Rozdział 4

Implementacja rozwiązania

4.1 Opis rozwiązania

4.1.1 Architektura systemu

4.1.2 Model danych

4.2 Problemy rozwiązane w trakcie realizacji pracy

Inżynieria oprogramowania polega między innymi na rozwiązywaniu problemów. Wiele z nich posiada więcej niż jedno rozwiązanie i wybranie tego odpowiedniego dla danego projektu spoczywa na programiście, który często musi wykazać się wiedzą techniczną jak i biznesową. Właśnie z tego powodu, w tej sekcji opiszę wybrane zagadnienia i problemy, które napotkałem i rozwiązałem w trakcie realizacji projektu.

4.2.1 Wybór technologii

W tej sekcji skupię się na opisaniu wybranych technologii **back-endu**, **front-endu**, technologii **baz danych** oraz **asynchronicznego przekazywania eventów**. Porównam je również z alternatywami, które były przeze mnie rozważane.

Rozpatrywałem kilka języków, które mogłem użyć po stronie serwera: **Java**, **Kotlin** oraz **Go**. Wszystkie te języki łączy:

- **Akceptowalna dla moich wymagań wydajność** (w zużyciu pamięci najlepiej wypada Go, wydajność zależnie od testu ale z lekką przewagą Go)[15].
- **Statyczne typowanie.**
- **Użycie *Garbage collectora*** - nie ma konieczności ręcznego zarządzania pamięcią, chociaż Go znajduje się, w porównaniu do Javy czy Kotlina, na niższym poziomie abstrakcji.
- **Bogaty ekosystem** (w Javie i Kotlinie skoncentrowany wokół ekosystemu Springa, w Go skupiony bardziej na bibliotece standardowej i jej pochodnych).

Trudno jest opisać czym jest odpowiedni poziom wydajności. O ile granica 500 ms na jedno żądanie wydaje się akceptowalna, jednak jest to miara, która jest chwiejna czyli zależy od środowiska, bazy danych (wszak prawdzie aplikacje muszą z nich korzystać) oraz wykonanej pracy. Dlatego przy porównywaniu wspomnianych przeze mnie języków skupiłem się na syntetycznych testach. Często stosuje się relatywne porównania wydajności aby ocenić języki programowania - więc dla wymagań tego projektu za akceptowalną wydajność uznaje się **nie** bycie w ogonie relatywnej wydajności czasowej i zużycia pamięci, a przynajmniej powyżej połowy stawki dla przykładowych porównań.

Dodatkowo, języki te są powszechnie używane przez firmy takie jak Netflix, Allegro czu Uber, które z powodzeniem wykorzystują je do budowania systemów opartych na mikroserwisach oraz budują wysoko-wydajne rozwiązania, specyficzne dla ich potrzeb [17][8][2]. Zdaje się to świadczyć o gotowości tych języków do bycia użytymi w środowisku produkcyjnym.

Początkowo rozważałem język Kotlin, jednak zdecydowałem, że uczenie się tego rozbudowanego w trakcie pisania pracy okaże się zbyt czasochłonne, dodatkowo język ten nie oferował dużo więcej w stosunku do Javy, chociażby ze względu na tożsamy ekosystem. W przypadku języka Go podjąłem decyzję, że w przypadku projektu dyplomowego będzie to rozwiązanie i wprowadzające niepotrzebne ryzyko projektowe, które wynika z odmienności tego języka od innych, z którymi miałem styczność. Przy ograniczonym czasie i stosunkowo dużym skomplikowaniu projektu zdecydowałem ograniczyć się do lepiej znanych mi technologii. **Java** była pragmatycznym wyborem, motywowanym przede wszystkim moim doświadczeniem w tym języku, które dodatkowo chciałem rozszerzyć opracowując aplikację od samego początku do końca.

Wybrany przeze mnie framework - **Spring Boot**[1] posiada bardzo bogaty ekosystem bibliotek oraz tzw. starterów[2], które dostarczają wiele przydatnych, sprawdzonych i bezpiecznych funkcjonalności. Chociaż Spring dominuje wśród użytkowników Javy, to pewną popularność zyskują także frameworki bardziej minimalistyczne jak Micronaut czy Quartus, rosnąc o pare procent w stosunku do poprzednich lat[12]. Dzięki architekturze mikroserwisów możliwe będzie przetestowanie tych technologii w pewnych częściach systemu.

W przypadku języka używanego po stronie klienta dwoma największymi konkurentami był "czysty" JavaScript[3] oraz TypeScript[4]. Wybrałem **TypeScript**, ponieważ tylko on spełniał mój wymóg dotyczący statycznego typowania[5].

Równie istotne było wybranie frameworka używanego po stronie front-endu - pisanie aplikacji opartych na czystym JavaScript jest możliwe i praktykowane z powodzeniem w nowoczesnych firmach [13], jednak nie mogłem sobie pozwolić na poświęcenie aż tak dużo czasu, gdyż wprowadzałoby to kolejne niepotrzebne ryzyko projektowe, a poza walorami edukacyjnymi nie wprowadziłoby znaczącej poprawy do jakości projektu. Wśród popularnych frameworków można wymienić między innymi React

[1] Strona projektu: `https://spring.io`.

[2] Lista projektów agregujących projekty pod nadzorem Springa jest dostępna tu: `https://spring.io/projects`.

[3] `https://developer.mozilla.org/en-US/docs/Web/JavaScript`

[4] `https://www.typescriptlang.org`

[5] Warto zauważyć, że w środowisku JavaScript istnieje również tzw. js-doc, które pozwalają na zastosowanie wsparcia podobnego do TypeScripta lecz bez kompilacji. `https://jsdoc.app`

(najbardziej popularny), Angular oraz Vue (zyskujący popularność niedawno) [12][16]. Zdecydowałem się jednak na wykorzystanie frameworka **SvelteKit**[6] (który jest tzw. metaframeworkiem[7] dla frameworka **Svelte**[8]). SvelteKit integruje się z frameworkiem Svelte (dostarczającym obsługę interfejsu użytkownika), rozszerzając go o router, API do obsługi żądań przeglądarki i przekierowywania ich do serwera, a także możliwości SSR[**serverside**]. Jest to dosyć nowo powstały framework, który obiecuje wysoką wydajność oraz dużą wygodę (prostotę) tworzenia aplikacji webowych. Warto tu zaznaczyć, że za pomocą SvelteKita można stworzyć kompletną aplikację internetową, łącznie z dostępem do bazy danych itd. - jest to zwyczajny serwer aplikacji. Okazało się to być dobrym wyborem między innymi ze względu na możliwość wykorzystania wzroca **BFF**[14] (Backend for frontend), o czym szerzej opowiedziane będzie w 4.2.2, chociaż podstawowym powodem była chęć poznania tej technologii.

Początkowo planowana była jedna baza danych. Naturalnym wyborem było rozwiązanie relacyjne, które dominuje obecnie na rynku wśród programistów - do najczęściej używanych relacyjncych DBMS należy **PostgreSQL**[9], chociaż popularne są również MySQL oraz MS SQL Server[16]. Do zalet **PostgreSQL** należy przede wszystkim otwarte źródło, ciągły rozwój i brak konieczności wykupywania licencji. Dodatkowo miałem z nim już styczność, więc postanowiłem zgłębiać dalej swoją wiedzę i oprzeć się o to rozwiązanie w trakcie implementacji. W toku prac okazało się, że konieczne jest rozdzielenie jednej bazy danych na kilka - tak, aby każdy serwis mógł posiadać i zarządzać własnymi danymi bez wpływu na inne serwisy. Mimo zwiększonego narzutu postanowiłem zachować oddzielne bazy danych na oddzielnych klastrach PostgresSQL co szerzej będzie opisane w kolejnych rozdziałach.

Jako broker wiadomości wykorzystałem **RabbitMQ**[10], który obok **Apache Kafka** jest jednym z najbardziej popularnych brokerów na rynku [16]. Dzięki wykorzystaniu protokołu AMQP ilość wysokiej jakości bibliotek klienckich jest większa niż dla Apache Kafka (która używa swojego własnego protokołu do komunikacji) co jest zaletą, szczególnie w perspektywie mikroserwisów, które mogą być pisane w różnych technologiach. W odróżnieniu od Apache Kafka, RabbitMQ jest prostszy do wdrożenia i konfiguracji (nie wymaga klastra i zarządcy), a dodatkowo wyróżnia się bardzo rozbudowanym sposobem trasowania wiadomości.

[6] Strona projektu: https://kit.svelte.dev

[7] Pojęcie głównie używane w ekosystemie JavaScript - https://prismic.io/blog/javascript-meta-frameworks-ecosystem

[8] Strona projektu: https://svelte.dev

[9] Strona projektu: https://www.postgresql.org

[10] Strona projektu: https://www.rabbitmq.com

Bibliografia

[1] Beck, K., *Test Driven Development: By Example*, English, 1st edition. Boston: Addison-Wesley Professional, list. 2002, ISBN: 978-0-321-14653-3.

[2] Blog, N. T., *Netflix OSS and Spring Boot — Coming Full Circle*, en, grud. 2018. adr.: https://netflixtechblog.com/netflix-oss-and-spring-boot-coming-full-circle-4855947713a0 (term. wiz. 10. 12. 2023).

[3] Dietrich, D. i Winkler, R., *Vavr User Guide*. adr.: https://docs.vavr.io/ (term. wiz. 01. 11. 2023).

[4] Evans, E., *Domain-Driven Design: Tackling Complexity in the Heart of Software*, English, 1st edition. Boston: Addison-Wesley Professional, sierp. 2003, ISBN: 978-0-321-12521-7.

[5] Foote, B. i Yoder, J., „Big Ball of Mud", wrz. 2003.

[6] Fowler, M., *Patterns of Enterprise Application Architecture*, English, 1st edition. Boston: Addison-Wesley Professional, list. 2002, ISBN: 978-0-321-12742-6.

[7] Fowler, M., *Refactoring: Improving the Design of Existing Code*, English, 2nd edition. Boston: Addison-Wesley Professional, list. 2018, ISBN: 978-0-13-475759-9.

[8] Kamiński, K., *Moving towards Micronaut*, en, list. 2021. adr.: https://blog.allegro.tech/2021/11/micronaut.html (term. wiz. 11. 12. 2023).

[9] Martin, R. C., *Clean Code: A Handbook of Agile Software Craftsmanship*, English, 1st edition. Upper Saddle River, NJ Munich: Pearson, sierp. 2008, ISBN: 978-0-13-235088-4.

[10] Martin, R. C., *Clean Architecture: A Craftsman's Guide to Software Structure and Design*, English, 1st edition. London, England: Pearson, wrz. 2017, ISBN: 978-0-13-449416-6.

[11] Masse, M., *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*, English, 1st edition. O'Reilly Media, paź. 2011.

[12] *Microservices - The State of Developer Ecosystem in 2022 Infographic*, en. adr.: https://www.jetbrains.com/lp/devecosystem-2022 (term. wiz. 31. 10. 2023).

[13] Mikuta, K., *Vanilla JS is not dead! Microfrontends without web performance issues.* en, list. 2022. adr.: https://blog.allegro.tech/2022/11/vanilla-js-is-not-dead.html (term. wiz. 11. 12. 2023).

[14] Newman, S., *Building Microservices: Designing Fine-Grained Systems*, English, 2nd edition. Beijing: O'Reilly Media, wrz. 2021, ISBN: 978-1-4920-3402-5.

[15] Pereira, R. i in., „Energy efficiency across programming languages: how do energy, time, and memory relate?", w *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering*, ser. SLE 2017, New York, NY, USA: Association for Computing Machinery, paź. 2017, s. 256–267, ISBN: 978-1-4503-5525-4. DOI: 10.1145/3136014.3136031. adr.: https://doi.org/10.1145/3136014.3136031 (term. wiz. 11. 12. 2023).

[16] *Stack Overflow Developer Survey 2023*, en. adr.: https://survey.stackoverflow.co/2023/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2023 (term. wiz. 10. 12. 2023).

[17] Varanasi, P., *Go at uber*, en, kw. 2016. adr.: https://www.slideshare.net/RobSkillington/go-at-uber (term. wiz. 11. 12. 2023).

[18] Vernon, V., *Implementing Domain-Driven Design*, English, 1st edition. Upper Saddle River, NJ: Addison-Wesley Professional, lut. 2013, ISBN: 978-0-321-83457-7.

[19] Walls, C., *Spring in Action, Sixth Edition*, English, 6th ed. edition. Shelter Island, NY: Manning, mar. 2022, ISBN: 978-1-61729-757-1.

[20] *What Is Your Definition of Software Architecture*, en, grud. 2010. adr.: https://insights.sei.cmu.edu/library/what-is-your-definition-of-software-architecture/ (term. wiz. 31. 10. 2023).