|  |
| --- |
| ATV Informatica |
| CRUD Restful Web API Pagamenti Digitali |
| Prototipo di esercitazione |

|  |
| --- |
| Vito Perrotta  27/01/2025 |

1. Abstract

Questo progetto illustra la modalità di gestione di Pagamenti Digitali in ambiente Java (17) Spring Boot in modalità Restful nelle operazioni princilali (CRUD) utilizzando un db MySql.

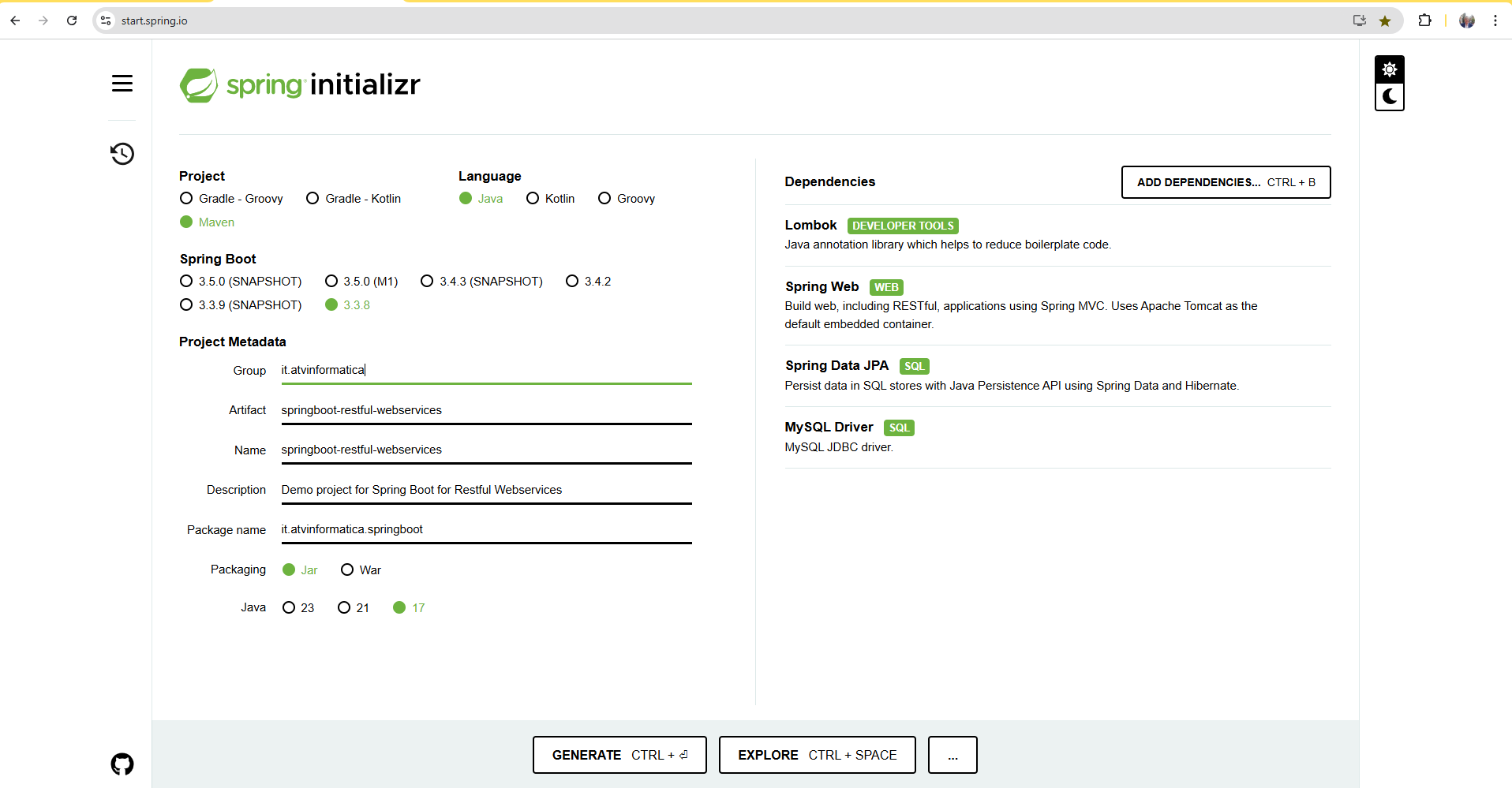
1. Prerequisiti del progetto

* Un istanza di MySQL a cui accedere con Uder e Password

1. Struttura del progetto

Il progetto inzialmente può essere creato con l’utility **Spring initializr** **https://start.spring.io/** per referenziare ed includere le librerie necessarie al funzionamento del prototipo. Che sono :

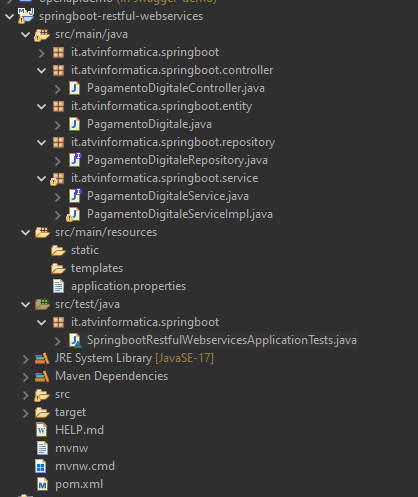
* Spring Web
* Spring Data JPA
* MySql Driver
* Lombok (Opzionale)



Successivamente al progetto sono aggiunti i folders principali dell’applicazione quali

* Controller
* Entity
* Repository
* Service

In modo da avere la seguente struttura :



1. Operazioni sul DB

Sul DB MySQL l’unica operazione è quella di creare il catabase che verrà utilizzato nelle operazioni CRUD : **create database pagamenti\_digitali**

1. Configurazione Spring Boot

Nel file application.properties (sotto resources), vengono definiti i valori di configurazione minimi per accedere al db MySql

spring.application.name=springboot-restful-webservices

spring.datasource.url=jdbc:mysql://localhost:3306/pagamenti\_digitali

spring.datasource.username=<UserName>

spring.datasource.password=<Password>

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

spring.jpa.hibernate.ddl-auto=update

1. Entità

L’unica entity che viene gestita dal prototipo è **PagamentoDigitale**

Nel folder springboot.entity creiamo la classe **PagamentoDigitale**

package it.atvinformatica.springboot.entity;

import java.time.Instant;

import jakarta.persistence.\*;

import lombok.AllArgsConstructor;

*@AllArgsConstructor*

*@Entity*

*@Table*(name = "pagamentiDigitali")

public class PagamentoDigitale {

*@Id*

*@GeneratedValue*(strategy = *GenerationType*.***IDENTITY***)

private Long id;

*@Column*(nullable = false, unique = true)

private float importo;

*@Column*(nullable = false)

private String valuta;

*@Column*(nullable = false)

private String causale;

*@Column*(nullable = false)

private Instant dataOra;

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public float getImporto() {

return importo;

}

public void setImporto(float importo) {

this.importo = importo;

}

public String getValuta() {

return valuta;

}

public void setValuta(String valuta) {

this.valuta = valuta;

}

public String getCausale() {

return causale;

}

public void setCausale(String causale) {

this.causale = causale;

}

public Instant getDataOra() {

return dataOra;

}

public void setDataOra(Instant dataOra) {

this.dataOra = dataOra;

}

}

Che è un POJO utilizzato per mappare l’entità sul DB da Spring JPA

1. Controller

Nella classe **PagamentoDigitaleController** vengono definiti gli endPoint delle API e definita la componente service PagamentoDigitaleService , ‘iniettata’ (DI) tramite annotation **@Autowired**

1. Repository

In questo folder abbiamo definito l’interfaccia PagamentoDigitaleRepository che verrà utilizzata dallla componente service per gestire le operazioni CRUD secondo il modello JPA (JpaRepository)

1. Service

Nel folder service abiamo la definizione di una interfaccia

**PagamentoDigitaleService**

E della classe **PagamentoDigitaleServiceImpl** che la implementa e in cui viene iniettata una istanza di **PagamentoDigitaleRepository** messa a disposizione dal context di Spring Boot.

***@Autowired***

private PagamentoDigitaleRepository pagamentoDigitaleRepository;

1. SWAGGER

Il prototipo, includendo nel pom.xml la dipendenza :

<dependency>

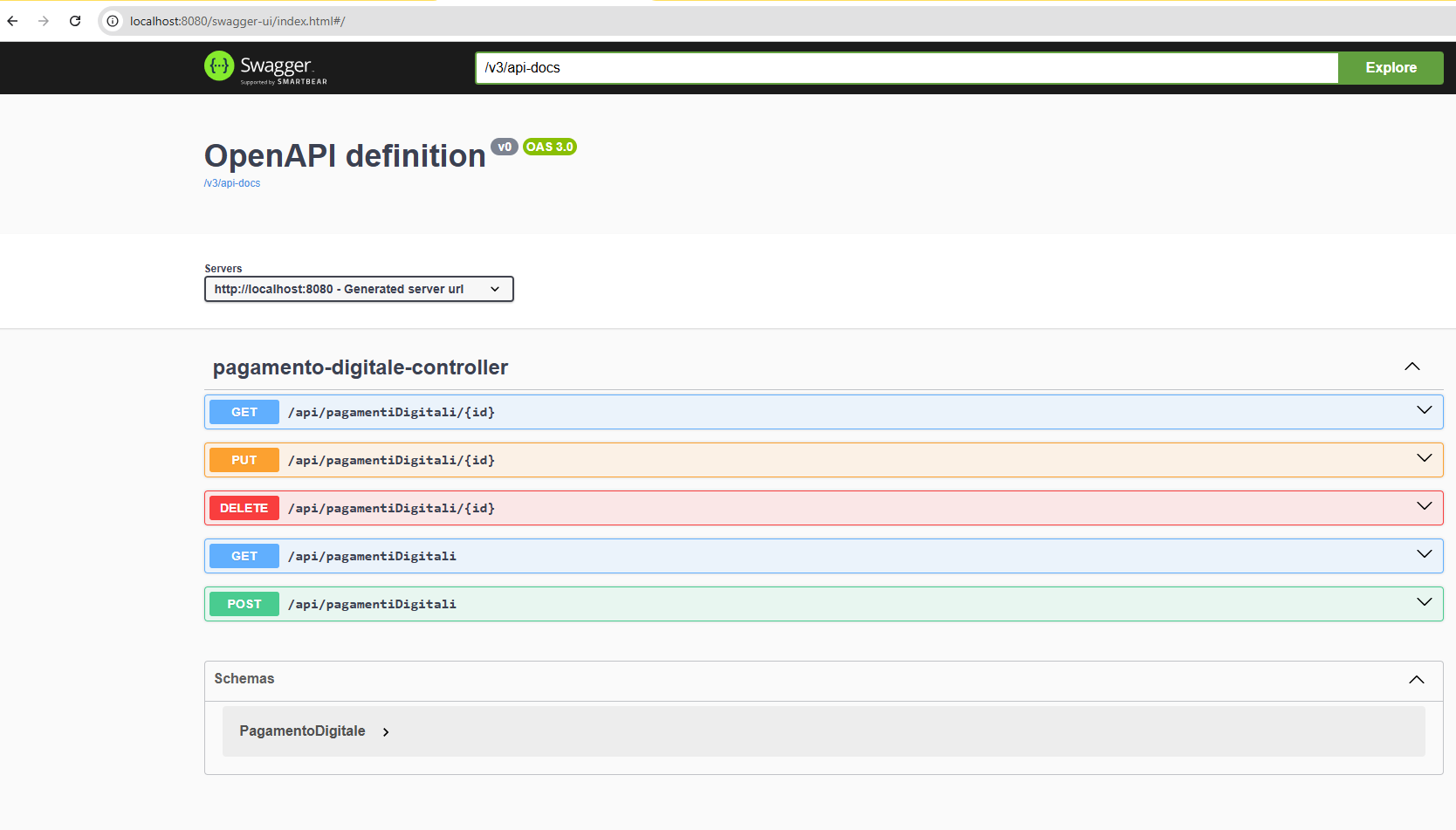
<groupId>org.springdoc</groupId>

<artifactId>**springdoc-openapi-starter-webmvc-ui**</artifactId>

<version>2.6.0</version>

</dependency>

ha la funzionalità dello Swaggher che permette un agevole modalità di chiamata alle operazioni CRUD del prototipo e nel contembo aggiunge, rispetto a Postman, ad esempio, una buona base di documentazione della applicazione stessa.



Lo swagger è fruibile l’url **: http://localhost:8080/swagger-ui/index.html**