Université Polytechnique HAUTS-DE-FRANCE	Web Avance CREATION D'API AVEC NODE.JS	INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES HAUTS-DE-FRANCE

Prénom : Yazid.

Nom : GAYA.

Lien du dépôt GitHub : https://github.com/YazidGaya/api-node-ts.git

TD API

I. Tuto création d'une API avec node.JS:

Création du dossier :

```
C:\Users\yazid>mkdir api-node-ts && cd api-node-ts
```

Création du fichier package.json :

```
C:\Users\yazid\api-node-ts>npm init -y
Wrote to C:\Users\yazid\api-node-ts\package.json:

{
  "name": "api-node-ts",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}
```

Installation de express et TypeScript :

```
C:\Users\yazid\api-node-ts>npm install express dotenv

added 69 packages, and audited 70 packages in 2s

17 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

```
C:\Users\yazid\api-node-ts>npm install -D typescript ts-node @types/node @types/express nodemon

added 58 packages, and audited 128 packages in 5s

21 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Configurer type script, Les commandes :

npx tsc → exécute le compilateur TypeScript.

nit → crée un fichier tsconfig.json à la racine du projet.

```
C:\Users\yazid\api-node-ts>npx tsc --init

Created a new tsconfig.json

You can learn more at https://aka.ms/tsconfig
```

Dans le fichier tsconfig.ts on écrit :

```json
{
  "compilerOptions": {
    "target": "ES6",
    "module": "CommonJS",
    "outDir": "./dist",
    "rootDir": "./src",
    "esModuleInterop": true,
    "resolveJsonModule": true,
    "strict": true
  }
}
```

Création d'API : création du dossier src ensuite le fichier index.ts et remplacer le code donné sur le pdf dans notre fichier:

```
C:\Users\yazid\api-node-ts>

C:\Users\yazid\api-node-ts>dir src
 Le volume dans le lecteur C s'appelle Windows
 Le numéro de série du volume est F871-D3DB

 Répertoire de C:\Users\yazid\api-node-ts\src

04/11/2025  02:47 PM    <DIR>          .
04/11/2025  02:47 PM    <DIR>          ..
               0 fichier(s)              0 octets
               2 Rép(s)  57 008 291 840 octets libres

C:\Users\yazid\api-node-ts>cd src

C:\Users\yazid\api-node-ts\src>echo // index.ts > index.ts

C:\Users\yazid\api-node-ts\src>cd ..

C:\Users\yazid\api-node-ts>npx ts-node src/index.ts
[dotenv@17.2.3] injecting env (0) from .env -- tip: 📡 add obser
vability to secrets: https://dotenvx.com/ops
✅ Serveur démarré sur http://localhost:3000
```

Vérification :

🚀 API Node.js avec TypeScript fonctionne !

Ajouter un routeur Express : Création du fichier user.routes.ts et remplacer le code donné sur le pdf dans notre fichier pour gérer les chemins users .

Explication :

- Router() : permet de regrouper les routes liées aux utilisateurs dans un mini-routeur.
- Chaque ligne router.get/post/put/delete associe un url et une méthode HTTP à une fonction du contrôleur

Ajouter un Contrôleur : Créez un dossier src/controllers/ et un fichier user.controller.ts

Explication :

- getUsers renvoie **la vraie liste d'utilisateurs** (JSON) depuis MongoDB.
- addUser crée un document User en base et renvoie l'objet créé.
- J'ai ajouté getUserById, updateUser, deleteUser () en suivant les consignes que notre professeur nous a donné (pour un CRUD complet)

```
:\Users\yazid>cd C:\Users\yazid\api-node-ts

:\Users\yazid\api-node-ts>npx ts-node src/index.ts
dotenv@17.2.3] injecting env (0) from .env -- tip: 🛠  run anywhe
✅ Serveur démarré sur http://localhost:3000
```

Ajouter un fichier .env :

```
C:\Users\yazid\api-node-ts>type nul > .env
```

```
.env
1   PORT = 4000
```

Configurer Nodemon : Ajoute d'un fichier nodemon.json

```
nodemon.jason
{
  "watch": ["src"],
  "ext": "ts",
  "exec": "ts-node -r dotenv/config src/index.ts"
}
```

Explication :

watch "src" : Nodemon pour surveille tous les fichiers dans src; ext: "ts" : réagit aux modifications de fichiers TS; exec : commande lancée par Nodemon à chaque redémarrage :

- ts-node exécute le fichier TypeScript directement.
- -r dotenv/config charge automatiquement les variables d'environnement.
- src/index.ts est le point d'entrée de l'application.

Ajouter les scripts dans package.json :

```
"scripts": {
  "start": "node dist/index.js",
  "dev": "nodemon src/index.ts",
  "build": "tsc"
},
```

"start": "node dist/index.js" : pour lancé le mode compilé; "dev": "nodemon src/index.ts" : lance le serveur en mode développement avec rechargement automatique; "build": "tsc" : compile le TS de src/ vers du JavaScript

Lancer l'API : *Démarrez l'API avec la commande npm run dev*

```
C:\Users\yazid\api-node-ts>type nul > nodemon.jason

C:\Users\yazid\api-node-ts>npm run dev
```

```
> api-node-ts@1.0.0 dev
> nodemon src/index.ts

[nodemon] 3.1.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: ts,json
[nodemon] starting `ts-node src/index.ts`
[dotenv@17.2.3] injecting env (1) from .env -- tip: ⚙️  su
h { quiet: true }
✅ Serveur démarré sur http://localhost:4000
[nodemon] restarting due to changes...
[nodemon] starting `ts-node src/index.ts`
[dotenv@17.2.3] injecting env (1) from .env -- tip: 🔐 pre
v to code: https://dotenvx.com/precommit
```

On vérifie bien que le serveur démarre sur le port 4000.

Tester l'API (avec cURL) :

Tester GET /users:

```
C:\Users\yazid>cd C:\Users\yazid\api-node-ts

C:\Users\yazid\api-node-ts>curl -X GET http://localhost:3000/users
{"message":"Liste des utilisateurs"}
C:\Users\yazid\api-node-ts>
```

Tester POST /users:

```
C:\Users\yazid\api-node-ts>curl -X POST http://localhost:3000/users -H "Content-Type: application/json" -d "{\"name\": \"Alice\", \"e
mail\": \"alice@example.com\"}"
{"message":"Utilisateur Alice ajouté avec succès !","email":"alice@example.com"}
C:\Users\yazid\api-node-ts>
```

Que faire en cas d'erreur (port déjà utilisé) :

Vérifications des processus :

```
C:\Users\yazid>netstat -ano | findstr :4000
  TCP    0.0.0.0:4000           0.0.0.0:0              LISTENING       36400
  TCP    [::]:4000              [::]:0                 LISTENING       36400
  TCP    [::1]:50974            [::1]:4000             TIME_WAIT       0

C:\Users\yazid>
```

Exécution :

```
C:\Users\yazid>taskkill /PID 36400 /F
Opération réussie : le processus avec PID 36400 a été terminé.

C:\Users\yazid>
```

Résultat :

```
[dotenv@17.2.3] injecting env (1) from .env -- tip: ⚙ load multiple .
✅ Serveur démarré sur http://localhost:4000
[nodemon] app crashed - waiting for file changes before starting...
```

2. Suite du TD / TP :

Ajouter un stockage en mémoire: On doit modifier user.controller.ts pour utiliser un tableau en mémoire et on rajoute une variable users pour stocker les utilisateurs (copier le code donnée sur le tp)

Pour stocker les utilisateur j'ai utilisé **MongoDB + Mongoose** :

Donc j'ai du modifier le Contrôleur: pour que getUsers renvoie **la vraie liste d'utilisateurs** JSON depuis MongoDB; addUser crée un document User en base et renvoie l'objet créé; et j'ai ajouté getUserById, updateUser, deleteUser pour avoir une version plus complète

```typescript
import { Request, Response } from "express";
import User from "../models/user.model";

// En utilisant GET /users pour récupérer tous les utilisateurs
export const getUsers = async (req: Request, res: Response) => {
  const users = await User.find();
  res.json(users);
};

// En utilisant GET /users/:id pour Récupérer un utilisateur spécifique
Complexity is 3 Everything is cool!
export const getUserById = async (req: Request, res: Response) => {
  const user = await User.findById(req.params.id);

  if (!user) {
    return res.status(404).json({ message: "Utilisateur non trouvé" });
  }

  res.json(user);
};

// En utilisant POST /users pour Ajouter un utilisateur
Complexity is 4 Everything is cool!
export const addUser = async (req: Request, res: Response) => {
  const { name, email } = req.body;

  if (!name || !email) {
    return res.status(400).json({ message: "Nom et email requis" });
  }

  const newUser = await User.create({ name, email });
  res.json(newUser);
};

// En utilisant PUT /users/:id pour mettre à jour un utilisateur
Complexity is 3 Everything is cool!
export const updateUser = async (req: Request, res: Response) => {
  const user = await User.findByIdAndUpdate(
    req.params.id,
    req.body,
    { new: true } // renvoie l'utilisateur mis à jour
  );

  if (!user) {
    return res.status(404).json({ message: "Utilisateur non trouvé" });
  }

  res.json(user);
};

// En utilisant DELETE /users/:id pour supprimer un utilisateur
Complexity is 3 Everything is cool!
export const deleteUser = async (req: Request, res: Response) => {
  const user = await User.findByIdAndDelete(req.params.id);

  if (!user) {
    return res.status(404).json({ message: "Utilisateur non trouvé" });
  }

  res.json({ message: "Utilisateur supprimé" });
};
```

J'ai rajouté le fichier models pour définir la structure d'un utilisateur en base de donnée en lui ajoutant un nom et un email

```ts
src > models > TS user.model.ts > ...
import mongoose from "mongoose";

const userSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true }
});

export default mongoose.model("User", userSchema);

```

Et j'ai ajouté le fichier db.tspour avoir une connexion à MongoDB, cette fonction définit dans mon fichier sera appelée dans index.ts (connectDB();) avant de démarrer le serveur.
Elle assure une **connexion à MongoDB locale** sur la base api-node-ts

Test MongoDB :

Au début notre liste est vide :

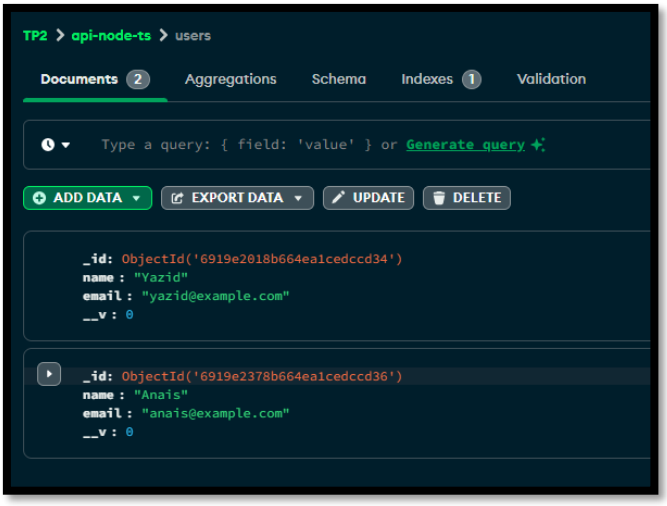
```
C:\Users\yazid>curl -X GET http://localhost:4000/users
[]
C:\Users\yazid>
```

Notre base de donnée est vide donc la liste des utilisateurs est bien vide.

POST /users (ajouter deux utilisateur)

```
C:\Users\yazid\api-node-ts>curl -X POST http://localhost:4000/users -H "Content-Type: application/json" -d "{\"name\":\"
Yazid\",\"email\":\"yazid@example.com\"}"
{"name":"Yazid","email":"yazid@example.com","_id":"6919e2018b664ea1cedccd34","__v":0}
C:\Users\yazid\api-node-ts>curl -X POST http://localhost:4000/users -H "Content-Type: application/json" -d "{\"name\":\"
Anais\",\"email\":\"anais@example.com\"}"
{"name":"Anais","email":"anais@example.com","_id":"6919e2378b664ea1cedccd36","__v":0}
C:\Users\yazid\api-node-ts>
```

Vérification sur ma base de donnée MongoDB :



GET /users/:id (récupérer les utilisateurs utilisateur)

```
c:\Users\yazid\api-node-ts>curl -X GET http://localhost:4000/users
[{"_id":"6919e2018b664ea1cedccd34","name":"Yazid","email":"yazid@example.com","__v":0},{"_id":"6919e2378b664ea1cedccd36"
,"name":"Anais","email":"anais@example.com","__v":0}]
c:\Users\yazid\api-node-ts>
```

Mettre à jour un utilisateur : changer l'adresse mail de Anais :

```
c:\Users\yazid\api-node-ts>curl -X PUT http://localhost:4000/users/6919e2378b664ea1cedccd36 ^
Plus ? -H "Content-Type: application/json" ^
Plus ? -d "{\"email\":\"anais.2004@example.com\"}"
{"_id":"6919e2378b664ea1cedccd36","name":"Anais","email":"anais.2004@example.com","__v":0}
c:\Users\yazid\api-node-ts>
```

Vérification sur la base de donnée :