DOCKER

- Pobranie obrazu np mongo:
  docker pull mongo:latest

- Uruchomienie servera mongoDB przy uzyciu Dockera:
  docker run –d –p 27017:27017 –name test-mongo mongo:latest
  -d : proces w tle
  -p : port

- Uruchomienie powloki mongo (najlepiej w drugim terminalu)
  - **Docker exec –it test-mongo bash**

- Logi kontenera:
  - **Docker logs test-mongo –follow**

- Przechowywanie za pomoca WOLUMINOW.
  - Uruchomienie servera + " **-v data-vol:/data/db mongo:latest** "
  - Domyslnie obraz Mongo jest w /data/db
  - Sprawdzenie volumina: **docker volume inspect data-vol**
  - woluminy używane w Dockerze: docker volume ls

- IMPORTOWANIE PLIKOW JSON
  - Przekopiowanie do kontenera:
    - **Docker cp /path/to/file.json test_mongo:/tmp/file.json**
  - MongoImport:
    - **Docker exec –it test_mongo mongoimport --db nazwaDb --collection nazwa kolekcji --file /tmp/file.json --jsonArray**

- IMPORTOWANIE PLIKOW CSV
  - **Kopiowanie takie samo tylko .csv**
  - **docker exec -it test_mongo mongoimport --db newDB --collection testC --type csv --headerline --file /tmp/data.csv**
  - **Headerline – oznacza ze pierwszy wiersz to naglowki kolumn**

- **Eksport**
    - docker exec -it test_mongo mongoexport --db=testDb --collection=persons_csv --out=/data/persons_csv.json

MONGO

- Stworzenie nowej kolekcji i bazy:
    - Bazy: **use newDB**
    - Kolekcji : **db.createCollection(“kolekcja”)**
- Wyswietlenie rzeczy:

1. Db.kolekcja.find() **nie wyswietli wszystkiego**

2. Db.kolekcja.find() .toArray() **wyswietli wszystko**

3. Db.kolekcja.find().limit(10) **wyswietli pierwsze 10**

4. Db.kolekcja.find({age:10}) **wyswietli tylko 10 latkow**

5. Db.kolekcja.find({},{id:0}) **wyswietli wszystko oprocz id**

6. db.kolekcja.find({}, { _id: 0, "grades.date": 0 }).sort({ restaurant_id: 1 }).limit(1) **wyswietli pierwsza wartosc bez ID i Grades.date posortowana wedlug restaurant ID**

7. db.restaurants.find({} ,{ borough:1, cuisine:1, name:1, restaurant_id:1})
   .sort({restaurant_id:1})
   .limit(3)
   .forEach(function(doc){printjson({borough:doc.borough,
   cuisine:doc.cuisine, name:doc.name,
   restaurant_id:doc.restaurant_id})}
   Printowanie w okreslonej kolejnosci dzieki forEach

8. db.restaurants.find({borough:"Bronx"},
   { _id:0,**grades:{date:0}}) .sort({restaurant_id:1}).limit(5)
   Pozbycie sie elemntu zagniezdzonego grades.date

9. db.restaurants.find({borough:"Bronx"},{grades:{date:0},_id:0}).sort({restaurant_id:1}).skip(5).limit(5)
   Wyswietlenie pierwszych 5 po skipnieciu pierwszych 5.

10. db.restaurants.find({"grades.score":{ $gt : 90 }},{_id:0,restaurant_id:0})
Wyswietlenie wynikow wiekszych od 90

11. Db.rest.find({“grades”:{ $elemMatch: { “score”: {$gt:80, $lt:100} } },{ “_id”:0,, “grades.date”:0}).sort({“restaurant_id”:1}).limit(5)
Wyswietlenie wynikow pomiedzy jakas wartoscia

- Update
  - Db.updateMany({gender:’female’},{$set{gender:’Kobieta’}}) zmieni wszystkie gender ‘female’ na ‘Kobieta’
  - {$set{x:y}} albo odwoluje sie do istniejacej rzeczy jesli nie istnieje to ja stworzy
  -

CONNECT TO DB JS:

1. **Npm install mongoose**

2. Tworzenie kontaktu

```
const mongoose = require('mongoose');

mongoose.connect('mongodb://localhost:27017/mydatabase', {
  useNewUrlParser: true,
  useUnifiedTopology: true
}).then(() => {
  console.log('Connected to MongoDB');
}).catch(err => {
  console.error('Could not connect to MongoDB', err);
});
```

3. Tworzenie schematu dla kolekcji

```
const blogSchema = new mongoose.Schema({
  title: { type: String, required: true },
  author: String,
  content: String,
  published: Boolean,
  createdAt: { type: Date, default: Date.now }
});

const Blog = mongoose.model('Blog', blogSchema);
```

Mozliwa walidacja:

```
const userSchema = new mongoose.Schema({
  username: { type: String, required: true, unique: true },
  password: { type: String, required: true, minlength: 6 }
});
```

Walidacja nieestandardowa:

```
const productSchema = new mongoose.Schema({
  name: { type: String, required: true },
  price: {
    type: Number,
    required: true,
    validate: {
      validator: function(v) {
        return v > 0;
      },
      message: props => `${props.value} is not a valid price!`
    }
  }
});
```

4.**Middleware ktory loguje informacje przed zapisem**

```
userSchema.pre('save', function(next) {
  console.log('Saving user:', this);
  next();
});
```

5. **CRUD**

READ – Find()

```
app.get('/items', async (req, res) => {
  const items = await Item.find();
  res.render('index', { items });
});
```

MOZEMY MODYFIKOWAC:

```
app.get('/items', async (req, res) => {
  const items = await Test.find({ age: { $gt: 18 } });
  res.render('index', { items });
});
```

CREATE – SAVE()

```
app.post('/items', async (req, res) => {
  const item = new Item({
    name: req.body.name,
    quantity: req.body.quantity
  });
  await item.save();
  res.redirect('/items');
});
```

UPDATE – findByIdAndUpdate

```
app.post('/items/update/:id', async (req, res) => {
  const { id } = req.params;
  await Item.findByIdAndUpdate(id, { name: req.body.name, quantity: req.body.quantity });
  res.redirect('/items');
});
```

```
await Test.updateMany(
  { age: { $gt: 10 } },
  {
    $set: {
      czyPelnoletni: 'false'
    }
  }
);
```

DELETE – findByIdAndDelete

```
app.delete('/items/delete', async (req, res) => {
  try {
    const result = await Test.deleteMany({ age: { $gte: 10, $lte: 12 } });
    if (result.deletedCount === 0) {
      return res.status(404).json({ error: 'No items found with age between 10
    }
    res.json({ message: `Deleted ${result.deletedCount} items with age between
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Server error' });
  }
});
```

```
app.post('/items/delete/:id', async (req, res) => {
  const { id } = req.params;
  await Item.findByIdAndRemove(id);
  res.redirect('/items');
});
```