

Database with MongoDB

TD2



mongoose



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



https://github.com/jsablonsEnseignement/td2_sources

- Déploiement d'une base de données MongoDB via Atlas
- Initialisation de la collection
- Insertion des données via MongoDB Compass
- Installation de mongoose
- 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
- Test via Thunder Client
- Utilisation des variables d'environnement

Créez un compte sur [Atlas](#)

MongoDB Atlas

- ✓ **Work with your data as code**
Documents in MongoDB map directly to objects in your programming language. Modify your schema as your apps grow over time.
- ✓ **Focus on building, not managing**
Let MongoDB Atlas take care of the infrastructure operations you need for performance at scale, from always-on security to point-in-time recovery.
- ✓ **Simplify your data dependencies**
Leverage application data for full-text search, real-time analytics, rich visualizations and more with a single API and minimal data movement.

Sign up

See what Atlas is capable of for free

First Name*

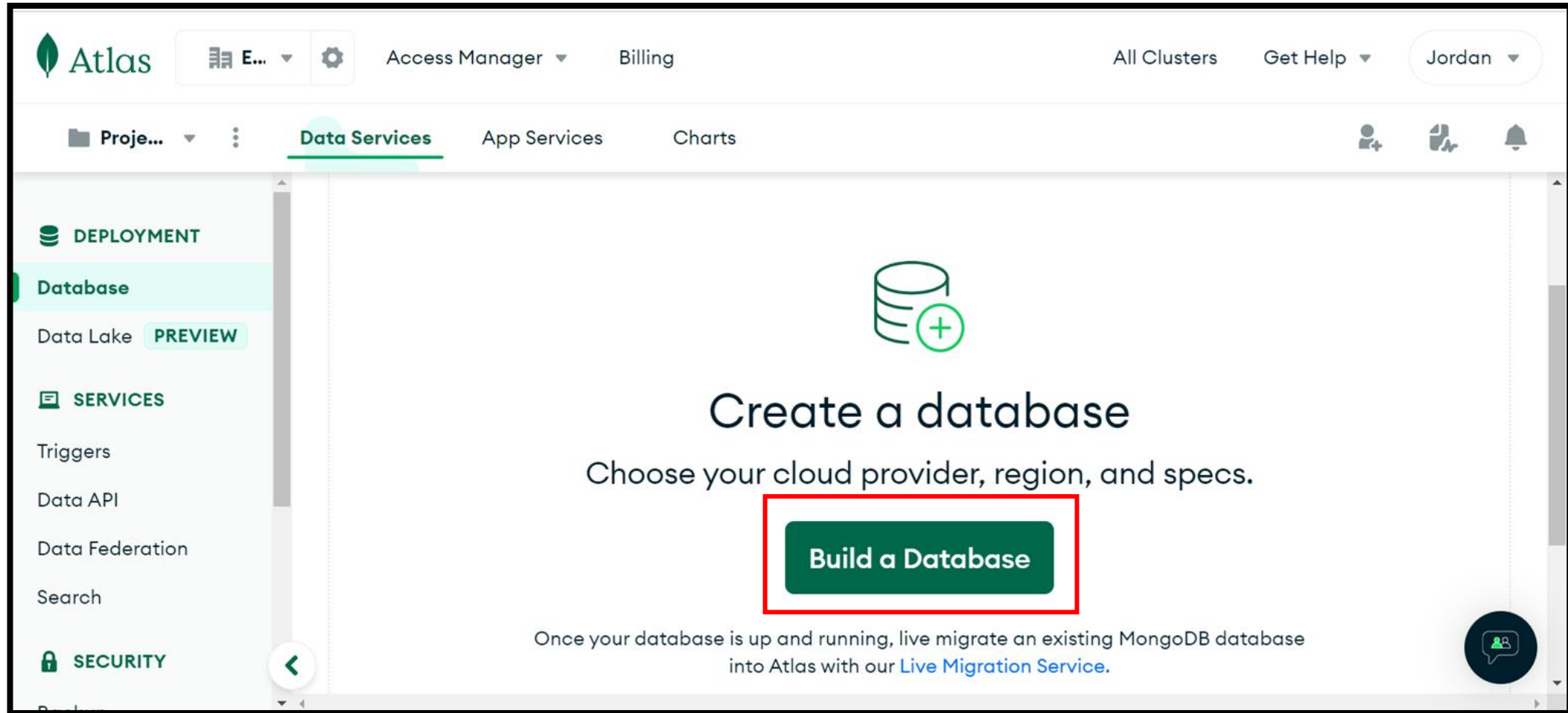
Last Name*

Company

Email*

Déploiement d'une base de données MongoDB via Atlas

Sélectionnez « Build a Database » pour créer votre cluster de base de données



The screenshot shows the MongoDB Atlas interface. At the top, there's a navigation bar with the Atlas logo, a menu icon, and links for 'Access Manager', 'Billing', 'All Clusters', 'Get Help', and a user profile 'Jordan'. Below this is a secondary navigation bar with 'Proje...', 'Data Services' (highlighted), 'App Services', and 'Charts'. A left sidebar contains categories: 'DEPLOYMENT' (with 'Database' selected), 'Data Lake' (marked 'PREVIEW'), 'SERVICES' (with 'Triggers', 'Data API', 'Data Federation', and 'Search'), and 'SECURITY'. The main content area features a large green icon of a database cylinder with a plus sign, followed by the heading 'Create a database' and the subtext 'Choose your cloud provider, region, and specs.'. A prominent green button labeled 'Build a Database' is centered and highlighted with a red rectangular border. At the bottom, there's a note: 'Once your database is up and running, live migrate an existing MongoDB database into Atlas with our [Live Migration Service](#).' A chat icon is visible in the bottom right corner.

Déploiement d'une base de données MongoDB via Atlas

Sélectionnez le cluster « M0 » qui est gratuit

edit these configuration options once the cluster is created.

\$0.08/hour

Production applications with dedicated workload requirements.

TYPE	RAM	vCPU
	2 GB	2 vCPUs

SERVERLESS **\$0.10/1M reads**

For application development and testing, or workloads with variable traffic.

STORAGE	RAM	vCPU
Up to 1 TB	Auto-scale	Auto-scale

M0 **FREE**

For learning and exploring MongoDB in a cloud environment.

STORAGE	RAM	vCPU
512 MB	Shared	Shared


FREE

Create

[Access Advanced Configuration](#)

Free forever! Your M0 cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.

[I'll deploy my database later](#)



1

2

Déploiement d'une base de données MongoDB via Atlas

Vous êtes ensuite invités à créer un utilisateur pour vous connecter à votre base de données

The screenshot shows the MongoDB Atlas interface for 'Project 0'. The left sidebar has 'SECURITY' highlighted (1) and 'Quickstart' selected (2). The main content area is titled 'Security Quickstart' and asks 'How would you like to authenticate your connection?'. Two options are shown: 'Username and Password' (3) and 'Certificate'. The 'Username and Password' option is selected, and its form fields are highlighted (3). The 'Create User' button is highlighted (4). The form fields contain 'jsabloneilco' for the username and a masked password. There are buttons for 'Autogenerate Secure Password' and 'Copy'.

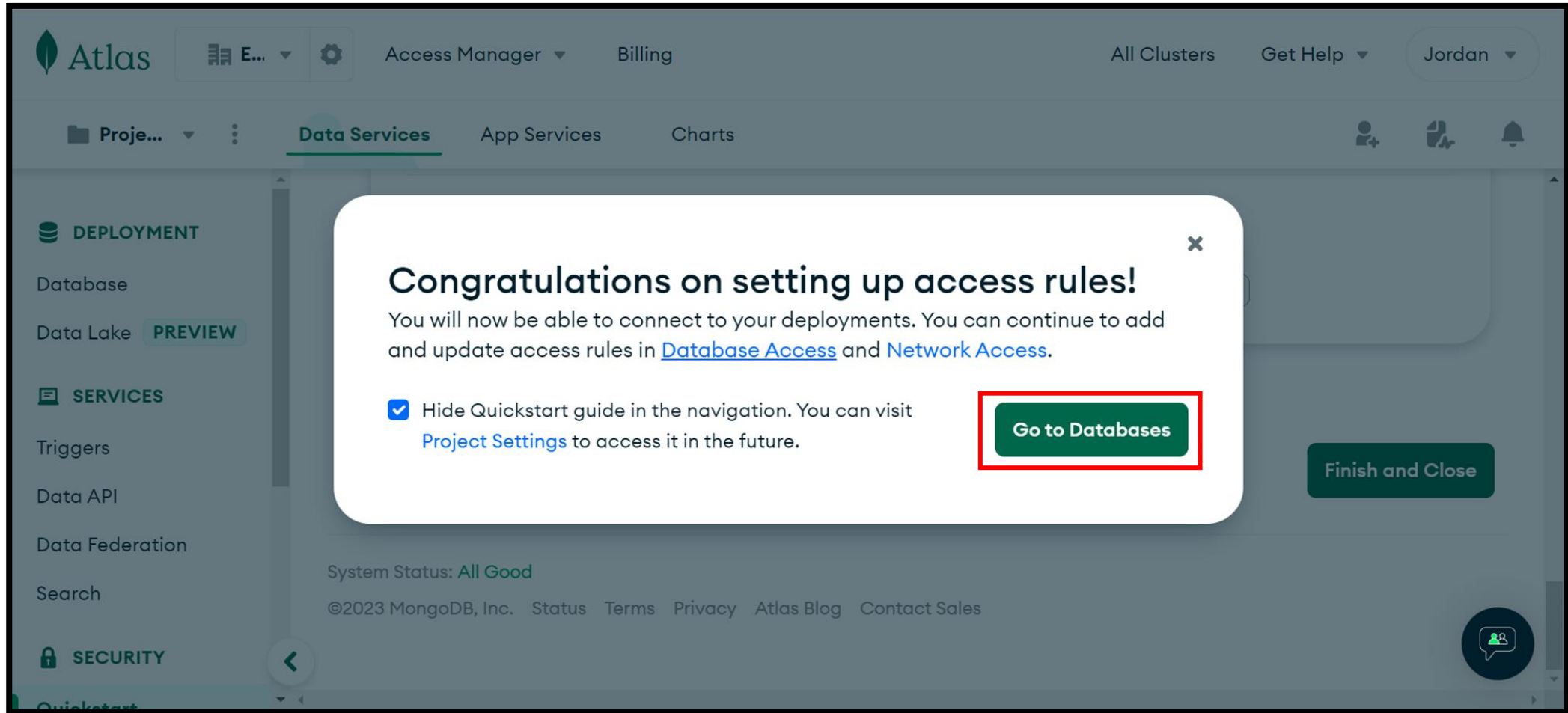
Déploiement d'une base de données MongoDB via Atlas

Si demandé, sélectionnez ensuite « My Local Environment »

The screenshot displays the MongoDB Atlas web interface. At the top, the 'Atlas' logo is on the left, and navigation links for 'Access Manager', 'Billing', 'All Clusters', 'Get Help', and a user profile 'Jordan' are on the right. Below this is a navigation bar with 'Data Services' selected. A sidebar on the left contains sections for 'DEPLOYMENT' (Database, Data Lake), 'SERVICES' (Triggers, Data API, Data Federation, Search), and 'SECURITY'. The main content area is titled 'Enable access for any network(s) that need to read and write data to your cluster.' It features two cards: 'My Local Environment' (highlighted with a red box) and 'Cloud Environment' (marked 'ADVANCED'). Below these cards is a blue information banner stating: 'We added your current IP address. You can connect to your cluster locally from this device.' The bottom section is titled 'Add entries to your IP Access List' and includes a note: 'Only an IP address you add to your Access List will be able to connect to your project's clusters. You can manage existing IP entries via the Network Access Page.'

Déploiement d'une base de données MongoDB via Atlas

Votre cluster de base de données est désormais créé



Déploiement d'une base de données MongoDB via Atlas

Ajoutez l'IP 0.0.0.0/0 pour permettre une connexion depuis n'importe où (*non recommandé*)

The screenshot shows the MongoDB Atlas interface for Project 0. The left sidebar is on the 'Network Access' page, which is highlighted with a red box and a red '1'. The main content area shows the 'Network Access' configuration for the selected cluster. At the top right, the '+ ADD IP ADDRESS' button is highlighted with a red box and a red '2'. Below this button, a yellow warning banner indicates that the current IP address is not added, preventing connections from that address. A table below the banner lists the current IP addresses:

IP Address	Comment	Status	Actions
37.170.164.249/32	My IP Address	Active	EDIT DELETE

The system status is 'All Good'.

Déploiement d'une base de données MongoDB via Atlas

Ajouter l'IP 0.0.0.0/0 pour permettre une connexion depuis n'importe où (*non recommandé*)

The screenshot shows the MongoDB Atlas interface with a modal dialog titled "Add IP Access List Entry". The dialog contains the following elements:

- Buttons: "ADD CURRENT IP ADDRESS" and "ALLOW ACCESS FROM ANYWHERE".
- Text: "Atlas only allows client connections to a cluster from entries in the project's IP Access List. Each entry should either be a single IP address or a CIDR-notated range of addresses. [Learn more.](#)"
- Form fields: "Access List Entry:" with the value "0.0.0.0/0" (highlighted with a red box and the number 1), and "Comment:" with the placeholder "Optional comment describing this entry".
- Options: A toggle switch "This entry is temporary and will be deleted in" set to "6 hours".
- Buttons: "Cancel" and "Confirm" (highlighted with a red box and the number 2).

The background shows the "Network Access" section of the Atlas console, including a table of IP addresses and a system status indicator.

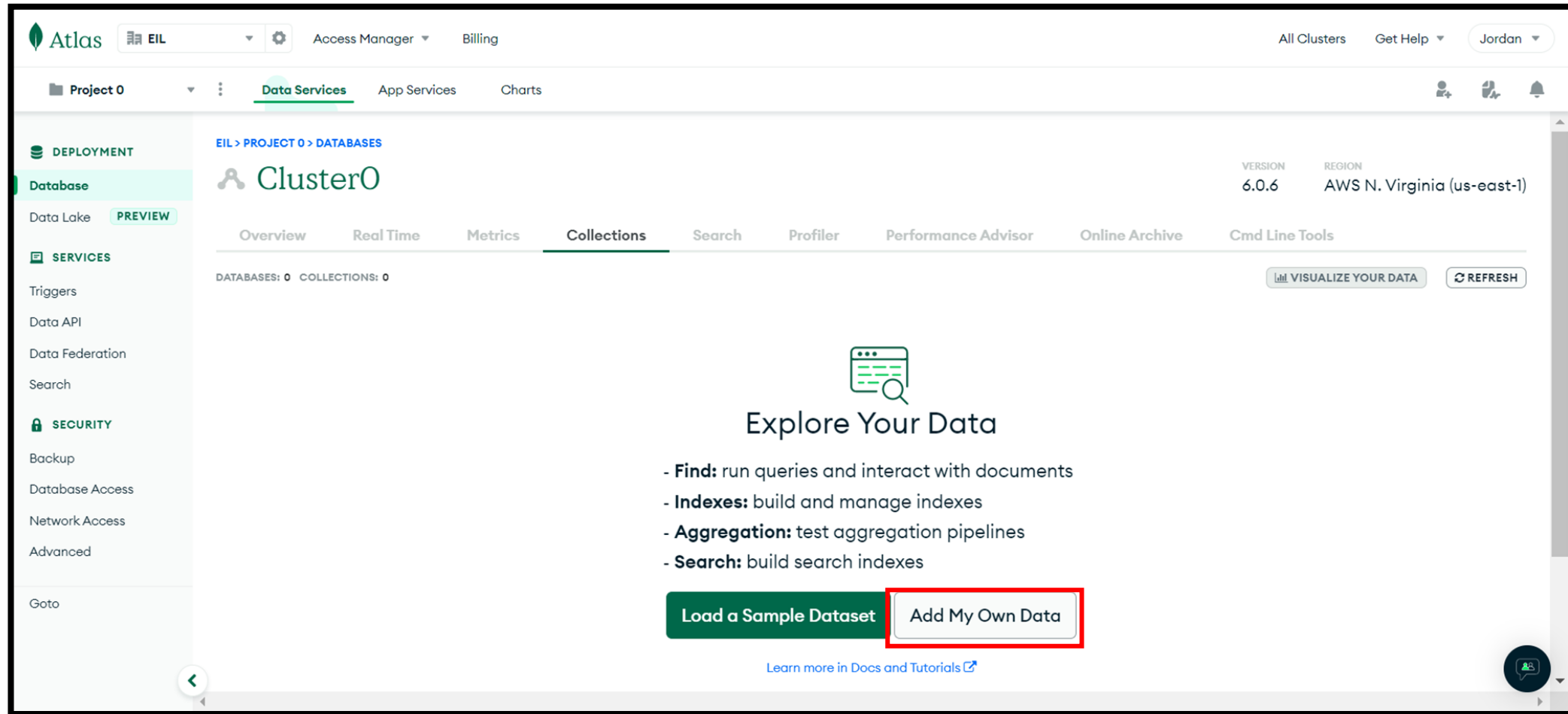
Initialisation de la collection

Depuis le menu « Database », cliquez sur « Browse Collections » pour afficher vos collections

The screenshot shows the MongoDB Atlas interface. The top navigation bar includes the Atlas logo, a dropdown menu for 'EIL', and links for 'Access Manager' and 'Billing'. On the right, there are links for 'All Clusters', 'Get Help', and a user profile 'Jordan'. The main navigation bar shows 'Project 0' and 'Data Services' (which is highlighted). A sidebar on the left contains categories: 'DEPLOYMENT' (with 'Database' selected), 'SERVICES', and 'SECURITY'. The main content area is titled 'Database Deployments' and features a search bar, a '+ Create' button, and a 'Load sample datasets to Cluster0.' section with 'Load sample dataset' and 'Dismiss' buttons. Below this, there are buttons for 'Cluster0': 'Connect', 'View Monitoring', 'Browse Collections' (highlighted with a red box), and a menu icon. At the bottom, there are performance metrics for 'Enhance Your Experience', including Read (R) and Write (W) operations, Connections, In/Out data rates, and Data Size.

Initialisation de la collection

Cliquez sur « Add My Own Data »



The screenshot shows the Atlas web interface for a project named 'Project 0'. The main navigation bar includes 'Atlas', 'EIL', 'Access Manager', and 'Billing'. The user is logged in as 'Jordan'. The current view is 'Data Services' for 'Cluster0', with a version of 6.0.6 and region 'AWS N. Virginia (us-east-1)'. The 'Collections' tab is selected, showing 'DATABASES: 0' and 'COLLECTIONS: 0'. A central message says 'Explore Your Data' with a list of actions: Find, Indexes, Aggregation, and Search. At the bottom, two buttons are visible: 'Load a Sample Dataset' and 'Add My Own Data', with the latter highlighted by a red box. A 'Learn more in Docs and Tutorials' link is also present.

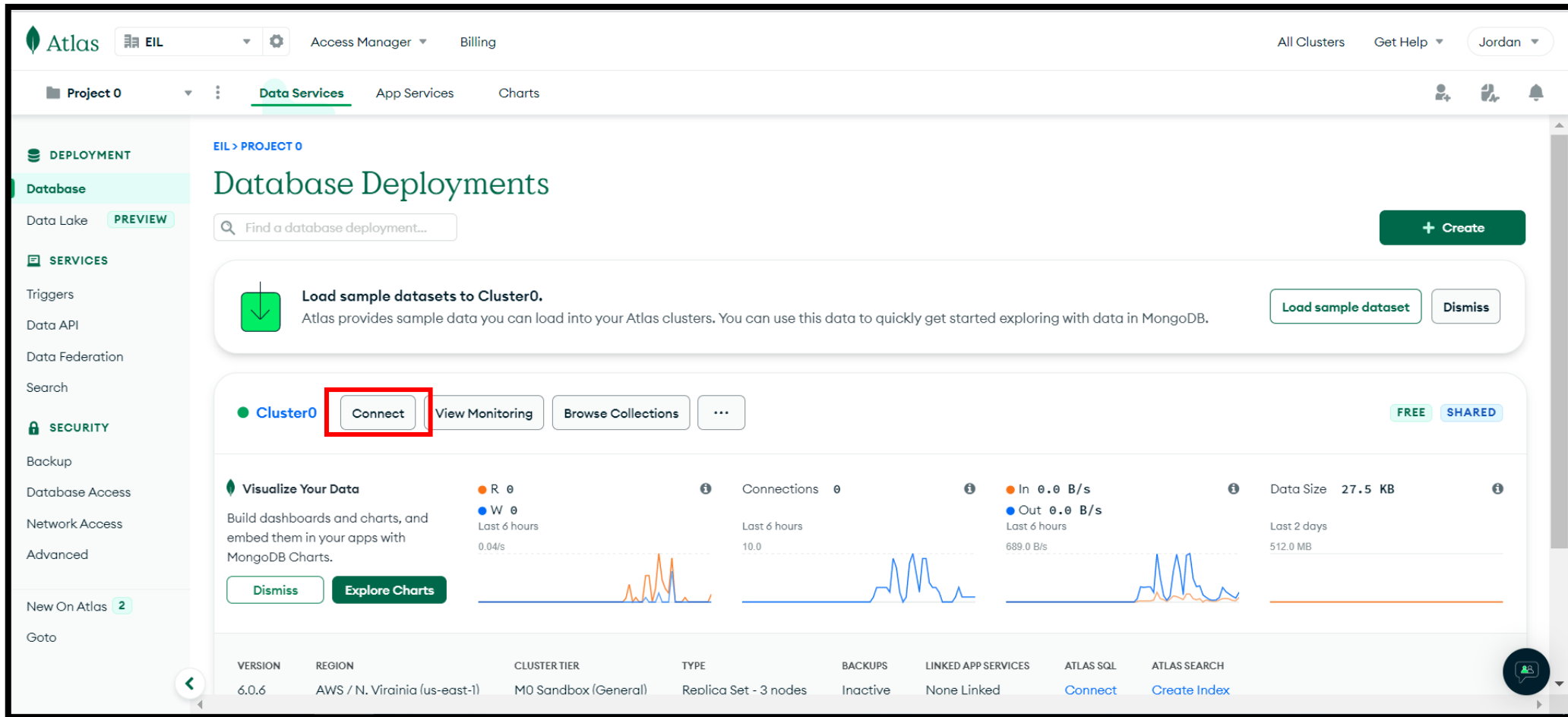
Initialisation de la collection

Renseignez les champs puis cliquez sur « Create » : votre collection est créée

The screenshot shows the Atlas 'Create Database' dialog box. The 'Database name' field contains 'eilco_web' and is marked with a red box and the number '1'. The 'Collection name' field contains 'students' and is marked with a red box and the number '2'. The 'Additional Preferences' section has three unchecked options: 'Capped Collection', 'Time Series Collection', and 'Clustered Index Collection'. The 'Create' button is highlighted with a red box and the number '3'. The background shows the Atlas interface for 'Cluster0' with 'Databases: 0' and 'Collections: 0'.

Insertion des données via MongoDB Compass

Depuis le menu « Database », cliquez sur « Connect » pour récupérer l'URL de connexion

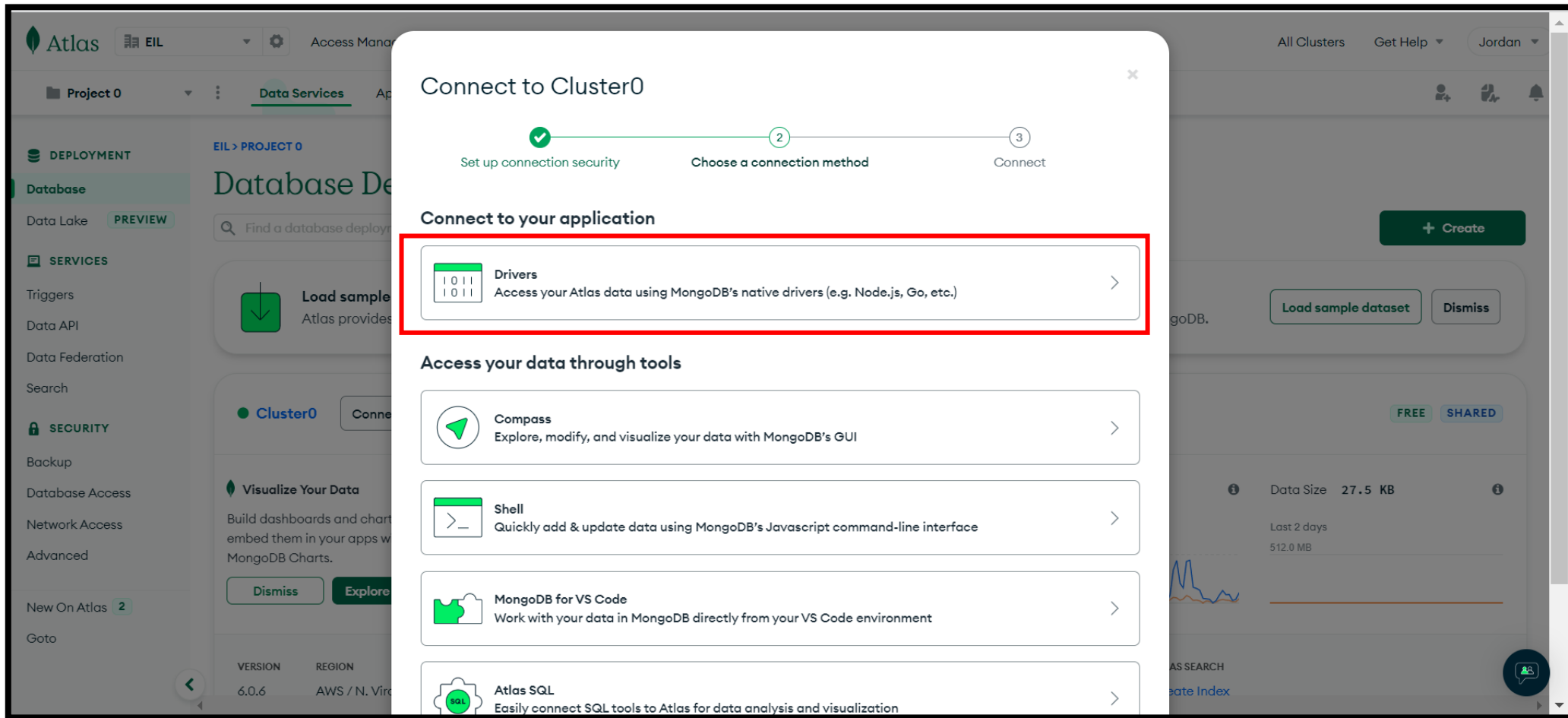


The screenshot displays the MongoDB Atlas interface for 'Project 0'. The left sidebar shows the 'Database' menu item selected. The main content area is titled 'Database Deployments' and features a search bar and a '+ Create' button. Below this, there is a section for 'Cluster0' with a 'Connect' button highlighted by a red box. Other buttons include 'View Monitoring', 'Browse Collections', and a three-dot menu. The 'Cluster0' section also shows 'FREE' and 'SHARED' labels. Below the buttons, there are four charts: 'Visualize Your Data', 'Connections', 'In/Out B/s', and 'Data Size'. At the bottom, a table lists cluster details:

VERSION	REGION	CLUSTER TIER	TYPE	BACKUPS	LINKED APP SERVICES	ATLAS SQL	ATLAS SEARCH
6.0.6	AWS / N. Virginia (us-east-1)	M0 Sandbox (General)	Replica Set - 3 nodes	Inactive	None Linked	Connect	Create Index

Insertion des données via MongoDB Compass

Récupération de l'URL de connexion



Insertion des données via MongoDB Compass

Récupération de l'URL de connexion

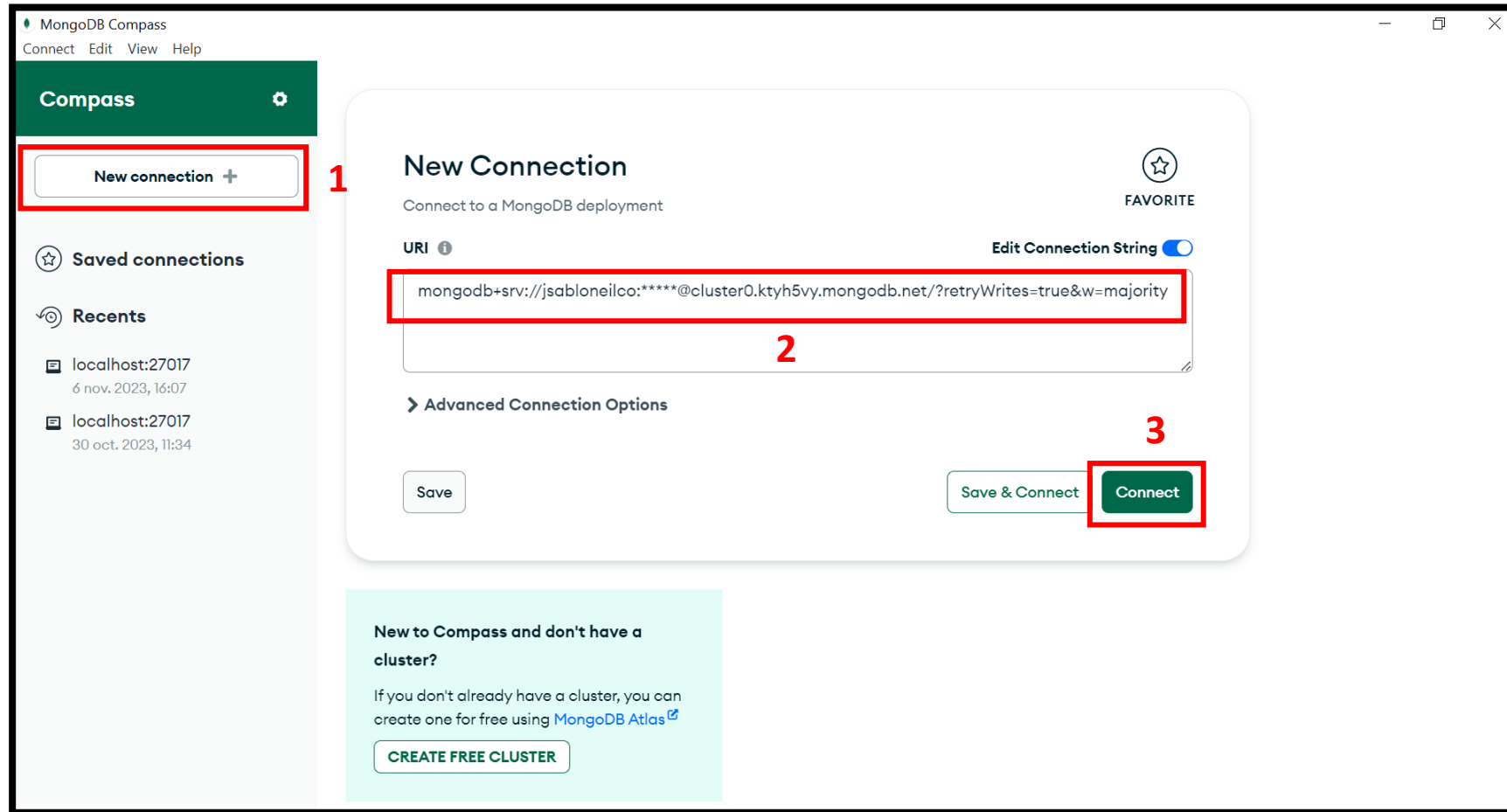
The screenshot displays the MongoDB Atlas interface with a modal window titled "Connecting with MongoDB Driver". The modal is divided into three main sections:

- 1. Select your driver and version**
We recommend installing and using the latest driver version.
Driver: Node.js (selected)
Version: 5.5 or later (selected)
- 2. Install your driver**
Run the following on the command line:
`npm install mongodb`
[View MongoDB Node.js Driver installation instructions.](#)
- 3. Add your connection string into your application code**
 View full code sample
`mongodb+srv://jsabloneilco:<password>@cluster0.ktyh5vy.mongodb.net/?retryWrites=true&w=majority`
Replace **<password>** with the password for the **jsabloneilco** user. Ensure any option params are [URL encoded](#).

At the bottom of the modal, there is a "RESOURCES" section with links to "Get started with the Node.js Driver", "Access your Database Users", "Node.js Starter Sample App", and "Troubleshoot Connections". A red box highlights the copy icon next to the connection string.

Insertion des données via MongoDB Compass

Démarrer le logiciel MongoDB Compass et se connecter à la base de données



!/ L'URL est à adapter selon vos informations

Insertion des données via MongoDB Compass

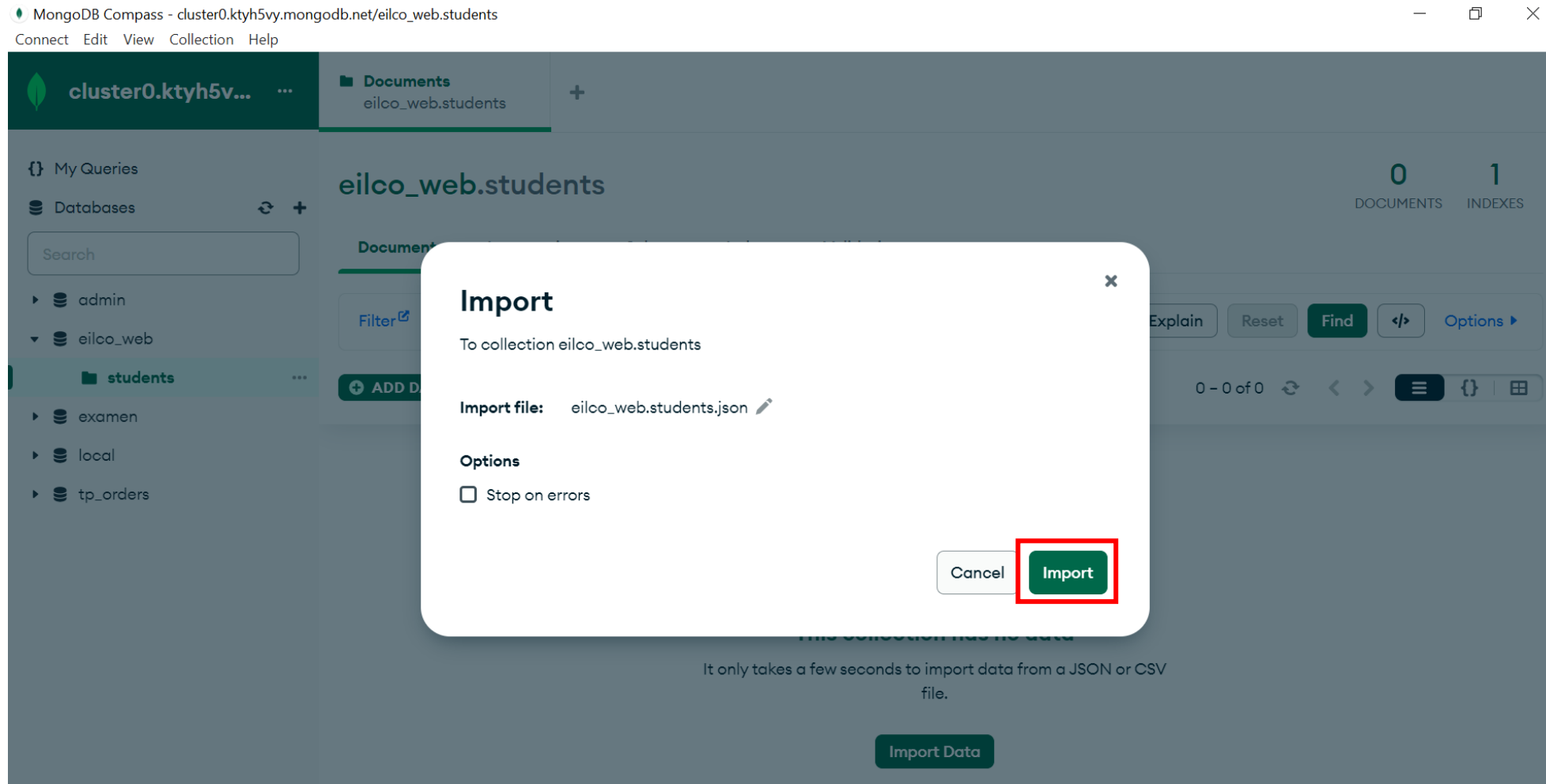
Téléchargez ce fichier de données : [eilco_web.students.json](#)

Sélectionnez la collection students puis « Import Data »

The screenshot shows the MongoDB Compass interface. On the left sidebar, the 'students' collection is highlighted with a red box and a red '1'. The main panel shows the 'eilco_web.students' collection with 0 documents and 1 index. Below the collection name, there are tabs for 'Documents', 'Aggregations', 'Schema', 'Indexes', and 'Validation'. A search bar is present with the text 'Type a query: { field: 'value' } or Generate query'. Below the search bar, there are buttons for 'ADD DATA' and 'EXPORT DATA'. The 'ADD DATA' button is highlighted with a red box and a red '2'. The main content area displays a message: 'This collection has no data. It only takes a few seconds to import data from a JSON or CSV file.'

Insertion des données via MongoDB Compass

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



Insertion des données via MongoDB Compass

Les données ont bien été importées

The screenshot shows the MongoDB Compass interface. The top bar indicates the connection to 'cluster0.ktyh5v...'. The left sidebar shows the database structure with 'eilco_web' selected and 'students' highlighted. The main area displays the 'eilco_web.students' collection with 2 documents and 1 index. The documents are:

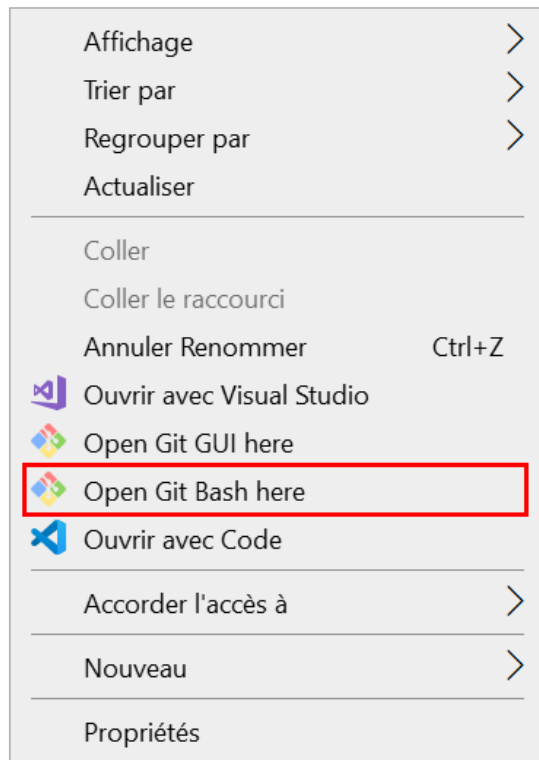
```
{ "_id": ObjectId('65d217baf2e0ff45ce929cc3'), "name": "Doe", "firstname": "Jane", "age": 20 }
```

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

A notification at the bottom left states: "Import completed. 2 documents imported."

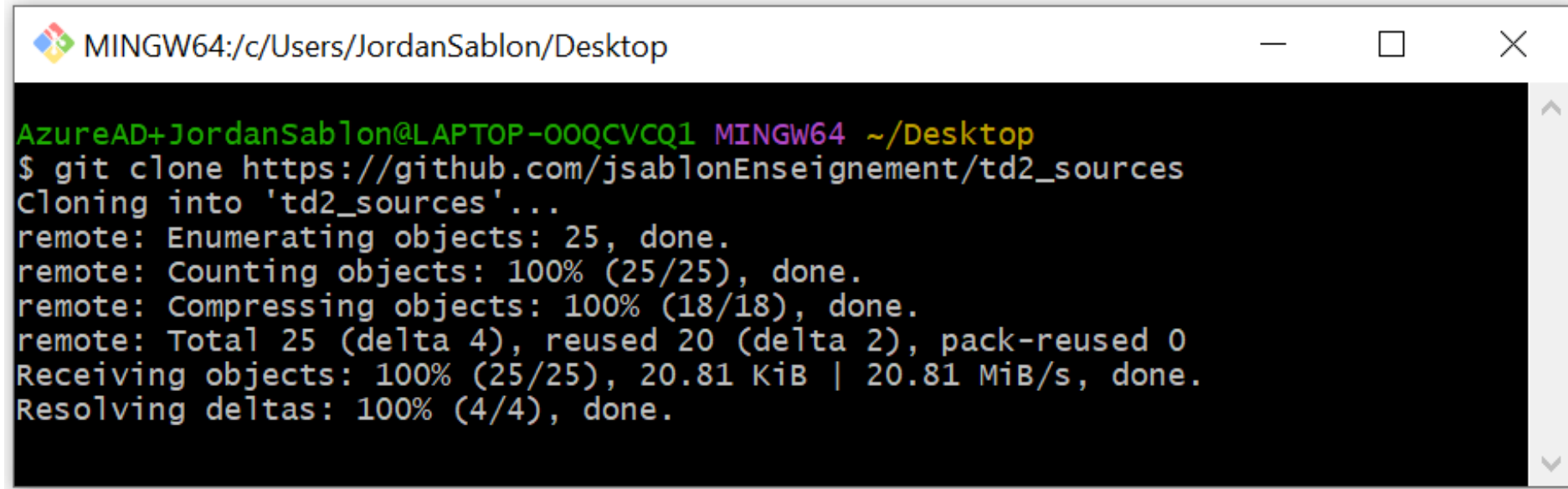
Récupération du code source du projet & installation des dépendances

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 :



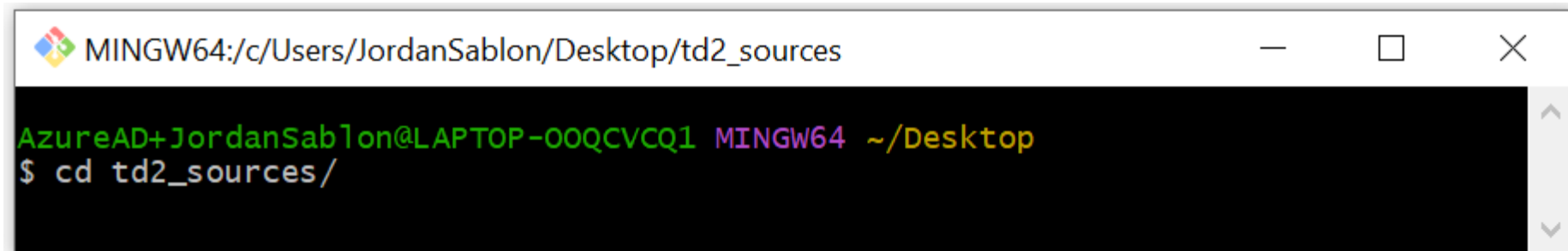
Récupération du code source du projet & installation des dépendances

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.
```

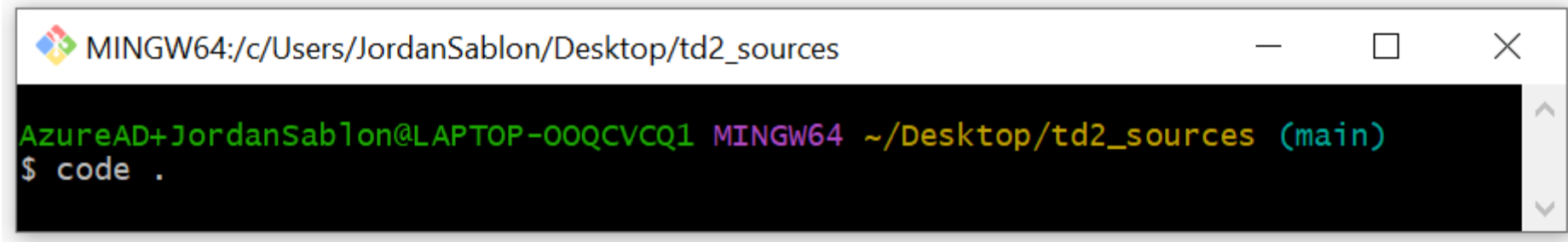
4. Se rendre dans le dossier précédemment cloné :



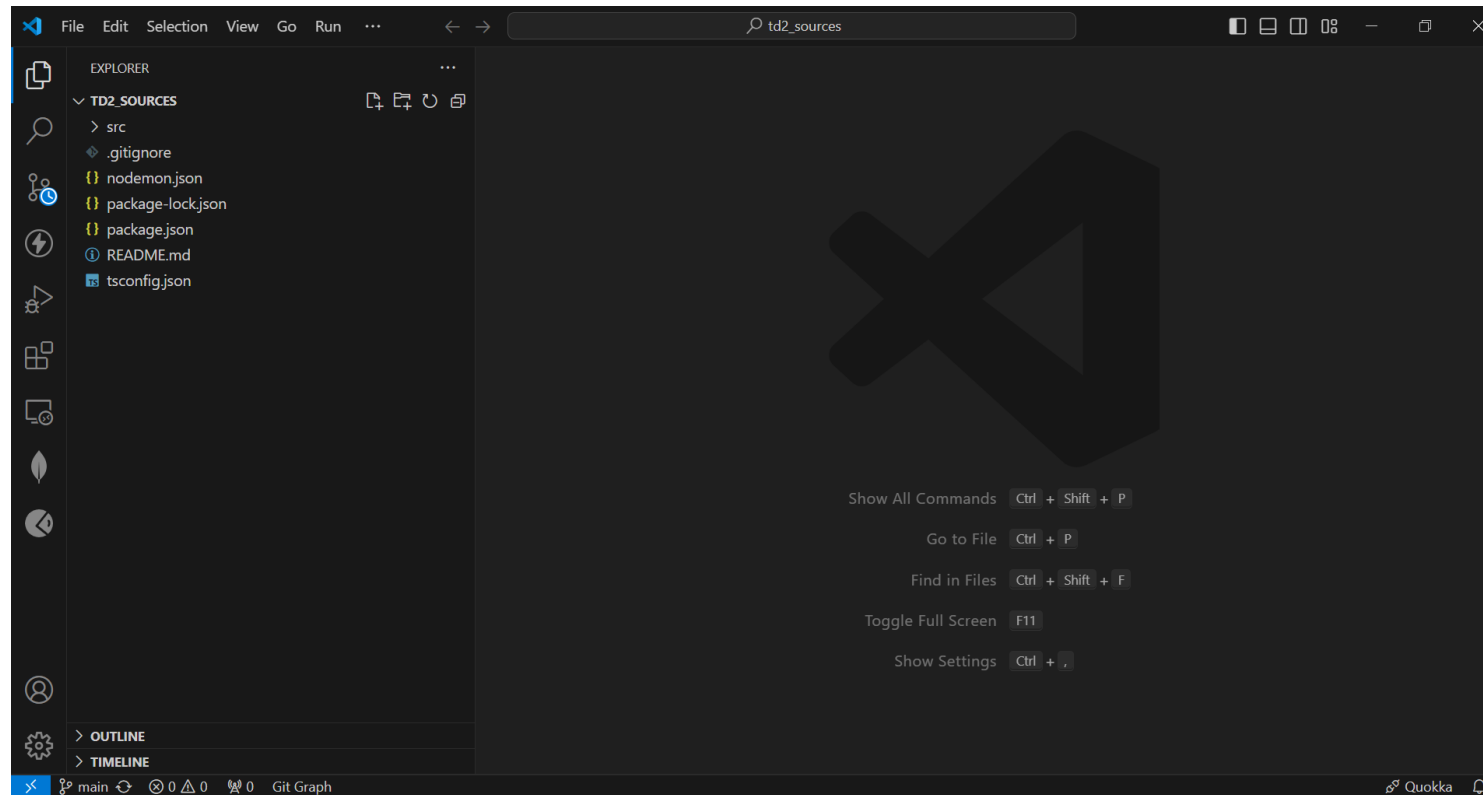
```
MINGW64:/c/Users/JordanSablon/Desktop/td2_sources  
AzureAD+JordanSablon@LAPTOP-00QCVCQ1 MINGW64 ~/Desktop  
$ cd td2_sources/
```

Récupération du code source du projet & installation des dépendances

5. Ouvrir le projet dans VSCode :

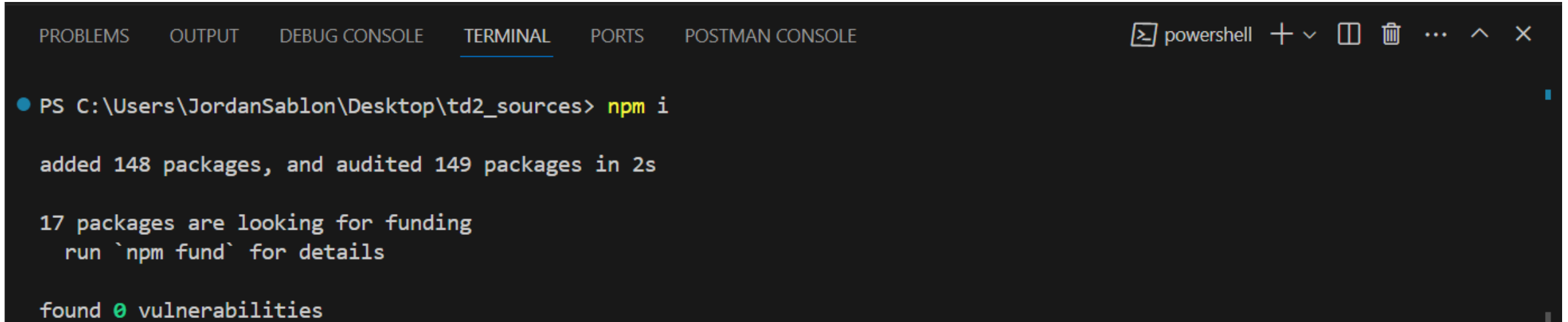


```
MINGW64:/c/Users/JordanSablon/Desktop/td2_sources  
AzureAD+JordanSablon@LAPTOP-00QCVCQ1 MINGW64 ~/Desktop/td2_sources (main)  
$ code .
```



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

Installez la dépendance **mongoose**

```
npm install mongoose
```

Importez **mongoose** dans le code

```
import mongoose from "mongoose";
```

Importez **mongoose** dans le fichier **index.ts**

```
import mongoose from "mongoose";
```

Remplacez la chaîne de connexion à la base de données dans le fichier **index.ts**

```
mongoose.connect(  
  "mongodb+srv://<username>:<password>@<domain>.mongodb.net/eilco_web?retryWrites=true&w=majority");  
const db = mongoose.connection;
```

/*! L'URL est à adapter selon vos informations

Ajout de logs en cas de succès / d'échec de connexion

```
db.on("error", console.error.bind(console, "connection error: "));
db.once("open", function () {
  console.log("Connected successfully");
});
```

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
```

Créez un dossier **models** dans le dossier **/src**

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

Implémentez le schéma représentant la collection **students** donné à la diapo suivante

Création du modèle

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

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

export interface Student {
  _id: Types.ObjectId;
  name: String;
  firstname: String;
  age: Number;
}

export const StudentModel = model(
  "students",
  StudentSchema
);
```

Récupération de tous les étudiants

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

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

Modifier la fonction *getStudents* :

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

- ① On applique une fonction `find()` sur le modèle **Student** 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'attendre du retour de la base de données pour retourner les étudiants. 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'attendre l'exécution de notre service pour retourner les étudiants. On précise alors 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 StudentModel.findOne({ _id: new Types.ObjectId(id) });  
};
```

- ① On applique une fonction `findOne()` avec un critère sur l'id sur le modèle **Student** 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'attendre du retour de la base de données pour retourner les étudiants. 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'attendre l'exécution de notre service pour retourner les étudiants. On précise alors que notre fonction est asynchrone (*async*)

Création d'un étudiant

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

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

- ① On crée une nouvelle instance du modèle **Student** 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'attendre du retour de la base de données pour retourner les étudiants. 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(200).json(students);  
};
```

- ① On utilise l'instruction *await* afin de forcer l'attendre l'exécution de notre service pour retourner les étudiants. On précise alors que notre fonction est asynchrone (*async*)

Mise à jour d'un étudiant

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

```
const updateStudent = (id: string, studentToUpdate: IStudent) => {
  await StudentModel.updateOne(
    {
      _id: new Types.ObjectId(id),
    },
    studentToUpdate
  );
  return getStudents();
};
```

- ① On applique une fonction `updateOne()` sur le modèle **Student** 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'attendre du retour de la base de données pour retourner les étudiants. 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 studentUpdated = await StudentsService.updateStudent(  
    id,  
    req.body  
  );  
  return res.status(200).json(studentUpdated);  
};
```

- ① On utilise l'instruction *await* afin de forcer l'attendre l'exécution de notre service pour retourner les étudiants. On précise alors 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 StudentModel.deleteOne({ _id: new Types.ObjectId(id) });  
};
```

- ① On applique une fonction `deleteOne()` sur le modèle **Student** 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'attendre du retour de la base de données pour retourner les étudiants. On précise alors que notre fonction est asynchrone (*async*)
- ① On fait le choix de ne pas retourner de données

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(`Student with id ${id} successfully deleted`);  
};
```

- ① On utilise l'instruction *await* afin de forcer l'attendre l'exécution de notre service pour retourner les étudiants. On précise alors que notre fonction est asynchrone (*async*)

Objectifs :

- éviter d'envoyer des informations sensibles sur votre repo git
- spécifier une configuration variable selon les environnements tout en conservant un code générique

Installez la dépendance **dotenv**

```
npm install dotenv
```

Importez **dotenv** dans le fichier **index.ts**

```
import dotenv from "dotenv";
```

Ajoutez cette ligne de code en début de fichier

```
dotenv.config();
```


Créez un fichier **.env** à la racine du dossier **/backend**

```
DB_USERNAME=your_username  
DB_PASSWORD=your_password  
DB_DOMAIN=your_domain.mongodb.net  
DB_NAME=your_db_name
```

Modifiez la ligne de connexion à la base de données dans le fichier **index.ts**

```
mongoose.connect(  
  `mongodb+srv://${process.env.DB_USERNAME}:${process.env.DB_PASSWORD}@${process  
.env.DB_DOMAIN}/${process.env.DB_NAME}?retryWrites=true&w=majority`  
)
```