

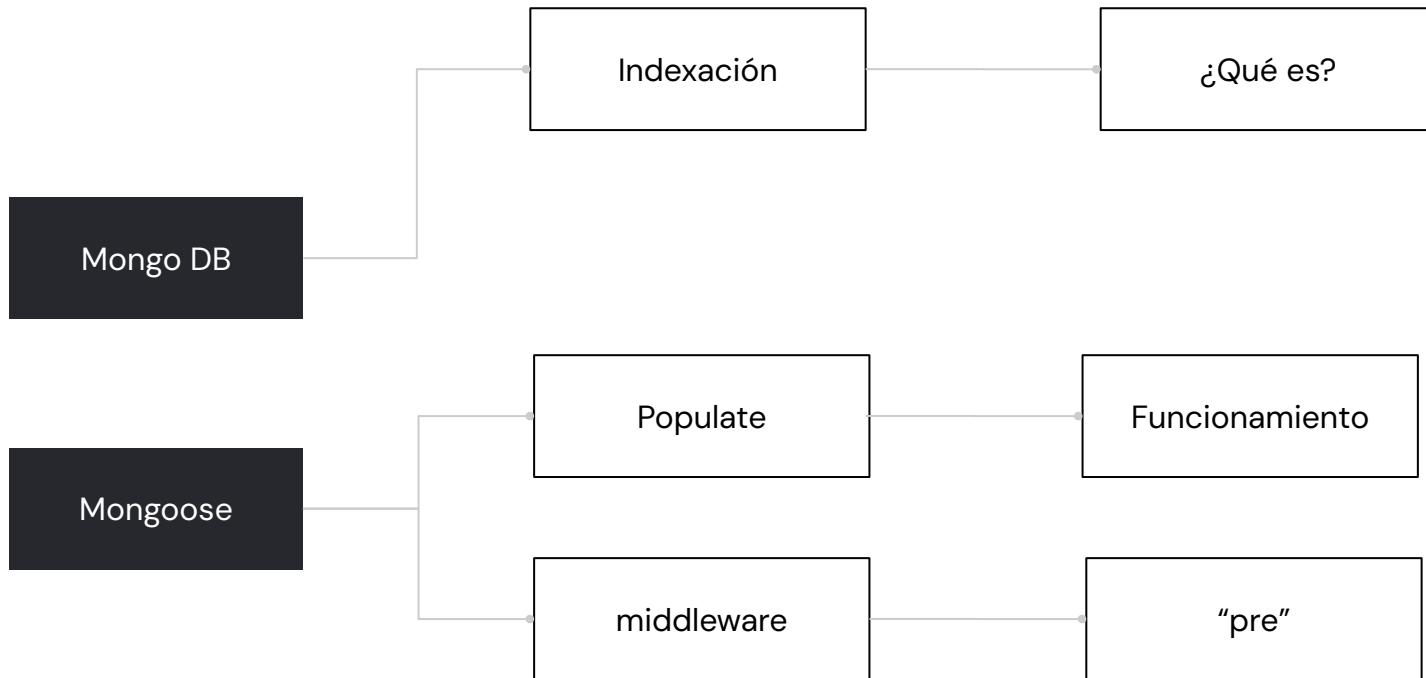
**Clase 16.** DESARROLLO AVANZADO DE BACKEND

# Mongo avanzado I

# Objetivos de la clase

- Comprender el funcionamiento de una indexación en MongoDB
- Analizar documentos para definir sus índices
- Conocer la operación “populate” de mongoose
- Utilización de middleware en mongoose

## MAPA DE CONCEPTOS



# Teoría de indexación

# Indexación

Es un recurso utilizado en MongoDB para poder hacer consultas mucho más rápidas.

Éste nos permitirá tener una referencia previa al momento de buscar un documento, con el fin de evitar recorrer toda la colección, documento por documento, hasta encontrar dicho valor.

El índice se asocia a un atributo del documento y permite que las búsquedas se hagan desde puntos específicos, evitando el recorrido completo de la colección.



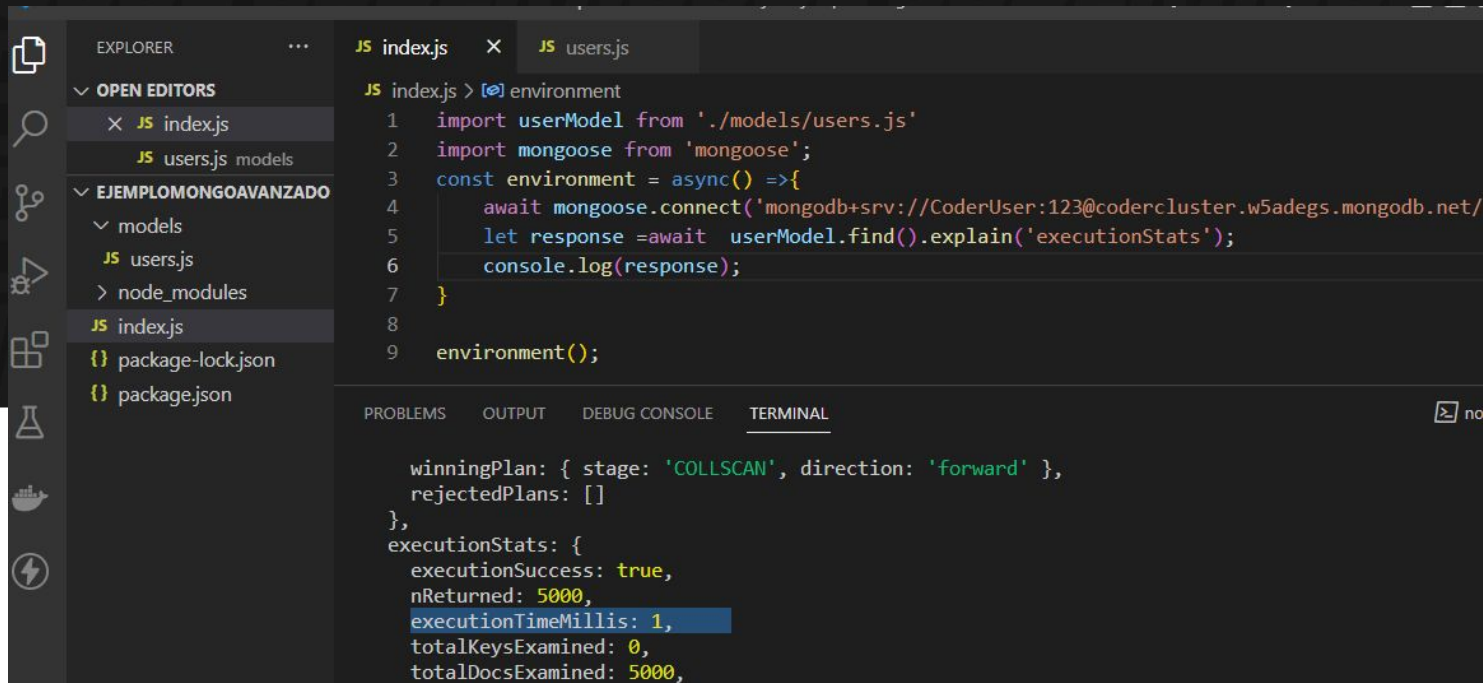


## Ejemplo en vivo

El profesor tendrá importado en su base de datos 5000 usuarios en una colección “users”, sobre éste, se realizarán ciertas búsquedas con y sin filtros.

**Se analizará el flujo y utilización de un índice para comparar resultados.**

# Ejemplo: Archivo index para realizar la búsqueda



The screenshot shows the Visual Studio Code editor with the Explorer sidebar on the left. The Explorer sidebar shows the project structure: **OPEN EDITORS** (index.js, users.js), **EJEMPLOMONGOAVANZADO** (models, users.js, node\_modules, index.js, package-lock.json, package.json). The main editor area shows the `index.js` file with the following code:

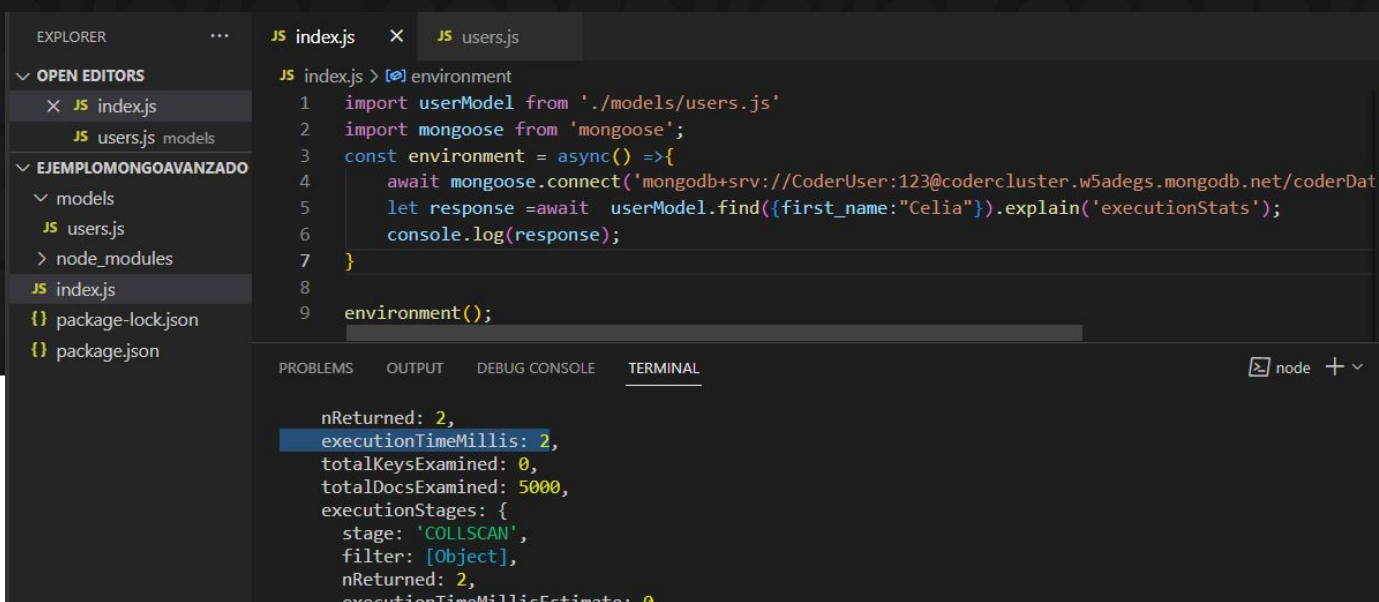
```
JS index.js > [?] environment
1 import userModel from './models/users.js'
2 import mongoose from 'mongoose';
3 const environment = async() =>{
4     await mongoose.connect('mongodb+srv://CoderUser:123@codercluster.w5adegs.mongodb.net/');
5     let response =await userModel.find().explain('executionStats');
6     console.log(response);
7 }
8
9 environment();
```

The bottom panel shows the **TERMINAL** output:

```
winningPlan: { stage: 'COLLSCAN', direction: 'forward' },
rejectedPlans: []
},
executionStats: {
  executionSuccess: true,
  nReturned: 5000,
  executionTimeMillis: 1,
  totalKeysExamined: 0,
  totalDocsExamined: 5000,
```

Tiempo de ejecución de la búsqueda para 5 mil usuarios: 1 milisegundo

# Ejemplo: Realizando búsqueda con filtro por nombre



The screenshot shows a VS Code editor with two files open: `index.js` and `users.js`. The `index.js` file contains a function `environment` that connects to a MongoDB database and performs a search for users with the first name "Celia". The terminal output shows the execution results, including the number of documents returned (2) and the execution time (2 milliseconds).

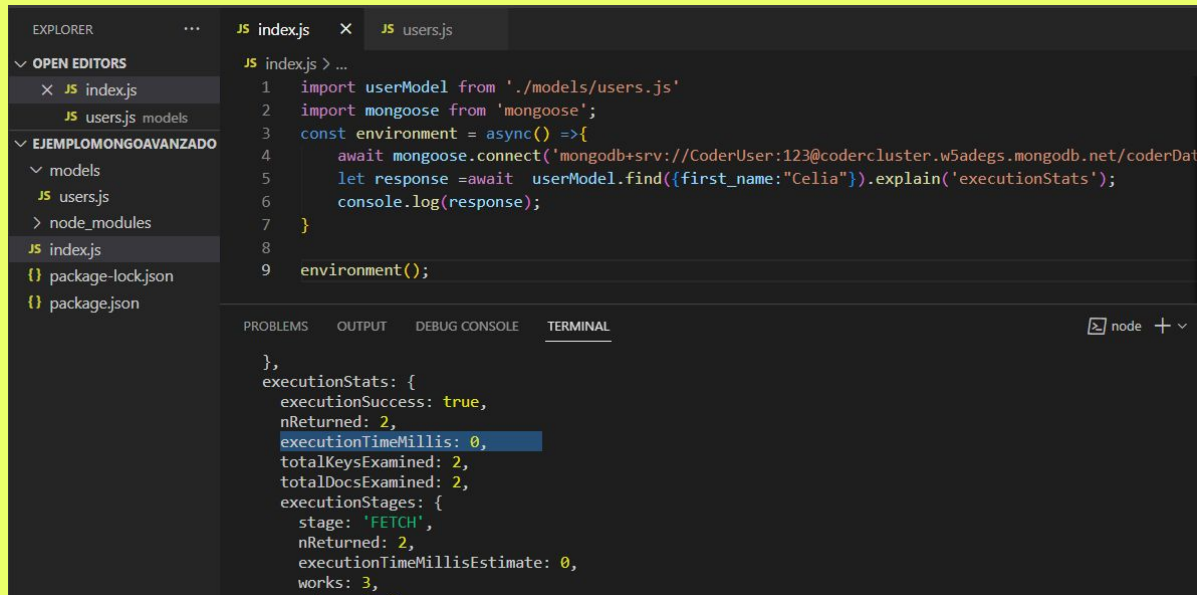
```
JS index.js > environment
1 import userModel from './models/users.js'
2 import mongoose from 'mongoose';
3 const environment = async() =>{
4   await mongoose.connect('mongodb+srv://CoderUser:123@codercluster.w5adegs.mongodb.net/coderDat
5   let response =await userModel.find({first_name:"Celia"}).explain('executionStats');
6   console.log(response);
7 }
8
9 environment();
```

```
nReturned: 2,
executionTimeMillis: 2,
totalKeysExamined: 0,
totalDocsExamined: 5000,
executionStages: {
  stage: 'COLLSCAN',
  filter: [Object],
  nReturned: 2,
  executionTimeMillisEstimate: 0
```

Tiempo de ejecución de la búsqueda para 5 mil usuarios buscando por nombre:  
2 milisegundos



# Ejemplo: Realizando búsqueda indexada



```
EXPLORER  ...  JS index.js  JS users.js

v OPEN EDITORS
  x JS index.js
    JS users.js models
v EJEMPLOMONGOAVANZADO
  v models
    JS users.js
  > node_modules
  JS index.js
  {} package-lock.json
  {} package.json

JS index.js > ...
1 import userModel from './models/users.js'
2 import mongoose from 'mongoose';
3 const environment = async() =>{
4   await mongoose.connect('mongodb+srv://CoderUser:123@codercluster.w5adegs.mongodb.net/coderDat
5   let response =await userModel.find({first_name:"Celia"}).explain('executionStats');
6   console.log(response);
7 }
8
9 environment();

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
node + v

},
executionStats: {
  executionSuccess: true,
  nReturned: 2,
  executionTimeMillis: 0,
  totalKeysExamined: 2,
  totalDocsExamined: 2,
  executionStages: {
    stage: 'FETCH',
    nReturned: 2,
    executionTimeMillisEstimate: 0,
    works: 3,
```

Tiempo de ejecución de la búsqueda para 5 mil usuarios buscando por nombre: 0 milisegundos (¡Tan bajo que no fue perceptible!)

# ¡Importante!

Un índice no debe ser utilizado en todos los campos, sólo deben ser utilizados en los campos que sepamos tienen repercusión en nuestras búsquedas.

Colocar un índice en cada campo de cada documento, significa alentar procesos de escritura en cada insert, así también como generar un almacenamiento adicional e innecesario en la base de datos.

# Otros tipos de índice

Otras de las indexaciones más comunes son:

- ✓ Índices compuestos (compound): Se utiliza cuando requerimos utilizar más de una indexación y queremos definir el orden con el cual se realiza el ordenamiento (ordenando con 1 para ascendente y -1 para descendente, igual que un sort: { campo: 1 , campo: -1 }
- ✓ Índice de múltiple llave (multikey): Se utiliza cuando requerimos hacer una indexación de valores que se encuentran de manera interna en un array.
- ✓ Índice de texto (text): Se utiliza para poder basarse en búsquedas de palabras “específicas” con el fin de poder tomar referencia de un texto a partir de dichas palabras.
- ✓ Índice geoespacial (geospatial): Se utiliza para almacenar data geoespacial de dos coordenadas, utiliza una esfera 2d para poder trabajar los datos.



# Actividad colaborativa

## Análisis de indexación

¿Qué índices colocar en cada documento?  
¡Revisaremos diferentes casos para practicar lo visto hasta ahora!

Duración: **15-20 minutos**



# Break

¡10 minutos y volvemos!

# Manejo de populations en Mongo

# Populations

## ¿Obtener data dentro de la data?

Una population implica obtener un documento referenciado dentro de otro documento, con el fin de obtener ambos en una sola búsqueda.

Consiste en almacenar el id de un documento, como propiedad de otro documento. A esto se le conoce como "referencia".

Populate hace referencia a "poblar" de un id a un documento completo. (referencia a la población humana)



1 doc → 2 docs → 4 docs → 5 docs → ...

# Funcionamiento

1. Al insertar un documento de tipo usuario que adopta mascotas, éste se crea con un ObjectId:

```
{  
  name: Mauricio,  
  pets: [ ]  
  _id: ObjectId("aaaaaaaa")  
}
```

2. Supongamos que ahora creamos dos mascotas, las cuales también tendrán su ObjectId:

```
{  
  name: Orejitas  
  _id:ObjectId("bbbbbbbbb")  
},  
{  
  name: Patitas  
  _id: ObjectId("ccccccccc")  
}
```



# Funcionamiento

3. Ahora, si este usuario quisiera “adoptar” las últimas dos mascotas, podríamos decir que estas mascotas deberían estar dentro del array “pets” del usuario. Pero NO agregamos todo el documento. sólo el `_id` que usaremos como “referencia” del documento

```
{  
  name: Mauricio,  
  pets: [ {pet:ObjectId("bbbbbbbbb")}, {pet:ObjectId("ccccccccc")}]  
  _id: ObjectId("aaaaaaaaa")  
}
```

# Funcionamiento

4. La próxima vez que hagamos la búsqueda del usuario, podemos programarlo para que no sólo obtenga al usuario, sino sus referencias de mascotas también.

El resultado final será la combinación del Documento usuario, más los documentos correspondientes a cada mascota

```
{
  _id: ObjectId("aaaaaaaaaa"),
  name: "Mauricio",
  pets: [
    {
      name: "Orejitas",
      _id: ObjectId("bbbbbbbbbbb")
    },
    {
      name: "Patitas",
      _id: ObjectId("ccccccccccc")
    }
  ]
}
```

# Populations en Mongoose

# Algunas cosas a considerar antes de comenzar con su uso

- ✓ populate es un método propio de mongoose, por lo que tenemos que instalarlo.
- ✓ Hay que tener siempre claro el nombre de la propiedad dentro del objeto, así también como la referencia de la colección, para poder hacer un populate efectivo.
- ✓ Recuerda no guardar directamente el valor a referenciar en el `_id`, sino asignarle otro nombre (se profundizará en el ejemplo)

# ¡Importante!

Una population es un puente entre dos documentos, como una relación unidireccional.

¡Jamás hagas una population bidireccional! Esto ocasionará que uno llame a otro, otro a uno, uno a otro, otro a uno... etc.

Por ejemplo, ¿qué pasaría si nuestro documento de cursos tuviera referencia del estudiante en su arreglo "students"?

# Configurando una population por default

# Middlewares para mongoose

Como habrás notado, para poder “poblar” el resultado de la operación `find()` del estudiante y obtener los cursos, fue necesario llamar a “`populate`” después de la operación.

Sin embargo, tener que colocar el `populate` puede resultar molesto si utilizamos constantemente el modelo de estudiante.

Mongoose tiene la posibilidad de definir “middlewares” para sus documentos y modelos con el fin de automatizar operaciones que consideremos recurrentes.

En esta ocasión utilizaremos un middleware conocido como “pre”

¿Preguntas?



**Opina y valora**  
**esta clase**

**Muchas gracias.**