

Moleculer – microservices framework for Node.js

Moleculer je framework za kreiranje mikroservisa u **Node.js**-u. Zahvaljujući njemu, ne moramo da se brinemo previše o implementacionim detaljima, već možemo da se skoncentrišemo na razvoj biznis logike. Relativno je jednostavan za korišćenje, efikasan je i aktivno se razvija. Kao glavna prednost ovog frejmworka ističe se event-driven arhitektura. Oficijalnu dokumentaciju možete naći [OVDE](#).

Ključni koncepti

- **Servis** – je jednostavan JavaScript modul koji sadrži delove kompleksne aplikacije. Izolovan je od ostatka aplikacije i samostalan je. To znači da i za slučaj da taj servis "pukne" ostali servisi će nastaviti da rade nesmetano.
- **Node** – jednostavan OS proces koji se izvršava u lokalnoj ili eksternoj mreži. Jedinствена instanca node-a može da hostuje jedan ili više servisa.
- **Lokalni Servisi** – dva ili više servisa koja se izvršavaju na jednom čvoru se nazivaju lokalnim. Oni dele hardverske resurse i koriste lokalnu magistralu da komuniciraju međusobno (transporter se ne koristi).
- **Udaljeni Servisi** – su distribuirani preko većeg broja node-ova. Komunikacija između ovih servisa se odvija preko transportera.
- **Servis Broker** – je srce Moleculer aplikacije. Odgovoran je za upravljanje i komunikaciju između servisa (lokalnih i udaljenih). Svaki node mora da ima instancu servis brokera.
- **Transporter** – je komunikaciona magistrala koju servisi koriste da razmenjuju poruke. Prenosi eventove, zahteve i odgovore.
- **Gateway** – "predstavlja" Moleculer servise krajnjim korisnicima. Gateway je regularan Moleculer servis koji pokreće server (npr. HTTP). On prima dolazne zahteve, mapira ih na pozive servisa i vraća neophodne odgovore.

Primer 1 – Online store

Online store, arhitekturno gledano, može da bude implementiran korišćenjem dva nezavisna servisa – product servis i gateway servis. Prvi je odgovoran za čuvanje i upravljanje proizvodima, a drugi za primanje klijentskih zahteva i rutiranje tih zahteva ka products servisu. U kontekstu Moleculer-a, da bismo osigurali da je naš sistem otporan na otkaze, izvršavaćemo product i gateway servise na zasebnim node-ovima - node-1 i node-2, respektivno. Da bismo izvršavali servise na različitim node-ovima neophodan

nam je transporter koji će da obavlja komunikaciju između servisa. Većina transportera koji se mogu koristiti u Moleculer-u se oslanjaju na message brokere za međuservisnu komunikaciju (u ovom primeru će biti korišćen NATS).

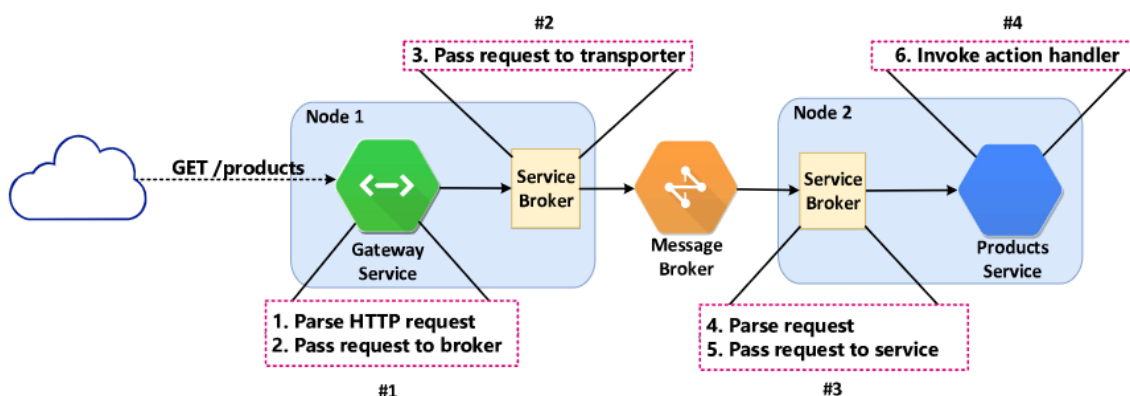
NATS je open-source sistem za razmenu poruka. Ključni principi koje podržava su dobre performanse, skalabilnost i lakoća korišćenja. Namenjen je za cloud native aplikacije, IoT messaging i mikroservisnu arhitekturu. Sastoji se od NATS servera (core publish-subscribe server za NATS), NATS Streaming-a (sistem za strimovanje podataka, koji omogućava perzistenciju, ponovno slanje poruka...) i klijentskih biblioteka (za različite programske jezike).

Korisni linkovi: [NATS](#), [NATS Documentation](#)

Ciklus pribavljanja proizvoda (slika 1)

Kada se kreiraju i pokrenu servisi i za pribavljanje proizvoda je neophodno poslati GET zahtev na /products, tok koji se dešava u pozadini kada se okine ovaj zahtev je sledeći:

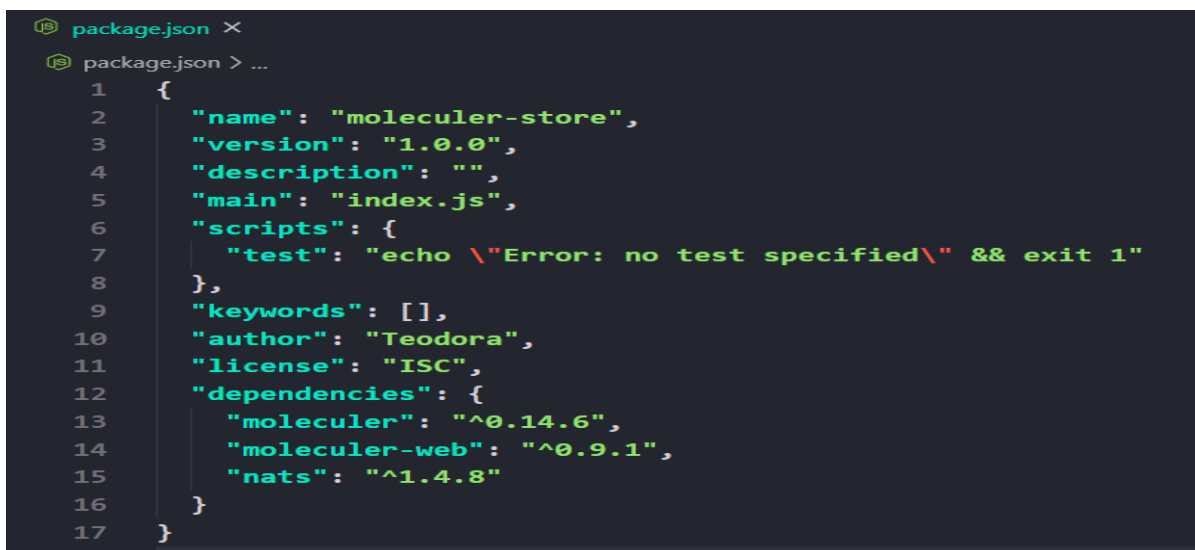
- HTTP server koji se izvršava na node-1 prima zahtev
- Zahtev se prenosi sa servera na gateway servis koji izvršava sva neophodna procesiranja
- Dalje se zahtev prenosi brokeru koji proverava da li je products servis lokalni ili udaljeni. S obzirom da se u ovom primeru radi o udaljenom servisu, potreban je transporter da se ovaj zahtev prosledi products servisu.
- Transporter pokuplja zahtev i prosleđuje ga preko komunikacione magistrale, a node-1 i node-2 su povezani preko message brokera, zahtev se dostavlja uspešno.
- Konačno, products servis okida neophodnu metodu, koja treba da se izvrši za dobijen zahtev.
- Odgovor se prosleđuje krajnjem korisniku.



Slika 1. Ciklus pribavljanja proizvoda na GET /products

Implementacija

- kreirati direktorijum (npr. Moleculer-store) – **mkdir moleculer-store**
- **cd moleculer-store**
- **npm init** (inicijalizacija projekta, da bismo odbrali sve default opcije možemo kao opciju navesti `npm init -y`)
- **npm install moleculer** (za kreiranje servisa) **moleculer-web** (kao HTTP gateway) **nats** (kao message broker)
- prikaz `package.json` fajla nakon ovog koraka je dat na slici 2.
- Sledeći korak je konfiguracija brokera – treba kreirati **index.js** fajl



```
package.json X
package.json > ...
1  {
2    "name": "moleculer-store",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \\\"Error: no test specified\\\" && exit 1"
8    },
9    "keywords": [],
10   "author": "Teodora",
11   "license": "ISC",
12   "dependencies": {
13     "moleculer": "^0.14.6",
14     "moleculer-web": "^0.9.1",
15     "nats": "^1.4.8"
16   }
17 }
```

Slika 2. `package.json` nakon `npm install`

Na slici 3 je prikazano kreiranje prvog mikroservisa. Ime mikroservisa je `gateway`, što ste postavljali navođenjem `name: "gateway"`. Nakon toga vidimo da je definisano `mixins:[HTTPServer]`. Mixins predstavljaju fleksibilan način za distribuiranje višestrukog korišćenja u Moleculer servisima. Konstruktor servisa spaja navedene mixin-e sa trenutnom šemom servisa. Vidimo, takođe, da je u `settings` delu definisan niz ruta koje naš servis nudi. On se, u ovom slučaju, samo sastoji od rute `GET /products` koja će se okinuti kada se uputi `GET` zahtev na `/products`. Kada se to desi treba izvršiti akciju `products.listProducts` koja će biti definisana u `Products` servisu koji je prikazan na slici 4.

Bitno je napomenuti da se prilikom kreiranja i `brokerNode1` i `brokerNode2` pored `nodeID`-ja koji se podešava, podešava i `transporter`. S obzirom da ovaj projekat pokrećemo samo lokalno, treba podesiti konfiguraciju transportera koji se koristi, tako da zna da se radi o `localhost`u, zato se navodi `nats://localhost:4222`.

```
1 // index.js
2 const { ServiceBroker } = require("moleculer");
3 const HTTPServer = require("moleculer-web");
4
5 // Create the broker for node-1
6 // Define nodeID and set the communication bus
7 const brokerNode1 = new ServiceBroker({
8   nodeID: "node-1",
9   transporter: "nats://localhost:4222"
10 });
11
12 // Create the "gateway" service
13 brokerNode1.createService({
14   // Define service name
15   name: "gateway",
16   // Load the HTTP server
17   mixins: [HTTPServer],
18
19   settings: {
20     routes: [
21       {
22         aliases: {
23           // When the "GET /products" request is made the "listProducts" action of "products"
24           // service is executed
25           "GET /products": "products.listProducts"
26         }
27       }
28     ]
29   }
30 });
31
```

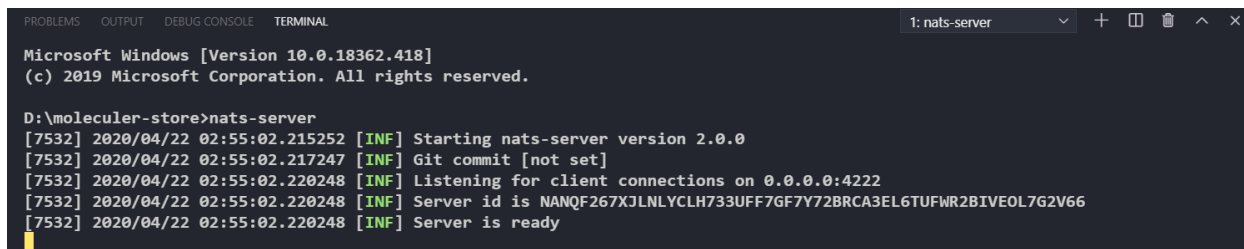
Slika 3. brokerNode1

```
33 // Create the broker for node-2
34 // Define nodeID and set the communication bus
35 const brokerNode2 = new ServiceBroker({
36   nodeID: "node-2",
37   transporter: "nats://localhost:4222"
38 });
39
40 // Create the "products" service
41 brokerNode2.createService({
42   // Define service name
43   name: "products",
44
45   actions: {
46     // Define service action that returns the available products
47     listProducts(ctx) {
48       return [
49         { name: "Apples", price: 5 },
50         { name: "Oranges", price: 3 },
51         { name: "Bananas", price: 2 }
52       ];
53     }
54   }
55 });
56
57 // Start both brokers
58 Promise.all([brokerNode1.start(), brokerNode2.start()]);
59
```

Slika 4. brokerNode2 i startovanje brokera

Testiranje

Pre pokretanja i testiranja online store aplikacije treba pokrenuti nats. To je moguće učiniti komandom **nats-server**.

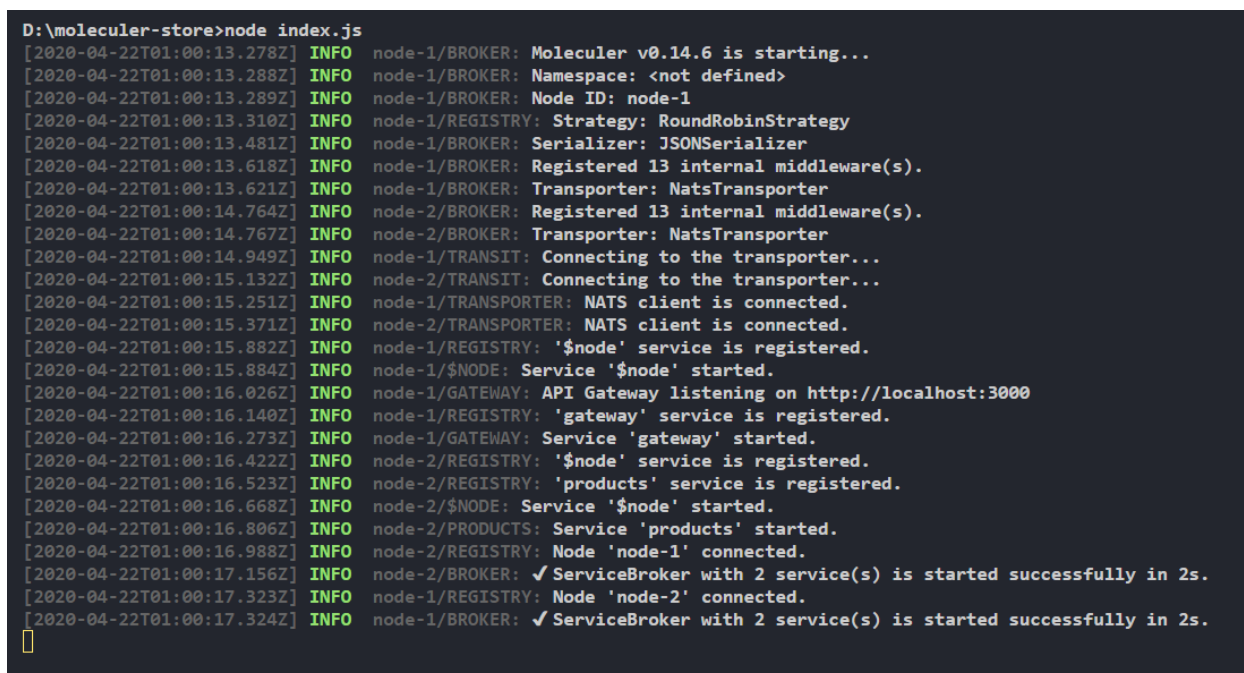


```
Microsoft Windows [Version 10.0.18362.418]
(c) 2019 Microsoft Corporation. All rights reserved.

D:\moleculer-store>nats-server
[7532] 2020/04/22 02:55:02.215252 [INF] Starting nats-server version 2.0.0
[7532] 2020/04/22 02:55:02.217247 [INF] Git commit [not set]
[7532] 2020/04/22 02:55:02.220248 [INF] Listening for client connections on 0.0.0.0:4222
[7532] 2020/04/22 02:55:02.220248 [INF] Server id is NANQF267XJLNLYCLH733UFF7GF7Y72BRCA3EL6TUFWR2BIVEOL7G2V66
[7532] 2020/04/22 02:55:02.220248 [INF] Server is ready
```

Slika 5. Pokretanje nats brokera

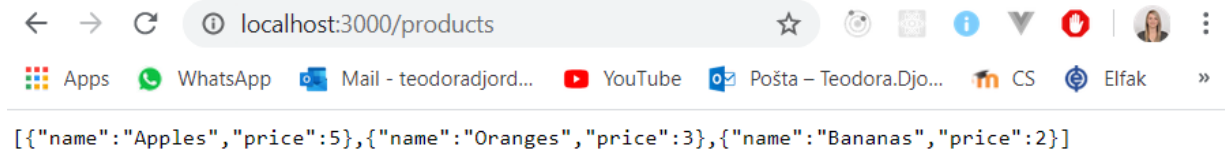
Sledi pokretanje aplikacije komandom **node index.js**. Nakon ovog koraka aplikaciju je moguće testirati na <http://localhost:3000/products>.



```
D:\moleculer-store>node index.js
[2020-04-22T01:00:13.278Z] INFO node-1/BROKER: Moleculer v0.14.6 is starting...
[2020-04-22T01:00:13.288Z] INFO node-1/BROKER: Namespace: <not defined>
[2020-04-22T01:00:13.289Z] INFO node-1/BROKER: Node ID: node-1
[2020-04-22T01:00:13.310Z] INFO node-1/REGISTRY: Strategy: RoundRobinStrategy
[2020-04-22T01:00:13.481Z] INFO node-1/BROKER: Serializer: JsonSerializer
[2020-04-22T01:00:13.618Z] INFO node-1/BROKER: Registered 13 internal middleware(s).
[2020-04-22T01:00:13.621Z] INFO node-1/BROKER: Transporter: NatsTransporter
[2020-04-22T01:00:14.764Z] INFO node-2/BROKER: Registered 13 internal middleware(s).
[2020-04-22T01:00:14.767Z] INFO node-2/BROKER: Transporter: NatsTransporter
[2020-04-22T01:00:14.949Z] INFO node-1/TRANSIT: Connecting to the transporter...
[2020-04-22T01:00:15.132Z] INFO node-2/TRANSIT: Connecting to the transporter...
[2020-04-22T01:00:15.251Z] INFO node-1/TRANSPORTER: NATS client is connected.
[2020-04-22T01:00:15.371Z] INFO node-2/TRANSPORTER: NATS client is connected.
[2020-04-22T01:00:15.882Z] INFO node-1/REGISTRY: '$node' service is registered.
[2020-04-22T01:00:15.884Z] INFO node-1/$NODE: Service '$node' started.
[2020-04-22T01:00:16.026Z] INFO node-1/GATEWAY: API Gateway listening on http://localhost:3000
[2020-04-22T01:00:16.140Z] INFO node-1/REGISTRY: 'gateway' service is registered.
[2020-04-22T01:00:16.273Z] INFO node-1/GATEWAY: Service 'gateway' started.
[2020-04-22T01:00:16.422Z] INFO node-2/REGISTRY: '$node' service is registered.
[2020-04-22T01:00:16.523Z] INFO node-2/REGISTRY: 'products' service is registered.
[2020-04-22T01:00:16.668Z] INFO node-2/$NODE: Service '$node' started.
[2020-04-22T01:00:16.806Z] INFO node-2/PRODUCTS: Service 'products' started.
[2020-04-22T01:00:16.988Z] INFO node-2/REGISTRY: Node 'node-1' connected.
[2020-04-22T01:00:17.156Z] INFO node-2/BROKER: ✓ ServiceBroker with 2 service(s) is started successfully in 2s.
[2020-04-22T01:00:17.323Z] INFO node-1/REGISTRY: Node 'node-2' connected.
[2020-04-22T01:00:17.324Z] INFO node-1/BROKER: ✓ ServiceBroker with 2 service(s) is started successfully in 2s.
```

Slika 6. node index.js

Na sledećoj slici je prikazano kako izgleda u browseru GET zahtev na [/products](http://localhost:3000/products).



Slika 7. GET zahtev na /products

Primer 2 – pokretanje mikroservisa u zasebnim docker kontejnerima

Drugi primer će obuhvatiti kreiranje većeg broja servisa, gde će se svaki od servisa izvršavati u zasebnom kontejneru. Kao message broker će takođe biti korišćen NATS.

Prvi servis će biti gateway koji treba da prihvata zahteve na registrovanim rutama, drugi servis će čuvati i upravljati filmovima, a treći servis će imati registrovane event-ove koji će "slati mejl" kada se neki događaj koji to zahteva okine.

Implementacija

- pokretanje komande za inicijalizaciju projekta - npm init
- npm install --save moleculer nats express body-parser
- kreiranje direktorijuma services i kreiranje fajlova email.service.js, movies.service.js i gateway.service.js u tom direktorijumu
- kreiranje Dockerfile-a u glavnom direktorijumu
-

```
Dockerfile > ...
1 FROM node:latest
2 RUN mkdir /app
3 WORKDIR /app
4 ADD package*.json /app/
5 RUN npm install
6 ADD . /app/
7 CMD [ "npm", "start" ]
```

Slika 8. Dockerfile

Dockerfile definiše da je base image node:latest, tj. najnovija slika za node, kopira package.json i package-lock.json fajlove u direktorijum app koji je kreiran i izvršava npm install da bi instalirao sve neophodne zavisnosti na osnovu package.json fajla. Instrukcija za pokretanje je npm start. Da bismo znali šta se zapravo izvršava pokretanjem npm start, u package.json fajlu treba da bude definisano kako se projekat startuje.

```
"scripts": {  
  "dev": "moleculer-runner --repl --hot services",  
  "start": "moleculer-runner",  
  "test": "echo \"Error: no test specified\" && exit 1"  
},
```

Slika 9. package.json - scripts

npm start pokreće moleculer-runner komandu. Moleculer-runner je pomoćna skripta u Moleculer projektu. Pomoću nje ne moramo da kreiramo instancu ServiceBroker-a sa opcijama, već je dovoljno kreirati moleculer.config.js fajl u glavnom direktorijumu projekta gde ćemo postaviti konfiguraciju projekta ili fajl koji će sadržati environment promenljive.

Dev skripta nam je potrebna ukoliko želimo da testiramo projekat u lokalu pozivom komande npm run dev. Ovde su dodate opcije --repl znači da se koristi moleculer-repl (interaktivna konzola za razvoj moleculer projekata – treba je prvo instalirati komandom npm install moleculer-repl). Pored ovoga dodata je opcija --hot koja obezbeđuje hot reload servisa kada dođe do promene. Pored ovih opcija postoje i druge koje mogu da se iskoriste ([moleculer-runner](#)). Iza opcija se navode ili fajlovi u kojima se nalaze servisi ili direktorijum u kome su smešteni svi servisi. U našem slučaju radi se o services direktorijumu. Treba napomenuti da pošto se radi o lokalnim servisima, korišćenje transportera u lokalu nije potrebno.

Na sledećoj slici je prikazano kako izgleda konzola (sve komande koje je moguće izvršiti mogu se naći [ovde](#)).

mol \$ **services**

| Service | Version | State | Actions | Events | Nodes |
|---------|---------|-------|---------|--------|-------|
| \$node | - | OK | 7 | 0 | 1 |
| email | - | OK | 0 | 1 | 1 |
| gateway | - | OK | 0 | 0 | 1 |
| movies | - | OK | 3 | 0 | 1 |

mol \$

Slika 10. Primer komande koju je moguće pozvati uz pomoć moleculer-repl

- dodavanje .dockerignore fajla za node_modules
- implementacija servisa – gateway.service.js, email.service.js i movies.service.js redom

```

services > gateway.service.js > ...
1  "use strict";
2  const express = require("express");
3  const bodyParser = require('body-parser');
4
5  module.exports = {
6    name: "gateway",
7    settings: {
8      port: process.env.PORT || 3000,
9    },
10   methods: {
11     initRoutes(app) {
12       app.get("/movies", this.getMovies);
13       app.get("/movies/:id", this.getMovie);
14       app.post("/movies", this.createMovie);
15     },
16     getMovies(req, res) {
17       return Promise.resolve()
18         .then(() => {
19           return this.broker.call("movies.listAll").then(movies => {
20             res.send(movies);
21           });
22         })
23       .catch(this.handleError(res));
24     },
25   },
26 };

```

Slika 11. gateway.service.js prvi deo

- Definisanje našeg servisa se obavlja eksportovanjem objekta prema određenoj šemi, koji sadrži veliki broj atributa. Pre sve sadrži ime servisa. Može da sadrži i verziju našeg servisa ukoliko je našem projektu potrebno verzionisanje. Settings predstavlja objekat koji sadrži globalne promenljive koje su potrebne servisu (npr. broj porta na kom će servis slušati zahteve). Pored toga, gateway servis treba da implementira metode koje će da se pozivaju prilikom dobijanja zahteva. To je obavljeno u methods delu, gde pored ovih metoda, treba implementirati i initRoutes metodu. Ovaj metod u okviru express aplikacije registruje sve rute na koje je moguće poslati neki od zahteva. Pored metoda, servis može da sadrži i eventove (email.service.js) i akcije (movies.service.js), što će biti prikazano u narednim servisima.

```

25     getMovie(req, res) {
26         const id = req.params.id;
27         return Promise.resolve()
28             .then(() => {
29             return this.broker.call("movies.getById", {id: id}).then(movie => {
30                 res.send(movie);
31             });
32         })
33         .catch(this.handleError(res));
34     },
35     createMovie(req, res) {
36         const payload = req.body;
37         return Promise.resolve()
38             .then(() => {
39             return this.broker.call("movies.create", { payload }).then(movie =>
40                 res.send(movie)
41             );
42         })
43         .catch(this.handleError(res));
44     },
45     handleError(res) {
46         return err => {
47             res.status(err.code || 500).send(err.message);
48         };
49     }
50 },

```

Slika 12. gateway.service.js drugi deo

Što se tiče životnog ciklusa servisa, razlikujemo sledeće metode koje odgovaraju njegovim mogućim stanjima: created, started i stopped. Da bismo zapravo kreirali i startovali našu express aplikaciju potrebna nam je implementacija created metode. Created metoda se okida kada se kreira instanca servisa. Started handler ser pokreće kada broker startuje servis, a stopped se pokreće kada broker zaustavlja rad servisa.

```

51     created() {
52         const app = express();
53         app.use(bodyParser.urlencoded({ extended: false }));
54         app.use(bodyParser.json());
55         app.listen(this.settings.port);
56         this.initRoutes(app);
57         this.app = app;
58     }

```

Slika 13. created() metod

U okviru ove metode se prvo kreira express aplikacija, zatim se podesi da koristi body-parser kao middleware koji će da se pokreće pre pozivanja hendlera i koji će da omogući da se sve što je prosleđeno u okviru zahteva nađe u req.body. Nakon toga treba podesiti na kom portu će naša aplikacija slušati za zahteve, kao i na kojim rutama se mogu slati zahtevi pozivom funkcije initRoutes.

```

services > movies.service.js > ...
1  "use strict";
2
3  const movies = [
4      {id: 1, title: 'Sharknado'},
5      {id: 2, title: 'Roma'},
6  ];
7
8  module.exports = {
9      name: "movies",
10
11     actions: {
12         listAll(ctx) {
13             return Promise.resolve({ movies: movies });
14         },
15         getById(ctx) {
16             const id = Number(ctx.params.id);
17             return Promise.resolve(movies.find(movie => movie.id === id));
18         },
19         create(ctx) {
20             const lastId = Math.max(...movies.map(movie => movie.id));
21             const movie = {
22                 id: lastId + 1,
23                 ...ctx.params.payload,
24             };
25             movies.push(movie);
26             this.broker.emit("movie.created", movie);
27             return Promise.resolve(movie);
28         }
29     },
30 };

```

Slika 14. movies.service.js

Movies servis obezbeđuje čuvanje filmova i definiše akcije koje je moguće pozvati. Radi se o sledećim akcijama: za čitanje svih filmova, za čitanje jednog filma po id-ju i za kreiranje novog filma. Čuvanje filmova je u ovom servisu "mokovano" (ovde bismo dodali čitanje i upisivanje filmova u bazu).

```
services > email.service.js > ...
1  "use strict";
2
3  module.exports = {
4    name: "email",
5    events: {
6      "movie.created": {
7        group: "other",
8        handler(payload) {
9          console.log('Recieved "movie.created" event in email service with payload: ', payload);
10        }
11      }
12    },
13  };|
```

Slika 15. email.service.js

Email servis "sluša" kada dođe do nekog eventa (u ovom slučaju radi se o eventu movie.created). Kada detektuje da je došlo do ovog događaja izvršava se odgovarajući handler gde je payload ono što je prosleđeno prilikom okidanja događaja. Događaj se registruje u grupi other, a ne u email grupi.

```
this.broker.emit("movie.created", movie);
```

U Movies servisu, prilikom dodavanja novog filma se emituje ovaj događaj, gde se kao payload događaja prosleđuje film. Kao prvi parametar se navodi ime događaja. Moguće je kao treći parametar navesti i grupu koja treba da primi ovaj događaj.

Kada se pozove akcija ili emituje event, broker kreira instancu context-a koja sadrži sve informacije o zahtevu i prosleđuje ih akciji ili event handler-u kao jedini argument.

- treba dodati i config fajl u glavnom direktorijumu koji sadrži konfiguraciju projekta

```
moleculer.config.js > ...
1  "use strict";
2  const os = require("os");
3
4  module.exports = {
5    nodeID: (process.env.NODEID ? process.env.NODEID + "-" : "") + os.hostname().toLowerCase(),
6    // metrics: true
7    // cachet: true
8  };|
```

Slika 16. moleculer.config.js

U ovom fajlu se pored nodeID, može uključiti automatsko keširanje, prikupljanje podataka, ali i mnoge druge opcije.

Da bismo implementirane servise pokrenuli u zasebnim kontejnerima iskoristićemo docker-compose. Pre kreiranja ovog fajla kreiraćemo docker-compose.env fajl koji će da sadrži neophodne environment promenljive.

```
docker-compose.env
1 LOGLEVEL=info
2 TRANSPORTER=nats://nats:4222
3 SERVICEDIR=services
```

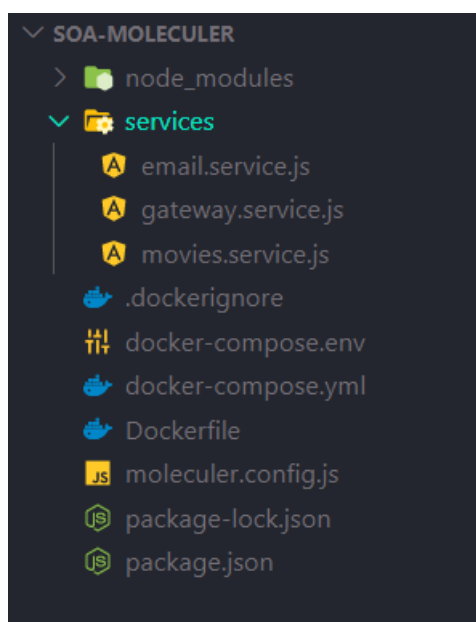
Slika 17. docker-compose.env

Ovde je podešeno u kom direktorijum se nalaze servisi, kao i podešavanje transportera. Pošto koristimo nats treba prvo navesti nats://, a nakon toga treba navesti ime pod kojim je naveden u docker-compose fajlu – to će u našem slučaju isto biti nats, kao i broj porta.

```
docker-compose.yml
1 version: '3.0'
2 services:
3   nats:
4     image: nats:latest
5   gateway:
6     build:
7       context: .
8     image: service-gateway
9     env_file: docker-compose.env
10    environment:
11      NODEID: "node-gateway"
12      SERVICES: gateway
13      PORT: 3000
14    ports:
15      - "3000:3000"
16    depends_on:
17      - nats
18    email:
19      build:
20        context: .
21      env_file: docker-compose.env
22      environment:
23        NODEID: "node-email"
24        SERVICES: email
25      depends_on:
26        - nats
27    movies:
28      build:
29        context: .
30      env_file: docker-compose.env
31      environment:
32        NODEID: "node-movies"
33        SERVICES: movies
34      depends_on:
35        - nats
```

Slika 18. docker-compose.yml

- docker-compose.yml definiše servise koji čine našu aplikaciju i koji će se izvršavati zajedno. Potreban nam je nats koji se kreira na osnovu slike nats:latest. Svaki od ostalih servisa zavisi od nats-a tako da treba staviti depends_on opciju. Za ostale servise se navodi naziv slike, na osnovu kog dockerfile-a (Dockerfile iz glavnog direktorijuma, zato je dovoljno samo navesti ., jer se docker-compose.yml nalazi u istom direktorijumu) se pravi ta slika, environment fajl koji smo prethodno definisali i još neke promenljive koje su neophodne. Vrlo je važno navesti SERVICES promenljivu kojoj ćemo dodeliti ime odgovarajućeg mikroservisa da bi se znalo koji mikroservis treba pokrenuti. Za gateway se dodatno specificira i broj porta.
- Struktura projekta je prikazana na sledećoj slici.

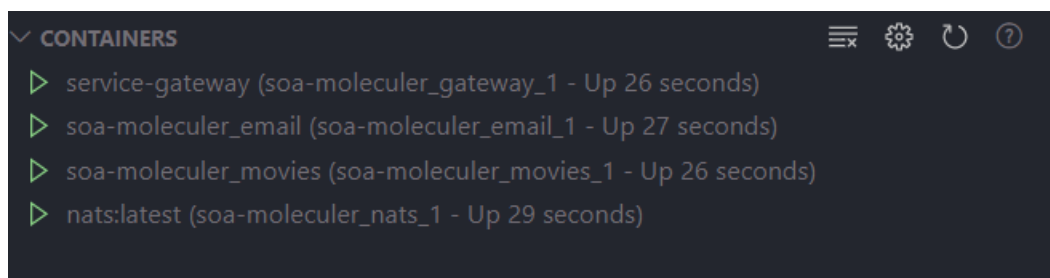


Slika 19. Struktura projekta

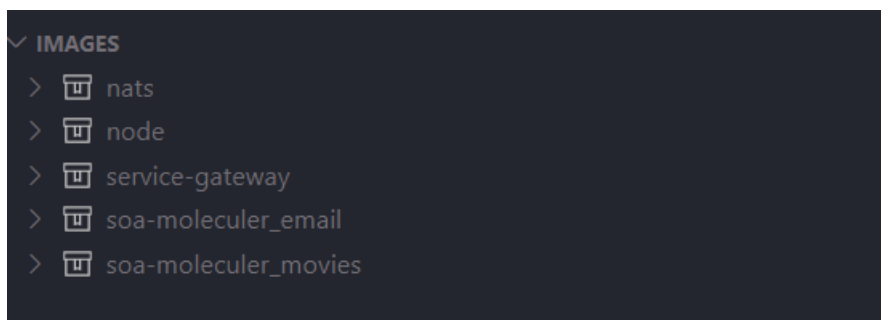
- Izvršavanje docker-compose up --build komande

```
movies_1 | [2020-04-22T02:27:55.402Z] INFO node-movies/BROKER: Molecular v0.14.6 is starting...
movies_1 | [2020-04-22T02:27:55.406Z] INFO node-movies/BROKER: Namespace: <not defined>
movies_1 | [2020-04-22T02:27:55.406Z] INFO node-movies/BROKER: Node ID: node-movies
movies_1 | [2020-04-22T02:27:55.410Z] INFO node-movies/REGISTRY: Strategy: RoundRobinStrategy
movies_1 | [2020-04-22T02:27:55.416Z] INFO node-movies/BROKER: Serializer: JSONSerializer
gateway_1 | [2020-04-22T02:27:55.450Z] INFO node-gateway/BROKER: Molecular v0.14.6 is starting...
gateway_1 | [2020-04-22T02:27:55.454Z] INFO node-gateway/BROKER: Namespace: <not defined>
gateway_1 | [2020-04-22T02:27:55.816Z] INFO node-gateway/TRANSPORTER: NATS client is connected.
gateway_1 | [2020-04-22T02:27:55.835Z] INFO node-gateway/REGISTRY: Node 'node-email' connected.
movies_1 | [2020-04-22T02:27:56.127Z] INFO node-movies/REGISTRY: '$node' service is registered.
movies_1 | [2020-04-22T02:27:56.131Z] INFO node-movies/REGISTRY: 'movies' service is registered.
movies_1 | [2020-04-22T02:27:56.133Z] INFO node-movies/$NODE: Service '$node' started.
movies_1 | [2020-04-22T02:27:56.133Z] INFO node-movies/MOVIES: Service 'movies' started.
movies_1 | [2020-04-22T02:27:56.140Z] INFO node-movies/BROKER: ✓ ServiceBroker with 2 service(s) is started successfully
in 627ms.
gateway_1 | [2020-04-22T02:27:56.168Z] INFO node-gateway/REGISTRY: Node 'node-movies' connected.
email_1 | [2020-04-22T02:27:56.172Z] INFO node-email/REGISTRY: Node 'node-movies' connected.
gateway_1 | [2020-04-22T02:27:56.352Z] INFO node-gateway/REGISTRY: '$node' service is registered.
gateway_1 | [2020-04-22T02:27:56.353Z] INFO node-gateway/REGISTRY: 'gateway' service is registered.
gateway_1 | [2020-04-22T02:27:56.353Z] INFO node-gateway/$NODE: Service '$node' started.
gateway_1 | [2020-04-22T02:27:56.353Z] INFO node-gateway/GATEWAY: Service 'gateway' started.
gateway_1 | [2020-04-22T02:27:56.361Z] INFO node-gateway/BROKER: ✓ ServiceBroker with 2 service(s) is started successfully
in 637ms.
movies_1 | [2020-04-22T02:27:56.365Z] INFO node-movies/REGISTRY: Node 'node-gateway' connected.
email_1 | [2020-04-22T02:27:56.363Z] INFO node-email/REGISTRY: Node 'node-gateway' connected.
```

Slika 20. Deo rezultata izvršenja docker-compose up komande



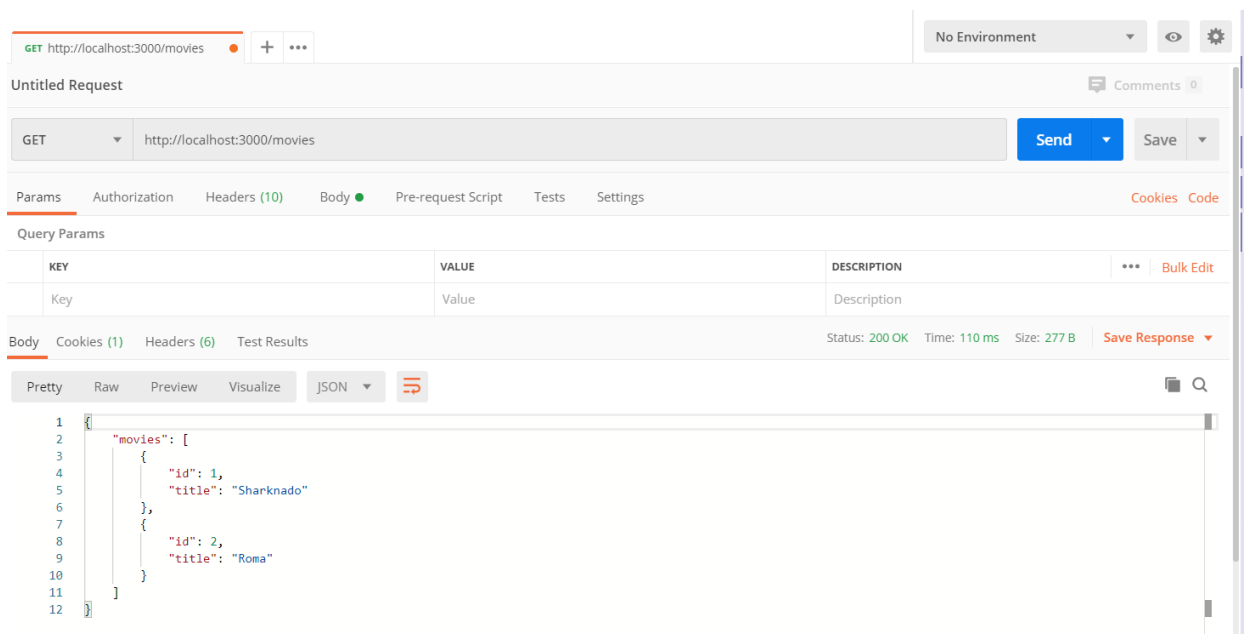
Slika 21. Kreirani kontejneri



Slika 22. docker images

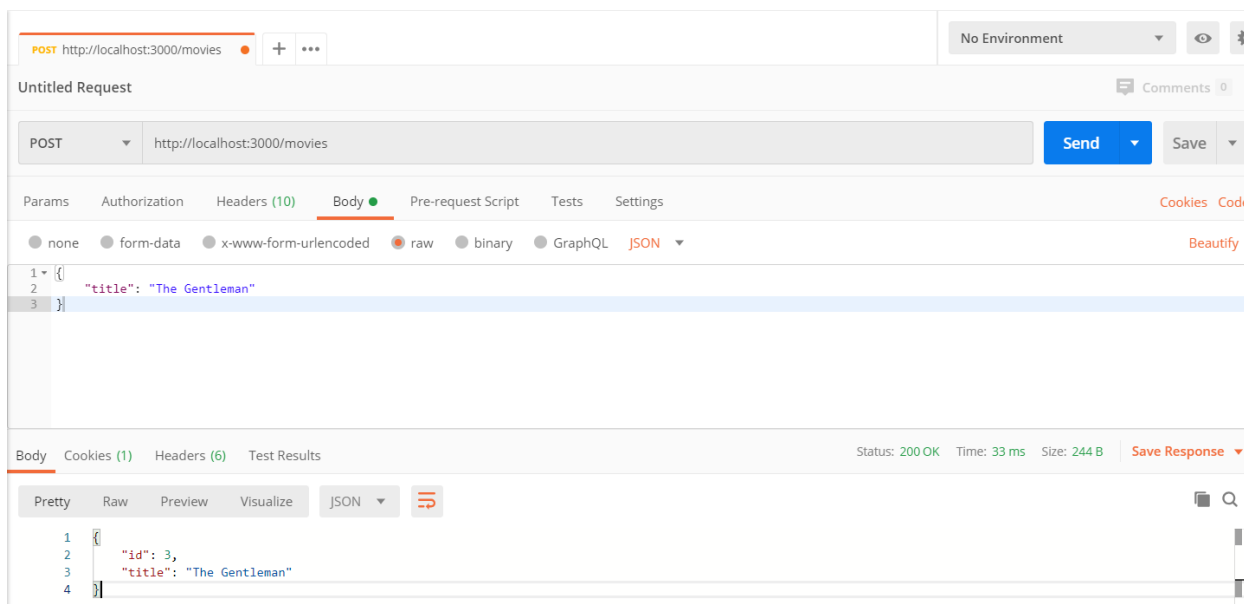
Testiranje aplikacije korišćenjem Postman-a

GET na /movies



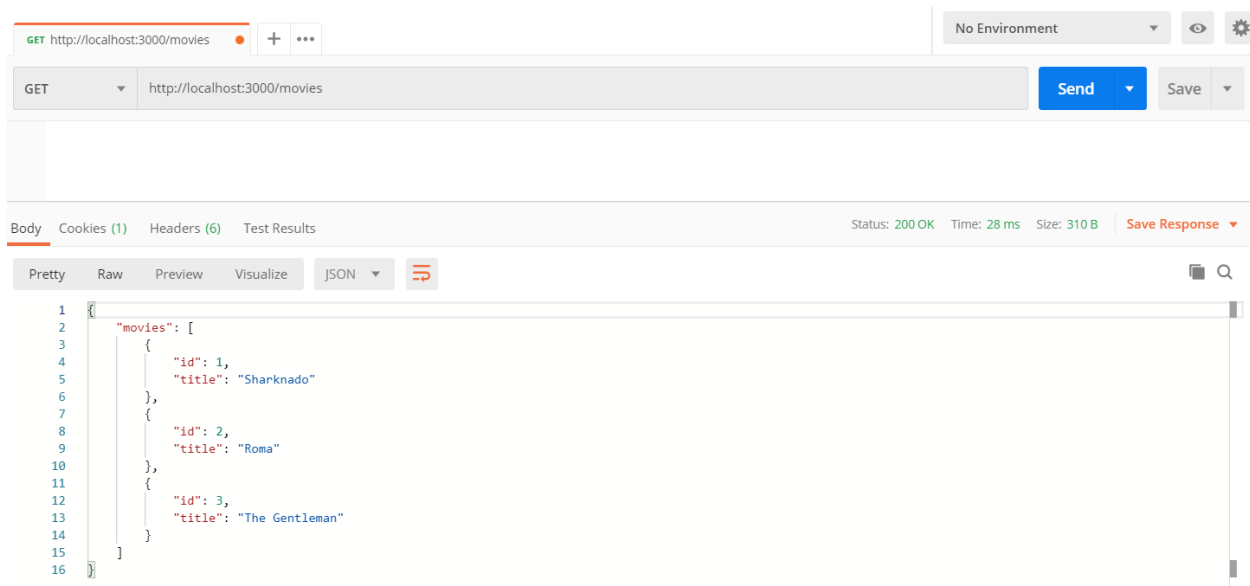
Slika 23. Čitanje svih filmova

POST na /movies



Slika 24. Kreiranje novog filma

GET na /movies nakon dodavanja novog filma



Slika 25. Prikaz svih filmova nakon dodavanja

Github repozitorijum u kome se nalazi kod možete naći [OVDE](#).