



DESARROLLO *WEB*

FULL STACK

NIVEL BÁSICO

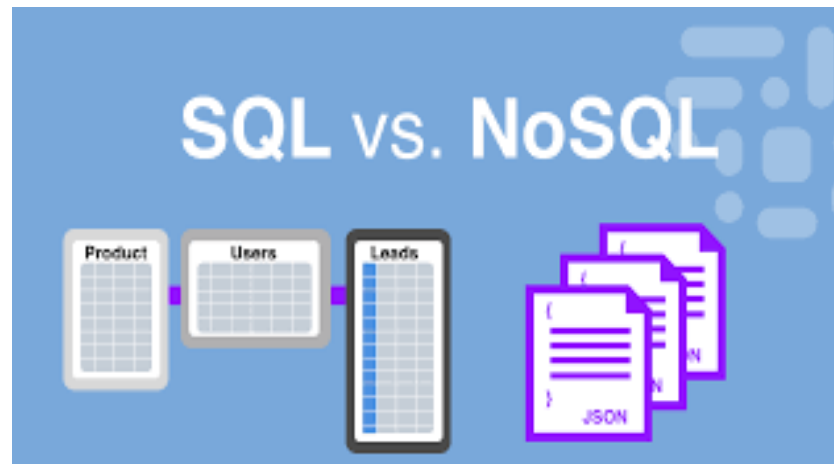
**Introducción Conexión a Bases de Datos con
Express.js**

Introducción Conexión a Bases de Datos con Express.js

Una **base de datos** es una colección organizada de información o datos que se almacenan y gestionan para facilitar su acceso, modificación y análisis. En términos sencillos, una base de datos es como una biblioteca digital donde se guarda información estructurada de manera que sea fácil de buscar y recuperar.

Conceptos Clave en Bases de Datos

1.Organización y Estructura: Los datos en una base de datos están organizados de manera que se puedan gestionar y manipular de forma eficiente. Dependiendo del tipo de base de datos, los datos se pueden organizar en **tablas** (en bases de datos relacionales) o en **colecciones** (en bases de datos NoSQL como MongoDB).



Introducción Conexión a Bases de Datos con Express.js

Propósito de una Base de Datos: Facilitar el almacenamiento, acceso y modificación de grandes volúmenes de datos de forma rápida y segura. Las bases de datos son fundamentales en aplicaciones como sitios web, aplicaciones móviles, y sistemas empresariales donde se necesita gestionar información de manera continua y en tiempo real.

Gestión de Datos: Las bases de datos permiten insertar nuevos datos, consultar información existente, actualizar datos y eliminar registros. Estas operaciones básicas son conocidas como operaciones CRUD (Create, Read, Update, Delete).

Sistema de Gestión de Bases de Datos (DBMS): Una base de datos generalmente se gestiona mediante un sistema de gestión de bases de datos (DBMS por sus siglas en inglés), que es un software que permite interactuar con los datos de manera segura y eficiente. Ejemplos de DBMS son MySQL, PostgreSQL, MongoDB y SQLite.

Introducción Conexión a Bases de Datos con Express.js

Tipos de Bases de Datos

1. Bases de Datos Relacionales (SQL):

- Almacenan los datos en tablas, con filas y columnas. Cada fila representa un registro y cada columna representa un campo de información. Ejemplos: MySQL, PostgreSQL, Oracle.

2. Bases de Datos No Relacionales (NoSQL):

- Utilizan estructuras de datos más flexibles como documentos, pares clave-valor o grafos. Son especialmente útiles para grandes volúmenes de datos que requieren alta escalabilidad y flexibilidad. Ejemplo: MongoDB (almacena datos en formato JSON).

¿Por Qué Son Importantes las Bases de Datos?

Las bases de datos permiten que las aplicaciones y sistemas almacenen y accedan a grandes cantidades de datos sin perder eficiencia.

Por ejemplo:

- Un sitio de comercio electrónico necesita una base de datos para almacenar información sobre productos, clientes y pedidos.
- Una aplicación de redes sociales usa una base de datos para guardar perfiles de usuario, publicaciones y comentarios.

Introducción a MongoDB

MongoDB es una base de datos NoSQL orientada a documentos, diseñada para almacenar grandes cantidades de datos de manera flexible y escalable. A diferencia de las bases de datos relacionales (como MySQL o PostgreSQL) que organizan la información en tablas y filas, MongoDB almacena los datos en **documentos JSON** (JavaScript Object Notation), que se organizan en **colecciones**. Esto permite que los datos tengan una estructura dinámica, lo que significa que cada documento puede tener una estructura diferente.

Características principales de MongoDB:

- **Orientado a documentos:** Almacena datos en documentos similares a JSON.
- **Escalabilidad horizontal:** Puede distribuir datos en múltiples servidores.
- **Flexibilidad:** No requiere un esquema fijo.
- **Consultas ricas:** Soporta consultas complejas y operaciones de agregación.
- **Alta disponibilidad:** Replicación automática de datos.

Conceptos clave en MongoDB:

- **Colección:** Equivalente a una tabla en bases de datos relacionales. Contiene documentos.
- **Documento:** Unidad básica de datos en MongoDB. Es un conjunto de pares clave-valor (similar a un objeto JSON).
- **Campo:** Par clave-valor dentro de un documento.
- **_id:** Identificador único de cada documento (similar a una clave primaria).

Principales Conceptos en MongoDB

Documentos:

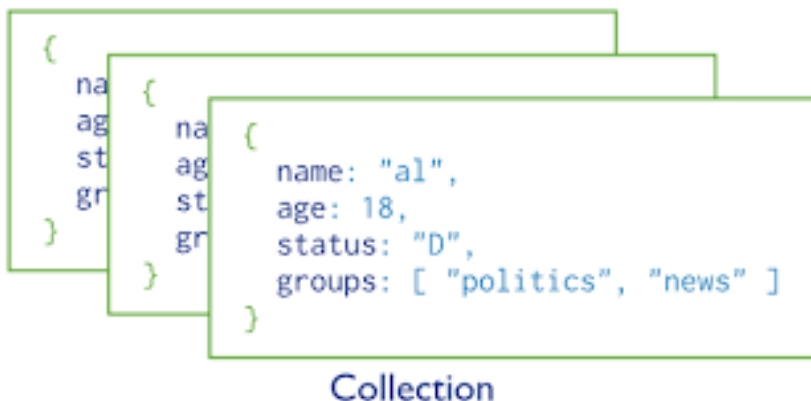
- La unidad básica de datos en MongoDB es un documento, que es un objeto JSON. Un documento en MongoDB puede contener datos anidados y listas, lo que permite representar estructuras complejas en un solo registro.
- Ejemplo de un documento en MongoDB:

```
{
  "nombre": "Juan Pérez",
  "edad": 30,
  "dirección": {
    "calle": "Av. Central",
    "ciudad": "Bogotá",
    "códigoPostal": "110001"
  },
  "aficiones": ["leer", "viajar", "ciclismo"]
}
```

Principales Conceptos en MongoDB

Colecciones:

- En lugar de tablas, MongoDB usa colecciones para agrupar documentos. Una colección es un conjunto de documentos que pueden tener estructuras similares, pero no necesariamente idénticas.
- Ejemplo: Una colección de usuarios podría contener documentos con información sobre cada usuario.



```
{
  "nombre": "Pepito",
  "direccion": "calle 1",
  "telefonos": [387222777, 0387754891],
  "telefono_emergencia": 3281597538
},
{
  "nombre": "Alberto",
  "direccion": "calle 2",
  "telefonos": 387222666
}
```

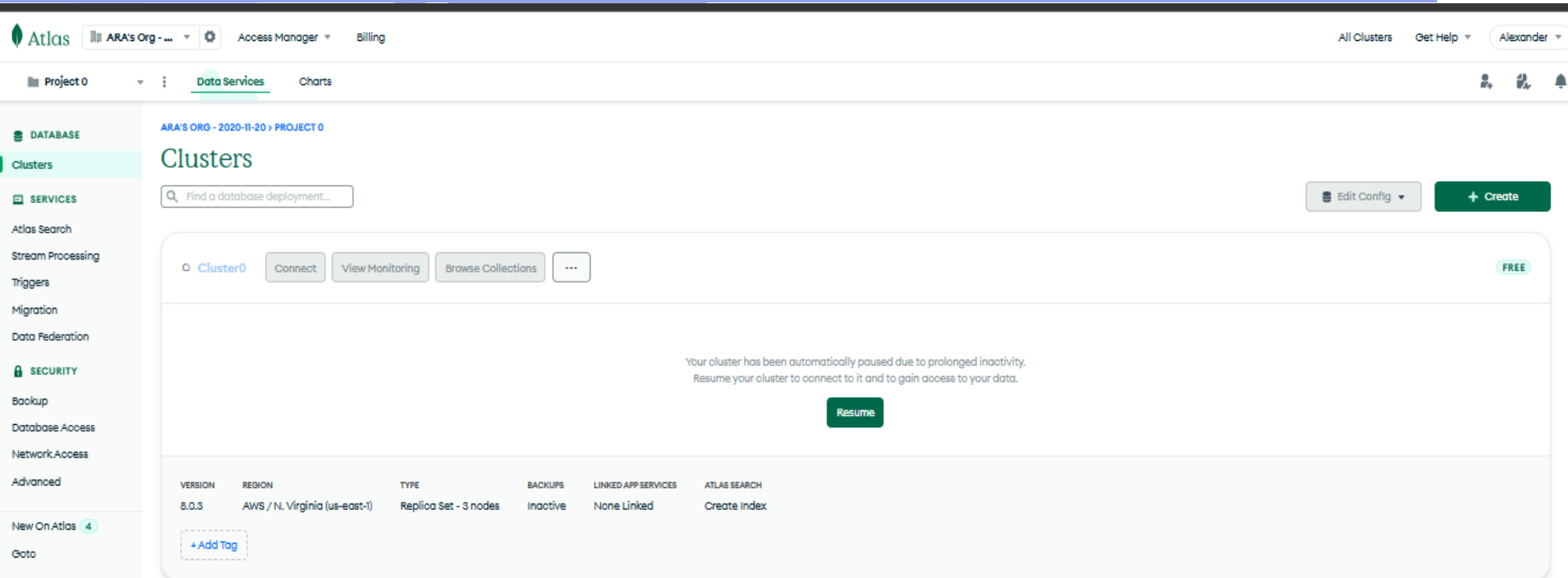
Principales Conceptos en MongoDB

_id Campo:

Cada documento en MongoDB contiene un campo especial llamado `_id`, que es un identificador único para el documento dentro de una colección. MongoDB asigna automáticamente un valor `_id` si no se especifica uno.



Creación de Cuenta en MongoDB Atlas



The screenshot displays the MongoDB Atlas interface. The top navigation bar includes the Atlas logo, organization name 'ARA's Org', and links for 'Access Manager' and 'Billing'. The right side shows 'All Clusters', 'Get Help', and a user profile 'Alexander'. The left sidebar contains a 'DATABASE' section with 'Clusters' highlighted, and a 'SERVICES' section with various options like 'Atlas Search', 'Stream Processing', 'Triggers', 'Migration', and 'Data Federation'. Below these are 'SECURITY' options like 'Backup', 'Database Access', and 'Network Access', followed by 'Advanced' and 'New On Atlas' (with a badge '4').

The main content area is titled 'Clusters' and shows a search bar 'Find a database deployment...'. Below the search bar, there's a card for 'Cluster0' with buttons 'Connect', 'View Monitoring', 'Browse Collections', and a 'FREE' badge. A message states: 'Your cluster has been automatically paused due to prolonged inactivity. Resume your cluster to connect to it and to gain access to your data.' with a 'Resume' button.

Below the cluster card is a table with the following data:

VERSION	REGION	TYPE	BACKUPS	LINKED APP SERVICES	ATLAS SEARCH
5.0.3	AWS / N. Virginia (us-east-1)	Replica Set - 3 nodes	Inactive	None Linked	Create Index

There is also a '+Add Tag' button in a dashed box below the table.

Conexión a MongoDB con Express.js

Introducción a Mongoose

¿Qué es Mongoose?

Mongoose es una librería de Node.js que proporciona una capa de abstracción sobre MongoDB. Permite definir **esquemas** y **modelos** para interactuar con la base de datos de manera más sencilla y estructurada.

Pasos para conectar Express.js a MongoDB:

1. Instalar dependencias:

Necesitas instalar `mongoose` (una librería para interactuar con MongoDB) y `express`.

```
bash
```

[Copy](#)

```
npm install express mongoose
```

Configurar la conexión:

Configurar la conexión:

En tu archivo `app.js` o `server.js`, configura la conexión a MongoDB usando Mongoose.

javascript

Copy

```
const express = require('express');
const mongoose = require('mongoose');
const app = express();

// Conectar a MongoDB
mongoose.connect('mongodb://localhost:27017/nombre_de_tu_base_de_datos', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
})
.then(() => console.log('Conexión a MongoDB establecida'))
.catch(err => console.error('Error conectando a MongoDB:', err));

// Iniciar el servidor
const PORT = 3000;
app.listen(PORT, () => {
  console.log(`Servidor corriendo en http://localhost:${PORT}`);
});
```

Manejo de errores y eventos:

Manejo de errores y eventos:

Mongoose emite eventos que puedes escuchar para manejar la conexión:

javascript

Copy

```
mongoose.connection.on('connected', () => {
  console.log('Mongoose conectado a la base de datos');
});

mongoose.connection.on('error', (err) => {
  console.error('Error en la conexión de Mongoose:', err);
});

mongoose.connection.on('disconnected', () => {
  console.log('Mongoose desconectado');
});
```

Conceptos clave en Mongoose:

- **Esquema:** Define la estructura de los documentos en una colección.
- **Modelo:** Representa una colección en la base de datos. Se crea a partir de un esquema.
- **Métodos:** Funciones que puedes agregar a los modelos para realizar operaciones personalizadas.

Definir un esquema y modelo en Mongoose:

javascript

Copy

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

// Definir un esquema
const usuarioSchema = new mongoose.Schema({
  nombre: { type: String, required: true },
  edad: { type: Number, min: 18, max: 100 },
  email: { type: String, required: true, unique: true },
  fechaRegistro: { type: Date, default: Date.now }
});

// Crear un modelo
const Usuario = mongoose.model('Usuario', usuarioSchema);

// Exportar el modelo
module.exports = Usuario;
```

Operaciones básicas con Mongoose:

Ejemplo completo: API REST con Express.js y Mongoose

```
/proyecto
├── public/
│   ├── index.html
│   ├── styles.css
│   └── script.js
├── models/
│   └── Usuario.js
├── routes/
│   └── usuarios.js
├── app.js
├── .env
└── package.json
```

Instalar :

Express

Mongoose

dotenv

Código del modelo (models/Usuario.js):

```
/proyecto
├── models/
│   └── Usuario.js
├── routes/
│   └── usuarios.js
├── app.js
└── package.json
```

javascript

Copy

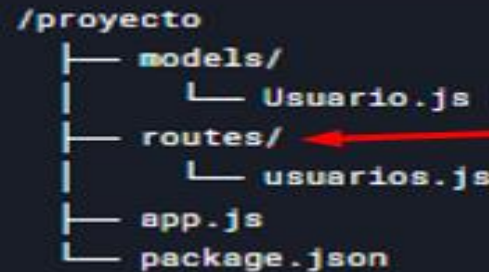
```
const mongoose = require('mongoose');

const usuarioSchema = new mongoose.Schema({
  nombre: { type: String, required: true },
  edad: { type: Number, min: 18, max: 100 },
  email: { type: String, required: true, unique: true },
  fechaRegistro: { type: Date, default: Date.now }
});

module.exports = mongoose.model('Usuario', usuarioSchema);
```

Código de las Rutas (routes/usuarios.js)

Este archivo define las rutas para crear, mostrar, actualizar y eliminar usuarios.



```
/proyecto
├── models/
│   └── Usuario.js
├── routes/
│   └── usuarios.js
├── app.js
└── package.json
```

El diagrama muestra la estructura de carpetas de un proyecto. Las carpetas 'models/' y 'routes/' están resaltadas con un fondo azul. Dentro de 'models/' hay un archivo 'Usuario.js'. Dentro de 'routes/' hay un archivo 'usuarios.js'. El archivo 'usuarios.js' en 'routes/' está señalado con una flecha roja.

```
const express = require('express');
const Usuario = require('../models/Usuario');
const router = express.Router();

// Crear un usuario (POST)
// Crear un usuario (POST)
router.post('/', async (req, res) => {
  try {
    const usuario = new Usuario(req.body); // Crear un nuevo usuario con los datos del body
    await usuario.save(); // Guardar el usuario en la base de datos
    res.status(201).send(usuario); // Responder con el usuario creado
  } catch (err) {
    res.status(400).send({ error: err.message }); // Manejar errores de validación o duplicados
  }
});
```


Código de las Rutas (routes/usuarios.js)

```
// Obtener todos los usuarios (GET)
router.get('/', async (req, res) => {
  try {
    const usuarios = await Usuario.find(); // Buscar todos los usuarios
    res.send(usuarios); // Responder con la lista de usuarios
  } catch (err) {
    res.status(500).send({ error: 'Error al obtener los usuarios' }); // Manejar errores del servidor
  }
});
```

```
// Obtener un usuario por ID (GET)
router.get('/:id', async (req, res) => {
  try {
    const usuario = await Usuario.findById(req.params.id); // Buscar usuario por ID
    if (!usuario) {
      return res.status(404).send({ error: 'Usuario no encontrado' }); // Si no existe, devolver 404
    }
    res.send(usuario); // Responder con el usuario encontrado
  } catch (err) {
    res.status(500).send({ error: 'Error al obtener el usuario' }); // Manejar errores del servidor
  }
});
```

Código de las Rutas (routes/usuarios.js)

```
// Actualizar un usuario por ID (PUT)
router.put('/:id', async (req, res) => {
  try {
    const usuario = await Usuario.findByIdAndUpdate(req.params.id, req.body, {
      new: true, // Devolver el usuario actualizado
      runValidators: true // Validar los datos antes de actualizar
    });
    if (!usuario) {
      return res.status(404).send({ error: 'Usuario no encontrado' }); // Si no existe, devolver
404
    }
    res.send(usuario); // Responder con el usuario actualizado
  } catch (err) {
    res.status(400).send({ error: err.message }); // Manejar errores de validación o duplicados
  }
});
```

Código de las Rutas (routes/usuarios.js)

```
// Eliminar un usuario por ID (DELETE)
router.delete('/:id', async (req, res) => {
  try {
    const usuario = await Usuario.findByIdAndDelete(req.params.id); // Buscar y eliminar usuario
    por ID
    if (!usuario) {
      return res.status(404).send({ error: 'Usuario no encontrado' }); // Si no existe, devolver
      404
    }
    res.send({ message: 'Usuario eliminado correctamente' }); // Responder con un mensaje de éxi
    to
  } catch (err) {
    res.status(500).send({ error: 'Error al eliminar el usuario' }); // Manejar errores del serv
    idor
  }
});

module.exports = router;
```

Código Principal (app.js)

Este archivo configura la aplicación Express.js, conecta a MongoDB y define las rutas.

```
/proyecto
├── models/
│   └── Usuario.js
├── routes/
│   └── usuarios.js
├── app.js ←
└── package.json
```

Crea un archivo `.env` en la raíz del proyecto con la siguiente línea:

```
MONGO_URI=mongodb://localhost:27017/nombre_de_tu_base_de_datos
PORT=3000
```

```
const express = require('express');
const mongoose = require('mongoose');
const usuarioRoutes = require('./routes/usuarios');
const dotenv = require('dotenv'); // Importar dotenv para manejar variables de entorno
const app = express();
```

```
// Cargar variables de entorno desde .env
dotenv.config();
```

```
// Middleware para parsear JSON
app.use(express.json());
```

Código Principal (app.js)

```
// Conectar a MongoDB usando MONGO_URI desde .env
mongoose.connect(process.env.MONGO_URI, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
})

.then(() => console.log('Conexión a MongoDB establecida'))
.catch(err => console.error('Error conectando a MongoDB:', err));

// Rutas
app.use('/usuarios', usuarioRoutes);

// Ruta de prueba
app.get('/', (req, res) => {
  res.send('API REST de usuarios funcionando');
});

// Manejo de errores global
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).send({ error: 'Algo salió mal en el servidor' });
});

// Iniciar el servidor
const PORT = process.env.PORT || 3000; // Usar el puerto de .env o 3000 por defecto
app.listen(PORT, () => {
  console.log(`Servidor corriendo en http://localhost:${PORT}`);
});
```

Iniciar el servidor:

archivos estáticos:

Revisamos los archivos estáticos (HTML, CSS, JS)

Ejecuta el servidor:

```
bash
```

```
node app.js
```

Pruebas

Crear usuario:

- Completa el formulario y haz clic en **Crear Usuario**.

Editar usuario:

- Haz clic en **Editar** junto al usuario que deseas modificar. Ingresa los nuevos datos en los cuadros de diálogo.

Eliminar usuario:

- Haz clic en **Eliminar** junto al usuario que deseas eliminar.