
POLITECHNIKA ŚLĄSKA

WYDZIAŁ AUTOMATYKI, ELEKTRONIKI I INFORMATYKI

Projekt inżynierski

# Aplikacja społecznościowa oparta na lokalizacji

Autor: Tytus Dragon

Kierujący pracą: dr inż. Alina Momot

Gliwice, Styczeń 2016 r.

# Spis treści

# Wstęp

Kilkanaście lat temu na rynku pojawiły się pierwsze smartfony. W ostatnich latach urządzenia te rozwijały się w błyskawicznym tempie. Producenci tworzą coraz lepsze i doskonalsze urządzenia, które służą do komunikacji, ułatwiają podróżowanie, zachowywanie bezpieczeństwa, dostarczają rozrywki oraz mają wiele różnych zastosowań. Dzięki rozwojowi technologii w ostatnich latach coraz bardziej stają się popularne aplikacje mobilne. Każdy może zarządzać swoim kontem bankowym przez aplikację, używać systemu nawigacji *GPS* podczas jazdy samochodem trafiając do celu bez problemu, rozmawiać z przyjaciółmi i rodziną będąc setki kilometrów od nich, a także opisywać swoje życiowe wydarzenia, publikować zdjęcia oraz filmy w Internecie.

Komunikacja międzyludzka przybrała wiele nowych form dzięki mobilnym technologiom. Wciąż powstają nowe sposoby i pomysły na realizację międzyludzkiej komunikacji takie jak blogi, sieci biznesowe, projekty ułatwiające pracę zespołową, fora dyskusyjne, portale umożliwiające udostępnianie zdjęć, recenzowanie produktów, gry społecznościowe i wiele innych. Niektóre z nich nie mają tylko celów rozrywkowych - możemy się spotkać z wieloma rozwiązaniami biznesowymi usprawniającymi pracę, umożliwiającymi przesyłanie i archiwizację plików, dokumentów. Co więcej, istnieją także aplikacje i portale, które stanowią w średnich i dużych firmach systemy ewidencji czasu pracy. Ogólnie pojęte korzystanie z internetowych i mobilnych technologii w celu komunikacji i interaktywnego dialogu niedawno przyjęło nazwę mediów społecznościowych (ang. *Social media*).

Celem tej pracy jest implementacja społecznościowej aplikacji mobilnej, służącej do komunikacji poprzez udostępnianie swojego położenia dzięki modułowi *GPS*, który obecnie posiada niemal każde urządzenie mobilne dostępne na rynku. Aplikacja będzie umożliwiała posiadanie własnego konta i profilu, możliwość udostępniania o sobie

podstawowych informacji, takich jak imię i zdjęcie z wizerunkiem. Dane te mają posłużyć do identyfikowania przyjaciół oraz wymiany danych między użytkownikami. Ponadto system ma pozwalać na zapraszanie użytkowników do listy znajomych, której celem jest łatwy i szybki dostęp do poszerzonych zasobów zwierających informacje o użytkownikach.

Niniejsza praca składa się z pięciu rozdziałów. Pierwszy przedstawia główne założenia projektu. Drugi zawiera informacje o systemie operacyjnym, któremu dedykowana jest aplikacja oraz opis narzędzi, które były potrzebne podczas projektowania i implementacji systemu. Kolejny rozdział dotyczy specyfikacji wewnętrznej. Są to: szczegóły implementacji, opis bazy danych i serwisu oraz informacje o bibliotekach, które stosowano. Czwarty rozdział to instrukcje dotyczące wymagań sprzętowych dla tworzonego systemu, omówienie instalacji oraz opis interfejsu użytkownika. Piąty rozdział obejmuje testowanie systemu i problemy, które napotkano podczas realizacji projektu. Ostatni rozdział stanowi podsumowanie pracy. Następnie praca zawiera opis literatury oraz załączniki, które zawierają fragmenty kodu źródłowego poszczególnych części implementowanego systemu.

# 1. Analiza tematu

W rozdziale tym znajduje się szerszy opis tematu pracy. Opisano tutaj również motywację tworzenia oprogramowania, które jest przedmiotem tej pracy. Drugą część tego rozdziału stanowią wymagania i założenia, które zdefiniowano, by spełniał je implementowany system.

## 1.1 Motywacje i opis tematu

Przedmiotem projektu jest mobilna aplikacja społecznościowa działająca na smartfonach z systemem operacyjnym Windows Phone 8.1 oraz Windows 10 Mobile. Aplikacja ma służyć do udostępniania danych o użytkownikach takich jak: login, zdjęcie i opis, krótka informacja o użytkowniku o dowolnej treści. Udostępniać ma również datę ostatniej wizyty, dane o lokalizacji użytkownika oraz, co bardzo istotne, mapę w celu przeglądania lokalizacji znajomych.

Inspiracją stworzenia systemu była chęć implementacji oprogramowania, które pozwoliłoby rodzicom na większą kontrolę swoich pociech. Dziecko, którego urządzenie mobilne miałoby zainstalowaną aplikację, która regularnie udostępnia dane *GPS*, byłoby bezpieczniejsze, a w przypadku zgubienia się, również szybko odnalezione. Drugą, choć wynikającą z pierwszej, motywacją była kontrola lokalizacji dzieci niepełnosprawnych, na przykład chorych umysłowo. Aplikacja mogłaby też znaleźć zastosowanie w krytycznych sytuacjach dla bliskich i być pomocniczo wykorzystywana w procedurach stosowanych przez np. organy ścigania.

Ostatecznie zdecydowano się na pomysł nieco bardziej rozbudowanej aplikacji społecznościowej, skierowanej do dowolnej grupy użytkowników, nieograniczonej wiekowo. Zrezygnowano z pomysłu wyłącznej kontroli dzieci ze względów technicznych i ograniczeń darmowych rozwiązań dostępnych technologii mobilnych.

## 1.2 Założenia projektowe

Podrozdział ten zawiera wymagania, które ma spełniać implementowany system. Wymagania te są podstawą do tworzenia kolejnych funkcji bazy danych, serwisu internetowego oraz aplikacji. Definiują również ograniczenia, które autor pracy zakładał przy projektowaniu aplikacji.

Aplikacja będzie przeznaczona dla grup ludzi niezależnych wiekowo. Należy jednak wziąć pod uwagę, że żadne urządzenie, jakim jest smartfon, nie powinno stanowić zabawki dla dzieci poniżej trzeciego roku życia.

Ogólne założenia:

- działanie aplikacji na wielu urządzeniach, należących do architektury: *ARM*, *Neutral* lub *x86*,
- docelowe systemy operacyjne: Windows Phone 8.1, Windows 10 Mobile,
- prosty i przejrzysty interfejs użytkownika, zgodny ze standardami współczesnych technologii Microsoftu [1],
- wyświetlana nazwa aplikacji: *Tillsammens* - oznaczające polskie słowo razem,
- dostęp do Internetu z urządzenia z zainstalowaną aplikacją.

Konta i użytkownicy:

- jeden typ użytkownika w aplikacji – *User*, brak użytkowników z dodatkowym dostępem do danych,
- logowanie do aplikacji za pomocą loginu i hasła,

- jeden użytkownik posiadający tylko jedno konto,
- konto tworzone w aplikacji przy pierwszym logowaniu,
- brak możliwości zmiany loginu, możliwość zmiany hasła użytkownika,
- możliwość trwałej dezaktywacji konta.

Dodatkowe możliwości użytkownika w aplikacji:

- dodawanie zdjęcia profilowego użytkownika,
- udostępnianie opisu, wiadomości dla innych użytkowników,
- udostępnianie danych GPS,
- logowanie ściśle związane z pozostawieniem w bazie informacji o lokalizacji, dacie oraz godzinie,
- możliwość wyszukiwania osób w bazie danych na podstawie loginu,
- lista znajomych każdego użytkownika, system zaproszeń do grona znajomych,
- możliwość przeglądania mapy w celu odnalezienia lokalizacji osób znajdujących się na liście znajomych.

# 2. Wybór narzędzi programistycznych

Ten rozdział zawiera informacje dotyczące systemu operacyjnego, dla którego implementowana jest aplikacja. Opisano także technologie, które wykorzystano i opisano krótko wykorzystywane języki programowania.

## 2.1 System operacyjny

Zdecydowano, że aplikacja będzie stworzona w technologii Windows Phone. Platforma ta jest trzecią najbardziej popularną obecnie na rynku europejskim. W ostatnim czasie traci na popularności i jest wypierana z rynku przez inne popularne systemy operacyjne. Uważa się jednak, że system został dobrze zaprojektowany, więc pomimo dużych ograniczeń, sprawdza się idealnie jako system dla celów biznesowych. Warto także wspomnieć o *Windows Store*, który jest witryną internetową oraz aplikacją – sklepem dostępnym na urządzeniach marki *Microsoft*, z którego pobierać można aplikacje. Docelowo projekt jest tworzony dla wersji systemu Windows Phone 8.1, jednak technologia jest kompatybilna wstecz, co znaczy, iż pliki projektu będą również działać z systemem Windows 10 Mobile.

## 2.2 Języki programowania i technologie

### *Visual Studio 2015*

Visual Studio 2015 Community jest zintegrowanym środowiskiem programistycznym umożliwiającym szeroko pojęte tworzenie oprogramowania [2]. Wykorzystano go w pracy do implementacji zarówno serwisu internetowego w technologii *ASP.NET MVC,* ale między innymi również do generowania bazy danych. Narzędzie to jest podstawowym środowiskiem do tworzenia aplikacji w technologiach *Microsoftu*. Posiada wiele narzędzi ułatwiających pracę programisty. Ponadto przy implementacji wykorzystano *Resharper* - dodatek do środowiska, który jest programem analizującym pisany kod, poprawiającym go, bądź sugerującym optymalniejsze rozwiązania.

### *.NET Framework*

Jest platformą programistyczną, którą opracowała firma *Microsoft*. Jej zadaniem jest zarządzanie elementami systemu, pamięcią oraz kodem [3]. Dostarcza biblioteki klas, które zapewniają podstawowe funkcjonalności oprogramowania. *.NET* umożliwia implementację oprogramowania w wielu różnych językach. Są to między innymi C++, J#, F# oraz C#, który wykorzystano dla implementacji niniejszego projektu.

### *ASP.NET*

Jest technologią wywodzącą się z rodziny *Microsoftu.* Służy do implementacji systemów internetowych, takich jak strony, czy API, wykorzystujących środowisko uruchomieniowe CLR zachowując pełną funkcjonalność dostępną w .NET [4]. Technologia ta w niniejszym projekcie stanowi interfejs komunikacyjny między bazą danych i aplikacją mobilną. Wykorzystując *ASP.NET MVC* zaimplementowano klasy, które należą do grupy kontrolerów, modeli i widoków. Wszystkie z nich pełnią w systemie swoje funkcje. Najważniejszymi z nich są kontrolery, które przyjmują dane

wejściowe od użytkownika odpowiadają na jego zapytania aktualizując modele oraz odświeżając widoki.

## *C#*

Stanowi wysokopoziomowy język obiektowy, wykorzystywany od parunastu lat przez wielu programistów na całym świecie. Mieści się w czołówce najbardziej popularnych współcześnie języków programowania. Program, który napisano w tym języku jest kompilowany do CIL – języka pośredniego, który uruchamia środowisko .NET [5]. Do stworzenia projektu wybrano ten właśnie język ze względu na najdłuższe doświadczenie autora pracy w technologiach związanych z tym językiem.

## *XAML*

Wspomaga pracę nad oprogramowaniem w technologiach takich jak *Windows Presentation Foundation, Silverlight* oraz *Windows Phone*. Jest to język opisu interfejsu użytkownika. Jest jedną z części .NET i tak jak C# jest zamieniany na język wspólny oraz interpretowany w locie. Rozdzielenie zdefiniowanego interfejsu od logiki pozwala na niezależność tworzenia pojedynczych elementów projektu.

## *GIT*

Git jest narzędziem służącym do kontroli wersji. [6] Oprogramowanie to okazuje się być niezwykle przydatnym podczas rozwijania projektów programistycznych. Pozwala na szybki wgląd do kodu. Łatwo w nim porównać dwa różne elementy, usunąć zmiany i wrócić do wcześniejszej wersji np. z powodu popełnienia błędów. Pozwala także na tworzenie gałęzi projektu, co stanowi ogromne wsparcie w przypadku dużych projektów, nad którymi pracuje wiele osób.

## *Entity Framework*

Narzędzie również powstało jako część platformy .NET. Jest narzędziem typu ORM (ang. Object Relational Mapping) [7]. Pozwala na wiele ułatwień przy implementacji baz danych. W projekcie służy jedynie do zdefiniowania i wygenerowania bazy danych, chociaż jest narzędziem wielu zastosowań. Przekształca stworzony schemat w plik bazy danych przy obecności platformy *Azure*.

## *Azure*

Jest również produktem *Microsoftu*. Stanowi platformę chmurową umożliwiającą uruchamianie aplikacji, stron internetowych, a także stanowi mechanizmy składujące dane oraz przetwarzające je [8]. Serwis, który implementowano w niniejszym projekcie korzysta z usług Azure. Pozwala na zasadniczo prosty proces wdrażania aplikacji do Internetu. W praktyce proces ten jest dość skomplikowany i łatwo wyłączyć zasoby nieświadomie lub zaburzyć ich współpracę, której potem przywrócenie w środowisku *Visual Studio* jest dużym wyzwaniem.

## *MVVM*

Jest to wzorzec architektoniczny. Stosuje się go w celu rozwiązania częstych problemów związanych z implementacją programu. *Model-View-ViewModel* ma wiele zalet i jest prosty w działaniu. Z powodu większej ilości plików i klas łatwo pomyśleć, że stosowanie jakichkolwiek wzorców projektowych to utrudnianie pracy i zbędna strata czasu. Zarówno *Model-View-Controller*, jak i *Model-View-ViewModel* pozwala na oszczędność czasu w późniejszych fazach projektu. Jest niezwykle przydatny pod względem modyfikowalności z uwagi na podział na odrębne moduły odpowiadające za widok oraz logikę. Pozwala na łatwe przenoszenie oprogramowania na inne technologie oraz umożliwia przeprowadzanie zautomatyzowanych testów [9].

# 3. Specyfikacja wewnętrzna

Niniejszy rozdział zawiera informacje związane ze specyfikacją wewnętrzną implementowanego systemu. Omówiono w nim bazę danych, wraz z jej elementami, wykorzystane biblioteki, strukturę projektu oraz zawarto niektóre ważne, zaimplementowane algorytmy.

## 3.1 Baza danych

Podrozdział ten zawiera wszelkie informacje dotyczące bazy danych wykorzystanej w niniejszym projekcie. Znajduje się tu również opis tabel i ich pól.

*Azure* umożliwia tworzenie własnych baz danych i osadzenie ich uruchomionych na serwerach platformy. Pustą bazę danych można utworzyć korzystając z portalu *Azure* w przeglądarce. Należy jednak najpierw zdefiniować serwer, na którym baza ta ma być osadzona. W celu utworzenia bazy danych dla niniejszego projektu zdefiniowano serwer o nazwie *project18server.database.windows.net*. Pierwsza część nazwy jest nazwą własną. W trakcie implementacji owego projektu tworzono te zasoby wielokrotnie w celach testowych, dlatego nazwa jest tak ogólna. Niemniej jednak baza danych nosi nazwę: *project18dataBase*, co stanowi nazwę podobną do serwera. W portalu można na wiele sposobów monitorować swoje zasoby. Przydatne okazuje się monitorowanie błędów lub sprawdzanie, czy zasób jest w trybie on-line oraz ile miejsca zajmują dane w bazie danych.

Baza danych składa się z dwóch tabel. Początkowo składać miała się z czterech elementów, jednak zoptymalizowano ją i uproszczono, gdyż proste rozwiązanie okazało

się być bardziej funkcjonalne i było wystarczającym rozwiązaniem na potrzeby niniejszego projektu.

Aby utworzyć bazę danych użyto narzędzia *Entity Framework*. Konieczne było zaprojektowanie bazy danych w podprogramie dostępnym w środowisku programistycznym, a następnie wygenerowanie jej przy pomocy owego narzędzia. Podczas tworzenia ważnym jest, by w zakładce *ServerExplorer* zdefiniować serwer i bazę danych, w których chcemy stworzyć zaprojektowany model. W niniejszym projekcie posłużono się pustą bazą danych, o której wcześniej mowa. W ten sposób generowana baza danych posiada ten sam schemat modeli, co implementowany projekt *ASP.NET*, co ułatwia pracę z wydobywaniem z bazy danych tablic i ich elementów. Fragment wygenerowanego skryptu *SQL* znajduje się w załączniku A.

Tabela *UserSet* jest pierwszym elementem struktury bazy danych. Reprezentuje jednego użytkownika aplikacji. Stanowi ją 10 pól różnych typów danych:

- **Id –** pole typu *Int32*; unikalny identyfikator każdego obiektu typu *User*, klucz główny, jego wartości są numerowane automatycznie,
- **Login –** łańcuch znaków; ograniczony do 25 elementów, również unikalny, jednak unikalnością zajmuje się serwis, który przy tworzeniu użytkownika zgłasza błąd w przypadku próby utworzenia konta o już istniejącym loginie,
- **Password –** zaszyfrowane hasło potrzebne przy logowaniu do aplikacji; nieograniczony łańcuch znaków,
- **LastVisit –** pole typu *DateTime*; przechowuje datę i godzinie ostatniej wizyty użytkownika w aplikacji, przy uruchomieniu aplikacji jest aktualizowane razem z polami dotyczącymi lokalizacji użytkownika,
- **OpenDate –** pole typu *DateTime*; znajduje się w nim data i godzina, w której założono konto danego użytkownika, nie jest modyfikowane przez żadne zapytanie w API.
- **CloseDate –** pole typu *DateTime* przechowujące datę i godzinę, w której użytkownik usunął konto, które nigdy nie jest usuwane trwale – jedynie aktualizowane jest, gdy następuje dezaktywacja konta, służy do filtrowania kont przy logowaniu, pobieraniu znajomych oraz przy wyszukiwaniu użytkowników w serwisie,

- **Desc –** łańcuch znakowy – pełni funkcję opisu danego użytkownika, a zakładający konto może napisać w aplikacji informacje o sobie, a opis jest przechowywany w tym polu; początkowo każdy użytkownik ma domyślny opis, jednak w aplikacji łatwo można go zmienić,
- **PhotoUri –** łańcuch znaków, który przechowuje adres *URL* pliku graficznego pełniącego funkcję zdjęcia profilowego danego użytkownika; jest pobierane przy logowaniu do aplikacji, przy szukaniu znajomych oraz przy wyświetlaniu listy znajomych,
- **X –** pole typu *double* stanowiące pierwszą wartość definiującą lokalizację użytkownika – szerokość geograficzną, aktualizowane przy każdym zalogowaniu do aplikacji przez użytkownika oraz w trakcie działania podczas odświeżania stanu aplikacji.
- **Y –** pole typu *double* będące drugą częścią lokalizacji użytkownika – stanowi długość geograficzną, aktualizowane tak często, jak powyższe pole *X* tabeli *UserSet*.

Tabela *InvitationSet* jest następnym elementem struktury bazy danych. Reprezentuje relację dwóch użytkowników aplikacji. Składa się z czterech pól:

- **Id –** pole typu *Int32;* unikalny identyfikator każdego obiektu typu *Invitation*; klucz główny, numerowane automatycznie,
- **SenderId –** pole typu *Int32* zawierające w sobie identyfikator użytkownika, który tworzył zaproszenie do grona znajomych dla innego użytkownika,
- **RecieverId –** pole typu *Int32* stanowiące identyfikator użytkownika, do którego zaproszenie do grona znajomych jest adresowane,
- **Status –** łańcuch znaków prezentujący status danego zaproszenia do grona znajomych. Na podstawie tego pola Kontroler filtruje użytkowników w zależności od rodzaju danego zapytania,
- **SenderLogin –** łańcuch znaków stanowiący *Login* użytkownika, który stworzył dane zaproszenie w aplikacji; pole to umożliwia wyświetlenie komunikatu zaproszenia do grona znajomych adresatowi bez dodatkowego pobierana całego obiektu *User*.

## 3.2 Struktura projektu

Implementowany projekt stanową trzy warstwy. Pierwsza warstwa jest bazą danych utworzoną przy pomocy języka *SQL* osadzoną na serwerach platformy *Azure*. Jest osobną warstwą pod względem logicznym, jednak jej struktura została wygenerowana przy pomocy narzędzia *Entity Framework* równocześnie z implementacją serwisu. Serwis stanowi następną warstwę logiczną. Jest napisany w technologii *ASP. NET MVC*. Na potrzeby aplikacji jest jedynie interfejsem komunikacyjnym między aplikacją a bazą danych, wydobywającym, filtrującym oraz przetwarzającym dane za pomocą modeli, które stanowią klasy utworzone razem z bazą danych oraz za pomocą kontrolerów, które odpowiadają na zapytania wysłane z poziomu aplikacji przetwarzając dane wydobyte z bazy danych. W owym serwisie wykorzystano także niektóre metody pochodzące z biblioteki *Linq*, która jest oparta na języku *SQL*. Technologia ta pozwala na stworzenie uniwersalnego serwisu typu *RESTfull* [10], dlatego chcąc na przykład rozbudować projekt, nie powinna stanowić problemu implementacja niniejszej aplikacji na inną platformę niż Windows Phone.

By stworzyć API, czyli interfejs komunikacyjny dla bazy danych oraz aplikacji Windows Phone skorzystano z technologii *ASP.NET MVC*. Nie wykorzystano całego potencjału owej technologii, gdyż tworzenie API to tylko jedna z wielu jej możliwość. Projekt stanowią klasy w dużej części wygenerowane przy tworzeniu projektu. Najważniejsze w implementacji tego serwisu było utworzenie połączenia z zewnętrznym zasobem, jakim jest baza danych oraz implementacja Kontrolerów. Połączenie to zostało zdeklarowane w projekcie w *Web.config*.

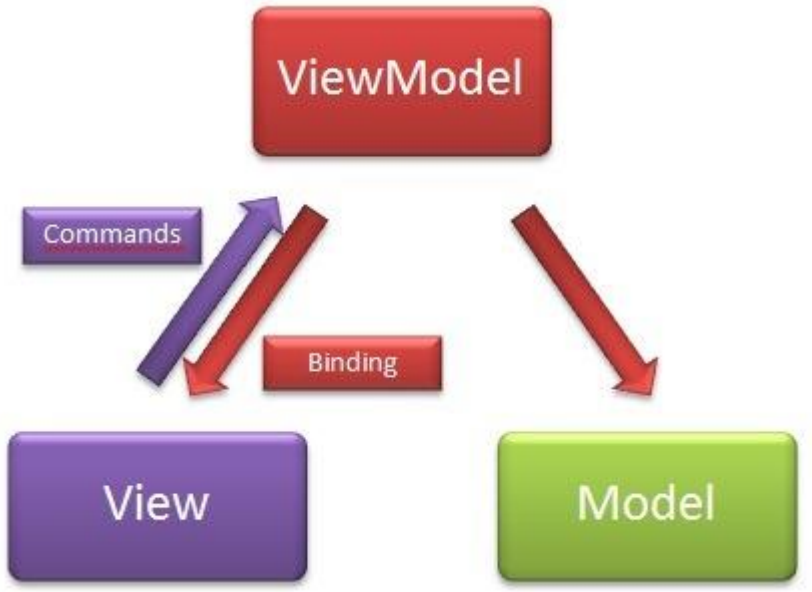
*MVC* stanowi wzorzec architektoniczny dla aplikacji w technologii *ASP.NET*. W projektowaniu serwisu wykorzystano właściwie tylko dwie części wzorca. Wygenerowany widok służy wyłącznie do ręcznego przeglądania danych i jest zbędny w przypadku tworzenia API typu *RESTfull*. W projekcie znajdziemy folder o nazwie *Controllers*. To w nim znajdują się najważniejsze z klas. Część z nich została również wygenerowana przy pomocy narzędzi *Entitty Framework*. Zaimplementowano 10 klas typu kontroler. Każdy z nich odpowiada za inne zapytania pełniąc różne funkcje w

systemie. Najważniejszym kontrolerem jest *UserController*, który odpowiada za tworzenie najważniejszego obiektu w systemie, jakim jest użytkownik. Kod klasy znajduje się w załączniku B1. Klasa posiada metodę *PostUser*, która tworzy obiekt odpowiadający za połączenie z bazą danych. Następnie sprawdza, czy znajdujący się w parametrach *Login* nowego użytkownika nie występuje już w bazie. W przypadku, gdy login jest unikalny, metoda dodaje użytkownika do bazy danych, a następnie zwraca kod *200,* weryfikując powodzenie procedury [11].

Najbardziej polecanym wzorcem architektonicznym dla Windows Phone jest *MVVM*, dlatego ten właśnie wybrano w niniejszym projekcie przy tworzeniu aplikacji.

Zastosowanie *MVVM* wiąże się z podziałem aplikacji na trzy moduły (Rys. 1):

- **Model** – zawiera logikę biznesową aplikacji, językiem jest C#,
- **View** – stanowi interfejs użytkownika, implementuje się go w *XAML* oraz C#,
- **ViewModel** – jest łącznikiem dla modelu oraz widoku, stosuje mechanizm Bindowania oraz Komendy, językiem implementacji jest C# [12].




RYSUNEK 1. SCHEMAT WZORCA ARCHITEKTONICZNEGO MODEL-VIEW-VIEWMODEL.

Stosowanie takiego modelu prowadzi także do możliwości podziału projektu na trzy odrębne moduły, podprojekty, które mają swoje określone funkcje i stanowią różne części oprogramowania. Solucja aplikacji mobilnej składa się z następujących podprojektów:

- Tillsammens.WindowsPhone.App,
- Tillsammens.WindowsPhone.Domain,
- Tillsammens.WindowsPhone.WebServices.

Pierwszy podprojekt stanowi zbiór klas głównie dedykowanych technologii Windows Phone. Zawiera całą logikę oprogramowania. Znajdują się w nim widoki oraz klasy dziedziczące po klasie *ViewModelBase*. Podprojekt ten zawiera także pliki typu *resource dictionares,* pisane w *XAML*, które odpowiadają za wygląd aplikacji i są odpowiednikiem plików w formacie *.css* dla aplikacji pisanych w ponadczasowym *HTML*. Podprojekt również zawiera folder z niezbędnymi klasami konwerterów, które wspomagają *XAML,* np. służą do interpretacji właściwości typu *bool* na właściwość *Visibility*, która odpowiada, czy dany element interfejsu użytkownika jest widoczny, bądź nie. Innym przykładem konwertera jest zmiana obiektu *DateTime* na odpowiedni, skrócony format, bez konieczności robienia tego w *ViewModel*.

*Domain* oraz *WebServices* to biblioteki klas, które zawierają funkcje i metody uniwersalne dla całej technologii .NET. Wyposażono je w klasy pomocnicze przy implementacji projektu dla Windows Phone, ale same stanowią odrębną część. Pozwala to na łatwe przeniesienie aplikacji na inną technologie, głównie przez zmianę interfejsu użytkownika, a w przypadku tej pracy szczególnie na przejrzystość implementowanego kodu i możliwość zachowania porządku w całej solucji.

W *WebServices* znajdziemy klasy, które odpowiadają za wysyłanie zapytań do serwisu typu *RESTfull*, który stanowi odrębną warstwę projektu. Znajduje się tam także folder *Dto*, w którym można znaleźć klasy stanowiące *Model* dla wzorca *MVVM*.

## 3.3 Biblioteki

W celu usprawnienia implementacji projekt jest wsparty zewnętrznymi bibliotekami. Ten krótki podrozdział jest poświęcony dodatkowym bibliotekom wykorzystanym do implementacji aplikacji dla systemu Windows Phone. Wszystkie użyte biblioteki dołączono do solucji przy pomocy *NuGet*, narzędzia dostępnym w *Visual Studio*. Dzięki temu narzędziu można sprawnie i szybko dodawać biblioteki do swojego oprogramowania.

Ważną biblioteką, którą użyto w niniejszym projekcie jest *RestSharp*. Biblioteka *RestSharp* jest zbiorem klas, które ułatwiają realizację aplikacji korzystających z Internetu. Przy użyciu tej biblioteki stworzono w podprojekcie *WebServices* uniwersalną dla wszystkich zapytań klasę *RestClientBase*, która odpowiada za komunikację aplikacji z serwisem. Skorzystano głównie z trzech klas tej biblioteki:

- RestClient – klasa definiująca połączenie z danym serwisem typu RESTfull, w jej konstruktorze podaje się główny adres serwisu, do którego aplikacja będzie wysyłać zapytania,
- RestRequest – klasa, która odpowiada za pojedyncze zapytanie, instancje tej klasy są tworzone przy każdym zapytaniu i pozwalają na zdefiniowanie zapytania w formie łańcucha znaków oraz wybranie rodzaju metody spośród dostępnych, takich jak: GET, POST, DELETE, czy PUT,
- RestResponse – klasa, która jest wykorzystywana w chwili zapytania, do niej przekazywana jest cała odpowiedź danego zapytania, a w przypadku pomyślnej odpowiedzi dane są serializowane i wysyłane w inne moduły programu.

Drugą ważną, zastosowaną biblioteką jest *GalaSoft.MvvmLight*. Instalacja tej biblioteki zaopatruje projekt w dodatkowe funkcje oraz klasy, które stanowią podstawę wzorca projektowego *Model-View-ViewModel,* o którym mowa w poprzedniej części tego rozdziału. W projekcie wykorzystano między innymi takie klasy jak *ViewModelLocator*, który ułatwia dostęp do klas, które dziedziczą po klasie *ViewModelBase*. Użyto także *RelayCommand* klasę – która, należy do tej biblioteki i

implementuje interfejs *ICommand*, z której korzysta się w mechanizmie bindowania komend np. do przycisków w widoku.

## 3.4 Implementacja – wybrane fragmenty

Podczas implementacji systemu zmierzono się z kilkoma zagadnieniami, które pod względem programistycznym należy krótko omówić. Ten podrozdział zawiera kilka komentarzy dotyczących tych fragmentów kodu.

### *Pobranie znajomych z bazy danych*

Pobranie znajomych przez serwis jest pierwszym zagadnieniem, które wybrano do omówienia w tej części pracy. W kontrolerze *FriendsConrtoller* znajduje się metoda typu *GET*, której zadaniem jest zwrócić wszystkich znajomych użytkownika o identyfikatorze podanym jako parametr wejściowy typu *int*.

Metoda pobiera z bazy danych obiekty typu *Invitation*. Przy pomocy metody *Where* pochodzącej z biblioteki *Linq* wydobywa zaproszenia, które dotyczą użytkownika o danym identyfikatorze oraz mają status zaakceptowanych.

```
var invitations = db.InvitationSet .Where(invitation =>
                      (invitation.RecieverId == id ||
                      invitation.SenderId == id) &&
                      invitation.Status.Contains("Accepted"))
                      .ToList();
```

Tylko na podstawie tych obiektów tworzy się obecna lista znajomych dla danego użytkownika. Następnie odbywa się wydobycie wszystkich potrzebnych obiektów typu *User* poprzez zawarte w liście identyfikatory.

```
foreach (var i in invitations)
   if (i.SenderId == id)friends.Add(db.UserSet.Find(i.RecieverId));
   else friends.Add(db.UserSet.Find(i.SenderId));
```

Pętla wykorzystująca zapytanie *Linq* filtruje aktywnych użytkowników i zwraca potrzebne dane.

```
var sortedFriends = (from f in friends
                        where f.CloseDate > DateTime.Now select new
                        User
                        {
                            Id = f.Id,
                            Login = f.Login,
                            PhotoUri = f.PhotoUri,
                            LastVisit = f.LastVisit,
                            Desc = f.Desc,
                            X = f.X,
                            Y = f.Y,
                        }).ToList();
                        return Ok(sortedFriends);
```

Całość procedury można znaleźć w załączniku B2.

## *Klasa bazowa dla struktur typu ViewModel*

W projekcie *Tillsammens.WindowsPhone.App* w folderze *ViewModels* znajduje się klasa, która jest podstawową dla klas typu *ViewModel*. Jako jedyna dziedziczy po klasie podstawowej dla tego wzorca architektonicznego. Wybrano ją do omówienia, gdyż zawiera w sobie te pola, właściwości i metody, które uznano za wspólne dla wszystkich klas. Zastosowanie dziedziczenia znacznie ułatwiło implementacje kolejnych klas typu *ViewModel*. Struktura klasy znajduje się w załączniku B3.

Charakterystyka klasy TillsammensViewModelBase:

- **TillsammensService** – pole, które pozwala na inicjowanie zapytań,
- **DialogService** – pole służące do komunikacji miedzy *View* a *ViewModel,*
- **IsWorking –** właściwość typu *bool*, odpowiada za wyświetlanie kontrolek w trakcie ładowania danych,
- **ShowWebResultCommunicate** – metoda odpowiedzialna za wyświetlanie komunikatów w przypadku niepowodzenia odbioru danych z serwisu,
- **StartWorking** – metoda zmieniająca pole *_startWorking* na wartość *true* oraz włączająca pasek ładowania na ekranie,
- **StopWorking –** metoda przeciwna metodzie *StartWorking*.

### *Uniwersalna klasa odpowiedzialna za zapytania http*

W projekcie *Tillsammens.WindowsPhone.WebServices* znajduje się abstrakcyjna klasa *RestClientBase* odpowiadająca za wywoływanie metody *CallAsync*. Metoda ta przyjmuje jako parametr obiekt implementujący interfejs *IRestRequest*. Zaimplementowana została jako typ Task, co znaczy, że jest wywoływana asynchronicznie z przedrostkiem *await*. Jako wartość zwracaną zdeklarowano obiekt implementujący interfejs *TResponse* pochodzący również z biblioteki RestSharp *[13]* . Wywołuje metodę *Execute* klasy *RestClient,* a następnie przypisuje zmiennej otrzymaną odpowiedź. Następnie jest sprawdzany rezultat otrzymany z zewnętrznych zasobów pod kątem błędów i zawartości. Jeśli status odpowiedzi jest zgodny z oczekiwanym dane są serializowane na obiekty, których typ podano na wejściu do metody. Struktura klasy oraz wspomnianej metody znajduje się w załączniku B4.

# 4. Specyfikacja zewnętrzna

Ten rozdział zawiera opis specyfikacji zewnętrznej niniejszego projektu. Omówiony w nim został szczegółowo interfejs użytkownika oraz podane zostały niezbędne informacje dotyczące uruchomienia aplikacji i korzystania z niej.

## 4.1 Wymagania sprzętowe

Podrozdział ten zawiera niezbędne informacje do uruchomienia aplikacji. Informacje dotyczą wymagań sprzętowych oraz dostępnych systemów operacyjnych, na których można uruchomić aplikację.

System Windows Phone 8.1 oraz Windows 10 Mobile obecnie można znaleźć jedynie na urządzeniach marki *Nokia* oraz *Microsoft* z drobnymi wyjątkami marki *HTC*, czy *Alcatel*. To spore ograniczenie, ale w przypadku braku dostępnego urządzenia można skorzystać z emulatora – maszyny wirtualnej instalowanej przy dodawaniu Windows Phone SDK do środowiska Visual Studio 2015 Community.

Program uruchamiano wielokrotnie na różnych urządzeniach z systemem operacyjnym Windows Phone 8.1 oraz Windows 10 Mobile. Najsłabszy sprzętowo smartfon, na którym testowano oprogramowanie uznano za telefon z minimalną konfiguracją sprzętową.

Oto minimalne parametry sprzętu:

- Procesor: Qualcomm MSM8227, 1,00GHz,
- RAM: 512MB,
- wolne miejsce w pamięci: 4MB,
- system Operacyjny: Windows Phone 8.1.

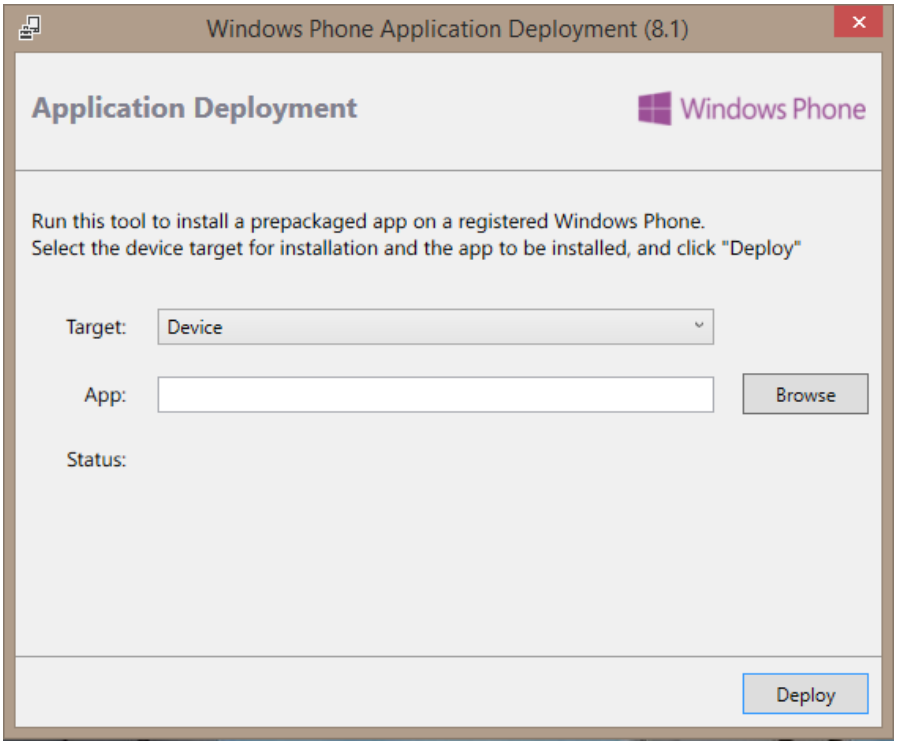
## 4.2 Instalacja

Program dedykowany systemowi Windows Phone 8.1, który nie został opublikowany w *Windows Store* – płatnej platformie, służącej do dystrybucji aplikacji należy instalować za pomocą narzędzia *Windows Phone Application Deployment 8.1*, które jest jednym z komponentów *Windows Software Development Kit for Windows 8.1*. Należy to zrobić w kilku krokach:

- Należy zainstalować *Windows Software Development Kit for Windows 8.1*, oprogramowanie to można pobrać ze strony internetowej [14],
- następnie urządzenie, na którym chcemy zainstalować program należy włączyć i odblokować,
- ważnym jest, by data i godzina na telefonie była ustawiona na prawidłową,
- należy podłączyć smartfon przez kabel USB do komputera, z którego chcemy dokonać instalacji pliku,
- jeśli na urządzenie wgrywane jest oprogramowanie pierwszy raz, należy je zarejestrować jako urządzenie wykorzystywane do celów programistycznych:
    1. uruchomić *Windows Phone Developer Registration 8.1*,
    2. gdy urządzenie zostanie wykryte przez program, należy wybrać przycisk *Register*,
    3. wyświetlone zostanie okno logowania do konta Microsoft, w którym należy się zalogować i zaakceptować logowanie,

- następnie należy uruchomić *Windows Phone Application Deployment 8.1* (Rys. 2),

- program wymaga wybrania z listy pozycji *Device*, aby zainstalować aplikację na urządzeniu zewnętrznym,
- następnie trzeba podać ścieżkę, w której znajduje się plik instalacji z rozszerzeniem *.appx* wybierając przycisk *Browse*,
- ostatnim krokiem jest wybranie przycisku *Deploy*, który rozpocznie proces instalacji aplikacji o wyświetlanej nazwie *Tillsammens,*
- po pomyślnej instalacji urządzenie posiada na liście zainstalowanych aplikacji program, który jest gotowy do uruchomienia.



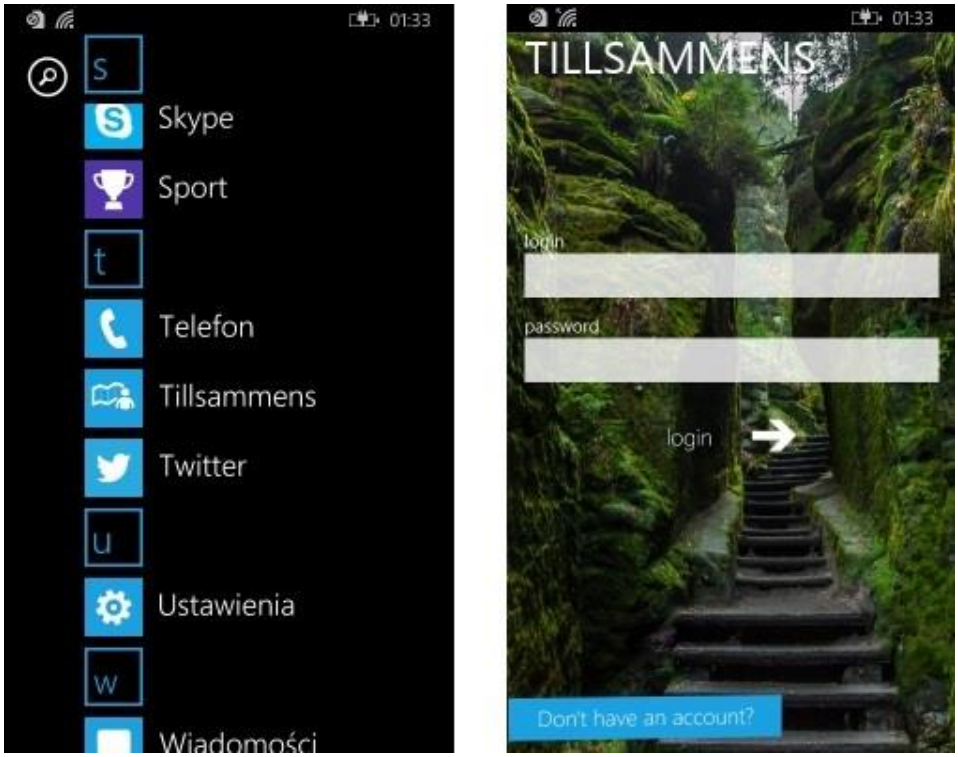

RYSUNEK 2 PROGRAM DO INSTALACJI APLIKACJI DLA WINDOWS PHONE 8.1.

## 4.3 Interfejs użytkownika

Podrozdział ten stanowi omówienie części oprogramowania, z którą użytkownik ma bezpośredni kontakt. Omówiono w nim funkcjonalności oraz użytkowanie programu.

## *Tworzenie konta i logowanie*

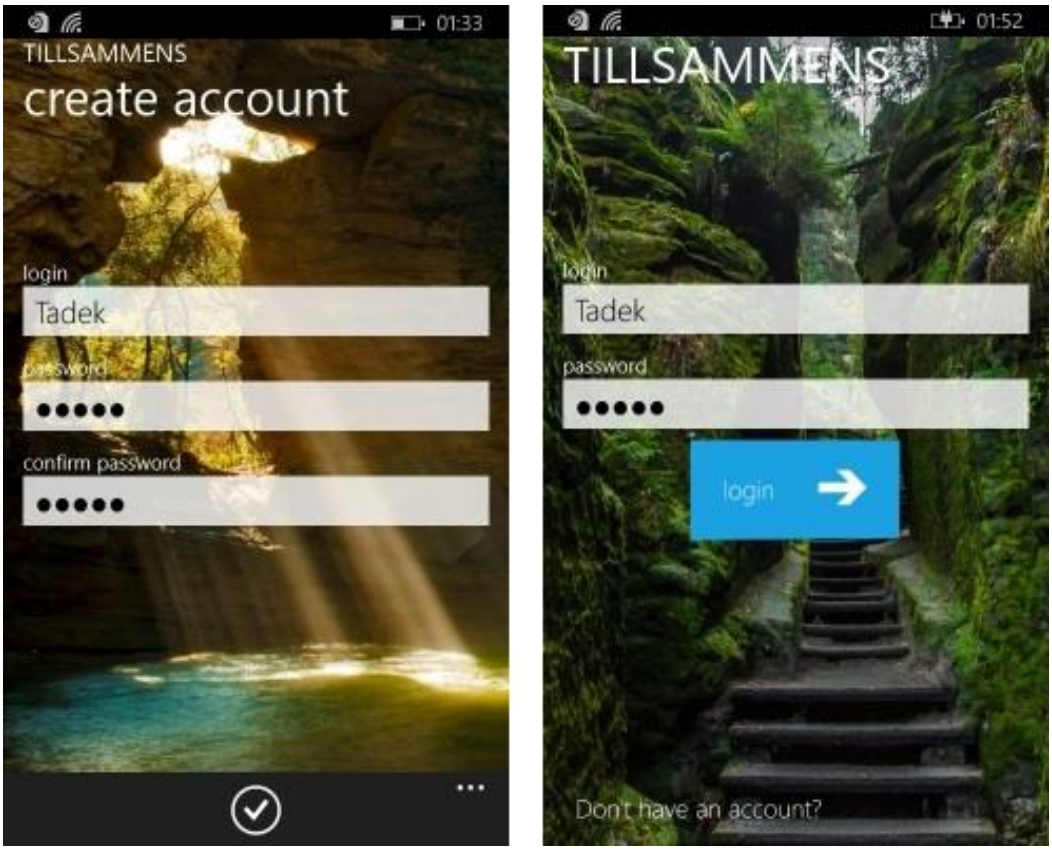
Jeśli smartfon pomyślnie zainstalował aplikację, możemy ją wybrać z listy aplikacji (Rys. 3A) w celu uruchomienia. Należy wybrać kafelek o nazwie *Tillsammns*, aby uruchomić aplikację. Wyświetlony użytkownikowi został ekran logowania. W lewym dolnym rogu znajduje się napis. Aby utworzyć konto należy go wybrać (Rys. 3B).




**RYSUNEK 3. A - ZAINSTALOWANA APLIKACJA NA LIŚCIE W MENU, B - EKRAN LOGOWANIA - TWORZENIE NOWEGO KONTA.**

Należy podać indywidualny login, a następnie podać hasło powtarzając je w dwóch polach typu *PasswordBox* (Rys. 4A). Jeśli pola zostaną puste bądź hasła nie będą zgodne, a przycisk podpisany słowem *Accept* zostanie wybrany, wyświetlona będzie informacja o popełnionym błędzie. W przypadku podania istniejącego loginu najpierw pojawi się napis *Loading*, a następnie komunikat o błędzie. Gdy konto zostanie utworzone pomyślnie użytkownik otrzyma powiadomienie, jednak od razu zostanie ponownie pokazany ekran logowania, w którym można się już zalogować stosując swój nowy login oraz hasło (Rys. 4B). Dane logowania należy podać uwzględniając wielkość liter i wybrać *Enter*, gdy klawiatura jest widoczna, bądź nacisnąć *login.* Jeśli telefon ma łączność z

Internetem oraz podane zostały poprawne dane logowania zostanie pokazany główny ekran aplikacji.




**RYSUNEK 4. A - TWORZENIE NOWEGO KONTA UŻYTKOWNIKA, B - LOGOWANIE DO APLIKACJI.**

## *Ekran główny aplikacji*

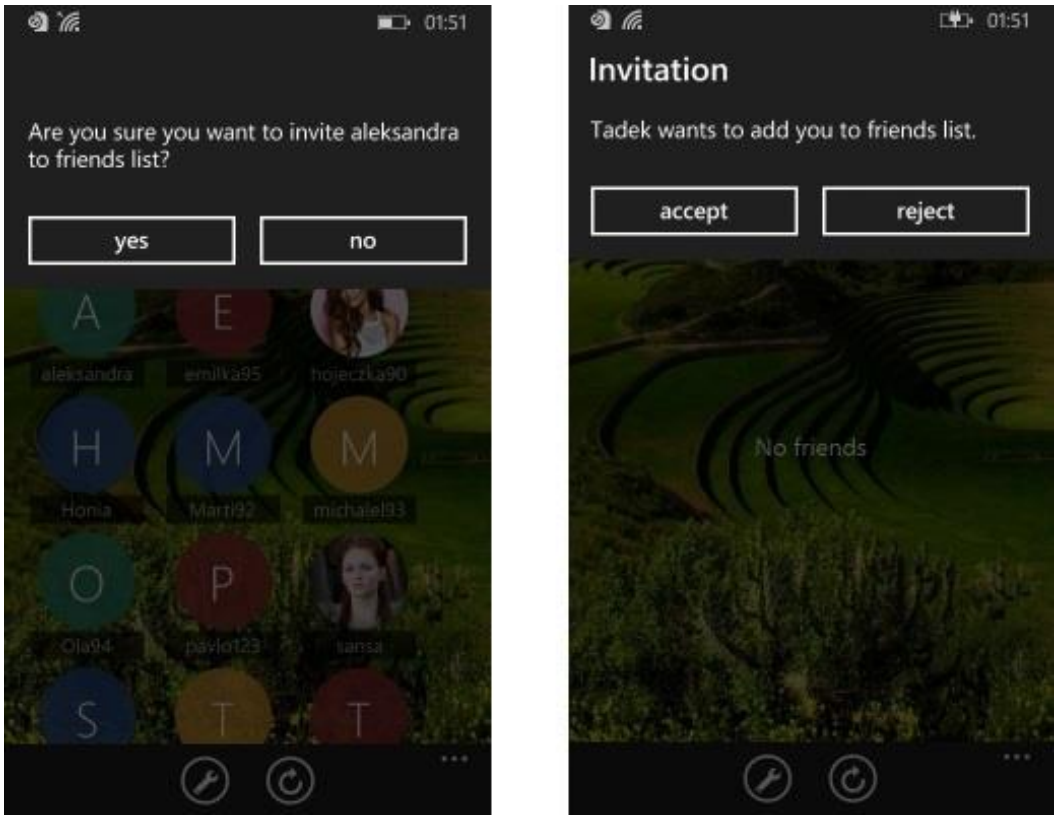
Jeśli użytkownik poprawnie został zalogowany do aplikacji, zostanie pokazany główny ekran (Rys. 5). Pierwszy moduł to lista znajomych. Łatwo zauważyć przyciski menu: *settings* oraz *refresh.* Pierwszy z nich nawiguje do ekranu z ustawieniami konta, drugi natomiast powoduje ponowne załadowanie listy znajomych, pobranie niezaakceptowanych zaproszeń do grona znajomych oraz uaktualnia pola o użytkowniku w module o nazwie *profile*. Dotykając ekranu i przesuwając po nim palcem w lewo lub w prawo ekran pokaże następny moduł. Wybierając jednego ze znajomych poprzez dotknięcie elementu listy użytkownikowi zostanie pokazana strona mapy, gdzie wyświetleni będą jego znajomi usytuowani w ostatnich udostępnionych lokalizacjach. Przytrzymując pozycję listy możemy usunąć użytkownika z grona znajomych.


**RYSUNEK 5. GŁÓWNY EKRAN APLIKACJI PO ZALOGOWANIU.**

Nawigując do modułu *search* w głównym oknie użytkownik ma możliwość przeszukania bazy danych przy pomocy podanej frazy. Należy podać frazę w białym elemencie typu *TextBox* i wybrać przycisk oznaczony lupą, by wysłać zapytanie do serwisu. Po chwili wyświetlona zostanie lista użytkowników, których loginy zawierają podaną wcześniej frazę (Rys. 6). Aplikacja nie wyświetli zalogowanego użytkownika, który tę procedurę szukania wywołał. Gdy nastąpiło wyświetlenie loginów użytkownik ma możliwość dotknięcia elementu listy, by zainicjować procedurę zaproszenia do grona znajomych.


**RYSUNEK 6. WYSZUKIWANIE UŻYTKOWNIKÓW W BAZIE DANYCH.**

Jeśli użytkownik wybierze osobę z listy zostanie wyświetlony komunikat potwierdzający chęć dodania osoby do grona znajomych (Rys. 7A). Gdy użytkownicy są już znajomymi, użytkownik zostanie o tym również poinformowany. W przypadku wysłania zaproszenia, zaproszony użytkownik po następnym logowaniu do aplikacji zostanie zapytany przy pomocy komunikatu o potwierdzenie znajomości (Rys. 7B).



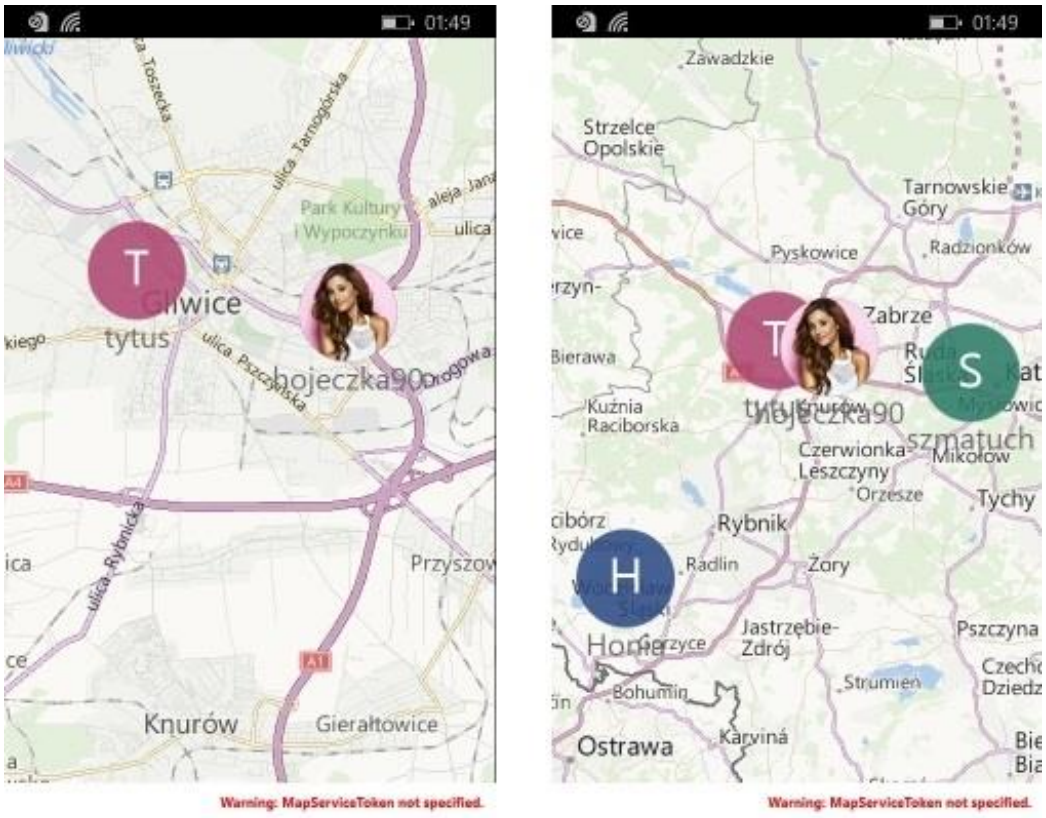

**RYSUNEK 7. A - INICJOWANIE ZAPROSZENIA DO GRONA ZNAJOMYCH, B – OTRZYMANIE ZAPROSZENIA PRZEZ ADRESATA ZAPROSZENIA.**

Ostatni moduł głównego ekranu dotyczy zalogowanego użytkownika (Rys.8). Można tutaj zobaczyć Login zalogowanego użytkownika oraz bieżące zdjęcie reprezentujące profil. Poniżej tych danych znajduje się miejsce, w którym można zmienić zdjęcie użytkownika zamieniając jego *Url* oraz potwierdzając przyciskiem wyboru ✓ oznaczającym akceptację zmian znajdującym się obok pola tekstowego. Następnie wyświetlony jest status – opis użytkownika, modyfikowalny w sposób analogiczny do zmiany zdjęcia profilowego. Poniżej jest informacja, mówiąca o tym, kiedy ostatnio zaktualizowano lokalizację użytkownika. Warto tutaj wspomnieć, że w momencie zalogowania zostaje uruchomiona procedura pobrania lokalizacji, a dane geograficzne tego użytkownika są natychmiast aktualizowane w systemie.


RYSUNEK 8. MODUŁ PROFILE ZALOGOWANEGO UŻYTKOWNIKA.

## *Mapa*

Jak wyżej wspomniano wybierając z listy znajomych jednego z użytkowników wyświetlony zostanie ekran mapy z umiejscowionymi na niej pozycjami znajomych. (Rys. 9) Pozycja pojedynczego znajomego składa się ze zdjęcia w okręgu, bądź kolorowego kształtu z wielką literą stanowiącą pierwszą literę nazwy użytkownika. Istnieje możliwość przesuwania, zbliżania bądź oddalania mapy poprzez dotyk ekranu palcami. Na mapie widać wszystkich znajomych i ich ostatnio zaktualizowane lokalizacje.




RYSUNEK 9 EKRANY MAPY - PRZYBLIŻENIE I ODDALENIE

## *Ustawienia konta*

Ostatnim ekranem w aplikacji jest moduł *settings* odpowiadający za zarządzanie kontem. (Rys. 10) Posiada funkcję usunięcia konta – można to uczynić wybierając pierwszą opcję. Drugą funkcjonalnością jest zmiana hasła. Naciskając przycisk *change password* pokazują się bądź znikają pola służące zmianie hasła. Po wprowadzeniu starego hasła jednorazowo, a nowego dwukrotnie, należy dotknąć symbol wyboru: ✓. Jeśli pomyślnie zmieniono hasło do konta, zostanie wyświetlony odpowiedni komunikat.


**RYSUNEK 10. EKRAN USTAWIEŃ KONTA.**

# 5. Testowanie i napotkane problemy

Tworzenie każdego nowego systemu rozproszonego wiąże się z występowaniem problemów związanych z programowaniem, zarówno z punktu widzenia autora programu, jak i potencjalnego użytkownika. Również w przypadku aplikacji będącej przedmiotem niniejszej pracy wystąpiło kilka problemów związanych z różnymi aspektami programowania. Ten ostatni rozdział stanowi raport z testowania aplikacji w końcowym stadium implementacji. W rozdziale omówiono również kilka problemów, które udało się rozwiązać.

## 5.1 Testy

Implementowany system był testowany w trakcie tworzenia poszczególnych jego modułów. Stosowano proste testy funkcjonalności, których celem było sprawdzenie zachowywania się aplikacji i serwisu w wielu przypadkach, np. zapytań i pobierania obiektów z bazy danych. Zdecydowano o takim sposobie testowania ze względu na to, że skupiono się przy implementacji na aplikacji mobilnej, w której najważniejsze jest to, co widzi użytkownik.

Ponadto po wdrożeniu bazy danych oraz serwisu do platformy Azure w celach testowych korzystano z *RESTclient*. Oprogramowanie to stanowi wtyczkę do przeglądarki internetowej *Mozilla Firefox*, która pozwala na sprawne i szybkie wysyłanie zapytań do serwisu internetowego. Pozwoliło to na testowanie funkcjonalności już zewnętrznego serwisu zanim zaimplementowano je w ostatecznie w aplikacji.

Narzędziem, które także można skojarzyć z testowaniem aplikacji jest również użyty w projekcie *Google Analytics SDK*. Biblioteka pozwala na raportowanie błędów, wysyłanie zdarzeń, odczytywanie informacji o ilości aktywnych w danym momencie użytkowników oraz średniego czasu trwania sesji w aplikacji oraz wiele innych. Zdecydowano, że dodatkowo każde uruchomienie aplikacji będzie raportowane i zgłaszać będzie informację o czasie logowania i nazwę użytkownika, który się zalogował.

## 5.2 Rozwiązane problemy

### *Błąd bazy danych po wdrożeniu API do chmury*

W początkowej fazie projektu stworzono bazę danych lokalnie, aby opracować najpierw wszystkie potrzebne metody oraz zaimplementować konieczne kontrolery. Zamierzone cele osiągnięto, lecz po opublikowaniu w chmurze serwisu każde zapytanie zwracało błąd o charakterze nieokreślonym. Ostatecznie okazało się, że podczas publikacji należy dodatkowo zmienić dane dotyczące połączenia z bazą danych, więc nie wystarczy jedynie zrobić tego w pliku *Web.config*, lecz procedurę należy powtórzyć w

oknie, w którym kompilator sprawdza zmiany strony internetowej i określa parametry serwisu.

Ponadto podczas debugowania kontrolerów często występował błąd pochodzący z grupy dotyczącej bazy danych. Problem był niejasny, gdyż błędy występowały nieregularnie. Początkowo zakładano, że to problem z połączeniem z Internetem, lecz okazało się, że połączenie zdeklarowane w sekcji *Server Explorer* w *Visual Studio* z przyczyn nieokreślonych sporadycznie zostaje zerwane najprawdopodobniej, gdy okno środowiska było zbyt długi czas zminimalizowane. Należało przed każdym budowaniem projektu powtórzyć proces połączenia lub odświeżyć powyższą sekcję z połączeniami.

### *Brak dostępu do zasobów Okna Dialogowego*

Problem wystąpił podczas implementowania metody należącej do klasy *MainViewModel*. Podczas wykonywania metody *Load* aktualizowane są dane o użytkowniku, jego lokalizacja, pobierana jest lista znajomych, rozpoczyna się proces wysłania informacji do *Google Analytics* oraz uruchamia się metoda odpowiedzialna za wyświetlanie zaproszeń do grona znajomych. Co ważne, funkcje te są wykonywane asynchronicznie, jeśli zdefiniowane z przedrostkiem *async*. Problem zaistniał, gdy użytkownik otrzymał więcej niż jedno zaproszenie do listy znajomych.

Wyświetlenie zaproszenia wiąże się z wyświetleniem okna dialogowego z przyciskami *accept* oraz *reject*. Podczas tworzenia metody wyświetlania tej informacji popełniono błąd, którego nie zgłosił kompilator, gdyż składniowo kod był poprawny. Gdy zaproszenie było wyświetlane, następne w kolejce *próbowało* wywołać tą samą metodę *ShowAsync*, tego samego obiektu *MessageDialog* z innymi parametrami, tj. innym tekstem. System z takimi zadaniami powinien sobie poradzić bez problemu wyłącznie, jeśli metodę zdeklarujemy jako *async Task*, a nie jako *async void*.

Błędnie zdefiniowana metoda:

```
private async void ShowInvitation(Invitation i);
```

Poprawnie zdefinowana metoda:

```
private async Task ShowInvitation(Invitation i);
```

## *Konwerter służący do interpretacji właściwości*

Podczas projektowania interfejsu użytkownika zakładano występowanie kontrolek, które odpowiadają za informowanie o działających procesach. Do tych elementów należy kontrolka typu *ProgressBar* oraz *TextBlock* wyświetlający statycznie napis *Loading*. Wykorzystując mechanizm bindowania zakładano, że właściwość owych elementów interfejsu użytkownika odpowiedzialna za widoczność ma postać typu *boolean*, gdyż są widoczne lub nie. W przypadku bindowania kontrolki w języku *XAML* nie otrzymuje się komunikatu, jeśli występuje konflikt typów. Przy testowaniu okazało się, że kontrolki nie wyświetlają się wcale.

Rozwiązanie problemu znaleziono na jednej ze stron internetowych dotyczących problemów programistycznych. Należało utworzyć klasy typu Konwerter. Następnie należało je zdeklarować w pliku *ConvertersDefinitions*, i dodać plik jako zasób typu *resource dictionary*. Po wykonaniu tych czynności edytując interfejs można się już odwołać do klas typu Konwerter.

Konwerter, który zastosowano to klasa, która implementuje interfejs *IvalueConverter*. Zbudowana jest z dwóch metod o nazwach: *Convert* oraz *ConvertBack* Metody należące do klas typu konwerter mają wiele zastosowań przy implementacji interfejsu użytkownika. Przy niniejszym projekcie wykorzystano również konwerter do interpretacji obiektu typu *DateTime,* a także przy wyświetlaniu elementów należących do listy znajomych. W tym drugim przypadku zastosowano konwersję *Loginu* użytkownika na pierwszą jego literę, którą wykorzystuje się przy grafice występującej, gdy znajomy nie posiada zdjęcia profilowego przypisanego do swojego konta.

# Podsumowanie

Przedmiotem niniejszej pracy była implementacja systemu złożonego z trzech modułów. Głównym modułem jest aplikacja społecznościowa zaimplementowana dla systemu operacyjnego Windows Phone 8.1. Następnym modułem jest interfejs komunikacyjny stworzony w technologii *ASP.NET* oraz baza danych stanowiąca moduł przechowujący wszelkie, niezbędne dane dla aplikacji.

System jest zgodny z wszystkimi założeniami, które zakładano podczas planowania projektu. Spełnia zamierzone funkcjonalności i posiada szereg zabezpieczeń przed występowaniem błędów. W systemie można stworzyć konto, wykreować swój wizerunek przez opis oraz zdjęcie. Ponadto można stworzyć sieć znajomych oraz śledzić ich lokalizacje. Aplikację można w prosty sposób rozszerzyć o dodatkowe funkcjonalności jak np. komunikator umożliwiający konwersacje między użytkownikami aplikacji. System implementowano w oparciu o nowoczesne technologie takie jak *ASP.NET,* platformę *Azure,* czy *Windows Phone Store,* co pozwala na dalszy komercyjny rozwój niniejszego oprogramowania i zaistnienie na rynku współczesnych aplikacji mobilnych. Biorąc pod uwagę kontynuację tworzenia owego oprogramowania istnieje możliwość zintegrowania aplikacji z innymi systemami, bądź aplikacjami, a ponadto opublikowania aplikacji w sklepie – *Windows Store*, w którym można pobierać oraz kupować aplikacje. Projekt był tworzony ze szczególną starannością i zachowaniem w nim powtarzających się reguł kodowania zgodnych ze standardami pracy w zespole, co znaczy, że mógłby być kontynuowany przez innego programistę.

Podsumowując, osiągnięto cel projektu. Oprogramowanie może służyć jako aplikacja społecznościowa do wyszukiwania wzajemnie lokalizacji użytkowników. Efektem pracy jest poszerzenie umiejętności i wiedzy autora projektu z zakresu wykorzystanych narzędzi i bibliotek.

# Literatura


[1] Dokumentacja standardów interfejsu użytkownika, dostępna pod adresem: https://developer.microsoft.com/en-us/windows/apps/design (styczeń 2017r.)

[2] Dokumentacja środowiska programistycznego, dostępna pod adresem: https://msdn.microsoft.com/pl-pl/library/dd831853.aspx (styczeń 2017r.)

[3] Dokumentacja *.NET*, dostępna pod adresem: https://www.microsoft.com/net/tutorials/csharp/getting-started (styczeń 2017r.)

[4] Dokumentacja *ASP.NET*, dostępna pod adresem: http://www.centrumxp.pl/dotNet/142,01-ASPNET-co-to-wlasciwie-jest.aspx (styczeń 2017r.)

[5] Andrew Troelsen, *Język C# 2008 i platforma .NET 3.5*, PWN, Warszawa 2009

[6] Strona z opisem *GIT*, dostępna pod adresem: https://git-scm.com/about (styczeń 2017r.)

[7] Opis *Entity Framework*, dostępny pod adresem: http://reset.ath.bielsko.pl/technologienet/artykuly/csharp/2012/wprowadzenie-do-entity-framework-adonet.aspx (styczeń 2017r.)

[8] Strona portalu *Azure*, dostępna pod adresem: https://portal.azure.com/ (styczeń 2017r.)

[9] Dokumentacja wzorca architektonicznego *MVVM*, dostępna pod adresem: https://msdn.microsoft.com/pl-pl/library/hh848247.aspx (styczeń 2017r.)

[10] Przewodnik do tworzenia serwisów typu *RESTfull*, dostępny pod adresem: https://www.asp.net/web-api/overview/older-versions/build-restful-apis-with-aspnet-web-api (styczeń 2017r.)

[11] Opis kodów protokołu *http*, dostępny pod adresem: https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html (styczeń 2017r.)

[12] Patryk Osowski, Bindowanie danych w *WPF*, dostępny pod adresem: http://patryknet.blogspot.com/2010/07/bindowanie-danych-w-wpf-cz-1.html (styczeń 2017r.)

[13] Dokumentacja biblioteki *RestSharp*, dostępna pod adresem: https://github.com/restsharp/RestSharp/wiki (styczeń 2017r.)

[14] Strona do pobrania *SKD for Windows 8.1,* dostępna pod adresem: https://developer.microsoft.com/en-us/windows/downloads/windows-8-1-sdk (styczeń 2017r.)

# Załącznik A. Skrypt tworzący bazę danych

```
SET QUOTED_IDENTIFIER OFF;
GO
USE [project18dataBase];
GO
IF SCHEMA_ID(N'dbo') IS NULL EXECUTE(N'CREATE SCHEMA [dbo]');
GO

IF OBJECT_ID(N'[dbo].[UserSet]', 'U') IS NOT NULL
    DROP TABLE [dbo].[UserSet];
GO
IF OBJECT_ID(N'[dbo].[InvitationSet]', 'U') IS NOT NULL
    DROP TABLE [dbo].[InvitationSet];
GO

-- Creating table 'UserSet'
CREATE TABLE [dbo].[UserSet] (
    [Id] int IDENTITY(1,1) NOT NULL,
    [Login] nvarchar(25)  NOT NULL,
    [Password] nvarchar(max)  NOT NULL,
    [LastVisit] nvarchar(max)  NOT NULL,
    [OpenDate] nvarchar(max)  NOT NULL,
    [CloseDate] nvarchar(max)  NOT NULL,
    [Desc] nvarchar(120)  NOT NULL,
    [PhotoUri] nvarchar(max)  NOT NULL,
    [X] float  NOT NULL,
    [Y] float  NOT NULL
);
GO

-- Creating table 'InvitationSet'
CREATE TABLE [dbo].[InvitationSet] (
    [Id] int IDENTITY(1,1) NOT NULL,
    [SenderId] int  NOT NULL,
    [RecieverId] int  NOT NULL,
    [Status] nvarchar(20)  NOT NULL
);
GO

-- Creating primary key on [Id] in table 'UserSet'
ALTER TABLE [dbo].[UserSet]
ADD CONSTRAINT [PK_UserSet]
    PRIMARY KEY CLUSTERED ([Id] ASC);
GO

-- Creating primary key on [Id] in table 'InvitationSet'
ALTER TABLE [dbo].[InvitationSet]
ADD CONSTRAINT [PK_InvitationSet]
    PRIMARY KEY CLUSTERED ([Id] ASC);
GO
```

# Załącznik B. Fragmenty kodu źródłowego

## B1. Klasa UserController.cs

```
public class UsersController : ApiController
    {
        private DataModelContainer db = new DataModelContainer();
        // POST: api/Users, metoda tworząca użytkownika
        [ResponseType(typeof (User))]
        public IHttpActionResult PostUser(User user)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }
            try
            {
                var users = db.UserSet.Where
                    (u => u.Login == user.Login).ToList();
                if (users.Any())
                {
                    return Unauthorized();
                }
              db.UserSet.Add(user);
                db.SaveChanges();
                return Ok(user);
            }
            catch (Exception ex)
            {
                return Unauthorized();
            }
        }
        // DELETE: api/Users/5, metoda usuwająca użytkowanika z bazy
        [ResponseType(typeof(User))]
        public IHttpActionResult DeleteUser(int id)
        {
            User user = db.UserSet.Find(id);
            if (user == null)
            {
                return NotFound();
            }

            db.UserSet.Remove(user);
            db.SaveChanges();

            return Ok(user);
        }
        protected override void Dispose(bool disposing)
        {
            if (disposing)
            {
                db.Dispose();
            }
            base.Dispose(disposing);
        }
    }
```

# B2. Metoda FriendsController.GetFriends

```
[ResponseType(typeof(IEnumerable<User>))]
public IHttpActionResult GetFriends(int id)
{
    try
    {
        using (var db = new DataModelContainer())
        {
            var invitations = db.InvitationSet .Where(invitation =>
            (invitation.RecieverId == id || invitation.SenderId == id)
             && invitation.Status.Contains("Accepted")).ToList();
            var friends = new List<User>();
            foreach (var i in invitations)
            {
                if (i.SenderId == id) friends.Add(db.UserSet.Find(i.RecieverId));
                else friends.Add(db.UserSet.Find(i.SenderId));
            }
            var sortedFriends = (from f in friends
                where f.CloseDate > DateTime.Now select new User
                {
                    Id = f.Id,
                    Login = f.Login,
                    PhotoUri = f.PhotoUri,
                    LastVisit = f.LastVisit,
                    Desc = f.Desc,
                    X = f.X,
                    Y = f.Y,
                }).ToList();
            return Ok(sortedFriends);
        }
    }
    catch(Exception ex)
    {
        return InternalServerError(ex);
    }
}
```

# B3. Klasa bazowa typu ViewModel

```
public class TillsammensViewModelBase : ViewModelBase
    {
        protected readonly IDialogService DialogService;
        protected readonly ITillsammensService TillsammensService;
        private bool _isWorking;

         public TillsammensViewModelBase(IDialogService dialogService,
         ITillsammensService tillsammensService)
        {
            DialogService = dialogService;
            TillsammensService = tillsammensService;
        }
        public bool IsWorking
        {
            get { return _isWorking; }
            set { Set(ref _isWorking, value); }
        }
        protected void ShowWebResultCommunicate(WebServiceStatus status)
        {
            string message = null;
            switch (status)
            {
                case WebServiceStatus.ConnectionError:
                    message = "Service is not available.";
                    break;

                case WebServiceStatus.ServiceError:
                    message = "Internal server error.";
                    break;


                case WebServiceStatus.Unauthorized:
                    message = "Wrong login or password.";
                    break;
            }
            DialogService.ShowMessageBox(message, string.Empty);
        }
        protected void StartWorking()
        {
            IsWorking = true;
            MessengerInstance.Send(new ShowTopProgressBarMessage(true));
        }
        protected void StopWorking()
        {
            IsWorking = false;
            MessengerInstance.Send(new ShowTopProgressBarMessage(false));
        }
    }
```

# B4. Klasa RestClientBase.cs

```
public abstract class RestClientBase
    {
        protected RestClient;

        protected RestClientBase(string baseAddress)
        {
            RestClient = new RestClient(baseAddress);
        }

        protected async Task<TResponse> CallAsync<TResponse>(IRestRequest request)
        {
            IRestResponse response;
            try
            {
                response = await RestClient.Execute(request);
            }
            catch (Exception ex)
            {
                if (ex.Message.Contains("Unauthorized"))
                {
                    throw new WebServiceException(ex, WebErrorStatus.Unauthorized);
                }
                if (ex.Message.Contains("InternalServerError"))
                {
                     throw new WebServiceException(ex, WebErrorStatus.InternalServerError);
                }
                throw new WebServiceException(ex, WebErrorStatus.HostNameNotResolved);
            }
            if (!response.IsSuccess &&
                Enum.IsDefined(typeof(WebErrorStatus), (int)response.StatusCode))
            {
                throw new WebServiceException((WebErrorStatus)response.StatusCode);
            }

            var resultString = Encoding.UTF8.GetString(response.RawBytes, 0,
                response.RawBytes.Length);
            try
            {
                var result = JsonConvert.DeserializeObject<TResponse>(resultString);
                return result;
            }
            catch (Exception)
            {
                throw new WebServiceException(WebErrorStatus.Unknown);
            }
        }
    }
```

# Zawartość CD

Do niniejszej pracy jest dołączona płyta CD, która zawiera:

- Tekst pracy w formatach *.docx*, *.pdf*.