

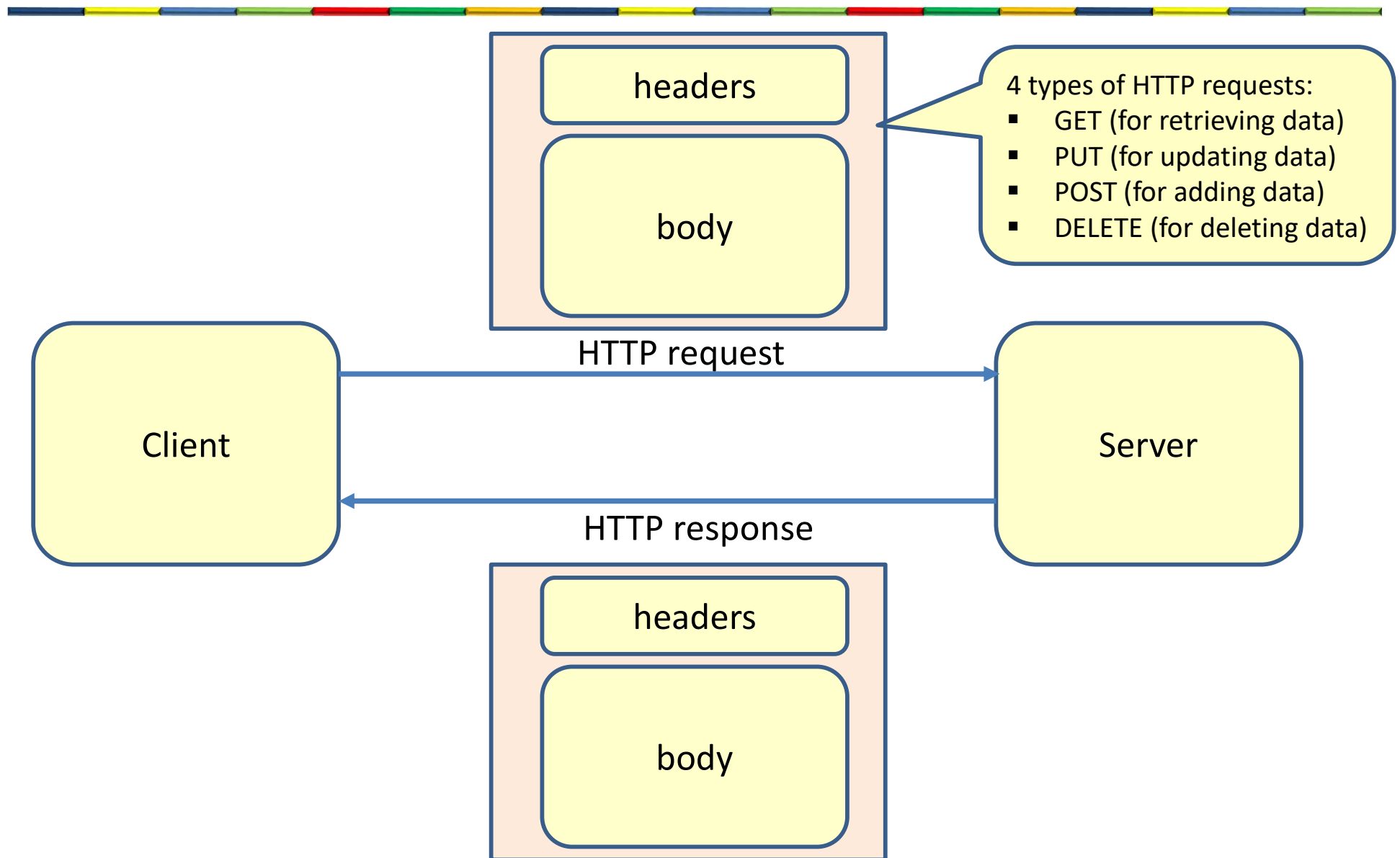
CS544

LESSON 9

REST WEBSERVICES

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
April 3 Lesson 1 Introduction Spring framework Dependency injection	April 4 Lesson 2 Spring Boot AOP	April 5 Lesson 3 JDBC JPA	April 6 Lesson 4 JPA mapping 1	April 7 Lesson 5 JPA mapping 2	April 8 Lesson 6 JPA queries	April 9
April 10 Lesson 7 Transactions	April 11 Lesson 8 MongoDB	April 12 Midterm Review	April 13 Midterm exam	April 14 Lesson 9 REST webservises	April 15 Lesson 10 SOAP webservises	April 16
April 17 Lesson 11 Messaging	April 18 Lesson 12 Scheduling Events Configuration	April 19 Lesson 13 Monitoring	April 20 Lesson 14 Testing your application	April 21 Final review/Project	April 22 Project	April 23
April 24 Final exam	April 25 Project	April 26 Project	April 27 Class celebration			

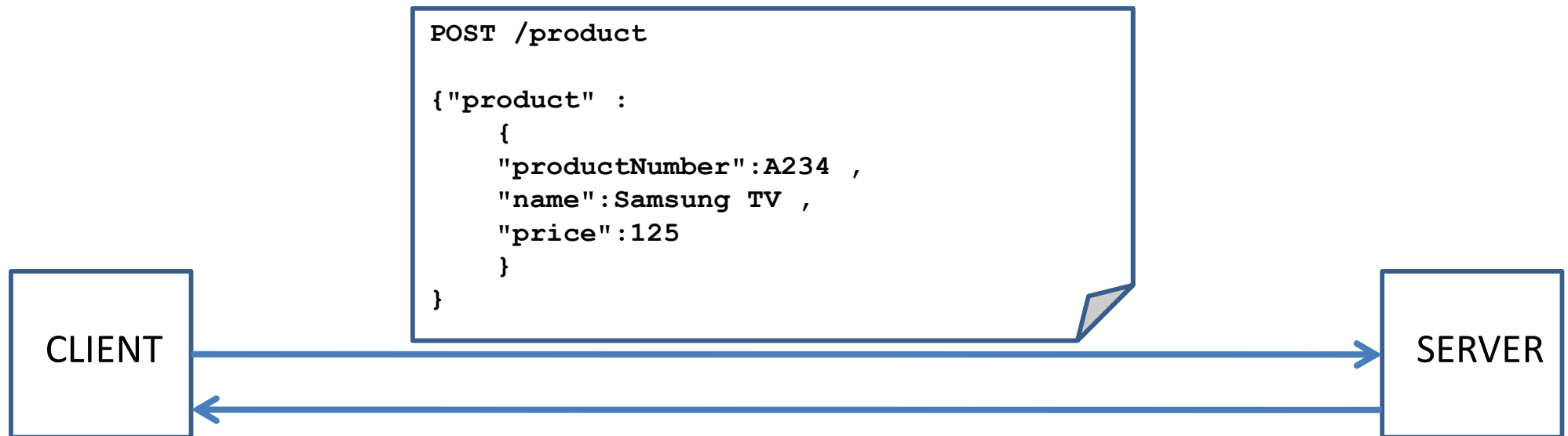
REST webservice



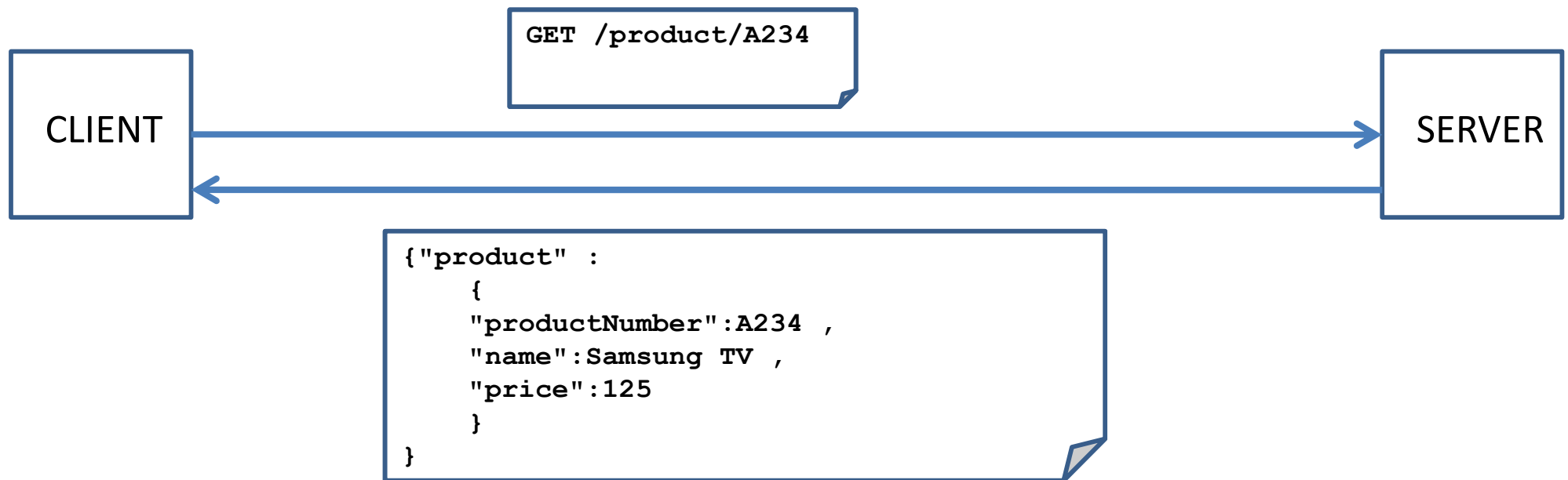
Http methods

Method	Idempotent
GET	YES
POST	NO
PUT	YES
DELETE	YES

POST method using JSON



GET method using JSON



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

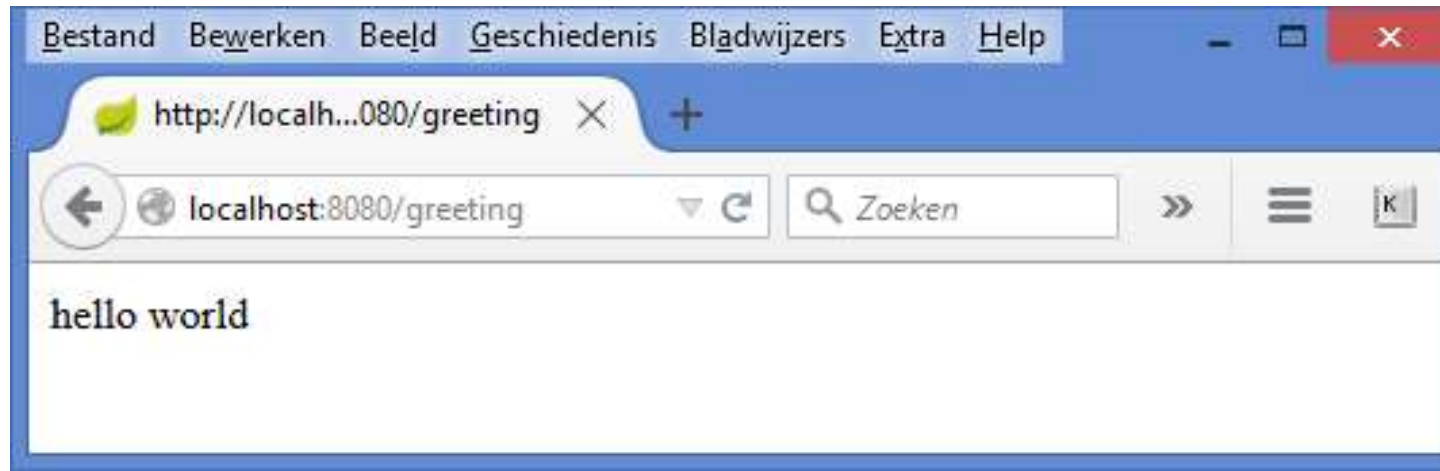
One annotations is same as these 3 together

@Configuration
@EnableConfiguration
@ComponentScan

```
@SpringBootApplication
public class GreetingRestApplication {

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

Simple Rest Example: calling the service



```
@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

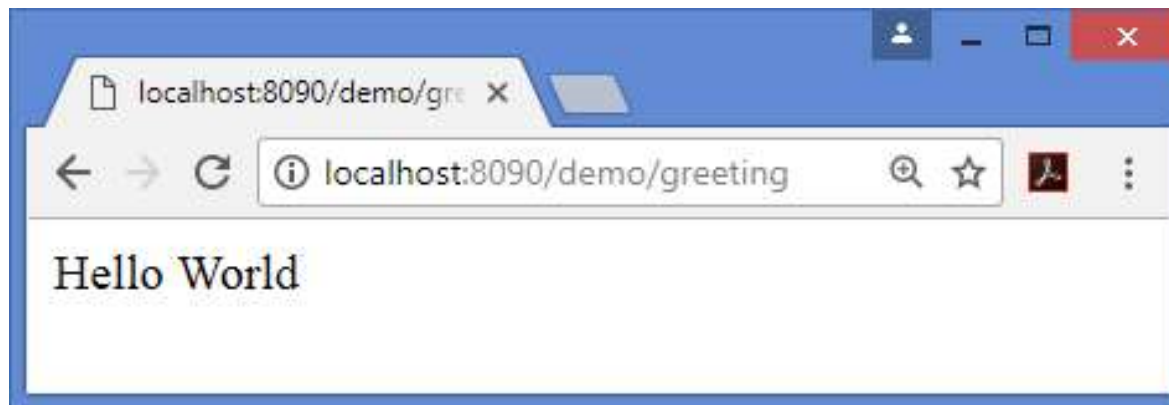
- Runs anywhere Java is setup (think cloud deployments)
- Container is embedded and the app directs how the container works
- Environment configuration is internal to your application

Configuration with application.properties

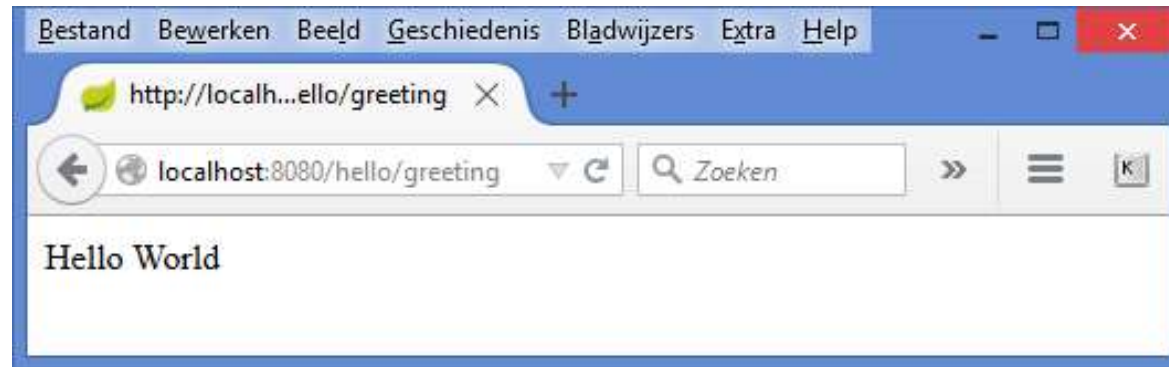


A screenshot of a code editor showing the contents of an `application.properties` file. The file has a tab labeled `application.properties` with a close button. The content consists of two lines of configuration: `1 server.port : 8090` and `2 server.servlet.context-path : /demo`. The first line is highlighted in blue, and the second line is highlighted in light blue.

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



Different URL



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

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

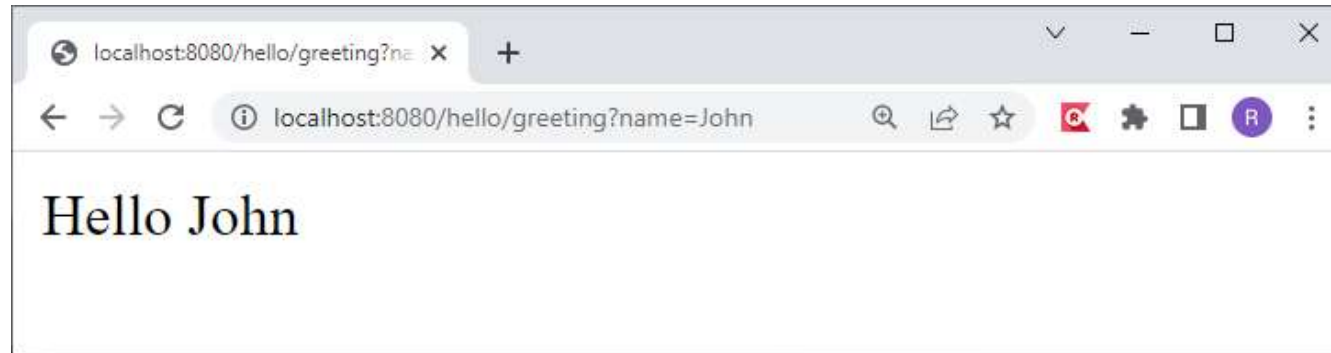
Path variables



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

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

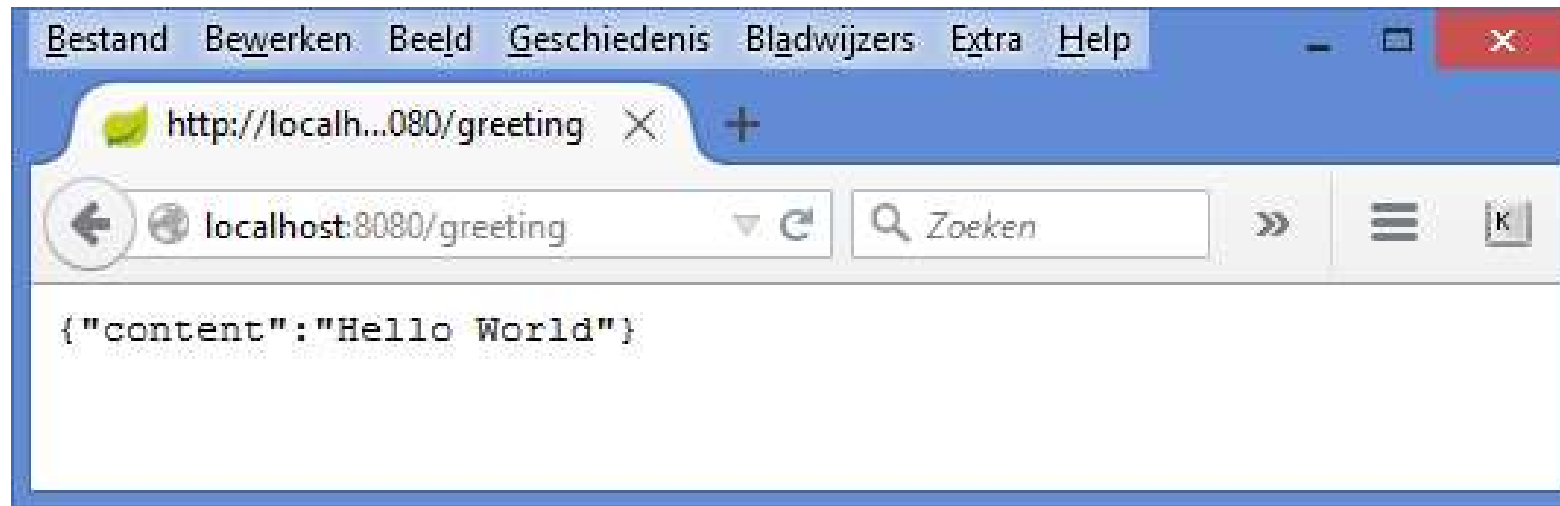
Query parameters



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

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

Returning a class



```
@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;
    }

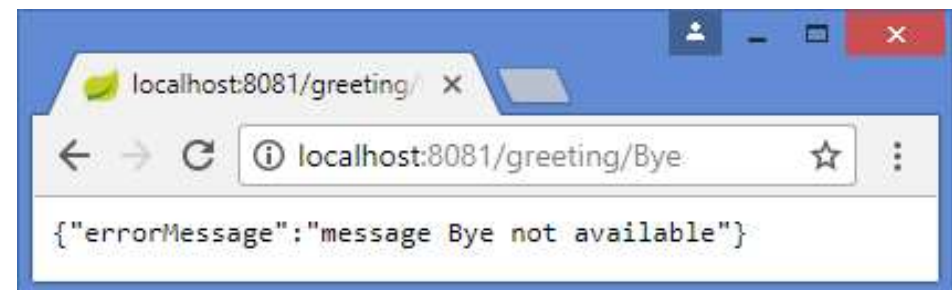
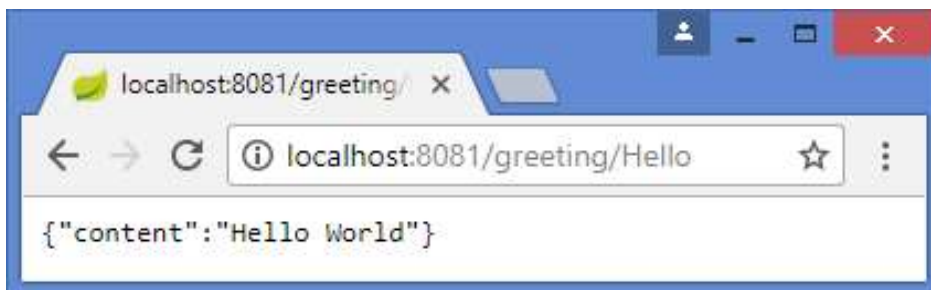
    public String getContent() {
        return content;
    }
}
```


ResponseEntity

```
@RestController
public class GreetingController {

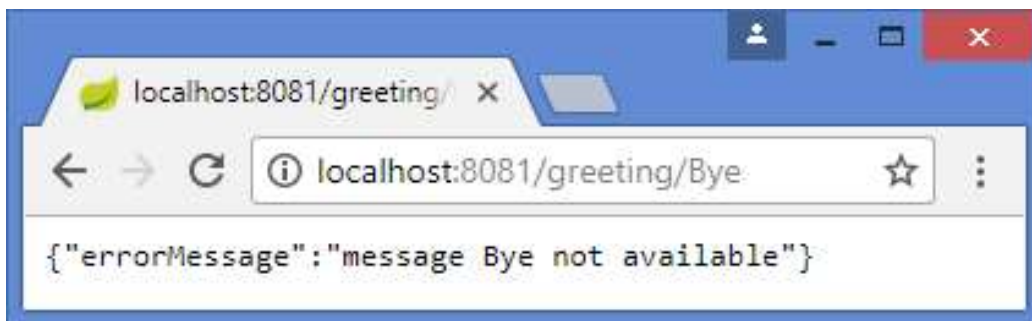
    @RequestMapping("/greeting/{message}")
    public ResponseEntity<?> getGreeting(@PathVariable("message") String message) {
        Greeting greeting = new Greeting("");
        if (message.equals("Hello")) {
            greeting.setContent("Hello World");
        }
        else{
            return new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

Set the content and the HttpStatus



CustomErrorType

```
public class CustomErrorType {  
    private String errorMessage;  
  
    public CustomErrorType(String errorMessage) {  
        this.errorMessage = errorMessage;  
    }  
  
    public String getErrorMessage() {  
        return errorMessage;  
    }  
}
```



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")`

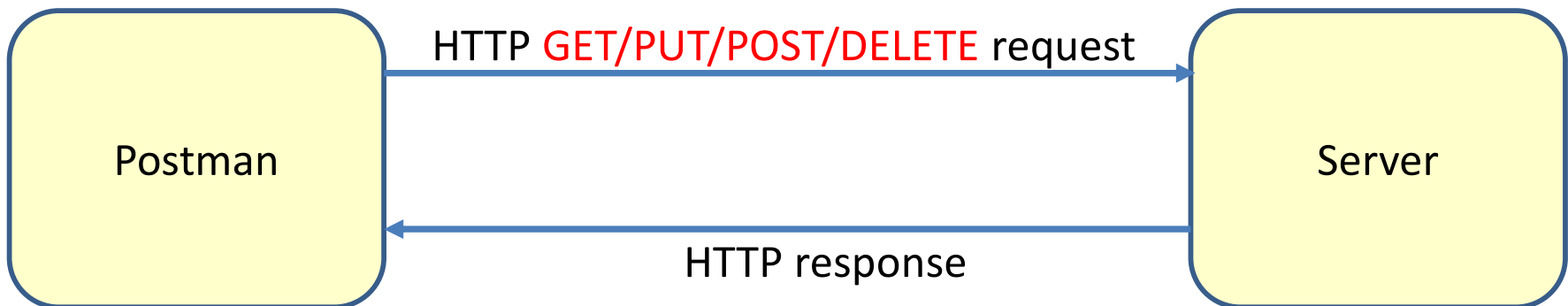
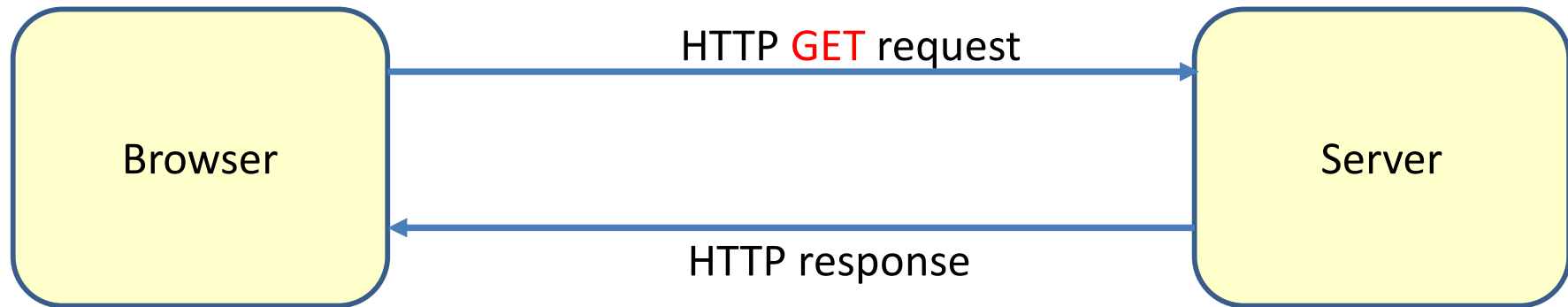
Same

Main point

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

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

REST client



REST Client: Postman

https://swapi.co/api/people/1

GET https://swapi.co/api/people/1...

Params Authorization Headers (1) Body ● Pre-reqs

TYPE

No Auth

Body Cookies (1) Headers (13) Test Results

Pretty Raw Preview JSON

```
1 {
2   "name": "Luke Skywalker",
3   "height": "172",
4   "mass": "77",
5   "hair_color": "blond",
6   "skin_color": "fair",
7   "eye_color": "blue",
8   "birth_year": "198BY",
9   "gender": "male",
10  "homeworld": "https://swapi.co/api/planets/1/",
11  "films": [
12    "https://swapi.co/api/films/2/",
13    "https://swapi.co/api/films/6/",
14    "https://swapi.co/api/films/3/",
15    "https://swapi.co/api/films/1/",
16    "https://swapi.co/api/films/7/"
17  ],
18  "species": [
```

https://swapi.co/api/people/5

GET https://swapi.co/api/people/5...

Params Authorization Headers (1) Body ● Pre-reqs

TYPE

No Auth

Body Cookies (1) Headers (13) Test Results

Pretty Raw Preview JSON

```
1 {
2   "name": "Leia Organa",
3   "height": "150",
4   "mass": "49",
5   "hair_color": "brown",
6   "skin_color": "light",
7   "eye_color": "brown",
8   "birth_year": "198BY",
9   "gender": "female",
10  "homeworld": "https://swapi.co/api/planets/2/",
11  "films": [
12    "https://swapi.co/api/films/2/",
13    "https://swapi.co/api/films/6/",
14    "https://swapi.co/api/films/3/",
15    "https://swapi.co/api/films/1/",
16    "https://swapi.co/api/films/7/"
17  ],
18  "species": [
```

ContactController

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

@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



Add a contact

POST request

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

Get the Contact class from the HTTP request message

Return the object that was send with the POST method

POST

URL

Body

The screenshot shows a REST client interface with a POST request to `localhost:8080/contacts/`. The request body is a JSON object: `{ "firstName": "John", "lastName": "Doe", "email": "jdoe@gmail.com", "phone": "65298765" }`. The response body is the same JSON object. Callouts identify the POST method, URL, request body, and response body.

POST

localhost:8080/contacts/

POST

localhost:8080/contacts/

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

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

```
1 {
2   "firstName": "John",
3   "lastName": "Doe",
4   "email": "jdoe@gmail.com",
5   "phone": "65298765"
6 }
```

raw

JSON

Body of the request

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "firstName": "John",
3   "lastName": "Doe",
4   "email": "jdoe@gmail.com",
5   "phone": "65298765"
6 }
```

Body of the response

Delete a contact

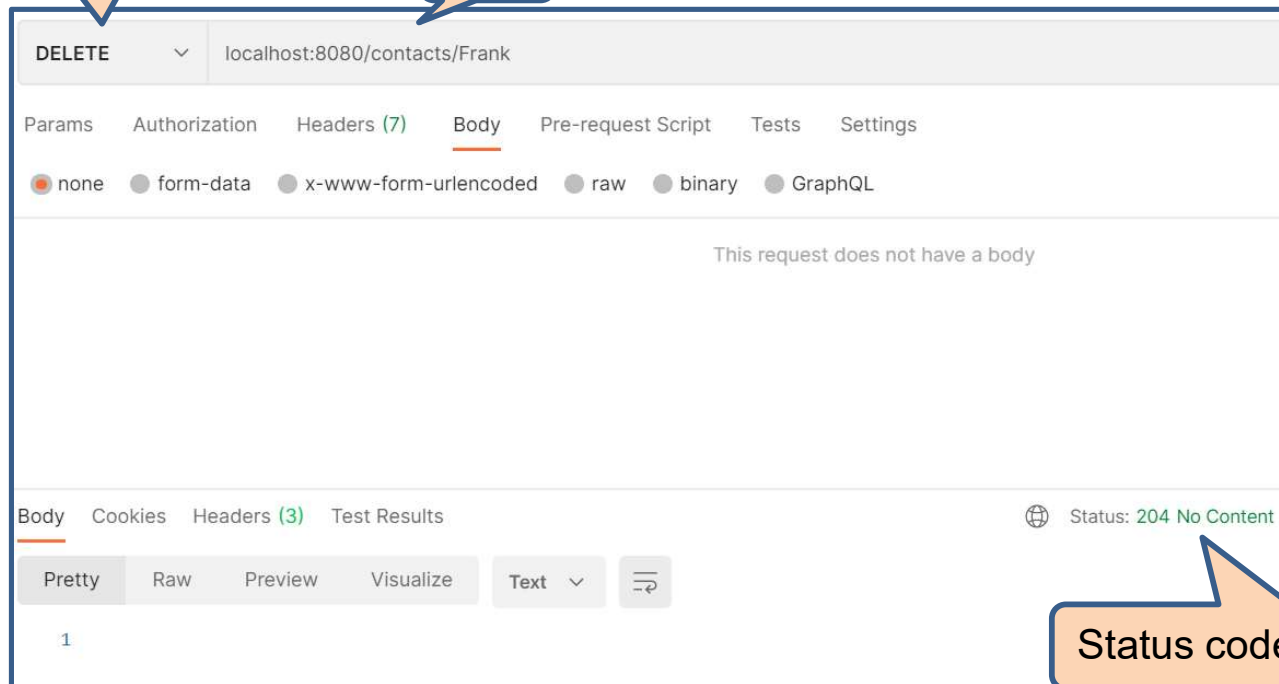
```
@DeleteMapping("/contacts/{firstName}")
public ResponseEntity<?> deleteContact(@PathVariable String firstName) {
    Contact contact = contacts.get(firstName);
    if (contact == null) {
        return new ResponseEntity<CustomErrorType>(new CustomErrorType("Contact with firstname= " +
         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

DELETE request

Return nothing

DELETE

URL



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

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

- Method:** PUT
- URL:** localhost:8080/contacts/
- Body Tab:** Selected, showing a JSON object:

```
{
  "firstName": "Frank",
  "lastName": "Brown",
  "email": "fbrown@gmail.com",
  "phone": "65298765"
}
```
- Response Tab:** Selected, showing the same JSON object as the request body.

```
{
  "firstName": "Frank",
  "lastName": "Brown",
  "email": "fbrown@gmail.com",
  "phone": "65298765"
}
```
- Status:** 200 OK

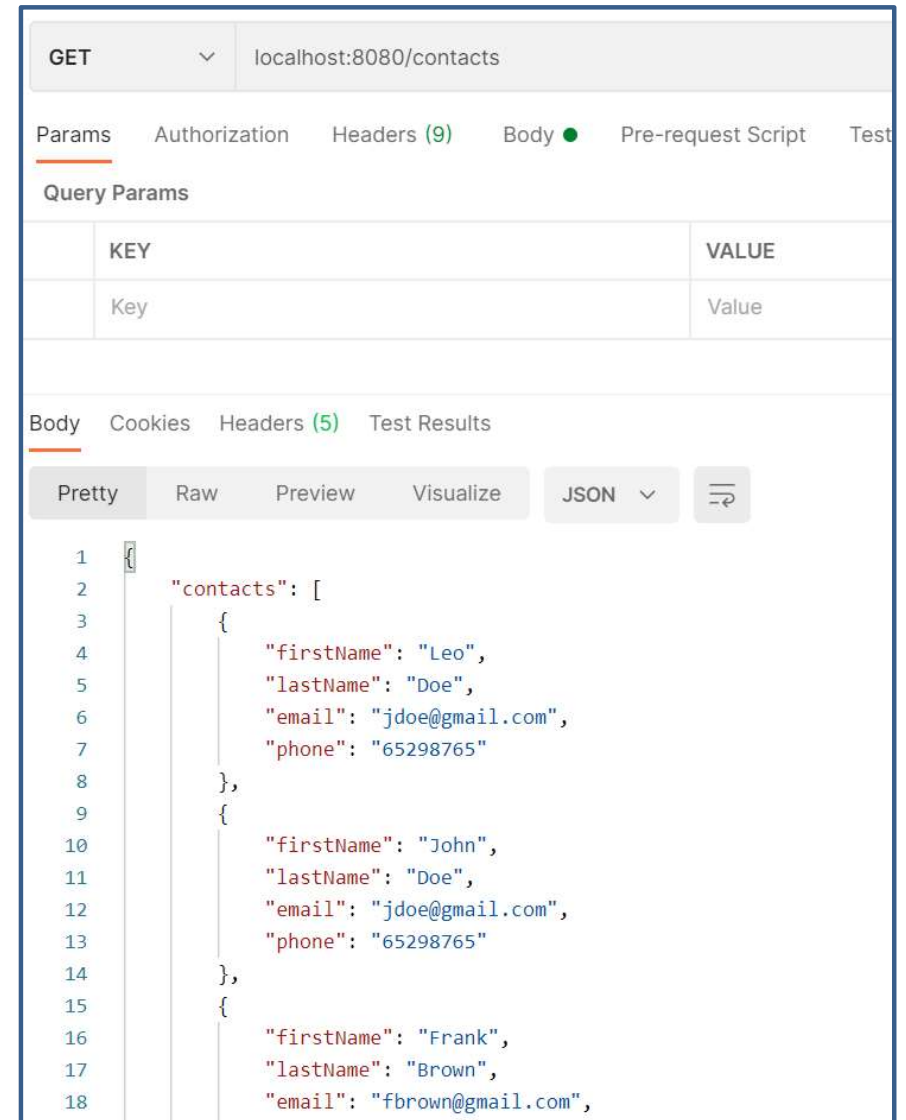
Return the object that was send with the PUT method

Get all contacts

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

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

Create a new class



GET localhost:8080/contacts

Params Authorization Headers (9) Body ● Pre-request Script Test

Query Params

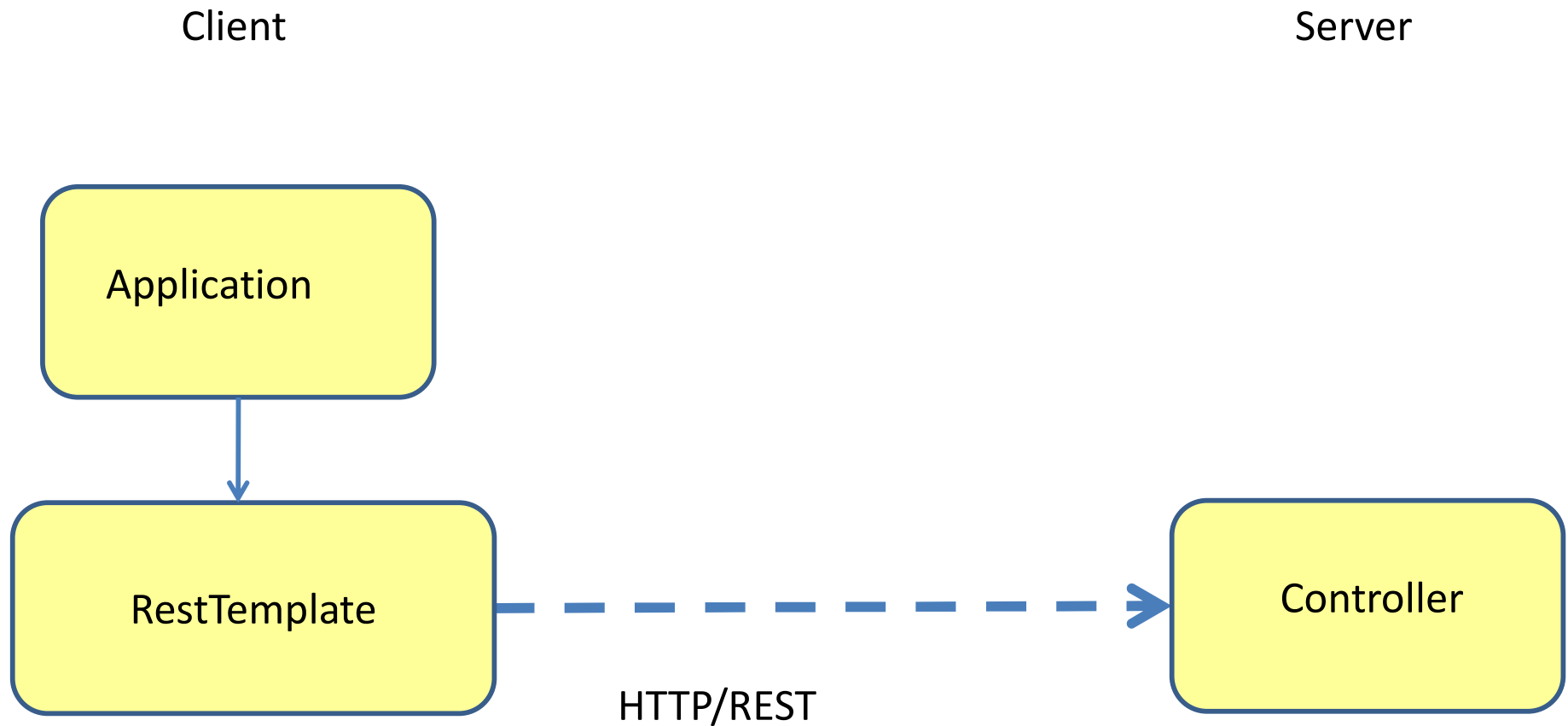
KEY	VALUE
Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "contacts": [
3     {
4       "firstName": "Leo",
5       "lastName": "Doe",
6       "email": "jdoe@gmail.com",
7       "phone": "65298765"
8     },
9     {
10      "firstName": "John",
11      "lastName": "Doe",
12      "email": "jdoe@gmail.com",
13      "phone": "65298765"
14    },
15    {
16      "firstName": "Frank",
17      "lastName": "Brown",
18      "email": "fbrown@gmail.com",
```

Creating a REST client



Creating a Rest client

@SpringBootApplication

```
public class Application implements CommandLineRunner {
```

```
    RestTemplate restTemplate = new RestTemplate();
```

```
    private String serverUrl = "http://localhost:8080/contacts";
```

```
    public static void main(String[] args) {
```

```
        SpringApplication.run(Application.class, args);
```

```
    }
```

@Override

```
public void run(String... args) throws Exception {
```

```
    // get frank
```

```
    Contact contact= restTemplate.getForObject(serverUrl+"/{firstName}", Contact.class, "Frank");
```

```
    System.out.println(contact);
```

```
}
```

```
}
```

Create a RestTemplate

URL (with param)

Return type

Param value

Complete Rest client

@Override

```
public void run(String... args) throws Exception {  
    // get frank  
    Contact contact= restTemplate.getForObject(serverUrl+"/{firstName}", Contact.class, "Frank");  
    System.out.println(contact);  
  
    // add John  
    restTemplate.postForLocation(serverUrl, new Contact("John","Doe", "jdoe@acme.com", "6739127563"));  
  
    // get john  
    contact= restTemplate.getForObject(serverUrl+"/{firstName}", Contact.class, "John");  
    System.out.println(contact);  
  
    // delete mary  
    restTemplate.delete(serverUrl+"/{firstName}", "Mary");  
  
    // update John  
    contact.setEmail("johndoe@acme.com");  
    restTemplate.put(serverUrl+"/{firstName}" , contact, "John");  
  
    // get john  
    contact= restTemplate.getForObject(serverUrl+"/{firstName}", Contact.class, "John");  
    System.out.println(contact);  
    // get all contacts  
    Contacts contacts = restTemplate.getForObject(serverUrl, Contacts.class);  
    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.

REST API DESIGN

Use nouns, not verbs

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

<code>/getCustomers</code>	<code>/saveCustomers</code>
<code>/getCustomersByName</code>	<code>/getCustomersByPhone</code>
<code>/getCustomersByContact</code>	<code>/getCustomersUsingPaging</code>
<code>/getNewCustomers</code>	<code>/getCurrentCustomers</code>
<code>/createNewCustomer</code>	<code>/deleteCustomer</code>



NOT OK

REST API design best practices

- Use verbs

Resource	GET (read)	POST (create)	PUT (update)	DELETE (delete)
/customers	Get List	Create Item	Update Batch	Error
/customers/123	Get Item	Error	Update Item	Delete Item



OK

- 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

Sub-objects



`http://.../api/Customers/123/Invoices`

`http://.../api/Games/halo-3/Ratings`

`http://.../api/Invoices/2003-01-24/Payments`

More complex functionality

- Use request parameters

```
http://.../api/Customers?state=GA
```

```
http://.../api/Customers?state=GA&salesperson=144
```

```
http://.../api/Customers?hasOpenOrders=true
```

Request parameters

```
public class Calculator {  
    private int value;  
  
    public int add(int number){  
        value = value + number;  
        return value;  
    }  
  
    public int subtract(int number){  
        value = value - number;  
        return value;  
    }  
  
    public int getValue() {  
        return value;  
    }  
  
    public void setValue(int value) {  
        this.value = value;  
    }  
}
```

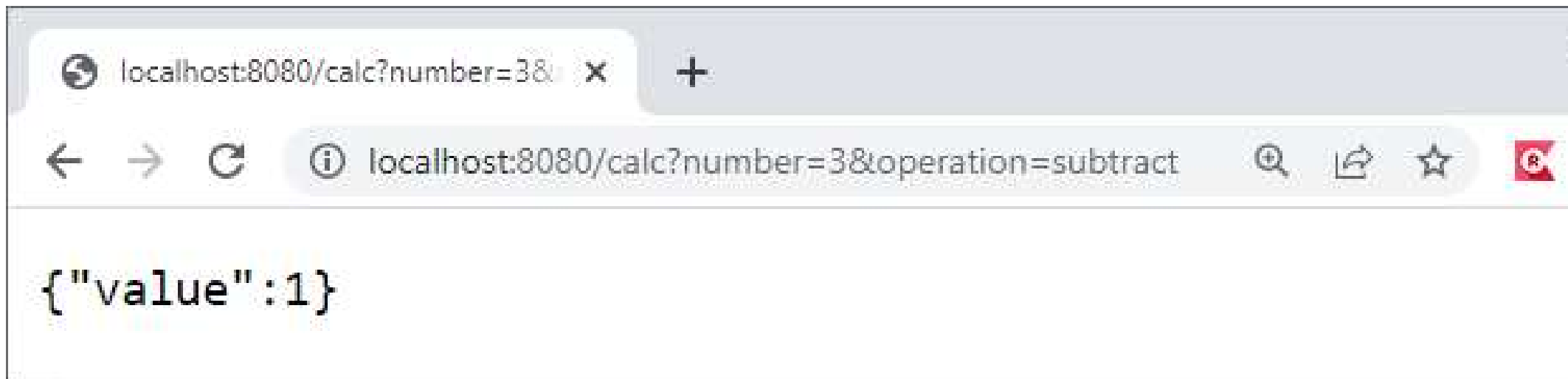
@RestController

```
public class CalcController {  
    private Calculator calculator = new Calculator();
```

@GetMapping("/calc")

```
public ResponseEntity<?> calculate(@RequestParam(value= "number") int number,  
                                   @RequestParam(value= "operation") String operation) {  
    if (operation.equals("add")) calculator.add(number);  
    if (operation.equals("subtract")) calculator.subtract(number);  
    return new ResponseEntity<Calculator>(calculator, HttpStatus.OK);  
}
```

Request parameters



Connecting the parts of knowledge with the wholeness of knowledge

1. Rest webservises 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.

