Kết nối MongoDB với NodeJS

Việc kết nối MongoDB với NodeJS rất dễ dàng với thư viện `mongoose`. Đầu tiên, cài đặt `mongoose`:

```
npm install mongoose
```

Sau đó sử dụng `mongoose.connect` ở trong file `index.js`:

```
import mongoose from "mongoose";

mongoose
    .connect("mongodb://127.0.0.1:27017/your-database-name", {
        useNewUrlParser: true,
        useUnifiedTopology: true,
    })
    .then(() => {
        console.log("Connect database successfully!");
    })
    .catch((error) => {
        console.error(`Connect failed: ${error}`);
    });
```

Với `your-database-name` là tên database của bạn, nếu bạn bỏ qua tên database, `mongoose` sẽ tạo một database mới với tên mặc định là `test`.

- `useNewUrlParser`: Sử dụng URL parser mới.
- `useUnifiedTopology`: Sử dụng cơ chế kết nối mới.

Khởi tạo model

Để tương tác với MongoDB, chúng ta cần khởi tạo một model. Ví dụ, chúng ta muốn tạo một model `User`:

```
// models/User.js
import mongoose from "mongoose";

const userSchema = new mongoose.Schema(
    {
        email: {
            type: String,
            required: true,
            unique: true,
        },
        password: {
            type: String,
            required: true,
        },
        role: {
```

```
            type: String,
            default: "member",
            enum: ["member", "admin", "superAdmin"],
        },
    },
    {
        versionKey: false,
        timestamps: true,
    }
);

export default mongoose.model("User", userSchema);
```

Có rất nhiều tuỳ chọn trong `mongoose.Schema`, chúng ta sẽ cùng tìm hiểu dần. Phổ biến nhất là các tuỳ chọn:

- `versionKey`: Loại bỏ trường `__v` mặc định của MongoDB.
- `timestamps`: Tự động thêm trường `createdAt` và `updatedAt`.
- `enum`: Chỉ cho phép giá trị trong mảng.
- `unique`: Giá trị phải là duy nhất.
- `required`: Giá trị không được để trống.
- `default`: Giá trị mặc định.
- `type`: Kiểu dữ liệu.
- `ref`: Tham chiếu đến một model khác.
- `populate`: Lấy dữ liệu từ model khác.
- `select`: Chọn trường cần lấy.
- `match`: Kiểm tra giá trị.
- `min` và `max`: Giá trị tối thiểu và tối đa.
- `validate`: Kiểm tra giá trị.
- `index`: Tạo index.
- `sparse`: Cho phép trường rỗng.

Giả sử chúng ta muốn tạo model Product:

```
// models/Product.js
import mongoose from "mongoose";

import mongoose, { mongo } from "mongoose";

const productSchema = new mongoose.Schema(
    {
        title: {
            type: String,
            required: true,
        },
        price: {
            type: Number,
            required: true,
        },
```

```
        description: {
            type: String,
            default: "Updating",
        },
    },
    { timestamps: true, versionKey: false }
);

export default mongoose.model("Product", productSchema);
```

Thao tác với model

Import model vào file `index.js`:

```
// index.js
import express from "express";
import mongoose from "mongoose";
import Product from "./models/Product.js";
```

Thêm dữ liệu

```
app.post("/products", async (req, res) => {
    try {
        const product = new Product(req.body);
        await product.save();

        // Thay vì sử dụng new và save, chúng ta có thể sử dụng create
        // const product = await Product.create(req.body);

        return res.status(201).send(product);
    } catch (error) {
        return res.status(400).send(error);
    }
});
```

Lấy dữ liệu

```
app.get("/products", async (req, res) => {
    try {
        const products = await Product.find();
        return res.send(products);
    } catch (error) {
        return res.status(400).send(error);
    }
});
```

Hoặc lấy dữ liệu theo id:

```
app.get("/products/:id", async (req, res) => {
    try {
        const product = await Product.findById(req.params.id);
        if (!product) {
            return res.status(404).send("Product not found");
        }
        return res.send(product);
    } catch (error) {
        return res.status(400).send(error);
    }
});
```

Sửa dữ liệu

```
app.put("/products/:id", async (req, res) => {
    try {
        const product = await Product.findByIdAndUpdate(req.params.id,
req.body, {
            new: true,
        });
        if (!product) {
            return res.status(404).send("Product not found");
        }
        return res.send(product);
    } catch (error) {
        return res.status(400).send(error);
    }
});
```

Xóa dữ liệu

```
app.delete("/products/:id", async (req, res) => {
    try {
        const product = await Product.findByIdAndDelete(req.params.id);
        if (!product) {
            return res.status(404).send("Product not found");
        }
        return res.send(product);
    } catch (error) {
        return res.status(400).send(error);
    }
});
```

Tìm kiếm dữ liệu

```
app.get("/products/search", async (req, res) => {
    try {
        const products = await Product.find({
            title: { $regex: req.query.title, $options: "i" },
        });
        return res.send(products);
    } catch (error) {
        return res.status(400).send(error);
    }
});
```

Sắp xếp dữ liệu

```
app.get("/products/sort", async (req, res) => {
    try {
        const products = await Product.find().sort({ price: -1 });
        return res.send(products);
    } catch (error) {
        return res.status(400).send(error);
    }
});
```

Phân trang

```
app.get("/products/paginate", async (req, res) => {
    try {
        const page = parseInt(req.query.page);
        const limit = parseInt(req.query.limit);

        const products = await Product.find()
            .limit(limit)
            .skip((page - 1) * limit);

        return res.send(products);
    } catch (error) {
        return res.status(400).send(error);
    }
});
```

Tổng số dữ liệu

```
app.get("/products/count", async (req, res) => {
    try {
        const count = await Product.countDocuments();
        return res.send({ count });
```

```
    } catch (error) {
        return res.status(400).send(error);
    }
});
```