Złożenie pracy online:
2018-09-13 09:34:17
Kod pracy:
1898/36166/CloudA

Piotr Kożuch
(nr albumu: 22120)

Praca inżynierska

Aplikacja wspomagająca obsługę serwisową sprzętu i oprogramowania KS-APTEKA

Application supporting hardware and KS-APTEKA software service

Wydział: Wydział Nauk Społecznych i Informatyki

Kierunek: Informatyka

Specjalność: programowanie aplikacji biznesowych

Promotor: dr Krzysztof Przybycień

Pragnę serdecznie podziękować Panu dr Krzysztofowi Przybycieniowi za pomoc w realizacji projektu, konsultacje i cenne uwagi umożliwiające dokończenie zamierzonego celu.
Jednocześnie składam serdeczne podziękowania Panu mgr Arturowi Kornatce za przekazanie niezbędnej wiedzy z zakresu aplikacji internetowych i programowania w języku C# oraz impuls który pozwolił na rozpoczęcie nowej pasji w moim życiu jaką jest programowanie.

Streszczenie

Projekt oraz implementacja aplikacji umożliwiającej przesyłanie informacji o parametrach sprzętu komputerowego zainstalowanego u klienta, konfiguracji oprogramowania KS-APTEKA, oraz danych "wrażliwych" które umożliwią identyfikację i ocenę stanu sprzętu i oprogramowania w celu poprawy jakości obsługi i przewidywania ewentualnych awarii.
Analiza przesłanych informacji oraz rejestracja zleceń.
Dane przesyłane i wprowadzane są automatycznie za pomocą serwera FTP.
Praca zawiera trzy projekty: instalator, aplikacja klienta przesyłająca dane, aplikacja web wprowadzająca przesłane dane i umożliwiająca ich analizę.

Słowa kluczowe

apteka, serwis, wsparcie, pomoc techniczna, strona internetowa

Abstract

Design and implementation of an application enabling the transmission of information on the parameters of the computer equipment installed at the customer, KS-APTEKA software configuration, and " sensitive" data that will enable identification and assessment of the hardware and software condition to improve service quality and predict possible failures.
Analysis of sent information and registration of orders.
The data is sent and entered automatically using the FTP server.
The work contains three projects: the installer, the client application sending the data, the web application that enters the sent data and enables their analysis.

Keywords

pharmacy, service, support, help desk, website

Spis treści **Strona**

Wstęp

Firmy serwisujące sprzęt i oprogramowanie borykają się na co dzień z wieloma problemami informatycznymi, które w oczekiwaniu klienta powinny być jak najszybciej rozwiązywane.

Branża farmaceutyczna to specyficzny segment rynku wymagający ciągłego wspomagania procesu obsługi pacjenta poprzez system informatyczny. Apteka musi zapewnić ciągłość sprzedaży a mnogość uprawnień pacjentów, sposobów obliczania opłaty pacjenta, konieczność weryfikacji recept on-line, realizacja recept elektronicznych, sprawozdania do odpowiednich instytucji państwowych powoduje, że sprawność systemów informatycznych jest kluczowym elementem funkcjonowania apteki.

Szybkość obsługi serwisowej, przewidywanie problemów i natychmiastowa reakcja w przypadku awarii to cechy serwisu wymagane w celu utrzymania współpracy.

Klienci, właściciele i personel apteczny nie posiadają wystarczającej wiedzy na temat sprzętu komputerowego i systemu operacyjnego co często powoduje problemy w komunikacji i w efekcie zdiagnozowanie problemu pozostaje w całości po stronie serwisanta.

Uzyskanie informacji o zainstalowanym sprzęcie, urządzeniach peryferyjnych, stanie aplikacji oraz okresowe monitorowanie pozwala na szybszą i dokładniejszą diagnozę problemów.

Oto typowy scenariusz zgłoszenia awarii w aptece.(Rozmowa klient-serwisant)

Klient: Dzień dobry. Program nie działa. Klienci stoją w kolejce. Proszę jak najszybciej go naprawić.

Serwisant: Dzień dobry: Czy mógłby Pan podać trochę więcej szczegółów.
Co się stało, czy ma pan jakiś komunikat na ekranie komputera?

Klient: Nie wiem co się dzieje nie mogę wejść do programu, przecież nie jestem informatykiem.

Serwisant znając problemy występujące wcześniej u klienta, posiadający dokumentację wykonywanych wcześniej czynności serwisowych może rozpocząć diagnozowanie problemu. Brak takiej ewidencji powoduje ciągłe badania problemu od początku.

Aby zapewnić szybkie rozwiązywanie problemów istnieje potrzeba wykorzystania do tego celu specjalistycznego oprogramowania. Wprawdzie na rynku istnieją aplikacje

wspomagające pracę serwisu, jednak ich działanie sprowadza się do rejestracji zdarzeń, obsługi zleceń. Brakuje informacji o sprzęcie , o aplikacji a ich ręczne wprowadzenie zajmuje dużo czasu . Nie istnieje kompleksowe rozwiązanie które zapewniłoby zaspokojenie potrzeb firm serwisowych obsługujących sprzęt i oprogramowanie KAMSOFT.

W niniejszej pracy podjęto próbę zbudowania aplikacji gromadzącej dane przesyłane z komputerów, systemu operacyjnego i bazy danych aplikacji Kamsoft oraz umożliwiającą prowadzenie rejestru zleceń i czynności serwisowych.

Cel i zakres pracy

Celem niniejszej pracy jest zaprojektowanie oraz zaimplementowanie aplikacji umożliwiającej przesyłanie informacji o parametrach sprzętu komputerowego zainstalowanego u klienta, konfiguracji oprogramowania KS-APTEKA, oraz danych "wrażliwych" które umożliwią identyfikację i ocenę stanu sprzętu i oprogramowania w celu poprawy jakości obsługi i przewidywania ewentualnych awarii.

Największą zaletą będzie całkowicie automatyczne wprowadzanie bardzo szczegółowych danych o kliencie, komputerach, konfiguracji aplikacji oraz możliwość analizy, szczegółowego przeszukiwania wprowadzonych danych.

Zakres pracy obejmuje następujące zagadnienia:

- Przegląd literatury na temat sposobu tworzenia aplikacji internetowych
- Wybór optymalnej technologii
- Stworzenie aplikacji zgodnie z wymaganiami klienta
- Optymalizacja kodu
- Testowanie aplikacji

Rozdział 1. Wywiad z użytkownikiem

1.1 Opis użytkownika:

W czasie wywiadu z użytkownikiem sporządzono ogólny opis oczekiwanej funkcjonalności aplikacji.

"Chciałbym aby można było z apteki wysłać informację o parametrach zamontowanych komputerów , zainstalowanego systemu operacyjnego, oraz konfiguracji programu aptecznego. Nie posiadam na razie własnego serwera www ale mam dostęp do serwera FTP. W przyszłości planuję zakup takiego serwera więc aplikacja powinna umożliwiać szybkie przeniesienie danych na serwer www.
Aktualnie aplikacja będzie użytkowana przez serwisanta lokalnie na stanowisku pracy. Aplikacja powinna też umożliwić zapisanie zleceń wykonania jakiejś pracy serwisowej
a wykonujący zlecenie serwisant może wpisywać wykonane czynności i ile czasu na to poświęcił.
W przyszłości aplikację będziemy rozwijać więc może zaprojektujcie ją tak aby ten rozwój był możliwy."

Rozdział 2. Projekt aplikacji

2.1 Wymagania technologiczne aplikacji

Projektowana aplikacja do poprawnego działania wymaga :

- zainstalowanego serwera WWW ,
- serwera FTP,
- oraz bazy MSSQL.
- komputera z zainstalowanym systemem Microsoft Windows

2.2 Opis funkcjonalności

Aplikacja ma udostępniać kilka podstawowych funkcjonalności:

- Możliwość przesłania danych o konfiguracji komputera , konfiguracji aplikacji KS-APTEKA
- Wczytanie przesyłanych informacji
- Przeglądanie
- Zaawansowane wyszukiwanie
- Rejestrację zleceń i prac wykonywanych w czasie realizacji zlecenia

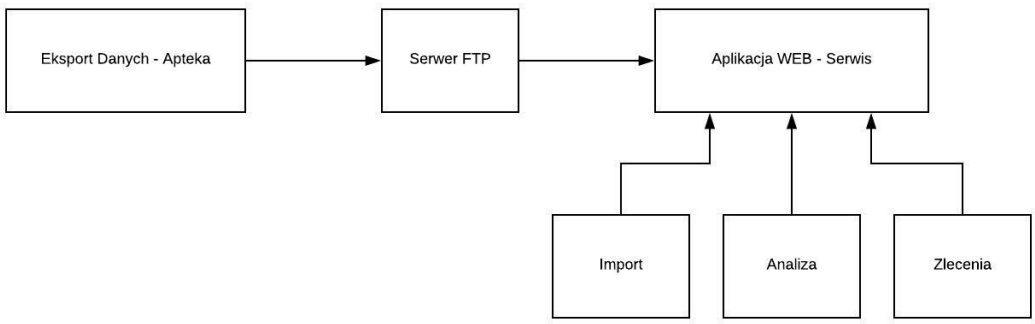
2.3 Przepływ danych

W celu zapewnienia bezpieczeństwa zagwarantowano jednokierunkowy przepływ danych.

Aplikacja po stronie klienta (Apteka) przesyła dane na serwer FTP.

Aplikacja po stronie serwisu pobiera dane i wprowadza do systemu.

Gwarantuje to przesłanie tylko tych danych które zostały zaprogramowane w aplikacji wysyłającej i ogranicza dostęp do komputera klienta.



2.4 Wybrane technologie

W celu spełnienia wymagań klienta i uwzględniając dalszy rozwój aplikacji wybrano implementację aplikacji za pomocą środowiska Microsoft ASP.NET MVC v.5,
oraz w części wysyłającej informacje od klienta Windows Application.

2.5 Dlaczego MVC

MVC To framework do budowy aplikacji internetowych w oparciu o wzorzec Model-View-Controller i technologii ASP.NET

Główne zalety MVC

MVC jest wzorcem który dzieli aplikację na trzy podstawowe warstwy:

1. Model (model)
2. View (widok)
3. Controller (kontroler)

Każda warstwa znajduję się w osobnym folderze, który zawiera odpowiednie pliki w zależności od przeznaczenia warstwy. Każda warstwa odpowiedzialna jest za wykonywanie innych funkcji.

1. Katalog Controllers zawiera kontrolery
2. Katalog Models zawiera modele danych
3. Katalog Views zawiera widoki , strony .cshtml

2.6 Dodatkowe warstwy aplikacji

W budowanej aplikacji model MVC został dodatkowo rozdrobniony na dodatkowe warstwy **Service** i **Repository**.

Warstwa Services została utworzona w celu uproszczenia kodu w kontrolerach i przeniesienia logiki biznesowej. Kontroler przeprowadza podstawową walidację danych z widoku i przenosi wszelkie operacje do Services , tam dane są przetwarzane, mapowane do obiektów i poprzez warstwę Repository zapisywane w bazie danych.

Warstwa Repository uniezależnia nas od technologii zapisu danych. Aktualnie wykorzystana zostanie baza MSSQL. Przy wykorzystaniu tej warstwy i zastosowaniu interfejsów umożliwiono w dowolną zmianę źródła danych.

Rozdział 3 Budowa aplikacji

3.1 Budowa projektów

Aplikacja wymaga wykonania czterech projektów

- instalatora aplikacji (Console Aplication)
- wspólnych klas dla wszystkich projektów (Class Library)
- Aplikacji wysyłającej dane od klienta (Windows Application)
- Aplikacji internetowej umożliwiającej wczytanie i analizę importowanych danych (Web Application)

3.2 Zastosowane technologie i frameworki wspomagające pisanie kodu:

Cały kod został napisany przy pomocy narzędzia Microsoft Visual Studio Community v 2017 przy wykorzystaniu External Tools.

Narzędzia wspomagające pracę:

- **ZenCoding** (Przyspiesza pisanie kodu html poprzez stosowanie specjalnej składni)
- **Browser Sync** (odświeża przeglądarkę internetową po zapisaniu strony)
- **Productivity Power Tools 2017** (szereg narzędzi wspomagający pisanie kodu)
- **Web Essentials 2017** (wspomaganie kodu HTML, CSS,JavaScript)
- **Web Compiler** (Automatyczna kompilacja LESS do CSS)
- **Google Docs** (Tworzenie dokumentacji)
- **Gimp** (Operacje graficzne)

3.3 Użyte frameworki

W celu przyspieszenia oraz ułatwienia programowania wykorzystano frameworki

- **MVC5** (platforma aplikacyjna do budowy aplikacji internetowych opartych na wzorcu
  Model-View-Controller)
- **Identity** (Zarządzanie tożsamością, autoryzacja, autentykacja)
- **Entity Framework** (ORM Mapowanie obiektów bazy danych do klas C#)
- **FirebirdSQL** (Biblioteka współpracy z bazami danych Firebird)
- **Autofac** (Wstrzykiwanie zależności)
- **Automapper** (Kopiowanie obiektów)
- **Bootstrap** (Framework CSS, ułatwia tworzenie interfejsu użytkownika)
- **Bootbox** (Biblioteka JavaScript - okna dialogowe)
- **JQuery** (Biblioteka ułatwiająca korzystanie z Java Script)
- **NwetonSoft** (Obsługa JSON w c#)
- **Linq** (Umożliwia łatwe zadawanie pytań na obiektach)
- **TaskScheduler** (Obsługa harmonogramu windows)
- **DataTables** (Biblioteka JavaScript - tabele)
- **Notify** (Biblioteka JavaScript - komunikaty i błędy)
-

Wszystkie użyte frameworki rozprowadzane są na licencji Open Source lub MIT
Licencje te umożliwiają ich wykorzystanie do stworzenia tej aplikacji.

3.4 Konwencje przyjęte w projektach

3.4.1 Budowa folderów na serwerze FTP

- Serwer FTP posiada folder Apteki
- W folderze tworzone są podfoldery z identyfikatorem apteki (identyfikator jest sześciocyfrowym numerem licencji programu KS-APTEKA)
- W folderze z identyfikatorem apteki jest podfolder IN w którym będą umieszczane pliki od klienta

3.4.2 Pliki przesyłane od klienta

Nazwa pliku posiada kilka informacji rozdzielonych spacjami

- Id Klienta (identyfikator licencji KS-APTEKA)
- Nr Stanowiska (numer nadawany w czasie instalacji KS-APTEKA i identyfikujący komputer w sieci)
- rok
- miesiąc
- dzień
- godzina
- minuta
- Rodzaj raportu (Pełny, Okresowy)
- Apteka (stały napis)
- Rozszerzenie pliku XML

np: “498574 1 2018-03-02 12-22 PELNY Apteka.XML”

3.4.3 Obiekty przechowujące dane

- Umieszczone w projekcie WspolneKlasy folder WymianaDanych
- Klasy publiczne zawierające properties typów prostych Int,String,Bool,Double

3.4.4 Elementy wspólne projektów

W celu zapewnienia integralności systemu został utworzony projekt WspolneKlasy typu Class Library zawierający klasy z obiektami służącymi do wymiany danych oraz obiektem Settings zawierającym statyczne ustawienia aplikacji.

Klasy wspólne projektów zostały zorganizowane w folderach:

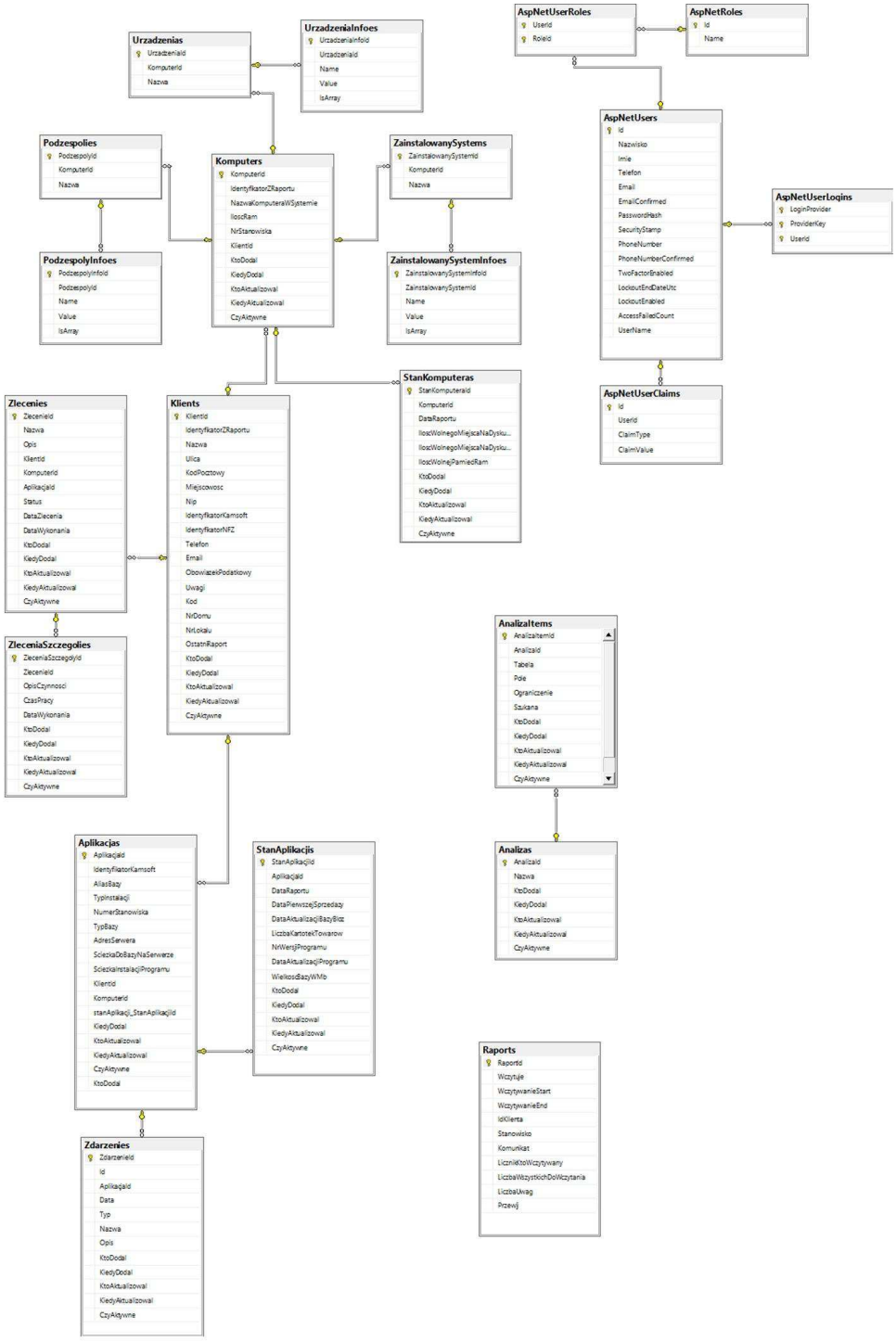
AppSettings - wspólne ustawienia projektów

Komunikacja - klasy służące do komunikacji z serwerem ftp

OperacjeXML - klasy wykonuje konwersję obiektów do XML przy pomocy plików

WymianaDanych - klasy obiektów zawierających dane przesyłane od klienta

3.5 Baza danych



Rozdział 4 Projekt Instalator

Projekt wykonany jako Console Application. Ponieważ nie potrzebujemy interakcji z użytkownikiem został wykorzystany ten typ aplikacji. W szczególnym przypadku użytkownik może zostać poproszony o podanie parametru ścieżka instalacji.

4.1 Funkcje programu InstalatorKlienta.exe

- Odszukanie programu KS-APTEKA na domyślnym dysku klienta i w domyślnej ścieżce instalacji. Program KS-APTEKA instalowany jest w lokalizacji C:\KS
  Jeśli program nie zostanie znaleziony użytkownik proszony jest o podanie lokalizacji. Należy ją wprowadzić i nacisnąć ENTER
- Pobranie z serwera FTP KlientPartner.exe wraz z wymaganymi bibliotekami dll
- Założenie na serwerze FTP folderów potrzebnych do komunikacji
- Dodanie KlientPartner.exe do Harmonogramu Windows i ustawienie uruchomienia z częstotliwością codziennie. Aby wyeliminować kolizje w wybrana zostaje losowa godzina z zakresu 9:00 - 15:00 i losowe minuty 0-59
- Uruchomienie KlientPartner.exe i wysłanie pierwszego raportu PEŁNEGO

4.2 Przykładowy kod wykonujący zaplanowanie zadania instalatora

przykład.cs

```
if (konfiguracja.FolderProgramu!="")
{
    Console.Write("Pobieram pliki z ftp :");
    PobierzPlikiZFtp(konfiguracja);
    Console.WriteLine("OK");
    Console.Write("Ustawiam PATH :");
    UstawPath(konfiguracja.FolderProgramu);
    Console.WriteLine("OK");
    Console.Write("Dodaje raportownik do harmonogramu :");
    DodajUruchomienieRaportownikaDoHarmonogramuZadanWindows(konfiguracja.FolderProgramu);
    Console.WriteLine("OK");
    Console.Write("Wysyłam raport pelny :");
    UruchomRaportownik(konfiguracja.FolderProgramu);
    Console.WriteLine("OK");
    Console.WriteLine();
    Console.Write("Nacisnij dowolny klawisz...");
    Console.Read();
}
```

Rozdział 5 Projekt KlientPartner

5.1 Sposoby uruchamiania

Aplikacja przesyłająca dane od klienta. Uruchamiana jest za pomocą wywołania w harmonogramie zadań które zostało utworzone przez instalator lub poleceniem systemowym klientpartner.exe

5.2 Parametry uruchomieniowe

W związku z tym że program KS-APTEKA może w szczególnych przypadkach zostać zainstalowany z ustawieniem niestandardowych parametrów (ścieżka instalacyjna i ścieżka do bazy danych KlientPartner.exe można uruchomić z parametrami które pozwolą odnaleźć konfigurację KS-APTEKA lub wybrać rodzaj przesłanego raportu

Lista parametrów:

-s : ścieżka instalacji programu KS-APTEKA (domyślnie C:\KS)

-r : raport do wysłania (-r PELNY, -r OKRESOWY)

-i : Identyfikator klienta (-i 495687)

-k : Nr stanowiska (-k 2)

5.3 Opis technologii

Aplikacja będzie wykonana jako Windows Application. Nie posiada żadnego interfejsu do interakcji z użytkownikiem. Może przyjmować parametry uruchomieniowe.

Taka technologia została wybrana ponieważ po stronie klienta zostaną wykonane akcje zgromadzenia i wysłania danych i niepożądana jest żadna interakcja a jakiekolwiek okno programu lub komunikat nawet w przypadku błędów nie może zakłócić pracy personelu aptecznego.

5.4 Funkcje programu KlientPartner

- wczytanie pliku konfiguracyjnego,
- sprawdzenie istnienie folderów wymiany danych . Jeśli nie istnieją - zakłada je.
- przygotowanie raportu (pełnego lub okresowego)
- przygotowanie obiektów z informacjami od klienta
- konwersja obiektów na format XML
- zapisanie obiektów do pliku
- wysłanie na serwer FTP

5.5 Szczegółowy opis działania

5.5.1 Wczytanie konfiguracji

Jeśli podano parametry uruchomieniowe ustawiany jest obiekt konfiguracji zgodnie z parametrami. W przeciwnym wypadku sprawdzane jest czy domyślny folder instalacji programu KS-APTEKA istnieje.

Jeśli odnaleziono folder tworzony jest obiekt z domyślnymi informacjami konfiguracyjnymi .

przykład.cs

```
1   public bool IsApteka { get; set; }
2   public string DyskNaKtorymJestBaza { get; set; }
3   public string SciezkaDoBazy { get; set; }
4   public string NrStanowiska { get; set; }
5   public RodzajeRaportow RodzajRaportu { get; set; }
6   public bool CzySerwer { get; set; }
7   public string IdKlienta { get; set; }
8   public string DataOstatniegoRaportu { get; set; }
9   public string serwerFTP { get; set; }
10  public string uzytkownikFTP;
11  public string hasloFTP;
12  public string FolderWymianyDanychNaFtp { get; set; }
```

5.5.2 Sprawdzenie istnienie folderów wymiany danych

Po ustawieniu konfiguracji uruchamiana jest komunikacja. Na początku sprawdzane jest istnienie folderów wymiany danych na serwerze FTP.

W celu wymiany informacji z serwerem FTP wykorzystywana jest klasa Ftp

przykład.cs

```
public class Ftp
{
public void download(string remoteFile, string localFile)
    public bool upload(string remoteFile, string localFile)
    public void delete(string deleteFile)
    public void rename(string currentFileNameAndPath, string newFileName)
    public void createDirectory(string newDirectory)
    public string[] directoryListSimple(string directory)
    public string[] directoryListDetailed(string directory)
    public bool ExistsDirectory(string directory)
}

```

Sprawdzenie i założenie folderów.

przykład.cs

```
if (!ftpKlient.ExistsDirectory(konfiguracja.FolderWymianyDanychNaFtp + "\\" + IdKlienta))
{
    ftpKlient.createDirectory(konfiguracja.FolderWymianyDanychNaFtp + "\\" + IdKlienta);
}
```

5.5.3 Przygotowanie obiektów z informacjami od klienta

Tutaj odbywa się cały proces zbierania informacji o komputerze, podzespołach, urządzeniach, systemie i oprogramowaniu.

Informacje pobierane są z kilku źródeł

1. Plik konfiguracyjny APMAN.INI
2. Baza danych Firebird WAPTEKA.FDB
3. Dysk na którym zainstalowana jest aplikacja KS-APTEKA
4. Rejestr systemu windows
5. Klasy WMI

5.5.4 Przykładowe wybrane metody

5.5.4.1 Pobranie informacji z bazy Firebird

Wymagane jest połączenie z bazą danych Firebird , odczytania informacji i zwrócenia obiektu.

Do pobierania kolejnych informacji z bazy danych firebird została stworzona metoda

przykład.cs

```
private Dictionary<string, string> PobierzRekordZBazy(string zapytanie)
{
    polaczenie.Open();
    FbCommand polecenie = new FbCommand(zapytanie, polaczenie);
    FbDataReader reader = polecenie.ExecuteReader();
    reader.Read();
    Dictionary<string, string> rekord = new Dictionary<string, string>();
    for (int i = 0; i < reader.FieldCount; i++)
    {
        rekord.Add(reader.GetName(i), reader.GetValue(i).ToString());
    }
    polaczenie.Close();
    return rekord;
}
```

5.5.4.2 Pobranie danych klienta z systemu KS-APTEKA

przykład.cs

```
public Klient PobierzDaneKlienta()
{
        Dictionary<string, string> wizytowka =
            PobierzRekordZBazy("select first 1 Id,Nazw1,Nazw2,Nazw3,Nazw4,Ulica,nrdom,nrlok,kdpcz,miast,nrnip,regon,kodkc,telef,email,awww1 from FIRM");
        //Wizytowka klienta
        Klient klient = new Klient()
        {
            Ulica = wizytowka["ULICA"],
            NrDomu = wizytowka["NRDOM"],
            NrLokalu=wizytowka["NRLOK"],
            email = wizytowka["EMAIL"],
            IdentyfikatorKamsoft = wizytowka["ID"],
            IdentyfikatorNFZ = wizytowka["KODKC"],
            KodPocztowy = wizytowka["KDPCZ"],
            Miejscowosc = wizytowka["MIAST"],
            Nazwa = wizytowka["NAZW1"] + " " + wizytowka["NAZW2"] + " " + wizytowka["NAZW3"] + " " + wizytowka["NAZW4"],
            Nip = wizytowka["NRNIP"],
            Telefon = wizytowka["TELEF"]
        };
        return klient;
}
```

5.5.4.3 Odczytanie informacji z dysku

przykład.cs

```
public double WielkoscBazyWMb()
{
    FileInfo plikBazy = new FileInfo(SciezkaDoBazy());
    double wielkoscPliku = plikBazy.Length / 1024 / 1024;
    return wielkoscPliku;
}
```

5.5.4.4 Odczytanie informacji z pliku konfiguracyjnego

przykład.cs

```
private string OdczytajUstawienie(string klucz)
{
    string LiniaZWartoscia = File.ReadAllLines(sciezkaPlikuKonfiguracyjnego)
                                 .Where(l => l.Contains(klucz)).LastOrDefault();
    LiniaZWartoscia = LiniaZWartoscia.Trim();
    LiniaZWartoscia = LiniaZWartoscia.Substring(LiniaZWartoscia.IndexOf("=") + 1);
    return LiniaZWartoscia;
}
```

5.5.4.5 Odczytanie konkretnej wartości np. ścieżka do bazy

przykład.cs

```
public string SciezkaDoBazy()
{
    return OdczytajUstawienie("DB_PATH");
}
```

5.5.4.6 Odczytanie z rejestru systemowego windows

przykład.cs

```
public string NumerStanowiska()
{
    var stanowisko =
        Registry.GetValue(@"HKEY_CURRENT_USER\SOFTWARE\KAMSOFT\KS-APW", "stanowisko", null);
    return stanowisko.ToString();
}
```

5.5.4.7 Odczytanie informacji o podzespołach i urządzeniach za pomocą klas WMI

Metoda pomocnicza przyjmująca zapytanie i zawierająca listę potrzebnymi informacjami:

przykład.cs

```
private List<List<Info>> getInfoFromQuery(string zapytanie)
        {
            ManagementObjectSearcher mos = new ManagementObjectSearcher(zapytanie);
            ManagementObjectCollection moc = mos.Get();
            List<List<Info>> wynik = new List<List<Info>>();
            List<Info> row = new List<Info>();
             foreach (var item in moc)
            {
                var kolekcja = item.Properties;
                foreach (var item2 in kolekcja)
                {
                    string value;
                    if (item2.IsArray)
                    {
                        value = String.Join(" | ", item2);
                    }
                    else
                    {
                        value = item2.Value!=null ? item2.Value.ToString() : "";
                    }
                    row.Add(new Info
                    {
                        IsArray = item2.IsArray,
                        Name=item2.Name,
                         Value=value
                    });
                }
                wynik.Add(row);
            }
            return wynik;
        }
```

5.5.4.8 Odczytanie listy podzespołów i podobna zwracająca listę urządzeń podłączonych do komputera

przykład.cs

```
private List<Urzadzenia> GetUrzadzeniaList(List<List<Info>> lista, string nazwa)
{
    List<Urzadzenia> listaUrzadzen = new List<Urzadzenia>();
    for (int i = 0; i < lista.Count(); i++)
    {
        var urzadzenie = new Urzadzenia
        {
            Nazwa = lista.Count() > 1 ? (nazwa + " " + (i + 1)) : nazwa,
            Parametry = lista[i].Select(d => new UrzadzeniaInfo
            {
                IsArray = d.IsArray,
                Name = d.Name,
                Value = d.Value
            }).ToList()
        };
        listaUrzadzen.Add(urzadzenie);
    }
    return listaUrzadzen;
}
```

5.5.4.9 Metody zwracające obiekty

Odpowiednie metody wywołują zapytanie a powyższa klasa zwraca informacje

Przykładowe wywołanie metod pobierających informacje o procesorze i płycie głównej

przykład.cs

```
private List<List<Info>> Procesory()
   => getInfoFromQuery("select CurrentClockSpeed,Name from Win32_Processor ");

private List<Info> PlytaGlowna()
   => getInfoFromQuery("select Caption,Manufacturer,Product,SerialNumber,Version  from Win32_BaseBoard ").FirstOrDefault();
```

Informacje o zainstalowanych drukarkach

przykład.cs

```
private List<List<Info>> ZainstalowaneDrukarki()
   => getInfoFromQuery("select Caption,Default,Name,Network,Shared,ShareName from Win32_Printer ");
```

Zwracane są także informacje o stanie komputera np. ilość wolnego miejsca na dysku, wielkość pliku bazy danych, ilość zainstalowanej pamięci RAM

przykład.cs

```
public double IloscWolnegoMiejscaNaDysku(string dysk)
{
    DriveInfo dyskBadany = new DriveInfo(dysk);
    var wolneMiejsceWGb = dyskBadany.AvailableFreeSpace / 1024 / 1024 / 1024;
    return (double)wolneMiejsceWGb;
}
```

Informacje o zainstalowanym systemie operacyjnym

przykład.cs

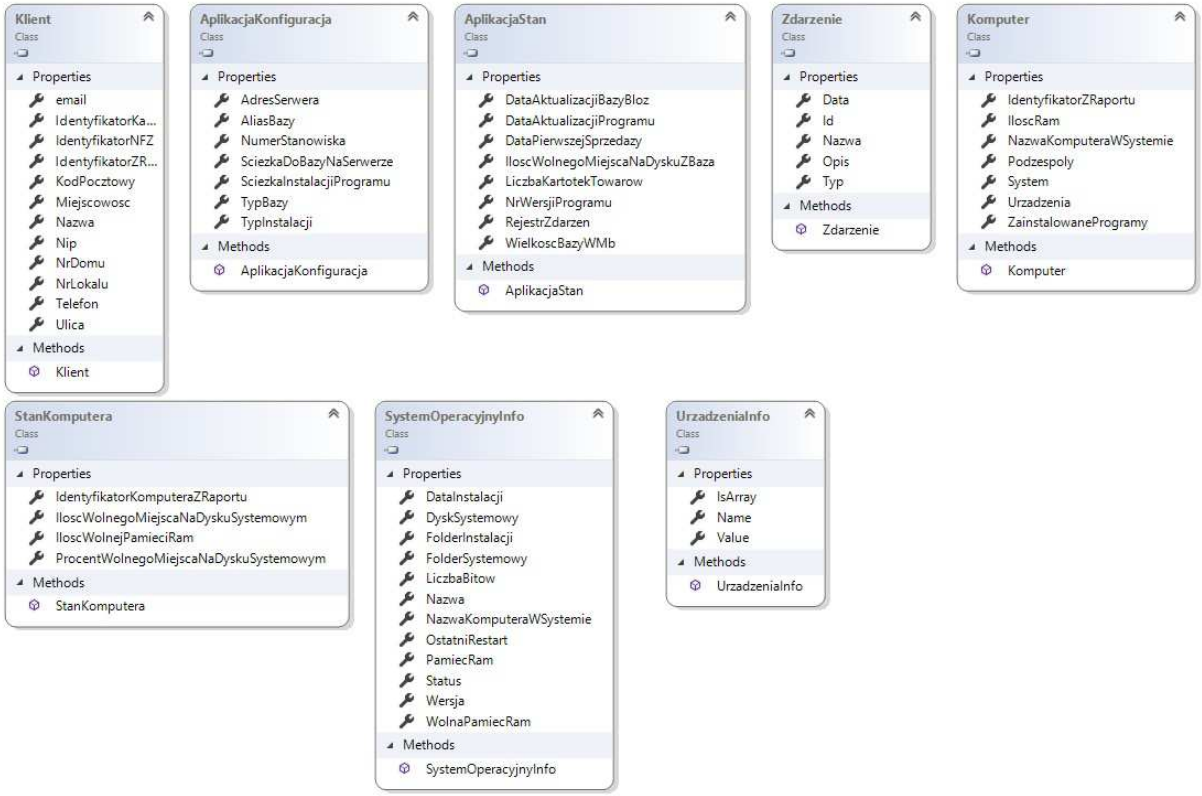
```
private List<Info> ZainstalowanySystem()
{
    string zapytanie = "select Caption," +
        "CSName," +
        "FreePhysicalMemory," +
        "InstallDate," +
        "LastBootUpTime," +
        "OSArchitecture," +
        "Status," +
        "SystemDevice," +
        "SystemDrive," +
        "TotalVisibleMemorySize," +
        "Version," +
        "WindowsDirectory" +
        " from Win32_OperatingSystem ";

    List<Info> info = getInfoFromQuery(zapytanie).FirstOrDefault();
    return info;
}
```

W zależności od ustawionego rodzaju raportu zostaje utworzony obiekt raporty odpowiedzialny za przygotowanie i wysłanie raportu

przykład.cs

```
switch (konfiguracja.RodzajRaportu)
{
    case RodzajeRaportow.PELNY:
        WyslijPlik(raporty.WygenerujRaportPelny(), "PELNY");
        break;
    case RodzajeRaportow.OKRESOWY:
        WyslijPlik(raporty.WygenerujRaportOkresowy(DateTime.Now.AddDays(-1)), "OKRESOWY");
        break;
    default:
        break;
}
```

5.5.5 Obiekty przesyłające informacje



5.5.6 Przygotowanie raportu

Wszystkie obiekty gromadzone są w obiekcie RaportPelny lub RaportOkresowy

Aktualnie dostępne są dwa rodzaje raportów

Pełny: Obejmuje

- Datę i godzinę przygotowania
- Identyfikator Klienta
- Nr stanowiska
- Informacje o aplikacji (konfiguracja, stan aplikacji)
- Informacje o komputerze (podzespoły, urządzenia, zainstalowane programy)
- Raport okresowy

Okresowy: Obejmuje

- Datę i godzinę przygotowania
- Identyfikator Klienta
- Nr stanowiska
- Informacje o stanie aplikacji
- Informacje o stanie komputera

Przygotowanie raportu (pełnego lub okresowego)

przykład.cs

```
public object WygenerujRaportPelny()
{
    RaportOkresowy raprortOkresowy = (RaportOkresowy) WygenerujRaportOkresowy(DateTime.Now.AddMonths(- LocalSettings.LiczbaMiesiecyDoRaportu ));
    InformacjeOKomputerze informacjeOKomputerze = new InformacjeOKomputerze(konfiguracja);
    RaportPelny raportPelny = new RaportPelny();
    raportPelny.IdentyfikatorKlienta = konfiguracja.IdKlienta;
    raportPelny.Stanowisko = konfiguracja.NrStanowiska;
    raportPelny.DataUtworzenia = DateTime.Now;
    if (konfiguracja.IsApteka)
    {
        InformacjeOAplikacji informacjeOAplikacji = new InformacjeOAplikacji(konfiguracja.FolderProgramu);
        raportPelny.klient = informacjeOAplikacji.PobierzDaneKlienta();
        raportPelny.klient.IdentyfikatorZRaportu = raportPelny.IdentyfikatorKlienta;
    }
    raportPelny.raportOkresowy = raprortOkresowy;
    raportPelny.komputer = informacjeOKomputerze.PobierzKonfiguracjeKomputera();
    return raportPelny;
}

```

5.5.7 Parsowanie obiektów do XML

Po skompletowaniu wszystkich informacji

- tworzony jest obiekt raportu
- parsowany do formatu XML

Do parsowania używana jest klasa OperacjeXML

przykład.cs

```
public static class OperacjeXml
{
        public static void ZapiszObjektDoPlikuXML(string sciezkaDoPliku, Object przesylanyObjekt)
    {
        XmlRootAttribute oRootAttr = new XmlRootAttribute();
        oRootAttr.IsNullable = true;
            XmlSerializer writer = new XmlSerializer(przesylanyObjekt.GetType(), oRootAttr);
        System.IO.StreamWriter file = new System.IO.StreamWriter(sciezkaDoPliku);
        writer.Serialize(file, przesylanyObjekt);
        file.Close();
    }
}

```

- zapisywany w pliku o opisanej wcześniej nazwie (konwencje)

5.5.8 Wysłanie plików na serwer FTP

Do operacji na serwerze ftp została utworzona klasa FTP.

Aplikacja sprawdza istnienie folderów ftp. Jeśli nie istnieją zakłada je i wysyłą plik z informacjami od klienta do odpowiedniego folderu

Rozdział 6 Aplikacja internetowa SerwisKa.Web

6.1 Opis rozwiązania

Aplikacja zbudowana w oparciu o wzorzec MVC ASP.NET w wersji 5.

Wprowadza automatycznie szczegółowe dane przesyłane z komputerów klientów o konfiguracji i stanie sprzętu komputerowego, podzespołach, urządzeniach, aplikacjach, konfiguracji aplikacji KS-APTEKA.

Zostały zaimplementowane także wszystkie operacje CRUD chociaż automatyczne wprowadzanie jest na tyle rozbudowane że nie wymaga się od użytkownika dodatkowej pracy manualnej.

Pozwala na synchronizację danych. Pobiera wysłane pliki z serwera FTP , odpowiednio mapuje obiekty i zapisuje do bazy danych. Jest to projekt odpowiedzialny za interakcję z użytkownikiem. Dane prezentowane są najczęściej w formie tabelarycznej z możliwością wyszukiwania, dodawania, edycji, usunięcia. Moduł analiza pozwala na dowolne przeszukiwanie informacji zgromadzonych w bazie danych oraz zapisywanie warunków wyszukiwania w celu ich późniejszego wykorzystania. Moduł Zlecenia pozwala zapamiętywać przebieg współpracy z klientem czynności, zlecenia serwisowe.

6.2 Wzorce projektowe

Ponieważ aplikacja będzie rozwijana i nie jest jasno określony kierunek rozwoju wzorzec MVC został rozszerzony o dodatkowe wzorce projektowe

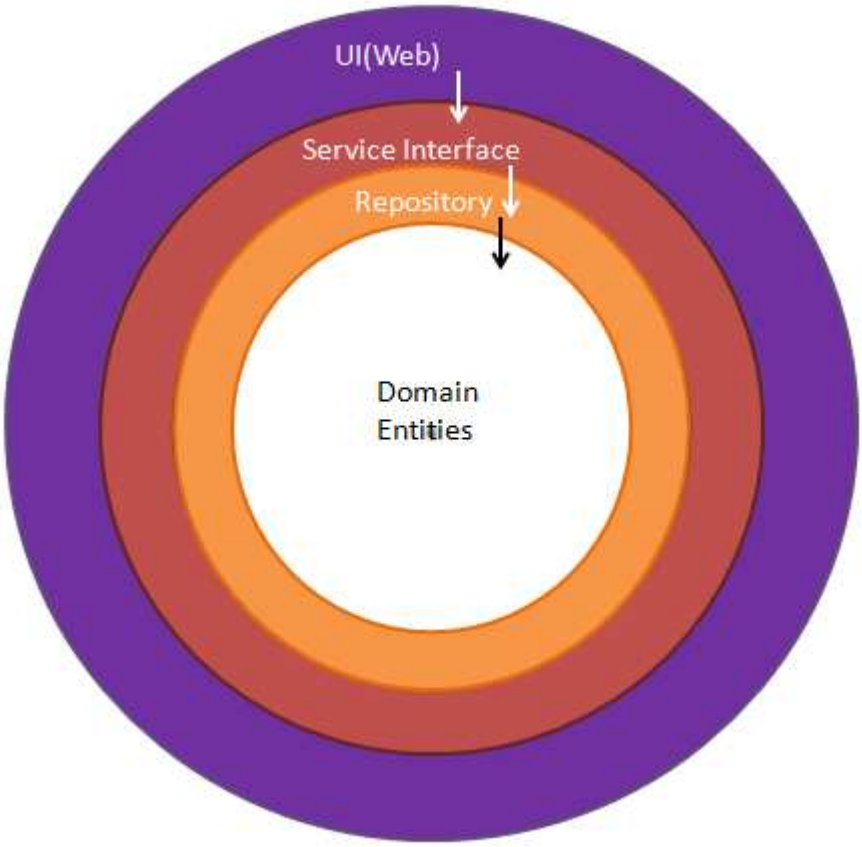
- **Dependency Injection**

- **Repozytorium**

oraz wykorzystanie klas services które przejmują rolę operowania na danych pochodzących z Controllera (Business Logic)

6.3 Ogólna budowa

W celu zapewnienia łatwej rozbudowy systemu oraz utrzymania "czystości kodu" aplikacja została zaprojektowana na podstawie założeń modelu Onion Architecture.



W centrum znajduje się model, warstwa która definiuje obiekty do przechowywania danych i ich zależności.

Dla zarządzania danymi a także uproszczenia operacji bazodanowych wykorzystany został Entity Framework.

Entity Framework to narzędzie mapowania obiektowo-relacyjnego ORM. Framework generuje obiekty biznesowe i encje zgodnie z tabelami w bazie danych. Obiekt biznesowy to encja (entity) w aplikacji wielowarstwowej, który funkcjonuje w połączeniu z dostępem do bazy danych i warstwą logiki biznesowej służącej do przesyłania danych. Entity Framework pozwala na:

1. Wykonywanie operacji CRUD (Create, Read, Update, Delete).
2. Łatwe zarządzanie relacjami typu 1 do 1, 1 do wielu oraz wiele do wielu.
3. Tworzenie relacji dziedziczenia pomiędzy różnymi encjami.

6.4 Funkcje poszczególnych warstw

Wzorzec MVC rozdziela strukturę aplikacji na poszczególne warstwy.
Wyróżniamy 3 najważniejsze. Każda z tych warstw odpowiada za inne funkcje.

1. Warstwa Model (to logika biznesowa, stanowiąca sposób reprezentacji danych w aplikacji. Zawiera klasy, pola, właściwości. Opisuje strukturę bazy danych.
2. Warstwa View (to warstwa prezentacji dla użytkownika. To style, pliki HTML, które widzi użytkownik końcowy)
3. Warstwa Controller (to warstwa odpowiedzialna za przechwytywanie żądań serwera i odpowiednio na nie reagująca. Nie zawiera żadnej logiki biznesowej

6.5 Wstrzykiwanie zależności (Autofac)

Dla zapewnienia dalszego rozwoju aplikacji oraz zapewnienie możliwości łatwego testowania wszystkie klasy utworzone w projekcie implementują Interfejsy definiujące metody wykorzystywane projekcie.

Wykorzystano w tym celu w**strzykiwanie zależności** (ang. Dependency Injection) – wzorzec projektowy i architektoniczny oprogramowania który polega na usuwaniu bezpośrednich zależności pomiędzy klasami i zastosowaniu architektury plug-in. Przekazuje gotowe, utworzone instancje obiektów udostępniających metody i właściwości obiektom, które je wykorzystują (np. jako parametry konstruktora). Wzorzec jest inną drogą w stosunku do podejścia, gdzie obiekty tworzą nowe instancje obiektów, i z których korzystają np. we własnym konstruktorze.

DI preferuje zewnętrzne tworzenie zależności pomiędzy komponentami , nad zlecaniem tworzenia zależności im samym. To wzorzec, w którym odpowiedzialność za tworzenie obiektów jest przeniesiona z obiektów do kontenera - fabryki (np. kontenera IoC). Na żądanie kodu IoC tworzy nowy obiekt, lub udostępnia obiekt z dostępnej puli, ustawiając mu powiązania z innymi obiektami (np. wstrzykiwanie poprzez konstruktor – Constructor Injection lub użycie interfejsu Interface Injection). DI jest realizacją tzw „odwrócenia sterowania” w sensie tworzenia obiektów i ich wiązania.

DI jest sposobem na osiągnięcie luźnych powiązań (ang. loose coupling).

Użycie tej techniki pozwala tworzyć łatwe do testowania obiekty. Polega ono w uproszczeniu na tworzeniu oprogramowania wg schematu: opracowujemy interfejsy (diagramy klas), a dopiero później implementujemy. Unikanie zależności od konkretnych implementacji klas współpracujących, i bazowanie wyłącznie na interfejsach daje nam możliwość skupienie się tylko na funkcjonalności wybranej klasy.
Dependency Injection realizowane jest za pomocą Autofac.

Wybrano tą bibliotekę ponieważ zapewnia przy minimalnej konfiguracji uzyskanie zakładanej funkcjonalności. Każdy interfejs posiada musi posiadać klasę implementującą - to jedyne wymaganie. Autofac jest w stanie przeanalizować kod aplikacji , odnaleźć interfejsy i klasy je implementujące oraz odpowiednio dostarczać obiekty jeśli występują żądania ich utworzenia (najczęściej w konstruktorze klasy)

Realizacja utworzenia powiązań klas i interfejsów sprowadza się do prostego kodu klasy

przykład.cs

```
public static class DependencyRegister
{
    public static void Register()
    {
        var builder = new ContainerBuilder();
        builder.RegisterControllers(typeof(MvcApplication).Assembly);
        builder.RegisterModelBinders(typeof(MvcApplication).Assembly);
        builder.RegisterModelBinderProvider();
        builder.RegisterModule<AutofacWebTypesModule>();
        builder.RegisterSource(new ViewRegistrationSource());
        builder.RegisterFilterProvider();

        //moje zaleznosci
        //builder.RegisterType<SerwiskaDbContext>().As<IDbContext>().InstancePerRequest();

        builder.RegisterAssemblyTypes(AppDomain.CurrentDomain.GetAssemblies())
        .Where(t => t.Name.EndsWith("Repository"))
        .AsImplementedInterfaces()
        .InstancePerRequest();

        builder.RegisterAssemblyTypes(AppDomain.CurrentDomain.GetAssemblies())
        .Where(t => t.Name.EndsWith("Service"))
        .AsImplementedInterfaces()
        .InstancePerRequest();

        var container = builder.Build();
        DependencyResolver.SetResolver(new AutofacDependencyResolver(container));
    }
}
```

Wszystko odbywa się automatycznie . Jeśli zajdzie potrzeba niestandardowej realizacji powiązań wystarczy dodać odpowiedni wpis do metody Register().

Aby automatyzm Autofac-a mógł działać przy powyższej konfiguracji należy stosować konwencję:

- Nazwa każdego interfejsu warstwy Repozytorium musi być zakończona Repository
- Nazwa każdego interfejsu warstwy Services musi być zakończona Service

przykład.cs

```
Rejestracja Autofac odbywa się w Global.asax
protected void Application_Start()
{
    AutoMapper.Mapper.Initialize(cfg => cfg.AddProfile<AutoMapperProfile>());
    AreaRegistration.RegisterAllAreas();
    DependencyRegister.Register();
    GlobalConfiguration.Configure(WebApiConfig.Register);
    FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
    RouteConfig.RegisterRoutes(RouteTable.Routes);
    BundleConfig.RegisterBundles(BundleTable.Bundles);
}
```

6.6 Data Transfer Objects

Aby zapewnić jak najmniejszy przepływ danych z serwera do klienta do przenoszenia danych zastosowano klasy ViewModel nazywane też **DTO** (Data Transfer Objects).
Nie użyto do tego celu klas modelu ponieważ nie wszystkie dane są potrzebne w widoku więc ograniczono nadmiarowość danych.
Nie wszystkie dane możemy także przekazywać. Np. dane użytkownika, logowania itp.
Obiekty ViewModel to klasy tworzące obiekty z danymi wykorzystywanymi przez warstwę View do wyrenderowania strony HTML.

Bardzo często obiekty ViewModel są podobne do obiektów Model (entities) .
Aby przyspieszyć proces kopiowania danych z Modelu do ViewModelu wykorzystano framework AutoMapper.

6.7 AutoMapper

AutoMapper biblioteka, która jest w stanie przepisywać wartości obiektu jednego typu, do analogicznych właściwości w innym obiekcie, innego typu. W przypadku gdy pola mają różne nazwy, istnieje możliwość konfiguracji mapowania uwzględniającego takie różnice pomiędzy obydwiema klasami. Biblioteka obsługuje także typy zagnieżdżone, dzięki czemu umożliwia mapowanie nawet bardzo złożonych obiektów.

Przepisywanie właściwości obiektów jest łatwe, lecz proces jest mozolny i nudny. Łatwo popełnić błąd tworząc ręczne mapowania. Narzędzia typu AutoMapper stosujemy gdy tworzymy aplikacje, w których rozróżniamy warstwy danych i prezentacji. Tutaj biblioteka ta daje największe możliwości. Za pomocą kilku linijek kodu, możliwe jest przepisanie bardzo złożonych obiektów.

Oto skrócona lista najważniejszych funkcjonalności:

- Automatyczne mapowanie obiektów gdy właściwości są identyczne
- Mapowanie obiektów złożonych
- Mapowanie właściwości o różnych nazwach
- Automatyczne zastąpienie null
- Możliwość konwertowania wartości na inny typ
- Wsparcie dla LINQ

Z biblioteki AutoMapper warto korzystać zawsze, gdy w naszej aplikacji transferujemy dane z podobnych obiektów, różniących się jednak typem.

Definicja mapowań klas dla AutoMappera implementowana jest w klasie AutoMapperProfile

przykład.cs

```
public AutoMapperProfile()
{
    CreateMap<Raport, RaportViewModel>();
    CreateMap<Klient, KlientViewModel>();
    CreateMap<KlientToFormViewModel, Klient>().ReverseMap();
    CreateMap<KlientToFormViewModel, KlientViewModel>().ReverseMap();
    CreateMap< WspolneKlasy.Klient, KlientBufor>().ReverseMap();
    CreateMap<Klient, WspolneKlasy.Klient>().ReverseMap();
    CreateMap<Komputer, WspolneKlasy.Komputer>().ReverseMap();
    ...
```

6.8 Warstwy systemu

6.8.1 Repository

Repository to warstwa pośrednicząca pomiędzy modelem z warstwą Services. Zadaniem tej warstwy jest wykonywanie operacji najczęściej CRUD na modelu danych. Metody klas repozytorium wywoływane są w warstwie Services która zajmuje się logiką biznesową i przetwarzaniem danych.

Każda klasa warstwy Repository dziedziczy po IBaseRepository<TEntity>

przykład.cs

```
1  public interface IBaseRepository<TEntity> where TEntity:class
2  {
3      TEntity Get(int id);
4      IEnumerable<TEntity> Find(Expression<Func<TEntity, bool>> predicate);
5  }
```

To wspólny - generyczny interfejs który zawiera metody wykorzystane w każdej klasie repozytorium. Klasa implementująca ten interfejs tworzy połączenie z bazą i definiuje operacje na bazie danych.

Zastosowanie wzorca repozytorium pozwala na dowolne składowanie danych w przyszłości. Aktualnie wykorzystywana jest baza MSSQL. Przy niewielkim nakładzie pracy możemy zmienić źródło przechowywania danych na dowolne implementując tylko na nowo interfejsy warstwy repozytorium.

Wszelkie zapytania na bazie danych są tworzone przy wykorzystaniu interfejsu **IQueryable**. Zapobiega to pobieraniu danych a tylko rozbudowuje zapytanie do bazy o kolejne warunki.

Dane pobierane są w końcowym etapie bezpośrednio przed przekazaniem do widoku

6.8.2 Services

Zadaniem tej warstwy jest otrzymywanie danych z kontrolerów, przetworzenie, analizowania, logika biznesowa i zlecenie operacji bazodanowych do repozytorium.

Dzięki zastosowaniu tej warstwy kontrolery stają się bardzo uproszczone, ich zadanie sprowadza się tylko do walidacji danych i zlecenia przetworzenia przez klasy serwisów.

Warstwa ta jest przeznaczona dla logiki biznesowej aplikacji.

6.8.3 Controller

Służy interakcji pomiędzy widokiem a danymi.

Metody w kontrolerach pobierają dane wysłane z widoku , przeprowadzają walidację jeśli jest wymagana i przekazują dane do warstwy Services. Tam dane są przetwarzane, analizowane oraz zapisywane w bazie przy wykorzystaniu warstwy Repository.

Kontrolery nie wykonują żadnych zadań związanych z logiką biznesową - to zadanie dla serwisów. Pobierają dane , walidują i przekazują do serwisów, W drugą stronę pobierają dane z serwisów i przekazują do widoków.

Mechanizm współpracy kontrolerów z widokami zaimplementowany jest w frameworku MVC .

W konstruktorach kontrolerów wstrzykiwane są obiekty klas serwisów potrzebne w procesie przetwarzania danych.

Przykład

przykład.cs

```
private readonly IKlientService _klientService;
private readonly IKomputerService _komputerService;
private readonly IResponseService _errorService;
private readonly IAplikacjaService _aplikacjaService;
private readonly IZleceniaService _zleceniaService;

public KlienciController(IKlientService klientService,
                         IKomputerService komputerService,
                         IResponseService errorService,
                         IAplikacjaService aplikacjaService,
                         IZleceniaService zleceniaService
    )
{
    _klientService = klientService;
    _komputerService = komputerService;
    _errorService = errorService;
    _aplikacjaService = aplikacjaService;
    _zleceniaService = zleceniaService;
}
```

Tworzone są zmienne prywatne które dają dostęp do metod logiki biznesowej.

W celu zabezpieczenia przed nieautoryzowanym dostępem do danych każdy kontroler opatrzony jest atrybutem [Authorize]

Zastosowanie frameworka Identity upraszcza proces rejestracji, logowania i autentykacji i autoryzacji użytkownika oraz gwarantuje maksymalne bezpieczeństwo.

Metody kontrolerów zwracają widoki renderowane przez silnik RAZOR. Często do widoków przekazywane są obiekty modeli które tworzone są w warstwie services.

W kontrolerach zastosowano odpowiednie atrybuty metod pozwalające na identyfikację operacji wykonywanych na danych:

[HttpGet] - może być pominięty służy zawsze do pobrania żądania pobrania danych do widoku

[HttpPost] - służy żadania do zapisu obiektu do bazy danych

[HttpDelete] - żądanie usunięcia

[HttpPut] - żądanie poprawy części danych

Akcje kontrolera

Przykładowa akcja kontrolera

przykład.cs

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Zapisz(KlientToFormViewModel klient)
{
    if (!ModelState.IsValid)
    {
        JsonResponseViewModel response = _errorService.GenerateJsonFromModelstateErrors(ModelState);
        return Json(new { response }, JsonRequestBehavior.AllowGet);
    }
     if (klient.KlientId==0)
    {
            try
            {
                _klientService.Add(klient);
                return new HttpStatusCodeResult(HttpStatusCode.OK);
            }
            catch
            {
                return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
            }
    }
    else
    {
        try
        {
            _klientService.Edit(klient);

            return new HttpStatusCodeResult(HttpStatusCode.OK);
        }
        catch
        {
            return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
        }
    }
}
```

6.8.4 UI

To warstwa View .Zawiera klasy odpowiedzialne za wygląd interfejsu użytkownika.

Zadaniem tej warstwy jest przetworzenie i wyrenderowanie widoku strony HTML dla klienta.

Wykorzystany został standardowy w MVC5 silnik **RAZOR**.

Razor to silnik renderujący, który w znacznym stopniu upraszcza projektowanie widoków. Posiada prostszą składnię, wymagającą mniejszej ilości kodu do uzyskania w HTML końcowych efektów.

Prezentowane strony HTML tworzone są w oparciu o wspólny wygląd zaprojektowany w klasie **_Layout.schtml**

Klasa ta jest odpowiednio przygotowana do renderingu poprzez zastosowanie skladni RAZOR.

Wykorzystywany jest mechanizm **bundli**. Służy on do grupowania potrzebnych zasobów w pakiety i używanie ich automatycznie przez silnik ASP .NET MVC podczas uruchamiania aplikacji. Wszystko dzieje się w tle. Silnik ASP .NET sam wybiera który plik zostanie w danej chwili użyty. Decyduje on również o wersji z której chce skorzystać.

Konfiguracja klasy

przykład.cs

```
public class BundleConfig
{
    public static void RegisterBundles(BundleCollection bundles)
    {
        bundles.Add(new ScriptBundle("~/bundles/lib").Include(
                    "~/Scripts/jquery-{version}.js",
                    "~/Scripts/bootstrap.min.js",
                    "~/Scripts/bootbox.min.js",
                    "~/Scripts/respond.js",
                    "~/Scripts/Datatables/datatables.min.js",
                    "~/Scripts/Notify/notify.min.js",
                    "~/Scripts/jquery-ui-1.12.1.min.js",
                    "~/Scripts/MyScripts/General.js"
                    ));

        bundles.Add(new ScriptBundle("~/bundles/jqueryvalidate").Include(
                    "~/Scripts/jquery.unobtrusive-ajax.min"));
        bundles.Add(new ScriptBundle("~/bundles/modernizr").Include(
                    "~/Scripts/modernizr-*"));

        bundles.Add(new LessBundle("~/Content/less").Include(
                    "~/Content/*.less"));

        bundles.Add(new StyleBundle("~/Content/css").Include(
                    "~/Content/bootstrap-cyborg.min.css",
                    "~/Scripts/DataTables/datatables.min.css",
                    "~/Content/themes/base/jquery-ui.min.css",
                    "~/Content/site.css"));
    }
}
```

Zdefiniowane są tam odwołania do skryptów **JavaScripts** oraz do plików stylów **css**.
Mechanizm bundli odczytuje zdefiniowane pliki , kompresuje je oraz dostarcza do pliku HTML w postaci pojedynczego pliku w maksymalnie zmniejszonej objętości. Powoduje to znaczne przyspieszenie pobierania i otwierania strony html oraz minimalizuje ilość zapytań do serwera.

Aby uprościć tworzenie plików stylów css wykorzystano także **Less**.
Less (Leaner CSS) to język arkuszy stylów. Jest zagnieżdżonym metajęzykiem - składnia CSS jest również poprawnym kodem Less. Różnica pomiędzy Less i innymi prekompilatorami CSS polega na możliwości kompilacji w czasie rzeczywistym kodu przez przeglądarkę. Less może działać po stronie klienta, także po stronie serwera, oraz jego kod może być skompilowany wcześniej do CSS.

W widokach szeroko stosowany jest **JQuery** – lekka biblioteka programistyczna dla języka JavaScript, ułatwiająca korzystanie z JavaScriptu . Kosztem niewielkiego spadku wydajności w stosunku do profesjonalnie napisanego kodu w nie wspomaganym JavaScripcie pozwala osiągnąć interesujące efekty animacji, dodać dynamiczne zmiany strony, wykonać zapytania **AJAX**.

Skrypty JavaScript dodawane są w sekcji @scripts bezpośrednio w kodzie HTML lub wstrzykiwane z plików zgromadzonych w folderze Scripts/MyScripts

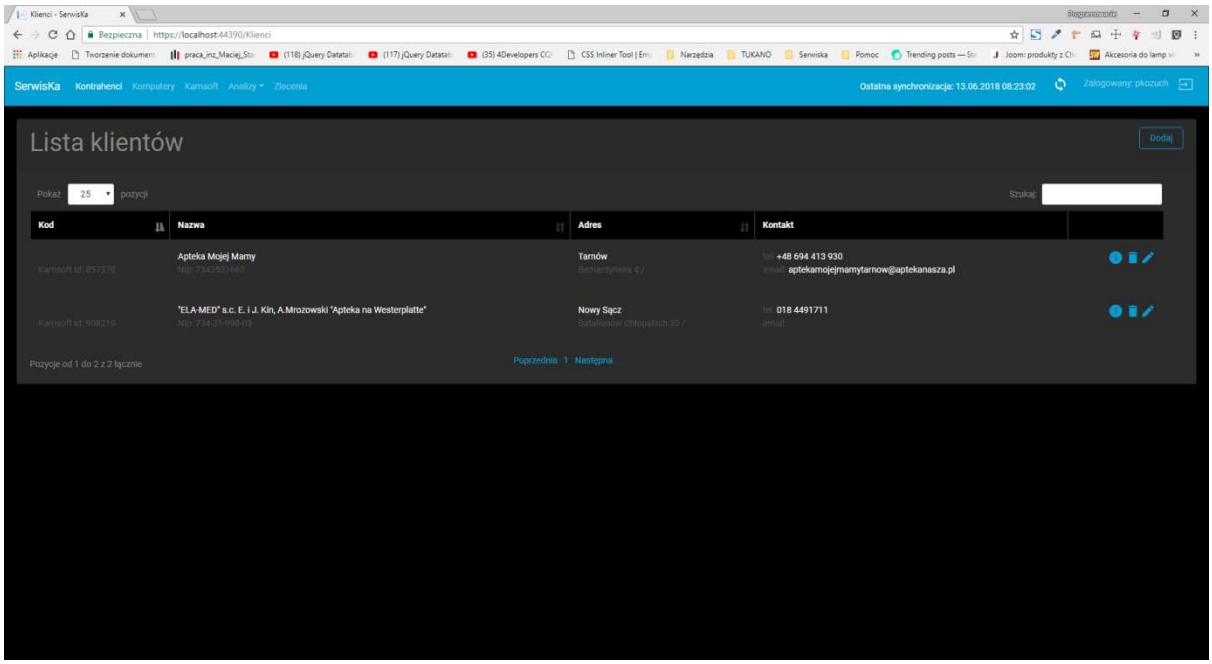
Większość danych prezentowana jest w aplikacji w formie tabelarycznej.
W celu zapewnienia wymaganej interakcji użytkownika z aplikacją i uproszczenia kodu została wykorzystana biblioteka **dataTables**.js
Pozwala ona na dowolne manipulowanie danymi (w naszym przypadku przesyłanymi za pomocą technologii AJAX oraz modelu **JSON**) przy niewielkim rozmiarze potrzebnego kodu. Tabele zapewniają wybór ilości wyświetlanych pozycji, przeszukiwanie, stronicowanie oraz wybór formularzy potrzebnych do wprowadzenia i modyfikacji danych.
W aplikacji w miejscach gdzie część strony powtarza się np. dane klienta stosowane wykorzystano widoki częściowe (**PartialView**). Eliminuje to powtarzalność kodu. Do partialView przekazywany jest model , tam renderowany i wklejony jako część strony. Widoki te wykorzystano także przy budowaniu formularzy. Mechanizm AJAX zapytuje serwer o formularz. Serwer przesyła partialView który jest wstrzykiwany do okna modalnego. W efekcie nie jest przesyłana cała strona a tylko niewielki fragment.

Wiele metod kontrolerów nie zwraca widoków. Zwracają dane obiektów zmapowane do standardu JSON.

JSON, JavaScript Object Notation – format wymiany danych komputerowych. Jest formatem tekstowym, bazującym na składni języka JavaScript. Typ MIME formatu JSON to application/json. JSON jest jednym z nieformalnych sposobów przekazywania danych wykorzystywanych do aplikacji opartych o technologię AJAX. Najczęściej dane w formacie JSON są pobierane z serwera w formacie tekst z wykorzystaniiem obiektu XMLHttpRequest języka JavaScript, i następnie przekształcane w obiekt. Tekst powinien być kodowany UTF-8, który jest w JSON domyślny.

Wszystkie operacje Dodawania, usuwania, modyfikacji, komunikaty błędów odbywają się za pomocą technologii AJAX i wymiany danych pomiędzy stroną www a serwerem za pomocą formatu JSON.

Przykładowy wygląd okna aplikacji:



Większość widoków została wykonane z zastosowaniem zasad SPA

Single Page Application (SPA). Technologia nie jest nowa, ale dopiero teraz zaczyna być szerzej używana. Przewiduje się że SPA stanie się kolejną rewolucją w realizacji stron internetowych.

SPA (Single Page Application) to aplikacja internetowa lub strona internetowa, która jest wczytywana w całości. Cały kod (HTML, CSS, JavaScript) przesyłany jest na początku lub dodawany dynamicznie w częściach, zwykle w odpowiedzi na działania użytkownika.

Zastosowanie takiego podejścia umożliwia zminimalizowanie odświeżeń strony, zapytań do serwera, i w efekcie prowadzi do szybszego działania. Przypomina to działanie aplikacji zainstalowanych na komputerze.

Wygląd strony zbudowany jest w oparciu o **Bootstrap** v 3.7

oraz theme **Bootswach Cyborg**

Bootstrap – to framework CSS, rozwijany przez Twittera, wydawany na licencji MIT Zawiera zestaw narzędzi ułatwiających tworzenie interfejsu graficznego stron HTML oraz aplikacji internetowych. Bazuje głównie na gotowych rozwiązaniach HTML oraz CSS. Pliki CSS kompilowane są z plików Less. Bootstrap może być stosowany m.in. do stylizacji tekstów, formularzy, przycisków, wykresów, nawigacji i innych komponentów wyświetlanych na stronie www. Framework korzysta również z języka JavaScript.

6.8.5 Model

W tej warstwie zgromadzono wszystkie klasy mapujące obiekty z bazy danych przy wykorzystaniu Entity framework, klasy DTO, oraz klasy pomocnicze Enum, StaticString.
Każda z klas posiada properties które przechowują dane oraz atrybuty pozwalające na dodatkową funkcjonalność np. walidację.

6.8.6 Scripts

W tej warstwie przechowywane są wszystkie skrypty i frameworki w języku JavaScript

6.9 Zakres prezentowanych danych

W aplikacji operujemy na danych Klientów, Komputerów, Aplikacji KS-APTEKA, oraz Zlecenia

6.10 Operacje na danych

6.10.1 Synchronizacja

Synchronizacja rozpoczynamy klikając na ikonę w górnej części strony.

Asynchronicznie wywołana zostaje akcja kontrolera która wczytuje dane i zapisuje do bazy.
Po rozpoczęciu synchronizacji w tabeli raporty wpisywane są informacje o przebiegu operacji, godzina rozpoczęcia, zakończenia, klient którego dane aktualnie są wczytywane. Zastosowanie funkcji interwałowej JavaScript pozwala na ciągłe informowanie użytkownika o stanie synchronizacji. Co 1 sekundę odczytywana jest informacja i prezentowana użytkownikowi. Wszelkie działania wykonywane są asynchronicznie więc użytkownik może kontynuować pracę i jednocześnie synchronizować dane.

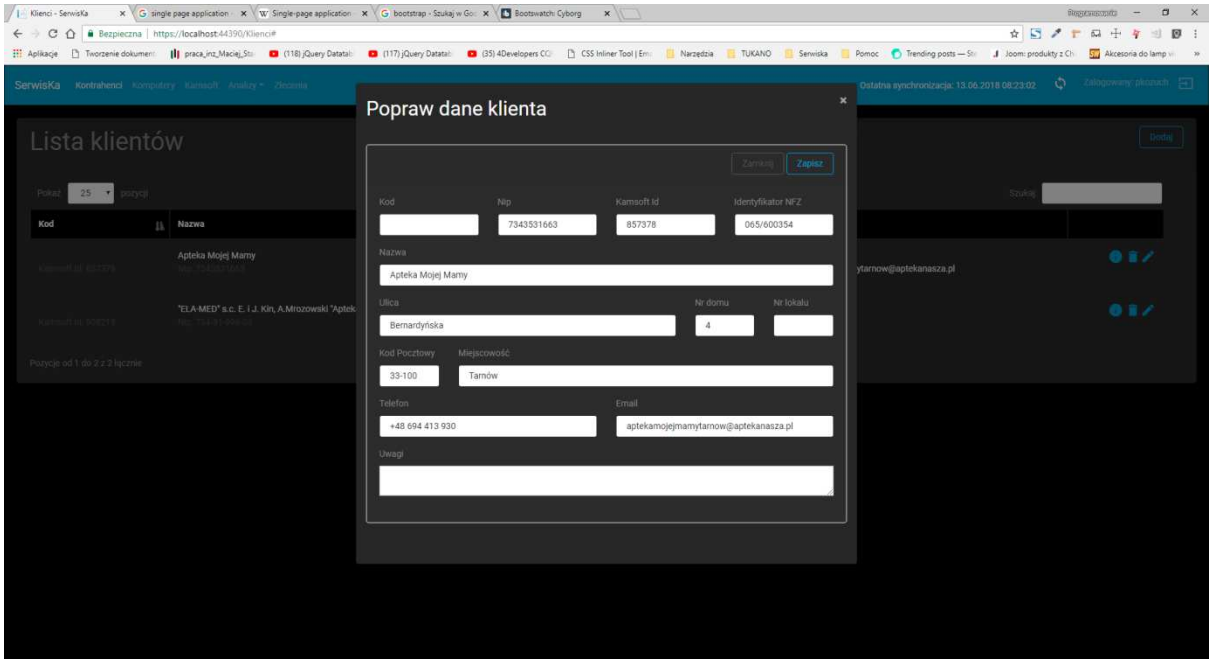
Funkcja wywołująca synchronizację i pobierająca informacje

przykład.cs

```
$(document).ready(function () {
    var interval;
    var statusWczytywania = function () {
        interval = setInterval(function () {
            $.ajax({
                url: "/Raporty/informacje"
            }).done(function (data) {
                console.log(data);
                $("#aktual-info").html(data.Komunikat)
                if (data.Wczytuje == false) {
                    $("#aktual-info").html("Ostatna synchronizacja: " + data.WczytywanieEnd);
                    clearInterval(interval);
                    console.log(data.Wczytuje);
                }
            }).fail(function () {
                $("#aktual-info").html("Wystąpił błąd");
            })

        }, 1000);
    }

    statusWczytywania();

    $("#btn-synchronizuj").on("click", function () {
            statusWczytywania();

        $.ajax({
            url: "/Raporty/wczytaj"
        });
    });
});
```

6.10.2 Formularze

Formularze dodawania i modyfikacji danych wykorzystują okna modalne bootstrap

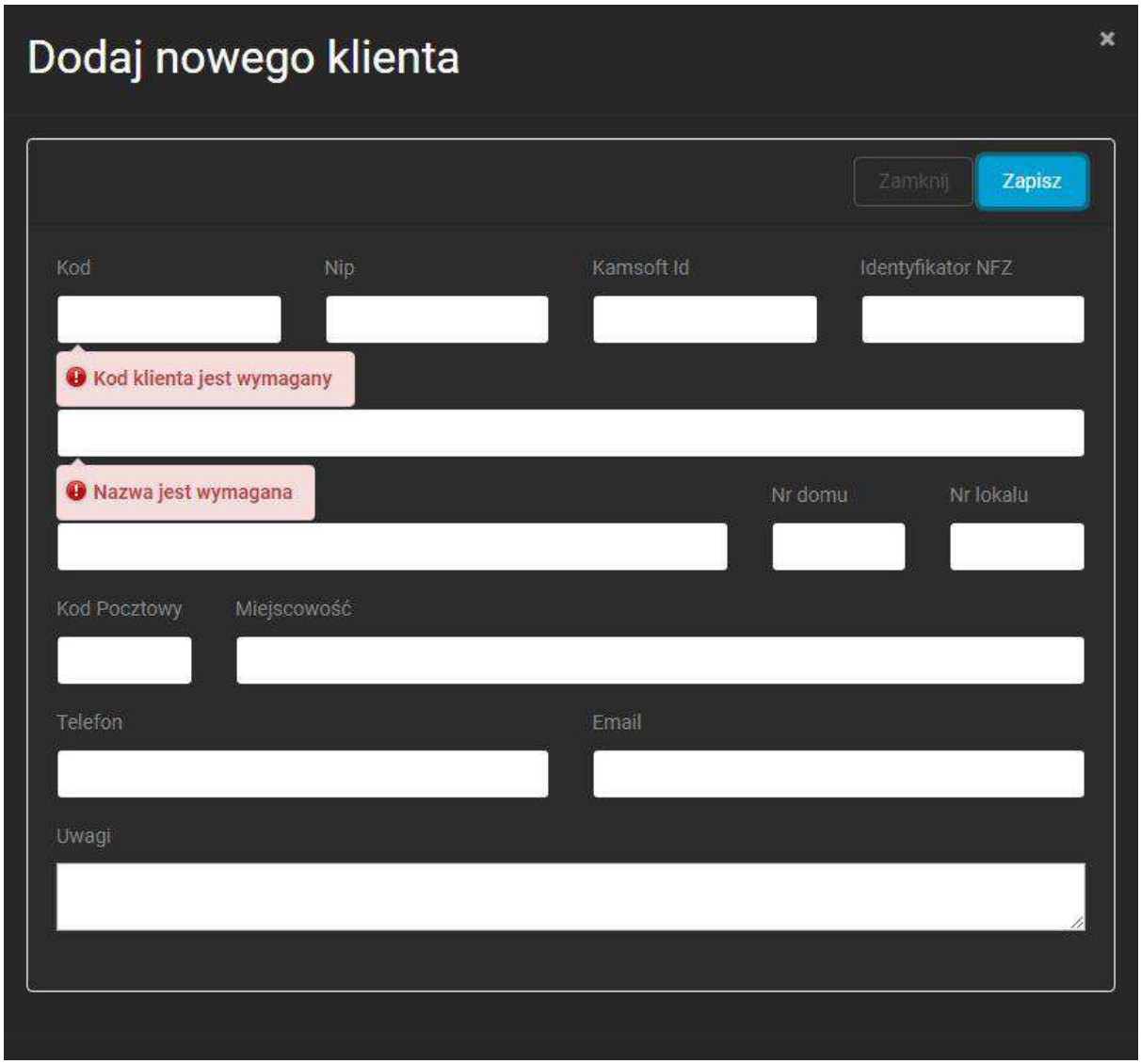


6.10.3 Mechanizm zapisu i obsługa błędów

Wszelkie komunikaty walidacyjne wyświetlane są pod polami wprowadzania danych.

Mechanizm wysyłania danych odbywa się za pomocą technologii AJAX.

Kontroler przyjmuje dane z widoku, przeprowadza walidację i jeśli dane są poprawne zleca klasie service wykonanie odpowiedniej akcji. Jeśli dane nie spełniają wymogów które deklarowane są za pomocą atrybutów Anotation w klasach ViewModel, kontroler zwraca obiekt JSON z błędami. Błędy są pobierane w widoku, interpretowane i wyświetlane w formularzu. Do wyświetlania informacji zwracanych z kontrolera wykorzystano bibliotekę JavaScript **Notify**



Przykładowe komunikaty błędów

6.10.4 Informacje

Okna informacyjne wykorzystują skrypt Notify

Przykładowa informacja

6.10.5 JavaScript AJAX

Obsługa w widoku za pomocą JavaScript / JQUERY

przykład.cs

```
function saveCustomer(id) {
    var customer = {
        KlientId: id,
        Kod: $("#Kod").val(),
        Nazwa: $("#Nazwa").val(),
        Ulica: $("#Ulica").val(),
        NrDomu: $("#NrDomu").val(),
        NrLokalu: $("#NrLokalu").val(),
        KodPocztowy: $("#KodPocztowy").val(),
        Miejscowosc: $("#Miejscowosc").val(),
        Nip: $("#Nip").val(),
        IdentyfikatorKamsoft: $("#IdentyfikatorKamsoft").val(),
        IdentyfikatorNFZ: $("#IdentyfikatorNFZ").val(),
        Telefon: $("#Telefon").val(),
        Email: $("#Email").val(),
        Uwagi: $("#Uwagi").val(),
        __RequestVerificationToken: $("input[name=__RequestVerificationToken]").val()
    }

    $.ajax({
        url: "/klienci/zapisz",
        method: "POST",
        data: customer,
        cache: false,
        success: function (data) {
            if (data && data.response.Errors) {
                showError(data.response.Errors);
            } else {
                location.reload();
                $('#myModal').modal('hide');
                $.notify("Zapisano klienta", "info");
            }
        },
        error: function (response, status, error) {
            $.notify("Problem z zapisem klienta", "error");
        }
    });
}
```

6.10.6 Wyświetlanie list z danymi (DataTables)

Dane prezentowane użytkownikowi są najczęściej w postaci tabelarycznej.

Do obsługi wykorzystano bibliotekę **Datatables**

Wszystkie dane pobierane są z serwera za pomocą AJAX, renderowane w Javascript i prezentowane w tabeli. Datatable zapewnia obsługę wyszukiwania, stronicowania i wyświetlania danych po stronie widoku. Obsługa tych zdarzeń odbywa się w kontrolerach.

Przykładowy kod tabeli (HTML)

przykład.cs

```
<table id="komputery" class="table table-hover " data-order='[[ 1, "asc" ]]' data-page-length='25'>
    <thead>
        <tr>
            <th></th>
            <th>
                Klient
            </th>
            <th>
                Nazwa i nr stanowiska
            </th>
            <th>
                System
            </th>
            <th>
                Podzepoły
            </th>
            <th data-orderable="false">
                 
            </th>
        </tr>
    </thead>
    <tbody></tbody>
</table>
```

Oraz obsługa po stronie JavaScript JQUERY

przykład.cs

```
$(document).ready(function () {
    var dataTableUrl = '/Komputery/DataTableGet';
    var detailUrl = '/Komputery/Szczegoly';

    $('#komputery').DataTable({
        serverSide: true,
        ajax: {
            url: dataTableUrl,
            type: 'POST',
        },
        "columnDefs": [
            {
                'data': 0,
                'defaultContent': '',
                'className': 'control',
                'orderable': false,
                'targets': 0,
                "render": function (data, type, rowData) {
                    var result = '';
                    return result;
                }
            },
            {
                "targets": 1,
                "data": 1,
                'orderData': 1,
                "render": function (data, type, rowData) {
                    var result = '';
                    result += '<div >' + rowData[1] + '</div>';
                    result += '<div >' + rowData[2] + '</div>';
                    result += '<div >' + rowData[3] + '</div>';
                    result += '<div >' + rowData[4] + '</div>';
                    return result;
                }
            },
            {
                "targets": 2,
                "data": 5,
                'orderData': 2,
                "render": function (data, type, rowData) {
                    var result = '';
```

przykład.cs

```
                render : function (data, type, rowData) {
                    var result = '';
                    result += '<a href="/Komputery/Szczegoly/' + rowData[0] + '" title="Szczegóły"><i class="material-icons">info</i></a>';
                    result += '<a href="#" class="computer-edit"><i class="material-icons" data-computer-id = ' + rowData[0] + '> edit</i ></a>';
                    result += '<a href="#" class="computer-delete"><i class="material-icons" data-computer-id = ' + rowData[0] + '> delete</i></a>';
                    return result;
                }
            },
        ],
        "fnCreatedRow": function (row, data, dataIndex) {
            bindActionsDataTable(row, data);
        },
        "paginate": true,
        "bFilter": true,
        "searchable": true,
        cache: false,
        stateSave: true,
        language: {
            "processing": "Przetwarzanie...",
            "search": "Szukaj:",
            "lengthMenu": "Pokaż _MENU_ pozycji",
            "info": "Pozycje od _START_ do _END_ z _TOTAL_ łącznie",
            "infoEmpty": "Pozycji 0 z 0 dostępnych",
            "infoFiltered": "(filtrowanie spośród _MAX_ dostępnych pozycji)",
            "infoPostFix": "",
            "loadingRecords": "Wczytywanie...",
            "zeroRecords": "Nie znaleziono pasujących pozycji",
            "emptyTable": "Brak danych",
            "paginate": {
                "first": "Pierwsza",
                "previous": "Poprzednia",
                "next": "Następna",
                "last": "Ostatnia"
            },
            "aria": {
                "sortAscending": ": aktywuj, by posortować kolumnę rosnąco",
                "sortDescending": ": aktywuj, by posortować kolumnę malejąco"
            }
        }
    });
```

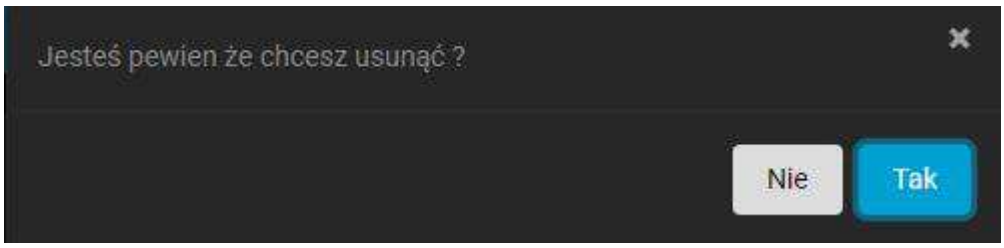
6.10.7 Usunięcie danych

Operacja usunięcia realizowana jest poprzez ustawienie odpowiedniego pola CzyAktywne

Podczas dodawania pole ustawiane jest na true a w czasie usunięcia pole ustawiane na false.

Żadne dane nie są fizycznie usuwane i muszą być potwierdzone w oknie modalnym

Do poprawienia wyglądu okien modalnych użyto frameworka javascript **BootBox**

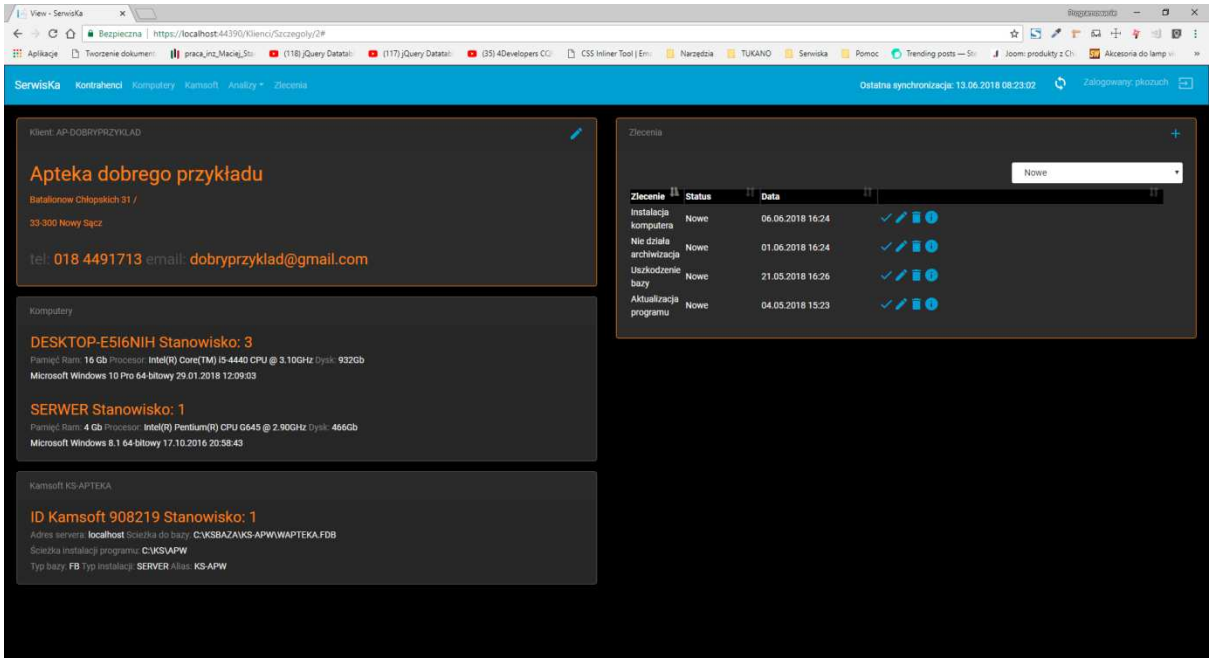


Przykładowe okno modalne BootBox

6.10.8 Widok szczegółów

Widoki szczegółów prezentują pełny zestaw informacji.

Po wybraniu szczegółów z ogólnej tabeli zostajemy przeniesieni do widoku

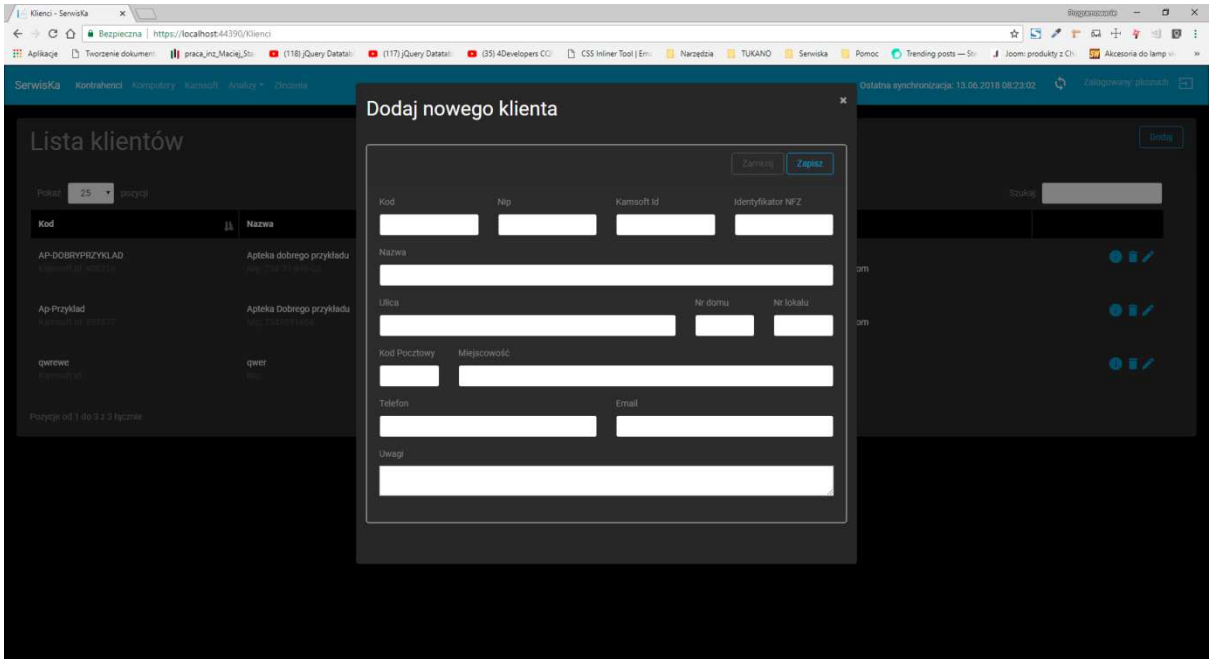


Strona w zależności od wyboru zawiera pełne potrzebne informacje np. o kliencie, komputerach, podzespołach, urządzeniach, systemie operacyjnym, aplikacji, zdarzeniach w aplikacji, zleceniach oraz czynności wykonywanych w ramach zlecenia.

Klikając na każdy z elementów widoku (klienta, komputer, program) zostajemy przeniesieni do szczegółów tego elementu.

6.10.9 Widok dodawania / edycji danych

Z widoku szczegółów możemy dodawać i nowe elementy i edytować istniejące.



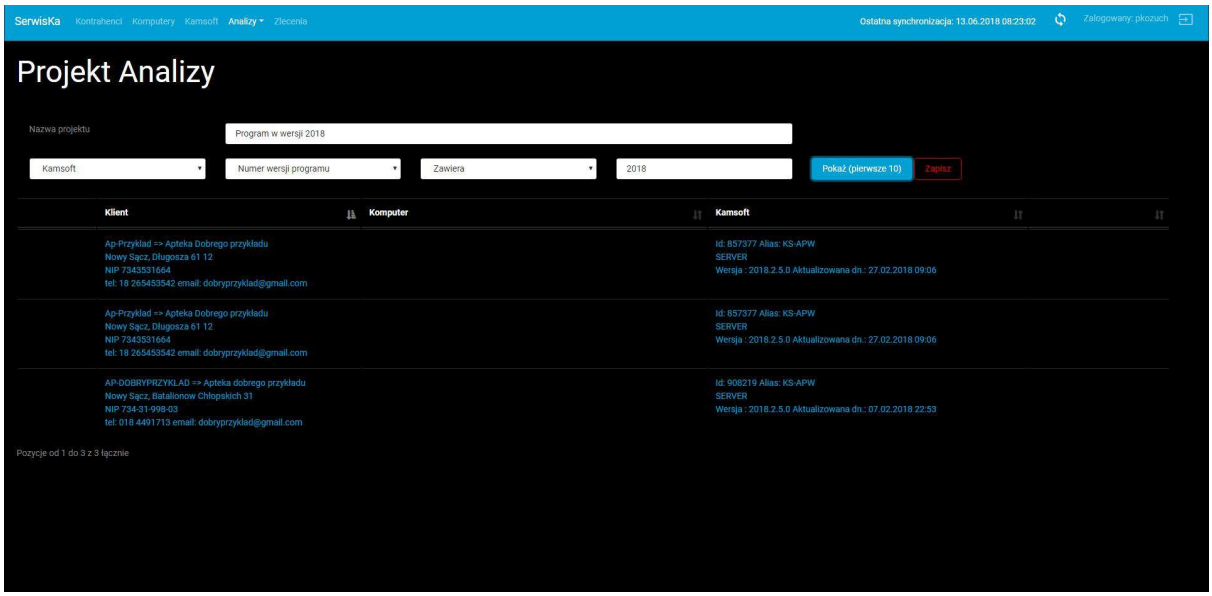
6.10.10 Moduł Analiza

Ciekawym elementem aplikacji jest możliwość zaawansowane do wyszukiwania informacji i zapisywania ich jako opcje menu Analizy

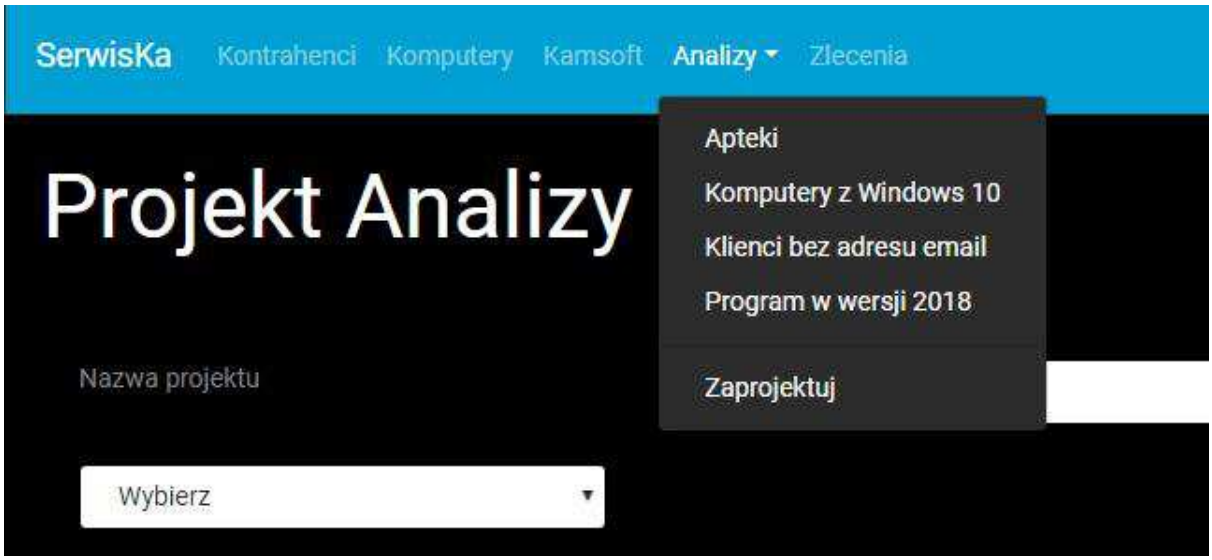
W oknie projektu wybieramy kryteria jakie powinny spełniać przeszukiwane dane, nadajemy im nazwę i zapisujemy.

W menu Analizy pojawia się nowa możliwość wyboru po kliknięciu której uzyskujemy wcześniej zaprojektowany wynik.

Możliwości tego mechanizmu są bardzo duże. Aktualnie zaprezentowano podstawowe funkcje przeszukiwania ale mechanizm ten może być łatwo rozbudowywany i możliwości tworzenia warunków ograniczyć mogą tylko gromadzone dane.



Przykładowy widok projektu analizy



Menu do wyboru zaprojektowanych analiz

6.10.11 Moduł Zlecenia

Zlecenia to moduł pozwalający na prowadzenie ewidencji prac wykonywanych u klienta oraz czynności serwisowych. Możemy je wprowadzać na karcie szczegółów klienta lub w menu zleceń.

Prezentacja zleceń w szczegółach klienta umożliwia szybkie śledzenie historii współpracy i odnalezienie powtarzających się problemów.

Rozdział 7 Testy

Aplikacja została przetestowana w firmie komputerowej zajmującej się serwisem sprzętu i oprogramowania firmy KAMSOFT przez dwóch serwisantów. Spełniła ich oczekiwania i może być użytkowana do tego celu. Pozwoliła na zwiększenie szybkości obsługi serwisowej oraz jej wykorzystanie poprawił stosunki apteka-serwis. Aplikacja uzyskała ocenę bardzo dobrą w zakresie gromadzenia i prezentacji danych, oraz dobrą w zakresie funkcjonalności zleceń i szczegółów zecenia.

Rozdział 8 Podsumowanie

8.1 Realizacja założeń

Aplikacja zgodnie z założeniami pomaga w obsłudze serwisowej klienta. Została dostosowana do wspomagania specjalistycznej obsługi serwisowej branży farmaceutycznej. Ułatwia komunikację z klientem oraz gromadzi dokumentację awarii i wykonywanych czynności serwisowych.

Typowy realizowany scenariusz:

1. Klient zgłasza awarię
2. W czasie rozmowy serwisant wyszukuje w tabeli klientów rozmówcy i wybiera szczegóły
3. Przeglądając informacje o komputerze, podzespołach, systemie, aplikacji, zdarzeniach , historii zleceń ocenia możliwości reakcji na zgłoszenie
4. Wprowadza zlecenie w aplikacji
5. Wykonuje czynności serwisowe i dokumentuje je w szczegółach zlecenia

Wykonana aplikacja umożliwia sprawne i szybkie realizowanie ww scenariusza.

8.2 Możliwości rozwoju

Aplikacja została zaprojektowana i wykonana w aktualnie najnowocześniejszych dostępnych technologiach.

Budowa aplikacji uwzględnia jej dalszy rozwój.

Podzielona została na szereg warstw które mogą być rozwijane niezależnie przez kilku programistów. Ułatwia to także nawigację w projektach i rozwijanie jej poprzez niezależne moduły.

Zastosowanie interfejsów pozwala na wymianę lub przebudowę logiki biznesowej , wprowadzenia testów jednostkowych, oraz szybką wymianę miejsca składowania danych.

Aktualnie wykorzystano mechanizm wymiany danych FTP. Po pojawieniu się odpowiedniej infrastruktury i klienta łatwo można zmienić sposób komunikacji a w razie potrzeby zaimplementować komunikację onLine.

Dla jeszcze większej poprawy jakości obsługi klienta w terenie konieczne stanie się w przyszłości wykonanie aplikacji mobilnej na smartfony. Logika biznesowa i obiekty wymiany danych są przygotowane, więc bardzo szybko można wykonać część widoków aplikacji mobilnej i wykorzystać wspólnie metody już istniejące w projekcie.

Rozdział 9 Źródła wiedzy

9.1 Literatura

Adam Freeman, ASP.NET MVC 5. Zaawansowane programowanie, 2015r.

9.2 Internet

Kursy on-line

- https://www.udemy.com/the-complete-aspnet-mvc-5-course/learn/v4/overview
  The Complete ASP.NET MVC 5 Course
- https://www.youtube.com/watch?v=G1ej2KdU-yo&list=PLRJ9PiYzypEeKTUjk2fxFjZk2RVbv1bnP
  Becoming a software developer - course introduction [PL]

Fora dyskusyjne i strony dokumentacji technicznej

- https://stackoverflow.com
- https://jquery.com
- https://devstyle.pl/2014/10/16/di-uzycie-autofac
- https://autofac.org
- https://automapper.org
- https://getbootstrap.com
- https://bootswatch.com/cyborg
- http://bootboxjs.com
- https://firebirdsql.org
- https://notifyjs.jpillora.com
- https://datatables.net
- https://msdn.microsoft.com
- https://pl.wikipedia.org
- https://www.w3.org
- https://marketplace.visualstudio.com

Dodatek A. Spis zawartości dołączonej płyty CD Praca inżynierska

- niniejszy dokument
- kod źródłowy aplikacji