Praca Dyplomowa Inżynierska

Dawid Wijata
205006

Porównanie aplikacji frontendowych opartych na mikrofrontendach z tradycyjną architekturą monolityczną na przykładzie aplikacji do zarządzania finansami osobistymi

Comparison of microfrontend applications and monolith frontend applications based on the example of expense tracker

Praca dyplomowa na kierunku:
Informatyka

Praca wykonana pod kierunkiem
dr inż. Piotra Wrzeciono
Instytut Informatyki Technicznej
Katedra Systemów Informacyjnych

Warszawa, rok 2023

SZKOŁA GŁÓWNA GOSPODARSTWA WIEJSKIEGO

Wydział Zastosowań Informatyki i Matematyki

Oświadczenie Promotora pracy

Oświadczam, że niniejsza praca została przygotowana pod moim kierunkiem i stwierdzam, że spełnia ona warunki do przedstawienia tej pracy w postępowaniu o nadanie tytułu zawodowego.

Data ..... Podpis promotora .....

Oświadczenie autora pracy

Świadom/a odpowiedzialności prawnej, w tym odpowiedzialności karnej za złożenie fałszywego oświadczenia, oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami prawa, w szczególności z ustawą z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (Dz. U. 2019 poz. 1231 z późn. zm.)

Oświadczam, że przedstawiona praca nie była wcześniej podstawą żadnej procedury związanej z nadaniem dyplomu lub uzyskaniem tytułu zawodowego.

Oświadczam, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną. Przyjmuję do wiadomości, że praca dyplomowa poddana zostanie procedurze antyplagiatowej.

Data ..... Podpis autora pracy .....

Streszczenie

Stworzenie klasy LaTeX-owej do użytku przy pisaniu pracy dyplomowej w SGGW

Tematem niniejszej pracy było zaimplementowanie klasy LaTeX-owej pozwalającej na formatowanie tekstu zgodnie z wytycznymi nałożonymi przez uczelnię. Praca zawiera dwie główne części. Pierwsza z nich zawiera opis najważniejszych aspektów implementacji klasy. Natomiast druga część skupia się na sposobie użycia klasy przez osoby piszące prace dyplomowe.

Słowa kluczowe – LaTeX, klasa, praca dyplomowa, implementacja, SGGW, Szkoła Główna Gospodarstwa Wiejskiego

Summary

Creation of the LaTeX Class to be Used When Writing a Thesis at the Warsaw University of Life Sciences – SGGW

The subject of this study was to implement a LaTeX class that allows for text formatting according to the guidelines imposed by the University. The work consists of two main parts. The first one describes the most important aspects of the implementation. The second part focuses on how to use the class by people writing the theses.

Keywords – LaTeX, class, thesis, implementation, SGGW, Warsaw University of Life Sciences

Spis treści

1 Wprowadzenie do mikroserwisów

1.1 Architektura monolityczna

1.2 Architektura mikroserwisów

1.3 Przełożenie mikroserwisów na interfejsy użytkownika

2 Mikrofrontendy

2.1 Założenia teoretyczne

2.2 Zmiany w architekturze względem mikroserwisów

3 Funkcjonalność badanej aplikacji

Na potrzeby porównania obu architektur stworzona została aplikacja do zarządzania finansami osobistymi o roboczej nazwie Midas. W założeniu odbiorcą aplikacji mają być pojedyncze osoby lub gospodarstwa domowe, które chcą zadbać o kontrolę nad swoim budżetem domowym. Głównymi funkcjonalnościami aplikacji są:

- możliwość wykonywania operacji CRUD w zakresie informacji o wydatkach i przychodach poszczególnych użytkowników
- system autoryzacji i uwierzytelniania spełniający aktualne normy w zakresie bezpieczeństwa aplikacji sieciowych
- system uprawnień w obrębie rodziny (przykładowo użytkownik o roli rodzica może edytować transakcje na kontach dzieci, a dzieci nie mogą podglądać transakcji rodziców)
- przechowywanie paragonów i faktur w wersji elektronicznej i możliwość przypisania ich do konkretnej transakcji

Podany zestaw funkcjonalności umożliwi wydzielenie kilku mikroserwisów, co zapewni wystarczającą bazę do symulowania połączeń między serwisami. Wykonanie jej w architekturze ściśle mikroserwisowej postanowi jednocześnie dobudować do tych mikroserwisów poszczególne mikrofrontendy tak, aby stanowiły one razem pełnoprawne aplikacje komunikujące się między sobą. Z drugiej strony pozwoli to też dobudować monolityczny frontend, który będzie komunikował się ze wszystkimi serwisami. Dzięki takiemu posunięciu dla obu badanych projektów - monolitycznego oraz mikrofrotendowego, zapewnione będą jednolite warunki w zakresie komunikacji z backendem, co będzie stanowiło bazę do porównania obu koncepcji architektonicznych w zakresie wydajności i dostępności.

4 Opis backendu projektu

4.1 Dobór technologii do projektu

Do zrealizowania tej części projektu został wykorzystany język C# oraz środowisko .NET. Użyto środowiska .NET w wersji 6.0, która jest jednocześnie najnowszą dostępną wersją LTS (Long Term Support). Wspomniane technologie są dobrym wyborem do realizacji mikroserwisów ze względu na:

- użycie paradygmatu programowania obiektowego i będący jego częścią polimorfizm, który sprzyja replikacji pojedynczego wzorca mikroserwisu
- będący częścią środowiska .NET framework ASP.NET realizujący architekturę REST przy niewielkim narzucie w ilości kodu
- istnienie wielu gotowych klas i modeli realizujących podstawowe funkcje sieciowe takie jak autoryzacja i autentykacja, komunikacja z bazą za pomocą Entity Framework
- generator NSwag służący do generowania klas w językach C# oraz TypeScript reprezentujących metody kontrolerów poszczególnych mikroserwisów wykonanych w ASP.NET

Środowiska mikroserwisowe są de facto oddzielnymi serwisami działającymi jednocześnie i pobierającymi dostępne zasoby. Do celów testów lokalnych na komputerze programisty zachodzi więc potrzeba zastosowania rozwiązania, które zminimalizuje użycie zasobów komputera tak, aby jednocześnie jak zachować wierność odwzorowania środowiska z wieloma serwerami. W celu osiągnięcia takiego efektu, zostało użyte oprogramowanie Docker służące do konteneryzacji środowisk. Za jego pomocą, używając specjalnych plików nazywanych obrazami można tworzyć kontenery, którym Docker przydziela zasoby w czasie rzeczywistym tak, aby zoptymalizować ich użycie. Razem z narzędziem Docker, użyto też narzędzia *docker-compose*, które umożliwia uruchomienie wielu kontenerów Dockera za pomocą pojedynczego skryptu.

Opisane technologie mogą też być komfortowo używane w połączeniu z oprogramowaniem Docker służącym do konteneryzacji. Producent środowiska .NET, Microsoft udostępnił w serwisie Docker Hub obraz środowiska .NET z ustawionym frameworkiem ASP.NET oraz połączeniem z bazą SQL Server. Pozwala to na skorzystanie z gotowego środowiska, które jest konfigurowalne za pomocą pliku konfiguracyjnego o nazwie Dockerfile.

4.2 Szablony projektów

W celu zapewnienia skalowalności oraz łatwego tworzenia nowych serwisów, na potrzeby projektu opracowano dwa szablony projektu - zawierający wstępną autentykację użytkownika poprzez sprawdzanie zawartości nagłówka HTTP oraz taki, który jej nie zawiera.

Najważniejszą cechą tych szablonów jest możliwość szybkiego ustawienia nowego rozwiązania Visual Studio zawierającego ustawienia dla testów jednostkowych i integracyjnych aplikacji, ustawienia plików Dockerfile i docker-compose.yml, oraz ściśle określonego rozłożenia projektów w rozwiązaniu. Zapewnia to spójność kodu w zakresie całej aplikacji. W przypadku, gdyby nad kodem pojedynczego serwisu pracowało kilku programistów, mają oni ściśle narzuconą przez szablon strukturę kodu i podstawowa jego struktura zostanie zachowana. To z kolei powoduje, że wynikowo mimo tego, że nad poszczególnymi serwisami pracują różne osoby, ich struktura jest w dużym stopniu podobna. Ma to wiele zalet w zakresie zarządzania projektami, przykładowo:

- pozwala na szybszą aklimatyzację programistów przenoszonych między projektami, bądź takich, których rola w zespole pomaga na wsparciu istniejących projektów
- przyspiesza czas tworzenia nowych funkcjonalności oprogramowania
- czas poświęcany na odtwórcze powtarzanie realizacji wzorca można poświęcić na ważniejsze czynności takie jak redukcja długu technologicznego, zwiększanie pokrycia testami, itd.

4.3 Podział backendu na serwisy

Logika backendowa aplikacji została podzielona według funkcjonalności na serwisy opisane w poniższych podsekcjach.

4.3.1 Hosting plików - File Storage Service

Serwis z hostingiem plików odpowiada za przechowywanie wszystkich plików przekazanych aplikacji przez użytkownika. Są to między innymi zdjęcia profilowe użytkowników oraz pliki z dowodami wykonania transakcji przypisanych do konkretnych transakcji w ramach logiki domenowej. Serwis hostingu plików pozwala też na pobranie samego pliku o znanym identyfikatorze UUID oraz pobranie informacji o umieszczeniu pliku oraz konkretnych pobrań pliku.

4.3.2 Autoryzacja - Authorization Service

Serwis odpowiada za autoryzację użytkowników oraz operacje związane z użytkownikami, które wymagają zachowania ostrożności pod kątem bezpieczeństwa aplikacji - tworzenie konta, logowanie do konta w aplikacji Midas, zmiana hasła. W tym serwisie przy logowaniu powstaje token JWT, który zapisany w ciasteczkach krąży po innych serwisach przekazywany przez nagłówek *Authorization* w zapytaniu HTTP.

4.3.3 Zarządzanie użytkownikami - User Service

Serwis odpowiada za przechowywanie oraz umożliwienie dostępu do informacji o użytkownikach. Obsługuje on też logikę związaną z kontami użytkowników, które nie wymagają zachowania szczególnej ostrożności w zakresie bezpieczeństwa aplikacji. Przykładem takiej operacji może być zmiana zdjęcia profilowego użytkownika.

4.3.4 Zarządzanie rodzinami - Family Service

Serwis odpowiada za przypisanie użytkowników do rodzin, których są członkami. Odpowiada też częściowo za uwierzytelnianie użytkownika - z tego serwisu pochodzą dane na temat tego, czy dany użytkownik ma wystarczające uprawnienia do edytowania danych dla swojej rodziny (ze względu na rolę przypisaną w systemie).

4.3.5 Logika domenowa - Transaction Service

Serwis ma za zadanie realizację operacji CRUD na transakcjach finansowych. Takimi operacjami może być wpisanie nowego wydatku do listy, usunięcie go z listy, pobranie listy wydatków w zależności do konkretnych parametrów (np. kategoria, czas, osoba z rodziny), dodanie pliku z dowodem wykonania transakcji.

5 Opis badanych frontendów

5.1 Dobór technologii do projektów

Do realizacji mikrofrontendów użyto biblioteki *single-spa*. Wspiera ona wszystkie najpopularniejsze frameworki do tworzenia aplikacji frontendowych. W badanych projektach użyto frameworka Angular ze względu na to, że najlepiej nadaje się do tworzenia dużych skalowalnych aplikacji. Ta cecha pozwoli utrzymać w ryzach potencjalnie rozwlekłą strukturę monolitycznej wersji projektu. W celu zapewnienia jak najmniejszych różnic technologicznych w projektach o różnej architekturze, wersja mikrofrontendowa będzie również używać frameworka Angular.

Do komunikacji z serwisami backendowymi, dla obu projektów użyto klas serwisów wygenerowanych przez generator NSwag. Będzie to działało tak jak w przypadku klas w języku C# generowanych na potrzeby komunikacji między serwisami backendowymi. Różnica będzie tu jedynie taka, że te klasy będą gotowymi klasami w języku TypeScript dostosowanymi do użycia we frameworku Angular.

Ze względu na to, że walory estetyczne nie są istotne w zakresie rozważań nad mikrofrontendami, użyto gotowej biblioteki z elementami wizualnymi o nazwie *angular-material*.

5.2 Wersja monolityczna

5.3 Wersja mikroserwisowa

6 Porównanie projektów frontendowych

6.1 Wersjonowanie kodu

6.2 Wydajność

6.3 Dostępność

6.4 Testowanie kodu

6.5 Zależności między modułami i projektami

6.6 Możliwości w zakresie zarządzania projektami

6.7 Skalowalność

6.8 Koszty ustawienia środowiska produckyjnego

7 Podsumowanie

8 Bibliografia

[1] Martin Fowler. *Micro frontends*. Czerwiec 2019.
URL: https://martinfowler.com/articles/micro-frontends.html.

[2] Michael Geers. *Micro frontends*. Sierpień 2017.
URL: https://micro-frontends.org/.

[3] Michael Geers. *Micro frontends in action*.
Shelter Island, Nowy Jork: Manning Publications Co., 2020. ISBN: 9781617296871.

[4] L. Mezzalira. *Building Micro-Frontends*. O'Reilly Media, 2021.
ISBN: 9781492082941.
URL: https://books.google.pl/books?id=MjpPEAAAQBAJ.

[5] Luca Mezzalira. *Adopting a micro-frontends architecture*. Kwiecień 2019.
URL: https://medium.com/dazn-tech/adopting-a-micro-frontends-architecture-e283e6a3c4f3.

[6] Luca Mezzalira. *Micro-frontends, the future of Frontend Architectures*.
Kwiecień 2019. URL: https://medium.com/dazn-tech/micro-frontends-the-future-of-frontend-architectures-5867ceded39a.

[7] Andrey Pavlenko i in.
"Micro-frontends: application of microservices to web front-ends."
W: *J. Internet Serv. Inf. Secur.* 10.2 (2020), s. 49–66.

[8] *single-spa*. URL: https://single-spa.js.org/.

[9] Olaf Sulich. *Fundamenty Architektury Mikrofrontendowej*. Październik 2022.
URL: https://frontlive.pl/blog/fundamenty-architektury-mikrofrontendowej.

[10] *The State of Frontend 2022*. Maj 2022. URL:
https://tsh.io/state-of-frontend/?fbclid=IwAR1NoxwAAX5qTprLKiJm4k-LBFWaQny3j3F7XIGTgWZRBpyYjQDb7bnq5DQ#report.

[11] Caifang Yang, Chuanchang Liu i Zhiyuan Su.
"Research and Application of Micro Frontends". W: *IOP Conference Series: Materials Science and Engineering* 490 (kwiecień 2019), s. 062082.
DOI: 10.1088/1757-899x/490/6/062082.
URL: https://doi.org/10.1088/1757-899x/490/6/062082.

Wyrażam zgodę na udostępnienie mojej pracy w czytelniach Biblioteki SGGW w tym w Archiwum Prac Dyplomowych SGGW.

. . . . . .
(czytelny podpis autora pracy)