Ciência da Computação - UFF.
Exercício de Projeto de Software.
Alunos:
Alonso Breda
Luisa Stellet
Miguel Jehle
Nicolle Meireles
Rafael Accetta.

1) Descrição do escopo do sistema:

O sistema do site Atelier Flor de Macela é uma plataforma de e-commerce voltada para a comercialização de produtos artesanais, naturais e veganos voltados ao bem-estar, como sabonetes, cosméticos, óleos e itens de aromaterapia. Com uma navegação intuitiva, o site oferece funcionalidades como catálogo de produtos por categorias, carrinho de compras, cálculo de frete, opções de pagamento online (Pix, cartão e boleto), além de cadastro de usuários.

Será uma loja virtual especializada na venda de cosméticos veganos. Ele permitirá que clientes naveguem por produtos, realizem compras online, utilizem cupons de desconto, calculem frete e recebam recomendações personalizadas. Administradores poderão gerenciar vendas e visualizar relatórios estratégicos.

2) Descrição dos requisitos arquiteturais, objetivos e restrições da arquitetura:

Objetivos da Arquitetura

- **Desempenho**: Garantir tempos de resposta rápidos, mantendo baixa latência e eficiência na comunicação entre os serviços.
- **Escalabilidade**: Permitir que o sistema suporte um número crescente de usuários, pedidos e produtos sem perda de desempenho.
- **Segurança**: Assegurar proteção contra ameaças externas, implementação de autenticação robusta e conformidade com a LGPD.
- **Disponibilidade**: Garantir alta disponibilidade do sistema, evitando downtime e falhas críticas.
- **Manutenibilidade**: Implementar um design modular que facilite atualizações, correções de erros e aprimoramentos contínuos.

- **Usabilidade**: Fornecer uma interface intuitiva que ofereça uma experiência fluida para usuários finais e administradores.
- **Portabilidade**: Assegurar compatibilidade entre diferentes dispositivos e navegadores sem perda de funcionalidade.
- **Confiabilidade**: Garantir que o sistema funcione corretamente sob diferentes condições e minimize falhas operacionais.
- **Testabilidade**: Desenvolver componentes que possam ser facilmente testados e validados para garantir qualidade contínua.

Requisitos Arquiteturais

- Interface construída com **React** para navegação responsiva e intuitiva.
- API desenvolvida com **Java (Spring Boot)**, utilizando **JPA** para mapear entidades e viabilizar uma comunicação eficiente entre o front-end e o banco de dados.
- Uso de **PostgreSQL**, conforme necessidade de armazenamento de dados.
- Implementação de **OAuth** ou **JWT** para segurança no login.
- Integração com API de pagamento para transações financeiras seguras.
- Hospedagem em serviços de nuvem como **AWS, Azure ou GCP** para escalabilidade e disponibilidade.

Restrições da Arquitetura

- **Dependência de terceiros**: APIs de pagamento devem garantir conformidade com LGPD.
- **Tempo de resposta**: O site deve carregar em menos de 3 segundos para evitar altas taxas de desistência.
- **Compatibilidade**: Suporte total a dispositivos móveis e diferentes navegadores.
- **Custos**: A escolha de tecnologias deve considerar o orçamento disponível para desenvolvimento e manutenção.
- **Manutenção e Suporte**: A infraestrutura deve permitir atualizações e correções sem interrupções prolongadas.

3) Definição dos padrões arquiteturais adotados:

Para garantir modularidade, organização e eficiência na comunicação entre os componentes do sistema, foram adotados os seguintes padrões arquiteturais:

- **Arquitetura em Camadas**: O sistema será estruturado em múltiplas camadas para garantir separação de responsabilidades.
- **Modelo MVC (Model-View-Controller)**: Organização do código para garantir que as responsabilidades do sistema sejam bem definidas:
  - **Model**: Representa a estrutura dos dados e interage com a camada de persistência.
  - **View**: Define a apresentação visual e interações com o usuário.
  - **Controller**: Atua como intermediário entre a View e o Model, processando requisições e respostas.
- **Cliente-Servidor**: Arquitetura baseada na separação entre os componentes cliente e servidor, garantindo escalabilidade e organização.

4) Justificativa das decisões arquiteturais adotadas com base nos atributos de qualidade do sistema. Descrição de como a arquitetura de software contribui para os atributos de qualidade importantes do sistema:

Para garantir um sistema modular, escalável e eficiente, escolhemos os seguintes padrões arquiteturais: Arquitetura em Camadas, Modelo MVC e Cliente-Servidor.

- **Arquitetura em Camadas**: Esse padrão foi adotado para promover a separação de responsabilidades, garantindo manutenibilidade e escalabilidade. Com camadas bem definidas, o sistema pode ser atualizado e expandido sem comprometer suas funcionalidades essenciais. Além disso, essa abordagem melhora o desempenho e a confiabilidade, pois cada camada é otimizada para sua função específica, reduzindo acoplamento e aumentando a coesão entre componentes.
- **Modelo MVC (Model-View-Controller)**: O uso de MVC facilita a organização do código, permitindo que a interface (View), a lógica de controle (Controller) e a estrutura de dados (Model) sejam independentes e bem estruturadas. Isso melhora a usabilidade e testabilidade do sistema, pois alterações na interface do usuário podem ser feitas sem impactar diretamente a lógica de negócios. Além disso, a confiabilidade e a segurança são reforçadas, pois a separação clara de responsabilidades reduz erros e vulnerabilidades.
- **Cliente-Servidor**: Esse modelo permite uma distribuição eficiente dos serviços, garantindo disponibilidade e desempenho. Com um servidor dedicado ao processamento de requisições e um cliente responsável pela interface e interatividade, o sistema é mais escalável. Esse padrão também melhora a portabilidade e compatibilidade, pois o cliente pode acessar os serviços por meio de diferentes dispositivos sem necessidade de alterações estruturais na arquitetura.

5) Documentação das visões arquiteturais:

Ao menos diagramas de casos de usos, modelo conceitual e diagramas de sequência do sistema:

- Diagrama de caso de uso

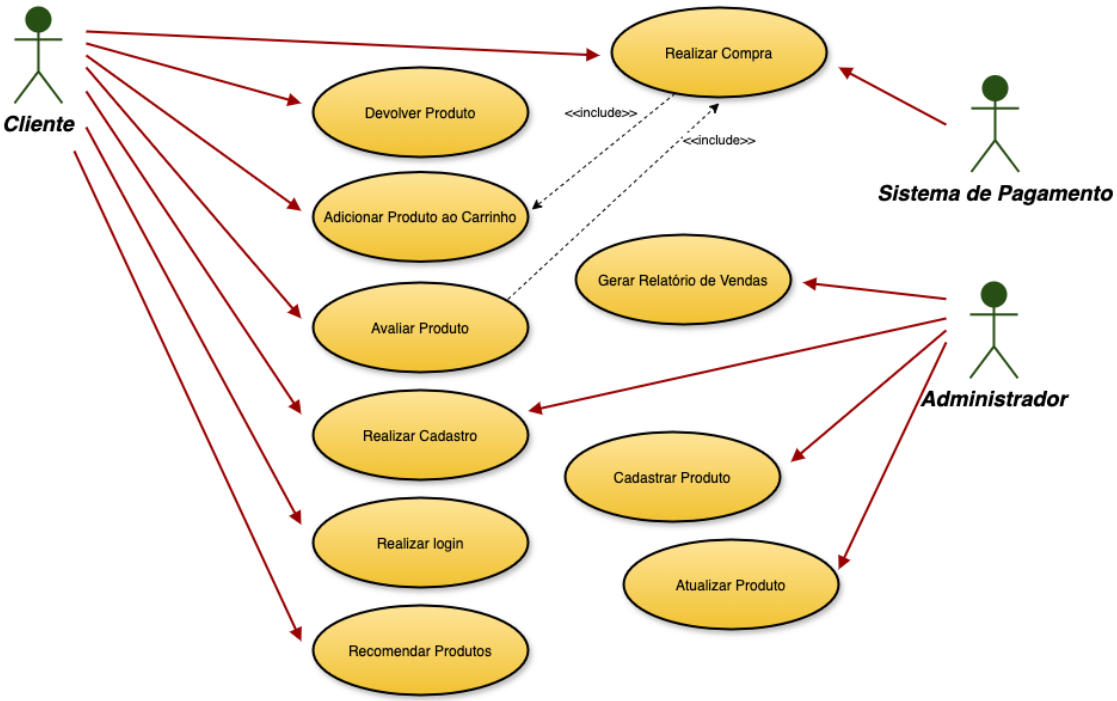

Figure 1: Diagrama de caso de uso.

- Modelo conceitual

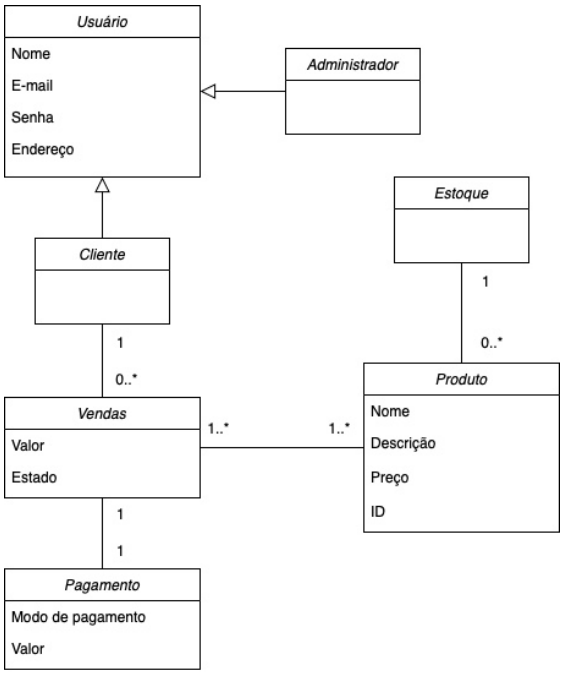

Figure 2: Modelo conceitual.

- DSS - Realizar Compra

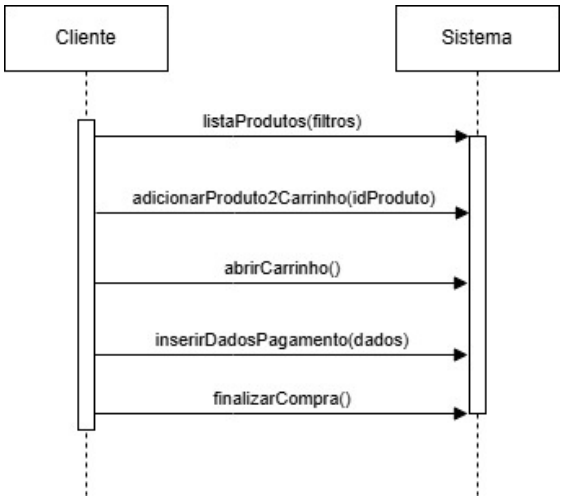

Figure 3: DSS - Realizar Compra.

- DSS - Devolver Produto

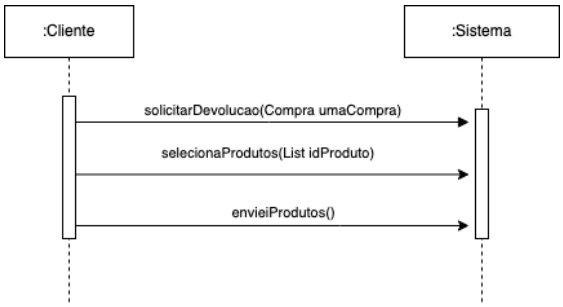

Figure 4: DSS - Devolver Produto.

- DSS - Realizar Cadastro

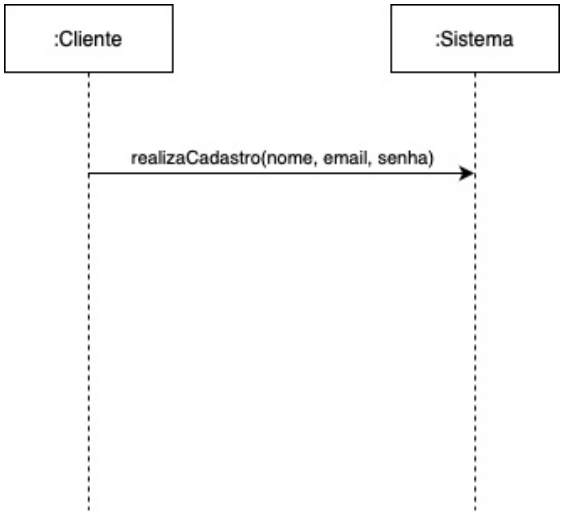

Figure 5: DSS - Realizar Cadastro.

- DSS - Recomendar Produto

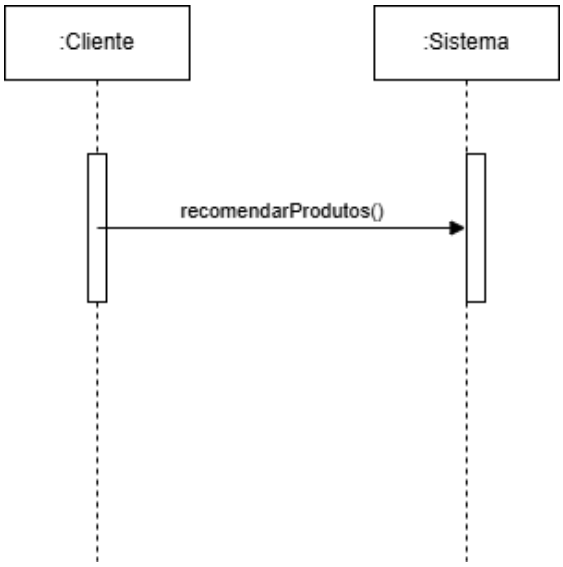

Figure 6: DSS - Recomendar Produto.

- DSS - Avaliar Produto

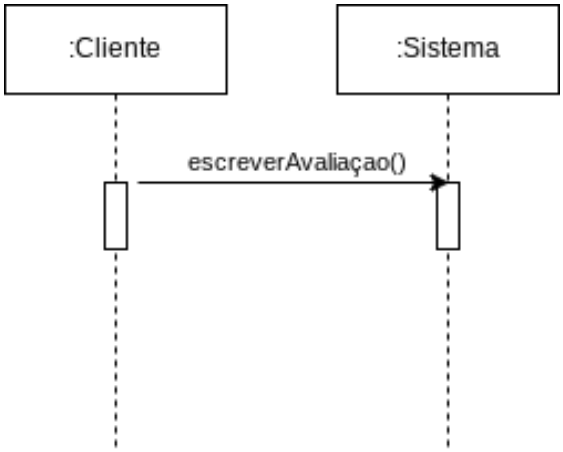

Figure 7: DSS - Avaliar Produto.

- DSS - Gerar Relatório

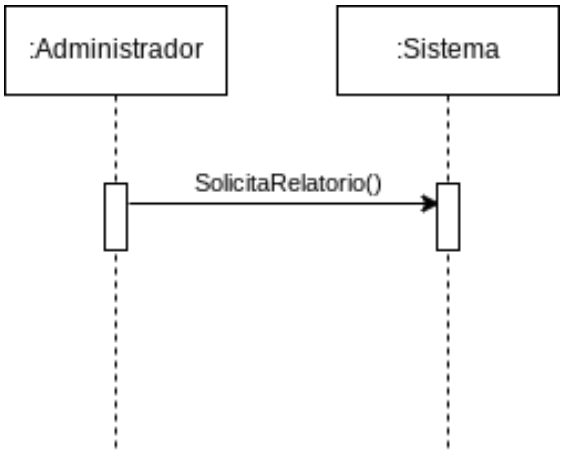

Figure 8: DSS - Gerar Relatório.

- Diagrama com visão geral de arquitetura do sistema e suas justificativas.

Arquitetura em Camadas
O sistema foi estruturado em camadas distintas (frontend, backend e banco de dados) para garantir uma separação clara de responsabilidades. Essa abordagem facilita a manutenção, permite a evolução modular do projeto e isola possíveis falhas, assegurando que mudanças em uma camada não afetem as outras desnecessariamente. Além disso, essa organização promove um desenvolvimento mais organizado e escalável.

Modelo MVC (Model-View-Controller)
A aplicação adota o padrão MVC para definir responsabilidades específicas em cada componente. O frontend (React) atua como a View, responsável pela interface do usuário, enquanto o backend (Spring Boot) implementa os Controllers, que gerenciam as requisições e respostas. O Model, representado pelas entidades e DAOs, lida com a estrutura dos dados e a comunicação com o banco. Essa divisão melhora a reusabilidade do código e simplifica a colaboração entre equipes.

Cliente-Servidor
A arquitetura cliente-servidor foi escolhida para separar claramente o frontend (cliente) do backend (servidor), permitindo que cada parte seja desenvolvida e escalada independentemente. O cliente consome APIs fornecidas pelo servidor, garantindo flexibilidade para futuras integrações ou migrações de hospedagem (como AWS, Azure ou GCP). Essa abordagem também favorece a segurança e o desempenho, distribuindo adequadamente as cargas de processamento.

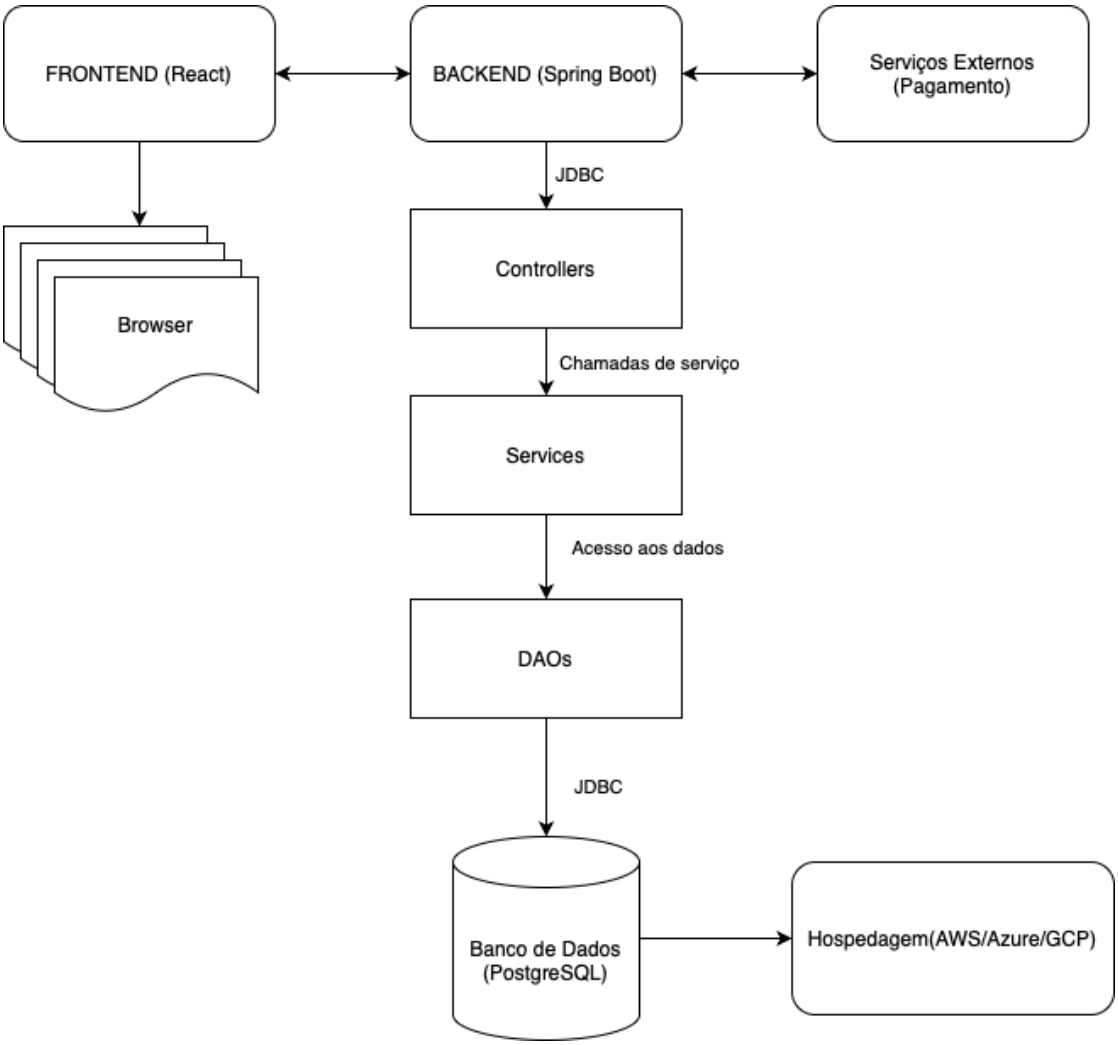

Figure 9: Diagrama com visão geral de arquitetura.

6) Diagramas de interação. Pelo menos um diagrama por caso de uso. Escolher a operação do DSS mais complexa. Obs: Descrever o contrato da operação escolhida. (Mínimo 1 diagrama por integrante da equipe):

- Realizar compra

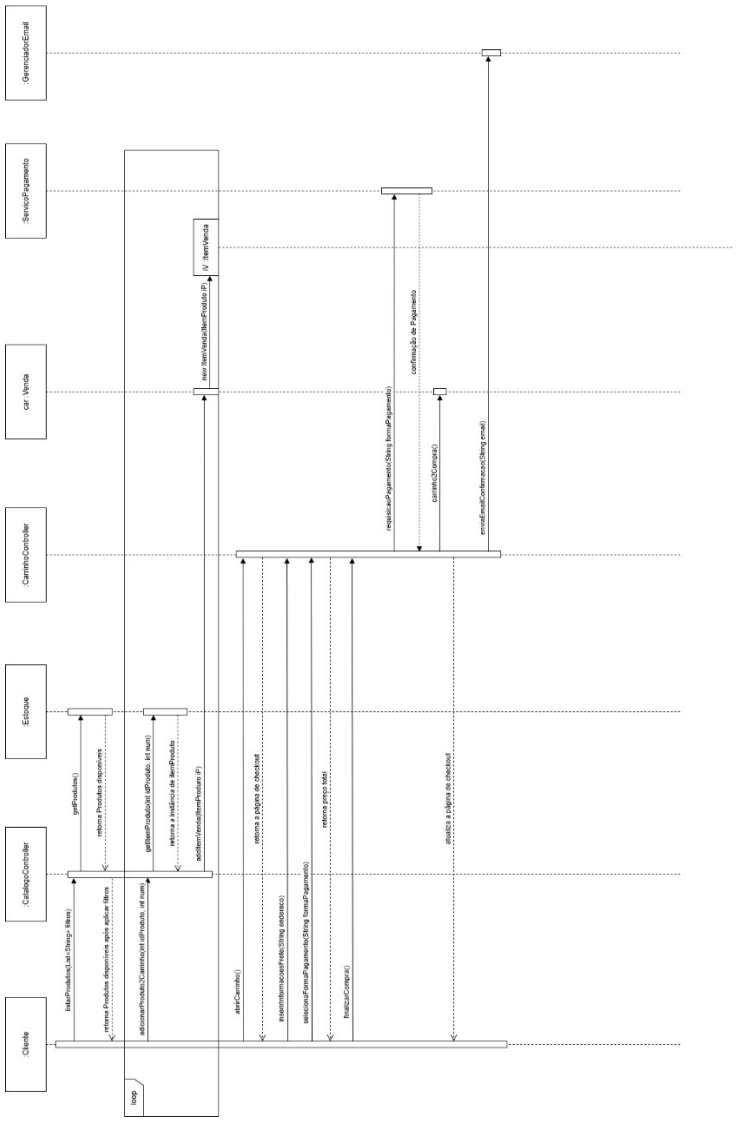

Figure 10: Diagrama de Sequência de Realizar Compra.

- Devolver Produto

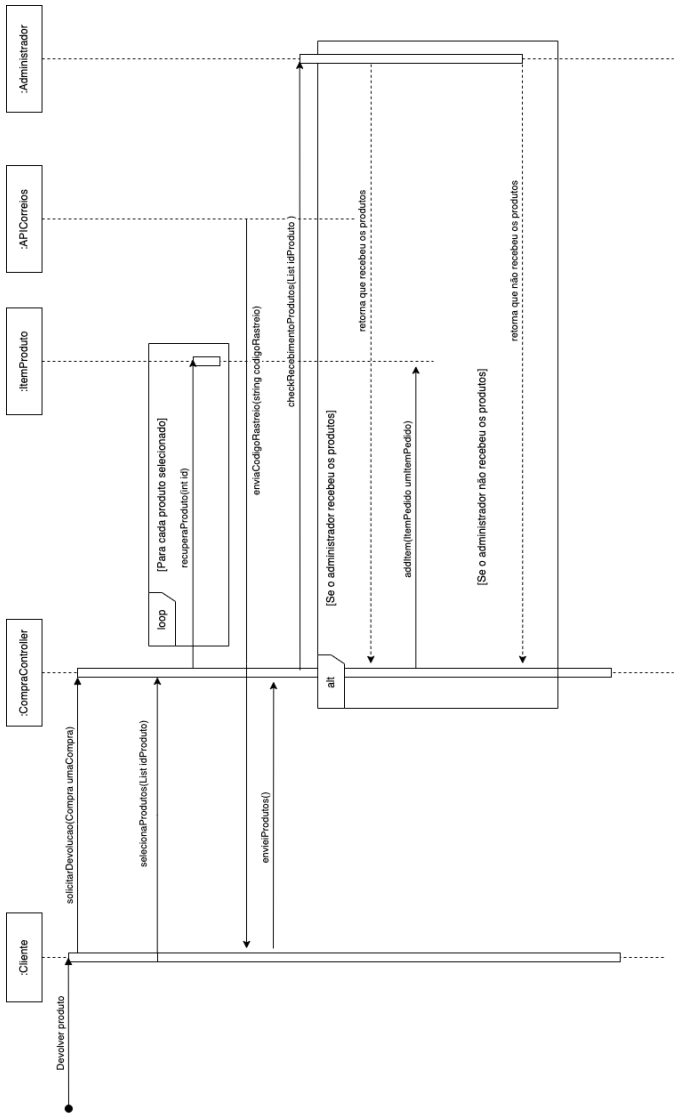

Figure 11: Diagrama de Sequência de Devolver Produto.

- Realizar Cadastro

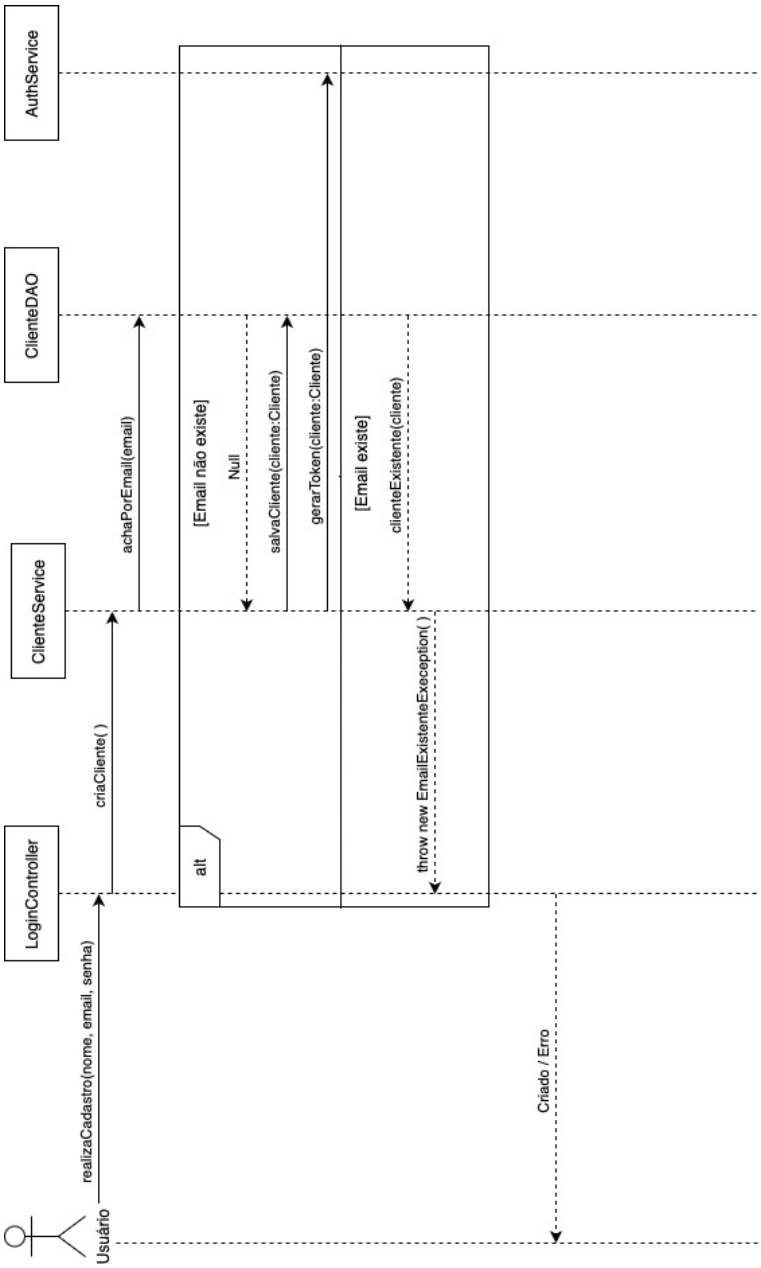

Figure 12: Diagrama de Sequência de Realizar Cadastro.

- Avaliar Produto

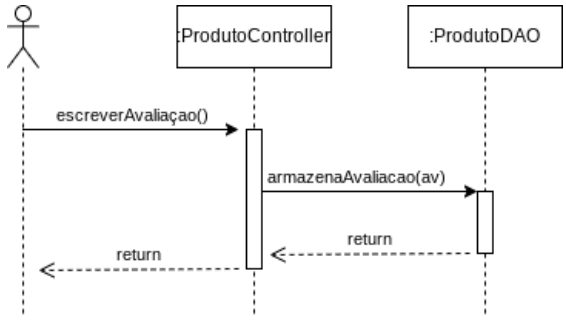

Figure 13: Diagrama de Sequência de Avaliar Produto.

- Gerar Relatório

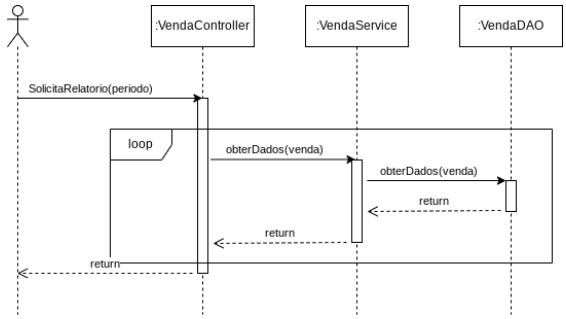

Figure 14: Diagrama de Sequência de Gerar Relatório.

- Recomendar Produto.

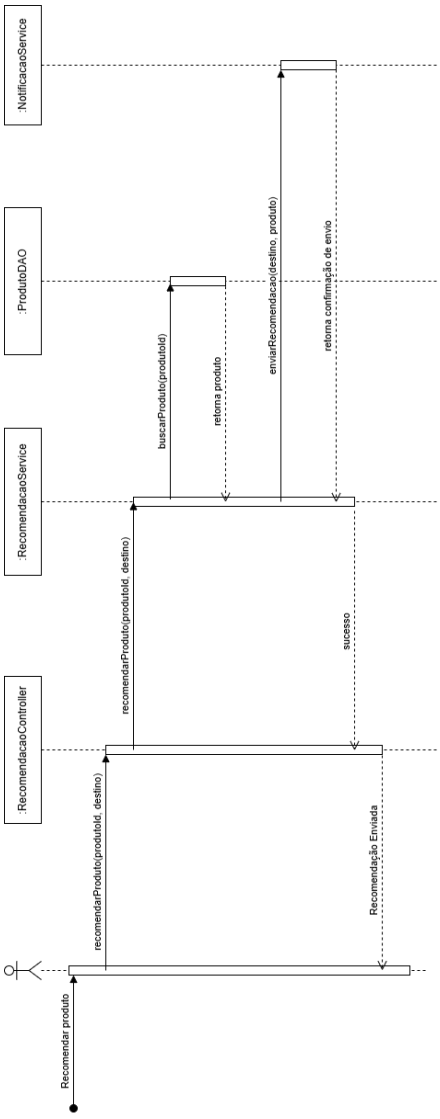

Figure 15: Diagrama de Sequência de Recomendar Produto.

7) Diagramas de estado ou atividades:

Resposta: Não vimos necessidade.

8) Diagramas de classe detalhado:

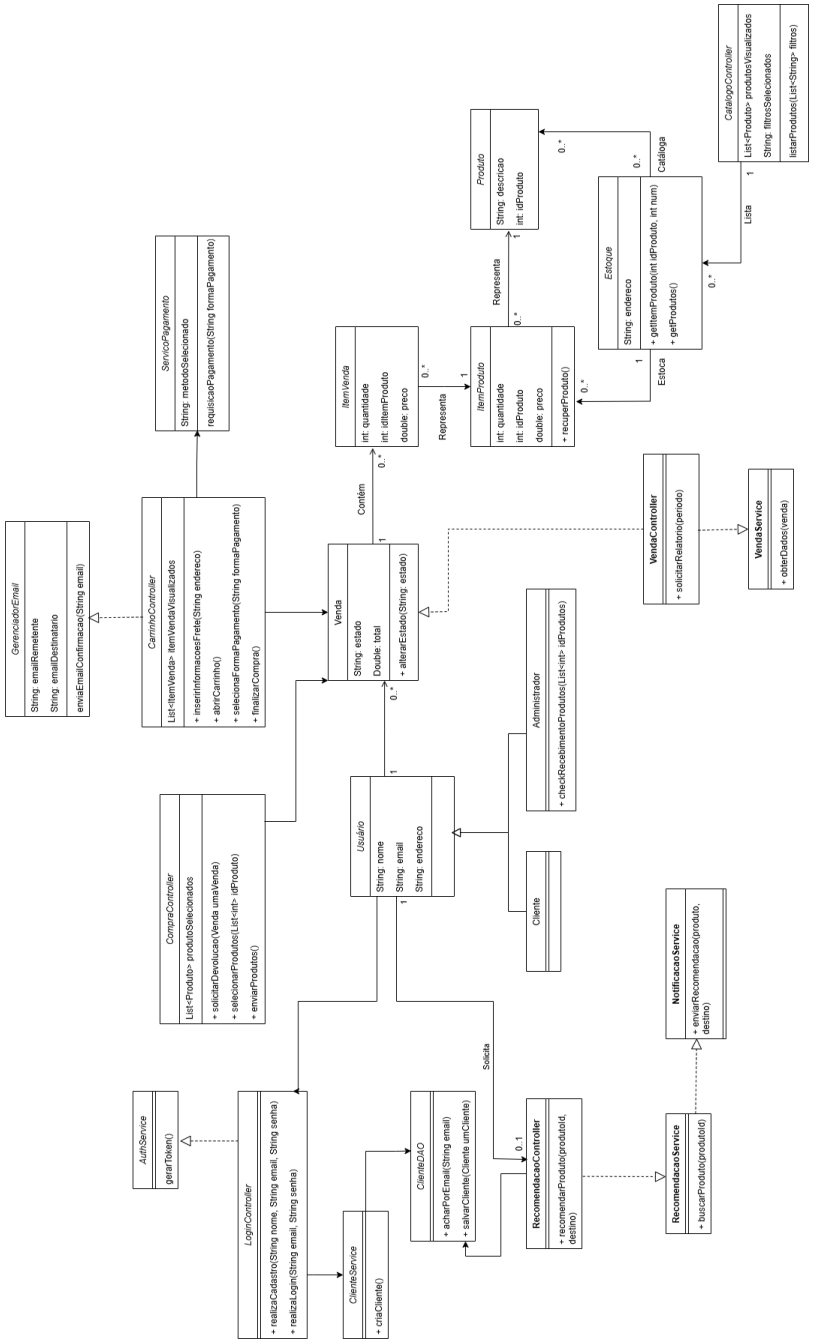

Figure 16: Enter Caption