Przemysław Dudycz 212787

**4. Przegląd technologii**

Wybór technologii w jakiej będzie wytwarzana aplikacja jest bardzo ważnym etapem, w ramach którego podejmowana jest decyzja o konkretnej strukturze rozwiązania oraz mechanizmy, które są w stanie nam pomóc rozwiązać najważniejsze problemy. Biorąc pod uwagę ilość możliwości dostępnych na rynku, możemy zastosować wiele podejść, które w późniejszym czasie bardzo kształtują to, jak działa dana aplikacja. Wybrana technologia w dużym stopniu także wpływa na architekturę, a co za tym idzie, jakość tworzonego kodu.

Patrząc z perspektywy wyboru technologii dla aplikacji przedstawionej w ramach niniejszej pracy, trzeba spojrzeć z perspektywy trzech warstw:

1. Warstwy prezentacyjnej (klienckiej)
2. Warstwy serwerowej
3. Warstwy bazodanowej

Dla każdej z wyżej wymienionych części, trzeba dobrać odpowiadające i spełniające większość wymagań frameworki oraz podejścia, które pomogą w rozwiązaniu wszystkich problemów pojawiających się przy implementacji.

**4.1. Warstwa prezentacyjna (kliencka)**

Pierwszą wyszczególnioną warstwą jest warstwa prezentacyjna. Jest ona odpowiedzialna za końcowy etap działania aplikacji – intuicyjny interfejs czy wyświetlanie danych w sposób przejrzysty dla użytkownika. Całość musi być ułożona tak, żeby dany użytkownik nie był zagubiony korzystając z aplikacji. Jako że jest to aplikacja webowa, podstawą, która będzie wykorzystana jest HTML oraz CSS wraz z Bootstrapem, który będzie odpowiedzialny za stylowanie i ogólny wygląd strony. Wyborem jakiego należy wykonać w tej warstwie jest framework javascript’owy, który będzie odpowiedzialny za komunikację klient-serwer oraz logikę po stronie klienta. Jest także możliwość użycia języku Javascript bez żadnej nakładki, jednakże dodanie frameworka daje dużo plusów. Usprawnieniami, które uzyskujemy używając frameworku javascript’owego po stronie klienta jest dużo mniejsza ilość kodu, jego lepsze ustrukturyzowanie oraz poukładanie. Ważne jest też to, że użyte framework’i często posiadają mechanizmy, które zwiększają bezpieczeństwo wytwarzanego produktu. Poniżej są zaprezentowane niektóre javascript’owe frameworki oraz ich plusy i minusy.

|  |  |  |
| --- | --- | --- |
| **Nazwa** | **Plusy** | **Minusy** |
| React/Redux | * Dojrzały framework z dużą ilością wsparcia środowiska deweloperów w internecie * Mocne rozgraniczenie pomiędzy modelami, a kodem prezentującym modele * Łatwość w pisaniu testów jednostkowych do tworzonych komponentów * Komponenty mogą być łatwo użyte raz jeszcze * Zmiana danych powodują automatyczne odświeżenie interfejsu użytkownika | * Możliwe trudności w nauce * Narzucony wzorzec używający komponentów * React nie jest w pełni framework’iem ponieważ nie posiada routingu czy zarządzania modelami i wymaga doświadczenia by dobrać jeszcze jedenego mechanizmu zapewniającego te funkcjonalności. W tym wypadku dopełnieniem jest Redux |
| Angular | * Dojrzały framework z dużą ilością wsparcia środowiska * Mechanizm komponentów pozwalający na użycie kodu kontrolki w różnych miejscach aplikacji * Użycie komponentów wprost bez konieczności użycia nakładki * Możliwość wykorzystania TypeScript’u i jego atutów * Wysoka wydajność w prezentowaniu danych na interfejsie użytkownika * Angular CLI pozwalający na szybkie tworzenie kodu | * Możliwe trudności w nauce * Zarządzanie DOM’em bezpośrednio powoduje lekkie spadki w efektywności wyświetlania strony |
| Blaze | * Łatwy do nauki * Intuicyjny mechanizm templatów * Bardzo dobrze zintegrowany z platformą Meteor | * Trudny do refaktorowania * Małe możliwości * Mała efektywność prezentacji |

Na jej podstawie powyższej tabeli, w aplikacji wykonywanej w ramach danej pracy, zostanie użyty framework React wraz z Redux’em. Framework ten i Angular są znacznie lepsze od Blaze’a w ramach jakości kodu, możliwości testowania funkcjonalności oraz wydajności. Porównując Angulara z Reactem łatwo dochodzi się do wniosku, że nie ma lepszego z pośród tych dwóch. Każdy ma swoje dobre i złe strony i trzeba odpowiednio dobrać je do aplikacji.

**4.2. Warstwa serwerowa**

Drugą warstwą w której należy dobrać odpowiednie mechanizmy, jest warstwa serwerowa. W tym przypadku mamy do wyboru różne języki i podejścia, które w dość znaczny sposób narzucają nam strukturę i architekturę kodu, jaką będziemy posiadać w naszej aplikacji. Wybierając odpowiednią technologię, dosyć ważną rzeczą jest dokumentacja oraz ilość pomocy, którą możemy uzyskać w internecie. Najbardziej aktualnymi i popularnymi rozwiązaniami, które chciałbym w tej pracy porównać są to rozwiązania z platformy .NET, rozwiązanie Javascriptowe, czyli NodeJS oraz Java.

ASP.NET Core jest to platforma wytwarzana przez firmę Microsoft, pozwalająca na tworzenie aplikacji webowych o różnych architekturach i możliwościach. Możliwe jest stworzenie tradycyjnej aplikacji

NodeJS jest to Javascriptowy framework służący do tworzenia warstwy serwerowej. Opiera się on na zdarzeniach oraz słuchaczach zdarzeń. Pomimo, że w teorii NodeJS jest jednowątkowy, dzięki pętli zdarzeń potrafi on wykonać wiele żądań w jednym momencie. Dzięki temu, NodeJS jest bardzo efektownym i elastycznym frameworkiem pozwalającym na budowanie dużych i skalowalnych systemów.

Bazując na powyższych opisach, można dojść do wniosku, że nie da się wprost porównać obydwu podejść ponieważ różnią się one w swoich fundamentach. Wybierając technologię do projektu potrzeba kierować się wymaganiami i potrzebami. Dla aplikacji posiadających głównie strony statyczne, dobrym rozwiązaniem jest tradycyjne podejście, ponieważ posiada ono mechanizmy optymalizujące, które powodują, że aplikacja działa wydajniej. Jednakże jeżeli produkt posiada wiele wielowątkowości oraz zmian w czasie rzeczywistym podejście jakie jest użyte w NodeJS jest bardziej optymalnym podejściem pozwalającym na ominięcie dużej ilości problemów jakie napotkalibyśmy przy użyciu tradycyjnych mechanizmów.

**4.3. Warstwa bazodanowa**

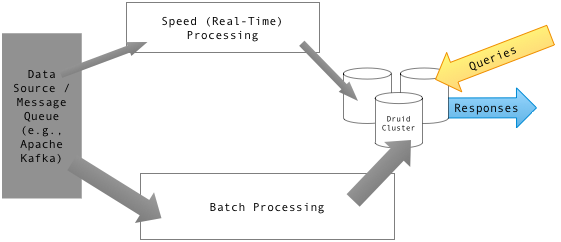
Ostatnim wyborem technologicznym, jest wybór w warstwie bazodanowej. Biorąc pod uwagę specyfikę, wybór jaki jest dokonany w tej części jest kluczowy dla efektywności działania aplikacji tworzonej w ramach tej pracy. Cała dyskusja wyboru technologii bazodanowej sprowadza się do wyboru pomiędzy dwoma podejściami – SQL oraz noSQL, czy inaczej bazy relacyjne i bazy nierelacyjne.

Bazy relacyjne reprezentują dane jako wiersze i kolumny w tabelach. Posiadają wiele mechanizmów optymalizujących, takich jak indeksy, które pozwalają na efektywną analizę danych w tabelach, obsługę skomplikowanych transakcji oraz zapytań łączących dane z różnych tabel. Oczywistą rzeczą jest to, że modele za pomocą których aplikacja łączy się z bazą muszą być zgodne ze schematem odpowiadającej tabeli.

W przypadku baz nierelacyjnych, uzasadnionym użyciem tego rozwiązania jest sytuacja gdy posiadamy duże ilości rozproszonych danych i użycie baz relacyjnych byłoby dużym spowolnieniem. Druga sytuacja, która pojawia się w aplikacji tworzonej w ramach niniejszej pracy, jest problem z odczytem danych z bazy. Dla baz nierelacyjnych wyciągniecie obiektu jest dużo szybsze niż wyciągniecie danych z tabeli za pomocą frameworka ORM, zmapowania go na model i dopiero przedstawienia w aplikacji.

Jednakże, jak się okazuje, wybór oraz użycie jedynie jednego serwera oraz podejścia bazy danych jest nieefektywne dla aplikacji, która musi działać z dużą ilością danych, raportów oraz analiz. Łącząc to z potrzebą odświeżania w czasie rzeczywistym, można dojść do wniosku, że ani podejście relacyjne, ani nierelacyjne nie dostarczają nam mechanizmów pozwalających stworzyć potrzebny produkt. Z pomocą przychodzi jeden z wzorców oraz mechanizmów Business Intelligence, zwany Lambda Architecture.

Wzorzec Lambda Architecture opiera się na podziale warstwy bazodanowej na jeszcze więcej części. Architektura, która zostanie stworzona, potrzebuje być efektywna, odporna na błędy oraz skalowalna. Biorąc pod uwagę ilość danych, które mogą trafić do aplikacji, potrzeba stworzyć parę elementów zapewniających nam dostęp do odpowiednich informacji w akceptowalnym dla użytkownika czasie.



Rys.4.1. Diagram architektury Lambda.

Rys.4.1. przedstawia ogólny zarys i idee, która ukazuje nam na czym polega wyżej wspomniana architektura. Posiadamy w niej dwie główne warstwy - Speed oraz Batch. Pierwsza z nich, jest to miejsce, gdzie ładowane są dane bez obróbki, które od razu są gotowe do zaczytania. Drugą warstwą, jest to warstwa Batch, w której zaimplementowane są procedury przetwarzające, łączące oraz przygotowujące raporty dla aplikacji. Odpowiedzią na żądania klienta są widoki składające się z obydwu części. Podejście takie, pomaga nam mieć dostęp do danych w czasie rzeczywistym za pomocą informacji z części Speed oraz do zaawansowanych raportów przygotowanych przez część Batch, które jednak są dostępne z pewnym opóźnieniem pozwalającym te raporty wygenerować.

Wybór konkretnego serwera SQL zależy z którą warstwą chcemy pracować. Do zadań z części Speed, najlepszym rozwiązaniem jest użycie bazy noSql, gdyż pozwala nam na szybsze dostanie się do informacji, które są nam potrzebne. W drugim przypadku, biorąc pod uwagę, że dany serwer SQL będzie musiał sobie poradzić z dużą ilością danych oraz wywoływaniem na nich skomplikowanych procedur, najlepszym wyborem jest użycie serwera SQL, ponieważ posiada on wiele mechanizmów optymalizujących takie operacje.