UNIVERSIDADE SALVADOR (UNIFACS)

Diogo Da Silva Souza - 12724143402
Caue de Souza Luz - 12724119349
Danilo Queiroz Nogueira - 1272419424
Fausto Bento Torres - 1272521583
Vinicius Lacerda Santos - 12725210686

PROJETO DE SOFTWARE - GELADEIRA INTELIGENTE

SALVADOR
2025

Diogo Da Silva Souza - 12724143402
Caue de Souza Luz - 12724119349
Danilo Queiroz Nogueira - 1272419424
Fausto Bento Torres - 1272521583
Vinicius Lacerda Santos - 12725210686

PROJETO DE SOFTWARE - GELADEIRA INTELIGENTE

Trabalho de Conclusão da Unidade Curricular de Gestão E Qualidade De Software ministrada pelo professor Thiago Neves cujo objetivo é ensinar como desenvolver a qualidade em um software criado pelos alunos, para a avaliação A3.

SALVADOR
2025

Sumário

1. INTRODUÇÃO

No dia que o projeto foi informado pelo professor Thiago já começamos a discutir como seria o projeto. Para começar, nosso projeto propõe a criação de um sistema de Geladeira Inteligente voltado para o controle doméstico de alimentos, permitindo que administradores (pais e mães) tenham controle sobre os itens que serão adicionados, removidos e restringidos. Os usuários clientes (filhos) podem retirar qualquer item não restrito. Todas as retiradas realizadas pelos filhos serão registradas e poderão ser consultadas pelos administradores. O sistema visa promover a organização e segurança alimentar no ambiente familiar, além de registrar o consumo dos filhos de forma automatizada.

2. BANCO DE DADOS

Três dias depois criamos o banco de dados para armazenar as informações das geladeiras, itens e usuários, porém algumas informações como token foram criadas com o passar do tempo. Utilizamos o banco de dados MySql, onde são criadas tabelas para cada tipo de usuário.

No usuário ***Fridge*** temos "**on**" que é um boolean que define se a geladeira está ligada (true) ou desligada (false), "**temperature**" em inteiro que recebe o valor da temperatura, "**id**" que significa o número da geladeira em string além de "**createdAt**" que captura a data e o horário de criação daquela geladeira.

No usuário ***Fridge-Items*** temos o "**name**" que representa o nome daquele item em String, "**ValidDate**" que significa a data da qual o alimento vencerá, "**itemType**" que representa qual o tipo daquele alimento em Cold (Frios - id 1), Freezer (Freezer - id 2), Drinks (Bebidas - id 3), Vegetables (Vegetais - id 4), "**FridgeId**" que é a qual geladeira aquele item estará. "**id"** que significa o número daquele item em string, "**created_at**" que representa o dia e o horário que o item foi criado, "**is_available_for_children**" que significa se o item é acessível às crianças ou não, "**quantity**" que significa a quantidade daquele item.

No usuário ***Users*** temos o "**id**" que representa o número daquele usuário, "**created_at**" que representa quando foi criado, "**email**" que representa o email daquele usuario, "**name**" o nome daquele usuario, "**password**" que é a senha daquele usuario, "**type**" o tipo daquele usuário, "**token**" que significa o token daquele usuário quando o usuário loga no sistema.

Temos também o ***fridge items log*** que registra todas as vezes que um item foi atualizado ou deletado. Foi implementado entre final de outubro e fim de novembro

3. POSTMAN

Decidimos utilizar o software Postman para criar as requisições para cada tipo de usuário e testar as funcionalidades de cada usuário. Url = http://localhost:8080

3.1 Fridges:

- Para criar uma geladeira é necessário inserir se a geladeira está ligada(on) e a temperatura dela no método POST e com a **URL/fridges**. Exemplo:

```
{
        “on” : “true”,
        “temperature” : 11
}
```

- É possível pegar todas as geladeiras com o endpoint **URL/fridges** e com o método GET, também é possível pegar um item específico com o id sendo informado no final do endpoint **URL/fridges/Id** com o mesmo método.

- É possível deletar uma geladeira informando o seu id no fim do endpoint **URL/fridges/Id** no método delete

- É possível atualizar e alterar uma geladeira informando o seu id no corpo da requisição . Exemplo:

```
{
        “id” : 1,
        “on” : “true”,
        “temperature” : 4
}
```

3.2 Users:

- Para criar um usuário é necessário inserir o nome, email, senha e o tipo do usuário no método POST e com a **URL/users**. Exemplo:

{
“name : “Gabriel Silva”,
“email” : “gabrielSilva@gmail.com”,
“password” : “102030”,
“type” : “Parent”
}

- É possível pegar todos os usuários cadastrados com o endpoint **URL/users** e com o método GET, também é possível pegar um item específico com o id sendo informado no final do endpoint **URL/users/Id** com o mesmo método.

- É possível deletar um usuário informando o seu id no fim do endpoint **URL/users/Id** no método delete

- É possível atualizar e alterar as informações de um usuário informando o seu id no endpoint **URL/users/Id**. Exemplo:

{
“name : “Gabriel Silva Santos”,
“email” : “gabrielSilva@gmail.com”,
“password” : “10224030”,
“type” : “Parent”
}

- É possível logar com um usuário no sistema e gerar um token para ele com o endpoint **URL/users/login** no método POST. Exemplo:

{
“email” : “gabrielSilva@gmail.com”,
“password” : “10224030”
}

Após isso o sistema informará o token do usuário!

3.3 Fridge-items:

- Para criar um usuário é necessário inserir o nome, email, senha e o tipo do usuário no método POST e com a **URL/fridge-items**. Exemplo:

{

```
        "name": "Vodka",
        "validDate": "2026-01-09T10:00:00",
        "availableForChildren": false,
        "quantity": 1,
        "itemType": "Drinks",
        "fridge": {
                "id": 1
        }

}
```


- É possível pegar todos os itens cadastrados com o endpoint **URL/fridge-items** e com o método GET, também é possível pegar um item específico com o id sendo informado no final do endpoint **URL/fridge-items/Id** com o mesmo método.

- É possível deletar um item informando o seu id no fim do endpoint **URL/fridge-items/Id** no método delete

- É possível atualizar e alterar as informações de um item informando o seu id no corpo da requisição. Exemplo:


```
{
        "name": "Vodka Com Energético",
        "validDate": "2026-01-19T10:30:00",
        "availableForChildren": false,
        "quantity": 1,
        "itemType": "Drinks",
        "fridge": {
                "id": 1
        }

}
```


4. CONVENTIONAL COMMITS

Durante o início do desenvolvimento fomos orientados pelo professor a utilizar o convencional commits, pois estávamos a enviar commits e pull requests sem especificar se era uma feature nova, um conserto de uma funcionalidade ou até mesmo a criação de um documento. A partir dessa aula começamos a utilizá-los em todos os commits e pull request

5. TDD - DESENVOLVIMENTO ORIENTADO A TESTES

Após finalizarmos os passos anteriores do projeto iniciamos a fase de desenvolvimento orientado a testes, onde nossa primeira tarefa foi testar a branch main na máquina de todos os estudantes da equipe, testando todas as suas funcionalidades e todos os usuários.

Começamos a fazer os testes unitários e de integração nos services implementation, controllers e repository cujo objetivo era rodar a aplicação de ponta a ponta e chamar os endpoints como se fosse um cliente usando o sistema.

Subimos a aplicação com Spring Boot em porta aleatória para os testes. Usamos H2 (banco em memória) para não depender do MySQL real durante os testes e usamos mocks para simular as requisições.
Cenários que testamos:

1. **Fridges:**

    - Criar geladeira: deve retornar 200 e salvar on, temperature, além de preencher createdAt e o id.

    - Atualizar temperatura: se mandar uma temperatura fora da faixa definida, retorna erro (400). Se for válida, salva e atualiza o updatedAt.

    - Buscar por ID: retorna 200 quando existe e 404 quando não existe.

    - Deletar geladeira: 200 se apagar, 404 se não encontrar.

2. **Itens da geladeira (fridge-items):**

    - Criar item: valida name, validDate no futuro, quantity >= 0 e itemType válido. Também checa se a fridge informada existe.

    - Atualizar item: consegue mudar quantity, validDate, availableForChildren e o updatedAt deve ser atualizado.

    - Deletar item: 204 se o item existe e foi apagado; 404 se não existir.

    - Listar itens próximos do vencimento: retorna só os itens cuja validade está perto (por exemplo, próximos 7 dias).

3. **Usuários:**

    - Criar usuário: valida email e o type (Admin/Children). Se estiver ok, retorna 200.

    - Consultar e deletar: respostas padrão (200/204/404) dependendo do caso.

    - Usuário do tipo Children não pode retirar item marcado como indisponível para crianças (availableForChildren = false). Nesse caso, a API retorna erro.

4. **Mudanças:**

- Apos esses testes que ocorreram em meados de novembro desenvolvemos o log de fridge items para salvar o nome do usuario que atualizou ou deletou algum item. Tambem fizemos melhorias na seção de autenticação dos usuarios.

6. TESTES COM KARATE DSL

Em meados de novembro criamos mais testes de integração, porem com o auxílio do framework karate dsl. Nele simulamos os testes de integração ja existentes com as sintaxes GIVEN para aa urls da api, WHEN que informa o método daquela requisição como post ou get, e THEN que retorna o resultado da requisição como 200, 500 etc.

7. CONCLUSÃO

O projeto da "Geladeira Inteligente" atingiu seu objetivo principal de demonstrar a aplicação de conceitos robustos de Gestão e Qualidade de Software no desenvolvimento de uma API REST funcional e segura.

Iniciamos com a definição do escopo, focando no controle doméstico de alimentos com diferenciação de perfis (Administrador e Cliente). A implementação da estrutura do Banco de Dados (MySQL) forneceu a base para o gerenciamento de Fridges, Users e Fridge-Items, incluindo a crucial tabela de fridge items log para rastreamento de consumo e alterações.

O uso do Postman permitiu a validação manual dos endpoints de CRUD para cada entidade. Um pilar fundamental na qualidade do projeto foi a adoção de metodologias de desenvolvimento: Conventional Commits trouxe clareza ao histórico e TDD (Desenvolvimento Orientado a Testes) foi rigorosamente aplicado. A utilização de testes unitários e de integração com Spring Boot e H2 assegurou que cenários críticos, como a restrição de acesso a itens por crianças e validação de dados, fossem cobertos. Por fim, o Karate DSL encerrou o ciclo de testes com a criação de testes de integração ponta a ponta em sintaxe Gherkin, criando uma suíte de testes de regressão de fácil manutenção.

Em suma, este trabalho demonstrou a capacidade da equipe em traduzir requisitos de negócio em uma solução técnica eficiente, utilizando um stack moderno de desenvolvimento e, mais importante, aplicando práticas de qualidade de software que garantem a estabilidade, confiabilidade e facilidade de

manutenção do sistema. O resultado é uma API de "Geladeira Inteligente" pronta para oferecer controle e segurança alimentar para a família.