/ JAVA DB SCHOOL

RESTful Web Services

/ REST Overview

REST

- **RE**presentational **S**tate **T**ransfer
- *Defined* in 2000 by Roy Fielding in his doctoral dissertation
- Software architectural style
- Created to guide the design and the architecture of the World Wide Web
- Defines a set of constraints for how the architecture of an Internet-scale distributed hypermedia system, such as the Web, *should* behave
- Widely accepted set of guidelines for creating stateless, reliable web services
- Platform and programming language independent

What is REST About

- Scalability of interactions between components
- Uniform interfaces
- Independent deployment of components
- Layered architecture
- Facilitate caching to reduce user latency
- Enforce security
- Encapsulate legacy systems

RESTful Web Services

- Any web service that *obeys* the REST constraints is informally described as RESTful
- Must provide textual representations of its **resources**
- Allow resources to be read and modified with a stateless protocol and a predefined set of *operations*
- Requests made to a resource's URI (Uniform Resource Identifier) usually provide a response formatted as JSON, XML, HTML or even text
- GET **http://example.com/customer/14**
- PUT **http://example.com/customer/14**
- DELETE **http://example.com/customer/14**

REST Architecture

- The REST architecture relies on HTTP requests sent to a web server using an **URL** and a specific HTTP **method**; the method should define one of the following behaviour:
- GET – Gets a representation of a target resource
- POST – Let the target resource process the representation enclosed in the request
- PUT – Create or replace the state of the target resource with the state defined by the representation enclosed in the request
- DELETE – Delete the target resource’s state
- Parameters for the methods can be sent either as query strings (in the URL) or inside the body of the request, but identifiers of resources must be part of the URI!

Comparison with CRUD

- CRUD operations should be defined by the following methods:
  - GET method = read
  - PUT method = create and update
  - DELETE method = delete

- The POST method is not a CRUD operation but a process operation that has target-resource-specific semantics excluding storage management semantics

Properties of REST Methods

- The GET, PUT, and DELETE methods are *idempotent* = applying them multiple times to a resource results in the same state change of the resource as applying them once

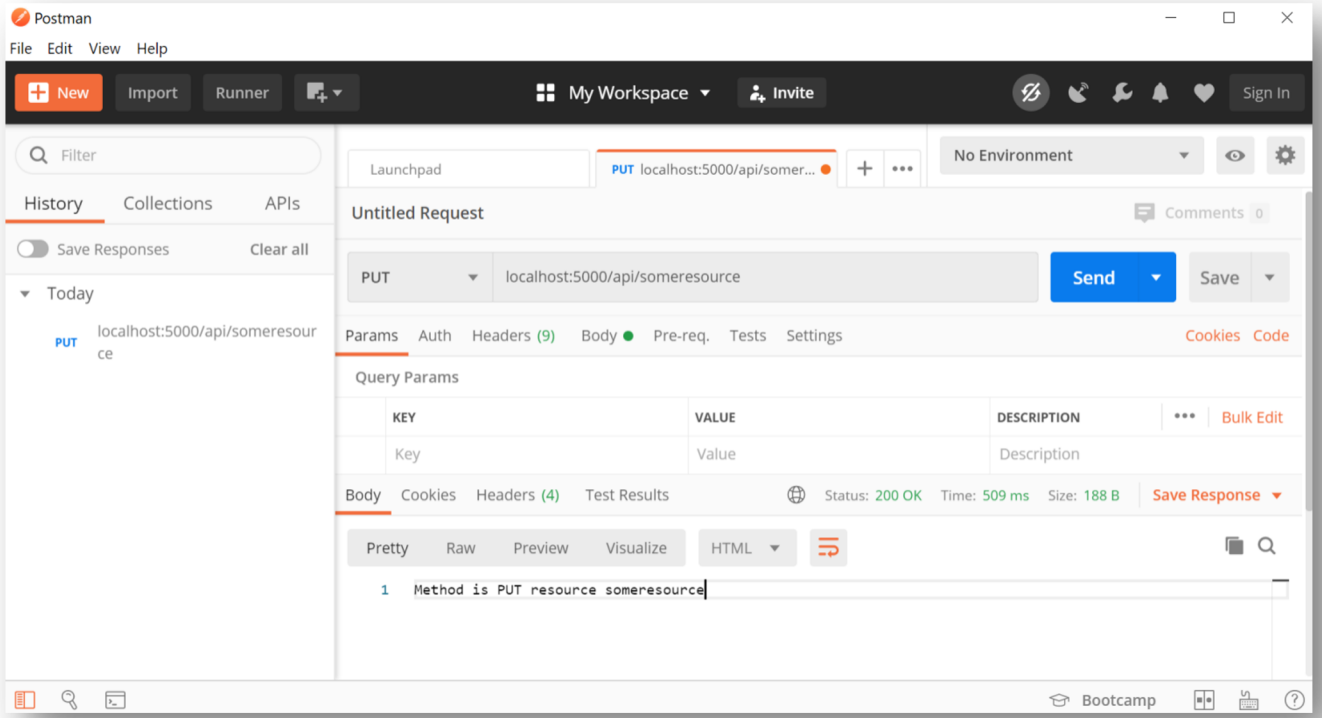
- The GET and POST methods are *cacheable* = responses to them are allowed to be stored (usually by browsers) for future reuse; no request will be sent when a resource is cached

REST Architectural constraints

- Client–server architecture
- Statelessness
- Cacheability
- Layered system
- Code on demand (optional)
- Uniform interface

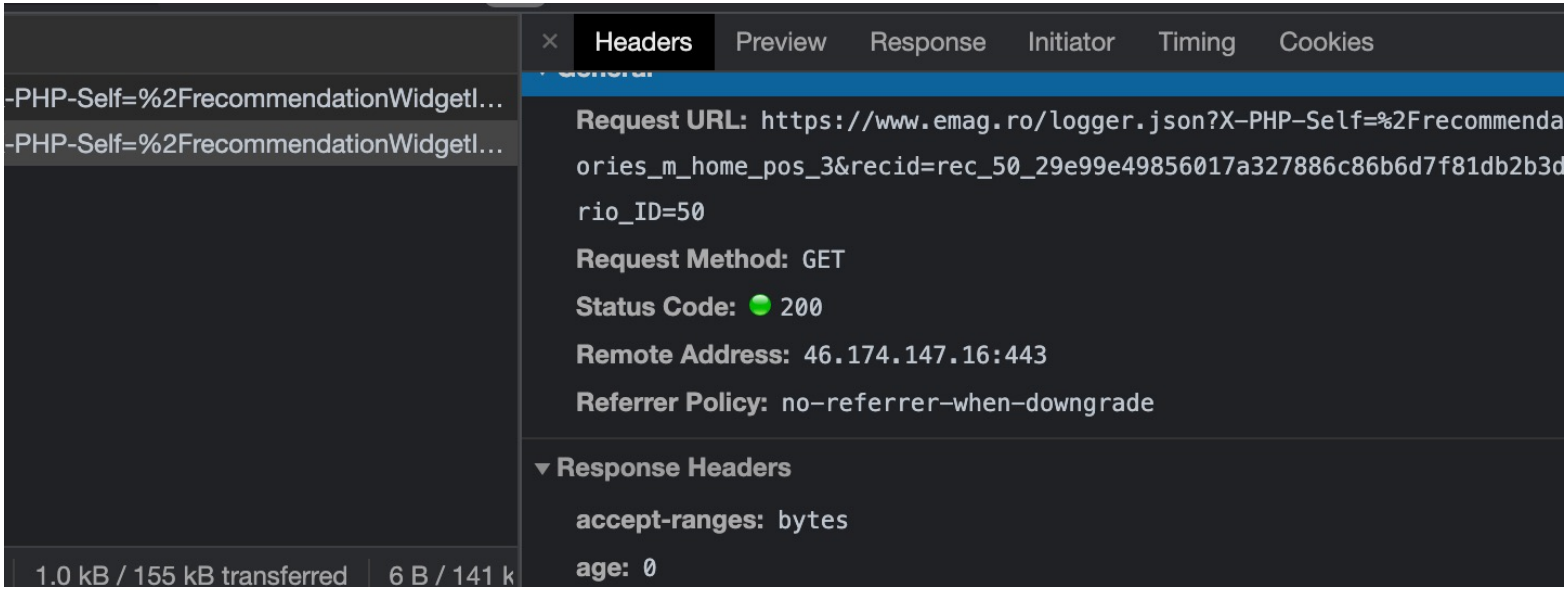
Postman

- Software application used for testing HTTP endpoints (RESTful web services, too)



Browser Console

- Used also for testing HTTP requests (RESTful web services, too)
- Pre-installed in most browsers



/ Java Spark

Java Spark

- Micro framework for creating web applications in Java 8 with minimal effort
- One of the many framework for developing REST web applications for Java

```
import static spark.Spark.*;

public class HelloWorld {
    public static void main(String[] args) {
        get("/hello", (req, res) -> "Hello World");
    }
}
```

- Wait, but where is it located? Of course, you need to install Spark jar or download it using Maven!

Java Spark, not Apache Spark!

- Beware: different package than Apache Spark!
- Maven import:

```
<dependency>
	<groupId>com.sparkjava</groupId>
	<artifactId>spark-core</artifactId>
	<version>2.9.3</version>
</dependency>
```

- https://mvnrepository.com/artifact/com.sparkjava/spark-core

Hello, Spark!

- *Hello, world!* with Spark

```
import static spark.Spark.*;

public class HelloWorldService {
    public static void main(String[] args) {

        get("/hello", (req, res)->"Hello, world!");

        get("/hello/:name", (req,res)->{
            return "Hello, "+ req.params(":name") + '!';
        });
    }
}
```

- Static imports: https://en.wikipedia.org/wiki/Static_import

- Java Spark documentation: https://sparkjava.com/documentation

Ooops… Spark

- Ooops… SLF4J Error

```
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
```

- Bring those damn SLF4J dependencies:

```
<dependency>
	<groupId>org.slf4j</groupId>
	<artifactId>slf4j-api</artifactId>
	<version>1.7.5</version>
</dependency>
<dependency>
	<groupId>org.slf4j</groupId>
	<artifactId>slf4j-log4j12</artifactId>
	<version>1.7.5</version>
</dependency>
```

Enjoy Spark

- Now you can enjoy Spark by accessing those two implemented endpoints:
- http://localhost:4567/hello
- http://localhost:4567/hello/Vasile
- 4567 is the default Java Spark port
- The server will stay up and running (forever waiting for your requests)

Bye bye, Spark!

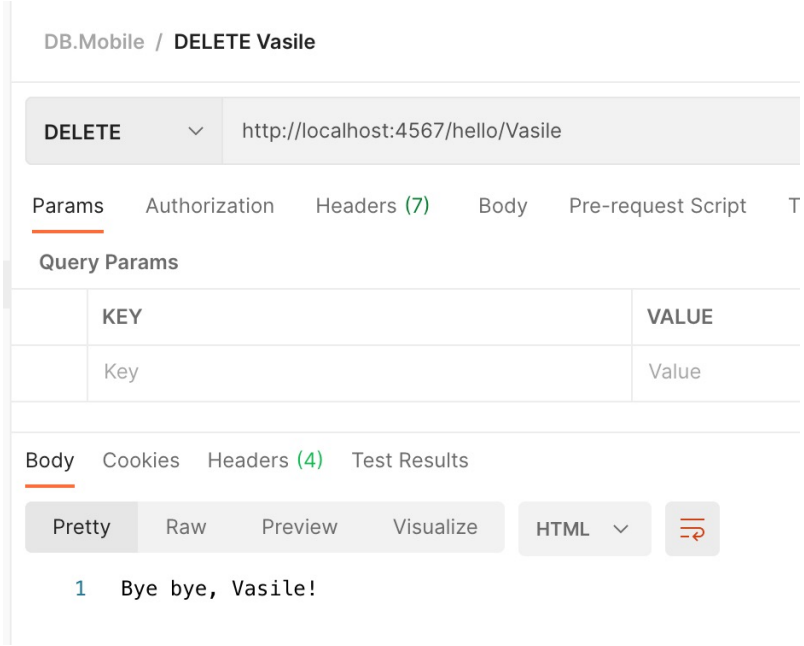
- Few more endpoints added (triggered by PUT and DELETE HTTP methods):

```
import spark.Spark; // notice you can import Spark without specifying static; in this case, you have to use
// the name of the class when calling static methods

public class HelloWorldService {
    public static void main(String[] args) {

        Spark.put("/hello/:name", (req,res)->{
            return req.params(":name") + " is being created or gets an upgrade!";
        });

        Spark.delete("/hello/:name", (req,res)->{
            return "Bye bye, "+ req.params(":name") + '!';
        });
    }
}
```

- How do we call those endpoints???

Postman for the win!

- Send Postman requests to those endpoints



Alternatives to trigger DELETE and PUT requests

- You developed the server. Now you need a client!
- Call endpoints using a JavaScript library and create a script in a web page
- Use cURL in a terminal to send HTTP requests with these methods
- Use another programming language and a dedicated library that is able to send HTTP requests

/ Practice, practice, practice

Shopify Gets Exposed

Starting from the functions created in the previous classes, implement a RESTful web service over the app and create corresponding endpoints for the following entities:

- Customers (GET customer by id, *PUT customer (both create and update)*, DELETE customer by id, GET all customers)
- Orders (GET order by id, PUT order (both create and update – modify status), DELETE order, GET all orders)
- Add filters for orders (GET all orders of a specific customer by id – query params)

```
GET /orders?customer_id=3
```

- OrderDetails (GET order products by order id – JSON) (Optional; *orderdetails* tables)

/ Q&A


OBILE/ACADEMY