

# Bancos de Dados

## NoSQL: Do Conceito ao MongoDB

UC-13 Desenvolvimento Back-End

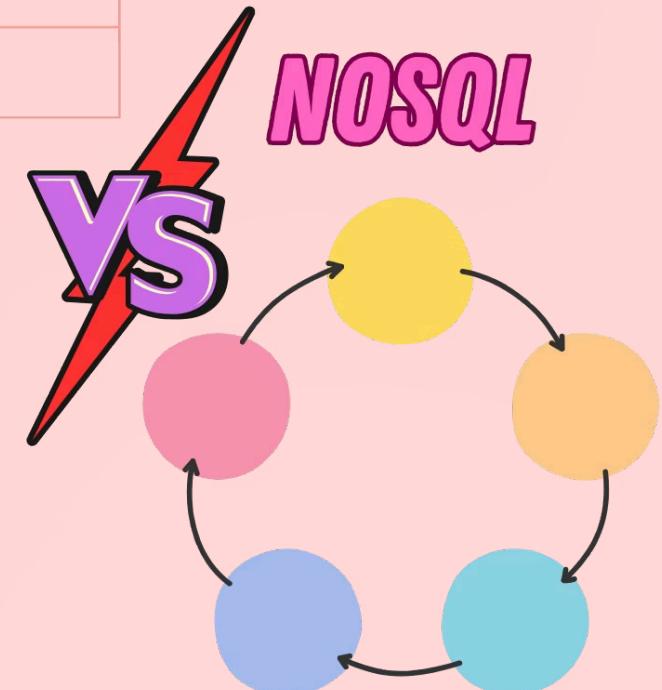
Curso Técnico em Desenvolvimento de  
Sistemas - Prof<sup>a</sup> Aline Timm

UniSenac Pelotas- RS

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?

SQL

# Fundamentos e Configuração

## Revisão do 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...

# A Virada de Jogo: O Surgimento do NoSQL

## História e Motivação

No início dos anos 2000, a internet explodiu com a chamada **Web 2.0**. Empresas como Google, Amazon e Facebook começaram a lidar com um volume de dados nunca antes visto, gerado por milhões de usuários simultaneamente. Eram dados de todos os tipos: posts, fotos, cliques, logs, etc.

Os bancos de dados relacionais começaram a mostrar suas limitações para esse novo cenário:

### Volume Massivo de Dados (Big Data)

Armazenar e consultar petabytes de dados em um único servidor SQL era inviável e caro.

### Variedade de Dados

Muitos dados não se encaixavam bem em tabelas rígidas. Eram dados não-estruturados ou semi-estruturados.

### Velocidade e Disponibilidade

Aplicações web modernas exigiam altíssima velocidade de leitura/escrita e precisavam estar sempre online (alta disponibilidade), mesmo que um servidor falhasse.

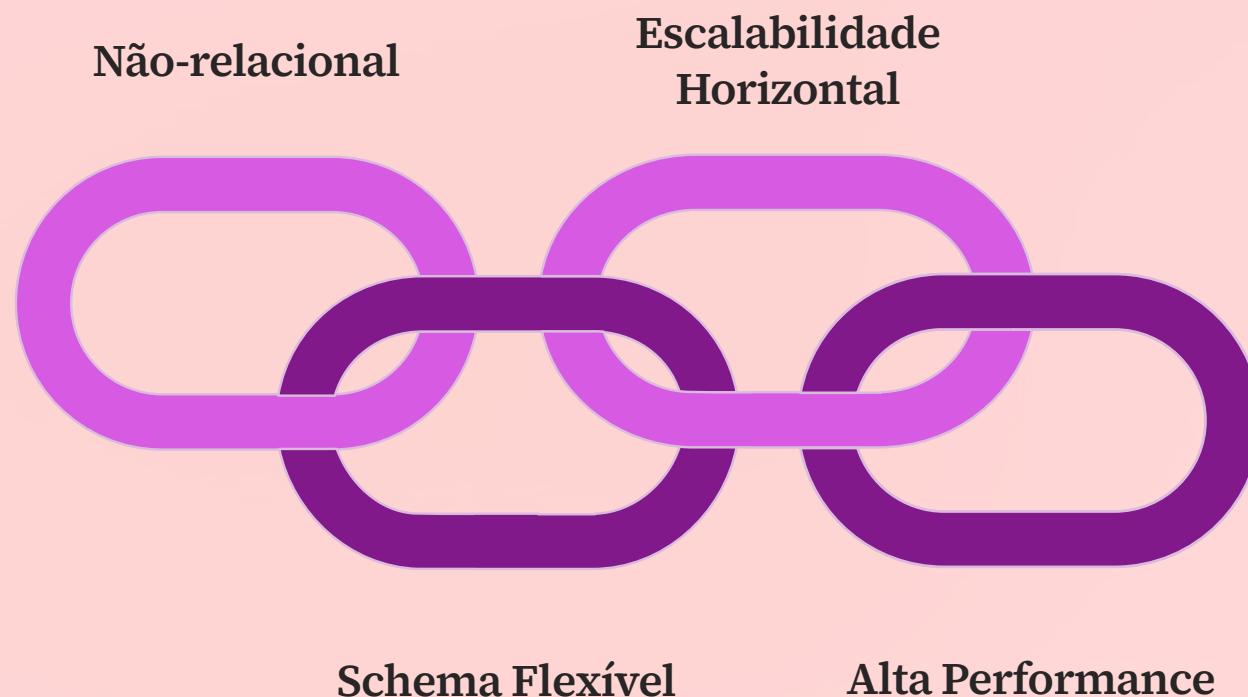
Foi nesse contexto que essas gigantes da tecnologia começaram a criar suas próprias soluções. O Google publicou o artigo sobre o **Bigtable** em 2006, e a Amazon sobre o **Dynamo** em 2007. Esses sistemas foram a base para o movimento que, em 2009, foi batizado de "**NoSQL**".

# O que Significa "NoSQL"?

Ao contrário do que parece, NoSQL não significa "Não ao SQL". A interpretação mais aceita hoje é "**"Not Only SQL"** (Não Apenas SQL). Isso significa que ele não veio para substituir os bancos relacionais, mas para oferecer uma alternativa para problemas que o SQL não resolve bem.

**As principais características de um banco de dados NoSQL são:**

- **Não-relacional:** Os dados não são armazenados em tabelas com relações complexas.
- **Schema Flexível:** Não é necessário definir uma estrutura rígida para os dados antes de inseri-los. Isso permite agilidade no desenvolvimento.
- **Escalabilidade Horizontal:** São projetados para rodar em um cluster de múltiplos servidores.
- **Alta Performance e Disponibilidade:** Otimizados para grandes volumes de leituras e escritas e para tolerar falhas.



# Escalabilidade

Para lidar com Big Data, a escalabilidade é a palavra-chave. Existem duas formas principais de escalar um sistema:

## Escalonamento Vertical (Scale Up)



É como "turbinar" um único servidor: adicionar mais memória RAM, um processador mais potente, um SSD mais rápido. É simples, mas tem limites. Chega um ponto em que fica extremamente caro e, se esse super-servidor falhar, toda a aplicação para (ponto único de falha).

## Escalonamento Horizontal (Scale Out) e Clusterização



É como adicionar mais servidores mais modestos para trabalhar em conjunto. Em vez de um super-servidor, você tem vários "servidores normais" distribuindo o trabalho. Esse conjunto de servidores trabalhando como um só sistema é chamado de **cluster**.

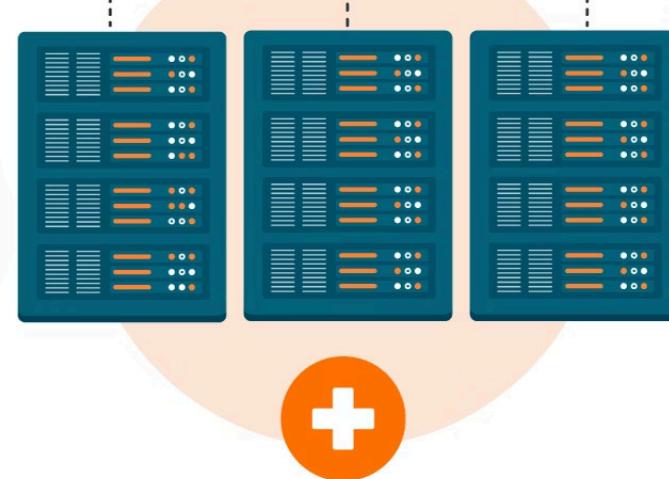
Ou seja...

## Scaling Up



Upgrading equipment to improve performance

## Scaling Out



Adding new equipment and distributing workloads across it



# A clusterização é o processo de agrupar essas máquinas. Isso permite:

- **Escalabilidade quase infinita:** Precisa de mais poder? Adicione mais uma máquina ao cluster.
- **Alta Disponibilidade:** Se um servidor (nó) do cluster falhar, os outros assumem o trabalho e a aplicação continua funcionando.
- **Custo-benefício:** Geralmente é mais barato comprar vários servidores comuns do que um único supercomputador.

Os bancos de dados NoSQL foram projetados desde o início para escalar horizontalmente.



## Escalabilidade

Adicione máquinas ao cluster facilmente

## Alta Disponibilidade

Nós substituem falhas e mantêm serviço

## Custo-benefício

Vários servidores comuns custam menos

# Os Modelos de Bancos de Dados NoSQL

NoSQL não é uma coisa só. É um termo que abrange diferentes modelos de armazenamento. Os quatro principais são:

## Os Quatro Modelos NoSQL

- **Chave-Valor (Key-Value)**

**O que é:** O modelo mais simples. Os dados são armazenados como um dicionário gigante, onde cada item tem uma chave única e um valor associado.

**Exemplos:** [Redis](#), Amazon DynamoDB.

**Onde usam:** Ideal para armazenar dados de sessão de usuários, carrinhos de compra, caching (guardar resultados de consultas pesadas para acesso rápido) e leaderboards de jogos em tempo real.

- **Grafos (Graph)**

**O que é:** Projetado para armazenar dados que são altamente conectados. Pense em redes. Ele usa nós (para representar entidades, como pessoas) e arestas (para representar os relacionamentos, como "amigo de").

**Exemplos:** [Neo4j](#), Amazon Neptune.

**Onde usam:** Redes sociais (relações de amizade), sistemas de recomendação ("quem comprou X também comprou Y"), detecção de fraudes (identificando conexões suspeitas).

- **Documentos (Document-Oriented)**

**O que é:** Armazena dados em estruturas de documentos, que são autossuficientes e encapsulam todos os dados de uma entidade. É como guardar "fichas" completas de informação. O formato mais comum para esses documentos é o [JSON](#).

**Exemplos:** [MongoDB](#), CouchDB, Firebase Firestore.

**Onde usam:** A maioria das aplicações web e mobile modernas. Catálogos de produtos em e-commerce, sistemas de gerenciamento de conteúdo, perfis de usuário.

# Modelo de Documentos: JSON e BSON

## JSON (JavaScript Object Notation)

É um formato de texto leve e legível para humanos, usado para transmitir dados. Ele usa pares de chave-valor.

```
{  
  "titulo": "Comprar leite",  
  "concluida": false  
}
```

## BSON (Binary JSON)

O MongoDB armazena os dados internamente em BSON. É uma representação binária (não legível para humanos) do JSON.

Vantagens do BSON sobre o JSON:

- **Tipos de Dados Adicionais:** Suporta tipos que não existem no JSON, como Date, ObjectId e dados binários.
- **Eficiência:** Por ser binário, ocupa menos espaço e é muito mais rápido para o banco de dados percorrer e consultar.

É por causa deste modelo que na disciplina usaremos o **MongoDB**. Ele é o banco de dados de documentos mais popular do mundo, mas hoje é considerado **multi-modelo**, pois também incorpora funcionalidades de outros modelos, como o chave-valor e busca geoespacial, tornando-o extremamente versátil.

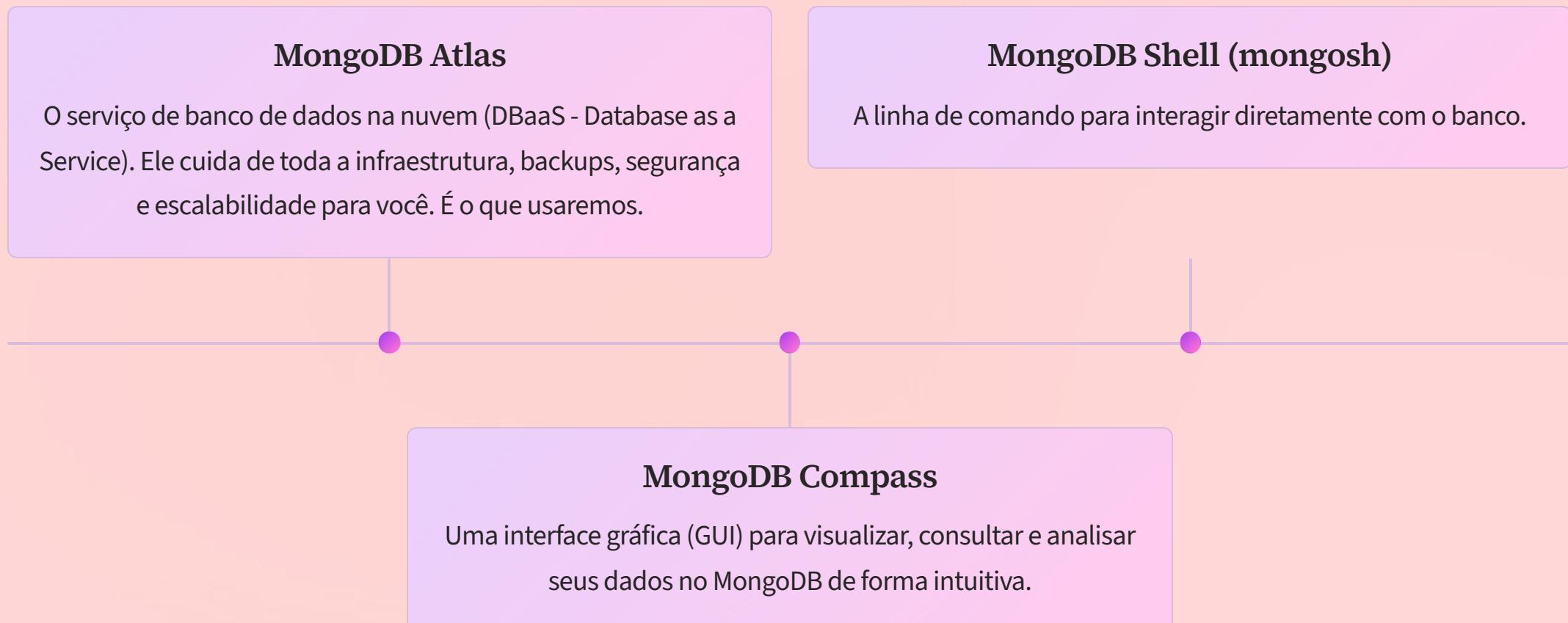
# MongoDB

## Origem e Conceitos

- **Nome:** O nome "MongoDB" vem da palavra em inglês "**humongous**" (enorme, gigantesco), para refletir sua capacidade de lidar com enormes volumes de dados.
- **Escalonamento:** Como já vimos, ele suporta escalonamento **vertical** e, principalmente, **horizontal** através de uma técnica chamada *sharding* (fragmentação), que distribui os dados entre os vários servidores do cluster.

## O Ecossistema MongoDB

O MongoDB é mais do que apenas um banco de dados. É um ecossistema de produtos:



# A Hierarquia do MongoDB

A organização dos dados no MongoDB é simples:



## Exemplo de Documento no MongoDB:

Veja como um documento de uma tarefa seria armazenado. Note o campo `_id`, que é um ObjectId gerado automaticamente pelo MongoDB para garantir que cada documento seja único.

```
{  
  "_id": ObjectId("60d5ec49f7e4e8a1b0b5c8e1"),  
  "titulo": "Aprender sobre NoSQL",  
  "descricao": "Estudar o material completo e tirar dúvidas.",  
  "concluida": false,  
  "dataCriacao": ISODate("2025-09-21T18:30:00.000Z")  
}
```

# Guia Prático: Usando o MongoDB Atlas

Vamos criar e configurar seu primeiro banco de dados na nuvem.

## Crie uma Conta

Acesse o site [MongoDB Atlas](#) e crie uma conta gratuita.

## Crie um Cluster Gratuito (M0)

- Após o login, você será direcionado para criar um novo "deployment".
- Escolha a opção "**Shared**" (**M0**), que é o plano gratuito.
- Escolha um provedor de nuvem (AWS, Google Cloud ou Azure) e uma região. **Dica:** Escolha a região mais próxima de você (ex: sa-east-1 em São Paulo) para menor latência.
- Não precisa alterar as configurações adicionais. Dê um nome ao seu cluster (ex: "ClusterTarefas") e clique em "Create".

continua...

...continuando

## Configure a Segurança (MUITO IMPORTANTE)

- **Usuário do Banco:** Enquanto o cluster está sendo criado, configure o acesso. Crie um **usuário e senha**. Anote-os bem, você precisará deles para conectar a aplicação.
- **Acesso de Rede (Network Access):** Você precisa dizer ao Atlas de quais endereços IP ele pode aceitar conexões. Para desenvolvimento, a forma mais fácil é liberar para todos os IPs. Clique em "Add IP Address" e depois em "Allow Access from Anywhere". Isso adicionará o IP 0.0.0.0/0. **Atenção:** Isso não é seguro para produção, mas é prático para estudar.

## Conecte-se ao Cluster

- Com o cluster pronto (pode levar alguns minutos), vá para a tela "Database" e clique no botão "**Connect**" do seu cluster.
- Selecione a opção "**Drivers**".
- Em "Select your driver", escolha **Node.js** e a versão mais recente.
- O Atlas vai te fornecer uma Connection String. É uma URL que se parece com isso: `mongodb+srv://<username>:<password>@clustertarefas.xxxxx.mongodb.net/?retryWrites=true&w=majority`
- Copie essa string. Ela é a chave para conectar sua aplicação ao banco.

# Agora vamos ver como isso funciona na prática:

## API de Tarefas com MongoDB

Agora, vamos conectar a teoria com a prática, analisando os arquivos do seu projeto. O objetivo é entender como essa aplicação Node.js usa o MongoDB Atlas para criar, ler, atualizar e deletar tarefas (operações CRUD).

### .env

Este arquivo, como de costume, guarda senhas e configurações que não devem ir para o código-fonte público.

```
MONGO_URI=mongodb+srv://clusterdesummerSmith@cluster0.iimmhfl.mongodb.net/?  
retryWrites=true&w=majority&appName=Cluster0
```

- **MONGO\_URI**: É a sua Connection String do Atlas. Você deve substituir <password> pela senha que você criou no passo anterior. A aplicação vai ler essa variável para saber onde o banco de dados está.

### /db.js

Este é o coração da conexão com o banco de dados.

```
const mongoose = require('mongoose');  
require('dotenv').config(); // Carrega as variáveis do arquivo .env  
  
const connectDB = async () => {  
  try {  
    const conn = await mongoose.connect(process.env.MONGO_URI);  
    console.log(`MongoDB Conectado com Sucesso: ${conn.connection.host}`);  
  } catch (error) {  
    console.error(`Erro ao conectar com o MongoDB: ${error.message}`);  
    process.exit(1); // Encerra a aplicação se não conseguir conectar  
  }  
};  
  
module.exports = connectDB;
```

- **mongoose**: É uma biblioteca (ODM - Object Document Mapper) que facilita MUITO a vida de quem trabalha com Node.js e MongoDB. Ela permite criar "modelos" para seus dados, validar informações e interagir com o banco de forma mais estruturada e segura.
- **mongoose.connect(...)**: É a função que efetivamente usa a MONGO\_URI do arquivo .env para estabelecer a conexão com seu cluster no Atlas.

## /Tarefa.js

Este arquivo define a "forma" que um documento de tarefa deve ter.

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

const TarefaSchema = new mongoose.Schema({
  titulo: {
    type: String,
    required: [true, 'O título da tarefa é obrigatório.'],
    trim: true
  },
  descricao: {
    type: String,
    required: false
  },
  concluida: {
    type: Boolean,
    default: false
  },
  dataCriacao: {
    type: Date,
    default: Date.now
  }
});
```

- **mongoose.Schema:** Aqui definimos a estrutura (o schema) de um documento na coleção tarefas.
- Para cada campo (titulo, descricao, etc.), definimos o tipo de dado (String, Boolean, Date), se ele é obrigatório (required: true) e valores padrão (default: false). Isso garante uma consistência mínima nos dados que salvamos.

## /tarefaController.js

Aqui está a lógica de negócio, as funções que executam as operações CRUD.

```
const Tarefa = require('../models/Tarefa'); // Importa o modelo

// Criar uma tarefa
exports.criarTarefa = async (req, res) => {
  // Usa o modelo para criar um novo documento no banco com os dados do corpo da requisição
  const tarefa = await Tarefa.create(req.body);
  res.status(201).json({ data: tarefa });
};

// Listar todas as tarefas
exports.listarTarefas = async (req, res) => {
  // .find() sem argumentos busca TODOS os documentos da coleção
  const tarefas = await Tarefa.find();
  res.status(200).json({ data: tarefas });
};

// Atualizar uma tarefa
exports.atualizarTarefa = async (req, res) => {
  // Procura um documento pelo ID (passado na URL) e o atualiza
  const tarefa = await Tarefa.findByIdAndUpdate(req.params.id, req.body, { new: true });
  res.status(200).json({ data: tarefa });
};

// Deletar uma tarefa
exports.deletarTarefa = async (req, res) => {
  // Procura um documento pelo ID e o remove
  await Tarefa.findByIdAndDelete(req.params.id);
  res.status(200).json({ data: {} });
};
```

O Mongoose nos dá métodos fáceis de usar como `create()`, `find()`, `findByIdAndUpdate()` e `findByIdAndDelete()`, que são traduzidos para as operações nativas do MongoDB.

## /servidor.js

Este é o ponto de entrada da sua API. Ele junta tudo.

```
const express = require('express');
const connectDB = require('./config/db'); // Importa a função de conexão
const tarefaRoutes = require('./routes/tarefas');

const app = express();

connectDB(); // Executa a conexão com o MongoDB assim que o servidor inicia

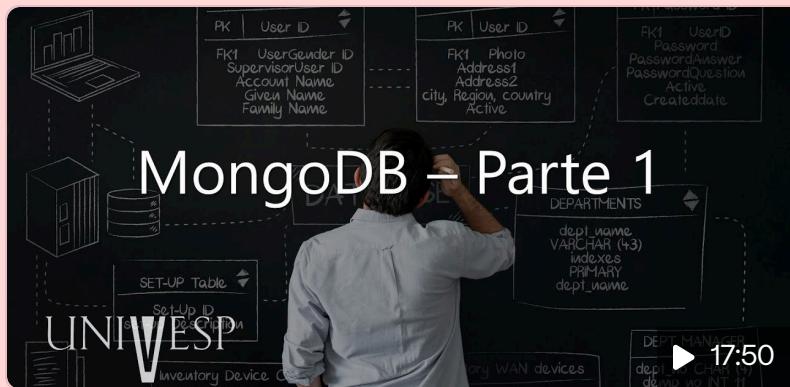
app.use(express.json()); // Permite que o servidor entenda requisições com corpo em JSON
app.use('/api/tarefas', tarefaRoutes); // Direciona todas as requisições para /api/tarefas para o arquivo de rotas

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Servidor rodando na porta ${PORT}`));
```

A linha mais importante aqui, para o nosso contexto, é `connectDB()`. Ela garante que, antes de começar a receber requisições, a API já esteja conectada ao banco de dados no Atlas.

# Conteúdos extras:

## Vídeos:



YouTube

### Banco de Dados – MongoDB – Parte 1

univesp.br Eixo de Computação – COM300 Univesp – Universidade Virtual do Estado de São Paulo Professor: José Eduardo Santarem Segundo Nesta...

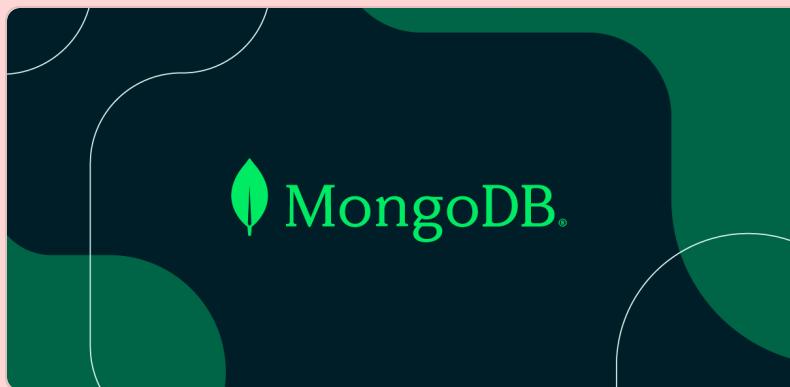


YouTube

### Banco de Dados – MongoDB – Parte 2

univesp.br Eixo de Computação – COM300 Univesp – Universidade Virtual do Estado de São Paulo Professor: José Eduardo Santarem Segundo Nesta...

## Documentação oficial:



MongoDB

### MongoDB Atlas: Cloud Document Database

Cloud-hosted MongoDB service on AWS, Azure, and GCP

## Clone o repositório para começar a testar e exercitar:

alinetimm/Aula-de-Banco-de-dados-n-o-...



1 Contributor 0 Issues 0 Stars 0 Forks

GitHub

### GitHub – alinetimm/Aula-de-Banco-de-dados-n-o-relacional

Contribute to alinetimm/Aula-de-Banco-de-dados-n-o-relacional development by creating an account on GitHub.