

AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

Wydział Nauk Inżynierijnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA
Bazy danych

Przychodnia Lekarska i Apteka

Autor:
Antoni Garczyński
Marcin Gonciarz

Prowadzący:
mgr inż. Nikodem Bulanda

Nowy Sącz 2023

Spis treści

1. Tytuł	4
2. Nazwa robocza	4
3. Cel	5
4. Zakres	6
4.1. Analiza wymagań	6
4.2. Wymagania funkcjonalne i niefunkcjonalne	7
4.3. Diagram przypadków użycia i diagram przepływu (opcjonalny)	9
4.4. Dobór technologii	13
4.5. Diagram ERD	15
5. Scenariusze	16
5.1. Lekarz	24
5.2. Aptekarz	27
5.3. Administrator	30
6. Estymacja czasowa	32
7. Implementacja	38
7.1. Backend	38
7.1.1. Logowanie	38
7.1.2. Rejestracja	40
7.1.3. Wyszukiwanie leków	42
7.1.4. Dodawanie i odczyt kartoteki pacjenta	43
7.1.5. Wyświetlanie recept	45
7.1.6. Wyszukiwanie recepty przez aptekarza i weryfikacja kodu	47
7.1.7. Edycja hasła	49
7.1.8. Konwersja pliku csv z bazą leków	51
7.1.9. Autoryzacja	55

7.1.10. Funkcje szyfrujące	57
7.1.11. Tworzenie harmonogramu przyjęć	58
7.1.12. Funkcje wyświetlające wolne terminy przyjęć	62
7.1.13. Wyświetlanie pacjentów do przyjęcie	64
7.2. Baza Danych	65
7.2.1. Tabela użytkowników	65
7.2.2. Tabela roli	66
7.2.3. Tabela specjalizacji	67
7.2.4. Tabela leków	68
7.2.5. Tabela ilości leków w aptece	69
7.2.6. Tabela kartotek	69
7.2.7. Tabela z receptami	70
7.2.8. Tabela z lekami z recepty	70
7.2.9. Trigger do tworzenia kodu recepty	71
7.2.10. Tabela z harmonogramem wizyt lekarskich	71
7.2.11. Tabela z zamówieniami	72
7.2.12. Tabela ze statusami zamówień	72
7.2.13. Tabela z lekami z zamówień	73
7.3. Frontend	74
7.3.1. Walidacja numeru PESEL i funkcja autodate	74
7.3.2. Funkcja przekazująca plik csv do backend'u	77
8. Testy i ich wyniki	79
Literatura	80
Spis rysunków	81
Spis tabel	82
Spis listingów	83
Raporty	83

1. Tytuł

Aplikacja webowa dla firmy FARM-MED

2. Nazwa robocza

Przychodnia lekarska i apteka

3. Cel

Celem stworzenia strony internetowej "Apteka i przychodnia" jest zapewnienie pacjentom łatwego i wygodnego dostępu do swoich recept, historii leczenia, zamówień w aptece oraz rezerwacji wizyt do lekarzy online. Lekarze na tej platformie mają dostęp do swojego harmonogramu, kartotek pacjentów oraz mogą wystawiać recepty pacjentom. Strona internetowa ma również sekcję apteki, gdzie pacjenci mogą realizować swoje recepty lub kupować lekarstwa, a farmaceuci mają możliwość przeglądania zamówień oraz edycji dostępności leków w aptece. Administrator ma pełną kontrolę nad aplikacją, może zarządzać lekarzami oraz aptekarzami (dodawać/usuwać konta) oraz ustalać harmonogram przyjęć lekarzy i dokonywać jego zmian. Celem strony jest zwiększenie wygody i dostępności do usług medycznych oraz umożliwienie lekarzom i farmaceutom efektywnego zarządzania swoimi zadaniami.

4. Zakres

4.1. Analiza wymagań

Aplikacja webowa ”Przychodnia Lekarska i Apteka” to platforma umożliwiająca pacjentom rezerwację wizyt online, wgląd do swoich recept i historii zamówień w aptece internetowej. Lekarze mają dostęp do bazy leków, kartotek pacjentów i mogą wykonywać recepty, a aptekarze mogą przygotowywać zamówienia do wysyłki.

Aplikacja składa się z kilku modułów takich jak:

- Moduł pacjenta (panel pacjenta)
- Moduł lekarza (panel lekarza)
- Moduł aptekarza (panel aptekarza)
- Moduł administratora (panel administratora)

W aplikacji dla różnych rodzajów kont, proces rejestracji oraz logowania wygląda podobnie, jednakże funkcjonalności dostępne w panelu użytkownika są zróżnicowane zgodnie z rodzajem konta.

Nowy pacjent, który chciałby skorzystać z usług przychodni, ma możliwość zarejestrowania się na portalu. Jeśli natomiast pacjent już posiada konto na platformie, po zalogowaniu zostaje przekierowany na swój panel. Lekarz, który chce rozpocząć swoją działalność w przychodni, zgłasza się do administratora portalu z prośbą o założenie konta na platformie. Administrator zakłada konto lekarzowi, . Lekarz, posiadający konto na platformie, loguje się i tak samo jak pacjent zostaje przekierowany do swojego panelu, który w znacznym stopniu różni się od panelu pacjenta. Nowo zatrudniony aptekarz, podobnie jak lekarz, musi zgłosić się do administratora portalu w celu założenia mu konta aptekarza. Po utworzeniu konta, aptekarz ma dostęp do panelu przeznaczonego dla niego. Administrator portalu, logując się do platformy, uzyskuje dostęp do panelu administratora w którym może zarządzać większością rzeczy znajdujących się na platformie

4.2. Wymagania funkcjonalne i niefunkcjonalne

Każdy użytkownik posiada panel o unikalnej funkcjonalności.

Panel pacjenta umożliwia łatwy dostęp do niezbędnych funkcji. Pacjent ma możliwość wglądu w swoje dane, zmiany hasła i numeru telefonu. Dodatkowo, pacjent ma dostęp do harmonogramu wizyt lekarza i może wybrać dogodny termin wizyty oraz zarejestrować się na nią. Po każdej wizycie, pacjent może sprawdzić swoją historię leczenia oraz zobaczyć przepisane leki i zalecenia dotyczące ich stosowania. Ostatnią opcją w panelu pacjenta jest możliwość przejścia do strony apteki.

Panel pacjenta w aptece umożliwia pacjentowi dostęp do swoich danych i recept. Pacjent może złożyć zamówienie w aptece, aby paczka była gotowa do odbioru. Klient może śledzić historię swoich zamówień i sprawdzić status bieżącego zamówienia. Produkty wybrane przez pacjenta trafiają do koszyka, który może on edytować (dodawać i usuwać produkty). Ostatnią opcją w panelu apteki jest przekierowanie do strony panelu pacjenta w przychodni.

Panel lekarza umożliwia lekarzowi przyjmowanie i obsługę pacjentów oraz łatwy dostęp do danych pacjenta. Lekarz ma dostęp do swojego harmonogramu, w którym wyświetlane są dane pacjentów, których musi przyjąć w danym dniu o wyznaczonych godzinach. Lekarz ma dostęp do kartotek wszystkich pacjentów, których może nadpisywać i uzupełniać historię leczenia. Ostatnią funkcją jest możliwość wystawiania recept przez lekarza na portalu, z dostępem do bazy leków i dawkowania. Bedzie mógł również sam dopisać jakie dawkowanie ustalił oraz zalecenia w przyjmowaniu danego leku.

Panel aptekarza umożliwia łatwe zarządzanie apteką i zamówieniami. Aptekarz ma dostęp do zamówień złożonych przez pacjentów i jest w stanie zmienić status zamówienia na "zamówienie gotowe". Aptekarz ma również dostęp do bazy leków i jest w stanie zmienić ilość posiadanych leków w aptece. Ostatnią funkcją jest dostęp do recept pacjentów posiadających konta w przychodni, bez konieczności podawania kodów recept. Wystarczy podać numer PESEL.

Administrator posiada najwięcej możliwości w portalu, które służą do zarządzania portalem. Ma on możliwość zarządzania kontami i ich uprawnieniami, dodawania i usuwania kont lekarzy oraz aptekarzy. Administrator zarządza harmonogramami przyjęć lekarzy, przypisując im dni i godziny oraz gabinety. Administrator ma również możliwość zarządzania bazą leków i jest w stanie zmienić ilość posiadanych dawek danego leku oraz dodać nowe leki do bazy. Dzięki temu system jest aktualny i pozwala na sprawną realizację recept przez aptekarzy.

Dodatkową funkcją która działała na koncie pacjenta będzie powiadomienie o

zwolnionym terminie wizyty w przeciągu najbliższego tygodnia.

Aplikacja działa w trybie klient-serwer. Klienci łączą się z serwerem poprzez protokół HTTP(S). Wszystkie dane są przechowywane w bazie danych. Interfejs użytkownika został zaprojektowany w oparciu o React, Taiwind i JavaScript. Aplikacja jest skalowalna i może obsługiwać wiele jednoczesnych użytkowników.

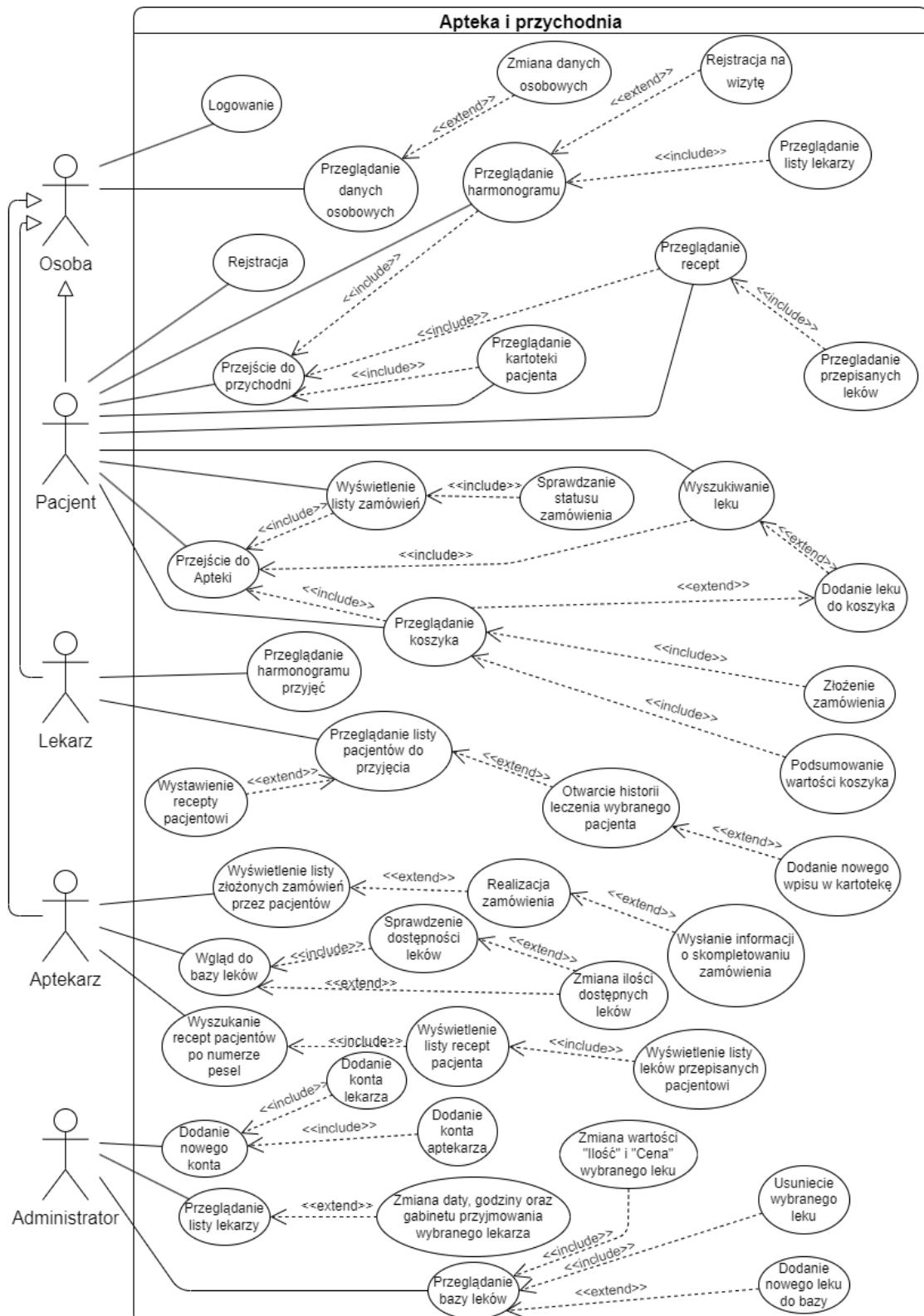
Dodatkowo, aplikacja jest w pełni responsywna i dostosowana do różnych urządzeń, co umożliwia korzystanie z niej zarówno na komputerze, tablecie, jak i smartfonie. Interfejs aplikacji został zaprojektowany w taki sposób, aby był czytelny i intuicyjny dla użytkowników o różnym stopniu zaawansowania technologicznego. Dostęp do różnych funkcjonalności jest łatwy i szybki, dzięki czemu użytkownicy mogą w prosty sposób wykonywać swoje zadania na platformie.

Aplikacja zapewnia również wysoki poziom bezpieczeństwa danych użytkowników poprzez wykorzystanie protokołu HTTPS oraz szyfrowania SSL. Każde konto użytkownika jest zabezpieczone hasłem, a także wymaga potwierdzenia adresu e-mail w celu weryfikacji tożsamości. Ponadto, wszystkie dane pacjentów są przechowywane w sposób zgodny z wymaganiami prawa i standardami branżowymi, a dostęp do nich ma tylko upoważniony personel medyczny i aptekarze.

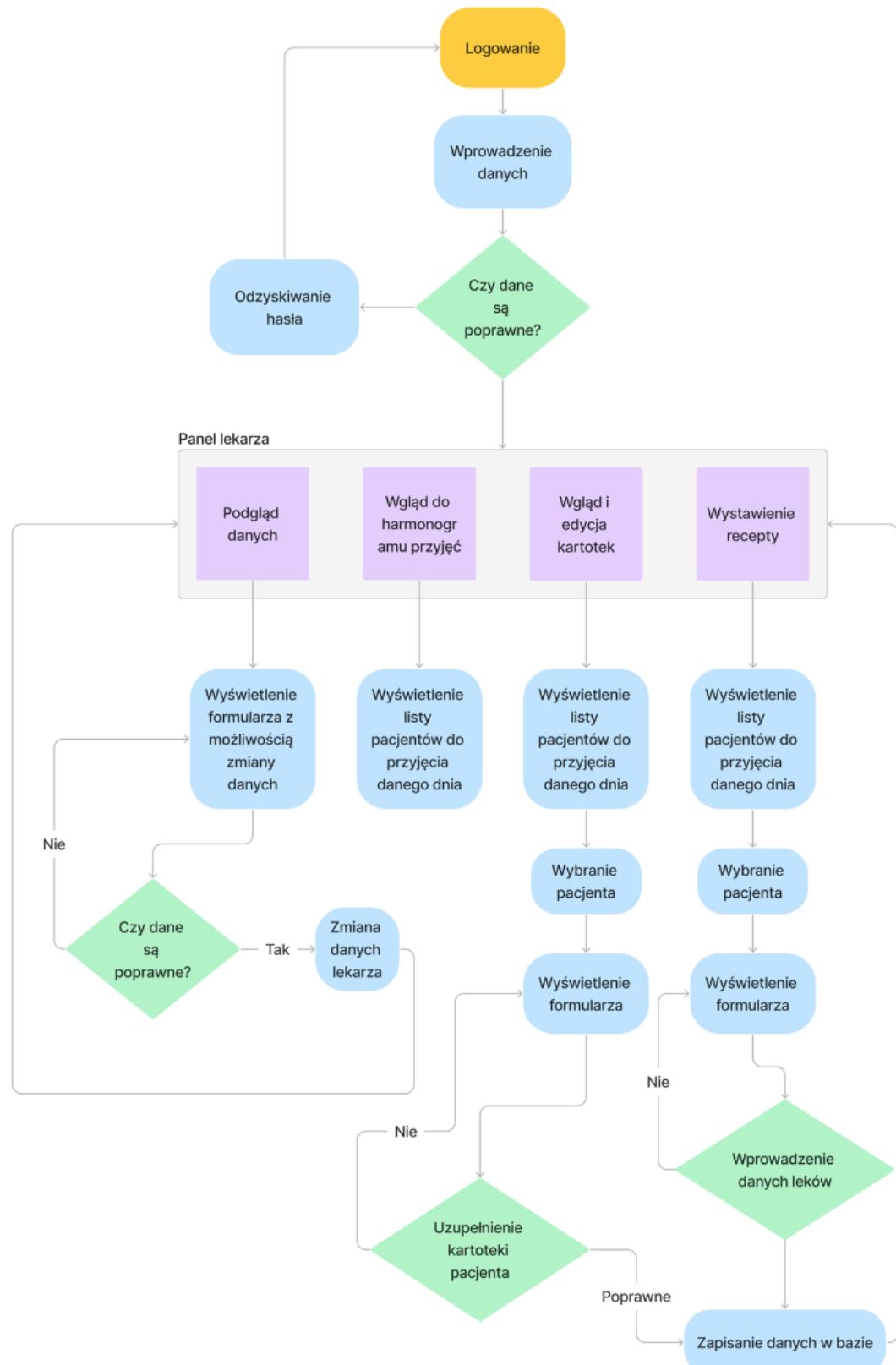
Aplikacja jest dostępna 24/7, z minimalnymi przerwami na konserwację i aktualizacje. Zadaniem administratorów systemu jest zapewnienie ciągłego działania serwera i aplikacji poprzez regularne przeglądy i utrzymanie infrastruktury sprzętowej.

Aplikacja "Przychodnia Lekarska i Apteka" jest ciągle rozwijana i ulepszana o nowe funkcjonalności, takie jak integracja z systemami e-recept czy e-zwolnień, aby zapewnić jak najlepsze doświadczenie dla użytkowników.

4.3. Diagram przypadków użycia i diagram przepływu (opcjonalny)

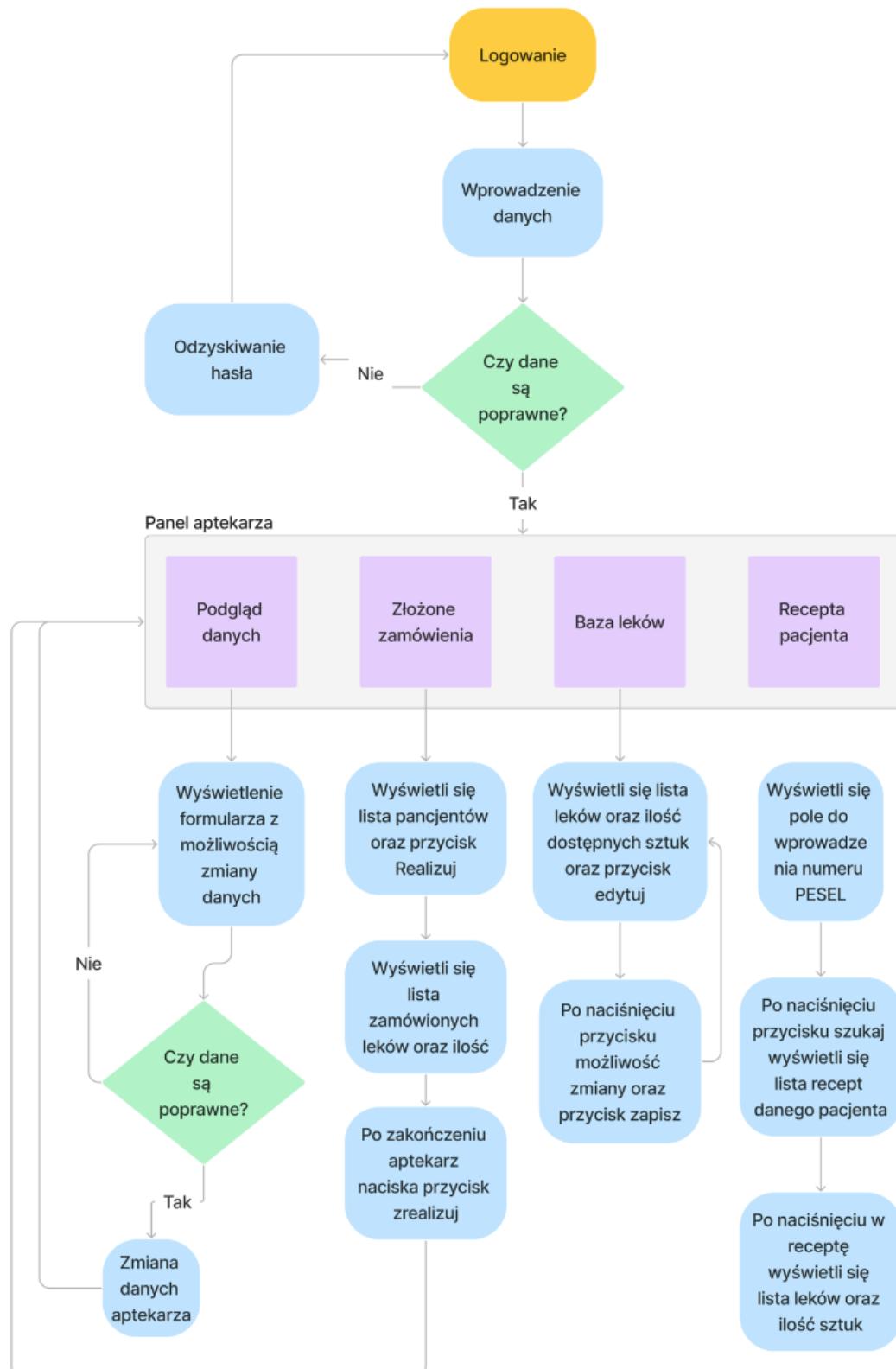


Rys. 4.1. Diagram przypadków użycia



Rys. 4.2. Panel lekarza

Logowanie aptekarza



Rys. 4.3. Panel aptekarza



Rys. 4.4. Panel administratora

4.4. Dobór technologii

W procesie tworzenia naszej aplikacji webowej zdecydowaliśmy się na wykorzystanie szeregu technologii, które pozwolą nam na stworzenie wysokiej jakości i wydajnej aplikacji. Poniżej przedstawiamy bardziej szczegółowe opisy każdej z używanych przez nas technologii:

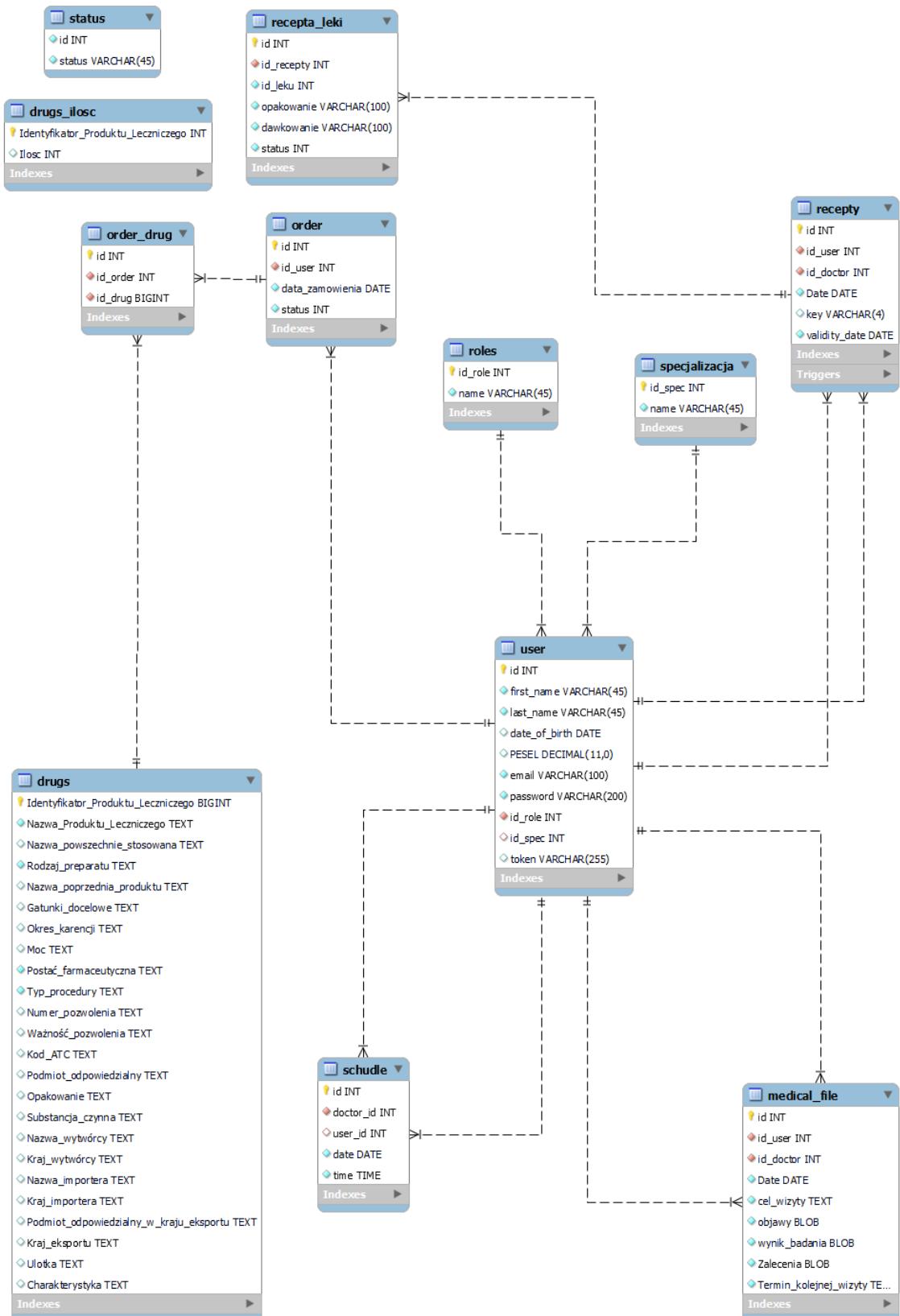
1. **Node.js** - jest to środowisko uruchomieniowe JavaScript, które pozwala nam na tworzenie aplikacji serwerowych w języku JavaScript. Node.js jest wydajne i skalowalne, co umożliwia nam tworzenie aplikacji, które są w stanie obsłużyć dużą ilość użytkowników. W naszej aplikacji używamy Node.js do stworzenia back-endu.
2. **Express.js** - to minimalistyczny framework dla Node.js, który pozwala nam na łatwe i szybkie tworzenie aplikacji webowych. Express.js umożliwia nam obsługę żądań HTTP, definiowanie ścieżek URL i przesyłanie danych z back-endu do front-endu. W naszej aplikacji używamy Express.js do tworzenia API naszej aplikacji.
3. **React** - to biblioteka JavaScript, która pozwala nam na tworzenie dynamicznych i interaktywnych interfejsów użytkownika. React jest bardzo popularny i dobrze udokumentowany, co ułatwia pracę z nim. React umożliwia nam tworzenie reużywalnych komponentów interfejsu użytkownika, co przyspiesza proces tworzenia aplikacji. W naszej aplikacji używamy React do tworzenia front-endu.
4. **JavaScript** - jest to język programowania, który umożliwia nam tworzenie dynamicznych i interaktywnych aplikacji webowych. JavaScript jest językiem skryptowym, co oznacza, że jest interpretowany przez przeglądarkę internetową. W naszej aplikacji używamy JavaScriptu w połączeniu z React, aby stworzyć wydajny i skuteczny front-end.
5. **HTML** - to język znaczników, który pozwala nam definiować struktury naszej aplikacji webowej. HTML jest bardzo ważnym elementem tworzenia stron internetowych, ponieważ określa on strukturę dokumentu. W naszej aplikacji używamy HTML w połączeniu z React, aby stworzyć strukturę naszego front-endu.
6. **CSS** - to język stylów, który pozwala nam definiować wygląd naszej aplikacji webowej. CSS umożliwia nam definiowanie stylów dla elementów HTML, co pozwala na tworzenie atrakcyjnego interfejsu użytkownika. W naszej aplikacji

używamy CSS w połączeniu z React i HTML, aby stworzyć interfejs użytkownika.

7. **Tailwind** - to framework CSS, który pozwala nam łatwo tworzyć responsywne i estetyczne interfejsy użytkownika. Tailwind jest bardzo popularnym framework'em CSS, który umożliwia tworzenie interfejsów użytkownika przy użyciu klas CSS, co znacznie przyspiesza proces tworzenia stylów dla naszej aplikacji. W naszej aplikacji używamy Tailwind do definiowania stylów naszych komponentów interfejsu użytkownika.
8. **MySQL** - to również relacyjna baza danych, która umożliwia nam składowanie i zarządzanie danymi w naszej aplikacji webowej. MySQL jest bardzo popularnym systemem zarządzania bazami danych, który oferuje wiele funkcji, takich jak transakcje, zapytania SQL i indeksowanie. W naszej aplikacji używamy MySQL do przechowywania danych, takich jak informacje o produktach i zamówieniach.

Podsumowując, wykorzystanie tych technologii umożliwiło nam stworzenie wydajnej, skalowalnej i estetycznej aplikacji webowej, która spełnia potrzeby naszych użytkowników.

4.5. Diagram ERD



Rys. 4.5. Diagram ERD

5. Scenariusze

Tab. 5.1. Rejestracja pacjenta

Nazwa	Scenariusz 1 - Rejestracja pacjenta
Aktor	Pacjent
Warunki początkowe	Pacjent nie jest zarejestrowany w serwisie
Opis	Scenariusz opisuje proces rejestracji pacjenta w serwisie
Ścieżki główne	<ol style="list-style-type: none"> 1. Pacjent wchodzi na stronę rejestracyjną serwisu i kliką przycisk "Zarejestruj się". 2. System wyświetla formularz rejestracyjny, w którym pacjent musi podać swoje dane osobowe i kontaktowe. 3. Pacjent wprowadza swoje dane osobowe i kontaktowe w odpowiednie pola formularza. 4. Pacjent kliką przycisk "Zarejestruj się". 5. System zapisuje dane pacjenta w bazie danych i przekierowuje go do "panelu pacjenta". 6. Pacjent ma dostęp do swojego konta i może korzystać ze wszystkich funkcjonalności serwisu.
Ścieżki alternatywne	<p>4a. Jeśli podane przez pacjenta dane są niepoprawne lub niekompletne, system wyświetla komunikat o błędzie i prosi o wprowadzenie poprawnych danych.</p> <p>5a. Jeśli w trakcie rejestracji wystąpił błąd, system wyświetla odpowiedni komunikat i proponuje ponowną rejestrację.</p>
Warunki końcowe	Pacjent jest zarejestrowany w serwisie i ma dostęp do swojego konta.

Tab. 5.2. Logowanie

Nazwa	Scenariusz 2 - Logowanie
Aktor	Pacjent, Lekarz, Aptekarz
Warunki początkowe	Użytkownik nie jest zalogowany do systemu
Opis	Scenariusz opisuje proces logowania użytkowników do systemu.
Ścieżki główne	<ol style="list-style-type: none">1. Użytkownik przechodzi na stronę logowania systemu.2. System wyświetla formularz logowania, w którym użytkownik wprowadza swój email oraz hasło.3. Użytkownik kliką przycisk „Zaloguj się”.4. System sprawdza poprawność danych logowania.5. Jeśli dane logowania są poprawne, użytkownik zostaje zalogowany do systemu i przekierowany do odpowiedniego panelu.6. Jeśli dane logowania są niepoprawne, system wyświetla komunikat o błędzie i prosi o ponowne wprowadzenie danych logowania.
Ścieżki alternatywne	<ol style="list-style-type: none">5a. Jeśli użytkownik nie pamięta hasła, może kliknąć opcję „Przypomnij hasło” i przejść do procesu resetowania hasła.5b. Jeśli użytkownik nie pamięta emaila, może skontaktować się z administratorem systemu w celu odzyskania dostępu do konta.
Warunki końcowe	Użytkownik zostaje zalogowany do systemu lub otrzymuje informację o błędnych danych logowania.

Tab. 5.3. Panel pacjenta w przychodni

Nazwa	Scenariusz 3 - Panel pacjenta w przychodni
Aktor	Pacjent
Warunki początkowe	Pacjent jest zalogowany do systemu przychodni i ma uprawnienia do panelu pacjenta.
Opis	Scenariusz opisuje proces korzystania z panelu pacjenta w przychodni przez pacjenta.
Ścieżki główne	<ol style="list-style-type: none">1. Pacjent wybiera opcję "Panel pacjenta" w menu głównym systemu przychodni.2. System wyświetla pięć zakładek: "Dane pacjenta", "Rejestracja na wizytę", "Kartoteka pacjenta", "Recepty", "Apteka".3. Pacjent wybiera jedną z zakładek, aby wyświetlić odpowiadające jej informacje i funkcjonalności.4. System wyświetla wybrane informacje i funkcjonalności.
Ścieżki alternatywne	
Warunki końcowe	Pacjent korzysta z wybranych informacji i funkcjonalności panelu pacjenta w przychodni.

Tab. 5.4. Panel pacjenta w aptece

Nazwa	Scenariusz 4 - Panel pacjenta w aptece
Aktor	Pacjent
Warunki początkowe	Pacjent jest zalogowany do panelu pacjenta w aptece i ma odpowiednie uprawnienia.
Opis	Scenariusz opisuje proces korzystania z panelu pacjenta w aptece przez pacjenta.
Ścieżki główne	<p>1. Pacjent ma do wyboru cztery zakładki: "Dane pacjenta", "Zamówienia", "Składanie zamówień" i "Koszyk".</p> <p>2. Po wybraniu "Dane pacjenta" pacjent zostaje przekierowany do strony, na której znajdują się jego dane osobowe, w tym możliwość zmiany wybranych z nich.</p> <p>3. Po wybraniu "Zamówienia" pacjent zostaje przekierowany do strony, na której znajduje się historia jego zamówień.</p> <p>4. Po wybraniu "Składanie zamówień" pacjent zostaje przekierowany do strony, na której może złożyć zamówienie.</p> <p>5. Po wybraniu "Koszyk" pacjent zostaje przekierowany do strony, na której znajdują się przedmioty dodane do koszyka.</p> <p>6. Pacjent ma możliwość wybrania zakładki "Przychodnia", która przekieruje go do panelu pacjenta w przychodni.</p>
Ścieżki alternatywne	
Warunki końcowe	Pacjent korzysta z wybranych funkcjonalności panelu pacjenta w aptece.

Tab. 5.5. Dane pacjenta

Numer	5
Temat	Dane pacjenta
Aktor	Pacjent
Przebieg	Aby użytkownik mógł zmienić numer telefonu i hasło naciska przycisk „Edytuj dane” po czym zostaje przekierowany do strony z formularzem na której znajdują się jego dane z możliwością zmiany numeru telefonu i hasła. Pozostałe dane użytkownika nie podlegają edycji. Po wypełnieniu pól należy nacisnąć przycisk „Zapisz” a dane zostaną zaktualizowane w bazie.
Zakończenie	Użytkownik zostaje przekierowany do ”Panelu pacjenta”.
Zakończenie alternatywne	Wyświetla się błąd o błędnych danych w formularzu.

Tab. 5.6. Rejestracja na wizytę

Numer	6
Temat	Rejestracja na wizytę
Aktor	Pacjent
Przebieg	Użytkownik po kliknięciu w sekcję „Rejestracja na wizytę” zostaje przekierowany na stronę na której znajduje się lista lekarzy. Po wybraniu konkretnego lekarza użytkownikowi zostaje wyświetlony harmonogram z wolnymi terminami. Po wybraniu terminu pacjent zostaje zarejestrowany na wizytę do lekarza.
Zakończenie	Użytkownik zostaje przekierowany do pierwszej strony ”Rejestracja na wizytę”.
Zakończenie alternatywne	Zostaje wyświetlony komunikat o błędzie rejestracji.

Tab. 5.7. Kartoteka pacjenta

Numer	7
Temat	Kartoteka pacjenta
Aktor	Pacjent
Przebieg	Użytkownik klikając w tą zakładkę przechodzi na stronę na której jest wyświetlona lista z historią leczenia.
Zakończenie	Użytkownik zostaje przeniesiony na stronę z listą historii leczenia.
Zakończenie alternatywne	Użytkownikowi zostaje wyświetlony stosowny komunikat.

Tab. 5.8. Recepty

Numer	8
Temat	Recepty
Aktor	Pacjent
Przebieg	Użytkownik klikając w tą zakładkę przechodzi na stronę na której jest wyświetlona lista z dostępnymi receptami.
Zakończenie	Użytkownikowi zostaje wyświetlona lista recept wraz z ich zawartością.
Zakończenie alternatywne	Użytkownikowi zostaje wyświetlony stosowny komunikat.

Tab. 5.9. Apteka

Numer	9
Temat	Przejście do apteki
Aktor	Pacjent
Przebieg	Po naciśnięciu przycisku „Apteka” użytkownik zostaje przekierowany do panelu pacjenta w aptece.
Zakończenie	Użytkownik zostaje przeniesiony do panelu pacjenta w aptece.
Zakończenie alternatywne	Zostaje wyświetlony komunikat o błędzie.

Tab. 5.10. Zamówienia

Numer	10	
Temat	Zamówienia	
Aktor	Pacjent	
<p>Po przejściu do sekcji „Zamówienia” użytkownikowi zostaje wyświetlona lista z historią zamówień dokonanych w aptece. Jest to lista na której wypisane są takie informacje jak: data, lekarstwa, cena i status. W górnej części listy znajdują się najnowsze zamówienia zrealizowane oraz niezrealizowane przez aptekę.</p>		
Zamówienie złożone nie potwierdzone przez aptekarza	Zamówienie potwierdzone przez aptekarza	Zamówienie zrealizowane i odebrane przez pacjenta
Zamówienia które oczekują na realizację mają status „W przygotowaniu”.	Zrealizowane ale nie odebrane, „Do odbioru ”.	Zamówienia odebrane „Odebrane”.

Tab. 5.11. Składanie zamówień

Numer	11
Temat	Składanie zamówień
Aktor	Pacjent
Przebieg	wyswietlane są wszystkie lekarstwa dostępne w aptece. Użytkownik chcąc dodać lek do koszyka musi wybrać ilość po przez wprowadzenie liczby oraz kliknąć przycisk „Dodaj do koszyka”.
Zakończenie	Lekarstwo zostało dodane do koszyka.
Zakończenie alternatywne	Wyskoczył błąd danych lub lek jest na receptę.

Tab. 5.12. Koszyk

Numer	12
Temat	Koszyk
Aktor	Pacjent
Przebieg	W tej sekcji znajdują się leki które zostały dodane do koszyka w sekcji. Przedstawiona jest nazwa leku, ilość oraz cena. Dodatkowo wyświetlana jest finalna suma wszystkich leków w koszyku. Oprócz tego po liście leków jest przycisk „Zamów”.
Zakończenie	Powrót do panelu pacjenta i dopisanie zamówienia.
Zakończenie alternatywne	Wyskakuje błąd zamówienia.

Tab. 5.13. Przychodnia

Numer	13
Temat	Przejście do przychodni
Aktor	Pacjent
Przebieg	Po naciśnięciu przycisku „Przychodnia” użytkownik zostaje przekierowany do panelu pacjenta w przychodni.
Zakończenie	Użytkownik zostaje przeniesiony do panelu pacjenta w przychodni.
Zakończenie alternatywne	Zostaje wyświetlony komunikat o błędzie.

5.1. Lekarz

Tab. 5.14. Panel lekarza

Numer	14		
Temat	Panel lekarza		
Aktor	Lekarz		
W tym panelu lekarz posiadający uprawnienia lekarza ma wybór między czterema zakładkami.			
Dane lekarza	Harmonogram przyjęć	Kartoteki pacjentów	Wystaw receptę
Lekarz zostaje przekierowany do strony na której znajdują się jego dane osobowe oraz ma możliwość zmiany wybranych z nich.	Lekarz zostaje przekierowany do strony na której znajduje się harmonogram z przyjęciami pacjentów	Lekarz zostaje przekierowany do strony na której znajduje się lista pacjentów oraz jego kartoteka	Lekarz zostaje przekierowany do strony na której znajduje się możliwość wystawienia recepta pacjentowi

Tab. 5.15. Dane lekarza/aptekarza

Numer	15		
Temat	Dane pracownika		
Aktor	Lekarz, Aptekarz		
Przebieg	Aby lekarz/aptekarz mógł zmienić numer telefonu i hasło naciska przycisk „Edytuj dane” po czym zostaje przekierowany do strony z formularzem na której znajdują się jego dane z możliwością zmiany numeru telefonu i hasła. Pozostałe dane użytkownika nie podlegają edycji. Po wypełnieniu pól należy nacisnąć przycisk „Zapisz” a dane zostaną zaktualizowane w bazie.		
Zakończenie	Użytkownik zostaje przekierowany do ”Panelu lekarza” lub aptekarza w zależności kto jest zalogowany.		
Zakończenie alternatywne	Wyświetla się błąd o błędnych danych.		

Tab. 5.16. Harmonogram przyjęć

Numer	16
Temat	Harmonogram przyjęć
Aktor	Lekarz
Przebieg	Lekarz klikając w przycisk „Harmonogram przyjęć” wyświetla się mu harmonogram przyjęć na dany dzień w którym są takie informacje jak godzina przyjęcia, imię , nazwisko pacjenta.

Tab. 5.17. Kartoteki pacjentów

Numer	17
Temat	Kartoteki pacjentów
Aktor	Lekarz
Przebieg	Po kliknięciu przycisku kartoteki pacjentów lekarzowi wyświetli się lista pacjentów na dany dzień w której po kliknięciu w pacjenta otwiera się strona z historią leczenia. Po kliknięciu w przycisk „Dodaj wpis” po którym otworzy się formularz w którym lekarz ma możliwość wpisania opisu choroby, zaleceń i leczenia. Po wypełnieniu formularza i kliknięciu przycisku „Zapisz” dane zostają zapisane do bazy.
Zakończenie	Lekarzowi zostaje wyświetlona kartoteka.
Zakończenie alternatywne	Zostaje wyświetlony komunikat o błędzie.

Tab. 5.18. Wystaw receptę

Numer	18
Temat	Wystawianie recept
Aktor	Lekarz
Przebieg	Po kliknięciu w przycisk „Wystaw receptę” lekarzowi zostaje wyświetlony formularz do wystawiania recepty. Ma on do uzupełnienia pola: leki, opis dawkowania oraz płatność. Po zakończeniu wystawiania recepty lekarz kliką przycisk „Zapisz”.
Zakończenie	Lekarzowi zostaje wyświetlony formularz z polami do wypełnienia danymi.
Zakończenie alternatywne	Zostaje wyświetlony stosowny komunikat.

5.2. Aptekarz

Tab. 5.19. Panel aptekarza

Numer	19		
Temat	Wystaw receptę		
Aktor	Aptekarz		
W tym panelu aptekarz posiadający uprawnienia aptekarza ma wybór między czterema zakładkami.			
Podgląd danych	Złożone zamówienia	Baza leków	Recepta pacjenta
Użytkownik zostaje przekierowany do strony na której znajdują się jego dane osobowe oraz ma możliwość zmiany wybranych z nich	Użytkownik zostaje przekierowany do strony na której ma możliwość złożenia zamówienia	Aptekarz przechodzi do strony w której znajduje się lista leków wraz z ilością dostępnych sztuk w aptece i cena	W tej sekcji wyświetlane jest pole w którym aptekarz wprowadza numer PESEL pacjenta a następnie klikając przycisk „Szukaj”. Następnie znajdują się wyświetlane recepty pacjenta z konkretnym numerem PESEL.

Tab. 5.20. Złożone zamówienia

Numer	20
Temat	Złożone zamówienia
Aktor	Aptekarz
Przebieg	Po kliknięciu w sekcję „złożone zamówienia” zostaje wyświetlona lista zamówień pacjentów. Znajduje się tam imię, nazwisko, data złożenia zamówienia oraz przycisk „Realizuj” po którym aptekarz zostaje przekierowany do strony z realizacją zamówień. Po skompletowaniu zamówienia aptekarz kliką przycisk „Gotowe do odbioru” i zostaje przekierowany ponownie do sekcji zamówień.
Zakończenie	Aptekarzowi zostaje wyświetlona lista zamówień po których może je zrealizować.
Zakończenie alternatywne	Brak wyświetlonej listy lub brak możliwości realizacji.

Tab. 5.21. Baza leków

Numer	21
Temat	Wgląd do leków
Aktor	Aptekarz
Przebieg	Po kliknięciu w sekcję „Baza leków” zostaje wyświetlona lista leków wraz z ich ilością. Przy każdym z leków znajduje się przycisk „Edytuj” po naciśnięciu którego jest możliwość wpisania nowych wartości w komórce „Ilość”. Po zmianie wartości należy kliknąć przycisk „Zapisz” ilość dostępnych leków zostanie zaktualizowana.
Zakończenie	Aptekarzowi zostanie wyświetlona lista leków oraz możliwość edycji dostępności.
Zakończenie alternatywne	Nie zostanie wyświetlona lista leków lub nie będzie możliwości edycji co zostanie wyświetlone jako komunikat.

Tab. 5.22. Recepta pacjenta

Numer	22
Temat	Recepta pacjenta
Aktor	Aptekarz
Przebieg	Po kliknięciu w sekcję „Recepta pacjenta” pole w którym aptekarz wprowadza numer PESEL pacjenta a następnie kliką przycisk „Szukaj”. Następnie zostają wyświetlane recepty pacjenta z konkretnym numerem PESEL. Po kliknięciu w receptę wyświetla się leki oraz ilość sztuk, które zostały przepisane pacjentowi.
Zakończenie	Zostanie wyświetlona lista recept.
Zakończenie alternatywne	Nie zostanie wyświetlona lista recept lub pacjent nie zostanie znaleziony w bazie.

5.3. Administrator

Tab. 5.23. Panel administratora

Numer	23	
Temat	Panel administratora	
Aktor	Administrator	
W tym panelu administrator posiada wybór między trzema sekcjami		
Zarządzanie kontami	Edycja harmonogramów	Baza leków
W tej sekcji administrator ma możliwość zmiany statusów kont lub dodania konta aptekarza lub lekarza.	W tej sekcji administratorowi wyświetli się lista lekarzy, gdzie po kliknięciu w danego specjalistę otworzy się jego harmonogram.	W tej sekcji administrator przechodzi do strony w której znajduje się lista leków wraz z ilością dostępnych sztuk w aptece i ceną. Przy każdym z leków znajduje się przycisk "Edytuj" i "Usuń".

Tab. 5.24. Zarządzanie kontami

Numer	24
Temat	Zarządzanie kontami
Aktor	Administrator
Przebieg	Po kliknięciu w sekcję „Zarządzanie kontami” administratorowi wyświetli się strona na której będzie mieć możliwość dodania nowego konta dla lekarza lub aptekarza po naciśnięciu przycisku „Dodaj”. Po naciśnięciu zostaje przekierowany do formularza do uzupełnienia danych odpowiednich dla danego konta.
Zakończenie	Zostanie wyświetlony formularz który trzeba wypełnić poprawnymi danymi a następnie nowe konto zostanie dodane do bazy.
Zakończenie alternatywne	Wartości wprowadzone w pola są nie poprawne oraz zostanie wyświetlony stosowny komunikat.

Tab. 5.25. Edycja harmonogramów

Numer	25
Temat	Edycja harmonogramów
Aktor	Administrator
Przebieg	W sekcji "Edycja harmonogramów" administratorowi wyświetli się lista lekarzy, gdzie po kliknięciu w danego specjalistę otworzy się jego harmonogram. Przycisk edytuj harmonogram otwiera stronę w której administrator może zmienić daty, godziny oraz gabinet przyjmowania wybranego wcześniej lekarza.
Zakończenie	Baza zostaje zaktualizowana o nowe dni, godziny i gabinet przyjęć.
Zakończenie alternatywne	Wyskakuje błąd danych.

Tab. 5.26. Baza leków

Numer	26	
Temat	Baza leków	
Aktor	Administrator	
Przebieg	W tej sekcji administrator przechodzi do strony w której znajduje się lista leków wraz z ilością dostępnych sztuk w aptece i ceną. Przy każdym z leków znajduje się przycisk "Edytuj" i "Usuń" oraz na górze "Dodaj".	
Edytuj	Usuń	Dodaj
Możliwość wpisania nowych wartości w komórkach "Ilość" i "Cena". Podczas edycji w miejscu "Edytuj" pojawi się przycisk "Zapis".	Wyskoczenie okienka w której będziemy musieli potwierdzić usunięcie leku z bazy.	Otworzy się formularz w którym do uzupełnienia będzie nazwa leku, ilość i cena. Po wpisaniu tych danych administrator naciska przycisk "Zapisz" i lek zostaje dopisany do bazy.

6. Estymacja czasowa

0. Analiza wymagań i projektowanie.

- Analiza wymagań: 15 godzin
- Projektowanie interfejsu użytkownika: 10 godzin
- Tworzenie diagramów: 10 godzin
- Opis implementacji i testów: 6 godzin
- Całkowity czas: 41 godzin

1. Implementacja scenariusza nr 1.

- Implementacja backendu: 13 godziny
- Implementacja frontendu: 6 godzin
- Testowanie: 2h godziny
- Całkowity czas: 21 godzin

2. Implementacja scenariusza nr 2.

- Implementacja backendu: 22 godziny
- Implementacja frontendu: 6 godzin
- Testowanie: 1 godzina
- Całkowity czas: 29 godzin

3. Implementacja scenariusza nr 3.

- Implementacja backendu: 0,5 godziny
- Implementacja frontendu: 1 godzina
- Testowanie: 0.5 godziny
- Całkowity czas: 2 godziny

4. Implementacja scenariusza nr 4.

- Implementacja backendu: -
- Implementacja frontendu: -
- Testowanie: -
- Całkowity czas: -

5. Implementacja scenariusza nr 5.

- Implementacja backendu: 2,5 godziny
- Implementacja frontendu: 2 godziny
- Testowanie: 0.5 godziny
- Całkowity czas: 5 godzin

6. Implementacja scenariusza nr 6.

- Implementacja backendu: 1 godzina
- Implementacja frontendu: 3 godzin
- Testowanie: 1 godzina
- Całkowity czas: 5 godzin

7. Implementacja scenariusza nr 7.

- Implementacja backendu: 3 godziny
- Implementacja frontendu: 3 godziny
- Testowanie: 0.5 godziny
- Całkowity czas: 6.5 godziny

8. Implementacja scenariusza nr 8.

- Implementacja backendu: 4 godziny
- Implementacja frontendu: 3 godziny
- Testowanie: 0.5 godziny
- Całkowity czas: 7.5 godziny

9. Implementacja scenariusza nr 9.

- Implementacja backendu: -
- Implementacja frontendu: -
- Testowanie: -
- Całkowity czas: -

10. Implementacja scenariusza nr 10.

- Implementacja backendu: 1 godzina
- Implementacja frontendu: -
- Testowanie: -
- Całkowity czas: 1 godzina

11. Implementacja scenariusza nr 11.

- Implementacja backendu: -
- Implementacja frontendu: -
- Testowanie: -
- Całkowity czas: -

12. Implementacja scenariusza nr 12.

- Implementacja backendu: -
- Implementacja frontendu: -
- Testowanie: -
- Całkowity czas: -

13. Implementacja scenariusza nr 13.

- Implementacja backendu: -
- Implementacja frontendu: -
- Testowanie: -
- Całkowity czas: -

14. Implementacja scenariusza nr 14.

- Implementacja backendu: -
- Implementacja frontendu: 1 godzina
- Testowanie: 0.5 godziny
- Całkowity czas: 1.5 godziny

15. Implementacja scenariusza nr 15.

- Implementacja backendu: 1 godzina
- Implementacja frontendu: 2 godziny
- Testowanie: 0.5 godziny
- Całkowity czas: 3.5 godziny

16. Implementacja scenariusza nr 16.

- Implementacja backendu: 1 godzina
- Implementacja frontendu: 1 godzina
- Testowanie: 0.5 godziny
- Całkowity czas: 2,5 godziny

17. Implementacja scenariusza nr 17.

- Implementacja backendu: 2 godziny
- Implementacja frontendu: 3 godziny
- Testowanie: 1 godzina
- Całkowity czas: 6 godzin

18. Implementacja scenariusza nr 18.

- Implementacja backendu: 5 godzin
- Implementacja frontendu: 4 godziny
- Testowanie: 1 godzina
- Całkowity czas: 10 godzin

19. Implementacja scenariusza nr 19.

- Implementacja backendu: -
- Implementacja frontendu: 1 godzina
- Testowanie: 0.5 godziny
- Całkowity czas: 1.5 godziny

20. Implementacja scenariusza nr 20.

- Implementacja backendu: 5 godzin
- Implementacja frontendu: -
- Testowanie: -
- Całkowity czas: 5 godzin

21. Implementacja scenariusza nr 21.

- Implementacja backendu: -
- Implementacja frontendu: 2 godziny
- Testowanie: 0.5 godziny
- Całkowity czas: 2.5 godziny

22. Implementacja scenariusza nr 22.

- Implementacja backendu: 5 godzin
- Implementacja frontendu: -
- Testowanie: -
- Całkowity czas: 5 godzin

23. Implementacja scenariusza nr 23.

- Implementacja backendu: 1 godzina
- Implementacja frontendu: 10 godzin
- Testowanie: 0.5 godziny
- Całkowity czas: 10.5 godziny

24. Implementacja scenariusza nr 24.

- Implementacja backendu: 6 godzin
- Implementacja frontendu: 6 godzin
- Testowanie: 1 godzina
- Całkowity czas: 13 godzin

25. Implementacja scenariusza nr 25.

- Implementacja backendu: 3 godziny
- Implementacja frontendu: 3 godziny
- Testowanie: 0.5 godziny
- Całkowity czas: 6.5 godziny

26. Implementacja scenariusza nr 26.

- Implementacja backendu: 10 godzin
- Implementacja frontendu: 4 godziny
- Testowanie: 2 godziny
- Całkowity czas: 16 godzin

27. Wdrożenie i utrzymanie

- Postawienie Docker'a: 10 godzin
- Tworzenie struktury bazy danych: 12 godzin
- Wdrożenie: 10 godzin
- Testowanie: 2 godziny
- Całkowity czas: 34 godziny

28. Podsumowanie

- Suma czasów dla wszystkich etapów: 235,5 godzin
- Research i poprawa dokumentacji: 36,5 godziny
- Ostateczna estymacja czasowa: 272 godziny

29. Czas z raportów

- Suma czasów dla wszystkich etapów: 272 godzin
- Rezerwa na nieprzewidziane opóźnienia lub problemy: 40 godzin
- Ostateczna estymacja czasowa: 312 godziny

7. Implementacja

7.1. Backend

7.1.1. Logowanie

Kod¹ poniżej to funkcja obsługująca logowanie użytkownika. Wykonuje ona następujące czynności:

1. Wykonuje zapytanie SQL, aby odczytać id, id_roli i hasło użytkownika na podstawie podanego adresu e-mail.
2. Sprawdza, czy zapytanie zwróciło jakiekolwiek dane. Jeśli nie, zwraca odpowiedź z kodem 404 i wiadomością ”Niepoprawny email lub hasło”.
3. Porównuje podane hasło z hasłem pobranym z bazy danych przy użyciu bcrypt. Jeśli hasła nie pasują do siebie, zwraca odpowiedź z kodem 404 i wiadomością ”Niepoprawny email lub hasło”.
4. Szyfruje id użytkownika.
5. Generuje token JWT, zawierający id użytkownika i id_roli. Token jest podpisany kluczem ”secretKey” i ma czas ważności 1 godziny.
6. Aktualizuje pole ”token” w bazie danych dla danego użytkownika.
7. Określa ścieżkę przekierowania w zależności od id_roli użytkownika.
8. Usuwa hasło użytkownika z danych.
9. Zwraca odpowiedź z kodem 200, ustawiając plik cookie ”accessToken” z tokenem JWT, z opcjami httpOnly, secure i sameSite.
10. Przesyła odpowiedź JSON zawierającą token, ścieżkę przekierowania, zaszyfrowane id i id_roli użytkownika.

```
1  export const login = (req, res) => {
2    const q = "SELECT id , id_role, password FROM farmmed.user WHERE email = ?";
3    db.query(q, [req.body.email], (err, data) => {
4      if (err) return res.status(500).json(err);
5      if (!data[0]) return res.status(404).json("Niepoprawny email lub hasło");
6      const checkPassword = bcrypt.compareSync(req.body.password,
7        data[0].password);
8      if (!checkPassword) return res.status(404).json("Niepoprawny email lub
9        hasło");
10     const id = encrypt(data[0].id)
11     const token = jwt.sign({ id: data[0].id, id_role: data[0].id_role },
12       "secretKey", { expiresIn: '1h' });
13     const q2 = "UPDATE farmmed.user SET token = ? WHERE (id = ?)";
14     db.query(q2, [token, data[0].id], (error, result) => {
15       if (error) return res.status(500).json(error);})
16     let redirectPath = "";
17     switch (data[0].id_role) {
18       case 1:
19         redirectPath = "/admin";
20         break;
21       case 2:
22         redirectPath = "/user";
23         break;
24       case 3:
25         redirectPath = "/doctor";
26         break;
27       case 4:
28         redirectPath = "/chemist";
29         break;
30       default:
31         console.error('Nieprawidłowa rola użytkownika');
32         break;}
33     const { password, ...others } = data[0];
34     res.status(200)
35     .cookie("accessToken", token, {
36       httpOnly: true,
37       secure: true,
38       sameSite: "none",})
39     .json({token: token, redirectPath: redirectPath, id: id, role:
40       data[0].id_role });
41   });
42 }
```

Listing 1. Logowanie

7.1.2. Rejestracja

Kod² poniżej to funkcja obsługująca rejestrację użytkownika. Oto opis poszczególnych kroków:

1. Wykonuje się zapytanie SQL, aby sprawdzić, czy istnieje użytkownik o podanym adresie e-mail lub numerze PESEL.
2. Jeśli istnieje użytkownik o takim samym adresie e-mail, zwracana jest odpowiedź z kodem 409 i wiadomością "Użytkownik z podanym adresem e-mail już istnieje".
3. Jeśli istnieje użytkownik o takim samym numerze PESEL, zwracana jest odpowiedź z kodem 409 i wiadomością "Użytkownik z podanym numerem PESEL już istnieje".
4. Sprawdzane jest, czy numer PESEL spełnia wymagania walidacyjne.
5. Wykorzystuje się wyrażenie regularne, aby sprawdzić, czy hasło spełnia określone wymagania.
6. Tworzy się sól za pomocą funkcji bcrypt.genSaltSync(10).
7. Hasło jest hashowane za pomocą funkcji bcrypt.hashSync() i wygenerowanej soli. Wykonuje się zapytanie SQL, które dodaje nowego użytkownika do tabeli z danymi. Użytkownik otrzymuje rolę o id równym 2.
8. Jeśli wystąpił błąd podczas wykonywania zapytania, serwer zwraca odpowiedź z kodem 500 i przesyła błąd jako JSON.
9. Jeśli operacja przebiegła pomyślnie, serwer zwraca odpowiedź z kodem 200 i wiadomością "Utworzono konto".

Funkcja validatePESEL() jest pomocniczą funkcją do walidacji numeru PESEL. Sprawdza ona, czy długość numeru PESEL wynosi 11 cyfr i czy suma kontrolna jest poprawna. Jeśli numer PESEL jest prawidłowy, zwraca true, w przeciwnym razie zwraca false.

```
1  export const register = (req, res) => {
2    const checkUserQuery = "SELECT * FROM farmmed.user WHERE email = ? OR PESEL =
3      ?";
4    db.query(checkUserQuery, [req.body.email, req.body.PESEL], (err, data) => {
5      if (err) return res.status(500).send(err);
6      if (data.length) {
7        const existingUser = data.find(user => user.email === req.body.email);
8        if (existingUser) return res.status(409).send("Użytkownik z podanym Email
9          już istnieje");
10       const existingPESELUser = data.find(user => user.PESEL === req.body.PESEL);
11       if (existingPESELUser) return res.status(409).send("Użytkownik z podanym
12         numerem PESEL już istnieje");
13     }
14     if (!validatePESEL(req.body.PESEL)) return res.status(400).send("Nieprawidłowy
15       numer PESEL");
16     const passwordRegex =
17       /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$/;
18     if (!passwordRegex.test(req.body.password)) return res.status(400).send("Hasło
19       musi mieć co najmniej 8 znaków, jedną małą i jedną dużą literę, cyfrę oraz
20       znak specjalny ! @ # $ % ^ & * ( ) _ + - = { } [ ] | \ : ; \" < > , . ?
21       /");
22     const salt = bcrypt.genSaltSync(10);
23     const hashedPassword = bcrypt.hashSync(req.body.password, salt);
24     const q = `INSERT INTO farmmed.user (first_name, last_name, PESEL,
25       date_of_birth, email, password, id_role) VALUES (?,?,?,?,?,?)`;
26     const values = [req.body.first_name, req.body.last_name, req.body.PESEL,
27       req.body.date_of_birth, req.body.email, hashedPassword, 2];
28     db.query(q, values, (err, data) => {
29       if (err) return res.status(500).json(err);
30       return res.status(200).json("Utworzono konto");
31     });
32   });
33 };
34 function validatePESEL(pesel) {
35   const weights = [1, 3, 7, 9, 1, 3, 7, 9, 1, 3];
36   const length = pesel.length;
37   if (length !== 11) return false;
38   let sum = 0;
39   for (let i = 0; i < length - 1; i++) sum += parseInt(pesel.charAt(i)) *
40     weights[i];
41   const checksum = (10 - (sum % 10)) % 10;
42   return checksum === parseInt(pesel.charAt(length - 1));
43 }
```

Listing 2. Rejestracja

7.1.3. Wyszukiwanie leków

Kod³ poniżej to funkcja obsługująca wyszukiwanie leków. Oto opis poszczególnych kroków:

1. Wykonuje się zapytanie SQL, które pobiera informacje o lekach z tabeli "drugs". Do zapytania dołączone jest lewe łączenie (LEFT JOIN) z tabelą "drugs_ilosc" w celu uzyskania informacji o dostępnej ilości leków.
2. Zapytanie zawiera warunki, które sprawdzają, czy rodzaj preparatu to "Ludzki" oraz czy nazwa leku zawiera część zapytania wyszukiwania. Wykorzystywane są trzy różne warunki LIKE, aby umożliwić elastyczne dopasowanie nazwy leku.
3. Wykonuje się zapytanie do bazy danych, przekazując odpowiednie parametry.
4. Jeśli wystąpił błąd podczas wykonywania zapytania, serwer zwraca odpowiedź z kodem 500 i przesyła błąd jako odpowiedź.
5. Jeśli operacja przebiegła pomyślnie, serwer zwraca odpowiedź z kodem 200 i przesyła dane o lekach jako odpowiedź.

Funkcja obsługuje wyszukiwanie leków na podstawie zapytania przekazanego w ciele żądania (req.body.searchQuery). Wyszukiwanie jest wykonywane w oparciu o podobieństwo nazw leków do podanego zapytania.

```
1 export const search_drug = (req, res) => {
2   const q = "SELECT d.Identyfikator_Produktu_Leczniczego,
3   ↳ d.Nazwa_Produktu_Leczniczego, d.Moc, d.Postać_farmaceutyczna,
4   ↳ d.Podmiot_odpowiedzialny, d.Opakowanie, d.Substancja_czynna,
5   ↳ COALESCE(d1.Ilosc, 0) AS Ilosc FROM farmmed.drugs d LEFT JOIN
6   ↳ farmmed.drugs_ilosc d1 ON d.Identyfikator_Produktu_Leczniczego =
7   ↳ d1.Identyfikator_Produktu_Leczniczego WHERE d.Rodzaj_preparatu = 'Ludzki'
8   ↳ AND (d.Nazwa_Produktu_Leczniczego LIKE ? OR d.Nazwa_Produktu_Leczniczego
9   ↳ LIKE ? OR d.Nazwa_Produktu_Leczniczego LIKE ?) LIMIT 1000;";
10  db.query(q, [req.body.searchQuery + '%', '%' + req.body.searchQuery + '%',
11   ↳ '%' + req.body.searchQuery], (err, data) =>{
12     if (err) return res.status(500).send(err);
13     else return res.status(200).send(data);
14   });
15};
```

Listing 3. Wyszukiwanie leków

7.1.4. Dodawanie i odczyt kartoteki pacjenta

Te funkcje⁴ dotyczą dodawania i odczytu kartoteki medycznej pacjenta. Oto opis ich działania:

1. **add_file**: Ta funkcja obsługuje dodawanie nowego wpisu do kartoteki medycznej pacjenta. Wykonuje następujące czynności:

- Deszyfruje identyfikator użytkownika (pacjenta) za pomocą funkcji "decrypt".
- Wykonuje zapytanie SQL do bazy danych, które dodaje nowy wpis do tabeli "medical_file". Wartości niektórych pól są zaszyfrowane za pomocą funkcji "AES_ENCRYPT".
- Jeśli wystąpił błąd podczas wykonywania zapytania, serwer zwraca odpowiedź z kodem 500 i przesyła błąd jako odpowiedź.
- Jeśli operacja przebiegła pomyślnie, serwer zwraca odpowiedź z kodem 200 i wiadomość "Dodano wpis do kartoteki".

2. **all_file**: Ta funkcja obsługuje odczyt wszystkich wpisów w kartotece medycznej danego pacjenta. Wykonuje następujące czynności:

- Deszyfruje identyfikator użytkownika (pacjenta) za pomocą funkcji "decrypt".
- Wykonuje zapytanie SQL do bazy danych, które pobiera informacje o wpisach w tabeli "medical_file" dla danego pacjenta. Niektóre pola są zaszyfrowane, więc są one konwertowane za pomocą funkcji "CONVERT" i "AES_DECRYPT".
- Jeśli wystąpił błąd podczas wykonywania zapytania, serwer zwraca odpowiedź z kodem 500 i przesyła błąd jako odpowiedź.
- Jeśli operacja przebiegła pomyślnie, serwer zwraca odpowiedź z kodem 200 i przesyła dane o wpisach w kartotece medycznej jako odpowiedź.

Obie funkcje korzystają z identyfikatora użytkownika (pacjenta), który jest deszyfrowany przed wykonaniem zapytania do bazy danych.

```
1  export const add_file = (req, res) => {
2    const user = decrypt(req.body.user);
3    const q =
4      'INSERT INTO farmmed.medical_file (id_user, id_doctor, cel_wizyty, objawy,
5       ↳ wynik_badania, Zalecenia, Termin_kolejnej_wizyty) Value (?, ?, ?,
6       ↳ AES_ENCRYPT(?, "medical_file"), AES_ENCRYPT(?, "medical_file"),
7       ↳ AES_ENCRYPT(?, "medical_file"), ?)';
8    db.query(q, [req.body.id, user, req.body.cel, req.body.objawy, req.body.wynik,
9     ↳ req.body.zalecenia, req.body.termin], (err, data) => {
10      if (err) return res.status(500).send(err);
11      else return res.status(200).send('Dodano wpis do kartoteki');
12    });
13  };
14
15  export const all_file = (req, res) => {
16    const user = decrypt(req.body.user);
17    const q = `SELECT
18      m.id,
19      DATE_FORMAT(m.Date, "%Y-%m-%d") AS data_wizyty,
20      m.cel_wizyty,
21      CONVERT(AES_DECRYPT(m.objawy, "medical_file") USING utf8mb4) AS
22      ↳ objawy,
23      CONVERT(AES_DECRYPT(m.wynik_badania, "medical_file") USING
24      ↳ utf8mb4) AS wynik_badania,
25      CONVERT(AES_DECRYPT(m.Zalecenia, "medical_file") USING utf8mb4)
26      ↳ AS Zalecenia,
27      m.Termin_kolejnej_wizyty,
28      (SELECT CONCAT(u1.first_name, " ", u1.last_name) FROM
29       ↳ farmmed.user u1 INNER JOIN farmmed.medical_file m1 ON u1.id = m1.id_doctor
30       ↳ WHERE m1.id = m.id) AS doctor_name
31      FROM farmmed.medical_file m
32      INNER JOIN farmmed.user u ON m.id_user = u.id
33      WHERE u.id = ?
34      ORDER BY m.id DESC`;
35    db.query(q, [user], (err, data) => {
36      if (err) return res.status(500).send(err);
37      else {
38        return res.status(200).send(data);
39      }
40    });
41  };
42
```

Listing 4. Dodawanie i odczyt kartoteki pacjenta

7.1.5. Wyświetlanie recept

Kod⁵ poniżej to funkcja ”all_prescription”, która obsługuje wyświetlanie recept pacjenta. Oto opis jej działania:

1. Funkcja rozpoczyna od deszyfrowania identyfikatora użytkownika (pacjenta) za pomocą funkcji ”decrypt”.
2. Następnie tworzony jest zapytanie SQL do bazy danych, które pobiera informacje o receptach danego pacjenta. Zapytanie łączy kilka tabel, takich jak ”recepty”, ”user”, ”recepta_leki” i ”drugs”, aby uzyskać odpowiednie informacje.
3. Jeśli wystąpił błąd podczas wykonywania zapytania, serwer zwraca odpowiedź z kodem 500 i przesyła błąd jako odpowiedź.
4. Jeśli operacja przebiegła pomyślnie, dane zwrocone z bazy danych są przekształcane w odpowiednią strukturę. Wynik jest transformowany przy użyciu pętli forEach i Map.
5. Ostatecznie, przekształcone rezultaty są zwracane jako odpowiedź z kodem 200 w formacie JSON.

Funkcja ”all_prescription” umożliwia pacjentowi wyświetlanie swoich recept. Zwraca informacje takie jak identyfikator recepty, data wystawienia, nazwa pacjenta, nazwa lekarza, ważność recepty oraz szczegóły dotyczące leków na recepcie (nazwa leku, moc, opakowanie, dawkowanie i ulotka).

```
1  export const all_prescription = (req, res) => {
2      const user = decrypt(req.body.user);
3      const q = `SELECT r.id, r.key, DATE_FORMAT(r.Date, "%Y-%m-%d") AS
4          data_wystawienia, CONCAT(u.first_name, " ", u.last_name) AS user_name,
5          (SELECT CONCAT(u1.first_name, " ", u1.last_name) FROM farmmed.user u1 INNER
6          JOIN farmmed.recepty r1 ON u1.id = r1.id_doctor WHERE r1.id = r.id) AS
7          doctor_name, DATE_FORMAT(r.validity_date, "%Y-%m-%d") AS ważnosc_recepty,
8          d.Nazwa_Produktu_Leczniczego AS nazwa_leku, d.moc AS moc, rl.opakowanie AS
9          opakowanie, rl.dawkowanie AS dawkowanie, d.ulotka AS ulotka FROM
10         farmmed.recepty r INNER JOIN farmmed.user u ON r.id_user = u.id INNER JOIN
11         farmmed.recepta_leki rl ON r.id = rl.id_recepty INNER JOIN farmmed.drugs d
12         ON rl.id_leku = d.Identyfikator_Produktu_Leczniczego WHERE u.id = ? AND
13         (r.validity_date > (SELECT CURDATE())) ORDER BY r.id DESC`;
14
15     db.query(q, [user], (err, data) => {
16         if (err) return res.status(500).send(err);
17         else {
18             const transformedResult = [];
19             const resultMap = new Map();
20             data.forEach((row) => {
21                 const { id, key, data_wystawienia, user_name, doctor_name,
22                     ważnosc_recepty, id_leku, nazwa_leku, moc, opakowanie, dawkowanie,
23                     ulotka } = row;
24                 if (!resultMap.has(id)) {
25                     resultMap.set(id, {id, key, data_wystawienia, user_name, doctor_name,
26                         ważnosc_recepty, leki: [], });
27                 }
28                 resultMap.get(id).leki.push({id_leku, nazwa_leku, moc, opakowanie,
29                     dawkowanie, ulotka,
30                 });
31             });
32             transformedResult.push(...resultMap.values());
33             const recepty = JSON.stringify(transformedResult, null, 2);
34             return res.status(200).send(recepty);
35         }
36     });
37 };
38 
```

Listing 5. Wyświetlanie recept

7.1.6. Wyszukiwanie recepty przez aptekarza i weryfikacja kodu

Poniżej⁶ znajdują się dwie funkcje: "find_users_prescription" i "check_prescription", które są odpowiedzialne za wyszukiwanie recepty przez aptekarza oraz weryfikację kodu recepty. Oto opis ich działania:

1. Funkcja "find_users_prescription":

- Funkcja ta pobiera informacje o receptach na podstawie numeru PESEL pacjenta.
- Tworzone jest zapytanie SQL, które łączy tabelę "recepty" z tabelą "user" w celu pobrania informacji o receptach danego pacjenta na podstawie jego numeru PESEL.
- Jeśli wystąpił błąd podczas wykonywania zapytania, serwer zwraca odpowiedź z kodem 500 i przesyła błąd jako odpowiedź.
- Jeśli operacja przebiegła pomyślnie, dane zwrocone z bazy danych są przesyłane jako odpowiedź z kodem 200.

2. Funkcja "check_prescription":

- Ta funkcja służy do weryfikacji kodu recepty.
- Wykonuje zapytanie SQL w celu pobrania kodu recepty na podstawie podanego identyfikatora recepty.
- Jeśli wystąpił błąd podczas wykonywania zapytania, serwer zwraca odpowiedź z kodem 500 i przesyła błąd jako odpowiedź.
- Jeśli kod recepty zgadza się z podanym kodem w żądaniu, funkcja zwraca kod recepty jako odpowiedź z kodem 200. W przeciwnym razie zwracane jest "null".

Funkcje "find_users_prescription" i "check_prescription" umożliwiają aptekarzowi wyszukiwanie recepty na podstawie numeru PESEL pacjenta oraz weryfikację poprawności kodu recepty.

```
1  export const find_users_prescription = (req, res) => {
2    const q =
3      'SELECT r.id, CONCAT(u.first_name, " ", u.last_name) AS name, u.PESEL, r.Date
4       FROM recepty r INNER JOIN user u ON u.id = r.id_user WHERE u.PESEL = ?
5       AND (r.validity_date > (SELECT CURDATE())) AND ((SELECT count(*) FROM
6       farmmed.recepta_leki l WHERE l.id_recepty = r.id) != 0) ORDER BY r.id
7       DESC';
8
9    db.query(q, [req.body.searchQuery], (err, data) => {
10      if (err) {
11        console.error(err);
12        return res.status(500).send(err);
13      }
14      return res.status(200).send(data);
15    });
16  };
17
18
19
20  export const check_prescription = (req, res) => {
21    const q = 'SELECT `key` FROM farmmed.recepty WHERE id= ?';
22    db.query(q, [req.body.id], (err, data) => {
23      if (err) {
24        console.error(err);
25        return res.status(500).send(err);
26      }
27      if (data[0].key === req.body.key) {
28        return res.status(200).send(data[0]);
29      } else return null;
30    });
31  };
32
```

Listing 6. Wyszukiwanie recepty przez aptekarza i weryfikacja kodu

7.1.7. Edycja hasła

Funkcja ”edit_password”⁷ jest odpowiedzialna za edycję hasła użytkownika. Oto opis jej działania:

1. Pobierane jest id użytkownika na podstawie zaszyfrowanego tokenu autoryzacyjnego.
2. Tworzone jest zapytanie SQL, które pobiera aktualne hasło użytkownika na podstawie jego id.
3. Jeśli wystąpił błąd podczas wykonywania zapytania, serwer zwraca odpowiedź z kodem 500 i przesyła błąd jako odpowiedź.
4. Jeśli operacja przebiegła pomyślnie, funkcja używa funkcji ”bcrypt.compare” do porównania podanego hasła w żądaniu z zaszyfrowanym hasłem pobranym z bazy danych.
5. Jeśli wystąpił błąd podczas porównywania haseł, serwer zwraca odpowiedź z kodem 500 i przesyła błąd jako odpowiedź.
6. Jeśli hasła się zgadzają, generowany jest nowy sól (salt) przy użyciu funkcji ”bcrypt.genSaltSync” oraz tworzone jest nowe zaszyfrowane hasło za pomocą funkcji ”bcrypt.hashSync” na podstawie nowego hasła podanego w żądaniu i wygenerowanej soli.
7. Następnie tworzone jest zapytanie SQL do aktualizacji hasła w bazie danych na nowe zaszyfrowane hasło.
8. Jeśli wystąpił błąd podczas aktualizacji hasła, serwer zwraca odpowiedź z kodem 500 i przesyła błąd jako odpowiedź.
9. Jeśli operacja przebiegła pomyślnie, serwer zwraca odpowiedź z kodem 200 i informuje, że dane zostały zmienione.
10. Jeśli hasła się nie zgadzają, serwer zwraca odpowiedź z kodem 400 i informuje, że hasło jest niepoprawne.

Funkcja ”edit_password” umożliwia użytkownikowi zmianę hasła poprzez porównanie podanego hasła z hasłem przechowywanym w bazie danych, a następnie aktualizację hasła na nowe, zaszyfrowane hasło.

```
1  export const edit_password = (req, res) =>{
2      const id = decrypt(req.body.user);
3      console.log(id);
4      const q = "SELECT password FROM farmmed.user WHERE id = ?";
5      db.query(q, [id], (err, data) => {
6          if (err) return res.status(500).json(err);
7          console.log("1");
8          console.log(data[0].password);
9          bcrypt.compare(req.body.password, data[0].password, (err, isMatch) => {
10             if (err) {
11                 return res.status(500).send(err);
12             }
13             console.log("2")
14             if (isMatch) {
15                 const salt = bcrypt.genSaltSync(10);
16                 const hashedPassword = bcrypt.hashSync(req.body.new_password, salt);
17                 const q1 = "UPDATE farmmed.user SET password = ? WHERE id = ?"
18                 db.query(q1, [hashedPassword , id], (err, data)=>{
19                     if (err) return res.status(500).json(err);
20                     else return res.status(200).send("Zmieniono dane");
21                 })
22             } else {
23                 return res.status(400).send("Hasło niepoprawne");
24             }
25         });
26     })
27 }
```

Listing 7. Edycja hasła

7.1.8. Konwersja pliku csv z bazą leków

Poniższe kody^{8 9 10} przedstawia proces konwersji pliku CSV z bazą leków do formatu SQL. Oto opis jego działania:

1. Importowane są wymagane moduły: csv-parser do analizy pliku CSV i fs do operacji na plikach, oraz db z connect.js, który zapewnia połączenie z bazą danych.
2. Funkcja convertCsvToSql jest eksportowana i przyjmuje ścieżkę pliku CSV jako parametr.
3. W funkcji tworzoną jest stała tableName, która określa nazwę tabeli w bazie danych, do której będą wstawiane dane z pliku CSV.
4. Wykonywane jest zapytanie SQL, które usuwa wszystkie wiersze z tabeli tableName. Jeśli wystąpił błąd podczas usuwania, zostaje wyświetlony błąd, połączenie z bazą danych zostaje zakończone, a funkcja kończy działanie.
5. Tworzona jest tablica rows, która będzie przechowywać przetworzone wiersze z pliku CSV.
6. Otwierany jest strumień do odczytu pliku CSV za pomocą funkcji fs.createReadStream(filePath).
7. Plik CSV jest analizowany w formacie strumieniowym za pomocą csv-parser.
8. Dla każdego wiersza odczytanego z pliku CSV:
 - Usuwane są niepotrzebne znaki z wartości w wierszu.
 - Wiersz jest dodawany do tablicy rows.
9. Po zakończeniu odczytu pliku CSV, wykonywane jest zapytanie SQL, które wstawia przetworzone wiersze do tabeli w bazie danych. Wykorzystywane są nagłówki kolumn jako nazwy kolumn, a wartości są wstawiane w postaci tablicy.
10. Jeśli wystąpił błąd podczas wstawiania danych, zostaje wyświetlony błąd, połączenie z bazą danych zostaje zakończone, a funkcja kończy działanie.
11. Jeśli operacja przebiegła pomyślnie, wyświetlany jest komunikat o zakończeniu konwersji pliku CSV na SQL.

12. Wykonane jest kolejne zapytanie SQL, które aktualizuje tabelę drugs_ilosc poprzez dodanie nowych rekordów na podstawie identyfikatorów produktów leczniczych z tabeli tableName. Wiersze są dodawane tylko wtedy, gdy nie ma już wiersza o tym samym identyfikatorze produktu leczniczego w tabeli drugs_ilosc.
13. Jeśli wystąpił błąd podczas aktualizacji tabeli, zostaje wyświetlony błąd.
14. Na końcu funkcji, w kodzie poza funkcją, jest zdefiniowany endpoint /update dla metody POST, który obsługuje przesłanie pliku CSV.
15. W obsłudze tego endpointa:
 - Pobierana jest przesłana plik za pomocą req.file.
 - Pobierana jest ścieżka pliku z file.path.
 - Wywoływana jest funkcja convertCsvToSql z przekazaną ścieżką pliku.
 - Odpowiedź "Przesłany plik CSV jest przetwarzany." jest wysyłana jako odpowiedź na zakończenie żądania.

Poniższe kody pozwalają na przetworzenie pliku CSV zawierającego bazę leków i zapisanie jej w bazie danych w formacie SQL.

```
1 router.post('/update', upload.single('file'), (req, res) => {
2     const file = req.file;
3     const filePath = file.path; // Pobranie ścieżki do pliku
4     // Wywołanie funkcji convertCsvToSql z przekazaną ścieżką pliku
5     convertCsvToSql(filePath);
6
7     // Zwrócenie odpowiedzi na zakończenie żądania
8     res.send('Przesłany plik CSV jest przetwarzany.');
9 });
```

Listing 8. Konwersja pliku csv z bazą leków cz.3

```
1 import csv from 'csv-parser';
2 import fs from 'fs';
3 import { db } from '../connect.js';
4 export const convertCsvToSql = (filePath) => {
5     const tableName = 'farmmed.drugs';
6     db.query(`DELETE FROM ${tableName}`, (error) => {
7         if (error) {
8             console.error('Błąd usuwania wierszy z tabeli:', error);
9             db.end();
10            return;
11        }
12        const rows = [];
13        fs.createReadStream(filePath)
14            .pipe(csv())
15            .on('data', (row) => {
16
17                const modifiedRow = {};
18                for (const key in row) {
19                    const value = row[key].replace(/\//g, ''); // Usunięcie
19                    → niepotrzebnych znaków
20                    modifiedRow[key] = value;
21                }
22                rows.push(modifiedRow);
23            })
24            .on('end', () => {
25                // Wstawianie wierszy do tabeli
26                const headers = ['Identyfikator_Produktu_Leczniczego',
27                                'Nazwa_Produktu_Leczniczego', 'Nazwa_powszechnie_stosowana',
28                                'Rodzaj_preparatu', 'Nazwa_poprzednia_produktu',
29                                'Gatunki_docelowe', 'Okres_karencji', 'Moc',
30                                'Postać_farmaceutyczna', 'Typ_procedury', 'Numer_pozwolenia',
31                                'Ważność_pozwolenia', 'Kod_ATC', 'Podmiot_odpowiedzialny',
32                                'Opakowanie', 'Substancja_czynna', 'Nazwa_wytwórcy',
33                                'Kraj_wytwórcy', 'Nazwa_importera', 'Kraj_importera',
34                                'Podmiot_odpowiedzialny_w_kraju_eksportu', 'Kraj_eksportu',
35                                'Ulotka', 'Charakterystyka'];
36                const values = rows
37                    .map((row) => Object.values(row)[0].split(';'))
38                    .filter((row) => row.length === headers.length)
```

Listing 9. Konwersja pliku csv z bazą leków cz.1

```
1      db.query(
2          `INSERT INTO ${tableName} (${headers.join(',')}) VALUES ?`,
3          [values],
4          (error) => {
5              if (error) {
6                  console.error('Błąd wstawiania wierszy do tabeli:',
7                      error);
8                  db.end();
9              } else {
10                  console.log('Konwersja pliku CSV na SQL
11                      zakończona.');
12                  db.query(
13                      `INSERT INTO drugs_ilosc
14                          (Identyfikator_Produktu_Leczniczego, Ilosc)
15                          SELECT d.Identyfikator_Produktu_Leczniczego, 0
16                          FROM ${tableName} d
17                          LEFT JOIN drugs_ilosc di ON
18                          d.Identyfikator_Produktu_Leczniczego = di.Identyfikator_Produktu_Leczniczego
19                          WHERE di.Identyfikator_Produktu_Leczniczego IS
20                          NULL`,
21                      (error) => {
22                          if (error) {
23                              console.error('Błąd aktualizacji tabeli
24                                  drugs_ilość:', error);
25                          } else {
26                              console.log('Aktualizacja tabeli
27                                  drugs_ilość zakończona.');
28                          }
29                      );
30                  }
31              );
32          );
33      );
34  );
35 );
```

Listing 10. Konwersja pliku csv z bazą leków cz.2

7.1.9. Autoryzacja

Poniższy kod¹¹ przedstawia funkcję autoryzacji, która służy do weryfikacji tokena dostępu. Oto opis jego działania:

1. Funkcja authorize jest eksportowana i przyjmuje parametry req (obiekt żądania) i res (obiekt odpowiedzi).
2. Inicjalizowana jest zmienna token na wartość null.
3. Pobierany jest identyfikator użytkownika (id) z req.body.
4. Wykonuje się deszyfrowanie identyfikatora użytkownika przy pomocy funkcji decrypt.
5. Jeśli id nie jest równy null:
 - Wykonuje się zapytanie SQL, które pobiera token dla użytkownika o danym identyfikatorze.
 - Jeśli wystąpił błąd podczas zapytania, zostaje zwrócony kod odpowiedzi 500 i błąd w formacie JSON.
 - Jeśli istnieją dane zwrócone z zapytania, przypisywany jest token do zmiennej token.
 - W przeciwnym razie zostaje zwrócony kod odpowiedzi 401 i komunikat "Nieprawidłowy token".
 - Jeśli token nie istnieje, zostaje zwrócony kod odpowiedzi 404 i komunikat "Nieprawidłowy token".
6. Następuje próba weryfikacji tokena:
 - Dekoduje się token przy pomocy funkcji jwt.verify, przy czym używany jest sekretny klucz "secretKey".
 - Jeśli weryfikacja przebiegnie pomyślnie, zdekodowany token jest przypisywany do zmiennej decodedToken.
 - Identyfikator użytkownika (id) jest szyfrowany przy pomocy funkcji encrypt i przypisywany do zmiennej id.
 - Zwrocona jest odpowiedź o kodzie 200, zawierająca obiekt JSON z za szyfrowanym identyfikatorem użytkownika (user) i identyfikatorem roli (id_role) zdekodowanego tokenu.
 - Dekodowany token jest dodawany do obiektu żądania req.user.

7. Jeśli wystąpi błąd podczas weryfikacji tokena, zwracany jest kod odpowiedzi 402 i komunikat "Nieprawidłowy token".

Poniższa funkcja służy do autoryzacji żądań, weryfikując token dostępu i dodając zdekodowany token do obiektu żądania.

```
1  export const authorize = (req, res) => {
2      let token = null;
3      const id = req.body;
4      const user = decrypt(id.user)
5      if (id !== null){
6          const q = "SELECT token FROM farmmed.user WHERE id = ?";
7          db.query(q, [user], (error, data) => {
8              if (error) {
9                  return res.status(500).json(error);
10             }
11             if (data.length) {
12                 token = data[0].token;
13             }else return res.status(401).json("Nieprawidłowy token");
14             if (!token) {
15                 // przekierowanie na stronę logowania z informacją, że trzeba się
16                 → zalogować
17                 res.status(404).json("Nieprawidłowy token");
18             }
19             try {
20                 // weryfikujemy token
21                 const decodedToken = jwt.verify(token, "secretKey");
22                 const id = encrypt(decodedToken.id)
23                 res.status(200).json({user: id, id_role: decodedToken.id_role});
24                 // dodajemy zdekodowany token do obiektu żądania
25                 req.user = decodedToken;
26
27             } catch (error) {
28                 res.status(402).json("Nieprawidłowy token");
29             }
30         })
31     }}}
```

Listing 11. Autoryzacja

7.1.10. Funkcje szyfrujące

Poniższy kod¹² przedstawia dwie funkcje szyfrujące (encrypt i decrypt), które wykorzystują bibliotekę node-rsa do szyfrowania i deszyfrowania identyfikatorów. Oto opis działania tych funkcji:

1. Tworzony jest obiekt klucza RSA o długości 512 bitów.
2. Funkcja encrypt przyjmuje identyfikator (id) jako argument.
3. Wykorzystywana jest metoda encrypt obiektu klucza (key.encrypt) w celu za-szyfrowania identyfikatora. Szyfrowanie odbywa się przy pomocy klucza pu-blicznego RSA.
4. Zaszyfrowany identyfikator jest zwracany jako rezultat.
5. Funkcja decrypt przyjmuje zaszyfrowany identyfikator (id) jako argument.
6. Wykorzystywana jest metoda decrypt obiektu klucza (key.decrypt) w celu de-szyfrowania zaszyfrowanego identyfikatora. Deszyfrowanie odbywa się przy po-mocy klucza prywatnego RSA.
7. Odszyfrowany identyfikator jest zwracany jako rezultat.

Funkcje encrypt i decrypt umożliwiają szyfrowanie i deszyfrowanie identyfikatorów, co może być przydatne w celu zabezpieczenia danych przed nieautoryzowanym dosta-tem. Należy jednak pamiętać, że bezpieczeństwo zależy również od sposobu przecho-wywania klucza prywatnego, dlatego ważne jest odpowiednie zabezpieczenie klucza.

```
1 import NodeRSA from 'node-rsa';
2
3 const key = new NodeRSA({ b: 512 });
4 export const encrypt = (id) => {
5     const encrypted = key.encrypt(id, 'base64');
6     return encrypted;
7 }
8 export const decrypt = (id) => {
9     const decrypted = key.decrypt(id, 'utf8');
10    return decrypted;
11 }
```

Listing 12. Funkcje szyfrujące

7.1.11. Tworzenie harmonogramu przyjęć

Poniższy kod¹³ ¹⁴ przedstawia funkcję `create_schedule`, która służy do tworzenia harmonogramu przyjęć. Poniżej znajduje się opis tej funkcji:

1. Funkcja odbiera dane z żądania (`req.body`) zawierające informacje takie jak `startDate`, `endDate`, `startTime`, `endTime`, `doctorId` i `dayOfWeek`.
2. Wyodrębniane są poszczególne komponenty daty (rok, miesiąc, dzień) oraz godziny (godzina, minuty) z danych wejściowych.
3. Tworzona jest data początkowa (`currentDate`) na podstawie podanej daty rozpoczęcia (`startDate`).
4. Tworzona jest data końcowa (`end`) na podstawie podanej daty zakończenia (`endDate`).
5. Inicjalizowana jest pusta tablica `Dates`, która będzie przechowywać daty dla danego dnia tygodnia.
6. Wykonuje się pętla `while`, która sprawdza daty od `currentDate` do `end`.
7. Jeśli dzień tygodnia daty `currentDate` odpowiada podanemu dniowi tygodnia (`daysOfWeek`), to data jest formatowana jako rok, miesiąc i dzień w formacie "YYYYMMDD" i dodawana do tablicy `Dates`.
8. Zmienna `currentDate` jest zwiększana o 1 dzień.
9. Inicjalizowana jest pusta tablica `hours`, która będzie przechowywać godziny.
10. Wykonuje się pętla `while`, która generuje godziny od podanej godziny rozpoczęcia (`startTime`) do podanej godziny zakończenia (`endTime`).
11. Godzina jest formatowana jako "HH:MM" i dodawana do tablicy `hours`.
12. Aktualne minuty są zwiększane o 20, a jeśli przekraczają 60, są odpowiednio modyfikowane i zwiększana jest aktualna godzina.
13. Inicjalizowane są puste tablice `result`, która będzie przechowywać dane harmonogramu.
14. Funkcja `fetchData` jest zdefiniowana jako funkcja asynchroniczna, która będzie odpowiedzialna za pobieranie danych z bazy danych.

15. Wewnątrz funkcji fetchData wykonywana jest pętla for, która przechodzi przez wszystkie daty w tablicy Dates.
16. Tworzony jest zapytanie SQL, które sprawdza, czy dla danej daty istnieją już wpisy w tabeli schedule.
17. Zapytanie jest wykonywane za pomocą funkcji db.query (zakładam, że jest to funkcja wykonująca zapytanie do bazy danych).
18. Jeśli zapytanie zwraca pustą tablicę danych, oznacza to, że dla tej daty nie istnieje harmonogram.
19. W pętli for dla każdej godziny w tablicy hours tworzona jest wartość do wstawienia do tabeli schedule, składająca się z doctorId, date i time.
20. Wartość jest dodawana do tablicy result.
21. Funkcja fetchData zawiera wywołanie funkcji fetchData2, która będzie odpowiedzialna za wstawienie danych do bazy danych.
22. Jeśli wystąpił błąd podczas wykonywania funkcji fetchData lub fetchData2, zwracany jest odpowiedni status i komunikat błędu.
23. Funkcja fetchData2 jest zdefiniowana jako funkcja asynchroniczna.
24. Jeśli tablica result jest pusta, oznacza to, że nie ma harmonogramu do dodania.
25. W przeciwnym razie tworzone jest zapytanie SQL, które wstawia dane z tablicy result do tabeli schedule.
26. Zapytanie jest wykonywane za pomocą funkcji db.query.
27. Jeśli wystąpił błąd podczas wykonywania zapytania, zwracany jest odpowiedni status i komunikat błędu.
28. Jeśli zapytanie zostało wykonane poprawnie, zwracany jest odpowiedni status i komunikat potwierdzający dodanie harmonogramu.

```
1  export const create_schedule = (req, res) => {
2    const { startDate, endDate, startTime, endTime, doctorId, dayOfWeek } =
3      req.body;
4    let daysOfWeek = parseInt(dayOfWeek);
5    const [forYear, forMonth, forDay] = startDate.split('-').map(Number);
6    const [toYear, toMonth, toDay] = endDate.split('-').map(Number);
7    let currentDate = new Date(forYear, forMonth - 1, forDay);
8    const end = new Date(toYear, toMonth - 1, toDay);
9    const Dates = [];
10   while (currentDate <= end) {
11     if (currentDate.getDay() === daysOfWeek) {
12       const year = currentDate.getFullYear();
13       const month = (currentDate.getMonth() + 1).toString().padStart(2, '0');
14       const day = currentDate.getDate().toString().padStart(2, '0');
15       Dates.push(`-${year}-${month}-${day}`);
16     }
17     currentDate.setDate(currentDate.getDate() + 1);
18   }
19   const [initialHour, initialMinutes] = startTime.split(':').map(Number);
20   const [finalHour, finalMinutes] = endTime.split(':').map(Number);
21   const hours = [];
22   let currentHours = initialHour;
23   let currentMinutes = initialMinutes;
24   while (currentHours < finalHour || (currentHours === finalHour &&
25     currentMinutes < finalMinutes)) {
26     const hour = currentHours.toString().padStart(2, '0');
27     const minutes = currentMinutes.toString().padStart(2, '0');
28     hours.push(`-${hour}:${minutes}`);
29     currentMinutes += 20;
30     if (currentMinutes >= 60) {
31       currentMinutes -= 60;
32       currentHours++;
33     }
34   }
35   const result = [];
```

Listing 13. Tworzenie harmonogramu przyjęć cz.1

```
1  const fetchData = async () => {
2    try {
3      for (let d = 0; d < Dates.length; d++) {
4        const q = 'SELECT * FROM farmmed.schudle WHERE date = ?';
5        await new Promise((resolve, reject) => {
6          db.query(q, Dates[d], (err, data) => {
7            if (err) {
8              console.error(err);
9              reject(err);
10           } else {
11             if (data.length === 0) {
12               for (let t = 0; t < hours.length; t++) {
13                 const value = [doctorId, Dates[d], hours[t]];
14                 result.push(value);
15               }
16             }
17             resolve();
18           }
19         });
20       });
21     }
22     await fetchData2();
23   } catch (error) {
24     return res.status(500).send(error);
25   }
26 };
27 const fetchData2 = async () => {
28   if (result.length === 0) {
29     return res.status(200).send('Brak harmonogramu do dodania.');
30   } else {
31     const q = 'INSERT INTO `farmmed`.`schudle` (`doctor_id`, `date`, `time`)
32     VALUES ?';
33     db.query(q, [result], (err, data) => {
34       if (err) {
35         console.error(err);
36         return res.status(500).send(err);
37       }
38       return res.status(200).send('Dodano harmonogram.');
39     });
40   }
41   fetchData();
42};
```

Listing 14. Tworzenie harmonogramu przyjęć cz.2

7.1.12. Funkcje wyświetlające wolne terminy przyjęć

Poniższy kod¹⁵ zawiera dwie funkcje: view_date_these_dates i view_time_these_dates. Oto opis tych funkcji:

1. view_date_these_dates: Ta funkcja służy do wyświetlania unikalnych dat, które mają dostępne wolne terminy przyjęć dla konkretnego lekarza o podanym identyfikatorze (req.body.id).

- Wykonuje się zapytanie SQL, które zwraca unikalne daty z tabeli schedule, gdzie identyfikator lekarza (doctor_id) jest równy podanemu identyfikatorowi (req.body.id), a pole user_id jest puste (null).
- Wynik zapytania jest zwracany jako odpowiedź HTTP z kodem statusu 200.

2. view_time_these_dates: Ta funkcja służy do wyświetlania dostępnych godzin na konkretną datę dla danego lekarza o podanym identyfikatorze (req.body.id) i dacie (req.body.date).

- Wykonuje się zapytanie SQL, które zwraca identyfikator (id) i godzinę (time) z tabeli schedule, gdzie identyfikator lekarza (doctor_id) jest równy podanemu identyfikatorowi (req.body.id), data (date) jest równa podanej dacie (req.body.date), a pole user_id jest puste (null).
- Wynik zapytania jest zwracany jako odpowiedź HTTP z kodem statusu 200.

```
1  export const view_date_these_dates = (req, res) => {
2    const q = 'SELECT DISTINCT DATE_ADD(date, INTERVAL 1 DAY) AS date FROM
3      farmmmed.schudle WHERE doctor_id = ? AND user_id IS NULL;';
4    db.query(q, [req.body.id], (err, data) => {
5      if (err) {
6        console.error(err);
7        return res.status(500).send(err);
8      }
9      return res.status(200).send(data);
10    });
11
12  export const view_time_these_dates = (req, res) => {
13    const q = 'SELECT id, time FROM farmmmed.schudle WHERE doctor_id = ? AND date =
14      ? AND user_id IS NULL;';
15    db.query(q, [req.body.id, req.body.date], (err, data) => {
16      if (err) {
17        console.error(err);
18        return res.status(500).send(err);
19      }
20      return res.status(200).send(data);
21    });
22  };
```

Listing 15. Funkcje wyświetlające wolne terminy przyjęć

7.1.13. Wyświetlanie pacjentów do przyjęcie

Powyższy kod¹⁶ zawiera funkcję displaying_patients_for_admission, która służy do wyświetlania listy pacjentów do przyjęcia dla danego lekarza na bieżącą datę.

Oto opis funkcji:

- Dekodowanie identyfikatora użytkownika (req.body.user) za pomocą funkcji decrypt.
- Wykonanie zapytania SQL, które pobiera imię i nazwisko pacjenta oraz godzinę przyjęcia z tabeli schedule oraz łączy się z tabelą user w celu pobrania pełnego imienia i nazwiska pacjenta.
- Zapytanie wyszukuje wiersze, w których identyfikator lekarza (doctor_id) jest równy podanemu identyfikatorowi lekarza, pole user_id nie jest puste (null) i data (date) jest równa bieżącej dacie (CURDATE()).
- Wynik zapytania zawiera nazwisko i imię pacjenta (name) oraz godzinę przyjęcia (time).
- Wynik zapytania jest zwracany jako odpowiedź HTTP z kodem statusu 200.

```
1  export const displaying_patients_for_admission = (req, res) => {
2    const id = decrypt(req.body.user);
3    const q = `SELECT CONCAT(u.first_name, ' ', u.last_name) AS name, s.time
4      FROM farmmed.schedule s
5      INNER JOIN farmmed.user u ON s.user_id = u.id
6      WHERE doctor_id = ? AND user_id IS NOT NULL AND s.date = CURDATE();`;
7    db.query(q, [user], (err, data) => {
8      if (err) {
9        console.error(err);
10       return res.status(500).send(err);
11     }
12     return res.status(200).send(res.data);
13   });
14};
```

Listing 16. Wyświetlanie pacjentów do przyjęcie

7.2. Baza Danych

7.2.1. Tabela użytkowników

```
1 CREATE TABLE `user` (
2     `id` int unsigned NOT NULL AUTO_INCREMENT,
3     `first_name` varchar(45) NOT NULL,
4     `last_name` varchar(45) NOT NULL,
5     `date_of_birth` date DEFAULT NULL,
6     `PESEL` decimal(11,0) unsigned zerofill DEFAULT NULL,
7     `email` varchar(100) NOT NULL,
8     `password` varchar(200) NOT NULL,
9     `id_role` int unsigned NOT NULL,
10    `id_spec` int unsigned DEFAULT '0',
11    `token` varchar(255) DEFAULT NULL,
12    PRIMARY KEY (`id`),
13    UNIQUE KEY `id_UNIQUE` (`id`),
14    UNIQUE KEY `email_UNIQUE` (`email`),
15    UNIQUE KEY `PESEL_UNIQUE` (`PESEL`),
16    KEY `role_idx` (`id_role`),
17    KEY `specjalizacja_idx` (`id_spec`),
18    CONSTRAINT `role` FOREIGN KEY (`id_role`) REFERENCES `roles` (`id_role`),
19    CONSTRAINT `specjalizacja` FOREIGN KEY (`id_spec`) REFERENCES `specjalizacja`  
    ↳ (`id_spec`)
20 ) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=utf8mb4  
    ↳ COLLATE=utf8mb4_0900_ai_ci
```

Listing 17. Tabela użytkowników

7.2.2. Tabela roli

```
1 CREATE TABLE `roles` (
2   `id_role` int unsigned NOT NULL AUTO_INCREMENT,
3   `name` varchar(45) NOT NULL,
4   PRIMARY KEY (`id_role`),
5   UNIQUE KEY `id_UNIQUE` (`id_role`),
6   UNIQUE KEY `name_UNIQUE` (`name`)
7 ) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb4
8 → COLLATE=utf8mb4_0900_ai_ci;
9 INSERT INTO `roles` VALUES (1,'Admin'),(4,'Aptekarz'),(3,'Doktor'),(2,'Pacjent');
```

Listing 18. Tabela roli

7.2.3. Tabela specjalizacji

```
1 CREATE TABLE `specjalizacja` (
2     `id_spec` int unsigned NOT NULL,
3     `name` varchar(45) NOT NULL,
4     PRIMARY KEY (`id_spec`),
5 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
6 INSERT INTO `specjalizacja` VALUES
7     (0,''),(1,'Alergologia'),(2,'Anestezjologia'),(3,'Angiologia'),(4,'Audiologia i
8     foniatria'),(5,'Balneologia'),(6,'Chirurgia dziecięca'),(7,'Chirurgia klatki
9     piersiowej'),(8,'Chirurgia naczyniowa'),(9,'Chirurgia ogólna'),(10,'Chirurgia
11     onkologiczna'),(11,'Chirurgia plastyczna'),(12,'Chirurgia szczekowo-twarzowa'),(13,'Choroby
14     płuc'),(14,'Choroby płuc dzieci'),(15,'Choroby wewnętrzne'),(16,'Choroby
17     zakaźne'),(17,'Dermatologia'),(18,'Diabetologia'),(19,'Diagnostyka
18     laboratoryjna'),(20,'Endokrynologia'),(21,'Endokrynologia ginekologiczna i
19     rozrodczość'),(22,'Endokrynologia i diabetologia
20     dziecięca'),(23,'Epidemiologia'),(24,'Farmakologia
21     kliniczna'),(25,'Gastroenterologia'),(26,'Gastroenterologia dziecięca'),(27,'Genetyka
22     kliniczna'),(28,'Geriatria'),(29,'Ginekologia
23     onkologiczna'),(30,'Hematologia'),(31,'Hipertensjologia'),(32,'Immunologia
24     kliniczna'),(33,'Intensywna terapia'),(34,'Kardiochirurgia'),(35,'Kardiologia'),(36,'Kardiologia
25     dziecięca'),(37,'Medycyna lotnicza'),(38,'Medycyna morska i tropikalna'),(39,'Medycyna
26     nuklearna'),(40,'Medycyna paliatywna'),(41,'Medycyna pracy'),(42,'Medycyna
27     ratunkowa'),(43,'Medycyna rodzinna'),(44,'Medycyna sądowa'),(45,'Medycyna
28     sportowa'),(46,'Mikrobiologia lekarska'),(47,'Nefrologia'),(48,'Nefrologia
29     dziecięca'),(49,'Neonatologia'),(50,'Neurochirurgia'),(51,'Neurologia'),(52,'Neurologia
30     dziecięca'),(53,'Neuropatologia'),(54,'Okulistyka'),(55,'Onkologia i hematologia
31     dziecięca'),(56,'Onkologia kliniczna'),(57,'Ortopedia i traumatologia narządu
32     ruchu'),(58,'Otorynolaryngologia'),(59,'Otorynolaryngologia
33     dziecięca'),(60,'Patomorfologia'),(61,'Pediatria'),(62,'Pediatria
34     metaboliczna'),(63,'Perinatologia'),(64,'Położnictwo i
35     ginekologia'),(65,'Psychiatria'),(66,'Psychiatria dzieci i młodzieży'),(67,'Radiologia i
36     diagnostyka obrazowa'),(68,'Radioterapia onkologiczna'),(69,'Rehabilitacja
37     medyczna'),(70,'Reumatologia'),(71,'Seksuologia'),(72,'Toksykologia
38     kliniczna'),(73,'Transfuzjologia kliniczna'),(74,'Transplantologia
39     kliniczna'),(75,'Urologia'),(76,'Urologia dziecięca'),(77,'Zdrowie publiczne');
```

Listing 19. Tabela specjalizacji

7.2.4. Tabela leków

```
1 CREATE TABLE `drugs` (
2     `Identyfikator_Produktu_Leczniczego` bigint NOT NULL,
3     `Nazwa_Produktu_Leczniczego` text NOT NULL,
4     `Nazwa_powszechnie_stosowana` text,
5     `Rodzaj_preparatu` text NOT NULL,
6     `Nazwa_poprzednia_produkту` text,
7     `Gatunki_docelowe` text,
8     `Okres_karencki` text,
9     `Moc` text,
10    `Postać_farmaceutyczna` text NOT NULL,
11    `Typ_procedury` text NOT NULL,
12    `Numer_pozwolenia` text,
13    `Ważność_pozwolenia` text,
14    `Kod_ATC` text,
15    `Podmiot_odpowiedzialny` text,
16    `Opakowanie` text,
17    `Substancja_czynna` text,
18    `Nazwa_wytwórcy` text,
19    `Kraj_wytwórcy` text,
20    `Nazwa_importera` text,
21    `Kraj_importera` text,
22    `Podmiot_odpowiedzialny_w_kraju_eksportu` text,
23    `Kraj_eksportu` text,
24    `Ulotka` text,
25    `Charakterystyka` text,
26    PRIMARY KEY (`Identyfikator_Produktu_Leczniczego`)
27 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
→   COMMENT='
```

Listing 20. Tabela leków

7.2.5. Tabela ilości leków w aptece

```
1 CREATE TABLE `drugs_ilosc` (
2   `Identyfikator_Produktu_Leczniczego` int NOT NULL,
3   `Ilosc` int unsigned DEFAULT '0',
4   PRIMARY KEY (`Identyfikator_Produktu_Leczniczego`)
5 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Listing 21. Tabela ilości leków w aptece

7.2.6. Tabela kartotek

```
1 CREATE TABLE `medical_file` (
2   `id` int unsigned NOT NULL AUTO_INCREMENT,
3   `id_user` int unsigned NOT NULL,
4   `id_doctor` int unsigned NOT NULL,
5   `Date` date NOT NULL DEFAULT (curdate()),
6   `cel_wizyty` text NOT NULL,
7   `objawy` blob NOT NULL,
8   `wynik_badania` blob NOT NULL,
9   `Zalecenia` blob NOT NULL,
10  `Termin_kolejnej_wizyty` text NOT NULL,
11  PRIMARY KEY (`id`),
12  UNIQUE KEY `id_UNIQUE` (`id`),
13  KEY `user` (`id_user`),
14  KEY `doctor` (`id_doctor`),
15  CONSTRAINT `doctor` FOREIGN KEY (`id_doctor`) REFERENCES `user` (`id`),
16  CONSTRAINT `user` FOREIGN KEY (`id_user`) REFERENCES `user` (`id`)
17 ) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8mb4
→  COLLATE=utf8mb4_0900_ai_ci
```

Listing 22. Tabela kartotek

7.2.7. Tabela z receptami

```
1 CREATE TABLE `recepty` (
2     `id` int unsigned NOT NULL AUTO_INCREMENT,
3     `id_user` int unsigned NOT NULL,
4     `id_doctor` int unsigned NOT NULL,
5     `Date` date NOT NULL DEFAULT (curdate()),
6     `key` varchar(4) DEFAULT NULL,
7     `validity_date` date NOT NULL DEFAULT ((curdate() + interval 30 day)),
8     PRIMARY KEY (`id`),
9     UNIQUE KEY `id_UNIQUE` (`id`),
10    KEY `user_idx` (`id_user`),
11    KEY `doctor_idx` (`id_doctor`),
12    CONSTRAINT `doctor_fk` FOREIGN KEY (`id_doctor`) REFERENCES `user` (`id`),
13    CONSTRAINT `user_fk` FOREIGN KEY (`id_user`) REFERENCES `user` (`id`)
14 ) ENGINE=InnoDB AUTO_INCREMENT=16 DEFAULT CHARSET=utf8mb4
→   COLLATE=utf8mb4_0900_ai_ci
```

Listing 23. Tabela z receptami

7.2.8. Tabela z lekami z recepty

```
1 CREATE TABLE `recepta_leki` (
2     `id` int unsigned NOT NULL AUTO_INCREMENT,
3     `id_recepty` int unsigned NOT NULL,
4     `id_leku` int NOT NULL,
5     `opakowanie` varchar(100) NOT NULL,
6     `dawkowanie` varchar(100) NOT NULL,
7     `status` int NOT NULL DEFAULT '0',
8     PRIMARY KEY (`id`),
9     UNIQUE KEY `id_UNIQUE` (`id`),
10    KEY `recepta` (`id_recepty`),
11    CONSTRAINT `recepta` FOREIGN KEY (`id_recepty`) REFERENCES `recepty` (`id`)
12 ) ENGINE=InnoDB AUTO_INCREMENT=11 DEFAULT CHARSET=utf8mb4
→   COLLATE=utf8mb4_0900_ai_ci
```

Listing 24. Tabela z lekami z recepty

7.2.9. Trigger do tworzenia kodu recepty

```
1 CREATE DEFINER=`root`@`localhost` TRIGGER `set_key_value` BEFORE INSERT ON
→ `recepty` FOR EACH ROW BEGIN
2     SET NEW.`key` = LPAD(FLOOR(RAND() * 10000), 4, '0');
3 END
```

Listing 25. Trigger do tworzenia kodu recepty

7.2.10. Tabela z harmonogramem wizyt lekarskich

```
1 CREATE TABLE `schudle` (
2     `id` int NOT NULL AUTO_INCREMENT,
3     `doctor_id` int unsigned NOT NULL,
4     `user_id` int unsigned DEFAULT NULL,
5     `date` date NOT NULL,
6     `time` time NOT NULL,
7     PRIMARY KEY (`id`),
8     KEY `uzytkownik_idx` (`user_id`),
9     KEY `doktor_idx` (`doctor_id`),
10    CONSTRAINT `doktor` FOREIGN KEY (`doctor_id`) REFERENCES `user` (`id`),
11    CONSTRAINT `uzytkownik` FOREIGN KEY (`user_id`) REFERENCES `user` (`id`)
12 ) ENGINE=InnoDB AUTO_INCREMENT=73 DEFAULT CHARSET=utf8mb4
→ COLLATE=utf8mb4_0900_ai_ci
```

Listing 26. Tabela z harmonogramem wizyt lekarskich

7.2.11. Tabela z zamówieniami

```
1 CREATE TABLE `order` (
2   `id` int unsigned NOT NULL AUTO_INCREMENT,
3   `id_user` int unsigned NOT NULL,
4   `data_zamowienia` date NOT NULL DEFAULT (curdate()),
5   `status` int NOT NULL,
6   PRIMARY KEY (`id`),
7   UNIQUE KEY `id_UNIQUE` (`id`),
8   KEY `id_user_idx` (`id_user`),
9   CONSTRAINT `id_user` FOREIGN KEY (`id_user`) REFERENCES `user` (`id`)
10 ) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8mb4
→   COLLATE=utf8mb4_0900_ai_ci
```

Listing 27. Tabela z zamówieniami

7.2.12. Tabela ze statusami zamówień

```
1 CREATE TABLE `status` (
2   `id` int NOT NULL,
3   `status` varchar(45) NOT NULL
4 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
5 INSERT INTO `status` VALUES (0, 'Oczekuje na realizację'), (1, 'Gotowe do
→  odbioru'), (2, 'Odrzucone'), (3, 'Odebrane');
```

Listing 28. Tabela ze statusami zamówień

7.2.13. Tabela z lekami z zamówień

```
1 CREATE TABLE `order_drug` (
2   `id` int unsigned NOT NULL AUTO_INCREMENT,
3   `id_order` int unsigned NOT NULL,
4   `id_drug` bigint NOT NULL,
5   PRIMARY KEY (`id`),
6   UNIQUE KEY `id_UNIQUE` (`id`),
7   KEY `drug_idx` (`id_drug`),
8   KEY `new_drug_idx` (`id_drug`),
9   KEY `order` (`id_order`),
10  CONSTRAINT `drug` FOREIGN KEY (`id_drug`) REFERENCES `drugs`
11    (`Identyfikator_Produktu_Leczniczego`),
11  CONSTRAINT `order` FOREIGN KEY (`id_order`) REFERENCES `order` (`id`)
12 ) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8mb4
12   COLLATE=utf8mb4_0900_ai_ci
```

Listing 29. Tabela z lekami z zamówień

7.3. Frontend

7.3.1. Walidacja numeru PESEL i funkcja autodate

Poniższym kod³⁰ zawiera funkcje służące do walidacji numeru PESEL oraz funkcję autoDate, która na podstawie numeru PESEL automatycznie oblicza datę urodzenia. Oto opis tych funkcji:

1. Funkcja isValidPesel przyjmuje numer PESEL (pesel) jako argument.
2. Wykorzystuje wyrażenie regularne ($/[0-9]11$/$) w celu sprawdzenia, czy numer PESEL składa się z 11 cyfr.
3. Jeśli numer PESEL nie spełnia tego warunku, funkcja zwraca wartość false.
4. Jeśli numer PESEL składa się z 11 cyfr, następuje obliczenie sumy kontrolnej PESEL.
5. Każda cyfra numeru PESEL jest mnożona przez odpowiedni współczynnik i sumowana.
6. Obliczony wynik jest poddawany operacji modulo 10.
7. Jeśli wynik modulo 10 wynosi 0, to suma kontrolna powinna być równa 0.
8. Jeśli wynik modulo 10 nie jest równy 0, to suma kontrolna powinna być równa $(10 - \text{wynik modulo 10})$.
9. Jeśli suma kontrolna nie spełnia warunku, funkcja zwraca wartość false.
10. W przeciwnym razie, funkcja zwraca wartość true, co oznacza, że numer PESEL jest poprawny.

Funkcja autoDate przyjmuje numer PESEL (pesel) jako argument.

1. Na podstawie cyfr w numerze PESEL obliczany jest rok, miesiąc i dzień urodzenia.
2. W zależności od wartości miesiąca, dodawane są odpowiednie liczby do roku, aby uzyskać poprawną wartość.
3. Tworzony jest obiekt Date z obliczonymi wartościami roku, miesiąca i dnia.
4. Wywoływana jest metoda toISOString(), aby uzyskać datę urodzenia w formacie ISO (np. "2023-06-05").

5. Przy użyciu substring(0, 10) pobierane są tylko pierwsze 10 znaków, które reprezentują datę w formacie "YYYY-MM-DD".
6. Ostatecznie, funkcja zwraca obliczoną datę urodzenia.

Dodatkowo, w kodzie znajduje się funkcja handlePeselChange, która jest wywoływana, gdy zmienia się wartość pola tekstowego z numerem PESEL. Funkcja sprawdza, czy podany numer PESEL jest poprawny i ma długość 11 cyfr. Jeśli tak, wywołuje funkcję autoDate w celu obliczenia daty urodzenia na podstawie numeru PESEL. Następnie aktualizuje stan aplikacji, dodając wartość numeru PESEL i obliczoną datę urodzenia do signupState. Jeśli numer PESEL jest nieprawidłowy, ustawia błąd w stanie aplikacji.

```
1  const isValidPesel = (pesel) => {
2    if (!/^([0-9]{11})$/.test(pesel)) return false;
3    const digits = pesel.split('').map(Number);
4    const checksum =
5      (1 * digits[0] + 3 * digits[1] + 7 * digits[2] + 9 * digits[3] + 1 * digits[4] + 3 * digits[5]
6       + 7 * digits[6] + 9 * digits[7] + 1 * digits[8] + 3 * digits[9]) % 10;
7    if (checksum === 0) return digits[10] === 0;
8    else return digits[10] === (10 - checksum);
9  }
10 const autoDate = (pesel) => {
11  let year = parseInt(pesel.substring(0,2));
12  let month = parseInt(pesel.substring(2,4));
13  let day = parseInt(pesel.substring(4,6));
14  if (month > 80) {
15    year += 1800;
16    month -= 80;
17  } else if (month > 60) {
18    year += 2200;
19    month -= 60;
20  } else if (month > 40) {
21    year += 2100;
22    month -= 40;
23  } else if (month > 20) {
24    year += 2000;
25    month -= 20;
26  } else year += 1900;
27  const dateOfBirth = new Date(year, month-1, day+1).toISOString().substring(0,10);
28  return dateOfBirth;
29};
30 const handlePeselChange = (e) => {
31  const pesel = e.target.value;
32  if (isValidPesel(pesel) && pesel.length === 11){
33    setErr(null);
34    const dateOfBirth = autoDate(pesel);
35    setSignupState({...signupState,[e.target.id]:e.target.value, date_of_birth: dateOfBirth});
36  }else if (!isValidPesel(pesel) && pesel.length === 11){
37    setErr('Nieprawidłowy numer PESEL');
38  }else setSignupState({...signupState,[e.target.id]:e.target.value});
};
```

Listing 30. Walidacja numeru PESEL i funkcja autodate

7.3.2. Funkcja przekazująca plik csv do backend'u

Poniższy kod³¹ zawiera funkcję handleFile, która obsługuje przesyłanie pliku CSV do backend'u. Oto opis tej funkcji:

1. Funkcja jest wywoływana, gdy użytkownik wybiera plik za pomocą elementu `<input type="file">` i następuje zmiana wartości tego elementu.
2. Pobierany jest wybrany plik zdarzenia (`e.target.files[0]`).
3. Tworzony jest obiekt `FormData`, który będzie zawierał plik.
4. Przy użyciu metody `append` dodawany jest plik do obiektu `FormData` pod kluczem 'file'.
5. Wykorzystując bibliotekę `axios`, wysyłane jest żądanie POST na adres '`http://127.0.0.1:8800/`'.
6. Do żądania dodawane są odpowiednie nagłówki, w tym 'Content-Type': 'multipart/form-data', który informuje serwer, że przesyłany jest formularz zawierający plik.
7. Gdy żądanie jest udane, wyświetlany jest komunikat w konsoli (`console.log('Plik został przesłany na backend', response)`).
8. Ustawiany jest stan aplikacji `setErr` z informacją, że plik został dodany i strona będzie odświeżana.
9. Po 4 sekundach strona zostaje odświeżona (`window.location.reload()`), aby zaktualizować dane.
10. Jeśli wystąpi błąd podczas przesyłania pliku, wyświetlany jest komunikat w konsoli (`console.error('Błąd podczas przesyłania pliku', error)`).

```
1  const handleFile = (e) => {
2      const file = e.target.files[0];
3      console.log(file);
4      const formData = new FormData();
5      formData.append('file', file);
6      console.log(formData.get('file'));
7      axios
8          .post('http://127.0.0.1:8800/api/drug/update', formData, {
9              headers: {
10                  'Content-Type': 'multipart/form-data',
11              },
12          })
13          .then((response) => {
14              console.log('Plik został przesłany na backend', response);
15              setErr(`Dodano, strona za chwilę się odświeży`);
16              setTimeout(() => {
17                  window.location.reload();
18              }, 4000);
19          })
20          .catch((error) => {
21              console.error('Błąd podczas przesyłania pliku', error);
22              // Dodaj tutaj kod obsługujący błąd
23          });
24      };

```

Listing 31. Funkcja przekazująca plik csv do backend'u

8. Testy i ich wyniki

Przeprowadzone testy przebiegły pomyślnie i można je pojedynczo znaleźć: [**tutaj**](#), oraz wszystkie te testy połączone w jeden film: [**tutaj**](#).

Bibliografia

- [1] mgr inż. Dawid Kotlarski. *Dokumentacja techniczna projektu - Szablon*. 2022.
- [2] *Film z testami*. URL: <https://www.youtube.com/watch?v=Wl3RD-9mBJo>.
- [3] *GPT 3.5 - OpenAI*. URL: <https://chat.openai.com>.
- [4] *Informacje o JavaScript*. URL: <https://kcmobile.pl/slownik-pojec/javascript-co-to/>.
- [5] *Informacje o Node.js*. URL: <https://pl.wikipedia.org/wiki/Node.js>.
- [6] *Informacje o React*. URL: <https://pl.reactjs.org/>.
- [7] *Informacje o SQL*. URL: <https://www.oracle.com/pl/database/what-is-a-relational-database/>.
- [8] *Informacje o tabelach w LATEX*. URL: <http://bitly.pl/hJQIl>.
- [9] *Informacje o Tailwind*. URL: <https://tailwindcss.com/>.
- [10] mgr inż. Nikodem Bulanda. *Dokumentacja projektu ZPI - Szablon*. 2021.
- [11] *Pojedyncze testy*. URL: <https://github.com/antonio-23/farmmed-website#readme>.
- [12] *Tworzenie diagramów*. URL: <https://app.diagrams.net>.
- [13] *Tworzenie wizualizacji strony*. URL: <https://www.figma.com>.

Spis rysunków

4.1.	Diagram przypadków użycia	9
4.2.	Panel lekarza	10
4.3.	Panel aptekarza	11
4.4.	Panel administratora	12
4.5.	Diagram ERD	15

Spis tabel

5.1. Rejestracja pacjenta	16
5.2. Logowanie	17
5.3. Panel pacjenta w przychodni	18
5.4. Panel pacjenta w aptece	19
5.5. Dane pacjenta	20
5.6. Rejestracja na wizytę	20
5.7. Kartoteka pacjenta	21
5.8. Recepty	21
5.9. Apteka	21
5.10. Zamówienia	22
5.11. Składanie zamówień	22
5.12. Koszyk	23
5.13. Przychodnia	23
5.14. Panel lekarza	24
5.15. Dane lekarza/aptekarza	24
5.16. Harmonogram przyjęć	25
5.17. Kartoteki pacjentów	25
5.18. Wystaw receptę	26
5.19. Panel aptekarza	27
5.20. Złożone zamówienia	28
5.21. Baza leków - Aptekarz	28
5.22. Recepta pacjenta	29
5.23. Panel administratora	30
5.24. Zarządzanie kontami	30
5.25. Edycja harmonogramów	31
5.26. Baza leków - Administrator	31

List of Listings

1.	Logowanie	39
2.	Rejestracja	41
3.	Wyszukiwanie leków	42
4.	Dodawanie i odczyt kartoteki pacjenta	44
5.	Wyświetlanie recept	46
6.	Wyszukiwanie recepty przez aptekarza i weryfikacja kodu	48
7.	Edycja hasła	50
8.	Konwersja pliku csv z bazą leków cz.3	52
9.	Konwersja pliku csv z bazą leków cz.1	53
10.	Konwersja pliku csv z bazą leków cz.2	54
11.	Autoryzacja	56
12.	Funkcje szyfrujące	57
13.	Tworzenie harmonogramu przyjęć cz.1	60
14.	Tworzenie harmonogramu przyjęć cz.2	61
15.	Funkcje wyświetlające wolne terminy przyjęć	63
16.	Wyświetlanie pacjentów do przyjęcie	64
17.	Tabela użytkowników	65
18.	Tabela roli	66
19.	Tabela specjalizacji	67
20.	Tabela leków	68
21.	Tabela ilości leków w aptece	69
22.	Tabela kartotek	69
23.	Tabela z receptami	70
24.	Tabela z lekami z recepty	70
25.	Trigger do tworzenia kodu recepty	71
26.	Tabela z harmonogramem wizyt lekarskich	71
27.	Tabela z zamówieniami	72
28.	Tabela ze statusami zamówień	72
29.	Tabela z lekami z zamówień	73
30.	Walidacja numeru PESEL i funkcja autodate	76
31.	Funkcja przekazująca plik csv do backend'u	78

Raport

AKADEMIA NAUK STOSOWANYCH W NOWYM SĄCZU Wydział Nauk Inżynierijnych, Katedra informatyki					
Przedmiot:	Bazy danych – laboratorium, mgr inż. Nikodem Bulanda				
Grupa:	IS-2(s)P1	Sprawozdanie:	1	Data oddania:	12.04.2023
Imię i nazwisko:	Antoni Garczyński, Marcin Gonciarz				

1 Wykonane zadania

- Stworzenie frontend'u strony głównej, logowania, rejestracji i przypomnienia hasła.
- Postawienie bazy danych MySQL i stworzenie pierwszej tabeli.
- Wykonanie backend'u logowania i rejestracji.
- Poprawienie scenariuszy logowania i rejestracji.

2 Zadania na kolejny tydzień

- Stworzenie frontend'u i backend'u dla administratora.
- Stworzenie zabezpieczonych ścieżek.

3 Bilans realizacji

	Przewidziany czas na realizację	Rzeczywisty czas realizacji
Stworzenie frontend'u strony głównej, logowania, rejestracji i przypomnienia hasła.	6h	8h
Postawienie bazy danych MySQL i podłączenie do backendu.	2h	2h
Wykonanie backend'u logowania i rejestracji.	4h	4h
Poprawienie scenariuszy logowania i rejestracji.	0.5h	0.5h

4 Pytania, problemy, eskalacje

- Pytanie odnośnie połączenie gałęzi w gicie?
- Problem z wykonaniem płynnego przejścia pomiędzy stronami na stronie głównej.

Raport

AKADEMIA NAUK STOSOWANYCH W NOWYM SĄCZU Wydział Nauk Inżynierijnych, Katedra informatyki					
Przedmiot:	Bazy danych – laboratorium, mgr inż. Nikodem Bulanda				
Grupa:	IS-2(s)P1	Sprawozdanie:	2	Data oddania:	19.04.2023
Imię i nazwisko:	Antoni Garczyński, Marcin Gonciarz				

1 Wykonane zadania

- Dodanie komunikatów błędów przy rejestracji:
 1. Nieprawidłowy numer PESEL.
 2. Adresy email nie są identyczne.
 3. Hasła nie są identyczne.
 4. Użytkownik z podanym Email już istnieje.
 5. Użytkownik z podanym numerem PESEL już istnieje.
 6. Hasło musi składać się z co najmniej 8 znaków, w tym co najmniej jednej małej i jednej dużej litery, jednej cyfry oraz jednego znaku specjalnego z zestawu:
! @ # \$ % ^ & * () _ + - = { } [] | \ : ; " < > , . ? /
- Dodanie komunikatów błędów przy logowaniu:
 1. Niepoprawny login lub hasło.
- Dodanie komunikatów błędów przy przypomnieniu hasła:
 1. Nieprawidłowy email.
 2. Użytkownik o podanym adresie email nie istnieje.
- Wykonanie validacji PESEL.
- Ustalenie warunków hasła.
- Pobieranie daty urodzenia z numeru PESEL.
- Dodanie cookies.
- Dodanie tokenu JWT (tworzenie przy logowaniu i unieważnienie przy wylogowywaniu).
- Dodanie sprawdzenia ról oraz przekierowanie na odpowiednią ścieżkę.
- Dodanie scroll'a do przewijania pomiędzy sekcjami strony głównej.
- Utworzenie strony admina oraz jego podstron.
- Dodanie strony 404.

2 Zadania na kolejny tydzień

- Dodanie dobrze działającej autoryzacji.
- Dodanie pozostałych stron paneli (pacjenta, lekarza, aptekarza).

3 Bilans realizacji

Założony czas: 60h

Sumaryczny czas: 34h + 30h = 64h

4 Pytania, problemy, eskalacje

- Problem z implementacją passport-jwt.
- Czy jest jakąś inną alternatywą zamiast passport-jwt?

I Wykousue eksploracji

- Dostosowanie portów dla dedykacji

1. Niestandardowa numer PESEL

2. Abyść numer nie był identyczny

3. Hasta nie będzie identyczny

4. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

5. Numer musi się różnić o co najmniej 2 znaki, a dla co najmniej 1 znaku

6. Numer musi się różnić o co najmniej 2 znaki, a dla co najmniej 1 znaku

7. Numer musi się różnić, jednakże może być identyczny z innym

8. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

9. Numer musi się różnić o co najmniej 2 znaki, a dla co najmniej 1 znaku

10. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

11. Numer musi się różnić o co najmniej 2 znaki, a dla co najmniej 1 znaku

12. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

13. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

14. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

15. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

16. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

17. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

18. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

19. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

20. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

21. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

22. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

23. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

24. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

25. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

26. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

27. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

28. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

29. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

30. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

31. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

32. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

33. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

34. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

35. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

36. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

37. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

38. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

39. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

40. Użycie innego kodu znaku w numerze PESEL (zamiast 1)

2 Zasady do polityka dystrybucji

- Dostosowanie portów dla dedykacji

• Dostosowanie portów dla dedykacji

Raport

AKADEMIA NAUK STOSOWANYCH W NOWYM SĄCZU Wydział Nauk Inżynierijnych, Katedra informatyki					
Przedmiot:	Bazy danych – projekt, mgr inż. Nikodem Bulanda				
Grupa:	IS-2(s)P1	Sprawozdanie:	3	Data oddania:	26.04.2023
Imię i nazwisko:	Antoni Garczyński, Marcin Gonciarz				

1 Wykonane zadania

- Dodanie autoryzacji (weryfikacja tokenu, czasowy token 1h, autoryzacja dla strony admina i jego podstron za pomocą tokenu JWT).
- Dodanie strony 404.
- Dodanie podstron dla każdego panelu.
- Dodanie bazy leków.

2 Zadania na kolejny tydzień

- Przygotowanie panelu admina do wyświetlania danych z bazy.

3 Bilans realizacji

Załóżony czas: 24h

Sumaryczny czas: 13h + 10h = 23h

4 Pytania, problemy, eskalacje

- Brak

- Ad 2.
- Przygotowanie funkcjonalności po stronie backend'u dla załącznika: „zarezerwowanie kontami” (dodanie, usunięcie, edycja)
 - Przygotowanie funkcjonalności po stronie backend'u dla załącznika „baza leków” (dodanie, usunięcie, edycja)

26.04
B24

Raport

AKADEMIA NAUK STOSOWANYCH W NOWYM SĄCZU Wydział Nauk Inżynierjowych, Katedra informatyki					
Przedmiot:	Bazy danych – projekt, mgr inż. Nikodem Bulanda				
Grupa:	IS-2(s)P1	Sprawozdanie:	4	Data oddania:	10.05.2023
Imię i nazwisko:	Antoni Garczyński, Marcin Gonciarz				

1 Wykonane zadania

- Modyfikacja autoryzacji
próba nieautoryzowanego dostępu kończy się przejęciem na stronę logowania, a nie jak to były poprzednio wyświetleniu render'u strony 404,
- Wyświetlenie bazy leków oraz wyszukiwanie ich po nazwie
wyszukiwanie odbywa się po nazwie leku, nie trzeba wpisać całej nazwy można tylko cześć z początku, ze środka lub z końca nazwy ,
- Edycja ilości leków
dodanie kolumny ilości do bazy danych leków pobranej ze strony ministerstwa zdrowia oraz uzupełnienie tej kolumny wartościami losowymi,
- Wyświetlenie użytkowników oraz możliwość zmiany danych i usunięcie
Przy edycji użytkowników obecne wartości się pobierane z bazy i uzupełniane w formularzu,
- Stworzenie strony do dodawania użytkowników
brak podłączenia z backend'em.

2 Zadania na kolejny tydzień

- Przygotowanie podstron dla panelu pacjenta
 - wyświetlanie recept - lista z receptami wystawionymi w przychodni przez lekarzy oraz przycisk rozwinięcia każdej recepty aby móc wyświetlić leki, moc, opakowanie, dawkowanie oraz ulotkę każdego przepisanego leku,
 - wyświetlanie kartoteki - wyświetlenie historii leczenia w przychodni oraz przycisk po kliknięciu którego rozwijają się dokładny opis choroby, cel wizyty, zalecenia i opcjonalnie termin kolejnej wizyty,
 - wyświetlanie danych pacjenta oraz jego edycja - edytować będzie można imię, nazwisko, email, hasło.
- Połączenie backend'u do dodawania użytkowników w panelu admina

3 Bilans realizacji

Założony czas: 60h
Sumaryczny czas: 24h + 30h = 54h

4 Pytania, problemy, eskalacje

Brak

10.10

Raport

AKADEMIA NAUK STOSOWANYCH W NOWYM SĄCZU Wydział Nauk Inżynierijnych, Katedra informatyki					
Przedmiot:	Bazy danych – projekt, mgr inż. Nikodem Bulanda				
Grupa:	IS-2(s)P1	Sprawozdanie:	5	Data oddania:	17.05.2023
Imię i nazwisko:	Antoni Garczyński, Marcin Gonciarz				

1 Wykonane zadania

- Dodanie możliwości wgrywania pliku *.csv z bazą leków pobraną ze strony ministerstwa zdrowia. Po wgraniu pliku strona zostaje odświeżona, a nowe dane są wyświetlane.
- Wyświetlanie danych użytkownika i możliwość edycji danych. Pacjent, lekarz po wejściu w zakładkę "Dane pacjenta", "Dane pacjenta" zobaczy swoje dane takie jak: imię, nazwisko, data urodzenia, PESEL, email i zahasowane hasło. Po naciśnięciu "Edycja" użytkownik zostaje przekierowany do formularza w którym może zmienić imię, nazwisko i email.
- Dodanie kartoteki pacjenta do jego konta. Użytkownikowi w tej zakładce wyświetlają się Data wizyty i lekarz. Po rozwinięciu danej zakładki użytkownik może sprawdzić cel wizyty, objawy, wyniki badań, zalecenia i termin kolejnej wizyty. Dane wrażliwe takie jak objawy, wyniki badań i zalecenia są haszowane po stronie bazy za pomocą algorytmu rsc (Rivesta-Shamir-Aadlemana) z biblioteki nodejsa, więc nikt poza użytkownikiem nie ma możliwości zobaczenia tych danych.
- Wyświetlanie recept pacjenta. Użytkownikowi w tej zakładce wyświetlają się Data wizyty i lekarz. Po rozwinięciu danej zakładki użytkownik może sprawdzić informacje o przepisanych lekach, dawkowanie każdego leku, kod recepty oraz jeżeli jest dostępna w bazie ulotka danego leku to pacjent może ją pobrać.

2 Zadania na kolejny tydzień

- Przygotowanie podstron dla panelu doktora
 - edycja kartotek - Lekarz będzie mógł dodać nowy wpis do kartoteki danego pacjenta,
 - wystawianie recept - Lekarz będzie mógł wystawić receptę danemu pacjentowi, przy wystawianiu recept lekarz będzie miał dostęp do bazy leków poprzez wyszukiwarkę w której wyświetlać się będą najważniejsze informacje o danym leku
- Naprawa komponentu do zmiany hasła użytkownika

3 Bilans realizacji

Załóżony czas: 36h
Sumaryczny czas: 15h + 15h = 30h

17.05

4 Pytania, problemy, eskalacje

- Problem z renderem komponentu do zmiany hasła użytkownika

Raport

AKADEMIA NAUK STOSOWANYCH W NOWYM SĄCZU Wydział Nauk Inżynieryjnych, Katedra informatyki				
Przedmiot:	Bazy danych – projekt, mgr inż. Nikodem Bulanda			
Grupa:	IS-2(s)P1	Sprawozdanie:	6	Data oddania: 24.05.2023
Imię i nazwisko:	Antoni Garczyński, Marcin Gonciarz			

1 Wykonane zadania

- Dodawanie recept
Lekarzowi po przejściu na stronę "Wystawienie recept" ukazuje się lista pacjentów wraz z polem do wyszukiwania danego pacjenta po imieniu i nazwisku lub numerze PESEL. Koło każdego pacjenta jest przycisk "Wystaw receptę" który przekierowuje na stronę na której tworzona jest recepta i lekarz może wybierać leki, wpisywać opakowania i dawkowanie.
- Edycja kartoteki
Lekarzowi po przejściu na stronę "Edycja kartotek" ukazuje się lista pacjentów wraz z polem do wyszukiwania danego pacjenta po imieniu i nazwisku lub numerze PESEL. Koło każdego pacjenta jest przycisk "Dodaj wpis" który przekierowuje na stronę na której jest formularz do uzupełnienia przez lekarza.
- Naprawienie zmiana hasła
(Problem z poprzedniego tygodnia)
- Wyświetlanie bazy leków w panelu aptekarza
Po wejściu na stronę "Baza leków", aptekarz ma możliwość wyszukania leku z bazy leków, zmiany ilości danego leku
Panel ten ma służyć aptekarzowi do sprawdzania dostępności leków i ewentualnej zmiany ilości.
- Zmiana swoich danych przez aptekarza
W zakładce "Podgląd danych", aptekarz ma możliwość podglądu swoich danych zapisanych w bazie oraz ewentualnej zmiany ich, wraz z hasłem.
- Utworzenie zapytań obsługi i realizacji recepty pacjenta przez aptekarza
(wyświetlenie recept pacjenta, realizacja recept pacjenta)

2 Zadania na kolejny tydzień

- Stworzenie formularza do realizacji recept pacjenta po stronie aptekarza(recepta pacjenta)
aptekarz będzie mieć możliwość znalezienia recept pacjenta po numerze PESEL. Po poprawnym wpisaniu numeru PESEL pojawią się wszystkie ważne, niezrealizowane recepty pacjenta, gdzie koło każdej będzie pole wraz z przyciskiem, do którego będzie się wpisywało kod recepty i jeżeli będzie poprawny, przepuści nas na stronę do jej realizacji.
Realizacja będzie polegała na tym że aptekarzowi wyświetla się przepisane pacjentowi leki które będzie mógł odznaczać, co będzie znaczyło wydanie tych leków pacjentowi.
- Zakładka złożone zamówienia w panelu aptekarza
Aptekarzowi w tej zakładce po wejściu w nią pokażą się wszystkie zamówienia posortowane od najstarszych, które będzie mógł zrealizować lub odrzucić realizację podając powód.

3 Bilans realizacji

Zalożony czas: 24h
Sumaryczny czas: 15h + 13h = 28h

Raport

AKADEMIA NAUK STOSOWANYCH W NOWYM SĄCZU Wydział Nauk Inżynierijnych, Katedra informatyki				
Przedmiot:	Bazy danych – projekt, mgr inż. Nikodem Bulanda			
Grupa:	IS-2(s)P1	Sprawozdanie:	7	Data oddania: 31.05.2023
Imię i nazwisko:	Antoni Garczyński, Marcin Gonciarz			

1 Wykonane zadania

- Wyświetlanie danych i ich zmiana w panelu aptekarza.
Po wejściu aptekarza w zakładkę "Podgląd danych", wyświetla się informacje takie jak: imię, nazwisko, email, data urodzenia, Pesel i ukryte hasło. Kliknięcie opcji "Edycja" otwiera formularz umożliwiający zmianę imienia, nazwiska i emaila. Przechodząc do kolejnej strony poprzez "Zmiana hasła", otwiera się formularz, w którym można wpisać stare i nowe hasło.
- Wyświetlanie bazy leków, wraz z ilością sztuk na stanie w aptece.
Po przejściu do zakładki "Baza leków", aptekarz zobaczy listę leków wraz z polem wyszukiwania, w którym można wpisać nazwę leku i wyszukać go. Przy każdym leku jest opcja zmiany ilości sztuk dostępnych w aptece.
- Wyszukiwanie i realizacja recept pacjenta przez aptekarza.
Aptekarz przechodząc do zakładki "Recepty pacjenta", wpisuje numer PESEL pacjenta w polu wyszukiwania. Po wpisaniu poprawnego numeru PESEL wyświetla się wszystkie ważne i niezrealizowane recepty pacjenta. Obok każdej z recept pojawia się pole, w które można wpisać 4-cyfrowy kod recepty. Kod ten jest sprawdzany po stronie backendu, a jeśli jest zgodny z zapisanym w bazie, aptekarz zostaje przekierowany na stronę, na której wyświetla się leki przepisane pacjentowi wraz z informacjami o opakowaniu i dawkowaniu. Obok każdego leku znajduje się pole wyboru, które umożliwia odznaczenie leków, które zostaną wydane pacjentowi. Po odznaczeniu, lek ten nie będzie już widoczny w tej recepcji.
- Realizacja zamówień złożonych przez pacjentów w panelu aptekarza (backend, baza danych).
Stworzone zostały tabele do obsługi zamówień. W backendzie zaimplementowane są funkcje, takie jak: wyświetlanie zamówień do zrealizowania przez aptekarza, wyświetlanie listy leków w wybranym zamówieniu oraz wyświetlanie zamówień wraz z ich statusem w panelu pacjenta. Aptekarz może zrealizować zamówienia, zmieniając ich statusy na "Gotowe do odbioru", "Odrzucone" lub "Odebrane".

2 Zadania na kolejny tydzień

- Wyświetlanie listy zamówień w panelu pacjenta.
Pacjent będzie miał możliwość przeglądania listy swoich zamówień w aptece, z informacjami o zamówionych lekach oraz ich statusie.
- Wyszukanie leku i dodanie go do koszyka w celu złożenia zamówienia.
Pacjent będzie mógł wyszukać lek i dodać go do koszyka, aby następnie złożyć zamówienie. Po złożeniu zamówienia, zostanie ono przesłane do apteki w celu realizacji. Pacjent będzie mógł śledzić status swojego zamówienia w dedykowanej zakładce, gdzie wyświetlane są wszystkie zamówienia.
- Dokończenie obsługi zamówień po stronie aptekarza.

3 Bilans realizacji

Założony czas: 20h
Sumaryczny czas: 10h + 7h = 17h

Raport

AKADEMIA NAUK STOSOWANYCH W NOWYM SĄCZU Wydział Nauk Inżynierijnych, Katedra informatyki				
Przedmiot:	Bazy danych – projekt, mgr inż. Nikodem Bulanda			
Grupa:	IS-2(s)P1	Sprawozdanie:	8	Data oddania: 07.06.2023
Imię i nazwisko:	Antoni Garczyński, Marcin Gonciarz			

1 Wykonane zadania

- Stworzenie harmonogramu przyjęć lekarskich
Administrator jako osoba zarządzająca tworzy harmonogram wybierając lekarza, następnie podając przez jaki okres czasu lekarz będzie przyjmował. W dalszej części wpisuje w obok wybranego dnia Tygodnia czas rozpoczęcia i zakończenia wizyt lekarskich, po czym wszystkim daje zapisz. Pacjent wchodzący w zakładkę "rejestracji na wizytę" ma do wyboru lekarza, po wyborze lekarza, wybiera datę przyjęcia i na samym końcu wolną godzinę. Po wybraniu pacjentowi ukazuje się informacja o pomyślnym zarejestrowaniu się.
Lekarz po wejściu w harmonogram może zobaczyć listę pacjentów i godzinę, których ma do przyjęcia w obecnym dniu.
- Stworzenie Docker'a
Zainstalowanie Docker'a desktop oraz skonfigurowanie wszystkich potrzebnych plików do działania Docker'a.
- Nagranie filmów przedstawiających działanie strony internetowej.
Nagrania można znaleźć: <https://github.com/antonio-23/farmmed-website#readme> lub jako cały film <https://www.youtube.com/watch?v=Wl3RD-9mBJo>.
- Dokończenie dokumentacji
Uzupełnienie estymacji czasowej, wykonanie implementacji i opis kodów.

2 Bilans realizacji

Założony czas: 34h
Sumaryczny czas: 24h + 18h = 42h

3 Pytania, problemy, eskalacje

- Problem z bazą danych na Dockerze.