TD1

Backend with Node.js & Express.js

Express

https://www.jordansablon.fr/enseignement

https://github.com/jsablonEnseignement/td1_sources

Sommaire

→ Récupération du code source du projet

→ Node Package Manager (NPM)

→ Présentation des fichiers

→ Installation des dépendances

→ Premier démarrage du serveur

→ Installation et configuration de Thunder Client

→ Mise en place de la structure des dossiers / fichiers

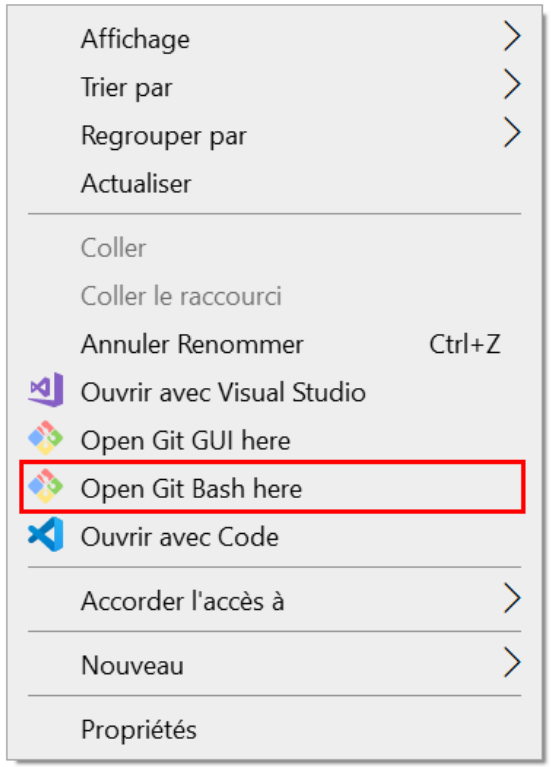
→ Utilisation de « mocks » de données

→ Implémentation des services / contrôleurs / routes

Récupération du code source du projet

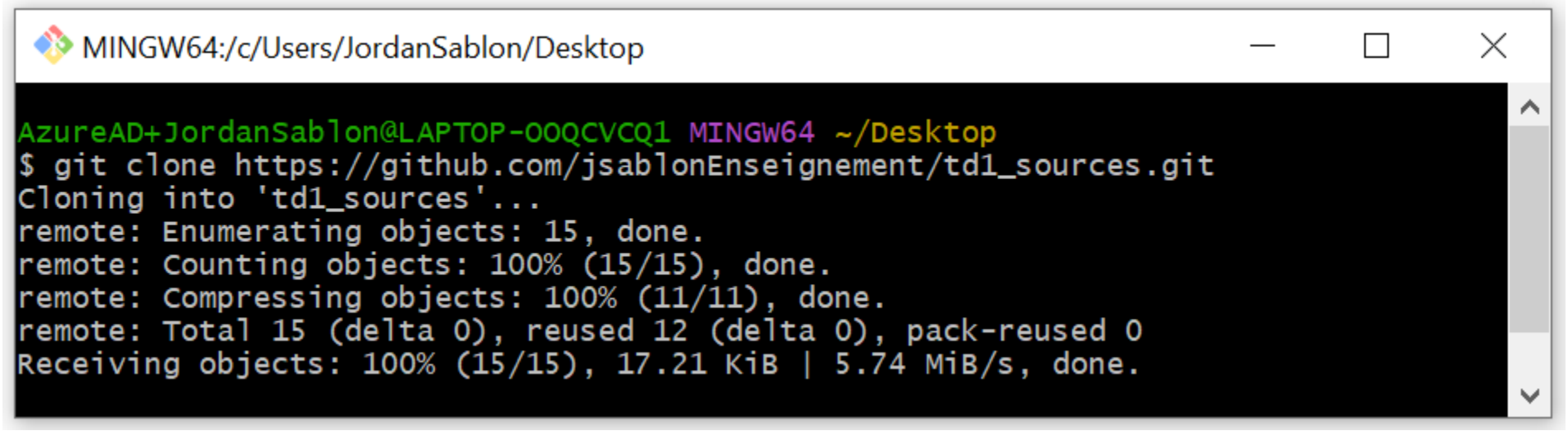
1. Se positionner à l'emplacement de votre choix (où se trouvera le futur dossier du projet)

2. Ouvrir un invité de commande à l'emplacement précédent :



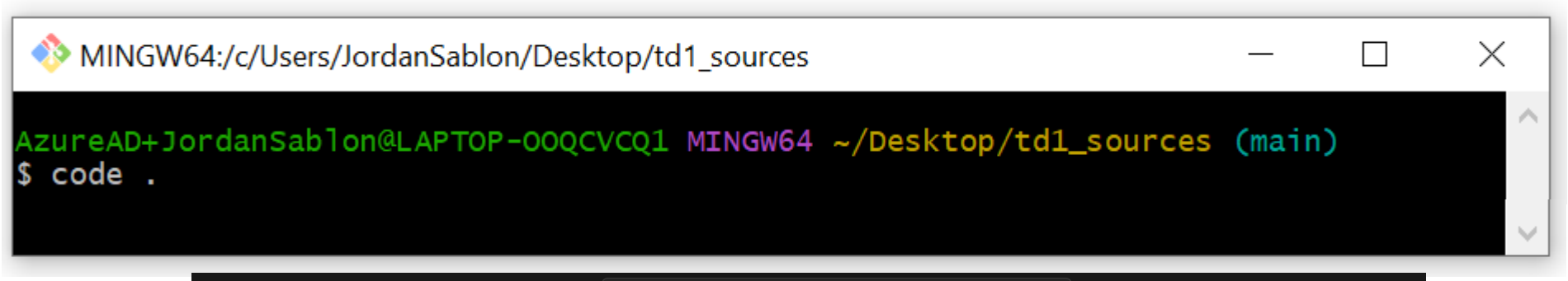
Récupération du code source du projet

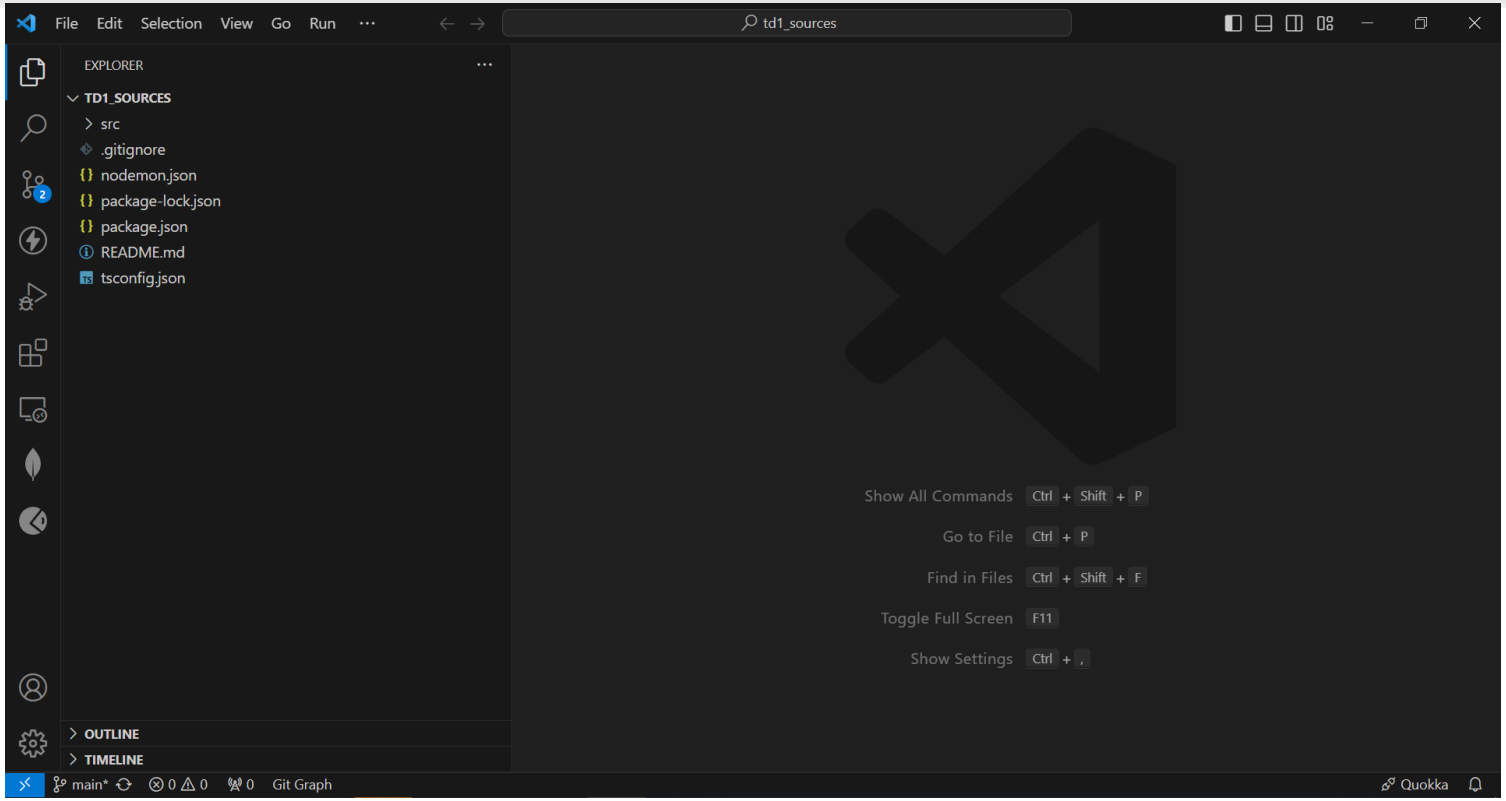
3. Cloner le répertoire Git : https://github.com/jsablonEnseignement/td1_sources



4. Se rendre dans le dossier précédemment cloné :

5. Ouvrir le projet dans VSCode :





Node Package Manager (NPM)

npm signifie **Node Package Manager**

C'est une bibliothèque et un registre pour les packages logiciels JavaScript.

npm dispose également d'outils de ligne de commande pour vous aider à installer les différents packages et à gérer leurs dépendances.

About npm

.gitignore

Permet de spécifier l'ensemble des extensions, fichiers, ou dossiers à exclure du contrôle de sources

About .gitignore

nodemon.json

Fichier de configuration de **nodemon**.

nodemon est un outil qui permet de redémarrer automatiquement l'application lorsque des modifications de fichier dans le répertoire sont détectées.

About nodemon

tsconfig.json

Spécifie les fichiers racines et les options du compilateur requises pour compiler le projet en TypeScript.

TypeScript ajoute un typage statique optionnel et des fonctionnalités avancées à JavaScript

Il permet de détecter les erreurs de syntaxe / compilation tout au long du développement.

Le code TypeScript est converti en JavaScript, il s'exécute donc partout où JavaScript s'exécute.

About ts-config.json

About TypeScript

TypeScript Documentation

package.json

Contient diverses métadonnées relatives au projet

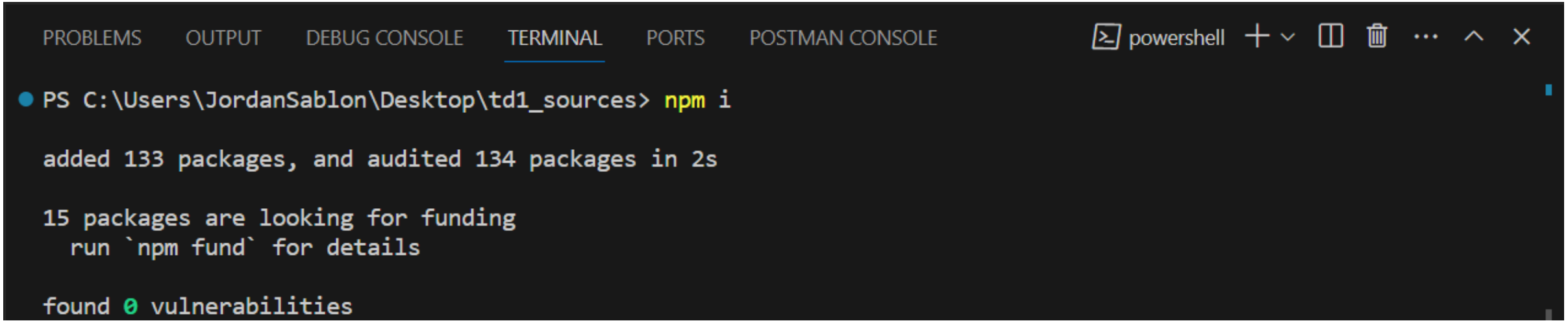
Ce fichier est utilisé par NPM pour identifier le projet et gérer les dépendances

Il peut également contenir d'autres métadonnées telles qu'une description du projet, sa version, des informations sur la licence, un ensemble de commandes, voire des données de configuration

About package.json

Installation des dépendances

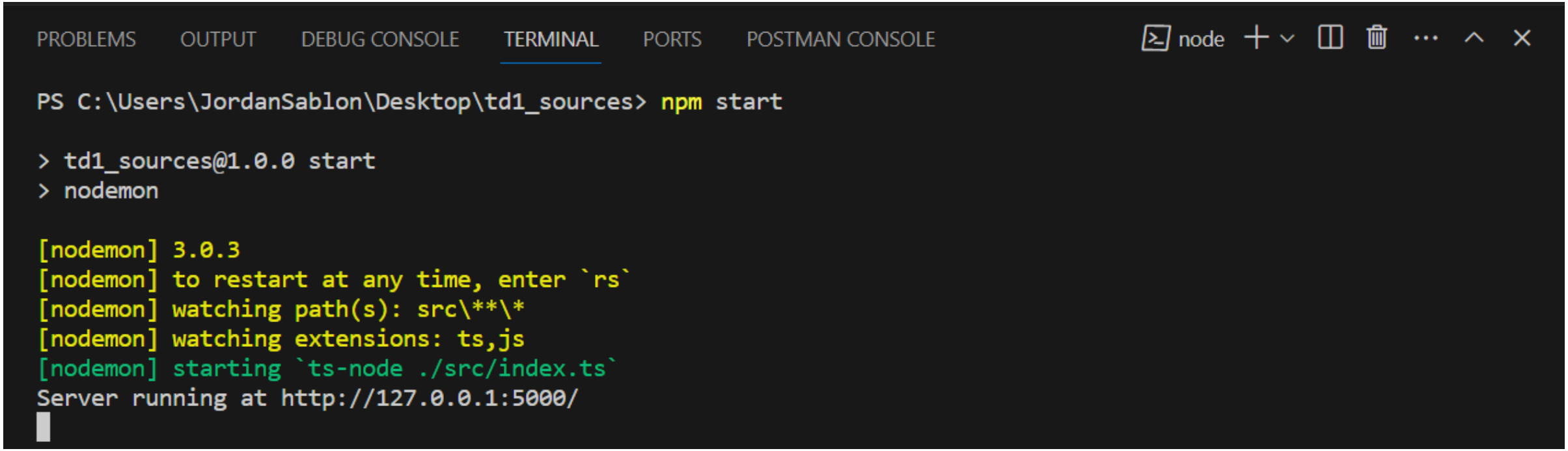
Depuis le terminal de VSCode, exécuter la commande suivante :



Les dépendances nécessaires au bon fonctionnement de l'application viennent d'être installées

Premier démarrage du serveur

Depuis le terminal de VSCode, exécuter la commande suivante :

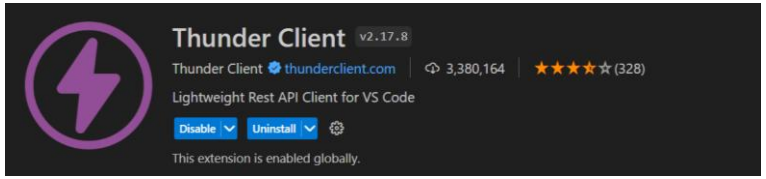


Le serveur est maintenant démarré et est accessible via l'url http://127.0.0.1:5000/

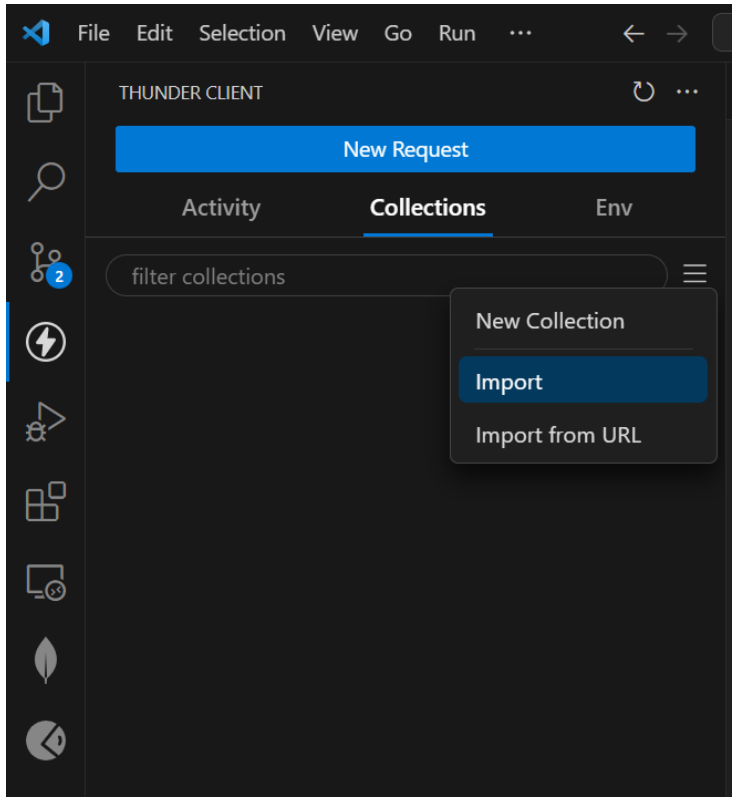
Installation et configuration de Thunder Client

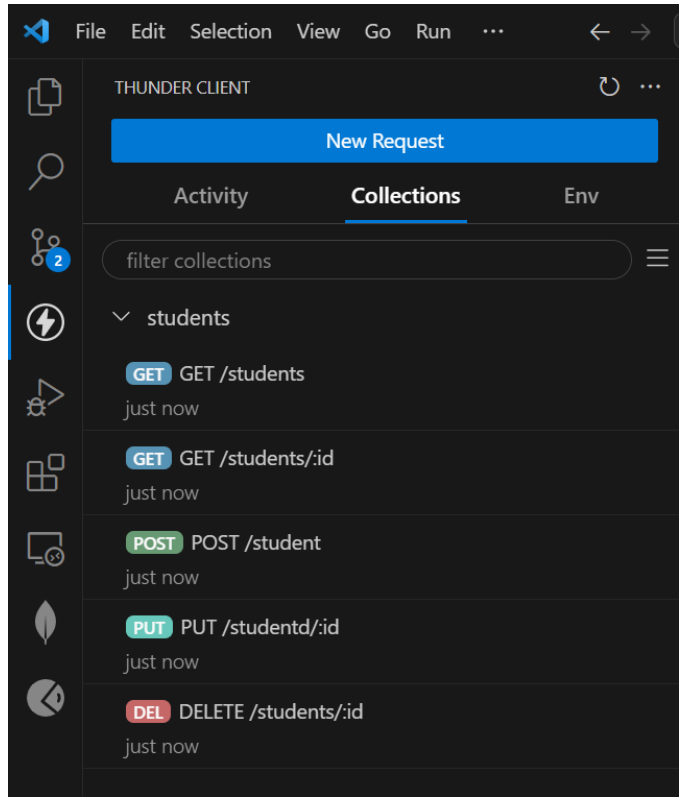
Télécharger la collection students

Pour utiliser Thunder Client, installez l'extension depuis VS Code

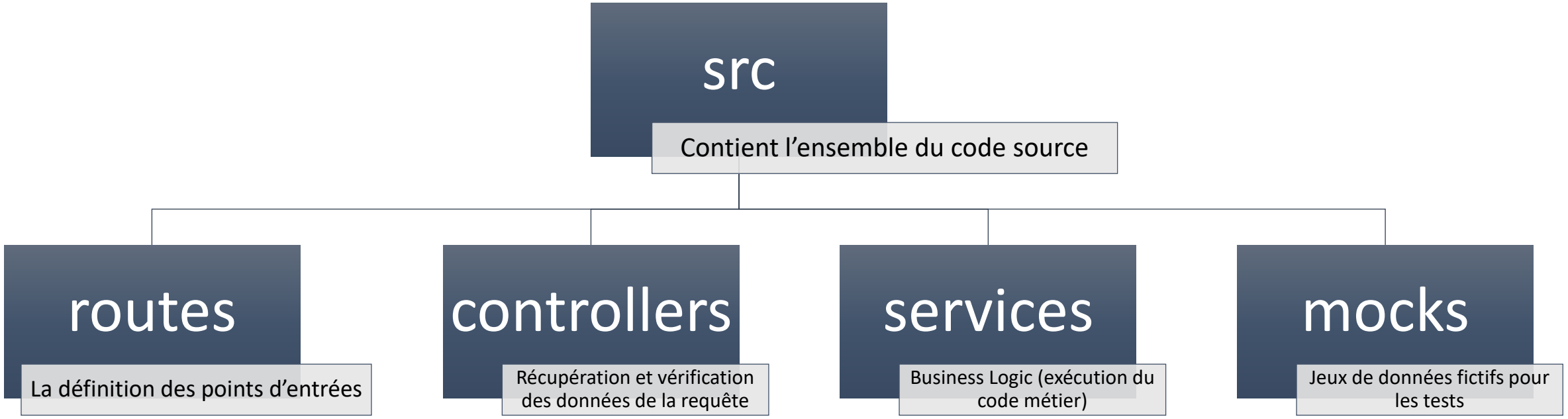


Ouvrez l'extension et importez la collection





Mise en place de la structure des dossiers / fichiers



Mise en place de la structure des dossiers / fichiers

Dans le dossier **/src**, créer les dossiers suivants :

- **routes**
- **controllers**
- **services**
- **mocks**

Dans le dossier **/routes**, créer un fichier **students.route.ts**

Dans le dossier **/controllers**, créer un fichier **students.controller.ts**

Dans le dossier **/services**, créer un fichier **students.service.ts**

Dans le dossier **/mocks**, créer un fichier **students.mock.ts**

Utilisation de « mocks » de données

Dans le fichier **students.mock.ts**, ajouter le code suivant :

```
export type Student = {
    id: number;
    name: string;
    firstname: string;
    age: number;
  };

  export let students: Array<Student> = [
    {
      id: 1,
      name: "DOE",
      firstname: "John",
      age: 18,
    },
    {
      id: 2,
      name: "DUPONT",
      firstname: "Martin",
      age: 20,
    },
  ];
```

Récupération de tous les étudiants

Dans le fichier **students.service.ts**, importer le jeu de données précédemment créé :

```
import { Student, students } from "../mocks/students.mock";
```

Puis créer une fonction qui retourne l'ensemble des étudiants du jeu de données :

```
const getStudents = () => {
  return students;
};
```

On exporte la fonction afin de pouvoir l'utiliser en dehors de notre fichier de service

```
export {
  getStudents
};
```

Récupération de tous les étudiants

Dans le fichier **students.controller.ts**, importer la totalité des fonctions du service :

```
import * as StudentsService from "../services/students.service";
```

Puis créer une fonction qui fait appel au service de récupération des étudiants :

```
export const getStudents = (req: any, res: any) => {
  const students = StudentsService.getStudents();
  return res.status(200).json(students);
};
```

ⓘ La fonction retournera les étudiants retournés par le service au format json avec un statut 200

ⓘ **req** et **res** correspondent respectivement à la requête reçue et la réponse retournée

Récupération de tous les étudiants

Le but est désormais de créer une route qui appellera le contrôleur **getStudents** lorsque l'utilisateur fera un appel HTTP de type GET sur l'URL **http://localhost/students/**

Pour cela, il faut spécifier dans le fichier **index.ts** que l'application doit utiliser le router **students.route.ts** lorsqu'un appel commençant par */students* est réalisé sur notre serveur :

```
app.use("/students", router);
```

Récupération de tous les étudiants

Dans le fichier **students.route.ts**, importer express :

```
import express from "express";
```

Importer la fonction **getStudents** du contrôleur :

```
import {
  getStudents
} from "../controllers/students.controller";
```

Déclarer un routeur :

```
const router = express.Router();
```

Créer la route :

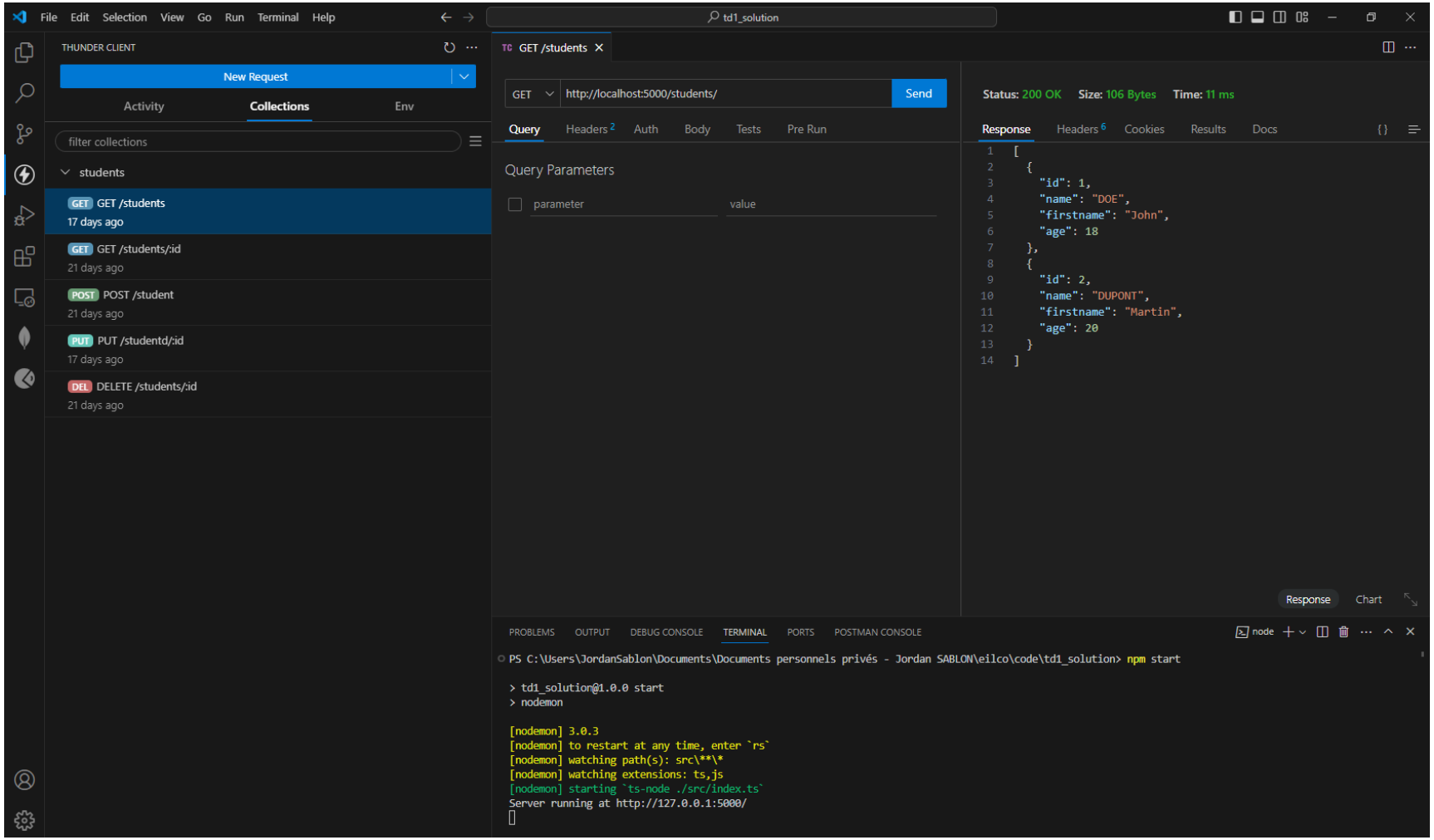
```
router.get("/", getStudents);
```

Exporter le routeur

```
export { router };
```

Récupération de tous les étudiants

Via l'extension Thunder Client, ouvrez la route **GET /students** et cliquez sur **Send**



Récupération d'un étudiant par son id

Dans le fichier **students.service.ts**, créer une fonction qui retourne un étudiant du jeu de données selon l'id passé en paramètre :

```
const getStudent = (id: string) => {
  return students.find((student) => student.id.toString() === id);
};
```

On exporte la fonction afin de pouvoir l'utiliser en dehors de notre fichier de service

```
export {
  getStudents,
  getStudent
};
```

Récupération d'un étudiant par son id

Dans le fichier **students.controller.ts**, créer une fonction qui fait appel au service de récupération d'un étudiant :

```
export const getStudent = (req: any, res: any) => {
  const { id } = req.params;
  const student = StudentsService.getStudent(id);
  return res.status(200).json(student);
};
```

- La fonction retournera l'étudiant retourné par le service au format json avec un statut 200

- L'id provient de l'URL, on le récupère depuis les paramètres de la requête : **req.params**

Récupération d'un étudiant par son id

Le but est désormais de créer une route qui appellera le contrôleur **getStudent** lorsque l'utilisateur fera un appel HTTP de type GET sur l'URL **http://localhost/students/:id**

Importer la fonction **getStudent** du contrôleur :

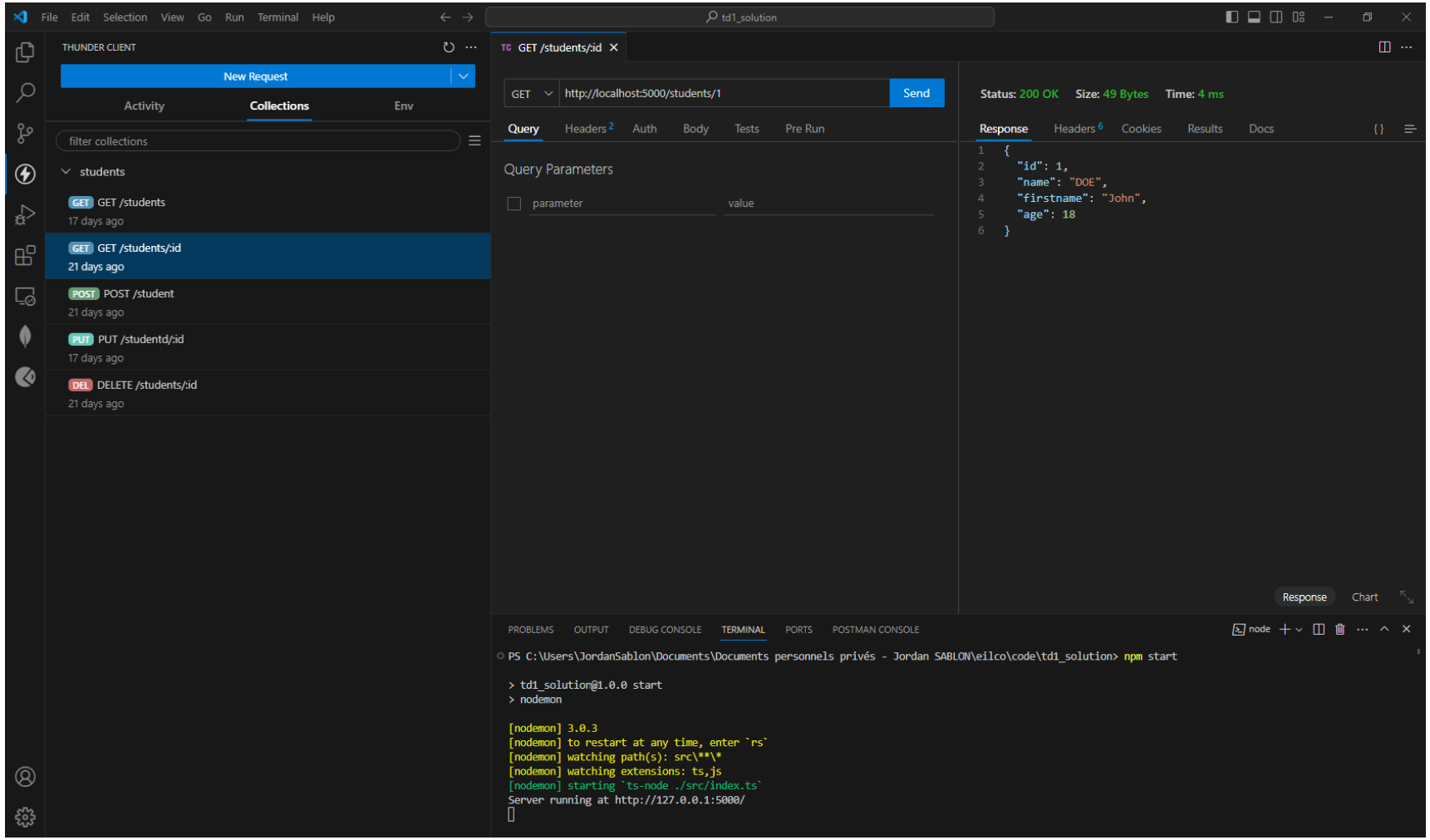
```
import {
  getStudents,
  getStudent,
} from "../controllers/students.controller";
```

Créer la route :

```
router.get("/:id", getStudent);
```

Récupération d'un étudiant par son id

Via l'extension Thunder Client, ouvrez la route **GET /student** et cliquez sur **Send**



Création d'un étudiant

Dans le fichier **students.service.ts**, créer une fonction qui ajoute dans le jeu de données un étudiant dont les informations sont passées en paramètre :

```
const createStudent = (name: string, firstname: string, age: number) => {
  const newStudent = {
    id: students.length + 1,
    name: name,
    firstname: firstname,
    age: age,
  };
  students.push(newStudent);
  return students;
};
```

On exporte la fonction afin de pouvoir l'utiliser en dehors de notre fichier de service

```
export {
  getStudents,
  getStudent,
  createStudent
};
```

Création d'un étudiant

Dans le fichier **students.controller.ts**, créer une fonction qui fait appel au service de création d'un étudiant :

```
export const createStudent = (req: any, res: any) => {
  const { name, firstname, age } = req.body;
  const students = StudentsService.createStudent(name, firstname, age);
  return res.status(200).json(students);
};
```

- La fonction retournera la liste de tous les étudiants au format json avec un statut 200

- Les informations de l'étudiant à créer proviennent du corps de la requête (body), on les récupère depuis le body de la requête : **req.body**

Création d'un étudiant

Le but est désormais de créer une route qui appellera le contrôleur **createStudent** lorsque l'utilisateur fera un appel HTTP de type POST sur l'URL **http://localhost/students/**

Importer la fonction **createStudents** du contrôleur :

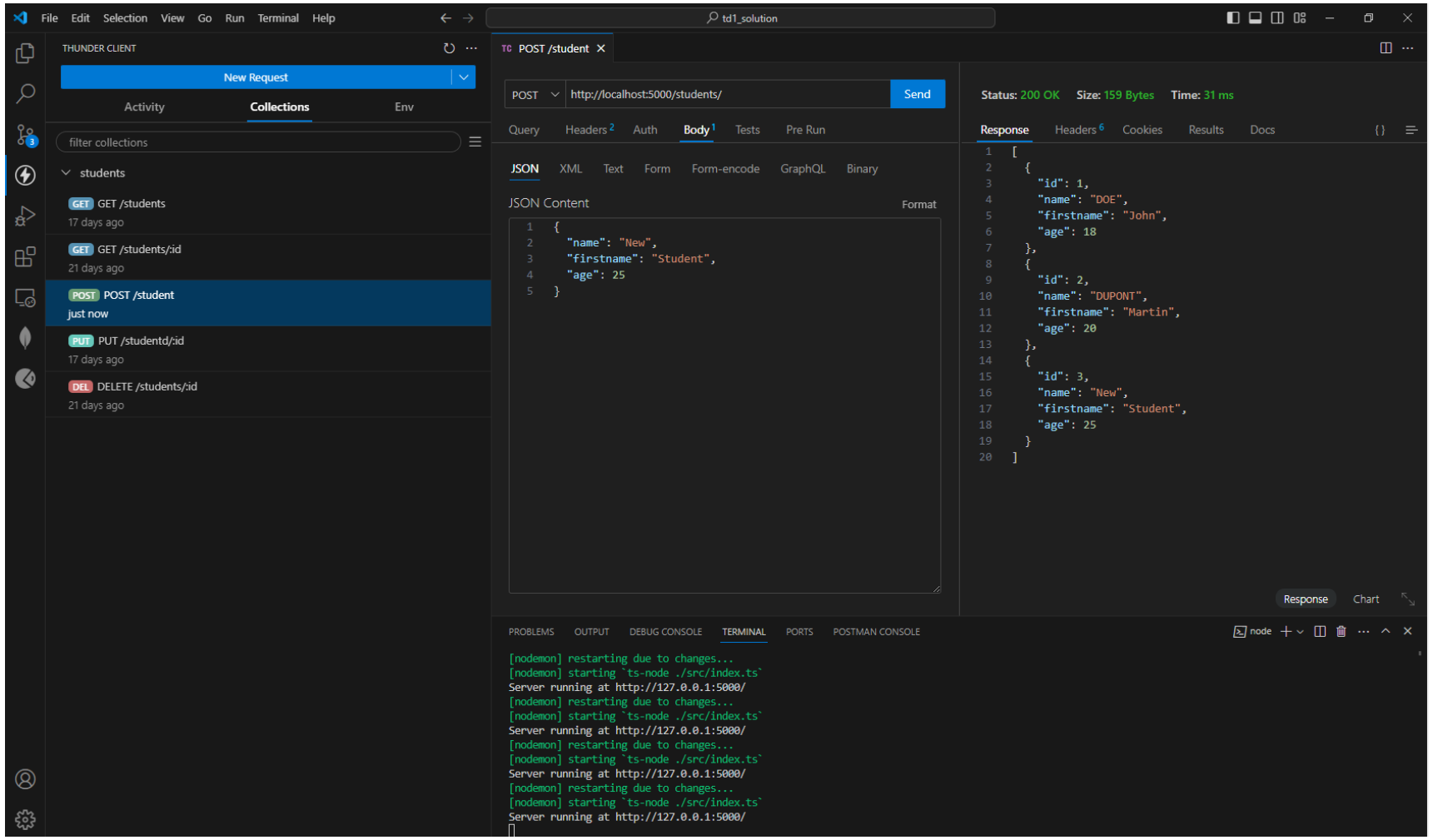
```
import {
  getStudents,
  getStudent,
  createStudent
} from "../controllers/students.controller";
```

Créer la route :

```
router.post("/", createStudent);
```

Création d'un étudiant

Via l'extension Thunder Client, ouvrez la route **POST /students** et cliquez sur **Send**



Mise à jour d'un étudiant

Dans le fichier **students.service.ts**, créer une fonction met à jour un étudiant dans le jeud e données selon l'id passé en paramètre :

```
const updateStudent = (id: string, body: Student) => {
  const studentToUpdate = getStudent(id);
  if (studentToUpdate) {
    studentToUpdate.name = body.name;
    studentToUpdate.firstname = body.firstname;
    studentToUpdate.age = body.age;
    return studentToUpdate;
  }
};
```

On exporte la fonction afin de pouvoir l'utiliser en dehors de notre fichier de service

```
export {
  getStudents,
  getStudent,
  createStudent,
  updateStudent,
};
```

Mise à jour d'un étudiant

Dans le fichier **students.controller.ts**, créer une fonction qui fait appel au service de mise à jour d'un étudiant :

```
export const updateStudent = (req: any, res: any) => {
  const { id } = req.params;
  const studentUpdated = StudentsService.updateStudent(id, req.body);
  return res.status(200).json(studentUpdated);
};
```

- La fonction retournera l'étudiant retourné par le service au format json avec un statut 200
- L'id provient de l'URL, on le récupère depuis les paramètres de la requête : **req.params**
- Les informations de l'étudiant à mettre à jour sont présentes dans le corps de la requête, on passe donc directement **req.body** à notre service

Mise à jour d'un étudiant

Le but est désormais de créer une route qui appellera le contrôleur **updateStudent** lorsque l'utilisateur fera un appel HTTP de type PUT sur l'URL **http://localhost/students/:id**

Importer la fonction **updateStudents** du contrôleur :

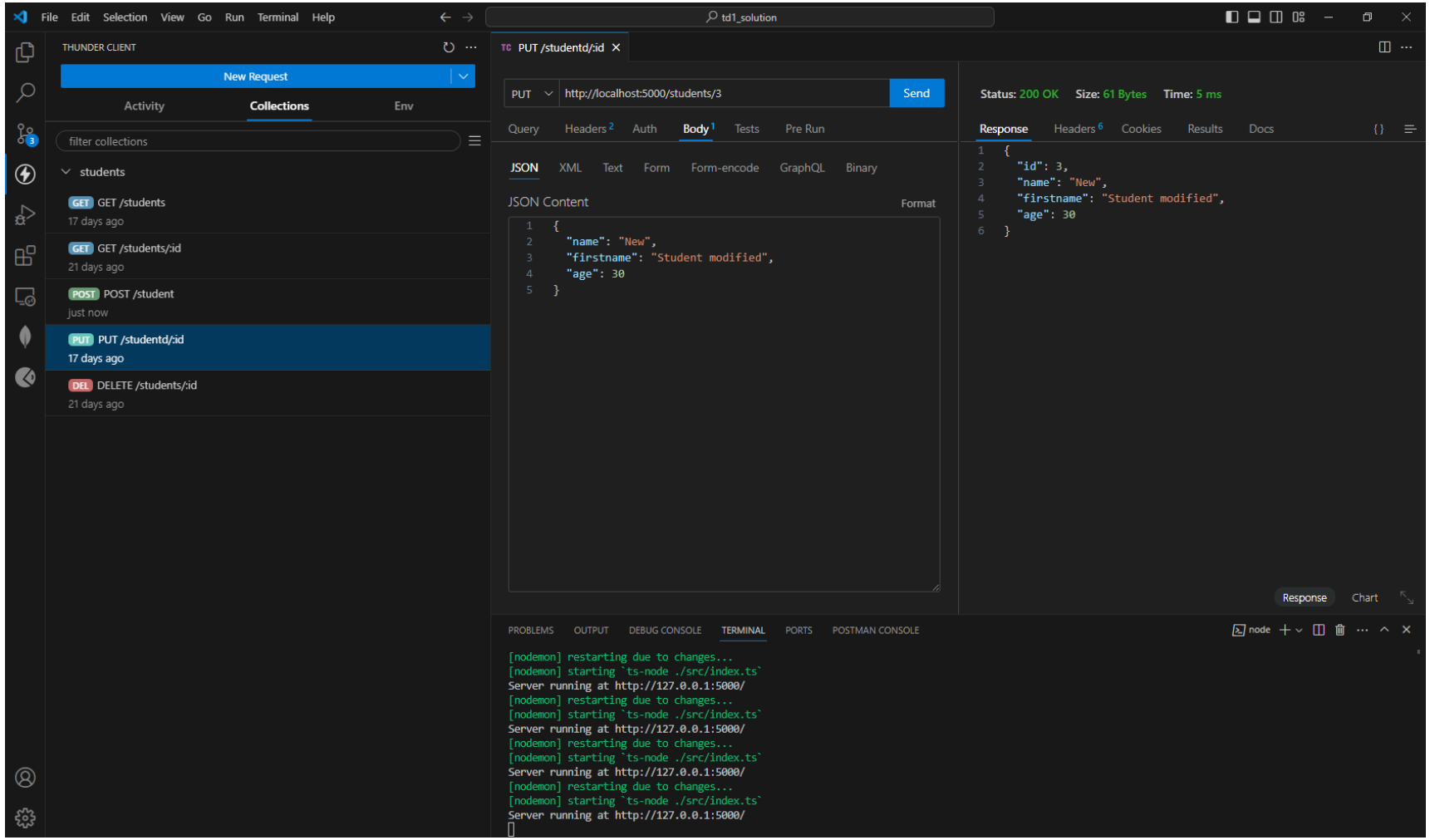
```
import {
  getStudents,
  getStudent,
  createStudent,
  updateStudent

} from "../controllers/students.controller";
```

Créer la route :

```
router.put("/:id", updateStudent);
```

Mise à jour d'un étudiant

Via l'extension Thunder Client, ouvrez la route **PUT /students/id** et cliquez sur **Send**



Suppression d'un étudiant

Dans le fichier **students.service.ts**, créer une fonction qui supprime un étudiant du jeu de données selon l'id passé en paramètre :

```
const deleteStudent = (id: string) => {
  const index = students.findIndex(
    (student) => student.id.toString() === id
  );
  students.splice(index, 1);
  return students;
};
```

On exporte la fonction afin de pouvoir l'utiliser en dehors de notre fichier de service

```
export {
  getStudents,
  getStudent,
  createStudent,
  updateStudent,
  deleteStudent,
};
```

Suppression d'un étudiant

Dans le fichier **students.controller.ts**, créer une fonction qui fait appel au service de suppression d'un étudiant :

```
export const deleteStudent = (req: any, res: any) => {
  const { id } = req.params;
  const students = StudentsService.deleteStudent(id);
  return res.status(200).json(students);
};
```

ⓘ La fonction retournera l'étudiant retourné par le service au format json avec un statut 200

ⓘ L'id provient de l'URL, on le récupère depuis les paramètres de la requête : **req.params**

Suppression d'un étudiant

Le but est désormais de créer une route qui appellera le contrôleur **deleteStudent** lorsque l'utilisateur fera un appel HTTP de type DELETE sur l'URL **http://localhost/students/:id**

Importer la fonction **deleteStudents** du contrôleur :

```
import {
  getStudents,
  getStudent,
  createStudent,
  updateStudent,
  deleteStudent,
} from "../controllers/students.controller";
```

Créer la route :

```
router.delete("/:id", deleteStudent);
```

Suppression d'un étudiant

Via l'extension Thunder Client, ouvrez la route **DELETE /students/:id** et cliquez sur **Send**