

RestFul Web Service Part 1

1. Create a simple REST ful service in Spring Boot which returns the Response "Welcome to spring boot".

CODE

```
@RestController
public class EmployeeController {

    @GetMapping(path = "/Hello-World")
    public String displayMessage()
    {
        return "Hello World";
    }
}
```

OUTPUT

The screenshot shows a REST client interface with a top bar containing request history (GET, POST, PUT, DEL) and a 'No Environment' dropdown. The main area is titled 'Untitled Request' and shows a 'GET' request to 'http://localhost:8080/Hello-World'. Below this are tabs for 'Params', 'Authorization', 'Headers (8)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Query Params' section is expanded, showing a table with columns 'KEY', 'VALUE', and 'DESCRIPTION'. The 'Body' tab is selected, showing the response 'Hello World' in a 'Pretty' view. The status bar at the bottom indicates 'Status: 200 OK', 'Time: 110ms', and 'Size: 185 B'.

KEY	VALUE	DESCRIPTION
Key	Value	Description

1 Hello World

2. Create an Employee Bean(id, name, age) and service to perform different operations related to employee.

CODE

Employee.java

```
public class Employee {
    private Integer id;

    @Size(min = 2,message = "Name should have atleast 2 characters.")
    private String name;
    private int age;
    @Past
    private Date birthDate;

    protected Employee(){

    }

    public Employee(Integer id, String name, int age, Date birthDate) {
        this.id = id;
        this.name = name;
        this.age = age;
        this.birthDate = birthDate;
    }

    public Integer getId() {
        return id;
    }

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

    public String getName() {
        return name;
    }

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

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
```

```

        this.age = age;
    }

    public Date getBirthDate() {
        return birthDate;
    }

    public void setBirthDate(Date birthDate) {
        this.birthDate = birthDate;
    }

    @Override
    public String toString() {
        return "Employee{" +
            "id=" + id +
            ", name=" + name + "\" +
            ", age=" + age +
            ", birthDate=" + birthDate +
            "}";
    }
}

```

EmployeeDao.java

```

package com.RestfullWebService.Assignment1.RestFullWebServiceAssignment1.Model;

import
com.RestfullWebService.Assignment1.RestFullWebServiceAssignment1.Service.Employee;
import org.springframework.stereotype.Component;

import java.util.*;

@Component
public class EmployeeDao {

    List<Employee> ls=new ArrayList<>();

    //Retrieve-list-of-Employees
    public List<Employee> findAll(){
        return ls;
    }

    //Add-Employee

```

```

public Employee addEmployee(Employee employee)
{
    ls.add(employee);
    return employee;
}

//Find-One-Employee
public Employee findOne(Integer id)
{
    for (Employee employee:ls){
        if(employee.getId()==id)
            return employee;
    }
    return null;
}

//Delete-One-Employee
public Employee deleteById(Integer id)
{
    Iterator<Employee> iterator = ls.iterator();
    while (iterator.hasNext()){
        Employee employee = iterator.next();
        if(employee.getId()==id) {
            iterator.remove();
            return employee;
        }
    }
    return null;
}

// Employee-Put-Mapping
public void putEmployee(Integer id, Employee employee){

    Iterator<Employee> iterator = ls.iterator();
    while (iterator.hasNext()){
        iterator.next().setName(employee.getName());
        iterator.next().setAge(employee.getAge());
        iterator.next().setBirthDate(employee.getBirthDate());
    }
}

```

3. Implement GET http request for Employee to get list of employees.

CODE

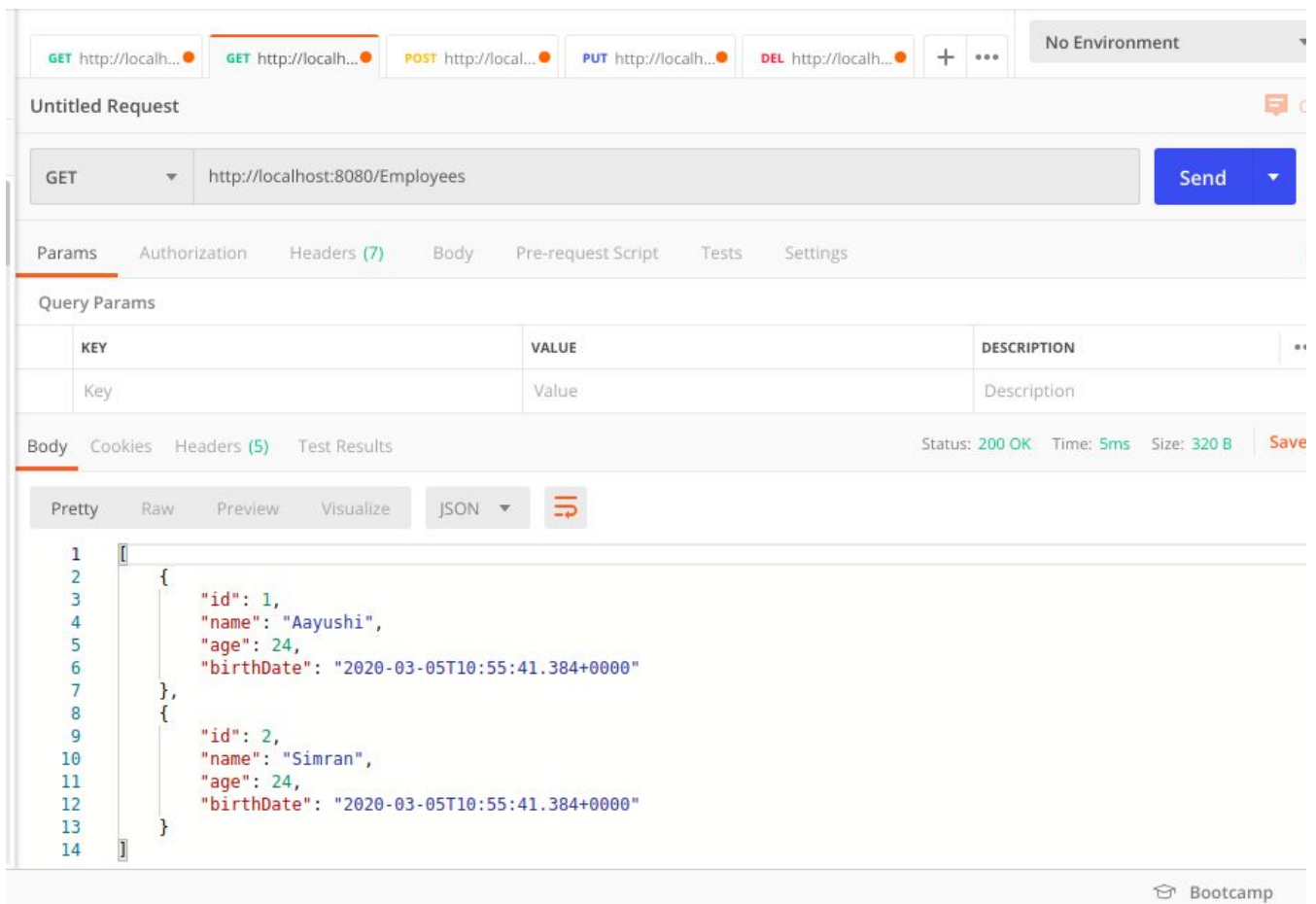
EmployeeController.java

```
//Retrieve-List-Of-Employees
@GetMapping(path="/Employees")
public List<Employee> retrieveAllEmployees(){
    return service.findAll();
}
```

EmployeeDao.java

```
List<Employee> ls=new ArrayList<>();
//Retrieve-list-of-Employees
public List<Employee> findAll(){
    return ls;
}
```

OUTPUT



The screenshot shows a REST client interface with a GET request to `http://localhost:8080/Employees`. The response status is `200 OK`, with a time of `5ms` and a size of `320 B`. The response body is displayed in JSON format, showing a list of two employees:

```
[
  {
    "id": 1,
    "name": "Aayushi",
    "age": 24,
    "birthDate": "2020-03-05T10:55:41.384+0000"
  },
  {
    "id": 2,
    "name": "Simran",
    "age": 24,
    "birthDate": "2020-03-05T10:55:41.384+0000"
  }
]
```

The interface also shows tabs for Params, Authorization, Headers (7), Body, Pre-request Script, Tests, and Settings. The Body tab is selected, and the response is formatted as JSON. The status bar at the bottom indicates the environment is 'No Environment' and the client is 'Bootcamp'.

4. Implement GET http request using path variable to get one employee.

CODE

EmployeeController.java

```
//Retrieve-Specific-Employee
@GetMapping(path="/Employees/{id}")
public Employee retrieveOneEmployee(@PathVariable Integer id){

    Employee employee = service.findOne(id);
    if(employee==null)
        throw new UserNotFoundException("id - "+ id);
    return employee;
}
```

EmployeeDao.java

```
//Find-One-Employee
public Employee findOne(Integer id)
{
    for (Employee employee:ls){
        if(employee.getId()==id)
            return employee;
    }
    return null;
}
```

OUTPUT

The screenshot shows a REST client interface with a tab for 'GET http://localhost:8080/employees/1'. The 'Send' button is visible. Below the request bar, the 'Query Params' section is empty. The 'Body' tab is selected, showing a JSON response in 'Pretty' format:

```

1 {
2   "id": 1,
3   "name": "Aayushi",
4   "age": 24,
5   "birthDate": "2020-03-05T10:55:41.384+0000"
6 }

```

At the bottom right, there is a 'Bootcamp' logo.

5. Implement POST http request for Employee to create a new employee.

CODE

EmployeeController.java

```

//Add-Employee
@PostMapping(path="/Employees")
public Employee createEmployee(@Valid @RequestBody Employee employee)
{
    Employee saveEmployee = service.addEmployee(employee);
    return employee;
}

```

EmployeeDao.java

```

//Add-Employee
public Employee addEmployee(Employee employee)
{
    ls.add(employee);
}

```

```
    return employee;  
}
```

OUTPUT

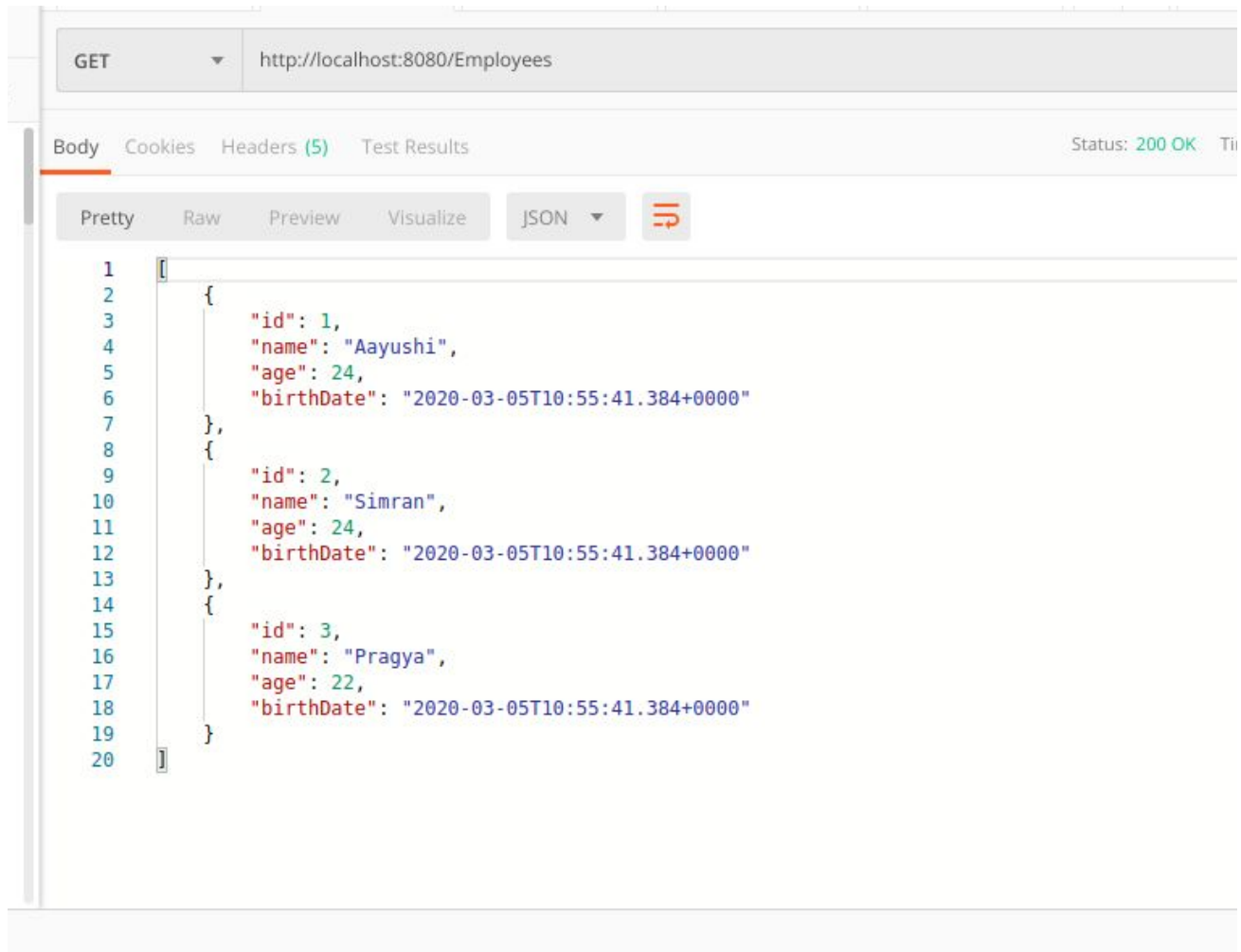
The screenshot displays a REST client interface with a POST request to `http://localhost:8080/employees`. The request body is a JSON object: `{ "id": 3, "name": "Pragya", "age": 22, "birthDate": "2020-03-05T10:55:41.384+0000" }`. The response status is `200 OK` with a time of `6ms` and a size of `240 B`. The response body is also shown in a pretty-printed JSON format.

REST Client Interface:

- Method: POST
- URL: `http://localhost:8080/employees`
- Body Type: JSON
- Body Content:

```
1 {  
2   "id": 3,  
3   "name": "Pragya",  
4   "age": 22,  
5   "birthDate": "2020-03-05T10:55:41.384+0000"  
6 }
```
- Status: 200 OK
- Time: 6ms
- Size: 240 B
- Response Body (Pretty):

```
1 {  
2   "id": 3,  
3   "name": "Pragya",  
4   "age": 22,  
5   "birthDate": "2020-03-05T10:55:41.384+0000"  
6 }
```

6. Implement Exception Handling for resource not found.

CODE

EmployeeController.java

```
//Retrieve-Specific-Employee
@GetMapping(path="/Employees/{id}")
public Employee retrieveOneEmployee(@PathVariable Integer id){

    Employee employee = service.findOne(id);
    if(employee==null)
        throw new UserNotFoundException("id - "+ id);
    return employee;
}
```

ExceptionResponse.java

```
public class ExceptionResponse {
```

```

private Date timestamp;
private String message;
private String details;

public ExceptionResponse(Date timestamp, String message, String details) {
    super();
    this.timestamp = timestamp;
    this.message = message;
    this.details = details;
}

public Date getTimestamp() {
    return timestamp;
}

public String getMessage() {
    return message;
}

public String getDetails() {
    return details;
}
}

```

UserNotFoundException.java

```

@ResponseStatus(HttpStatus.NOT_FOUND)
public class UserNotFoundException extends RuntimeException {

    public UserNotFoundException(String s) {
        super(s);
    }
}

```

CustomizedResponseEntityExceptionHandler.java

```

@ControllerAdvice //shared across multiple classes
@RestController
public class CustomizedResponseEntityExceptionHandler
    extends ResponseEntityExceptionHandler {

    @ExceptionHandler(Exception.class)
    public final ResponseEntity<Object> handleAllExceptions
        (Exception ex, WebRequest request) {

```

```

@ExceptionHandler(UserNotFoundException.class)
public final ResponseEntity<Object> handleUserNotFoundExceptions
    (UserNotFoundException ex, WebRequest request) {

    ExceptionResponse exceptionResponse = new ExceptionResponse(new
Date(),ex.getMessage(),request.getDescription(false));
    return new ResponseEntity(exceptionResponse, HttpStatus.NOT_FOUND);
}
}

```

OUTPUT

The screenshot shows a REST client interface with a request bar at the top. The request is a GET to `http://localhost:8080/Employees/500`. Below the request bar, there are tabs for Params, Authorization, Headers (7), Body, Pre-request Script, Tests, and Settings. The Headers tab is selected, showing a table with 7 headers. Below the headers, there are tabs for Body, Cookies, Headers (5), and Test Results. The Body tab is selected, showing a JSON response in Pretty format. The response status is 404 Not Found, with a time of 12ms and a size of 267 B. The JSON response is:

```

{
  "timestamp": "2020-03-07T11:06:57.696+0000",
  "message": "id - 500",
  "details": "uri=/Employees/500"
}

```

At the bottom right of the interface, there is a Bootcamp logo.

7. Implement DELETE http request for Employee to delete employee.

CODE

EmployeeController.java

```
//Delete-A-Employee
@DeleteMapping(path = "/Employees/{id}")
public void deleteUSer(@PathVariable Integer id)
{
    Employee employee = service.deleteById(id);
}
```

EmployeeDao.java

```
//Delete-One-Employee
public Employee deleteById(Integer id)
{
    Iterator<Employee> iterator = ls.iterator();
    while (iterator.hasNext()){
        Employee employee = iterator.next();
        if(employee.getId()==id) {
            iterator.remove();
            return employee;
        }
    }
    return null;
}
```

OUTPUT

The screenshot shows a REST client interface with the following details:

- Method:** DELETE
- URL:** http://localhost:8080/Employees/2
- Status:** 200 OK
- Time:** 11ms
- Size:** 123 B
- Response Body:** Empty (indicated by a single line in the Pretty view)

KEY	VALUE	DESCRIPTION
Key	Value	Description

GET http://localhost:8080/Employees Send

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (5) Test Results Status: 200 OK Time: 8ms Size: 320 B

Pretty Raw Preview Visualize JSON

```

1  [
2    {
3      "id": 1,
4      "name": "Aayushi",
5      "age": 24,
6      "birthDate": "2020-03-05T10:55:41.384+0000"
7    },
8    {
9      "id": 3,
10     "name": "Pragya",
11     "age": 22,
12     "birthDate": "2020-03-05T10:55:41.384+0000"
13   }
14 ]

```

8. Implement PUT http request for Employee to update employee.

CODE

EmployeeController.java

```

@PutMapping(path = "/Employees/{id}")
public void putEmployee(@PathVariable Integer id, @RequestBody Employee employee){
    service.putEmployee(id,employee);
}

```

EmployeeDao.java

```

// Employee-Put-Mapping
public void putEmployee(Integer id, Employee employee){

    Iterator<Employee> iterator = ls.iterator();
    while (iterator.hasNext()){
        iterator.next().setName(employee.getName());
        iterator.next().setAge(employee.getAge());
        iterator.next().setBirthDate(employee.getBirthDate());
    }
}

```

OUTPUT

GET http://localhost:8080/Employees Send

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (5) Test Results Status: 200 OK Time: 5ms Size: 319 B Save

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "id": 1,
4     "name": "Simran",
5     "age": 24,
6     "birthDate": "2020-03-05T10:55:41.384+0000"
7   },
8   {
9     "id": 3,
10    "name": "Pragya",
11    "age": 22,
12    "birthDate": "2020-03-05T10:55:41.384+0000"
13  }
14 ]
```

GET http://localh... GET http://localh... POST http://localh... PUT http://localh... DEL http://localh... + ... No Environmen

Untitled Request

PUT http://localhost:8080/Employees/1

Params Authorization Headers (9) Body Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON

```
1 {
2   "id": 1,
3   "name": "Aayushi",
4   "age": 24,
5   "birthDate": "2020-03-05T10:55:41.384+0000"
6 }
```

GET http://localhost:8080/Employees

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (5) Test Results Status: 200 OK Time: 15ms Size: 321

```

1  [
2    {
3      "id": 1,
4      "name": "Aayushi",
5      "age": 24,
6      "birthDate": "2020-03-05T10:55:41.384+0000"
7    },
8    {
9      "id": 3,
10     "name": "Pragya",
11     "age": 24,
12     "birthDate": "2020-03-05T10:55:41.384+0000"
13   }
14 ]
  
```

9. Apply validation while create a new employee using POST http Request.

CODE

EmployeeController.java

```

//Add-Employee
@PostMapping(path="/Employees")
public Employee createEmployee(@Valid @RequestBody Employee employee)
{
    Employee saveEmployee = service.addEmployee(employee);
    return employee;
}
  
```

Employee.java

```

@Size(min = 2,message = "Name should have atleast 2 characters.")
private String name;
private int age;
@Past
private Date birthDate;
  
```

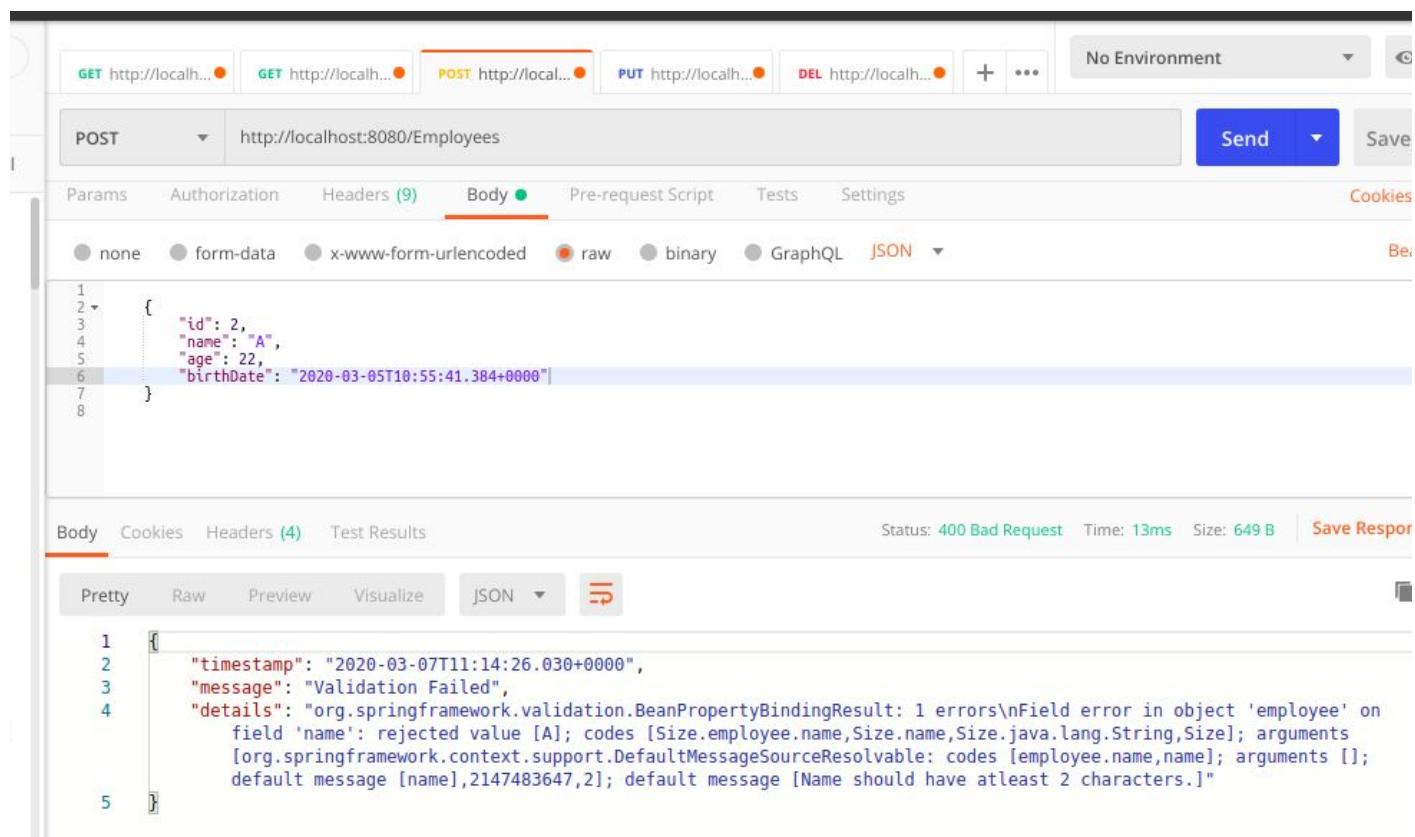
CustomizedResponseEntityExceptionHandler.java

@Override

```
protected ResponseEntity<Object> handleMethodArgumentNotValid(  
    MethodArgumentNotValidException ex, HttpHeaders headers, HttpStatus status,  
    WebRequest request) {
```

```
    ExceptionResponse exceptionResponse = new ExceptionResponse(new Date(),  
        "Validation Failed",ex.getBindingResult().toString());  
    return new ResponseEntity(exceptionResponse, HttpStatus.BAD_REQUEST);  
}
```

OUTPUT



10. Configure actuator in your project to check the health of application and get the information about various beans configured in your application.

CODE

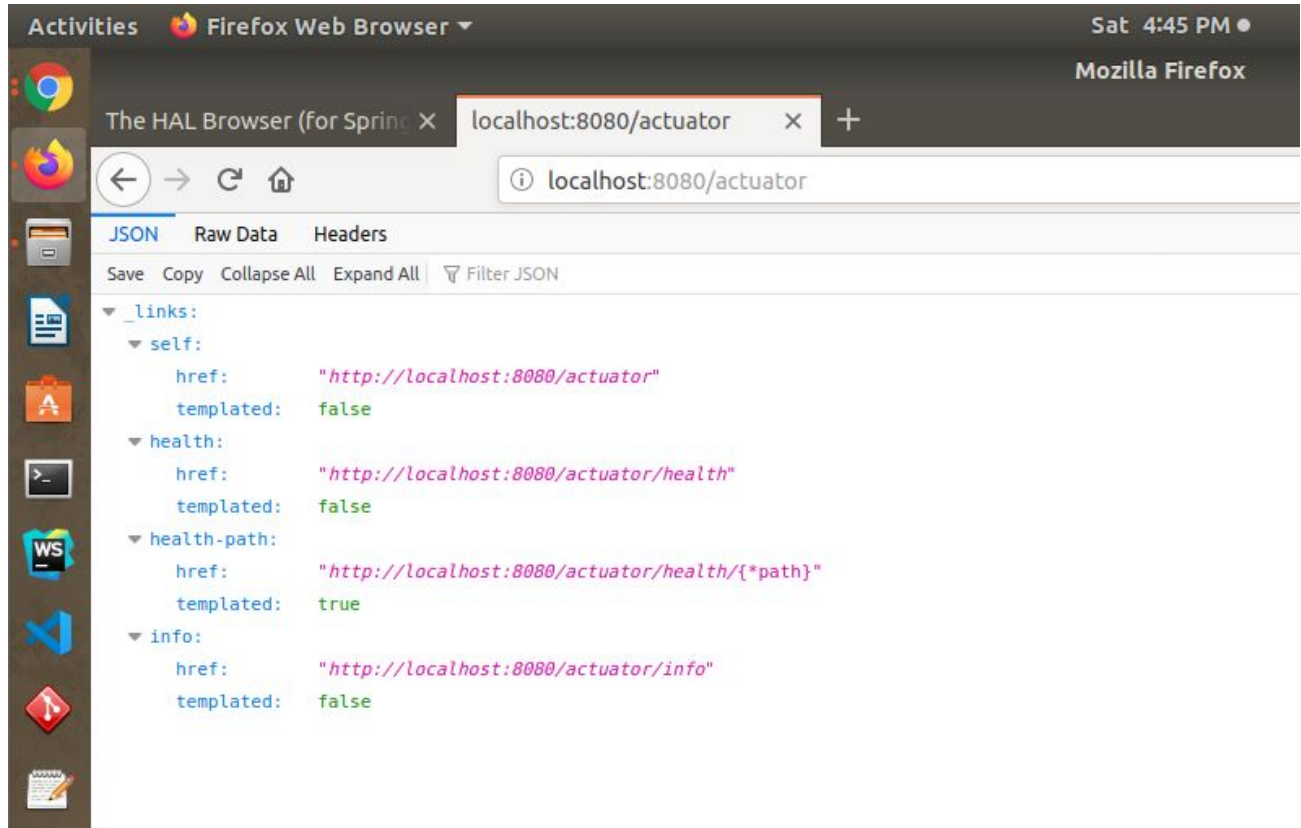
Build.gradle (dependencies)

`compile group: 'org.springframework.data', name: 'spring-data-rest-hal-browser', version: '3.2.5.RELEASE'`

`compile group: 'org.springframework.boot', name: 'spring-boot-starter-actuator', version: '2.2.5.RELEASE'`

OUTPUT

<http://localhost:8080/actuator>



<http://localhost:8080/browser/index.html#/actuator>

Activities Firefox Web Browser Sat 4:45 PM

The HAL Browser (for Spring Data REST) - Mozilla Firefox

The HAL Browser (for Spring Data REST) localhost:8080/actuator

localhost:8080/browser/index.html#/actuator

The HAL Browser (for Spring Data REST) Go To Entry Point About The HAL Browser (for Spring Data REST)

Explorer

/actuator Go!

Custom Request Headers

Properties

{ }

Links

rel	title	name / index	docs	GET	NON-GET
self				→	🚫
health				→	🚫
health-path				🔍	🚫

Inspector

Response Headers

200 success

connection: keep-alive
content-type: application/json
date: Sat, 07 Mar 2020 10:25:11 GMT
keep-alive: timeout=60
transfer-encoding: chunked

Response Body

```
{
  "_links": {
    "self": {
      "href": "http://localhost:8080/actuator",
      "templated": false
    },
    "health": {
      "href": "http://localhost:8080/actuator/health",
      "templated": false
    },
    "health-path": {
      "href": "http://localhost:8080/actuator/health/{*path}",
```