Event Manager API

Struttura del progetto

```
event-manager/
├── src/main/java/it/epicode/
│   ├── config/
│   │   └── FakerConfig.java
│   ├── eventi/
│   │   ├── Evento.java
│   │   ├── EventoController.java
│   │   ├── EventoRepository.java
│   │   └── EventoService.java
│   ├── partecipanti/
│   │   ├── Partecipante.java
│   │   ├── PartecipanteController.java
│   │   ├── PartecipanteRepository.java
│   │   └── PartecipanteService.java
│   └── EventManagerApplication.java
├── src/main/resources/
│   └── application.properties
├── pom.xml
```

Step 1 - Creazione progetto

1. Apri IntelliJ IDEA
2. Vai su `File > New > Project`

3. Scegli **Spring Initializr**
4. Inserisci:
    - Group: `it.epicode`
    - Artifact: `event-manager`
5. Seleziona le dipendenze:
    - Spring Web
    - Spring Data JPA
    - Lombok
    - PostgreSQL Driver
    - Spring Boot DevTools
    - Springdoc OpenAPI UI (aggiunta manuale nel pom.xml)
    - Faker (aggiunta manuale nel pom.xml)
6. Clicca su `Finish`

Step 2 - Configurazione `pom.xml`

Aggiungi le seguenti dipendenze:

```
<dependency>
    <groupId>com.github.javafaker</groupId>
    <artifactId>javafaker</artifactId>
    <version>1.0.2</version>
</dependency>
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
```

```
    <version>2.3.0</version>
</dependency>
```

Step 3 - Configurazione `application.properties`

```
spring.datasource.url=jdbc:postgresql://localhost:5432/event_manager
spring.datasource.username=postgres
spring.datasource.password=your_password_here
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

Step 4 - Creazione entità `Evento` e `Partecipante`

Evento.java

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Evento {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    private String luogo;
```

```
    private LocalDate data;
}
```

Partecipante.java

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Partecipante {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    private String cognome;
    private String email;
    private LocalDate dataDiNascita;
}
```

Step 5 - Creazione Repository, Service e Controller

> Ripetere le stesse strutture per entrambe le entità:

Esempio per Evento

- `EventoRepository` estende `JpaRepository<Evento, Long>`
- `EventoService` contiene i metodi `getAll()`, `getById()`, `create()`, `update()`, `delete()`
- `EventoController` espone gli endpoint REST: `GET`, `POST`, `PUT`, `DELETE` su `/eventi`

Step 6 - Configurazione `Faker`

FakerConfig.java

```
@Configuration
public class FakerConfig {
    @Bean
    public Faker faker() {
        return new Faker(Locale.it));
    }
}
```

Step 7 - Inizializzazione DB con Faker

EventManagerApplication.java

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

    @Bean
    CommandLineRunner init(EventoService eventoService, PartecipanteService partecipanteService, Faker faker) {
        return args -> {
            Random rnd = new Random();

            for (int i = 0; i < 10; i++) {
```

```
                eventoService.create(Evento.builder()
                        .nome("Evento " + faker.book().title())
                        .luogo(faker.address().city())
                        .data(LocalDate.now().plusDays(rnd.nextInt(30)))
                        .build());
            }

            for (int i = 0; i < 10; i++) {
                partecipanteService.create(Partecipante.builder()
                        .nome(faker.name().firstName())
                        .cognome(faker.name().lastName())
                        .email(faker.internet().emailAddress())
                        .dataDiNascita(LocalDate.now().minusYears(18 + rnd.nextInt(20)))
                        .build());
            }

            System.out.println("Database inizializzato con dati di prova.");
        };
    }
}
```

Step 8 - Avvio e Test

1. Avvia il progetto da IntelliJ (Run `EventManagerApplication`)
2. Accedi alla documentazione Swagger:

    `http://localhost:8080/swagger-ui.html`