Food Delivery API

Desenvolvimento Web I

Grupo inf25dw1g13

Felipe Castilho	a047152
Juliana Moreira	a047188
Marta Vieira	a046756

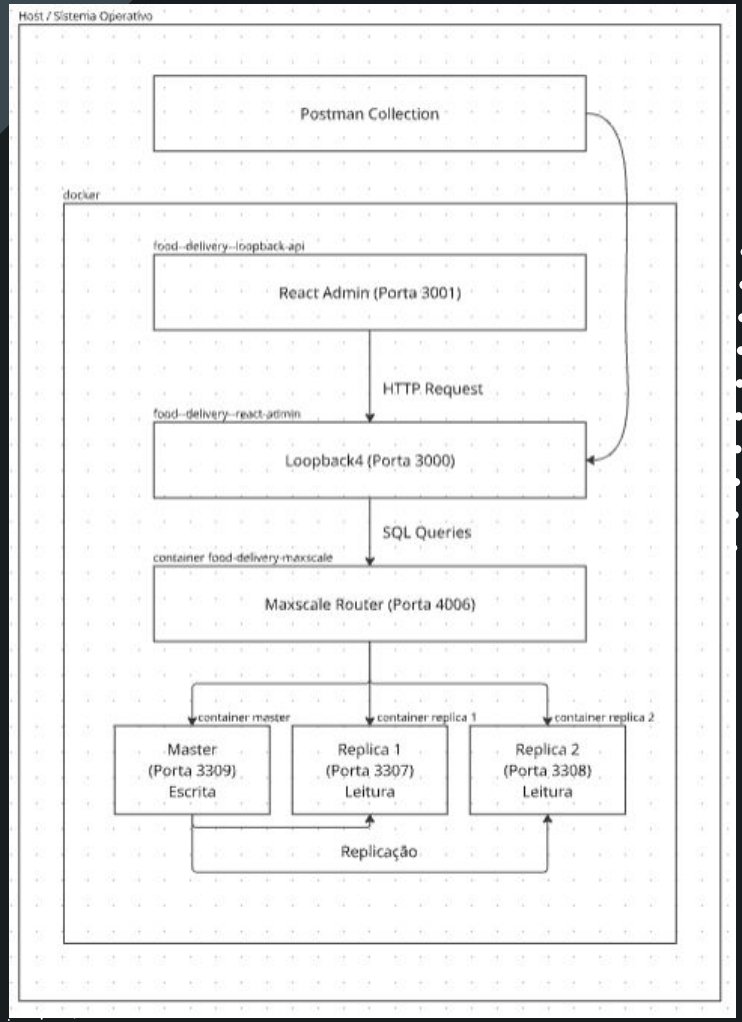
Introdução

Este trabalho tem como objetivo desenvolver uma *API* REST que, através dos métodos *POST*, *GET*, *PUT*, *PATCH* e *DELETE*, interage com uma base de dados, simulando uma plataforma de entregas de comida.

O projeto consiste numa *API* desenvolvida com LoopBack 4 seguindo uma abordagem *Code-first*, que permite criar, ler, atualizar e eliminar dados relacionados ao sistema de entregas. Para suporte dos dados, é utilizada uma base de dados em *MariaDB*, responsável pela estrutura e armazenamento das informações relacionadas ao projeto.

Arquitetura

- ***DB-master***:
  Servidor principal responsável pelas operações de escrita.
- ***DB-réplicas***:
  Servidores secundários configurados como réplicas, para operações de leitura.
- ***MaxScale***:
  Proxy e roteador que implementa o *read/write splitting*, e encaminha automaticamente os pedidos de escrita e leitura para as respetivas bases de dados.
- ***API***:
  Serviço *Node.js* desenvolvido com *LoopBack 4* que implementa os *endpoints* da *API* e comunica com a base de dados exclusivamente através do *MaxScale*.
- ***Backoffice***:
  Interface de administração desenvolvida com *React Admin* que permite a gestão de todos os recursos da aplicação.



Tabelas principais

- Restaurantes
- Pratos
- Clientes
- Pedidos
- Ingredientes
- Entregas
- Categorias de Pratos
- Entregadores
- Códigos Postais

Relação entre Recursos

Relações 1:n (Um para Muitos)

- Restaurantes → Pratos: Um restaurante tem muitos pratos
- Restaurantes → Pedidos: Um restaurante tem muitos pedidos
- Clientes → Pedidos: Um cliente faz muitos pedidos
- Categorias → Pratos: Uma categoria tem muitos pratos
- Códigos Postais → Clientes /Restaurantes/ Entregadores: Um código postal pode ter múltiplos registos

Relações m:n (Muitos para Muitos)

- Pratos ↔ Ingredientes: Um prato tem muitos ingredientes e um ingrediente pode estar em muitos pratos (através da tabela pratos_ingredientes)
- Pedidos ↔ Pratos: Um pedido contém muitos pratos e um prato pode estar em muitos pedidos (através da tabela pedidos_pratos)

LoopBack

1. Controllers

Lida com as **requisições HTTP** (rotas).

- Recebe as requisições do cliente (GET, POST, PUT, DELETE).
- Valida dados de entrada (quando configurado).
- Chama métodos do **Repository**.
- Retorna a resposta HTTP.

O controller **não** acessa o banco de dados diretamente.

2. Models

Define a **estrutura dos dados** e suas regras.

- Propriedades (campos).
- Tipos.
- Validações e relações.
  Representa uma entidade do domínio.

LoopBack

3. Datasources

Configura a **conexão com a fonte de dados**.

- Banco de dados (MariaDB).
- API externa (React Admin).
- Arquivo em memória.

4. Repositórios

Faz a **ponte entre Model e Datasource**.

- Executa operações no banco (find, create, update, delete).
- Usa o Model para saber a estrutura dos dados.
- Usa a Datasource para saber **onde** salvar/buscar.

Estrutura do Projeto

Após Docker-compose (*dev*: docker compose -f docker-compose.dev.yaml up --build) ou (prod: docker login e docker compose -f docker-compose.prod.yaml up)

1. Rede e Volumes
Rede: *food_network*
Volumes: *master_data*, *replica1_data*, *replica2_data*

2. Master
Iniciada com os scripts de criação da base de dados e inserção dos dados iniciais.
Executa scripts na ordem:

- Cria utilizador para *MaxScale* (*00_maxscale_user.sql*)
- Cria base de dados e tabelas (*01_schema.sql*)
- Insere dados iniciais (*02_seed.sql*)
- Cria *triggers* de *audit* (*03_audit.sql*)

Após a inicialização, a base de dados fica acessível na porta 3309.

Estrutura do Projeto

3. Réplicas
Iniciadas nas portas 3307 e 3308, na *BD-replica* 1 e na *BD-replica2*, respetivamente.
Cada réplica espera *Master* estar ligado para se conectar e iniciar o processo de replicação até ficarem totalmente sincronizadas.

4. Inicialização do MaxScale
Lê as configurações definidas no ficheiro *maxscale.cnf*, que cria o *monitor* e configura o maxscale.
Estabelece ligação às três bases de dados: *Master*, *Replica1* e *Replica2*.

5. MariaDB Monitor
Verifica o estado do *Master* e das Réplicas a cada 2 segundos com *pings*. - *health check*
Promove réplica a *Master* se *Master* cair - *auto-failover*
Reconecta servidores que voltam *online* - *auto-rejoin*
Router: *ReadWriteSplit* - *read-only enforcement*

- *SELECTs* são encaminhados para as réplicas.
- *INSERTs* / *UPDATEs* / *DELETEs* são enviados para o *Master*.

Imagens

Imagem API (Loopback 4 Server)

src/food-delivery-lb4/Dockerfile

```
FROM docker.io/library/node:24-slim  # Faz download da imagem base Node.js
USER node                             # Define utilizador não-root
RUN mkdir -p /home/node/app           # Cria diretório de trabalho
WORKDIR /home/node/app                # Define o diretório de trabalho
COPY --chown=node package*.json ./    # Copia o package.json
RUN npm install                       # Instala as dependências
COPY --chown=node . .                 # Copia o resto do código
RUN npm run build                     # Compila o TypeScript
ENV HOST=0.0.0.0 PORT=3000           # Define variáveis de ambiente
EXPOSE 3000                           # Indica a porta usada pelo LoopBack
CMD ["node", "."]                    # Inicia o servidor LoopBack
```

Imagens

Imagem React Admin

src/food_react_app/Dockerfile.prod (Produção)

```
# Multi-stage
# Stage 1: Build
FROM node:20-alpine AS builder
WORKDIR /app
COPY package*.json ./
RUN npm install --legacy-peer-deps
COPY . .
RUN npm run build

# Stage 2: Production
FROM nginx:alpine
COPY --from=builder /app/build /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Imagens

Imagem React Admin

src/food_react_app/Dockerfile (Desenvolvimento)

```
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3001
CMD ["npm", "start"]
```

Imagens

Imagem *mariadb-master*: **src/mariadb-master/Dockerfile**

```
FROM mariadb:11.2                          # Faz download da imagem base MariaDB
COPY my.cnf /etc/mysql/conf.d/             # Aplica configurações personalizadas do servidor
COPY scripts/ /docker-entrypoint-initdb.d/ # Copia os scripts de inicialização
```

Scripts Executados (em ordem de execução):

- 00_maxscale_user.sql - Cria o utilizador utilizado pelo *MaxScale*
- 01_schema.sql - Cria a base de dados e as tabelas
- 02_seed.sql - Insere dados iniciais na base de dados
- 03_audit.sql - Configura *audit* e *triggers*

Imagens

Imagem *mariadb-replicas*: **src/mariadb-replica/Dockerfile**

```
FROM mariadb:11.2                                   # Faz download da imagem base MariaDB
COPY my.cnf /etc/mysql/conf.d/                      # Aplica configurações personalizadas do servidor
COPY scripts/ /docker-entrypoint-initdb.d/          # Copia os scripts de inicialização

RUN chmod +x /docker-entrypoint-initdb.d/*.sh 2>/dev/null || true
# Permite que os bash possam ser executados quando o MariaDB estiver pronto
```

Scripts Executados:

- 01_setup_replication.sh - Configura a réplica e conecta ao *Master* para iniciar a replicação

Imagens

Imagem *maxscale*: **src/maxscale/Dockerfile**

```
FROM mariadb/maxscale:23.08                    # Descarrega a imagem do maxscale
COPY maxscale.cnf.ini /etc/maxscale.cnf        # Copia a configuração do servidor, monitor e routing
```

- Assim, o *MaxScale* liga-se ao *mariadb-master*, *mariadb-replica1*, *mariadb-replica2*. Depois Inicia o *MariaDB monitor*, ativa o *Read/Write Split Router* e cria o *listener* da *API* e o painel de administração.

Serviços ativos

- *localhost*:3000 – *LoopBack 4 API (c/endpoint para testar ping)*
- *localhost*:3001 – *React Admin Interface*
- *localhost*:3309 – *MariaDB Master*
- *localhost*:3307 – *MariaDB Replica* 1
- *localhost*:3308 – *MariaDB Replica* 2
- *localhost*:4006 – *MaxScale* (*router*)
- *localhost*:8989 – *MaxScale Dashboard*

Maxscale Credentials para *Dashboard*: *admin* / *mariadb*

Postman

A *Postman Collection* está disponível em: src/food-api.postman_collection.json

A collection inclui exemplos de todos os recursos da API:

- Restaurantes, Pratos, Clientes, Pedidos, Entregas, Ingredientes, Entregadores, Categorias de Pratos, Códigos Postais

Todos os endpoints seguem o padrão REST e podem ser testados através da collection do Postman ou através da interface Swagger UI disponível em http://localhost:3000/explorer.

Filtros

Filtros através de parâmetros *HTTP*:

A *API* suporta filtros através de *query parameters* usando os recursos do LoopBack 4.

Exemplo de um controller: **RestaurantesController.ts**

Controllers contém filtros como...

```
@get('/restaurantes')
@response(200, {
  description: 'Array of Restaurantes model instances',
  content: {
    'application/json': {
      schema: {
        type: 'array',
        items: getModelSchemaRef(Restaurantes, {includeRelations: true}),
      },
    },
  },
})
async find(
  @param.filter(Restaurantes) filter?: Filter<Restaurantes>,
): Promise<Restaurantes[]> {
  return this.restaurantesRepository.find(filter);
}
```

Filtros

Filtros disponíveis (usando sintaxe LoopBack 4):

- `GET /restaurantes?filter[where][estado]=aberto` - Filtrar restaurantes por estado
- `GET /pratos?filter[where][disponivel]=true` - Filtrar pratos disponíveis
- `GET /pratos?filter[where][vegetariano]=true` - Filtrar pratos vegetarianos
- `GET /pedidos?filter[where][cliente_id]=1` - Filtrar pedidos por cliente
- `GET /ingredientes?filter[where][alergeno]=true` - Filtrar ingredientes alergénios
- `GET /entregas?filter[where][estado]=pendente` - Filtrar entregas por estado
- `GET /entregadores?filter[where][estado]=disponivel` - Filtrar entregadores disponíveis

Exemplos de filtros avançados:

- `GET /pratos?filter[where][and][0][restaurante_id]=1&filter[where][and][1][vegetariano]=true` - Pratos vegetarianos de um restaurante
- `GET /pedidos?filter[order]=hora_pedido DESC&filter[limit]=10` - Últimos 10 pedidos ordenados por data de pedido

Para mais detalhes

Relatório completo em: https://github.com/...