Esercizio completo: REST API Pets con Spring Boot + Security (JWT)

1) Struttura del progetto

```
shop-animali/
├── pom.xml
├── src/
│   ├── main/
│   │   ├── java/it/rpicode/shop_animali/
│   │   │   ├── auth/            ← cartella copiata da progetto `security`
│   │   │   │   ├── AppUser.java
│   │   │   │   ├── AppUserRepository.java
│   │   │   │   ├── AppUserService.java
│   │   │   │   ├── AuthController.java
│   │   │   │   ├── AuthResponse.java
│   │   │   │   ├── AuthRunner.java
│   │   │   │   ├── CorsConfig.java
│   │   │   │   ├── CustomErrorController.java
│   │   │   │   ├── CustomUserDetailsService.java
│   │   │   │   ├── ExceptionHandlerClass.java
│   │   │   │   ├── JwtAuthenticationEntryPoint.java
│   │   │   │   ├── JwtRequestFilter.java
│   │   │   │   ├── JwtTokenMissingException.java
│   │   │   │   ├── JwtTokenUtil.java
│   │   │   │   ├── LoginRequest.java
│   │   │   │   ├── OpenApiConfig.java
│   │   │   │   ├── RegisterRequest.java
│   │   │   │   ├── Role.java
│   │   │   │   └── SecurityConfig.java
```

```
│   │   │   ├── pets/
│   │   │   │   ├── Pet.java
│   │   │   │   ├── PetRequest.java
│   │   │   │   ├── PetResponse.java
│   │   │   │   ├── CreationResponse.java
│   │   │   │   ├── PetRepository.java
│   │   │   │   ├── PetService.java
│   │   │   │   ├── PetController.java
│   │   │   │   └── PetRunner.java
│   │   └── resources/
│   │       └── application.properties
```

2) `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

# JWT
jwt.secret=G7dLkPzS3LmHqT9UwXyIbV28nHjPeR5ZGTRU77891Yht
jwt.expiration.ms=3600000
```

3) Package `auth` (copiato da IntelliJ, solo le classi da cambiare)

AppUser.java

```java
package it.rpicode.shop_animali.auth;

import jakarta.persistence.*;
import lombok.*;

@Entity @Data @NoArgsConstructor @AllArgsConstructor
public class AppUser {
    @Id @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Long id;
    private String username;
    private String password;
    private String email;    // nuovo campo
    @Enumerated(EnumType.STRING)
    private Role role;
}
```

RegisterRequest.java

```java
package it.rpicode.shop_animali.auth;

import lombok.Data;

@Data
public class RegisterRequest {
    private String username;
    private String password;
    private String email;    // nuovo campo
}
```

AppUserService.java

```java
package it.rpicode.shop_animali.auth;

import lombok.RequiredArgsConstructor;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

@Service @RequiredArgsConstructor
public class AppUserService {
    private final AppUserRepository repo;
    private final PasswordEncoder encoder;
    private final JwtTokenUtil jwtUtil;

    public AuthResponse registerUser(RegisterRequest req) {
        AppUser u = new AppUser();
        u.setUsername(req.getUsername());
        u.setPassword(encoder.encode(req.getPassword()));
        u.setEmail(req.getEmail());            // nuovo campo
        u.setRole(Role.USER);
        repo.save(u);
        String token = jwtUtil.generateToken(u.getUsername());
        return new AuthResponse(token);
    }

    // il resto uguale
}
```

AuthRunner.java (popola un utente demo)

```java
package it.rpicode.shop_animali.auth;

import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;
import lombok.RequiredArgsConstructor;
```

```
@Component @RequiredArgsConstructor
public class AuthRunner implements CommandLineRunner {
    private final AppUserRepository repo;
    private final PasswordEncoder encoder;

    @Override public void run(String... args) throws Exception {
        if (repo.count()==0) {
            // modificato per aggiungere l'email
            AppUser u = new AppUser(null, "admin", encoder.encode("admin"), "admin@demo.com", Role.ADMIN);
            repo.save(u);
        }
    }
}
```

4) Package `pets`

Pet.java

```
package it.rpicode.shop_animali.pets;

import jakarta.persistence.*;
import lombok.*;

@Entity @Data @NoArgsConstructor @AllArgsConstructor
@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;
}
```

PetRequest.java

```
package it.rpicode.shop_animali.pets;

import lombok.Data;

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

PetResponse.java

```
package it.rpicode.shop_animali.pets;

import lombok.Data;

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

CreationResponse.java

```java
package it.rpicode.shop_animali.pets;

import lombok.AllArgsConstructor;
import lombok.Data;

@Data @AllArgsConstructor
public class CreationResponse {
    private Long id;
}
```

PetRepository.java

```java
package it.rpicode.shop_animali.pets;

import org.springframework.data.jpa.repository.JpaRepository;

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

PetService.java

```java
package it.rpicode.shop_animali.pets;

import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.beans.BeanUtils;
import jakarta.persistence.EntityNotFoundException;
import java.util.List;

@Service @RequiredArgsConstructor
```

```
public class PetService {
    private final PetRepository repo;

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

    public PetResponse findById(Long id) {
        Pet p = repo.findById(id).orElseThrow(EntityNotFoundException::new);
        PetResponse res = new PetResponse();
        BeanUtils.copyProperties(p,res);
        return res;
    }

    public CreationResponse save(PetRequest r) {
        Pet p = new Pet();
        BeanUtils.copyProperties(r,p);
        repo.save(p);
        return new CreationResponse(p.getId());
    }

    public PetResponse update(Long id, PetRequest r) {
        Pet p = repo.findById(id).orElseThrow(EntityNotFoundException::new);
        BeanUtils.copyProperties(r,p);
        repo.save(p);
        PetResponse res = new PetResponse();
        BeanUtils.copyProperties(p,res);
        return res;
    }

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

PetController.java

```java
package it.rpicode.shop_animali.pets;

import lombok.RequiredArgsConstructor;
import org.springframework.web.bind.annotation.*;
import org.springframework.security.access.prepost.PreAuthorize;
import java.util.List;

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

    @GetMapping
    @PreAuthorize("isAuthenticated()")
    public List<Pet> findAll() { return svc.findAll(); }

    @GetMapping("/{id}")
    // @PreAuthorize("hasRole('ADMIN')")
    // @PreAuthorize("hasAnyRole('ADMIN','USER')")

    @PreAuthorize("isAuthenticated()")
    public PetResponse findById(@PathVariable Long id) {
        return svc.findById(id);
    }

    @PostMapping
    @PreAuthorize("isAuthenticated()")
    public CreationResponse save(@RequestBody PetRequest r) {
        return svc.save(r);
    }

    @PutMapping("/{id}")
    @PreAuthorize("isAuthenticated()")
    public PetResponse update(@PathVariable Long id, @RequestBody PetRequest r) {
```

```
        return svc.update(id,r);
    }

    @DeleteMapping("/{id}")
    @PreAuthorize("isAuthenticated()")
    public void delete(@PathVariable Long id) {
        svc.deleteById(id);
    }
}
```

PetRunner.java (popola dati)

```
package it.rpicode.shop_animali.pets;

import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;
import lombok.RequiredArgsConstructor;
import com.github.javafaker.Faker;

@Component @RequiredArgsConstructor
public class PetRunner implements CommandLineRunner {
    private final PetService svc;
    private final Faker faker;

    @Override public void run(String... args) {
        for(int i=0;i<20;i++){
            PetRequest r = new PetRequest();
            r.setName(faker.animal().name());
            r.setType(faker.animal().name());
            r.setAge(faker.number().numberBetween(1,20));
            r.setGender(i%2==0?"m":"f");
            r.setDescription("Animale generato");
            svc.save(r);
        }
```

```
    }
}
```

Ecco la tabella aggiornata con `hasRole` e `hasAuthority`, usando i ruoli `USER` e `ADMIN`:

Espressione	Cosa fa	Esempio
`@PreAuthorize("isAuthenticated()")`	qualsiasi utente autenticato	accesso a tutti gli utenti loggati
`@PreAuthorize("hasRole('ADMIN')")`	solo utenti con ruolo **ADMIN** (verifica authority `ROLE_ADMIN`)	endpoint riservato agli admin
`@PreAuthorize("hasRole('USER')")`	solo utenti con ruolo **USER** (verifica authority `ROLE_USER`)	endpoint riservato agli user
`@PreAuthorize("hasAnyRole('ADMIN','USER')")`	almeno uno dei due ruoli (`ROLE_ADMIN` o `ROLE_USER`)	admin o user
`@PreAuthorize("hasRole('ADMIN') and hasRole('USER')")`	deve avere **entrambi** i ruoli (`ROLE_ADMIN` **e** `ROLE_USER`)	super-user con doppio ruolo
`@PreAuthorize("hasAuthority('ROLE_ADMIN')")`	verifica esattamente l'autorità `ROLE_ADMIN` (senza aggiungere prefissi)	come `hasRole('ADMIN')`, ma esplicito
`@PreAuthorize("hasAnyAuthority('ROLE_ADMIN','ROLE_USER')")`	almeno una delle authority prefissate (`ROLE_ADMIN` o `ROLE_USER`)	admin o user con prefisso
`@PreAuthorize("permitAll()")`	nessuna restrizione (accesso libero)	endpoint pubblico
`@PreAuthorize("denyAll()")`	nessun accesso	blocca tutti