNCTIONAL SPECIFICATION** | |
|  |  |  |
| **Project Code:** |  |  |
| **Project Name:** |  | Hostel Management |

**FUNCTIONAL SPECIFICATION**

**Table of Contents**

1. INTRODUCTION 3

2. SYSTEM OVERVIEW 3

3. SUB-SYSTEM DETAILS 4

4. DATA ORGANIZATION 5

5. REST APIs to be Built 6

6. ASSUMPTIONS 9

7. EXPECTATIONS 9

8. ACCEPTANCE CRITERIA 9

9. TRACEABILITY TO REQUIREMENTS 9



**FUNCTIONAL SPECIFICATION**

1. **Introduction**

“Hostel Booking System” is a web-based application. The main purpose of this application is to

provide convenient way for students to book the hostel rooms.

The company plans to develop "Hostel Booking System" - web application [J2EE Batches - Web Application], where users can sign up, login, book the hostel rooms according to their convince and make online payments. It is an easiest platform for all students which can be easily booked and know all the details.

**Scope and Overview:**

The scope of the “Hostel Booking System” will be to provide the functionality as described below. The system will be developed on a Windows operating system using Java/J2EE, Hibernate, Spring.

**2 System Overview**

The “Hostel Management System” should support basic functionalities (explained in section 2.1) for all below listed users.

* Admin
* Student
* Staff

***2.1* Authentication & Authorization**

**2.1.1 Authentication:**

Any end-user should be authenticated using a unique Username and Password

**2.1.2** **Authorization**

The operations supported and allowed would be based on the user type. For example, Admin has the right to add and modify the staff details, can modify Student details, add rooms, approve the allotment and also see all the invoices generated.

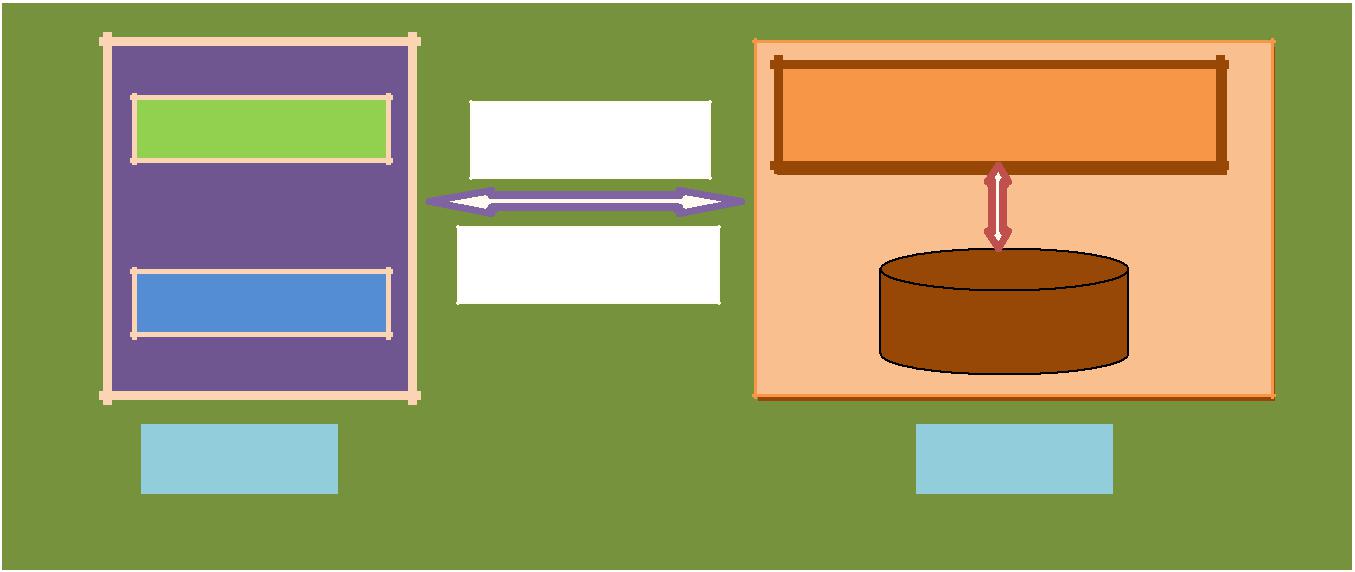
Whereas Student has right to add his details, Book a room, make a payments and download the invoice.

Student cannot book a room without filling his Student details. Once his Booking request is approved by the Admin then he cannot book a room. After the Approval of his booking request, he has to make the payment within the due time or else his booking status can be rejected by the Admin.

**2.2 Functional Flow**

The functional flow of the messages across different application components is shown below.

Ex. - Web Application.



|  |  |  |
| --- | --- | --- |
| **Admin** | **DES Application** |  |
| HTTP/GET |  |
|  | HTTP/Response |  |
| **Student** | Database |  |
|  |  |
| Client GUI | Server |  |
|  | **Online Hostel Booking Application** |  |
|  |  |  |
| **2.3 Environment** |  |  |

The system will be developed on any Windows OS machine using J2EE, Hibernate and Spring.

* Intel hardware machine (PC P4-2.26 GHz, 512 MB RAM, 40 GB HDD)
* Server – Apache Tomcat 9 or higher
* Database – Oracle 11g or higher
* JRE 8
* Spring Tool Suite



**FUNCTIONAL SPECIFICATION**

**3 Sub-system Details**

The Hostel Booking Application is defined, wherein all users need to login successfully before performing any of their respective operations.

Find below (section 3.1 & 3.2) tables that provides functionality descriptions for each type of user / sub-system. Against each requirement, indicative data is listed in column ‘Data to include’. Further, suggested to add/modify more details wherever required with an approval from customer/faculty.

**3.1 Admin**

The admin as a user is defined to perform below listed operations after successful login.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Objects | Operations | Data to include | Remarks |
| AD-001  To  AD-004 | User | View | Username, First name, Last name, Date of Birth, Address, Mobile, Email Address, Role, Photo |  |
| AD-005  To  AD-0010 | Staff | Add  View  Delete  Modify | Staff ID, Username, Date of Joining, Designation and Salary |  |
| AD – 0011  to  AD - 0013 | Student | Modify  Delete  View | Student ID, Username, Guardian First name, Guardian Last name, Guardian Mobile |  |
| AD – 0014  to  AD - 0016 | Room | Add  Modify  Delete  View | Room ID, Room type, Room Occupancy, Room Capacity, Fee, Booking Status |  |
| AD – 0017  to  AD - 0020 | Allotment | Approve  Reject | Allotment ID, Room\_Room ID, Student\_Student ID, Allocation Date, Status |  |
| AD – 0021  to  AD - 0024 | Invoice | View | Invoice ID, Allotment\_AllotmetID, Invoice Date |  |

**3.2 Student**

The student as a user is defined to perform below listed operations after successful login.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | Objects | Operations | Data to include |  | Remarks | | | | |
| US-001 | Student | Add | Student ID, Username, Guardian First name, Guardian Last name, Guardian Mobile |  |  |  |  |  |  |
| US-002 | Room | Book | Room ID, Room type, Room Occupancy, Room Capacity, Fee, Booking Status |  |  |  |  |  |  |
| US-003 | Invoice | Make Payment  Download | Invoice ID, Allotment\_AllotmetID, Invoice Date |  |  |  |  |  |  |

**FUNCTIONAL SPECIFICATION**

**3.3 Signup | Login | Logout**

**[Web Application - J2EE, Hibernate, Spring]**

* Go to the Signup Page if you are new user, fill all the details required then click on Signup button.
* Go to Home page when you login successfully after entering valid username & password fetched from the database.
* Redirect back to same login screen if username & password are not matching and also display the error message saying “Login Failed”.
* Implement Session tracking for all logged in users before allowing access to application features. Anonymous users should be checked, unless explicitly mentioned.

**4 Data Organization**

This section explains the data storage requirements of the Hostel Booking Application and **indicative** data description along with suggested table (database) structure. The following section explains few of the tables (fields) with description. However, in similar approach need to be considered for all other tables.

**4.1 Table: Usermaster**

The user specific details such as username, email, phone etc. Authentication, and authorization / privileges should be kept in one or more tables, as necessary and applicable.

|  |  |
| --- | --- |
| **Field Name** | **Description** |
| *Username* | Username of the person |
|  |  |
| *password* | User Password |
| *First name* | First name of user |
| *Last name* | Last name of the user |
| *Date of Birth* | Birth date of the user |
| *Address* | Complete Address of the user |
| *Mobile* | 10 digit mobile number of the user |
| *Email Address* | Email Id of the user |
| *Role* | Role of the user either Admin, Student or Staff |
| *Photo* | Image of the user |

**4.2 Table: Staff Details**

This table contains information related to a staff details

|  |  |
| --- | --- |
| **Field Name** | **Description** |
| *Staff Id* | Unique Staff Id, Here Staff Id will be Primary Key |
| *User Name* | Username of that particular Staff, this is the foreign key |
| *Date of Joining* | The joining date of the particular Staff |
| *Designation* | Designation of the staff for staff. For example, DBA, Electrician etc |
| *Salary* | Salary depending of the designation |

**4.3 Table: Student Details**

This table contains information related to student details

|  |  |
| --- | --- |
| **Field Name** | **Description** |
| *Student Id* | Unique Student ID Auto Generated |
| *Username* | Username corresponding to logged in user |
| *Guardian First name* | The student’s guardian first name |
| *Guardian Last name* | The student’s guardian last name |
| *Guardian Mobile* | The Mobile number of the student’s guardian |

**4.4 Table: Room**

This table contains information related to Room Booking deatils

|  |  |
| --- | --- |
| **Field Name** | **Description** |
| *Room Id* | Unique Room Id, this field is a primary key |
| *Room type* | Type of the rooms available. For example, Deluxe, Standard etc |
| *Room Occupancy* | The currently available vacancy in a room |
| *Room capacity* | The maximum number of Student allowed to stay in the particular room |
| *Fee* | The fee of the room per year |
| *Booking Status* | The Status which is “Available” by default and becomes “Booked once the Occupancy is zero. |

**4.5 Table: Allotment**

This table contains information related to Allotment details

|  |  |
| --- | --- |
| **Field Name** | **Description** |
| *Allotment Id* | Unique Allotment Id, this field is a primary key |
| *Room Id* | Room Id is foreign key |
| *Student Id* | The Id of Student who has requested for booking a room. This is a foreign key |
| *Allocation Date* | The date on which the allotment will be “Approved”. |
| *Status* | The status will be “Pending” by default. Later changes to “Approved” or “Rejected”. |

**4.6 Table: Invoice**

This table contains information related to final checkout details

|  |  |
| --- | --- |
| **Field Name** | **Description** |
| *Invoice Id* | Unique invoice Id, this field is a primary key |
| *Allotment Id* | The Allotment Id is foreign key |
| *Invoice Date* | The date on which the payment is done. |

**FUNCTIONAL SPECIFICATION**

1. **REST APIs to be Built**

Create following REST resources which are required in the application,

Creating **User** Entity: Create Spring Boot with Microservices Application with Spring Data JPA

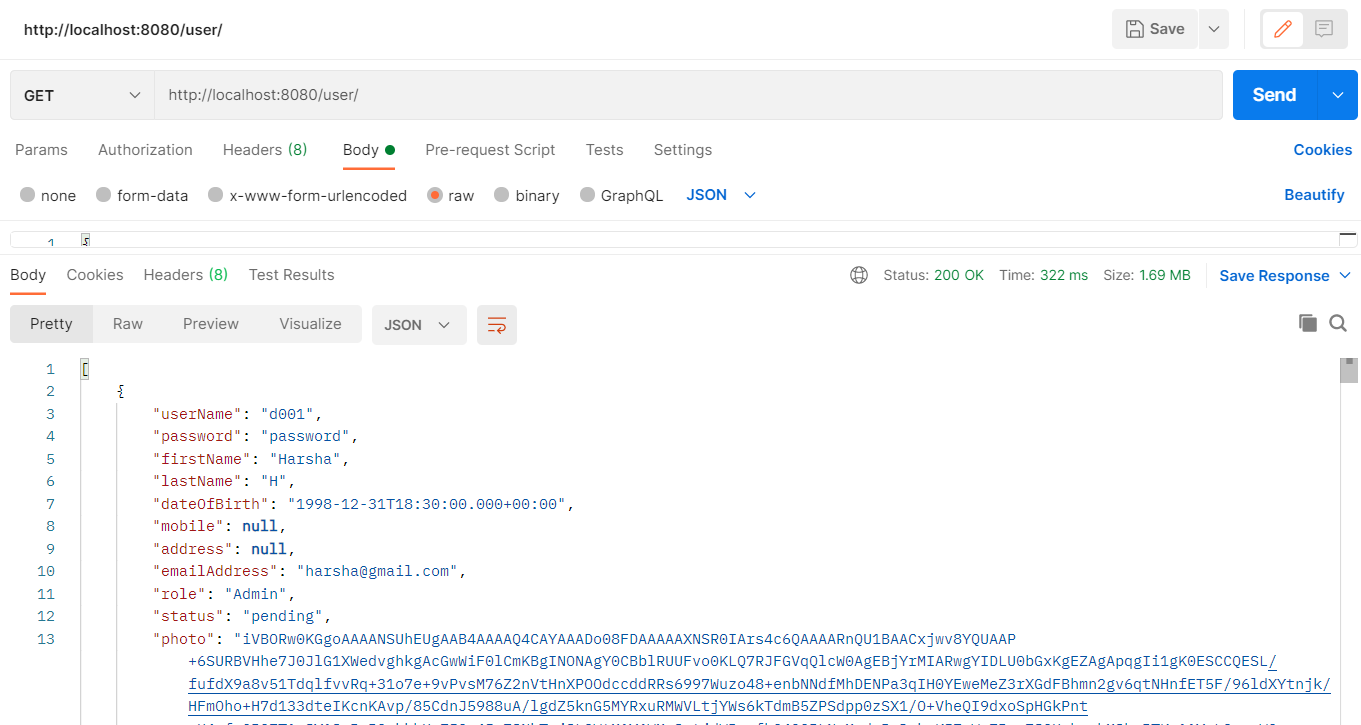
**Technology stack:**

* Spring Boot
* Spring REST
* Spring Data JPA

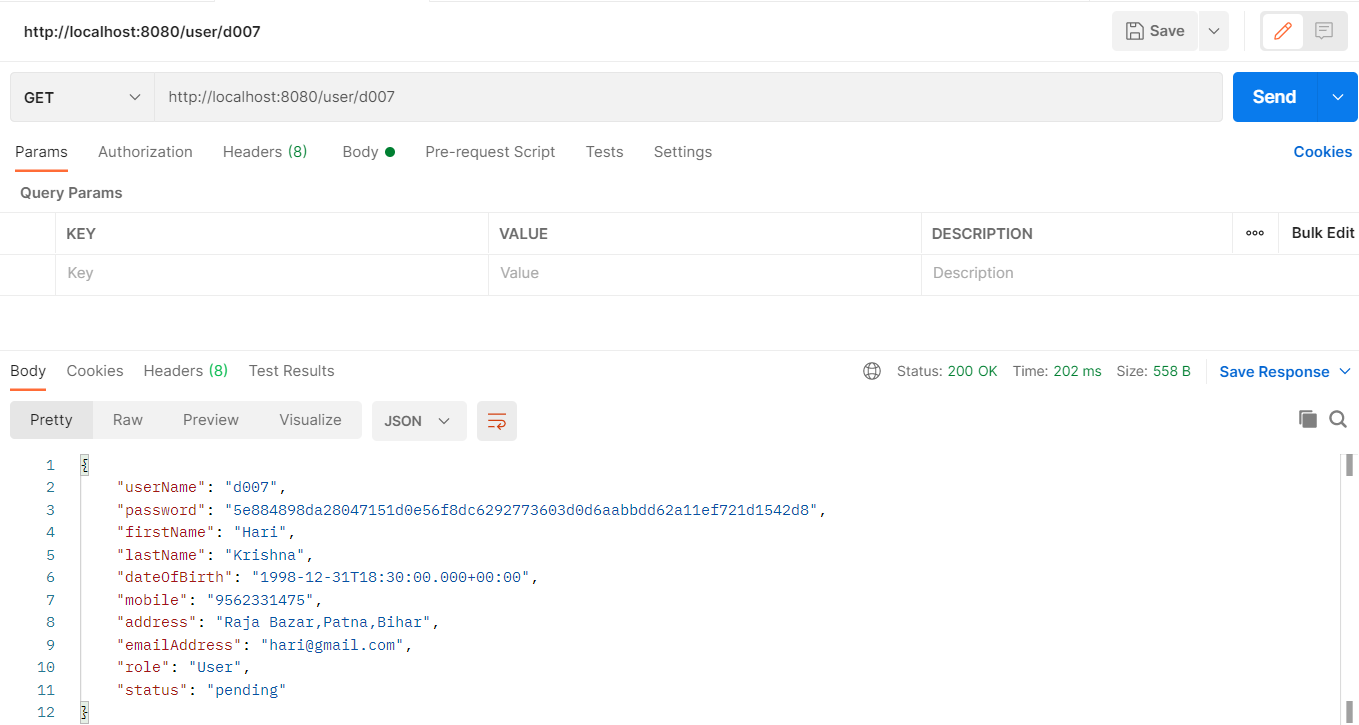
1. Here will have multiple layers into the application:
2. Create an Entity: User
3. Create a UserRepository interface and will make use of Spring Data JPA
4. Will have validateLogin method
5. Will have getAllUserName method
6. Create a UserService class and will expose all these services
7. Finally, create a UserController will have the following Uri’s:

|  |  |  |  |
| --- | --- | --- | --- |
| URI | METHODS | Description | Format |
| /user | GET | This is to get the details of all “Users” | JSON |
| /users/userName | GET | Give a single user description searched based on userName | String |
| /user | POST | Add the user details | JSON |
| /user | PUT | Update the user details | JSON |
| /user/userName | DELETE | Delete user by userName | String |
| /getAllUserName | GET | This is to get all Usernames | String |
| /user/otp | POST | This will send the OTP to the email id provided in the signup form | String |
| /user/login | POST | Validates the username and password before login. |  |

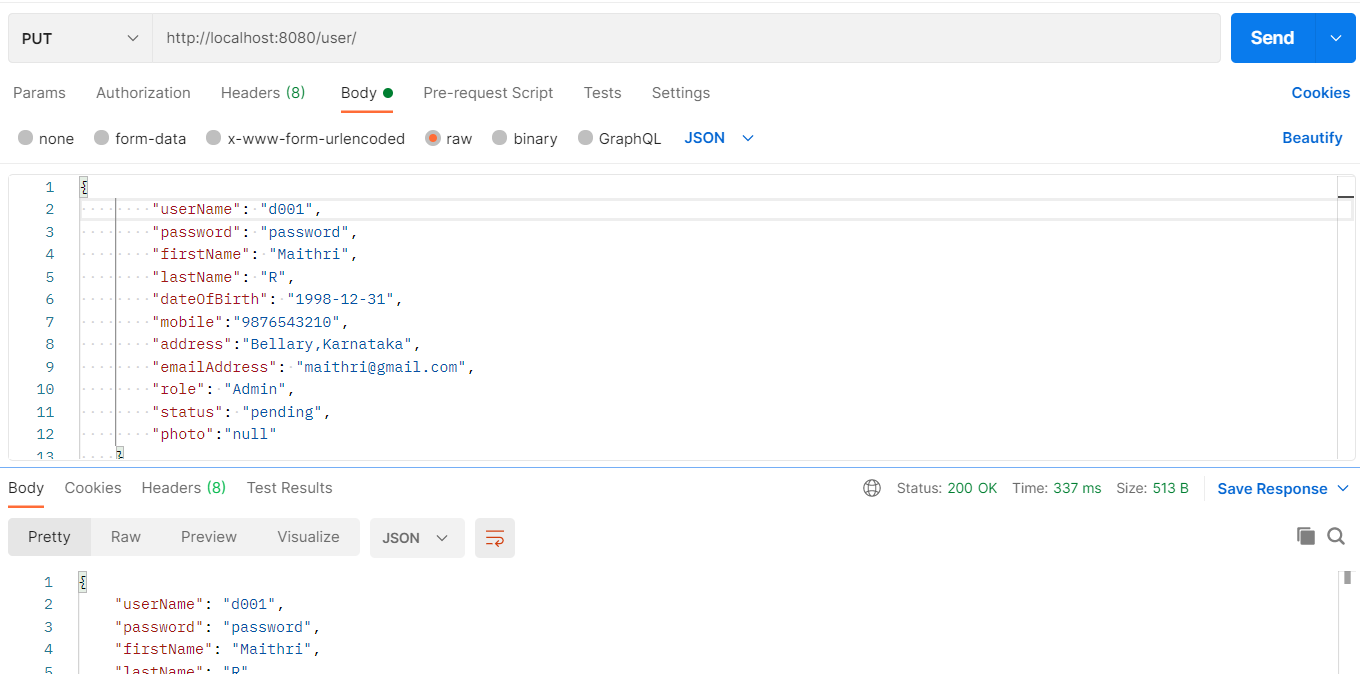
1. POSTMAN :



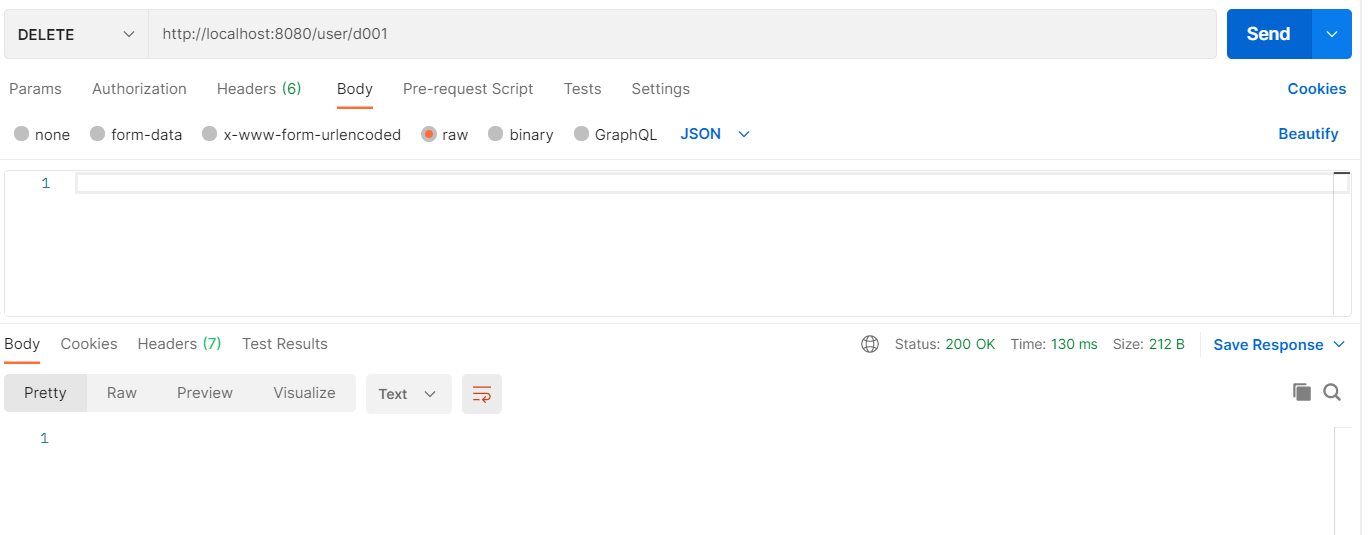
Read all operation in POSTMAN



Read by ID operation in POSTMAN



Modify Operation in POSTMAN



Delete Operation in POSTMAN

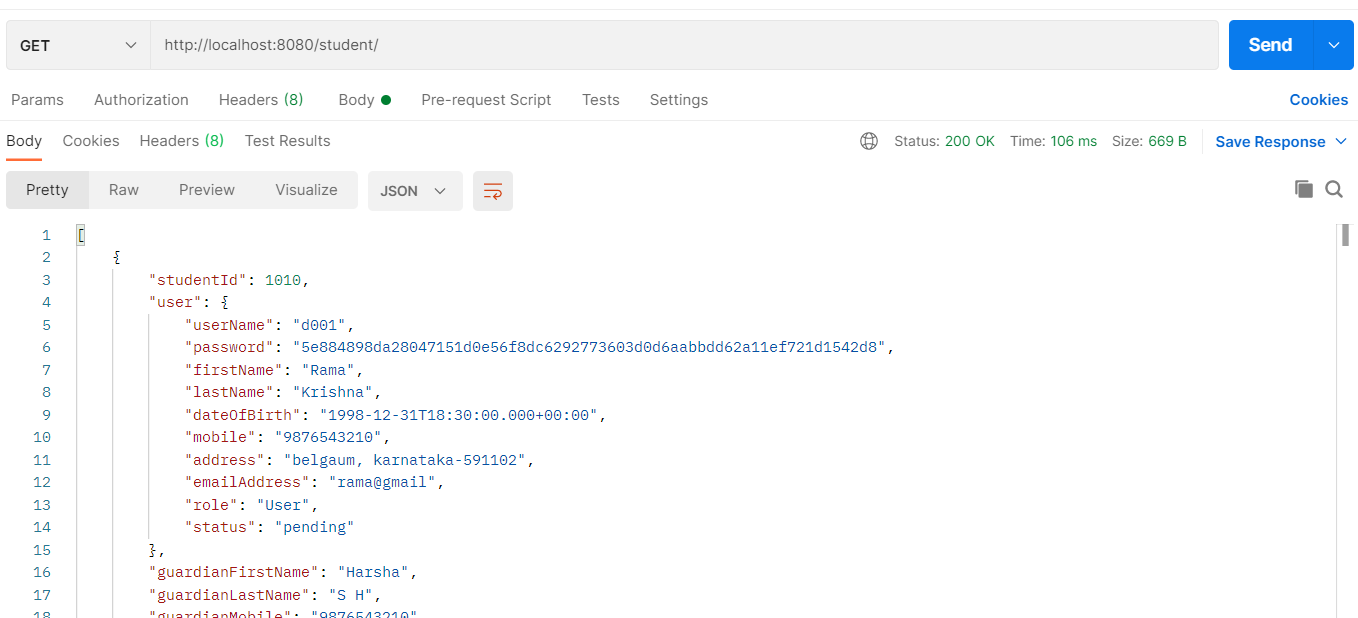
1. Creating **Student** Entity:

Build a RESTful resource for **Student** manipulations, where CRUD operations to be carried out. Here will have multiple layers into the application:

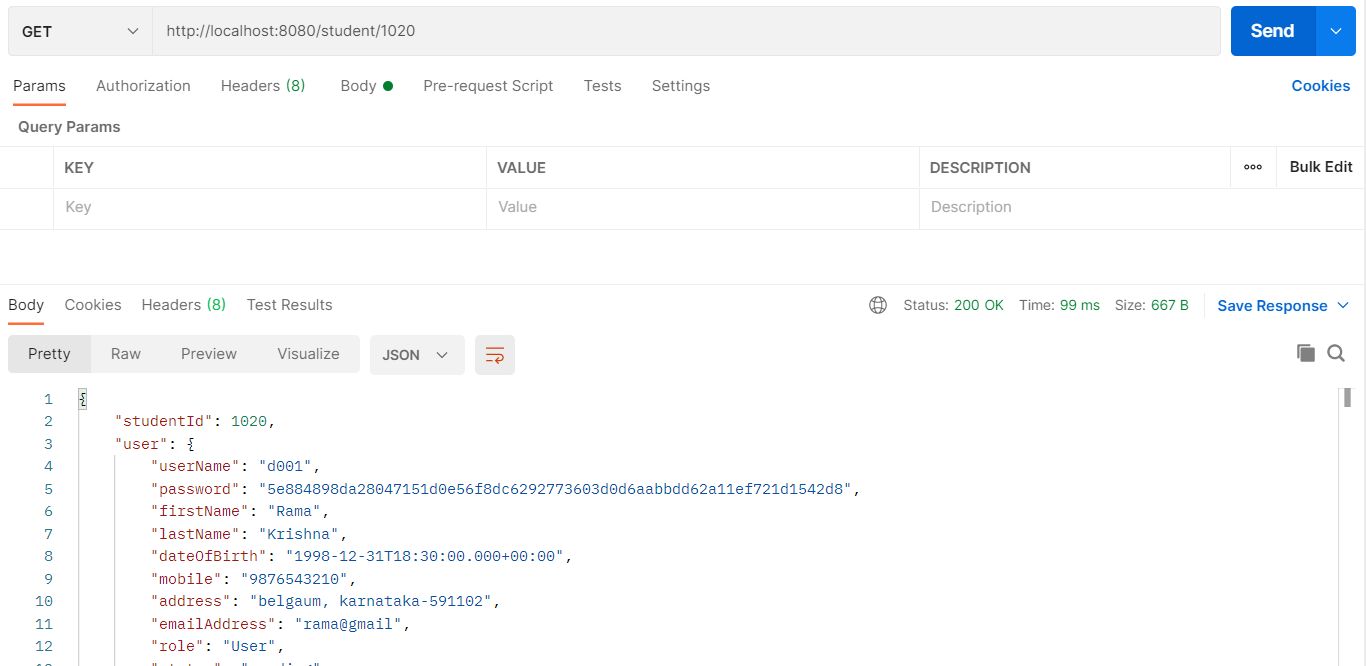
1. Create an Entity: Student
2. Create a StudentRepository interface and will make use of Spring Data JPA
3. Will have FindStudentByUserName method
4. Will have FindUserByStudentId method
5. Create a StudentService class and will expose all these services
6. Finally, create a StudentController will have the following Uri’s:

|  |  |  |  |
| --- | --- | --- | --- |
| URI | METHODS | Description | Format |
| /student/ | GET | Get all the students | JSON |
| /student/studentId | GET | Give a single student description searched based on Student Id | JSON |
| /student/ | POST | Add the student details | JSON |
| /student/ | PUT | Modify the student details | JSON |
| /student/studentId | DELETE | Delete a student based on student id | JSON |
| /student/findByUserName | GET | Find the Student by particular Username | JSON |

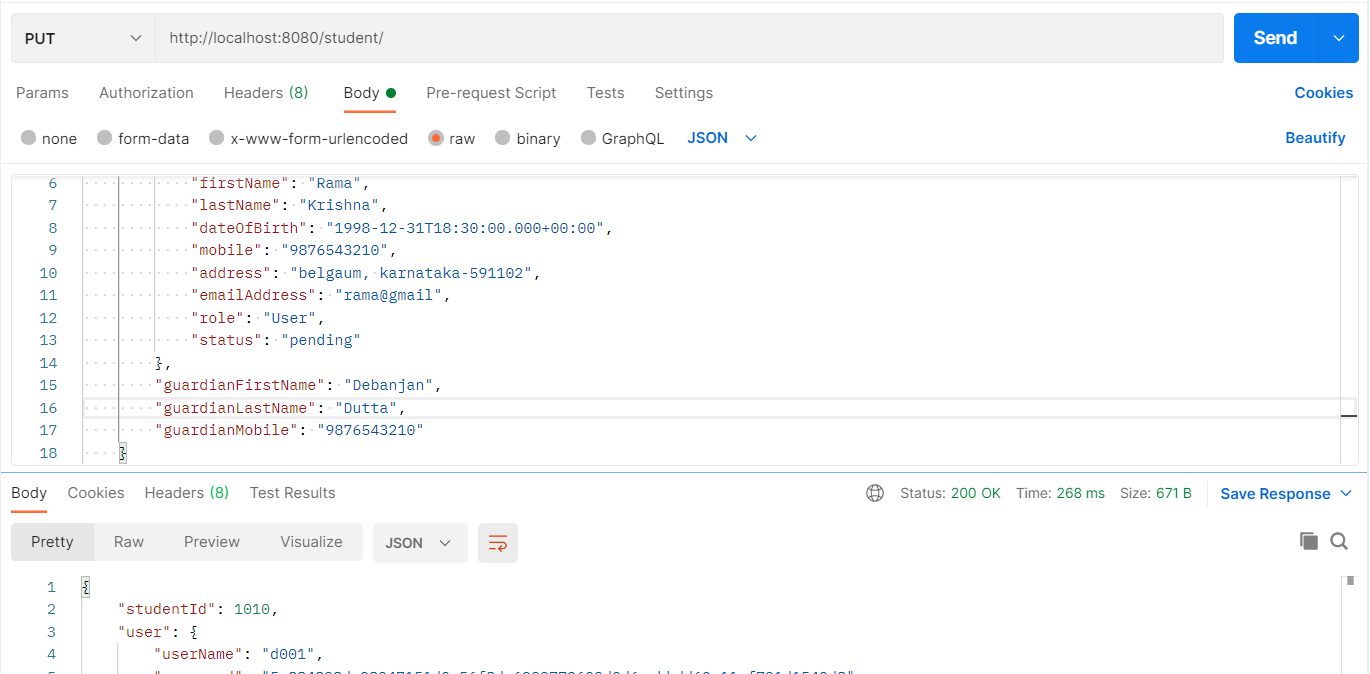
1. POSTMAN:



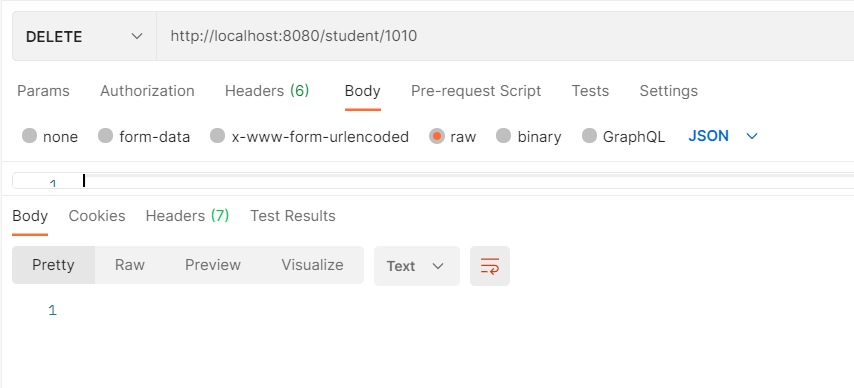
Read all operation in POSTMAN



Read by ID operation in POSTMAN



Modify operation in POSTMAN



Delete operation in POSTMAN

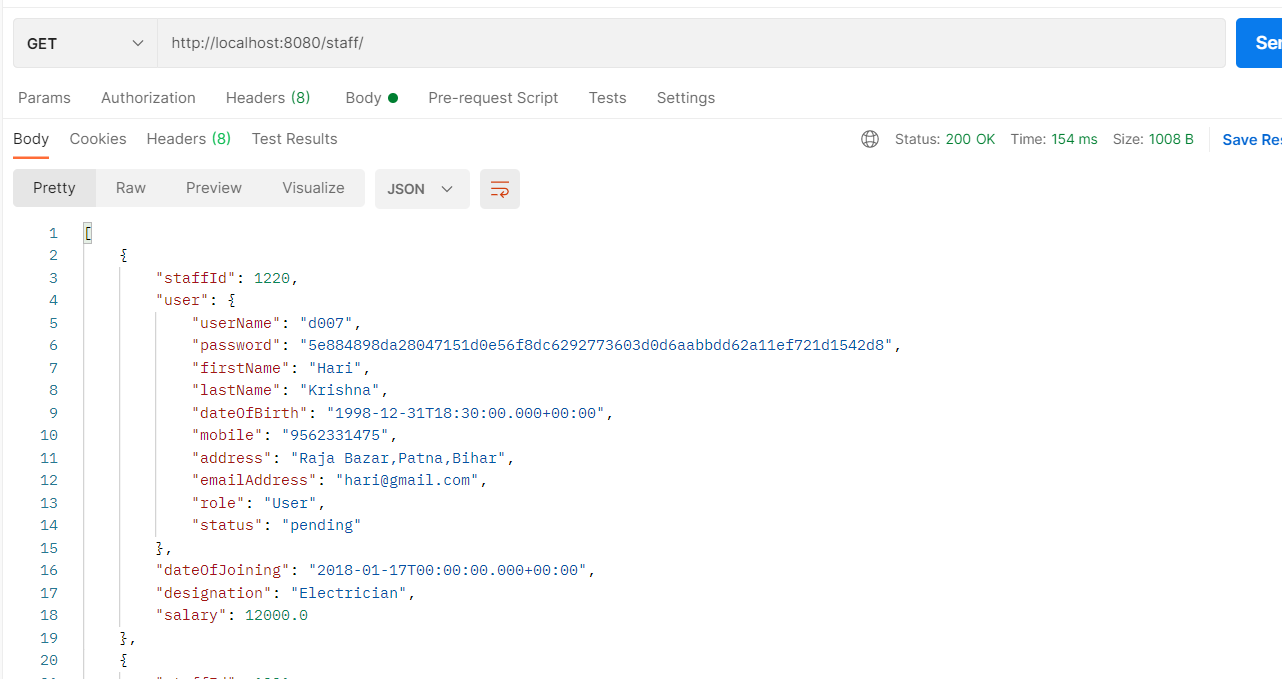
1. Creating **Staff** Entity:

Build a RESTful resource for **Staff** manipulations, where following operations to be carried out. Here will have multiple layers into the application:

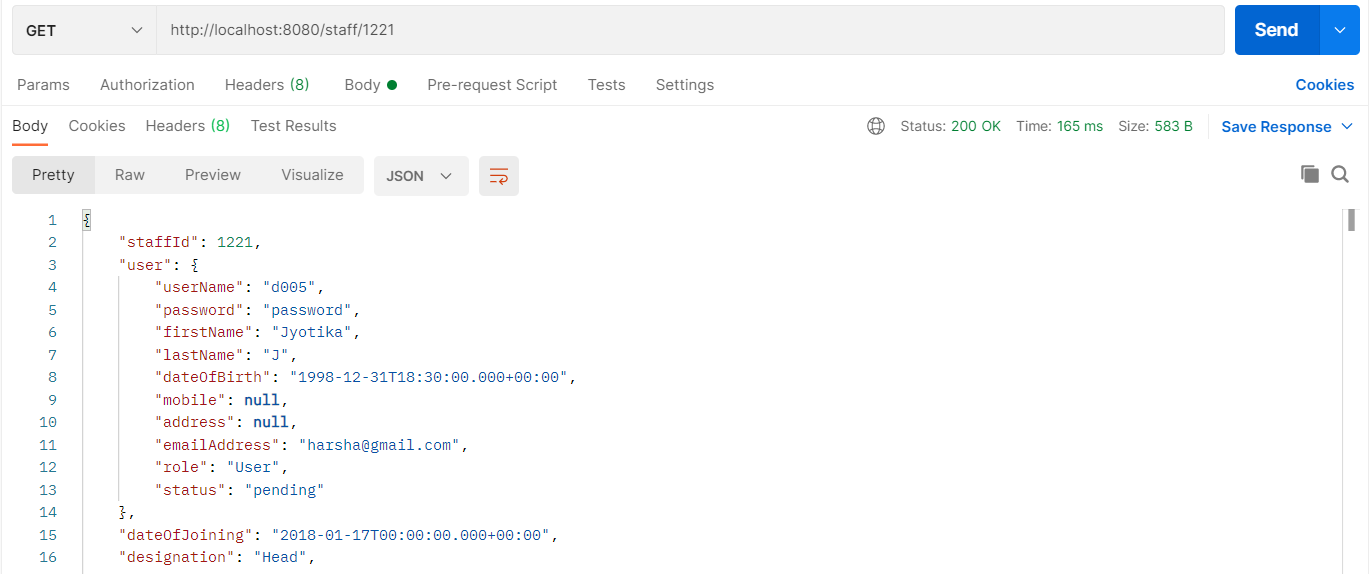
1. Create an Entity: Staff
2. Create a StaffRepository interface and will make use of Spring Data JPA
3. Create a StaffService class and will expose all these services
4. Finally, create a StaffController will have the following Uri’s:

|  |  |  |  |
| --- | --- | --- | --- |
| URI | METHODS | Description | Format |
| /staff/ | GET | Gets the details of all staff | JSON |
| /staff/staffId | GET | Give a single staff description searched based on Staff Id | JSON |
| /staff/ | POST | Add the staff details | JSON |
| /staff/ | PUT | Modify the staff details | JSON |
| /staff/staffId | DELETE | Delete a student based on staff id | JSON |

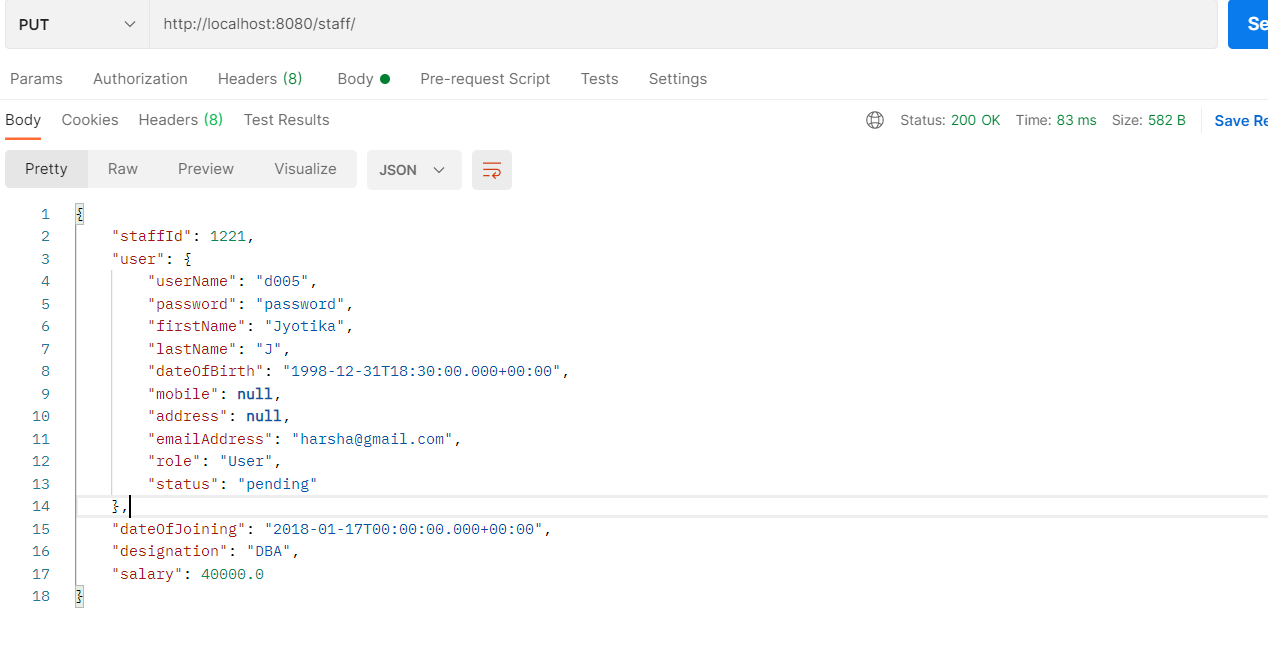
1. POSTMAN:



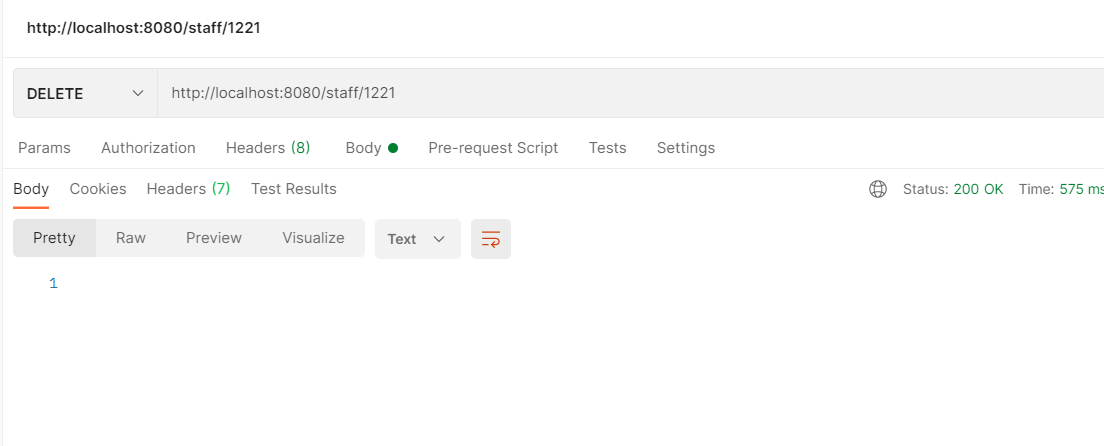
Read all operation in POSTMAN



Read by ID operation in POSTMAN



Modify Operation in POSTMAN



Delete operation by ID in POSTMAN

1. Creating **Room** Entity:

Build a RESTful resource for **Room** manipulations, where following operations to be carried out. Here will have multiple layers into the application:

1. Create an Entity: Room

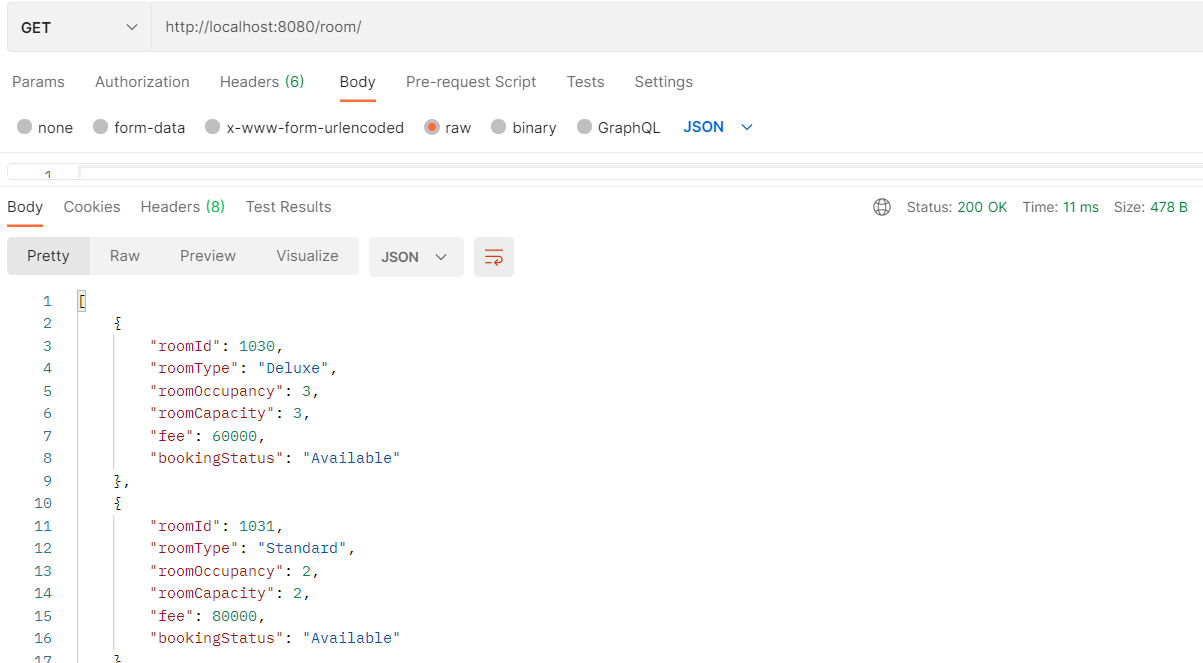
2. Create a RoomRepository interface and will make use of Spring Data JPA

3. Create a RoomService class and will expose all these services

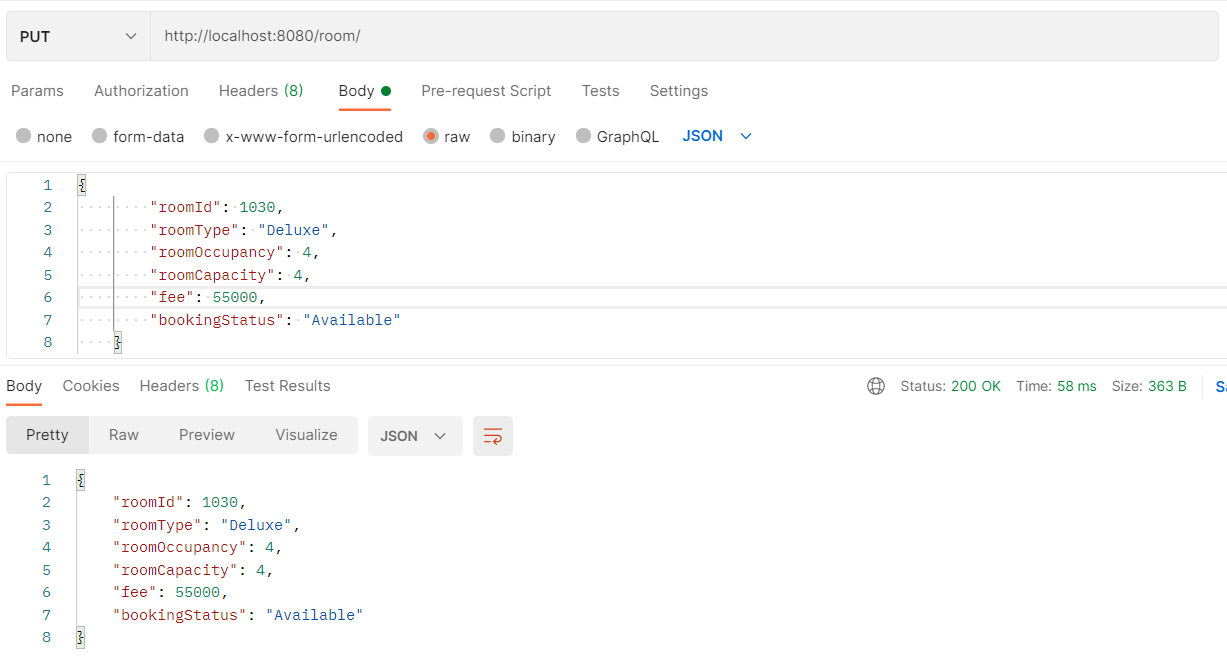
4. Finally, create a RoomController will have the following Uri’s:

|  |  |  |  |
| --- | --- | --- | --- |
| URI | METHODS | Description | Format |
| /room/ | GET | Gets the details of all room | JSON |
| /room/roomId | GET | Give a single room description searched based on Room Id |  |
| /room/ | POST | Add the room details |  |
| /room/ | PUT | Modify the room details |  |
| /room/roomId | DELETE | Delete the room based on the Room Id |  |
| /room/roomOccupancy | PUT | Update the occupancy of a room (either increase or decrease) |  |

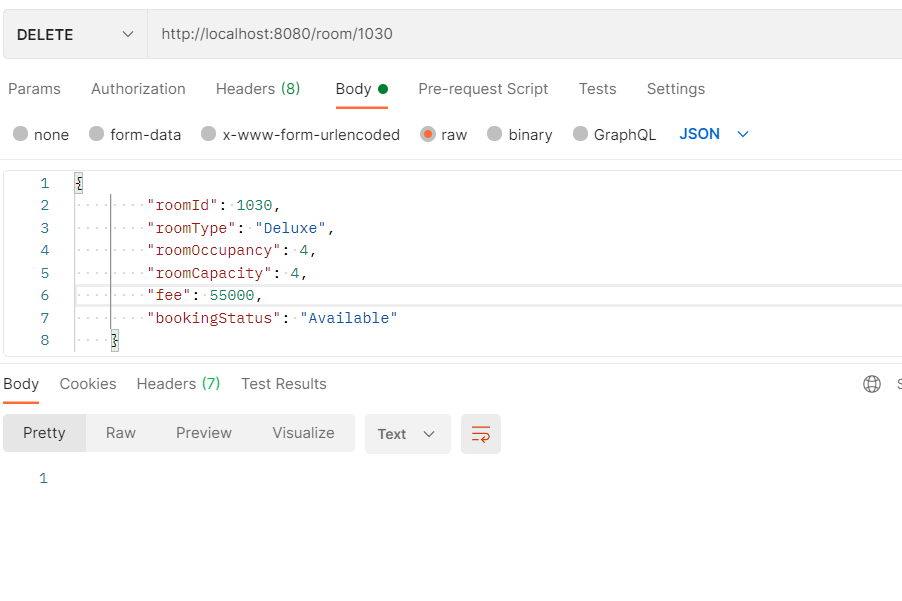
5. POSTMAN:



Read all operation in POSTMAN



Modify Room details operation in POSTMAN



Delete room operation in POSTMAN

1. Creating **Allotment** Entity:

Build a RESTful resource for **Allotment** manipulations, where following operations to be carried out. Here will have multiple layers into the application:

1. Create an Entity: Allotment

2. Create an AllotmentRepository interface and will make use of Spring Data JPA

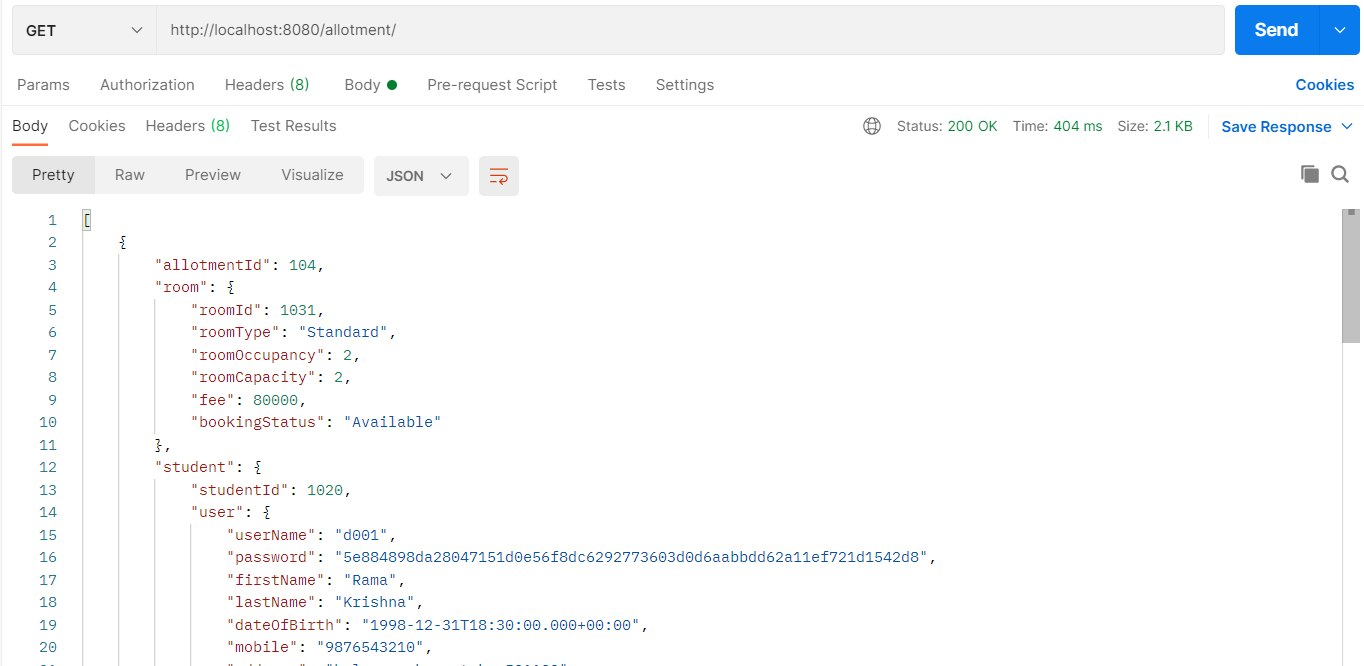
1. Will have updateStatus method
2. Will have findAllotmentByStudentId method
3. Will have findPaidAllotmentByStudentId method
4. Will have findUnpaidAllotmentbyStudentId method
5. Will have findAllotmentForStatusNotApproved method

3. Create a AllotmentService class and will expose all these services

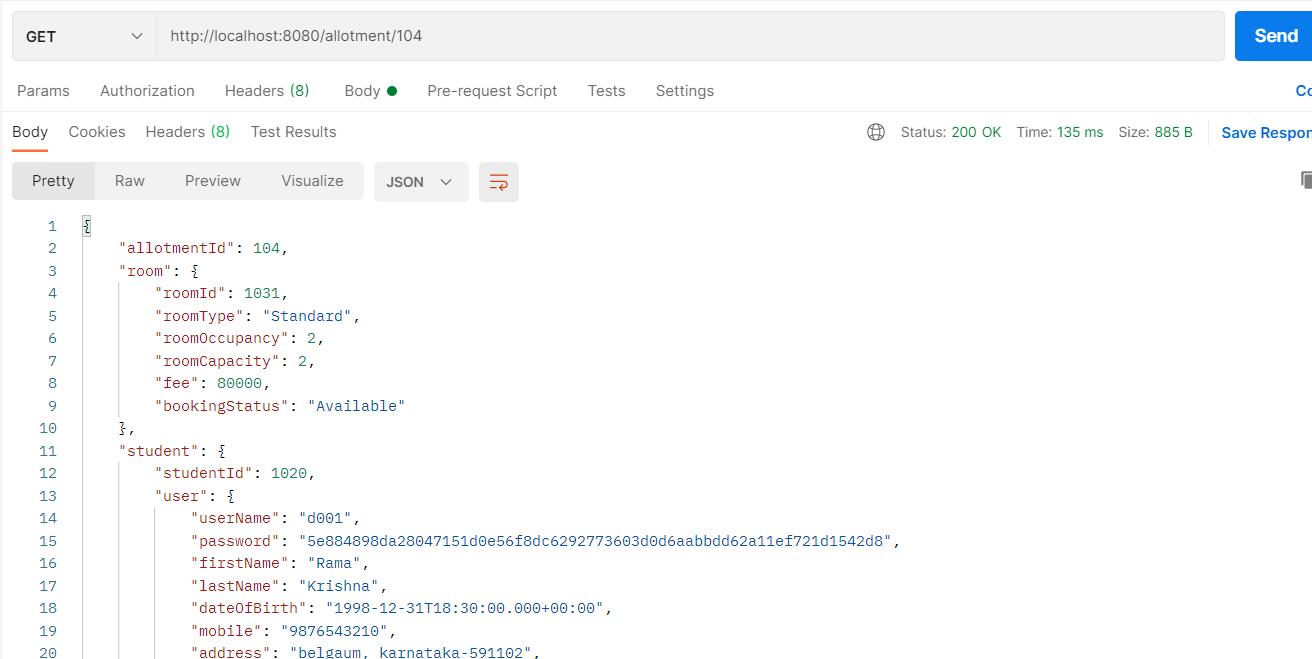
4. Finally, create a AllotmentController will have the following Uri’s:

|  |  |  |
| --- | --- | --- |
| URI | METHODS | Description |
| /allotment/ | GET | Gets the details of all allotment |
| /allotment/allotmentId | GET | Give a single allotment description searched based on Allotment Id |
| /allotment/ | POST | Add the allotment details |
| /allotment/ | PUT | Modify the allotment details |
| /allotment/allotmentId | DELETE | Delete the allotment based on the Allotment Id |
| /allotment/updateStatus/allotmentId/status | PUT | Updates the Status as either “Approved” or “Rejected” based on the Allotment ID |
| /allotment/findAllotmentByStudentId/studentId | GET | Gives the Allotment details for the particular StudentId |
| /allotment/findPaidAllotmentByStudentId/studentId | GET | Gives the Allotment details for the particular student who have already paid for the particular allotment |
| /allotment/findUnPaidAllotmentByStudentId/studentId | GET | Gives the Allotment details for the particular student who has not paid for the particular allotment |
| /allotment/findAllotmentForStatusNotApproved/studentId | GET | This method return the number as 0 or 1. If 0 then the Student based on that studentId has either not booked the room yet or his booking request has been rejected. And if 1 then his Booking Status has been Approved. |

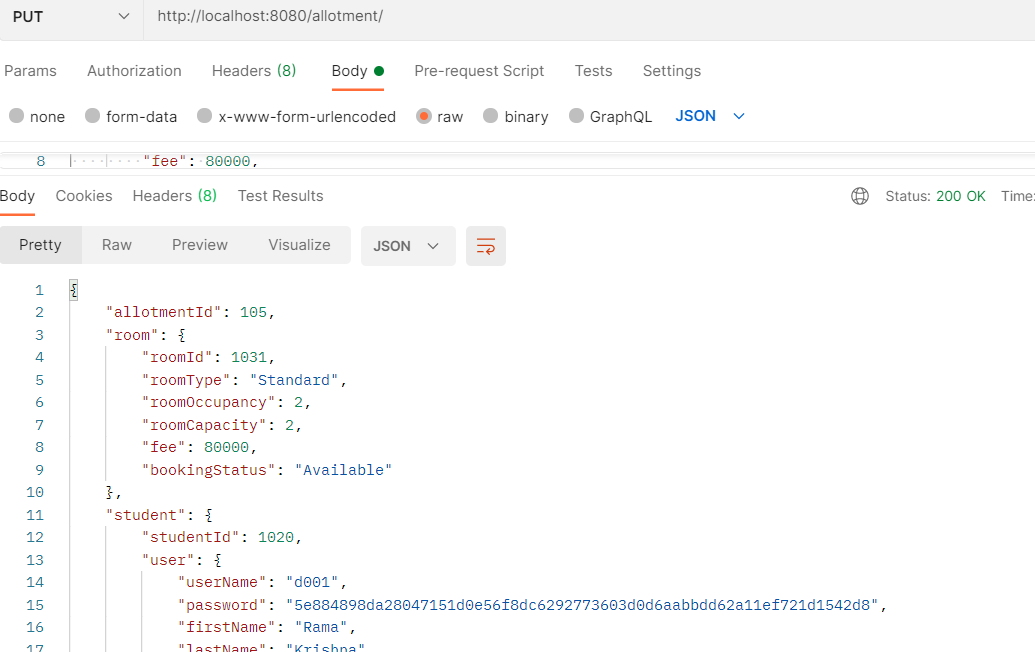
1. POSTMAN:



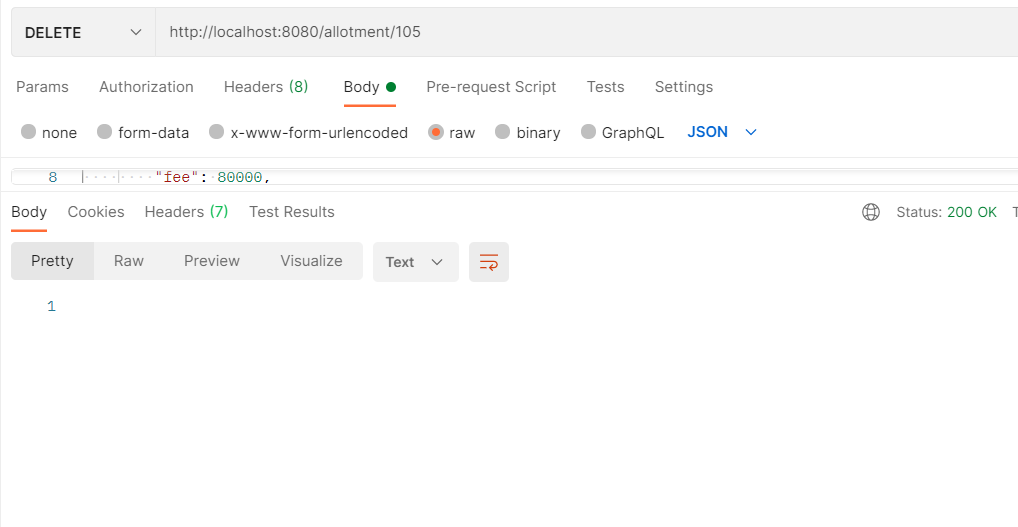
Read all allotment operation in POSTMAN



Read by ID operation in POSTMAN



Modify operation in POSTMAN



Delete operation by ID in POSTMAN

1. Creating **Invoice** Entity:

Build a RESTful resource for **Invoice** manipulations, where following operations to be carried out. Here will have multiple layers into the application:

1. Create an Entity: Invoice

2. Create a InvoiceRepository interface and will make use of Spring Data JPA

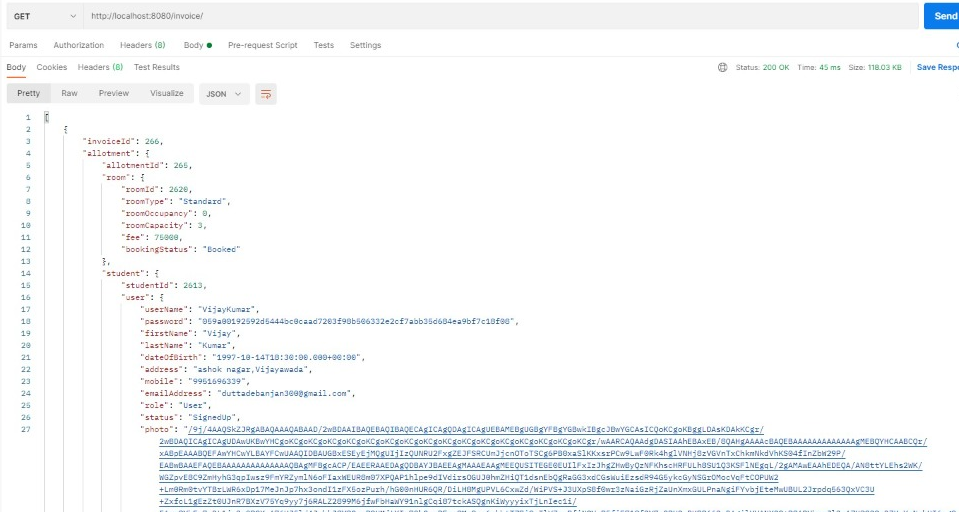
1. Will have findInvoiceByAllotmentId method

3. Create a InvoiceService class and will expose all these services

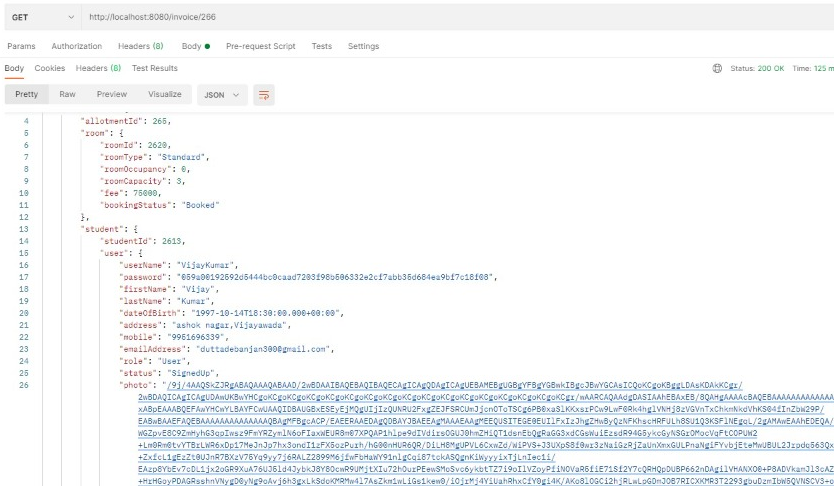
4. Finally, create a InvoiceController will have the following Uri’s:

|  |  |  |
| --- | --- | --- |
| URI | METHODS | Description |
| /invoice/ | GET | Gets the details of all invoices |
| /invoice/invoiceId | GET | Give a single invoice description searched based on Invoice Id |
| /invoice/ | POST | Add the invoice details |
| /invoice/ | PUT | Modify the invoice details |
| /invoice/invoiceId | DELETE | Delete the invoice based on the Invoice Id |
| /invoice/ /findInvoiceByAllotmentId/allotmentId | GET | Gives the Invoice details for particular Allotment ID |

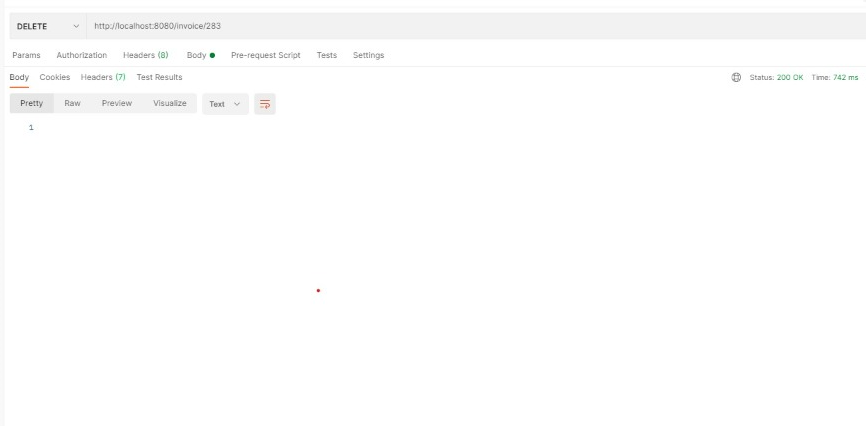
5. POSTMAN:



Read all operation in POSTMAN



Read by ID operation in POSTMAN



Delete by ID operation in POSTMAN

**FUNCTIONAL SPECIFICATION**

1. **Assumptions**

* User Interface: The type of client interface (front-end) to be supported - Angular based.
* The Admin can add, modify and remove rooms into the database
* Without filling the Student Details, Student cannot book a room.
* Once the student fills Student details, he will not be allowed to fill the details again. For any Student Detail modification the student must contact the Admin.
* Once the student request the booking for the particular room, Only the Admin can approve it or reject it.
* After the Approval student can make the payment in the invoice page and download his invoice. And also he won’t be allowed to book any other room for next one year.
* And if the payment isn’t done within the due time (24 hours) his booking request will be rejected.
* After the rejection of request a Student can again book a room.

1. **General Expectations**

* Participants must create the **Class Diagram, Sequence Diagram and ER Diagram**.
* Participants must do **Unit testing and Functional Testing using POSTMAN tool**
* Integration of Angular and Spring Boot with Microservices should be done, referring **Project** **2 -Frond End Development Project.**
  + The server should be a concurrent server servicing multiple client.
  + Database can be implemented using Oracle 11g or above.
  + To begin with, the application should support at least 1 admin and 2 customers.
  + Compilation and Build should be done using Eclipse IDE or STS
  + Source-code and all documents must be maintained (checked-in) in configuration management system (subversion)
  + Coding standards (for Java) should be followed

NOTE:

1. **Validation of user Data**

* Spring MVC using JSR-303 annotations
* AJAX validation without forcing the page to reload (Wherever applicable)
* JavaScript validation (if necessary)

1. **UI Design – (for Web Application) Use DIV/CSS or Semantic Elements to control the style and layout**
2. **Create at least one SQL DML-statement inside PL/SQL blocks**

**8 Acceptance Criteria**

All P1 requirements must be mandatorily implemented

**9 Traceability to Requirements**

Appropriate requirements from RS and FS are mapped here.

**Document Reference ID & Description: (Doc ID from which this document is derived)**

|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **Reference document: RS** | **Current document: FS** |
|  | **Requirement/Feature (Section ID/Name)** | **Location (Section ID/Name)** |

1.

2.