

POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: AUTOMATYKA I ROBOTYKA (AIR)

SPECJALNOŚĆ: TECHNOLOGIE INFORMACYJNE W AUTOMATYCE (ART)

PRACA DYPLOMOWA
INŻYNIERSKA

Aplikacja desktopowa wspomagająca pracę
przychodni pediatrycznej.

Desktop application supporting the pediatric
outpatient clinic.

AUTOR:

Kacper Weiss

PROWADZĄCY PRACĘ:

Dr inż. Jarosław Mierzwa

OCENA PRACY:

Spis treści

1. Wprowadzenie	6
1.1. Wstęp	6
1.2. Cel i zakres pracy	6
1.3. Układ pracy	7
2. Istniejące rozwiązania	8
2.1. Medfile	8
2.2. e-Scanmed	9
3. Wymagania funkcjonalne i нефункционалне	10
3.1. Wymagania funkcjonalne	10
3.2. Wymagania нефункционалне	10
4. Zastosowane technologie i narzędzia	12
4.1. C#	12
4.2. XAML i WPF	13
4.3. Microsoft Visual Studio 2017	15
4.4. Microsoft SQL Server 2017	15
4.5. .NET Framework	15
4.6. Entity Framework	16
4.7. Git	16
5. Projekt i implementacja	18
5.1. Struktura projektu na dysku	18
5.1.1. Folder „DataBaseStuff”	20
5.1.2. Foldery „Enums” i „Exceptions”	23
5.1.3. Folder „Logics”	23

5.1.4. Folder „User Controlers”	23
5.2. Logowanie użytkowników	25
5.3. Rejestracja nowych użytkowników	26
5.4. Zmiana hasła	27
5.5. Dodawanie nowych pacjentów	28
5.6. Rejestracja wizyt pacjentów	30
5.7. Obsługa pacjentów i wystawianie druków	32
6. Testy aplikacji	35
6.1. Testy jednostkowe	35
6.2. Testy manualne	36
6.2.1. Test próby wpisania błędnego hasła	37
6.2.2. Test próby usunięcia jedyne administratora	37
6.2.3. Test próby przejścia do widoku szczegółowego pacjenta, bez wybrania wcześniej pacjenta z listy	38
7. Podsumowanie i wnioski	40
7.1. Możliwości rozwoju	41
Bibliografia	42
A. Instrukcja obsługi	43
A.1. Przed pierwszym uruchomieniem	43
A.2. Pierwsze kroki	43
A.3. Tworzenie kont nowych użytkowników	43
A.4. Usuwanie kont użytkowników	44
A.5. Edycja uprawnień użytkowników	44
A.6. Tworzenie nowych kont pacjentów	44
A.7. Edycja danych pacjenta	44
A.8. Dostęp do szczepień	45
A.9. Rejestrowanie nowych wizyt	45
A.10.Usuwanie i edycja opisu wizyt	45
A.11.Obsługa wizyty pacjenta przez lekarza	45

Spis rysunków

4.1. Proces uruchamiania oprogramowania w języku C# [5].	13
4.2. Rezultat przykładowego elementu interfejsu w XAML. Źródło: Strona internetowa Developer Notes [4].	14
4.3. Przykładowy fragment drzewa „commit’ów” z repozytorium „jira_clone” na stronie GitHub[2] z dnia 28.01.2020r.	17
5.1. Ogólna struktura projektu na dysku.	18
5.2. Okno główne bez wczytanej zawartości. Widok z designera VS2017	20
5.3. Zawartość folderu „DataBaseStuff”.	21
5.4. Wygląd <i>User Controlera</i> służącego do przeglądania listy pacjentów.	24
5.5. Wygląd <i>User Controlera</i> służącego do logowania użytkownika do systemu.	25
5.6. Wygląd okna z wczytaną zakładką „ <i>UserManagementTab</i> ”.	27
5.7. Wygląd okna z wczytaną zakładką „ <i>PatientsTab</i> ” i widokiem „ <i>Nowy Pacjent</i> ”.	29
5.8. Wygląd okna z wczytaną zakładką „ <i>PatientsTab</i> ” na jego głównym widoku.	31
6.1. Rezultat testów jednostkowych dla <i>PasswordGeneratorTest</i>	35
6.2. Rezultat testu manualnego próby wpisania błędnego hasła.	37
6.3. Rezultat testu manualnego dla próby usunięcia jedyne administratora.	38
6.4. Rezultat testu manualnego dla próby przejścia do widoku szczegółowego pacjenta, bez wybrania wcześniej pacjenta z listy.	39

Spis listingów

4.1. Przykład treści pliku „StackPanel.xaml”.	
Źródło: Strona internetowa Developer Notes [4].	14
5.1. Kod zamieszczony w pliku „App.xaml” aplikacji.	19
5.2. Kod zamieszczony w pliku „packages.config” aplikacji.	19
5.3. Fragment pliku „DbEntity.cs”.	21
5.4. Kod zamieszczony w pliku „DataBaseContext.cs” aplikacji.	22
5.5. Jedna z metod statycznych klasy „CustomEnumToString”.	23
5.6. Przykładowe użycie <i>User Controller’a</i> w elemencie typu <ListView>.	24
5.7. Kod metody „LoginButton_Click” będącej wydarzeniem w „LoginForm”.	26
5.8. Kod metody „ChangePasswordButton_Click” będącej wydarzeniem w „SettingsForm”.	28
5.9. Kod metody „AddPatientButton_Click” będącej wydarzeniem w „PatientsTab”.	30
5.10. Fragment kodu odpowiedzialnego za wybór pacjenta.	31
5.11. Fragment kodu metody odpowiadającej za rejestrację wizyt pacjentów.	32
5.12. Fragment kodu odpowiedzialnego za wybór pacjenta.	33
5.13. Kod metody „FinishVisitButton_Click” kończącej wizytę pacjenta.	34
6.1. Klasa testowa „PasswordGeneratorTest” z dwoma przykładowymi testami.	36

Rozdział 1

Wprowadzenie

1.1. Wstęp

W postępującej epoce cyfryzacji, coraz więcej różnych instytucji i przedsiębiorstw zaczyna korzystać z wszelkiego rodzaju oprogramowania wspomagającego, lub automatyzującego pracę i służącego do bezpiecznego przechowywania danych niezbędnych dla biznesu. Zwiększenie znaczenia i zainteresowania rozwiązaniami IT zarówno ze strony klientów, jak i inżynierów i developerów z całego świata zmieniło sposób w jaki funkcjonują wszyscy. Zaczynając od pojedynczych użytkowników, po działalności wielkich korporacji. Rozwiązania takie znajdziemy m.in. w takich dziedzinach jak:

- telekomunikacja,
- transport,
- automatyka budynkowa,
- rekreacja,
- edukacja,
- medycyna,
- i wiele innych.

1.2. Cel i zakres pracy

Celem pracy jest projekt i implementacja aplikacji desktopowej służącej do wspomagania pracy przychodni pediatrycznej, tj. aplikacji typu klient-serwer umożliwiającej przetwarzanie i przechowywanie danych pacjentów, dziecięcej dokumentacji medycznej, a także wystawianie

wszelkiego rodzaju druków takich jak recepty, czy skierowania. Projekt został zrealizowany wyłącznie z myślą o systemie operacyjnym Windows, z tego powodu oprogramowanie napisane zostało w języku C# wykorzystującym platformę .NET. Z tego powodu niezbędne było zapoznanie się z metodami rozwoju oprogramowania na platformie .NET, zdobycie podstawowej wiedzy z zakresu tworzenia baz danych z wykorzystaniem takich narzędzi dostępnych na tej platformie, a także struktury takiego oprogramowania. Do realizacji pracy konieczne było opanowanie narzędzi takich jak:

- środowisko programistyczne Microsoft Visual Studio 2017,
- Material Design Themes dla Windows Presentation Foundation,
- metodyka tworzenia baz danych Code First w Entity Framework 6.

Ponadto należało opracować sposób realizacji wszystkich założeń projektu w sposób jak najbardziej spójny i otwarty na dalsze rozszerzanie jego funkcjonalności. Do wykonania projektu kluczowa okazała się wiedza i umiejętności nabyte w trakcie trwania studiów, m.in. z dziedziny baz danych, oraz programowania w językach wysokiego poziomu.

1.3. Układ pracy

Praca składa się z sześciu rozdziałów. W pierwszym rozdziale przedstawione zostały wprowadzenie do tematu, cel, zakres i układ pracy. Rozdział drugi poświęcony został opisowi już istniejących rozwiązań. W rozdziale trzecim przedstawione zostały wymagania funkcjonalne i нефункционалне projektu. Rozdział czwarty to omówienie zastosowanych w trakcie pracy technologii i narzędzi. Rozdział piąty opisuje strukturę projektu, oraz sposób jego implementacji. W szóstym rozdziale przedstawione zostały testy aplikacji wraz z ich rezultatami. W ostatnim siódmym rozdziale zamieszczone zostało podsumowanie wykonanej pracy, wnioski z zakończonej pracy, a także dalsze możliwości rozwoju projektu. Na koniec zostały zamieszczone spis literatury, a także załącznik w postaci instrukcji obsługi.

Rozdział 2

Istniejące rozwiązania

W ramach pracy dyplomowej, jednym z zadań miało być zapoznanie się z istniejącymi rozwiązaniami. Poniżej pozwoliłem sobie opisać przykładowe dwie aplikacje.

2.1. Medfile

Medfile [6] jest to usługa, która łączy w jednym miejscu dostęp do takich funkcji jak:

- Elektroniczna Dokumentacja Medyczna (EDM)
- e-Recepta
- e-ZLA - wystawianie elektronicznych zwolnień lekarskich
- zestandaryzowane karty historii chorób pacjenta
- a także wiele innych.

Jej największą zaletą jest dostęp do wszystkich potrzebnych usług w jednym miejscu, zarówno z poziomu komputera, jak i urządzeń mobilnych. Wszelkie dane przechowywane są w chmurze, a aplikacja spełnia wszystkie wymagania prawne dotyczące ochrony danych osobowych pacjentów. Dodatkowym atutem jest wersja bezpłatna dla małych gabinetów w których miałby istnieć tylko 1 użytkownik. A także płatne pakiety pozwalające na korzystanie ze znacznie większej puli funkcjonalności, takich jak nielimitowana liczba wizyt przechowywanych w chmurze, zwiększenie rozmiaru dostępnego na przechowywanie załączników, oraz możliwość tworzenia własnych formularzy.

2.2. e-Scanmed

Jest aplikacją internetową firmy Scanmed [3]. Pozwala ona pacjentom na samodzielną rejestrację wizyt w wybranych przez nich przychodniach, u wybranych lekarzy i odpowiedniej dla nich porze. Daje ona dostęp zalogowanym pacjentom do swojej historii zdrowia, sprawdzenie informacji o lekach a także wiele innych funkcji. Użytkownik może za jej pomocą samodzielnie wykonywać podstawowe czynności bez konieczności angażowania recepcjonisty.

Rozdział 3

Wymagania funkcjonalne i niefunkcjonalne

Podstawowe cele projektu pozwoliłem sobie przedstawić w formie tabel, a założenia podzieliłem na wymagania funkcjonalne i wymagania niefunkcjonalne.

3.1. Wymagania funkcjonalne

Poniższe wymagania definiują podstawowe funkcje programu, jakie na projekt powinien móc wykonywać w chwili zakończenia pracy nad nim, aby mógł zostać uznany za zakończony. Wymagania te pozwalają nam również na określenie przypadków użycia aplikacji.

- tworzenie kont nowych pracowników
- logowanie pracowników do systemu
- przechowywanie dokumentacji medycznej pacjentów
- rejestracja wizyt
- dodawanie nowych usług do oferty
- wystawianie druków takich jak recepty i skierowania

3.2. Wymagania niefunkcjonalne

Poniższe wymagania definiują podstawowe wymagania techniczne, ograniczenia przy których system musi nadal spełniać swoje podstawowe funkcje, a także jakie udogodnienia może oczekiwać użytkownik ze strony aplikacji.

- poprawnie funkcjonowanie oprogramowania na urządzeniach działających w systemie Windows
- logowanie i rejestracja użytkowników muszą być przeprowadzone w sposób bezpieczny
- interfejs graficzny musi być intuicyjny i łatwy w użyciu
- aplikacja musi być w miarę możliwości zoptymalizowana na tyle, aby użytkownik nie odczuwał dyskomfortu
- bezpieczna obsługa błędów i wyjątków
- bezpieczny dostęp do danych
- całość oprogramowania implementowana za pomocą platformy .NET w języku C#
- aplikacja musi wykorzystywać niekomercyjną bazę danych

Rozdział 4

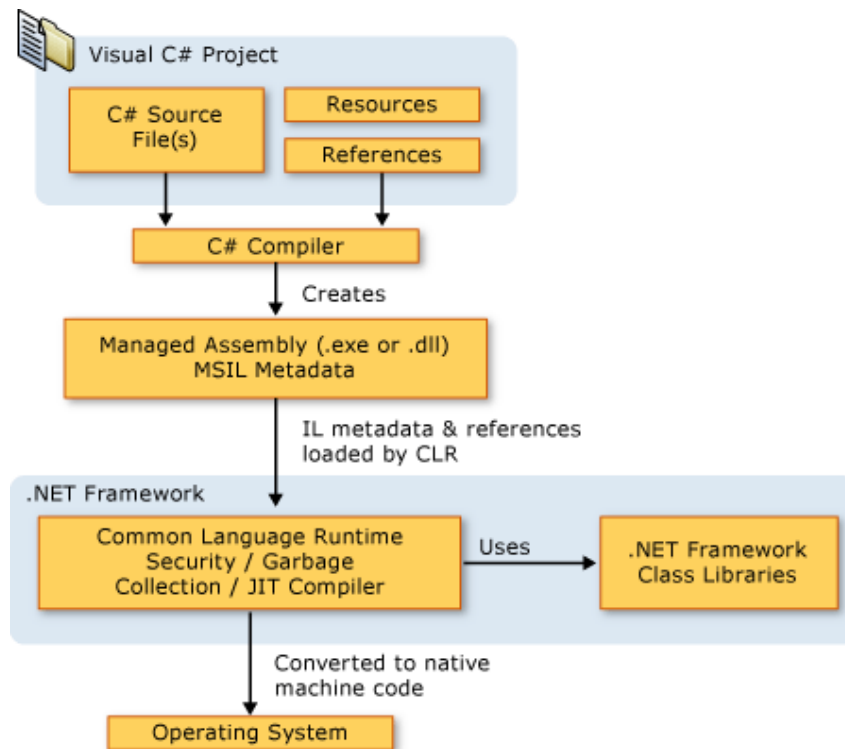
Zastosowane technologie i narzędzia

W tym rozdziale opisane są wszelkiego rodzaju programy, narzędzia i technologie, które zostały wykorzystane w procesie tworzenia aplikacji. Zostanie opisana platforma .NET, usługa Microsoft SQL Server 2017 w wersji Developer w której skład wchodzi użyta baza danych, a także narzędzia developerskie takie jak Microsoft SQL Server Management Studio 17. Krótko zostanie zaprezentowane zintegrowane środowisko developerskie Microsoft Visual Studio 2017 w wersji Community, wykorzystywany tam język programowania C# i język opisu interfejsu użytkownika eXtensible Application Markup Language (XAML).

4.1. C#

Cała logika aplikacji została wykonana w języku C#. C# jest powszechnie znanym językiem programowania wysokiego poziomu, zorientowanym wokół programowania obiektowego. Jego pierwsza wersja powstała na pod koniec lat dziewięćdziesiątych ubiegłego wieku, a projekt pod okiem Andersa Hejlsberga początkowo nosił nazwę COOL, która miała oznaczać wtedy "Ć like Object Oriented Language". Ostatecznie nazwę tę zmieniono na C# w celu podkreślenia jego korzeni w języku C++, a wybór znaku # był spowodowany jego podobieństwem do 4 znaków "+". Obecnie język C# coraz prężniej się rozwija, m.in. w ramach części jej otwarto-źródłowej implementacji zwanej .NET Core, która poza Windowsem pozwala tworzyć aplikacje również na platformy Linux, oraz macOS, czyniąc język C# coraz bardziej uniwersalnym narzędziem.

Do poprawnego uruchomienia, działania, czy tworzenia aplikacji pisanych w języku C# konieczne jest posiadanie platformy .NET Framework, która obecnie jest integralnym składnikiem systemu Windows, lub w wypadku systemu Linux i macOS platformy .NET Core. Dzieje się tak ponieważ kod napisany w języku C# jest kompilowany do CIL (Common Intermediate Language), który następnie jest uruchamiany przez CLR (Common Language Runtime) i interpretowany przez kompilator JIT (Just-In-Time), a tam po konwersji na natywny kod maszynowy, program wykonywany jest już przez system operacyjny.



Rys. 4.1: Proces uruchamiania oprogramowania w języku C# [5].

4.2. XAML i WPF

Język XAML oparty jest w dużej mierze o język XML. Jest on wykorzystywany do tworzenia interfejsu użytkownika w technologii WPF (Windows Presentation Foundation). Wykorzystanie języka XAML umożliwia developerom rozdzielenie kodu logicznego, od definicji kodu źródłowego. Dzięki postawieniu granicy pomiędzy interfejsem, a logiką uzyskuje się większą spójność i czytelność kodu. Kod odpowiadający za logikę często znajduje się w tym samym module dzięki zastosowaniu plików code behind. Poszczególne elementy interfejsu można bindować z odpowiednimi danymi, które przy ustawieniu DataContext na odpowiednie okno, oraz wykorzystanie odpowiednich zdarzeń, będzie na bieżąco aktualizowane z niewielkim kosztem

dla aplikacji. Pracę z językiem XAML ułatwiają różne narzędzia wchodzące w skład Microsoft Visual Studio m.in. Microsoft Blend. Umożliwia to pracę nad interfejsem zarówno za pomocą designera graficznego, jak i edytora tekstowego z bezpośrednim podglądem ostatecznego wyglądu interfejsu aplikacji. Na listingu 4.1 zamieściłem przykładowy element interfejsu napisany w języku XAML.

```

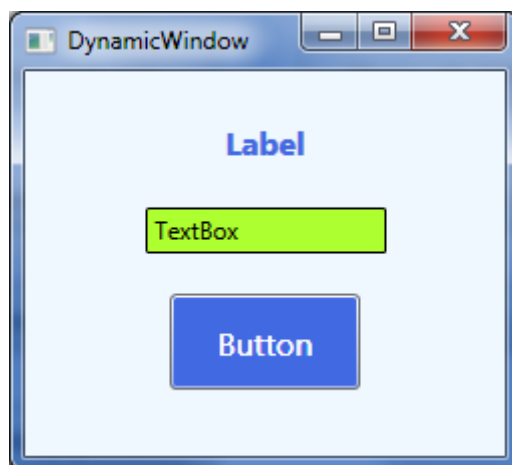
1 <StackPanel xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
2   Name="stackPanel1" Orientation="Vertical" Background="#FFF0F8FF">
3   <Label Name="label1" Foreground="#FF4169E1" FontSize="16" FontWeight="Bold"
4     HorizontalContentAlignment="Center" Width="80" Height="28" Margin="0,20,0,0">
5     Label
6   </Label>
7   <TextBox Name="textBox1" BorderBrush="#FF000000" Background="#FFADFF2F"
8     Width="120" Height="23" Margin="0,20,0,0">
9     TextBox
10  </TextBox>
11  <Button Name="button1" Background="#FF4169E1" Foreground="#FFFFFFF"
12    FontSize="16" Width="95" Height="48" Margin="0,20,0,0">
13    Button
14  </Button>
15 </StackPanel>

```

Listing 4.1: Przykład treści pliku „StackPanel.xaml”.

Źródło: Strona internetowa Developer Notes [4].

Poniżej zamieściłem rysunek 4.2 przedstawiający utworzony za pomocą powyższego kodu element interfejsu.



Rys. 4.2: Rezultat przykładowego elementu interfejsu w XAML.

Źródło: Strona internetowa Developer Notes [4].

4.3. Microsoft Visual Studio 2017

Microsoft Visual Studio jest zintegrowanym środowiskiem programistycznym, jego pierwsze wersje powstały już pod koniec XX wieku. Microsoft Visual Studio 97 znacznie różni się od wersji użytej w projekcie, ponieważ to IDE powstało jeszcze przed powstaniem platformy .NET, a na celu miała przede wszystkim połączenie istniejących już środowisk w jedno uniwersalne. Wersja Visual Studio 7.0 z 2002 roku wprowadziła platformę .NET jako podstawę działania środowiska, co już zostało do najnowszych wersji. Obecnie najnowsza wersja Visual Studio 2019, która różni się od Visual Studio 2017 bardzo niewielkim stopniem, są to zmiany głównie co do wersji używanych narzędzi, tj. zaktualizowane zostały .NET Core do wersji 3.0, F# do wersji 4.6, a także C# do wersji Preview 8.0. Visual Studio jest środowiskiem które udostępnia użytkownikowi wiele opcji szybkiej edycji, refaktoryzacji kodu, oraz bardzo wygodny, zintegrowany debugger i narzędzia pozwalające na bardzo wygodną pracę w języku C#. Poza wsparciem dla języka C# środowisko to wspiera m.in.:

- Microsoft Visual Basic,
- Microsoft Visual C++,
- Microsoft Visual J#,
- Microsoft Visual F#,
- a także wiele innych po zainstalowaniu dodatkowych zasobów np. JavaScript.

4.4. Microsoft SQL Server 2017

Microsoft SQL Server (MS SQL) jest głównym bazodanowym produktem firmy Microsoft. Pełni funkcje systemu zarządzania bazą danych, a jego charakterystyczną cechą jest wykorzystanie języka zapytań Transact-SQL stanowiącego rozwinięcie standardu ANSI/ISO. Istnieją wersje tego oprogramowania przeznaczone do zastosowań zarówno komercyjnych, jak i niekomercyjnych, w zależności od wykorzystywanej przez użytkownika edycji programu. Od wersji 2005 wraz z edycją Developer możemy pobrać graficzne narzędzia do obsługi bazy danych, takie jak Microsoft SQL Server Management Studio.

4.5. .NET Framework

.NET Framework jest platformą programistyczną stworzoną i zaprojektowaną przez firmę Microsoft. Obejmuje ona środowisko uruchomieniowe CLR (Common Language Runtime)

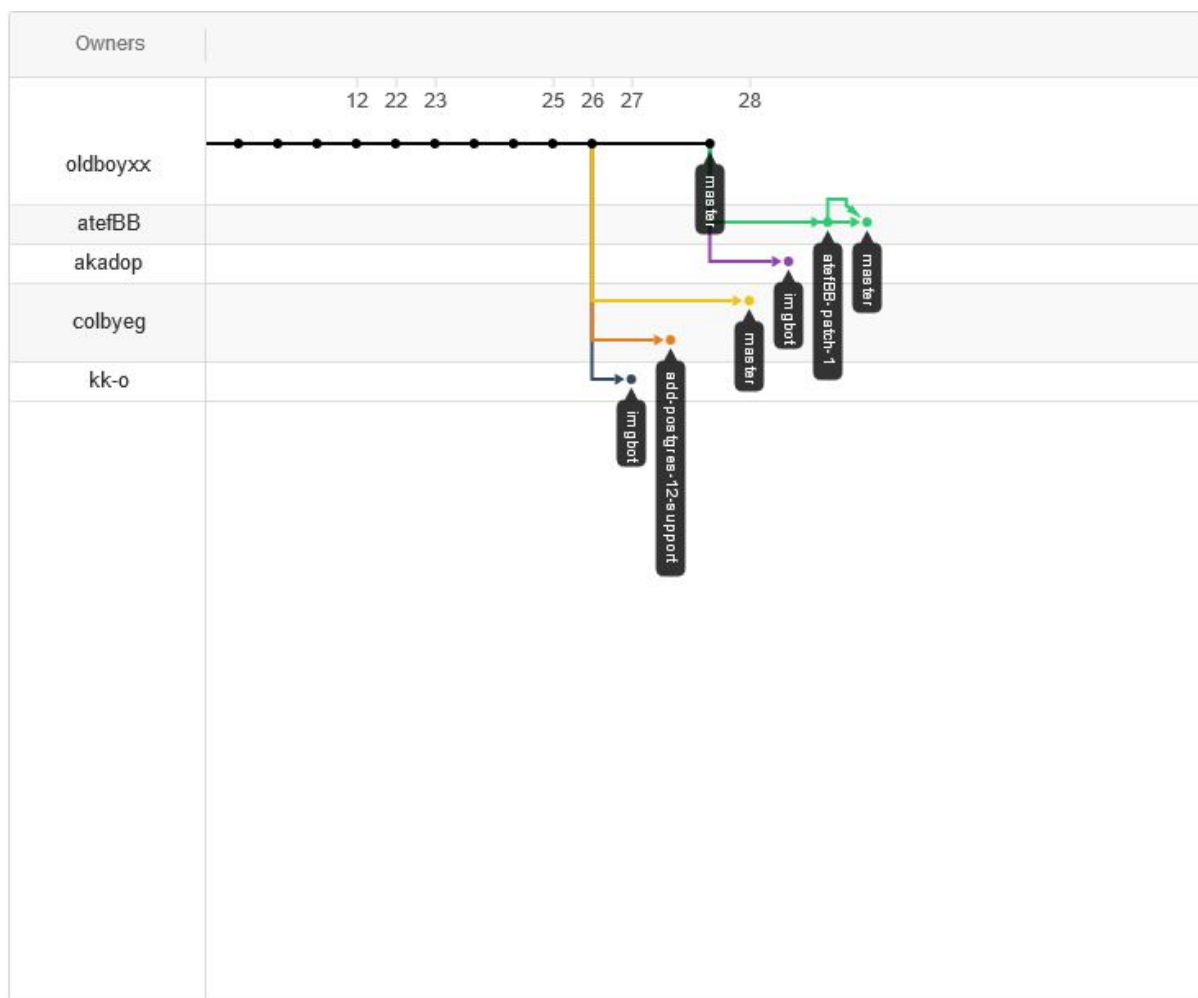
które zarządza aplikacjami przeznaczonymi na tą platformę. Zapewnia ono odpowiednie zarządzanie pamięcią, usługi systemowe takie jak garbage collector, szeroki wybór wbudowanych bibliotek klas dających deweloperom dostęp do zaufanego, zoptymalizowanego kodu dla większości dziedzin rozwoju oprogramowania. Istotną zaletą środowiska platformy .NET jest interoperacyjność języków programowania. Znaczy to, że kod pisany w językach kompilowalnych do CIL (Common Intermediate Language) mogą ze sobą współpracować w ramach platformy za pomocą narzędzi i technologii takich jak między innymi COM Interop.

4.6. Entity Framework

Entity Framework (EF) jest zbiorem technologii ORM (object-relational mapping) znajdującym się w komponencie ADO.NET. Wspiera on aplikacje komputerowe zorientowane wokół przechowywania i przetwarzania danych. Udostępnia on wiele narzędzi dla deweloperów ułatwiających i przyspieszających produkcje oprogramowania służącego do bezpiecznej pracy z bazami danych. Entity Framework od wersji 4.1 wspiera podejście Code-First służące do wygodnego tworzenia wszelkich struktur zaczynając od napisania klas zwanych „Entities”, które następnie w połączonej bazie danych zostają odzwierciedlane poprzez komunikacje za pomocą stworzonej przez dewelopera klasy kontekstu dziedziczącej wszelkie funkcje z klasy DbContext. Podejście Code-First zostało wykorzystane w projekcie, z tego powodu wygenerowana baza danych ma strukturę bardzo prostą, charakterystyczna dla tej metodyki.

4.7. Git

Niezbędnym narzędziem dla każdego projektu zajmującym się wytwarzaniem oprogramowania jest system kontroli wersji. W tej pracy zdecydowałem się użyć systemu kontroli wersji Git stworzonego przez Linusa Torvaldsa. Jest to oprogramowanie typu Open-Source. Praca z tym systemem jest bardzo wygodna, pomaga usystematyzować pracę nad projektem, a także przy wykorzystywaniu serwisów takich jak GitHub zapewnia bezpieczne miejsce do przechowywania danych, a także historii zmian w kodzie, czy tzw. „commit’ów”. Git sprawdza się zarówno w pracy samodzielnej jak i zespołowej. Istnienie tzw. „branch’y” pozwala na przechowywanie niezależnych od siebie gałęzi zmian kodu. Poniżej pozwoliłem sobie zamieścić przykładową strukturę „branch’y” (na rys. 4.3). W trakcie pracy nad kodem możemy w dowolnej chwili wrócić do wybranego „commit’a”, np. w przypadku gdy podejmiemy decyzję o zmianie



Rys. 4.3: Przykładowy fragment drzewa „commit’ów” z repozytorium „jira_clone” na stronie GitHub[2] z dnia 28.01.2020r.

podejścia do sposobu rozwiązania jakiegoś problemu w projekcie. Git udostępnia wiele komend z których może korzystać deweloper, należą do nich:

- git init - służący do inicjalizacji repozytorium,
- git add - dodawanie plików obsługiwanych przez kontrolę wersji w projekcie,
- git commit - zapisanie zmian w repozytorium,
- git push - wysyłanie zmian do zdalnego repozytorium,
- git pull - ściąganie zmian ze zdalnego repozytorium,
- git log - wyświetlanie historii zmian repozytorium.

Oprogramowanie to można za darmo pobrać z oficjalnej strony wraz z narzędziami wspomagającymi przyszłą pracę z tym systemem kontroli wersji dla dowolnej platformy.

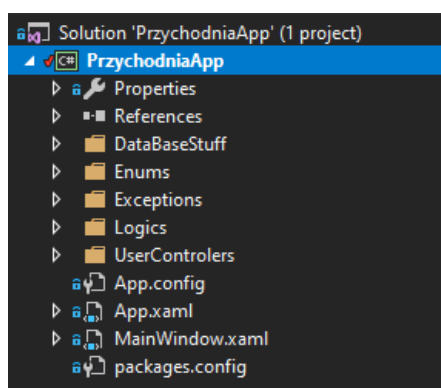
Rozdział 5

Projekt i implementacja

Rozdział ten przeznaczony jest do zaznajomienia czytelnika z projektem, jego działaniem i implementacją. Szczególna uwaga zostanie poświęcona aspektom funkcjonalnym aplikacji („Back-End”), oraz części wizualnej projektu („Front-End”). Opis techniczny rozpocznę od omówienia struktury projektu, najważniejszych jego komponentów - folderów i plików, a następnie przejdę do bardziej szczegółowych opisów funkcjonalności.

5.1. Struktura projektu na dysku

Projekt prowadzony był przy jak najbliższym przestrzeganiu ogólnie przyjętych standardów, które można odnaleźć na oficjalnej stronie z dokumentacją sporządzoną przez firmę Microsoft [1]. Ze względu na dotychczasowe stosunkowo niewielkie doświadczenie o charakterze wyłącznie akademickim, zdecydowałem się na nie wprowadzanie do projektu zbędnych komplikacji, a projekt zrealizowałem w sposób jak najbardziej uporządkowany. Najbardziej ogólną strukturę projektu przedstawiam na rys. 5.1.



Rys. 5.1: Ogólna struktura projektu na dysku.

W głównym folderze projektu znajdują się, zgodnie z konwencją:

- „App.config” które w wypadku mojej aplikacji konfiguruje przede wszystkim Entity Framework, wraz jego wersją i połączeniem do bazy danych,
- „App.xaml” określające nazwę pliku z głównym oknem aplikacji, oraz zasobami które załączymy np. „MaterialDesignThemes” do użycia w plikach XAML aplikacji, kod ten zaprezentowałem na listingu 5.1

```

1 <Application x:Class="PrzychodniaApp.App"
2     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4     xmlns:local="clr-namespace:PrzychodniaApp"
5     StartupUri="MainWindow.xaml">
6     <Application.Resources>
7         <ResourceDictionary>
8             <ResourceDictionary.MergedDictionaries>
9                 <ResourceDictionary
10                     ↪ Source="pack://application:,,,/MaterialDesignThemes.Wpf;component/Themes/
11                     ↪ MaterialDesignTheme.Light.xaml" />
12                 <ResourceDictionary
13                     ↪ Source="pack://application:,,,/MaterialDesignThemes.Wpf;component/Themes/
14                     ↪ MaterialDesignTheme.Defaults.xaml" />
15                 <ResourceDictionary
16                     ↪ Source="pack://application:,,,/MaterialDesignColors;component/Themes/
17                     ↪ Recommended/Primary/MaterialDesignColor.DeepPurple.xaml" />
18                 <ResourceDictionary
19                     ↪ Source="pack://application:,,,/MaterialDesignColors;component/Themes/
20                     ↪ Recommended/Accent/MaterialDesignColor.Lime.xaml" />
21             </ResourceDictionary.MergedDictionaries>
22         </ResourceDictionary>
23     </Application.Resources>
24 </Application>

```

Listing 5.1: Kod zamieszczony w pliku „App.xaml” aplikacji.

- „packages.config” określające wersje użytych bibliotek, czyli tzw. „paczek”, co przedstawiłem na listingu 5.2

```

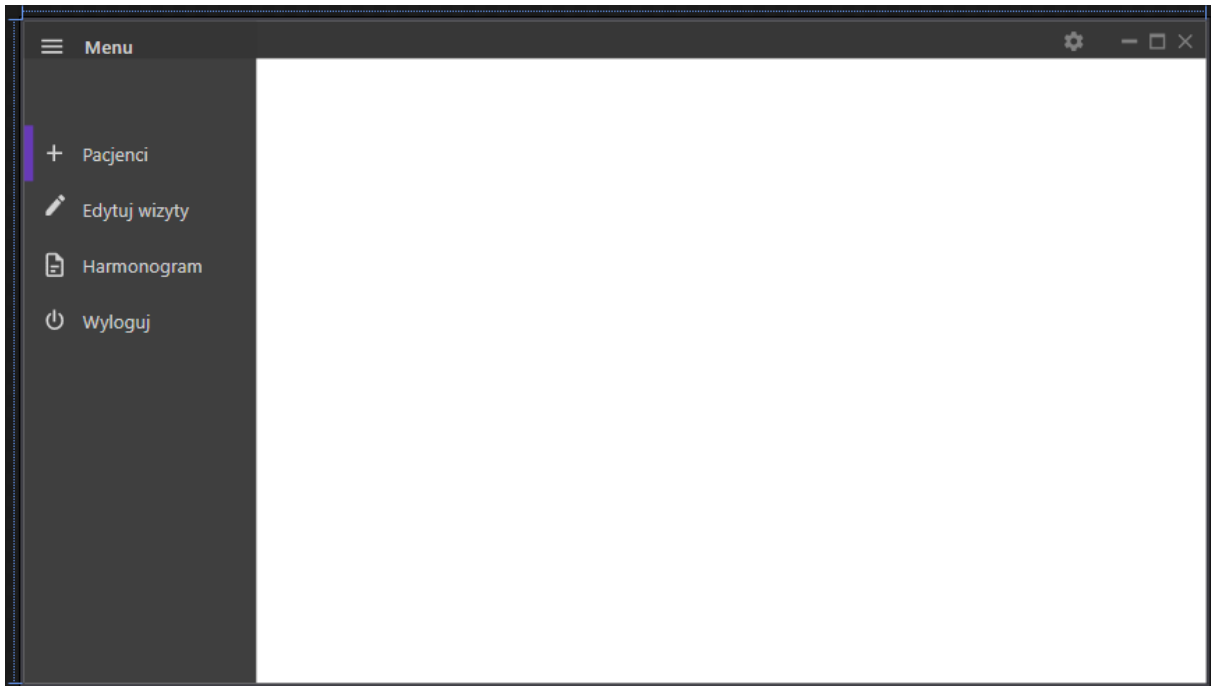
1 <?xml version="1.0" encoding="utf-8"?>
2 <packages>
3     <package id="EntityFramework" version="6.4.0" targetFramework="net462" />
4     <package id="MaterialDesignColors" version="1.2.2" targetFramework="net462" />
5     <package id="MaterialDesignThemes" version="3.0.1" targetFramework="net462" />
6     <package id="TimePeriodLibrary.NET" version="2.1.1" targetFramework="net462" />
7 </packages>

```

Listing 5.2: Kod zamieszczony w pliku „packages.config” aplikacji.

- „*MainWindow.xaml*” będące oknem głównym aplikacji,

Okno główne „*MainWindow.xaml*”, jest to plik z kodem XAML, korzystający także z tzw. „*Code-Behind*” napisanym w języku C#. Klasa „*MainWindow*” definiuje pierwsze funkcje, czy aktywności udostępniane użytkownikom, jest swoistym punktem wejścia do programu. Okno główne początkowo wyświetla ekran logowania, a następnie po zalogowaniu użytkownika wyświetla odpowiednie dla niego menu, które bez zawartości wygląda w designerze w sposób przedstawiony na rys. 5.2

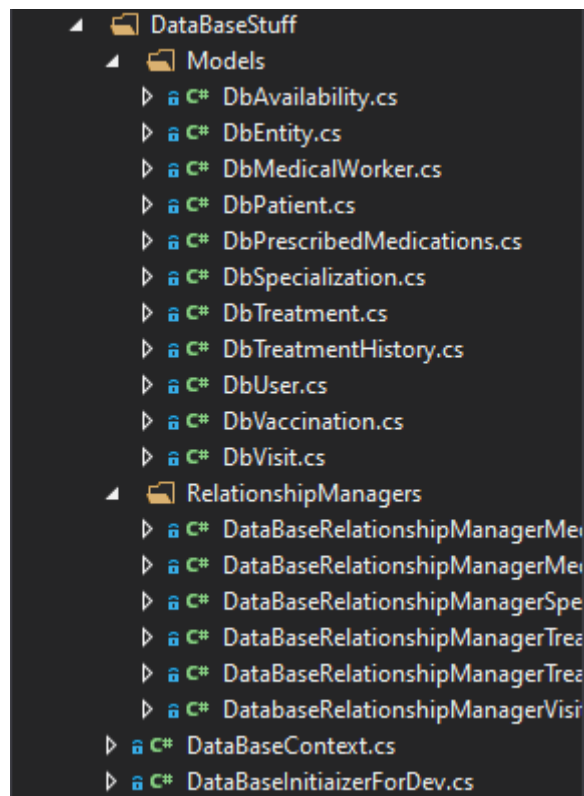


Rys. 5.2: Okno główne bez wczytanej zawartości. Widok z designera VS2017

Pozostałe pliki aplikacji zostały zamieszczone w odpowiednich folderach, które zostały omówione w poniższych podsekcjach.

5.1.1. Folder „*DataBaseStuff*”

W folderze „*DataBaseStuff*” zostały zamieszczone stworzone przez kreator ADO.NET Entity Data Model komponenty odpowiadające za kontekst bazy danych Entity Framework 6 dla podejścia Code First, modele przedstawiające oczekiwane struktury znajdujące się później w bazie danych, a także statyczne klasy służące do modyfikowania relacji pomiędzy nimi. Zawartość tego folderu została przedstawiona na rys. 5.3

Rys. 5.3: Zawartość folderu „*DataBaseStuff*”.

Folder „*Models*” zawiera modele „*Entities*” z których Entity Framework generuje bazy danych i za pomocą których później się z nią komunikuje. W moim projekcie każdy z modeli posiada przedrostek „*Db-*” w celu podkreślenia, że są to klasy służące bezpośrednio do pracy z bazą danych, a także każda z nich dziedziczy po abstrakcyjnej klasie „*DbEntity*” (przedstawionej na listingu 5.3) zawierającej tylko „*Id*”. Miało to na celu ułatwienie wykorzystania w późniejszej fazie implementacji z klas i metod szablonowych.

```
public abstract class DbEntity
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }
}
```

Listing 5.3: Fragment pliku „*DbEntity.cs*”.

Plik o nazwie „*DataBaseContext.cs*” definiuje nazwę utworzonej przez Entity Framework bazy danych, tabele jakie zostaną w niej umieszczone, oraz w pozwala na wybór inicjalizatora bazy danych w konstruktorze, który może być jednym z trzech podstawowych:

- „*CreateDatabaseIfNotExists*” - który jest domyślnym inicjalizatorem, który jest ustawiany automatycznie i tworzy bazę danych tylko w wypadku gdy ona nie istnieje,
- „*DropCreateDatabaseIfModelChanges*” - jest inicjalizatorem który będzie burzył i od nowa tworzył bazę danych, za każdym razem gdy zmieni się którykolwiek z modeli,
- „*DropCreateDatabaseAlways*” - jest inicjalizatorem który przy każdym uruchomieniu aplikacji będzie burzył i tworzył bazę danych na nowo.

Inicjalizator może zostać też stworzony przez Developera, za pomocą dziedziczenia jednego z powyższych inicjalizatorów i na przykład przeciążenie metody „*Seed*”. W przypadku tego projektu w trakcie rozwoju i testów aplikacji wykorzystany został inicjalizator znajdujący się w pliku o nazwie „*DataBaseInitializerForDev.cs*” który dziedziczy po „*DropCreateDatabaseIfModelChanges*”. Na poniższym listingu 5.4 pozwoliłem sobie zamieścić kod omawianego wcześniej pliku „*DataBaseContext.cs*”.

```
namespace PrzychodniaApp.DataBaseStuff
{
    using PrzychodniaApp.DataBaseStuff.Models;
    using System;
    using System.Data.Entity;
    using System.Linq;

    public class DataBaseContext : DbContext
    {
        public DataBaseContext() : base("name=MainContext")
        {
            Database.SetInitializer(new DataBaseInitiaizerForDev());
        }

        public virtual DbSet<DbAvailability> Availabilities { get; set; }
        public virtual DbSet<DbMedicalWorker> MedicalWorkers { get; set; }
        public virtual DbSet<DbPatient> Patients { get; set; }
        public virtual DbSet<DbPrescribedMedications> PrescribedMedications { get; set; }
        public virtual DbSet<DbSpecialization> Specializations { get; set; }
        public virtual DbSet<DbTreatment> Treatments { get; set; }
        public virtual DbSet<DbTreatmentHistory> TreatmentHistories { get; set; }
        public virtual DbSet<DbVaccination> Vaccinations { get; set; }
        public virtual DbSet<DbVisit> Visits { get; set; }

        public virtual DbSet<DbUser> Users { get; set; }
    }
}
```

Listing 5.4: Kod zamieszczony w pliku „*DataBaseContext.cs*” aplikacji.

5.1.2. Foldery „Enums” i „Exceptions”

Folder „Enums” zawiera wszystkie „Enum’y” jakie zostały wykorzystane przy tworzeniu aplikacji, a także klasę statyczną „CustomEnumToString”, której statyczne funkcje służą do zwracania ciągu znaków „string” w języku Polskim, jak pokazano w listingu 5.5 poniżej. Podobnie, folder „Exceptions” służy do przechowywania własnych konfigurowalnych wyjątków aplikacji.

```
public static string GetEmailContactText(EmailContact contact)
{
    switch (contact)
    {
        case EmailContact.Full:
            return "Pełny kontakt";

        case EmailContact.ConfirmationsAndReminders:
            return "Przypomnienia";

        case EmailContact.OnlyForConfirmations:
            return "Potwierdzenia";

        case EmailContact.No:
            return "Brak";

        default:
            return "";
    }
}
```

Listing 5.5: Jedna z metod statycznych klasy „CustomEnumToString”.

5.1.3. Folder „Logics”.

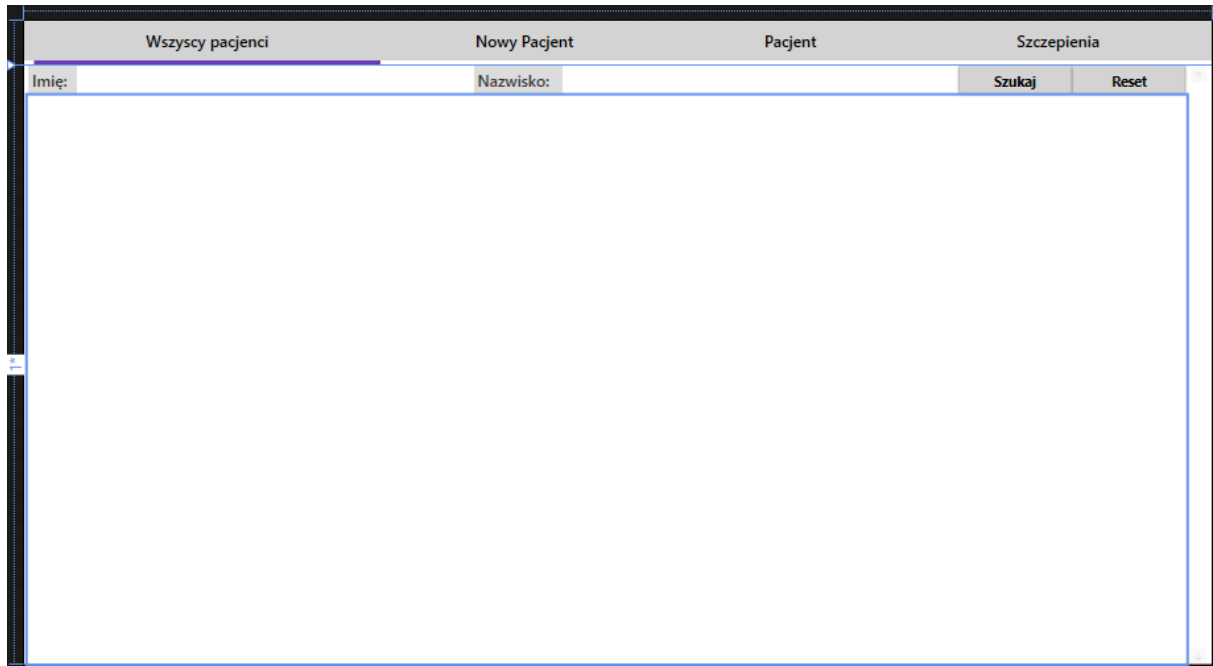
W tym folderze zostały zamieszczone pomniejsze klasy statyczne odpowiedzialne za wykonywanie jakichś pomniejszych funkcji w aplikacji, takich jak np generowanie w pełni losowych ciągów znaków np. do wykorzystania przy tworzeniu haseł.

5.1.4. Folder „User Controlers”.

Folder „User Controlers” jest przeznaczony do przechowywania tzw. „User Controler’ów”, czyli elementów charakterystycznych dla projektów pisanych za pomocą Windows Presentation Foundations. Są one komponentami za pomocą których można stworzyć pomniejsze elementy okien, szablon do tworzenia z jakichś struktur danych pozycji w listach znajdujących się w

WPF’ie, albo po prostu rozdzielić elementy interfejsu na mniejsze, łatwiejsze do zaprojektowania komponenty. W moim projekcie podzieliłem je na 2 typy znajdujące się w folderach o odpowiadającej im nazwie:

- „*Tabs*” będące mniejszym komponentem głównego okna aplikacji, zawierają one interfejs użytkownika potrzebny przy korzystaniu z poszczególnych zakładek z menu, a także implementują odpowiednie dla nich interakcje. Wygląd przykładowego *User Controler* z tego folderu zamieściłem na rys. 5.4,



Rys. 5.4: Wygląd *User Controlera* służącego do przeglądania listy pacjentów.

- „*Entries*” służące do reprezentowania pozycji zwykle znajdujących się w elementach typu `<ListView>`, jak pokazano na poniższym listingu 5.6.

```

1 <ListView Grid.Row="1" x:Name="PatientsListView" ItemsSource="{Binding PatientsList}">
2   <ListView.ItemTemplate>
3     <DataTemplate DataType="ShortPatientForPatientsTab">
4       <Entries:PatientsEntry/>
5     </DataTemplate>
6   </ListView.ItemTemplate>
7 </ListView>

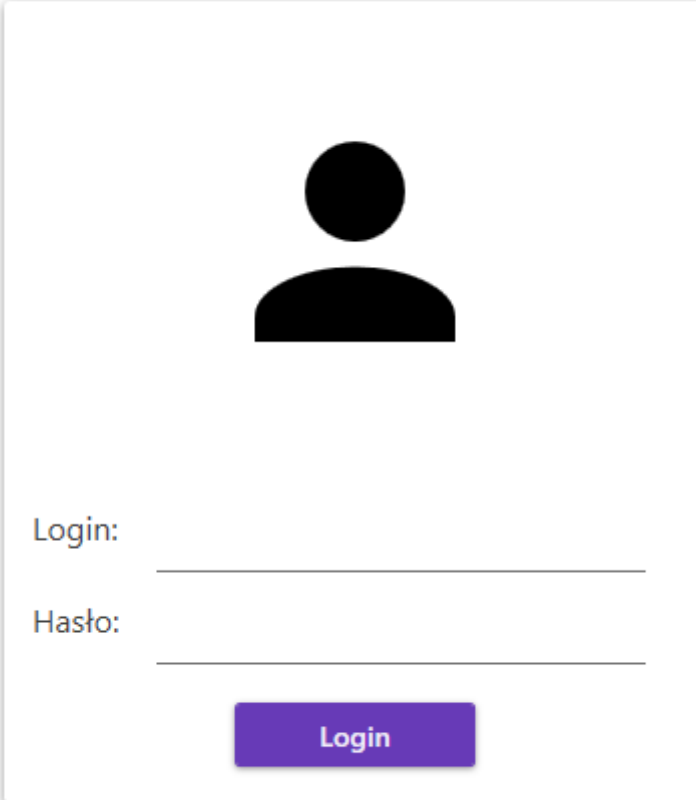
```

Listing 5.6: Przykładowe użycie *User Controler’a* w elemencie typu `<ListView>`.

Poza powyższymi podfolderami znajduje się tam jeszcze podfolder „*DataRepresentations*” zawierający klasy pomocnicze.

5.2. Logowanie użytkowników

Logowanie użytkowników składa się z 3 etapów. Najpierw w *User Controlerze* o nazwie „*LoginForm*” zaprezentowanym na rys 5.5.

The image shows a user login form. At the top center is a black silhouette of a person's head and shoulders. Below this, on the left side, are the labels "Login:" and "Hasło:" (Password:). To the right of each label is a horizontal text input field. At the bottom center of the form is a purple rectangular button with the word "Login" written in white text.

Rys. 5.5: Wygląd *User Controlera* służącego do logowania użytkownika do systemu.

Znajduje się tam wydarzenie „*LoginButton_Click*” przedstawione na listingu 5.7, które odczytuje wartości z *TextBox'a* o nazwie „*LoginTextBox*” i z *PasswordBox'a* o nazwie „*PasswordBox*” i następnie wykorzystując metodę „*LoginAs*” z klasy „*LoginManagement*” pobiera odpowiedniego użytkownika z bazy danych, zapisując go do do klasy statycznej służącej do przechowywania danych do których łatwy dostęp potrzebny jest z poziomu wielu *User Controlerów* aplikacji. Na zakończenie procesu logowania przekazywana jest informacja o typie użytkownika do głównego okna aplikacji, aby zostały pokazane odpowiednie zakładki w menu po lewej stronie ekranu.

```

private void LoginButton_Click(object sender, RoutedEventArgs e)
{
    foreach (Window window in Application.Current.Windows)
    {
        if (window.GetType() == typeof(MainWindow))
        {
            MainWindow parentWindow = (window as MainWindow);
            if (!String.IsNullOrEmpty(LoginTextBox.Text) &&
                !String.IsNullOrEmpty>PasswordBox.Password))
            {
                try
                {
                    DataHolderForMainWindow.User =
                        LoginManagement.LoginAs(LoginTextBox.Text,
                                                PasswordBox.Password);

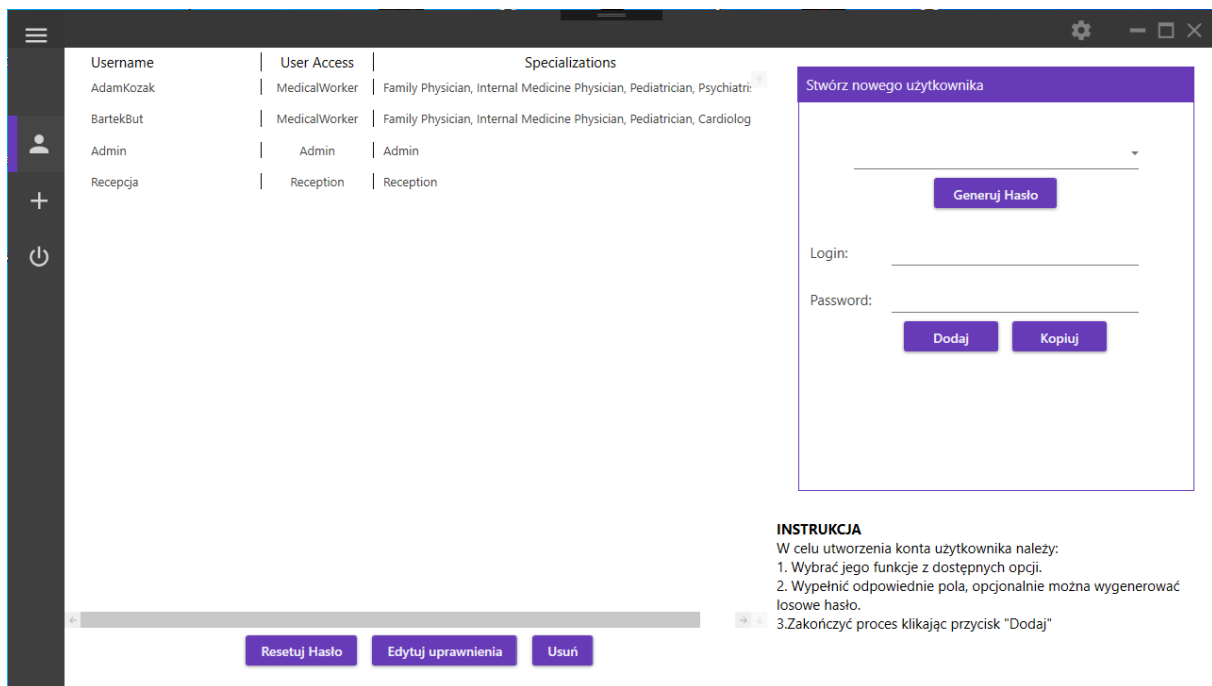
                    parentWindow.LogInType(
                        DataHolderForMainWindow.User.UserAccess);
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }
            }
            else
            {
                MessageBox.Show("Aby się zalogować należy podać login i hasło");
            }
        }
    }
}

```

Listing 5.7: Kod metody „LoginButton_Click” będącej wydarzeniem w „LoginForm”.

5.3. Rejestracja nowych użytkowników

Zamysł procesu rejestracji jest całkiem prosty. Nowi użytkownicy mogą zostać dodani wyłącznie przez administratora, który stworzy użytkownika z odpowiednimi uprawnieniami w systemie, a następnie przekaze login i wygenerowane hasło użytkownikowi. Stworzony w ten sposób użytkownik będzie miał możliwość zmiany hasła po zalogowaniu się do systemu. Całość tego procesu odbywa się w *User Controlerze* o nazwie „UsersManagementTab” przedstawionym na rys. 5.6.



Rys. 5.6: Wygląd okna z wczytaną zakładką „UserManagementTab”.

W celu stworzenia nowych kont użytkownika administrator powinien zacząć od wybrania jakiego rodzaju dostęp ma uzyskać nowy użytkownik. Następnie wybrać jego login i wygenerować hasło. Oba te pola może w prosty sposób skopiować, w celu przekazania ich zawartości nowemu pracownikowi. Jeśli tworzone jest konto nowego lekarza, pojawiają się dodatkowe dwa *TextBox*'y na jego imię i nazwisko, a w momencie dodania takiego użytkownika, dodawany jest jednocześnie nowy lekarz do bazy danych, który wstępnie posiada tylko Imię i Nazwisko, a specjalizacje i dostępności może przypisać sobie dopiero lekarz po zalogowaniu się do systemu.

5.4. Zmiana hasła

Po stworzeniu nowego konta użytkownika, pracownik może się zalogować na swoje konto i klikając ikonkę ustawień otwiera interfejs znajdujący się w pliku „*SettingsForm.xaml*” pozwalający mu na zmianę hasła. Zmiana hasła dotyczy obecnie zalogowanego użytkownika, tak więc w celu ustawienia nowego hasła użytkownik wpisuje tylko obecne hasło, a następnie dwukrotnie wpisuje nowe hasło. Jeśli proces przejdzie pomyślnie, użytkownik wróci do głównego menu programu. Kod odpowiadający za ten proces przedstawiłem na listingu 5.8.

```

private void ChangePasswordButton_Click(object sender, RoutedEventArgs e)
{
    try
    {
        if (NewPasswordBox.Password != ConfirmPasswordBox.Password)
        {
            throw new Exception("Hasła nie są identyczne!");
        }
        using (var context = new DataBaseContext())
        {
            if (context.Users
                .Single(x => x.Id == DataHolderForMainWindow.User.Id)
                .Password == CurrentPasswordBox.Password)
            {
                throw new Exception("Wpisano błędne hasło!
                Aby zmienić hasło należy wpisać poprawne hasło,
                a dopiero wtedy wpisać nowe.");
            }
            context.Users
                .Single(x => x.Id == DataHolderForMainWindow.User.Id)
                .Password = NewPasswordBox.Password;
            context.SaveChanges();

            foreach (Window window in Application.Current.Windows)
            {
                if (window.GetType() == typeof(MainWindow))
                {
                    MainWindow parentWindow = (window as MainWindow);
                    parentWindow.LogInType(
                        DataHolderForMainWindow.User.UserAccess);
                }
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

```

Listing 5.8: Kod metody „*ChangePasswordButton_Click*” będącej wydarzeniem w „*SettingsForm*”.

5.5. Dodawanie nowych pacjentów

Dodawaniem nowych pacjentów zajmują się użytkownicy z uprawnieniami recepcjonistów. Aby dodać nowego pacjenta do bazy danych recepcjonista powinien zalogować się, wybrać z menu

zakładkę „Pacjenci”, która wyświetli *User Controller* o nazwie „*PatientsTab*”. Po wybraniu w widoku „*Nowy Pacjent*” powinien ukazać się interfejs pozwalający na dodawanie nowych pacjentów, który przedstawiłem z wypełnionymi przykładowymi danymi na rys. 5.7.

The screenshot shows a desktop application window with a dark sidebar and a top bar. The top bar has four tabs: "Wszyscy pacjenci", "Nowy Pacjent" (which is highlighted with a purple border), "Pacjent", and "Szczepienia". The sidebar contains several icons, including a plus sign, a pencil, a document, and a power button. The main area of the window displays a form for adding a new patient. At the top of the form is a placeholder for a patient's photo, showing a stylized black and white face. Below the photo are several input fields arranged in two columns. The left column contains: "Imię:" with the value "Katarzyna", "Nazwisko:" with "Nowak", "Email:" with "katarzyna.nowak@testmail.com", and "Pesel:" with "109111397861". The right column contains: "Data urodzenia:" with "13.11.2009" and a calendar icon, "Płeć:" with a dropdown menu showing "Kobieta", and "Kontakt email:" with a dropdown menu showing "Pełny". At the bottom center of the form is a purple button labeled "Dodaj".

Rys. 5.7: Wygląd okna z wczytaną zakładką „*PatientsTab*” i widokiem „*Nowy Pacjent*”.

Działanie tego komponentu jest bardzo proste, po wypełnieniu przez recepcjonistę formularza znajdującego się na tym widoku, po naciśnięciu przycisku „*Dodaj*” uruchamia się metoda (*Event*) przedstawiona na listingu 5.9. Na początku tworzy ona pustą historię leczenia klasy *DbTreatmentHistory*, a następnie dodaje ją do bazy danych. Jest to potrzebne do tego do utworzenia pacjenta w następnej kolejności, ponieważ każdy pacjent musi posiadać własną historię leczenia w momencie jego tworzenia jako nowego obiektu klasy *DbPatient*. Pola tworzonego pacjenta są odpowiednio ustawiane według tego, co zostało wybrane w formularzu, a następnie dodawane do kontekstu *Entity Framework* bazy danych i zapisywane.

```

private void AddPatientButton_Click(object sender, RoutedEventArgs e)
{
    using (var context = new DataBaseContext())
    {
        var treatmentHistory = new DbTreatmentHistory()
        {
            PastVaccinations = new List<DbVaccination>(),
            RequiredVaccinations = new List<DbVaccination>(),
            Treatments = new List<DbTreatment>()
        };
        context.TreatmentHistories.Add(treatmentHistory);
        context.SaveChanges();

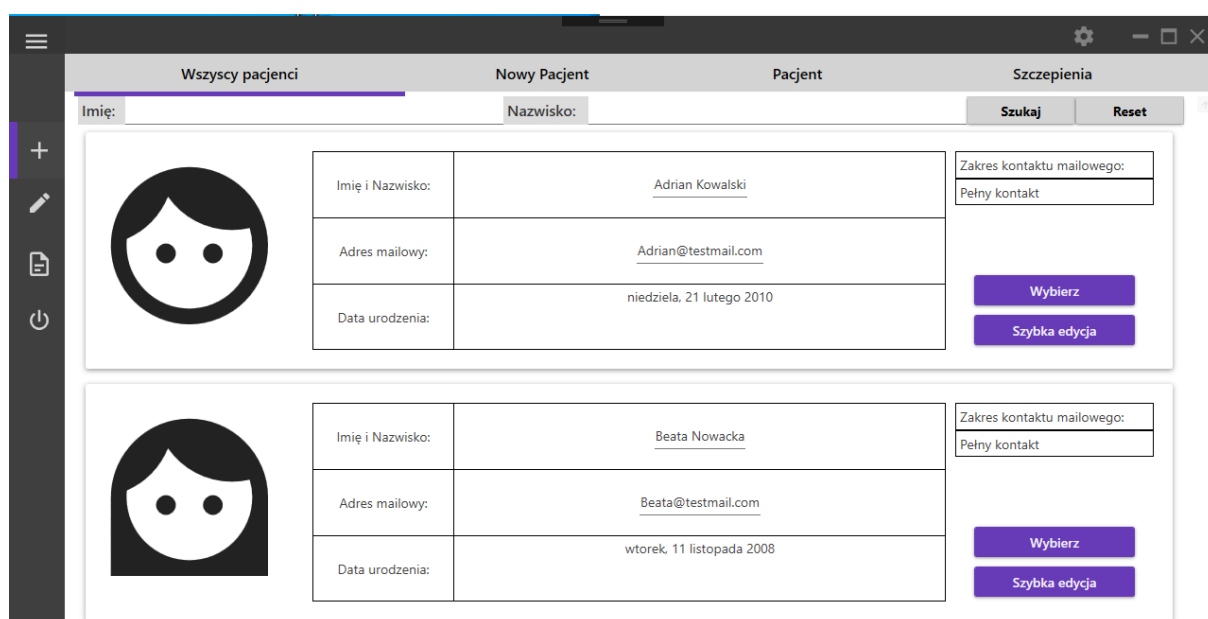
        var newPatient = new DbPatient()
        {
            FirstName = NameTextBox.Text,
            LastName = SurnameTextBox.Text,
            EmailAdress = EmailTextBox.Text,
            DateOfBirth = BirthdayDatePicker.SelectedDate.Value,
            PatientsPronounce =
                GenderComboBox.SelectedIndex == 0 ? Pronouns.MR : Pronouns.MRS,
            EmailContact = GetEmailContact(AdressComboBox.SelectedIndex),
            TreatmentHistory = treatmentHistory,
            Visits = new List<DbVisit>()
        };
        context.Patients.Add(newPatient);
        context.SaveChanges();
    }
}

```

Listing 5.9: Kod metody „AddPatientButton_Click” będącej wydarzeniem w „PatientsTab”.

5.6. Rejestracja wizyt pacjentów

Rejestracja wizyt składa się z dwóch części, najpierw recepcjonista korzystając z głównego widoku zakładki „Pacjenci” zaprezentowanego na rys. 5.8, musi wybrać pacjenta którego będzie chciał zarejestrować wciskając przycisk „Wybierz”.



Rys. 5.8: Wygląd okna z wczytaną zakładką „PatientsTab” na jego głównym widoku.

Wybranie pacjenta następuje w bardzo prosty sposób. W kodzie odpowiada za to tylko jedna linijka z listingu 5.10, lecz ze względu na to, że przycisk ten odblokowuje możliwość dostępu do widoków „Pacjent” i „Szczepienia” wymagane jest wykonanie przez to wydarzenie dodatkowych czynności, które nie są ważne w omawianym w tej sekcji funkcjonalności.

```

DataHolderForMainWindow.PatientId = Convert.ToInt32(IdHolderHack.Text);

```

Listing 5.10: Fragment kodu odpowiedzialnego za wybór pacjenta.

Gdy pacjent jest już wybrany, w zakładce o nazwie *Harmonogram*, której implementacja znajduje się w pliku „ScheduleTab.xaml” widzimy dostępne terminy wizyt. Określane są tylko dniami tygodnia, ponieważ mają za zadanie zarejestrować wizytę o wybranej godzinie w następnym dostępnym terminie. Po wybraniu godziny, dostajemy informację zwrotną z dokładną datą i godziną na którą pacjent został zarejestrowany. Poniżej na listingu 5.11 przedstawiłem kod będący odpowiedzialny za dodanie wizyty do bazy danych i przypisanie jej do odpowiedniego pacjenta, a także lekarza.

```

DbVisit visit = new DbVisit()
{
    MedicalWorker = context.MedicalWorkers
        .Single(x => x.Id ==
            Convert.ToInt32(MedicalWorkerIdHolderHack.Text)),
    Specialization = context.Specializations
        .Single(x => x.Id ==
            Convert.ToInt32(SpecializationIdHolderHack.Text)),
    OptionalDescription = "",
    Patient = context.Patients
        .Single(x => x.Id == DataHolderForMainWindow.PatientId),
    TimeStart = TimeForNewVisit
};
if (context.Visits.Any(x => x.MedicalWorker == visit.MedicalWorker
    && x.TimeStart == visit.TimeStart))
{
    throw new Exception("Nie mogą odbywać się 2 wizyty w tym samym
        czasie. Proszę wybrać inną godzinę.");
}
context.Visits.AddOrUpdate(x => x.Id, visit);
context.SaveChanges();

context.MedicalWorkers
    .Single(x => x.Id == Convert.ToInt32(MedicalWorkerIdHolderHack.Text))
    .Visits
    .Add(visit);
context.Patients
    .Single(x => x.Id == DataHolderForMainWindow.PatientId)
    .Visits
    .Add(visit);
context.SaveChanges();

MessageBox.Show("Wizyta odbędzie się: " + dateTime.ToString());

```

Listing 5.11: Fragment kodu metody odpowiadającej za rejestrację wizyt pacjentów.

5.7. Obsługa pacjentów i wystawianie druków

W celu intuicyjnej pracy z aplikacją, lekarz po zalogowaniu ma dostęp do listy wszystkich nadchodzących wizyt w zakładce o nazwie „Lista wizyt” znajdującej się w pliku „VisitsListTab.xaml”. Dane dotyczące wizyt są przedstawione w elemencie typu „ListView”, znajdującym się wewnątrz elementu „ScrollView” co zaprezentowałem na listingu 5.12. Przedstawiają one dane w sposób minimalistyczny, co znaczy że udostępniają lekarzowi tylko te

informacje, które są niezbędne do określenia celu wizyty pacjenta, a także identyfikacji go za pomocą imienia i nazwiska.

```

1 <ScrollView x:Name="VisitsScrollView" Grid.Column="0" Margin="0 25 0 60"
  ↳ PreviewMouseWheel="VisitsScrollView_PreviewMouseWheel">
2   <ListView x:Name="VisitsListView" ItemsSource="{Binding VisitsList}">
3     <ListView.ItemTemplate>
4       <DataTemplate DataType="VisitForVisitsEditorTab">
5         <Grid Margin="10 0">
6           <Grid.ColumnDefinitions>
7             <ColumnDefinition Width="140"/>
8             <ColumnDefinition Width="100"/>
9             <ColumnDefinition Width="120"/>
10            <ColumnDefinition Width="*/>
11          </Grid.ColumnDefinitions>
12          <Border Grid.Column="1" BorderBrush="Black" BorderThickness="1 0 1
13            ↳ 0" Background="{x:Null}" />
14          <Border Grid.Column="3" BorderBrush="Black" BorderThickness="1 0 0
15            ↳ 0" Background="{x:Null}" />
16          <TextBlock Text="{Binding Patient}" Grid.Column="0" Margin="10 0"
17            ↳ VerticalAlignment="Center" TextWrapping="Wrap"/>
18          <TextBlock Text="{Binding VisitWhen}" Grid.Column="1" Margin="10
19            ↳ 0" VerticalAlignment="Center" HorizontalAlignment="Center"
20            ↳ TextWrapping="Wrap"/>
21          <TextBlock Text="{Binding Specialization}" Grid.Column="2"
22            ↳ Margin="10 0" VerticalAlignment="Center"
23            ↳ HorizontalAlignment="Center" TextWrapping="Wrap"/>
24          <TextBlock Text="{Binding Description}" Grid.Column="3" Margin="10
25            ↳ 0" VerticalAlignment="Center" HorizontalAlignment="Center"
26            ↳ TextWrapping="Wrap"/>
27        </Grid>
28      </DataTemplate>
29    </ListView.ItemTemplate>
30  </ListView>
31</ScrollView>

```

Listing 5.12: Fragment kodu odpowiedzialnego za wybór pacjenta.

Po dokonaniu wyboru odpowiedniej wizyty, lekarz może skorzystać z zakładki o nazwie „Obecna wizyta”. Ta zakładka przedstawia lekarzowi podstawowe dane o pacjencie, oraz listę ostatnich przebytych chorób. Po dokonaniu przez lekarza badania pacjenta, lekarz może zamieścić opis objawów, oraz nazwę zdiagnozowanej choroby, lub dolegliwości na specjalnym formularzu, a także dodawać leki do recepty. Gdy to wszystko zostanie wykonane, lekarz ma możliwość wygenerowania odpowiedniego skierowania, a także recepty. Zajmuję się tym specjalna klasa statyczna o nazwie „PDFCreator”. Wygenerowane pliki powinny znaleźć się w specjalnym folderze o nazwie „Moje Dokumenty”, lub jego odpowiedniku dla wybranej w systemie operacyjnym kultury i języka, na przykład przy ustawionej w systemie kulturze *Us-en* folder ten będzie miał nazwę „My Documents”. Po wszystkich czynnościach, klikając

odpowiedni przycisk w aplikacji pracownik medyczny może zakończyć wizytę, a wszystkie dane zostaną odpowiednio przetworzone, oraz zmiany zapisane w bazie danych za pomocą metody przedstawionej na listingu 5.13.

```
private void FinishVisitButton_Click(object sender, RoutedEventArgs e)
{
    using (var context = new DataBaseContext())
    {
        PrescriptionList.ForEach(
            x => context.PrescribedMedications.AddOrUpdate(i => i.Id, x));
        context.SaveChanges();

        var treatment = new DbTreatment()
        {
            IllnessName = DiagnosedSicknessTextBox.Text,
            SymptomsDescription = SymptomsDescriptionTextBox.Text,
            Prescription = PrescriptionList,
            Visit = context.Visits
                .Single(x => x.Id == DataHolderForMainWindow.ChosenVisitId)
        };
        context.Treatments.AddOrUpdate(x => x.Id, treatment);
        context.SaveChanges();

        context.TreatmentHistories
            .Single(x => x.Id == CurrentVisit.TreatmentHistoryId)
            .Treatments.Add(treatment);
        context.SaveChanges();
    }
}
```

Listing 5.13: Kod metody „*FinishVisitButton_Click*” zakończonej wizytę pacjenta.

Rozdział 6

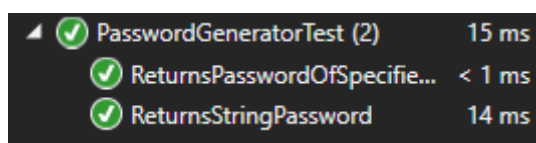
Testy aplikacji

W celu sprawdzenia i zagwarantowania poprawnego działania aplikacji przeprowadzono 2 rodzaje testów aplikacji:

- testy jednostkowe,
- testy manualne.

6.1. Testy jednostkowe

Testy jednostkowe zostały wykonane za pomocą „Unit Test Project (.Net Framework)”. Miały one na celu kontrolę spójności, oraz monitorowanie działania pomniejszych klas i metod. Przykładowymi testami tego typu są testy klasy testowej *PasswordGeneratorTest*, które przedstawiłem na listingu 6.1. Te testy miały na celu upewnić się, iż *PasswordGenerator* zwracał ciągi znaków spełniające kryteria dotyczące wygenerowanych haseł. Jak zostało przedstawione na rys. 6.1 testy te były wykonywane pomyślnie.



Rys. 6.1: Rezultat testów jednostkowych dla *PasswordGeneratorTest*.

```

[TestClass]
public class PasswordGeneratorTest
{
    [TestMethod]
    public void ReturnsStringPassword()
    {
        // Act
        var testPassword = PasswordGenerator.GetRandomPassword();

        // Assert
        Assert.IsInstanceOfType(testPassword, typeof(string), "Zwrócone hasło nie jest stringiem");
    }

    [TestMethod]
    public void ReturnsPasswordOfSpecifiedLength()
    {
        // Arrange
        int requiredLength = 20;

        // Act
        var testPassword = PasswordGenerator.GetRandomPassword();

        // Assert
        Assert.AreEqual(testPassword.Length, requiredLength);
    }
    // (...)
}

```

Listing 6.1: Klasa testowa „*PasswordGeneratorTest*” z dwoma przykładowymi testami.

6.2. Testy manualne

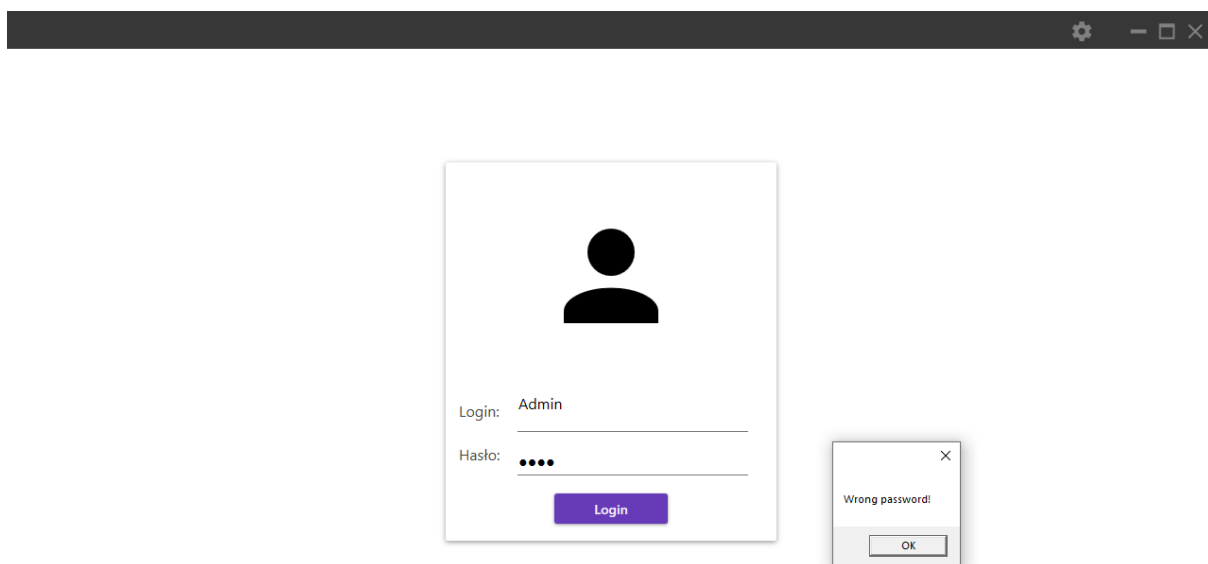
Przeprowadzone testy manualne miały na celu między innymi:

- odnajdywanie błędów w powstałych w przypadku niepoprawnego korzystania z oprogramowania,
- zabezpieczanie przed błędami w powstałych w przypadku niepoprawnego korzystania z oprogramowania poprzez poprawę kodu aplikacji,
- wykrycie nieintuicyjnych dla użytkownika funkcjonalności aplikacji, oraz ich poprawa,
- wykrycie potencjalnych błędów interfejsu.

Poniżej wymienię przykładowe testy manualne jakim zostało poddane oprogramowanie.

6.2.1. Test próby wpisania błędnego hasła

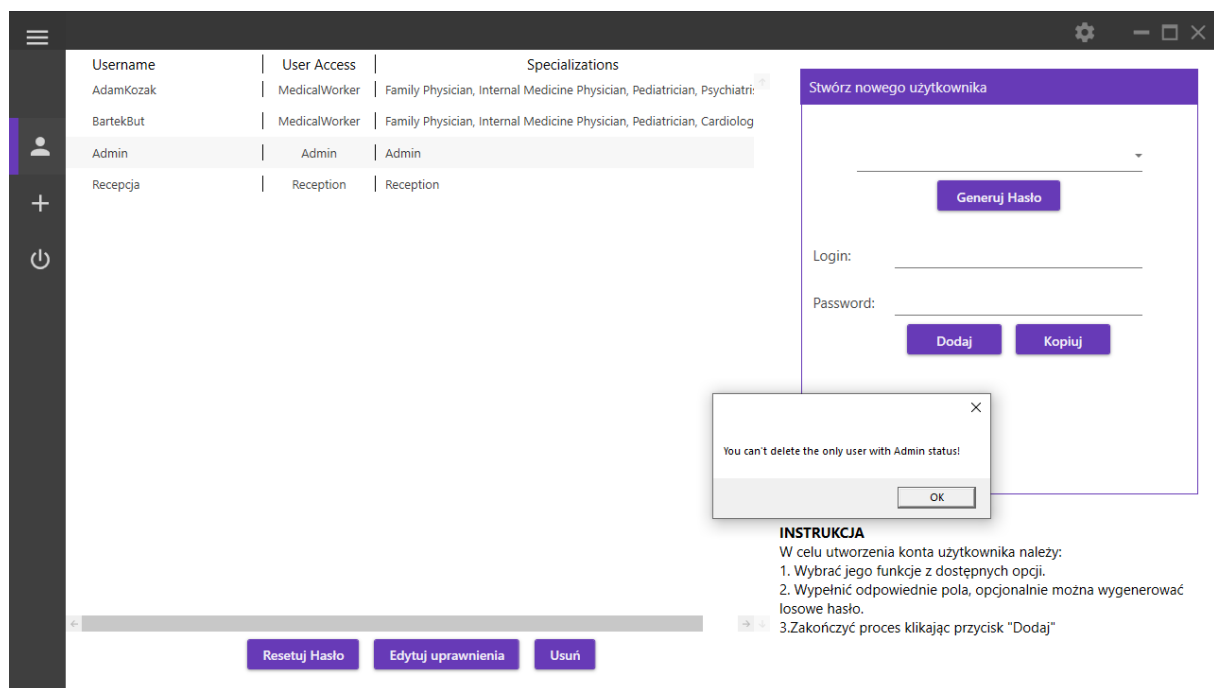
Nie powinno być możliwe zalogowanie się na konto użytkownika pomimo wpisania błędnego hasła. W celu sprawdzenia weryfikacji hasła, no i co wtedy wykona program przeprowadzono odpowiedni test manualny. Przy próbie wpisania błędnego hasła, został wyświetlony komunikat informujący o wpisaniu niepoprawnego hasła, a użytkownik musiał ponownie wpisywać hasło. Rezultat tego testu przedstawiono na rys. 6.2.



Rys. 6.2: Rezultat testu manualnego próby wpisania błędnego hasła.

6.2.2. Test próby usunięcia jedynego administratora

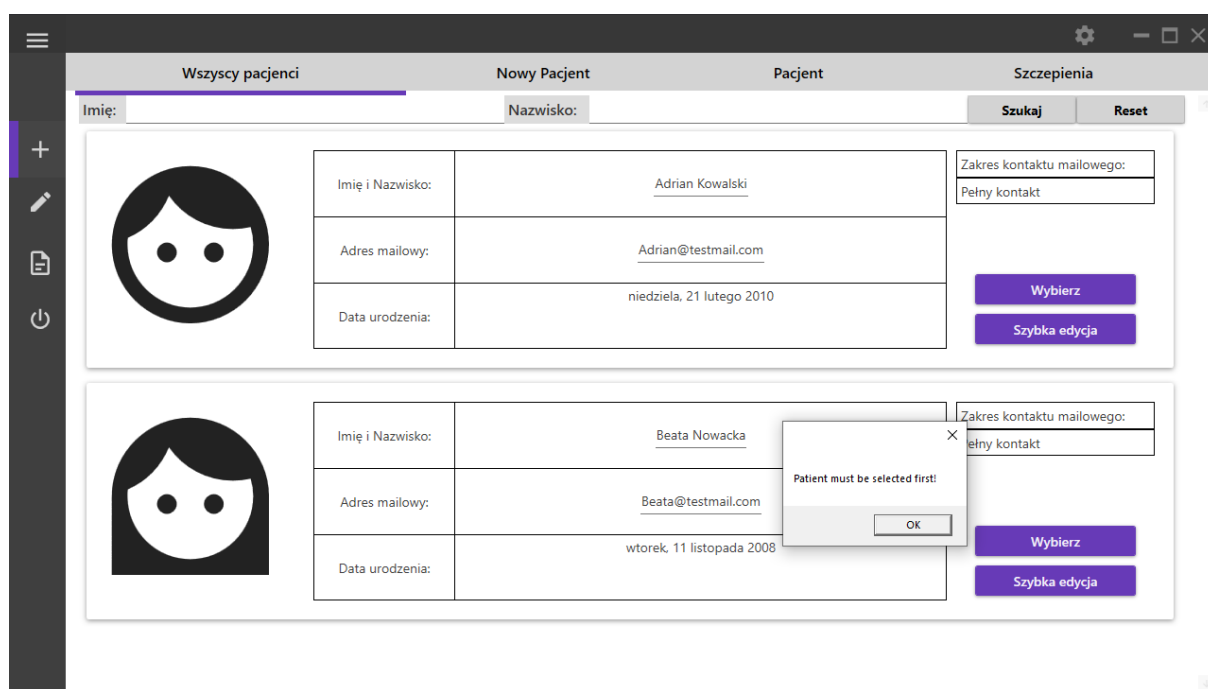
W systemie musi istnieć przynajmniej jeden administrator, dlatego użytkownik nie powinien być w stanie usunąć konta jedynego administratora. W celu sprawdzenia, czy tak rzeczywiście się dzieje, przeprowadzony został test, gdzie administrator zalogował się na swoje konto, a następnie w zakładce „Zarządzaj kontami” wybrał z listy swoje konto i nacisnął przycisk „Usuń”. W rezultacie otrzymał powiadomienie informujące go że nie można usunąć ostatniego konta z uprawnieniami administratora. Test został zakończony z powodzeniem, a jego rezultat został przedstawiony na rys. 6.3.



Rys. 6.3: Rezultat testu manualnego dla próby usunięcia jedynego administratora.

6.2.3. Test próby przejścia do widoku szczegółowego pacjenta, bez wybrania wcześniej pacjenta z listy

Aby został wyświetlony szczegółowy widok pacjenta, należy najpierw wybrać pacjenta z listy. Dopóki nie zostanie wybrany, elementy znajdujące się w widoku o nazwie „Pacjent” nie mają odpowiednich danych, aby zostać załadowane. Z tego powodu, wykonany został test sprawdzający co się stanie, jeśli użytkownik spróbuje zmienić widok bez uprzedniego wyboru pacjenta z listy, którego detale pragnie ujrzeć. Po zalogowaniu się na konto recepcjonisty wybrano zakładkę o nazwie „Pacjenci”. Tam bez klikania na jakąkolwiek kartę z listy istniejących pacjentów kliknięto na przycisk zmieniający widok na widok „Pacjent”. W odpowiedzi użytkownik otrzymał okienko z informacją, że w celu przejścia do tamtego widoku, najpierw trzeba wybrać odpowiedniego pacjenta z listy. Rezultat testu przedstawiony został na rys. 6.4.



Rys. 6.4: Rezultat testu manualnego dla próby przejścia do widoku szczegółowego pacjenta, bez wybrania wcześniej pacjenta z listy.

Rozdział 7

Podsumowanie i wnioski

Celem pracy dyplomowej było stworzenie aplikacji desktopowej na system Windows, której zastosowaniem i główną funkcjonalnością miało być przetwarzanie danych takich jak rejestracja wizyt, przechowywanie dziecięcej dokumentacji medycznej, a także wystawianie odpowiednich druków. Całość oprogramowania została napisana w języków C#, oraz XAML przy wykorzystaniu zintegrowanego środowiska programistycznego (IDE) Microsoft Visual Studio. Zrealizowana aplikacja posiada funkcjonalności takie jak:

- tworzenie kont nowych użytkowników
- logowanie pracowników do systemu
- edycja kont użytkowników
- tworzenie kont nowych pacjentów
- przechowywanie dokumentacji medycznej pacjentów
- rejestracja wizyt
- wystawianie druków

Tym samym oprogramowanie spełnia założone uprzednio wymagania. Wykonanie projektu okazało się procesem znacznie trudniejszym i bardziej czasochłonnym niż początkowo zakładałem. Z tego powodu widoczna jest w interfejsie aplikacji pewna prostota podyktowana trudnościami związanymi z aranżacją układu elementów interfejsu w sposób sensowny, a jednocześnie estetyczny i co najważniejsze spełniający swoją podstawową rolę, jaką jest intuicyjność dla użytkownika. Ostateczny efekt jest bardzo zadowalający pomimo swojej prostoty.

7.1. Możliwości rozwoju

Projekt z oczywistych powodów nie wyczerpuje kwestii wsparcia pracy przychodni pediatrycznej, istnieje więc wiele ścieżek rozwoju. Między innymi możliwe jest dodanie serwisu obsługującego wysyłanie biuletynów mailowych z aktualną ofertą, a także informacji przypominającej pacjentom o nadchodzących wizytach. Inną ścieżką rozwoju mogłoby być rozwinięcie interfejsu i bazy danych, zastąpienie ikonek dostępnych z pakietu *MaterialDesign* ilustracjami takimi jak zdjęcia konkretnych leków, graficzne przedstawienie harmonogramu dostępności lekarzy, a także dodanie innych efektów wizualnych.

Bibliografia

- [1] *Konwencje kodowania C#*. <https://docs.microsoft.com/pl-pl/dotnet/csharp/programming-guide/inside-a-program/coding-conventions>. (Term. wiz. 28.01.2020).
- [2] I. Reic. *A simplified Jira clone built with React/Babel (Client), and Node/TypeScript (API). Auto formatted with Prettier, tested with Cypress*. https://github.com/oldboyxx/jira_clone. (Term. wiz. 28.01.2020).
- [3] *Strona aplikacji internetowej e-scanmed*. <https://www.e-scanmed.pl/ron-www/placowkaMedyczna/znajdz>. (Term. wiz. 06.02.2020).
- [4] *Strona internetowa Developer Notes*. <https://mndevnotes.wordpress.com/2012/06/18/wpf-dynamiczne-tworzenie-i-wczytywanie-kodu-xaml/>. (Term. wiz. 27.01.2020).
- [5] *Strona internetowa z dokumentacją języka C#*. <https://docs.microsoft.com/pl-pl/dotnet/csharp/>. (Term. wiz. 26.01.2020).
- [6] *Strona oferty aplikacji Medfile*. <https://www.medfile.pl/elektroniczna-dokumentacja-medyczna>. (Term. wiz. 06.02.2020).

Dodatek A

Instrukcja obsługi

A.1. Przed pierwszym uruchomieniem

Zanim po raz pierwszy uruchomi się aplikację, należy zadbać o to, aby na komputerze zainstalowany był pakiet Microsoft SQL Server 2017, lub nowszy. Jest to konieczne, ponieważ aplikacja została zaprojektowana z myślą o korzystaniu z bazy danych opartej o właśnie ten pakiet.

A.2. Pierwsze kroki

Aby uzyskać dostęp do aplikacji należy się zalogować. Na samym początku utworzone jest domyślne konto administratora, o loginie i hasle „Admin”. Po pierwszym uruchomieniu należy zmienić hasło, klikając przycisk ustawień, znajdujący się na samej górze po lewej stronie od przycisku minimalizacji okna. Alternatywnie można utworzyć też profil nowego administratora według sposobu opisanego w sekcji A.3, przelogować się na takie nowo utworzone konto, a następnie usunięcie domyślnego konta administratora w sposób opisany w sekcji A.4.

A.3. Tworzenie kont nowych użytkowników

Aby utworzyć konto nowego użytkownika, konieczne jest zalogowanie się na konto administratora. Administrator następnie korzystając z zakładki „Zarządzaj kontami” korzystając z formularzy po lewej stronie wybiera rodzaj nowego konta użytkownika. W wypadku tworzenia konta recepcjonisty, lub administratora wystarczy podać login i wygenerować hasło, podczas gdy dla konta pracownika medycznego potrzebne jest podanie również imienia i nazwiska lekarza.

Gdy formularz zostanie wypełniony, należy kliknąć przycisk „Kopiuj” i „Dodaj”, a następnie przekazać dane logowania nowemu użytkownikowi.

A.4. Usuwanie kont użytkowników

Usuwanie kont użytkowników może ponownie wykonywać tylko administrator z poziomu zakładki „Zarządzaj kontami”. Administrator wybiera z listy odpowiedniego użytkownika, a następnie naciska przycisk „Usuń”. Uwaga! Jeśli wybrany użytkownik jest jedynym administratorem, program powiadomi o błędzie i odmówi wykonania operacji.

A.5. Edycja uprawnień użytkowników

Należy postępować podobnie jak w wypadku usuwania kont, z tą różnicą, że trzeba wybrać nowy rodzaj uprawnień tak jak w sekcji A.3 i kliknąć przycisk „Edytuj uprawnienia”.

A.6. Tworzenie nowych kont pacjentów

W celu stworzenia nowego pacjenta, należy zalogować się na konto recepcjonisty. W menu wybrać zakładkę „Pacjenci”, a następnie wybrać u góry zakładki widok „Nowy Pacjent”. Tam po wypełnieniu formularza, należy kliknąć przycisk „Dodaj”.

A.7. Edycja danych pacjenta

Edycji danych pacjenta można dokonać na dwa sposoby. W obu sposobach należy zacząć od zalogowania się na konto recepcjonisty, a następnie wybrania w menu zakładkę „Pacjenci”.

- Pierwszy ze sposobów pozwala na wykonanie szybkiej niewielkiej edycji imienia, lub nazwiska. Jest ona użyteczna zaraz po stworzeniu pacjenta, w wypadku gdy recepcjonista popełni niewielki błąd przy wypełnianiu formularza. Gdy chcemy z niej skorzystać klikamy przycisk „Szybka edycja”, wtedy dostaniemy możliwość zmian pól *Imię* i *Nazwisko*, oraz pokaże się przycisk „Zapisz”, po którego przyciśnięciu zakończymy edycje.
- Drugi sposób pozwala na edycję większej ilości pól. Wymaga on kliknięcia przycisku „Wybierz”, a następnie przejścia do widoku „Pacjent”. Tam w podobny sposób co powyżej można dokonać edycji.

A.8. Dostęp do szczepień

Aby sprawdzić informacje dotyczące szczepień należy zalogować się na konto recepcjonisty, tam w zakładce „*Pacjenci*” wybrać pacjenta którego szczepienia nas interesują i przejść do widoku o nazwie „*Szczepienia*”.

A.9. Rejestrowanie nowych wizyt

W celu rejestracji nowej wizyty, należy zalogować się na konto recepcjonisty, w zakładce „*Pacjenci*” wybrać pacjenta, a następnie przejść do zakładki „*Harmonogram*”. W tej zakładce należy wybrać interesującą pacjenta usługę, godzinę na którą chcemy pacjenta zarejestrować, a na koniec kliknąć przycisk „*Rejestruj*”. W odpowiedzi pracownik otrzyma komunikat z dokładną datą wizyty przewidzianą dla pacjenta.

A.10. Usuwanie i edycja opisu wizyt

Aby edytować, lub usuwać wizyty należy zalogować się na konto recepcjonisty. Korzystając z zakładki „*Edytuj wizyty*” należy wybrać interesującą użytkownika wizytę, a następnie w celu usunięcia wizyty nacisnąć przycisk „*Usuń*”. W wypadku edycji opisu, należy najpierw wpisać opis, a następnie kliknąć przycisk „*Edytuj opis wizyty*”.

A.11. Obsługa wizyty pacjenta przez lekarza

W celu obsługi wizyt należy zalogować się na konto lekarza. Z menu należy wybrać zakładkę „*Lista wizyt*”, a tam po zidentyfikowaniu interesującej nas wizyty należy wybrać odpowiednią pozycję z listy i nacisnąć przycisk „*Wybierz*”. W następnej kolejności korzystając z zakładki „*Obecna wizyta*” należy po zapoznaniu się z dolegliwościami pacjenta odpowiednio wypełnić formularz. Na zakończenie wizyty można wygenerować receptę, oraz skierowanie. Po kliknięciu odpowiedniego przycisku, odpowiedni plik w formacie PDF zostanie umieszczony w folderze „*Moje Dokumenty*”. Dokument ten należy wydrukować i przekazać pacjentowi. Po wszystkim w celu zakończenia wizyty i zapisania jej w bazie danych należy wcisnąć przycisk „*Zakończ wizytę*”.