**4. Przegląd technologii**

Wybierając technologię do pisania aplikacji tworzonej w ramach tej pracy, pierwszym pytaniem jakim się nasuwa jest pytanie o platformę którą należy użyć do danego zadania. Jeżeli chodzi o aplikacje webowe, jest dostępnych wiele możliwości: .NET, Ruby on Rails. Biorąc jednak pod uwagę specyfikę aplikacji, potrzeba dobrać mechanizmy i narzędzia które w akceptowalnym czasie odświeżają dane zmieniające się w czasie rzeczywistym oraz skuteczny sposób przesyłają duże porcję informacji z warstwy serwerowej na warstwę kliencką. Jeżeli spojrzymy na tradycyjne podejście do wytwarzania aplikacji webowych, takie jak w platformie .NET od razu nasuwają się pytania jak tym wymaganiom sprostać. Biorąc pod uwagę, że jest to tradycyjne wysyłanie danych, które zazwyczaj musi zainicjować klient szybko można dojść do wniosku że nie jest to dobry wybór to tego rodzaju aplikacji. Platformą jaka została wybrana do aplikacji tworzonej w ramach niniejszej pracy, która umożliwia radzenie sobie ze zmianą informacji w czasie rzeczywistym, jest Meteor.

Meteor jest to platforma używająca języku Javascript po stronie klienta jak i po stronie serwera. Pozwala ona na bardzo efektowną wymianę danych pomiędzy tymi dwoma warstwami. Zmiana w modelu danych powoduje automatyczne odświeżenie na widoku i głównie przez tą funkcjonalność platforma ta jest wykorzystywana w aplikacjach webowych czasu rzeczywistego.

Patrząc z perspektywy wyboru technologii w ramach Meteora, 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 tworzonej w ramach tej pracy. Jako że jest to aplikacja webowa, podstawą, która będzie wykorzystana jest HTML oraz CSS. Jednakże jeżeli chodzi o stylowanie i ogólny wygląd strony użyty będzie gotowy wzorzec używający Bootstrap’a. 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 oryginalnego Javascript’u, jednakże dodanie dodatkowej nakładki na ten język daje dużo plusów. Pierwszym usprawnieniem użycia frameworku javascript’owego po stronie klienta jest fakt, że ilość kodu znacznie się zmniejsza, jest on ustrukturyzowany oraz lepiej poukładany, a także znacznie wzrasta prędkość wytwarzania aplikacji. 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 |

Powyższa tabela zawiera wszystkie framework’i, które udostępnia platforma Meteor w warstwie prezentacyjnej. Na jej podstawie, 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 jednakże, wybór jest dokonywany wcześniej ponieważ platforma Meteor narzuca nam technologię NodeJS jako jedyny wybór w tej warstwie. Alternatywą dla podejścia Meteora jest tradycyjny ASP.NET.

APS.NET jest to tradycyjne narzędzie do tworzenia aplikacji webowych stale rozwijane przez firmę Microsoft. Podejście jakie charakteryzuje te narzędzie, jest najbardziej optymalne przy aplikacjach webowych ze statycznymi stronami. Przy tym podejściu, dużą trudnością jest napisanie aplikacji asynchronicznej, która posiada w sobie dużo żądań do baz danych czy zewnętrznych serwisów.

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 aplikacji, wybór jaki jest dokonany w tej części jest kluczowy dla efektywności działania aplikacji tworzonej w ramach tej pracy. Podobnie jak w części serwerowej Meteor ma gotową i zsynchronizowaną bazę danych w technologii MongoDB. Jednakże, nie jest to rozwiązanie narzucane i przy odrobinie pracy można wybrać inną drogę. 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.