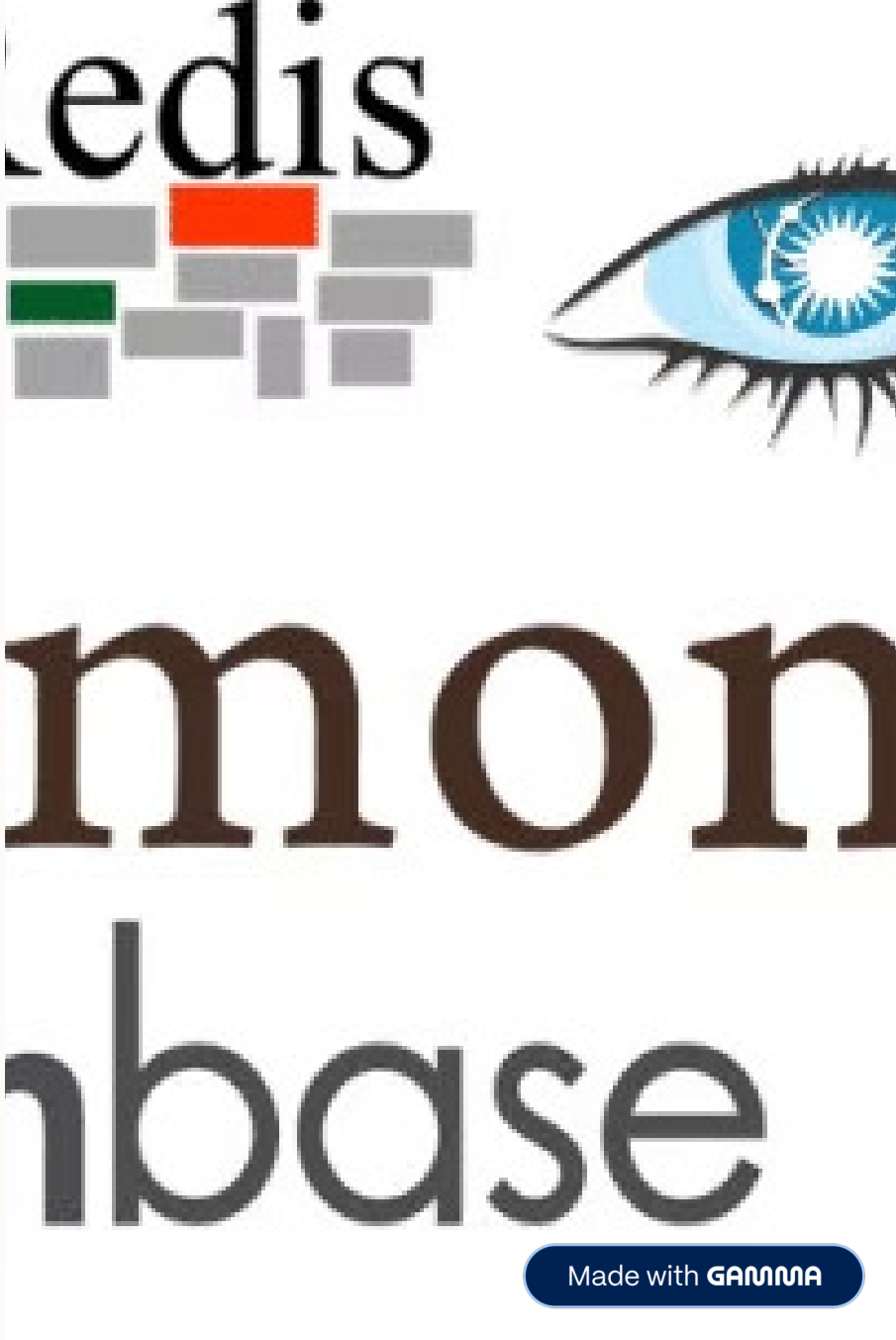
Introdução ao NoSQL com MongoDB, Mongoose e Express

O objetivo é que, ao final, todos aqui se sintam confortáveis para configurar, modelar e construir uma aplicação completa usando MongoDB, a tecnologia líder nesse universo. Vamos começar?



Fundamentos e Configuração Minutos

Revisão Rápida: O Modelo Relacional (SQL)

Pensem no que a maioria de nós já conhece: bancos de dados SQL, como MySQL ou PostgreSQL. A melhor analogia é um conjunto de planilhas do Excel que se conectam umas com as outras.

Tabelas

Cada "planilha" é uma tabela (ex: Usuarios, Pedidos).

Linhas

Cada linha na planilha é um registro único (ex: um usuário específico).

Colunas

Cada coluna define um atributo com um tipo fixo (ex: nome é sempre texto, idade é sempre um número).

Schema Rígido: A estrutura é definida ANTES de inserirmos qualquer dado. Se você quiser adicionar uma nova coluna, precisa alterar a tabela inteira com um ALTER TABLE.

JOINs: A grande força do SQL. Se você quer os dados de um usuário e todos os pedidos que ele já fez, você usa um JOIN para "cruzar" as informações das tabelas Usuarios e Pedidos usando uma chave comum (como usuario_id).

O SQL é fantástico para dados bem estruturados e quando a consistência dos dados é a prioridade máxima (transações bancárias, sistemas de controle de estoque, etc.). Mas o mundo da tecnologia mudou...

As Limitações do SQL e a Ascensão do NoSQL

No início dos anos 2000, com a explosão da internet, surgiram novos desafios que o modelo relacional não foi projetado para resolver. Pensem nos 3 V's do Big Data:

Volume

A quantidade de dados gerada se tornou astronômica. Pense no feed do Facebook, nos vídeos do YouTube, nos dados de sensores de IoT.

Velocidade

Esses dados são gerados e precisam ser acessados em altíssima velocidade. Milissegundos importam.

Variedade

Os dados não são mais apenas tabelas arrumadinhas. Temos textos, imagens, vídeos, JSONs, logs. São dados semi-estruturados ou não-estruturados.

Isso expôs algumas limitações do SQL:

- **Escalabilidade:** Para um banco SQL aguentar mais carga, geralmente fazemos um scale-up (escalonamento vertical): compramos um servidor mais caro, com mais RAM, mais CPU. Isso é caro e tem um limite. O NoSQL foi pensado para o scale-out (escalonamento horizontal): distribuímos a carga em vários servidores mais baratos. É como ter uma frota de 100 carros populares em vez de um único super-caminhão.
- **Flexibilidade:** Em uma startup, o modelo de negócio muda rápido. Hoje você armazena o email do usuário. Amanhã, quer adicionar login_com_google e login_com_facebook, mas não para todos os usuários. Em SQL, isso exige um ALTER TABLE, o que pode ser lento e complexo. No NoSQL, você simplesmente adiciona os novos campos aos novos usuários.
- **Performance:** Aquele JOIN que conecta 5, 10 tabelas pode se tornar o grande vilão da performance em sistemas com milhões de acessos.

É nesse cenário que o NoSQL ("Not Only SQL") surge, não para substituir o SQL, mas para oferecer uma alternativa melhor para certos tipos de problemas.

Vantagens e Desvantagens do NoSQL

Vantagens

Schema Flexível: Adicione campos a novos documentos sem afetar os antigos. Ideal para desenvolvimento ágil.

Escalabilidade Horizontal: Mais barato e eficiente para escalar, distribuindo dados em um cluster de máquinas.

Alta Performance: Otimizado para um grande volume de leituras e escritas simples. A ausência de JOINs ajuda.

Modelos de Dados Específicos: Há um tipo de banco NoSQL para cada necessidade:

• Documentos (MongoDB): Ótimo para a maioria das aplicações web, blogs, catálogos de produtos.

• Chave-Valor (Redis): Perfeito para caching, sessões de usuário. Extremamente rápido.

• Colunar (Cassandra): Ideal para séries temporais, telemetria, onde você lê colunas específicas.

• Grafos (Neo4j): Imbatível para redes sociais, sistemas de recomendação, detecção de fraudes.

Desvantagens

- **Menos Consistência (às vezes):** Muitos priorizam a "Consistência Eventual". A informação pode levar um momento para se propagar por todo o sistema. Não é ideal para transações financeiras, por exemplo.
- **Sem JOINs Nativos:** Exige que você pense diferente sobre a modelagem. Muitas vezes, dados são duplicados (desnormalizados) para melhorar a performance de leitura.
- **Ferramentas de Análise (BI):** O ecossistema SQL é historicamente mais maduro para relatórios complexos, embora o MongoDB tenha avançado muito nisso.

MongoDB e o Modelo de Documentos

O que é o MongoDB?

É o banco de dados NoSQL mais popular do mundo, e ele usa o modelo de documentos. A ideia é simples: armazenar os dados da forma como eles são usados na aplicação, geralmente em um formato muito parecido com JSON.

Vamos fazer uma tradução direta do mundo SQL para o MongoDB:

Conceito SQL	Conceito MongoDB
Database (Banco de Dados)	Database
Table (Tabela)	Collection (Coleção)
Row (Linha / Registro)	Document (Documento)
Column (Coluna)	Field (Campo)
JOIN	Embedded Documents (Documentos Embutidos)

O Documento BSON

No nosso código, escrevemos em JSON, mas o MongoDB armazena os dados em BSON (Binary JSON). Pense no BSON como um JSON com superpoderes: ele é binário (mais rápido para o banco processar) e suporta mais tipos de dados, como Date, ObjectId (um ID único gerado pelo Mongo) e dados binários.

Vejamos um exemplo. Em SQL, um aluno e suas notas poderiam estar em tabelas separadas. No MongoDB, eles vivem juntos em um único documento, o que torna a leitura muito mais rápida.

```
// Documento de um Aluno na collection 'alunos'
{
  "_id": ObjectId("6324b1c2d3e4f5a6b7c8d9e0"), // Gerado automaticamente pelo Mongo
  "nomeCompleto": "Summer Smith",
  "matricula": "202510123",
  "curso": "Ciência da Computação",
  "dataNascimento": ISODate("2004-05-15T00:00:00Z"), // Tipo de dado de data
  "ativo": true,
  "notas": [8.5, 9.0, 7.8], // Um array de números
  "endereco": { // Um sub-documento ou documento embutido
    "rua": "Av. Principal",
    "numero": 100,
    "cidade": "Pelotas"
  }
}
```

Notem a beleza disso: para pegar todas as informações da Ana, eu faço uma única busca nesse documento. Não preciso de JOIN.

Módulo 3: Preparando o Ambiente

Temos duas formas de usar o MongoDB. Vou mostrar as duas, mas recomendo fortemente que todos usem a Opção 2 (Atlas) para a nossa aula, pois evita qualquer problema de instalação local.

Opção 1: MongoDB Local

01

Download

Acesse o site oficial do MongoDB e baixe o "MongoDB Community Server" para o seu sistema operacional.

02

Instalação

Siga o instalador padrão. No Windows, ele geralmente instala como um serviço que inicia com o sistema. No Mac/Linux, você pode usar gerenciadores como o Homebrew (brew install mongodb-community) ou apt.

03

Execução

Após instalar, você precisa iniciar o serviço do MongoDB, geralmente com o comando mongod.

04

GUI - MongoDB Compass

Recomendo fortemente baixar o Compass. É uma ferramenta visual que permite ver seus bancos, collections e documentos, além de executar consultas, sem precisar usar o terminal o tempo todo.

Opção 2: MongoDB Atlas

O Atlas é a plataforma de nuvem do MongoDB. Vamos criar um banco de dados gratuito que podemos acessar de qualquer lugar.

1

Cadastro

Acesse cloud.mongodb.com e crie uma conta gratuita.

2

Novo Projeto

Crie um novo projeto e dê um nome a ele (ex: "Aula CRUD").

3

Criar Cluster

Clique em "Build a Database". Escolha a opção "Shared" (M0 Sandbox), que é gratuita para sempre.

4

Configuração do Cluster

- Provedor de Nuvem: Pode deixar o padrão (AWS).
- Região: Escolha uma perto de você (ex: sa-east-1 em São Paulo) para menor latência.
- Nome do Cluster: Pode deixar o padrão (Cluster0) ou dar um nome mais descritivo.

Clique em "Create Cluster". A criação pode levar de 3 a 5 minutos.

Construindo uma API

Setup do Projeto Node.js/Express

Vamos criar a estrutura do nosso projeto. Abram o terminal na pasta onde vocês guardam seus projetos.

```
mkdir api-tarefas
cd api-tarefas

npm init -y

npm install express mongoose dotenv
```

- **express:** Nosso framework web para criar o servidor e as rotas da API.
- **mongoose:** A biblioteca que fará a "ponte" entre nossa aplicação Node.js e o banco de dados MongoDB.

O que é um ODM? Introdução ao Mongoose

Mongoose é um ODM - Object Document Mapper.

Pensem nele como um tradutor fluente e muito prestativo. Nós, no nosso código JavaScript, gostamos de trabalhar com objetos. O MongoDB, por sua vez, trabalha com documentos BSON. O Mongoose fica no meio do caminho, traduzindo nossos objetos para documentos e vice-versa.

Por que usar o Mongoose em vez de falar direto com o driver do MongoDB?

Modelagem de Dados (Schemas)

Ele nos permite definir uma estrutura para nossos documentos. Podemos dizer "toda tarefa PRECISA ter um título que seja texto". Isso traz de volta um pouco da previsibilidade do mundo SQL para o mundo flexível do NoSQL.

Validação

Garante que os dados que tentamos salvar estão corretos, antes mesmo de enviá-los ao banco.

Middleware

Permite executar código antes ou depois de certas operações. Por exemplo, antes de salvar um usuário, podemos usar um middleware para criptografar a senha dele.

Facilidade de Uso

Fornece métodos simples e encadeáveis para realizar todas as operações do CRUD, como Tarefa.find({ concluida: true }).

Criando a Conexão com o Banco

Vamos organizar nosso projeto.

Crie um arquivo na raiz do projeto chamado `.env`. É aqui que vamos colocar nossa Connection String para mantê-la segura.

```
// .env
MONGO_URI=mongodb+srv://aulacruduser:SUA_SENHA_AQUI@cluster0.xxxxx.mongodb.net/minhas-tarefas?
retryWrites=true&w=majority
```

Importante: Não vai esquecer de substituir o SUA_SENHA_AQUI pela senha que você criou no Atlas.

Crie uma pasta `config` e dentro dela um arquivo `db.js`.

```
// config/db.js
const mongoose = require('mongoose');
require('dotenv').config(); // Carrega as variáveis do arquivo .env

const connectDB = async () => {
  try {
    // Tenta conectar ao MongoDB usando a URI do nosso arquivo .env
    const conn = await mongoose.connect(process.env.MONGO_URI);
    console.log(`MongoDB Conectado com Sucesso: ${conn.connection.host}`);
  } catch (error) {
    // Se houver um erro, exibe o erro e encerra a aplicação
    console.error(`Erro ao conectar com o MongoDB: ${error.message}`);
    process.exit(1);
  }
};

module.exports = connectDB;
```

Crie o arquivo principal do servidor, `server.js`.

```
// server.js
const express = require('express');
const connectDB = require('./config/db');

// Inicializa o Express
const app = express();

// Conecta ao Banco de Dados
connectDB();

// Middleware para o Express entender JSON vindo no corpo das requisições
app.use(express.json());

app.get('/', (req, res) => res.send('API de Tarefas Rodando!'));

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => console.log(`Servidor rodando na porta ${PORT}`));
```

Agora, no terminal, rode `node server.js`. Se tudo deu certo, você verá as mensagens "MongoDB Conectado com Sucesso!" e "Servidor rodando na porta 5000".

Modelar uma Coleção no MongoDB

Definindo o Schema

Agora que estamos conectados, vamos definir como será a "cara" de uma Tarefa no nosso banco.

Crie uma pasta `models` e dentro dela um arquivo `Tarefa.js`.

```
// models/Tarefa.js
const mongoose = require('mongoose');

const TarefaSchema = new mongoose.Schema({
 titulo: {
 type: String,
 required: [true, 'O título da tarefa é obrigatório.'], // Campo obrigatório
 trim: true // Remove espaços em branco do início e do fim do título
 },
 descricao: {
 type: String,
 required: false // Campo não obrigatório
 },
 concluida: {
 type: Boolean,
 default: false // Se não for informado, o valor padrão será 'false'
 },
 dataCriacao: {
 type: Date,
 default: Date.now // O valor padrão é a data/hora do momento da criação
 }
});

// Estamos dizendo ao Mongoose: "Crie um modelo chamado 'Tarefa' baseado no schema que acabamos de definir"
// O Mongoose, por convenção, criará uma collection no MongoDB chamada 'tarefas' (plural e minúsculas).
module.exports = mongoose.model('Tarefa', TarefaSchema);
```

Implementar um CRUD Completo

Esta é a hora da verdade! Vamos implementar cada uma das operações do CRUD. Usem uma ferramenta como o Postman ou Insomnia para testar cada endpoint que criarmos.

1

CREATE

POST /api/tarefas
Criar uma nova tarefa

2

READ

GET /api/tarefas
Listar todas as tarefas

3

UPDATE

PUT /api/tarefas/:id
Atualizar uma tarefa

4

DELETE

DELETE /api/tarefas/:id
Deletar uma tarefa

Encerramento e Próximos Passos

O que nós aprendemos hoje:

- Por que o NoSQL existe e quais problemas ele resolve.
- As vantagens e desvantagens em relação ao SQL.
- O que é o MongoDB, seu modelo de documentos e como usar comandos básicos.
- Como configurar um banco de dados na nuvem com o MongoDB Atlas, uma habilidade essencial no mercado hoje.
- O que é um ODM como o Mongoose e como ele facilita nossa vida.
- Como modelar nossos dados com Schemas para garantir consistência.
- E, finalmente, como implementar um CRUD completo (Create, Read, Update, Delete) usando Node.js, Express e Mongoose.

Isso é um alicerce muito sólido, mas a jornada não termina aqui. O que vocês podem explorar a seguir?

Consultas Avançadas

Aprendam a usar sort(), limit(), select() para filtrar e paginar resultados.

Índices

A ferramenta mais importante para otimizar a performance. Aprender a criar índices nos campos mais buscados pode fazer uma consulta levar milissegundos em vez de segundos.

Aggregation Framework

A ferramenta de análise de dados do MongoDB. É como ter um canivete suíço para transformar e agrupar seus dados, permitindo gerar relatórios complexos diretamente no banco.

Modelagem de Relacionamentos

Como lidar com dados que se relacionam? Estudem as duas abordagens principais: Embedding (embutir, como fizemos com o endereço do aluno) e Referencing (usar IDs para referenciar documentos em outras coleções).

Autenticação e Autorização

Protejam sua API usando estratégias como JWT (JSON Web Tokens).

Obrigado a todos pela participação e pela atenção. E lembrem-se sempre: Don´t be a Jerry!