Gautier Lemaitre 22870 https://github.com/Gautier78/back_end_dorset

1)Set up the environment

In our folder project:

First, we must set up our project and the environment. We have multiple commands to write in the terminal:

- *npm init (to initialize your Node.Js project)*
- *npm install express (to install express for routes)*
- *npm install mongodb (to install the mongodb database driver)*
- *npm install bodyparser (to be able to use multiple http requests)*
- *npm install –g nodemon (restarts the application when you save the document)*

On Github:

We create a new repository on Github, the mine is named “back_end_dorset”

Again, we have multiple commands to write in the terminal:

- *git init (only run once)*
- *git add .*
- *git remote add origin “your GitHub repository link” (only run once)*
- *git commit –m “your comment”*
- *git push origin master*

On GitPod:

In my case, I didn’t use GitPod. We can use it if we need help. Just write “gitpod.io/# “ before your repository link. So, anyone that have access can modify your code.

On MongoDB:

Once logged, you can create a new project on the top left corner. In this project you must create a new cluster. In this cluster, select AWS Cloud, the Ireland region and choose the free options. You can rename the cluster.

You will have the option to change the ip address. Use 0.0.0.0/0 or your Own ip to allow the connection with the database.

Then choose the second option “Connect your application” and copy/paste the long link in your main JavaScript file of your project.

The link look like this:

mongodb+srv://gautier:<password>@clusterbackend.wcefp.mongodb.net/<dbname>?retryWrites=true&w=majority

In yellow, this is the name of the admin, in green you have the password and the database name to replace

2) Let's start coding

Code/template breakout:

We have been using this code at each backend course:

```
const express = require('express')                    // require the Express library and
const app = express()                                 //
const { MongoClient, ObjectId } = require("mongodb");  //the MongoDb driver

const bodyParser = require('body-parser')             //to use all the Http requests
app.use(bodyParser.urlencoded({ extended: false}))    //we need to require body-parser
```

It allows us to use the Express library, the MongoDb driver and Http requests.

Connection to the Database:

We can see again the long link. Just under I created an instance of Mongo Client and a function dbName equal to "building" because it's the name that I've given to my database.

```
const url = "mongodb+srv://gautier:backend@clusterbackend.wcefp.mongodb.net/building?retryWrites=true&w=majority"
const client = new MongoClient(url, {useUnifiedTopology: true}); //create an instance of MongoClient
let col, data

// The database to use
const dbName = "building";  //building is the name of my DB in mongoDB
```

Async function Run():

wrap the calls to functions that interact with the database in a try/catch statement so it can handle any unexpected errors.

If the connection succeed, the function run the code in the Try but if the connection fail, the function run the code in the catch.

```
async function run() {
    try {
        await client.connect();//await  indicate that we should block further execution until the connexion has been completed
        // client.connect() return a promise (represents the eventual completion (or failure) of an asynchronous operation and its resulting value)
        console.log("Connected correctly to server"); //if we succeed to connect to the server, this message is displayed
        col = client.db(dbName);
        data = col.collection("house")//link to the table
        app.listen(3000);// invoke the callback to connect to mongo database

    } catch (err) {     //if there is an unexpected error...
        console.log(err.stack); //print the error in the console
    }


}
```

The class

Create a class and give it a name (in UpperCamelCase). In this class we build a special method named constructor, to create and initiate an object.

The parameters of the constructors are the column names that fills the table “house”.

```
class House {           //create the class named House
    constructor(nbrOfRoom, garage = false , m2, streetName){ //create and initiate an object
        this.nbrOfRoom = nbrOfRoom;
        this.garage = garage;
        this.m2 = m2;
        this.streetName = streetName;
    }

    printValues(){
        console.log(this.nbrOfRoom, this.garage, this.m2, this.streetName);
    }
}
```

Crud Operations

Create:

I create a new house and I define on Postman the nbrOfRoom, if there is a garage, the number of square meter and the name of the street. Once I have set the values on Postman, I send the request and it create a new house.

```
//Create operation
app.post('/building', (req,res) => {
    async function postBuilding(){
        let house = new House (req.body.nbrOfRoom,req.body.garage,req.body.m2,req.body.streetName)//we will set the values of these elements in postman
        data.insertOne(house);//insert in the bdd the new house we add with postman
        res.send(200);//it says that the request as been done
    }
    postBuilding();//run the function that Insert a new house in the bdd
})
```

Read:

- All:

On the Url “building” all my database is read.

```
//Read operation
//simply display my bdd. it get all my objects
app.get('/building', (req,res) => {    //bdd displayed on the url localhost:3000/building
    async function getBuilding(){
        const get = await data.find().toArray();//read all the elements in my bdd
        res.json(get);
    }
    getBuilding();//run the function that Read all my bdd
})
```

- One:

On the Url “building/id” replace the “id” in the url by a real id number and it will read the house link to this id.

```
//Read operation
app.get('/building/:id', (req,res) => {  //bdd displayed on the url localhost:3000/building/id
    async function getBuilding(){
        const find = await data.findOne({"_id": ObjectId(req.params.id)})//I read one object by his id
        res.json(find);
    }
    getBuilding();//run the function that Read one element in my bdd thanks to his Id
})
```

Update:

In postman, you have to select the id of the house you want to update. Then, you have to modify the option that you want to change. For exemple, you can select numberOfRoom that was equal to 6 before the update. You set it to 4 and then send the PUT request with postman. You will see in your database that the value has been updated.

```
//Update operation
app.put('/building', (req,res) => {
    async function getBuilding(){
        const find = await data.findOne({"_id": ObjectId(req.body.id)})
        let house = new House(find.nbrOfRoom, find.garage, find.m2, find.streetName)//we will Update the values of these elements in postman
        house.nbrOfRoom = req.body.nbrOfRoom,
        house.garage = req.body.garage,
        house.m2 = req.body.m2,
        house.streetName = req.body.streetName

        await data.updateOne({"_id": ObjectId(req.body.id)}, {$set: house})
    }
getBuilding();//run the function that Update elements in my bdd
})
```

Delete:

You can delete with postman a house thanks to his Id number. Once you have choose, send the request DELETE and it's done.

```
//Delete operation
app.delete('/building', (req,res) => {
    data.deleteOne({"_id": ObjectId(req.body.id)})//I delete one object by his id thanks to postman
})
```

Sources:

- The link underneath helped me to comment my code especially for the code that is about the database connexion or operations.

  https://www.mongodb.com/blog/post/quick-start-nodejs-mongodb--how-to-get-connected-to-your-database

- To understand what app.listen(3000) do.

  https://stackoverflow.com/questions/55984104/connection-to-mongodb-server-using-node-is-unresponsive-on-localhost

- Crud operations

  https://developer.mongodb.com/quickstart/node-crud-tutorial#std-label-node-tutorial-update