

**UNIVERSIDADE FEDERAL DE ALAGOAS**  
**INSTITUTO DE COMPUTAÇÃO**

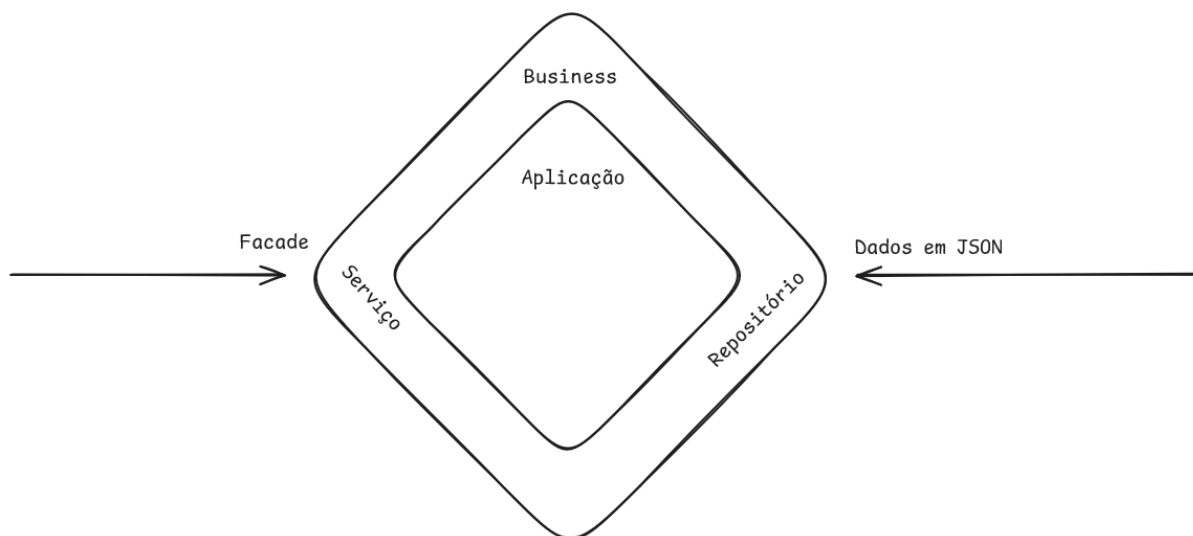
Disciplina: Programação 2  
Professor: MARIO HOZANO LUCAS DE SOUZA

**RELATÓRIO DO PROJETO MYFOOD**

RICARDO VINICIUS DE ALMEIDA FERNANDES  
*rvaf@ic.ufal.br*

## 1 Descrição Geral do Design Arquitetural do Sistema

A arquitetura do projeto foi desenvolvida utilizando alguns princípios básicos de Domain Driven Design, objetivando separar em camadas de domínio o projeto. De forma simples o projeto foi separado na camada de serviço, que lida com a lógica de validação básica dos dados passados e a lógica de negócio; e na camada de repositório, que lida com a obtenção e persistência dos dados no sistema.

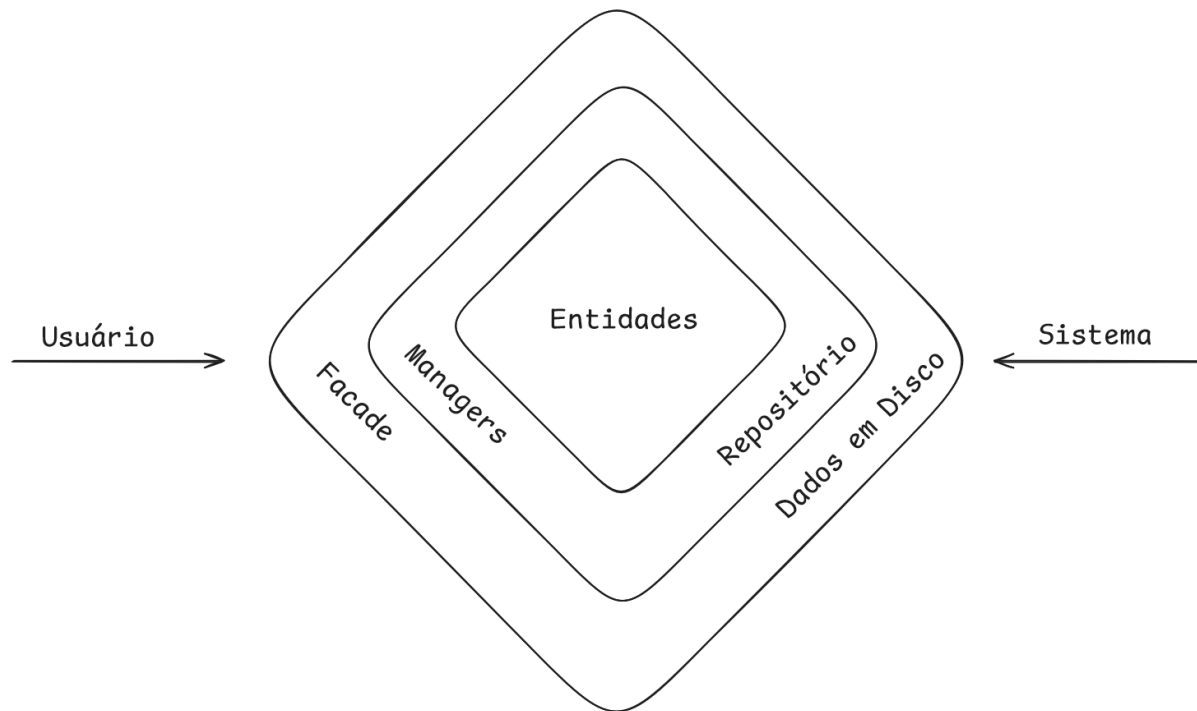


A responsabilidade pela lógica de negócio ficou atrelada ao *manager*, enquanto as lógicas de aplicação ficaram associadas às entidades em si. De modo que a *Facade* serve como uma camada de abstração para o usuário poder realizar ações no sistema.

A camada de repositório foi implementada através de uma classe abstrata genérica, que fornece um comportamento base para persistência de dados em JSON, usando como auxiliar a biblioteca *Jackson Databind*, que fornece o método *ObjectMapper* que permite um mecanismo fácil de mapeamento dos POJOs para JSON, e versa. Através desta classe

genérica, é possível criar repositórios específicos para cada classe do sistema, de maneira fácil e modular.

Para cada entidade do sistema apresentada, foi criado um *model*, que define os seus campos de dados e métodos específicos. Todas as classes de entidade herdam de uma classe abstrata genérica *Persistent* que define uma base para os objetos persistentes do sistema.



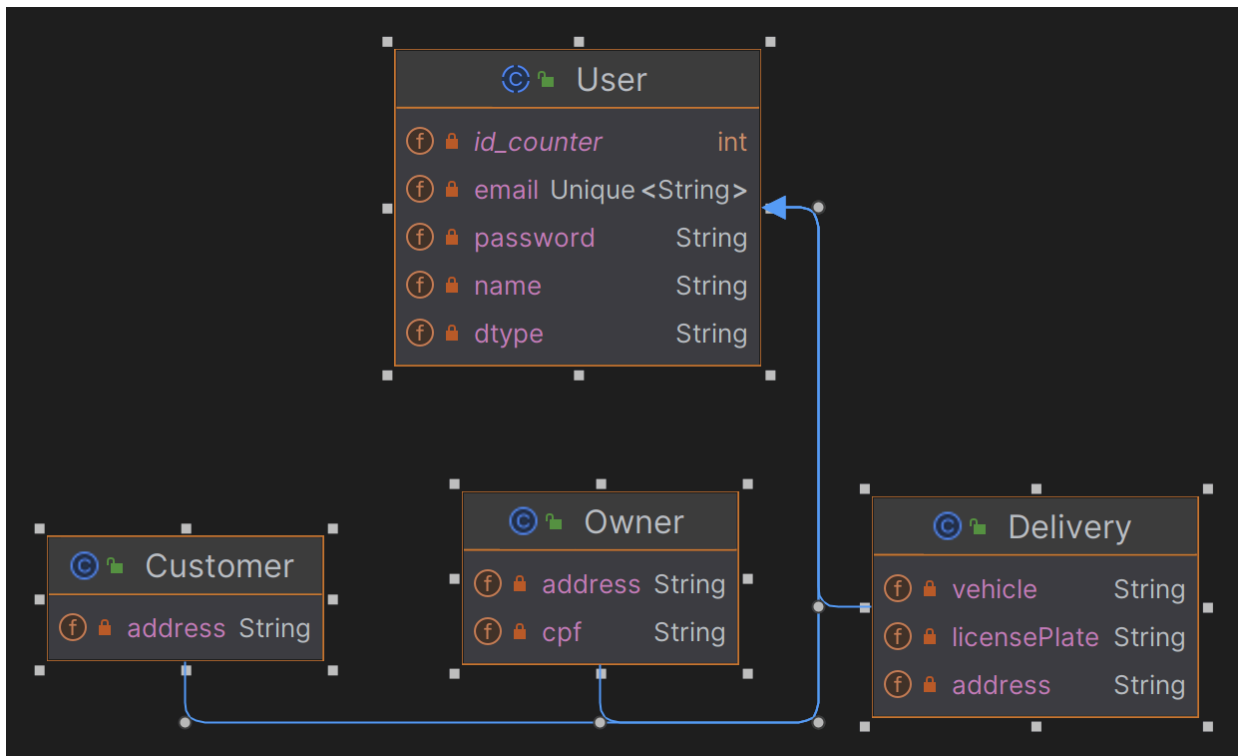
Essas três camadas definem a arquitetura geral do sistema.

## 2 Principais Componentes e suas Interações

### 2.1 Entidades

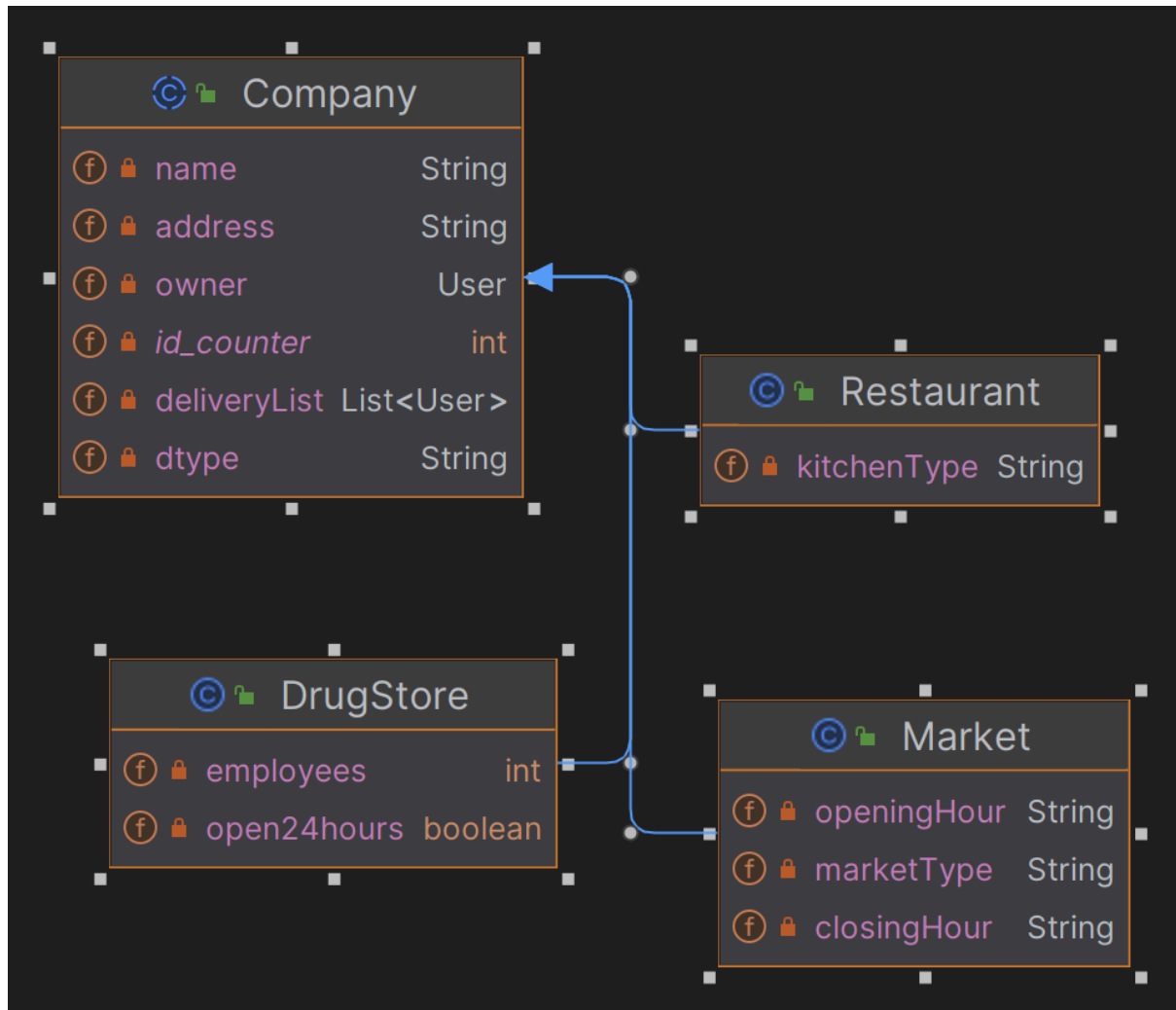
#### 2.1.1 User

A entidade *User* define os comportamentos básicos de usuário, e os campos comuns. É herdada pelas entidades *Customer*, *Owner* e *Delivery*, cada uma com seus comportamentos e campos específicos.



### 2.1.2 Company

A entidade *Company* define os comportamentos e campos básicos das empresas no sistema, e é herdada por *Restaurant*, *DrugStore* e *Market* que exibem campos distintos entre si.



### 2.1.3 Product

A entidade de *Product* descreve os campos e comportamentos dos produtos do sistema. Não é herdada por nenhum outro, porém possui relacionamentos.

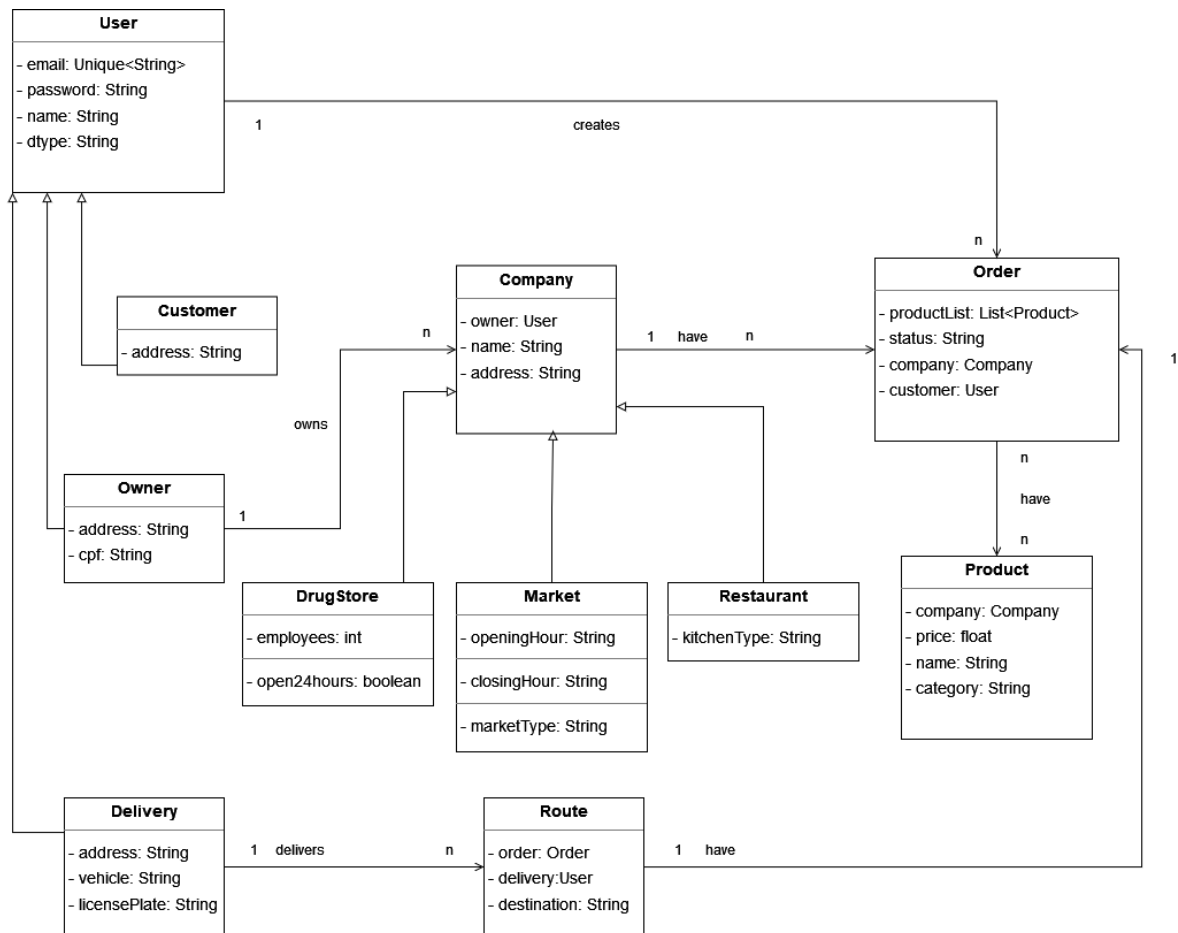
### 2.1.4 Order

A entidade de *Order* descreve os campos e comportamentos dos pedidos do sistema.

#### 2.1.4 Route

A entidade de *Order* representa as entregas dentro do sistema, estando atrelada a um pedido e um entregador.

## 2.2 Relacionamentos



## 2.3 Tipos

### 2.3.1 Unique

O tipo *Unique* foi criado como uma classe que envolve outras classes e dá a elas a característica de serem um campo único na persistência dos dados da entidade que pertence.

### 2.3.2 Persistent

O tipo *Persistent* é a base para todas as entidades persistentes do sistema, sendo pré-requisito para serem processadas pelo *Repository*. Fornece a obrigatoriedade de possuir um identificador e métodos para filtrar se existem campos únicos (*Unique*) repetidos.

## 2.4 Repository

Os repositórios que lidam com a persistência e obtenção dos dados estendem uma classe pai chamada *Repository*, que serve os métodos e campos base para a persistência dos

dados em JSON. Existe um repositório para cada grande classe das entidades, com métodos que auxiliam na obtenção dos dados como POJOs ou iteráveis de Java.

## 2.5 Outros

Foram implementados alguns outros artefatos auxiliares como o pacote *Validators* com algumas classes úteis para validação de dados enviados para os *managers*. Também foram implementadas exceções personalizadas para aquelas que não tinham correspondente adequado no pacote padrão do Java.

## 3 Explicação sobre os padrões de projetos adotados

### 3.1 Facade

Descrição geral: O padrão Facade fornece uma interface simplificada para interações com subsistemas complexos, encapsulando as operações em um único ponto de controle. No MyFood, a classe Facade delega as operações principais para os respectivos *managers*, como criação de usuários, empresas, produtos, e pedidos.

Problema resolvido: O padrão Facade ajuda a esconder a complexidade do sistema ao usuário, permitindo que ações que envolveriam múltiplas interações com várias classes sejam realizadas através de uma interface única e simplificada.

Identificação da Oportunidade: Através da descrição do projeto.

Aplicação no Projeto: A classe Facade foi implementada para servir como ponto de acesso a todas as operações principais, utilizando os repositórios e manipulando a lógica de negócios e validação de dados. Isso reduz a complexidade do código cliente, que não precisa lidar diretamente com a interação entre entidades, repositórios e validações.

### 3.2 Repository

Descrição geral: O padrão Repository abstrai o acesso a dados, encapsulando as operações de persistência e recuperação. Ele oferece uma interface genérica para manipulação de objetos do domínio sem expor os detalhes de persistência, como o formato JSON usado no MyFood.

Problema resolvido: Facilita o acesso a dados e sua manipulação de forma genérica, permitindo que a camada de aplicação não precise lidar diretamente com as complexidades de persistência.

Identificação da Oportunidade: O padrão Repository foi adotado para evitar a duplicação de código de persistência e oferecer uma maneira flexível de armazenar diferentes

tipos de dados (por exemplo, User, Company, Product), preservando a separação de preocupações.

**Aplicação no Projeto:** Cada entidade principal do sistema possui um repositório correspondente (como UserRepository, CompanyRepository), que estende a classe genérica Repository. A persistência dos dados é feita em JSON, utilizando a biblioteca Jackson. O repositório oferece métodos especializados para busca e manipulação de dados, como filtragem por atributos únicos ou obtenção de listas relacionadas a uma chave estrangeira (como os produtos de uma empresa).

### **3.3 Factory**

**Descrição geral:** O padrão Factory é utilizado para criar objetos sem expor a lógica de instanciamento ao cliente. Ele centraliza a criação de objetos de forma flexível e reutilizável.

**Problema resolvido:** A Factory evita que a lógica de criação de objetos esteja espalhada pelo código e facilita a adição de validações ou configurações específicas no momento da criação.

**Identificação da Oportunidade:** A Factory foi aplicada no sistema MyFood para padronizar a criação de entidades como Customer, Owner, Company e Product, garantindo que todos os objetos sejam criados de forma consistente, com as validações necessárias.

**Aplicação no Projeto:** Métodos create foram implementados nas classes de entidades, que garantem que os objetos sejam criados com os valores e estados adequados, além de realizar as validações no momento da criação.

### **3.4 Singleton**

**Descrição geral:** O padrão Singleton assegura que uma classe tenha apenas uma única instância, fornecendo um ponto global de acesso a ela.

**Problema resolvido:** O Singleton garante que o sistema tenha apenas uma instância dos repositórios, evitando problemas de inconsistência ou duplicação de dados durante a execução do sistema.

**Identificação da Oportunidade:** A necessidade de repositórios centralizados e acessíveis por todo o sistema motivou o uso do padrão Singleton, garantindo que todos os módulos do sistema manipulem as mesmas instâncias de repositório.

**Aplicação no Projeto:** Todos os repositórios (UserRepository, CompanyRepository, ProductRepository, OrderRepository) utilizam o padrão Singleton, de modo que a instância desses objetos é compartilhada por toda a aplicação. Isso garante que os dados persistidos sejam consistentes e manipulados corretamente durante a execução.