Intro

1. Instalacija
2. Dodati u env. Varijable
3. Pokreniti mongod.exe (server)
4. Pokreniti mongosh.exe (klijent/shell)

```
show dbs
show collections
db
use nastava
db
db.createCollection("predavanje")
show collections
db.predavanje.renameCollection("predavanja")

use examples
db.dropDatabase()
use examples
db.inventory.insertOne({ item: "journal", qty: 25 })
db.inventory.insertOne({ item: "canvas", qty: 100 })
```

Node.js (old school)

- Inicijalizirati node projekt, instalirati `mongodb` paket

```
const MongoClient = require("mongodb").MongoClient;
const assert = require("assert");

const url = 'mongodb:// 127.0.0.1:27017';
const dbName = "examples";

MongoClient.connect(url, { useUnifiedTopology: true }, (err, client) => {
    assert.strictEqual(err, null);

    const db = client.db(dbName);

    db.collection("inventory").find({})
        .toArray(function(err, results) {
            console.log(results);
        })
});
```

Node.js (async/await)

```
const mongodb = require("mongodb");
(async () => {
    const url = "mongodb://127.0.0.1:27017";
    const client = new mongodb.MongoClient(url);

    try {
        await client.connect();

    } catch(e) {
        console.log(e);
    }
    const dbName = "examples";
    const db = client.db(dbName);


    const collection = db.collection("inventory");
    // ..
})();
```

Create

```
const collection = db.collection("inventory");
// insert one
let j = await collection.insertOne({
    item: 'canvas',
    qty: 100,
    tags: ['cotton'],
    size: { h: 28, w: 35.5, uom: 'cm' }
});
console.log(j.insertedCount);
console.log(j.insertedId);

// insert many
await collection.insertMany([
    {
        item: "journal",
        qty: 25,
        tags: ["blank", "red"],
        size: { h: 14, w: 21, uom: "cm" }
    },
    {
        item: "mat",
```

```
        qty: 85,
        tags: ["gray"],
        size: { h: 27.9, w: 35.5, uom: "cm" }
    },
    {
        item: "mousepad",
        qty: 25,
        tags: ["gel", "blue"],
        size: { h: 19, w: 22.85, uom: "cm" }
    }
]);
```

READ

```
const fs = require("fs/promises");

// prepare
const collection = db.collection("inventory");

try {
    await collection.drop();
} catch(e) {}

const data = JSON.parse(await fs.readFile("./data1.json"));
await collection.insertMany(data);
console.log(data);


    // ex. 0: access via cursor, iterate with hasNext & next
    const cursor = collection.find({});
    while (await cursor.hasNext()) {
        const datum = await cursor.next()
        console.log(datum);
    }


    // ex. 1: toArray instead of hasNext & next
    const items = await collection.find({}).toArray();
    for (const item of items)
        console.log(item);


    // ex. 2: query criteria
    const items = await collection.find({ status: "D" }).toArray();
```

```
// ex. 3: query criteria with $in operator
const items = await collection.find({ status: {$in: ["A", "D"]} }).toArray();

// ex. 4: query criteria with $or operator
const items = await collection.find({
    $or: [{status: "A"}, {status: "D"}]
}).toArray();

// ex. 5: kad smo kod operatora, query kriterij {a: b} je skraćenica za
const items = await collection.find({ status: { $eq: "D" } }).toArray();

// ex. 6: $gt/$lt operator
const items = await collection.find({
    qty: { $gt: 30 }
}).toArray();

// ex. 7: AND
const items = await collection.find({
    qty: { $gt: 85 }, status: "D"
}).toArray();

// ex. 8, logicki operatori: uz $or, postoje: $and, $not, $nor

// ex. 9: Comparison operatori: $eq, $gt, $gte, $in, $lt, $lte, $ne, $nin

// ex. 10: Nestani dokumenti
// bitan i poredak i jednakost svakog property-a
const items = await collection.find({
    size: { h: 14, w: 21, uom: "cm" }
}).toArray();

// ex. 11. nestani dokumenti, pt. 2
const items = await collection.find({
    "size.h": 14
}).toArray();

// sad mi trebaju podaci iz data2.json


// ex. 12: egzaktno sadrži te stringove u tom poretku
const items = await collection.find({
    tags: ["blank", "red"]
}).toArray();
```

```
// ex. 13: poredak nije bitan & dohvaćeni zapisi mogu imati i druge tagove
const items = await collection.find({
    tags: { $all: ["blank", "red"] }
}).toArray();


// ex. 14: isto sto i ex. 13, ali za 1 tag
const items = await collection.find({ tags: "red"}).toArray();


// ex. 15: barem jedan od elementata zadovoljava > 15 || < 20
const items = await collection.find({
    dim_cm: { $gt: 15, $lt: 20 }
}).toArray();

// ex. 16: barem jedan element u polju je veći od 15 i manji od 20
const items = await collection.find({
    dim_cm:  { $elemMatch: { $gt: 15, $lt: 20 }}
}).toArray();


// ex. 17: $size operator
const items = await collection.find({
    tags: { $size: 3 }
}).toArray();



// PROJECTION
// sad mi trebaju podaci iz data3.json

// ex. 18: projekcija nekih fieldova uz projection objekt
const items = await collection.find({})
    .project({ status: 1, "size.h": 1 })
    .toArray();

// ex. 19: _id nužno ostaje osim ako ga expl. ne odbacimo
const items = await collection.find({})
    .project({ status: 1, "size.h": 1, "_id": 0 })
    .toArray();


// cursor metode: Link

// ex. 19. već smo vidjeli haxNext & next te toArray
```

```
// ex. 20, jako bitno: skip & limit
const items = await collection.find({}).skip(1).limit(3).toArray();

// ex. 21, također jako bitno: sort
// iako smo ga primijenili nakon skim & limit, vidimo
// da se sort primijenio prije...
const items = await collection.find({})
    .skip(1).limit(3).sort({ "size.h": 1, "size.w": -1 })
    .toArray();


// ex. 22: prebrojavanje
const len = await collection.countDocuments();
console.log(len);

// ex. 23: findOne by id
// ovo namjerno pretvaram u string jer će u takvom obliku biti na klijentu
// odn. klijent ne zna što je mongodb.ObjectId klasa
const id = items[0]._id.toString();
// obsolete: new mongodb.ObjectId
// const item = await collection.findOne({ _id: new mongodb.ObjectId(id) });
const item = await collection.findOne({
    _id: mongodb.ObjectId.createFromHexString(id)
});
console.log(item);
```

UPDATE

```
// sad mi trebaju podaci iz data4.json

// ex. 1: updateOne, operator $set, $currentDate
await collection.updateOne(
    { item: "paper" },
    {
        $set: { "size.uom": "cm", "status": "P" },
        $currentDate: { lastModified: true }
    }
);
console.log(await collection.findOne({ item: "paper" }));
```

```
// ex. 2: updateMany, ponovno koristimo update filter i $set
await collection.updateMany(
    { qty: { $lt: 50 } },
    {
      $set: { 'size.uom': 'in', status: 'P' },
      $currentDate: { lastModified: true }
    }
  );
```

```
// ex. 3: replaceOne služi za potpunu izmjenu dokumenta, dok _id ostaje sačuvan
const res = await collection.replaceOne(
    { item: 'paper' },
    {
      item: 'paper',
      instock: [
        { warehouse: 'A', qty: 60 },
        { warehouse: 'B', qty: 40 }
      ]
    }
);
console.log(res, await collection.findOne({item: "paper"}));
```

```
// ex. 4., $inc, findOneAndUpdate
const x = await collection.findOneAndUpdate(
    { item: "postcard" },
    { $inc: { qty: 5 } },
    { returnDocument: "after" } // check "before"
);
console.log(x);
const y = await collection.findOne({ _id: x._id });
console.log(y);
```

DELETE

```
// ex. 1 delete filter je jedina stvar o kojoj vodimo računa
const filter = { status: "D" };
// await collection.deleteOne(filter);
await collection.deleteMany(filter);
console.log(await collection.find({ status: "D" }).toArray());
```