# Template

DEMO

# Next Chapter 123

# LESSON 28

# Spring REST API

# Agenda

1. Create REST APIs / Web Services with Spring
2. Discuss REST concepts, JSON and HTTP messaging
3. Install REST client tool – Postman
4. Develop REST APIs / Web Services with @RestController
5. Build a CRUD interface to the database with Spring REST
6. Build a Web app to call REST APIs using Spring RestTemplate
7. Add Spring Secuity to Spring REST APIs

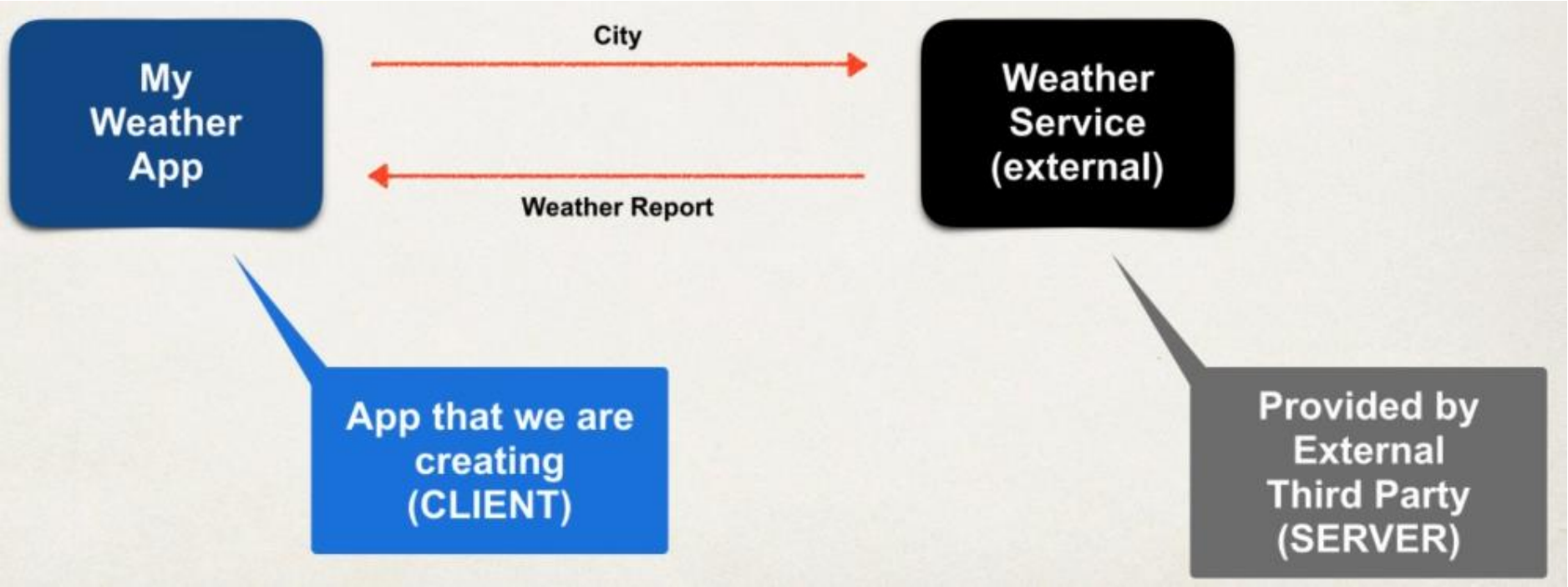
# Reference

- https://spring.io/guides/gs/rest-service/
- https://spring.io/guides/gs/consuming-rest/

# Business Problem

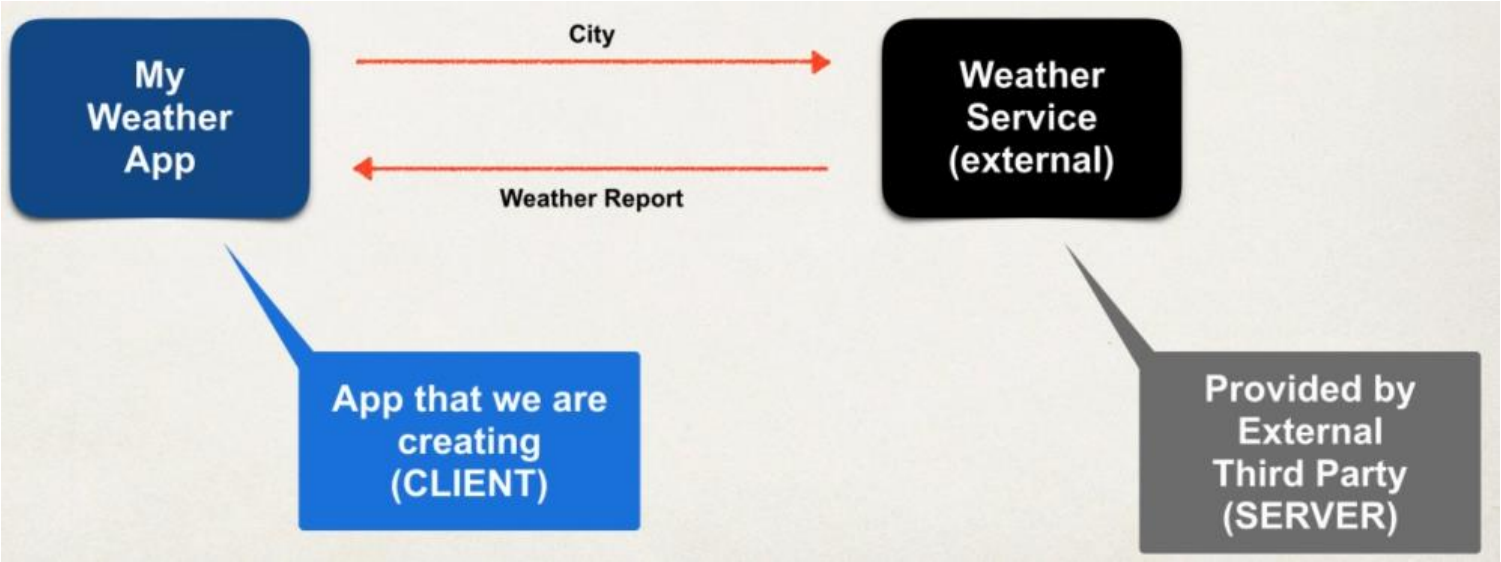
- Build a client app that provides the weather report for a city
- Need to get weather data from external services

# Application Architecture



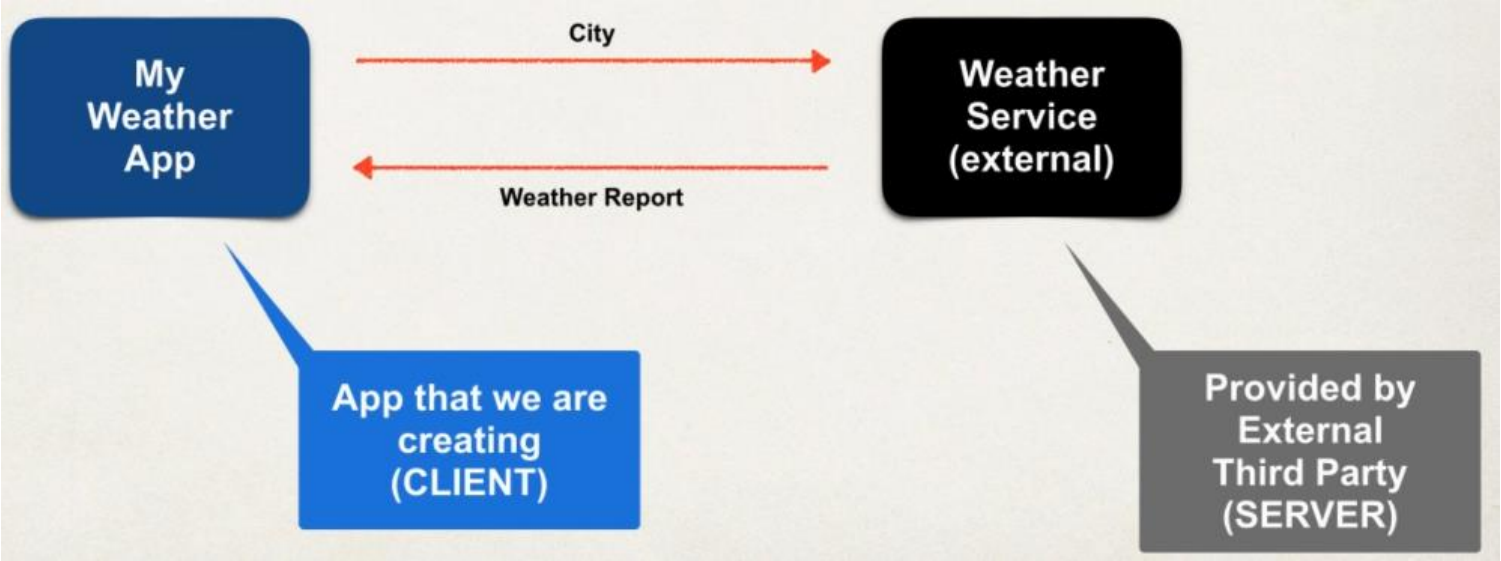
# Questions

- How will we connect to the Weather Service ?
- What programing language do we use ?
- How to get data from Weather Service ? What is the data format ?



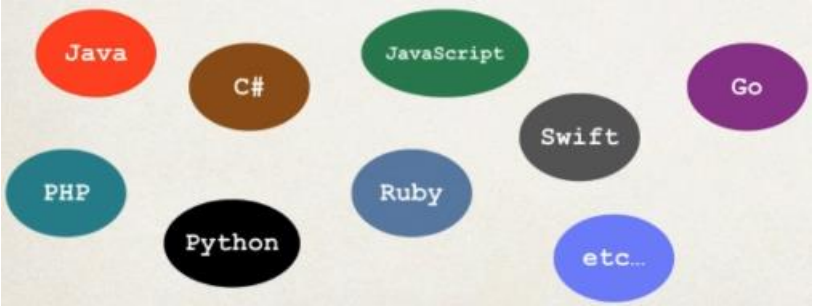
# Answers

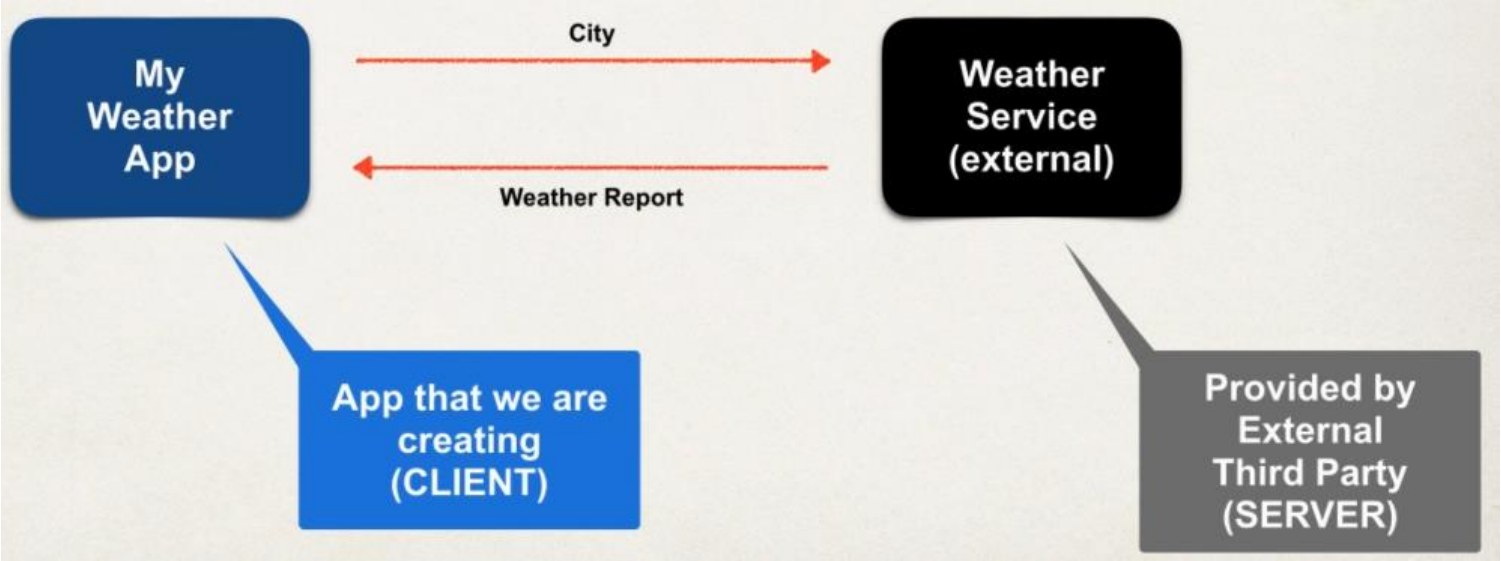
- How will we connect to the Weather Service ?
  - We can make **REST API calls over HTTP**
  - REST: **Re**presentational **S**tate **T**ransfer
  - Lightweight approach for communicating between applcations



# Answers

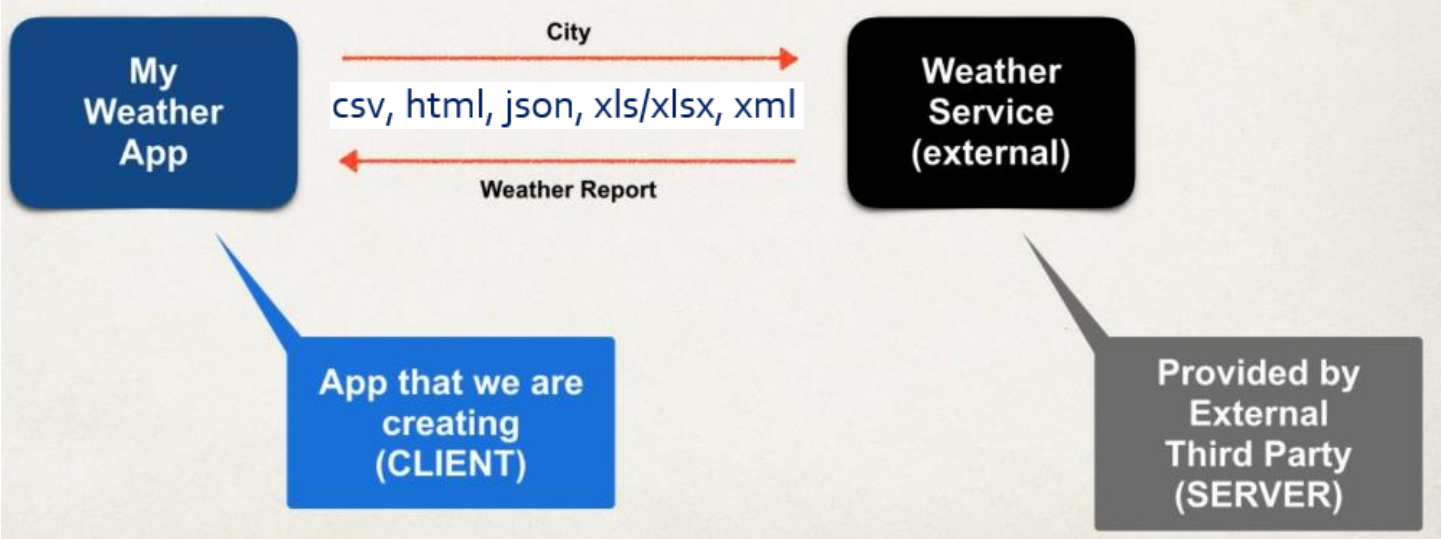
- What programing language do we use ?
  - REST is language independent
  - The **client** app can use **ANY** programing language
  - The **server** app can use **ANY** programing language



# Answers

- How to get data from Weather Service ? What is the data format ?
  - REST application can return and use any data format: csv, html, json, xls/xlsx, xml
  - Commonly use JSON and XML
  - JSON is most popular, modern and best for programing
  - JSON: **J**ava**S**cript **O**bject **N**otation



# Solution

- Use free Weather Service provided by: https://openweathermap.org

Dec 18, 03:42am

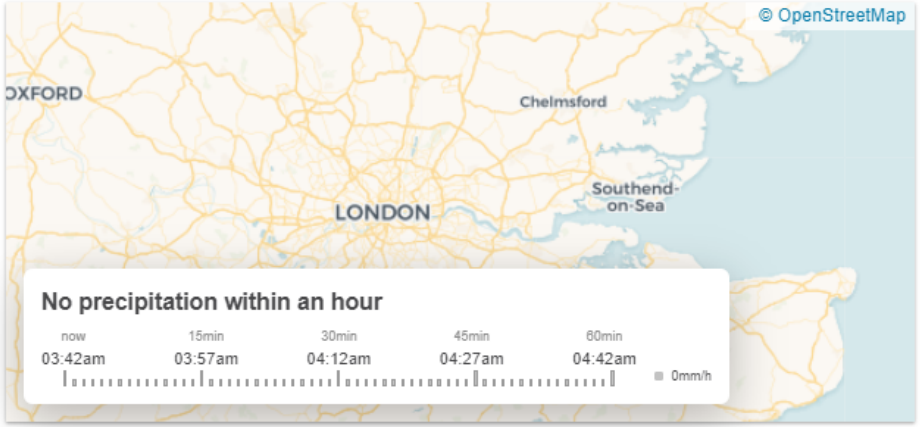
**London, GB**

8°C

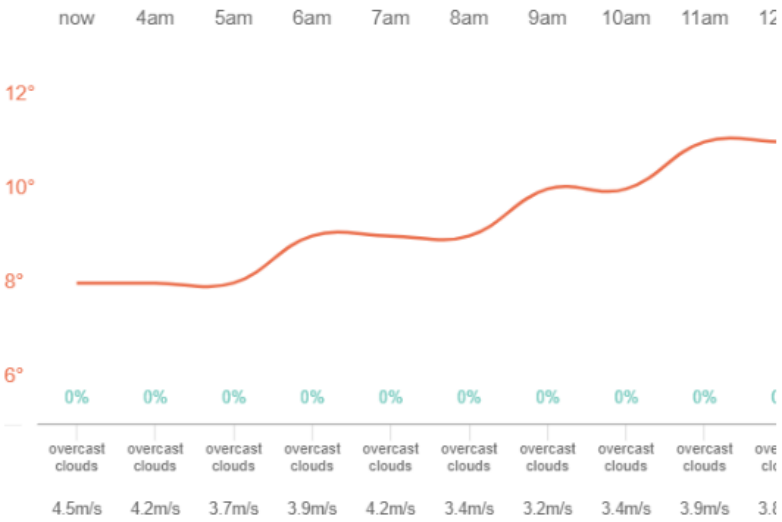
Yellow fog warning

**Feels like 7°C. Overcast clouds. Light air**

1.3m/s NE  1038hPa
Humidity: 94%  Dew point: 7°C
Visibility: 7.0km




**Hourly forecast**




**8-day forecast**

| | | |
|---|---|---|
| Sat, Dec 18 | 11 / 8°C | overcast clouds |
| Sun, Dec 19 | 10 / 6°C | broken clouds |
| Mon, Dec 20 | 8 / 5°C | few clouds |
| Tue, Dec 21 | 6 / 3°C | clear sky |
| Wed, Dec 22 | 4 / 1°C | broken clouds |
| Thu, Dec 23 | 5 / 2°C | overcast clouds |
| Fri, Dec 24 | 11 / 3°C | light rain |
| Sat, Dec 25 | 10 / 8°C | moderate rain |

# Solution

- Use free Weather Service provided by: https://openweathermap.org/current

**API call**

```
api.openweathermap.org/data/2.5/weather?q={city name}&appid=
{API key}
```

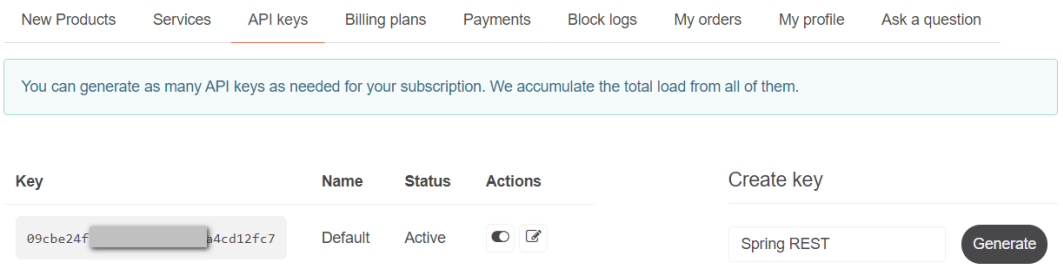
```
api.openweathermap.org/data/2.5/weather?q={city name},{state
code}&appid={API key}
```

```
api.openweathermap.org/data/2.5/weather?q={city name},{state
code},{country code}&appid={API key}
```

**Parameters**

| | | |
|---|---|---|
| q | required | City name, state code and country code divided by comma, Please, refer to ISO 3166 for the state codes or country codes. You can specify the parameter not only in English. In this case, the API response should be returned in the same language as the language of requested location name if the location is in our predefined list of more than 200,000 locations. |
| appid | required | Your unique API key (you can always find it on your account page under the "API key" tab) |

https://openweathermap.org/api_keys



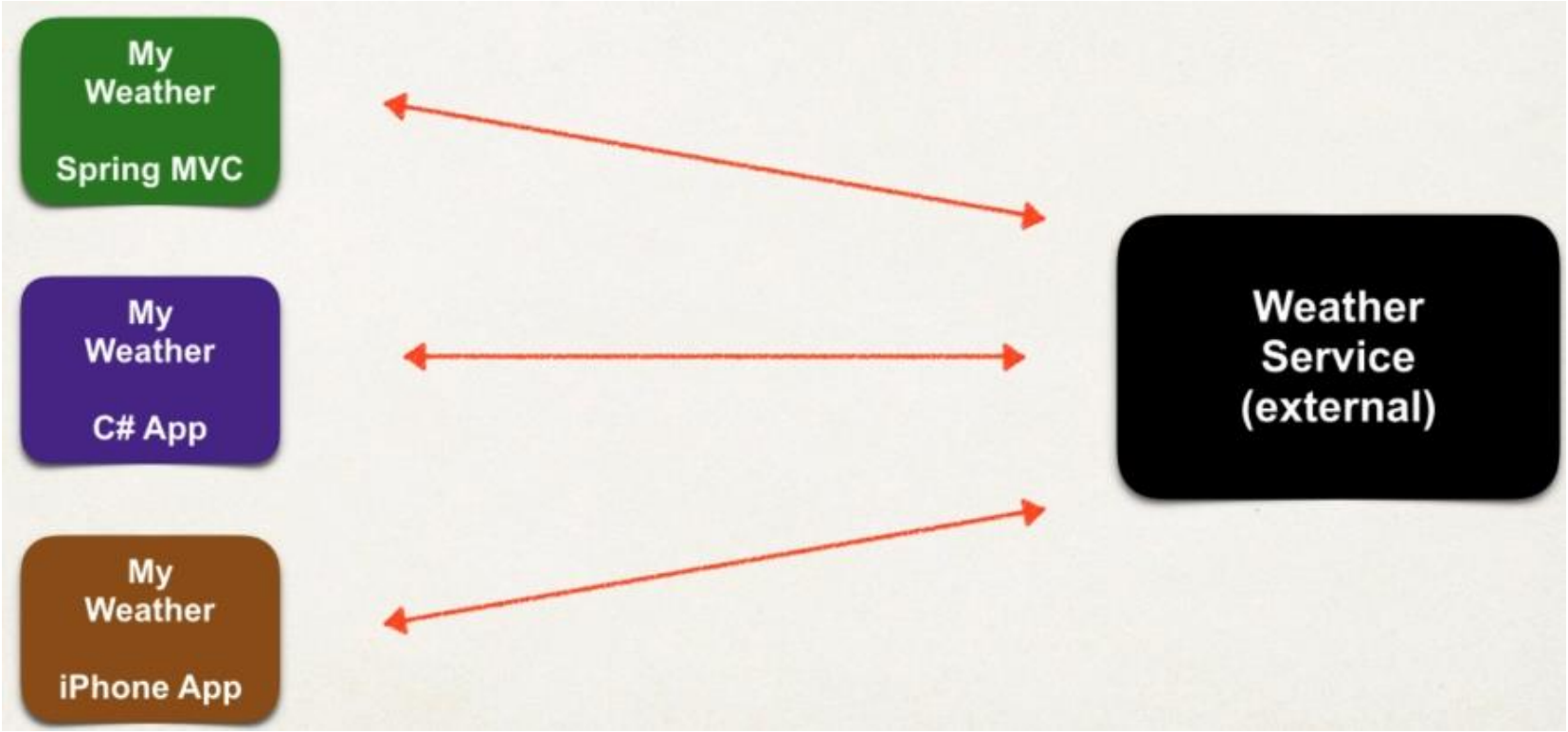
# Solution

- https://api.openweathermap.org/data/2.5/weather?q=Danang& bbb7b4b5a4cd12fc7
- https://chrome.google.com/webstore/detail/json-formatter/bcji mgkkhgoa/related?hl=en
- https://addons.mozilla.org/vi/firefox/addon/json-formatter/
- https://en.wikipedia.org/wiki/List_of_cities_in_Vietnam
- Weather Service responds with JSON format


```
{
  "coord": {
      "lon": 108.2208,
      "lat": 16.0678
  },
  "weather": [ … ], // 1 item
  "base": "stations",
  "main": {
      "temp": 297.14,
      "feels_like": 297.5,
      "temp_min": 297.14,
      "temp_max": 297.18,
      "pressure": 1019,
      "humidity": 73
  },
  "visibility": 8000,
  "wind": { … }, // 2 items
  "clouds": { … }, // 1 item
  "dt": 1639799276,
  "sys": {
      "type": 1,
      "id": 9306,
      "country": "VN",
      "sunrise": 1639782508,
      "sunset": 1639822729
  },
  "timezone": 25200,
  "id": 1583992,
  "name": "Turan",
  "cod": 200
}
```

# Multiple Client Apps

**Remember:**
**REST calls can be made over HTTP**
**REST is language independent**



# How we understand REST

Reference: https://www.redhat.com/en/topics/api/what-is-a-rest-api

## Overview

A REST API (also known as RESTful API) is an application programming interface (API or web API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services. REST stands for representational state transfer and was created by computer scientist Roy Fielding.

REST API (còn được gọi là RESTful API) là một giao diện lập trình ứng dụng (API hoặc web API) tuân theo các ràng buộc của phong cách kiến trúc REST và cho phép tương tác với các dịch vụ web RESTful. REST là viết tắt của chuyển trạng thái biểu diễn và được tạo ra bởi nhà khoa học máy tính Roy Fielding.

We can call make REST API calls over HTTP request

# How do we CALL it

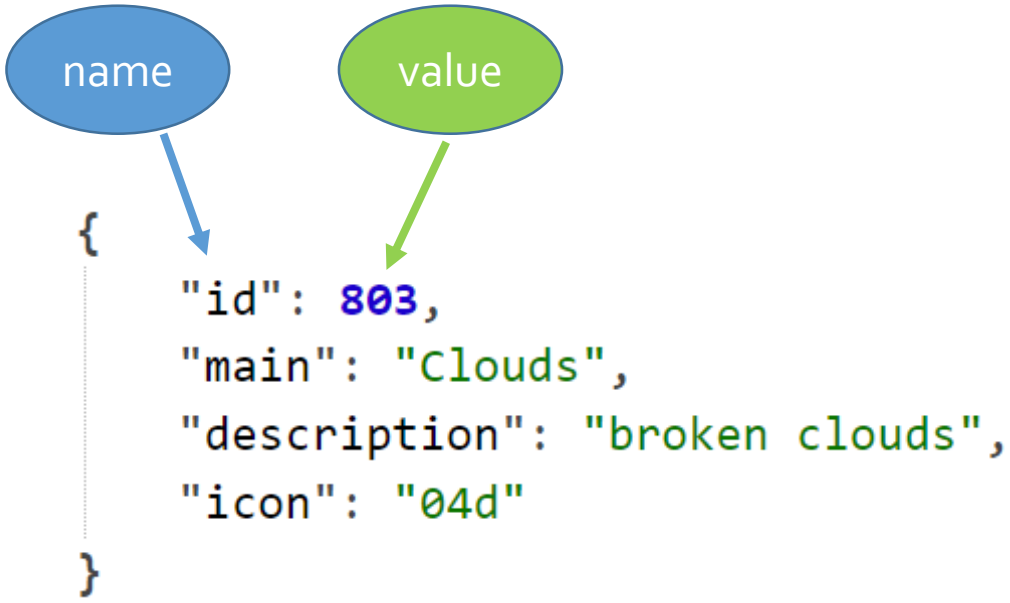
# Spring REST- JSON Data Binding

# What is JSON

- JavaScript Object Notation
- Lightwight data format for storing and exchainging data
- Language independent … not just for JavaScript
- Can use with ANY programing language
- **JSON is just plain text data**

# Simple JSON

- Curley braces define objects in JSON
- Objects members are **name / value** pairs
- Delimeted by colons
- Name is always in double-quotes



# JSON value

- Numbers: no quotes
- String: in double quotes
- Boolean: true false
- Array
- Nested JSON object
- null

Nested object

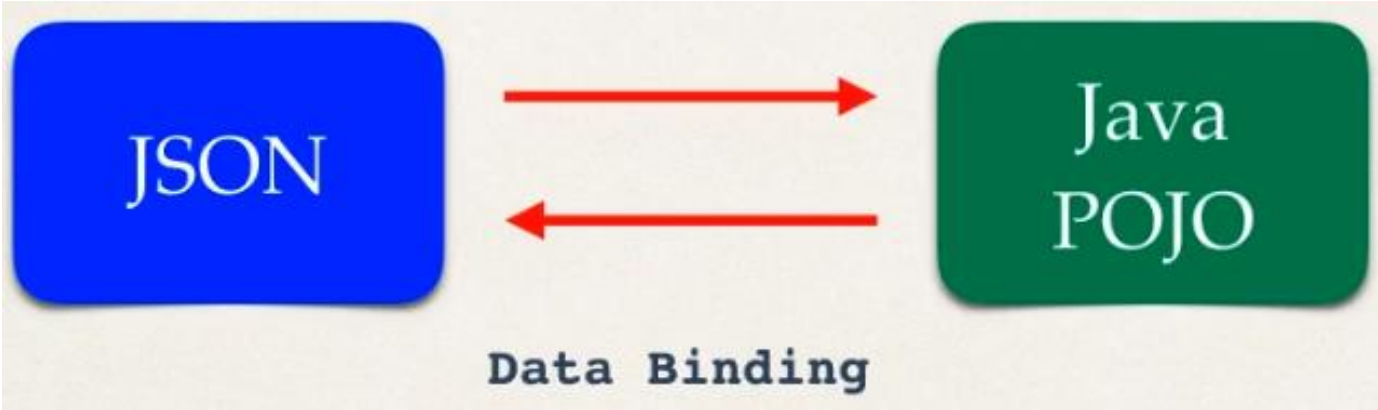
```
{
  "id": 14,
  "firstName": "Mario",
  "lastName": "Rossi",
  "active": true,
  "address" : {
                "street" : "100 Main St",
                "city" : "Philadelphia",
                "state" : "Pennsylvania",
                "zip" : "19103",
                "country" : "USA"
              }
}
```

Array

```
{
  "id": 14,
  "firstName": "Mario",
  "lastName": "Rossi",
  "active": true,
  "languages" : ["Java", "C#", "Python", "Javascript"]
}
```

# Java-JSON data binding

- Data binding is the process of converting **JSON** data to a **Java** POJO(plain old object)




*Also known as*

Mapping

Serialization / Deserialization

Marshalling / Unmarshalling

# JSON data binding with Jackson

- Spring uses the **Jackson Project** behind the scenes
- Jackson handles data binding between JSON and Java POJO
- https://github.com/FasterXML/jackson-databind
- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind

# Jackson data binding

- Jackson data binding API
  - Package: com.fasterxml.jackson.databind

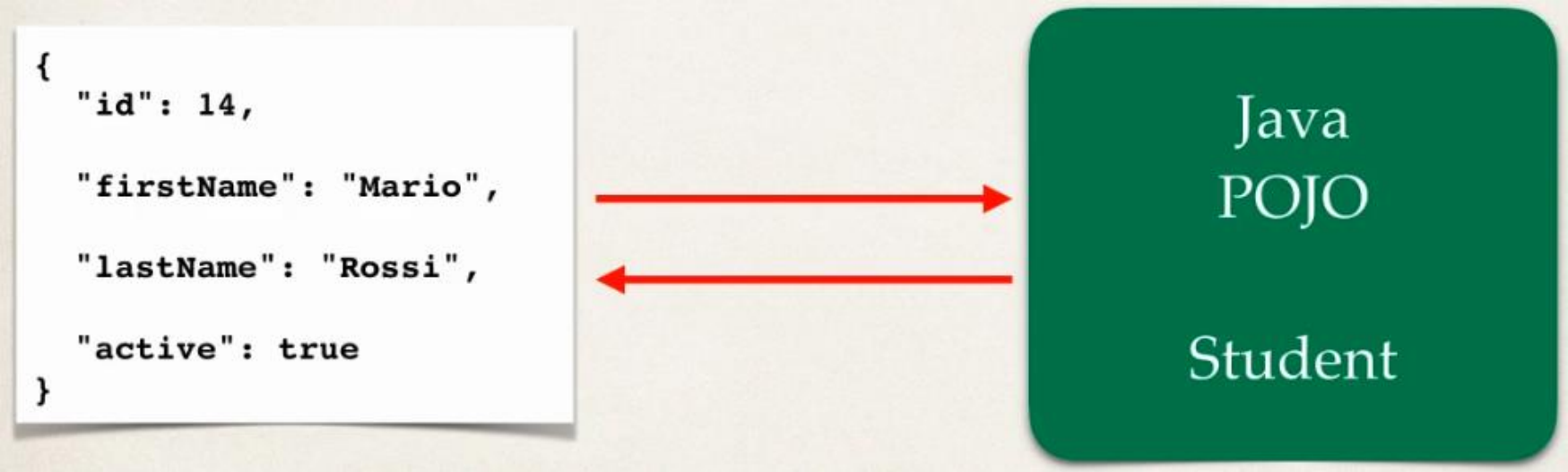
- Maven Dependency

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.12.3</version>
</dependency>
```

- Jackson supports both XML and JSON

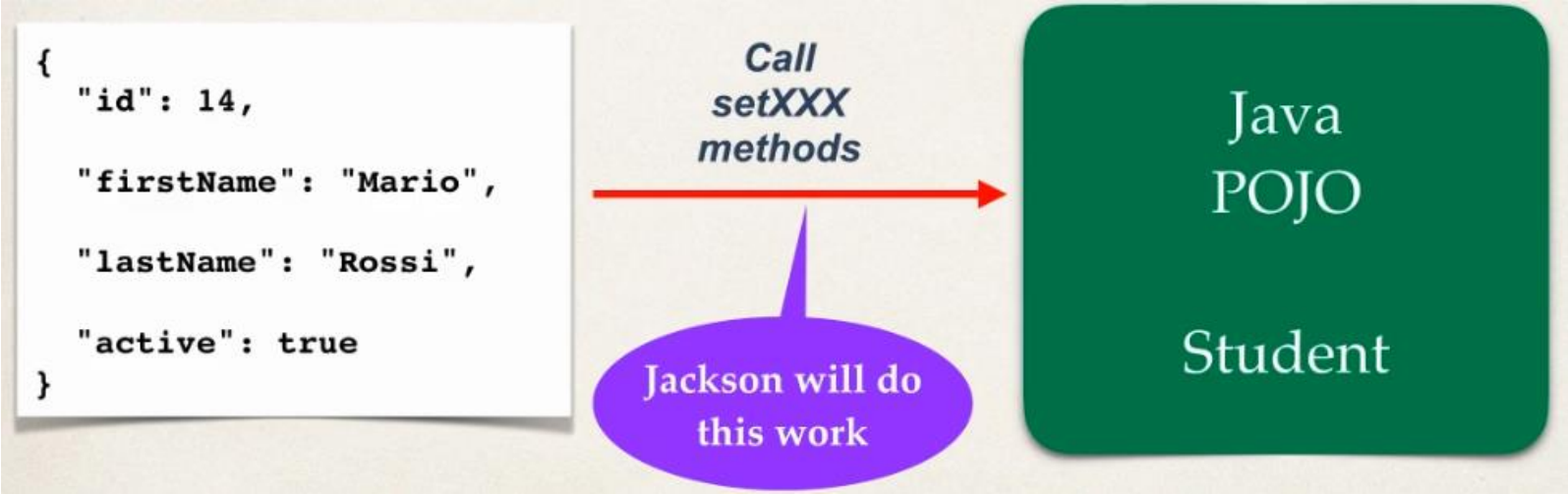
# Jackson data binding

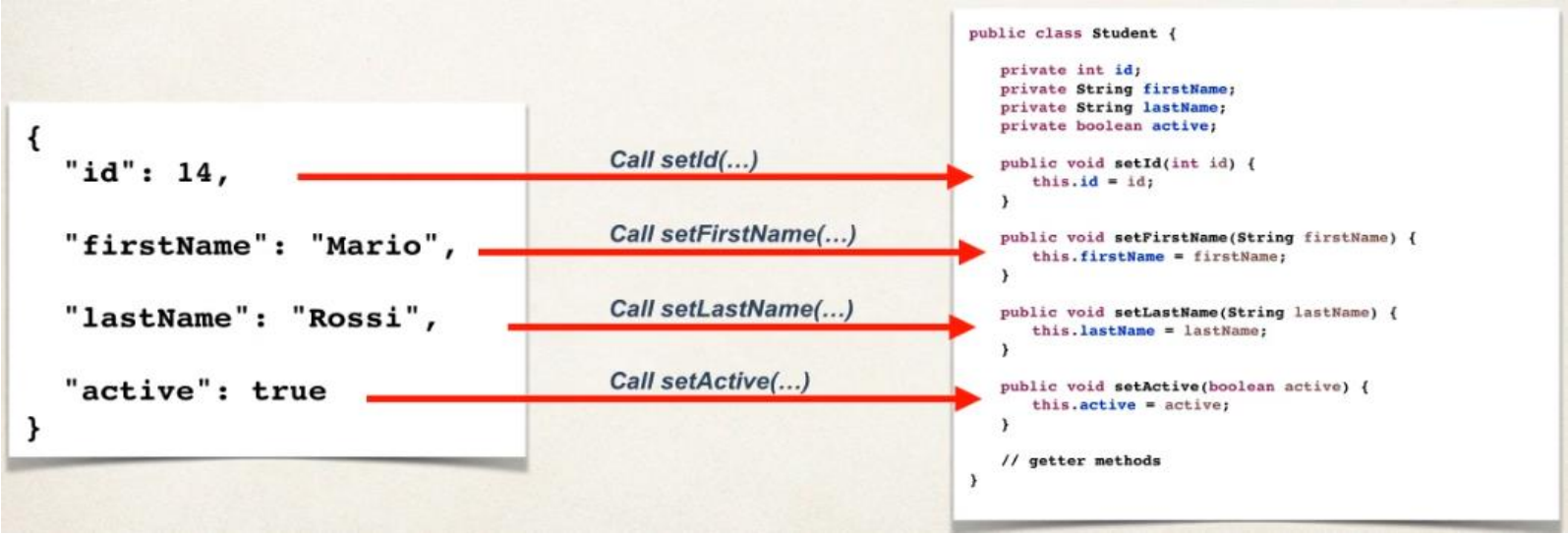
- By default, Jackson will call appropriate getter/setter method



# JSON to Java POJO

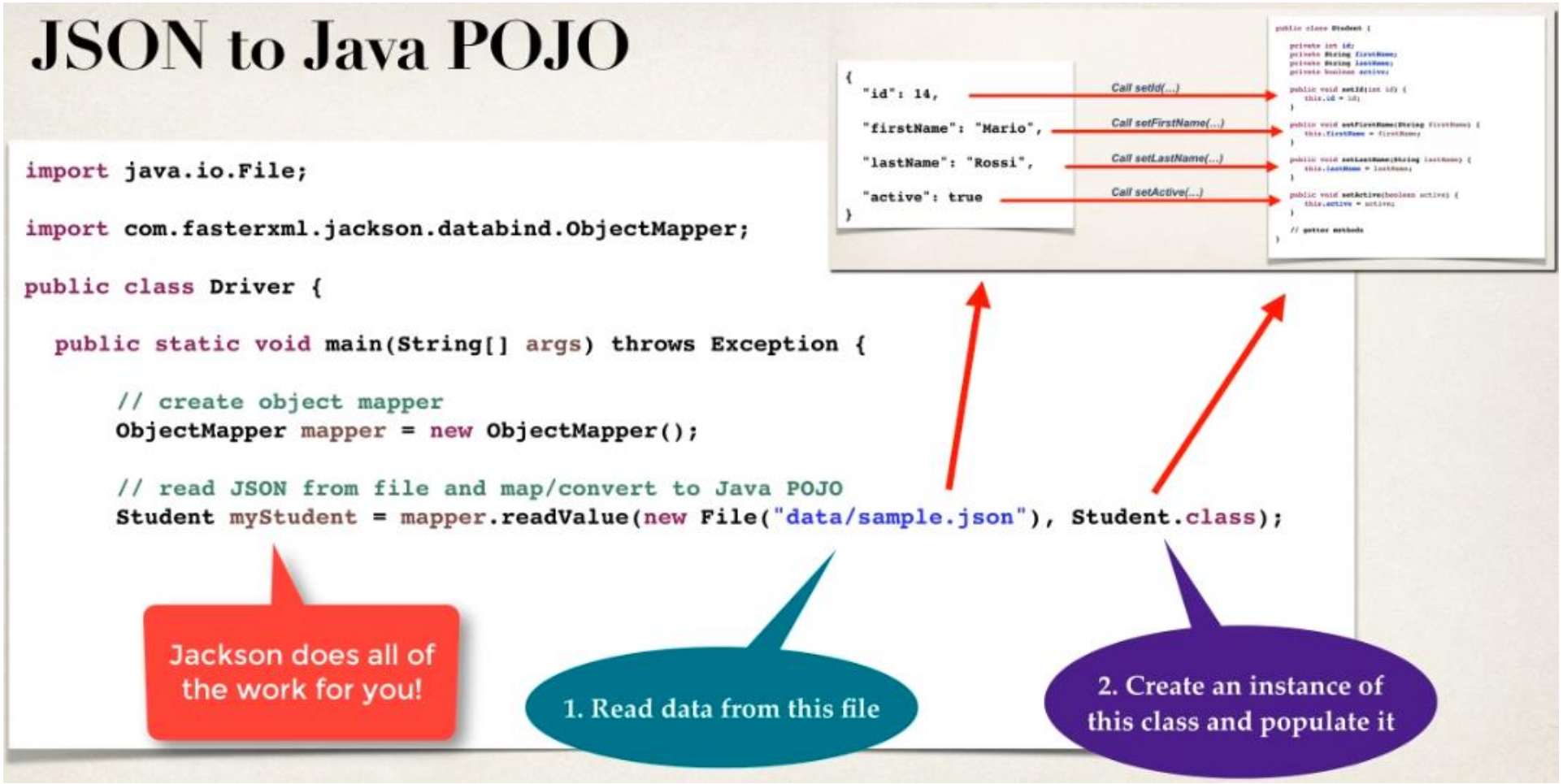
- Convert JSON to JAVA POJO … automatically call **setter** methods on POJO



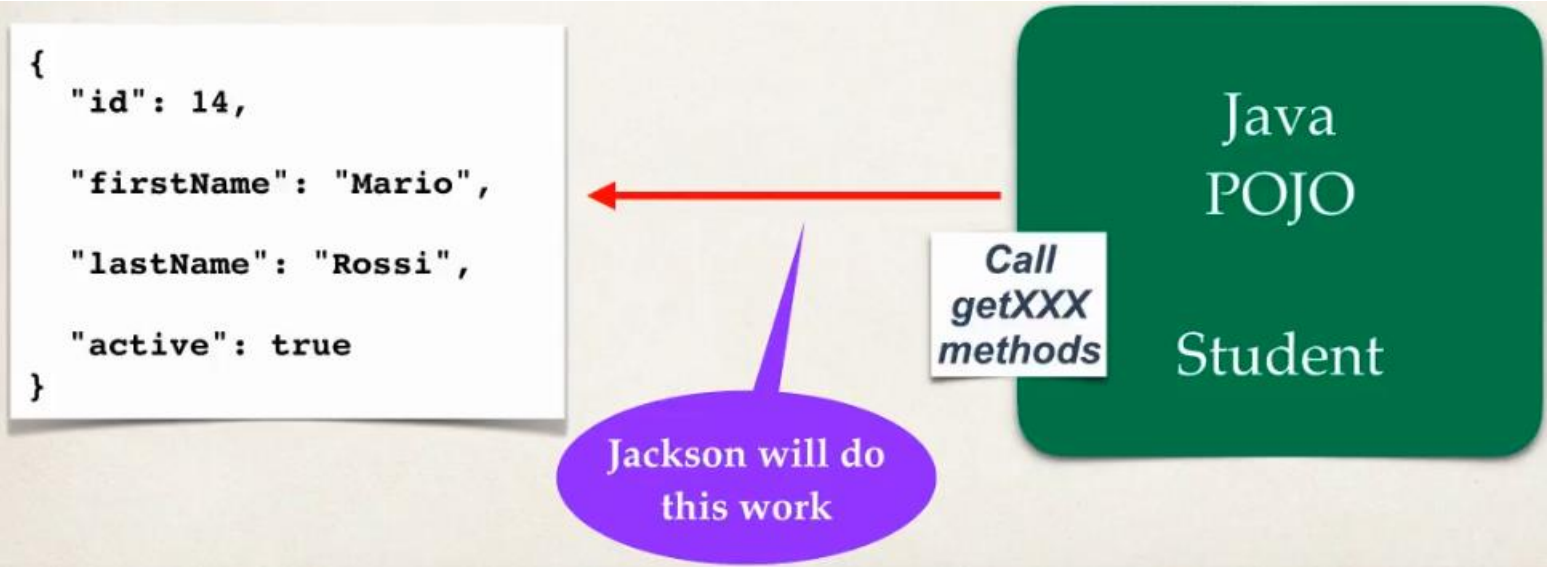
# JSON to Java POJO

- Convert JSON to JAVA POJO … automatically call **setter** methods on POJO



# Java POJO to JSON

- Convert JAVA POJO to JSON … automatically call **getter** methods on POJO



# JSON to Java POJO

- Convert JAVA POJO to JSON … automatically call **getter** methods on POJO

```java
// create object mapper
ObjectMapper mapper = new ObjectMapper();

// read JSON from file and map/convert to Java POJO
Student myStudent = mapper.readValue(new File("data/sample.json"), Student.class);
...

// now write JSON to output file
mapper.enable(SerializationFeature.INDENT_OUTPUT);
mapper.writeValue(new File("data/output.json"), myStudent);
```

Jackson calls the getter methods on Student POJO to create JSON output file

File: data/output.json

```
{
  "id": 14,
  "firstName": "Mario",
  "lastName": "Rossi",
  "active": true
}
```

# Spring and Jackson support

DEMO

- When building Spring REST application
- Spring will automatically handle Jackson Integration
- JSON data binding passed to REST controller is converted to POJO
- POJO being returned from REST controller is converted to JSON
- Ignore unknown properties – non match between JSON and Java POJO
  - **@JsonIgnoreProperties(ignoreUnknown=true)**

# Spring REST- Create a REST controller

# REST over HTTP

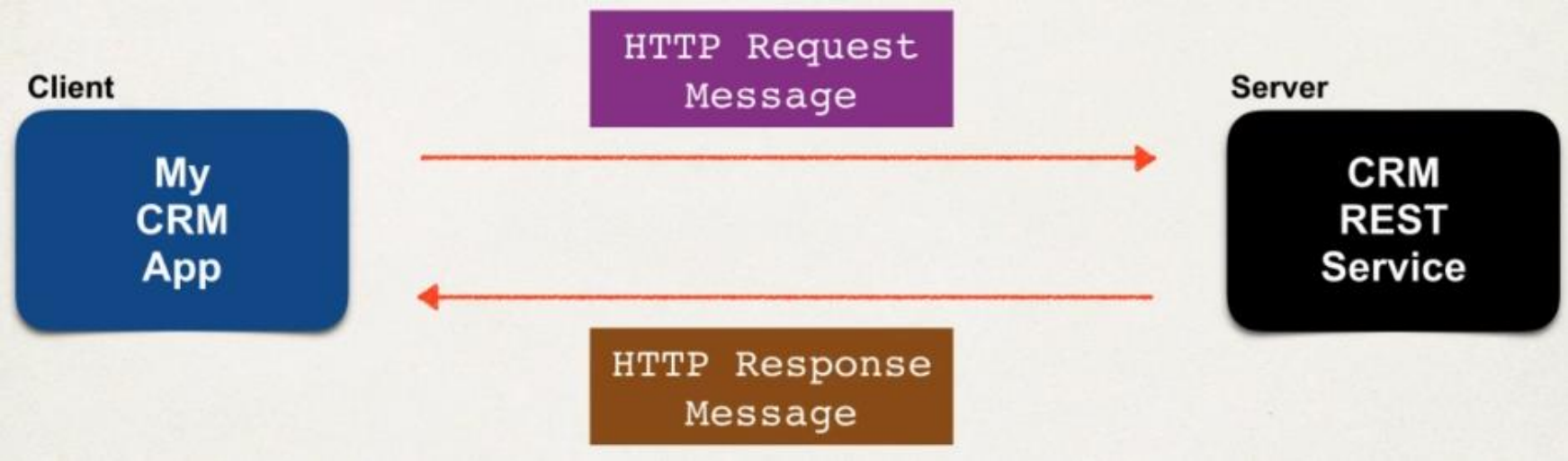
- Most common use of REST is over HTTP

- Leverage HTTP methods for CRUD operations

| HTTP Method | CRUD Operation |
|---|---|
| POST | Create a new entity |
| GET | Read a list of entities or single entity |
| PUT | Update an existing entity |
| DELETE | Delete an existing entity |

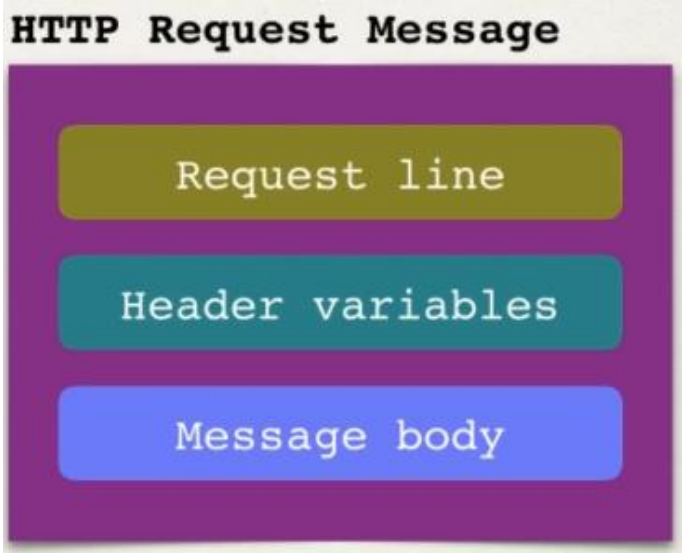
Leverage: Tận dụng

# HTTP messages



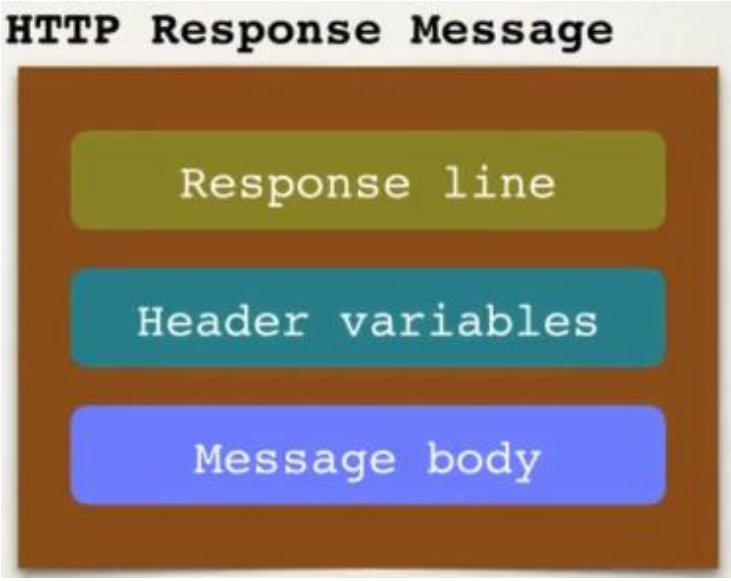
# HTTP Request Message

- Request line: the HTTP command
- Header variables: request metadata
- Message Body: contents of message



# HTTP Response Message

- Response line: server protocol and status code
- Header variables: response metadata
- Message Body: contents of message



# HTTP Response – Status Codes

| Code Range | Description |
|---|---|
| 100 - 199 | Informational |
| 200 - 299 | Successful |
| 300 - 399 | Redirection |
| 400 - 499 | Client error |
| 500 - 599 | Server error |

401 Authentication Required
404 File Not Found

500 Internal Server Error

# MIME Content Types

- Message Format is describes by MIME content type
  - **M**ultipurpose **I**nternet **M**ail-**E**xtension

- Syntax: type/sub-type

- Examples:
  - text/html, text/plain
  - tpplication/json, application/xml

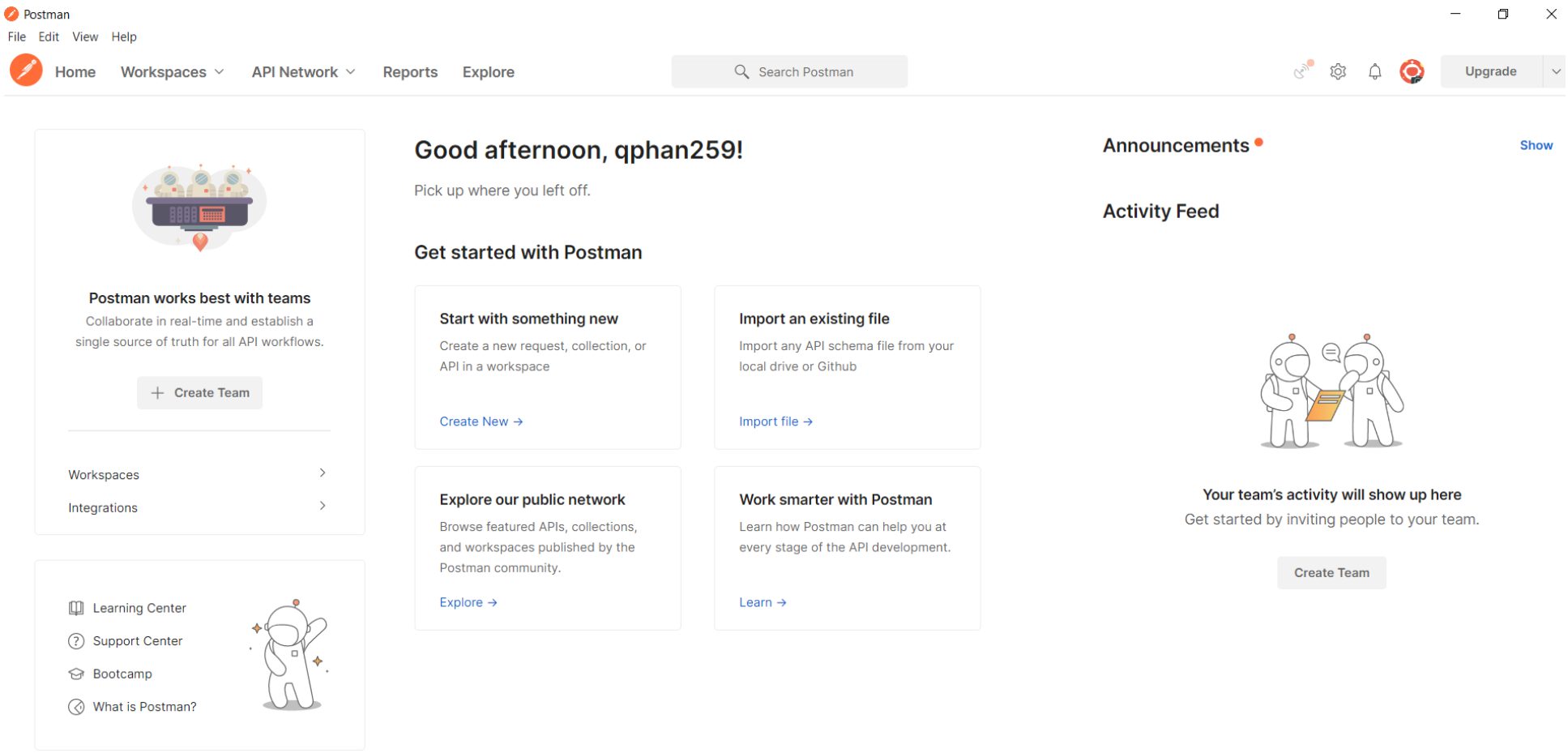
# Client Tool

- We need  a client tool
- Send HTTP requests to REST web services / API
- Plenty of tools available: curl, Postman, etc …

# Install Postman now

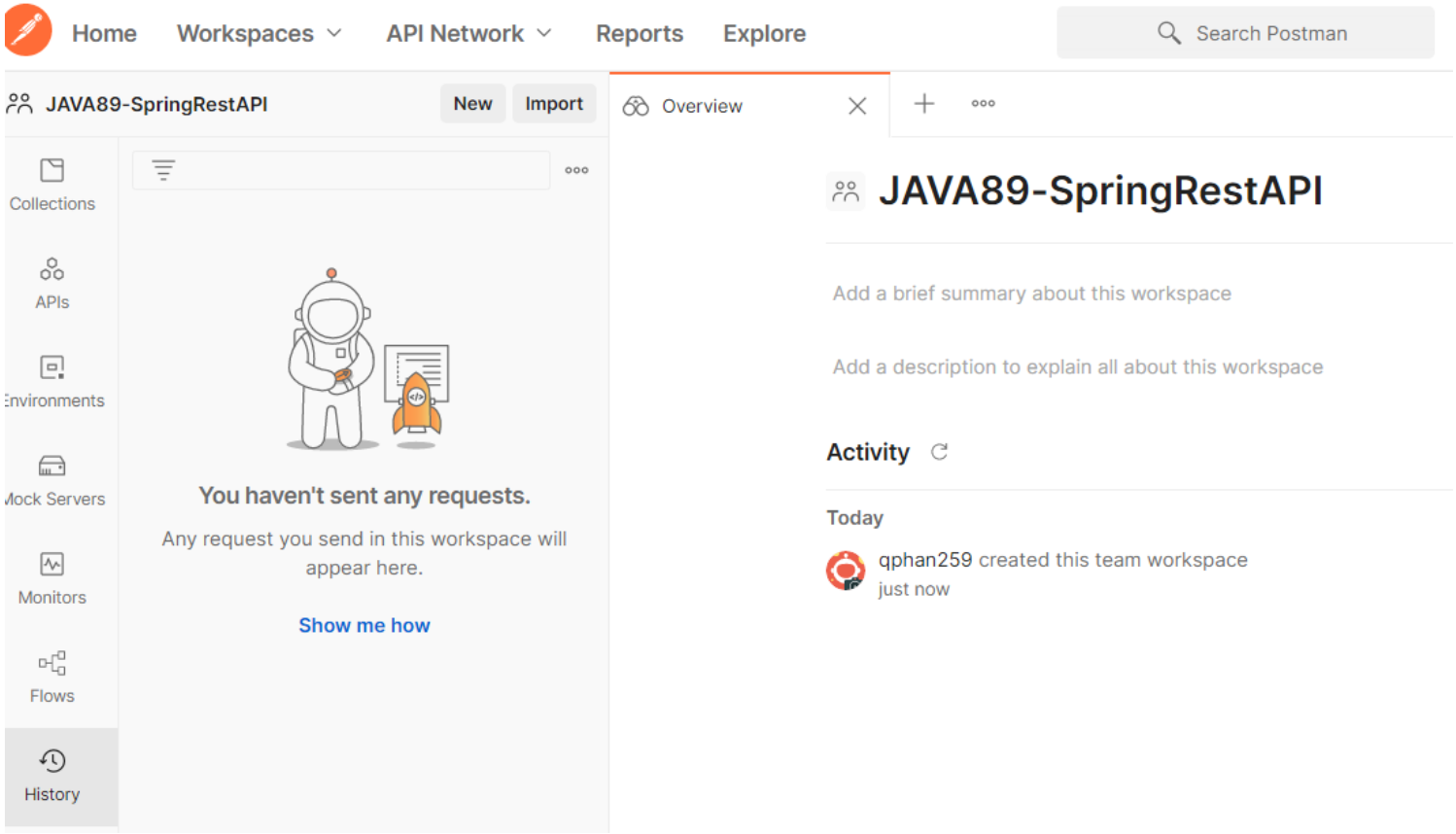
- https://www.postman.com/downloads/

- Time: 9.5.0 – 18.12.2021



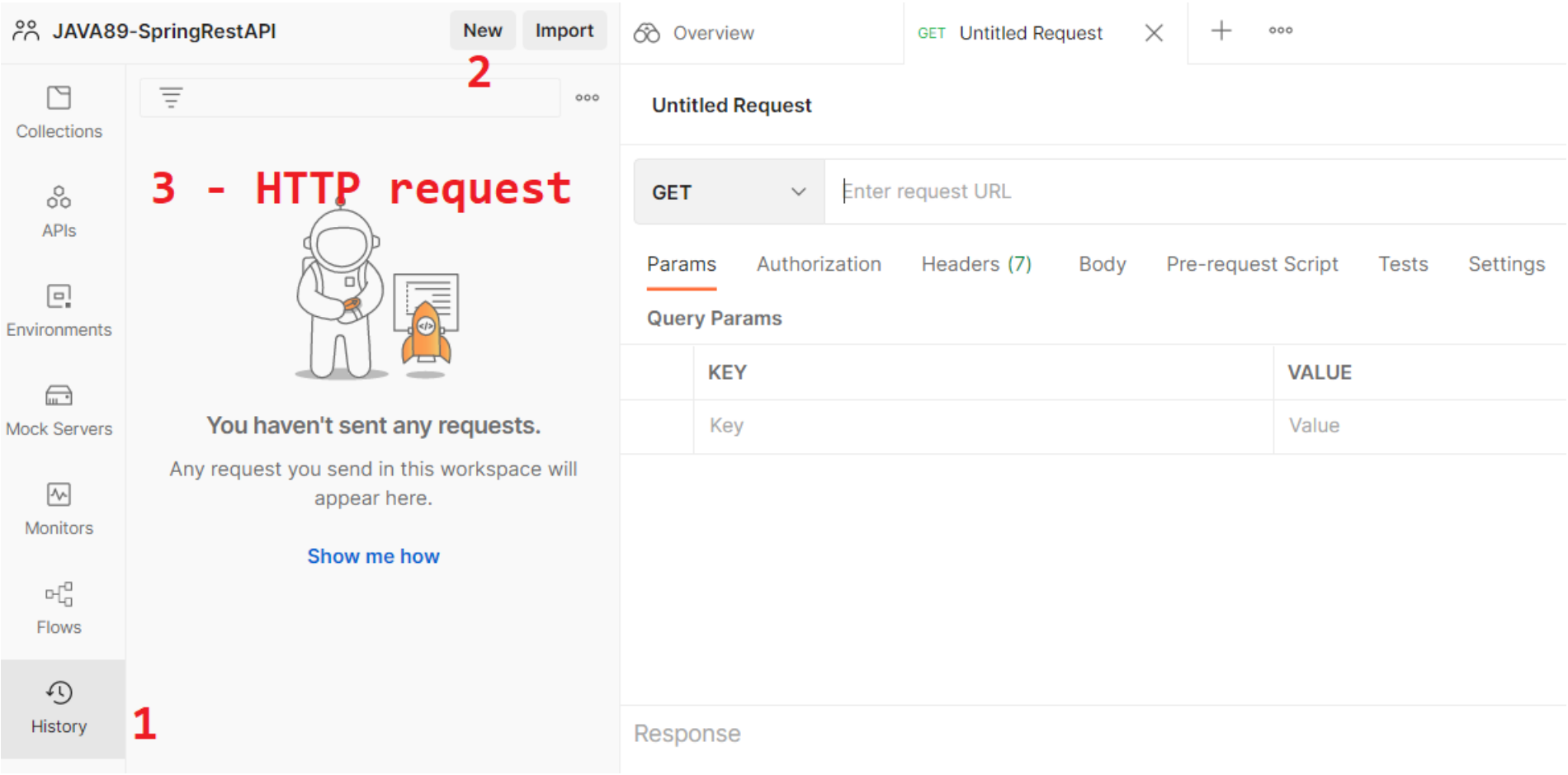
# Postman Testing

- Workspace > Create Workspace >> JAVA89 – SpringRestAPI



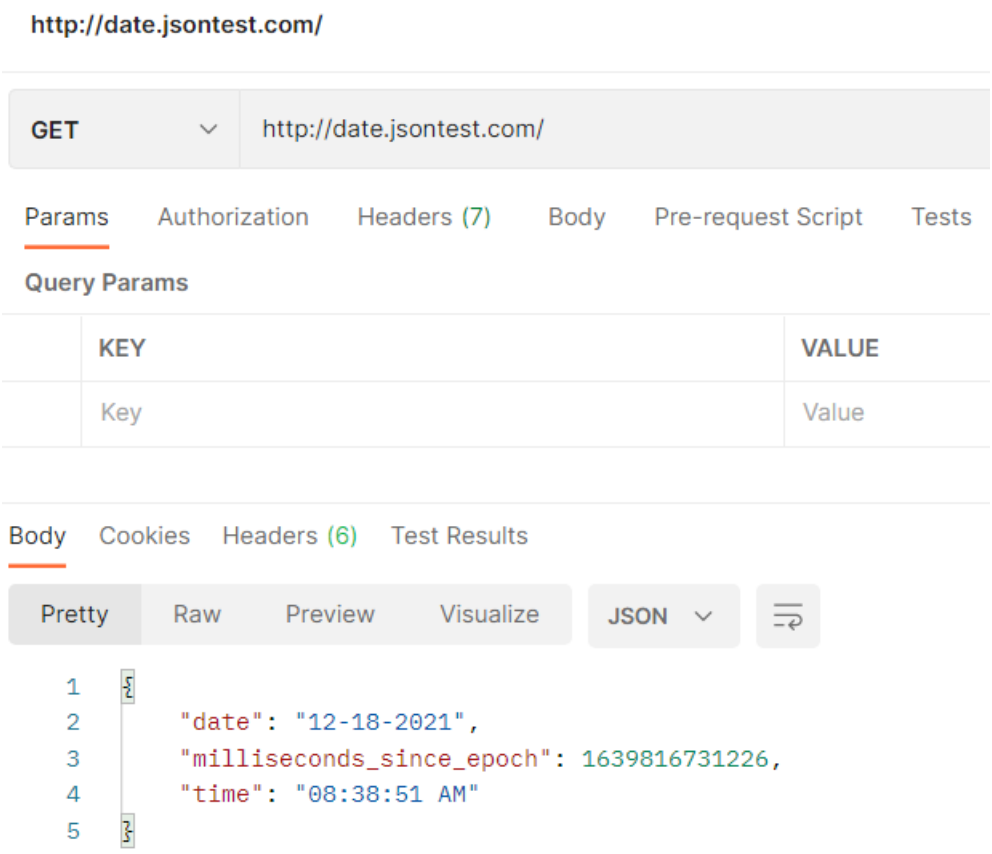
# Postman Testing

- Create a HTTP request



# Postman Testing

- Make some REST API calls
- https://www.jsontest.com/
- https://jsonplaceholder.typicode.com/



# Postman Testing

- https://api.openweathermap.org/data/2.5/weather?q=Danang&appid=09cbe24f3fbf907bbb7b4b5a4cd12fc7

**DEMO**

https://api.openweathermap.org/data/2.5/weather?q=Danang&appid=09cbe24f3fbf907bbb7b4b5a4cd12fc7

GET https://api.openweathermap.org/data/2.5/weather?q=Danang&appid=09cbe24f3fbf907bbb7b4b5a4cd12fc7

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

Query Params

| KEY | VALUE |
|---|---|
| q | Danang |
| appid | 09cbe24f3fbf907bbb7b4b5a4cd12fc7 |

Body Cookies Headers (9) Test Results Sta

Pretty Raw Preview Visualize JSON


```
{
    "coord": {
        "lon": 108.2208,
        "lat": 16.0678
    },
    "weather": [
        {
            "id": 701,
            "main": "Mist",
            "description": "mist",
            "icon": "50d"
        }
    ],
    "base": "stations",
    "main": {
        "temp": 295.14,
        "feels_like": 295.69,
```


Body Cookies Headers (9) Test Results Status: 200 OK Time: 215 ms Size: 782 B

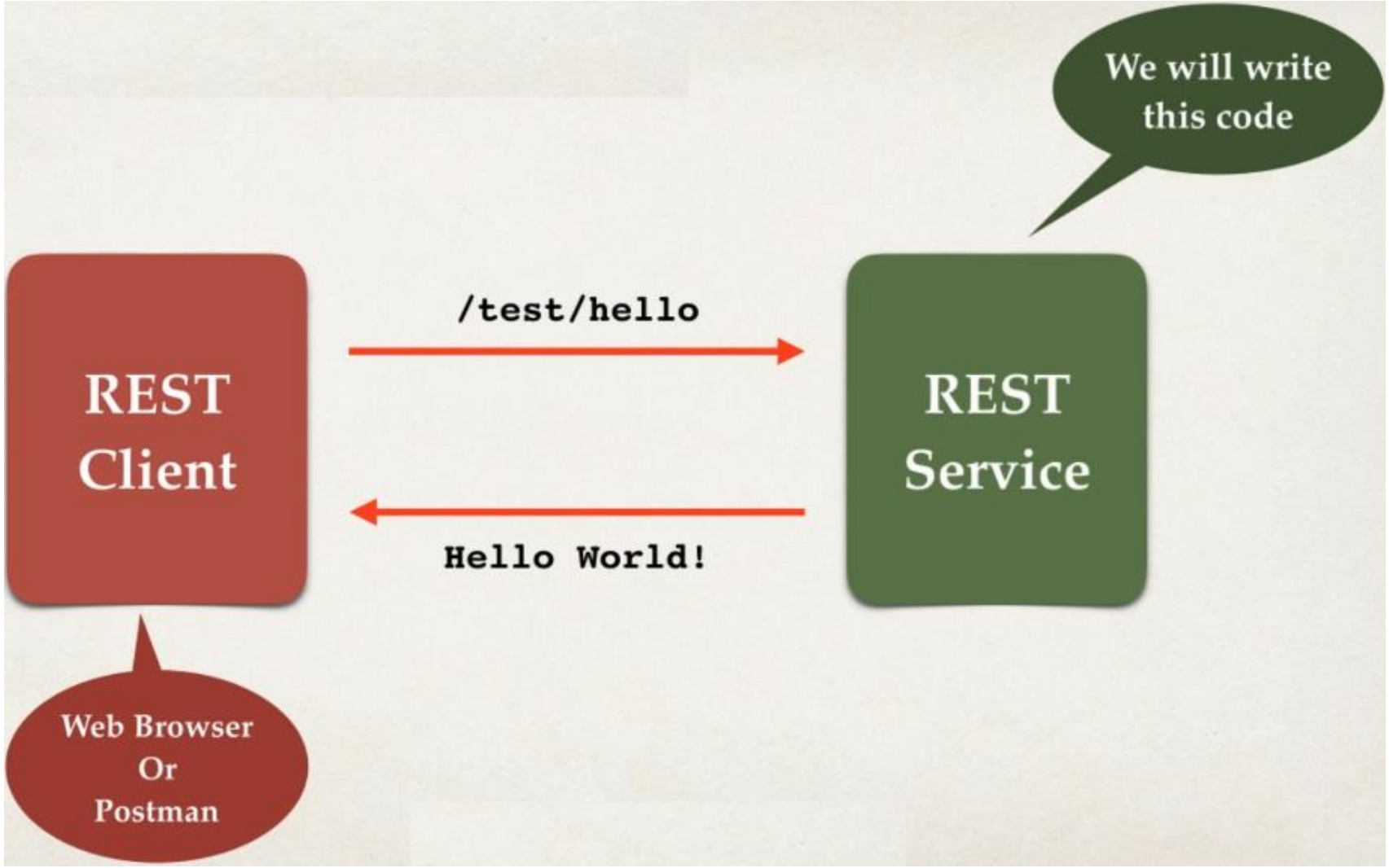
| KEY | VALUE |
|---|---|
| Server | openresty |
| Date | Sat, 18 Dec 2021 08:39:42 GMT |
| Content-Type | application/json; charset=utf-8 |
| Content-Length | 461 |
| Connection | keep-alive |
| X-Cache-Key | /data/2.5/weather?q=danang |
| Access-Control-Allow-Origin | * |
| Access-Control-Allow-Credentials | true |
| Access-Control-Allow-Methods | GET, POST |

# Spring Rest Controller

# Spring REST support

- Spring Web MVC provides support for Spring REST
- New annotation **@RestController**
  - Extension of @Controller
  - Handles REST request and response
- Spring REST will also automatically convert Java POJOs to JSON
  - Required: Jackson project / dependency is on classpath / pom.xml

# Spring REST – Hello world



# Spring REST – Hello world

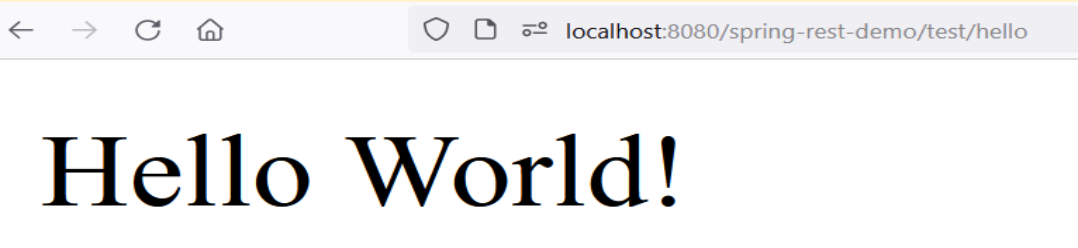
```
@RestController
@RequestMapping("/test")
public class DemoRestController {

    @GetMapping("/hello")
    public String sayHello() {
        return "Hello World!";
    }

}
```

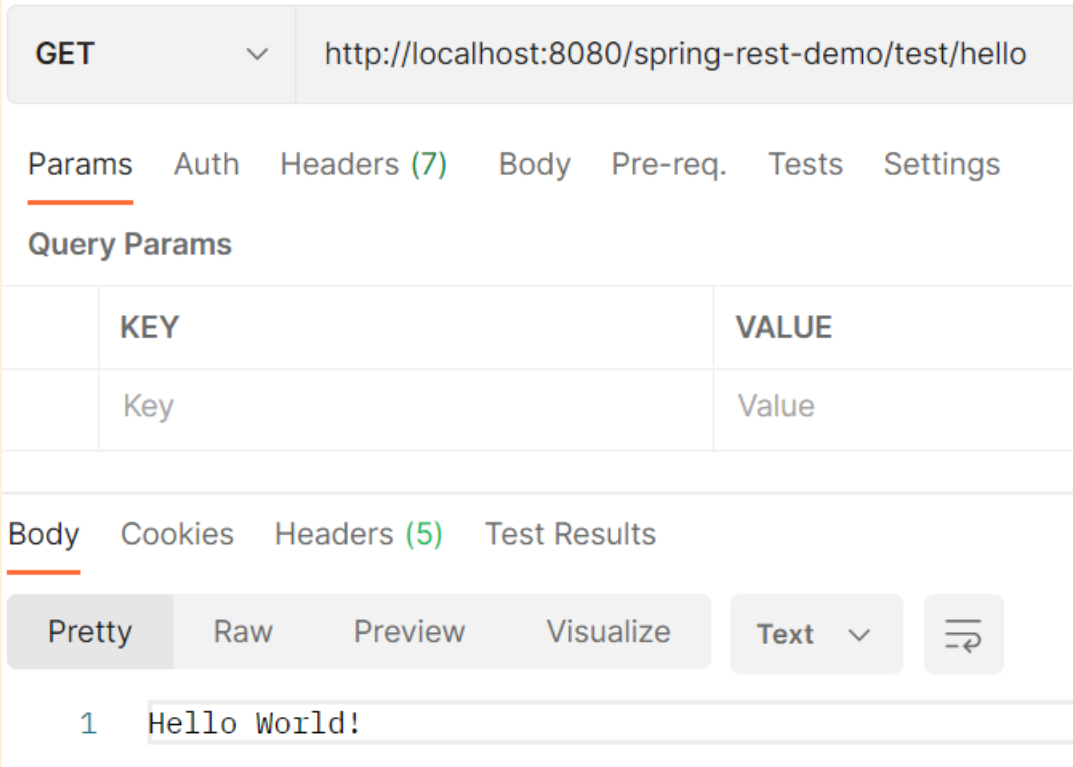
Add REST support

Access the REST endpoint at /test/hello

Returns content to client

# Testing with Postman and Web Browser



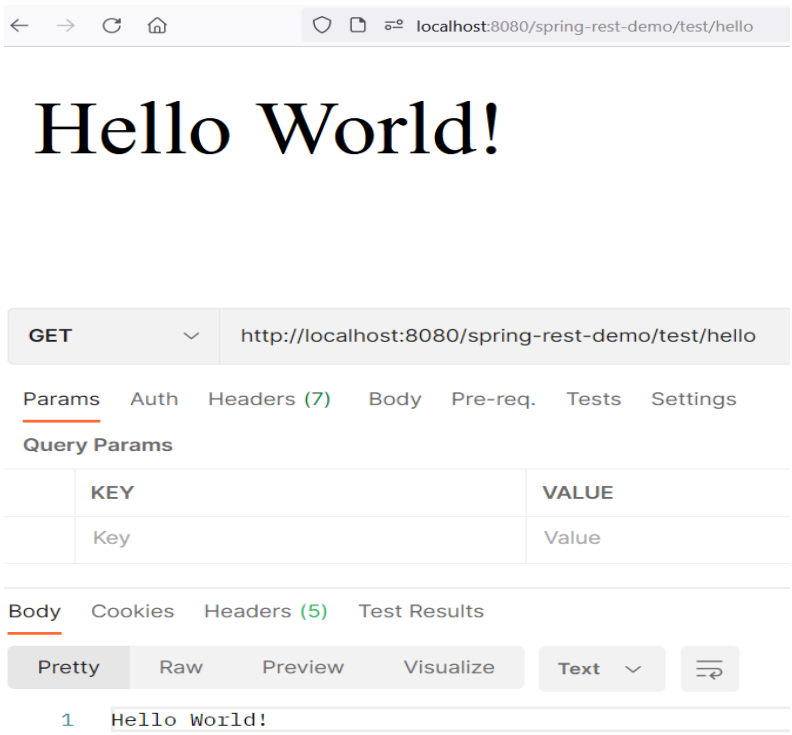




REST endpoint
http://localhost:8080/spring-rest-demo/test/hello

# Web Browser vs Postman

- For simple REST testing with GET request
  - Web Browser and Postman are similar

- However, for andvance testing with POST, PUT etc …
  - Postman has much better support
  - POSTing JSON data, setting content type
  - Passing HTTP request headers, authentication etc ..



# Spring REST – Hello world – Development Process

DEMO

- 1. Add Maven dependency for Spring MVC and Jackson project
- 2. Add code for ALL Java Config: @Configuration
- 3. Add code for ALL Java Config: Servlet Initializer
- 4. Create Spring REST Service using @RestController
- 5. Add default page

# Spring REST – Hello world – Development Process

DEMO

- 1. Add Maven dependency for Spring MVC and Jackson project
- 2. Add code for ALL Java Config: @Configuration
- 3. Add code for ALL Java Config: Servlet Initializer
- 4. Create Spring REST Service using @RestController
- 5. Add default page

# Spring REST – Hello world – Development Process

- 1. Add Maven dependency for Spring MVC and Jackson project

DEMO

```
<!-- Add Spring MVC and REST support -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.0.5.RELEASE</version>
</dependency>

<!-- Add Jackson for JSON converters -->
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.9.5</version>
</dependency>

<!-- Add Servlet support for Spring's AbstractAnnotationConfigDispatcherServletInitializer -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
</dependency>

<!-- Add support for JSP ... get rid of Eclipse error -->
<dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>javax.servlet.jsp-api</artifactId>
    <version>2.3.1</version>
</dependency>
```

# Spring REST – Hello world – Development Process

- 2. Add code for ALL Java Config: @Configuration

DEMO

```
@Configuration
@EnableWebMvc
@ComponentScan("com.spring.rest")
public class WepAppConfigurer implements WebMvcConfigurer {

}
```

# Spring REST – Hello world – Development Process

- 3. Add code for ALL Java Config: Servlet Initializer

DEMO

```
public class WebAppServletInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return null;
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[] { WepAppConfigurer.class };
    }

    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }

}
```

# Spring REST – Hello world – Development Process

- 4. Create Spring REST Service using @RestController

DEMO

```
@RestController
@RequestMapping("/test")
public class DemoRestController {

    @GetMapping("/hello")
    public String sayHello() {
        return "Hello World!";
    }

}
```

# Spring REST – Hello world – Development Process

## 5. Add default page

DEMO

```html
<html>
<body>

<h3>Spring REST Demo</h3>

<hr>

<a href="${pageContext.request.contextPath}/test/hello">Hello</a>

</body>
</html>
```

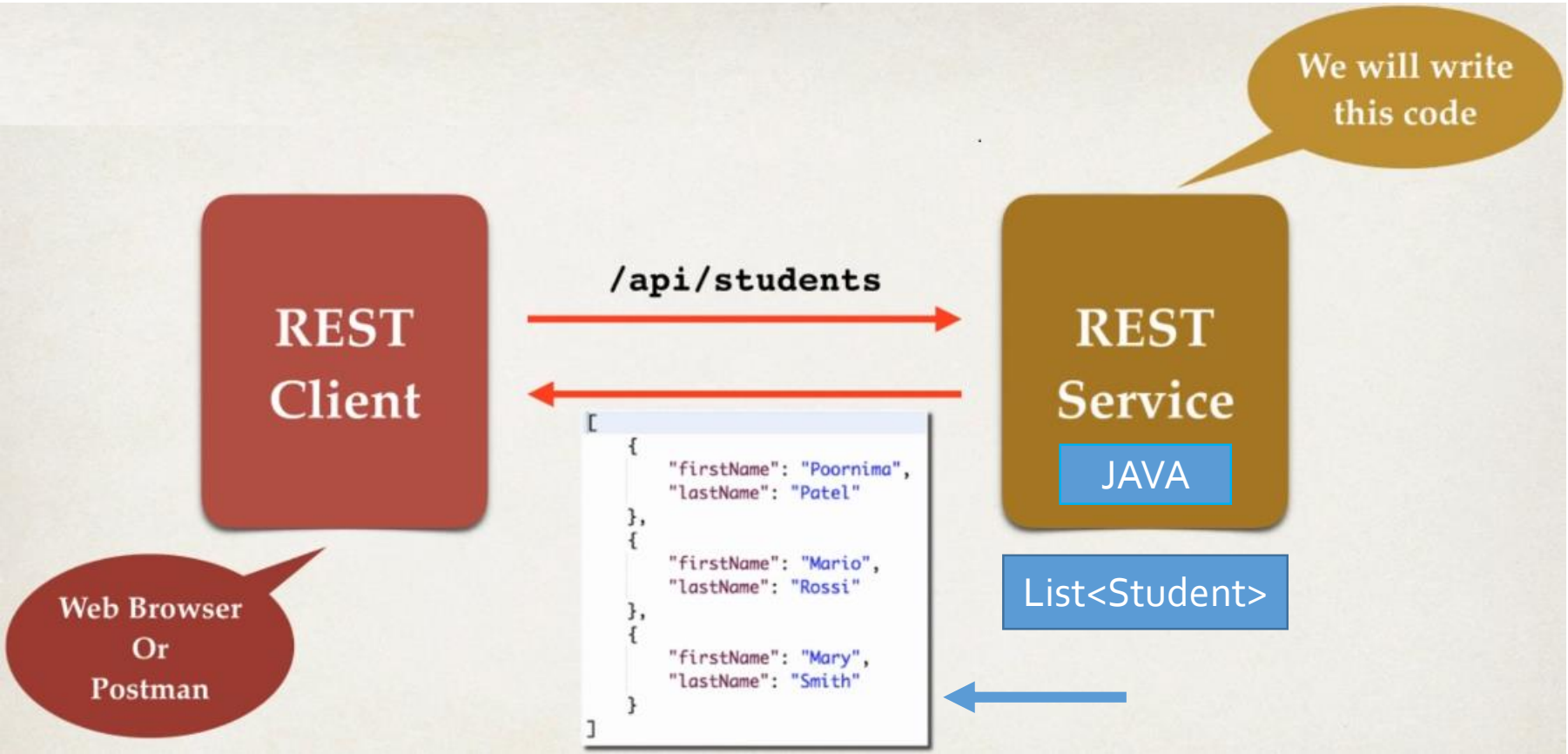
- spring-rest-demo
  - Deployment Descriptor: spring-rest-demo
  - JAX-WS Web Services
  - Java Resources
    - src/main/java
      - com.spring.rest.configuration
        - WebAppServletInitializer.java
        - WepAppConfigurer.java
      - com.spring.rest.controller
        - DemoRestController.java
    - Libraries
  - JavaScript Resources
  - Deployed Resources
  - src
    - main
      - java
        - com
          - spring
            - rest
              - configuration
                - WebAppServletInitializer.java
                - WepAppConfigurer.java
              - controller
                - DemoRestController.java
      - webapp
        - index.jsp
  - target
  - pom.xml
  - spring-rest-demo.iml

# Spring REST – Retrieve POJOs as JSON – Student App

# Create a New Service

- Return a list of students

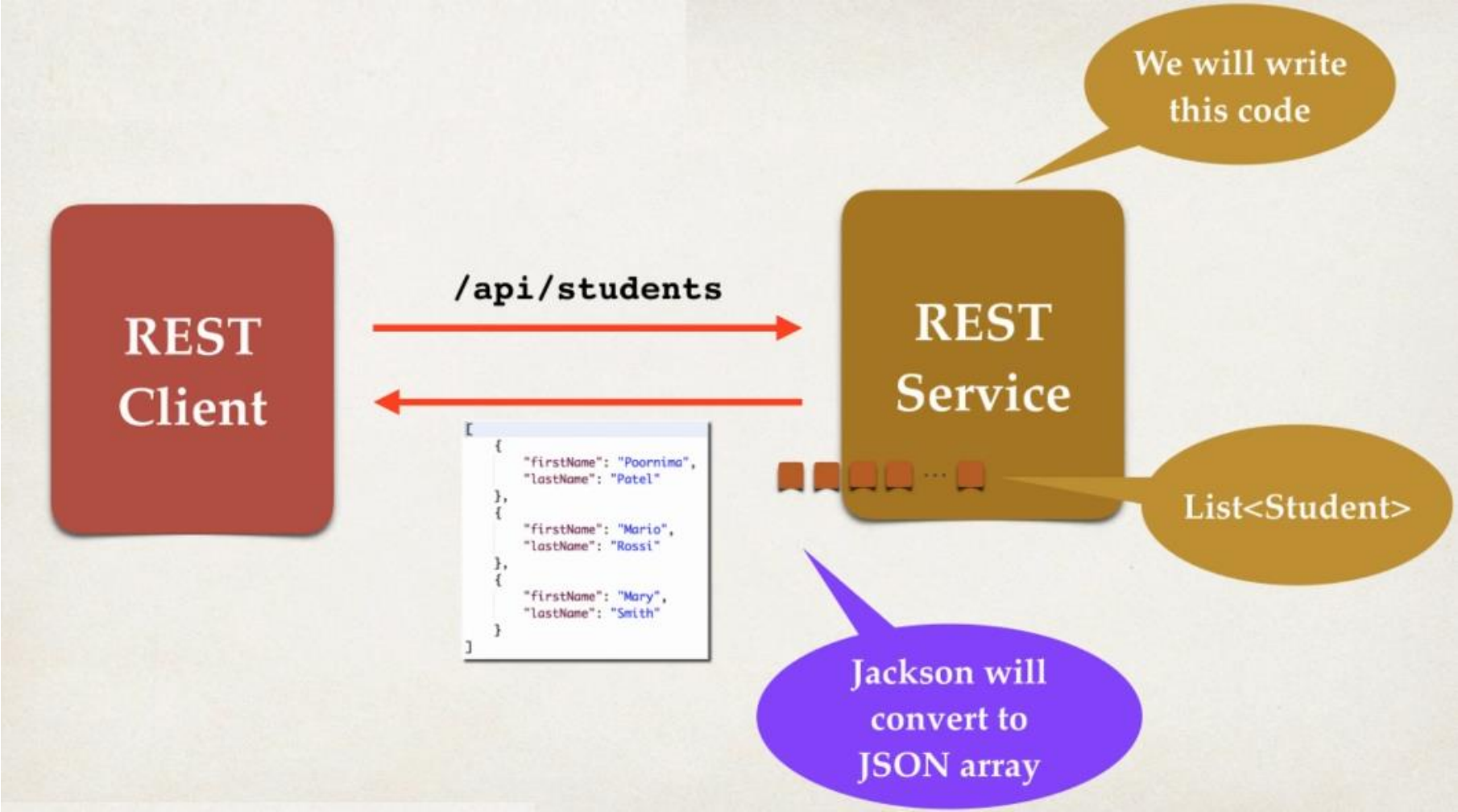
- GET /api/students



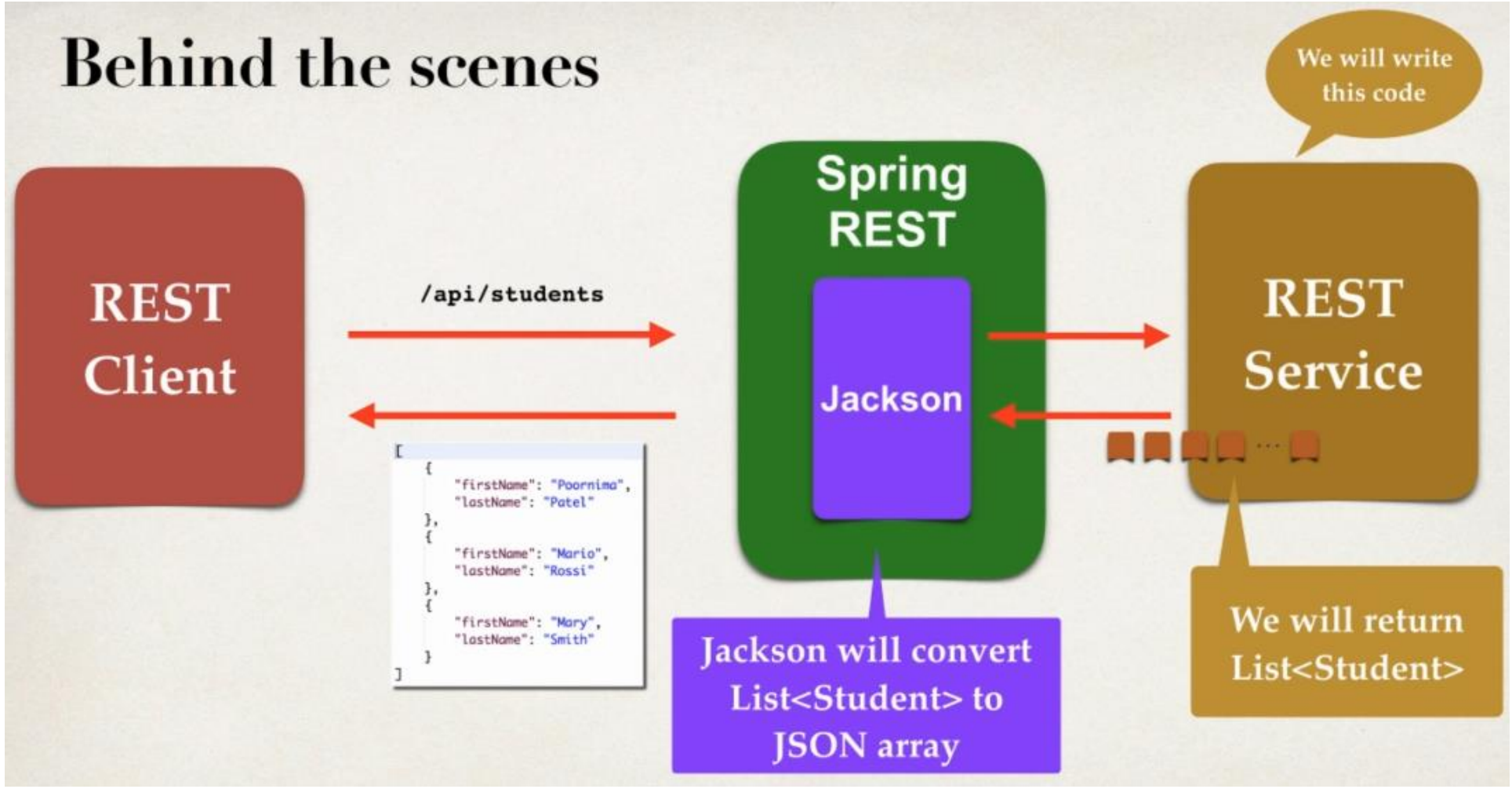
# Convert JAVA POJO to JSON

- REST service will return **List<Student>**
- Need to convert **List<Student>** to JSON
- **Jackson** can help us out with this …
  - JSON data being passed to **REST controller** is converted to JAVA POJO
  - Java POJO being returned from **REST controller** is converted to JSON

# Spring REST Service – Behind the scenes



# Spring REST Service – Behind the scenes



# Spring REST – Student service – Development Process

- 1. Create JAVA POJO class for **Student**
- 2. Create Spring REST Service using **@RestController**

DEMO

# Spring REST – Student service – Development Process

- 1. Create JAVA POJO class for **Student**

DEMO

```
public class Student {
    private String firstName;
    private String lastName;

    // constructor
    // getter, setter
```

# Spring REST – Student service – Development Process

- 2. Create Spring REST Service using **@RestController**

**DEMO**

```java
@RestController
@RequestMapping("/api")
public class StudentRestController {

    @GetMapping("/students")
    public List<Student> getStudents() {
        List<Student> students = DataModel.students();
        return students;
    }

}
```

```java
public static List<Student> students() {
    List<Student> theStudents = new ArrayList<>();
    theStudents.add(new Student("Poornima", "Patel"));
    theStudents.add(new Student("Mario", "Rossi"));
    theStudents.add(new Student("Mary", "Smith"));
    return theStudents;

}
```

**Load data with @PostConstruct**

```java
@PostConstruct
public void loadData() {

    List<Student> theStudents = new ArrayList<>();

    theStudents.add(new Student("Poornima", "Patel"));
    theStudents.add(new Student("Mario", "Rossi"));
    theStudents.add(new Student("Mary", "Smith"));

}
```

# Spring REST – Student service – Development Process

GET http://localhost:8080/spring-rest-demo/api/students

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

Query Params

| KEY | VALUE | DESCRIPTION |
|---|---|---|

Body | Cookies | Headers (5) | Test Results — Status: 200 OK Time: 19 ms Size: 304 B

Pretty | Raw | Preview | Visualize | JSON


```
[
    {
        "firstName": "Poornima",
        "lastName": "Patel"
    },
    {
        "firstName": "Mario",
        "lastName": "Rossi"
    },
    {
        "firstName": "Mary",
        "lastName": "Smith"
    }
]
```

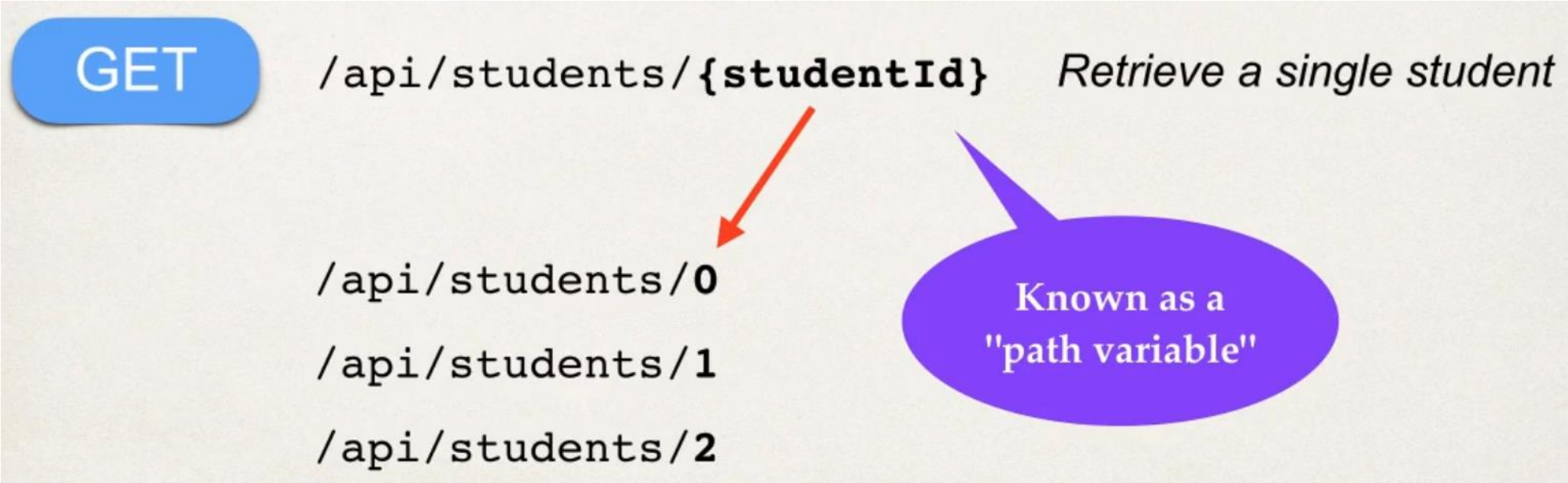

**RESULT**


```
▼ 0:
    firstName:    "Poornima"
    lastName:     "Patel"
▼ 1:
    firstName:    "Mario"
    lastName:     "Rossi"
▼ 2:
    firstName:    "Mary"
    lastName:     "Smith"
```

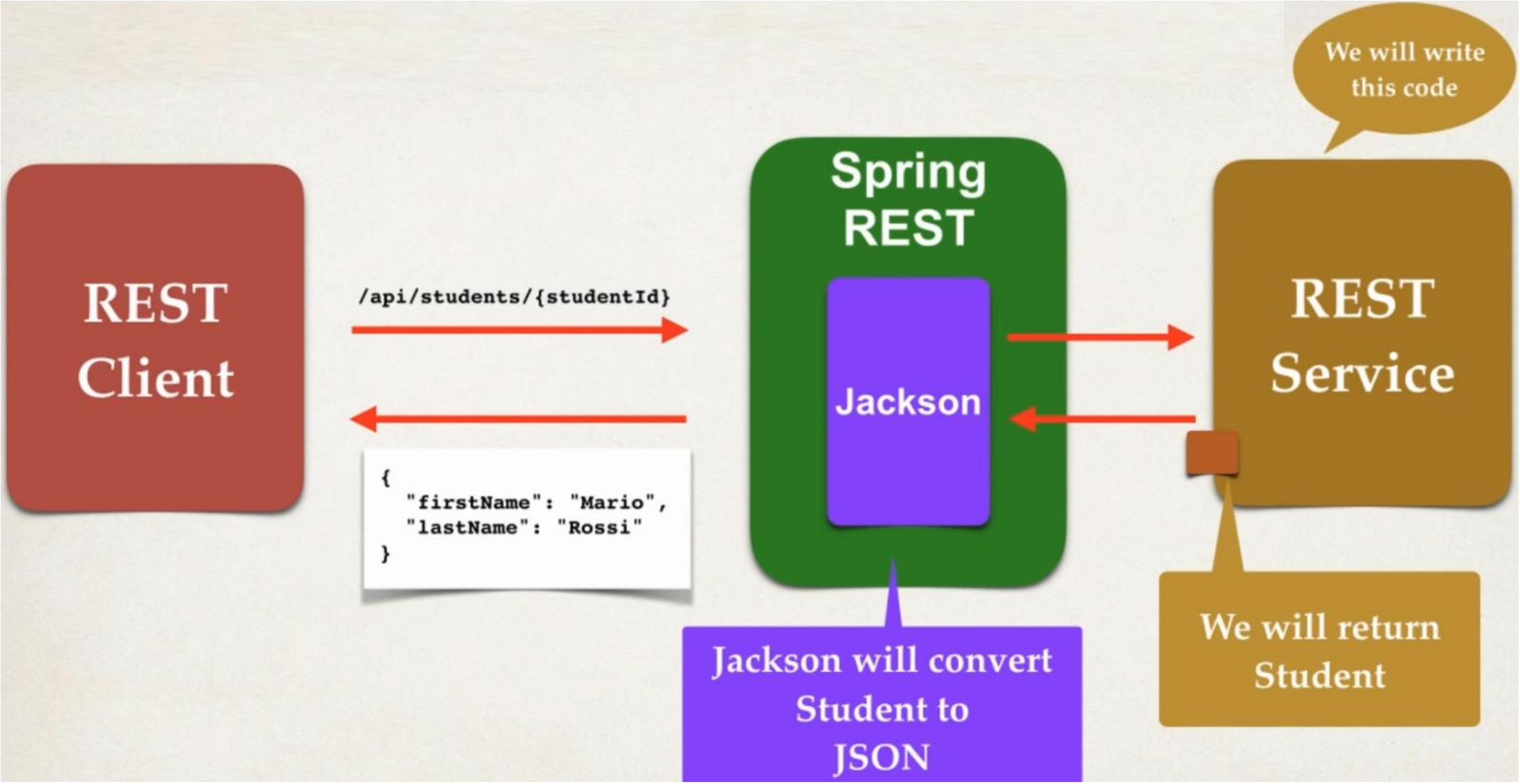
# Spring REST – Path Variables

# Path Variables

- Example: Retrieve single student by id



# Spring REST Service



# Spring REST – Path Variables – Development Process

- 1. Add request mapping to Spring REST Service
  - Using @PathVariable

- 2. Testing with Postman and Web Browser

```java
@GetMapping("/students/{studentId}")
public Student getStudent(@PathVariable int studentId) {
    return students.stream()
                    .filter(student -> studentId == student.getId())
                    .findFirst()
                    .orElse(null);
}
```

GET http://localhost:8080/spring-rest-demo/api/students/2

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

Query Params

| KEY | VALUE |
|---|---|

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2     "id": 2,
3     "firstName": "Mario",
4     "lastName": "Rossi"
5 }
```

# Spring REST – Exception Handling

# Problems

- Bad student id of non-number …

- Bad student index – out of array range

localhost:8080/spring-rest-demo/api/students/abc 270%

## HTTP Status 400 – Bad Request

**Type** Status Report

**Description** The server cannot or will not process the request due to something that is perceived to be a client error (e.g., malformed request syntax, invalid request message framing, or deceptive request routing).

**Apache Tomcat/9.0.54**

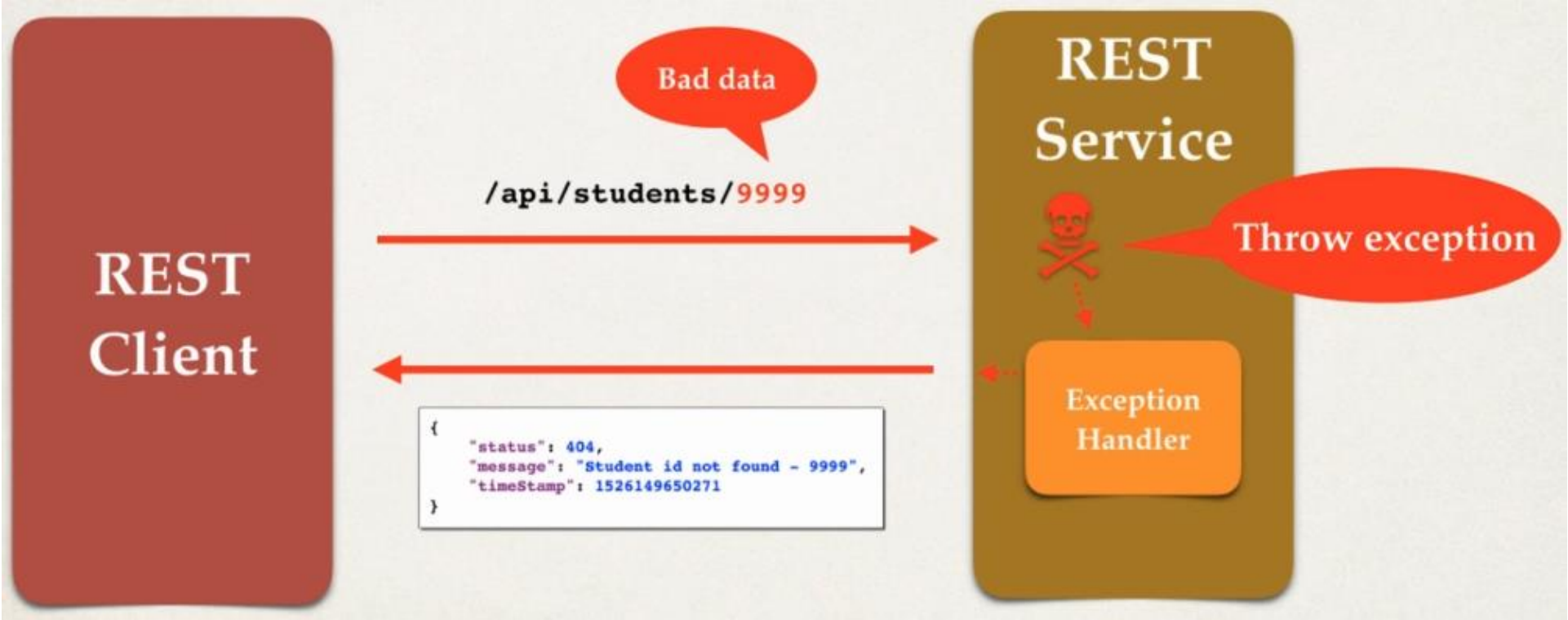
# How to solve it

- Handle the exception and return error as JSON

```
{
    "status": 404,
    "message": "Student id not found - 9999",
    "timeStamp": 1526149650271
}
```

Our desired output exception / error formatted as JSON

# Spring REST – Exception Handler



# Spring REST – Exception Handler – Development Process

DEMO

- 1. Create a customer error response class
- 2. Create a custom exception class
- 3. Update REST service to throw exception if student not found
- 4. Add an exception handler method using **@ExceptionHandler**

# Spring REST – Exception Handler – Development Process

- 1. Create a customer error response class

DEMO

- CER class will be sent back to client as JSON
- We will define as Java class
- Jackson with handle converting it to JSON

```
public class StudentErrorResponse {

    private int status;
    private String message;
    private LocalDateTime timeStamp;

    // constructor
    // getter, setter
```

```
{
    "status": 404,
    "message": "Student id not found - 9999",
    "timeStamp": 1526149650271
}
```

Our desired output exception / error formatted as JSON

# Spring REST – Exception Handler – Development Process

DEMO

- 2. Create a custom exception class
  - The custom student exception will used by our REST service
  - Business, if we can't find student then throw an exception
  - Need to define a customer student class - **StudentNotFoundException**

```java
public class StudentNotFoundException extends RuntimeException {

    private static final long serialVersionUID = 1698807995923929200L;

    public StudentNotFoundException(String message, Throwable cause) {
        super(message, cause);
    }

    public StudentNotFoundException(String message) {
        super(message);
    }

    public StudentNotFoundException(Throwable cause) {
        super(cause);
    }

}
```

# Spring REST – Exception Handler – Development Process

- 3. Update REST service to throw exception if student not found

DEMO

```
@GetMapping("/students/{studentId}")
public Student getStudent(@PathVariable int studentId) {
    Optional<Student> opt = students.stream()
            .filter(student -> studentId == student.getId())
            .findFirst();

    if (!opt.isPresent()) {
        throw new StudentNotFoundException("Student not found - " + studentId);
    }

    return opt.get();
}
```

# Spring REST – Exception Handler – Development Process

- 4. Add an exception handler method using **@ExceptionHandler**

DEMO

  - Define exception handler method(s) with @ExceptionHandler annotation
  - Exception Handler will return a ResponseEntity
  - ResponseEntity
    - is a wrapper for the HTTP response object
    - provides fine-grained control to sepecify: HTTP status code, HTTP headers and Response Body

# Spring REST – Exception Handler – Development Process

- 4. Add an exception handler method using **@ExceptionHandler**

DEMO

```
@GetMapping("/students/{studentId}")
public Student getStudent(@PathVariable int studentId) {
    Optional<Student> opt = students.stream()
            .filter(student -> studentId == student.getId())
            .findFirst();

    if (!opt.isPresent()) {
        throw new StudentNotFoundException("Student not found - " + studentId);
    }

    return opt.get();
}
```

Exception handler method

Type of Response Body

Exception type to catch / handle

```
@ExceptionHandler
public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc) {
    StudentErrorResponse error = new StudentErrorResponse();
    error.setStatus(HttpStatus.NOT_FOUND.value());
    error.setMessage(exc.getMessage());
    error.setTimeStamp(LocalDateTime.now());
    return new ResponseEntity<StudentErrorResponse>(error, HttpStatus.NOT_FOUND);
 }
```

Body

Status code

# Spring REST – Exception Handler – Development Process

**RESULT**

localhost:8080/spring-rest-demo/api/students/1

JSON Raw Data Headers

Save Copy Collapse All Expand All

```
id:          1
firstName:   "Poornima"
lastName:    "Patel"
```

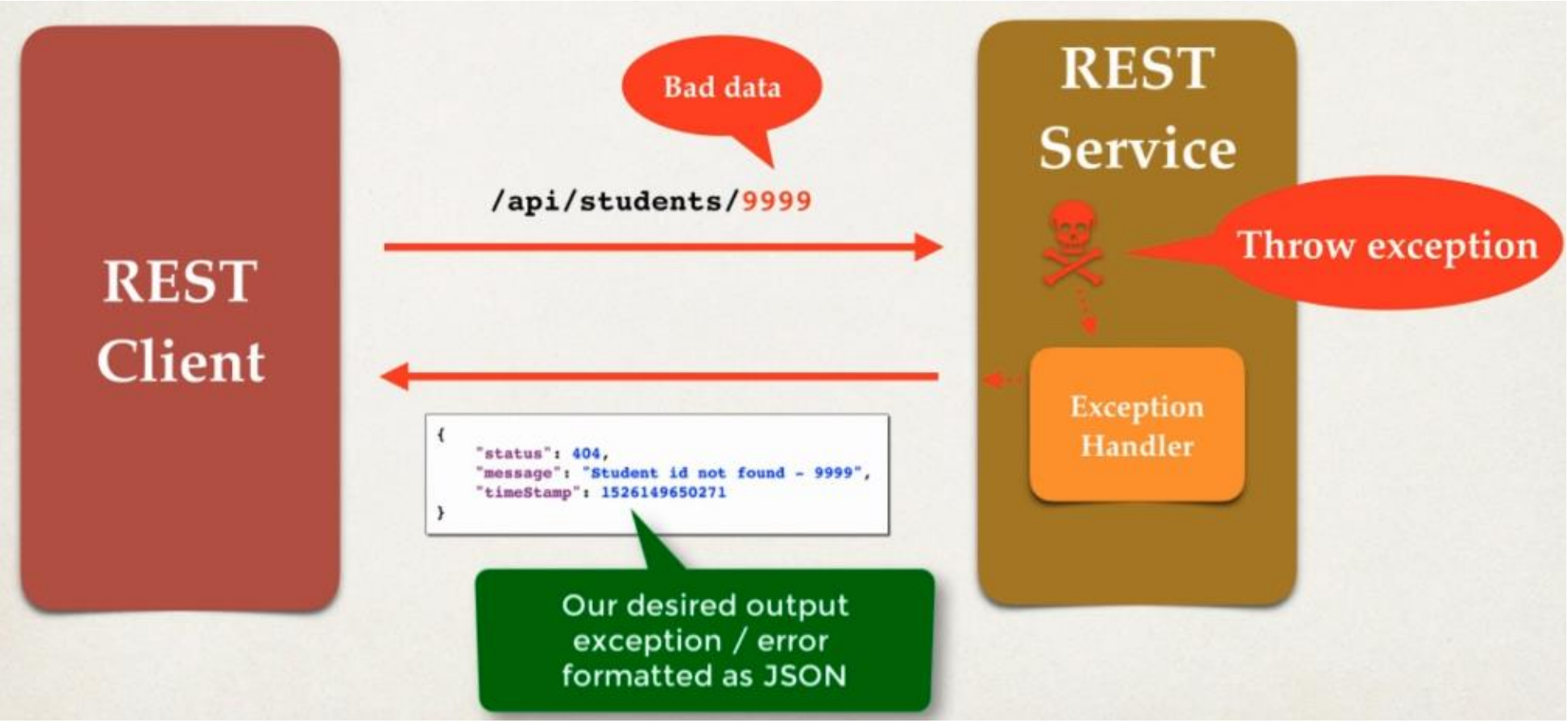
localhost:8080/spring-rest-demo/api/students/4

JSON Raw Data Headers

Save Copy Collapse All Expand All Filter JSON

```
status:      404
message:     "Student not found - 4"
▶ timeStamp: {…}
```

# Spring REST – Exception Handler - Overview



# Spring REST – Add generic exception

- For any bad request, handle with Exception class

localhost:8080/spring-rest-demo/api/students/wrong_id 260%

## HTTP Status 400 – Bad Request

**Type** Status Report

**Description** The server cannot or will not process the request due to something that is perceived to be a client error (e.g., malformed request syntax, invalid request message framing, or deceptive request routing).

```
@ExceptionHandler
public ResponseEntity<StudentErrorResponse> handleException(Exception exc) {

    StudentErrorResponse error = new StudentErrorResponse();

    error.setStatus(HttpStatus.BAD_REQUEST.value());
    error.setMessage(exc.getMessage());
    error.setTimeStamp(LocalDateTime.now());

    return new ResponseEntity<>(error, HttpStatus.BAD_REQUEST);
}
```

localhost:8080/spring-rest-demo/api/students/wrong_id 260%

JSON Raw Data Headers

Save Copy Collapse All Expand All Filter JSON

```
status:    400
message:   "Failed to convert value of type 'java.lang.String' to required type 'int'; nested exception is java.lang.NumberFormatException: For input string: \"wrong_id\""
```

# Spring REST – Global Exception Handling

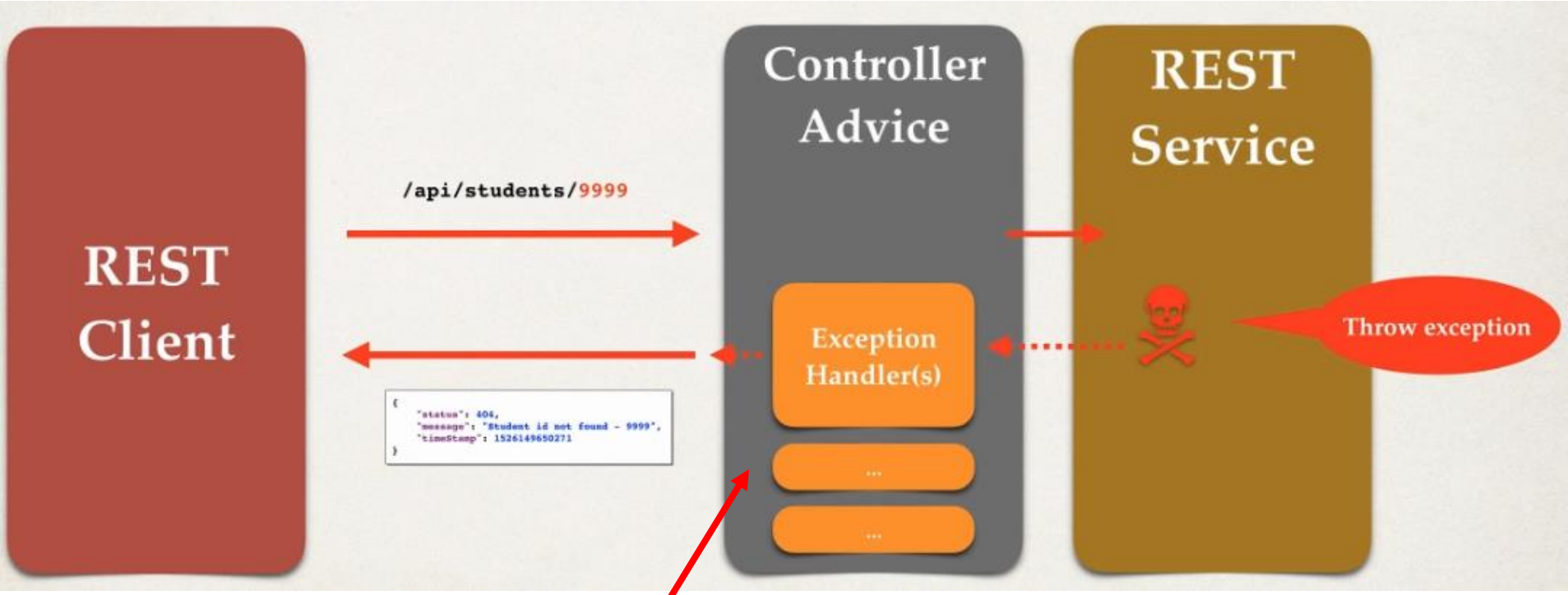
- It works, but
  - Exception handler code is only for the specific REST controller
  - Can't be resued by other controllers
- We need global exception handler
  - Promotes reuse
  - Centralizes exception handling

# Spring **@ControllerAdvice**

- @ControllerAdvice is similar to an interceptor / filter
- Pre-process request to controllers
- Post-process response to handle exceptions
- Perfect for global exception handling

AOP behind the scenes

# Spring @ControllerAdvice




Global Exception Handling

# Spring **@ControllerAdvice –** Development process

DEMO

- 1. Create new @ControllerAdvice
- 2. Refactor REST service … remove exception handling code
- 3. Add exception handling code to @ControllerAdvice

# Spring **@ControllerAdvice** – Development process

DEMO

- 1. Create new @ControllerAdvice

```
@ControllerAdvice
public class StudentRestExceptionHandler {
```

# Spring **@ControllerAdvice** – Development process

- 2. Refactor REST service … remove exception handling code

**DEMO**

File: StudentRestController.java

```
@RestController
@RequestMapping("/api")
public class StudentRestController {
  …

  @ExceptionHandler
  public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc) {

    StudentErrorResponse error = new StudentErrorResponse();

    error.setStatus(HttpStatus.NOT_FOUND.value());
    error.setMessage(exc.getMessage());
    error.setTimeStamp(System.currentTimeMillis());

    return new ResponseEntity<>(error, HttpStatus.NOT_FOUND);
  }

}
```

Remove this code

# Spring **@ControllerAdvice** – Development process

- 3. Add exception handling code to @ControllerAdvice

**DEMO**

File: StudentRestExceptionHandler.java

Same code as before

```
@ControllerAdvice
public class StudentRestExceptionHandler {

  @ExceptionHandler
  public ResponseEntity<StudentErrorResponse> handleException(StudentNotFoundException exc) {

    StudentErrorResponse error = new StudentErrorResponse();

    error.setStatus(HttpStatus.NOT_FOUND.value());
    error.setMessage(exc.getMessage());
    error.setTimeStamp(System.currentTimeMillis());

    return new ResponseEntity<>(error, HttpStatus.NOT_FOUND);
  }

}
```

# Template----------