



P2#ADEFNIPA

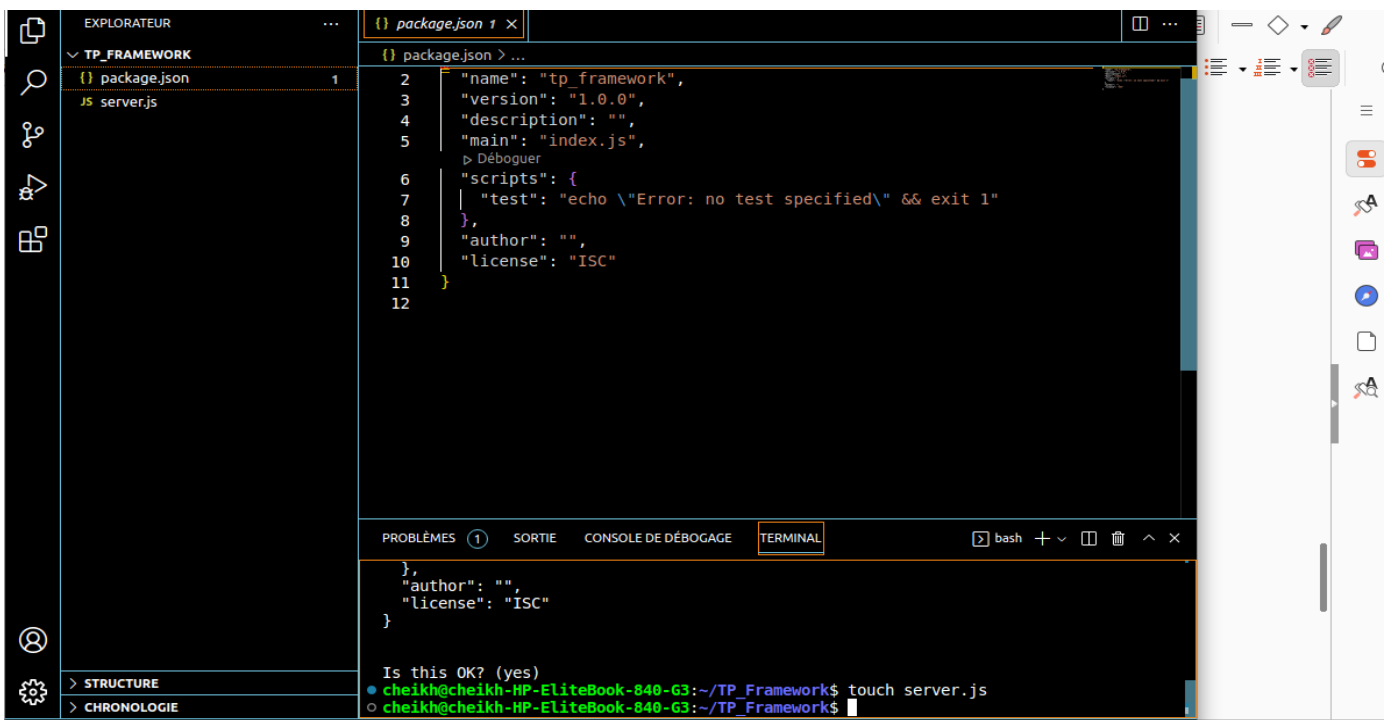
Documentation API CRUD

Par

Mohamed FALL
Mbayang GUEYE
Khadija DIALLO
Cheikh DIOUF

SOMMAIRE

1- Généralités sur l'API	4
2- La création de notre API Rest	4
Technologies utilisées	
2-1-Node js	4
2-2-Express js	4
3- Définition des ressources et les routes de l'API	7
3-1-Les Ressources de l'API.....	7
3-2-Création des routes de l'API	7
4- Paramétrages et connexion à la base de données MongoDB	8
La base de données MongoDB	
5- Fonctions de L'API	9
5-1- Le CRUD et les méthodes HTTP	9
5-2-Les Codes d'états des requêtes HTTP	9
6- Tests de fonctionnalité de l'API	10
6-1-Outils de test Postman	10
6-2-Test 1: requête d'inscription	11
6-3-Test 2: requête de modification	11
6-4-Test 3: requête de récupération d'un seul inscrit	12
6-5-Test 3: requête de récupération de tout les inscrits	12
6-6-Test 3: Réponse à une requête de réinscription avec un email déjà existant	13
6-7-Test 2: requête de suppression	14
Conclusion	15

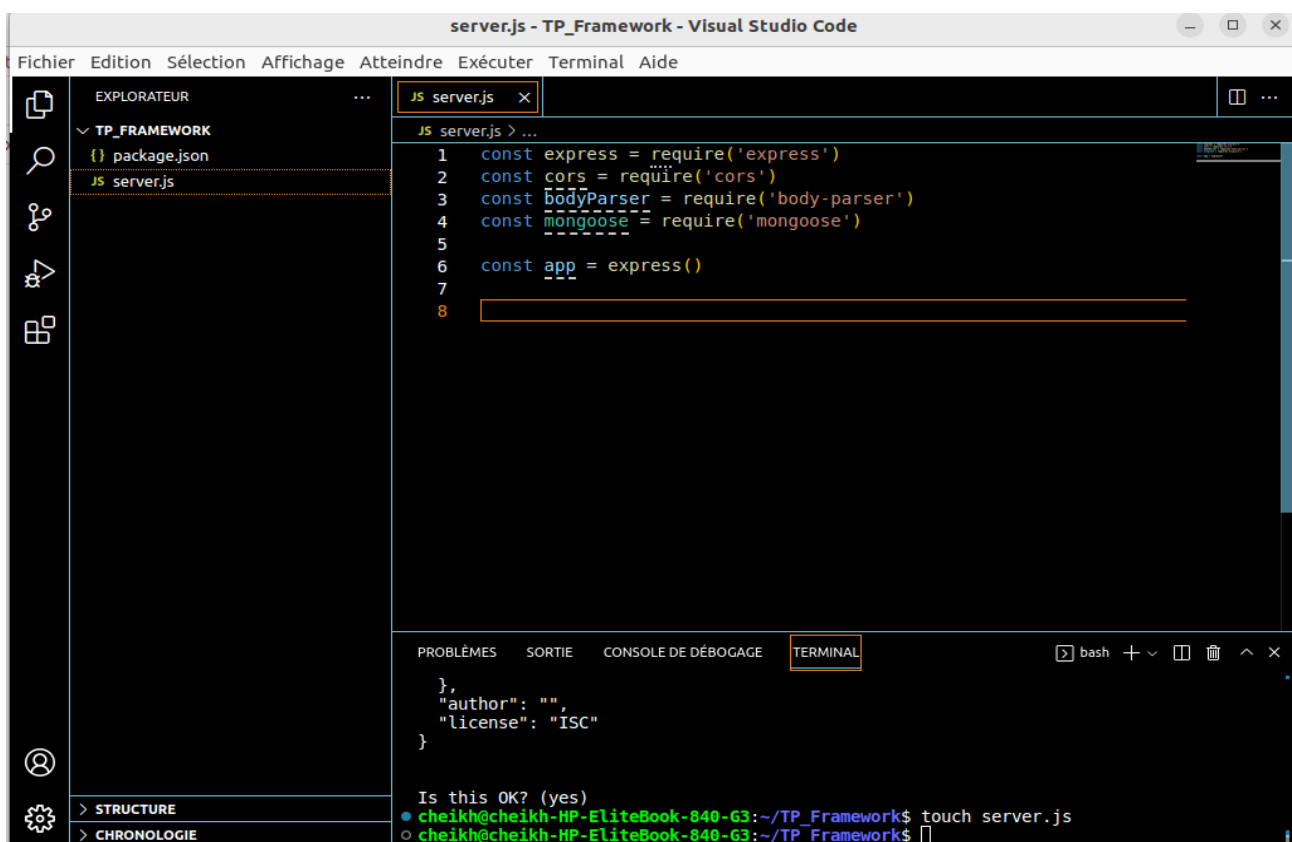


Ajoutons maintenant Express pour la gestion du serveur avec **npm install express**. Cette commande a pour but de télécharger depuis la registry NPM puis d'installer la librairie express ainsi que l'ensemble des librairies dont express a besoin pour fonctionner dans votre répertoire de travail.

Maintenant que tout est installé, on peut créer le serveur dans le fichier nommé **server.js**. Commençons par intégrer la librairie express dans notre fichier server.js comme suit:

```
const express = require('express')const app = express()
```

require('express') est une façon d'importer la librairie express et ses fonctions dans notre code. La constante **app** est l'instanciation d'un objet Express, qui va contenir notre serveur ainsi que les méthodes dont nous aurons besoin pour le faire fonctionner.

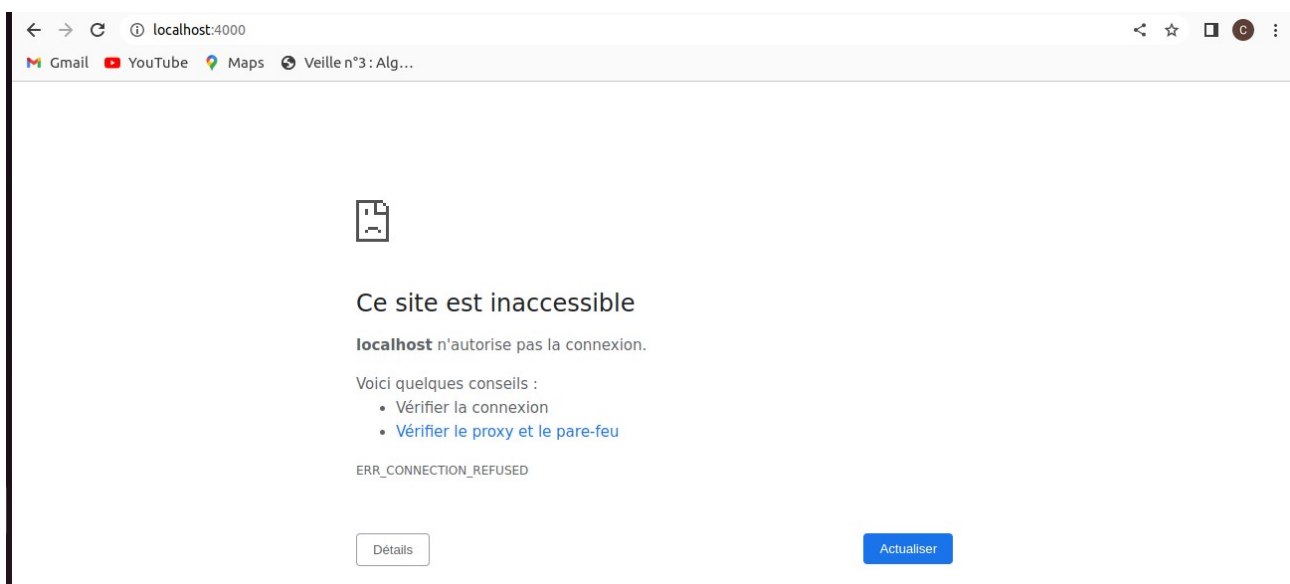


The screenshot shows the Visual Studio Code interface with the file explorer on the left showing the project structure: TP_FRAMEWORK, package.json, and server.js. The main editor displays the content of server.js, which includes the following code:

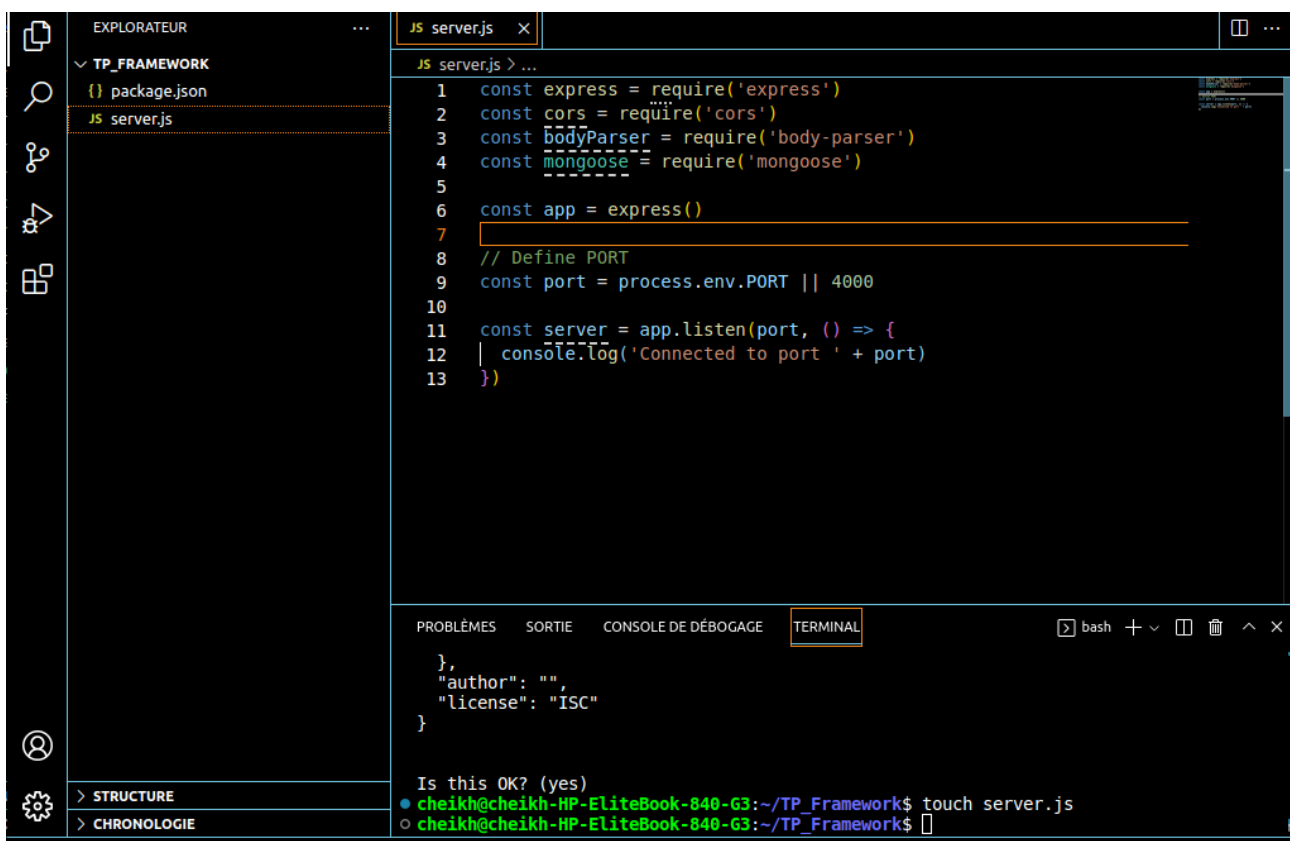
```
1 const express = require('express')
2 const cors = require('cors')
3 const bodyParser = require('body-parser')
4 const mongoose = require('mongoose')
5
6 const app = express()
7
8
```

The terminal at the bottom shows the output of the command `touch server.js` and the prompt `Is this OK? (yes)`.

Pour le moment, le serveur est préparé mais pas encore lancé. Si vous vous rendez sur `localhost:4000` depuis votre navigateur, vous devriez avoir une erreur.



Pour que notre serveur puisse être à l'écoute il faut maintenant utiliser la méthode **listen** fournie dans app et lui spécifier un port. Nous définissons le port 4000 pour notre server.



En lançant la commande **node server.js** (ou **nodemon**) dans votre terminal, vous verrez qu'il affichera le message «**connected to port 4000**». Cela veut dire que tout fonctionne bien. S'il y a une erreur, vous aurez droit à un message d'erreur sur votre terminal. Votre serveur Node est à l'écoute mais ne sait pas quoi faire pour l'instant.

Si vous vous rendez sur votre navigateur à l'adresse localhost:4000 votre serveur répond à votre navigateur. N'ayant pour l'instant aucune route configurée, il vous retourne cette erreur **Cannot GET**/mais il est bel et bien fonctionnel.

3- Définition des ressources et les routes de l'API

Du point de vue fonctionnalité, les ressources sont des entités sur lesquelles peuvent s'appliquer les actions associées aux verbes http. Les routes quant à elles permettent de spécifier la cible (ressource) sur laquelle l'action s'exécute.

3-1-Les Ressources de l'API

Pour notre exemple, nous aurons besoin des fonctionnalités suivantes:

- Ajouter un utilisateur/admin
- Modifier données d'un inscrit
- Archiver un inscrit
- Changer le rôle d'un inscrit
- Modifier le mot de passe d'une personne particulière (connectée)
- Récupère les données des inscrits

Ces opérations sont plus communément appelées CRUD, pour CREATE, READ, UPDATE, DESTROY. Dans notre exemple, notre API dispose principalement une ressource: **c'est la personne inscrite (admin ou simple user)**.

3-2-Les Routes de l'API

Le standard d'API REST impose que nos routes soient centrées autour de nos ressources et que les méthodes HTTP utilisées reflètent l'intention de l'action. Dans notre cas nous aurons besoin des routes suivantes:

Rappelons justement que le point d'entrées de notre serveur (endpoint) est:

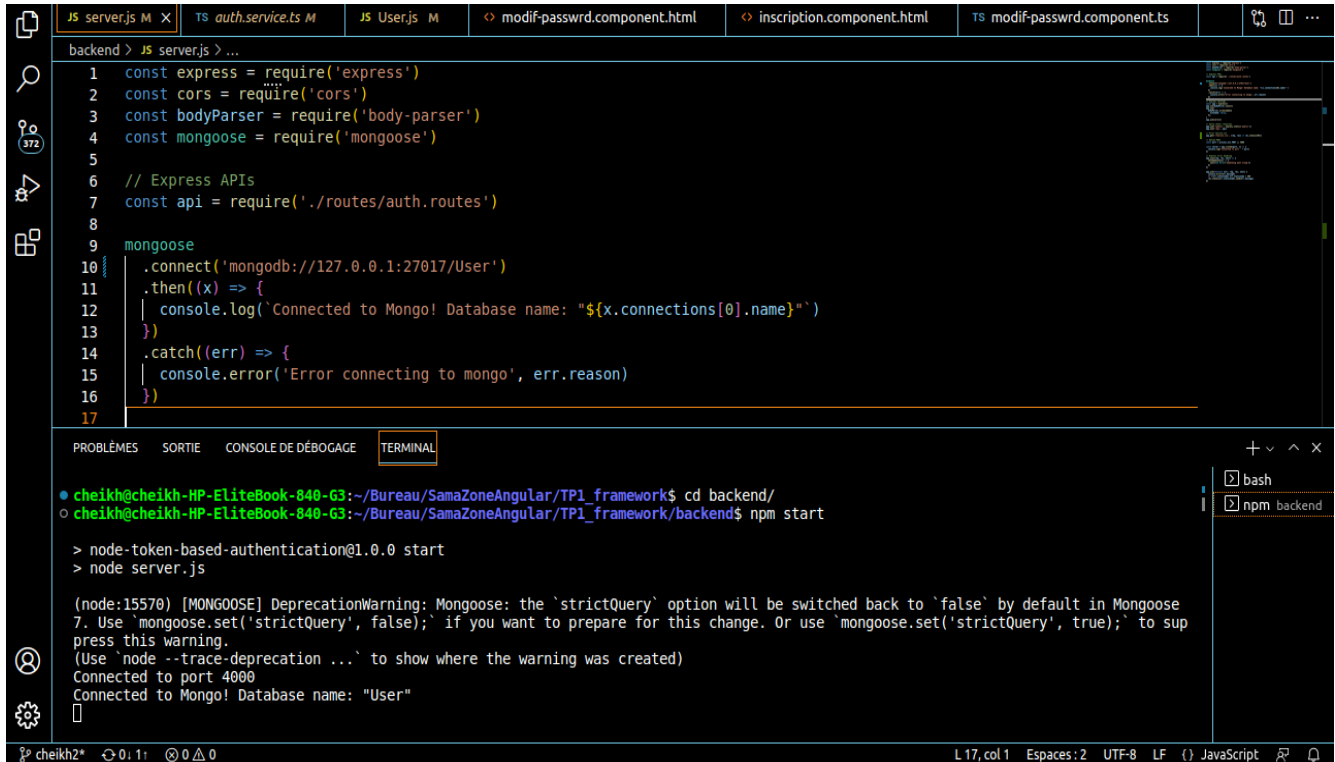
http://localhost:4000/api

- **GET/endpoint:** Pour récupérer tous les utilisateurs inscrits.
- **GET/endpoint/read-user/id:** Pour récupérer un inscrit en particulier.
- **GET/endpoint/user-profil/id:** Pour récupérer le profil d'un utilisateur en particulier.
- **PUT/endpoint/update-user/id:** Pour modifier un inscrit en particulier.
- **DELETE/endpoint/delete-user/id:** Pour archiver un inscrit en particulier.
- **POST/endpoint/register-user:** pour enregistrer un nouvel utilisateur.
- **POST/endpoint/signin:** pour connecter un nouvel utilisateur.

En fonction donc de l'URI fournie, l'API pourra connaître avec précision la demande de l'utilisateur et pourra par la suite faire parvenir la requête du côté de la base de données.

4- Paramétrages et connexion à la base de données MongoDB

Pour pouvoir jouer à plain son rôle, l'API a besoin de se lier avec la base de données; condition sans laquelle, elle aurait failli à sa mission. Pour ce faire, on écrit un code dans le fichier server.js.



The screenshot shows a code editor with several tabs: 'JS server.js M X', 'TS auth.service.ts M', 'JS User.js M', 'modif-passwrd.component.html', 'inscription.component.html', and 'TS modif-passwrd.component.ts'. The 'server.js' tab is active, displaying the following code:

```
1 const express = require('express')
2 const cors = require('cors')
3 const bodyParser = require('body-parser')
4 const mongoose = require('mongoose')
5
6 // Express APIs
7 const api = require('./routes/auth.routes')
8
9 mongoose
10 .connect('mongodb://127.0.0.1:27017/User')
11 .then((x) => {
12   console.log('Connected to Mongo! Database name: "${x.connections[0].name}"')
13 })
14 .catch((err) => {
15   console.error('Error connecting to mongo', err.reason)
16 })
17
```

Below the code editor is a terminal window with the following output:

```
● cheikh@cheikh-HP-EliteBook-840-G3:~/Bureau/SamaZoneAngular/TP1_framework$ cd backend/
○ cheikh@cheikh-HP-EliteBook-840-G3:~/Bureau/SamaZoneAngular/TP1_framework/backend$ npm start

> node-token-based-authentication@1.0.0 start
> node server.js

(node:15570) [MONGODB] DeprecationWarning: Mongoose: the 'strictQuery' option will be switched back to 'false' by default in Mongoose 7. Use 'mongoose.set('strictQuery', false);' if you want to prepare for this change. Or use 'mongoose.set('strictQuery', true);' to suppress this warning.
(Use 'node --trace-deprecation ...' to show where the warning was created)
Connected to port 4000
Connected to Mongo! Database name: "User"

```

The terminal window also shows the commands 'bash' and 'npm backend' in the right-hand pane.

5- Fonctions de L'API

Principalement, l'API fait office de link entre l'utilisateur (application angular) et la base de données mongoDB. Mais parler des fonctions revient à présenter les verbes HTTP utilisés et qui sont vecteurs des actions effectuées dans la plateforme (le CRUD).

5-1- Le CRUD et les méthodes HTTP

Dans notre plateforme, chaque action effectuée se fait au moyens d'un verbe http spécifique. Le client passe une requete, l'API capte le message et le comprend avec les informations comme la ressource sollicitée, le chemin ...

L'API va à son tour faire parvenir la requête au serveur pour enfin retourner un resultat. Voyons comment cela fonctionne en détails.

- Le POST et l'envoi de données (CREATE)

Quant on veut enregistrer un nouvel utilisateur dans notre base de données, nous utilisons la methode POST comme suit:

POST: <http://localhost:4000/api/register-user>

- Le DELETE pour l'archivage

On utilise la requete suivante pour pouvoir archiver un inscrit dans la base de données.

- Le GET pour la récupération de tous les inscrit de la base de données

GET: <http://localhost:4000/api>

On ecrit simplement la requete suivante pour récupérer l'ensemble des inscrits dans la base de données. Ce qui nous a permis de les afficher dans notre tableau.

- Le PUT pour la modification (UPDATE).

Il arrive qu'on ait besoin de changer des champs d'un utilisateur. Pour cela, il faut employer PUT comme suit en spécifiant son id:

PUT: <http://localhost:4000/api/update-user/id>

- Le DELETE pour l'archivage d'un inscrit

Dans certains cas, un inscrit peut ne plus faire partie du système. Dans ce cas, on sera obliger de l'archiver (ne plus l'afficher mais continuer à le stocker dans la base de données avec une propriété caractéristique de son état d'archivage.

DELETE: <http://localhost:4000/api/delete-user/id>

5-2-Les Codes d'états des requêtes HTTP

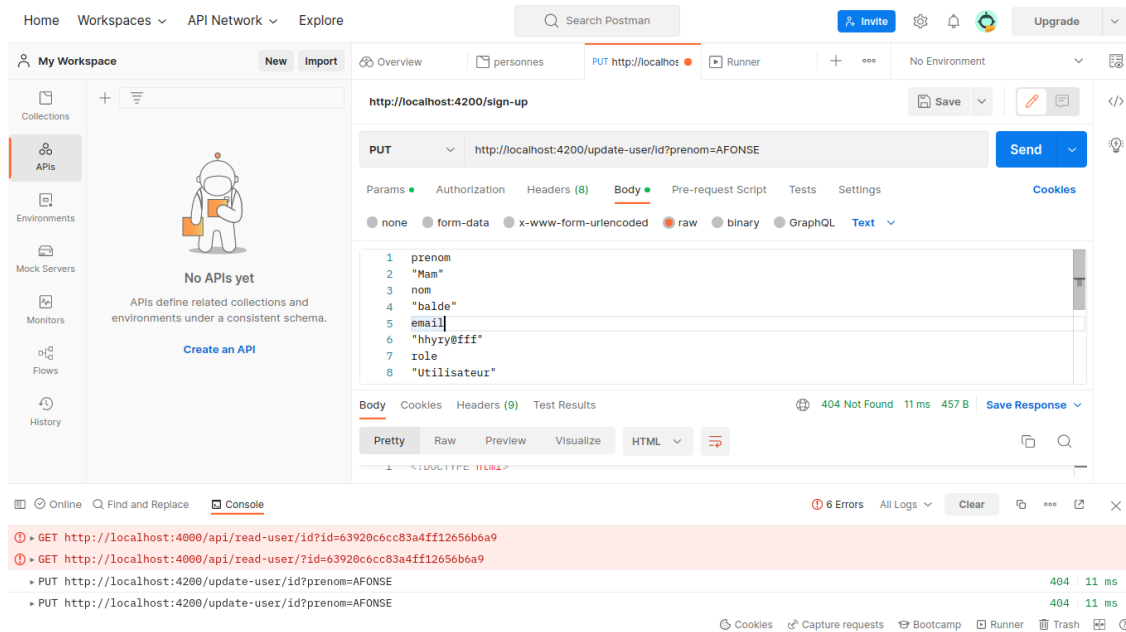
Quant on sollicite une chose, il n'est pas toujours garanti que tout se passera à la perfection. C'est valable aussi bien dans la vie courante que dans le domaine du web. Il peut y arriver que la requete envoyer dans le cas d'un CRUD (dans notre exemple) ne donne pas le resultat attendu. Il faut alors pouvoir retourner des messages pour informer au client de l'état de sa requete. Ces message son t codés sous format numérique (200, 300, 400, 500).

En voici quelques un de ces messages rencontrés dans notre plateforme: 200 signifie que tout est ok, 401 montre qu'aucun resultat n'est trouvé (not found) et les erreurs de code 500 montrent un probleme côté serveur.

6- Tests de fonctionnalité de l'API

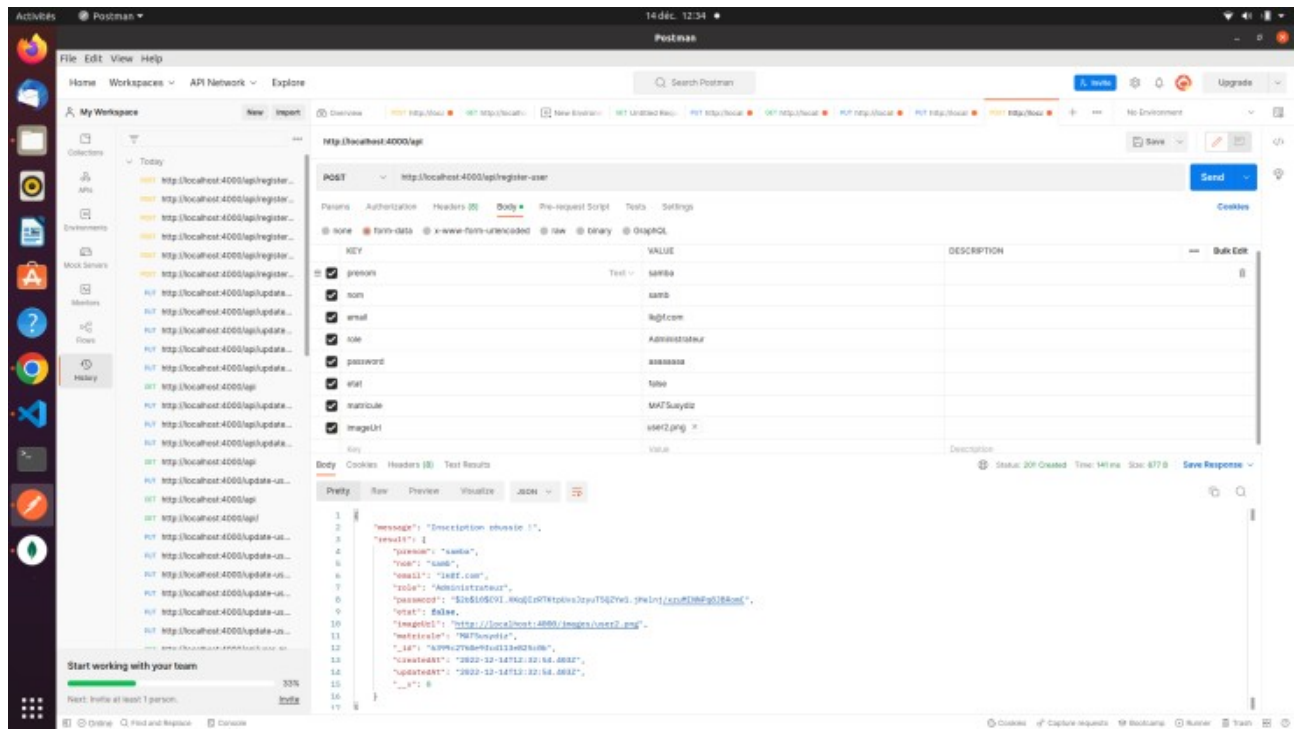
Après la conception de l'api, pour s'assurer de son bon fonctionnement, on a besoin de faire des tests avant de penser à son déploiement. Pour cela, nous utilisons principalement Postman comme outils de test.

6-1-Outils de test Postman



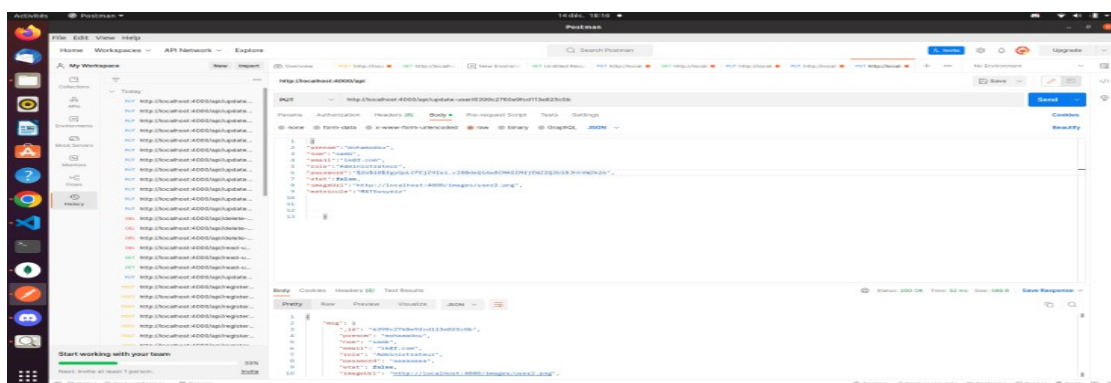
Postman nous permet avec son interface de pouvoir mettre notre URI (la requête) et de choisir le verbe http correspondant à notre action. Le bouton en bleu (send) permet de valider la requête.

6-2-Test 1: requête d'inscription



comme vous pouvez le voir, post est utilisé pour envoyer les données de l'inscription vers la base de données.

6-3-Test 2: requête de modification



Activities

Postman

14 dec. 10:05

File

Edit

View

Help

Home

Workspaces

API Network

Explore

My Workspace

New

Import

Overview

http://localhost:3000

New Collection

New Environment

Unlinked Request

http://localhost:4000

No Environment

Save

Upgrade

Collectors

New Collection

APIs

yarn

Environments

Mock Servers

Monitors

Roots

History

Search Postman

http://localhost:4000/api

GET

http://localhost:4000/api

Params

Authorization

Headers (0)

Body

Pre-request Script

Tests

Settings

None

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

Send

Cookies

Beauty

Body

Cookies

Headers (0)

Test Results

Pretty

Raw

Preview

Visualize

JSON

Status: 200 OK

Time: 57ms

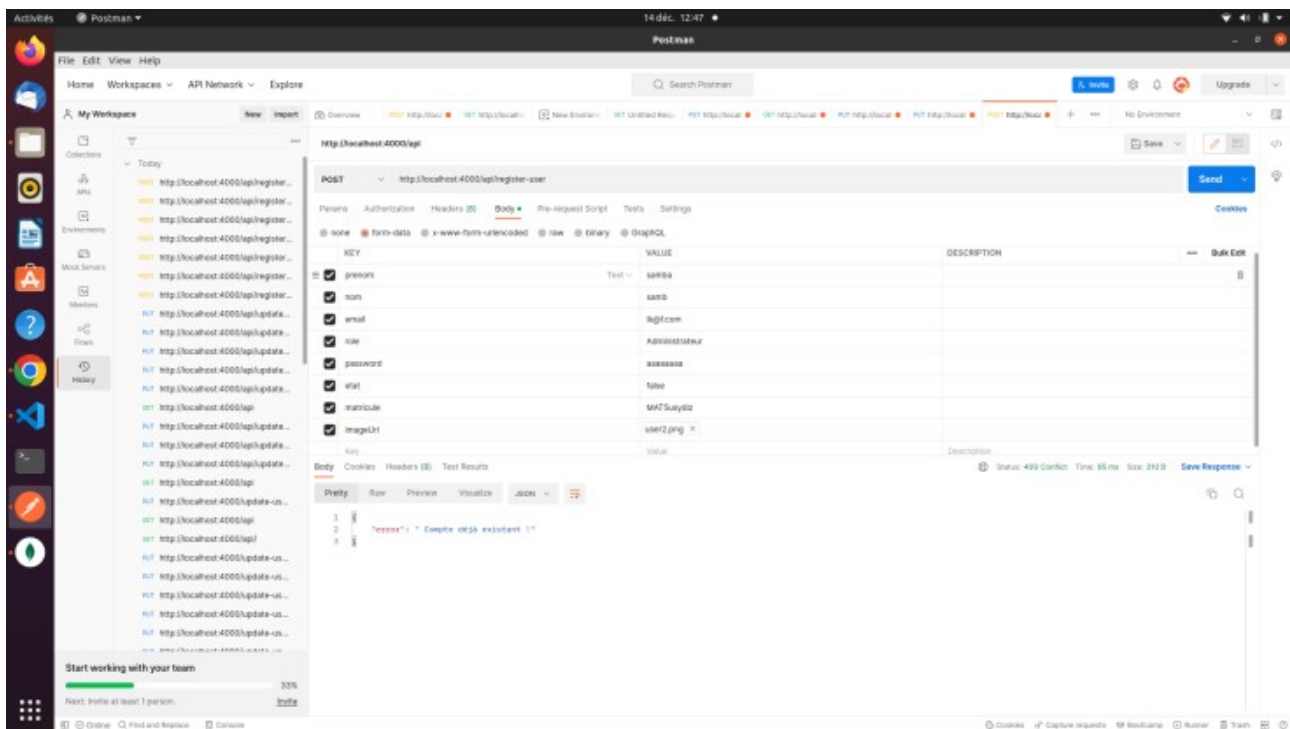
Size: 6.69 KB

Save Response

```

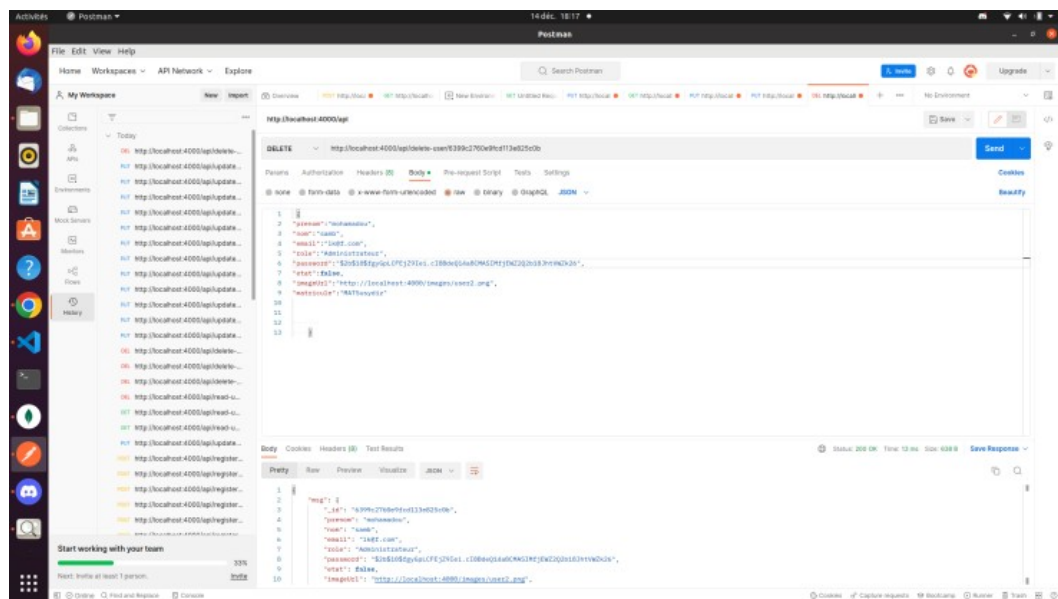
140      "role": "Administrate",
141      "password": "52b165905132e4530b1129c53a6b671a41a388c9f9c5f9b342",
142      "email": "false",
143      "loginUrl": "http://localhost:4000/public/test_page",
144      "createdAt": "2022-12-15T16:21:50.163Z",
145      "updatedAt": "2022-12-12T22:27:44.887Z",
146      "_id": 0
147    },
148    ],
149    {
150      "_id": "63963a1472e1f774a6b954a4",
151      "password": "password",
152      "role": "test",
153      "email": "xj@n.com",
154      "login": "testlogin",
155      "password": "52b165905132e4530b1129c53a6b671a41a388c9f9c5f9b342",
156      "loginUrl": "http://localhost:4000/public/test_page",
157      "createdAt": "2022-12-12T20:31:38.648Z",
158      "updatedAt": "2022-12-12T20:31:38.648Z",
159      "_id": 0
160    },
161    ],
162    {
163      "_id": "63963a1472e1f774a6b954a4",
164      "password": "password",
165      "role": "test",
166      "email": "xj@n.com",
167      "login": "testlogin",
168      "password": "52b165905132e4530b1129c53a6b671a41a388c9f9c5f9b342",
169      "loginUrl": "http://localhost:4000/public/test_page",
170      "createdAt": "2022-12-12T20:31:38.648Z",
171      "updatedAt": "2022-12-12T20:31:38.648Z",
172      "_id": 0
173    },
174    ],
175    {
176      "_id": "63963a1472e1f774a6b954a4",
177      "password": "password",
178      "role": "test",
179      "email": "xj@n.com",
180      "login": "testlogin",
181      "password": "52b165905132e4530b1129c53a6b671a41a388c9f9c5f9b342",
182      "loginUrl": "http://localhost:4000/public/test_page",
183      "createdAt": "2022-12-12T20:31:38.648Z",
184      "updatedAt": "2022-12-12T20:31:38.648Z",
185      "_id": 0
186    },
187    ],
188    {
189      "_id": "63963a1472e1f774a6b954a4",
190      "password": "password",
191      "role": "test",
192      "email": "xj@n.com",
193      "login": "testlogin",
194      "password": "52b165905132e4530b1129c53a6b671a41a388c9f9c5f9b342",
195      "loginUrl": "http://localhost:4000/public/test_page",
196      "createdAt": "2022-12-12T20:31:38.648Z",
197      "updatedAt": "2022-12-12T20:31:38.648Z",
198      "_id": 0
199    },
200    ],
201    {
202      "_id": "63963a1472e1f774a6b954a4",
203      "password": "password",
204      "role": "test",
205      "email": "xj@n.com",
206      "login": "testlogin",
207      "password": "52b165905132e4530b1129c53a6b671a41a388c9f9c5f9b342",
208      "loginUrl": "http://localhost:4000/public/test_page",
209      "createdAt": "2022-12-12T20:31:38.648Z",
210      "updatedAt": "2022-12-12T20:31:38.648Z",
211      "_id": 0
212    },
213    ],
214    {
215      "_id": "63963a1472e1f774a6b954a4",
216      "password": "password",
217      "role": "test",
218      "email": "xj@n.com",
219      "login": "testlogin",
220      "password": "52b165905132e4530b1129c53a6b671a41a388c9f9c5f9b342",
221      "loginUrl": "http://localhost:4000/public/test_page",
222      "createdAt": "2022-12-12T20:31:38.648Z",
223      "updatedAt": "2022-12-12T20:31:38.648Z",
224      "_id": 0
225    },
226    ],
227    {
228      "_id": "63963a1472e1f774a6b954a4",
229      "password": "password",
230      "role": "test",
231      "email": "xj@n.com",
232      "login": "testlogin",
233      "password": "52b165905132e4530b1129c53a6b671a41a388c9f9c5f9b342",
234      "loginUrl": "http://localhost:4000/public/test_page",
235      "createdAt": "2022-12-12T20:31:38.648Z",
236      "updatedAt": "2022-12-12T20:31:38.648Z",
237      "_id": 0
238    },
239    ],
240    {
241      "_id": "63963a1472e1f774a6b954a4",
242      "password": "password",
243      "role": "test",
244      "email": "xj@n.com",
245      "login": "testlogin",
246      "password": "52b165905132e4530b1129c53a6b671a41a388c9f9c5f9b342",
247      "loginUrl": "http://localhost:4000/public/test_page",
248      "createdAt": "2022-12-12T20:31:38.648Z",
249      "updatedAt": "2022-12-12T20:31:38.648Z",
250      "_id": 0
251    },
252    ],
253    {
254      "_id": "63963a1472e1f774a6b954a4",
255      "password": "password",
256      "role": "test",
257      "email": "xj@n.com",
258      "login": "testlogin",
259      "password": "52b165905132e4530b1129c53a6b671a41a388c9f9c5f9b342",
260      "loginUrl": "http://localhost:4000/public/test_page",
261      "createdAt": "2022-12-12T20:31:38.648Z",
262      "updatedAt": "2022-12-12T20:31:38.648Z",
263      "_id": 0
264    },
265    ],
266    {
267      "_id": "63963a1472e1f774a6b954a4",
268      "password": "password",
269      "role": "test",
270      "email": "xj@n.com",
271      "login": "testlogin",
272      "password": "52b165905132e4530b1129c53a6b671a41a388c9f9c5f9b342",
273      "loginUrl": "http://localhost:4000/public/test_page",
274      "createdAt": "2022-12-12T20:31:38.648Z",
275      "updatedAt": "2022-12-12T20:31:38.648Z",
276      "_id": 0
277    },
278    ],
279    {
280      "_id": "63963a1472e1f774a6b954a4",
281      "password": "password",
282      "role": "test",
283      "email": "xj@n.com",
284      "login": "testlogin",
285      "password": "52b165905132e4530b1129c53a6b671a41a388c9f9c5f9b342",
286      "loginUrl": "http://localhost:4000/public/test_page",
287      "createdAt": "2022-12-12T20:31:38.648Z",
288      "updatedAt": "2022-12-12T20:31:38.648Z",
289      "_id": 0
290    },
291    ],
292    {
293      "_id": "63963a1472e1f774a6b954a4",
294      "password": "password",
295      "role": "test",
296      "email": "xj@n.com",
297      "login": "testlogin",
298      "password": "52b165905132e4530b1129c53a6b671a41a388c9f9c5f9b342",
299      "loginUrl": "http://localhost:4000/public/test_page",
300      "createdAt": "2022-12-12T20:31:38.648Z",
301      "updatedAt": "2022-12-12T20:31:38.648Z",
302      "_id": 0
303    },
304    ],
305    {
306      "_id": "63963a1472e1f774a6b954a4",
307      "password": "password",
308      "role": "test",
309      "email": "xj@n.com",
310      "login": "testlogin",
311      "password": "52b165905132e4530b1129c53a6b671a41a388c9f9c5f9
```

6-6-Test 3: Réponse à une requête de réinscription avec un email déjà existant



Quant on essaye d'insérer un utilisateur qui existe déjà dans la base de données, on obtient un message d'erreur indiquant que l'email existe déjà.

6-6-Test 6: Requete de suppression



Conclusion

L'API est comme un relais qui fait office de pont entre notre application et la base de données. Elle est développée avec les technologies Express et nodeJs et, devrait permettre à d'autres développeurs qui l'utiliseraient de gagner beaucoup plus en termes de temps, de coût et d'innovation. Elle est donc disponible en ligne et demeure ouverte à toutes tentatives d'amélioration pour l'intérêt de tous.