



Clase 19. Programación Backend

Mongoose





OBJETIVOS DE LA CLASE

- Conectarse a una base de datos MongoDB a través de Node.js.
- Utilizar mongoose para definir esquemas, modelos e interactuar con la base.
- Realizar un CRUD utilizando mongoose

CRONOGRAMA DEL CURSO

Clase 18



CRUD en MongoDB

Clase 19



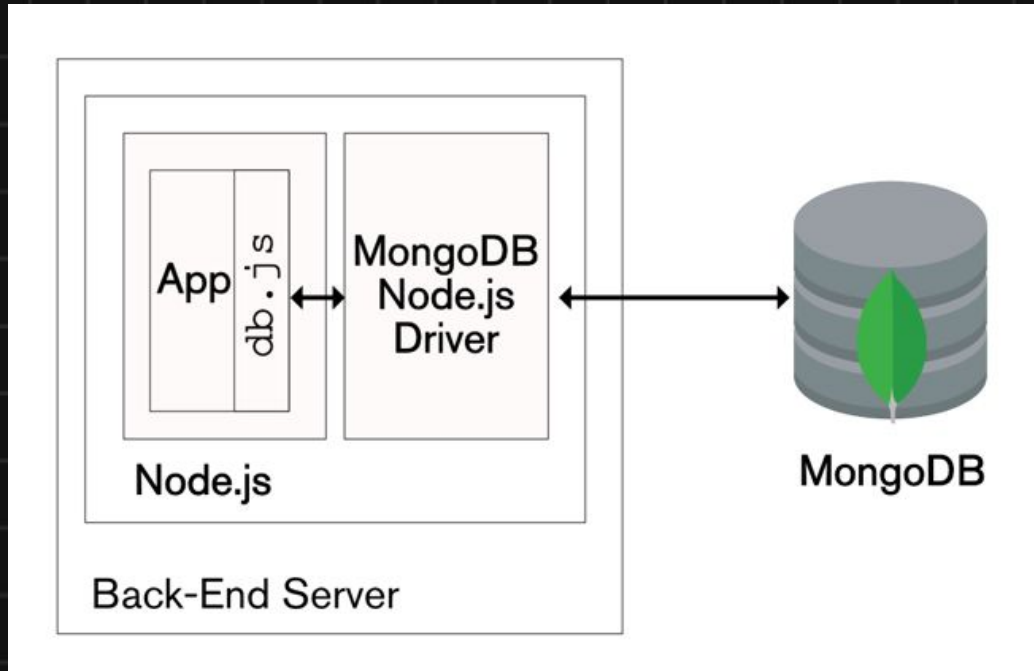
Mongoose

Clase 20



DBaaS & Firebase

MongoDB con Node.js



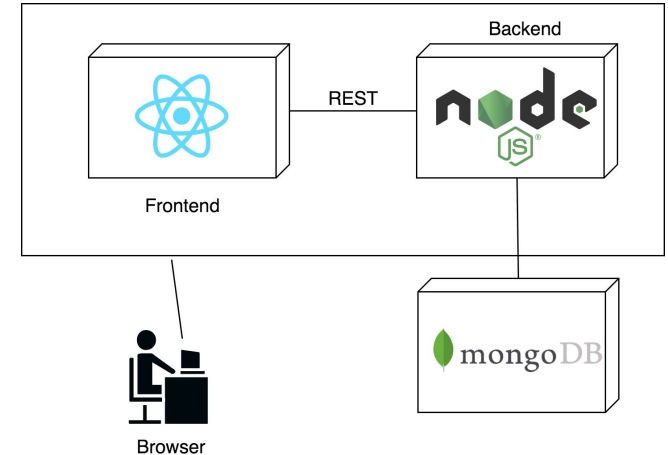
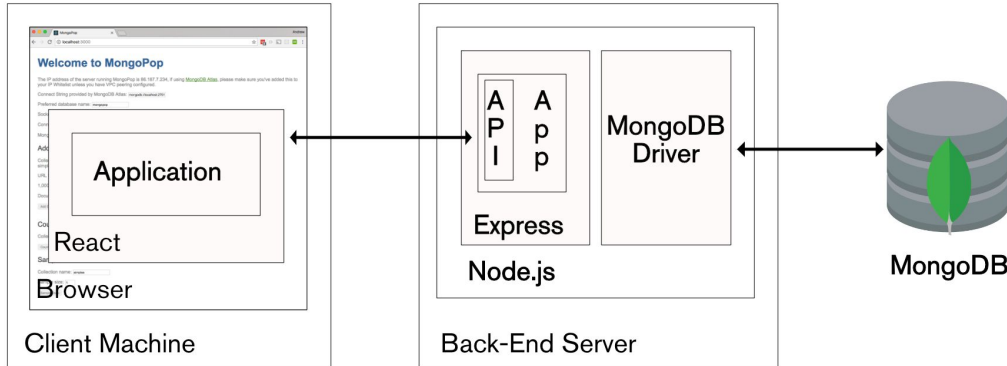
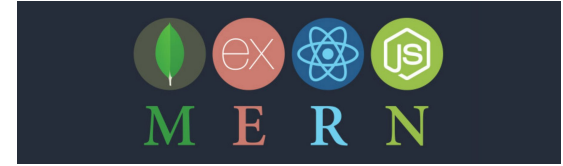
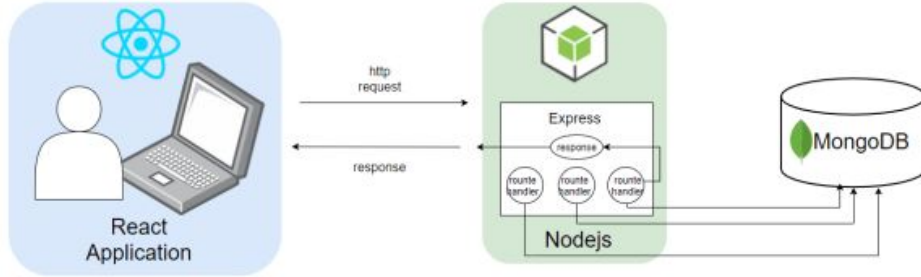
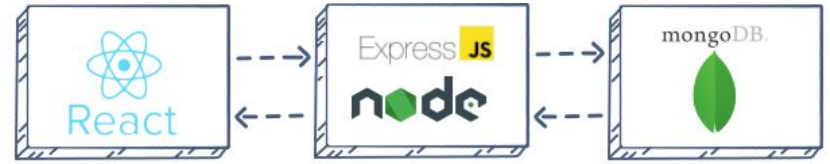


MongoDB en MERN Stack

Recordemos que **MongoDB** constituye una de las herramientas recomendadas de uso en el MERN Stack:

- **MongoDB** es un base de datos NoSQL que está orientada a documentos.
- **Express** es una infraestructura de aplicaciones web Node.js
- **React JS** es una biblioteca para crear componentes de interfaz de usuario.
- **Node.js** es un entorno de ejecución para JavaScript que puede permitirle ejecutar JavaScript fuera del navegador, por ejemplo del lado servidor.

MERN Stack Esquemas



Mongoose

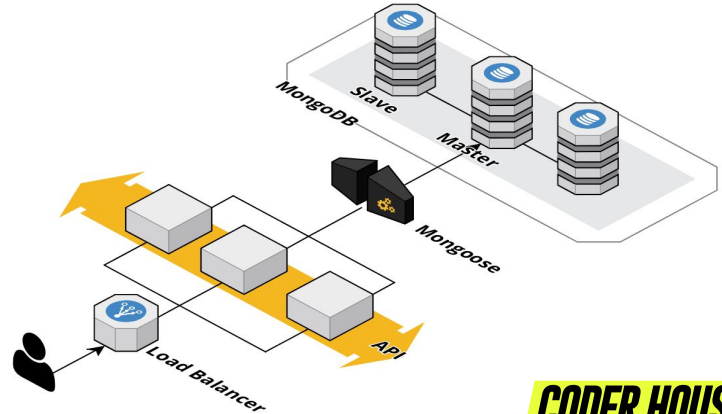
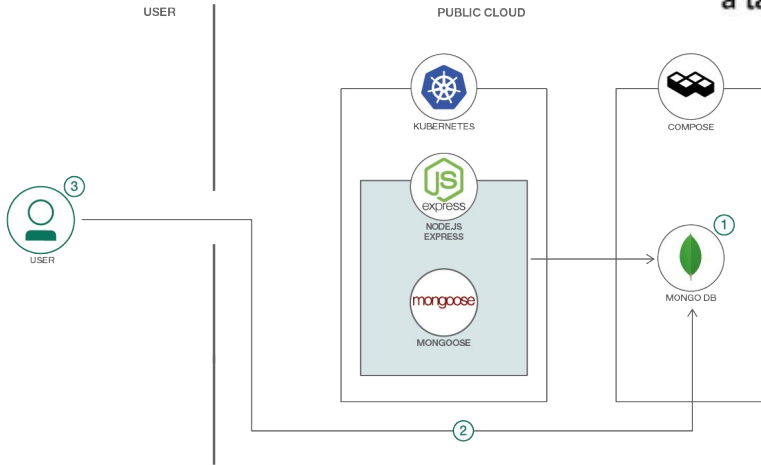
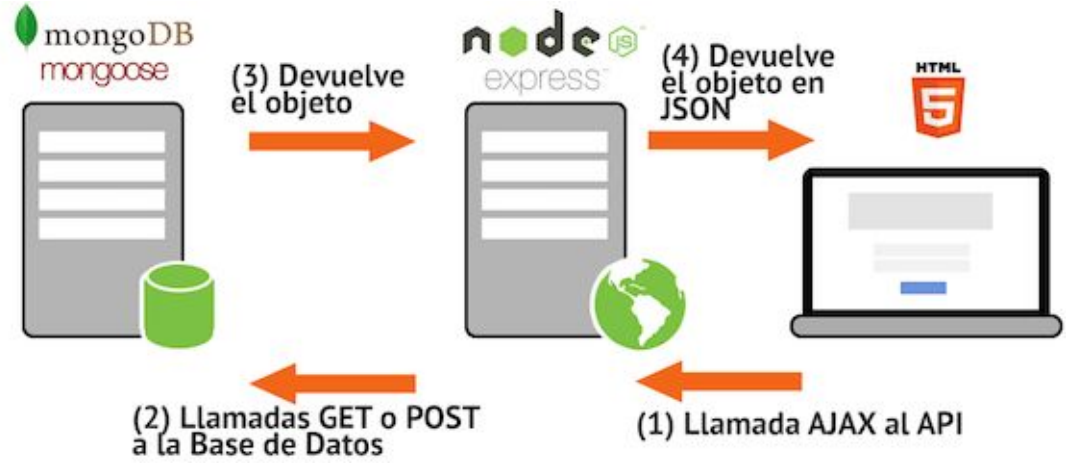
mongoose

elegant **mongodb** object modeling for **node.js**

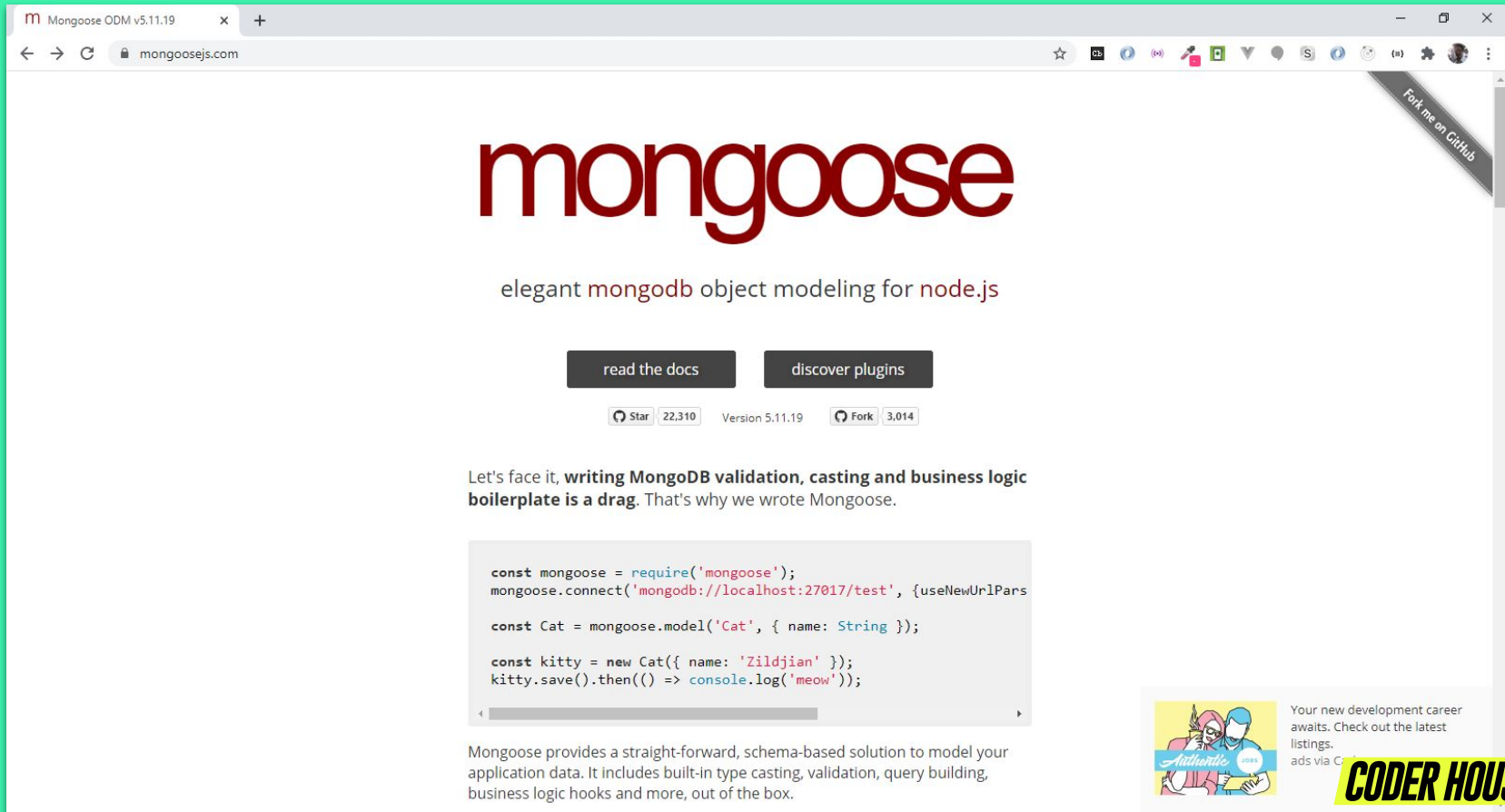
¿Qué es Mongoose?

- Mongoose es una **dependencia Javascript** que realiza la **conexión** a la **instancia** de **MongoDB**
- Pero la magia real del módulo Mongoose es la **habilidad** para **definir** un **esquema del documento**.
- *MongoDB* usa colecciones para almacenar múltiples documentos, los cuales no necesitan tener la misma estructura.
- Cuando tratamos con objetos es necesario que los documentos sean algo parecido. En este punto nos ayudan los esquemas y modelos de **Mongoose**.

Esquemas Mongoose



Website oficial: <https://mongoosejs.com/>



The screenshot shows the official Mongoose ODM website in a web browser. The browser's address bar displays 'mongoosejs.com'. The website features the 'mongoose' logo in a large, dark red font. Below the logo, the tagline 'elegant mongodb object modeling for node.js' is centered. Two dark buttons, 'read the docs' and 'discover plugins', are positioned below the tagline. Underneath these buttons, GitHub statistics are shown: 22,310 stars and 3,014 forks. A paragraph of text describes Mongoose as a solution for MongoDB validation, casting, and business logic boilerplate. A code block contains a JavaScript snippet demonstrating how to use Mongoose to connect to a database and create a 'Cat' model. In the bottom right corner, there is a small advertisement for 'CODER HOUSE' featuring a cartoon illustration of two people and text about development career opportunities.

Mongoose ODM v5.11.19

mongoosejs.com

mongoose

elegant **mongodb** object modeling for node.js

[read the docs](#) [discover plugins](#)

Star 22,310 Version 5.11.19 Fork 3,014


Let's face it, **writing MongoDB validation, casting and business logic boilerplate is a drag.** That's why we wrote Mongoose.

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/test', {useNewUrlParser: true});

const Cat = mongoose.model('Cat', { name: String });

const kitty = new Cat({ name: 'Zildjian' });
kitty.save().then(() => console.log('meow'));
```

Mongoose provides a straight-forward, schema-based solution to model your application data. It includes built-in type casting, validation, query building, business logic hooks and more, out of the box.

 Your new development career awaits. Check out the latest listings. ads via C
CODER HOUSE

Mongoose: Schema y Model

- Mongoose usa un **objeto Schema** para definir una lista de **propiedades del documento**, cada una con su propio tipo y características para forzar la estructura del documento.
- Después de especificar un esquema deberemos definir un **Modelo constructor** para así poder crear instancias de los documentos de MongoDB



mongoose
elegant mongodb object modeling for node.js

Mongoose: Schema y Model

```
const Schema = mongoose.Schema;
```

```
const employeeSchema = new Schema({  
  name : { type : String, required : true, max : [127, "Max Length is 127  
characters"] },  
  age : { type : Number, required : true},  
  salary : Number,  
  designation : { type : String, required : true}  
});
```

```
module.exports = mongoose.model('Employee', employeeSchema);
```



Schema y Model : Validaciones



- Mongoose es un **Object Document Mapper (ODM)**. Esto significa que permite definir objetos con un **esquema fuertemente tipado** que se asigna a un documento MongoDB.
- Mongoose proporciona una amplia cantidad de funcionalidades para crear y trabajar con esquemas.



Schema y Model : Validaciones



Actualmente contiene ocho ***SchemaTypes*** definidos para una propiedad

- *String (Cadena)*
- *Number (Número)*
- *Date (Fecha)*
- *Buffer*
- *Boolean (Booleano)*
- *Mixed (Mixto)*
- *ObjectId*
- *Array (Matriz)*



Schema y Model : Validaciones



- **Cada tipo de dato permite especificar:**
 - Un valor predeterminado
 - Una función de validación personalizada
 - La indicación de campo requerido
 - Una función get que le permite manipular los datos antes de que se devuelva como un objeto
 - Una función de conjunto que le permite manipular los datos antes de guardarlos en la base de datos
 - Crear índices para permitir que los datos se obtengan más rápido



Schema y Model : Validaciones



Además de estas opciones comunes, ciertos tipos de datos permiten **personalizar** cómo se almacenan y recuperan los datos de la base de datos.

Por ejemplo, un **String** especifica **opciones adicionales**:

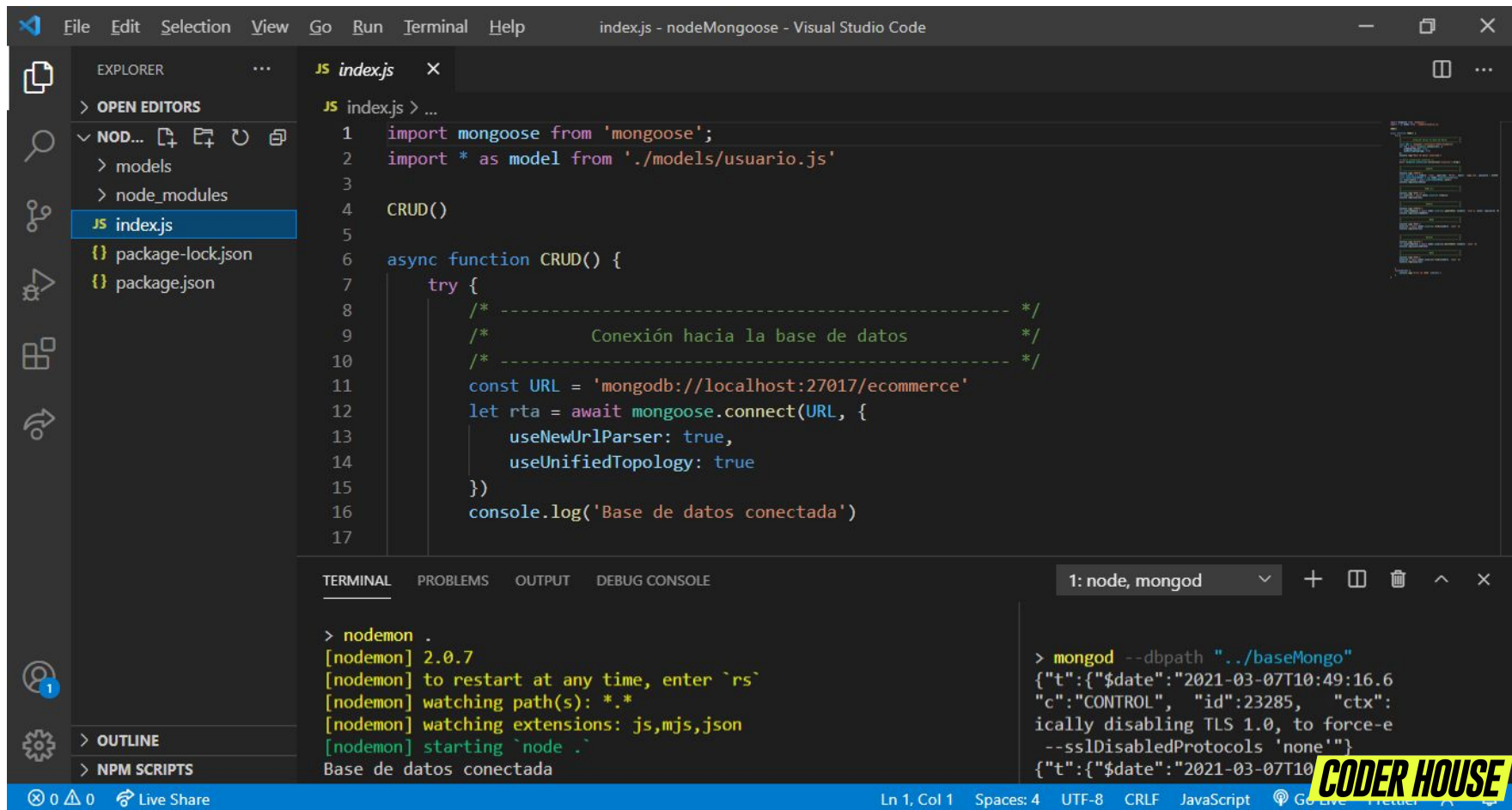
- Convertir en minúsculas y a mayúsculas
- Recortar datos antes de guardar
- Una expresión regular que puede limitar los datos que se pueden guardar durante el proceso de validación
- Una enumeración que puede definir una lista de cadenas que son válidas

Integrando Mongoose en un proyecto Node.js

Configuración del proyecto: pasos a seguir

1. Creamos un proyecto Node.js con **npm init -y**
2. Instalamos la dependencia mongoose con **npm i mongoose**
3. Describimos nuestro modelo de datos (Schema + Model) con las validaciones necesarias.
4. Levantamos el motor de base de datos MongoDB.
5. Creamos la función de conexión mediante mongoose, con las opciones configuradas.
6. Con mongoose realizamos las operaciones CRUD hacia MongoDB: Read, Create, Update y Delete.
7. Mostramos consultas con distintos filtros de Query y con el uso de projection, funciones sort, limit y skip

Mongoose: Conexión hacia la base de datos



The screenshot displays the Visual Studio Code interface for a Node.js project named 'nodeMongoose'. The Explorer sidebar on the left shows the project structure, including 'package-lock.json', 'package.json', and 'index.js'. The Editor window shows the code for 'index.js', which imports Mongoose and connects to a MongoDB instance at 'localhost:27017/ecommerce'. The Terminal window at the bottom shows the execution of 'nodemon' and 'mongod' commands, indicating that the database is running and the application is monitoring for changes.

```
index.js - nodeMongoose - Visual Studio Code
```

EXPLORER

OPEN EDITORS

NOD...
models
node_modules
JS index.js
package-lock.json
package.json

index.js

```
1 import mongoose from 'mongoose';
2 import * as model from './models/usuario.js'
3
4 CRUD()
5
6 async function CRUD() {
7   try {
8     /* ----- */
9     /*     Conexión hacia la base de datos     */
10    /* ----- */
11    const URL = 'mongodb://localhost:27017/ecommerce'
12    let rta = await mongoose.connect(URL, {
13      useNewUrlParser: true,
14      useUnifiedTopology: true
15    })
16    console.log('Base de datos conectada')
17  }
```

TERMINAL

1: node, mongod

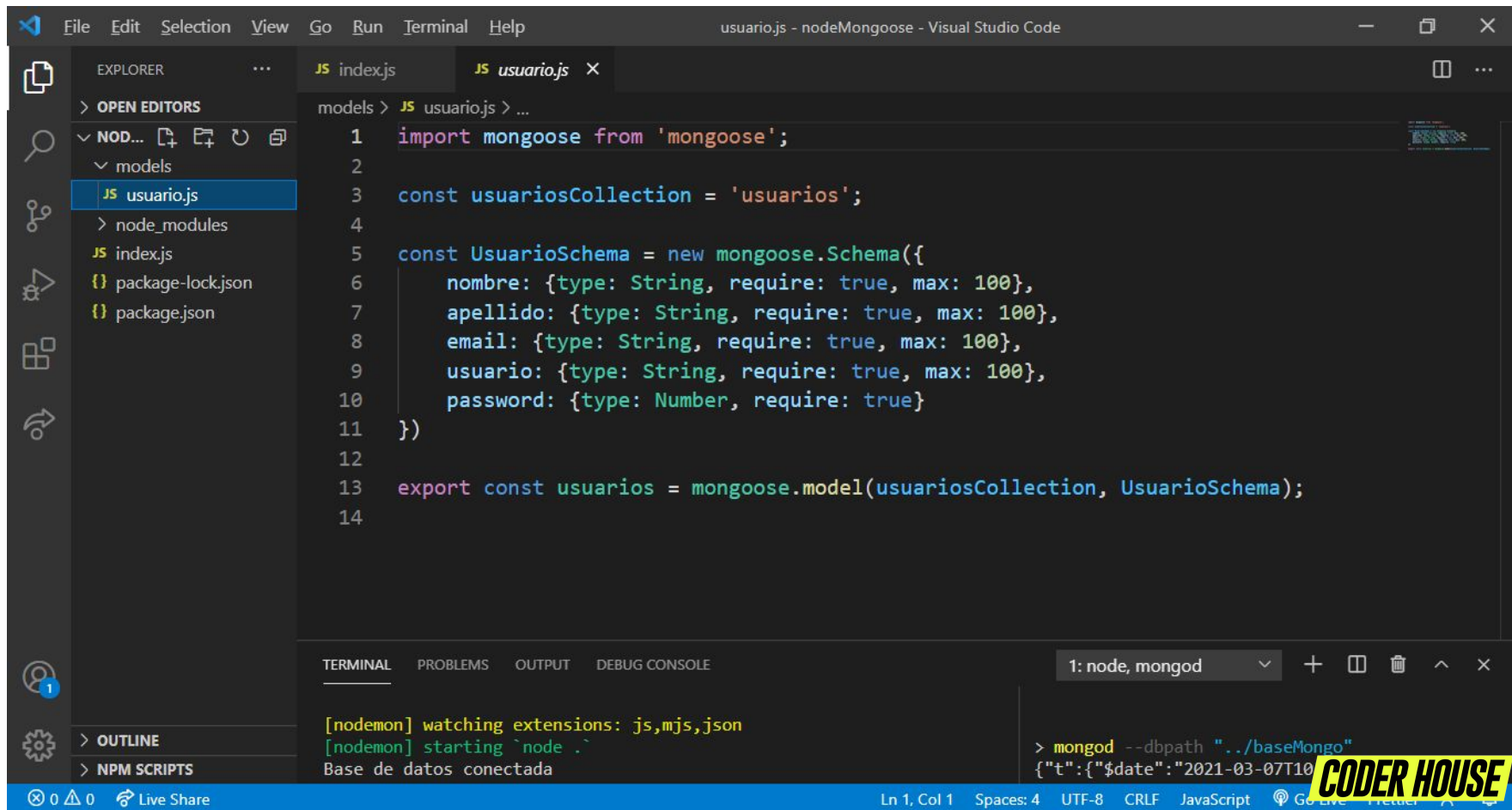
```
> nodemon .
[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node .`
Base de datos conectada
```

```
> mongod --dbpath "../baseMongo"
{"t":{"$date":"2021-03-07T10:49:16.6
"c":"CONTROL", "id":23285, "ctx":
ically disabling TLS 1.0, to force-e
--sslDisabledProtocols 'none'}}
{"t":{"$date":"2021-03-07T10:49:16.6
```

Ln 1, Col 1 Spaces: 4 UTF-8 CRLF JavaScript

CODER HOUSE

Mongoose: Modelo de datos



Visual Studio Code interface showing a project structure with a file named `usuario.js` in the `models` directory. The file contains a Mongoose schema for a user model.

```
1 import mongoose from 'mongoose';
2
3 const usuariosCollection = 'usuarios';
4
5 const UsuarioSchema = new mongoose.Schema({
6   nombre: {type: String, require: true, max: 100},
7   apellido: {type: String, require: true, max: 100},
8   email: {type: String, require: true, max: 100},
9   usuario: {type: String, require: true, max: 100},
10  password: {type: Number, require: true}
11 })
12
13 export const usuarios = mongoose.model(usuariosCollection, UsuarioSchema);
14
```

The terminal output shows the following commands and results:

```
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node .`
Base de datos conectada

1: node, mongod
> mongod --dbpath "../baseMongo"
{"t":{"$date":"2021-03-07T10:00:00.000Z","$type":"date"},"v":{"$date":"2021-03-07T10:00:00.000Z","$type":"date"},"s":1,"op":"insert","ns":"test.usuario","key":{"usuario":"1","password":1},"obj":{"usuario":"1","password":1}}
```

Visual Studio Code status bar at the bottom shows: Ln 1, Col 1 | Spaces: 4 | UTF-8 | CRLF | JavaScript | Go Live | Predefined | A | G

CODER HOUSE

Mongoose: CREATE / READ ALL

The screenshot displays the Visual Studio Code interface with a project named 'index.js - nodeMongoose'. The Explorer sidebar on the left shows the file structure with 'index.js' selected. The main editor window displays the following JavaScript code:

```
20
21  /* ----- */
22  /*             CREATE             */
23  /* ----- */
24  console.log('CREATE')
25  const usuario = { nombre: 'Juan', apellido: 'Perez', email: 'jp@g.com', password : 123456 }
26  const usuarioSaveModel = new model.usuarios(usuario);
27  let usuarioSave = await usuarioSaveModel.save()
28  console.log(usuarioSave)
29
30  /* ----- */
31  /*             READ all             */
32  /* ----- */
33  console.log('READ all')
34  let usuarios = await model.usuarios.find({})
35  console.log(usuarios)
36
37  /* ----- */
```

The TERMINAL panel at the bottom shows the output of the application:

```
CREATE
{
  _id: 6044d9bec530193c6034529c,
  nombre: 'Juan',
  apellido: 'Perez',
  email: 'jp@g.com',
  password: 123456,
```

On the right side of the terminal, a command prompt window shows the execution of the application:

```
1: node, mongod
> mongod --dbpath "../baseMongo"
{"t":{"$date":"2021-03-07T10:49:16.6
"c":"CONTROL", "id":23285, "ctx":
ically disabling TLS 1.0, to force-e
--sslDisabledProtocols 'none'}}
{"t":{"$date":"2021-03-07T10
```

The status bar at the bottom indicates the current file is 'Ln 1, Col 1' with 'Spaces: 4' and 'UTF-8' encoding. The 'CODER HOUSE' logo is visible in the bottom right corner.



MongoDB con mongoose

Tiempo: 10 minutos



- 1) Realizar un proyecto en Node.js que se conecte a una base de datos MongoDB local llamada colegio. Utilizar mongoose importándolo en Módulo (import) y gestionar sus acciones a través de promesas.
- 2) 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 number

Todos los campos deben ser requeridos obligatoriamente ({ required: true })



3) Tomar los valores del siguiente array de objetos

```
[
  { nombre: 'Pedro', apellido: 'Mei', edad: 21, dni: '31155898', curso: '1A', nota: 7 },
  { nombre: 'Ana', apellido: 'Gonzalez', edad: 32, dni: '27651878', curso: '1A', nota: 8 },
  { nombre: 'José', apellido: 'Picos', edad: 29, dni: '34554398', curso: '2A', nota: 6 },
  { nombre: 'Lucas', apellido: 'Blanco', edad: 22, dni: '30355874', curso: '3A', nota: 10 },
  { nombre: 'María', apellido: 'García', edad: 36, dni: '29575148', curso: '1A', nota: 9 },
  { nombre: 'Federico', apellido: 'Perez', edad: 41, dni: '320118321', curso: '2A', nota: 5 },
  { nombre: 'Tomas', apellido: 'Sierra', edad: 19, dni: '38654790', curso: '2B', nota: 4 },
  { nombre: 'Carlos', apellido: 'Fernández', edad: 33, dni: '26935670', curso: '3B', nota: 2 },
  { nombre: 'Fabio', apellido: 'Pieres', edad: 39, dni: '4315388', curso: '1B', nota: 9 },
  { nombre: 'Daniel', apellido: 'Gallo', edad: 25, dni: '37923460', curso: '3B', nota: 2 }
]
```

4) Verificar con el cliente Mongo Shell (CLI) que los datos estén almacenados en la base y colección que corresponda.



BREAK

¡5/10 MINUTOS Y VOLVEMOS!

Mongoose: UPDATE/ READ

The image shows a Visual Studio Code editor window with the file `index.js` open. The file contains two sections: `UPDATE` and `READ`. The `UPDATE` section (lines 37-44) logs the operation and updates the password of a user named 'Juan' to '654321'. The `READ` section (lines 47-51) logs the operation and finds the user by name. A red arrow points to the `$set` property in the update operation.

```
37  /* ----- */
38  /*                UPDATE                */
39  /* ----- */
40  console.log('UPDATE')
41  let usuarioUpdate = await model.usuarios.updateOne( {nombre: 'Juan'}, {
42    $set: {password: 654321}
43  })
44  console.log(usuarioUpdate)
```

The terminal output shows the result of the update operation:

```
UPDATE
{ n: 1, nModified: 1, ok: 1 }
READ
[
  {
    _id: 6044d9bec530193c6034529c,
    nombre: 'Juan',
    apellido: 'Perez',
    email: 'jp@g.com',
    password: 654321,
```

The MongoDB shell command executed is:

```
> mongod --dbpath "../baseMongo"
{"t":{"$date":"2021-03-07T10:49:16.6
"c":"CONTROL", "id":23285, "ctx":
ically disabling TLS 1.0, to force-e
--sslDisabledProtocols 'none'}}
{"t":{"$date":"2021-03-07T10:49:17.0
"c":"ASIO", "id":22601, "ctx":
sportLayer configured during Network
{"t":{"$date":"2021-03-07T10:49:17.0
"c":"NETWORK", "id":4648602
```

A red arrow points to the `password: 654321` field in the terminal output.

CODER HOUSE

Mongoose: DELETE/ READ

The screenshot shows a Visual Studio Code editor with a project named 'index.js - nodeMongoose'. The Explorer sidebar on the left shows the project structure, including 'package-lock.json' and 'package.json'. The main editor displays the code for the 'index.js' file, which is currently open in the 'CRUD' view. The code includes comments for 'DELETE' and 'READ' operations, and the corresponding Mongoose queries. The Output console at the bottom shows the results of the database queries, with the 'DELETE' operation output highlighted by a red box.

```
52  /* ----- */
53  /*           DELETE           */
54  /* ----- */
55  console.log('DELETE')
56  let usuarioDelete = await model.usuarios.deleteOne( {nombre: 'Juan' })
57  console.log(usuarioDelete)
58
59  /* ----- */
60  /*           READ           */
61  /* ----- */
62  console.log('READ')
63  usuarios = await model.usuarios.find({nombre: 'Juan' })
64  console.log(usuarios)
65
66
67
68  catch(error) {
69    console.log(`Error en CRUD: ${error}`)
70  }
```

Output console (highlighted):

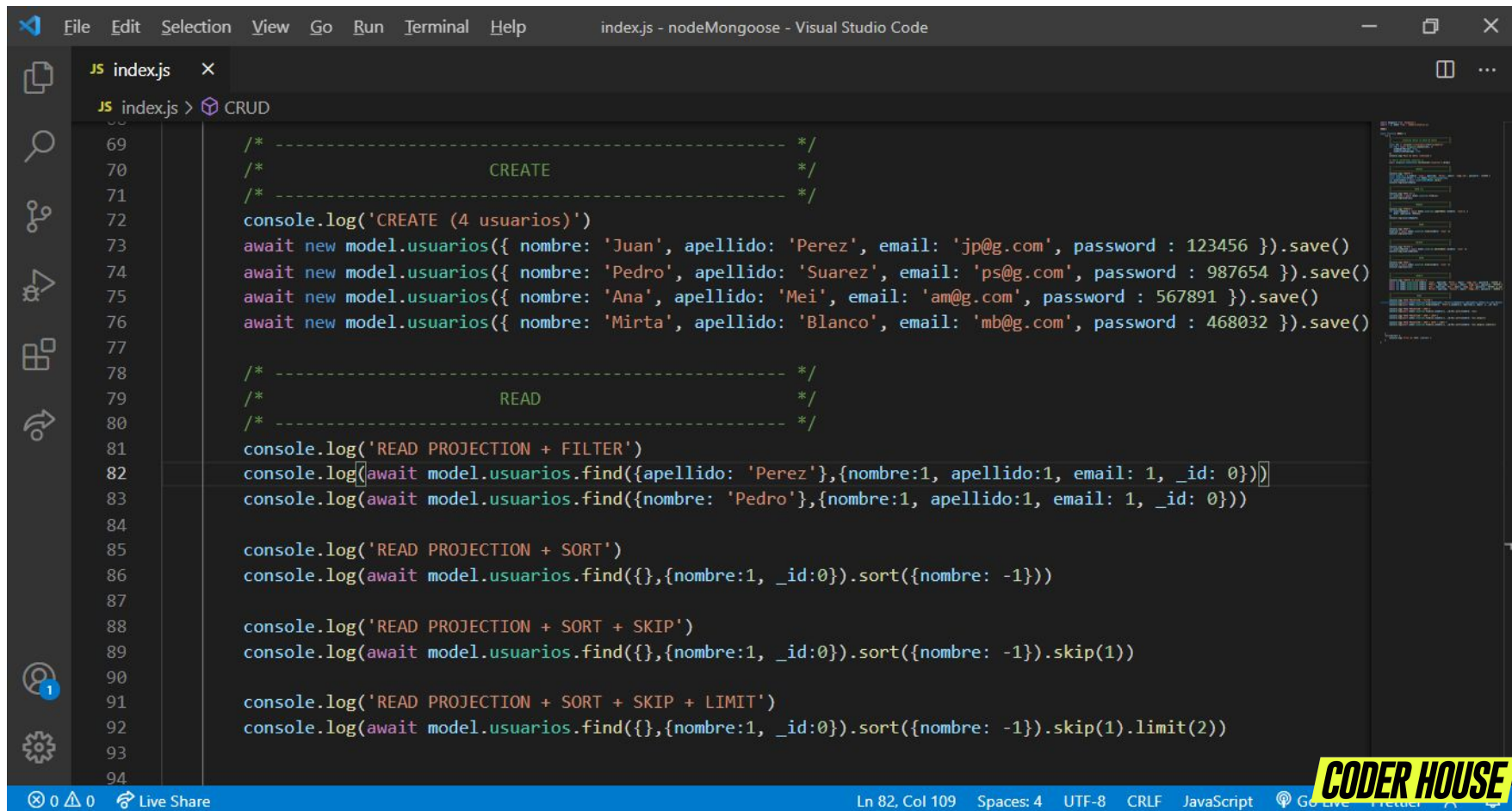
```
DELETE
{ n: 1, ok: 1, deletedCount: 1 }
READ
[]
```

Terminal output:

```
> mongod --dbpath "../baseMongo"
{"t":{"$date":"2021-03-07T10:49:16.6
"c":"CONTROL", "id":23285, "ctx":
ically disabling TLS 1.0, to force-e
--sslDisabledProtocols 'non
```

Footer: Ln 65, Col 1 Spaces: 4 UTF-8 CRLF JavaScript

Mongoose: READ PROJECTION + SORT + SKIP + LIMIT



```
index.js - nodeMongoose - Visual Studio Code

JS index.js x
JS index.js > CRUD

69  /* ----- */
70  /*           CREATE           */
71  /* ----- */
72  console.log('CREATE (4 usuarios)')
73  await new model.usuarios({ nombre: 'Juan', apellido: 'Perez', email: 'jp@g.com', password : 123456 }).save()
74  await new model.usuarios({ nombre: 'Pedro', apellido: 'Suarez', email: 'ps@g.com', password : 987654 }).save()
75  await new model.usuarios({ nombre: 'Ana', apellido: 'Mei', email: 'am@g.com', password : 567891 }).save()
76  await new model.usuarios({ nombre: 'Mirta', apellido: 'Blanco', email: 'mb@g.com', password : 468032 }).save()
77
78  /* ----- */
79  /*           READ           */
80  /* ----- */
81  console.log('READ PROJECTION + FILTER')
82  console.log(await model.usuarios.find({apellido: 'Perez'}, {nombre:1, apellido:1, email: 1, _id: 0}))
83  console.log(await model.usuarios.find({nombre: 'Pedro'}, {nombre:1, apellido:1, email: 1, _id: 0}))
84
85  console.log('READ PROJECTION + SORT')
86  console.log(await model.usuarios.find({}, {nombre:1, _id:0}).sort({nombre: -1}))
87
88  console.log('READ PROJECTION + SORT + SKIP')
89  console.log(await model.usuarios.find({}, {nombre:1, _id:0}).sort({nombre: -1}).skip(1))
90
91  console.log('READ PROJECTION + SORT + SKIP + LIMIT')
92  console.log(await model.usuarios.find({}, {nombre:1, _id:0}).sort({nombre: -1}).skip(1).limit(2))
93
94
```

0 0 0 Live Share Ln 82, Col 109 Spaces: 4 UTF-8 CRLF JavaScript Go Live

CODER HOUSE



- 1) Desarrollar un proyecto en Node.js que realice la lectura de los estudiantes de la base colegio (creada anteriormente) mostrándolos en consola, cumpliendo con los siguientes puntos:
 - a) Los estudiantes ordenados por orden alfabético según sus nombres.
 - b) El estudiante más joven.
 - c) Los estudiantes que pertenezcan al curso '2A'.
 - d) El segundo estudiante más joven.
 - e) Sólo los nombres y apellidos de los estudiantes con su curso correspondiente, ordenados por apellido descendente (z a a).
 - f) Los estudiantes que sacaron 10.
 - g) El promedio de notas del total de alumnos.
 - h) El promedio de notas del curso '1A'.
- 2) Utilizar la interfaz basada en Promises de Mongoose, sintaxis then/catch con importación de módulos en formato CommonJS.
- 3) Los resultados se deben imprimir en orden según los puntos citados (Promesas anidadas con .then)



MongoDB: CRUD

Tiempo: 15 minutos



Realizar un proyecto en Node.js que sobre la base colegio realice las siguientes acciones:

- 1) Actualizar el dni del estudiante Lucas Blanco a 20355875
- 2) Agregar un campo 'ingreso' a todos los documentos con el valor false
- 3) Modificar el valor de 'ingreso' a true para todos los estudiantes que pertenezcan al curso 1A
- 4) Listar los estudiantes que aprobaron (hayan sacado de 4 en adelante) sin los campos de _id y __v
- 5) Listar los estudiantes que posean el campo 'ingreso' en true sin los campos de _id y __v
- 6) Borrar de la colección de estudiantes los documentos cuyo campo 'ingreso' esté en true

¿PREGUNTAS?



¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- MongoDB con node.js
- Introducción a mongoose
- CRUD con mongoose



OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDO LA EDUCACIÓN

CODER HOUSE