

Implementacija Api gateway-a u .Net-u

FUNKCIJSKA SPECIFIKACIJA

V1.0

Povijest revizija

Datum revizije	Verzija	Što je napravljeno	Tko
19.05.2021.	0.1	Inicijalna verzija	Antonio Supan
27.05.2021	0.2	Logiranje, detaljni opis projekata u arhitekturi	Antonio Supan
16.06.2021	1.0	Arhitektura dotjerana, flowchart	Antonio Supan

Sadržaj

[Uvod](#)

[Projektna arhitektura](#)

[Grafikon slijeda](#)

[Napomene](#)

- [Dobro je znati](#)
- [Bitno](#)
- [Prednosti](#)
- [Nedostatci](#)

[Opis funkcionalnosti](#)

- [Konfiguracija](#)
- [Rutiranje](#)
- [Logiranje](#)
- [Autentikacija pristupnika i servisa](#)
- [Klijentska autentikacija](#)
- [Bijela lista IP adresa](#)
- [Rate limiting](#)
- [Otkrivanje servisa](#)

Uvod

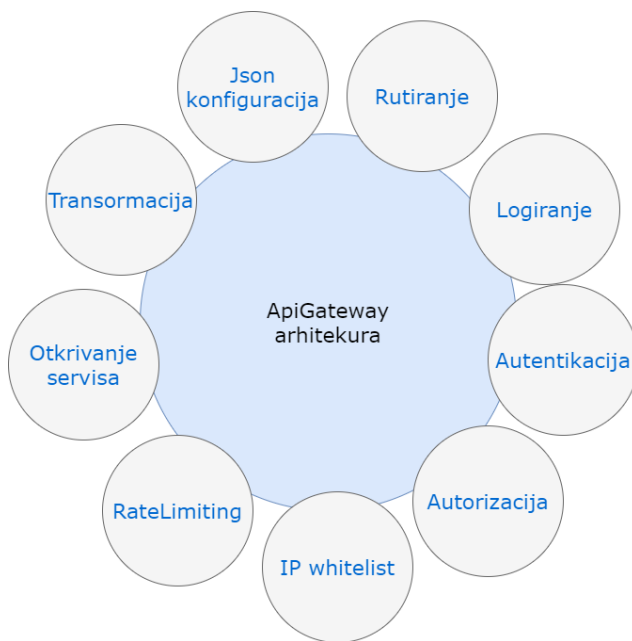
API gateway tj. pristupnik uzima sve API pozive od klijenata, a zatim ih usmjerava na odgovarajuću mikroservisu arhitekturu s funkcijama usmjeravanja zahtjeva, transformiranja i prevođenja protokola.

Obično obrađuje zahtjev pozivanjem više mikroservisa i agregiranjem rezultata kako bi odredio najbolji put do rezultata za klijentsku stranu. Može poslužiti u svim web protokolima, koji se koriste javno ili interno.

Ovaj projekt nije projekt koji ovisi sam o sebi, nego treba svog host-a, najčešće web api aplikaciju koja će koristiti funkcionalnosti iz ovog projekta, po tipu projekta ovo je dinamička biblioteka klasa, koju nakon što pretvorimo u Nuget paket, vrlo lako možemo koristiti u svim .NET projektima.

.NET tehnologija bit će korištena za izradu ovog projekta, a glavne su mu značajke :

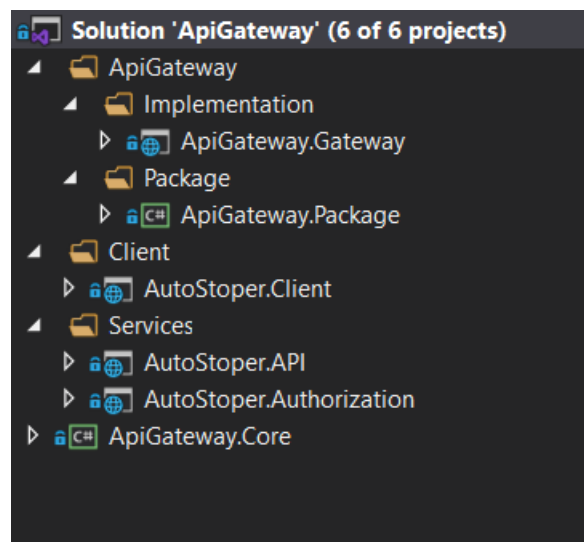
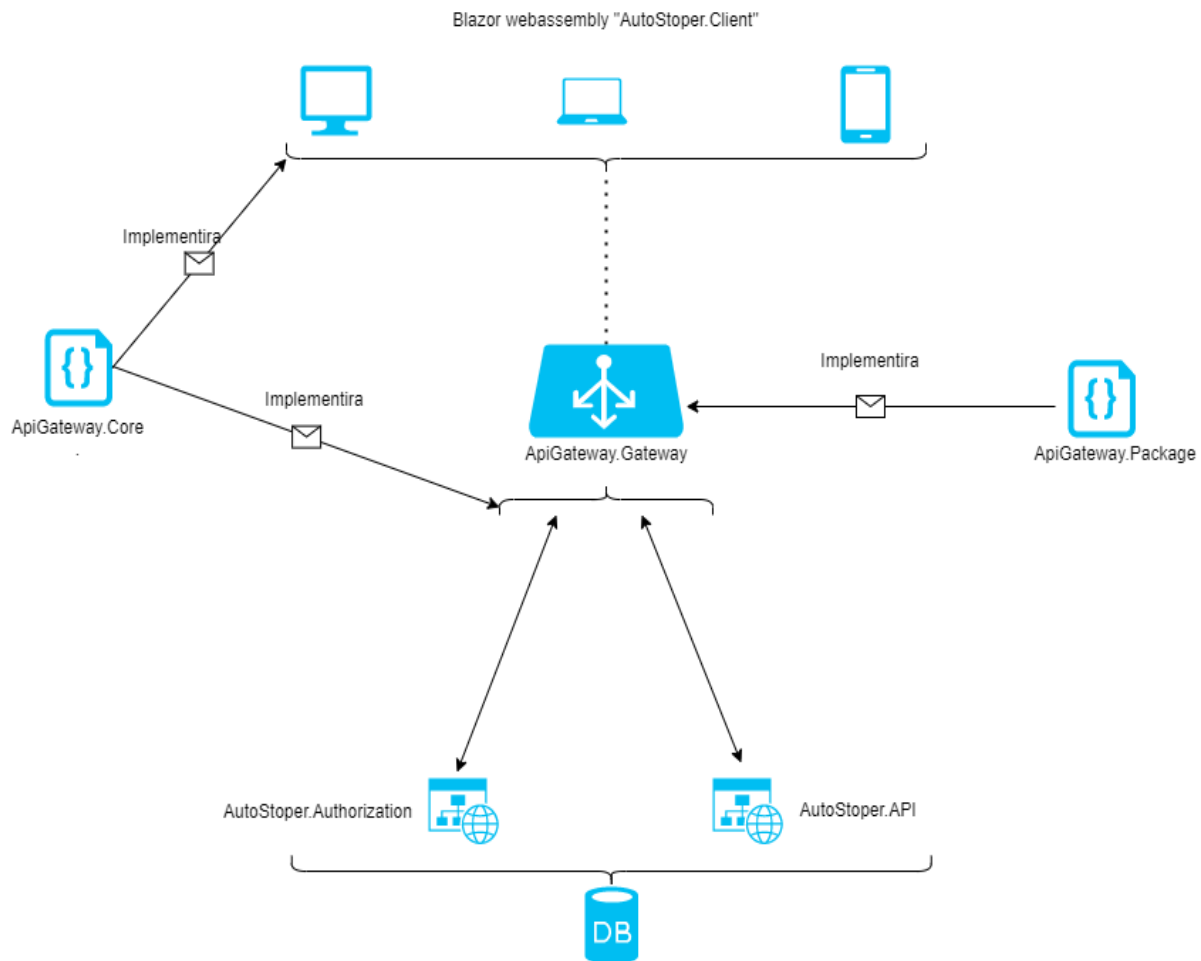
- Rutiranje web zahtjeva
- Logiranje određenih zapisa
- Autentikacija između pristupnika te mikroservisa
- Klijentska autentikacija
- Bijela lista IP adresa koje šalju web zahtjev
- Rate limiting – ograničavanje pozivanja određenog servisa sa neke IP adrese
- Otkrivanje odgovarajućih mikroservisa putem ID-a
- Transformacija web zahtjeva
- Json konfiguracija Api pristupnika



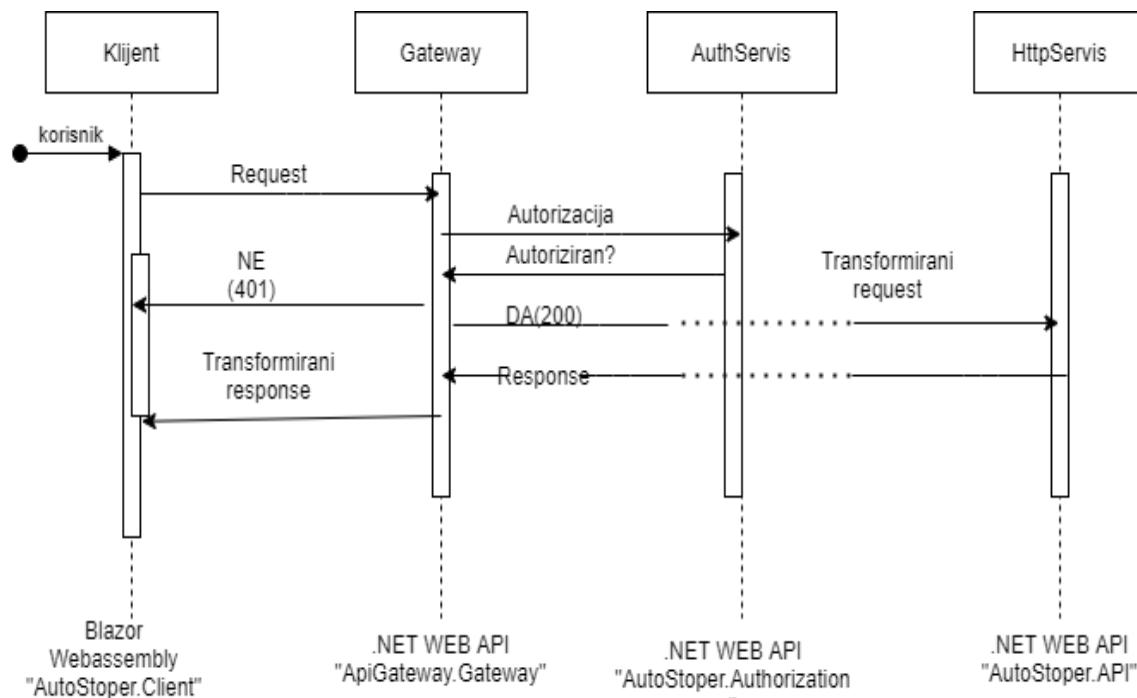
Projektna arhitektura

- Za potrebe pokazivanja rada funkcionalnosti iz .NET api gateway nuget paketa

u pozadini ću koristiti Blazor webassembly PWA aplikaciju na klijentu, te na backend-u autentikacijski servis, AutoStoper api za dohvat podataka te ApiGateway api za prikaz funkcionalnosti ApiGateway nuget paket



GRAFIKON SLIJEDA



OPIS ARHITEKTURE

AutoStoper.Client (Blazor webassembly)

Blazor WebAssembly nova je Microsoftova tehnologija (programski okvir) za izradu korisničkog sučelja, službeno objavljena s .NET Core verzijom 3.1 i primajući ažuriranja u .NET 5 verziji. Blazor omogućuje programerima stvaranje aplikacija na jednoj ili više stranica pomoću C#-a i .NET-a koristeći arhitekturu zasnovanu na komponentama. Blazor WebAssembly je klijentska implementacija Blazora u pregledniku koja uključuje .NET runtime implementiran kroz WebAssembly. AutoStoper.Client naziv je za klijentsku stranu projekta AutoStoper, a koja koristi gore navedenu tehnologiju.

ApiGateway.Package (Class Library - .NET 5.0)

Biblioteka klasa (DLL). DLL sadrži programski kod, podatke i resurse koje drugi programi mogu koristiti i lako ih se može implementirati u druge C# projekte. **ApiGateway.Package naziv je za biblioteku klasa koja sadrži cijeli skup funkcionalnosti za funkcioniranje ApiGateway.Gateway projekta uz pozivanje samo jedne linije koda i projekt je od najveće važnosti u cijeloj arhitekturi.** ApiGateway.Package u sebi ima funkcionalnosti ekstenzijskih metoda, rutiranja i transformiranja web zahtjeva, logiranja u datoteku, blokiranja IP adresa, rate limiting-a, otkrivanja servisa itd.

ApiGateway.Core (Class Library - .NET 5.0)

Biblioteka klasa (DLL). ApiGateway.Core naziv je za biblioteku klasa koja sadrži zajedničke modele i razne pomoćne klase te servise koji se mogu koristiti u bilo kojem drugom projektu u ovoj arhitekturi.

ApiGateway.Gateway (ASP.NET Web API)

ASP.NET Web API okvir je za izgradnju HTTP servisa kojima se može pristupiti s bilo kojeg klijenta, uključujući preglednike i mobilne uređaje. To je idealna platforma za izgradnju RESTful aplikacija na .NET programskom okviru. ApiGateway.Gateway naziv je za HTTP servis implementiran kroz programski okvir ASP.NET-a kako bi web zahtjevom klijenta uz pomoć ApiGateway.Package-a dohvaćao pravovaljane podatke iz ostalih HTTP servisa, obrađivao ih, provjeravao, te na kraju vratio klijentu.

AutoStoper.API (ASP.NET Web API)

AutoStoper.API naziv je za HTTP servis implementiran kroz programski okvir ASP.NET-a kako bi web zahtjevom klijenta dohvaćao podatke iz baze podataka, obradio ih, te ih vratio klijentu.

AutoStoper.Authorization (Autentikacijski servis, ASP.NET Web API)

AutoStoper.Authorization naziv je za HTTP servis implementiran kroz programski okvir ASP.NET-a kako bi bilo koji endpoint sa ostalih HTTP servisa, a za koje je potrebna autorizacija, zaštitio i provjerio ispravnost podataka koje šalje klijent kako bi poništio ili odobrio daljnju akciju.

NAPOMENE

Dobro je znati

- Od projektne funkcionalnosti najviše koristi mogu imati razvojni inženjeri koji planiraju koristiti mikroservisnu arhitekturu za izradu projekta
- Glavni projekt je ApiGateway.Package – svi ostali projekti te arhitektura ovdje su samo zbog prikaza rada glavnog projekta
- Projekt se nalazi na web lokaciji <https://github.com/ASxCRO/Api-Gateway-.NET> te se od tamo može povući izvorni kod te je moguće pregledati aktivnosti na projektu

Bitno

- Pretpostavka je da se koristi JWT autentifikacija te autorizacija za klijenta na serverskoj strani
- Api Gateway te web servisi međusobno se štite promjenjivim kriptiranim api ključem koji ima svoj privatni ključ koji samo api gateway i web servisi znaju
- Ovu arhitekturu dobro je koristiti tek onda kad imamo dobro složen plan te više mikroservisa kako bi klijentu olakšali pozive, a kako nebi klijentima do znanja dali adrese pravih web servisa
- Prije odluke sve stvari treba uzeti u obzir

Prednosti

- Mikroservisi mogu se u potpunosti orijentirati na poslovnu logiku
- Klijentu je posao olakšan pošto uvijek poziva samo jedan servis – api gateway
- Sigurnost pojačana jer HTTP servisi nisu izloženi javnosti
- Api gateway upravlja autentikacijom, logiranjem te rutiranjem
- Daje punu fleksibilnost pri blokiranju IP adresa, ograničavanju broja poziva itd.

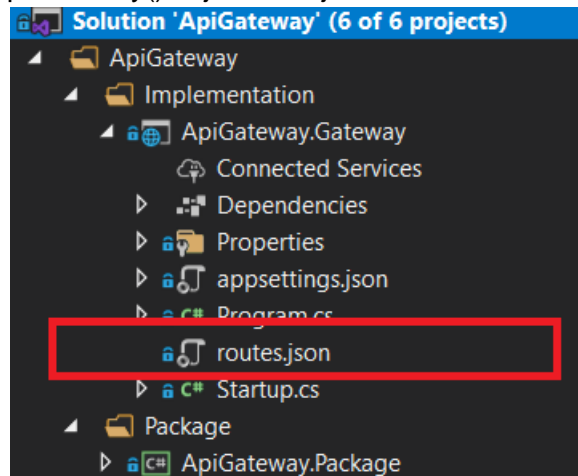
Nedostatci

- Jedinstvena točka pada sustava
- Povećana kompleksnost sustava
- Žrtvovanje performansi te latencije zbog rutiranja poziva

OPIS FUNKCIONALNOSTI

Konfiguracija

- Naš nuget paket koristit će JSON konfiguracija koju ćemo napraviti na host-u te se obvezno mora zvati “routes.json” – host u svoj http request pipeline mora uglaviti metodu app.UseApiGateway() koja dobavlja funkcionalnost iz Nuget paketa



- routes.json

```
{
  "routes": [
    {
      "endpoint": "/weather",
      "destination": {
        "serviceId": 1,
        "uri": "WeatherForecast/Get",
        "requiresAuthentication": "true"
      }
    }
  ],
  "services": [
    {
      "id": 0,
      "name": "authenticationService",
      "baseUri": "https://localhost:44309"
    },
    {
      "id": 1,
      "name": "autostoperService",
      "baseUri": "https://localhost:44386",
      "ipSafelist": [
        "0.0.0.1"
      ]
    }
  ],
  "ratelimiting": [
    {
      "ipAddress": "0.0.0.1",
      "serviceId": 1,
      "ratePerDay": 10
    }
  ]
}
```


- Model za učitavanje konfiguracije iz nuget paketa

```
public class Router
{
    private readonly IConfiguration configuration;

    2 references
    public List<Route> Routes { get; set; }
    3 references
    public List<Service> Services { get; set; }
    0 references
    public List<RateLimit> RateLimits { get; set; }
    2 references
    private RateLimitingCache _rateLimitingCache { get; }

    1 reference
    public Router(string routeConfigFilePath, RateLimitingCache rateLimitingCache, IConfiguration configuration)
    {
        dynamic router = JsonLoader.LoadFromFile<dynamic>(routeConfigFilePath);
        Routes = JsonSerializer.Deserialize<List<Route>>(Convert.ToString(router.routes));
        Services = JsonSerializer.Deserialize<List<Service>>(Convert.ToString(router.services));
        _rateLimitingCache = rateLimitingCache;
        this.configuration = configuration;
    }
}
```

Rutiranje

- Odvija se na način da klijent pozove endpoint na api gateway-u koji obrađuje zahtjev te ga šalje odgovarajućem servisu dalje na obrađivanje uz provjeru valjanosti zahtjeva putem glavne metode Router klase – RouteRequest()

Logiranje

- Plan je bio ugraditi klasični logger u api gateway u smislu – koji korisnik je pokušao u koje vrijeme sa kojeg uređaja i koje IP adrese pristupiti kojem servisu te kojim njegovim podacima i sve to zajedno zapisivati u datoteku
- Odvija se putem Serilog nuget paketa – a u kodu se poziva linija “_logger.Log(poruka);” i svaki request i njegove osobine zapišu se u datoteku logs.txt

Autentikacija pristupnika te servisa

- Do servisa može doći samo api gateway, stoga ograničenje leži u potrebi za blokiranjem prometa sa bilo kojeg uređaja osim api gateway-a, rješenje je u promjenjivom kriptiranom apikey-u koji svoju privatnu šifru djeli samo sa tim servisom pa se tako može provjeriti ispravnost hasheva koji se šalje kroz transformirana zaglavlja web zahtjeva, koristi se SHA256 algoritam za dobivanje tzv. “salted hash”
- U http mikroservisima u Startup.cs datoteci te u Configure metodi potrebno je dodati liniju “ app.UseApiKeyValidation();” kako bi mogli koristiti rečene funkcionalnosti, u appsettings.json (isto na api gateway host-u) mora se dodati čvor -

```
"ApiKeyOptions": {
    "Secret": "ProizvoljniPrivatniKljuc"
}
```

Autentikacija klijenta

- U routes.json datoteci, na svaki čvor u polju "routes", možemo dodati property "requiresAuthentication", te mu možemo dati vrijednosti true/false.
- Ako mu dodamo vrijednost true, paket će potražiti servis pod nazivom "authenticationService" te će ga pitati da li taj korisnik koji ima taj jwt token, stvarno ima validan token i prava na određeni servis
- Ako vrijednost bude false, korisnička prava se neće provjeravati tj. To je tzv. allowanonymous endpoint
- Na našem autentikacijskom servisu sve što moramo napraviti je koristiti ekstenzijsku metodu iz paketa - "app.UseApiGatewayAuthorization();"

Bijela lista IP adresa

- Kroz konfiguraciju servisa u routes.json možemo ograničiti sa koje IP adrese se uopće može doći do tog servisa

Rate limiting

- Kroz konfiguraciju servisa u routes.json možemo ograničiti broj poziva koji se mogu izvršavati sa određene IP adrese prema određenom servisu, broj poziva određuje se prema danu.

Otkrivanje servisa

- Centralizirano mjesto servisa pronalazimo u routes.json datoteci te iste servise višestruko možemo koristiti u čvoru "routes" kako nebi svaki put razmišljati koji je bazna http adresa do našeg željenog servisa