

---

# Documentação de Projeto

para o sistema

# Picnic Caseiros

**Versão 3.0**

Projeto de sistema elaborado pelo(s) aluno(s) Douglas Fernandes, Davi Brandão, Guilherme Costa, Filipe Iannarelli e Raul Goulart como parte da disciplina **Projeto de Software**.

23 de junho de 2024

## Tabela de Conteúdo

1.	Introdução .....	1
2.	Modelos de Usuário e Requisitos .....	1
2.1	Descrição de Atores .....	1
2.2	Modelo de Casos de Uso.....	1
2.3	Diagrama de Sequência do Sistema.....	3
3.	Modelos de Projeto .....	6
3.1	Arquitetura .....	6
3.2	Diagrama de Componentes e Implantação .....	8
3.3	Diagrama de Classes .....	10
3.4	Diagramas de Sequência .....	12
3.5	Diagramas de Comunicação .....	18
3.6	Diagramas de Estados .....	19
4.	Modelos de Dados.....	19

## Histórico de Revisões

Nome	Data	Razões para Mudança	Versão
Guilherme Costa	20/03/2024	Emissão inicial	1.0
Guilherme Costa	31/05/2024	Emissão da versão revisada do documento	2.0
Guilherme Costa	23/06/2024	Emissão da versão final do documento	3.0

## 1. Introdução

Este documento agrega a elaboração e revisão de modelos de domínio e de projeto para o sistema Picnic Caseiros. Nosso projeto visa desenvolver um sistema sob medida para a Picnic Caseiros, uma empresa dedicada à produção e venda de bolos caseiros e outros alimentos artesanais. Nosso objetivo é otimizar a gestão de insumos, receitas, vendas e fidelização de clientes, impulsionando o crescimento sustentável e a eficiência operacional.

## 2. Modelos de Usuário e Requisitos

### 2.1 Descrição de Atores

Na modelagem desenvolvida para o projeto, foram mapeados dois atores distintos:

- Usuário: pode apenas realizar agendamentos de pedidos pela aplicação;
- Administrador: pode realizar todas as demais tarefas previstas, como gerenciar os insumos, receitas, formas de pagamentos, dentre outros.

### 2.2 Modelo de Casos de Uso

Nessa seção será apresentado o diagrama de caso de uso do projeto. Neste diagrama são elencados os atores que participam da interação, bem como as suas possíveis ações. Conforme pode ser observado na Figura 1 a seguir, o ator Usuário possui acesso às funcionalidades relacionadas ao agendamento de pedidos. O ator Administrador, que herda as permissões de Usuário, pode interagir com diversas outras funcionalidades, como monitorar o processo de fidelização dos clientes, gerenciar os insumos cadastrados, gerenciar as formas de venda dos produtos, gerenciar as vendas realizadas, gerenciar as receitas, gerenciar a precificação dos produtos e gerenciar fornecedor e clientes. Outro ponto relevante, é que o Administrador também pode gerenciar e emitir relatórios das funcionalidades de fidelização de clientes, gerenciamento de insumos, formas de venda e vendas realizadas.

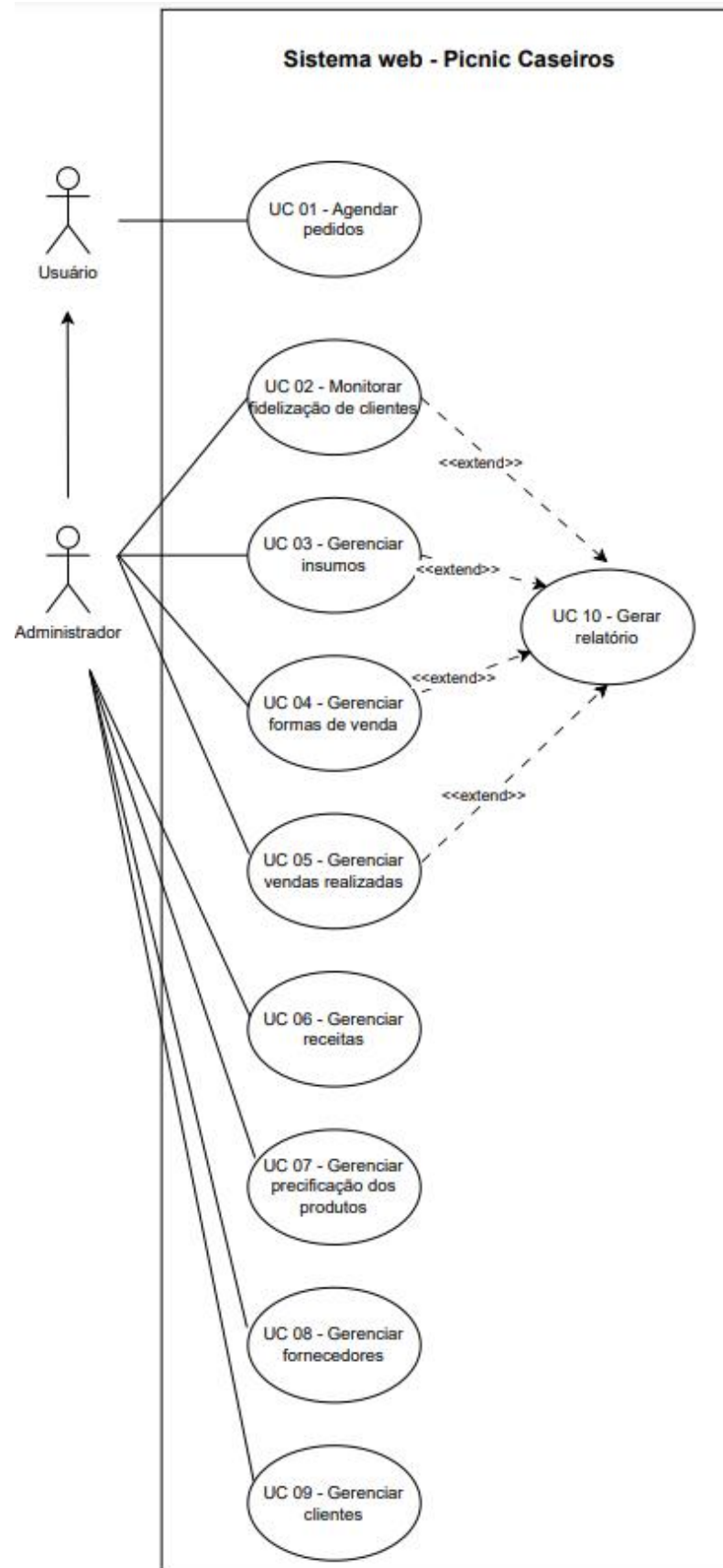


Figura 1 - Diagrama de casos de uso do projeto

## 2.3 Diagrama de Sequência do Sistema

A seguir são apresentados diagramas de sequência do sistema para três cenários distintos de execução, sendo eles:

- Administrador deseja criar um produto;
- Administrador gerencia fornecedores;
- Administrador gerencia clientes.

Para cada um desses diagramas, são tabelas de contrato que relacionam e descrevem detalhadamente os casos de uso envolvidos, fluxo de tarefas, dentre outros.

O primeiro caso de uso abordado é o de criar produto, conforme ilustrado pela Figura 2 a seguir.

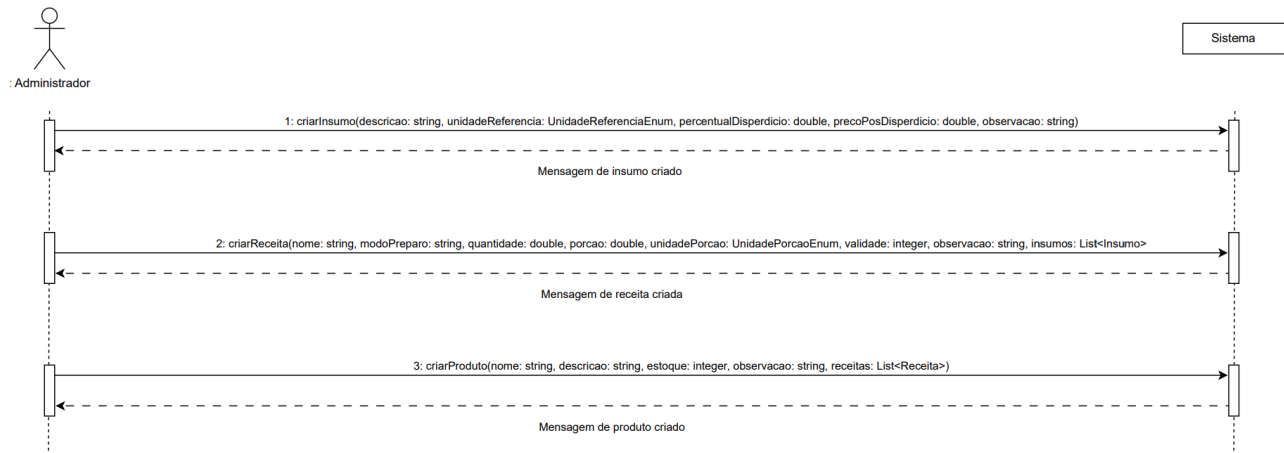


Figura 2 - Diagrama de sequência do sistema - Criar um produto

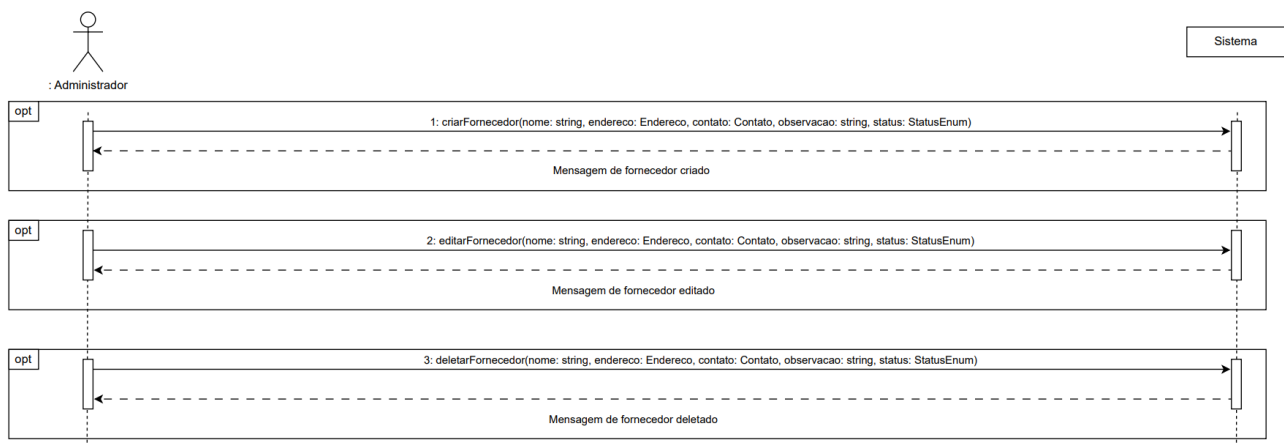
Contrato	Criar um insumo
Operação	criarInsumo(descricao: string, unidadeReferencia: UnidadeReferenciaEnum, percentualDisperdicio: double, precoPosDisperdicio: double, observacao: string)
Referências cruzadas	UC-03 – Gerenciar insumos
Pré-condições	Usuário estar autenticado
Pós-condições	Um novo insumo deve ser criado e uma mensagem indicando que o insumo foi criado deve aparecer no canto inferior direito da tela.

Contrato	Criar uma receita
----------	-------------------

Operação	criarReceita(nome: string, modoPreparo: string, quantidade: double, porcao: double, unidadePorcao: UnidadePorcaoEnum, validade: integer, observacao: string, insumos: List<Insumos>)
Referências cruzadas	UC-06 – Gerenciar receitas
Pré-condições	Usuário estar autenticado e lista de insumos previamente cadastrada
Pós-condições	Uma nova receita deve ser criada e uma mensagem indicando que a receita foi criada deve aparecer no canto inferior direito da tela.

Contrato	Criar um produto
Operação	criarProduto(nome: string, descricao: string, estoque: integer, observacao: string, receitas: List<Receita>)
Referências cruzadas	UC-07 – Gerenciar receitas
Pré-condições	Usuário estar autenticado e lista de receitas previamente cadastradas
Pós-condições	Um novo produto deve ser criado e uma mensagem indicando que o produto foi criado deve aparecer no canto inferior direito da tela.

O segundo diagrama de sequência do sistema elaborado diz respeito ao gerenciamento de fornecedores, conforme ilustrado na figura a seguir.



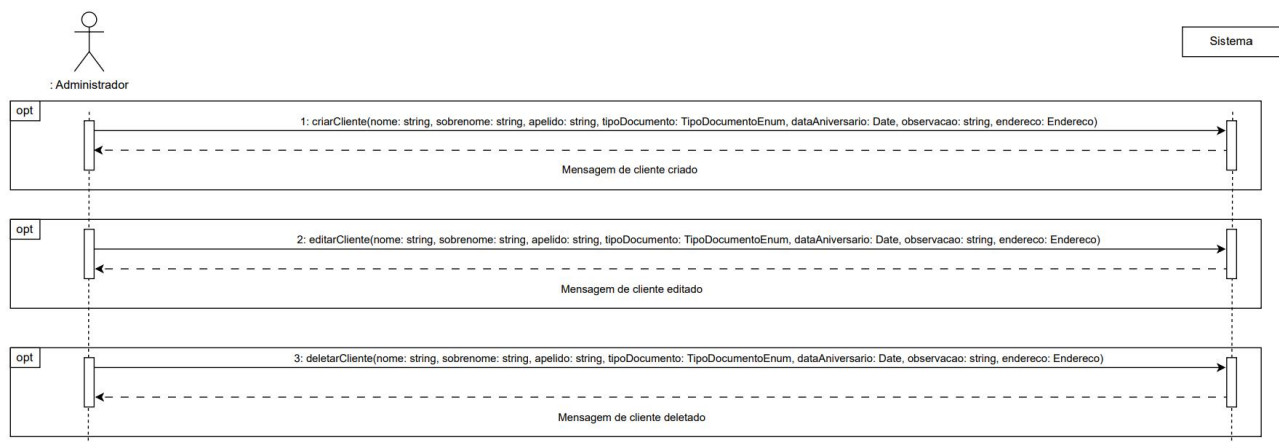
Contrato	Criar um fornecedor
Operação	criarFornecedor(nome: string, endereco: Endereco, contato: Contato, observacao: string, status: StatusEnum)
Referências cruzadas	UC-08 – Gerenciar fornecedores
Pré-condições	Usuário estar autenticado e lista de receitas previamente cadastradas
Pós-condições	Um novo fornecedor deve ser criado e uma mensagem indicando que o fornecedor foi criado deve aparecer no canto inferior direito da tela.

Contrato	Editar um fornecedor
----------	----------------------

Operação	editorFornecedor(nome: string, endereco: Endereco, contato: Contato, observacao: string, status: StatusEnum)
Referências cruzadas	UC-08 – Gerenciar fornecedores
Pré-condições	Usuário estar autenticado e lista de receitas previamente cadastradas
Pós-condições	Os dados do fornecedor devem ser atualizados e uma mensagem indicando que o fornecedor foi atualizado deve aparecer no canto inferior direito da tela.

Contrato	Deletar um fornecedor
Operação	deletarFornecedor(nome: string, endereco: Endereco, contato: Contato, observacao: string, status: StatusEnum)
Referências cruzadas	UC-08 – Gerenciar fornecedores
Pré-condições	Usuário estar autenticado e lista de receitas previamente cadastradas
Pós-condições	O fornecedor deve ser excluído e uma mensagem indicando que o fornecedor foi excluído deve aparecer no canto inferior direito da tela.

Por fim, de maneira similar ao gerenciamento de fornecedor, foi elaborado o diagrama para o gerenciamento de clientes.



Contrato	Criar um cliente
Operação	criarCliente(nome: string, sobrenome: string, apelido: string, tipoDocumento: TipoDocumentoEnum, dataAniversario: Date, observacao: string, endereco: Endereco)
Referências cruzadas	UC-09 – Gerenciar clientes
Pré-condições	Usuário estar autenticado e lista de receitas previamente cadastradas
Pós-condições	O cliente deve ser criado e uma mensagem indicando que o cliente foi criado deve aparecer no canto inferior direito da tela.

Contrato	Editar um cliente
Operação	editarCliente(nome: string, sobrenome: string, apelido: string, tipoDocumento: TipoDocumentoEnum, dataAniversario: Date, observacao: string, endereco: Endereco)
Referências cruzadas	UC-09 – Gerenciar clientes
Pré-condições	Usuário estar autenticado e lista de receitas previamente cadastradas
Pós-condições	Os dados do cliente devem ser atualizados e uma mensagem indicando que o cliente foi atualizado deve aparecer no canto inferior direito da tela.

Contrato	Deletar um cliente
Operação	criarCliente(nome: string, sobrenome: string, apelido: string, tipoDocumento: TipoDocumentoEnum, dataAniversario: Date, observacao: string, endereco: Endereco)
Referências cruzadas	UC-09 – Gerenciar clientes
Pré-condições	Usuário estar autenticado e lista de receitas previamente cadastradas
Pós-condições	O cliente deve ser excluído e uma mensagem indicando que o cliente foi excluído deve aparecer no canto inferior direito da tela.

### 3. Modelos de Projeto

#### 3.1 Arquitetura

No framework Laravel, adotado para desenvolvimento da aplicação, a arquitetura em camadas é organizada de forma a promover uma estrutura modular e escalável para o desenvolvimento de aplicativos da web, facilitando a manutenção e teste de código. Essa arquitetura consiste em várias camadas interconectadas:

A camada de Views é responsável pela apresentação da interface do usuário e é onde o conteúdo é renderizado para o navegador. As views podem conter HTML, CSS, JavaScript e elementos dinâmicos incorporados usando a sintaxe Blade do Laravel. As Routes definem as URLs do aplicativo e direcionam essas URLs para os respectivos controladores. As rotas podem ser definidas de forma explícita no arquivo de rotas ou usando convenções de nomenclatura RESTful.

Os Middlewares são camadas intermediárias que podem ser aplicadas a rotas individuais ou a grupos de rotas. Eles fornecem uma maneira de filtrar as solicitações HTTP antes que elas alcancem os controladores, permitindo a execução de ações como autenticação, autorização e manipulação de requisições. Os Controllers são responsáveis por receber as requisições do cliente, processá-las e



retornar uma resposta apropriada. Eles agem como intermediários entre as rotas e os serviços de negócios, coordenando a lógica de aplicativo e manipulando os dados recebidos das requisições.

Os Services são classes que encapsulam a lógica de negócios do aplicativo. Eles fornecem uma interface limpa para acessar e manipular os dados do aplicativo, promovendo a reutilização e a manutenção do código. Os Models representam e gerenciam os dados do aplicativo. Eles mapeiam diretamente para as tabelas do banco de dados e fornecem métodos para realizar operações CRUD (Create, Read, Update, Delete) nos dados.

A Figura 3 a seguir ilustra as informações supra apresentadas.

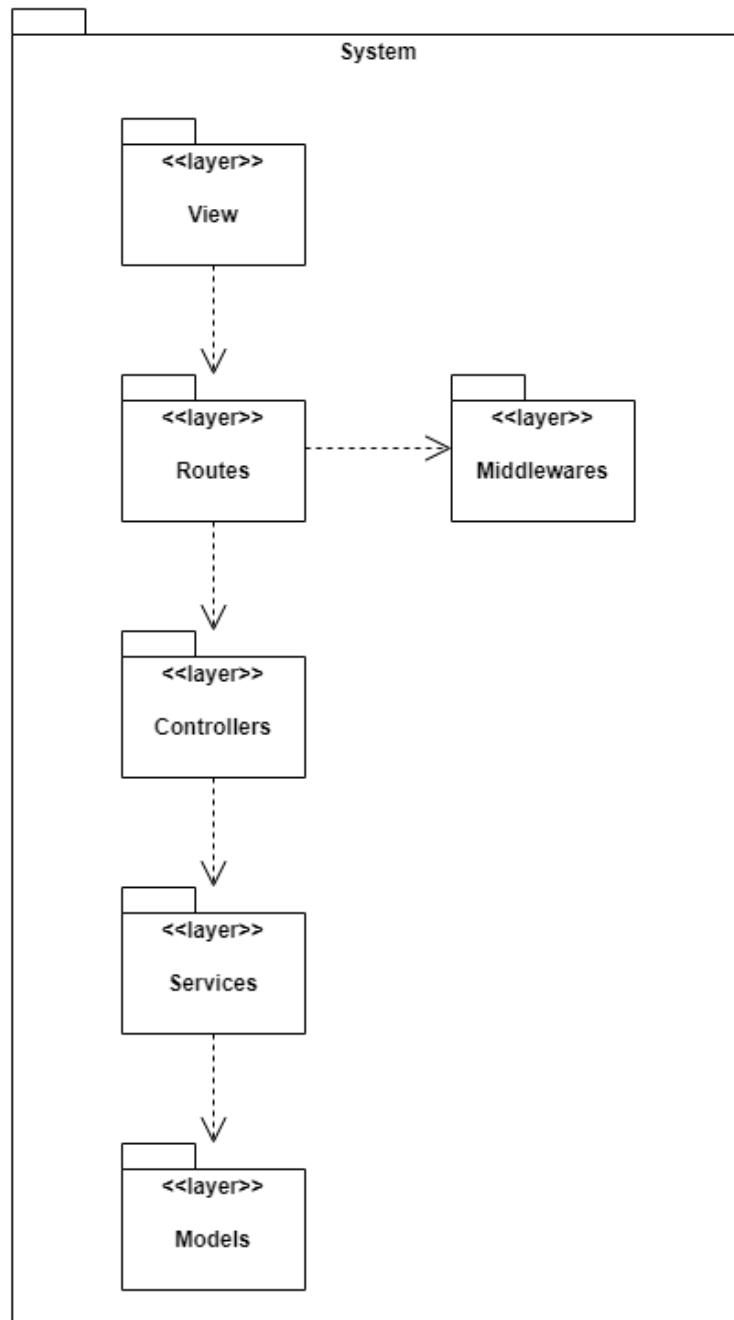


Figura 3 - Arquitetura do sistema

### 3.2 Diagrama de Componentes e Implantação

Serão apresentados nesta seção os diagramas de componente e implantação do projeto.

O primeiro deles, o diagrama de componentes (Figura 4), indica os principais componentes do sistema, e como eles são relacionados em termos de dependência um do outro, além das interfaces que utilizam e disponibilizam. O diagrama de implantação (Figura 5), por sua vez, representa os recursos físicos e de software que são necessários para que o sistema seja implementado corretamente.

Conforme pode ser observado na Figura 4, o principal componente do diagrama de componentes é o servidor web, que vai se relacionar diretamente com diversos outros componentes, como *controllers*, *services*, *models* e *configuration*. Portanto, é nesse componente que estará a comunicação com o banco de dados e disponibilização de API externa para a aplicação web.

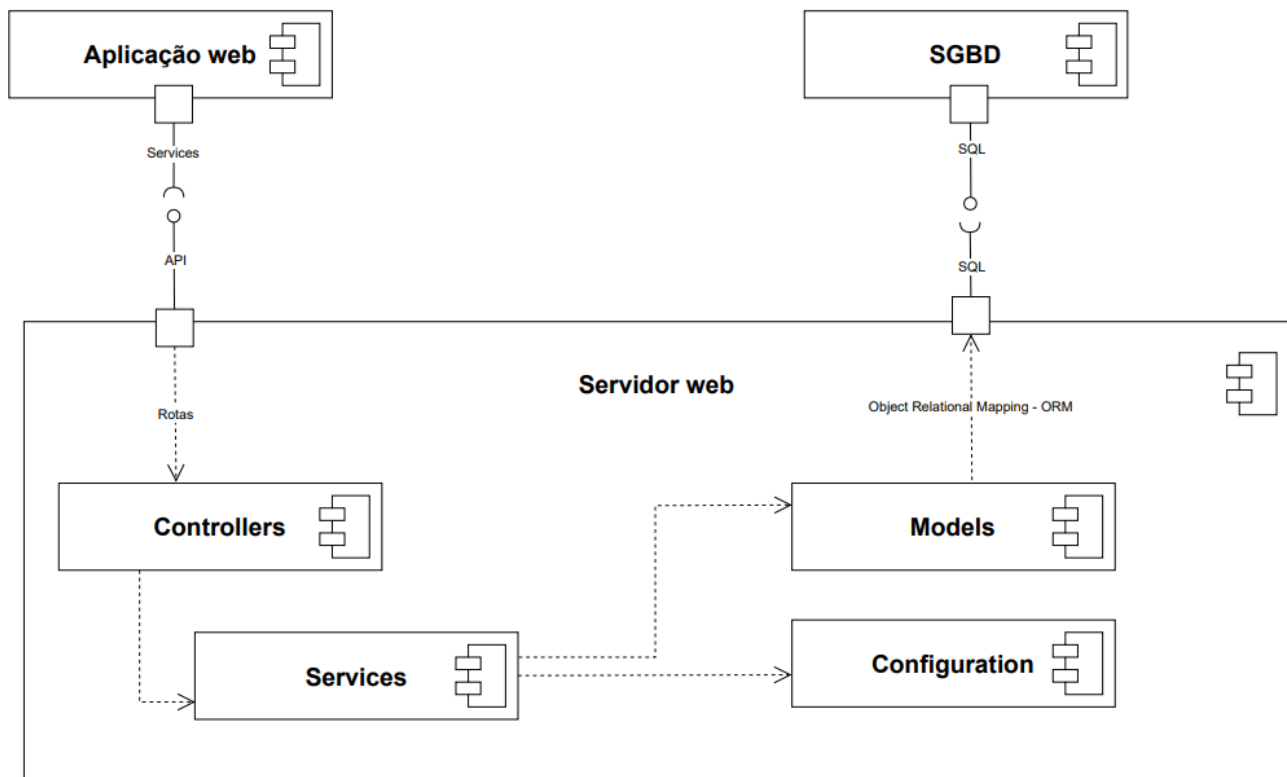


Figura 4 - Diagrama de componentes

Na figura a seguir (Figura 5), é apresentado o diagrama de implantação do sistema. Neste diagrama são apresentados os pontos de processamento do projeto, e onde está sendo utilizado. Conforme ilustrado nessa figura, o navegador web (acessado diretamente pelo usuário) faz requisições ao nosso *back-end*, que é encapsulado por uma máquina virtual em um serviço de nuvem, por meio de protocolos HTTPs. O *back-end*, por sua vez, está relacionando com o banco de dados, fazendo as atualizações / modificações necessárias de acordo com as requisições recebidas.

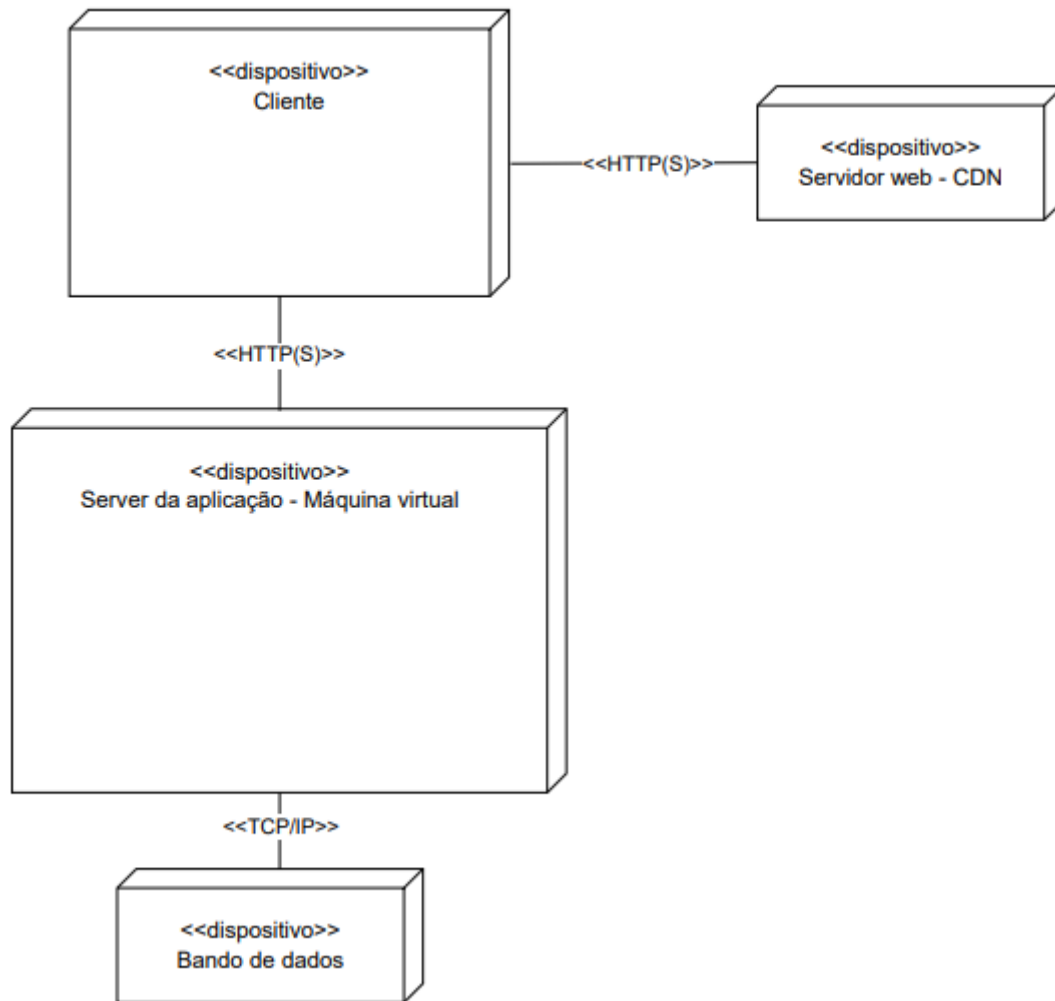


Figura 5 - Diagrama de implantação

### 3.3 Diagrama de Classes

A seguir é apresentado o diagrama de classes, elencando as principais classes do sistema. Além das classes apresentadas, existem classes de apoio que foram ocultadas para fins didáticos, como a classe ‘produto\_ingredientes’, por exemplo, que relaciona os ingredientes em um produto.



*Figura 6 - Diagrama de classes do sistema*

### 3.4 Diagramas de Sequência

Nesta seção, são apresentados diagramas de sequência referentes aos casos de uso do sistema implementado. Esses diagramas demonstram o fluxo entre os componentes para cada cenário de uso, dessa forma, eles apresentam como funcionam os fluxos de chamadas entre as entidades que participam de uma interação.

A Figura 7 representa o diagrama de sequência responsável pelo fluxo de agendamento de pedidos. Nesse fluxo, o usuário Cliente envia mensagens síncronas ao sistema com o objetivo de listar os produtos à venda, adicionar ou excluir produtos do carrinho de compras, adicionar informações de comprador e de entrega e confirmar o pedido. A partir disso, as mensagens chegam aos componentes de execução responsáveis, `ProdutosController` ou `PedidosController`, que tratam os dados da requisição e enviam para os componentes de execução conforme responsabilidade, podendo ser o `ProdutosService`, `PedidosService`, `ClientesService` ou `EnderecosService`, que manipulam as respectivas entidades, `Produtos`, `Pedidos`, `Clientes` ou `Enderecos`, dentro do sistema conforme o objetivo requerido pelo usuário. Esse diagrama contempla o caso de uso UC01.

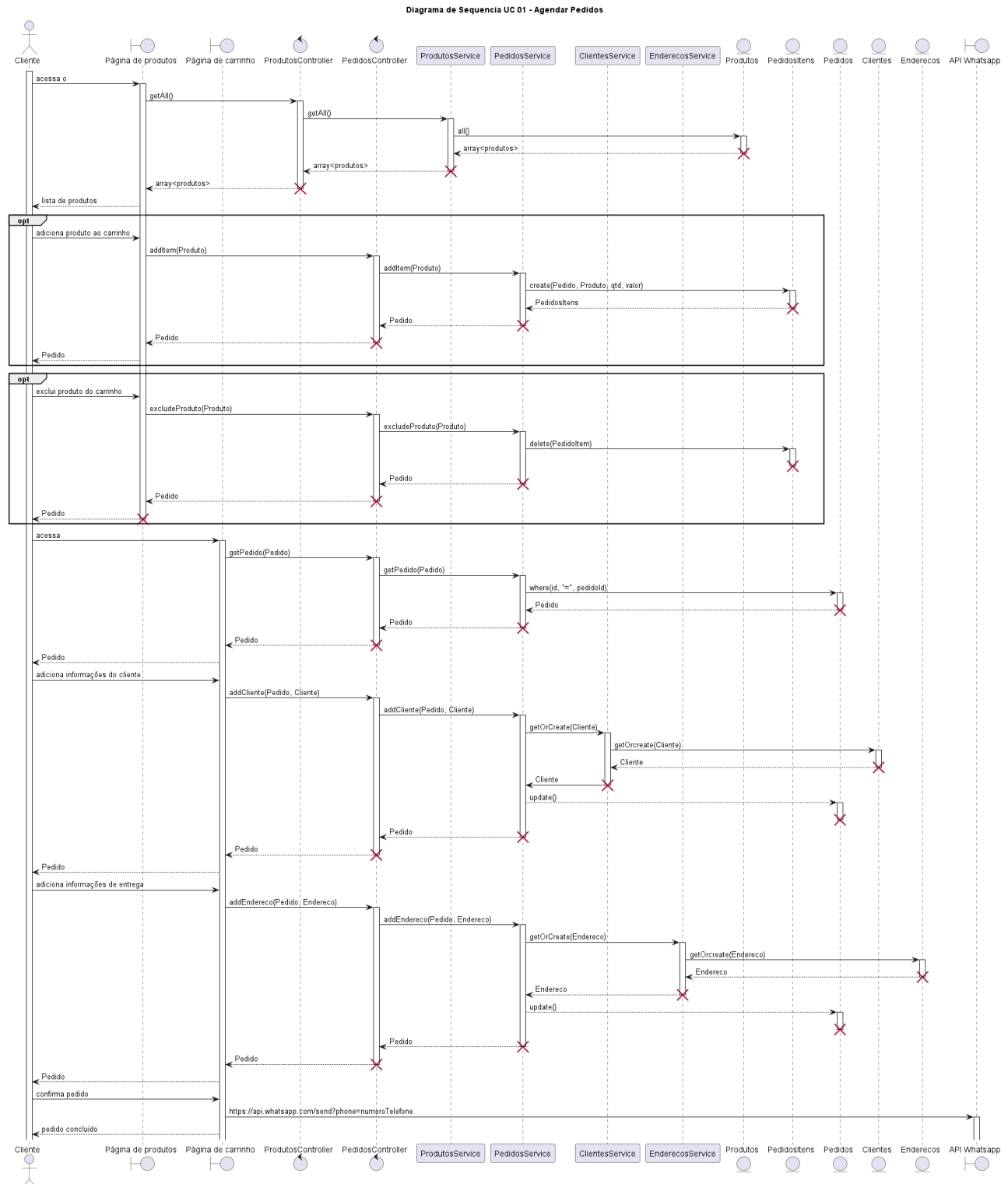
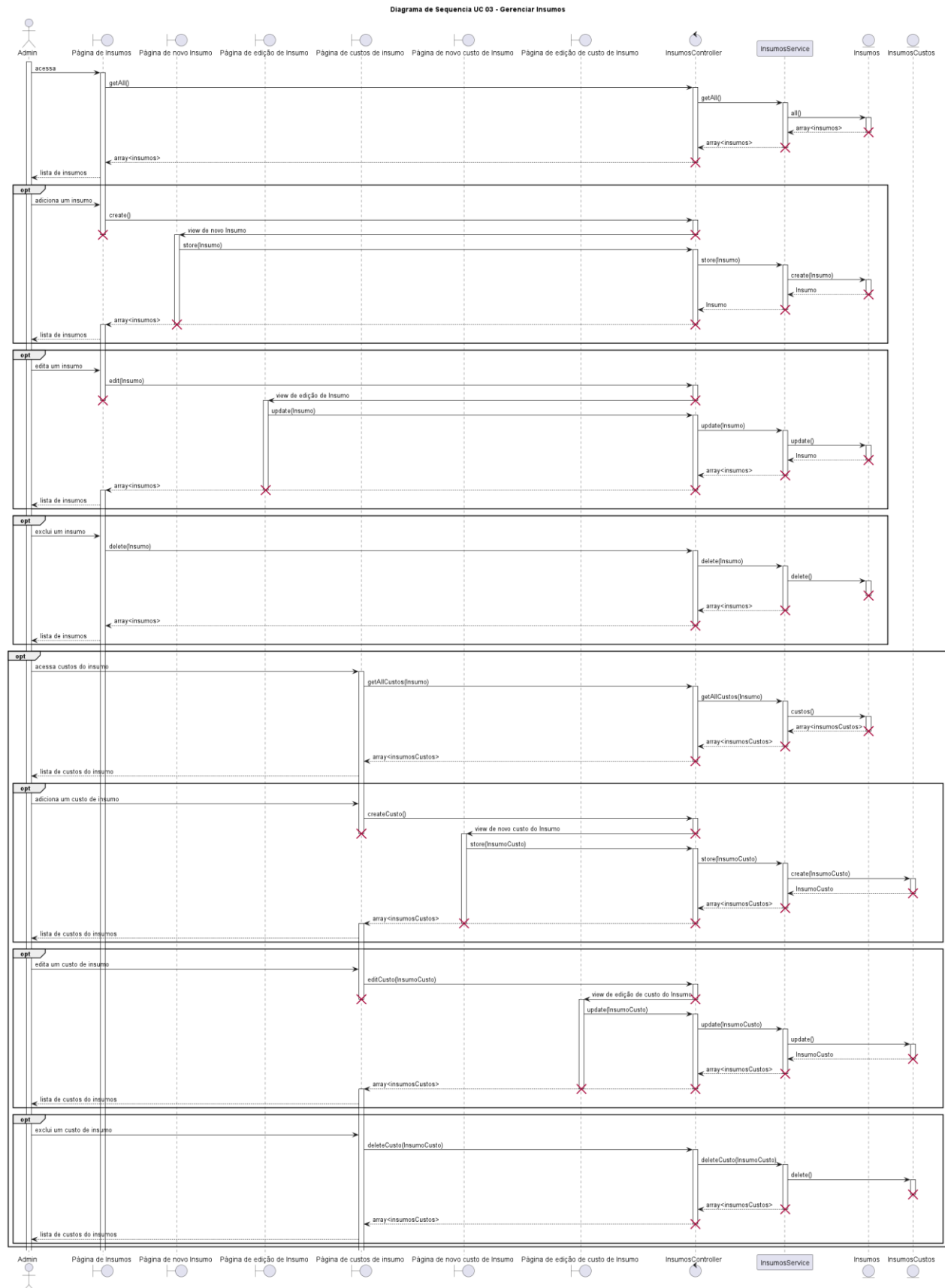


Figura 7 – DS1 – Agendar pedidos

A Figura 8 representa o diagrama de sequência responsável pelo fluxo de gerenciamento de insumos. Nesse fluxo, o usuário Admin envia mensagens síncronas ao sistema com o objetivo de listar, adicionar, editar ou excluir insumos. É possível, ainda, requisitar a listagem, adição, edição ou exclusão de custos dos insumos. A partir disso, as mensagens chegam ao componente de execução `InsumosController`, que trata os dados da requisição e envia para o componente de execução `InsumosService`, que manipula as entidades `Insumos` ou `InsumosCustos` dentro do sistema conforme o objetivo requerido pelo usuário. Esse diagrama contempla o caso de uso UC03.





*Figura 8 – DS2 – Gerenciamento de insumos*

A Figura 9 representa o diagrama de sequência responsável pelo fluxo de gerenciamento de receitas. Nesse fluxo, o usuário Admin envia mensagens síncronas ao sistema com o objetivo de listar, adicionar, editar ou excluir receitas. A partir disso, as mensagens chegam ao componente de execução `ReceitasController`, que trata os dados da requisição e envia para o componente de execução `ReceitasService`, que manipula a entidade `Receitas` dentro do sistema conforme o objetivo requerido pelo usuário. Esse diagrama contempla o caso de uso UC06.

Diagrama de Sequência UC 06 - Gerenciar Receitas

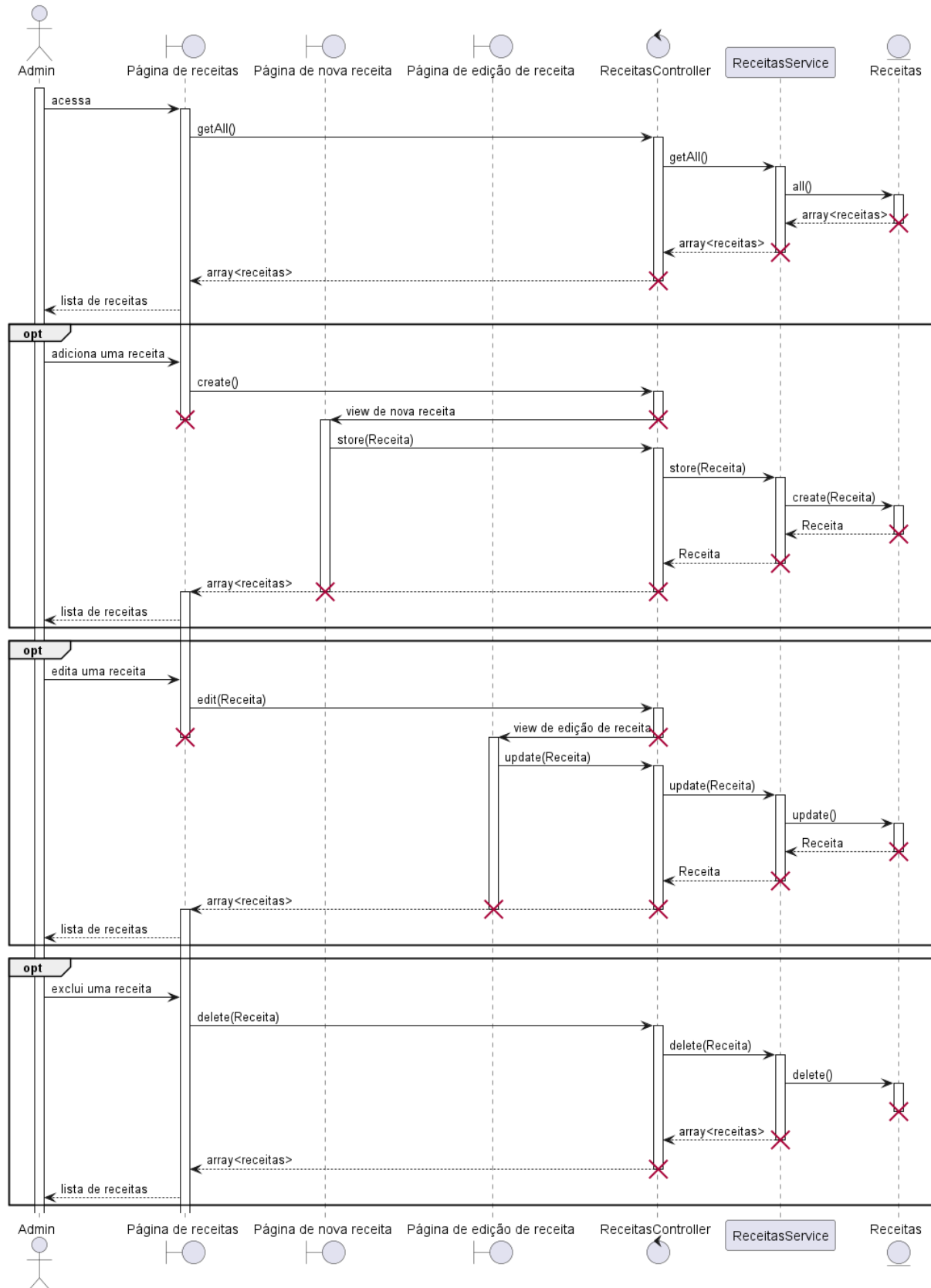


Figura 9 – DS3 – Gerenciamento de receitas

### 3.5 Diagramas de Comunicação

Para ilustrar o diagrama de comunicação do sistema, foi utilizado como referência o caso de uso UC-06: Gerenciar receitas. Conforme ilustrado no diagrama abaixo, o processo se inicia na camada de serviço de receitas, e, ao passo que avança em sua execução, segue para a classe de serviço de ingredientes.

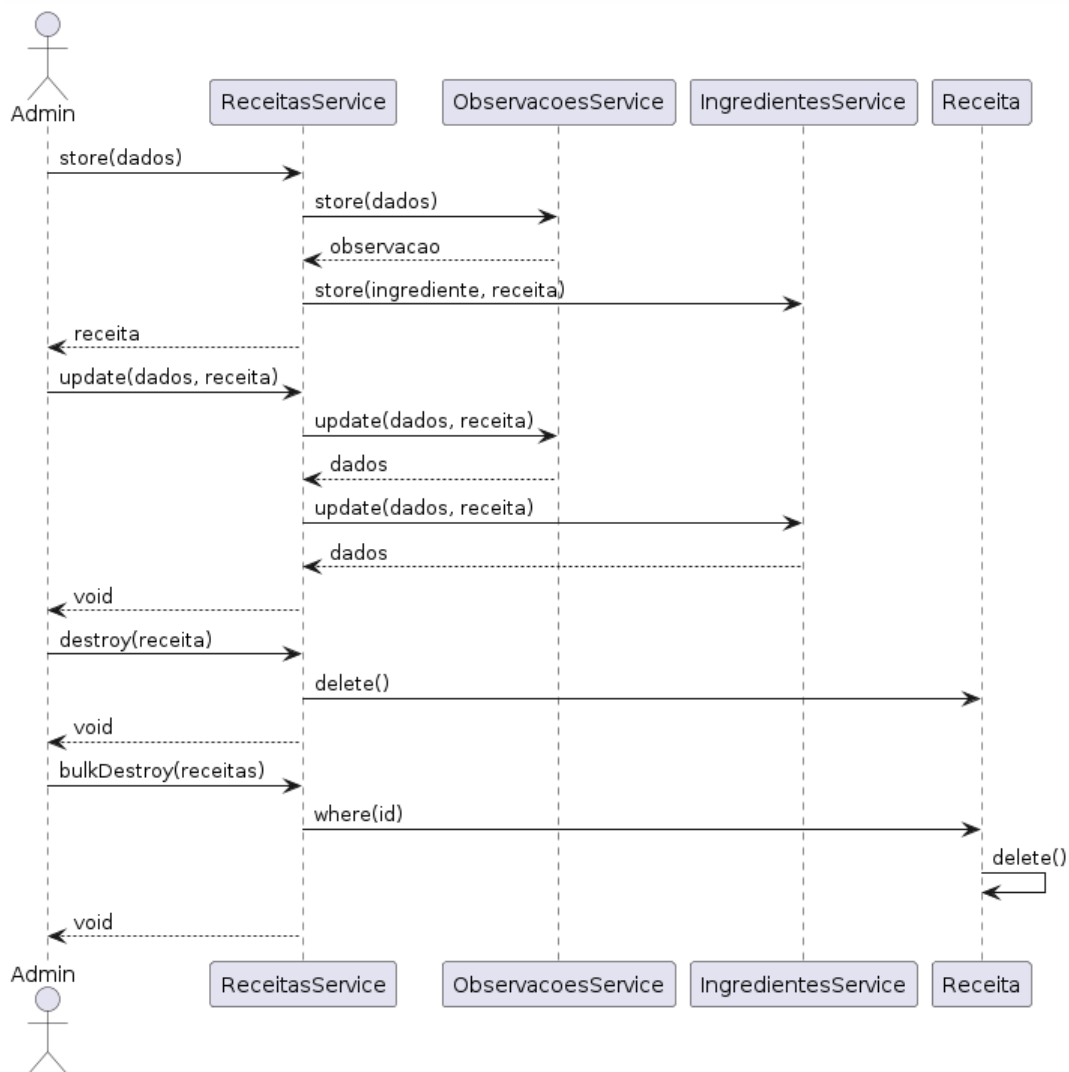


Figura 10 - Diagrama de comunicação

### 3.6 Diagramas de Estados

Para elaboração do diagrama de estado do sistema, foram utilizados todos os casos de uso descritos nesse documento. Após a ocorrência de algum evento, o diagrama de evento mostra qual é o estado (caso de uso) que entra em vigência a partir daquele ponto.

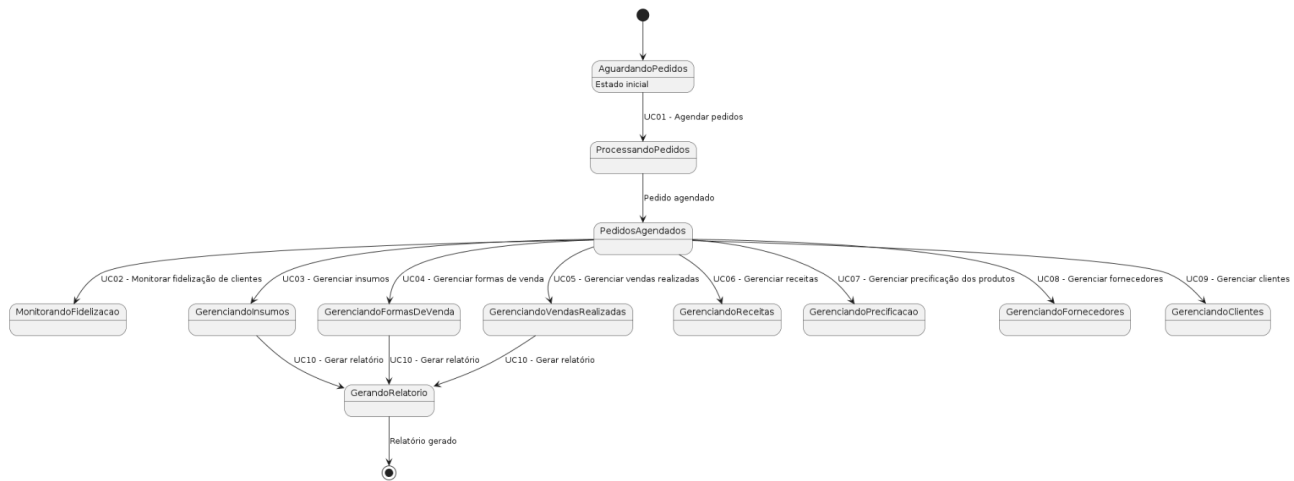





Figura 11 - Diagrama de estado

## 4. Modelos de Dados

O mapeamento dos objetos no banco de dados foi realizado por meio migrations. As migrations são classes responsáveis por executar scripts de banco de dados, inserindo os atributos e relacionamentos entre cada uma das tabelas criadas. A seguir é apresentado um exemplo de migration, responsável pelo armazenamento dos logs de serviços (Jobs) que apresentaram falhas durante a execução.

```
1  <?php
2  
3  use Illuminate\Database\Migrations\Migration;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Support\Facades\Schema;
6
7  return new class extends Migration
8  {
9      /**
10       * Run the migrations.
11       */
12     public function up(): void  lannarelli
13     {
14         Schema::create( table: 'failed_jobs', function (Blueprint $table) {
15             $table->id();
16             $table->string( column: 'uuid')->unique();
17             $table->text( column: 'connection');
18             $table->text( column: 'queue');
19             $table->longText( column: 'payload');
20             $table->longText( column: 'exception');
21             $table->timestamp( column: 'failed_at')->useCurrent();
22         });
23     }
24
25     /**
26      * Reverse the migrations.
27      */
28     public function down(): void  lannarelli
29     {
30         Schema::dropIfExists( table: 'failed_jobs');
31     }
32 };
33
```

*Figura 12 - Migration - failed jobs*