

# Servisno-orijentisane arhitekture

## Opis projekta

### Studenti:

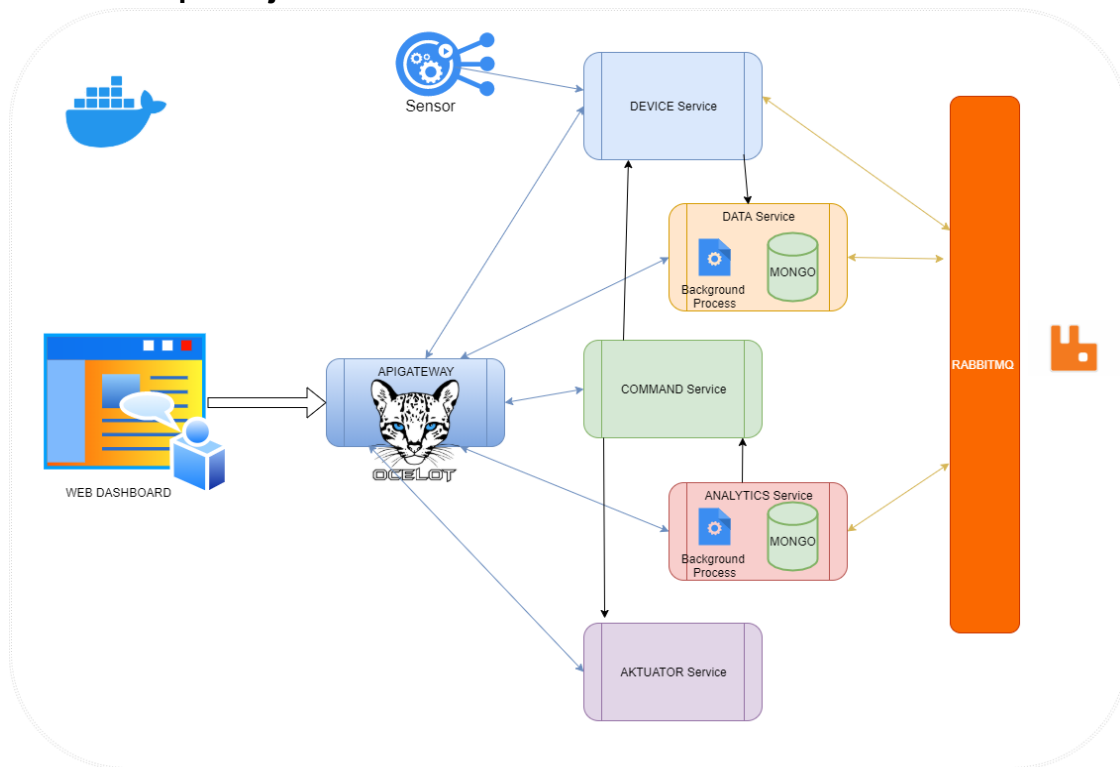
Predrag Antić 15995

Miloš Veljković 16021

### Opis aplikacije:

Aplikacija iz predmeta SOA je realizovana korišćenjem **.Net Core** tehnologije. Komunikacija u aplikaciji se odvija preko brokera **RabbitMQ-a**, dok se skladištenje podataka vrši u NoSql **Mongo** bazama. Osnovna ideja aplikacije je čitanje podataka o kvalitetu vazduha sa senzora, skladištenje tih podataka u NoSql bazi, ali i slanje asinhronih obaveštenja na klijentski deo aplikacije. Aplikacija se sastoji iz sledećih servisa: **Device**, **Data**, **Command**, **Aktuator**, **Analytics**, **ApiGateway** i **WebDashboard (Angular aplikacija)**.

### Osnovna šema aplikacije:

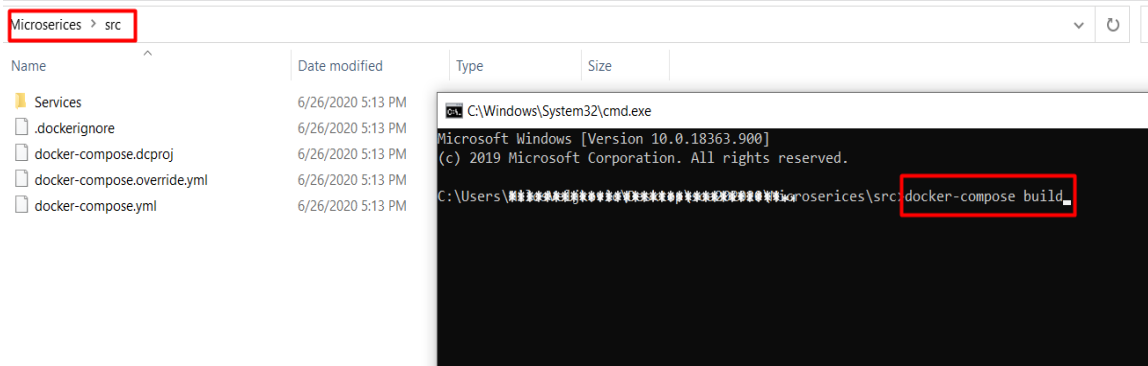


### Git repozitorijum projekta:

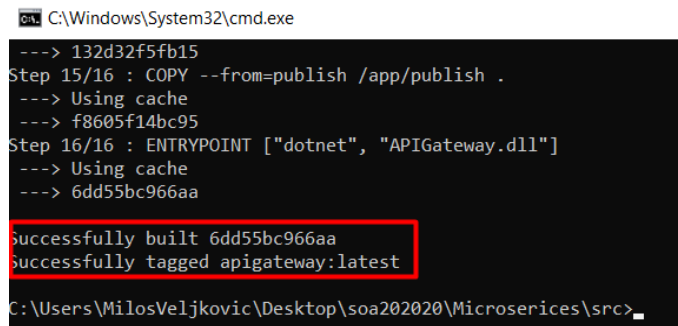
<https://github.com/milosveljkovic/Microserices>

## Kako pokrenuti projekat?

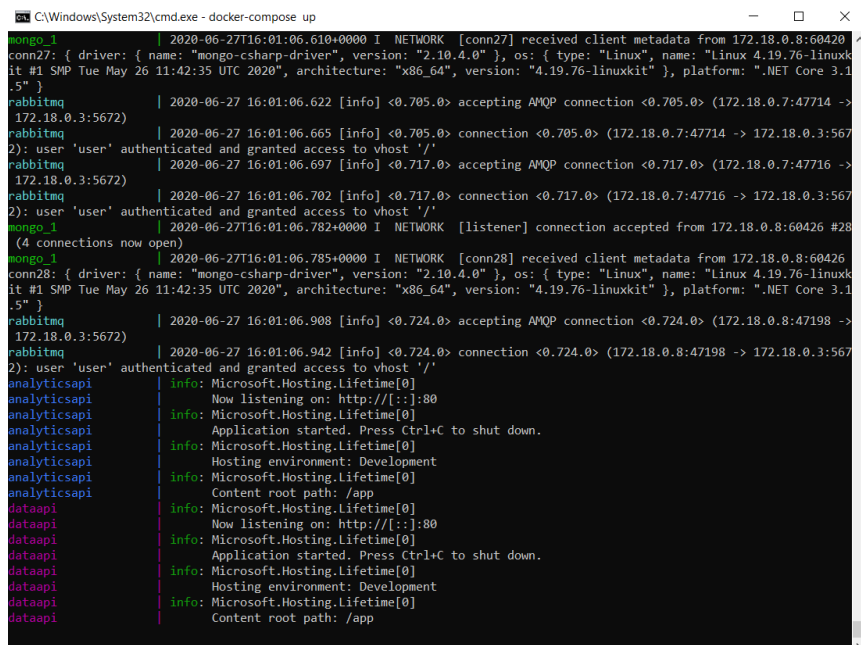
1. Klonirati projekat komandom **git clone** <https://github.com/milosveljkovic/Microserices.git>
2. Otvoriti **cmd** ili **shell** na putanji Microservices/src i ukucajte **docker-compose build**



Nakog build-a trebalo bi da vidite u konzoli poruku za uspešno izvršenu operaciju:



3. Nakon toga unutar iste konzole ukucajte **docker-compose up**  
U konzoli ćete imati nešto slično kao na sledećoj slici:

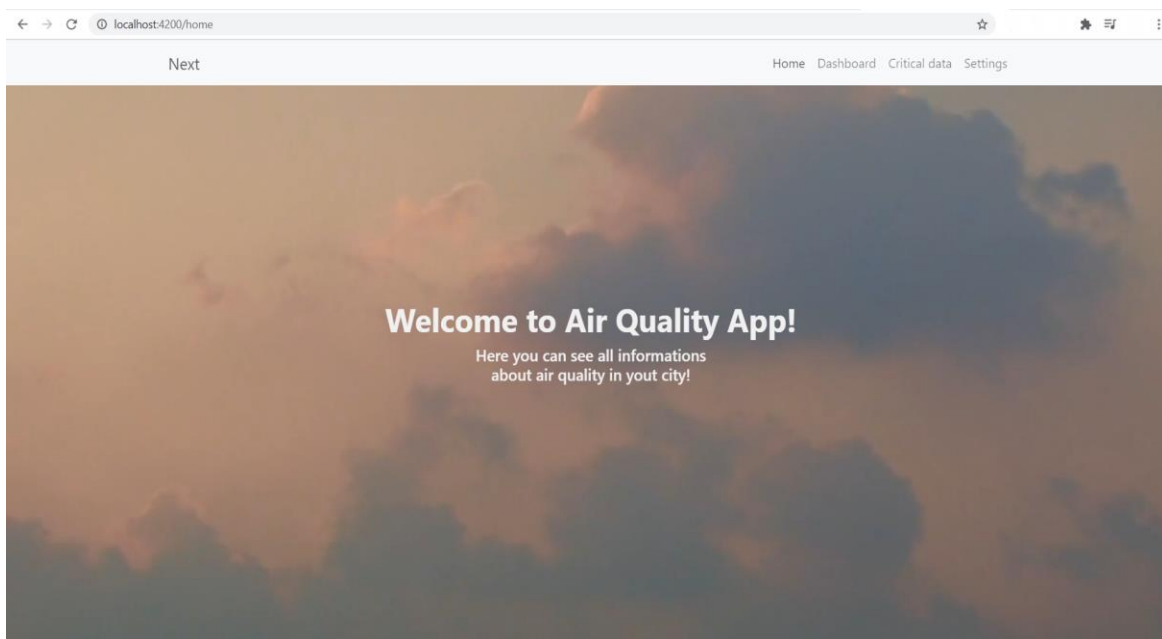


Nakon trećeg koraka, ukoliko je sve prošlo uspešno, startovali ste sve mikroservise.

Sada je potrebno starovati klijet aplikaciju tj WebDashboard.

1. Otvoriti WebDashboard folder u nekom IDE-u ili Code-editoru (npr. VSCode)  
(Ukoliko nemate ni jedan, sledeće korake možete izvršiti I u cmd-u / shell-u sa putanjom gde se nalazi WebDashboard)
2. Sada je potrebno instalirati sve neophodne zavisnosti
3. Izvršiti komandu ***npm install***
4. Startovanje aplikacije: ***npm run start***

Na adresi **localhost:4200** će vam biti otvoren **WebDashboard (Angular aplikacija)** koja bi trebalo da izgleda kao na sledećoj slici:



U nastavku ćemo objasniti ulogu svakog od servisa i dati detaljniji opis.

### **Device Microservice:**

Zadatak ovog mikroservisa je da čita podatke iz senzora ali isto tako da podešava parametre na samom senzoru (brzina čitanja, brzina slanja, treshold...). Deo ovog servisa je i 'čistač vazduha' koji je u aplikaciji predstavljen kao 'Mi Air Purfier' koji ima za zadatak da 'pročisti' vazduh. Device servis šalje pročitane podatke Data servisu preko RabbitMQ-a. Setovanje parametara očitavanja sa senzora ili samo podešavanje 'Mi Air Purfier'-a ide preko Command mikroservisa koji šalje komande Device-u.

Device Microservice obezbeđuje sledeći API:

- **POST - api/device/setSensorSendPeriod (u ms se šalje)**

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:5002/api/device/setSensorSendPeriod
- Body:**

```
{ 1: { 2: "periodValue": 15000 3: }
```
- Status:** 200 OK
- Response Body:**

```
1
```

- **POST - api/device/setSensorReadPeriod (u ms se šalje)**

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:5002/api/device/setSensorReadPeriod
- Body:**

```
1 { 2: "periodValue": 10000 3 }
```
- Status:** 200 OK

- **POST - api/device/turnOnOff (1 - turnOn, 0 - turnOff)**

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:5002/api/device/turnOnOff
- Body:**

```
1 1
```
- Status:** 200 OK

- **POST - api/device/setTreshold (min 10, max 50)**

POST

http://localhost:5002/api/device/setTreshold

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

20

Body

Cookies

Headers (3)

Test Results

Status: 200 OK

- **POST - api/device/turnOnOffMiAirPurifier (1- turnOn, 0- turnOff)**

POST

http://localhost:5002/api/device/turnOnOffMiAirPurifier

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

1

Body

Cookies

Headers (3)

Test Results

Status: 200 OK

- **POST - api/device/setMiAirPurifierCleaningStrength (min 10, max 50)**

POST

http://localhost:5002/api/device/setMiAirPurifierCleaningStrength

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

30

Body

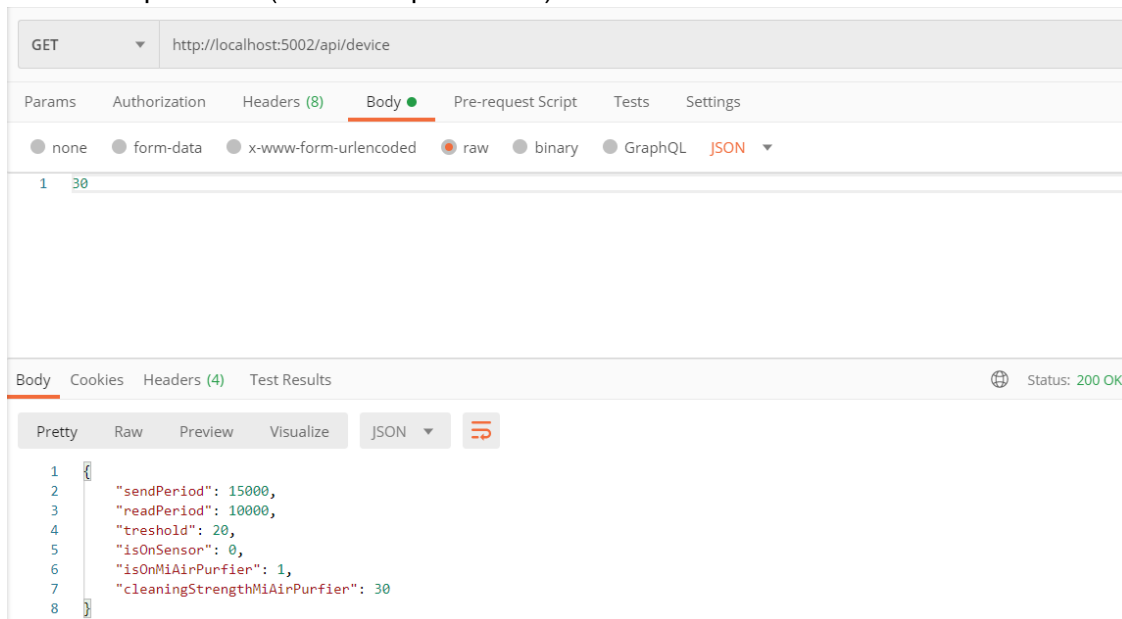
Cookies

Headers (3)

Test Results

Status: 200 OK

- **GET - api/device** (vraca sve parametre)

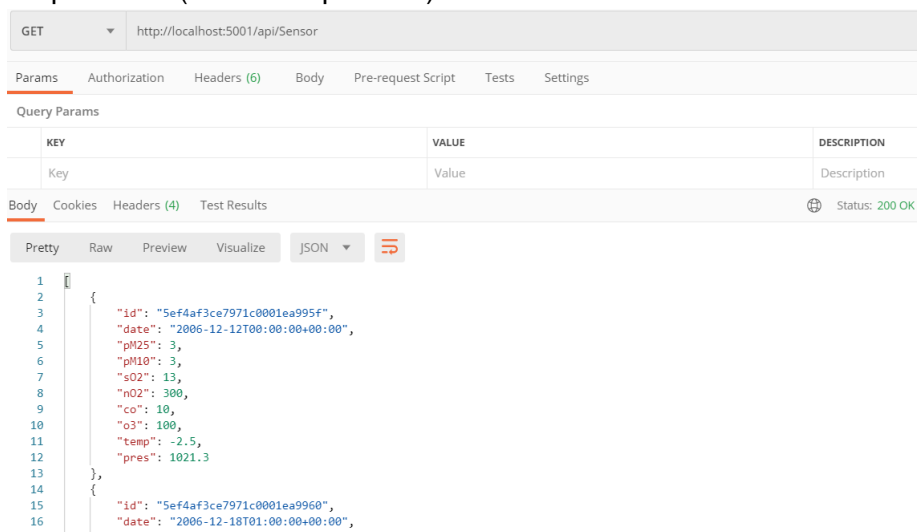


### Data Microservice:

Data Microservice je subskrajbovan na kanal “**device-data**”, ovde očekuje da primi podatke od Device mikroservisa kako bi upisao te podatke u svoju **NoSql Mongo** bazu. Takodje, po primanju podataka, publišuje podatke ka Analytics mikroservisu koji izvršava odredjene akcije u zavisnosti od podataka koje je primio. Takodje deo ovog servisa je da filtrira podatke u zavisnosti od ‘filtera’ koje je zahtevao korisnik na WebDashboard-a.

Data Microservice obezbedjuje sledeći API:

- **GET - /api/sensor** (vraća sve podatke)



- **POST** - /api/sensor/getSensorsBetweenDates (vraca podatke na osnovu filtera)

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:5001/api/Sensor/GetSensorsBetweenDates
- Body Tab:** Selected, showing a JSON body with filter criteria:
 

```
{
  "dateFrom": "2005-12-14",
  "dateTo": "2026-12-15",
  "PM25": 0,
  "PM10": 0,
  "SO2": 0,
  "NO2": 0,
  "CO": 0,
  "O3": 0
}
```

 A red box highlights the body, and the word "FILTERI" is written in red next to it.
- Response Tab:** Selected, showing a JSON response:
 

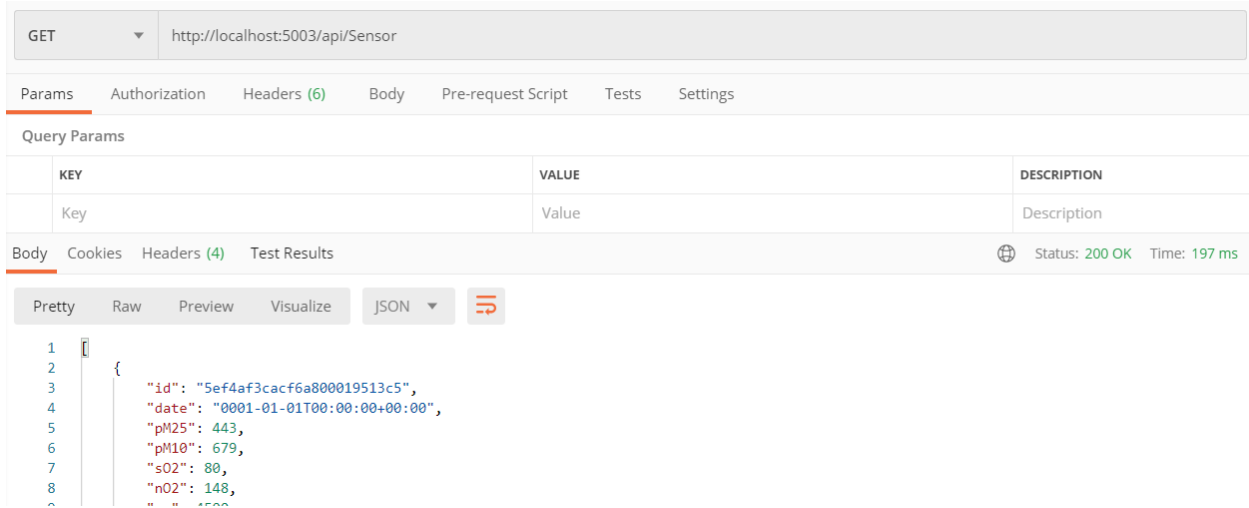
```
{
  "id": "5ef4af3ce7971c0001ea995f",
  "date": "2006-12-12T00:00:00+00:00",
  "PM25": 3,
  "PM10": 3,
  "SO2": 13,
  "NO2": 300,
  "CO": 10
}
```
- Status:** 200 OK

### **Analytics Microservice:**

Već je pomenuto da Analytics prima podatke od Data mikroservisa, što znači da je on subskrajbovan na kanal "**data-analytics**". Po primanju podataka, vrši analizu istih u cilju detektovanja kritične vrednosti za kvalitet vazduha(AQI). Ukoliko je primljeni senzorski podatak kritičan, upisuje se u **NoSql Mongo** bazu. Po upisu u bazu, šalje se upozorenje Command mikroservisu koji po primanju upozorenja izvršava odgovarajuće akcije. Takodje, po detektovanju 'kritičnog' događaja, šalje se '**Notification**' na **WebDashboard** uz pomoć **SignalR** biblioteke.

Analytics Microservice obezbeđuje sledeći API:

- **GET - api/sensor** (vraća podatke sa kritičnim vrednostima vazduha)



GET <http://localhost:5003/api/Sensor>

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (4) Test Results Status: 200 OK Time: 197 ms

Pretty Raw Preview Visualize JSON

```

1 {
2   "id": "5ef4af3cac6a800019513c5",
3   "date": "0001-01-01T00:00:00+00:00",
4   "pM25": 443,
5   "pM10": 679,
6   "sO2": 80,
7   "nO2": 148,
8   "CO": 1500

```

U nastavku je dat deo koda koji šalje Command mikroservisu '**Warning Notification**':

```

if (s.PM25 >= 91 && s.PM25 <= 120)
{
    WarningNotification w = new WarningNotification()
    {
        name = "PM 2.5",
        value = s.PM25,
        type = "Poor"
    };
    await PostRequest(urlController, w);
}

```

Deo koda koji salje WebDashboard-u **notifikaciju**:

```

if (s.PM25 > 250 || s.PM10 > 430 || s.O3 > 748 || s.CO > 3400)
{
    await _repository.Create(s);
    _messageHubContext.Clients.All.SendAsync("send", s);
}

```

### **Command Microservice:**

Osnovna uloga Command mikroservisa je da se preko njega prosledjuju komande za setovanje parametara sa WebDashboard-a ka **Device-u** ali isto tako se preko njega salje upozoravajuća akcija **Aktuatoru**.

Command Microservice obezbedjuje sledeći API:



- **POST - api/command/setSensorReadPeriod**

POST

http://localhost:5004/api/command/setSensorReadPeriod

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

● none

● form-data

● x-www-form-urlencoded

● raw

● binary

● GraphQL

JSON

1

{

2

"periodValue": 3000

3

}

Body

Cookies

Headers (3)

Test Results

Status: 200 OK

- **POST - api/command/setSensorSendPeriod**

POST

http://localhost:5004/api/command/setSensorSendPeriod

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

● none

● form-data

● x-www-form-urlencoded

● raw

● binary

● GraphQL

JSON

1

{

2

"periodValue": 4000

3

}

Body

Cookies

Headers (3)

Test Results

Status: 200 OK

- **POST - api/command/turnOnOff (turnOn - 1, turnOff - 0)**

POST

http://localhost:5004/api/command/turnOnOff

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

● none

● form-data

● x-www-form-urlencoded

● raw

● binary

● GraphQL

JSON

1

1

Body

Cookies

Headers (3)

Test Results

Status: 200 OK

- **POST - api/command/setTreshold**

POST http://localhost:5004/api/command/setTreshold

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

1 35

Body Cookies Headers (3) Test Results Status: 200 OK

- **POST - api/command/sendNotification (slanje akcije **Aktuatoru**)**

POST http://localhost:5004/api/command/sendNotification

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

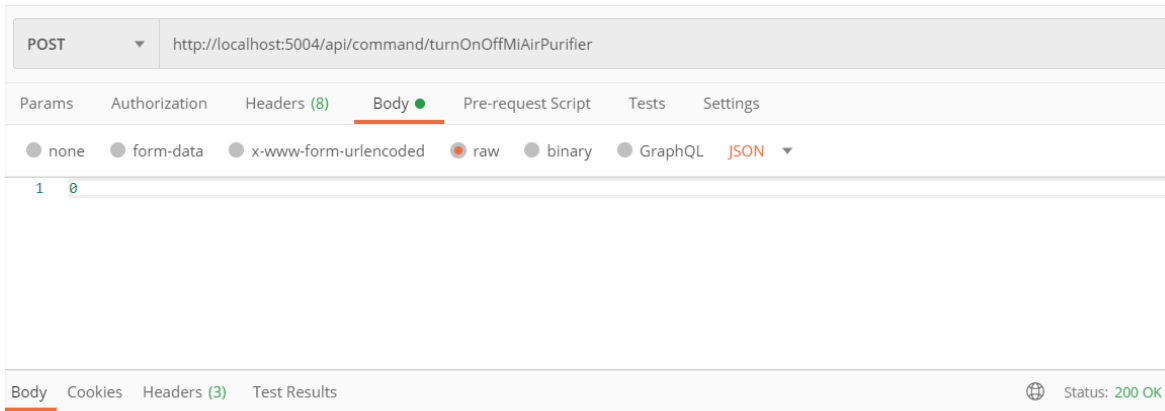
none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "name": "PM 2.5",
3   "value": 500,
4   "type": "Severe"
5 }
```

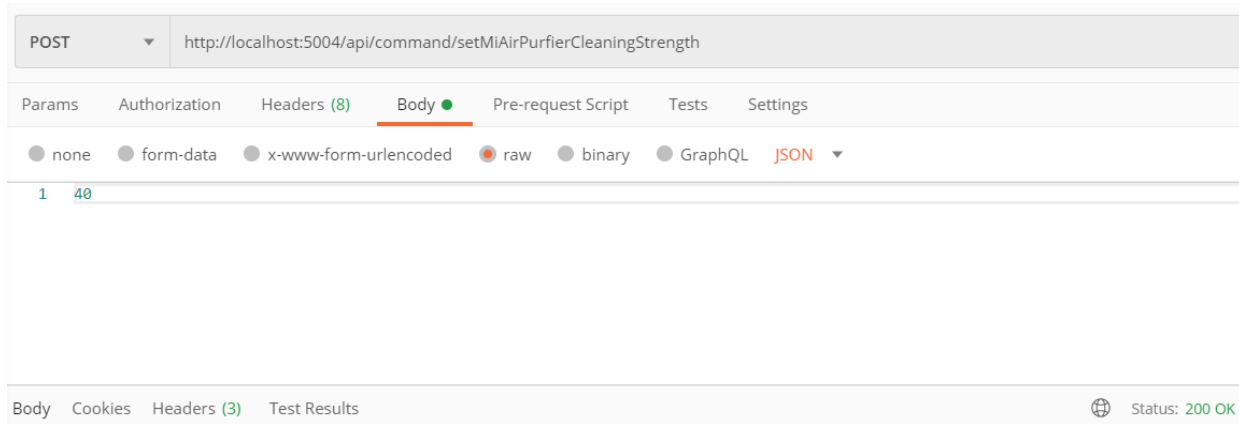
Body Cookies Headers (3) Test Results Status: 200 OK

```
2020-06-25T17:54:43.999+0000 I NETWORK [conn30] end connection
deviceapi | Read every 1 sec, just read--->Device.Entities.SensorData
mongo_1 | 2020-06-25T17:54:43.999+0000 I NETWORK [listener] connection a
mongo_1 | 2020-06-25T17:54:44.001+0000 I NETWORK [conn31] received client
pxkit #1 SMP Tue May 26 11:42:35 UTC 2020", architecture: "x86_64", version: "4.19.76-1
mongo_1 | 2020-06-25T17:54:44.175+0000 I NETWORK [listener] connection a
mongo_1 | 2020-06-25T17:54:44.177+0000 I NETWORK [conn32] received client
pxkit #1 SMP Tue May 26 11:42:35 UTC 2020", architecture: "x86_64", version: "4.19.76-1
aktuatorapi | [Severe Warning] PM 2.5 has 1000 - AQI [>401]
analyticsapi | OK
deviceapi | Read every 1 sec, just read--->Device.Entities.SensorData
aktuatorapi | [Severe Warning] PM 2.5 has 350 - AQI [>401]
analyticsapi | OK
deviceapi | Read every 1 sec, just read--->Device.Entities.SensorData
aktuatorapi | [Severe Warning] PM 2.5 has 500 - AQI [>401]
aktuatorapi | [Severe Warning] PM 2.5 has 500 - AQI [>401]
```

- **POST** - `api/command/turnOnOffMiAirPurifier` (turnOn -1, turnOff -0)



- **POST** - `api/command/setMiAirPurifierCleaningStrength` ('Mi Air Purifier' mora biti **uključen** da bi mogla da se setuje njegova 'snaga')



### **Aktuator Microservice:**

Aktuator je u projektu minimalistički servis koji ima zadatak samo da štampa upozorenja koja prima sa **Command** mikroservisa. Na sledećoj slici vidimo da je Aktuator primio '**Severe Warning**' od **Command** mikroservisa.

```

mongo_1 | 2020-06-25T17:44:00.103+0000 I NETWORK [conn30] end connection
deviceapi | Read every 1 sec, just read-->Device.Entities.SensorData
mongo_1 | 2020-06-25T17:54:43.999+0000 I NETWORK [listener] connection a
mongo_1 | 2020-06-25T17:54:44.001+0000 I NETWORK [conn31] received clie
uxkit #1 SMP Tue May 26 11:42:35 UTC 2020", architecture: "x86_64", version: "4.19.76-1
mongo_1 | 2020-06-25T17:54:44.175+0000 I NETWORK [listener] connection a
mongo_1 | 2020-06-25T17:54:44.177+0000 I NETWORK [conn32] received clie
uxkit #1 SMP Tue May 26 11:42:35 UTC 2020", architecture: "x86_64", version: "4.19.76-1
aktuatorapi | [Severe Warning] PM 2.5 has 1000 - AQI [>401]
analyticsapi | OK
deviceapi | Read every 1 sec, just read-->Device.Entities.SensorData
aktuatorapi | [Severe Warning] PM 2.5 has 350 - AQI [>401]
analyticsapi | OK
deviceapi | Read every 1 sec, just read-->Device.Entities.SensorData
aktuatorapi | [Severe Warning] PM 2.5 has 500 - AQI [>401]
aktuatorapi | [Severe Warning] PM 2.5 has 500 - AQI [>401]

```

### ApiGateway:

Ovaj deo sistema predstavlja ulaznu tačku za pristup svim mikroservisima u sistemu. Prosleđuje sve zahteve sa klijenta odgovarajućim mikroservisima. Prosleđivanje predstavlja mapiranje paterna zahteva. Primer 'prosleđivanja' je dat u nastavku, dok se sva mapiranja nalaze u fajlu **routes** u ApiGateway mikroservisu. Za implementaciju ovog mikroservisa korišćen je **Ocelot**.

```
{
  "DownstreamPathTemplate": "/api/sensor",
  "DownstreamScheme": "http",
  "DownstreamHostAndPorts": [
    {
      "Host": "dataapi",
      "Port": "80"
    }
  ],
  "UpstreamPathTemplate": "/dataservice/sensor",
  "UpstreamHttpMethod": [ "GET" ]
},
```