PROGRAMAÇÃO 2

CIÊNCIA DA COMPUTAÇÃO
ERICK KEVEN DA SILVA ALVES
SANDRO GOMES PAULINO

RELATÓRIO DE PROGRAMAÇÃO 2
MYFOOD

MACEIÓ/AL 2024

1. INTRODUÇÃO

O MyFood é um sistema de delivery de produtos, o sistema permite a realização de cadastro de empresas (restaurantes, mercados, farmácias, etc..), inicialmente permitindo apenas restaurantes, cadastro de clientes e donos das empresas. Além da criação e edição de pedidos e produtos dos dados associados a eles. O projeto foi desenvolvido usando Desenvolvimento Orientado a Testes, possuindo uma simulação de banco de dados do sistema usando XML, garantindo a persistência de dados.

2. DESIGN ARQUITETURAL DO SISTEMA

Utilizando a Facade para organizar o sistema de maneira mais centralizada, fazendo com que o subsistema do projeto fique mais unificado e simplificado. Para garantir a organização, manutenção e reutilização do projeto, o projeto foi organizado por camadas, separando-as em Controller, Exception, Model,repository e service e interagindo com a Facade.

2.1. Controller

Atua como intermediário entre a camada Facade e as diversas classes do sistema. Ela gerencia as requisições do usuário e coordena a interação entre a Facade e os serviços necessários para processar essas requisições.

2.2. Exception

Descrição:

É responsável por gerenciar e lidar com todas as exceções e situações inesperadas que possam ocorrer durante a execução do sistema. Isso facilita a manutenção e garante que o sistema trate erros de forma centralizada e organizada.

Responsabilidades:

- Fornecer mensagens de erro claras e compreensíveis para os usuários.

- Manter o código limpo e reduzir o acoplamento entre diferentes partes do sistema ao lidar com erros de forma centralizada.

2.3. Model

Descrição:

Contém as principais classes do projeto, representando as entidades e a estrutura de dados do sistema. Esta camada é responsável por definir as classes e suas propriedades, bem como o relacionamento entre elas.

Responsabilidades:

- Definir as entidades e seus atributos (por exemplo, User, Company, Product, Order).
- Gerenciar a lógica de validação e manipulação dos dados.

2.4. Repository

Descrição:

Fornece uma abstração sobre a lógica de armazenamento e a persistência de dados. Utiliza XML para simular o banco de dados e fornece métodos para acessar e modificar os dados.

Responsabilidades:

- Implementar métodos para salvar, carregar e excluir dados.
- Gerenciar a persistência de dados em arquivos XML.
- Fornecer uma interface clara para o acesso aos dados sem expor os detalhes da implementação de armazenamento.

2.5. Service

Descrição:

É responsável pela lógica de negócios da aplicação. Manipula as regras de negócios, validações e coordena as operações entre as diferentes partes do sistema.

Responsabilidades:

- Implementar a lógica de negócios e regras de validação.
- Coordenar operações complexas que envolvem múltiplas entidades ou serviços.
- Fornecer uma interface para que os Controllers acessem e manipulem dados de forma encapsulada.

2.6. Facade

Descrição:

O padrão Facade é utilizado para criar uma interface simplificada e unificada para interagir com o subsistema do MyFood. Isso ajuda a ocultar a complexidade do sistema e facilita a utilização das funcionalidades principais.

Responsabilidades:

- Fornecer uma interface única para acessar funcionalidades dos diferentes subsistemas.
- Simplificar a interação com o sistema, reduzindo a necessidade de interação direta com múltiplas classes.
- Facilitar a manutenção e evolução do sistema ao centralizar a complexidade.

3. PRINCIPAIS COMPONENTES E SUAS INTERAÇÕES

3.1. User, Customer e CompanyOwner

- **User**
  - Descrição: Classe abstrata que define os atributos e comportamentos básicos para todos os usuários do sistema. Ela não pode ser instanciada diretamente e serve como uma classe base para outras classes.
  - Atributos: Nome, email, senha, endereço.
  - Interações: Fornece uma base comum para Customer e CompanyOwner, garantindo que ambos herdem os atributos essenciais de um usuário.

- **Customer**
    - Descrição: Herda de User e adiciona atributos específicos para um cliente.
    - Atributos: Além dos atributos herdados de User, a classe Customer inclui um atributo para armazenar os pedidos realizados.
    - Interações: Herda funcionalidades básicas de User e adiciona funcionalidades específicas relacionadas aos pedidos de um cliente.

- **CompanyOwner**
    - Descrição: Herda de User e representa os donos de empresas.
    - Atributos: Além dos atributos herdados de User, inclui o CPF do dono e uma lista de empresas que ele possui.
    - Interações: Herda funcionalidades básicas de User e adiciona atributos e funcionalidades relacionados à gestão das empresas.

3.2. Company e Restaurant

- **Company**
    - Descrição: Classe base para representar empresas, fornecendo uma estrutura comum para outras classes relacionadas a empresas.
    - Atributos: Tipo de empresa, dono (referência a CompanyOwner), nome, endereço, listas de produtos e pedidos.
    - Interações: Serve como base para Restaurant, permitindo que Restaurant herde seus atributos e funcionalidades.

- **Restaurant**
    - Descrição: Herda de Company e representa um tipo específico de empresa.
    - Atributos: Além dos atributos herdados de Company, inclui o tipo de cozinha do restaurante.

- Interações: Adiciona atributos específicos relacionados ao tipo de cozinha, complementando a estrutura básica fornecida por Company.

3.3. Order e Product

- **Order**
  - Descrição: Representa um pedido realizado no sistema.
  - Atributos: Número do pedido, cliente (referência a Customer), empresa (referência a Company), status do pedido, lista de produtos e valor total.
  - Interações: Gerencia todos os dados associados a um pedido, incluindo a relação com o cliente e a empresa, bem como os produtos incluídos.
- **Product**
  - Descrição: Representa um produto disponível em uma empresa.
  - Atributos: Nome, valor, categoria.
  - Interações: Está associado a uma empresa (Company) e é incluído em pedidos (Order), fornecendo detalhes sobre cada produto.

3.4. UserService

- Descrição: Responsável por gerenciar a criação e autenticação de usuários.
- Responsabilidades:
  - Criar novos usuários (clientes ou donos de empresas).
  - Recuperar informações de usuários existentes.
  - Realizar login e validações necessárias.
- Interações: Interage com User e suas subclasses (Customer, CompanyOwner) para gerenciar a criação e autenticação de usuários.

3.5. CompanyService

- Descrição: Gerencia operações relacionadas a empresas.
- Responsabilidades:
  - Criar novas empresas.

    - Listar empresas associadas a um determinado dono.
    - Recuperar informações de empresas existentes.
    - Obter o ID de uma empresa.
- Interações: Trabalha com Company e suas subclasses (Restaurant) para gerenciar informações e operações relacionadas às empresas.

3.6. ProductService

- Descrição: Gerencia operações relacionadas a produtos.
- Responsabilidades:
    - Criar e adicionar novos produtos a uma empresa.
    - Editar produtos existentes.
    - Verificar informações de produtos.
    - Listar produtos de uma empresa específica.
- Interações: Interage com Product e Company para adicionar, modificar e listar produtos.

3.7. OrderService

- Descrição: Gerencia operações relacionadas a pedidos.
- Responsabilidades:
    - Criar novos pedidos.
    - Visualizar e gerenciar pedidos existentes.
    - Adicionar e remover produtos de um pedido.
    - Gerenciar o status dos pedidos.
- Interações: Trabalha com Order e Product para criar e gerenciar pedidos, incluindo a adição e remoção de produtos.

3.8. Data e Persistence

- **Data**
    - Descrição: Classe responsável por criar e gerenciar a instância única do banco de dados simulado, armazenando listas de usuários e empresas.
    - Responsabilidades:
        - Gerenciar a instância de dados.
        - Fornecer acesso às listas de usuários e empresas.

- Interações: Utiliza Persistence para salvar, carregar e apagar dados.

- **Persistence**
  - Descrição: Classe responsável pela manipulação de arquivos para persistência de dados.
  - Responsabilidades:
    - Salvar dados em um arquivo XML.
    - Carregar dados de um arquivo XML.
    - Apagar dados quando necessário.
  - Interações: Interage diretamente com Data para realizar operações de persistência e carregamento de dados.

3.9. MyFoodSystem

- Descrição: Classe principal do sistema que coordena as interações entre a Facade e o restante do sistema.
- Responsabilidades:
  - Orquestrar ações do sistema e interação com a Facade.
  - Gerenciar a execução das funcionalidades principais do MyFood.
- Interações: Serve como o ponto central de controle, integrando a Facade com os serviços e operações do sistema.

4. PADRÕES DE PROJETO

4.1. Facade

- **Descrição Geral:** O padrão Facade fornece uma interface simplificada para um sistema complexo, oferecendo uma camada de intermediação que oculta as complexidades e expõe uma interface mais amigável ao cliente. Este padrão é utilizado para realizar testes de forma eficiente e simplificada.
- **Problema Resolvido:** O Facade lida com a complexidade dos sistemas e subsistemas, facilitando a interação com o sistema ao esconder as complexidades internas. Ele proporciona um acesso

simplificado às funcionalidades do sistema, reduzindo a necessidade de entender as complexidades subjacentes.

- **Identificação da Oportunidade:** O projeto MyFood envolve diferentes modelos e processos complexos, com várias validações e interações. A aplicação do padrão Facade facilita o acesso a todos os processos do sistema através de métodos que atendem aos requisitos de todo o sistema, proporcionando uma interface mais amigável para os usuários e para a realização de testes.
- **Aplicação no Projeto:** A Facade foi implementada para fornecer uma interface única e simplificada que coordena as interações entre o cliente e o sistema, abstraindo a complexidade e oferecendo métodos que permitem a realização de operações sem a necessidade de entender a lógica interna detalhada.

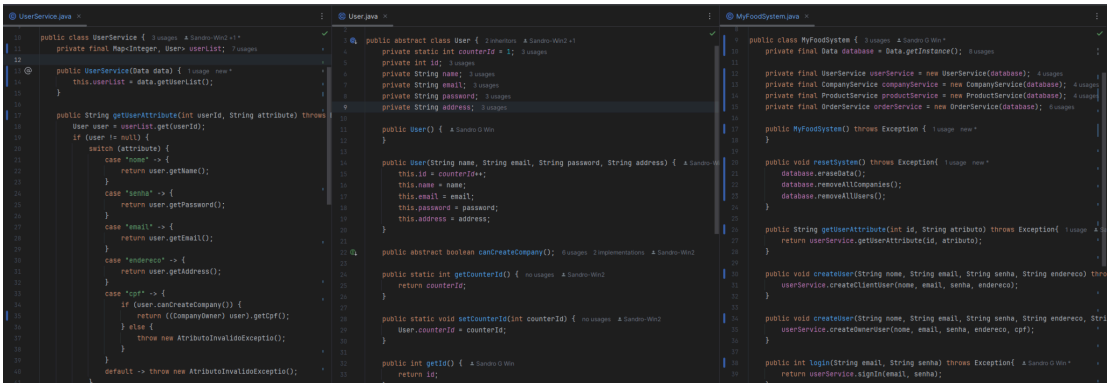
```
public void zerarSistema() throws Exception {  no usages  Sandro G Win *
    sys.resetSystem();
}

public String getAtributoUsuario(int id, String atributo) throws Exception {  no usages  Sandro G Win *
    return sys.getUserAttribute(id, atributo);
}

public void criarUsuario(String nome, String email, String senha, String endereco) throws Exception {  no usages
    sys.createUser(nome, email, senha, endereco);
}

public void criarUsuario(String nome, String email, String senha, String endereco, String cpf) throws Exception {
    sys.createUser(nome, email, senha, endereco, cpf);
}

public int login(String email, String senha) throws Exception {  Sandro G Win *
    return sys.login(email, senha);
}

public void encerrarSistema() throws Exception {  no usages  Sandro G Win *
    sys.endSystem();
}
```

4.2. MVC (Model-View-Controller)

- **Descrição Geral**: O padrão MVC divide uma aplicação em três componentes principais:
    - Model (Modelo): Gerencia os dados e as regras de negócio.
    - View (Visão): Cuida da apresentação e da interface com o usuário.
    - Controller (Controlador): Atua como intermediário entre o Model e a View, manipulando a entrada do usuário e atualizando o Model.

- **Problema Resolvido:** O MVC ajuda a manter o código organizado e modular, facilitando a manutenção e a implementação de novas funcionalidades. Ele permite que diferentes partes da aplicação sejam desenvolvidas e modificadas independentemente, sem causar efeitos colaterais em outras partes do sistema.
- **Identificação da Oportunidade:** Durante o desenvolvimento do MyFood, foi identificado que a separação das responsabilidades entre a lógica de negócios, a manipulação de dados e a interface do usuário facilitaria a manutenção e a expansão do sistema. O MVC foi utilizado para garantir essa separação de responsabilidades.

  No caso do projeto na separação por camadas, o MVC foi utilizado como base para fazer essa separação. Usado nas classes em Model, Controller e Service, que não necessariamente faz parte do MVC.



4.3. Singleton

- **Descrição Geral:** O padrão Singleton garante que uma classe tenha apenas uma instância e que essa instância seja acessível globalmente. É útil quando é necessário garantir uma única fonte de verdade para dados ou configurações durante todo o ciclo de vida da aplicação.
- **Problema Resolvido:** O Singleton evita a criação de múltiplas instâncias de uma classe, garantindo a consistência dos dados e a centralização da gestão, o que é crucial em situações onde uma única instância é necessária para coordenar dados ou configurações.

- **Identificação da Oportunidade:** No projeto MyFood, foi necessário ter uma única instância da classe Data para gerenciar e armazenar informações sobre usuários e empresas de forma centralizada. O uso do Singleton garantiu que todas as operações relacionadas a dados fossem coordenadas por uma única instância, evitando problemas de consistência.
- **Aplicação no Projeto:** A classe Data implementa o padrão Singleton através de:
    - Variável de Instância Estática: private static volatile Data instance.
    - Método Sincronizado: getInstance() para garantir que apenas uma instância da classe seja criada e acessada globalmente.

```java
public static synchronized Data getInstance() throws Exception {
    if (instance == null) {
        instance = Persistence.loadData();
        if (instance == null) {
            instance = new Data();
        }
    }
    return instance;
}
```