**4. Przegląd technologii**

Wybór technologii, jakie będą użyte w procesie wytwarzania projektowanej aplikacji jest bardzo ważnym etapem, wpływającym w znacznym stopniu na kształt tworzonego rozwiązania. Bez podstawowej wiedzy o istniejących narzędziach, których można użyć, oraz możliwościach tych narzędzi, bardzo trudno jest o zaprojektowanie i stworzenie dobrze działającej aplikacji. Wpływ wybieranych technologii, można zaobserwować w architekturze, jakości kodu oraz co najważniejsze, efektywności działania aplikacji. Patrząc z perspektywy użytkownika docelowego, efektywność działania jest najważniejszym czynnikiem wpływającym na zadowolenie z korzystania z dostarczonego produktu. W aplikacjach Business Intelligence, w których występuje przetwarzanie ogromnej liczby danych, efektywność aplikacji jest bardzo trudna do osiągnięcia dlatego wybór odpowiedniego narzędzia, pozwalającego przetworzyć te struktury danych jest bardzo istotna.

Mówiąc o wyborze technologii dla aplikacji przedstawionej w ramach niniejszej pracy, trzeba spojrzeć na problem z perspektywy trzech warstw:

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

Warstwa prezentacyjna dotyczy wszystkiego, co zobaczy użytkownik docelowy. Wybór w tej warstwie jest wyborem z bardzo szerokiego spectrum rozwiązań dostępnych na rynku. Różnice na jakie można napotkać, są to głównie różnice w architekturze tworzonego kodu, dlatego wskazując na któreś z rozwiązań, powinno kierować się wygodą w trakcie pracy z daną architekturą. Drugą warstwą jest warstwa serwerowa, która odpowiada za część logiki biznesowej, tworzenie zapytań do zewnętrznych serwisów oraz kontakt z bazą danych. Wybór w tej części aplikacji powinien być kierowany łatwością stworzenia połączenia z wybranym serwerem bazodanowym oraz efektywnością transferu danych z warstwą kliencką. Ostatnią warstwą, którą należy wyróżnić przy wyborze technologii, jest warstwa bazodanowa. Z perspektywy wcześniej wspomnianej efektywności aplikacji, w tym przypadku jest to wybór najbardziej znaczący, wpływający na prędkość działania dostarczanych funkcjonalności. Jednakże wskazanie technologii w części bazodanowej nie ogranicza się tylko do wskazania odpowiedniego serwera bazodanowego, ale także wyboru architektury, która będzie zaimplementowana za pomocą tego serwera, pozwalającej na przetwarzanie w akceptowalnym czasie dużych zbiorów danych.

W poniższym rozdziale będą opisane technologie, pomiędzy którymi następował wybór, w każdej z trzech, wyżej wymienionych warstw.

**4.1. Warstwa prezentacyjna (kliencka)**

Pierwszą wyszczególnioną warstwą jest warstwa prezentacyjna. Jest ona odpowiedzialna za etap działania aplikacji bezpośrednio związany z użytkownikiem docelowym – intuicyjny interfejs oraz wyświetlanie danych w sposób przejrzysty. Całość musi być ułożona tak, żeby dany użytkownik nie był zagubiony korzystając z aplikacji. Zadowolenie użytkownika końcowego bezpośrednio jest związany z warstwą kliencką, dlatego należy przywiązać dużą uwagę w jaki sposób prezentujemy nasze funkcjonalności. Jako że aplikacja wytwarzana w ramach tej pracy, jest to aplikacja webowa, podstawą, która będzie wykorzystywana to HTML oraz CSS z rozszerzeniem SASS. Wykorzystany zostanie także framework o nazwie Bootstrap w celu łatwiejszego stylowania strony za pomocą gotowych komponentów dostarczonych przez ten framework. Wybór, jaki dokonujemy w tej warstwie, jest to wybór pomiędzy mechanizmami zapewniającymi wyświetlanie danych pochodzących z serwera oraz obsługującymi interakcję użytkownika ze stroną. W ramach tego rozdziału wybór ten będzie ograniczony do narzędzi związanych z językiem Javascript.

Javascript jest to język skryptowy, stworzony przez Brendana Eicha z firmy Netscape, z myślą o przeglądarkach internetowych, pozwalający na implementację zachowania strony w związku z interakcją użytkownika. W ostatnich latach, język ten zyskał na popularności, wynikiem czego jest przeniesienie mechanizmów związanych z tym językiem na stronę serwera (NodeJS). Pomimo rozszerzenia swojego wykorzystania na stronę serwerową, język ten ciągle jest kojarzony z kodem klienckim [1]. Pierwszym pytaniem, jakie pojawia się przy wyborze technologii związanej z tą warstwą, jest pytanie o słuszność wyboru jakiegokolwiek frameworku javascriptowego. Biorąc pod uwagę efektywność działania strony, podstawowe badania wykazują, że sam javascript jest dużo szybszy w działaniu, niż jego odpowiedniki [2], jednakże jeżeli weźmiemy pod uwagę współczesne komputery jakimi posługują się użytkownicy, dla większości aplikacji, różnica ta będzie niezauważalna. Jeżeli weźmiemy pod uwagę jakość tworzonego kodu, prędkość wytwarzania aplikacji oraz łatwość z jaką dane są transferowane z serwera na stronę html, użycie frameworka wypada dużo lepiej niż wykorzystanie czystego Javascriptu.

Ilość dostępnych frameworków javascriptowych jest bardzo duża. Porównanie tych narzędzi należy dokonywać w perspektywie architektury, efektywności, łatwości wytwarzania produktu oraz aktywności społeczności powiązanej z danym frameworkiem. Biorąc pod uwagę popularność frameworków w 2017 roku, do porównania wybrane zostały trzy javascriptowe frameworki – Angular, ReactJs oraz EmberJs [3].

**Angular** jest to jest to open-sourcowy framework, stworzony przez firmę Google. Opiera się on na rozszerzeniu Javascriptu – TypeScriptcie. Dzięki temu jesteśmy w stanie tworzyć kod w pełni obiekowo zoorientowany, używając mechanizmu klas, dziedziczenia czy interfejsów. Angular jest narzędziem multiplatformowym, pozwalającym na tworzenie warstwy klienckiej za pomocą reużywalnych komponentów, tworząc w ten sposób kod czytelny oraz łatwy do utrzymania [4].

**ReactJs** jest to biblioteka, stworzona przez społeczność facebookową, opierająca się na notacji JSX. Pozwala ona na renderowanie podkomponentów w kodzie HTML, co pozwala na tworzenie reużywalnego kodu. Patrząc z perspektywy architektury warstwy klienckiej, ReactJs nie posiada mechanizmów do tworzenia odpowiedniej struktury. Zazwyczaj dodawane jest jeszcze jedno narzędzie – najczęściej Flux – pozwalające na stworzenie takowej architektury, jednakże połączenie tych dwóch mechanizmów komplikuje naukę tworzenia aplikacji za pomocą Reacta [5][6].

**EmberJs** jest to open-sourcowy framework, stworzony przez Yehuda Katza. Opiera się on na architekturze MVC i pozwala na budowanie aplikacji typu „single page”. Posiadane mechanizmy dwubiegunowego wiązania danych oraz podejścia bazującego na tym, że URL przetrzymuje stan aplikacji pozwala na tworzenie skalowalnych systemów[7][8].

W tabeli Tab. 1.1 zastawione są różnice w znaczących mechanizmach dla trzech, wyżej wymienionych frameworków.

Tab 1. 1 Wady i zalety frameworków javascriptowych (Źródło: [9])

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Angular** | **ReactJs** | **EmberJs** |
| **Zalety** | * Efektywność działania * Renderowanie po stronie serwera * Posiada wszystkie mechanizmy pozwalające na zbudowanie aplikacji * Wszystkie aktualizacje językowe są od razu dostępne * Stworzony kod jest łatwy do testowania * Łatwość tworzenia reużywalnego kodu | * Efektywność działania * Renderowanie po stronie serwera * Prostota * Podejście biblioteki – wybiera się mechanizmy potrzebne deweloperowi * Wszystkie aktualizacje językowe są od razu dostępne | * Efektywność działania * Renderowanie po stronie serwera * Posiada wszystkie mechanizmy pozwalające na zbudowanie aplikacji * Dobra dokumentacja * Łatwość tworzenia reużywalnego kodu |
| **Wady** | * Słaba dokumentacja * Twórcy Angulara nie dbają o kompatybilność nowych wersji frameworka względem poprzednich. Trudności w uaktualnieniu frameworka | * Brak mechanizmów tworzenia architektury aplikacji – użycie przykładowo Fluxa mocno utrudnia naukę wytwarzania aplikacji * Zła renoma notacji JSX | * Mniejsza społeczność internetowa * Skomplikowany w użyciu przy nietypowych przypadkach |

Porównując wszystkie trzy frameworki, można dojść się do wniosku, że ciężko jest wybrać najlepsze narzędzie, zapewniające rozwiązanie na problemy, które można spotkać w trakcie implementacji. Każdy z tych trzech frameworków posiada swoje podejście i często wybór danego rozwiązania, będzie kierowany wygodą w użyciu danego frameworka. Biorąc pod uwagę wymienione wady i zalety (Tab. 1.1), w aplikacji tworzonej w ramach niniejszej pracy, do stworzenia warstwy klienckiej zostanie wybrany framework Angular.

**4.2. Warstwa serwerowa**

Drugą warstwą dla której należy dobrać odpowiednie mechanizmy, jest warstwa serwerowa. W tym przypadku są do wyboru różne języki i podejścia, które w dość znaczny sposób narzucają projektowi strukturę i architekturę kodu, jaką będzie posiadać aplikacja. Wybierając odpowiednią technologię, ważną rzeczą jest dokumentacja oraz ilość pomocy, którą można uzyskać w internecie. Najbardziej aktualnymi i popularnymi rozwiązaniami, które będą w tej pracy porównane są 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**

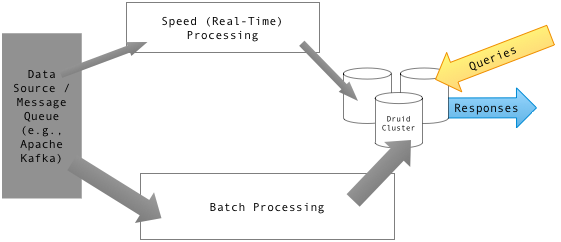
Ostatnim wyborem technologicznym, jest wybór w warstwie bazodanowej. Biorąc pod uwagę specyfikę, wybór jaki jest dokonany w tej części jest kluczowy dla efektywności działania aplikacji tworzonej w ramach tej pracy. Cała dyskusja wyboru technologii bazodanowej sprowadza się do wyboru pomiędzy dwoma podejściami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