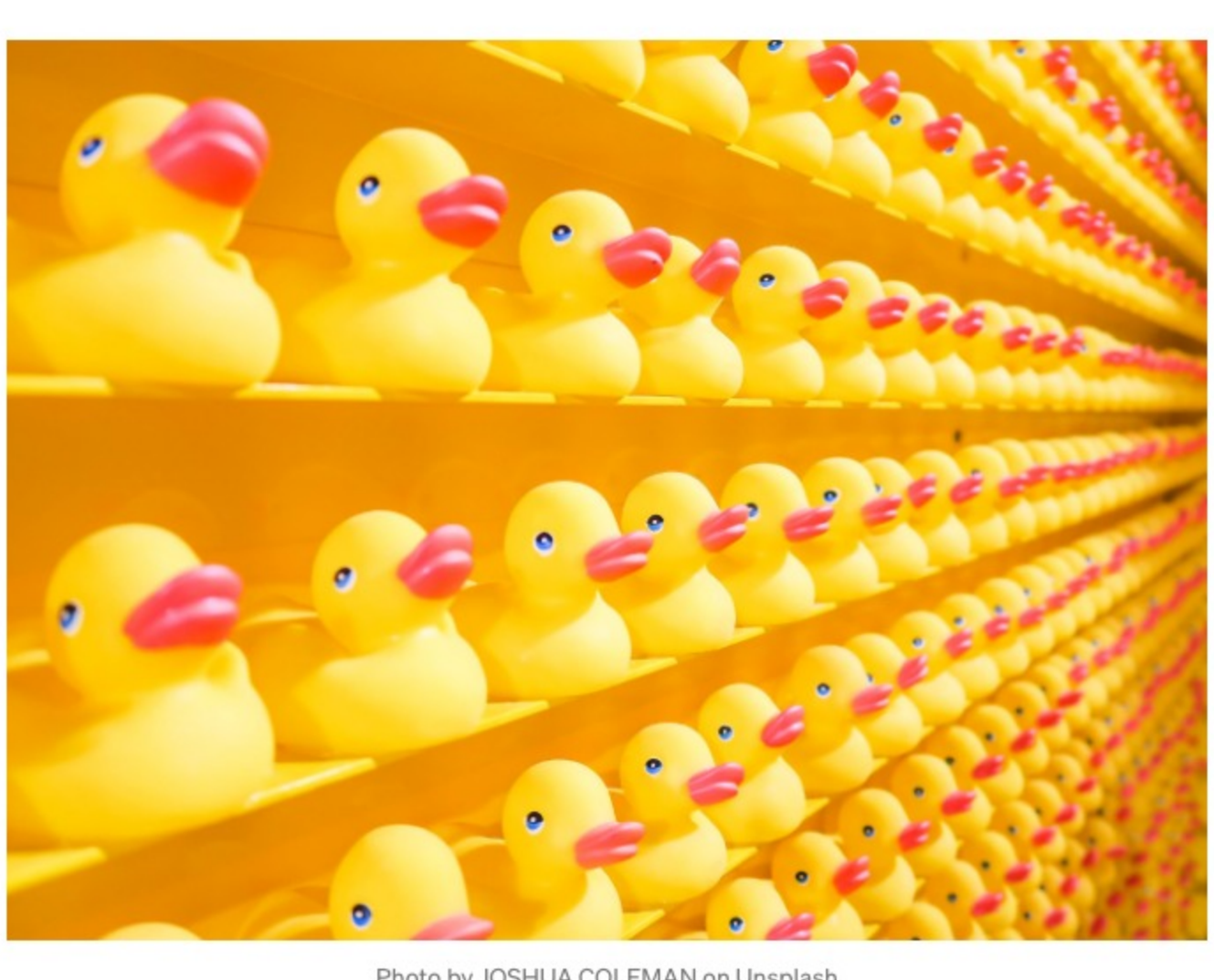

Photo by JOSHUA COLEMAN on Unsplash

One-To-Many Relationships with MongoDB and Mongoose in Node/Express

Brandon Lau Nov 20, 2019 · 3 min read

. . .

One to many relationships with mongoDB can easily be mapped using mongoose. Although MongoDB isn't a relational database like PostgreSQL, you can still create relationships that are either embedded or referenced. Referenced relationships are more akin to the relations using in relational databases.

Let's first define the two Schemas we'll be making. Let's say that a car belongs to a user, and a user has many cars. Therefore, the schemas will look like this:

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;


let UserSchema = new Schema({
  name: String,
  age: String,
  cars: [{
    type: Schema.Types.ObjectId,
    ref: "Car"
  }]
})

module.exports = mongoose.model("User", UserSchema);
```

User Schema

The User Schema has a name field, an age field (albeit in string format, can also use Number), and also an array of cars. The cars are of type mongoose.Schema.Types.ObjectID, which are filled with the unique IDs of other objects. The ref denotes in what collection the object(or car) resides on. This is an array, meaning users possess the capability to have MANY cars.

The Car Schema looks as follows :

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;


let CarSchema = new Schema({
  make: String,
  model: String,
  owner: {
    type: Schema.Types.ObjectId,
    ref: "User"
  }
})

module.exports = mongoose.model("Car", CarSchema);
```

Car Schema

Unlike the User model, the Car model does not possess an array of Owners, rather, its owner field points to a single ObjectID in the User collection.

Let's now see how we can query a User, and see all of its cars, or query a car, and also see its user.

```
let UserModel = require("../models/UserModel.js");

let UserController = {
  find: async (req,res) => {
    let found = await UserModel.find({name: req.params.username});
    res.json(found);
  },
  all: async (req,res) => {
    let allUsers = await UserModel.find()
    res.json(allUsers);
  },
  create: async(req,res) => {
    let newUser = new UserModel(req.body);
    let savedUser = await newUser.save();
    res.json(savedUser);
  },
  getAllCars: async (req,res) => {
    let foundUser = await UserModel.find({name: req.params.username}).populate("cars");
    res.json(foundUser);
  }
}

module.exports = UserController;
```

User Controller

In the User controller, we have the basic methods to find a specific car (find), get all the cars (all), and create a new car (create). However, there's a method getAllCars that finds a user by their name (located in the params), and then populate its car field. Let's see our main server.js file to see the routes available first.

```
const express = require("express");
const app = express();
const mongoose = require("mongoose");
const cors = require("cors");
const PORT = process.env.PORT || 5000;
require("dotenv").config();

//Connect to db
mongoose.connect(process.env.DB_URL, {
  useNewUrlParser: true,
  useUnifiedTopology: true
},() => {
  console.log("Connected");
})

//Middleware
app.use(express.json());
app.use(cors());

//Routes
const UserControls = require("./controllers/UserController.js");
const CarControls = require("./controllers/CarController.js");
app.get("/users", UserControls.all);
app.get("/users/create", UserControls.create);
app.get("/users/:username", UserControls.find);
app.get("/users/:username/cars", UserControls.getAllCars);

app.get("/cars", CarControls.all);
app.get("/cars/:username/create", CarControls.create);

//Start serv
app.listen(PORT);
```

When we make a GET request to /users/:username/cars, we will be greeted by:

```
[
  {
    "cars": [
      {
        "_id": "5dd47b8130c3ca598048f61e",
        "make": "Toyota",
        "model": "Tacoma",
        "owner": "5dd47b0930c3ca598048f61c",
        "__v": 0
      },
      {
        "_id": "5dd47b9230c3ca598048f61f",
        "make": "Tesla",
        "model": "Model-X",
        "owner": "5dd47b0930c3ca598048f61c",
        "__v": 0
      }
    ],
    "_id": "5dd47b0930c3ca598048f61c",
    "name": "Brandon",
    "age": "28",
    "__v": 2
  }
]
```

Because of the populate method, we are able to retrieve an array of all the car objects, rather than just their ObjectID. That is a simple one-to-many relationship!

Nodejs Mongoose Mongodb

Related

JAVASCRIPT

Connect MongoDB to Node using Express and Mongoose

NodeJs(Express) and

More from Brandon Lau

Nov 12, 2019

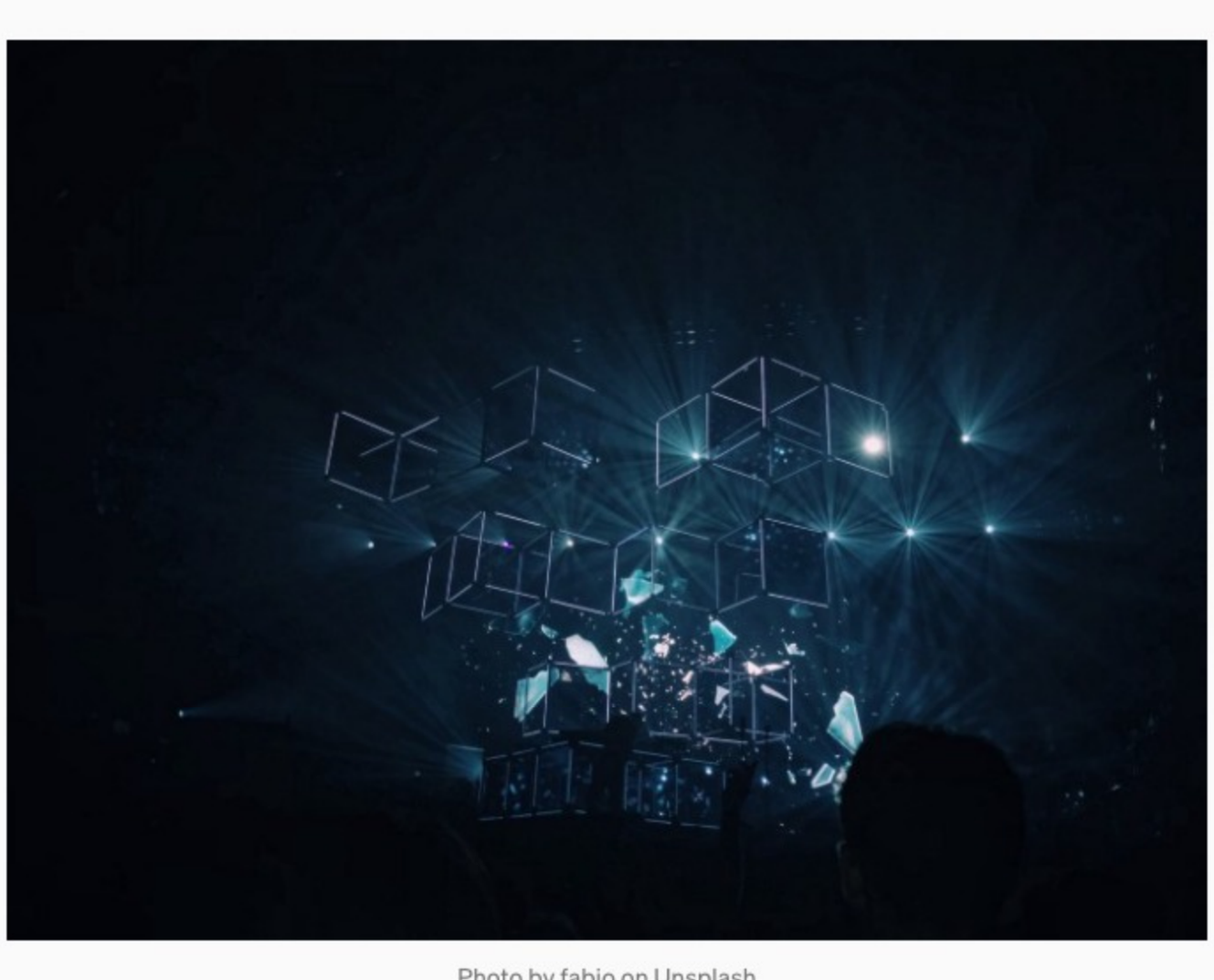

Photo by fabio on Unsplash

Node.js, Express, and MongoDB. Get started fast.

In this guide, I'll be walking through how to quick and easily get set up using Express and MongoDB, complete with controllers and routes using mongoose!

To get started, create a blank folder.

1. run npm init to generate a package.json file.
2. npm install mongoose express dotenv(used to load content into…

Read more · 3 min read

Nov 7, 2019

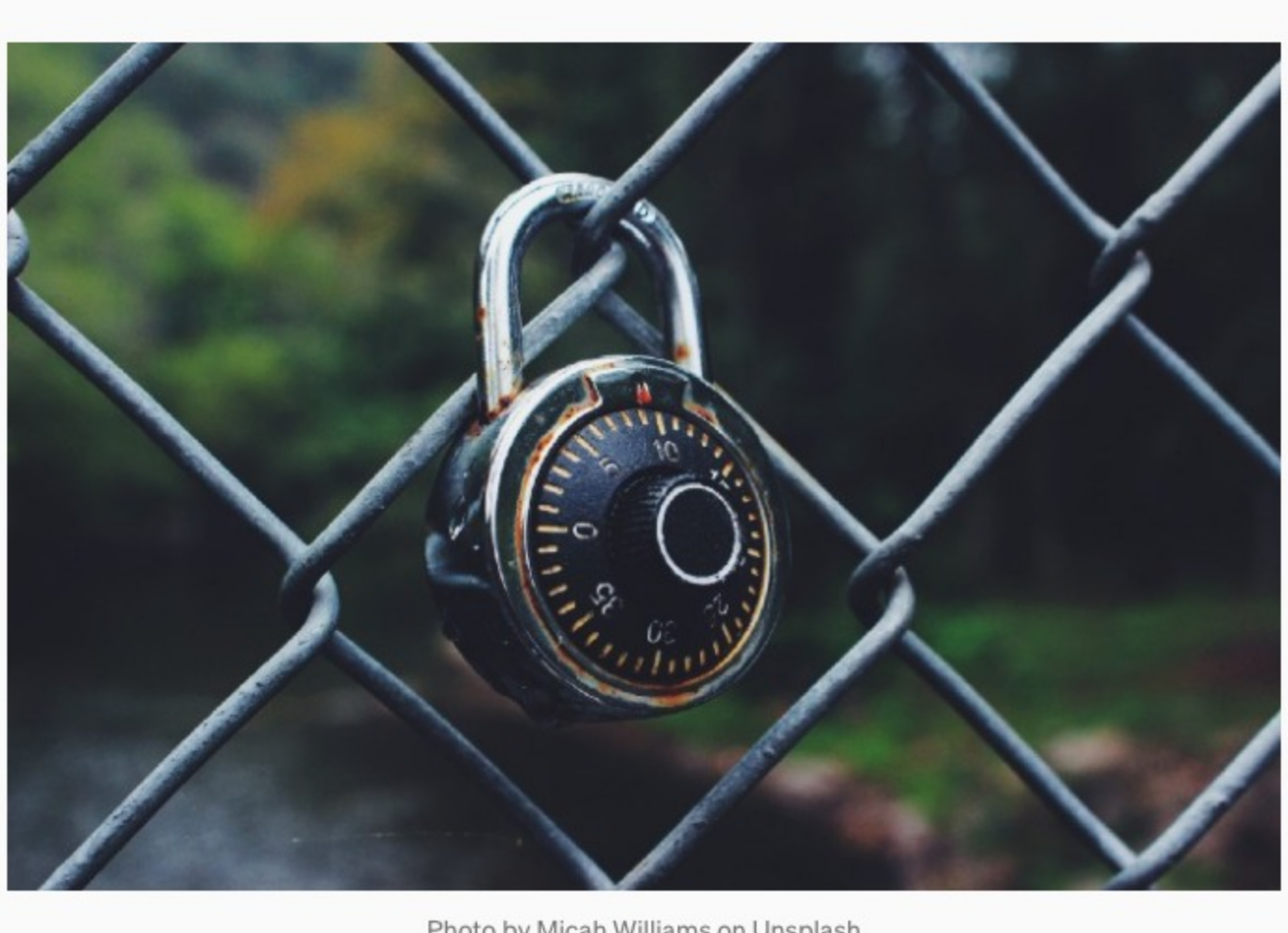

Photo by Micah Williams on Unsplash

Rails Authentication with BCrypt & JWT, made simple and clear

. . .

Every time the topic turns to Authentication, people seem to lose motivation. I'm going to present an easy, step-by-step authentication guide with Ruby on Rails, and the gems BCrypt and JWT. I promise this will be as clear cut as can be!

First, let's go through the logic of BCrypt…

Read more · 5 min read

Oct 27, 2019

Photo by Christian Wiediger on Unsplash

Chrome Browser Alternatives (Chromium)

. . .

Looking at GlobalStats statcounter for Desktop market share, one can see that Google Chrome has approximately a 65% market share in the USA.

Read more · 2 min read

Aug 13, 2019

chinePhoto by Franki Chamaki on Unsplash

Google DeepMind and (Alpha)Go

I've recently started reading about Machine Learning, and stumbled across Google's DeepMind AlphaGo. AlphaGo is a program meant to play the game Go. For anyone unfamiliar with Go, here's what it looks like:

Read more · 2 min read

Jul 25, 2019

Photo by Kobu Agency on Unsplash

A Brief Regular Expressions Introduction

A regular expression is a convenient way to match patterns in a Language of choice. For a regular expression to be valid for a certain language, the expression must adhere to any rules of the Language, and account for them. …

Read more · 3 min read