**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 2025

Table of Contents

[WSTĘP 5](#_Toc181864243)

[1. Analiza biznesowa – założenia projektowe aplikacji oraz problemy do rozwiązania 6](#_Toc181864244)

[1.1 Funkcjonalność – w jakim zakresie projekt powinien wspierać użytkownika. 6](#_Toc181864245)

[1.2 Dostępność –Problem różnicy wieku, urządzeń mobilnych oraz niepełnosprawności 6](#_Toc181864246)

[1.3 Bezpieczeństwo – jak przetwarzać oraz jak je przechowywać dane w bezpieczny sposób. 7](#_Toc181864247)

[Podsumowanie 8](#_Toc181864248)

[2. Omówienie wykorzystanych technologii 8](#_Toc181864249)

[2.1 Ogólny opis wykorzystanych technologii w projekcie. 8](#_Toc181864250)

[2.2 .NET – Platforma programistyczna od Microsoft 10](#_Toc181864251)

[2.3 C# - Główny język programowania wykorzystywany w technologii .NET 11](#_Toc181864252)

[2.4 EntityFramework – zaawansowane rozwiązanie ORM dla programistów .NET 12](#_Toc181864253)

[2.5 ASP .NET CORE – Platforma przeznaczona do budowy aplikacji internetowych 13](#_Toc181864254)

[2.6 AutoMapper – Mapowanie obiektów między wersjami aplikacji 13](#_Toc181864255)

[2.7. FluentValidator – walidacja danych .NET 13](#_Toc181864256)

[2.8. NLog – narzedzie do logowania danych w aplikacji 14](#_Toc181864257)

[2.9. JWT BEARER – narzędzie do uwierzytelniania użytkownika 14](#_Toc181864258)

[2.10. Swashbuckle/swawgger – narzędzie do automatycznego tworzenia dokumentacji API 15](#_Toc181864259)

[2.11. HTML 5 / CSS3 – Fundamenty warstwy prezentacji każdej aplikacji webowej. 15](#_Toc181864260)

[2.12. REACT – podstawowe narzędzie podczas tworzenia nowoczesnych interfejsów graficznych. 15](#_Toc181864261)

[2.13. VITE – świeży powiew przy tworzeniu aplikacji front-end 16](#_Toc181864262)

[2.14. React Redux – biblioteka do zarządzania stanem aplikacji 16](#_Toc181864263)

[2.15. React Router – technologia tworzenia routingu w aplikacjach react 17](#_Toc181864264)

[2.16. Tanstack Query – biblioteka wspierająca komunikację serwer-client 18](#_Toc181864265)

[2.17 FramerMotion – biblioteka do tworzenia animacji 19](#_Toc181864266)

[2.18. MailKit – biblioteka do obsługi protokołów poczty elektronicznej 19](#_Toc181864267)

[2.19. XUnit – testowanie aplikacji z rodziny .NET 19](#_Toc181864268)

[2.20. Git / GitHub – System kontroli wersji 20](#_Toc181864269)

[3. Implementacja aplikacji – budowa aplikacji w architekturze trójwarstwowej 21](#_Toc181864270)

[3.1 Implementacja warstwy prezentacji – aplikacja kliencka 21](#_Toc181864271)

[3.1.1 Omówienie wymogów wizualnych aplikacji 22](#_Toc181864272)

[3.1.2 Komunikacja komponentów w oparciu o technologie REDUX 23](#_Toc181864273)

[3.1.3 Omówienie routingu aplikacji z wykorzystaniem bibloteki React Router 27](#_Toc181864274)

[3.1.4 Zaimplemetowanie własnych Hook’s 29](#_Toc181864275)

[3.1.4 Komunikacja z backendem z wykorzystaniem React TanstackQuery 30](#_Toc181864276)

[3.1.5 Walidacja w warstwie prezentacji 31](#_Toc181864277)

[3.1.6 Zapewnienie nowoczesnego designu dzięki FramerMotion 32](#_Toc181864278)

[3.1.7 Zawansowany serwer aplikacji Vite 33](#_Toc181864279)

[3.2 Implementacja warstwy dostępu do danych 34](#_Toc181864280)

[3.2.1 Omówienie wymogów przechowywania danych 34](#_Toc181864281)

[3.2.2. Code First – podejście definiowania modeli baz danych 34](#_Toc181864282)

[3.2.2. Implementacja kontekstu baz danych w projekcie 37](#_Toc181864283)

[3.3.3 Wykorzystane Modele (DO NAPISANIA) 39](#_Toc181864284)

[3.3.4. Wykorzystanie wzorca projektowego „Repository” 39](#_Toc181864285)

[3.3.5. Data Transfer Object 42](#_Toc181864286)

[3.4. Implementacja warstwy logiki biznesowej (backend) 44](#_Toc181864287)

[3.4.1. Omówienie struktury aplikacji. 44](#_Toc181864288)

[3.4.2. Podział aplikacji na Kontrolery, Serwisy,i modele 44](#_Toc181864289)

[3.4.3. Zaimplementowane walidowanie danych 47](#_Toc181864290)

[3.4.4.Wykorzystanie systemu dependency injection 48](#_Toc181864291)

[3.4.5. Wykorzystanie własnych middleware w pipelinie 48](#_Toc181864292)

[3.4.6. Obsługa protokołów poczty elektronicznej z wykorzystaniem MailKit 50](#_Toc181864293)

[3.4.7) Komunikacja z bazą danych 52](#_Toc181864294)

[3.5. Omówienie autentykacji oraz autoryzacji użytkownika 52](#_Toc181864295)

[3.5.1. Implementacja tokenu JWTBarer 53](#_Toc181864296)

[3.5.2. Generowanie tokenu 54](#_Toc181864297)

[3.5.3. Wymiana tokenu pomiędzy warstwami aplikacji 55](#_Toc181864298)

[Zakończenie 57](#_Toc181864299)

[Bibliografia 58](#_Toc181864300)

[Spis rysunków 59](#_Toc181864301)

[Streszczenie 60](#_Toc181864302)

Skonczylem czytac strona 31

# 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.  
Szczególnie problem ten ukazał niedawny kataklizm powodziowy na obszarze całego dolnego śląska w wyniku którego zalane i zniszczone zostały papierowe archiwa w wielu przychodniach co spowodowało nie odwracalne straty informacji.

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 temat niniejszego projektu, 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 o zaplanowanej wizycie co pozwoli zautomatyzować procesy planowania oraz kontaktu z pacjentem. Unikniemy również wielu błędów komunikacyjnych takich jak brak poinformowania 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 w przyszłości również 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. 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.

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

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

### 1.1 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, przechowywania 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, przechowywania oraz modyfikacji dokumentacji medycznej zawierającej informacje o postępach w 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 sprawnie utworzyć zalecenia dla swojej pacjentki, które zostaną automatycznie sformatowane przez system do czytelnej i eleganckiej 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). Wdrożenie wysyłania zaleceń poprzez komunikacje email będzie dobrym kolejnym krokiem w rozwoju aplikacji.

### 1.1 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 ponurym i nudnym designem. 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 intuicyjny 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, kolor czcionki oraz tło dobrane zostało aby ułatwić pracę osobą mającym problem ze wzrokiem. Rozmiar 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 czy 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

**Rysunek 1 Wydajność platformy .NET**

**Źródło:** https://research.com/software/mobile-vs-desktop-usage#4

### 1.3 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 jednak, że do celów obsługi protokołów poczty elektronicznej serwer na, którym uruchomiona zostanie aplikacja wymaga stałego połączenia z siecią Ethernet co naraża go na potencjalne ataki. Z tego powodu koniecznym było staranne zaprojektowanie strategii zabezpieczenia i ochrony składowanych 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[[3]](#footnote-3) (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. (należy pozostawić możliwość prostej implementacji mechanizmów szyfrowania w projekcie)
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)[[4]](#footnote-4). Ta funkcjonalność została jednak odroczona jako kolejna iteracja zwinnego wdrażania funkcjonalności w projekcie.
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, 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 możliwości na udostępnienie projektu w sieci z wykorzystaniem sieci VPN.

# 2. Omówienie wykorzystanych technologii

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[[5]](#footnote-5).** Każda z tych warstw pełni konkretne 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 wykorzystano 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 aplikacji. Ostatnią warstwą 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 sprostać wszystkim powierzonym jej zadaniom wykorzystano do jej utworzenia następujące technologie:

* C#[[6]](#footnote-6)
* .NET[[7]](#footnote-7)
* Entity Framework[[8]](#footnote-8)
* ASP .NET CORE – REST API [[9]](#footnote-9)
* AutoMapper[[10]](#footnote-10)
* FluentValidator[[11]](#footnote-11)
* Nlog[[12]](#footnote-12)
* JwtBearer[[13]](#footnote-13)
* Swashbuckle/swawgger[[14]](#footnote-14)
* xUnit[[15]](#footnote-15)
* FluentAssertions[[16]](#footnote-16)
* HTML 5 / CSS3[[17]](#footnote-17)
* JavaScript[[18]](#footnote-18)
* React Redux[[19]](#footnote-19)
* React Router[[20]](#footnote-20)
* React @Tanstack Query[[21]](#footnote-21)
* Node[[22]](#footnote-22)

### .NET – Platforma programistyczna od Microsoft

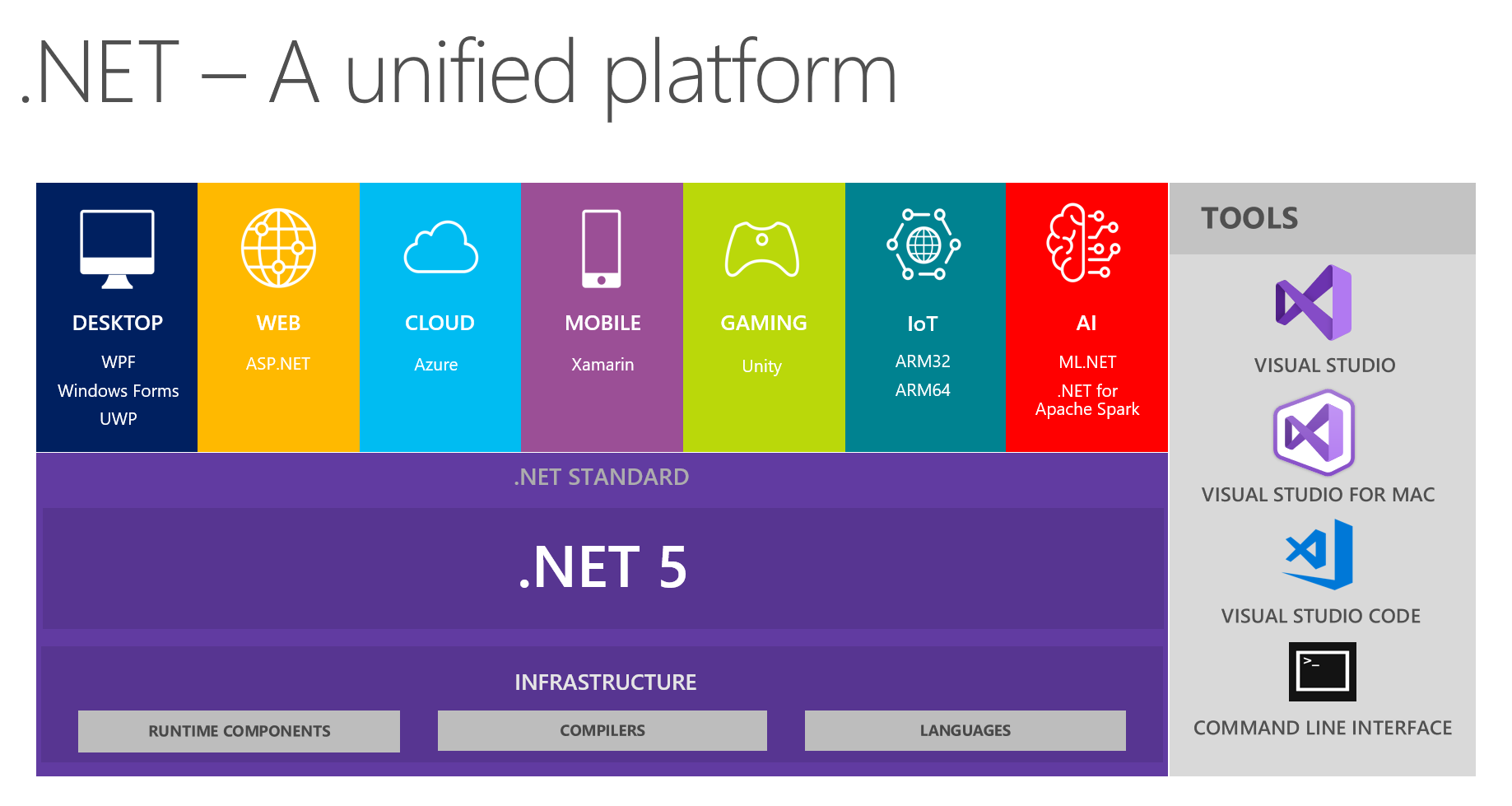
.NET jest to wszechstronna platforma programistyczna stworzona przez firmę Microsoft. Jest częstym wyborem podczas projektowania oraz tworzenia rozwiązań takich jak: aplikacje webowe, aplikacje desktopowe, mobilne oraz gry wideo. Platforma Net oferuje wysoce rozbudowane środowisko uruchomieniowe, wiele 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)[[23]](#footnote-23).

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



**Rysunek 2 Zastosowanie platformy .NET**

**Źródło:** https://auth0.com/blog/what-is-dotnet-platform-overview/

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

C# jest językiem wywodzącym się z rodziny C oraz C++. Mówi się nawet, że jego nazwa wskazuje na kolejne rozwinięcie tych technologii. Płotek (eng. hasztag) ma przypominać cztery plusy ustawione w dwóch rzędach i dwóch kolumnach. Technologia Microsoftu czerpie również wiele elementów z języka Java jak na przykład mechanizmy automatycznego odzyskiwania pamięci. C# jest językiem obiektowym ścisłe powiązanym z środowiskiem uruchomieniowym .NET oraz aktualną jego wersją .NET Core. Posiada wbudowane najważniejsze mechanizmy takie jak obsługa wyjątków czy programowanie asynchroniczne. [[24]](#footnote-24) C# podczas kompilacji jest zamieniany przez środowisko uruchomieniowe .NET Runtime na język CIL - Common Intermediate Language. CIL jest językiem pośrednim między kodem C# a kodem maszynowym. Środowisko .NET wykorzystuje CIL w celu zapewnienia wieloplatformowości aplikacji. Standardowa kompilacja generuje kod maszynowy gotowy do uruchomienia. Niestety kod nie jest wolny od ograniczeń – „Otóż, kod maszynowy jest dość ciężki w utrzymaniu, różni się w zależności od architektury komputera, a nawet posiadanego procesora.“[[25]](#footnote-25). W celu uniknięcia takiego problemu środowisko uruchomieniowe .NET w pierwszej kolejnośći konwertuje kod wytworzony w C# na wspomniany język pośredni, który będzie skompilowany przez odpowiednio skofigurowane środowiska uruchomieniowe dostosowane do konkretnego systemu na kod maszynowy zgodny z wykorzystywanym systemem operacyjnym oraz procesorem.

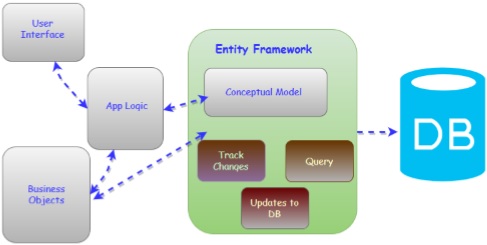
### 2.4 EntityFramework – zaawansowane rozwiązanie ORM dla programistów .NET

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) [[26]](#footnote-26).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)[[27]](#footnote-27) 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 bazy danych:

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[[28]](#footnote-28)”
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)[[29]](#footnote-29), który jest wydajniejszym i jeszcze bardziej zoptymalizowanym rozwiązaniem.



**Rysunek 3 Schemat działania technologii EntityFramework**

**Źródło:** https://www.tutorialspoint.com/

### 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[[30]](#footnote-30)
2. ASP.NET MVC (MODEL-VIEW-CONTROLLER)[[31]](#footnote-31)
3. ASP.NET WEB API[[32]](#footnote-32)
4. ASP.NET MINIMAL API[[33]](#footnote-33)
5. ASP.NET RAZOR PAGES[[34]](#footnote-34)

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) [[35]](#footnote-35) 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[[36]](#footnote-36). Dodatkowo Fluent Validator wspiera również walidacje warunkową. Dzięki czemu możemy tworzyć reguły, które będą obowiązywały dopiero po spełnieniu jakiegoś warunku.

### 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 autentykacji. 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 są bardzo wrażliwe, 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[[37]](#footnote-37) 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 kodem tzw. „secret” 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 przedstawiono schemat działania takiej komunikacji z wykorzystaniem JWT BEARER.

A screenshot of a computer

Description automatically generated

**Rysunek 4 Schemat działania tokenu JWT**

**Źródło:** https://netstorage.ringcentral.com

### 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[[38]](#footnote-38). 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 wspieranym oraz rozwijanym projektem. Działanie tego rozwiązania jest bardzo proste. Po rejestracji swaggera w assembly [[39]](#footnote-39) 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 o znaczniki, które umożliwiają zapisanie szablonu wizualnej części aplikacji. 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 na potrzeby Facebooka. Pozwala ona budowanie dużych aplikacji internetowych z wykorzystaniem komponentów. Technologia ta znacznie usprawnia pisanie kodu ponieważ wykorzystuje paradygmaty programowania deklaratywnego. To znaczy, że tworzony 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, Reacta uznano 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 bibliotek jak np. Vue.js. Korzysta on z natywnego serwera modułów ES dzięki czemu jest jednym z najszybszych dostępnych aktualnie narzędzi przeznaczonych do tworzenia projektów frontendowych. Zasada działania 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.

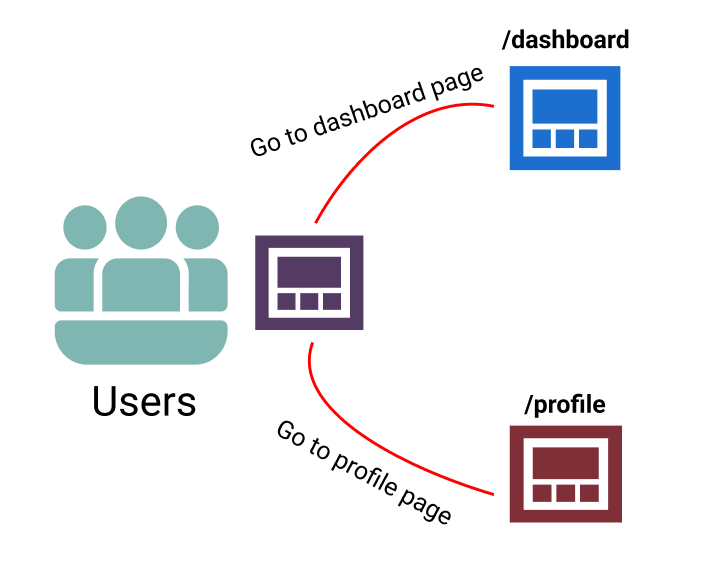
Vite dostarcza wiele pomocniczych funkcjonalności. Jedna z nich jest możliwość zbudowania aplikacji po zakończeniu prac deweloperskich. W momencie kiedy prace nad aplikacją dobiegną końca, i aplikacja jest gotowa do wdrożenia na serwery produkcyjne vite przekonwertuje kod (np. reacta) na czysty kod JavaScript, który będzie zrozumiały dla przeglądarki. Dzięki temu produkt zostanie odpowiednio przygotowany, skompresowany oraz gotowy do wdrożenia[[40]](#footnote-40). Vite umożliwia dodatkowo uruchomienie aplikacji w sieci lokalnej[[41]](#footnote-41), dzięki czemu możemy udostępniać aplikacje bezpośrednio z lokalnego komputera a zdalny dostęp do niej zapewniony zostaje dzięki technologii VPN[[42]](#footnote-42).

### 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.[[43]](#footnote-43). 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 abstrakcyjne. 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[[44]](#footnote-44). Podstawowym celem tej technologii jest zapewnienie wyszukiwania za pomocą asynchronicznych adresów URL[[45]](#footnote-45).Sensem React Router jest projektowanie i tworzenie aplikacji w architekturze single-page [[46]](#footnote-46) z zachowaniem standardowej struktury i funkcjonalności stron internetowych. Zasada jego działania jest stosunkowo prosta. Użytkownik wprowadza adres URL w przeglądarce internetowej, Router dopasowuje jego zapytanie URL do konkretnej „ścieżki” (en. Pathways[[47]](#footnote-47)), 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[[48]](#footnote-48)” oraz „actions[[49]](#footnote-49)” dostarczonych przez tą technologie.



**Rysunek 5 Schemat działania routingu w aplikacjach SPA**

**Źródło:** https://medium.com

### 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[[50]](#footnote-50)” 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” [[51]](#footnote-51) 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[[52]](#footnote-52)
* POP3[[53]](#footnote-53)
* SMTP[[54]](#footnote-54)

MailKit zapewnia implementacje tych protokołów z zachowaniem wymogów RFC[[55]](#footnote-55). 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[[56]](#footnote-56)
2. Testy integracyjne[[57]](#footnote-57)

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 [[58]](#footnote-58)przy korzystaniu z rozwiązań Continious integration [[59]](#footnote-59)oraz continious delivery[[60]](#footnote-60).

Podsumowując korzystanie z rozwiązania Microsoft jest nieocenione podczas tworzenia nowoczesnych aplikacji na platformie .NET ponieważ pozwala ona szybko i efektywnie zagwarantować niezawodność aplikacji na wszystkich etapach jej tworzenia. Dzięki intuicyjnemu sposobowi pracy podczas zarówno kompilacji jak i wysłaniu ( en. deploy[[61]](#footnote-61)) projektu na serwer produkcyjny czy repozytorium zdalne. Aplikacja LucyCover została przygotowana do implementacji i wdrożenia tego rozwiązania jednak zostało to przeniesione jako kolejna iteracja rozwoju projektu.

### 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 programistów 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 [[62]](#footnote-62)– Linus Torvalds[[63]](#footnote-63) 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 [[64]](#footnote-64) w systemach operacyjnych możemy nazwać commit „migawką[[65]](#footnote-65)” (eng. Snapsot). Commit jest konkretną wersją projektu, 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 przez 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 tworzonego projektu.

# 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 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 intuicyjnej i bezproblemowej analizy 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) [[66]](#footnote-66), Clickjacking[[67]](#footnote-67) oraz atakami typu phishing[[68]](#footnote-68)

### 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 wyświetlały jak najbardziej intuicyjne i czytelne UI. 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.

A screenshot of a computer

Description automatically generated

**Rysunek 6 Prezentacja graficznej części projektu**

**Źródło:** Opracowanie własne – zrzut ekranu z aplikacji LucyCover

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. 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 [[69]](#footnote-69) oraz overlays[[70]](#footnote-70). Dzięki temu na ekranie przysłonięte zostają aktualnie nie potrzebne funkcjonalności a wyświetlone jedynie potrzebne opcje. Pozwala nam to skupić uwagę użytkownika na aktualnie istotnych widokach. W ten sposób zmniejszamy ryzyko popełnienia błędu przez pracownika podczas wprowadzania danych do systemów. W dalszej części przedstawione zostało w jaki sposób wspomniane „popup” zwraca uwagę użytkownika na konkretną aktualnie istotną treść.

A computer screen shot of a computer

Description automatically generated

**Rysunek 7 Zastosowanie okien typu „popup” do zwiększenia przejrzystości aplikacji**

***Źródło:*** *Opracowanie własne – zrzut ekranu z aplikacji LucyCover*

### 3.1.2 Komunikacja komponentów w oparciu o technologie REDUX

Aplikacje klienckie tworzone w technolog React opierają się o tzw. komponenty (eng. componets[[71]](#footnote-71)). 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[[72]](#footnote-72)). Pojęcie to oznacza, że każdy komponent przechowuje dane, które mogą być zmieniane w czasie i wpływać 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 ponownie wykonując 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[[73]](#footnote-73)” lub stosowanie wbudowanego mechanizmu Context[[74]](#footnote-74).

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[[75]](#footnote-75) 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 splicy. 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

**Rysunek 8 Implementacja „store” biblioteki Redux**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

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

**Rysunek 9 Implementacja „slice” biblioteki Redux**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

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 takich klas będzie się stale zwiększać.

A screen shot of a computer program

Description automatically generated

**Rysunek 10 Rejestracja „store” w komponencie „Provider” biblioteki Redux**

**Źródło: Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code**

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 slicy.

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

Routing w aplikacjach typu SPA [[76]](#footnote-76) (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 [[77]](#footnote-77) (czyt. nawigacji) . Pozwala w względnie prosty sposób skonfigurować trasy (eng. routes) oraz zarządzać nimi przy pomocy ścieżek URL[[78]](#footnote-78). Dodatkowo React router udostępnia szereg różnych metod oraz wbudowanych Hooks [[79]](#footnote-79) 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 loadera) lub odpowiednie przygotowania danych do wysłania po przykładowym 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*** 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) udostępniającą 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

**Rysunek 11 Implementacja „routera” biblioteki React Router**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

A computer screen with white text

Description automatically generatedNastępnie 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.

**Rysunek 12 Przypisanie „routera” do komponentu „provider” biblioteki React Router**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

### 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.

A screen shot of a computer program

Description automatically generated 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). Jest to funkcja wykorzystywana w wielu miejscach aplikacji pozwalająca na pobrania gotowych wartości z obiektów zapisanych w slice [[82]](#footnote-82), które następnie zostają wysłane na serwer.

**Rysunek 13 Implementacja „useFormData”**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

Powyższy przykład przedstawia sposób implementacji useFormData. Jak widać na załączonym fragmencie kodu hooks jako wyniki jego wywołania zwraca funkcje, która przekonwertuje dane z postaci obiektu przechowywanego w slice na zestaw danych gotowych do wysłania. W pierwszej kolejności następuje sprawdzenie czy przekazany parametr jest obiektem o takiej samej strukturze jak te przechowywane w store. Następnie wykonywana jest pętla sprawdzająca wszystkie pola obiektu i pobiera dane z właściwości value, które następnie zostają przypisane do obiektu „FormDataValues”. Inaczej mówiąc 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[[83]](#footnote-83)) oraz chmur obliczeniowych uruchamiający aplikacje backendowe przetwarzające dane uzyskano 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 dwiema aplikacjami. Wbudowane mechanizmy dostarczane przez JavaScript są dość skomplikowane w implementacji oraz mogą pogarszać czytelność kodu. 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 backendową. Wszystkie funkcje zawierające instrukcje połączeniowe (fetch) [[84]](#footnote-84) 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 zapytania do serwera automatycznie są wykonywane 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. Jako useMutation oraz useQuery zwraca zmienne zawierające informacje takie jak isPending, isSuccess, isError, które umożliwiają wyświetlenie konkretnych widoków w zależności od stanu procesowania zapytania. Dla przykłady podczas przetwarzania zapytania isPending zawiera wartość true, którą możemy wykorzystać aby wyświetlić tzw. loading spiner.

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 pochodzi z pliku api/http.js przchowuje on instrukcje 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.



**Rysunek 14 Przykład pobierania danych z serwera za pośrednictwem Tanstack Query**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

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 informatycznych. Mechanizmy walidacji zaimplementowane w warstwie prezentacji stanowią podstawową ochronne spójności i prawidłowości przesyłanych 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 walidacja 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 będących typem./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 wykorzystany podczas walidacji wartości wprowadzonych przez użytkownika

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[[85]](#footnote-85) 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 [[86]](#footnote-86)wydając do użytku bibliotekę JQUERY[[87]](#footnote-87), 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[[88]](#footnote-88).

Aby móc rozwijać aplikacje w technologii react potrzebujemy specjalnego narzędzia uruchomieniowego. Przykładem takiego narzędzia jest program vite, który szybko stał 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 systemie. 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)[[89]](#footnote-89) oraz HMR (Hot Module Replacement)[[90]](#footnote-90) dzięki czemu zmiany wprowadzane w kodzie widzimy natychmiast bez konieczności ciągłego odświeżania strony po wprowadzeniu zmian.

Po zakończeniu fazy deweloperskiej vite dostarcza nam funkcje „budowania” aplikacji i przystosowania jej do wdrożenia na serwer zdalny. Dzięki komendzie „build” vite dostarcza nam gotową paczkę projektu z przekonwertowanym kodem JSX na czysty kod javaScript, który jest mało czytelny i zrozumiały za to bardzo skompresowany i wydajny podczas działania.

W ramach podsumowania należy 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 zmaga się szeregiem wytycznych i norm jakie muszą zostać spełnione 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. Implementacja mechanizmów dostępu do danych powinna zachować możliwość prostej rozbudowy o implementacje mechanizmów szyfrujących.
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 osobna abstrakcją realizujący wszystkie założenia niezbędne do prawidłowej komunikacji z serwerem bazodanowym. Do obsługi centralizacji danych wykorzystano bazę danych Microsoft SQL współpracującą z językiem zapytań T-SQL będącym rozbudowaną wersją języka SQL. Na potrzeby jednolitej centralizacji wszystkich modeli występujących w programie powstała osobna biblioteka przechowująca wszystkie wykorzystywane modele w bazie danych oraz klasy typu DTO wykorzystywane w aplikacji podczas przesyłania danych pomiędzy systemem a użytkownikiem. Na potrzeby mapowania encji bazodanowych na modele DTO zaimplementowano technologię mapowania, 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 – MS Sql. Baza danych jest rozbudowywana na bieżąco podczas implementowania kolejnych usług aplikacji. Podejście to nazywa się CodeFirst. Zamysł tego podejścia polega na definiowaniu modeli, które powinny znajdować się w bazie danych. Modele te są aplikacyjnym odzwierciedleniem tablic przechoywanych 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 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 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[[91]](#footnote-91) (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[[92]](#footnote-92), 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.

A screen shot of a computer program

Description automatically generated 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 rysunek prezentuje zadeklarowany model „Recommendations”

**Rysunek 15 Definicja modelu „Recommendation”**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

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. Na podstawie obiektów tych klas, bibliotek Microsoftu jest w stanie wyselekcjonować, które modele aplikacji powinny mieć swoich odpowiedników w postaci tabel po stronie systemu bazodanowego.

A computer screen shot of a code

Description automatically generated

**Rysunek 16 Przypisanie modeli bazodanowych do klasy „DbConnection”**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

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

**Rysunek 17 Wygenerowana migracja przy użyciu pakietu EntityFramework Tools**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

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 metodę Down, która zawiera instrukcje cofające migracje. 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 informacji wymagających wykorzystania zaawansowanych aspektów przechowywania takich jak indeksy czy też 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 do nich sprawny dostęp.

A computer screen shot of a program

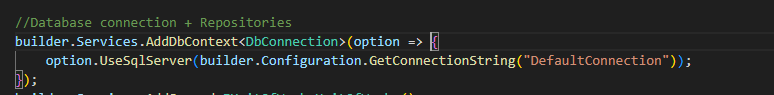
Description automatically generated Tak jak wspomniano wcześniej LucyCover korzysta z rozwiązania opartego o system ORM – EntityFrame worke core. U podstaw tej technologii jest odpowiednie sformułowanie sposobu dostępu do bazy danych. Do tego celu wykorzystuje się specjalnie przygotowaną klasę dbContext oraz specjalnie przygotowany i skonfigurowany connectionString. W ramach wyjaśnienia czym są po krótce wspomniane wyżej elementy, element DbContext jest klasą dostarczaną 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 dokumentacją LucyCover implementuje własną klasę DbConnection, która dziedziczy po klasie DbContext uzyskując dostęp do koniecznego konstruktora bazowego. DbConnection zaimplementowany w LucyCover pobiera argument typu DbContextOptions<>, który przechowuje odpowiednio skonfigurowaną procedurę łączenia się z bazą danych z wykorzystaniem pojęcia connectionString, które zostanie omówione w dalszej części rozdziału.

**Rysunek 18 Implementacja kontekstu baz danych w projekcie**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

Powyższy fragment kodu prezentuje klasę DbConnection utworzoną na potrzeby konfiguracji połączenia między aplikacją backendową a serwerem bazodanowym. 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 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 konfiguracji głównego buildera [[93]](#footnote-93) aplikacji ASP.Net Core. Poniższy fragment aplikacji przedstawia sposób przekazania nowej instancji klasy DbConnection do kontenera zależności aplikacji.



**Rysunek 19 Rejestracja „DbContext” w kontenerze zależności aplikacji**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

A screenshot of a computer

Description automatically generatedNie 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. Przechowuje on wszystkie niezbędne informacje jak host bazy danych, dane autentykacji jak login oraz hasło, port na którym występuje komunikacja oraz nazwa bazy danych.

**Rysunek 20 Konfiguracja aplikacji w pliku „appsettings.json”**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

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 aspektów aplikacji bez konieczności ponownego kompilowania kodu. 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. Tak skonfigurowany connectionString zostaje wykorzystany przez klasę DbContext do nawiązania połączenia.. 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 komunikacji z bazą danych oraz przypadki 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

W odróżnieniu od systemów typu NoSql, relacyjne bazy danych wymagają przechowywania danych w tabelach o jednolitej, spójnej strukturze. Oznacza to, że przechowując dane w konkretnej tabeli wszystkie rekordy muszą implementować te same kolumny i spełniać te same zasady. W przypadku korzystania z entity framework oraz podejściu „code first” tabele powstają w skutek mapowania modeli zdefiniowanych w kodzie aplikacji za pośrednictwem migracji. W związku z tym zdefiniowane w projekcie modele mają swoje odzwierciedlenie w systemie bazodanowym.

W projekcie LucyCover modele przechowywane są w odrębnej bibliotece zwanej „LucyCover\_Model”, które rozdzielono na trzy rodzaje:

* Modele bazodanowe – katalog „Database\_Entities przechowuje definicje modeli zmapowanych na tabele przechowywane w bazie danych. To właśnie na te modele ORM mapuje dane pobierane z bazy danych oraz w tej formie przesyłane są dane do bazy danych
* Modele DTO – katalog „DTO\_Modeles” przechowuje definicje modeli Data transfer object, które wykorzystywane są do ograniczania i dostosowywania pobranych z bazy danych modeli na potrzeby wyświetlanych przez warstwę prezentacji (aplikacje react) widoków.
* Model autentykacji – jest to specjalny odrębny model, który został wykorzystany na potrzeby płynnego udostępniania za pośrednictwem kontenera zależności informacji dotyczących konfiguracji algorytmów generowania tokenów autentykacji. Są to informacje na temat domeny wystawiającej oraz posługującej się tokenem, ostateczna data wygaśnięcia tokenu oraz specjalny klucz służący do zaszyfrowania tokenu. Wytłumaczenie zasad rozwiązania problemu autentykacji został przedstawiony w późniejszych rozdziałach.

### 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 znaczny wpływ 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 serwisami 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 jakie opcje udostępnia generyczne repozytorium. 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

**Rysunek 21 Implementacja interfejsu generycznego „IRepository”**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

A screenshot of a computer program

Description automatically generated

**Rysunek 22 Implementacja klasy „Repository"**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

A screenshot of a computer program

Description automatically generatedZe względu na ilość instrukcji wchodzących w skład implementacji 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 unikalna dla każdego modelu.

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 wszystkich konstruktorów serwisów korzystających z tego interfejsu dostępu do danych. Na rysunku przedstawiono strukturę unitOfWork oraz jego rejestracje w kontenerze.

**Rysunek 23 Przykład implementacji repozytorium**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

A computer screen shot of a program

Description automatically generated

**Rysunek 24 Implementacja kontenera repozytoriów – „UnitOFWork”**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio



**Rysunek 25 Rejestracja repozytoriów w kontenerze zależności**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

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 staje się zapewnienie wydajnej i płynnej komunikacji oraz zachowanie jak najlepszego poziomu zabezpieczenia danych. Nowoczesne web serwisy chronią przetrzymywane dane wykorzystując mechanizmy autoryzacji[[94]](#footnote-94), autentykacji[[95]](#footnote-95), wykorzystują możliwości zabezpieczeń CORSE oraz wielu innych niezależnych bibliotek wspomagających ich pracę. Niemniej jednak nawet jeżeli użytkownik końcowy faktycznie zostanie prawidłowo uwierzytelniony należy zachować odpowiednie formy bezpiecznego przekazywania informacji. Jednym z takich założeń jest ograniczanie ilości przekazywanych danych do absolutnie wymaganego minimum. To znaczy, że dane pobierane z baz danych powinny zostać ograniczone do niezbędnych informacji potrzebnych użytkownikowi. Aby odpowiednio wyselekcjonować konieczne informacje od danych przechowywanych w bazie danych wykorzystujemy modele DTO (Data Transfer Object). Są to obiekty przeznaczone do przesyłania informacji 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 dokumentacji medycznej. Każdy endpoint kontrolera ma przygotowany specjalna klasę DTO, która przechowuje tylko informacje potrzebne do prawidłowego wyświetlenia widoku przez aplikacje pracującą w warstwie prezentacji. 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 sposobu w jaki 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

**Rysunek 26 Pobranie zaleceń z bazy danych oraz mappowanie rezultatów na obiekt DTO**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

Jest to fragment serwisu odpowiedzialnego za obsługę zakładki „Zalecenia”. Serwis ten jest odpowiedzialny za prawidłowe pobieranie, dodawanie oraz edycję zaleceń pacjenta. 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[[96]](#footnote-96) następuje pobranie zalecenia z bazy o podanym identyfikatorze oraz sprawdzenie czy dane zalecenie istnieje w systemie.

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 klasą DTO. Do 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

**Rysunek 27 Konfiguracja mapowania biblioteki „AutoMapper”**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

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 web serwisów zorientowanych na bezpieczeństwo przetwarzanych informacji. Pozwalają one ograniczyć ilość zwracanych do użytkownika danych do minimum przez co znacząco zwiększają bezpieczeństwo informacji. Kolejnym atutem jest możliwość intuicyjnego mapowania obiektów z wykorzystanie odrębnej biblioteki AutoMapper dzięki, której 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.

### 3.4.1. Omówienie struktury aplikacji.

Do stworzenia warstwy logiki biznesowej dla systemu LucyCover wykorzystana została technologia dostarczona przez firmę Microsoft Asp .Net core. Pozwala ona na tworzenie wydajnych web serwisów możliwych do uruchomiania na wszystkich dostępnych platformach (Linux,Windows oraz MacOs). Warstwa ta przede wszystkim odpowiada za przetwarzanie dostarczanych oraz udostępnianych danych, kontroluje przepływ informacji pomiędzy bazą danych a aplikacja kliencką. Jest odpowiedzialna za autoryzację oraz autentykację użytkowników podczas prób dostępów do interfejsów API. Web serwis waliduje wszystkie otrzymywane informacje zapewniając ich spójność oraz minimalizując wprowadzanie błędnych informacji do systemu.

Aby zapewnić niezawodność działania, solidną jakość kodu, bezpieczeństwo oraz skalowalność web serwisy oparte o technologie ASP.NET Core tworzone są zgodnie z jasno określoną strukturą pozwalającą zachować pewny poziom modułowości oraz abstrakcji poszczególnych funkcjonalności aplikacji. Warstwa logiki biznesowej aplikacji LucyCover podzielona została na:

* Kontrolery
* Kontenery zależności
* Serwisy
* Wyjątki
* Middleware

Jest to kilka podstawowych struktur składających się na całość web serwisu. W kolejnych rozdziałach zostaną wytłumaczone ich odpowiedzialności, zasady działania i zależności między sobą.

### 3.4.2. Podział aplikacji na Kontrolery, Serwisy,i modele

W początkowych latach funkcjonowania web serwisów Microsoft dostarczał rozwiązanie ASP .NET obsługujące w głównej mierze wzorzec MVC [[97]](#footnote-97)(model-view-controller). Było to rozwiązanie, które z reguły sprowadzało się do tworzenia dużych aplikacji monolitycznych [[98]](#footnote-98), gdzie wszystkie warstwy były tworzone w ramach jednego projektu ASP .NET. W roku 2016 firma Microsoft wprowadziła następcę poprzedniej technologii webowej – ASP .NET CORE, który zmienił dotychczasowy trend MVC na projektowanie web serwisów w oparciu o architekturę REST Api.

Mimo wprowadzenia nowej technologii i wraz z nią szeregu nowych dodatkowych możliwości, Microsoft pozostał wierny dawnemu podziałowi aplikacji na modele oraz obsługujące je kontrolery. Jednakże warstwa widoku została oddelegowana do odrębnych technologii specjalizujących się w tworzeniu wysoce wydajnych interfejsów użytkownika.

Zgodnie z tym trendem wszystkie interfejsy dostępowe API zadeklarowano w specjalnie przygotowanych kontrolerach. Kontrolery aplikacji zaprojektowane zostały w taki sposób aby zachować odpowiedni poziom abstrakcji obsługując każdą funkcjonalność przez osobny konkretny kontroler. Kontrolery LucyCover odpowiedzialne są za odbieranie zapytań z aplikacji klienckiej a następnie oddelegowywanie ich do odpowiednich serwisów. W zależności od wyniku działania serwisu kontroler zwraca konkretne informacje takie jak kody odpowiedzi, dane, czy pliki cookie[[99]](#footnote-99).

Kolejna abstrakcją są serwisy, które wykonują cała logikę biznesową aplikacji. Każdy serwis zarejestrowany jest w kontrolerze dependency injaction i przygotowany do wykorzystania w konkretnych kontrolerach. To w tym miejscu dane są przetwarzane oraz przygotowywane do zwrócenia klientowi poprzez kontroler. Każdy kontroler korzysta z własnego serwisu, który implementuje wszystkie metody potrzebne do wykonania przekazanych zadań. Dzięki temu projekt zachowuje czytelność kodu i jest wierny wszystkim zasadą z grupy SOLID.

A computer screen with text on it

Description automatically generated Poniższy fragmenty przedstawiają przykład obsługi zapytania API dot. autentykacji użytkownika.

**Rysunek 28 Implementacja kontrolera do obsługi zapytań dot. autentykacji**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

W pierwszej kolejności zadeklarowany został kontroler działający pod adresem „api/auth”, który odpowiedzialny będzie za odbieranie zapytań od aplikacji klienckiej oraz odpowiadanie na próby autentykacji użytkownika. Na przykładzie zauważyć można definiowanie ścieżki dostępowej, oraz zadeklarowanie zmiennych przechowujących instancje wykorzystywanych serwisów.

Kolejny fragment kodu przedstawia implementacje endpoina LoginUser, który przyjmuje dane logowania od użytkownika w postaci json następnie taki obiekt zostaje przekazany do metody „GetJwtToken” serwisu „IAuthenticationService”. W metodzie tej następuje próba odnalezienia użytkownika w systemie oraz autentykacji. W zależności od wyniku pracy, w przypadku udanej autentykacji kontroler wysyła do aplikacji klienckiej informacje o powodzeniu kod statusu 200 oraz token JWT potrzebny do identyfikacji użytkownika. W sytuacji gdy, użytkownik nie został znaleziony lub dane logowania nie są poprawne zwrócone zostaną kody 401 (Unauthorized) lub 400 (BadRequest).

A screen shot of a computer

Description automatically generated

**Rysunek 29 Implementacja „LoginUser” enpoint**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

A screen shot of a computer program

Description automatically generated

**Rysunek 30 Implementacja „GetJwtToken” - serwis „AuthService”**

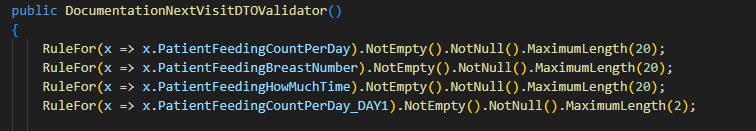
**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

W ramach podsumowania, podejście wykorzystywane przez LucyCover dotyczące podziału aplikacji na abstrakcje kontrolera, serwisu i modelu pozwala zachować czytelny i łatwo skalowalny kod. Co jest podstawa zasadą dobrych wzorców programowania obiektowego.

### 3.4.3. Zaimplementowane walidowanie danych

Nowoczesne technologie webowe dostarczają wiele wbudowanych mechanizmów pozwalających zautomatyzować i usprawnić prace zespołów deweloperskich. Jednym z takich mechanizmów jest automatyczna wstępna walidacja otrzymywanych danych przez kontrolery zapewniana przez framework ASP .NET Core. Walidacja danych jest szczególnie istotna przed przekazaniem do bazy danych. Dzięki temu możemy uchronić aplikację backendową oraz dane przez utratą, niespójnością oraz atakami hackerskimi. ASP .NET Core udostępnia atrybut [ApiController], który dba aby dostarczane na endpointy dane były zgodne z zadeklarowanym typem danych. Jego głównym zadaniem jest sprawdzenie aby otrzymywane dane były spójne i odpowiednie dla oczekiwanego typu po stronie serwera.

Niestety w niektórych przypadkach wymagana jest bardziej zaawanasowana walidacja sprawdzająca więcej aspektów w otrzymywanych danych. Do tego celu wykorzystano bibliotekę FluentValidator udostępniającą interfejsy do tworzenia własnych bardziej zaawansowanych walidacji.

 Poniższy fragment prezentuje przykład sprawdzający czy otrzymywane dane dotyczące dokumentacji pacjenta spełniają warunki maksymalnej długości ciągu znaków.

**Rysunek 31 Implementacja walidatora dla „documentationNextVisitDTO”**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

A computer screen shot of a black screen

Description automatically generatedDodatkowym poziomem ochrony jest bezpośrednia walidacja danych przez bazę danych, która dba o spójność oraz odpowiednie typy przechowywanych informacji. Taką walidacje uzyskano wykorzystując atrybuty dostarczone przez system ORM entity framework pozwalające kontrolować wprowadzane informacje.

**Rysunek 32 Deklaracje atrybutów dla modelu „recommendation"**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

### 3.4.4.Wykorzystanie systemu dependency injection

Wstrzykiwanie zależności (eng. Dependency injection) to technika usprawniająca strukturę aplikacji. Pozwala na izolowanie poszczególnych fragmentów kodu oraz wspomaga budowanie modułowej i bardziej czytelnej aplikacji. Technika ta znacząco poprawia możliwości testowania aplikacji oraz zwiększa efektywność pracy deweloperów.

Technika ta polega na dostarczaniu instancji zarejestrowanych klas wszędzie tam gdzie są one oczekiwane. Dzięki temu zależności nie muszą być tworzone bezpośrednio w komponentach oraz pozwala na odczyt i edycje poszczególnych obiektów w wielu miejscach aplikacji. Dzięki DI moduły programu stają się bardziej niezależne od siebie. Zasada działania tej techniki polega na rejestracji modułów w kontenerze, który będzie monitorował i dostarczał zależności wszędzie tam gdzie będą w danej chwili wymagane.

Dependency injection dostarcza trzy cykle życia zależności w kontrolerze:

1. Singletone – obiekt tworzony jest tylko raz podczas uruchomienia aplikacji i wykorzystywany jest wszędzie tam gdzie jest potrzebny
2. Transient – obiekt tworzony jest za każdym razem gdy jest potrzebny (np. osobny dla każdego konstruktora
3. Scoped – obiekt tworzony jest raz na zakres i trwa tylko przez czas jednego zakresu na przykład przez czas trwania żądania http

W projekcie LucyCover w kontenerze zależności zdefiniowane zostały wszystkie serwisy używane w projekcie oraz wykorzystywane biblioteki. Tak przygotowane serwisy pozwalają na uzyskiwanie dostępu do obiektu z poziomu konstruktorów klas.

Poniższy przykład przedstawia zarejestrowane serwisy do obsługi kontrolerów.

A screen shot of a computer program

Description automatically generated

**Rysunek 33 Rejestracja serwisów w kontenerze „Dependency injection”**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

W ramach podsumowania, kontenery zależności są niezastąpionym mechanizmem używanym w nowoczesnych aplikacjach, znacząco usprawniającym prace poprzez zapewnianie instancji wymaganych klas we wszystkich miejscach aplikacji gdzie są one wykorzystywane.

### 3.4.5. Wykorzystanie własnych middleware w pipelinie

A screen shot of a computer program

Description automatically generatedW aplikacja webowych podczas przetwarzania zapytania http wykorzystywane są tak zwane middleware. Jak podaje masterbranch.pl - „To nic innego jak metoda (Action<HttpContext>), która w jakiś sposób przetwarza żądanie HTTP. Może je odczytywać, może zapisać coś w odpowiedzi na to żądanie, a także w jakiś sposób na nie zareagować. Więc – middleware to jest metoda, która przyjmuje w parametrze HttpContext (i dodatkowo kolejny middleware). Profesjonalnie nazywa się „oprogramowaniem pośredniczącym”. Żądanie http przechodzi przez szereg różnych takich metod tworzących tak zwany pipeline. Innymi słowy każde zapytanie http przychodzące do webserwisu jest obsługiwane przez szereg następujących po sobie metod zwanych middleware. Metody te tworzą pipline dosłownie tłumacząc „rurociąg”. Metody te maja za zadnie w określony sposób przetwarzać otrzymane żądanie i odpowiednio na nie odpowiadać.

**Rysunek 34 Implementacja middleware do obsługi wyjątków**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

Aplikacja LucyCover poza wykorzystaniem standardowych obowiązkowych middleware do działania webserwisu dodatkowo wykorzystuje własny autorski middleware przeznaczony do przechwytywania wyjątków, które wystąpiły podczas procesowania zapytania i odpowiednio reagowania na nie. Mowa tu o ExceptionHandler, którego zadaniem jest próba wywołania następnych po nim middleware i wykrycie czy na którymś etapie przetwarzania nie zostanie zwrócony wyjątek. Po wystąpieniu wyjątku sprawdzony zostaje powód jego wystąpienia i do klienta zwracany jest odpowiedni kod statusu oraz wiadomość. Rysunek nr 32 przedstawia opisywaną metodę. Tak przygotowane funkcja musi zostać zarejestrowana w opisanym w poprzednim rozdziale kontenerze zależności oraz zostać odpowiednio zaimplementowana w pipline aplikacji pośród pozostałych middleware.

A black background with text and letters

Description automatically generated with medium confidence

**Rysunek 35 Rejestracja „ExceptionHandler” w kontenerze zależności**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

A black rectangle with white text

Description automatically generated

**Rysunek 36 Dodanie middleware jako element pipeline**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

### 3.4.6. Obsługa protokołów poczty elektronicznej z wykorzystaniem MailKit

Jednym z założeń projektowych aplikacji LucyCover jest zapewnienie stałej komunikacji pacjenta z doradcą. Pacjent powinien być na bieżąco informowany o planowanych wizytach. Dodatkowo położna powinna mieć możliwość udostępniania plików zawierających materiały edukacyjne za pośrednictwem poczty elektronicznej. W związku z tym koniecznym było zaimplementowanie technologii obsługującej protokoły pocztowe IMAP [[100]](#footnote-100) oraz SMTP[[101]](#footnote-101).

Do tego celu wykorzystano dostępną darmową technologie MailKit. Technologia ta umożliwia łatwą konfigurację protokołów pocztowych oraz korespondowania za pośrednictwem poczty z poziomu aplikacji. Autorzy artykułu dotyczącego tej technologii na stronie tempmail.us.com w superlatywach wyrażają się na temat możliwości i dostępności API tej technologi – „MailKit wyróżnia się solidnością i elastycznością, zapewniając programistom narzędzia potrzebne do skutecznego wysyłania, odbierania i zarządzania wiadomościami e-mail. Obsługuje różne protokoły, takie jak IMAP, POP3 i SMTP, zapewniając kompatybilność i funkcjonalność na różnych serwerach i usługach e-mail. Wykorzystując MailKit, programiści mogą łatwo dołączać pliki do wiadomości e-mail, poprawiając wygodę użytkownika, ułatwiając proste udostępnianie plików.”

Biorąc pod uwagę korzyści wynikające z użytkowania dostępnego rozwiązania, prostotę jego implementacji oraz poziom zoptymalizowania. Projekt LucyCover z powodzeniem wykorzystuję bibliotekę MailKit podczas zarządzania korespondencją pomiędzy pacjentem a pracownikiem przychodni. Aplikacja wykorzystuje protokoły pocztowe w sytuacji:

* Zaplanowanie nowej wizyty dla pacjenta
* Przesłanie do użytku materiałów edukacyjnych
* Wysyłanie i odczytywanie korespondencji z wybranym pacjentem z poziomu aplikacji.

Do celu realizacji takiej komunikacji utworzony została specjalny serwis – EmailService, który nawiązuje połączenie z zdalnym serwerem poczty elektronicznej gmail. Pełna konfiguracja protokołu IMAP oraz SMTP została ustalona w globalnych ustawieniach aplikacji – appsetings.json. EmailService dziedziczy po interfejsie IEmailService udostępniającym metody umożliwiające komunikacje pocztową:

* SendEmailAsync – Wysyłanie wiadomości pocztowych na wskazany adres email
* ReciveEmailsAsync – Pobieranie pełnej korespondencji użytkownika z wskazanym adresem.

Na poniższym rysunku wykazano sposób przechowywania konfiguracji pocztowej w pliku appsetings.json.

A screenshot of a computer

Description automatically generated

**Rysunek 37 Konfiguracja protokołów poczty elektronicznej w pliku „appsettings.json"**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

Konfiguracja ta pozwala na nawiązanie bezpiecznego połączenia z serwerem pocztowym. Aby wykorzystać ją w bibliotece MailKit koniecznym jest przekazanie jej do konfiguracji co zostało pokazane na kolejnych rysunkach.

A computer code on a black background

Description automatically generated

**Rysunek 38 Implementacja metody „ReciveEmailAsync**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

Powyższy przykład przedstawia implementacje metody ReciveEmailAsync przeznaczona do pobierania pełnej korespondencji pomiędzy systemem a pacjentem w formie kolekcji.

Dzięki tak zaimplementowanej technologii MailKit oraz wstrzyknięciem serwisu IEmailService do zależności kontrolera dependency injaction, aplikacja LucyCover jest w stanie wykorzystywać dostęp do poczty elektronicznej z każdego punktu aplikacji. Przyczynia się to do zachowania modułowości aplikacji oraz możliwości rozwoju. Dla przykładu, tak przygotowany serwis można wykorzystać podczas rejestracji użytkownika do wysyłania poprzez pocztę elektroniczną linków autoryzacyjnych służących do potwierdzenia adresu email podczas rejestracji.

### 3.4.7) Komunikacja z bazą danych

W tym rozdziale przedstawiony został sposób komunikacji z bazą danych w używanych serwisach wykorzystując interfejsy opisanego wcześniej wzorce projektowego repository.

Wszystkie serwisy zarejestrowane w LucyCover wymagające do pracy komunikacji z bazą danych korzystają z instancji klasy UnitOfWork, zawierającej spis interfejsów wszystkich repozytoriów. Dzięki tej klasie serwisy za pośrednictwem kontenera zależności mogą korzystać z dostępnych metod przetwarzania danych.

Poniższy przykład przedstawia wykorzystanie repozytoriów w celu dodania nowego pacjenta do systemu.

A screen shot of a computer

Description automatically generated

**Rysunek 39 Wykorzystanie repozytoriów podczas dodawania pacjenta do bazy danych**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

Wykorzystując metody „update” oraz „Add”, aplikacja modyfikuje dane przechowywane w bazie danych. W ramach doprecyzowania, należy zaznaczyć, że zastosowana implementacja repozytoriów oraz zasada działania entity framework tak naprawdę opiera się na tworzeniu transakcji ACID[[102]](#footnote-102), które następnie są przesyłane do serwera po wywołaniu metody „Save”. Dzięki temu wprowadzane przez nas zmiany działają na zasadzie „wprowadź wszystko albo nic”. Jest to szczególnie istotne w sytuacjach kiedy dane połączone ze sobą relacja są edytowane i ważnym jest aby dane modyfikowane były jednocześnie. Unikamy sytuacji w, której jakiś rekord został zmodyfikowany a drugi równolegle edytowany nie.

## 3.5. Omówienie autentykacji oraz autoryzacji użytkownika

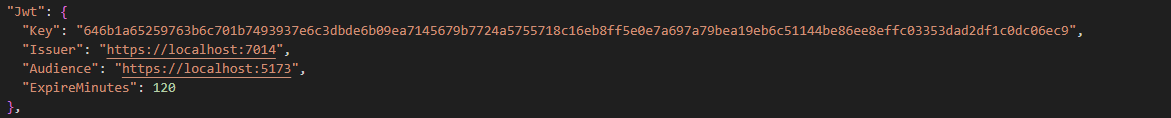
Budując nowoczesne aplikacje webowe kluczowym aspektem jest zwrócenie szczególnej uwagi na aspekty bezpieczeństwa jak i funkcjonalności aplikacji. Podstawowym mechanizmem kontroli dostępu do danych jest wdrożenie autentykacji. Pozwala kontrolować aby tylko uprawnione osoby miały dostęp do poszczególnych danych oraz funkcji aplikacji. Nie zapewniając takiej kontroli narażamy się na wycieki wrażliwych danych oraz próby wykonywania szkodliwych operacji. Aby uniknąć tego problemu oraz zapewnić bezpieczeństwo korzystającym użytkownikom LucyCover implementuje specjalne technologie przeznaczone do celów autentykacji oraz autoryzacji.

### 3.5.1. Implementacja tokenu JWTBarer

„JWT, czyli JSON Web Token to otwarty standard (RFC 7519) definiujący kompaktowy i samozawierający sposób przesyłania informacji między stronami jako obiekt [JSON](https://boringowl.io/tag/json). Informacje te mogą być sprawdzone i zaufane dzięki cyfrowemu podpisowi. Jest stosowany głównie do uwierzytelniania i autoryzacji w aplikacjach internetowych.“ – tak jak możemy przeczytać w artykule strony boringowl.io tokeny JWT są prostym ale niezawodnym mechanizmem autentykacji z wykorzystaniem JSON. Tokeny JWT oferują szereg możliwości do wprowadzenia w projekcie:

* Przechowywanie danych zalogowanego użytkownika wewnątrz tokenu za pośrednictwem tak zwanych „Claims”
* Ustalenie konkretnej daty oznaczającej kiedy token przestanie być ważny z punktu widzenia serwera
* Zaszyfrowanie wszystkich informacji skomplikowanymi algorytmami haszującymi uniemożliwiając ingerencje i zmianę wartości tokena
* Bezpośrednie ustalenie jakie domeny mogą posługiwać się tokenem podczas komunikacji co znacząco utrudnia ataki z nie znanych domen
* Wygodna forma przechowywania po stronie frontendu. W związku z tym, że token zapisany jest jako szyfr będący ciągiem znaków, taki token można wygodnie przechowywać w plikach cookies aplikacji i wykorzystywać go do komunikacji.

LucyCover implementuje mechanizm tokenów JWT wykorzystując specjalnie przygotowaną do tego celu bibliotekę „Microsoft.AspNetCore.Authentication.JwtBearer”. Wszystkie aspekty dotyczące konfiguracji przechowywane są w pliku appsetings.json. Jako potencjalna opcja rozwoju zaleca się aby poświadczenia szyfrowania przenieść do globalnych zmiennych systemowych serwera uruchamiającego aplikację w celu zwiększenia bezpieczeństwa aplikacji. Niestety przechowując kod źródłowy aplikacji na przykład w systemie kontroli wersji, ktoś obcy mógłby wyświetlić plik appsetings.json i wykorzystać przechowywaną tam konfigurację do celów ataków.



**Rysunek 40 Konfiguracja tokenu JWT w pliku „appsettings.json"**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

A screen shot of a computer program

Description automatically generatedNa rysunku 37 pokazano sposób przygotowania aplikacji do odbierania i walidowania tokenu JWT oraz konfiguracje appsetings.json.. Konfiguracja ta wykorzystywana jest następnie w głównym pliku aplikacji program.cs podczas implementacji mechanizmu deszyfrowania i walidowania tokenu.

**Rysunek 41 Implementacja mechanizmu obsługującego tokenu JWT w klasie „program.cs”**

**Źródło:** Opracowanie własne – zrzut ekranu z edytora tekstowego Visual Studio Code

### 3.5.3. Wymiana tokenu pomiędzy warstwami aplikacji

Kluczowym elementem w kontekście realizacji autentykacji użytkownika w systemie jest konkretne określenie sposobu przekazywania informacji dotyczących aktualnie zalogowanego użytkownika między warstwami aplikacji. Aplikacje wielowarstwowe, w których każda warstwa wykonana jest w odrębnej niezależnej technologii muszą mieć zdefiniowany pewien plan komunikacji. Korzystając z tokenu JWT aplikacje wymieniają między sobą zaszyfrowany ciąg znaków przechowujący konkretne informacje. Szczególnie warty zauważenia jest problem potencjalnej ingerencji w aplikacje aby zmanipulować obecność prawidłowego tokenu.

W celu ochrony przed takim atakiem oraz potrzebą zmaksymalizowania bezpieczeństwa aplikacji LucyCover wykorzystuje protokół http oraz mechanizm ciasteczek. W pierwszej kolejności z aplikacji klienckiej ( warstwy prezentacji ) wysyłane jest żądanie do serwera o próbę autentykacji oraz przekazane przez użytkownika danych. W przypadku prawidłowej autentykacji serwer generuje specjalny token JWT zapisując w nim wszystkie zdefiniowane wcześniej informacje, ustalając jego ostateczny termin ważności oraz szyfrując wszystkie informacje algorytmem haszującym. Następnie za pośrednictwem protokołu http zwracana jest odpowiedź z informacją o pomyślnym przebiegu algorytmu oraz następuje zapisanie tokenu w pliku cookie określonym jako http only. Korzystając z tokenu httponly zabezpieczamy się przed atakami typu XXS (Cross-Site Scripting)[[103]](#footnote-103) uniemożliwiając wykorzystanie tokenu za pośrednictwem API języka JavaScript. „Mimo że są one przechowywane w przeglądarce dokładnie tak samo jak zwykłe pliki cookie, mogą być dołączane tylko do żądań HTTP. Na przykład możesz ustawić opcję credentials:'include' pobierania danych z API, aby uwzględnić wszystkie pliki cookie, w tym ciasteczek HTTP-only, w nagłówkach żądania. W ten sposób atakujący XSS nie ma możliwości zdobycia ciasteczka HTTP-only, a więc nie uda mu się przejąć tokenów.” – tak jak podaje buldogjob.pl przechowywanie tokenów w ciasteczkach typu htttpOnly jest dobrą i bezpieczną praktyką. Tworząc takie ciasteczko w przeglądarce każda następna próba zapytania o zasoby do serwera będzie dodatkowo rozbudowana o nagłówek zawierający token. Podczas gdy zapytanie dociera do web serwera token jest walidowany i sprawdzane są jego poświadczenia, jeżeli wszystkie dane są poprawne, użytkownik otrzymuje dostępy do zasobów.

Tak zrealizowana komunikacja czyni projekt LucyCover bezpiecznym a dane pozostają odizolowane od osób trzecich. Wykorzystując token JWT oraz możliwości nowoczesnych przeglądarek internetowych implementacja mechanizmów logowania staje się łatwiejsza oraz bezpieczniejsza. Nie mniej jednak istotnym jest aby trzymać wszystkie krytyczne elementy tego mechanizmu takie jak klucz szyfrowania oraz sam token w bezpiecznych miejscach jak na przykład ciastka http only dla tokenu czy systemowe zmienne globalne dla klucza szyfrowania.

# Zakończenie

Niniejsza praca inżynierska dotyczyła zaprojektowania oraz implementacji aplikacji webowej LucyCover, która będzie wsparciem oraz innowacja w pracy położnych środowiskowych oraz doradców laktacyjnych w zakresie tworzenia, przetwarzania oraz przechowywania dokumentacji medycznej. System ten jest odpowiedzią na wyzwania związane z dotychczasowym przechowywaniem dokumentacji w formie papierowej w archiwach przychodni. W ramach realizacji projektu przedstawiono sposoby na zautomatyzowanie procesów planowania wizyt, komunikacji personelu z pacjentami oraz zapewnienia lepszej jakości dokumentacji.

Projekt bezpośrednio przyczynił się wzrostowi efektywności pracy personelu, bezpieczeństwa danych oraz wpłynął na ochronę środowiska poprzez zminimalizowanie zużycia papieru oraz tuszu. W wyniku realizacji projektu osiągnięto wszystkie z założonych celów co bezpośrednio przełożyło się na eliminację większości błędów wynikających z manualnego wprowadzania danych oraz zmniejszeniu zagrożeń i ryzyka utraty informacji. Wprowadzone mechanizmy walidowania danych spowodowały, znaczący spadek pomyłek w dokumentacjach oraz wzrost spójności danych z obowiązującymi standardami medycznym. System gotowy jest do uruchomienia w sieci lokalnej przedsiębiorstwa oraz pracy z wieloma użytkownikami poprzez implementacje kont użytkownika oraz wprowadzenia mechanizmu autentykacji.

Podczas rozwoju aplikacji napotkano szereg wyzwań związanych przede wszystkim z koniecznością zapewnienia najwyższych standardów bezpieczeństwa danych. Dodatkowo ze względu na potencjalny rozwój projektu w przyszłości o możliwość komunikacji zdalnej za pośrednictwem sieci VPN ważnym było już na etapie projektowania podstawowych aspektów aplikacji zachowywać spójność z standardami cyberbezpieczeństwa. Nie mniej jednak osiągnięto wstępne założenia projektowe i udało się stworzyć system, który w obecnej formie spełnia wymagania bezpieczeństwa dla lokalnych sieci.

W przyszłości zaplanowano rozwój obecnej bazowej aplikacji. Kolejnym kamieniem milowym będzie wprowadzenie pełnego szyfrowania danych zapisywanych do bazy danych za pomocą algorytmów szyfrujących. Na uwagę na pewno zasługuje pomysł przystosowania systemu do możliwości dostępu zdalnego za pośrednictwem zabezpieczonych sieci VPN. Z punktu widzenia jakości wytwarzanego kodu następnym krokiem będzie wprowadzenie testów jednostkowych zarówno dla warstwy biznesowej jaki i warstwy prezentacji aby zapewnić stabilność i jakość już istniejącej aplikacji podczas rozbudowy. Aby ułatwić administratorom systemu pracę świetnym pomysłem byłoby zaimplementowanie mechanizmu do tworzenia logów aby łatwiej monitorować ruch w aplikacji oraz rozwiązywać nieprawidłowości. W sytuacji kiedy aplikacja zyska na popularności wprowadzony zostanie specjalny panel administratora, który pozwoli szybko i prosto zarządzać systemem, użytkownikami oraz umożliwi szybkie tworzenie i przywracanie kopi zapasowych systemu z poziomu aplikacji.

Finalnym efektem pracy jest aplikacja, która z powodzeniem może zostać wdrożona do lokalnych sieci placówek medycznych oraz udostępniona do użytku pracownikom. LucyCover stanowi kolejny krok w stronę cyfryzacji oferując rozwiązania które poza zwiększaniem bezpieczeństwa znacząco przyczyniają się wzrostowi jakość, efektywności pracy oraz ochronie środowiska. Nie mniej jednak należy pamiętać, że sektor medyczny wymaga szczególnej uwagi i podążania wraz z nowymi trendami cyberbezpieczeństwa. Opisany projekt pozostawia wiele otwartych dróg, których wykorzystanie znacząco wpłynie dalsze usprawnienie pracy.

# 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/>
10. <https://boringowl.io/blog/dependency-injection-jak-poprawic-kod-i-zwiekszyc-jego-testowalnosc>
11. <https://masterbranch.pl/middleware-pipeline-czyli-serce-net-core/>
12. <https://boringowl.io/blog/jwt-klucz-do-bezpieczenstwa-w-aplikacjach-internetowych>
13. <https://bulldogjob.pl/readme/nie-dla-tokenow-uwierzytelniajacych-we-frontendzie>

# Spis rysunków

[**Rysunek 1 Wydajność platformy .NET** 7](#_Toc182579138)

[**Rysunek 2 Zastosowanie platformy .NET** 11](#_Toc182579139)

[**Rysunek 3 Schemat działania technologii EntityFramework** 12](#_Toc182579140)

[**Rysunek 4 Schemat działania tokenu JWT** 14](#_Toc182579141)

[**Rysunek 5 Schemat działania routingu w aplikacjach SPA** 17](#_Toc182579142)

[**Rysunek 6 Prezentacja graficznej części projektu** 21](#_Toc182579143)

[**Rysunek 7 Zastosowanie okien typu „popup” do zwiększenia przejrzystości aplikacji** 22](#_Toc182579144)

[**Rysunek 8 Implementacja „store” biblioteki Redux** 24](#_Toc182579145)

[**Rysunek 9 Implementacja „slice” biblioteki Redux** 25](#_Toc182579146)

[**Rysunek 10 Rejestracja „store” w komponencie „Provider” biblioteki Redux** 25](#_Toc182579147)

[**Rysunek 11 Implementacja „routera” biblioteki React Router** 27](#_Toc182579148)

[**Rysunek 12 Przypisanie „routera” do komponentu „provider” biblioteki React Router** 27](#_Toc182579149)

[**Rysunek 13 Implementacja „useFormData”** 28](#_Toc182579150)

[**Rysunek 14 Przykład pobierania danych z serwera za pośrednictwem Tanstack Query** 30](#_Toc182579151)

[**Rysunek 15 Definicja modelu „Recommendation”** 34](#_Toc182579152)

[**Rysunek 16 Przypisanie modeli bazodanowych do klasy „DbConnection”** 35](#_Toc182579153)

[**Rysunek 17 Wygenerowana migracja przy użyciu pakietu EntityFramework Tools** 35](#_Toc182579154)

[**Rysunek 18 Implementacja kontekstu baz danych w projekcie** 36](#_Toc182579155)

[**Rysunek 19 Rejestracja „DbContext” w kontenerze zależności aplikacji** 37](#_Toc182579156)

[**Rysunek 20 Konfiguracja aplikacji w pliku „appsettings.json”** 37](#_Toc182579157)

[**Rysunek 21 Implementacja interfejsu generycznego „IRepository”** 39](#_Toc182579158)

[**Rysunek 22 Implementacja klasy „Repository"** 40](#_Toc182579159)

[**Rysunek 23 Przykład implementacji repozytorium** 40](#_Toc182579160)

[**Rysunek 24 Implementacja kontenera repozytoriów – „UnitOFWork”** 41](#_Toc182579161)

[**Rysunek 25 Rejestracja repozytoriów w kontenerze zależności** 41](#_Toc182579162)

[**Rysunek 26 Pobranie zaleceń z bazy danych oraz mappowanie rezultatów na obiekt DTO** 42](#_Toc182579163)

[**Rysunek 27 Konfiguracja mapowania biblioteki „AutoMapper”** 43](#_Toc182579164)

[**Rysunek 28 Implementacja kontrolera do obsługi zapytań dot. autentykacji** 45](#_Toc182579165)

[**Rysunek 29 Implementacja „LoginUser” enpoint** 46](#_Toc182579166)

[**Rysunek 30 Implementacja „GetJwtToken” - serwis „AuthService”** 46](#_Toc182579167)

[**Rysunek 31 Implementacja walidatora dla „documentationNextVisitDTO”** 47](#_Toc182579168)

[**Rysunek 32 Deklaracje atrybutów dla modelu „recommendation"** 47](#_Toc182579169)

[**Rysunek 33 Rejestracja serwisów w kontenerze „Dependency injection”** 48](#_Toc182579170)

[**Rysunek 34 Implementacja middleware do obsługi wyjątków** 49](#_Toc182579171)

[**Rysunek 35 Rejestracja „ExceptionHandler” w kontenerze zależności** 50](#_Toc182579172)

[**Rysunek 36 Dodanie middleware jako element pipeline** 50](#_Toc182579173)

[**Rysunek 37 Konfiguracja protokołów poczty elektronicznej w pliku „appsettings.json"** 51](#_Toc182579174)

[**Rysunek 38 Implementacja metody „ReciveEmailAsync** 51](#_Toc182579175)

[**Rysunek 39 Wykorzystanie repozytoriów podczas dodawania pacjenta do bazy danych** 52](#_Toc182579176)

[**Rysunek 40 Konfiguracja tokenu JWT w pliku „appsettings.json"** 53](#_Toc182579177)

[**Rysunek 41 Implementacja mechanizmu obsługującego tokenu JWT w klasie „program.cs”** 54](#_Toc182579178)

# 

# Streszczenie

Uniwersytet WSB Merito Wrocław Wrocław, luty 2025

Wydział Finansów i Zarządzania

Kierunek: Informatyka

**Streszczenie pracy dyplomowej**

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

Autor: Jakub Czarnecki

Promotor: mgr. Inż. Wojciech Barczyński

Słowa kluczowe: informatyka, webserwis, medycyna

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. Praca podzielona została na część opisującą wykorzystane technologie oraz część omawiającą sposób implementacji aplikacji wykorzystując technikę budowania aplikacji wielowarstwowej.

Do stworzenia aplikacji przede wszystkim wykorzystano technologie React, Net 7, Vite, MSSQL server oraz Entity Framework.

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://poradnikprzedsiebiorcy.pl/-roznice-miedzy-http-a-https-i-ich-wplyw-na-pozycje-strony [↑](#footnote-ref-3)
4. 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-4)
5. https://www.ibm.com/docs/pl/was/9.0.5?topic=overview-three-tier-architectures [↑](#footnote-ref-5)
6. https://learn.microsoft.com/en-us/dotnet/csharp/ [↑](#footnote-ref-6)
7. https://dotnet.microsoft.com/en-us/ [↑](#footnote-ref-7)
8. https://learn.microsoft.com/en-us/ef/ [↑](#footnote-ref-8)
9. https://dotnet.microsoft.com/en-us/apps/aspnet [↑](#footnote-ref-9)
10. https://automapper.org/ [↑](#footnote-ref-10)
11. https://fluentvalidation.net/ [↑](#footnote-ref-11)
12. https://nlog-project.org/ [↑](#footnote-ref-12)
13. https://sandrino.dev/blog/aspnet-core-5-jwt-authorization [↑](#footnote-ref-13)
14. https://github.com/domaindrivendev/Swashbuckle.AspNetCore [↑](#footnote-ref-14)
15. https://boringowl.io/blog/zwiekszenie-wydajnosci-testowania-kodu-za-pomoca-xunit [↑](#footnote-ref-15)
16. https://fluentassertions.com/ [↑](#footnote-ref-16)
17. https://medium.com/@ojbrot/everything-you-need-to-know-about-html5-and-css3-e83c80b5bb8b [↑](#footnote-ref-17)
18. https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First\_steps/What\_is\_JavaScript [↑](#footnote-ref-18)
19. https://redux.js.org/ [↑](#footnote-ref-19)
20. https://reactrouter.com/en/main [↑](#footnote-ref-20)
21. https://tanstack.com/query/latest [↑](#footnote-ref-21)
22. <https://nodejs.org/en> [↑](#footnote-ref-22)
23. https://www.techopedia.com/definition/24290/intermediate-language-il-net [↑](#footnote-ref-23)
24. C# Marcin Lis Wydanie III Helion [↑](#footnote-ref-24)
25. https://bulldogjob.pl/readme/czym-sa-clr-jit-i-il-w-net [↑](#footnote-ref-25)
26. https://mindboxgroup.com/pl/co-to-jest-orm-oraz-czym-sie-rozni-od-sql-object-relational-mapping/ [↑](#footnote-ref-26)
27. https://www.codecademy.com/article/what-is-crud [↑](#footnote-ref-27)
28. https://learn.microsoft.com/pl-pl/ef/core/managing-schemas/migrations/?tabs=dotnet-core-cli [↑](#footnote-ref-28)
29. https://learn.microsoft.com/en-us/ef/core/what-is-new/ef-core-8.0/breaking-changes [↑](#footnote-ref-29)
30. https://learn.microsoft.com/pl-pl/aspnet/web-forms/what-is-web-forms [↑](#footnote-ref-30)
31. https://pl.wikipedia.org/wiki/Model-View-Controller [↑](#footnote-ref-31)
32. https://www.sote.pl/slownik-ecommerce/webapi [↑](#footnote-ref-32)
33. https://learn.microsoft.com/en-us/aspnet/core/tutorials/min-web-api?view=aspnetcore-8.0&tabs=visual-studio [↑](#footnote-ref-33)
34. https://www.rlogical.com/blog/benefit-and-drawbacks-of-asp-net-core-razor-pages/ [↑](#footnote-ref-34)
35. https://clockworkjava.pl/2020/12/obiekt-domenowy-dto-dao/ [↑](#footnote-ref-35)
36. https://justjoin.it/blog/solid-dobre-praktyki [↑](#footnote-ref-36)
37. https://pl.wikipedia.org/wiki/HTTP\_cookie [↑](#footnote-ref-37)
38. https://www.techtarget.com/searchapparchitecture/definition/API-endpoint [↑](#footnote-ref-38)
39. https://blog.theprotocol.it/slownik-it/assembly [↑](#footnote-ref-39)
40. https://kamee.pl/blog/122.wdrozenie-aplikacji-webowej [↑](#footnote-ref-40)
41. https://www.netia.pl/pl/srednie-i-duze-firmy/youtro-strefa-wiedzy/co-to-jest-siec-lan [↑](#footnote-ref-41)
42. https://www.expressvpn.com/pl/what-is-vpn [↑](#footnote-ref-42)
43. https://jerzywickowski.pl/flux/co-to-jest-flux/ [↑](#footnote-ref-43)
44. https://www.cloudflare.com/learning/serverless/glossary/client-side-vs-server-side/ [↑](#footnote-ref-44)
45. https://pl.wikipedia.org/wiki/Uniform\_Resource\_Locator [↑](#footnote-ref-45)
46. https://developer.mozilla.org/en-US/docs/Glossary/SPA [↑](#footnote-ref-46)
47. https://dev.to/mjubair/the-url-pathway-4e7j [↑](#footnote-ref-47)
48. https://reactrouter.com/en/main/route/loader [↑](#footnote-ref-48)
49. https://reactrouter.com/en/main/route/action [↑](#footnote-ref-49)
50. https://mediamarkt.pl/pl/content/technologie/komputery/pamiec-cache-co-to-jest-jak-ja-wyczyscic [↑](#footnote-ref-50)
51. https://pl.legacy.reactjs.org/docs/hooks-overview.html [↑](#footnote-ref-51)
52. 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-52)
53. https://pl.wikipedia.org/wiki/POP3 [↑](#footnote-ref-53)
54. https://pl.wikipedia.org/wiki/Simple\_Mail\_Transfer\_Protocol [↑](#footnote-ref-54)
55. https://en.wikipedia.org/wiki/Request\_for\_Comments [↑](#footnote-ref-55)
56. https://pl.wikipedia.org/wiki/Test\_jednostkowy [↑](#footnote-ref-56)
57. https://agile247.pl/podejscie-iteracyjne-oraz-przyrostowe/ [↑](#footnote-ref-57)
58. https://www.redhat.com/en/topics/devops/what-cicd-pipeline [↑](#footnote-ref-58)
59. https://www.atlassian.com/continuous-delivery/continuous-integration [↑](#footnote-ref-59)
60. https://www.atlassian.com/pl/continuous-delivery [↑](#footnote-ref-60)
61. https://zeet.co/blog/deploy-to-production [↑](#footnote-ref-61)
62. https://pl.wikipedia.org/wiki/Linux [↑](#footnote-ref-62)
63. https://pl.wikipedia.org/wiki/Linus\_Torvalds [↑](#footnote-ref-63)
64. https://pl.wikipedia.org/wiki/System\_plik%C3%B3w [↑](#footnote-ref-64)
65. https://blog.qnap.com/pl/snapshot-different-backup-pl/ [↑](#footnote-ref-65)
66. https://pl.wikipedia.org/wiki/Cross-site\_scripting [↑](#footnote-ref-66)
67. https://opencart.com.pl/co-to-jest-clickjacking [↑](#footnote-ref-67)
68. https://www.cloudflare.com/learning/access-management/phishing-attack/ [↑](#footnote-ref-68)
69. https://pl.wikipedia.org/wiki/Cross-site\_scripting [↑](#footnote-ref-69)
70. https://blog.trustisto.com/pl/przyklady-komunikatow-pop-up/#Poznaj\_Trust\_Overlay [↑](#footnote-ref-70)
71. <https://react.dev/learn/your-first-component> [↑](#footnote-ref-71)
72. https://legacy.reactjs.org/docs/faq-state.html [↑](#footnote-ref-72)
73. https://www.geeksforgeeks.org/what-is-prop-drilling-and-how-to-avoid-it/ [↑](#footnote-ref-73)
74. https://legacy.reactjs.org/docs/context.html [↑](#footnote-ref-74)
75. https://redux.js.org/api/store [↑](#footnote-ref-75)
76. https://kissdigital.com/pl/blog/single-page-application-jak-dziala-spa-i-czym-sie-rozni-od-mpa [↑](#footnote-ref-76)
77. https://bulldogjob.pl/readme/routing-po-stronie-klienta-z-react-router [↑](#footnote-ref-77)
78. https://pl.wikipedia.org/wiki/Uniform\_Resource\_Locator [↑](#footnote-ref-78)
79. https://legacy.reactjs.org/docs/hooks-overview.html [↑](#footnote-ref-79)
80. https://reactrouter.com/en/main [↑](#footnote-ref-80)
81. https://www.sonarsource.com/solutions/clean-code/ [↑](#footnote-ref-81)
82. Pojęcie zostało wytłumaczone w rozdziale 3.1.2 [↑](#footnote-ref-82)
83. https://httpd.apache.org/ [↑](#footnote-ref-83)
84. https://developer.mozilla.org/en-US/docs/Web/API/Fetch\_API/Using\_Fetch [↑](#footnote-ref-84)
85. https://portswigger.net/kb/issues/00501200\_dom-data-manipulation-dom-based [↑](#footnote-ref-85)
86. https://johnresig.com/ [↑](#footnote-ref-86)
87. https://jquery.com/ [↑](#footnote-ref-87)
88. https://kursjs.pl/kurs/es6/webpack [↑](#footnote-ref-88)
89. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules [↑](#footnote-ref-89)
90. https://webpack.js.org/concepts/hot-module-replacement/ [↑](#footnote-ref-90)
91. https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations/?tabs=dotnet-core-cli [↑](#footnote-ref-91)
92. https://learn.microsoft.com/en-us/ef/core/cli/ [↑](#footnote-ref-92)
93. https://learn.microsoft.com/en-us/aspnet/core/fundamentals/?view=aspnetcore-8.0&tabs=windows [↑](#footnote-ref-93)
94. https://instytutcyber.pl/encyklopedia/autoryzacja/ [↑](#footnote-ref-94)
95. https://instytutcyber.pl/encyklopedia/autentykacja/ [↑](#footnote-ref-95)
96. https://www.tutorialsteacher.com/linq/what-is-linq [↑](#footnote-ref-96)
97. https://learn.microsoft.com/pl-pl/aspnet/core/mvc/overview?view=aspnetcore-8.0 [↑](#footnote-ref-97)
98. https://main.pl/wiki/aplikacje-monolityczne-a-mikroserwisy/ [↑](#footnote-ref-98)
99. https://pl.wikipedia.org/wiki/HTTP\_cookie [↑](#footnote-ref-99)
100. https://pl.wikipedia.org/wiki/Internet\_Message\_Access\_Protocol [↑](#footnote-ref-100)
101. https://pl.wikipedia.org/wiki/Simple\_Mail\_Transfer\_Protocol [↑](#footnote-ref-101)
102. https://www.databricks.com/glossary/acid-transactions [↑](#footnote-ref-102)
103. https://owasp.org/www-community/attacks/xss/ [↑](#footnote-ref-103)