MongoDB & Mongoose

스키마: 컬렉션 도큐먼트 구조 정의. 하나의 Mongoose 스키마는 하나의 MongoDB 컬랙션과 직접적으로 사상된다(map).

const blog = new Schema({

title: String,

slug: String,

published: Boolean,

author: String,

content: String,

tags: [String],

createdAt: Date,

updatedAt: Date,

comments: [{

user: String,

content: String,

votes: Number

}]

});

모델: 스키마를 사용하여 컬렉션에 있는 각각의 도큐먼트에 그것을 적용한다. 모든 CRUD 작업에 관여한다.

모델에 사용되는 첫번째 인수는 컬렉션 이름의 단수형이어야 한다. Mongoose가 이것을 자동적으로 복수형으로 바꾸어서, 소문자로 변환하고, 데이터베이스 컬랙션 이름으로 사용한다.

const Blog = mongoose.model(‘Blog’, blog)

이 예에서 Blog는 blogs 컬렉션으로 번역된다.

실행 환경 설정

터미널에서 다음 명령 실행

> md mongodb-mongoose

> cd mongodb-mongoose

> npm init -y

> npm i mongoose

> npm i -D nodemon

> code .

package.json 파일에 스크립트 추가

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1",

    "dve": "nodemon index.js"

  },

추가할 때 위에처럼 test 이런 게 있으면 ‘,’ 이걸 한 다음에 추가해야 한다.

...

"scripts": {

"dev": "nodemon index.js"

},

"type": "module",

…

위에거 완성 예시

{

  "name": "mongodb-mongoose",

  "version": "1.0.0",

  "main": "index.js",

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1",

    "dev": "nodemon index.js"

  },

  "type": "module",

  "keywords": [],

  "author": "",

  "license": "ISC",

  "description": "",

  "dependencies": {

    "mongoose": "^8.4.0"

  },

  "devDependencies": {

    "nodemon": "^3.1.0"

  }

}

MongoDB 에 연결

인증 연결 실행

mongod --auth --port 27017 --dbpath /path/to/유

또는

mongod.exe --auth --port 27017 --dbpath ..\data

가끔 – 이렇게 하나로 보일 때가 있다. 1개가 아니라 2개다. – 이걸 2번 써야 한다.

admin 사용자 생성

use admin

db.createUser(

{

user: “myUserAdmin”,

pwd: “abc123”,

roles: [{role: “userAdminAnyDatabase”, db: “admin”},

{role: “readWriteAnyDatabase”, db: “admin”}]

}

)

사용자 생성

use lms

db.createUser (

{

user: “myLmser”,

pwd: “abc123”,

roles: [ { role: “readWrite”, db:”lms”},

{ role: “read”, db:”reporting”}]

}

)

스키마와 모델 생성(model/Blog.js) vscode에서 mongodb-mongoose 폴더를 만들었는데, 거기에서 그냥 새 폴더를 또 만들고(model) 그 안에 Blog.js 파일을 만들면 된다.

import mongoose from 'mongoose';

const { Schema, model } = mongoose;

const blogSchema = new Schema({

title: String,

slug: String,

published: Boolean,

author: String,

content: String,

tags: [String],

createdAt: Date,

updatedAt: Date,

comments: [{

user: String,

content: String,

votes: Number

}]

});

const Blog = model('Blog', blogSchema);

export default Blog;

>>> 여기까지는 vscode에서 Blog.js에 작성

vscode에서 index.js 파일 생성 여기 아래는 index.js 에서 작성

import mongoose from 'mongoose';

import Blog from './model/Blog.js';

mongoose.connect("mongodb://myLmser:abc123@localhost/lms?retryWrites=true&w=majority")

>>>> 여기 위에 주소는 lms까지만 적어도 된다. ? 다음부터는 안 적어도 괜찮음.

// Create a new blog post object

>>> 여기 아래 코드는 index.js에서 작성.

const article = new Blog({

title: 'Awesome Post!',

slug: 'awesome-post',

published: true,

content: 'This is the best post ever',

tags: ['featured', 'announcement'],

});

// Insert the article in our MongoDB database

await article.save();

>>> 여기까지는 index.js 파일에 작성(2024-05-22에 여기까지 했음).

// Find a single blog post

const firstArticle = await Blog.findOne({});

console.log(firstArticle);

몽고DB에서 스키마(schema)는 문서의 구조를 정하는 것.

npm run dev >>> 이거는 cmd창에서 mongodb-mongoose로 cd 명령어로 이동한 다음에 거기에서 입력하면 됨. 그 다음에 몽고 DB 프로그램 직접 실행하고, 거기에서 lms에 가서 blogs 폴더에 가면 자료가 나온다.

위에서 순서대로 하고 위에 npm run dev 명령어를 하면 몽고DB에서 lms > blogs 폴더에 있는 데이터와 같은 게 나온다. 같은 내용의 자료가 나온다.

다음은 객체 생성과 저장을 한 번에 한다.

// Create a new blog post and insert into database

const article = await Blog.create({

title: 'Awesome Post!',

slug: 'awesome-post',

published: true,

content: 'This is the best post ever',

tags: ['featured', 'announcement'],

});

console.log(article);

데이터 갱신

article.title = "The Most Awesomest Post!!";

await article.save();

console.log(article);

데이터 검색

const article = await Blog.findById("62472b6ce09e8b77266d6b1b").exec();

console.log(article);

>> 여기서 findById에 쓰는 주소인 “”에 들어가는 문자는 CMD에 나온 내 ObjectId를 복사해서 붙여넣기로 사용해야 한다. 위에 ID를 그대로 사용하면 안 된다.

도큐먼트 필트 추출

const article = await Blog.findById("62472b6ce09e8b77266d6b1b", "title slug content").exec();

console.log(article);

>> 여기도 마찬가지로 “” 안의 문자열 긴거는 내꺼에 배정된 고유 ID를 따로 써야 한다. 위에 거를 그대로 사용하면 안 된다.

두번째 패러미터는 Object|String|Array<String>으로 지정할 수도 있다. >> GPT 에서는 위의 방식은 올바른 게 아니라고 한다. 굳이 사용할 이유는 없을 듯.

데이터 삭제

데이터를 검색해서 가장 먼저 나온 데이터 1개만 삭제.

const blog = await Blog.deleteOne({ author: "Jesse Hall" })

console.log(blog)

위와 같지만 모든 데이터 삭제.

const blog = await Blog.deleteMany({ author: "Jesse Hall" })

console.log(blog)

>> 위와 같은 코드를 vscode에서 사용하면 실제로 몽고DB에서 데이터가 추가, 삭제, 수정으로 반영이 된다.

유효성 검증

도큐먼트에 실제적으로 필요한 필드를 정의한다.

>> Blog.js 코드를 수정한다. 그런데 여기를 이렇게 수정을 하면 index.js에서 위에 create부분을 주석처리하거나 없애줘야 한다. 데이터 타입이 바뀌었는데 index.js에서 위에 부분은 그 부분을 반영하지 않고 예전의 방식으로 작성이 되었기 때문이다.

const blogSchema = new Schema({

title: {

type: String,

required: true,

},

slug: {

type: String,

required: true,

lowercase: true,

},

published: {

type: Boolean,

default: false,

},

author: {

type: String,

required: true,

},

content: String,

tags: [String],

createdAt: {

type: Date,

default: () => Date.now(),

immutable: true,

},

updatedAt: Date,

comments: [{

user: String,

content: String,

votes: Number

}]

});

그 밖의 유용한 메서드

exists()

null 이나 쿼리에 일치하는 도큐먼트의 ObjectId를 반환

* 이 메서드는 주어진 조건에 해당하는 데이터가 존재하는지 여부를 확인합니다.
* 존재하는 경우에는 true를 반환하고, 그렇지 않으면 false를 반환합니다.
* 따라서 위의 코드에서는 author 필드가 'Jesse Hall'인 데이터가 존재하는지 확인합니다.

const blog = await Blog.exists({ author: "Jesse Hall" })

console.log(blog)

where()

* findOne 메서드는 주어진 조건에 해당하는 첫 번째 데이터를 반환합니다.
* 조건에 해당하는 데이터가 없으면 null을 반환합니다.
* 따라서 위의 코드에서는 author 필드가 'Jesse Hall'인 첫 번째 데이터를 조회합니다.

// Instead of using a standard find method

const blogFind = await Blog.findOne({ author: "Jesse Hall" });

// Use the equivalent where() method

Blog.where('author').equals('Jesse Hall')은 Mongoose의 쿼리 빌더를 사용하여 MongoDB에서 특정 조건에 해당하는 문서를 검색하는 방법 중 하나입니다. 이 코드는 MongoDB의 find() 메서드를 호출하는 것과 유사한 역할을 합니다.

여기서 각 부분을 살펴보겠습니다:

* Blog: Mongoose 모델을 나타냅니다. MongoDB의 컬렉션과 대응됩니다.
* where('author'): MongoDB에서 author 필드를 기준으로 검색하겠다는 것을 나타냅니다.
* equals('Jesse Hall'): author 필드가 'Jesse Hall'과 정확히 일치하는 문서를 찾겠다는 것을 나타냅니다.

따라서 위의 코드는 author 필드가 'Jesse Hall'인 모든 문서를 찾는 것입니다. 이렇게 하면 find() 메서드와 동일한 결과를 얻을 수 있습니다.

이를 사용하여 해당 조건을 만족하는 문서를 조회하고, 이후에는 적절한 방식으로 결과를 처리할 수 있습니다. 예를 들어, find(), findOne(), 또는 count() 등의 메서드를 호출하여 문서를 조회하거나, 결과를 반복하여 처리할 수 있습니다.

const blogWhere = await Blog.where("author").equals("Jesse Hall");

console.log(blogWhere)

select()

Blog.where('author').equals('Jesse Hall').select('title author')는 MongoDB에서 특정 조건에 해당하는 문서를 검색하고, 검색된 문서의 특정 필드만 선택하여 가져오는 방법입니다. 이 코드는 Mongoose의 쿼리 빌더를 사용하여 작성되었습니다.

여기서 각 부분을 살펴보겠습니다:

* Blog: Mongoose 모델을 나타냅니다. MongoDB의 컬렉션과 대응됩니다.
* where('author').equals('Jesse Hall'): MongoDB에서 author 필드가 'Jesse Hall'과 정확히 일치하는 문서를 검색하겠다는 것을 나타냅니다.
* select('title author'): 검색된 문서에서 title과 author 필드만 선택하여 가져오겠다는 것을 나타냅니다.

따라서 위의 코드는 author 필드가 'Jesse Hall'인 모든 문서를 찾고, 해당 문서들 중에서 title과 author 필드만 선택하여 가져오는 것입니다.

이를 사용하여 해당 조건을 만족하는 문서를 조회하고, 이후에는 선택한 필드를 포함한 적절한 방식으로 결과를 처리할 수 있습니다. 예를 들어, find(), findOne(), 또는 count() 등의 메서드를 호출하여 문서를 조회하거나, 결과를 반복하여 처리할 수 있습니다. 선택한 필드에 따라 결과의 형태가 달라질 수 있습니다.

const blog = await Blog.where("author").equals("Jesse Hall").select("title author")

console.log(blog)

다중 스키마

새로운 파일 User.js를 model 폴더에 생성

import mongoose from 'mongoose';

const {Schema, model} = mongoose;

const userSchema = new Schema({

name: {

type: String,

required: true,

},

email: {

type: String,

minLength: 10,

required: true,

lowercase: true

},

});

const User = model('User', userSchema);

export default User;

조인 데이터 모델

블로그 스키마에서 user 모델을 참조한다.

import mongoose from 'mongoose';

const { Schema, SchemaTypes, model } = mongoose;

const blogSchema = new Schema({

...,

author: {

type: SchemaTypes.ObjectId,

ref: 'User',

required: true,

},

...,

comments: [{

user: {

type: SchemaTypes.ObjectId,

ref: 'User',

required: true,

},

content: String,

votes: Number

}];

});

…

index.js 파일 갱신

...

import User from './model/User.js';

...

const user = await User.create({

name: 'Jesse Hall',

email: 'jesse@email.com',

});

const article = await Blog.create({

title: 'Awesome Post!',

slug: 'Awesome-Post',

author: user.\_id,

content: 'This is the best post ever',

tags: ['featured', 'announcement'],

});

console.log(article);

populate() Mongoose 메서드 사용

* populate 메서드는 author 필드에 저장된 ObjectId를 실제 User 문서로 대체합니다.
* 즉, author 필드가 참조하는 사용자의 데이터를 함께 가져옵니다.
* Step 2: .populate('author')는 찾은 문서에서 author 필드에 저장된 ObjectId를 실제 User 문서로 대체하여, author 필드에 사용자 데이터를 포함시킵니다.
* [
* { "\_id": "1", "title": "Awesome Post!", "author": "A1" },
* { "\_id": "2", "title": "Awesome Post!", "author": "A2" },
* { "\_id": "3", "title": "Another Post", "author": "A3" }
* ]
* [
* { "\_id": "A1", "name": "Author One" },
* { "\_id": "A2", "name": "Author Two" }
* ]

const article1 = await Blog.findOne({ title: 'Awesome Post!' }).populate('author');

* console.log(article1);

{

"\_id": "1",

"title": "Awesome Post!",

"author": {

"\_id": "A1",

"name": "Author One"

}

* }

const article = await Blog.findOne({ title: "Awesome Post!" }).populate("author");

console.log(article);

Middleware

Mongoose에서 middleware는 스키마 레벨에서 비동기 함수의 시작 전이나 실행 중에 돌이가는 함수

‘updated’ date를 기사가 저장되거나 갱신될 때 마다 업데이트하기 위해서 Blog.js에 다음을 추가한다.

blogSchema.pre('save', function(next) {

this.updated = Date.now(); // update the date every time a blog post is saved

next();

});

index.js 파일에서 기사를 검색하고, 제목을 갱신하고, 저장한다.

const article = await Blog.findById("6247589060c9b6abfa1ef530").exec();

article.title = "Updated Title";

await article.save();

console.log(article);

pre() 에 더해서 post() mongoose() 미들웨어 함수도 있다.

(2024-05-23에 여기까지 했음).