|  |
| --- |
| WYDZIAŁ INFORMATYKI POLITECHNIKI BIAŁOSTOCKIEJ    Zaawansowane bazy danych i hurtownie danych |

**Projekt 1: baza danych typu NoSQL**

**Temat:** Rozszerzenie aplikacji TrainsOnline o   
raportowanie informacji o przebiegu trasy   
pociągu w bazie MongoDB

Białystok, 4.05.2020

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

Spis treści

[1. Wprowadzanie 3](#_Toc42957970)

[2. Funkcjonalności 4](#_Toc42957971)

[3. Projekt i implementacja apliakcji 5](#_Toc42957972)

[3.1.Model danych 5](#_Toc42957973)

[3.2. Architektura aplikacji i wykorzystane technologie 6](#_Toc42957974)

[3.2.1.Po stronie serwera 6](#_Toc42957975)

[3.2.2. Po stronie klienta 12](#_Toc42957976)

[2.3. Profil użytkownika końcowego 22](#_Toc42957977)

# 1. Wprowadzanie

Zakresem projektu było rozszerzenie aplikacji TrainsOnline. Istniejący system pozwalał 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. System rozszerzono o trzy funkcjonalności opierające się o nierelacyjną bazę danych:

1. logger pociągu,
2. cache
3. analityka.

Pierwsze z rozszerzeńm tj. logger pociągu pozwala na gromadzenie danych z trasy pociągu np. data i położenie geograficzne wszystko będzie zapisywane w bazie MongoDB. Po każdej trasie do bazy generowany jest raport zawierający takie dane jak średnia prędkość pociągu. W tym celu należało rozszerzyć WebService dostarczający wszystkie funkcjonalności w API oraz aplikację kliencką umożliwiającą łatwy dostęp do zasobów serwisowych, a także stworzyć trzecią aplikacji symulująca pociąg.

Drugie rozszerzenia, rozwiązano problem z dostępem do danych wymagających złożonych operacji na bazie lub generowania plików PDF poprzez zaimplementowanie mechanizmu cechowania danych opartego na nierelacyjnej bazie typu klucz-wartość.

Ostatnie z rozszerzeń, tj. rozszerzenie WebSerwisu o analitykę rozwiązano umożliwiło zbieranie danych analitycznych dotyczących ruchu sieciowego do WebAPI w taki sposób aby możliwa była analiza ilości wejść (zapytań) do poszczególnych adresów w serwisie, a także zebranie informacji o stosowanych klientach (np. przeglądarkach) oraz wykrycie zapytań od hakerów.

## 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)

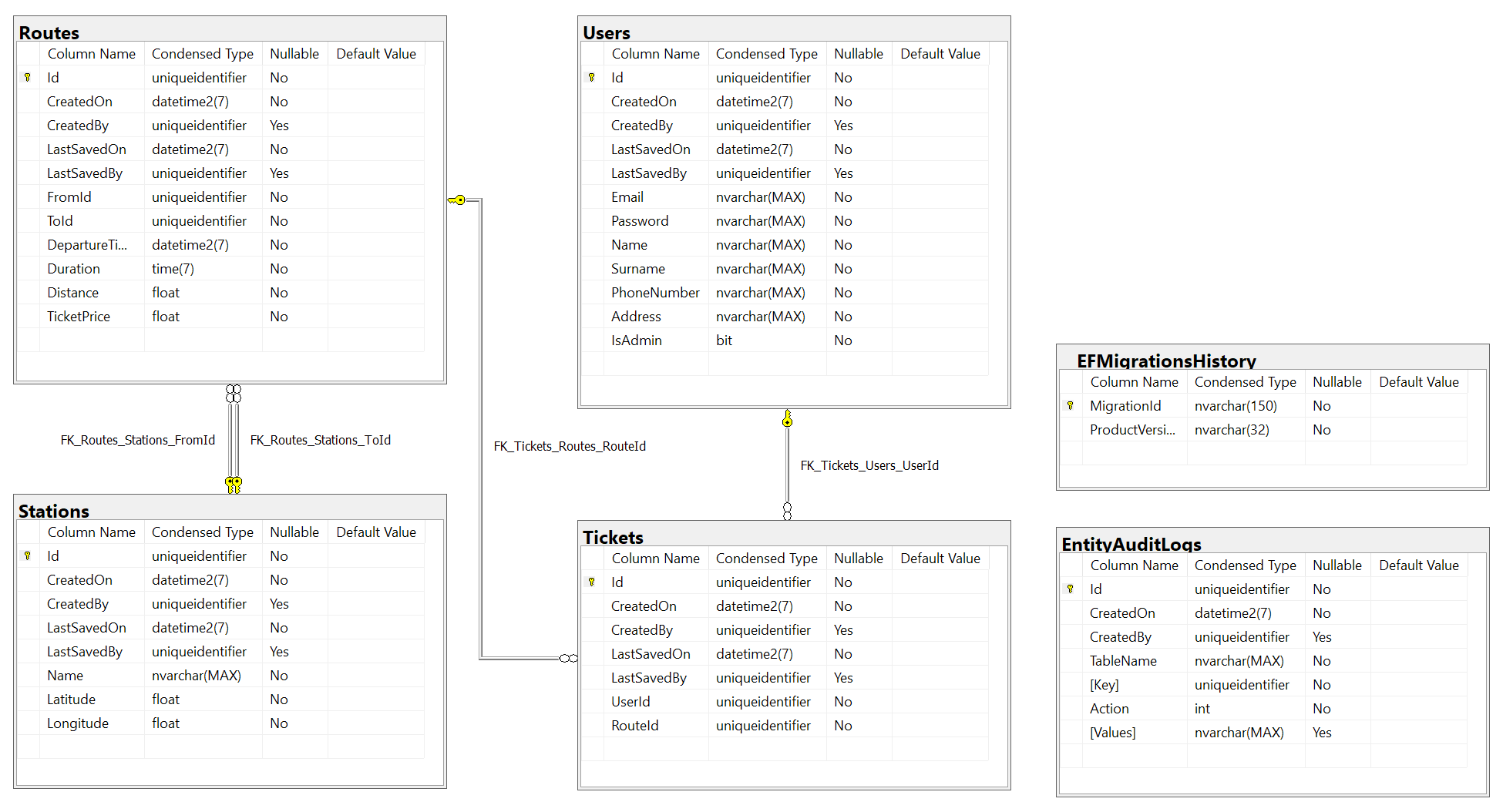
a także funkcjonalności dodatkowe:

* 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

# 3. Projekt i implementacja apliakcji

## 3.1.Model danych

Aplikacja wykorzystuje dwa modele danych: relacyjny i nierelacyjny. Model relacyjny składa się z 6 tabel, zawierających dane o użytkowniku, trasie, biletach, stacjach, historii zmian w elementach bazy (entity audit log).



Z kolei model nierelacyjny składa się z 3 kolekcji przechowywanych w MonogoDb, przechowujących logi z pociągu, raporty z tras pociągów, a także dane analityczne związane z ruchem po WebAPI i całej aplikacji serwerowej.

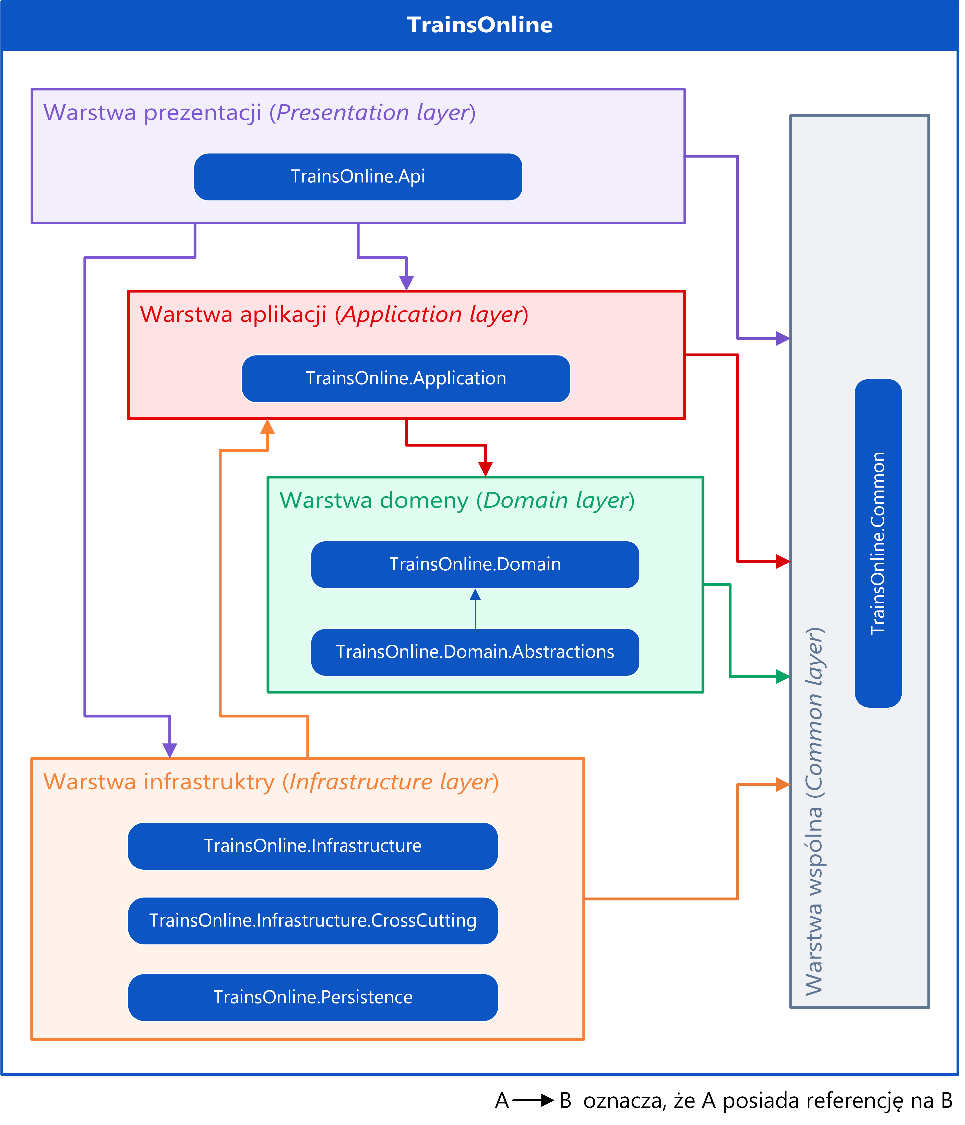


## 3.2. Architektura aplikacji i wykorzystane technologie

### 3.2.1.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 |
| Ceras | 4.1.7 |
| 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.

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.

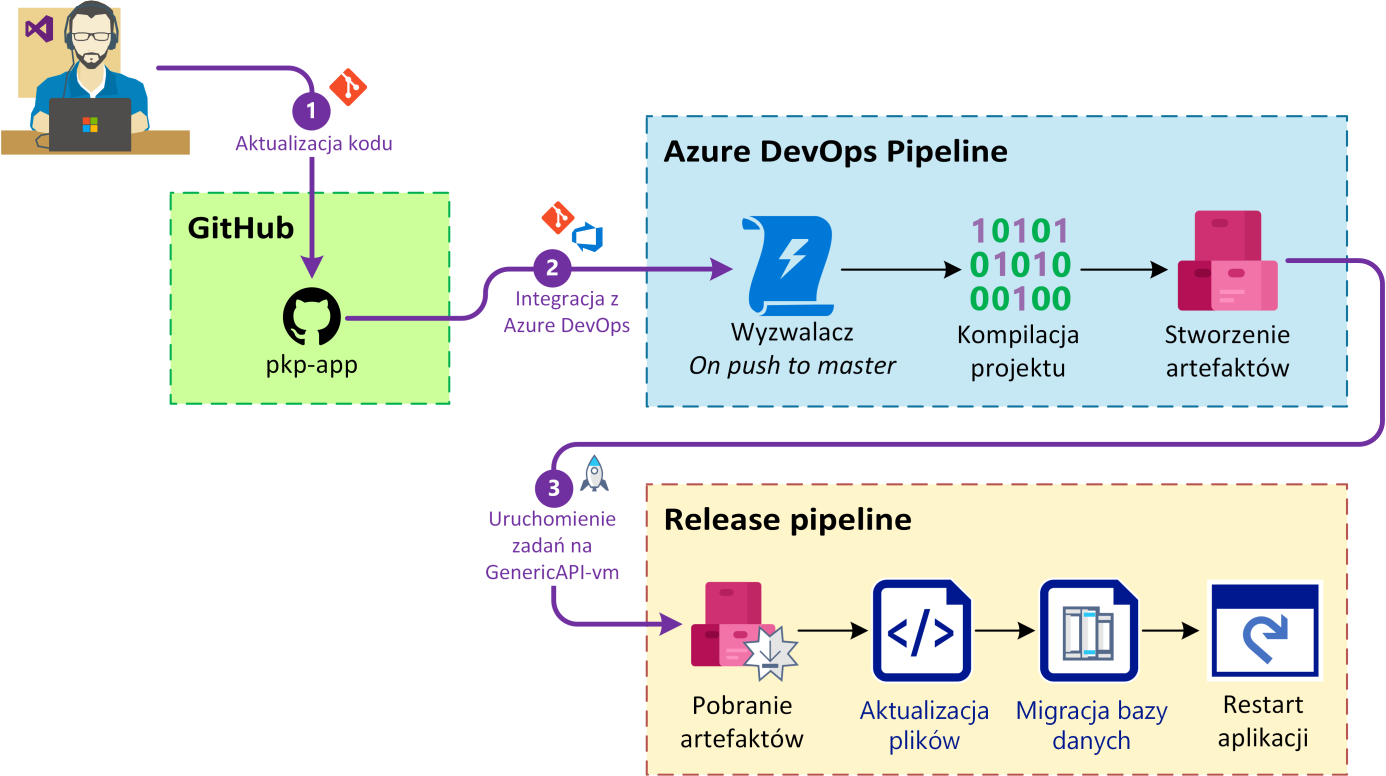
Zdalny dostęp do maszyny wirtualnej odbywa się poprzez protokół Secure Shell (SSH) z wykorzystaniem klucza RSA jako metody uwierzytelniania. Do połączenia się z maszyną wirtualną wykorzystano program Termius z pakietu GitHub Student Developer Pack. Dostarcza on wieloplatformowy terminal ze zintegrowanym 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.



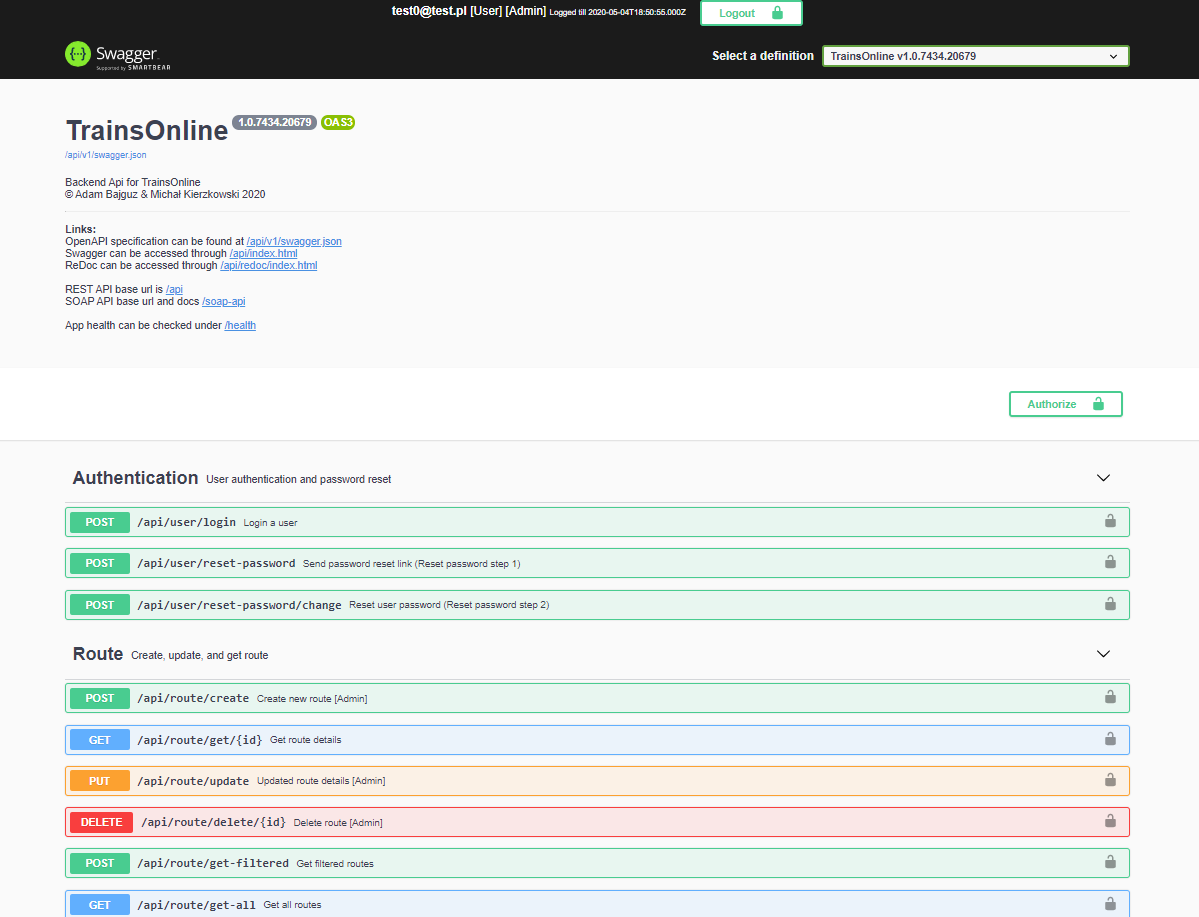
Zastosowanie ciągłej integracji (continuousintegration) oraz ciągłego dostarczania (continuousdelivery) umożliwiło łatwiejszą obsługę środowiska produkcyjnego TrainsOnline. Aktualizacje plików wykonywalnych, a także modyfikacje struktury bazy danych (migracje) odbywają się przy pomocy dwóch potoków:

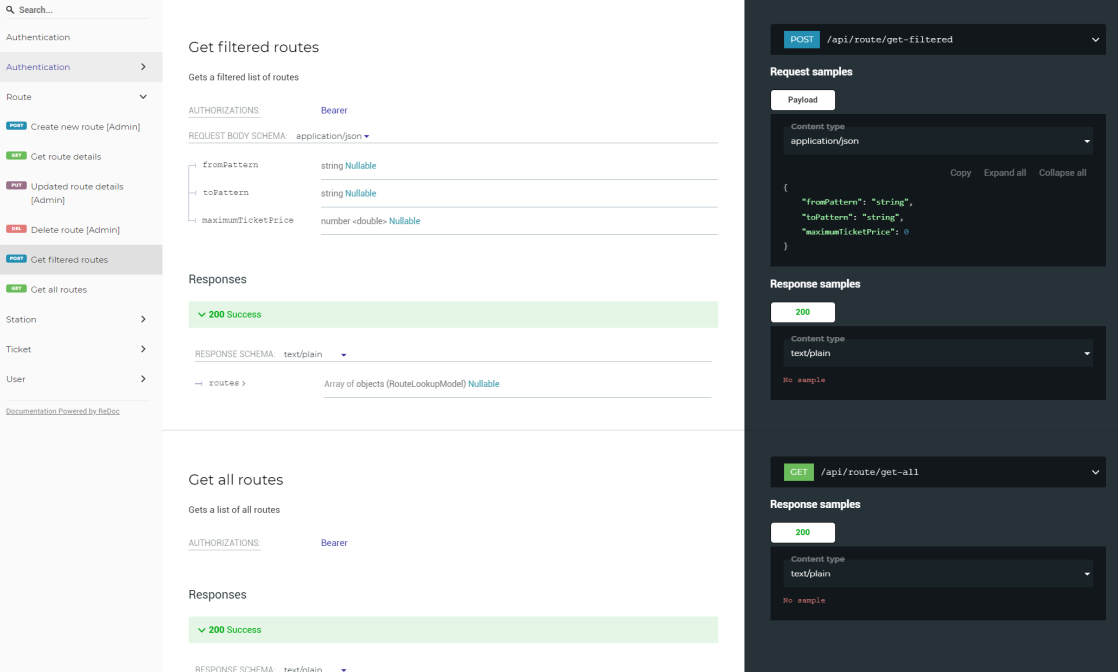
1. uruchamianego na AzureDevOps w momencie pojawienia się zmian w gałęzi master repozytorium pkp-app, który buduje aplikację oraz przygotowuje paczkę z artefaktami;

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

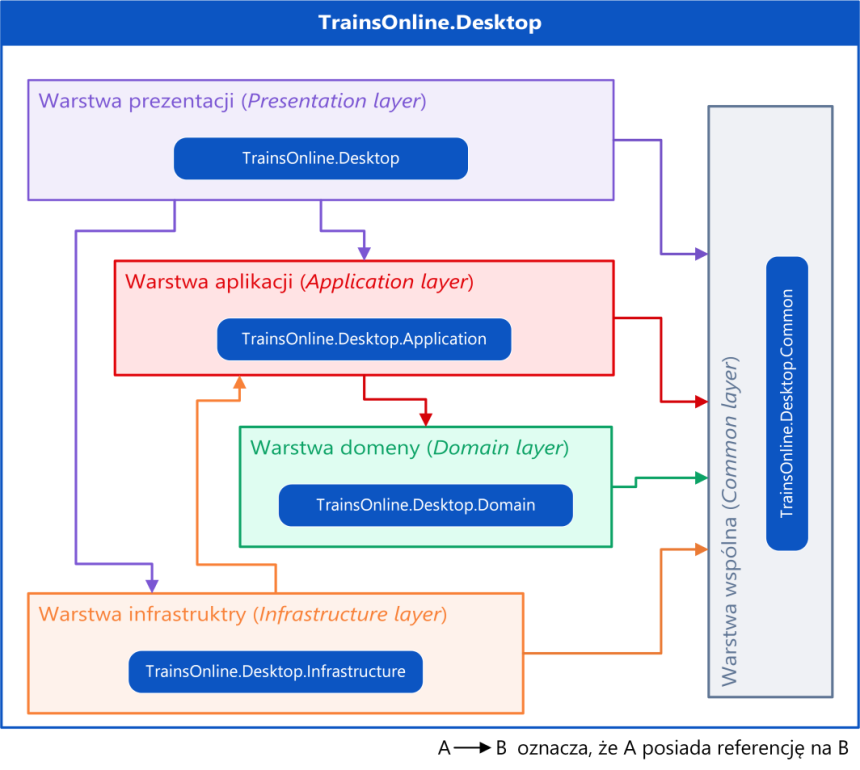


WebAPI zostało udokumentowane za pomocą Swaggera i ReDoc. Ponadto Swagger powala na łatwe testowanie i dostęp do funkcjonalności nawet przez niedoświadczonych użytkowników.

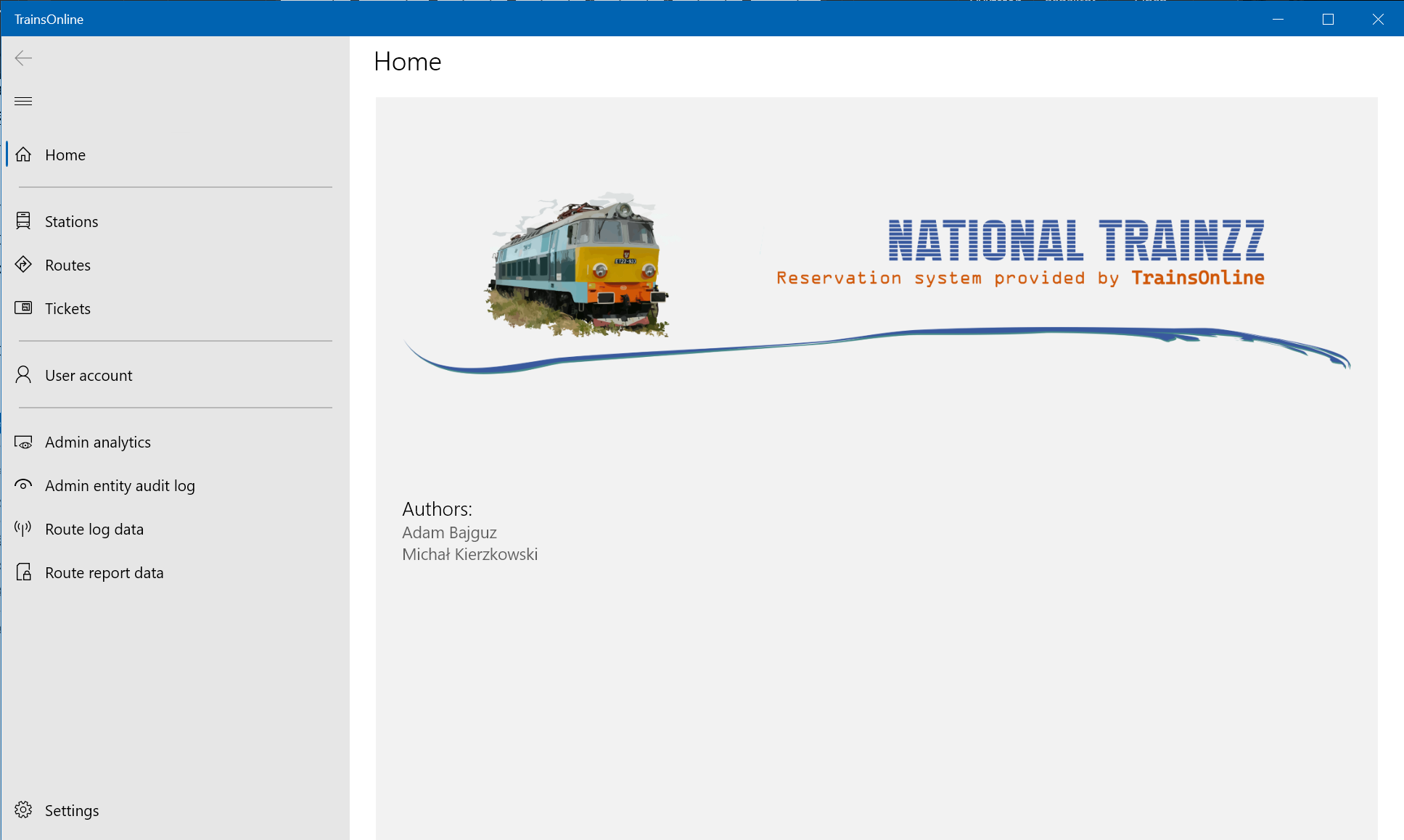


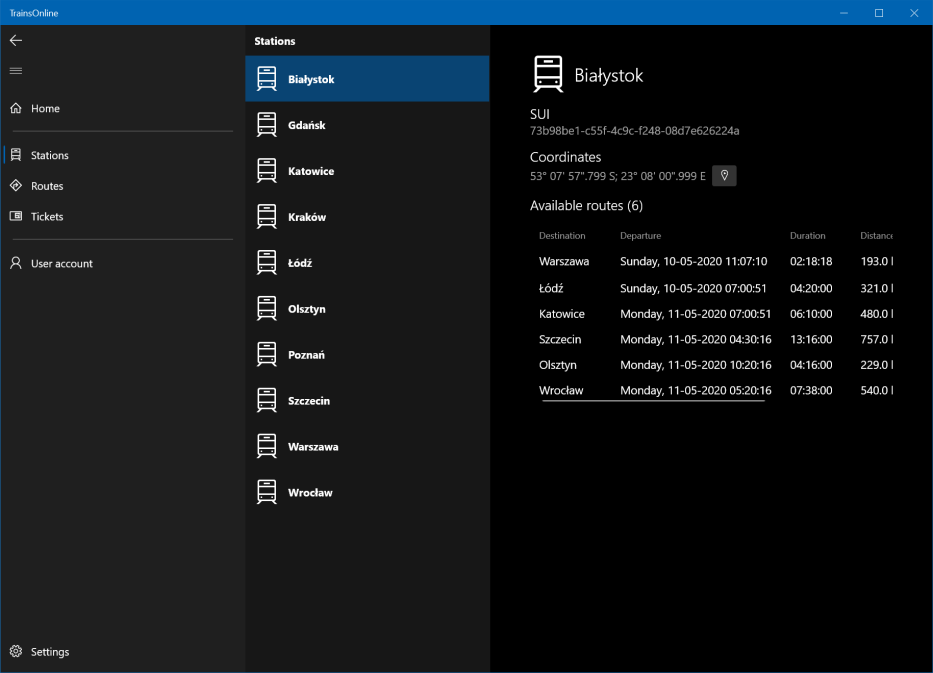


### 3.2.2. Po stronie klienta

 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.

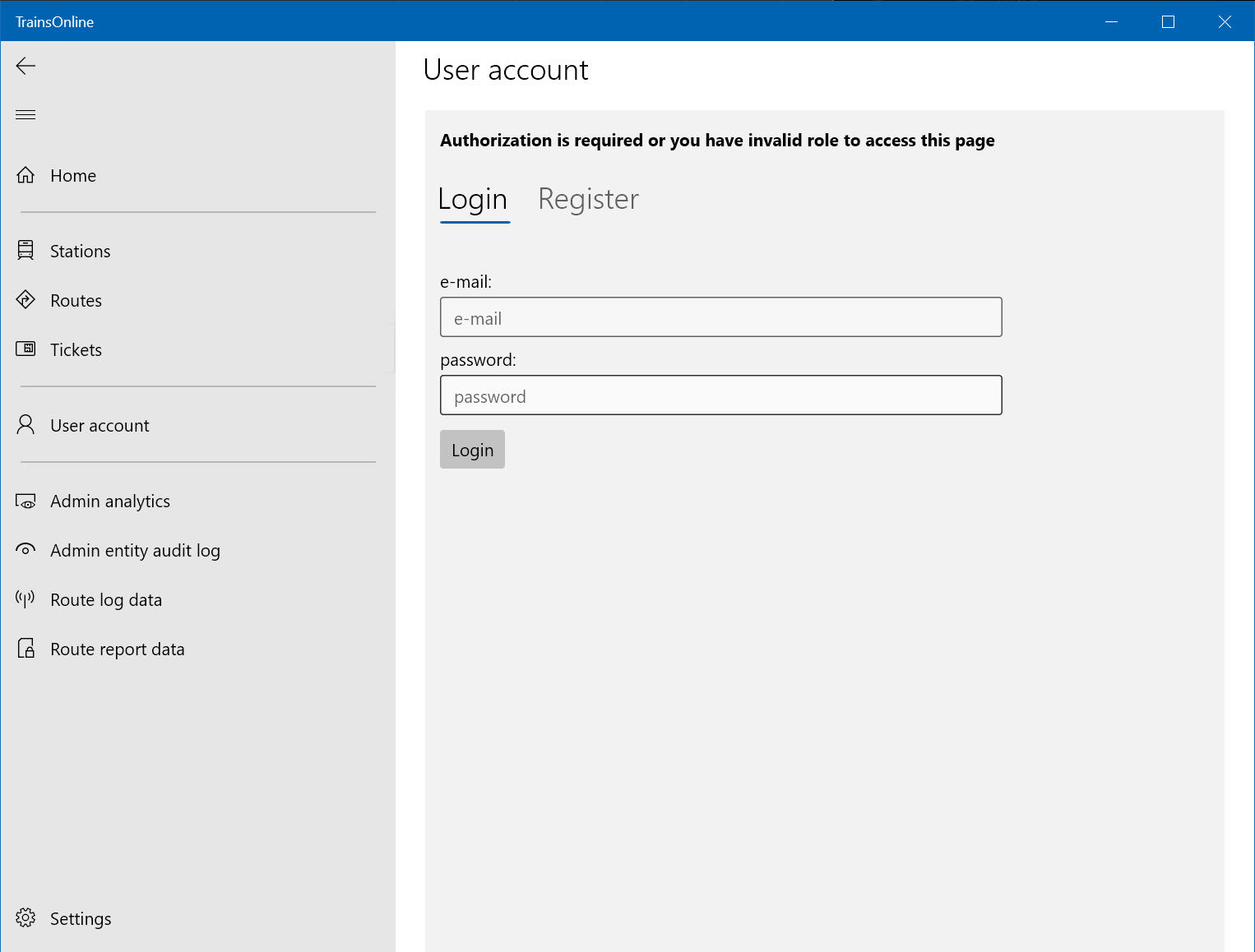
Program kliencki jest prostą i intuicyjną w obsłudze responsywną aplikacją okienkową powstałą w technologii Universal Windows Platform (UWP) i działa zarówno na komputerach, jak i telefonach z system Windows.

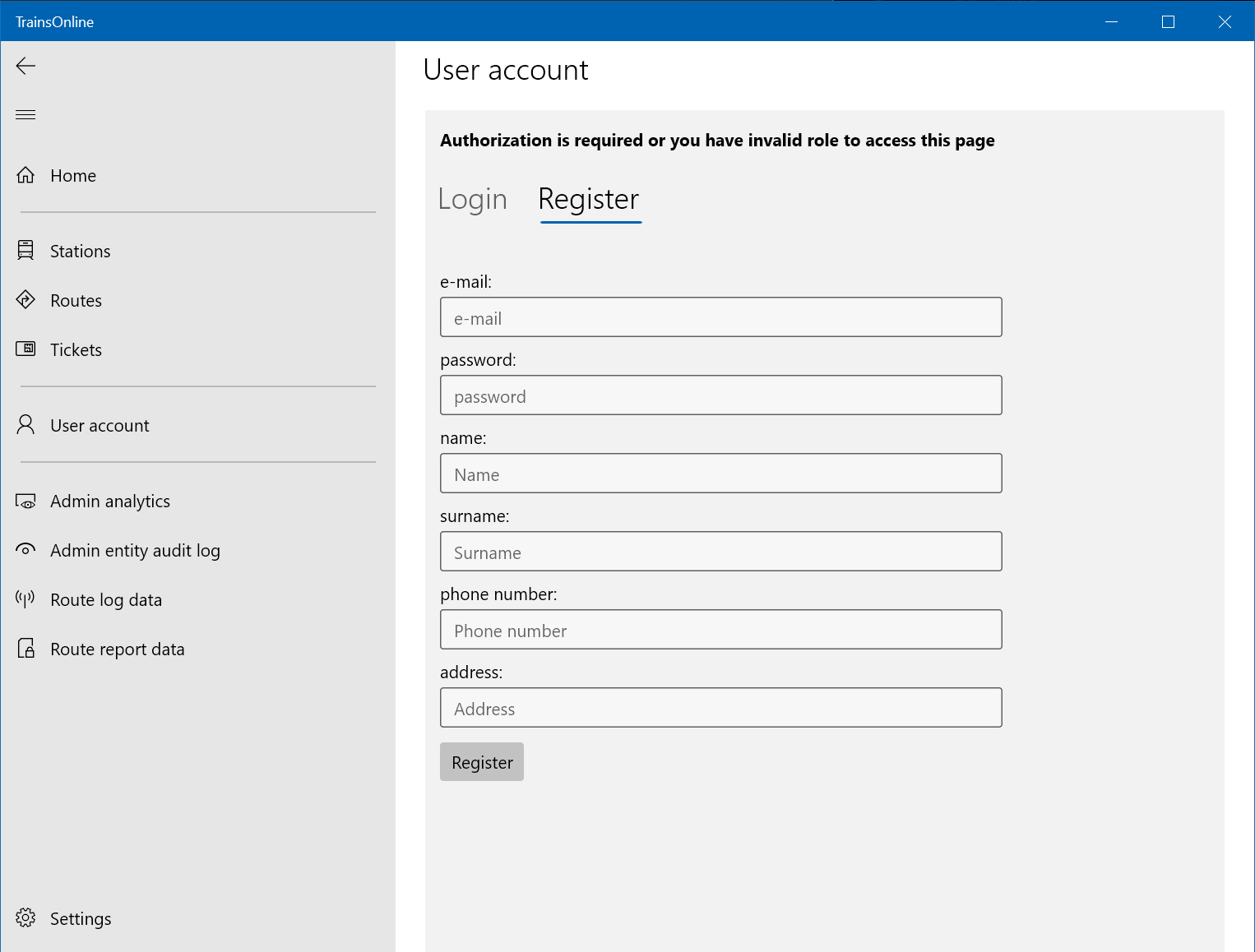




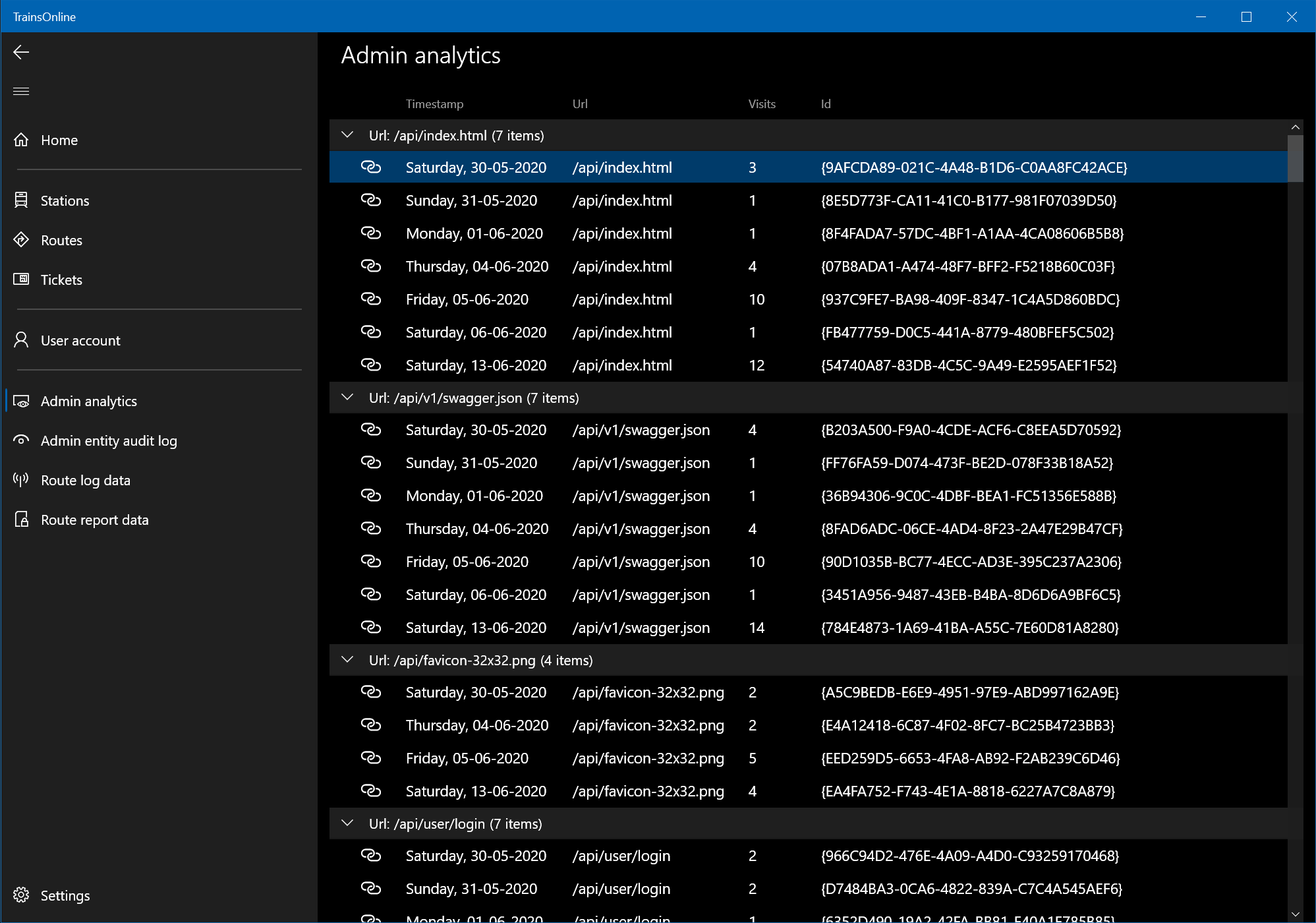
Posiada ona obsługę funkcjonalności API takich jak:

* logowanie i rejestracja;

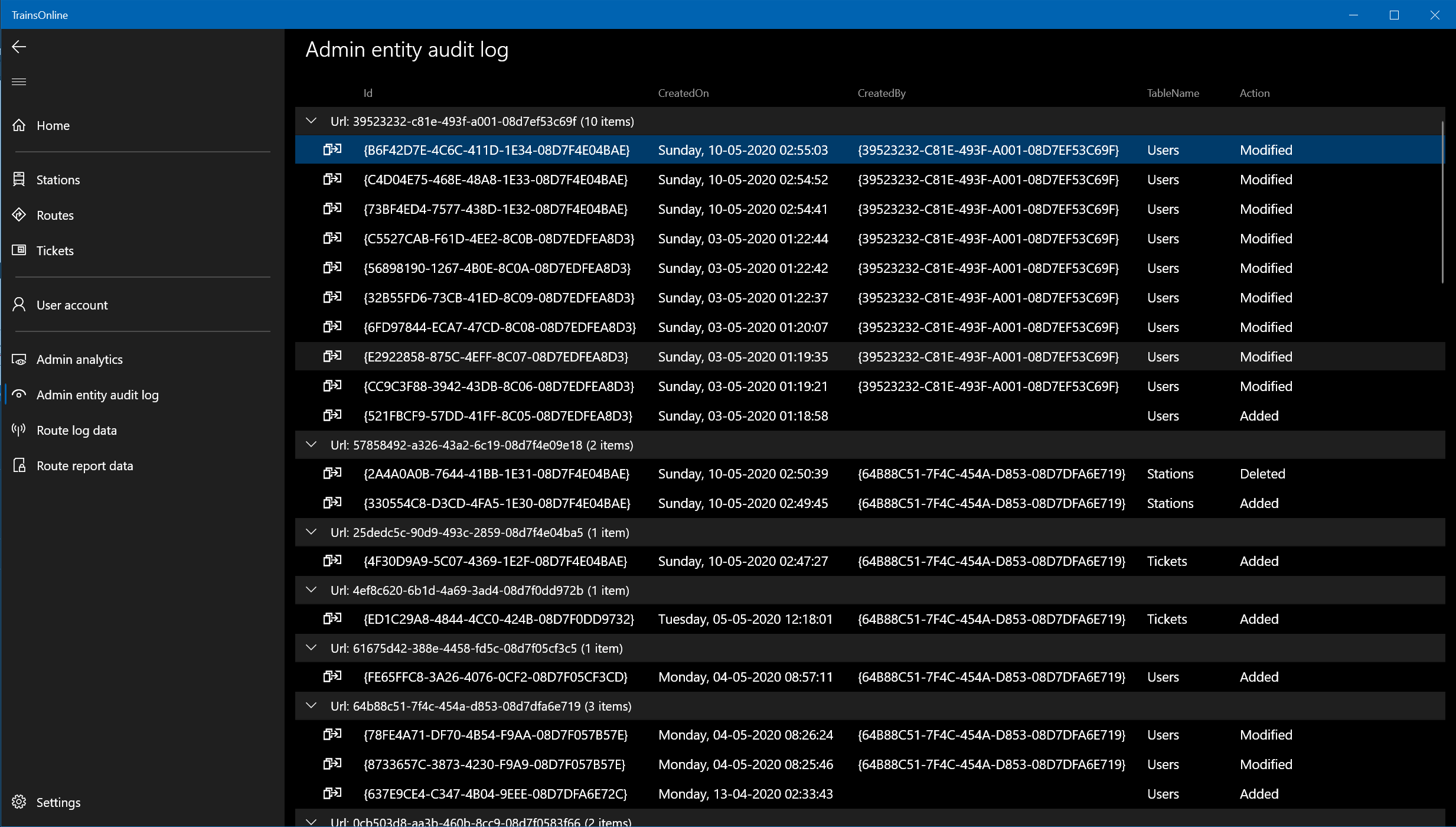




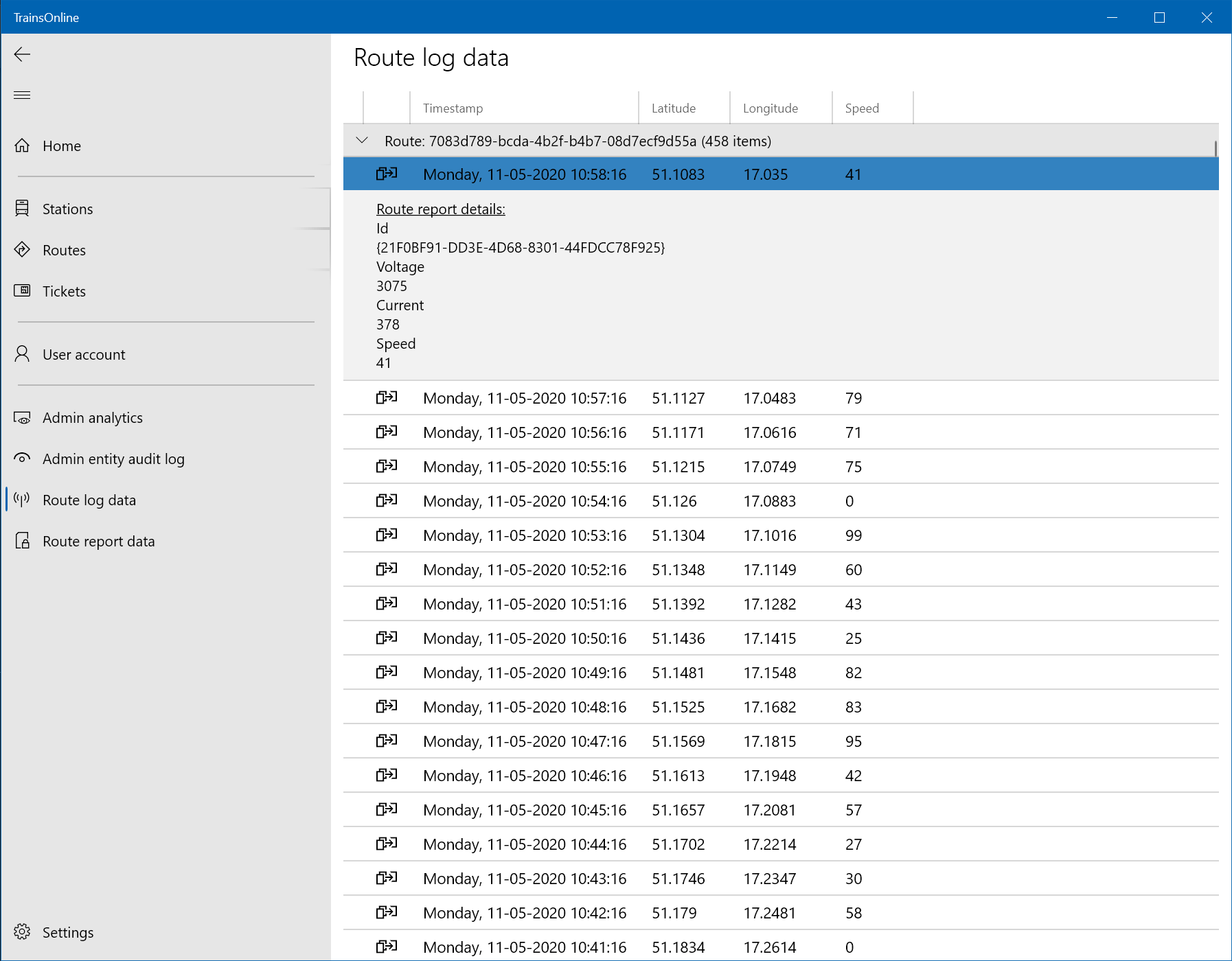
* analityka



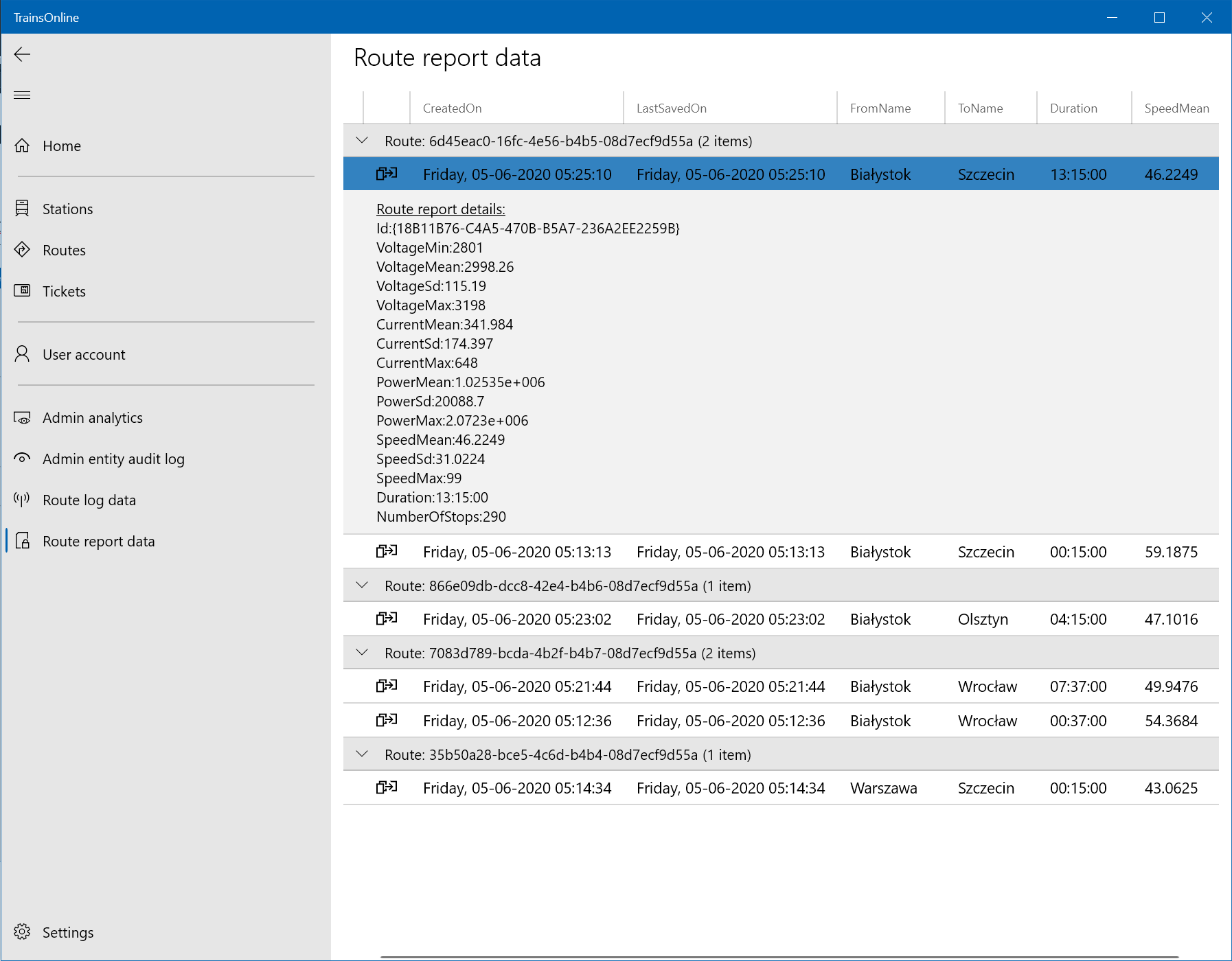
* logi zmian w bazie relacyjnej



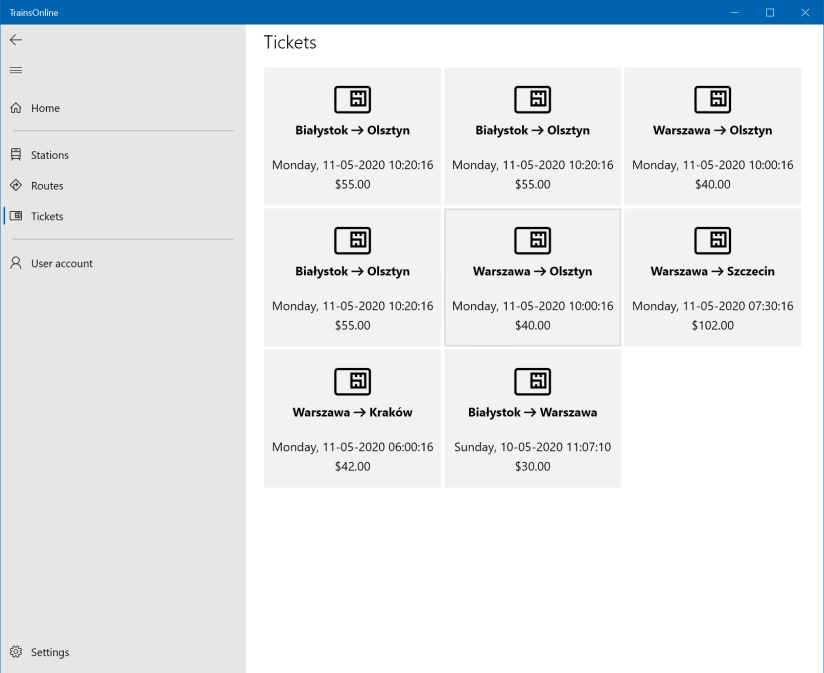
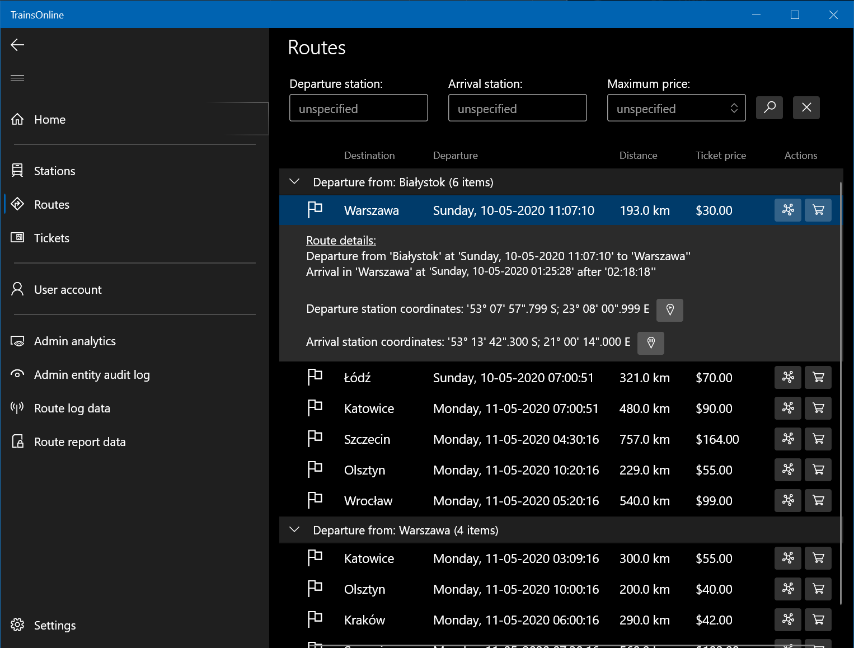
* logi z tras:



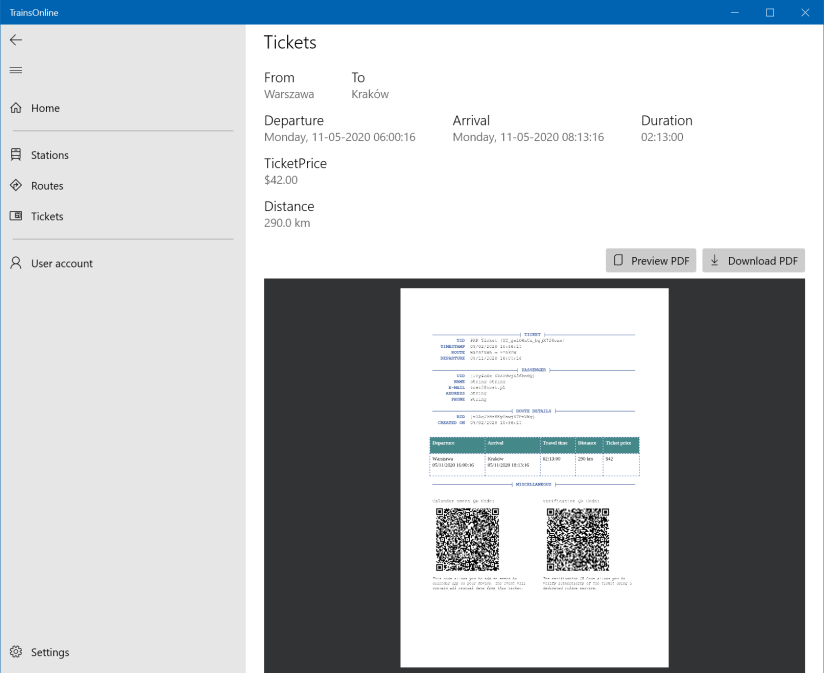
* raporty z tras

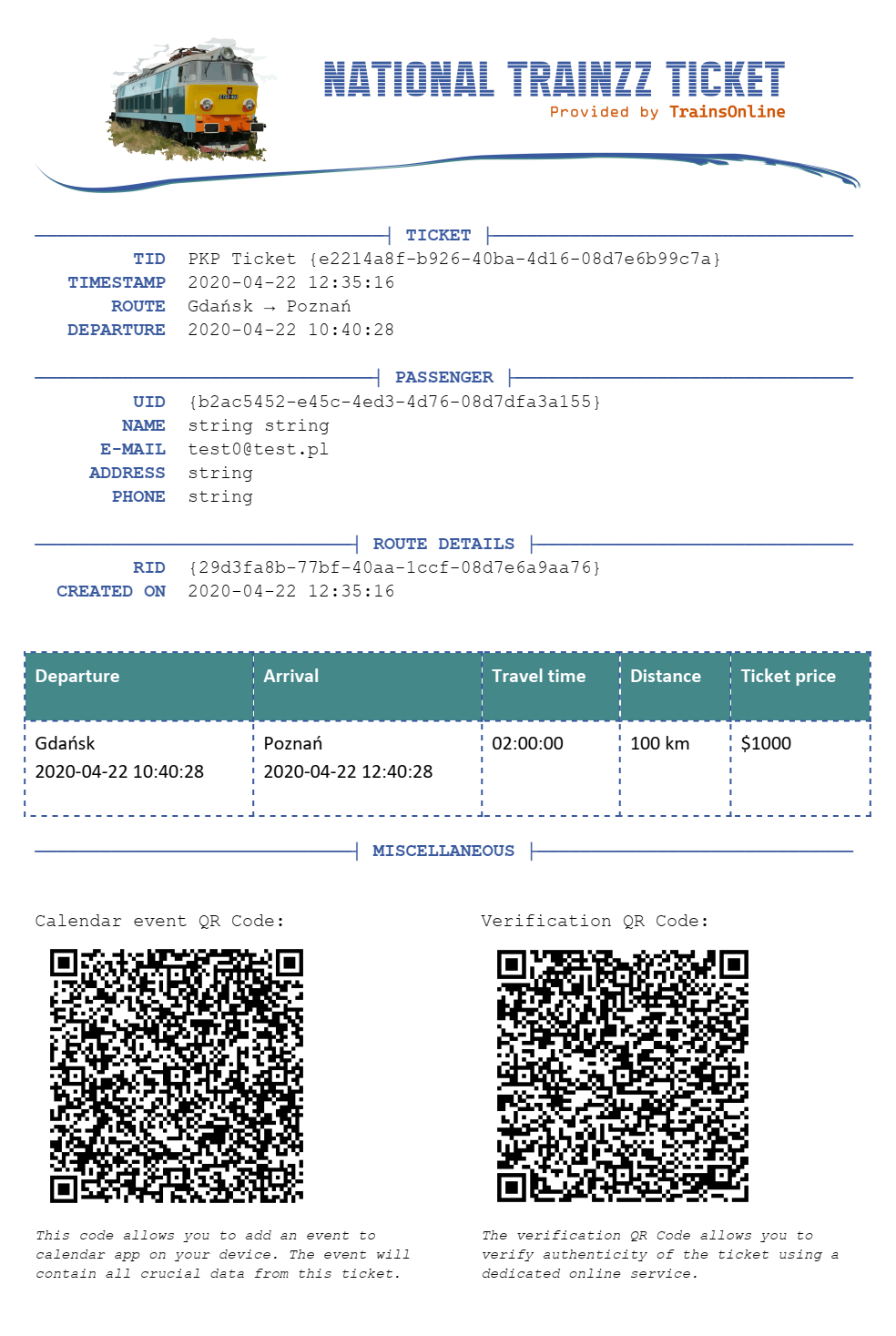


* możliwość kupna biletów;



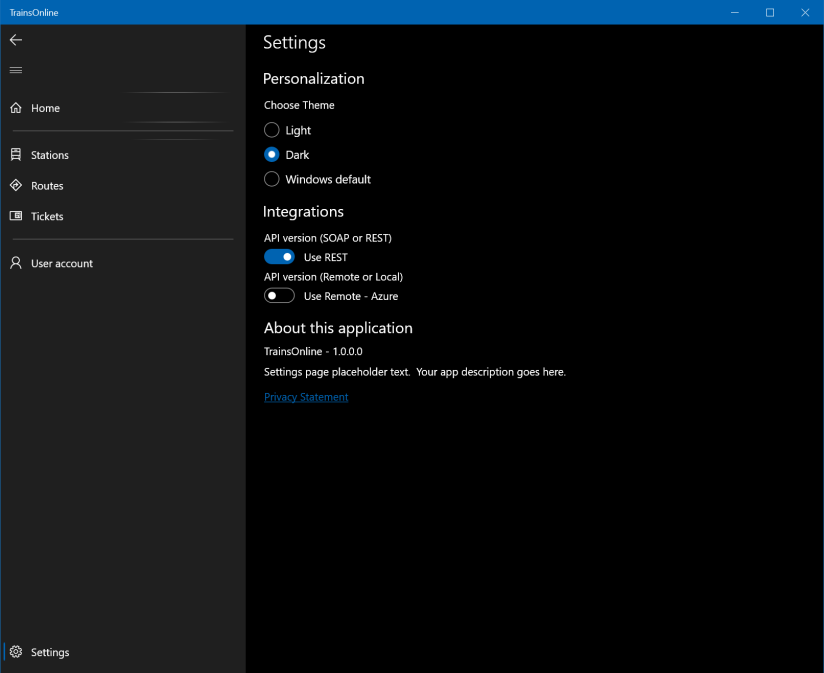
* pobranie biletu w formacie PDF;



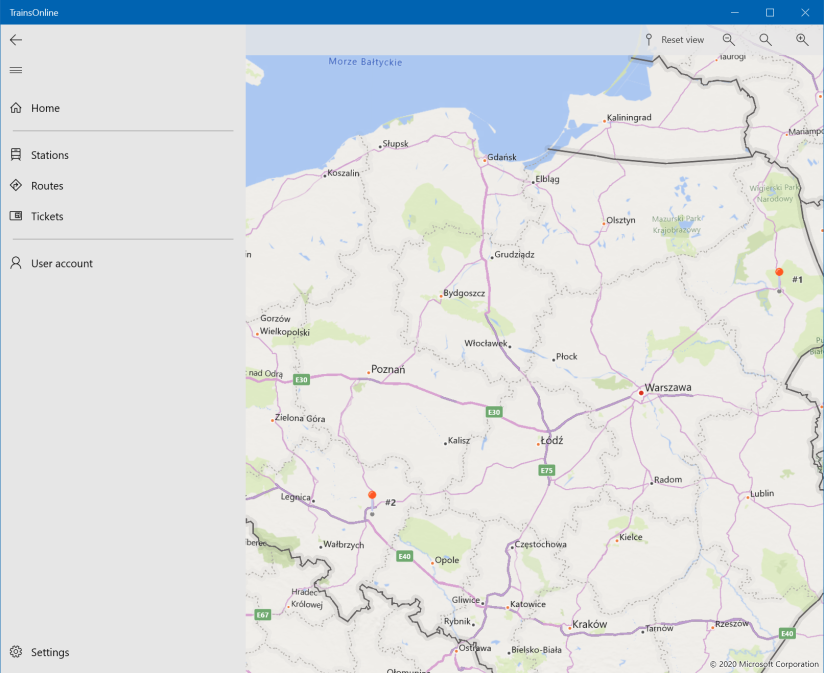


Poza spełnieniem podstawnych założeń aplikacja posiada dodatkowe funkcjonalności taki jak:

* możliwość wyboru stylu aplikacji( ciemny lub jasny)



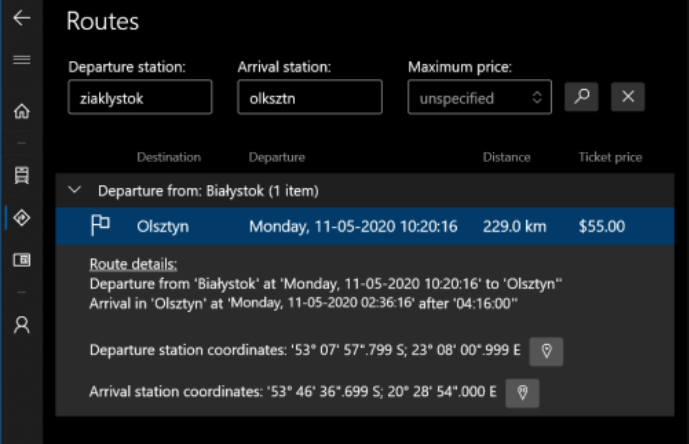
* możliwość podejrzenia lokalizacji stacji na mapie;



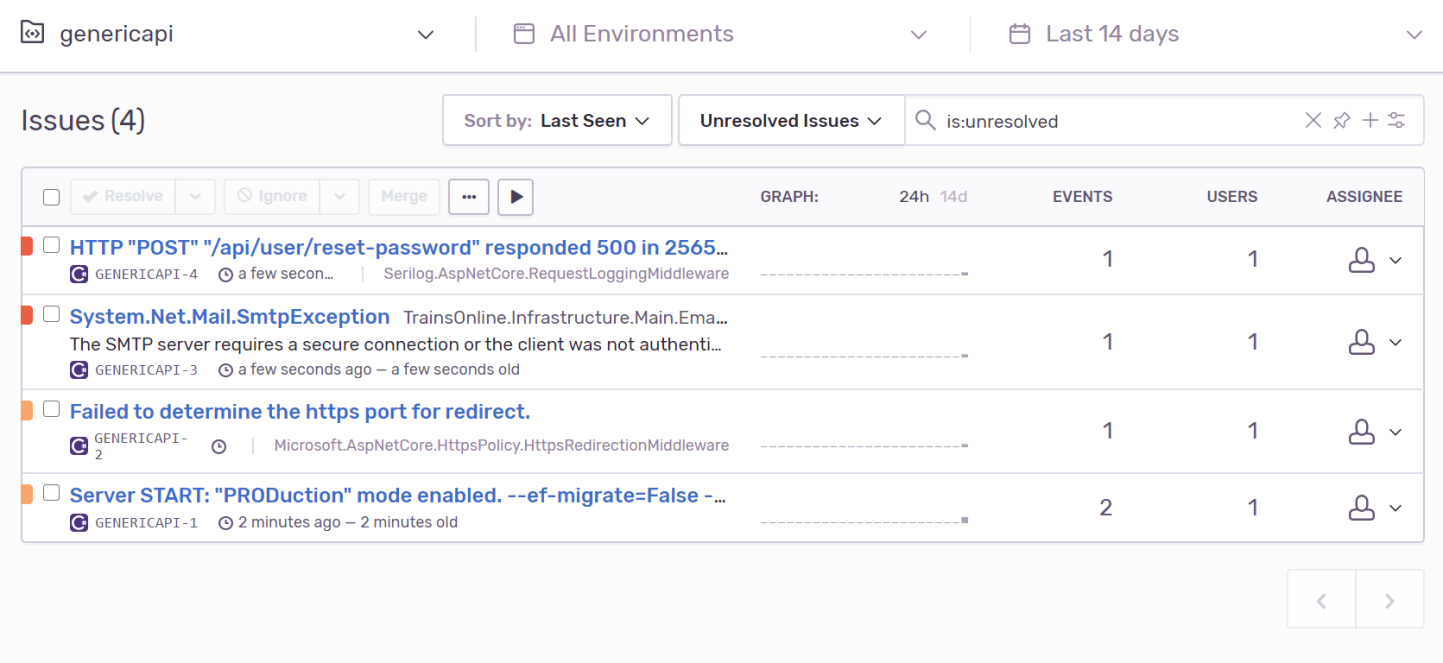
* generowanie kodów QR

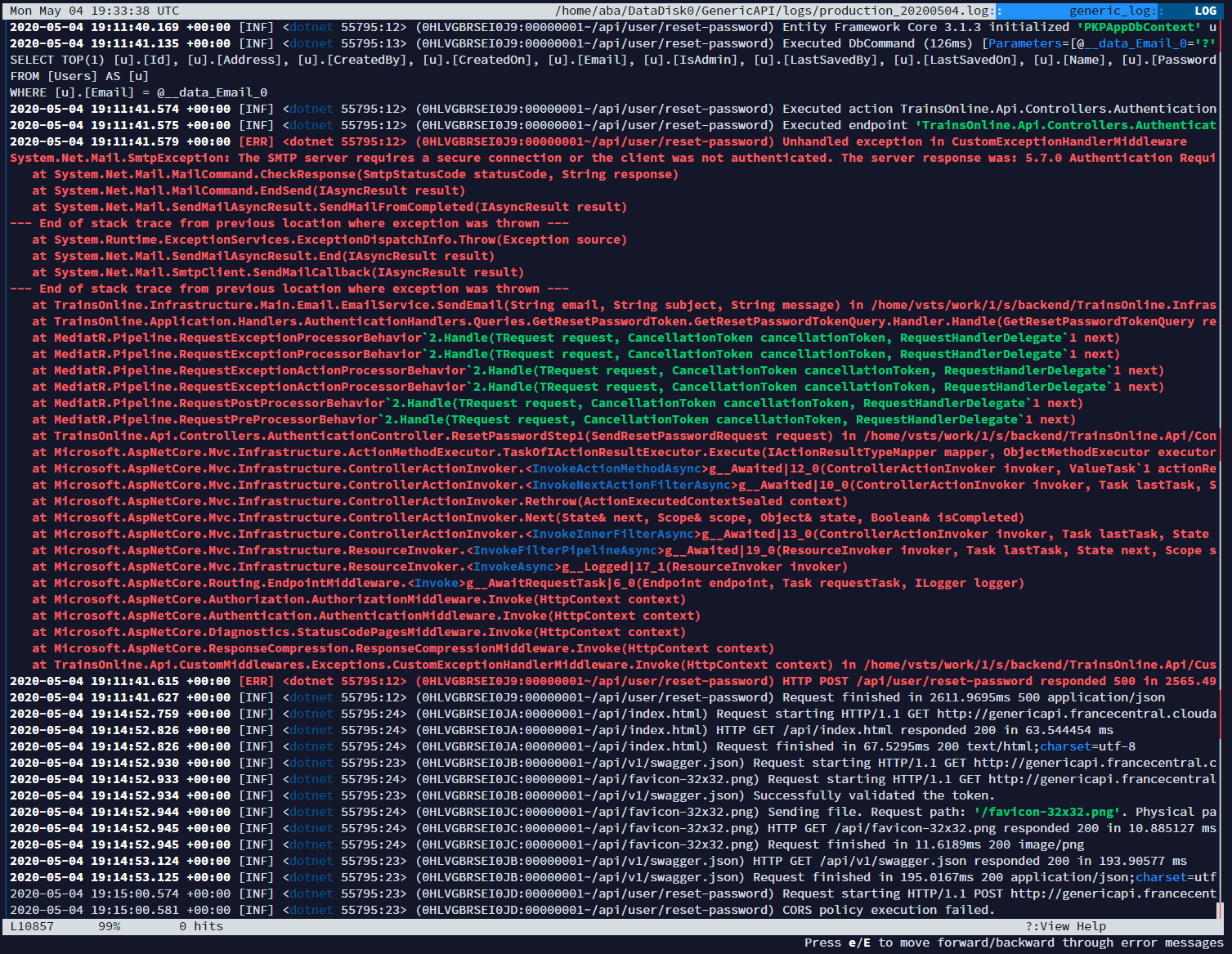


* wykorzystnia algorytmu Jaro-Winkler

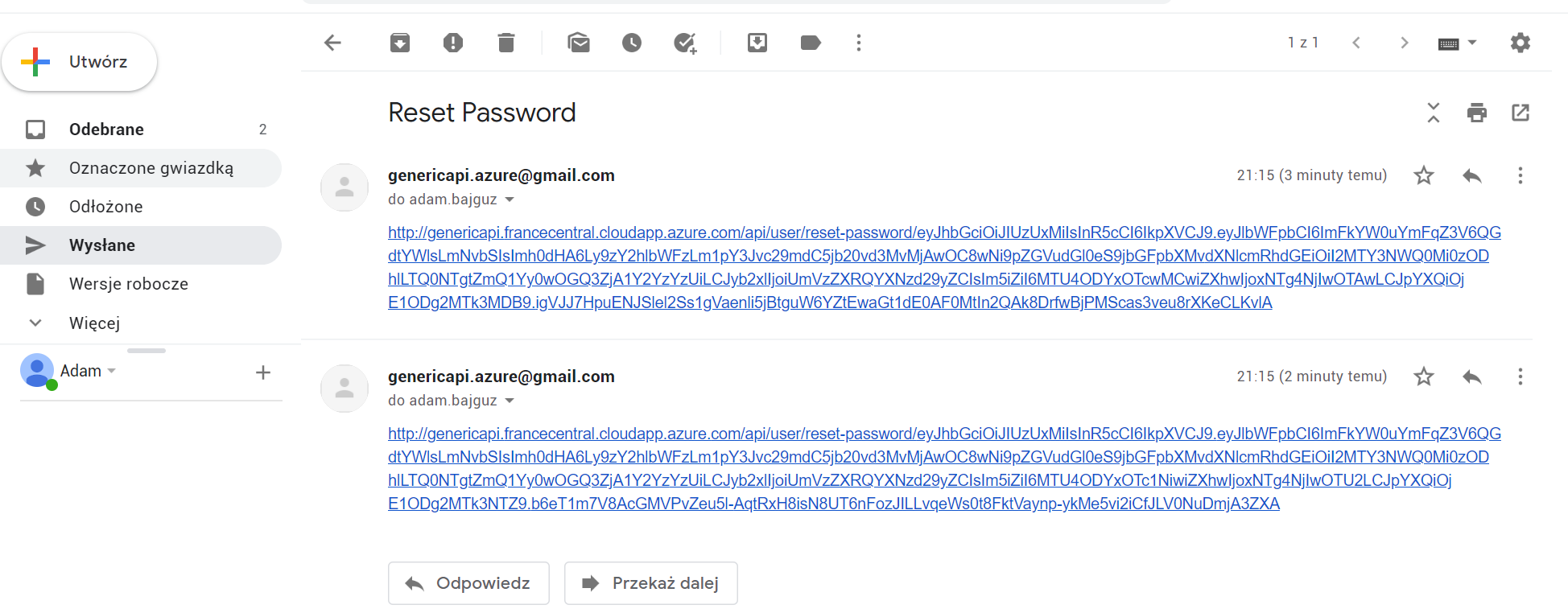
****

* dziennik logów zdarzeń na serwerze





* wysyłanie emaili



## 2.3. Profil użytkownika końcowego

Użytkownikiem końcowym aplikacji mogą być przedsiębiorstwa zajmujące się transportem kolejowym i przewozem osób. Aplikacja może posłużyć do zbierania danych z tras, celem ich późniejszej analizy, a także tworzenia dokumentacji końcowej dla przewoźników i podsumowywania działalności firmy w danych okrasach czasowych.

Dane mogą posłużyć także do optymalizacji tras oraz analizy sytuacji specjalnych np. zmiany napięcia w trakcji kolejowej i analizy przyczyny uszkodzeń taboru. W przypadku właściwego wdrożenia na pociągach była by również używana przez maszynistów to kontroli sposobu prowadzenia pociągu, a także mogła by posłużyć jako system ostrzegania np. przed kolizja pociągów.