goto

goto impact foundation

Generasi Gigih

Full Stack Engineer

Module 3.3: Integrating MongoDB with NodeJS

Setting Up

Initializing Project

Create an empty directory and initialize a new NodeJS project inside of it:


```
npm init
```

Installing Necessary Packages

To install the necessary packages, run the following command:


```
npm install express body-parser mongoose nodemon dotenv
```

But, what are they?

Mongoose

MongoDB object data modelling (ODM) for NodeJS. Mongoose has the following features:

- Schema definition
- Data models
- Query building
- Validation

Dotenv

Zero-dependency module that loads environment variables from `.env` file.

Nodemon

A command-line interface (CLI) utility that wraps your NodeJS app, watches the file system, and automatically restarts the process.

Let's write some code...

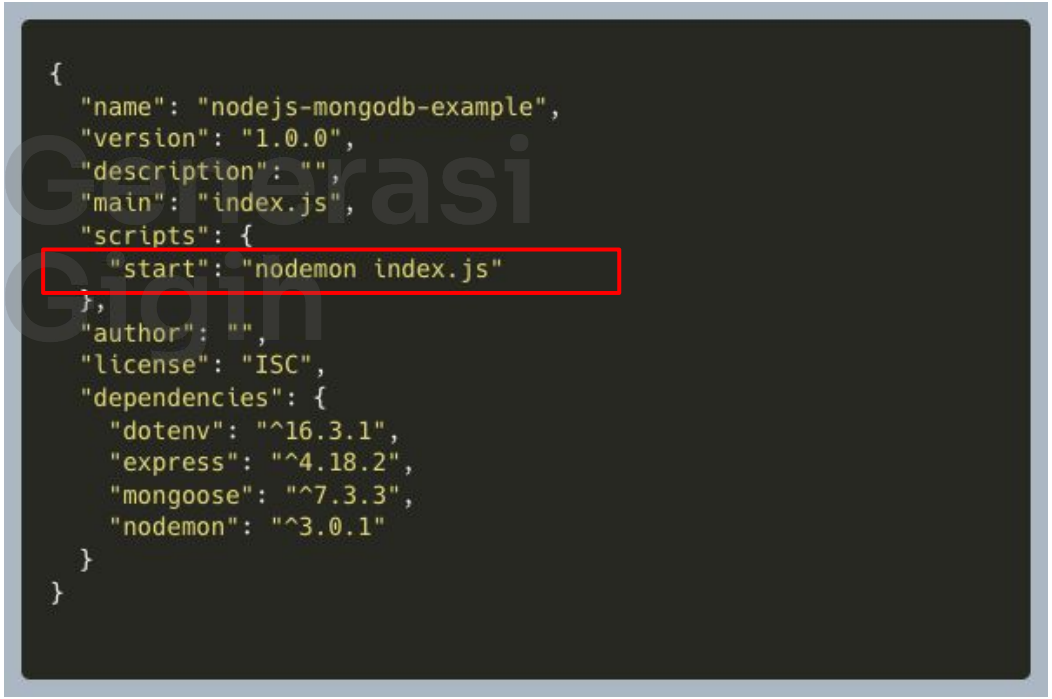
Initializing Express JS App Server (1)

Add the following snippet to your index.js:

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

const app = express();

app.use(express.json());

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

Setting Up Express JS App Server (2)

Modify this line in your package.json:



Setting Up MongoDB Connection with Mongoose (1)

Add the following lines:

```
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}`)
})
```

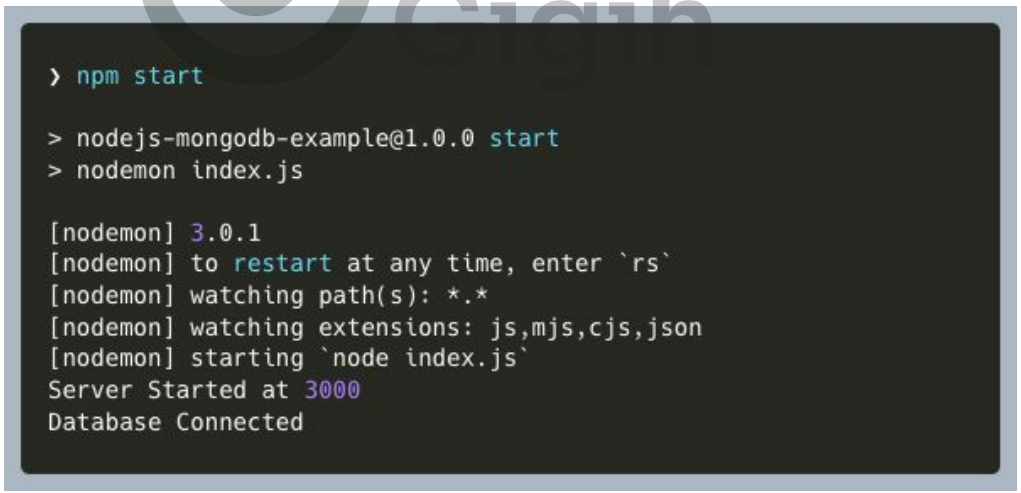
Setting Up MongoDB Connection with Mongoose (2)

Add the following in .env:


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

Up and Running

You can now run `npm start` and see the following result:



CRUD with MongoDB and NodeJS

Routes

Create a folder named `routes` in your project's root directory, add the following snippet, and save it as `routes.js`:


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

const router = express.Router()

module.exports = router;
```

Model

Create a folder named `models` in your project's root directory and write a file named `movie.js`. The following snippet is how you define a Mongoose schema and export a Mongoose model:

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

const movieSchema = new mongoose.Schema({
    title: {
        required: true,
        type: String
    },
    year: {
        required: true,
        type: Number
    }
})

module.exports = mongoose.model('Movie', movieSchema)
```

Routes

Edit `routes.js`. The following snippet is how we define a route for a post REST API that will save a Mongoose model as a MongoDB document:

```
const express = require('express');
const router = express.Router();
const Movie = require('../models/movie');

router.post('/post', (req, res) => {
  const movie = new Movie({
    title: req.body.title,
    year: req.body.year
  })

  try {
    const movieToSave = movie.save();
    res.status(200).json(movieToSave)
  }
  catch (error) {
    res.status(400).json({message: error.message})
  }
})

module.exports = router;
```

Index.js

Modify `index.js` to use body-parser to help us parsing Json requests:

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

const express = require('express');
const bodyParser = require('body-parser');
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 routes = require('./routes/routes');
const app = express();

app.use(bodyParser.json());
app.use(
  bodyParser.urlencoded({
    extended: true,
  }),
);
app.use('/api', routes)

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

Trying It Out

Now you can try creating a new movie with the following command:


```
❯ curl --header "Content-Type: application/json" \
  --request POST \
  --data '{"title":"Sample Movie","year":2023}' \
  http://localhost:3000/api/post
```

Verify

Verify it from your MongoDB console:


```
cinema> db.movies.find()
[
  {
    _id: ObjectId("64af975be175e61ce8b0f10b"),
    title: 'Sample Movie',
    year: 2023,
    __v: 0
  }
]
```

Read All

Edit `routes.js`. The following snippet is how we define a route for a get REST API that will fetch data from MongoDB documents:

```
// Some codes are truncated

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

module.exports = router;
```

Try it Out

Now you can try getting all movies with the following command:


```
❯ curl --header "Content-Type: application/json" \
  --request GET \
  http://localhost:3000/api/getAll
```

Read One

Edit `routes.js`. The following snippet is how we define a route for a get REST API that will fetch a specific data based on an id from MongoDB documents:

```
// Some codes are truncated

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

module.exports = router;
```

Try it Out

Now you can try getting a movie based on a specific id with the following command:


```
❯ curl --header "Content-Type: application/json" \
  --request GET \
  http://localhost:3000/api/get/[put-an-id-here]
```

Update

Edit `routes.js`. The following snippet is how we define a route for a patch REST API that will update a document from MongoDB database:

```
// Some codes are truncated

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

    const result = await Movie.findByIdAndUpdate(
        id, updatedMovieData, options
    )

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

module.exports = router;
```

Try it Out

Now you can try updating a movie based on a specific id with the following command:


```
❯ curl --header "Content-Type: application/json" \
  --request PATCH \
  --data '{"title":"Tenet","year":2021}' \
  http://localhost:3000/api/update/[put-an-id-here]
```

Delete

Edit `routes.js`. The following snippet is how we define a route for a delete REST API that will delete a document from MongoDB database:

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

module.exports = router;
```

Try it Out

Now you can try updating a movie based on a specific id with the following command:


```
❯ curl --header "Content-Type: application/json" \
  --request PATCH \
  --data '{"title":"Tenet","year":2021}' \
  http://localhost:3000/api/update/[put-an-id-here]
```

Congratulations!
You've just wrote your first simple CRUD app with MongoDB and NodeJS

Transaction

Definition

A transaction is a logical unit of processing in a database that includes one or more database operations, which can be read or write operations.

There are situations where your application may require reads and writes to multiple documents (in one or more collections) as part of this logical unit of processing.

An important aspect of a transaction is that it is never partially completed—it either succeeds or fails.

ACID

ACID, an acronym for Atomicity, Consistency, Isolation, and Durability, is the accepted set of properties a transaction must meet to be a “true” transaction.

ACID transactions guarantee the validity of your data and of your database’s state even where power failures or other errors occur.

ACID - Atomicity

Atomicity ensures that all operations inside a transaction will either be applied or nothing will be applied. A transaction can never be partially applied; either it is committed or it aborts.

ACID - Consistency

Consistency ensures that if a transaction succeeds, the database will move from one consistent state to the next consistent state.

ACID - Isolation

Isolation is the property that permits multiple transactions to run at the same time in your database. It guarantees that a transaction will not view the partial results of any other transaction, which means multiple parallel transactions will have the same results as running each of the transactions sequentially.

ACID - Durability

Durability ensures that when a transaction is committed all data will persist even in the case of a system failure.

ACID-Compliance in MongoDB

A database is said to be ACID-compliant when it ensures that all these properties are met and that only successful transactions can be processed. In situations where a failure occurs before a transaction is completed, ACID compliance ensures that no data will be changed.

MongoDB is a distributed database with ACID compliant transactions across replica sets and/or across shards.

Example of MongoDB Transaction with Javascript

```
const { MongoClient } = require('mongodb');

async function runTransaction() {
  const uri = 'mongodb://localhost:27017';
  const client = new MongoClient(uri);

  try {
    await client.connect();

    const session = client.startSession();
    const transactionOptions = {
      readPreference: 'primary',
      readConcern: { level: 'snapshot' },
      writeConcern: { w: 'majority' },
    };

    session.startTransaction(transactionOptions);

    const db = client.db('cinema');
    const collection = db.collection('movies');

    try {
      await collection.insertOne({ name: 'The Fellowship of The Ring' }, { session });
      await collection.insertOne({ name: 'The Two Towers' }, { session });
      await collection.insertOne({ name: 'The Return of The King' }, { session });

      await session.commitTransaction();
      console.log('Transaction committed successfully.');
    } catch (error) {
      console.error('Error occurred during transaction. Aborting...', error);
      await session.abortTransaction();
    } finally {
      session.endSession();
    }
  } catch (error) {
    console.error('Error occurred while connecting to the database:', error);
  } finally {
    client.close();
  }
}

runTransaction().catch(console.error);
```

Questions?