Xideral

Back End Academy

REST API

Project By:

Kenneth Rodríguez García

REST API

Summary

This project consists of a REST API, built with Java and Spring, which connects with a MySQL database of various video game titles. By building this project, I have learnt the basics of building a REST application, as well as understanding the principle behind the Model-View-Controller pattern.

Research

REST APIs

A REST (which stands for Representational State Transfer) API, is a style of building web services that interact with a client using HTTP requests and responses. This style follows a set of principles to ensure interoperability, scalability, and ease of use:

- Statelessness: Each request is independent and does not rely on other requests or session tokens.
- Client-Server Architecture: The client (a web browser or application) sends a request to the server, which processes it and sends a response back to the client.
- Methods and Formats: APIS most commonly use the HTTP methods **GET**, **POST**, **PUT**, and **DELETE**, and handles data with formats like **XML** or **JSON**.

REST APIs provide many advantages to back end developers. Scalability, for instance, allows multiple instances of an API to handle large volumes of traffic, while the interoperability of this APIS allows for platform-independent development (which means different technologies and programming languages can interact).

The **Endpoints** of an API are the specific routes an API “observes” to handle requests. These endpoints are usually after the domain name, according to the following naming scheme: ***domain.tld/api/endpoint***

Spring Web

Implementing a REST API from scratch is a very interesting project, but also a very complicated one. Fortunately for us, there is no need to “reinvent the wheel”, because there are solutions available for API development, such as **Spring Web**.

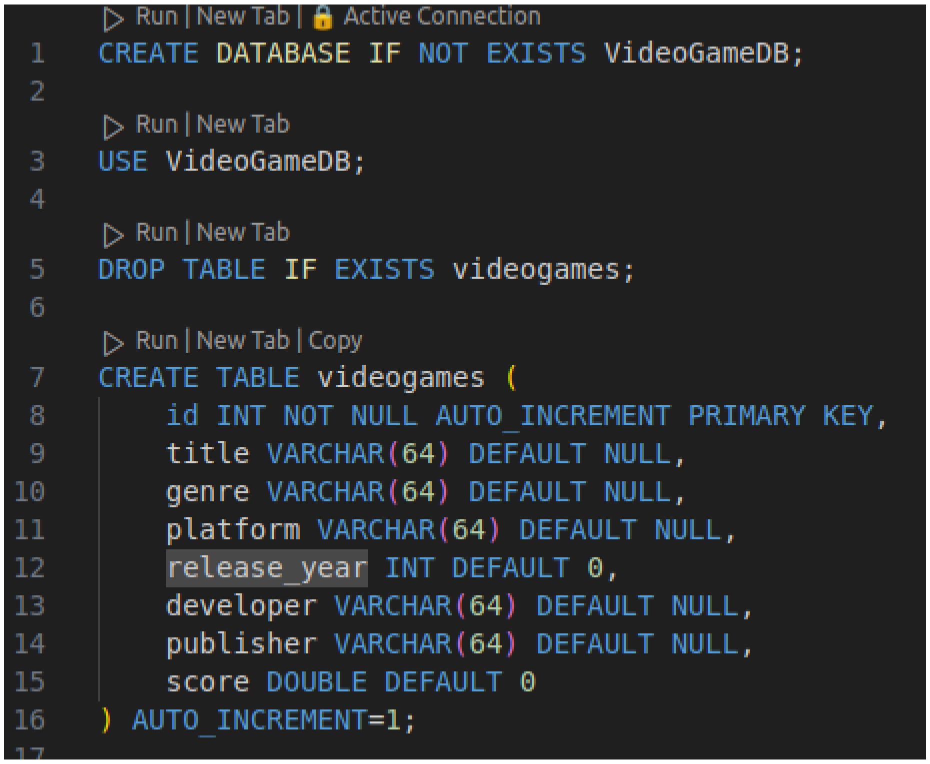
Spring Web is a framework within the Spring ecosystem which simplifies the development of Java web applications, as it handles requests and responses.

This framework uses a Model-View-Controller, or **MVC**, architecture, which means that data is not transferred directly from the server to the client and vice-versa. Instead, data is mapped according to the requests, which allows developers to change some components without affecting the others.

By taking full advantage of this framework, development of our API will be much easier.

Design

For our database, we will build a very simple table containing video game data, such as the title, genre or release year.



```sql
CREATE DATABASE IF NOT EXISTS VideoGameDB;

USE VideoGameDB;

DROP TABLE IF EXISTS videogames;

CREATE TABLE videogames (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(64) DEFAULT NULL,
    genre VARCHAR(64) DEFAULT NULL,
    platform VARCHAR(64) DEFAULT NULL,
    release_year INT DEFAULT 0,
    developer VARCHAR(64) DEFAULT NULL,
    publisher VARCHAR(64) DEFAULT NULL,
    score DOUBLE DEFAULT 0
) AUTO_INCREMENT=1;
```

The classes in the project fall into 4 categories: Entity, Data Access Objects, Services, and a Rest Controller.

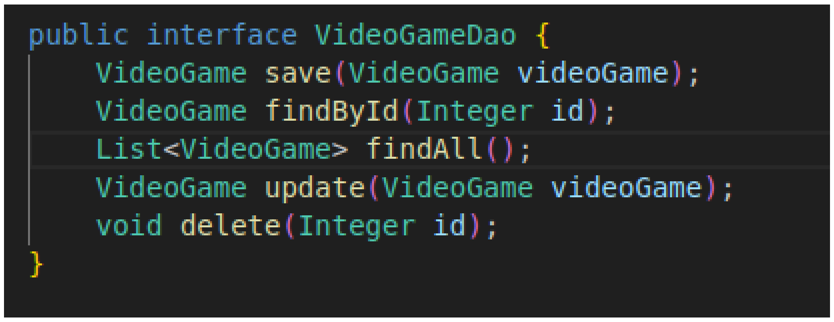
Entity - VideoGame

The VideoGame class is the entity, which means it is a single object where data is encapsulated. It is used to properly map the data to and from the database.

Data Access Objects - VideoGameDAO and VIdeoGameDAOImpl

These classes provide an abstraction layer between the logic (methods) and data in our application. By serving as a middle-layer, Data Access Objects, or DAOs, help to develop more modular and maintainable code bases.

The interface VideoGameDAO provides the different method definitions, such as saving, updating, or finding a VideoGame object.



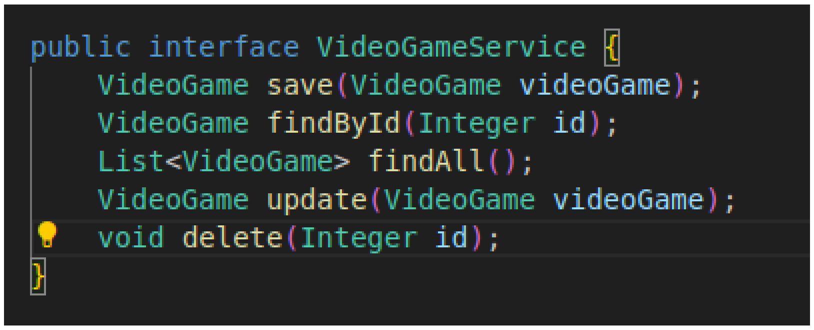
```java
public interface VideoGameDao {
    VideoGame save(VideoGame videoGame);
    VideoGame findById(Integer id);
    List<VideoGame> findAll();
    VideoGame update(VideoGame videoGame);
    void delete(Integer id);
}
```

On the other hand, the VideoGameDAOImpl class, as its name suggests, provides the method implementations. This class also uses Jakarta to provide persistence to the program and transactional methods, preventing database updates from taking place in case of an error.

Service - VideoGameService and VideoGameServiceImpl

These classes are analogous to the previous DAOs, as they are also an intermediate layer between the logic and data. The services use the DAOs to provide a service, which will be consumed by the **Rest Controller.**

The VideoGameService interface only provides the method definitions to the service, while VideoGameServiceImpl implements these methods.

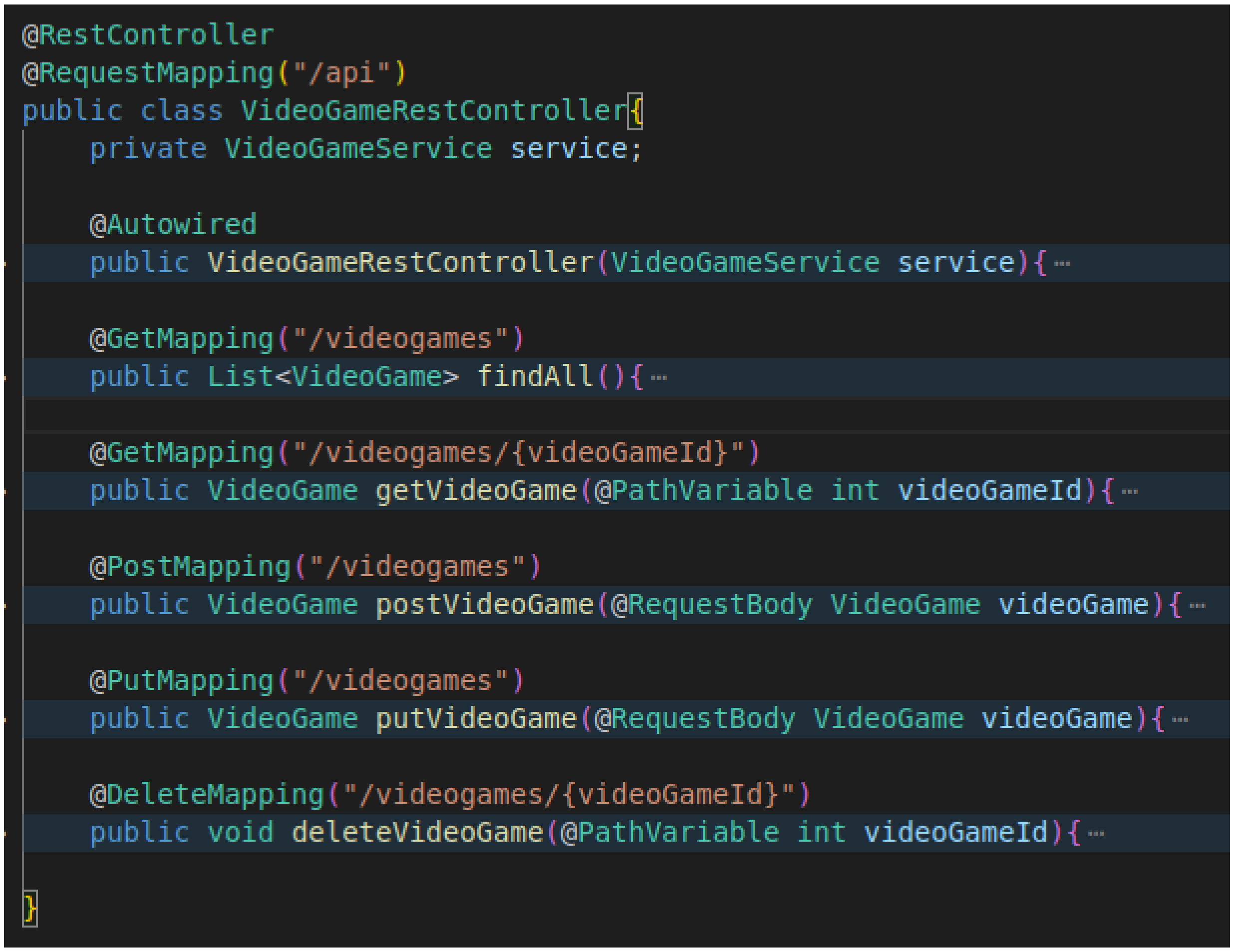


```java
public interface VideoGameService {
    VideoGame save(VideoGame videoGame);
    VideoGame findById(Integer id);
    List<VideoGame> findAll();
    VideoGame update(VideoGame videoGame);
    void delete(Integer id);
}
```

Rest Controller - VideoGameRestController

The Rest Controller is the class that handles the HTTP requests received by the API. It defines the routes used by the API, as well as each of the methods associated with it.

The API route is “/api”, but these by itself won’t consume any service. The proper use is to use one of the associated routes, such as “/api/videogames”.



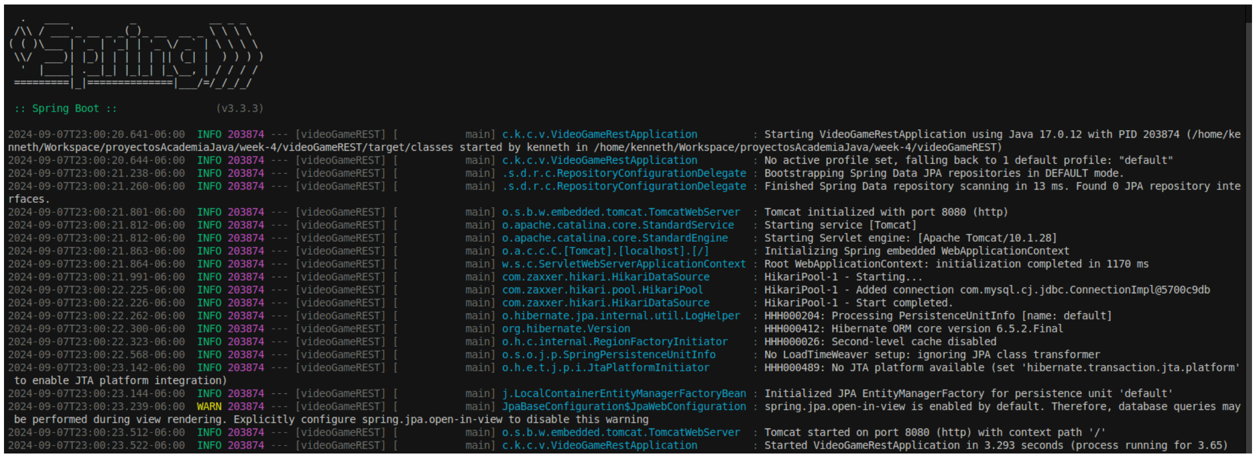
```java
@RestController
@RequestMapping("/api")
public class VideoGameRestController{
    private VideoGameService service;

    @Autowired
    public VideoGameRestController(VideoGameService service){⋯

    @GetMapping("/videogames")
    public List<VideoGame> findAll(){⋯

    @GetMapping("/videogames/{videoGameId}")
    public VideoGame getVideoGame(@PathVariable int videoGameId){⋯

    @PostMapping("/videogames")
    public VideoGame postVideoGame(@RequestBody VideoGame videoGame){⋯

    @PutMapping("/videogames")
    public VideoGame putVideoGame(@RequestBody VideoGame videoGame){⋯

    @DeleteMapping("/videogames/{videoGameId}")
    public void deleteVideoGame(@PathVariable int videoGameId){⋯

}
```

The Rest Controller provides 5 different methods

- findAll(): requests the full list of all VideoGames in the database.
- getVideoGame(): this method uses a parameter (retrieved from the path) to search for a video game with the same ID.
- postVideoGame(): this method inserts the data of a VideoGame object into the database
- putVIdeoGame(): this method update the data of a VideoGame object already in the database. To identify which entry must be updated, the ID of the object serves as the identifying key.
- deleteVideoGame(): this method retrieves a VideoGame object using the id parameter obtained from the path. If one such object is found, the corresponding entry is removed from the database.

Execution

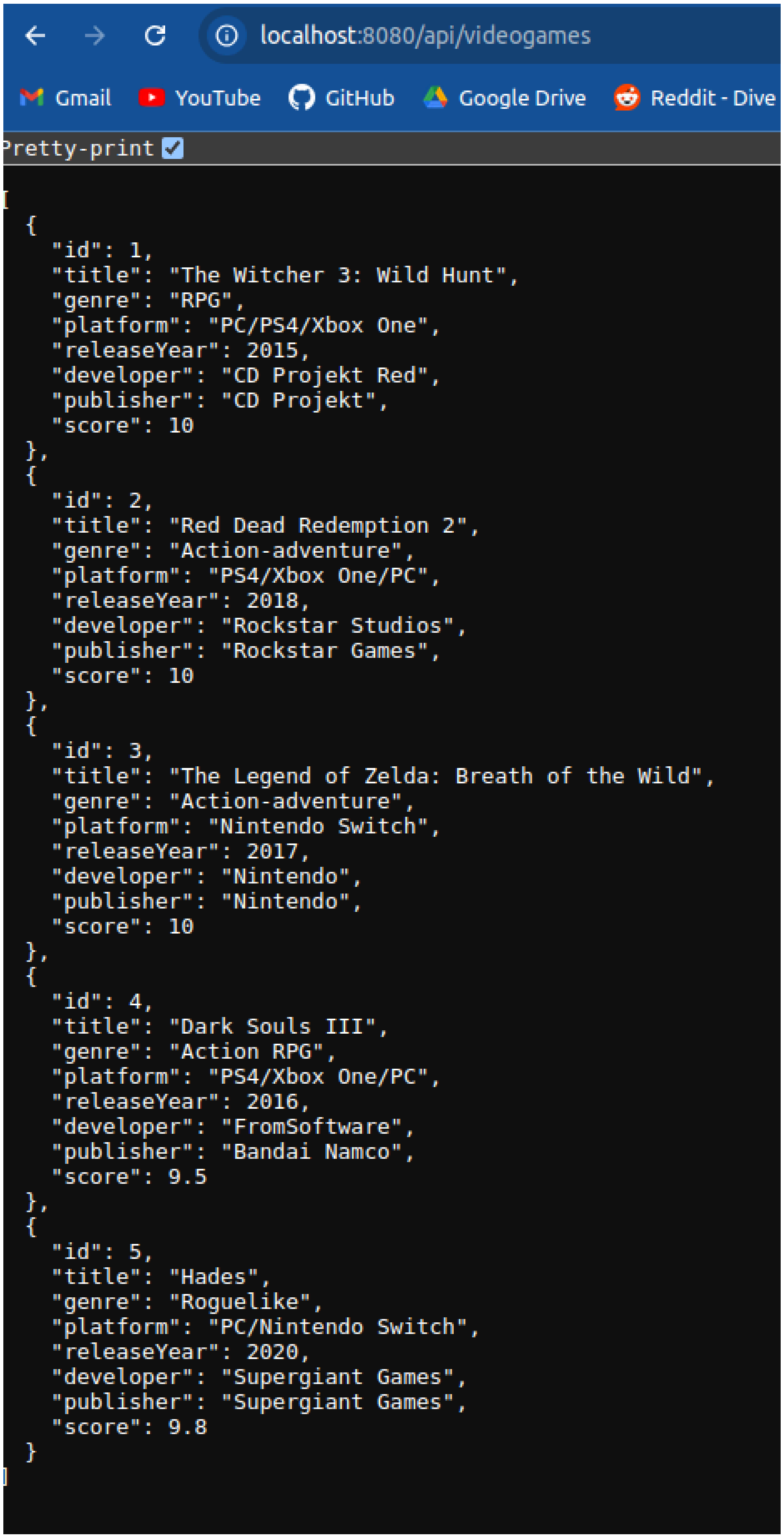
When running the application, we will only be greeted by the Spring banner.



```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/

 :: Spring Boot ::                (v3.3.3)

2024-09-07T23:00:20.641-06:00  INFO 203874 --- [videoGameREST] [           main] c.k.c.v.VideoGameRestApplication         : Starting VideoGameRestApplication using Java 17.0.12 with PID 203874 (/home/ke
nneth/Workspace/proyectosAcademiaJava/week-4/videoGameREST/target/classes started by kenneth in /home/kenneth/Workspace/proyectosAcademiaJava/week-4/videoGameREST)
2024-09-07T23:00:20.644-06:00  INFO 203874 --- [videoGameREST] [           main] c.k.c.v.VideoGameRestApplication         : No active profile set, falling back to 1 default profile: "default"
2024-09-07T23:00:21.238-06:00  INFO 203874 --- [videoGameREST] [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2024-09-07T23:00:21.260-06:00  INFO 203874 --- [videoGameREST] [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 13 ms. Found 0 JPA repository inte
rfaces.
2024-09-07T23:00:21.801-06:00  INFO 203874 --- [videoGameREST] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
2024-09-07T23:00:21.812-06:00  INFO 203874 --- [videoGameREST] [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-09-07T23:00:21.812-06:00  INFO 203874 --- [videoGameREST] [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.28]
2024-09-07T23:00:21.863-06:00  INFO 203874 --- [videoGameREST] [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-09-07T23:00:21.864-06:00  INFO 203874 --- [videoGameREST] [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1170 ms
2024-09-07T23:00:21.991-06:00  INFO 203874 --- [videoGameREST] [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2024-09-07T23:00:22.225-06:00  INFO 203874 --- [videoGameREST] [           main] com.zaxxer.hikari.pool.HikariPool        : HikariPool-1 - Added connection com.mysql.cj.jdbc.ConnectionImpl@5700c9db
2024-09-07T23:00:22.226-06:00  INFO 203874 --- [videoGameREST] [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2024-09-07T23:00:22.262-06:00  INFO 203874 --- [videoGameREST] [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
2024-09-07T23:00:22.300-06:00  INFO 203874 --- [videoGameREST] [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 6.5.2.Final
2024-09-07T23:00:22.323-06:00  INFO 203874 --- [videoGameREST] [           main] o.h.c.internal.RegionFactoryInitiator    : HHH000026: Second-level cache disabled
2024-09-07T23:00:22.568-06:00  INFO 203874 --- [videoGameREST] [           main] o.s.o.j.p.SpringPersistenceUnitInfo      : No LoadTimeWeaver setup: ignoring JPA class transformer
2024-09-07T23:00:23.142-06:00  INFO 203874 --- [videoGameREST] [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000489: No JTA platform available (set 'hibernate.transaction.jta.platform'
 to enable JTA platform integration)
2024-09-07T23:00:23.144-06:00  INFO 203874 --- [videoGameREST] [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2024-09-07T23:00:23.239-06:00  WARN 203874 --- [videoGameREST] [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may
 be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2024-09-07T23:00:23.512-06:00  INFO 203874 --- [videoGameREST] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path '/'
2024-09-07T23:00:23.522-06:00  INFO 203874 --- [videoGameREST] [           main] c.k.c.v.VideoGameRestApplication         : Started VideoGameRestApplication in 3.293 seconds (process running for 3.65)
```

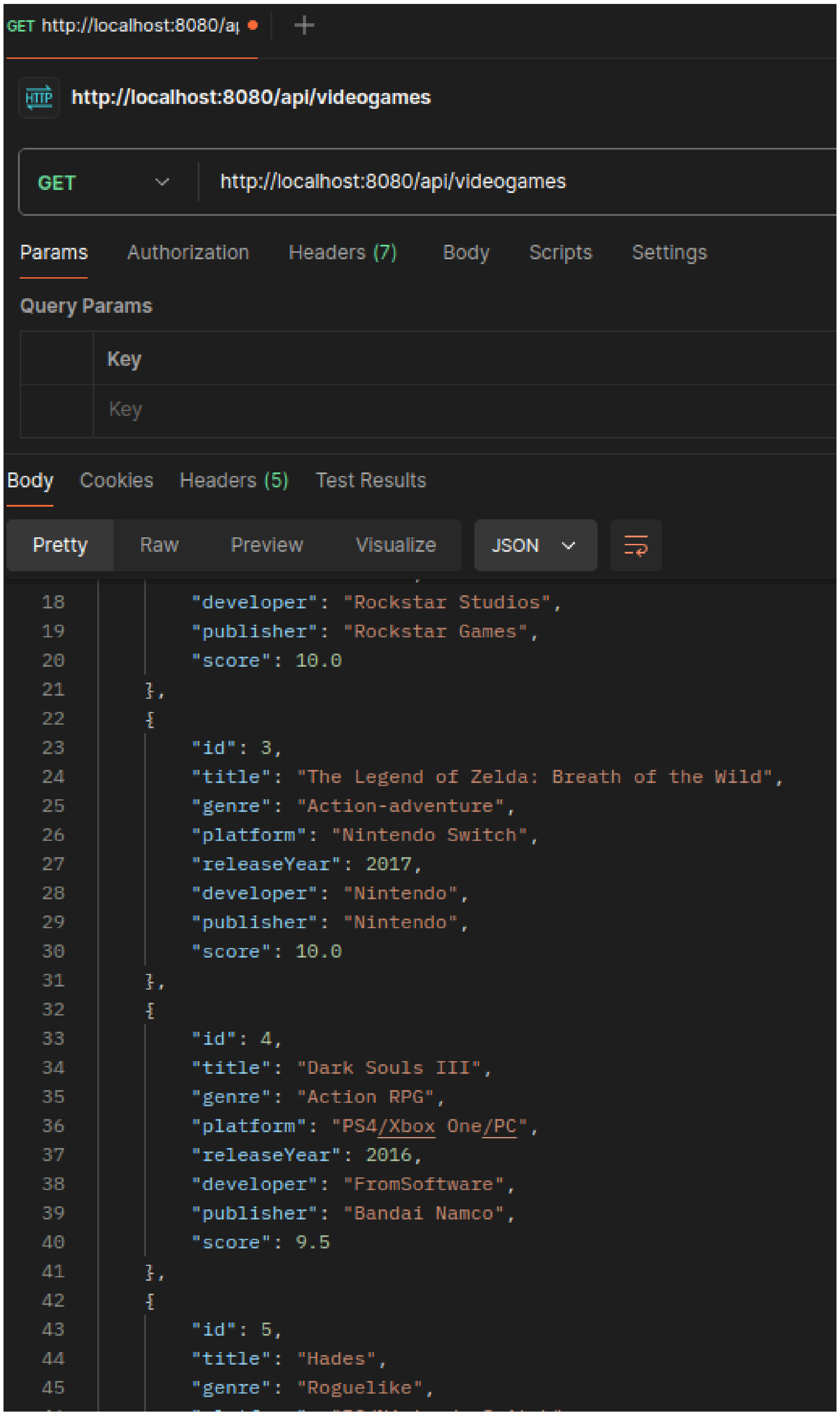
To properly consume the API, we must use another program. In a production environment, our API would run on a dedicated server or container, and apps all over the world would consume it. Instead, we can use other tools to consume the application locally.

By following the link localhost:8080/api/videogames, we can observe the requested data on the browser.

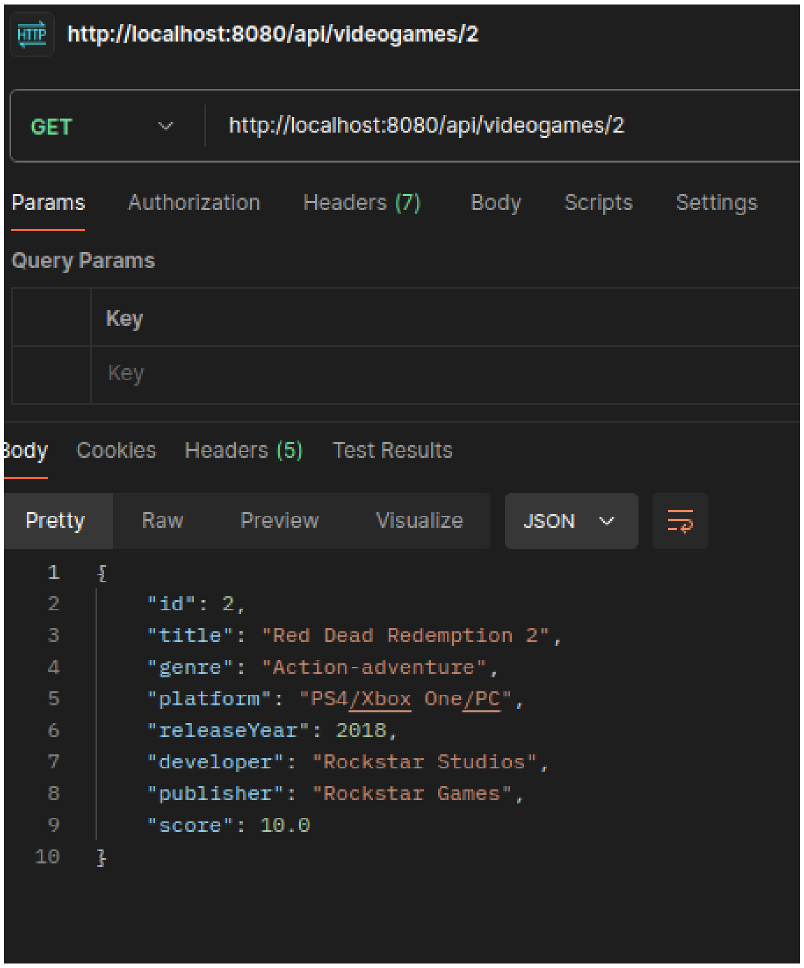


```
[
  {
    "id": 1,
    "title": "The Witcher 3: Wild Hunt",
    "genre": "RPG",
    "platform": "PC/PS4/Xbox One",
    "releaseYear": 2015,
    "developer": "CD Projekt Red",
    "publisher": "CD Projekt",
    "score": 10
  },
  {
    "id": 2,
    "title": "Red Dead Redemption 2",
    "genre": "Action-adventure",
    "platform": "PS4/Xbox One/PC",
    "releaseYear": 2018,
    "developer": "Rockstar Studios",
    "publisher": "Rockstar Games",
    "score": 10
  },
  {
    "id": 3,
    "title": "The Legend of Zelda: Breath of the Wild",
    "genre": "Action-adventure",
    "platform": "Nintendo Switch",
    "releaseYear": 2017,
    "developer": "Nintendo",
    "publisher": "Nintendo",
    "score": 10
  },
  {
    "id": 4,
    "title": "Dark Souls III",
    "genre": "Action RPG",
    "platform": "PS4/Xbox One/PC",
    "releaseYear": 2016,
    "developer": "FromSoftware",
    "publisher": "Bandai Namco",
    "score": 9.5
  },
  {
    "id": 5,
    "title": "Hades",
    "genre": "Roguelike",
    "platform": "PC/Nintendo Switch",
    "releaseYear": 2020,
    "developer": "Supergiant Games",
    "publisher": "Supergiant Games",
    "score": 9.8
  }
]
```

This is still not very practical, because we can only execute GET requests. Instead, we can use tools such as **Postman**, which will allow us to test the API locally.

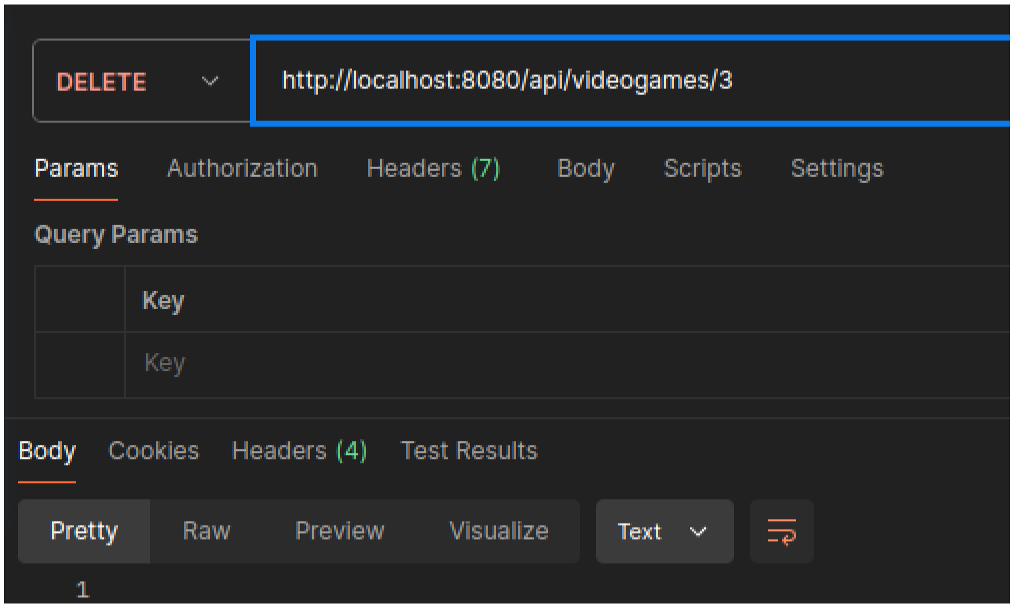


```
            "developer": "Rockstar Studios",
            "publisher": "Rockstar Games",
            "score": 10.0
        },
        {
            "id": 3,
            "title": "The Legend of Zelda: Breath of the Wild",
            "genre": "Action-adventure",
            "platform": "Nintendo Switch",
            "releaseYear": 2017,
            "developer": "Nintendo",
            "publisher": "Nintendo",
            "score": 10.0
        },
        {
            "id": 4,
            "title": "Dark Souls III",
            "genre": "Action RPG",
            "platform": "PS4/Xbox One/PC",
            "releaseYear": 2016,
            "developer": "FromSoftware",
            "publisher": "Bandai Namco",
            "score": 9.5
        },
        {
            "id": 5,
            "title": "Hades",
            "genre": "Roguelike",
```


http://localhost:8080/api/videogames/2

GET http://localhost:8080/api/videogames/2

```
{
    "id": 2,
    "title": "Red Dead Redemption 2",
    "genre": "Action-adventure",
    "platform": "PS4/Xbox One/PC",
    "releaseYear": 2018,
    "developer": "Rockstar Studios",
    "publisher": "Rockstar Games",
    "score": 10.0
}
```


DELETE http://localhost:8080/api/videogames/3

By thoroughly testing our API we can determine that all endpoints and methods execute the desired and intended behavior.

Conclusion

Having previously worked with Node.js and Express.js, I was already familiar with building APIs (although I had only used them with NoSQL databases).

I found this project as a very rewarding learning experience, because it allowed me to relearn concepts I was already familiar with, and take them even further.

There are still a few libraries, like the Jakarta Persistence API, which are a bit like “black boxes” to me, so I will try to learn more about them in the near future.