

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](#).

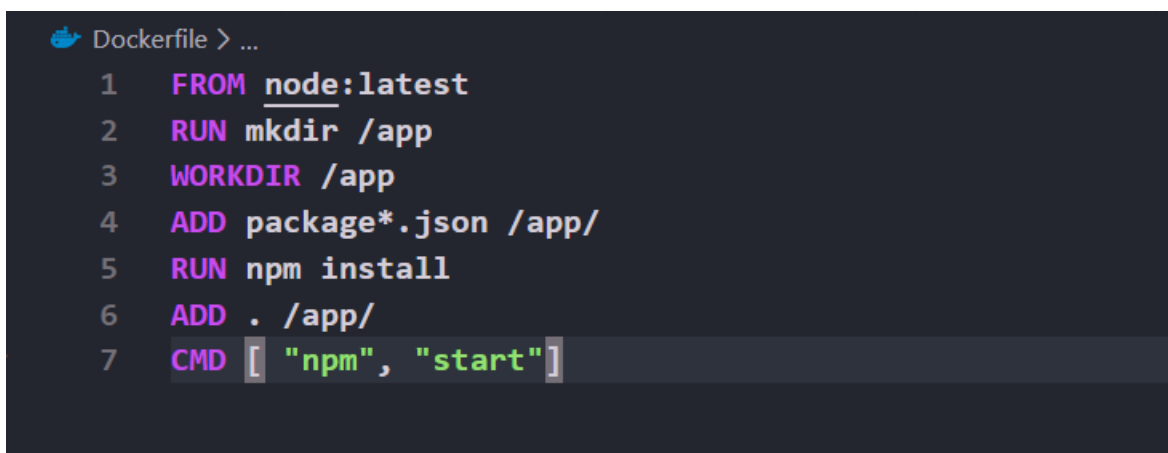
Moleculer – pokretanje mikroservisa u zasebnim docker kontejnerima

Ovaj 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 1. 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 2. 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 3. 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

```
moleculer-intro > services > gateway.service.js > <unknown> > methods > createMovie
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.post("/movies", this.createMovie);
14     },
15     getMovies(req, res) {
16       return Promise.resolve()
17         .then(() => {
18           return this.broker.call("movies.list").then(movies => {
19             res.send(movies);
20           });
21         })
22         .catch(this.handleError(res));
23   },
```

Slika 4. 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.

```

24     createMovie(req, res) {
25         const payload = req.body;
26         return Promise.resolve()
27             .then(() => {
28             return this.broker.call("movies.create", { movie: payload }).then(movie =>
29                 res.send(movie)
30             );
31         })
32         .catch(this.handleError(res));
33     },
34     handleError(res) {
35         return err => {
36             res.status(err.code || 500).send(err.message);
37         };
38     }
39 },

```

Slika 5. 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 se 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 6. created() metod

U okviru ove metode se prvo kreira express aplikacija, zatim se podese da koristi body-parser kao middleware koji će da se pokreće pre pozivanja hendlera i koji će da omogućiti 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.

Movies servis će za čuvanje servisa koristiti MongoDB, pa je neophodno kreirati mixin koji ćemo importovati u naš movie servis i pozivati njegove metode. Mixin će biti kreiran u mixin folderu u fajlu - db.mixin.js. Sadržaj ovog fajla je prikazan u nastavku.

```
moleculer-intro > mixins > db.mixin.js > <unknown> > module.exports
1  "use strict";
2
3  const path = require("path");
4  const mkdir = require("mkdirp").sync;
5
6  const DbService = require("moleculer-db");
7
8  module.exports = (collection) => {
9    if (process.env.MONGO_URL) {
10      const MongoAdapter = require("moleculer-db-adapter-mongo");
11
12      return {
13        mixins: [DbService],
14        adapter: new MongoAdapter(process.env.MONGO_URL, {
15          useUrlParser: true,
16          useUnifiedTopology: true
17        }),
18        collection
19      };
20    }
21
22    mkdir(path.resolve("./data"));
23  }
```

Slika 7. db.mixin.js prvi deo

Da bismo se u okviru našeg projekta povezali sa Mongo bazom, potrebno je instalirati sledeće pakete komandom npm install mongoddb moleculer-db i moleculer-db-adapter-mongo. Moleculer-db je moleculer-ov servis za čuvanje entiteta u bazi, dok je moleculer-db-adapter-mongo MongoDB native adapter za moleculer db servise. Ukoliko je podešen connection URL - MONGO_URL kreiraće se adapter za Mongo s podešenim url-om i prosleđenom kolekcijom. U ovom projektu će biti iskorišćen za rad sa movies kolekcijom.

Ukoliko URL nije podešen, konekcija sa bazom neće biti moguća, pa se vrši "simuliranje" baze tako što se kreira data direktorijum u tekućem direktorijumu u kom će se kreirati fajl ime_kolekcije.db korišćenjem MemoryAdapter-a.

```

23
24     return {
25         mixins: [DbService],
26         adapter: new DbService.MemoryAdapter({ filename: `./data/${collection}.db` })
27     };
28 };

```

Slika 8. db.mixin.js drugi deo

Movies servis obezbeđuje čuvanje filmova i definiše akcije koje je moguće pozvati. Radi se o sledećim akcijama: za čitanje svih filmova i za kreiranje novog filma. Kod je prikazan u nastavku.

```

moleculer-intro > services > movies.service.js > actions
1 | "use strict";
2
3 | const DbService = require("../mixins/db.mixin");
4
5 | module.exports = {
6 |     name: "movies",
7 |     mixins: [
8 |         DbService("movies")
9 |     ],
10
11 |     settings: {
12 |         fields: ["_id", "title", "description", "year", "director", "mainActor"],
13 |
14 |         entityValidator: {
15 |             title: { type: "string", min: 1 },
16 |             description: { type: "string", min: 1 },
17 |             year: { type: "string", min: 1 },
18 |             director: { type: "string", min: 1 },
19 |             mainActor: { type: "string", optional: true }
20 |         }
21 |     },
22

```

Slika 9. movies.service.js podešavanja

Kreiramo servis pod imenom movies i importujemo db.mixin koji je ranije definisan da bismo iskoristili njegove funkcije. Kao jedini parametar ovom servisu se prosleđuje naziv kolekcije – movies. Svi Db adapteri imaju zajednički skup podešavanja. U ovom servisu se u settings delu podešavaju polja koja će entitet da ima (title, description...) i definiše se validator za entitet. Sva polja su stringovi i svi moraju da budu definisani osim mainActor-a.

Nakon ovoga se implementiraju akcije koje se pozivaju prilikom kreiranja novog filma i listanja svih filmova.

```

23   actions: {
24
25     /**
26      * Create a new movie.
27      *
28      * @actions
29      * @param {Object} movie - Movie entity
30      *
31      * @returns {Object} Created entity
32      */
33     create: {
34       params: {
35         movie: { type: "object" }
36       },
37       async handler(ctx) {
38         let entity = ctx.params.movie;
39         await this.validateEntity(entity);
40
41         const doc = await this.adapter.insert(entity);
42         this.broker.emit("movie.create", doc);
43         return doc;
44       }
45     },
46

```

Slika 10. Akcija za kreiranje novog filma

Za kreiranje novog filma koristimo adapter da u našu kolekciju insertujemo novi entitet koji je prosleđen kao parametar pod imenom movie. Pre nego što upišemo ovaj entitet u bazu potrebno je izvršiti validaciju prosleđenih podataka prema šemi koja je definisana u settings delu.

```

46     /**
47      * List all movies.
48      *
49      * @actions
50      *
51      * @returns {Object} List of movies
52      */
53     list: {
54       async handler(ctx) {
55
56         let params = {
57           sort: ["title"]
58         };
59
60         const res = await this.adapter.find(params);
61         return res;
62       }
63     },
64   }
65 };

```

Slika 11. Čitanje svih filmova

U list akciji se vrši čitanje svih filmova pozivom find metode adaptera. Filmovi koji se pročitaju su sortirani po naslovu što je definisano u params objektu.

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

Slika 12. 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 hendler 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.create", doc);
```

Slika 13. movie.create

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    // cacher: true
8  };|
```

Slika 14. moleculer.config.js

U ovom fajlu se pored nodelID, 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.

```
molecular-intro > ❏ docker-compose.env
1 LOGLEVEL=info
2 TRANSPORTER=nats://nats:4222
3 SERVICEDIR=services
4 MONGO_URL=mongodb://mongo/movies
```

Slika 15. docker-compose.env

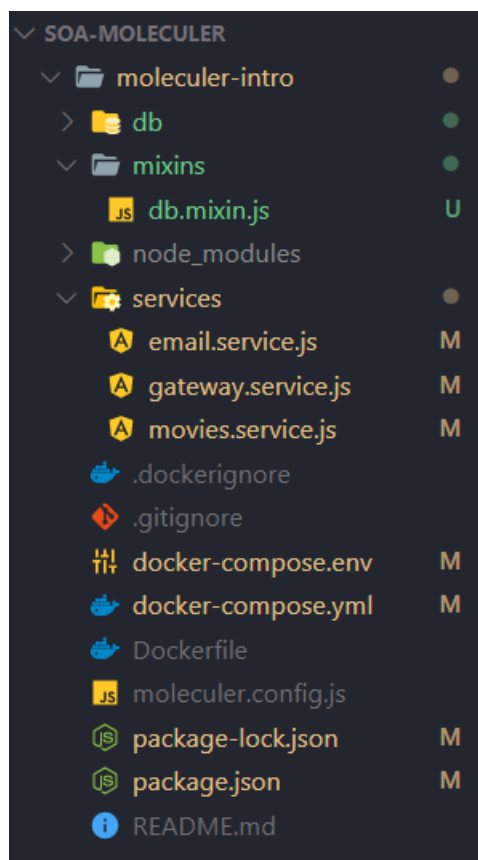
Ovde je podešeno u kom direktorijum se nalaze servisi, kao i podešavanje transportera. Pošto koristimo nats treba prvo navesti nats://, a nakon toga treba navesti ime pod kojim je naveden u docker-compose fajlu – to će u našem slučaju isto biti nats, kao i broj porta.

docker-compose.yml 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. Potreban nam je i mongo koji će se kreirati na osnovu najnovije mongo slike. Movies servis zavisi od monga pa je potrebno to naznačiti u ovom fajlu podešavanjem depends_on i links dela. Docker-compose.yml je prikazan u nastavku.

```
moleculer-intro > docker-compose.yml
1  version: '3.0'
2  services:
3      mongo:
4          image: mongo
5          volumes:
6              - mongodata:/data/db
7      nats:
8          image: nats:latest
9      gateway:
10         build:
11             context: .
12         image: service-gateway
13         env_file: docker-compose.env
14         environment:
15             NODEID: "node-gateway"
16             SERVICES: gateway
17             PORT: 3000
18         ports:
19             - "3000:3000"
20         depends_on:
21             - nats
22     email:
23         build:
24             context: .
25         env_file: docker-compose.env
26         environment:
27             NODEID: "node-email"
28             SERVICES: email
29         depends_on:
30             - nats
31
32     movies:
33         build:
34             context: .
35         env_file: docker-compose.env
36         environment:
37             NODEID: "node-movies"
38             SERVICES: movies
39         depends_on:
40             - nats
41             - mongo
42         links:
43             - mongo
44     volumes:
45         mongodata:
```

Slika 16. docker-compose.yml

- Struktura projekta je prikazana na sledećoj slici.

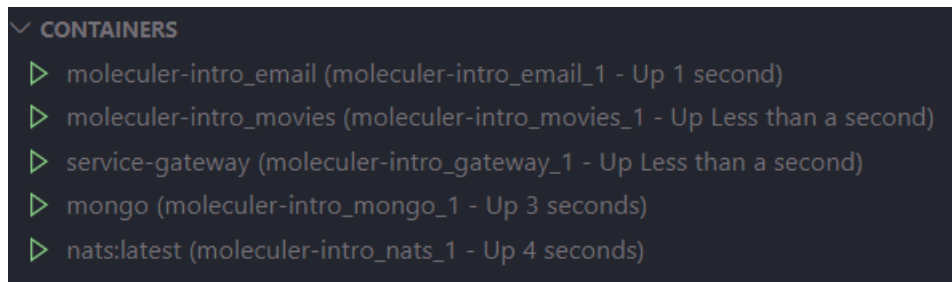


Slika 17. Struktura projekta

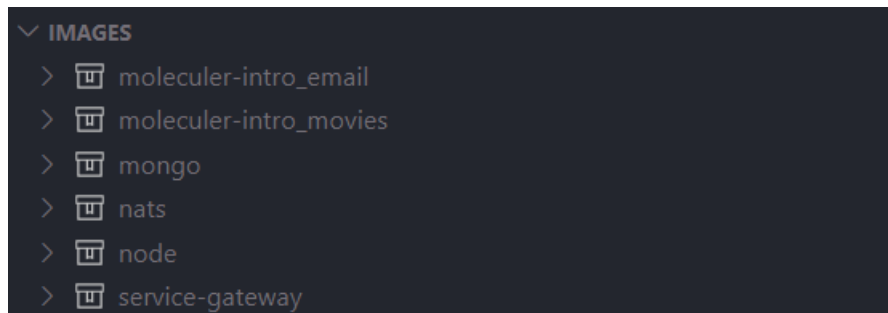
- Izvršavanje docker-compose up --build komande

```
movies_1 | [2020-04-22T02:27:55.402Z] INFO node-movies/BROKER: Moleculer 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: Moleculer 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 18. Deo rezultata izvršenja docker-compose up komande



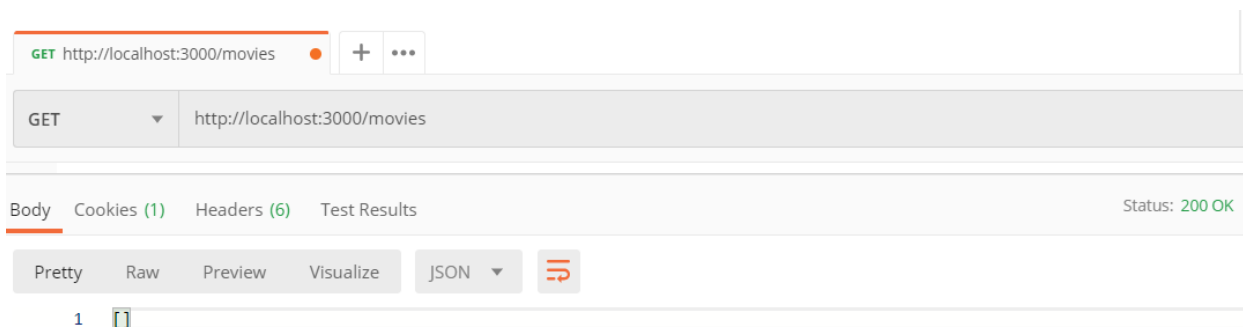
Slika 19. Kreirani kontejneri



Slika 20. docker images

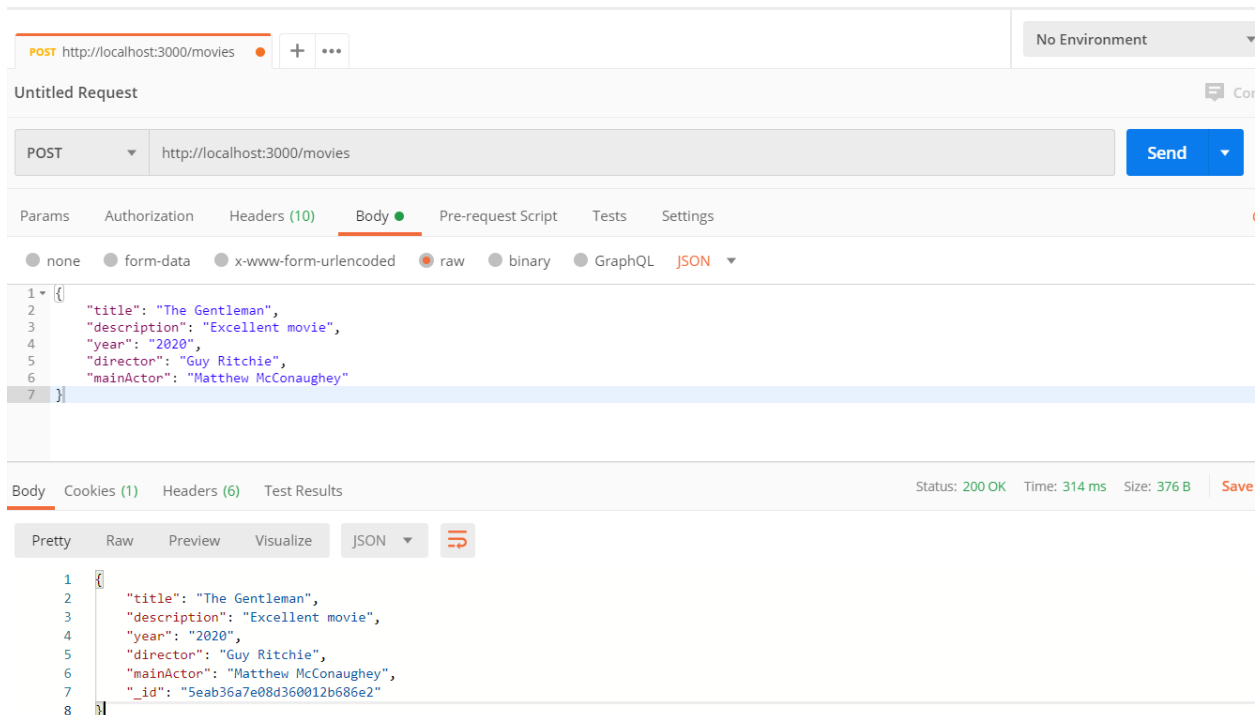
Testiranje aplikacije korišćenjem Postman-a

GET na /movies



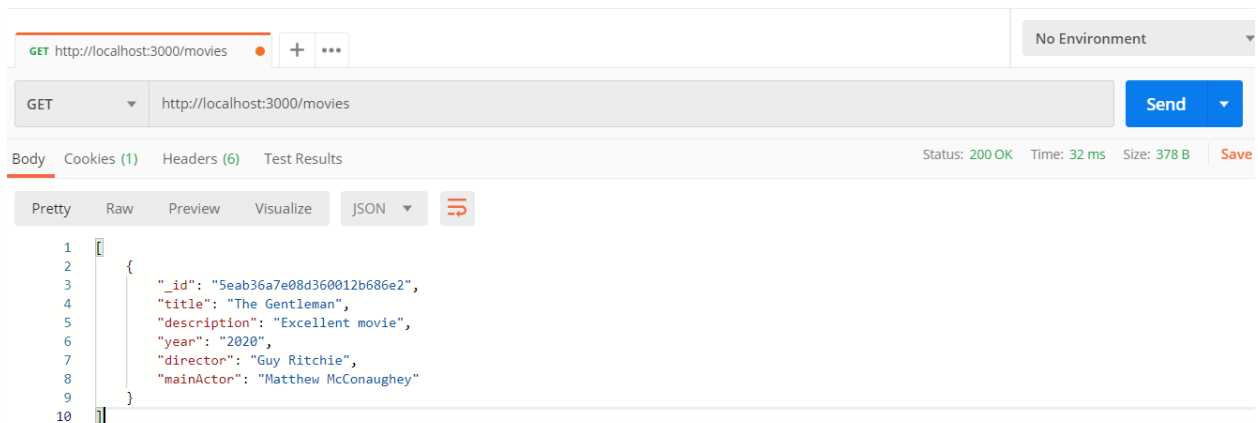
Slika 21. Čitanje svih filmova

POST na /movies



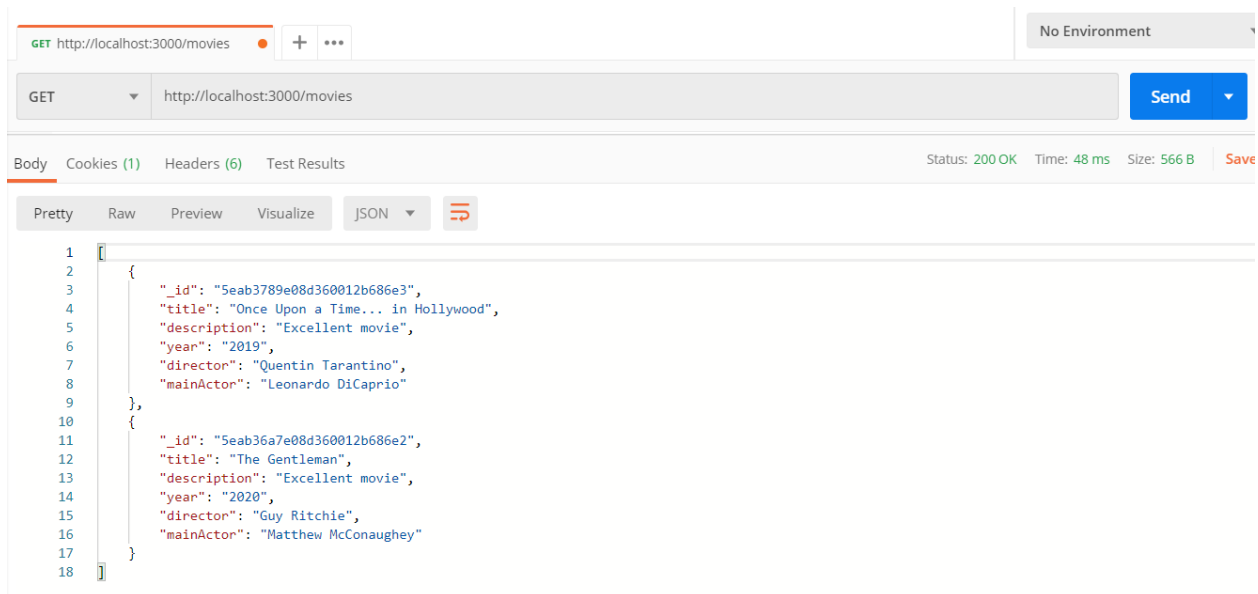
Slika 22. Kreiranje novog filma

GET na `/movies` nakon dodavanja novog filma



Slika 23. Prikaz svih filmova nakon dodavanja

GET na `/movies` nakon dodavanja novog filma – možemo da vidimo da su filmovi sortirani po naslovu, kao što je podešeno u parametrima



Slika 24. Prikaz filmova nakon dodavanja drugog filma

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