Object-Relational Mapping

Michał Kowalczyk, Bartłomiej Zajda, Katarzyna Pyrczak, Filip Szołdra

February 17, 2021

1 Cel projektu

Celem projektu jest zaimplementowanie klas realizujacych mapowanie O-R dowolnego modelu dziedziny w technologii Spring. Jako motor bazy danych wybrany został SQL. Docelowo bedzie możliwe odwzorowanie architektury systemu informatycznego przedstawionego w postaci obiektowej na relacyjna baze danych.

2 JPA

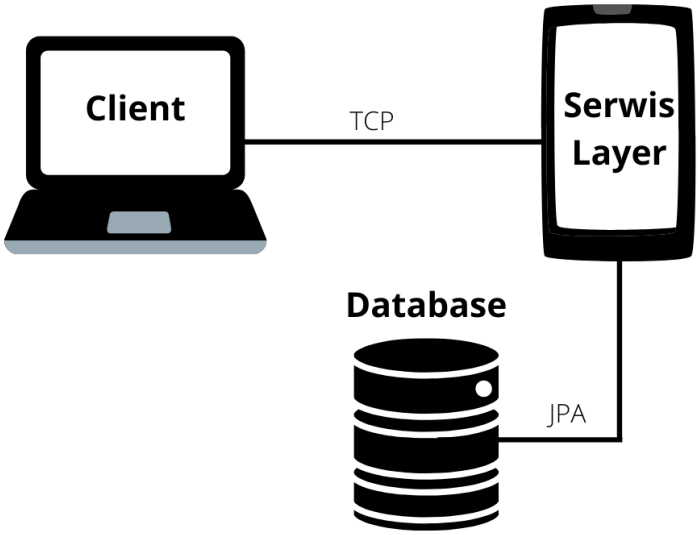
JavaPersistence API - standard ORM dla jezyka JAVA. Jest to sposób operowania na obiektach i zapisywanie wyników do relacyjnej bazy danych za pomoca obiektu EntityManagera. Relacje pomiedzy elementami bazy danych moga być zdefiniowane za pomoca adnotacji. JPA definiuje również jezyk zapytań JPA Query Language.

3 Użyte technologie

- Java
- SpringBoot
- postgreSQL
- Maven

4 Diagram deployment

Zaprojektowana przez nas aplikacja bazuje na architekturze client-server. W tej architekturze w komunikacji aplikacji klienckiej z baza danych pośredniczy serwer aplikacji. Użyty przez nas model architektury client-serwer pozwala na odizolowanie logiki aplikacji od aplikacji klienckiej. Użycie technologii Spring Boot powoduje umieszczenie logiki aplikacji na serwerze aplikacji oraz w bazie danych. Dzieki podzieleniu na warstwy, aplikacja stanie sie bezpieczniejsza oraz wydajniejsza, pozwala to też na prostsze utrzymanie kodu.



5 Implementacja mappingu

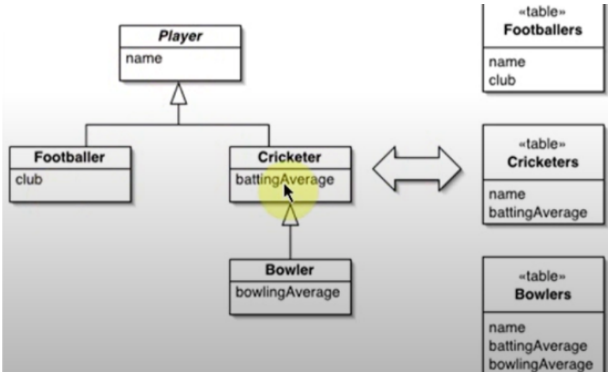
Inheritance Mapping jest to wzorzec odpowiadajacy za sposób dziedziczenia klas i polimorficzne querowanie. Jest kilka implementacji inheritance mapping my w naszym projekcie przyjmiemy 3.

- single table
- concrete table
- table per class

Single table jest to najprostsza implementacja, klasy 'dzieci' beda zapisywane w klasie rodzic, i bedzie tworzona dodatkowa kolumna z informacja o typie dziecka np:

Tutaj w przykładzie klasy Cat i Dog dziedzicza z klasy Animal.

Concreate Table jest to wzorzec mapowania polegajacy na tym, że podklasa posiada w bazie danych wszystkie artybuty nadklasy, czyli wyglada to nastepujaco.



W **Table per Class** tabela jest zdefiniowana dla każdej klasy w hierarchi dziedziczenia i zawiera tylko lokalne atrybuty klasy. Wszystkie klasy w takiej hierarchii musza mieć taki sam atrybut id. Table per Class natomiast zachowowa sie nastepujaco dla przypadku z Kotem i Psem:

Podczas querowania, takie tabele zostaja złaczone (za pomoca opercji UNION).

Wybór odpowiedniego typu mapowania dla dziedziczenia klas, bedzie możliwy za sprawa adnotacji **DatabaseTable**. Jako argument takiej adnotacji bedzie przypisanie do pola **inheritanceMappingType** wybranego typu ***SingleTable, ConcreteTable, ClassTable***.

```
@DatabaseTable(inheritanceType=InheritanceMappingType.ConcreteTable)
class A{

}

class B extends A{
}
```

W przypadku mapowania SingleTable skorzystamy z kolejnych dwóch adnotacji **DiscriminatorColumn** oraz **DiscriminatorValue**. Dzieki nim bedziemy mogli utworzyć dodatkowa kolumne zawierajaca typ podklasy. W naszym przykładzie z psem i kotem bedzie to wygladało mniej wiecej tak:

```
@DatabaseTable(inheritanceType=InheritanceMappingType.SingleTable)
@DiscriminatorColumn(name="dtype")
class Animal{

}
@DiscriminatorValue(name="dog")
class Dog extends Animal{
}

@DiscriminatorValue(name="cat")
class Cat extends Animal{
}
```

W przypadku gdy nasze odwzorowanie bedzie tworzyć osobne tabele dla każdej klasy, czyli ClassTable musimy mieć jakieś połaczenie miedzy nimi, aby móc je później łaczyć podczas zapytań. Odpowiedzialna bedzie za to kolejna adnotacja **JoinColumn**.

```
@DatabaseTable(inheritanceType=InheritanceMappingType.ClassTable)
class Animal{

}
@JoinColumn(name="dID")
class Dog extends Animal{
}

@JoinColumn(name="cID")
class Cat extends Animal{
}
```

6 Tworzenie bazy danych

W pakiecie resources beda dwa pliki:

- ***application.properties*** - tam podajemy namiary na url bazy danych, użytkownika, i hasło, ustawiamy tam również na true flage, która bedzie usuwac wszysktie tabele na starcie oraz podajemy namiary na package, w której znajdować sie beda klasy z adnotacjami
- ***plik z rozszerzeniem .sql***, w którym tworzymy użytkownika i hasło i tworzymy tabele

Adnotacje implementujemy jako interfejsy, i bedziemy je z czytywać z każdej klasy za pomoca funkcji getAnnotations() z pakietu Fields.
Działać to bedzie w nastepujacy sposób:

- DatabaseField.class - odpowiada za pojedyncza kolumne
- DatabaseTable.class - odpowiada za nazwe tabeli
- DiscirminatorColumn.class - odpowiada za typ podklasy z mappingu 'single table'
- Id.class - odpowiada za klucz, jest unikatowe
- JoinColmun.class - adnotacja potrzebna podczas 'joinowania' columny podczas mappingu, kiedy nie korzystamy z implementacji single table
- JoinTable.class - adnotacja potrzebna podczas 'joinowania' tabeli podczas mappingu, kiedy nie korzystamy z implementacji single table
- ManyToMany.class - adnotacja potrzebna podczas definiowania relacji miedzy classami
- OneToMany.class - adnotacja potrzebna podczas definiowania relacji miedzy classami

W Annotacji użyjemy utrzymania jako **RETENTION.POLICY = RUNTIME**
(beda dostepne dla JVM tylko podczas czasu trwania programu).

W @TARGET użyjemy najbardziej ogólnego jako **ElementType.TYPE**.

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
```

Poniższy kod jest dla najbradziej podstawowego przypadku, ustawiamy pole w interfejsie..

```
@DatabaseTable(inheritanceType=InheritanceMappingType.CLASS_TABLE)

public class classA{
    @DatabaseField
    public String stringA;

    @Id
    @DatabaseField
    private int intA;
}
```

Poniżej przykład dla przypadku, kiedy mapping ma znaczenie, gdyż dziedziczymy jedna klase.

```
@DatabaseTable(inheritanceType=InheritanceMappingType.SINGLE_TABLE)
public class SingleB extends SingleA{
    @DatabaseField
    protected int intB;

    @DatabaseField
    String stringB;
}
```

7 Realizacja paradygmatu "by exception"

Antywzorzec coding by exception polega na dodawaniu nadmiernej liczby osobnych typów wyjatków dla każdego możliwego błedu oraz kodu do ich obsługi, prowadzi do niekontrolowanego wzrostu zbednej złożoności.
Aby uniknać tego problemu w implementacji, przepływ strumienia danych zostaje sprawdzony warunkami, uwzgledniajacymi powszechne błedy. Sprawdzenie to pozwala do rozwiazać problem, bez wywoływania wyjatku. Korzystanie z wyjatków w przypadkach, kiedy jest to konieczne, jest oparte na dodatkowych klasach w pakiecie Exceptions, co wyróżnia przypadki i pozwala zachować przejrzystość kodu źródłówego.

8 Wzorce Projektowe

8.1 Data Access Object

Zawiera wszystkie operacje wykonywane na obiekcie z bazy danych(insert,
select, update, delete), ukrywa pod warstwa biznesowa szczegóły implementacyjne, które zawarte sa w klasach insertExecutor, selectExecutor, updateExecutor i deleteExecutor.

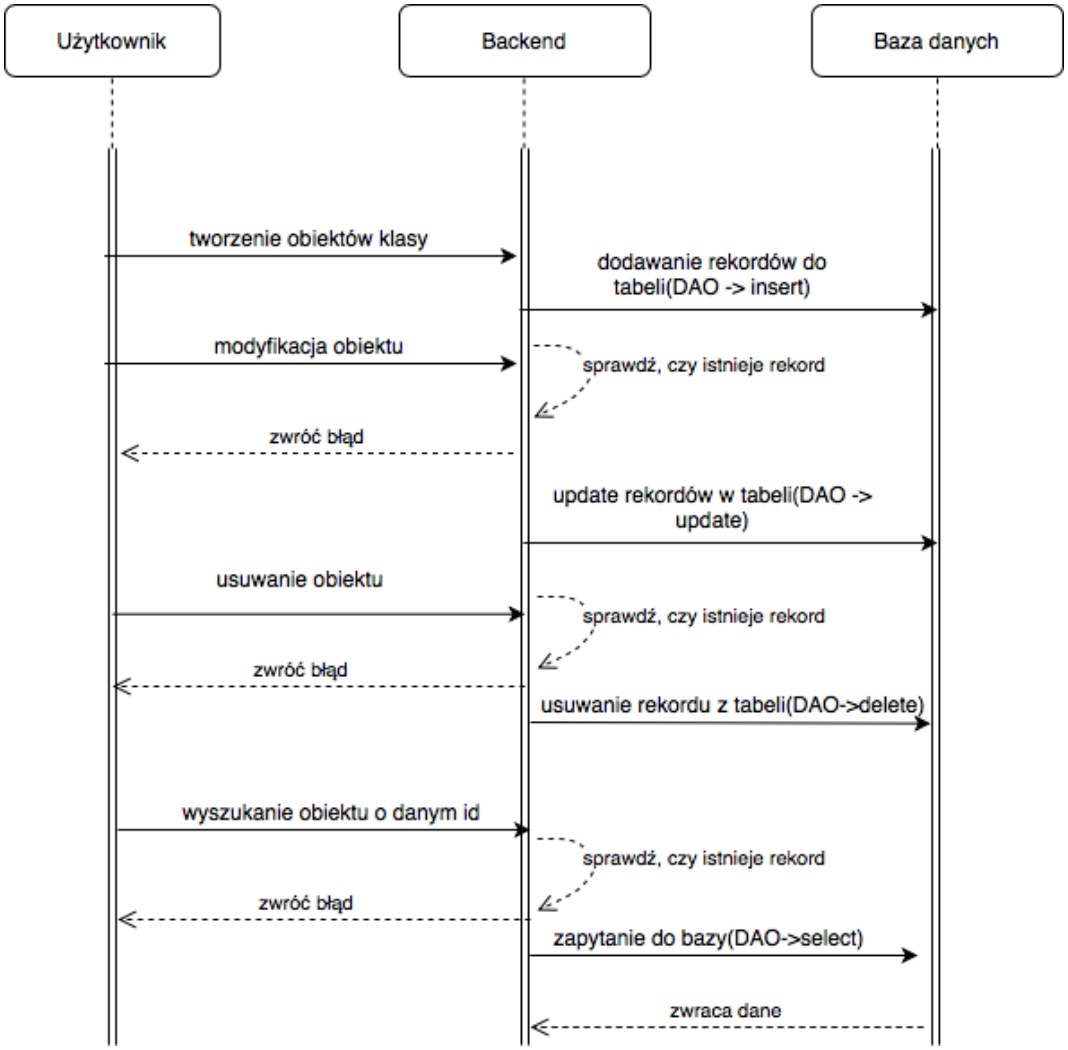
8.2 Singleton

Wykorzystany w klasie DatabaseSchema, gwarantuje istnienie tylko jednej instancji tej klasy. Dzieki temu zmieniajac ten obiekt w jednej klasie, inna klasa dostaje ten sam obiekt i bedzie miała dostep do wprowadzonych zmian.

8.3 Builder

wykorzystany w klasie TableSchema. Umożliwia dołaczanie kolejnych kolumn do tworzonej tabeli.

9 Dynamika systemu



Gdy użytkownik tworzy nowy obiekt klasy, na backend zostaje przysłana informacja o konieczności stworzenia nowego rekordu w odpowiedniej tabeli. Jeżeli dana tabela już istnieje, to backend dodaje kolejny rekord w tej tabeli w bazie danych za pomoca klasy insertExecutor wyoływana przed DAO.

Gdy użytkownik chce zmodyfikować obiekt, na backend jest najpierw wysylane zapytanie, czy rekord odpowiadajacy temu obiektowi w ogóle istnieje. Jeżeli istnieje, to odpowiedni rekord w bazie danych zostaje updatowany za pomoca klasy updateExecutor wywołana przez DAO. W przeciwnym razie zostaje zwrócona informacja o błedzie.

Jeżeli użytkownik chce usunać obiekt, to również na poczatku przesylane jest zapytanie na backend, czy odpowiadajacy mu rekord w bazie danych istnieje. Jeżeli nie, to zwracana jest informacja o błedzie. Jeżeli tak, to odpowiedi rekord w bazie danych zostaje usuniety za pomoca klasy deleteExecutor wywoływana przez DAO.

Rónież jeżeli użytkownik chce znaleźć obiekt o danym id, to najpierw jest wysyłane zapytanie na backend, czy w bazie danych istnieje odpowiadajacy mu rekord. Jeżeli nie, to zostaje zwrócona informacja o błedzie, a jeżeli tak, to wysylane jest zapytanie do bazy za pomoca klasy selectExecutor wywołana przez DAO i zwracane sa odpowednie dane..

10 Schemat projektu