PARTIE A Installations et configurations

1) Installation Node JS

Installer la dernier version sur : https://nodejs.org/en/download
 node -v

2) Création de projet

- Dans le répertoire de travail taper npm init.
- Tapez entrée

```
→ tp_produit_back npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.
See `npm help init` for definitive documentation on these fields
and exactly what they do.
Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.
Press ^C at any time to quit.
package name: (tp_produit_back)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to /Users/avan@r8k/Documents/tp_produit_back/package.json:
  "name": "tp_produit_back",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  "author": "",
  "license": "ISC"
}
```

3) Installation des dépendances

npm install express npm install typescript npm install ts-node

Is this OK? (yes)

npm install cors npm install nodemon --save-dev npm i typeorm reflect-metadata – save npm i body-parser

4) Configuration typescript

- Taper tsc init
- Dans le fichier **tsconfig.json**, dé-commenter la ligne remplacer la ligne commençant par //"outDir par "outDir": "./dist". Sauvegarder

```
57 // "outFile": "./", /* Specify a file that bundles all outputs into
58 "outDir": "./dist", /* Specify an output folder for all emitted fi
59 // "removeComments": true, /* Disable emitting comments. */
```

Dans le fichier package.json, ajouter cette ligne :

```
"scripts": {
    "start": "nodemon index.ts",
    "test": "echo \"Error: no test specified\" && exit 1"
},
```

5) Installation des types

```
npm i -- save-dev @types/express
npm i -- save-dev @types/cors
```

6) Tests

Créer un fichier index.js et ajouter ce code :

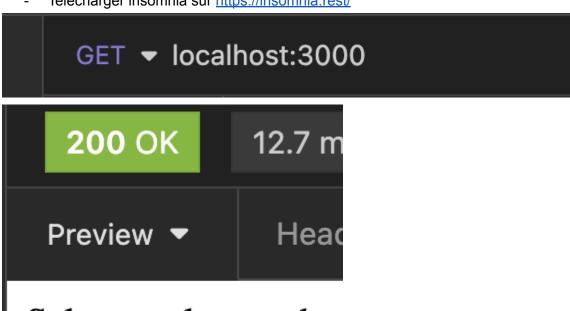
```
import express from 'express';
     import cors from 'cors';
     const app = express();
4
     const port = 3000;
     app.use(cors());
6
     app.get('/', (req, res) => {
      res.send('Hello World!');
8
     });
     app.listen(port, () => {
      console.log(`Le serveur est en écoute sur le port ${port}`);
10
     });
11
12
```

Exécuter la commande npm start

```
tp_produit_back npm start
> tp_produit_back@1.0.0 start
> nodemon index.ts
[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: ts,json
[nodemon] starting `ts-node index.ts`
Le serveur est en écoute sur le port 3000
```

7) Tester l'api sur Insomnia

Telecharger insomnia sur https://insomnia.rest/



Salut tout le monde

PARTIE B Intégration MYSQL et TYPEORM

1) Création de base de données

- Installer mysql
- Taper mysql -u root p : si zsh: command not found: mysql, faire
 → ~ export PATH=\${PATH}:/usr/local/mysql/bin/
 - → ~ source ~/.zshrc # If you use Oh-My-Zsh (MAC OS ONLY)
- Créer une base de données

```
[mysql> CREATE DATABASE tp_produit;
Query OK, 1 row affected (0.00 sec)
```

2) Création de la table/connexion BDD/TypeORM

- Dans le dossier src, créer un dossier entity
- Dans le dossier entity, créer un fichier produit.entity.ts

```
produit.entity.ts
import { Entity, Column, PrimaryGeneratedColumn } from "typeorm"

// Définition d'une entité avec la décoration @Entity()
@Entity()
export class Produit {

// Définition d'une colonne primaire auto-générée avec la décoration
@PrimaryGeneratedColumn()
@PrimaryGeneratedColumn()
id: string | number

// Définition d'une colonne nommée "nomProduit" de type string
@Column()
nomProduit: string
// Définition d'une colonne nommée "nomClient" de type string
@Column()
nomClient: string
// Définition d'une colonne nommée "prix" de type nombre
@Column()
prix: number
```

- Dans la racine du projet, créer un fichier app-data-source.ts

```
app-data-source.ts
import { DataSource } from "typeorm"

export const myDataSource = new DataSource({
    type: "mysql", // Type de base de données
    host: "localhost", // Hôte de la base de données
    port: 3306, // Port utilisé par la base de données
    username: "root", // Nom d'utilisateur pour la connexion à la base de données
    password: "00600060", // Mot de passe pour la connexion à la base de données
    database: "tp_produit", // Nom de la base de données
    entities: ["src/entity/*.ts"],// Chemin des fichiers d'entités TypeScript
    //logging: true, // Activation des journaux de requêtes SQL
    synchronize: true, // Synchronisation automatique des schémas (création des
    tables, etc.)
})
```

- Dans le dossier src. créer un dossier services
- Dans le dossier service, creer un fichier services.ts
- Définir les fonctions

```
import { Request, Response } from 'express';
import { FindOneOptions } from 'typeorm';
import { myDataSource } from '../../app-data-source';
import { Produit } from '../produit.entity';

// Fonction pour obtenir tous les produits
export async function getProduits(req: Request, res: Response) {
const produits = await myDataSource.getRepository(Produit).find();
res.json(produits);
}

// Fonction pour obtenir un produit par son ID
export async function getProduitByID(req: Request, res: Response) {
const id: number = parseInt(req.params.id, 10);
const options: FindOneOptions<Produit> = {
where: { id },
```

```
const produit = await myDataSource
.getRepository(Produit)
.findOne(options);
if (produit) {
res.json(produit);
res.sendStatus(404);
export async function addProduit(req: Request, res: Response) {
const produit = myDataSource.getRepository(Produit).create(req.body);
const results = await myDataSource.qetRepository(Produit).save(produit);
return res.send(results);
const { id } = req.params;
const { prix } = req.body;
await myDataSource.getRepository(Produit).update(id, {    prix });
res.sendStatus(200);
export async function deleteProduit(req: Request, res: Response) {
const results = await myDataSource
.getRepository(Produit)
.delete(req.params.id);
return res.send(results);
```

```
getRepository(Produit).find(): Cette fonction récupère tous les produits de la base de données en utilisant le repository associé à l'entité Produit.
```

```
spécifique de la base de données en utilisant le repository associé
à l'entité Produit. La fonction findOne prend un paramètre
options de type FindOneOptions<Produit> qui peut inclure
des conditions de recherche, comme la recherche par ID.
getRepository(Produit).create(reg.body): Cette fonction crée une
nouvelle instance de l'entité Produit en utilisant
les données contenues dans req.body.
getRepository(Produit).save(produit): Cette fonction sauvegarde
un produit dans la base de données en utilisant le
repository associé à l'entité Produit. Elle prend en
paramètre l'instance de l'entité Produit à sauvegarder.
getRepository(Produit).update(id, { prix }): Cette fonction met
à jour le prix d'un produit dans la base de données en utilisant
le repository associé à l'entité Produit. Elle prend en
paramètre l'ID du produit à mettre à jour et un objet
getRepository(Produit).delete(req.params.id):Supprime
un produit de la bdd en utilisant le repository associé
à l'entité Produit. Son parametre l'ID du produit à supprimer.
```

- Dans le dossier src, créer un dossier routes
- Dans le dossier service, creer un fichier routes, ts
- Définir les routes

```
routes.ts
import express from 'express';
```

```
import {
getProduits,
getProduitByID,
addProduit,
setPrix,
deleteProduit,
const bodyParser = require('body-parser');
const router = express.Router();
router.use(bodyParser.json());
// Route pour obtenir tous les produits
router.get('/produits', getProduits);
router.get('/produits/:id', getProduitByID);
// Route pour ajouter un produit
router.post('/produits', addProduit);
router.put('/produits/:id', setPrix);
router.delete('/produits/:id', deleteProduit);
export default router;
```

Modifier le fichier index.ts comme suit

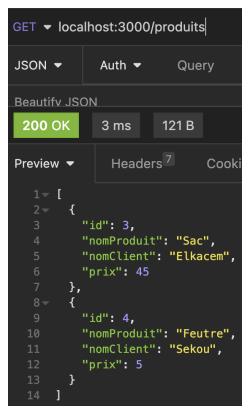
```
index.ts
import express from "express";
import { myDataSource } from "./app-data-source";
import router from "./src/entity/routes/routes";
const bodyParser = require('body-parser')
const app = express();
```

```
app.use(bodyParser.json());
// Connexion avec la base de données
myDataSource
.initialize()
.then(() => {
  console.log("Data Source has been initialized!");
  app.listen(3000, () => {
    console.log("Server started on port 3000");
  });
});
catch((err) => {
  console.error("Error during Data Source initialization:", err);
});
app.use("/",router); //Utilisation des routes
```

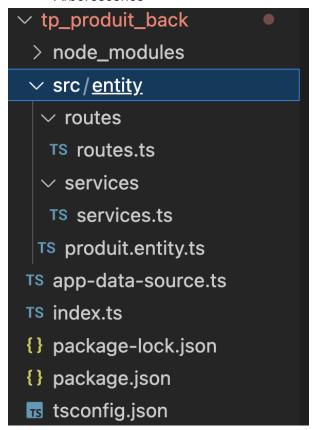
- Taper npm start

Data Source has been initialized! Server started on port 3000

TEST INSOMNIA



- Arborescence



- Fichier tsconfig.json

```
"compilerOptions": {
"experimentalDecorators": true,
"strictPropertyInitialization": false,
"emitDecoratorMetadata" : true,
"skipLibCheck": true,
"target": "es2016",
"outDir": "./dist",
"strict": true,
"forceConsistentCasingInFileNames": true,
"esModuleInterop": true,
"module": "commonjs",
}
```