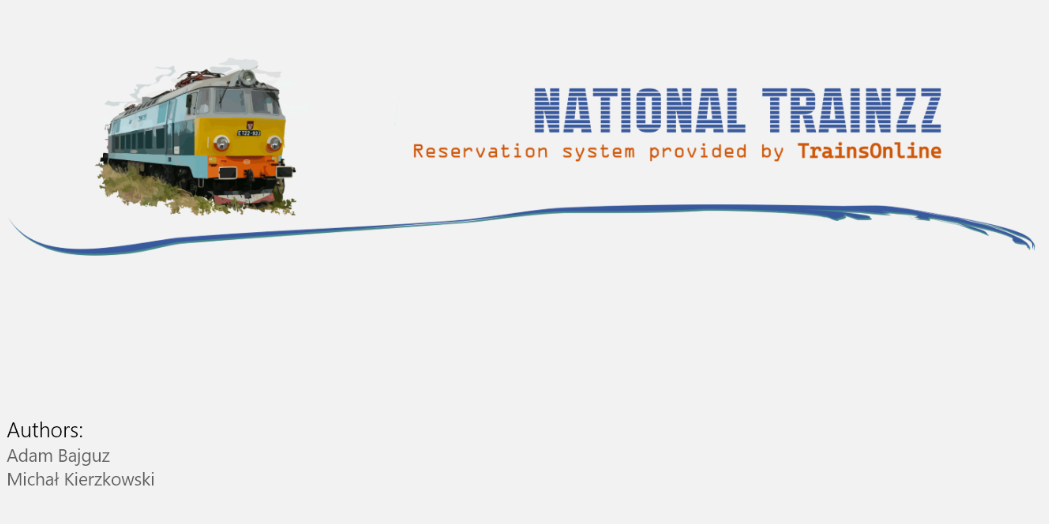
|  |
| --- |
| WYDZIAŁ INFORMATYKI POLITECHNIKI BIAŁOSTOCKIEJ    ROZPROSZONE SYSTEMY INTERNETOWE |

**Projekt z użyciem SOAP WS**

**Temat: System rezerwacji biletów kolejowych**



Białystok, 4.05.2020

|  |  |
| --- | --- |
| Wykonujący:  Adam Bajguz  Michał Kierzkowski  Grupa PS3 | Prowadzący:  dr hab. inż. Agnieszka Drużdżel prof. PB |

Spis treści

[1.Wprowadzanie 3](#_Toc42361666)

[1.1.Zakres projektowy 3](#_Toc42361667)

[1.2.Funkcjonalności 3](#_Toc42361668)

[1.3. Definicje, Akronimy, Skróty 4](#_Toc42361669)

[1.4. Użyte technologie 6](#_Toc42361670)

[1.4.1.Po stronie serwera 6](#_Toc42361671)

[1.4.2. Po stronie klienta 11](#_Toc42361672)

[1.5. Swagger 12](#_Toc42361673)

[1.6.ReDoc 12](#_Toc42361674)

[2. Specyfikacja usługi Web Service 13](#_Toc42361675)

[2.1. Standardy sieciowe 13](#_Toc42361676)

[2.2. Adres usługi 13](#_Toc42361677)

[2.5. Dostępne usługi 13](#_Toc42361678)

[3.Instrukcja użytkownika aplikacji klienckiej 14](#_Toc42361679)

# 1.Wprowadzanie

Celem tego dokumentu jest przedstawienie funkcjonalności, technologii oraz zasad działania(w tym wymiany informacji) w usłudze sieciowej systemu rezerwacji biletów kolejowych oraz interfejsowym systemie klienckim.

## 1.1.Zakres projektowy

Zakresem projektu było stworzenie systemy, który pozwoli na przechowywanie bazy stacji kolejowych i tras, na jakich przemieszczają się pociągi, a także umożliwi zakup biletów na daną trasę oraz pobranie go w formacie PDF. W tym celu należało stworzyć dwie aplikacje jedną, jako WebService dostarczający wszystkie funkcjonalności w API, zaś drugą, jako aplikacje kliencką umożliwiającą łatwy dostęp do zasobów serwisowych.

## 1.2.Funkcjonalności

System posiada funkcjonalności podstawowe takie jak:

* rejestracja klienta;
* logowanie na serwer;
* wyświetlanie bazy tras kolejowych (Miasto od , Miasto do, dzień, godzina);
* wyszukiwarkę tras;
* kupno biletu;
* odbiór potwierdzenia kupna w formacie PDF;
* sprawdzenie rezerwacji na podstawie podanego numeru biletu;
* dodawanie, usuwanie, modyfikowanie tras oraz stacji (funkcjonalność dostępna na Swagger lub SOAP UI tylko dla użytkowników z prawami administratora)

a także funkcjonalności dodatkowe:

* przypominanie hasła na podany podczas rejestracji adres email;
* szczegóły danej stacji kolejowej;
* prezentowanie na mapie położenia stacji;
* autoryzacja oparta na JSON Web Token;
* generowanie kodów QR które służą do dodania wydarzania do kalendarza o planie podróży oraz weryfikacji biletu
* dziennik logów zapisywany wewnętrznie na serwerze przy pomocy loggeraSerilog, a także możliwość zdalnego przechowywania logów na platformie Sentry
* wykorzystanie algrytmu Jaro-Winkler podczas wyszukiwania
* zgodność z OpenAPI 3.0.1
* wykorzystanie Swagger, SwaggerUI, ReDoc

## 1.3. Definicje, Akronimy, Skróty

**Representational State transfer (REST)** – styl architektoniczny definiujący format przesyłanych danych, utworzony w 2000 roku przez Roya Fieldinga w ramach rozprawy doktorskiej, jako element standaryzacji protokołu HTTP. Używany ze względu na elastyczność, szybkość i prostotę. Nie jest protokołem – jako usługę RESTową można zdefiniować cache’owany, bezstanowy, komunikujący się na zasadzie klient-serwer serwis.

**Web Service (WS)** - podstawowa technologia wykorzystywana w architekturze SOA. Jest to usługa systemu informatycznego, która może być używana przez inne komponenty programowe, również komponenty zewnętrzne projektowanego systemu. Web Service w paradygmacie SOA stanowi najistotniejszy interfejs projektowanego systemu. Web Service jest również wykorzystywany wewnętrznie do komunikacji pomiędzy komponentami systemu.

**Hypermedia as the Engine of Application State (HATEOAS** **) -** jest składnikiem architektury aplikacji REST, który odróżnia ją od innych architektur aplikacji sieciowych. Dzięki HATEOAS klient wchodzi w interakcję z aplikacją sieciową, której serwery aplikacji dostarczają informacje dynamicznie poprzez hipermedia . Klient REST nie potrzebuje żadnej wiedzy na temat interakcji z aplikacją lub serwerem, poza ogólną znajomością hipermediów.

**Secure Sockets Layer (SSL)**– jest protokołem do zabezpieczania komunikacji odbywającej się w Internecie. Dzięki niemu, wszystkie informacje wysyłane pomiędzy przeglądarką internetową a serwerem są szyfrowane. Aby z niego skorzystać, wymagana jest instalacja certyfikatu SSL.

**HTTP Strict Transport Security** (**HSTS**) – mechanizm bezpieczeństwa sieci, który chroni strony przed atakami takimi, jak wymuszone zmniejszenie poziomu protokołu oraz przechwytywanie sesji. Dzięki niemu do serwerów można połączyć się tylko za pomocą przeglądarek, korzystających z bezpiecznych połączeń HTTPS, natomiast nigdy nie dopuszcza on połączeń na bazie niezabezpieczonego protokołu HTTP. HSTS jest uznawany za standardowy protokół przez IETF.

**JSON Web Token** - to internetowy standard do tworzenia, tokenów dostępu opartych na JSON. Tokeny są podpisywane przy użyciu prywatnego tajnego klucza lub klucza publicznego / prywatnego. Na przykład serwer może wygenerować token z roszczeniem „zalogowany, jako administrator” i przekazać go klientowi. Klient może następnie użyć tego tokena, aby udowodnić, że jest zalogowany, jako administrator.

**Model-view-viewmodel** ( **MVVM** ) to wzorzec architektoniczny oprogramowania, który ułatwia oddzielenie rozwoju graficznego interfejsu użytkownika ( *widok* ) - czy to poprzez język znaczników lub kod GUI - od rozwoju logiki biznesowej lub back- logika końcowa ( *model* ), dzięki czemu widok nie jest zależny od żadnej konkretnej platformy modelu.

**Universal Windows Platform** (**UWP**) – interfejs API stworzony przez Microsoft i po raz pierwszy użyty w systemie Windows 10. Jego celem jest ułatwienie pisania aplikacji uniwersalnych, działających zarówno na komputerach, jak i telefonach, konsolach Xbox One oraz okularach HoloLens bez potrzeby ich przepisywania dla każdego rodzaju urządzeń.

**Globally Unique Identifier (GUID),** identyfikator globalnie unikatowy – identyfikator obiektów między innymi w systemie Windows lub wszędzie, gdzie potrzebny jest unikatowy identyfikator.

**OpenAPISpecification** – pierwotnie znana jako Swagger Specification, jest specyfikacją plików interfejsu do odczytu maszynowego do opisywania, tworzenia, używania i wizualizacji RESTful web services.

**Postman** – bezpłatnym narzędziem, które możemy wykorzystać do testów API, które zapewnia możliwości automatyzacji testowania, debugowania a także współpracy wielu osób za pomocą kolekcji, środowisk i wsparcia dla ciągłej integracji

**Swagger** – to platforma oprogramowania typu open source wspierana przez duży ekosystem narzędzi, które pomagają programistom projektować, budować, dokumentować i wykorzystywać usługi sieciowe RESTful.

**HTTP verbs** – definiuje zestaw metod żądań w celu wskazania pożądanej akcji do wykonania dla danego zasobu np.

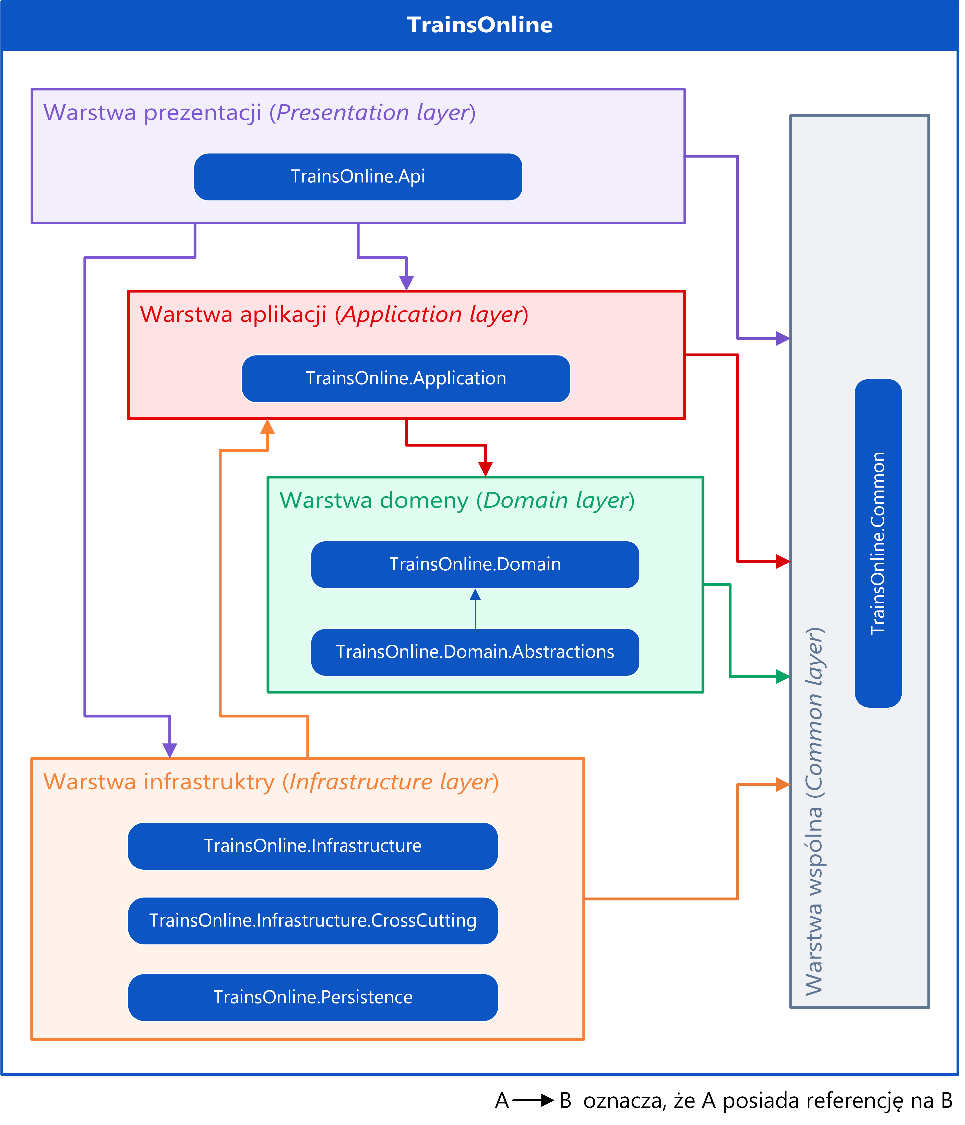
* GET - Metoda żąda reprezentacji określonego zasobu.
* POST - Metoda służąca przesłaniu encji do określonego zasobu, często powodując zmianę stanu.
* PUT - Metoda zastępuje wszystkie aktualne reprezentacje zasobu docelowego.
* DELETE - Metoda usuwa określony zasób.
* PATCH - Metoda służy do stosowania częściowych modyfikacji zasobu.

## 1.4. Użyte technologie

### 1.4.1.Po stronie serwera

Moduł TrainsOnline został zaimplementowany z użyciem podejścia określnego jako Clean Architecture, którego celem jest zapewnienie wysokiej skalowalności aplikacji i uniknięcie tworzenia monolitycznej architektury.Podejście to zrealizowano przy pomocy wzorca architektonicznego Domain Driven Design (DDD). Do stworzenia serwera aplikacji użyto: C# 8.0, NET Core 3.1.0, ASP.NET Core MVC, Microsoft SQL Server 2017. W implementacji zastosowano także następujące biblioteki:

| **Nazwa** | **Wersja** |
| --- | --- |
| AutoMapper.Extensions.Microsoft.DependencyInjection | 7.0.0 |
| AspNetCore.HealthChecks.MongoDB | 3.1.1 |
| CommandLineParser | 2.8.0 |
| CSharpVitamins.ShortGuid | 1.0.1 |
| FluentValidation.AspNetCore | 8.6.2 |
| F23.StringSimilarity | 3.1.0 |
| GemBox.Document | 31.0.1175 |
| System.Drawing.Common | 5.0.0-preview.3.20214.6 |
| SoapCore | 1.1.0-alpha |
| QRCoder | 1.3.9 |
| Humanizer.Core | 2.8.26 |
| Sciensoft.Hateoas | 3.1.0 |
| MediatR | 8.0.1 |
| MediatR.Extensions.Microsoft.DependencyInjection | 8.0.0 |
| Microsoft.AspNetCore.Authentication.JwtBearer | 3.1.3 |
| Microsoft.AspNetCore.Mvc | 2.2.0 |
| Microsoft.EntityFrameworkCore | 3.1.3 |
| Microsoft.EntityFrameworkCore.Design | 3.1.3 |
| Microsoft.EntityFrameworkCore.InMemory | 3.1. |
| Microsoft.EntityFrameworkCore.SqlServer | 3.1.3 |
| Microsoft.EntityFrameworkCore.Tools | 3.1.3 |
| Microsoft.Extensions.CommandLineUtils | 1.1.1 |
| Microsoft.Extensions.Configuration.EnvironmentVariables | 3.1.0 |
| Microsoft.Extensions.Configuration.Json | 3.1.0 |
| Microsoft.NET.Test.Sdk | 16.4.0 |
| MongoDB.Bson | 2.10.4 |
| MongoDB.Driver | 2.10.4 |
| MongoDB.Driver.Core | 2.10.4 |
| Newtonsoft.Json | 12.0.3 |
| Sentry.Serilog | 2.1.3 |
| Serilog | 2.9.0 |
| Serilog.AspNetCore | 3.4.0-dev-00168 |
| Serilog.Enrichers.Memory | 1.0.4 |
| Serilog.Enrichers.Process | 2.0.1 |
| Serilog.Enrichers.Thread | 3.2.0-dev-00747 |
| Serilog.Exceptions | 5.5.0 |
| Serilog.Sinks.Async | 1.4.0 |
| Serilog.Sinks.Console | 3.1.1 |
| Serilog.Sinks.File | 4.1.0 |
| SerilogTimings | 2.3.0 |
| Shouldly | 3.0.2 |
| Swashbuckle.AspNetCore | 5.4.1 |
| Swashbuckle.AspNetCore.Newtonsoft | 5.4.1 |
| Swashbuckle.AspNetCore.ReDoc | 5.4.1 |
| Swashbuckle.AspNetCore.Annotations | 5.4.1 |
| System.IdentityModel.Tokens.Jwt | 5.6.0 |
| System.Data.HashFunction.Core | 2.0.0 |
| System.Data.HashFunction.MurmurHash | 2.0.0 |
| xunit | 2.4.1 |
| xunit.runner.visualstudio | 2.4.1 |



Wzorzec DDD wymaga również podziału projektu na następujące warstwy:

1. wspólną – obejmuje wszystkie kwestie przekrojowe, czyli klasy i interfejsy wspólne dla różnych warstw;
2. prezentacji – zawiera kontrolery, punkty dostępowe do REST API oraz aplikację Swagger;
3. aplikacji – zawiera całą logikę. Warstwa jest zależna wyłącznie od warstwy domeny, są w niej zdefiniowane interfejsy, np. repozytorium lub jednostki pracy (Unit of Work), których implementacja znajduje się w innych warstwach. Oznacza to, że DDD wykorzystuje wzorzec odwrócenia sterowania (inversion of control). Przykładowo, jeśli aplikacja musi uzyskać dostęp do usługi powiadomień, do warstwy aplikacji zostanie dodany nowy interfejs, a implementacja zostanie utworzona w ramach warstwy infrastruktury;
4. domeny – zawiera dane, wyliczenia, wyjątki i logikę specyficzną dla domeny;
5. infrastruktury – zawiera klasy dostępu do zasobów zewnętrznych, takich jak: baza danych, systemy plików i poczta elektroniczna. Klasy te opierają się na interfejsach zdefiniowanych w warstwie aplikacji. W warstwie znajduje się również podwarstwa Persistence, w której zawarta jest cała konfiguracja bazy danych.

W projekcie z uwagi na małą liczbę obiektów domenowych, a tym samym tabel w bazie danych, zastosowano podejście grupowania klas w:

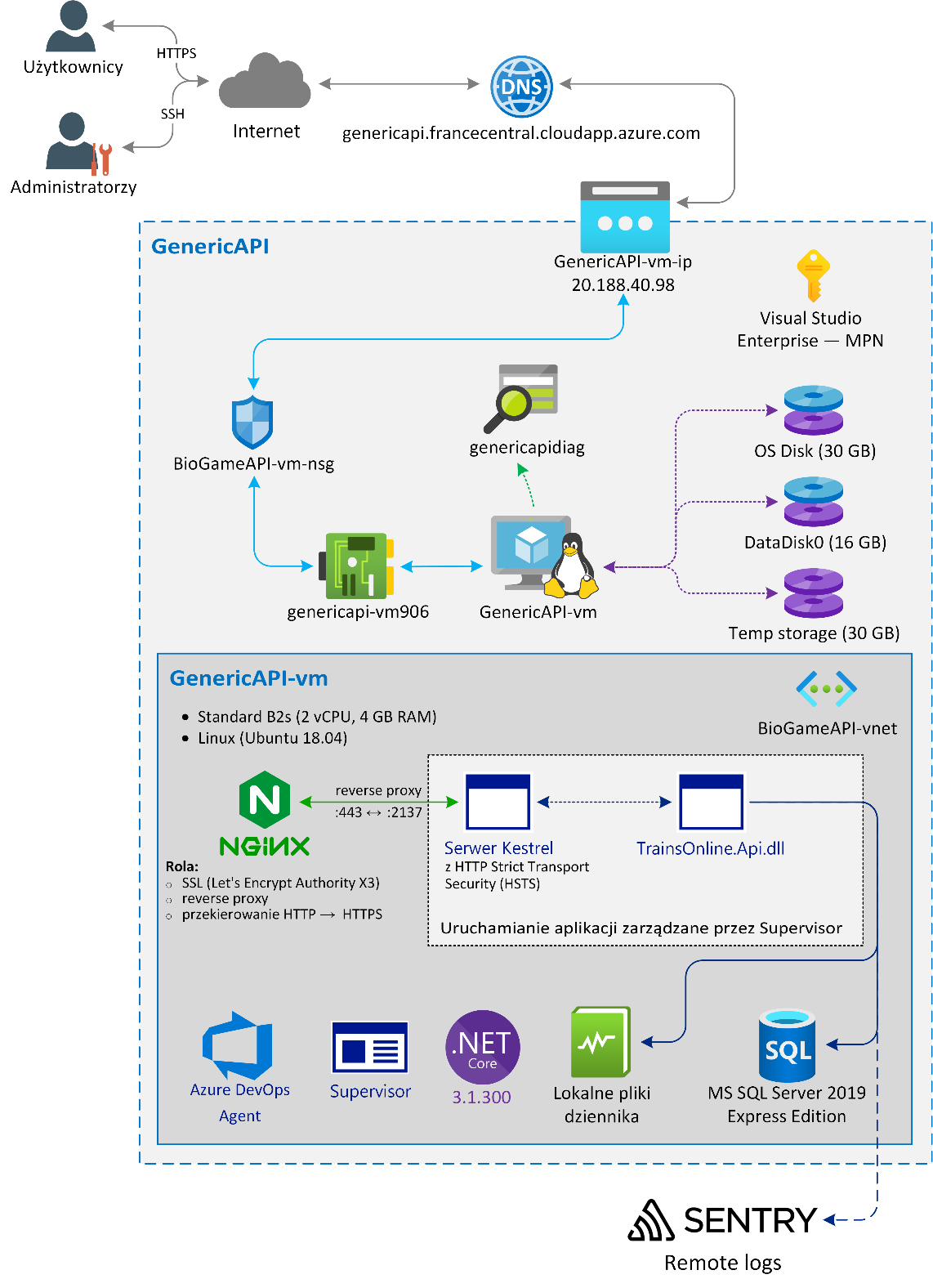
1. projekty (biblioteki) w oparciu o warstwy określone przez DDD (package by layer),
2. funkcjonalności (package by feature) na poziomie folderów i przestrzeni nazw w poszczególnych warstwach.

Rozwiązanie te jest wystarczające do zapewnienia czystego kodu oraz architektury w przypadku tego modułu. Ponadto nie wymaga implementowania m.in. koncepcji ograniczonych kontekstów (bounded contexts).

Moduł TrainsOnline został wdrożony na platformie Azure na maszynie wirtualnej typu Standard B2s, wyposażonej w dwa wirtualne procesory oraz 4 GB pamięci operacyjnej. Do maszyny podpięte są dwa dyski SSD typu Premium SSD, charakteryzujące się maksymalną liczbą operacji na sekundę (IOPS) równą 120 i przepływnością maksymalną na poziomie 25 MB/s, oraz jeden dysk tymczasowy o maksymalnej wartości IOPS równej 1600 i maksymalnej przepływności 15 MB/s :

* + OS Disk – dysk systemowy o pojemności 30 GB z systemem operacyjnym Ubuntu w wersji 18.04 oraz aplikacjami: .NET Core 3.1.0 Runtime, Microsoft SQL Server 2017 Express Edition, nginx, Azure DevOps Agent, supervizor;
  + DataDisk0 – dysk o pojemności 16 GB zawierający aplikację ‘’’oraz pliki dziennika aplikacji, tzw. logi;
  + Temp storage – dysk o pojemności 30 GB służący do przechowywania danych tymczasowych, np. plików tymczasowych serwera SQL, który ulega wyczyszczeniu m.in. po ponownym uruchomieniu maszyny wirtualnej.

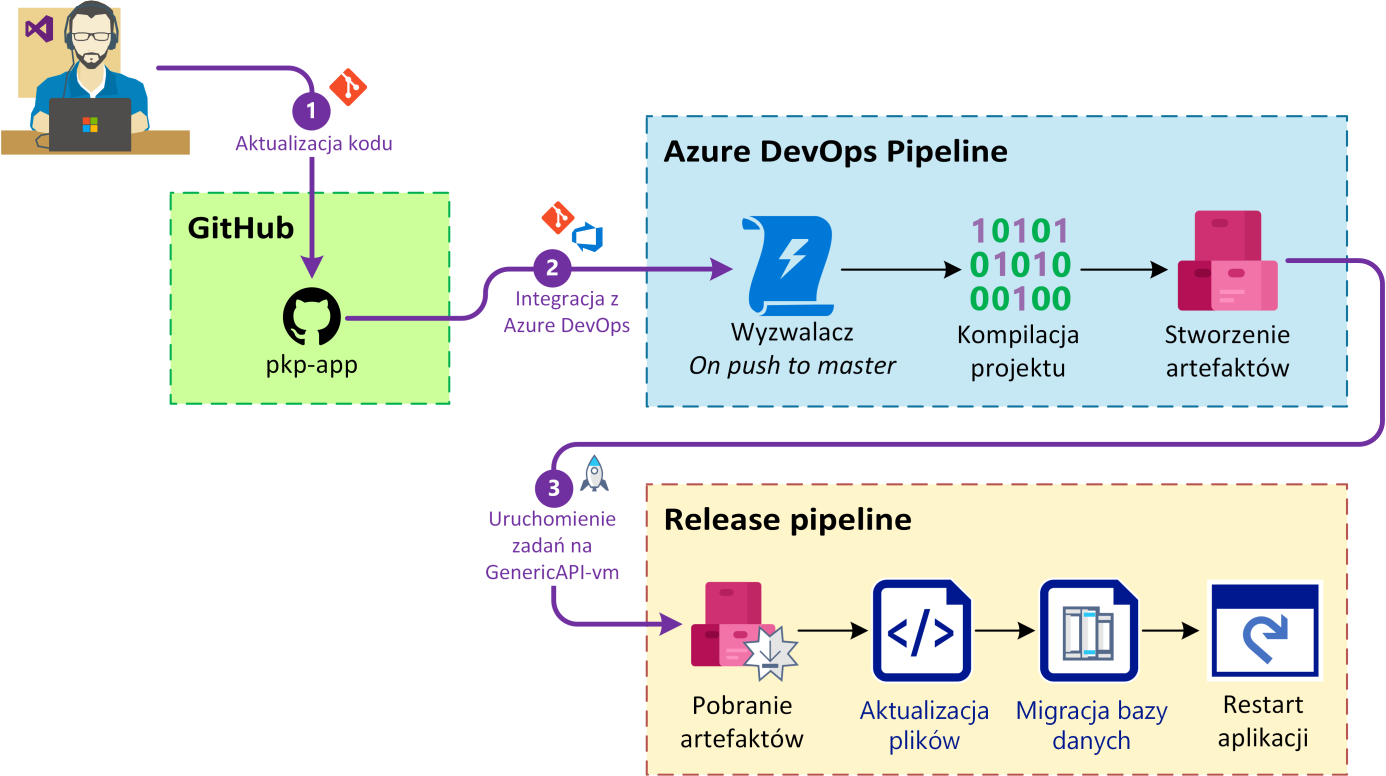
Zdalny dostęp do maszyny wirtualnej odbywa się poprzez protokół Secure Shell (SSH) z wykorzystaniem klucza RSA jako metody uwierzytelniania. Do połączenia się z maszyną wirtualną wykorzystano program Termius z pakietu GitHub Student Developer Pack. Dostarcza on wieloplatformowy terminal ze zintegrowanym klientem SSH. Dostęp przez użytkowników (klientów) do API odbywa się za pośrednictwem protokołu Hypertext Transfer ProtocolSecure (HTTPS). Działanie aplikacji można sprawdzić pod adresem https://genericapi.francecentral.cloudapp.azure.com/soap-api. Certyfikat na potrzeby protokołu HTTPS uzyskano za pomocą programu Certbot, urzędem certyfikacji jest Let'sEncrypt Authority X3. Aplikacja TrainsOnline uruchomiona jest na porcie 2137 za pomocą serwera Kestrel, który został użyty jako serwer graniczny. Komunikacja aplikacji z Internetem odbywa się za pomocą serwera nginx skonfigurowanego jako reverseproxy pomiędzy portami o numerach 443 oraz 2137. Aplikacja komunikuje się również z platformą Sentry, na której zapisywane są wszelkie zdarzenia błędów.



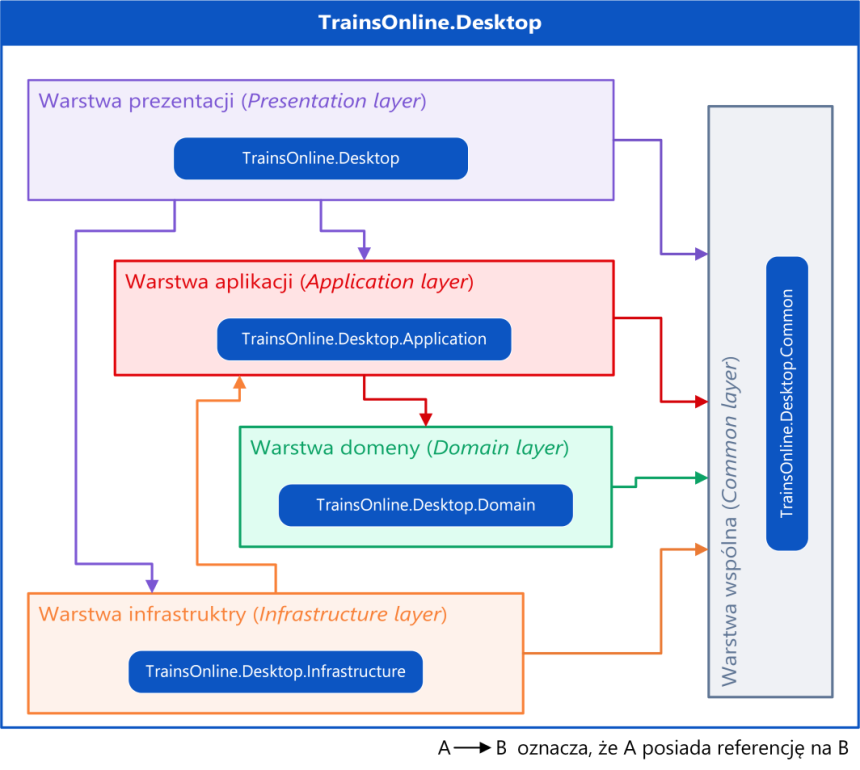
Zastosowanie ciągłej integracji (continuousintegration) oraz ciągłego dostarczania (continuousdelivery) umożliwiło łatwiejszą obsługę środowiska produkcyjnego TrainsOnline. Aktualizacje plików wykonywalnych, a także modyfikacje struktury bazy danych (migracje) odbywają się przy pomocy dwóch potoków:

1. uruchamianego na AzureDevOps w momencie pojawienia się zmian w gałęzi master repozytorium pkp-app, który buduje aplikację oraz przygotowuje paczkę z artefaktami;

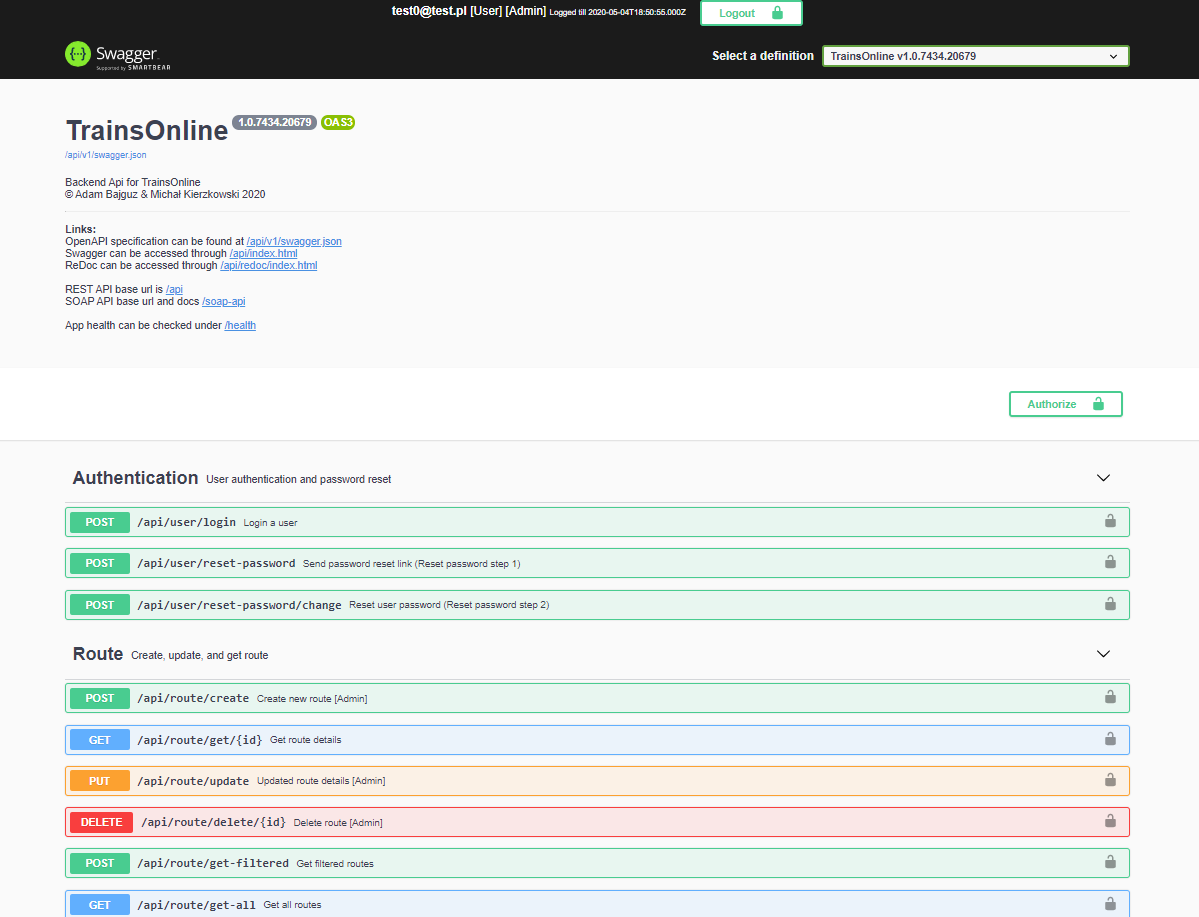
2. uruchamianego na maszynie wirtualnej z poziomu AzureDevOps (z udziałem programu AzureDevOps Agent), którego zadaniem jest m.in. pobranie paczki artefaktów i aktualizacja plików.



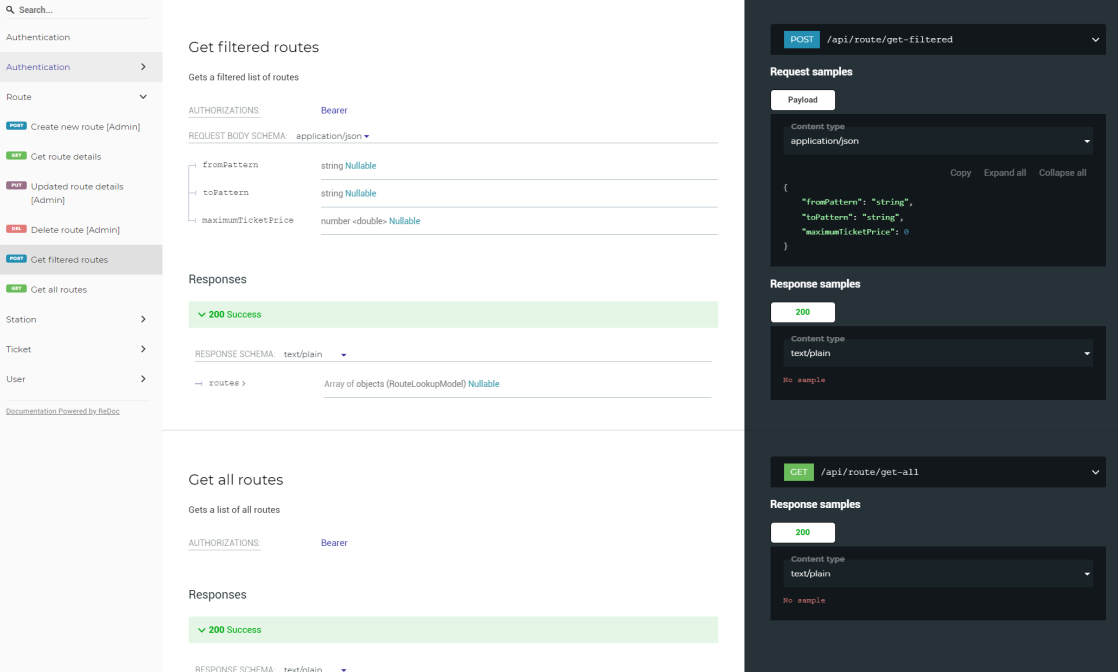
### 1.4.2. Po stronie klienta

 Aplikacja TrainsOnline.Desktop jest aplikacją UWPzawierającą całą logikę niezbędną do komunikacji z serwerem rezerwacji biletów kolejowych. Technologia UWP została zastosowana w celu implementacji łatwego i przyjaznego dla użytkownika interfejsu okienkowego o spójnym wyglądzie. Architektura aplikacji wykorzystuje wzorzec MVVM w celu zapewnienia oddzielenia interfejsu użytkownika od logiki biznesowej. Do implementacji MVVM zastosowano bibliotekę Caliburn.Micro. Ponadto w implementacji zastosowano podejście DDD – podobnie jak w aplikacji serwerowej. Jedyną różnicą jest brak warstwy Persistence.

## 1.5. Swagger



## 1.6.ReDoc



# 2. Specyfikacja usługi Web Service

## 2.1. Standardy sieciowe

Usługa „TrainsOnline” zaimplementowana została jako usługa sieciowa (Web Service) z użyciem protokołu SOAP. Usługa dostępna jest poprzez protokół HTTPS . Usługa dostępna jest pod adresem: https://genericapi.francecentral.cloudapp.azure.com/api, który prowadzi do strony zawierającej wszelkie dostępne endpointy na serwerze oraz dokumentację za pomocą swaggera.

## 2.2. Adres usługi

https://genericapi.francecentral.cloudapp.azure.com/api

## 2.5. Dostępne usługi

Usługi zostały podzielone na 5 grup użytkowych, reprezentowanych przez odpowiednie serwisy.

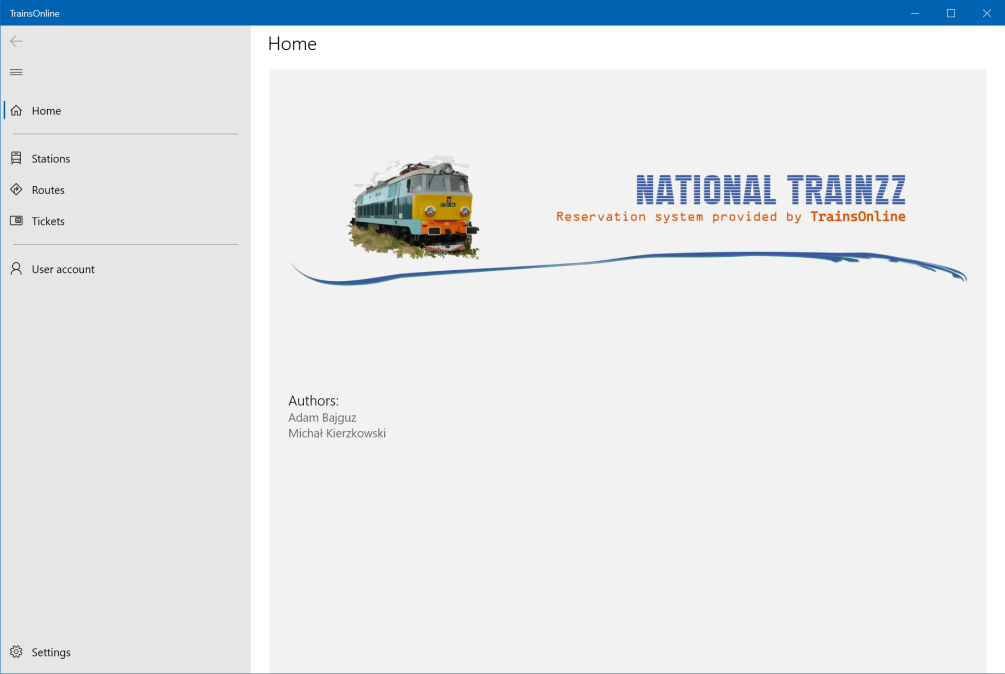
|  |  |
| --- | --- |
| Grupa użytkowa | Dostępne usługi |
| Autoryzacja | * Logowanie * Reset Hasła (krok 1 i 2) |
| Trasy kolejowe | * Utwórz trasę (A) * Szczegóły trasy (U) (A) * Modyfikacja trasy (A) * Usuwanie trasy (A) * Wypisanie wszystkich tras |
| Stacje kolejowe | * Utwórz stacji (A) * Szczegóły stacji (U) (A) * Modyfikacja stacji (A) * Usuwanie stacji (A) * Wypisanie wszystkich stacji |
| Bilety | * Tworzenie biletu (U) (A) * Szczegóły biletu (U) (A) * Tworzenie PDF biletu (U) (A) * Modyfikacja biletu (A) * Usuwanie biletu (A) * Wypisywanie wszystkich biletów danego użytkownika (U) (A) * Wypisywanie wszystkich biletów utworzonych w serwisie (A) |
| Użytkownik | * Tworzenie użytkownika * Szczegóły użytkownika (U) * Modyfikacja użytkownika (U) * Usuwanie użytkownika (U) * Zmiana hasła (U) * Wypisani wszystkich użytkowników (A) |

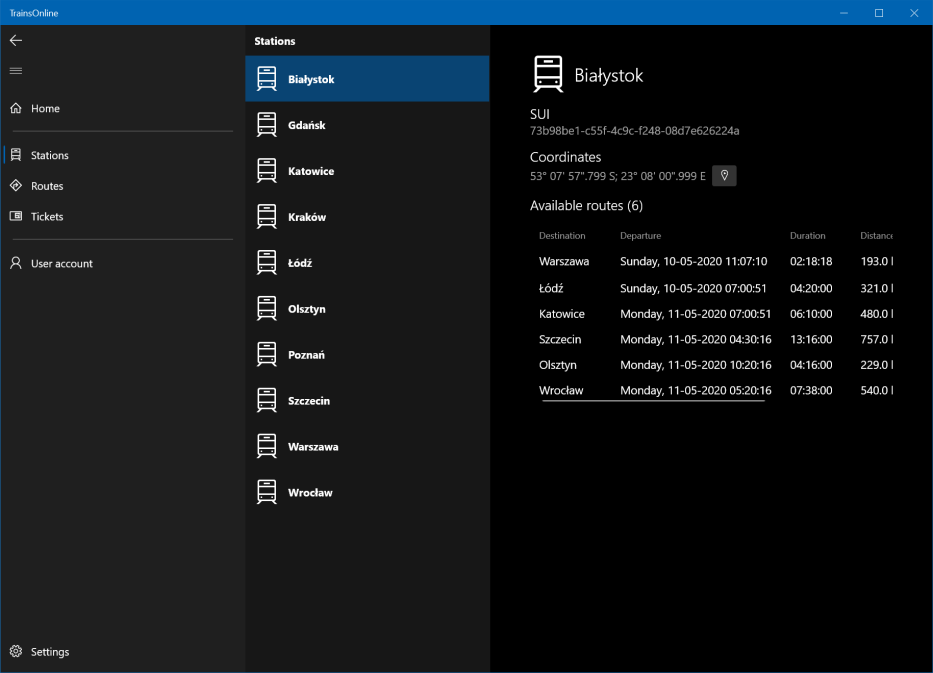
U - Usługa dostępna dla użytkownika serwisu

A -Usługa dostępna dla administratora serwisu

# 3.Instrukcja użytkownika aplikacji klienckiej

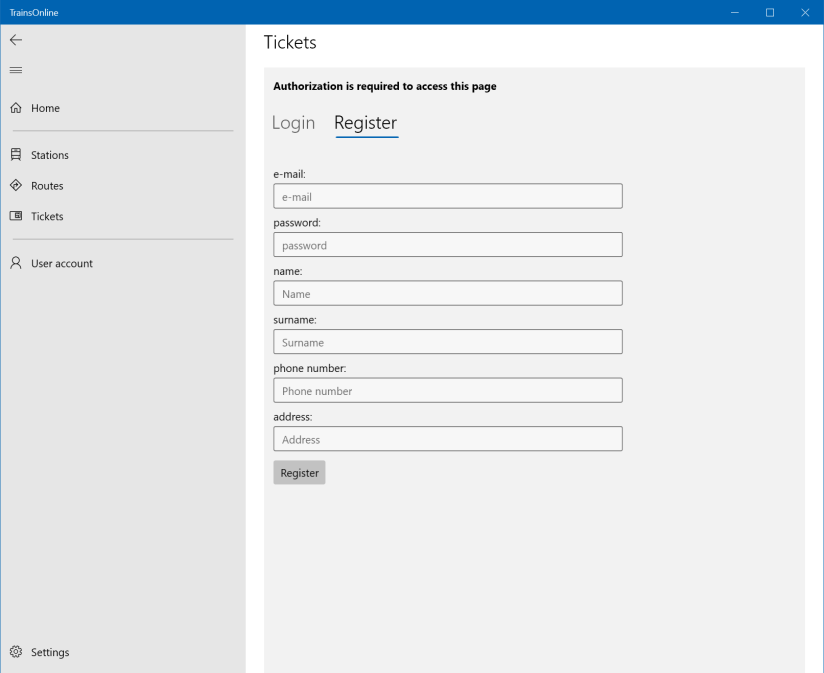
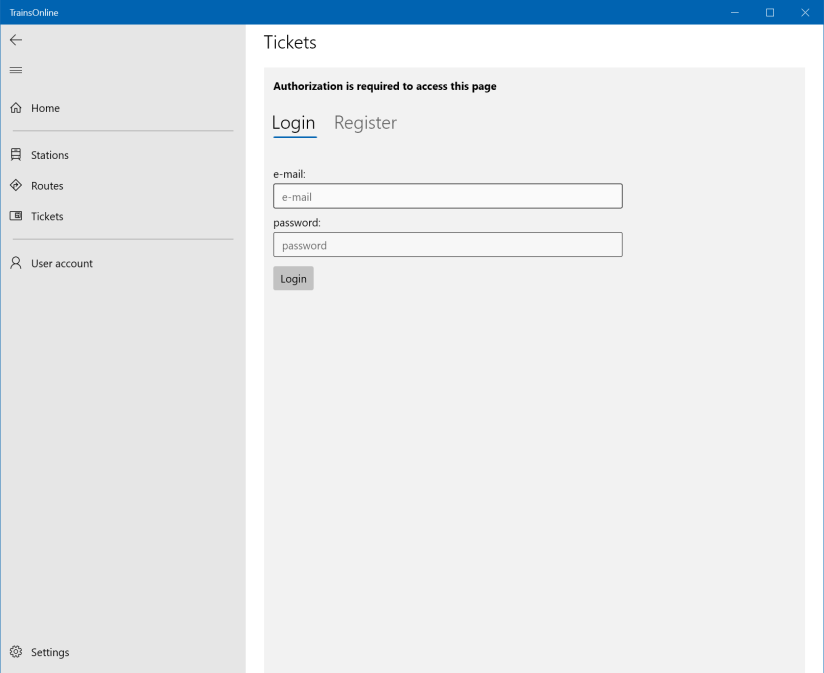
Program kliencki jest prostą i intuicyjną w obsłudze responsywną aplikacją okienkową powstałą w technologii Universal Windows Platform (UWP) i działa zarówno na komputerach, jak i telefonach z system Windows.



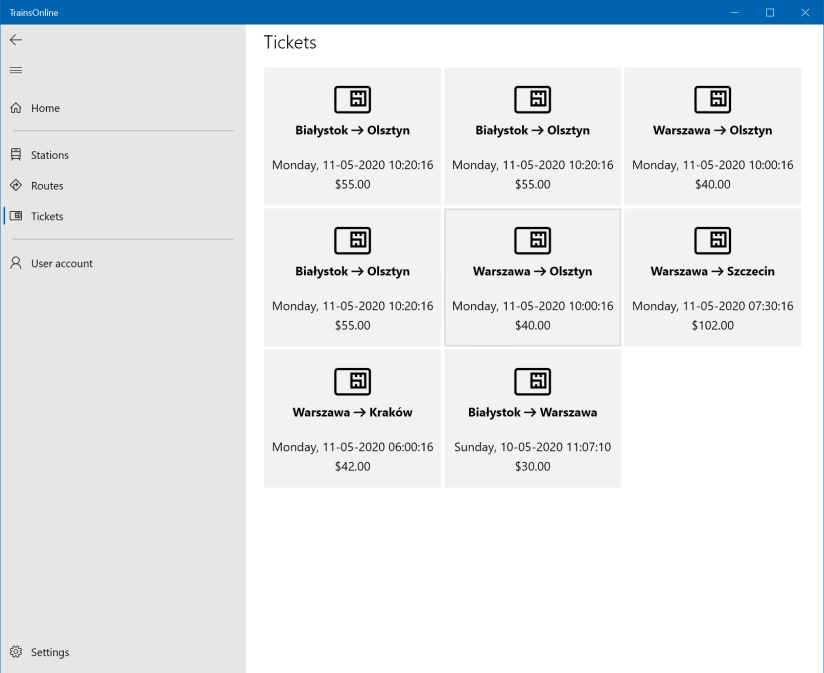
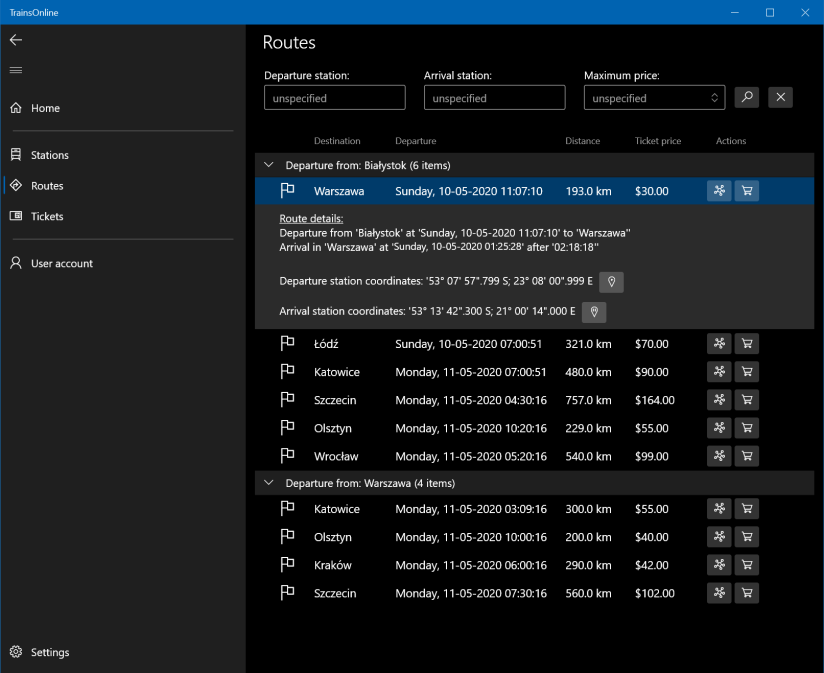


Posiada ona obsługę funkcjonalności API takich jak:

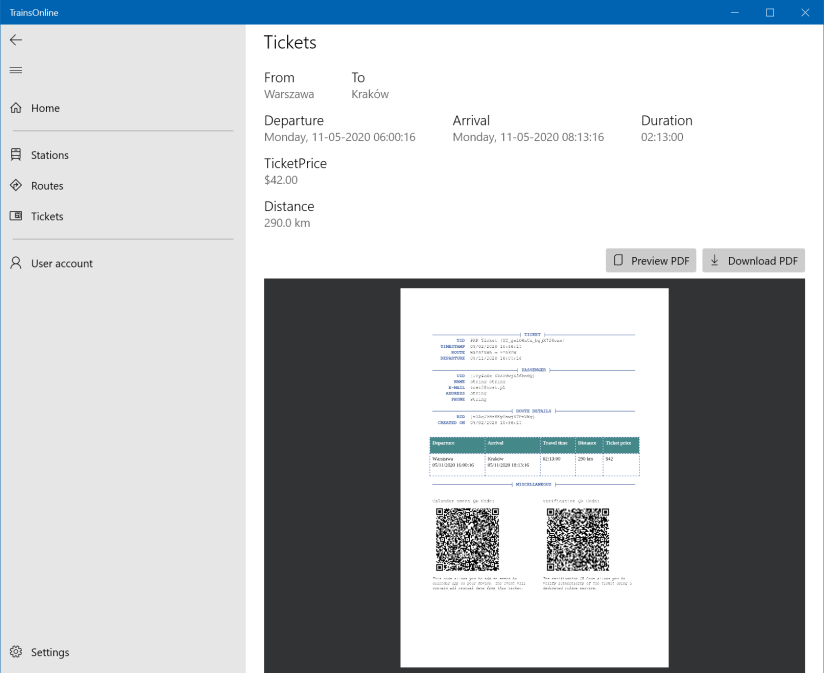
* logowanie i rejestracja;



* możliwość kupna biletów;



* pobranie biletu w formacie PDF;

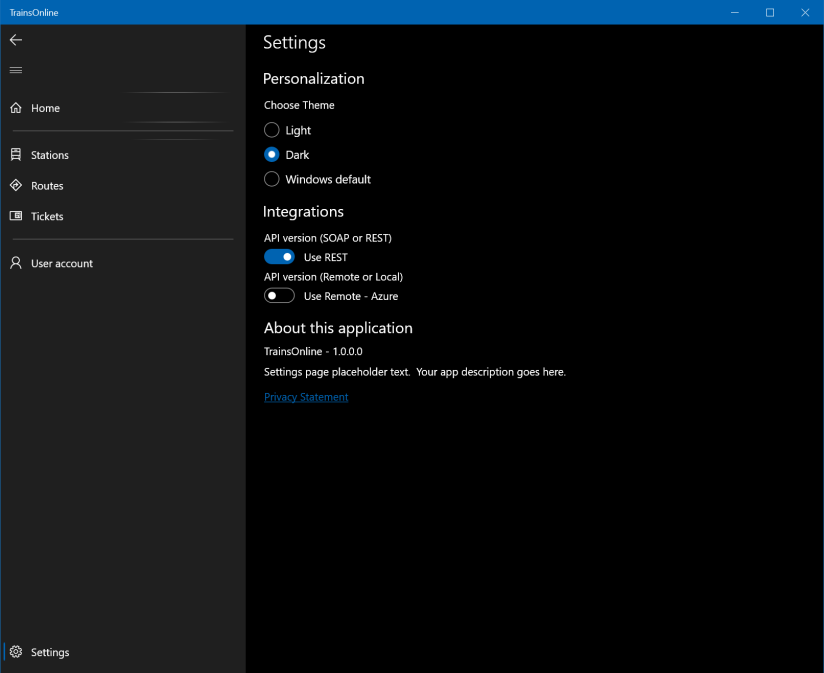




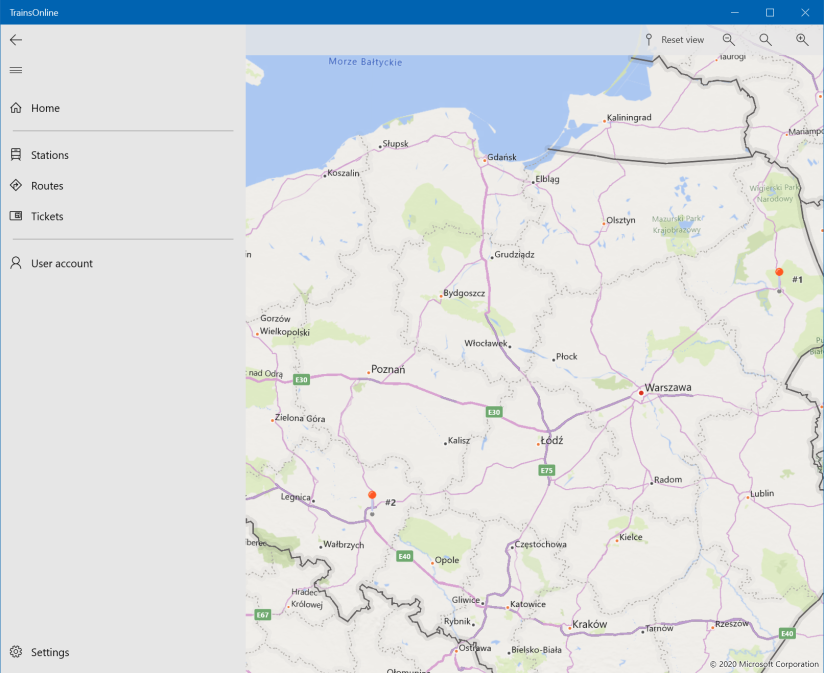
i wiele innych.

Poza spełnieniem podstawnych założeń aplikacja posiada dodatkowe funkcjonalności taki jak:

* możliwość wyboru stylu aplikacji( ciemny lub jasny)



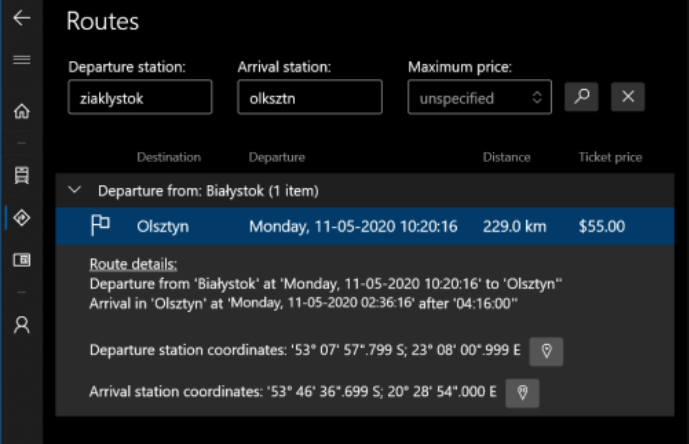
* możliwość podejrzenia lokalizacji stacji na mapie;



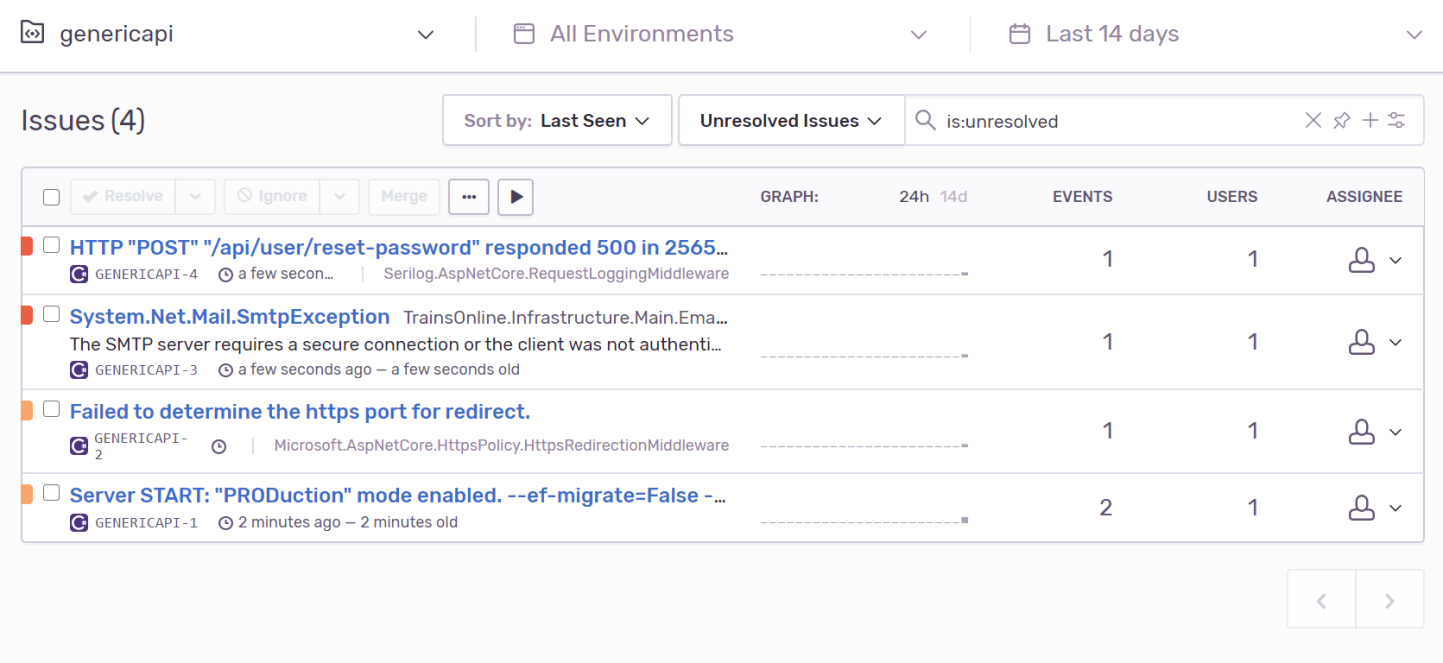
* generowanie kodów QR

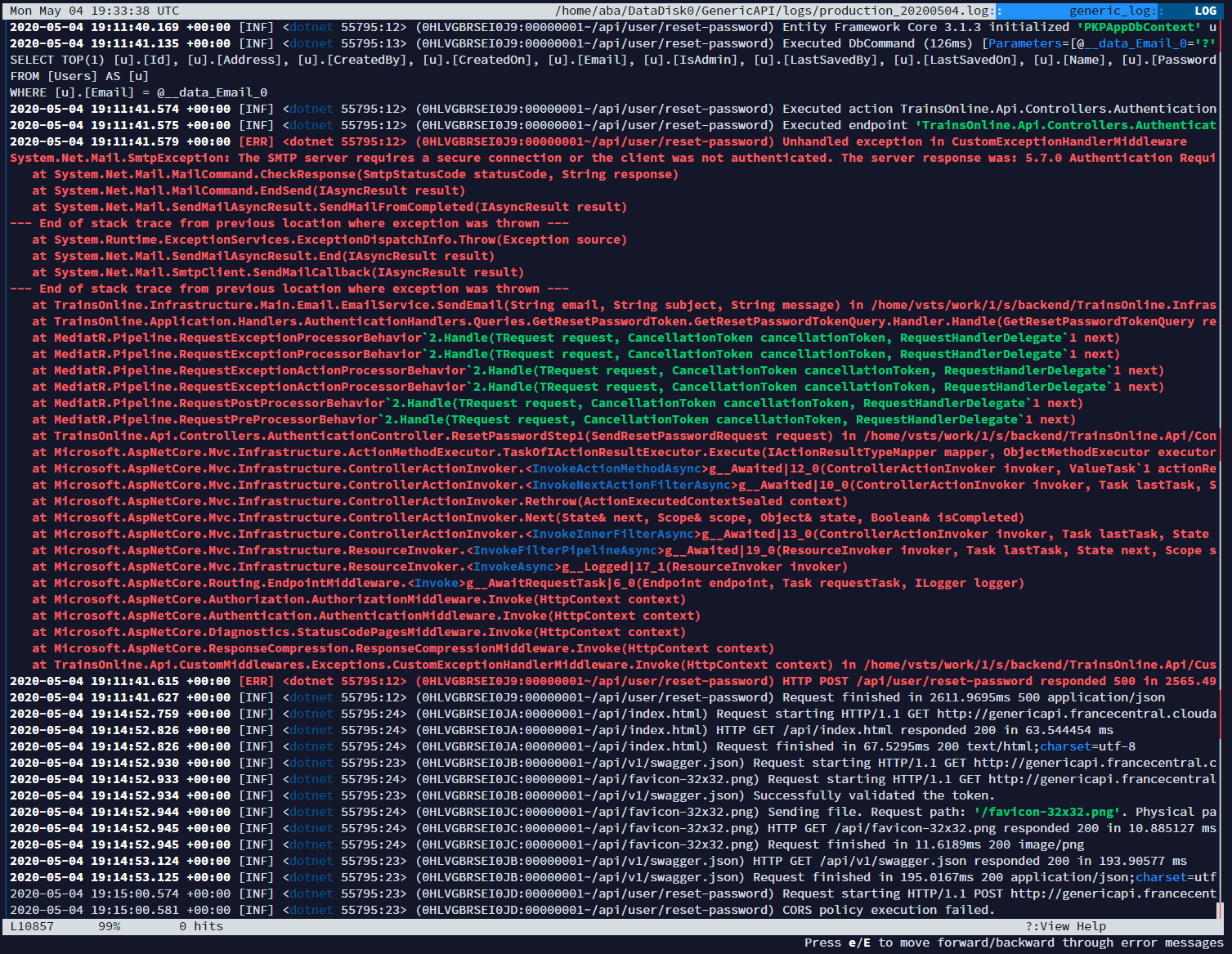


* wykorzystnia algorytmu Jaro-Winkler

****

* dziennik logów





* wysyłanie emaili

