



REST APIs

Part 1

MARIANO CECCATO (mariano.ceccato@univr.it)

SOFIA MARI (sofia.mari@univr.it)

Table of contents



Rest API



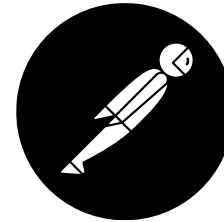
HTTP



Spotify



OpenAPI

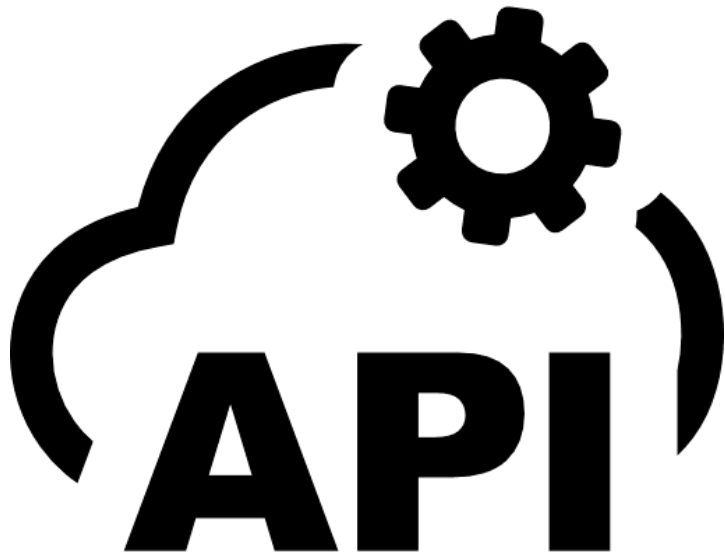


Postman



Final
Exercise

REST APIs



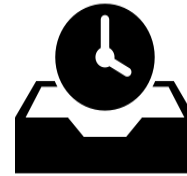
Application **P**rogramming **I**nterface

REpresentational **S**tate **T**ransfer

REST APIs



Uniform Interface



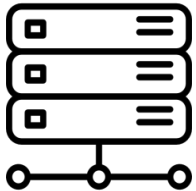
Cacheable



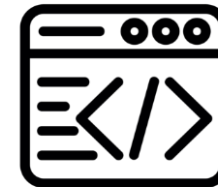
Client-Server



Layered System



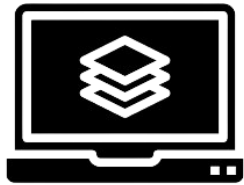
Stateless



Code on-demand

HTTP Protocol

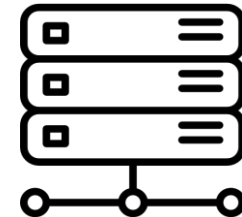
Hyper **T**ext **T**ransfer **P**rotocol



Application
Layer

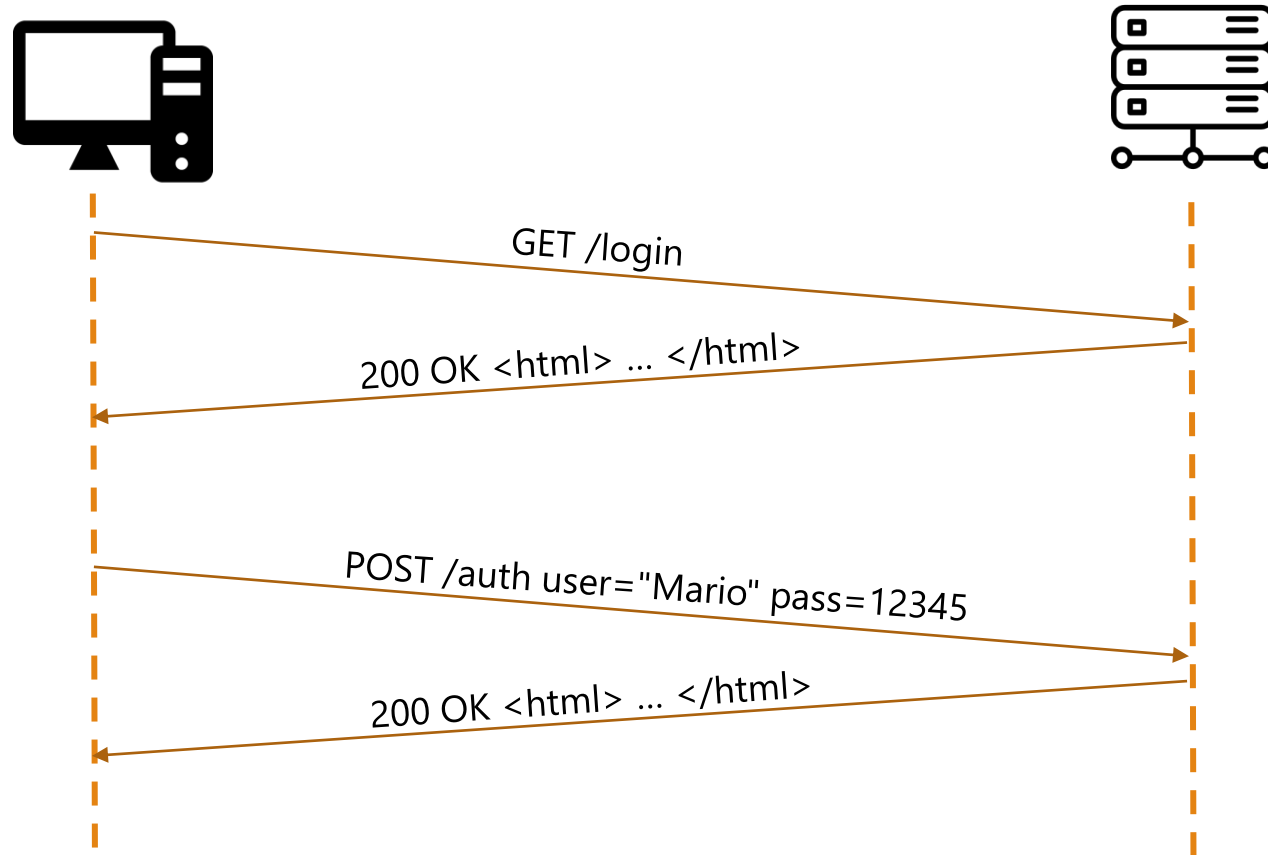


Client-Server



Stateless

HTTP Interaction



HTTP Request

		Path	Protocol Version
Method	POST	/auth	HTTP/1.1
Headers	Host: www.example.com Accept: application/json Content-Type: application/x-www-form-urlencoded Content-Length: 29		
	username=Mario&password=12345		
		Body	

HTTP Methods

GET

Retrieve a resource from the server.

POST

Send a resource to the server.

PUT

Replaces the current representation of the target resource with the request payload.

DELETE

Delete a resource from the server.

HTTP Response

	Status Code			
Protocol Version	HTTP /1.1	200	OK	Status Message
Headers	Date: Thu, 11 Nov 2021 17:13:27 GMT			
	Content-Type: text/html			
	Content-Length: 16			
	Connection: keep-alive			
	<html> ... </html>			
	Body			

HTTP Response Status Code¹

1XX

Informational
Responses

2XX

Successful
Responses

3XX

Redirection
Messages

4XX

Client Error

5XX

Server Error

HTTP Response Status Code¹

1XX

100 *Continue*
101 *Switching Protocols*
102 *Processing*

2XX

Successful
Responses

3XX

Redirection
Messages

4XX

Client Error

5XX

Server Error

HTTP Response Status Code¹

1XX

100 *Continue*
101 *Switching Protocols*
102 *Processing*

2XX

200 *OK*
201 *Created*

3XX

Redirection
Messages

4XX

Client Error

5XX

Server Error

HTTP Response Status Code¹

1XX

100 *Continue*
101 *Switching Protocols*
102 *Processing*

2XX

200 *OK*
201 *Created*

3XX

301 *Moved Permanently*
303 *Found*

4XX

Client Error

5XX

Server Error

HTTP Response Status Code¹

1XX

100 *Continue*
101 *Switching Protocols*
102 *Processing*

2XX

200 *OK*
201 *Created*

3XX

301 *Moved Permanently*
303 *Found*

4XX

400 *Bad Request*
404 *Not Found*
405 *Method Not Allowed*

5XX

Server Error

HTTP Response Status Code¹

1XX

100 *Continue*
101 *Switching Protocols*
102 *Processing*

2XX

200 *OK*
201 *Created*

3XX

301 *Moved Permanently*
303 *Found*

4XX

400 *Bad Request*
404 *Not Found*
405 *Method Not Allowed*

5XX

500 *Internal Server Error*
502 *Bad Gateway*

JSON

JavaScript **O**bject **N**otation

- ✓ Text Format for **storing** and **transporting** data.
- ✓ Easy to **understand**.
- ✓ Self describing.
- ✓ An **object** is an unordered set of **name/value pairs**.

```
{  
    "firstName": "Mario",  
    "lastName": "Rossi",  
    "occupation": "Professor",  
    "courses": [  
        "Course1",  
        "Course2"  
    ]  
}
```


A Real REST API



Get Album

Get Current User's Profile

Get Artist

Follow/Unfollow Playlist

Get Playlist

Create Playlist

A Real Interaction



Some Real Operations

GET https:// api.spotify.com/v1/artists/ {id}

Method Protocol Server Base Path Path Path Parameters

POST https:// api.spotify.com/v1/users/ {user_id}/ playlists

Method Protocol Server Base Path Path Path Parameters

The OpenAPI Specification

- ✔ Defines a **standard** interface to describe REST APIs.
- ✔ **JSON** or **YAML** format.
- ✔ **Highly readable** for both humans and computers.
- ✔ Describes available **endpoints**, accepted **parameters**, **response** formats.
- ✔ <https://editor.swagger.io/>



The OpenAPI Specification

```
"paths": {  
  "/book/{bookId}": { Path  
    "get": { Method  
      "operationId": "getBookById",  
      "parameters": [{ Parameters information (location,  
        "name": "bookId", name, schema, mandatory or not)  
        "description": "The unique identifier of the book in the system.",  
        "in": "path",  
        "required": true,  
        "schema": { "type": "integer", "format": "int64"}  
      }],  
      "responses": {  
        "200": {  
          "content": {  
            "application/json": {  
              "schema": {"$ref": "#/components/schemas/ReadBook"}  
            }  
          }  
        }  
      }  
    }  
  }  
  ...  
}
```

Format of the response

Postman

- ✓ Allows the user to **make requests, inspect responses**.
- ✓ It enables **manual testing** of APIs.
- ✓ Available as **desktop** or **web** application.
- ✓ Download it at: <https://www.postman.com/downloads/>
or try the web version at: <https://web.postman.co/home>



HomeWorkspacesAPI Network

Upgrade

GET localhost:8080/books

No environment

Method

URL

SaveShare

GET

localhost:8080/books

Send

Params

Authorization

Headers (7)

Body

Scripts

Tests

Settings

Cookies

Query Params

Key	Value	Description
Key	Value	Description

Body

Cookies

Headers (5)

Test Results

200 OK

19 ms

321 B

JSON

Pretty

Raw

Preview

Visualize

```
1  [  
2    {  
3      "id": 1,  
4      "title": "Divina Commedia",  
5      "author": "Dante Alighieri",  
6      "price": 33.33  
7    },  
8    {  
9      "id": 2,  
10     "title": "I promessi Sposi",  
11     "author": "Alessandro Manzoni",  
12     "price": 15.52  
13   }  
14 ]
```

Input
Parameter

Response

The Bookstore

Get Books

Get a specific book

Delete a Book

Create a Book

Update a book

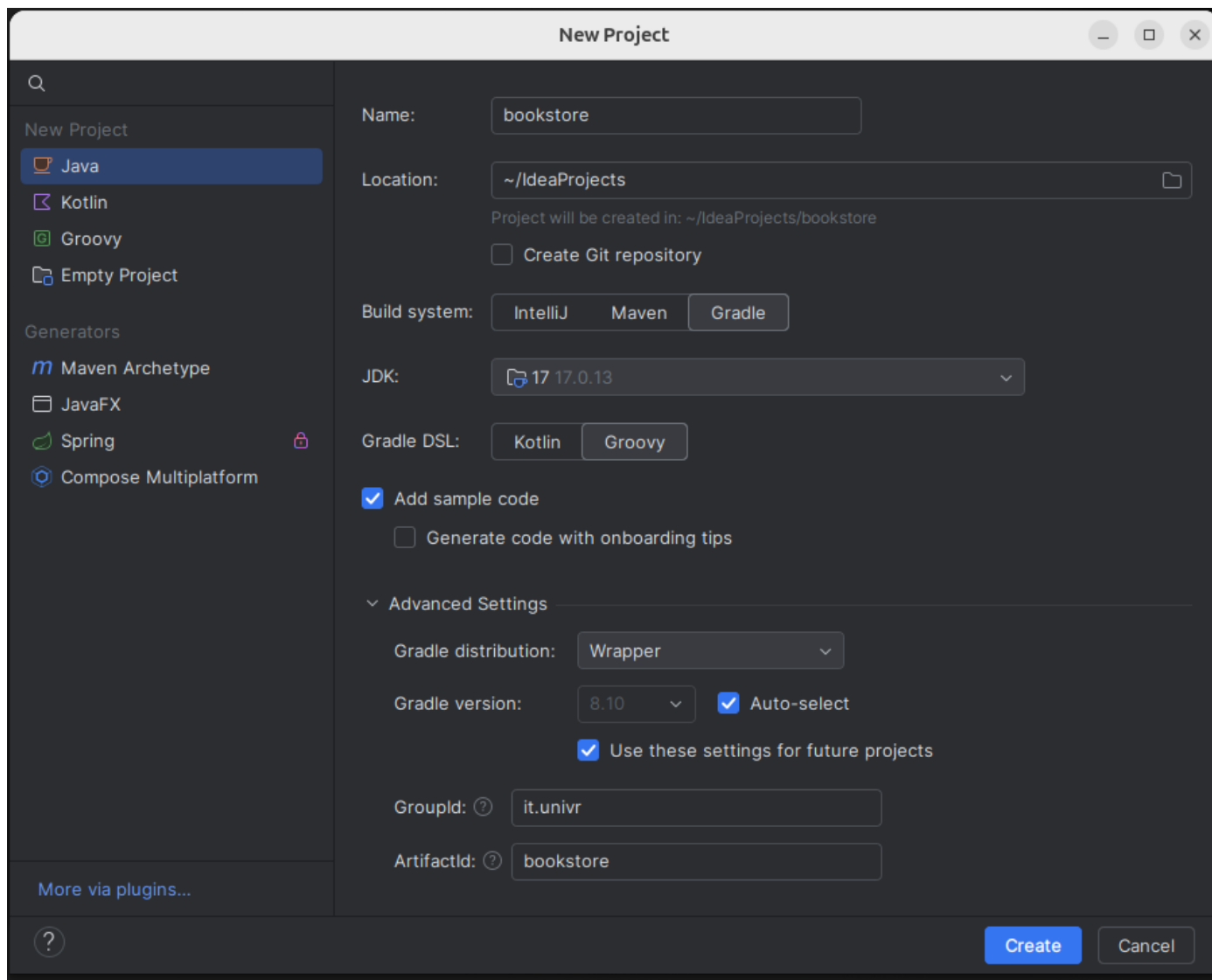


Exercise

- 1) Look at the Bookstore OpenAPI Specification available at: <https://github.com/SofiaMari/Progettazione-Validazione-Sistemi-Software-bookstore>
- 2) Use Postman to interact with the Bookstore and do the following tasks (<https://bookstore-rzkp.onrender.com>):
 - Retrieve the list of all books present.
 - Add a new book with yourself as author.
 - Edit one of the existing books (id from 1 to 10): set yourself as the author and the price equals to 14.25.



Copy the specification and paste into the <https://editor.swagger.io/> editor and look at the GUI.



Building a REST API

Set up a new Gradle Project in IntelliJ.

Use **Java 17** and **Gradle Version 8.10**

build.gradle

```
plugins {  
    id 'java'  
    id 'org.springframework.boot' version '3.4.0'  
    id 'io.spring.dependency-management' version '1.1.3'  
}
```

```
java {  
    sourceCompatibility = JavaVersion.VERSION_17  
    targetCompatibility = JavaVersion.VERSION_17  
}
```

```
apply plugin: 'io.spring.dependency-management'
```

```
group = 'it.univr'  
version = '1.0-SNAPSHOT'
```

```
repositories {  
    mavenCentral()  
}
```

```
dependencies {  
    implementation('org.springframework.boot:spring-boot-starter-web')  
    implementation('org.springframework.boot:spring-boot-starter-data-jpa')  
    runtimeOnly 'com.h2database:h2'  
    testImplementation 'org.springframework.boot:spring-boot-starter-test'  
}
```

```
test {  
    useJUnitPlatform()  
}
```

Add Spring plugins and dependencies.

Building a REST API

- 1) Create the *BookstoreApplication* class that will be used as entrypoint.
- 2) The annotations tell Spring to set up the whole infrastructure.

```
package it.univr;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

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

Building a REST API

- 1) We need to define a **repository** interface.
- 2) We don't have to implement any methods: we will only use methods already available in the superclass.

```
package it.univr.bookstore;

import org.springframework.data.repository.CrudRepository;

public interface BookRepository extends CrudRepository<Book, Long> {

}
```

```
package it.univr;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;
import jakarta.persistence.GenerationType;

@Entity
public class Book {
    @Id @GeneratedValue(strategy=GenerationType.AUTO) private Long id;
    private String title;
    private String author;
    private Float price;

    public Book(String title, String author, Float price){
        this.title = title;
        this.author = author;
        this.price = price;
    }
    . . .
}
```

Building a REST API

Annotations tell Spring that this class is an **Entity** and which of the attributes is the **ID**.

```

package it.univr;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.Optional;

@RestController
public class BookController {

    @Autowired
    private BookRepository bookRepository;

    @GetMapping("/books") @GetMapping maps the GET method
    public Iterable<Book> getBooks() {
        return bookRepository.findAll();
    }

    @GetMapping("/book/{bookId}")
    public Optional<Book> readBook(@PathVariable("bookId") Long id) {
        return bookRepository.findById(id);
    }

    . . .

```

Building a REST API

The controller handles the requests that are coming in.

(1/2)

```
@PostMapping("/book")    @RequestParam gets query parameters
public Book createBook(@RequestParam("title") String title,
                      @RequestParam("author") String author,
                      @RequestParam("price") Float price){

    Book book = new Book(title, author, price);
    bookRepository.save(book);
    return book;
}
```

```
@PutMapping("/book")    @RequestBody gets object in the body
public Book updateBook(@RequestBody Book book){

    bookRepository.save(book);
    return book;
}
```

```
@DeleteMapping("/book")
public void deleteBook(@RequestParam("id") Long id){

    bookRepository.deleteById(id);
}

}
```

Building a REST API

The controller handles the requests that are coming in.

(2/2)

Exercise

- 1) Download the source code of the Book Store from <https://github.com/SofiaMari/Progettazione-Validazione-Sistemi-Software-bookstore>.
- 2) Implements 4 new operations in the Bookstore REST API:
 - Search books by title.
 - Search books by author name.
 - PRO: Search books by author name and title (simultaneously).
 - PRO: Search for books priced between a minimum and a maximum value.