**A blue and black logo

Description automatically generatedWydział Finansów i Zarządzania**

**Kierunek: Informatyka**

**Jakub Czarnecki**

(nr albumu: 81495)

**Nowoczesna rozwiązanie Webowe wspierające pracę położnych środowiskowych i doradców laktacyjnych w sektorze medycznym**

**LUCYCOVER**

#### Inżynierska praca projektowa

**Opiekun merytoryczny**

mgr inż. Wojciech Barczyński

Wrocław 2024

Table of Contents

[WSTĘP 6](#_Toc179270844)

[1. Analiza biznesowa – założenia projektowe aplikacji oraz problemy do rozwiązania 7](#_Toc179270845)

[1.1 Funkcjonalność – w jakim zakresie projekt powinien wspierać użytkownika. 7](#_Toc179270846)

[1.2 Dostępność –Problem różnicy wieku, urządzeń mobilnych oraz niepełnosprawności 7](#_Toc179270847)

[1.3 Bezpieczeństwo – jak przetwarzać oraz jak je przechowywać dane w bezpieczny sposób. 8](#_Toc179270848)

[Podsumowanie 9](#_Toc179270849)

[2. Omówienie wykorzystanych technologii 9](#_Toc179270850)

[2.1 Ogólny opis wykorzystanych technologii w projekcie. 9](#_Toc179270851)

[2.2 .NET – Platforma programistyczna od Microsoft 11](#_Toc179270852)

[2.3 C# - Główny język programowania wykorzystywany w technologii .NET 12](#_Toc179270853)

[2.4 EntityFramework – zaawansowane rozwiązanie ORM dla programistów .NET 12](#_Toc179270854)

[2.5 ASP .NET CORE – Platforma przeznaczona do budowy aplikacji internetowych 13](#_Toc179270855)

[2.6 AutoMapper – Mapowanie obiektów między wersjami aplikacji 14](#_Toc179270856)

[2.7. FluentValidator – walidacja danych .NET 14](#_Toc179270857)

[2.8. NLog – narzedzie do logowania danych w aplikacji 14](#_Toc179270858)

[2.9. JWT BEARER – narzędzie do uwierzytelniania użytkownika 14](#_Toc179270859)

[2.10. Swashbuckle/swawgger – narzędzie do automatycznego tworzenia dokumentacji API 15](#_Toc179270860)

[2.11. HTML 5 / CSS3 – Fundamenty warstwy prezentacji każdej aplikacji webowej. 16](#_Toc179270861)

[2.12. REACT – podstawowe narzędzie podczas tworzenia nowoczesnych interfejsów graficznych. 16](#_Toc179270862)

[2.13. VITE – świeży powiew przy tworzeniu aplikacji front-end 17](#_Toc179270863)

[2.14. React Redux – biblioteka do zarządzania stanem aplikacji 17](#_Toc179270864)

[2.15. React Router – technologia tworzenia routingu w aplikacjach react 18](#_Toc179270865)

[2.16. Tanstack Query – biblioteka wspierająca komunikację serwer-client 19](#_Toc179270866)

[2.17 FramerMotion – biblioteka do tworzenia animacji 19](#_Toc179270867)

[2.18. MailKit – biblioteka do obsługi protokołów poczty elektronicznej 20](#_Toc179270868)

[2.19. XUnit – testowanie aplikacji z rodziny .NET 20](#_Toc179270869)

[2.20. Git / GitHub – System kontroli wersji 21](#_Toc179270870)

[3. Implementacja aplikacji – budowa aplikacji w architekturze trójwarstwowej 22](#_Toc179270871)

[3.1 Implementacja warstwy prezentacji – aplikacja kliencka 22](#_Toc179270872)

[3.1.1 Omówienie wymogów wizualnych aplikacji 22](#_Toc179270873)

[3.1.2 Komunikacja komponentów w oparciu o technologie REDUX 24](#_Toc179270874)

[3.1.3 Omówienie routingu aplikacji z wykorzystaniem bibloteki React Router 27](#_Toc179270875)

[3.1.4 Zaimplemetowanie własnych Hook’s 29](#_Toc179270876)

[3.1.4 Komunikacja z backendem z wykorzystaniem React TanstackQuery 30](#_Toc179270877)

[3.1.5 Walidacja w warstwie prezentacji 31](#_Toc179270878)

[3.1.6 Zapewnienie nowoczesnego designu dzięki FramerMotion 32](#_Toc179270879)

[3.1.7 Zawansowany serwer aplikacji Vite 33](#_Toc179270880)

[3.2 Implementacja warstwy dostępu do danych 34](#_Toc179270881)

[3.2.1 Omówienie wymogów przechowywania danych 34](#_Toc179270882)

[3.2.2. Code First – podejście definiowania modeli baz danych 34](#_Toc179270883)

[3.2.2. Implementacja kontekstu baz danych w projekcie 37](#_Toc179270884)

[3.3.3 Wykorzystane Modele (DO NAPISANIA) 39](#_Toc179270885)

[3.3.4. Wykorzystanie wzorca projektowego „Repository” 39](#_Toc179270886)

[3.3.5. Data Transfer Object 42](#_Toc179270887)

[3.4. Implementacja warstwy logiki biznesowej (backend) 44](#_Toc179270888)

3) Implementacja aplikacji

3.2) Implementacja warstwy prezentacji (frontend)

3.2.1) Omówienie wymogów wizualnych aplikacji

**3.2.2) Komunikacja komponentów w oparciu o REDUX**

3.2.3) Omówienie routingu z wykorzystaniem React Router

3.2.4) Zaimplemetowanie własnych Hook’s

3.2.5) Komunikacja z backendem z wykorzystaniem React TanstackQuery

3.2.6) Walidacja w warstwie prezentacji.

3.2.7) Zapewnienie nowoczesnego designu dzięki FramerMotion

3.2.8) Zawansowany serwer aplikacji Vite

3.3) Implementacja warstwy dostępu do danych (bazy danych) oraz sposób komunikacji

3.3.1) Omówienie wymogów przechowywania danych

3.3.2) Code First – podejście definiowania modeli baz danych

3.3.3) Implementacja kontekstu bazy danych w projekcie

3.3.4) Wykorzystanie modele

3.3.5) Wykorzystanie wzorca projektowego „Repository”

3.3.6) Data transfer object (DTO)

3.4) Implementacja warstwy logiki biznesowej (backend)

3.4.1) Omówienie struktury aplikacji.

3.4.1) Podział aplikacji na Kontrolery,Serwisy,i modele

3.4.2) Zaimplementowane walidowanie danych

3.4.3) Wykorzystanie systemu dependency injection

3.4.4) Wykorzystanie własnych middleware w pipelinie

3.4.5) Walidacja z wykorzystaniem FluentValidator

3.4.6) Obsługa protokołów poczty elektronicznej z wykorzystaniem MailKit

3.4.7) Komunikacja z bazą danych

3.4.8) Wsparcie administratorów systemu zapewnione przez Logger

3.6) Omówienie autentykacji oraz autoryzacji użytkownika

3.6.1) Implementacja framework JWTBarer

3.6.2) Generowanie tokenu

3.6.3) Wymiana tokenu pomiędzy warstwami aplikacji

3.6.4) Dostęp do zasobów za pomocą autoryzacji endpointów

3.7) Omówienie problemu szyfrowania danych

3.7.1) Implementacja szyfrowania symetrycznego AES

3.7.2) Przechowywanie haseł w bazie danych

3.7.3) Szyfrowanie komunikacji pomiędzy warstwami z wykorzystaniem HTTPS

3.8) Omówienie sposobu testowania aplikacji (xUnit)

3.8.1) Zaimplementowanie testowania działania endpointów

3.8.2) Zaimplementowanie testowania serwisów

3.8.3) Zaimplementowania testowania mechanizmów szyfrujących

3.8.4) Zaimplementowanie testowania middleware

4) Zakończenie

5) Bibilografia

6) Spis rysunków

7) Streszczenie

# WSTĘP

W dzisiejszych czasach sektor medyczny stale ewoluuje, wprowadzając nowoczesne technologie w celu poprawy efektywności i bezpieczeństwa opieki nad pacjentami. Jednym z kluczowych wyzwań, przed którymi stoją placówki medyczne, są archiwa papierowe, które nie tylko spowalniają procesy pracy, ale także zwiększają ryzyko utraty danych oraz błędów w dokumentacji.

Celem niniejszej pracy dyplomowej jest zaprojektowanie i implementacja aplikacji dedykowanej dla sektora medycznego, której zadaniem będzie wspieranie pracy położnych środowiskowych oraz doradców laktacyjnych działających na terenie Jeleniej Góry. Ta webowa aplikacja będzie centralnym systemem tworzenia oraz przechowywania dokumentacji medycznej pacjentów, bazą zaleceń wystawionych po każdej wizycie domowej oraz komunikatorem pomiędzy pacjentem a obsługującym go lekarzem/położną.

Obecnie wiele lokalnych przychodni nadal polega na fizycznych archiwach papierowych, co prowadzi do nieefektywności oraz zagrożeń związanych z utratą danych. LucyCover – bo tak nazywa się nasz projekt ma na celu przyspieszenie pracy personelu medycznego, zwiększenie bezpieczeństwa danych oraz ochronę środowiska poprzez zmniejszenie zużycia papieru, tuszu i energii elektrycznej.

Dodatkowo aplikacja ta wesprze personel medyczny w planowaniu codziennej pracy. Dzięki zaimplementowanemu terminarzowi, lekarz/położna będzie mogła szybko i efektywnie zaplanować swój dzień pracy. Planując wizyty poszczególnych pacjentów system automatycznie będzie informował wszystkich interesariuszy o zaplanowanej wizycie co pozwoli zautomatyzować procesy planowania oraz kontaktu z pacjentem. Unikniemy również wielu błędów komunikacyjnych takich jak nie poinformowanie pacjenta o zaplanowanej wizycie lub też jej odwołaniu.

Dodatkowo, dzięki zaawansowanej walidacji danych wprowadzanych do systemu, eliminujemy możliwość błędów w dokumentacji, co jest kluczowe dla zapewnienia wysokiej jakości opieki pacjentom.

LucyCover jako aplikacja webowa działająca w sieci lokalnej placówki będzie również niezwykle bezpiecznym rozwiązaniem. Odizolowanie baz danych oraz aplikacji od sieci publicznej praktycznie uniemożliwi wykradnięcie danych przez zdalnych cyberprzestępców. Dostęp do aplikacji uzyskają jedynie autoryzowane osoby posiadające swoje osobiste dane uwierzytelniania a wszystkie przechowywane w bazie dane będą szyfrowane za pomocą specjalnych algorytmów.

Niniejsza praca dyplomowa tworzona jest z myślą o wykorzystaniu w sieciach lokalnych odizolowanych od sieci Ethernet w celach bezpieczeństwa (jw.). Nie mniej jednak pozostawiona została możliwość wdrożenie aplikacji do działania w sieci VPN w przyszłości. Umożliwi to dostęp do systemu z każdego miejsca na Ziemi, co pozwoli położnym na wygodne i bezpieczne zarządzanie dokumentacją pacjentów bez konieczności przenoszenia ze sobą wielu fizycznych dokumentów. Wymaga to jednak ingerencji specjalistów dot. Cyberbezpieczeństwa w sieciach komputerowych, czego niniejsza praca nie obejmuje.

Praca ta ma na celu szczegółowe omówienie procesu projektowania, implementacji oraz potencjalnych korzyści wynikających z wprowadzenia opisywanej aplikacji dla sektora medycznego. Zaprezentowane rozwiązanie nie tylko zwiększy efektywność pracy personelu medycznego, ale również przyczyni się do podniesienia standardów bezpieczeństwa i ochrony danych w placówkach medycznych.

## Analiza biznesowa – założenia projektowe aplikacji oraz problemy do rozwiązania

W rozdziale dot. Analizy biznesowej zamieściłem wszystkie założenia projektowe jakie zostały ustalone z przedstawicielami sektora medycznego przed rozpoczęciem pracy nad projektem. W sekcji tej można się dowiedzieć jakie problemy należało rozwiązać z punktu widzenia zarówno funkcjonalności, dostępności dla użytkownika oraz bezpieczeństwa przetwarzania i przechowywania danych medycznych.

### Funkcjonalność – w jakim zakresie projekt powinien wspierać użytkownika.

Aplikacja LucyCover została zaprojektowana aby sprostać specyficznym problemom położnych środowiskowych oraz doradców laktacyjnych na terenie Jeleniej Góry. Poniżej przedstawiono kluczowe

funkcjonalności, które zostały uwzględnione w projekcie:

1. Możliwość wprowadzania, przechowywanie oraz modyfikacji danych osobowych pacjentów oraz ich dokumentacji medycznej dot. Stanu zdrowia fizycznego oraz psychicznego zarówno matki jak i noworodka w pierwszych tygodniach życia po opuszczeniu szpitala.
2. Możliwość wprowadzania, przechowywanie oraz modyfikacji dokumentacji medycznej zawierającej informacje o stanach rozwoju noworodka w pierwszych tygodniach życia oraz informacje o stanie matki po porodzie.
3. Wprowadzenie intuicyjnego kanału szybkiej komunikacji między personelem przychodni położniczej a pacjentem. Z poziomu aplikacji położna powinna mieć możliwość wysłania wiadomości email do swojego pacjenta.
4. Aplikacja powinna umożliwić użytkownikowi efektywne planowanie swojego dnia pracy/wizyt. Dostępny powinien być terminarz w, którym będzie można planować wizyty na kolejne dni pracy.
5. Wyżej wspomniany terminarz powinien jednocześnie posiadać opcje, która umożliwi użytkownikowi szybkie wysłanie informacji do wybranego pacjenta na temat zaplanowania wizyty położnej w jego domu lub jej odwołania za pomocą poczty elektronicznej (email)
6. Projekt powinien rozwiązywać problem ręcznego wypisywania zaleceń medycznych dla pacjenta po wizycie. Po każdej wizycie położna będzie mogła w aplikacji szybko wypisać zalecenia dla swojej pacjentki, które zostaną automatycznie sformatowane przez systemu do czytelnej i profesjonalnej formy. Zalecenia te zostaną zapisane na koncie pacjenta do momentu ich ręcznego usunięcia przez użytkownika. Dodatkowo użytkownik ma, możliwość wydrukowania zaleceń aby przekazać je pacjentowi w wydrukowanej formie. Na tą chwilę jest to jedyna forma przekazania pacjentowi zalecań. (Aplikacja nie wysyła zaleceń drogą elektroniczną jednak pozostawia opcje zapisania jej jako plik PDF)

### Dostępność –Problem różnicy wieku, urządzeń mobilnych oraz niepełnosprawności

Projekt został zaprojektowany z myślą o intuicyjności oraz prostocie użytkowania. Aplikacja musi być prosta i dostępna dla osób starszych i mniej zaawansowanych technologicznie jednocześnie nie odstraszać młodszego pokolenia szaro-czarnym wyglądem. Z tego też powodu UI (user interface) [[1]](#footnote-1) zbudowany jest z dużych widocznych bloków oraz wyraźnej czcionki. W kluczowych miejscach aplikacji zastąpiono tekst ikonami lub zdjęciami aby kontent był czytelny oraz intuicyjna a zarazem estetyczny i elegancki. Jako aplikacja o tematyce położniczej dominującym kolorem jest róż i biel nawiązujący do odzieży medycznej noszonej przez położne.

LucyCover stworzone zostało również z myślą o dostępności dla osób niepełnosprawnych. Wielkość i kolor czcionki oraz tło na jakim się znajduje dobrane zostało również aby ułatwić pracę osobą mający problem ze wzrokiem. Wielkość ikon oraz formularzy do wprowadzania danych również została powiększona dla osób mających problemy z płynną obsługą myszki komputerowej.

Premiera iPhone’a w 2007 roku znacząco wpłynęła na rozwój aplikacji mobilnych[[2]](#footnote-2).Zdaniem firmy Research.com statystyki pokazują, że w dzisiejszych czasach ponad połowa osób korzysta z aplikacji webowych z poziomu urządzeń mobilnych takich jak smartfony, tablety. LucyCover wychodzi naprzeciw tym oczekiwania i zostało stworzona w taki sposób aby być dostępnym zarówno dla urządzeń mobilnych jak i desktopowych. Dzięki temu aplikacja po zainstalowaniu na serwerze i uruchomieniu w sieci lokalnej będzie mogła być używana bezpośrednio podczas wizyty w domu pacjenta za pomocą sieci VPN.

A graph of a bar chart

Description automatically generated with medium confidence[[3]](#footnote-3)

### Bezpieczeństwo – jak przetwarzać oraz jak je przechowywać dane w bezpieczny sposób.

Wraz z rozwojem sieci internetowej oraz korzystających z niej aplikacji webowych pojawił się problem związany z dostępem do zasobów. Aplikacje udostępnione w sieci narażone są na bezpośrednie ataki ze strony cyberprzestępców. Omawiana przez nas aplikacja jako że dotyczy sektora medycznego będzie miała styczność z danymi, które są poufne i dostęp do nich powinien być szczególnie chroniony. W związku z tym ustalone zostało, że aplikacja zostanie uruchomiona i dostępna jedynie w sieci lokalnej. W związku z koniecznością stałego połączenia aplikacji z siecią zewnętrzną do celów obsługi protokołów poczty elektronicznej koniecznym było staranne zaprojektowanie strategii zabezpieczenia i ochrony składowanych danych w bazie danych oraz ich przetwarzania. W związku z tym zaplanowano następujące instrukcje:

1. Komunikacja między warstwą prezentacji oraz warstwą logiki biznesowej powinna być nawiązana z wykorzystaniem protokołu HTTPS[[4]](#footnote-4) (szyfrowanie danych podczas transmisji)
2. Przetwarzane dane powinny być odpowiednio walidowane oraz szyfrowane przed zapisem do bazy danych
3. Baza danych powinna być oddzielną zabezpieczoną strukturą.
4. Wszystkie dane w bazie danych powinny być zaszyfrowane i niezrozumiałe dla człowieka oraz sztucznej inteligencji
5. Dostęp do wszystkich interfejsów API powinien być poprzedzony autentykacją oraz autoryzacją.
6. Aplikacja powinna wymagać od użytkownika planowania oraz wykonywania regularnych kopi zapasowych (backup) z wykorzystaniem reguły (3-2-1)[[5]](#footnote-5)
7. Warstwa aplikacji powinna zapewniać pierwszą linię obrony poprzez walidacje i sprawdzenie wprowadzanych danych przez użytkownika zanim zostaną one wysłane na poszczególne interfejsy końcowe warstwy logiki biznesowej.

### Podsumowanie

W ramach analizy biznesowej aplikacji omówiono założenia projektowe, uwzględniając potrzeby funkcjonalne, dostępność oraz bezpieczeństwo danych medycznych. Aplikacja ma wspierać położne i doradców laktacyjnych, oferując m.in. możliwość zarządzania danymi pacjentów, intuicyjną komunikację oraz efektywne planowanie wizyt. Projekt zakłada prosty i dostępny interfejs dla użytkowników w różnym wieku i o różnym stopniu zaawansowania technologicznego, a także zapewnienie bezpieczeństwa poprzez odpowiednie szyfrowanie, walidację danych oraz dbałość o reguły bezpiecznego tworzenia aplikacji webowych. Projekt będzie działać w sieci lokalnej, co ma minimalizować ryzyko ataków cybernetycznych. Jednak nie wyklucza oraz nie zamyka furtki na udostępnienie projektu w sieci z wykorzystaniem sieci VPN.

# Omówienie wykorzystanych technologii

Rozdział ten szczegółowo tłumaczy jakie problemy rozwiązuje projekt oraz jakie wytyczne zostały ustalone podczas analizy biznesowej obsługiwanego sektora.

Rozdział ten szczegółowo opisuje jakie technologie zostały wykorzystane w projekcie i w jaki sposób zrealizowana została komunikacja między nimi. Wytłumaczone zostaje również dlaczego zostały podjęte decyzje o wykorzystaniu konkretnych technologii oraz w jaki sposób zostały zaimplementowane.

### Ogólny opis wykorzystanych technologii w projekcie.

Projekt LucyCover wykorzystuje nowoczesne technologie, które zapewniają wysoką wydajność, skalowalność oraz bezpieczeństwo. Aby w pełni odizolować obszary odpowiedzialności projekt został podzielony na 3 oddzielne aplikacje tworząc tak zwaną **architekturę trójwarstwową[[6]](#footnote-6).** Każda z tych warstw pełni specyficzne funkcje i odpowiada za inne aspekty działania aplikacji.

Pierwsza warstwa **FROTNEND (warstwa prezentacji)**odpowiedzialna jest za interakcje użytkownika z aplikacją. Została ona utworzona przy wykorzystaniu języka programowania JavaScript, który swoja szybkością i wysoką dostępnością rozwiązań webowych idealnie sprawdził się w tej roli. Niestety tworzenie zaawansowanych aplikacji frontendowych przy użyciu jedynie tego języka programowania byłoby bardzo czasochłonne i nieefektywne. Naprzeciw tym problemom wychodzi biblioteka REACT w wersji 18. To w tej technologii i jej towarzyszących została utworzona cała wizualna warstwa projektu. Drugą warstwą tego rozwiązania jest **Backend (warstwa logiki biznesowej).** Jako, że projekt wspiera sektor medyczny istnieje wiele krytycznych aspektów które muszą zostać pokryte aby zachować skalowalność, wydajność a przede wszystkim bezpieczeństwo. To właśnie backend odpowiedzialny jest za wykonywanie wszystkich założeń logiki biznesowej aplikacji taki jak przyjmowanie danych, walidacje, autoryzacje oraz autentykacje, odpowiednie przygotowanie danych do zapisu w bazie danych oraz ich pobrania. Do obsługi tego ważnego zadania wykorzystałem język programowania utworzony przez firmę Microsoft – C#. Język ten wspiera wiele bardzo zaawansowanych technologii do obsługi aplikacji po stronie serwera. Dodatkowym powodem wykorzystania tej technologii jest świetna współpraca tego języka z serwisami dostępnymi na chmurze obliczeniowej Azure (LucyCover pozostawia możliwość uruchomienia aplikacji w sieci). W języku C# stworzona została technologia ASP .NET CORE, która wychodzi naprzeciw problemom stawianym przez nasz projekt. To właśnie ta technologia została wykorzystana do przygotowania warstwy logiki biznesowej mojej aplikacji. Ostatnią warstwą naszego projektu jest **warstwa danych (czyt. Baza danych)**. MS SQL odpowiedzialny jest za przechowywanie i zarządzanie danymi aplikacji. To tutaj dane są zapisywane, odczytywane, aktualizowane i usuwane.

W architekturze trójwarstwowej każda warstwa jest od siebie niezależna, co pozwala na większą elastyczność, skalowalność oraz łatwiejsze zarządzanie i rozwój aplikacji.

Każda warstwa projektu korzysta ze swojej głównej technologii oraz wielu bibliotek wspomagających oraz usprawniających ich pracę. Aby LucyCover mogła wydajnie sprostać wszystkim powierzonym jej zadaniom wykorzystałem do jej utworzenia następujące technologie:

* C#[[7]](#footnote-7)
* .NET[[8]](#footnote-8)
* Entity Framework[[9]](#footnote-9)
* ASP .NET CORE – REST API [[10]](#footnote-10)
* AutoMapper[[11]](#footnote-11)
* FluentValidator[[12]](#footnote-12)
* Nlog[[13]](#footnote-13)
* JwtBearer[[14]](#footnote-14)
* Swashbuckle/swawgger[[15]](#footnote-15)
* xUnit[[16]](#footnote-16)
* FluentAssertions[[17]](#footnote-17)
* HTML 5 / CSS3[[18]](#footnote-18)
* JavaScript[[19]](#footnote-19)
* React Redux[[20]](#footnote-20)
* React Router[[21]](#footnote-21)
* React @Tanstack Query[[22]](#footnote-22)
* Node[[23]](#footnote-23)

### .NET – Platforma programistyczna od Microsoft

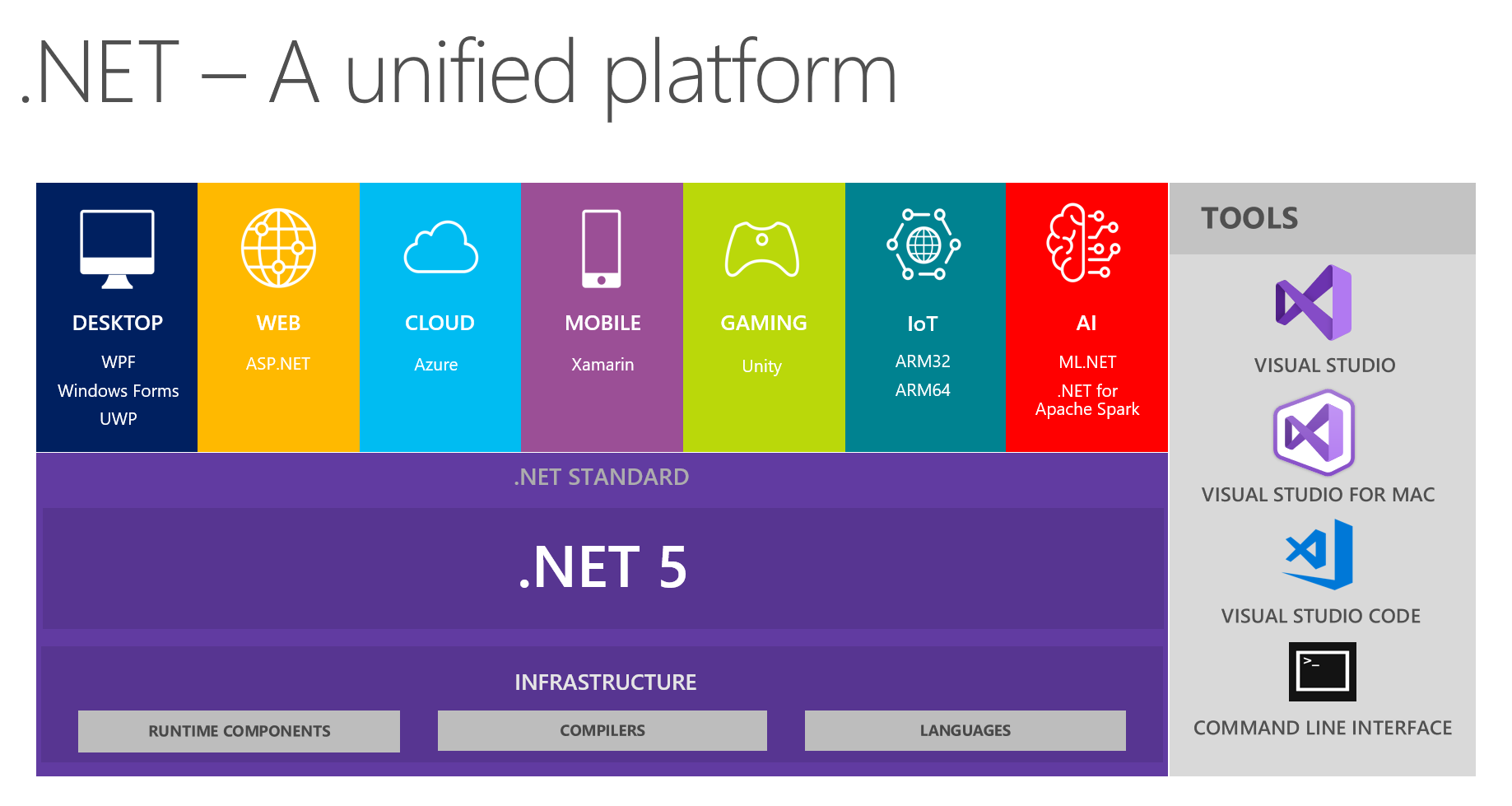
.NET jest to wszechstronna platforma programistyczna stworzona przez firmę Microsoft. Jest częstym wyborem podczas projektowania oraz tworzenia rozwiązań takich jak: aplikacje webowe, aplikacje desktopowe , mobilne oraz gry wideo. Platforma Net oferuje wysoce rozbudowane środowisko uruchomieniowe, wiele różnorodnych bibliotek, które ułatwiają programowanie oraz rozwój oprogramowania. Środowisko .Net pozwala na tworzenie aplikacji w językach C#, F# oraz Visual Basic .NET.

Na uwagę zasługuje fakt, że .NET jest platformą darmową typu opensource. Aktualnie dot NET pozwala tworzyć oprogramowanie na różne systemy operacyjne (Linux, Windows oraz McOS) co czyni go technologią wieloplatformową wspierająca róże architektury systemów. Co ciekawe .NET umożliwia publikacje aplikacji wraz z całym środowiskiem uruchomieniowym dzięki czemu możemy uruchomić aplikację na urządzeniach takich jak telefony tablety a nawet konsole XBOX bez konieczności instalowania, żadnego dodatkowego oprogramowania.

Dzięki ogromnej społeczności platformy .NET mamy dostęp do ciągle to nowszych oraz bezpieczniejszych narzędzi oraz bibliotek. Starsze rozwiązania również są regularnie aktualizowane co przyczynia się do utrzymania wysokiej jakości bezpieczeństwa oraz stabilności. Piecze nad stabilnością oraz bezpieczeństwem rdzenia całego systemu sprawuje firma Microsoft, która zapewnia stabilny rozwój tej technologii.

Warto zaznaczyć, że platforma Microsoftu umożliwia nam korzystanie z bibliotek napisanych w różnych językach wchodzących w skład .NET. Działa to ponieważ wszystkie języki z tej rodziny tłumaczone są na język wspólny tak zwany IL (Intermediate Language)[[24]](#footnote-24).

Technologię tą zastosowałem ze względu na jej wysoka wydajność, kompatybilność oraz mnogość dostępnych bibliotek.



(https://auth0.com/blog/what-is-dotnet-platform-overview/)

### 2.3 C# - Główny język programowania wykorzystywany w technologii .NET

KSIĄŻKA MARCIN LIS

### 2.4 EntityFramework – zaawansowane rozwiązanie ORM dla programistów .NET

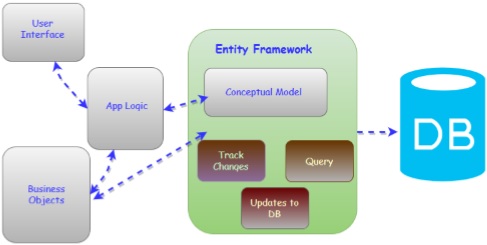
Entity Framework (EF) to jeden z najpopularniejszych frameworków powiązany z wyżej opisaną technologią .NET. Rozwiązanie to wykorzystywane jest do mapowania obiektowo realcyjnego (Object-Relational Mapping). [[25]](#footnote-25)Umożliwia programistą łatwe tworzenie i zarządzanie danymi gromadzonymi w aplikacji. Entiy Framework umożliwia mapowanie obiektów z poziomu kodu na relacyjne bazy danych. Jego głównymi zadaniami jest pokrycie podstawowych operacji CRUD (Create Read, Update, Delete)[[26]](#footnote-26) czyli tworzenie, aktualizowanie, odczyt oraz usuwanie danych z tabeli.

Bibiloteka ta pozwala na pracę na poziomie obiektów w kodzie aplikacji, które następnie są tłumaczone przez Entity Framework na odpowiednie zapytania SQL, które będą wykonane w bazie danych.

Rozwiązanie to proponuje nam dwa modele tworzenie bazydanych:

1. „Code First” – model polega na utworzeniu całej struktury bazy danych oraz połączeń i relacji już podczas pisania kodu aplikacji. Następnie Entity Framework przetłumaczy te instrukcje na odpowiednie zapytania SQL za pomocą tzw. „Migracji[[27]](#footnote-27)”
2. „Database First” – model polega na utworzeniu struktury bazy danych na podstawie, której Entity Framework wygeneruje modele w kodzie aplikacji.

W 2016 firma Microsoft wprowadziła Entity Framework Core (EF Core)[[28]](#footnote-28), który jest wydajniejszym i jeszcze bardziej zoptymalizowanym rozwiązaniem.



(https://www.tutorialspoint.com/entity\_framework/entity\_framework\_overview.htm)

### 2.5 ASP .NET CORE – Platforma przeznaczona do budowy aplikacji internetowych

ASP .NET CORE jest to framework będący częścią rodziny .NET przeznaczony do dynamicznej budowy wydajnych aplikacji internetowych. Opracowane przez firmę Microsoft rozwiązanie pozwala tworzyć aplikację w oparciu o architektury:

1. ASP .NET WEB FORMS[[29]](#footnote-29)
2. ASP.NET MVC (MODEL-VIEW-CONTROLLER)[[30]](#footnote-30)
3. ASP.NET WEB API[[31]](#footnote-31)
4. ASP.NET MINIMAL API[[32]](#footnote-32)
5. ASP.NET RAZOR PAGES[[33]](#footnote-33)

ASP.NET CORE stworzone zostało z myślą o współpracy z protokołami HTTP oraz HTTP’S. Cechuje się bardzo dobrą wydajnością optymalizacją oraz wstępnie skonfigurowanym rdzeniem pod względami bezpieczeństwa. Rozwiązanie to jest następcą poprzedniej technologii ASP.NET, która wykonywana mogła być jedynie na urządzeniach z systemami Windows. Aktualnie rozbudowanie tej technologii do wersji ASP.NET CORE umożliwiło uruchamianie systemu na wszystkich urządzeniach nie zależnie od wersji systemu (Linux,Windows,MCOS)

### 2.6 AutoMapper – Mapowanie obiektów między wersjami aplikacji

AutoMapper to popularna biblioteka przygotowana na potrzeby platformy .NET, wspomagająca programistów podczas mapowania obiektów między różnymi warstwami aplikacji. Pozwala przekształcać obiekty jednej klasy bazowej na instancje klasy docelowej.

Przykładem wykorzystania tego rozwiązania jest na przykład mapowanie otrzymywanych obiektów DTO (Data Transfer Object) [[34]](#footnote-34) na encje wykorzystywane w bazie danych ( i odwrotnie ).

Automapper skutecznie ułatwia szybką i elegancką zamianę obiektów zawierających podobne typy danych.

### 2.7. FluentValidator – walidacja danych .NET

FluentValidator jest to kolejny framework z grupy .NET, który umożliwia deweloperom tworzenie zaawansowanej logiki walidowania danych w płynny czytelny i łatwy w utrzymaniu sposób. Jego głównym oraz podstawowym zadaniem jest walidowanie obiektów i ich właściwości w odniesieniu do konkretnych reguł.

Walidację obiektów umieszczamy w specjalnej dedykowanej klasie walidatora dzięki czemu w prosty sposób możemy trzymać się reguł spisanych w podejściu SOLID[[35]](#footnote-35).

Dodatkowo Fluent Validator wspiera również walidacje warunkową. Dzięki czemu możemy tworzyć reguły, które będą obowiązywały dopiero po spełnieniu jakiegoś warunku.

### 2.8. NLog – narzedzie do logowania danych w aplikacji

Prowadzenie logów w aplikacji jest kluczową funkcjonalnością, która znacząco wpływa na utrzymanie i rozwój oprogramowania.Nlog jest popularnym narzędziem powstałym do tworzenia danych z logami powstałymi podczas korzystania z aplikacji napisanej na platformie .NET. Framework umożliwia rejestrowanie wszystkich zdarzeń, błędów lub innych istotnych informacji diagnostycznych w aplikacjach. Jest to narzędzie względnie proste w konfiguracji dzięki czemu jesteśmy w stanie szybko uruchomi go w naszym programie.

### 2.9. JWT BEARER – narzędzie do uwierzytelniania użytkownika

JWT Bearer Token inaczej „token na okaziciela” jest specjalnym tokenem wystawianym przez serwer uwierzytelniający w momencie, gdy użytkownik podaje swoje prawidłowe dane autoryzujące. Z technicznego punktu widzenia jest to zwykły ciąg znaków przechowujący wrażliwe dane użytkownika, który został zakodowany w Base64url. Ze względu na to, że dane, które się w nim zawierają nie są zaszyfrowane to komunikacja pomiędzy klientem a serwerem odbywać powinna się wyłącznie przez protokół HTTPS.

Token musi być odpowiednio chroniony ponieważ osoba, która zdobyła/wykradła taki token mogłaby za pomocą niego wykonywać operację na endpointach API (ponieważ zostałaby pomyślnie zautoryzowana)

Przewagą JWT Bearer Token nad na przykład autoryzacją za pomocą plikow cookies[[36]](#footnote-36) jest dodatkowa ochrona w postaci ,,czasu życia” takiego tokana. Podczas konfiguracji możemy ustalić jak długo system ma akceptować otrzymywany token a kiedy ma zacząć go ignorować i wygenerować nowy.

Nad spójnością tokenu i uniemożliwieniem ingerencji w dane, które się w nim znajdują piecze sprawuje system sumy kontrolnej. Suma kontrolna jest generowana podczas generowania tokena, kiedy serwer koduje w nim dane oraz zabezpiecza je secretem bezpośrednio zdefiniowanym w systemie. Innymi słowy potencjalna próba ingerencji w zawartość tokena musiałaby zostać potwierdzona hasłem kodującym, który jest zapisany i chroniony na serwerze.

Na obrazku na następnej stronie przedstawiono schemat działania takiej komunikacji z wykorzystaniem JWT BEARER.

A screenshot of a computer

Description automatically generated

(https://netstorage.ringcentral.com/dpw/guide/images/oauth-password-flow.png?v=2024-06-14)

### 2.10. Swashbuckle/swawgger – narzędzie do automatycznego tworzenia dokumentacji API

Tworzenie dokumentacji technicznej jest bardzo czasochłonnym i żmudnym zajęciem. Tworząc aplikacje webowe w oparciu o architekturę REST API musimy już na etapie projektowania dokumentować w jaki sposób korzystać można z naszych endpointów[[37]](#footnote-37). Sensowna dokumentacja powinna zawierać informację jakiego typu dane przyjmuje dany punkt końcowy, jakie czasowniki protokołu HTTP muszą zostać zastosowane oraz jakiego rezultatu powinniśmy się spodziewać w przypadku otrzymania zarówno prawidłowych jak i nieprawidłowych danych.

Z pomocą przychodzi Swagger wspierany przez firmę SMARTBEAR. Jest to rozwiązanie wspierające deweloperów w całkowicie lub częściowo automatyzowaniu procesu tworzenia dokumentacji technicznej projektu. Jako rozwiązanie typu open-source mamy pewność, że swagger jest ciągle wspierany oraz rozwijanym projektem. Działanie tego rozwiązania jest bardzo proste. Po rejestracji swaggera w assembly [[38]](#footnote-38)naszego projektu biblioteka automatycznie będzie śledzić nowo tworzone endpointy oraz wszystkie właściwości jakie przyjmują. Następnie tworzona jest automatycznie dokumentacja w formie strony internetowej na której możemy przetestować działanie takiego endpointu oraz ustalić w jaki sposób możemy z niego skorzystać.

Taka dokumentacja jest szczególnie przydatna w momencie kiedy poszczególne warstwy aplikacji są tworzone przez dwie różne grupy deweloperskie. Osoby odpowiedzialne za warstwę aplikacji nie zawsze są wystarczająco wtajemniczone w proces rozwoju warstwy biznesowej projektu i musza pozyskiwać informację w jaki sposób odwołać swoje widoki do odpowiednich punktów końcowych projektu. Dzięki swaggerowi możemy w prosty i przede wszystkim spójny / jednolity sposób przedstawiać zmiany wprowadzane na naszych endpointach.

### 2.11. HTML 5 / CSS3 – Fundamenty warstwy prezentacji każdej aplikacji webowej.

Zarówno HTML jak i CSS są strukturalnymi językami przeznaczonym do tworzenia współczesnych struktur stron aplikacji internetowych. Wersja 5 HTML oraz wersja CSS 3 jest najnowszym rozwinięciem tych języków. Podstawową zaletą HTML 5 jest możliwość obsługiwania grafiki oraz wideo bez konieczności instalowania dodatkowych wyczek.

Głównym zadaniem stawianym przed HTML jest definiowanie struktur aplikacji webowej. Jest to język oparty na znacznikach, które umożliwiają zapisanie szablonu wizualnej części aplikacji za pomocą znaczników.

CSS jest dodatkowym odrębnym narzędziem jednak ściśle powiązanym z HTML. Wykorzystujemy go do definiowania styli dla utworzonych obiektów. CSS może pracować wewnątrz plików HTML jednak z reguły jest on tworzony jako zewnętrzny plik oraz połączony relacją z plikiem HTML.

Dzięki tym dwóm technologią, możemy w czysty i czytelny sposób rozdzielić strukturę aplikacji od zdefiniowanych dla niej styli.

### 2.12. REACT – podstawowe narzędzie podczas tworzenia nowoczesnych interfejsów graficznych.

React jest otwarto źródłową biblioteką stworzoną przez Facebooka. Pozwala ona budowanie dużych aplikacji internetowych z wykorzystaniem komponentów. Technologia ta znacznie usprawnia pisanie kodu ponieważ wykorzystuje paradygmaty programowania deklaratywnego. To znaczy, że nasz kod definiuje co powinno zostać zrobione a nie jak powinno zostać zrobione. Dzięki temu jesteśmy w stanie zaoszczędzić mnóstwo czasu podczas pracy zachowując przy tym większą czytelność kodu oraz znacząco usprawniając wydajność aplikacji.

Jedną z głównych zalet Reacta jest komponentowa architektura. Dzięki temu korzystając z komponentów możemy tworzyć izolowane mniejsze fragmenty kodu, które są czytelne, skalowalne i możliwe do wykorzystania w wielu fragmentach programu.

Kolejna zaletą Reacta jest jego popularność a co się z tym wiąże szeroka społeczność wykorzystujących go deweloperów. Dzięki temu mamy dostęp do ogromnej ilości różnego rodzaju rozszerzeń oraz bibliotek wspomagających. Pozwalają one usprawnić naszą pracę dostarczając nam proste interfejsy z, których możemy korzystać. Zaoszczędzamy dzięki temu czas oraz przejrzystość kodu.

Niestety React nie jest biblioteką kompatybilną wstecznie przez co w momencie wydawania dużych aktualizacji musimy dostosowywać aplikację do nowych wymogów aktualnej wersji. Kolejnym problemem jest fakt, że aplikacje utworzone na tym frameworku ładowane sa dynamicznie. To znaczy, że nie otrzymujemy gotowego kodu HTML / CSS3 po uruchomieniu tylko cała treść zostaje wyświetlana i dostosowywana w trakcie działania aplikacji. To powoduje problem z dostosowaniem strony internetowej do warunków SEO. W przypadku projektowania aplikacji LucyCover niedogodność ta nie miała znaczenia ponieważ aplikacja nie będzie udostępniana dla społeczności internetowej tylko klientów indywidualnych.

W ramach podsumowania, uznałem Reacta jako świetne rozwiązanie dla problemów stawianych przed tematem tej pracy inżynierskiej ponieważ, pozwala on w szybki sposób tworzyć wydajne, stabilne i skalowalne aplikacje. Problemy SEO jakie pojawiają się podczas korzystania nie wpłyną negatywnie na korzystanie czy odbiór LucyCover.

### 2.13. VITE – świeży powiew przy tworzeniu aplikacji front-end

Vite jest nowoczesnym rozwiązaniem przeznaczonym do budowania aplikacji front-end. Narzędzie to stworzone zostało przez twórców takich języków programowania jak np. Vue.js korzysta on z natywnego serwera modułów ES dzięki czemu jest jednym z najszybszych dostępnych aktualnie narzędzi.

Struktura Vite jest bardzo prosta, deweloper podczas instalacji dokonuje wyboru z jakich technologii chcę korzystać podczas tworzenia nowego projektu następnie vite przygotowuje katalog projektu ze stałą dla siebie strukturą. Vite bazuje na dwóch plikach main.js który służy do konfiguracji aplikacji i zainicjowaniu plików wybranej technologii (czyli na przykład korzystając z React importujemy plik App.js który jest podstawową strukturą reacta) oraz na pliku index.js, który natomiast definiuje podstawową strukturę HTML jako baza. Następnie za pomocą wybranych technologii front-end będzie ona rozbudowywana dynamicznie.

Vite dostarcza wiele pomocniczych funkcjonalności. Jedna z nich jest możliwość zbudowania aplikacji po zakończeniu prac deweloperskich. W momencie kiedy prace nad aplikacją dobiegną końca, i aplikacje jest gotowa do wdrożenia na serwery produkcyjne vite przekonwertuje kod (np. reacta) na czysty kod JavaScript, który będzie zrozumiały dla przeglądarki. Dzięki temu produkt zostanie odpowiednio przygotowany, skompresowany oraz gotowy do wdrożenia[[39]](#footnote-39). Vite umożliwia dodatkowo uruchomienie aplikacji w sieci lokalnej[[40]](#footnote-40), dzięki czemu możemy uruchomić aplikacje bezpośrednio na naszym komputerze a dostęp do niej zdalny zapewnimy dzięki protokołowi VPN[[41]](#footnote-41).

### 2.14. React Redux – biblioteka do zarządzania stanem aplikacji

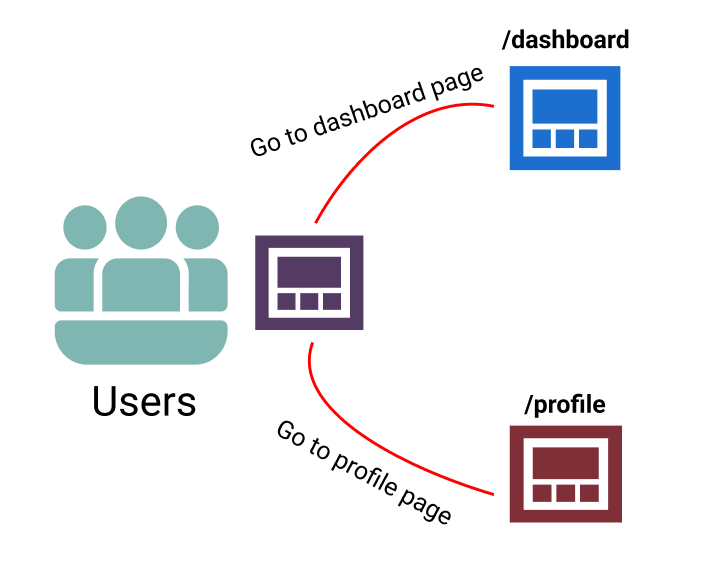
REDUX jest biblioteką przeznaczoną do zarządzania stanem z poziomu całej aplikacji. Framework ten jest swoistym rozwinięciem wzorca architektury aplikacji – flux.[[42]](#footnote-42) Biblioteka została napisana w JavaScript i jest używana przez reacta. W szczególności ułatwia zarządzanie i zcentralizowanie stanu aplikacji podczas tworzenia oraz utrzymania zaawansowanych interfejsów użytkownika. Jest szczególnie pomocna w sytuacji, gdy tworzymy aplikacje z zachowaniem dobrych paradygmatów programowania. Redux pozwala nam odizolować miejsce przechowywania danych od głównej logiki komponentu. Dzięki temu poszczególne segmenty aplikacji stają się bardziej atomowe. Centralizacja stanu aplikacji dodatkowo ułatwia zarzadzanie oraz debugowanie. Kolejnym atutem przemawiającym na korzyść tej technologii jest zachowanie spójności w działaniu w różnych środowiskach takich jak serwer i przeglądarka internetowa. Jako, że Redux jest scentralizowanym stanem aplikacji ułatwia programiście testowanie projektu.

### 2.15. React Router – technologia tworzenia routingu w aplikacjach react

React Router jest biblioteką napisaną w języku JavaScript w 2013 roku. Przeznaczona została do wspierania deweloperów wykorzystujących reacta do tworzenia skomplikowanych aplikacji client-side[[43]](#footnote-43)

Podstawowym celem tej technologii jest zapewnienie wyszukiwania za pomocą asynchronicznych adresów URL[[44]](#footnote-44). Podstawowym zastosowanie React Router jest projektowanie i tworzenie aplikacji w architekturze single-page [[45]](#footnote-45) z zachowaniem standardowej struktury i funkcjonalności. Zasada jego działania jest stosunkowo prosta. Użytkownik wprowadzają adres URL w przeglądarce internetowej, Router dopasowuje jego zapytanie URL do konkretnej „ścieżki” (en. Pathways[[46]](#footnote-46)), która odnosi się do konkretnego komponentu. React Router w prosty sposób pozwala programiście zdefiniować szkielet / wzór dostępnych odnośników URL oraz jakie komponenty mają zostać wyświetlone po ich dopasowaniu.

Dodatkowym atutem technologii React Router jest proste zarządzanie stanem aplikacji. Na przykład w prosty sposób, możemy przenosić dane pomiędzy backendem a aplikacją kliencką z wykorzystaniem „Loader[[47]](#footnote-47)” oraz „actions[[48]](#footnote-48)” dostarczonych przez tą technologie.



(https://medium.com/nerd-for-tech/what-is-the-difference-between-react-router-and-conventional-routing-9b11159d92a4)

### 2.16. Tanstack Query – biblioteka wspierająca komunikację serwer-client

Tanstack Query jest zaawansowaną biblioteką służącą do pobierania danych z serwera, synchronizacji danych, aktualizacji w tle oraz „cachowania[[49]](#footnote-49)” danych.

Do jego podstawowych zadań należy:

1. Przesyłania danych pomiędzy klientem a serwerem z wykorzystaniem zcentralizowanej konfiguracji. Dzięki Tanstack query wszystkie zapytania do interfejsów API zarówno modyfikujące (POST/PUT/DELETE) jak i pobierające dane (GET) przechowywane są w jedynym pliku jako funkcje fetch. Dzięki temu możemy korzystać z tych samych zapytań w wielu miejscach aplikacji i nie musimy martwić się o zachowanie spójności w różnych miejscach projektu. Odpowiednio skonfigurowana biblioteka pozwala na pobieranie danych i automatyczne zapisywanie ich w pamięci podręcznej przeglądarki. Dzięki temu znacznie zyskujemy na wydajności względem korzystania ze standardowych mechanizmów języka JavaScript.
2. Tanstack Query automatycznie będzie kontrolował spójność plików przechowywanych na serwerze i w pamięci podręcznej aplikacji klienckiej. Technologia ta umożliwia nam dostosowanie częstotliwości aktualizowania danych gromadzonych w pamięci podręcznej w określonych przez nas ramach czasowych. Dodatkowo aplikacja pozwala nam aktualizować wyświetlane dane na przykład po wykonaniu operacji dodawania nowych danych bez konieczności przeładowania strony jak to ma miejsce w standardowych aplikacjach webowych nie korzystających z tej technologii.

Dodatkowym atutem TanstackQuery jest udostępnienie specjalnych zmiennych przechowujących aktualny stan komunikacji pomiędzy serwerem a klientem. Na przykład „hook” [[50]](#footnote-50) useQuery zwraca obiekt zawierający pola z informacją na jakim etapie jest komunikacja (isPending,isSuccess,isError) dzięki temu możemy w prosty sposób przygotować nasze komponenty do wyświetlania odpowiedni informacji na konkretnych etapach komunikacji.

### 2.17 FramerMotion – biblioteka do tworzenia animacji

FramerMotion to mała biblioteka stworzona do rozbudowy aplikacji opartych o technologie React służąca do tworzenia wydajnych oraz płynnych animacji bez konieczności pisania skomplikowanego kodu. Jest to rozwiązanie typu open-source zapoczątkowane przez studio Framer. Dzięki temu prostemu mechanizmowi, możemy zaoszczędzić czas na zaawansowaną rozbudowę arkuszy stylów. Kolejnym atutem tego rozwiązania jest zwiększenie czytelności kodu oraz jego optymalizacji. FramerMotion, może zostać wykorzystywane do tworzenia prostych animacji ale również umożliwia on tworzenie całych nowoczesnych interfejsów użytkownika. Umożliwia animacje wszystkich elementów struktury HTML zapewniając ich własną implementację z dodatkowymi możliwościami konfiguracji. Animacje gestów czy płynne przejścia pomiędzy ekranami nie są żadnym wyzwaniem. Technologia ta umożliwia wyświetlenie interfejsu w sposób bardziej płynny i naturalny.

### 2.18. MailKit – biblioteka do obsługi protokołów poczty elektronicznej

MailKit jest to kolejna open-sourcowa biblioteka przeznaczona do obsługi protokołów poczty elektronicznej w aplikacjach tworzonych na platformie .NET takich jak:

* IMAP[[51]](#footnote-51)
* POP3[[52]](#footnote-52)
* SMTP[[53]](#footnote-53)

MailKit zapewnia implementacje tych protokołów z zachowaniem wymogów RFC[[54]](#footnote-54). Korzystając z tej technologii możemy nawiązywać połączenia z zewnętrznymi serwerami pocztowymi oraz prowadzić szyfrowaną komunikację. Jest to rozwiązanie darmowe świetnie sprawdzające się w projekcie LucyCover.

### 2.19. XUnit – testowanie aplikacji z rodziny .NET

Nieodłącznym elementem sukcesu każdego poważnego projektu są odpowiednio przygotowane testy. Wraz z rozwojem projektu i stopniowo pojawiającymi się kolejnymi funkcjonalnościami nie sposób jest zapanować nad utrzymaniem odpowiednich rezultatów zwracanych przez już wytworzony kod. W takiej sytuacji obowiązkowe okazuje się regularne tworzenie testów dla już istniejących algorytmów i implementacji. Programiści od lat zmagają się z wyzwaniem zwiększania wydajności testowania. Jednym z rozwiązań pozwalającym osiągnąć ten cel jest Xunit. Jest to technologia stworzona dla aplikacji z rodziny Microsoftu (.NET). Rozwiązanie to pozwala na tworzenie wszystkich typów testów:

1. Testy jednostkowe[[55]](#footnote-55)
2. Testy integracyjne[[56]](#footnote-56)

XUnit jest wysokowydajnym rozwiązaniem pozwalającym na pisanie nowoczesnych, elastycznych i skalowalnych testów. Które są strukturyzowane i organizowane w odpowiedni sposób.

Wykorzystując funkcje testów parametrycznych możemy przeprowadzić test różnych zestawów danych za pomocą jednego testu. Xunit udostępnia deweloperom atrybuty takie jak Fact i Theory który umożliwia precyzyjne określenie jakie warunki są wymagane aby test został uruchomiony. Kolejnym atutem XUnit jest możliwość skonfigurowania go w taki sposób aby mógł być uruchamiany jego jeden z elementów pipelinów [[57]](#footnote-57)przy korzystaniu z rozwiązań Continious integration [[58]](#footnote-58)oraz continious delivery[[59]](#footnote-59).

Podsumowując korzystanie z rozwiązania Microsoft jest nieocenione podczas tworzenia nowoczesnych aplikacji na platformie .NET ponieważ pozwala ona szybko i efektywnie zagwarantować sobie niezawodność aplikacji na wszystkich etapach jej tworzenia. Dzięki intuicyjnemu sposobowi pracy podczas zarówno kompilacji jak i wysłaniu ( en. deploy[[60]](#footnote-60)) go na serwer produkcyjny czy repozytorium zdalne.

### 2.20. Git / GitHub – System kontroli wersji

Tworzenie nowoczesnych rozwiązań IT stawia przed deweloperami szereg problemów do rozwiązania. Jednym z nich jest ustalenie w jaki sposób skutecznie i bezpiecznie kontrolować i łączyć efekty prac całego zespołu deweloperów. Każdy deweloper pracujący nad swoim fragmentem projektu tak naprawdę pracuje na tych samych danych co reszta zespołu. Prowadzi to często do komplikacji kiedy dwóch deweloperów pracujących nad tym samych projektem nie posiadają spójnych wersji kodu. Co gorsza mogą pojawiać się problemy kiedy w jednym czasie w ramach jednej iteracji dwóch deweloperów edytuje ten sam fragment oprogramowania. Co w takiej sytuacji ? Która wersja powinna zostać przyjęta jako ta prawidłowa ?

W rozwiązaniu tych zdarzeń przychodzi narzędzie stworzone przez twórcę Linuxa [[61]](#footnote-61)– Linus Torvalds[[62]](#footnote-62)

Git jest projektem stworzonym na jądrze LINUX i wykonywanym na unixowych konsolach. Dzięki niemu deweloperzy mogą na bieżąco mogą zapisywać i dokumentować efekty swojej pracy tworząc tzw. „Commity”. Potocznie mówiąc możemy nazwać je punktami odniesień w danym projekcie. Przyrównując git do systemów plików [[63]](#footnote-63)w systemach operacyjnych możemy nazwać commit „migawką[[64]](#footnote-64)” (eng. Snapsot). Commit jest konkretną wersją projektu do którą możemy na przykład traktować jako kopie zapasową albo punkt kontrolny. Kolejną nieoceniona funkcjonalnością gita jest automatyczne wykrywanie konfliktów powstałych w momencie próby jednoczesnej zmiany fragmentu kodu przed kilku deweloperów. W takiej sytuacji wspólnie można ustalić, która wersja kodu jest bardziej optymalna. Git jest rozwiązaniem, które jest zainstalowane na komputerze każdego dewelopera i pozwala na tworzenie lokalnych repozytoriów. Aby każdy deweloper mógł korzystać z aktualnej wersji projektu i wszystkie efekty prac deweloperów przechowywać w zcentralizowany i bezpieczny sposób powstał Git HUB. Jest to aplikacja webowa świadcząca wiele usług, które nie sposób wszystkie wypisać. Jednak jej głównymi funkcjonalnościami, które zostały wykorzystane w tym projekcie są:

1. Przechowywanie kopi projektu oraz pełnej listy historii zmian aplikacji w zcentralizowany sposób z możliwością bezpiecznego dostępu zdalnego
2. Zaimplementowanie do repozytorium akcji continious integration, która na bieżąco monitoruje stan i projektu gwarantując spójność i niezawodność aplikacji.
3. Możliwość współpracy nad projektem przez cały zespół deweloperski. Każdy programista może utworzyć kopie repozytorium na swoim lokalnym urządzeniu wprowadzać na niej zmiany a następnie implementować je do głównego repozytorium po przejściu wszystkich wymaganych testów.

Podsumowując system kontroli wersji jest niezbędnym narzędziem podczas tworzenia aplikacji na współczesny rynek IT. Git w połączeniu z GITHUB są najpopularniejszym rozwiązaniem w tej kategorii i świadczone przez nie usługi świetnie sprawdzają się podczas rozwoju tworzonej aplkikacji.

# 3. Implementacja aplikacji – budowa aplikacji w architekturze trójwarstwowej

Aplikacja LucyCover została zaprojektowana w architekturze trójwarstwowej. Architektura ta pozwala na zachowanie większej czytelności kodu, zapewnia dużo lepszą skalowalność aplikacji i pozwala zmniejszyć ryzyko występowania błędów. Podział polega na rozdzieleniu całej aplikacji na warstwę biznesową, warstwę prezentacji oraz warstwę dostępu do danych. Każda warstwa wykonuje inne zadanie i często jest odrębną aplikacją która komunikuje się za pomocą specjalnie przygotowanych interfejsów.

## 3.1 Implementacja warstwy prezentacji – aplikacja kliencka

Warstwa aplikacji jest jednym z kluczowych elementów architektury warstwowej. Możemy nazwać ją swego rodzaju interfejsem pomiędzy użytkownikiem a algorytmami przetwarzającymi dane. Warstwa ta jest odpowiedzialna za odpowiednie wyświetlenie przetworzonych danych, zapewnienie efektów wizualnych, czy umożliwienie użytkownikowi intuicyjną i bezproblemową analizę zwracanych danych. Warstwa prezentacji odpowiedzialna jest również za pobieranie danych od użytkownika i prawidłowe dostarczenie ich do aplikacji przetwarzającej te dane. W tym miejscu wykonywana jest również wstępna walidacja otrzymywanych danych oraz ochrona przed potencjalnymi atakami zanim zostanie nawiązane połączenie z serwerem. Chociaż główna odpowiedzialność oraz ochrona przed atakami ciąży na warstwie logiki biznesowej to dobrze zaprojektowana warstwa prezentacji, może skutecznie wyeliminować na wstępnie dużą część ataków lub zminimalizować ich skutki. Ataki, które mogą być wstępnie zablokowane przez warstwę prezentacji to Cross-Site Scripting(XSS) [[65]](#footnote-65), Clickjacking[[66]](#footnote-66) oraz atakami typu phishing[[67]](#footnote-67)

### 3.1.1 Omówienie wymogów wizualnych aplikacji

Aplikacje przeznaczone dla sektora medycznego mają przed sobą wiele wymogów i standardów, które musza spełniać. Branża medyczna jest bardzo wrażliwa na punkcie prawidłowego przetwarzania i przechowywania danych. Każdy błąd wprowadzony do dokumentacji pacjenta może powodować szereg poważnych problemów takich jak nieprawidłowo dobrane leczenie, nietrafione zalecenia lub inne bardzo poważne błędy. Z tego powodu przyjmuje się, że aplikacje przeznaczone dla personelu medycznego powinny być tworzone w taki sposób aby być jak najbardziej intuicyjne i czytelne. Dodatkowo powinny w jak największym stopniu analizować wprowadzane dane przez użytkownika aby zminimalizować ryzyko wprowadzenia błędnych danych do systemu. Podczas projektowania warstwy prezentacji dla systemu LucyCover przeanalizowano jakie błędy może popełnić potencjalny użytkownik i w jaki sposób zminimalizować te ryzyko. W związku z tym podczas tworzenia forntendu zwrócono szczególna uwagę na czytelność i intuicyjność aplikacji. Wszystkie kluczowe elementy powinny być odpowiednio duże i rzucające się w oczy. Wszystkie istotne i najczęściej używane funkcjonalności powinny być widoczne na pierwszym planie. W całej aplikacji zastosowano jednolitą i spójna kolorystyka z podkreśleniem najczęściej wybieranych opcji kontrastowym kolorem. Dominującym kolorem jest jasny róż, który nawiązuje do tematyki kobiecej, macierzyństwa i sektora położniczego.

A screenshot of a computer

Description automatically generated

(Wykorzystanie jednolitej kolorystyki)

Podczas projektowania starano się w jak największym stopniu ograniczyć ilość wyświetlanych komunikatów aby nie rozpraszać użytkownika podczas korzystania z aplikacji. Dodatkowo podczas korzystania z konkretnych funkcjonalności aplikacja powinna ograniczyć liczbę niepotrzebnie wyświetlanych elementów. Z tego powodu w wielu miejscach zastosowano tzw. Popup [[68]](#footnote-68) oraz overlays[[69]](#footnote-69). Dzięki temu na ekranie przysłaniamy aktualnie nie potrzebne funkcjonalności a wyświetlamy jedynie potrzebne opcje. Pozwala nam to skupić uwagę użytkownika na tych elementach na których nam aktualnie zależy. Dzięki temu zmniejszamy ryzyko popełnienia błędu przez pracownika podczas wprowadzania danych do systemów. Na następnej stronie przedstawione jest w jaki sposób wspomniane „popup” zwraca uwagę użytkownika na konkretną aktualnie istotną treść.

A computer screen shot of a computer

Description automatically generated

(Wykorzystaie popup do skupienia uwagi użytkownika na konkretnych funkcjonalnościach)

### 3.1.2 Komunikacja komponentów w oparciu o technologie REDUX

Aplikacje klienckie tworzone w technolog React opierają się o tzw. Komponenty (eng. Componets[[70]](#footnote-70)). Są to specjalnie moduły wykonujące funkcje, które mają za zadanie przygotować, wyświetlić oraz prawidłowo przetworzyć otrzymywane/pobierane dane. W komponentach reactowych spotykamy się z pojęciem „stanu komponentu” (eng. Component state[[71]](#footnote-71)). Opiera się na tym, że każdy komponent przechowuje własne dane, które mogą być zmieniane w czasie i wpływają na jego wygląd oraz zachowanie. Stan jest jedną z podstawowych koncepcji reactowych. Jego zmiany wpływają na cały cykl „życia” komponentu. Zmiana stanu pozwala na ponowne przeładowanie statycznego komponentu odświeżając widok oraz wykonując ponownie utworzone wewnętrzne skrypty.

Problem pojawia się w sytuacji kiedy aktualnie wyświetlany na ekranie widok/strona utworzony jest z wielu różnych powiązanych ze sobą komponentów. Zamysł stanu polega na tym, że dane wewnątrz jednego komponentu są odseparowane i niema do nich dostępu z innego miejsca aplikacji. Co w sytuacji kiedy na przykład jeden główny komponent odpowiada za pobranie jakiś danych z serwera za pośrednictwem interfejsu API a inny podległy komponent wykorzystuje te dane do wyświetlenia ? Niestety komponenty nie działają w sposób podobny do klas. Komponenty podrzędne nie mają dostępu do danych przechowywanych w komponentach im nadrzędnym. Na rozwiązanie tego problemu jest kilka rozwiązań takich jak „prop drilling[[72]](#footnote-72)” lub stosowanie wbudowanego mechanizmu Context[[73]](#footnote-73).

W aplikacji Lucycover zastosowano technologie REDUX która rozwiązuje problem globalizacji stanu w sposób bardzo wydajny i stosunkowo prosty w implementacji. Redux jest zewnętrzną biblioteką, która nie jest częścią Reacta ale wykorzystuje wbudowane w niego mechanizmy. Deweloperzy technologii Redux zaproponowali aby cały główny stan aplikacji przechowywany był w jednym centralnym miejscu nazywanym store[[74]](#footnote-74) do którego wszystkie komponenty będą miały dostęp globalny i za pomocą specjalnych funkcji zwanych „akcje” (eng. Actions) oraz reduktory (eng.”reducers”) będą mogły pobierać i modyfikować dane w konkretny sposób.

Bardzo upraszczając zasadę działania tej technologii tworzymy globalny magazyn w, którym będziemy przechowywać wszystkie potrzebne dane pokategoryzowane za pomocą tzw. Kawałków (eng. Slice). Następnie dajemy dostęp dla wszystkich komponentów do tego magazynu za pomocą specjalnego providera, który jest swego rodzaju komponentem globalnym nadrzędnym nad wszystkimi innymi.

Sposób implementacji tej technologii w LucyCover jest dokładnie taki sam. Na poniższym fragmencie aplikacji pokazano w jaki sposób został zdefiniowany wspomniany wyżej store do którego następnie przypisano reducery przechowujące dane udostępniane przez poszczególne slicy. Przedstawione zdjęcia pokazują, że faktycznie STORE, można przyrównać do globalnego magazynu, w którym przechowywane są obiekty nazywane SLICE, składające się z danych oraz funkcji zwanych ACTIONS pozwalających je modyfikować.

A screen shot of a computer program

Description automatically generated

(Definicja store)

Kolejny fragment przedstawia w jaki sposób utworzony został jeden ze SLICE przechowujący dane wpisywane przez użytkownika do formularza zbierającego dane na temat zaleceń wystawionych przez lekarza dla pacjenta. Jak widać na załączonym wycinku kodu jako domyślny stan przechowujemy obiekt zawierający kolejne obiekty zawierające dane na temat wartości wprowadzonej do pola tekstowego oraz informacji czy dana wartość jest prawidłowa z punktu widzenia mechanizmu walidacji. Pole reducers zawiera funckje tzw. Akcje, które pozwalają modyfikować dane w określny sposób.

Dla przykładu akcja „setInput” przyjmuje z pola tekstowego obiekt do parametru action zawierający informacje jaka wartość została wpisana oraz czy jest ona prawidłowa.

A screen shot of a computer program

Description automatically generated

(definicja slice)

Aktualnie w aplikacji przechowywany jest szereg różnych obiektów typu SLICE przechowujących dane na temat danych wprowadzanych do wszystkich formularzy aplikacji, informacji na temat wyświetlania opisywanych wcześniej widoków popup czy chociażby konfiguracji menu głównego. Wraz z rozwojem aplikacji i co za tym idzie zwiększającą się potrzebą powstawania kolejnych struktur danych liczba slice będzie się stale zwiększać.

Tak przygotowany STORE został przypisany do specjalnego providera, który jak już wspomniano wcześniej jest elementem nadrzędnym całej aplikacji. Zdefiniowany jest w głównym pliku „main.js” i obejmuje on wszystkie podrzędne komponenty dając im dostęp do globalnego stanu zawierającego zdefiniowane SLICE.

A screen shot of a computer program

Description automatically generated

### 3.1.3 Omówienie routingu aplikacji z wykorzystaniem bibloteki React Router

Routing w aplikacjach typu SPA [[75]](#footnote-75) (Single page applications) odnosi się do procesu nawigacji pomiędzy różnymi widokami aplikacji. Dzięki niemu jesteśmy w stanie zapewnić użytkownikowi treści zmieniające się w sposób dynamiczny w obrębie jednej strony. Dzięki temu otrzymujemy bardziej płynne doświadczenia, oraz możliwość czytelnego i eleganckiego kategoryzowania widoków na odpowiednie podstrony. Niestety Routing nie jest wbudowaną funkcjonalnością technologii react. Na szczęście istnieją zewnętrzne biblioteki stworzone specjalnie na potrzeby dostarczenia funkcjonalności routingu do aplikacji typu SPA.

Przykładem takiego rozwiązania jest React Router. Jest to wysoko wydajna biblioteka służąca do obsługi routingu [[76]](#footnote-76) (czyt. Nawigacji) . Pozwala w względnie prosty sposób skonfigurować trasy (eng. routes) oraz zarządzać nimi przy pomocy ścieżek URL[[77]](#footnote-77). Dodatkowo React router udostępnia szereg różnych metod oraz wbudowanych Hooks [[78]](#footnote-78) pozwalających w jeszcze prostszy sposób przełączać się pomiędzy widokami. React router dodatkowo oferuje wbudowane mechanizmy takie jak „loader” oraz „action” umożliwiających uzupełnienie komponentu o dane i przygotowanie komponentu przed wyświetleniem ( w przypadku Loader’a) lub odpowiednie przygotowania danych do wysłania po na przykład wysłaniu formularza ( w przypadku actions).

Aplikacja LucyCover posiada proste menu pozwalające przełączać się pomiędzy poszczególnymi zakładkami takimi jak

* **Kalendarz wizy**t – planowanie wizyt i zadań w kolejnych dniach.
* **Pacjenci** – Zarządzanie pacjentami w systemie, obsługa dokumentacji oraz rekomendacji pacjenta
* **Materiały** **edukacyjne** – Menadżer plików umożliwiający przechowywanie materiałów edukacyjnych oraz udostępnianie ich pacjentom
* **Korespondencja** – Zakładka pozwalająca korespondencje pomiędzy systemem a pacjentem

Dzięki wykorzystaniu React router zachowano czytelność kodu i intuicyjny podział modułów aplikacji na konkretne usługi.

Cała technologia routingu została zaimplementowana w pliku ./ LucyCover - Frontend/src

/App.jsx, W tym miejscu utworzono router korzystając z wbudowanej metody ***createBrowserRouter*** [[79]](#footnote-79) pozwalającej na zdefiniowanie ścieżek aplikacji. Z uwagi na mnogość stron i podstron szablon wykorzystanego routera przypomina drzewo tworzone w systemach plików. Następnie tak przygotowany router został przypisany do wbudowanego komponentu udostępnianego przez technologie wspomagająca React-router-dom[[80]](#footnote-80) udostępniająca między innymi komponent RouterProvider, który staje głównym komponentem całej aplikacji.

Poniższy przykład prezentuje fragment zdefiniowanego routera który definiuje trasę dostępu do panelu głównego aplikacji oraz podległą mu podstronę „patients”

A computer screen shot of a program code

Description automatically generated

(fragment utworzonego providera)

Następnie możemy zauważyć jak tak przygotowany router zostaje przypisany do specjalnego providera dostarczonego przez React-router-dom, który zmienia zasadę działania aplikacji. Od teraz dostęp do poszczególnych komponentów nie zależy od ich zdefiniowania w komponencie App.jsx tylko od konkretnej ścieżki wpisanej przez użytkownika, którą obsługuje router wyświetlając przypisany do niej komponent.

A computer screen with white text

Description automatically generated

### 3.1.4 Zaimplemetowanie własnych Hook’s

W React 16.8 przedstawiono zamysł nowej funkcjonalności zwanej React Hooks. Są to specjalne fukcje, które pozwalają zarządzać stanem oraz wykorzystywać funkcje udostępniane przez reacta bez konieczności tworzenia komponentów funkcyjnych z logiką podobną do tej wykorzystywanej w komponentach bazowych.

Aplikacja LucyCover wykorzystuje tą funkcjonalność wszędzie tam gdzie pojawia się temat zarządzania stanem aplikacji lub wykorzystania jakiś funkcji reactowych w wielu miejscach w sposób generyczny. Autorskie Hooks wykorzystywane są aby zwiększyć czytelność i wydajność kodu oraz aby utrzymywać tok pisania aplikacji w systemie „clean code[[81]](#footnote-81)”. Dzięki tej technologii aplikacja sprawnie unika sytuacji kiedy ten sam lub bardzo zbliżony fragmentu kodu występuje wiele razy. Dzięki odpowiednio utworzonym Hooks możemy napisać jeden fragment kodu wykonujący jakąś operacje bazując na generycznych typach danych, który następnie zostanie wykonany w wielu miejscach aplikacji.

LucyCover wszystkie autorskie Hooks przechowuje w katalogu ./src/hooks. Przykładem takiej funkcji jest useFormData ( zgodnie z dokumentacją wszystkie Hooks muszą zaczynać się od słowa kluczowego use)[[82]](#footnote-82). Jest to funkcja wykorzystywana w wielu miejscach aplikacji pozwalająca na pobrania gotowych wartości z obiektów zapisanych w slice [[83]](#footnote-83), które następnie zostają wysłane na serwer.

A screen shot of a computer program

Description automatically generated

Powyższy przykład przedstawia sposób implementacji useFormData. Jak widać na załączonym fragmencie kodu. Hooks jako wyniki jego wywołania zwraca funkcje wykonującą zapisane polecenia. W pierwszej kolejności następuje sprawdzenie czy przekazany parametr jest obiektem. Następnie wykonywana jest pętla sprawdzająca wszystkie pola obiektu i pobiera dane z właściwości value, które następnie zostają przypisane do obiektu FormDataValues.

Mówiąc kolokwialnie funkcja ta zamienia obiekt typu {value:string; isValid:bool} na obiekt {value:string}, pozbywając się informacji na temat walidacji konkretnego pola formularza, które nie są już potrzebne podczas zapisu do bazy danych.

Używanie własnych autorskich Hooks znacząco przekłada się na jakoś wytworzonego kodu, pozwala odseparować często używaną logikę systemu od komponentu odpowiedzialnego za wyświetlenie, i w elegancki sposób wykorzystuje założenia przedstawione przez deweloperów Reacta.

### 3.1.4 Komunikacja z backendem z wykorzystaniem React TanstackQuery

Wraz ze wzrostem popularności aplikacji webowych znacząco wzrosło tępo powstawania nowych technologii oraz wzorców projektowych. Pojawienie się podziału na aplikacje klienckie oraz aplikacje serwerowe pozwoliło zwiększyć wydajność obsługi kolejnych zapytań http. Dzięki podziałowi pracy na dwa niezależne serwery odpowiadające kolejno za dostarczenie skompilowanych statycznych plików html (np. apache[[84]](#footnote-84)) oraz chmur obliczeniowych uruchamiający aplikacje backendowe przetwarzające dane zyskaliśmy segmentowość aplikacji przyczyniających się bezpośrednio na wzrost wydajności oraz skalowalności aplikacji.

Niestety takie rozwiązanie przyniosło też kolejne problemy do rozwiązania takie jak zaimplementowanie odpowiednio wydajnego sposobu komunikacji pomiędzy dwoma aplikacjami. Wbudowane mechanizmy dostarczane przez JavaScript są dość skomplikowane w implementacji, jeżeli chcemy zachować jak najbardziej przystępny user experience [[85]](#footnote-85). Naprzeciw tej sytuacji przychodzi nowa technologia dostarczona przez studio TanStack – TanStack Query. Technologia ta dostarcza kompleksowe wsparcie podczas pisania w pełni wydajnych i złożonych zapytań http. Udostępnia również informacje na temat statusu procesowania aktualnego zapytania co pozwala płynnie wyświetlać użytkownikowi na jakim etapie jest jego praca.

LucyCover wykorzystuje wyżej wspomnianą bibliotekę podczas wszystkich prób łączenia się z aplikacją serwerową. Wszystkie funkcje zawierające instrukcje „fetch[[86]](#footnote-86)” zostały spisane i wyeksportowane w pliku http.js. Plik ten stanowi zapis proceduralny w jaki sposób technologia TanStack Query ma wykonać zapytanie do serwera. Każda funkcja składa się z czasownika http obsługi zwracanych wartości i statusów. W tym miejscu również znajdziemy instrukcje mówiące jak obsłużyć każdy ze spodziewanych wyjątków. Tak przygotowana instrukcja może zostać przypisana do hook’s udostępnianych przez TanstackQuery takich jak „useMutation” czy „useQuery”. Dzięki wykorzystaniu tej technologii automatycznie wykonyujemy zapytania do serwera w sposób asynchroniczny co znacząco wpływa na płynność aplikacji, ponieważ zamrożone podczas czekania na odpowiedź serwera wątki zwracane są do puli a użytkownik nie ma wrażenia, że aplikacja się zawiesiła. Zamiast tego jako wynik zwracany przez useMutation oraz useQuery otrzymujemy zmienne zawierające informacje takie jak isPending, isSuccess, isError, które umożliwiają wyświetlenie konkretnych widoków na każdy przypadek. Dla przykłady podczas procesowania zapytania isPending zawiera wartość true, którą możemy wykorzystać aby wyświetlić tzw. Loading spiner [[87]](#footnote-87).

Na poniższym rysunku zaprezentowano przykład działania zapytania GET do serwera, który pozwoli załadować dane z dokumentacji pacjenta przed wyświetleniem widoku na ekranie. Pierwszy fragment kodu znjaduje się w pliku api/http.js i jego zadaniem jest zapisanie instrukcji w jaki sposób należy połączyć się z endpointem i jak obsłużyć wyniki przez niego zwracane. Kolejny fragment kodu to loader wykorzystujący jedną z funkcji TanStack Query do pobrania danych z serwera i przesłanie ich do widoku.



Podsumowując informacje zawarte w wyżej przedstawionym rozdziale. Podział aplikacji webowych na aplikacje klienckie i serwerowe jest swego rodzaju przełomem w tym sektorze programowania a dostępne technologie takie jak TanStack Query w prosty i intuicyjny sposób pozwalają nam szybko tworzyć bezpieczne i wydajne połączenia pomiędzy dwiema aplikacjami. LucyCover chętnie wykorzystuje dostępne nowe i stabilne technologie w ten sposób zapewniając dobrą jakość i niezawodność użytkowania.

### 3.1.5 Walidacja w warstwie prezentacji

Walidacja danych w warstwie aplikacji jest kluczowym aspektem tworzenia bezpiecznych i stabilnych systemów informatyczny. Jest to swego rodzaju interfejs pomiędzy serwerem wykonującym cała logikę biznesową oprogramowania oraz zarządzający komunikacją z bazą danych. Z tego powodu aby zminimalizować ryzyko potencjalnych ataków lub pomyłek użytkownika, które mogłyby dostać się do serwera należy zawsze przeanalizować w jaki sposób uniemożliwić lub utrudnić użytkownikowi wprowadzanie niechcianych informacji. W większości przypadków podczas budowy profesjonalnego oprogramowania aplikacje backendowe również są przygotowane do walidowania otrzymywanych danych na swoich endpointach przed przekazaniem ich do serwisów przetwarzających. Jednakże każde takie błędne zapytanie przyczynia się już do obciążenia serwera. Z tego powodu zakłada się, że większość błędów lub ataków można zatrzymać już na etapie aplikacji klienckich.

LucyCover jako aplikacja przeznaczona dla sektora medycznego bardzo rygorystycznie podchodzi do przyjmowanych danych. Każdy błąd może mieć bardzo poważne konsekwencje a ataki, które mogą skutkować wyciekiem lub utratą danych będą katastrofalne w skutkach. Z tego powodu zastosowano koncepcje „Defense in depth”. Koncepcja ta polega na stosowaniu wielu warstw zabezpieczeń aby zapewnić maksymalną ochronę. Pierwszą warstwą jest właśnie aplikacja kliencka tworzona w technologii React. W związku z tym zaprojektowano własny system obsługi formularzy wyświetlanych klientowi. Zaimplementowano autorski komponent o nazwie ./LabelInput.jsx, który współpracuje z odpowiednio przygotowanymi SLICE zawierających informacje na temat statusu walidacj danych.

Uproszczając zasadę działania systemu walidacyjnego LucyCover można przyjąć, że opiera się ona o 3 elementy. Pierwszym elementem jest formularz tworzony w konkretnym widoku. Taki formularz zawiera szereg pól tekstowych zdefiniowanych w pliku ./LabelInput.jsx. Jest to komponent zawierający zwykły element HTML wspierany przez wiele funkcji react. Komponent ten przyjmuje kilka parametrów. Jednym z nich jest REQUIRED informujący, że dopóki użytkownik nie poda żadnej wartości pole tekstowe zawsze będzie zwracało informacje o błędnie wypełnionej wartości. Kolejnym istotnym parametrem z punktu widzenia bezpieczeństwa i walidacji jest parametr validationFunction. Pozwala on przekazać do komponentu wzorzec, który następnie będzie sprawdzony w kontekście wprowadzonej przez użytkownika wartości.

Po tak przeprowadzonej walidacji wartości wprowadzanych przez użytkownika, komponent labelInput wymaga podania funkcji onInput. Definiuje ona w co ma zostać wykonane po wprowadzeniu danych przez użytkownika oraz ich walidacji. W tym miejscu w komponencie w którym definiujemy taki formularz utworzony jest specjalny handler który zapisuje informacje o każdym polu tekstowych w swoim SLICE. Podczas próby wysłania formularza wszystkie zmagazynowane w slice pola tekstowe zostają pobrane, następnie następuje sprawdzenie czy każde pole zwróciło pozytywną informację dot. Poprawności danych. Po sprawdzeniu wszystkich pól tekstowych formularz umożliwia naciśniecie przycisku uruchamiającego funkcję wysyłania danych. Niestety w związku z wzrostem ilości ataków typu DOM-based Manipulation[[88]](#footnote-88) funkcja wysyłająca, również ponownie musi sprawdzić poprawność danych na wypadek gdyby osoba atakująca manualnie usunęła zabezpieczenia przycisku przed niepoprawnym wysłaniem formularza.

W ramach podsumowania warto zaznaczyć, że dzisiejsze możliwości tworzenia nowoczesnych aplikacji webowych przynoszą wiele korzyści ale i równie wiele kolejnych zagrożeń. W związku z tym zakłada się, że podczas tworzenia oprogramowania powinno się osobno zabezpieczyć każdą warstwę aplikacji i konsekwentnie sprawdzać wszystkie dane dostarczane przez użytkowników końcowych.

### 3.1.6 Zapewnienie nowoczesnego designu dzięki FramerMotion

Rozwój technologii do budowania nowoczesnych aplikacji webowych pozwolił na znacznie szybsze i wydajniejsze implementowanie animacji. Podczas gdy kilka lat temu większość aplikacji klienckich wspierana była przez czysty kod JavaScript pisanie ciekawych i zaawansowanych animacji zajmowało wiele czasu i generowało sporą ilość dodatkowego kodu co wpływało na zmniejszenie wydajności a przede wszystkim czytelności kodu. W 2006 roku przełom wprowadził John Resig [[89]](#footnote-89)wydając do użytku bibliotekę JQUERY[[90]](#footnote-90), która ustandaryzowała sposób tworzenia aplikacji i usprawniła ich działania na różnych przeglądarkach. Dzięki temu tworzenie animacji w aplikacjach webowych stało się znacznie prostsze.

Aktualnie programiści webowi mają do dyspozycji potężną bazę nowych dużych technologii oraz tysięcy bibliotek wspomagających ich w pracy. Z tego też powodu JQUERY mimo swej popularności zaczęło odstawać pod względem możliwości i wydajności ponieważ nie było tak szybko rozwijane i dostosowywane do aktualnych standardów. Jedną z nowoczesnych bibliotek rozbudowujących wbudowane funkcje technologii react jest Framer Motion dodający mnóstwo nowych rozwiązań i gotowych animacji do projektu.

LucyCover wykorzystuje tą technologie podczas wyświetlania informacji na ekranie, płynnych przejść pomiędzy widokami oraz do atrakcyjnego wyświetlania komunikatów i treści na ekranie. Sposób implementacji jest prosty. Framer Motion dostarcza szereg komponentów zastępczych dla aktualnie wbudowanych elementów DOM dla JSX.

Dzięki tej technologii mamy wrażenie, że aplikacja działa w sposób bardziej dynamiczny i płynny. Dane wyświetlane są w atrakcyjny sposób a komunikaty o potencjalnych błędach bardziej zwracają uwagę użytkownika. Jednocześnie wszystkie te animacje nie zmniejszają czytelności kodu i szybkości jego wykonywania.

### 3.1.7 Zawansowany serwer aplikacji Vite

Przyglądając się dokładniej w jaki sposób tak naprawdę wykonywane są aplikacje tworzone w technologiach takich jak React pojawia się wiele różnych pytań. Skoro przeglądarki internetowe przystosowane są tylko do interpretowania języków HTML,CSS oraz JavaScript to dlaczego aplikacja utworzona w technologii React uruchamia się na niej prawidłowo ? Struktura aplikacji reactowej opiera się na wykorzystaniu własnego języka JSX, który niestety w żaden sposób nie będzie zrozumiały dla przeglądarki mimo, że wykorzystujemy tam znaczniki dobrze znane z HTML. Aby odpowiedzieć na te pytanie musimy zrozumieć w jaki sposób wykonywany jest kod reactowy i w jaki sposób działają bundlery[[91]](#footnote-91).

Aby móc rozwijać aplikacje w technologi react potrzebujemy specjalnego narzędzia uruchomieniowego dla tej technologii. Przykładem takiego narzędzia jest technologia VITE, która szybko stała się hitem wśród deweloperów.

Vite jest niezbędnikiem działającym w tle podczas wszystkich faz rozwoju aplikacji LucyCover. Zaczynając od fazy deweloperskiej, kiedy aplikacja jest rozwijana technologia Vite zapewnia serwer pozwalający uruchomić aplikację na lokalnym serwerze. W związku z tym, że przeglądarka nie jest w stanie interpretować języka JSX serwer dostarczony przez Vite w ciągu ułamków sekund konwertuje wytwarzany kod na wysoko wydajny i skompresowany kod JavaScript który następnie jest uruchamiany w przeglądarce. Kluczowym atutem Vite jest wykorzystywanie ESM (ECMAScript Modules)[[92]](#footnote-92) oraz HMR (Hot Module Replacement)[[93]](#footnote-93) dzięki czemu zmiany wprowadzane w kodzie widzimy natychmiast bez konieczności ciągłego odświeżania strony po wprowadzeniu zmian.

Po zakończeniu fazy deweloperskiej Vite dostarcza nam funkcjonalność „budowania” aplikacji i przystosowania jej do wdrożenia na serwer zdalny do obsługi protokołu http. Dzięki komendzie „build” Vite dostarcza nam gotową paczkę projektu z zamienionym kodem JSX na czysty kod JavaScript, który jest mało czytelny i zrozumiały dla człowieka za to bardzo skompresowany i wydajny podczas działania.

W ramach podsumowania musimy uświadomić sobie, że React jest tak naprawdę technologią stworzoną do celów usprawnienia pracy zespołów deweloperskich. Pozwala tworzyć aplikacje szybciej korzystając z paradygmatu programowania deklaratywnego oraz zapewnia szereg wygodnych i wydajnych bibliotek. Niestety jego składnia jest całkowicie niezrozumiała dla przeglądarki w związku z tym nie jesteśmy w stanie wykonać takiego kodu bezpośrednio. Naprzeciw tym problemom wychodzi technologia Vite, która udostępnia specjalny serwer konwertujący w czasie rzeczywistym wytwarzany kod reactowy na JavaScript, oraz udostępnia funkcje przeznaczone do finalnego „zbudowania” aplikacji na kod gotowy do bezpośredniego wdrożenia na serwery http.

## 3.2 Implementacja warstwy dostępu do danych

W architekturze trójwarstwowej warstwa dostępu do danych jest jedną z trzech głównych warstw. Odpowiedzialna jest za bezpośredni dostęp do źródła danych korzystając z komunikacji z serwerami bazodanowymi czy plikami przechowywanymi w systemie. Warstwa ta zapewnia stabilną komunikację z bazą danych oraz pozwala zachować pewny rodzaj abstrakcji. Modyfikacje i dostępy do bazy danych są odseparowane w tej warstwie od reszty aplikacji co pozwala zwiększyć bezpieczeństwo, skalowalność i prostotę aplikacji.

### 3.2.1 Omówienie wymogów przechowywania danych

Aplikacja LucyCover ma postawione przed sobą szereg wytycznych i norm jakie muszą zostać spełnione/zachowane aby móc funkcjonować w sektorze medycznym. Sektor ten jest bardzo wrażliwy na wszelkiego rodzaju wycieki czy utraty danych. Dane medyczne klasyfikują się do ściśle strzeżonych i ich utrata bądź wyciek niesie ze sobą szereg poważnych konsekwencji. W związku z tym podczas analizy biznesowej wymogów aplikacji ustanowiono szereg wytycznych i norm jakie powinny zostać zachowane podczas projektowania i zarządzania aplikacją:

1. Dostęp do danych powinien być osobna abstrakcją, która komunikuje się z pozostałymi modułami za pomocą udostępnionych interfejsów.
2. Zarządzanie bazą danych powinno opierać się o model ORM wykorzystując technologię EntityFramework Core
3. Tworzenie i zarządzanie bazą danych powinno być wykonane w modelu Code first
4. Dostęp do danych powinien być zapewniany w wzorcu projektowym Repository
5. Dane zapisywane w bazie danych powinny być każdorazowo szyfrowane
6. Użytkownik końcowy powinien dostawać tylko niezbędne informacje. Wszystkie dane pobierane z bazy powinny być ograniczane do minimum korzystając z modeli DTO

Mając na uwadze te wszystkie informacje zaprojektowany został specjalnie odseparowany osobny projekt będący abstrakcją realizujący wszystkie założenia nie zbędne do prawidłowej komunikacji z serwerem bazodanowym. Do obsługi centralizacji danych wykorzystano bazę danych Microsoft SQL współpracującą z językiem zapytań T-SQL będącym inicjacją języka SQL. Na potrzeby jednolitej centralizacji wszystkich modeli występujących w programie powstała osobna biblioteka przechowująca wszystkie wykorzystywane modele w bazie danych oraz modele DTO wykorzystywane w aplikacji podczas zwracania rezultatów. Na potrzeby mapowania encji bazodanowych na modele DTO zaimplementowano technologię mappe, która zostanie omówiona w kolejnych rozdziałach.

### 3.2.2. Code First – podejście definiowania modeli baz danych

Technologia Entity FrameWork dostarcza szereg funkcjonalności i możliwości definiowania komunikacji pomiędzy aplikacją a serwerem bazodanowym. Z tego powodu projektant aplikacji musi podjąć decyzje w jaki sposób planuje budować swój model bazodanowy. LucyCover wszystkie dane gromadzi na serwerze bazodanowym dostarczanym darmowo od firmy Microsoft – Microsoft SQL Serwer. Baza danych jest rozbudowywana na bieżąco podczas implementowania kolejnych usług aplikacji. Podejście to nazywa się CodeFirst. Zamysł tego podejścia polega na definiowaniu wykorzystywanych modeli które chcemy aby znajdowały się w bazie danych. Modele te są aplikacyjnym odzwierciedleniem tablic magazynowanych w bazach danych. Definiowane są bezpośrednio w kodzie programu i nie wymagają znajomości żadnego z typowych bazodanowych języków zapytań. Entity Framework oraz inne pomocnicze biblioteki dostarczają szereg różnych atrybutów pozwalających w prosty i elegancki sposób definiować sposób inicjacji takiego modelu w bazie danych. Dla przykładu definiując model w, którym jednym z pól będzie pole przechowujące identyfikator, możemy poprzedzić go wbudowanym atrybutem [KEY], który poinformuje narzędzie dostarczane wraz z EntityFramework Core, że podczas tworzenia tablicy bazy danych takie pole ma zostać zapisane jako klucz główny.

Podejście code first wymaga wykorzystywania odpowiedniego narzędzia dostarczanego najczęściej razem z wybranym systemem ORM (Object-Relational Mapping) do tworzenia tak zwanych migracji[[94]](#footnote-94) (eng. Migrations). Migracje są tworzone przez specjalne narzędzie, które analizuje modele i kod jaki został wytworzony w aplikacji i generuje specjalny plik pozwalający utworzyć bądź dostosować bazę danych.

LucyCover do zapewniania odpowiednio przygotowanych migracji wykorzystuje biliotekę Microsoft EntityFrameworkCore Tools[[95]](#footnote-95), która pozwala wygenerować migracje, wdrożyć ją na serwer bazodanowy oraz dodatkowo zapewnia historię wprowadzanych zmian w bazie danych pozwalając na szybkie wycofanie i przywrócenie poprzedniego stanu.

Na poniższy zrzutach ekranu zaprezentowano sposób w jaki dla przykładu została zdefiniowana tablica przechowująca informacje na temat zaleceń wystawionych przez położna dla swojego podopiecznego. Pierwszy scren prezentuje zadeklarowany model recommendations

A screen shot of a computer program

Description automatically generated

Na kolejnym załączniku pokazany został sposób jego inicjalizacji w kontekście bazodanowym skonfigurowanym dla LucyCover. Załącznik pokazuje, że modele definiowane są w kontekście jako obiekt generyczny typu DbSet

A computer screen shot of a code

Description automatically generated

Taka inicjalizacja modelu w kontekście bazy danych pozwala na odpowiednie przygotowanie migracji która wprowadzona do bazy danych utworzy tablice recommendations konwertując pola zdefiniowane w modelu na kolumny w tabeli bazy danych. Na kolejnym załączniku wykazano jak narzędzie EntityFrameworkCore Tools zrealizowało taką konwersję modelu na migracje.

A screen shot of a computer program

Description automatically generated

Powyższy załączniki przedstawia sposób utworzenia tabeli recommendation zawierający cztery pola. Na uwagę zasługuje sposób w jaki narzędzie rozpoznało, że pole ID powinno zostać ustawione jako klucz główny tabeli a pole PatientId powinno stanowić klucz obcy do tabeli Patient. Co ciekawe migracja dodatkowo pozostawia nam możliwość szybkiego wycofania zmian udostępniając metode Down, która zawiera instrukcje przywracające. Jest to bardzo pomocne w sytuacji pomyłek.

### 3.2.2. Implementacja kontekstu baz danych w projekcie

Korzystając z nowoczesnych rozwiązań wspomagających komunikację projektu z bazami danych należy rozważyć wybór pomiędzy małymi lekkimi bibliotekami ułatwiającymi szybkie tworzenie prostych zapytań do bazy danych oraz dużymi systemami ORM. Systemy ORM są dobrym wyborem podczas tworzenia dużych aplikacji wykonujących wiele zapytań do bazy danych w krótkich odstępach czasu, oraz kiedy deweloper ma do czynienia z większymi zbiorami danych wymagających wykorzystania zaawansowanych aspektów przechowywania takich jak indeksy, transakcje. Po krótce mówiąc system ORM jest dobrym wyborem kiedy ilość danych jest odpowiednio wysoka a ich przyrost występuje szybko i regularnie. Wówczas systemy

Object–relational mapping zapewniają wbudowane mechanizmy pozwalające zwiększyć optymalizacje podczas przechowywania danych oraz zapewnić sprawny dostęp do bazy danych.

Tak jak wspomniano wcześniej LucyCover korzysta z rozwiązania opartego o system ORM – EntityFrame worke core. U podstaw tej technologii jest odpowiednie sformułowanie sposobu dostępu do bazy danych. Do tego celu wykorzystuje się specjalnie przygotowaną klasę dbContext oraz specjalnie przygotowany i skonfigurowany connectionString. W ramach wyjaśnienia czym są po krótce wspomniane wyżej elementy, element DbContext jest klasą dostarczną domyślnie przez EntityFramework. Pozwala na utworzenie instancji zawierającej pełna konfigurację połączenia z bazą danych, zdefiniowanie modeli danych, które będą w niej przetrzymywane oraz jakie mechanizmy i funkcje powinny zostać wykorzystane podczas tworzenie i uruchamiania bazy danych. Zgodnie z dostarczoną przez studio deweloperskie dokumentacją LucyCover implementuje własną klase DbConnection, która dziedziczy po klasie DbContext uzyskując dostęp do koniecznego konstruktora bazowego. DbConnection zaimplementowany w LucyCover pobiera argument typu DbContextOptions<>, który wskazuje na odpowiednio skonfigurowany connectionString, który zostanie omówiony w dalszej części rozdziału.

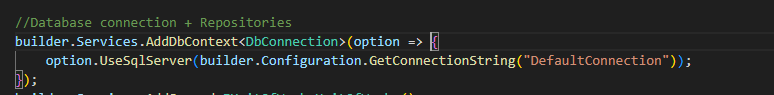
Poniższy fragment kodu prezentuje klasę DbConnection utworzoną na potrzeby konfiguracji połączenia między aplikacją backendową a serwerem bazodanowym.

A computer screen shot of a program

Description automatically generated

W pierwszych liniach widać deklarację klasy DbConnection bazującą na klasie dostarczonej przez system ORM (DbContext). Następnie zadeklarowany został konstruktor przyjmujący argument będący reprezentacją connectionString, który następnie przekazany zostaje do konstruktora bazowego klasy DbContext. W dalszej części następuje zadeklarowanie modeli danych, które następnie staną się wzorem do utworzenia tabeli w bazie danych po wykonaniu migracji. Działanie migracji oraz sposobu mapowania modeli na tabele został wyjaśniony w rozdziale 3.2.2.

Następnym krokiem podczas implementacji kontekstu bazy danych jest rejestracja klasy DbConnection w kontenerze dependency injaction oraz dostarczenie wcześniej wspomnianego connectionString do konstruktora tej klasy. EntityFramework udostępnia nam klasy oraz metody pozwalające wykonać te dwie czynności jednocześnie podczas gdy konfigurowany jest główny buildera [[96]](#footnote-96) aplikacji ASP.Net Core. Poniższy fragment aplikacji przedstawia sposób przekazania nowej instancji klasy DbConnection do zasobów aplikacji.



Nie wyjaśnione jednak dalej zostało w czym jest wspominany kilkukrotnie connectionString i w jaki sposób przyczynia się do nawiązania połączenia pomiędzy aplikacją a serwerem bazodanowym. ConnectionString jest ciągiem znaków używanym do nawiązywania połączeń z bazą danych oraz innym zasobem sieciowym. W ciągu tym przechowywane są wszystkie niezbędne informacje jak host bazy danych, dane autentykacji jak login oraz hasło, port na którym występuje komunikacja oraz nazwa bazy danych. Tak skonfigurowany connectionString zostaje wykorzystany przez klase DbContext do nawiązania połączenia.

Ze względów bezpieczeństwa oraz skalowalności aplikacji LucyCover przechowuje taki ciąg znaków w pliku appsettings.json. Plik ten przechowuje konfiguracje aplikacji, która pozwala na zmianę pewnych istotnych zachowań bez konieczności ponownego kompilowania aplikacji. Dzięki takiemu podejściu umożliwiamy zmianę wykorzystywanej bazy danych lub nawet całego serwera bazodanowego bez konieczności ingerencji w kompilacje projektu. W ten sposób pozostaje spójna z zasadami SOLID. Pliki konfiguracyjne w formacie JSON pozwalają również na przechowywanie kilku różnych konfiguracji osobnych na przykład dla wersji developerskiej oraz produkcyjnej.

Poniższy fragment przedstawia konfiguracje dla wersji developerskiej programu LucyCover. Jeden z zapisów przechowuje domyślnie używany connectionString.  
  
A screenshot of a computer

Description automatically generated

To właśnie z tego pliku metoda “builder.Configuration.GetConnectionString(„DefaultConnection”)” pobiera tą konfigurację.

Podsumowując rozdział dotyczący implementacji kontekstu baz danych w aplikacji LucyCover, należy zauważyć korzyści wynikające ze stosowania dużych systemów ORM do komunikacją z bazą danych oraz warunki w jakich warto ich używać. LucyCover jest aplikacją obszerną, pokrywająca wiele funkcjonalności i przystosowana do ciągłego rozwoju. W związku z tym wymaga wykorzystywania odpowiednich technologii pozwalających sprostać obsłudze dużych zbiorów danych medycznych. Kontekst oraz konfiguracja połączenia zostały skonfigurowane w sposób pozwalający na łatwą edycję, skalowalność, zachowano czytelność kodu oraz utrzymano założenia dyktowane przez zasady projektowe SOLID.

### 3.3.3 Wykorzystane Modele (DO NAPISANIA)

### 3.3.4. Wykorzystanie wzorca projektowego „Repository”

Repository to wzorzec projektowych stosowany w programowaniu obiektowym, którego głównym zadaniem jest oddzielenie logiki aplikacji od logiki dostępu do danych. Wzorzec ten tworzy swego rodzaju pośrednika pomiędzy tymi dwiema warstwami co ułatwia zarządzanie zestawami danych, pozwala zachować możliwości skalowania aplikacji, wpływa pozytywnie na tworzenie testów oraz ma znaczne odziaływanie na zachowanie czytelności wytwarzanego kodu.

Wzorzec opiera się na dwóch kluczowych elementach takich jak interfejs repozytorium oraz jego implementacje w typie generycznym. W interfejsie definiujemy wszystkie wykorzystywane operacje na danych takie jak: dodaj, usuń, znajdź, zaktualizuj. Natomiast klasa dziedzicząca zawiera implementacje i logikę pokrywającą te funkcje. Dzięki temu możemy utworzyć kilka niezależnych implementacji tego interfejsu wykonujących te same operacje dla różnych systemów bazodanowych. Po utworzeniu głównego interfejsu oraz implementacji dla repozytoriów, należy utworzyć osobne niezależne repozytoria dla każdego modelu danych, który będzie wykonywał wyżej zaimplementowane operacje na konkretnym modelu. Dodatkowo repozytorium przygotowane pod konkretny typ danych oraz dziedziczące po głównym generycznym interfejsie repozytorium mogą przysłaniać dziedziczone metody pozwalając na przetwarzanie danych w inny, specyficzny sposób.

LucyCover korzysta z wyżej opisanego wzorca i jest on jedynym i głównym interfejsem pomiędzy aplikacją oraz bazą danych. Każdy model w aplikacji ma przygotowany osobne repozytorium dziedziczące po głównym generycznym typie. Wzorzec projektowy został zaimplementowany w osobnej bibliotece ./LucyCover-Backend zawierającej omówiony wcześniej DbContext, migracje oraz repozytoria Poniższy przykład pokazuje w jakie opcje udostępnia generyczne repozytorium. Widoczne jest, że interfejs dostarcza ogólną logikę pobierania, dodawania oraz usuwania danych z systemu w sposób ogólny dla wszystkich typów danych. Następnie wykazany został przykład implementacji tego interfejsu a dokładnie metody Add().

A screen shot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generated

Ze względu na ilość instrukcji implementacje pozostałych metod nie zostały wykazane. Załączone przykłady pokazują ogólny sposób przetwarzania danych. Wszystkie pozostałe repozytoria dziedziczą po interfejsie IRepository<T> dodatkowo deklarując model jaki reprezentują. Po takiej deklaracji metody generyczne wykonują operacje na dostarczonym konkretnym modelu. Na kolejnym przykładzie przedstawiono sposób tworzenia repozytoriów specyficznych dla swojego modelu. Takie repozytoria dziedziczą wszystkie metody klasy IRepository dodatkowo udostępniając własną metodę Update, która jest specyficzna dla każdego modelu.

A screenshot of a computer program

Description automatically generated

Pełna lista repozytoriów znajduje się w projekcie „LucyCover-Database” w katalogu „repository”.

Dla celów centralizacji wszystkich repozytoriów jako jeden ogólny serwis zadeklarowano dodatkowo klasę „unitOfWork” wraz z interfejsem, która umożliwia dostęp do wszystkich repozytoriów. Taki serwis przekazany został do kontenera dependency injaction, który przekazuje instancje dla konstruktorów wszystkich korzystających serwisów. W rysunku przedstawiono strukturę unit of work oraz jego rejestracje w kontenerze.

A computer screen shot of a program

Description automatically generated



LucyCover aktywnie korzysta z pojęcia abstrakcji podczas definiowania najważniejszych mechanizmów i struktur aplikacji. Kolejnym przykładem tego podejścia jest wykorzystanie wzorca projektowego repository, który zapewnia nam odizolowanie sposobu komunikacji pomiędzy bazą danych a aplikacja backendową oraz zapewnia nam korzyści takie jak prostota testowania, łatwa zamiana systemów bazodanowych oraz czytelność kodu.

### 3.3.5. Data Transfer Object

W nowoczesnych aplikacjach internetowych przepływ danych pomiędzy aplikacjami jest podstawowym założeniem działania. W każdej sekundzie następuje setki a czasem nawet i tysiące zapytań do web serwerów. Serwery przetwarzają zapytania i zwracają odpowiedzi. Podstawowym wyzwaniem zawsze staje się zapewnienie wydajnej i płynnej komunikacji oraz zachowanie jak najlepszego poziomu zabezpieczenia danych. Nowoczesne web serwisy chronią przetrzymywane dane wykorzystując mechanizmy autoryzacji[[97]](#footnote-97), autentykacji[[98]](#footnote-98), wykorzystują możliwości zabezpieczeń CORSE oraz wielu innych niezależnych bibliotek wspomagających ich pracę. Niemniej jednak nawet jeżeli użytkownik końcowy faktycznie zostanie uwierzytelniony prawidłowo należy zachować odpowiednie formy bezpiecznego przekazywania informacji. Jednym z takich założeń jest ograniczanie ilości przekazywanych danych do absolutnego wymaganego minimum. To znaczy, że dane pobierane z baz danych powinny zostać ograniczone do niezbędnych informacji potrzebnych użytkownikowi. Aby odpowiednio oddzielić dane niezbędne od danych przechowywanych w bazie danych wykorzystujemy modele DTO (Data Transfer Object). Są to obiekty przeznaczone do przesyłania danych pomiędzy różnymi aplikacjami lub procesami. Pojęcie DTO należy rozpatrywać jako dobre wzorce projektowe.

LucyCover wykorzystuje wzorzec projektowy oparty o modele DTO w celu ograniczenia udostępnianych danych dotyczących pacjentów oraz ich danych medycznych. Każdy endpoint kontrolera ma przygotowany specjalny obiekt DTO, który zawiera tylko te informacje potrzebne do prawidłowego wyświetlenia widoku przez aplikacje pracującą w warstwie prezentacji. W aplikacji modele DTO powstają na skutek mapowania pobranych encji bezpośrednio z bazy danych na odrębnie przygotowane modele. Aby zachować czytelność i atomowość kodu wykorzystano technologie AutoMapper, która umożliwia zadeklarowanie sposobów w jaki sposób dwa modele mają zostać mapowane. Następnie AutoMapper zostaje przypisany do kontrolera dependency injaction, który dostarcza jego instancje we wszystkich korzystających serwisach. Poniższy przykład przedstawia przygotowanie danych do wysyłania za pośrednictwem interfejsu API z wykorzystaniem modeli DTO.

A screen shot of a computer program

Description automatically generated

Jest to fragment serwisu odpowiedzialnego za obsługę zakładki „Zalecenia”. Serwis ten jest odpowiedzialny za prawidłowe pobieranie, dodawanie oraz edycję zaleceń pacjenta z serwera bazodanowego. Przedstawiona metoda dotyczy pobrania szczegółów zalecenia, które mają zostać wysłane za pomocą protokołu http do aplikacji klienckiej. W pierwszej kolejności następuje pobranie argumentów dotyczących identyfikatora pacjenta oraz zalecenia. Następnie za pomocą LINQ[[99]](#footnote-99) następuje pobranie zalecenia z bazy o podanym identyfikatorze oraz sprawdzenie czy dane zalecenie na pewno należy do odpytującego pacjenta.

W sytuacji kiedy w bazie danych znalezione zostanie takie zalecenie zostaje ono mapowane na jego odpowiednik obiektu DTO, który zawiera odpowiednio przygotowane dane oraz ograniczone do wymaganego minimum. Następny fragment przedstawia w jaki sposób został skonfigurowany mapper. Czerwonym znacznikiem wskazano, która konfiguracja dotyczy wyżej wymienionego przykładu mapowania encji bazodanowej zalecenia na obiekt DTO. Tak przygotowana Mapa dziedzicząca po wbudowanej w bibliotekę AutoMapper klasie zostaje przypisana do kontenera DI oraz wykorzystana w korzystających serwisach. Poniższy przykład dodatkowo przedstawia sposób obsługi różnicy w strukturach obiektów zwracanych z bazy danych a obiektem DTO. W wykonania prawidłowej konwersji wykorzystano metody „ForMember” wbudowane w klasę Profile, które pozwalają bezpośrednio zadeklarować jakie wartości z obiektu źródłowego mają zostać przypisane do pola obiektu docelowego.

A screen shot of a computer program

Description automatically generated

Kwestia przypisania instancji tego serwisu do dependecy injaction została pominięta ze względu na ograniczenie ilości powtarzających się rysunków.

Podsumowując modele DTO są nieocenionym rozwiązaniem podczas budowy wrażliwych na dane web serwisów. Pozwalają one ograniczyć ilość zwracanych do użytkownika informacji do minimum przez co znacząco zwiększają bezpieczeństwo przechowywanych informacji. Kolejnym atutem jest możliwość intuicyjnego mapowania obiektów z wykorzystanie odrębnej biblioteki AutoMapper dzięki czemu web serwis może sprostać specyficznym strukturą danych wymaganych przez aplikację kliencka. Różne widoki wymagają danych przygotowanych w odpowiedniej formie, która może odbiegać od tej dostarczanej przez serwer bazodanowych. Korzystając z modeli DTO, możemy przygotować te dane bez konieczności wykonywania nieczytelnych i mało wydajnych konwersji w kodzie aplikacji.

## 3.4. Implementacja warstwy logiki biznesowej (backend)

W tym rozdziale został opisany proces tworzenia kluczowej części aplikacji, która jest odpowiedzialna za przetwarzanie danych i realizacje wszystkich założeń biznesowych. W tej sekcji zostały omówione reguły integracji backendu z bazą danych, sposoby komunikacji z aplikacją kliencką z wykorzystaniem interfejsów sieciowych API, omówione szczegółowo zostały reguły walidacji otrzymywanych danych, sposoby obsługi błędów i inne najważniejsze zadania stawiane przed web serwisem. Rozdział przedstawia także techniki optymalizacji oraz sposoby zabezpieczenia aplikacji przed problemami.

BIBLIOGRAFIA

1. <https://solwit.com/blog/dlaczego-warto-wybrac-technologie-net-do-realizacji-projektow-informatycznych/>
2. <https://1stplace.pl/blog/framework-co-to-entity-framework-net-framework-i-inne-rozwiazania/>
3. <https://asaricrm.com/slownik-crm/asp-net/>
4. KSIAZKA MARCIN LIS
5. <https://nofluffjobs.com/pl/log/wiedza-it/wszystko-co-powinienes-wiedziec-o-net-core/>
6. <https://cezarywalenciuk.pl/blog/programing/automapper-z-aspnet-core>
7. <https://justjoin.it/blog/solid-dobre-praktyki>
8. <https://tech.wp.pl/html5-co-to-jest-wyjasniamy-w-prostych-slowach,6548601655617153a>
9. https://mindboxgroup.com/pl/czym-jest-dto-data-transfer-object/

1. https://ks.pl/slownik/co-to-jest-ui [↑](#footnote-ref-1)
2. https://www.dbc.wroc.pl/Content/39276/Ratalewska\_Rozwoj\_Rynku\_Aplikacji\_Mobilnych\_2017.pdf [↑](#footnote-ref-2)
3. https://research.com/software/mobile-vs-desktop-usage#4 [↑](#footnote-ref-3)
4. https://poradnikprzedsiebiorcy.pl/-roznice-miedzy-http-a-https-i-ich-wplyw-na-pozycje-strony [↑](#footnote-ref-4)
5. https://www.google.com/search?q=technologia+backup+3-2-1&rlz=1C1GCEJ\_enPL1061PL1061&oq=technologia+backup+3-2-1&gs\_lcrp=EgZjaHJvbWUyBggAEEUYOTIICAEQABgWGB4yCAgCEAAYFhgeMggIAxAAGBYYHjIICAQQABgWGB4yCAgFEAAYFhgeMggIBhAAGBYYHjIICAcQABgWGB4yCAgIEAAYFhgeMggICRAAGBYYHtIBCDUzNTBqMGo0qAIAsAIA&sourceid=chrome&ie=UTF-8 [↑](#footnote-ref-5)
6. https://www.ibm.com/docs/pl/was/9.0.5?topic=overview-three-tier-architectures [↑](#footnote-ref-6)
7. https://learn.microsoft.com/en-us/dotnet/csharp/ [↑](#footnote-ref-7)
8. https://dotnet.microsoft.com/en-us/ [↑](#footnote-ref-8)
9. https://learn.microsoft.com/en-us/ef/ [↑](#footnote-ref-9)
10. https://dotnet.microsoft.com/en-us/apps/aspnet [↑](#footnote-ref-10)
11. https://automapper.org/ [↑](#footnote-ref-11)
12. https://fluentvalidation.net/ [↑](#footnote-ref-12)
13. https://nlog-project.org/ [↑](#footnote-ref-13)
14. https://sandrino.dev/blog/aspnet-core-5-jwt-authorization [↑](#footnote-ref-14)
15. https://github.com/domaindrivendev/Swashbuckle.AspNetCore [↑](#footnote-ref-15)
16. https://boringowl.io/blog/zwiekszenie-wydajnosci-testowania-kodu-za-pomoca-xunit [↑](#footnote-ref-16)
17. https://fluentassertions.com/ [↑](#footnote-ref-17)
18. https://medium.com/@ojbrot/everything-you-need-to-know-about-html5-and-css3-e83c80b5bb8b [↑](#footnote-ref-18)
19. https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First\_steps/What\_is\_JavaScript [↑](#footnote-ref-19)
20. https://redux.js.org/ [↑](#footnote-ref-20)
21. https://reactrouter.com/en/main [↑](#footnote-ref-21)
22. https://tanstack.com/query/latest [↑](#footnote-ref-22)
23. <https://nodejs.org/en> [↑](#footnote-ref-23)
24. https://www.techopedia.com/definition/24290/intermediate-language-il-net [↑](#footnote-ref-24)
25. https://mindboxgroup.com/pl/co-to-jest-orm-oraz-czym-sie-rozni-od-sql-object-relational-mapping/ [↑](#footnote-ref-25)
26. https://www.codecademy.com/article/what-is-crud [↑](#footnote-ref-26)
27. https://learn.microsoft.com/pl-pl/ef/core/managing-schemas/migrations/?tabs=dotnet-core-cli [↑](#footnote-ref-27)
28. https://learn.microsoft.com/en-us/ef/core/what-is-new/ef-core-8.0/breaking-changes [↑](#footnote-ref-28)
29. https://learn.microsoft.com/pl-pl/aspnet/web-forms/what-is-web-forms [↑](#footnote-ref-29)
30. https://pl.wikipedia.org/wiki/Model-View-Controller [↑](#footnote-ref-30)
31. https://www.sote.pl/slownik-ecommerce/webapi [↑](#footnote-ref-31)
32. https://learn.microsoft.com/en-us/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio [↑](#footnote-ref-32)
33. https://www.rlogical.com/blog/benefit-and-drawbacks-of-asp-net-core-razor-pages/ [↑](#footnote-ref-33)
34. https://clockworkjava.pl/2020/12/obiekt-domenowy-dto-dao/ [↑](#footnote-ref-34)
35. https://justjoin.it/blog/solid-dobre-praktyki [↑](#footnote-ref-35)
36. https://pl.wikipedia.org/wiki/HTTP\_cookie [↑](#footnote-ref-36)
37. https://www.techtarget.com/searchapparchitecture/definition/API-endpoint [↑](#footnote-ref-37)
38. https://blog.theprotocol.it/slownik-it/assembly [↑](#footnote-ref-38)
39. https://kamee.pl/blog/122.wdrozenie-aplikacji-webowej [↑](#footnote-ref-39)
40. https://www.netia.pl/pl/srednie-i-duze-firmy/youtro-strefa-wiedzy/co-to-jest-siec-lan [↑](#footnote-ref-40)
41. https://www.expressvpn.com/pl/what-is-vpn [↑](#footnote-ref-41)
42. https://jerzywickowski.pl/flux/co-to-jest-flux/ [↑](#footnote-ref-42)
43. https://www.cloudflare.com/learning/serverless/glossary/client-side-vs-server-side/ [↑](#footnote-ref-43)
44. https://pl.wikipedia.org/wiki/Uniform\_Resource\_Locator [↑](#footnote-ref-44)
45. https://developer.mozilla.org/en-US/docs/Glossary/SPA [↑](#footnote-ref-45)
46. https://dev.to/mjubair/the-url-pathway-4e7j [↑](#footnote-ref-46)
47. https://reactrouter.com/en/main/route/loader [↑](#footnote-ref-47)
48. https://reactrouter.com/en/main/route/action [↑](#footnote-ref-48)
49. https://mediamarkt.pl/pl/content/technologie/komputery/pamiec-cache-co-to-jest-jak-ja-wyczyscic [↑](#footnote-ref-49)
50. https://pl.legacy.reactjs.org/docs/hooks-overview.html [↑](#footnote-ref-50)
51. https://support.microsoft.com/pl-pl/office/co-to-s%C4%85-protoko%C5%82y-imap-i-pop-ca2c5799-49f9-4079-aefe-ddca85d5b1c9 [↑](#footnote-ref-51)
52. https://pl.wikipedia.org/wiki/POP3 [↑](#footnote-ref-52)
53. https://pl.wikipedia.org/wiki/Simple\_Mail\_Transfer\_Protocol [↑](#footnote-ref-53)
54. https://en.wikipedia.org/wiki/Request\_for\_Comments [↑](#footnote-ref-54)
55. https://pl.wikipedia.org/wiki/Test\_jednostkowy [↑](#footnote-ref-55)
56. https://agile247.pl/podejscie-iteracyjne-oraz-przyrostowe/ [↑](#footnote-ref-56)
57. https://www.redhat.com/en/topics/devops/what-cicd-pipeline [↑](#footnote-ref-57)
58. https://www.atlassian.com/continuous-delivery/continuous-integration [↑](#footnote-ref-58)
59. https://www.atlassian.com/pl/continuous-delivery [↑](#footnote-ref-59)
60. https://zeet.co/blog/deploy-to-production [↑](#footnote-ref-60)
61. https://pl.wikipedia.org/wiki/Linux [↑](#footnote-ref-61)
62. https://pl.wikipedia.org/wiki/Linus\_Torvalds [↑](#footnote-ref-62)
63. https://pl.wikipedia.org/wiki/System\_plik%C3%B3w [↑](#footnote-ref-63)
64. https://blog.qnap.com/pl/snapshot-different-backup-pl/ [↑](#footnote-ref-64)
65. https://pl.wikipedia.org/wiki/Cross-site\_scripting [↑](#footnote-ref-65)
66. https://opencart.com.pl/co-to-jest-clickjacking [↑](#footnote-ref-66)
67. https://www.cloudflare.com/learning/access-management/phishing-attack/ [↑](#footnote-ref-67)
68. https://pl.wikipedia.org/wiki/Cross-site\_scripting [↑](#footnote-ref-68)
69. https://blog.trustisto.com/pl/przyklady-komunikatow-pop-up/#Poznaj\_Trust\_Overlay [↑](#footnote-ref-69)
70. <https://react.dev/learn/your-first-component> [↑](#footnote-ref-70)
71. https://legacy.reactjs.org/docs/faq-state.html [↑](#footnote-ref-71)
72. https://www.geeksforgeeks.org/what-is-prop-drilling-and-how-to-avoid-it/ [↑](#footnote-ref-72)
73. https://legacy.reactjs.org/docs/context.html [↑](#footnote-ref-73)
74. https://redux.js.org/api/store [↑](#footnote-ref-74)
75. [↑](#footnote-ref-75)
76. [↑](#footnote-ref-76)
77. [↑](#footnote-ref-77)
78. [↑](#footnote-ref-78)
79. [↑](#footnote-ref-79)
80. [↑](#footnote-ref-80)
81. [↑](#footnote-ref-81)
82. [↑](#footnote-ref-82)
83. Pojęcie zostało wytłumaczone w rozdziale 3.1.2 [↑](#footnote-ref-83)
84. [↑](#footnote-ref-84)
85. [↑](#footnote-ref-85)
86. [↑](#footnote-ref-86)
87. [↑](#footnote-ref-87)
88. [↑](#footnote-ref-88)
89. [↑](#footnote-ref-89)
90. [↑](#footnote-ref-90)
91. [↑](#footnote-ref-91)
92. [↑](#footnote-ref-92)
93. [↑](#footnote-ref-93)
94. [↑](#footnote-ref-94)
95. [↑](#footnote-ref-95)
96. [↑](#footnote-ref-96)
97. [↑](#footnote-ref-97)
98. [↑](#footnote-ref-98)
99. [↑](#footnote-ref-99)