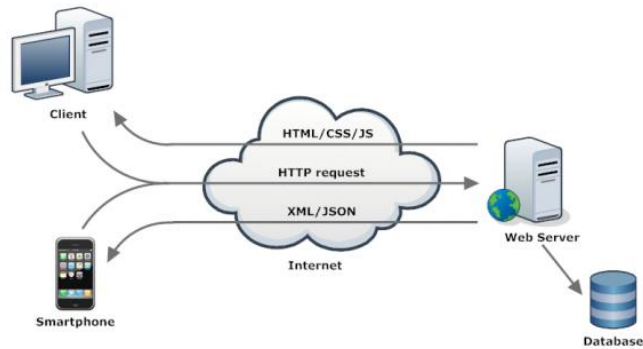




Webservices REST



1

1



1. REST

Postman

p3

p6

2. Example: Spring Boot & REST

Employee.java

LocalDateTimeSerializer

LocalDateTimeDeserializer

RestController

Error handling

RestControllerAdvice

p7

p10

p15

p16

p17

p28

p30

3. Reactive Web Client

p37

4. JUnit & Mockito & REST

p51

2

1. REST



- **RE**presentation **S**tate **T**ransfer
 - Http en het web hebben als kenmerk dat alle resources uniek benaderbaar zijn d.m.v. URI's: Resource Oriented Architecture (ROA)
- Term gelanceerd in 2000 door Roy Fielding (<http://roy.gbiv.com>)
 - Meegeschreven aan de HTTP specificaties
 - Medeoprichter van de Apache Software Foundation
- Het kernidee achter REST is dat het web zich inmiddels qua schaalbaarheid dubbel en dwars bewezen heeft en dat een ontwikkelaar met kennis over HTTP perfect aan de slag kan met REST.

3

REST

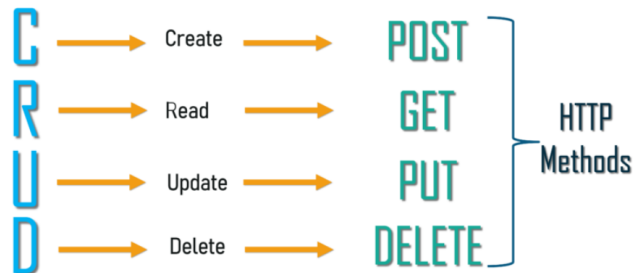


4

4

REST

- Lichter, simpeler, meestal sneller → geen speciaal formaat
- **RESTful webservices** kennen maar 4 methoden: HTTP commando's **POST**, **GET**, **PUT** en **DELETE** voor CRUD operaties (Create/Read/Update/Delete)



5

5

Postman – REST Client

<https://www.getpostman.com/apps>

The screenshot shows the Postman website's download page. The header includes navigation links like 'Product', 'Pricing', 'Enterprise', 'Resources', and 'API Network', along with 'Contact Sales', 'Sign In', and 'Sign Up for Free' buttons. The main heading is 'Download Postman', followed by a paragraph encouraging users to download the app or use the web version. Below this, there's a section titled 'The Postman app' with a 'Windows 64-bit' download button. A preview of the Postman application interface is shown on the right, displaying a workspace with a collection of API requests, including 'Retrieve a database' and 'Query a database'.

6



2. Example: Spring Boot & REST

URI	HTTP Method	Details
/rest/emp/dummy	GET	Health Check service, to insert a dummy data in the Employees data storage
/rest/emp/{id}	GET	To get the Employee object based on the id
/rest/emps	GET	To get the list of all the Employees in the data store
/rest/emp/create	POST	To create the Employee object and store it
/rest/emp/delete/{id}	DELETE	To delete the Employee object from the data storage based on the id

7

7



2. Example: Spring Boot & REST

The screenshot displays two windows from the Spring Boot IDE. The 'New Spring Starter Project' window on the left is configured with the following details:

- Service URL: `https://start.spring.io`
- Name: `Spring_Boot_rest_example1`
- Type: `Maven`, Packaging: `jar`
- Java Version: `21`, Language: `Java`
- Group: `com.springBoot`
- Artifact: `Spring_Boot_rest_example1`
- Version: `0.0.1-SNAPSHOT`
- Description: `Demo project for Spring Boot`
- Package: `com.springBoot.restExample1`
- Working sets: `SpringREST`

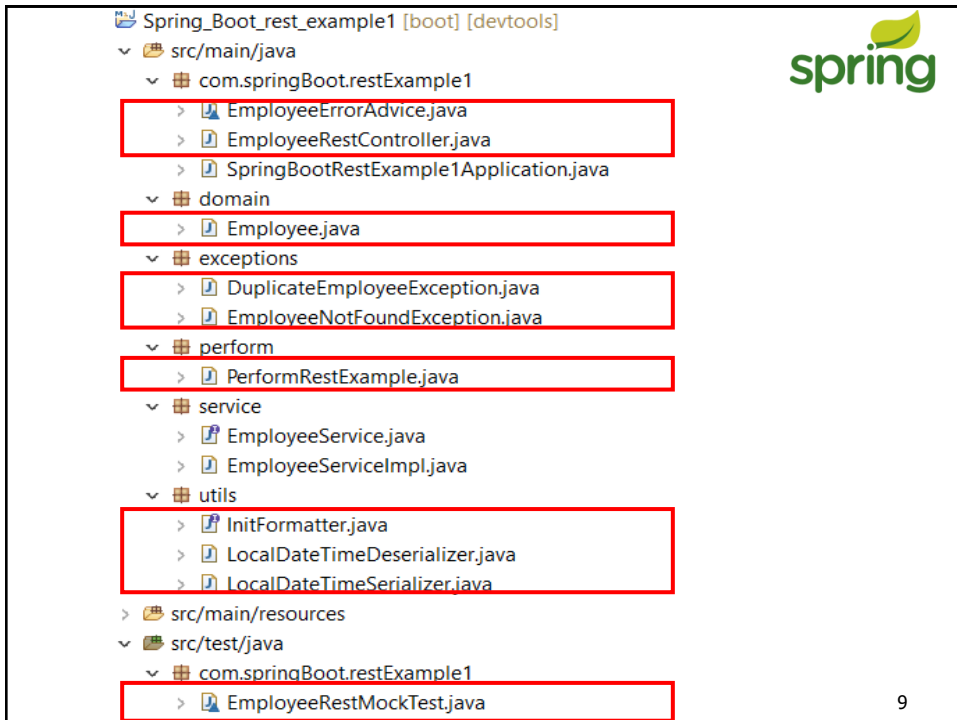
The 'New Spring Starter Project Dependencies' window on the right shows the following dependencies:

- Available:**
 - ☒ web
 - ☒ Thymeleaf
 - ☒ Spring Web
 - ☒ Spring Reactive Web
 - ☒ Spring Web Services
- Selected:**
 - ☒ Spring Boot DevTools
 - ☒ Lombok
 - ☒ Validation
 - ☒ Thymeleaf
 - ☒ Spring Web
 - ☒ Spring Reactive Web
 - ☒ Spring Web Services

A callout box highlights the selected dependencies: **X Spring Reactive Web** and **X Spring Web Services**.

8

8



9

domain
 Employee.java

A simple POJO class that will serve as **input** and **output** to our **Restful web service** methods.

```
package domain;
...
```

@Getter
@Setter
@NoArgsConstructor
@EqualsAndHashCode(exclude = "createdDateTime")
@ToString

@JsonPropertyOrder({"employee_id", "name", "createdDateTime"})

```
public class Employee implements Serializable {

    "employee_id": 9999,
    "name": "Dummy",
    "createdDateTime": "2025-03-03 19:39:25"
```

to specify the order for the fields in the JSON output.

10

10

domain

Employee.java

```
@JsonProperty("employee_id")
private int id;

private String name;

@JsonSerialize(using = LocalDateTimeSerializer.class)
@JsonDeserialize(using = LocalDateTimeDeserializer.class)
private LocalDateTime createdDateTime;

public Employee(int id, String name) {
    this.id = id;
    this.name = name;
    this.createdDateTime = LocalDateTime.now();
}
}
```

11

11

domain

Employee.java

```
@JsonProperty("employee_id")
private int id;
```

- In this example, the id field of the Employee class will be serialized as **employee_id** in the JSON output.
- When deserializing JSON input, Jackson will also look for a property named **employee_id** to set the value of the id field.

```
private String name;
```

By default, Jackson uses the field name as the JSON property name

```
"employee_id": 9999,
"name": "Dummy",
"createdDateTime": "2025-03-03 19:39:25"
```

12

12

domain

Employee.java

```
import com.fasterxml.jackson.databind.annotation.JsonDeserialize;
import com.fasterxml.jackson.databind.annotation.JsonSerialize;
import utils.LocalDateTimeDeserializer;
import utils.LocalDateTimeSerializer; → see next slide
```

@JsonSerialize(using = LocalDateTimeSerializer.class)
@JsonDeserialize(using = LocalDateTimeDeserializer.class)
private LocalDateTime createdDateTime;

- the **@JsonSerialize** annotation is used to specify a custom serializer class, **LocalDateTimeSerializer**, that will be used to convert a **LocalDateTime** object to a **JSON** string.
- the **@JsonDeserialize** annotation is used to specify a custom deserializer class, **LocalDateTimeDeserializer**, that will be used to convert a **JSON** string to a **LocalDateTime** object.

13

13

```
import utils.LocalDateTimeDeserializer;
import utils.LocalDateTimeSerializer;
```

@JsonSerialize(using = LocalDateTimeSerializer.class)
@JsonDeserialize(using = LocalDateTimeDeserializer.class)
private LocalDateTime createdDateTime;

utils

InitFormatter.java

LocalDateTimeDeserializer.java

LocalDateTimeSerializer.java

14

14

```

public interface InitFormatter {
    DateTimeFormatter FORMATTER =
        DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
}

import static utils.InitFormatter;
public class LocalDateTimeSerializer extends
        JsonSerializer<LocalDateTime> {

    @Override
    public void serialize(LocalDateTime value, JsonGenerator gen,
        SerializerProvider serializers) throws IOException {
        gen.writeString(value.format(FORMATTER));
    }
}

```

**LocalDateTimeSerializer is a custom serializer class.
This serializer is used
to convert a LocalDateTime object
to a JSON string representation
using a specified DateTimeFormatter.**

15

15

```

package utils;
...
import static utils.InitFormatter.*;
public class LocalDateTimeDeserializer
        extends JsonDeserializer<LocalDateTime> {

    @Override
    public LocalDateTime deserialize(JsonParser p,
        DeserializationContext ctxt) throws IOException, JsonProcessingException {
        String valueAsString = p.getValueAsString();
        if (valueAsString == null || valueAsString.isBlank()) {
            return null;
        }
        return LocalDateTime.parse(valueAsString, FORMATTER);
    }
}

```

- to deserialize a **JSON string to a LocalDateTime object.**
- it uses the **DateTimeFormatter** class to parse the string representation of the date-time value in the JSON string to a **LocalDateTime** object.

16

16

Spring Restful web service Controller class

```
package com.springboot.restExample1;
```

```
import java.time.LocalDateTime;
```

```
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.RequestBody;
```

```
import org.springframework.web.bind.annotation.RequestMapping;
```

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

```
import domain.Employee;
```

```
import service.EmployeeService;
```

EmployeeRestController.java

@RestController

@RequestMapping(value = "/rest")

public class EmployeeRestController {

@Autowired

private EmployeeService employeeService;

17

17

REST Web Services

EmployeeRestController.java

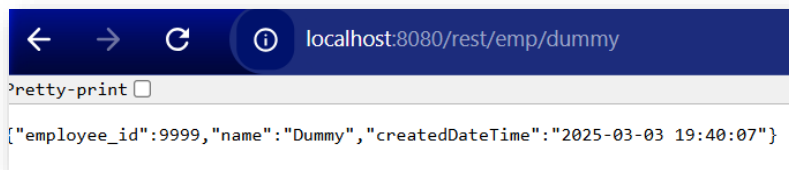
@GetMapping(value = "/emp/dummy")

public Employee getDummyEmployee() {

return employeeService.createDummyEmployee();

}

```
public Employee createDummyEmployee() {  
    Employee emp = new Employee(9999, "Dummy");  
    empData.put(9999, emp);  
    return emp;  
}
```



18

18

EmployeeRestController.java

```

@GetMapping(value = "/emp/dummy")
public Employee getDummyEmployee() {
    return employeeService.createDummyEmployee();
}

```

/rest/emp/dummy	GET	Health Check service, to insert a dummy data in the Employees data storage
-----------------	-----	--

GET http://localhost:8080/rest/emp/dummy Send Save

Body ▾ 200 OK 20 ms 236 B Save Response ▾

Pretty Raw Preview Visualize JSON i

```

1  {
2    "employee_id": 9999,
3    "name": "Dummy",
4    "createdDateTime": "2025-03-03 19:39:25"
5  }

```

19

EmployeeRestController.java

REST Web Services

```

@PostMapping(value = "/emp/create")
public Employee createEmployee(@RequestBody Employee emp) {
    emp.setCreatedDateTime(LocalDateTime.now());
    return employeeService.createEmployee(emp);
}

```

When a client sends a POST request to create an Employee, the Employee object is passed in the request body as JSON.

By using `@RequestBody`, Spring Boot is able to automatically deserialize the JSON request body into a Java object (in this case, Employee) so that you can work with it in your controller method.

20

REST Web Services

EmployeeRestController.java

```

@PostMapping(value = "/emp/create")
public Employee createEmployee(@RequestBody Employee emp) {
    emp.setCreatedDateTime(LocalDateDateTime.now());
    return employeeService.createEmployee(emp);
}

```

/rest/emp/create	POST	To create the Employee object and store it
------------------	------	--

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

- Method:** POST
- URL:** http://localhost:8080/rest/emp/create
- Headers (1):**
 - Key:** Content-Type
 - Value:** application/json
- Body:** (Empty in this view)
- Status:** 200 OK, Time: 32 ms, Size: 307 B

Annotations in the image:

- An arrow points from the `@PostMapping` annotation in the code to the POST method in the REST client.
- An arrow points from the `Content-Type` header in the code to the **Content-Type** header in the REST client.
- An arrow points from the `application/json` value in the code to the `application/json` value in the REST client header.

Content-Type

application/json

21

21

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

- Method:** POST
- URL:** http://localhost:8080/rest/emp/create
- Body:**

```

{
  "employee_id": 123,
  "name": "CreateTest"
}

```
- Status:** 200 OK, Time: 6 ms, Size: 243 B
- Response Body:**

```

{
  "employee_id": 123,
  "name": "CreateTest",
  "createdDateTime": "2025-03-03 19:45:41"
}

```

Annotations in the image:

- An arrow points from the `@RequestBody` annotation in the code to the **Body** tab in the REST client.
- An arrow points from the `JSON` dropdown in the REST client to the `JSON` dropdown in the response body.

22

22

REST Web Services
EmployeeRestController.java

```

@GetMapping(value = "/emp/{id}")
public Employee getEmployee(@PathVariable("id") int empId) {
    return employeeService.getEmployee(empId);
}

```

Method	URL	Description
GET	/rest/emp/{id}	To get the Employee object based on the id

GET
http://localhost:8080/rest/emp/9999
Send

Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings

none
form-data
x-www-form-urlencoded
raw
binary
GraphQL

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body
Cookies
Headers (5)
Test Results
Status: 200 OK
Time: 14 ms
Size: 230 B
Save

Pretty
Raw
Preview
Visualize
JSON
⌵

```

1 {
2   "employee_id": 9999,
3   "name": "Dummy",
4   "createdDateTime": "2025-03-03 19:48:07"
5 }

```

23

REST Web Services
EmployeeRestController.java

```

@GetMapping(value = "/emp/{id}")
public Employee getEmployee(@PathVariable("id") int empId) {
    return empData.get(empId);
}

```

Method	URL	Description
GET	/rest/emp/{id}	To get the Employee object based on the id

GET
http://localhost:8080/rest/emp/123
Send

Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings

none
form-data
x-www-form-urlencoded
raw
binary
GraphQL

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body
Cookies
Headers (5)
Test Results
Status: 200 OK
Time: 7 ms
Size: 234 B
Save

Pretty
Raw
Preview
Visualize
JSON
⌵

```

1 {
2   "employee_id": 123,
3   "name": "CreateTest",
4   "createdDateTime": "2025-03-03 19:45:41"
5 }

```

24

REST Web Services
EmployeeRestController.java

@GetMapping(value = "/emps")

```
public List<Employee> getAllEmployees() {
    return employeeService.getAllEmployees();
}
```

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

- Request:** GET /rest/emps. Description: To get the list of all the Employees in the data store.
- Response:** Status: 200 OK, Time: 15 ms, Size: 387 B. The response body is a JSON array of employee objects.
- JSON Response:**

```
[
  {
    "employee_id": 100,
    "name": "new employee",
    "createdDateTime": "2025-03-03 18:38:10"
  },
  {
    "employee_id": 123,
    "name": "CreateTest",
    "createdDateTime": "2025-03-03 19:45:41"
  },
  {
    "employee_id": 9999,
    "name": "Dummy",
    "createdDateTime": "2025-03-03 19:40:07"
  }
]
```

25

25

REST Web Services
EmployeeRestController.java

@DeleteMapping(value = "/emp/delete/{id}")

```
public Employee deleteEmployee(@PathVariable("id") int empId) {
    return employeeService.deleteEmployee(empId);
}
```

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

- Request:** DELETE /rest/emp/delete/123. Description: To delete the Employee object from the data storage based on the id.
- Response:** Status: 200 OK, Time: 5 ms, Size: 234 B. The response body is a JSON object representing the deleted employee.
- JSON Response:**

```
{
  "employee_id": 123,
  "name": "CreateTest",
  "createdDateTime": "2025-03-03 19:45:41"
}
```

26

26

```
public List<Employee> getAllEmployees() {  
    return employeeService.getAllEmployees();  
}
```

/rest/emps	GET	To get the list of all the Employees in the data store
------------	-----	--



Error handling



14

Error handling

exceptions
DuplicateEmployeeException.java
EmployeeNotFoundException.java

```
package exceptions;

public class EmployeeNotFoundException extends RuntimeException {

    private static final long serialVersionUID = 1L;

    public EmployeeNotFoundException(Integer id) {
        super(String.format("Could not find employee %s", id));
    }
}
```

```
package exceptions;

public class DuplicateEmployeeException extends RuntimeException {

    private static final long serialVersionUID = 1L;

    public DuplicateEmployeeException(Integer id) {
        super(String.format("Duplicate employee %s", id));
    }
}
```

29

29

Error handling

src/main/java
com.springboot.restExample1
EmployeeErrorAdvice.java

```
package com.springboot.restExample1;
```

```
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestControllerAdvice;
import exceptions.DuplicateEmployeeException;
import exceptions.EmployeeNotFoundException;
```

@RestControllerAdvice

```
class EmployeeErrorAdvice {
```

When an exception is thrown from a **@RestController** method, the **@RestControllerAdvice** class can handle the exception and return an appropriate response.

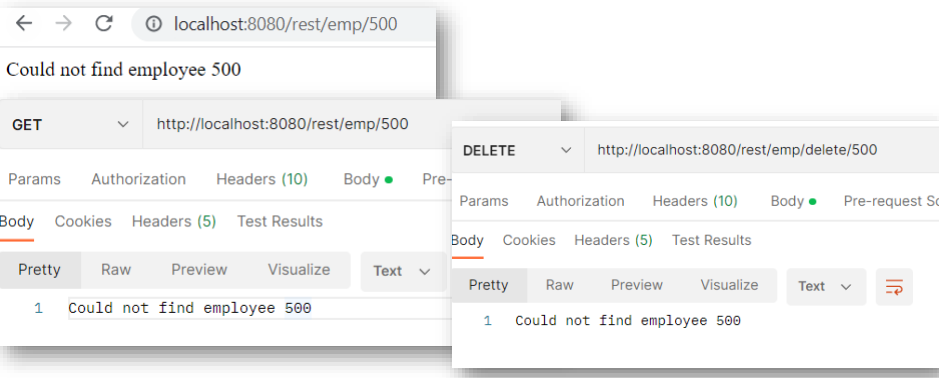
This can be especially useful for **returning error messages** and **status codes** in a consistent format across an entire application.

30

30

Error handling

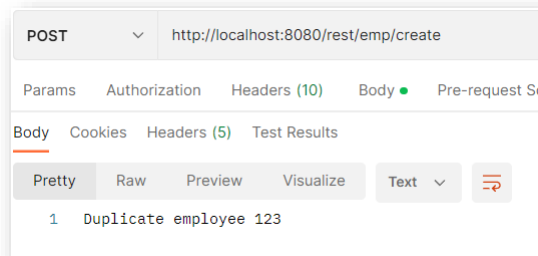
```
@ResponseBody
@ExceptionHandler(EmployeeNotFoundException.class)
@ResponseStatus(HttpStatus.NOT_FOUND)
String employeeNotFoundHandler(
    EmployeeNotFoundException ex) {
    return ex.getMessage();
}
```



31

Error handling

```
@ResponseBody
@ExceptionHandler(DuplicateEmployeeException.class)
@ResponseStatus(HttpStatus.CONFLICT)
String duplicateEmployeeHandler(
    DuplicateEmployeeException ex) {
    return ex.getMessage();
}
```



32

32

Error handling

EmployeeRestController.java

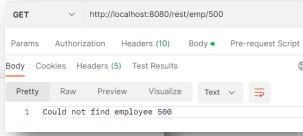
```
@GetMapping(value = "/emp/{id}")
public Employee getEmployee(@PathVariable("id") int empId) {
    return employeeService.getEmployee(empId);
}
```

EmployeeServiceImpl.java

```
public Employee getEmployee(int empId){
    Employee employee = empData.get(empId);
    if (employee == null)
        throw new EmployeeNotFoundException(empId);
    return employee;
}
```

EmployeeErrorAdvice.java

```
@ResponseBody
@ExceptionHandler(EmployeeNotFoundException.class)
@ResponseStatus(HttpStatus.NOT_FOUND)
String employeeNotFoundHandler(EmployeeNotFoundException ex) {
    return ex.getMessage();
}
```



33

33

Error handling

EmployeeRestController.java

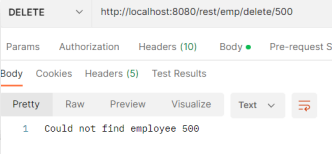
```
@DeleteMapping(value = "/emp/delete/{id}")
public Employee deleteEmployee(@PathVariable("id") int empId) {
    return employeeService.deleteEmployee(empId);
}
```

EmployeeServiceImpl.java

```
public Employee deleteEmployee(int empId) {
    Employee employee = empData.get(empId);
    if (employee == null)
        throw new EmployeeNotFoundException(empId);
    empData.remove(empId);
    return employee;
}
```

EmployeeErrorAdvice.java

```
@ResponseBody
@ExceptionHandler(EmployeeNotFoundException.class)
@ResponseStatus(HttpStatus.NOT_FOUND)
String employeeNotFoundHandler(EmployeeNotFoundException ex) {
    return ex.getMessage();
}
```



34

34

Error handling

```

EmployeeRestController.java
@PostMapping(value = "/emp/create")
public Employee createEmployee(@RequestBody Employee emp) {
    emp.setCreatedDateTime(LocalDateTime.now());
    return employeeService.createEmployee(emp);
}

EmployeeServiceImpl.java

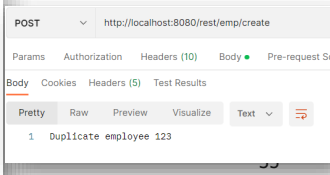
public Employee createEmployee(Employee emp) {
    if (empData.containsKey(emp.getId()))
        throw new DuplicateEmployeeException(emp.getId());
    empData.put(emp.getId(), emp);
    return emp;
}

EmployeeErrorAdvice.java
package exceptions;

public class DuplicateEmployeeException extends RuntimeException {
    private static final long serialVersionUID = 1L;

    public DuplicateEmployeeException(Integer id) {
        super(String.format("Duplicate employee %s", id));
    }
}

```



35

```


package com.springboot.restExample1;
...
import perform.PerformRestExample;

@SpringBootApplication
public class SpringBootRestExample1Application
    implements WebMvcConfigurer {

    public static void main(String[] args){
        SpringApplication.run(
            SpringBootRestExample1Application.class, args);

        try {
            new PerformRestExample();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    ...

```



36

36

3. Reactive Web Client



```
Spring_Boot_rest_example1 [boot] [devtools]
├── src/main/java
│   ├── com.springboot.restExample1
│   │   ├── EmployeeErrorAdvice.java
│   │   ├── EmployeeRestController.java
│   │   └── SpringBootRestExample1Application.java
│   ├── domain
│   │   └── Employee.java
│   ├── exceptions
│   │   ├── DuplicateEmployeeException.java
│   │   └── EmployeeNotFoundException.java
│   └── perform
│       └── PerformRestExample.java
├── service
│   ├── EmployeeService.java
│   └── EmployeeServiceImpl.java
├── utils
│   ├── InitFormatter.java
│   ├── LocalDateTimeDeserializer.java
│   └── LocalDateTimeSerializer.java
├── src/main/resources
└── src/test/java
    ├── com.springboot.restExample1
    └── EmployeeRestMockTest.java
```

Rest Clients are good to test our rest web service but most of the times, we need to invoke rest services through our program. We can use Spring **WebClient** to invoke these methods easily.

WebClient: a reactive web client that's being introduced in Spring 5

PerformRestExample is a simple program invoking our application rest methods using WebClient.

37

37

Reactive Web Client



```
package perform;
import java.util.LinkedHashMap;
import java.util.List;
import org.springframework.http.MediaType;
import org.springframework.web.reactive.function.BodyInserters;
import org.springframework.web.reactive.function.client.WebClient;
import com.fasterxml.jackson.databind.ObjectMapper;
import domain.Employee;

public class PerformRestExample {

    private final String SERVER_URI = "http://localhost:8080/rest";

    private WebClient webClient = WebClient.create();
```

38

38

```

public TestRestExample() {public PerformRestExample()
                                throws Exception {

    System.out.println("\n----- GET ALL -----");
    getAllEmployee();
    System.out.println("----- GET DUMMY ----- ");
    getDummyEmployee();
    System.out.println("\n----- GET 9999 ----- ");
    getEmployee("9999");
    System.out.println("\n----- INSERT 100 ----- ");
    insertEmployee(100, "new employee");
    try {
        System.out.println("\n----- INSERT 100 second time ----- ");
        insertEmployee(100, "new employee");
    }catch(Exception e){
        System.out.println(e.getMessage());
    }

    System.out.println("\n----- GET ALL -----");
    getAllEmployee();

```



39

39

```

    System.out.println("\n----- DELETE 9999 -----");
    deleteEmployee("9999");
    System.out.println("\n----- GET ALL -----");
    getAllEmployee();
    try{
        System.out.println("\n----- GET 9998 ----- ");
        getEmployee("9998");
    }catch(Exception e){
        System.out.println(e.getMessage());
    }
    try{
        System.out.println("\n----- DELETE 9998 ----- ");
        deleteEmployee("9998");
    }catch(Exception e){
        System.out.println(e.getMessage());
    }
}
}
private void printEmpData(Employee emp) {
    System.out.printf("ID=%0s, Name=%0s, CreatedDateTime=%0s%n",
        emp.getId(), emp.getName(), emp.getCreatedDateTime());}

```



40

40

```
System.out.println("\n----- GET ALL -----");
getAllEmployee();
```

```
private void getAllEmployee() {
    WebClient.get().uri(SERVER_URI + "/emps")
        .retrieve()
        .bodyToFlux(Employee.class)
        .flatMap(emp -> {
            printEmpData(emp);
            return Mono.empty();
        })
        .blockLast();
}
```

REST Web Services

retrieve a stream of Employee objects from the response body using **bodyToFlux(Employee.class)**

ignore the results and return an empty Mono

is used to obtain the last element from a Flux

----- GET ALL -----

41

```
System.out.println("----- GET DUMMY -----");
getDummyEmployee();
```

```
private void getDummyEmployee() {
    getAnEmployee(SERVER_URI + "/emp/dummy");
}
private void getAnEmployee(String uri){
    WebClient.get().uri(uri)
        .retrieve()
        .bodyToMono(Employee.class)
        .doOnSuccess(emp -> printEmpData(emp))
        .block();
}
```

REST Web Services

retrieve method is the shortest path to fetching a body directly.

block method on **Monos** to subscribe and retrieve an actual data which was sent with the response.

```
----- GET DUMMY -----
ID=9999, Name=Dummy, CreatedDateTime=2025-03-03T18:38:10
```

42

```
System.out.println("\n----- GET 9999 ----- ");  
getEmployee("9999");
```

```
private void getEmployee(String number) {  
    getAnEmployee(SERVER_URI + "/emp/" + number);  
}  
  
private void getAnEmployee(String uri){  
    webClient.get().uri(uri).retrieve()  
    .bodyToMono(Employee.class)  
    .doOnSuccess(emp -> printEmpData(emp))  
    .block();  
}
```

```
----- GET 9999 -----  
ID=9999, Name=Dummy, CreatedDateTime=2025-03-03T18:38:10
```

43

```
System.out.println("\n----- INSERT 100 ----- ");  
insertEmployee("100", "new employee");
```

```
private void insertEmployee(String id, String name) {  
  
    Employee emp = new Employee(id, name);  
  
    webClient.post().uri(SERVER_URI + "/emp/create")  
    .contentType(MediaType.APPLICATION_JSON).  
    .body(BodyInserters.fromValue(emp))  
    .retrieve().bodyToMono(Employee.class).block();  
  
}
```

```
----- INSERT 100 -----
```

44

```
try{
    System.out.println("\n----- INSERT 100 second time ----- ");
    insertEmployee(100, "new employee");
}
catch(Exception e){
    System.out.println(e.getMessage());
}
```

```
----- INSERT 100 second time -----
2025-03-03T18:38:10.563+01:00 WARN 18384 --- [nio-8080-exec-2] .m.m.a.ExceptionHandlerExceptionHandlerResolver :
409 Conflict from POST http://localhost:8080/rest/emp/create
```

```
Resolved [exceptions.DuplicateEmployeeException: Duplicate employee 100]
```

45

```
System.out.println("\n----- GET ALL -----");
getAllEmployee();
```

```
private void getAllEmployee() {
    webClient.get().uri(SERVER_URI + "/emps")
        .retrieve()
        .bodyToFlux(Employee.class)
        .flatMap(emp -> {
            printEmpData(emp);
            return Mono.empty();
        })
        .blockLast();
}
```

```
----- GET ALL -----
ID=100, Name=new employee, CreatedDateTime=2025-03-03T18:38:10
ID=9999, Name=Dummy, CreatedDateTime=2025-03-03T18:38:10
```

46

```
System.out.println("\n----- DELETE 9999 -----");  
deleteEmployee("9999");
```

```
private void deleteEmployee(String number) {  
    webClient.delete().  
        uri(SERVER_URI + "/emp/delete/"+number).  
        retrieve().bodyToMono(Employee.class).block();  
}
```

```
----- DELETE 9999 -----
```

47

```
System.out.println("\n----- GET ALL -----");  
getAllEmployee();
```

```
----- GET ALL -----  
ID=100, Name=new employee, CreatedDateTime=2025-03-03T18:38:10
```

48


```
try{
    System.out.println("\n----- GET 9998 ----- ");
    getEmployee("9998");
}catch(Exception e){
    System.out.println(e.getMessage());
}
```

```
----- GET 9998 -----
2025-03-03T18:38:10.573+01:00 WARN 18384 --- [nio-8080-exec-8] .m.m.a.ExceptionHandlerExceptionResolver :
404 Not Found from GET http://localhost:8080/rest/emp/9998
```

Resolved [[exceptions.EmployeeNotFoundException](#): Could not find employee 9998]

49

```
try{
    System.out.println("\n----- DELETE 9998 ----- ");
    deleteEmployee("9998");
}catch(Exception e){
    System.out.println(e.getMessage());
}
```

```
----- DELETE 9998 -----
2025-03-03T18:38:10.576+01:00 WARN 18384 --- [nio-8080-exec-7] .m.m.a.ExceptionHandlerExceptionResolver :
404 Not Found from DELETE http://localhost:8080/rest/emp/delete/9998
```

Resolved [[exceptions.EmployeeNotFoundException](#): Could not find employee 9998]

50

4. JUnit, Mockito and REST



```
Spring_Boot_rest_example1 [boot] [devtools]
├── src/main/java
│   ├── com.springboot.restExample1
│   │   ├── EmployeeErrorAdvice.java
│   │   ├── EmployeeRestController.java
│   │   └── SpringBootRestExample1Application.java
│   ├── domain
│   │   └── Employee.java
│   ├── exceptions
│   │   ├── DuplicateEmployeeException.java
│   │   └── EmployeeNotFoundException.java
│   ├── perform
│   │   └── PerformRestExample.java
│   ├── service
│   │   ├── EmployeeService.java
│   │   └── EmployeeServiceImpl.java
│   └── utils
│       ├── InitFormatter.java
│       ├── LocalDateTimeDeserializer.java
│       └── LocalDateTimeSerializer.java
├── src/main/resources
└── src/test/java
    ├── com.springboot.restExample1
    │   └── EmployeeRestMockTest.java
```

This is a test class for a Spring REST API.

It is annotated with `@SpringBootTest`.

It also uses Mockito to mock an `EmployeeService` instance.

51

51

Runs: 9/9 Errors: 0 Failures: 0

EmployeeRestMockTest [Runner: JUnit 5] (0,390 s)

testDeleteEmployee() (0,253 s)

testGetAllEmployees_emptyList() (0,015 s)

testGetEmployee_notFound() (0,015 s)

testGetEmployee_isOk() (0,011 s)

testDummyEmployee_isOk() (0,012 s)

testGetAllEmployees_noEmptyList() (0,012 s)

testCreateEmployee() (0,042 s)

testCreateEmployee_duplicateKey() (0,011 s)

testDeleteEmployee_notFound() (0,011 s)

testDummyEmployee_isOk() (0,012 s)

testGetEmployee_notFound() (0,015 s)

testGetEmployee_isOk() (0,011 s)

testCreateEmployee() (0,042 s)

testCreateEmployee_duplicateKey() (0,011 s)

testDeleteEmployee() (0,253 s)

testDeleteEmployee_notFound() (0,011 s)

testGetAllEmployees_emptyList() (0,015 s)

testGetAllEmployees_noEmptyList() (0,012 s)

52

```

...
import static utils.InitFormatter.*;
@SpringBootTest
class EmployeeRestMockTest {

    @Mock
    private EmployeeService mock;

    private EmployeeRestController controller;
    private MockMvc mockMvc;

    private final int ID = 1234;
    private final String NAME = "Test";
    private String expectedFormattedDateTime;

```

53

```

@BeforeEach
public void before() {
    MockitoAnnotations.openMocks(this);
    controller = new EmployeeRestController();
    mockMvc = standaloneSetup(controller).build();
    ReflectionTestUtils.setField(
        controller, "employeeService", mock);
}

```

```

@RestController
@RequestMapping(value = "/rest")
public class EmployeeRestController {

    @Autowired
    private EmployeeService employeeService;
}

```



54

54

```

@Test
public void testDummyEmployee_isOk() throws Exception {
    Mockito.when(mock.createDummyEmployee())
        .thenReturn(anEmployee(ID, NAME));
    performRest("/rest/emp/dummy");
    Mockito.verify(mock).createDummyEmployee();
}

private Employee anEmployee(int id, String name) {
    Employee emp = new Employee(id, name);
    expectedFormattedDateTime =
        emp.getCreatedDateTime().format(FORMATTER);
    return emp;
}

public interface InitFormatter {
    DateTimeFormatter FORMATTER =
        DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
}

```

55

```

@Test
public void testDummyEmployee_isOk() throws Exception {
    Mockito.when(mock.createDummyEmployee())
        .thenReturn(anEmployee(ID, NAME));
    performRest("/rest/emp/dummy");
    Mockito.verify(mock).createDummyEmployee();
}

private void performRest(String uri) throws Exception {
    mockMvc.perform(get(uri))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.employee_id").value(ID))
        .andExpect(jsonPath("$.name").value(NAME))
        .andExpect(jsonPath("$.createdDateTime")
            .value(expectedFormattedDateTime));
}

public class Employee implements Serializable {
    @JsonProperty("employee_id")
    private int id;

    private String name;

    @JsonSerialize(using = LocalDateTimeSerializer.class)
    @JsonDeserialize(using = LocalDateTimeDeserializer.class)
    private LocalDateTime createdDateTime;
}

```

56

testGetEmployee_isOk()

```
@Test
public void testGetEmployee_isOk() throws Exception {
    Mockito.when(mock.getEmployee(ID)).
        thenReturn(anEmployee(ID, NAME));
    performRest("/rest/emp/" + ID);
    Mockito.verify(mock).getEmployee(ID);
}
```

57

testGetEmployee_notFound()

```
@Test
public void testGetEmployee_notFound() throws Exception {
    Mockito.when(mock.getEmployee(ID)).
        thenThrow(new EmployeeNotFoundException(ID));

    Exception exception = assertThrows(Exception.class, () -> {
        mockMvc.perform(get("/rest/emp/" + ID)).andReturn();
    });

    assertTrue(exception.getCause() instanceof
        EmployeeNotFoundException);
    Mockito.verify(mock).getEmployee(ID);
}
```

it uses **MockMvc** to perform a GET request to the **/rest/emp/{id}** endpoint.

The test asserts that an **Exception** will be thrown with the **root cause** being an **EmployeeNotFoundException**.

58

@Test

testCreateEmployee()

```
public void testCreateEmployee() throws Exception {  
    Employee emp = new Employee(ID, NAME);  
    String empJson =  
        new ObjectMapper().writeValueAsString(emp);
```

- an instance of the Employee class is created with an ID and NAME.
- the **ObjectMapper** class is used to convert this Employee object into a **JSON string** using the **writeValueAsString** method.

```
Mockito.when(mock.createEmployee(  
    Mockito.any(Employee.class))).thenReturn(emp);
```

59

testCreateEmployee()

```
mockMvc.perform(post("/rest/emp/create")  
    .contentType(MediaType.APPLICATION_JSON)  
    .content(empJson))  
    .andExpect(status().isOk())  
    .andExpect(jsonPath("$.employee_id").value(ID))  
    .andExpect(jsonPath("$.name").value(NAME))  
    .andExpect(jsonPath("$.createdDateTime").isNotEmpty());
```

```
Mockito.verify(mock)  
    .createEmployee(Mockito.any(Employee.class));
```

```
}
```

- using the **MockMvc** class, an **HTTP POST** request is sent to the createEmployee endpoint with the **empJson** string in the request body and the MediaType set to **APPLICATION_JSON**.

60

@Test testCreateEmployee_duplicateKey()

```

public void testCreateEmployee_duplicateKey() throws
Exception {
    String empJson = new ObjectMapper()
        .writeValueAsString(new Employee(ID, NAME));
    Mockito.when(mock.createEmployee(
        Mockito.any(Employee.class))).
        thenThrow(new DuplicateEmployeeException(ID));

    Exception exception = assertThrows(Exception.class, () -> {
        mockMvc.perform(post("/rest/emp/create")
            .contentType(MediaType.APPLICATION_JSON)
            .content(empJson))
            .andReturn(); >});

    assertTrue(exception.getCause()
        instanceof DuplicateEmployeeException);
    Mockito.verify(mock).
        createEmployee(Mockito.any(Employee.class));
}

```

The andReturn() method is used to return the result of the performed action in the MockMvc object.

61

@Test testDeleteEmployee()

```

public void testDeleteEmployee() throws Exception {
    Mockito.when(mock.deleteEmployee(ID)).
        thenReturn(anEmployee(ID, NAME));

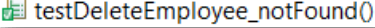
    mockMvc.perform(delete("/rest/emp/delete/" + ID))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.employee_id").value(ID))
        .andExpect(jsonPath("$.name").value(NAME))
        .andExpect(jsonPath("$.createdDateTime")
            .value(expectedFormattedDateTime));

    Mockito.verify(mock).deleteEmployee(ID);
}

```

The test expects the response status to be 200 OK, and checks that the returned **JSON object** contains the expected values for the deleted employee's id, name, and createdDateTime fields. Finally, it verifies that the **deleteEmployee()** method of the mocked service was called with the correct ID argument.

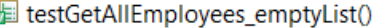
62

 `@Test`
`public void testDeleteEmployee_notFound() throws Exception`
`{`
`Mockito.when(mock.deleteEmployee(ID)).`
`thenThrow(new EmployeeNotFoundException(ID));`
`Exception exception = assertThrows(Exception.class, () -> {`
`mockMvc.perform(delete("/rest/emp/delete/" + ID))`
`.andReturn();`
`});`
`assertTrue(exception.getCause()`
`instanceof EmployeeNotFoundException);`
`Mockito.verify(mock).deleteEmployee(ID);`
`}`

It uses the **mockMvc** framework to perform a DELETE request to the `"/rest/emp/delete/{id}"` endpoint with the specified ID.

It expects an **exception** to be thrown, and asserts that the exception is an instance of **EmployeeNotFoundException**.

63

 `@Test`
`public void testGetAllEmployees_emptyList() throws Exception`
`{`
`Mockito.when(mock.getAllEmployees())`
`.thenReturn(new ArrayList<>());`
`mockMvc.perform(get("/rest/emps"))`
`.andExpect(status().isOk())`
`.andExpect(jsonPath("$").isArray())`
`.andExpect(jsonPath("$").isEmpty());`
`Mockito.verify(mock).getAllEmployees();`
`}`

It sends a GET request to the endpoint `/rest/emps`, and then verifies that the response status is `isOk()` (i.e., HTTP status code 200) and **the response body is an empty JSON array**.

64

@Test

```
public void testGetAllEmployees_noEmptyList() throws Exception {  
    Employee employee1 = anEmployee(ID, NAME);  
    String expectedFormattedDateTime1 =  
        expectedFormattedDateTime;
```

```
    Employee employee2 = anEmployee(5678, "Test2");  
    String expectedFormattedDateTime2 =  
        expectedFormattedDateTime;
```

```
    List<Employee> listEmployee =  
        List.of(employee1, employee2);
```

```
    Mockito.when(mock.getAllEmployees())  
        .thenReturn(listEmployee);
```

```
private Employee anEmployee(int id, String name) {  
    Employee emp = new Employee(id, name);  
    expectedFormattedDateTime = emp.getCreatedDateTime().format(FORMATTER);  
    return emp;  
}
```

65

```
mockMvc.perform(get("/rest/emps"))  
    .andExpect(status().isOk())  
    .andExpect(jsonPath("$").isArray())  
    .andExpect(jsonPath("$").isEmpty())  
  
    .andExpect(jsonPath("$[0].employee_id").value(ID))  
    .andExpect(jsonPath("$[0].name").value(NAME))  
  
    .andExpect(jsonPath("$[0].createdDateTime")  
        .value(expectedFormattedDateTime1))  
  
    .andExpect(jsonPath("$[1].employee_id").value(5678))  
    .andExpect(jsonPath("$[1].name").value("Test2"))  
    .andExpect(jsonPath("$[1].createdDateTime")  
        .value(expectedFormattedDateTime2));  
  
Mockito.verify(mock).getAllEmployees();  
}
```

66