

CHAPITRE 2



CRÉER DES APIS REST

Par :

- AIT TAMGHART Abdelghani
- DAHA Salahdine
- BEN HADDOU Redouane

Plan



1. | Introduction

2. | CRUD sur un fichier **Json**

3. | Tests avec **Postman**

4. | API REST et MongoDB

5. | CRUD sur **MongoDB**

6. | Le module **mongoose**



01

Introduction





Les API REST (Representational State Transfer) sont devenues un élément essentiel de l'architecture logicielle moderne. Elles permettent une communication facile et efficace entre les applications, en utilisant un protocole standardisé pour les échanges de données.

C'est pourquoi la création d'une API REST est devenue une compétence clé pour les développeurs web et les ingénieurs logiciels.

02

CRUD sur un fichier Json





Notre fichier JSON

```
{...} teams.json ●
project-2 > {...} teams.json > ...
1  [
2    {
3      "id": 1,
4      "name": "Barcelone",
5      "country": "Spain"
6    },
7    {
8      "id": 2,
9      "name": "PSG",
10     "country": "France"
11   },
12   {
13     "id": 3,
14     "name": "Real Madrid",
15     "country": "Spain"
16   },
```

```
17   {
18     "id": 4,
19     "name": "AC Milan",
20     "country": "Italy"
21   },
22   {
23     "id": 5,
24     "name": "Liverpool",
25     "country": "England"
26   },
27   {
28     "id": 6,
29     "name": "Marseille",
30     "country": "France"
31   }
32 ]
```

Importer les données

```
const teams = JSON.parse(fs.readFileSync("./teams.json"));
```



➤ GET (/teams)

La méthode HTTP GET est utilisée pour récupérer des données à partir d'un serveur en utilisant un endpoint spécifique. Dans ce code, l'endpoint est "/api/teams". La réponse peut être retournée dans différents formats tels que JSON, XML ou HTML.

```
app.get('/api/teams', (req, res) => {  
  try{  
    teams ?  
      res.status(200).json(teams)  
      :  
      res.status(404).send("No data available")  
    }  
  catch(err){  
    console.log("\x1b[31m%s\x1b[0m", err.message);  
    res.status(500).send({message: "Server error !", error: err.message});  
  }  
});
```

En général, la méthode HTTP GET ne devrait pas modifier l'état du serveur. Elle est utilisée pour récupérer des données uniquement.



➤ GET (/teams/:id)

Dans ce code, l'endpoint est `/api/teams/:id` et le paramètre `:id` est utilisé pour identifier l'équipe à récupérer. Si l'équipe est trouvée, elle est renvoyée sous forme de réponse JSON avec un code d'état 200. Sinon, une réponse "Not found" avec un code d'état 404 est renvoyée.

```
app.get('/api/teams/:id', (req, res) => {  
  const id = parseInt(req.params.id);  
  let team = teams.find(team => team.id === id);  
  try{  
    team ?  
      res.status(200).json(team)  
      :  
      res.status(404).send("Not found")  
  }  
  catch(err){  
    console.log("\x1b[31m%s\x1b[0m", err.message);  
    res.status(500).send({message: "Server error !", error: err.message});  
  }  
});
```

Il est possible d'utiliser d'autres clés comme paramètres dans la requête, pas seulement l'ID. Cela permet de récupérer une ressource spécifique en utilisant différentes clés de recherche.



➤ POST (/teams)

La méthode HTTP POST est utilisée pour créer une nouvelle ressource sur le serveur. Dans ce code, l'endpoint est "/api/teams" et les données de la nouvelle équipe sont envoyées dans le corps de la requête HTTP. Après avoir validé les données envoyées, une nouvelle équipe est créée dans la base de données et renvoyée avec un code d'état 201 (Créé).

```
app.post('/api/teams', (req, res) => {  
  let team = req.body;  
  // You can add here validity code  
  if (!team) res.status(400).send("Invalide request !");  
  try{  
    teams.push(team);  
    fs.writeFileSync("./teams.json", JSON.stringify(teams));  
    res.status(201).json(team);  
  }  
  catch(err){  
    console.log("\x1b[31m%s\x1b[0m", err.message);  
    res.status(500).send({message: "Server error !", error: err.message});  
  }  
});
```

Si les données envoyées ne sont pas valides, un code d'état 400 (Mauvaise demande) est renvoyé avec un message d'erreur indiquant les problèmes rencontrés avec les données.



➤ PUT (/teams/:id)

La méthode HTTP PUT est utilisée pour mettre à jour une ressource existante sur le serveur en utilisant un endpoint spécifique. Dans ce code, l'endpoint est "/api/teams/:id". Les données mises à jour sont envoyées dans le corps de la requête au format JSON.

Le serveur renvoie ensuite une réponse indiquant si la mise à jour a réussi ou échoué, avec un code d'état approprié (par exemple, 200 pour réussite, 400 pour données invalides, 404 pour ressource introuvable, etc.).

```
app.put('/api/teams/:id', (req, res) => {
  const id = parseInt(req.params.id);
  let team = teams.find(team => team.id === id);
  let newdata = req.body;
  // You can add here validity code
  try{
    team.name = newdata.name;
    team.country = newdata.country;
    fs.writeFileSync("./teams.json", JSON.stringify(teams));
    res.status(200).json(teams);
  }
  catch(err){
    console.log("\x1b[31m%s\x1b[0m", err.message);
    res.status(500).send({message: "Server error !", error: err.message});
  }
});
```



➤ DELETE (/teams/:id)

La méthode HTTP DELETE est utilisée pour supprimer une ressource spécifique sur le serveur en utilisant un endpoint spécifique. Dans ce code, l'endpoint est "/api/teams/:id".

```
app.delete('/api/teams/:id', (req, res) => {  
  const id = parseInt(req.params.id);  
  let team = teams.find(team => team.id === id);  
  if (team) {  
    try{  
      teams.splice(teams.indexOf(team), 1);  
      fs.writeFileSync("./teams.json", JSON.stringify(teams));  
      res.status(200).json(teams);  
    }  
    catch(err){  
      console.log("\x1b[31m%s\x1b[0m", err.message);  
      res.status(500).send({message: "Server error !", error: err.message});  
    }  
  }else res.status(404).send("Not found !");  
});
```

Si la ressource est supprimée avec succès, le serveur renvoie un code de statut 200. Si la ressource n'existe pas, le serveur renvoie un code de statut 404.

03

Tests avec Postman





Postman est un outil populaire utilisé par les développeurs pour tester et déboguer des API. Il fournit une interface facile à utiliser pour effectuer des requêtes HTTP, inspecter les réponses et tester différents paramètres et charges de données.

Avec Postman, les développeurs peuvent facilement créer et gérer des collections de demandes d'API, et collaborer avec les membres de l'équipe pour développer, tester et documenter des API.



Installation de Postman

<https://www.postman.com/downloads/>

The Postman app

Download the app to get started with the Postman API Platform.

 Windows 64-bit

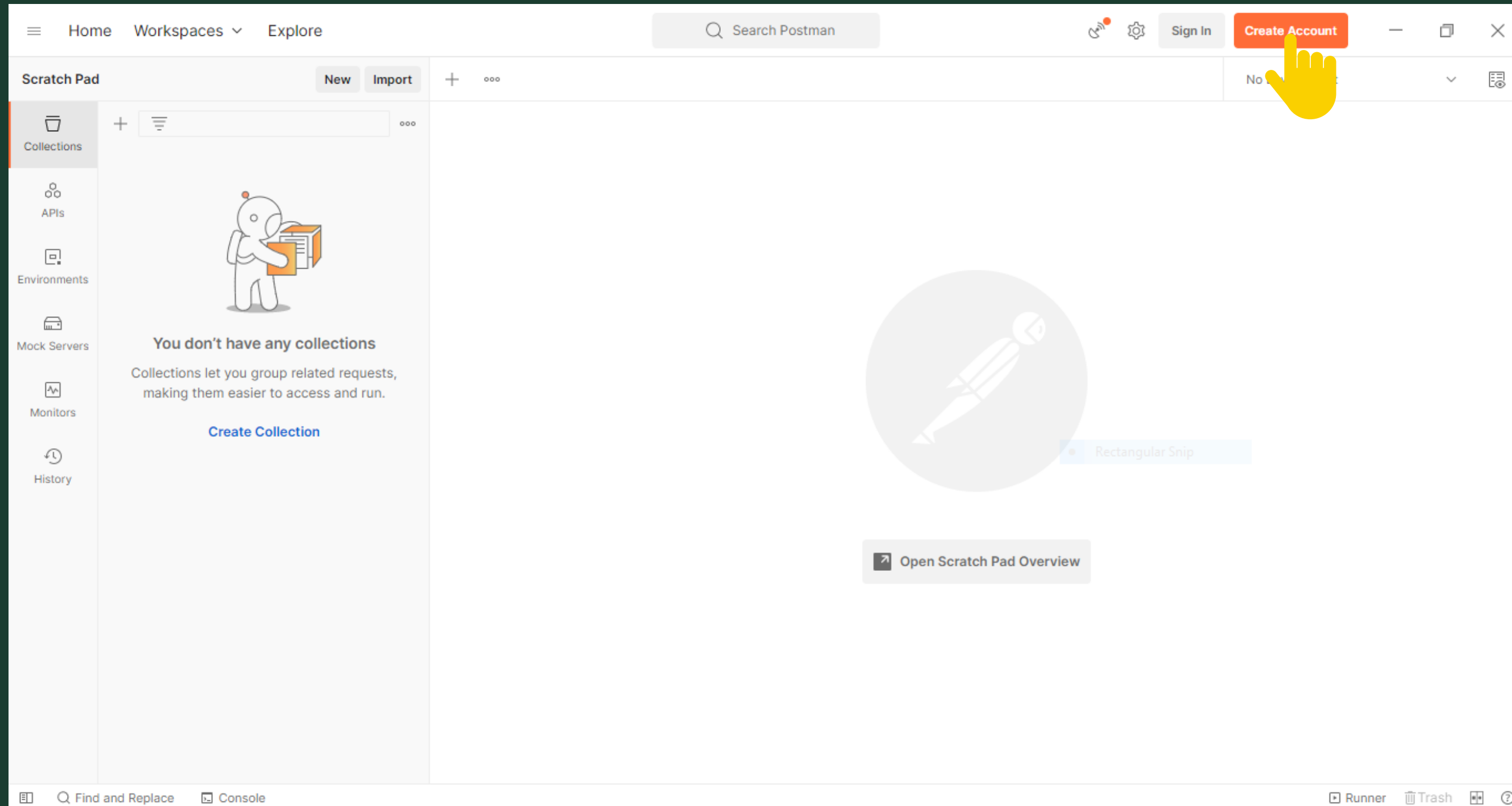
By downloading and using Postman, I agree to the [Privacy Policy](#) and [Terms](#).

[Release Notes](#) · [Product Roadmap](#)

Not your OS? Download for Mac ([Intel Chip](#), [Apple Chip](#)) or Linux ([x64](#), [arm64](#))




Installation de Postman






➔ Installation de Postman

 **POSTMAN**

Why sign up?

- Organize all your API development within Postman Workspaces
- Sync your Postman data across devices
- Backup your data to the Postman cloud
- It's free!



Create Postman Account

[Sign In instead?](#)

Email

Username

Password [SHOW](#)


☐ Sign up to get product updates, news, and other marketing communications.

☒ Stay signed in for 30 days

By creating an account, I agree to the [Terms](#) and [Privacy Policy](#).

[Create free account](#)

or

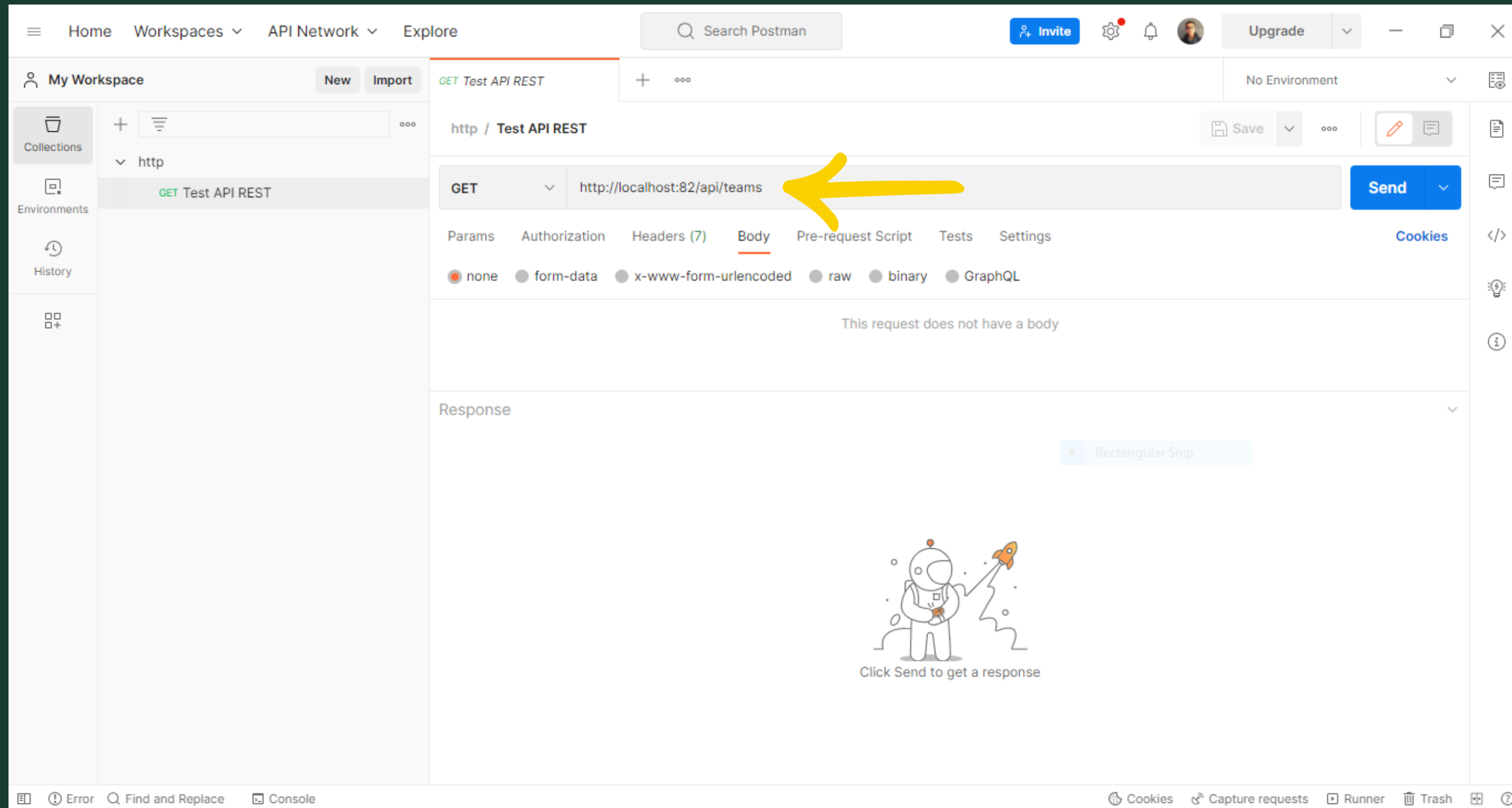
 [Sign up with Google](#)

[Sign in with SSO](#)

You've been redirected from the Postman Desktop App on win32 10.0.19045 to create an account.

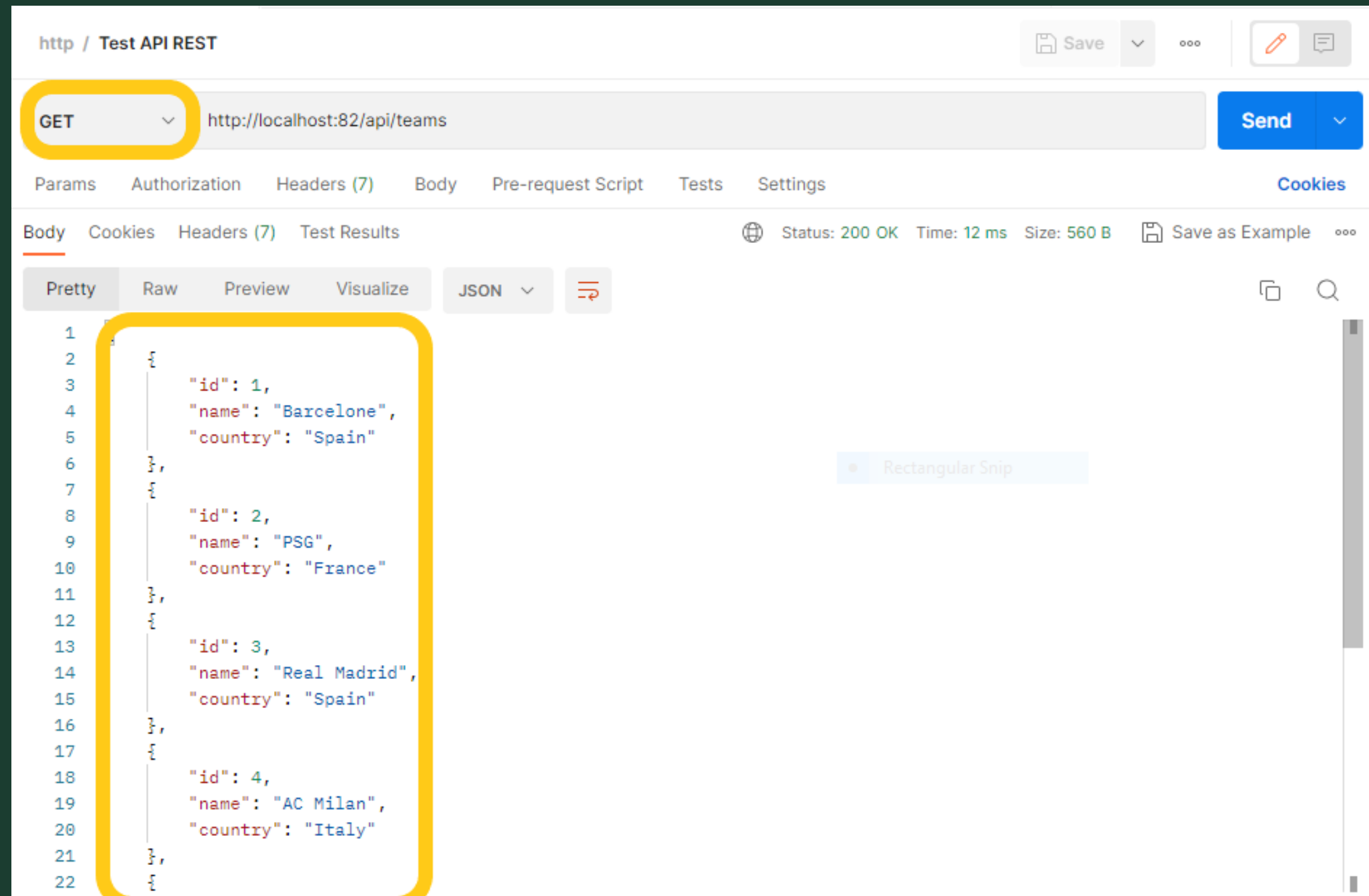


Ajoute de URL





➔ Route GET (**api/teams**)





➡ Route GET (**api/teams/:id**)

http / Test API REST

GET http://localhost:82/api/teams/5

Send

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Body Cookies Headers (7) Test Results

Status: 200 OK Time: 5 ms Size: 282 B Save as Example

Pretty Raw Preview Visualize JSON

```
1 {  
2   "id": 5,  
3   "name": "Liverpool",  
4   "country": "England"  
5 }
```



➔ Route POST (**api/teams**)

http / Test API REST

POST http://localhost:82/api/teams

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded **raw** binary GraphQL **JSON**

```
1 {
2   ... "id": 7,
3   ... "name": "Raja CA",
4   ... "country": "Morocco"
5 }
```

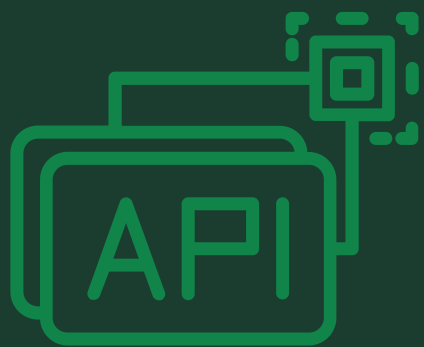
Body Cookies Headers (7) Test Results

Status: 201 Created Time: 199 ms Size: 285 B Save as Example

Rectangular Ship

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 7,
3   "name": "Raja CA",
4   "country": "Morocco"
5 }
```

➡ Route PUT (**api/teams/:id**)

http / Test API REST

PUT http://localhost:82/api/teams/1

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded **raw** binary GraphQL **JSON**

```
1 {
2   "id": 1,
3   "name": "Barcelone FC",
4   "country": "Spain"
5 }
```

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize **JSON**

```
1 [
2   {
3     "id": 1,
4     "name": "Barcelone FC",
5     "country": "Spain"
6   },
7 ]
```

Status: 200 OK Time: 11 ms Size: 562 B Save as Example



Route DELETE (**api/teams/:id**)

DELETE

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Body Cookies Headers (7) Test Results

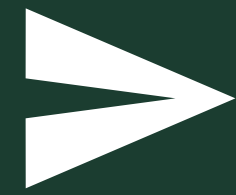
Pretty Raw Preview Visualize JSON

```
18      "id": 4,  
19      "name": "AC Milan",  
20      "country": "Italy"  
21    },  
22    {  
23      "id": 5,  
24      "name": "Liverpool",  
25      "country": "England"  
26    },  
27    {  
28      "id": 6,  
29      "name": "Marseille",  
30      "country": "France"  
31    }  
32  ]
```

04

API REST et MongoDB





Rappel Mongodb



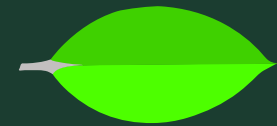
- ✓ Comme nous l'avons vu dans le module gestion de données MongoDB est une base de données NoSQL orientée document Elle se distingue des bases de données relationnelles par sa flexibilité et ses performances.
- ✓ Contrairement à une base de données **relationnelle SQL** traditionnelle, MongoDB ne repose pas sur des tableaux et des colonnes Les données sont stockées sous forme de collections et de documents.
- ✓ **Les documents sont des paires de valeurs/clés** servant d'unité de données de base Les collections quant à elles contiennent des ensembles de documents et de fonctions Elles sont l'équivalent des tableaux dans les bases de données relationnelles classiques.



OBJECTIF :

**On va persister les données directement de et vers
une base de données MongoDB**





Créer une base de donnée et une Collection

New Connection

Connect to a MongoDB deployment

URI ⓘ

Edit Connection String ☒

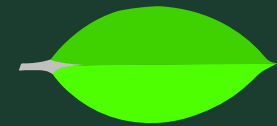
`mongodb://localhost:27017`

[➤ Advanced Connection Options](#)

Save

Save & Connect

Connect

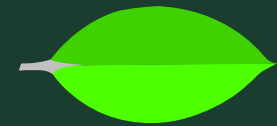


Créer une base de donnée et une Collection

My Queries **Databases** Performance

[+ Create database](#) [Refresh](#) View [≡](#) [⌵](#) Sort by Database Name [⌵](#)

admin			
Storage size:	Collections:	Indexes:	
40.96 kB	2	2	
Company			
Storage size:	Collections:	Indexes:	
36.86 kB	1	1	
config			
Storage size:	Collections:	Indexes:	
24.58 kB	1	2	
local			
Storage size:	Collections:	Indexes:	
69.63 kB	1	1	



Créer une base de donnée et une Collection

×

Create Database

Database Name

myDBAPI

Collection Name

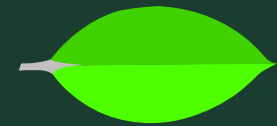
teams

☐ Time-Series
Time-series collections efficiently store sequences of measurements over a period of time. [Learn More](#)

> Additional preferences (e.g. Custom collation, Capped, Clustered collections)

Cancel

Create Database



Importer les données

myDBAPI.teams

0 DOCUMENTS1 INDEXES

Documents

Aggregations

Schema

Explain Plan

Indexes

Validation

Filter

Type a query: { field: 'value' }

Reset

Find

</>

More Options

ADD DATA

EXPORT COLLECTION

0 - 0 of 0

Rectangular Snip

This collection has no data

It only takes a few seconds to import data from a JSON or CSV file.

Import Data



Importer les données

myDBAPI.teams

61
DOCUMENTSINDEXES

Documents

Aggregations

Schema

Explain Plan

Indexes

Validation

Filter

⌚

Type a query: { field: 'value' }

Reset

Find

⌘

More Options

ADD DATA

EXPORT COLLECTION

1 - 6 of 6

↺

↻

⋮

{ }

⌂

_id: ObjectId('6457ede1d2d174b3a98b1cdf')

id: 1

name: "Barcelone"

country: "Spain"

Rectangular Snip

_id: ObjectId('6457ede1d2d174b3a98b1ce0')

id: 2

name: "PSG"

country: "France"

_id: ObjectId('6457ede1d2d174b3a98b1ce1')

id: 3

name: "Real Madrid"

country: "Spain"

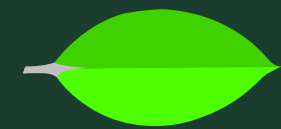
_id: ObjectId('6457ede1d2d174b3a98b1ce2')

id: 4

05

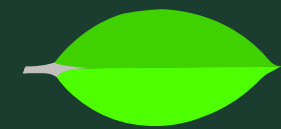
CRUD sur MongoDB





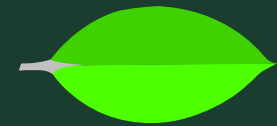
Configuration de l'application Express avec MongoDB

```
const express = require("express");  
const MongoClient = require("mongodb").MongoClient;  
  
const url = "mongodb://localhost:27017";  
const dbName = 'myDBAPI'  
  
const app = express();  
app.use(express.json());
```



Connexion à MongoDB et récupération de la base de données

```
async function connectDB() {  
  try {  
    const client = await MongoClient.connect(url);  
    console.log('\x1b[32m%s\x1b[0m', '\nConnected to mongoDB successfully');  
    return client.db(dbName);  
  } catch (err) {  
    console.log('\x1b[31m%s\x1b[0m', err);  
    throw err;  
  }  
}
```



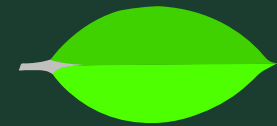
GET (api/teams)

```
app.get('/api/teams', async (req, res) => {  
  try {  
    const db = await connectDB();  
    let teams = await db.collection('teams').find({}).toArray();  
    res.status(200).json(teams);  
  } catch (err) {  
    console.log( '\x1b[31m%s\x1b[0m', err);  
    res.status(500).send("DataBase error !!")  
  }  
});
```



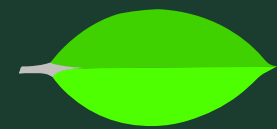
GET (`api/teams/:id`)

```
app.get('/api/teams/:id', async (req, res) => {
  try {
    const id = parseInt(req.params.id);
    const db = await connectDB();
    let team = await db.collection('teams').findOne({
      "id": id
    });
    team ? res.status(200).json(team)
        : res.status(404).send("Team not found !")
  } catch (err) {
    console.log('\x1b[31m%s\x1b[0m', err);
    res.status(500).send("DataBase error !!");
  }
});
```



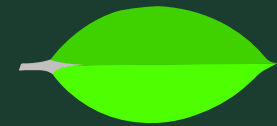
POST (**api/teams**)

```
app.post('/api/teams', async (req, res) => {  
  try {  
    let data = req.body; // add validity check ..  
    const db = await connectDB();  
    const result = await db.collection('teams').insertOne(data);  
    res.status(200).json(result);  
  } catch (err) {  
    console.log(err);  
    res.status(500).send("DataBase error !!");  
  }  
});
```



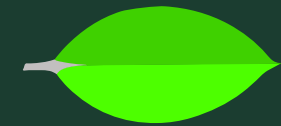
PUT (`api/teams/:id`)

```
app.put('/api/teams/:id', async (req, res) => {  
  try {  
    const id = parseInt(req.params.id);  
    let newData = req.body;  
    const db = await connectDB();  
    let result = await db.collection('teams')  
      .updateOne(  
        { "id": id },  
        { $set: newData }  
      );  
    res.status(200).json(result);  
  } catch (err) {  
    console.log(err);  
    res.status(500).send("DataBase error !!");  
  }  
});
```

DELETE (`api/teams/:id`)

```
app.delete('/api/teams/:id', async (req, res) => {  
  try {  
    const id = parseInt(req.params.id);  
    const db = await connectDB();  
    const result = await db.collection('teams').deleteOne({"id": id});  
    res.status(200).json(result);  
  }  
  catch(err){  
    console.log(err);  
    res.status(500).send("DataBase error !!");  
  }  
});
```



Connexion à la base de données et démarrage du serveur

```
connectDB().then(() => {  
  app.listen(82, () => {  
    console.log('Server listen at port:82')  
  });  
}).catch(err => console.log( '\x1b[31m%s\x1b[0m', err));
```



Remarque :

Pour tester les routes CRUD que nous avons créées en utilisant MongoClient, vous pouvez utiliser les mêmes paramètres que nous avons utilisés précédemment avec Postman.



06

Le module mongoose





Mongoose est une bibliothèque de modélisation de données objet (ODM) pour MongoDB et Node.js. Il fournit un moyen simple de définir des structures de schéma pour les collections MongoDB et simplifie les interactions avec MongoDB en fournissant une API de niveau supérieur qui prend en charge des fonctionnalités comme la validation, le middleware, et plus encore.

Essentiellement, c'est une couche sur le dessus du pilote MongoDB qui fournit des fonctionnalités supplémentaires pour rendre le travail avec MongoDB dans Node.js plus facile et plus efficace.



Model - Team



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

const TeamSchema = new mongoose.Schema({
  id :{
    type: Number,
    required: true
  },
  name: {
    type: String,
    required: true
  },
  country: {
    type: String,
    required: true
  }
},
{timestamps:true}
);
module.exports = mongoose.model("Team", TeamSchema);
```



Connexion à MongoDB et récupération de la base de données

```
const express      = require("express");
const TeamModel    = require("../models/Team");
const mongoose     = require("mongoose");
const app          = express();
const url          = "mongodb://localhost:27017/myDBAPI";

async function connectDB() {
  await mongoose.connect(url, { useNewUrlParser: true, useUnifiedTopology: true })
    .then(() => console.log('\x1b[32m%s\x1b[0m', "\nConnected to mongoDB successfully"))
    .catch(err => console.log('\x1b[31m%s\x1b[0m', err.message));
}
```




GET (**api/teams**)

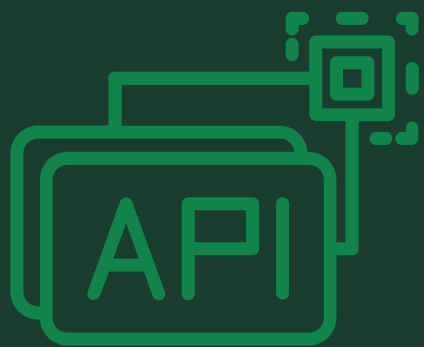
```
app.get('/api/teams', async (req, res) => {  
  try{  
    let teams = await TeamModel.find({});  
    teams ?  
      res.status(200).json(teams)  
      :  
      res.status(404).send("No data available")  
  }  
  catch(err){  
    console.log(err.message);  
    res.status(500).send({message: "Server error !", error: err.message});  
  }  
});
```

• Rectangular



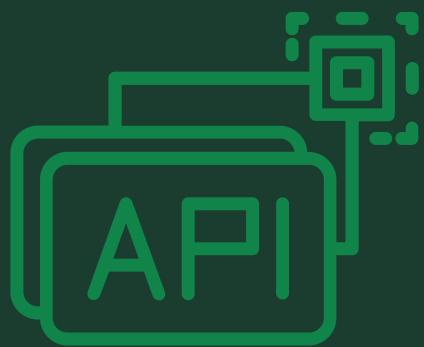
GET (**api/teams/:id**)

```
app.get('/api/teams/:id', async (req, res) => {  
  try{  
    const id = parseInt(req.params.id);  
    const team = await TeamModel.find({id});  
    team.length !== 0 ?  
      res.status(200).json(team)  
      :  
      res.status(404).send("Not found !")  
  }catch(err){  
    console.log(err.message);  
    res.status(500).send({message: "Server error !", error: err.message});  
  }  
});
```



POST (**api/teams**)

```
app.post('/api/teams', (req, res) => {  
  try{  
    const {id, name, country} = req.body;  
    let postedTeam = new TeamModel({id, name, country});  
    postedTeam ?  
      postedTeam.save().then(  
        res.status(201).json(postedTeam)  
      )  
      :  
      res.status(400).send("Invalid request !")  
  }catch(err){  
    console.log(err.message);  
    res.status(500).send({message: "Server error !", error: err.message});  
  }  
});
```



PUT (**api/teams/:id**)

```
app.put('/api/teams/:id', async (req, res) => {  
  try {  
    const id = parseInt(req.params.id);  
    const updatedTeam = await TeamModel.findOneAndUpdate(  
      { id: id },  
      { $set: req.body },  
      { new: true }  
    );  
    updatedTeam ?  
      res.status(200).json(updatedTeam)  
      :  
      res.status(404).send('Team not found!')  
  } catch(err) {  
    console.log(err.message);  
    res.status(500).send({ message: 'Server error!', error: err.message });  
  }  
});
```



DELETE (**api/teams/:id**)

```
app.delete('/api/teams/:id', async (req, res) => {  
  try {  
    const id = req.params.id;  
    const deletedTeam = await TeamModel.findOneAndDelete(  
      { id: id }  
    );  
    deletedTeam ?  
      res.status(200).json(deletedTeam)  
      :  
      res.status(404).send('Team not found!')  
  } catch(err) {  
    console.log(err.message);  
    res.status(500).send({ message: 'Server error!', error: err.message });  
  }  
});
```



CONCLUSION

Nous avons exploré diverses façons d'effectuer des opérations CRUD sur des données, en commençant par un simple fichier JSON et en passant à l'utilisation de MongoDB et de la bibliothèque Mongoose. Nous avons vu comment Postman peut être utilisé pour tester les API et s'assurer qu'elles fonctionnent correctement.

En tirant parti de ces technologies, nous pouvons créer des applications puissantes et évolutives qui peuvent gérer une grande quantité de données et offrir une expérience utilisateur fluide.

Merci de vous joindre à moi pour ce voyage et j'espère que vous l'avez trouvé instructif et utile.