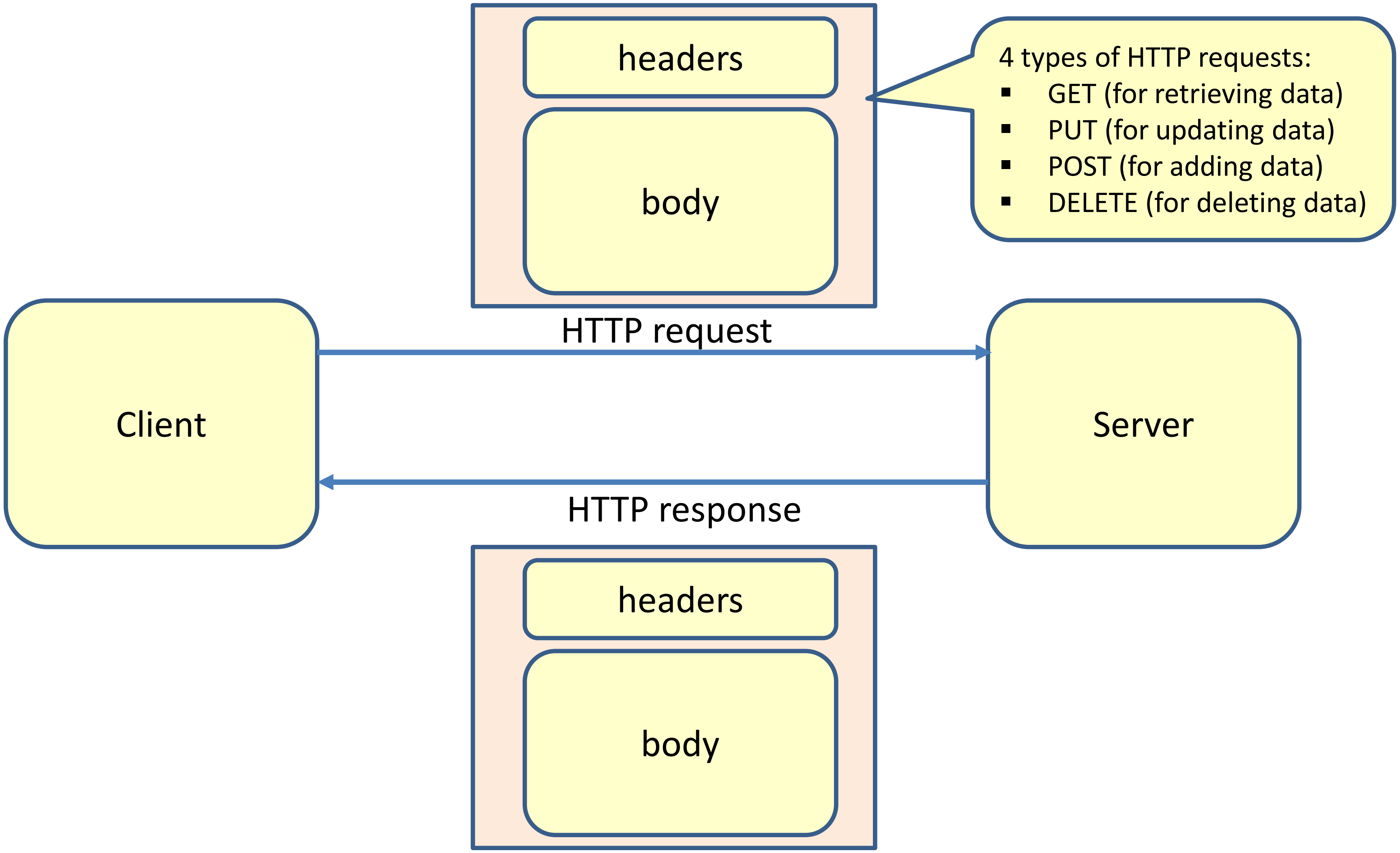
CS544

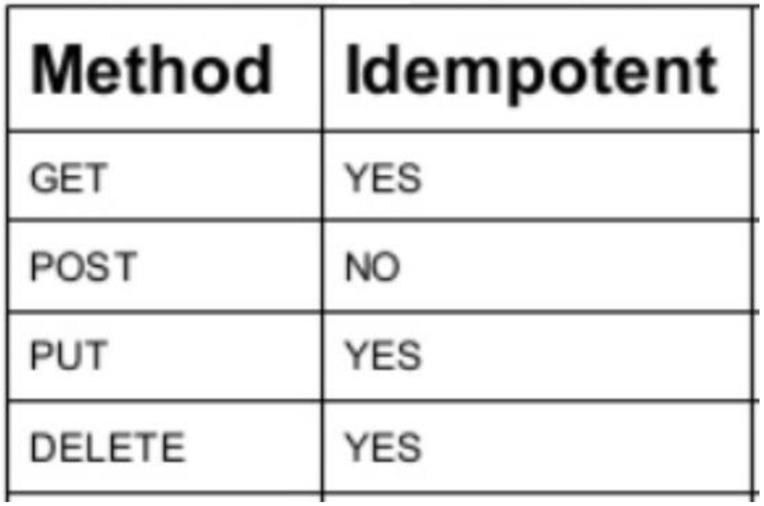
LESSON 9
REST WEBSERVICES

REST webservices



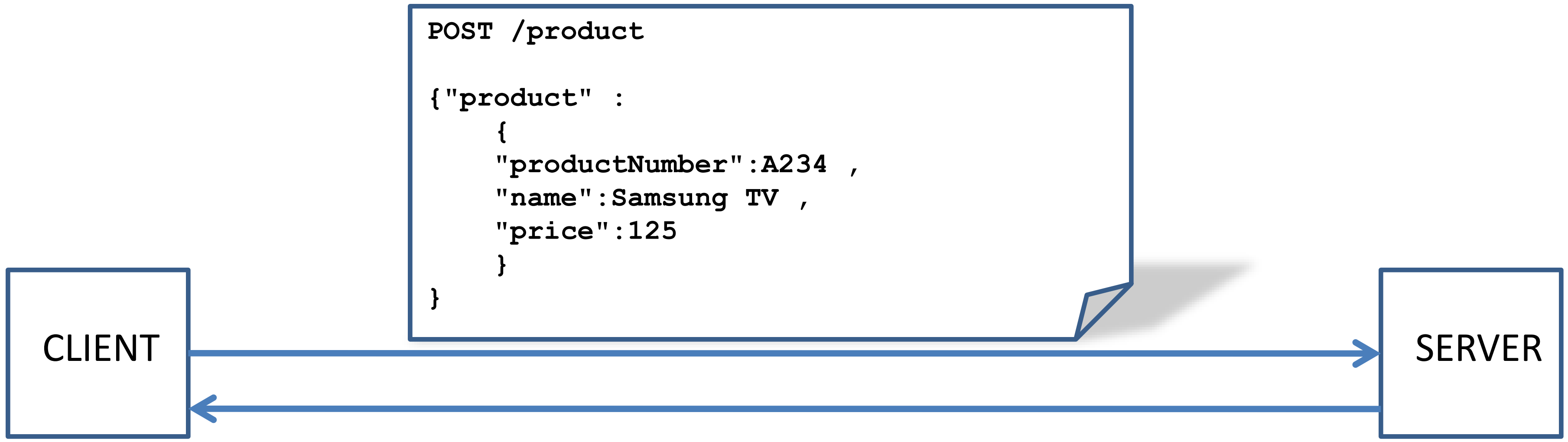
4 types of HTTP requests:
- GET (for retrieving data)
- PUT (for updating data)
- POST (for adding data)
- DELETE (for deleting data)

Http methods



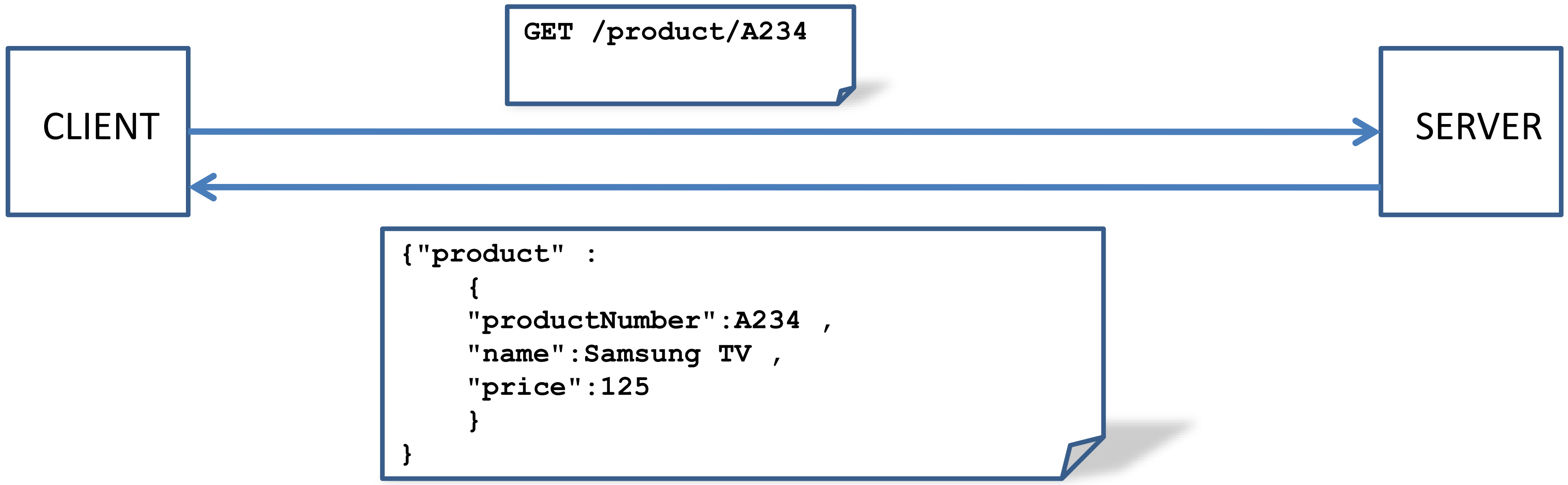
Method	Idempotent
GET	YES
POST	NO
PUT	YES
DELETE	YES

POST method using JSON



```
POST /product

{"product" :
     {
     "productNumber":A234 ,
     "name":Samsung TV ,
     "price":125
     }
}
```

GET method using JSON



```
GET /product/A234
```

```
{"product" :
    {
    "productNumber":A234 ,
    "name":Samsung TV ,
    "price":125
    }
}
```

Spring REST libraries

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

Simple Rest Example: the controller

```
@RestController
public class GreetingController {

    @RequestMapping("/greeting")
    public String greeting() {
        return "Hello World";
    }
}
```

@RestController tells Spring that this class is a controller that is called by sending HTTP REST requests, and that returns HTTP response messages

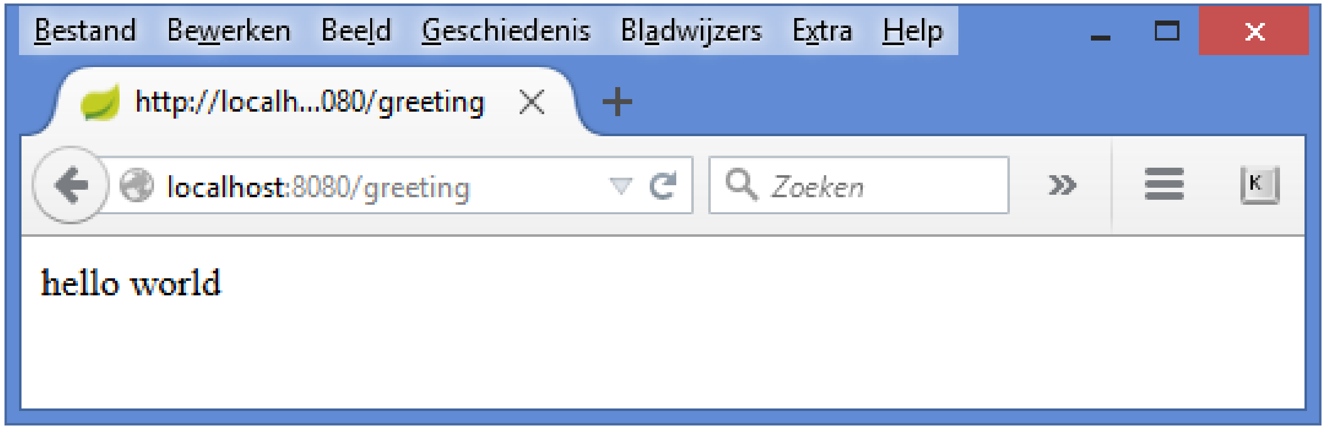
The URL to call this method ends with /greeting

Simple Rest Example: configuration

One annotations is same as these 3 together
@Configuration
@EnableConfiguration
@ComponentScan

```
@SpringBootApplication
public class GreetingRestApplication {

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

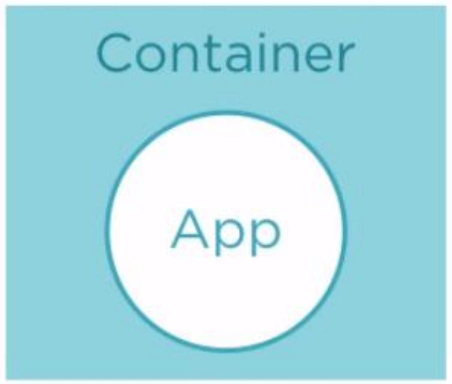
Simple Rest Example: calling the service

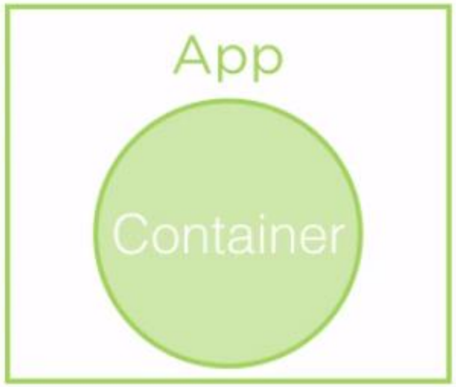

Bestand Bewerken Beeld Geschiedenis Bladwijzers Extra Help
http://localh...080/greeting
localhost:8080/greeting
Zoeken

hello world

```
@RestController
public class GreetingController {

    @RequestMapping("/greeting")
    public String greeting() {
        return "Hello World";
    }
}
```

Containerless deployment





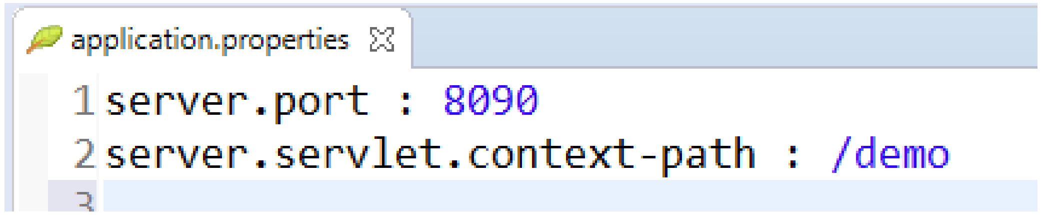
Container Deployments

- Pre-setup and configuration
- Need to use files like web.xml to tell container how to work
- Environment configuration is external to your application

Application Deployments

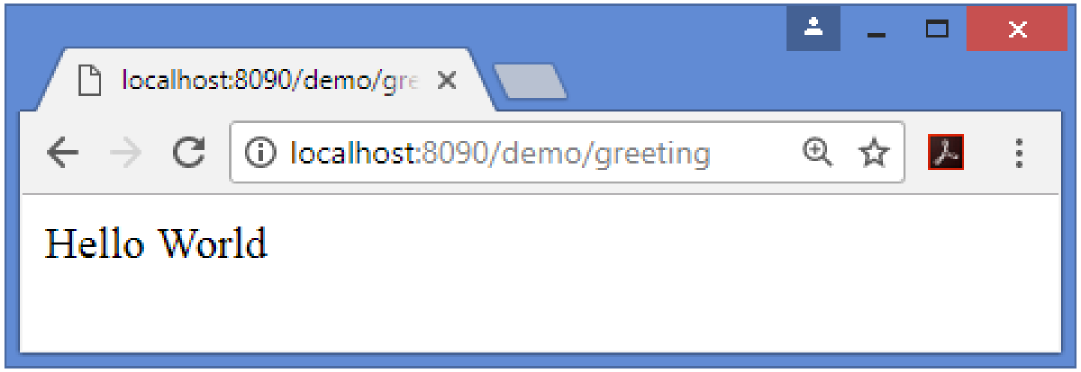
- Runs anywhere Java is setup (think cloud deployments)
- Container is embedded and the app directs how the container works
- Environment configuration is internal to your application

Configuration with application.properties



application.properties

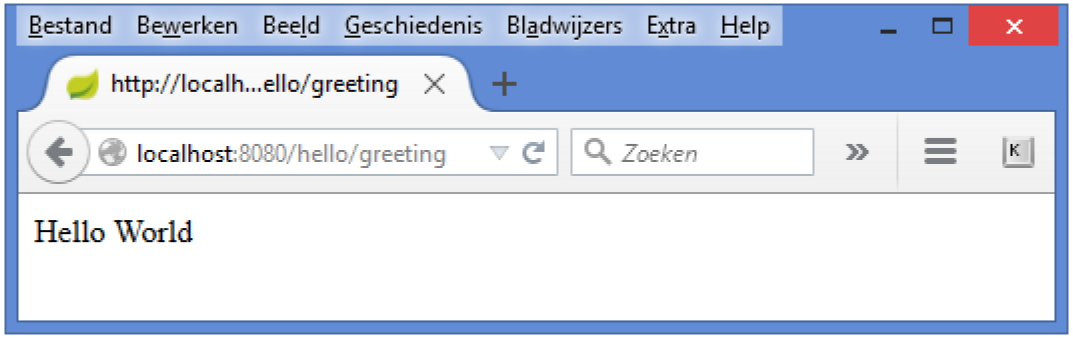
```
server.port : 8090
server.servlet.context-path : /demo
```



localhost:8090/demo/greeting

```
Hello World
```

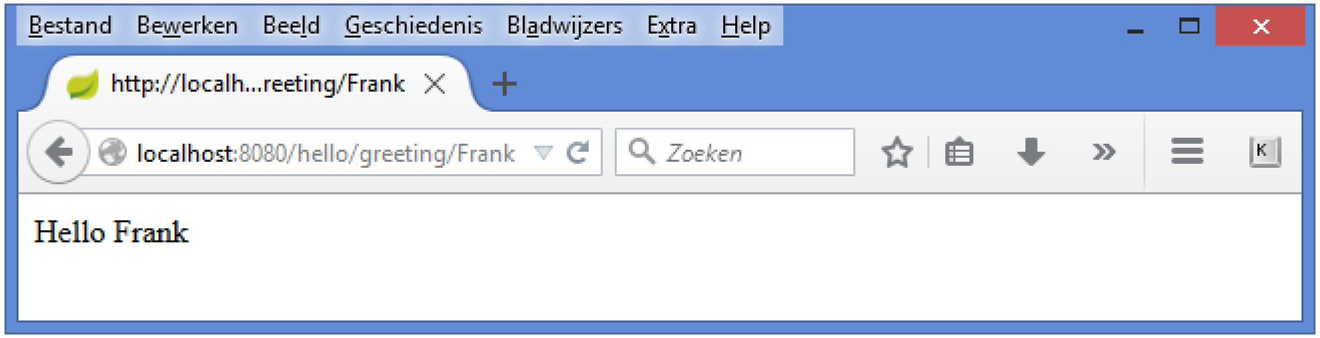
Different URL


Bestand Bewerken Beeld Geschiedenis Bladwijzers Extra Help
http://localh...ello/greeting
localhost:8080/hello/greeting
Zoeken

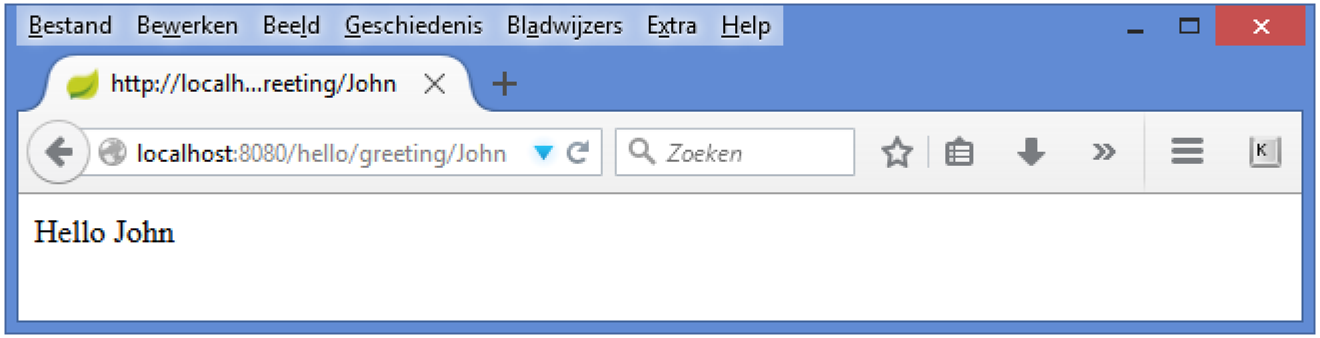
Hello World

```
@RestController
@RequestMapping("/hello")
public class GreetingController {

    @RequestMapping(value="/greeting")
    public  String greetingJSON() {
        return "Hello World";
    }
}
```

Path variables


Bestand Bewerken Beeld Geschiedenis Bladwijzers Extra Help
http://localh...reeting/Frank
localhost:8080/hello/greeting/Frank
Zoeken

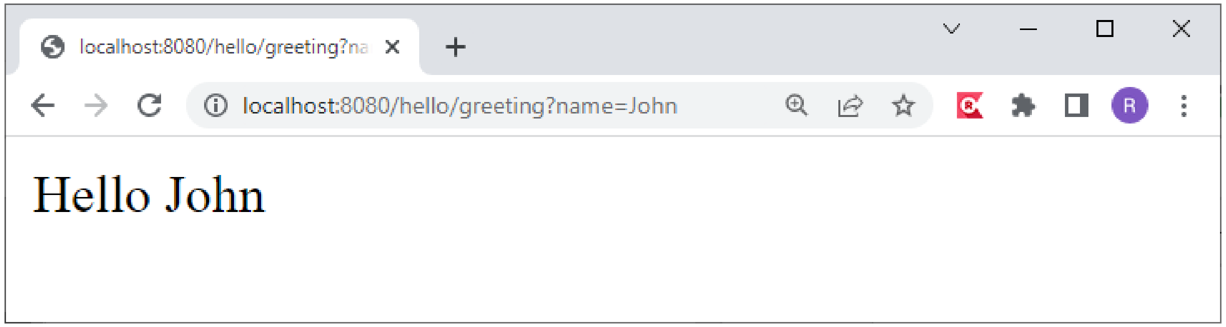
Hello Frank


Bestand Bewerken Beeld Geschiedenis Bladwijzers Extra Help
http://localh...reeting/John
localhost:8080/hello/greeting/John
Zoeken

Hello John

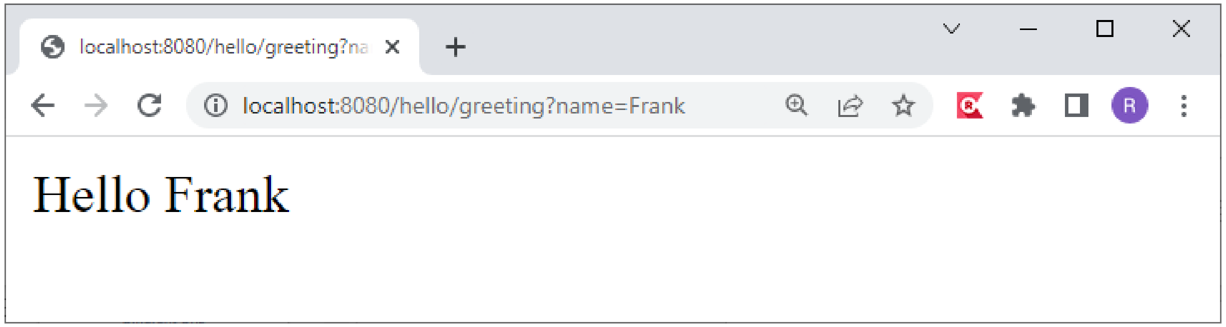
```
@RestController
@RequestMapping("/hello")
public class GreetingController {

    @RequestMapping(value="/greeting/{name}")
    public  String greeting(@PathVariable String name) {
        return "Hello "+name;
    }
}
```

Query parameters


localhost:8080/hello/greeting?na

localhost:8080/hello/greeting?name=John

Hello John

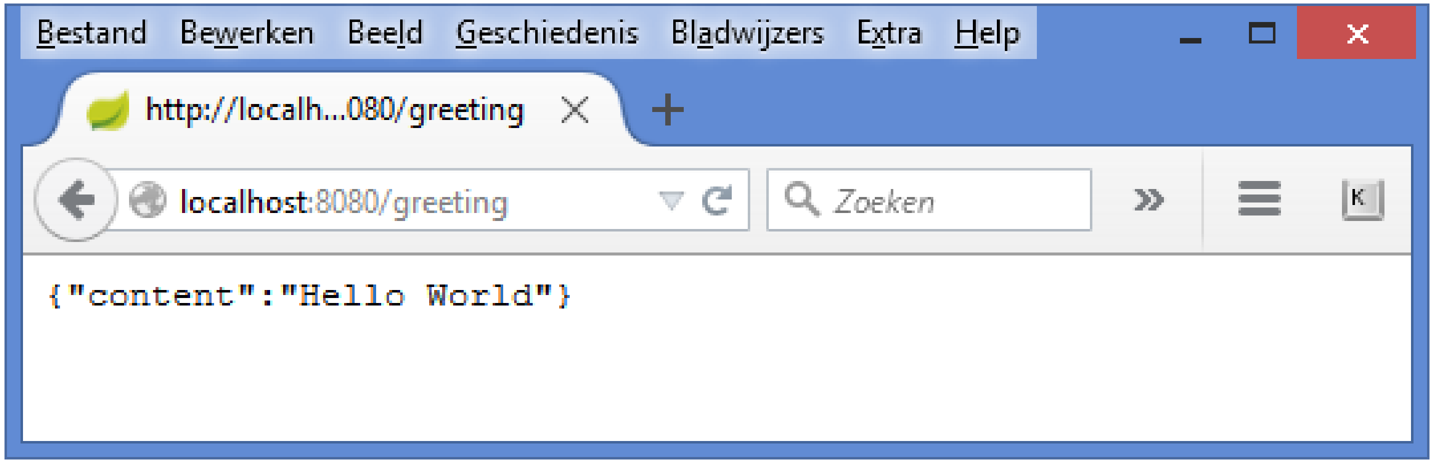

localhost:8080/hello/greeting?na

localhost:8080/hello/greeting?name=Frank

Hello Frank

```
@RestController
@RequestMapping("/hello")
public class GreetingController {

  @RequestMapping(value="/greeting")
  public  String greeting(@RequestParam String name) {
    return "Hello "+name;
  }
}
```

Returning a class


Bestand Bewerken Beeld Geschiedenis Bladwijzers Extra Help
http://localh...080/greeting
localhost:8080/greeting
Zoeken

```
{"content":"Hello World"}
```

```
@RestController
public class GreetingController {

    @RequestMapping("/greeting")
    public Greeting greeting() {
        return new Greeting("Hello World");
    }
}
```

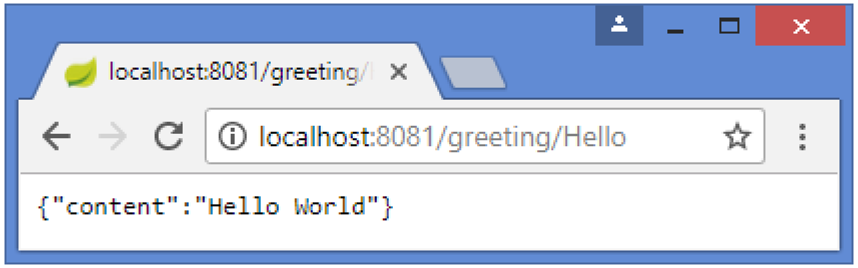
Return a Greeting class

```
public class Greeting {

  private final String content;

  public Greeting(String content) {
    this.content = content;
  }

  public String getContent() {
    return content;
  }
}
```

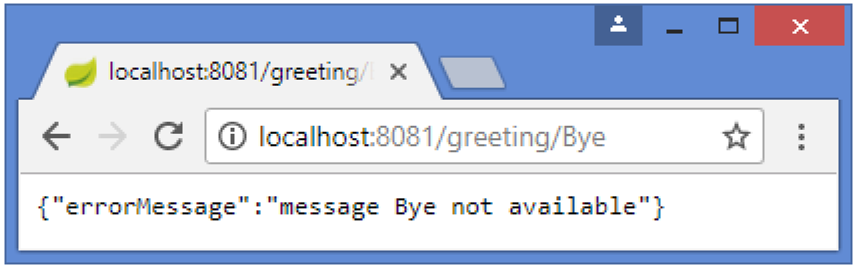
ResponseEntity

```
@RestController
public class GreetingController {

    @RequestMapping("/greeting/{message}")
    public ResponseEntity<?> getGreeting(@PathVariable("message") String message) {
      Greeting greeting = new Greeting("");
      if (message.equals("Hello")) {
        greeting.setContent("Hello World");
      }
      else{
        return new ResponseEntity(new CustomErrorType("message " + message+
          " not available"), HttpStatus.NOT_FOUND);
      }
        return new ResponseEntity<Greeting>(greeting, HttpStatus.OK);
    }
}
```

Set the content and the HttpStatus



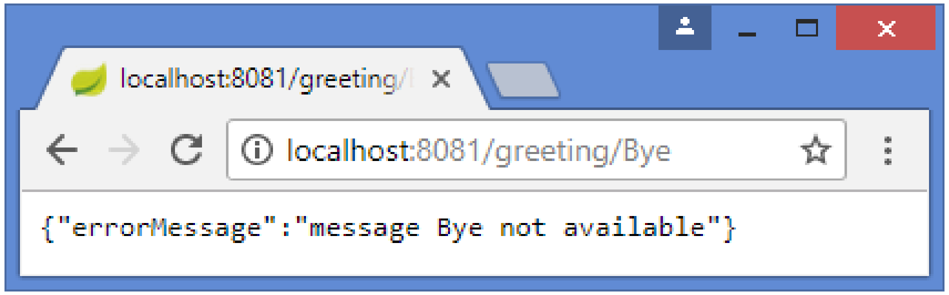
```
{"content":"Hello World"}
```



```
{"errorMessage":"message Bye not available"}
```

CustomErrorType

```java
public class CustomErrorType {
  private String errorMessage;

  public CustomErrorType(String errorMessage) {
    this.errorMessage = errorMessage;
  }

  public String getErrorMessage() {
    return errorMessage;
  }
}
```



```
{"errorMessage":"message Bye not available"}
```

Mapping annotations

```
@RequestMapping(value = "/add", method = RequestMethod.GET)
```

```
@GetMapping("/add")
```

Same

```
@RequestMapping(value = "/add", method = RequestMethod.POST)
```

```
@PostMapping("/add")
```

Same

```
@RequestMapping(value = "/del", method = RequestMethod.DELETE)
```

```
@DeleteMapping("/del")
```

Same

```
@RequestMapping(value = "/mod", method = RequestMethod.PUT)
```

```
@PutMapping("/mod")
```

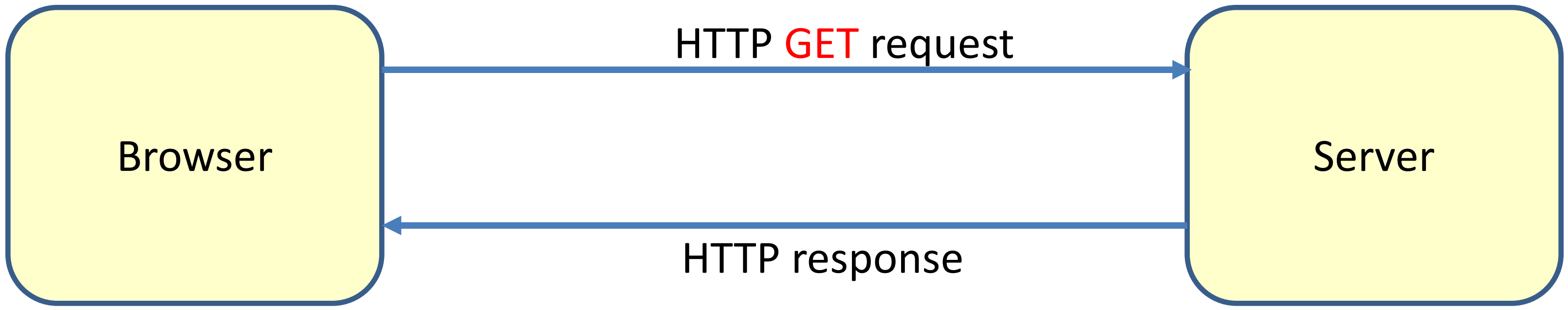
Same

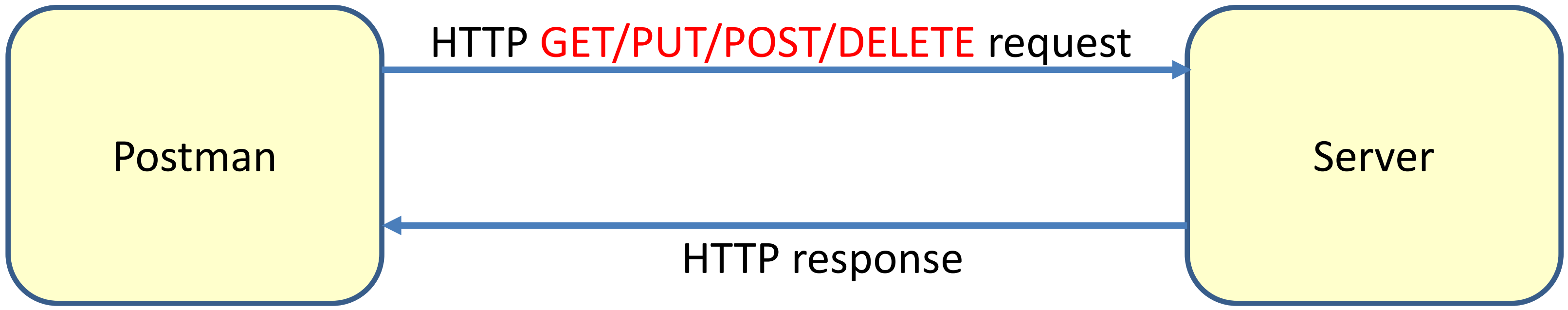
Main point

- Spring Boot makes it simple to write a RestController that can be accessed through REST webservices.

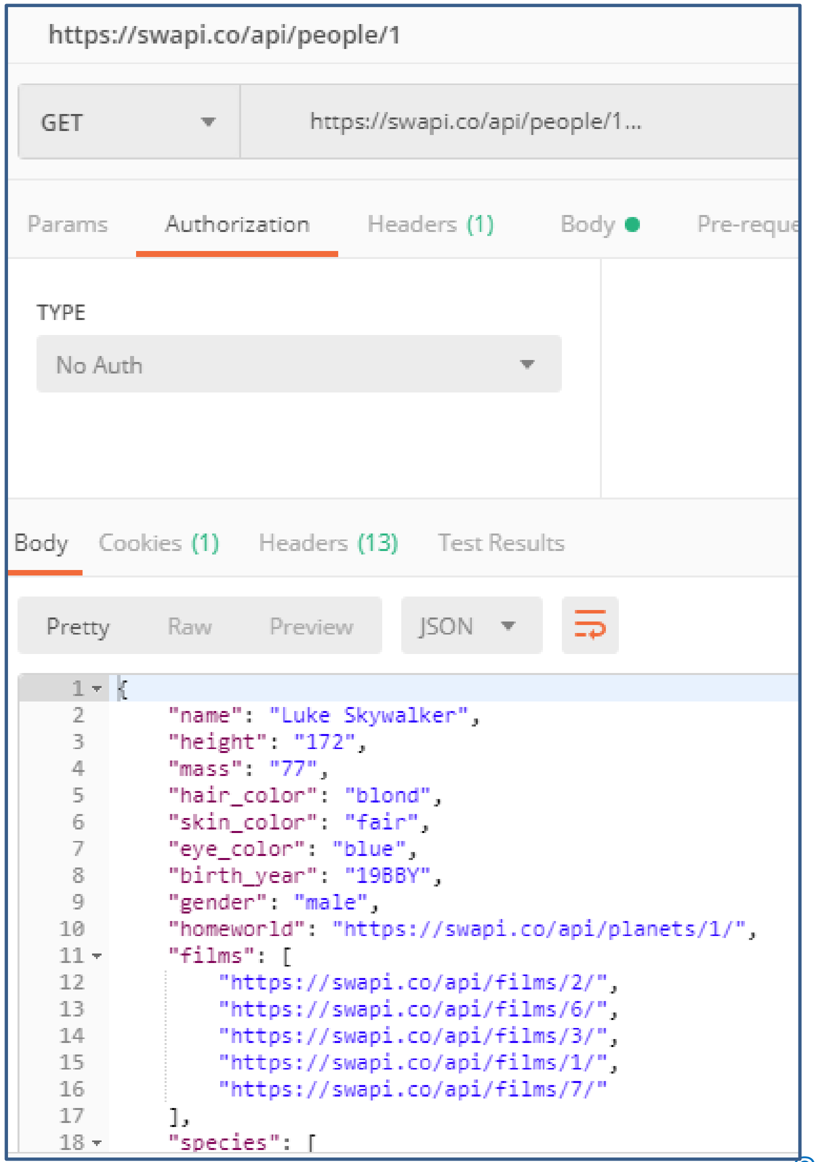
  Science of Consciousness: The human nervous system has the natural ability to transcend and experience pure consciousness.

REST client

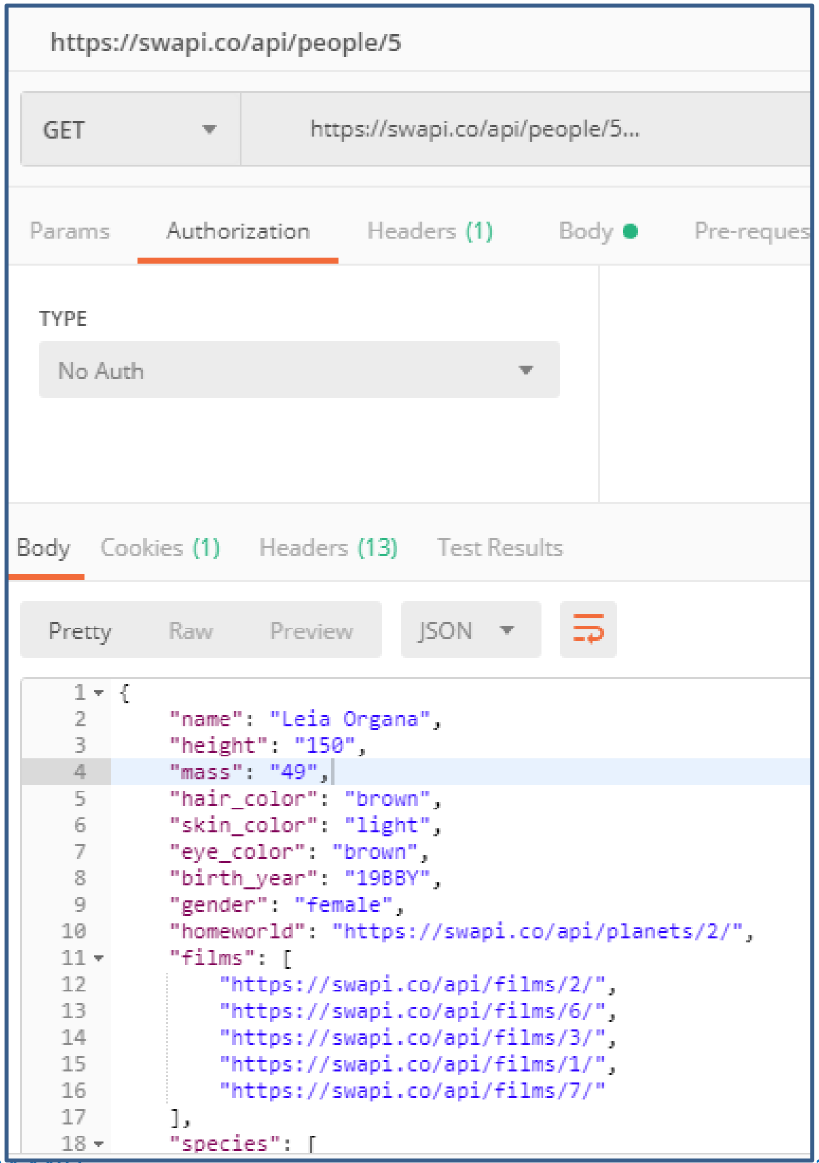

Browser
HTTP GET request
Server
HTTP response


Postman
HTTP GET/PUT/POST/DELETE request
Server
HTTP response

REST Client: Postman



```
{
    "name": "Luke Skywalker",
    "height": "172",
    "mass": "77",
    "hair_color": "blond",
    "skin_color": "fair",
    "eye_color": "blue",
    "birth_year": "19BBY",
    "gender": "male",
    "homeworld": "https://swapi.co/api/planets/1/",
    "films": [
        "https://swapi.co/api/films/2/",
        "https://swapi.co/api/films/6/",
        "https://swapi.co/api/films/3/",
        "https://swapi.co/api/films/1/",
        "https://swapi.co/api/films/7/"
    ],
    "species": [
```



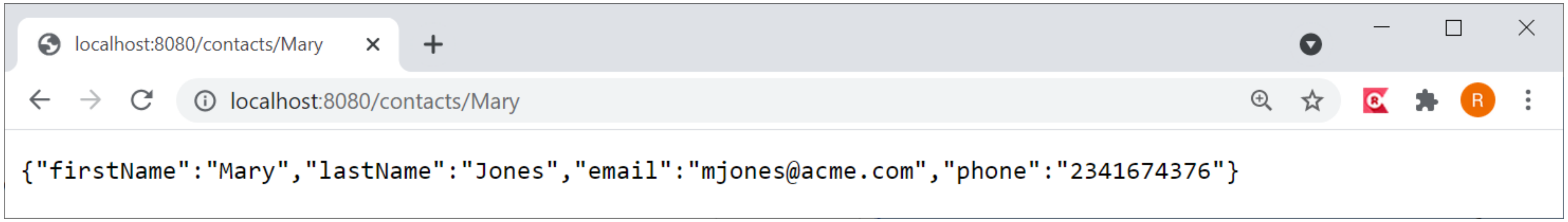
```
{
    "name": "Leia Organa",
    "height": "150",
    "mass": "49",
    "hair_color": "brown",
    "skin_color": "light",
    "eye_color": "brown",
    "birth_year": "19BBY",
    "gender": "female",
    "homeworld": "https://swapi.co/api/planets/2/",
    "films": [
        "https://swapi.co/api/films/2/",
        "https://swapi.co/api/films/6/",
        "https://swapi.co/api/films/3/",
        "https://swapi.co/api/films/1/",
        "https://swapi.co/api/films/7/"
    ],
    "species": [
```

ContactController

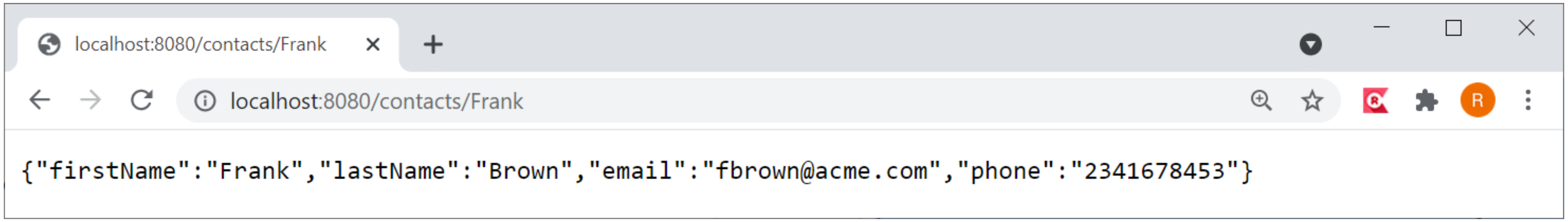
```java
public class Contact {
   private String firstName;
   private String lastName;
   private String email;
   private String phone;
...
```

```java
@RestController
public class ContactController {

   private Map<String, Contact> contacts = new HashMap<String, Contact>();

   public ContactController() {
      contacts.put("Frank", new Contact("Frank", "Brown", "fbrown@acme.com", "2341678453"));
      contacts.put("Mary", new Contact("Mary", "Jones", "mjones@acme.com", "2341674376"));
   }

   @GetMapping("/contacts/{firstName}")
   public ResponseEntity<?> getContact(@PathVariable String firstName) {
      Contact contact = contacts.get(firstName);
      if (contact == null) {
         return new ResponseEntity<CustomErrorType>(new CustomErrorType("Contact with firstname= "
               + firstName + " is not available"), HttpStatus.NOT_FOUND);
      }
      return new ResponseEntity<Contact>(contact, HttpStatus.OK);
   }
```

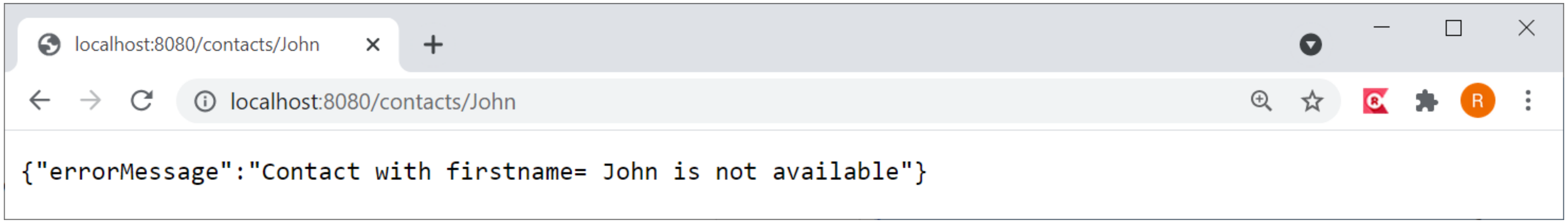
ContactController



```
{"firstName":"Mary","lastName":"Jones","email":"mjones@acme.com","phone":"2341674376"}
```



```
{"firstName":"Frank","lastName":"Brown","email":"fbrown@acme.com","phone":"2341678453"}
```



```
{"errorMessage":"Contact with firstname= John is not available"}
```

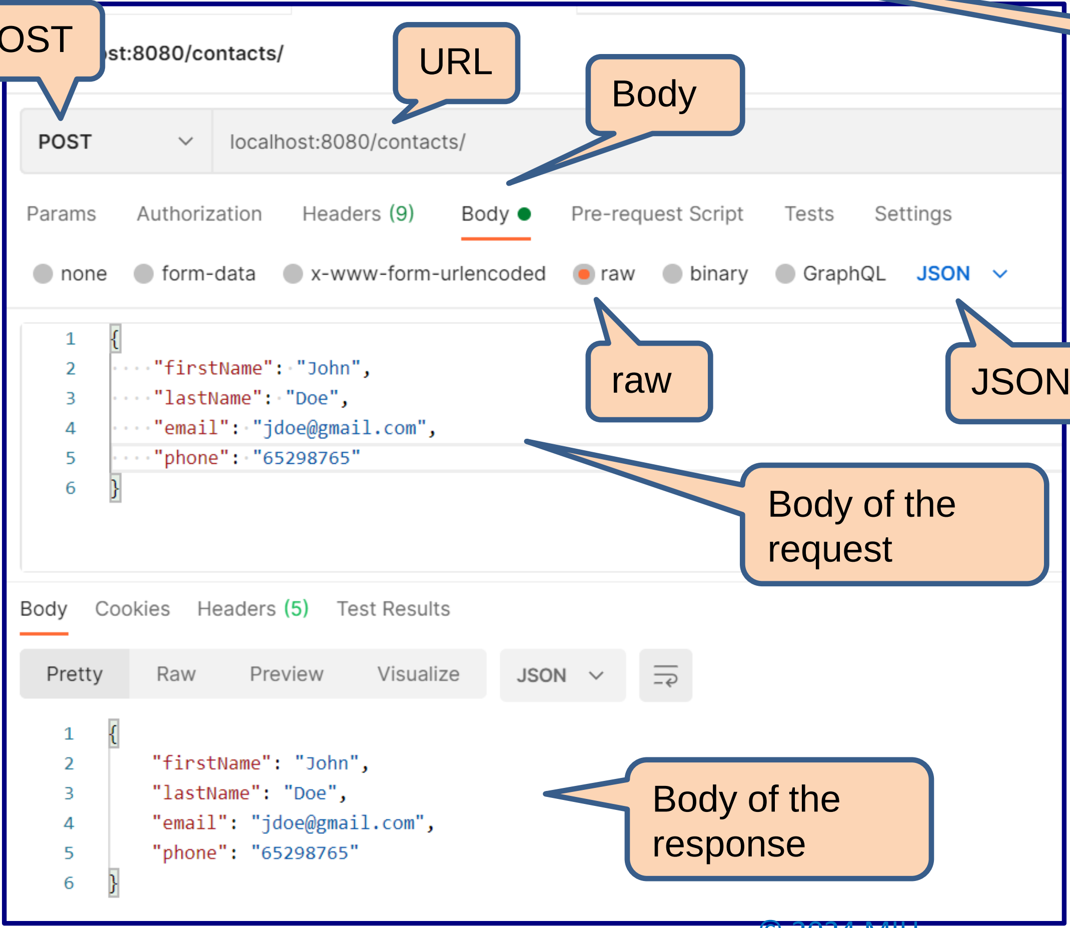
Add a contact

```
@PostMapping("/contacts")
public ResponseEntity<?> addContact(@RequestBody Contact contact) {
   contacts.put(contact.getFirstName(), contact);
   return new ResponseEntity<Contact>(contact, HttpStatus.OK);
}
```

POST request

Get the Contact class from the HTTP request message

Return the object that was send with the POST method



POST localhost:8080/contacts/

```
{
    "firstName": "John",
    "lastName": "Doe",
    "email": "jdoe@gmail.com",
    "phone": "65298765"
}
```

Response Body (Pretty, JSON):

```
{
    "firstName": "John",
    "lastName": "Doe",
    "email": "jdoe@gmail.com",
    "phone": "65298765"
}
```

Delete a contact

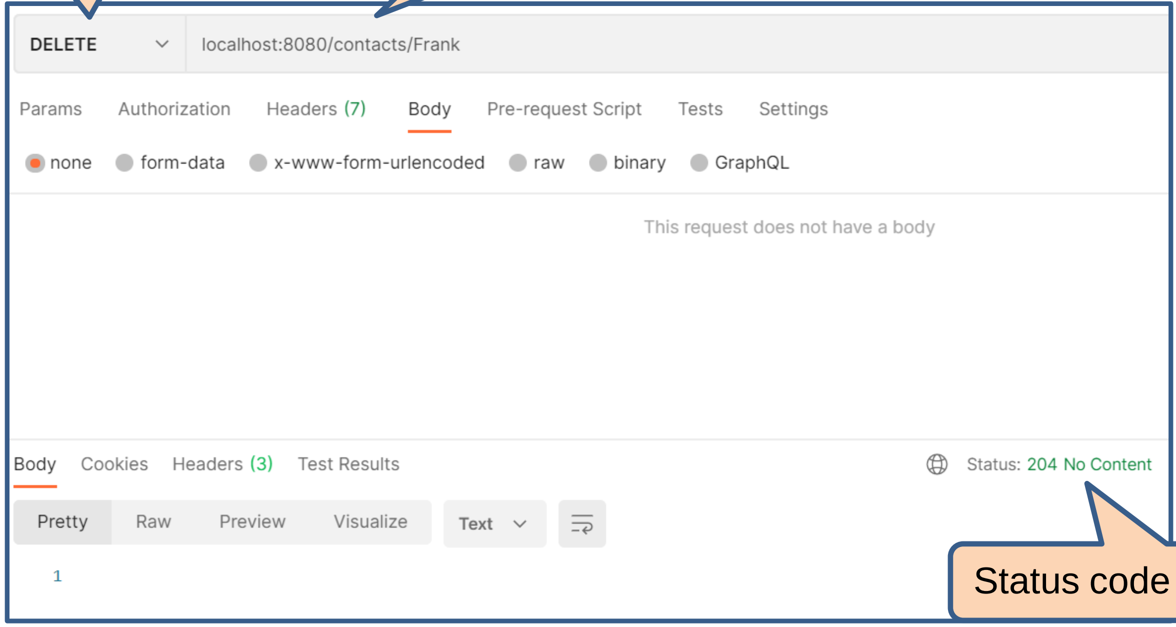
```
@DeleteMapping("/contacts/{firstName}")
public ResponseEntity<?> deleteContact(@PathVariable String firstName) {
   Contact contact = contacts.get(firstName);
   if (contact == null) {
      return new ResponseEntity<CustomErrorType>(new CustomErrorType("Contact with firstname= " +
              firstName + " is not available"),HttpStatus.NOT_FOUND);
   }
   contacts.remove(firstName);
   return new ResponseEntity<>(HttpStatus.NO_CONTENT);
}
```

DELETE request

Return nothing

DELETE

URL



DELETE localhost:8080/contacts/Frank

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

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

This request does not have a body

Body Cookies Headers (3) Test Results Status: 204 No Content

Pretty Raw Preview Visualize Text

1

Status code

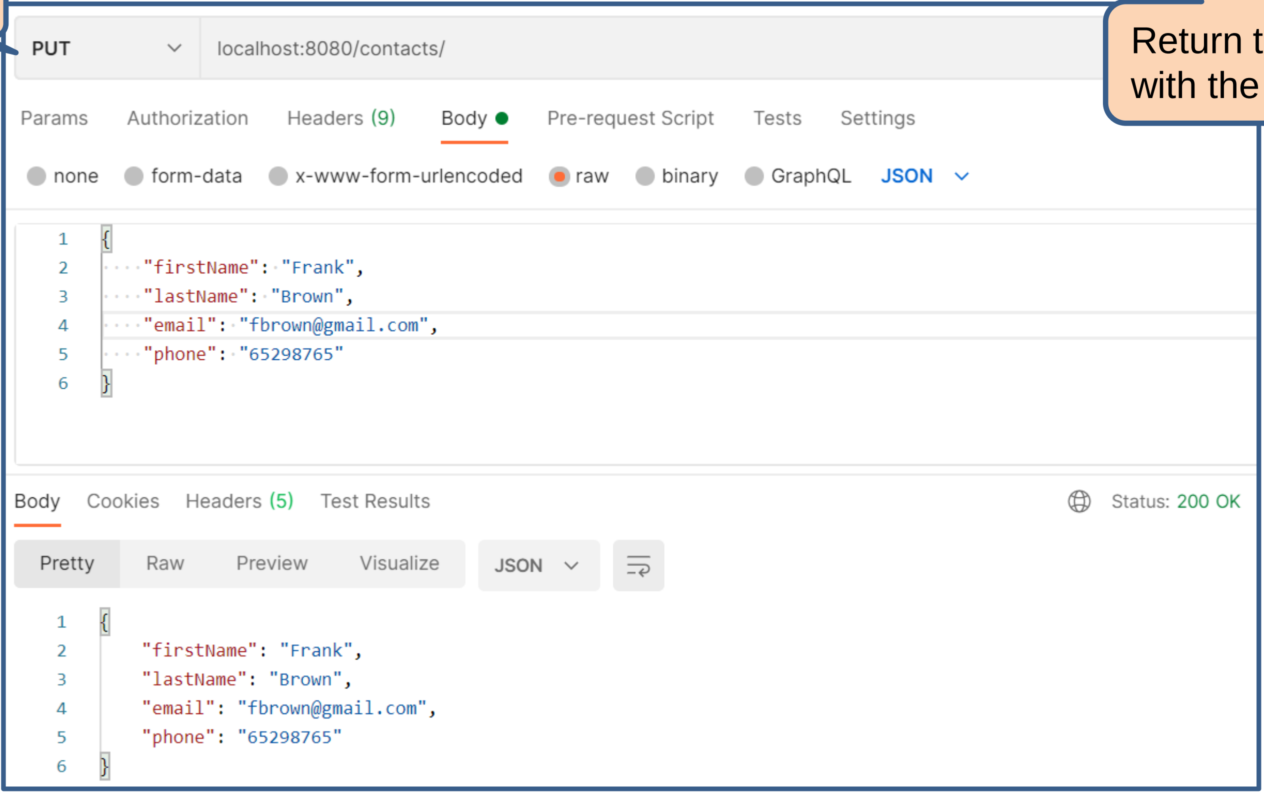
Update a contact

PUT request

```
@PutMapping("/contacts/{firstName}")
public ResponseEntity<?> updateContact(@PathVariable String firstName, @RequestBody Contact contact) {
   contacts.put(firstName, contact);
   return new ResponseEntity<Contact>(contact, HttpStatus.OK);
}
```

PUT

Return the object that was send with the PUT method



```
{
    "firstName": "Frank",
    "lastName": "Brown",
    "email": "fbrown@gmail.com",
    "phone": "65298765"
}
```

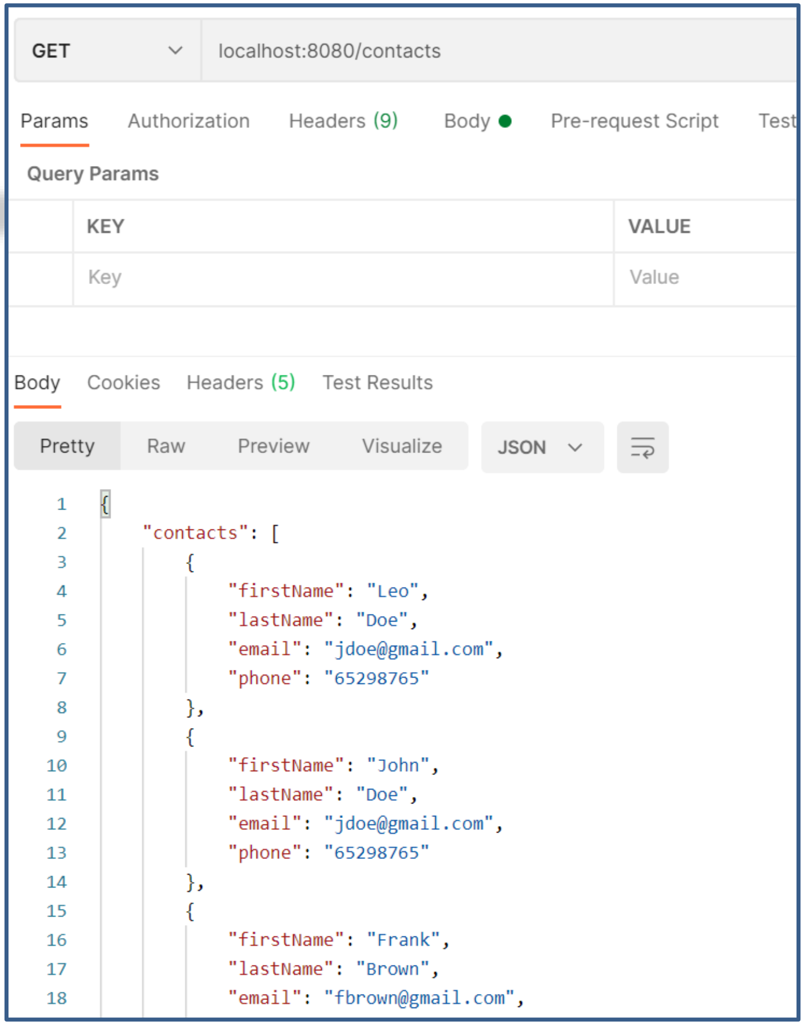
```
{
    "firstName": "Frank",
    "lastName": "Brown",
    "email": "fbrown@gmail.com",
    "phone": "65298765"
}
```

Get all contacts

```
@GetMapping("/contacts")
public ResponseEntity<?> getAllContacts() {
   Contacts allcontacs = new Contacts(contacts.values());
   return new ResponseEntity<Contacts>(allcontacs, HttpStatus.OK);
}
```

```
public class Contacts {
   private Collection<Contact> contacts;


…
```

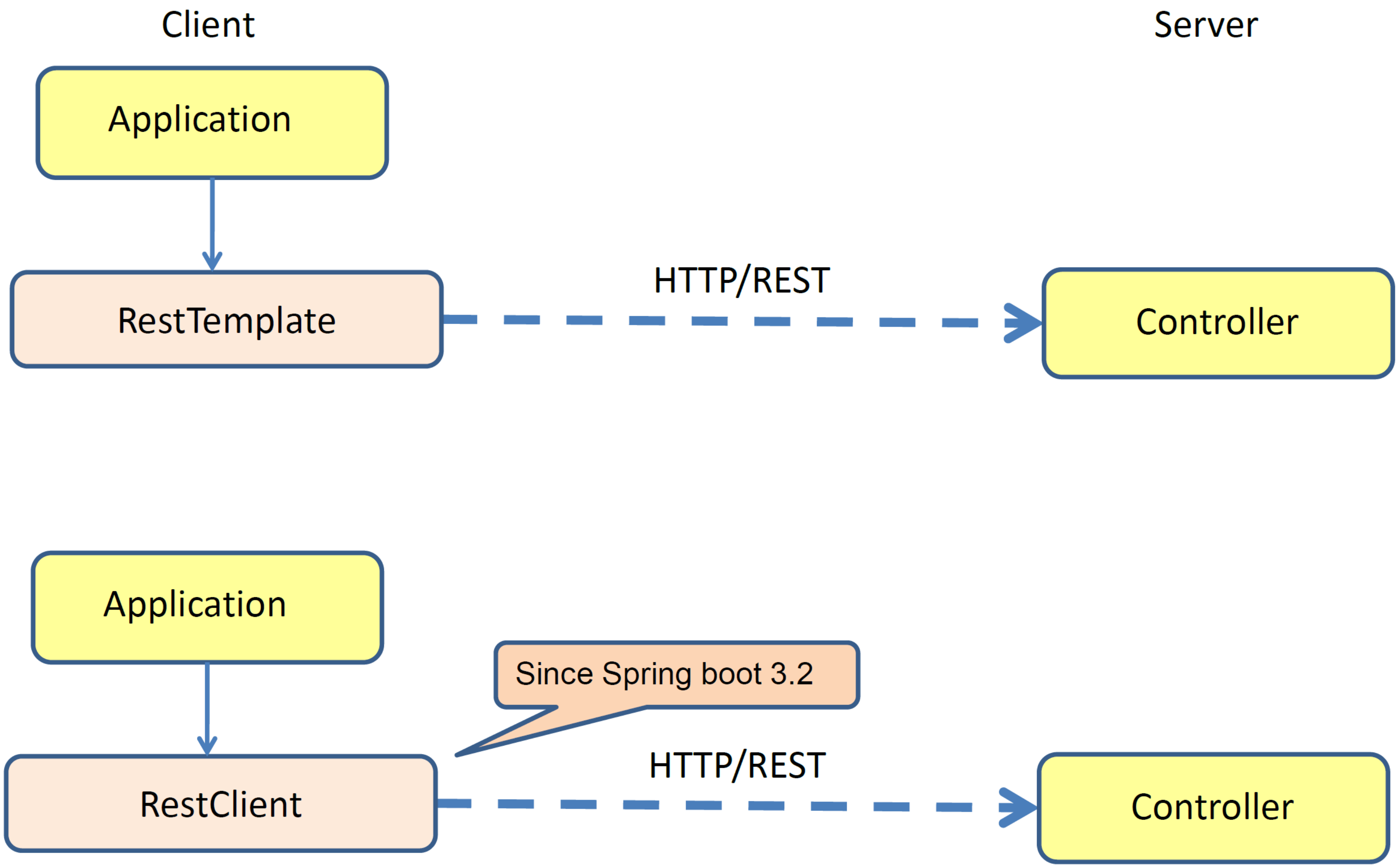
Create a new class



localhost:8080/contacts

```
{
    "contacts": [
        {
            "firstName": "Leo",
            "lastName": "Doe",
            "email": "jdoe@gmail.com",
            "phone": "65298765"
        },
        {
            "firstName": "John",
            "lastName": "Doe",
            "email": "jdoe@gmail.com",
            "phone": "65298765"
        },
        {
            "firstName": "Frank",
            "lastName": "Brown",
            "email": "fbrown@gmail.com",
```

Creating a REST client



REST server

```
@RestController
public class Controller {
   @RequestMapping("/greeting")
   public Greeting greeting() {
      return new Greeting("Hello World");
   }

}
```

```
public class Greeting {

   private String content;

   public Greeting() {
   }

   public Greeting(String content) {
      this.content = content;
   }

   public String getContent() {
      return content;
   }

   @Override
   public String toString() {
      return "Greeting{" +
            "content='" + content + '\''
+
            '}';
   }
}
```

Client with REST template

```
@SpringBootApplication
public class RestClientApplication implements CommandLineRunner {

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

   @Override
   public void run(String... args) throws Exception {
     RestTemplate restTemplate = new RestTemplate();

     Greeting message = restTemplate.getForObject("http://localhost:8080/greeting", Greeting.class);
     System.out.println(message);
   }
}
```

RestTemplate

Client with RestClient

```
@SpringBootApplication
public class RestClientApplication implements CommandLineRunner {

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

   @Override
   public void run(String... args) throws Exception {
     RestClient restClient = RestClient.builder()
         .baseUrl("http://localhost:8080")
         .build();

     Greeting message = restClient.get()
         .uri("/greeting")
         .retrieve()
         .body(Greeting.class);

     System.out.println(message);
   }
}
```

RestClient

Contact client(1/2) with RestTemplate

```
@SpringBootApplication
public class RestClientApplication implements CommandLineRunner {

  private RestTemplate restTemplate = new RestTemplate;

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

  @Override
  public void run(String... args) throws Exception {
    String serverUrl = "http://localhost:8080/contacts";

    // add Frank
    restTemplate.postForLocation(serverUrl, new Contact("Frank","Browns", "fbrowns@acme.com",
        "0639332163"));
    // add John
    restTemplate.postForLocation(serverUrl, new Contact("John","Doe", "jdoe@acme.com",
        "6739127563"));
    // get frank
    Contact contact= restTemplate.getForObject(serverUrl+"/{firstName}", Contact.class, "Frank");
    System.out.println("----------- get John----------------------");
    System.out.println(contact.getFirstName()+" "+contact.getLastName());
```

Contact client(2/2) with RestTemplate

```
      // get all
     Contacts contacts= restTemplate.getForObject(serverUrl, Contacts.class);
     System.out.println("----------- get all contacts-----------------------");
     System.out.println(contacts);

     // delete John
     restTemplate.delete(serverUrl+"/{firstName}", "John");

     // update frank
     contact.setEmail("franky@gmail.com");
     restTemplate.put(serverUrl+"/{firstName}", contact, contact.getFirstName());

     // get all
     contacts= restTemplate.getForObject(serverUrl, Contacts.class);
     System.out.println("----------- get all contacts-----------------------");
     System.out.println(contacts);
   }


 }
```

Contact client (1/2) with RestClient

```java
@SpringBootApplication
public class RestClientApplication implements CommandLineRunner {

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

   @Override
   public void run(String... args) throws Exception {
     RestClient restClient = RestClient.builder()
         .baseUrl("http://localhost:8080")
         .build();

     // get frank
     Contact frank = restClient.get()
         .uri("/contacts/{firstname}", "Frank")
         .retrieve()
         .body(Contact.class);
     System.out.println(frank);

     // add John
     Contact johnResponse = restClient.post()
         .uri("/contacts")
         .contentType(MediaType.APPLICATION_JSON)
         .body(new Contact("John","Doe", "jdoe@acme.com", "6739127563"))
         .retrieve()
         .body(Contact.class);
```

Contact client (2/2) with RestClient

```
// get john
Contact john = restClient.get()
     .uri("/contacts/{firstname}", "John")
     .retrieve()
     .body(Contact.class);
System.out.println(john);

// delete mary
restClient.delete()
     .uri("/contacts/{firstName}", "Mary")
     .retrieve()
     .toBodilessEntity();

// update John
john.setEmail("johndoe@acme.com");
johnResponse = restClient.post()
     .uri("/contacts")
     .contentType(MediaType.APPLICATION_JSON)
     .body(john)
     .retrieve()
     .body(Contact.class);

// get john
john = restClient.get()
     .uri("/contacts/{firstname}", "John")
     .retrieve()
     .body(Contact.class);
System.out.println(john);
  }
}
```

Get all contacts

```
@RequestMapping(value="/contacts", method=RequestMethod.GET)
public ResponseEntity<?> getAllContacts() {
   return new ResponseEntity<Collection<Contact>>(contacts.values(), HttpStatus.OK);
}
```

```
List<Contact> contacts = restClient.get()
        .uri("/contacts")
        .retrieve()
        .body(new ParameterizedTypeReference<List<Contact>>() {});
System.out.println(contacts);
```

Main point

- A RestClient has 4 methods. One method for sending a GET request, one method for sending a POST request , one method for sending a PUT request and one method for sending a DELETE request.

  Science of Consciousness: There are many ways to transcend, but TM is an effective and effortless technique.

EXCEPTION HANDLING

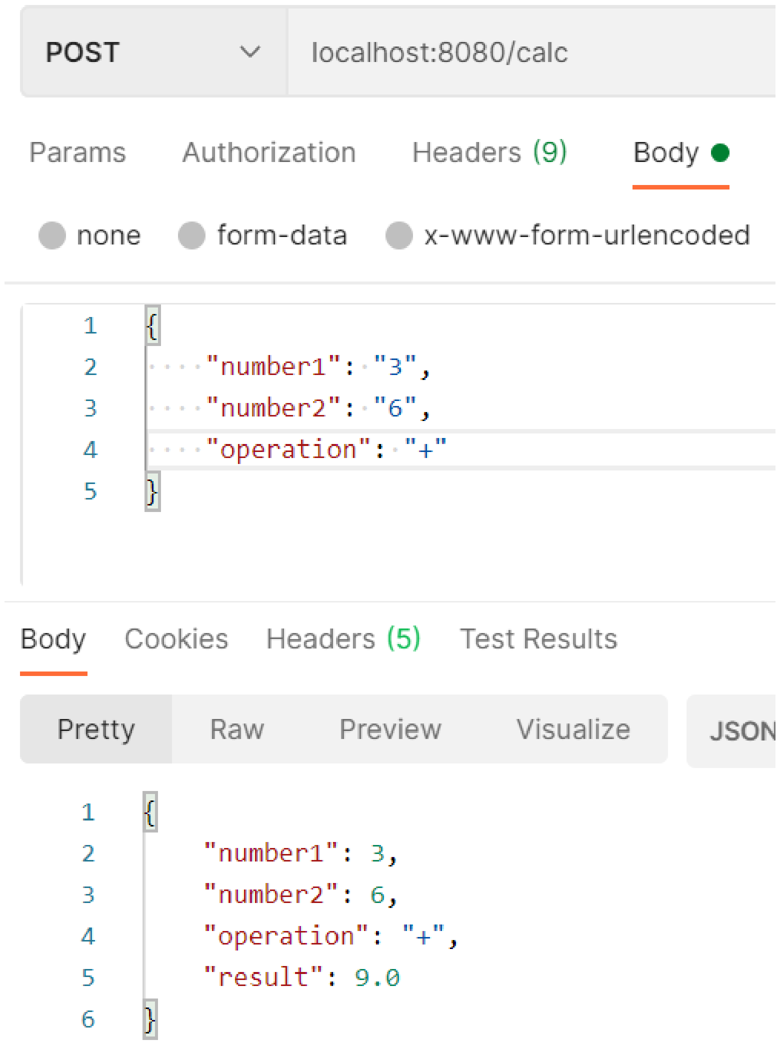
Calculator

```
@RestController
public class CalcController {

  @PostMapping("/calc")
  public ResponseEntity<?> calculate(@RequestBody Calculation calculation) {
    double result=0.0;

    switch(calculation.getOperation()){
      case "+" : {result = calculation.getNumber1() + calculation.getNumber2(); break;}
      case "-" : {result = calculation.getNumber1() - calculation.getNumber2(); break;}
      case "*" : {result = calculation.getNumber1() * calculation.getNumber2(); break;}
      case "/" : {result = calculation.getNumber1() / calculation.getNumber2(); break;}
    }
    CalculationResult calculationResult = new CalculationResult(calculation.getNumber1(),
            calculation.getNumber2(),calculation.getOperation(), result);
    return new ResponseEntity<CalculationResult>(calculationResult, HttpStatus.OK);
  }
}
```

```
public class Calculation {

  private int number1;
  private int number2;
  private String operation;
...
```

```
public class CalculationResult {
  private int number1;
  private int number2;
  private String operation;
  private double result;
...
```

Calculator

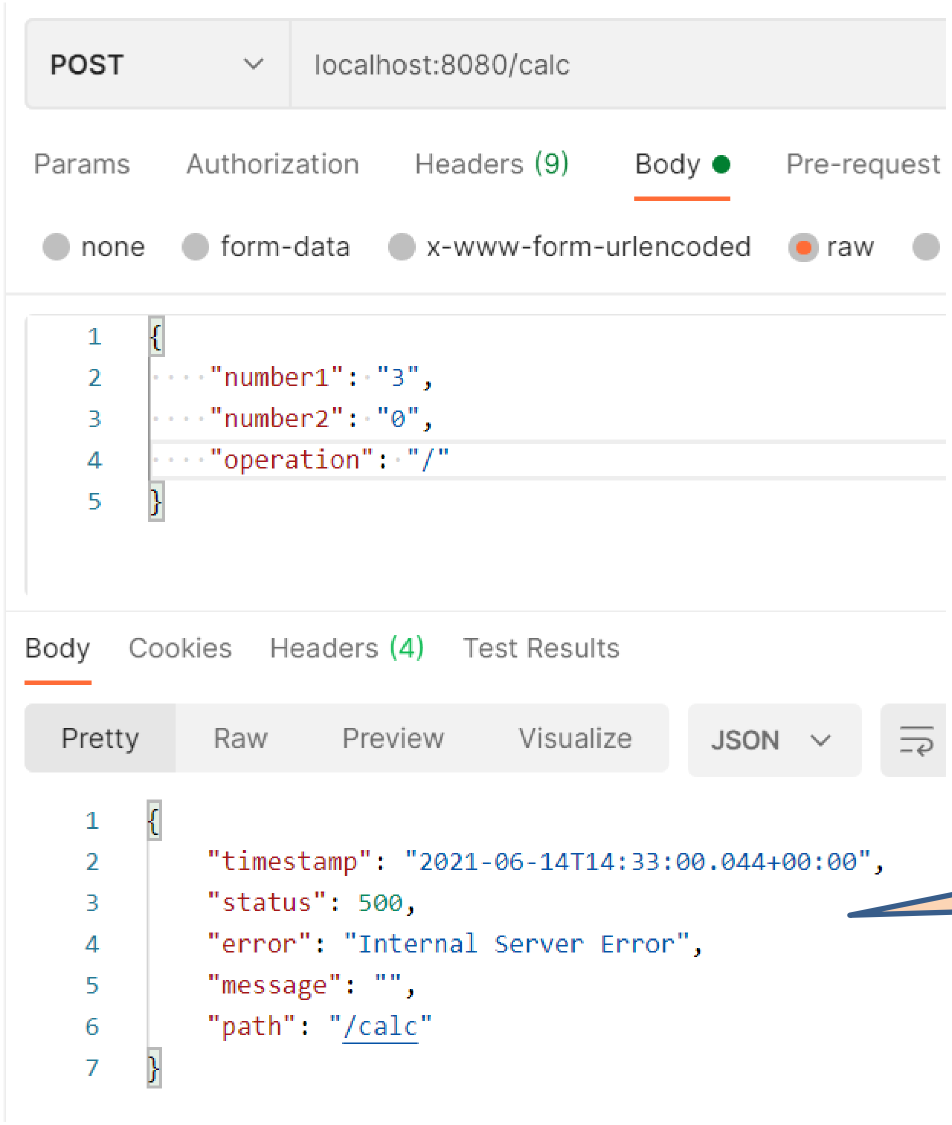


```
localhost:8080/calc
```

```
{
    "number1": "3",
    "number2": "6",
    "operation": "+"
}
```

```
{
    "number1": 3,
    "number2": 6,
    "operation": "+",
    "result": 9.0
}
```

Divide by zero



```
{
    "number1": "3",
    "number2": "0",
    "operation": "/"
}
```

```
{
    "timestamp": "2021-06-14T14:33:00.044+00:00",
    "status": 500,
    "error": "Internal Server Error",
    "message": "",
    "path": "/calc"
}
```

Default response in case of an exception

3 ways to handle the exception yourself

1. Handle the exception in the controller method
2. Exception handler per controller
3. Global exception handler

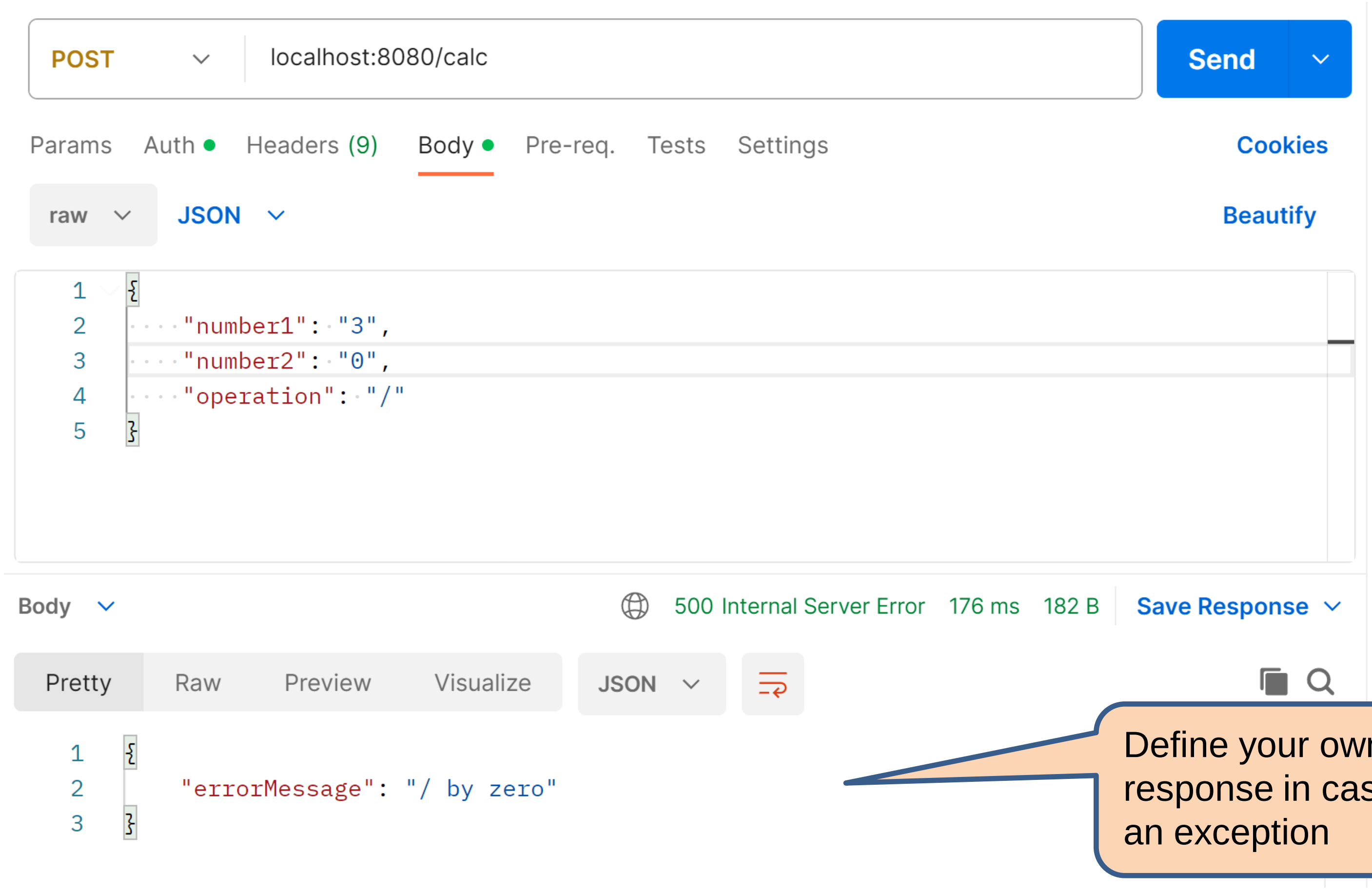
Handle the error in the controller method

```
@RestController
public class CalcController {

  @PostMapping("/calc")
  public ResponseEntity<?> calculate(@RequestBody Calculation calculation) {
    System.out.println("operation = "+calculation.getOperation());
    double result=0.0;
    try{
      switch(calculation.getOperation()){
        case "+" : {result = calculation.getNumber1() + calculation.getNumber2(); break;}
        case "-" : {result = calculation.getNumber1() - calculation.getNumber2(); break;}
        case "*" : {result = calculation.getNumber1() * calculation.getNumber2(); break;}
        case "/" : {result = calculation.getNumber1() / calculation.getNumber2(); break;}
      }
      CalculationResult calculationResult = new CalculationResult(calculation.getNumber1(),
            calculation.getNumber2(),calculation.getOperation(), result);
      return new ResponseEntity<CalculationResult>(calculationResult, HttpStatus.OK);
    }
    catch (Exception exception){
      System.out.println("exception = "+exception.getMessage());
      return new ResponseEntity<CalculationError>(new CalculationError(exception.getMessage()),
           HttpStatus.INTERNAL_SERVER_ERROR);
    }
  }
}
```

Handle the exception

You have to do this for all controller methods

Handle the error in the controller method



```
{
    "number1": "3",
    "number2": "0",
    "operation": "/"
}
```

```
{
    "errorMessage": "/ by zero"
}
```

Define your own response in case of an exception

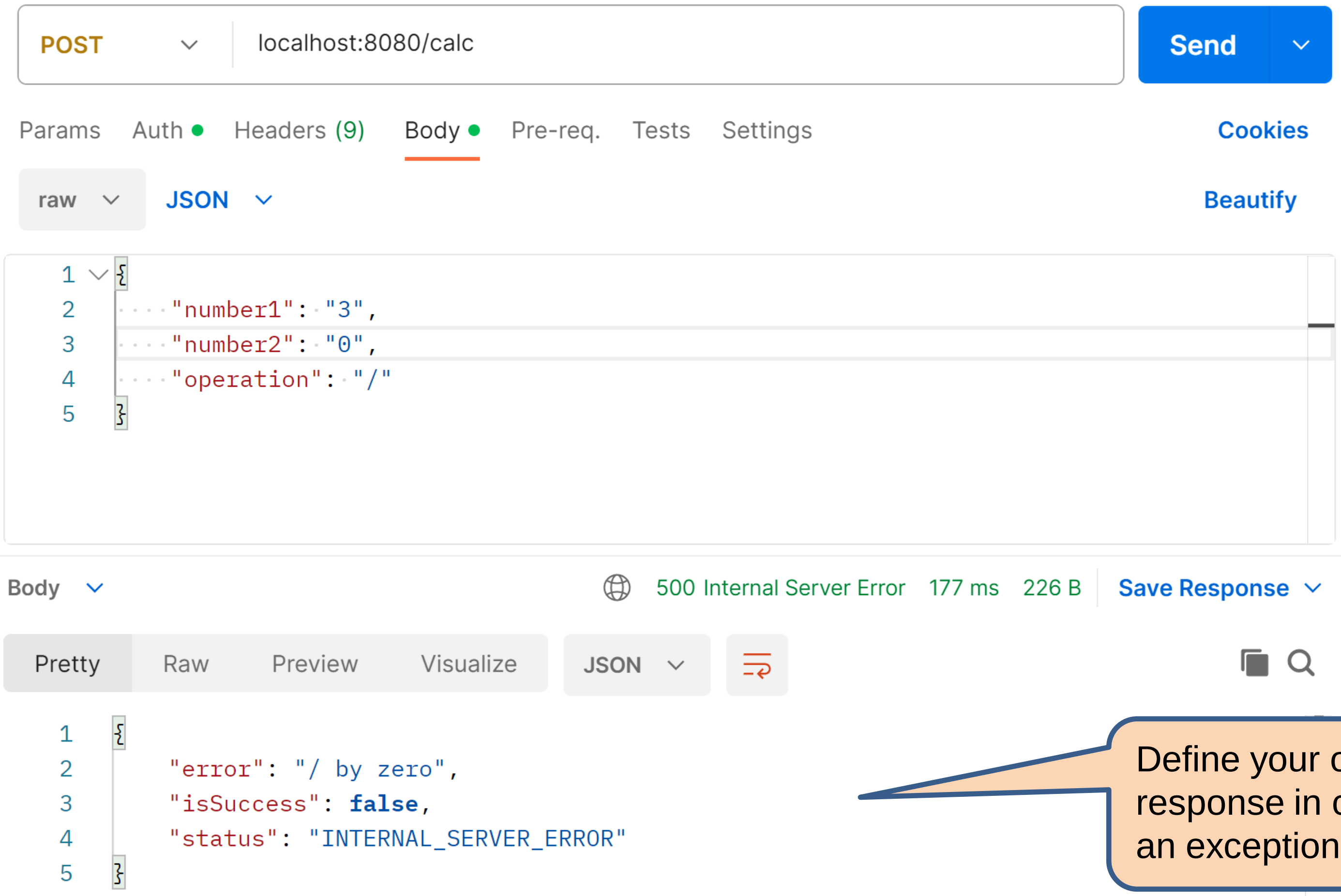
Exception handler per controller

```
@RestController
public class CalcController2 {

  @PostMapping("/calc")
  public ResponseEntity<?> calculate(@RequestBody Calculation calculation) {
    double result=0.0;
    switch(calculation.getOperation()){
      case "+" : {result = calculation.getNumber1() + calculation.getNumber2(); break;}
      case "-" : {result = calculation.getNumber1() - calculation.getNumber2(); break;}
      case "*" : {result = calculation.getNumber1() * calculation.getNumber2(); break;}
      case "/" : {result = calculation.getNumber1() / calculation.getNumber2(); break;}
    }
    CalculationResult calculationResult = new CalculationResult(calculation.getNumber1(),
            calculation.getNumber2(),calculation.getOperation(), result);
    return new ResponseEntity<CalculationResult>(calculationResult, HttpStatus.OK);
  }

  @ExceptionHandler(Exception.class)
  public ResponseEntity<Object> handleExceptions(Exception exception) {
    Map<String, Object> map = new HashMap<>();
    map.put("isSuccess", false);
    map.put("error", exception.getMessage());
    map.put("status", HttpStatus.INTERNAL_SERVER_ERROR);
    return new ResponseEntity<Object>(map,HttpStatus.INTERNAL_SERVER_ERROR);
  }
}
```

One method to handle all exceptions within this controller

You have to do this for every controller

Exception handler per controller



localhost:8080/calc

```
{
    "number1": "3",
    "number2": "0",
    "operation": "/"
}
```

500 Internal Server Error 177 ms 226 B

```
{
    "error": "/ by zero",
    "isSuccess": false,
    "status": "INTERNAL_SERVER_ERROR"
}
```

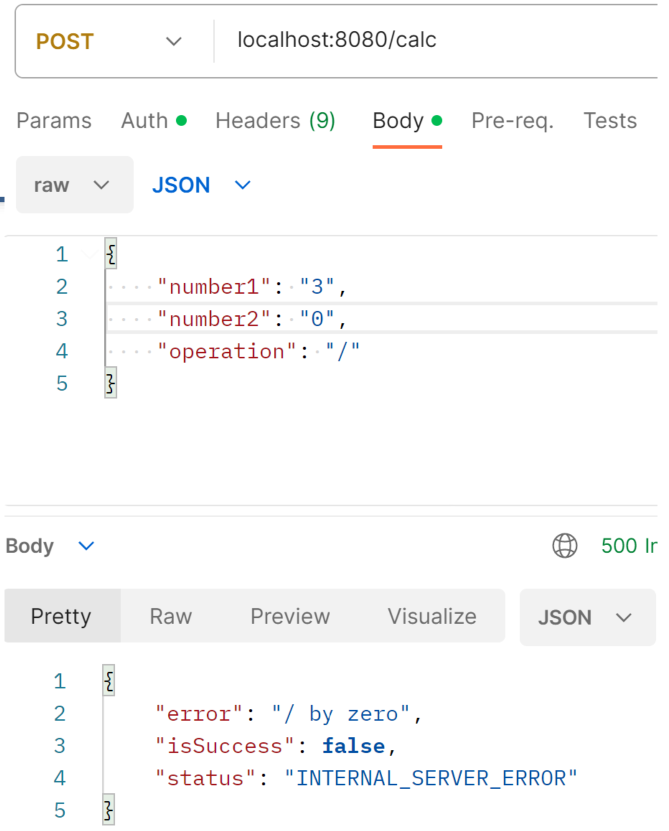
Define your own response in case of an exception

Global exception handler

```java
@ControllerAdvice
public class CustomExceptionHandler extends ResponseEntityExceptionHandler {

    @ExceptionHandler(value = { Exception.class})
    protected ResponseEntity<Object> handleConflict(RuntimeException exception, WebRequest request) {
        Map<String, Object> map = new HashMap<>();
        map.put("isSuccess", false);
        map.put("error", exception.getMessage());
        map.put("status", HttpStatus.INTERNAL_SERVER_ERROR);
        return new ResponseEntity<Object>(map,HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Array of exceptions

This method handles the exceptions of all controller methods



```json
{
    "number1": "3",
    "number2": "0",
    "operation": "/"
}
```

```json
{
    "error": "/ by zero",
    "isSuccess": false,
    "status": "INTERNAL_SERVER_ERROR"
}
```

REST API DESIGN

Use nouns, not verbs

- Do not create a URL for every action you need todo:

/getCustomers	/saveCustomers
/getCustomersByName	/getCustomersByPhone
/getCustomersByContact	/getCustomersUsingPaging
/getNewCustomers	/getCurrentCustomers
/createNewCustomer	/deleteCustomer

NOT OK

Use verbs

OK

Resource	POST	GET	PUT	DELETE
/customers	Create a new customer	Retrieve all customers	Bulk update of customers	Remove all customers
/customers/1	Error	Retrieve the details for customer 1	Update the details of customer 1 if it exists	Remove customer 1
/customers/1/orders	Create a new order for customer 1	Retrieve all orders for customer 1	Bulk update of orders for customer 1	Remove all orders for customer 1

What should the method return?

Resource	GET (read)	POST (insert)	PUT (update)	DELETE (delete)
/customers	List	New Item	Status Code Only	Status Code Only*
/customers/123	Item	Status Code Only*	Updated Item	Status Code Only

* Error code

Use correct status codes

Code	Description	Code	Description
200	OK	400	Bad Request
201	Created	401	Not Authorized
202	Accepted	403	Forbidden
302	Found	404	Not Found
304	Not Modified	405	Method Not Allowed
307	Temp Redirect	409	Conflict
308	Perm Redirect	500	Internal Error

Filtering, pagination, sorting

- **Filtering**: Return only results that match a filter by using field age as a parameter.
  - GET /users?age=30
- **Pagination**: Don’t overload clients and servers by providing everything.
  - GET /users?page=3&results_per_page=20
- **Sorting**: Provide a way to sort or some use cases will still require paging through all results to find what’s needed.
  - GET /users?sort_by=first_name&order=asc

More complex functionality

- Use query string

```
http://…/api/Customers?state=GA
http://…/api/Customers?state=GA&salesperson=144
http://…/api/Customers?hasOpenOrders=true
```

Connecting the parts of knowledge with the wholeness of knowledge

1. Rest webservices is a simple HTTP based technique that allow other applications to call your application over HTTP.
2. The RestClient in Spring Boot allows you to send REST calls and hides all underlying details.

---

3. **Transcendental consciousness** is the field of all knowledge.
4. **Wholeness moving within itself:** In unity consciousness, one experiences that the whole creation is just an expression of one's own Self.