

# Aymane Hinane 4G51 Emsi centre




## Configurer mon app

1# installer les packages dont on aurait besoin :

```
npm install express
npm install dotenv
npm install mongodb
npm install nodemon
npm install mongoose
npm install body-parser
```

```
{ } package.json > ...
1   {
2     "dependencies": {
3       "body-parser": "^1.20.1",
4       "dotenv": "^16.0.3",
5       "express": "^4.18.2",
6       "mongodb": "^4.11.0",
7       "mongoose": "^6.8.0",
8       "nodemon": "^2.0.20"
9     },
10    "name": "tp2",
11    "version": "1.0.0",
12    "main": "app.js",
13    "scripts": {
14      "test": "echo \"Error: no test\"",
15    },
16    "author": "",
17    "license": "ISC",
18    "description": ""
19  }
20
```

## 2# créer un serveur

```
js app.js >  app.listen() callback
1  const express = require("express"); //1
2  const app = express(); //2
3  const mongoose = require("mongoose"); //3
4  const route = require("./routes/productRoute"); //4
5  require ("dotenv").config();//5
6
7  const uri = process.env.MONGO_URL;//6
8  const port = 8000;//7
9
10 const options = { //8
11   useNewUrlParser: true,
12   useUnifiedTopology: true
13 };
14
15 mongoose.connect(uri ,{dbName:"db_catalogue"}).then(() => { //9
16   |   |   console.log("Database connection established!");
17   |   },
18   |   err => {
19   |   |   {
20   |   |   |   console.log("Error connecting Database instance due to:", err);
21   |   |   }
22   |   });
23
24 app.use(express.json()); //10
25 app.use(express.urlencoded({extended:true})); //11
26
27 app.use("/product",route);//12
28
29 app.listen(process.env.PORT, () =>  //13
30   |   console.log(`Example app listening on port ${port}`);|
31   |   );
32
```

1 – on importe le module express, se module contient la class express

Express est un framework javascript base sur node.js qui simplifie le processus de codage

2-on instancie la class express, est on stock l'objet instancier dans la variable app

3-mongoose est un ORM (object data modeling),il permet de simplifier l'accès a la base de donne

Parmi c'est fonctionnalité :

- Migrer le model vers le cluster est créer une Collection correspondant
- Ajouter un document a ma collection
- S'simplifier l'écriture des requête (no sql)
- Il gère la cohérence, puisque mongo db est une base de donne no sql, donc mes données seront stocker selon la même structure de mon model

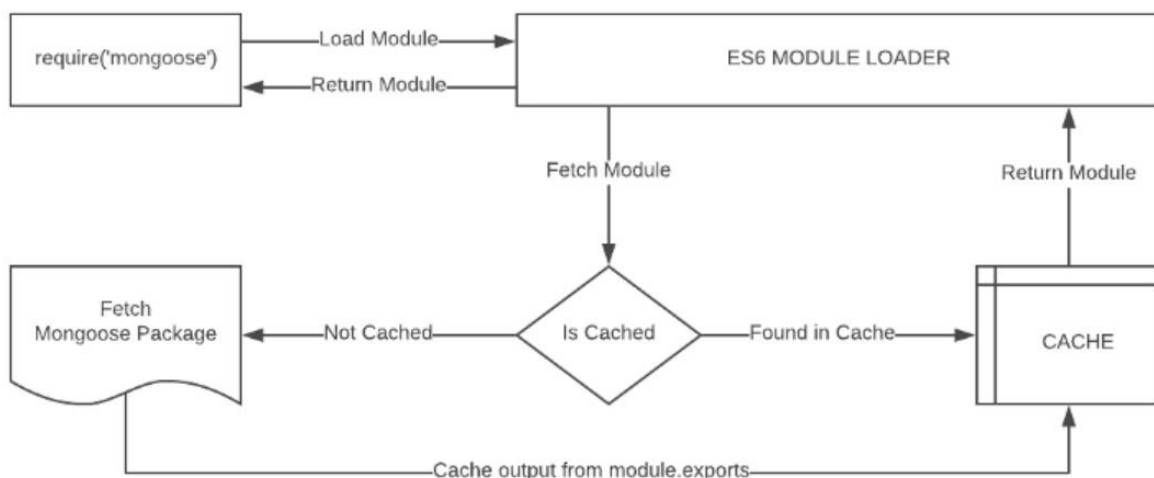
4- j'importe le module route qui contient tous les chemin vers mes Controller  
La fonction route serait passer comme paramètre a un middleware

5-6 le module dotenv permet d'accéder au fichier de paramètre .env pour récupérer la chaine de connexion de ma base de donne ainsi que le numero de port qui serait utiliser par mon app

9- je fais appelle a la fonction connect de l'objet mongoose, cette fonction prend 2 paramètre

- L'url de connexion a ma bd
- Le nom de ma collection (base de donne)

La fonction connect me connecte au document de mon cluster (une instance de connexion serait créer), puisque l'objet mongoose est un objet singleton, donc il serait accessible globalement



## Module import/require work-flow

La méthode connect est une méthode asynchrone

Donc elle return une promesse

Si la connexion est valide , resolve serait retourner (resolve est capturer par l'objet then)

Si la connexion a ma BD échoue , reject serait retourner (reject est une error donc le block catch() va intercepter l'erreur)

NB :

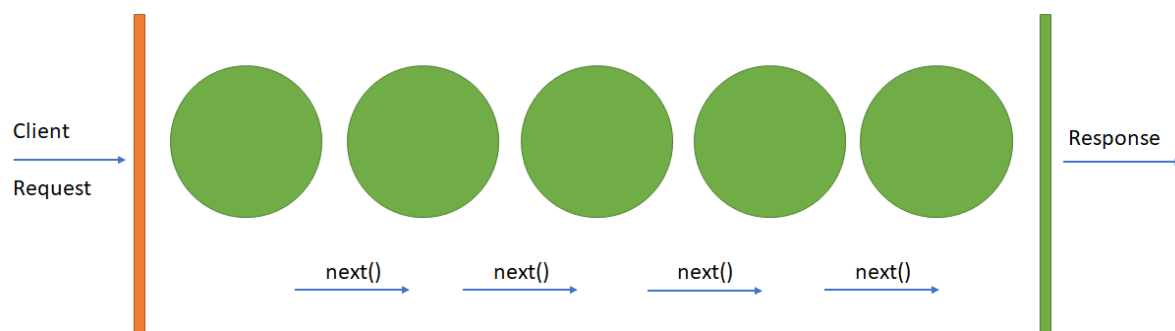
Resolve et reject sont eux même des promesse

10-11 c'est 2 ligne permette de récupérer le body de mon url on fessant un parse

12 – la méthode use est une middleware

Une middleware est une fonction qui accède a l'objet res et req sont role est de faire des modification sur c'est 2 object et elle gère aussi le cycle d'exécution de ma requête

All middleware has access to req,res and next



Next() permet de donne la main a la prochaine middelware stocker dans ma pile

13 – c'est le port qui serait utiliser par mon application pour écouter les requête

3# créer un model

```
1  var mongoose = require('mongoose');
2
3  //const { Schema } = mongoose;
4
5  const productSchema = mongoose.Schema({
6    name:String,
7    description:String,
8    price:Number
9  })
10
11
12  const Product = mongoose.model('Product',productSchema);
13
14  module.exports = Product;
```

Mongoose.Schema : cree un shema qui serat lier a ma collection, se shema me permet de donner une forme au document qui serat ajouter a ma collection

Mongoose.model : permet de mapper mon shema (document) a ma collection lors de sa creation.

NB : l'id de mon document serat generer automatiquement

4# créer un router

```
1
2  const express = require("express");
3  const router = express.Router();
4  const productController = require("../controllers/productController");
5
6  router.post("/", productController.AddProduct);
7  router.get("/", productController.getAllProducts);
8  router.get("/:id", productController.getProductById);
9  router.delete("/:id", productController.deleteProductById);
10 router.put("/:id", productController.updateProductById);
11
12 module.exports = router;
```

Le router est une middleware qui vat intercepter l'objet request, et se dernier vat vérifier le path et ainsi que la méthode de l'objet request

-le middleware qui serait exécuter serait celui qui a la même méthode request et le même path  
La fonction exécuter est un Controller

## 5# créé un Controller

un Controller sert implémenter les services de mon application

### Ajouter un produit

```
4  const AddProduct = (req,res)=>{  
5  
6      var name = req.body.name;  
7      var description = req.body.description;  
8      var price = req.body.price;  
9  
10     var NewProduct  = new product({...req.body});  
11  
12     NewProduct.save(function(err,book){  
13         if(err) return res.status(500).json(err);  
14         res.status(200).json(NewProduct);  
15     });  
16 };  
17
```

L'objet req.body sert a récupérer les valeur stocker dans l'objet envoyer dans le body contenue dans l'objet request

{...req.body} = vat copier le contenue de l'objet body dans un nouveau objet

Save est une méthode de l'objet model, elle permet d'ajouter un nouveau document a ma collection

.status définie le status de ma réponse

.json() dans mon body un fichier json serait envoyer contenant le résultat

http://localhost:8000/product

POST

http://localhost:8000/product

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

```
1 {
2   ... "name": "hp omen 15",
3   ... "description": "most powerfull gaming laptop on the world",
4   ... "price": 18000
5 }
```

Body

Cookies

Headers (7)

Test Results

Pretty

Raw

Preview

Visualize

JSON

```
1 {
2   "name": "hp omen 15",
3   "description": "most powerfull gaming laptop on the world",
4   "price": 18000,
5   "_id": "63909aca07e97e665fde85bb",
6   "__v": 0
7 }
```

Overview

Real Time

Metrics

Collections

Search

Profiler

Performance Advisor

Online Archive

Cmd Line Tools

DATABASES: 1 COLLECTIONS: 1

VISUALIZE YOUR DATA

REFRESH

+ Create Database

Search Namespaces

db\_catalogue

products

db\_catalogue.products

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 122B TOTAL DOCUMENTS: 1 INDEXES TOTAL SIZE: 36KB

Find

Indexes

Schema Anti-Patterns

Aggregation

Search Indexes

INSERT DOCUMENT

FILTER { field: 'value' }

OPTIONS

Apply

Reset

QUERY RESULTS: 1-1 OF 1

```
_id: ObjectId('63909aca07e97e665fde85bb')
name: "hp omen 15"
description: "most powerfull gaming laptop on the world"
price: 18000
__v: 0
```



## Récupérer produit par Id

```
const getProductById=(req, res)=>{
  const idP=req.params.id;
  product.findById (idP)
    .then (result=>res.status(200).json(result))
    .catch (error=>res.status(500).json(error));
};
```

http://localhost:8000/product/63909aca07e97e665fde85bb

GET

⌵

http://localhost:8000/product/63909aca07e97e665fde85bb

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Query Params

	KEY	VALUE
	Key	Value

Body

Cookies

Headers (7)

Test Results

Pretty

Raw

Preview

Visualize

JSON ⌵

⌵

```
1 {
2   "_id": "63909aca07e97e665fde85bb",
3   "name": "hp omen 15",
4   "description": "most powerfull gaming laptop on the world",
5   "price": 18000,
6   "__v": 0
7 }
```



## Récupère tous les produits

http://localhost:8000/product

**GET** ⌵ http://localhost:8000/product

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Query Params

KEY	VALUE
Key	Value

Body

Cookies

Headers (7)

Test Results

Pretty

Raw

Preview

Visualize

JSON ⌵

↻

```
1  [
2    {
3      "_id": "63909aca07e97e665fde85bb",
4      "name": "hp omen 15",
5      "description": "most powerfull gaming laptop on the world",
6      "price": 18000,
7      "__v": 0
8    }
9  ]
```

## Modifier un produit

```
38
39  const updateProductById=( req,res)=>{
40
41      const idP=req.params.id;
42
43      product.findByIdAndUpdate(req.params.id,req.body)
44      .then (result=>res.status(200).json(result))
45      .catch (error=>res.status(500).json(error));
46  };
47
```

http://localhost:8000/product/63909aca07e97e665fde85bb

PUT



http://localhost:8000/product/63909aca07e97e665fde85bb

Params

Authorization

Headers (8)

Body ●

Pre-request Script

Tests

Settings

● none

● form-data

● x-www-form-urlencoded

● raw

● binary

● GraphQL

JSON



```
1 {  
2   ... "name": "asus 15",  
3   ... "description": "most powerfull gaming laptop on the world",  
4   ... "price": 15000  
5 }
```

http://localhost:8000/product/63909aca07e97e665fde85bb

GET



http://localhost:8000/product/63909aca07e97e665fde85bb

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Query Params

	KEY	VALUE
	Key	Value

Body

Cookies

Headers (7)

Test Results

Pretty

Raw

Preview

Visualize

JSON



```
1 {  
2   "_id": "63909aca07e97e665fde85bb",  
3   "name": "asus 15",  
4   "description": "most powerfull gaming laptop on the world",  
5   "price": 15000,  
6   "__v": 0  
7 }
```

DATABASES: 1 COLLECTIONS: 1

VISUALIZE YOUR DATA

REFRESH

+ Create Database

Search Namespaces

db\_catalogue

products

## db\_catalogue.products

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 119B TOTAL DOCUMENTS: 1 INDEXES TOTAL SIZE: 36KB

Find

Indexes

Schema Anti-Patterns 0

Aggregation

Search Indexes 0

INSERT DOCUMENT

FILTER { field: 'value' }

OPTIONS

Apply

Reset

QUERY RESULTS: 1-1 OF 1

```
_id: ObjectId('63909aca07e97e665fde85bb')
name: "asus 15"
description: "most powerfull gaming laptop on the world"
price: 15000
__v: 0
```

Supprimer un produit

```
30
31 const deleteProductById=(req,res)=>{
32     const idP=req.params.id;
33     //product.deleteOne({_id:idP})
34     product.findByIdAndDelete(idP)
35     .then (result=>res.status(200).json(result))
36     .catch (error=>res.status(500).json(error));
37 };
```

http://localhost:8000/product/63909aca07e97e665fde85bb

DELETE



http://localhost:8000/product/63909aca07e97e665fde85bb

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Query Params

KEY

VALUE

Body

Cookies

Headers (7)

Test Results

Pretty

Raw

Preview

Visualize

JSON



```
1 {
2   "_id": "63909aca07e97e665fde85bb",
3   "name": "asus 15",
4   "description": "most powerfull gaming laptop on the world",
5   "price": 15000,
6   "__v": 0
7 }
```

http://localhost:8000/product

GET



http://localhost:8000/product

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Query Params

KEY

VALUE

Key

Value

Body

Cookies

Headers (7)

Test Results

Pretty

Raw

Preview

Visualize

JSON



```
1 []
```