# Guida alla Creazione di API REST con Spring Boot

## Struttura del Progetto

Di seguito è riportata la struttura del progetto con le relative cartelle e file:

```
shop-animali/
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   ├── it/
│   │   │   │   ├── rpicode/
│   │   │   │   │   ├── shop_animali/
│   │   │   │   │   │   ├── pets/
│   │   │   │   │   │   │   ├── Pet.java
│   │   │   │   │   │   │   ├── PetRequest.java
│   │   │   │   │   │   │   ├── PetResponse.java
│   │   │   │   │   │   │   ├── PetRepository.java
│   │   │   │   │   │   │   ├── PetService.java
│   │   │   │   │   │   │   ├── PetController.java
│   │   │   │   │   │   │   ├── PetRunner.java
│   │   │   │   │   │   ├── responses/
│   │   │   │   │   │   │   ├── CreationRespons.java
│   │   │   │   │   │   ├── configurations/
│   │   │   │   │   │   │   ├── FakerConfig.java
│   ├── resources/
│   │   ├── application.properties
├── pom.xml
```

## Configurazione del Database (PostgreSQL)

Per connettersi a un database PostgreSQL, aggiungere nel file `application.properties` :

```
spring.datasource.url=jdbc:postgresql://localhost:5432/shop_animali
spring.datasource.username=utente
spring.datasource.password=password
spring.jpa.hibernate.ddl-auto=update
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
```

## Dipendenze e loro Utilità ( `pom.xml` )

- **Spring Boot Starter Web**: Contiene le dipendenze necessarie per creare API RESTful.
- **Spring Boot Starter Data JPA**: Fornisce il supporto per la gestione della persistenza dei dati.
- **PostgreSQL Driver**: Permette la connessione al database PostgreSQL.
- **Lombok**: Riduce la necessità di scrivere codice boilerplate come getter e setter.
- **Springdoc OpenAPI**: Fornisce la documentazione automatica delle API tramite Swagger.
- **Java Faker**: Genera dati casuali per i test.

## Creazione delle Componenti

### 1. Creazione dell'Entity ( `Pet.java` )

```
@Entity
@Table(name = "pets")
```

```
public class Pet {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Long id;
    private String name;
    private String type;
    private Integer age;
    private String gender;
    private String description;
}
```

## 2. Creazione del Repository ( `PetRepository.java` )

```
public interface PetRepository extends JpaRepository<Pet, Long> {}
```

## 3. Creazione dei DTO (Data Transfer Object)

I DTO sono fondamentali nelle API REST in quanto permettono di separare i dati trasmessi dalle entità di database, evitando l'esposizione diretta dei modelli persistenti.

### DTO di Richiesta ( `PetRequest.java` )

```
public class PetRequest {
    private String name;
    private String type;
    private Integer age;
    private String gender;
    private String description;
}
```

### DTO di Risposta ( `PetResponse.java` )

```
public class PetResponse {
    private Long id;
    private String name;
    private String type;
}
```

### DTO di Creazione ( `CreationRespons.java` )

```
public class CreationRespons {
    Long id;
}
```

## 4. Creazione del Service ( `PetService.java` )

```
@Service
public class PetService {
    private final PetRepository petRepository;

    public List<Pet> findAll() {
        return petRepository.findAll();
    }

    public CreationRespons save(PetRequest pet) {
        Pet petEntity = new Pet();
        BeanUtils.copyProperties(pet, petEntity);
        petRepository.save(petEntity);
        return new CreationRespons(petEntity.getId());
    }

    public void deleteById(Long id) {
        petRepository.deleteById(id);
    }
```

```
    public PetResponse findById(Long id) {
        Pet pet = petRepository.findById(id).orElseThrow(EntityNotFoundException::new);
        PetResponse response = new PetResponse();
        BeanUtils.copyProperties(pet, response);
        return response;
    }
}
```

## 5. Creazione del Controller ( `PetController.java` )

```
@RestController
@RequestMapping("/api/pets")
public class PetController {
    private final PetService petService;

    @GetMapping
    public List<Pet> findAll() {
        return petService.findAll();
    }

    @PostMapping
    public CreationRespons save(@RequestBody PetRequest pet) {
        return petService.save(pet);
    }
}
```

## 6. Configurazione del Faker ( `FakerConfig.java` )

```
@Configuration
public class FakerConfig {
    @Bean
```

```
    public Faker faker() {
        return new Faker(Locale.ITALIAN);
    }
}
```

### 7. Creazione del Runner per Popolare i Dati ( `PetRunner.java` )

```
@Component
public class PetRunner implements CommandLineRunner {
    private final PetService petService;
    private final Faker faker;

    @Override
    public void run(String... args) {
        for (int i = 0; i < 20; i++) {
            PetRequest pet = new PetRequest();
            pet.setName(faker.animal().name());
            pet.setType(faker.animal().name());
            pet.setAge(faker.number().numberBetween(1, 20));
            pet.setGender("m");
            petService.save(pet);
        }
    }
}
```

## Documentazione con Swagger

Swagger è accessibile tramite:

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

Aggiunge una documentazione interattiva delle API e permette di testarle direttamente dall’interfaccia web.