Relatório Individual – Food Delivery Project

Identificação dos Membros do Grupo

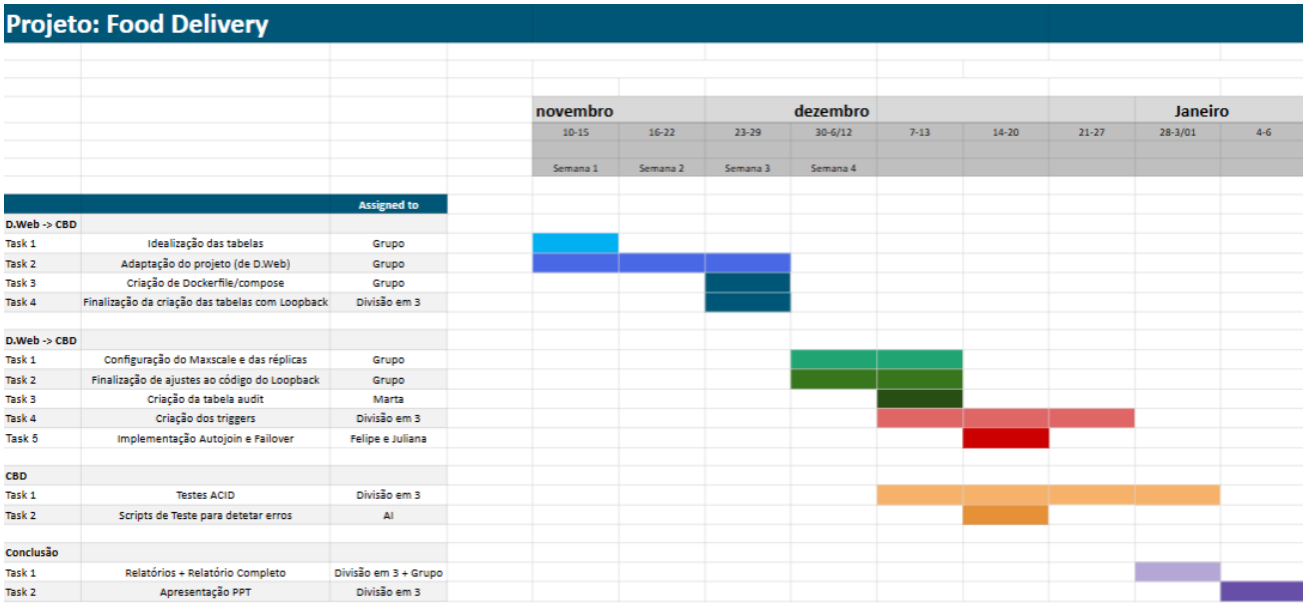
Felipe Castilho a047152@umaia.pt

Juliana Moreira a047188@umaia.pt

Marta Vieira a046756@umaia.pt

Cronograma de Execução (Diagrama de Gantt)

O projeto foi desenvolvido ao longo do semester.O cronograma abaixo representa as principais fases do desenvolvimento:



Descrição Técnica do Projeto

O projeto consiste no desenvolvimento de uma API REST para uma plataforma de entregas de comida, utilizando **LoopBack 4** para o backend e **React Admin** para o frontend. A base de dados utiliza **MariaDB 11.2** com arquitetura de replicação Master-Replica e **MaxScale** para read/write splitting.

O sistema implementa uma arquitetura de microserviços containerizados com Docker, incluindo:

- **API LoopBack 4** (Node.js 24) - Porta 3000
- **React Admin Backoffice** - Porta 3001
- **MariaDB Master** - Porta 3309
- **MariaDB Replica 1** - Porta 3307
- **MariaDB Replica 2** - Porta 3308
- **MaxScale** - Porta 4006 (router) e 8989 (dashboard)

Contribuições Individuais

1. Sistema de Audit - Triggers de Base de Dados

Tarefa: Implementação do sistema de audit para rastreamento de alterações na base de dados.

Tabelas Implementadas: - codpostal - 3 triggers (INSERT, UPDATE, DELETE) - categorias_pratos - 3 triggers (INSERT, UPDATE, DELETE) - ingredientes - 3 triggers (INSERT, UPDATE, DELETE)

Total: 9 triggers implementados

Estrutura da Tabela `audit_log`:

```
CREATE TABLE IF NOT EXISTS audit_log (
    id INT AUTO_INCREMENT PRIMARY KEY,
    tabela_nome VARCHAR(50) NOT NULL,
    operacao ENUM('INSERT','UPDATE','DELETE') NOT NULL,
    registro_id INT,
    dados_anteriores JSON,
    dados_novos JSON,
    utilizador VARCHAR(50),
    data_hora TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    INDEX idx_tabela (tabela_nome),
    INDEX idx_data (data_hora)
);
```

Exemplo de Trigger Implementado (codpostal):

```
-- Trigger para INSERT
CREATE TRIGGER audit_codpostal_insert
AFTER INSERT ON codpostal
FOR EACH ROW
BEGIN
    INSERT INTO audit_log(tabela_nome, operacao, registro_id, dados_novos
)
    VALUES ('codpostal','INSERT', NULL,
        JSON_OBJECT('codpostal', NEW.codpostal,
                    'localidade', NEW.localidade,
                    'cidade', NEW.cidade)
    );
END;

-- Trigger para UPDATE
CREATE TRIGGER audit_codpostal_update
AFTER UPDATE ON codpostal
FOR EACH ROW
BEGIN
    INSERT INTO audit_log(tabela_nome, operacao, registro_id, dados_anter
iores, dados_novos)
    VALUES ('codpostal','UPDATE', NULL,
        JSON_OBJECT('codpostal', OLD.codpostal,
                    'localidade', OLD.localidade,
                    'cidade', OLD.cidade),
        JSON_OBJECT('codpostal', NEW.codpostal,
                    'localidade', NEW.localidade,
                    'cidade', NEW.cidade)
    );
END;

-- Trigger para DELETE
CREATE TRIGGER audit_codpostal_delete
AFTER DELETE ON codpostal
FOR EACH ROW
BEGIN
    INSERT INTO audit_log(tabela_nome, operacao, registro_id, dados_anter
iores)
    VALUES ('codpostal','DELETE', NULL,
        JSON_OBJECT('codpostal', OLD.codpostal,
                    'localidade', OLD.localidade,
                    'cidade', OLD.cidade)
    );
END;
```

2. View de Consulta do Audit Log

Tarefa: Criação de uma view para facilitar a consulta do log de auditoria.

View Implementada:

```sql
CREATE OR REPLACE VIEW view_audit_log AS
SELECT
    id,
    tabela_nome,
    operacao,
    registro_id,
    JSON_PRETTY(dados_anteriores) AS dados_anteriores_formatados,
    JSON_PRETTY(dados_novos) AS dados_novos_formatados,
    utilizador,
    data_hora
FROM audit_log
ORDER BY data_hora DESC;
```

Benefícios: - Formatação automática dos dados JSON - Simplificação de consultas complexas

Exemplo de Utilização:

```sql
-- Ver todas as alterações recentes
SELECT * FROM view_audit_log LIMIT 10;

-- Ver alterações numa tabela específica
SELECT * FROM view_audit_log
WHERE tabela_nome = 'codpostal'
ORDER BY data_hora DESC;

-- Ver alterações num registo específico
SELECT * FROM view_audit_log
WHERE tabela_nome = 'ingredientes' AND registro_id = 5;
```

3. Testes ACID - Consistência e Durabilidade

Tarefa: Implementação de testes automatizados em Python para validar as propriedades ACID da base de dados.

Testes de Consistência Implementados (2 testes):

1. Teste de Consistência 1: Constraints de Chave Estrangeira

```python
def test_consistency_1():
    """Teste 1: Constraints de chave estrangeira devem ser respeitadas"""
    # Tenta criar um prato com restaurante_id inválido
    # Verifica que a foreign key constraint é respeitada
    # Garante integridade referencial
```

Objetivo: Validar que a base de dados rejeita operações que violam integridade referencial.

2. Teste de Consistência 2: Constraints de Valores Únicos (UNIQUE)

```python
def test_consistency_2():
    """Teste 2: Constraints de valores únicos devem ser respeitadas"""
    # Tenta inserir duas categorias com o mesmo nome
    # Verifica que apenas a primeira é aceite
    # Valida constraint UNIQUE em categorias_pratos.nome
```

Objetivo: Garantir que constraints UNIQUE são respeitadas, evitando duplicação de dados críticos.

Testes de Durabilidade Implementados (3 testes):

1. Teste de Durabilidade 1: Persistência após Commit

```
def test_durability_1():
    """Teste 1: Dados commitados persistem após reconexão"""
    # Cria um registo e faz commit
    # Fecha a conexão
    # Reconecta e verifica que os dados persistem
```

Objetivo: Validar que dados commitados são permanentemente armazenados, mesmo após encerramento da conexão.

2. Teste de Durabilidade 2: Múltiplos Commits

```
def test_durability_2():
    """Teste 2: Múltiplos commits persistem corretamente"""
    # Realiza múltiplas operações com commits
    # Verifica que todas as alterações foram persistidas
```

Objetivo: Garantir que múltiplas transações commitadas são todas persistidas corretamente.

3. Teste de Durabilidade 3: Dados Complexos

```
def test_durability_3():
    """Teste 3: Dados complexos (com relações) persistem"""
    # Cria entidades relacionadas (restaurante, pratos, etc.)
    # Faz commit
    # Verifica que todas as relações foram preservadas
```

Objetivo: Validar que estruturas de dados complexas com relações são corretamente persistidas.

Outras Contribuições

Além das responsabilidades principais, contribuí também para:

- **Documentação do Projeto**
- **Desenvolvimento da API**
- **Desenvolvimento do App React Admin**

Principais Dificuldades

1. Implementação do Sistema de Audit

Compreender a sintaxe de triggers em MariaDB e a estrutura JSON para armazenamento de dados.

Aprender a sintaxe específica de triggers em MariaDB (DELIMITER, FOR EACH ROW) - Estruturação correta de objetos JSON para diferentes tipos de operações (INSERT, UPDATE, DELETE) - Gestão de valores NULL em operações INSERT e DELETE

2. Desenvolvimento dos Testes ACID

Criar testes que validem corretamente as propriedades ACID sem interferir com dados de produção.

Compreensão teórica das propriedades ACID e como testá-las na prática - Gestão de transações e commits/rollbacks em Python - Isolamento dos testes para não afetar dados existentes

3. Integração com o Sistema de Replicação

Garantir que os triggers de audit funcionem corretamente em ambiente de replicação Master-Replica.

Compreensão de como os triggers se comportam em réplicas (read-only) - Garantir que o audit_log seja criado apenas no Master - Sincronização dos dados de audit entre Master e Réplicas

Conclusão

O desenvolvimento deste projeto proporcionou uma experiência valiosa no desenvolvimento de sistemas distribuídos, gestão de bases de dados e implementação de funcionalidades críticas como audit e validação de propriedades ACID.

Marta Vieira (a046756@umaia.pt)