educación IT
CODE YOUR FUTURE

Bootcamp
Full Stack Engineer

Fase 3 - Desarrollo Back End

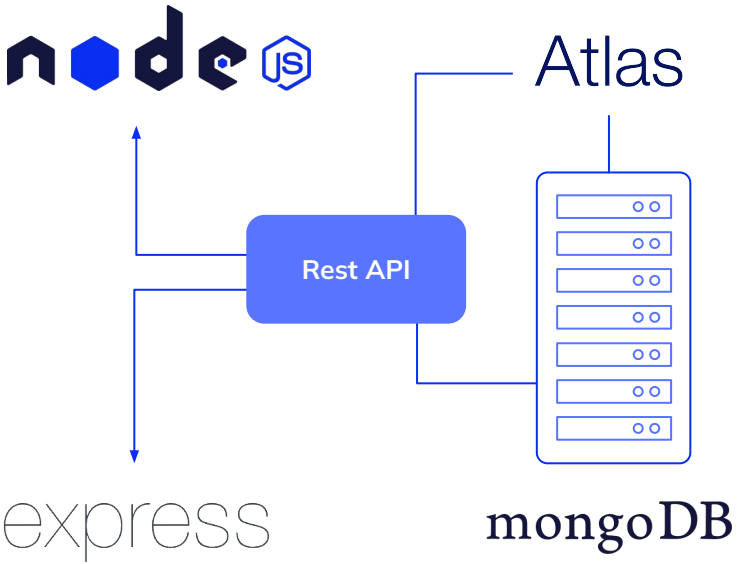
Módulo 35

educación IT
CODE YOUR FUTURE

MongoDB Atlas CRUD con Mongoose y Express

Cómo conectar MongoDB a Node.js con Mongoose

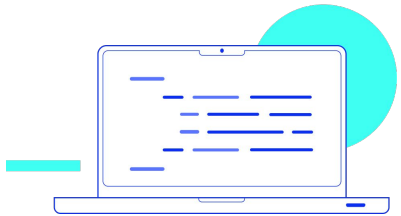
MongoDB Atlas es una de las bases de datos noSQL más utilizadas por los desarrolladores en la actualidad. Las bases de datos noSQL permiten **enviar y recuperar datos como documentos JSON, en lugar de objetos SQL**. Para trabajar con MongoDB en una aplicación Node.js, podemos usar **Mongoose**.



Prerrequisitos

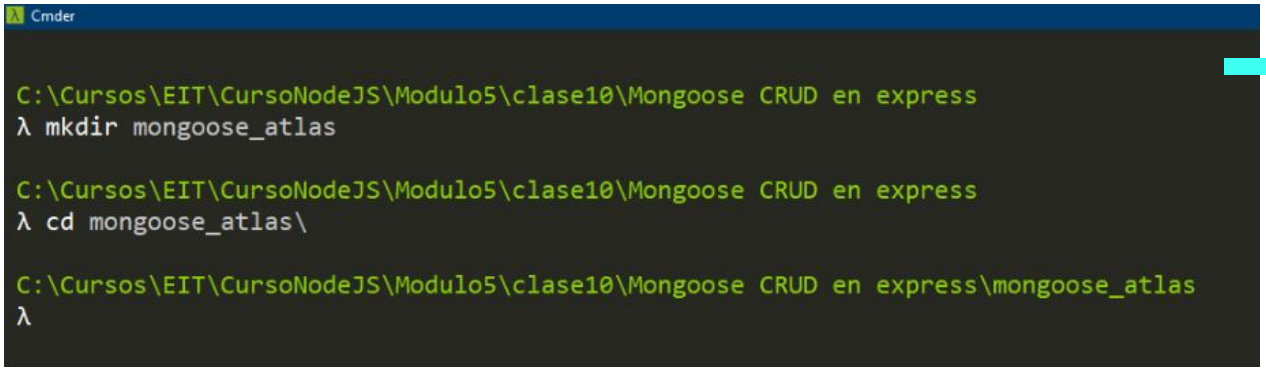
Antes de continuar, necesitaremos lo siguiente:

- Node.js **instalado en su máquina**.
- Un servicio de **MongoDB Atlas en operación**.
- Un servidor **HTTP basado en Express.js**.

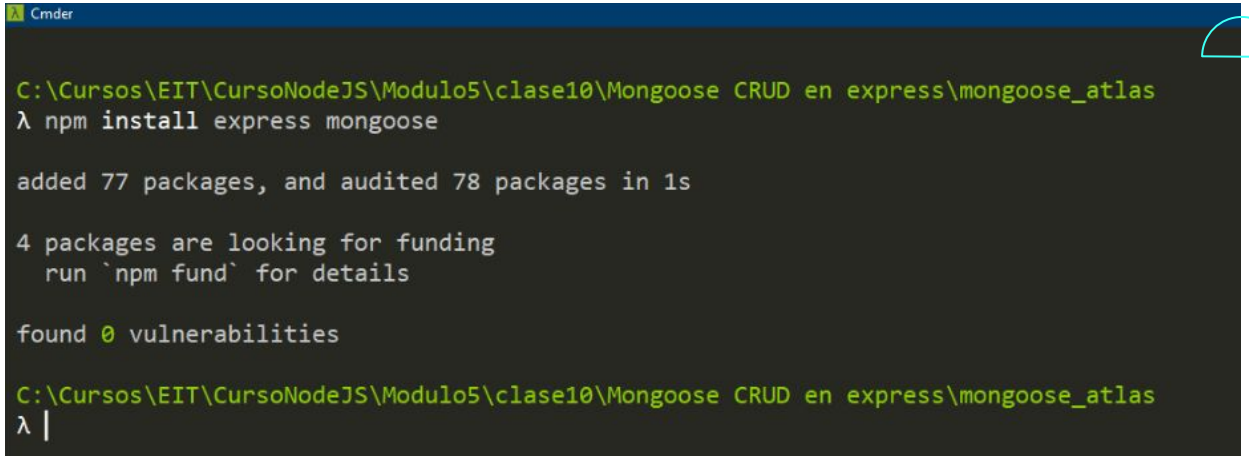


Paso 1: instalación de Mongoose en un entorno Node.js

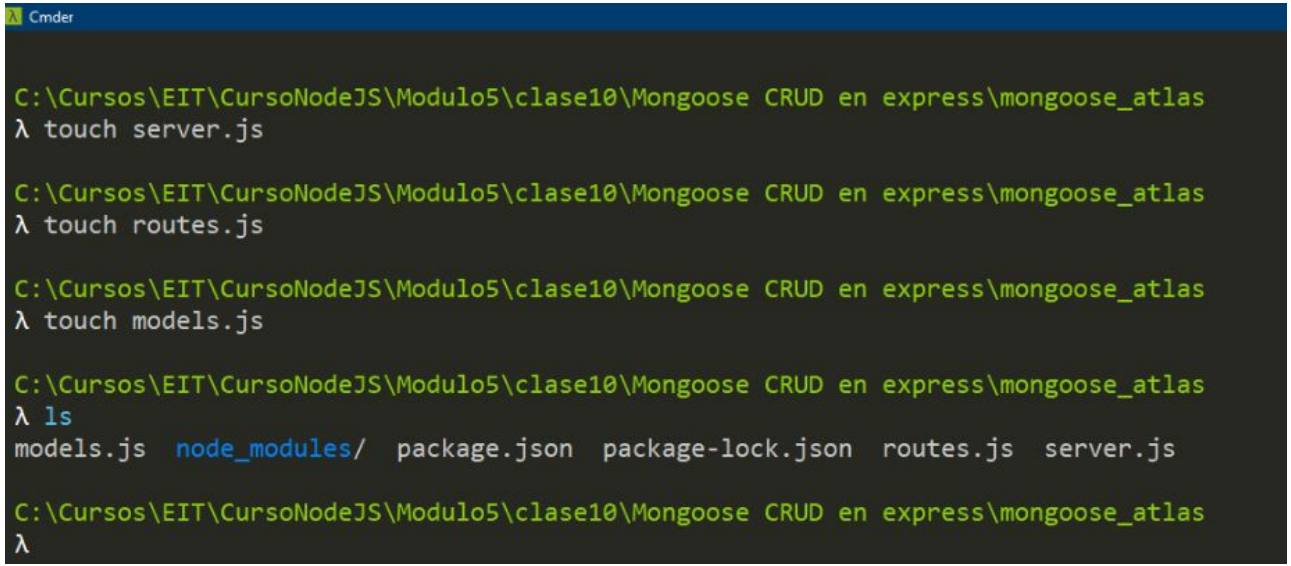
1. Cree y navegue a **una nueva carpeta ejecutando los siguientes comandos en una terminal**:



2. Instale **Express y Mongoose** ejecutando el siguiente comando **en una terminal**:



3. Luego creamos los archivos **server.js**, **routes.js** y **models.js** necesarios para este proyecto:

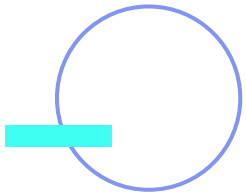


Paso 2: creación de la conexión

Cree un nuevo archivo **server.js** para iniciar nuestro servidor **Express.js**. Cargue Mongoose y Express y añada el siguiente código a **server.js**:

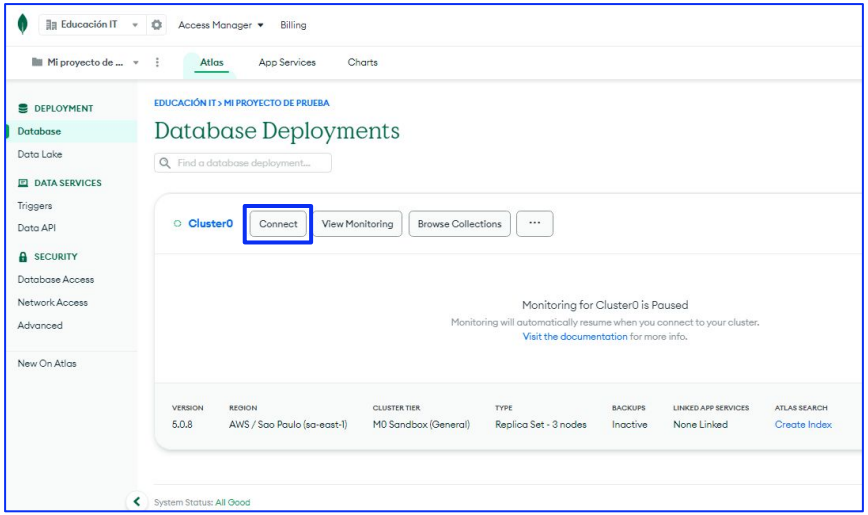
server.js

```
const express = require("express");
const mongoose = require("mongoose");
const router = require("./routes")

const app = express();

app.use(express.json());
```

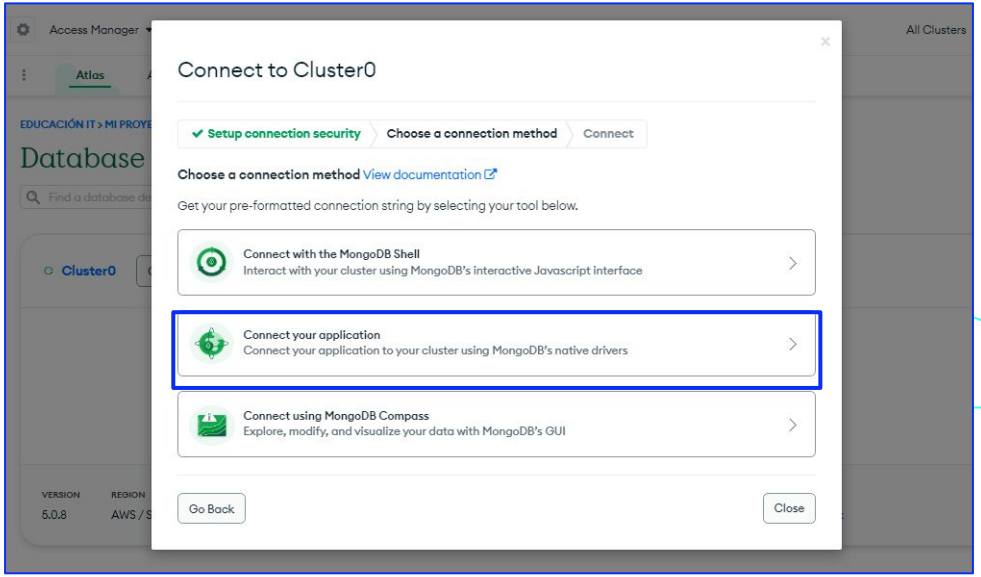


Cómo crear una conexión a MongoDB Atlas

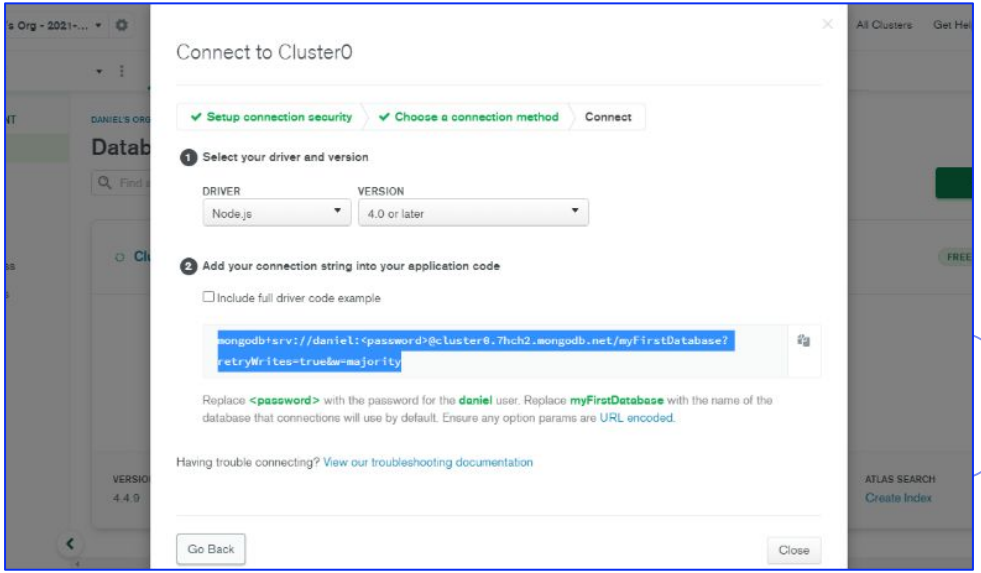
1. **Ingrese en MongoDB Atlas** y abra la pestaña ***Cluster***. Luego, haga clic en ***Connect***.



2. Seleccione ***Connect your application*** y elija ***Node.js*** para el controlador.



3. Copie la **cadena de conexión.**



4. Luego, nos conectamos a una instancia de MongoDB Atlas usando la función **`mongoose.connect()`**. Creamos **las siguientes variables y reemplazamos sus valores usando sus credenciales reales**. Veamos la imagen de la derecha.

> **Nota:** es importante tener en cuenta que la variable de `cluster` son los valores que aparecen entre **`@`** y **`.mongodb.net`**. En este caso, la variable de clúster es **`cluster0.7hch2`**.

server.js

```
const username = "daniel";
const password = "xxxxxxxxx";
const cluster = "cluster0.7hch2";
const dbname = "myFirstDatabase";

mongoose.connect(

`mongodb+srv://${username}:${password}@${clust
er}.mongodb.net/${dbname}?retryWrites=true&w=m
ajority`,
  {
    useNewUrlParser: true,
    useUnifiedTopology: true
  }
);
```

5. Para asegurarse de que su conexión fue exitosa, agregue el siguiente código justo debajo de su **`mongoose.connect()`**:

server.js

```
const db = mongoose.connection;
db.on("error", console.error.bind(console,
"connection error: "));
db.once("open", function () {
  console.log("MongoDB Atlas connected");
});
```

6. Luego, configure la aplicación para escuchar el puerto **8080**, junto a los eventos de **error**, **salida y excepciones**:

server.js

```
app.use('/api', router);

const PORT = 8080
const server = app.listen(PORT, () => {
  console.log(`Server is running at port ${server.address().port}`);
});
server.on('error', error => console.log(`Error en servidor: ${error.message}`))

process.on('exit', () => {
    console.log('database disconnect')
    mongoose.disconnect()
})
```

...

```
process.on('SIGINT', () => {
    console.log('control-C detectado')
    process.exit(0)
})
process.on('uncaughtException', err => {
    console.error('Hay una excepción:', err.message)
})
```

Nota: Crearemos el Router más tarde.

Paso 3: creación del esquema

1. Ahora definamos un esquema de colección para nuestra aplicación sobre el archivo **models.js** y agregamos **el siguiente código**:

models.js

```
const mongoose = require("mongoose");

const UserSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
  },
  age: {
    type: Number,
    default: 0,
  },
});
```

...

```
const User = mongoose.model("User",
UserSchema);

module.exports = User;
```

2. Creamos un esquema **`UserSchema`** usando el método **`mongoose.Schema()`**. Este recopila los campos **name** y **age** enviados desde la solicitud. Luego exportamos el esquema usando **las últimas 2 líneas**.

Paso 4: creación del endpoint POST

1. Crea un nuevo archivo **routes.js**. Este archivo **define los *endpoints* de nuestra aplicación**.
2. Cargue Express y el esquema que creamos en el **paso 3** agregando **el siguiente código**:

routes.js

```
const express = require("express");
const userModel = require("./models");

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

3. Luego cree el *endpoint* **POST** agregando **el siguiente código**:

routes.js

```
router.post("/add_user", async (request,
response) => {
    const user = new userModel(request.body);

    try {
        await user.save();
        response.send(user);
    } catch (error) {
        response.status(500).send(error);
        throw error
    }
});
```

4. Creamos una ruta **`/add_user`** para agregar **un nuevo usuario a la base de datos**.
5. Inicializamos el contenido que se guardará en la base de datos usando la línea

```
const user = new userModel(request.body);
```

6. Luego usamos un bloque **`try/catch`** para guardar el objeto en la base de datos usando el método **`.save()`**.

Paso 5: creación del endpoint GET

1. Agregue las siguientes líneas de código al archivo **routes.js**:

routes.js

```
router.get("/users/:id?", async (request, response) => {
    let {id} = request.params
    try {
        const users = await userModel.find(id? {_id:id}:{});
        response.send(users);
    } catch (error) {
        response.status(500).send(error.message);
        throw error
    }
});
```

2. Creamos una ruta **`/users`** para recuperar todos los usuarios guardados usando la ruta **`/add_user`**. Recopilamos estos usuarios de la base de datos utilizando el método **`.find()`**. Luego usamos un bloque **`try/catch`** para *enviar* a los usuarios a este *endpoint*.

Paso 6: creación del endpoint PUT

1. Cree el endpoint **PUT** agregando el siguiente código:

routes.js

```
router.put("/update_user/:id", async (request, response) => {
    let {id} = request.params
    try {
        console.log(request.body)
        let rta = await userModel.updateOne({_id:id}, {$set: request.body});
        response.send(rta);
    } catch (error) {
        response.status(500).send(error);
        throw error
    }
});
```

2. Creamos una ruta **`/update_user`** para actualizar un usuario por su ID en la base de datos.
3. Luego usamos un bloque **`try/catch`** para actualizar el objeto en la base de datos usando el método **`.updateOne()`**.

Paso 7: creación del endpoint DELETE

1. Cree el endpoint **`DELETE`** agregando el siguiente código:

   routes.js

```
router.delete("/delete_user/:id", async (request, response) => {
    let {id} = request.params
    try {
        let rta = await userModel.deleteOne({_id:id});
        response.send(rta);
    } catch (error) {
        response.status(500).send(error);
        throw error
    }
});
```

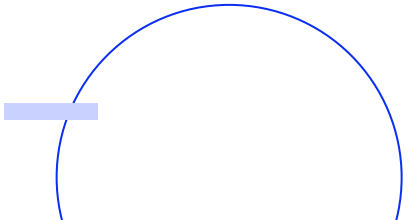
2. Creamos una ruta **`/delete_user`** para eliminar un usuario por su ID en la base de datos.
3. Luego usamos un bloque **`try/catch`** para borrar el objeto en la base de datos usando el método **`.deleteOne()`**.
4. Finalmente, exporte estos *endpoints* agregando la línea a continuación:

routes.js

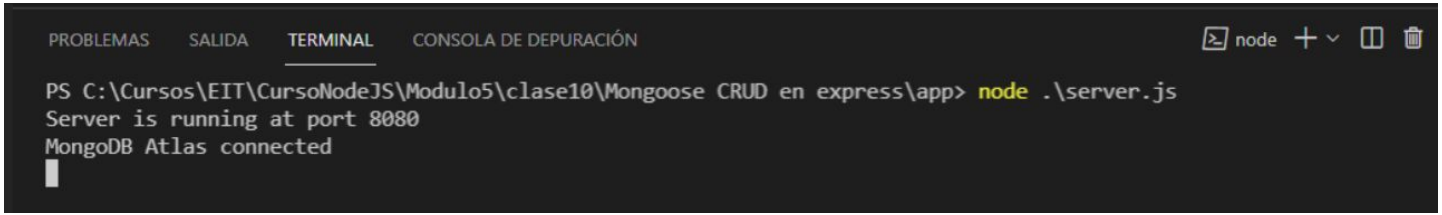
```
module.exports = router;
```

5. En este punto, **su aplicación está lista**. Sirva la aplicación ejecutando el siguiente comando:

```
node server.js
```

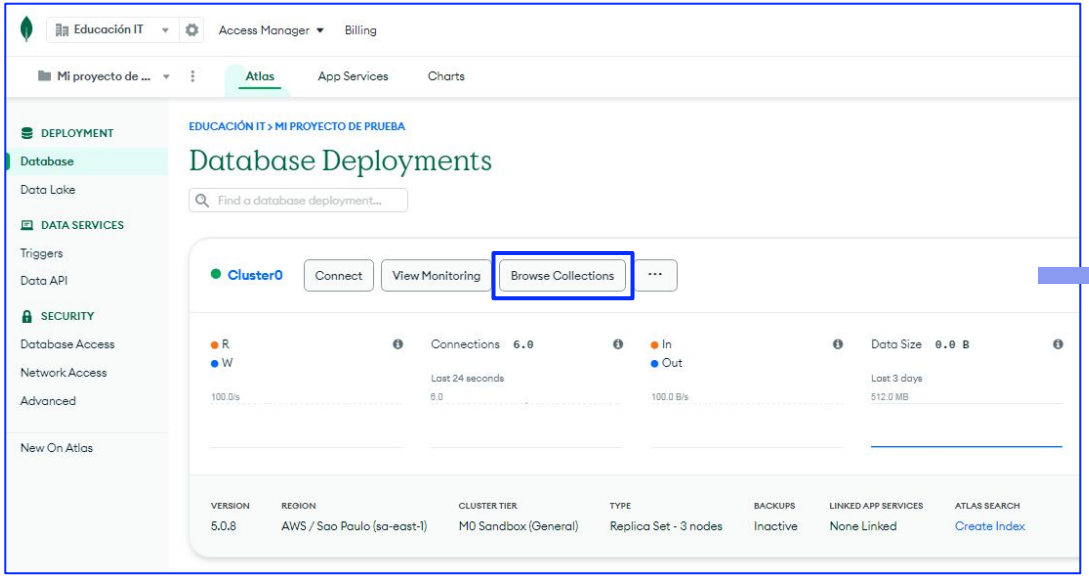


6. En la consola se verá:

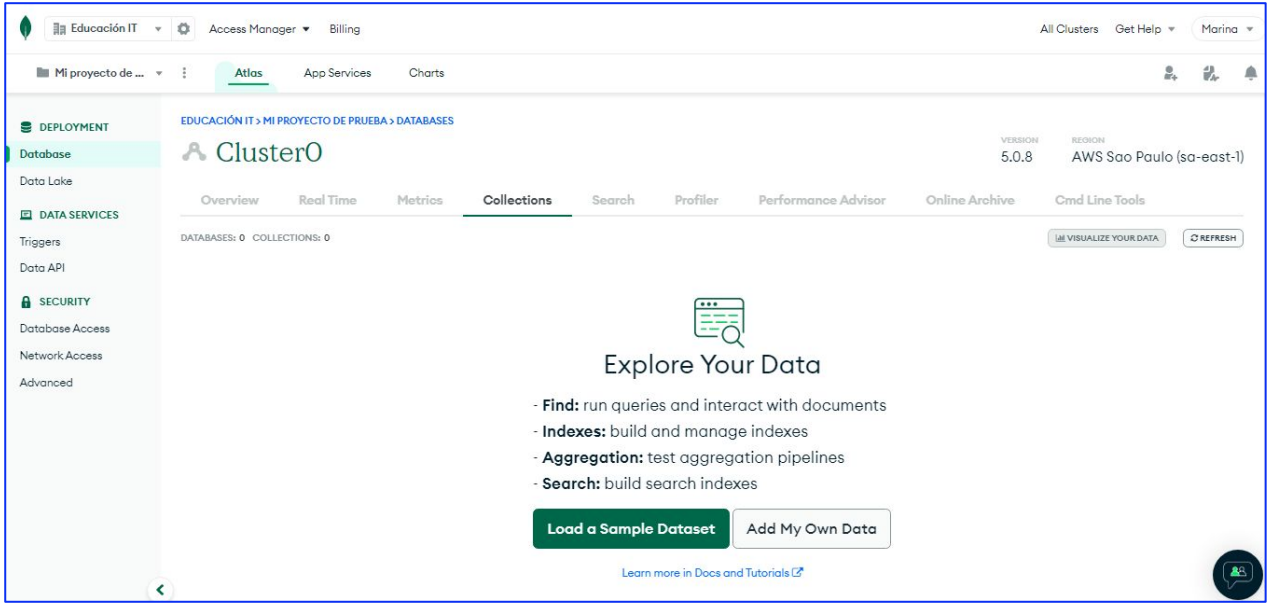


Paso 8: prueba de los endpoints

1. Abrimos **el cliente web de MongoDB Atlas** y vamos al menú ***Browse Collections***.



2. En este punto vemos un servidor de base de datos **sin ninguna información**:

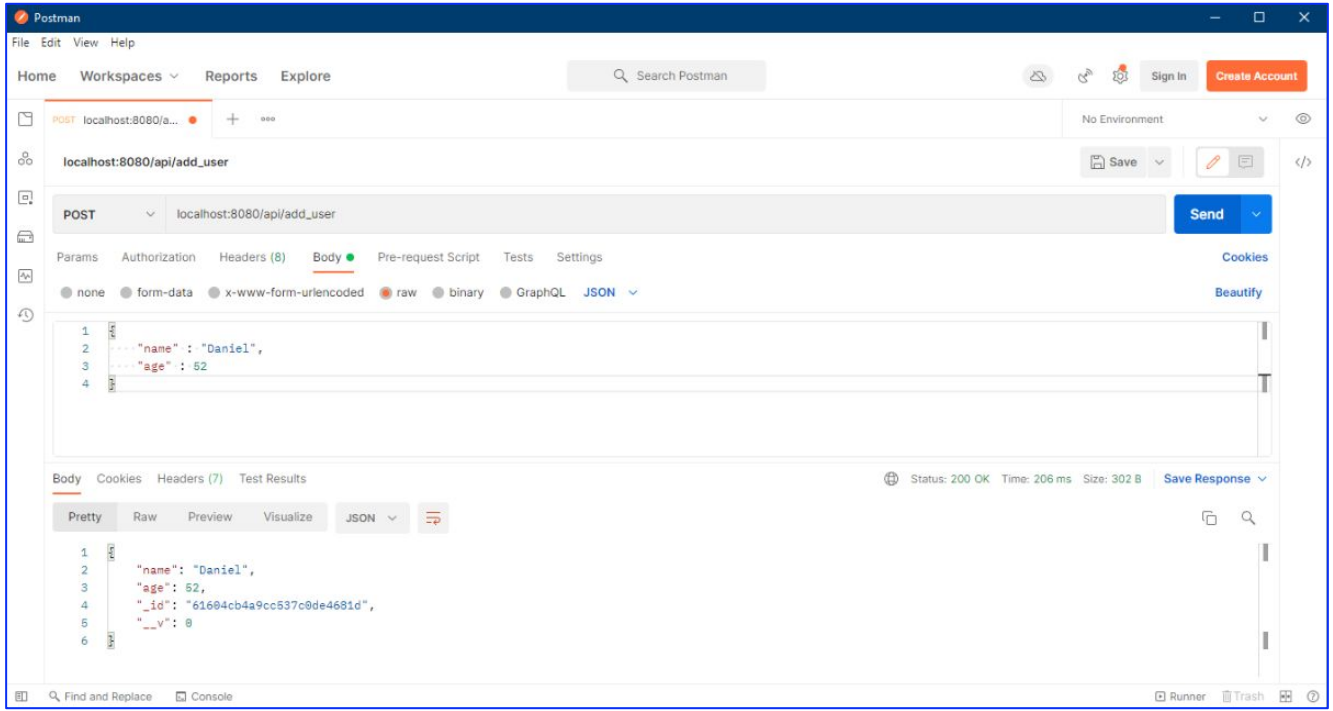


3. Ahora, probemos los *endpoints* **`GET`, `POST`, `PUT`** y **`DELETE`** que creamos anteriormente.

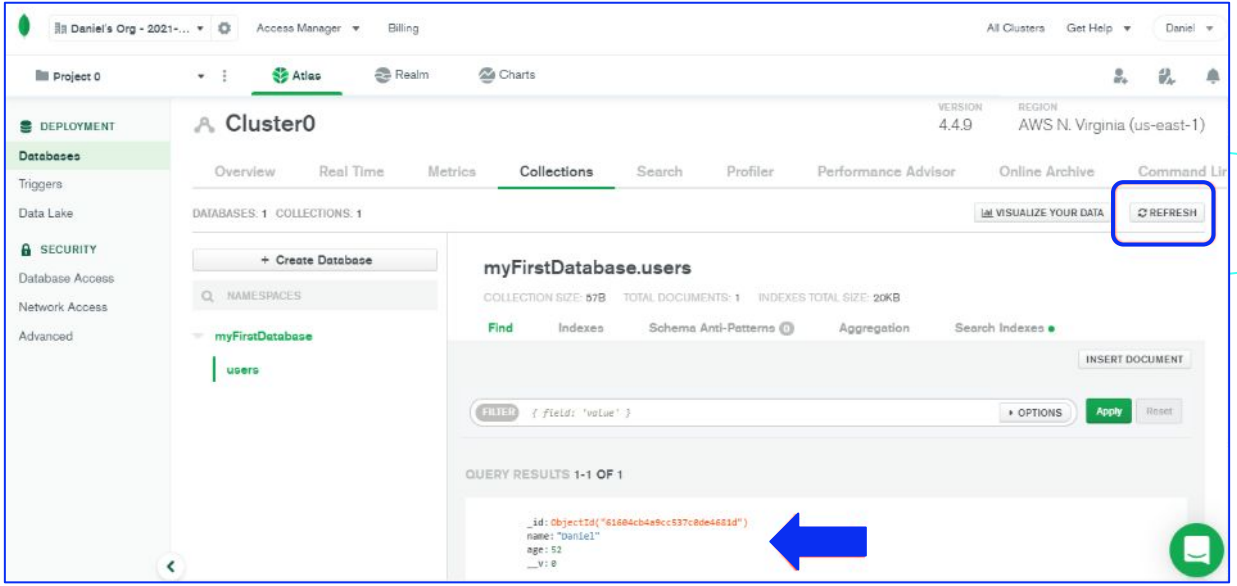
4. Abra **Postman** y realice una **solicitud `POST`** al *endpoint* http://localhost:8080/api/add_user agregando en **`body` / `raw` / `json`** el siguiente documento:

```
{
    "name" : "Daniel",
    "age" : 52
}
```

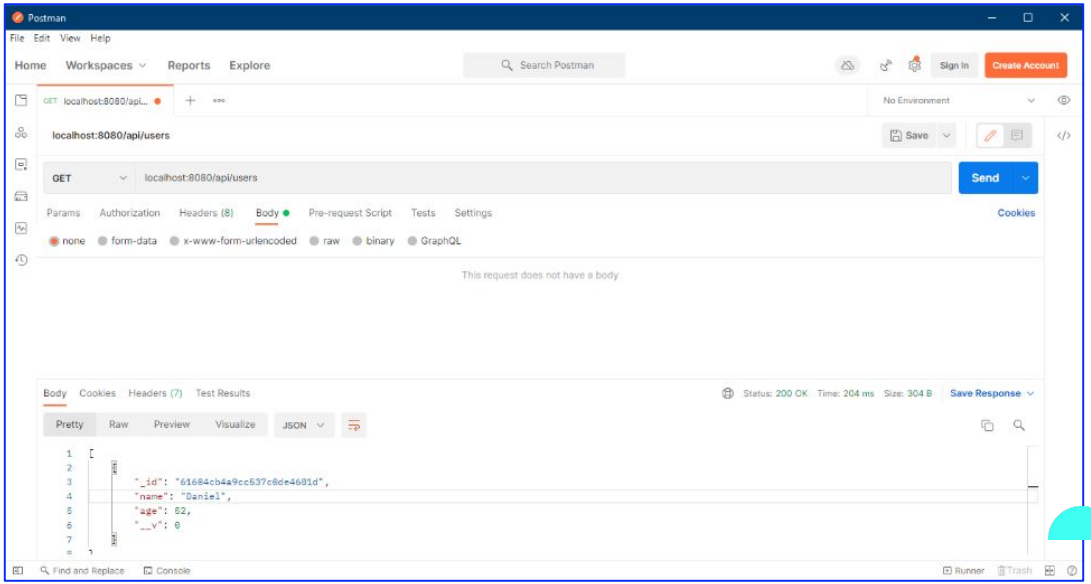
Veamos la imagen que ilustra la explicación, en la siguiente diapositiva.


Postman
File Edit View Help
Home Workspaces Reports Explore
Search Postman
Sign In
Create Account
POST localhost:8080/a...
No Environment
localhost:8080/api/add_user
Save
POST
localhost:8080/api/add_user
Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
1 {
2 "name" : "Daniel",
3 "age" : 52
4 }
Body Cookies Headers (7) Test Results
Status: 200 OK Time: 206 ms Size: 302 B
Save Response
Pretty Raw Preview Visualize JSON
1 {
2 "name": "Daniel",
3 "age": 52,
4 "_id": "61604cb4a9cc537c0de4681d",
5 "__v": 0
6 }
Find and Replace
Console
Runner
Trash

5. Si vamos a la consola de **MongoDB Atlas** y accionamos el botón ***Refresh*** veremos el documento subido.

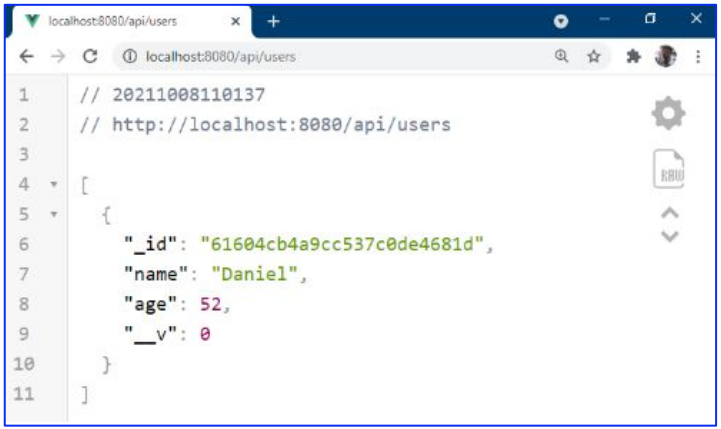


6. Realizamos ahora una **solicitud GET** al *endpoint* http://localhost:3000/api/users. En **body** marcamos **none** porque no vamos a enviar ningún dato.



7. El endpoint devuelve una lista de **todos los users agregados a la base de datos**.

8. Si lo vemos en el navegador se verá como la imagen a continuación:



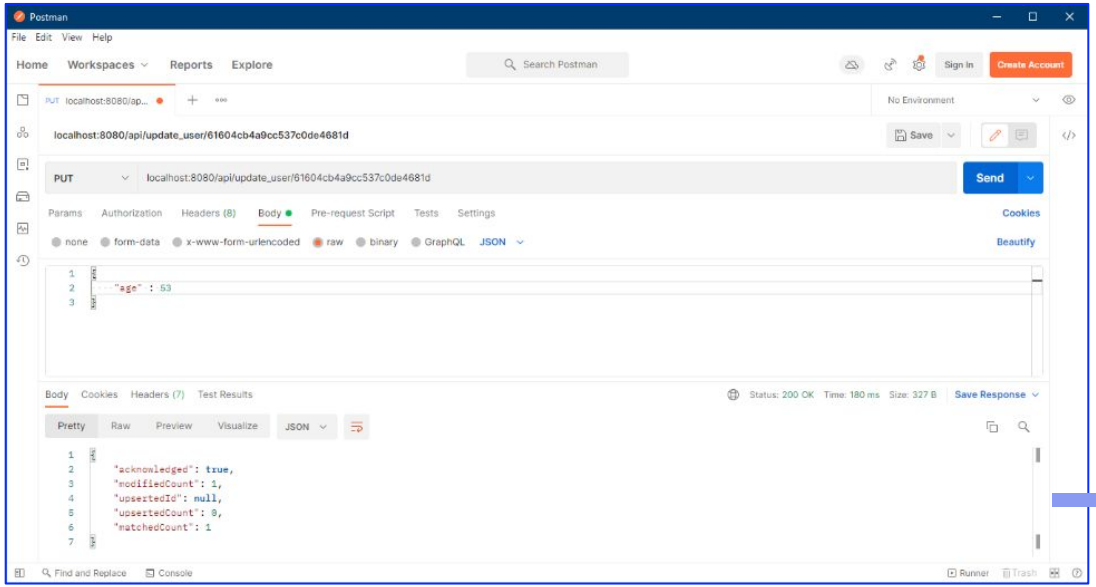
9. Utilizando **Postman**, vamos a realizar una solicitud PUT al endpoint http://localhost:8080/api/update_user.

10. Tenemos que agregarle al endpoint el ID del documento a actualizar: **`61604cb4a9cc537c0de4681d`** (en este caso).

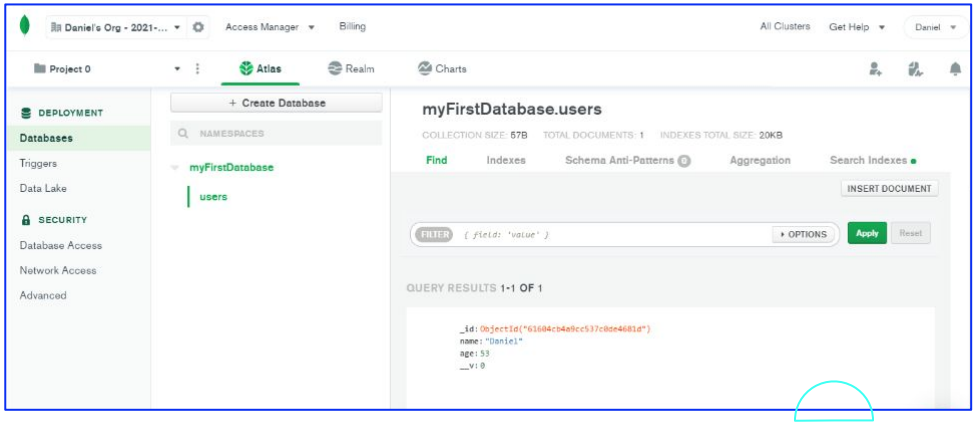
11. Definimos en **`body`** / **`raw`** / **`json`** el siguiente objeto para sólo actualizar la edad:

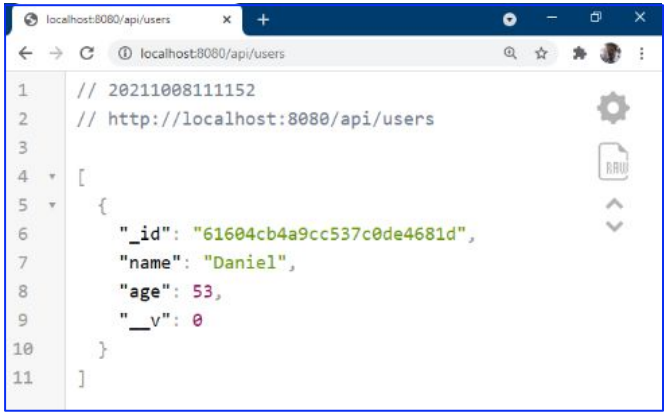
```
{
    "age" : 53
}
```

12. La actualización puede ser **parcial** (algunos campos del documento) o **total**.



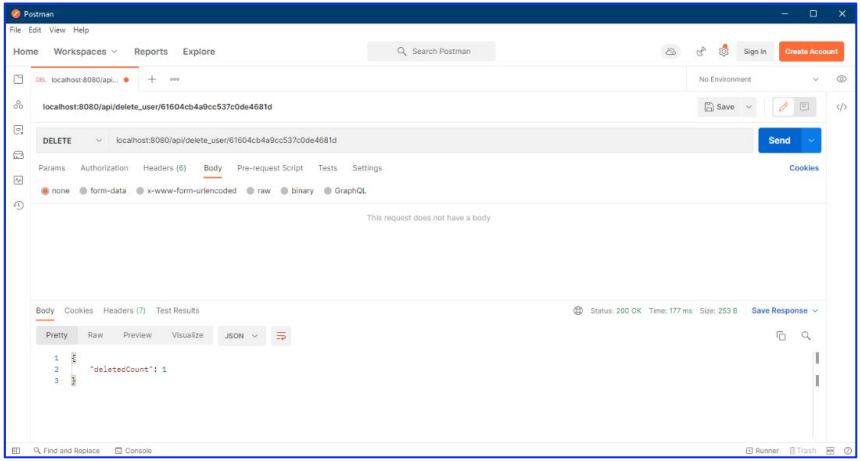
13. En la consola de MongoDB Atlas se verá reflejada **la actualización del documento al igual que en el navegador**.



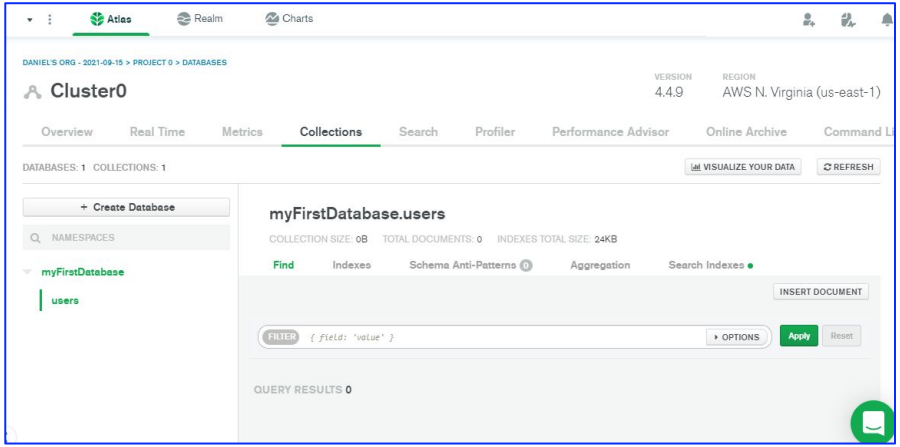


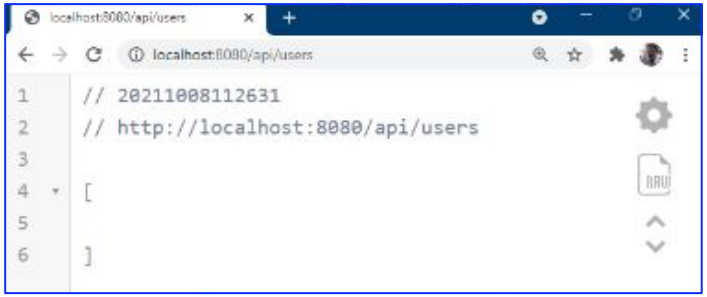
14. Por último, realizaremos una **solicitud DELETE** a través de **Postman** al *endpoint* http://localhost:8080/api/delete_user. Tenemos que agregarle al *endpoint* el id del documento a eliminar: **61604cb4a9cc537c0de4681d** (en este caso).

    En **body** marcamos **none** porque no vamos a enviar ningún dato.



En la consola de MongoDB Atlas se verá reflejada **la eliminación del documento al igual que en el navegador**:





educación IT
CODE YOUR FUTURE

¡Sigamos trabajando!