**POLITECHNIKA CZĘSTOCHOWSKA**

**WYDZIAŁ INŻYNIERII MECHANICZNEJ I INFORMATYKI**



PRACA DYPLOMOWA LICENCJACKA INŻYNIERSKA

**Implementacja Systemu Obsługi Pacjenta Kliniki Zdrowia**

*Implementation Of The Clinic’s Customer Service*

David Jabłoński

Nr albumu: 126068

Kierunek: Informatyka

Studia: stacjonarne

Poziom studiów : pierwszego

Promotor pracy:

dr hab. inż. Prof. Pcz Janusz Starczewski

Praca przyjęta dnia:

Podpis promotora:

*Częstochowa, 2019*

**Spis treści**

[Wstęp 5](#_Toc29331230)

[Cel pracy 6](#_Toc29331231)

[1 Przegląd dostępnych rozwiązań 7](#_Toc29331232)

[1.1 Rozwiązanie pierwsze 7](#_Toc29331233)

[1.2 Rozwiązanie drugie 7](#_Toc29331234)

[1.3 Podsumowanie przeglądu 8](#_Toc29331235)

[2 Architektura aplikacji 9](#_Toc29331236)

[3 Wykorzystane technologie oraz narzędzia 10](#_Toc29331237)

[3.1 Baza danych 10](#_Toc29331238)

[3.2 Backend 12](#_Toc29331239)

[3.3 Frontend 16](#_Toc29331240)

[3.4 Inne 19](#_Toc29331241)

[4 Założenia projektowe 20](#_Toc29331242)

[4.1 Wymagania funkcjonalne 20](#_Toc29331243)

[4.2 Diagramy sekwencji 26](#_Toc29331244)

[4.3 Schemat bazy danych 29](#_Toc29331245)

[5 Prezentacja opracowanego rozwiązania 33](#_Toc29331246)

[5.1 Rejestracja i logowanie 33](#_Toc29331247)

[5.2 Zmiana hasła 33](#_Toc29331248)

[5.3 Ekran główny pacjenta 33](#_Toc29331249)

[5.4 Ekran główny doktora 33](#_Toc29331250)

[5.5 Lista doktorów 33](#_Toc29331251)

[5.6 Umawianie wizyty 33](#_Toc29331252)

[5.7 Wypisywanie recepty 33](#_Toc29331253)

[5.8 Podgląd wizyty 33](#_Toc29331254)

[5.9 Edycja profilu 33](#_Toc29331255)

[6 Opis wybranych rozwiązań programistycznych 34](#_Toc29331256)

[6.1 Mechanizm rezerwacji wizyty 34](#_Toc29331257)

[6.2 Mechanizm wypisywania recepty 37](#_Toc29331258)

[6.3 Mechanizm wysyłania powiadomienia 38](#_Toc29331259)

[6.4 Mechanizm edycji profilu 38](#_Toc29331260)

[6.5 Mechanizm seedowania (wypełniania) bazy danych 39](#_Toc29331261)

[Podsumowanie 41](#_Toc29331262)

[Streszczenie 42](#_Toc29331263)

[Spis rysunków 43](#_Toc29331264)

[Spis listingów 44](#_Toc29331265)

[Dodatek. Zawartość płyty CD 45](#_Toc29331266)

[Bibliografia 46](#_Toc29331267)

# Wstęp

Opis rozwoju technologii, zapotrzebowania na pewne rozwiązanie, opis tego co skłoniło do wybrania i napisania właśnie tej pracy.

# Cel pracy

Celem pracy jest stworzenie systemu opartego na jednym z najbardziej popularnych obecnie frameworków wykorzystującego wzorzec architektoniczny MVC – Laravel, oraz przedstawienie największych zalet dobrodziejstw omawianej biblioteki.

Aplikacja cechować się będzie prostotą budowy, szybkością działania oraz przyjaznym i intuicyjnym interfejsem użytkownika. Wymienione aspekty zostaną omówione na przykładzie aplikacji internetowej, która będzie umożliwiać rezerwację wizyt u lekarza prywatnej kliniki oraz wgląd i wypisywanie recept z pozycji doktora, wraz z przykładową implementacją systemu powiadomień.

Dodatkowym celem pracy jest przedstawienie potencjalnych rozwiązań rozwojowych aplikacji, takich jak np. rozbudowa aplikacji o panel administratora.

# Przegląd dostępnych rozwiązań

Tytuł rozdziału musi odpowiadać treści rozdziału. Na początku każdego dużego rozdziału przed podrozdziałem, muszą być dwa, trzy zdania opisujące zawartość rozdziału.

## Rozwiązanie pierwsze

Praca powinna spełniać następujące wymogi:

a) format pracy - A4

b) kolejność stron w pracy:

* strona tytułowa
* spis treści -automatyczny
* wstęp
* rozdziały
* podsumowanie
* literatura
* załączniki

c) czcionka nr 12; 1,5 odstępu między wierszami; margines z lewej strony – 3,5 cm; z prawej oraz górny i dolny – 2,0 cm;

d) 4 egzemplarze pracy (miękka oprawa 3 szt.: promotor, recenzent i dziekanat, 1 szt. twarda oprawa, którą student przynosi na obronę, a po zabiera ze sobą na pamiątkę.

1. do pracy dołączyć wersję elektroniczną pracy – w kopercie przyczepionej do wewnętrznej strony, tylnej okładki podpisanym imieniem i nazwiskiem oraz z tytułem pracy na samej płycie/nośniku.

## Rozwiązanie drugie

### Numeracja rozdziałów

Rozdziały numeruje się do drugiego poziomu tzn. 1; 1.1, 1.2, 1.3; 2: 2.1 ,2.2, .....

Jeżeli jakaś część pracy wymaga wydzielenia to stosujemy nagłówek trzeciego poziomu bez numeracji i nie umieszcza się go w spisie treści. (przykład niniejszy podrozdział).

## Podsumowanie przeglądu

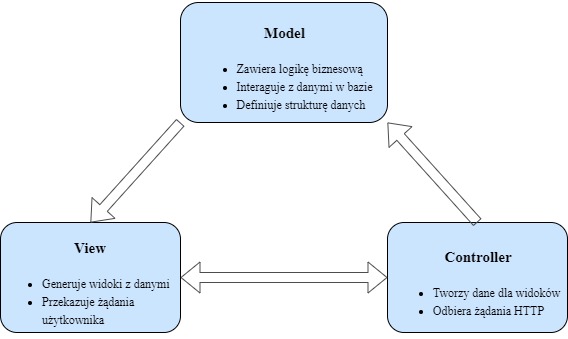
### Tytuł podsumowania przeglądu

Dlaczego proponowane rozwiązanie będzie się wyróżniało na tyle tych które umieszczono w przeglądzie.

# Architektura aplikacji

Aplikacja została zaprojektowana zgodnie z wzorcem architektonicznym MVC (z ang. Model View Controller). Jest to tradycyjne podejście do projektowania aplikacji internetowych. Pozwala to na luźne powiązanie ze sobą różnych części systemu i utrzymywanie ich w tej samej strukturze co ułatwia śledzenie i naprawę błędów. Największą wadą tego zastosowania jest jej zasobożerność przy większych projektach, gdyż wszystko odbywa się po stronie serwera.

Rysunek 2.1 Schemat architektury aplikacji



# Wykorzystane technologie oraz narzędzia

W tej części przedstawione i opisane zostaną wszystkie technologie i narzędzia które wykorzystane zostały podczas tworzenia aplikacji internetowej Health-Clinic.

## Baza danych

Pierwsze relacyjne bazy danych pojawiły się już w latach 70. Twórcą teorii relacyjnej która stanowi fundament relacyjnych baz danych była przedstawiona przez Franka Edgara Codda praca pt. „*A Relational Model of Data for Large Shared Data Banks*”. Opisano w niej podstawowe zależności jakie mogą zaistnieć między danymi.

Baza danych jest uporządkowanym zbiorem informacji (danych), który zazwyczaj przechowywany jest w formie elektronicznej na fizycznym nośniku danych należącym do jakiegoś systemu komputerowego. Do zarządzania bazą danych wykorzystywany jest DMBS (z ang. Database Management System).

Przyjęło się, że powiązane z bazą danych aplikacje i usługi wraz z DBMS w skrócie nazywane są bazą danych.[[1]](#endnote-1)

### Relacyjna baza danych

Relacyjna baza danych jest rodzajem bazy w której informacje (dane) przechowywane są w postaci zbioru tabel (relacji). Dane zawarte w tabelach odpowiadają modelom wyabstrahowanym na potrzeby zaprojektowania działającego systemu informatycznego bądź aplikacji. Każda z tabel zawiera co najmniej jedną kolumnę (atrybut danych), która jest opisana słownie (np. marka, model, pojemność silnika) oraz wiersze (nazywane również rekordem lub krotką), które mogą zawierać wartości dla każdej z kolumn (atrybutu) i są kolejnymi instancjami modelu danych. Każdy rekord posiada swój unikalny identyfikator nazywany kluczem (najczęściej w postaci nieujemnej liczby całkowitej), dzięki czemu możliwe jest definiowanie relacji pomiędzy tabelami - tabela podrzędna może posiadać kolumnę wskazującą na klucz główny tabeli nadrzędnej.

Taka struktura sprawia że jest to obecnie najbardziej akceptowany i spójny sposób przechowywania danych, dzięki czemu analiza jej zawartości jest prostsza a instancje danych są niepodzielne i nie powtarzają się.[[2]](#endnote-2)

Większość dostępnych silników bazodanowych ORM (z ang. Object-Relational Mapping) do wydobywania danych wykorzystuje API (z ang. Application Programming Interface), którym jest strukturalny język zapytań - SQL (z ang. Structural Query Language).

### Serwer bazy danych MySQL

W omawianej aplikacji użyto Serwera MySQL (5.7.26) Community Server (GPL). Jest to otwartoźródłowy system umożliwiający zarządzanie relacyjnymi bazami danych. Opracowany i rozwijany jest przez firmę Oracle. MySQL powstawał z myślą o szybkości działania, dlatego we wcześniejszych wersjach nie obsługiwał nawet transakcji, jednakże w następnych wersjach wprowadzono kolejne funkcje i rozszerzenia, m.in (widoki, harmonogram zadań, transakcje). [[3]](#endnote-3)

Obecnie najnowszą stabilną wersją MySQL jest wersja 8.0.16. MySQL dostępny jest na wszystkich najpopularniejszych platformach systemowych oraz wielu różnych architekturach procesorów, co sprawia że jest to najczęściej wybierana przy współpracy z usługami i serwisami internetowymi baza danych na świecie. [[4]](#endnote-4)

### WampServer

Wamp Server 3.1.9 jest otwartoźródłowym, darmowym pakietem pozwalającym na szybkie uruchomienie w systemie Windows serwera WWW. Pierwsza wersja powstała 16 lutego 2004 roku, a jej autorem jest Romain BOURDON. [[5]](#endnote-5)

Pakiet Wamp opiera się na serwerze Apache, interpreterze skryptów języka PHP oraz serwerze baz danych MySQL. Największą zaletą tego narzędzia jest brak potrzeby manualnego konfigurowania zaawansowanych opcji serwera baz danych oraz http, co znacznie ułatwia i przyspiesza pracę programistom, którym zależy na jak najszybszym uruchomieniu i przetestowaniu środowiska. Oferuje ono takie funkcjonalności jak: [[6]](#endnote-6)

* Zarządzanie serwisami Apache i MySQL
* Włączanie/wyłączanie serwera (z możliwością przyznania dostępu każdemu, lub tylko lokalnemu hostowi)
* Instalacja i zmiana wersji Apache, MySQL, i PHP
* Zarządzanie ustawieniami serwera
* Dostęp do wpisów dziennika

### phpMyAdmin

PhpMyAdmin (wersja 4.8.5) jest darmowym, otwartoźródłowym (wydawanym na licencji GNU), oprogramowaniem napisanym w języku PHP, służącym jako narzędzie do łatwego administrowania serwerami MySQL poprzez przeglądarkę internetową. Narzędzie to pozwala na wykonywanie wielu często używanych w procesie tworzenia aplikacji internetowych operacji (m.in. tworzenie baz danych, dodawania i usuwanie tabel, dodawanie i usuwanie rekordów, relacji, indeksów, użytkowników, import oraz export baz danych itd.). Użytkownik dokonuje tego za pośrednictwem przyjaznego dla użytkownika interfejsu graficznego, jednocześnie mając możliwość bezpośredniego wykonywania zapytań SQL w celu wydobycia danych z bazy.[[7]](#endnote-7)

## Backend

### PHP 7.3.10

PHP jest interpretowanym, skryptowym językiem programowania przeznaczonym do projektowania głównie aplikacji internetowych, na wielu platformach. Wykonywany jest po stronie serwera i może być przeplatany z kodem HTML, jednakże przyjęło się, że nie jest to dobra praktyka. PHP jest darmowy i wydany na otwartej licencji.

Stworzony został przez Rasmusa Lerdorfa w 1994 roku i rozprowadzany był pod nazwą „Personal Home Page” - skąd skrót PHP. Pierwsze wersje tego języka służyły do monitorowania aktywności internautów odwiedzających stronę jego twórcy. W kolejnych latach narzędzie udostępnione przez twórcę było udoskonalane. Przełom w rozwoju PHP przyniosła wydana w 1998 roku wersja 3.0 która dała temu językowi początek paradygmatu programowania obiektowego oraz modułowość, dzięki której użytkownicy mogli łatwiej i szybciej rozszerzać funkcjonalności. Znaczna rozbudowa i zwiększenie szybkości działania PHP nastąpiła dzięki Facebook’owi, który zainwestował w jego rozwój dzięki HHVM (Hip Hop Virtual Machine), który przetwarza skrypt do kodu maszynowego „w locie”.[[8]](#endnote-8)

PHP jest dynamicznym językiem programowania, posiadającym obecnie wszystkie zalety programowania obiektowego. Nie jest językiem silnie typowanym – typy danych mogą być zmieniane w trakcie wykonywania programu, co oznacza że jest elastyczny. Około 80% stron internetowych wykorzystuje język PHP. Korzystają z niego m.in.: IBM, Wikipedia, Tesla, Facebook. W oparciu o ten język zaprojektowano również wiele popularnych bibliotek i frameworków takich jak: Symfony, Laravel, jQuery-File-Upload, composer, Guzzle, Dispatch. W chwili pisania tych słów jest to najpopularniejszy język programowania w tym segmencie.[[9]](#endnote-9)

### Composer

Composer zaczął powstawać w kwietniu 2011 roku, a jego autorami są Nils Adermann oraz Jordi Boggiano. Composer jest narzędziem pozwalającym na zarządzanie zależnościami w projektach wykorzystujących język PHP. Uruchamiany jest jako aplikacja wiersza poleceń, gdzie poprzez wywołanie odpowiednich komend programista może pobrać z repozytorium i zainstalować odpowiednie biblioteki i pakiety rozszerzające funkcjonalności aplikacji.

Dzięki temu rozwiązaniu nie jest konieczne gromadzenie wszystkich bibliotek i zależności w systemie kontroli wersji, a aktualizacja już istniejących jest prosta, ponieważ potrzebny jest jedynie plik konfiguracyjny dla Composer’a.[[10]](#endnote-10)

Plik composer.json dla każdej zależności zawiera etykietę, która składa się z: nazwy pakietu, opisu, słów kluczowych, informacji o licencji, listę pakietów wymaganych do utworzenia tego pakietu itd.[[11]](#endnote-11)

### Laravel Installer 2.1.0

Laravel Installer jest to pakiet Composer’a pozwalający instalować aplikacje, np. Laravel na maszynie na której będzie uruchamiana, za pośrednictwem komendy wiersza poleceń.[[12]](#endnote-12)

### Artisan CLI

Artisan jest domyślnym, dołączanym wraz z pakietem Laravel interfejsem wiersza poleceń. Dostarcza wiele pomocnych komend, które mogą zostać uruchomione w trakcie procesu budowania aplikacji Laravel, co znacznie przyspiesza i ułatwia pracę z w tym środowisku. Interfejs ten oparty jest o komponent konsoli należącej do biblioteki Symfony. Najczęściej używane komendy Artisan to m.in.: [[13]](#endnote-13)

* php artisan list – pozwala wyświetlić wszystkie dostępne komendy
* php artisan make:auth – wstępnie generuje mechanizmy autoryzacji
* php artisan make:model model-name – tworzy nowy model o podanej nazwie
* php artisan make:controller MyController – tworzy kontroler
* php artisan make:migration create\_users\_table – tworzy model Eloquent

### Laravel 6.2

Laravel to platforma programistyczna przeznaczona do budowania aplikacji internetowych, której autorem jest Tylor Otwell. Napisany jest w języku PHP, oraz opiera się na wzorcu architektonicznym MVC (z ang. Model-View-Controller). Laravel używa wiele trzeciorzędnych bibliotek, z czego większość to komponenty framework’u Symfony. Kod źródłowy wydany został na licencji MIT i dostępny jest za darmo dla każdego na platformie GitHub. Obecnie jest to jeden z najpopularniejszych frameworków do aplikacji internetowych. Zawdzięcza to bardzo dobrej dokumentacji, przystępnemu i spójnemu kodowi, dużej rzeszy aktywnych programistów i stałemu rozwojowi.

Oferuje podobne możliwości co inne technologie internetowe, jednak nie wymaga tak dużego nakładu pracy we wstępnej fazie projektu (konfiguracja, ułożenie struktury projektu) jak np. w porównaniu z .NET Core 3.0. Po pierwszej instalacji, aplikacja Laravel gotowa jest do współpracy z frameworkiem Vue.js, bez konieczności konfiguracji czy instalacji dodatkowych pakietów. Rozpoczęcie pracy jest szybkie. Istnieje możliwość np. automatycznego wygenerowania mechanizmu autoryzacji, korzystając z jednego polecenia linii poleceń Artisan (php artisan make:auth). Laravel wyposażony jest w mechanizmy pozwalające używać następujących wzorców projektowych:

* Budowniczy (z ang. Builder)
* Wstrzykiwanie zależności (z ang. Dependency Injection)
* Metoda wytwórcza (z ang. Factory)
* Active Record (będąca segmentem Model w architekturze MVC)
* Fasada (z ang. Facade)
* Repozytorium
* Strategia

Trasy (route’y) i uruchamianie akcji kontrolerów odbywa się w osobnym pliku gdzie również bezpośrednio można wywoływać widoki Blade. Dzięki integracji z Composer’em możliwe jest w zależności od potrzeb wykorzystanie w projekcie oficjalnych komponentów Laravel. Są to m.in.:

* Lumen – umożliwiający tworzenie Restowego API
* Socialite – pozwalający na integrację OAuth z Google’m, Twitterem, czy Facebook’iem
* Passport – pozwalający na uwierzytelnianie użytkowników przez API
* Scout – umożliwia implementację wyszukiwania tekstowego w modelach Eloquent

Zalety tego frameworka sprawiają że świetnie nadaje się do rozwoju małych i średnich projektów, gdzie znajduje swoje największe zastosowanie. W porównaniu z innymi frameworkami jak np. Symfony, czy .NET Core, może sprawiać problemy przy organizacji kodu w miarę powiększania się klas i struktury projektu. [[14]](#endnote-14)

### Eloquent ORM

Jednym z pierwszych narzędzi do mapowania było TopLink dla Javy, które powstało w 1996 roku dzięki firmie Oracle. W kolejnych latach koncepcja ta rozwijała się i stabilizowała coraz bardziej aż do teraz, kiedy jest standardem i istnieje w wielu różnych wariantach. [[15]](#endnote-15)

Mapowanie obiektowo-relacyjne (z ang. Object-Relational Mapping) jest to koncepcja odwzorowania dowolnych zapytań SQL przy użyciu dowolnego języka programowania, który umożliwia programowanie z wykorzystaniem paradygmatu programowania obiektowego. Chodzi zatem o zamianę danych w postaci obiektowej (w programie gdzie obiekty są opisane za pomocą klasy), na dane w postaci tabelarycznej (relacji w bazie danych) lub też w drugą stronę.

Taki sposób działania nazywany jest wzorcem mapowania danych (data mapper). Istnieją darmowe i komercyjne pakiety pozwalające na wykonywanie operacji konwertowania danych pomiędzy niekompatybilnymi typami systemów. Laravel domyślnie wykorzystuje wydawany na licencji MIT pakiet Eloquent.

W odróżnieniu od innych pakietów ORM (np. Doctrine, Entity Framework), Eloquent wykorzystuje wzorzec Active Record. Oznacza to, że każdej tabeli z bazy danych odpowiada model ze struktury MVC aplikacji. Tworzenie modelu Eloquent jest podobne do tworzenia zwyczajnej klasy, a każdy z nich dziedziczy po specjalnej klasie Eloquent. W tym pakiecie migracje nie są generowane automatycznie. Każda z nich stanowi model odpowiadający tabeli w bazie danych, gdzie ręcznie opisana jest nazwa tabeli jej atrybuty i ograniczenia.[[16]](#endnote-16)

### Laravel Telescope 2.1

Laravel Telescope stanowi część platformy Laravel i rozwijany jest przez ten sam zespół. Jest to narzędzie które nazywane jest asystentem debugowania gdyż pozwala na bardzo dokładny wgląd i analizę tego co dzieje się w trakcie działania aplikacji, bez konieczności przerywania jej działania. Instalacja tego pakietu odbywa się z przez system zarządzania pakietami Composer, a uruchamiany jest z poziomu przeglądarki.

Telescope zapewnia możliwość śledzenia żądań kierowanych do aplikacji, komunikatów o błędach, wpisów do dziennika zdarzeń, zapytań SQL wykonywanych do bazy danych, renderowanych szablonów widoku wraz z ich szczegółami, zadań oczekujących w kolejce, wiadomości e-mail, notyfikacji, operacji pamięci podręcznej, zaplanowanych zadań, rzutów zawartości zmiennych oraz więcej. Telescope jest obszernym narzędziem znacznie ułatwiającym i przyspieszającym pracę programistów wykorzystujących framework Laravel.[[17]](#endnote-17)

### PhpStorm 2019.1.3

PhpStorm to zintegrowane środowisko programistyczne (z ang. Integrated Development Environment) wykonany przez firmę JetBrains, na licencji komercyjnej. Jest to środowisko wieloplatformowe przeznaczone na komputery z systemami Microsoft Windows, MacOS, oraz Linux. Przeznaczone jest głównie do edycji kodu języka PHP, HTML oraz Javascript, analizuje kod w trakcie pisania i zapobiega błędom i automatycznie refaktoryzuje kod PHP. Istnieje możliwość ustawienia automatycznej refaktoryzacji do wybranego standardu. Wyposażony jest w funkcje zbliżone do tych które zawierają inne popularne środowiska, takie jak:

* Zaawansowane wyszukiwanie słów kluczowych w strukturze projektu
* Zaawansowany edytor SQL z edytowalnymi rezultatami zapytań
* Możliwość podglądu bazy danych i zaawansowany edytor diagramów
* Narzędzia do generowania kodu i ich szablonów
* Narzędzie REST Client (możliwość łatwego odpytywania Restowych API)
* Zaznaczanie kontekstowe, skróty klawiszowe z możliwością definiowania własnych
* Możliwość dostosowania interfejsu pod własne potrzeby

Oraz wiele innych. Największą wadą w porównaniu do innych środowisk IDE jest jednak brak wbudowanego debugera. Istnieje jednak możliwość instalacji silnika Zend Engine, który uzupełnia braki tego środowiska. Nie jest to jednak tak zaawansowany i niezawodny jak np. ten znajdujący się w środowisku Microsoftu – Visual Studio. [[18]](#endnote-18)

## Frontend

### Język znacznikowy HTML 5

Hipertekstowy język znaczników (z ang. Hypertext Markup Language) – jest to język znaczników wykorzystywany do określania i budowania struktury informacji zawartych w stron internetowych.

Pierwszą wersję hipertekstowego języka znaczników stworzył w 1980 r. fizyk Tim Berns – Lee. Prototyp ten powstał na potrzeby systemu informacyjnego ENQUIRE i umożliwiał podgląd dokumentów, które fizycznie znajdowały się w innych miejscach na świecie. Kolejna wersja (projekt WorldWideWeb), opierała się na sieciach Internet i została wydana w 1989 r. którą zaakceptował CERN (Europejska Organizacja Badań Jądrowych). Wydana w 1991 r. specyfikacja zawierała 22 znaczniki, które istnieją do tej pory. Obecnie standardem jest HTML5 wykorzystywany przez większość hipertekstowych dokumentów w Internecie. [[19]](#endnote-19)

HTML złożony jest z różnych znaczników (tagów), za pomocą których programista może nadać dowolnemu fragmentowi tekstu zawartego w dokumencie odpowiednie znaczenie semantyczne. Przeglądarka interpretując kod HTML nadaje zawartym w tagach informacjom konkretne cechy, np. tekst zawarty w znaczniku <footer>Tekst</footer>, będzie traktowany jako stopka strony internetowej, <h1>Nagłówek</h1>, będzie traktowany jako największy nagłówek, itd.

### Blade

Blade jest silnikiem szablonów dostarczanym wraz z frameworkiem Laravel. W przeciwieństwie do innych silników tego typu, Blade nie ogranicza programisty w używaniu czystego kodu PHP w widokach. Możliwe jest jednoczesne stosowanie znaczników html oraz skryptów PHP. Zawartość widoków Blade kompilowana jest do czystego języka PHP, co oznacza, że w warstwie reprezentacji Blade zasadniczo niczego nie narzuca. Widoki generowane są po stronie serwera.

Blade jest zaawansowanym i obszernym w funkcjonalności silnikiem. Jednymi z najważniejszych funkcji jakie oferuje Blade są dziedziczenie oraz sekcje. Pozwala to na zdefiniowanie widoku głównego, po którym dziedziczyć mogą wszystkie podstrony (muszą używać dyrektywy @extends). Dyrektywa @section pozwala wypełniać go innym widokiem. Istnieje szereg innych dyrektyw blade, które ułatwiają zarządzanie renderowaniem treści w odpowiednich miejscach strony, np. @parent, @child, lub @component pozwalający powtórnie używać istniejącego fragmentu w dowolnym innym szablonie. Możliwe jest przekazywanie do widoków zmiennych w różnej formie. Blade dobrze wspiera wyświetlanie danych, np.:

* Zawartość zmiennych określonych i nieokreślonych
* Renderowanie danych w formacie JSON

Nie brakuje struktur kontrolnych takich jak wyrażenia: @if, @switch, @foreach, @while @forelse, @empty. W widokach Blade można używać serwisów z kontenera serwisów Laravel, poprzez wstrzykiwanie ich dyrektywą @inject. Dużym udogodnieniem jest możliwość zmieniania formatu wyświetlanych danych czasu i godziny, bezpośrednio w widoku, dzięki rozszerzeniu takiemu jak Carbon, który jest częścią Laravel’a.

### Intervention Image

Intervention Image jest wolnodostępną i darmowa biblioteką napisaną w języku PHP, która dostępna jest w repozytorium Packagist, a instalacja możliwa jest poprzez narzędzie Composer. Biblioteka ta umożliwia łatwe i bardziej ekspresyjne manipulowanie obrazami w projektach. Służy do tworzenia miniatur, formatowania dużych plików graficznych, czy tworzenia znaków wodnych. [[20]](#endnote-20)

### Bootstrap 4.3.1

Bootstrap jest otwartoźródłowym zestawem narzędzi CSS (z ang. Cascading Styling Sheet), zaprojektowanym i rozwijanym przez programistów Twitter’a rozpowszechnianym na licencji MIT. Pierwsza wersja została wydana 19 sierpnia 2011r. Najbardziej aktualną, stabilną wersją jest wydana w 21.12.2018 r. wersja 4.2.1. [[21]](#endnote-21)

Biblioteka ta zaprojektowana została do rozwijania projektów z takimi technologiami jak HTML (z ang. Hypertext Markup Language), CSS, Javascript. Umożliwia szybkie i wygodne prototypowanie takich elementów wyglądu jak formularze, przyciski, nawigacje, wykresy itd. za pomocą predefiniowanych stylów. Style bootstrap’a określone są za pomocą klas o konkretnych nazwach, które następnie mogą być umieszczone w atrybucie „class” jakiegoś znacznika. W ten sposób przeglądarka przypisuje konkretnym znacznikom np. kolor, margines, wielkość itd. Możliwa jest instalacja lokalnie z poziomu konsoli za pośrednictwem Node Package Manager, bądź umieszczenie linku w szablonie widoku odnoszącego się do Rozproszonego Systemu Dostarczania treści (z ang. Content Delivery Network). [[22]](#endnote-22)

Bootstrap korzysta z Javascriptu oraz JQuery, co zwiększa poziom responsywności strony oraz ogólnych wrażeń użytkownika wynikających z zachowania komponentów na stronie. Ważnym i szeroko wykorzystywanym elementem Bootstrap’a jest Fluid Grid System, który umożliwia stylizowanie znaczników w taki sposób by niezależnie od rozdzielczości wyświetlacza i urządzenia, zachowywały się zgodnie z oczekiwaniami programisty.

## Inne

### Git

Git jest rozproszonym systemem kontroli wersji, opracowanym przez twórcę Linuxa, Linusa Torvaldsa. Pierwsza wersja została wydana 7 kwietnia 2005 r. Git jest ogólnodostępnym i darmowym oprogramowaniem, wydanym na licencji GNU GPL. Korzysta z niego bardzo wiele projektów lub ich fragmentów o otwartym źródle, np: Reddit, Facebook, Ruby on Rails, Arch Linux. Dzięki temu oprogramowaniu programiści mogą rozwijać projekt z dowolnego miejsca na świecie, niezależnie na swoich maszynach i sprawnie łączyć lub odrzucać zmiany w kodzie, nie zakłócając przy tym działania projektu dopóki nie zostanie on oficjalnie wydany jako wersja stabilna. Git wspiera takie sieciowe protokoły wymiany danych jak: http(s), SSH, FTP, rsync. Można z niego korzystać również off-line, ponieważ każdy programista posiada własną kopię repozytorium projektu. [[23]](#endnote-23)

### Github

GitHub powstał w kwietniu 2008 roku, właścicielem jest Microsoft. Jest hostingowym serwisem internetowym, przeznaczonym do magazynowania i udostępniania projektów programistycznych, które wykorzystują system kontroli wersji Git. Użytkownicy tego systemu mogą tworzyć repozytoria prywatne i publiczne (darmowe w ograniczonej ilości). Korzysta z niego wiele projektów o otwartym źródle. Za pośrednictwem interfejsu tego serwisu internetowego można korzystać z takich funkcji jak:

* Tworzenie osobnych repozytoriów, należących do innego użytkownika (z ang. fork)
* Rejestrowanie i zarządzanie informacjami o błędach
* Śledzenie statystyk
* Akceptowanie i odrzucanie zmian oraz śledzenie ich szczegółów
* Tworzenie dokumentacji projektu

# Założenia projektowe

W tym rozdziale przedstawione zostaną wymagania funkcjonalne (uwzględniające diagramy przypadków użycia i wybranych procesów) oraz schemat bazy danych wraz z opisem.

## Wymagania funkcjonalne

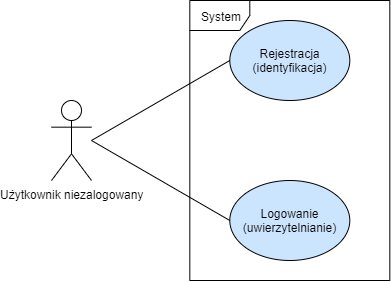
### Diagramy przypadków użycia

W tej części przedstawione zostaną funkcjonalności jakie oferuje aplikacja w zależności od potrzeb aktora (użytkownika z określoną rolą).

#### Użytkownik niezalogowany

Pierwszym omawianym aktorem, jest użytkownik niezalogowany, czyli taki który nie jest uwzględniony w tabeli użytkowników bazy danych. Użytkownik niezautoryzowany nie posiada dostępu do żadnych zasobów, jednak ma możliwość rejestracji oraz zalogowania się w systemie.

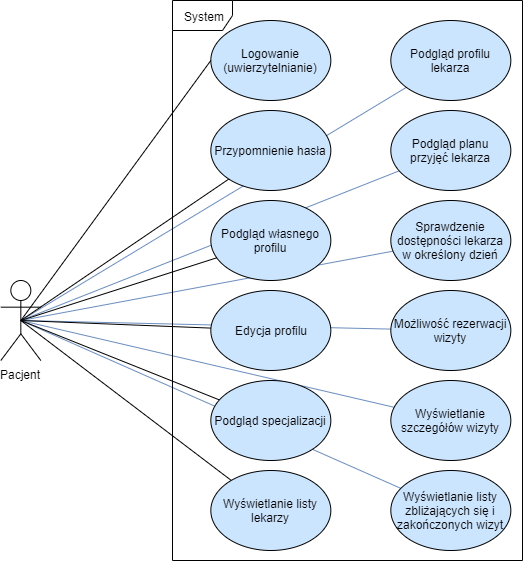
Rysunek 4.1 Diagram przypadków użycia użytkownika niezalogowanego



#### Użytkownik zalogowany (pacjent)

Drugim typem użytkownika jest użytkownik zalogowany (posiada rolę pacjenta). Jest uwzględniony w tabeli użytkowników i posiada dostęp do określonych zasobów. Ma możliwość uwierzytelniania się. Pacjent może przypomnieć hasło, podglądać swój profil, edytować go, wyświetlać listę lekarzy w zależności od ich specjalizacji, wyświetlać profil lekarza, wyświetlać plan przyjęć określonego lekarza, sprawdzać dostępność lekarza w określony dzień, umawiać wizytę na określoną datę i godzinę, wyświetlać listę zbliżających się i zakończonych wizyt oraz ich szczegóły, wyświetlać receptę dodaną do wizyty przez lekarza.

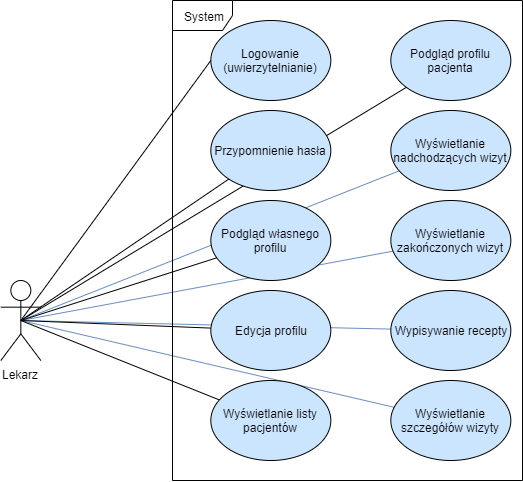
Rysunek 4.2 Diagram przypadków użycia użytkownika zalogowanego (pacjenta)



#### Użytkownik zalogowany (lekarz)

Kolejnym typem użytkownika jest zalogowany użytkownik posiadający rolę doktora. Możliwości doktora różnią się od możliwości pacjenta. Doktor może wyświetlać listę pacjentów i podglądać ich profil, wyświetla listę nadchodzących wizyt, ma możliwość zakończenia wizyty poprzez dodanie recepty w której diagnozuje pacjenta i wypisuje leki, wyświetla listę zakończonych wizyt oraz ich szczegóły.

Rysunek 4.3 Diagram przypadków użycia użytkownika zalogowanego (doktora)



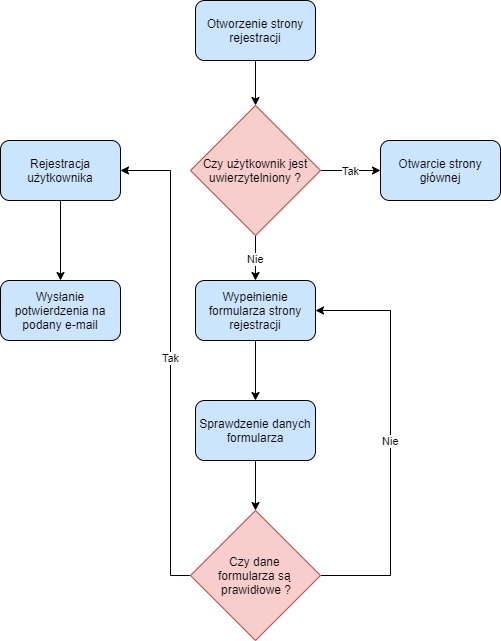
### Diagramy procesów

Poniżej wymienione są procesy przedstawiające kroki jakie są uwzględniane w procesie implementowania danej funkcjonalności.

#### Rejestracja

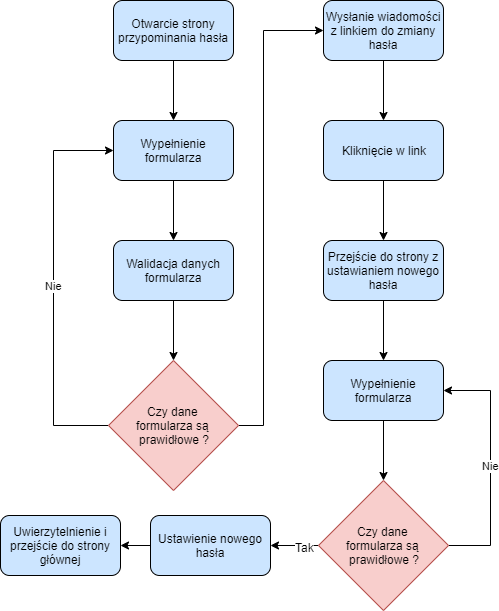
Pierwszym najważniejszym procesem jest proces rejestracji. Proces ten dostępny jest tylko dla niezalogowanych użytkowników. Po wypełnieniu i poprawnej walidacji danych formularza system tworzy profil danego użytkownika oraz generuje wiadomość potwierdzającą poprawną rejestrację, która jest wysyłana na podany przez użytkownika adres e-mail. Użytkownik który jest już zalogowany, przekierowywany jest do strony głównej odpowiadającej jego profilowi.

Rysunek 4.4 Diagram procesu rejestracji



#### Zmiana hasła

Kolejny schemat przedstawia proces zmiany hasła. Ten proces może być rozpoczęty przez każdy rodzaj użytkownika. Strona z logowaniem zawiera link przekierowujący do strony w której użytkownik uzupełnia formularz z mailem. System sprawdza czy istnieje użytkownik o podanym mailu, jeżeli tak, to wysyła na ten adres wiadomość z linkiem do strony na której można ustawić nowe hasło. Na tej stronie użytkownik podaje nowe hasło, jeżeli formularz wypełniony zostanie poprawnie, to użytkownik zostanie automatycznie zalogowany i przeniesiony na główną stronę.

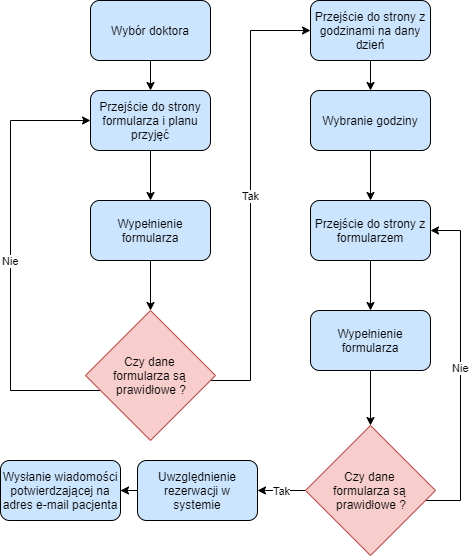


Rysunek 4.5 Diagram procesu zmiany hasła

#### Rezerwacja wizyty

Proces rezerwacji wizyty może rozpocząć tylko użytkownik z rolą pacjenta. Po wybraniu doktora z list, pacjent zostaje przekierowany do strony z formularzem, oraz informacją o planie przyjęć doktora. W formularzu pacjent wybiera datę wizyty, system waliduje, czy wprowadzona data jest poprawna. Następnie następuje przekierowanie do strony z listą godzin na dany dzień. Użytkownik widzi terminy zajęte, i może wybrać wolne, a następnie podać informację o symptomach i zatwierdzić termin. Po zatwierdzeniu i wypełnieniu ostatniego formularza, system powinien wysłać na adres e-mail pacjenta wiadomość z potwierdzeniem zarezerwowanej wizyty.

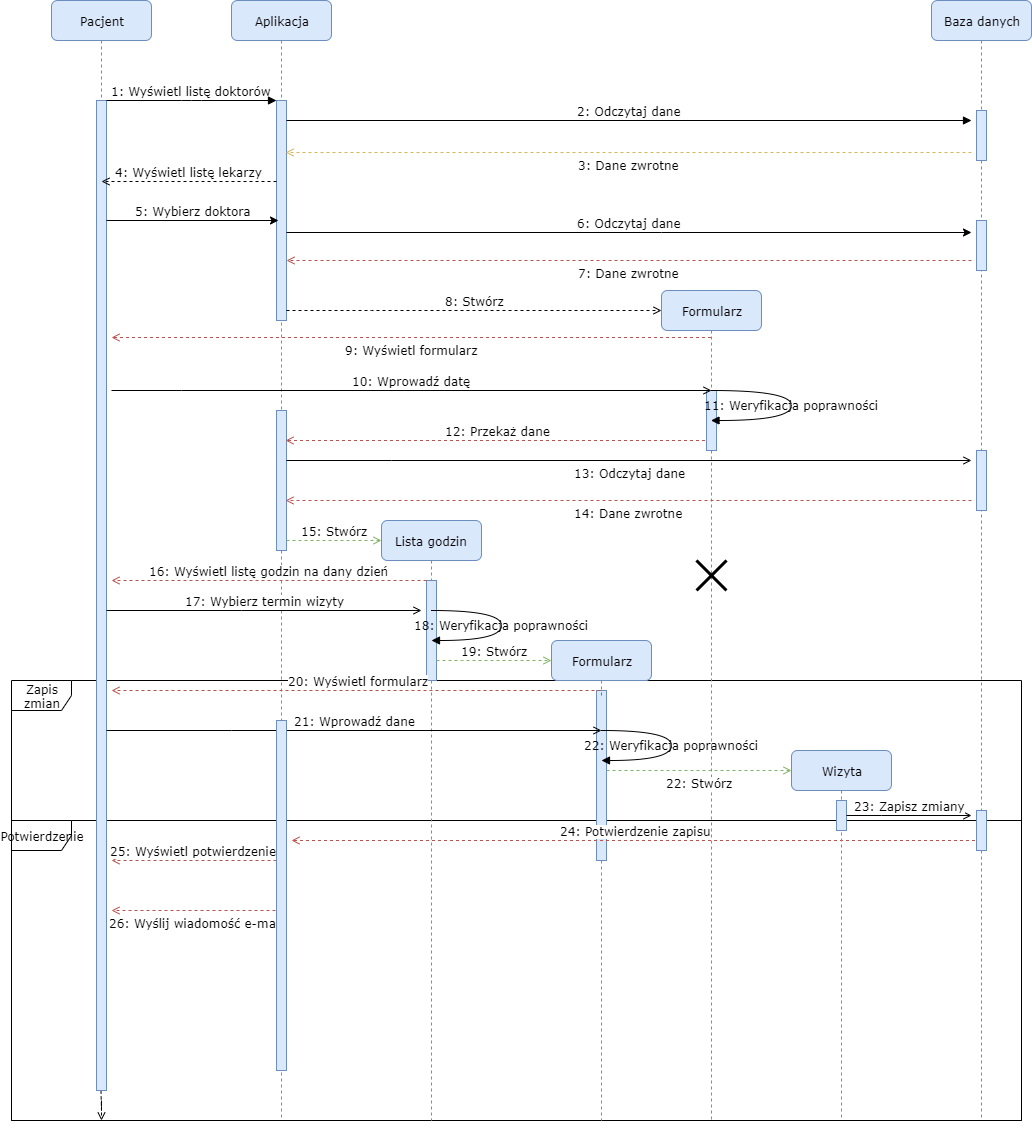
Rysunek 4.6 Diagram procesu rezerwacji wizyty



## Diagramy sekwencji

### Diagram sekwencji rezerwacji wizyty

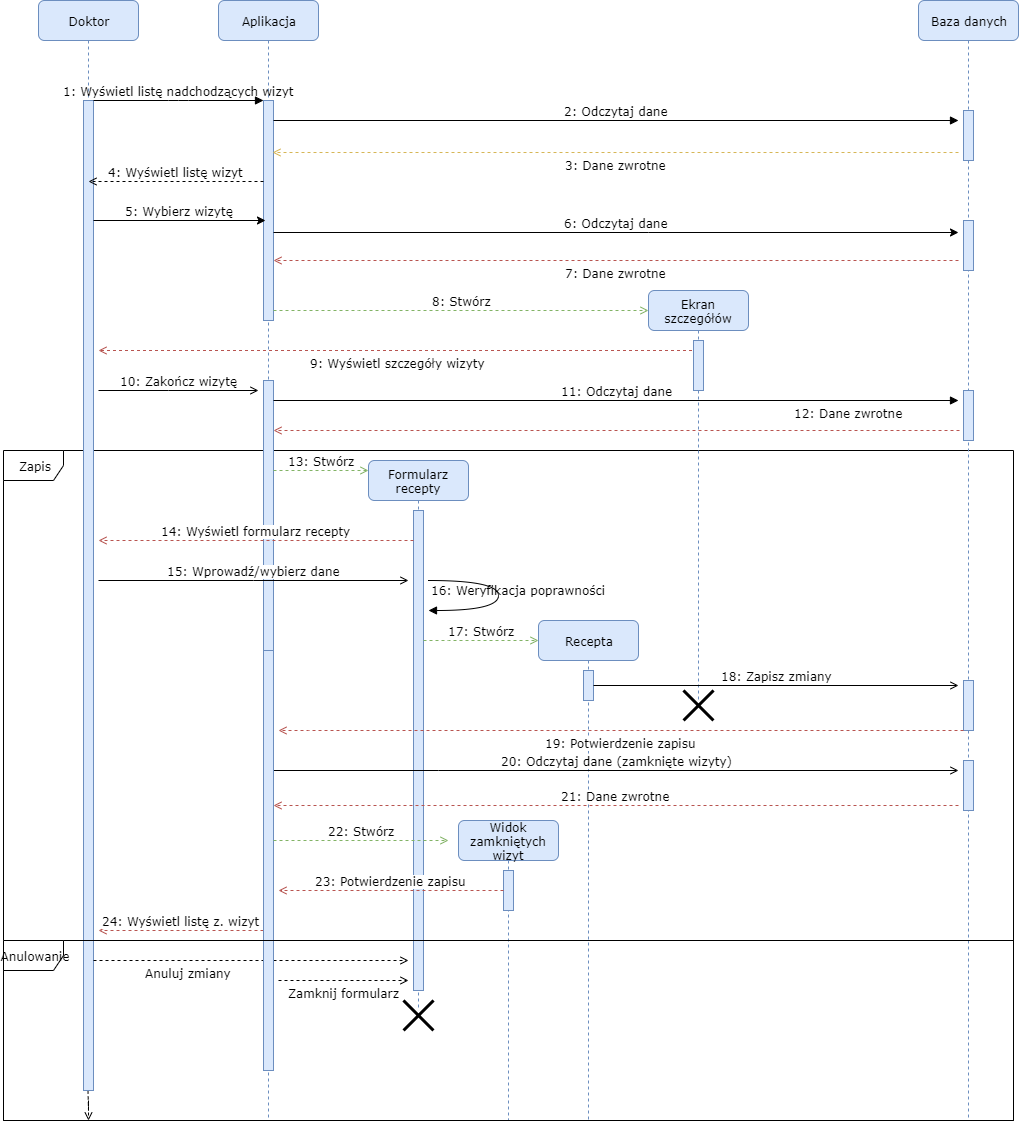
Sekwencja rozpoczyna się od wyświetlenia przez pacjenta listy doktorów. Z bazy pobierana jest lista doktorów i przedstawiona w widoku listy. Pacjent następnie wybiera jednego z lekarzy, z bazy pobierane są dane (tygodniowy plan przyjęć doktora), który wyświetlany jest pacjentowi w widoku, nad formularzem. Następnie pacjent, może w formularzu wybrać datę swojej wizyty. Następuje walidacja (nie może zostać wybrana data z przeszłości). Jeżeli walidacja przebiegnie poprawnie, to z bazy zostaje pobrany plan dla doktora na dany dzień. Tworzony zostaje widok z listą godzin dostępnych i zajętych na dany dzień, który zostaje wyświetlony na ekranie pacjenta. Następnie pacjent może wybrać jedną z dostępnych godzin, co skutkuje stworzeniem nowego formularza, który jest wyświetlany na ekranie pacjenta. W formularzu wizyty pacjent podaje powód wizyty i akceptuje wizytę, po czym dokonana zostaje walidacja a jeśli to przebiegnie poprawnie, to baza zapisuje nowy rekord wizyty. W momencie utworzenia nowego obiektu, wywołana zostaje procedura wysłania wiadomości e-mail potwierdzającej dokonanej rezerwacji.



Rysunek 4.7 Diagram sekwencji rezerwacji wizyty

### Diagram sekwencji wypisania recepty

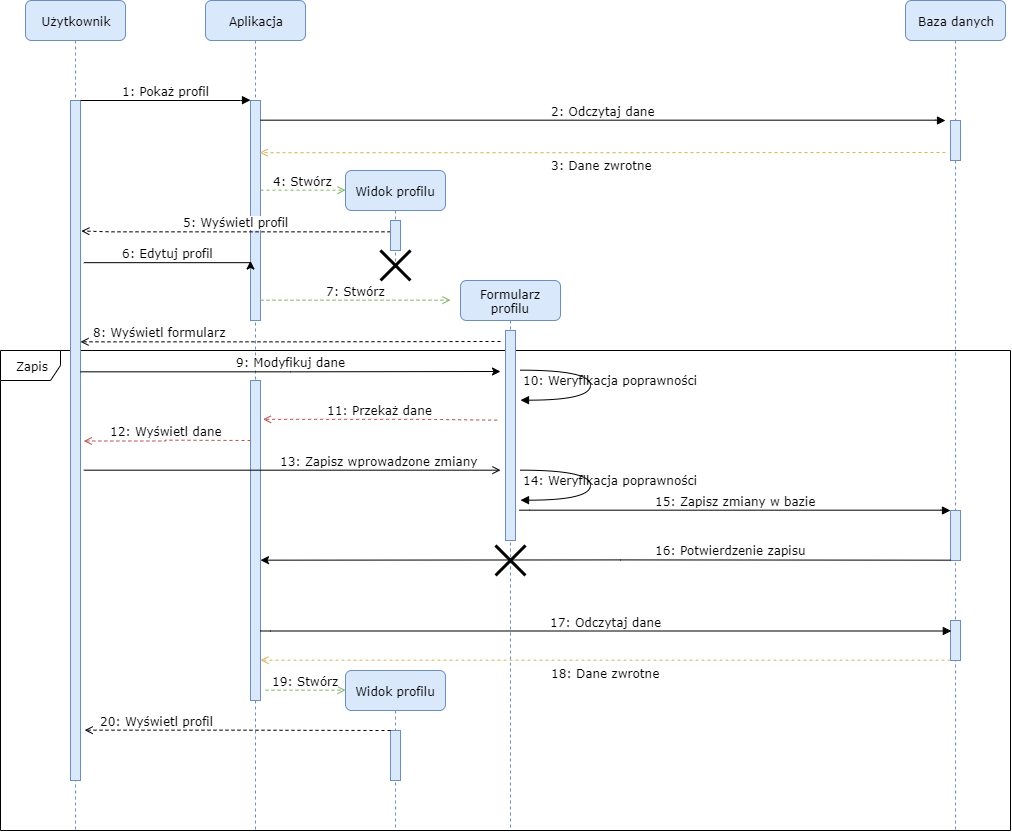
Sekwencja rozpoczyna się w momencie wyświetlenia listy nadchodzących spotkań przez użytkownika z rolą doktora. Z bazy pobierana jest ich lista, a następnie generowany jest widok, który wyświetlany jest użytkownikowi. Doktor wybiera jedną z pozycji, po czym z bazy pobierane są dane tego obiektu i generowany jest widok szczegółów. Widok szczegółów umożliwia wyświetlenie formularza kończącego wizytę. Z bazy pobierane są dane potrzebne do stworzenia opcji wyboru (leki, choroby). Doktor wprowadza i wybiera dane, które są walidowane, a następnie tworzony jest obiekt recepty (użytkownik ma możliwość zamknięcia formularza i odrzucenia zmian). Poprawne zapisanie relacji recepty powinno zakończyć wizytę. Po tej czynności z bazy pobrane są zakończone wizyty. Generowany jest widok wizyt posiadających receptę, na którą użytkownik jest przekierowywany.



Rysunek 4.8 Diagram sekwencji wystawiania recepty

### Diagram sekwencji edycji profilu

Sekwencję może rozpocząć użytkownik z rolą pacjenta lub doktora. Sekwencja rozpoczyna się od uruchomienia akcji pokazującej profil. Następnie pobierane z bazy pobierane są dane profilu aktualnie zalogowanego użytkownika. W zależności od roli, zostaje wyrenderowany odpowiedni widok, do którego przekazane są pobrane dane. Użytkownik może następnie uruchomić akcję edycji profilu, która przekazuje dane z widoku poprzedniego, generując formularz edycji. W formularzu użytkownik może zmienić dane, i wprowadzić nowe. Może również cofnąć do poprzedniej strony, lub zapisać dane, co spowoduje walidację danych i uruchomienie akcji która aktualizuje dany profil o podane dane. Po poprawnym zapisaniu danych, użytkownik powinien zostać przekierowywany do widoku profilu z widocznymi zmianami, które zostały wprowadzone.



Rysunek 4.9 Diagram sekwencji edycji profilu

## Schemat bazy danych

Poniżej przedstawiono diagram związków encji. Opisuje on sposób w jaki została zaprojektowana baza danych w celu implementacji rozwiązania.

### Users

Pierwszą najważniejszą tabelą systemu jest tabela „users”. Gromadzi ona obiekty typu „User”, który opisany jest przez takie atrybuty jak: Id (klucz główny), name, oraz służące do uwierzytelniania – email oraz password (do haszowania użyto algorytmu Bcrypt). Dodatkowo tabela ta posiada 2 kolumny typu timestamp, które określają kiedy dany rekord został zapisany do bazy oraz kiedy został zmodyfikowany.

### Roles

Jest to tabela definiująca role użytkowników zapisanych w systemie. Na podstawie roli, encji User zostaje stworzona relacja typu jeden do jednego, do jednej z tabeli (patient\_profiles, lub doctor\_profiles).

### RoleUser

Jest to tabela z relacją typu wiele do wielu. Posiada ona 2 kolumny (role\_id, oraz user\_id), które opisują klucze obce, odpowiednio z tabeli roles, oraz users. Dzięki temu rozwiązaniu każdy użytkownik może posiadać wiele ról i odwrotnie – co sprawia, że system jest łatwo rozszerzalny o kolejne funkcje i role. Rola zostaje przyznana w trakcie zapisywania rekordu użytkownika do bazy.

### PatientProfiles

To tabela profilu pierwszego typu użytkownika z rolą „ROLE\_PATIENT”. Posiada następujące kolumny: name, surname, birthDate, gender, pesel, phone, locality, street, image, oraz timestamps. Opisują one podstawowe dane osobowe pacjenta, takie jak imię i nazwisko, dane kontaktowe, płeć oraz zdjęcie profilowe. Posiada ona klucz obcy user\_id odnoszący się do tabeli users, który definiuje do jakiego użytkownika dany profil należy.

### DoctorProfiles

Ta tabela należy do drugiego typu użytkownika z rolą „ROLE\_DOCTOR”. Różni się od tabeli profilu pacjenta. Nie posiada kolumny pesel, natomiast posiada kolumnę doctor\_specialization\_id, która zawiera klucze obce z tabeli doctor\_specializations. Definiują one w specjalizację doktora, na podstawie której użytkownik z rolą pacjenta może wybierać specjalistę.

### Appointments

Jest to tabela opisująca wizyty. Każdy pacjent może umawiać się z wieloma doktorami, oraz odwrotnie – każdy doktor może być umówiony z wieloma pacjentami, dlatego zawiera relację typu wiele do wielu. Klucze obce profilu pacjenta oraz profilu doktora opisane są odpowiednio przez kolumny patient\_profile\_id oraz docotr\_profile\_id. Kolumna symptom opisuje powód wizyty podawany przez pacjenta w formularzu zatwierdzania spotkania. Wizyta może być zakończona tylko, kiedy lekarz wypisze receptę i zdiagnozuje chorobę. Kolumna is\_closed typu boolean opisuje status wizyty, który zmienia się kiedy lekarz zdiagnozuje chorobę opisaną w kolumnie disease\_id, zawierająca klucze obce tabeli leczonych chorób. Tabela przechowuje jeszcze dokładną datę i czas wizyty, w kolumnach appointment\_date, oraz appointment\_time.

### Diseases

To tabela gromadząca choroby jakie może zdiagnozować lekarz w recepcie. Posiada takie kolumny jak: name, description, oraz timestamps. Opisują one nazwę i krótki opis choroby.

### Prescriptions

Tabela recept zawiera kolumnę opisującą wypisane w recepcie zalecenia lekarza. Każdy rekord tej tabeli powinien posiadać klucz obcy opisany w kolumnie appointment\_id, który odnosi się do konkretnej wizyty poprzez relację jeden do jednego.

### Medicines

W tej tabeli przetrzymywane są lekarstwa jakie może wypisać pacjentowi lekarz w recepcie. Posiada następujące kolumny: status, name, name\_international, form, dose, package\_quantity, price. Opisują one: status lekarstwa (np. OTC - lek dostępny bez recepty, RX – wydawany na receptę), nazwę, nazwę międzynarodową, postać leku (np. tabl. Dojelitowe, kaps., krople doustne.), dawkę (np. 75 mg., 3g), opakowanie (60 szt., 10 sasz.), oraz cenę. Tabela inspirowana jest oficjalną bazą leków zawartą w leksykonie informacji medycznej.

### MedicinePrescriptions

Jest to tabela będąca łącznikiem między tabelą leków i recept (relacja wiele do wielu). Każdy lek może być w wielu receptach, oraz każda recepta może zawierać wiele przepisanych leków. Tabela posiada 2 klucze obce (prescription\_id, oraz medicine\_id), odnoszące się do odpowiednich kluczy głównych poszczególnych tabel prescriptions i medicines.

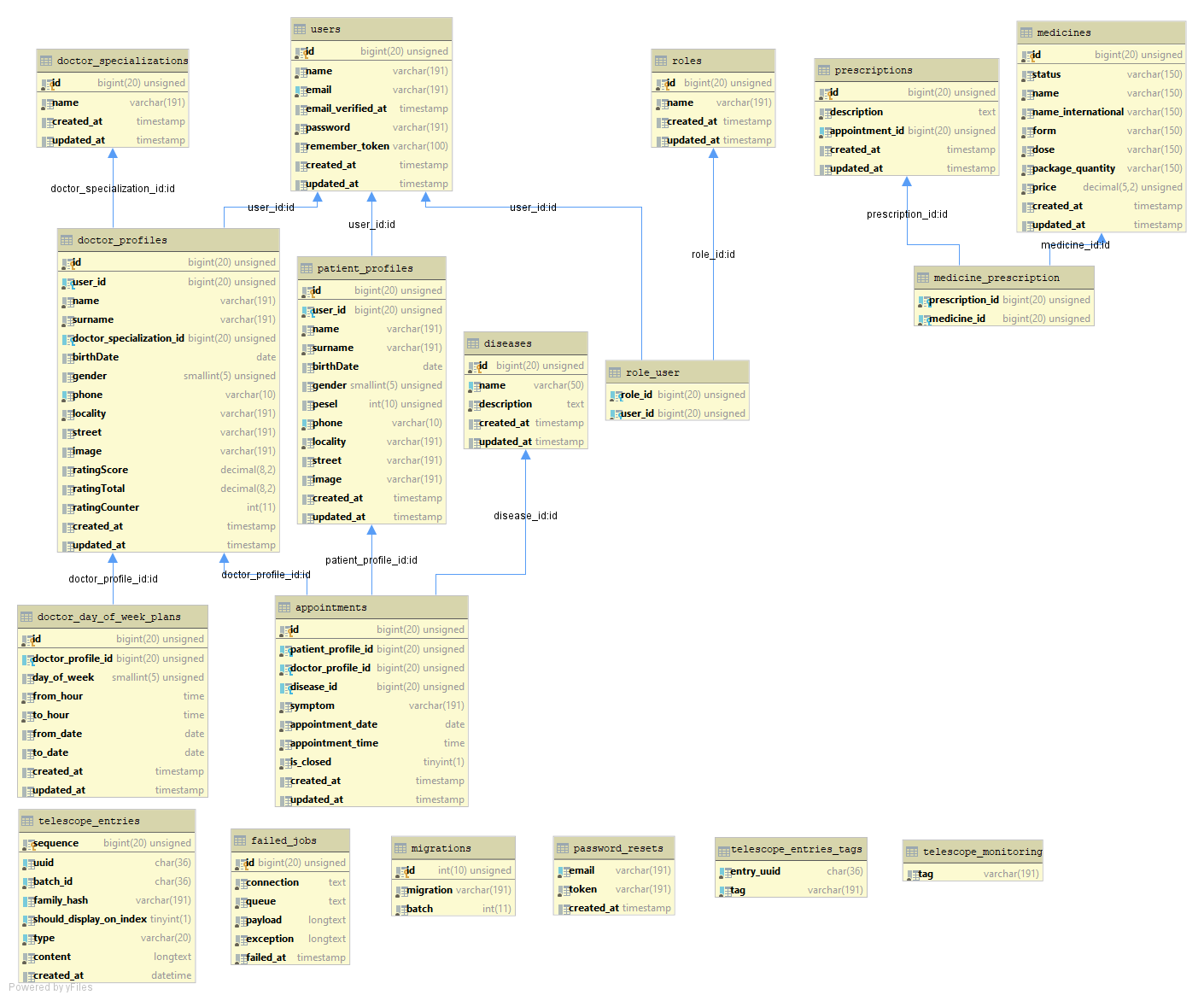
### DoctorSpecializations

Ta tabela przechowuje rodzaje specjalizacji lekarskich. Kolumna name opisuje nazwę każdej z nich. Klucz główny tej tabeli jest kluczem obcym w tabeli doctor\_profiles. Każdy lekarz ma tylko jedną specjalizację, ale każda specjalizacja może być posiadana przez wielu lekarzy (jeden do wielu).

### DoctorDayOfWeekPlans

Ta tabela służy do przechowywania planu przyjęć doktorów. Kolumna doctor\_profile\_id opisuje klucze obce odnoszące się do tabeli doktorów. Każdy lekarz może mieć wiele planów, ale każdy plan należy do tylko jednego lekarza. Kolumna day\_of\_week, zawiera numer dnia tygodnia, którego dotyczy dany plan. Kolumny from\_hour, oraz to\_hour opisują zakres godzinowy lekarza na dany dzień. Dzięki kolumnom from\_date i to\_date system można łatwiej rozszerzyć o zakres dat w jakim obowiązuje dany plan godzinowy.

Rysunek 4.10 Schemat bazy danych



# Prezentacja opracowanego rozwiązania

## Rejestracja i logowanie

## Zmiana hasła

## Ekran główny pacjenta

## Ekran główny doktora

## Lista doktorów

## Umawianie wizyty

## Wypisywanie recepty

## Podgląd wizyty

## Edycja profilu

W przypadku aplikacji i urządzeń należy kolejno pokazać funkcjonalność (wraz z zrzutami ekranu i zdjęciami). W przypadku projektu sieci należy pokazać symulacje działania sieci (z wykresami), obliczenia energii, kosztów, przykładowe aplikacje sieciowe, ewentualnie fragment rzeczywistej zbudowanej sieci (np. kilka jej elementów z jednego pomieszczenia)

# Opis wybranych rozwiązań programistycznych

W tym rozdziale przedstawiono i opisano wybrane fragmenty kodu źródłowego.

## Mechanizm rezerwacji wizyty

Na poniższym listingu przedstawiono sposób w jaki oprogramowano rezerwację wizyty u danego doktora. Po kliknięciu przycisku „Appointment” na liście doktorów, wywołana zostaje akcja „selectDate()”.Znajduje się ona w kontrolerze „AppointmentController”, która w odpowiedzi na żądania wywołuje akcje dotyczące. Przyjmuje 1 parametr – id profilu doktora. Odpowiada ona za pobranie informacji o dniach przyjęć lekarza i podaniu ich w odpowiedni sposób jako parametru do widoku, który wyświetla je użytkownikowi oraz umożliwia poprzez formularz sprawdzenie dostępności godzin na podaną datę.

Listing 1 Mechanizm rezerwacji – tworzenie widoku z formularzem sprawdzania daty

1. **public** **function** \_\_construct()
2. {
3. $this->middleware('auth');
4. }
6. **public** **function** selectDate($doctorId){
7. $doctor = DoctorProfile::find($doctorId);
8. $doctorPlans = $doctor->plans;
9. $weekMap = [
10. 0 => 'Niedziela',
11. 1 => 'Poniedziałek',
12. 2 => 'Wtorek',
13. 3 => 'Środa',
14. 4 => 'Czwartek',
15. 5 => 'Piątek',
16. 6 => 'Sobota',
17. ];
18. **return** view('select-date', compact('doctorId', 'doctorPlans', 'weekMap'));
19. }

Na listingu 2 przedstawiono mechanizmy pozwalające na weryfikację i odpowiednie wyświetlenie listy godzin na dany dzień. Po wybraniu przez pacjenta daty i zatwierdzeniu formularza, wywoływana jest akcja „showPlan()” z kontrolera „DoctorController”. Jej zadaniem jest walidacja wprowadzonej przez użytkownika daty, zwrócenie odpowiednich komunikatów użytkownikowi w przypadku niezgodności, oraz pobranie i sformatowanie danych o godzinach przyjęć lekarza w wybranej dacie, które następnie przekazywane są do widoku, z którego pacjent jest w stanie wybrać wolną godzinę.

Listing 2 Mechanizm rezerwacji wizyty - weryfikacja i pobranie godzin

1. **public** **function** showPlan($doctorId){
2. $data = request()->validate([
3. 'doctorId' => 'integer',
4. 'selectedDate' => 'required|date|after\_or\_equal:today'
5. ]);
7. $selectedDate = Carbon::createFromDate($data['selectedDate']);
9. $doctorProfile = DoctorProfile::findOrFail($doctorId);
11. $dayPlan = $doctorProfile->getDayPlan($selectedDate);
13. **if** (!$dayPlan){
14. **return** Redirect::back()->with('message', 'Doctor is not available this day. Please select different date');
15. }
17. $patientProfileId = Auth::user()->patientProfile->id;
19. **if**(Auth::user()->patientProfile->checkAppointments($doctorId, $patientProfileId, $selectedDate)){
20. **return** Redirect::back()->with('message', 'You already have appointment with this doctor on selected date !');
21. }
23. $busyHours = Appointment::all()
24. ->where('appointment\_date', $selectedDate->format('Y-m-d'))
25. ->where('doctor\_profile\_id', $doctorId)
26. ->pluck('appointment\_time')
27. ->toArray();
29. $dayHours = **array**();
30. **for**($i = Carbon::parse($dayPlan->from\_hour)->subtract('minutes', 30); $i < Carbon::parse($dayPlan->to\_hour); $i++)
31. {
32. array\_push($dayHours, $i->addMinutes(30)->format('H:i:s'));
33. }
35. **return** view('terms', compact('dayPlan','selectedDate', 'doctorId', 'busyHours', 'dayHours'));
36. }

Na listingu nr 3 przedstawiono metodykę dzięki której rozwiązany został problem wyświetlania zajętych i wolnych godzin w danym dniu (wybranym przez pacjenta w poprzednim formularzu). Do widoku przekazywane są godziny już zarezerwowane, oraz godziny w jakich doktor przyjmuje (początkowa i końcowa).

Listing 3 Mechanizm rezerwacji wizyty - wyświetlanie godzin

1. @foreach($dayHours as $dayHour)
2. @if(in\_array($dayHour, $busyHours))
3. **<div** class="card text-center card border-danger mb-3"**>**
4. **<div** class="card-header text-danger alert-danger alert-link"**>**{{ $dayHour }}**</div>**
5. **<div** class="card-body"**>**
6. **<h3>**This appointment is occupied**</h3>**
7. **</div>**
8. **</div>**
9. @else
10. **<div** class="card text-center card border-success mb-3"**>**
11. **<div** class="card-header alert-link alert-success"**>**{{ $dayHour }}**</div>**
12. **<div** class="card-body"**>**
13. **<form** action="/home/doctors/{{$doctorId}}/appointment" method="GET"**>**
14. **<input** type="hidden" value="{{$doctorId}}" name="doctorId"**>**
15. **<input** type="hidden" value="{{$selectedDate}}" name="date"**>**
16. **<input** type="hidden" value="{{$dayHour}}" name="hour"**>**
17. **<button** class="btn btn-primary" type="submit"**>**Make Appointment**</button>**
18. **</form>**
19. **</div>**
20. **</div>**
21. @endif
22. @endforeach

Listing nr 4 przedstawia w jaki sposób system tworzy obiekt wizyty. Akcja „formAppointment()” odpowiedzialna jest za stworzenie widoku z formularzem, w którym pacjent podaje przyczyny i zatwierdza wizytę. Akcja „createAppointment()” waliduje dane, zwraca komunikaty użytkownikowi oraz tworzy i zapisuje wizytę do bazy.

Listing 4 Mechanizm rezerwacji wizyt - tworzenie rezerwacji

1. **public** **function** formAppointment($doctorId)
2. {
3. $data = request()->validate([
4. 'doctorId' => 'required|integer',
5. 'date' => 'required|date',
6. 'hour' => 'required|date\_format:H:i:s'
7. ]);
9. $date = $data['date'];
10. $hour = $data['hour'];
12. **return** view('appointment', compact('doctorId', 'date', 'hour'));
13. }
15. **public** **function** createAppointment(){
16. $data = request()->validate([
17. 'doctor\_profile\_id' => 'required|integer',
18. 'patient\_profile\_id' => 'required|integer',
19. 'appointment\_date' => 'required|date|after\_or\_equal:today',
20. 'appointment\_time' => 'required|date\_format:H:i:s',
21. 'symptom' => 'required|string|max:255',
22. ]);
23. dd($data);
25. $appointmentToInsert = Appointment::where('appointment\_date', $data['appointment\_date'])
26. ->where('appointment\_time', $data['appointment\_time'])
27. ->where('doctor\_profile\_id', $data['doctor\_profile\_id'])
28. ->get();
30. **if** (($appointmentToInsert->isEmpty()))
31. {
32. Appointment::create($data);
33. **return** Redirect::route('doctor-list')->with('message', 'Your appointment is succesfully scheduled on '.Carbon::parse($data['appointment\_date'])->format('d-m-Y').' at '.$data['appointment\_time']);
34. }
35. **else**
36. **return** Redirect::route('doctor-list')->with('message', 'This appointment already exist');
37. }

## Mechanizm wypisywania recepty

Na listingu nr .. przedstawiono w jaki sposób przebiega kończenie wizyty poprzez dodawanie recepty. Zadaniem tej akcji jest powtórna walidacja danych przesyłanych z formularza, przez doktora, aktualizacja obiektu wizyty oraz stworzenie recepty i dołączenie jej relacji. Rolą tej funkcji jest również przekierowanie użytkownika i dołączenie odpowiedniego komunikatu.

Listing 5 Mechanizm wypisywania recepty

1. **public** **function** createPrescription(){
2. $data = request()->validate([
3. 'disease' => 'required|integer|',
4. 'medicines' => 'required|array',
5. 'description' => 'required|string|max:255',
6. 'appointment\_id' => 'required|integer'
7. ]);
9. dd($data);
11. $appointment = Appointment::all()
12. ->where('id', $data['appointment\_id'])->first();
14. $prescription = Prescription::create([
15. 'description' => $data['description'],
16. 'appointment\_id' => $data['appointment\_id']
17. ]);
19. **foreach** ($data['medicines'] **as** $medicine){
20. $prescription->medicines()->attach($medicine);
21. }
23. $prescriptionToCreate = **array**(
24. 'appointment\_id' => $data['appointment\_id'],
25. 'disease\_id' => $data['disease'],
26. 'is\_closed' => true
27. );
29. $appointment->update($prescriptionToCreate);
31. **return** redirect(route('doctor-appointments'))->with('message', 'Prescription created successfully without errors');
32. }

## Mechanizm wysyłania powiadomienia

Na poniższym listingu nr przedstawiono sposób w jaki działa mechanizm wysyłania powiadomień. Metoda „boot” uruchamiana jest w momencie tworzenia obiektu klasy w której jest zdefiniowana (w tym przypadku Appointment). Uruchamiana jest funkcja która wysyła szablon z w oparciu o pobrane dane. Podobnie jest przy rejestracji użytkownika.

Listing 6 Tworzenie i wysyłanie powiadomienia

1. **protected** **static** **function** boot()
2. {
3. parent::boot();
4. **static**::created(**function** ($appointment){
5. try{
6. $patientProfile = PatientProfile::where('id', $appointment->patient\_profile\_id)->first();
7. $doctorProfile = DoctorProfile::where('id', $appointment->doctor\_profile\_id)->first();
8. $specialization = DoctorSpecialization::where('id', $doctorProfile->doctor\_specialization\_id)->first();
9. } catch (Exception $exception){
10. report($exception);
11. **return** Redirect::back()->with('message', 'Something went wrong during mail confirmation');
12. }
13. $date = Carbon::parse($appointment->appointment\_date)->format('d-m-Y');
14. $hour = Carbon::parse($appointment->appointment\_time)->format('H:i');
15. $email = $patientProfile->user->email;
16. $name = $patientProfile->user->name;
17. $doctorName = $doctorProfile->name;
18. $doctorSurname = $doctorProfile->surname;
19. $specializationName = $specialization->name;
21. Mail::to($email)->send(**new** AppointmentConfirmationMail($date, $hour, $name, $doctorName, $doctorSurname, $specializationName));
22. });
23. }

## Mechanizm edycji profilu

Lisint nr… prezentuje akcję aktualizacji profilu, w kontrolerze pacjenta. Metoda ta odpowiada za walidację danych przekazanych z formularza edycji profilu oraz zapisanie ich do bazy. Po poprawnym przebiegu tych czynności następuje przekierowanie do widoku profilu. Podobny mechanizm zastosowano w przypadku profilu doktora.

Listing 7 Mechanizm edycji profilu

1. **public** **function** updatePatient(){
2. $data = request()->validate([
3. 'name' => 'required|string|max:100',
4. 'surname'=> 'required|string|max:100',
5. 'phone'=> 'string|max:11',
6. 'pesel'=> 'string|max:11',
7. 'locality'=> 'string|max:50',
8. 'street'=> 'string|max:50',
9. 'birthDate'=> 'required|date|before:today',
10. 'image' => 'mimes:jpeg,png',
11. ]);
13. **if** (request('image')){
14. $imagePath = request('image')->store('uploads/patient-profiles', 'public');
16. $image = Image::make(public\_path("storage/{$imagePath}"))->fit(1000, 1000);
17. $image->save();
19. $imageArray = ['image' => $imagePath];
20. }
22. auth()->user()->patientProfile->update(array\_merge(
23. $data,
24. $imageArray ?? []
25. ));
27. **return** redirect('home/profile/patient');
28. }

Listing nr… przedstawia funkcję której rolą jest pobieranie profilu zdjęcia. W przypadku kiedy profil nie posiada ustawionego zdjęcia, metoda zwraca zdjęcie zastępcze.

Listing 8 Wyświetlanie zdjęcia profilowego

1. **public** **function** profileImage(){
2. $imagePath = ($this->image) ? $this->image : 'uploads/profile\_placeholder.jpg';
3. **return** '/storage/'. $imagePath;
4. }

## Mechanizm seedowania (wypełniania) bazy danych

Listing nr .. określa sposób wstępnie wypełniana zostaje baza danych przed uruchomieniem aplikacji. Funkcja „run()” odpowiedzialna jest ciągu funkcji wykonujących zapytania dla poszczególnych klas.

Listing 9 Mechanizm seedowania

1. **class** DatabaseSeeder **extends** Seeder
2. {
3. /\*\*
4. \* Seed the application's database.
5. \*
6. \* @return void
7. \*/
8. **public** **function** run()
9. {
10. $this->call(RolesTableSeeder::**class**);
11. $this->call(DoctorSpecializationSeeder::**class**);
12. $this->call(DiseaseSeeder::**class**);
13. $this->call(MedicineSeeder::**class**);
14. $this->call(DoctorDayOfWeekPlanSeeder::**class**);
16. // $this->call(UsersTableSeeder::class);
17. //        php artisan db:seed
18. //        php artisan make:seeder UsersTableSeeder
19. }
20. }

Na listingu nr .. przedstawiono przykładową metodę wykonującą zapytanie zapisu danych do bazy danych.

Listing 10 Mechanizm seedowania - metody insert

1. **class** RolesTableSeeder **extends** Seeder
2. {
3. /\*\*
4. \* Run the database seeds.
5. \*
6. \* @return void
7. \*/
8. **public** **function** run()
9. {
10. Role::truncate();
12. Role::create(['name' => 'ROLE\_ADMIN']);
13. Role::create(['name' => 'ROLE\_DOCTOR']);
14. Role::create(['name' => 'ROLE\_PATIENT']);
15. }
16. }

# Podsumowanie

Dyskusja nad dalszym rozwojem pracy. Wnioski. Omówienie wyników. Co zrobiono w pracy i jakie uzyskano wyniki? Czy i w jakim zakresie praca stanowi nowe ujęcie problemu? Sposób wykorzystania pracy (publikacja, udostępnienie instytucjom, materiał źródłowy dla studentów). Co uważa autor za własne osiągnięcia ?

 Opis własnymi słowami tego co zostało zrobione

 Opis tego jak można rozwinąć rozwiązanie w przyszłości

 Zdanie w którym należy stwierdzić, ze założony cel został osiągnięty (lub należy go uznać za osiągnięty)

# Streszczenie

Pracę rozpoczęto od …. W rozdziale I pracy zrobiono …, w kolejnym rozdziale umieszczono …, itd..

# Spis rysunków

[Rysunek 2.1 Schemat architektury aplikacji 8](file:///D:\Users\davej\Desktop\inzynierka-health-clinic\engineering-work\praca-inzynierska.docx#_Toc29244239)

[Rysunek 4.1 Diagram przypadków użycia użytkownika niezalogowanego 19](file:///D:\Users\davej\Desktop\inzynierka-health-clinic\engineering-work\praca-inzynierska.docx#_Toc29244240)

[Rysunek 4.2 Diagram przypadków użycia użytkownika zalogowanego (pacjenta) 20](file:///D:\Users\davej\Desktop\inzynierka-health-clinic\engineering-work\praca-inzynierska.docx#_Toc29244241)

[Rysunek 4.3 Diagram przypadków użycia użytkownika zalogowanego (doktora) 21](file:///D:\Users\davej\Desktop\inzynierka-health-clinic\engineering-work\praca-inzynierska.docx#_Toc29244242)

[Rysunek 4.4 Diagram procesu rejestracji 22](file:///D:\Users\davej\Desktop\inzynierka-health-clinic\engineering-work\praca-inzynierska.docx#_Toc29244243)

[Rysunek 4.5 Diagram procesu zmiany hasła 23](file:///D:\Users\davej\Desktop\inzynierka-health-clinic\engineering-work\praca-inzynierska.docx#_Toc29244244)

[Rysunek 4.6 Diagram procesu rezerwacji wizyty 24](file:///D:\Users\davej\Desktop\inzynierka-health-clinic\engineering-work\praca-inzynierska.docx#_Toc29244245)

[Rysunek 4.7 Diagram sekwencji rezerwacji wizyty 25](file:///D:\Users\davej\Desktop\inzynierka-health-clinic\engineering-work\praca-inzynierska.docx#_Toc29244246)

[Rysunek 4.8 Diagram sekwencji wystawiania recepty 26](file:///D:\Users\davej\Desktop\inzynierka-health-clinic\engineering-work\praca-inzynierska.docx#_Toc29244247)

[Rysunek 4.9 Schemat bazy danych 30](file:///D:\Users\davej\Desktop\inzynierka-health-clinic\engineering-work\praca-inzynierska.docx#_Toc29244248)

# Spis listingów

[Listing 1 Mechanizm rezerwacji – tworzenie widoku z formularzem sprawdzania daty 34](#_Toc29331029)

[Listing 2 Mechanizm rezerwacji wizyty - weryfikacja i pobranie godzin 35](#_Toc29331030)

[Listing 3 Mechanizm rezerwacji wizyty - wyświetlanie godzin 36](#_Toc29331031)

[Listing 4 Mechanizm rezerwacji wizyt - tworzenie rezerwacji 36](#_Toc29331032)

[Listing 5 Mechanizm wypisywania recepty 37](#_Toc29331033)

[Listing 6 Tworzenie i wysyłanie powiadomienia 38](#_Toc29331034)

[Listing 7 Mechanizm edycji profilu 38](#_Toc29331035)

[Listing 8 Wyświetlanie zdjęcia profilowego 39](#_Toc29331036)

[Listing 9 Mechanizm seedowania 39](#_Toc29331037)

[Listing 10 Mechanizm seedowania - metody insert 40](#_Toc29331038)

# Dodatek. Zawartość płyty CD

# Bibliografia

1. <https://www.oracle.com/pl/database/what-is-database.html> -styczeń 2020 [↑](#endnote-ref-1)
2. <https://www.oracle.com/pl/database/what-is-a-relational-database/> - styczeń 2020 [↑](#endnote-ref-2)
3. <https://www.oracle.com/pl/mysql/> - styczeń 2020 [↑](#endnote-ref-3)
4. <https://pl.wikipedia.org/wiki/MySQL> - styczeń 2020 [↑](#endnote-ref-4)
5. <https://pl.wikipedia.org/wiki/WampServer> [↑](#endnote-ref-5)
6. <http://www.wampserver.com/en/download-wampserver-64bits/> [↑](#endnote-ref-6)
7. <https://www.phpmyadmin.net/> - styczeń 2020 [↑](#endnote-ref-7)
8. <https://rk.edu.pl/pl/co-jest-php/> - styczeń 2020 [↑](#endnote-ref-8)
9. <https://jaki-jezyk-programowania.pl/technologie/php/> - styczeń 2020 [↑](#endnote-ref-9)
10. <https://www.phpdevs.pl/projekt-obiektowy-internetowy-notes/1-composer> - styczeń 2020 [↑](#endnote-ref-10)
11. <https://pl.wikipedia.org/wiki/Composer> - styczeń 2020 [↑](#endnote-ref-11)
12. <https://laravel.com/docs/5.0> - styczeń 2020 [↑](#endnote-ref-12)
13. <https://laravel.com/docs/5.0/artisan> - styczeń 2020 [↑](#endnote-ref-13)
14. <http://laravel-docs.pl/> - styczeń 2020 [↑](#endnote-ref-14)
15. <http://math.uni.lodz.pl/~kowalcr/Bazy/Temat8.pdf> - styczeń 2020 - styczeń 2020 [↑](#endnote-ref-15)
16. <https://cubettech.com/resources/blog/laravels-query-builder-and-eloquent-orm/> - styczeń 2020 - styczeń 2020 [↑](#endnote-ref-16)
17. <https://laravel.com/docs/6.x/telescope#introduction> - styczeń 2020 [↑](#endnote-ref-17)
18. <https://en.wikipedia.org/wiki/PhpStorm> - styczeń 2020 [↑](#endnote-ref-18)
19. <https://pl.wikipedia.org/wiki/HTML> - styczeń 2020 [↑](#endnote-ref-19)
20. <http://image.intervention.io/> - styczeń 2020 [↑](#endnote-ref-20)
21. <https://pl.wikipedia.org/wiki/Bootstrap_(framework)> - styczeń 2020 [↑](#endnote-ref-21)
22. <https://getbootstrap.com/docs/4.4/getting-started/introduction/> - styczeń 2020 [↑](#endnote-ref-22)
23. <https://pl.wikipedia.org/wiki/Git_(oprogramowanie)> - styczeń 2020 [↑](#endnote-ref-23)