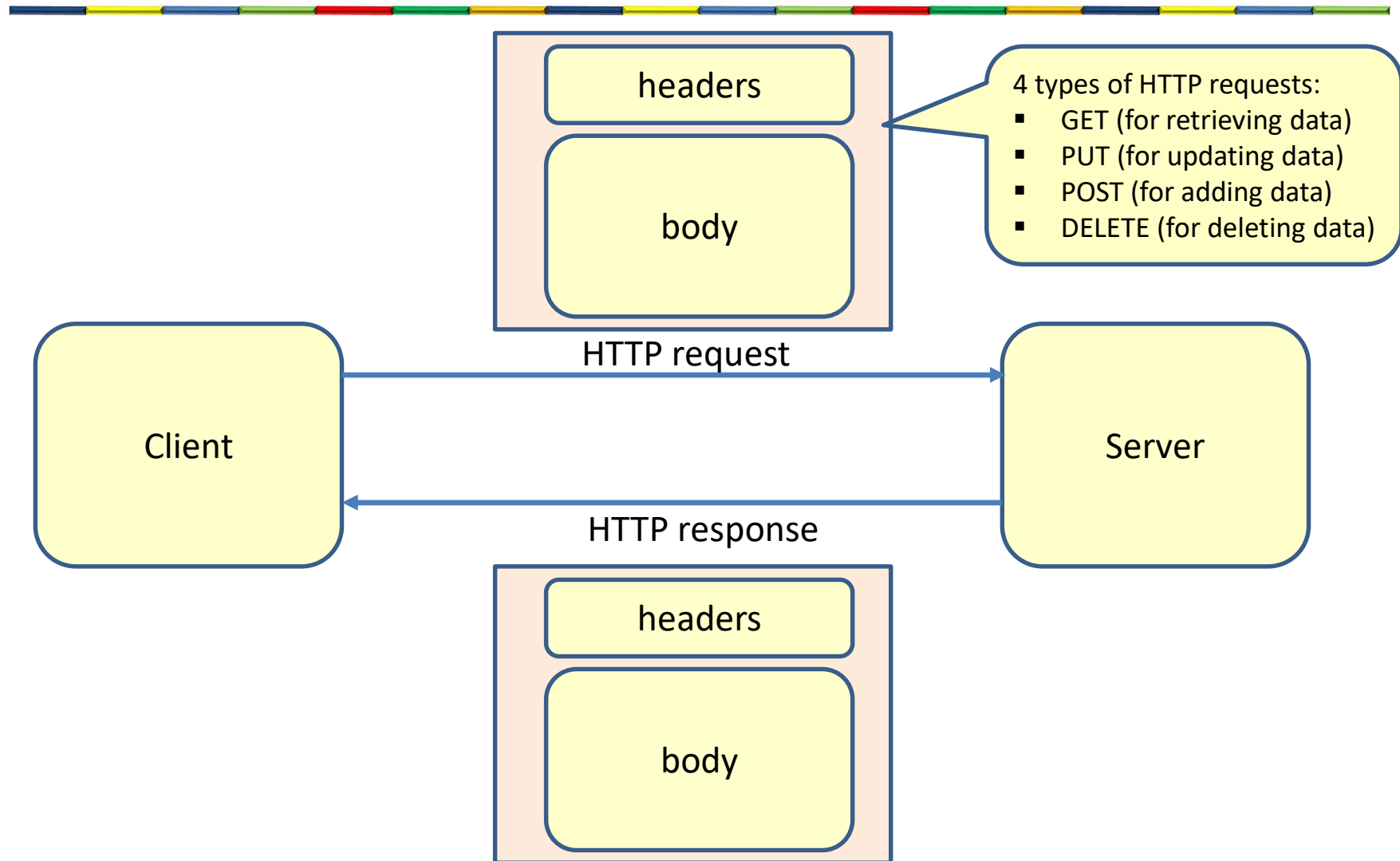


Lesson 3

REST API DESIGN TESTING REST

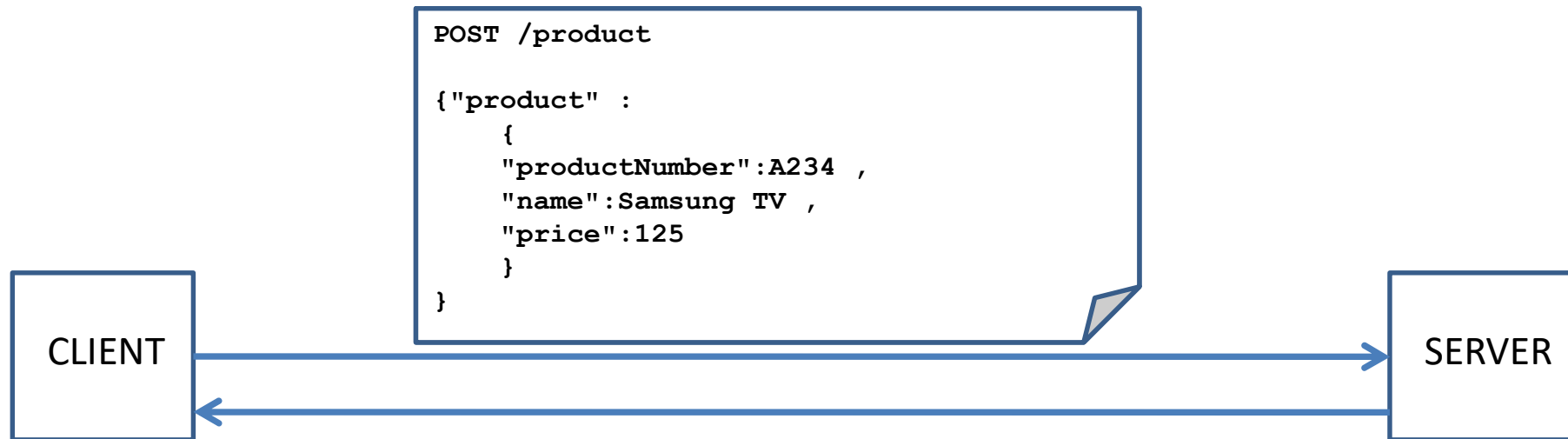
REST webservice



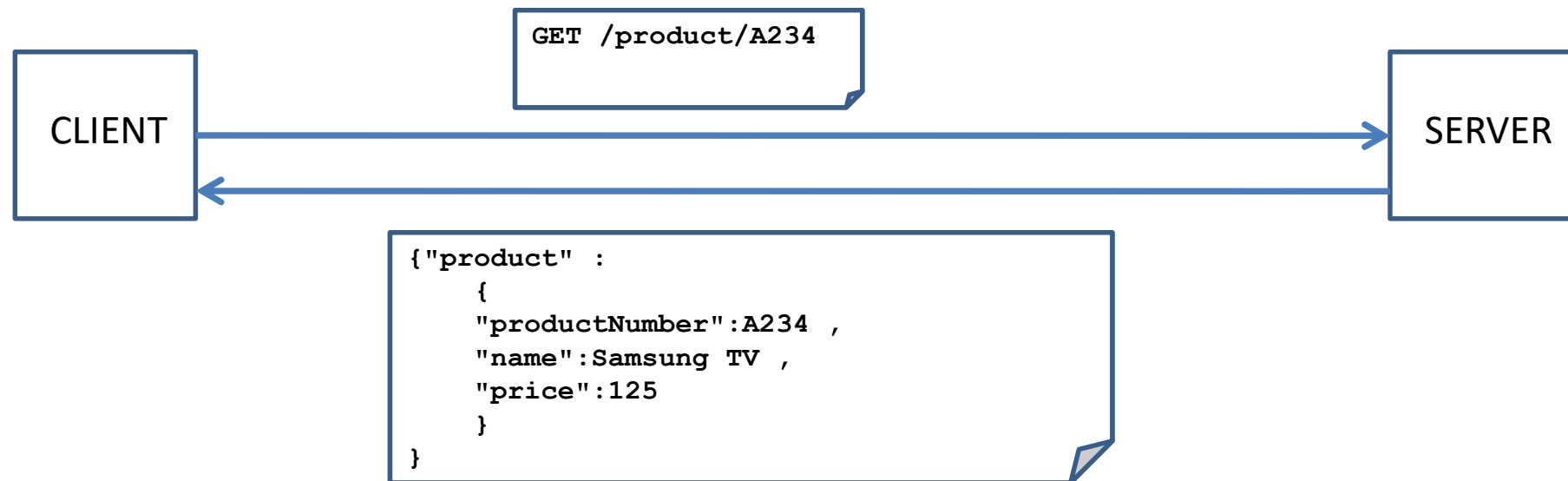
Http methods

Method	Idempotent
GET	YES
POST	NO
PUT	YES
DELETE	YES

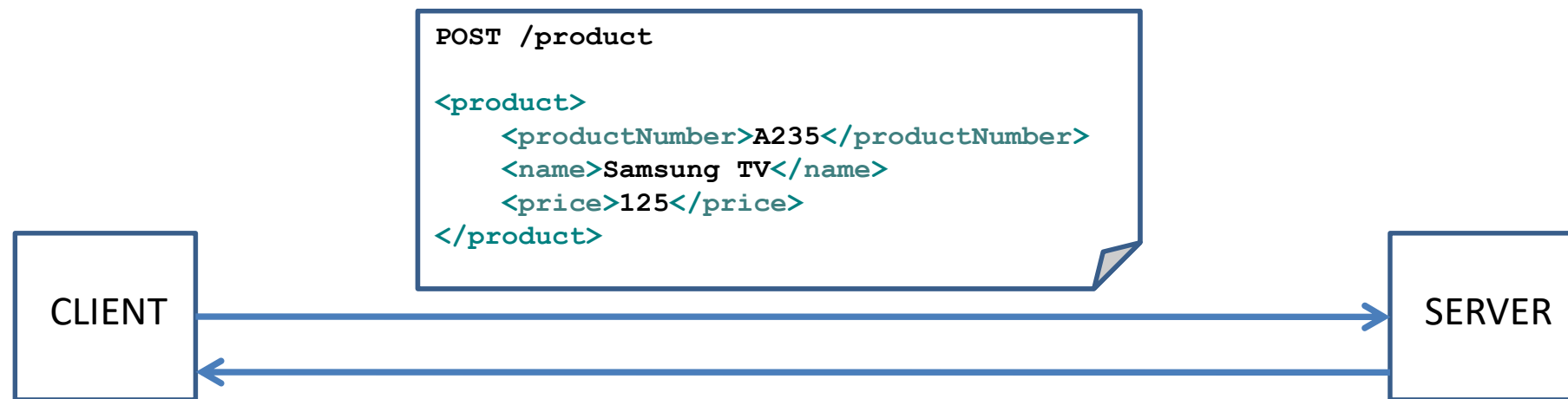
POST method using JSON



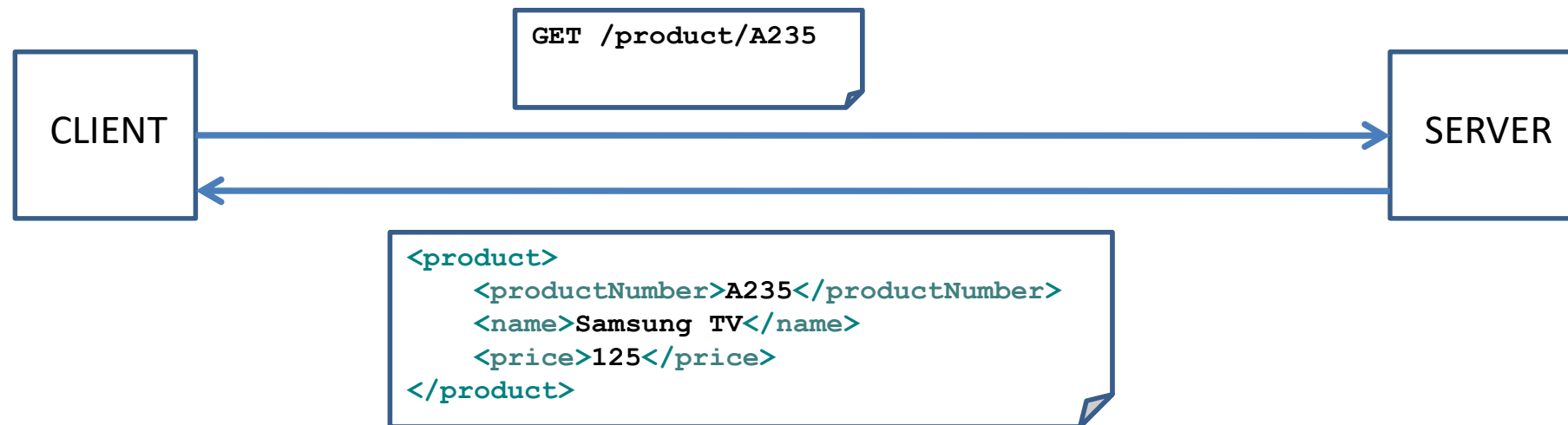
GET method using JSON



POST method using XML



GET method using XML



XML vs. JSON

```
<empinfo>
  <employees>
    <employee>
      <name>Scott Philip</name>
      <salary>£44k</salary>
      <age>27</age>
    </employee>
    <employee>
      <name>Tim Henn</name>
      <salary>£40k</salary>
      <age>27</age>
    </employee>
    <employee>
      <name>Long yong</name>
      <salary>£40k</salary>
      <age>28</age>
    </employee>
  </employees>
</empinfo>
```

```
{ "empinfo" :
  {
    "employees" : [
      {
        "name" : "Scott Philip",
        "salary" : £44k,
        "age" : 27,
      },
      {
        "name" : "Tim Henn",
        "salary" : £40k,
        "age" : 27,
      },
      {
        "name" : "Long Yong",
        "salary" : £40k,
        "age" : 28,
      }
    ]
  }
}
```


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

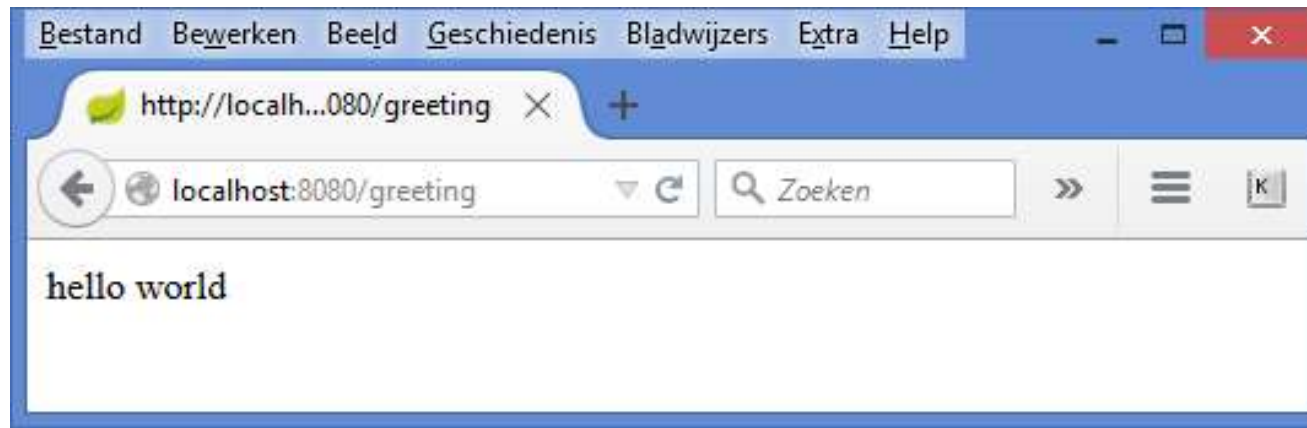
Simple Rest Example: configuration

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);
    }
}
```

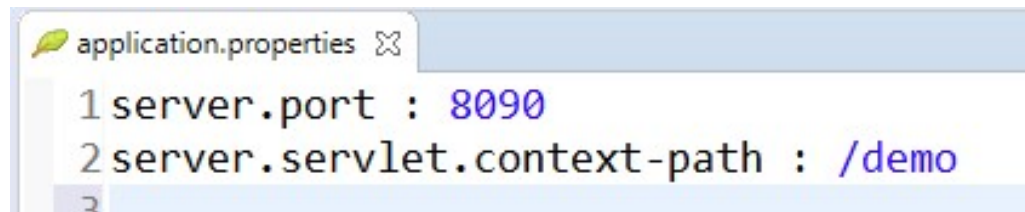
Simple Rest Example: calling the service



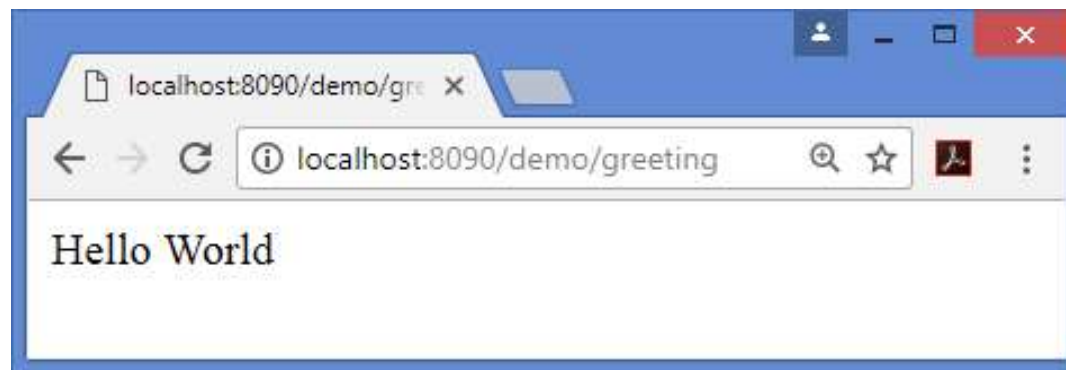
```
@RestController
public class GreetingController {

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

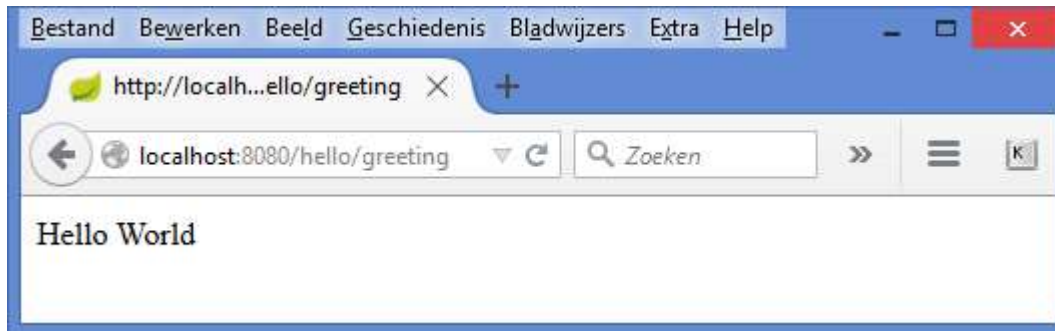
Configuration with application.properties

A screenshot of a text editor window titled 'application.properties'. It contains two lines of configuration: '1 server.port : 8090' and '2 server.servlet.context-path : /demo'.

```
application.properties
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";
    }
}
```

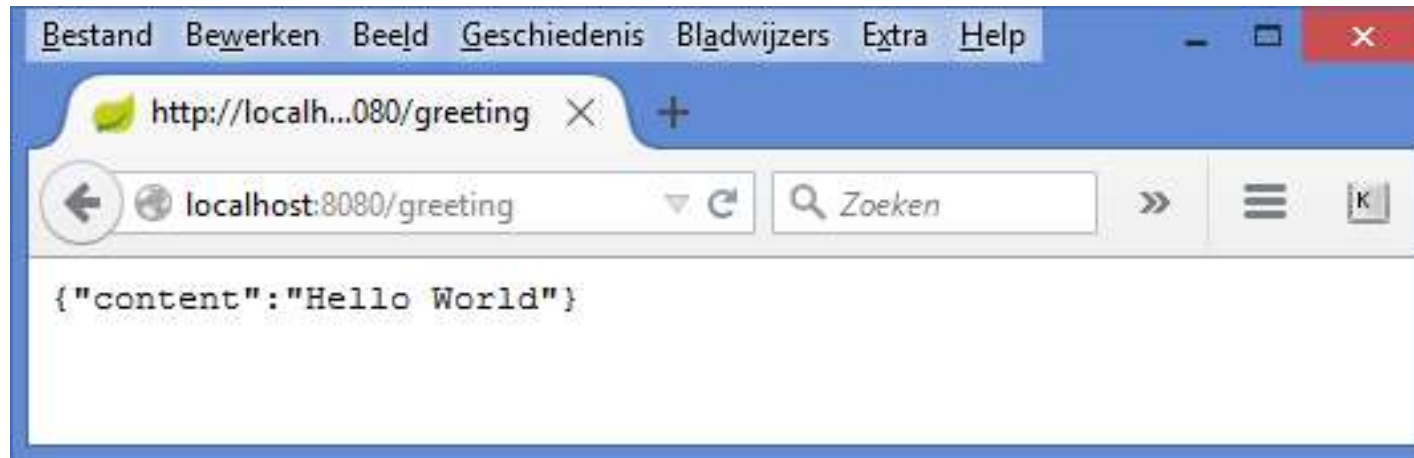
Path variables



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

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

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;
    }
}
```


XML instead of JSON

```
@XmlElement
public class Greeting {
    @XmlElement
    private String content="";

    public Greeting() {}

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

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

JAXB annotations

Dit XML-bestand lijkt geen geassocieerde stijlinformatie te hebben. De documentstructuur is hieronder weergegeven.

```
- <greeting>
  <content>Hello World XML</content>
</greeting>
```

Supporting XML and JSON

```
@Controller
public class GreetingController {

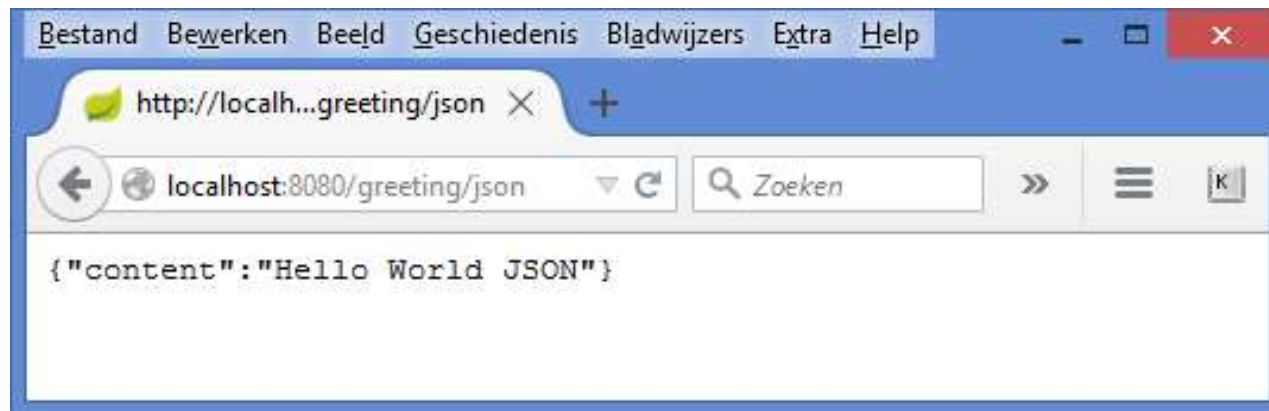
    @RequestMapping(value="/greeting/json", produces="application/json")
    public @ResponseBody Greeting greetingJSON() {
        return new Greeting("Hello World JSON");
    }

    @RequestMapping(value="/greeting/xml", produces="application/xml")
    public @ResponseBody Greeting greetingXML() {
        return new Greeting("Hello World XML");
    }
}
```

Convert result to JSON

Convert result to XML

Supporting XML and JSON

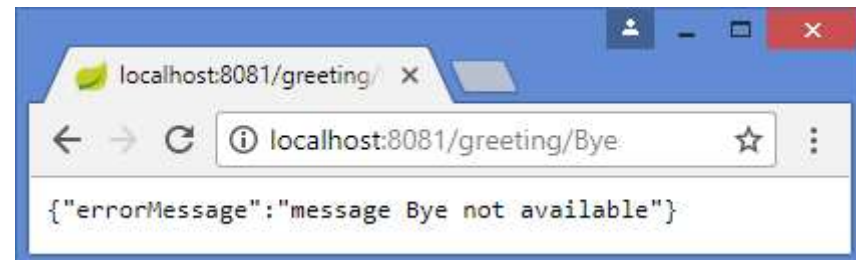
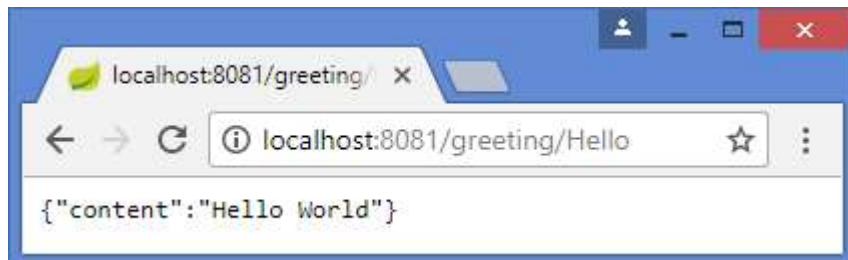


ResponseEntity

```
@RestController
public class GreetingController {

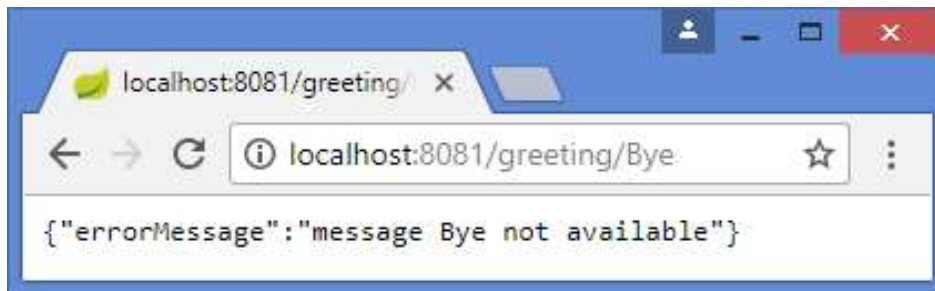
    @RequestMapping("/greeting/{message}")
    public ResponseEntity<?> getGreeting(@PathVariable("message") String message) {
        Greeting greeting = new Greeting("");
        if (message.equals("Hello")) {
            greeting.setContent("Hello World");
        }
        else{
            return new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

Set the content and the HttpStatus

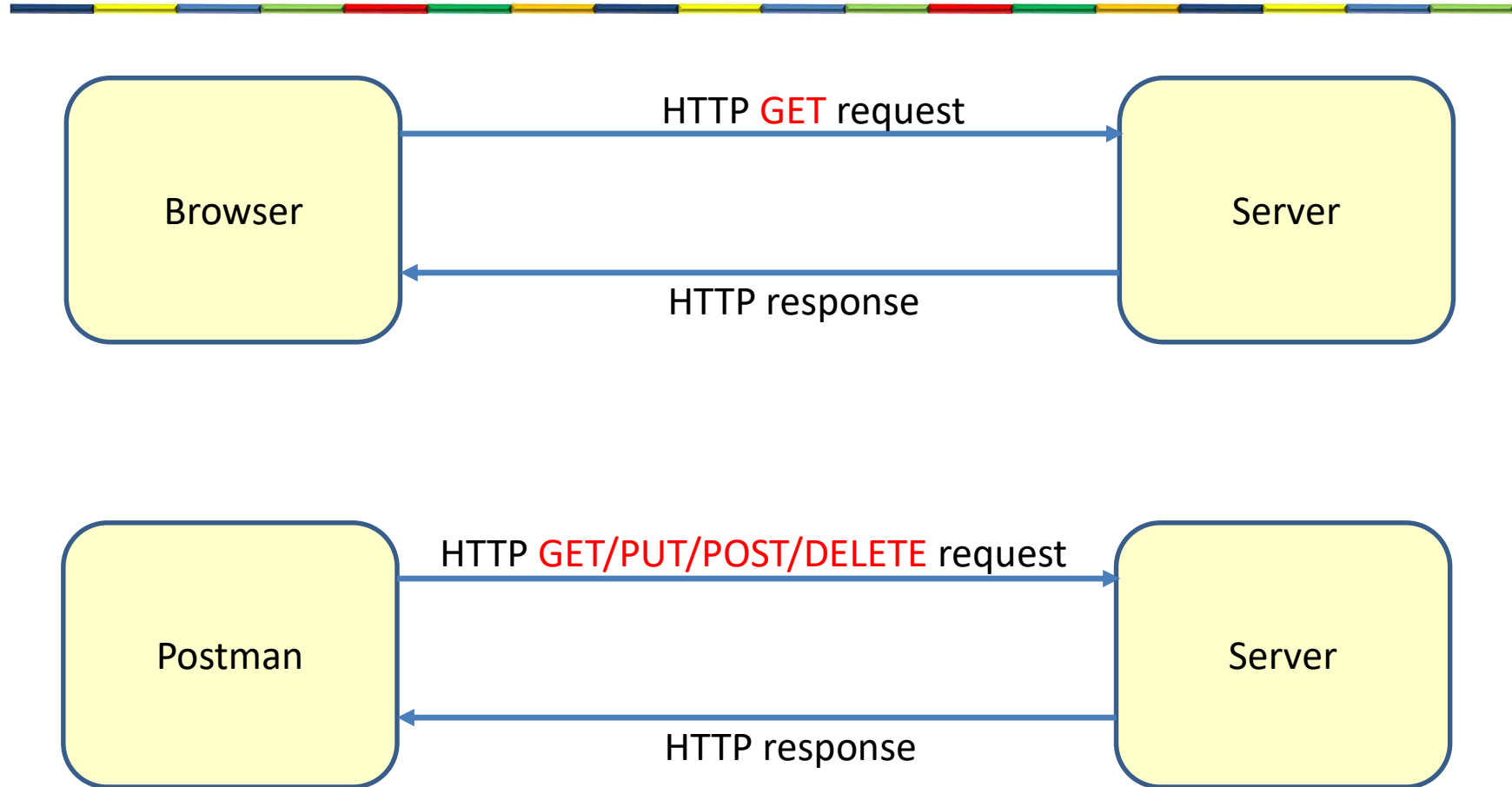


CustomErrorType

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



REST client



REST Client: Postman

Postman interface showing a GET request to `https://swapi.co/api/people/1`. The request is configured with the following details:

- Method: GET
- URL: `https://swapi.co/api/people/1`
- Params: (empty)
- Authorization: No Auth
- Headers: (1)
- Body: (empty)
- Pre-request: (empty)

The response is displayed in the Body tab, showing a JSON object for Luke Skywalker:

```
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": "1988Y",
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": [
```

Postman interface showing a GET request to `https://swapi.co/api/people/5`. The request is configured with the following details:

- Method: GET
- URL: `https://swapi.co/api/people/5`
- Params: (empty)
- Authorization: No Auth
- Headers: (1)
- Body: (empty)
- Pre-request: (empty)

The response is displayed in the Body tab, showing a JSON object for Leia Organa:

```
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": "1988Y",
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;  
    ...  
}
```

@Controller

```
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) {
```

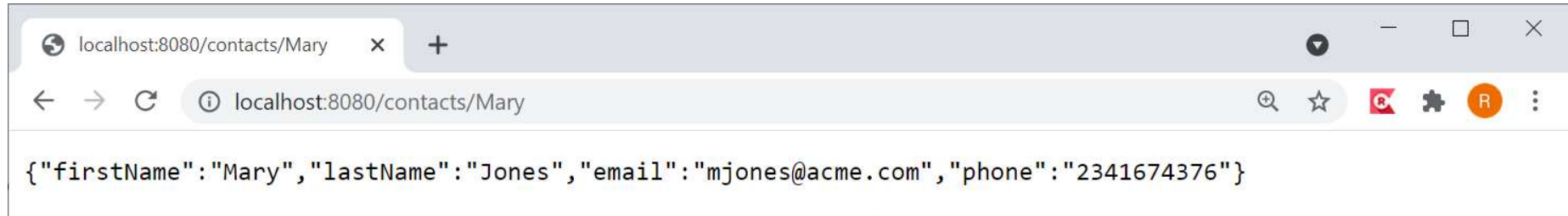
```
            return new ResponseEntity<CustomErrorType>(new CustomErrorType("Contact with firstname= " +  
                firstName + " is not available"), HttpStatus.NOT_FOUND);
```

```
        }
```

```
        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

POST

localhost:8080/contacts/

URL

Body

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

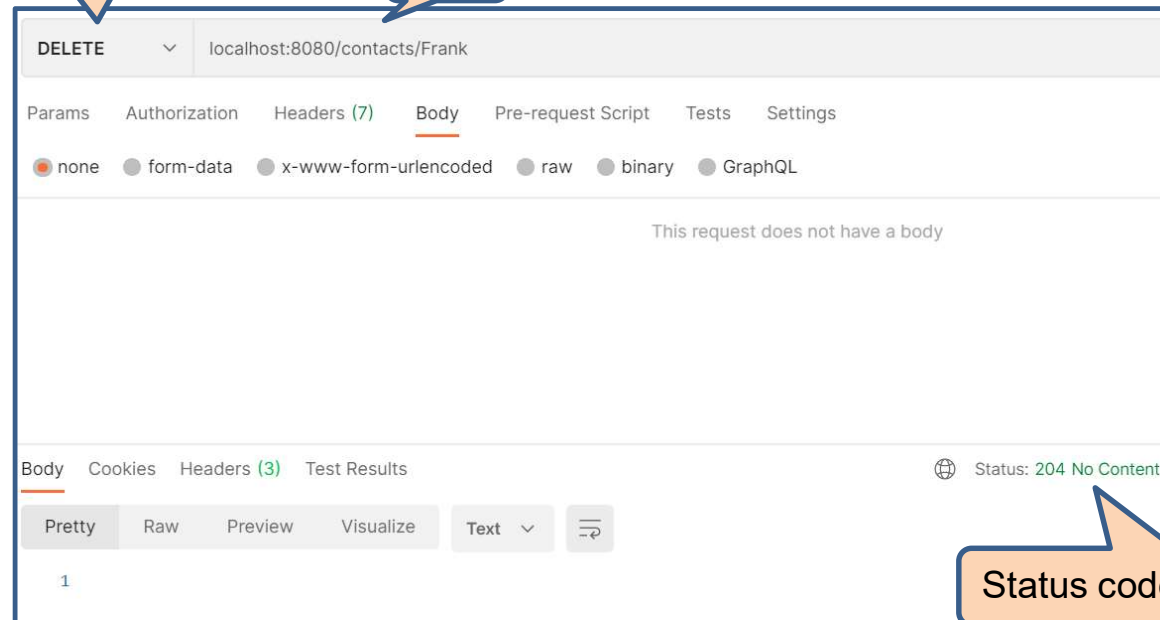
Delete a contact

```
@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

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 Type:** JSON
- Request Body:**

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

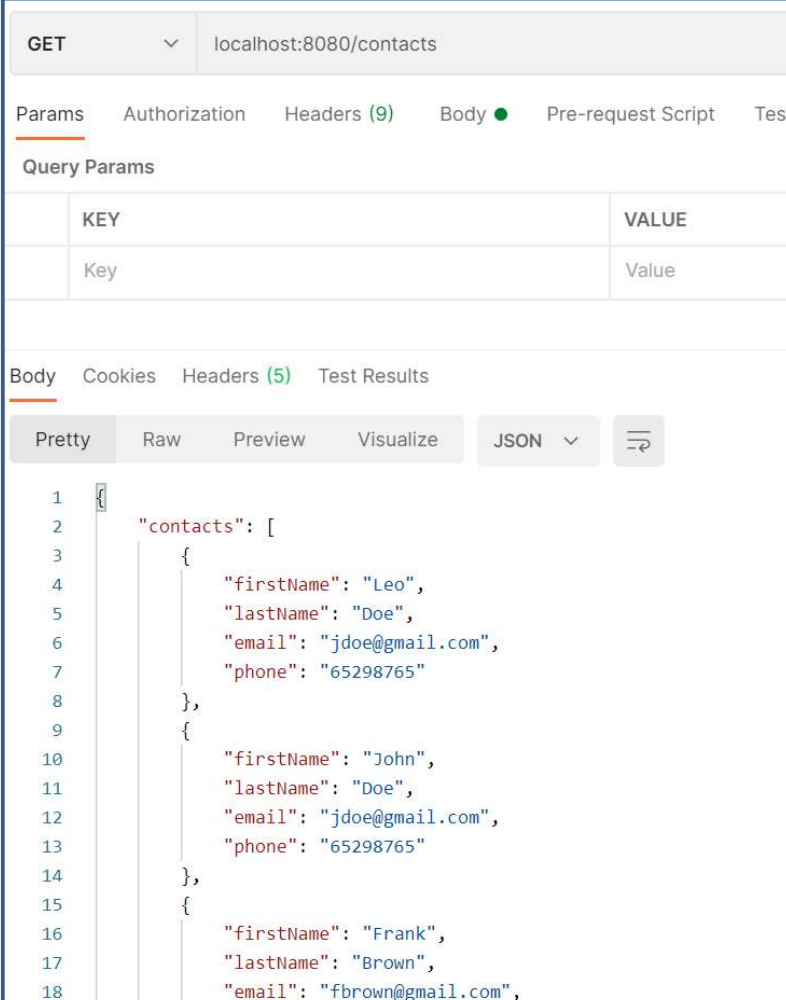
```
{
  "firstName": "Frank",
  "lastName": "Brown",
  "email": "fbrown@gmail.com",
  "phone": "65298765"
}
```

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;

    ...
}
```



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",
```

Main point

- REST makes use of the plain HTTP protocol where HTTP request need to be mapped to server methods. *Every action has a reaction.*

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 query string

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

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

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

TESTING REST

Example of unit testing

```
package count;

public class Counter {
    private int counterValue=0;

    public int increment(){
        return ++counterValue;
    }
    public int decrement(){
        return --counterValue;
    }
    public int getCounterValue() {
        return counterValue;
    }
}
```

Example of unit testing

```
import static org.junit.Assert.*;
import org.junit.*
```

```
public class CounterTest {
    private Counter counter;
```

Initialization

```
@Before
```

```
public void setUp() throws Exception {
    counter = new Counter();
}
```

Test method

```
@Test
```

```
public void testIncrement() {
    assertEquals("Counter.increment does not work correctly", 1, counter.increment());
    assertEquals("Counter.increment does not work correctly", 2, counter.increment());
}
```

Test method

```
@Test
```

```
public void testDecrement() {
    assertEquals("Counter.decrement does not work correctly", -1, counter.decrement());
    assertEquals("Counter.decrement does not work correctly", -2, counter.decrement());
}
```

```
public class Counter {
    private int counterValue=0;

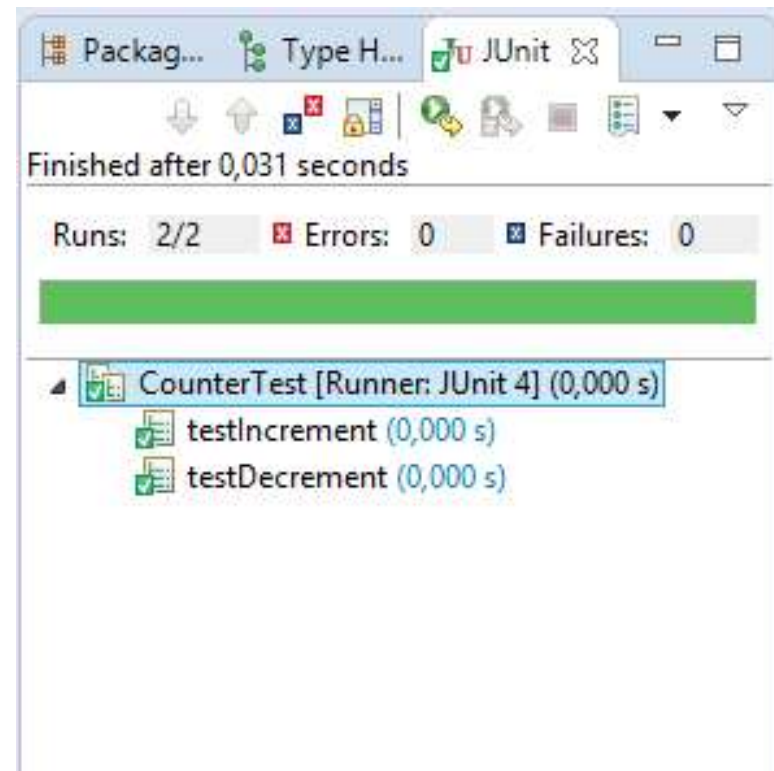
    public int increment(){
        return ++counterValue;
    }
    public int decrement(){
        return --counterValue;
    }
    public int getCounterValue() {
        return counterValue;
    }
}
```

Running the test

```
package count;

public class Counter {
    private int counterValue=0;

    public int increment(){
        return ++counterValue;
    }
    public int decrement(){
        return --counterValue;
    }
    public int getCounterValue() {
        return counterValue;
    }
}
```



Running the test

```
package count;
```

```
public class Counter {  
    private int counterValue=0;
```

```
    public int increment() {  
        return ++counterValue;  
    }
```

```
    public int decrement() {  
        return counterValue;  
    }
```

```
    public int getCounterValue() {  
        return counterValue;  
    }  
}
```

Package Explorer Type Hierarchy JUnit

Finished after 0,032 seconds

Runs: 2/2 Errors: 0 Failures: 1

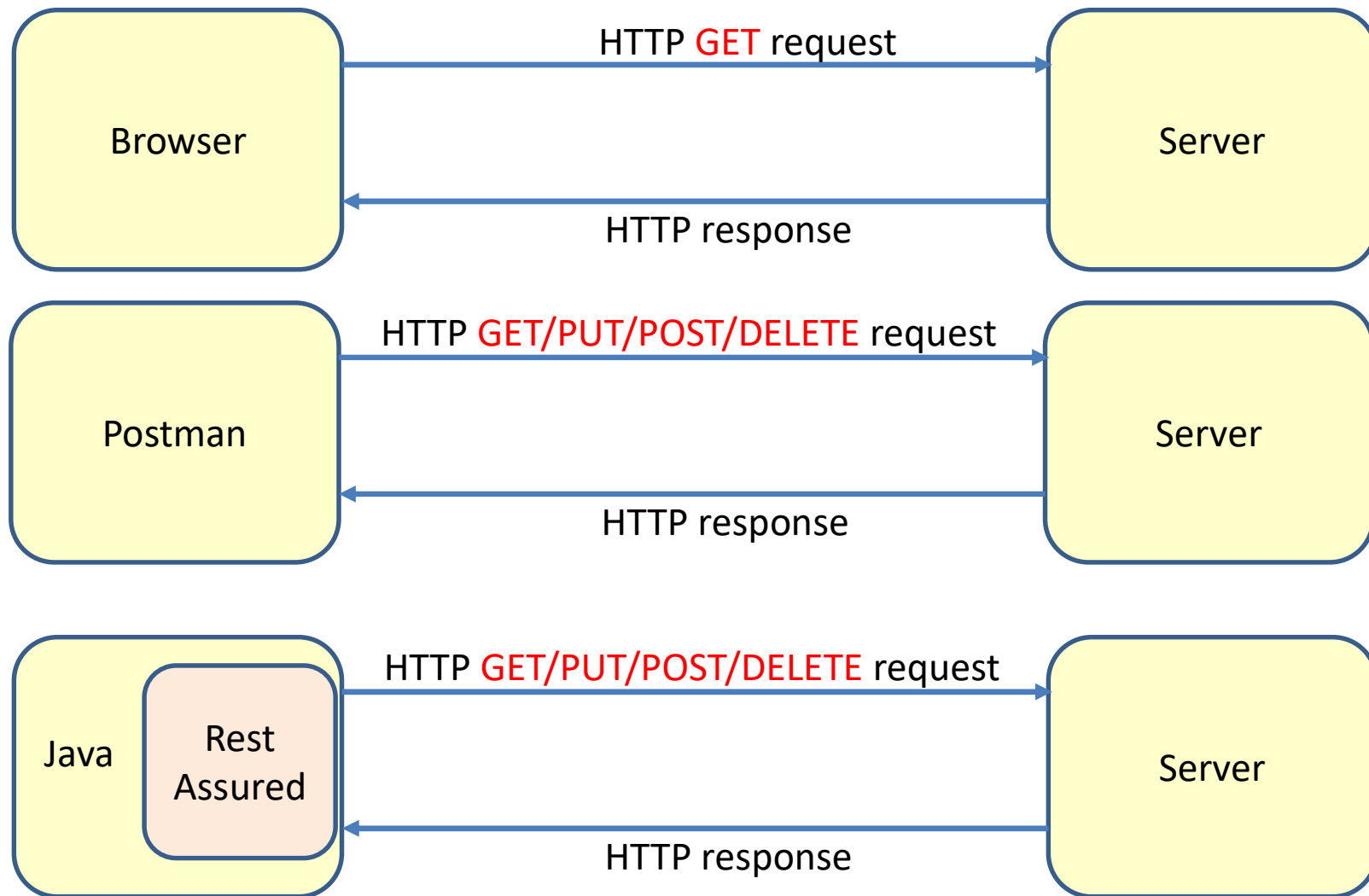
CounterTest [Runner: JUnit 4] (0,000 s)

- testIncrement (0,000 s)
- testDecrement (0,000 s)

Failure Trace

java.lang.AssertionError: Counter.decrement does not work correctly expected: <-1> but was: <0>
at CounterTest.testDecrement(CounterTest.java:21)

REST client



RestAssured example

```
import org.junit.BeforeClass;
import org.junit.Test;
import io.restassured.RestAssured;
import static io.restassured.RestAssured.*;
import static org.hamcrest.Matchers.equalTo;

public class RestTest {

    @BeforeClass
    public static void setup() {
        RestAssured.port = Integer.valueOf(8080);
        RestAssured.baseURI = "http://swapi.co";
        RestAssured.basePath = "/api/people/";
    }

    @Test
    public void test() {
        given()
            .relaxedHTTPSValidation("TLSv1.2")
            .when()
            .get("1")
            .then()
            .body("name", equalTo("Luke Skywalker"));
    }
}
```

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

- URL: `https://swapi.co/api/people/1`
- Method: `GET`
- Params: (empty)
- Authorization: `No Auth`
- Headers: (1 header)
- Body: (empty)
- Pre-request: (empty)
- Response Body: `JSON` (Pretty view)
- Response Headers: (13 headers)
- Test Results: (empty)

The JSON response body is as follows:

```
{
  "name": "Luke Skywalker",
  "height": "172",
  "mass": "77",
  "hair_color": "blond",
  "skin_color": "fair",
  "eye_color": "blue",
  "birth_year": "1988",
  "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/"
  ]
}
```

This means that you'll trust all hosts regardless if the SSL certificate is invalid.

statusCode

```
@Test
public void testStatusLuke() {
    given()
        .relaxedHTTPSValidation("TLSv1.2")
        .when()
        .get("1")
        .then()
        .statusCode(200)
        .body("name", equalTo("Luke Skywalker"));
}
```

```
@Test
public void testStatusLuke() {
    given()
        .relaxedHTTPSValidation("TLSv1.2")
        .when()
        .get("123")
        .then()
        .statusCode(404);
}
```

contentType

```
@Test
public void test() {
    given().relaxedHTTPSValidation("TLSv1.2")
        .when()
        .get("1")
        .then()
        .contentType(ContentType.JSON)
        .and()
        .body("name", equalTo("Luke Skywalker"));
}
```

Data driven test

```
@RunWith(DataProviderRunner.class)
public class RestTest {
    @DataProvider
    public static Object[][] dataProviderStarWars() {
        return new Object[][] {
            {"1", "Luke Skywalker"},
            {"2", "C-3PO"},
            {"3", "R2-D2"}
        };
    }

    @Test
    @UseDataProvider("dataProviderStarWars")
    public void testStarWarsPeople(String key, String name) {
        given().relaxedHTTPSValidation("TLSv1.2")
            .when()
            .get(key)
            .then()
            .contentType(ContentType.JSON)
            .and()
            .body("name", equalTo(name));
    }
}
```

Example REST Bookservice

Request	Response
<p>GET /book/{isbn}</p> <p>Get localhost:8081/book/123</p>	<p>Return book with this isbn</p> <pre>{ "isbn": "123", "title": "Book 1", "price": 20.95, "author": "James Brown" }</pre>
<p>GET /books</p> <p>Get localhost:8081/books</p>	<p>Return all books</p> <pre>[{ "isbn": "123", "title": "Book 1", "price": 20.95, "author": "James Brown" }, { "isbn": "124", "title": "Book 2", "price": 20.95, "author": "Mary Jones" }]</pre>

Example REST Bookservice

Request	Response
<p>DELETE /book/{isbn}</p> <p><i>DELETE localhost:8081/book/123</i></p>	<p>Delete book with this isbn</p>
<p>POST /book</p> <p><i>POST localhost:8081/book</i></p> <pre>{ "isbn": "125", "title": "Book 3", "price": 26.95, "author": "Mary Brown" }</pre>	<p>Add new book</p> <pre>{ "isbn": "125", "title": "Book 3", "price": 26.95, "author": "Mary Brown" }</pre>
<p>PUT /book</p> <p><i>PUT localhost:8081/book</i></p> <pre>{ "isbn": "125", "title": "Book 4", "price": 45.95, "author": "Mary Brown" }</pre>	<p>Update existing book</p> <pre>{ "isbn": "125", "title": "Book 4", "price": 45.95, "author": "Mary Brown" }</pre>

Get one book

```
public class BookTest {  
  
    @BeforeClass  
    public static void setup() {  
        RestAssured.port = Integer.valueOf(8081);  
        RestAssured.baseURI = "http://localhost/";  
        RestAssured.basePath = "";  
    }  
  
    @Test  
    public void testGetOneBook() {  
        given()  
        .when()  
        .get("book/123")  
        .then()  
        .contentType(ContentType.JSON)  
        .and()  
        .body("isbn", equalTo("123"))  
        .body("title", equalTo("Book 1"))  
        .body("price", equalTo(20.95f))  
        .body("author", equalTo("James Brown"));  
    }  
}
```

```
{  
  "isbn": "123",  
  "title": "Book 1",  
  "price": 20.95,  
  "author": "James Brown"  
}
```

Use f for real numbers

Get all books: test isbn

```
@Test
public void testIsbnAllBooks() {
    given()
        .when()
        .get("books")
        .then()
        .contentType(ContentType.JSON)
        .body("isbn", hasItems("123", "124"));
}
```

```
[
  {
    "isbn": "123",
    "title": "Book 1",
    "price": 20.95,
    "author": "James Brown"
  },
  {
    "isbn": "124",
    "title": "Book 2",
    "price": 20.95,
    "author": "Mary Jones"
  }
]
```

Get all books: test number of books

```
@Test
public void testNumberOfAllBooks() {
    given()
        .when()
        .get("books")
        .then()
        .contentType(ContentType.JSON)
        .body("isbn", hasSize(2));
}
```

```
[
  {
    "isbn": "123",
    "title": "Book 1",
    "price": 20.95,
    "author": "James Brown"
  },
  {
    "isbn": "124",
    "title": "Book 2",
    "price": 20.95,
    "author": "Mary Jones"
  }
]
```

Queries

```
@Test
public void testNumberOfAllBooksWithPrice() {
    given()
        .when()
        .get("books")
        .then()
        .contentType(ContentType.JSON)
        .body("findAll{it.price < 23 }", hasSize(2));
}
```

'it' is the iterator

```
@Test
public void testNumberOfAllBooksFromAuthor() {
    given()
        .when()
        .get("books")
        .then()
        .contentType(ContentType.JSON)
        .body("findAll{it.author == 'James Brown' }.title", hasItem("Book 1"));
}
```

```
[
  {
    "isbn": "123",
    "title": "Book 1",
    "price": 20.95,
    "author": "James Brown"
  },
  {
    "isbn": "124",
    "title": "Book 2",
    "price": 20.95,
    "author": "Mary Jones"
  }
]
```

Delete

```
@Test
public void testDelete() {
    // add the to be deleted book
    Book book = new Book("123", "Book 1", 20.95, "James Brown");
    given()
        .contentType("application/json")
        .body(book)
        .when().post("/book").then()
        .statusCode(200);

    given()
        .when()
        .delete("book/123");

    given()
        .when()
        .get("books")
        .then()
        .body("isbn", hasSize(1));
}
```

Add book with isbn "123"

Delete book with isbn "123"

Test the number of books

Post

```
@Test
public void testPost() {
    Book book = new Book("234", "Book 3", 34.75, "Jack Johnson");

    given()
        .contentType("application/json")
        .body(book)
        .when().post("/book").then()
        .statusCode(200);

    given()
        .when()
        .get("books")
        .then()
        .contentType(ContentType.JSON)
        .body("isbn", hasItems("123", "124", "234"));

    //delete the book again
    given()
        .when()
        .delete("book/234");
}
```

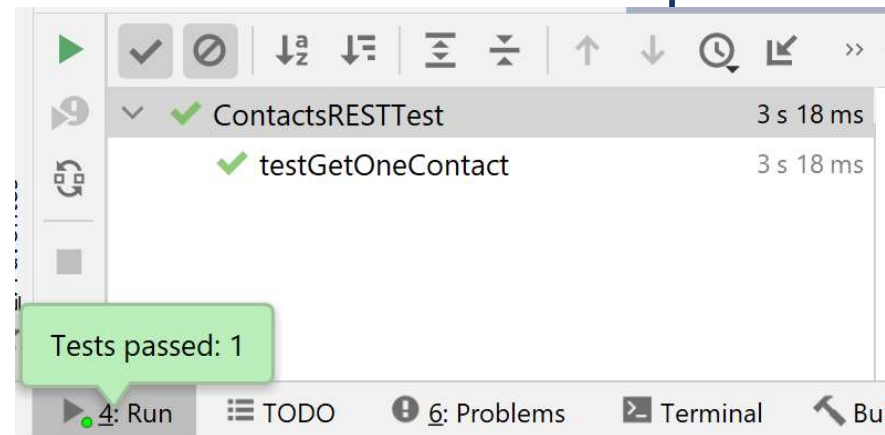
Add book with isbn "234"

Test if the books is added

Delete book with isbn "234"

GET contact

```
public class ContactsRESTTest {  
    @BeforeClass  
    public static void setup() {  
        RestAssured.port = Integer.valueOf(8080);  
        RestAssured.baseURI = "http://localhost";  
        RestAssured.basePath = "";  
    }  
    @Test  
    public void testGetOneContact() {  
        // add the contact to be fetched  
        Contact contact = new Contact("Mary", "Jones", "mjones@acme.com", "2341674376");  
        given()  
            .contentType("application/json")  
            .body(contact)  
            .when().post("/contacts").then()  
            .statusCode(200);  
        // test getting the contact  
        given()  
            .when()  
            .get("contacts/Mary")  
            .then()  
            .contentType(ContentType.JSON)  
            .and()  
            .body("firstName", equalTo("Mary"))  
            .body("lastName", equalTo("Jones"))  
            .body("email", equalTo("mjones@acme.com"))  
            .body("phone", equalTo("2341674376"));  
        //cleanup  
        given()  
            .when()  
            .delete("contacts/Mary");  
    }  
}
```



DELETE contact

@Test

```
public void testDeleteContact() {  
    // add the contact to be deleted book  
    Contact contact = new Contact("Bob", "Smith", "bobby@hotmail.com", "76528765498");  
    given()  
        .contentType("application/json")  
        .body(contact)  
        .when().post("/contacts").then()  
        .statusCode(200);  
  
    given()  
        .when()  
        .delete("contacts/Bob");  
  
    given()  
        .when()  
        .get("contacts/Bob")  
        .then()  
        .statusCode(404)  
        .and()  
        .body("errorMessage", equalTo("Contact with firstname= Bob is not available"));  
}
```

✓	ContactsRESTTest	3 s 719 ms
✓	testGetOneContact	3 s 74 ms
✓	testDeleteContact	645 ms

POST contact

```
@Test
public void testAddContact() {
    // add the contact
    Contact contact = new Contact("Bob", "Smith", "bobby@hotmail.com", "76528765498");
    given()
        .contentType("application/json")
        .body(contact)
        .when().post("/contacts").then()
        .statusCode(200);
    // get the contact and verify
    given()
        .when()
        .get("contacts/Bob")
        .then()
        .statusCode(200)
        .and()
        .body("firstName", equalTo("Bob"))
        .body("lastName", equalTo("Smith"))
        .body("email", equalTo("bobby@hotmail.com"))
        .body("phone", equalTo("76528765498"));
    //cleanup
    given()
        .when()
        .delete("contacts/Bob");
}
```

✓	ContactsRESTTest	4 s 181 ms
✓	testGetOneContact	3 s 378 ms
✓	testDeleteContact	673 ms
✓	testAddContact	130 ms

PUT contact

@Test

```
public void testUpdateContact() {
```

```
    // add the contact
```

```
    Contact contact = new Contact("Bob", "Smith", "bobby@hotmail.com", "76528765498");
```

```
    Contact updateContact = new Contact("Bob", "Johnson", "bobby@gmail.com", "89765123");
```

```
    given()
```

```
        .contentType("application/json")
```

```
        .body(contact)
```

```
        .when().post("/contacts").then()
```

```
        .statusCode(200);
```

```
    //update contact
```

```
    given()
```

```
        .contentType("application/json")
```

```
        .body(updateContact)
```

```
        .when().put("/contacts/"+updateContact.getFirstName()).then()
```

```
        .statusCode(200);
```

```
    // get the contact and verify
```

```
    given()
```

```
        .when()
```

```
        .get("contacts/Bob")
```

```
        .then()
```

```
        .statusCode(200)
```

```
        .and()
```

```
        .body("firstName",equalTo("Bob"))
```

```
        .body("lastName",equalTo("Johnson"))
```

```
        .body("email",equalTo("bobby@gmail.com"))
```

```
        .body("phone",equalTo("89765123"));
```

```
    //cleanup
```

```
    given()
```

```
        .when()
```

```
        .delete("contacts/Bob");
```

```
}
```

✓	ContactsRESTTest	5 s 230 ms
✓	testGetOneContact	4 s 118 ms
✓	testDeleteContact	779 ms
✓	testUpdateContact	183 ms
✓	testAddContact	150 ms

Get all contacts

@Test

```
public void testGetAllContacts() {
```

```
    // add the contacts
```

```
    Contact contact = new Contact("Bob", "Smith", "bobby@hotmail.com", "76528765498");
```

```
    Contact contact2 = new Contact("Tom", "Johnson", "tomjohnson@gmail.com", "543256789");
```

```
    given()
```

```
        .contentType("application/json")
```

```
        .body(contact)
```

```
        .when().post("/contacts").then()
```

```
        .statusCode(200);
```

```
    given()
```

```
        .contentType("application/json")
```

```
        .body(contact2)
```

```
        .when().post("/contacts").then()
```

```
        .statusCode(200);
```

```
    // get all contacts and verify
```

```
    given()
```

```
        .when()
```

```
        .get("contacts")
```

```
        .then()
```

```
        .statusCode(200)
```

```
        .and()
```

```
        .body("contacts.firstName", hasItems("Bob", "Tom"))
```

```
        .body("contacts.lastName", hasItems("Smith", "Johnson"))
```

```
        .body("contacts.email", hasItems("bobby@hotmail.com", "tomjohnson@gmail.com"))
```

```
        .body("contacts.phone", hasItems("76528765498", "543256789"));
```

```
    //cleanup
```

```
    given()
```

```
        .when()
```

```
        .delete("contacts/Bob");
```

```
    given()
```

```
        .when()
```

```
        .delete("contacts/Tom");
```

✓	ContactsRESTTest	4 s 572 ms
✓	testGetOneContact	3 s 298 ms
✓	testDeleteContact	698 ms
✓	testUpdateContact	173 ms
✓	testGetAllContacts	214 ms
✓	testAddContact	189 ms

Main point

- RestAssured is a library used for testing REST API's. *Harmony exists in diversity.*

Connecting the parts of knowledge with the wholeness of knowledge

1. REST webservices creates a web API so that functionality is accessible over HTTP.
2. RestAssured makes it easy to test REST webservices

-
3. **Transcendental consciousness** is the field of all knowledge.
 4. **Wholeness moving within itself:** In Unity Consciousness, all of the intelligence and structure at the basis of the universe is realized as the lively qualities of one's own inner intelligence.

