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 rozpatrywaną warstwą aplikacji tworzonej w ramach niniejszej pracy jest warstwa serwerowa. W tym miejscu należy dobrze zastanowić się nad wyborem technologii z jaką będzie się pracowało ponieważ duża część logiki biznesowej oraz transfer danych z bazy danych do klienta odbywa się właśnie na serwerze. Operacje te mają znaczący wpływ na efektywność aplikacji, dlatego narzędzia jakie wybierzemy, muszą w odpowiedni sposób radzić sobie z tymi zadaniami. Rozwiązaniami jakie będą porównywane w tym miejscu, są to rozwiązania z platform .NET, Java oraz rozwiązanie javascriptowe - Node.js.

Platforma .NET jest to dobrze znana platforma rozwijana przez firmę Microsoft. Opiera się ona głównie na języku C#, jednak dzięki narzędziu Common Language Runtime deweloper ma możliwość programować także w innych językach programowania. Platforma .NET oferuje bardzo szerokie spectrum narzędzi dla tworzenia różnego rodzaju zadań i projektów. Narzędziem do tworzenia aplikacji webowych jest framework ASP.NET, który dostarcza wiele mechanizmów służących do tworzenia aplikacji przeznaczonych do różnego rodzaju zadań. Biorąc pod uwagę, że warstwa serwerowa ma jedynie przetwarzać dane otrzymane z bazy danych i przesyłać je dalej do warstwy prezentacyjnej, rozwiązaniem jakie najlepiej pasuje do tego scenariusza, jest rozwiązanie RESTowe – aplikacja ASP.NET Core Web API [10].

Drugą platformą, konkurencyjną dla .NET, jest platforma Java. Platforma opiera się głównie na języku Java. Jest to rozwiązanie open-sourcowe z dużą ilością użytkowników biorących czynny udział w rozwoju platformy Java w internecie. Frameworkiem odpowiedzialnym za aplikacje webowe w ramach tej platformy jest Java EE. W ramach tego frameworka, również istnieje możliwość zbudowania RESTowej aplikacji, która jest rozwiązaniem, jakie aplikacja tworzona w ramach niniejszej pracy, będzie wykorzystywała[11].

Node.js jest to środowisko pozwalające na uruchomienia kodu javascriptowego po stronie serwera. Środowisko te zostało stworzone przez Ryana Dahla. Node.js działa na silniku javascriptowym Chrome V8. Opiera się on na zdarzeniach oraz słuchaczach zdarzeń. Pomimo, że w teorii Node.js jest jednowątkowy, dzięki pętli zdarzeń potrafi on wykonać wiele żądań w jednym momencie. Dzięki temu, Node.js jest bardzo efektownym i elastycznym narzędziem, pozwalającym na budowanie dużych i skalowalnych systemów [12].

Porównanie trzech, wyżej wymienionych platform jest zadaniem bardzo trudnym. Biorąc pod uwagę różnicę podejścia, użytkownik nie jest w stanie jednoznacznie wskazać środowisko, które w będzie sprawowało się lepiej w każdej sytuacji. Biorąc pod uwagę fakt, że każda z tych platform identyfikuje się konkretnym językiem programowania, z pewnością wybór będzie kierowany wygodą oraz subiektywną oceną języka. Czynnikiem, który z pewnością jest znaczący przy

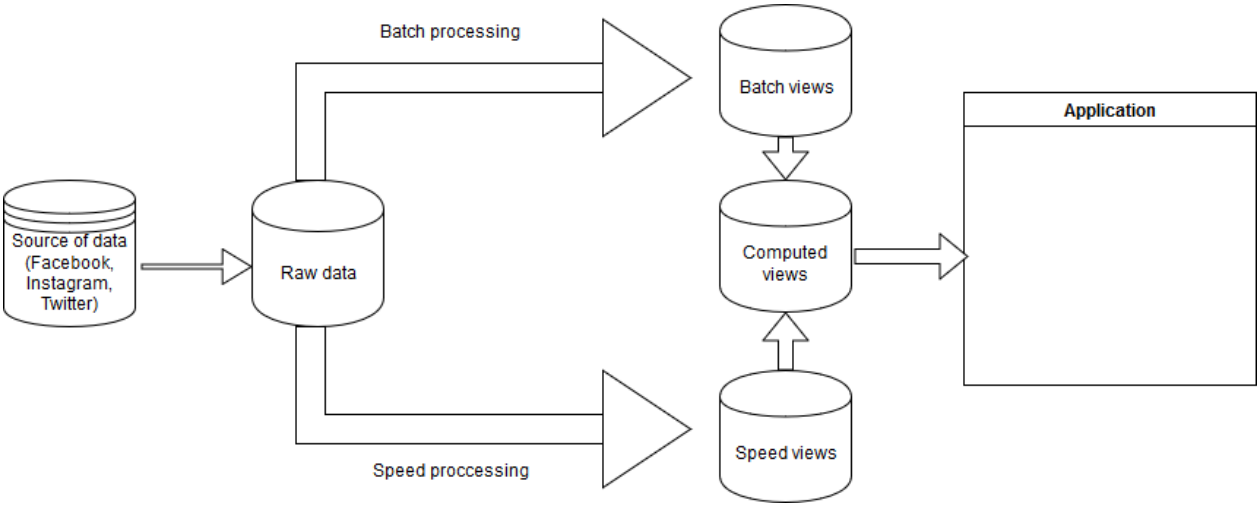
wyborze podejścia, jest wsparcie społeczności internetowej w ramach każdego z narzędzi. Jednakże wszystkie trzy wyżej wymienione, biorąc pod uwagę, że są w tym momencie najbardziej popularnymi rozwiązaniami dla aplikacji serwerowych, posiadają bardzo duże wsparcie w internecie. Dla aplikacji tworzonej w ramach tej pracy, platforma jaka została wybrana, jest to platforma .NET. Wymagania postawione dla warstwy serwerowej, najlepiej będą realizowane za pomocą aplikacji w architekturze REST – ASP.NET Core WEB API.

4.3. Warstwa bazodanowa

Ostatnim wyborem technologicznym, jest wybór w warstwie bazodanowej. W aplikacji tworzonej w ramach niniejszej pracy, procesowanie danych oraz radzenie sobie z ilością wczytywanych informacji do bazy danych jest głównym problemem do rozwiązania.

Sposoby przetrzymywania i zapisywania danych znacznie zmieniły się na przestrzeni ostatnich kilkudziesięciu lat. Wpierw był to zapis i odczyt z pliku, jednakże ze względu na rosnącą liczbę danych nie był to wystarczający sposób. Dlatego w pewnym momencie przedstawiono serwery bazodanowe wraz ze środowiskami zarządzającymi tymi serwerami. Optymalizacje oraz ulepszenia tych serwerów pozwalają na skuteczne radzenie sobie z zapisem i odczytem danych dla większości aplikacji. Jednakże rozrost sieci, serwerów oraz aplikacji, które co raz częściej posiadają wymóg radzenia sobie z ogromną liczbą informacji spowodował, że nawet klasyczne użycie bazy danych nie wystarcza to optymalnego przetwarzania tych informacji. Ze względu na to powstały schematy i rozwiązania tak zwanych hurtowni danych pozwalających na analizę i przetworzenie duży ilości danych. Zaistniały także rozwiązania z grupy nazwanej NOSQL, które w pewnym sensie rozwiązały wiele problemów, jednakże nie są skuteczne we wszystkich przypadkach. Do problemu ilości przetwarzanych danych, dochodzi także kwestia skalowalności tworzonego systemu, które przy klasycznych serwerach SQL jest bardzo trudne do utrzymania. Ze względu na wszystkie, wyżej wymienione kwestie, stworzona została architektura nazwaną Architekturą Lambda.

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 dostęp do odpowiednich informacji w akceptowalnym dla użytkownika czasie.



Rys.4.1. Diagram architektury Lambda (Źródło: wkład własny).

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. Warstwa Speed przygotowuje raporty z informacji, które zostały zaczytane w czasie od ostatniego przetworzenia przez warstwę Batch do czasu aktualnego. 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.

Implementacja Architektury Lambda może zostać dokonana na wiele sposobów. Ważną rzeczą, o której należy pamiętać, jest dobranie odpowiednich technologii do warstwy Batch i warstwy Speed. Warstwa Batch potrzebuje narzędzia posiadającego zoptymalizowane operacje procesowania i wyliczania potrzebnych nam widoków na podstawie dostępnych informacji. Natomiast warstwa Speed, powinna zostać zaimplementowana za pomocą narzędzia, które udostępnia dane w jak najkrótszym czasie.

Procesowanie oraz radzenie sobie z dużą ilością danych, ciągle jest dużym problemem dla tworzonych systemów informatycznych. By ułatwić to zadanie, zostały stworzone mechanizmy takie jak Hadoop, Cassandra czy framework MapReduce. Rozwiązaniem próbującym rozwiązać podobne problemy, jest Apache Spark. Apache Spark jest to open-sourcowa platforma, stworzona na Uniwersytecie Kalifornii. Platforma ta stosuje klastrowe przetwarzanie danych, opierając się na rozproszonych zbiorach danych RDD (Resilient Distributed Dataset). Posiada ona także interfejsy API dla różnych języków programowania, w których można pisać różne operacje przetwarzania, procesowania czy transformacji dostępnych nam danych.

Dla aplikacji tworzonej w ramach niniejszej pracy, do przetrzymywania danych, z którym będzie kontaktowała się warstwa serwerowa, będzie wykorzystany serwer PostgreSQL. Procesowanie w warstwie Batch, będzie dokonywane za pomocą Apache Sparka oraz języka Scala, natomiast warstwa Speed, będą to widoki tworzone za pomocą PostgreSQLa.