LAB 11 – REST API

In this lab we are going to create a RESTful API as a DBMS to our MongoDB database, which we created in the last Lab.

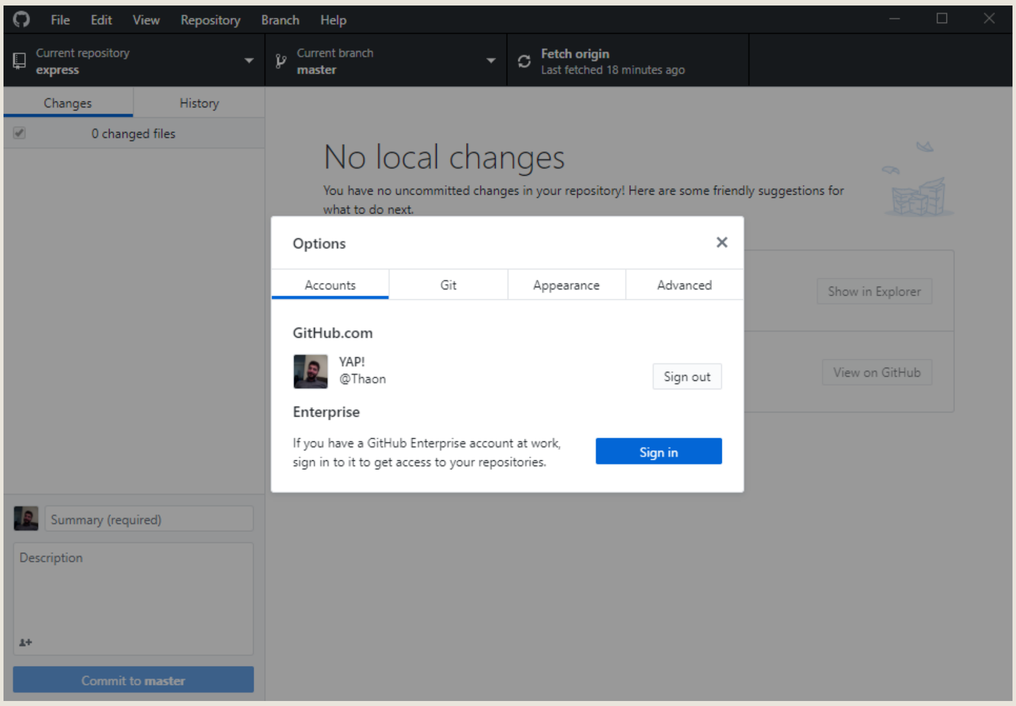
In that API we are going to specify the endpoints and behaviours of the DBMS and they will regulate how we define our data structures, in anticipation to connect everything together.

The last step of our journey will consist of modifying the Notes App created in Lab 6 so that our notes will be saved and retrieved online!

Part 1 – Requirements Setup

As our first step, we will clone the GitHub project containing our server into a folder on the desktop.

- Open GitHub Desktop.
- Make sure you are the user connected by Clicking on **File > Options**



No local changes

You have no uncommitted changes in your repository! Here are some friendly suggestions for what to do next.

GitHub.com

YAP!
@Thaon

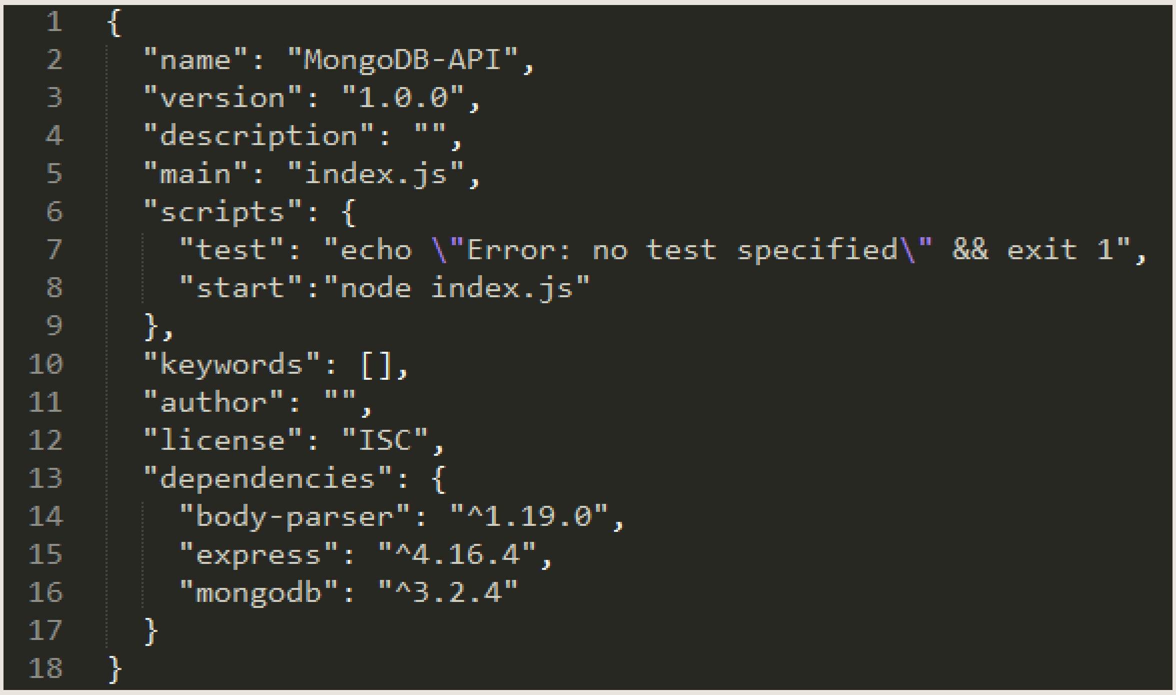
Enterprise

If you have a GitHub Enterprise account at work, sign in to it to get access to your repositories.

Remember that every time you push changes to this Repo, your online server will be updated by Zeit in real time, after a few seconds!

After this project has been cloned, do the following:

- locate it on the desktop and delete ALL the files inside the project's folder.
- Open the program **"Sublime Text Editor"** and type the following in it:



```
1  {
2    "name": "MongoDB-API",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1",
8      "start":"node index.js"
9    },
10   "keywords": [],
11   "author": "",
12   "license": "ISC",
13   "dependencies": {
14     "body-parser": "^1.19.0",
15     "express": "^4.16.4",
16     "mongodb": "^3.2.4"
17   }
18 }
```

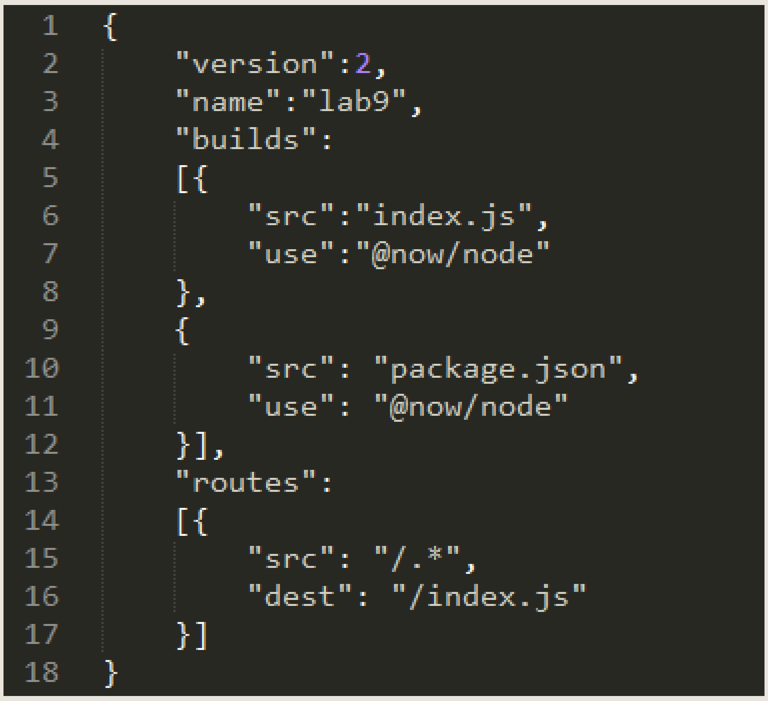
- Save this file as **"package.json"** in the Repo's folder.

This file is going to tell Zeit what packages are needed to be installed on the server and what command needs to be run when the server is started.

Note that in line 8 we are setting up the start command for Zeit to run **node** on the **index.js** file, we don't have that file yet, but we will soon.

Line 13 sets up the dependencies to be installed, Zeit will perform this installation automatically, but only if we specify it in the configuration file, let's do that now!

- In **Sublime** Click on **File > New File** and type the following in it:



```
1  {
2      "version":2,
3      "name":"lab9",
4      "builds":
5      [{
6          "src":"index.js",
7          "use":"@now/node"
8      },
9      {
10         "src": "package.json",
11         "use": "@now/node"
12     }],
13     "routes":
14     [{
15         "src": "/.*",
16         "dest": "/index.js"
17     }]
18 }
```

We can now save this file in the Repo's folder and call it **"now.json"**.

In line 3 we specify the name of the project.

In line 4 through 12, we specify how to build the **"index.js"** and **"package.json"** files, we do that by using **node**.

In line 13 we specify the **ROUTES** that will have access to our Server, and where they should be directed: in this case, all routes should divert inside **"/index.js"** which is, naturally, our Server.

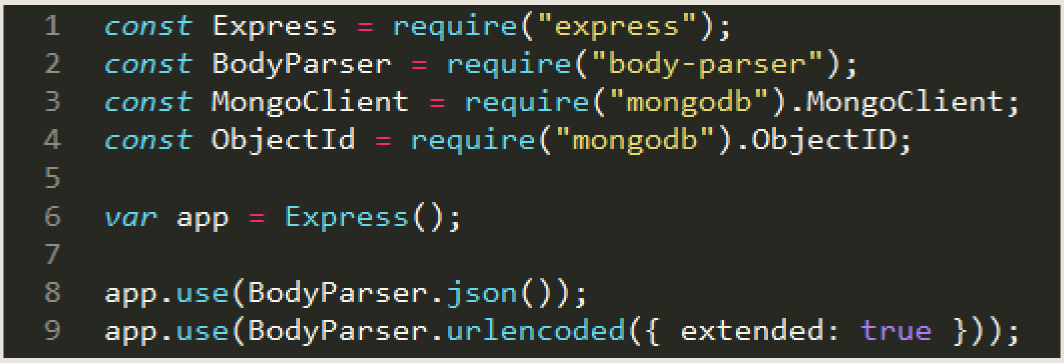
Part 2 – Creating the API

So far, so good, let's now create a new file in Sublime and save it as **index.js** inside the Repo's folder again.

We are going to use this file as our server, by following these steps:

- Import all the necessary libraries
- Test a route by using a default return to the "**/hi**" endpoint
- Setup a connection to our MongoDB database
- Create handlers for the following operations:
  - Get all notes
  - Create a new note
  - Get a specific note

Let's start by importing the necessary libraries in the **index.js** file:



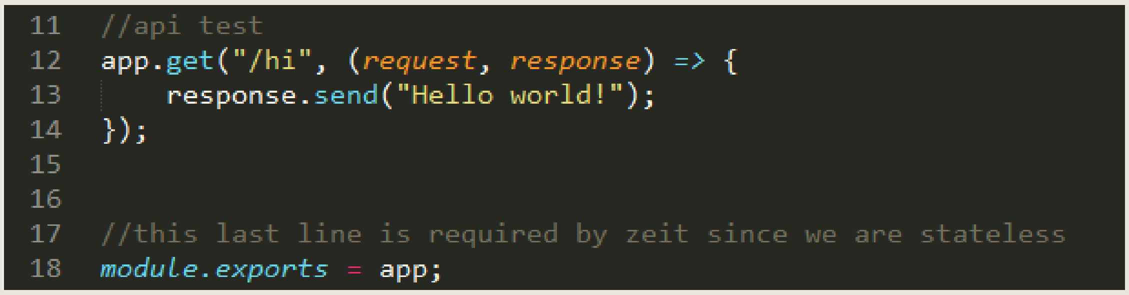
```
const Express = require("express");
const BodyParser = require("body-parser");
const MongoClient = require("mongodb").MongoClient;
const ObjectId = require("mongodb").ObjectID;

var app = Express();

app.use(BodyParser.json());
app.use(BodyParser.urlencoded({ extended: true }));
```

We are also instantiating an object of the express class and setting it up to use the JSON and URL encoded body parsers. We need them to access HTTP responses!

Once this is done, let's create a testing ROUTE and export our app for Zeit to use:

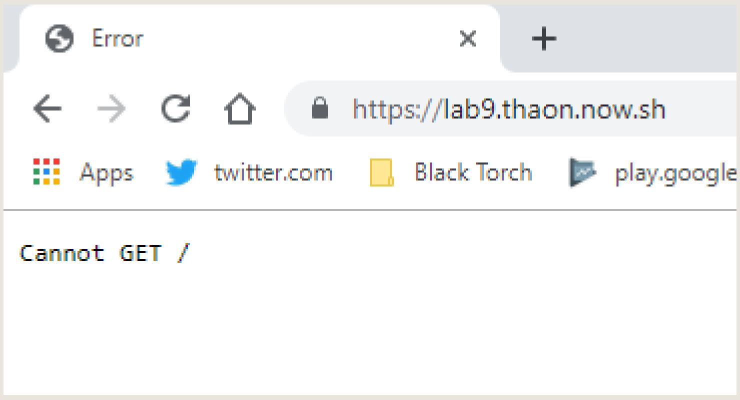


```
//api test
app.get("/hi", (request, response) => {
    response.send("Hello world!");
});


//this last line is required by zeit since we are stateless
module.exports = app;
```

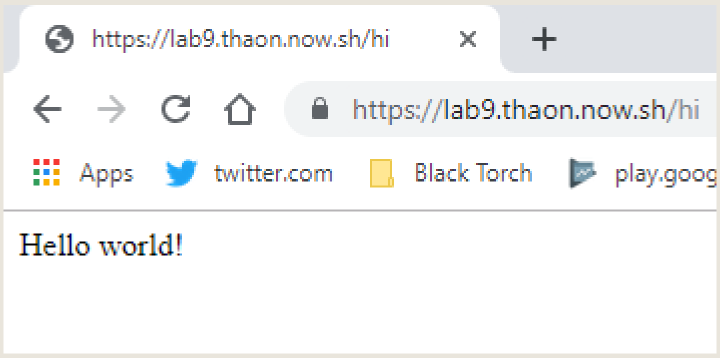
We can now commit the changes to GitHub and access our server online to test that this ROUTE works!

Go into your Zeit dashboard and click on your server to check that it runs, it should present you with the following screen:



```
Cannot GET /
```

This is because we don't have a handler for the "/" ROUTE, but if we add **"/hi"** at the end, we will instead see this:



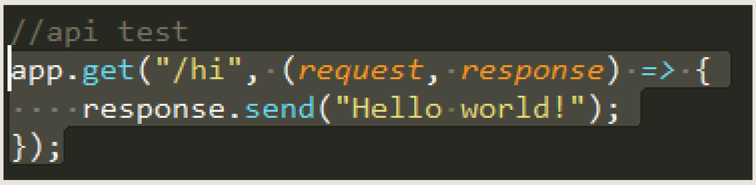
```
Hello world!
```

Yay, it works!

Let's finish this lab by adding the remaining API endpoints and handlers.

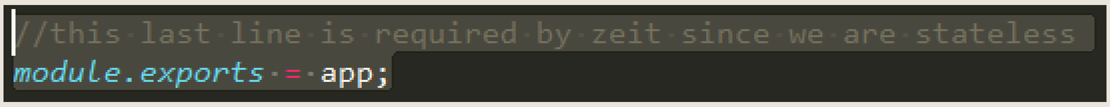
After this is done, we will test the functionality of our database by using custom **GET** and **POST** requests, these requests will be then used inside our App to gather data from the DataBase!

After the:



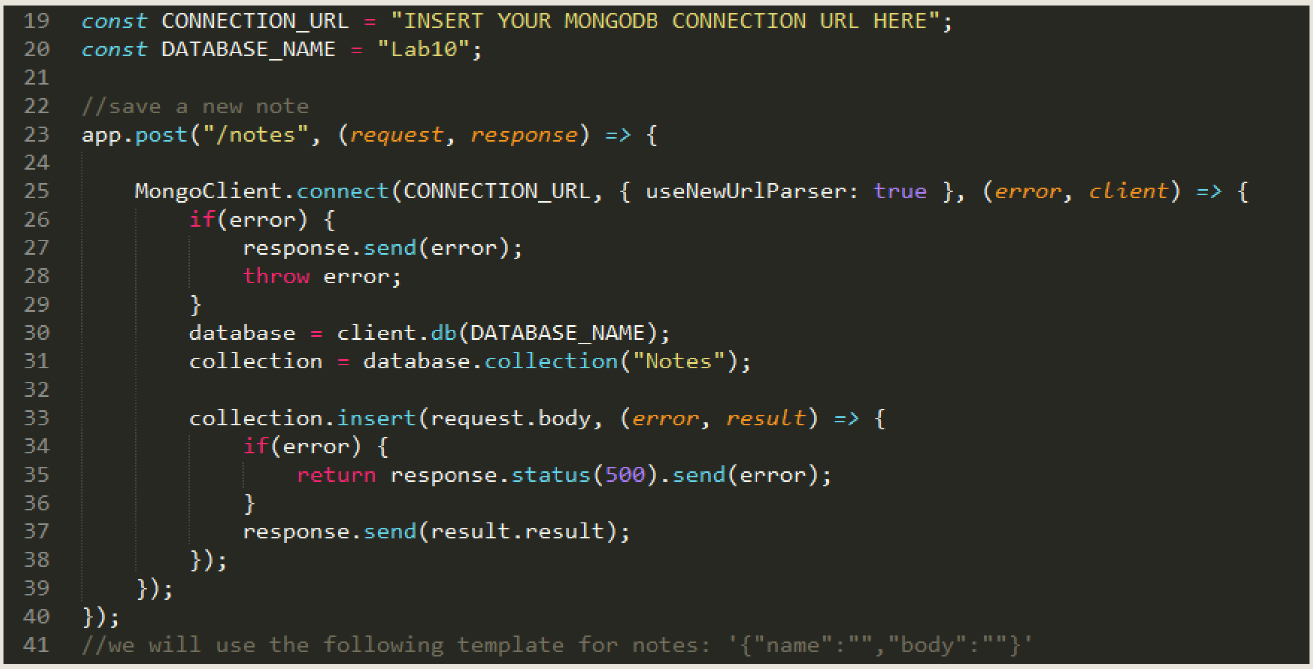
```
//api test
app.get("/hi", (request, response) => {
    response.send("Hello world!");
});
```

But before the:



```
//this last line is required by zeit since we are stateless
module.exports = app;
```

Let's write down the following functions:



```
const CONNECTION_URL = "INSERT YOUR MONGODB CONNECTION URL HERE";
const DATABASE_NAME = "Lab10";

//save a new note
app.post("/notes", (request, response) => {

    MongoClient.connect(CONNECTION_URL, { useNewUrlParser: true }, (error, client) => {
        if(error) {
            response.send(error);
            throw error;
        }
        database = client.db(DATABASE_NAME);
        collection = database.collection("Notes");

        collection.insert(request.body, (error, result) => {
            if(error) {
                return response.status(500).send(error);
            }
            response.send(result.result);
        });
    });
});
//we will use the following template for notes: '{"name":"","body":""}'
```

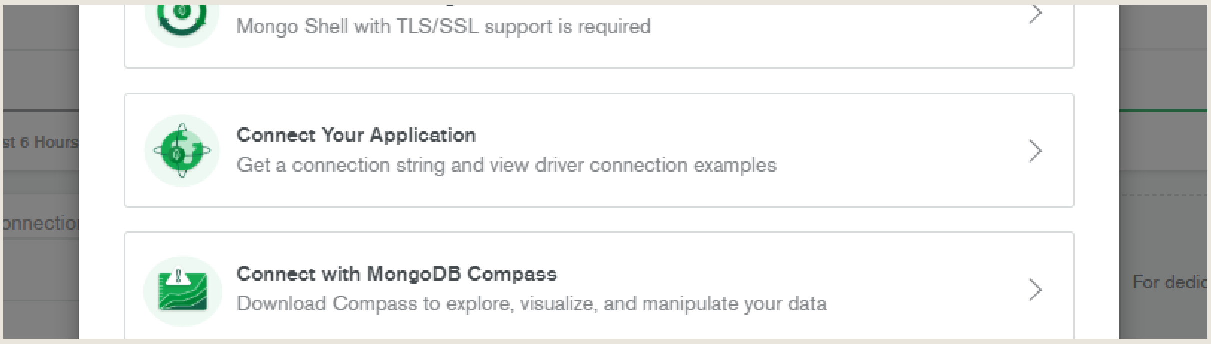
That will get all the notes stored in the **Notes collection** of our DataBase.

Note that this collection does not exist, but will be created as soon as we have a request to it, be it GET or POST.

We need to setup the connection to our DB though, let's go into the **MongoDB Atlas** dashboard again and click on the **CONNECT** button:

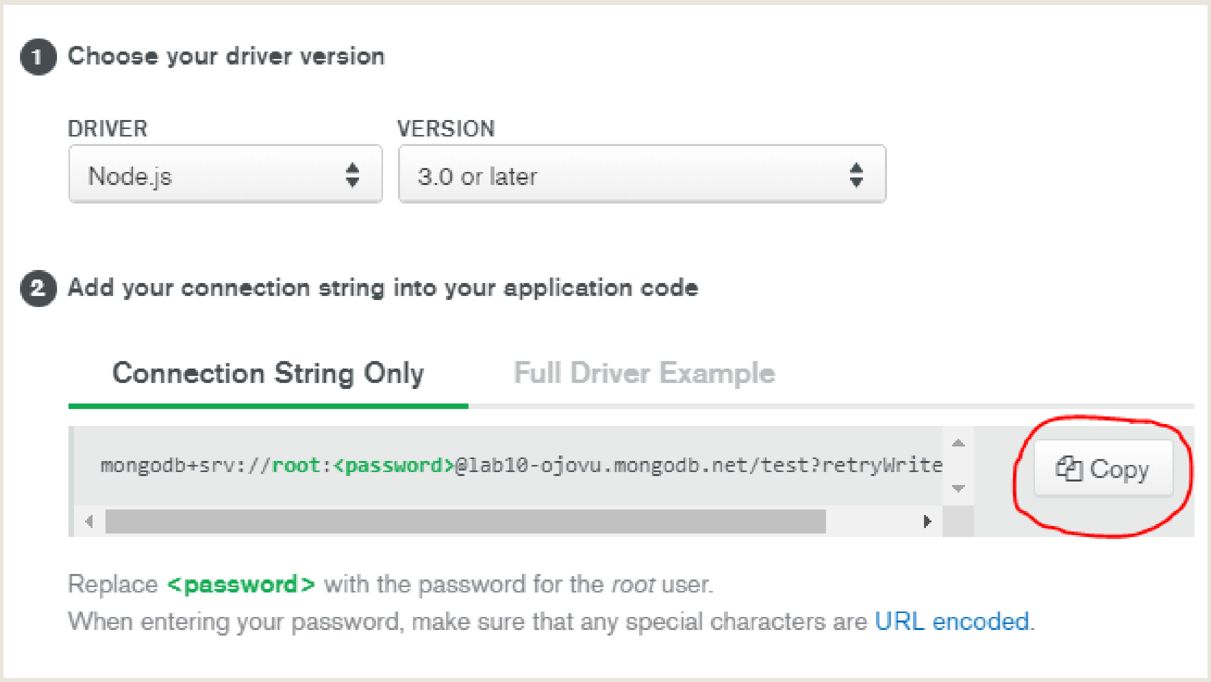
Note, the MongoDB Atlas dashboard can be accessed from: https://cloud.mongodb.com.

In there, select **CONNECT YOUR APPLICATION** and copy the link you are presented with, that link will be the string to feed to the **CONNECTION_URL** variable


Mongo Shell with TLS/SSL support is required

Connect Your Application
Get a connection string and view driver connection examples

Connect with MongoDB Compass
Download Compass to explore, visualize, and manipulate your data


1 Choose your driver version

DRIVER: Node.js
VERSION: 3.0 or later

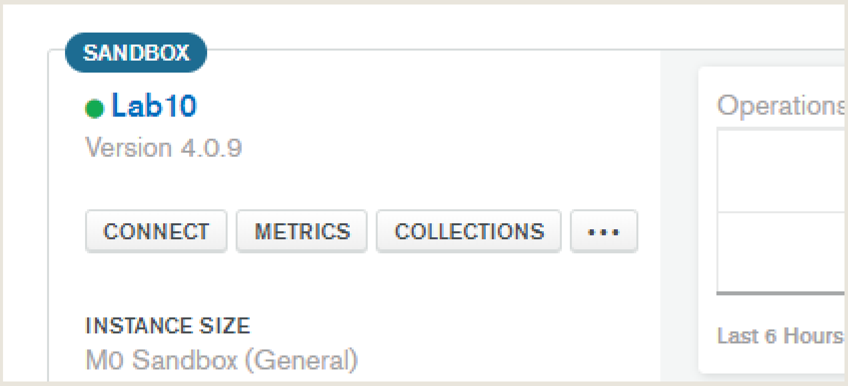
2 Add your connection string into your application code

Connection String Only | Full Driver Example

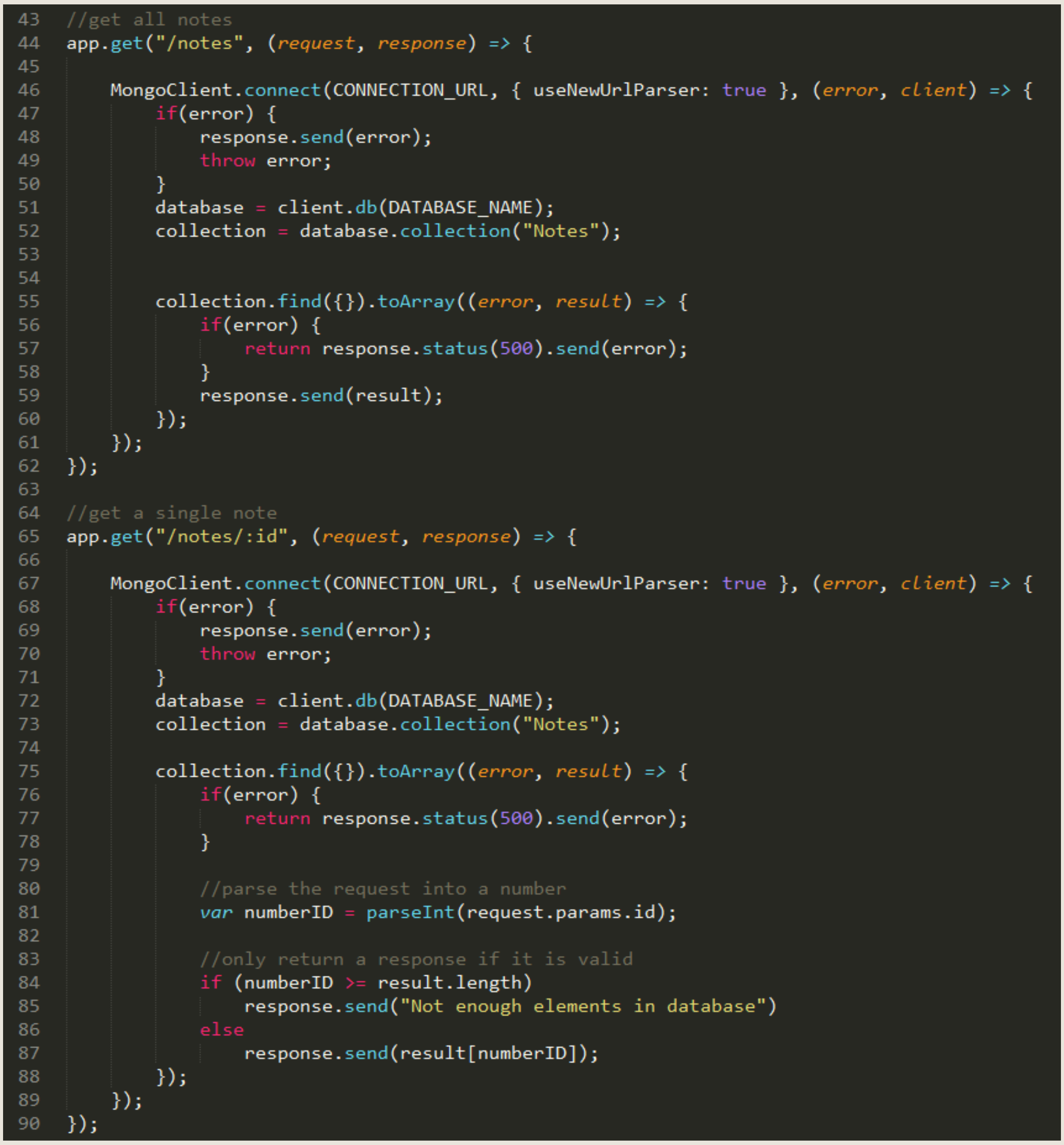
```
mongodb+srv://root:<password>@lab10-ojovu.mongodb.net/test?retryWrite
```

Replace <password> with the password for the *root* user.
When entering your password, make sure that any special characters are URL encoded.

Since we are here, let's also replace the **DATABASE_NAME** variable with the name of your DataBase. You can find that on the left.



That is all we need to do to connect to our database, so after this is done, we can write down the rest of our API:



```javascript
//get all notes
app.get("/notes", (request, response) => {

    MongoClient.connect(CONNECTION_URL, { useNewUrlParser: true }, (error, client) => {
        if(error) {
            response.send(error);
            throw error;
        }
        database = client.db(DATABASE_NAME);
        collection = database.collection("Notes");


        collection.find({}).toArray((error, result) => {
            if(error) {
                return response.status(500).send(error);
            }
            response.send(result);
        });
    });
});

//get a single note
app.get("/notes/:id", (request, response) => {

    MongoClient.connect(CONNECTION_URL, { useNewUrlParser: true }, (error, client) => {
        if(error) {
            response.send(error);
            throw error;
        }
        database = client.db(DATABASE_NAME);
        collection = database.collection("Notes");

        collection.find({}).toArray((error, result) => {
            if(error) {
                return response.status(500).send(error);
            }

            //parse the request into a number
            var numberID = parseInt(request.params.id);

            //only return a response if it is valid
            if (numberID >= result.length)
                response.send("Not enough elements in database")
            else
                response.send(result[numberID]);
        });
    });
});
```

That's it, we are done, let's commit and push our changes, we now should have our server up and running, to check that it works, we can send a request by using the **CURL** format, let's try this out using the website: https://onlinecurl.com/.

Try and enter the following **CURL**s in the form:

curl https://yourServerAddress /notes

curl -d "{'name':'Test', 'body':'This is a test message'}"
https://yourServerAddress/notes

Then this again:

https://yourServerAddress /notes

Then

https://yourServerAddress /notes/0

https://yourServerAddress /notes/1

See how our API responds to all these requests? In the next lab, we are going to integrate online functionality to our Notes App!