Projekat 1 – Internet stvari i servisa

Odabir skupa podataka

Podaci se mogu naći na sledećem linku: https://www.kaggle.com/datasets/deepcontractor/smoke-detection-dataset

Skup podataka se sastoji od oko 60.000 očitavanja, pri čemu su mereni sledeći podaci:

-temperatura, pritisak i vlažnost vazduha

-nestabilne organske čestice u vazduhu

-koncentracija CO2, H2, etanola

-veličine i koncentracije čestica

-aktivacija protivpožarnog alarma

Baza podataka

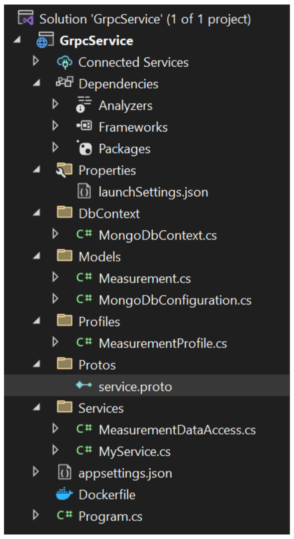
Za skladištenje merenja korišćena je MongoDB.

Za popunjavanje koristi se insertData.python skripta.

Implementacija gRPC mikroservisa

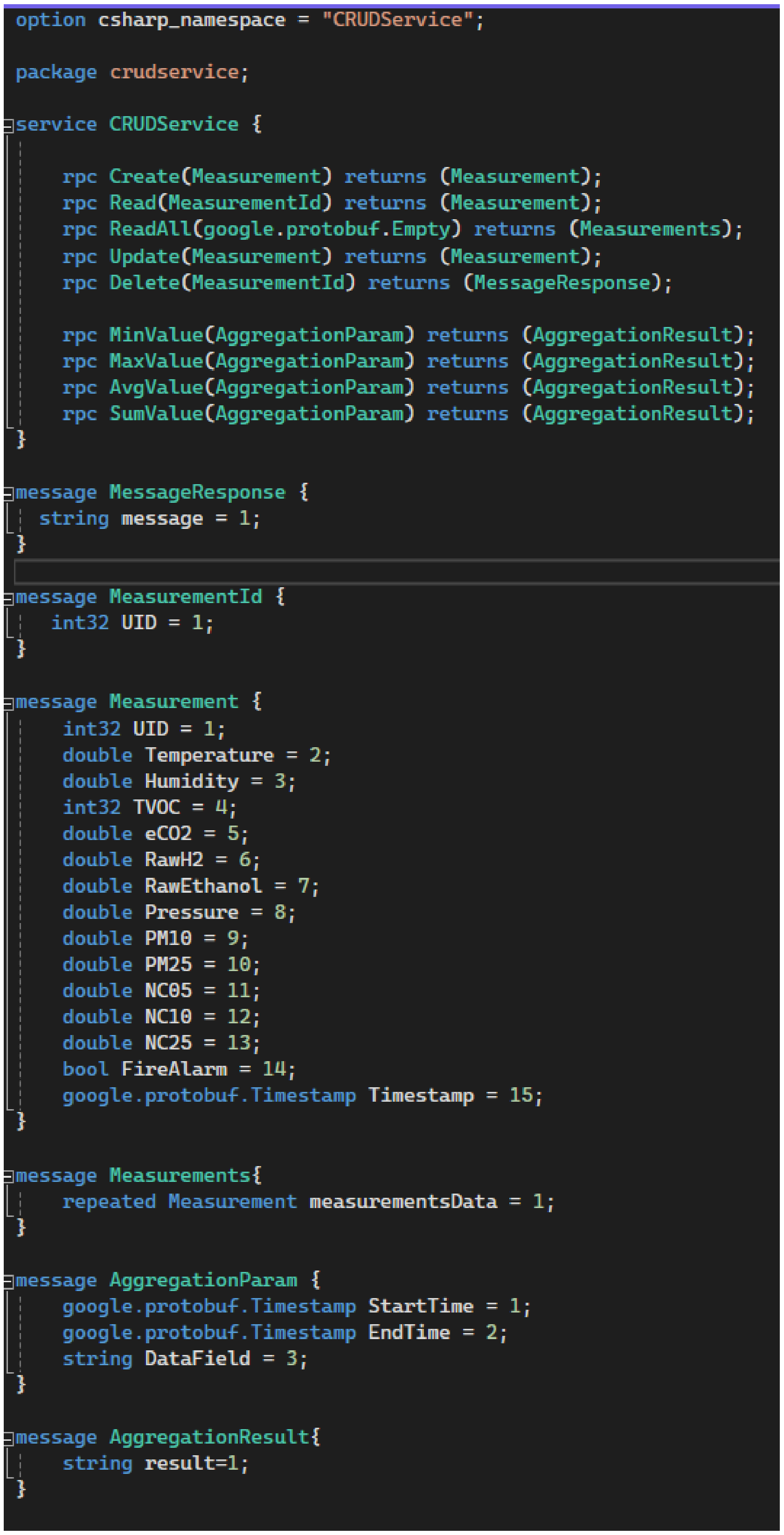
Implementiran je .NET Core-u. Obezbedjuje komunikaciju sa bazom i drugim mikroservisom.

Struktura projekta prikazana je na sledećoj slici:



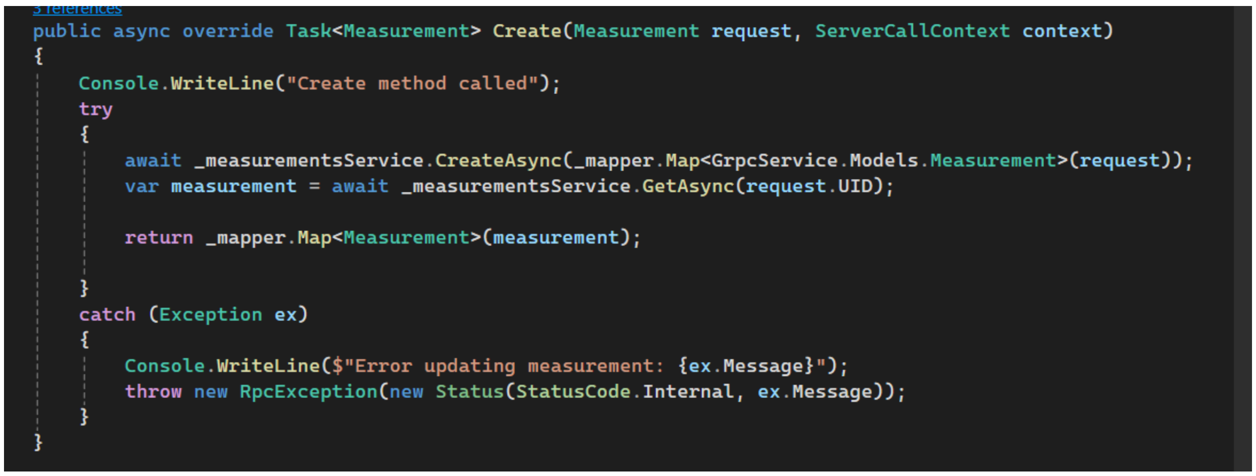
```
Solution 'GrpcService' (1 of 1 project)
◢ GrpcService
  ▷ Connected Services
  ◢ Dependencies
    ▷ Analyzers
    ▷ Frameworks
    ▷ Packages
  ◢ Properties
      launchSettings.json
  ◢ DbContext
    ▷ MongoDbContext.cs
  ◢ Models
    ▷ Measurement.cs
    ▷ MongoDbConfiguration.cs
  ◢ Profiles
    ▷ MeasurementProfile.cs
  ◢ Protos
      service.proto
  ◢ Services
    ▷ MeasurementDataAccess.cs
    ▷ MyService.cs
  ▷ appsettings.json
    Dockerfile
  ▷ Program.cs
```

Izgled Protobuf specifikacije:



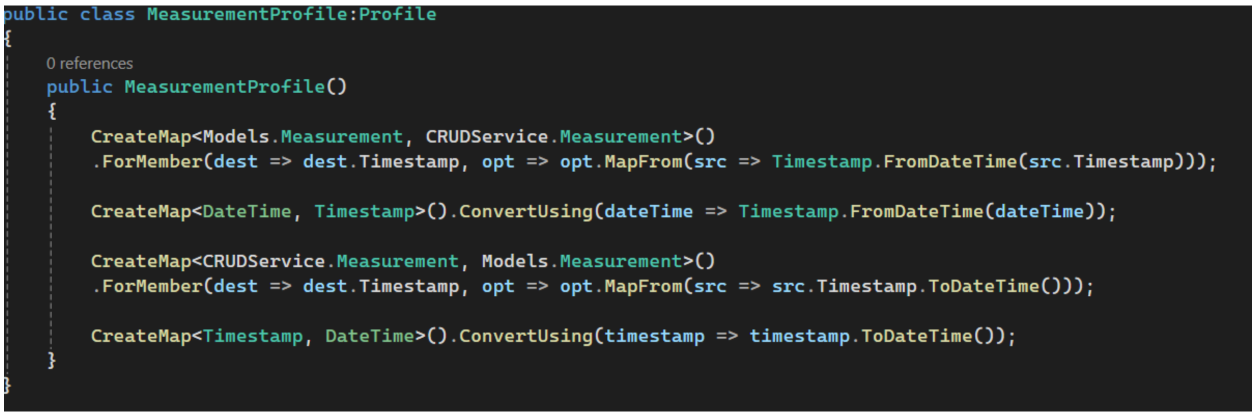
```
option csharp_namespace = "CRUDService";

package crudservice;

service CRUDService {

    rpc Create(Measurement) returns (Measurement);
    rpc Read(MeasurementId) returns (Measurement);
    rpc ReadAll(google.protobuf.Empty) returns (Measurements);
    rpc Update(Measurement) returns (Measurement);
    rpc Delete(MeasurementId) returns (MessageResponse);

    rpc MinValue(AggregationParam) returns (AggregationResult);
    rpc MaxValue(AggregationParam) returns (AggregationResult);
    rpc AvgValue(AggregationParam) returns (AggregationResult);
    rpc SumValue(AggregationParam) returns (AggregationResult);
}

message MessageResponse {
  string message = 1;
}

message MeasurementId {
   int32 UID = 1;
}

message Measurement {
    int32 UID = 1;
    double Temperature = 2;
    double Humidity = 3;
    int32 TVOC = 4;
    double eCO2 = 5;
    double RawH2 = 6;
    double RawEthanol = 7;
    double Pressure = 8;
    double PM10 = 9;
    double PM25 = 10;
    double NC05 = 11;
    double NC10 = 12;
    double NC25 = 13;
    bool FireAlarm = 14;
    google.protobuf.Timestamp Timestamp = 15;
}

message Measurements{
    repeated Measurement measurementsData = 1;
}

message AggregationParam {
    google.protobuf.Timestamp StartTime = 1;
    google.protobuf.Timestamp EndTime = 2;
    string DataField = 3;
}

message AggregationResult{
    string result=1;
}
```

MyService.cs: (Create procedura, po istom principu i ostale procedure)



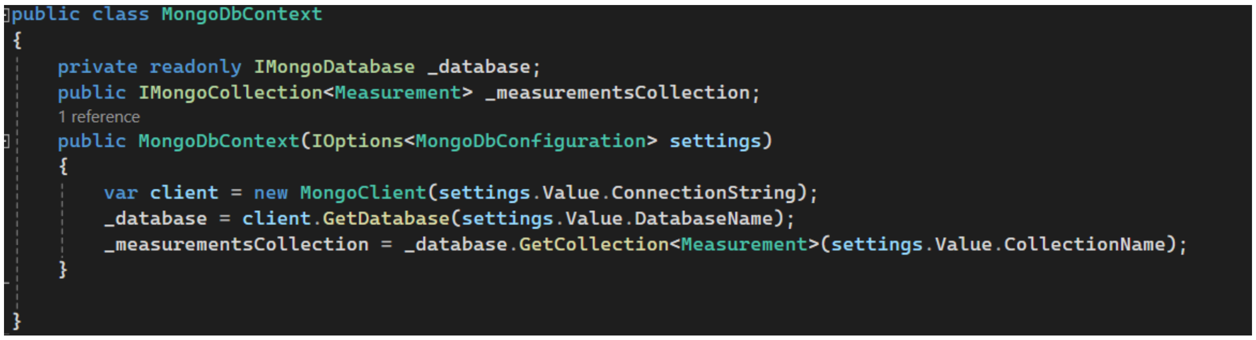
```
3 references
public async override Task<Measurement> Create(Measurement request, ServerCallContext context)
{
    Console.WriteLine("Create method called");
    try
    {
        await _measurementsService.CreateAsync(_mapper.Map<GrpcService.Models.Measurement>(request));
        var measurement = await _measurementsService.GetAsync(request.UID);

        return _mapper.Map<Measurement>(measurement);

    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error updating measurement: {ex.Message}");
        throw new RpcException(new Status(StatusCode.Internal, ex.Message));
    }
}
```

MeasurementProfile.cs:



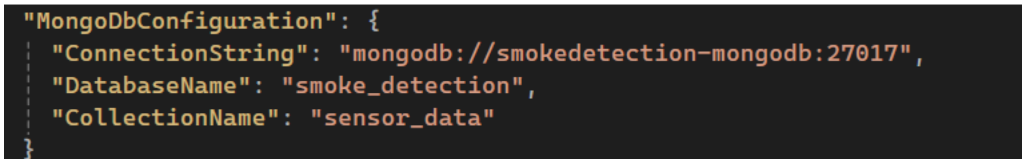
```
public class MeasurementProfile:Profile
{
    0 references
    public MeasurementProfile()
    {
        CreateMap<Models.Measurement, CRUDService.Measurement>()
        .ForMember(dest => dest.Timestamp, opt => opt.MapFrom(src => Timestamp.FromDateTime(src.Timestamp)));

        CreateMap<DateTime, Timestamp>().ConvertUsing(dateTime => Timestamp.FromDateTime(dateTime));

        CreateMap<CRUDService.Measurement, Models.Measurement>()
        .ForMember(dest => dest.Timestamp, opt => opt.MapFrom(src => src.Timestamp.ToDateTime()));

        CreateMap<Timestamp, DateTime>().ConvertUsing(timestamp => timestamp.ToDateTime());
    }
}
```

MongoDbContext.cs :



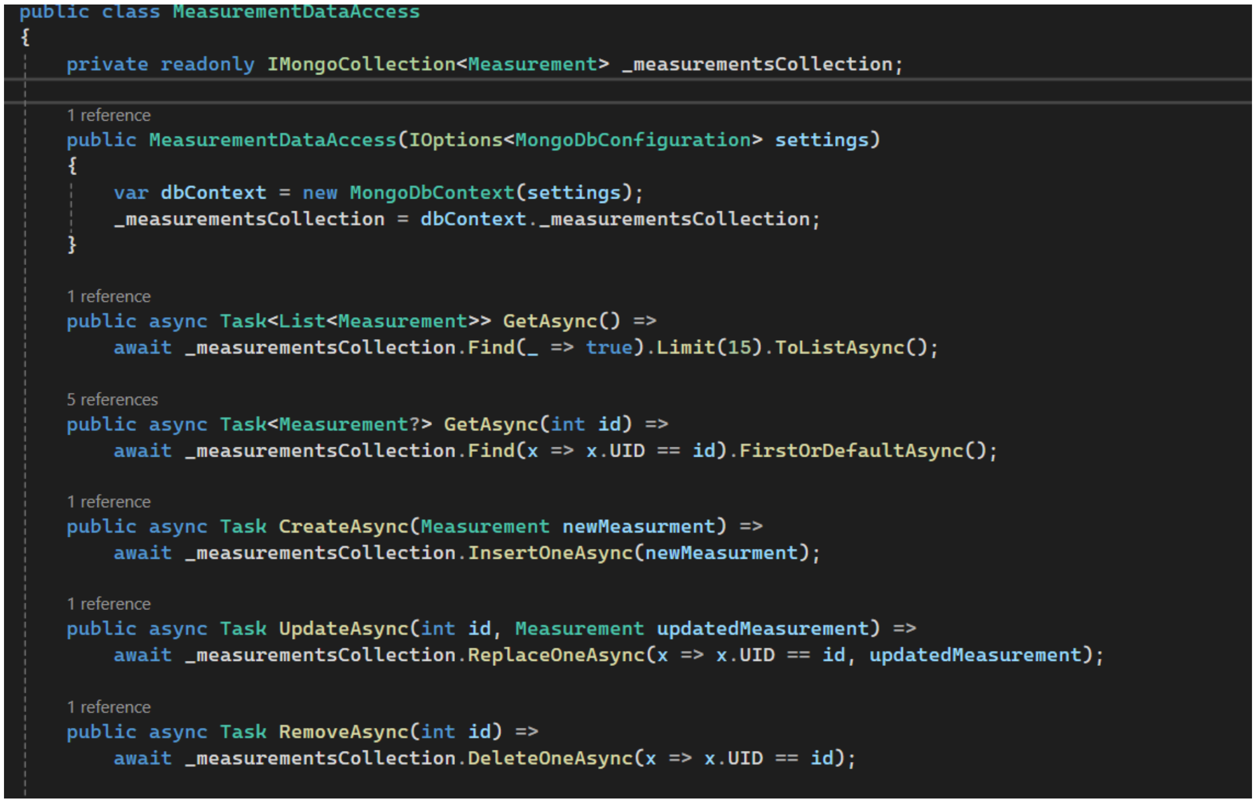
```
public class MongoDbContext
{
    private readonly IMongoDatabase _database;
    public IMongoCollection<Measurement> _measurementsCollection;
    1 reference
    public MongoDbContext(IOptions<MongoDbConfiguration> settings)
    {
        var client = new MongoClient(settings.Value.ConnectionString);
        _database = client.GetDatabase(settings.Value.DatabaseName);
        _measurementsCollection = _database.GetCollection<Measurement>(settings.Value.CollectionName);
    }

}
```

Konfiguracija u appsettings.json:



```
"MongoDbConfiguration": {
  "ConnectionString": "mongodb://smokedetection-mongodb:27017",
  "DatabaseName": "smoke_detection",
  "CollectionName": "sensor_data"
}
```

MeasurementDataAccess.cs:

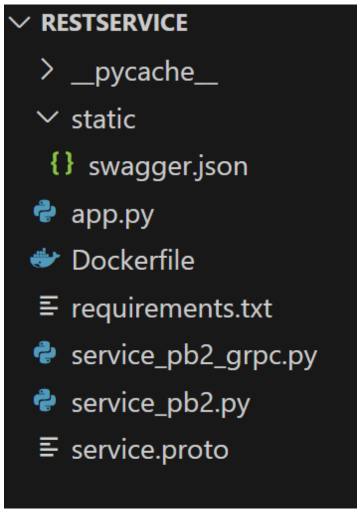


```
public class MeasurementDataAccess
{
    private readonly IMongoCollection<Measurement> _measurementsCollection;

    1 reference
    public MeasurementDataAccess(IOptions<MongoDbConfiguration> settings)
    {
        var dbContext = new MongoDbContext(settings);
        _measurementsCollection = dbContext._measurementsCollection;
    }

    1 reference
    public async Task<List<Measurement>> GetAsync() =>
        await _measurementsCollection.Find(_ => true).Limit(15).ToListAsync();

    5 references
    public async Task<Measurement?> GetAsync(int id) =>
        await _measurementsCollection.Find(x => x.UID == id).FirstOrDefaultAsync();

    1 reference
    public async Task CreateAsync(Measurement newMeasurment) =>
        await _measurementsCollection.InsertOneAsync(newMeasurment);

    1 reference
    public async Task UpdateAsync(int id, Measurement updatedMeasurement) =>
        await _measurementsCollection.ReplaceOneAsync(x => x.UID == id, updatedMeasurement);

    1 reference
    public async Task RemoveAsync(int id) =>
        await _measurementsCollection.DeleteOneAsync(x => x.UID == id);
```

Implementacija REST servisa

Implementiran kao Flask aplikacija u Python-u. Služi za komunikaciju sa klijentom.

Struktura projekta prikazana je na sledećoj slici:



```
RESTSERVICE
  __pycache__
  static
    swagger.json
  app.py
  Dockerfile
  requirements.txt
  service_pb2_grpc.py
  service_pb2.py
  service.proto
```

Potrebno je instalirati grcpio-tools.

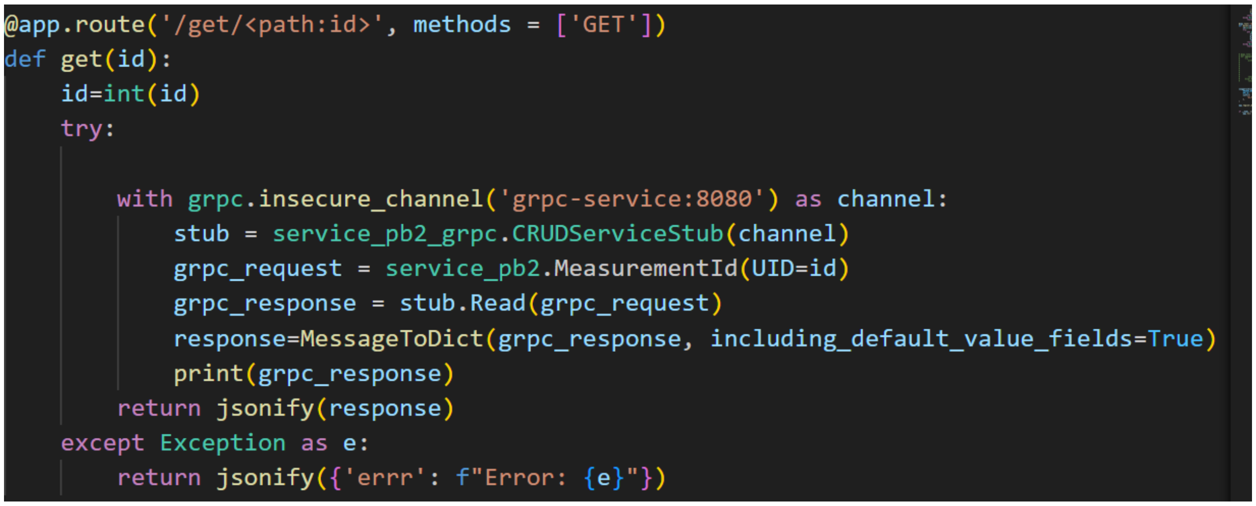
```
pip install grpcio grpcio-tools
```

Za kreiranje service_pb2_grpc.py (generisana servis klasa) i service_pb2.py (sadrži response i request klase) potrebno je pokrenuti sledeću komandu:

```
python -m grpc_tools.protoc -I. --python_out=. --grpc_python_out=. service.proto
```

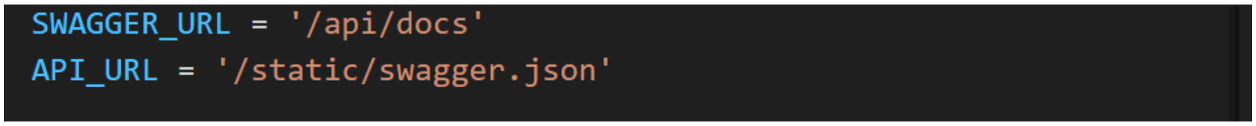
app.py:

get metod:

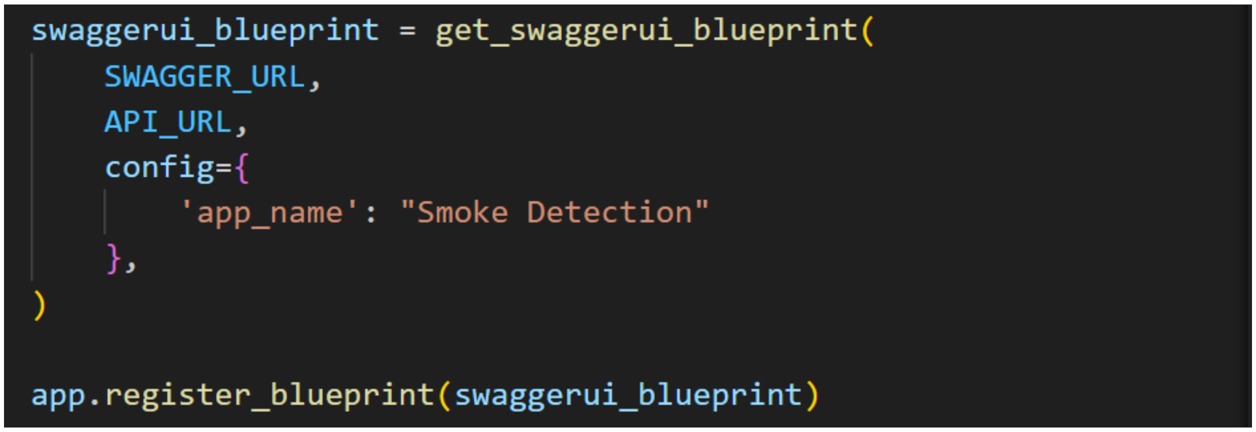


```python
@app.route('/get/<path:id>', methods = ['GET'])
def get(id):
    id=int(id)
    try:

        with grpc.insecure_channel('grpc-service:8080') as channel:
            stub = service_pb2_grpc.CRUDServiceStub(channel)
            grpc_request = service_pb2.MeasurementId(UID=id)
            grpc_response = stub.Read(grpc_request)
            response=MessageToDict(grpc_response, including_default_value_fields=True)
            print(grpc_response)
        return jsonify(response)
    except Exception as e:
        return jsonify({'errr': f"Error: {e}"})
```

Omogućavanje OpenAPI specifikacije pomoću flask_swagger_ui.

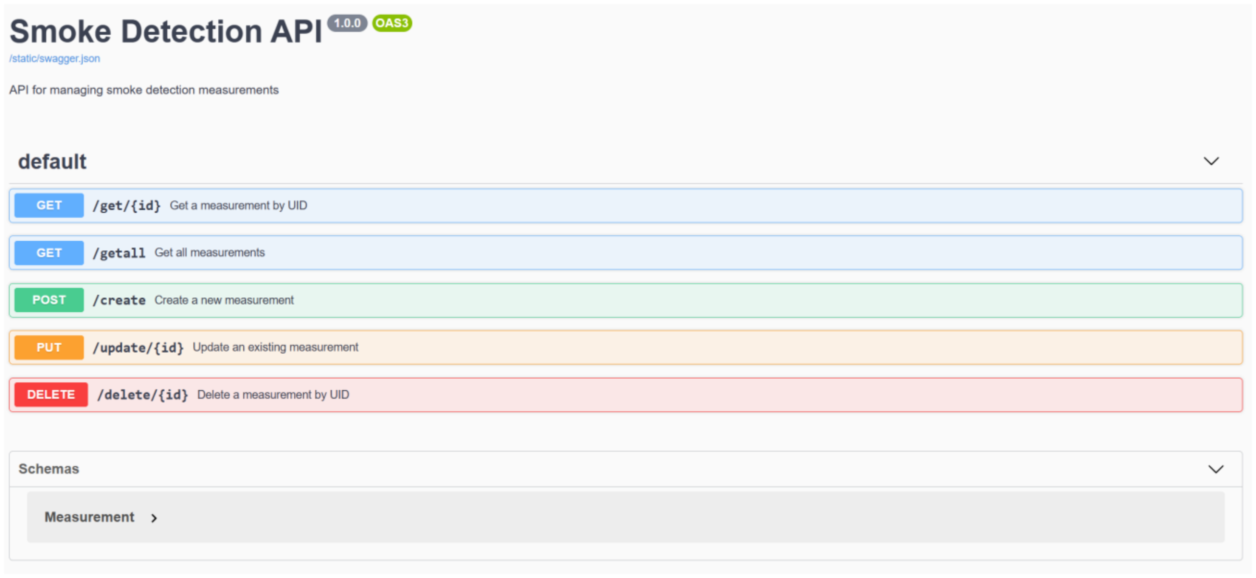


```python
SWAGGER_URL = '/api/docs'
API_URL = '/static/swagger.json'
```



```python
swaggerui_blueprint = get_swaggerui_blueprint(
    SWAGGER_URL,
    API_URL,
    config={
        'app_name': "Smoke Detection"
    },
)

app.register_blueprint(swaggerui_blueprint)
```

Swagger:



Smoke Detection API 1.0.0 OAS3

/static/swagger.json

API for managing smoke detection measurements

default

GET	/get/{id}	Get a measurement by UID
GET	/getall	Get all measurements
POST	/create	Create a new measurement
PUT	/update/{id}	Update an existing measurement
DELETE	/delete/{id}	Delete a measurement by UID

Schemas

Measurement

<u>Startovanje mikroservisa i baze kao Docker kontejnera</u>

Kreiranje mreze:

```
docker network create mynetwork
```

Baza:

```
docker run  --network mynetwork  -d --name smokedetection-mongodb -p 27017:27017 mongo
```

GRPC servis:

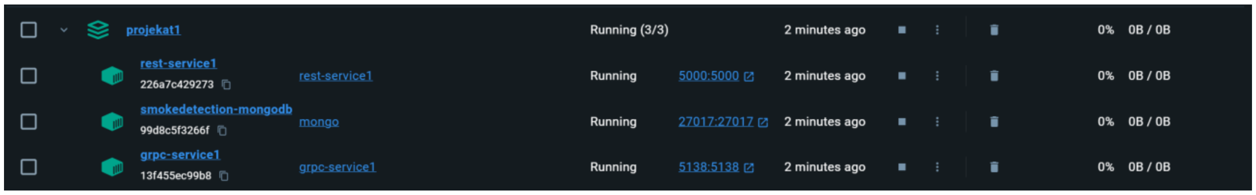
```
docker build -t grpc-service .
docker run -d --network mynetwork -p 5138:8080 --name grpc-service grpc-service
```

REST servis:

```
docker build -t rest-service .
docker run --network mynetwork -p 5000:5000 --name rest-service  rest-service
```

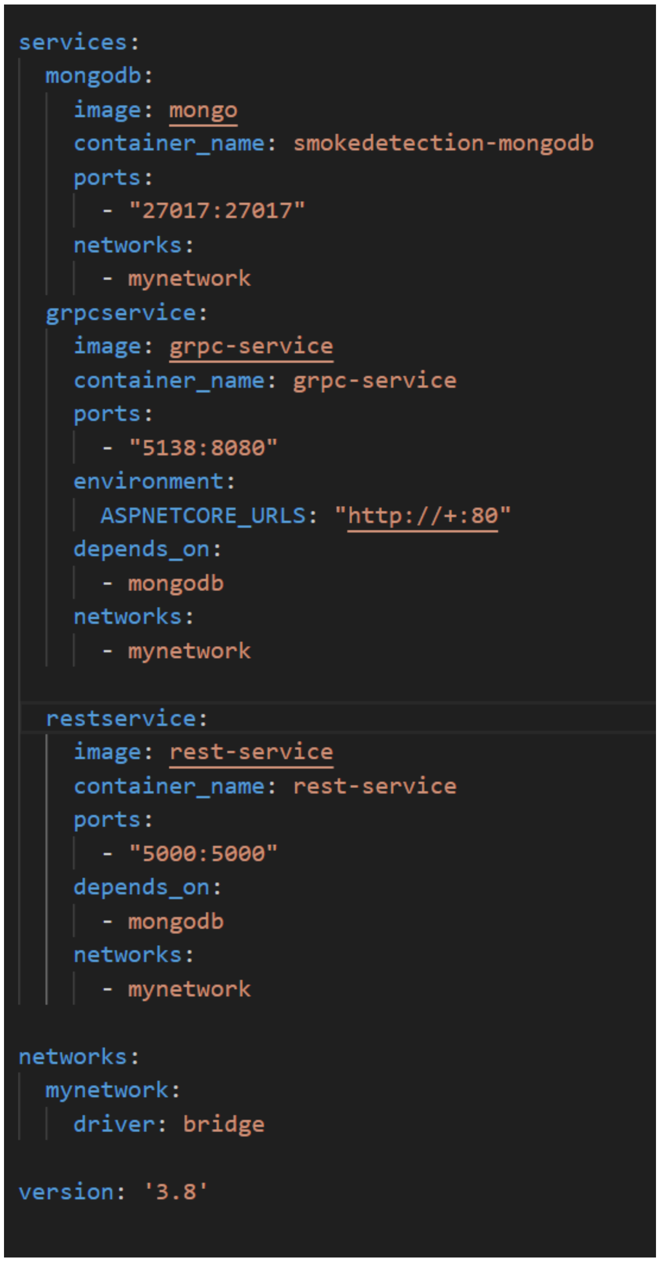
Pokretanje iz Docke-compose fajla:

```
docker-compose build
docker-compose up
```



projekat1		Running (3/3)		2 minutes ago	0%	0B / 0B
rest-service1 226a7c429273	rest-service1	Running	5000:5000	2 minutes ago	0%	0B / 0B
smokedetection-mongodb 99d8c5f3266f	mongo	Running	27017:27017	2 minutes ago	0%	0B / 0B
grpc-service1 13f455ec99b8	grpc-service1	Running	5138:5138	2 minutes ago	0%	0B / 0B

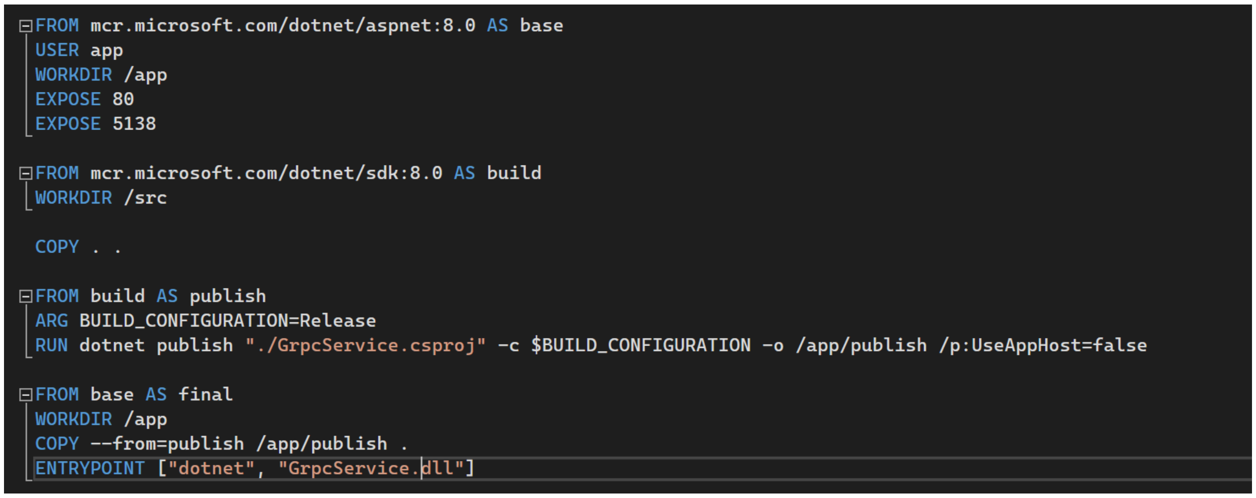
Izgled Docker compose fajla:



```yaml
services:
  mongodb:
    image: mongo
    container_name: smokedetection-mongodb
    ports:
      - "27017:27017"
    networks:
      - mynetwork
  grpcservice:
    image: grpc-service
    container_name: grpc-service
    ports:
      - "5138:8080"
    environment:
      ASPNETCORE_URLS: "http://+:80"
    depends_on:
      - mongodb
    networks:
      - mynetwork

  restservice:
    image: rest-service
    container_name: rest-service
    ports:
      - "5000:5000"
    depends_on:
      - mongodb
    networks:
      - mynetwork

networks:
  mynetwork:
    driver: bridge

version: '3.8'
```

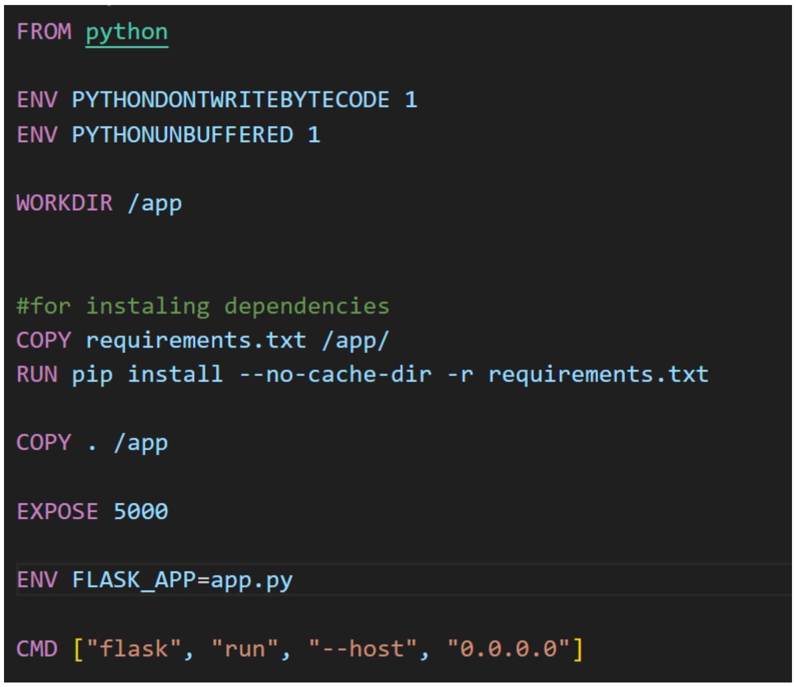
Izgled Docker fajlova:

Za gRPC servis:



```
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
USER app
WORKDIR /app
EXPOSE 80
EXPOSE 5138

FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /src

COPY . .

FROM build AS publish
ARG BUILD_CONFIGURATION=Release
RUN dotnet publish "./GrpcService.csproj" -c $BUILD_CONFIGURATION -o /app/publish /p:UseAppHost=false

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "GrpcService.dll"]
```

Za REST servis:



```
FROM python

ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

WORKDIR /app


#for instaling dependencies
COPY requirements.txt /app/
RUN pip install --no-cache-dir -r requirements.txt

COPY . /app

EXPOSE 5000

ENV FLASK_APP=app.py

CMD ["flask", "run", "--host", "0.0.0.0"]
```

Testiranje

Testiranje se može obaviti pomoću Postman-a ili na http://localhost:5000/api/docs/ .