Uniwersytet Mikołaja Kopernika
Wydział Fizyki, Astronomii i Informatyki Stosowanej
Katedra Informatyki Stosowanej

Paweł Stopiński
nr albumu: 244675

Praca magisterska
na kierunku Informatyka Stosowana

Aplikacje uniwersalne - mapy, geolokacja i bazy danych SQLite

Opiekun pracy dyplomowej
dr hab. Jacek Matulewski
Instytut Fizyki

Toruń 2015

Pracę przyjmuję i akceptuję

.....
data i podpis opiekuna pracy

Potwierdzam złożenie pracy dyplomowej

.....
data i podpis pracownika dziekanatu

Podziękowania dla kadry Wykładowców Wydziału Fizyki, Astronomii i Informatyki Stosowanej Uniwersytetu Mikołaja Kopernika w Toruniu za pełne kompetencji i zaangażowania przekazywanie wiedzy.

Za inspirację, opiekę merytoryczną i pomoc w nadaniu właściwego kształtu mojej pracy dziękuję promotorowi dr. hab. Jackowi Matulewskiemu.

Spis treści

1 Wstęp

1.1 Wprowadzenie

Zazwyczaj tworzenie aplikacji na różne platformy wymaga stworzenia niezależnych programów o zbliżonych funkcjonalnościach. Współdzielenie kodu źródłowego między wersjami na różne platformy, choć skracałoby czas tworzenia aplikacji, jest często ograniczone. Dla umożliwienia jednoczesnego tworzenia programów dla systemów operacyjnych Windows 8.1 oraz Windows Phone 8.1 została stworzona przez Microsoft architektura aplikacji Windows Runtime. Obsługuje ona szereg urządzeń, takich jak komputery osobiste, telefony oraz tablety. Architekturze Windows Runtime towarzyszy projekt typu Universal Apps, który pozwala na jednoczesne tworzenie aplikacji na systemy Windows 8.1 oraz Windows Phone 8.1 wraz ze współdzieleniem dużej ilości kodu i zasobów.

W pracy zostaną opisane istotne zagadnienia Aplikacji Uniwersalnych. W zestawieniu z innymi zagadnieniami opisanymi w pracach dyplomowych studentów Wydziału Fizyki Astronomii i Informatyki Stosowanej UMK Mateusza Dudka i Rafała Wołowskiego stanowiły będą praktyczny tutorial aplikacji uniwersalnych.

1.2 Cel pracy

Celem pracy jest zobrazowanie istotnych zagadnień w programowaniu aplikacji uniwersalnych przy użyciu projektu typu Universal Apps. Praca ta stanowi podstawę do łatwej nauki stosowania w praktyce w Universal Apps powiązanych zagadnień takich jak:

- przechowywanie danych w bazach SQLite,
- umieszczanie map w aplikacji,
- używanie geolokacji,
- prezentacja danych.

W pracy została opisana metoda działania zgodna z definicją ujętą w *„Traktacie o dobrej robocie”* prof. Tadeusza Kotarbińskiego: *„Metoda, czyli system postępowania, jest to sposób wykonywania czynu złożonego, polegający na określonym doborze i układzie jego*

działań składowych, a przy tym uplanowiony i nadający się do wielokrotnego stosowania"[1].

Pracy towarzyszy aplikacja *Delivery Helper*, która obrazuje większość opisanych zagadnień.

[1] Kotarbiński Tadeusz, *Traktat o dobrej robocie*, Zakład Narodowy im. Ossolińskich Wydawnictwo Polskiej Akademii Nauk, Wrocław-Warszawa-Kraków-Gdańsk-Łódź 1982, str. 79

2 Windows Runtime

2.1 Wstęp

Wraz z premierą systemu operacyjnego Windows 8 Microsoft wprowadził architekturę aplikacji Windows Runtime. Celem było zbliżenie do siebie dwóch platform Windows: przeznaczonej na komputery osobiste oraz na telefony. Windows Runtime umożliwia tworzenie aplikacji Windows apps, znanych wcześniej także pod nazwami Metro, Modern UI oraz Windows Store apps. Aplikacje mogą być tworzone na systemy Windows 8 i 8.1, Windows Phone 8.1 oraz Windows RT[2]. W przypadku Windows 10 Windows Runtime został przekształcony w Universal Windows Platform, który składa się ze wspólnego API oraz API dedykowanych poszczególnym typom urządzeń. Dostępne platformy docelowe to komputery typu desktop, telefony i tablety, Konsole Xbox, tablice interaktywne Surface Hub oraz urządzenia IoT[3], takie jak Arduino oraz Raspberry Pi 2.

Dla urządzeń Windows Phone 8 nieprzystosowanych do działania z wersją 8.1 przygotowano aktualizację pod nazwą Windows Phone Silverlight 8.1 z częściową kompatybilnością. Pełna zbieżność nastąpiła dopiero w wersjach Windows 8.1 i Windows Phone 8.1.

Windows Runtime jest zarazem następcą, jak i rozszerzeniem API Win32. W praktyce WinRT opakowuje Win32, lecz wprowadza także całkiem nowe API[4]. Przy jego użyciu aplikacje można tworzyć w językach programowania takich jak C#, Visual Basic, C++/CX, JavaScript oraz TypeScript. WinRT został napisany w języku C++ i jest zorientowany obiektowo.

[2] Windows RT jest edycją Windows 8.1 przeznaczoną na urządzenia z procesorami o architekturze ARM. Nie należy mylić systemu Windows RT z architekturą Windows Runtime, której dotyczy powyższy rozdział. Windows Runtime jest nazywany w skrócie WinRT.

[3] Internet of Things - *internet rzeczy* - idea urządzeń wbudowanych podłączonych do internetu, dzięki czemu mogą się komunikować w celu przetwarzania danych. Urządzeniami typu IoT mogą być czujniki, transmitery, systemy kontrolne itp.

[4] Zob. L. Osterman, *http://stackoverflow.com/questions/7475775/winrt-and-build-in-windows-8-apps#comment9135316_7476222*, [dostęp 20 października 2015]

2.2 Sandbox

W celu zapewnienia większego bezpieczeństwa wszystkie aplikacje stworzone w architekturze Windows Runtime są uruchamiane w tak zwanej piaskownicy (z ang. *sandbox*) - zamkniętym środowisku. Dzięki temu aplikacje są odseparowane od siebie oraz nie mają bezpośredniego dostępu do plików oraz zasobów systemowych. Domyślnie programy mogą jedynie zapisywać oraz czytać z plików należących do nich. Aby aplikacja mogła używać bibliotek multimediów takich jak muzyka, obrazy czy wideo musi w jawny sposób zadeklarować tę funkcjonalność w App package manifest, który opisuje wszystkie jej możliwości i zasoby, których wymaga ona do działania. Do funkcjonalności, które aplikacja może zadeklarować należy dostęp do:

- bibliotek multimediów: muzyki, obrazów, wideo,
- plików z zewnętrznych urządzeń,
- połączenia internetowego lub sieci lokalnej,
- kalendarza spotkań,
- kontaktów,

oraz urządzeń takich jak:

- GPS,
- mikrofon,
- kamera,
- Bluetooth,
- WiFi.

Podczas instalacji lista uprawnień aplikacji jest przedstawiana użytkownikowi. Dodatkowo wszystkie aplikacje publikowane w Windows Store są sprawdzane pod kątem bezpieczeństwa podczas procesu certyfiakcji.

2.3 Asynchroniczność

Windows Runtime kładzie nacisk na płynne działanie interfejsu użytkownika. W celu wyeliminowania spowolnień oraz maksymalizacji szybkości reakcji interfejsu użytkownika, programistom zostały udostępnione narzędzia ułatwiające pisanie asynchronicznych aplikacji. Samo API zostało także zaprojektowane w taki sposób, aby nie wstrzymywało działania aplikacji w wypadku długo wykonujących się fragmentów kodu. Domyślnie wszystkie metody, których wykonanie może trwać więcej niż 50 milisekund są zaimplementowane jako asynchroniczne, dzięki czemu nie powodują blokowania. Są one oznaczone poprzez zastosowanie przyrostka `Async`. Synchroniczne wersje metod nie są dostępne, dzięki temu zapewnione jest poprawne ich wykorzystanie. w WinRT metody asynchroniczne dostępne są w API dotyczących:

- dostępu do sieci (klasy `HttpClient`, `SyndicationClient`),
- dostępu do plików (klasy `StorageFile`, `StreamWriter`, `StreamReader`, `XmlReader`)
- pozyskiwania oraz obróki obrazów (klasy `MediaCapture`, `BitmapEncoder`, `BitmapDecoder`),
- Windows Communication Foundation.

3 Universal Apps

3.1 Wstęp

Universal Apps to typ aplikacji wprowadzony przez Microsoft. Zbudowane one są w architekturze Windows Runtime. Solucje aplikacji Universal składają się z trzech projektów:

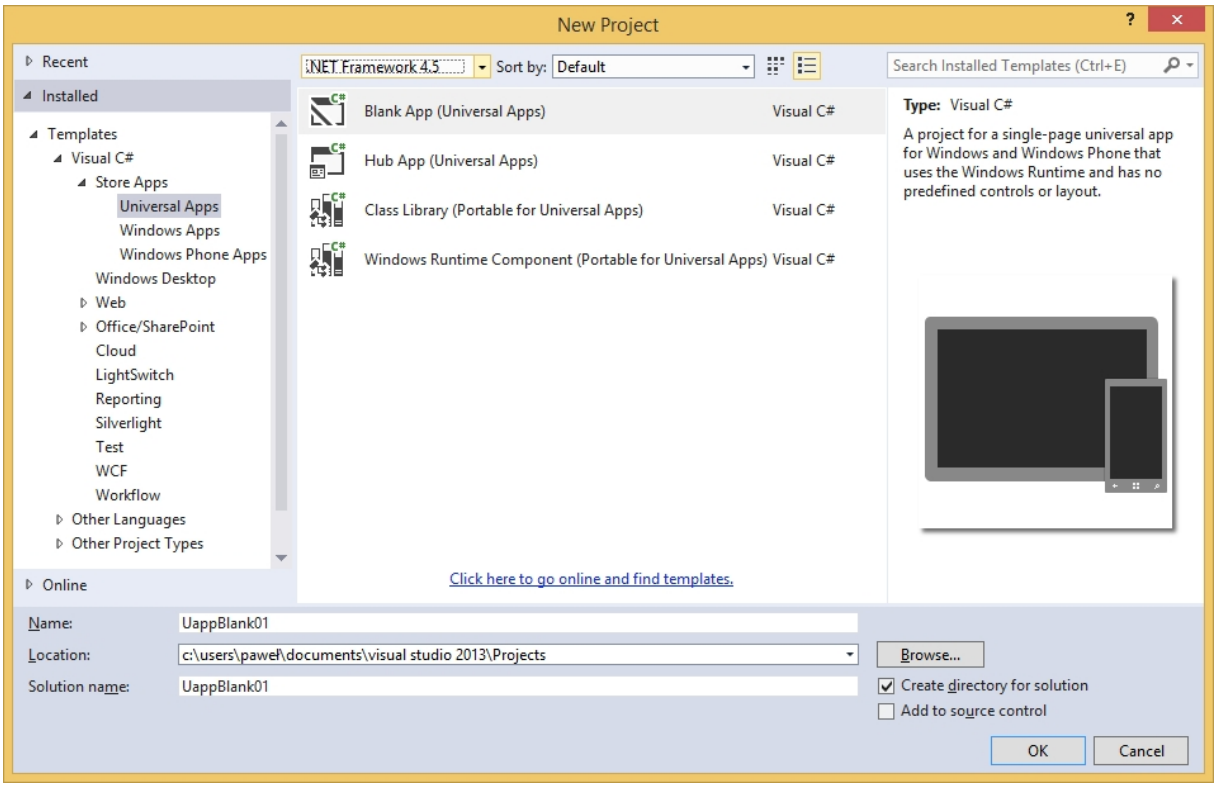
- Windows 8.1 - zawierający elementy specyficzne dla systemu w wersji desktop,
- Windows Phone 8.1 - jak wyżej, lecz dla wersji na urządzenia mobilne,
- Shared - projekt zawierający wspólny, niezależny od platformy kod, np. model aplikacji.

Typowym podejściem przy projektowaniu programu jest wydzielenie logiki aplikacji i umieszczenie jej w projekcie współdzielonym. Części specyficzne dla plaformy zawierają wtedy jedynie elementy związane z prezentacją, np. interfejsy użytkownika. W wyniku kompilacji tworzone są dwa niezależne pliki wykonywalne na obie docelowe platformy. W obu zawarty jest kod z projektu Shared.

3.2 Tworzenie projektu Universal Apps

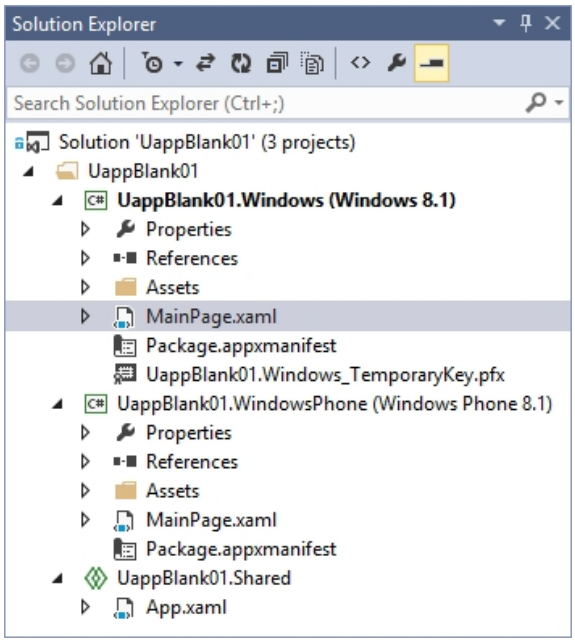
Tworzenie aplikacji Universal Apps jest możliwe w środowisku Visual Studio w wersji 2013 Update 2 bądź nowszej.

Poniżej opisano procedurę tworzenia projektu aplikacji na przykładzie języka Visual C#. W menu *File* należy wybrać opcję *New*, a następnie *Project...*, po czym w oknie *New Project* w lewym panelu wybieramy pozycję znajdującą się w drzewie kolejno: *Installed*, *Templates*, *Visual C#*, *Store Apps*, *Universal Apps*. W głównym panelu należy wybrać pozycję *Blank App* by stworzyć szkielet pustego projektu Universal App. W polu *Name* wpisujemy wybraną nazwę projektu. Dodatkowo możemy wybrać lokalizację, w której zostanie on stworzony poprzez kliknięcie przycisku *Browse...* obok pola *Location*. Powyższy proces został zobrazowany na Rys. 1.



Rys. 1: Okno tworzenia projektu.

Na poniższej ilustracji widoczne są trzy projekty wspomniane w punkcie (3.1), czyli *Windows 8.1*, *Windows Phone 8.1* oraz *Shared*.



Rys. 2: Solution Explorer.

4 SQLite w Universal Apps

4.1 Bazy danych SQLite

SQLite to biblioteka pozwalająca na tworzenie, odczyt i edycję relacyjnych baz danych. Nie wymaga dużej ilości pamięci operacyjnej, dzięki czemu może być stosowany w urządzeniach mobilnych, takich jak telefony, odtwarzacze MP3 oraz systemy wbudowane. Istotną cechą bazy SQLite jest to, że nie tworzy osobnego trwałego procesu, który pośredniczyłby w wymianie danych. Wymaga jedynie minimalnego wsparcia ze strony systemu operacyjnego oraz zewnętrznych bibliotek. Całość logiki działania bazy danych zawarta jest w samej aplikacji, zaś baza danych mieści się w pojedynczym pliku, którego format jest identyczny dla różnych platform. Ponadto biblioteka SQLite jest udostępniana na zasadach domeny publicznej, zatem możliwe jest nieodpłatne kopiowanie, modyfikowanie i publikowanie aplikacji z niej korzystających. Kod źródłowy, napisany w języku C, jest także dostępny publicznie. Biblioteka SQLite została zaprojektowana tak, aby nie wymagała wstępnej konfiguracji. Do jej działania nie jest potrzebny plik konfiguracyjny ani modyfikacja ustawień systemu[5].

Implementacja SQL

SQLite implementuje większość elementów składni języka SQL. Do funkcjonalności SQL niedostępnych w SQLite należą między innym złączenia typu *Right Outer* oraz *Full Outer*, możliwość zapisywania do widoków. Wsparcie dla wyzwalaczy oraz polecenia *Alter Table* jest ograniczone[6].

Zgodność z ACID

SQLite zapewnia zgodność z wzorcem ACID, który gwarantuje poprawność i spójność bazy danych. Główne założenia ACID to:

[5] Zob. *SQLite Is A Zero-Configuration Database*, *https://www.sqlite.org/zeroconf.html* [dostęp 20 października 2015]

[6] Zob. *SQL Features That SQLite Does Not Implement*, *https://www.sqlite.org/omitted.html* [dostęp 20 października 2015]

- *Atomicity* - kilka kolejnych operacji na bazie danych tworzy tak zwaną transakcję. Jeśli w trakcie jej przetwarzania wystąpi błąd, to cała transakcja jest anulowana.
- *Consistency* - zapewnienie, że w wyniku wykonania transakcji otrzymany stan bazy będzie poprawny, zaś wszystkie ograniczenia spełnione.
- *Isolation* - właściwość ta zapewnia, że wynik działania transakcji wykonujących się równolegle jest taki sam, jak w przypadku wykonywania serializowanego.
- *Durability* - wynik działania transakcji zostanie zachowany nawet w przypadku awarii systemu. Zanim nastąpi trwały zapis operacje nie mogą zostać uznane za sfinalizowane.

4.2 Wymagane pakiety i rozszerzenia

W przedstawionym poniżej podejściu organizacji solucji, kod bazy danych zawarty jest w oddzielnym projekcie typu Portable Class Library. Dodać go do solucji możemy poprzez menu *File>Add >New Project...*, gdzie z kategorii *Installed >Visual C# >Store Apps >Universal Apps* wybieramy typ projektu *Class Library (Portable for Universal Apps)*. Na potrzeby tego przykładu projekt nazwany będzie *Database*.

Aby użyć SQLite w aplikacji Windows Runtime konieczne jest:

- zainstalowanie rozszerzeń *Precompiled Binaries for Windows Phone 8 (sqlite-wp81-winrt-30081002.vsix)* i *Precompiled Binaries for Windows Runtime (sqlite-winrt81-30081002.vsix)*. Można je znaleźć na oficjalnej stronie SQLite: *https://www.sqlite.org/download.html*. Po ich instalacji niezbędne jest ponowne uruchomienie Visual Studio.
- dodanie referencji do pobranych rozszerzeń do projektów Windows 8.1 i Windows Phone 8.1. Klikając prawym klawiszem myszy na *References*, wybieramy opcję *Add reference...*, po czym w kategorii *Windows 8.1* bądź *Windows Phone 8.1 > Extensions* zaznaczamy pole wyboru przy pozycji *SQLite for Windows (Phone) 8.1*. Operację musimy wykonać osobno dla projektów dla obu platform.
- zmodyfikowanie ustawień platformy docelowej. Polega to na wybraniu w panelu

Configuration Manager dostępnym w menu *Build* architektur procesorów. Rozszerzenia nie obsługują opcji *Any CPU*, zatem konieczne jest wybranie w polu *Platform* konkretnej architektury, np. *x86*. Czynność tę należy wykonać dla każdego projektu widocznego na liście.

- dodanie do projektów odpowiednich pakietów. Możemy to zrobić w menedżerze: *Tools >NuGet Package Manager >Manage NuGet Packages for Solution...*. W kategorii *Online >nuget.org* należy wyszukać i zainstalować pakiet *SQLite.Net Extensions-PCL*. Powyższy pakiet nie jest niezbędny, lecz udostępnia dodatkowe funkcjonalności. Instalacja zależności, w tym *SQLite.Net PCL* następuje automatycznie.
- dodanie referencji do projektu *Database* do projektów *Windows (Phone) 8.1*.

4.3 Tworzenie modelu bazy danych

Pierwszym krokiem do stworzenia działającej bazy danych jest zdefiniowanie klas, których obiekty będziemy zapisywać w bazie. W przykładzie zawartym w tej pracy zostanie stworzony model zawierający dwie klasy: `Client` oraz `Delivery`.

Listing 1: Kod klasy Client.

```
using System;
using System.Collections.Generic;
using SQLiteNetExtensions.Attributes;
using SQLite.Net.Attributes;

namespace SQLiteExample
{
    [Table("Clients")]
    public class Client
    {
        [PrimaryKey, AutoIncrement]
        public int IdClient { get; set; }
        public string Name { get; set; }
        public string Address { get; set; }

        [OneToMany(CascadeOperations = CascadeOperation.All)]
        public List<Delivery> Deliveries { get; set; }

        public Client()
        {
            Deliveries = new List<Delivery>();
        }
    }
}
```

Listing 2: Kod klasy Delivery.

```
using System.Collections.Generic;
using SQLiteNetExtensions.Attributes;
using SQLite.Net.Attributes;
```

```
4
5  namespace SQLiteExample
6  {
7      public enum State { Active, Delayed, Completed };
8
9      [Table("Deliveries")]
10     public class Delivery
11     {
12         [PrimaryKey, AutoIncrement]
13         public int IdDelivery { get; set; }
14         public DateTime Date { get; set; }
15         public State State { get; set; }
16
17         [ForeignKey(typeof(Client))]
18         public int IdClient { get; set; }
19
20         [ManyToOne(CascadeOperations = CascadeOperation.CascadeRead
               | CascadeOperation.CascadeInsert)]
21         public Client Client { get; set; }
22     }
23 }
```

Definicja tabeli

Atrybutem `Table` wskazujemy, że obiekty tej klasy będą przechowywane w tabeli o określonej nazwie.

Określanie klucza głównego

Aby wskazać pole, które będzie użyte jako klucz główny dla tabeli, poprzedzamy je atrybutem `PrimaryKey`. Ponadto możemy dodać funkcjonalność jego autoinkrementacji, dzięki czemu nie będzie konieczne dbanie o poprawną wartość klucza z poziomu kodu. Możemy to uczynić dodając atrybut `AutoIncrement`. *SQLite.Net* obecnie nie wspiera kluczy złożonych.

Definicje relacji

Dzięki rozszerzeniu *SQLite.Net Extensions* możemy bezpośrednio w kodzie definiować relacje pomiędzy tabelami. W przypadku relacji typu „jeden-do-wielu" w klasie po stronie „jeden" (`Client`) używamy atrybutu `OneToMany`. Pole, do którego odnosi się ten atrybut będzie uzupełniane odpowiednimi obiektami, które znajdują się w relacji w momencie pobierania obiektu z bazy. Aby działo się to automatycznie należy określić odpowiednią operację kaskadową. Opis ich działania znajduje się w następnym punkcie. Pole

to musi być typu `List` lub `Array`.

Po drugiej stronie relacji, tj. „wiele" musimy zdefiniować pole, które jest kluczem obcym za pomocą atrybutu `ForeignKey`. Jako parametr podajemy typ do którego obiektu będzie odnosił się klucz. Możliwe jest zdefiniowanie pola, które będzie referencją na obiekt znajdujący się po drugiej stronie relacji, tj. „jeden". W podanym przykładzie w klasie `Delivery` jest to pole typu `Client`. Pole te musi być opatrzone atrybutem `ManyToOne`. Podobnie jak w poprzednim przypadku możliwe jest określenie operacji kaskadowych.

Warto zaznaczyć, że pola, które odnoszą się do relacji nie są zapisywane jako kolumny w bazie. Wypełniane są one podczas pobierania rekordów z bazy na podstawie kluczy obcych.

SQLite.Net Extensions pozwala tworzyć także relacje typu „wiele do wielu". Wymaga to manualnego stworzenia tabeli pośredniej. Powinny się w niej znaleźć, oprócz klucza głównego, dwa klucze obce (atrybut `ForeignKey`). W obu tabelach znajdujących się w relacji „wiele do wielu" muszą być umieszczone pola typu `List` opatrzone atrybutem `ManyToMany` wraz z typem klasy definiującej tabelę pośrednią.

Dodatkowo możliwe jest określenie operacji kaskadowych, które są wykonywane podczas operacji na rekordach. Możliwe opcje to:

- usuwanie - `CascadeDelete` - podczas usuwania klienta z bazy zostaną usunięte wszystkie jego dostawy,
- wstawianie - `CascadeInsert` - jeśli stworzymy nowego klienta i utworzymy przypisane mu dostawy, to przy dodawaniu klienta do bazy one także zostaną dodane,
- wczytywanie - `CascadeRead` - gdy wczytamy z bazy klienta, to lista przypisanych mu dostaw również zostanie pobrana i umieszczona w odpowiednim polu.

Operacje kaskadowe można łączyć za pomocą operacji bitowych jak zademonstrowano w przykładowym kodzie źródłowym - listing 2, linia 20.

Inicjalizacja bazy danych

Baza danych SQLite przechowywana jest w lokalnym pliku binarnym. Obsługa bazy, w tym jej tworzenie, wykonywane jest przy użyciu obiektu klasy `SQLiteConnection`.

Obiekt działa także jako interfejs do bazy i musi być powiązany z konkretnym plikiem którego ścieżka jest określana w konstruktorze. Jeśli podany plik nie istnieje, zostanie utworzony. Tworzenie plików na platformie Windows Runtime zalecane jest w folderze przeznaczonym dla danej aplikacji. Obiekt `SQLiteConnection` podczas tworzenia przyjmuje jako argument obiekt implementujący interfejs `ISQLitePlatform`. W przypadku aplikacji Windows Runtime należy przekazać konstuktorowi obiekt klasy `SQLitePlatformWinRT`.

Posługując się obiektem `SQLiteConnection`, określa się tabele jakie mają znaleźć się w bazie. Służy do tego metoda `CreateTable`. Tabele zostają nazwane zgodnie z atrybutem `Table` znajdującym się w definicji klasy. Jeśli dana tabela już istnieje, nie zostaje podjęta żadna akcja, a dane zostają zachowane. Dzięki temu bezpieczne jest ponowne wykonanie tej metody bez groźby utraty zapisanych informacji. Tabele mogą zostać dodane do istniejącej bazy w ten sam sposób.

Listing 3: Tworzenie bazy danych.

```
1  using System;
2  using System.Collections.Generic;
3  using System.IO;
4
5  using SQLite.Net;
6  using SQLite.Net.Platform.WinRT;
7  using Windows.Storage;
8
9  namespace SQLiteExample
10 {
11     public class Database
12     {
13         public static string dbName = Path.Combine(
14             ApplicationData.Current.LocalFolder.Path,"Database.db");
15
16         public static void CreateDatabase()
17         {
18             SQLiteConnection sqliteConnection = new SQLite.Net.
                   SQLiteConnection(new SQLitePlatformWinRT(), dbName);
19             sqliteConnection.CreateTable<Delivery>();
20             sqliteConnection.CreateTable<Client>();
21         }
22     }
23 }
```

4.4 Używanie bazy danych

Podobnie jak w poprzednim punkcie, do obsługi bazy używany jest obiekt `SQLiteConnection`. Jego funkcjonalność jest rozszerzona przez pakiet *SQLite.Net Extensions*, który dodaje nowe wersje wielu metod.

Pobieranie obiektów z bazy danych

Do pobrania wszystkich elementów znajdujących się w danej tabeli służy metoda `GetAllWithChildren<Tabela>`. Pochodzi ona z rozszerzenia, zatem podczas wczytywania z bazy do każdego obiektu zostaną podpięte obiekty z jego relacji zgodnie z określonymi operacjami kaskadowymi. Dla przykładu do znajdujących się w obiektach klasy `Client` list `Deliveries` zostaną dodane referencje do dostaw przypisanych danym klientom. Do zawężenia wyników można wykorzystać wyrażenie Linq przekazane w argumencie metody. Do pobrania pojedynczego elementu o zadanym kluczu głównym służy metoda `GetWithChildren`. Pobieranie z bazy zostało zobrazowane w listingu 4 w liniach 1-2.

Dodawanie obiektów do bazy danych

Aby zapisać obiekt w bazie, należy wykorzystać metodę `InsertWithChildren`. Metoda autmatycznie rozpoznaje typ obiektu i dodaje go do odpowiedniej tabeli. Nowo stworzone obiekty, które znajdują się w relacji i zostaną wpisane do odpowiednego pola również zostaną dodane, jeśli określona jest operacja kaskadowa `CascadeInsert`. Pozwala to np. stworzyć nowego klienta oraz kilka przypisanych mu dostaw i jednym wywołaniem dodać wszystkie obiekty do bazy.

Dodatkową metodą pozwalającą na dodanie całej kolekcji obiektów jest `InsertAllWithChildren`. Powyższe uwagi również odnoszą się do tej metody. Dodawanie obiektów do bazy danych zobrazowano w listingu 4 w liniach 4-6.

Aktualizacja rekordów

Rekordy możemy pobrać z bazy w postaci obiektów i je zmodyfikować. Do zapisania zmian służy metoda `InsertOrReplaceWithChildren`. Podobnie jak w poprzednich punktach zmiany są odzwierciedlane zarówno w obiektach, które są bezpośrednio aktualizowane, ale także w obiektach których referencje są w nich zapisane. W wypadku gdy obiekt nie jest obecny w bazie, zostaje do niej dodany. Analogicznie do poprzednich punktów, do aktualizacji kolekcji obiektów służy metoda `InsertOrReplaceAllWithChildren`. Aktualizacja rekordów została zobrazowana w listingu 4 w liniach 8-13.

Listing 4: Pobieranie, dodawanie i modyfikowanie rekordów.

```
SQLiteConnection sqliteConnection = new SQLiteConnection(new
    SQLitePlatformWinRT(), Database.dbName);
var ClientsA = sqliteConnection.GetAllWithChildren<Client>(x => x.
    Name.StartsWith("A"));

Client Newton = new Client() { Name = "Newton" };
Newton.Deliveries.Add(new Delivery());
sqliteConnection.InsertWithChildren(Newton);

foreach (Client C in ClientsA)
{
    C.Name += " Modified";
    C.Deliveries.Add(new Delivery());
}
sqliteConnection.InsertOrReplaceAllWithChildren(ClientsA);
```

Wykonywanie instrukcji SQL

Pakiet *SQLite.Net* umożliwia wykonywanie instrukcji SQL. Możemy je podzielić na dwie grupy: takie, które zwracają rekordy (zapytania *select*) oraz takie, które modyfikują bazę (na przykład polecenia *insert*, *update* i *delete*).

Zapytania zwracające rekordy można wykonać przy użyciu metody `Query` klasy `SQLiteConnection`, gdzie argumentem jest kod SQL. Metoda ta jest generyczna, zatem musimy także określić na jaki typ mają być mapowane pobrane rekordy. Drugą możliwością wykonywania zapytań zwracających rekordy jest stworzenie obiektu klasy `SQLiteCommand` dzięki metodzie `CreateCommand`. Wykonanie polecenia następuje poprzez wywołanie metody `ExecuteQuery` obiektu `SQLiteCommand`.

Listing 5: Zapytania Select.

```
List<Client> AllClients = sqliteConnection.Query<Client>("SELECT *
    FROM CLIENTS");

string SQL = "SELECT * FROM CLIENTS WHERE Name LIKE ?";
SQLiteCommand command = sqliteConnection.CreateCommand(SQL, "A%");
List<Client> ClientsA = command.ExecuteQuery<Client>();
```

Drugim typem są polecenia niezwracające rekordów. Ich uruchamianie jest zbliżone do poprzedniego przykładu, lecz należy skorzystać z innych metod. Do bezpośredniego wykonania polecenia służy metoda `Execute` - analogicznie do metody `Query` opisanej powyżej. W przypadku, gdy używamy obiektu `SQLiteCommand` należy użyć metody `ExecuteNonQuery`. Metody te zwracają liczbę rekordów na które został wywarty wpływ.

Listing 6: Polecenia Insert.

```
int numRows = sqliteConnection.Execute("INSERT INTO CLIENTS (Name,
    Address) VALUES (?, ?)", "Adam", "ul. Warszawska 22");
```

```
2
3 string InsertSQL = "INSERT INTO CLIENTS (Name, Address) VALUES (?,
     ?)";
4 SQLiteCommand commandInsert = sqliteConnection.CreateCommand(
     InsertSQL, "Anna", "ul. Toruńska 15");
5 int numRowsInsert = commandInsert.ExecuteNonQuery();
```

W obu przypadkach możliwe jest parametryzowanie komend poprzez wstawienie odpowiedniej liczby znaków zapytania w treści wywołania. W ich miejsce wstawiane będą wartości podane w kolejnych argumentach wywoływanej metody. Podejście te zostało zobrazowane w listingu 6 w liniach 3. i 4.

5 Mapy w Universal Apps

5.1 Wstęp

Dla Windows 8.1 dostępne są mapy Bing. W Windows Phone do wersji 7.1 Bing Maps także były domyślnymi mapami, lecz później zostały zastąpione nową kontrolką. Bing Maps co prawda są dostępne w nowszych wersjach, ale są uznane za przestarzałe i zostały pozostawione jedynie w celu utrzymania wstecznej kompatybilności z aplikacjami przenoszonymi do nowszej wersji systemu. Nowa kontrolka z mapami w Windows Phone 8.1 to `MapControl`. W wersji na komputery stacjonarne jest to `Bing.Maps.Map`. W związku z powyższym niemożliwe jest współdzielenie jednej kontrolki w obu wersjach aplikacji, co jest sprzeczne z ideami Universal Apps.

Brak jednej wspólnej kontrolki z mapą wymusza rozdział logiki dwóch wersji aplikacji. Możliwe jest jednak stworzenie nowej, wspólnej kontrolki, która będzie implementowała obie mapy i używała odpowiedniej dla danej wersji systemu. W praktyce sprowadza się to do stworzenia w projekcie *Shared* klasy opakowującej, zgodnie z metodą przyjętą przez R. Brundritta opisaną w *How to Make Use of Maps in Universal Apps*[7], co umożliwia używanie wspólnego kodu w logice aplikacji.

5.2 Wymagane pakiety i rozszerzenia

Kontrolka `MapControl` w Windows Phone 8.1 jest dostępna domyślnie. Bing Maps dla Windows Store apps istnieje w postaci rozszerzenia do Visual Studio i wymaga pobrania i instalacji. Rozszerzenie można pobrać ze strony pod adresem: *https://visualstudiogallery.msdn.microsoft.com/224eb93a-ebc4-46ba-9be7-90ee777ad9e1*. Po jego instalacji niezbędne jest ponownie uruchomienie środowiska Visual Studio. Kolejnym krokiem jest dodanie referencji do Bing Maps SDK do projektu Windows 8.1. W menu *References > Add Reference...* w kategorii *Windows 8.1 > Extensions* należy zaznaczyć pola wyboru dla rozszerzeń *Bing Maps for C#, C++, or Visual Basic* oraz *Microsoft Visual C++2013 Runtime Package for Windows* (drugie rozszerzenie zainstaluje się au-

[7] R. Brundritt, *How to Make Use of Maps in Universal Apps*, *http://blogs.msdn.com/b/rbrundritt/archive/2014/06/24/how-to-make-use-of-maps-in-universal-apps.aspx*, [dostęp 20 października 2015]

tomatycznie także przy zaznaczeniu jedynie pierwszej z wyżej wymienionych opcji).

5.3 Tworzenie kontrolki opakowującej

Pierwszym krokiem jest stworzenie w projekcie *Shared* nowej klasy, która na potrzeby tej pracy nazwana została `UniversalMap`. Aby można było ją umieścić w kodzie XAML strony jako kontrolkę, musi dziedziczyć po klasie istniejącej kontrolki. Dodatkowym warunkiem jest obecność w bazowej kontrolce pola `Children`, do którego dodane będą mapy, a także jej dostępność zarówno w systemie w wersji Desktop i Phone. Obostrzenia te spełnia typ `Grid`, który może zostać użyty jako klasa bazowa naszej kontrolki.

W klasie powinny znaleźć się kontrolki z mapami odpowiednie dla wersji systemu. Aby zawrzeć je w jednej klasie, konieczne jest korzystanie z dyrektyw preprocesora dotyczących kompilacji warunkowej. Pozwalają one na wydzielenie fragmentów kodu, które mają się kompilować tylko w projekcie na zadany system.

Listing 7: Przykład kodu z kompilacją warunkową.

```
#if WINDOWS_APP
        private Bing.Maps.Map _map;
#elif WINDOWS_PHONE_APP
        private Windows.UI.Xaml.Controls.Maps.MapControl _map;
#endif
```

Pierwszy warunek kompilacji z podanego wyżej przykładu zachodzi w przypadku kompilowania na wersję Windows 8.1 i deklaruje obiekt Map z przestrzeni nazw `Bing.Maps`, która jest dostępna tylko dla tej wersji systemu. Drugi warunek odpowiada wersji Windows Phone i zawiera deklarację obiektu `MapControl`.

Do umieszczenia mapy na ekranie konieczne jest stworzenie konstruktora klasy i umieszczenie w nim kodu tworzącego kontrolki z mapami. Obie kontrolki tworzy się przy użyciu słowa kluczowego `new`. Po stworzeniu jednej z nich należy ją dodać do listy `Children` kontrolki `UniversalMap`.

Listing 8: Dotychczasowy kod klasy UniversalMap wraz z jej konstruktorem.

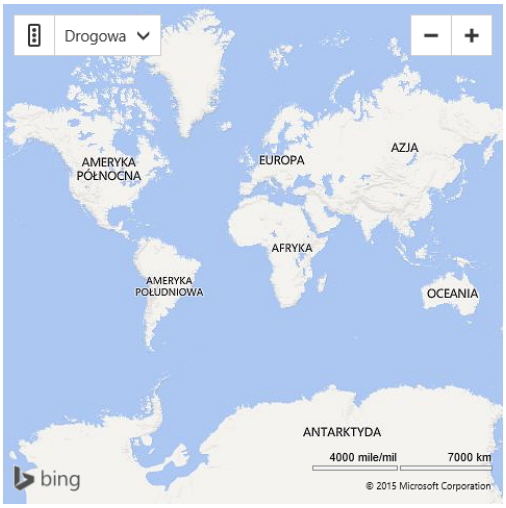
```
public class UniversalMap : Grid
{
#if WINDOWS_APP
    private Bing.Maps.Map _map;
#elif WINDOWS_PHONE_APP
    private Windows.UI.Xaml.Controls.Maps.MapControl _map;
#endif

```

```
9       public UniversalMap()
10      {
11  #if WINDOWS_APP
12          _map = new Map();
13  #elif WINDOWS_PHONE_APP
14          _map = new MapControl();
15  #endif
16          this.Children.Add(_map);
17      }
18  }
```

Po wykonaniu opisanych czynności możliwe jest już umieszczenie kontrolki z mapą na ekranie aplikacji:

Listing 9: Fragment kodu XAML służący do umieszczenia mapy na ekranie.

```
1 <local:UniversalMap x:Name="Map"/>
```



Rys. 3: Kontrolka z Bing Maps.

Rejestrowanie mapy

Aby mapa działała poprawnie musi zostać zarejestrowana indywidualnym kluczem. Dla Bing Maps użytych w Windows 8.1 wygenerować go można przez stworzenie konta deweloperskiego na stronie Bing Dev Center - *www.bingmapsportal.com*. Przy tworzeniu klucza należy podać nazwę aplikacji oraz jej typ. Jeśli mapa nie zostanie uwierzytelniona, użytkownik otrzyma komunikat o nieprawidłowych poświadczeniach. Aby aktywować mapę przy pomocy klucza należy wpisać go do pola `Credentials`, co zobrazowane jest poniżej w listingu 11 w linii 4.

Klucz dla Map używanych w Windows Phone 8.1 można uzyskać w portalu Windows Dev Center - *dev.windows.com* przu użyciu konta developera. Aby tego dokonać

należy stworzyć nową aplikację przy użyciu opcji *Create a new app*. Po zarezerwowaniu wybranej nazwy aplikacji dostępna jest opcja *Services >Maps >Get token*. W wyniku jej działania zostaje wyenerowany numer ID aplikacji *(Map service Application ID)* oraz klucz uwierzytelniający *Map service Authentication Token*.

ID aplikacji musi zostać umieszczone w pliku *Package.appxmanifest* w projekcie Windows Phone. Miejsce do wpisania klucza nie jest dostępne w interfejsie graficznym, stąd konieczne jest otworzenie pliku w postaci edytowalnego kodu. W Visual Studio dokonać tego można poprzez kliknięcie prawym klawiszem myszy i wybranie opcji *View Code*. W tagu `map:PhoneIdentity` w atrybucie `PhoneProductId` należy umieścić otrzymany identyfikator (por. listing 10). Do uwierzytelnienia kontrolki z mapą używany jest drugi z otrzymanych kluczy. Musi on być umieszczony w polu `MapServiceToken` w kontrolce `MapControl`. Proces ten został zobrazowany w listingu 11 w linii 6.

Listing 10: Umieszczanie ID Aplikacji w pliku Package.appxmanifest.

```
1 <mp:PhoneIdentity PhoneProductId="--Application-ID--"
    PhonePublisherId="00000000-0000-0000-0000-000000000000" />
```

Listing 11: Uwierzytelnianie mapy.

```
1 public void Register(string token)
2 {
3 #if WINDOWS_APP
4     _map.Credentials = token;
5 #elif WINDOWS_PHONE_APP
6     _map.MapServiceToken = token;
7 #endif
8 }
```

W powyższym listingu umieszczona została metoda `Register` która przyjmuje klucz i umieszcza go w odpowiedniej zmiennej w mapie. Dzięki temu klucze mogą znajdować się w kodzie samej aplikacji, zaś klasa UniversalMap może być użyta w innym projekcie bez potrzeby modyfikacji jej kodu.

Ustawianie widoku na zadany punkt

Kontrolki map w obu wersjach systemu używają różnych klas do obsługi koordynat. W Windows Phone kontrolka `MapControl` korzysta z klasy `Geopoint`, zaś Bing Maps dla wersji Desktop używa klasy `Location`. Z tego powodu niezbędna jest konwersja między obiektami tych klas. Jednym ze sposobów, aby to zrobić jest używanie w części wspólnej

klasy `Geopoint`, a konwersji na typ używany przez konkretną mapę dokonywać w klasie opakowującej, w kodzie odpowiadającym danej wersji systemu.

Aby zminimalizować ilość redundantnego kodu konwersję między typami można wyodrębnić do oddzielnej klasy statycznej. Jej metody są rozszerzeniami istniejących klas i mogą być wywoływane w identyczny sposób, jak ich metody natywne. Kod ten jest niezbędny tylko podczas kompilacji dla wersji Desktop, gdyż klasa `Geopoint` jest obsługiwana przez Windows Phone. Z tego powodu metody powinny zostać objęte dyrektywami preprocesora dotyczącymi kompilacji warunkowej.

Listing 12: Klasa z metodami służącymi do konwersji typów.

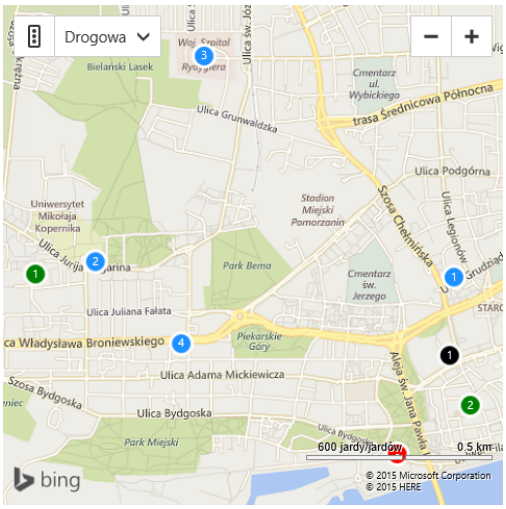
```
using Windows.Devices.Geolocation;

#if WINDOWS_APP
using Bing.Maps;
#endif

public static class Conversion
{
#if WINDOWS_APP
    public static Location ToLocation(this BasicGeoposition
        basicGeoposition)
    {
        return new Location(basicGeoposition.Latitude,
            basicGeoposition.Longitude);
    }

    public static Geopoint ToGeopoint(this Location location)
    {
        return new Geopoint(new BasicGeoposition() { Latitude =
            location.Latitude, Longitude = location.Longitude });
    }
#endif
}
```

Ustawianie widoku na zadany punkt w kontrolce `Bing.Maps.Map` dokonywane jest przy użyciu metody `SetView` dla której określamy punkt oraz poziom zbliżenia. W `MapControl` dokonuje się poprzez ustawienie pól `Center` oraz `ZoomLevel`.

Listing 13: Metoda do centrowania mapy.

```
public void SetView(Geopoint center, double zoom = 7)
{
#if WINDOWS_APP
    _map.SetView(center.Position.ToLocation(), zoom);
#elif WINDOWS_PHONE_APP
    _map.Center = center;
    _map.ZoomLevel = zoom;
#endif
}
```

Ustawianie widoku na kilka punktów



Rys. 4: Kontrolka z Bing Maps po ustawieniu widoku na zadane punkty.

Analogicznie możliwe jest ustawienie widoku w taki sposób, aby wszystkie zadane punkty zmieściły się na widocznym fragmencie mapy. Polega to na wyznaczeniu bryły brzegowej (bounding box) dla zbioru punktów. Na potrzeby ninejszej pracy stworzona została metoda `SetViewAll`, która jako argument przyjmuje listę punktów typu `Geopoint`. W mapach Bing dla Windows 8.1 konieczne jest stworzenie kolekcji typu `LocationCollection` i wypełnienie jej punktami. Niezbędne jest konwertowanie typu `Geopoint` do typu `Location`. Do tego można wykorzystać metodę statyczną `ToLocation` która została dodana w podrozdziale *Ustawianie widoku na zadany punkt* w niniejszym rozdziale. Bryła brzegowa generowana jest przy pomocy obiektu typu `LocationRect`, przy tworzeniu którego należy podać wcześniej uzyskaną kolekcję lokacji. Uzyskany bounding box przekazany jest do metody `SetView` obiektu typu `Map`. Zostało to zilustrowane na listingu 14 w liniach 9-10.

Dla Windows Phone 8.1 proces ten jest zbliżony. Tworzenie obiektu `LocationCollection` zostaje zastąpione stworzeniem listy z obiektami typu `BasicGeoposition`. Obiekty tego typu zawarte są bezpośrednio w obiektach `Geopoint`, zatem konieczne jest ich „wyłuskanie”. Aby stworzyć bryłę brzegową skorzystać należy z metody statycznej `GeoboundingBox.TryCompute`. Uzyskana bryła powinna zostać przekazana metodzie `TrySetViewBoundsAsync`. Jako jej argumenty, oprócz bounding box, należy podać szerokość marginesu tak, aby punkty nie pojawiały się na krawędzi widoku oraz typ animacji

przesunięcia widoku.

Listing 14: Ustawianie widoku na kilka punktów.

```
1  public async void SetViewAll(List<Geopoint> PointsList)
2  {
3  #if WINDOWS_APP
4      LocationCollection Locations = new LocationCollection();
5      foreach (var Point in PointsList)
6      {
7          Locations.Add(Point.Position.ToLocation());
8      }
9      LocationRect rect = new LocationRect(Locations);
10     _map.SetView(rect);
11 #elif WINDOWS_PHONE_APP
12     List<BasicGeoposition> Locations = new List<BasicGeoposition>();
13     foreach (var Point in PointsList)
14     {
15         Locations.Add(Point.Position);
16     }
17     GeoboundingBox Box = GeoboundingBox.TryCompute(Locations);
18     await _map.TrySetViewBoundsAsync(Box, new Thickness(5),
           MapAnimationKind.Bow);
19 #endif
20 }
```

Oznaczanie miejsc na mapie

Do oznaczania punktów na mapie służą pinezki. W Bing Maps dostępna jest klasa `Pushpin` automatycznie tworząca pinezkę, lecz jest ona niedostępna w przypadku map dla Windows Phone 8.1. W związku z powyższym aby ujednolicić kod metody oraz wygląd pinezki można ręczne stworzyć obiekt służący za pinezkę. Może on się składać z kilku kontrolek, na przykład prostych kształtów. W podanym poniżej przykładzie pinezka składa się z elipsy oraz pola tekstowego. Przed dodaniem pinezki do listy pochodnych mapy konieczne jest ustawienie jej pozycji względem koordynat mapy. Dla Bing Maps dokonywane jest to przy użyciu statycznej metody `MapLayer.SetPosition`, zaś w mapach Windows Phone 8.1 służy do tego metoda `MapControl.SetLocation`, co zostało zobrazowane w listingu 15 w liniach 25-29.

Listing 15: Tworzenie i dodawanie pinezki.

```
1  public void AddPushpin(Geopoint point, string text)
2  {
3      int PinSize = 24;
4      var Pin = new Grid()
5      {
6          Width = PinSize, Height = PinSize,
7          Margin = new Windows.UI.Xaml.Thickness(-PinSize*0.5)
8      };
9
10     Pin.Children.Add(new Ellipse()
11     {
12         Fill = new SolidColorBrush(Colors.DodgerBlue),
```

```
13          Stroke = new SolidColorBrush(Colors.White),
14          StrokeThickness = 3,
15          Width = PinSize,  Height = PinSize
16      });
17
18      Pin.Children.Add(new TextBlock()
19      {
20          Text = text,
21          HorizontalAlignment = Windows.UI.Xaml.HorizontalAlignment.
                Center,
22          VerticalAlignment = Windows.UI.Xaml.VerticalAlignment.Center
23      });
24
25  #if WINDOWS_APP
26      MapLayer.SetPosition(Pin, point.Position.ToLocation());
27  #elif WINDOWS_PHONE_APP
28      MapControl.SetLocation(Pin, point);
29  #endif
30      _map.Children.Add(Pin);
31  }
```

Wyznaczanie lokalizacji przy użyciu adresu

Obie kontrolki map pozwalają na wyznaczanie lokalizacji przy użyciu adresu. Proces ten nazywany jest geocodingiem. Aby użyć tej funkcjonalności w Bing Maps, konieczne jest dołączenie przy użyciu słowa kluczowego `using` przestrzeni nazw `Bing.Maps.Search`. Zapytanie tworzy się przy użyciu obiektu typu `GeocodeRequestOptions`. Wyszukaniem lokalizacji zajmuje się asynchroniczna metoda `GeocodeAsync` obiektu `SearchManager`. Zwracany przez nią obiekt `LocationDataResponse` zawiera wyniki procesu geocodingu. Zawierać może on jeden bądź więcej wyników, a w przypadku błędu jest pusty.

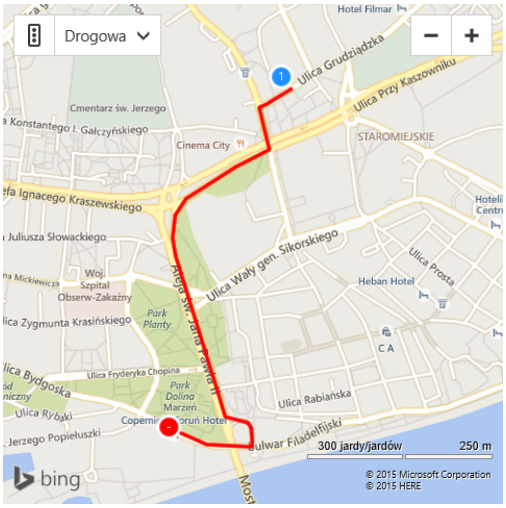
Dla systemu Windows Phone 8.1 niezbędne obiekty znajdują się w przestrzeni nazw `Windows.Services.Maps`. Do wyznaczenia lokalizacji służy metoda `MapLocationFinder.FindLocationsAsync`. Wymaga ona podania szukanego adresu oraz punktu, który służy jako podpowiedź przy wyszukiwaniu. W podanym poniżej przykładzie jako wyżej wymieniony punkt użyta została aktualna pozycja widoku mapy. Rezultat działania wyszukiwania zostaje zwrócony jako obiekt `MapLocationFinderResult`. Podobnie jak w Bing Maps należy upewnić się, że zwracany rezultat jest poprawny oraz zawiera dane o położeniu.

Listing 16: Geocoding.

```
1  public async Task<Geopoint> GetByAddress(string Address)
2  {
3  #if WINDOWS_APP
4      GeocodeRequestOptions GeocodeRequest = new GeocodeRequestOptions
           (Address);
```

```
5       LocationDataResponse Response = await _map.SearchManager.
            GeocodeAsync(GeocodeRequest);
6
7       if (!Response.HasError && Response.LocationData.Count > 0)
8       {
9           var res = Response.LocationData[0].Location.ToGeopoint();
10          return res;
11      }
12  #elif WINDOWS_PHONE_APP
13      Geopoint Hint = _map.Center;
14      MapLocationFinderResult result = await MapLocationFinder.
            FindLocationsAsync(Address, Hint);
15
16      if (result.Status == MapLocationFinderStatus.Success && result.
            Locations.Count > 0)
17      {
18          return result.Locations[0].Point;
19      }
20  #endif
21      return null;
22  }
```

Wyznaczanie trasy



Rys. 5: Kontrolka z Bing Maps z wyświetloną wyznaczoną trasą.

Obydwie biblioteki do obsługi map pozwalają na wyznaczanie trasy pomiędzy zadanymi punktami, także z uwzględnieniem punktów pośrednich.

Do obsługi logiki tras w Mapach Bing służy obiekt `DirectionsManager` znajdujący się w obiekcie mapy. Przed wyznaczeniem trasy konieczne jest umieszczenie punktów docelowych oraz pośrednich w kolekcji `Waypoints` obiektu `DirectionsManager`. W podanym poniżej przykładzie metody `GetRoute` punkty są przyjmowane w postaci listy obiektów typu `Geopoint`. Podczas wypełniania kolekcji `Waypoints` punkty są konwertowane do

typu `Location`.

Obiekt `RequestOptions` daje możliwość zdefiniowania bardziej szczegółowych parametrów wyszukiwania, takich jak: wybór środka transportu, wybór jednostek miary odległości. Możliwa jest także optymalizacja według czasu, odległości, kosztu, ilości przesiadek itp. oraz wskazanie typu dróg lub środków komunikacji, których chce się unikać. Zagadnienie to zostało zobrazowane w listingu 17 w liniach 10-13.

Obiekt `RenderOptions` daje możliwość określenia parametrów wyświetlania danej trasy, takich jak kolor i szerokość linii obrazującej trasę. Kolejną opcją są znaczniki punktów. Przy wyświetlaniu trasy dodają się one automatycznie, lecz można z nich zrezygnować na przykład gdy już wcześniej zostały dodane w postaci pinezek. Zagadnienie to zostało zobrazowane w listingu 17 w liniach 15-17.

W przypadku map używanych w Windows Phone 8.1 do wyznaczania tras służy klasa statyczna `MapRouteFinder`. Posiada ona metody specyficzne dla jazdy samochodem oraz dla ruchu pieszego. W przypadku ruchu drogowego użyć należy metody `GetDrivingRouteFromWaypointsAsync`. Jako argumenty przyjmuje ona listę punktów zapisanych w obiektach typu `Geopoint`. Podobnie jak w Mapach Bing możliwe jest określenie preferowanej metody optymalizacji trasy, to jest względem dystansu, czasu lub czasu z uwzględnieniem ruchu ulicznego. Także możliwe jest wskazanie typu dróg, których chce się unikać.

Opisywana metoda zwraca obiekt typu `MapRouteFinderResult` w którym znajduje się pole `Status`. Na jego podstawie możemy określić czy wyszukiwanie trasy zakończyło się pomyślnie. Jeśli trasa została poprawnie zwrócona do jej wyświetlenia służy obiekt `MapRouteView`. Do jego konstruktora należy przekazać wyszukaną trasę znajdującą się w obiekcie `MapRouteFinderResult`. Obiekt pozwala na określenie koloru linii obrazującej trasę. Ostatecznie aby wyświetlić trasę na mapie należy dodać obiekt `MapRouteView` do kolekcji `Routes` znajdującej się w obiekcie mapy.

Listing 17: Geocoding.

```
1 public async Task GetRoute(List<Geopoint> Points)
2 {
3 #if WINDOWS_APP
4     _map.DirectionsManager.ClearActiveRoute();
5     _map.DirectionsManager.Waypoints.Clear();
6     foreach (Geopoint Gpt in Points)
7     {
```

```
8           _map.DirectionsManager.Waypoints.Add(new Bing.Maps.
               Directions.Waypoint(Gpt.Position.ToLocation()));
9       }
10      _map.DirectionsManager.RequestOptions.RouteMode = Bing.Maps.
           Directions.RouteModeOption.Driving;
11      _map.DirectionsManager.RequestOptions.DistanceUnit = Bing.Maps.
           Directions.DistanceUnitOption.Kilometer;
12      _map.DirectionsManager.RequestOptions.Optimize = Bing.Maps.
           Directions.OptimizeOption.Time;
13      _map.DirectionsManager.RequestOptions.Avoid = Bing.Maps.
           Directions.AvoidOption.Highways | Bing.Maps.Directions.
           AvoidOption.ExpressTrain;
14
15      _map.DirectionsManager.RenderOptions.ActiveRoutePolylineOptions.
           LineColor = Colors.Peru;
16      _map.DirectionsManager.RenderOptions.ActiveRoutePolylineOptions.
           LineWidth = 8.0;
17      _map.DirectionsManager.RenderOptions.WaypointPushpinOptions.
           Visible = false;
18
19      try
20      {
21          await _map.DirectionsManager.CalculateDirectionsAsync();
22      }
23
24  #elif WINDOWS_PHONE_APP
25      _map.Routes.Clear();
26
27      MapRouteFinderResult RouteResult = await MapRouteFinder.
           GetDrivingRouteFromWaypointsAsync(Points,
           MapRouteOptimization.Time, MapRouteRestrictions.DirtRoads);
28
29      if (RouteResult.Status == MapRouteFinderStatus.Success)
30      {
31          MapRouteView RouteView = new MapRouteView(RouteResult.Route)
               ;
32          RouteView.RouteColor = Colors.Blue;
33          RouteView.OutlineColor = Colors.Red;
34          _map.Routes.Add(RouteView);
35      }
36  #endif
37  }
```

6 Geolokacja

6.1 Wstęp

Geolokacja to proces określania położenia geograficznego danego urządzenia, to jest telefonu bądź komputera. Polega on na wyznaczeniu współrzędnych geograficznych, czasem włączając w to wysokość. Jednym z najpopularniejszych systemów geolokacji jest Global Positioning System, którego działanie opiera się na odbiorze sygnału z satelit znajdujących się na orbicie okołoziemskiej. Możliwe jest także określanie połozenia na podstawie adresu IP, lecz jest ono mniej dokładne.

6.2 Geolokacja w Windows Runtime

Windows Runtime udostępnia funkcjonalność geolokacji w przestrzeni nazw `Windows.Devices.Geolocation`. Oprócz określania pozycji urządzenia dostępna jest także funkcjonalność geofencingu, czyli określania obszaru o którego opuszczeniu aplikacja ma być informowana. Geolokacja w WinRT może obsługiwać kilka różnych metod wyznaczania położenia. Każda z nich ma określoną przybliżoną dokładność:

- GPS - 10 metrów
- Wi-Fi-based positioning system - 30 - 500 metrów
- wieże przekaźnikowe - 300 - 3000 metrów
- adres IP - 1000 - 5000 metrów

6.3 Używanie geolokacji w Windows Runtime

Aby móc korzystać z geolokacji w Windows Runtime konieczne jest zadeklarowanie jej użycia. Aby to uczynić należy w pliku *Package.appxmanifest* w karcie *Capabilities* zaznaczyć opcję *Location*. Proces ten wykonać trzeba dla obu projektów.

Do geolokacji w Windows Runtime służą obiekty klasy `Geolocator` należącej do przestrzeni nazw `Windows.Devices.Geolocation`. Obiekt `Geolocator` pozwala na określenie oczekiwanej dokładności pomiaru. Domyślna wartość pola `DesiredAccuracy` to

`Default`, która odpowiada dokładności 500 metrów. Ustawienie wartości na `High` zwiększa dokładność do 10 metrów. Możliwe jest także określenie oczekiwanej dokładności bezpośrednio w metrach poprzez ustawienie pola `DesiredAccuracyInMeters`. Ustawienie wartości na `High` może spowodować większe zużycie energii.

Uzyskanie aktualnej pozycji

Aby uzyskać aktualne położenie należy użyć metody `GetGeopositionAsync`. Jest to metoda asynchroniczna zwracająca obiekt typu `Geoposition`. Możliwe jest przekazanie w argumentach dwóch parametrów. Pierwszy z nich pozwala na zwrócenie pozycji wyznaczonej poprzednio, ale nie starszej niż określa to parametr. Dzięki temu wartość może zostać podana natychmiast bez konieczności wykonywania nowszych pomiarów. Drugi parametr określa maksymalny czas, w którym `Geolocator` powiniem zwrócić wynik. Jeśli zostanie przekroczony, aplikacja wygeneruje wyjątek.

Ze względu na to, że funkcja geolokacji w urządzeniu nie zawsze jest dostępna, konieczne jest uwzględnienie tego w kodzie. Status obiektu `Geolocator` zawarty jest w polu `LocationStatus`. Określa on dostępność geolokacji i w zależności od stanu jego działanie może być różne. Przy próbie uzyskania położenia, podczas gdy użytkownik wyłączył takie uprawnienia dla aplikacji, wygenerowany zostanie wyjątek `UnauthorizedAccessException`. Z tego powodu dobrym podejściem jest użycie bloków `try/catch`. Obsługa `LocationStatus` oraz wspomnianego wyjątku zobrazowana jest w listingu 18.

Listing 18: Pobranie aktualnej pozycji.

```
1  public static Geolocator geolocator = new Geolocator();
2
3  async static public Task<Geopoint> GetPosition()
4  {
5      Geopoint Point = null;
6      try
7      {
8          switch (geolocator.LocationStatus)
9          {
10         case Windows.Devices.Geolocation.PositionStatus.Ready:
11         case Windows.Devices.Geolocation.PositionStatus.
               NotInitialized:
12         case Windows.Devices.Geolocation.PositionStatus.Initializing
               :
13             Geoposition GeoPos = await geolocator.
                   GetGeopositionAsync(TimeSpan.FromMinutes(5), TimeSpan
                   .FromSeconds(20));
14             Point = GeoPos.Coordinate.Point;
15             break;
16         case Windows.Devices.Geolocation.PositionStatus.NoData:
17         case Windows.Devices.Geolocation.PositionStatus.Disabled:
```

```
18          case Windows.Devices.Geolocation.PositionStatus.NotAvailable
                :
19          default:
20              // Uzyj domyslnej lokalizacji lub zwroc null
21              break;
22          }
23      }
24      catch (UnauthorizedAccessException ex)
25      {
26          // Uzyj domyslnej lokalizacji lub zwroc null
27      }
28      return Point;
29  }
```

Metoda `GetGeopositionAsync` zwraca obiekt typu `Geoposition`, zaś informacje o położeniu zawarte są w jego polu `Coordinate`. Znajdują się tam, oprócz samych współrzędnych, także dane o dokładności pomiaru oraz źródle użytym do jego wykonania.

Subskrybowanie zdarzenia zmiany pozycji

Powyżej została opisana metoda jednokrotnego pobrania aktualnej pozycji. Oprócz tego możliwe jest także subskrybowanie zdarzenia zmiany pozycji urządzenia. Niezbędne jest ustawienie opcji `ReportInterval`, która określa jak często tj. co ile milisekund aplikacja powinna otrzymywać aktualizację położenia. Dla programów, które nie wymagają bardzo dokładnej pozycji można określić progową zmianę położenia w metrach `MovementThreshold`. Aby zasubskrybować zdarzenie `PositionChanged` należy stworzyć metodę obsługującą to zdarzenie, a następnie dodać ją do delegatu `PositionChanged`.

Listing 19: Subskrybowanie zdarzenia zmiany pozycji.

```
1  geolocator.MovementThreshold = 10;
2  geolocator.PositionChanged += this.OnPositionChanged;
3  ...
4  public void OnPositionChanged(Geolocator G, PositionChangedEventArgs
       Args)
5  {
6      // Obsługa zdarzenia
7  }
```

Geofencing

Kolejną funkcjonalnością związaną z geolokacją jest tzw. *geofencing*. Pod określeniem tym kryje się tworzenie obszarów oraz monitorowanie położenia urządzenia względem nich. Możliwe jest otrzymywanie informacji o wejściu do zadanego obszaru bądź o jego opuszczeniu.

Klasy potrzebne do obsługi geofencingu znajdują się w prezstrzeni nazw `Windows.Devices.Geolocation.Geofencing`. Aby zdefiniować aktywny obszar konieczne jest stworzenie i zarejestrowanie obiektu typu `Geofence`. Jego konstruktor wymaga podania unikatowego identyfikatora, a także obiektu implementującego interfejs `IGeoshape`, który definiuje obszar. W Windows 8.1 jedynym takim obiektem jest `Geocircle`, który określany jest poprzez punkt centralny oraz promień. Dodatkowo przy tworzeniu obieku `Geofence` możliwe jest wybranie zdarzeń, o których aplikacja ma być informowana, to jest wkroczeniu, opuszczeniu bądź usunięciu obszaru. Dokonuje się tego poprzez przekazanie wartości typu `MonitoredGeofenceStates`. Monitorowanie `Geofence` może zakończyć się po jednokrotnej zmianie stanu lub działać do czasu jego usunięcia. Wskazuje na to wartość argumentu `singleUse` w konstruktorze. Wartość `dwellTime` oznacza natomiast czas, jaki urządzenie musi spędzić w obszarze, żeby aplikacja została powiadomiona. Kolejnymi opcjonalnymi argumentami przy tworzeniu `Geofence` są: czas startu monitorowania oraz okres jego trwania, czyli `startTime` oraz `duration`. Konfigurację obiektu przedstawiono w listingu 20.

Listing 20: Tworzenie obiektu Geofence.

```
1 Geocircle Cir = new Geocircle(this.CurrentPosition, 30);
2 MonitoredGeofenceStates GeoFenceStates = MonitoredGeofenceStates.
     Entered;
3 GeoFenceStates |= MonitoredGeofenceStates.Exited;
4 TimeSpan dwellTime = TimeSpan.FromSeconds(1);
5 DateTimeOffset GeoFenceStartTime = DateTimeOffset.Now;
6 Geofence geofence = new Geofence("id-obszar", Cir, GeoFenceStates,
     false, dwellTime, GeoFenceStartTime, TimeSpan.FromDays(1));
```

Aby rozpocząć obserwację konieczne jest dodanie stworzonego obiektu `Geofence` do singletonu `GeofenceMonitor`. Dostęp do aktywnego monitora można uzyskać poprzez statyczne pole `GeofenceMonitor.Current`. Lista śledzonych obszarów znajduje się w polu `Geofences`. Dodanie *Geofence* do listy możliwe jest dzięki metodzie `Add`. Podobnie jak w poprzednim punkcie, do obsługi zdarzeń konieczne jest stworzenie odpowiedniej metody i dodanie jej do delegatu `GeofenceStateChanged`. Informacje o zdarzeniach można odczytać z raportów, które znajdują się w monitorze i są dostępne dzięki metodzie `ReadReports`. Iterując po zwracanej przez metodę kolekcji mozna odczytać zawarte w raportach powiadomienia. W każdym raporcie znajduje się referencja do obiektu `Geolocator`, którego dotyczy oraz jego nowy stan, np. informacja o wkroczenu bądź opuszczeniu obszaru. Dodawanie obiektu `Geofence` do monitora oraz odczytywanie wydarzeń zobrazowane zostało w listingu 21.

Listing 21: Dodanie obiektu Geofence do GeofenceMonitor.

```
GeofenceMonitor.Current.Geofences.Add(geofence);
GeofenceMonitor.Current.GeofenceStateChanged +=
    Current_GeofenceStateChanged;

private async void Current_GeofenceStateChanged(GeofenceMonitor
    sender, object args)
{
     var Reports = sender.ReadReports();
     foreach (GeofenceStateChangeReport R in Reports)
     {
         var newState = R.NewState;
         Geofence geofenceSource = R.Geofence;
     }
}
```

7 Wizualizacja danych w Universal Apps

7.1 Wstęp

Środowisko Windows Runtime nie posiada zbyt wielu wbudowanych kontrolek służących do obrazowania danych. Dostępne są jednak zewnętrzne biblioteki, takie jak *WinRT XAML Toolkit*. Projekt ten w dużej mierze bazuje na bibliotece *Silverlight Toolkit* przeznaczonej do starszej technologii Microsoftu. Biblioteka zawiera wiele różnych kontrolek, których można użyć w aplikacji. Dodatkowo dostępne jest dla niej rozszerzenie *Data Visualization Controls*, w którym zawarte są kontrolki użyteczne do wizualizacji danych, m.in. wykresy słupkowe, kołowe, liniowe i inne. Kod źródłowy biblioteki jest dostępny na stronie projektu pod adresem *https://winrtxamltoolkit.codeplex.com*.

7.2 Instalacja pakietów

WinRT XAML Toolkit dostępny jest w menadżerze pakietów NuGet. Aby dodać bibliotekę do projektów, należy otworzyć menu *References > Manage NuGet Packages....* W przypadku projektu *Windows 8.1* należy zainstalować pakiety *WinRT XAML Toolkit for Windows 8.1* oraz *WinRT XAML Toolkit - Data Visualization Controls for Windows 8.1*. Dla projektu *Windows Phone 8.1* pakiety mają nazwy *WinRT XAML Toolkit for Windows Phone 8.1* oraz *WinRT XAML Toolkit - Data Visualization Controls for Windows Phone 8.1*.

7.3 Dodawanie wykresów do aplikacji

Aby dodać wykres na danej stronie w aplikacji należy w pliku *.xaml* wewnątrz znacznika `Page` zadeklarować odpowiednie przestrzenie nazw. Przykładowa deklaracja znajduje się w listingu 22.

Listing 22: Dodawanie przestrzeni nazw do pliku.

```
1 <Page
2 ...
3     xmlns:charting="using:WinRTXamlToolkit.Controls.
          DataVisualization.Charting"
4     xmlns:datavis="using:WinRTXamlToolkit.Controls.DataVisualization
          "
5     >
```

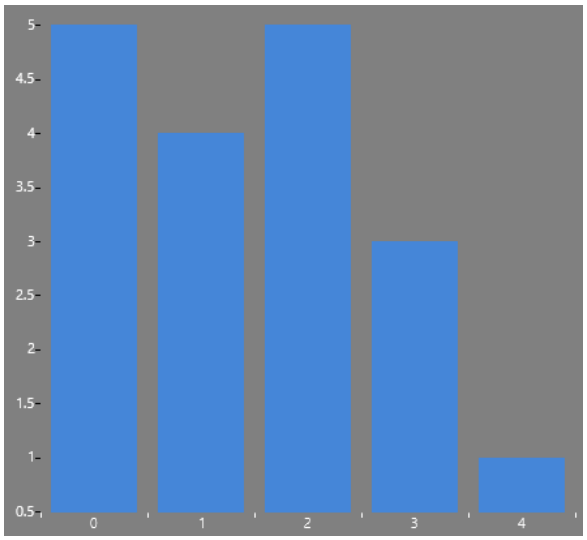
Po wykonaniu powyższej czynności, możliwe jest dodanie kontrolki z wykresem. Kontrolka jest wspólna dla wszystkich typów wykresów. Ma ona nazwę `charting:Chart`. Typ wykresu jest zależny od typu umieszczonej w nim serii danych. Odpowiednio:

- kołowy - znacznik `charting:PieSeries`,
- kolumnowy - znacznik `charting:ColumnSeries`,
- warstwowy - znacznik `charting:AreaSeries`,
- słupkowy - znacznik `charting:BarSeries`,
- liniowy - znacznik `charting:LineSeries`,
- punktowy - znacznik `charting:ScatterSeries`,
- bąbelkowy - znacznik `charting:BubbleSeries`.

Na jednym wykresie możliwe jest umieszczenie różnych typów serii. Dla przykładu jeden zbiór danych można reprezentować wykresem kolumnowym, a inny liniowym nałożonym na pierwszy.Przykładowy wykres z jedną serią dodany w kodzie XAML został zobrazowany w poniższym listingu 23.

Listing 23: Przykładowy wykres.

```
1 <charting:Chart x:Name="DataChart">
2     <charting:ColumnSeries IndependentValuePath="Argument"
          DependentValuePath="Value"/>
3 </charting:Chart>
```



Rys. 6: Przykładowy wykres.

Dane do serii przekazywane są w postaci kolekcji implementującej interfejs `IEnumerable`, na przykład `List`. Obiekty służące za punkty danych muszą zawierać przynajmniej dwa pola - etykietę oraz wartość. Przypisanie danych do wykresu następuje poprzez ustawienie w obiekcie serii pola `ItemsSource`. Na przykładzie wykresu stworzonego w listingu 23 proces ten może wyglądać następująco:

Listing 24: Dodawanie źródła danych.

```
1  public class Data
2  {
3      public int Argument { get; set; }
4      public int Value { get; set; }
5  }
6
7  ...
8
9  List<Data> DataList;
10
11 ...
12
13 (DataChart.Series[0] as ColumnSeries).ItemsSource = DataList;
```

Określenie pól używanych w kontrolce jako etykiety oraz wartości zostało pokazane w listingu 23 podczas deklarowania serii danych. Parametrem `IndependentValuePath` określone jest pole obiektu danych, które zawiera etykietę, zaś parametrem `DependentValuePath` pole wartości. Za etykietę może służyć wartość liczbowa, ciąg znaków (`string`) bądź daty (typ `DateTime`). Możliwe jest także użycie jako etykiety innego typu obiektu - wtedy oznaczeniami na osi X będą wartości zwracane z metody `ToString` tych obiektów.

7.4 Dostosowywanie wykresów

Wygląd wykresów można dostosować poprzez modyfikację ich stylu, zarówno w zakresie kolorystyki, kształtu, jak i czytelności prezentacji danych.

Zmiana koloru serii danych

Dla wszystkich wykresów za wyjątkiem kołowego, seria danych reprezentowana jest przy użyciu jednego koloru. Kolejne serie otrzymują kolor z używanej przez wykres palety. Domyślna paleta zawiera style o wielu kolorach, które pozwalają na łatwe odróżnienie od siebie poszczególnych serii. Jeżeli aplikacja używa określonej kolorystyki, możliwe jest odwzorowanie tej kolorystyki także na wykresie. Aby tego dokonać, należy zdefiniować dla

wykresu nową paletę. Paleta jest definiowana wewnątrz tagu `charting:Chart.Palette`, który jest zawarty wewnątrz `charting:Chart`.

Głównym elementem definicji palety jest kolekcja słowników `charting:ResourceDictionaryCollection`, w której zawarte są zasoby opisujące styl poszczególnych serii. Każdej serii danych odpowiada kolejny element tej kolekcji typu `ResourceDictionary`. Jeżeli na wykresie znajduje się więcej serii niż zadeklarowanych definicji, ponownie aplikowane zostają style z początku kolekcji. Dodawanie palety składającej się z dwóch barw zostało zobrazowane w listingu 25. W wykresie tym umieszczone zostały trzy serie danych, zatem pierwsza oraz trzecia będą używały tego samego koloru.

Listing 25: Zmiana barw serii w wykresie.

```
<charting:Chart x:Name="DataChart">
  <charting:ColumnSeries IndependentValuePath="Argument"
     DependentValuePath="Value"/>
  <charting:ColumnSeries IndependentValuePath="Argument"
     DependentValuePath="Value"/>
  <charting:ColumnSeries IndependentValuePath="Argument"
     DependentValuePath="Value"/>
  <charting:Chart.Palette>
    <charting:ResourceDictionaryCollection>
      <ResourceDictionary>
        <SolidColorBrush x:Key="BG" Color="Firebrick" />
        <Style x:Key="DataPointStyle" TargetType="Control">
          <Setter Property="Background" Value="{StaticResource BG}"
             />
        </Style>
      </ResourceDictionary>
      <ResourceDictionary>
        <SolidColorBrush x:Key="BG" Color="OldLace" />
        <Style x:Key="DataPointStyle" TargetType="Control">
          <Setter Property="Background" Value="{StaticResource BG}"
             />
        </Style>
      </ResourceDictionary>
    </charting:ResourceDictionaryCollection>
  </charting:Chart.Palette>
</charting:Chart>
```

Wykres kołowy pozwala na reprezentację tylko jednej serii danych. Kolejne jej wartości odróżniane są kolorami. Wykres kołowy posiada kilka wbudowanych palet. Dodanie własnej palety w opisany powyżej sposób w przypadku wykresu kołowego skutkuje określeniem kolorów kolejnych wartości w serii.

Modyfikowanie osi

W zależności od typów, jakie zostały użyte w danych jako etykiety i wartości osie wykresu mogą być różnego typu. Dostępne typy osi to:

- `LinearAxis` do reprezentowania wartości liczbowych,
- `CategoryAxis` do przedstawiania kategorii opisowych,
- `DateTimeAxis` do reprezentowania czasu (daty oraz godziny).

Dostępne opcje dostosowywania osi zależne są od jej typu. Poniżej znajdują się przykłady częstych modyfikacji.

Dostęp do obiektów reprezentujących osie możliwy jest poprzez pole `ActualAxes` znajdujące się w obiekcie wykresu. Pole to zostaje uzupełnione po załadowaniu wykresu, zatem modyfikacja osi może zostać dokonana w metodzie obsługującej zdarzenie `Loaded` kontrolki wykresu. `ActualAxes` jest kolekcją obiektów reprezentujących osie wykresu. Kolekcja ta zazwyczaj posiada dwa elementy: oś etykiet i oś wartości. Aby modyfikować parametry specyficzne dla danego typu osi, konieczne jest stosowanie konwersji przy użyciu słowa kluczowego `as` jak zostało zobrazowane w listingu 26.

Oś liczbowa `LinearAxis` domyślnie jest wyskalowana z użyciem wartości ułamkowych. W zależności od potrzeby możliwe jest ustalenie interwału pomiędzy kolejnymi wartościami na osi, a także jej początku i końca. W listingu poniżej zaprezentowana została zmiana interwału oraz wartości początkowej osi. Dokonuje się tego w kodzie C#, gdyż wartości te są typu *Nullable*, co uniemożliwia ich ustawienie bezpośrednio z kodu XAML bez użycia konwerterów.

Listing 26: Modyfikacja osi wartości.

```
<charting:Chart x:Name="Chart" Loaded="Chart_Loaded">
...
```

```
private void Chart_Loaded(object sender, RoutedEventArgs e)
{
    (Chart.ActualAxes[1] as LinearAxis).Interval = 1.0;
    (Chart.ActualAxes[1] as LinearAxis).Minimum = 0.0;
}
```

Do reprezentowania czasu (zarówno daty jak i godzny) służy oś typu `DateTimeAxis`. Aby jej użyć konieczne jest zadeklarowanie jej w kontrolce wykresu, jak zostało to zobrazowane w listingu 27.

Listing 27: Deklarowanie osi DateTimeAxis.

```
<charting:Chart x:Name="Chart" Loaded="Chart_Loaded">
    <charting:Chart.Axes>
        <charting:DateTimeAxis Orientation="X">
            <charting:DateTimeAxis.AxisLabelStyle>
                <Style TargetType="charting:DateTimeAxisLabel">
```

```
6                       <Setter Property="StringFormat" Value="{}{0:mm:
                            hh - dd/MM}"/>
7                   </Style>
8               </charting:DateTimeAxis.AxisLabelStyle>
9           </charting:DateTimeAxis>
10      </charting:Chart.Axes>
11      <charting:ColumnSeries IndependentValuePath="Date"
            DependentValuePath="Value"/>
12  </charting:Chart>
```

Moliwe jest także formatowanie opisów wyświetlanych na skali. Służy do tego właściwość `StringFormat`, która należy do stylu stosowanego do etykiet - `DateTimeAxisLabel`. W listingu 27 formatowanie etykiet zostało pokazane w liniach 4-8.

Osobnym zagadnieniem jest rozmieszczenie etykiet na osi typu `DateTimeAxis`. Wymagane jest określenie jednostki interwału orez jego wartości. Zostało to zobrazowane w listingu 28. W podanym przykładzie etykiety umieszczane są na osi w odstępach równych dwóm dniom.

Listing 28: Modyfikacja interwału etykiet na osi DateTimeAxis.

```
1  (Chart.ActualAxes[0] as DateTimeAxis).IntervalType =
       DateTimeIntervalType.Days;
2  (Chart.ActualAxes[0] as DateTimeAxis).Interval = 1.0;
```

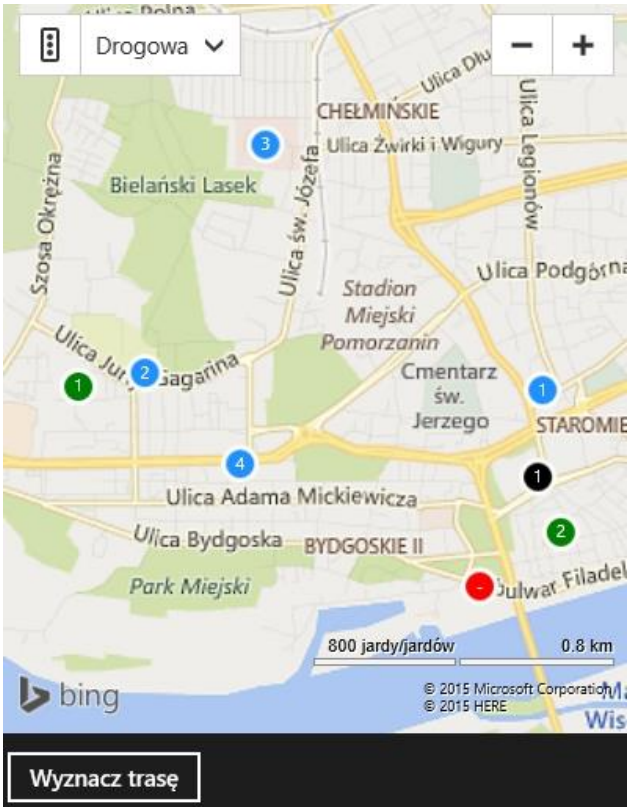
8 Aplikacja

W oparciu o przedstawione w niniejszej pracy zagadnienia dotyczące różnych funkcjonalności dostępnych w przypadku Aplikacji Uniwersalnych, zbudowana została aplikacja o nazwie *Delivery Helper*. Została ona stworzona zarówno dla systemu Windows 8.1, jak i Windows Phone 8.1. Dzięki zastosowaniu projektu typy Universal Apps możliwe było uwspólnienie dużej części kodu, co pozwoliło zmniejszyć ilość pracy potrzebnej do przygotowania aplikacji. Interfejs użytkownika w obu wersjach aplikacji jest zbliżony i oferuje identyczną funkcjonalność.

Aplikacja została zaprojektowana z myślą o użyciu jej przez dostawcę w firmie kurierskiej lub serwisanta w firmie wykonującej usługi u klienta. Aplikacja pozwala na zarządzanie listą zleceń, przedstawianiem miejsc docelowych na mapie z wyznaczeniem optymalnej trasy oraz wizualizuje postęp wykonania zleceń na wykresie. Aplikacja może stanowić pożyteczne narzędzie dla logistyki w firmie.

8.1 Mapa

Głównym elementem interfejsu użytkownika jest mapa. Umieszczone na niej są punkty wskazujące miejsca docelowe, a także aktualną pozycję uzyskaną w procesie geolokacji. Lokalizacje zleceń określane są w procesie geokodowania, co zostało opisane w podrozdziale *Wyznaczanie lokalizacji przy użyciu adresu*. Poniżej mapy znajduje się klawisz służący do wyznaczania trasy dla wskazanych miejsc. Kolejność zleceń jest optymalizowana tak, aby uzyskać najkrótszą trasę.

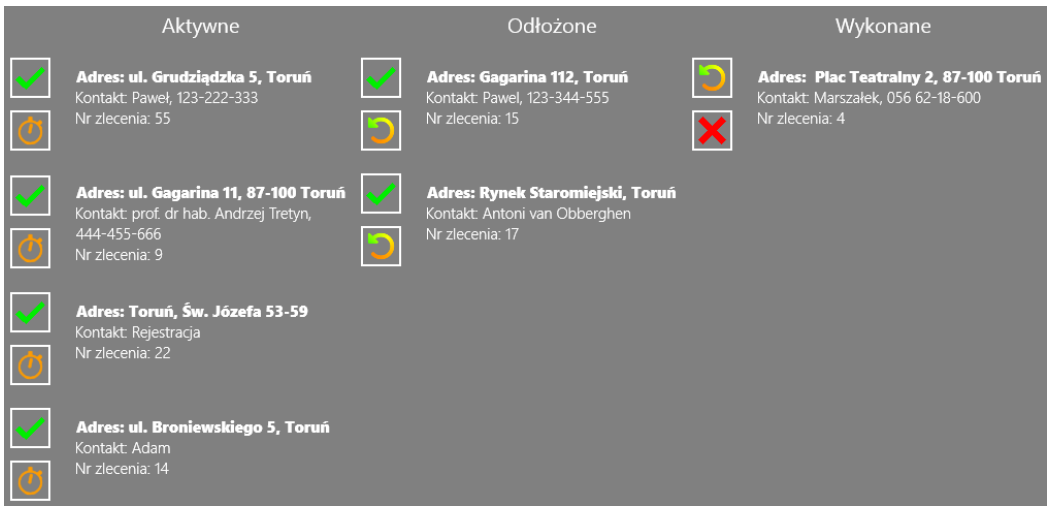


Rys. 7: Strona z mapą.

8.2 Listy zleceń

Zlecenia są pogrupowane w trzech listach: *Aktywne*, *Odłożone* oraz *Wykonane*. Każdemu aktywnemu zleceniu odpowiadają przyciski pozwalające na zmianę statusu zlecenia poprzez przesunięcie go do listy zleceń wykonanych bądź odłożonych na później. Każdemu odłożonemu zleceniu odpowiadają przyciski służące do oznaczenia zlecenia jako wykonane oraz do przywrócenia go do listy zleceń aktywnych. Trzecia lista grupuje zlecenia już zrealizowane. Odpowiadające im przyciski pozwalają na przywrócenie zlecenia

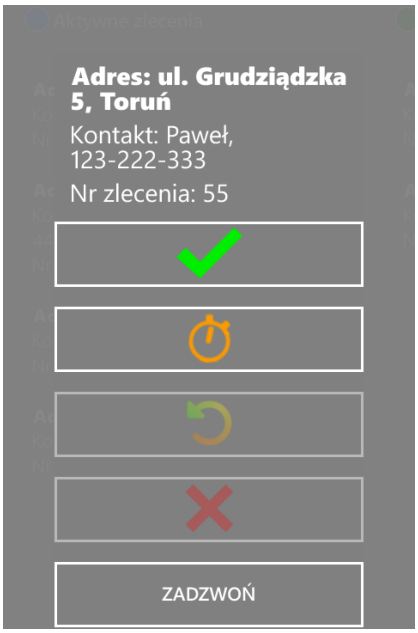
do aktywnych oraz usunięcie zlecenia z programu. Przycisk usuwania zabezpieczony jest przed przypadkowym użyciem przez konieczność powtórnego kliknięcia.



Rys. 8: Strona z listami zleceń w wersji dla Windows 8.1.

W wersji dla Windows 8.1 przyciski znajdują się obok opisów zleceń. W wersji dla Windows Phone 8.1 przyciski stają się widoczne dopiero po kliknięciu na wybrane zlecenie. Na wywołanym ekranie, oprócz opisanych powyżej przycisków, znajduje się także przycisk pozwalający na wybranie numeu telefonu, o ile ten został dodany do opisu zlecenia.

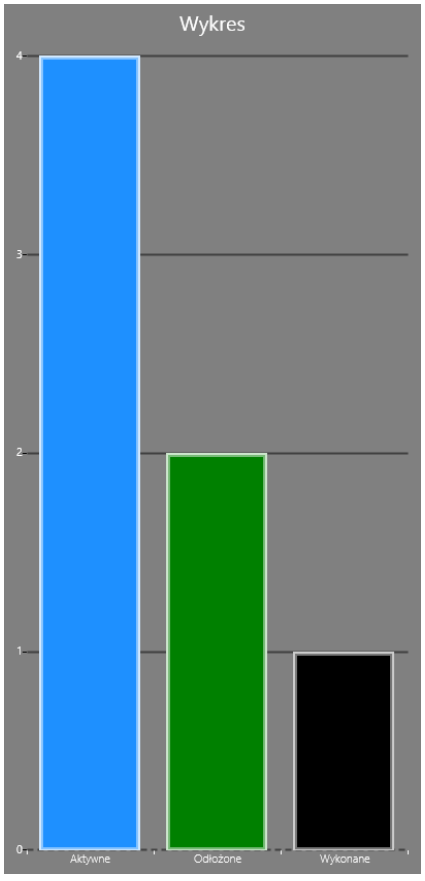
Rys. 9: Strona z listą zleceń aktywnych w wersji dla Windows Phone 8.1.



Rys. 10: Szczegóły zlecenia z opcją wyboru numeru telefonu.

8.3 Wizualizacja liczby zleceń

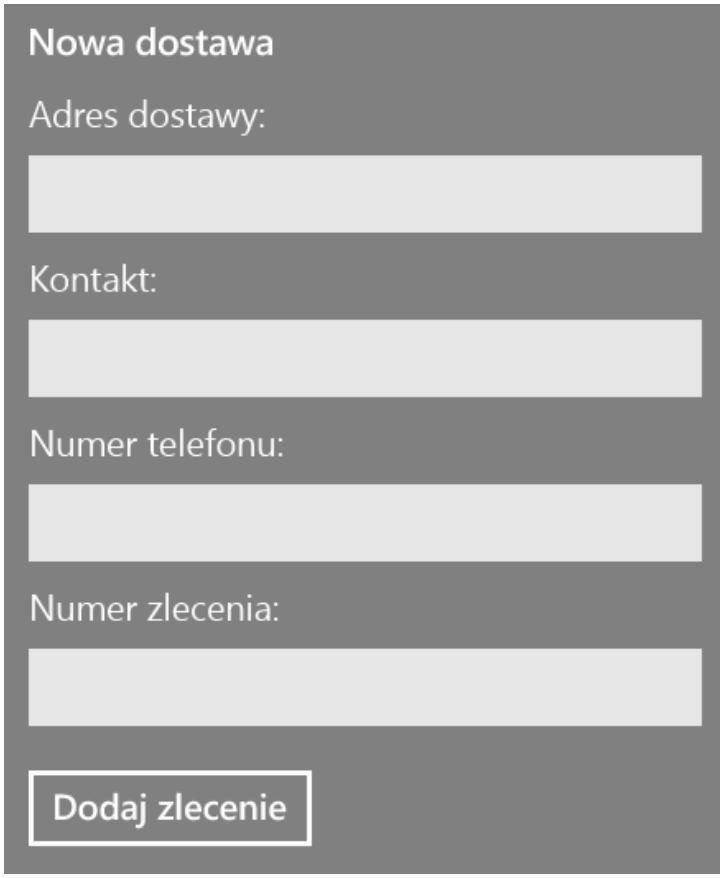
Strona prezentująca liczbę zleceń zawiera wykres słupkowy, którego tworzenie zostało opisane w rozdziale 7. Przedstawia liczbę zleceń każdego typu. Kolorystyka wykresu jest odzwierciedleniem kolorów pinezek użytych do lokalizacji poszczególnych typów zleceń na mapie. Wykres jest automatycznie odświeżany po każdej zmianie w listach zleceń.



Rys. 11: Strona wizualizacji w wersji dla Windows 8.1.

8.4 Wprowadzanie nowych zleceń

Ostatnią stroną w aplikacji jest formularz służący do wprowadzania danych o nowych zleceniach, który zawiera pola do wprowadzenia adresu, osoby do kontaktu, numeru telefonu oraz numeru zlecenia. Wprowadzony numer zlecenia może służyć do identyfikacji przesyłki lub dokumentu zlecenia usługi. Poprzez kliknięcie przycisku *Dodaj zlecenie* generowane jest nowe zlecenie, które jest umieszczane w liście aktywnych.



Rys. 12: Formularz wprowadzania nowych zleceń w wersji dla Windows Phone 8.1.

9 Podsumowanie

W pracy omówiono zagadnienia dotyczące aplikacji uniwersalnych, w szczególności:

- przechowywanie danych w bazach SQLite,
- umieszczanie map w aplikacji,
- używanie geolokacji,
- prezentacja danych.

Stosowanie opisanych w pracy metod pozwala współdzielić kod między dwiema wersjami aplikacji, a także używać go w innych projektach, co prowadzi do zmniejszenia nakładu pracy. Towarzysząca pracy aplikacja uwzględnia i implementuje opisane zagadnienia i metody, a przy tym ma wartość użytkową.

Z obszernego tematu jakim są Aplikacje Uniwersalne opisane zostały wybrane zagadnienia. Praca ta jest elementem całego tutorialu, na który składają się także inne prace dyplomowe studentów Wydziału Fizyki Astronomii i Informatyki Stosowanej UMK. Wyodrębnione w tej pracy zagadnienia Aplikacji Uniwersalnych demonstrują skuteczne metody współdzielenia kodu przeznaczonego dla Windows 8.1 i Windows Phone 8.1 w tworzeniu i zarządzaniu bazami danych, używaniu map, geolokacji i wizualizacji danych. W celu poszerzenia praktycznej wiedzy dotyczącej innych zagadnień związanych z Aplikacjami Uniwersalnymi, należy sięgnąć do pozostałych prac dyplomowych studentów Wydziału Fizyki Astronomii i Informatyki Stosowanej UMK.

10 Literatura

[1] Windows Dev Center, *App capability declarations*, *https://msdn.microsoft.com/en-us/library/windows/apps/mt270968.aspx*, [dostęp 20 października 2015]

[2] P. Bright, *Turning to the past to power Windows' future: An in-depth look at WinRT*, *http://arstechnica.com/features/2012/10/windows-8-and-winrt-everything-old-is-new-again*, [dostęp 20 października 2015]

[3] Oficjalna strona SQLite, *About SQLite*, *http://sqlite.org/about.html*, [dostęp 20 października 2015]

[4] R. Brundritt, *How to Make Use of Maps in Universal Apps*, *http://blogs.msdn.com/b/rbrundritt/archive/2014/06/24/how-to-make-use-of-maps-in-universal-apps.aspx*, [dostęp 20 października 2015]

[5] Windows Dev Center, *MapControl class*, *https://msdn.microsoft.com/en-us/library/windows/apps/windows.ui.xaml.controls.maps.mapcontrol.aspx*, [dostęp 20 października 2015]

[6] Microsoft Developer Network, *Bing Maps for Windows Store Apps*, *https://msdn.microsoft.com/en-us/library/hh846481.aspx*, [dostęp 20 października 2015]

[7] Windows Dev Center, *Geolocator class*, *https://msdn.microsoft.com/en-us/library/windows/apps/windows.devices.geolocation.geolocator*, [dostęp 20 października 2015]

[8] Strona WinRT XAML Toolkit, *https://winrtxamltoolkit.codeplex.com/*, [dostęp 20 października 2015]

[9] A. A. Eren, *Using Graphs and Charts in Windows Store Apps*, *http://eren.ws/2013/10/15/using-graphs-and-charts-in-windows-store-apps*, [dostęp 20 października 2015]