Egyszerű RESTful API építése NodeJS, Express és MongoDB segítségével

link

1. Projektmappa létrehozása és megnyitása a VS Code-ban

2. `npm init -y`

3. `npm i express mongoose nodemon dotenv`

4. Az index.js (app.js, server.js) fájl létrehozása (az alkalmazás belépési pontja)

```
const express = require('express');
const mongoose = require('mongoose');
mongoose.set('strictQuery', true);

const app = express();

app.use(express.json());

app.listen(3000, () => {
    console.log(`Server Started at ${3000}`)
})
```

5. A package.json fájlhoz adjuk hozzá a következőt:

```
"scripts": {
    "start": "nodemon index.js"
},
```

6. Most már elindíthatjuk a szerverünket az `npm start` paranccsal

A mongoDB adatbázis konfigurálása

1. A **.env** fájl létrehozása

2. Ebben:

```
DATABASE_URL = mongodb://localhost:27017/datas
```

3. A kiegészített kód:

```
require('dotenv').config();

const express = require('express');
const mongoose = require('mongoose');
const mongoString = process.env.DATABASE_URL;

mongoose.connect(mongoString);
const database = mongoose.connection;

database.on('error', (error) => {
    console.log(error)
})

database.once('connected', () => {
    console.log('Database Connected');
})
const app = express();

app.use(express.json());

app.listen(3000, () => {
    console.log(`Server Started at ${3000}`)
})
```

A routok létrehozása a végpontokhoz

1. routes mappa létrehozása, benne egy routes.js fájllal

index.js-be:

```
    const routes = require('./routes/routes');
    app.use('/api/datas', routes); // Minden végpont ezzel az URL-lel fog kezdődni
```

routes.js-be:

```
    const express = require('express');
    const router = express.Router();
    module.exports = router;
```

2. Végpontok a routes.js-ben:

- Post Method (Új adat felviteléhez (CREATE/INSERT))
  `router.post('/', (req, res) => { res.send('Post API') })`
- Get all Method (Összes adat lekérése (READ))
  `router.get('/', (req, res) => { res.send('Get All API') })`
- Get by ID Method (Adott azonosítójú dokumentum (rekord) lekérése)
  `router.get('/:id', (req, res) => { res.send('Get by ID API') })`
- Update by ID Method (Adott azonosítójú dokumentum (rekord) frissítése)
  `router.patch('/:id', (req, res) => { res.send('Update by ID API') })`
- Delete by ID Method (Adott azonosítójú dokumentum (rekord) törlése)
  `router.delete('/:id', (req, res) => { res.send('Delete by ID API') })`

3. Postman/ThunderClient (A végpontok tesztelése)

A `POST localhost:3000/api/datas` kérésre a `POST API` választ kell kapnunk A GET kérésben: `res.send(req.params.id)`, majd például a `GET localhost:3000/api/datas/1000` kérésre az `1000` választ kell kapnunk.

A modell elkészítése

1. A models mappa létrehozása, benne model.js

```
const mongoose = require('mongoose');
const dataSchema = new mongoose.Schema({
    name: {
        required: true,
        type: String
    },
    age: {
        required: true,
        type: Number
    }
})
module.exports = mongoose.model('Data', dataSchema)
```

2. Import a routes.js fájlba:

```
const Model = require('../models/model');
```

CREATE (POST)

```
router.post('/', async (req, res) => {
    const data = new Model({
        name: req.body.name,
        age: req.body.age
    })
    try{
        const dataToSave = await data.save();
        res.status(200).json(dataToSave)
    }
    catch(error){
        res.status(400).json({message: error.message})
    }
})
```

A `POST localhost:3000/api/datas` URL-re: `{ "name": "Ali", "age": 25 }` body-val: `{"name": "Ali", "age": 25, "_id": ...}` választ kapjuk és az adatbázisba beszúródik.

READ - Az összes adat listázása:

```
router.get('/', async (req, res) => {
    try{
        const data = await Model.find();
        res.json(data)
    }
    catch(error){
        res.status(500).json({message: error.message})
    }
})
```

A `GET localhost:3000/api/datas` kérésre egy objektum tömböt `[{...}, {...}, ...]` kapunk válaszul.

Adatok lekérése azonosító alapján:

```
router.get('/:id', async (req, res) => {
    try{
        const data = await Model.findById(req.params.id);
        res.json(data)
    }
    catch(error){
        res.status(500).json({message: error.message})
    }
})
```

A `GET localhost:3000/api/datas/63a5de94553f5b53df74e67f` kérésre visszakapjuk az adott objektumot: `{...}`

PATCH

```
router.patch('/:id', async (req, res) => {
    try {
        const id = req.params.id;
        const updatedData = req.body;
        const options = { new: true };

        const result = await Model.findByIdAndUpdate(
            id, updatedData, options
        )

        res.send(result)
    }
    catch (error) {
        res.status(400).json({ message: error.message })
    }
})
```

Az options-ben a new beállítással meghatározzuk, hogy a **frissített adatokat** adjuk-e vissza a törzsben vagy sem. A `PATCH localhost:3000/api/datas/63a5de94553f5b53df74e67f` kérésre a válasz a `{ módosított dokumentum }`-ot kapjuk vissza.

Törlés (DELETE)

```
router.delete('/:id', async (req, res) => {
    try {
        const id = req.params.id;
        const data = await Model.findByIdAndDelete(id)
        res.send(`Document with ${data.name} has been deleted..`)
    }
    catch (error) {
        res.status(400).json({ message: error.message })
    }
})
```

A `DELETE localhost:3000/api/datas/63a5de94553f5b53df74e67f` kérésre a `Document with malagi has been deleted..` üzenet lesz a válaszban.