Spring Boot Gelateria con salvataggio in PostgreSQL

🏗️ 1. Creazione del progetto da Spring Initializr

Vai su 👉 https://start.spring.io

🔧 Configurazione base:

Campo	Valore
Project	**Maven**
Language	**Java**
Spring Boot	**3.2.x** o superiore
Group	`it.epicode`
Artifact	`gelateria`
Name	`gelateria`
Package Name	`it.epicode.gelateria`
Packaging	**Jar**
Java Version	**17**

📦 Dipendenze da aggiungere:

1. **Spring Web**
2. **Spring Data JPA**
3. **PostgreSQL Driver**
4. **Spring Boot DevTools**
5. **Lombok**

➡️ Premi **"Generate"** per scaricare il `.zip` .

📁 2. Scompattazione del progetto

Estrai il progetto ZIP e aprilo in **IntelliJ IDEA**.

🔧 3. Modifiche a `pom.xml`

Aggiungi **Faker** (manuale):

```xml
<dependency>
    <groupId>com.github.javafaker</groupId>
    <artifactId>javafaker</artifactId>
    <version>1.0.2</version>
</dependency>
```

✅ Ora il tuo `pom.xml` avrà tutte queste dipendenze:

```xml
<dependencies>
    <!-- Spring Boot Starter -->
    <dependency>
```

```xml
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!-- JPA -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <!-- PostgreSQL Driver -->
    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
    </dependency>

    <!-- Lombok -->
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>

    <!-- Faker -->
    <dependency>
        <groupId>com.github.javafaker</groupId>
        <artifactId>javafaker</artifactId>
        <version>1.0.2</version>
    </dependency>

    <!-- DevTools -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
    </dependency>
</dependencies>
```

⚙️ 4. Configurazione `application.properties`

In `src/main/resources/application.properties` :

```
# PostgreSQL Connection
spring.datasource.url=jdbc:postgresql://localhost:5432/gelateria_db
spring.datasource.username=postgres
spring.datasource.password=your_password

# Hibernate / JPA
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true

# Info app
spring.application.name=GelateriaSpringApp
server.port=8081
```

🗂️ 5. Struttura del progetto (consigliata)

```
src/main/java/it/epicode/gelateria/
├── GelateriaApplication.java
├── config/          ← configurazioni future
├── model/           ← entità JPA (Gelato, Bevanda)
├── repository/      ← interfacce JpaRepository
├── service/        ← logica di business
└── runner/         ← CommandLineRunner (per salvataggio automatico)
```

✅ 6. Entità base

📄 `Gelato.java`

```java
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Gelato {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String nome;
    private boolean maxi;
    private double prezzo;
    private int calorie;
}
```

📄 `Bevanda.java`

```java
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Bevanda {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String nome;
    private double prezzo;
    private int calorie;
}
```

📚 7. Repository

```java
@Repository
public interface GelatoRepository extends JpaRepository<Gelato, Long> {}
```

```java
@Repository
public interface BevandaRepository extends JpaRepository<Bevanda, Long> {}
```

🧠 8. Service

```java
@Service
public class MenuService {
    private final GelatoRepository gelatoRepository;
    private final BevandaRepository bevandaRepository;
    private final Faker faker = new Faker();

    public MenuService(GelatoRepository g, BevandaRepository b) {
        this.gelatoRepository = g;
        this.bevandaRepository = b;
    }

    public void salvaProdotti() {
        gelatoRepository.save(new Gelato(null, "Vaniglia", false, 2.5, 200));
        gelatoRepository.save(new Gelato(null, "Cioccolato Maxi", true, 4.0, 350));
        bevandaRepository.save(new Bevanda(null, faker.beer().name(), 1.5, 90));
    }
}
```

🚀 9. Runner

```java
@Component
public class MenuRunner implements CommandLineRunner {

    private final MenuService menuService;

    public MenuRunner(MenuService menuService) {
        this.menuService = menuService;
    }

    @Override
    public void run(String... args) {
        menuService.salvaProdotti();
        System.out.println("Prodotti salvati nel database PostgreSQL.");
    }
}
```