Mongoose 101

Adapted from https://zellwk.com/blog/mongoose/

# Introduction

Mongoose is a library that makes MongoDB easier to use. It does two things:

* It gives structure to MongoDB Collections
* It gives you helpful methods to use

# Prerequisites

Assume you have done the following:

* You have installed MongoDB on your computer or a connection string to Mongo Atlas
* You know how to set up a local MongoDB connection
* You know how to see the data you have in your database
* You know what are “collections” in MongoDB

# Connecting to a database

## First, you need to download Mongoose

npm install mongoose –save

You can connect to a database with the connect method. Let’s say we want to connect to a database called street-fighters. Here’s the code you need:

const mongoose = require('mongoose')

const url = 'mongodb://127.0.0.1:27017/street-fighters'

mongoose.connect(url, { useNewUrlParser: true })

## To check whether the connection has succeeded, we can use the open event. To check whether the connection failed, we use the error event.

const db = mongoose.connection

db.once('open', \_ => {

console.log('Database connected:', url)

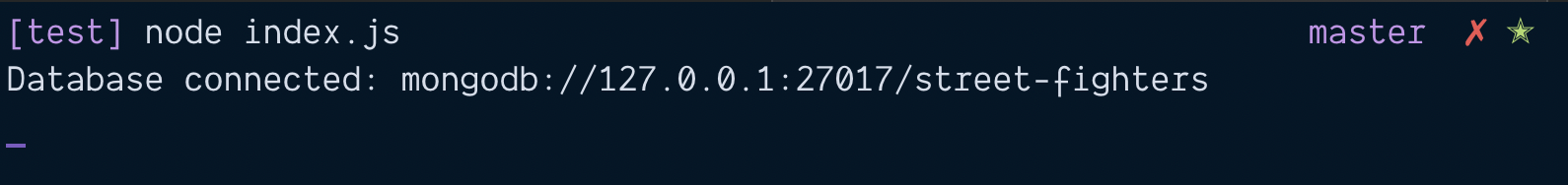
})

db.on('error', err => {

console.error('connection error:', err)

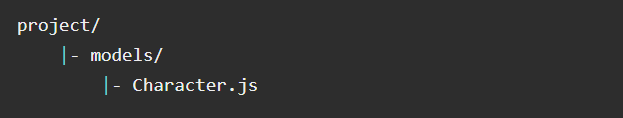
})

You should see a log like this



# Creating a Model

In Mongoose, it’s a normal practice to put each model in its own file. So we will create a Character.js file first. This Character.js file will be placed in the models folder



In Character.js, we create a characterSchema

const mongoose = require('mongoose')

const Schema = mongoose.Schema

const characterSchema = new Schema({

name: String,

ultimate: String

})

module.exports = mongoose.model('Character', characterSchema)

# Creating a character

Let’s say you have a file called index.js. This is where we’ll perform Mongoose operations for this tutorial.



const Character = require('./models/Character')

const ryu = new Character ({

name: 'Ryu',

ultimate: 'Shinku Hadoken'

})

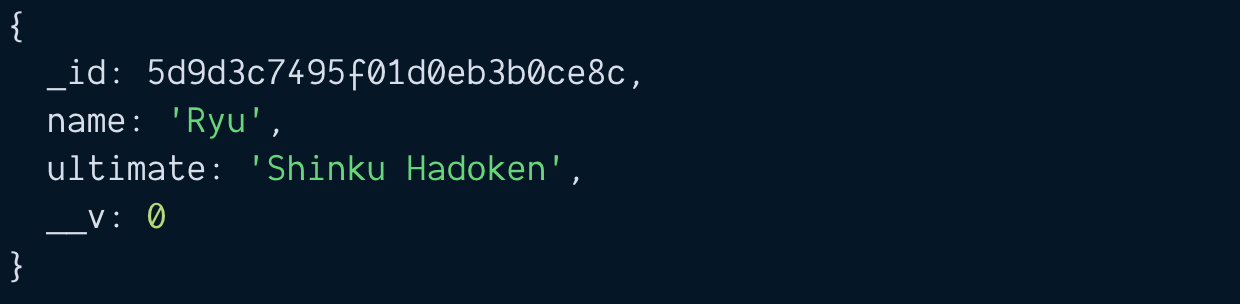
ryu.save(function (error, document) {

if (error) console.error(error)

console.log(document)

})

If you run the code above, you should see this in the console



# Promises and Async/await

Mongoose supports promises. It lets you write nicer code like this:

// This does the same thing as above

function saveCharacter (character) {

const c = new Character(character)

return c.save()

}

saveCharacter({

name: 'Ryu',

ultimate: 'Shinku Hadoken'

})

.then(doc => { console.log(doc) })

.catch(error => { console.error(error) })

You can also use the await keyword if you have an asynchronous function.

async function runCode() {

const ryu = new Character({

name: 'Ryu',

ultimate: 'Shinku Hadoken'

})

const doc = await ryu.save()

console.log(doc)

}

runCode()

.catch(error => { console.error(error) })

# Uniqueness

We don’t want to have three Ryus in the database. We want to have ONE Ryu only. To do this, we can use the unique option.

const characterSchema = new Schema({

name: { type: String, unique: true },

ultimate: String

})

For unique to work properly, you need to clear the Characters collection. To clear the Characters collection, you can use this:

await Character.deleteMany({})