
	Uniwersytet Technologiczno-Przyrodniczy im. J. i J. Śniadeckich w Bydgoszczy Wydział Telekomunikacji, Informatyki i Elektrotechniki <b>Zakład Techniki Cyfrowej</b>		
<b>Przedmiot</b>	Programowanie w środowisku Windows		
<b>Prowadzący</b>	dr inż. M. Sulima		
<b>Temat</b>	Projekt – Uzupełnianie geo lokalizacji na podstawie Tyret		
<b>Student</b>	Mateusz Kalksztejn , Filip Nowicki		
<b>Data oddania spr.</b>	25.01.2021	<b>Data wykonania</b>	25.01.2021
<b>Ocena</b>		-	-

## Wstęp

Celem projektu było utworzenie aplikacji webowej która przy pomocy użytkownika uzupełnia informację o szerokości i długości geograficznej dla danej ulicy. Projekt był realizowany przy pomocy **.net framework** , **.net Core** oraz **Blazor**.

## Implementacja

Łączność z **Api Tyret** pozyskiwana jest przy pomocy **Api** konsumującego w **.net framework**.

Implementacja kontrolera wygląda następująco:

```

public class HelloController : ApiController
{
    public string Get()
    {
        var random = new Random();

        var proxy = new ChannelFactory<ITerytWs1>("custom");
        proxy.Credentials.UserName.UserName = "Arventill";
        proxy.Credentials.UserName.Password = "MvM2E19N0";
        var result = proxy.CreateChannel();

        int rInt2 = random.Next(0, CityList.Count);
        var randomCity = CityList[rInt2];

        var miejscowosci = result.PobierzListeUlicDlaMiejscowosci(randomCity.Woj, randomCity.Pow,
        randomCity.Gmi, randomCity.Rd, randomCity.Msc, randomCity.U, randomCity.A, DateTime.Now);
        int liczbaMiejscowosci = miejscowosci.Count();
        int rInt = random.Next(0, liczbaMiejscowosci);

        return randomCity.MscFull + ", " + miejscowosci[rInt].Nazwa2 + " " + miejscowosci[rInt].Nazwa1;
    }

    public bool Post(DataRequest request)
    {
        if (string.IsNullOrEmpty(request.City) || string.IsNullOrEmpty(request.StreetName)
        || request.StreetCordX == 0 || request.StreetCordY == 0
        || request.StreetCordX == null || request.StreetCordY ==
        null)
            return false;

        return true;
    }
}

```

Figure 1 AnyApi HelloController.cs

Kolejnym krokiem było utworzenie kolejnego **Api** które zapisuje w bazie danych uzupełnione dane oraz odpowiada za przesyłanie i odbieranie informacji z warstwy wizualnej. Ta warstwa napisana jest w **.net Core**.

```
[Route("api/[action]")]
public class HomeController : Controller
{
    private readonly ILogger _logger;
    private readonly IProjectRepository _projectRepository;

    public HomeController(
        ILogger<HomeController> logger,
        IProjectRepository projectRepository)
    {
        _logger = logger;
        _projectRepository = projectRepository;
    }

    [HttpGet]
    public string GetStreet()
    {
        WebRequest request = WebRequest.Create("http://localhost:59294/api/hello");
        // If required by the server, set the credentials.
        request.Credentials = CredentialCache.DefaultCredentials;
        // Get the response.
        HttpWebResponse response = (HttpWebResponse)request.GetResponse();
        // Display the status.
        Console.WriteLine(response.StatusDescription);
        // Get the stream containing content returned by the server.
        Stream dataStream = response.GetResponseStream();
        // Open the stream using a StreamReader for easy access.
        StreamReader reader = new StreamReader(dataStream);
        // Read the content.
        string responseFromServer = reader.ReadToEnd();

        return responseFromServer;
    }

    [HttpPost]
    public bool SaveToDatabase([FromBody]DataRequest request)
    {
        if (string.IsNullOrEmpty(request.City)
            || string.IsNullOrEmpty(request.StreetName)
            || string.IsNullOrEmpty(request.StreetCordX)
            || string.IsNullOrEmpty(request.StreetCordY))
            return false;

        return _projectRepository.SaveToDatabase(request) == 1;
    }
}
```

Figure 2 WindowsyProjekt HomeController.cs

Metoda **GetStreet** zwraca łańcuch znaków w formie: Miasto, Ulica

Metoda **SaveToDatabase** przyjmuje obiekt typu **DataRequest** wyglądający następująco :

```
public class DataRequest
{
    public string City { get; set; }

    public string StreetName { get; set; }

    public string StreetCordX { get; set; }

    public string StreetCordY { get; set; }

    public string Additional { get; set; }
}
```

## Front oraz logika aplikacji

Po uzyskaniu informacji o nazwie miasta oraz ulicy front przy użyciu „geokodera” od API **OpenStreetMap** pobiera „geolokalizację” danej ulicy. Następnie jest wyświetlana na mapie.

Użytkownik ma możliwość edycji lokalizacji oraz wyłania uzupełnionego rekordu do bazy danych.

**OpenStreetMap** jest to narzędzie udostępniające opcję związane z mapami globalnymi.

### Realizacji

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8"/>
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>BlazorOSM</title>
  <base href="/" />
  <link rel="stylesheet" href="css/bootstrap/bootstrap.min.css"/>
  <link href="css/site.css" rel="stylesheet"/>
  <link href="BlazorOSM.styles.css" rel="stylesheet"/>
  <!-- Load Leaflet from CDN -->
  <link rel="stylesheet" href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css"
    integrity="sha512-xodZBNTCSn17Xt2atTPuE1HxjVMSvLVW9ocqUKLsCC5CXdbqCmblAshOMAS6/keqq/sMZM19scR4PsZChSR7A=="
    crossorigin="" />
  <script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js"
    integrity="sha512-XQoYMqMTK8LvdxXYG3nZ448hOEQiglfqkJs1NOQV44CwNurBc8PKAOCxY20w0v1aXaVUearIOBhiXZ5V3ynxwA=="
    crossorigin=""></script>

  <!-- Load Esri Leaflet from CDN -->
  <script src="https://unpkg.com/esri-leaflet@2.5.0/dist/esri-leaflet.js"
    integrity="sha512-ucw7Grpc+IEQZa711gcjgMBnmd9qju1CICsRaryvX7HJklKpG1/prxKvtHwpgm5ZHdvAil7YPxI1oWP0WK3UQ=="
    crossorigin=""></script>

  <!-- Load Esri Leaflet Geocoder from CDN -->
  <link rel="stylesheet" href="https://unpkg.com/esri-leaflet-geocoder@2.3.3/dist/esri-leaflet-geocoder.css"
    integrity="sha512-IM3Hs+feyi40yZdH6kV8vQMg4Fh20s90zInIIAc4nx7aMYMfo+IenRUekoYsHZqGkREUGx0Vv1Esgm7nCDW9g=="
    crossorigin="" />
  <script src="https://unpkg.com/esri-leaflet-geocoder@2.3.3/dist/esri-leaflet-geocoder.js"
    integrity="sha512-HrFUYCEtIpxZloTgEKKMq4RFYhxjJkCiF5sDxuAokk10eZ68U2NPfh4MFtyIVW1sKtVbK56D2/JzFyAfvT5ejA=="
    crossorigin=""></script>
</head>
<body>
<component type="typeof(App)" render-mode="ServerPrerendered"/>

<div id="blazor-error-ui">
  <environment include="Staging,Production">
    An error has occurred. This application may no longer respond until reloaded.
  </environment>
  <environment include="Development">
    An unhandled exception has occurred. See browser dev tools for details.
  </environment>
  <a href="" class="reload">Reload</a>
  <a class="dismiss">X</a>
</div>

<script src="_framework/blazor.server.js"></script>
<script src="OSM.js"></script>
</body>
</html>
```

Figure 3 \_Host.cshtml

Zaciągnięcie skryptów do **OpenStreetMap** oraz zaimplementowanych skryptów.

Wygląd poszczególnych komponentów zaczerpnięty jest z biblioteki **leaflet**

```

var latlngt;
var popup = L.popup();
var map = L.map('map').setView([40.91, -96.63], 4);
var tiles = L.esri.basemapLayer("Streets").addTo(map);
var resultsMarker = L.layerGroup().addTo(map);

L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
  attribution: '&copy; <a href="https://osm.org/copyright">OpenStreetMap</a> contributors'
}).addTo(map);

export function onButtonGet() {
  // console.log(addressText);
  var addressText = document.getElementById('address').innerHTML;
  //działa
  L.esri.Geocoding.geocode().text(addressText).run(function (err, results, response) {
    console.log(results.results[0].latlng.lat);
    latlngt = results.results[0].latlng;
    document.getElementById('xy').innerHTML = latlngt.toString().slice(6);
    map.setView([results.results[0].latlng.lat, results.results[0].latlng.lng], 16);
    resultsMarker.addLayer(L.marker(results.results[0].latlng));
  });
}

export function getX() {
  return latlngt.lat;
}

export function getY() {
  return latlngt.lng;
}

function onButtonSet() {
  console.log("wysłane");
}

function onMapClick(e) {
  resultsMarker.clearLayers();
  popup.setLatLng(e.latlng)
    .setContent("GEOCODE: " + e.latlng.toString().slice(6))
    .openOn(map);
  resultsMarker.addLayer(L.marker(e.latlng));
  latlngt = e.latlng;
  document.getElementById('xy').innerHTML = latlngt.toString().slice(6);
}

map.on('click', onMapClick);

```

Figure 4 OSM.js

Skrypt odpowiadający za implementację *mapy* , „*geokodera*” oraz wszelkich akcji z nimi związanymi.

```

@page "/"
using BlazorOSM.Models;
using System.Net.Http.Json;
using BlazorOSM.Services;
@inject IJSRuntime JsRuntime
@inject HttpClient http
<div class="container">
    <h1>Mateusz Kalksztejn , Filip Nowicki</h1>
    <h2 class="text-muted">Programowanie w środowisku Windows Projekt</h2>
    <div id="map"></div>
    <div>
        <button type="button" class="btn btn-primary" @onclick="GetData">Pobierz ulice</button>
        <button type="button" class="btn btn-primary" @onclick="GetGeo">Załaduj mapę</button>
        <button type="button" class="btn btn-primary" @onclick="SendCord">Wyślij</button>
    </div>
    <label id="address">@address</label>
    <div>
        <label id="xy"></label>
    </div>
</div>

@code
{
    DataRequest dataRequest = new DataRequest();

    String address = "";
    String xy;
    String cityAndStreet;
    IJSObjectReference mapModule;

    protected override async Task OnAfterRenderAsync(bool firstRender)
    {
        if (firstRender)
        {
            mapModule = await JsRuntime.InvokeAsync<IJSObjectReference>(
                "import", "./OSM.js");
        }
    }

    Task setMap() =>
        mapModule.InvokeVoidAsync("onButtonGet").AsTask();

    async Task GetData()
    {
        address = "Proszę czekać...";
        cityAndStreet = await http.GetJsonAsync<String>("https://localhost:5001/api/GetStreet");
        address = cityAndStreet;
        Console.WriteLine(cityAndStreet);
    }

    async Task GetGeo()
    {
        await mapModule.InvokeVoidAsync("onButtonGet").AsTask();
    }

    async Task SendCord()
    {
        dataRequest.City = cityAndStreet.Split(",")[0]; //todo
        dataRequest.StreetName = cityAndStreet.Split(",")[1]; //todo
        dataRequest.StreetCordX = await mapModule.InvokeAsync<float>("getX");
        dataRequest.StreetCordY = await mapModule.InvokeAsync<float>("getY");

        Console.WriteLine(dataRequest.StreetCordX + " " + dataRequest.StreetCordY);

        await http.PostJsonAsync<Boolean>("https://localhost:5001/api/SaveToDatabase", dataRequest);
    }
}

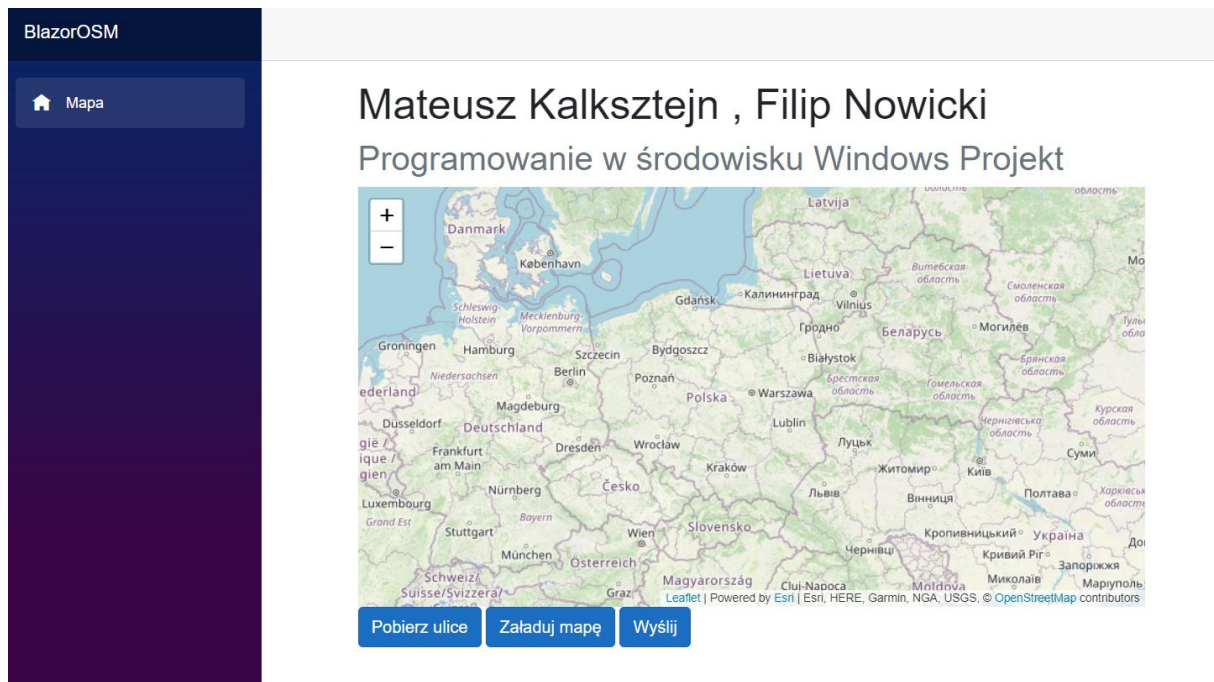
```

Figure 5 Index.razor

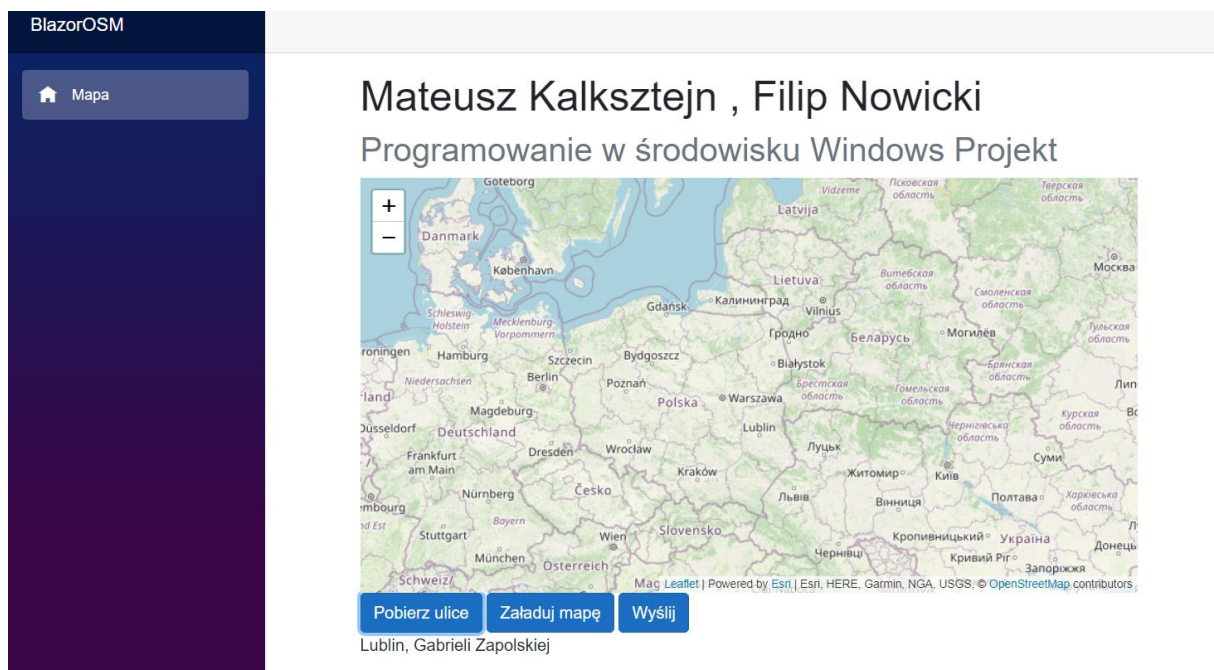
Implementacja **html** oraz logiki elementów widoku. Logika realizowana jest przy użyciu zadań wykonujących metody z skryptu w języku **JavaScript**.

## Wygląd aplikacji oraz sposób działania

Widok po uruchomieniu wygląda następująco:



Następnie używamy przycisku „Pobierz ulice„ by pobrać ulicę wraz z miastem:



Gdy znamy tą ulicę klikamy przycisk „Załaduj mapę” :





## Wnioski

**Api Tyret** nie jest przystosowane do zadań określonych w temacie zadania lecz portal posiada pliki z których pomocą wykonanie zadania zawartego w temacie projektu jest możliwe.

**Blazor** jest narzędziem dzięki któremu przy znajomości języka C# możliwa jest implementacja graficzna aplikacji webowej.

Wykorzystane zostało w projekcie Api **OpenStreetMap** które jest całkowicie darmowe ,co jest wielkim plusem w porównaniu do map **Google**.

W związku na konieczność implementacji map oraz „geokodera” konieczne było wykorzystanie języka **JavaScript**.