Backend Pre Requisite Setup

BE1.1

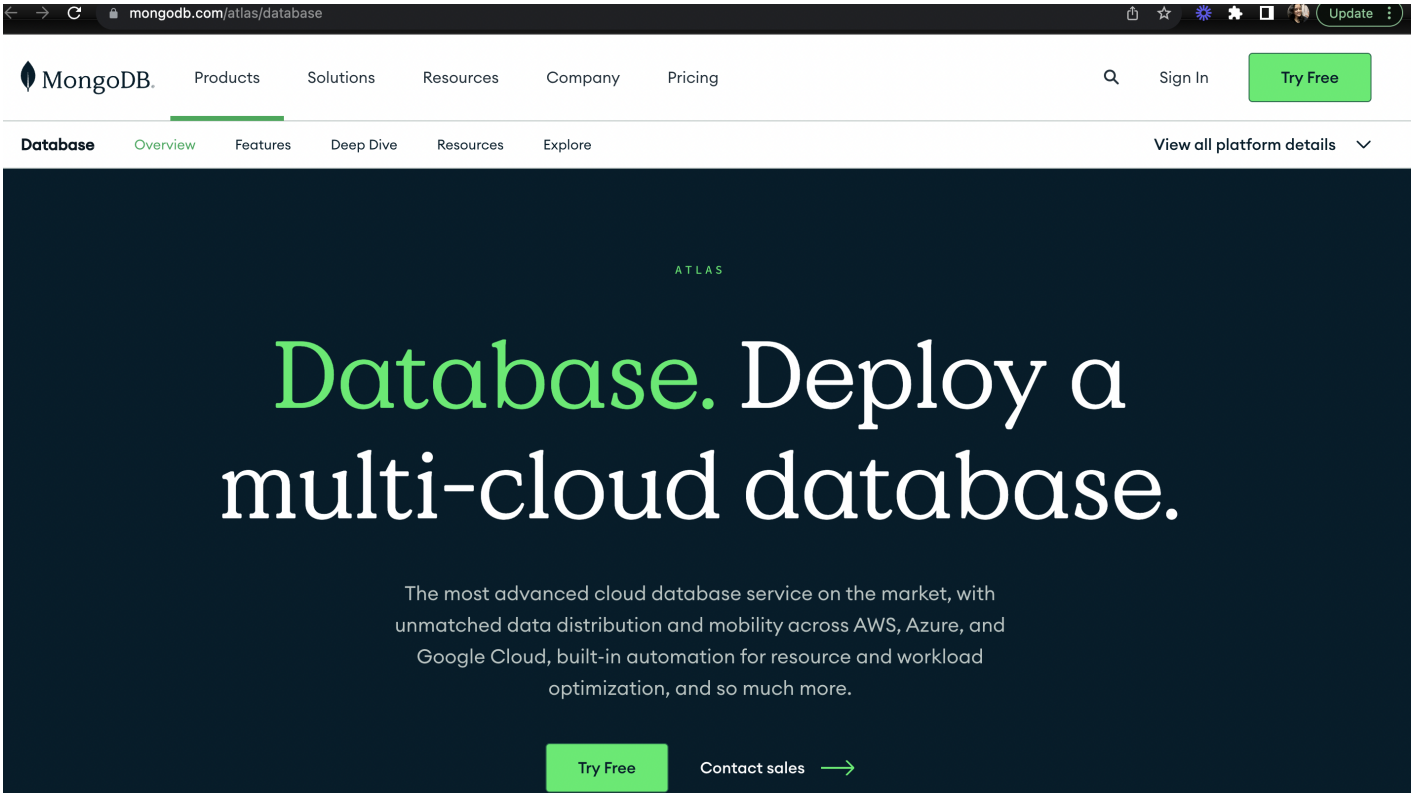
setting up mongoDB with mongoDB atlas and replit

ex01: create your mongoDB atlas account

challenge

Let's start by heading over to the MongoDB Atlas website and creating a free account.

This will be our central hub for managing our MongoDB clusters. Click “Try Free”.

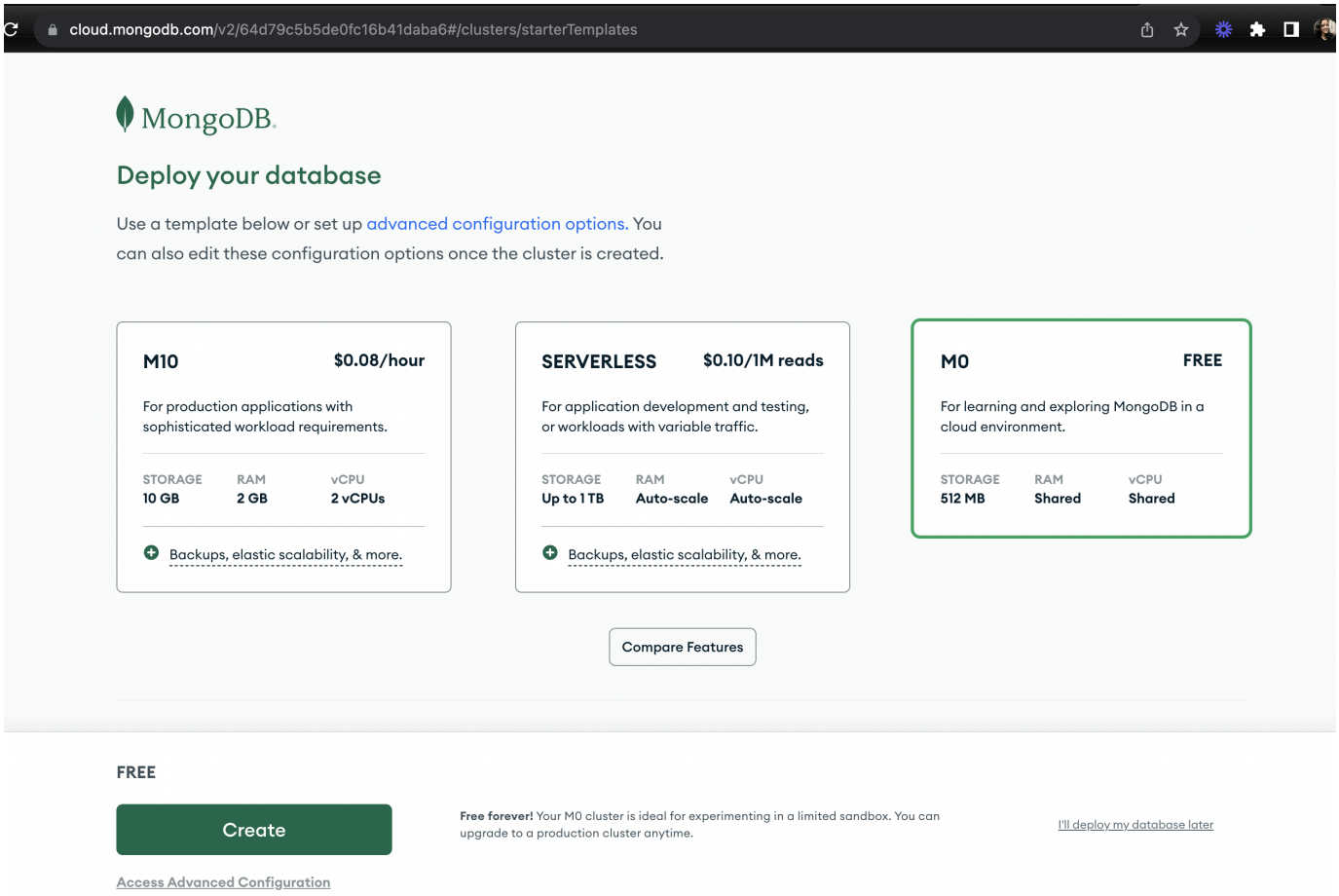


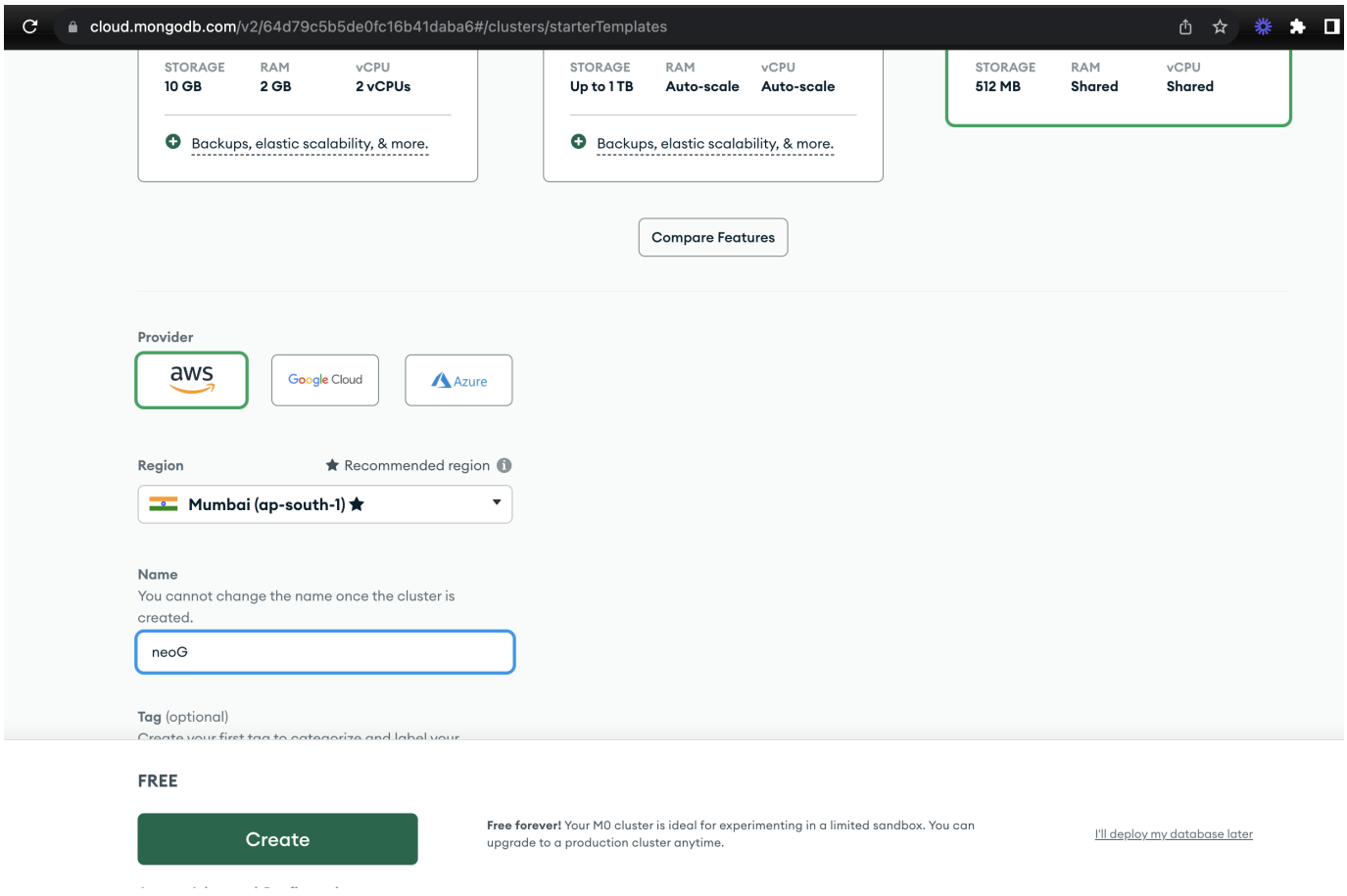
ex02: set up your "neoG" cluster

challenge

1. Once you're logged in to MongoDB Atlas, click on "Build a Cluster" to create a new MongoDB cluster.

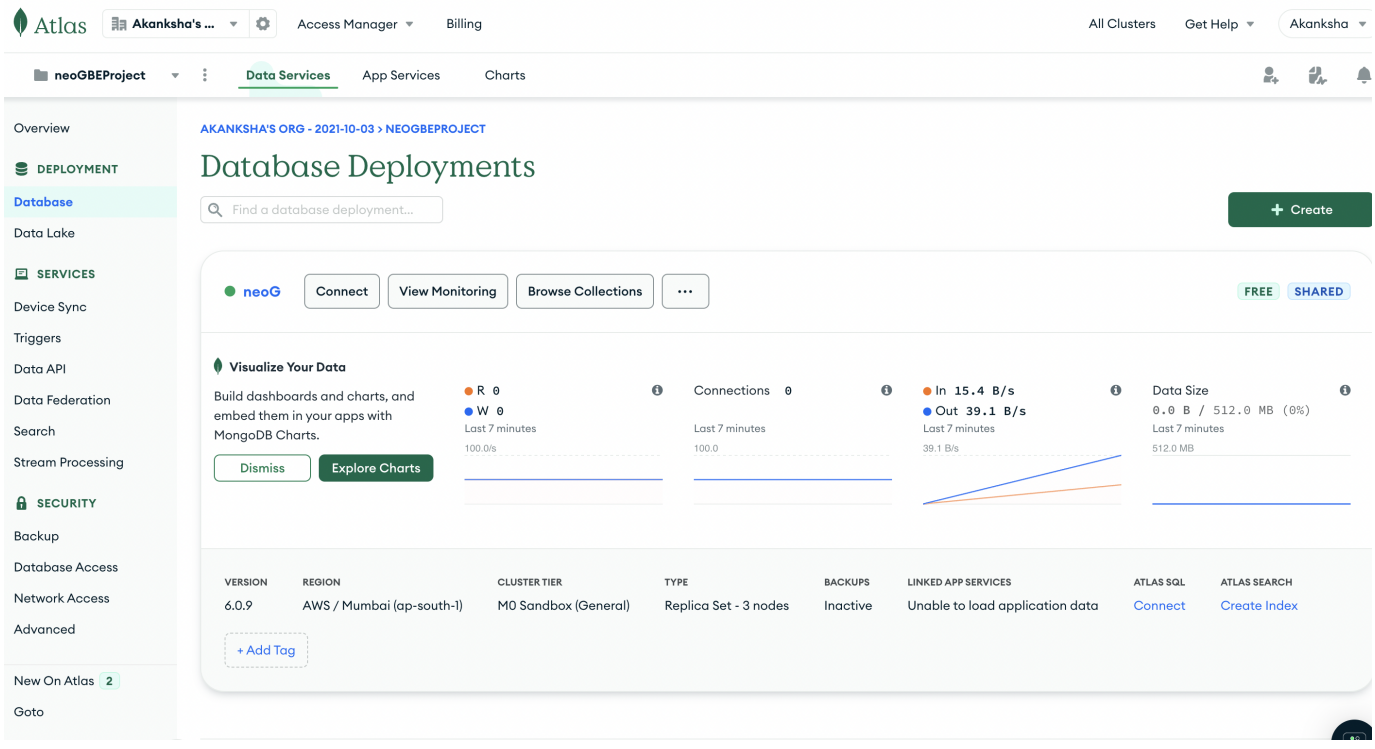
2. Choose your preferred cloud provider and region for the cluster. Let's name the cluster "neoG" for our class.





ex03: connect to the "neoG" cluster

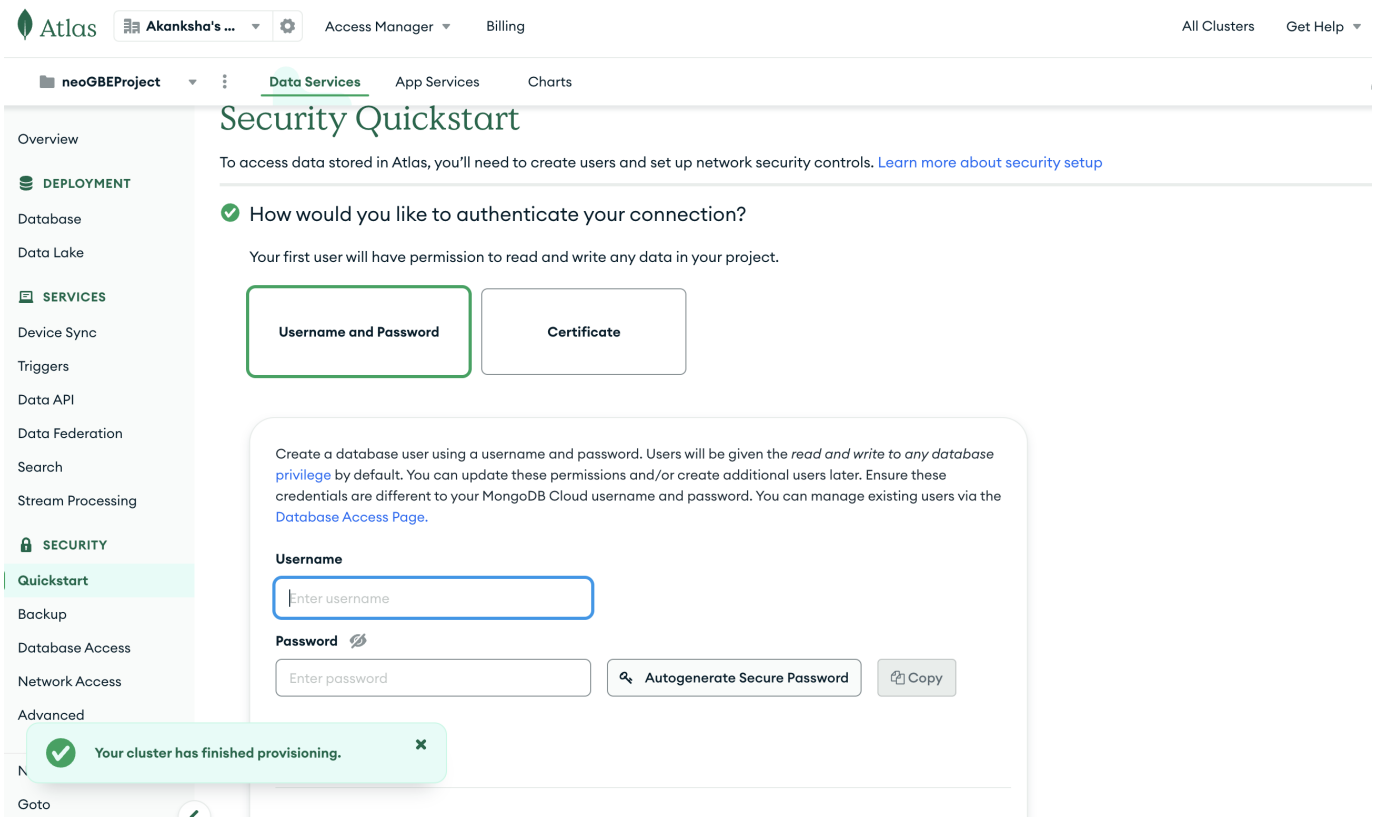
1. Click on "Connect" to set up the connection to your "neoG" cluster.

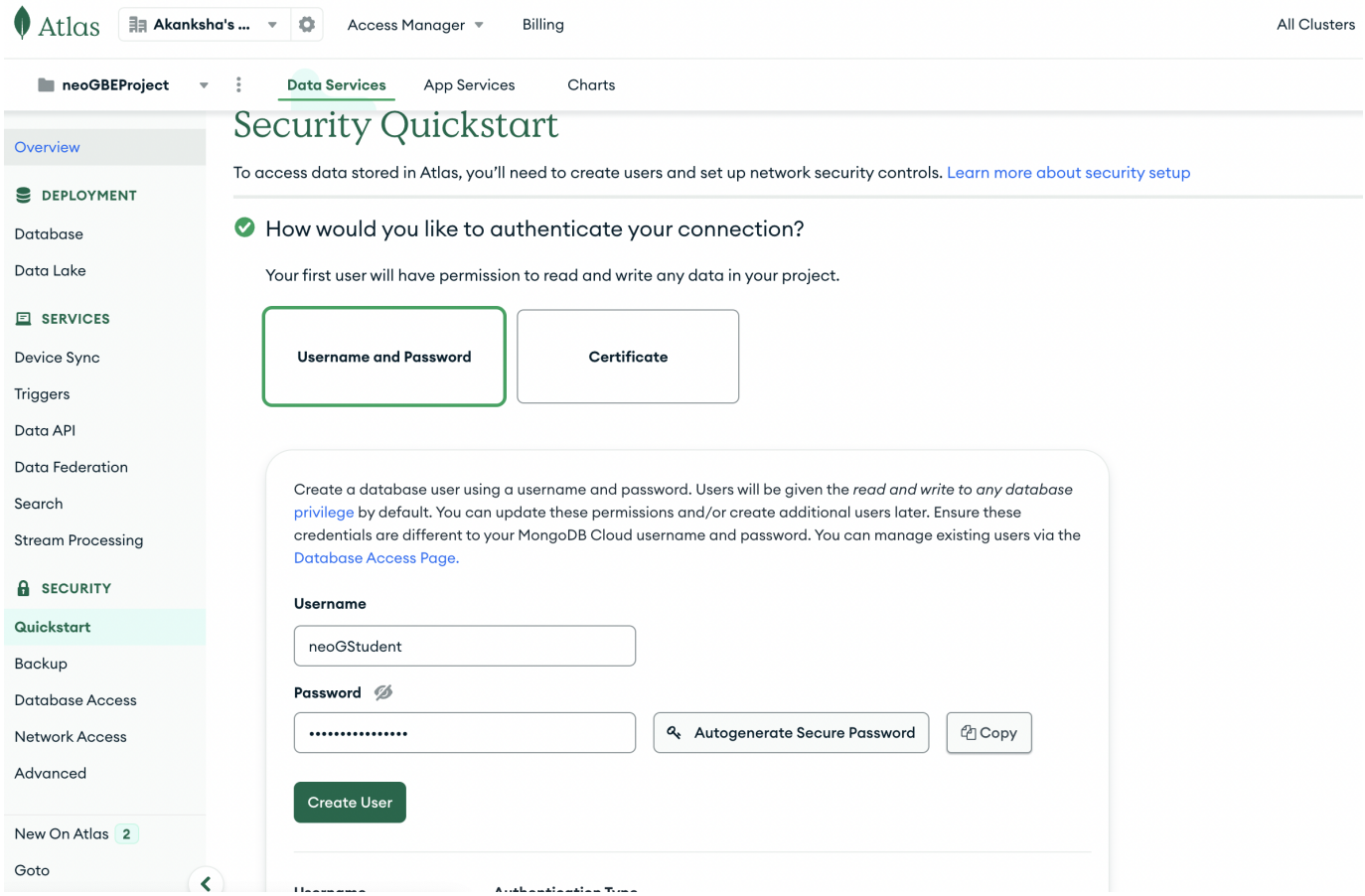


ex04: create your "neoGStudent" database user

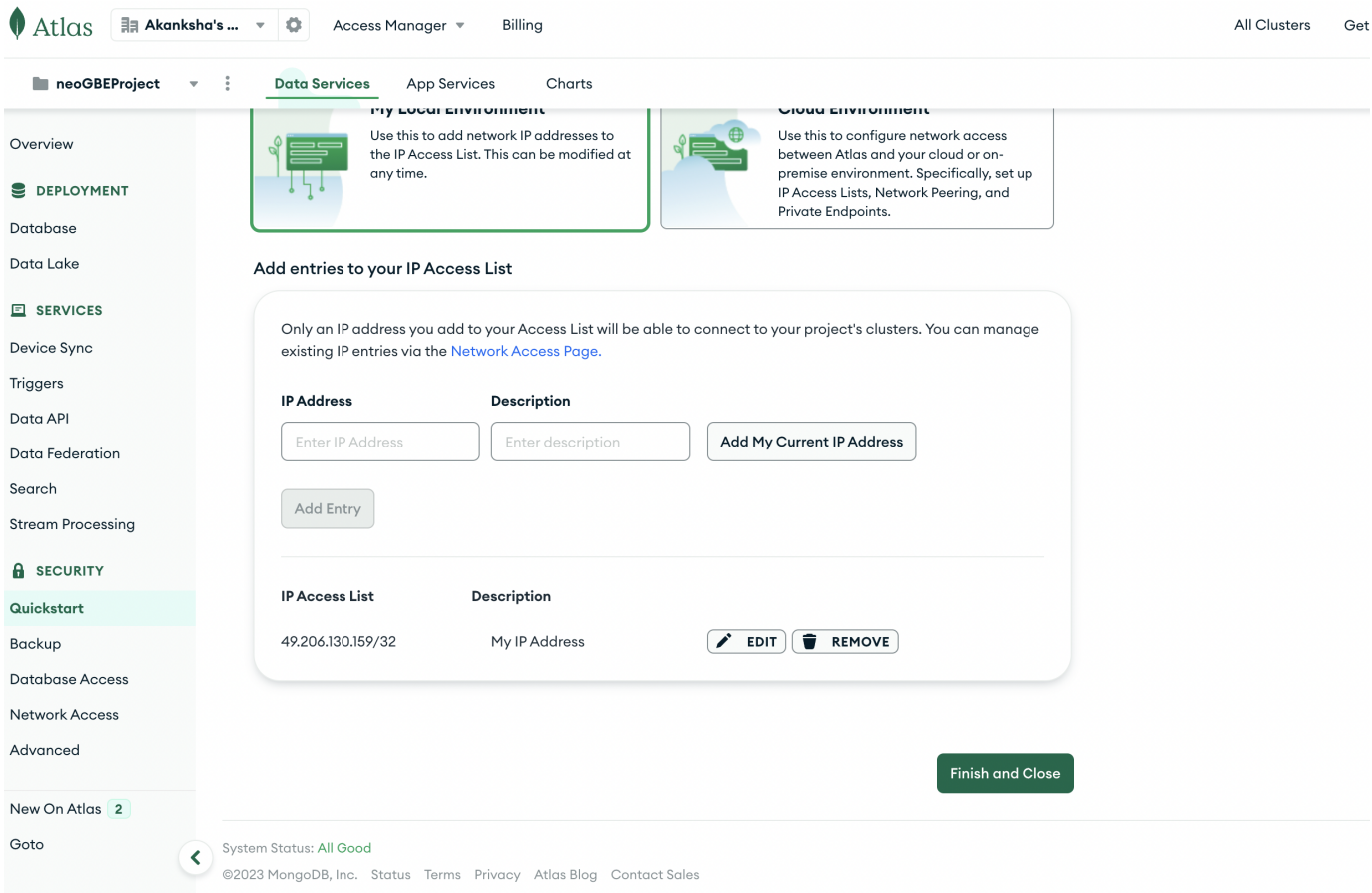
challenge

1. Create a database user with the username "neoGStudent" and grant the necessary privileges for the database you'll be working with in our class.



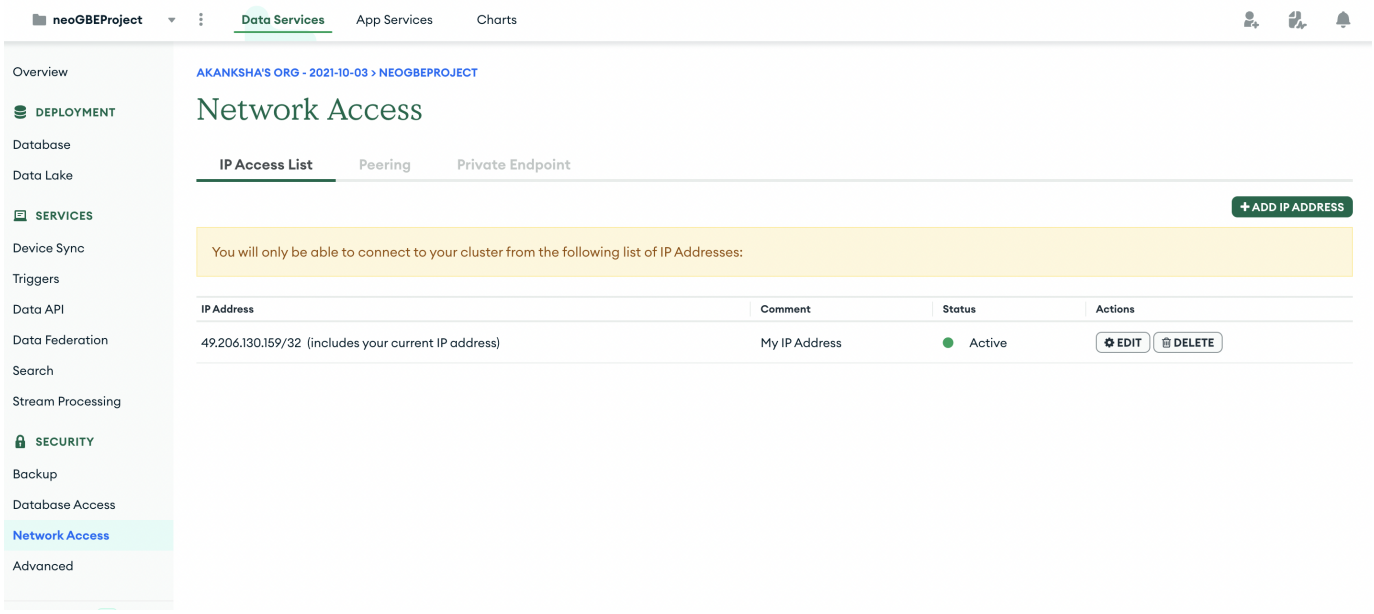


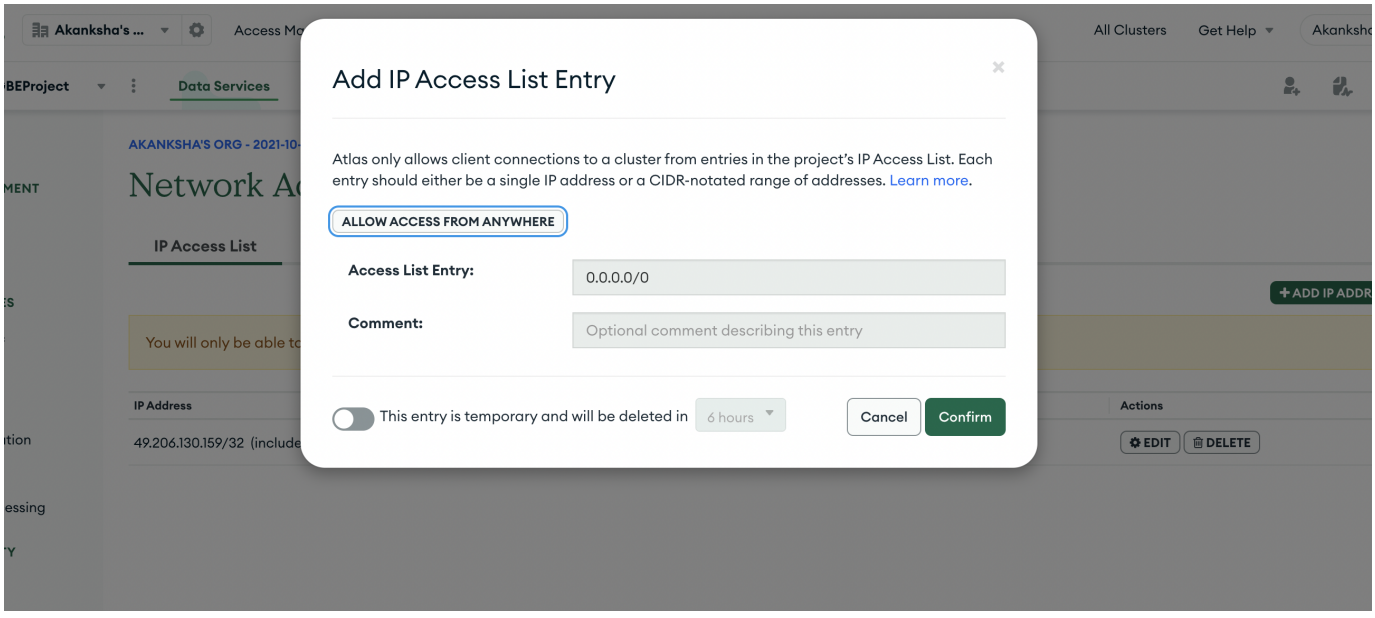
2. Finally Finish and Close.



3. Add your IP address to the IP Whitelist. For simplicity during our class, you can also allow access from anywhere.

1. On the left-hand navigation menu, under Security, click the "Network Access" link.
2. Click the "Add IP Address" button. The "Add IP Whitelist Entry" modal will appear.
3. To make your database accessible from anywhere, click "Allow Access From Anywhere".
4. Click Confirm.

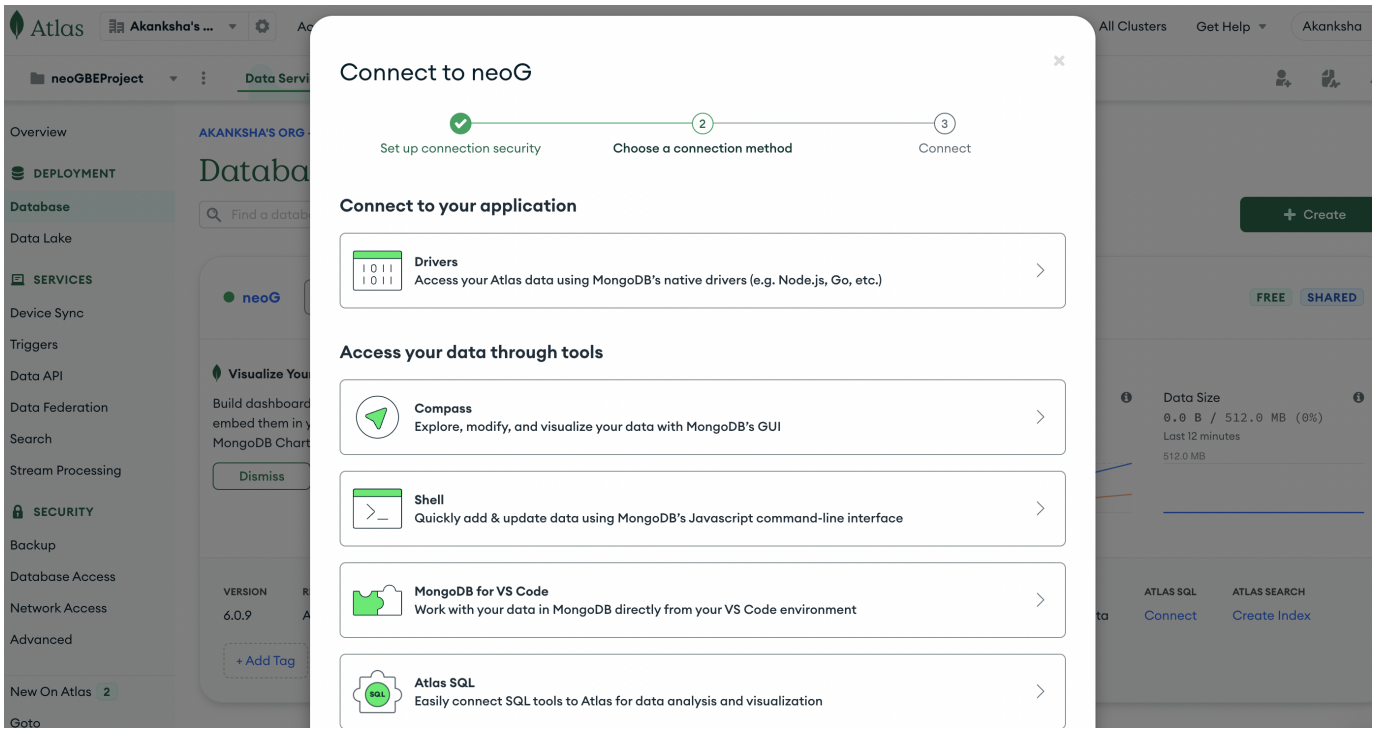




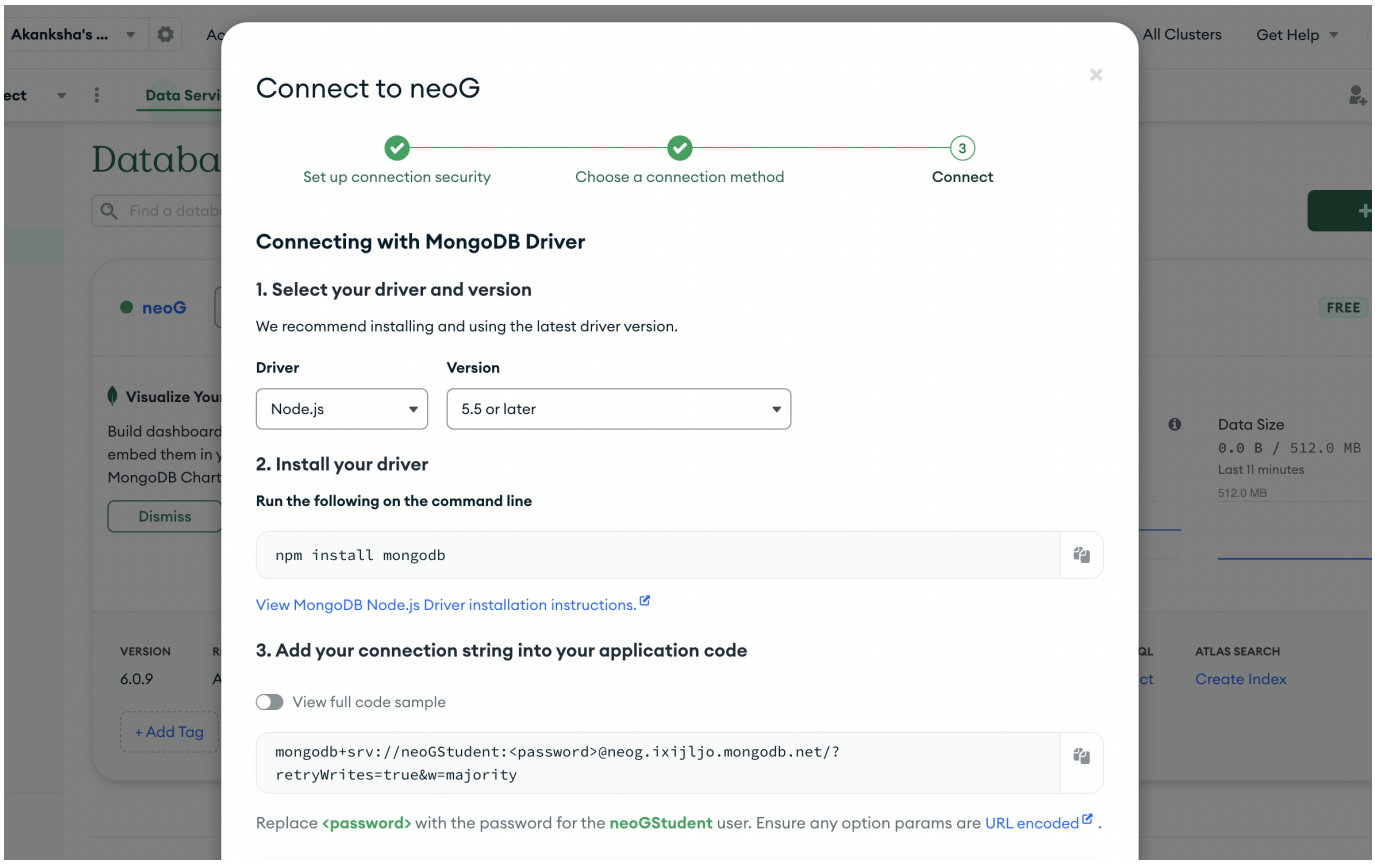
ex05: get your connection string

challenge

1. Click on "Connect" button again and select "Connect your application."



2. Copy the connection string. This is your golden ticket to connecting to your "neoG" MongoDB cluster from your Replit project. Your connection string is the one starting from “mongodb+srv://neoGStudent:……”



ex06: create your replit account

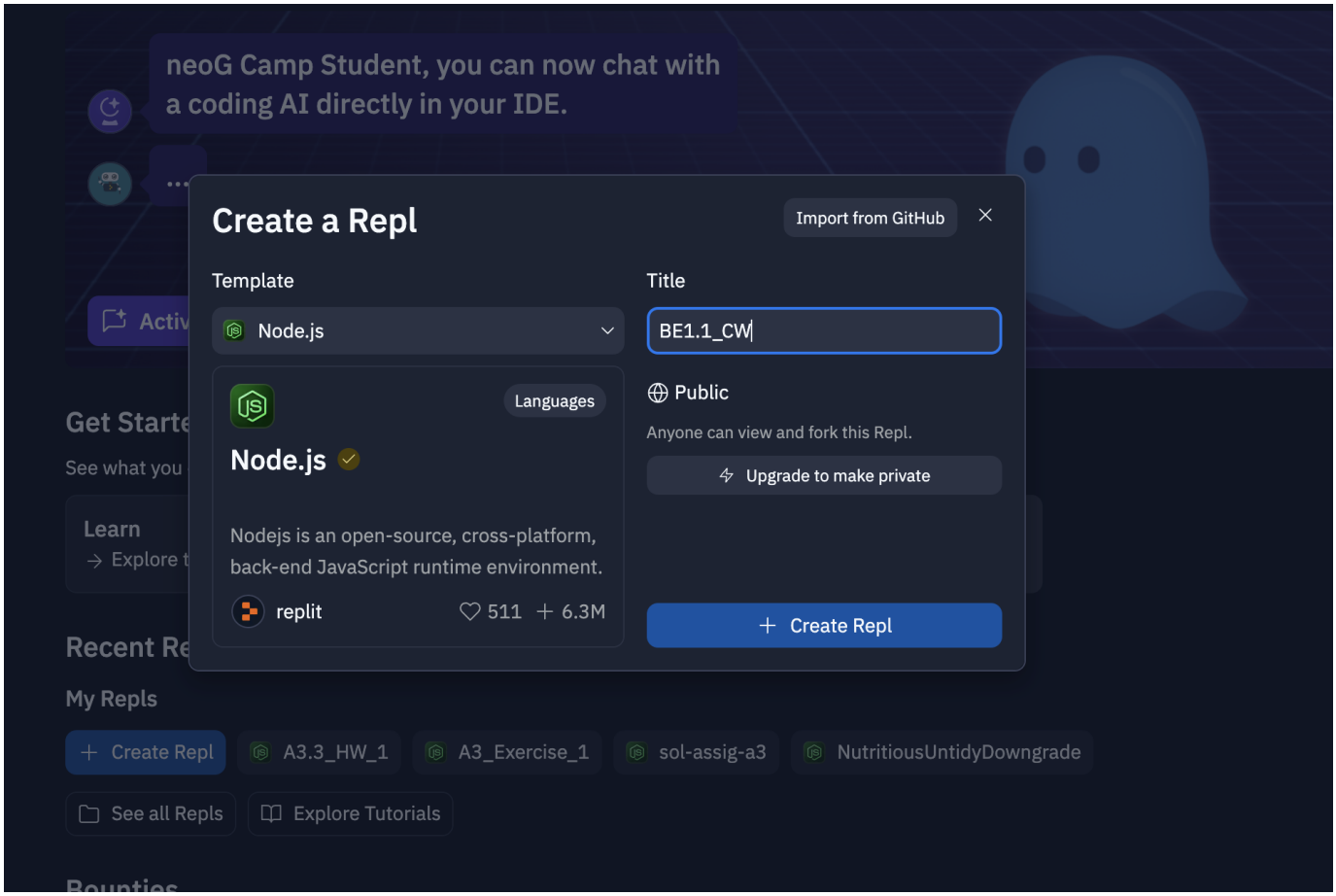
challenge

1. Let's now create a free account on Replit, which is where we'll be writing our Node.js code.

ex07: create a new replit project with node.js

challenge

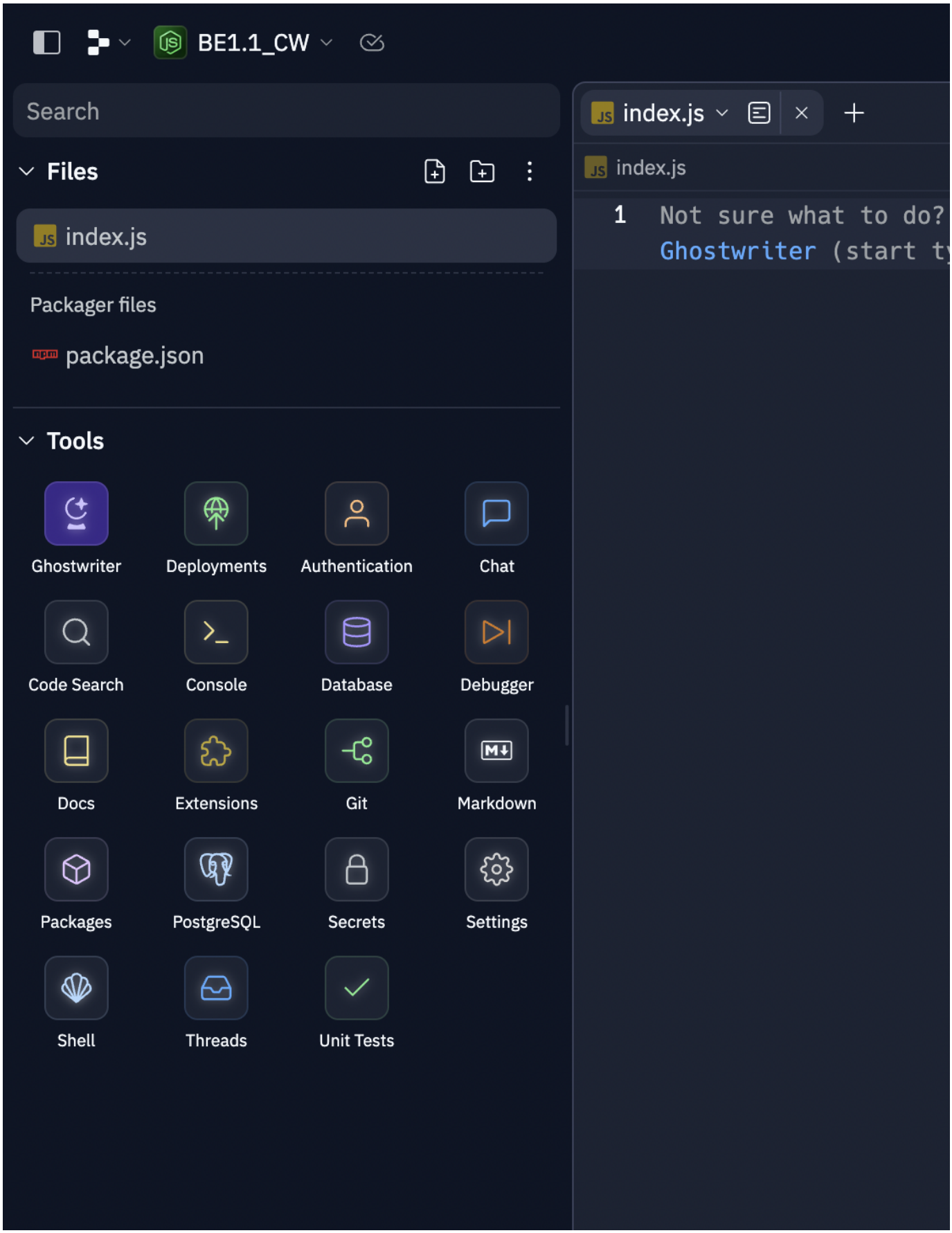
1. Set up a new Replit project and choose Node.js as the programming language you'll be using in our class.

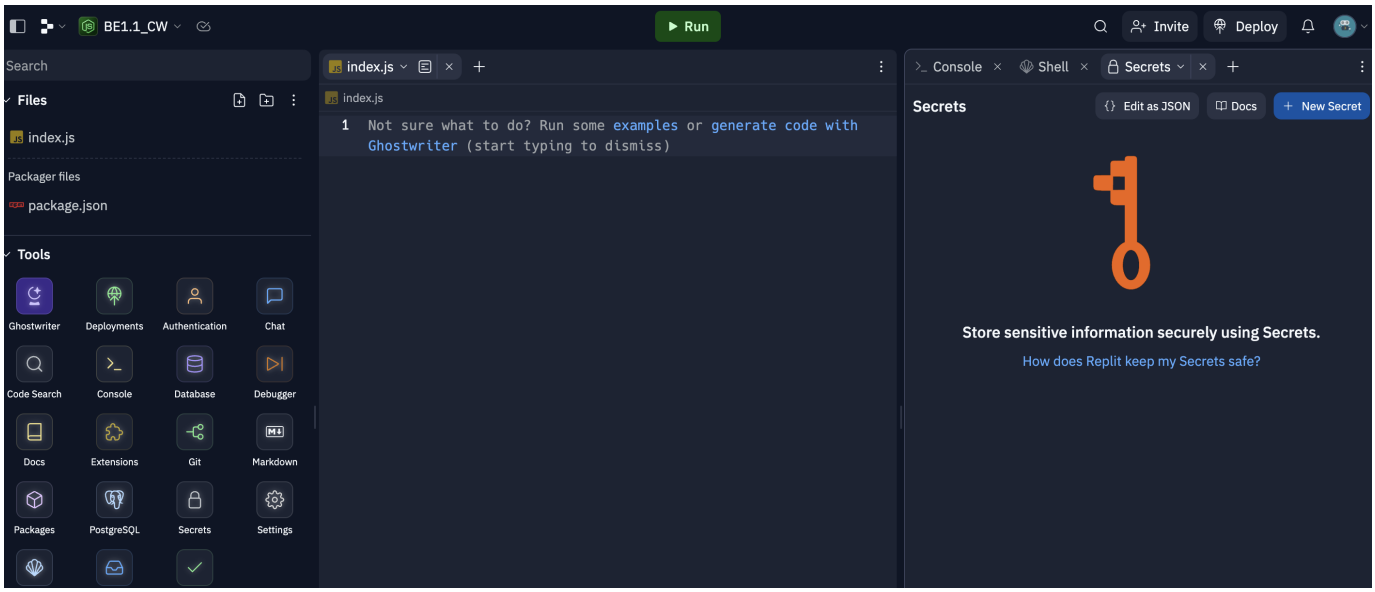


ex08: keep your connection details safe

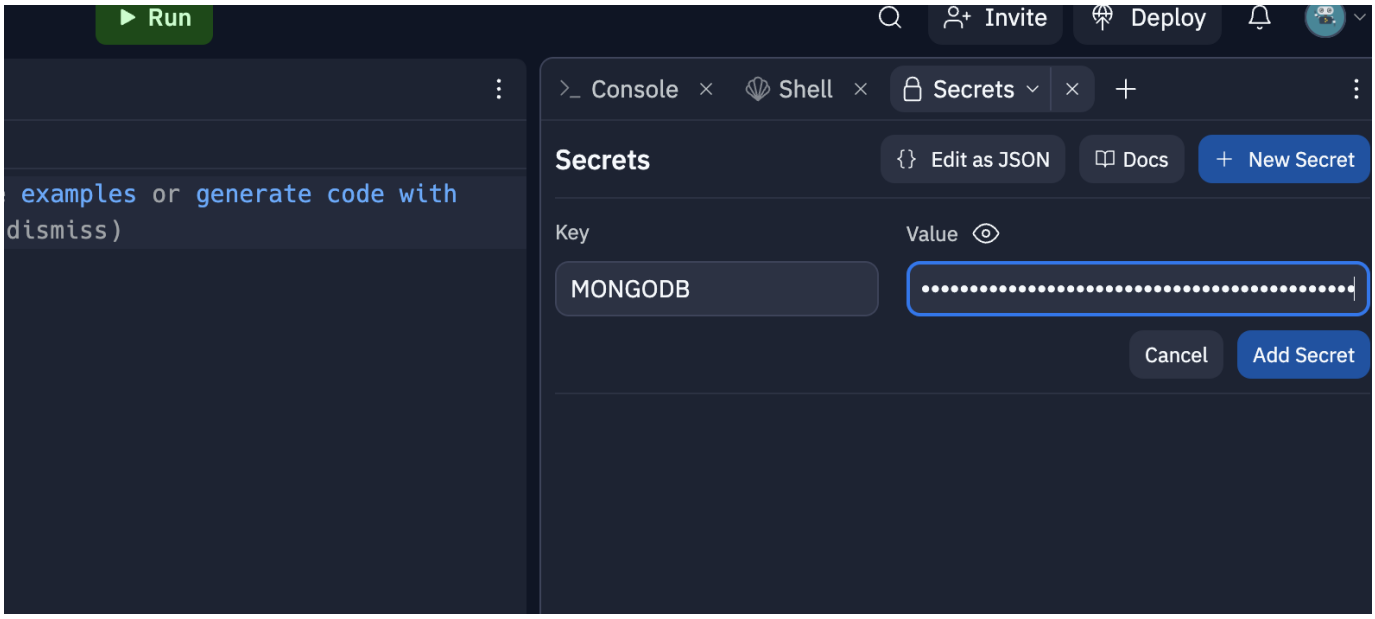
challenge

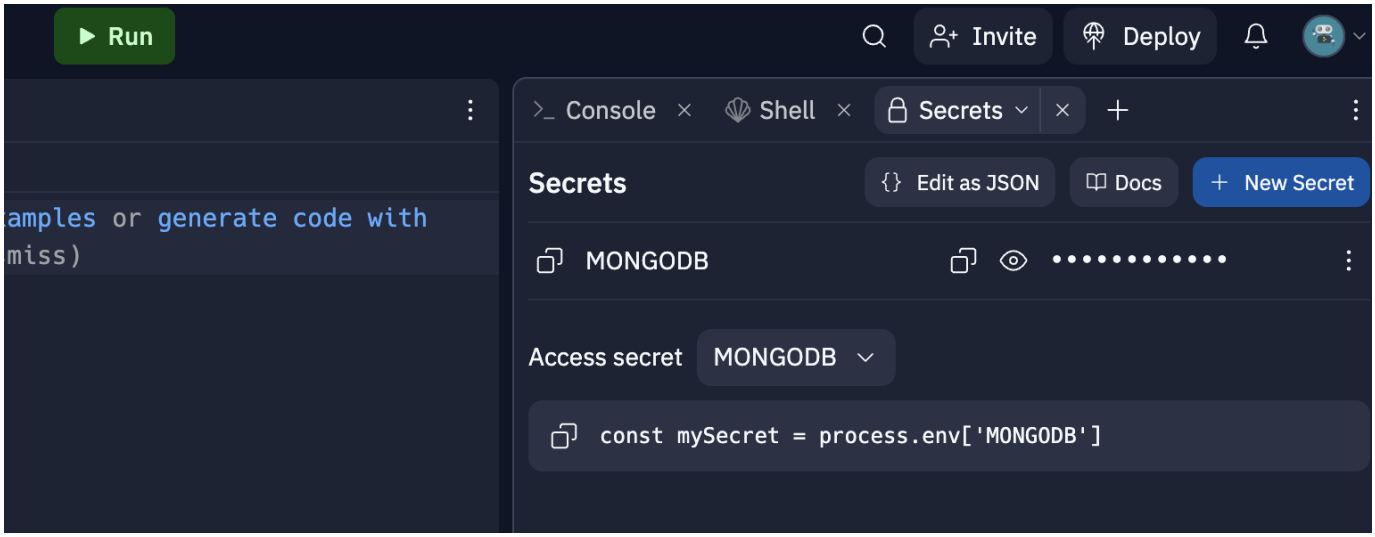
1. Go to the `Secrets` tab on the left sidebar in Replit.


BE1.1_CW
Search
Files
index.js
Packager files
package.json
Tools
Ghostwriter
Deployments
Authentication
Chat
Code Search
Console
Database
Debugger
Docs
Extensions
Git
Markdown
Packages
PostgreSQL
Secrets
Settings
Shell
Threads
Unit Tests
index.js
index.js
1 Not sure what to do?
Ghostwriter (start t



2. Create new secrets for your MongoDB connection string and any other sensitive information you need to keep secure. Make sure to replace the `<password>` with your actual password in the connection string.





ex09: connect to mongodb in replit

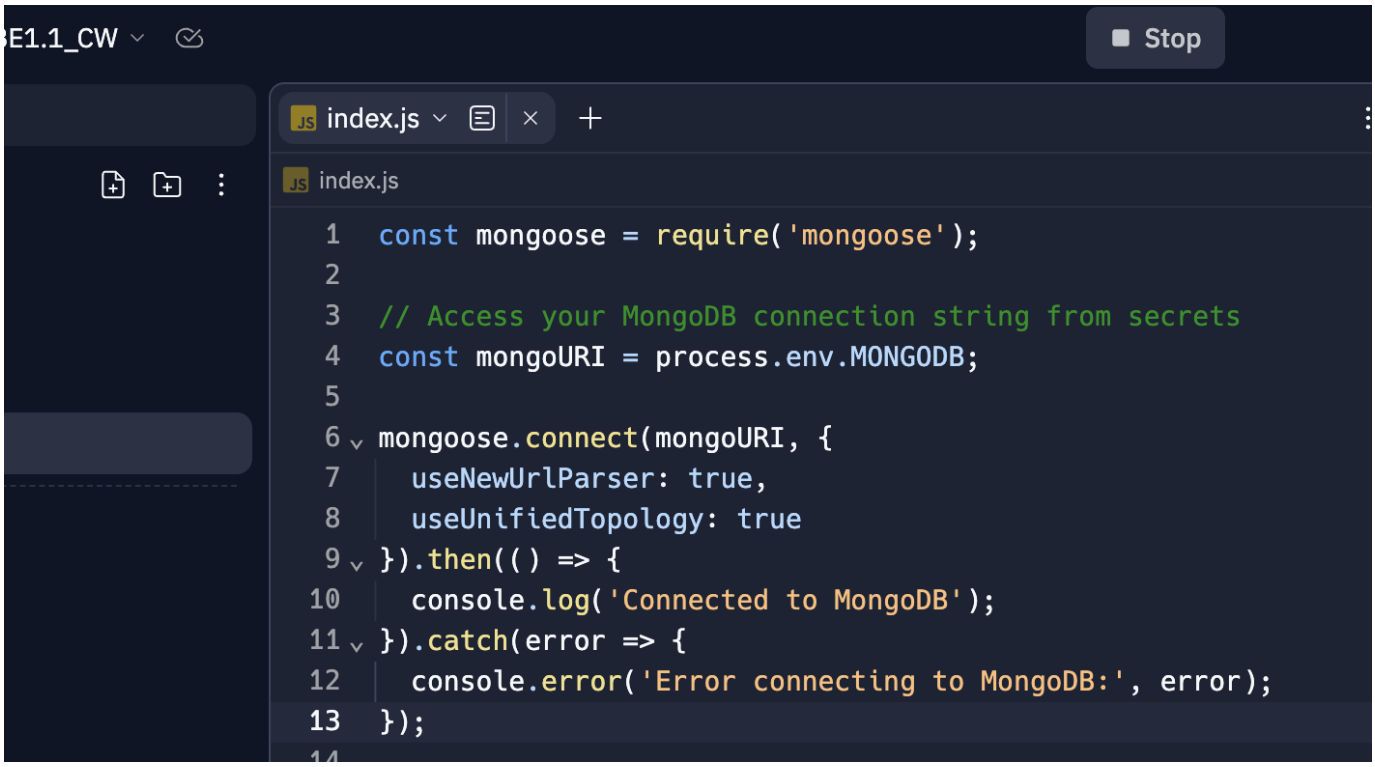
challenge

1. Let's write some Node.js code! Here's a snippet to help you connect to MongoDB in your Replit project. There are 2 ways given in the code. Use any one:

```
const mongoose = require('mongoose')
const { MongoClient } = require('mongodb')

// Access your MongoDB connection string from secrets
const mongoURI = process.env.MONGODB_URI

// Use mongoose to connect
mongoose
  .connect(mongoURI, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  })
  .then(() => {
    console.log('Connected to MongoDB')
  })
  .catch((error) => {
    console.error('Error connecting to MongoDB:', error)
  })

// **Or Use MongoClient to connect (alternative)**
const client = new MongoClient(mongoURI)
client
  .connect()
  .then(() => {
    console.log('Connected to MongoDB using MongoClient')
  })
  .catch((error) => {
    console.error('Error connecting to MongoDB using MongoClient:', error)
  })
```

COPY

2. Replace `process.env.MONGODB_URI` with the actual secret name you've used for your MongoDB connection string. See line number 5 in image below.



introducing mongoose: basic operations with the "Car" model

ex01: add mongoose to your project

challenge

1. Open your Replit project and make sure you're connected to your MongoDB Atlas cluster.
2. Install Mongoose by running `npm install mongoose` in the Replit terminal.
3. In your index.js code, require Mongoose at the top:

solution

```
const mongoose = require('mongoose')
```

COPY

ex02: introduce the "Car" model

challenge

1. Let's start by creating a basic Mongoose model. In the same file (index.js), add the following code below your Mongoose connection:

solution

```
const carSchema = new mongoose.Schema({
  make: String,
  model: String,
  year: Number,
})

const Car = mongoose.model('Car', carSchema)
```

COPY

ex03: define functions for mongoose operations

challenge

Define a named function to add data to the "Car" model:

solution

```
async function addCarData() {
  const newCar = new Car({
    make: 'Toyota',
    model: 'Corolla',
    year: 2022,
  })

  try {
    const savedCar = await newCar.save()
    console.log('Car data saved successfully:', savedCar)
  } catch (error) {
    console.error('Error saving car data:', error)
  }
}

addCarData()
```

COPY

ex04: add one more data entry

challenge

Define a function to add another data entry:

solution

```
async function addAnotherCarData() {
  const anotherCar = new Car({
    make: 'Honda',
    model: 'Civic',
    year: 2023,
  })
```

```
  try {
    const savedAnotherCar = await anotherCar.save()
    console.log('Another car data saved successfully:', savedAnotherCar)
  } catch (error) {
    console.error('Error saving another car data:', error)
  }
}

addAnotherCarData()
```

COPY

ex05: perform a `findAll` operation

challenge

Define a function to retrieve all car data from the database:

solution

```
async function findAllCars() {
  try {
    const allCars = await Car.find()
    console.log('All cars in the database:', allCars)
  } catch (error) {
    console.error('Error fetching all cars:', error)
  }
}

findAllCars()
```

COPY

get ready for production

ex01: create a "models" directory

challenge

In your Replit project, create a new directory named "models" to house your Mongoose models.

ex02: move the "Car" model

challenge

Move your existing "Car" model code from the main file to a new file named `car.js` inside the "models" directory. The code in `car.js` should look like this:

solution

```
const mongoose = require('mongoose')

const carSchema = new mongoose.Schema({
  make: String,
  model: String,
  year: Number,
})

const Car = mongoose.model('Car', carSchema)

module.exports = Car
```

COPY

ex03: create a "db.js" file for database connection

challenge

In the root of your project, create a new file named `db.js` to handle the database connection. The code in `db.js` should look like this:

solution

```
const mongoose = require('mongoose')

// Access your MongoDB connection string from secrets
const mongoURI = process.env.MONGODB_URI

mongoose
  .connect(mongoURI, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  })
  .then(() => {
    console.log('Connected to MongoDB')
  })
  .catch((error) => {
    console.error('Error connecting to MongoDB:', error)
  })
```

COPY

ex04: update your main file

challenge

1. In your main file (the one where you were working earlier), require the `db.js` file at the top:
2. Next, require the "Car" model at the top as well:

solution

```
require('./db')
const Car = require('./models/car')
```

COPY

ex05: test the database connection

challenge

1. Open your Replit project's console and run your main file.
2. If the connection is successful, you'll see the message "Connected to MongoDB."