08.01.2023

PZSP2 – Konsultacje na temat oprogramowania

Zespół 7 MHMD – Mateusz Krakowski, Hubert Soroka, Maciej Tymoftyjewicz, Daniel Kobiałka

1. Temat

Tematem naszego projektu jest platforma edukacyjna, pozwalająca na dodawanie pytań połączonych z odpowiedzią i źródłem wiedzy (taki zestaw będzie dalej zwany fiszką), przeglądanie fiszek oraz grupowanie ich z użyciem grup i tagów. Użytkownicy muszą się zalogować, aby przeglądać i dodawać nowe fiszki i grupy, które mają ustawienia publiczności.

2. Funkcjonalności aplikacji:

- Założenie konta użytkownika
- Zalogowanie się na konto
- Wylogowanie się z konta
- Tworzenie, edytowanie oraz usuwanie fiszek
- Tworzenie, edytowanie oraz usuwanie grup
- Wyszukanie fiszek i grup po nazwie
- Dodawanie tagów
- Wyszukanie fiszek i grup po tagu
- Dodanie fiszki do grupy (podczas dodawania lub edycji grupy, oraz podczas dodawania lub edycji fiszki)
- Przechodzenie przez tryb nauki dla danej grupy (przeglądanie fiszek)
- Przechodzenie przez tryb testu dla danej grupy (użytkownik widzi pytanie, może udzielić odpowiedzi i po przejściu przez test porównać swoje odpowiedzi do odpowiedzi prawidłowych)

3. Przechodzenie przez tryb testu dla danej grupy (użytkownik widzi pytanie, może udzielić odpowiedzi i po przejściu przez test porównać swoje odpowiedzi do odpowiedzi prawidłowych)

4. Ogólne założenia architektury

Projekt wykonywany z użyciem bazy danych Oracle (udostępnianej przez uczelnię), backendu w Javie z użyciem Spring oraz niezależnego frontendu w typescripcie z Reactem. Widać jasno trzy rozłączne części, każda z nich funkcjonująca niezależnie od pozostałych, każda uruchamiania oddzielnie.

Baza danych komunikuje się z backendem poprzez interfejs Spring Data JPA. Backend wystawia Restowe API, przez które frontend nawiązuje połączenie. Dane przesyłane Jsonem, po http. Uwierzytelnienie i autoryzacja następują z użyciem tokena JWT, generowanego po każdym zalogowaniu się. Na jego podstawie przyznawane uprawnienia.

5. Szczegóły architektury backendu

Diagram przedstawiający architekturę backendu (w razie problemów, na moodlu także załączony plik svg w wersji ciemnej):

UI oraz ZAPYTANIA są realizowane po stronie aplikacji Reactowej. Nie stworzyliśmy diagramu dla komponentów frontendowych, ponieważ z punktu widzenia organizacji architektury, całość danych, które przetwarzają, będzie przechodzić przez klasę Requests z diagramu. Klasy wyżej niż ZAPYTANIA odpowiadają tylko prezentacji danych, a nie ich przetwarzaniu.

Klasy z grupy MODELE są zamieszczone na boku warstw SERWISÓW oraz REPOZYTORIÓW, ponieważ są często wykorzystywane przez obie z nich. Nie zamieszczono linii notacji UML przy każdym wykorzystaniu tych klas, ponieważ uczyniłoby to diagram nieczytelnym. Te klasy są bezpośrednio mapowane na tabele relacyjne w bazie danych, reprezentują wszystkie główne tabele, pozostałe służą tylko reprezentacji związków N-N bez atrybutów pomiędzy nimi.

Istnieją też klasy pomocnicze wykorzystywane do obsługi api, w katalogu models/api. Unikamy wysyłania klas modeli mapowanych do bazy danych na frontend, aby uniknąć przesyłania nadmiarowych danych, oraz nie udostępniać pól wszystkich klas powiązanych (np. nie chcemy udostępniać maila twórcy każdej fiszki przy pobieraniu ich listy).

Kiedy niemożliwe lub zabronione jest dalsze wykonanie zapytania, rzucany jest wyjątek typu BackendException. Te wyjątki są obsługiwane przez ApiExceptionHandler, który wysyła na frontend komunikat o błędzie z kodem 418 (stosowany na potrzeby testów, przy rozszerzeniu obsługi wyjątków będzie zmieniony).

Na frontendzie w pliku Requests.ts jest zaimplementowane generyczne obsługiwanie zapytań, dzięki czemu niezależnie od typu danych, w kontrolkach reactowych można bardzo podobnie pobierać dane z backendu.

Nasz pomysł na architekturę najprościej można podsumować następująco:

I. Komponent frontendowy, potrzebując konkretnych danych, wywołuje jedną z funkcji z klasy statycznej Requests
II. Każda funkcja tej klasy odpowiada jednej akcji w jednym z kontrolerów
III. Kontrolery wywołują metodę z odpowiedniego serwisu, który jest polem kontrolera. Kontrolery zawierają minimum logiki „biznesowej” programu. Odsyłają dane w klasie z sufiksem Response, są to proste klasy bez metod, które zawierają tylko to, co będzie potrzebne na frontendzie w kontekście danej metody.
IV. Serwis wykonuje operacje na obiektach z modelu dostarczonych przez repozytoria tak, aby móc zwrócić wymagane dane lub rzucić wyjątek, który jest obsłużony przez ApiExceptionHanlder i generowana jest odpowiedź zawierająca komunikat wyjątku.
V. Repozytoria dziedziczą po JpaRepository, dostarczonym przez Spring. Dzięki czemu nie trzeba ich implementować, wystarczy stworzyć sam interfejs, nazywając metody zgodnie ze standardem Spring. Zapytania im odpowiadające wygenerują się automatycznie. W przypadku, gdy potrzeba wykonać trudniejsze zapytanie, wystarczy dodać adnotację @Query(…) i podać zapylanie SQLowe.
VI. Spring wykonuje zapytania i zwraca obiekty z modelu przedstawiające relacyjne dane z bazy danych.

Architekturę będziemy rozwijać i utrzymywać w oparciu o powyższe założenia, wydają się nam wartościową podstawą do dalszego rozwoju systemu.

6. Testy

Zaimplementowaliśmy testy jednostkowe oraz integracyjne. Testy integracyjne skupiają się na dodawaniu powiązanych obiektów do bazy danych i sprawdzaniu, czy występują pomiędzy nimi poprawne relacje.

Testy systemowe miały formę testów manualnych, gdzie sprawdzaliśmy scenariusze, które nas interesowały, np. „Czy zwykły użytkownik będzie widział świeżo dodane grupy prywatne administratora?”. W momencie ukończenia dużej funkcjonalności i wystawieniu Merge Requesta na GitHubie, były przeprowadzone testy manualne, celem sprawdzenia, czy wszystko działa jak należy. Testy manualne są zwłaszcza kluczowe w kontekście zmian w komunikacji frontendu z backendem, zwłaszcza w dziedzinie uprawnień i uniemożliwiania pewnych akcji.

7. Link do repozytorium

https://gitlab-stud.elka.pw.edu.pl/mkrakows/pzsp_poniedzialek_grupa_7