EDUBRIDGE

COURSE

Certified Java Full Stack Professional

A PROJECT REPORT ON

INVENTORY MANAGEMENT SYSTEM

Submitted By

Ms.Deepika S

Ms.Lokeshwari S

Mrs.Renugadevi R

Ms.Shipra Mahato

Mrs.Sri Vidhya C

Batch No. 2021-6769

Under the Guidence of

Trainer Mrs. Indrakka Mali Mam

Table of Contents

INTRODUCTION

Inventory management System

An inventory management system is the process by which you track your goods throughout your entire supply chain, from purchasing to production to end sales. In our project we can implement the shopkeeper module maintain the product details and store all the information of customer on the database.

Functions of Inventory Management System

- Increase productivity: An inventory management system enhances the productivity of the company to a greater extent.
- Avoid stock-outs and over-stocks: An inventory management system assists companies with avoiding over-stocks and stock-out scenarios. So, the company never stops functioning.
- Manage quality and Inspect products: This system is responsible for quality inspection and management of the product. Thus, inventory software potentially manages the inventory of the company in a planned and convenient manner.

OBJECTIVIES

- It provides "better and efficient" service".
- Faster way to get details about the product and customer information.
- Provide facility for proper monitoring and reduce paper work.
- All details will be available on a click.

System View

- The Inventory management system will be automated the traditional system.
- There is no need to use paper and pen.
- Checking a product details is very easy.
- Adding new customer record is very easy.
- Creating, Updating, or Deleting a product and customer field is very to implement using some certain methods.

SYSTEM SPECIFICATION

Hardware Requirements

Processor	Intel Core i3-3210M CPU
RAM	8 GB
Hard Disk	I TB

Software Requirements

Operating System	Windows 10
Technology	Postman
Language	java 8
IDE	Spring Tool Suite
Database	MySQL

Language Descriptions

Java 8

- Java provided supports for functional programming, new JavaScript engine, new APIs for date time manipulation, new streaming API, etc.
- Java 8 is a revolutionary release of the world's 1 development platform.
- Java 8 include features for productivity, ease of use, improved the programming, security and improved performance.
- It include a huge upgrade to the java programming model and a coordinated evolution of the JVM, java language and libraries.
- In Java 8, a new notion called functional interfaces was introduced. A Functional Interface is an interface that has exactly one abstract method.
- The @FunctionalInterface annotation prevents abstract methods from being accidentally added to functional interfaces. It's similar to a @Override annotation, and it's recommended that you use it. java.lang.

MySQL

- MySQL is a widely used relational database management system (RDBMS).
- MySQL is free and open-source.MySQL is ideal for both small and large applications.
- SQL is used to insert, search, update, and delete database records.
- MySQL is ideal for both small and large applications.
- MySQL is very fast, reliable, scalable, and easy to use and MySQL is cross-platform

Spring Tool Suite

- Spring Tool Suite is built on the top of the spring and contains all the features of spring.
- And is becoming a favourite of developers these days because it’s a rapid production-ready environment that enables the developers to directly focus on the logic instead of struggling with the configuration and setup.
- Spring Boot is a microservice-based framework and making a production-ready application in it takes very little time.

Postman

- Postman is a standalone software testing API (Application Programming Interface) platform to build, test, design, modify, and document APIs.
- It is a simple Graphic User Interface for sending and viewing HTTP requests and responses.
- The Postman platform includes **a comprehensive set of tools that help accelerate the API lifecycle**—from design, testing, documentation, and mocking to the sharing and discoverability of your APIs.

MODULE DESCRIPTION

Shopkeeper Module

- In this class we implement the information of product details, and create the join column to combine the customer class to view the details of product to buy the customer.
- Shopkeeper class using variables are total stock, product id, product name, product type, product brand, product quality, product price, current stock and total amount.
- Details of product will be stored on the database will be viewed using the postman get the details of stock description.
- Using GET method to view the all details of product and customer information.

Customer Module

- In Customer Class used to define the customer details and product brought by customer.
- Custom variables are customer id, customer name, customer mobile number and customer address are used on this class connection to shopkeeper class.
- Fetch the details of product brought by customer using foreign key reference get the product details of customer.
- Using search by id, update and delete the customer details.

CRUD OPERTION

CRUD stands for Create, Read/Retrieve, Update and Delete and these are the four basic operations that we perform on persistence storage. CRUD is data-oriented and the standardized use of HTTP methods.

So, standard CRUD Operations is as follows:

- **POST**: Creates a new resource
- **GET**: Reads/Retrieve a resource
- **PUT**: Updates an existing resource
- **DELETE**: Deletes a resource

As the name suggests

- **CREATE Operation**: Performs the INSERT statement to create a new record.
- **READ Operation**: Reads table records based on the input parameter.
- **UPDATE Operation**: Executes an update statement on the table. It is based on the input parameter.
- **DELETE Operation**: Deletes a specified row in the table. It is also based on the input parameter.

ANNOTATIONS

1**.@entity**: The @Entity annotation specifies that the class is an entity and is mapped to a database table.

2**.@Table**: annotation specifies the name of the database table to be used for mapping.

3. **@Id**: annotation used for the primary key in database

4**.@NotNull**: content is not null on the table.

5**.@NotBlank**: used to content is not empty on the table

6. **@length**: used to specify the length of content placed on the table and using minimum, maximum length can be defined on the table.

7**.@Service**: We mark beans with @Service to indicate that they're holding the business logic. Besides being used in the service layer, there isn't any other special use for this annotation.

8. **@Repository**: @Repository's job is to catch persistence-specific exceptions and re-throw them as one of spring's unified unchecked exceptions.

9. **@Autowired**: The spring framework enables automatic dependency injection. In other words, by declaring all the bean dependencies in a spring configuration file, Spring container

can autowire relationships between collaborating beans. This is called spring bean auto wiring.

10**.@RestController**: RestController is used for making restful web services with the help of the @RestController annotation. This annotation is used at the class level and allows the class to handle the requests made by the client.

11**.@GetMapping**: The @GetMapping annotation is a specialized version of @RequestMapping annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.GET).

12. **@PostMapping**: The @PostMapping is specialized version of @RequestMapping annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.POST). The @PostMapping annotated methods in the @Controller annotated classes handle the HTTP POST requests matched with given URI expression.

13**. @DeleteMapping**: annotation maps HTTP DELETE requests onto specific handler methods. It is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.DELETE).

14. ***@RequestBody:*** annotation maps the *HttpRequest* body to a transfer or domain object, enabling automatic deserialization of the inbound HTTP Request *body onto a Java object.*

15**. @OneToMany**: A one-to-many relationship between two entities is defined by using the @OneToMany annotation in Spring Data JPA. It declares the mapped By element to indicate the entity that owns the bidirectional relationship. Usually, the child entity is one that owns the relationship and the parent entity contains the @OneToMany annotation.

16**. @generatedValue**: Marking a field with the @GeneratedValue annotation specifies that a value will be automatically generated for that field. This is primarily intended for primary key fields but Object DB also supports this annotation for non-key numeric persistent fields as well.

17**. @SequenceGenerator**: The annotation defines a primary key generator that may be referenced by name when a generator element is specified for the GeneratedValue annotation.

18**.@JoinColumn**: Used to combine the foreign key column on this table.

19**.@SpringBootApplication**: Annotation is used to mark a configuration class that declares one or more @Bean methods and also triggers auto-configuration and component scanning. It's same as declaring a class with @Configuration, @EnableAutoConfiguration and

20**.@ComponentScan annotation**: is used with the @Configuration annotation to tell Spring the packages to scan for annotated components. @ComponentScan also used to specify base packages and base package classes using thebasePackageClasses or basePackages attributes of @ComponentScan·

21.@ResponseStatus: marks a method or exception class with the status code and reason message that should be returned. The status code is applied to the http response when the handler method is invoked, or whenever specified exception is thrown.

22.@ExceptionHandler: The annotation is used to annotate the method(s) in the controller class for handling the exceptions raised during the execution of the controller methods.

23.@Column: Annotation is used to change the table name on database.

24.@PathVariable: Spring annotation which indicates that a method parameter should be bound to a URI template variable. It has the following optional elements: name - name of the path variable to bind to. required - tells whether the path variable is required. value - alias for name.

APPENDIX

Source Code

(inventorymanagementsystemprojectapplication)

default package class:

```
package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class InventoryManagementSystemProjectApplication {

public static void main(String[] args) {

SpringApplication.run(InventoryManagementSystemProjectApplication.class, args);

}}
```

controller code:

(customer controller)

```
package com.example.demo.controller;

/**

 * In this customer controller Some annotations are used
 * @RestController->used for defining the class is controller
 * @Autowired->used for object injection
 * @GetMapping->used for select operation in postman
 * @PutMapping->used for update operation in postman
 * @DeleteMapping->used for delete operation in postman
 */
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;
import com.example.demo.entity.Customer;
import com.example.demo.error.CustomerNotFoundException;
import com.example.demo.service.CustomerService;
@RestController
public class CustomerController {
@Autowired
CustomerService customerService;
@GetMapping("/customer/selectbyid/{id}")
public Customer getCustomers(@PathVariable ("id") int id) throws CustomerNotFoundException
{
return customerService.getCustomers(id);
}
@GetMapping("/customer/selectall/")
public List<Customer> selectAllProduct()            {

return customerService .selectAllCustomers();

}@PutMapping("/customer/updatebyid/{cid}")

public Customer updateProductById(@PathVariable ("cid") int id, @RequestBody Customer
customer) throws CustomerNotFoundException {

return  customerService .updateCustomerById(id,customer);

}@DeleteMapping("/customer/deletebyid/{cid}")

public   String deleteCustomerById(@PathVariable ("cid") int cid)
throwsCustomerNotFoundException{

customerService.deleteCustomerById(cid);

 return "Product is deleted successfully";}}
```

controller code:

(shopkeeper controller)

```
package com.example.demo.controller;

/**

 * In this customer controller Some annotations are used

 * @RestController->used for defining the class is controller

 * @Autowired->used for object injection in postman

 * @PostMapping->used for insert operation in postman

 * @GetMapping->used for select operation in postman

 * @PutMapping->used for update operation in postman

 * @DeleteMapping->used for delete operation in postman

 */

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.DeleteMapping;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.PutMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RestController;

import com.example.demo.entity.ShopKeeper;

import com.example.demo.error.ProductNotFoundException;

import com.example.demo.service.ShopKeeperService;

@RestController

public class ShopKeeperController {

@Autowired

ShopKeeperService shopKeeperService;

@PostMapping("/product/insertion/")

public  ShopKeeper insertProduct(@RequestBody ShopKeeper shopkeeper){

return  shopKeeperService.insertProduct(shopkeeper);

}@GetMapping("/product/selectall/")

public List<ShopKeeper> selectAllProduct(){

return shopKeeperService.selectAllProduct();

}

@GetMapping("/product/selectbyid/{pid}")

public   ShopKeeper selectProductById(@PathVariable ("pid") int pid)
throwsProductNotFoundException{

return shopKeeperService.selectProductById(pid);

}//select by name

@GetMapping("/product/selectbyname/{pname}")

public   ShopKeeper selectProductByName(@PathVariable ("pname") String pname){
```

```
return shopKeeperService.findByproductName(pname);}

//select by type

@GetMapping("/product/selectbytype/{ptype}")

public   ShopKeeper selectProductByType(@PathVariable ("ptype") String ptype){

return shopKeeperService.findByproductType(ptype);

}

@GetMapping("/product/selectbybrand/{pbrand}")

public   ShopKeeper selectProductByBrand(@PathVariable ("pbrand") String pbrand){

return shopKeeperService.findByproductBrand(pbrand);

}

@GetMapping("/product/selectbyquantity/{pquantity}")

public   ShopKeeper selectProductByQuantity(@PathVariable ("pquantity") int pquantity){

return shopKeeperService.findByproductQuantity(pquantity);

}

@GetMapping("/product/selectbyprice/{pprice}")

public   ShopKeeper selectProductByPrice(@PathVariable ("pprice") double pprice)          {

return shopKeeperService.findByproductPrice(pprice);}

@PutMapping("/product/updatebyid/{id}")

public ShopKeeper updateProductById(@PathVariable ("id") int id, @RequestBody
ShopKeeper shopkeeper) throws ProductNotFoundException{

return  shopKeeperService.updateProductById(id,shopkeeper);

}

@DeleteMapping("/product/deletebyid/{pid}")

public   String deleteProductById(@PathVariable ("pid") int pid) throws
ProductNotFoundException{

shopKeeperService.deleteProductById(pid);

return "Product is deleted successfully";
```

```
}}}
```

Entity code:

(customer entity class)

```
package com.example.demo.entity;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.validation.constraints.NotBlank;
import org.hibernate.validator.constraints.Length;
/**
 * This customer class is converted as table and the variables are converted into columns
 * Here some annotations are used like entity,id,table,column,generated value...
 * Some of validations also performed not blank,not null,length
 */
//This annotation is used to convert the class into the tables.
@Entity
//This annotation is used to change table name.
@Table(name="Customer_Table")
public class Customer {
//This annotation is used to create the primary key.
@Id
//This annotation is used to create the primary key value as auto generated one.
@GeneratedValue
@Column(name="Customer_Id")
private int customerId;
//This annotation is used to check the message is blank or not if it is blank it will not allow.
@NotBlank(message="Customer name cannot be blank")
@Column(name="Customer_Name")
private String customerName;
//This annotation is used to check the message length if we want only fixed digit values
means then we can use it.
@Length(min=10,max=13,message="Mobile Number must be 10 digits")
@Column(name="Customer_MobileNo")
private String customerMobileNo;
//This annotation also check the message is present or not.
@NotBlank(message="Customer Address cannot be blank ")
@Length(min=3,message="Address alteast have 3 letters")
@Column(name="Customer_Address")
private String customerAddress;
//constructor without arguments
public Customer() {
super();
}
//constructor with arguments
public Customer(int customerId, String customerName,
String customerMobileNo,
String customerAddress) {
super();
this.customerId = customerId;
this.customerName = customerName;
this.customerMobileNo = customerMobileNo;
this.customerAddress = customerAddress;
}
//getter and setter methods
public int getCustomerId() {
return customerId;
}
public void setCustomerId(int customerId) {
this.customerId = customerId;
}           public String getCustomerName() {
return customerName;
}
public void setCustomerName(String customerName) {
this.customerName = customerName;
}
public String getCustomerMobileNo() {
return customerMobileNo;
}
 public void setCustomerMobileNo(String customerMobileNo) {
this.customerMobileNo = customerMobileNo;
}
public String getCustomerAddress() {
return customerAddress;
}
public void setCustomerAddress(String customerAddress) {
this.customerAddress = customerAddress;
}
//toString method to display the result in the output screen
@Override
public String toString() {
return "Customer [customerId=" + customerId + ", customerName=" + customerName + ",
customerMobileNo="+ customerMobileNo + ", customerAddress=" + customerAddress + "]";
}}
```

Entity code:

(shopkeeper entity class)

```
package com.example.demo.entity;
```

```
import java.util.List;

import javax.persistence.CascadeType;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.OneToMany;

import javax.persistence.SequenceGenerator;

import javax.persistence.Table;

import javax.validation.constraints.Min;

import javax.validation.constraints.NotBlank;

import javax.validation.constraints.NotNull;

/**

 * This shopkeeper class is converted as table and the variables are converted into columns

 * Here some annotations are used like entity,id,table,column,generated value...

 * Some of validations also performed not blank,not null,length

 *

 * Mainly the relation is created by using @OneToMany relation.

 */

//This annotation is used to convert the class into the tables.

@Entity

//This annotation is used to change table name.

@Table(name="Shopkeeper_Table")

public class ShopKeeper {
```

```
//This annotation is used to create the primary key.

@Id

//This annotation is used to create the primary key value as auto generated one.

@GeneratedValue(generator="seq",strategy = GenerationType.AUTO)

@SequenceGenerator(name = "seq",initialValue = 1000)

@Column(name="Product_Id")

private int productId;

//This annotation is used to check the message is blank or not if it is blank it will not allow.

@NotBlank(message="Product name cannot be blank")

@Column(name="ProductName")

private String productName;

@NotBlank(message="Product type cannot be blank ")

@Column(name="Product_Type")

private String productType;

//This annotation also check the message is present or not.

@NotNull(message="Product brand cannot be blank")

@Column(name="Product_Brand")

private String productBrand;

//This annotation we are declaring to check atleast single value

@Min(value=1)

@Column(name="Product_Quantity")

private int productQuantity;

@Column(name="Product_Price")

private double productPrice;

public int currentstock;

public  double totalamount;
```

```
//constructor without arguments
public ShopKeeper() {
super();
}
//constructor with arguments
public ShopKeeper(int productId,String productName,
String productType,
String productBrand, @Min(1) int productQuantity,
double productPrice) {
super();
this.productId = productId;
this.productName = productName;
this.productType = productType;
this.productBrand = productBrand;
this.productQuantity = productQuantity;
this.productPrice = productPrice;
}/**This annotation is create the relationship between two entity classes
 * targetEntity->this specifies which entity we are making relationship to this entity
 * cascade->this handle the parent and child relation*/
@OneToMany(targetEntity = Customer.class,cascade = CascadeType.ALL)
//this annotation eliminate one table and makes one column in the target entity
@JoinColumn(name="product_id")
//creating target entity object
private List<Customer> customer;
//getter and setter methods
public int getProductId() {
```

```
return productId;

}

public void setProductId(int productId) {

this.productId = productId;

}public String getProductName() {

return productName;

}

public void setProductName(String productName) {

this.productName = productName;

}

public String getProductType() {

return productType;

}

public void setProductType(String productType) {this.productType = productType;

}

public String getProductBrand() {

return productBrand;

}

public void setProductBrand(String productBrand) {

this.productBrand = productBrand;

}

public int getProductQuantity() {

return productQuantity;

}

public void setProductQuantity(int productQuantity) {

this.productQuantity = productQuantity;
```

```
}
public double getProductPrice() {
return productPrice;
}
public void setProductPrice(double productPrice) {
this.productPrice =productPrice;
}public List<Customer> getCustomer() {
return customer;
}
public void setCustomer(List<Customer> customer) {
this.customer = customer;
}
//toString method
@Override
public String toString() {
return "ShopKeeper [productId=" + productId + ", productName=" + productName + ", productType=" + productType+ ", productBrand=" + productBrand + ", productQuantity=" + productQuantity + ", productPrice="+ productPrice + ", currentstock=" + currentstock + ", totalamount=" + totalamount + ", customer="+ customer + "]";
}}
```

Entity Code:

(Error Message Entity)

```
package com.example.demo.entity;
import org.springframework.http.HttpStatus;
/**
 * This class is mainly used for dispaly the message
 * when the error occurs the http status and message is displayed in the output screen
 * http status: What is the status of the execution that is displayed here
 * Message: What message we want that will be displayed here
 */
public class ErrorMessage {
private HttpStatus status;
private String message;
public ErrorMessage() {
super();}
public ErrorMessage(HttpStatus status, String message) {
super();
this.status = status;
this.message = message;
}
public HttpStatus getStatus() {
return status;
}public void setStatus(HttpStatus status) {
this.status = status;
}public String getMessage() {
return message;
}public void setMessage(String message) {
this.message = message;
}@Override
public String toString() {
return "ErrorMessage [status=" + status + ", message=" + message + "]";
}}
```

Error Package:

(customer not found class)

```
package com.example.demo.error;

/**

 * This customer not found exception class is

 * used to display the customer operation serror message in the output screen */

public class CustomerNotFoundException extends Exception {

private static final long serialVersionUID = 1L;

public CustomerNotFoundException(String s){

super(s);

}}
```

ErrorPackage:

(product not found class)

```
package com.example.demo.error;

/**

 * This product not found exception class is

 * used to display the product operation error message in the output screen

 *

 */

public class ProductNotFoundException extends Exception {

private static final long serialVersionUID = 1L;

public ProductNotFoundException(String s){

super(s);

}}
```

ErrorPackage:

(rest response class)

```
package com.example.demo.error;

/**

 * The RestResponseEntityExceptionHandler class extends class
ResponseEntityExceptionHandler

 * Here some annotations are used controller advice,response status,exception handler

 */

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.ControllerAdvice;

import org.springframework.web.bind.annotation.ExceptionHandler;

import org.springframework.web.bind.annotation.ResponseStatus;

import org.springframework.web.context.request.WebRequest;

import  org.springframework.web.servlet.mvc.method.annotation.

ResponseEntityExceptionHandler;

import com.example.demo.entity.ErrorMessage;

@ControllerAdvice

@ResponseStatus

public class RestResponseEntityExceptionHandler extends ResponseEntityExceptionHandler{

//here the product exceptions are handling

@ExceptionHandler(ProductNotFoundException.class)

public ResponseEntity<ErrorMessage>
ProductNotFoundException(ProductNotFoundException exception,WebRequest request) {

ErrorMessage message=new
ErrorMessage(HttpStatus.NOT_FOUND,exception.getMessage());//constructor

return ResponseEntity.status(HttpStatus.NOT_FOUND).body(message);

}

//here the customer exceptions are handling
@ExceptionHandler(CustomerNotFoundException.class)

public
ResponseEntity<ErrorMessage>CustomerNotFoundException(CustomerNotFoundException
exception,WebRequest request) {

ErrorMessage message=new
ErrorMessage(HttpStatus.NOT_FOUND,exception.getMessage());//constructor

return ResponseEntity.status(HttpStatus.NOT_FOUND).body(message);

}}
```

Repository:

(customer repository)

```
package com.example.demo.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import com.example.demo.entity.Customer;

/**

 * The customer repository interface is extending the JpaRepository for

 * use the predefined methods.

 */

@Repository

public interface CustomerRepository extends JpaRepository<Customer, Integer>{

}
```

Respository:

(shopkeeper repository)

```
package com.example.demo.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import com.example.demo.entity.ShopKeeper;
```

```
/**
 * The shopkeeper repository interface is extending the JpaRepository for
 * use the predefined methods.
 *
 * This is the entry point for custom methods
 */
@Repository
public interface ShopKeeperRepository extends JpaRepository<ShopKeeper, Integer> {
//select product using product name
shopKeeper findByproductName(String pname);
//select product using product type
ShopKeeper findByproductType(String ptype);
//select product using product brand
ShopKeeper findByproductBrand(String pbrand);
//select product using product quantity
ShopKeeper findByproductQuantity(int pquantity);
//select product using product price
ShopKeeper findByproductPrice(double pprice);
}
```

Service:

(customer service)

```
package com.example.demo.service;
import java.util.List;
import com.example.demo.entity.Customer;
import com.example.demo.error.CustomerNotFoundException;
/**
```

```
 * This customer interface is used for create the abstract methods
 *
 */

public interface CustomerService {
 //select customers
 List<Customer> selectAllCustomers();
 // select customer using id
Customer getCustomers(int id) throws CustomerNotFoundException;
 // delete customer using id
void deleteCustomerById(int cid) throws CustomerNotFoundException;
// update customer using id
Customer updateCustomerById(int id, Customer customer) throws
CustomerNotFoundException;
}
```

Service:

(shopkeeper service)

```
package com.example.demo.service;
import java.util.List;
import com.example.demo.entity.ShopKeeper;
import com.example.demo.error.ProductNotFoundException;
/**
 * This service interface is used for create the abstract methods
 *
 */
public interface ShopKeeperService {   //insert product
ShopKeeper insertProduct(ShopKeeper shopkeeper);

//select product using id

ShopKeeper selectProductById(int pid) throws ProductNotFoundException;

//delete product using id

void deleteProductById(int pid) throws ProductNotFoundException;

//update product using id

ShopKeeper updateProductById(int id, ShopKeeper shopkeeper) throws
ProductNotFoundException;//select all products

List<ShopKeeper> selectAllProduct();

//select product using name

ShopKeeper findByproductName(String pname);

//select product using type

ShopKeeper findByproductType(String ptype);

//select product using brand

ShopKeeper findByproductBrand(String pbrand);

//select product using quantity

ShopKeeper findByproductQuantity(int pquantity);

//select product using price

ShopKeeper findByproductPrice(double pprice);

}
```

Service:

(customer service implemented class)

```
package com.example.demo.service;

import java.util.List;

import java.util.Objects;

import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.example.demo.entity.Customer;

import com.example.demo.error.CustomerNotFoundException;

import com.example.demo.repository.CustomerRepository;

@Service

public class CustomerServiceImpl implements CustomerService{

@Autowired

CustomerRepository customerRepo;

//here the customer select operation select is overridden

@Override

public List<Customer> selectAllCustomers() {

return customerRepo.findAll();

}

//here the select customer operation using id is overridden

@Override

public Customer getCustomers(int id) throws CustomerNotFoundException {

Optional<Customer> customer1= customerRepo.findById(id);

if(!customer1.isPresent()){

throw new CustomerNotFoundException("Customer id not exists");

}

return customerRepo.getById(id);

}

 //here the customer delete operation using id is overridden

@Override

public void deleteCustomerById(int cid) throws CustomerNotFoundException {
```

```
Optional<Customer> customer1= customerRepo.findById(cid);

if(!customer1.isPresent()){

throw new CustomerNotFoundException("Customer id not exists");

}

customerRepo.deleteById(cid);

}

//here the customer update operation using id is overridden

@Override

public Customer updateCustomerById(int id, Customer customer) throws
CustomerNotFoundException {

Optional<Customer> shopkeeper1= customerRepo.findById(id);

Customer customerDB=null;

if(shopkeeper1.isPresent()) {

customerDB=  customerRepo.findById(id).get();

if(Objects.nonNull(customer.getCustomerName()) &&
!"".equalsIgnoreCase(customer.getCustomerName())) {

customerDB.setCustomerName((customer.getCustomerName()));

}

if(Objects.nonNull(customer.getCustomerMobileNo()) &&
!"".equalsIgnoreCase(customer.getCustomerMobileNo())) {

customerDB.setCustomerMobileNo(customer.getCustomerMobileNo());

System.out.println(customer.getCustomerMobileNo());

}

if(Objects.nonNull(customer.getCustomerAddress())
&&!"".equalsIgnoreCase(customer.getCustomerAddress())) {

customerDB.setCustomerAddress(customer.getCustomerAddress());

System.out.println(customer.getCustomerAddress());
}}else{
throw new CustomerNotFoundException("Customer id not exists");
}
return customerRepo.save(customerDB);
}}
```

Service:

(Shopkeeper service implemented class)

```
package com.example.demo.service;
import java.util.List;
import java.util.Objects;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.example.demo.entity.ShopKeeper;
import com.example.demo.error.ProductNotFoundException;
import com.example.demo.repository.ShopKeeperRepository;
@Service
public class ShopKeeperServiceImpl implements ShopKeeperService{
@Autowired
ShopKeeperRepository shopKeeperRepository;
//here the shopkeeper insertion operation is overridden
@Override
public ShopKeeper insertProduct(ShopKeeper shopkeeper) {
 int soapStock=100;int pasteStock=80;int biscutsStock=200;
int chocolateStock=100;
int coolDrinksStock=200;
```

```
if(shopkeeper.getProductName().equalsIgnoreCase("dove") &&
soapStock>shopkeeper.getProductQuantity()){

shopkeeper.currentstock=soapStock-shopkeeper.getProductQuantity();

shopkeeper.totalamount=shopkeeper.getProductPrice()*shopkeeper.getProductQuantity();

shopKeeperRepository.save(shopkeeper);

//soapStock=shopkeeper.currentstock;

//shopkeeper.currentstock=0;

}

else if(shopkeeper.getProductName().equalsIgnoreCase("colgate") &&
pasteStock>shopkeeper.getProductQuantity()){

shopkeeper.currentstock=pasteStock-shopkeeper.getProductQuantity();

shopkeeper.totalamount=shopkeeper.getProductPrice()*shopkeeper.getProductQuantity();

shopKeeperRepository.save(shopkeeper);

}

else if(shopkeeper.getProductName().equalsIgnoreCase("jimjam") &&
pasteStock>shopkeeper.getProductQuantity()){

shopkeeper.currentstock=biscutsStock-shopkeeper.getProductQuantity();

shopkeeper.totalamount=shopkeeper.getProductPrice()*shopkeeper.getProductQuantity();

shopKeeperRepository.save(shopkeeper);

}

else if(shopkeeper.getProductName().equalsIgnoreCase("munch") &&
pasteStock>shopkeeper.getProductQuantity()){

shopkeeper.currentstock=chocolateStock-shopkeeper.getProductQuantity();

shopkeeper.totalamount=shopkeeper.getProductPrice()*shopkeeper.getProductQuantity();

shopKeeperRepository.save(shopkeeper);

}

else if(shopkeeper.getProductName().equalsIgnoreCase("fanta") &&
pasteStock>shopkeeper.getProductQuantity()){
```

```
shopkeeper.currentstock=coolDrinksStock-shopkeeper.getProductQuantity();
shopkeeper.totalamount=shopkeeper.getProductPrice()*shopkeeper.getProductQuantity();
shopKeeperRepository.save(shopkeeper);
}
return shopkeeper;
}//here the shopkeeper select all operation is overridden
@Override
public List<ShopKeeper> selectAllProduct() {
return shopKeeperRepository.findAll();
}//here the shopkeeper select using id operation is overridden
@Override
public ShopKeeper selectProductById(int pid) throws ProductNotFoundException {
Optional<ShopKeeper> shopkeeper1= shopKeeperRepository.findById(pid);
if(!shopkeeper1.isPresent()){
throw new ProductNotFoundException("Product id not exists");
}return shopKeeperRepository.findById(pid).get();
}//here the shopkeeper insertion operation is overridden
@Override
public void deleteProductById(int pid) throws ProductNotFoundException {
Optional<ShopKeeper> shopkeeper1= shopKeeperRepository.findById(pid);
if(!shopkeeper1.isPresent()){
throw new ProductNotFoundException("Product id not exists");
}shopKeeperRepository.deleteById(pid);


}//here the shopkeeper update operation is overridden
@SuppressWarnings("unlikely-arg-type")
```

```
@Override

public ShopKeeper updateProductById(int id, ShopKeeper shopkeeper) throws
ProductNotFoundException {

optional<ShopKeeper> shopkeeper1= shopKeeperRepository.findById(id);

ShopKeeper shopkeeperDB=null;

if(shopkeeper1.isPresent()) {

shopkeeperDB=          shopKeeperRepository.findById(id).get();

if(Objects.nonNull(shopkeeper.getProductName()) &&
!"".equalsIgnoreCase(shopkeeper.getProductName())) {

shopkeeperDB.setProductName((shopkeeper.getProductName()))

}if(Objects.nonNull(shopkeeper.getProductType()) &&
!"".equalsIgnoreCase(shopkeeper.getProductType())) {

shopkeeperDB.setProductType(shopkeeper.getProductType());

System.out.println(shopkeeper.getProductType());

}if(Objects.nonNull(shopkeeper.getProductBrand()) &&
!"".equalsIgnoreCase(shopkeeper.getProductBrand())) {

shopkeeperDB.setProductBrand(shopkeeper.getProductBrand());

System.out.println(shopkeeper.getProductBrand());

}

if(Objects.nonNull(shopkeeper.getProductQuantity()) &&
!"".equals(shopkeeper.getProductQuantity())) {

shopkeeperDB.setProductQuantity(shopkeeper.getProductQuantity());

System.out.println(shopkeeper.getProductQuantity());

}if(Objects.nonNull(shopkeeper.getProductPrice()) &&
!"".equals(shopkeeper.getProductPrice())) {

shopkeeperDB.setProductPrice(shopkeeper.getProductQuantity());

System.out.println(shopkeeper.getProductPrice());

}}else{
```

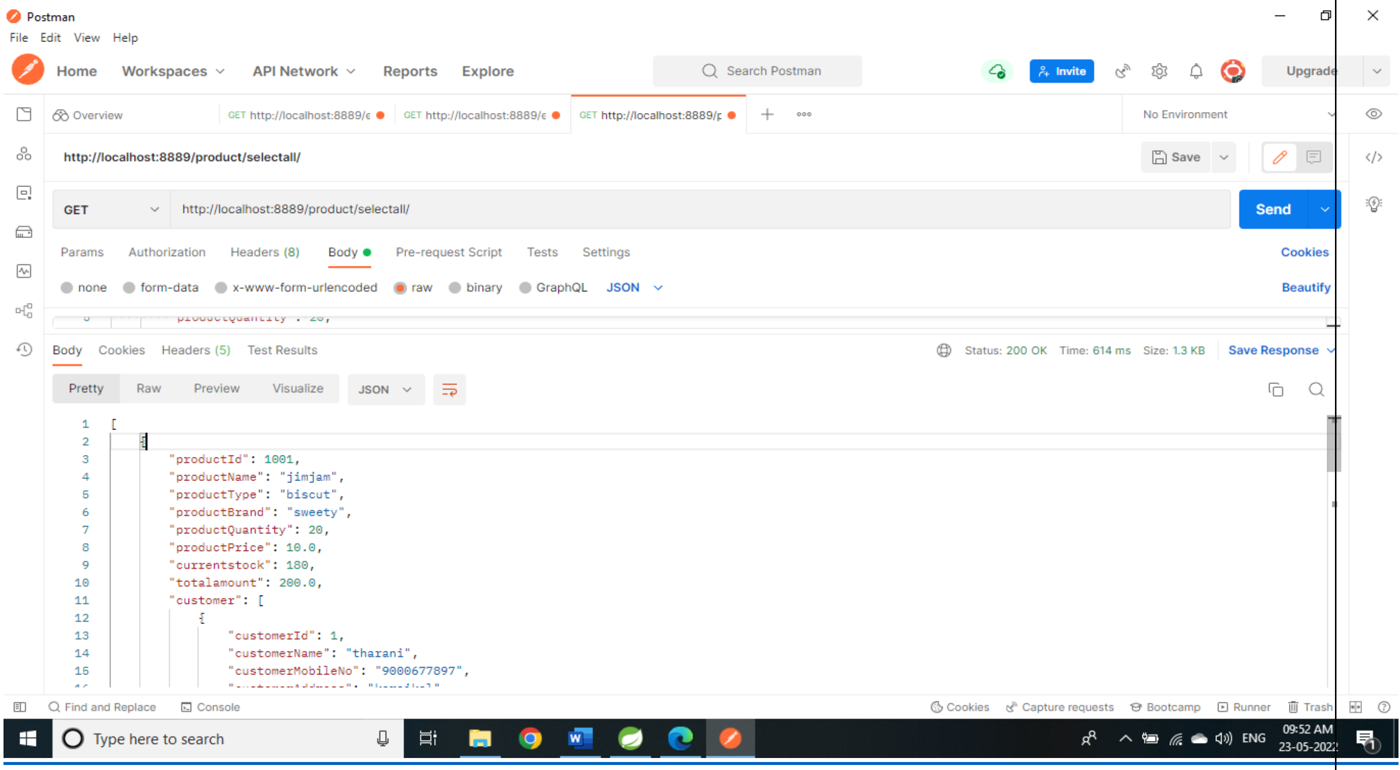
```
throw new ProductNotFoundException("Product id not exists");

}return shopKeeperRepository.save(shopkeeperDB);

}//here the shopkeeper select product using name is overridden

@Override

public ShopKeeper findByproductName(String pname) {

return shopKeeperRepository.findByproductName(pname);

}//here the shopkeeper select product using type is overridden

@Override

public ShopKeeper findByproductType(String ptype) {

return shopKeeperRepository.findByproductType(ptype);

}//here the shopkeeper select product using brand is overridden

@Override

public ShopKeeper findByproductBrand(String pbrand) {

return shopKeeperRepository.findByproductBrand(pbrand);

}//here the shopkeeper select product using quantity is overridden

@Override

public ShopKeeper findByproductQuantity(int pquantity) {

// TODO Auto-generated method stub

return shopKeeperRepository.findByproductQuantity(pquantity);

}//here the shopkeeper select product using price is overridden

@Override

public ShopKeeper findByproductPrice(double pprice) {

return

shopKeeperRepository.findByproductPrice(pprice);

}}
```

ApplicationProperties:

```
server.port = 8889

spring.datasource.driver-class-name = com.mysql.cj.jdbc.Driver

spring.datasource.url = jdbc:mysql://localhost:3306/inventory_management_system

spring.datasource.username = root

spring.datasource.password = root

spring.jpa.show-sql = true

spring.jpa.generate-ddl= true

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect

# Hibernate ddl auto property

spring.jpa.hibernate.ddl-auto=create
```

shopkeeper screenshots:

Get: select all products http://localhost:8889/product/selectall/



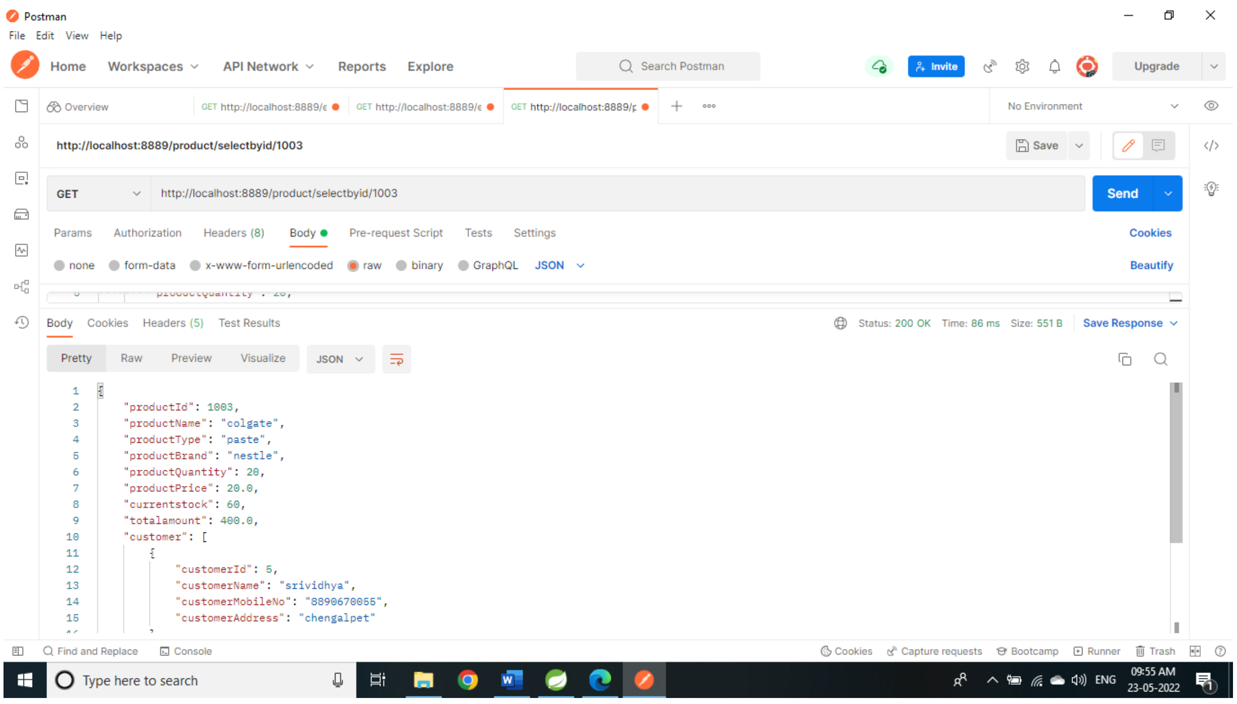
GET http://localhost:8889/product/selectall/

Status: 200 OK Time: 614 ms Size: 1.3 KB

```
[
    {
        "productId": 1001,
        "productName": "jimjam",
        "productType": "biscut",
        "productBrand": "sweety",
        "productQuantity": 20,
        "productPrice": 10.0,
        "currentstock": 180,
        "totalamount": 200.0,
        "customer": [
            {
                "customerId": 1,
                "customerName": "tharani",
                "customerMobileNo": "9000677897",
```

Get: select product by id:

http://localhost:8889/product/selectbyid/1003



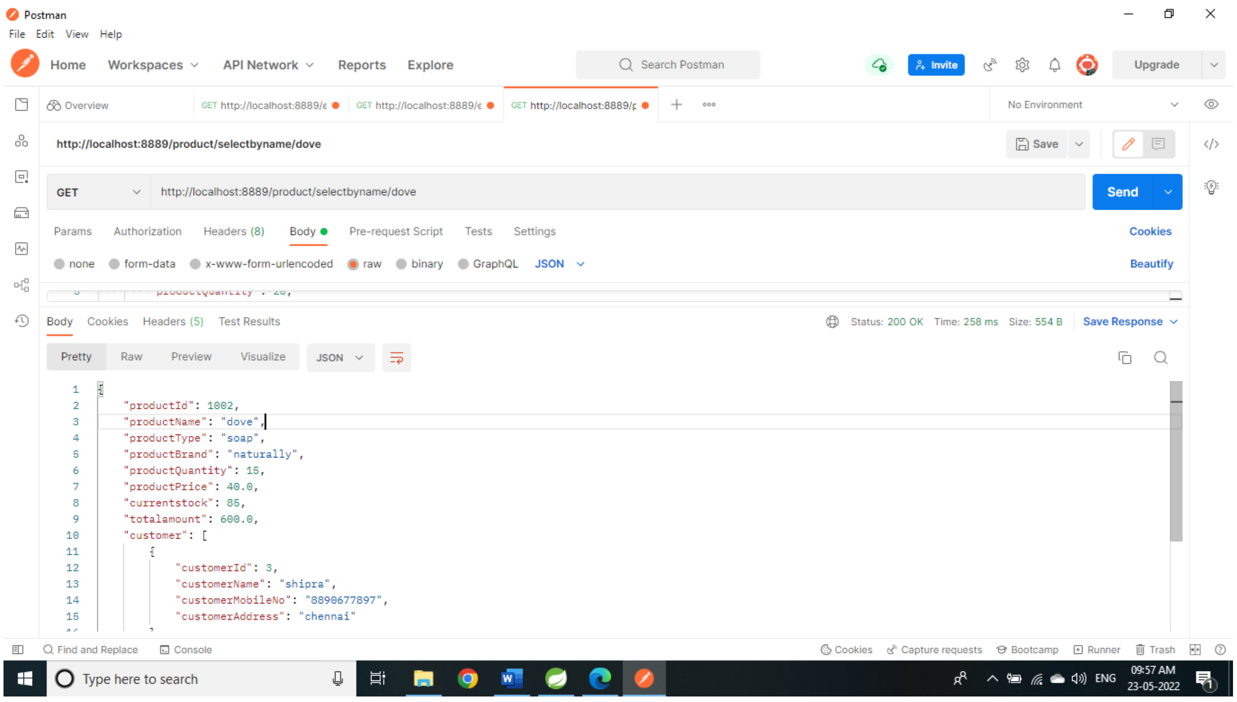
```
GET http://localhost:8889/product/selectbyid/1003
```

Status: 200 OK Time: 86 ms Size: 551 B

```
{
    "productId": 1003,
    "productName": "colgate",
    "productType": "paste",
    "productBrand": "nestle",
    "productQuantity": 20,
    "productPrice": 20.0,
    "currentstock": 60,
    "totalamount": 400.0,
    "customer": [
        {
            "customerId": 5,
            "customerName": "srividhya",
            "customerMobileNo": "8890670055",
            "customerAddress": "chengalpet"
```

Get: select product by name

http://localhost:8889/product/selectbyname/dove



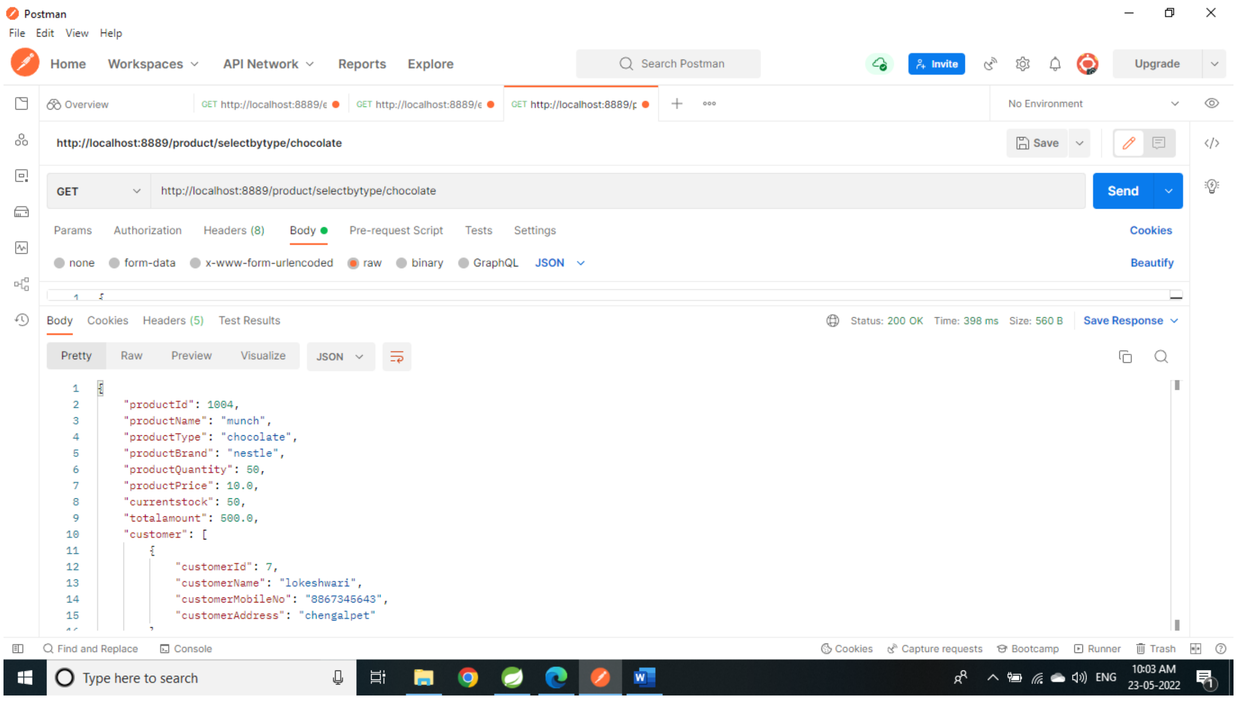
```
GET http://localhost:8889/product/selectbyname/dove
```

Status: 200 OK Time: 258 ms Size: 554 B

```
{
    "productId": 1002,
    "productName": "dove",
    "productType": "soap",
    "productBrand": "naturally",
    "productQuantity": 15,
    "productPrice": 40.0,
    "currentstock": 85,
    "totalamount": 600.0,
    "customer": [
        {
            "customerId": 3,
            "customerName": "shipra",
            "customerMobileNo": "8890677897",
            "customerAddress": "chennai"
```

Select product by type:

Get: http://localhost:8889/product/selectbytype/chocolate



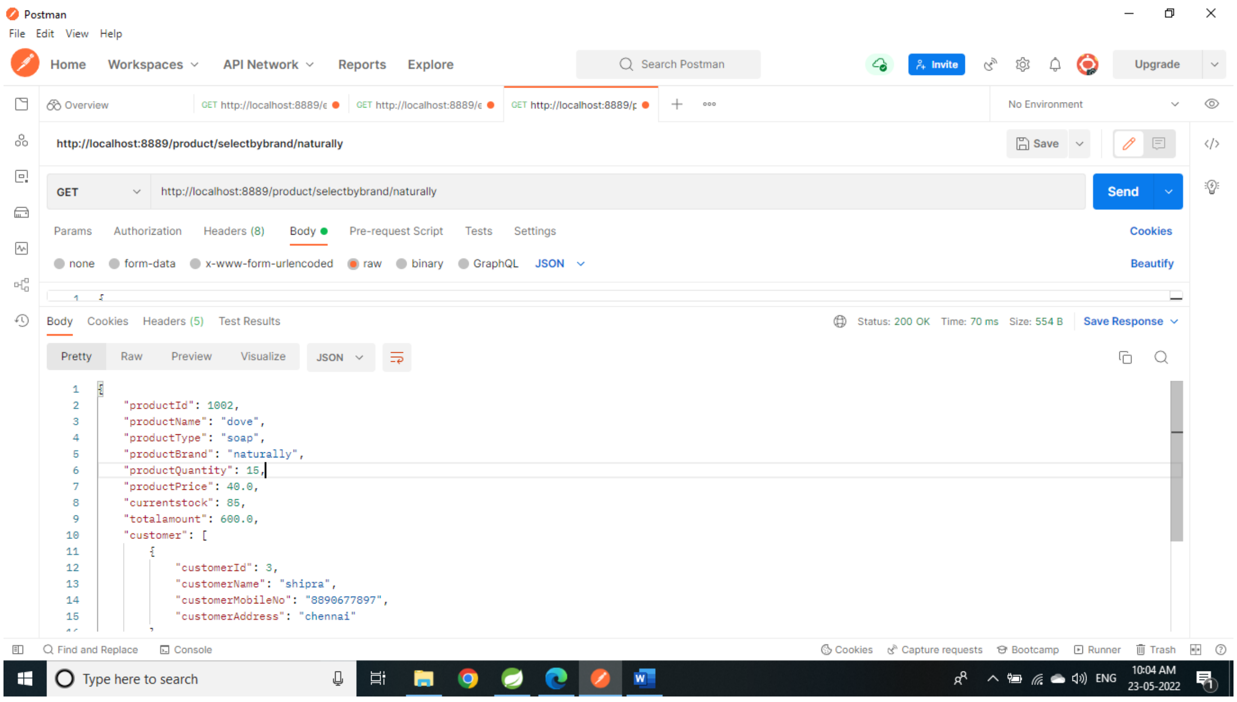
http://localhost:8889/product/selectbytype/chocolate

GET http://localhost:8889/product/selectbytype/chocolate

Status: 200 OK Time: 398 ms Size: 560 B

```
{
    "productId": 1004,
    "productName": "munch",
    "productType": "chocolate",
    "productBrand": "nestle",
    "productQuantity": 50,
    "productPrice": 10.0,
    "currentstock": 50,
    "totalamount": 500.0,
    "customer": [
        {
            "customerId": 7,
            "customerName": "lokeshwari",
            "customerMobileNo": "8867345643",
            "customerAddress": "chengalpet"
```

Select product by brand: http://localhost:8889/product/selectbybrand/naturally



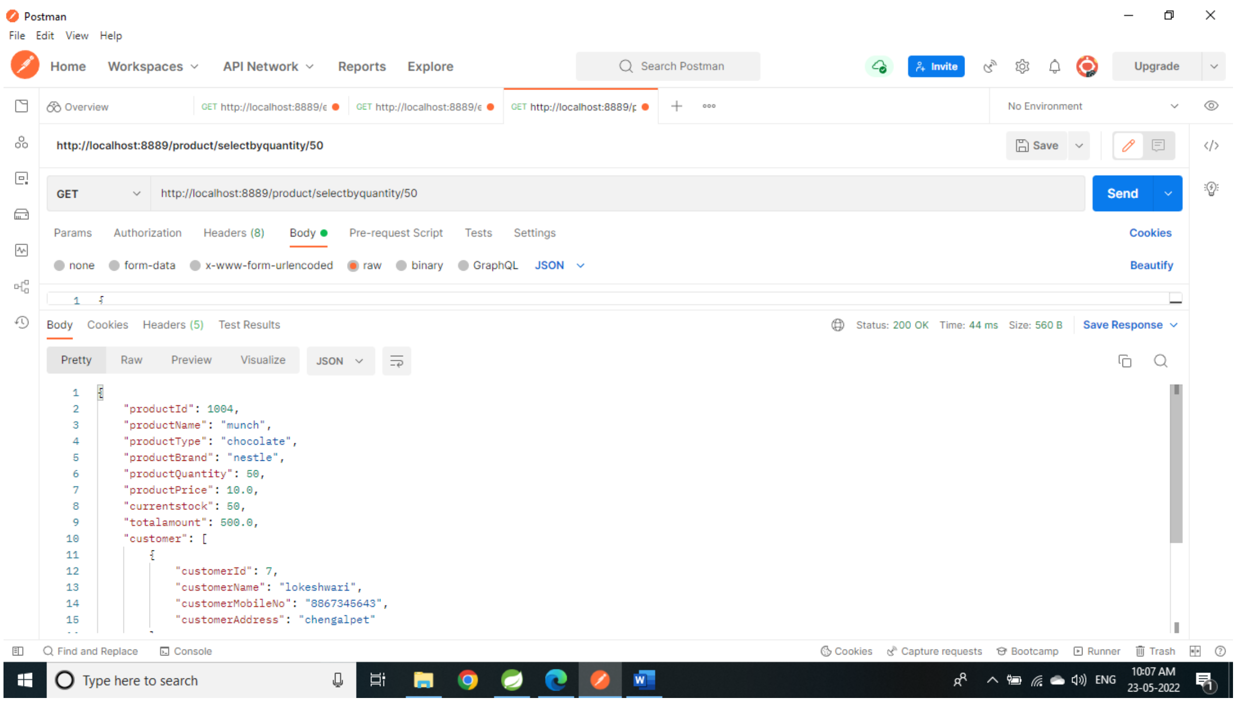
http://localhost:8889/product/selectbybrand/naturally

GET http://localhost:8889/product/selectbybrand/naturally

Status: 200 OK Time: 70 ms Size: 554 B

```
{
    "productId": 1002,
    "productName": "dove",
    "productType": "soap",
    "productBrand": "naturally",
    "productQuantity": 15,
    "productPrice": 40.0,
    "currentstock": 85,
    "totalamount": 600.0,
    "customer": [
        {
            "customerId": 3,
            "customerName": "shipra",
            "customerMobileNo": "8890677897",
            "customerAddress": "chennai"
```

Select product by quantity: http://localhost:8889/product/selectbyquantity/50


Postman
File Edit View Help
Home Workspaces API Network Reports Explore
Search Postman
Invite
Upgrade
Overview
No Environment
Save
Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings
Cookies
Beautify
Pretty Raw Preview Visualize JSON
Save Response
Find and Replace Console
Cookies Capture requests Bootcamp Runner Trash
Type here to search

http://localhost:8889/product/selectbyquantity/50

GET http://localhost:8889/product/selectbyquantity/50

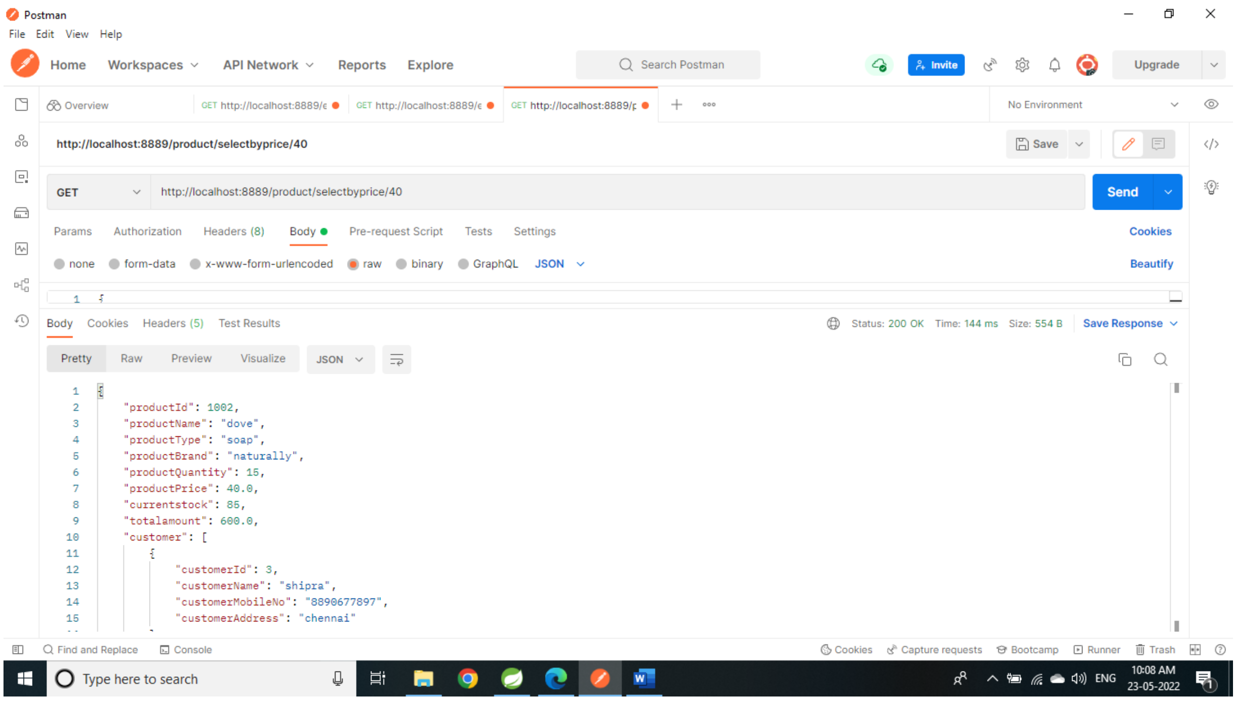
Body: none form-data x-www-form-urlencoded [✓ raw] binary GraphQL JSON

Body Cookies Headers (5) Test Results   Status: 200 OK Time: 44 ms Size: 560 B

```
{
    "productId": 1004,
    "productName": "munch",
    "productType": "chocolate",
    "productBrand": "nestle",
    "productQuantity": 50,
    "productPrice": 10.0,
    "currentstock": 50,
    "totalamount": 500.0,
    "customer": [
        {
            "customerId": 7,
            "customerName": "lokeshwari",
            "customerMobileNo": "8867346643",
            "customerAddress": "chengalpet"
```

10:07 AM 23-05-2022

Select by price: http://localhost:8889/product/selectbyprice/40


Postman
File Edit View Help
Home Workspaces API Network Reports Explore
Search Postman
Invite
Upgrade
Overview
No Environment
Save
Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings
Cookies
Beautify
Pretty Raw Preview Visualize JSON
Save Response
Find and Replace Console
Cookies Capture requests Bootcamp Runner Trash
Type here to search

http://localhost:8889/product/selectbyprice/40

GET http://localhost:8889/product/selectbyprice/40

Body: none form-data x-www-form-urlencoded [✓ raw] binary GraphQL JSON

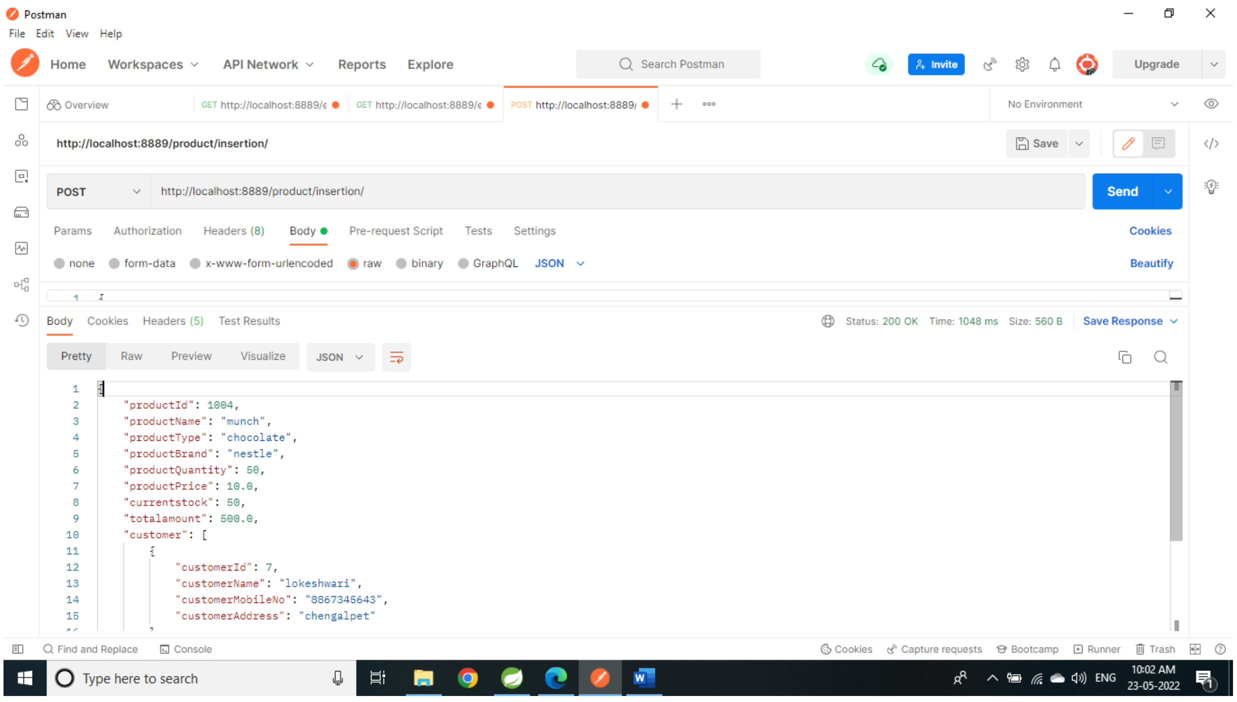
Body Cookies Headers (5) Test Results   Status: 200 OK Time: 144 ms Size: 554 B

```
{
    "productId": 1002,
    "productName": "dove",
    "productType": "soap",
    "productBrand": "naturally",
    "productQuantity": 15,
    "productPrice": 40.0,
    "currentstock": 85,
    "totalamount": 600.0,
    "customer": [
        {
            "customerId": 3,
            "customerName": "shipra",
            "customerMobileNo": "8890677897",
            "customerAddress": "chennai"
```

10:08 AM 23-05-2022

Post: insertion

http://localhost:8889/product/insertion/


Postman
Home Workspaces API Network Reports Explore
Params Authorization Headers (8) Body Pre-request Script Tests Settings
Send

http://localhost:8889/product/insertion/

POST http://localhost:8889/product/insertion/

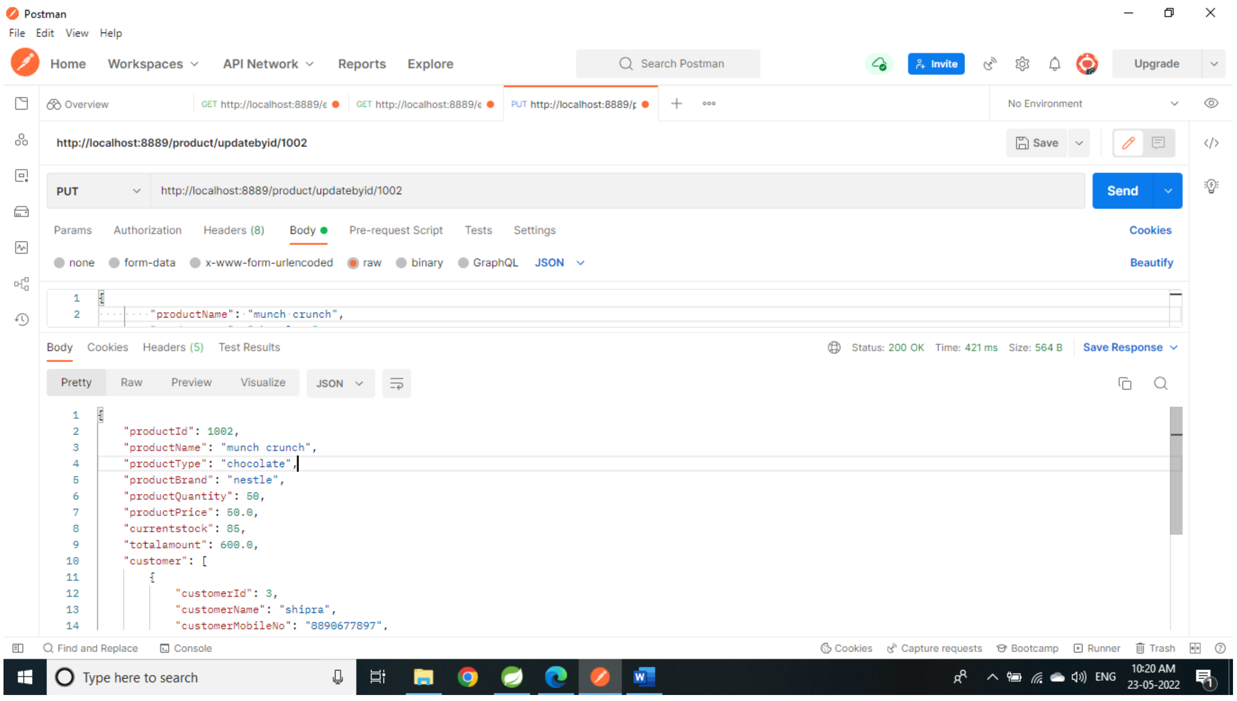
Body: raw JSON

Status: 200 OK Time: 1048 ms Size: 560 B

```
{
    "productId": 1004,
    "productName": "munch",
    "productType": "chocolate",
    "productBrand": "nestle",
    "productQuantity": 50,
    "productPrice": 10.0,
    "currentstock": 50,
    "totalamount": 500.0,
    "customer": [
        {
            "customerId": 7,
            "customerName": "lokeshwari",
            "customerMobileNo": "8867345643",
            "customerAddress": "chengalpet"
```

Update product by id:

Put: http://localhost:8889/product/updatebyid/1002


Postman
Home Workspaces API Network Reports Explore
Params Authorization Headers (8) Body Pre-request Script Tests Settings
Send

http://localhost:8889/product/updatebyid/1002

PUT http://localhost:8889/product/updatebyid/1002

Body: raw JSON

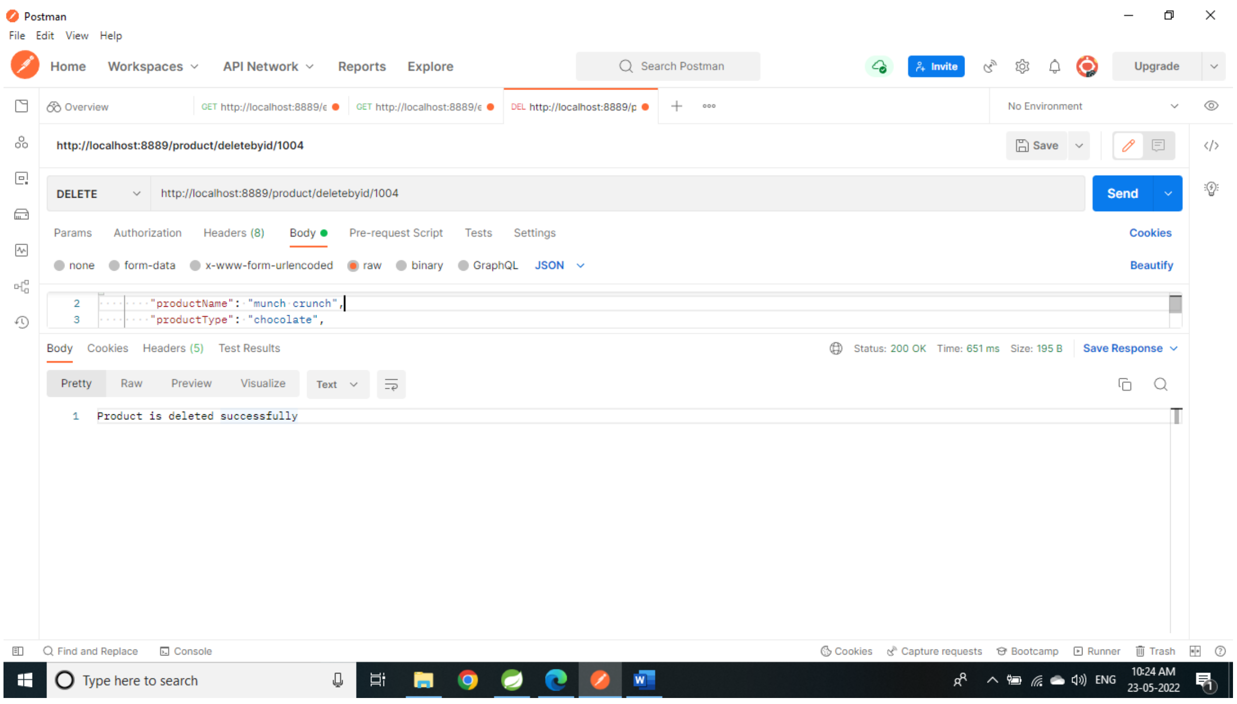
```
{
    "productName": "munch crunch",
```

Status: 200 OK Time: 421 ms Size: 564 B

```
{
    "productId": 1002,
    "productName": "munch crunch",
    "productType": "chocolate",
    "productBrand": "nestle",
    "productQuantity": 50,
    "productPrice": 50.0,
    "currentstock": 85,
    "totalamount": 600.0,
    "customer": [
        {
            "customerId": 3,
            "customerName": "shipra",
            "customerMobileNo": "8890677897",
```

Delete product by id:

http://localhost:8889/product/deletebyid/1004



http://localhost:8889/product/deletebyid/1004

DELETE http://localhost:8889/product/deletebyid/1004

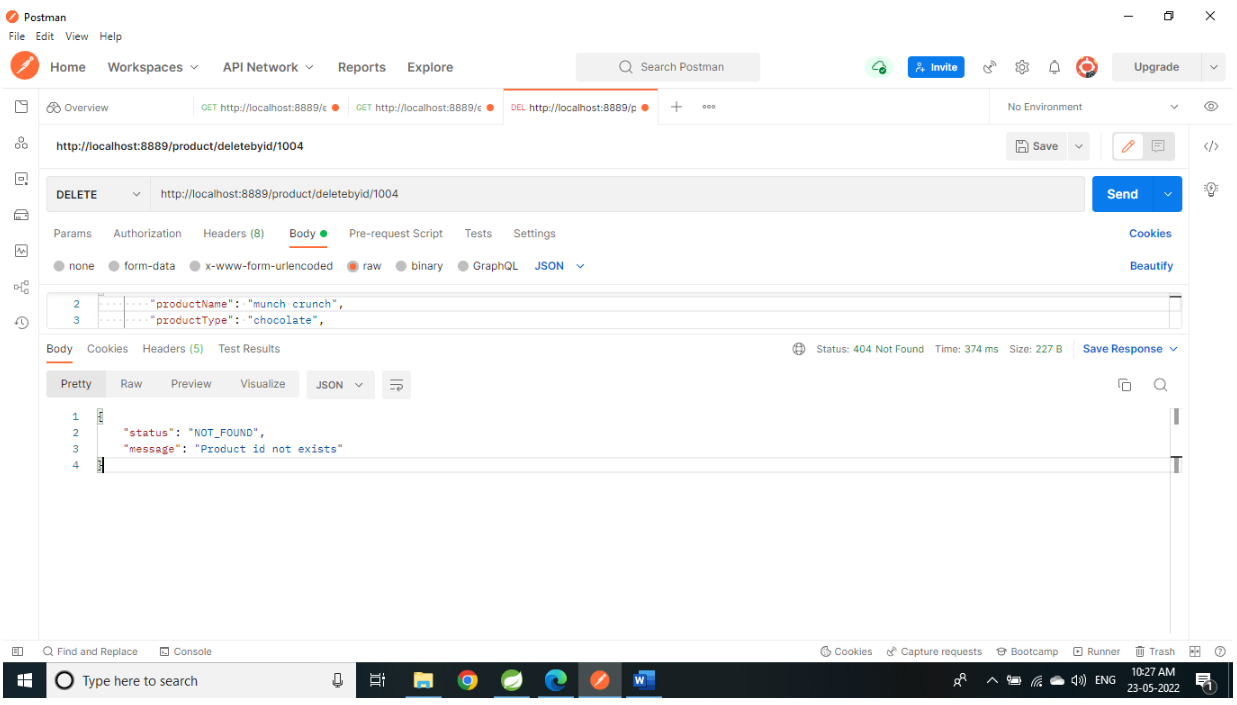
```
"productName": "munch crunch",
"productType": "chocolate",
```

Status: 200 OK Time: 651 ms Size: 195 B

```
Product is deleted successfully
```

Not found display:

Delete: http://localhost:8889/product/deletebyid/1004



http://localhost:8889/product/deletebyid/1004

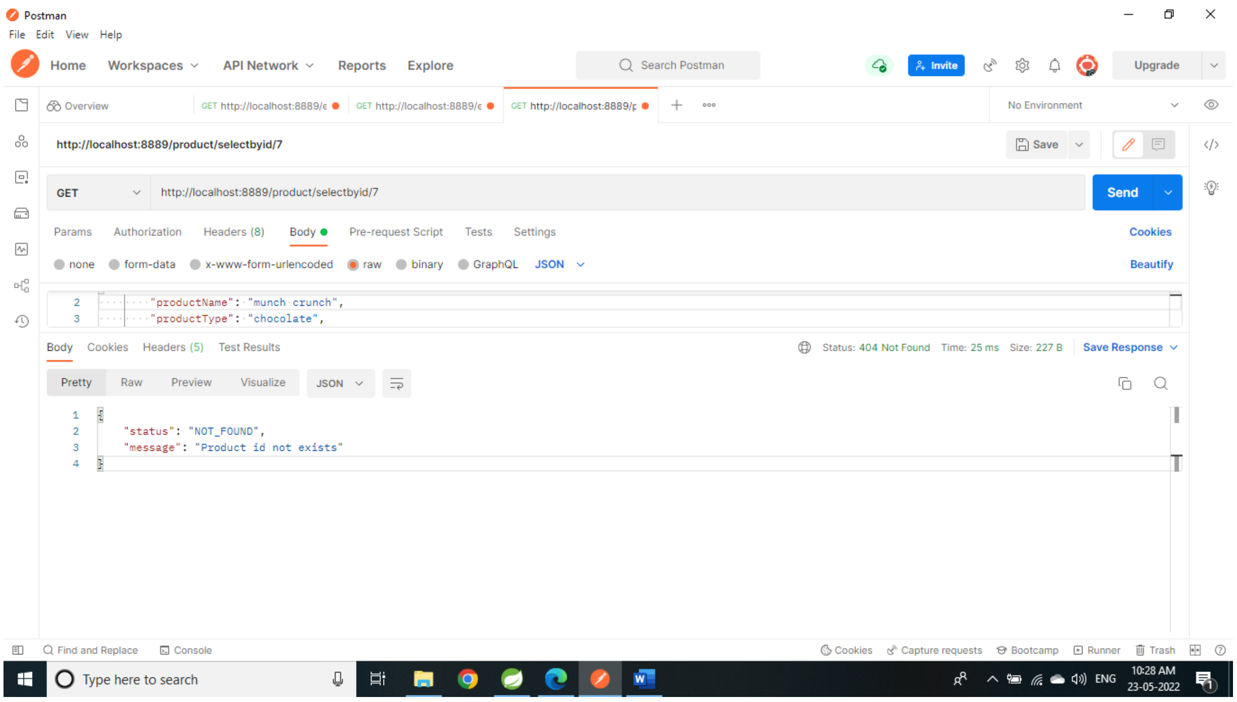
DELETE http://localhost:8889/product/deletebyid/1004

```
"productName": "munch crunch",
"productType": "chocolate",
```

Status: 404 Not Found Time: 374 ms Size: 227 B

```
{
    "status": "NOT_FOUND",
    "message": "Product id not exists"
}
```

Get : http://localhost:8889/product/selectbyid/7


Postman
File Edit View Help
Home Workspaces API Network Reports Explore
Search Postman
Invite
Upgrade
Overview
No Environment
Save
Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
Body Cookies Headers (5) Test Results
Save Response
Pretty Raw Preview Visualize JSON
Find and Replace Console
Cookies Capture requests Bootcamp Runner Trash
Type here to search
ENG
10:28 AM
23-05-2022

http://localhost:8889/product/selectbyid/7

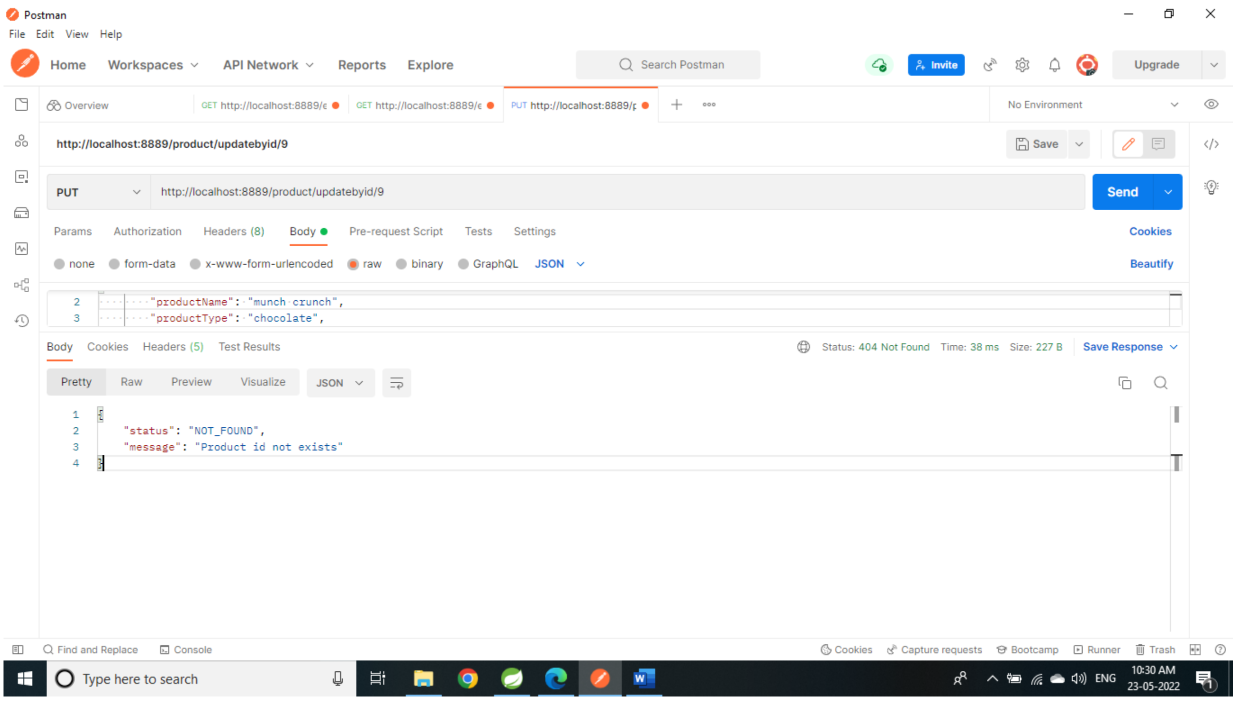
GET http://localhost:8889/product/selectbyid/7

```
    "productName": "munch crunch",
    "productType": "chocolate",
```

Status: 404 Not Found Time: 25 ms Size: 227 B

```
{
    "status": "NOT_FOUND",
    "message": "Product id not exists"
}
```

Update: http://localhost:8889/product/updatebyid/9


Postman
File Edit View Help
Home Workspaces API Network Reports Explore
Search Postman
Invite
Upgrade
Overview
No Environment
Save
Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
Body Cookies Headers (5) Test Results
Save Response
Pretty Raw Preview Visualize JSON
Find and Replace Console
Cookies Capture requests Bootcamp Runner Trash
Type here to search
ENG
10:30 AM
23-05-2022

http://localhost:8889/product/updatebyid/9

PUT http://localhost:8889/product/updatebyid/9

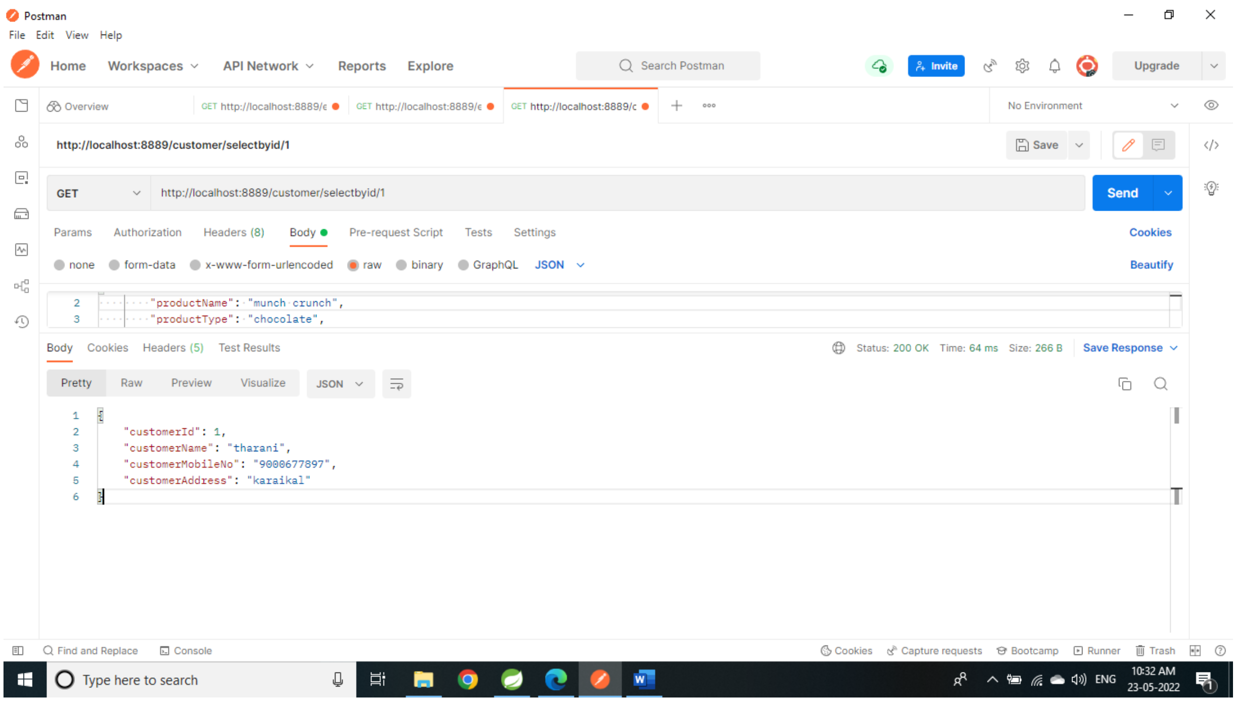
```
    "productName": "munch crunch",
    "productType": "chocolate",
```

Status: 404 Not Found Time: 38 ms Size: 227 B

```
{
    "status": "NOT_FOUND",
    "message": "Product id not exists"
}
```

Customer screen shot:

Get: http://localhost:8889/customer/selectbyid/1


Postman
File Edit View Help
Home Workspaces API Network Reports Explore
Search Postman
Invite Upgrade
Overview
No Environment
Save
Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
Body Cookies Headers (5) Test Results
Save Response
Pretty Raw Preview Visualize JSON
Find and Replace Console
Cookies Capture requests Bootcamp Runner Trash
Type here to search
ENG

http://localhost:8889/customer/selectbyid/1

GET http://localhost:8889/customer/selectbyid/1

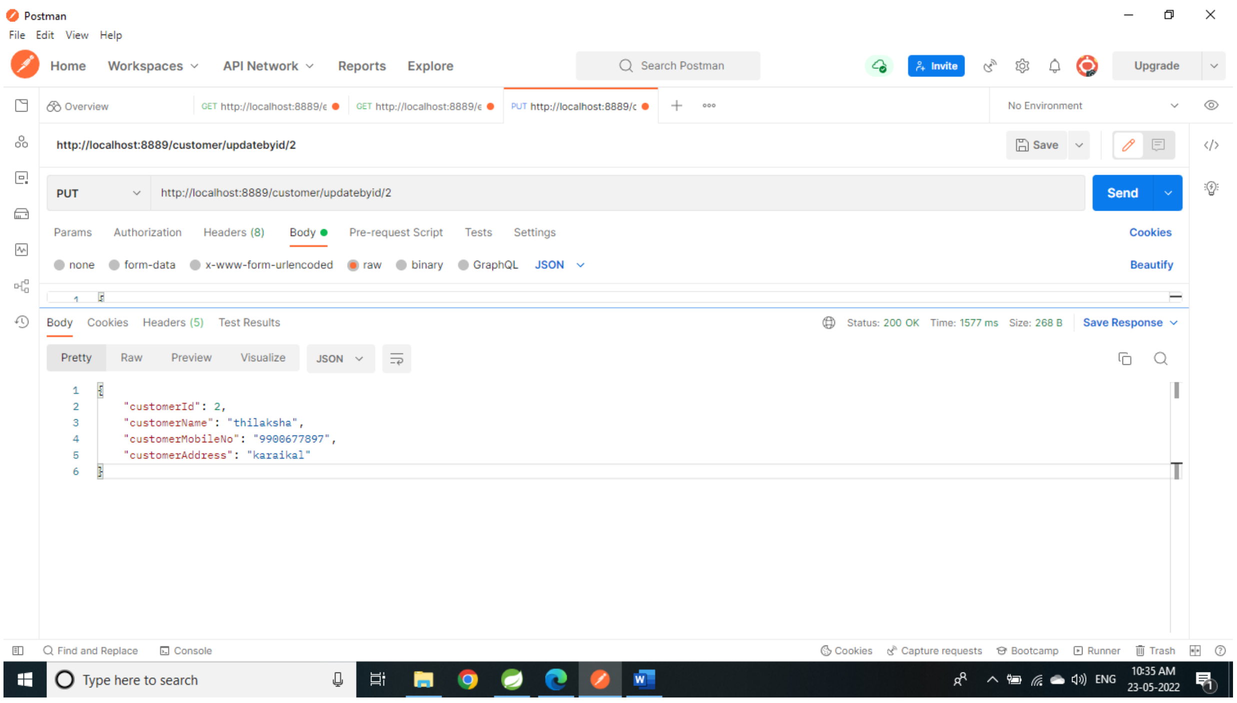
```
    "productName": "munch crunch",
    "productType": "chocolate",
```

Status: 200 OK Time: 64 ms Size: 266 B

```
{
    "customerId": 1,
    "customerName": "tharani",
    "customerMobileNo": "9000677897",
    "customerAddress": "karaikal"
}
```

10:32 AM 23-05-2022

Update by id: http://localhost:8889/customer/updatebyid/2


Postman
File Edit View Help
Home Workspaces API Network Reports Explore
Search Postman
Invite Upgrade
Overview
No Environment
Save
Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
Body Cookies Headers (5) Test Results
Save Response
Pretty Raw Preview Visualize JSON
Find and Replace Console
Cookies Capture requests Bootcamp Runner Trash
Type here to search
ENG

http://localhost:8889/customer/updatebyid/2

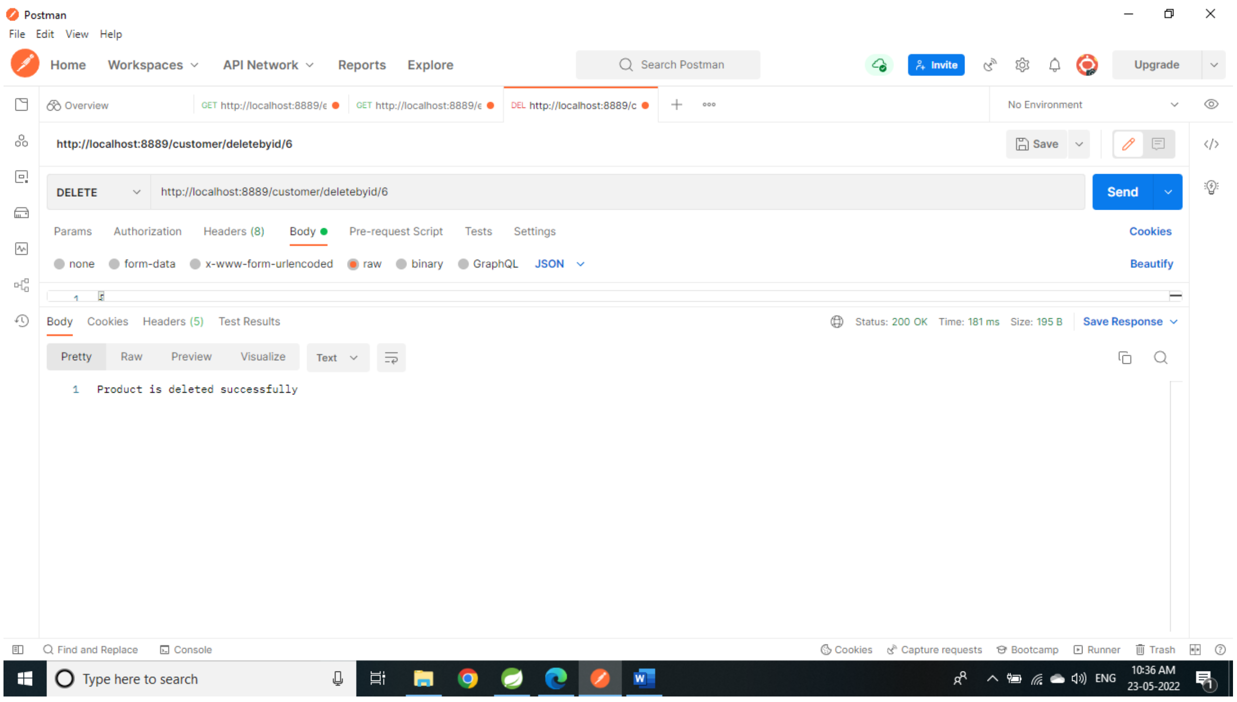
PUT http://localhost:8889/customer/updatebyid/2

Status: 200 OK Time: 1577 ms Size: 268 B

```
{
    "customerId": 2,
    "customerName": "thilaksha",
    "customerMobileNo": "9900677897",
    "customerAddress": "karaikal"
}
```

10:35 AM 23-05-2022

Delete by id: http://localhost:8889/customer/deletebyid/6


Postman
File Edit View Help
Home Workspaces API Network Reports Explore
Search Postman
Invite
Upgrade
Overview
No Environment
Save
Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
Body Cookies Headers (5) Test Results
Save Response
Pretty Raw Preview Visualize Text

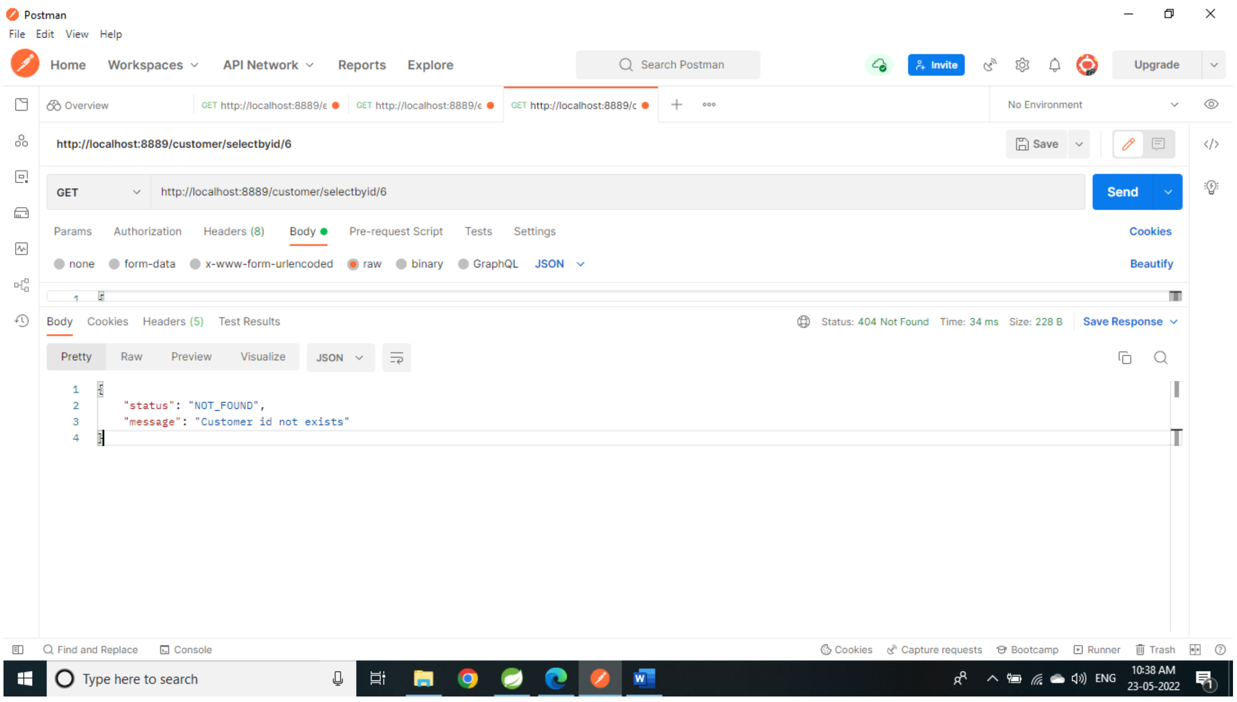
http://localhost:8889/customer/deletebyid/6

DELETE http://localhost:8889/customer/deletebyid/6

Status: 200 OK Time: 181 ms Size: 195 B

```
Product is deleted successfully
```

Select by id: http://localhost:8889/customer/selectbyid/6


Postman
File Edit View Help
Home Workspaces API Network Reports Explore
Search Postman
Invite
Upgrade
Overview
No Environment
Save
Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
Body Cookies Headers (5) Test Results
Save Response
Pretty Raw Preview Visualize JSON

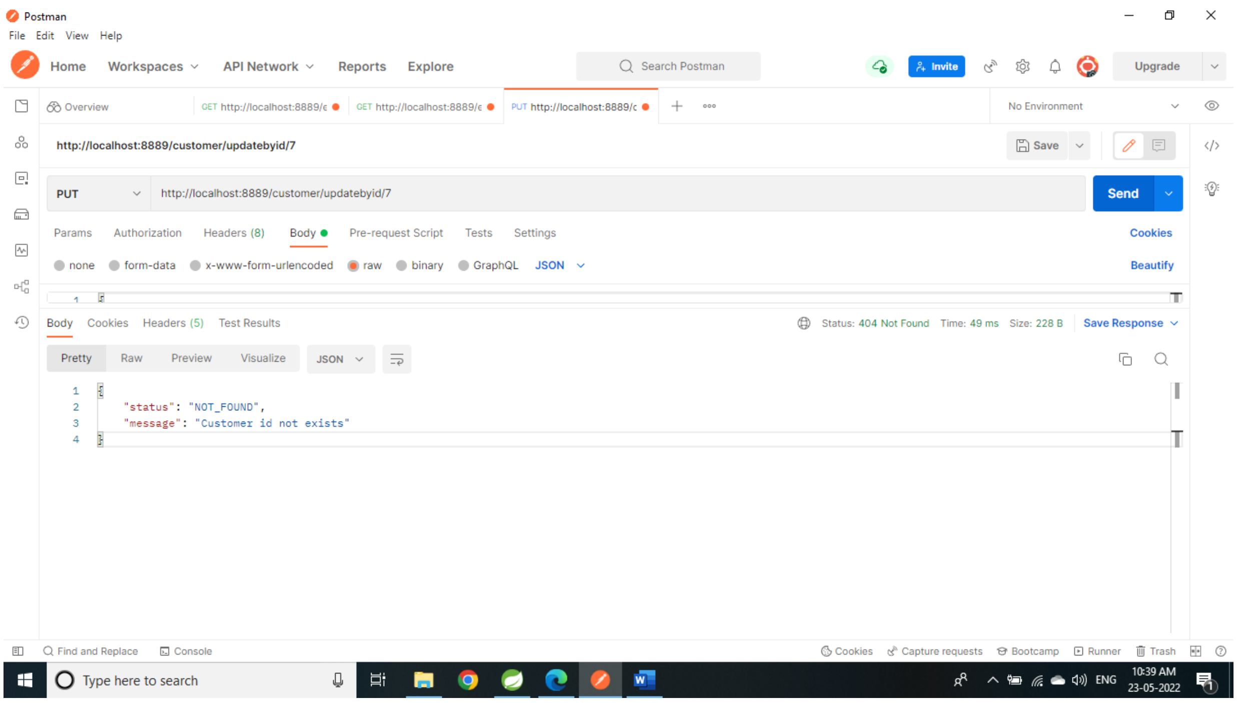
http://localhost:8889/customer/selectbyid/6

GET http://localhost:8889/customer/selectbyid/6

Status: 404 Not Found Time: 34 ms Size: 228 B

```
{
    "status": "NOT_FOUND",
    "message": "Customer id not exists"
}
```

Update: http://localhost:8889/customer/updatebyid/7


Postman
File Edit View Help
Home Workspaces API Network Reports Explore
Search Postman
Invite
Upgrade
Overview
No Environment
Save
Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
Body Cookies Headers (5) Test Results
Save Response
Pretty Raw Preview Visualize JSON
Find and Replace Console
Cookies Capture requests Bootcamp Runner Trash
Type here to search
ENG
10:39 AM
23-05-2022

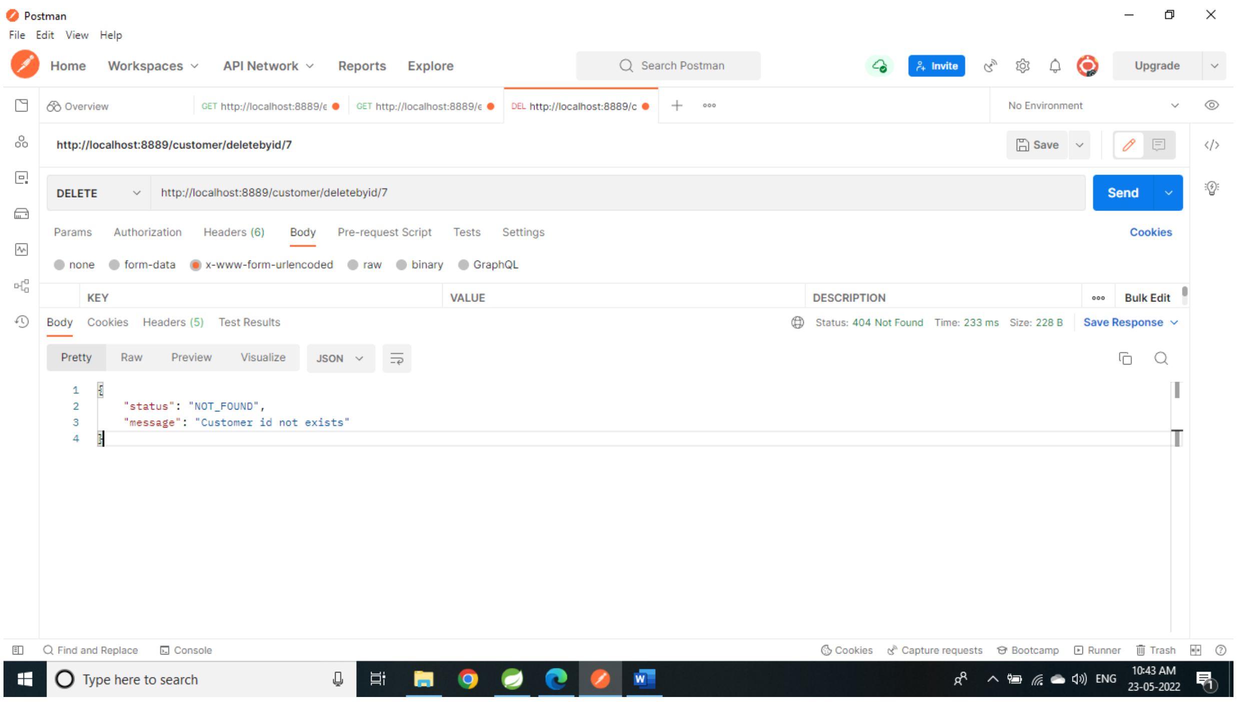
http://localhost:8889/customer/updatebyid/7

PUT http://localhost:8889/customer/updatebyid/7

Status: 404 Not Found Time: 49 ms Size: 228 B

```
{
    "status": "NOT_FOUND",
    "message": "Customer id not exists"
}
```

Delete: http://localhost:8889/customer/deletebyid/7


Postman
File Edit View Help
Home Workspaces API Network Reports Explore
Search Postman
Invite
Upgrade
Overview
No Environment
Save
Send
Params Authorization Headers (6) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL
KEY VALUE DESCRIPTION Bulk Edit
Body Cookies Headers (5) Test Results
Save Response
Pretty Raw Preview Visualize JSON
Find and Replace Console
Cookies Capture requests Bootcamp Runner Trash
Type here to search
ENG
10:43 AM
23-05-2022

http://localhost:8889/customer/deletebyid/7

DELETE http://localhost:8889/customer/deletebyid/7

Status: 404 Not Found Time: 233 ms Size: 228 B

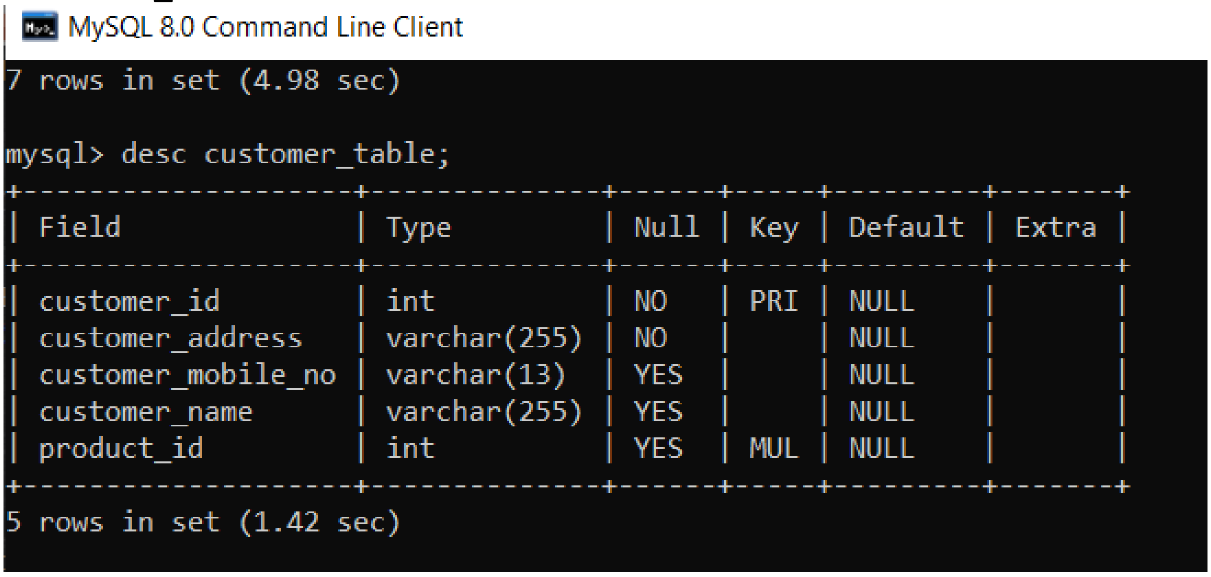
```
{
    "status": "NOT_FOUND",
    "message": "Customer id not exists"
}
```

Input format:

```
{

      "productName": "jimjam",
      "productType": "biscuit",
      "productBrand": "sweety",
      "productQuantity": 20,
      "productPrice": 10.0,

      "customer": [
         {

            "customerName": "tharani",
            "customerMobileNo": "9000677897",
            "customerAddress": "karaikal"
         },
         {

            "customerName": "thilaksha",
            "customerMobileNo": "9900677897",
            "customerAddress": "karaikal"
         }
      ]
   }
```

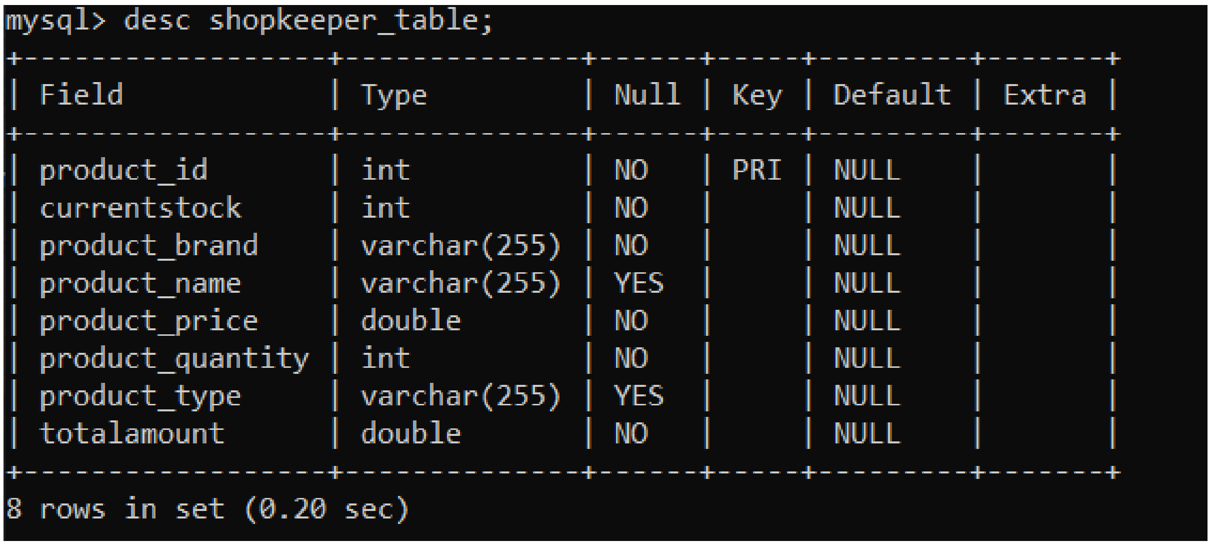
DATABASE TABLES:

Customer_table


MySQL 8.0 Command Line Client

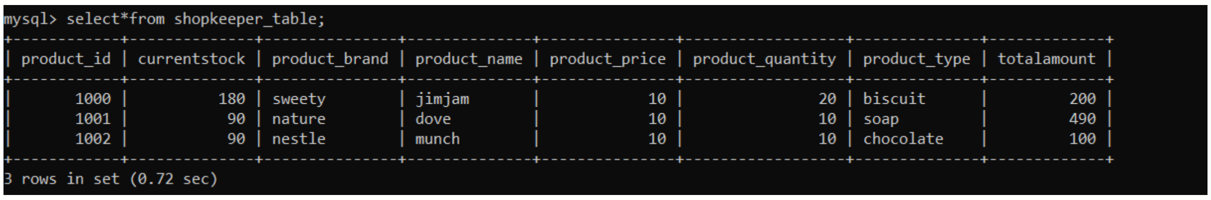
```
7 rows in set (4.98 sec)

mysql> desc customer_table;
+--------------------+--------------+------+-----+---------+-------+
| Field              | Type         | Null | Key | Default | Extra |
+--------------------+--------------+------+-----+---------+-------+
| customer_id        | int          | NO   | PRI | NULL    |       |
| customer_address   | varchar(255) | NO   |     | NULL    |       |
| customer_mobile_no | varchar(13)  | YES  |     | NULL    |       |
| customer_name      | varchar(255) | YES  |     | NULL    |       |
| product_id         | int          | YES  | MUL | NULL    |       |
+--------------------+--------------+------+-----+---------+-------+
5 rows in set (1.42 sec)
```

Shopkeeper_table



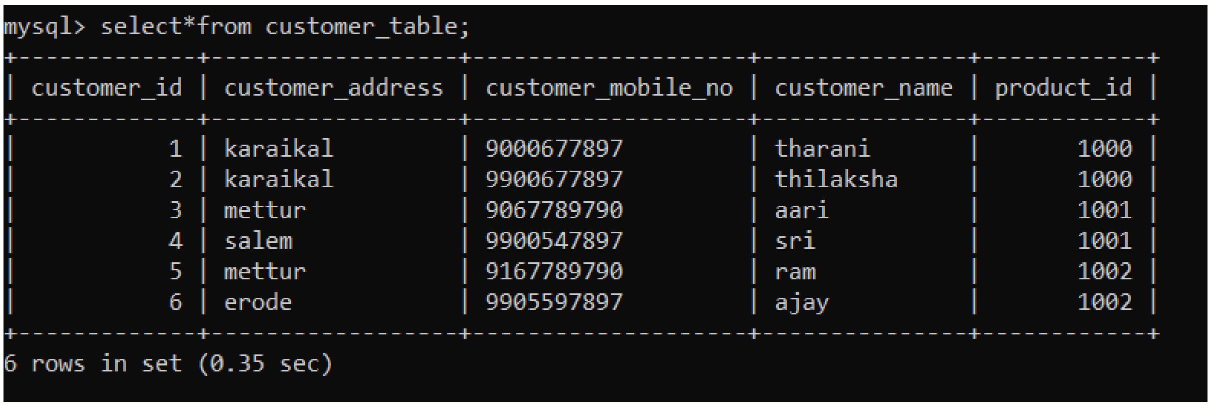
```
mysql> desc shopkeeper_table;
+------------------+--------------+------+-----+---------+-------+
| Field            | Type         | Null | Key | Default | Extra |
+------------------+--------------+------+-----+---------+-------+
| product_id       | int          | NO   | PRI | NULL    |       |
| currentstock     | int          | NO   |     | NULL    |       |
| product_brand    | varchar(255) | NO   |     | NULL    |       |
| product_name     | varchar(255) | YES  |     | NULL    |       |
| product_price    | double       | NO   |     | NULL    |       |
| product_quantity | int          | NO   |     | NULL    |       |
| product_type     | varchar(255) | YES  |     | NULL    |       |
| totalamount      | double       | NO   |     | NULL    |       |
+------------------+--------------+------+-----+---------+-------+
8 rows in set (0.20 sec)
```

Product Database



```
mysql> select*from shopkeeper_table;
+------------+--------------+---------------+--------------+---------------+------------------+--------------+-------------+
| product_id | currentstock | product_brand | product_name | product_price | product_quantity | product_type | totalamount |
+------------+--------------+---------------+--------------+---------------+------------------+--------------+-------------+
|       1000 |          180 | sweety        | jimjam       |            10 |               20 | biscuit      |         200 |
|       1001 |           90 | nature        | dove         |            10 |               10 | soap         |         490 |
|       1002 |           90 | nestle        | munch        |            10 |               10 | chocolate    |         100 |
+------------+--------------+---------------+--------------+---------------+------------------+--------------+-------------+
3 rows in set (0.72 sec)
```

Customer Database



```
mysql> select*from customer_table;
+-------------+------------------+--------------------+---------------+------------+
| customer_id | customer_address | customer_mobile_no | customer_name | product_id |
+-------------+------------------+--------------------+---------------+------------+
|           1 | karaikal         | 9000677897         | tharani       |       1000 |
|           2 | karaikal         | 9900677897         | thilaksha     |       1000 |
|           3 | mettur           | 9067789790         | aari          |       1001 |
|           4 | salem            | 9900547897         | sri           |       1001 |
|           5 | mettur           | 9167789790         | ram           |       1002 |
|           6 | erode            | 9905597897         | ajay          |       1002 |
+-------------+------------------+--------------------+---------------+------------+
6 rows in set (0.35 sec)
```

CONCLUSION

In Inventory management system we implement using spring boot to build the program in very easy way and using postman to run the program and database to store all the information of the product and customer details. Fetching details are very easy on this system.