

Esta clase va a ser

- grabada

Unidad 8 . DESARROLLO AVANZADO DE BACKEND

Unidad 8: Desarrollo Avanzado de Backend con Node.js y MongoDB

Objetivos de la clase

- **Comprender** los diferentes clientes para utilizar MongoDB
- **Comprender** una DBaaS y configurar MongoDB Atlas para conectar con Nodejs
- **Utilizar** Mongoose para comenzar a utilizar MongoDB a nivel aplicación.
- Hacer una integración práctica de todos los conceptos vistos hasta el momento, bajo el desarrollo de un proyecto paralelo a nuestro proyecto final.

CLASE N°13

Glosario

CRUD: Acrónimo que hace referencia a las cuatro operaciones fundamentales de una base de datos (Create Read Update Delete)

Proyecciones: Una proyección se incluye al momento de hacer una búsqueda.

Sort: Sirve para poder hacer un ordenamiento de la información. La sintaxis es:
`db.collection.find().sort({val_A:1,val_B:-1})`

Skip: Omite el número de documentos indicados. Su sintaxis es: `.skip(offset)`

Limit: Limita el número de documentos devueltos. Su sintaxis es: `.limit(num)`

Configuración de Monto Atlas



APROXIMACIÓN AL PROCESO

1. Llenar el formulario con nuestros datos

See what Atlas is capable of for free

First Name* ⓘ

This field is required

Last Name*

Company

Email*

Password* ⓘ

☐ Acepto las **Terms of Service** y la **Privacy Policy**.



APROXIMACIÓN AL PROCESO

2. Llenamos algunas preguntas

What is your goal today?

Your answer will help us guide you to successfully getting started with MongoDB Atlas.

- ☐ Migrate an existing application
- ☐ Explore what I can build
- ☐ Build a new application
- ☐ Learn MongoDB

What type of application are you building?

Select...


What is your preferred language?

We'll use this to customize code samples and content we share with you. You can always change this later.



APROXIMACIÓN AL PROCESO

3. Seleccionamos la opción gratuita


**Serverless**

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

- ✓ Pay only for the operations you run
- ✓ Resources scale seamlessly to meet your workload
- ✓ Always-on security and backups

Create

Starting at
\$0.10/1M reads


ADVANCED
**Dedicated**

For production applications with sophisticated workload requirements. Advanced configuration controls.

- ✓ Network isolation and fine-grained access controls
- ✓ On-demand performance advice
- ✓ Multi-region and multi-cloud options available

Create

Starting at
\$0.08/hr*
*estimated cost \$56.94/month

FREE
**Shared**

For learning and exploring MongoDB in a cloud environment. Basic configuration options.

- ✓ No credit card required to start
- ✓ Explore with sample datasets
- ✓ Upgrade to dedicated clusters for full functionality

Create

Starting at
FREE



APROXIMACIÓN AL PROCESO

4. Configuramos nuestro Cluster y seteamos el nombre a trabajar

Cluster Tier

M0 Sandbox (Shared RAM, 512 MB Storage)
Encrypted

Additional Settings

MongoDB 5.0, No Backup

Cluster Name

CoderCluster

One time only: once your cluster is created, you won't be able to change its name.

CoderCluster

Cluster names can only contain ASCII letters, numbers, and hyphens.

FREE

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

Back

Create Cluster



APROXIMACIÓN AL PROCESO

5. Seteamos a nuestro usuario y su respectivo password


Username

CoderUser

Password 

...

 Autogenerate Secure Password

 Copy

Create User



6. Colocamos la dirección 0.0.0.0 en la lista de IP, el cual significa "cualquiera"

Add entries to your IP Access List

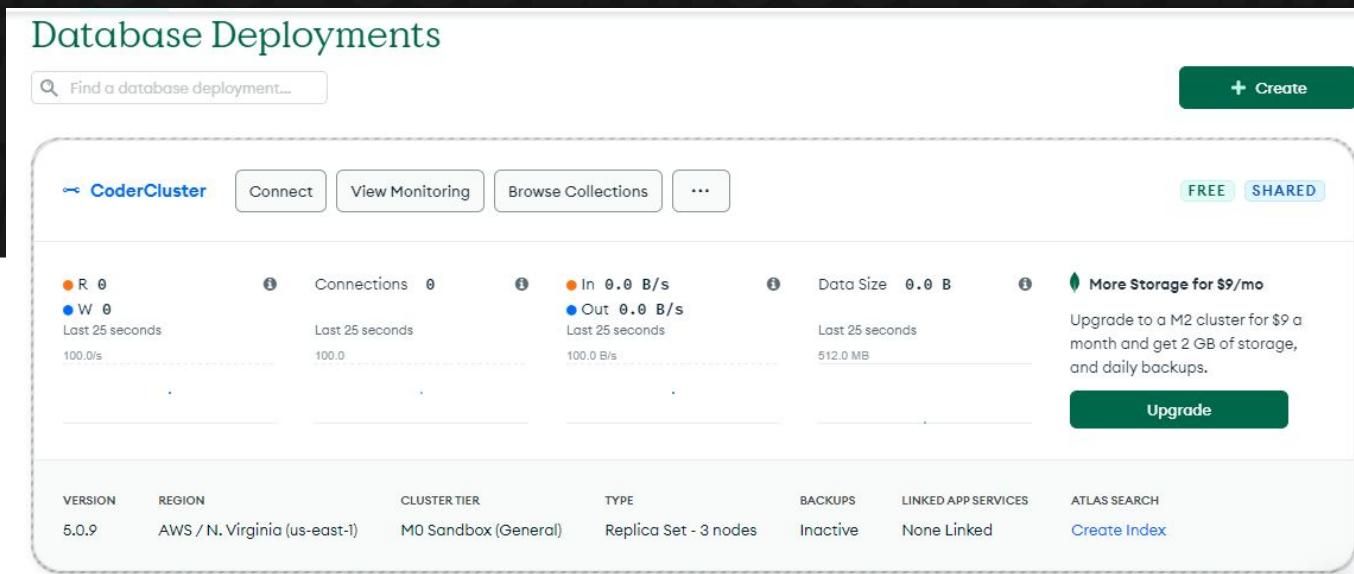
Only an IP address you add to your Access List will be able to connect to your project's clusters. You can manage existing IP entries via the [Network Access Page](#).

IP Address	Description		
<input type="text" value="Enter IP Address"/>	<input type="text" value="Enter description"/>	<input type="button" value="Add Entry"/>	<input type="button" value="Add My Current IP Address"/>

IP Access List	Description	
0.0.0.0/0		<input type="button" value="REMOVE"/>

¡Listo! Ahora podremos conectar nuestra aplicación a la nube para el almacenamiento.

Pero primero, hay que utilizar mongo en nodejs.

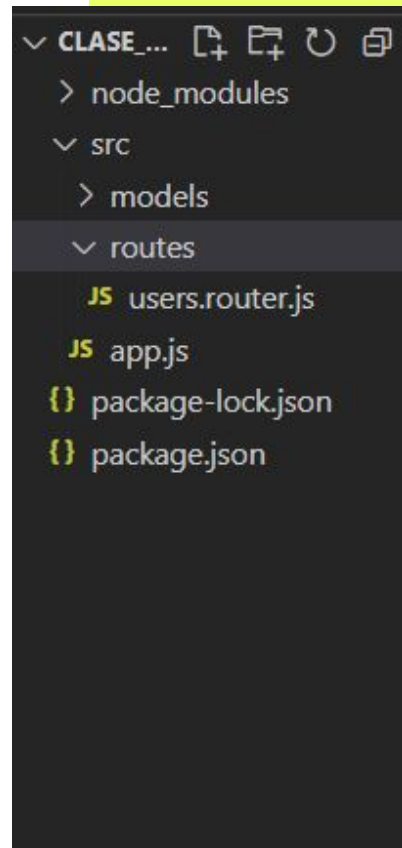


Instalación y configuración de Mongoose

1. Tener listo nuestro proyecto con express

Contaremos con su proyecto dentro de la carpeta src, siguiendo la estructura que hemos estructurado. Además, con un router de users para poder ejemplificar un CRUD con Mongoose.

Por último, contaremos con una nueva carpeta llamada "models" donde guardaremos cada modelo (esquema) que queramos modelar de la base de datos.





APROXIMACIÓN AL PROCESO

2. Instalación de Mongoose

Para poder utilizar mongoose, bastará con utilizar el comando

```
Clase_Mongoose> npm install mongoose
```

Recuerda que para este punto ya debiste haber hecho npm install express y debiste hacer el router comentado en el paso 1. Esto debido a que necesitaremos especificar exactamente dónde queremos utilizar mongoose



APROXIMACIÓN AL PROCESO

3. Archivo user.model.js

En nuestra carpeta “models” crearemos nuestro primer modelo **user**. Utilizaremos mongoose para definir el esquema de nuestra base de datos.

Un esquema **debe** contener las **propiedades y tipos de datos** que **aparecerán en la base de datos**.

Antes de hacer un esquema, debemos tener bien definido qué propiedades deberá tener para poder trabajar con él.

```
JS user.model.js X
src > models > JS user.model.js > ...
1 import mongoose from 'mongoose';
2
3 const userCollection = 'usuarios' //Así es como se llamará la colección en nuestra base de datos.
4
5 const userSchema = new mongoose.Schema({
6   //Aquí procederemos a escribir todas las propiedades que queremos que tenga un usuario en nuestra base
7   first_name:String, //Si sólo necesitamos delimitar el tipo de dato, lo hacemos con los dos puntos.
8   last_name:String,
9   email://{Si necesitamos especificar más detalles (como el "unique") tenemos que hacerlo como objeto.
10     type:String,
11     unique:true
12 })
13
14
15 /**
16  * Ahora, con mongoose.model, generaremos el modelo funcional de un usuario conectado a la base de datos
17  * La parte del cuerpo es el userSchema, pero el userModel ya refiere a un aspecto funcional de éstos.
18  */
19
20 export const userModel = mongoose.model(userCollection,userSchema);
```




4. Ahora podemos importar nuestro “userModel” y utilizarlo en el router de usuarios

JS users.router.js X

src > routes > JS users.router.js > ...

```
1  import {Router} from 'express';
2  import { userModel } from '../models/user.model.js'; //Aquí importamos el userModel
3
4
5  const router = Router();
6
7  router.get('/', async (req, res) => { //La función SIEMPRE debe ser asíncrona al trabajar con mongoose
8    try{
9      let users = await userModel.find() //Nota que es un find idéntico al que hicimos en consultas pasadas.
10     res.send({result: "success", payload: users})
11    }
12    catch(error){
13      console.log("Cannot get users with mongoose: "+error)
14    }
15  })
```

Solo falta un detalle:
Conectar Mongoose a nuestra base
de Atlas



APROXIMACIÓN AL PROCESO

Volviendo a nuestra cuenta de Atlas, notaremos que hay un botón de “Connect”.

Database Deployments

Find a database deployment...

+ Create

CoderCluster

Connect

View Monitoring

Browse Collections

...

FREE

SHARED



APROXIMACIÓN AL PROCESO

Nos da diferentes formas de conexión. Utilizaremos el "Connect your application"

Connect to CoderCluster


✓ Setup connection security


Choose a connection method


Connect

Choose a connection method [View documentation](#)

Get your pre-formatted connection string by selecting your tool below.

 **Connect with the MongoDB Shell**
Interact with your cluster using MongoDB's interactive Javascript interface

 **Connect your application**
Connect your application to your cluster using MongoDB's native drivers

 **Connect using MongoDB Compass**
Explore, modify, and visualize your data with MongoDB's GUI

Go Back

Close



APROXIMACIÓN AL PROCESO

Recibiremos una liga para poder conectarnos, la utilizaremos en nuestra aplicación de nodejs

×

Connect to CoderCluster

✓ Setup connection security ✓ Choose a connection method Connect

1 Select your driver and version

DRIVER

Node.js

VERSION

4.1 or later

2 Add your connection string into your application code

☐ Include full driver code example

mongodb+srv://CoderUser:<password>@codercluster.w5adegs.mongodb.net/?retryWrites=true&w=majority

📋

Replace **<password>** with the password for the **CoderUser** user. Ensure any option params are [URL encoded](#).

Having trouble connecting? [View our troubleshooting documentation](#)

Go Back Close

Volviendo a nuestro código 🤔

Agregando la conexión en nuestro archivo app.js

JS app.js x

src > JS app.js > ...

```
1  import express from 'express';
2  import userRouter from './routes/users.router.js'
3  import mongoose from 'mongoose'; // Importamos mongoose también en nuestro archivo principal.
4
5  const app = express();
6  const server = app.listen(8080, () => console.log("Listening on PORT: 8080"));
7  /**
8   * Utilizaremos el método "mongoose.connect", donde podremos pasar la url que recibimos de Atlas
9   * ¡OJO! la liga no contempla el password, por lo que tendrás que sustituir el <password> por la contraseña que
10  * hayas elegido del usuario que hayas creado para dicho cluster (no es la contraseña de tu cuenta de atlas)
11  * por último, colocamos un argumento de callback en caso de que haya habido un error de conexión
12  * (Usualmente, si no se logró conectar a la base de datos, no podemos continuar con el servidor, así que se suele
13  * preferir cerrar el proceso.)
14  */
15  mongoose.connect('mongodb+srv://CoderUser:123@codercluster.w5adegs.mongodb.net/?retryWrites=true&w=majority', (error) =>
16    if(error){
17      console.log("Cannot connect to database: "+error)
18      process.exit()
19    }
20  })
21
22
23
24  app.use('/api/users', userRouter);
```

Probando en Postman el endpoint, debemos obtener respuesta del router al llamar a mongoose.

New Collection / Obtener los usuarios de la base de datos

GET http://localhost:8080/api/users/ **Send**

Params Authorization Headers (8) Body ● Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
-----	-------	-------------	-----	-----------

Body Cookies Headers (7) Test Results 200 OK 68 ms 268 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {  
2   "result": "success",  
3   "payload": []  
4 }
```


Complementando el CRUD

Método Post para crear el usuario

```
14     }  
15   })  
16  
17   router.post('/', async(req,res)=>{  
18     //Primero obtenemos los datos que necesitaremos, según lo definido en nuestro schema.  
19     let {first_name,last_name,email} = req.body;  
20     //Evaluamos que los valores sí existan  
21     if(!first_name||!last_name||!email) return res.send({status:"error",error:"Incomplete values"});  
22     //Si todo está en orden, pedimos a mongoose que inserte el nuevo documento.  
23     //Nota que aquí, para poder crear un modelo, utilizaremos el  
24     let result = await userModel.create({  
25       first_name,  
26       last_name,  
27       email  
28     });  
29     //Devolvemos el usuario recién creado.  
30     res.send({status:"success",payload:result})  
31   })  
32  
33   export default router;
```

Recuerda agregar en app.js el "app.use(express.json())" para recibir correctamente req.body.

Creando el usuario con Postman

The screenshot shows the Postman interface for a new collection named "Crear un usuario". The request is a POST to `http://localhost:8080/api/users/`. The "Body" tab is selected, and the "raw" radio button is chosen. The response is displayed in the "Body" tab, showing a 200 OK status with a response time of 127 ms and a body size of 388 B. The response is formatted as JSON and shows a successful user creation with a payload containing user details.

POST `http://localhost:8080/api/users/` **Send**

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies Beautify

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

1 5

Body Cookies Headers (7) Test Results 200 OK 127 ms 388 B **Save Response**

Pretty Raw Preview Visualize **JSON**

```
1 {
2   "status": "success",
3   "payload": {
4     "first_name": "Marisol",
5     "last_name": "Cadena",
6     "email": "correoMarisol@correo.com",
7     "_id": "62c2c7dd6aea0685884f1255",
8     "__v": 0
9   }
10 }
```

Métodos PUT y DELETE para finalizar CRUD de usuarios

JS users.router.js X

src > routes > JS users.router.js > router.delete('/:uid') callback

```
32
33 router.put('/:uid', async(req, res) => {
34   //Obtenemos el userId (uid) de los params
35   let {uid} = req.params;
36   //Tomamos todo el usuario a reemplazar.
37   let userToReplace = req.body;
38   if(!userToReplace.first_name || !userToReplace.last_name || !userToReplace.email)
39     return res.send({status: "error", error: "Incomplete Values"})
40   //Nota que estamos buscando un _id y no un id, Mongo maneja internamente el valor _id
41   let result = await userModel.updateOne({_id: uid}, userToReplace)
42   res.send({status: "success", payload: result})
43 }
44
45 router.delete('/:uid', async(req, res) => {
46   //Obtenemos el userId (uid) de los params
47   let {uid} = req.params;
48   //Nota que estamos buscando un _id y no un id, Mongo maneja internamente el valor _id
49   let result = await userModel.deleteOne({_id: uid})
50   res.send({status: "success", payload: result})
51 }
52 export default router;
```



CRUD con Mongoose

Duración: 10 – 15min



ACTIVIDAD EN CLASE

CRUD con Mongoose

- ✓ Realizar un proyecto en Node.js que se conecte a una base de datos MongoDB Atlas llamada colegio. Utilizar mongoose importándolo en Módulo (import) y gestionar sus acciones a través de promesas.
- ✓ Crear una colección llamada 'estudiantes' que incorporará 10 documentos con la siguiente estructura y datos que se detallan a continuación:
 - a) nombre: tipo string
 - b) apellido: tipo string
 - c) edad: tipo number
 - d) dni: tipo string (campo único)
 - e) curso: tipo string
 - f) nota: tipo numberTodos los campos deben ser requeridos obligatoriamente (`{ required: true }`)



ACTIVIDAD EN CLASE

CRUD con Mongoose

- ✓ Insertar un arreglo de estudiantes a dicha colección
- ✓ Desarrollar los endpoints correspondientes al CRUD pensado para trabajar con esta colección
- ✓ Corroborar los resultados con Postman.

