INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE
ESCOLA SUPERIOR DE TECNOLOGIA

Web Programming

6th Activity -Node/Angular/Mongo

2020/2021

1. Mongo

NoSQL Database: This is a non-relational database. It has no tables, no definition of patterns hence allows room for a lot more flexibility with nested data. It looks like JavaScript Object with a bunch of key/value pairs and here is what an example of information about a user stored in a relational database looks like:

```
{
  name: "Vanessa",
  age: 20,
  city: "Lagos",
  comments: [
      {text: "Learning everyday"},
      {text: "Javascript for the win!"}
    ]
}
```

Installing mongo

https://docs.mongodb.com/manual/tutorial/install-mongodb-on-os-x/

Download the production release that fits your system, for windows it would give you an MSI file that will run through a standard Windows install. It will default to installing into your Program Files directory (in a \server\3.4\ subfolder) but Mongo itself is quite small, so I'll be storing the database in the directory where I would be working.

Mongod and mongo: To get started lets create a `demo` directory in the C drive and in it create a sub directory called `data` . Then in the command prompt navigate to the directory where the MongoDB file installed into :

```
C:\program files\mongodb\server\3.4\bin
```

and into the `bin` directory as shown above type the following:ing:

```
mongod --dbpath c:\demo\data
```

You'll see the Mongo server start up. Once it says `[initandlisten] waiting for connections on port 27017` then all is good, the server is up and running. Now **open a second command prompt,** navigate to your mongoDB installation directory (again) and type `mongo` you'll see something like the following:

```
MongoDB shell version v3.4.6
connecting to: mongodb://127.0.0.1:27017

//you might also see connecting to: test (a default database Mongo decides to use if you don't
specify one on the command line)
```

Mongo command prompt

show dbs: Type `show dbs` into the mongo client and you will see the names of databases that are available.

use: This allows us specify a database to use and the command to type looks like this `use demo` demo being the name of the database. *Note: If the demo database has not been created, it will go ahead and create it, then use it.* But if we check our database again ie `show dbs` we won't still see `demo` and that's because it's empty. So we need to add things into `demo` and the way to do this is by creating `collections` (grouping things together).

insert: `insert` is just like `use` in the sense that we don't have to create a `collection` ahead of time we can just go ahead and type `db.users.insert()` where `db` is the database that we are on— `demo` , `users` is the collection that doesn't exist but will be created and `insert` is the command that will insert data into the `users` collection in the `demo` database. Now lets go ahead and populate our database by adding in objects.

```
db.users.insert({name: "Vanessa", age: 20})
db.users.insert({name: "Josephine", age: 23})

//then hit enter and you will get a WriteResult message back
```

find: This command will enable us view all users in the database, to use it type `db.users.find()` and without passing in anything it will automatically return all the users in the collection. To make it return all the `users` in a nice format with line breaks type this `db.users.find().pretty()` . Sometimes you might want to find a particular user and not all the users, and for that you can do this `db.users.find({ name: "Vanessa"})` . This will only return the `user` with the `name: Vanessa`

update: In a case where we enter the wrong age for a user , we can use the `update` command to change it. The update command takes in two different things:

- What to select by + what to update with (In case we want to overwrite what is selected with the updated information)

```
db.users.update({name: "Vanessa"}, {name: "Sandy"})
```

- What to select by + how/what we want to update with (this will preserve the selected and only change the updated information)

```
db.users.update({name: "Vanessa"}, {$set: {name: "Gift", age:
21}})
```

remove: This command does what the name implies. it removes a specific user from the database by passing in the name of the user we want to remove.

```
db.users.remove({name: "Gift"})
```

and this removes every user in the database that matches the specified name. But if you want to specify the number of users to remove that matches the name you can do that by typing this:

```
db.users.remove({name: "Gift"}).limit(1)
```

1 can be any number of users you want to be deleted.

ctrl + c: This is used to shut down both the `mongod server` and the `mongo shell` anytime we are done working.

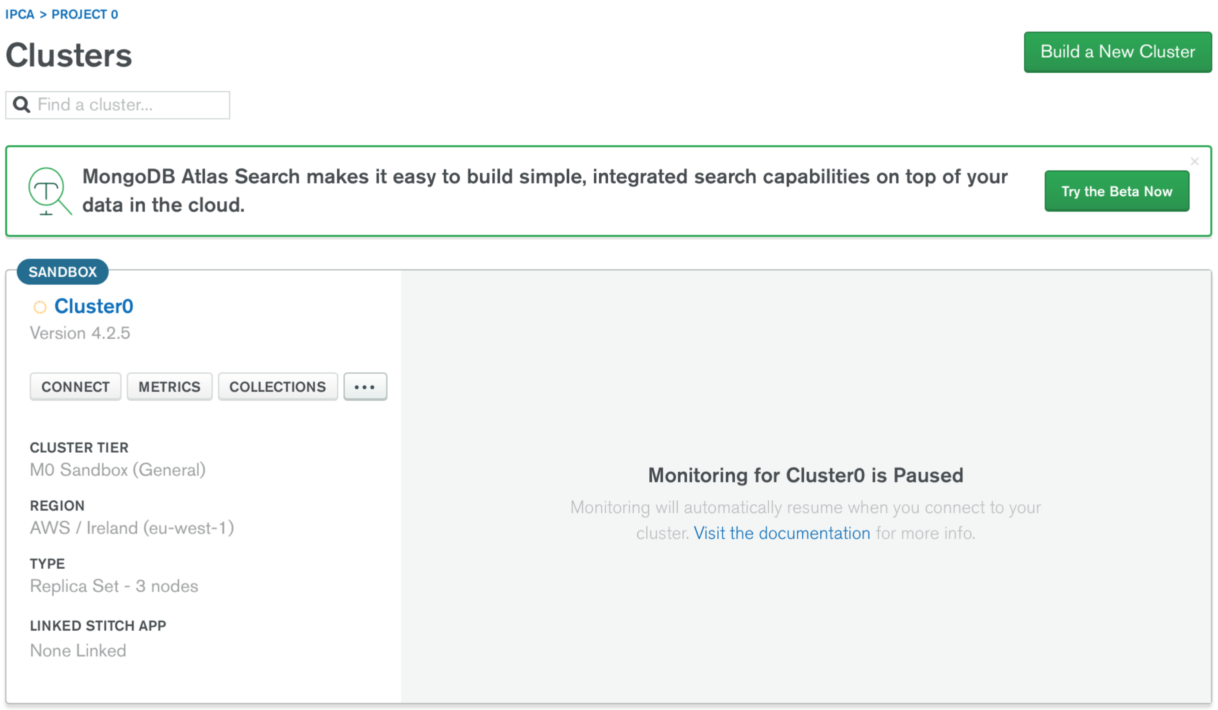
Compass

https://www.mongodb.com/download-center/compass

Adding Mongo via Cloud

Follow the link Mongo Atlas and click in start free and you should create a new Cluster. You can choose the default options, and be sure you are using the free cluster tier, M0. And finish the process.

After this you will have a mongo cluster. Let's configure some things.


IPCA > PROJECT 0

Clusters

Build a New Cluster

Find a cluster...

MongoDB Atlas Search makes it easy to build simple, integrated search capabilities on top of your data in the cloud.

Try the Beta Now

SANDBOX

Cluster0

Version 4.2.5

CONNECT METRICS COLLECTIONS •••

CLUSTER TIER
M0 Sandbox (General)

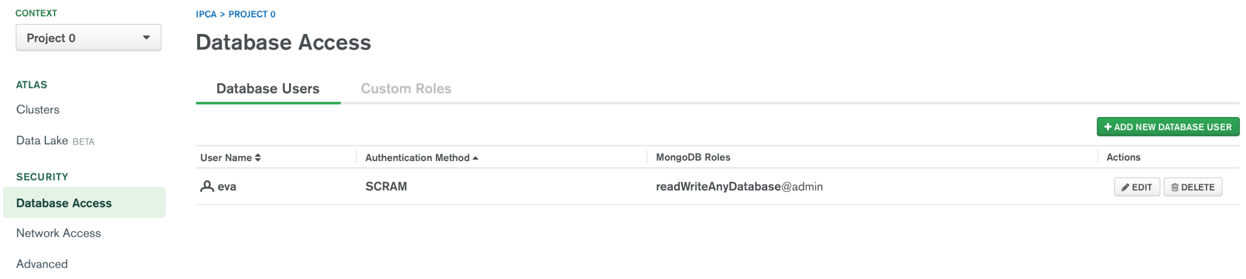
REGION
AWS / Ireland (eu-west-1)

TYPE
Replica Set - 3 nodes

LINKED STITCH APP
None Linked

Monitoring for Cluster0 is Paused

Monitoring will automatically resume when you connect to your cluster. Visit the documentation for more info.


CONTEXT
Project 0

ATLAS
Clusters
Data Lake BETA

SECURITY
Database Access
Network Access
Advanced

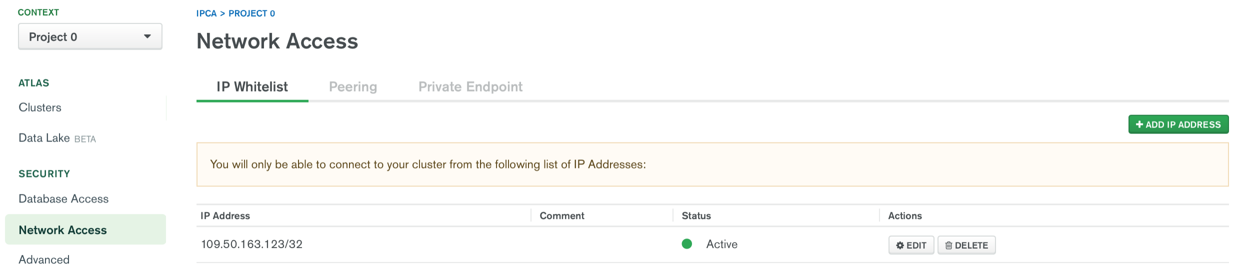
IPCA > PROJECT 0

Database Access

Database Users | Custom Roles

+ ADD NEW DATABASE USER

User Name	Authentication Method	MongoDB Roles	Actions
eva	SCRAM	readWriteAnyDatabase@admin	EDIT DELETE


CONTEXT
Project 0

ATLAS
Clusters
Data Lake BETA

SECURITY
Database Access
Network Access
Advanced

IPCA > PROJECT 0

Network Access

IP Whitelist | Peering | Private Endpoint

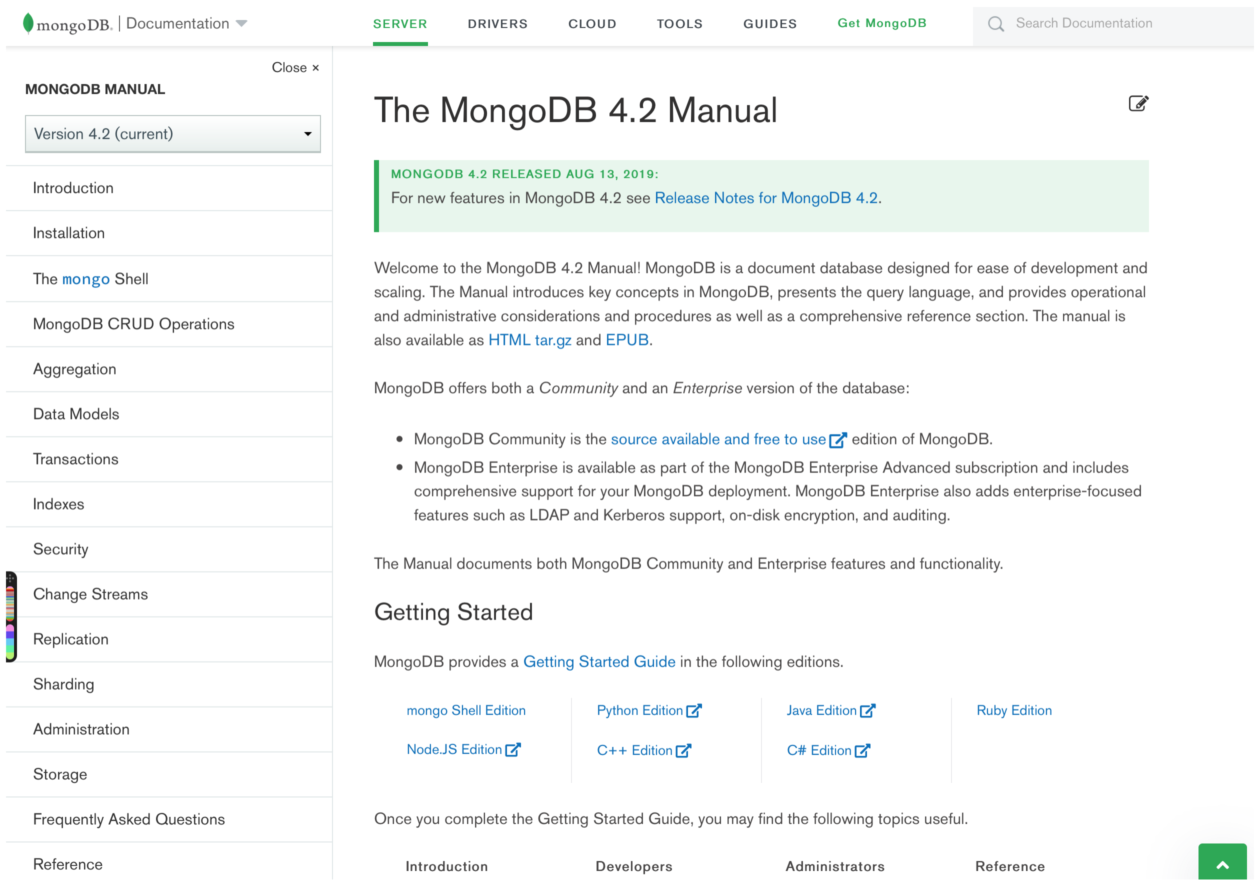
+ ADD IP ADDRESS

You will only be able to connect to your cluster from the following list of IP Addresses:

IP Address	Comment	Status	Actions
109.50.163.123/32		● Active	EDIT DELETE

We added our local IP to the "IP Whitelist". One important note on that: If you're away from the project for a couple of days (or maybe even after one day), you might've received a new local IP by your internet provider. Hence you should update that "IP Whitelist" if you're facing any connection issues!

In the next section we are going to connect to mongo, via Mongoose, a package that helps to perform some operations, but if you want to use only node, you can check this page, and follow the link NodeJs Edition.



The MongoDB 4.2 Manual

MONGODB 4.2 RELEASED AUG 13, 2019:
For new features in MongoDB 4.2 see Release Notes for MongoDB 4.2.

Welcome to the MongoDB 4.2 Manual! MongoDB is a document database designed for ease of development and scaling. The Manual introduces key concepts in MongoDB, presents the query language, and provides operational and administrative considerations and procedures as well as a comprehensive reference section. The manual is also available as HTML tar.gz and EPUB.

MongoDB offers both a *Community* and an *Enterprise* version of the database:

- MongoDB Community is the source available and free to use edition of MongoDB.
- MongoDB Enterprise is available as part of the MongoDB Enterprise Advanced subscription and includes comprehensive support for your MongoDB deployment. MongoDB Enterprise also adds enterprise-focused features such as LDAP and Kerberos support, on-disk encryption, and auditing.

The Manual documents both MongoDB Community and Enterprise features and functionality.

Getting Started

MongoDB provides a Getting Started Guide in the following editions.

mongo Shell Edition	Python Edition	Java Edition	Ruby Edition
Node.JS Edition	C++ Edition	C# Edition	

Once you complete the Getting Started Guide, you may find the following topics useful.

Introduction	Developers	Administrators	Reference

Adding Mongoose

npm install --save mongoose

mongoose

elegant mongodb object modeling for node.js

read the docs   discover plugins

Star 20,615   Version 5.9.10   Fork 2,782

Let's face it, **writing MongoDB validation, casting and business logic boilerplate is a drag**. That's why we wrote Mongoose.

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/test', {useNewU

const Cat = mongoose.model('Cat', { name: String });

const kitty = new Cat({ name: 'Zildjian' });
kitty.save().then(() => console.log('meow'));
```

Mongoose provides a straight-forward, schema-based solution to model your application data. It includes built-in type casting, validation, query building, business logic hooks and more, out of the box.

Get Professionally Supported Mongoose

2. Creating a model

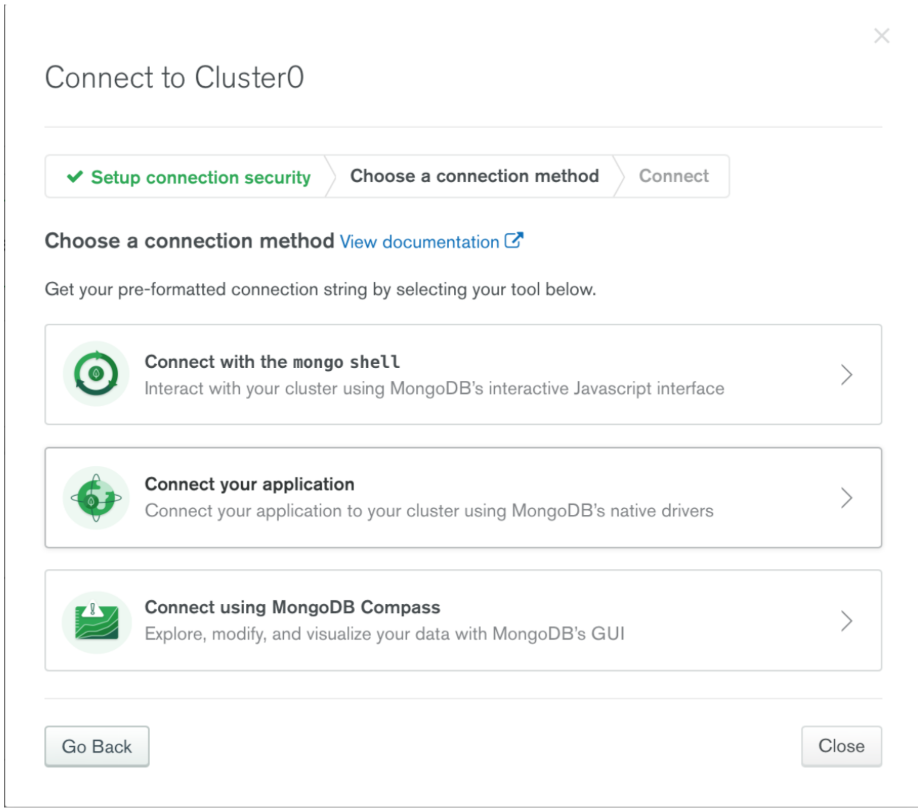
`models/post.js`

```
const mongooose = require("mongoose");

const postSchema = mongoose.Schema({
 title: { type: String, required: true },
 content: { type: String, required: true },
});

module.exports = mongoose.model("Post", postSchema);
```

Create a bd in the cloud



Connect to Cluster0

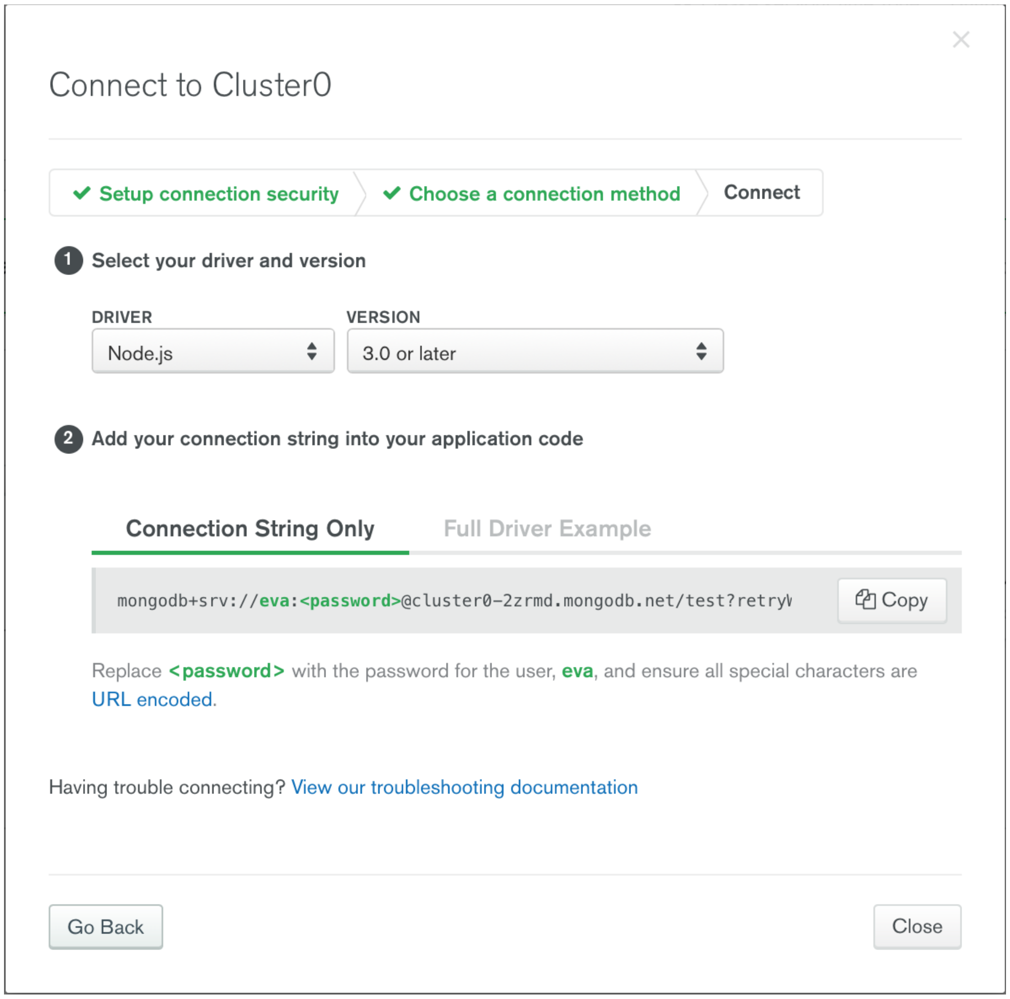
Choose a connection method View documentation

Get your pre-formatted connection string by selecting your tool below.

Connect with the `mongo shell`
Interact with your cluster using MongoDB's interactive Javascript interface

Connect your application
Connect your application to your cluster using MongoDB's native drivers

Connect using MongoDB Compass
Explore, modify, and visualize your data with MongoDB's GUI



Connect to Cluster0

1 Select your driver and version

DRIVER: Node.js

VERSION: 3.0 or later

2 Add your connection string into your application code

```
mongodb+srv://eva:<password>@cluster0-2zrmd.mongodb.net/test?retryW
```

Replace <password> with the password for the user, eva, and ensure all special characters are URL encoded.

Having trouble connecting? View our troubleshooting documentation

You need to copy the connection string

mongodb+srv://eva:\<password\>@cluster0-2zrmd.mongodb.net/test?retryWrites=true&w=majority

Use Compass to manage the database.

3. Connecting to bd

backend/app.js

```
const express = require("express");
const app = express();
const bodyParser = require("body-parser");
const mongoose = require("mongoose");

const Post = require("./models/post");

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));

mongoose
  .connect(
 "mongodb+srv://eva:<password>@cluster0-2zrmd.mongodb.net/test?retryWrites=true&w=majority",
    { useNewUrlParser: true, useUnifiedTopology: true }
  )
  .then(() => {
    console.log("connected do db");
  })
  .catch(() => {
    console.log("Connection failed");
  });


app.use((req, res, next) => {
  res.setHeader("Access-Control-Allow-Origin", "*");
  res.setHeader(
    "Access-Control-Allow-Headers",
    "Origin, X-Requested-With, Content-Type, Accept"
  );
  res.setHeader(
    "Access-Control-Allow-Methods",
    "GET, POST, PUT, DELETE, PATCH, OPTIONS"
  );
  next();
});
```

```
app.post("/api/posts", (req, res, next) =>
  console.log(req.body);
  res.status(201).json({
    message: "Post added",
  });
});


app.get('/api/posts', (req, res, next) => {
	const posts = [
		{
			id: '23hr23r8',
			title: 'First server-side post',
			content: 'This is coming from the server'
		},
		{
			id: 'fd7yfdyf',
			title: 'Second server-side post',
			content: 'This is also coming from the server'
		}
	];
	res.status(200).json({
		message: 'Post sent with success!',
		posts: posts
	});
});
```

Adding a post to the database

backend/app.js

```
….

app.post("/api/posts", (req, res, next) => {

  const post = new Post({
    title: req.body.title,
    content: req.body.content,
  });
  console.log(post);
  post.save();
  res.status(201).json({
    message: "Post added",
  });
});



….

module.exports = app;
```

Fetching posts from the database

backend/app.js

```
….

app.get("/api/posts", (req, res, next) => {
  Post.find()
    .then((documents) => {
      res.status(200).json({
        message: "Post sent with success!",
        posts: documents,
      });
    })
    .catch(() => {
      res.status(201).json({
        message: "There was an error fetching posts",
      });
    });
});
….

module.exports = app;
```

Exercise

Try to program the delete. Note that you have to send and id.