

Database with MongoDB

TD2



mongoose



<https://www.jordansablons.fr/enseignement>



https://github.com/jsablonsEnseignement/td2_sources

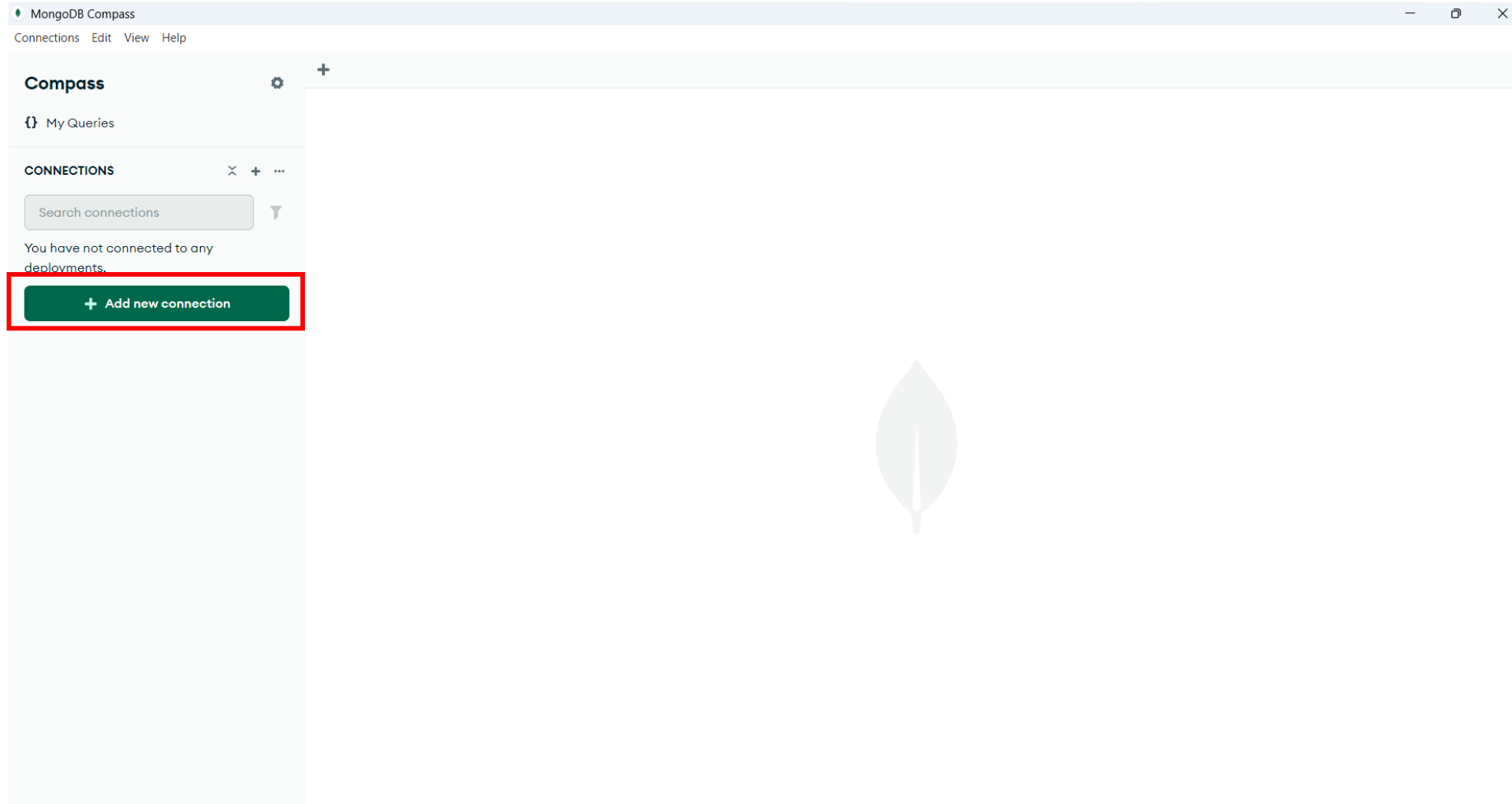
- Hébergement d'une base de données MongoDB en local
- Initialisation de la collection
- Insertion des données via MongoDB Compass
- Récupération du code source du projet & installation des dépendances
- Connecter l'application à la base de données
- Création du modèle
- Mise à jour des services et contrôleurs

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.

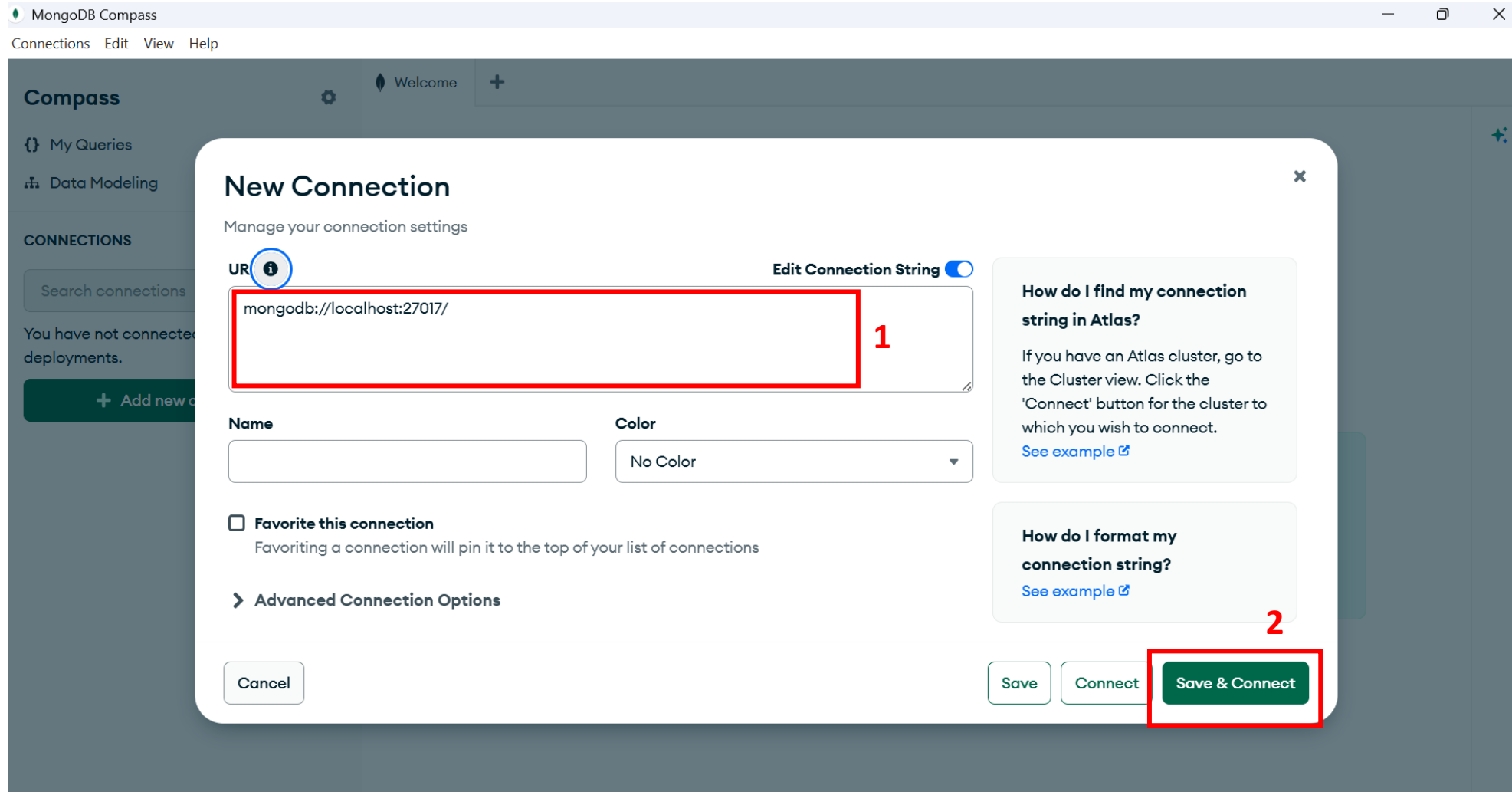
Voir la partie dédiée dans le support de **TD0 – Notions de base & Configuration**

Démarrer le logiciel MongoDB Compass et ajouter une nouvelle connexion

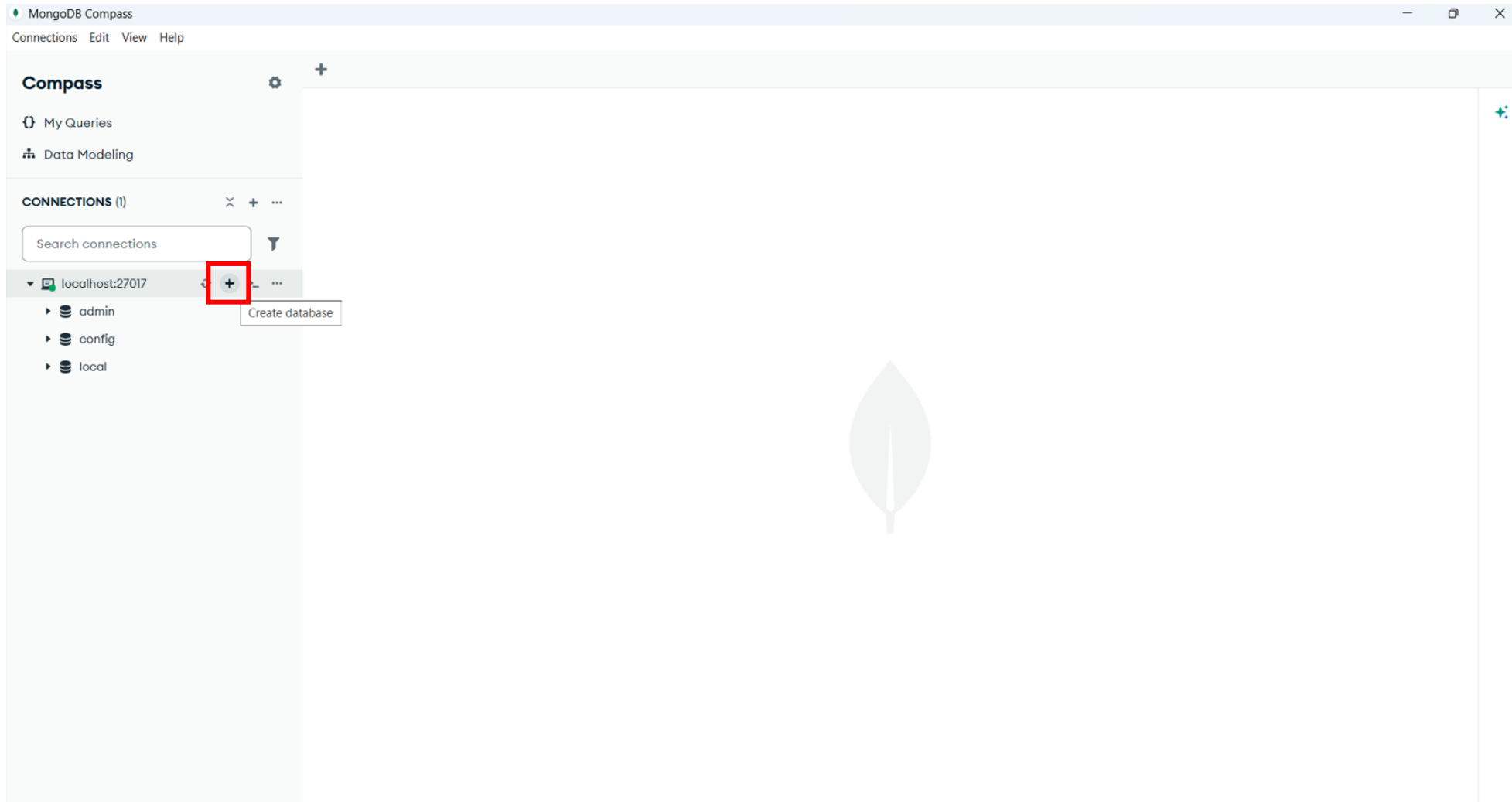


Insertion des données via MongoDB Compass

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

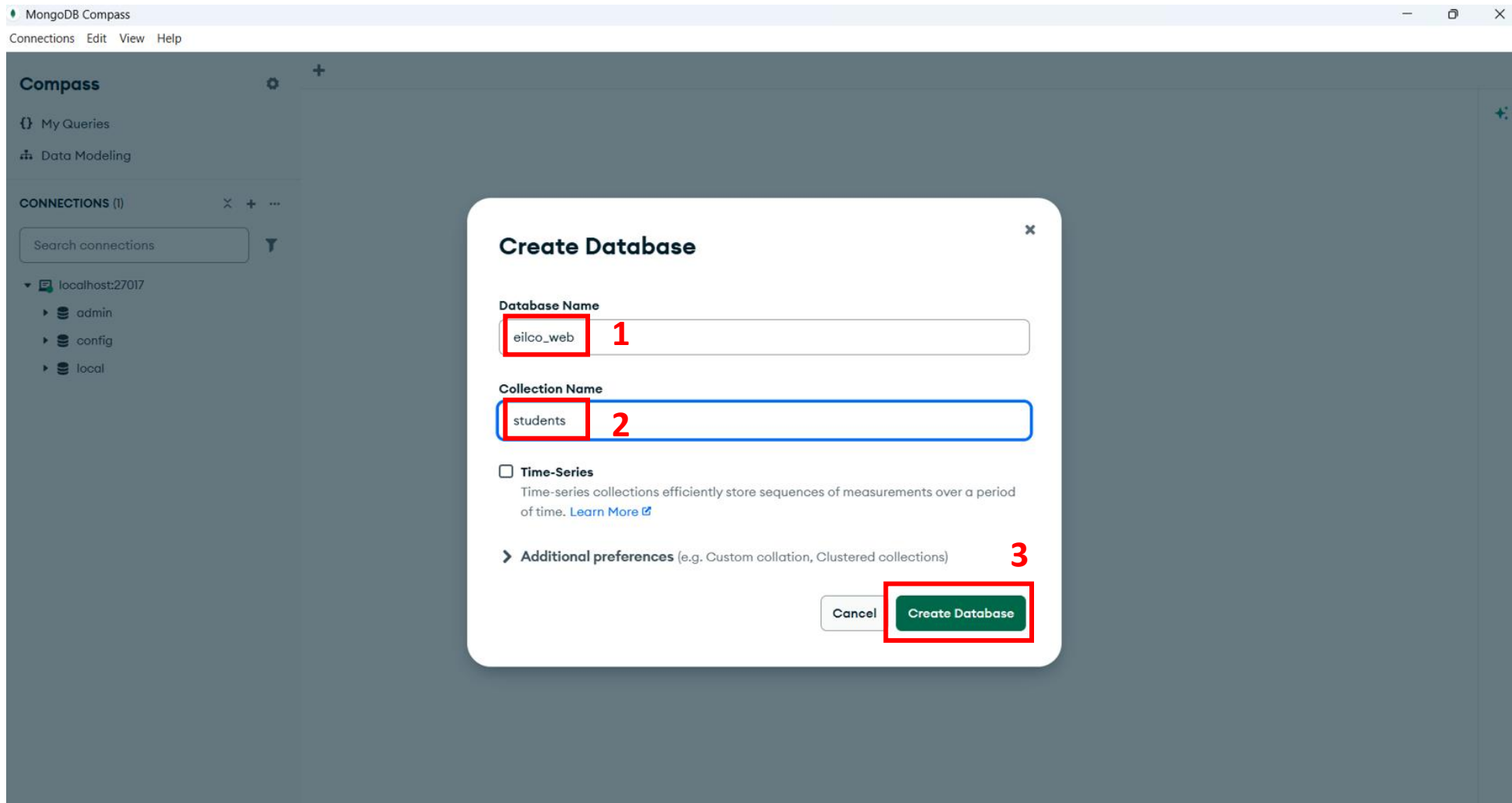


Ajouter une nouvelle base de données



Insertion des données via MongoDB Compass

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



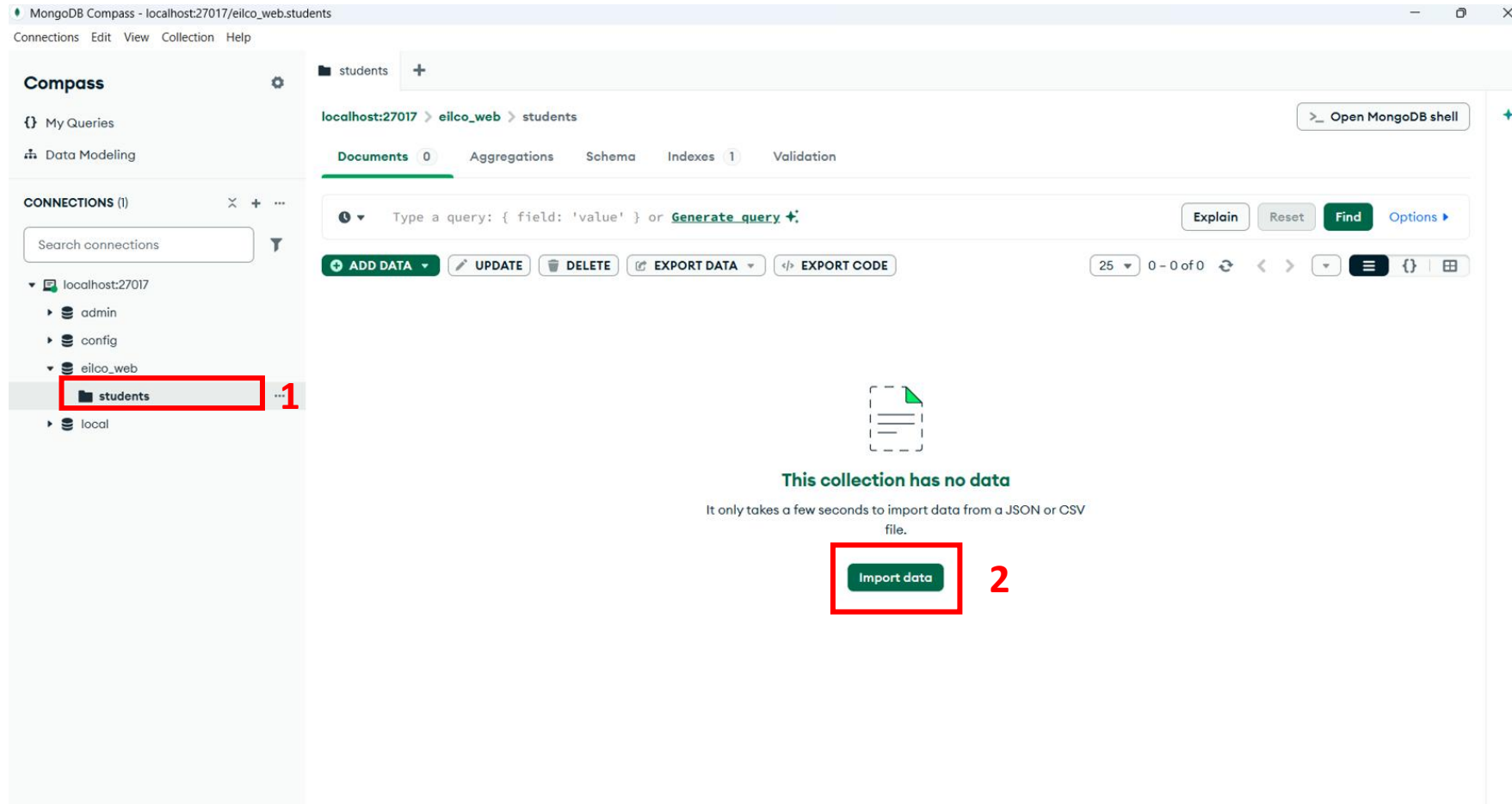
Cliquer sur le lien suivant pour accéder au fichier contenant le de données :
[eilco_web.students.json](#)

```
Impression élégante   
  
[  
  {  
    "_id": {  
      "$oid": "65d217baf2e0ff45ce929cc3"  
    },  
    "name": "Doe",  
    "firstname": "Jane",  
    "age": 20  
  },  
  {  
    "_id": {  
      "$oid": "65d21d3ef2e0ff45ce929cc4"  
    },  
    "name": "Dupont",  
    "firstname": "Jean",  
    "age": 25  
  }  
]
```

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

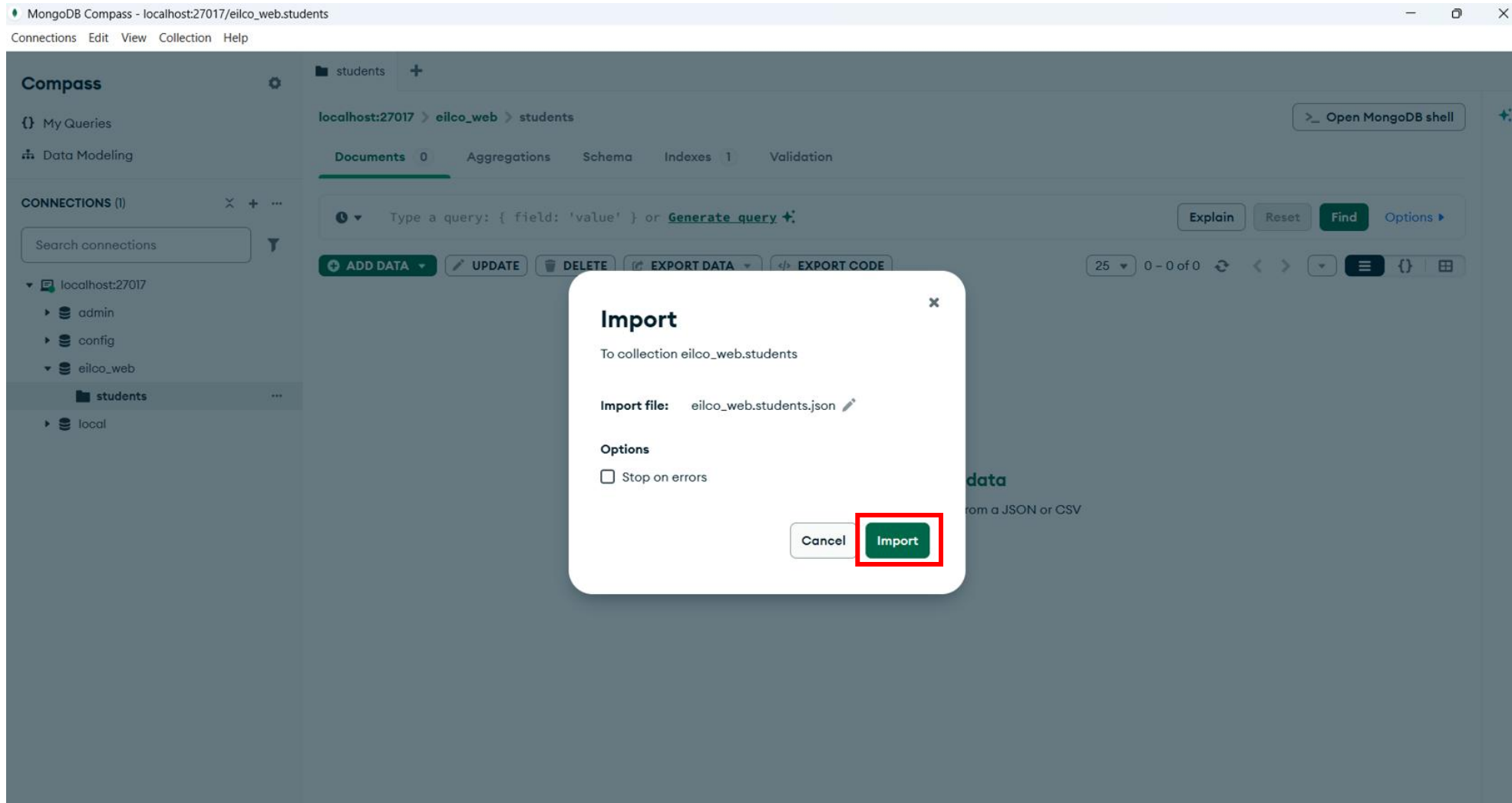
Insertion des données via MongoDB Compass

Sélectionner la collection students puis « Import Data »

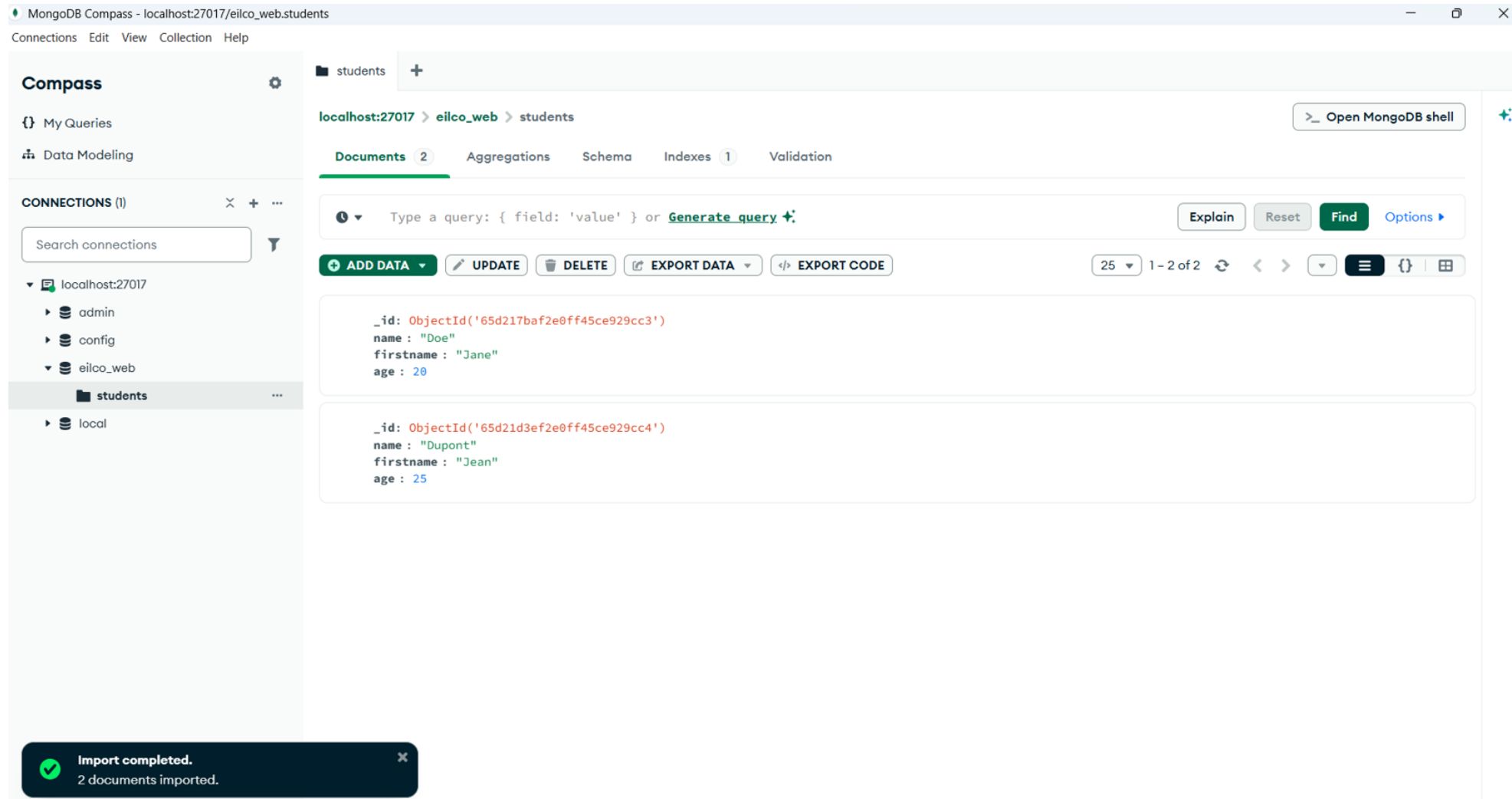


Insertion des données via MongoDB Compass

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



Les données ont bien été importées



The screenshot shows the MongoDB Compass interface. The left sidebar displays the 'connections' list with 'localhost:27017' expanded to show the 'students' collection. The main area shows the 'students' collection with two documents listed. A notification at the bottom left states 'Import completed. 2 documents imported.'

students +

localhost:27017 > eilco_web > students Open MongoDB shell

Documents 2 Aggregations Schema Indexes 1 Validation

Type a query: { field: 'value' } or [Generate query](#) Explain Reset Find Options

ADD DATA UPDATE DELETE EXPORT DATA EXPORT CODE 25 1 - 2 of 2

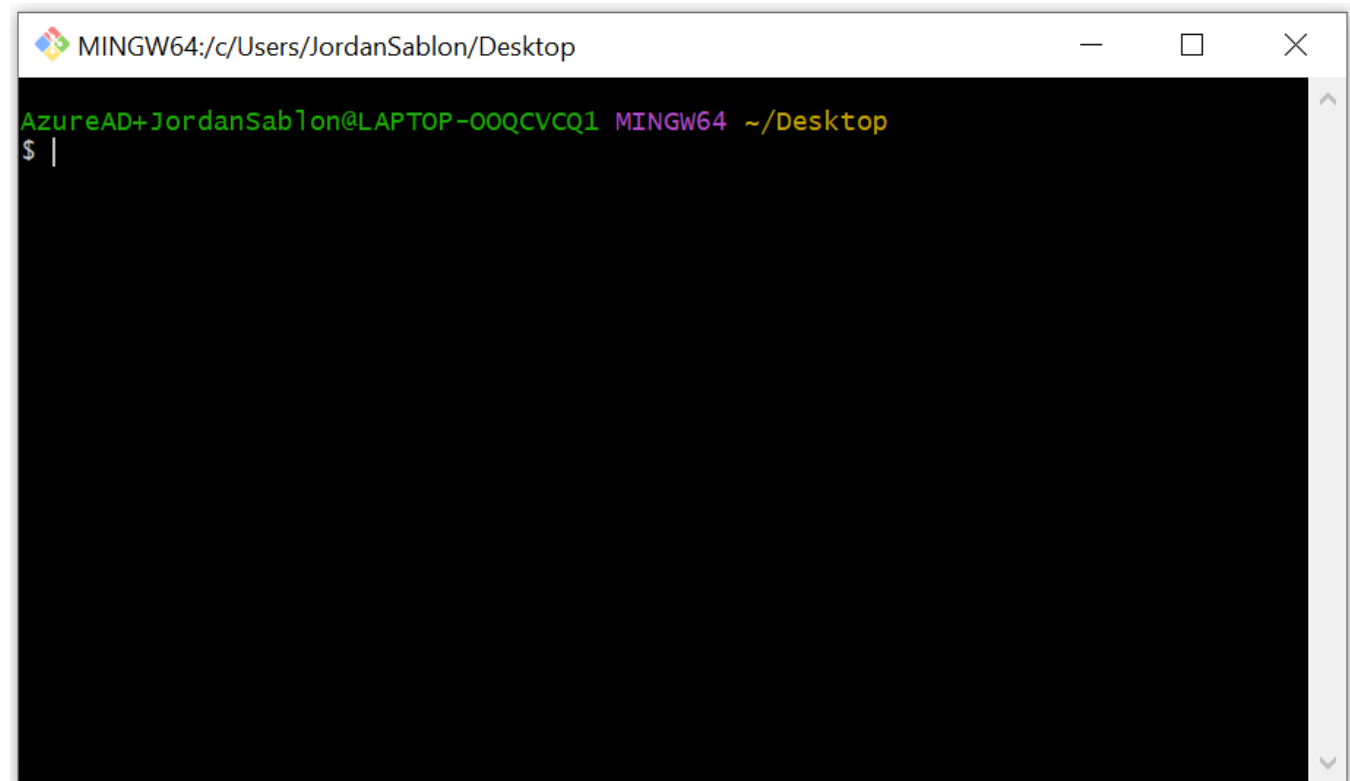
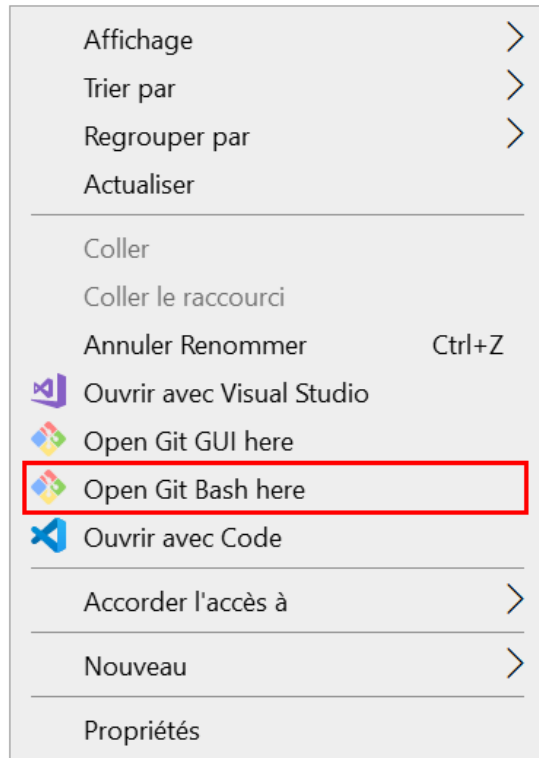
```
{ "_id": ObjectId('65d217baf2e0ff45ce929cc3'),  
  "name": "Doe",  
  "firstname": "Jane",  
  "age": 20 }
```

```
{ "_id": ObjectId('65d21d3ef2e0ff45ce929cc4'),  
  "name": "Dupont",  
  "firstname": "Jean",  
  "age": 25 }
```

Import completed.
2 documents imported.

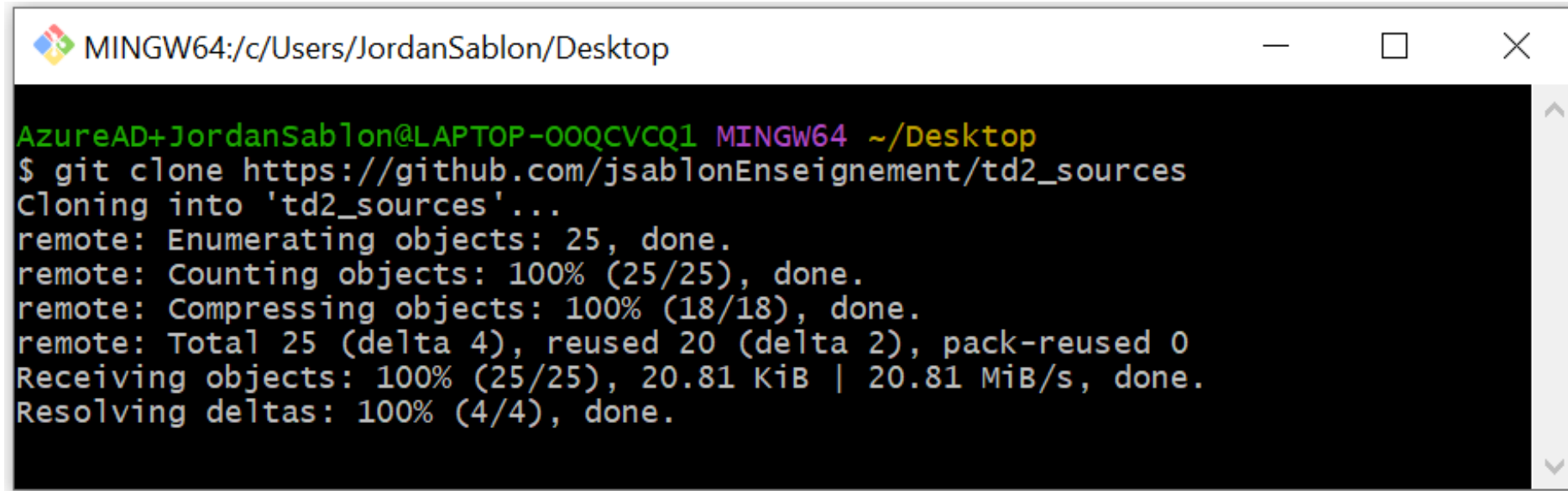
Récupération du code source du projet & installation des dépendances

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 :



Récupération du code source du projet & installation des dépendances

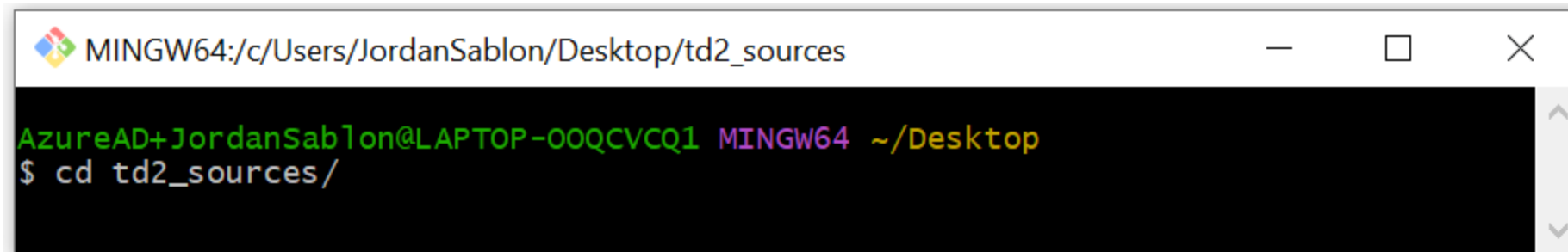
3. Cloner le répertoire Git : https://github.com/jsablonEnseignement/td2_sources



```
MINGW64:/c/Users/JordanSablon/Desktop

AzureAD+JordanSablon@LAPTOP-00QCVCQ1 MINGW64 ~/Desktop
$ git clone https://github.com/jsablonEnseignement/td2_sources
Cloning into 'td2_sources'...
remote: Enumerating objects: 25, done.
remote: Counting objects: 100% (25/25), done.
remote: Compressing objects: 100% (18/18), done.
remote: Total 25 (delta 4), reused 20 (delta 2), pack-reused 0
Receiving objects: 100% (25/25), 20.81 KiB | 20.81 MiB/s, done.
Resolving deltas: 100% (4/4), done.
```

4. Se rendre dans le dossier précédemment cloné :

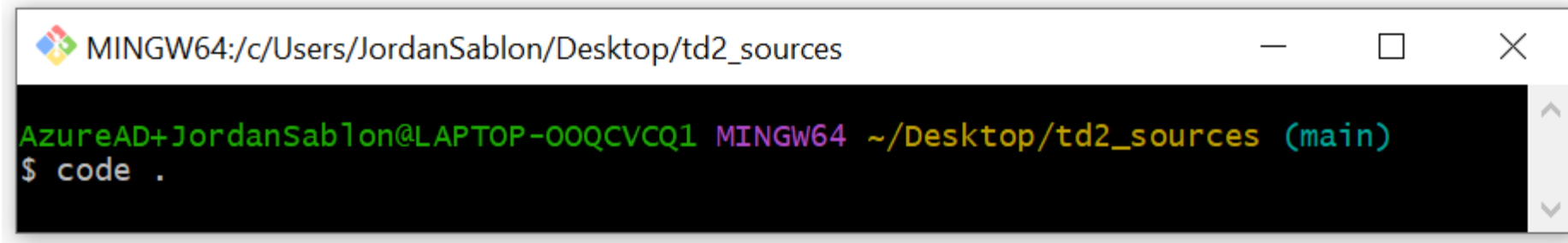


```
MINGW64:/c/Users/JordanSablon/Desktop/td2_sources

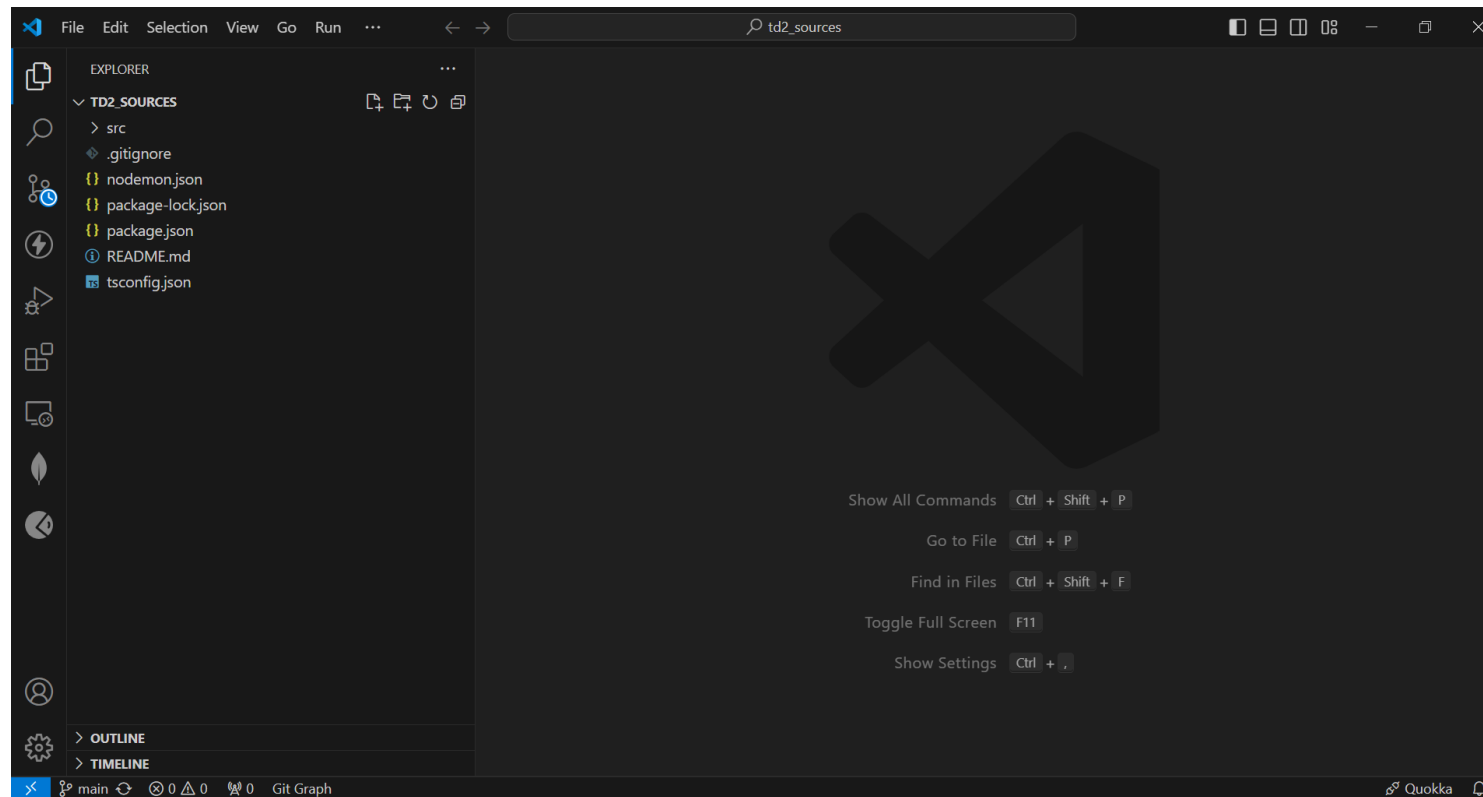
AzureAD+JordanSablon@LAPTOP-00QCVCQ1 MINGW64 ~/Desktop
$ cd td2_sources/
```

Récupération du code source du projet & installation des dépendances

5. Ouvrir le projet dans VSCode :

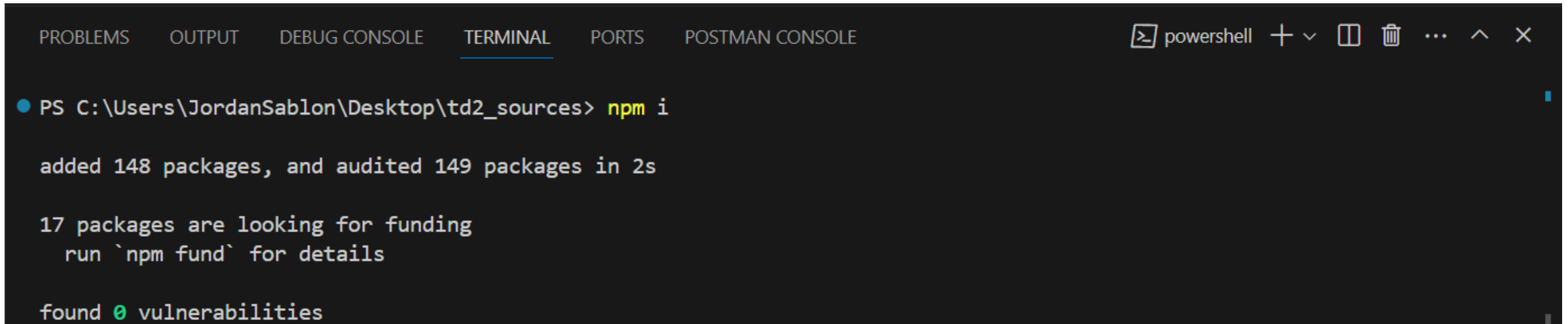


```
MINGW64:/c/Users/JordanSablon/Desktop/td2_sources  
AzureAD+JordanSablon@LAPTOP-00QCVCQ1 MINGW64 ~/Desktop/td2_sources (main)  
$ code .
```



Récupération du code source du projet & installation des dépendances

Depuis le terminal de VSCode, exécuter la commande suivante :



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE powershell + v [ ] [ ] ... ^ X  
● PS C:\Users\JordanSablon\Desktop\td2_sources> npm i  
  
added 148 packages, and audited 149 packages in 2s  
  
17 packages are looking for funding  
  run `npm fund` for details  
  
found 0 vulnerabilities
```

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 chaîne 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
```

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();  
};
```

- ① On applique une fonction `find()` sur le modèle **StudentsModel** afin de récupérer tous les étudiants dans la collection **students** de notre base de données
- ① 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.controller.ts**, modifier la fonction *getStudents* :

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

- ① 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);  
};
```

- ① 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
- ① 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);  
};
```

- ① 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();  
};
```

- ① 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
- ① 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);  
};
```

- ① 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();  
};
```

- ① 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
- ① 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);  
};
```

- ① 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();  
};
```

- ① On applique une fonction `findByIdAndDelete()` sur le modèle **StudentsModel** afin de supprimer l'étudiant correspondant de la collection **students** de notre base de données
- ① 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);};
```

- ① 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*)