

CREO PROGETTO ED INSERISCO LE DIPENDENZE

DATA JPA - DEVTOOLS - WEB - THYMELEAF - LOMBOK

## DOPO QUELLI DEL DB SCELTO

## AVUIO E MODIFICO LE APPLICATION.PROPERTIES

CREO ENTITIES DEL PROGETTO, RICORDARSI

6 ENTITY E 6 TABLE SU PK VA MESSO 6 ID

• GENERARE GETTERS, SETTERS E CONSTRUCTOR

REPOSITORY CON <sup>INTERFACE</sup> ENTITY REPOSITORY CHE ESTENDE  
JPA REPOSITORY / CHE SERVE AD OTTENERE METODI  
CRUD < ENTITY, TIPO PK >

< ENTITY, TYPE PK >  
STRING, INTEGER

DENTRO METTO SOLO NOME METODI E PARAMETRI

KEY <sup>↓</sup> WORDS CRUD

SOPRA @REPOSITORY

## 6 COMPONENT

CREO SERVICE ENTITY SERVICE, CREO OGGETTO >1

TIPO ENTITY REPOSITORY E GLI METTO IL SUO COSTRUTTORE

CON IN CIMA 6 AUTO WIRED

C, POSSO METTERE DEI METODI TIPO GETENTITIES

CON UNA LIST E POI RETURN USANDO L'OGGETTO

```
CREATE . FINDALL()
```

```
public List<Player> getPlayersByTeamAndPosition(String team, String position){  
    return playerRepository.findAll().stream()  
        .filter(player -> team.equals(player.getTeam()) && position.equals(player.getPos()))  
        .collect(Collectors.toList());  
}
```

RESTITUISCE UNA LISTA DI PLAYER CHE GIOCANO IN UN CERTO TEAM IN UN DATO RUOLO

**FINDALL** RECUPERA TUTTI GLI OGGETTI

**STREAM** SI USA PER FARE UNA SEQUENZA SU CUI APPLICARE CERTE OPERAZIONI

**FILTER** SI USA SOLO PER SELEZIONARE ELEMENTI CHE SODDISFANO DEI CRITERI SPECIFICI

**PLAYER** → INDICA OGNI ELEMENTO PLAYER DELLA FUNZIONE

**TEAM.EQUALS** CONFRONTA TEAM GIOCATORE CORRENTE CON QUELLO PASSATO IN INGRESSO

**POSITION.EQUALS** CONFRONTA POSIZIONE PASSATA CON QUELLA DEL GIOCATORE CORRENTE

**COLLECT** RACCOGLIE ELEMENTI FILTRATI IN UNA LISTA

```

public Player addPlayer(Player player){
    playerRepository.save(player);
    return player;
}

public Player updatePlayer(Player updatedPlayer){
    Optional<Player> existingPlayer = playerRepository.findByName(updatedPlayer.getName());

    if (existingPlayer.isPresent()){
        Player playerToUpdate = existingPlayer.get();
        playerToUpdate.setName(updatedPlayer.getName());
        playerToUpdate.setTeam(updatedPlayer.getTeam());
        playerToUpdate.setPos(updatedPlayer.getPos());
        playerToUpdate.setNation(updatedPlayer.getNation());

        playerRepository.save(playerToUpdate);
        return playerToUpdate;
    }
    return null;
}
}

```

SAVE SALVA PLAYER NEL DATABASE  
CON JPA - FA SIA INSERT CHE UPDATE

CREO ENTITYRESTCONTROLLER CHE SI OCCUPA DELLA GESTIONE  
DI CHIAMATE API, PER FARLO BISOGNA INSERIRE  
ANNOTAZIONE @RESTCONTROLLER COSÌ I METODI  
RITORNANO UN DOMINIO E NON UNA VISTA  
INSERISCO ANCHE @REQUESTMAPPING CHE INDICA  
URL DI PARTENZA

@PUTMAPPING

PER OGNI METODO VA MESSO @GET MAPPING CHE INDICA  
URL DEL METODO DA AGGIUNGERE A QUELLO BASE  
UN METODO ACCETTA PARAMETRI TRAMITE  
@REQUESTPARAM NELLA SEZIONE PARAMETRI

RITORNA DATI, MAI DELLE PAGINE

```

@PostMapping
public ResponseEntity<Player> addPlayer(@RequestBody Player player) {
    Player createdPlayer = playerService.addPlayer(player);
    return new ResponseEntity<>(createdPlayer, HttpStatus.CREATED);
}

@PutMapping
public ResponseEntity<Player> updatePlayer(@RequestBody Player updatedPlayer) {
    Player resultPlayer = playerService.updatePlayer(updatedPlayer);
    if (resultPlayer != null) {
        return new ResponseEntity<>(resultPlayer, HttpStatus.OK);
    } else {
        return new ResponseEntity<>(HttpStatus.NOT_FOUND);
    }
}

@DeleteMapping("/{playerName}")
public ResponseEntity<String> deletePlayer(@PathVariable String playerName) {
    playerService.deletePlayer(playerName);
    return new ResponseEntity<>(body:"Player deleted successfully", HttpStatus.OK);
}

```

USA METODO  
ADD PLAYER DEL  
SERVICE

SEMPRE METODI  
SERVICE

RESPONSE ENTITY CLASSE JAVA CHE RAPPRESENTA  
INTERA RISPOSTA HTTP CON STATUS CODE 200, 400.....  
EVENTUALI HEADER ED IL CORPO DELLA RISPOSTA  
NEL BODY AD ESEMPIO MANDA PLAYER APPENA AGGIUNTO  
CON MESSAGGIO HTTP 201 DI CONFERMA CREAZIONE RISORSA  
OPPURE SOLO MESSAGGIO CHE INDICA RISORSA NON  
TROVATA

PER TOGLIERE UN GIOCATORE BISOGNA PASSARE IL NOME  
NELL' URL DELLA RICHIESTA

IN QUESTO CASO PUNTANO AGLI STESSI ENDPOINT, DEVO  
SOLO OCCUPARMI DI CAMBIARE IL TIPO DI  
RICHIESTA HTTP

SOLO ENTITYCONTROLLER, ANNOTATO CON @CONTROLLER SI  
OCCUPA DI RITORNARE VISTE HTML SALVATE IN  
RESOURCES → TEMPLATES


SEMPRE INDICARE INDIRIZZO HTML

```
@Controller
public class PageController {

    @GetMapping("/")
    public String home() {
        return "home"; // carica home.html
    }
}
```

```
@RestController
@RequestMapping("/api/photos")
public class PhotoRestController {

    @GetMapping
    public List<Photo> getPhotos() {
        return photoService.GetPhotos(); // JSON automatico
    }
}
```



```

1 package com.imagedb.controller;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Controller;
5 import org.springframework.web.bind.annotation.GetMapping;
6
7 import com.imagedb.entities.Photo;
8 import com.imagedb.service.PhotoService;
9
10 @Controller
11 public class PhotoController {
12
13     private final PhotoService photoService;
14
15     @Autowired
16     public PhotoController(PhotoService photoService) {
17         this.photoService = photoService;
18     }
19
20     @GetMapping("/")
21     public String home() {
22         return "home";
23     }
24
25     @GetMapping("/photos")
26     public String showPhotos(org.springframework.ui.Model model) {
27         Iterable<Photo> photos = photoService.getPhotos();
28         model.addAttribute("photos", photos);
29         return "photos"; // carica photos.html da templates
30     }
31 }
32
33

```

MODEL È UN OGGETTO CONTENITORE  
USATO DA JAVA PER PASSARE DATI  
AD HTML

OTTIENE LISTA DI OGGETTI PHOTO  
TRAMITE SERVICE

NOTE COLLEZIONE E FILE RELATIVO

IN QUESTO MODO INTEGRO METODI SERVICE ANCHE  
NEL CONTROLLER PER MANDARE DATI FOTO AD UN  
HTML

```

1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3 <head>
4     <title>Galleria Foto</title>
5 </head>
6 <body>
7     <h1>Foto disponibili</h1>
8     <ul>
9         <li th:each="photo : ${photos}">
10             <span th:text="${photo.id}">Titolo</span>
11             
12         </li>
13     </ul>
14 </body>
15 </html>
16

```

MODEL PASSA COLLEZIONE  
DI OGGETTI PHOTOS E QUI  
CICLA IN OGNIUNO PER VEDERE  
SINGOLI DATI

SOSTITUISCE URL CON  
/img/01.png, LUI SA CHE STA  
DENTRO RESOURCES/STATIC/URL

