Uwagi

1: Tu bym zrobił cytowanie bibliograficzne.
3: Na razie proszę się tym nie przejmować. Zajmiemy się tym, gdy będzie więcej treści.
5: bez przecinka
5: bez przecinka
6: Zamiast przypisu zrobiłbym cytowanie.
6: tu podobnie
6: bez przecinka
6: SVN też umożliwia rozgałęzianie
6: zamiast «czyli tzw.» napisałbym «realizowana jako»
6: bez przecinka
6: przecinek
6: zamiast «tzw. *Cross Origin Requests*, czyli zapytania z innej domeny» napisałbym «zapytania z innej domeny (*Cross Origin Requests*)».
7: «Owe modele» → «Modele (te)»
7: zamiast «typu *Single Page Application*, czyli aplikacji internetowych, z których można korzystać bez przeładowania strony» napisałbym «internetowych, z których można korzystać bez przeładowania strony (*Single Page Application*)»
8: zamiast przypisu: «(*Model View ViewModel*, *vide* rozdział 4.1.3)»
8: cytowanie do manuala/strony www/...
9: spacja
9: nie dywiz '-', ale pauza '–'
9: przecinek
9: «świat» wydaje się OK
9: bez spacji przed przecinkiem
10: ,
10: ,
10: ,
10: to jakoś trzeba będzie napisać po polsku
10: bez przecinka
10: bez przecinka
10: przecinek
10: zamiast «jako *Single Page Application*, czyli bez przeładowania strony» napisałbym «bez przełądowania strony (*Single Page Application*)»
11: rozwinięcie angielkiego akronimu
11: zamiast «następnie łączone» napisałbym «połączone»
11: Tu przydałby się rysunek modelu MVVM.
11: bez przecinka
11: Dla spójności napisałbym słowo *commands* wielką literą.
11: wiążących
11: wiążące
11: To zdanie jest trochę niejasne, warto by je nieco rozwinąć.
11: bez przecinka
11: wiązania / wiążącego

Politechnika Śląska
Wydział Automatyki, Elektroniki i Informatyki

Katarzyna Biernat

Wirtualny przewodnik dla urządzeń mobilnych

projekt inżynierski

Promotor/kierujący pracą: dr inż. Krzysztof Simiński

Gliwice, 9 grudnia 2014

Spis treści

Rozdział 1

Wstęp

1.1 Temat

1.2 Cel

Celem niniejszego projektu jest stworzenie przyjaznej platformy dedykowanej dla osób podróżujących. Program ma ułatwić użytkownikowi zwiedzanie nowych miejsc oraz zupełnie zastąpić tradycyjny, książkowy przewodnik. Na całość składa się aplikacja mobilna na platformę Windows Phone 8.1 oraz aplikacja działająca w przeglądarce internetowej.

1.3 Motywacja

Obecnie na rynku istnieje nisza, jeśli chodzi o tego typu rozwiązania. W Internecie można znaleźć bardzo wiele różnych informacji o ciekawych miejscach turystycznych, brakuje jednak przyjaznej formy przedstawienia ich podróżnikowi.

Rysunek 1.1 przedstawia wyniki ankiety przeprowadzonej wśród 234 użytkowników forum Fly4Free.pl [1] [Tu bym zrobił cytowanie bibliograficzne.]. Jak się okazuje, zdecydowana większość podróżników zakupuje przewodnik o miejscu docelowym bądź pobiera stosowne informacje z sieci i drukuje je na swój użytek.

Aplikacja *TravelGuide* ma na celu połączenie zalet obu tych rozwiązań. Reprezentuje informacje w podobny sposób jak tradycyjny, książkowy przewodnik przy jednoczesnej możliwości szybkiej aktualizacji informacji. Ponadto dzięki

Rysunek 1.1: Wyniki ankiety przeprowadzonej wśród użytkowników forum fly4free.pl

temu, że zawiera się w smartfonie podróżnika, zwalnia go z noszenia ciężkich książek lub nieporęcznych drukowanych kartek.

Dzięki wykorzystaniu nowoczesnych technologii, aplikacja może podawać użytkownikowi najciekawsze informacje bazując na jego danych geolokalizacyjnych.

Rozdział 2

Wprowadzenie do problemu

[Na razie proszę się tym nie przejmować. Zajmiemy się tym, gdy będzie więcej treści.]

Rozdział 3

Technologia

3.1 Narzędzia

3.1.1 Środowisko programistyczne

Całość projektu została zrealizowana za pomocą Visual Studio 2013. Jest to jedno z popularniejszych środowisk programistycznych, wyprodukowane przez firmę Microsoft. Mimo,[bez przecinka] że nie jest to środowisko bezpłatne, projekt mógł zostać zrealizowany bez ponoszenia kosztów dzięki licencji akademickiej *DreamSpark*.

Visual Studio 2013 w pełni wspiera rozwijanie oprogramowania w platformie .NET; zarówno rozwiązania mobilne jak i internetowe. Poza tym, dzięki szeregowi dostępnych bibliotek, wspomaga pracę z HTML, LESS oraz JavaScript.

Środowisko to zostało wybrane ze względu na następujące zalety:

- bardzo dobra obsługa *IntelliSense* dla języka C#;
- wiele pomocnych rozszerzeń, np. *Web Essentials*;
- wsparcie dla składni języków HTML[1], CSS[2] i JavaScript;
- wbudowany emulator Windows Phone;
- gotowe szablony, które przyspieszają pracę z projektem.

3.1.2 System kontroli wersji

Pomimo,[bez przecinka] że projekt realizowany był indywidualnie, zdecydowano się na wykorzystanie systemu kontroli wersji. Na rynku dostępnych jest wiele takich systemów. Do najpopularniejszych należą: GIT [2] [Zamiast przypi-

[1] *HyperText Markup Language* - hipertekstowy język znaczników wykorzystywany do tworzenia stron internetowych.

[2] *Cascading Style Sheets* - kaskadowe arkusze stylów służące do opisu formy prezentacji strony WWW.

su zrobiłbym cytowanie.], Mercurial[3] [tu podobnie], SVN[4], Bazaar[5] oraz TFS[6]. Podczas projektu zdecydowano się na wykorzystanie systemu GIT. Został on wybrany z wielu powodów. Jest to system rozproszony, więc jego repozytoria są relatywnie małe (np. w porównaniu do SVN). Ponadto,[bez przecinka] GIT oferuje wiele funkcjonalności, które ułatwiają pracę z projektem, m.in. tzw. gałęzie [SVN też umożliwia rozgałęzianie], etykiety czy lokalną przestrzeń roboczą. Nie bez znaczenia był także fakt, że autorka ma doświadczenie z tym systemem kontroli wersji. Repozytorium zostało umieszczone w serwisie Github.com[7], który agreguje projekty z całego świata. Portal oferuje przejrzysty interfejs oraz możliwość graficznego przeglądania historii projektu.

3.2 Technologia wykorzystana w części serwerowej

Część serwerowa, [zamiast «czyli tzw.» napisałbym «realizowana jako»] czyli tzw. WebService, wykonana została z wykorzystaniem platformy ASP.NET WebApi 2[8]. Całość opiera się na platformie .NET 4.5 i napisana jest w języku C#.

W tej części wykorzystano szereg bibliotek. Aby przyspieszyć pracę i zapewnić większe bezpieczeństwo aplikacji, do celów identyfikacji i uwierzytelniania użytkowników została wykorzystana biblioteka AspNet.Identity[9].

Do pracy z kolekcjami wykorzystano bibliotekę LINQ. Jest to zestaw niezwykle przydatnych narzędzi, które ułatwiają pracę z takimi strukturami jak listy, kolejki i inne implementujące interfejs `IEnumerable`.

Nieodzowna okazała się być biblioteka Newtonsoft.Json[10], która wspomaga formatowanie obiektów na kod JSON[11]. Wszystkie akcje zwracające dane zwracają je właśnie w tym formacie. Komunikat w formacie JSON jest literałem obiektu języka JavaScript. Dane przekazywane są jako tablica asocjacyjna a wszystkie dane są zmiennymi.

Jako,[bez przecinka] że WebService działa w innej domenie niż aplikacja użytkownika, niezbędne było dodanie odpowiednich nagłówków HTTP[przecinek] aby umożliwić [zamiast «tzw. *Cross Origin Requests*, czyli zapytania z innej domeny» napisałbym «zapytania z innej domeny (*Cross Origin Requests*)».]. Bardzo ułatwia to biblioteka Microsoft.AspNet.Cors[12]. Dodaje ona szereg na-

[3] Oficjalna strona projektu znajduje się pod adresem `http://mercurial.selenic.com/`
[4] Oficjalna strona projektu znajduje się pod adresem `https://subversion.apache.org/`
[5] Oficjalna strona projektu znajduje się pod adresem `http://bazaar.canonical.com/`
[6] Informacje o projekcie znajdują się pod adresem `http://www.visualstudio.com/en-us/products/tfs-overview-vs.aspx`
[7] Serwis dostępny pod adresem `http://github.com`
[8] Informacje na temat platformy dostępne pod adresem `http://msdn.microsoft.com/en-us/library/dn448365%28v=vs.118%29.aspx`
[9] Biblioteka dostępna pod adresem `https://aspnetidentity.codeplex.com/`
[10] Strona projektu dostępna pod adresem `http://james.newtonking.com/json`
[11] *JavaScript Object Notation* - tekstowy format wymiany danych
[12] Biblioteka do pobrania pod adresem `https://www.nuget.org/packages/Microsoft.`

główków wraz z odpowiednio dobranymi parametrami.

WebService komunikuje się z bazą danych, która działa w oparciu o silnik SQL Server 2014[13]. Komunikacja odbywa się z wykorzystaniem biblioteki Entity Framework[14]. Jest to biblioteka typu ORM[15], która umożliwia mapowanie wyników pobranych z bazy danych na wcześniej przygotowane modele. [«Owe modele» → «Modele (te)»] są następnie mapowane na modele przejściowe (*ViewModels*) za pomocą biblioteki AutoMapper[16].

3.3 Technologia wykorzystana w części internetowej

Aplikacja internetowa napisana została w modelu *client-side*. Oznacza to, że cała aplikacja interpretowana jest po stronie przeglądarki i nie wymaga specjalistycznego serwera. Aby działała poprawnie, wystarczy dowolny serwer HTTP. Kod został napisany przy użyciu następujących języków: Javascript, HTML i LESS[17]. Aplikacja stworzona jest w oparciu o platformę AngularJS[18]. Jest to zestaw otwartych bibliotek języka JavaScript wspierany przez firmę Google. Umożliwia tworzenie aplikacji [zamiast «typu *Single Page Application*, czyli aplikacji internetowych, z których można korzystać bez przeładowania strony» napisałbym «internetowych, z których można korzystać bez przeładowania strony (*Single Page Application*)»]. Dane ładowane są w tle dzięki asynchronicznym zapytaniom do WebService realizowanym przez JavaScript.

Zostały również zastosowane poboczne biblioteki. Do najważniejszych zależą:

LeafletJS [19] – umożliwia sprawne dodawanie map do strony. Do wyświetlania map wybrany został silnik OpenStreetMap[20] ze względu na swoją otwartą licencję.

jQuery [21] – jest to zestaw bibliotek pomocniczych dla języka JavaScript. Rozszerza funkcjonalność i upraszcza składnię. W prostszy sposób pozwala na dostęp i manipulację elementami DOM.

Bootstrap [22] – zestaw bibliotek, który dołączony jest do Twitter Bootstrap. Dostarcza takie moduły jak np. interaktywne wyskakujące okna, animacje zamknięcie i in.

`AspNet.Cors/`

[13] Opis rozwiązania dostępny pod adresem `http://www.microsoft.com/pl-pl/server-cloud/products/sql-server/`

[14] Opis biblioteki dostępny pod adresem `http://msdn.microsoft.com/en-us/data/ef.aspx`

[15] *Object-Relational Mapping* - mapowanie obiektowo-relacyjne.

[16] Biblioteka dostępna pod adresem `https://github.com/AutoMapper/AutoMapper`

[17] *Leaner CSS* – dynamiczny język arkuszy stylów.

[18] Oficjalna strona projektu dostępna pod adresem `https://angularjs.org/`

[19] Oficjalna strona biblioteki znajduje się pod adresem `http://leafletjs.com/`

[20] Oficjalna strona projektu znajduje się pod adresem `http://www.openstreetmap.org/`

[21] Oficjalna strona biblioteki znajduje się pod adresem `http://jquery.com/`

[22] Oficjalna strona biblioteki znajduje się pod adresem `http://getbootstrap.com/`

AngularTranslate.js [23] – biblioteka rozszerzająca możliwości platformy Angular o implementację wielojęzyczności.

Za część graficzną aplikacji odpowiada zmodyfikowany przez autorkę szablon AdminLTE[24] napisany w oparciu o Twitter Bootstrap. Szablon opisany jest językiem LESS, który jest następnie transformowany do CSS.

3.4 Technologia wykorzystana w części mobilnej

Na część mobilną składa się aplikacja dedykowana dla platformy Windows Phone 8.1. Projekt zrealizowany został w modelu Universal Apps[25], więc cała logika wydzielona jest do osobnego projektu.

Aplikacja wykonana została przy wykorzystaniu platformy .NET, języka C# oraz XAML. Zaimplementowany został wzorzec architektoniczny MVVM [zamiast przypisu: «(*Model View ViewModel*, *vide* rozdział 4.1.3)»][26]. W tym celu do projektu dołączona została biblioteka MVVM Light [cytowanie do manuala/strony www/...].

Aplikacja pobiera dane z WebService i zapisuje je w swojej lokalnej bazie danych. Działa ona dzięki SQLight[27]. Do projektu dołączony jest adapter SQLite. Pobrane dane parsowane są do obiektów za pomocą biblioteki Newtonsoft.JSON.

[23] Oficjalna strona biblioteki znajduje się pod adresem `http://angular-translate.github.io/`

[24] Szablon dostępny pod adresem `http://almsaeedstudio.com/AdminLTE/`

[25] `http://dev.windows.com/en-us/develop/building-universal-windows-apps`

[26] *Model View ViewModel* - informacje na temat wzorca w rozdziale 4.1.3

[27] Strona projektu dostępna pod adresem `http://www.sqlite.org/`

Rozdział 4

Specyfikacja wewnętrzna

4.1 Architektura

Na cały system składają się trzy aplikacje: aplikacja mobilna (*Mobile app*), aplikacja internetowa (*Website*) oraz aplikacja serwerowa[spacja](*WebService*). Związek pomiędzy tymi aplikacjami przedstawiony jest na rysunku 4.1.

4.1.1 Architektura aplikacji serwerowej

Aplikacja serwerowa stworzona jest przy wykorzystaniu platformy ASP.NET WebApi 2 i zgodnie ze wzorcem architektonicznym REST[1]. Wzorzec ten zakłada kilka warunków:

Klient-serwer - [nie dywiz '-', ale pauza '–'] aplikacja powinna składać się z dwóch odrębnych części: klienta i serwera. Klient nie powinien mieć dostępu do magazynu danych. Serwer z kolei nie powinien zawierać informacji na temat interfejsu użytkownika. I tak też aplikacja serwerowa systemu *TravelGuide* jest jedyną, która ma dostęp do bazy danych. Nie zawiera ona żadnych widoków[przecinek] a ze światem [«świat» wydaje się OK] komunikuje się za pomocą zapytań HTTP[2].

Bezstanowość - każde żądanie HTTP (*request*) wysyłane do serwera jest niezależne od pozostałych. I tak też w aplikacji serwerowej systemu *TravelGuide* ,[bez spacji przed przecinkiem] za odbiór żądań odpowiada warstwa kontrolera. Każda akcja kontrolera obsługuje jedno żądanie. Akcje są od siebie niezależne.

[1] *Representational State Transfer*
[2] *Hypertext Transfer Protocol* – protokół przesyłania dokumentów hipertekstowych

Rysunek 4.1: Architektura systemu

Rysunek 4.2: Architektura aplikacji serwerowej

Rysunek 4.3: Architektura aplikacji internetowej

Wielowarstwowość - klient nie wie[,] czy jest podłączony bezpośrednio do serwera[,] czy też otrzymuje informacje przez systemy pośrednie. Dzięki takiej skalowalności, systemy pośrednie mogą zostać wykorzystane[,] aby zwiększyć wydajność całości. Aplikacja serwerowa systemu *TravelGuide* przyjmuje żądania HTTP a odpowiedź wysyła w formacie JSON. Można więc dołożyć kolejne systemu pośrednie zgodnie z zapotrzebowaniem.

Cache'owalność - [to jakoś trzeba będzie napisać po polsku] serwer powinien zwracać informację, które z odpowiedzi mogą zostać zapisane w pamięci podręcznej cache przeglądarki. W przypadku aplikacji serwerowej systemu *TravelGuide* nie jest to zaimplementowane, gdyż żadna z odpowiedzi nie powinna być zapisywana.

Wewnętrznie,[bez przecinka] aplikacja składa się z wielu warstw, które komunikują się ze sobą. Każde odebrane zapytanie HTTP jest obsługiwane przez odpowiednią akcję kontrolera. Zapytania realizowane są czterema standardowymi metodami HTTP: *GET*, *POST*, *PUT*, *DELETE*.

Zazwyczaj,[bez przecinka] akcja wymaga interakcji z bazą danych. Kontroler nie komunikuje się z nią jednak bezpośrednio. Zamiast tego składa odpowiednio sparametryzowane zapytanie do warstwy repozytorium. Parametrem może być cyfra (`Integer`), ciąg znaków (`String`) lub obiekt typu modelu bindującego (*Binding Model*). Parametr może być także pusty.

Repozytorium przetwarza żądanie. Jeżeli parametrem był obiekt typu modelu bindującego mapuje go na model[przecinek] a następnie za pomocą *Entity Framework* komunikuje się z bazą danych. Rezultat jest następnie ponownie mapowany na model bindujący i przekazywany do akcji kontrolera. Kontroler zwraca wynik w postaci odpowiedzi HTTP (*HTTP Response*).

Schemat architektury aplikacji mobilnej przedstawiony jest na rysunku 4.2.

4.1.2 Architektura aplikacji internetowej

Aplikacja internetowa została stworzona przy użyciu platformy AngularJS i w całości działa [zamiast «jako *Single Page Application*, czyli bez przeładowania strony» napisałbym «bez przełądowania strony (*Single Page Application*)»]. Zawiera główny widok napisany w języku HTML, do którego w zależności od kontekstu załączane są widoki cząstkowe (*partial views*). Z każdym widokiem powiązany jest kontroler.

Kiedy użytkownik wchodzi w interakcję z aplikacją, jego akcje są wysyłane do odpowiednich akcji kontrolera. Kontroler następnie obsługuje daną operację

Rysunek 4.4: Architektura aplikacji mobilnej

lub, jeżeli jest ona bardziej skomplikowana, odwołuje się do odpowiedniej metody z warstwy serwisu. Jeżeli wymagane jest pobranie danych z WebService, warstwa serwisu komunikuje się z warstwą repozytorium, która następnie wysyła odpowiednio skonstruowane zapytanie. Otrzymany wynik jest przekazywany z powrotem do warstwy serwisu, przetwarzany, przekazywany do kontrolera i bindowany z widokiem.

Schemat architektury aplikacji mobilnej przedstawiony jest na rysunku 4.3.

4.1.3 Architektura aplikacji mobilnej

Aplikacja mobilna dedykowana jest dla platformy Windows Phone 8.1. Została stworzona przy użyciu wzorca MVVM [rozwinięcie angielkiego akronimu]. Wzorzec ten polega na rozdzieleniu warstwy widoku (*View*) od warstwy danych (*Model*). Są one następnie łączone [zamiast «następnie łączone» napisałbym «połączone»] za pomocą warstwy pośredniej (*ViewModel*), która zawiera większą część logiki związanej z prezentacją danych. [Tu przydałby się rysunek modelu MVVM.]

Program przy pierwszym uruchomieniu łączy się z aplikacją serwerową i pobiera dane, do których uprawniony jest zalogowany użytkownik. Dane te są zapisywane w lokalnej bazie danych obsługiwanej przez silnik SQLite.

Użytkownik wchodzi w interakcję z aplikacją przy pomocy zachowań typowych dla obsługi aplikacji mobilnych. Są to gesty takie jak dotknięcie ekranu, dłuższe przytrzymanie ekranu czy posuwisty ruch dłonią po ekranie smartfona. Po stronie aplikacji,[bez przecinka] odbiorcą takich poleceń jest warstwa widoku (*View*). Komunikuje się ona następnie z warstwą pośrednią (*ViewModel*) wprost lub za pomocą konwerterów (*Converter*). Komunikacja odbywa się dzięki wykorzystaniu poleceń (*commands* [Dla spójności napisałbym słowo *commands* wielką literą.]).

Warstwa pośrednia realizuje zadanie zadane przez użytkownika. Jeżeli wymaga to komunikacji z bazą danych, łączy się ona z warstwą repozytorium (*Repository*). Repozytorium operuje na dwóch typach obiektów: na modelach (*Model*) i na modelach bindujących [wiążących] (*Binding model*). Modele odwzorowują schemat bazy danych w postaci obiektów języka C# i są wykorzystywane w komunikacji z nią. Modele bindujące [wiążące] to modele zmodyfikowane na potrzeby danego zadania [To zdanie jest trochę niejasne, warto by je nieco rozwinąć.]. Za komunikację z aplikacją serwerową odpowiada warstwa serwisów (*Service*), która wysyła do niej odpowiednie zapytania w formacie RESTful.

Warstwa pośrednia nie wie o istnieniu modeli. Od repozytorium otrzymuje ona jedynie kolekcje modeli bindujących. Finalnie,[bez przecinka] są one przetwarzane i przekazywane do warstwy widoku za pomocą mechanizmu bindowania [wiązania / wiążącego] (*bindings*).

Schemat architektury aplikacji mobilnej przedstawiony jest na rysunku 4.4.

4.2 Najważniejsze algorytmy

4.3 Przypadki użycia

Rozdział 5

Specyfikacja zewnętrzna

5.1 Instrukcja obsługi

5.2 Wymagania

5.3 Instalacja

Rozdział 6

Testowanie

Rozdział 7

Wnioski końcowe

Rozdział 8

Zakończenie

Bibliografia

[1] Fly4free.pl. http://www.fly4free.pl/forum/wasze-sposoby-na-planowanie-zwiedzania-nowych-miejsc,18,59116 [data dostępu: 6 grudnia 2014].

[2] Git. http://git-scm.com/.