CA2 – JPA mapping and REST

David Wroblewski

Daniel Jensen

Casper Tobias Hansen

Martin Weber Hansen

Henrik Dyrting Ørvald

1: Anti injection koden er taget fra et tidligere projekt hvor koden bliver leveret af Lars Mortensen

1. Introduktion

Dette er en rapport, lavet omhandlende ”CA2 – JPA mapping and REST”, på cphbusiness lyngby.

Rapporten skal bruges som dokumentation, samt forklaringer på nogle af de ting, der er blevet lavet.

Rapporten belyser også eventuelle mangler, fejl og forbedringer af disse.

2. Design

2.1 Fejlhåndtering

Vi har en exeptions klasse med exeption NotFoundExeption. Denne klasse bør udvides med flere exeptions der kan bruges i koden. Derudover bør der skrives til en log fil så vi nemmere kan vedligeholde programmet og hurtigere finde fejl.

2.2 Endeligt valg af entity til DB

Efter et par start vanskeligheder med forsøg på at oprette entity klasser fra sql og den anden vej, bestemte vi os for at oprette vores sql udfra entity klasserne. Dette indeholder sine plusser og minusser.

Vi vælger denne metode da det tillader os at behandle vores kode som objekter i stedet for at skulle forholde os særlig meget til SQL. Derudover udnytter vi også fordelene af det som relationelle databaser gør bedst.

2.3 DTYPE's ansvars område

DTYPE er vores pegepind der holder styr på data når et eller flere tables er afhængige af hinanden.

3. Test strategi

I dette afsnit vil vi beskrive vores test af vores PersonFacadeDB klasse ud fra opgave teksten.

Vores test kører op imod vores database. Testen er ikke optimal da det enten kræver at vi lukker for tilgang til databasen fra net siden imens vi kører testen, eller retter stien for databasen til en test database. Dette kunne gøres via en separat persistance fil til testen.

Testen er for vores PersonFacadeDB som indeholder alle vores metoder til at manipulere en persons data, samt vores fejlhåndtering af forkerte data inputs.

Der blev påbegyndt en test sådan som Anders viste den i torsdags med interfaces, simpleclasses og mock men vi var ikke sikre på hvad den skulle indeholde da vi kunne sætte en test op til at blive 100% uden databasen.

1: Anti injection koden er taget fra et tidligere projekt hvor koden bliver leveret af Lars Mortensen

Testen var ikke sat op til at kører mod en testdatabase da vi allerede havde lavet en anden test kaldt PersonFacadeTest.

Testen har en fejl pga et sequence drop vi ikke kan se hvorfor ikke skulle virke. Ellers virker testen.

4. Who did what

Database/entityklasser: Henrik , David, Casper, Daniel, Martin

Test: Martin, David, Daniel, Henrik

HTML/script: Daniel, Henrik

Rapport: Henrik, Casper

Azure: Daniel

For at tjekke Commits bedes i bruge Pulse i stedet for Graph for at få de rigtige tal her vil master også være inkluderet.

5. Ekstra funktioner

EditMetoden

EditPerson bruges til og ændre i en allerede eksisterende person, Fx en der har fået nyt telefon nummer, eller hvor en anden attribut ændres fra bruger siden.

1: Anti injection koden er taget fra et tidligere projekt hvor koden bliver leveret af Lars Mortensen

```java
@Override
public Person editPerson(String json) throws NotFoundException
{
    try (ClosableManager em = new ClosableManager(emf)) {
        Person person = gson.fromJson(json, Person.class);
        Person oldValue = em.find(Person.class, person.getId());
        if (oldValue == null) {
            throw new NotFoundException("No person exist with the given id");
        }
        em.getTransaction().begin();
        em.merge(person);
        em.getTransaction().commit();
        Person newValue = em.find(Person.class, person.getId());
        return newValue;
    }
}
```

EditPerson er en af de ekstra metoder som vi har tilføjet programmet, som er foreslået i opgave beskrivelsen som en ekstra funktionalitet. Dette har vi gjort da vi mener at den var relevant for programmet og da vi havde tiden til at færdiggøre den.

Et kald fra brugeren om at ændre en person bliver sendt til vores PersonHandler som et ”PUT” –request.

PersonHandleren finder id for den person der skal rettes via URL’en. Herefter kontrollerer PersonHandler lige om String newValues indeholder <> for at undgå injection[1] . Id hentet fra URL bliver brugt til at konstruere en Json string som vi bruger i vores editPerson.

Da vores Person klasse er ORM kompatibel kan vi finde en person udfra vores json string . Vi bruger herefter vores EntityManager til at starte en transaktion og merge de nye informationer med de, for personen, allerede eksisterende informationer. Herefter returneres den nye person hele vejen tilbage til bruger siden.

Hvis en person indeholdende et ugyldigt id finder frem til vores edit metode, vil metoden smide en ”NotFoundExeption”.

6. Vigtige funktioner

AddRole

AddRole tildeler en eller flere rolle/r til en person, som bliver brugt til at se, hvilken rolle den pågældende person har på skolen. Fx Bo er en Student og sætter et navn for hans semester.

Vi har delt koden op i 3 dele.

1: Anti injection koden er taget fra et tidligere projekt hvor koden bliver leveret af Lars Mortensen

Del 1

```
public RoleSchool addRole(String json, int id) throws NotFoundException
{
    try (ClosableManager em = new ClosableManager(emf)) {
        Person person = em.find(Person.class, id);
        RoleSchool r = null;
        JsonElement jelement = new JsonParser().parse(json);
        JsonObject jobject = jelement.getAsJsonObject();
        String roleName = jobject.get("roleName").getAsString();
        System.out.println("roleName = " + roleName);
```

Vi henter den person der har det pågældende id, som vi får med når metoden kaldes. Herefter kommer en lille manøvre hvor vi opretter et JElement via en JsonParser, som vi herefter laver om til et JObject. Dette object har en get metode hvor vi kan søge efter vores ”roleName”.

Vi henter value ”roleName” fra den json String vi får med, når metoden kaldes, og gemmer den ned i en ny String ”roleName”.

Del 2

```
switch (roleName.toUpperCase()) {

    case "TEACHER":
        String degree = jobject.get("discription").getAsString();
        r = new Teacher(degree);
        person.addRole(r);
        break;

    case "STUDENT":
        String semester = jobject.get("discription").getAsString();
        r = new Student(semester);
        person.addRole(r);
        break;

    case "ASSISTANTTEACHER":
        r = new AssistantTeacher();
        person.addRole(r);
        break;
    default:
        throw new NotFoundException("String rolename not found");
```

Vores roleName kan vi så bruge i vores switch case.

1: Anti injection koden er taget fra et tidligere projekt hvor koden bliver leveret af Lars Mortensen

Vores case kan håndtere vores tre roller som givet i opgaveteksten. Herefter henter vi, via vores objekt, parameteret ”discription”, som vi har brug for til vores rolle. Hviis ”roleName” ikke er en af de 3 muligheder der er håndterbare af vores switch case , kaster programmet en ”NotFoundException.

Del 3

```
em.getTransaction().begin();
em.merge(person);
em.getTransaction().commit();
return r;
```

Tilsidst opretter vi forbindelse til databasen via EntityManageren, og merger rollen ind på personen.

1: Anti injection koden er taget fra et tidligere projekt hvor koden bliver leveret af Lars Mortensen