Car dealership - Projekt

Spis treści

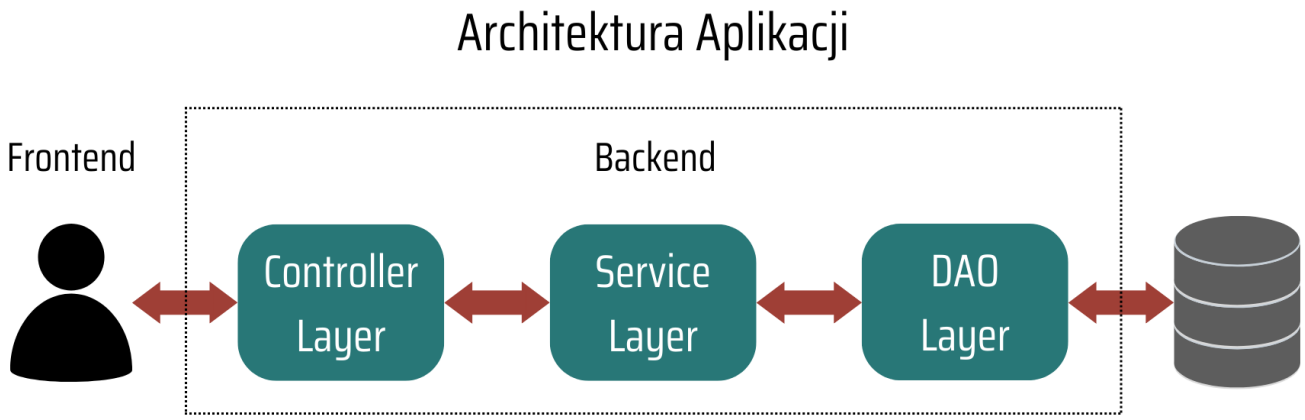
Opis zagadnienia

W ramach tego zagadnienia oprzemy się na zaprojektowanej wcześniej bazie danych w projekcie **Car Dealership**.

Chcę tutaj zaznaczyć, że w swoich przykładowych rozwiązaniach oprę się o przykładową bazę danych zaprojektowaną w poprzednim zagadnieniu. Jeżeli masz ochotę, to możesz oczywiście oprzeć się o zaprojektowaną przez siebie bazę danych. Natomiast ja w przykładowych rozwiązaniach oprę się o projekt bazy danych, który przedstawiłem jako przykładowe rozwiązanie poprzedniego zagadnienia.

Po realizacji poprzedniego zagadnienia powinieneś/powinnaś mieć przygotowany projekt bazy danych razem z potrzebnymi skryptami DDL. Teraz przyjdzie kolej na przygotowanie kodu, który będzie korzystał z przygotowanego wcześniej projektu.

Przypomnij sobie, że rozmawialiśmy wcześniej o tematyce architektury aplikacji i wspomnieliśmy wtedy o warstwowej architekturze aplikacji:



Obraz 1. Warstwowa architektura aplikacji

- Frontend - aplikacja, która uruchamia się użytkownikowi w przeglądarce internetowej,
- Controller Layer (*warstwa kontrolerów*) - ta część aplikacji będzie służyła do wystawiania backendu na świat i do tego, żeby frontend mógł z naszym backendem rozmawiać. Jeszcze tego nie umiemy, ale spokojnie, przejdziemy do tego,
- Service Layer (*warstwa usług*, *warstwa logiki biznesowej*) - ta część aplikacji będzie służyła do realizacji logiki biznesowej. Cały czas uczymy się technik, które pozwalają nam na realizację jakiejś logiki biznesowej,
- DAO Layer (*warstwa dostępu do danych*) - ta część aplikacji będzie odpowiedzialna za dostęp do

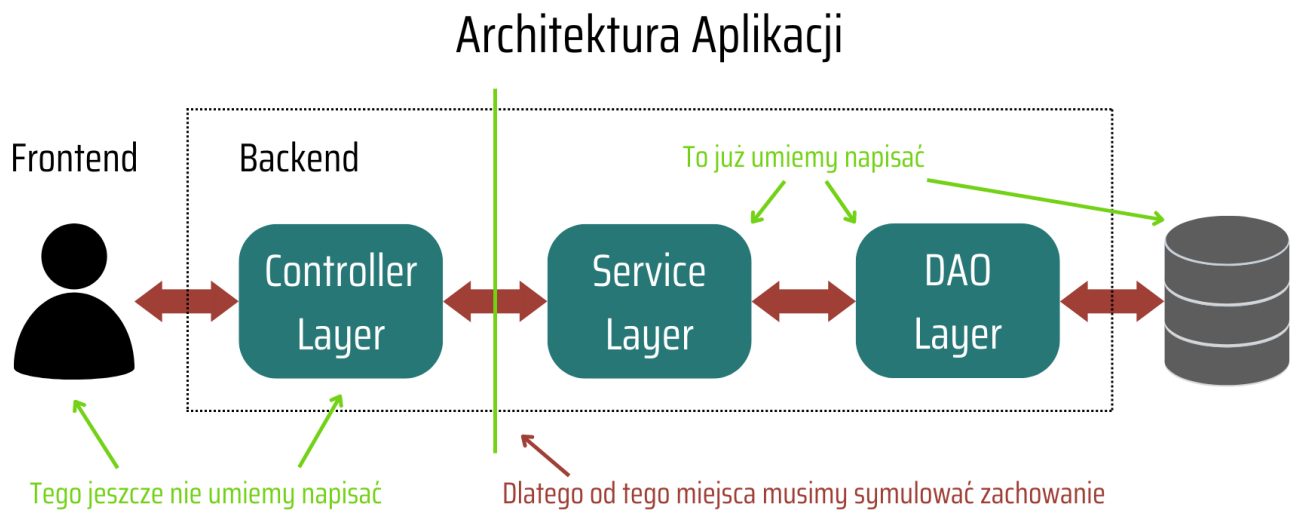
szeroko rozumianych danych. Dane takie mogą być przetrzymywane w bazie danych lub nawet w pliku. W obrębie tego warsztatu oraz innych, gdzie poruszamy się w tematyce baz danych, tak na prawdę uczymy się cały czas o warstwie DAO oraz komunikacji z bazą danych.

Chciałbym żebyśmy w ramach tego zagadnienia przygotowali warstwy **DAO layer** oraz **Service Layer**. Czyli napiszemy część aplikacji, która będzie odpowiedzialna za logikę biznesową takiej aplikacji oraz za zapisywanie danych w bazie danych. Opis logiki biznesowej znajdziesz poniżej, natomiast wywołanie takiej logiki biznesowej będzie następowało z poziomu metody `main()`.

W praktyce użytkownicy takiej aplikacji podczas wykonywania swoich akcji na stronie (pobieranie danych, zapisywanie danych itp) doprowadzaliby do modyfikacji danych w bazie danych. Użytkownik taki korzystałby z aplikacji frontendowej, a jego ruchy byłby przekazywane do backendu. Backend przy wykorzystaniu warstwy **Controller Layer** odbierałby żądania od aplikacji frontendowej i realizował pewną logikę biznesową, która opierałaby się również o warstwę **DAO**.

Oczywiście cały ten opis to wielkie uproszczenie i o tym wszystkim będziemy się jeszcze uczyć. Szkopuł tego projektu polega na tym, że chcemy przygotować warstwę logiki biznesowej oraz warstwę repozytoriów bazodanowych, ale musimy zasymulować w jakiś sposób ruch użytkowników, żeby móc napisać tylko część prawdziwego backendu, bo całości jeszcze nie umiemy.

Nazywam to częścią, bo cały backend składałby się z kontrolerów, serwisów oraz repozytoriów. My napiszemy tylko serwisy i repozytoria, czyli musimy w jakiś sposób nagiąć wywołania serwisów, które byłby wołane przez kontrolery - bo kontrolerów jeszcze nie znamy i ich nie napiszemy. Dlatego właśnie część aplikacji będziemy symulować. Mam nadzieję, że dobrze wyjaśni to poniższa grafika.



Obraz 2. Warstwowa architektura aplikacji

Dlatego właśnie został przygotowany plik `car-dealership-traffic-simulation.md`, który zawiera "zrzut" ruchu użytkowników w aplikacji. Napisz warstwy logiki biznesowej oraz repozytorium w taki sposób, żeby cały plik `car-dealership-traffic-simulation.md` mógł zostać wczytany i przetworzony poprawnie.

Cały projekt będzie polegał na napisaniu aplikacji w oparciu o Hibernate (nie poruszaliśmy w tym warsztacie Springa, ale to jak zrealizujesz projekt to jest Twój wybór), tak żeby była ona w stanie wczytać i przetworzyć plik `car-dealership-traffic-simulation.md` i zapisać dane do bazy danych, a w efekcie na końcu mamy mieć uzupełnioną bazę danych, której projekt stworzyliśmy wcześniej.

Schemat bazy danych został wypracowany przez nas wcześniej. Diagram ERD możesz znaleźć w pliku

`car_dealership_erd_diagram.png`, a DDL znajdziesz w pliku `car_dealership_ddl.sql`.

W ramach realizacji zagadnienia możesz napisać np. test, który będzie definiował kolejne metody testowe, które będą uruchamiana w określonej kolejności.

Mała dygresja. Diagram `car_dealership_erd_diagram.png` oraz skrypt `car_dealership_ddl.sql` uległ nieznacznym zmianom w stosunku do tego, co zostało wypracowane wspólnie. Zrobiłem to specjalnie, żeby zobrazować Ci, że w praktyce może wyglądać to tak, że wypracujesz coś, a następnie kolega z zespołu lekko to pozmienia, bo będzie chciał poprawić stosowane nazewnictwo. Żeby zasymulować taką sytuację, wyobraź sobie, że ja jestem takim kolegą i zmieniłem nieznacznie wspomniane pliki. Zapoznaj się z nimi, zanim zaczniesz działać.