

Obsługa gabinetu stomatologicznego

Alan Abdi

Karol Dziadkowiec

Kamil Dziewa

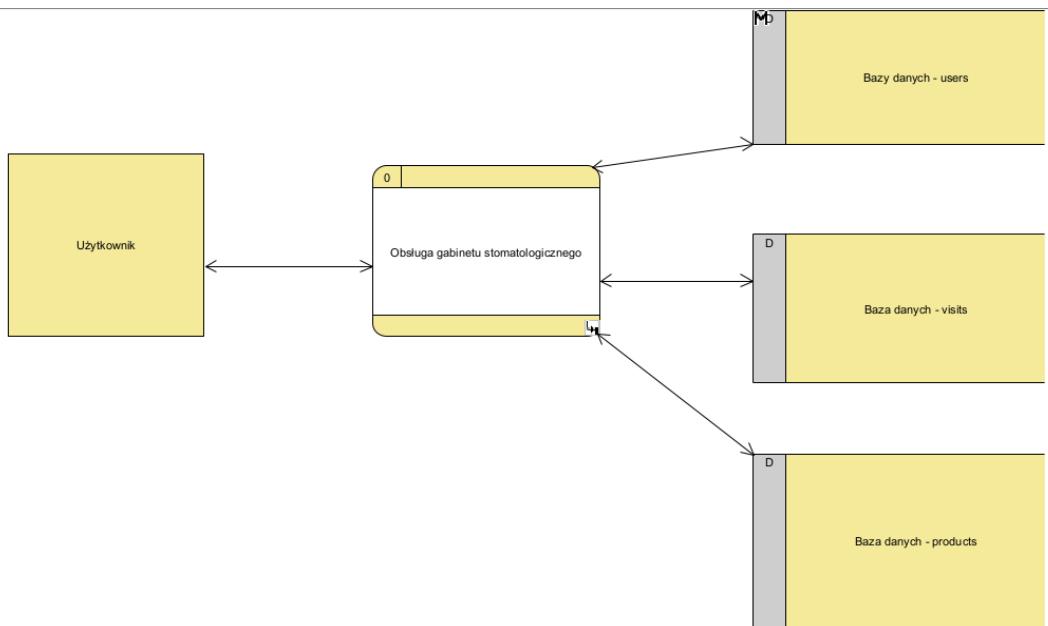
Spis treści

1. Specyfikacje projektu.	3
2. Opis wybranej metodyki wytwarzania.	4
3. Słownik pojęć i terminów.	5
4. Diagram przypadków użycia.	7
5. Wymagania funkcjonalne i niefunkcjonalne.	8
6. Diagramy wymagań.	9
7. Wykaz zastosowanych framework-ów, bibliotek, środowiska, technologii oraz informacje dotyczące zastosowanych rozwiązań informatycznych.	10
8. Diagramy aktywności.	13
9. Diagram klas.	17
10. Diagramy sekwencji	20
11. Diagram komponentów.	24
12. Diagram wdrożenia.	25
13. Przykłady współpracy zespołu z wykorzystaniem narzędzia Trello.	26
14. Testy jednostkowe.	31
15. Przykłady refaktoryzacji kodu.	32
16. Przykład wzorca projektowego, który mógłby zostać wykorzystany podczas tworzenia projektu.	35
17. Instrukcja użytkowania programu.	37
18. Podsumowanie.	40

1. Specyfikacje projektu.

- **Koncepcja systemu:** Celem zbudowanego projektu jest zapewnienie użytkownikowi możliwości zdalnej rejestracji do gabinetu stomatologicznego. Dzięki tej aplikacji wychodzenie z domu, bądź bezpośrednie dzwonienie do gabinetu w celu rejestracji wizyty nie jest potrzebne. Głównym założeniem stworzenia programu jest zagwarantowanie wygody pacjentom, chcącym się umówić na wizytę oraz zapewnienie stomatologowi bezpiecznego przechowywania danych pacjentów.
- **Opis systemu:** System został stworzony jako aplikacja desktopowa(okienkowa). Możliwe jest uruchomienie programu za pomocą pliku o rozszerzeniu .exe. System został stworzony z pomocą framework'a .NET oraz środowiska programowania Visual Studio 2022. Kod systemu wygenerowano w języku C++.
- **Podział systemu:** System opiera się na dwóch głównych częściach: aplikacji okienkowej oraz bazie danych. Aplikacja desktopowa jest szkieletem całego projektu, odpowiada za funkcjonowanie całego systemu. Natomiast praca na danych w tej aplikacji wymaga użycia zewnętrznej bazy danych.
- **Przepływ danych w systemie:** Po uruchomieniu aplikacji dane zalogowanych użytkowników, ich wizyt oraz produktów magazynu pobierane są z zewnętrznego serwera bazodanowego SQL. Pobrane dane działają aż do końca funkcjonowania aplikacji i można na nich pracować podczas uruchomionego programu. Po dodaniu nowego elementu do bazy danych(np. nowe konto, produkt, wizyta) jest ono natychmiast zapisywane w bazie danych, więc nigdy nie zostanie utracone.

Ogólny diagram przepływu danych dla całego systemu:



Reszta diagramu znajduje się w załączonym do dokumentacji pliku z Visual Paradigm.

https://drive.google.com/file/d/1--j_CRzX0-SFb4SLiQ16zdrzsypsyHOG/view?usp=sharing

2. Opis wybranej metodyki wytwarzania.

Do wytwarzania oprogramowania wybrano model kaskadowy(*ang. waterfall model*). Polega on na wykonywaniu podstawowych czynności jako odrębnych faz projektowych, kolejno po sobie. Każda czynność to schodek (inaczej kaskady). Każda z nich podlega osobnym testom i sprawdzeniu użyteczności. Wprowadza się tyle iteracji, ile trzeba, aby pomyślnie uzyskać założony cel. Wytwarzanie polega na następujących po sobie etapach:



Przyczyny wyboru danej metodyki:

- ❖ chęć uporządkowania procesu tworzenia oprogramowania już od samego początku,
- ❖ precyjnie określony zakres prac,
- ❖ brak nieporozumień(szczegóły ustalane na starcie tworzenia programu),
- ❖ mały rozmiar grupy - ułatwia organizację pracy,
- ❖ orientacja na czas(dokładna ocena czasu realizacji projektu).

3. Słownik pojęć i terminów.

C++ - język programowania ogólnego przeznaczenia wspierający paradymat imperatywny i obiektowy oraz programowanie generyczne.

.NET Framework(.NET) – platforma programistyczna opracowana przez Microsoft. Obejmuje środowisko uruchomieniowe oraz biblioteki klas dostarczające standardowej funkcjonalności dla aplikacji.

Aplikacje desktopowe(okienkowe) - standardowe aplikacje pracujące w systemie Windows, których interfejs opiera się na okienkach, obsługuje się je za pomocą myszy i klawiatury.

Aplikacja DentApp jest to program, który pozwala użytkownikowi zdalnie zarządzać wizytą w gabinecie stomatologicznym. Program jest aplikacją okienkową, która jest prosta i bardzo czytelna w obsłudze. Można korzystać z programu z 3 perspektyw:

→ **Obserwator**

Po pierwszym otwarciu aplikacji użytkownik staje się obserwatorem. Nie posiada wielu funkcji, gdyż nie jest zalogowany do systemu. Obserwator ma możliwość korzystania z panelu logowania oraz rejestracji.

→ **Pacjent**

Tuż po zarejestrowaniu konta oraz zalogowaniu się do systemu użytkownik staje się pacjentem, który posiada funkcje takie jak: zarejestrowanie oraz anulowanie wizyty, przeglądanie swoich przeszłych oraz nadchodzących wizyt, przeglądanie swoich danych wraz z możliwością edycji. Może także wylogować się z systemu i powrócić do stanu obserwatora.

→ **Admin(Stomatolog)**

Admin jest użytkownikiem, dla którego z góry przypisany jest stały login logowania o nazwie: *admin*. Dane admina mogą zostać zmienione w każdym momencie wewnątrz aplikacji. Tuż po zarejestrowaniu konta oraz zalogowaniu się do systemu loginem admin użytkownik staje zwiększa ilość funkcji względem pacjenta. Posiada funkcję takie jak: zarejestrowanie oraz anulowanie wizyty pacjentom poprzez podanie ich peselu, przeglądanie swoich przeszłych oraz nadchodzących wizyt, przeglądanie listy pacjentów wraz z ich danymi oraz wizytami, możliwość zwiększania oraz zmniejszania ilości danego produktu w magazynie, dodanie do bazy magazynu nowego produktu, trwale usunięcie z bazy magazynu danego produktu, przeglądanie swoich danych wraz z możliwością edycji. Może także wylogować się z systemu i powrócić do stanu obserwatora.

Program korzysta z 3 baz danych o nazwach:

- **Users** - przechowywane są dane zalogowanych użytkowników takie jak: Id pacjenta, pesel, imię, nazwisko, login, hasło, telefon, adres.
- **Visits** - przechowywane są dane zalogowanych użytkowników takie jak: Id wizyty, data, pesel pacjenta jeśli wizyta jest zarezerwowana(w innym przypadku komórka pesel posiada wartość 0).
- **Products** - przechowywane są dane produktów magazynu takie jak: Id produktu, nazwa, ilość, cena.

Za prawidłowe funkcjonowanie całej aplikacji odpowiada plik App.cpp, który służy przejściom między zakładkami programu. Jest to tak zwany "szkielet programu", bez którego program nie byłby w stanie prawidłowo działać. Potrzebna była do tego odpowiednie zbudowana pętla w kodzie pliku App.cpp. "Szkielet" jest połączony z poniższymi zakładkami:

```
#include "RegisterPage.h"
#include "LoginPage.h"
#include " MainPage.h"
#include "CreateVisit.h"
#include "MyVisitsPage.h"
#include "DataPage.h"
#include "AdminMainPage.h"
#include "AdminCreateVisit.h"
#include "AdminMyVisitPage.h"
#include "AdminPatients.h"
#include "AdminStore.h"
#include "AdminStoreAdd.h"
```

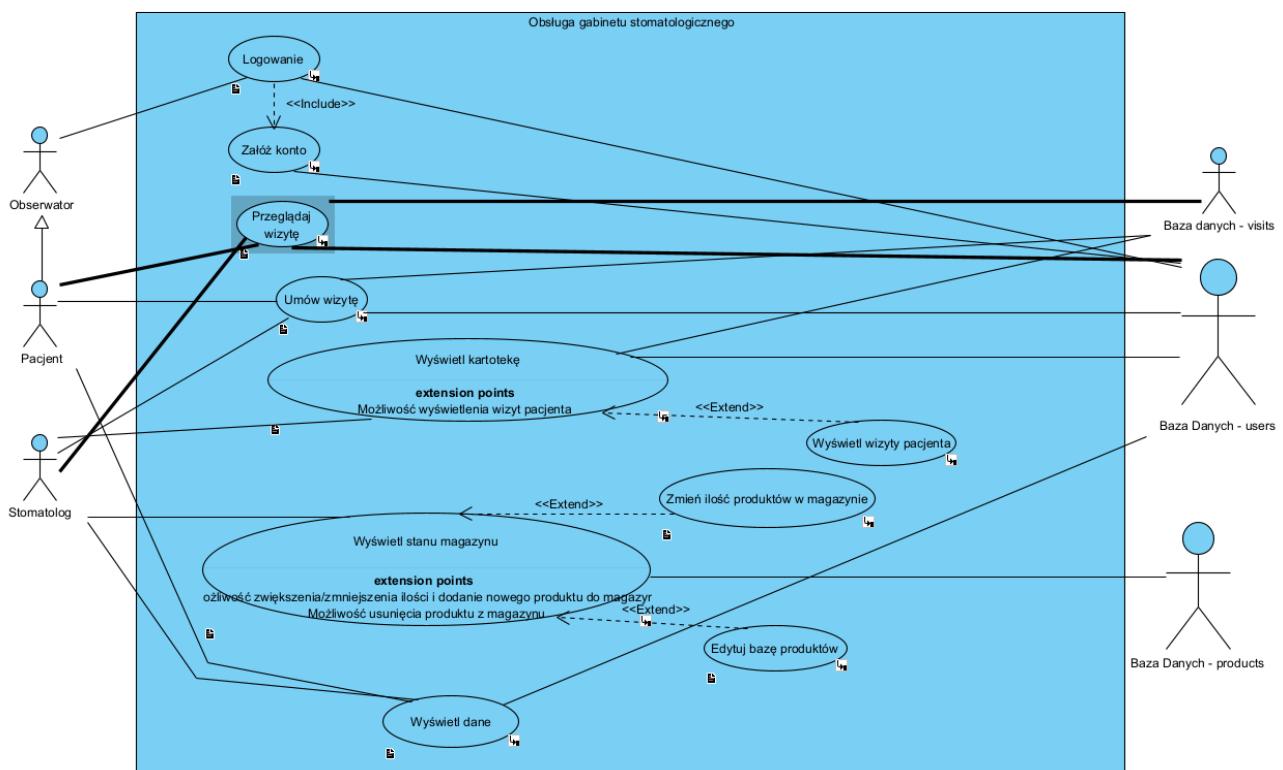
Kod aplikacji zbudowany jest z wielu przestrzeni nazw oraz funkcji bibliotecznych, które ułatwiało oraz przyspieszyło proces tworzenia kodu. Zapewniło to wybrane przez nas środowisko programistyczne. Przykłady:

```
using namespace System;
using namespace System::ComponentModel;
using namespace System::Collections;
using namespace System::Windows::Forms;
using namespace System::Data;
using namespace System::Drawing;
```

Za korzystanie z bazy danych oraz przechowywanie w nich istotnych informacji odpowiadał:

```
using namespace System::Data::SqlClient;
```

4. Diagram przypadków użycia.



Ze względu na specyfikację diagramu przypadków użycia cały jest możliwy do zobaczenia w dołączonym pliku programu Visual Paradigm.

https://drive.google.com/file/d/1--j_CRzX0-SFb4SLiQ16zdrzsPsyHOG/view?usp=sharing

5. Wymagania funkcjonalne i niefunkcjonalne.

Wymagania funkcjonalne

Zarządzanie kontem

- użytkownik może założyć swoje konto
- użytkownik może zmienić swoje dane
- użytkownik może sprawdzić swoje dane
- stomatolog (admin) może sprawdzić czyjeś dane

Zarządzanie wizytami

- użytkownik może umówić wizytę
- użytkownik może anulować wizytę
- stomatolog może ustalić/anulować wizytę danego pacjenta

Zarządzanie danymi

- stomatolog może przejrzeć kartotekę w której znajdują się dane pacjentów
- stomatolog może przeglądać wizyty pacjentów
- użytkownik może sprawdzić swoje dane osobowe

Zarządzanie magazynem

- stomatolog może dodać całkowicie nowy przedmiot do magazynu
- stomatolog może usunąć całkowicie dany przedmiot z magazynu
- stomatolog może zwiększyć ilość danego produktu w magazynie
- stomatolog może zmniejszyć ilość danego produktu w magazynie

Wymagania niefunkcjonalne

Przejrzystość

- Program jest czytelny, intuicyjny i prosty w użyciu

Bezpieczeństwo

- Bezpieczne przechowywanie danych konta

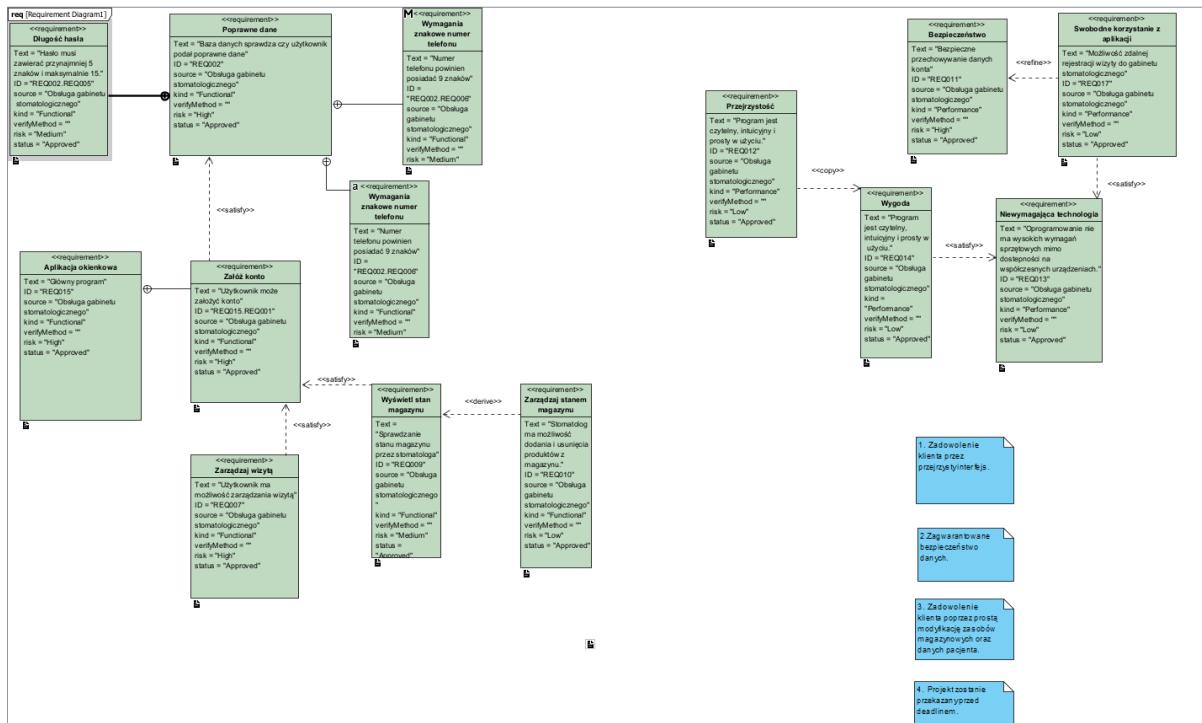
Wygoda

- Program jest bardzo prosty w użyciu

Niewymagająca technologia

- Oprogramowanie nie ma wysokich wymagań sprzętowych mimo dostępności na współczesnych urządzeniach

6. Diagramy wymagań.



7. Wykaz zastosowanych framework-ów, bibliotek, środowiska, technologii oraz informacje dotyczące zastosowanych rozwiązań informatycznych.

Środowisko programowania: Visual Studio 2022, Microsoft SQL Server 2019

Środowisko do zarządzania bazą danych: Microsoft SQL Server Management Studio 18

Zastosowane frameworki: .NET Framework

Zastosowane biblioteki: Windows Forms(CLR)

Technologie programowania: C++, SQL

Narzędzie wspierające programowanie: Trello

Dodatki potrzebne do uruchomienia projektu w środowisku Visual Studio 2022:

- CLR:



Programowanie aplikacji klasycznych w języku C++



Kompiluj nowoczesne aplikacje C++ dla systemu Windows przy użyciu wybranych przez siebie narzędzi, takich jak M...

▼ Programowanie aplikacji klasyczny... ●

▼ Dołączone

Podstawowe funkcje języka C++ dla komp...

▼ Opcjonalne

MSVC wersja 143 — VS 2022 — narzędzia...

C++ ATL dla najnowszych narzędzi kompil...

Windows 11 SDK (10.0.22000.0)

Debugger just in time

Narzędzia profilowania dla języka C++

Narzędzia CMake języka C++ dla systemu...

Adapter testowy dla platformy Boost.Test

Test Adapter for Google Test

Live Share

IntelliCode

C++ AddressSanitizer

C++ MFC dla najnowszych narzędzi kompilacj...

Moduły języka C++ dla narzędzi kompilacj...

Windows 11 SDK (10.0.22621.0)

Obsługa C++/CLI dla narzędzi kompilacji...

Narzędzia C++ Clang dla systemu Window...

Diagnostyka JavaScript

Incredibuild — przyspieszanie kompilacji

Windows 10 SDK (10.0.20348.0)

Windows 10 SDK (10.0.19041.0)

Windows 10 SDK (10.0.18362.0)

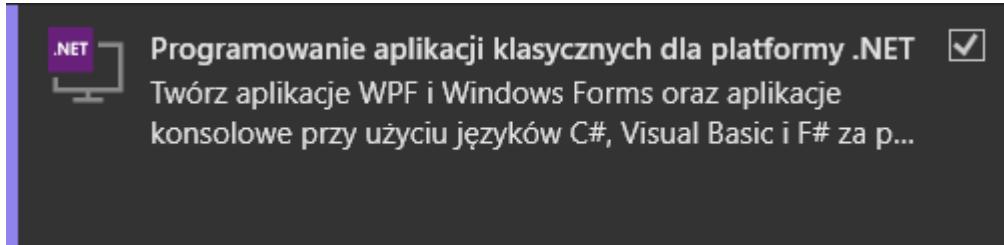
MSVC wersja 142 — VS 2019 — narzędzia...

MSVC wersja 141 — VS 2017 — narzędzia...

MSVC wersja 140 — VS 2015 — narzędzia...

Szablony języka C++ zestawu SDK do aplik...

- .NET Framework



The screenshot shows the Visual Studio installer interface. A purple sidebar on the left has a 'Dołącz do instalacji' (Add to installation) button. The main area displays the '.NET Framework' section under 'Programowanie aplikacji klasycznych dla platformy .NET'. It includes a checked checkbox for 'Twórz aplikacje WPF i Windows Forms oraz aplikacje konsolowe przy użyciu języków C#, Visual Basic i F# za p...'. Below this, there are two expandable sections: 'Dołączone' (Attached) and 'Opcjonalne' (Optional). The 'Opcjonalne' section contains several checkboxes, some of which are checked.

▼ Programowanie aplikacji klasycznych...

▼ Dołączone

- ✓ Narzędzia do programowania aplikacji klas...
- ✓ .NET Framework 4.7.2 Developer Tools
- ✓ C# i Visual Basic

▼ Opcjonalne

- ✓ Narzędzia programistyczne dla platformy ...
- ✓ Narzędzia programistyczne platformy .NET...
- ✓ Entity Framework 6 Tools
- ✓ Narzędzia profilowania dla programu .NET
- ✓ IntelliCode
- ✓ Debugger just in time
- ✓ Live Share
- ✓ ML.NET Model Builder
- ✓ Blend for Visual Studio
- Obsługa języka F# dla komputerów
- PreEmptive Protection - Dotfuscator
- Narzędzia programistyczne platformy .NET...
- Dodatek Targeting Pack dla biblioteki prze...

- ✓ Windows Communication Foundation
- ✓ SQL Server Express 2019 LocalDB
- ✓ MSIX Packaging Tools
- ✓ Diagnostyka JavaScript
- Szablony języka C# zestawu SDK do aplikacj...
- Narzędzia programistyczne platformy .NET...

Do uruchomienia aplikacji konieczne jest utworzenie i podłączenie do programu bazy danych Microsoft SQL Server o nazwie: **Dentist**

Następnie należy utworzyć 3 tabele: **Users**, **Visits**, **Products**. Poniżej wklejony jest kod danej tabeli.

➤ Users

```
CREATE TABLE [dbo].[Users] (
    [Id]      INT      IDENTITY (1, 1) NOT NULL,
    [pesel]   VARCHAR (11) NOT NULL,
    [name]    VARCHAR (20) NOT NULL,
    [surname] VARCHAR (20) NOT NULL,
    [login]   VARCHAR (20) NOT NULL,
    [password] VARCHAR (20) NOT NULL,
    [phone]   VARCHAR (9) NOT NULL,
    [address] VARCHAR (50) NOT NULL,
    PRIMARY KEY CLUSTERED ([Id] ASC),
    UNIQUE NONCLUSTERED ([pesel] ASC),
    UNIQUE NONCLUSTERED ([login] ASC)
);
```

➤ Visits

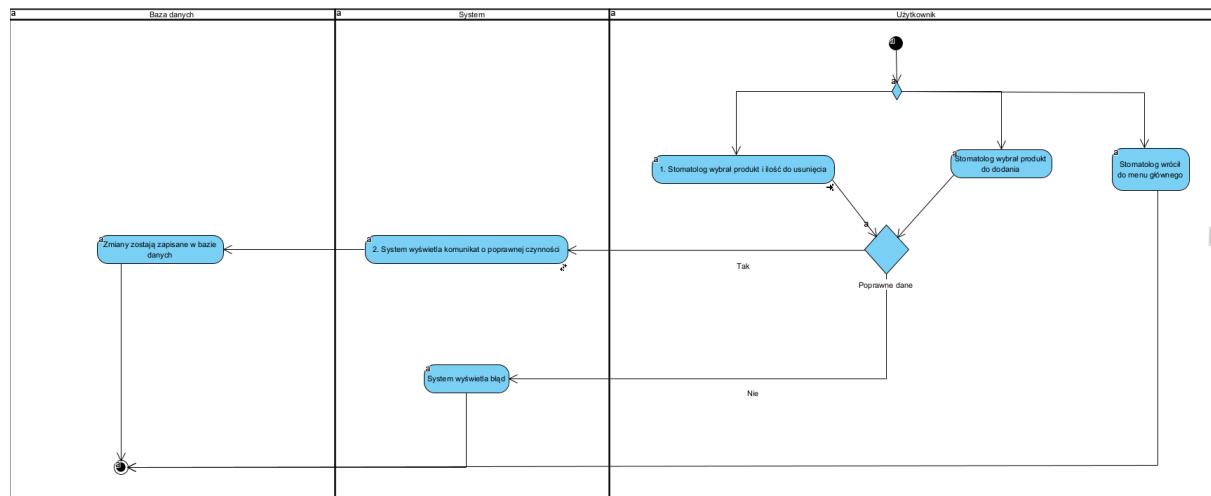
```
CREATE TABLE [dbo].[Visits] (
    [Id] INT IDENTITY (1, 1) NOT NULL,
    [date] DATETIME NOT NULL,
    [pesel] VARCHAR (11) NOT NULL,
    PRIMARY KEY CLUSTERED ([Id] ASC),
    UNIQUE NONCLUSTERED ([date] ASC)
);
```

➤ Products

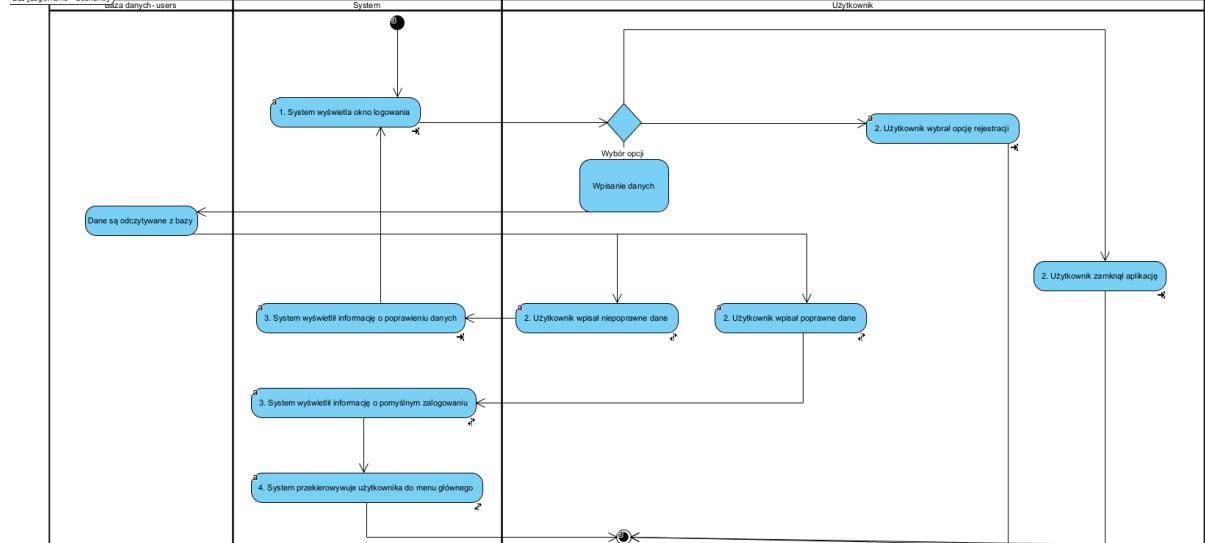
```
CREATE TABLE [dbo].[Products] (
    [Id] INT IDENTITY (1, 1) NOT NULL,
    [name] VARCHAR (20) NOT NULL,
    [amount] INT NOT NULL,
    [price] FLOAT (53) NOT NULL,
    PRIMARY KEY CLUSTERED ([Id] ASC),
    UNIQUE NONCLUSTERED ([name] ASC)
);
```

8. Diagramy aktywności.

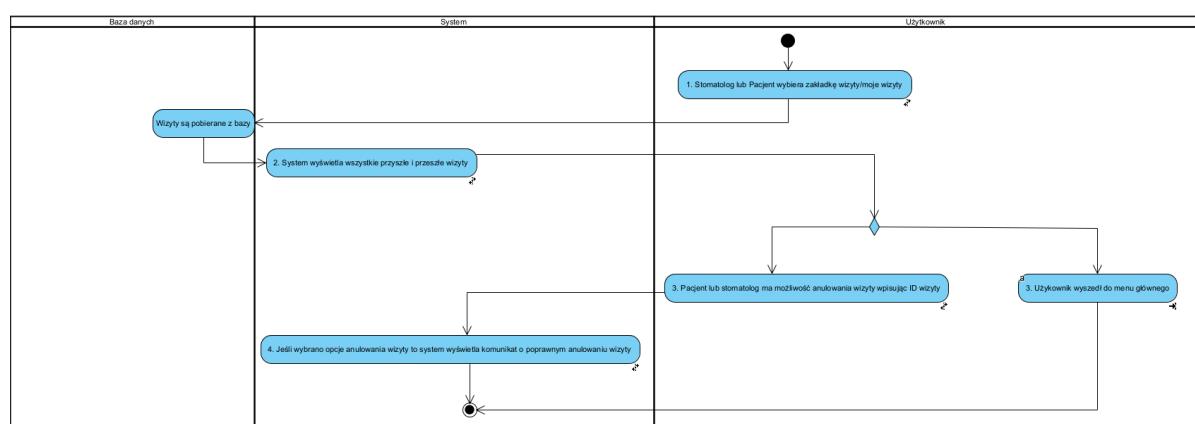
act [Edytuj bazę produktów - Scenariusz]

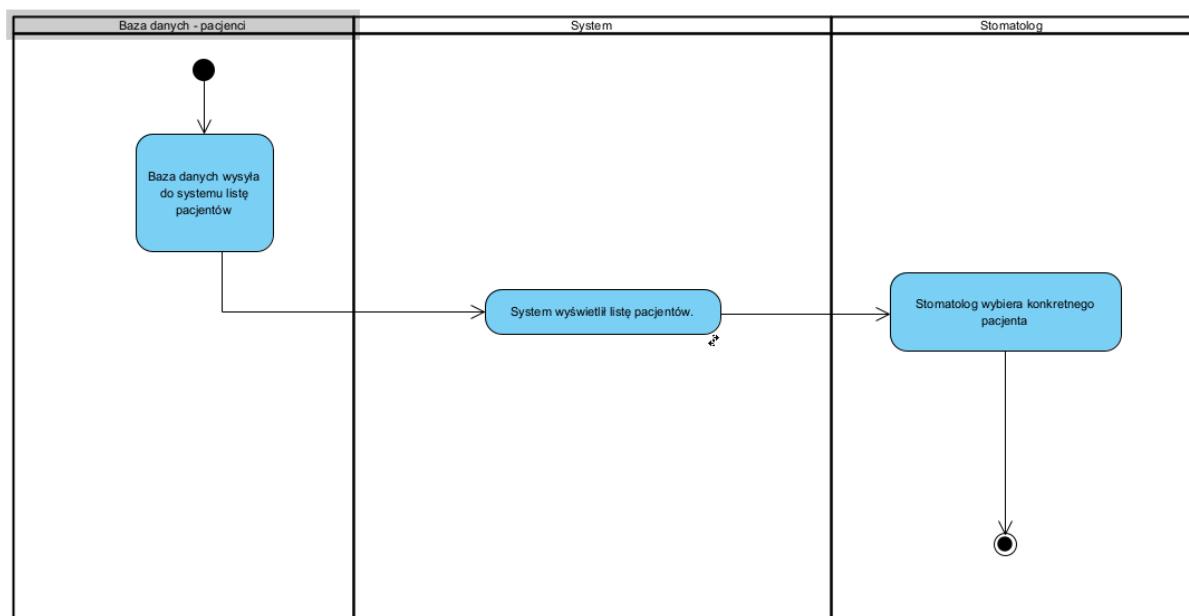
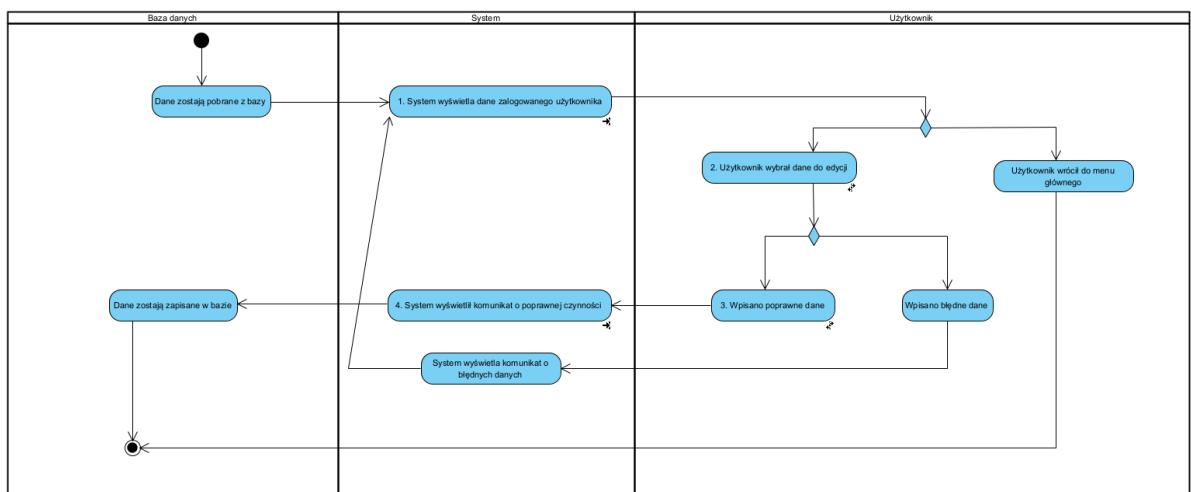
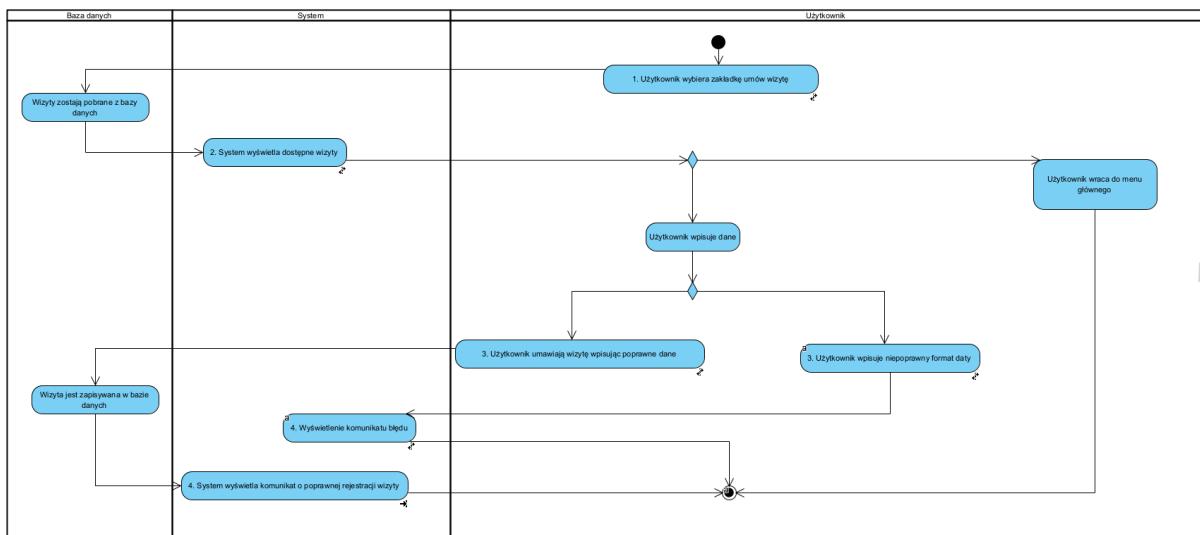


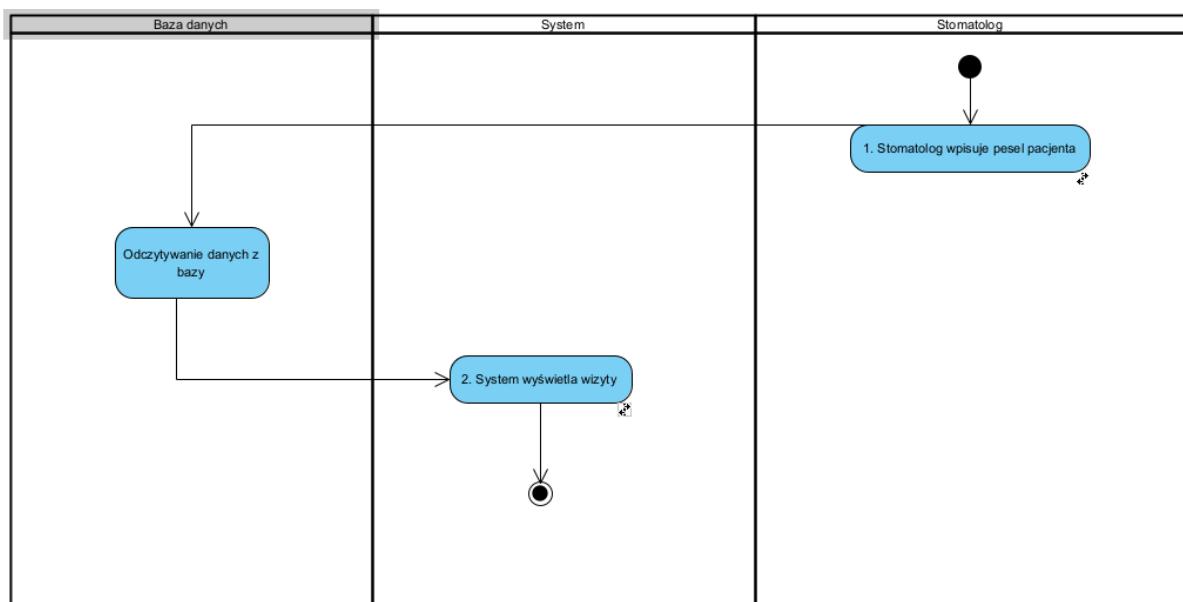
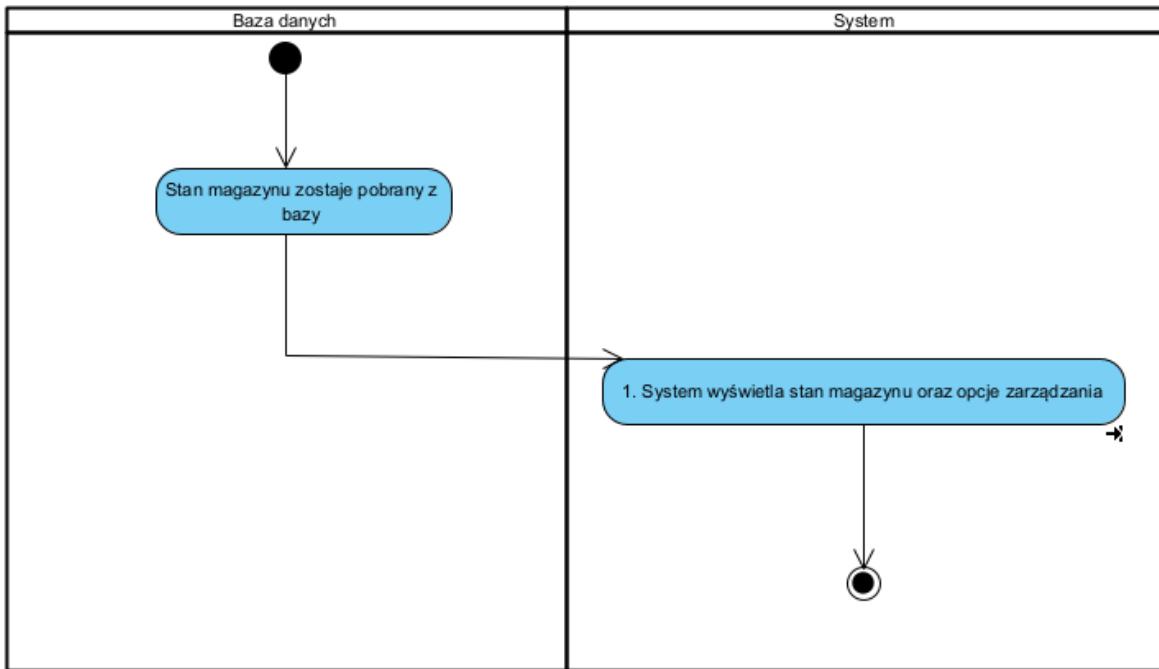
act [Logowanie - Scenario]

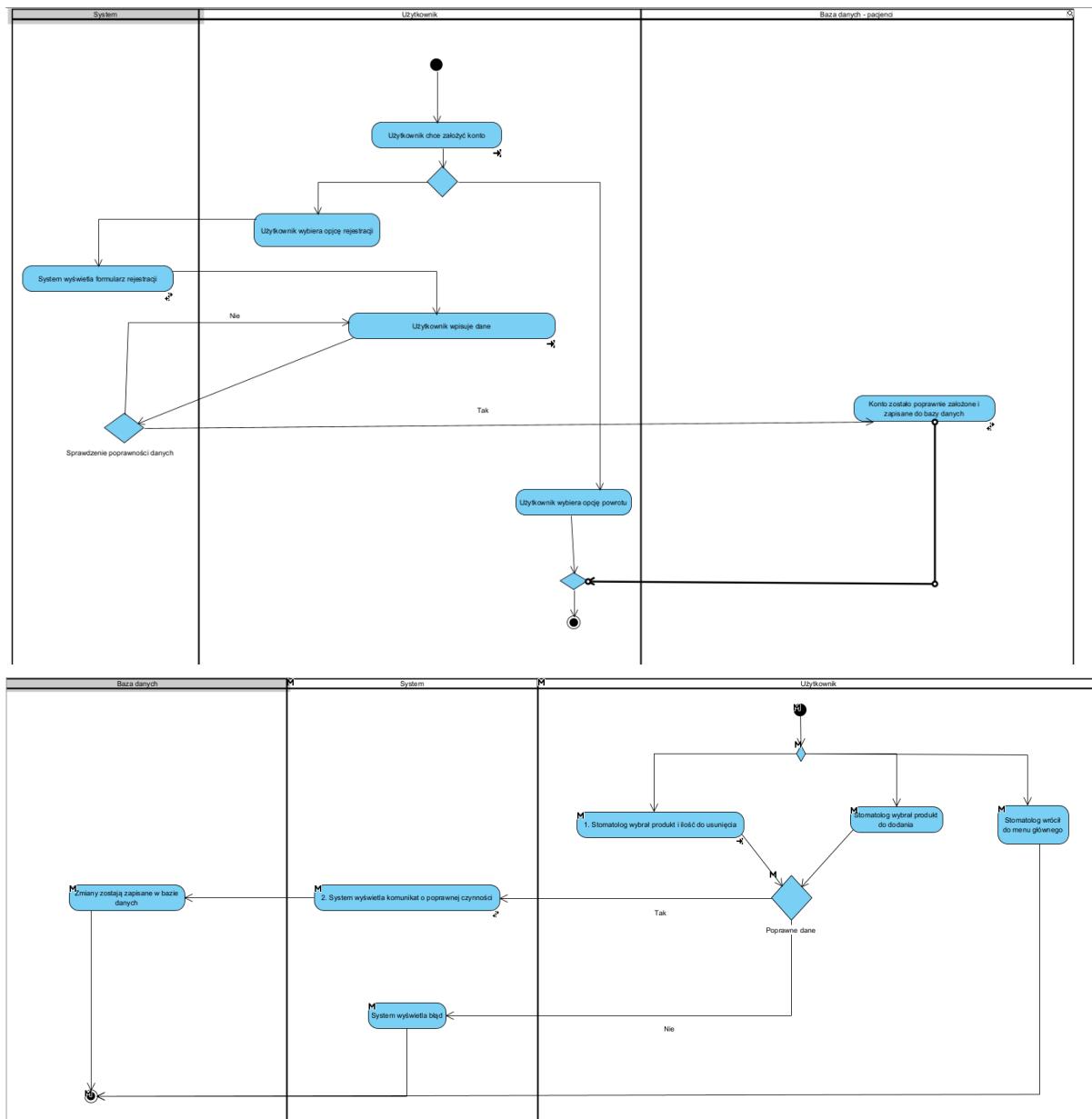


→ (Downloaded with the - Connection)

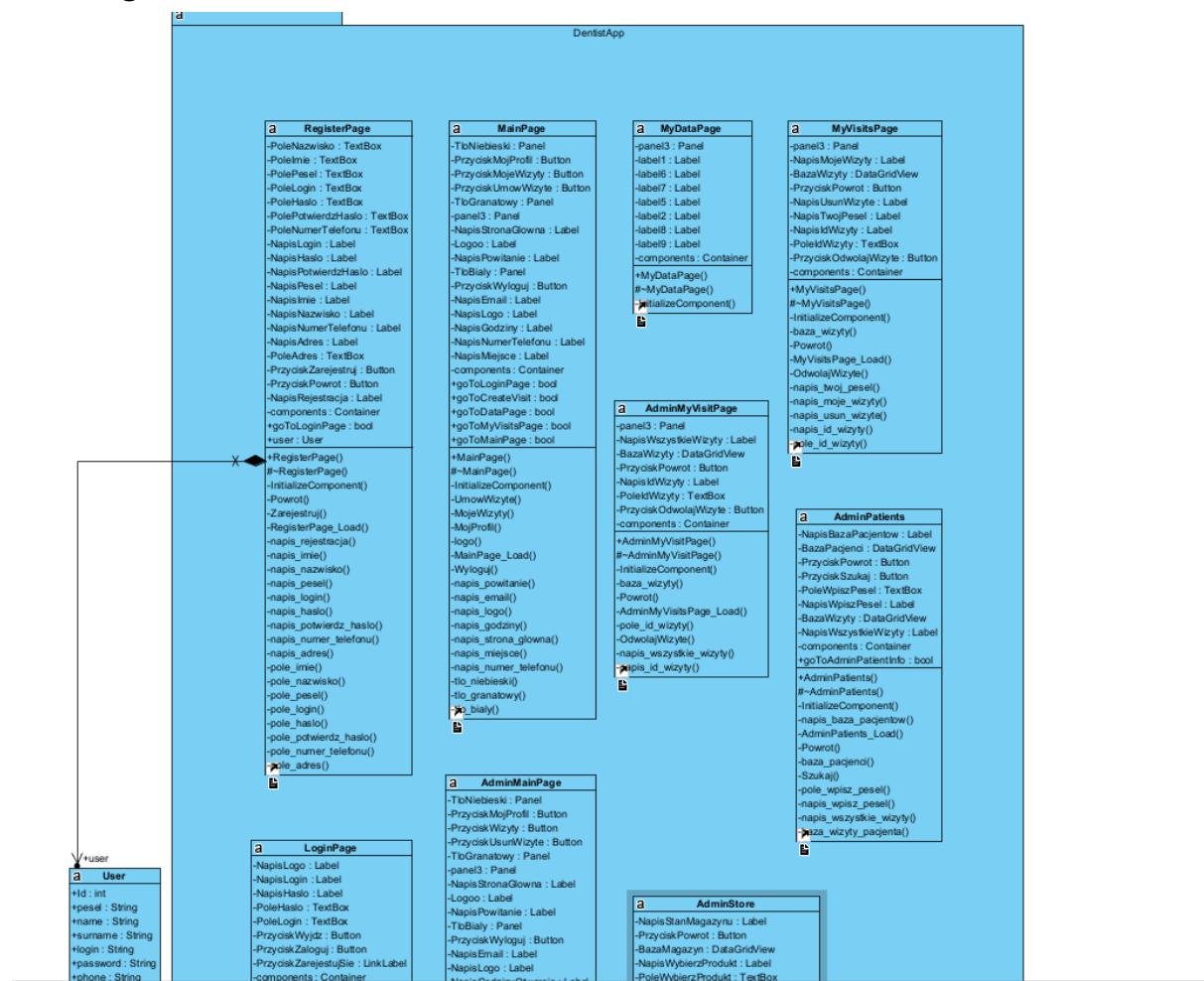


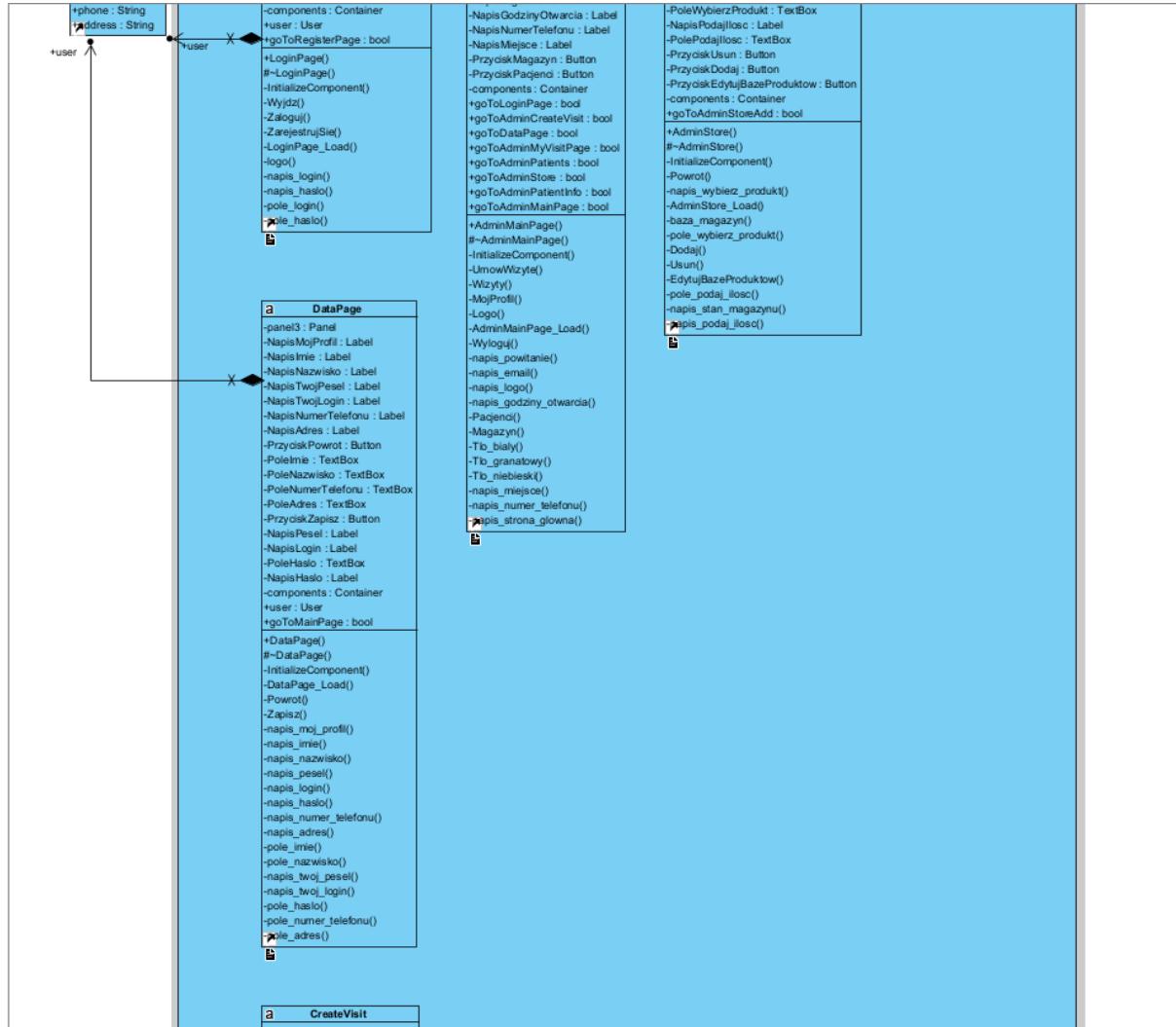


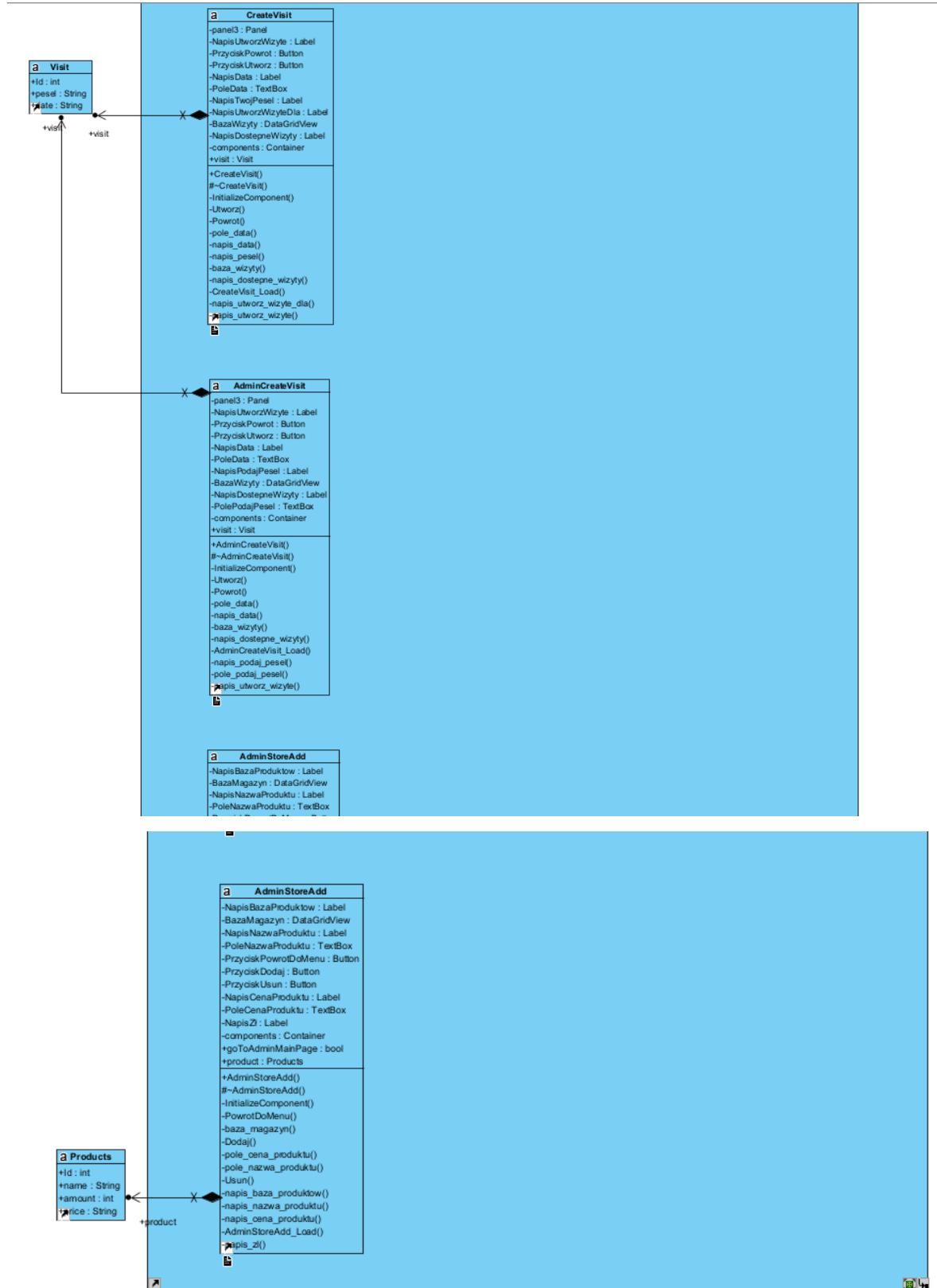




9. Diagram klas.

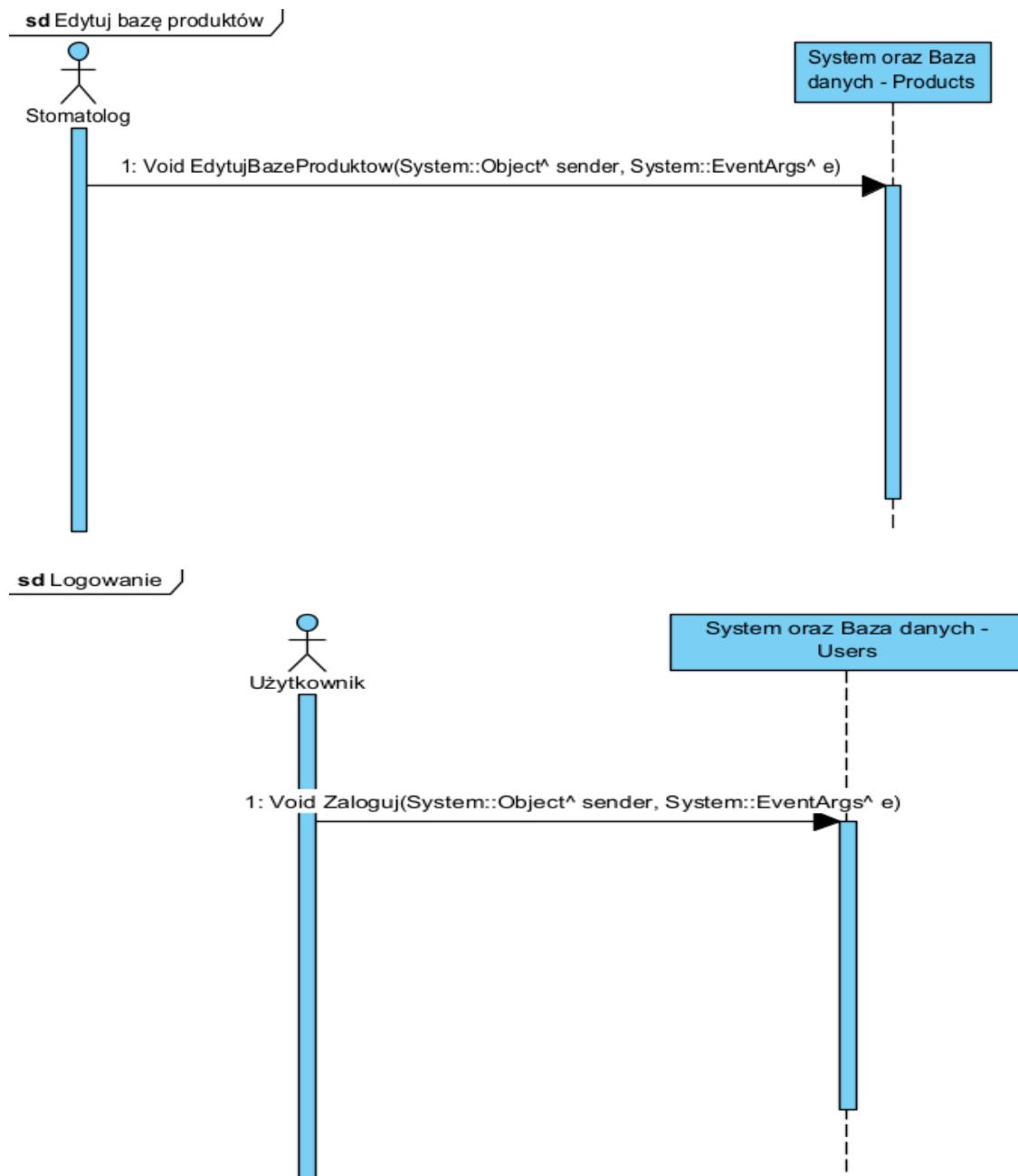


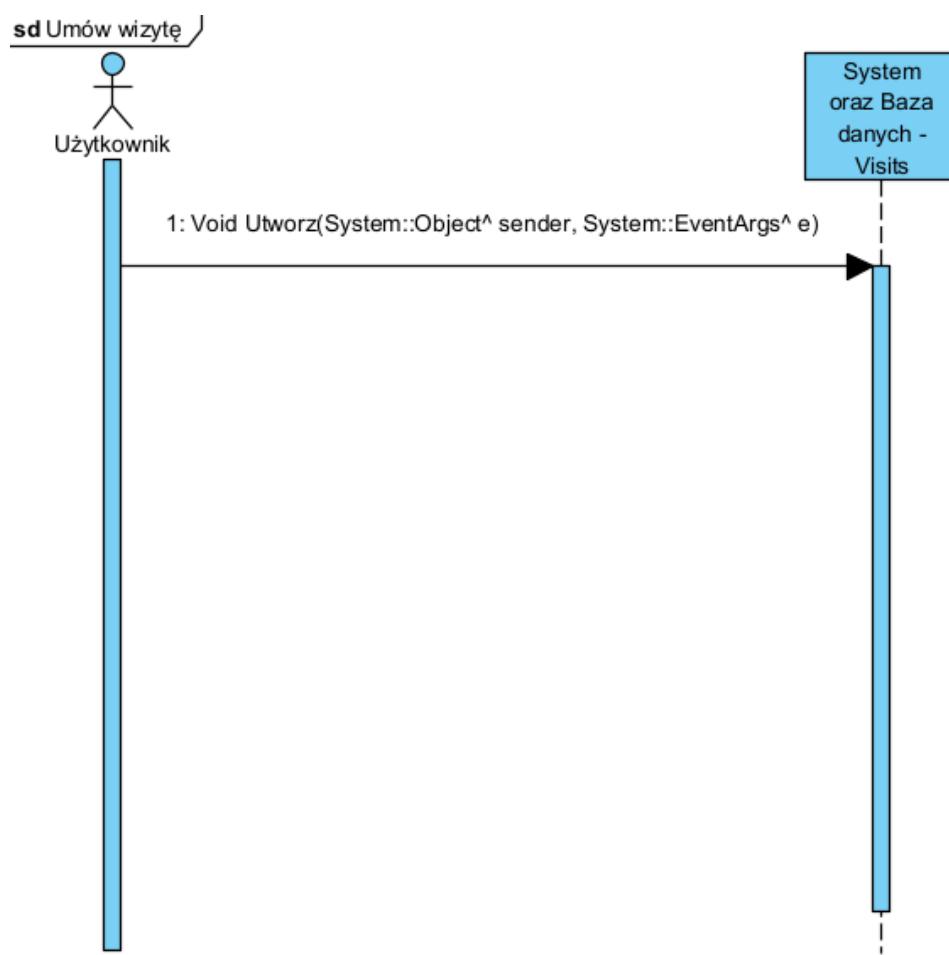
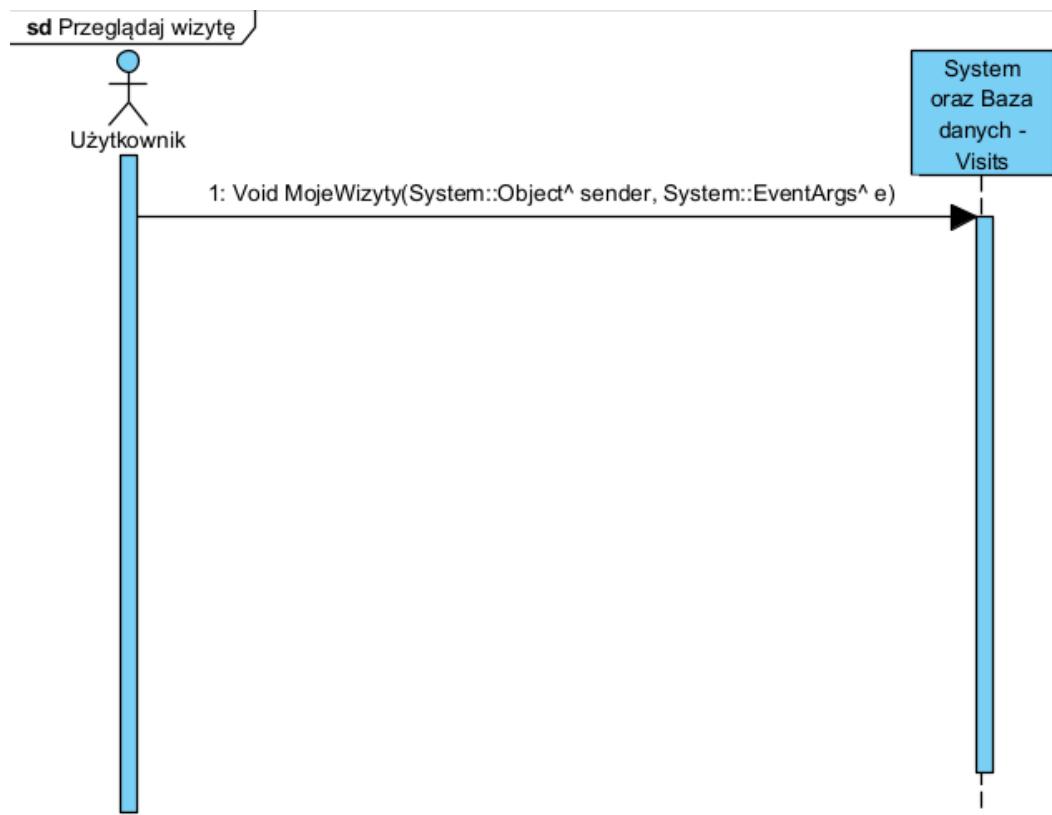


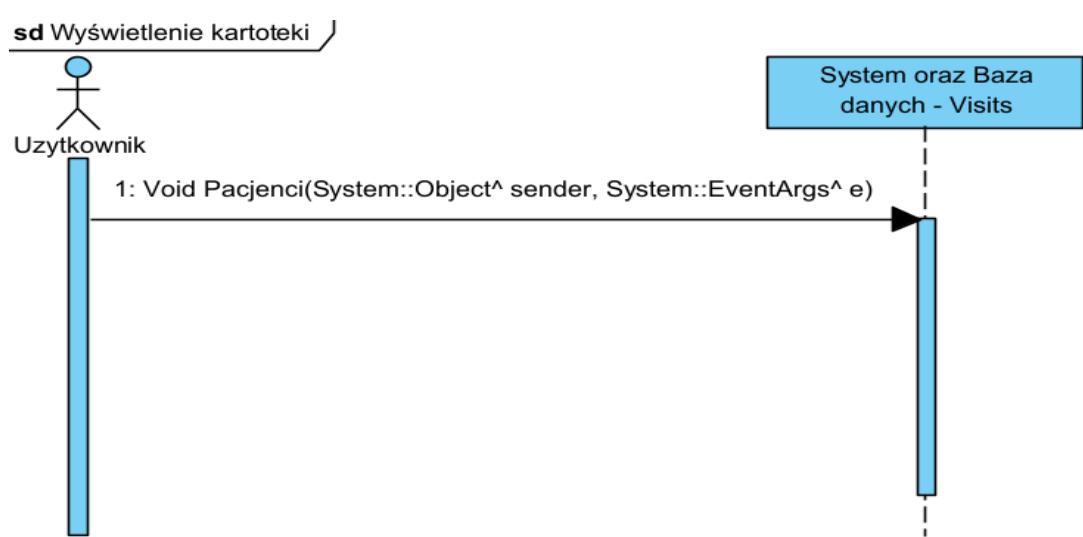
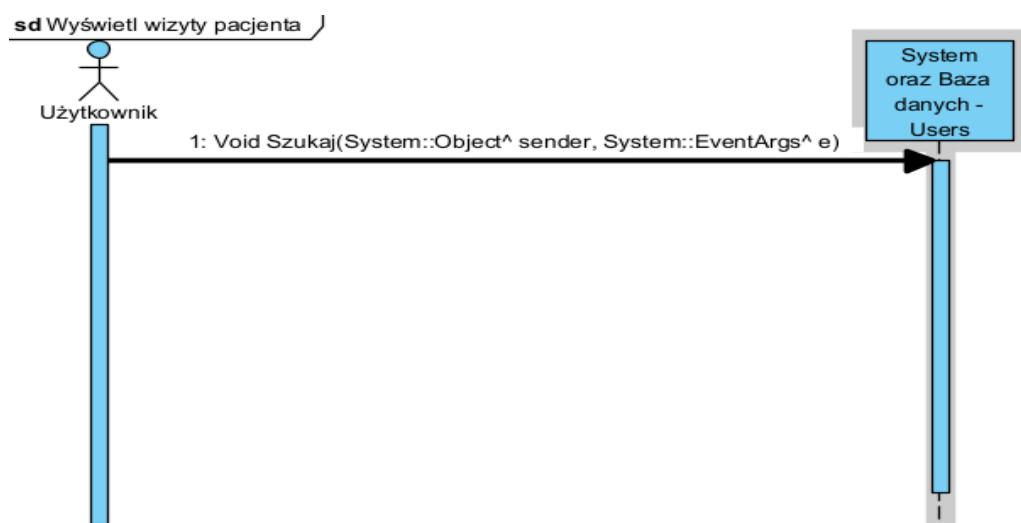
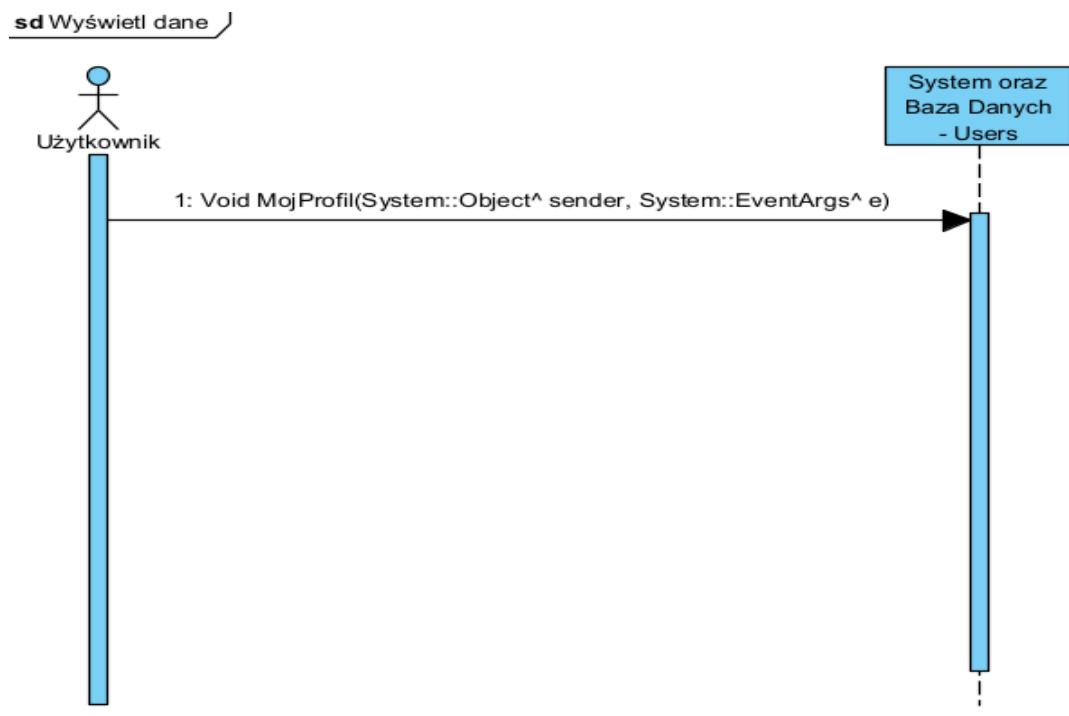


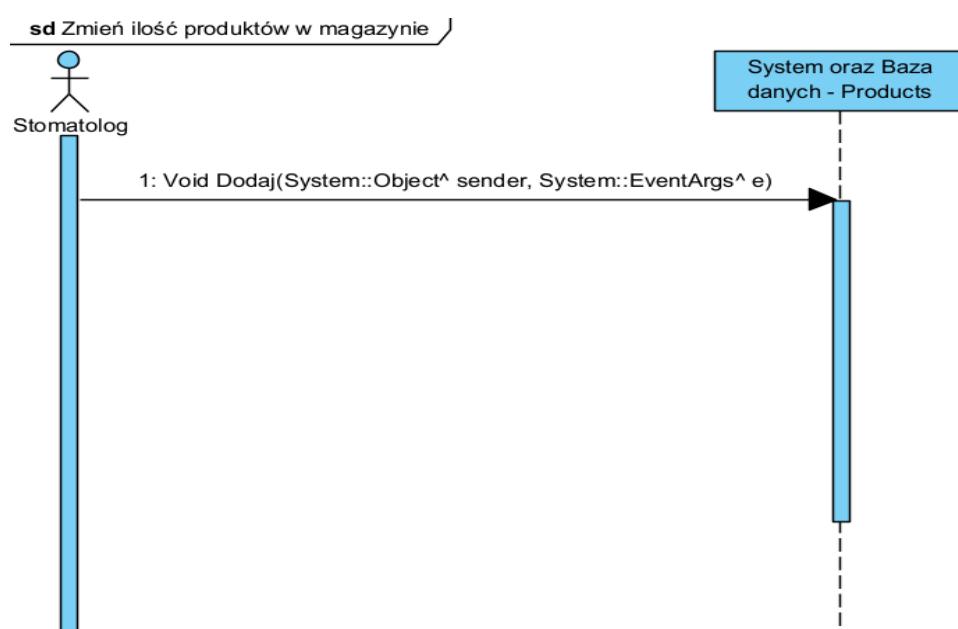
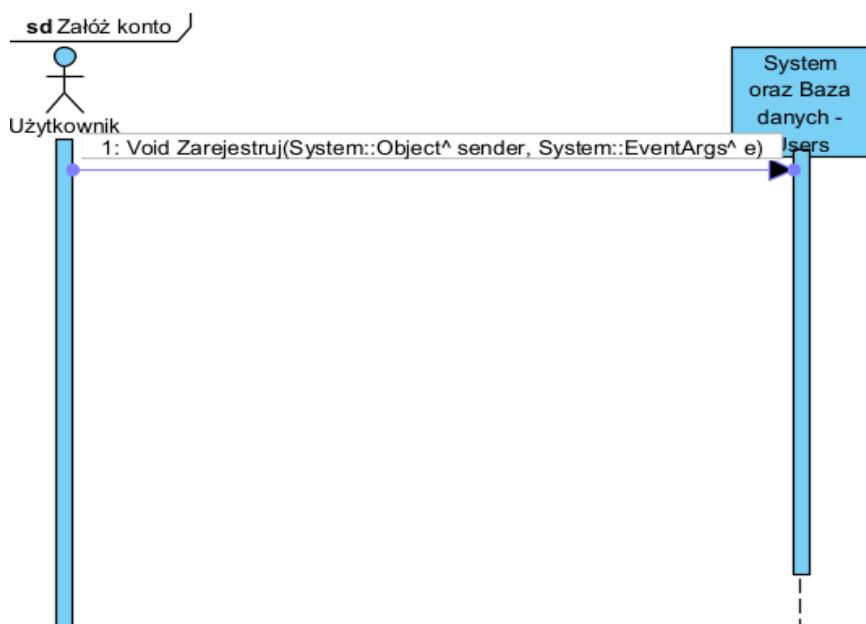
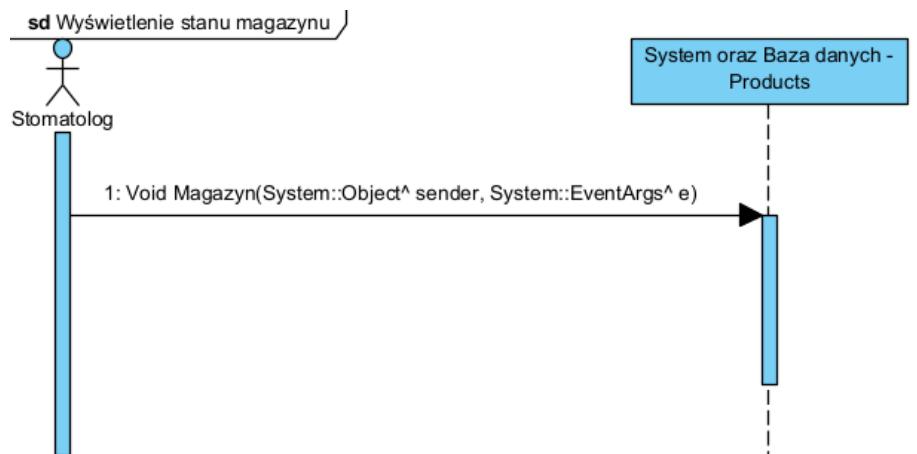
10. Diagramy sekwencji

Diagramy skonstruowano w jak najprostszy sposób, gdyż działania każdego z przypadku użycia zawierają się w jednej funkcji, a nie zostały podzielone na kilka mniejszych metod. Z tego powodu każdy diagram sekwencji zawiera jedną strzałkę oznaczającą wywoływanie funkcji, która posiada wszystkie działania takie jak np. odczytywanie danych z bazy czy wyświetlenie finalnego komunikatu.

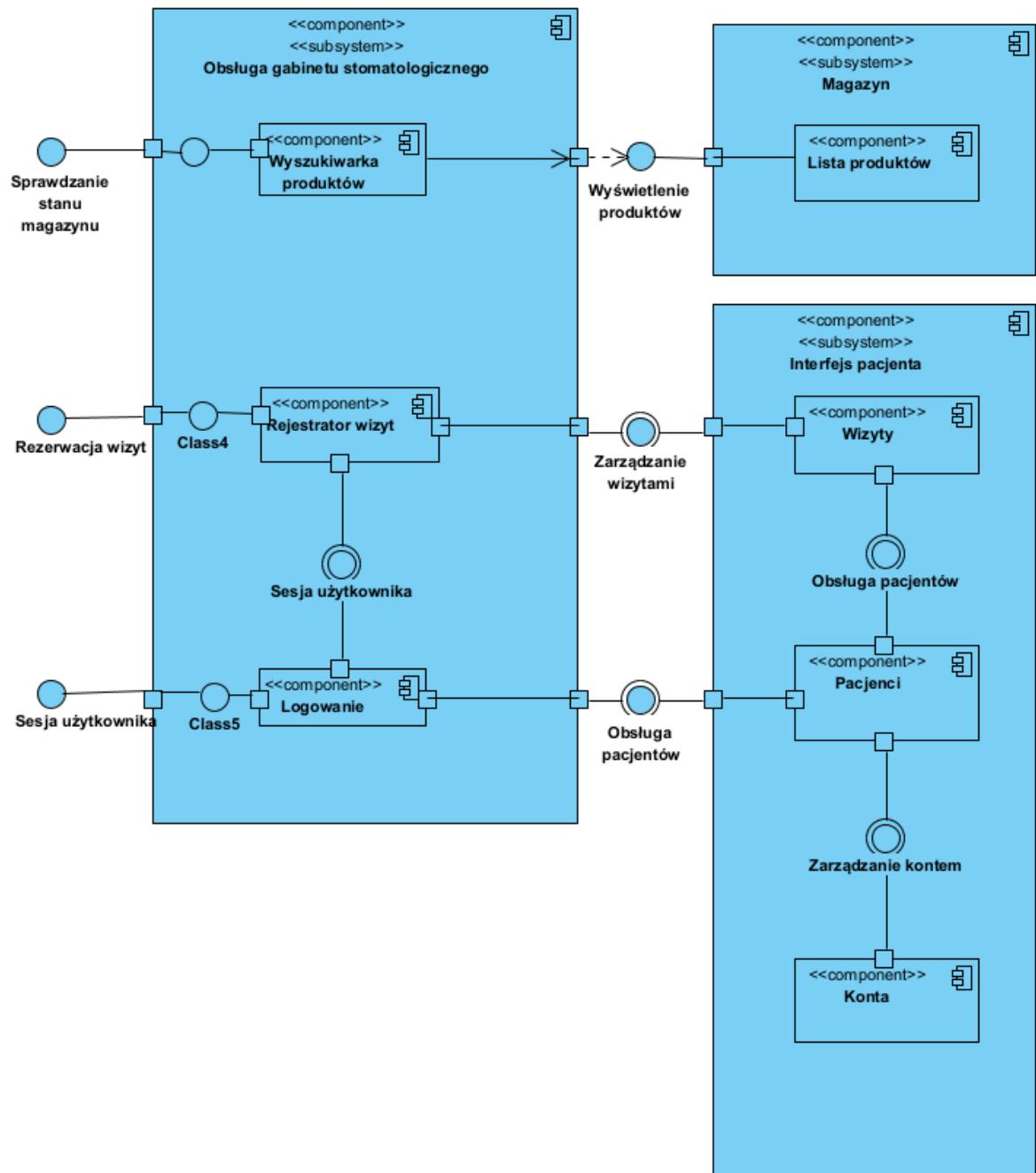




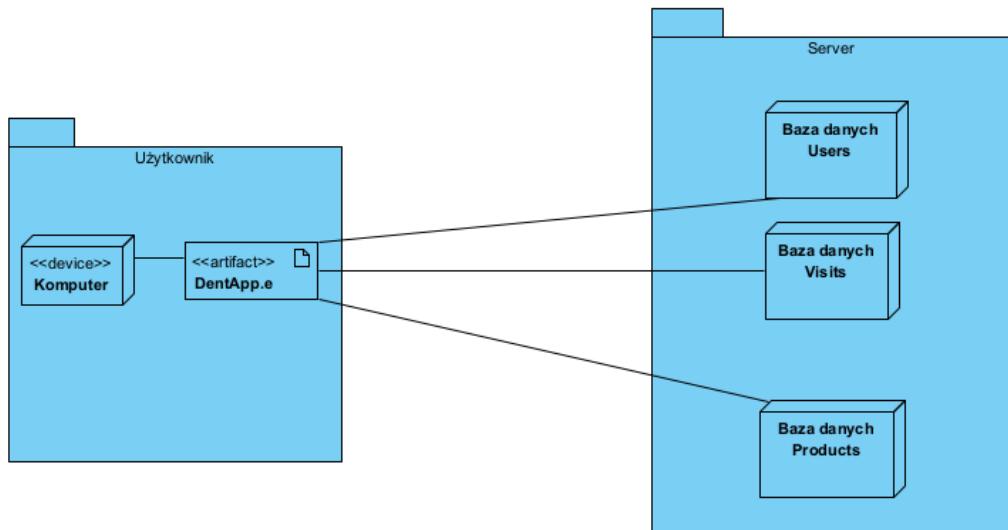




11. Diagram komponentów.



12. Diagram wdrożenia.



13. Przykłady współpracy zespołu z wykorzystaniem narzędzia Trello.

W pracy przy projekcie zdecydowaliśmy się skorzystać z takich narzędzi jak Trello oraz Discord na którym łączymy się czatem głosowym.

Przykładowe zastosowanie Trello:

Gabinet Stomatologiczny Darmowe

Dokumentacja

Widoczne dla Przestrzeni roboczej | Tablica

Do zrobienia

- Testy jednostkowe
- Przykład refaktoryzacji kodu
- Przykład wzorca projektowego
- Podsumowanie

+ Dodaj kartę

W trakcie

- Diagramy 4/7
- Słownik pojęć i terminów

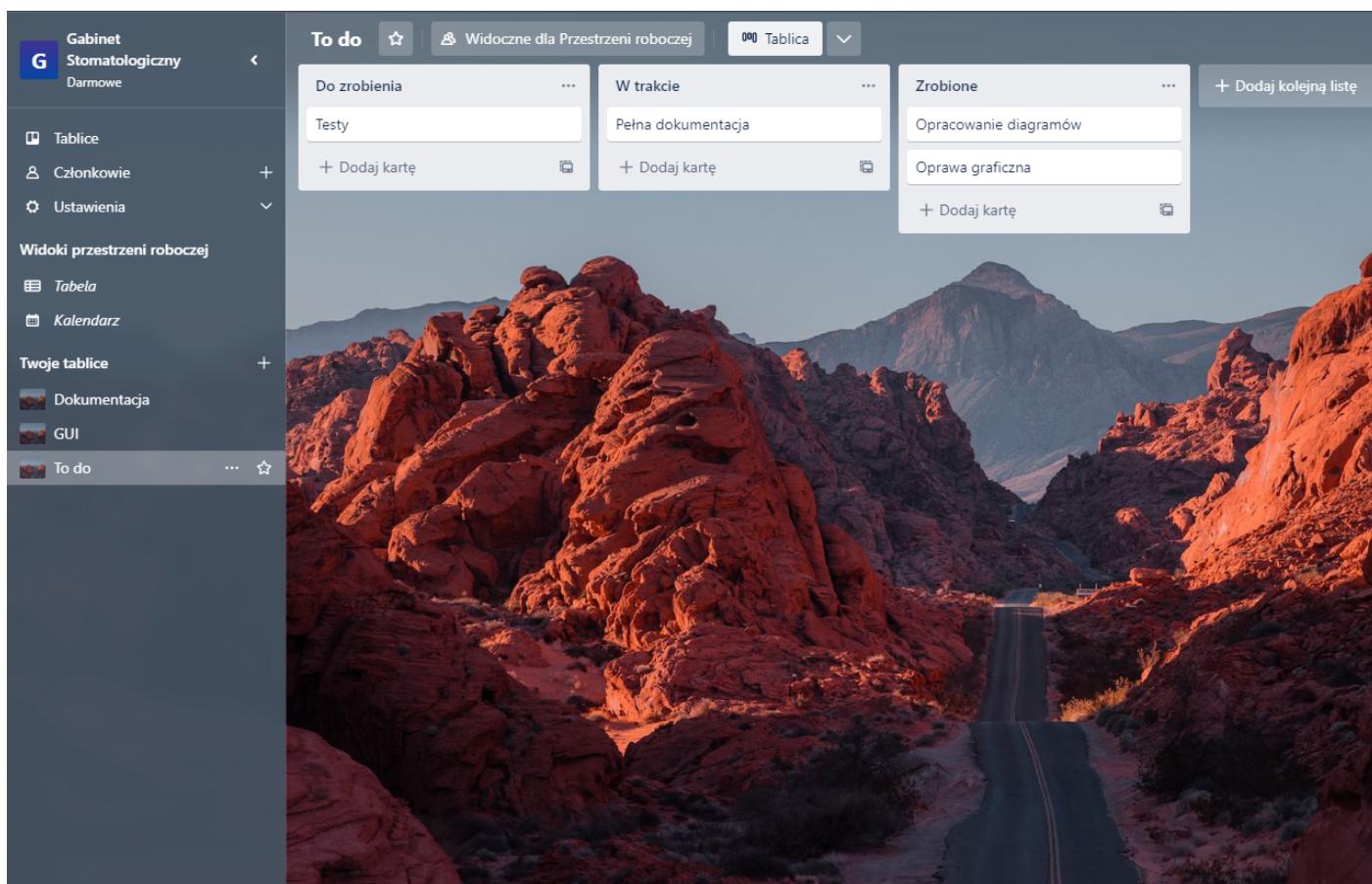
+ Dodaj kartę

Zrobione

- Strona tytułowa
- Wymagania funkcjonalne i niefunkcjonalne
- Opis metodyki
- Specyfikacja projektu
- Wykaz zastosowanych frameworków

+ Dodaj kartę

Tablice | Członkowie | Ustawienia | Widoki przestrzeni roboczej | Twoje tablice | GUI | To do



Diagramy

na liście [W trakcie](#)

Opis

Dodaj bardziej szczegółowy opis...

Diagram przypadków użycia (pełny opis)

[Ukryj zaznaczone elementy](#)

[Usuń](#)

100%



s

[Dodaj element...](#)

Diagram wymagań

[Ukryj zaznaczone elementy](#)

[Usuń](#)

100%



:

[Dodaj element...](#)

Diagram aktywności

[Ukryj zaznaczone elementy](#)

[Usuń](#)

100%



:

[Dodaj element...](#)

Diagram klas

[Ukryj zaznaczone elementy](#)

[Usuń](#)

100%



:

[Dodaj element...](#)

[Dodaj do karty](#)

Członkowie

Etykiety

Lista zadań

Daty

Załącznik

Okładka

Pola niestandardo...

Dodawaj listy rozwijane, pola tekstowe, daty i wiele więcej do swoich kart.

Rozpocznij bezpłatny okres próbny

Dodatki Power-Up

[+ Dodaj dodatek Po...](#)

Automatyzacja



[+ Dodaj przycisk](#)

Działania

Przenieś

Kopiuj

Utwórz szablon

Obserwuj

Zarchiwizuj

Udostępnij

Dodaj element...

→ Przenieś
Kopiuj
Utwórz szablon
Obserwuj
Zarchiwizuj
Udostępnij

Diagram klas [Ukryj zaznaczone elementy](#) [Usuń](#)

100%

Dodaj element...

Diagram sekwencji [Usuń](#)

0%

Dodaj element...

Diagram komponentów [Usuń](#)

0%

Dodaj element...

Diagram wdrożenia [Usuń](#)

0%

Dodaj element...

Aktywność [Pokaż Szczegóły](#)

Napisz komentarz...

Menu X

- KD** zastosowanych frameworków z Do zrobienia do Zrobione
Zeszły poniedziałek o 21:57
- KD** Kamil Dziewa przeniósł Opis metodyki z Do zrobienia do Zrobione
Zeszły poniedziałek o 21:57
- KD** Kamil Dziewa przeniósł Instrukcja użytkowania programu z Do zrobienia do W trakcie
Zeszły poniedziałek o 21:56
- KD** Kamil Dziewa przeniósł Przykłady współpracy (trello) z Do zrobienia do W trakcie
Zeszły poniedziałek o 21:56
- KD** Kamil Dziewa przeniósł Spis treści z Do zrobienia do W trakcie
Zeszły poniedziałek o 21:56
- KD** Kamil Dziewa przeniósł Wymagania funkcjonalne i niefunkcjonalne z Do zrobienia do Zrobione
Zeszły poniedziałek o 21:55
- KD** Kamil Dziewa ukończył . w Diagramy
Zeszły poniedziałek o 21:55
- KD** Kamil Dziewa ukończył . w Diagramy
Zeszły poniedziałek o 21:55
- KD** Kamil Dziewa ukończył . w Diagramy
Zeszły poniedziałek o 21:55
- KD** Kamil Dziewa ukończył s w Diagramy
Zeszły poniedziałek o 21:55
- KD** Kamil Dziewa dodał Wymagania funkcjonalne i niefunkcjonalne do Do zrobienia
3 sty o 16:04

Aby zobaczyć szczegóły należy udać się pod adres:

https://trello.com/invite/b/1fg738xd/ATTlb10e7bf97e4b51c75519b0fbb37db57025651775/do_kumentacja

14. Testy jednostkowe.

```
#include "pch.h"
#include "gtest/gtest.h"
#include "../DentistApp/AdminStore.h"

using namespace DentistApp;

TEST(TestCaseName, TestName) {
    int a=Dodaj(15);
    EXPECT_EQ(1, a);
    EXPECT_TRUE(true);
}
```

Testowanie czy dodawanie ilości produktu jako liczba przechodzi poprawnie przez program. Test został pomyślnie zakończony.

```
#include "pch.h"
#include "gtest/gtest.h"
#include "../DentistApp/AdminStore.h"

using namespace DentistApp;

TEST(TestCaseName, TestName) {
    testing::internal::string a=Utworz("2030.01.24");//wizyte
    EXPECT_EQ("2030-01-24", a);
    EXPECT_TRUE(false);
}
```

Test sprawdzający poprawność formatu wpisanej daty. W naszym programie data powinna być wpisana w formacie YEAR-MM-DD. A w teście wprowadziliśmy YEAR.MM.DD. Dlatego spodziewamy się wartości false. Test został pomyślnie zakończony.

15. Przykłady refaktoryzacji kodu.

Przed refaktoryzacją kod był nieczytelny, nazwy typu button1, label3 itp. nic nie mówiły.

```
private: System::Windows::Forms::Panel^ panel3;
protected:
private: System::Windows::Forms::Label^ label1;
private: System::Windows::Forms::Button^ button1;
private: System::Windows::Forms::Button^ button2;
private: System::Windows::Forms::Label^ label6;
private: System::Windows::Forms::TextBox^ textBox1;
private: System::Windows::Forms::Label^ label3;
private: System::Windows::Forms::DataGridView^ dataGridView1;
private: System::Windows::Forms::Label^ label4;
private: System::Windows::Forms::TextBox^ textBox2;
```

Dlatego należało je zmienić, w ten sposób osoba, która przejmie kod będzie wiedziała z czym ma do czynienia.

```
private: System::Windows::Forms::Panel^ panel3;
private: System::Windows::Forms::Label^ NapisUtworzWizyte;
private: System::Windows::Forms::Button^ PrzyciskPowrot;
protected:
private: System::Windows::Forms::Button^ PrzyciskUtworz;
private: System::Windows::Forms::Label^ NapisData;
private: System::Windows::Forms::TextBox^ PoleData;
private: System::Windows::Forms::Label^ NapisPodajPesel;
private: System::Windows::Forms::DataGridView^ BazaWizyty;
private: System::Windows::Forms::Label^ NapisDostepneWizyty;
private: System::Windows::Forms::TextBox^ PolePodajPesel;
```

Kod bez komentarzy może być nieczytelny.

```
this->label1->Font = (gcnew System::Drawing::Font(L"Comic Sans MS", 18, System::Drawing::FontStyle::Bold));
static_cast<System::Byte>(238)));
this->label1->ForeColor = System::Drawing::Color::White;
this->label1->Location = System::Drawing::Point(0, 0);
this->label1->Name = L"label1";
this->label1->Size = System::Drawing::Size(608, 73);
this->label1->TabIndex = 0;
this->label1->Text = L"Utwórz wizytę dla pacjenta";
this->label1->ContentAlignment = System::Drawing::ContentAlignment::MiddleCenter;

this->button1->BackColor = System::Drawing::SystemColors::Menu;
this->button1->Font = (gcnew System::Drawing::Font(L"Comic Sans MS", 7.8F, System::Drawing::FontStyle::Bold));
static_cast<System::Byte>(238)));
this->button1->Location = System::Drawing::Point(360, 453);
this->button1->Name = L"button1";
this->button1->Size = System::Drawing::Size(225, 51);
this->button1->TabIndex = 29;
this->button1->Text = L"Powrót";
this->button1->UseVisualStyleBackColor = false;
this->button1->Click += gcnew System::EventHandler(this, &AdminCreateVisit::button1_Click);
```

Z tego powodu tym też się zajęliśmy.

```
// NapisUtworzWizyte
//
this->NapisUtworzWizyte->Font = (gcnew System::Drawing::Font(L"Comic Sans MS", 18,
    static_cast<System::Byte>(238)));
this->NapisUtworzWizyte->ForeColor = System::Drawing::Color::White;
this->NapisUtworzWizyte->Location = System::Drawing::Point(0, 0);
this->NapisUtworzWizyte->Name = L"NapisUtworzWizyte";
this->NapisUtworzWizyte->Size = System::Drawing::Size(608, 73);
this->NapisUtworzWizyte->TabIndex = 0;
this->NapisUtworzWizyte->Text = L"Utwórz wizytę dla pacjenta";
this->NapisUtworzWizyte-> TextAlign = System::Drawing::ContentAlignment::MiddleCenter;
this->NapisUtworzWizyte->Click += gcnew System::EventHandler(this, &AdminCreateVisit);
//
// PrzyciskPowrot
//
this->PrzyciskPowrot->BackColor = System::Drawing::SystemColors::Menu;
this->PrzyciskPowrot->Font = (gcnew System::Drawing::Font(L"Comic Sans MS", 7.8F,
    static_cast<System::Byte>(238)));
this->PrzyciskPowrot->Location = System::Drawing::Point(360, 453);
this->PrzyciskPowrot->Name = L"PrzyciskPowrot";
this->PrzyciskPowrot->Size = System::Drawing::Size(225, 51);
this->PrzyciskPowrot->TabIndex = 29;
this->PrzyciskPowrot->Text = L"Powrót";
this->PrzyciskPowrot->UseVisualStyleBackColor = false;
this->PrzyciskPowrot->Click += gcnew System::EventHandler(this, &AdminCreateVisit)
```

Kolejny przykład

```
this->label1->Font = (gcnew System::Drawing::Font(L"Comic Sans MS", 18, System::Drawing::FontStyle::Bold,
    static_cast<System::Byte>(238)));
this->label1->ForeColor = System::Drawing::Color::White;
this->label1->Location = System::Drawing::Point(0, 0);
this->label1->Name = L"label1";
this->label1->Size = System::Drawing::Size(608, 73);
this->label1->TabIndex = 0;
this->label1->Text = L"Utwórz wizytę";
this->label1-> TextAlign = System::Drawing::ContentAlignment::MiddleCenter;

this->button1->BackColor = System::Drawing::SystemColors::Menu;
this->button1->Font = (gcnew System::Drawing::Font(L"Comic Sans MS", 7.8F, System::Drawing::FontStyle::Bold,
    static_cast<System::Byte>(238)));
this->button1->Location = System::Drawing::Point(360, 448);
this->button1->Name = L"button1";
this->button1->Size = System::Drawing::Size(225, 51);
this->button1->TabIndex = 29;
this->button1->Text = L"Powrót";
this->button1->UseVisualStyleBackColor = false;
this->button1->Click += gcnew System::EventHandler(this, &CreateVisit::button1_Click);
```

```
//  
// NapisUtworzWizyte  
//  
this->NapisUtworzWizyte->Font = (gcnew System::Drawing::Font(L"Comic Sans MS", 18,  
    static_cast<System::Byte>(238)));  
this->NapisUtworzWizyte->ForeColor = System::Drawing::Color::White;  
this->NapisUtworzWizyte->Location = System::Drawing::Point(0, 0);  
this->NapisUtworzWizyte->Name = L"NapisUtworzWizyte";  
this->NapisUtworzWizyte->Size = System::Drawing::Size(608, 73);  
this->NapisUtworzWizyte->TabIndex = 0;  
this->NapisUtworzWizyte->Text = L"Utwórz wizytę";  
this->NapisUtworzWizyte->ContentAlignment = System::Drawing::ContentAlignment::MiddleCenter;  
this->NapisUtworzWizyte->Click += gcnew System::EventHandler(this, &CreateVisit::OnCreateVisit);  
//  
// PrzyciskPowrot  
//  
this->PrzyciskPowrot->BackColor = System::Drawing::SystemColors::Menu;  
this->PrzyciskPowrot->Font = (gcnew System::Drawing::Font(L"Comic Sans MS", 7.8F,  
    static_cast<System::Byte>(238)));  
this->PrzyciskPowrot->Location = System::Drawing::Point(360, 448);  
this->PrzyciskPowrot->Name = L"PrzyciskPowrot";  
this->PrzyciskPowrot->Size = System::Drawing::Size(225, 51);  
this->PrzyciskPowrot->TabIndex = 29;  
this->PrzyciskPowrot->Text = L"Powrót";  
this->PrzyciskPowrot->UseVisualStyleBackColor = false;  
this->PrzyciskPowrot->Click += gcnew System::EventHandler(this, &CreateVisit::OnCreateVisit);
```

16. Przykład wzorca projektowego, który mógłby zostać wykorzystany podczas tworzenia projektu.

Jako przykład wzorca wybrano wzorzec budowniczy, którego celem jest rozdzielenie sposobu tworzenia obiektów od ich reprezentacji. Cechy:

- proces tworzenia obiektu podzielony jest na kilka mniejszych etapów a każdy z tych etapów może być implementowany na wiele sposobów,
- stosowany jest w konstrukcji obiektów złożonych, których konfiguracja i inicjalizacja jest procesem wieloetapowym,
- tworzy drobną część skomplikowanego produktu za każdym swoim wywołaniem jednocześnie kontrolując stan wykonanej pracy.

Przyczyny wyboru podanego wzorca:

- 1) klient otrzymuje produkt po zakończeniu jego pracy, a nie bezzwłocznie, dlatego nie trzeba żyć w ciągłym stresie z myślą o rutynowej kontroli kodu,
- 2) proces tworzenia programu podzielony jest na kilka mniejszych etapów i może ulegać zmianom w każdym momencie,
- 3) można odkładać niektóre etapy lub wykonywać je rekursywnie,,
- 4) kod może być implementowany na wiele sposobów.

W projekcie można by było zastosować wzorzec budowniczy w klasie *CreateVisit*. Posiada on metodę *Utwórz()*, w której zachodzi cały proces umawiania wizyty. Podzielono metodę na mniejsze części bogate w podmetody stosując wzorzec budowniczy. Stwierdzono jednak, aby nie implementować tego rozwiązania wewnątrz projektu. Niżej przedstawiony jest przykład użycia tego wzorca na podstawie napisanego kodu.

Metoda *Utwórz()* przed zastosowaniem wzorca budowniczego:

```
private: System::Void Utwórz(System::Object^ sender, System::EventArgs^ e) {  
    String^ date = this->PoleData->Text;  
    String^ pesel = this->NapisTwojPesel->Text;  
    if (date->Length == 0)  
    {  
        MessageBox::Show("Błąd w rejestracji. Uzupełnij puste pole.", "DentApp", MessageBoxButtons::OK);  
        return;  
    }  
    try  
    {  
        String^ connectionString = "Data Source=.;Initial Catalog=Dentist;Integrated Security=True";  
        SqlConnection sqlConnectionString(connectionString);  
        sqlConnectionString.Open();  
  
        //SELECT date FROM Visits WHERE date != NULL;  
  
        String^ sqlQuery = "UPDATE Visits " +  
            "SET pesel=@pesel " +  
            "WHERE date=@date";  
  
        SqlCommand command(sqlQuery, % sqlConnectionString);  
        command.Parameters->AddWithValue("@pesel", pesel);  
        command.Parameters->AddWithValue("@date", date);  
        command.ExecuteNonQuery();  
  
        MessageBox::Show("Pomyślnie zarejestrowano wizytę.", "DentApp", MessageBoxButtons::OK);  
        this->Close();  
    }  
    catch (Exception^ ex)  
    {  
        MessageBox::Show("Błąd w rejestracji. Wpisz poprawne dane.", "DentApp", MessageBoxButtons::OK);  
    }  
}
```

Metoda *Utworz()* po zastosowaniu wzorca budowniczego:

```
private: System::Void Utworz(System::Object^ sender, System::EventArgs^ e) {
    String^ date;
    String^ pesel;
    PrzypiszDane(date, pesel);
    SprawdzDate(date);
    try
    {
        WczytajDaneSQL(date, pesel);
        KomunikatAkceptacji();
    }
    catch (Exception^ ex)
    {
        KomunikatBledu();
    }
}
```

```
private: System::Void PrzypiszDane(String^ date, String^ pesel) {
    String^ date = this->PoleData->Text;
    String^ pesel = this->NapisTwojPesel->Text;
}

private: System::Void SprawdzDate(String^ date) {
    if (date->Length == 0)
    {
        MessageBox::Show("Błąd w rejestracji. Uzupełnij puste pole.", "DentApp", MessageBoxButtons::OK);
        return;
    }
}

private: System::Void WczytajDaneSQL(String^ date, String^ pesel) {
    String^ connectionString = "Data Source=.;Initial Catalog=Dentist;Integrated Security=True";
    SqlConnection sqlConnectionString(connectionString);
    sqlConnectionString.Open();

    String^ sqlQuery = "UPDATE Visits " +
        "SET pesel=@pesel " +
        "WHERE date=@date";

    SqlCommand command(sqlQuery, % sqlConnectionString);
    command.Parameters->AddWithValue("@pesel", pesel);
    command.Parameters->AddWithValue("@date", date);
    command.ExecuteNonQuery();
}

private: System::Void KomunikatAkceptacji() {
    MessageBox::Show("Pomyślnie zarejestrowano wizytę.", "DentApp", MessageBoxButtons::OK);
    this->Close();
}

private: System::Void KomunikatBledu() {
    MessageBox::Show("Błąd w rejestracji. Wpisz poprawne dane.", "DentApp", MessageBoxButtons::OK);
}

✔ Nie znaleziono żadnych problemów
```

17. Instrukcja użytkowania programu.

Wstęp

Program komputerowy DentApp umożliwia pacjentom zdalną rejestrację wizyty do gabinetu stomatologicznego, jak i obsługę całego systemu gabinetu stomatologicznego w przypadku pracownika gabinetu. System jest bardzo sprawny oraz czytelny, tak aby osoba w każdym przedziale wiekowym bez problemu mogła z niego korzystać. Niniejsza instrukcja użytkowania skierowana jest do pacjentów oraz pracowników gabinetu stomatologicznego.

Rejestracja konta oraz logowanie

Tuż po otwarciu aplikacji DentApp wyświetla się okno logowania. Aby korzystać z programu użytkownik musi posiadać konto. Jeżeli użytkownik nie posiada założonego konta należy wejść w zakładkę *Zarejestruj się*. Następnie należy wypełnić podane pola swoimi danymi. W przypadku nieprawidłowych danych lub zajętych parametrów wyskoczy komunikat z błędem. Aby utworzyć konto należy te dane poprawić. Po poprawieniu danych należy wejść w okno *Zarejestruj*. Po założeniu konta bez problemu można zalogować się do systemu podanymi danymi klikając w okno *Zaloguj*. Użytkownik może się wylogować ze swojego konta klikając w zakładkę *Wyloguj* w menu głównym. Aplikacje można zamknąć w każdym momencie klikając w krzyżyk znajdujący się u góry po prawej stronie

Interfejs pacjenta

Po zalogowaniu się jako pacjent system wyświetla stronę główną z danymi gabinetu. Po lewej stronie użytkownik ma do wyboru 4 zakładki. Aby otworzyć poszczególną zakładkę należy w nią kliknąć lewym przyciskiem myszy. Z każdej zakładki można wyjść klikając przycisk Powrót.

Rejestracja wizyty

Po wybraniu opcji *Umów wizytę* system wyświetla tabelę z dostępnymi wizytami. Aby utworzyć wizytę w podane pole należy poprawnie wpisać datę wyświetlana wyżej w tabeli. Format zapisu daty podany jest obok pola. Po wpisaniu daty należy kliknąć opcję *Utwórz*. Jeżeli wizyta została zarejestrowana poprawnie system wyświetla komunikat o prawidłowej rejestracji oraz powraca do głównego menu. W przypadku wpisania nieprawidłowej daty wyskoczy komunikat z błędem, który należy poprawić.

Anulowanie wizyty

Aby anulować wizytę należy wejść w zakładkę *Moje wizyty*. System wyświetla tabelę z umówionymi wizytami. W podane pole należy wpisać id wizyty wyświetlane wyżej w tabeli. Po wpisaniu id należy kliknąć opcję *Odwołaj wizytę*. Jeżeli wizyta została poprawnie anulowana system wyświetla komunikat o usunięciu wizyty oraz powraca do głównego menu. W przypadku wpisania nieprawidłowego id wyskoczy komunikat z błędem, który należy poprawić.

Przegląd swoich wizyt

Po wybraniu opcji *Moje wizyty* w tabeli zostaną wyświetlane wszystkie wizyty pacjenta.

Edycja danych konta

Aby edytować dane swojego konta należy wejść w zakładkę *Mój profil*. Pola możliwe do edycji wyświetlane są w ramkach. Po poprawieniu swoich danych należy je zapisać klikając w pole *Zapisz*. W przypadku wpisania nieprawidłowych danych wyskoczy komunikat z błędem, który należy poprawić.

Interfejs pracownika

Po zalogowaniu się jako pracownik system wyświetla stronę główną z danymi gabinetu. Po lewej stronie użytkownik ma do wyboru 6 zakładek. Aby otworzyć poszczególną zakładkę należy w nią kliknąć lewym przyciskiem myszy. Z każdej zakładki można wyjść klikając przycisk Powrót.

Rejestracja wizyty

Po wybraniu opcji *Umów wizytę* system wyświetla tabelę z dostępnymi wizytami. Aby utworzyć wizytę w podane pola należy wpisać pesel pacjenta oraz wolną datę wyświetlana wyżej w tabeli. Format zapisu daty podany jest obok pola. Po wpisaniu daty należy kliknąć opcję *Utwórz*. Jeżeli wizyta została zarejestrowana poprawnie system wyświetla komunikat o prawidłowej rejestracji oraz powraca do głównego menu. W przypadku wpisania nieprawidłowej daty lub peselu wyskoczy komunikat z błędem, który należy poprawić.

Anulowanie wizyty

Aby anulować wizytę należy wejść w zakładkę *Wizyty*. System wyświetla tabelę z umówionymi wizytami. W podane pole należy wpisać id wizyty wyświetcone wyżej w tabeli. Po wpisaniu id należy kliknąć opcję *Odwołaj wizytę*. Jeżeli wizyta została anulowana poprawnie system wyświetla komunikat o usunięciu wizyty oraz powraca do głównego menu. W przypadku wpisania nieprawidłowego id wyskoczy komunikat z błędem, który należy poprawić.

Przegląd wizyt

Po wybraniu opcji *Wizyty* w tabeli zostaną wyświetlane wszystkie przeszłe oraz zbliżające się wizyty.

Edycja danych konta

Aby edytować dane swojego konta należy wejść w zakładkę *Mój profil*. Pola możliwe do poprawienia wyświetlane są w ramkach. Po poprawieniu swoich danych należy je zapisać klikając w pole *Zapisz*. W przypadku wpisania nieprawidłowych danych wyskoczy komunikat z błędem, który należy poprawić.

Przegląd kartoteki pacjenta

Aby wyświetlić dane pacjentów należy wejść w zakładkę *Pacjenci*. W wyświetlonej tabeli można znaleźć szukanego pacjenta wraz z jego danymi selekcjonując daną kategorię rosnąco lub malejąco. Aby wyświetlić wszystkie wizyty pacjenta należy wpisać w podane pole jego pesel oraz nacisnąć przycisk *Szukaj*.

Wyświetlenie stanu magazynu

Po wybraniu zakładki *Magazyn* w tabeli zostaną wyświetlane wszystkie dostępne produkty wraz z ilością oraz ceną.

Dodawanie/usuwanie ilości produktu z magazynu

Aby dodać lub usunąć podaną ilość produktu należy wejść w zakładkę *Magazyn*. W podane pola należy wpisać nazwę produktu oraz ilość do dodania/usunięcia. Po wpisaniu parametrów należy kliknąć opcję *Dodaj/Usuń*. Jeżeli produkty zostały poprawnie dodane/usunięte z magazynu system wyświetla komunikat o prawidłowej operacji oraz powraca do głównego menu. W przypadku wpisania nieprawidłowych parametrów wyskoczy komunikat z błędem, który należy poprawić.

Dodawanie nowego produktu/usuwanie z magazynu

Aby dodać nowy produkt lub stale go usunąć z magazynu należy wejść w zakładkę *Magazyn*. Następnie wejść w *Edytuj bazę produktów*. W przypadku dodawania produktu w podane pola należy wpisać nazwę produktu oraz jego nową cenę. Po wpisaniu parametrów należy kliknąć opcję *Dodaj*. W przypadku usuwania produktu z magazynu w podane pola należy wpisać nazwę produktu. Podanie ceny nie jest konieczne. Po wpisaniu parametrów należy kliknąć opcję *Usuń*. Jeżeli produkty zostały poprawnie dodane/usunięte z magazynu system wyświetla komunikat o prawidłowej operacji oraz powraca do głównego menu. W przypadku wpisania nieprawidłowych parametrów wyskoczy komunikat z błędem, który należy poprawić.

18. Podsumowanie.

Opis prac:

Pracę nad projektem rozpoczęto już na początku października 2022 roku. Do jego realizacji przyjęto model kaskadowy i rozpoczęto pracę nad planowaniem oraz analizą tego oprogramowania. Już na początku zakładano, że projekt zostanie oddany w pierwszym terminie deadline'u. Szczególnie dbano o to, aby regularnie utrzymywać kontakt w sprawie projektu. Z tygodnia na tydzień do dokumentacji załączano nowy diagram poznany wcześniej na zajęciach. Kreowano je na takie, które odpowiadały wizji projektujących oprogramowanie. Tuż po wykonaniu diagramu klas, kiedy głębiej poznano funkcjonowanie aplikacji zabrano się za pisanie programu. Pisanie kodu podzielone zostało na trzy osoby i regularnie komunikowano się w sprawie jego funkcjonalności. Po zakończeniu pisania oprogramowania podjęto kroki w celu napisania dokumentacji programu. Pisanie jej również zostało podzielone między trzema osobami. Wcześniej zrobione diagramy zostały przez nas edytowane i wklejone w podaną dokumentację.

Problemy napotkane podczas realizacji projektu:

Podczas realizacji projektu pierwszym problemem piszących oprogramowanie było jego realizacja, ponieważ z początku nie wiedziano jak się do takiego projektu zabrać z powodu pierwszego kontaktu z pisaniem aplikacji okienkowej. Konieczne również było poznanie oraz nauczenie się nowego framework'a, gdyż nikomu nie był wcześniej znany. Kolejną trudnością było nauczenie się w krótkim czasie korzystania z bazy danych SQL, ponieważ był to pierwsze zetknięcie się z pracą nad danymi. W szybkim tempie poznano jego funkcje. Podane w programie elementy bez problemu trafiają do bazy danych i można na nich swobodnie pracować. Następną komplikacją było tworzenie diagramów. Niektóre z typów diagramów nie były dla projektujących czytelne oraz zrozumiałe, lecz po spędzeniu nad nimi dłuższego czasu bez zawiązania sprostano wymaganiom

Cechy oraz funkcjonalności wyróżniające opracowane rozwiązanie:

- niezawodny - bezbłędnie wykonuje zadane mu funkcje,
- przydatny - oprogramowanie bez problemu może być użyte jako system gabinetu stomatologicznego,
- z oprogramowania może korzystać pacjent jak i stomatolog potrzebujący system do przechowywania danych pacjentów,
- modyfikowalny - w każdym momencie możliwa jest opcja ulepszania kodu,
- w przypadku stomatologa łatwy dostęp do danych pacjentów oraz możliwość ich modyfikacji,
- duży zasób pamięci - bazy danych mogą zapisać tysiące informacji,
- szybkie działanie - program nie zawiesza się bez powodu,
- zrozumiałe - prosta budowa programu.