

Bootcamp Full Stack Engineer

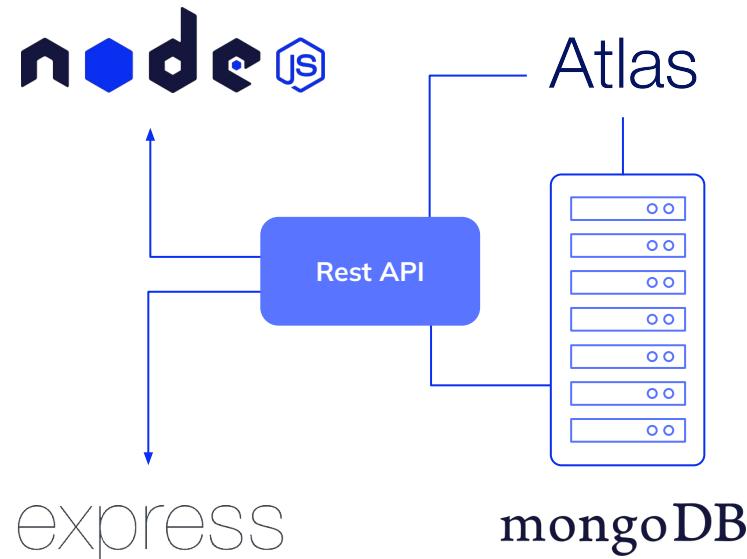
Fase 3 - Desarrollo Back End

Módulo 35

MongoDB Atlas CRUD con Mongoose y Express

Cómo conectar MongoDB a Node.js con Mongoose

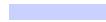
MongoDB Atlas es una de las bases de datos noSQL más utilizadas por los desarrolladores en la actualidad. Las bases de datos noSQL permiten **enviar y recuperar datos como documentos JSON, en lugar de objetos SQL**. Para trabajar con MongoDB en una aplicación Node.js, podemos usar **Mongoose**.



Prerrequisitos

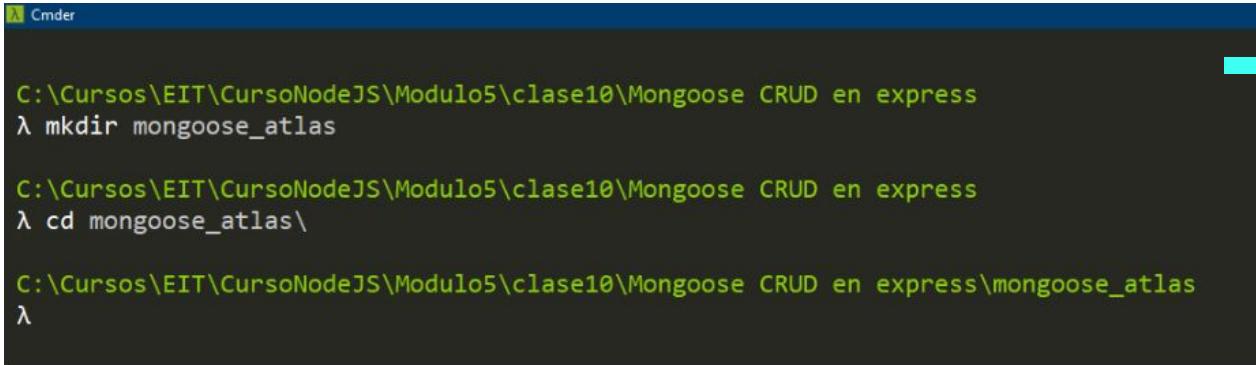
Antes de continuar, necesitaremos lo siguiente:

- 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:

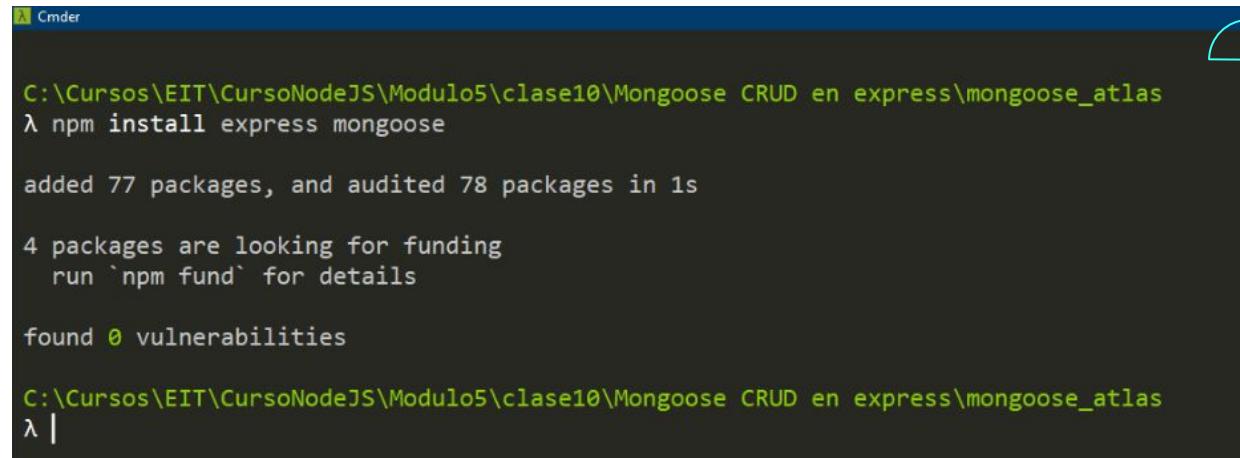


```
C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\Mongoose CRUD en express
λ mkdir mongoose_atlas

C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\Mongoose CRUD en express
λ cd mongoose_atlas\

C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\Mongoose CRUD en express\mongoose_atlas
λ
```

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



```
C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\Mongoose CRUD en express\mongoose_atlas
λ npm install express mongoose

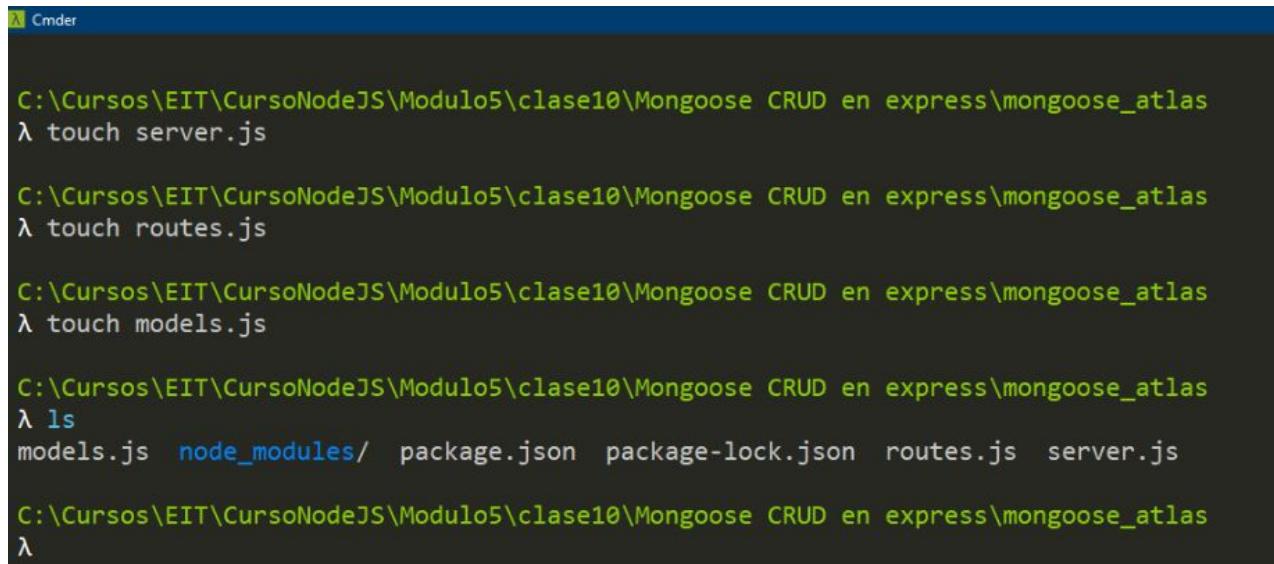
added 77 packages, and audited 78 packages in 1s

4 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\Mongoose CRUD en express\mongoose_atlas
λ |
```

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



```
C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\Mongoose CRUD en express\mongoose_atlas
λ touch server.js

C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\Mongoose CRUD en express\mongoose_atlas
λ touch routes.js

C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\Mongoose CRUD en express\mongoose_atlas
λ touch models.js

C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\Mongoose CRUD en express\mongoose_atlas
λ ls
models.js  node_modules/  package.json  package-lock.json  routes.js  server.js

C:\Cursos\EIT\CursoNodeJS\Modulo5\clase10\Mongoose CRUD en express\mongoose_atlas
λ
```

Paso 2: creación de la conexión

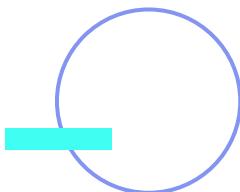
Cree un nuevo archivo **server.js** para iniciar nuestro servidor **Express.js**. Cargue Mongoose y Express y añada el siguiente código a **server.js**:

server.js

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

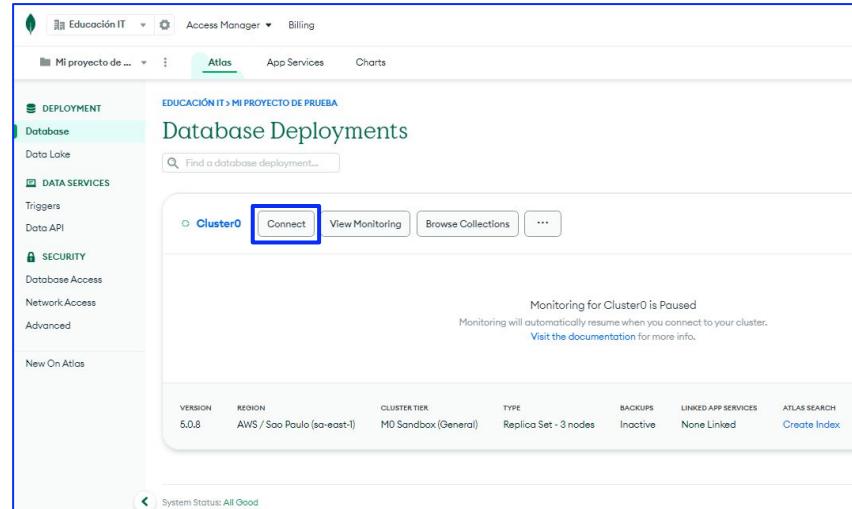
const app = express();

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

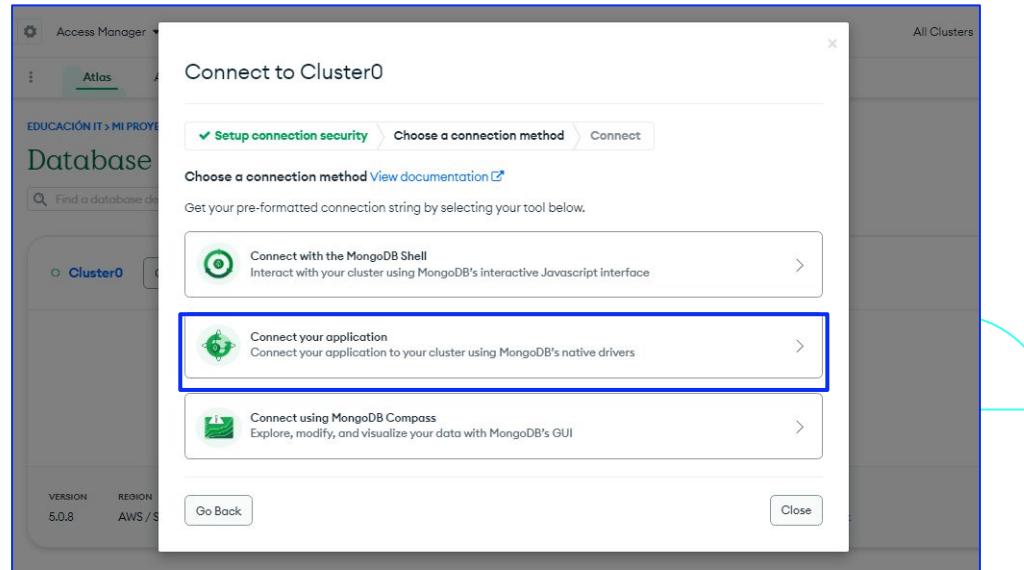


Cómo crear una conexión a MongoDB Atlas

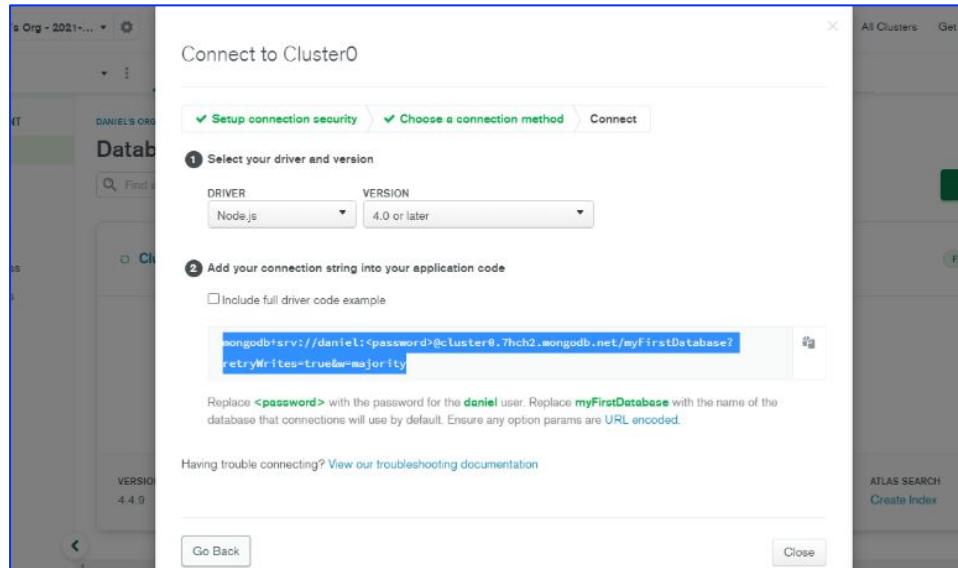
1. Ingrese en MongoDB Atlas y abra la pestaña **Cluster**. Luego, haga clic en **Connect**.



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}@${cluster}.mongodb.net/${dbname}?retryWrites=true&w=majority`,
  {
    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

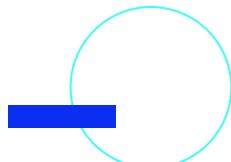
```
router.post("/add_user", async (request,
response) => {
  const user = new userModel(request.body);

  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()**.



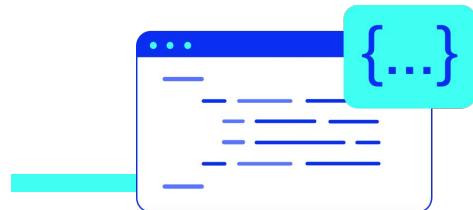
Paso 5: creación del endpoint GET

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
  }
});
```

1. 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.
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

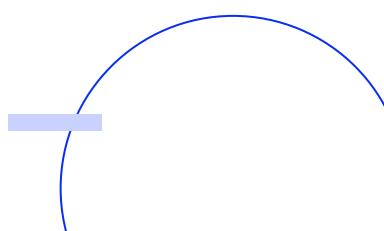
```
router.delete("/delete_user/:id", async (request, response) => {
  let {id} = request.params
  try {
    let rta = await userModel.deleteOne({_id:id});
    response.send(rta);
  } catch (error) {
    response.status(500).send(error);
    throw error
  }
});
```

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:

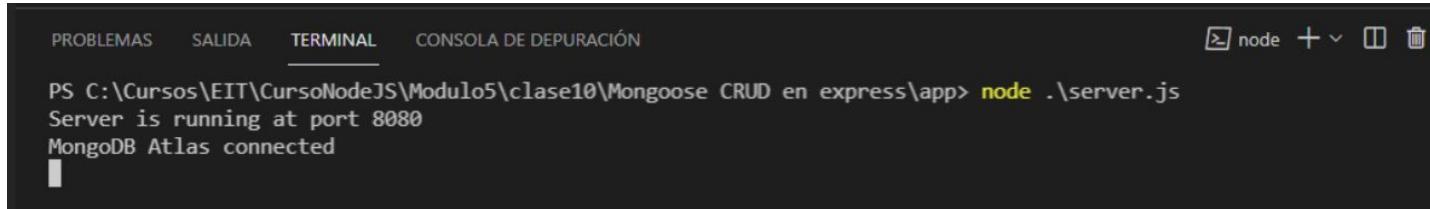
routes.js

```
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:



A screenshot of a terminal window with a dark background. At the top, there are tabs labeled 'PROBLEMAS', 'SALIDA', 'TERMINAL' (which is underlined), and 'CONSOLA DE DEPURACIÓN'. On the right side of the header, there are icons for opening a new terminal ('node'), creating a new tab ('+'), closing the tab ('✖'), and deleting the terminal ('trash'). Below the header, the terminal output is displayed in white text. It shows the command 'PS C:\Cursos\EIT\CursoNodeJS\Modules\clase10\Mongoose CRUD en express\app> node .\server.js' followed by three lines of text indicating the server is running at port 8080 and that MongoDB Atlas is connected.

```
PS C:\Cursos\EIT\CursoNodeJS\Modules\clase10\Mongoose CRUD en express\app> node .\server.js
Server is running at port 8080
MongoDB Atlas connected
```

Paso 8: prueba de los endpoints

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



The screenshot shows the MongoDB Atlas interface. On the left, there's a sidebar with sections like Deployment, Database (which is selected and highlighted in green), Data Services, Triggers, Data API, Security, Database Access, Network Access, Advanced, and New On Atlas. The main area is titled 'Database Deployments' and shows 'EDUCACIÓN IT > MI PROYECTO DE PRUEBA'. It displays a table for 'Cluster0' with metrics such as R/W connections, latency, throughput, and data size. At the top of this table, there are buttons for 'Connect', 'View Monitoring', 'Browse Collections' (which is highlighted with a blue box), and '...'. Below the table, there's a detailed row for the cluster with columns for Version (5.0.8), Region (AWS / Sao Paulo (sa-east-1)), Cluster Tier (M0 Sandbox (General)), Type (Replica Set - 3 nodes), Backups (Inactive), Linked App Services (None Linked), and Atlas Search (Create Index).

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

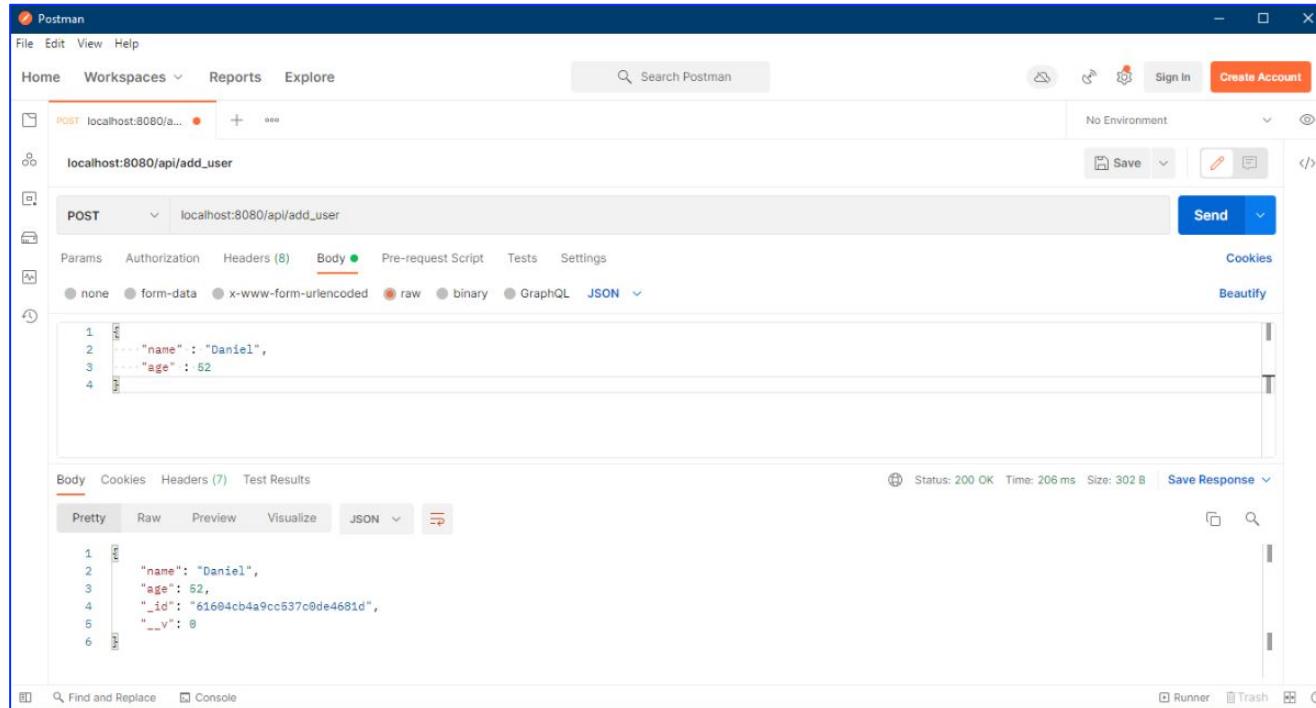
The screenshot shows the MongoDB Atlas interface for a project named "EDUCACIÓN IT > MI PROYECTO DE PRUEBA". The left sidebar is collapsed, and the main area displays the "Cluster0" configuration. The "Collections" tab is selected, showing "DATABASES: 0" and "COLLECTIONS: 0". A central "Explore Your Data" section features a magnifying glass icon and a list of actions: "Find", "Indexes", "Aggregation", and "Search". At the bottom are buttons for "Load a Sample Dataset" and "Add My Own Data". The top right corner shows the cluster version "5.0.8" and region "AWS Sao Paulo (sa-east-1)".

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.



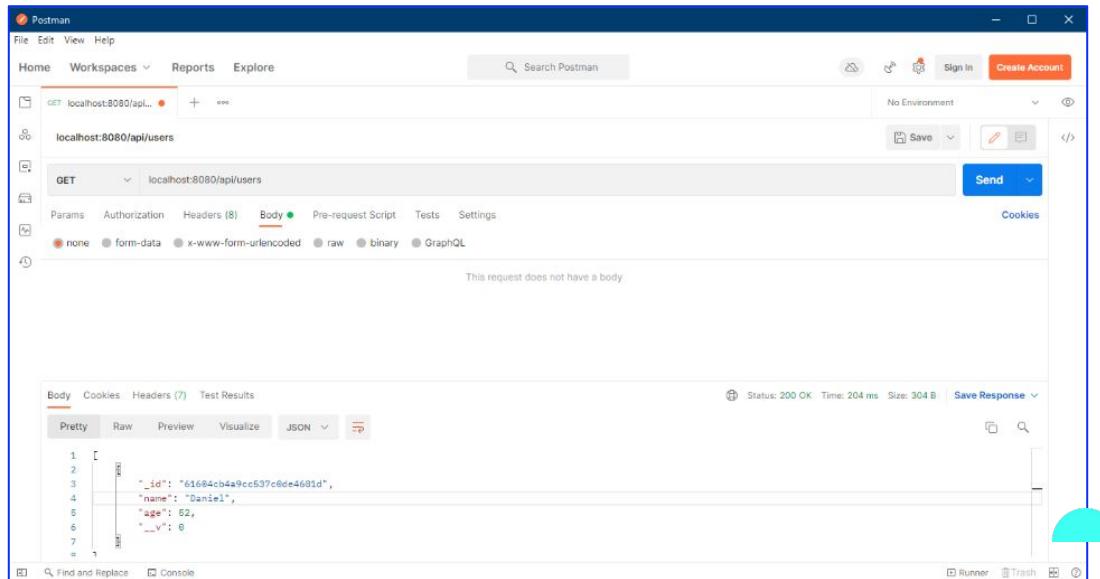


5. Si vamos a la consola de **MongoDB Atlas** y accionamos el botón **Refresh** veremos el documento subido.

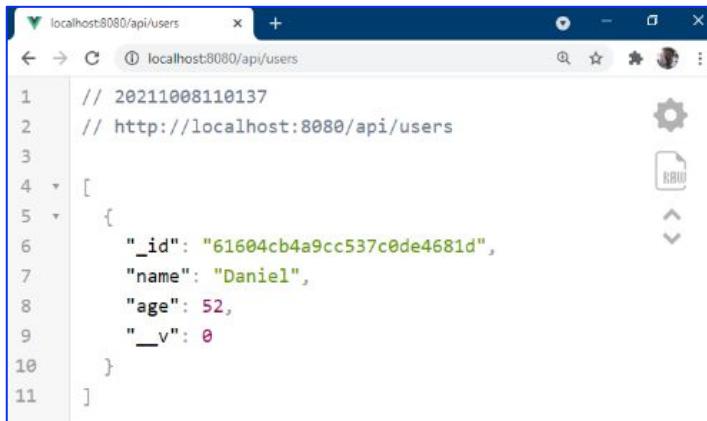
The screenshot shows the MongoDB Atlas interface. On the left, there's a sidebar with 'Project 0' selected. Under 'DEPLOYMENT', 'Databases' is highlighted, showing 'Triggers', 'Data Lake', and 'myFirstDatabase'. Under 'SECURITY', it shows 'Database Access', 'Network Access', and 'Advanced'. The main area is titled 'Cluster0' with 'VERSION 4.4.9' and 'REGION AWS N. Virginia (us-east-1)'. It has tabs for 'Overview', 'Real Time', 'Metrics', 'Collections' (which is active), 'Search', 'Profiler', 'Performance Advisor', 'Online Archive', and 'Command Line'. A 'REFRESH' button is circled in blue at the top right of the collections section. Below it, under 'myFirstDatabase.users', it shows 'COLLECTION SIZE: 578 TOTAL DOCUMENTS: 1 INDEXES TOTAL SIZE: 20KB'. There are tabs for 'Find', 'Indexes', 'Schema Anti-Patterns', 'Aggregation', and 'Search Indexes'. A 'FILTER' bar contains the query '{ field: 'value' }'. At the bottom, a 'QUERY RESULTS 1-1 OF 1' section shows a single document: '_id: ObjectId("61684cb4a9cc537c8de4681d")', 'name: "Daniel"', 'age: 52', and '__v: 0'. A large blue arrow points from the bottom right towards this document.

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:



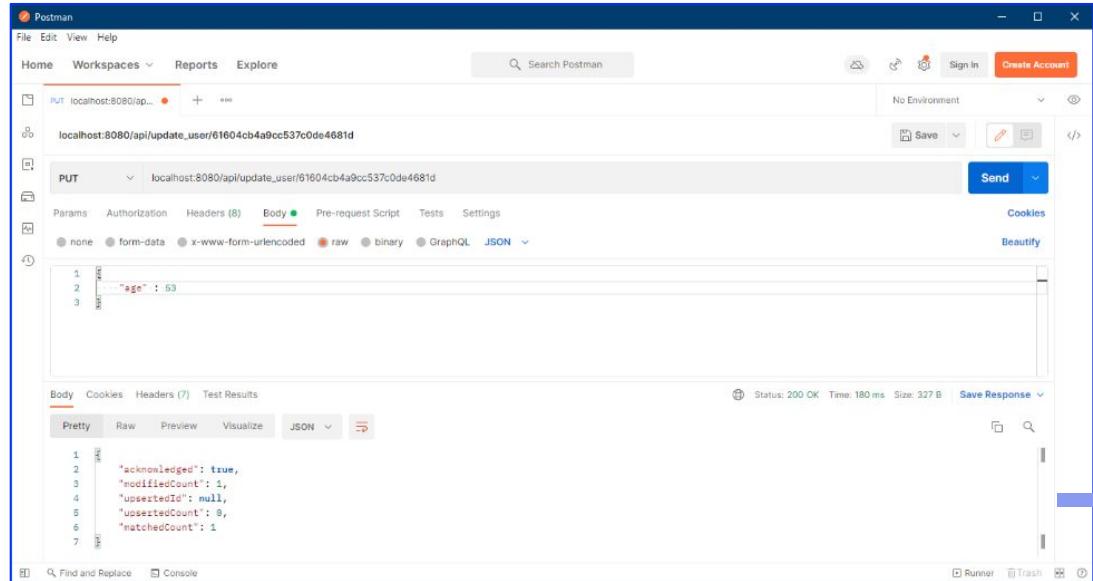
A screenshot of a browser window displaying a JSON response. The URL in the address bar is `localhost:8080/api/users`. The page content shows the following JSON data:

```
// 20211008110137
// http://localhost:8080/api/users
[
  {
    "_id": "61604cb4a9cc537c0de4681d",
    "name": "Daniel",
    "age": 52,
    "__v": 0
  }
]
```

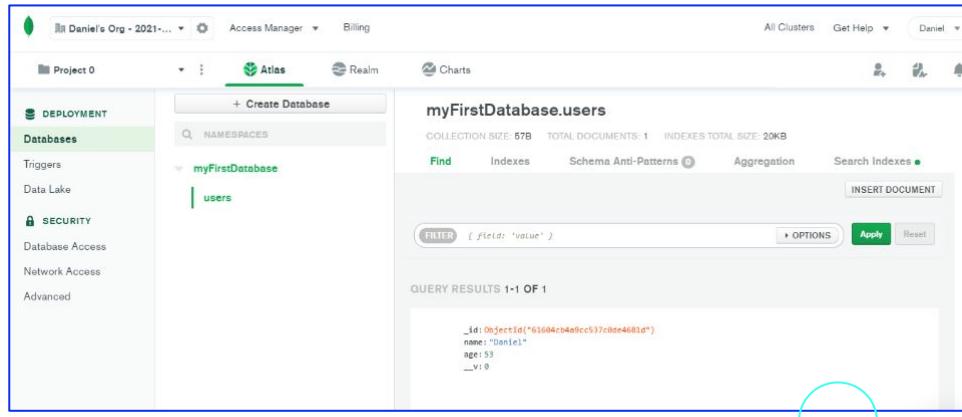
9. Utilizando **Postman**, vamos a realizar una solicitud PUT al endpoint [`http://localhost:8080/api/update_user`](http://localhost:8080/api/update_user).
10. Tenemos que agregarle al endpoint el ID del documento a actualizar: **61604cb4a9cc537c0de4681d** (en este caso).
11. Definimos en **body / raw / json** el siguiente objeto para sólo actualizar la edad:

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

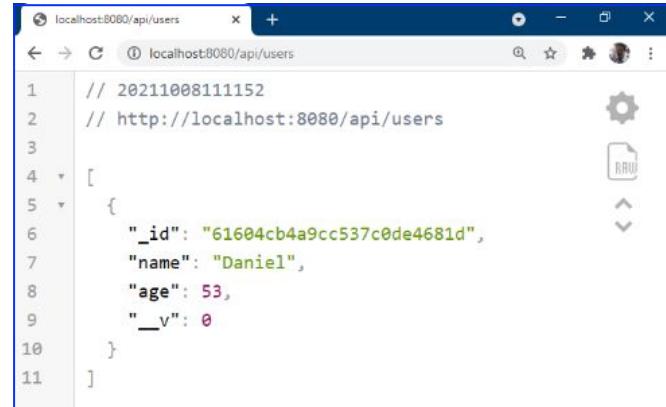
12. La actualización puede ser **parcial** (algunos campos del documento) o **total**.



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



The screenshot shows the MongoDB Atlas interface. On the left, there's a sidebar with 'Project 0', 'Deployment' (selected), 'Databases', 'Triggers', 'Data Lake', 'Security', 'Database Access', 'Network Access', and 'Advanced' options. In the center, under 'Atlas', there's a 'Create Database' button and a 'myFirstDatabase' section with a 'users' collection. The 'users' collection has a size of 67B, 1 document, and 20KB total size. It includes tabs for 'Find', 'Indexes', 'Schema Anti-Patterns', 'Aggregation', and 'Search Indexes'. Below these are 'INSERT DOCUMENT', 'FILTER', 'OPTIONS', 'Apply', and 'Reset' buttons. The 'QUERY RESULTS' section shows 1 document: '_id: ObjectId("61604cb4a9cc537c0de4681d")', 'name: "Daniel"', 'age: 53', and '__v: 0'. A cyan arrow points from this document to the browser window below.

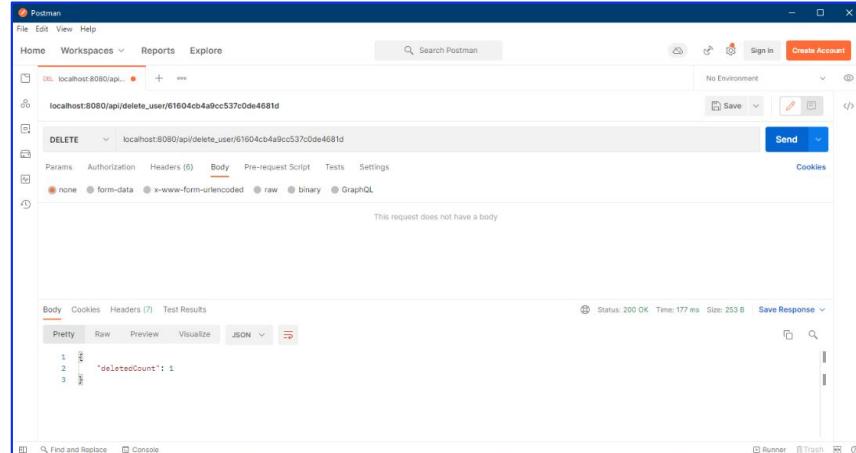


The screenshot shows a browser window with the URL 'localhost:8080/api/users'. The page displays a JSON response:

```
// 20211008111152
// http://localhost:8080/api/users
[{"_id": "61604cb4a9cc537c0de4681d", "name": "Daniel", "age": 53, "__v": 0}]
```

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:

The screenshot shows the MongoDB Atlas interface. At the top, it displays 'Atlas', 'Realm', and 'Charts'. Below that, 'DANIEL'S ORG - 2021-09-15 > PROJECT 0 > DATABASES' is shown. A cluster named 'Cluster0' is selected, with 'VERSION 4.4.9' and 'REGION AWS N. Virginia (us-east-1)'. Under 'Collections', there is one database 'myFirstDatabase' containing one collection 'users'. The 'users' collection has a size of 0B, 0 documents, and 24KB of indexes. It includes tabs for 'Find', 'Indexes', 'Schema Anti-Patterns', 'Aggregation', and 'Search Indexes'. A 'FILTER' button with the query '{ field: 'value' }' is present, along with 'OPTIONS', 'Apply', and 'Reset' buttons. An 'INSERT DOCUMENT' button is also visible. The 'QUERY RESULTS' section shows 0 results.

The screenshot shows a browser window with the URL 'localhost:8080/api/users'. The page displays a JSON response with the following content:

```
// 20211008112631
// http://localhost:8080/api/users
[ ]
```

**¡Sigamos
trabajando!**