TD2

Database with MongoDB

mongoose

https://www.jordansablon.fr/enseignement

https://github.com/jsablonEnseignement/td2_sources

Sommaire

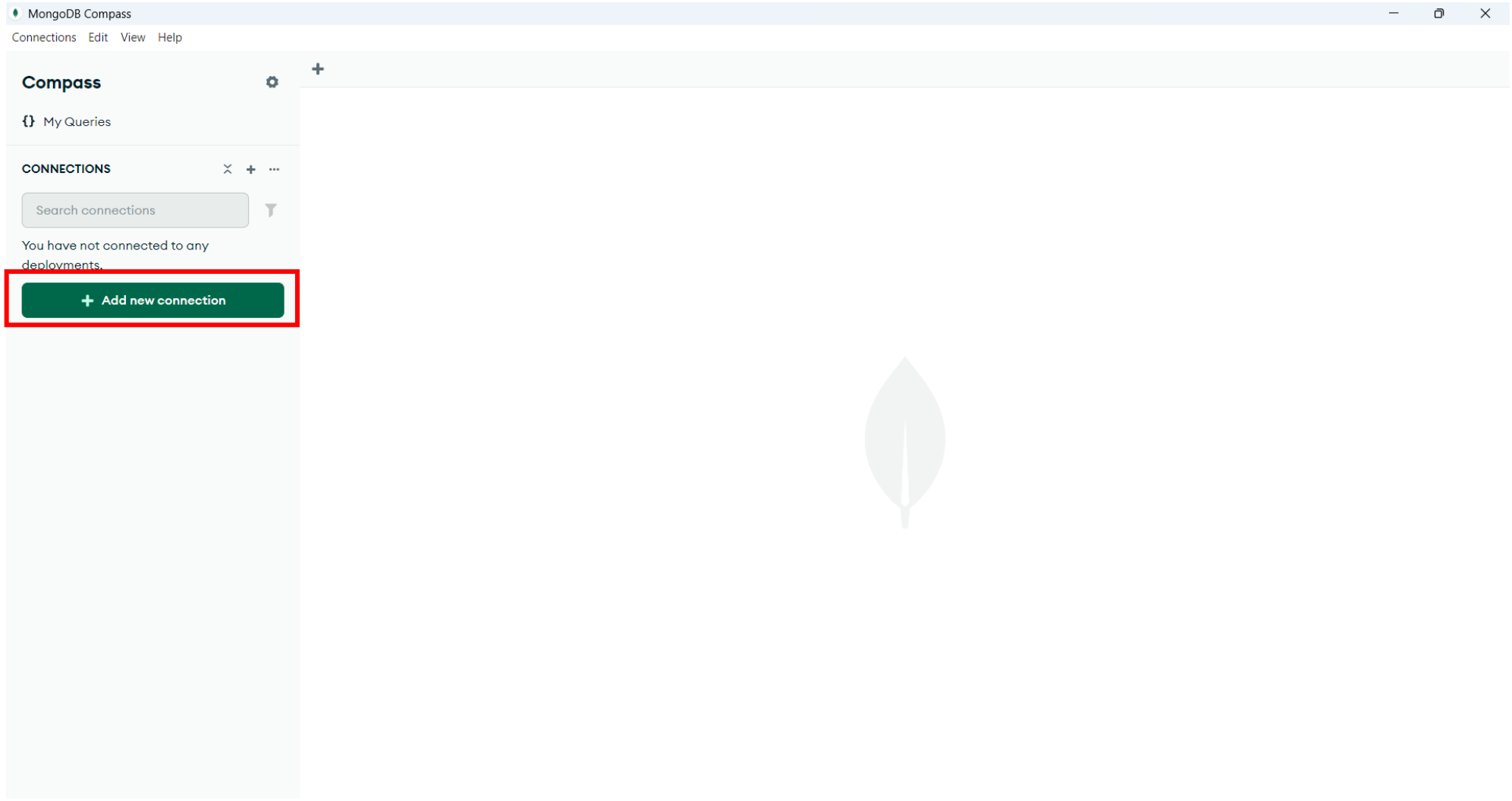
Hébergement d'une base de données en local

Si ce n'est pas déjà fait, téléchargez et installez MongoDB Community Server qui vous permettra de démarrer une base de données MongoDB en local.

Voir la partie dédiée dans le support de **TD0 – Notions de base & Configuration**

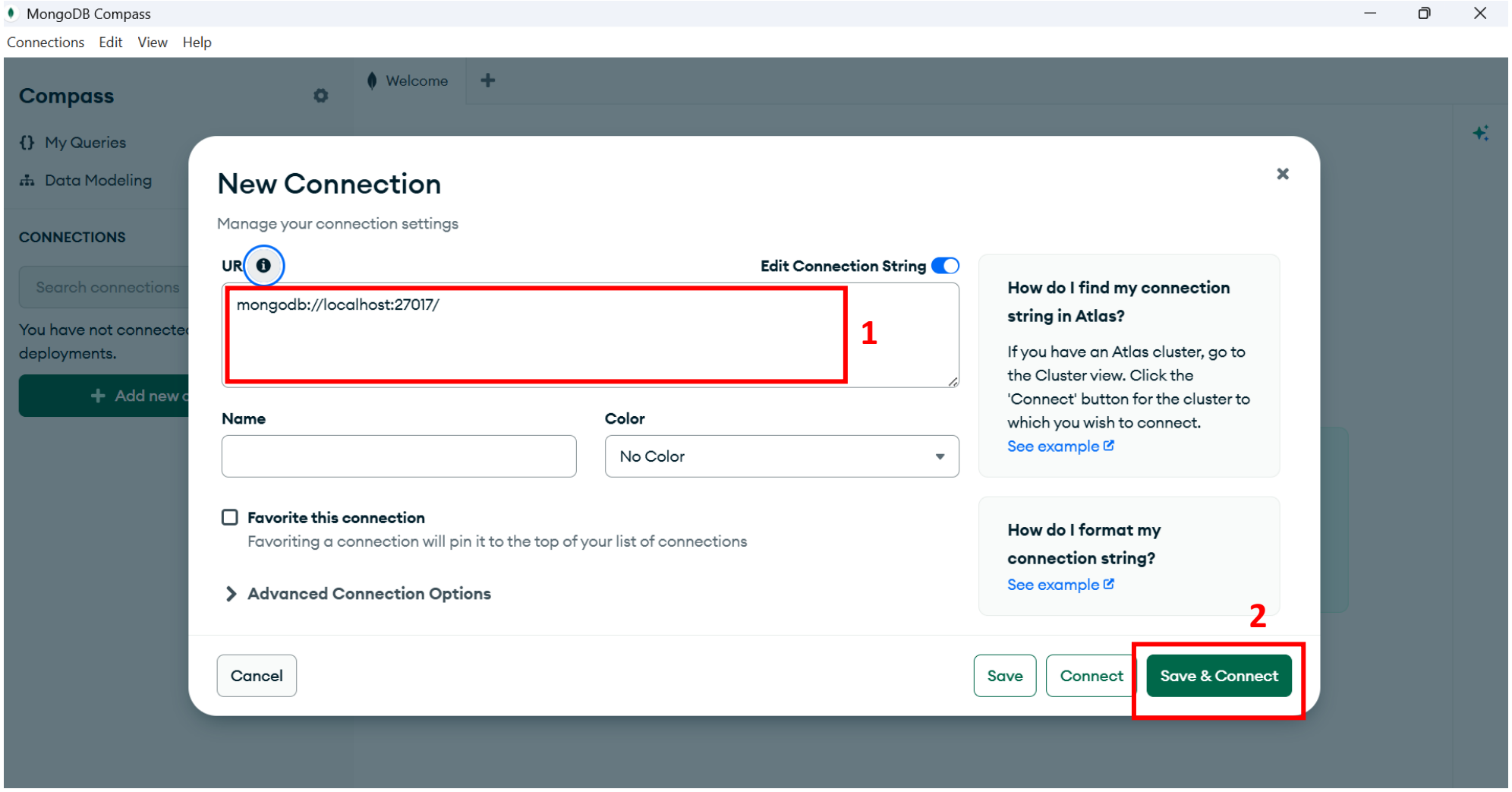
Insertion des données via MongoDB Compass

Démarrer le logiciel `MongoDBCompass` et ajouter une nouvelle connexion

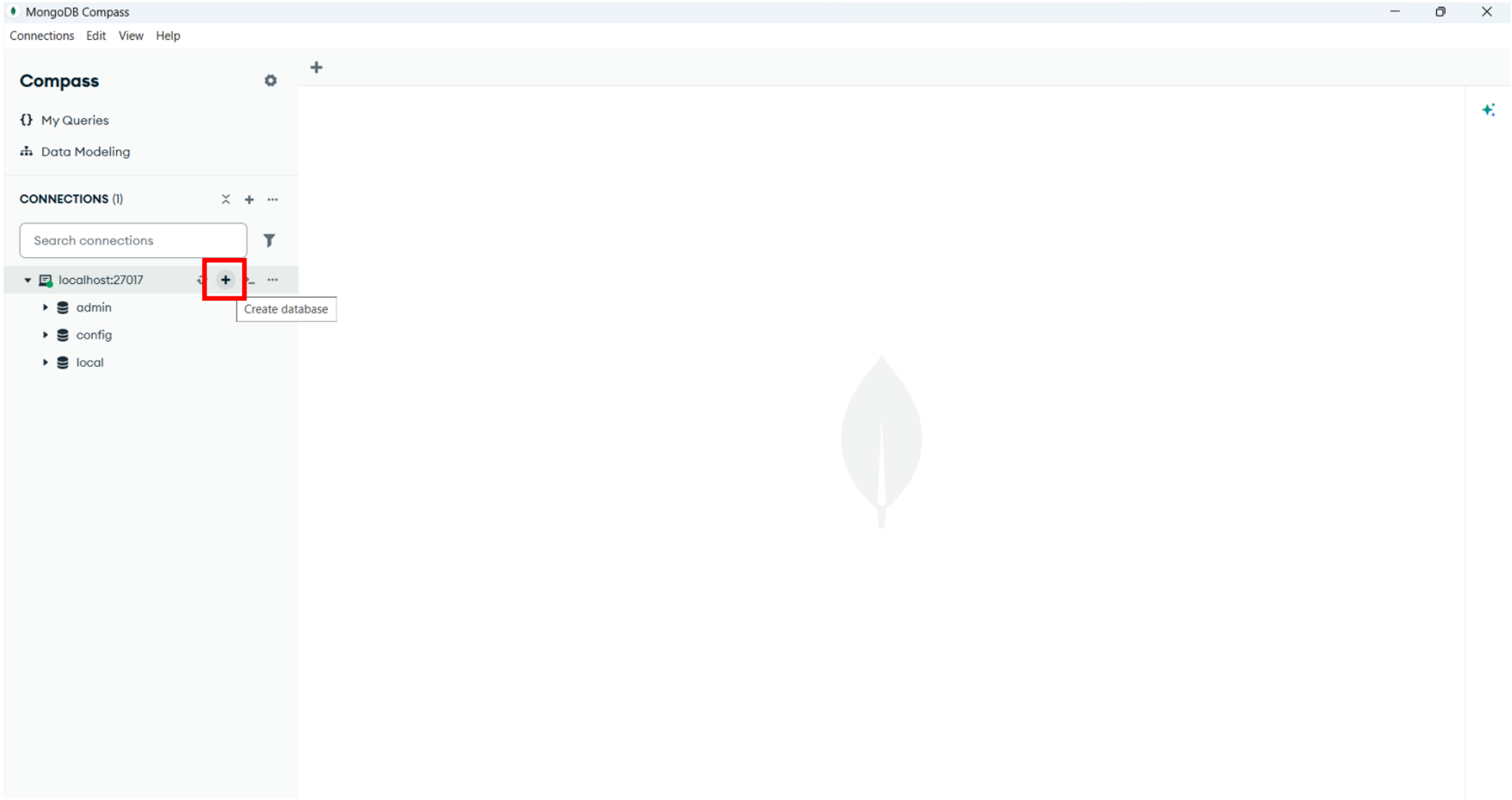


Insertion des données via MongoDB Compass

Laisser l'URI par défaut (`mongodb://localhost:27017`) et sauvegarder la connexion

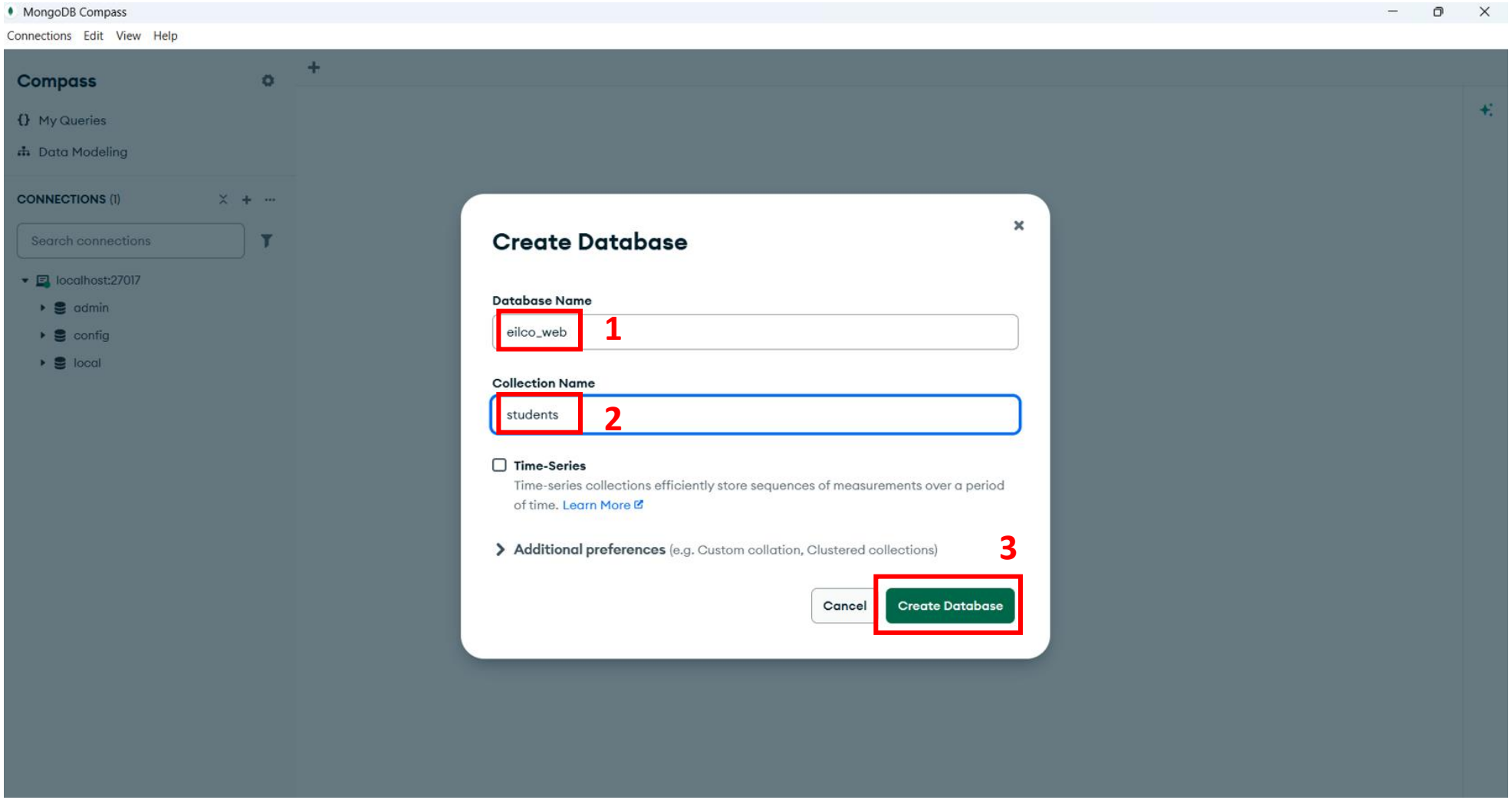


Ajouter une nouvelle base de données



Insertion des données via MongoDB Compass

Renseigner le nom de la base de données ainsi que celui de la collection puis valider



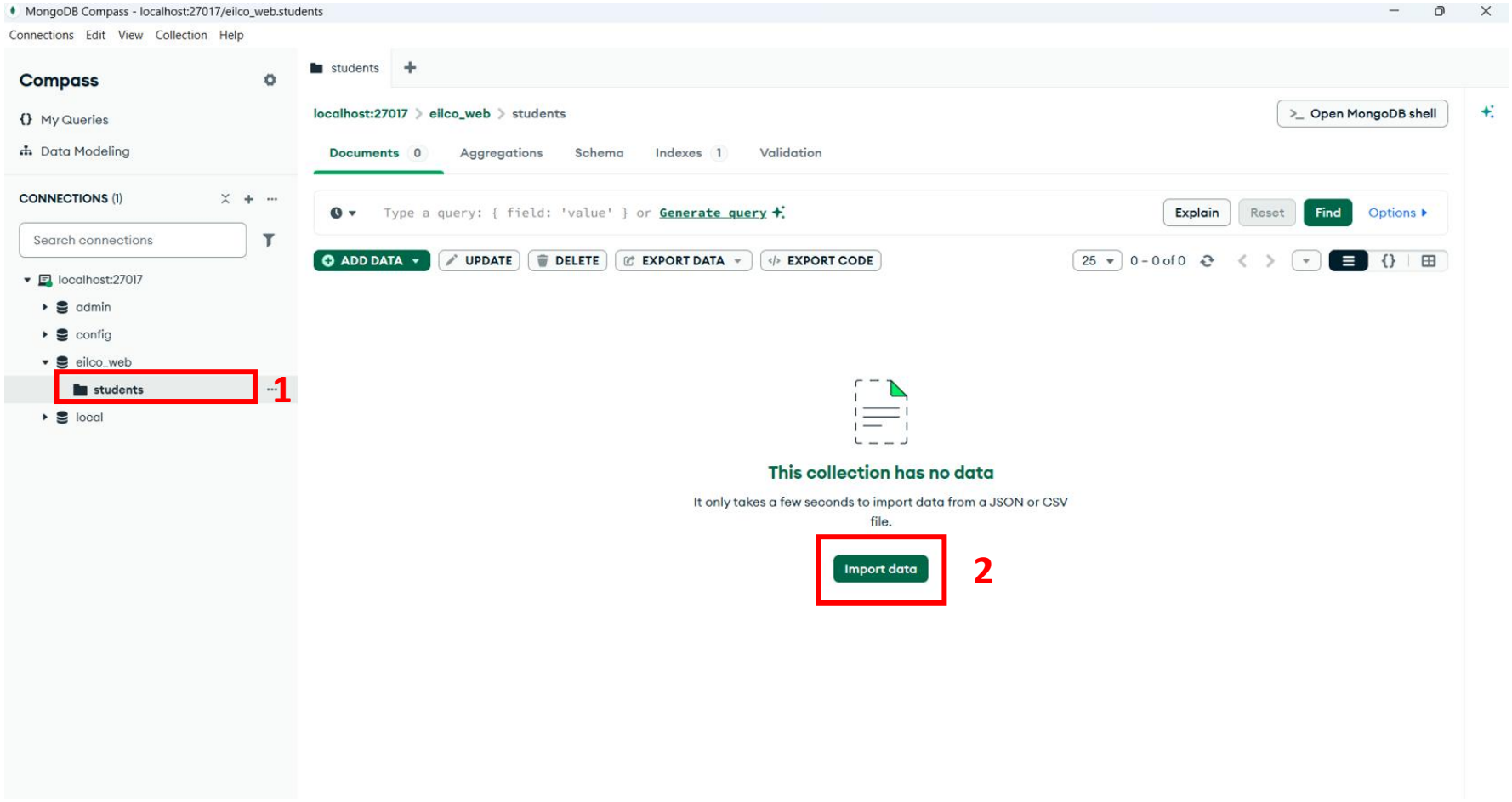
Insertion des données via MongoDB Compass

Cliquer sur le lien suivant pour accéder au fichier contenant le de données :
[eilco_web.students.json]

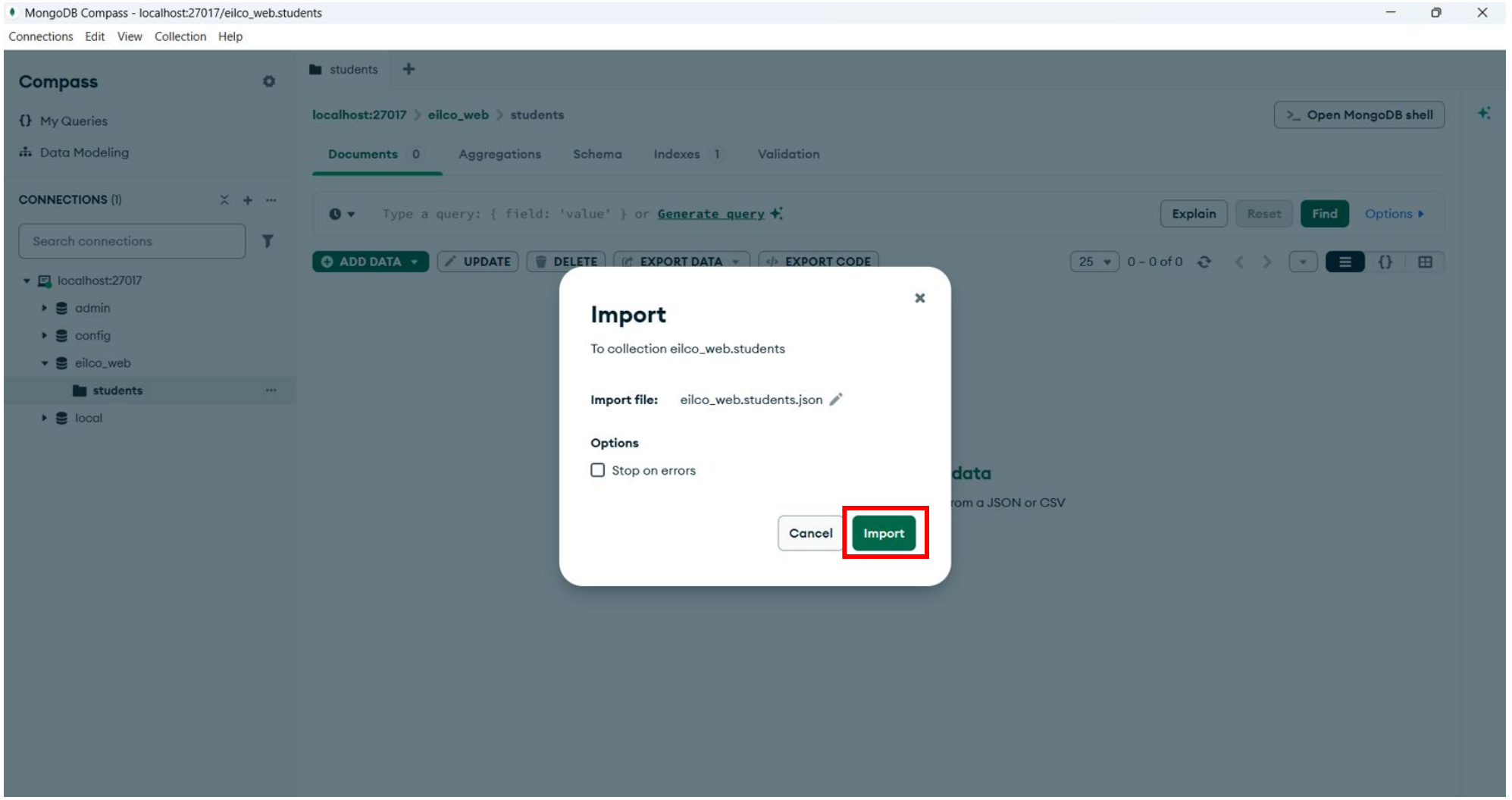
Faire un clic droit puis sélectionner « Enregistrer sous… » et enregistrer le fichier à l'endroit de votre choix

Insertion des données via MongoDB Compass

Sélectionner la collection `students` puis « Import Data »

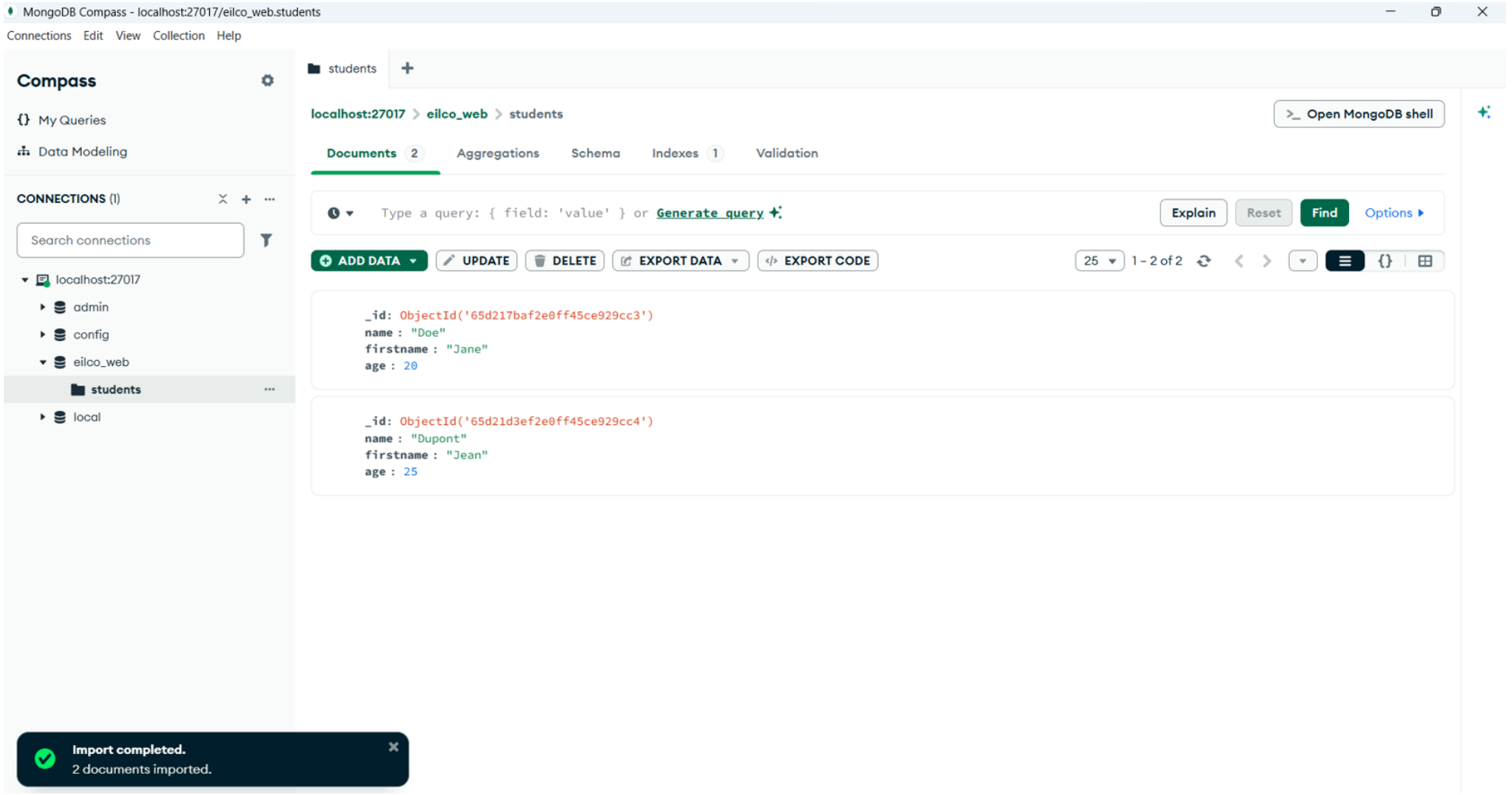


Sélectionner le fichier `eilco_web.students.json` puis cliquer sur « Import »



Insertion des données via MongoDB Compass

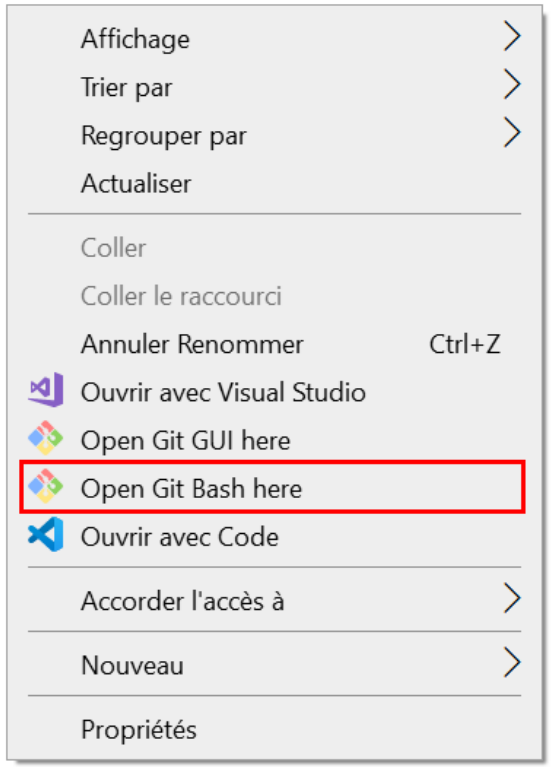
Les données ont bien été importées

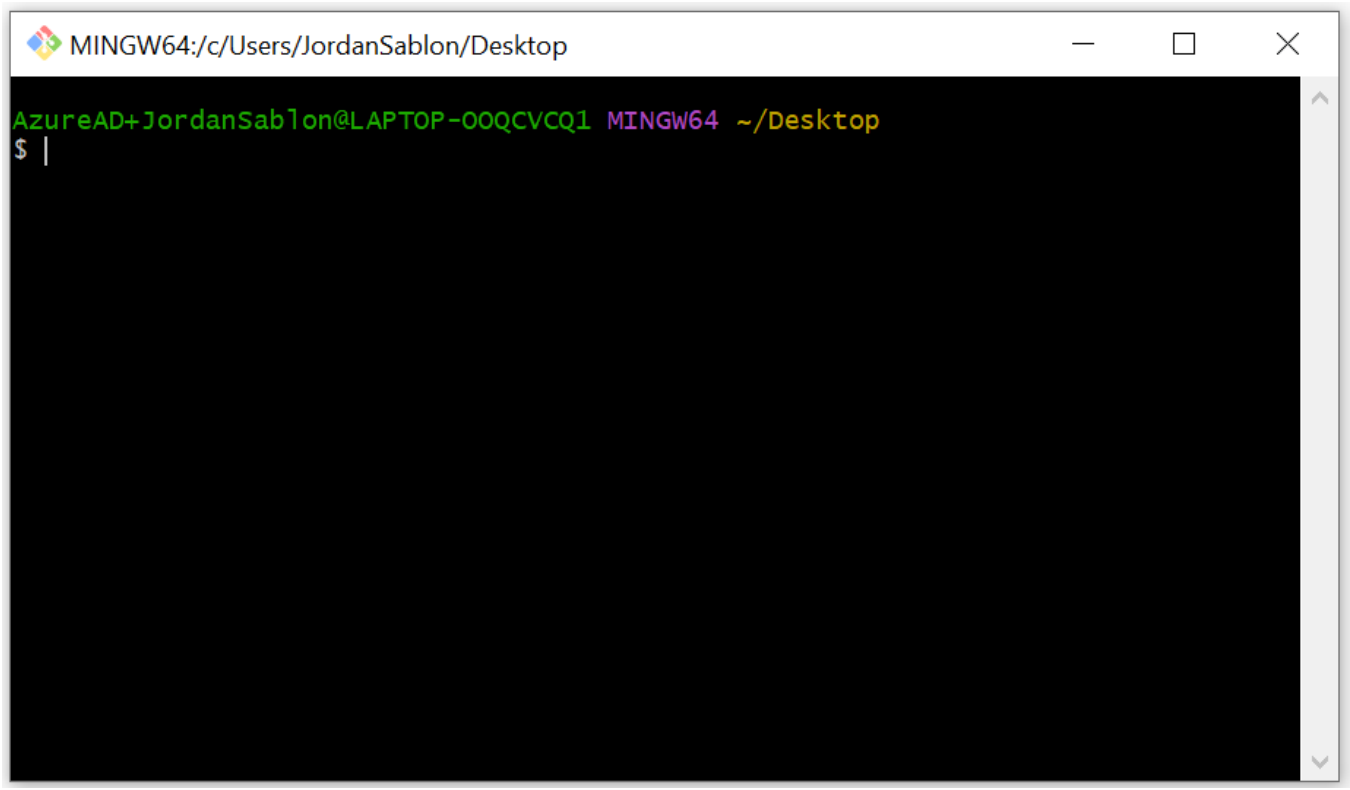


Récupération du code source du projet & installation des dépendances

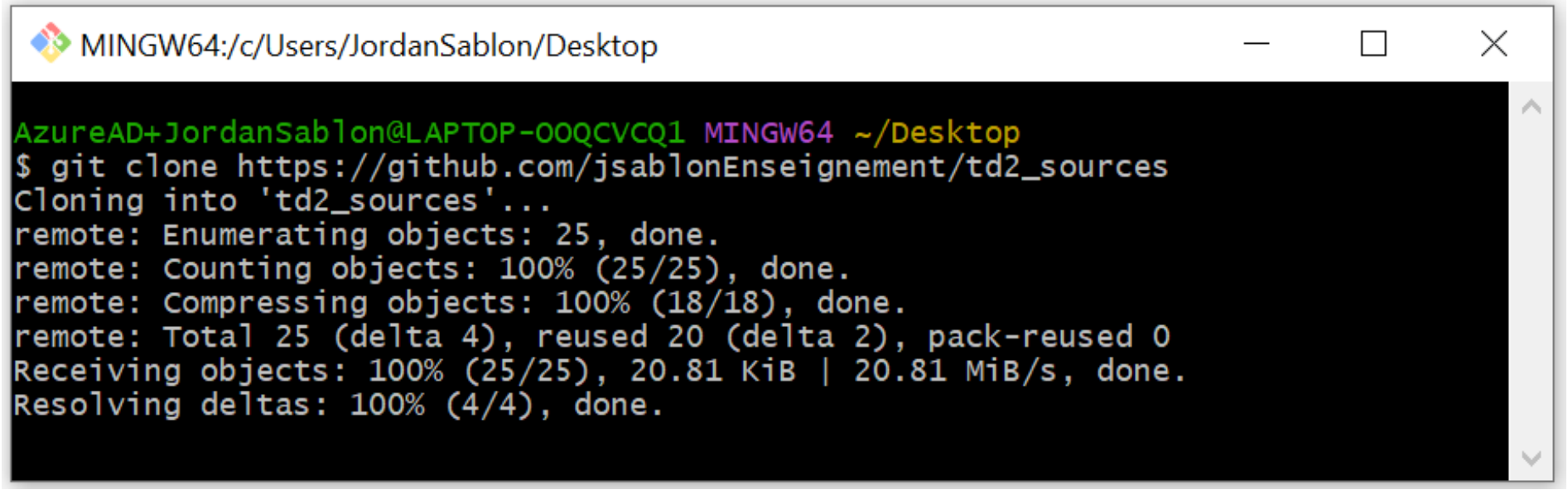
1. Se positionner à l'emplacement de votre choix (où se trouvera le futur dossier du projet)

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



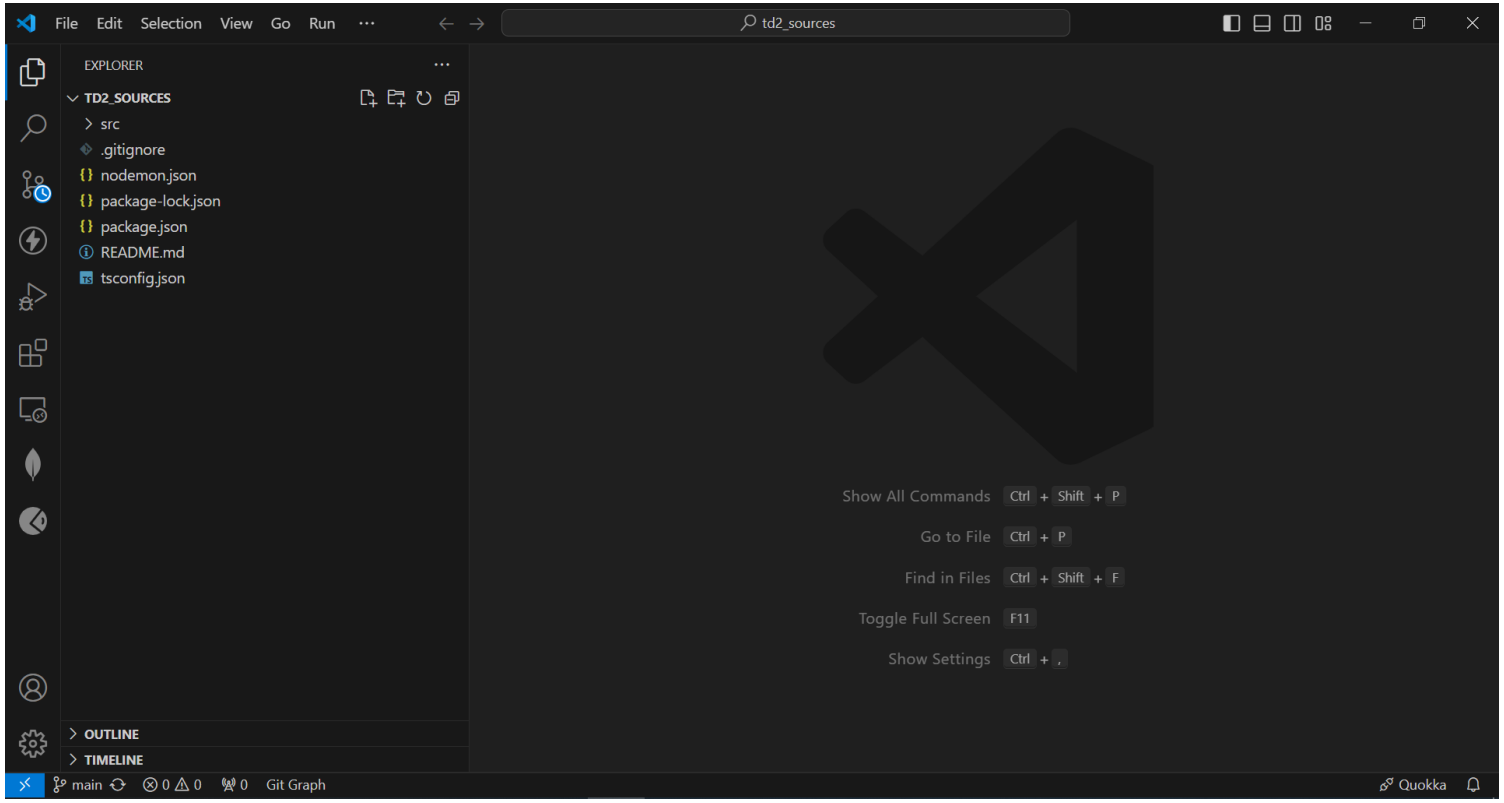


3. Cloner le répertoire Git : https://github.com/jsablonEnseignement/td2_sources



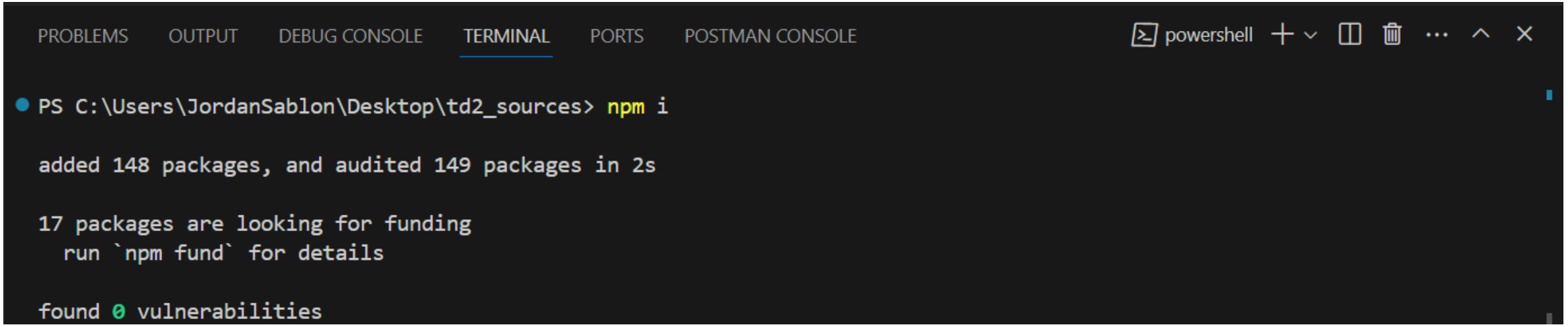
4. Se rendre dans le dossier précédemment cloné :

5. Ouvrir le projet dans VSCode :



Récupération du code source du projet & 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

Connecter l'application à la base de données

La connexion de notre application Node.js à la base de données MongoDB est réalisée via une chaine de connexion dans le fichier **`index.ts`** :

```
mongoose.connect(`mongodb://localhost:27017/eilco_web`);

const db = mongoose.connection;
```

Démarrage du serveur

```
PS C:\Users\JordanSablon\Documents\dev\eilco_web\backend> npm start

> backend@1.0.0 start
> npx tsc && nodemon

[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): src\**\*
[nodemon] watching extensions: ts
[nodemon] starting `ts-node-esm src/index.ts`
Server running at http://127.0.0.1:5000/
Connected successfully
```

Création du modèle

Créer un dossier **`models`** dans le dossier **`/src`**

Créer un fichier **`students.model.ts`** dans le dossier **`/models`**

Implémenter le schéma représentant la collection **`students`** donné à la diapo suivante

Création du modèle

```
import { model, Schema, Model, Types, Document } from "mongoose";

const StudentsSchema: Schema = new Schema(
  {
    name: {
      type: String,
      required: true,
    },
    firstname: {
      type: String,
      required: true,
    },
    age: {
      type: Number,
      default: 0,
    },
  },
  { versionKey: false }
);

export interface IStudent {
  _id: Types.ObjectId;
  name: String;
  firstname: String;
  age: Number;
}
export interface Student extends Omit<IStudent, "_id">, Document {}

export const StudentsModel = model<Student, Model<Student>>(
  "students",
  StudentSchema
);
```

Récupération de tous les étudiants

Dans le fichier **students.service.ts**, importer le modèle ainsi que l'interface précédemment créés :

```
import { Student, StudentsModel } from "../models/students.model";
```

Modifier la fonction *getStudents* :

```
const getStudents = async () => {
  return await StudentsModel.find();
};
```

- ⓘ On applique une fonction **find()** sur le modèle **StudentsModel** afin de récupérer tous les étudiants dans la collection **students** de notre base de données
- ⓘ On utilise l'instruction ***await*** afin de forcer l'attente du retour de la base de données pour récupérer les étudiants et les transmettre à notre contrôleur.
  On précise alors que notre fonction est asynchrone (***async***)

Récupération de tous les étudiants

Dans le fichier **students.controler.ts**, modifier la fonction *getStudents* :

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

ⓘ On utilise l'instruction ***await*** afin de forcer l'attente de l'exécution de notre service et ainsi pouvoir retourner son résultat dans la réponse.
On précise alors également que notre fonction est asynchrone (***async***)

Récupération d'un étudiant par son id

Dans le fichier **students.service.ts**, modifier la fonction *getStudent* :

```
const getStudent = async (id: string) => {
  return await StudentsModel.findById(id);
};
```

- ⓘ On applique la fonction **findById()**, avec comme paramètre l'id de l'étudiant, sur le modèle **StudentsModel** afin de récupérer l'étudiant correspondant dans la collection **students** de notre base de données
- ⓘ On utilise l'instruction ***await*** afin de forcer l'attente du retour de la base de données pour récupérer l'étudiant concerné et le transmettre à notre contrôleur.
  On précise alors que notre fonction est asynchrone (***async***)

Récupération d'un étudiant par son id

Dans le fichier **students.controller.ts**, modifier la fonction *getStudent* :

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

ⓘ On utilise l'instruction ***await*** afin de forcer l'attente de l'exécution de notre service et ainsi pouvoir retourner son résultat dans la réponse.
On précise alors également que notre fonction est asynchrone (***async***)

Création d'un étudiant

Dans le fichier **students.service.ts**, modifier la fonction *createStudent* :

```
const createStudent = async (studentToCreate: Student) => {
  const newStudent = new StudentsModel(studentToCreate);
  await newStudent.save();
  return getStudents();
};
```

ⓘ On crée une nouvelle instance du modèle **StudentsModel** avec les données de notre étudiant puis on applique la fonction *save()* afin d'insérer l'étudiant en base de données

ⓘ On utilise l'instruction *await* afin de forcer l'attente du retour de la base de données pour récupérer les étudiants et les transmettre à notre contrôleur.
On précise alors que notre fonction est asynchrone (*async*)

Création d'un étudiant

Dans le fichier **students.controller.ts**, modifier la fonction *createStudent* :

```
export const createStudent = async (req: any, res: any) => {
  const studentToCreate = req.body;
  const students = await StudentsService.createStudent(studentToCreate);
  return res.status(201).json(students);
};
```

ⓘ On utilise l'instruction ***await*** afin de forcer l'attente de l'exécution de notre service et ainsi pouvoir retourner son résultat dans la réponse.
On précise alors également que notre fonction est asynchrone (***async***)

Mise à jour d'un étudiant

Dans le fichier **students.service.ts**, modifier la fonction *updateStudent* :

```
const updateStudent = async (id: string, studentToUpdate: Student) => {
  await StudentsModel.findByIdAndUpdate(id, studentToUpdate);
  return getStudents();
};
```

- ⓘ On applique une fonction findOneByIdAndUpdate() sur le modèle **StudentsModel** afin de mettre à jour l'étudiant concerné dans la collection **students** de notre base de données

- ⓘ On utilise l'instruction *await* afin de forcer l'attente du retour de la base de données pour récupérer les étudiants et les transmettre à notre contrôleur.
  On précise alors que notre fonction est asynchrone (*async*)

Mise à jour d'un étudiant

Dans le fichier **students.controller.ts**, modifier la fonction *updateStudent* :

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

ⓘ On utilise l'instruction ***await*** afin de forcer l'attente de l'exécution de notre service et ainsi pouvoir retourner son résultat dans la réponse.
On précise alors également que notre fonction est asynchrone (***async***)

Suppression d'un étudiant

Dans le fichier **students.service.ts**, modifier la fonction *deleteStudent* :

```
const deleteStudent = async (id: string) => {
  await StudentsModel.findByIdAndDelete(id);
  return getStudents();
};
```

- ⓘ On applique une fonction **findByIdAndDelete()** sur le modèle **StudentsModel** afin de supprimer l'étudiant correspondant de la collection **students** de notre base de données
- ⓘ On utilise l'instruction *await* afin de forcer l'attente du retour de la base de données pour récupérer les étudiants et les transmettre à notre contrôleur.
  On précise alors que notre fonction est asynchrone (*async*)

Suppression d'un étudiant

Dans le fichier **students.controller.ts**, modifier la fonction *deleteStudent* :

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

ⓘ On utilise l'instruction ***await*** afin de forcer l'attente de l'exécution de notre service et ainsi pouvoir retourner son résultat dans la réponse.
On précise alors également que notre fonction est asynchrone (***async***)