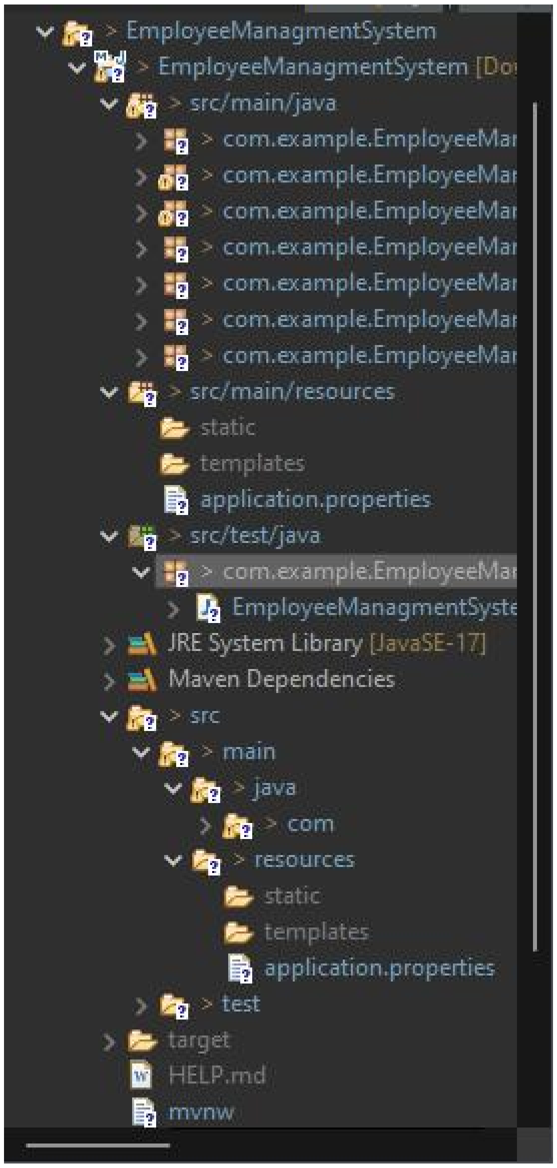
SPRING BOOT

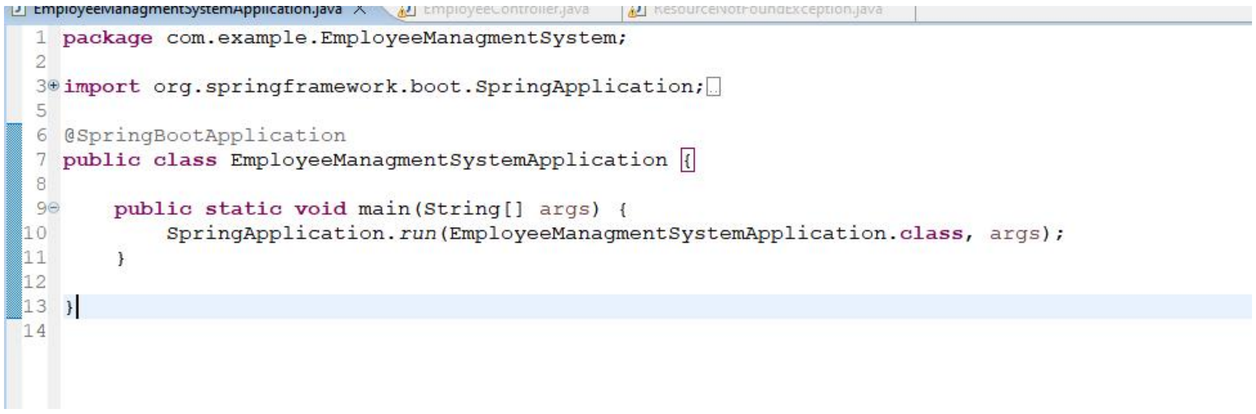
1) **Design a Spring Boot program to create a CRUD (Create, Read, Update, Delete) application using Hibernate for managing employee records. The program should allow users to perform the following operations on the employee database:**

    a) **Add a new employee: The user can enter details like employee name, department, and salary, and the program should add the employee to the database.**
    b) **Update employee details: The user can update the name, department, or salary of an existing employee based on their employee ID.**
    c) **Delete an employee: The user can delete an employee from the database based on their employee ID.**
    d) **Display all employees: The program should retrieve and display a list of all employees and their details from the database.**
    e) **Requirements:**
        i) **Use Spring Boot to create the application and Hibernate to manage the database.**
        ii) **Implement JPA (Java Persistence API) for data access.**
        iii) **Provide a RESTful API for performing CRUD operations on employees.**
        iv) **Implement exception handling to handle possible errors during database interactions.**
        v) **Cover Spring Boot and Hibernate topics, such as entity classes, repositories, services, and controllers.**
    f) **Note: Before running the program, make sure you have set up the database and configured the connection in the application.properties file.**

EmployeeManagement System Setup ;



`EmployeeManagmentSystem Application.java;`



```java
package com.example.EmployeeManagmentSystem;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class EmployeeManagmentSystemApplication {

    public static void main(String[] args) {
        SpringApplication.run(EmployeeManagmentSystemApplication.class, args);
    }

}
```

Employee Controller.java ;

```java
package com.example.EmployeeManagmentSystem.controller;
@RestController
@RequestMapping("/api/employeese")
public class EmployeeController {
@Autowired
private EmployeeService employeeService;
//restapi we created for saving employee
@PostMapping("/save")
public ResponseEntity<Employee> SaveEmployee(@RequestBody Employee employee){
return new
ResponseEntity<Employee>(employeeService.saveEmployee(employee),HttpStatus.CREATED);
}
//restapi for retrieving all employees
@GetMapping("/getemps")
public List<Employee> getEmployees(Employee employee){
return employeeService.getAllEmployees(employee);
}
//restapi for getting one employee
@GetMapping("{id}")
public ResponseEntity<Employee> getEmployee(@PathVariable("id") int id) {
return new ResponseEntity<Employee>(employeeService.getEmployee(id), HttpStatus.OK);
}
//restapi for updating the employee
@PutMapping("{id}")
public ResponseEntity<Employee> updateEmployee(@PathVariable("id") int id,
@RequestBody Employee employee){
return new ResponseEntity<Employee>(employeeService.updateEmployee(employee,
id),HttpStatus.OK);
}
//restapi for deleting the employee
@DeleteMapping("{id}")
public ResponseEntity<Employee> deleteEmployee(@PathVariable("id") int id){
return new ResponseEntity<Employee>(employeeService.deleteEmployee(id),HttpStatus.OK);
}
}
```

ResourceNotFound Exception ;

```java
package com.example.EmployeeManagmentSystem.exception;

@ResponseStatus(value=HttpStatus.NOT_FOUND)
public class ResourceNotFoundException extends RuntimeException {

private String resourceName;
private String fieldname;
private int fileldvalue;
public ResourceNotFoundException(String resourceName, String fieldname, int id) {
super(String.format(resourceName, fieldname,id,"%s resource not found %s :%s"));
this.resourceName = resourceName;
this.fieldname = fieldname;
this.fileldvalue = id;
}
public String getResourceName() {
return resourceName;
}
public void setResourceName(String resourceName) {
this.resourceName = resourceName;
}
public String getFieldname() {
return fieldname;
}
public void setFieldname(String fieldname) {
this.fieldname = fieldname;
}
public int getFileldvalue() {
return fileldvalue;
}
public void setFileldvalue(int fileldvalue) {
this.fileldvalue = fileldvalue;
}
}
```

Employee.java ;

```
package com.example.EmployeeManagmentSystem.pojo;

@Entity
@Table(name="employees")
public class Employee {

@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private int id;
@Column(name="ename")
private String name;
@Column(name="email")
private String email;
@Column(name="phone",nullable=false)
private long phoneno;

public int getId() {
return id;
}

public void setId(int id) {
this.id = id;
}

public String getName() {
return name;
}

public void setName(String name) {
this.name = name;
}
public String getEmail() {
return email;
}

public void setEmail(String email) {
this.email = email;
}

public long getPhoneno() {
return phoneno;
}

public void setPhoneno(long phoneno) {
this.phoneno = phoneno;
}

@Override
public String toString() {
return "Employee [id=" + id + ", name=" + name + ", email=" + email + ", phoneno=" +
phoneno + "]";
}
}
```

EmployeeRepo.java ;

```
package com.example.EmployeeManagmentSystem.repository;
import org.springframework.data.jpa.repository.JpaRepository;
@Repository
public interface EmployeeRepo extends JpaRepository<Employee, Integer> {

}
```

EmployeeManagementSyst ;

```
package com.example.EmployeeManagmentSystem;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class EmployeeManagmentSystemApplication {

public static void main(String[] args) {
SpringApplication.run(EmployeeManagmentSystemApplication.class, args);
}

}
```

Resources Application Properties ;

```
spring.application.name=EmployeeManagmentSystem
spring.application.name=springbootdemo
spring.datasource.url=jdbc:mysql://localhost:3306/employeesSpringBoot?createDatabaseIfNot
Exist=true
spring.datasource.username=root
spring.datasource.password=root

#Hibernate JPA PROPERTIES:
```

```
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
spring.jpa.hibernate.ddl-auto=update
```

EmployeeManagement Application.java ;

```java
package com.example.EmployeeManagmentSystem;

import org.junit.jupiter.api.Test;

@SpringBootTest
class EmployeeManagmentSystemApplicationTests {

@Test
void contextLoads() {
}

}
```

Pom.xml ;

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-
4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<parent>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>3.3.1</version>
<relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.example</groupId>
<artifactId>EmployeeManagmentSystem</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>EmployeeManagmentSystem</name>
<description>Employee Crud operations Spring Boot</description>
<url/>
<licenses>
<license/>
</licenses>
<developers>
<developer/>
</developers>
<scm>
<connection/>
<developerConnection/>
<tag/>
<url/>
</scm>
<properties>
<java.version>17</java.version>
</properties>
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

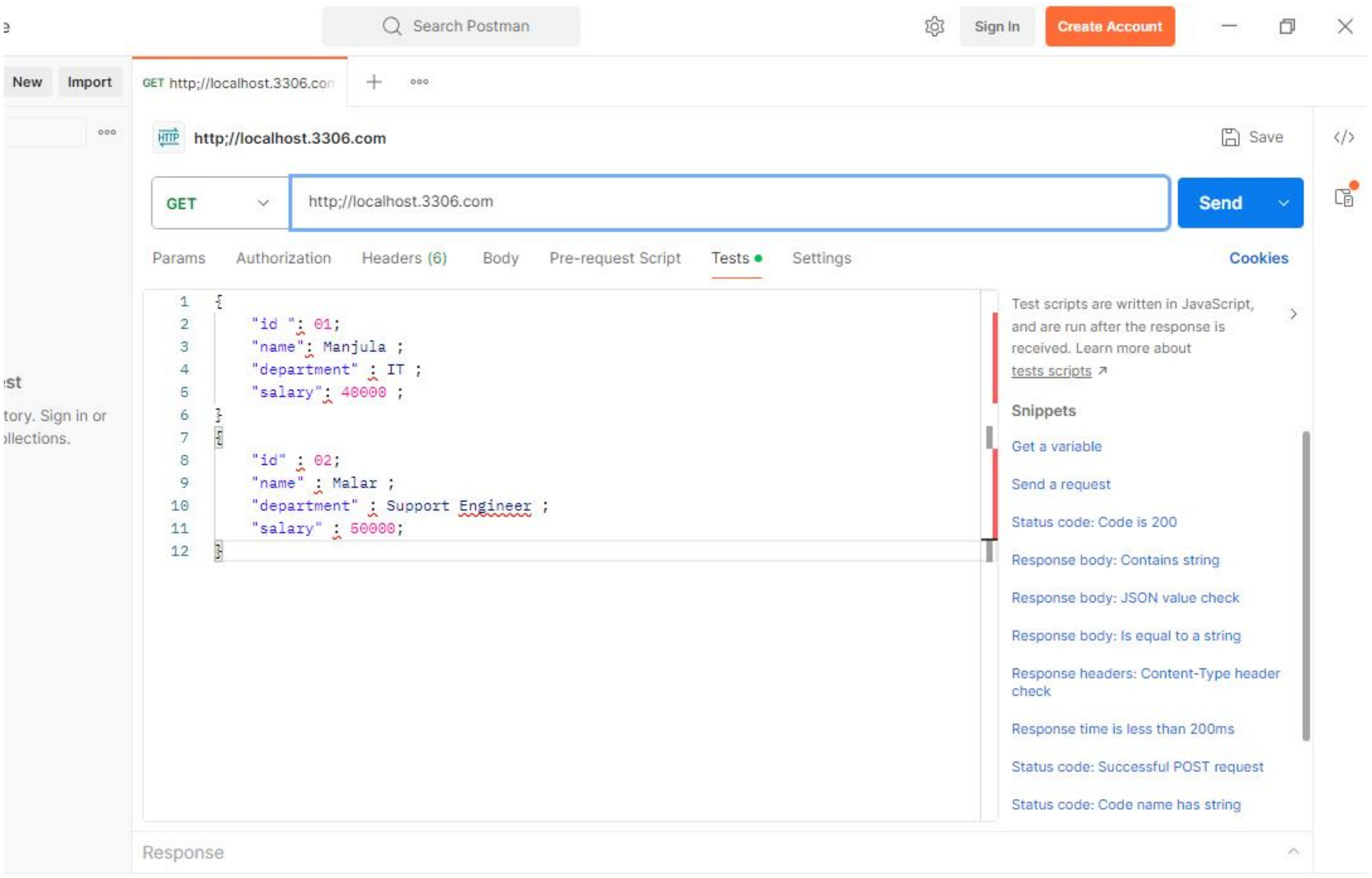
```xml
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-devtools</artifactId>
<scope>runtime</scope>
<optional>true</optional>
</dependency>
<dependency>
<groupId>com.mysql</groupId>
<artifactId>mysql-connector-j</artifactId>
<scope>runtime</scope>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<scope>test</scope>
</dependency>
</dependencies>

<build>
<plugins>
<plugin>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
</plugins>
</build>

</project>
```

Output ;



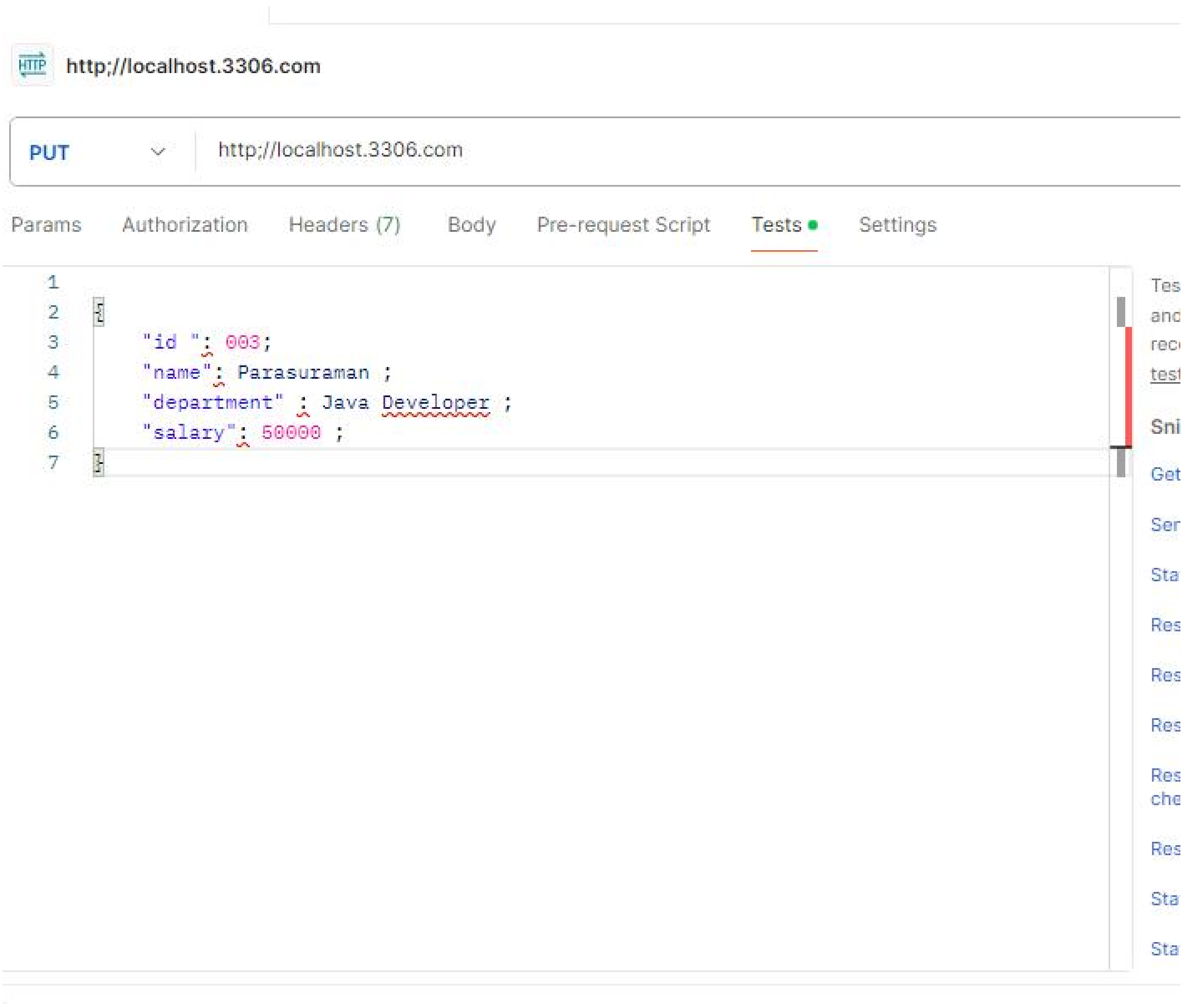
GET http;//localhost.3306.com

Tests

```
{
    "id ": 01;
    "name": Manjula ;
    "department" : IT ;
    "salary": 40000 ;
}
{
    "id" : 02;
    "name" : Malar ;
    "department" : Support Engineer ;
    "salary" : 50000;
}
```

Test scripts are written in JavaScript, and are run after the response is received. Learn more about tests scripts

Search Postman
Sign In
Create Account
New
Import
Save
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Settings
Cookies
Snippets
Get a variable
Send a request
Status code: Code is 200
Response body: Contains string
Response body: JSON value check
Response body: Is equal to a string
Response headers: Content-Type header check
Response time is less than 200ms
Status code: Successful POST request
Status code: Code name has string
Response

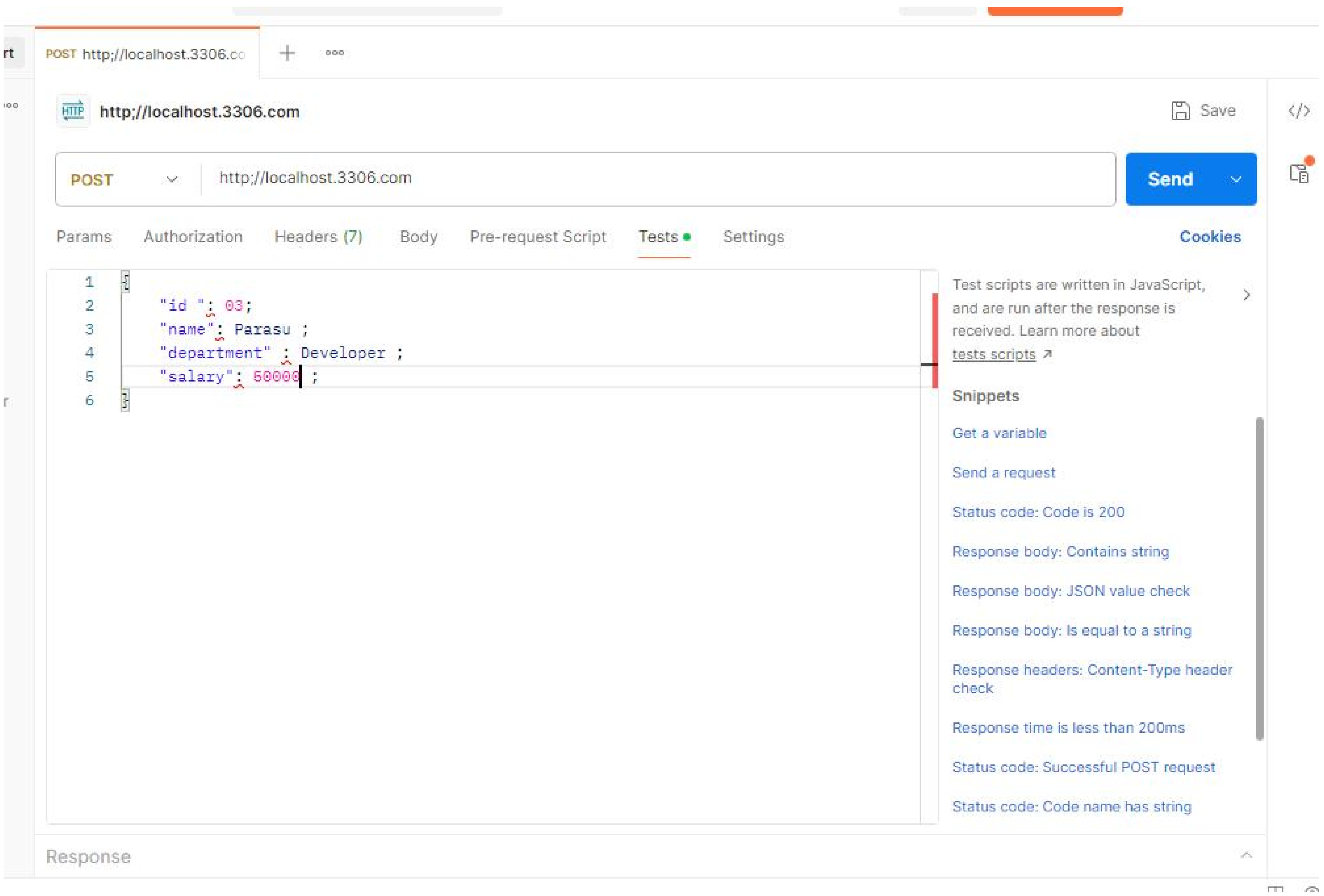


http;//localhost.3306.com

PUT http;//localhost.3306.com

Params Authorization Headers (7) Body Pre-request Script Tests Settings

```
{
    "id ": 003;
    "name": Parasuraman ;
    "department" : Java Developer ;
    "salary": 50000 ;
}
```



```
{
    "id ": 03;
    "name": Parasu ;
    "department" : Developer ;
    "salary": 50000 ;
}
```

Test scripts are written in JavaScript, and are run after the response is received. Learn more about tests scripts ↗

Snippets

Get a variable

Send a request

Status code: Code is 200

Response body: Contains string

Response body: JSON value check

Response body: Is equal to a string

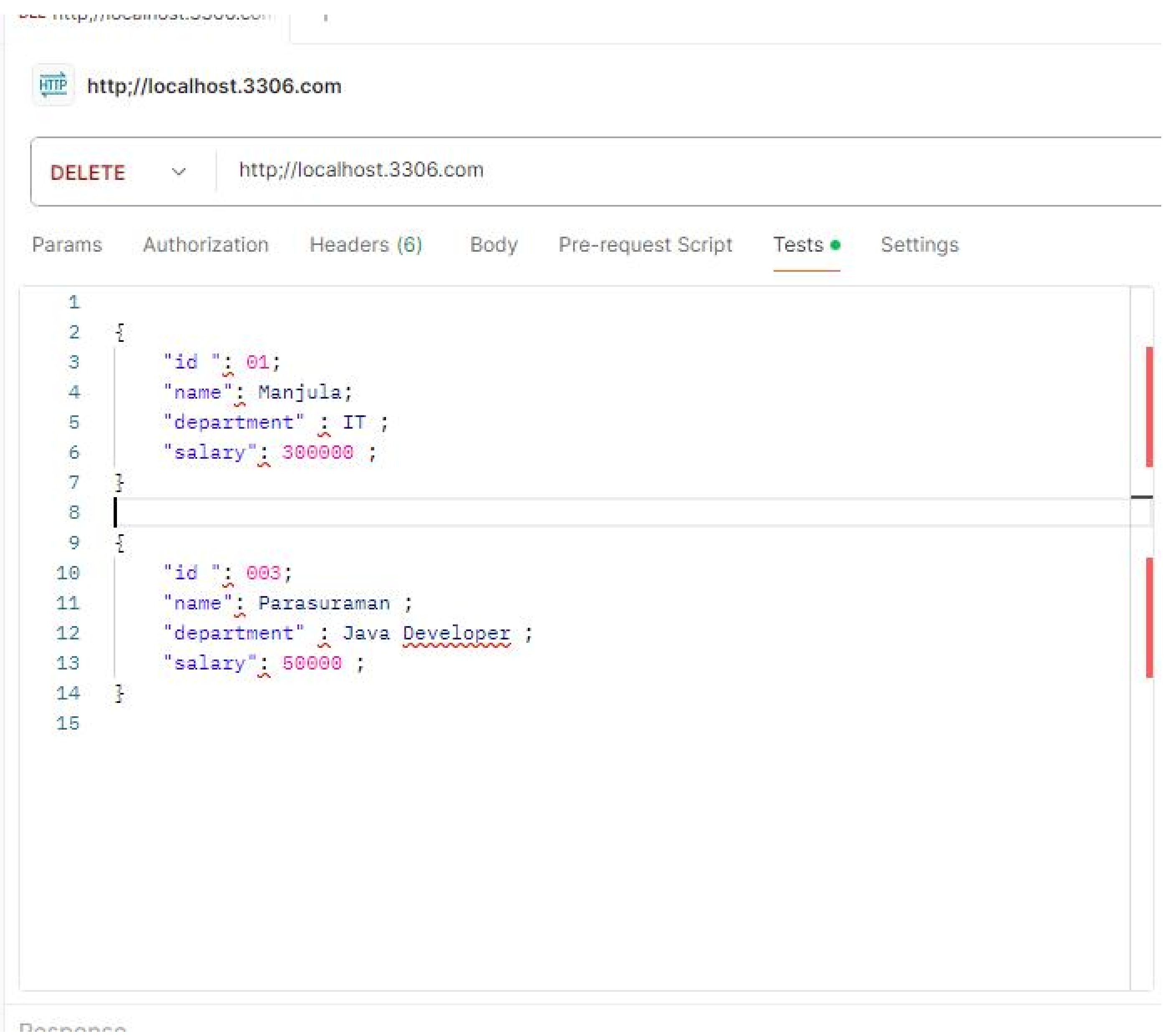
Response headers: Content-Type header check

Response time is less than 200ms

Status code: Successful POST request

Status code: Code name has string

Response



http;//localhost.3306.com

```

{
    "id ": 01;
    "name": Manjula;
    "department" : IT ;
    "salary": 300000 ;
}

{
    "id ": 003;
    "name": Parasuraman ;
    "department" : Java Developer ;
    "salary": 50000 ;
}

```