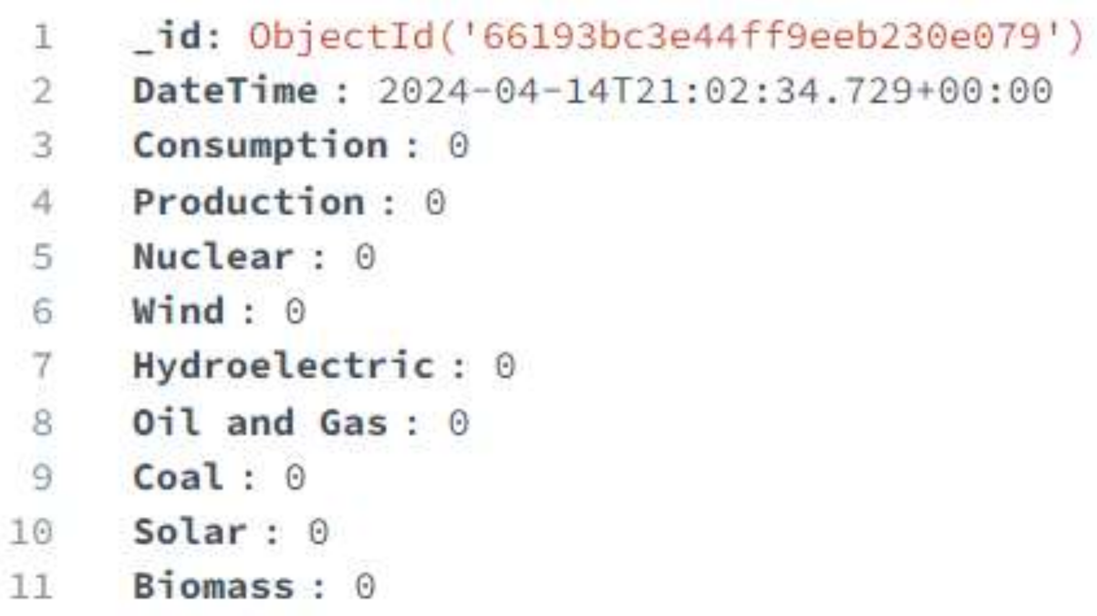
Elektronski fakultet
Univerzitet u Nišu

Autor: Nikola Nikolić
Broj indeksa: 18308

Podaci

Podaci su smešteni u lokalnoj MongoDB bazi. Jedan document sadrži id, datum upisa, kao I podatke o proizvodnji I ukupnoj potrošnji električne energije tog dana. Ukupna proizvodnja se dobija zbirom svih tipova energije proizvedenih tog dana (nuklearna, energija vetra, solarna, hidroelektrična, biomasa I energija ulja I uglja). Id je tipa ObjectId, datum upisa Date, sve ostalo Int32.

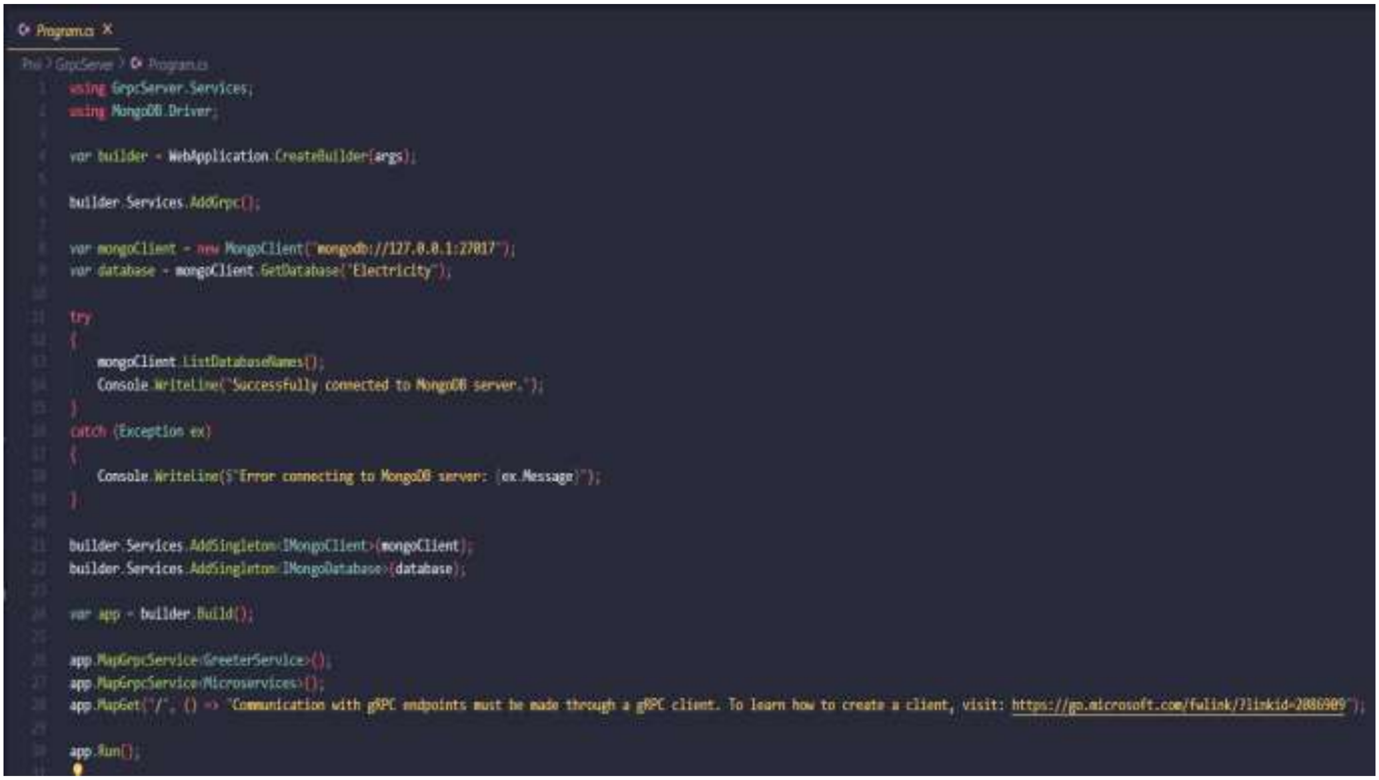


```
_id: ObjectId('66193bc3e44ff9eeb230e079')
DateTime : 2024-04-14T21:02:34.729+00:00
Consumption : 0
Production : 0
Nuclear : 0
Wind : 0
Hydroelectric : 0
Oil and Gas : 0
Coal : 0
Solar : 0
Biomass : 0
```

Prvi

Prvi projekat se sastoji od 2 mikroservisa, prvi je .NET gRPC, dok je drugi NodeJs aplikacija.

Prvi mikroservis služi kao server i ima zadatak da se poveže sa bazom I izvrši CRUD operacije i da obezbedi funkcije agregacije (MIN, MAX, AVG, SUM) nad svim podacima.

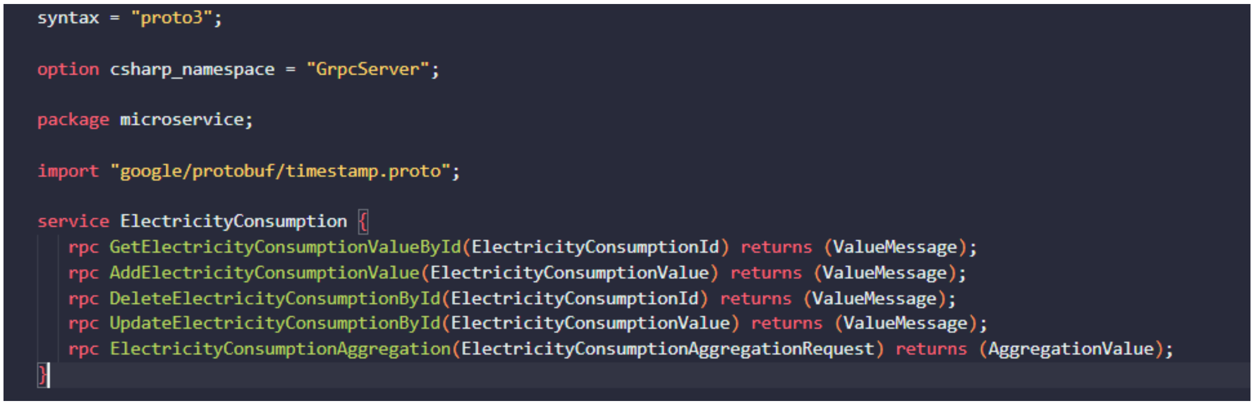


```
using GrpcServer.Services;
using MongoDB.Driver;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddGrpc();

var mongoClient = new MongoClient("mongodb://127.0.0.1:27017");
var database = mongoClient.GetDatabase("Electricity");

try
{
    mongoClient.ListDatabaseNames();
    Console.WriteLine("Successfully connected to MongoDB server.");
}
catch (Exception ex)
{
    Console.WriteLine($"Error connecting to MongoDB server: {ex.Message}");
}

builder.Services.AddSingleton<IMongoClient>(mongoClient);
builder.Services.AddSingleton<IMongoDatabase>(database);

var app = builder.Build();

app.MapGrpcService<GreeterService>();
app.MapGrpcService<Microservices>();
app.MapGet("/", () => "Communication with gRPC endpoints must be made through a gRPC client. To learn how to create a client, visit: https://go.microsoft.com/fwlink/?linkid=2086909");

app.Run();
```

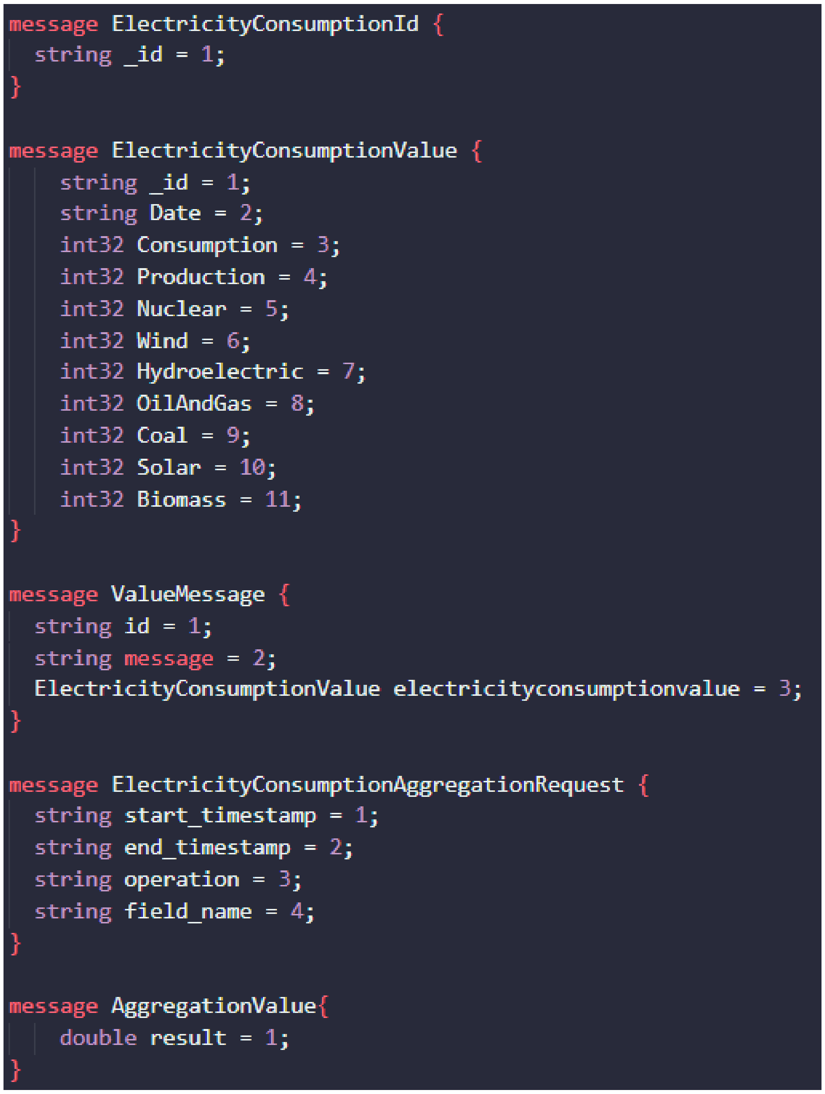
Povezivanje sa bazom se vrši u Program.cs, neophodno je instalirati MongoDB.Driver. U Program.cs se takođe mapiraju gRPC servisi.

Koristi se .proto fajl za definisanje poruka koje se koriste u servisu, i stavlja se csharp_namespace naredba da bi se naznačilo u koji jezik se proto prevodi.

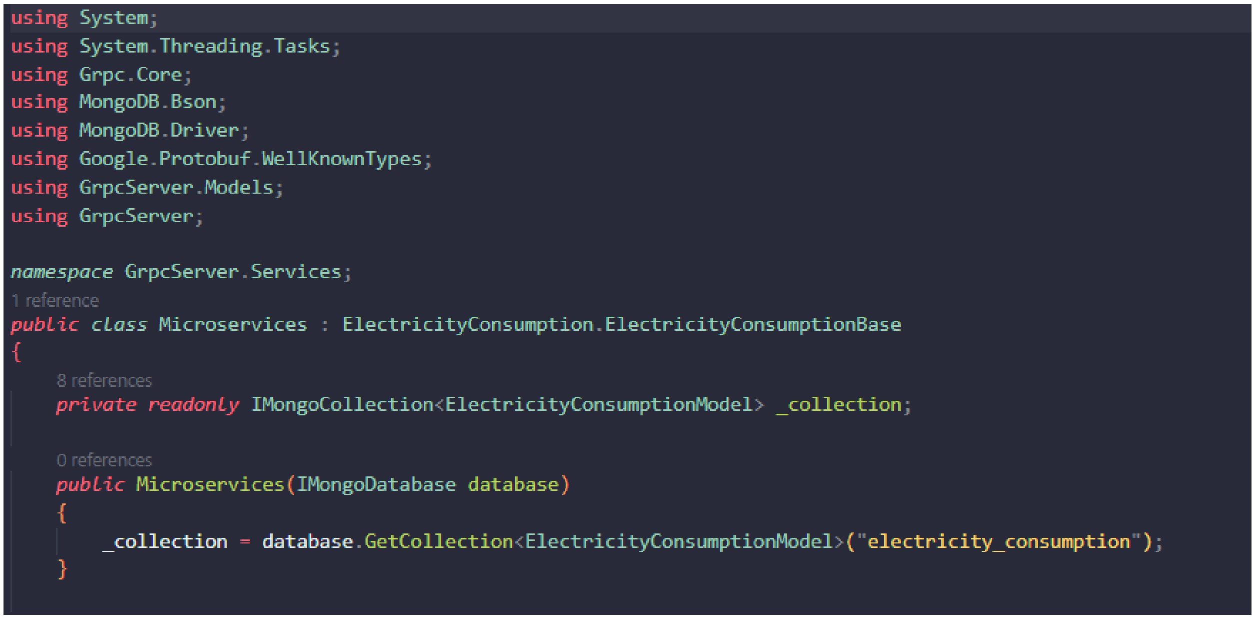
Zatim se pravi cs fajl koji predstavlja mikroservis, on nasledjuje servis iz proto fajla i implementira CRUD metode i funkciju agregacije.



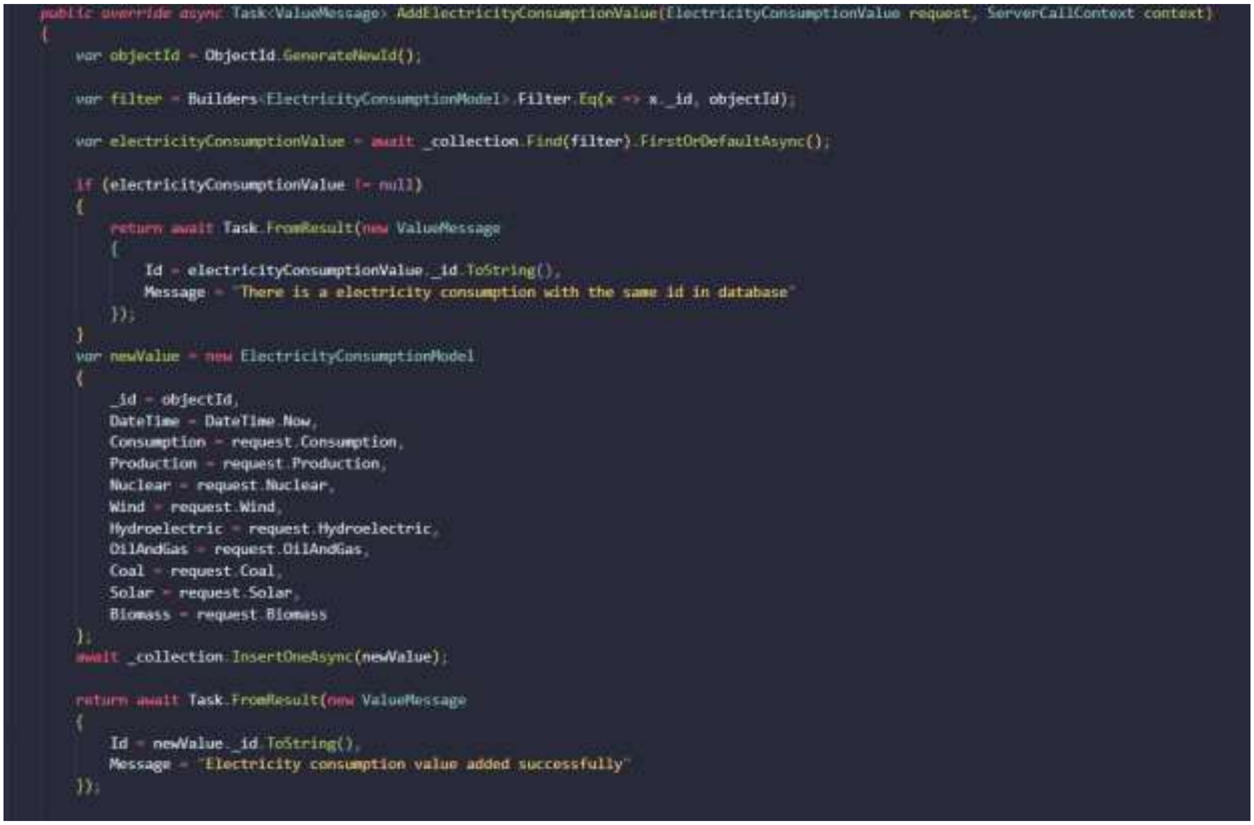
```
syntax = "proto3";

option csharp_namespace = "GrpcServer";

package microservice;

import "google/protobuf/timestamp.proto";

service ElectricityConsumption {
    rpc GetElectricityConsumptionValueById(ElectricityConsumptionId) returns (ValueMessage);
    rpc AddElectricityConsumptionValue(ElectricityConsumptionValue) returns (ValueMessage);
    rpc DeleteElectricityConsumptionById(ElectricityConsumptionId) returns (ValueMessage);
    rpc UpdateElectricityConsumptionById(ElectricityConsumptionValue) returns (ValueMessage);
    rpc ElectricityConsumptionAggregation(ElectricityConsumptionAggregationRequest) returns (AggregationValue);
}
```



```
message ElectricityConsumptionId {
  string _id = 1;
}

message ElectricityConsumptionValue {
    string _id = 1;
    string Date = 2;
    int32 Consumption = 3;
    int32 Production = 4;
    int32 Nuclear = 5;
    int32 Wind = 6;
    int32 Hydroelectric = 7;
    int32 OilAndGas = 8;
    int32 Coal = 9;
    int32 Solar = 10;
    int32 Biomass = 11;
}

message ValueMessage {
  string id = 1;
  string message = 2;
  ElectricityConsumptionValue electricityconsumptionvalue = 3;
}

message ElectricityConsumptionAggregationRequest {
  string start_timestamp = 1;
  string end_timestamp = 2;
  string operation = 3;
  string field_name = 4;
}

message AggregationValue{
    double result = 1;
}
```



```
using System;
using System.Threading.Tasks;
using Grpc.Core;
using MongoDB.Bson;
using MongoDB.Driver;
using Google.Protobuf.WellKnownTypes;
using GrpcServer.Models;
using GrpcServer;

namespace GrpcServer.Services;
1 reference
public class Microservices : ElectricityConsumption.ElectricityConsumptionBase
{
    8 references
    private readonly IMongoCollection<ElectricityConsumptionModel> _collection;

    0 references
    public Microservices(IMongoDatabase database)
    {
        _collection = database.GetCollection<ElectricityConsumptionModel>("electricity_consumption");
    }
```

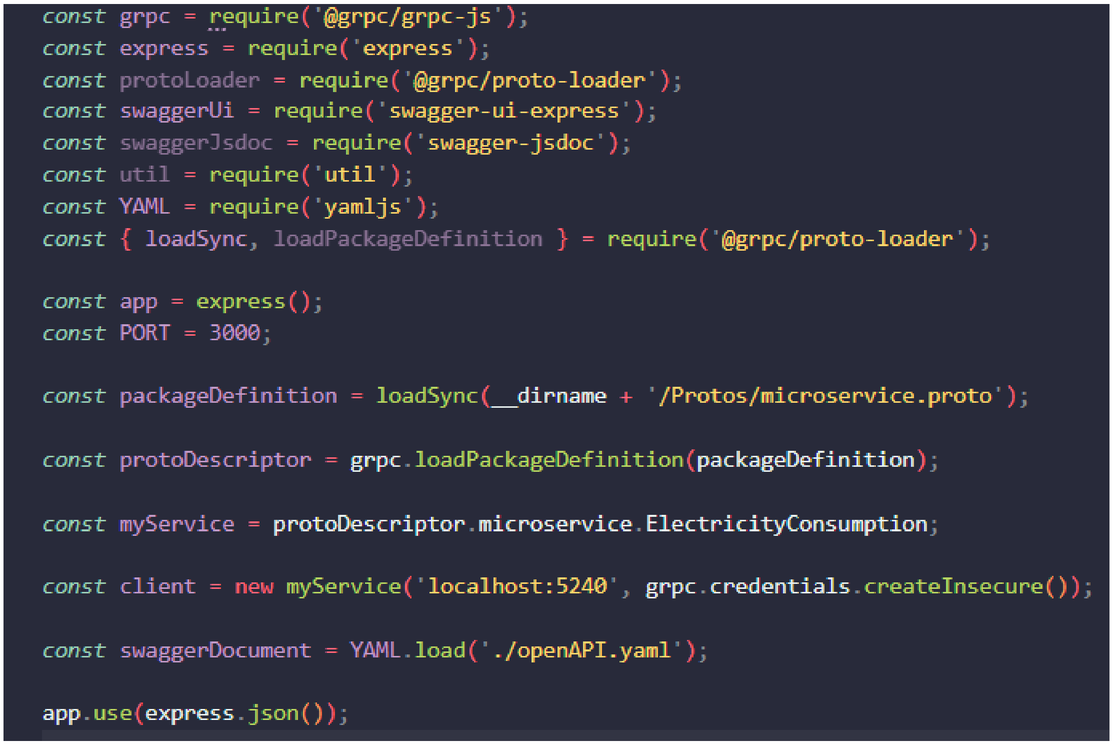


```
public override async Task<ValueMessage> AddElectricityConsumptionValue(ElectricityConsumptionValue request, ServerCallContext context)
{
    var objectId = ObjectId.GenerateNewId();

    var filter = Builders<ElectricityConsumptionModel>.Filter.Eq(x => x._id, objectId);

    var electricityConsumptionValue = await _collection.Find(filter).FirstOrDefaultAsync();

    if (electricityConsumptionValue != null)
    {
        return await Task.FromResult(new ValueMessage
        {
            Id = electricityConsumptionValue._id.ToString(),
            Message = "There is a electricity consumption with the same id in database"
        });
    }
    var newValue = new ElectricityConsumptionModel
    {
        _id = objectId,
        DateTime = DateTime.Now,
        Consumption = request.Consumption,
        Production = request.Production,
        Nuclear = request.Nuclear,
        Wind = request.Wind,
        Hydroelectric = request.Hydroelectric,
        OilAndGas = request.OilAndGas,
        Coal = request.Coal,
        Solar = request.Solar,
        Biomass = request.Biomass
    };
    await _collection.InsertOneAsync(newValue);

    return await Task.FromResult(new ValueMessage
    {
        Id = newValue._id.ToString(),
        Message = "Electricity consumption value added successfully"
    });
```

Funkcija za dodavanje novog dokumenta.

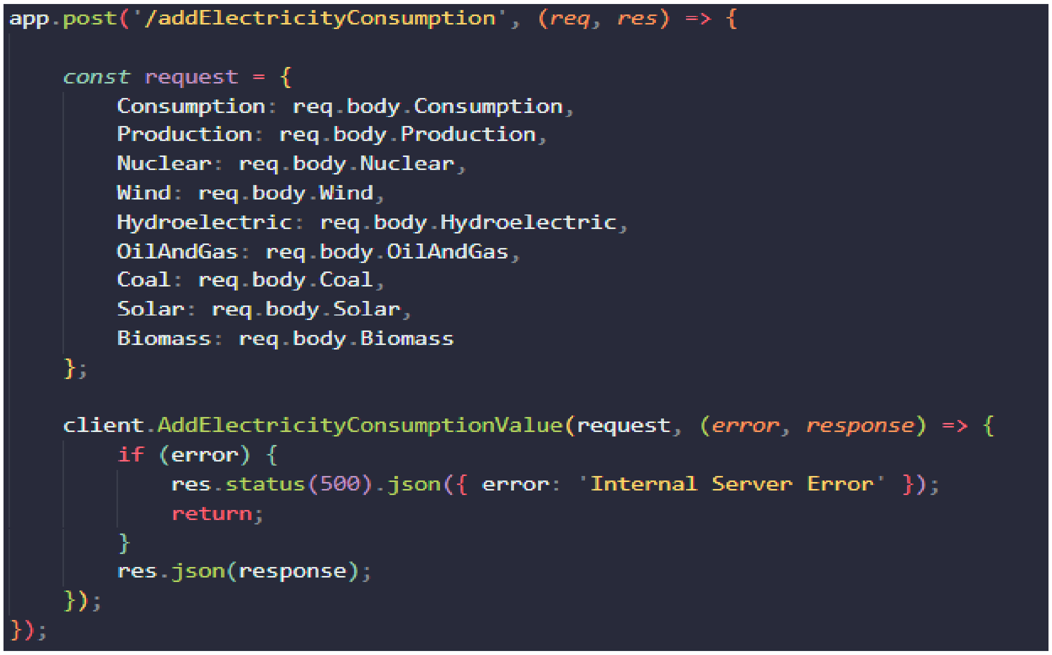
Aplikacija se pokreće sa dotnet run ili dotnet watch run.

Drugi mikroservis služi kao “klijent” prvom mikroservisu i ima ulogu da obesbedi RESTful servis. Kopira se protobuf fajl iz prvog mikroservisa, samo bez naznake da se prevodi u c#.



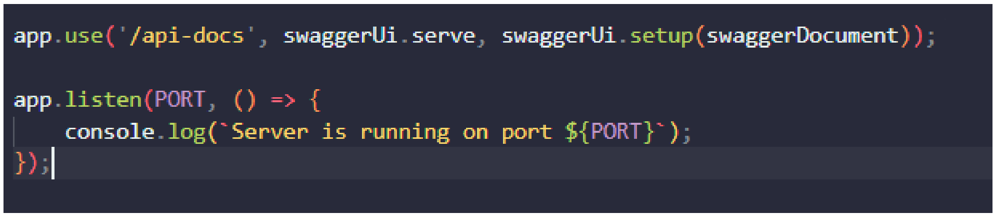
```
const grpc = require('@grpc/grpc-js');
const express = require('express');
const protoLoader = require('@grpc/proto-loader');
const swaggerUi = require('swagger-ui-express');
const swaggerJsdoc = require('swagger-jsdoc');
const util = require('util');
const YAML = require('yamljs');
const { loadSync, loadPackageDefinition } = require('@grpc/proto-loader');

const app = express();
const PORT = 3000;

const packageDefinition = loadSync(__dirname + '/Protos/microservice.proto');

const protoDescriptor = grpc.loadPackageDefinition(packageDefinition);

const myService = protoDescriptor.microservice.ElectricityConsumption;

const client = new myService('localhost:5240', grpc.credentials.createInsecure());

const swaggerDocument = YAML.load('./openAPI.yaml');

app.use(express.json());
```

Za kreiranje ruta se koristi express, i učitava se yaml fajl koji sadrži OpenAPI specifikaciju za swagger gde se aplikacija pokreće.



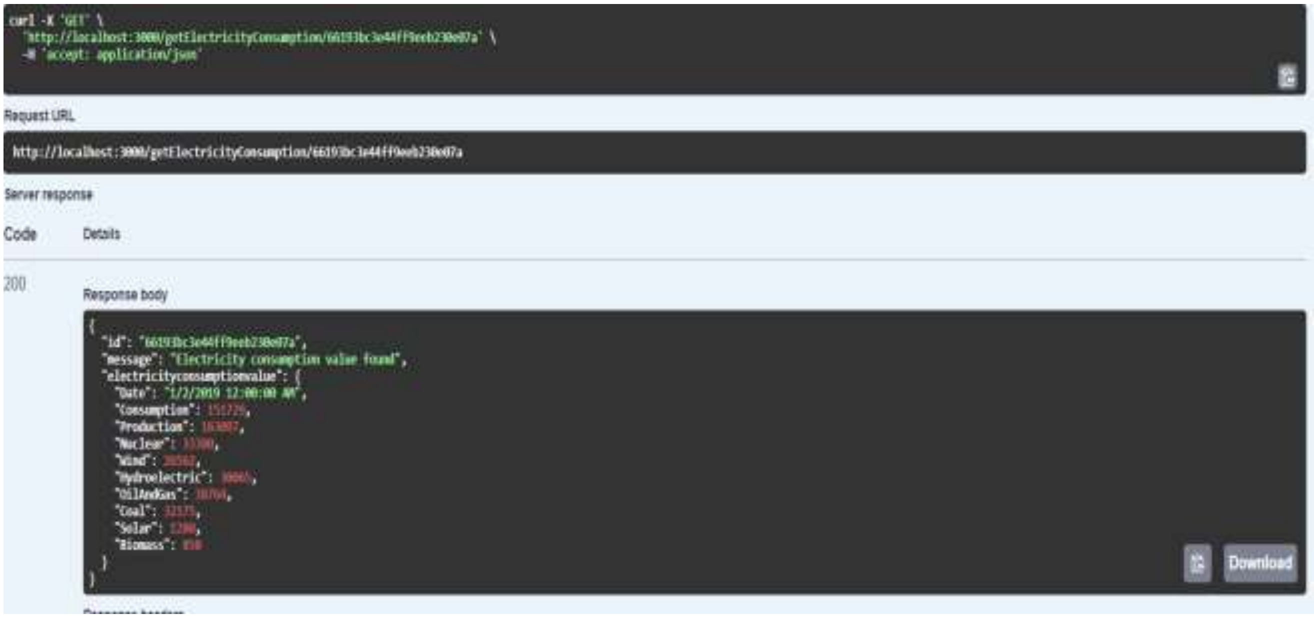
```
app.post('/addElectricityConsumption', (req, res) => {

    const request = {
        Consumption: req.body.Consumption,
        Production: req.body.Production,
        Nuclear: req.body.Nuclear,
        Wind: req.body.Wind,
        Hydroelectric: req.body.Hydroelectric,
        OilAndGas: req.body.OilAndGas,
        Coal: req.body.Coal,
        Solar: req.body.Solar,
        Biomass: req.body.Biomass
    };

    client.AddElectricityConsumptionValue(request, (error, response) => {
        if (error) {
            res.status(500).json({ error: 'Internal Server Error' });
            return;
        }
        res.json(response);
    });
});
```

Postavlja se klijent da sluša na portu gde je pokrenut prvi mikroservis, a iznad je primer kako se poziva metoda za kreiranje iz prvog mikroservisa.



```
app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerDocument));

app.listen(PORT, () => {
    console.log(`Server is running on port ${PORT}`);
});
```

Aplikacija se pokreće na swagger una određenom portu.



```
curl -X 'GET' \
  'http://localhost:3000/getElectricityConsumption/66193bc3e44ff9eeb230e07a' \
  -H 'accept: application/json'
```

Request URL

```
http://localhost:3000/getElectricityConsumption/66193bc3e44ff9eeb230e07a
```

Server response

Code Details

200 Response body

```
{
  "id": "66193bc3e44ff9eeb230e07a",
  "message": "Electricity consumption value found",
  "electricityconsumptionvalue": {
    "Date": "1/2/2019 12:00:00 AM",
    "Consumption": 151726,
    "Production": 163007,
    "Nuclear": 33300,
    "Wind": 20562,
    "Hydroelectric": 30065,
    "OilAndGas": 18704,
    "Coal": 32175,
    "Solar": 1290,
    "Biomass": 850
  }
}
```

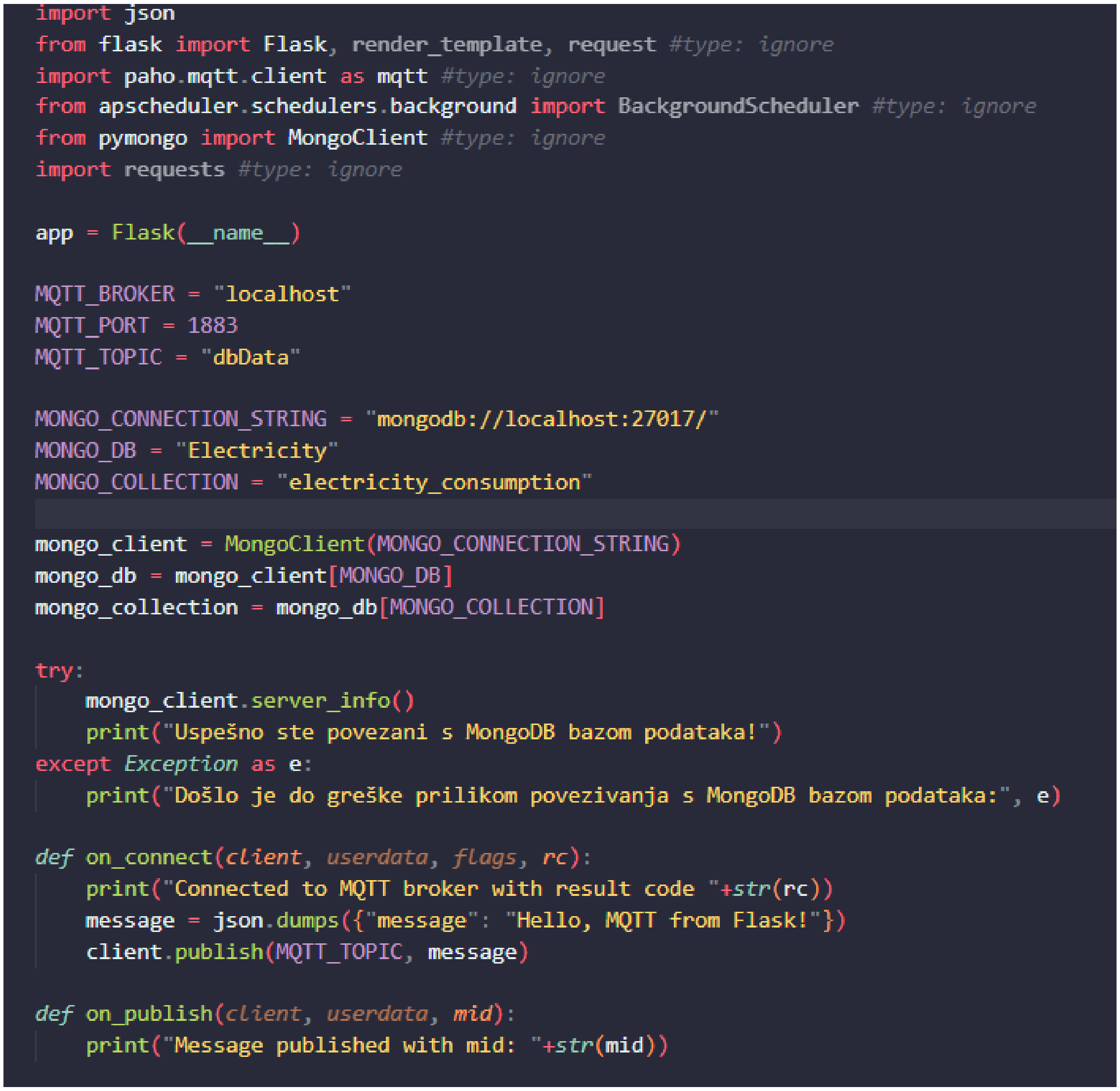
Primer jednog get zahteva.

Svaki mikroservis sadrži Dockerfile koji služi za kreiranje docker image, a u root direktorijumu se nalazi docker-compose koji na istu mrežu objedinjuje docker kontejnere za obe aplikacije.

Drugi

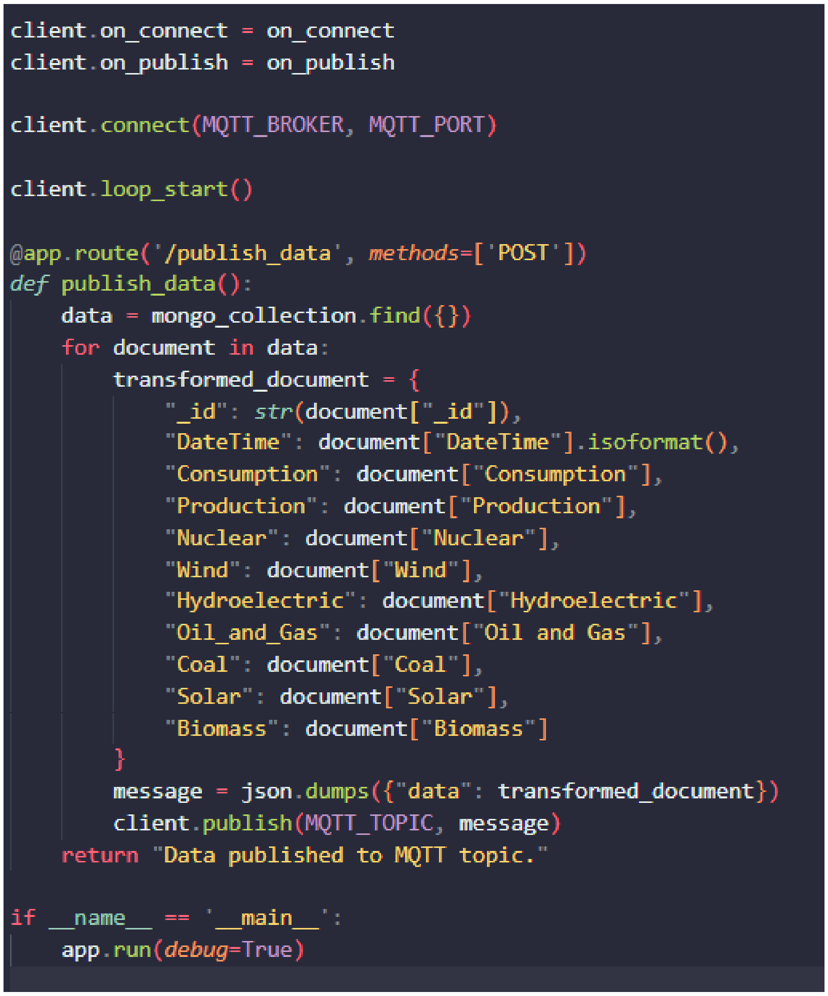
Drugi projekat se sastoji od 3 mikroservisa, prva 2 su kreirana kao Flask aplikacije (python), 3. je kreiran kao .net webapi.

Prvi mikroservis (Sensor) ima zadatak da se poveže na mqtt broker i na određeni topic pošalje sve podatke iz baze podataka.



```python
import json
from flask import Flask, render_template, request #type: ignore
import paho.mqtt.client as mqtt #type: ignore
from apscheduler.schedulers.background import BackgroundScheduler #type: ignore
from pymongo import MongoClient #type: ignore
import requests #type: ignore

app = Flask(__name__)

MQTT_BROKER = "localhost"
MQTT_PORT = 1883
MQTT_TOPIC = "dbData"

MONGO_CONNECTION_STRING = "mongodb://localhost:27017/"
MONGO_DB = "Electricity"
MONGO_COLLECTION = "electricity_consumption"

mongo_client = MongoClient(MONGO_CONNECTION_STRING)
mongo_db = mongo_client[MONGO_DB]
mongo_collection = mongo_db[MONGO_COLLECTION]

try:
    mongo_client.server_info()
    print("Uspešno ste povezani s MongoDB bazom podataka!")
except Exception as e:
    print("Došlo je do greške prilikom povezivanja s MongoDB bazom podataka:", e)

def on_connect(client, userdata, flags, rc):
    print("Connected to MQTT broker with result code "+str(rc))
    message = json.dumps({"message": "Hello, MQTT from Flask!"})
    client.publish(MQTT_TOPIC, message)

def on_publish(client, userdata, mid):
    print("Message published with mid: "+str(mid))
```

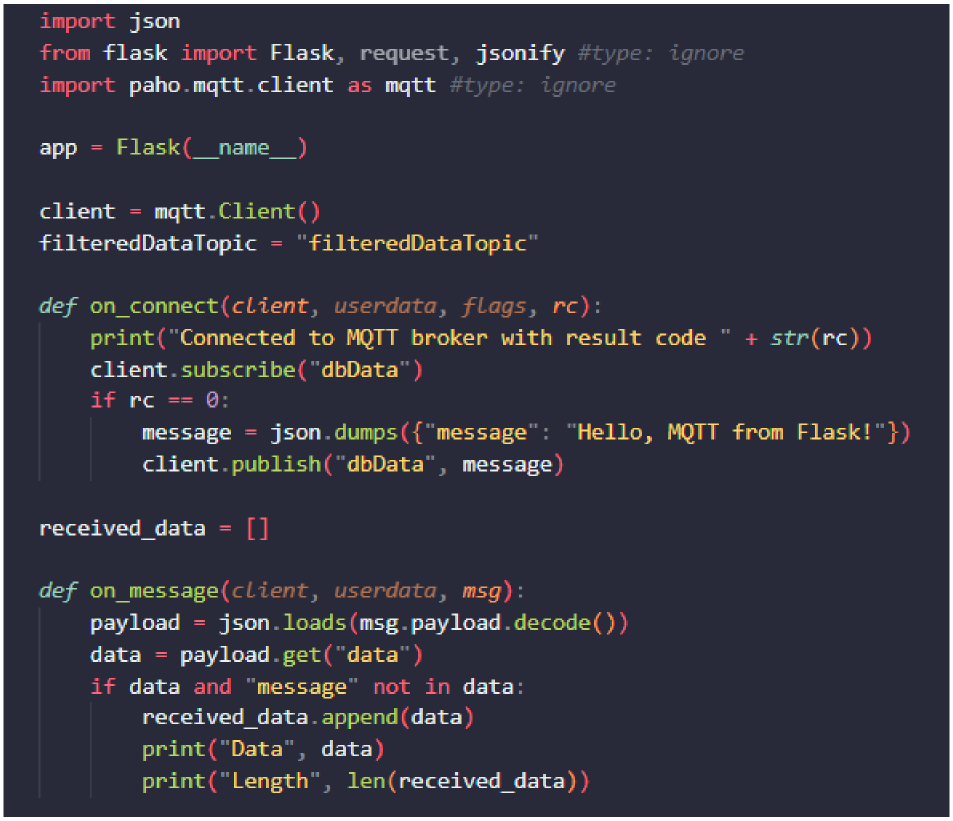
Ovde se vrši povezivanje sa bazom i mqtt brokerom i definišu se 2 metode, on_connect koja šalje poruku na topic kada se aplikacija uspešno poveže sa brokerom i on_publish koja u konzoli ispisuje poruku svaki put kada se nešto postavi na topic.



```python
client.on_connect = on_connect
client.on_publish = on_publish

client.connect(MQTT_BROKER, MQTT_PORT)

client.loop_start()

@app.route('/publish_data', methods=['POST'])
def publish_data():
    data = mongo_collection.find({})
    for document in data:
        transformed_document = {
            "_id": str(document["_id"]),
            "DateTime": document["DateTime"].isoformat(),
            "Consumption": document["Consumption"],
            "Production": document["Production"],
            "Nuclear": document["Nuclear"],
            "Wind": document["Wind"],
            "Hydroelectric": document["Hydroelectric"],
            "Oil_and_Gas": document["Oil and Gas"],
            "Coal": document["Coal"],
            "Solar": document["Solar"],
            "Biomass": document["Biomass"]
        }
        message = json.dumps({"data": transformed_document})
        client.publish(MQTT_TOPIC, message)
    return "Data published to MQTT topic."

if __name__ == '__main__':
    app.run(debug=True)
```

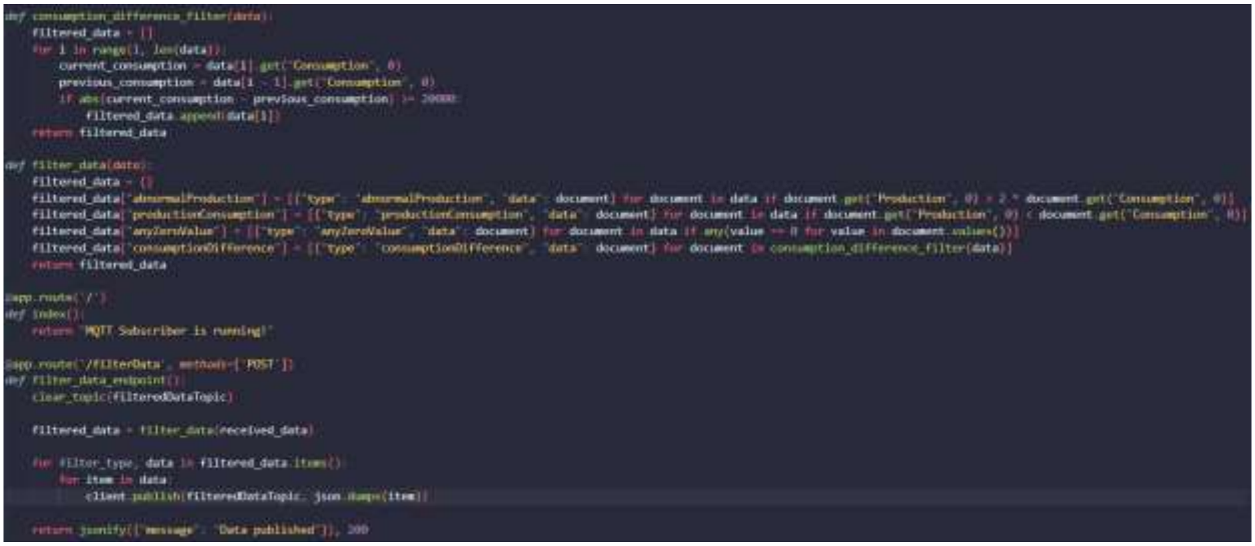
Klijentu se dodeljuju on_connect i on_publish metode i povezuje se sa određenim brokerom na određenom portu. Metoda loop_start uspostavlja mrežu na zasebnoj niti tako da aplikacija može da nastavi da radi dalje. Dodata je ruta koja uzima svaki document iz kolekcije i šalje ga na topic u json formatu gde je “data” ključ a dokument vrednost.

Drugi mikroservis ima ulogu da se subscribuje na topic gde prvi šalje podatke, primi ih i odredi neke anomalije.



```python
import json
from flask import Flask, request, jsonify #type: ignore
import paho.mqtt.client as mqtt #type: ignore

app = Flask(__name__)

client = mqtt.Client()
filteredDataTopic = "filteredDataTopic"

def on_connect(client, userdata, flags, rc):
    print("Connected to MQTT broker with result code " + str(rc))
    client.subscribe("dbData")
    if rc == 0:
        message = json.dumps({"message": "Hello, MQTT from Flask!"})
        client.publish("dbData", message)

received_data = []

def on_message(client, userdata, msg):
    payload = json.loads(msg.payload.decode())
    data = payload.get("data")
    if data and "message" not in data:
        received_data.append(data)
        print("Data", data)
        print("Length", len(received_data))
```

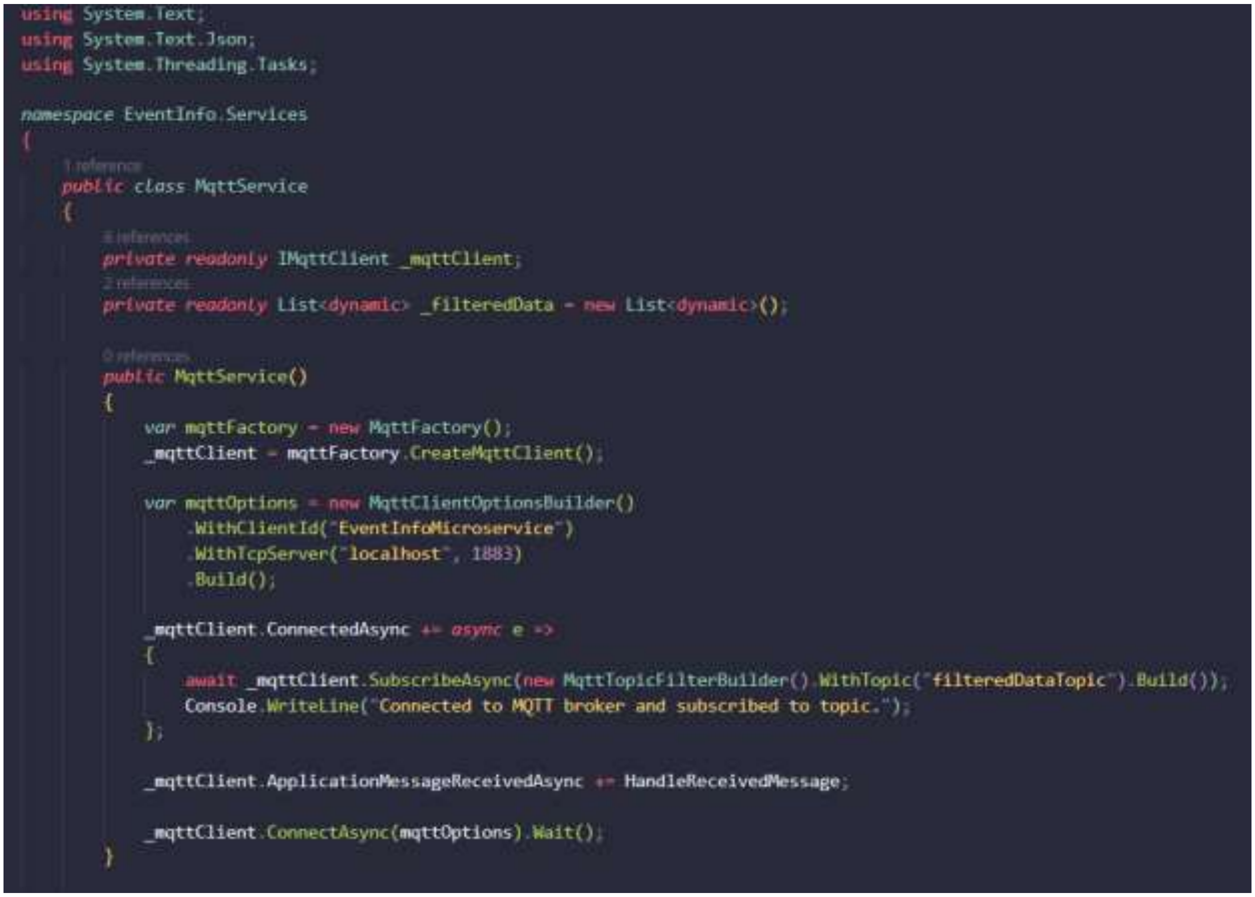
Aplikacija se povezuje na topic, i prilikom konekcije publishuje prvu poruku. Nakon svake primljene poruke, smešta njen sadržaj u niz received_data.



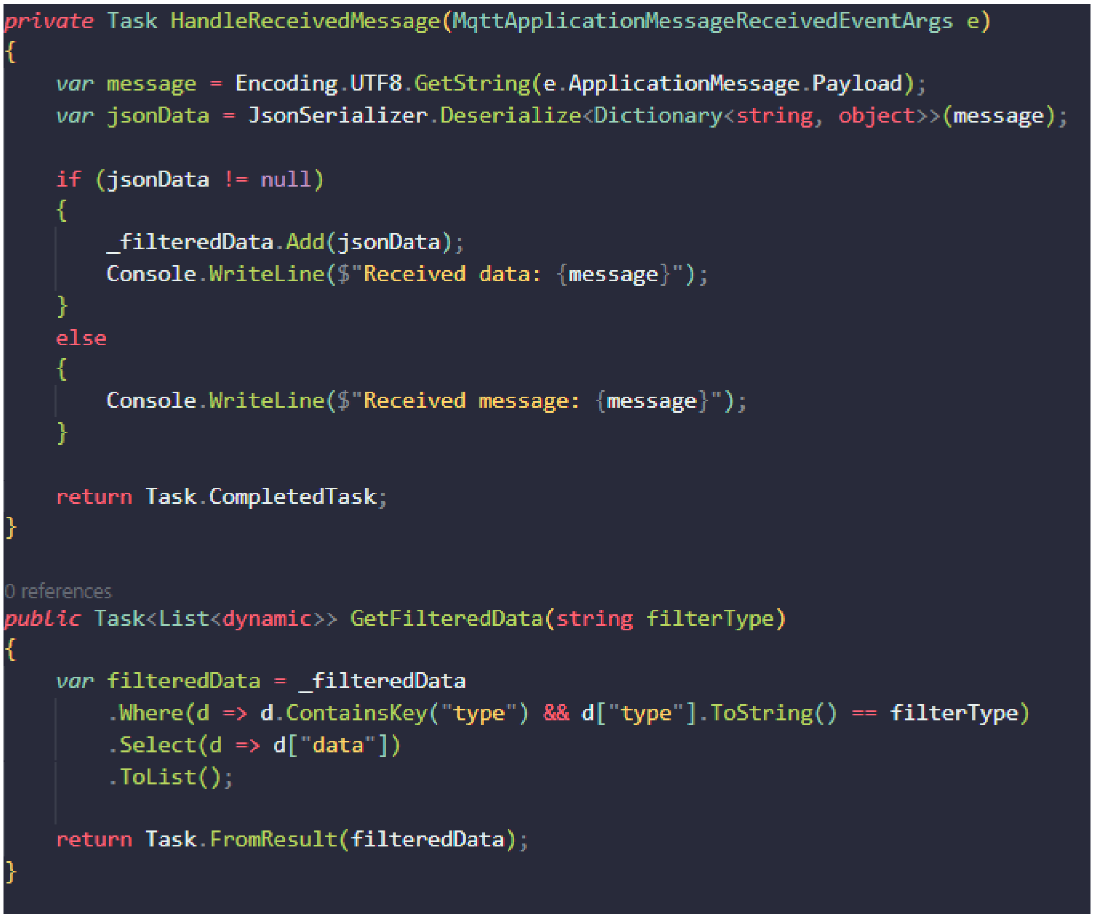
```python
def consumption_difference_filter(data):
    filtered_data = []
    for i in range(1, len(data)):
        current_consumption = data[i].get("Consumption", 0)
        previous_consumption = data[i - 1].get("Consumption", 0)
        if abs(current_consumption - previous_consumption) >= 20000:
            filtered_data.append(data[i])
    return filtered_data

def filter_data(data):
    filtered_data = {}
    filtered_data['abnormalProduction'] = [{'type': 'abnormalProduction', 'data': document} for document in data if document.get('Production', 0) > 2 * document.get('Consumption', 0)]
    filtered_data['productionConsumption'] = [{'type': 'productionConsumption', 'data': document} for document in data if document.get('Production', 0) < document.get('Consumption', 0)]
    filtered_data['anyZeroValue'] = [{'type': 'anyZeroValue', 'data': document} for document in data if any(value == 0 for value in document.values())]
    filtered_data['consumptionDifference'] = [{'type': 'consumptionDifference', 'data': document} for document in consumption_difference_filter(data)]
    return filtered_data

@app.route('/')
def index():
    return 'MQTT Subscriber is running!'

@app.route('/filterData', methods=['POST'])
def filter_data_endpoint():
    clear_topic(filteredDataTopic)

    filtered_data = filter_data(received_data)

    for filter_type, data in filtered_data.items():
        for item in data:
            client.publish(filteredDataTopic, json.dumps(item))

    return jsonify({'message': 'Data published'}), 200
```

filter_data funkcija kreira rečnik koji razdvaja o kojim filterima se radi i postavlja 4 različita filtera. U ruti se svi filtrirani podaci postavljaju kao json stringovi na topic.

Treći deo koji je dotnet webapi, ima ulogu da primi ove filtrirane podatke i prikaže ih preko REST servisa.



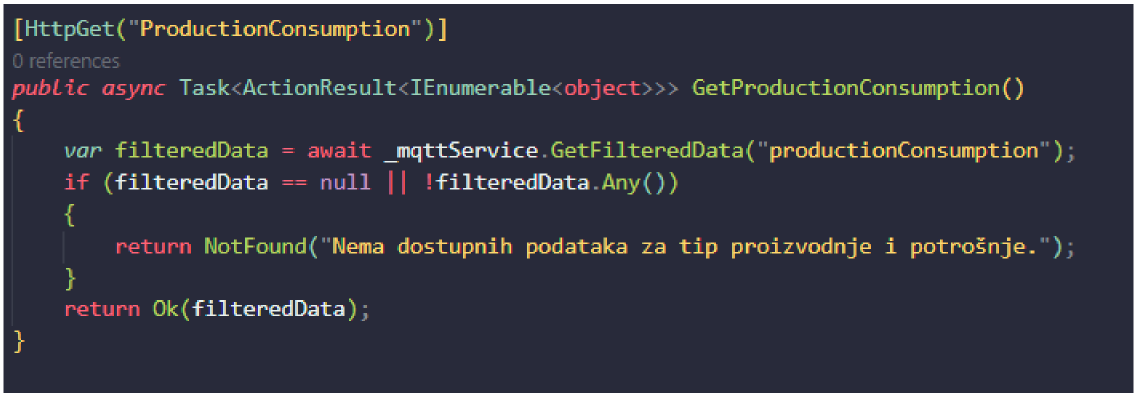
```
using System.Text;
using System.Text.Json;
using System.Threading.Tasks;

namespace EventInfo.Services
{
    1 reference
    public class MqttService
    {
        8 references
        private readonly IMqttClient _mqttClient;
        2 references
        private readonly List<dynamic> _filteredData = new List<dynamic>();

        0 references
        public MqttService()
        {
            var mqttFactory = new MqttFactory();
            _mqttClient = mqttFactory.CreateMqttClient();

            var mqttOptions = new MqttClientOptionsBuilder()
                .WithClientId("EventInfoMicroservice")
                .WithTcpServer("localhost", 1883)
                .Build();

            _mqttClient.ConnectedAsync += async e =>
            {
                await _mqttClient.SubscribeAsync(new MqttTopicFilterBuilder().WithTopic("filteredDataTopic").Build());
                Console.WriteLine("Connected to MQTT broker and subscribed to topic.");
            };

            _mqttClient.ApplicationMessageReceivedAsync += HandleReceivedMessage;

            _mqttClient.ConnectAsync(mqttOptions).Wait();
        }
```



```
private Task HandleReceivedMessage(MqttApplicationMessageReceivedEventArgs e)
{
    var message = Encoding.UTF8.GetString(e.ApplicationMessage.Payload);
    var jsonData = JsonSerializer.Deserialize<Dictionary<string, object>>(message);

    if (jsonData != null)
    {
        _filteredData.Add(jsonData);
        Console.WriteLine($"Received data: {message}");
    }
    else
    {
        Console.WriteLine($"Received message: {message}");
    }

    return Task.CompletedTask;
}

0 references
public Task<List<dynamic>> GetFilteredData(string filterType)
{
    var filteredData = _filteredData
        .Where(d => d.ContainsKey("type") && d["type"].ToString() == filterType)
        .Select(d => d["data"])
        .ToList();

    return Task.FromResult(filteredData);
}
```

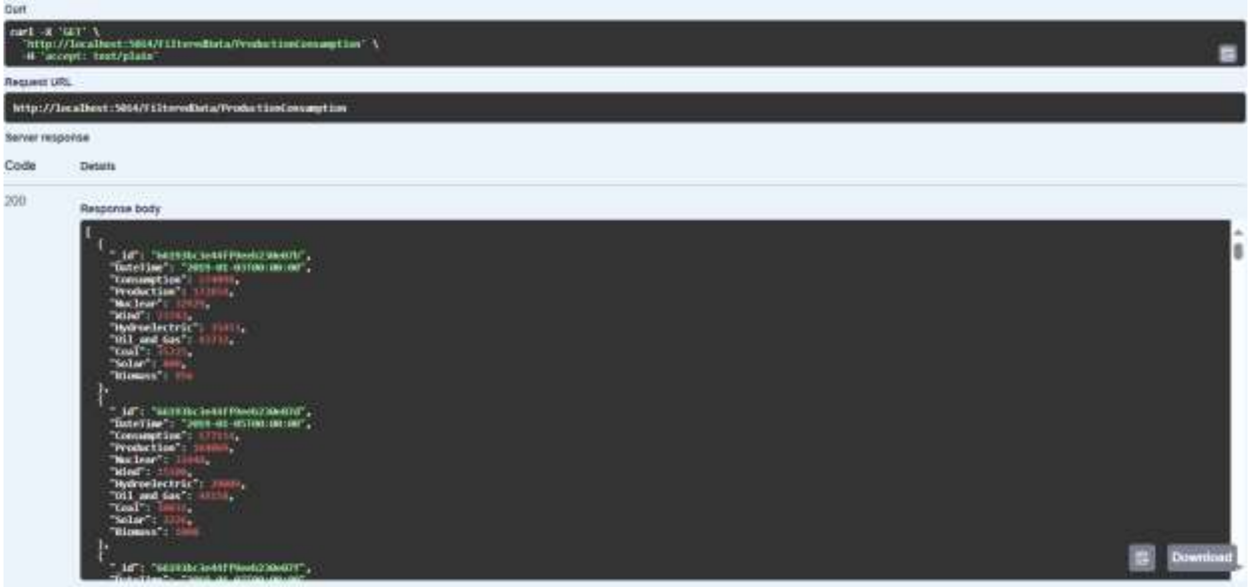
Funkcija HandleReceivedMessage, pretvara json string u rečnik za svaku poruku i smešta ga u listu, a GetFilteredData filtrira listu na osnovu vrednosti ključa type.

Primer kreiranje jedne rute za get zahtev. Funkciji GetFilteredData se prosleđuje type kao tip filtera po kome će se lista u servisu filtrirati.



```
[HttpGet("ProductionConsumption")]
0 references
public async Task<ActionResult<IEnumerable<object>>> GetProductionConsumption()
{
    var filteredData = await _mqttService.GetFilteredData("productionConsumption");
    if (filteredData == null || !filteredData.Any())
    {
        return NotFound("Nema dostupnih podataka za tip proizvodnje i potrošnje.");
    }
    return Ok(filteredData);
}
```

Primer izlaza na swaggeru:



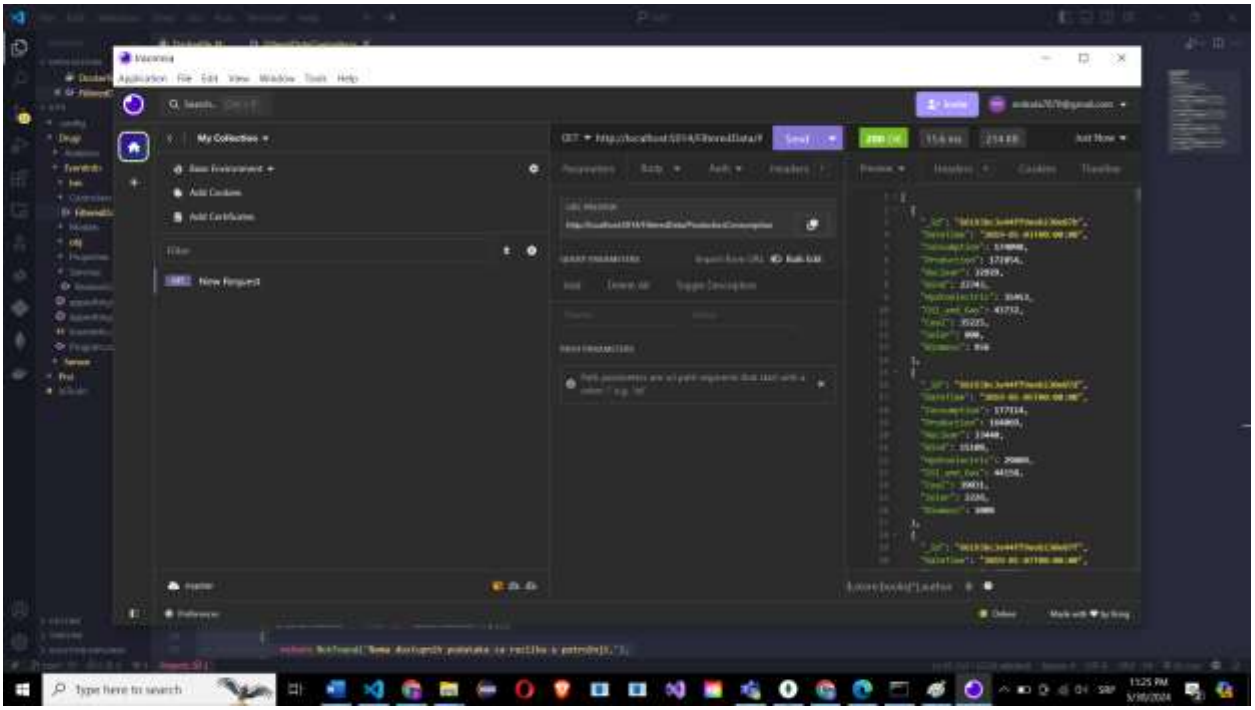
```
curl -X 'GET' \
  'http://localhost:5014/FilteredData/ProductionConsumption' \
  -H 'accept: text/plain'
```

```
http://localhost:5014/FilteredData/ProductionConsumption
```

200

```
[
  {
    "_id": "[illegible]",
    "DateTime": "2019-01-03T00:00:00",
    "Consumption": 174898,
    "Production": 172054,
    "Nuclear": 22929,
    "Wind": 22741,
    "Hydroelectric": 35453,
    "Oil and Gas": 43732,
    "Coal": 35225,
    "Solar": 800,
    "Biomass": 958
  },
  {
    "_id": "[illegible]",
    "DateTime": "2019-01-05T00:00:00",
    "Consumption": 177114,
    "Production": 164869,
    "Nuclear": 23440,
    "Wind": 15186,
    "Hydroelectric": 29869,
    "Oil and Gas": 44156,
    "Coal": 39831,
    "Solar": 2226,
    "Biomass": 3086
  },
  {
    "_id": "[illegible]",
```

Primer izlaza u insomnia:



```
GET http://localhost:5014/FilteredData/ProductionConsumption
```

```
[
  {
    "_id": "[illegible]",
    "DateTime": "2019-01-03T00:00:00",
    "Consumption": 174898,
    "Production": 172054,
    "Nuclear": 22929,
    "Wind": 22741,
    "Hydroelectric": 35453,
    "Oil_and_Gas": 43732,
    "Coal": 35225,
    "Solar": 800,
    "Biomass": 958
  },
  {
    "_id": "[illegible]",
    "DateTime": "2019-01-05T00:00:00",
    "Consumption": 177114,
    "Production": 164869,
    "Nuclear": 23440,
    "Wind": 15186,
    "Hydroelectric": 29869,
    "Oil_and_Gas": 44156,
    "Coal": 39831,
    "Solar": 2226,
    "Biomass": 3086
  },
  {
    "_id": "[illegible]",
    "DateTime": "[illegible]",
```

```
return NotFound("Nema dostupnih podataka za razliku u potrošnji.");
```