Trabalho de banco de dados NoSQL- MongoDB

Grupo:

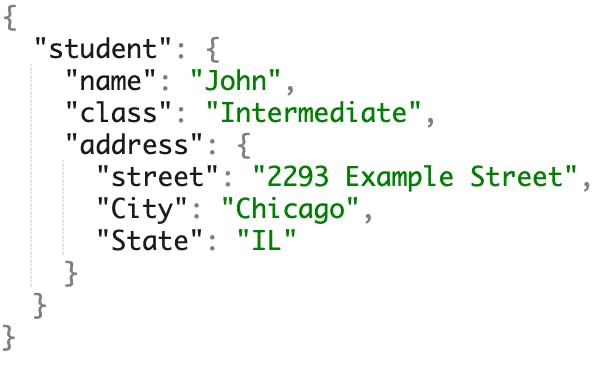
Kamylly Simeão

Victoria

Luiz Filipe

# Tipo de banco de dados que será utilizado:

Para o desenvolvimento do projeto iremos utilizar o banco MongoDB, já que ele é de acesso livre, possui o tipo de estrutura adequada para o projeto que iremos desenvolver.



### Exemplo de estrutura de dados utilizado pelo MongoDB

## Tema abordado:

Um site de receita onde o usuário poderá visualizar os ingredientes, o modo de preparo e dicas de como fazer os pratos, haverá também disponibilidade para ver imagens do prato pronto.

## Dados que serão processados pelo banco.

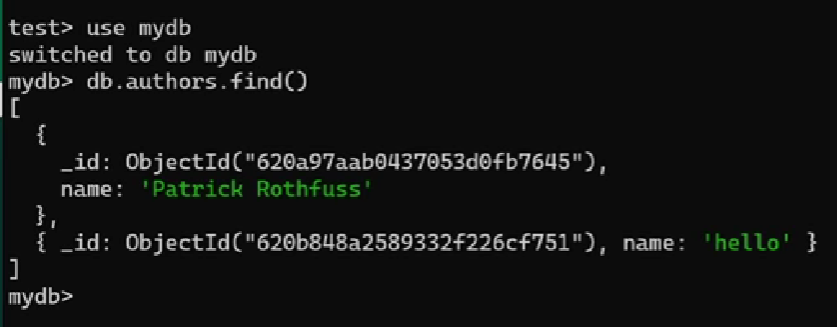
O site utilizará de uma estrutura de documentos, onde armazenará as seguintes estruturas:

1. Nome do prato;
2. Ingredientes;
3. Modo de preparo;
4. Tempo de preparo;
5. Valor médio do prato;
6. Dicas;
7. Imagens.

De modo que cada prato terá os nomes, ingredientes e etc. Estruturando dessa maneira conseguiremos diversificar a quantidade de pratos que serão exibidos nos sistemas.

## Documentação e modelagem

As documentações do MongoDB estão disponíveis no site: <https://www.mongodb.com/pt-br/docs/>, onde podemos ver como instalar, estruturar e manusear os dados usando o mongoDB.



### Exemplo de comando no prompt

É importante que tenhamos em mente as diferenças do MongoDB para o MySQL, por exemplo, é que enquanto os dados estruturados (tabelas) do MySQL precisam ser feitos necessariamente de maneiras iguais, nos documentos do mongo pode haver a liberdade de os pares chave/valor serem estruturas de forma diferente de um documento para outro, e mesmo assim oferecer uma segurança.

A documentação nos mostra de inicio como poderemos proceder com o banco nos mostrando o ‘manual do banco de dados’:



Lá nos vamos encontrar as instruções iniciais de como realizar uma aplicação com o Mongo, explicitando quais são as opções nas quais poderíamos realizar as operações, e como funciona a alocação de dados no MongoDB.

Por exemplo uma das maneiras de implementar o MongoDB para nossas soluções pode ser de maneira remota com o Atlas, serviço de implementação de solução na nuvem, ou com as implementações locais com o mongoDB enterprise e comunnity, as versões paga e gratuita do mongo, respectivamente.

Qualquer uma das possibilidades apresentadas pelo site oficial nos fornecerá uma vasta possibilidade de aplicações, por exemplo, o Atlas no possibilita trabalhar com as nuvens da AWS, Azure ou GoogleCloud, deixando nossas aplicações mais performáticas em comparação as aplicações locais (Enterprise/Community).

## 3.Comandos do MongoDB

Exemplo básico de como você poderia configurar um servidor Node.js com o MongoDB usando o framework Express:

// Importando os módulos necessários

const express = require('express');

const mongoose = require('mongoose');

// Configurações iniciais

const app = express();

const PORT = 3000;

// Conexão com o MongoDB

mongoose.connect('mongodb://localhost:27017/meuBancoDeDados', { useNewUrlParser: true, useUnifiedTopology: true })

.then(() => console.log('Conexão com o MongoDB estabelecida com sucesso!'))

.catch(err => console.error('Erro ao conectar com o MongoDB:', err));

// Definindo o esquema para as receitas

const receitaSchema = new mongoose.Schema({

titulo: String,

ingredientes: [String],

instrucoes: String,

autor: String

});

const Receita = mongoose.model('Receita', receitaSchema);

// Exemplo de rota para adicionar uma nova receita

app.post('/receitas', async (req, res) => {

try {

const novaReceita = await Receita.create(req.body);

res.status(201).json(novaReceita);

} catch (err) {

res.status(400).json({ message: err.message });

}

});

// Exemplo de rota para obter todas as receitas

app.get('/receitas', async (req, res) => {

try {

const receitas = await Receita.find();

res.json(receitas);

} catch (err) {

res.status(500).json({ message: err.message });

}

});

// Iniciando o servidor

app.listen(PORT, () => {

console.log(`Servidor rodando na porta ${PORT}`);

});

Para armazenar os dados das postagens no MongoDB de forma que fiquem disponíveis no site, você precisará definir rotas em seu servidor Node.js para lidar com operações de criação, leitura, atualização e exclusão (CRUD) das postagens. Aqui está um exemplo de como você pode fazer isso:

1. **Definindo o esquema de dados**: Você precisa definir um esquema de dados para suas postagens no MongoDB. Por exemplo:

const mongoose = require('mongoose');

const postagemSchema = new mongoose.Schema({

titulo: { type: String, required: true },

conteudo: { type: String, required: true },

autor: { type: String, required: true },

dataPublicacao: { type: Date, default: Date.now }

});

const Postagem = mongoose.model('Postagem', postagemSchema);

module.exports = Postagem;

1. **Rotas para manipulação das postagens**: Defina rotas no seu servidor para manipular as operações CRUD das postagens. Por exemplo:

const express = require('express');

const router = express.Router();

const Postagem = require('../models/postagem');

// Rota para criar uma nova postagem

router.post('/postagens', async (req, res) => {

try {

const novaPostagem = await Postagem.create(req.body);

res.status(201).json(novaPostagem);

} catch (err) {

res.status(400).json({ message: err.message });

}

});

// Rota para obter todas as postagens

router.get('/postagens', async (req, res) => {

try {

const postagens = await Postagem.find().sort({ dataPublicacao: -1 });

res.json(postagens);

} catch (err) {

res.status(500).json({ message: err.message });

}

});

// Rota para obter uma postagem específica por ID

router.get('/postagens/:id', async (req, res) => {

try {

const postagem = await Postagem.findById(req.params.id);

if (postagem == null) {

return res.status(404).json({ message: 'Postagem não encontrada' });

}

res.json(postagem);

} catch (err) {

res.status(500).json({ message: err.message });

}

});

// Rota para atualizar uma postagem

router.put('/postagens/:id', async (req, res) => {

try {

const postagem = await Postagem.findByIdAndUpdate(req.params.id, req.body, { new: true });

res.json(postagem);

} catch (err) {

res.status(400).json({ message: err.message });

}

});

// Rota para excluir uma postagem

router.delete('/postagens/:id', async (req, res) => {

try {

await Postagem.findByIdAndDelete(req.params.id);

res.json({ message: 'Postagem excluída com sucesso' });

} catch (err) {

res.status(500).json({ message: err.message });

}

});

module.exports = router;

1. **Utilizando as rotas**: Use essas rotas no seu servidor principal (**app.js** ou equivalente) para que o Express saiba como lidar com as solicitações HTTP para criar, obter, atualizar e excluir postagens.

const express = require('express');

const mongoose = require('mongoose');

const postagensRouter = require('./routes/postagens');

const app = express();

const PORT = 3000;

mongoose.connect('mongodb://localhost:27017/meuBancoDeDados', { useNewUrlParser: true, useUnifiedTopology: true })

.then(() => console.log('Conexão com o MongoDB estabelecida com sucesso!'))

.catch(err => console.error('Erro ao conectar com o MongoDB:', err));

app.use(express.json());

app.use('/api', postagensRouter);

app.listen(PORT, () => {

console.log(`Servidor rodando na porta ${PORT}`);

});