**UNIWERSYTET PEDAGOGICZNY**

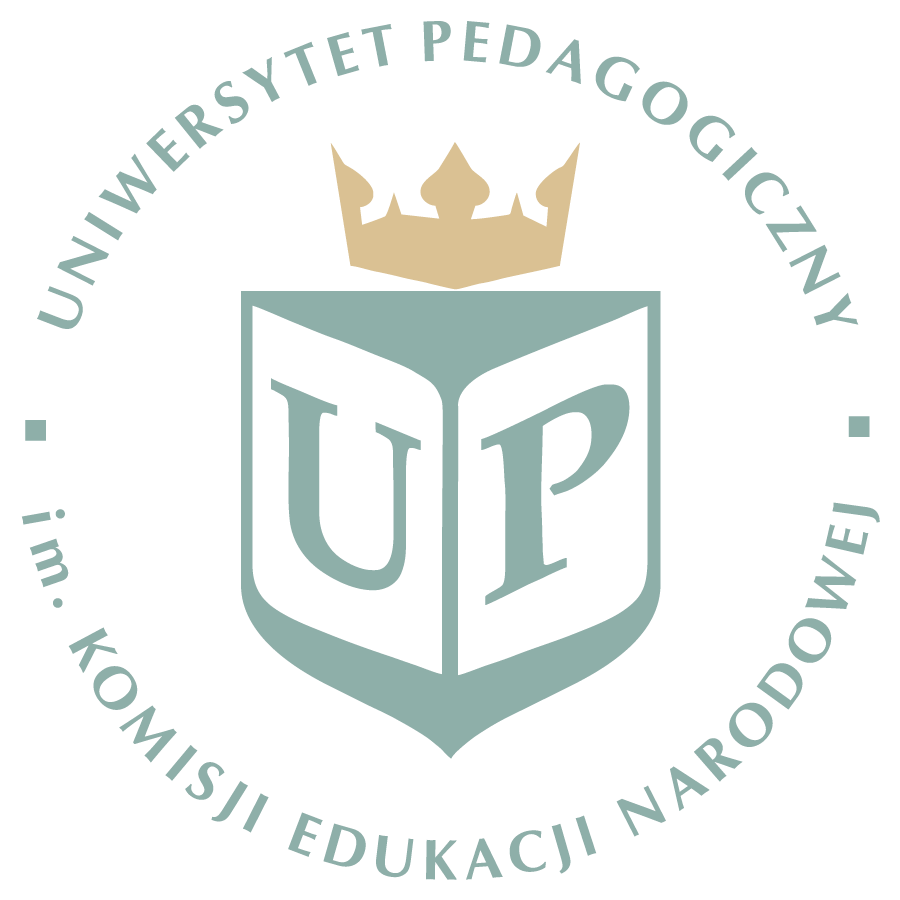
**im. Komisji Edukacji Narodowej**

**W KRAKOWIE**

**Wydział Matematyczno-Fizyczno-Techniczny**

**Instytut Informatyki**

Kierunek Informatyka



**JAKUB KRZEMIŃSKI**

**Serwisy webowe REST w środowisku .net  
 REST web services in .net**

Praca magisterska

napisana pod kierunkiem

dr. Krzysztofa Mazeli

**KRAKÓW 2018**

Spis Treści

[Spis Treści 2](#_Toc524377723)

[Wstęp 3](#_Toc524377724)

[1. Representational State Transfer - REST 4](#_Toc524377725)

[Założenia architektoniczne 4](#_Toc524377726)

[Interfejs Programowania Aplikacji 6](#_Toc524377727)

[Adres URL 7](#_Toc524377728)

[Zapytania HTTP 7](#_Toc524377729)

[Metody zapytań 9](#_Toc524377730)

[Statusy odpowiedzi protokołu HTTP 10](#_Toc524377731)

[2. .NET Framework 11](#_Toc524377732)

[Historia 11](#_Toc524377733)

[Cechy platformy .NET 13](#_Toc524377734)

[Technologie w .NET 14](#_Toc524377735)

[3. Biblioteki implementujące REST 18](#_Toc524377736)

[Flurl.Http 18](#_Toc524377737)

[RestSharp 19](#_Toc524377738)

[4. Autorska implementacja biblioteki 20](#_Toc524377739)

[Założenia projektowe 20](#_Toc524377740)

[Baza danych 20](#_Toc524377741)

[Visual Studio 2017 22](#_Toc524377742)

[C ♯ (C Sharp) 23](#_Toc524377743)

[Web Serwis 24](#_Toc524377744)

[Aplikacja kliencka 27](#_Toc524377745)

[Biblioteka RESTLib 29](#_Toc524377746)

[Testy efektywności systemu 30](#_Toc524377747)

[5. Podsumowanie 32](#_Toc524377748)

[Bibliografia 33](#_Toc524377749)

[Spis rysunków 35](#_Toc524377750)

Wstęp

Przeciętny użytkownik aplikacji bankowej nie zastanawia się na temat tego, w jaki sposób program, z którego korzysta, komunikuje się z serwerem. Większość aplikacji, dostępnych w Internecie, pobiera aktualne informacje dzięki zastosowaniu web serwisów. Usługi sieciowe (*web services*) są to programy komputerowe, które komunikują się z innymi aplikacjami, dostarczając żądanej funkcjonalności [1]. Web serwisy najczęściej przekazują informacje w formie JSON za pomocą protokołu HTTP.

Celem pracy jest utworzenie biblioteki programistycznej, wspomagającej tworzenie usług sieciowych udostępniających REST API w środowisku .NET Framework.

Niniejsza publikacja została podzielona na pięć części:

1. Rozdział przedstawiający podstawowe informacje oraz założenia technologii HTTP oraz architektury REST.
2. Informacje o platformie .NET Framework, dla której implementowany będzie projekt.
3. Przykłady istniejących bibliotek, które pozwalają na komunikację z REST API.
4. Opis stworzonego systemu, który składa się z web serwisu oraz aplikacji klienckiej, a komunikacja między nimi zachodzi poprzez REST API.
5. Testy systemu, które obrazują wydajność zastosowanych mechanizmów.

Wszystkie trudne pojęcia, które pojawią się w tekście, będą tłumaczone na bieżąco lub ich opisy pojawią się w przypisach dolnych.

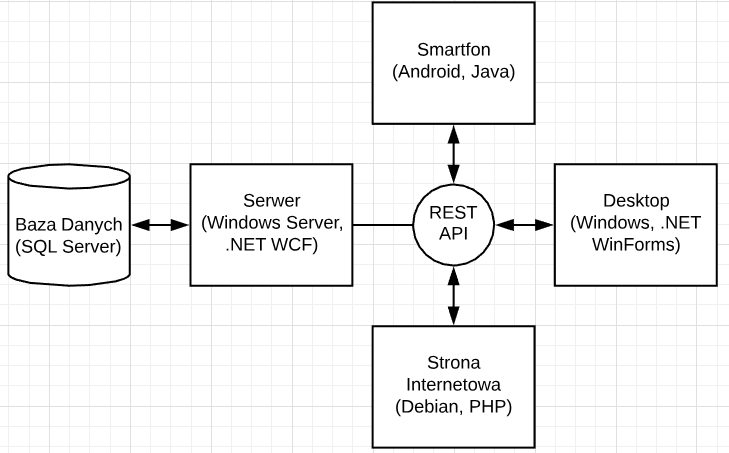
1. Representational State Transfer - REST

REST został opracowany przez Roya T. Fieldinga (jednego z założycieli Apache Software Foundation) w 2000 roku oraz przedstawiony w dokumencie „*Principled Design of the Modern Web Architecture*”[2]. Styl architektury był opracowywany w latach 1996 – 1999 i bazował na projekcie HTTP[[1]](#footnote-1) 1.0.

Założenia architektoniczne

Architektura REST posiada sześć założeń, które definiują zachowanie aplikacji serwerowej i klienckiej [3]. Pozwala to ograniczyć powstawanie niekorzystnych cech dla budowanego systemu. Do głównych założeń należą:

1. Architektura Klient – Serwer   
   Pozwala na podział ról elementów systemu. Serwer powinien zapewniać obsługę żądań, które zostały wysłane od klientów, co pozwala zwiększyć bezpieczeństwo aplikacji oraz wydajność systemu. Zaletą tego podejścia jest również możliwość tworzenia elementów systemu z zastosowaniem różnych technologii. Przykładem może być pobieranie informacji z bazy danych



Rysunek 1 Architektura klient – serwer, źródło: opracowanie własne

1. Bezstanowość  
   W trakcie wykorzystania serwisu REST, stan musi być przechowywany po stronie klienta. Istnieje możliwość przekazania go na serwer i zwrócenia zmienionego w odpowiedzi. Takie podejście ogranicza możliwość utworzenia sesji użytkownika, lecz poprawia to skalowanie wydajności serwisu REST.
2. Indeksowanie  
   Możliwość indeksowania danych jest jednym z podstawowych mechanizmów zwiększania wydajności odczytu informacji. Polega to na zbieraniu danych, które są często wykorzystywane i przesłaniu ich do pamięci podręcznej (cache) po stronie klienta. Jeżeli zastosowany zostaje ten mechanizm, klient może najpierw pobrać sumę kontrolną i porównać z sumą kontrolną informacji zapisanej w pamięci podręcznej. Jeżeli nie są one identyczne, zachodzi ponowne pobranie aktualnej informacji. Przykładem zastosowania tego mechanizmu jest pobieranie grafik ze stron internetowych, które są rzadko modyfikowane, a ich ponowne pobieranie przy każdym wejściu na stronę internetową może być głównym powodem długiego czasu oczekiwania na pojawienie się zawartości.
3. Jednolity interfejs   
   Komunikacja między aplikacją kliencką a serwerową musi być utworzona w sposób jednolity. Zastosowanie REST w komunikacji między dwoma programami wymaga wykorzystania metod HTTP. Są to między innymi GET, POST, PUT oraz DELETE. Dla ujednolicenia interfejsu, stosuję serializację przesyłanych informacji. Serializacja jest to proces zamiany struktury danych do postaci szeregowej. Informacja przetworzona w ten sposób może być przechowywana miedzy innymi jako strumień bajtów, plik XML lub JSON.
4. Warstwowość  
   Dzięki zastosowaniu warstwowej budowy systemu, można w łatwy sposób wyodrębnić moduły odpowiedzialne za komunikację z klientami, wykonujące logikę biznesową oraz pobieranie informacji z bazy danych.   
   Takie podejście upraszcza również rozproszenie aplikacji serwerowej pomiędzy kilka serwerów, dodając warstwę odpowiedzialną za routing[[2]](#footnote-2).

Dodatkowym założeniem, które nie jest wymagane w web serwisie REST jest:

1. Kod na żądanie  
   Reguła ta polega na umożliwieniu przesyłania kodu przez serwer do aplikacji klienckiej, gdy użytkownik systemu chce uruchomić funkcjonalność. Przykładem jest pobieranie kodu JavaScript[[3]](#footnote-3) i uruchamianie go, gdy użytkownik chce wykonać czynność za jego pośrednictwem.

Interfejs Programowania Aplikacji

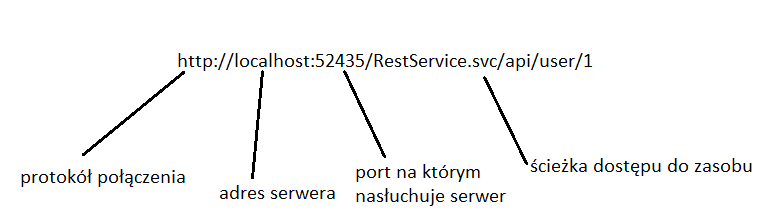
API jest to akronim od *Application Programming Interface* (ang. Interfejs Programowania Aplikacji). Jest to sposób komunikacji miedzy dwoma programami komputerowymi w sposób ustandaryzowany. API posiadają zestawy reguł oraz opisów, które pozwalają programiście w prosty sposób utworzyć funkcje służące do wymiany danych.

Interfejsy tego typu wykorzystywane są w wielu dziedzinach informatyki. Przykładem zastosowania API, jest komunikacja pomiędzy aplikacjami a systemem operacyjnym za pomocą POSIX API. W celu komunikacji miedzy programem napisanym w języku C, a elementami systemu dostępnymi przez POSIX, można wykorzystać funkcje z biblioteki C POSIX[[4]](#footnote-4).

REST API jest typem Web API, który służy do komunikacji poprzez sieć miedzy dwoma programami. Zastosowanie takiej komunikacji nie wymaga połączenia sieciowego, jeżeli aplikacje uruchomione są na tej samej maszynie (naprzykład podczas tworzenia systemu lub gdy aplikacja kliencka jest również uruchomiona na serwerze). Komunikacja z wykorzystaniem REST API polega na wysłaniu zapytania HTTP na URL web serwisu.

Adres URL

Każdy web serwis oparty o REST API, posiada adresy, które odwołują się do zasobów dostępnych w aplikacji. Adres sieciowy URL[[5]](#footnote-5) składa się z  następujących elementów [6]:



Rysunek 2 Struktura adresu URL, źródło: opracowanie własne

Zapytania HTTP

Serwisy oparte o REST wykorzystują zapytania protokołu HTTP. Zapytania składają się z dwóch części: nagłówka (*header*) oraz ciała (*body*).   
GET/HTTP/1.1,   
gdzie:  
GET – nazwa metody jaka zostaje wykorzystana

HTTP/1.1 - wybrany protokół

Warto podkreślić, że pod przedstawioną linią kodu można dodać kolejne elementy nagłówka, należy pamiętać jednak, aby nie pozostawić pustej linii między fragmentami nagłówka. Przerwa taka oznacza zakończenie *headera* oraz rozpoczyna *body* zapytania.

W nagłówku możemy zawrzeć następujące elementy:

1. HTTP Referer – jest to adres odsyłający do miejsca, z którego nastąpiło przekierowanie do serwisu.

Referer: https://www.czektul.pl/

1. USER Agent – informacja o aplikacji klienckiej, z której wysłano zapytanie oraz podstawowe informacje o urządzeniu. Przykładem może być przeglądarka internetowa.

Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0

1. X-Robots-Tag – informacja dla robotów o możliwości indeksacji zawartości witryny w przeglądarkach stron internetowych.

X-Robots-Tag "noindex, nofollow"

1. Server – są to dane na temat aplikacji znajdującej się na serwerze, która umożliwia połączenie z serwisem.

Server: Apache/2.4.1 (Unix)

1. Cache-control – wpis w nagłówku, który informuje aplikację kliencką o możliwości indeksacji danego zasobu w pamięci podręcznej.

Cache-Control: public, max-age=31536000

1. Content-Type – informacja o typie przesyłanych informacji.

Content-Type: application/json

1. Accept – nagłówek tego typu przekazuje serwerowi informacje o formacie informacji, która jest oczekiwana w aplikacji klienckiej jako zwrotna. Nie musi być to ten sam typ jak wysyłany do serwisu.

Accept: application/xml

Przedstawione powyżej przykłady należą do obszernej grupy[[6]](#footnote-6) nagłówków protokołu HTTP. Warte odnotowania jest to, iż oprócz standardowych nagłówków, programista może dodać własne w formacie KLUCZ: WAROŚĆ.

Ciałem zapytania HTTP są wszystkie informacje znajdujące się po pierwszej pustej linii.

Metody zapytań

Podstawowymi źródłami informacji o celu zapytania, jakie zostają przesłane miedzy klientem a serwerem, są metody [7]. Programiści stron internetowych, podczas przesyłania formularzy, najczęściej wykorzystują dwie z nich: POST oraz GET. Poza dwiema przedstawionymi, istnieje więcej metod, które można użyć podczas pisania kodu. Poniżej przedstawiono ich zestawienie wraz z opisem wykorzystania.

|  |  |
| --- | --- |
| Nazwa metody | Opis |
| GET | Metoda służy do pobrania zasobu z serwera. |
| HEAD | Pozwala otrzymać informacje o zasobie. Zastosowaniem tej metody jest sprawdzenie dostępu do danych. |
| PUT | Polega na wysłaniu informacji do aktualizacji zasobów serwera. Należy również sprecyzować, który zasób ma zostać zmieniony. |
| POST | Metoda ta przesyła informacje na serwer w celu utworzenia nowego zasobu (na przykład wpisu w bazie danych). |
| DELETE | Przesłanie zapytania DELETE ma na celu usunięcie konkretnego zasobu z systemu. |
| OPTIONS | Pozwala pobrać informacje o możliwościach, opcjach oraz wymaganiach, które istnieją w kanale komunikacyjnym. |
| TRACE | Metoda ta służy do diagnostyki kanału komunikacyjnego |
| CONNECT | Przekazywane przez nią są żądania dla serwerów pośredniczących, które są użyte do tunelowania |
| PATCH | Za pośrednictwem tej metody można aktualizować porcje zasobów. |

Statusy odpowiedzi protokołu HTTP

Każda odpowiedź serwera, podczas komunikacji z użyciem protokołu HTTP, posiada kod statusu [7]. Dostarczają one informacji o stanie połączenia pomiędzy serwerem a klientem, poprawności wykonywanych operacji oraz błędach. Kody składają się z trzech cyfr. Pierwszy znak, znajdujący się w miejscu setek, mówi o przynależności do typu zwracanej informacji. Istnieje pięć kategorii i należą do nich:

1. 1 – kody informacyjne, dotyczą one głównie statusu środowiska.
2. 2 – kody poprawnego wykonania zapytania. Zwracane są, jeżeli operacje na serwerze zostały wykonane poprawnie.
3. 3 – kody przekierowań. Zwracane są, jeżeli operacja przenosi użytkownika do innego zasobu w serwisie.
4. 4 – błędy użytkownika, które są zwracane w razie nieprawidłowego użytkowania zasobów systemu.
5. 5 – błędy serwera. Jest to grupa informująca o wszelkich zdarzeniach, które nie zostały przewidziane przez programistę.

Wartości na miejscach dziesiętnych i jedności odnoszą się do konkretnych komunikatów. Przykładowo kod *404* informuje o braku zasobu, *503* przekazuje informację o przekroczeniu czasu oczekiwania na odpowiedź serwera, natomiast *200* o poprawnym wykonaniu zapytania. Pełna lista kodów dostępna jest w dokumentacji technicznej protokołu HTTP [7].

1. .NET Framework

Obraz zawierający zwierzę

Opis wygenerowany przy wysokim poziomie pewności

Rysunek 3 Logo .NET Framework 4.5 źródło: https://www.microsoft.com/net

Historia

W roku 2002 firma *Microsoft* wydała pierwszą wersję swojej platformy programistycznej *.NET Framework*[[7]](#footnote-7). Wersja 1.0 przeznaczona była do tworzenia aplikacji dla  systemów operacyjnych z rodziny Windows (98, ME, NT 4.0, 2000 oraz XP). Platforma w wersji 1.0 otrzymała trzy pakiety aktualizacji (*Service Pack*), a wsparcie techniczne zakończyło się w 2007 roku. 3 kwietnia 2003 roku zostało wydane rozszerzenie .NET 1.1. Platforma została powiększona miedzy innymi o wsparcie dla tworzenia dynamicznych stron internetowych (*ASP.NET*), *.NET Compact Framework*[[8]](#footnote-8) oraz protokół *IPv6*.

Wersja 2.0 została przekazana programistom 22 stycznia 2006 roku i niosła za sobą szereg nowości. .NET 2.0 pozwalał na tworzenie oprogramowania przeznaczonego dla systemów operacyjnych z 64-bitową architekturą. Dodatkowo platforma została rozszerzona o nowe klasy i struktury pozwalające implementować typy *nullable*[[9]](#footnote-9). Programy pisane w wersji 2.0 mogły korzystać z wielu wątków procesora.

21 listopada 2006 roku na rynek został wypuszczony *.NET Framework 3.0*. Wprowadzał on nowe technologie tworzenia oprogramowania, jednak maszyna wirtualna uruchamiająca oprogramowanie napisane w .NET (*CLR*) pozostała taka sama jak w platformie w wersji 2.0. Do najważniejszych zmian, jakie wprowadził .NET 3.0 było dodanie *Windows Communication Foundation* (*WCF*), *Windows Presentation Foundation* (*WPF*) oraz aktualizacja *Compact Framework*. W listopadzie 2007 roku została opublikowana ulepszona wersja 3.5. Nowością było *LINQ* – komponent odpowiedzialny za przetwarzanie kolekcji obiektów, aby wydobyć z nich dane interesujące programistę. Dodatkowo wersja 3.5 powiększała możliwości implementacji programów przeznaczonych na urządzenia z systemem *Windows Embedded CE* oraz *Windows Mobile*.

.*NET Framework 4.0* został udostępniony 12 kwietnia 2010 roku. Rozwijał on możliwości systemów implementujących obliczenia równolegle[[10]](#footnote-10), które dzięki coraz szerszemu dostępowi do procesorów wielordzeniowy i wielowątkowych, stały się powszechne. .NET 4.0 wprowadzał również nową wersję maszyny wirtualnej *CLR*,która wspierała nowe wersje języków *C#* oraz *Visual Basic .NET.*

Wersja 4.5 dodała wsparcie dla *Universal Windows Platform* (*UWP*), które było nowym podejściem firmy *Microsoft* do budowania interfejsów graficznych. Platforma w tej wersji stałą się dostępna dla systemów Windows Vista i nowszych. Dzięki zastosowaniu *UWP*, możliwe stało się pisanie aplikacji na smartfony z systemem Windows 8. W kolejnych latach rozwoju *.NET Framework*, powstawały rozszerzenia komponentów odpowiedzialnych za łączność aplikacji oraz szyfrowanie. W wersji 4.6 zakończono wsparcie dla *Windowsa Vista*.

5 kwietnia 2017, wraz z wyjściem łatki *„Creators Update”* dla *Windows 10*, miała miejsce aktualizacja platformy .NET do wersji 4.7. Usprawniała ona istniejące komponenty oraz dodawała bibliotekę *.NET Standard* odpowiedzialną za tworzenie kodu współdzielonego przez projekty programów na wiele typów urządzeń.

Aktualnie trwają prace nad rozwojem *.NET Framework 4.8*, która ma być wydane w 2019 roku.

Cechy platformy .NET

Podczas tworzenia oprogramowania, przeznaczonego do uruchomienia w *.NET Framework*, kod źródłowy[[11]](#footnote-11) zamieniany jest na kod pośredni *CIL*. Pozwala to na łączenie elementów programu, które zostały napisane w różnych językach (na przykład kod C#, Python oraz C++) [8]. Na chwilę obecną dostępne jest ponad 40 języków, które można uruchomić za pomocą platformy .NET. Kompilacja metod, czyli zamiana funkcji z kodu pośredniego na kod maszynowy, ma miejsce w momencie pierwszego uruchomienia. Kolejne wywołania tego samego kodu wywołują już skompilowane informacje. Istnieje jednak możliwość pełnej kompilacji, w momencie instalacji oprogramowania.

Platforma składa się z trzech podstawowych bloków [9]:

1. *CLR – Common Language Runtime*   
   Jest to maszyna wirtualna, która odpowiada za kompilację oraz uruchamianie kodu *CIL*.
2. *CTS* *– Common Type System*Opisuje informacje, które zostały udostępnione przez środowisko.
3. *CLS – Common Language Specyfication*Zawiera zasady, które definiują zgodność kodu maszynowego ze standardami.

Warto zaznaczyć, iż *.NET Framework* został zaimplementowany nie tylko przez firmę *Microsoft*. Powstały projekty takie jak *MONO* (przez *Xamarin*) oraz *DotGNU*, które miały na celu utworzenie narzędzi spełniające standardy *ECMA* oraz dawać możliwość tworzenia oprogramowania w technologiach .NET na inne systemy operacyjne. *MONO* jest stale rozwijany jako projekt *opensource*[10]. Prace nad *DotGNU* zostały zawieszone w 2012 roku z powodu braku wolontariuszy do tworzenia platformy [11].

Technologie w .NET

Środowisko *.NET Framework* posiada wiele komponentów, które pomagają programistom budować aplikacje. Do najważniejszych technologii dostępnych w .NET zaliczają się zastępujące elementy składowe:

1. *Windows Forms*  
   Jest to pierwszy interfejs wspierający budowanie graficznych interfejsów użytkownika, który został przekazany w ręce programistów wraz z wprowadzeniem *.NET Framework 2.0.* Pozwala on na natywny[[12]](#footnote-12) dostęp do graficznego API systemów operacyjnych z rodziny Windows. Budowa GUI polega na przeciąganiu kontrolek z „przybornika” do ekranu *Designera* (rys. 3), wspierając metodę *WYSIWYG*[[13]](#footnote-13).

Obraz zawierający zrzut ekranu, monitor

Opis wygenerowany przy bardzo wysokim poziomie pewności

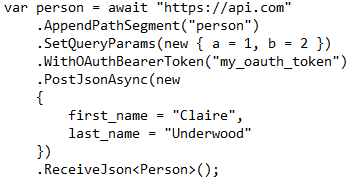
Rysunek Ekran Designera w Visual Studio 2017, źródło: opracowanie własne

1. *ASP.NET WebForms*  
   Jest to technologia analogiczna do *WinForms* z tą różnicą, że program, który może zostać utworzony z wykorzystaniem *WebForms*, jest dynamiczną stroną internetową. Aby uruchomić napisany program na serwerze, wymagana będzie instalacja *Internet Information Services* (*IIS*) lub wdrożenie w chmurze Azure[[14]](#footnote-14).
2. *Windows Presentation Foundation (WPF)*WPF jest kolejnym silnikiem graficznym, pozwalającym tworzyć graficzne interfejsy użytkownika. Moduł został dodany wraz z publikacją .NET 3.0. API odpowiedzialne za renderowanie kontrolek nazywa się *XAML* i jest to specjalnie przygotowana implementacja języka *XML*.Dzięki zastosowaniu *WPF*, możliwe jest budowanie bardziej zaawansowanych *GUI*, jednak wymaga to również większych umiejętności programistycznych.
3. *Windows Communication Foundation (WCF)  
   .NET Framework* umożliwia tworzenie usług sieciowych z wykorzystaniem między innymi *WCF.* Technologia ta została wdrożona do platformy .NET razem z *WPF* w wersji 3.0. Serwisy utworzone z zastosowaniem WCF, komunikują się z aplikacjami klienckimi za pomocą *endpointów[[15]](#footnote-15)*. Głównym protokołem komunikacyjnym, jaki wykorzystywany jest w *WCF*, jest *SOAP[[16]](#footnote-16)*. Od wersji .*NET Framework 4*. poszerzone zostało wsparcie dla budowania serwisów opartych na REST.
4. *XAMARIN*Ze względu na bardzo duży wpływ systemu operacyjnego *Android* [[17]](#footnote-17)na rynek urządzeń mobilnych, firma *Xamarin* (która zajmowała się tworzeniem platformy *MONO*) rozpoczęła prace nad stworzeniem środowiska umożliwiająca pisanie aplikacji dla *Androida* w środowisku .NET. Od 2011 roku prowadzone są prace nad rozwojem projektu, natomiast w 2016 roku *Xamarin* został wykupiony przez *Microsoft* i dodany do platformy .NET.   
   Pozwala on na tworzenie aplikacji przeznaczonych zarówno na *Androida*, jak i na *iOS* oraz *MacOS*[[18]](#footnote-18).
5. *.NET Standard*[12]  
   Wraz z umożliwieniem uruchomienia platformy .NET na systemach operacyjnych innych niż Windows, pojawiło się zapotrzebowanie na tworzenie części kodu, które były by współdzielone miedzy różnymi aplikacjami. W przypadku utworzenia aplikacji klienckiej na różne platformy (*Androida*, *Windowsa* lub *iOS*), wymagane było pisanie kodu wykonującego takie same czynności w każdym z projektów. Problem nadmiarowego kodu rozwiązuje *.NET Standard*, który jest „zbiorem interfejsów” zaimplementowanych we wszystkich instancjach .NET. Pozwala on na tworzenie bibliotek dostępnych na wszystkich platformach. Nowsze wersje *.NET Standard* posiadają więcej wbudowanych API, lecz nie wszystkie technologie i platformy posiadają implementację interfejsów. Przez rozpoczęciem prac programistycznych należy sprawdzić, czy środowisko, dla którego tworzone jest oprogramowanie wspiera wybraną wersję *.NET Standard.* Informacje o kompatybilnych komponentach, można znaleźć w dokumentacji [13] technologii.
6. *.NET Core*Ostatnim opisywanym elementem jest *.NET Core*, którego pierwsza wersja została udostępniona 27 czerwca 2016 roku [14]. Technologia ta jest implementacją *.NET Framework* dostępną na systemy operacyjne z rodzin *Windows*, *MacOS* oraz *Linux.* Dzięki jej zastosowaniu, możliwe jest tworzenie oraz uruchamianie aplikacji na niedostępnych wcześniej platformach. Obecnie obsługiwane technologie to *ASP.NET Core* oraz aplikacje konsolowe. Wersja 2.0 została wydana 14 sierpnia 2017 roku rozszerzając listę kompatybilnych API oraz dodając wsparcie dla przygotowanych specjalnie dla tej platformy *ASP.NET Core 2* oraz *Entity Framework Core 2*.   
   Aktualnie trwają prace nad .*NET Core 3* [15], które ma wprowadzić możliwość tworzenia aplikacji desktopowych (*WinForms* oraz *WPF*), rozwoju sztucznej inteligencji oraz „Internetu rzeczy”[[19]](#footnote-19).
7. Biblioteki implementujące REST

Celem pracy jest zbudowanie biblioteki wspierającej komunikację poprzez REST API. W Internecie można jednak znaleźć kilka gotowych sposobów, dzięki którym można zbudować połączenie z web serwisem. Poniżej przedstawiono kilka z nich, które są dostępne w *NuGet*.

NuGet jest to menedżer pakietów dla *.Net Framework.* Dostępny jest jako element środowiska programistycznego Visual Studio i pozwala na szybkie wyszukiwanie, pobieranie oraz instalacje pakietów dla projektu. W repozytorium NuGet można znaleźć paczki między innymi dla języków programowania C# oraz C++.

Flurl.Http  
Jest to biblioteka [16] pozwalająca na komunikację poprzez wysyłanie zapytań HTTP. Można ją wykorzystać w projektach komercyjnych ze względu na udostępnienie tej biblioteki, przez twórcę, na licencję MIT. Nazwa to skrót od *„Fluent URL”* i jest elementem projektu *Fluent URL Builder* służącego do prostego generowania adresów *URL*. Flurl pozwala na budowę zapytania w następujący sposób:



Rysunek Pobieranie danych z pomocą biblioteki Flurl.Http, źródło: https://flurl.io/

RestSharp

Kolejną biblioteką, jaką można wykorzystać do komunikacji z serwisem, jest RestSharp [17]. Jest to zaawansowany projekt, który zawiera mechanizmy pozwalające wysyłać zapytania oraz parsować dane do formatów XML i JSON.   
Dodatkową cechą tej biblioteki jest obsługa synchronicznych oraz asynchroniczncych zapytań. RestSharp pozwala na wykonanie metod: GET, POST, PUT, DELETE, HEAD oraz OPTIONS. Biblioteka rozpowrzechniana jest na licencji Apache-2.0. Według statystyk na GitHub, projekt tworzyło 173 programistów, a ostatnia aktualizacja miała miejsce 24 sierpnia 2018r. Poniżej przedstawione zostały przykładowe fragmenty kodu ze strony projektu, które przedstawiają metody biblioteki.

Obraz zawierający zrzut ekranu

Opis wygenerowany przy bardzo wysokim poziomie pewności

Rysunek Przykładowy fragment kodu RestSharp, źródło: http://restsharp.org/

1. Autorska implementacja biblioteki

Założenia projektowe

Przed rozpoczęciem prac programistycznych, należy zaplanować, co oraz w jakiej kolejności będzie projektowane i pisane. Aby można było wykonać system, komunikujący się z wykorzystaniem *REST API*, należy zaimplementować web serwis udostępniający takie *API*. System udostępniać będzie cztery metody HTTP: GET, POST, PUT oraz DELETE. Do komunikacji z serwisem wybrano aplikację, której *GUI* zostanie wykonane z wykorzystaniem *WinForms*. Aplikacja kliencka będzie komunikować się z serwisem za pomocą zapytań protokołu *HTTP*. Funkcjonalności te zostaną umieszczone w bibliotece przygotowanej w ramach niniejszej pracy. Informacje, które będą przetwarzane w trakcie pracy systemu, zostaną umieszczone w bazie danych *SQL Server Express*, a całość prac programistycznych zostanie wykonana w *Visual Studio 2017 Community*.

Obraz zawierający wewnątrz, tekst

Opis wygenerowany przy bardzo wysokim poziomie pewności

Rysunek Schemat przedstawiający strukturę projektu, źródło: opracowanie własne

System przygotowany w celach wizualizacji działania biblioteki, ma być prostym narzędziem do zarządzania pracownikami i projektami w firmie. Nie przewidziano zaawansowanych funkcjonalności, gdyż najważniejszym celem jest testowanie komunikacji między programami.

Baza danych

Informacje przetwarzane w systemie, powinny być przechowywane w odpowiednim miejscu, do czego służy baza danych. Dla projektu, wybrany został system dostarczony przez firmę *Microsoft* – *SQL Server 2017 Express*.  
Jest to darmowa wersja SQL Server posiadająca ograniczenia, które nie mają wpływu na działanie budowanego systemu (na przykład ograniczenie wielkości bazy danych do 10GB). Do administrowania bazą danych, użyty został SQL Server Management Studio – bezpłatnego oprogramowanie przygotowane przez twórcę bazy danych, służące do modyfikowania i zarządzania zasobami dostępnymi w systemie.

Obraz zawierający zrzut ekranu

Opis wygenerowany przy bardzo wysokim poziomie pewności

Rysunek Ekran SQL Server Management Studio po zalogowaniu do bazy danych,   
źródło: opracowanie własne

Baza przygotowana dla projektu składa się z czterech tabel, których relacje zostały przedstawione na Rysunku 9. Tabele *Projects*, *Users* oraz *Rooms* przechowują główne informacje w systemie. Zbiór nazwany *ProjectsUsers* przechowuje informacje o połączeniu między *Projects* i *Users* ze względu na relację wiele-do-wielu. *User* przechowuje *Id* pomieszczenia, w którym się znajduje, dzięki czemu można w łatwy sposób dowiedzieć się w jakim pomieszczeniu się znajduje.

Obraz zawierający zrzut ekranu

Opis wygenerowany przy bardzo wysokim poziomie pewności

Rysunek Diagram przedstawiający strukturę bazy danych, źródło: opracowanie własne

Visual Studio 2017

Wraz z rozwojem *.NET Framework*, *Microsoft* aktualizuje środowisko programistyczne stworzone do pracy z platformą. *Visual Studio* jest to zaawansowane *IDE* (ang. *Integrated Development Enviroment,* środowisko programistyczne przeznaczone do pracy z kodem aplikacji) rozwijane od 1997 roku.

Obraz zawierający zrzut ekranu, czarny, monitor, ściana

Opis wygenerowany przy bardzo wysokim poziomie pewności

Rysunek Ekran startowy Visual Studio 2017, źródło: opracowanie własne

Główną cechą *IDE* jest pełna integracja z *.NET Framework* oraz usługami firmy *Microsoft*. W przypadku tworzenia aplikacji sieciowej, możliwa jest automatyczna publikacja w usłudze *Azure*. W przypadku pracy z projektem udostępnionym w repozytorium systemu kontroli wersji Git (na przykład *GitHub*), *Visual Studio* automatycznie przetwarza informacje z repozytorium informując o zmianach i pochodzeniu elementów kodu. Środowisko programistyczne posiada również zaawansowany *debuger*, wsparcie dla wielu języków programowania, menedżer pakietów NuGet oraz zaawansowany system kontroli składowych, dzięki któremu można dostosować środowisko do własnych potrzeb (bez potrzeby pobierania wszystkich elementów IDE). *Visual Studio* posiada wiele gotowych szkieletów aplikacji, dzięki czemu rozpoczęcie budowy projektu jest szybsze i prostsze.

C ♯ (C Sharp)

Głównym językiem, jaki stosowany jest podczas pisania aplikacji na platformę .NET, jest C Sharp. Pierwsza wersja języka była opracowywana w latach 1998 - 2001 przez firmę *Microsoft*. Jest to obiektowy język wysokiego poziomu, którego składnia jest kompilowana do języka pośredniego *.NET Framework*[8].

Do podstawowych cech C♯ należą:

1. *Garbage Collector* – mechanizm zajmujący się czyszczeniem pamięci automatycznie, gdy informacje przestają być potrzebne. Programista nie musi zajmować się pisaniem kodu zwalniającego zasoby, co ma miejsce na przykład w języku C, C++.
2. Każda klasa napisana w języku C♯, dziedziczy po klasie Object.
3. Delegaty oraz Zdarzenia – rozwinięcie wskaźników do funkcji z języka C++.
4. Uruchamianie kodu w wirtualnej maszynie *.NET Framework*.
5. Dobre wsparcie języka do programowania asynchronicznego.
6. Wyrażenia lambda oraz LINQ

C♯ może być zastosowany do pisania programów na większość dostępnych urządzeń. Dzięki technologiom zawartym w .NET, możliwe jest tworzenie zaawansowanych stron internetowych, web serwisów, aplikacji desktopowych i mobilnych, a także gier wideo z użyciem silnika Unity3D[[20]](#footnote-20).

Język ten jest stale rozwijany, a jego najnowsza wersja (C♯ 7.3) została udostępniona programistom 7 maja 2018 roku.

Web Serwis

Solucja, która implementuje serwis udostępniający REST API składa się z dwóch projektów: *ProjectClasses oraz WCF\_*REST, a ich struktura przedstawiona jest na Rysunku 11. Pierwszy z nich (*ProjectClasses*) jest biblioteką klas, dołączoną także do aplikacji klienckiej. Przechowuje ona wszystkie klasy, na których operować będzie użytkownik w trakcie działania aplikacji. W bibliotece znajduje się także plik Configuration.cs, przechowujący adresy do połączeń.

Obraz zawierający monitor, zrzut ekranu, ekran, sprzęt elektroniczny

Opis wygenerowany przy wysokim poziomie pewności

Rysunek Struktura Serwisu, źródło: opracowanie własne

Projekt *WCF\_REST* jest startowym projektem WCF, który implementuje web serwis oparty o REST. W katalogu Controlleres znajduje się klasa Logic.cs przechowująca wszystkie procedury wykorzystane w systemie. Rysunek 12 przedstawia kod źródłowy funkcji GetUser(), zwracającej wszystkie osoby zapisane w tabeli Users.

Obraz zawierający zrzut ekranu

Opis wygenerowany przy bardzo wysokim poziomie pewności

Rysunek Kod źródłowy funkcji GetUser(), źródło: opracowanie własne

Ważnym elementem tego projektu jest interfejs IRestService.cs. Przechowuje on funkcje, które mogą być wywołane w API. Plik został podzielony na cztery regiony, zależnie od typu metod HTTP.

Obraz zawierający wewnątrz

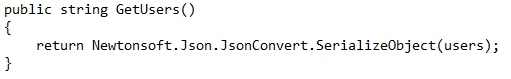
Opis wygenerowany przy wysokim poziomie pewności

Rysunek Fragment metody GetUser() w interfejsie serwisu, źródło: opracowanie własne

Na rysunku 13 przedstawiony został fragment kodu opisującego zapytanie pobierające użytkowników. Atrybut OperationContract informuje kompilator o tym, że funkcja, do której został dopisany, jest elementem kontraktu usługi *WCF*.   
WebInvoke z kolei oznacza, że dana operacja może zostać wywołana w modelu REST. Poszczególne składowe oznaczają:

1. Method – Metoda HTTP, która obsługuje daną funkcję. Dostępne warianty zostały opisane w rozdziale „***Metody zapytań***”.
2. ResponseFormat – Format wiadomości w jakim serwis odpowiada klientowi. Wariantami typu są XML oraz JSON
3. ReqestFormat – Format wiadomości, jaki jest oczekiwany przez serwis. Możliwe są, tak samo jak w przypadku odpowiedzi, XML oraz JSON
4. BodyStyle – Informuje serwis o stylu ciała informacji wymienianych przez serwer i klienta. Możliwe jest wybranie „nagiej informacji”, opakowanej (dla zapytania i odpowiedzi), opakowanego tylko zapytania lub opakowanej tylko odpowiedzi.
5. UriTemplate – służy do definiowania ścieżki dostępu do zasobu jaki ma być przetwarzany poprzez zapytanie.

Metody interfejsu IRestService.cs implementowane są w pliku RestService.svc. Przechowuje on w sobie plik klasy oraz jest głównym elementem API, do którego odwołują się aplikacje klienckie podczas wywoływania zapytań HTTP. Na rysunku 14 przedstawiony został kod odpowiedzialny za zwracanie informacji w formacie JSON.



Rysunek Funkcja serwisu REST zwracająca użytkowników systemu

Informacje przed przekazaniem przez REST API zostają poddane serializacji za pomocą biblioteki Json.NET [18].

Aplikacja kliencka

Do pobierania oraz przetwarzania danych z serwisu, utworzona została aplikacja desktopowa wykorzystująca technologię WinForms. Składa się ona z czterech elementów (Rysunek 15), z czego XPTable jest projektem wykonanym przez Mathew Hall’a implementującym wygodniejszy w użyciu GridView [19]. Pozostałe trzy składowe zostały wykonane na potrzeby publikacji.

Obraz zawierający czarny, siedzi, stół

Opis wygenerowany przy wysokim poziomie pewności

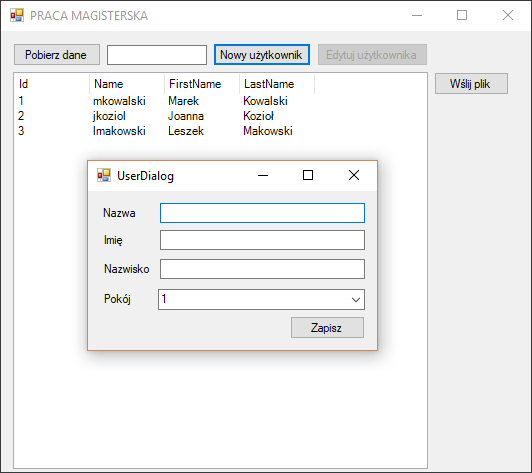
Rysunek Struktura aplikacji klienckiej, źródło: opracowanie własne

Projekt ProjectClasses, został opisany w poprzednim rozdziale – jest to biblioteka przechowująca klasy wykorzystane w systemie. RESTLib jest to część odpowiadająca za komunikację z serwisem i został jej poświęcony kolejny rozdział.

Aby można było komfortowo korzystać z systemu, potrzebny jest interfejs graficzny. Za prezentację informacji na ekranie odpowiada projekt ClientApp.

Aplikacja składa się z dwóch okien:

1. głównego, odpowiadającego za wyświetlanie informacji o użytkownikach
2. okna edycji/dodawania nowego rekordu.



Rysunek Uruchomiona aplikacja kliencka, źródło: opracowanie własne

Po uruchomieniu, aplikacja pobiera automatycznie wszystkie rekordy z tabeli Users. Aby zawęzić ilość odbieranych informacji do pojedynczego wpisu, należy podać Id pracownika w polu tekstowym znajdującym się na górze ekranu, a następnie wcisnąć „Pobierz dane”. Przycisk „Nowy użytkownik” otwiera okno dialogowe „UserDialog”, w którym możliwe utworzenie nowego pracownika. Ten sam ekran pojawia się podczas wybrania rekordu w *GridView* głównego widoku i wciśnięciu „Edytuj użytkownika”. „Zapisz” w oknie spowoduje wysłanie informacji do serwisu oraz wprowadzenie zmian w bazie danych. Zależnie od tego, czy edytowany jest rekord, lub czy zachodzi proces dodawania nowego, wywoływana jest odpowiednia metoda REST. Ostatnim elementem aplikacji klienckiej jest przycisk „Wyślij plik”. Wykorzystanie go powoduje otwarcie okna dialogowego, w którym możliwe jest podanie dowolnego źródła z dysku komputera, a następnie przesłanie go na serwer.

Biblioteka RESTLib

Jest to najważniejszy element niniejszej pracy, który implementuje bibliotekę do połączeń klient-serwer poprzez REST API. Projekt wykonany został w technologii .NET Framework 4.7. Biblioteka składa się z:

**Receiver.cs**Jest to klasa, której obiekt służy do odbierania danych metodą HTTP GET. Posiada w sobie zaimplementowaną generyczną funkcję, do której przekazywane są nazwa metody, parametry oraz flaga isArray. Podany ciąg znaków method jest składową, która dodawana jest jako pierwszy element do adresu API i odnosi się do konkretnej funkcji serwisu. Tablica parameters przechowuje w sobie zmienne, które należy podać dodatkowo (na przykład Id rekordu z bazy danych). Flaga isArray informuje program, czy pobrany ma zostać pojedynczy element, lub czy oczekiwana jest kolekcja.

**Sender.cs**Metoda HTTP POST została zaimplementowana w klasie Sender. Wewnątrz znajdują się dwie funkcje: SendData oraz SendFileAsStream. Pierwsza z nich odpowiada za transfer obiektów przy pomocy REST API. Podczas wywołania trzeba przekazać nazwę metody API oraz informacje, które mają zostać przesłane. Za pomocą SendFileAsStream możliwy jest transfer dużych plików, niezależnie od ograniczeń środowiska w którym uruchomiony został serwer. Wskazany obiekt zostaje najpierw rozłożony na tablicę bajtów, a następnie przesłany w porcjach po 1000 bajtów. Ostatnia ramka przekazuje informację o zakończeniu transferu.

**Updater.cs**Aktualizacja istniejącego rekordu możliwa jest poprzez wywołanie metody PUT. Klasa Updater, podobnie jak Sender, posiada funkcję generyczną, do której przekazywane są dwa parametry: method oraz dataToSend. Dla wskazanego rekordu zostają ustalone nowe informacje. Serwis wie, który wpis ma zostać zmieniony pobierając Id z przesyłanej klasy.

**Deleter.cs**Ostatnia klasa, która znajduje się w przygotowanej bibliotece służy do usuwania danych. Zaimplementowana metoda DeleteData, w pierwszym parametrze oczekuje podania nazwy funkcji API, natomiast w kolejnym listy Id do usunięcia w formacie tablicy ciągów znaków (string []).

Funkcje klas przedstawionych powyżej, zwracają zmienną bool o wartości zależnej od poprawności wykonanych operacji. Wyjątkiem jest klasa Receiver, która zwraca obiekty pobranych informacji.

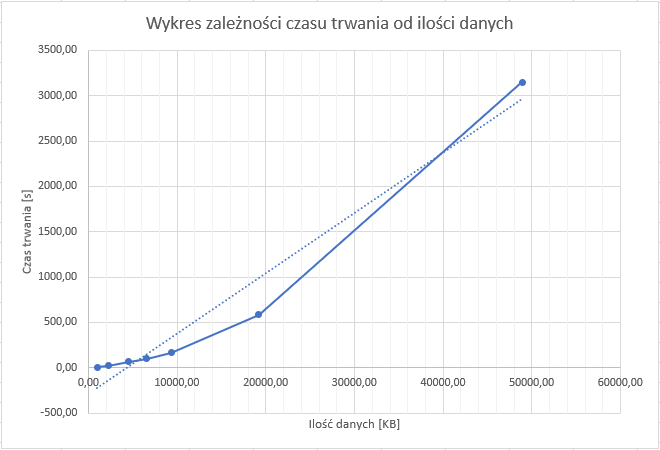
Testy efektywności systemu

Poniżej przedstawiona została tabela obrazująca czasy wykonywania operacji, z wykorzystaniem przygotowanej biblioteki oraz systemu. Czas trwania operacji został zmierzony za pomocą dostępnej w języku C# klasy Stopwatch, usługa sieciowa oraz aplikacja desktopowa uruchomione zostały na tej samej maszynie.

Tabela Wyniki pomiarów

|  |  |  |
| --- | --- | --- |
| METODA | Struktura danych | Czas trwania operacji |
| GET | User [] | 0,5187978 sec |
| GET | User | 0,0034832 sec |
| POST | User | 0,0050815 sec |
| PUT | User | 0,0068145 sec |
| DELETE | User | 0,0175763 sec |
| POST | 972KB | 8,009 sec |
| POST | 2203KB | 20,320 sec |
| POST | 4526KB | 63,970 sec |
| POST | 6548KB | 99,073 sec |
| POST | 9308KB | 166,208 sec |
| POST | 19187KB | 583,874 sec |
| POST | 48917KB | 3148,882 sec |

Rysunek 17 przedstawia wyniki testów przeprowadzonych podczas przesyłania plików. Transferowane były paczki informacji w wielkościach 972KB, 2203KB, 4526KB, 6548KB, 9308KB, 19187KB oraz 48917KB.



Rysunek Wykres zależności czasu trwania operacji od ilości przesyłanych danych, źródło: opracowanie własne

Z wykresu wynika, iż sama aplikacja poprzez sposób implementacji funkcjonalności, ma wpływ na czas przesyłania informacji – wykres ma charakter funkcji kwadratowej. Dla plików do 10MB, zauważalna jest zależność liniowa. Przy przesyłaniu większych ilości informacji, widoczny jest wzrost czasu wymagany do przesłania jednego kilobajta.

1. Podsumowanie

Niniejsza publikacja stworzona została, aby zaprezentować jak w łatwy sposób utworzyć działający system oparty na architekturze klient – serwer oraz aby zbudować bibliotekę pozwalającą pobierać dane z REST API.   
Znajomość opisanych w pierwszym i drugim rozdziale technologii jest kluczowa, jeśli programista chciałby napisać własną wersję przygotowanego systemu. Zawarte w tych częściach publikacji informacje, na pewno przybliżyły czytelnikowi to, w jaki sposób działa protokół HTTP. Każdy ma z nim styczność (lub jego szyfrowaną odmianą HTTPS) codziennie, podczas połączeń ze stronami internetowymi. Treść zawarta w tych rozdziałach mogła być również pouczająca ze względu na zgłębienie tematu głównych technologii firmy *Microsoft*, które służą do tworzenia oprogramowania.

Przedstawione w rozdziale trzecim biblioteki programistyczne mogą znacząco przyspieszyć prace osób, które chcą się skupić na tworzeniu aplikacji klienckich – bez zagłębiania się w zasady działania usług sieciowych.

Część czwarta pracy pokazała w jaki sposób oraz z użyciem jakich technologii zbudować działający szkielet systemu w architekturze klient – serwer. Przedstawione dane, obrazujące wydajność implementacji autora, mogą nakierować na wrażliwe fragmenty kodu.

Przeprowadzone testy wykazały wpływ implementacji na działanie systemu. Warto pochylić się nad elementami które wykazują wolne działanie i spróbować zoptymalizować te elementy.

Dostrzegalna jest również możliwość rozwinięcia projektu (oprócz optymalizacji) – na przykład dodanie implementacji dla pozostałych zapytań HTTP lub wsparcie dla API, które wymagają uwierzytelnienia.

Kod źródłowy systemu wykonanego jako część składową pracy, dostępny jest w serwisie GitHub pod adresem https://github.com/Czektul/MasterThesis

Bibliografia

1. W3C, „Web Services Glossary”, https://www.w3.org/TR/ws-gloss/#webservice, dostęp: 06.09.2018r.
2. R. Fielding „*Principled Design of the Modern Web Architecture*”, https://www.ics.uci.edu/~taylor/documents/2002-REST-TOIT.pdf,   
   dostęp 01.09.2018r.
3. S. Czech *„Cechy dobrego RESTful API”*, wpis na blogu z 20.02.2017 http://sebastianczech.com/2017/02/20/cechy-dobrego-restful-api/,   
   dostęp 02.09.2018r.
4. https://developer.mozilla.org/en-US/docs/Web/HTTP/, dostęp 02.09.2018r.
5. K. Bielski „*Implementacja REST w Spring MVC 3.0*” wpis na blogu z 26.08.2012 https://kbielski.wordpress.com/tag/rest-web-service/ ,   
   dostęp 02.09.2018r.
6. T. Berners-Lee, L. Masinter, M. McCahill „*Uniform Resource Locators (URL)”, Grudzień 1995,* https://tools.ietf.org/html/rfc1738,   
   dostęp 04.09.2018r.
7. R. Fielding, J. Reschke *„Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content”*, ISSN: 2070-1721, Czerwiec 2014, https://tools.ietf.org/html/rfc7231, dostęp: 04.09.2018r.
8. J. Albahari, B. Albahari, „*C# 7.0 w pigułce. Wydanie VII*”, 2018, Wydawnictwo Helion, ISBN: 978-83-283-4075-6
9. „*C# Language Specification*”, https://www.ecma-international.org/ publications/files/ECMA-ST/Ecma-334.pdf, ECMA-334, Grudzień 2017,   
   dostęp: 04.09.2018r.
10. XAMARIN „Mono”, Repozytorium w serwisie Github, https://github.com/mono/mono, dostęp: 04.09.2018r.
11. „DotGNU”, Strona projektu http://www.dotgnu.org/, dostęp: 04.09.2018r.
12. „.NET Standard”, dokumentacja techniczna https://docs.microsoft.com/pl-pl/dotnet/standard/net-standard, dostęp: 04.09.2018r.
13. J. Gutkowski „Moje podejście do .NET Standard”,   
    wpis na blogu z 13.03.2017r. https://blog.gutek.pl/2017/03/13/moje-podejscie-net-standard/, dostęp: 06.09.2018r.
14. A. Freeman „ASP.NET Core MVC 2”, 2017, Wydawnictwo Helion, ISBN: 978-83-283-4601-7
15. M. J. Foley „Microsoft showcases SQL Server, .NET Core on Red Hat Enterprise Linux deliverables”, artykuł ZDNet, 27 czerwca 2016,  
    https://www.zdnet.com/article/microsoft-showcases-sql-server-net-  
    core-on-red-hat-enterprise-linux-deliverables/  
    dostęp: 06.09.2018r.
16. T. Menier, „Flurl”, https://flurl.io/, strona projektu, dostęp: 06.09.2018r.
17. „RestSharp”, http://restsharp.org/, strona projektu, dostęp: 06.09.2018r.
18. Newtonsoft „Json.NET”, http://json.net, strona projektu,   
    dostęp: 06.09.2018r.
19. Mathew Hall, „XPTables”, https://www.codeproject.com/Articles/11596/XPTable-NET-ListView-meets-Java-s-JTable, 17 września 2005, strona projektu,   
    dostęp 06.09.2018r.

Spis rysunków

Rysunek 1 Architektura klient – serwer, źródło: opracowanie własne 4

Rysunek 2 Struktura adresu URL, źródło: opracowanie własne 7

Rysunek 3 Logo .NET Framework 4.5 źródło: https://www.microsoft.com/net 11

Rysunek 4 Ekran Designera w Visual Studio 2017, źródło: opracowanie własne 14

Rysunek 5 Pobieranie danych z pomocą biblioteki Flurl.Http, źródło: https://flurl.io/ 18

Rysunek 6 Przykładowy fragment kodu RestSharp, źródło: http://restsharp.org/ 19

Rysunek 7 Schemat przedstawiający strukturę projektu, źródło: opracowanie własne 20

Rysunek 8 Ekran SQL Server Management Studio po zalogowaniu do bazy danych, źródło: opracowanie własne 21

Rysunek 9 Diagram przedstawiający strukturę bazy danych, źródło: opracowanie własne 21

Rysunek 10 Ekran startowy Visual Studio 2017, źródło: opracowanie własne 22

Rysunek 11 Struktura Serwisu, źródło: opracowanie własne 24

Rysunek 12 Kod źródłowy funkcji GetUser(), źródło: opracowanie własne 25

Rysunek 13 Fragment metody GetUser() w interfejsie serwisu, źródło: opracowanie własne 25

Rysunek 14 Funkcja serwisu REST zwracająca użytkowników systemu 26

Rysunek 15 Struktura aplikacji klienckiej, źródło: opracowanie własne 27

Rysunek 16 Uruchomiona aplikacja kliencka, źródło: opracowanie własne 28

Rysunek 17 Wykres zależności czasu trwania operacji od ilości przesyłanych danych, źródło: opracowanie własne 31

1. HTTP - protokół przesyłania plików hipertekstowych w sieci WWW [↑](#footnote-ref-1)
2. Routing – wyznaczanie trasy i przesyłanie informacji w sieci kilku komputerów. [↑](#footnote-ref-2)
3. JavaScript – skryptowy język programowania, wykorzystywany głównie na stronach internetowych. [↑](#footnote-ref-3)
4. C POSIX Library - https://pl.wikipedia.org/wiki/C\_POSIX\_library [↑](#footnote-ref-4)
5. URL (ang. Uniform Resource Locator) – ujednolicony format dostępu do informacji w sieci. [↑](#footnote-ref-5)
6. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers [↑](#footnote-ref-6)
7. Framework – Platforma programistyczna. Dzięki zastosowaniu narzędzi tego typu, programista może w szybszy sposób utworzyć aplikację. Frameworki zwykle dostarczają gotowe narzędzia oraz biblioteki programistyczne ułatwiające pracę. [↑](#footnote-ref-7)
8. .NET Compact – wersja .Net Framework przeznaczona na urządzenia mobilne. [↑](#footnote-ref-8)
9. Typ nullable – typ danych pozwalający programiście przypisać NULL jako wartość zmiennej. NULL jest to specjalna wartość informująca program o braku odniesienia do jakiejkolwiek wartości. [↑](#footnote-ref-9)
10. Obliczenia równoległe – obliczenia, które wykonywane są w tym samym czasie. Programy implementujące taką formę obliczeń wymagają użycia wielu procesorów lub rdzeni w CPU. [↑](#footnote-ref-10)
11. Kod źródłowy – treść programu napisana przez programistę. [↑](#footnote-ref-11)
12. Natywny – „właściwy danemu środowiskowi informatycznymu”. [↑](#footnote-ref-12)
13. WYSIWYG (ang. what you see is what you get) – akronim stosowany przez programistów dla opisania metod budowania aplikacji, gdzie otrzymana aplikacja wygląda identycznie jak na ekranie edytora. [↑](#footnote-ref-13)
14. Microsoft Azure – platforma firmy Microsoft, pozwalająca uruchamiać oprogramowanie w chmurze obliczeniowej. [↑](#footnote-ref-14)
15. Endpoint – typ węzła komunikacji, może to być na przykład adres URL do WebAPI. [↑](#footnote-ref-15)
16. SOAP – Protokół transmisji danych, przesyłający dokumenty XML głównie z użyciem HTTP. Istnieje możliwość wykorzystania innych protokołów komunikacyjnych. [↑](#footnote-ref-16)
17. Android – system operacyjny przeznaczony dla urządzeń mobilnych (na przykład smartfonów) oparty na jądrze Linuxa. [↑](#footnote-ref-17)
18. iOS, MacOS – systemy operacyjne przeznaczony na urządzenia firmy Apple [↑](#footnote-ref-18)
19. Internet rzeczy – znane również jako *IoT* lub *Internet of Things*, jest to koncepcja mówiąca o komunikacji między przedmiotami za pośrednictwem różnych kanałów komunikacyjnych. [↑](#footnote-ref-19)
20. Unity - środowisko pozwalające tworzyć gry komputerowe. Posiada szereg gotowych funkcjonalności, takich jak silnik fizyczny czy generator plików binarnych na wiele urządzeń. [↑](#footnote-ref-20)