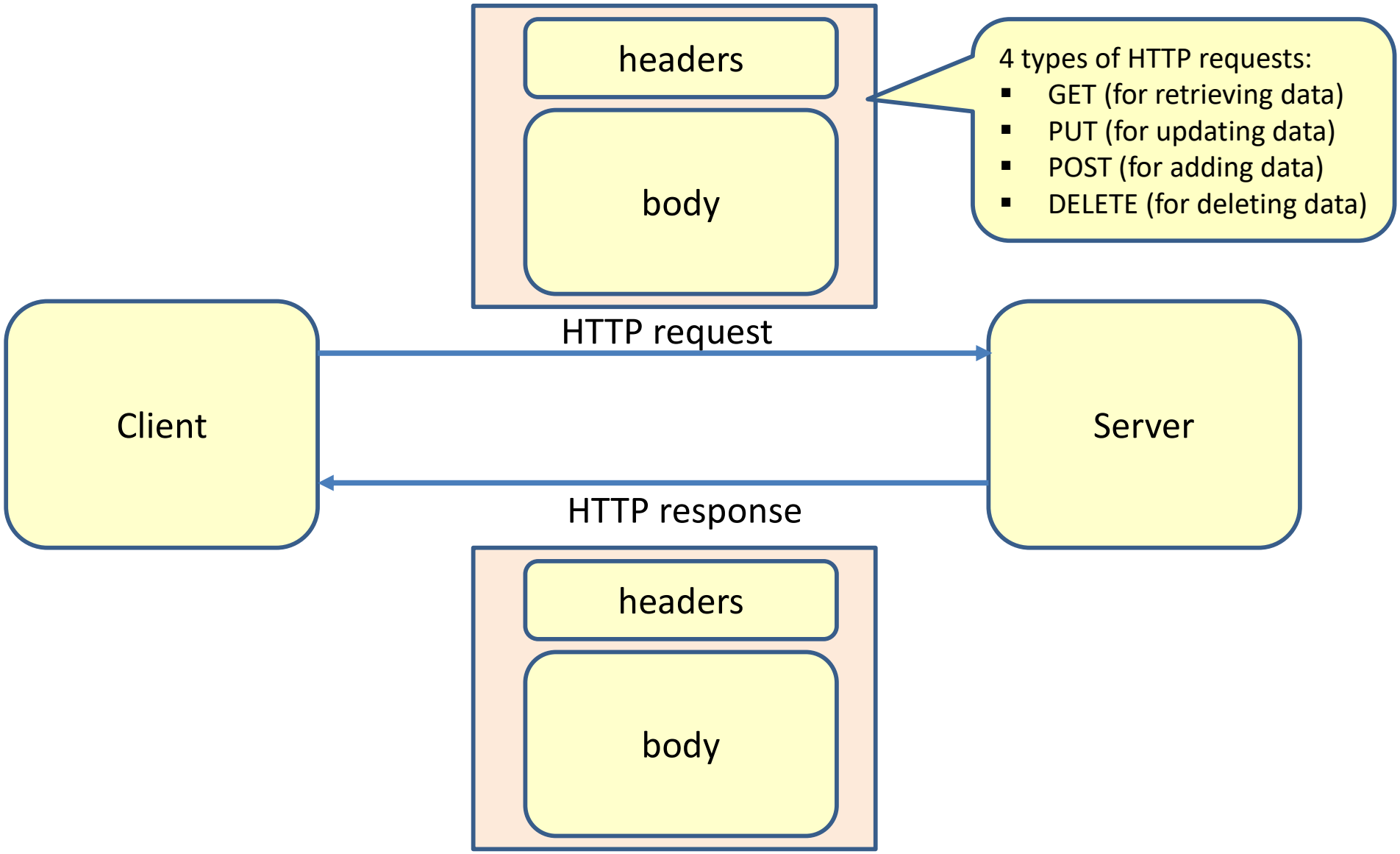
CS544

LESSON 7
REST WEBSERVICES

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
March 28 **Lesson 1** Enterprise Architecture introduction and Spring Boot	March 29 **Lesson 2** Dependency injection AOP	March 30 **Lesson 3** JDBC JPA	March 31 **Lesson 4** JPA mapping 1	April 1 **Lesson 5** JPA mapping 2	April 2 **Lesson 6** JPA queries	April 3
April 4 **Lesson 7** Transactions	April 5 **Lesson 8** MongoDB	April 6 **Midterm Review**	April 7 **Midterm exam**	April 8 **Lesson 9** REST webservices	April 9 **Lesson 10** SOAP webservices	April 10
April 11 **Lesson 11** Messaging	April 12 **Lesson 12** Scheduling Events Configuration	April 13 **Lesson 13** Monitoring	April 14 **Lesson 14** Testing your application	April 15 **Final review**	April 16 **Final exam**	April 17
April 18 **Project**	April 19 **Project**	April 20 **Project**	April 21 **Presentations**			

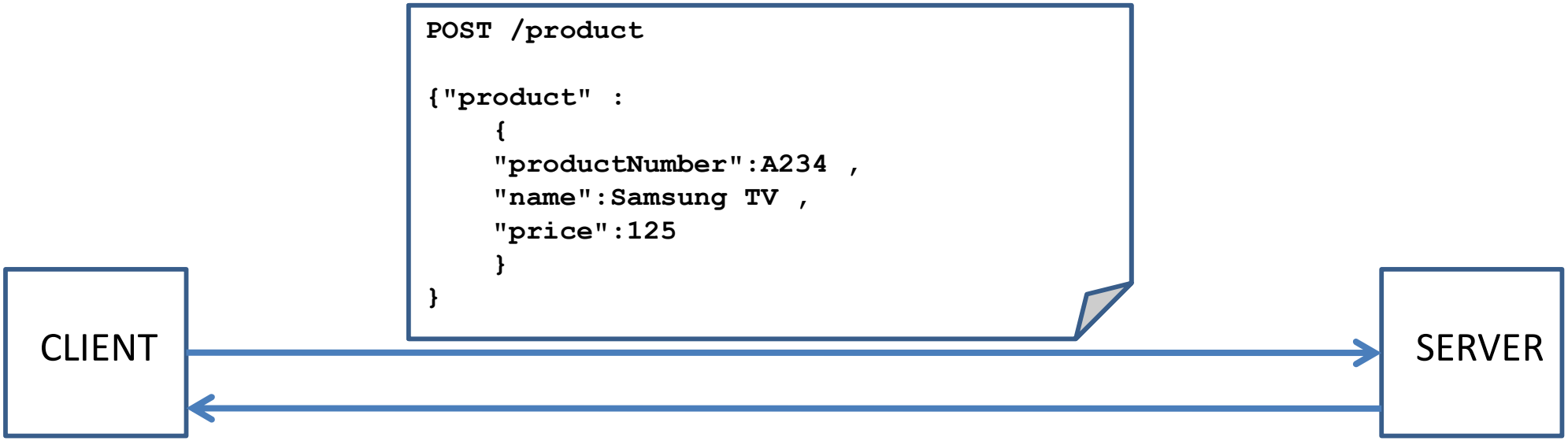
REST webservices



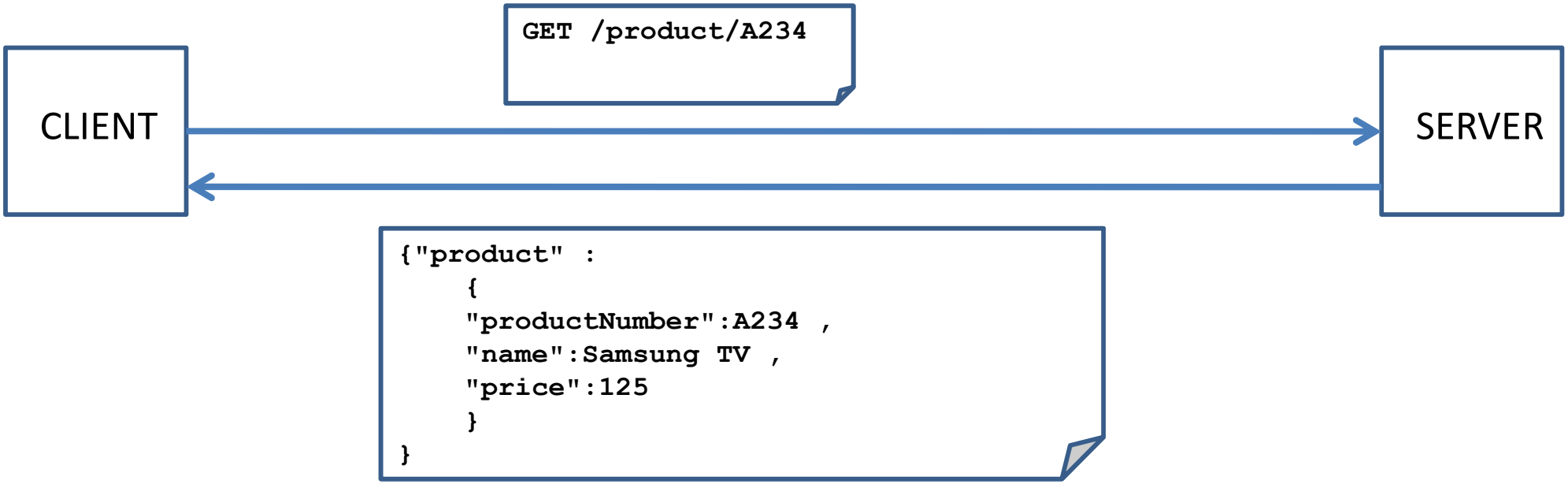
Http methods

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 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

```
@RestController
public class GreetingController {

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

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

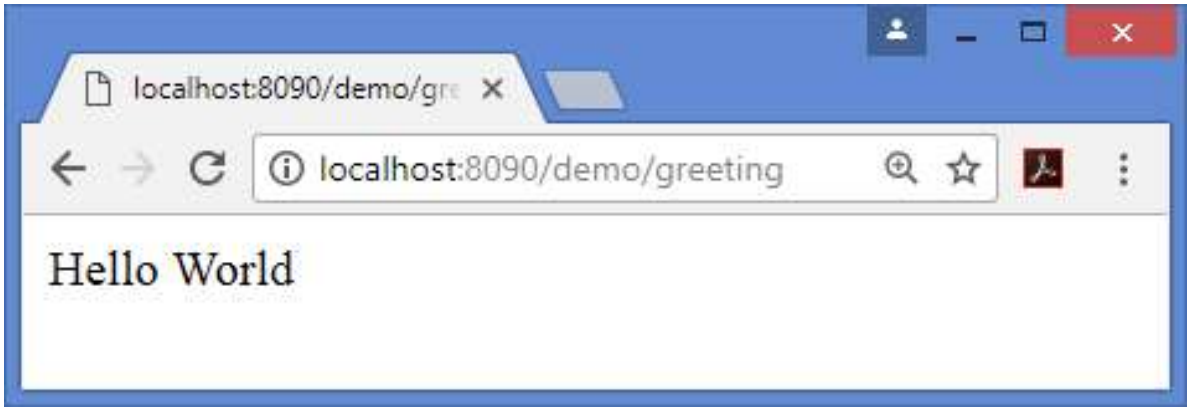
Application Deployments

- 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

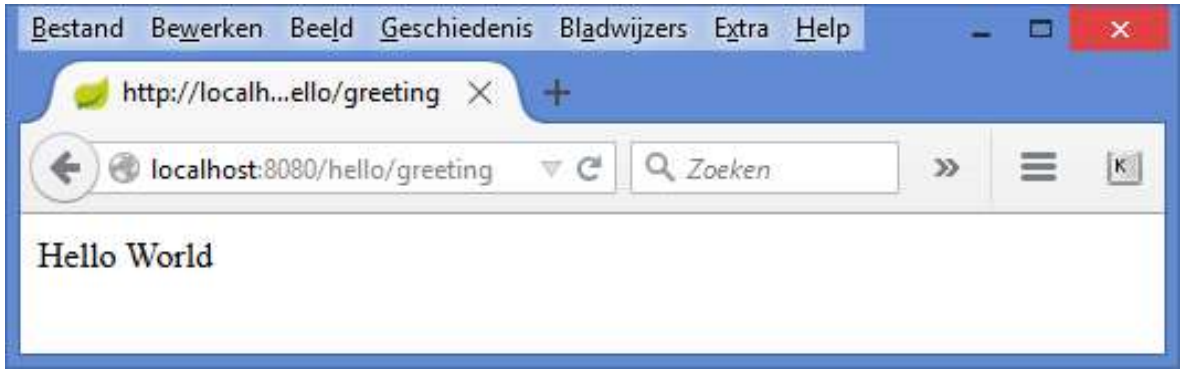
Configuration with application.properties

application.properties

```
1 server.port : 8090
2 server.servlet.context-path : /demo
3
```

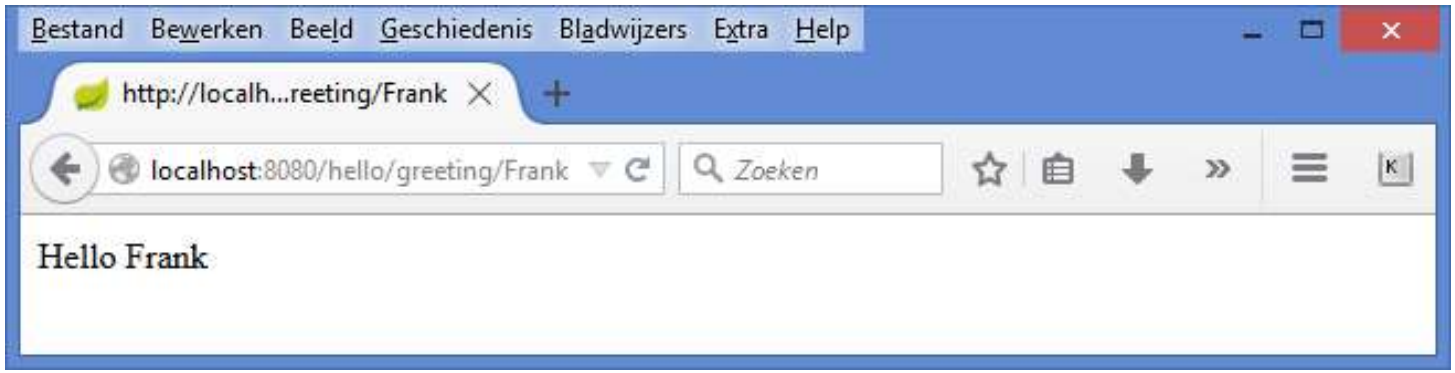


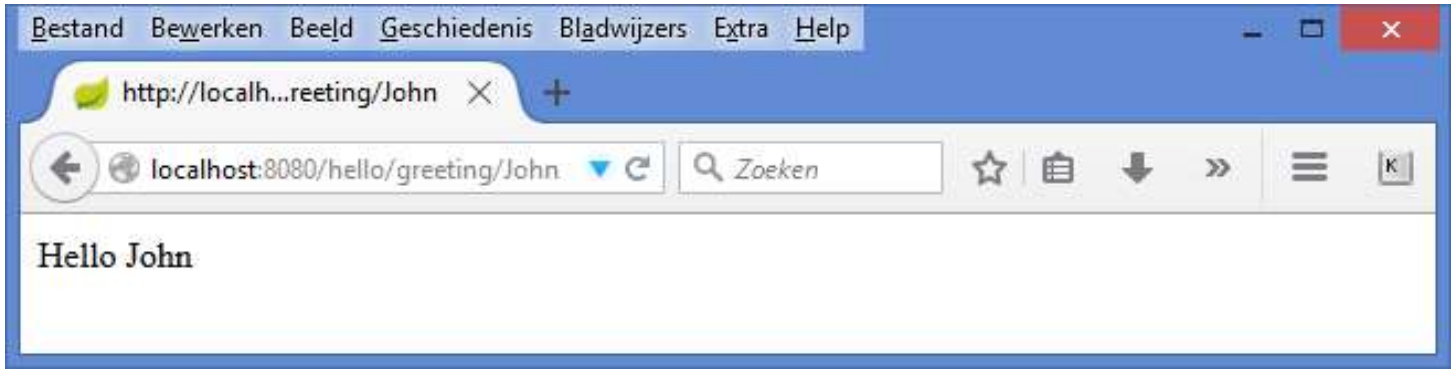
Different URL




```
@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");
    }
}
```

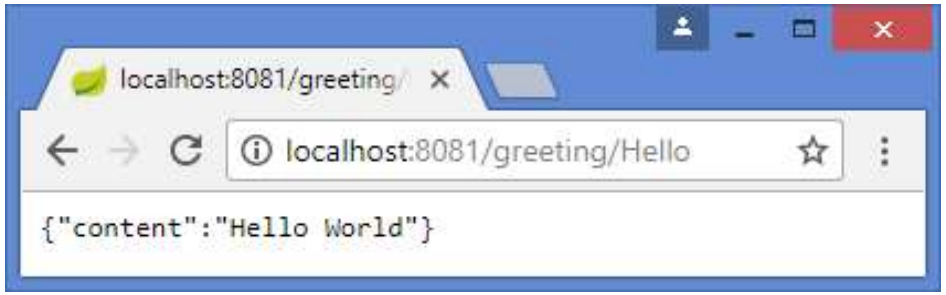
Return a Greeting class

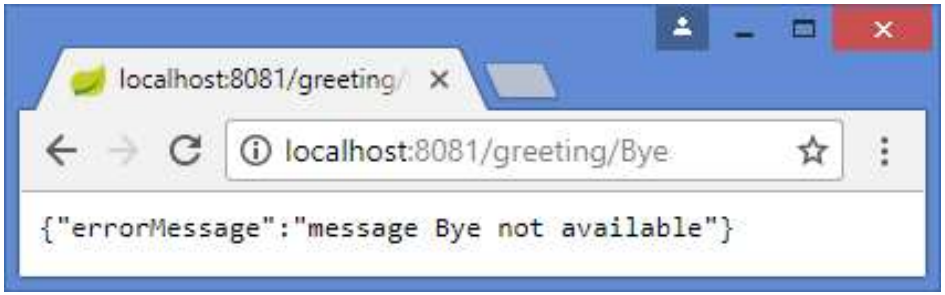
```
public class Greeting {

  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);
    }
}
```

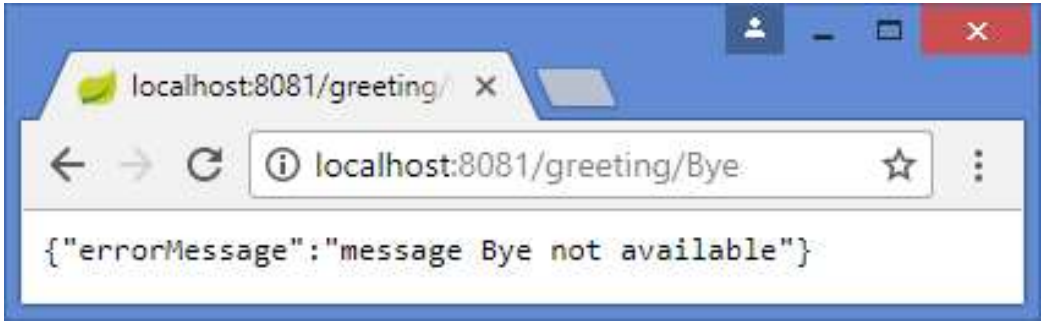
Set the content and the HttpStatus





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")
```

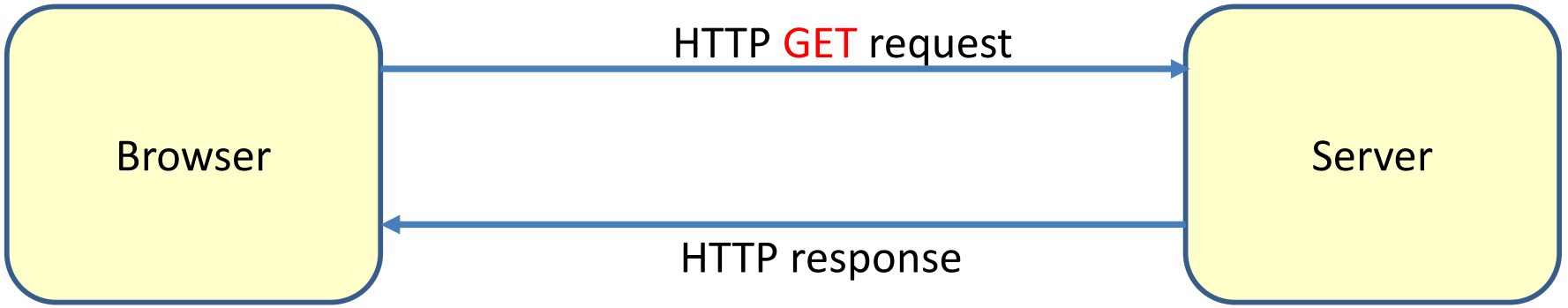
Same

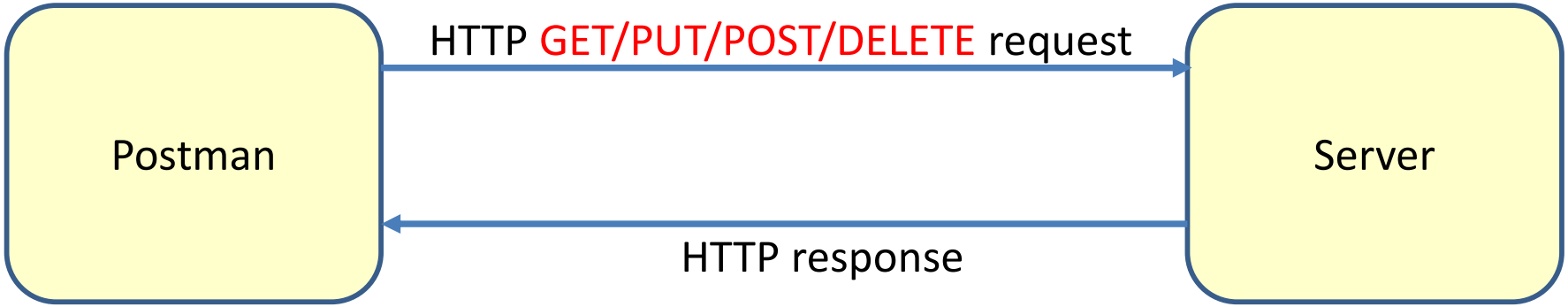
Main point

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

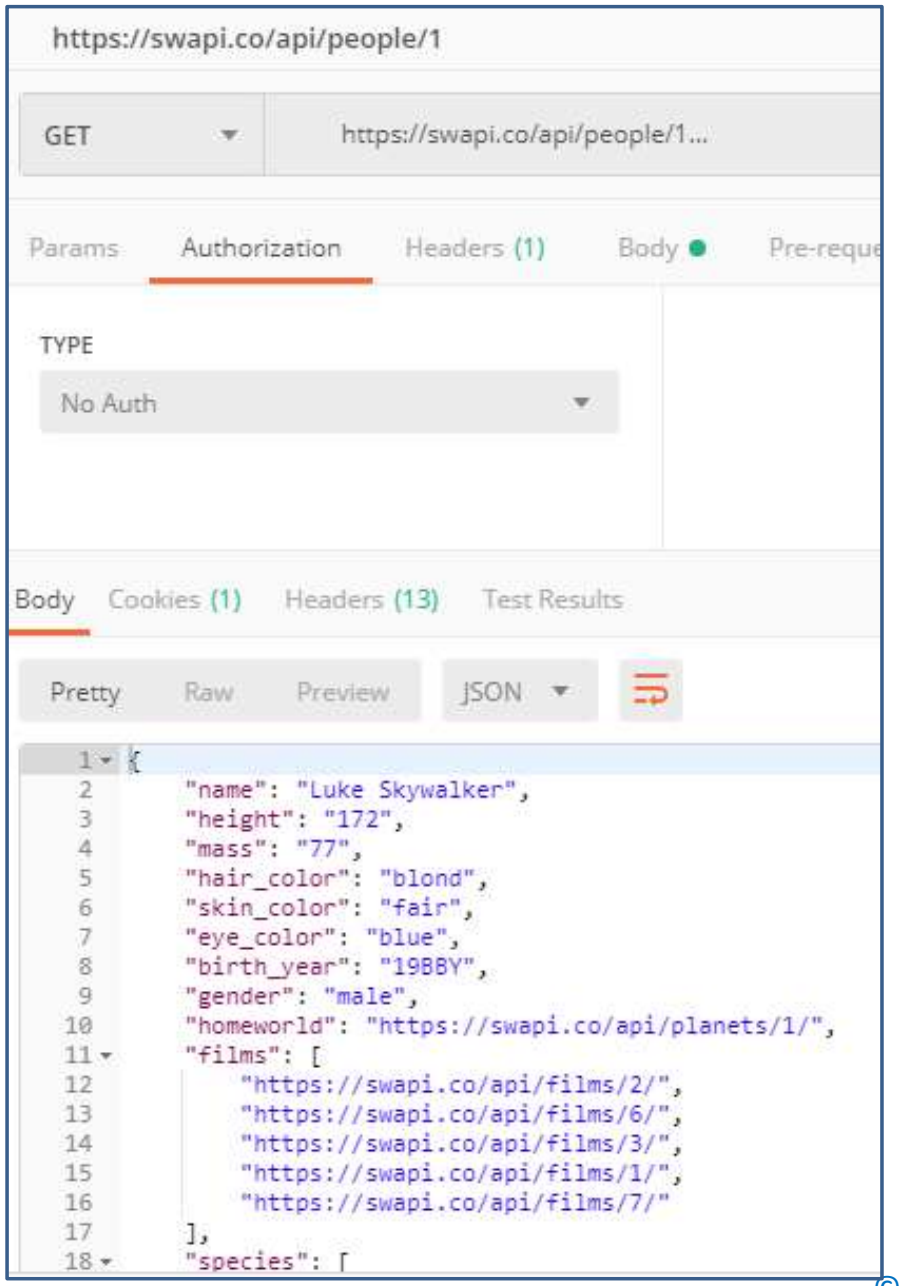
  Science of Consciousness: The human nervous system has the natural ability to transcend and experience pure consciousness.

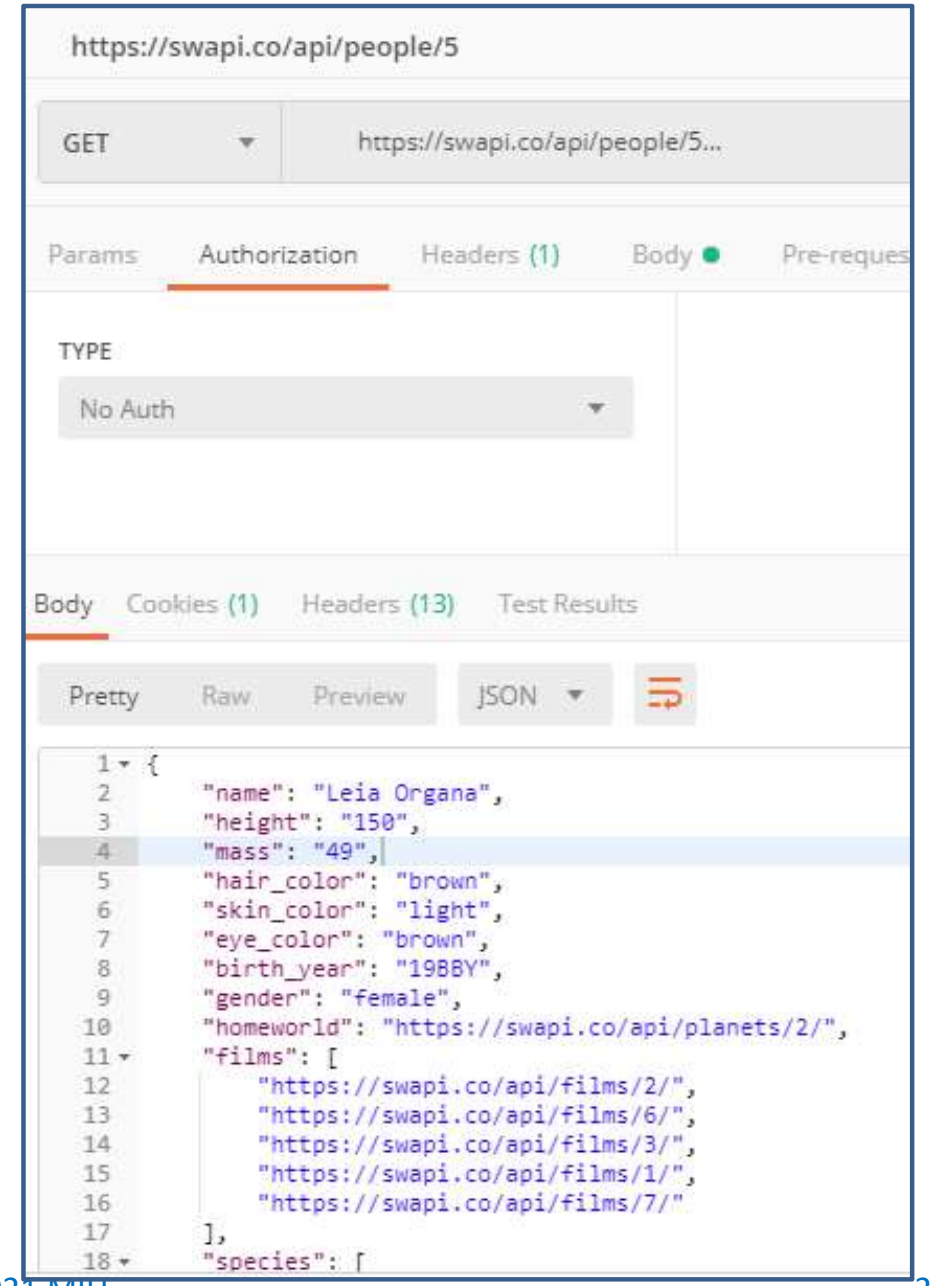
REST client


Browser
HTTP GET request
Server
HTTP response


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

REST Client: Postman





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

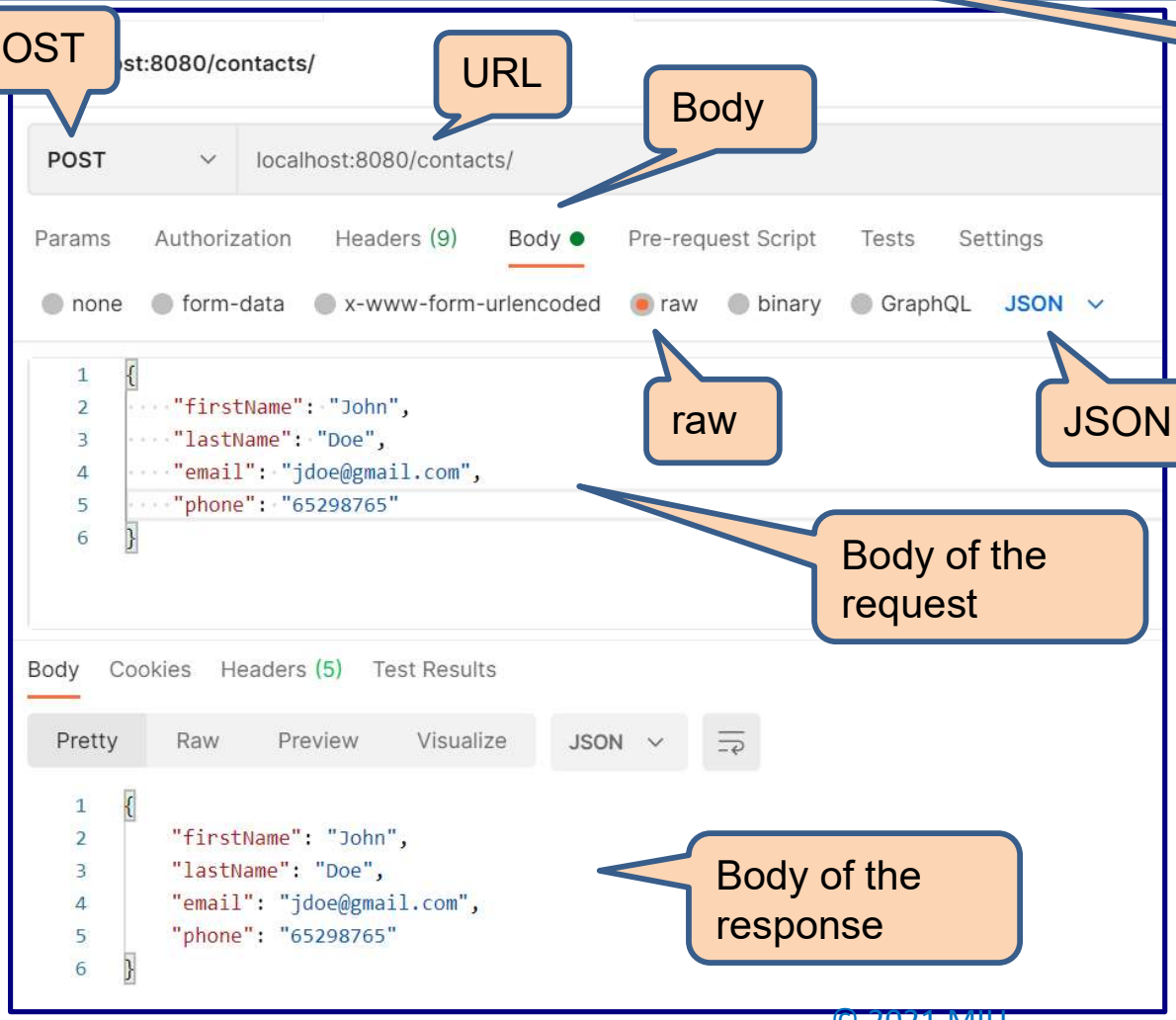
Add a contact

```
@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



Delete a contact

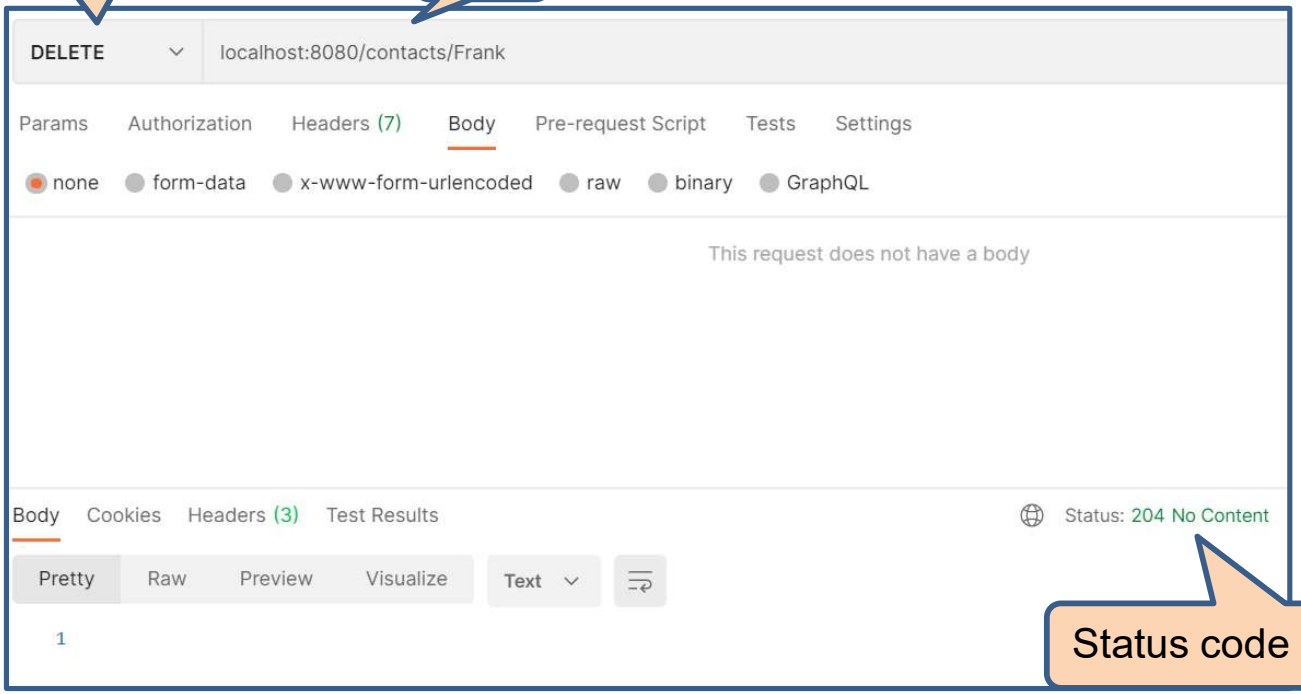
```
@DeleteMapping("/contacts/{firstName}")
public ResponseEntity<?> deleteContact(@PathVariable String firstName) {
   Contact contact = contacts.get(firstName);
   if (contact == null) {
      return new ResponseEntity<CustomErrorType>(new CustomErrorType("Contact with firstname= " +
            firstName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

DELETE request

Return nothing

DELETE

URL



Status code

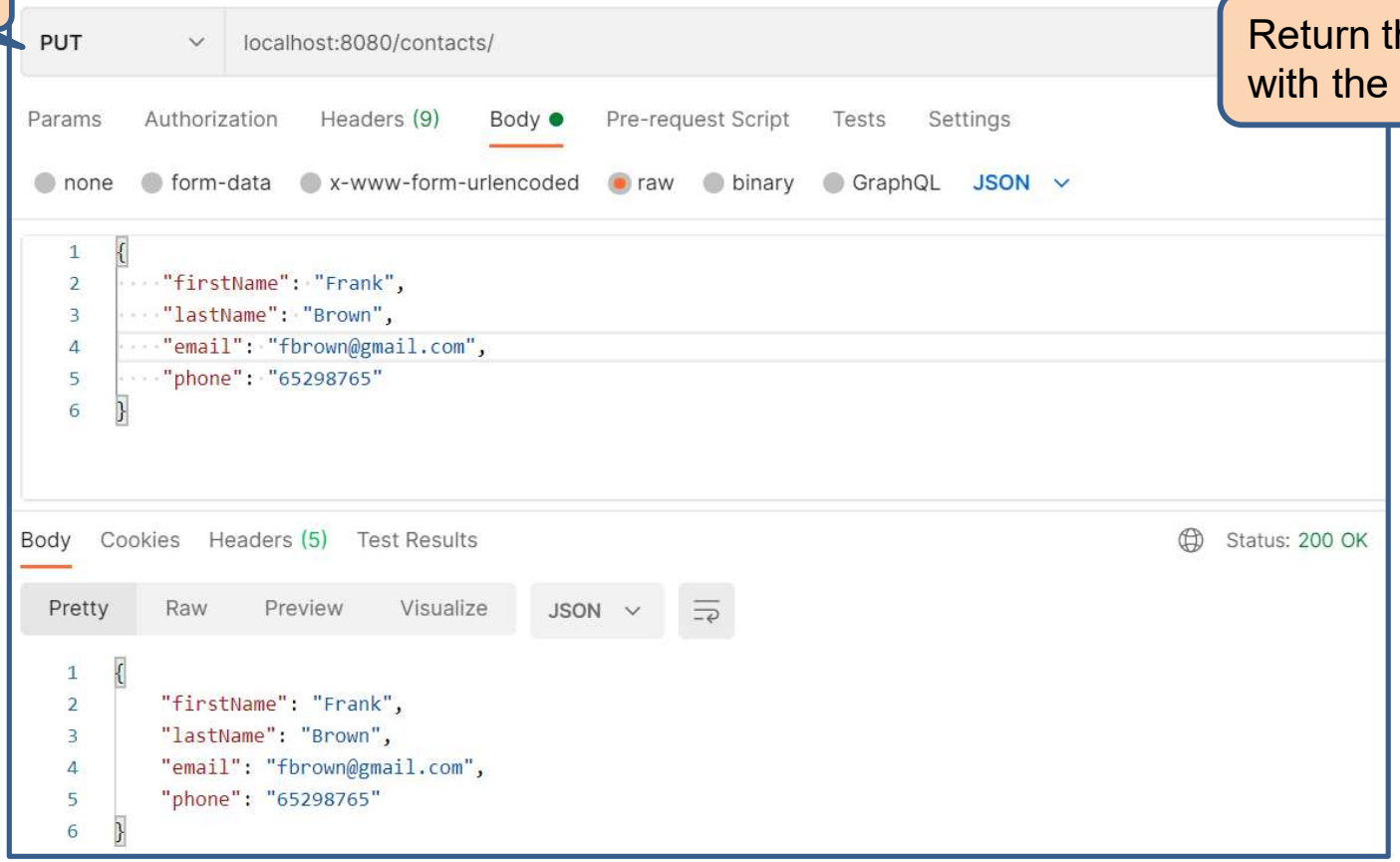
Update a contact

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

Return the object that was send with the PUT method

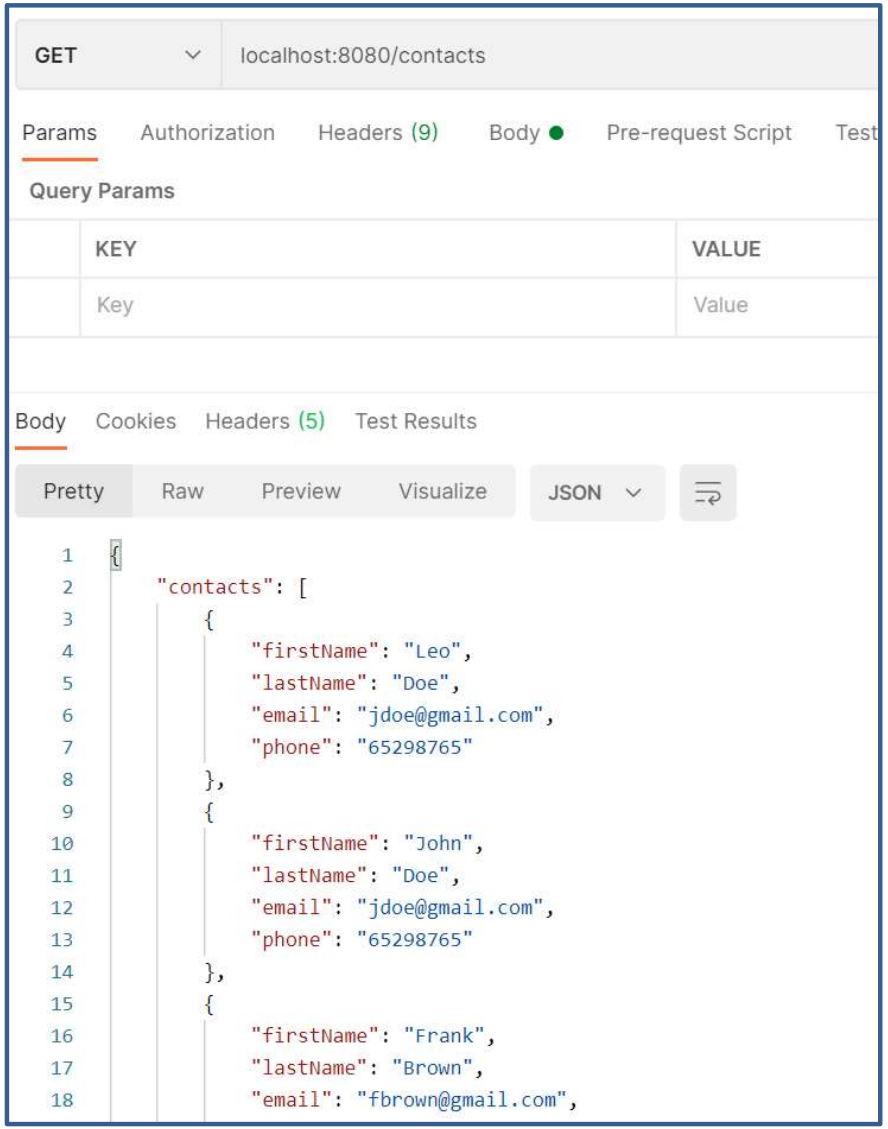


Get all contacts

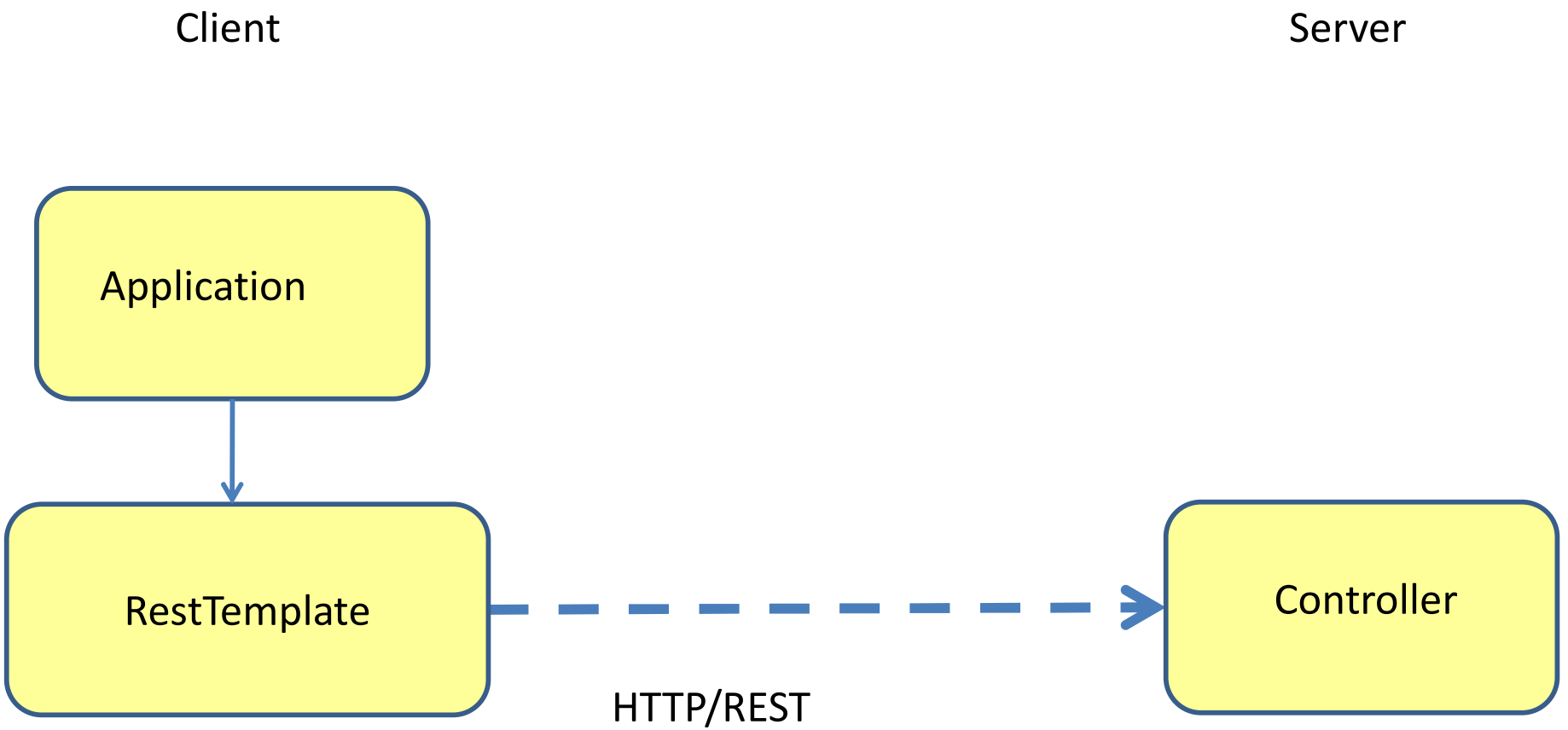
```java
@GetMapping("/contacts")
public ResponseEntity<?> getAllContacts() {
    Contacts allcontacs = new Contacts(contacts.values());
    return new ResponseEntity<Contacts>(allcontacs, HttpStatus.OK);
}
```

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

...
```

Create a new class



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:

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

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 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.