# Integrantes do Grupo

* Beatriz Lucas - RM99104
* Enzo Farias - RM 98792
* Ewerton Gonçalves - RM98571
* Guilherme Tantulli - RM97890
* Thiago Zupelli - RM99085

Este projeto aborda a migração de um banco de dados relacional (Oracle) para um banco de dados NoSQL utilizando MongoDB. O banco de dados contém informações sobre auditoria, campanhas, categorias, compras, créditos, usuários e produtos. O MongoDB foi selecionado como a tecnologia NoSQL apropriada por sua capacidade de lidar com grandes volumes de dados, escalabilidade e flexibilidade em relação à estrutura de esquema de dados.

**Justificativa para o Uso do MongoDB**:

* **Escalabilidade**: MongoDB é altamente escalável horizontalmente através de sharding, permitindo o particionamento eficiente dos dados entre vários servidores. Esta abordagem melhora o desempenho e a capacidade de distribuição dos dados.
* **Flexibilidade de esquema**: O MongoDB oferece um esquema flexível que permite a inserção de novos campos nos documentos sem a necessidade de modificar ou migrar o banco de dados, facilitando a evolução contínua dos dados.
* **Compatibilidade com Big Data**: MongoDB pode lidar com grandes volumes de dados e permite consultas eficientes sobre grandes conjuntos de dados.
* **Facilidade de consulta**: A linguagem de consulta baseada em JSON do MongoDB facilita a execução de operações CRUD (Create, Read, Update, Delete) e consultas complexas de forma simples e intuitiva.

A estrutura de dados que estamos utilizando no MongoDB segue o paradigma NoSQL, onde os dados são armazenados em **coleções de documentos**, cada documento sendo representado em **formato JSON/BSON**. Aqui está uma explicação sucinta de como organizamos esses dados:

### **Coleções e Documentos:**

* Cada **coleção** (como USERS, COMPRAS, CREDIT, etc.) agrupa documentos que possuem dados semelhantes. Diferentemente de bancos de dados relacionais, não há tabelas com colunas fixas; em vez disso, os documentos podem variar em formato e campos.
* **Documentos** dentro de uma coleção são como registros individuais. Cada documento é um objeto JSON que contém um conjunto de **pares chave-valor**, e esses campos podem ser diferentes entre documentos.

### **Estrutura de Documentos:**

* Cada documento possui um campo **\_id**, que é o identificador único do documento dentro da coleção. Esse campo pode ser gerado automaticamente ou definido manualmente, como fizemos nos exemplos.
* Os demais campos são definidos com base nas necessidades da aplicação. Por exemplo:
  + Em USERS, armazenamos informações básicas dos usuários, como nome, email, CPF e telefone.
  + Em COMPRAS, armazenamos dados de compras, como o **id\_usuario** (que referencia um documento na coleção USERS), a quantidade de produtos e o valor total da compra.
  + Em coleções como CREDIT\_COMPRAS, utilizamos campos como **id\_credit** e **id\_compra** para referenciar documentos de outras coleções, relacionando dados de crédito e compras.

### **Flexibilidade e Eficiência:**

* Não há a rigidez de esquemas como em bancos relacionais. Por exemplo, novos campos podem ser adicionados aos documentos sem necessidade de alterar a estrutura das coleções.
* Em vez de relações complexas entre tabelas (joins), usamos referências simples (como id\_usuario e id\_compra) ou agrupamos informações diretamente dentro de um documento, como em alguns casos de dados mais estruturados, evitando a necessidade de joins caros.

### **Exemplo de Referência e Integração:**

* **Referências**: Em coleções como COMPRAS e CREDIT\_COMPRAS, utilizamos referências a outras coleções (por exemplo, associando um usuário a uma compra através do campo id\_usuario). Essas referências nos permitem acessar dados relacionados facilmente, enquanto mantemos os documentos separados e mais eficientes para leitura e escrita.

### **Escalabilidade e Manutenção:**

* Esta estrutura é fácil de escalar horizontalmente, pois os dados são distribuídos por coleções independentes, o que facilita a adição de novos servidores (sharding) sem comprometer a performance.
* A manutenção é simplificada, pois podemos modificar ou expandir os dados de uma coleção sem afetar as demais, mantendo flexibilidade no crescimento da aplicação.

## Funcionalidades

### 1. CRUD de Usuários

* Listar todos os usuários.
* Adicionar um novo usuário.
* Editar um usuário existente.
* Excluir um usuário.

### 2. CRUD de Compras

* Listar todas as compras realizadas.
* Adicionar uma nova compra, escolhendo o usuário e o produto.
* Editar os detalhes de uma compra.
* Excluir uma compra.

### 3. CRUD de Categorias

* Listar todas as categorias cadastradas.
* Adicionar uma nova categoria.
* Editar uma categoria existente.
* Excluir uma categoria.

### 4. CRUD de Produtos

* Listar todos os produtos cadastrados.
* Adicionar um novo produto.
* Editar um produto existente.
* Excluir um produto.

### 5. Exportação de Dados

* Exporta os dados de usuários, campanhas e produtos em um arquivo CSV.

## Estrutura do Projeto

* app.py: Arquivo principal que contém as rotas e a lógica principal da aplicação.
* functions: Contém as funções modulares que gerenciam diferentes aspectos da aplicação, como login, registro, dashboard, manipulação de dados de compras, categorias e usuários.
* templates: Contém os arquivos HTML, organizados para renderizar os dados dinamicamente no frontend.
* static: Contém os arquivos estáticos como CSS e JavaScript (se necessário).

## Dependências e Extensões Utilizadas

### 1. Flask

* Framework web utilizado para criar a aplicação e manipular as rotas.

### 2. Flask-PyMongo

* Extensão utilizada para integrar o Flask com o banco de dados MongoDB.

### 3. bcrypt

* Biblioteca utilizada para realizar o hash das senhas de usuários no registro.

### 4. dotenv

* Utilizado para carregar variáveis de ambiente, como a URI do MongoDB e a chave secreta do Flask.

### 5. csv e json

* Bibliotecas padrão do Python utilizadas para manipulação de arquivos CSV e JSON.

### 6. io

* Utilizado para criar um arquivo em memória para a exportação dos dados.

### 7. bson

* Extensão necessária para manipular objetos como ObjectId do MongoDB.

## Pré-requisitos

1. Python 3.9 ou superior.
2. MongoDB: Certifique-se de ter um banco de dados MongoDB configurado, seja local ou hospedado na nuvem (como o MongoDB Atlas).
3. pip: Instalador de pacotes do Python para gerenciar as dependências.

## Como Instalar e Rodar o Projeto

1. Clone o repositório:

git clone https://github.com/usuario/projeto-flask-mongo.git

cd projeto-flask-mongo

1. Crie um ambiente virtual:

* python3 -m venv venv
* source venv/bin/activate # No Windows use: venv\Scripts\activate

1. Instale as dependências:

* pip install -r requirements.txt

1. Crie um arquivo .env na raiz do projeto com as seguintes variáveis de ambiente:

* SECRET\_KEY='sua-chave-secreta-aqui'
* MONGO\_URI='mongodb://seu-usuario:senha@localhost:27017/seu-banco'

1. Inicie a Aplicação:

* flask run

1. A aplicação estará disponível no endereço [http://127.0.0.1:5000](http://127.0.0.1:5000/)

## Estrutura das Rotas

### Usuários

* /usuarios: Lista os usuários cadastrados.
* /usuarios/create: Formulário para criar um novo usuário.
* /usuarios/update/<id>: Formulário para atualizar um usuário existente.
* /usuarios/delete/<id>: Exclui um usuário.

### Compras

* /compras: Lista todas as compras.
* /compras/cadastrar: Formulário para cadastrar uma nova compra.
* /compras/editar/<id>: Formulário para editar uma compra existente.
* /compras/deletar/<id>: Exclui uma compra.

### Categorias

* /categorias: Lista todas as categorias.
* /categorias/create: Formulário para criar uma nova categoria.
* /categorias/update/<id>: Formulário para atualizar uma categoria existente.
* /categorias/delete/<id>: Exclui uma categoria.

### Produtos

* /produtos: Lista todos os produtos cadastrados.
* /produtos/create: Formulário para criar um novo produto.
* /produtos/update/<id>: Formulário para atualizar um produto existente.
* /produtos/delete/<id>: Exclui um produto.

### Exportação

* /export\_db: Exporta os dados de usuários, campanhas e produtos em um arquivo CSV.

## Considerações Finais

Este projeto foi desenvolvido para ser uma solução completa de gerenciamento de banco de dados MongoDB em um ambiente web utilizando Flask. Todas as operações de CRUD foram implementadas com interface amigável e validação de dados. A funcionalidade de exportação de dados permite facilmente extrair informações do banco em formato CSV, facilitando o uso e análise externa.