REST APIs

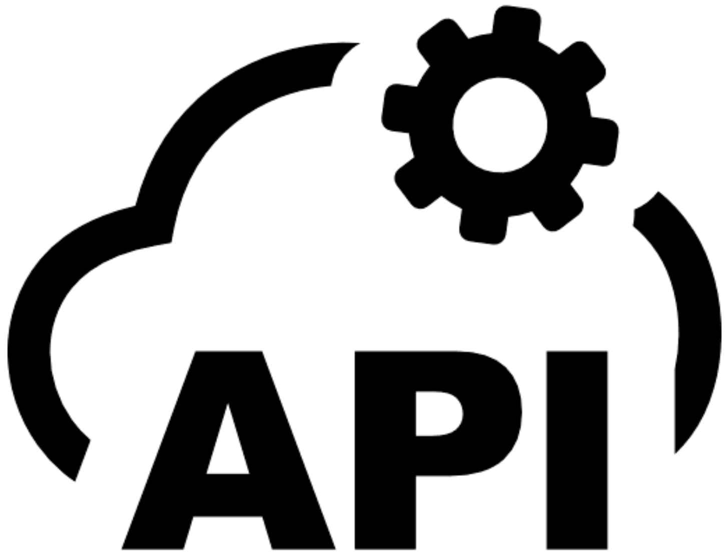
Part 1

MARIANO CECCATO (mariano.ceccato@univr.it)

SOFIA MARI (sofia.mari@univr.it)

Table of contents

REST APIs



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

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

REST APIs

- Uniform Interface
- Client-Server
- Stateless
- Cacheable
- Layered System
- Code on-demand

HTTP Protocol

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

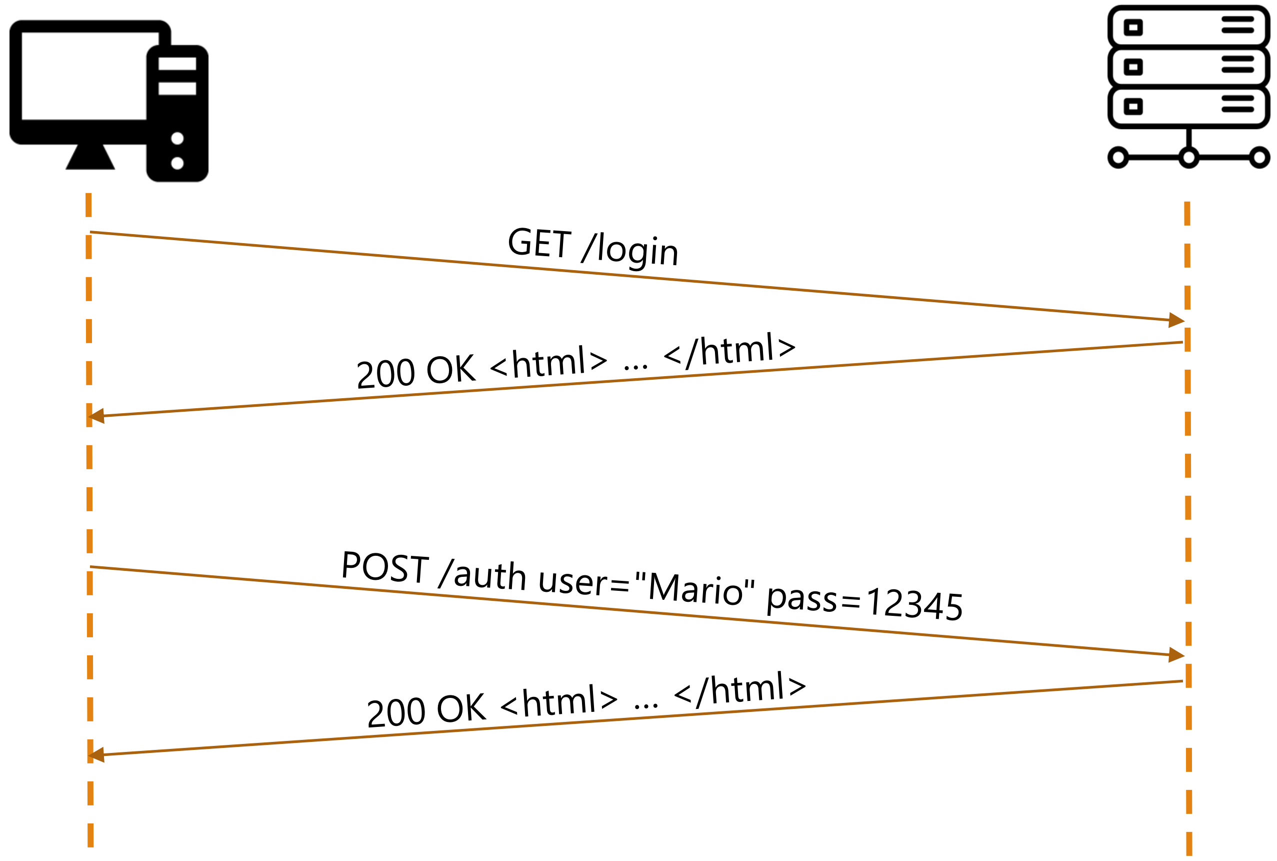
HTTP

Application Layer

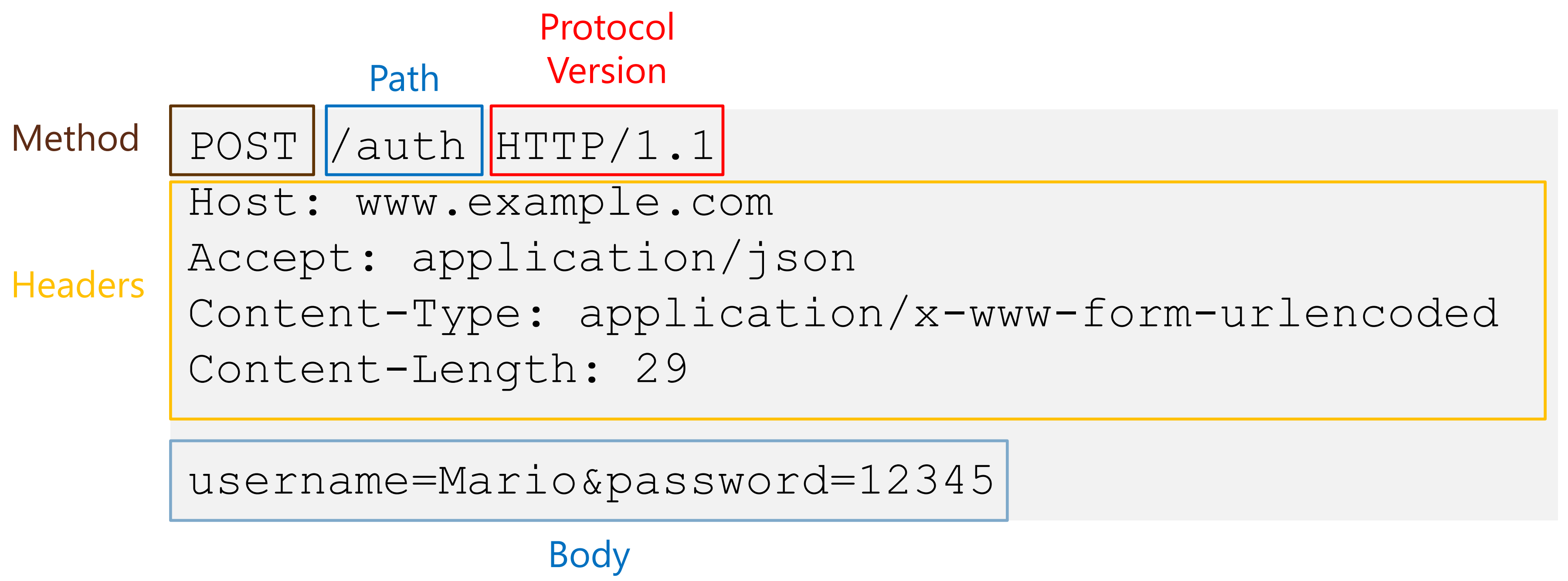
Client-Server

Stateless

HTTP Interaction


GET /login
200 OK <html> ... </html>
POST /auth user="Mario" pass=12345
200 OK <html> ... </html>

HTTP Request



```
POST /auth HTTP/1.1
Host: www.example.com
Accept: application/json
Content-Type: application/x-www-form-urlencoded
Content-Length: 29

username=Mario&password=12345
```

HTTP Methods

GET

Retrieve a resource from the server.

POST

Send a resource to the server.

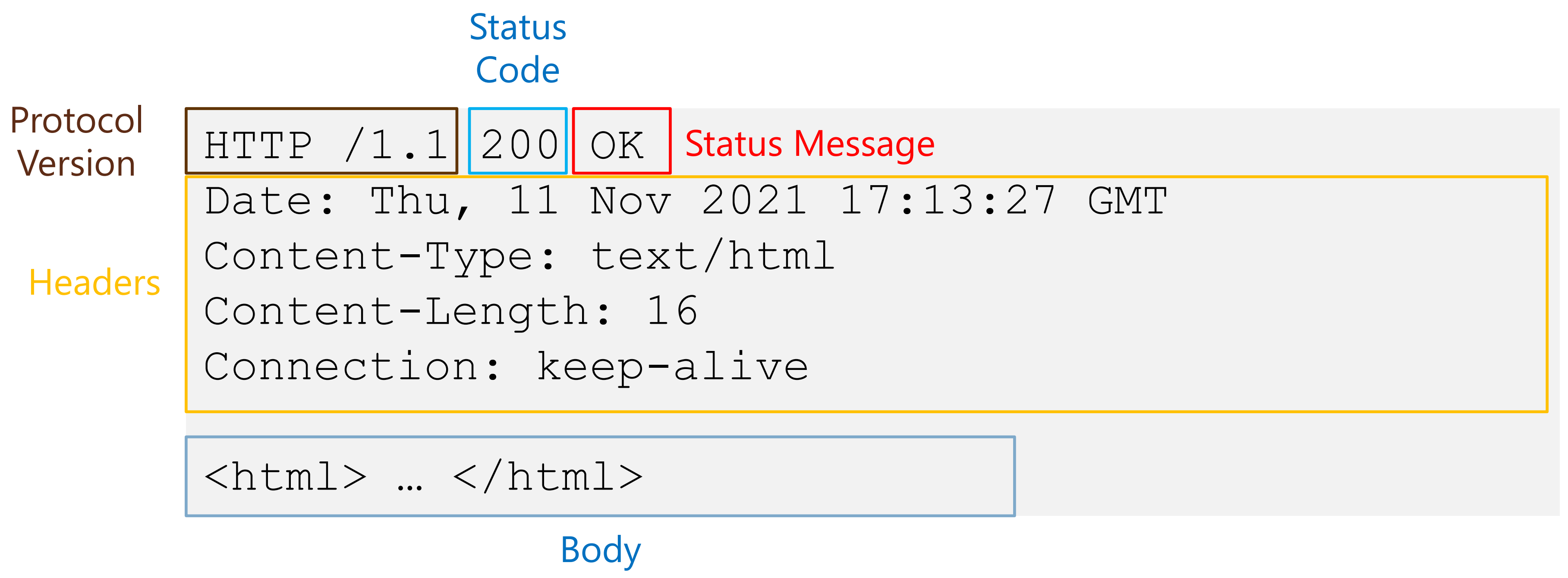
PUT

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

DELETE

Delete a resource from the server.

HTTP Response



```
HTTP /1.1 200 OK
Date: Thu, 11 Nov 2021 17:13:27 GMT
Content-Type: text/html
Content-Length: 16
Connection: keep-alive

<html> … </html>
```

HTTP Response Status Code[1]

1XX

Informational Responses

2XX

Succesful Responses

3XX

Redirection Messages

4XX

Client Error

5XX

Server Error

https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

HTTP Response Status Code[1]

1XX

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

2XX

Succesful Responses

3XX

Redirection Messages

4XX

Client Error

5XX

Server Error

HTTP Response Status Code[1]

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 Code[1]

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 Code[1]

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

https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

HTTP Response Status Code[1]

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

Java**S**cript **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

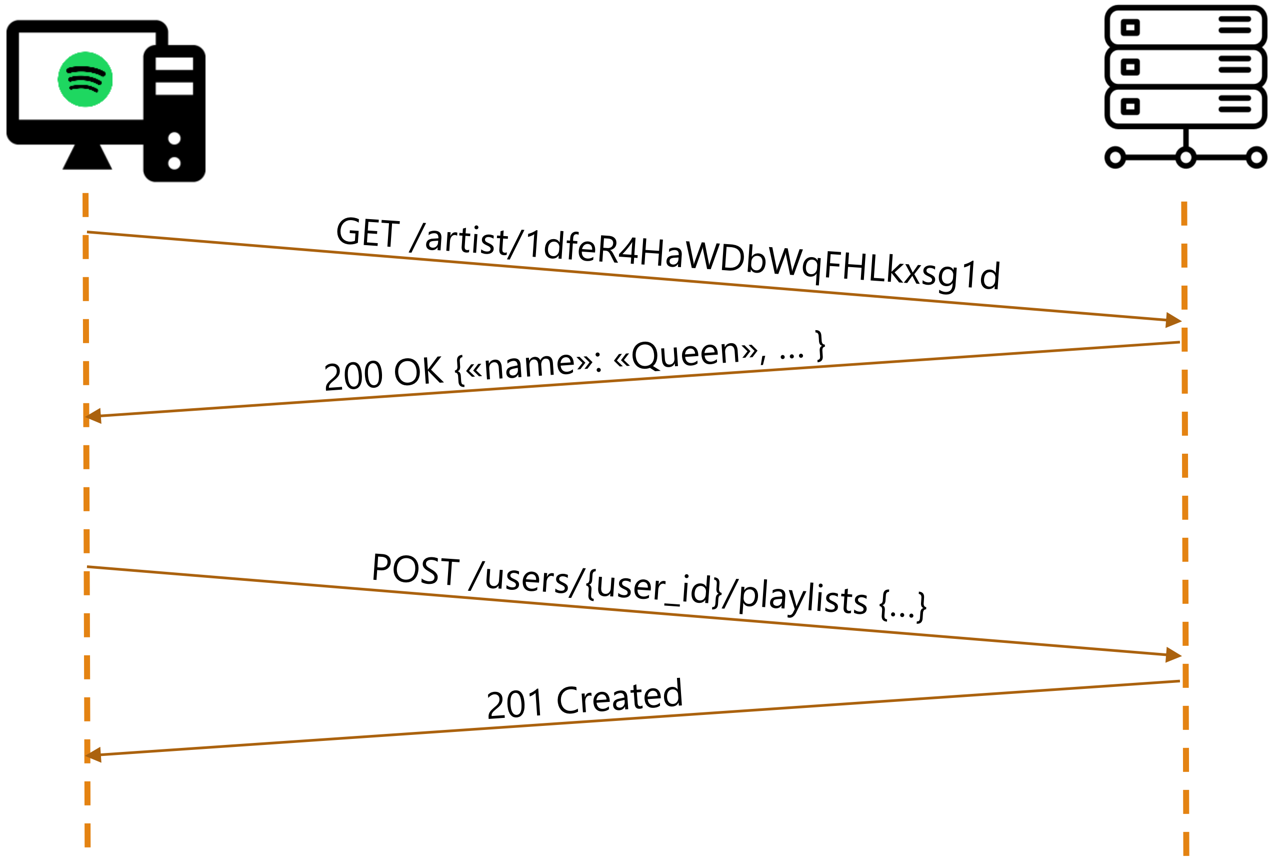
Get Artist

Follow/Unfollow Playlist

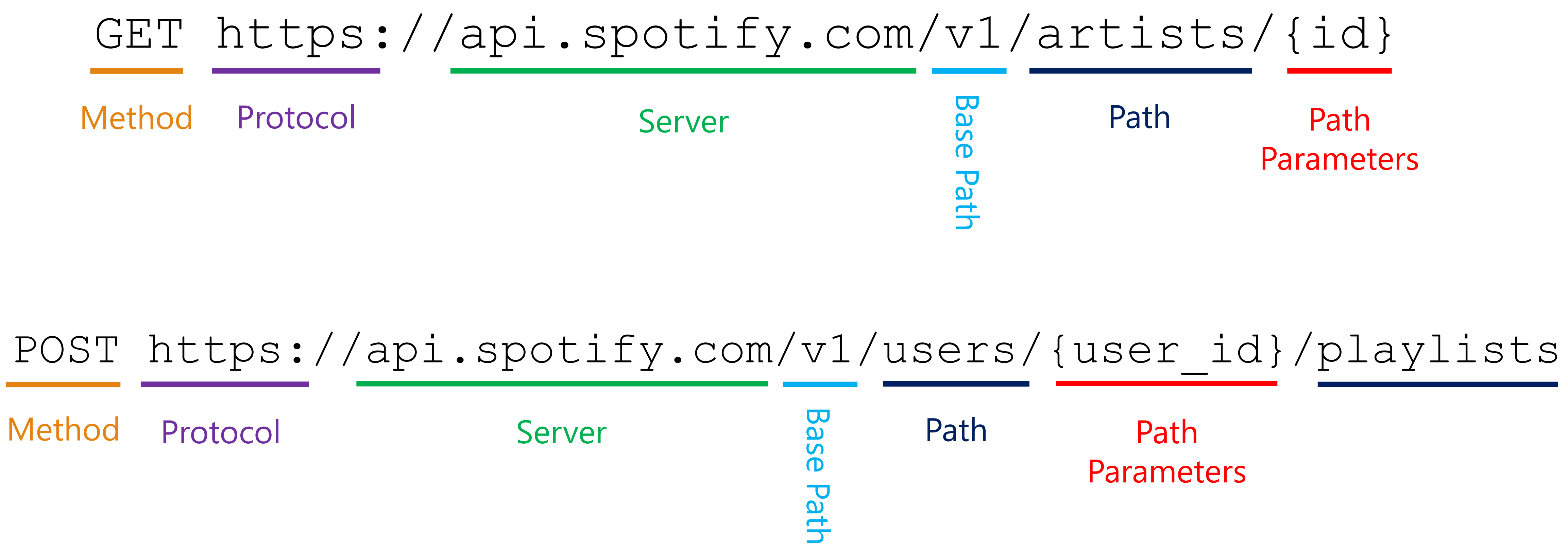
Get Playlist

Create Playlist

A Real Interaction


GET /artist/1dfeR4HaWDbWqFHLkxsg1d
200 OK {«name»: «Queen», ... }
POST /users/{user_id}/playlists {...}
201 Created

Some Real Operations



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

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

https://developer.spotify.com/documentation/web-api

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

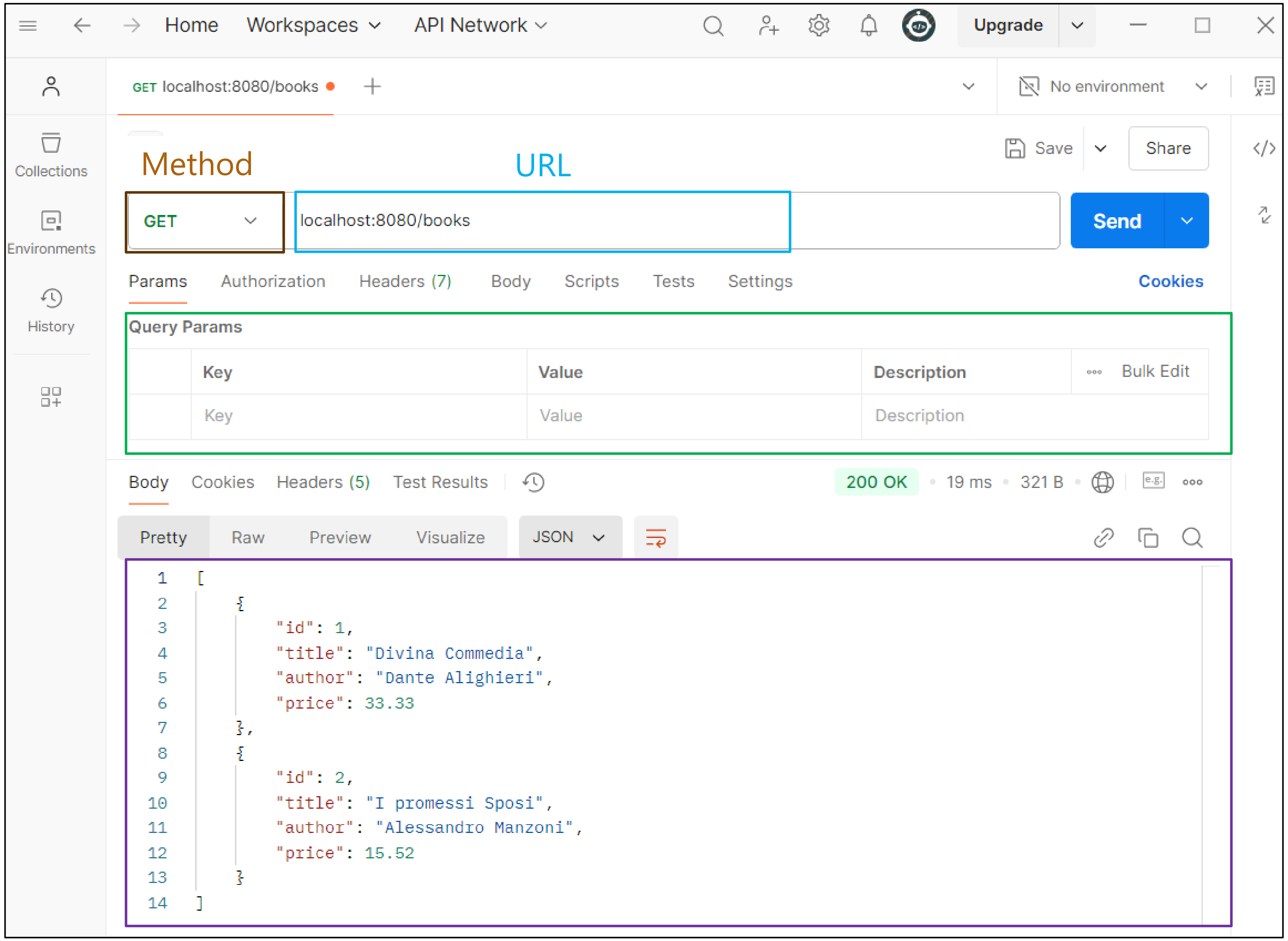
```
"paths": {
     "/book/{bookId}": { Path
Method    "get": {
            "operationId": "getBookById",
            "parameters": [{
                 "name": "bookId",
                 "description": "The unique identifier of the book in the system.",
                 "in": "path",
                 "required": true,
                 "schema": { "type": "integer", "format": "int64"}
                 }],
            "responses": {
                 "200": {
                      "content": {
                            "application/json": {
                               "schema": {"$ref": "#/components/schemas/ReadBook"
                            }
                      }
                 }
        ...
```

Parameters information (location, name, schema, mandatory or not)

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



```
[
    {
        "id": 1,
        "title": "Divina Commedia",
        "author": "Dante Alighieri",
        "price": 33.33
    },
    {
        "id": 2,
        "title": "I promessi Sposi",
        "author": "Alessandro Manzoni",
        "price": 15.52
    }
]
```

Input Parameter

Response

The Bookstore

Get Books

Create a Book

Get a specific book

Update a book

Delete a Book

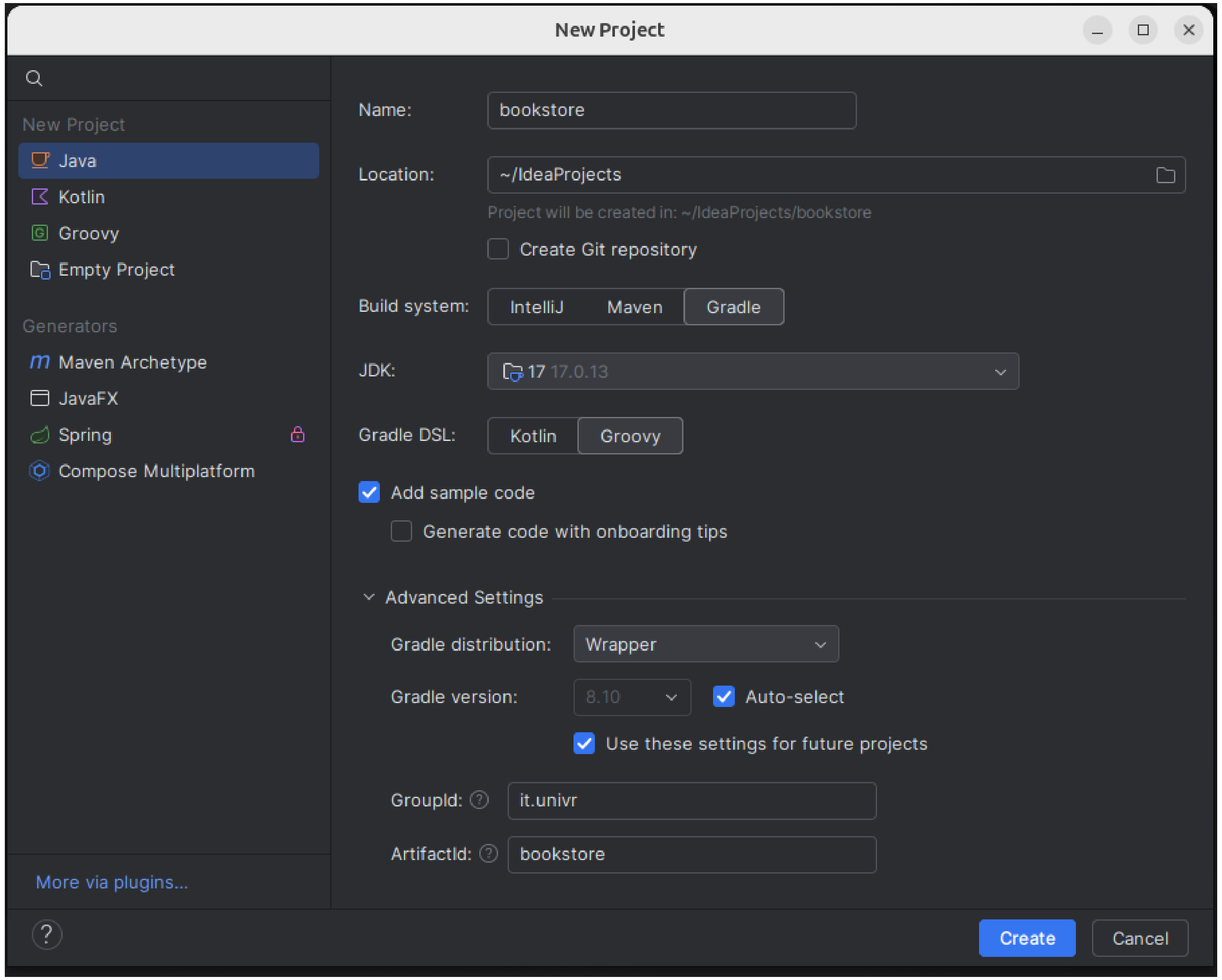
Exercise

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-labpart1.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.

HINT

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


New Project

New Project
- Java
- Kotlin
- Groovy
- Empty Project

Generators
- Maven Archetype
- JavaFX
- Spring
- Compose Multiplatform

More via plugins...

Name: bookstore

Location: ~/IdeaProjects

Project will be created in: ~/IdeaProjects/bookstore

Create Git repository

Build system: IntelliJ Maven [✓ Gradle]

JDK: 17 17.0.13

Gradle DSL: Kotlin [✓ Groovy]

[✓] Add sample code

Generate code with onboarding tips

Advanced Settings

Gradle distribution: Wrapper

Gradle version: 8.10 [✓] Auto-select

[✓] Use these settings for future projects

GroupId: it.univr

ArtifactId: bookstore

Create Cancel

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.

```java
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 methos: 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> {

}
```

Building a REST API

```java
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;
    }
    . . .
}
```

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

Building a REST API

The controller handles the requests that are coming in.

(1/2)

```java
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")
    public Iterable<Book> getBooks(){
        return bookRepository.findAll();
    }

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

`@GetMapping` maps the GET method

`@PathVariable` gets path parameters

Building a REST API

The controller handles the requests that are coming in.

(2/2)

@RequestParam gets query parameters

@RequestBody gets object in the body

```
    @PostMapping("/book")
    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")
    public Book updateBook(@RequestBody Book book){
        bookRepository.save(book);
        return book;
    }

    @DeleteMapping("/book")
    public void deleteBook(@RequestParam("id") Long id){
        bookRepository.deleteById(id);
    }
}
```

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.