

## {MINGGU IV}.OPERASI CRUD DENGAN SPRING BOOT

Oleh: TMP - 2020/2021

*Note: praktikum ini mengasumsikan anda telah menginstal dan mengkonfigurasi eclipse dengan Spring Boot plugin, serta terkoneksi dengan internet.*

Pada praktikum kali ini anda akan belajar tentang operasi CRUD dengan menggunakan Spring Boot. Operasi-operasi dasar ini sudah dipermudah dengan adanya Service DAO dari JpaRepository. Seperti pada konsepnya kalau service DAO, POJO (mirip Java Beans) dan atau logik-logik bisnis terdapat pada layer kedua, Business tier, bila dilihat dari sudut pandang Software Architecture. Berikut contoh aplikasi sederhana untuk bisa mengerti bagaimana caranya client aplikasi (misal: browser) berkomunikasi dengan beberapa layer sehingga bisa menampilkan data.

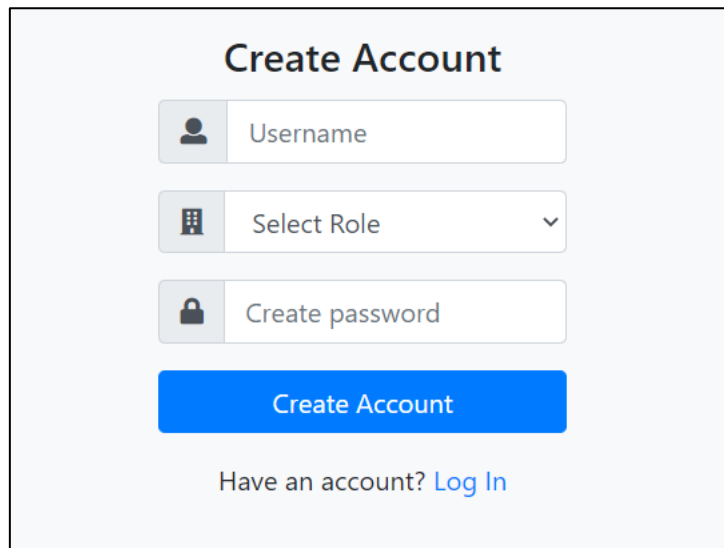


Figure 1. Form Daftar Pengguna Baru

Jika anda lihat form sederhana di atas, form tersebut merupakan *java servlet* yang berada di dalam *web container*. *Java servlet* berperan dalam *me-render* halaman web menjadi tag HTML untuk dapat ditampilkan ke dalam browser. Ketika pengguna menginput data lewat form tersebut, *java servlet* juga yang membawa data ini ke layer *business tier* untuk dapat diolah untuk proses selanjutnya (bila ada).

Sebelum membuat aplikasi sederhana di atas, buatlah sebuah database dengan nama "db\_happytravelling" di dalam container MySQL anda, kemudian buatlah 3 tables dengan nama table t\_user, t\_role dan t\_login. Insert data untuk table t\_role dimana datanya adalah admin dengan roleid '1' dan non-admin roleid '2'.

```
create database db_happytravelling;
use db_happytravelling;

create table t_user(username varchar(50), pwd varchar(50), roleid int);
create table t_role(roleid int, roledesc varchar(50));
create table t_login(username varchar(50), roleid int, lastlogin datetime, isactive tinyint(1));

insert into t_role values(1, 'Admin');
insert into t_role values(2, 'Non-admin');
```

Buatlah proyek Spring Boot baru lalu buka application.properties. Edit file tersebut, isi dengan informasi datasource anda.

```
## MySQL
spring.datasource.url=jdbc:mysql://192.168.164.222:6603/db_happytraveling
spring.datasource.username=root
spring.datasource.password=root
```

Note: sesuaikan **url, username dan password datasource** dengan environment anda.

Buat beberapa POJOs berdasarkan table fisik yang ada di dalam database anda. Letakan POJOs anda ke dalam folder src/main/java dengan nama package **del.ac.id.demo.jp**

```
File: User.java
package del.ac.id.demo.jp;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "t_user")
public class User {
    @Id
    @Column(name="username")
    private String username;
    @Column(name="pwd")
    private String pwd;
    @Column(name="roleid")
    private int roleid;

    public User() {}
    public User(String username, String pwd, int roleid) {
        this.username = username;
        this.pwd = pwd;
        this.roleid = roleid;
    }

    public void setUsername(String username) {this.username = username;}
    public void setPwd(String pwd) {this.pwd = pwd;}
    public void setRoleid(int roleid) {this.roleid = roleid;}

    public String getUsername() {return this.username;}
    public String getPwd() {return this.pwd;}
    public int getRoleid() {return this.roleid;}
}
```

Penjelasan: bila anda perhatikan kode program di atas disana terdapat anotasi @Entity yang maksudnya adalah kelas tersebut merupakan "mappingan" dari sebuah table fisik di database, atau dengan kata lain merupakan representasi physical table ke dalam bentuk java. Sedangkan anotasi @Table dengan parameter 'name' memberitahukan bahwa kelas User.java merupakan representasi sebuah table dengan nama "t\_user".

Pembuatan kelas POJO haruslah mengikuti kaidah Java Beans, salah satunya adalah deklarasi variables (fields) memiliki akses spesifikier 'private', dan anda harus menyediakan methods 'setter-getter' untuk masing-masing fields tersebut. Naming convention untuk methods 'setter'-getter' kelas POJO haruslah diawali dengan karakter lower-case kemudian diikuti upper-case untuk kata kedua. contoh: **setUsername**.

Anotasi @Id menunjukan bahwa field tersebut merupakan Id di dalam physical table anda; sedangkan anotasi @Column dengan parameter 'name' menunjukan mappingan 'column-name' yang ada di dalam database. Sedangkan untuk konstruktor sendiri akses spesifiernya haruslah 'public'.

File: Role.java

```
package del.ac.id.demo.jpjpa;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "t_role")
public class Role {
    @Id
    @Column(name="roleid")
    private int roleid;
    @Column(name="roledesc")
    private String roledesc;

    protected Role() {}
    public Role(int roleid, String roledesc) {
        this.roleid = roleid;
        this.roledesc = roledesc;
    }

    public void setRoleid(int roleid) {this.roleid = roleid;}
    public void setRoledesc(String roledesc) {this.roledesc = roledesc;}

    public int getRoleid() {return this.roleid;}
    public String getRoledesc() {return this.roledesc;}
}
```

Penjelasan: hampir sama dengan User.java

File: Login.java

```
package del.ac.id.demo.jpjpa;

import java.util.Date;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "t_login")
public class Login {
    @Id
    @Column(name="username")
    private String username;
    @Column(name="roleid")
    private int roleid;

    @Column(name="lastlogin")
    private Date lastlogin;
}
```

```

@Column(name="isactive")
private boolean isactive;

public Login() {}
public Login(String username, int roleid, Date lastlogin, boolean
isactive) {
    this.username = username;
    this.roleid = roleid;
    this.lastlogin = lastlogin;
    this.isactive = isactive;
}

public void setUsername(String username) {this.username = username;}
public void setRoleId(int roleid) {this.roleid = roleid;}
public void setLastlogin(Date lastlogin) {this.lastlogin = lastlogin;}
public void setIsactive(boolean isactive) {this.isactive = isactive;}

public String getUsername() {return this.username;}
public int getRoleId() {return this.roleid;}
public Date getLastlogin() {return this.lastlogin;}
public boolean getIsactive() {return this.isactive;}
}

```

Penjelasan: hampir sama dengan User.java

Setelah anda membuat POJOs, anda harus membuat service DAO untuk masing-masing kelas tersebut. Hal ini bertujuan untuk mengijinkan business layer berkomunikasi dengan persistence layer (database). Letakan DAO services ini ke dalam folder src/main/java dengan nama package **del.ac.id.demo.jpaa**

File: UserRepository.java

```

package del.ac.id.demo.jpaa;

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

public interface UserRepository extends JpaRepository<User, String> {
    User findByUsername(String username);
}

```

Penjelasan: bila anda perhatikan kode program di atas, anda sedang membuat service DAO untuk table 'user'. Service anda ini akan meng-extends service JPA (yaitu: JpaRepository). Segala fungsionalitas dasar CRUD untuk setiap fields di dalam kelas POJO sudah disediakan. Misalnya: 'findByUsername', 'findByRoleId', dsb.

Perlu anda ketahui, anda tidak diperkenankan oleh Spring framework untuk menambah abstrak method lain di dalam service DAO anda selain fields yang terdaftar di kelas POJO. Contoh: boolean isValidUser(String username); **tidak diperkenankan**. Mengapa demikian? Karena itu aturan dari framework tersebut.

Bagaimana bila anda ingin mendaftarkan berbagai service lain di luar dari service CRUD fields POJO? Buat service JPA repository sendiri (lihat tutorial dari [link](#) ini). Kalau dilihat dari kode program di atas, anda sedang menggunakan service JPA repository yang telah disediakan oleh Spring Framework (**import org.springframework.data.jpa.repository.JpaRepository**).

Bagaimana bila anda tetap ingin menggunakan service JPA repository yang telah digunakan, tapi ingin mendaftarkan service-service untuk melakukan berbagai operasi lain? Anda dapat membuat kelas java baru (kelas controller) yang me-load seluruh data yang ada lewat service JPA repository, lalu gunakan data tersebut untuk proses yang anda perlukan. Namun, ini tidak pendekatan yang bagus, karena akan memakai banyak

memory, meskipun performansi baik; yang disarankan itu adalah anda membuat sendiri service JPA repository sendiri.

File: RoleRepository.java

```
package del.ac.id.demo.jpa;

import java.util.List;

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

public interface RoleRepository extends JpaRepository<Role, Integer> {
    Role findByRoleid(int roleid);
    List<Role> findAll();
}
```

Penjelasan: hampir sama dengan UserRepository.java

File: LoginRepository.java

```
package del.ac.id.demo.jpa;

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

public interface LoginRepository extends JpaRepository<Login, String> {
    Login findByUsername(String username);
}
```

Penjelasan: hampir sama dengan UserRepository.java

Setelah anda berhasil membuat service DAO untuk masing-masing kelas POJOs, sekarang anda dapat membuat controller yang handle proses registrasi. Buatlah file java dengan nama 'RegistrationController.java', letakan kode program ini ke dalam folder src/main/java dengan nama package **del.ac.id.demo.controller**

File: RegistrationController.java

```
package del.ac.id.demo.controller;

import java.util.List;

import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import del.ac.id.demo.jpa.Role;
import del.ac.id.demo.jpa.RoleRepository;
import del.ac.id.demo.jpa.User;
import del.ac.id.demo.jpa.UserRepository;

@RestController
public class RegistrationController {
    private UserRepository userRepository;
    private RoleRepository roleRepository;
```

```

    public RegistrationController(UserRepository userRepository,
RoleRepository roleRepository) {
        this.userRepository = userRepository;
        this.roleRepository = roleRepository;
    }

    @GetMapping("/registration")
    public ModelAndView registration() {
        List<Role> listRoles = roleRepository.findAll();
        System.out.println(listRoles.size());

        ModelAndView mv = new ModelAndView("registration");
        mv.addObject("roles", listRoles);
        mv.addObject("user", new User());

        return mv;
    }

    @RequestMapping(value="/registration", method = RequestMethod.POST)
    public String registrationSubmit(@ModelAttribute User user, BindingResult
bindingResult, Model model) {
        if (bindingResult.hasErrors()) {
            System.out.println("Error");
        }

        model.addAttribute("user", user);
        userRepository.save(user);
        return "redirect:login";
    }
}

```

Penjelasan: bila anda perhatikan kode program di atas, anotasi @RestController merupakan anotasi Spring MVC, kombinasi 2 buah anotasi, yaitu @Controller dan @ResponseBody. Perlu anda ketahui bahwa kelas yang ditandai dengan anotasi @Controller akan menangani request yang berasal dari klien, kemudian kelas ini akan meng-invoke business class untuk mengerjakan proses-proses yang dibutuhkan kemudian me-redirect hasilnya ke logical view. Sedangkan anotasi @ResponseBody bertujuan untuk mem-binding return value dari sebuah method ke dalam HTTP response body.

Terakhir adalah membuat tampilan dari form registrasi (viewer). Buatlah sebuah file HTML dengan nama registration.html lalu letakan ke dalam folder src/main/resources/templates

```

<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">

<head>
    <meta charset="utf-8"/>
    <link
href="//maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"
rel="stylesheet" id="bootstrap-css">
    <script
src="//maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"></script>
    <script
src="//cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
    <!------- Include the above in your HEAD tag ----->

    <link rel="stylesheet"
href="https://use.fontawesome.com/releases/v5.0.8/css/all.css">

```

```

</head>

<body>
  <div class="card bg-light">
    <article class="card-body mx-auto" style="max-width: 400px;">
      <h4 class="card-title mt-3 text-center">Create Account</h4>

      <form action="#" th:action="@{/registration}" th:object="${user}"
method="post">
        <div class="form-group input-group">
          <div class="input-group-prepend">
            <span class="input-group-text"> <i class="fa fa-user"></i>
          </span>
          <input name="" class="form-control" placeholder="Username" type="text"
th:field="*{username}">
        </div> <!-- form-group// -->
        <div class="form-group input-group">
          <div class="input-group-prepend">
            <span class="input-group-text"> <i class="fa fa-building"></i>
          </span>
          <select class="form-control" name="dropRole" id="dropRole"
th:field="*{roleId}">
            <option value="0"> Select Role</option>
            <option th:each="role : ${roles}" th:value="${role.roleId}"
th:text="${role.roledesc}"></option>
          </select>
        </div> <!-- form-group end.// -->
        <div class="form-group input-group">
          <div class="input-group-prepend">
            <span class="input-group-text"> <i class="fa fa-lock"></i>
          </span>
          <input class="form-control" placeholder="Create password"
type="password" th:field="*{pwd}">
        </div> <!-- form-group// -->

        <div class="form-group">
          <button type="submit" class="btn btn-primary btn-block"> Create Account
        </button>
        <div> <!-- form-group// -->
        <p class="text-center">Have an account? <a th:href="@{/Login}">Log In</a>
      </p>
    </form>
  </article>
</div> <!-- card.// -->

</body>
</html>

```

Penjelasan: bila anda perhatikan kode HTML yang di-highlight berwarna biru, itu merupakan objek yang sudah di-embed lewat kelas controller Registration.java. Perhatikan juga cara mengembodkannya di kelas java ini lewat potongan kode highlight warna biru. Setelah anda berhasil membuat seluruh kode program, jalankan aplikasi anda, lalu akses ke URL: <http://localhost:8080/registration>

### TUGAS:

1. Kerjakan secara lengkap untuk proses login dan juga tampilkan halaman utama (index.html). Bila anda berhasil login, tampilkan username dan juga role di halaman index.html.
2. Buatlah aplikasi web sederhana (happy-travelling) dengan asumsi semua penerbangan (dengan maskapai dan tanggal berapapun) available disitus tersebut. Pengguna tinggal memilih departure, destination dan juga tanggal sekaligus jamnya. Ketika user biasa mengakses situs happy-travelling, form booking yang muncul di index.html adalah sebagai berikut:
- 3.

<b>From:</b>	<b>To:</b>	<b>No. of Pessangers</b>
<input type="text" value="Medan (KNO)"/>	<input type="text" value="Jakarta (CGK)"/>	<input type="text" value="2"/>
<b>Departure Date:</b>	<b>Return Date:</b>	<input type="button" value="Book"/>
<input type="text" value="2 Oktober 2020 14:35:00"/>	<input type="text" value="5 Oktober 2020 12:00:00"/>	

Sedangkan untuk admin, admin dapat menambah data asal/tujuan penerbangan. Admin juga dapat menghapus penerbangan yang telah dipesan pengguna. Untuk table dan rancangan ERD nya diserahkan kepada anda sepenuhnya. Ingat: setiap booking tiket pesawat harus memiliki ID transaksi yang akan dipakai untuk sistem lain (akan dikembangkan di week selanjutnya).