**Análise de Requisitos**

**Requisitos Funcionais**

1. **Cadastro e Gerenciamento de Usuários:**
   * Permitir o registro de usuários com autenticação segura (e.g., email e senha ou login social).
   * Gerenciamento de perfis, incluindo a edição de dados pessoais e preferências.
   * Suporte para diferentes tipos de usuários (clientes, administradores e vendedores).
2. **Catálogo de Produtos:**
   * Exibir um catálogo de produtos com categorias específicas para itens sustentáveis (recicláveis, orgânicos etc.).
   * Filtros avançados (por tipo de produto, preço, impacto ambiental etc.).
   * Sistema de busca com sugestões inteligentes.
3. **Carrinho de Compras e Pagamentos:**
   * Função de adicionar, editar e remover produtos do carrinho.
   * Processamento de pagamentos com integração a gateways seguros.
   * Suporte a métodos de pagamento como cartões de crédito, débito, Pix e carteiras digitais.
4. **Gestão de Pedidos:**
   * Histórico de pedidos para os usuários.

Rastreamento de pedidos com atualização de status.

* + Sistema de notificações (e.g., e-mail ou push) para atualizações de pedidos.

1. **Sistema de Avaliação e Feedback:**
   * Avaliações de produtos por parte dos clientes.
   * Comentários e classificações para ajudar outros consumidores.
2. **Promoções e Sustentabilidade:**
   * Configuração de descontos e promoções específicas.
   * Informações sobre o impacto ambiental de cada produto.
3. **Área do Vendedor:**
   * Permitir que vendedores gerenciem seus produtos, preços e estoques.
   * Relatórios sobre vendas e desempenho.
4. **Integrações com Terceiros:**
   * Integração com serviços de logística para cálculo de frete e rastreamento.
   * Conexão com plataformas de marketing e redes sociais.

**Requisitos Não Funcionais**

1. **Escalabilidade:**
   * Arquitetura baseada em microsserviços para suportar crescimento rápido.
   * Capacidade de lidar com picos de acessos e transações simultâneas.
2. **Desempenho:**
   * Tempo de resposta inferior a 2 segundos para ações críticas.
   * Sistema otimizado para alta disponibilidade (99,9% de uptime).
3. **Segurança:**
   * Criptografia de ponta a ponta para dados sensíveis (e.g., HTTPS e dados de pagamento).
   * Implementação de autenticação multifator (MFA).
   * Conformidade com LGPD e GDPR para proteção de dados.
4. **Documentação:**
   * Documentação clara e atualizada para facilitar a entrada de novos desenvolvedores.
   * Uso de ferramentas como Swagger/OpenAPI para documentar APIs.
5. **Manutenibilidade:**
   * Estrutura modular para facilitar a adição de novas funcionalidades.
   * Testes automatizados para garantir qualidade no código.
6. **Disponibilidade e Resiliência:**
   * Sistema configurado para recuperação de desastres (DR) e backups regulares.
   * Balanceamento de carga para evitar quedas.
7. **Usabilidade:**
   * Design responsivo para funcionar bem em dispositivos móveis e desktop.
   * Interface amigável e acessível, atendendo a critérios de acessibilidade (e.g., WCAG).
8. **Eficiência de Recursos:**
   * Utilização otimizada da infraestrutura em nuvem para reduzir custos.
   * Adoção de práticas de CI/CD para agilidade no desenvolvimento.

**Principais Casos de Uso**

**Para o Cliente**

1. **Cadastrar-se no Sistema**
   * **Descrição**: O cliente cria uma conta com seus dados pessoais.
   * **Fluxo Principal**:
     1. O cliente acessa a página de cadastro.
     2. Preenche dados como nome, email, senha e aceita os termos de uso.
     3. Confirma o cadastro via email (se necessário).
2. **Navegar no Catálogo de Produtos**
   * **Descrição**: O cliente visualiza a lista de produtos e usa filtros ou busca para encontrar itens específicos.
   * **Fluxo Principal**:
     1. O cliente acessa o catálogo.
     2. Aplica filtros ou utiliza a barra de busca.
     3. Visualiza detalhes do produto.
3. **Adicionar Produtos ao Carrinho**
   * **Descrição**: O cliente seleciona produtos e os adiciona ao carrinho de compras.
   * **Fluxo Principal**:
     1. O cliente seleciona um produto.
     2. Escolhe a quantidade desejada.
     3. Adiciona o item ao carrinho.
4. **Finalizar Compra**
   * **Descrição**: O cliente conclui o processo de compra.
   * **Fluxo Principal**:
     1. O cliente revisa o carrinho.
     2. Fornece informações de entrega e escolhe o método de pagamento.
     3. Finaliza o pedido e recebe confirmação.
5. **Avaliar Produtos**
   * **Descrição**: O cliente avalia e comenta sobre produtos adquiridos.
   * **Fluxo Principal**:
     1. O cliente acessa o histórico de pedidos.
     2. Seleciona um produto comprado.
     3. Adiciona uma avaliação e/ou comentário.
6. **Gerenciar Conta**
   * **Descrição**: O cliente edita suas informações de perfil e visualiza histórico de compras.
   * **Fluxo Principal**:
     1. O cliente acessa seu perfil.
     2. Atualiza informações como endereço ou senha.
     3. Visualiza o histórico de compras.

**Para o Administrador**

1. **Gerenciar Usuários**
   * **Descrição**: O administrador adiciona, edita ou bloqueia contas de usuários (clientes e vendedores).
   * **Fluxo Principal**:
     1. O administrador acessa o painel de usuários.
     2. Visualiza a lista de contas.
     3. Realiza ações, como edição ou suspensão de acesso.
2. **Supervisionar Produtos**
   * **Descrição**: O administrador revisa produtos cadastrados por vendedores.
   * **Fluxo Principal**:
     1. O administrador acessa a lista de produtos.
     2. Analisa informações do produto.
     3. Aprova, edita ou remove produtos conforme necessidade.
3. **Monitorar Pedidos e Transações**
   * **Descrição**: O administrador acompanha o status de pedidos e transações.
   * **Fluxo Principal**:
     1. O administrador acessa o painel de pedidos.
     2. Filtra por status (em processamento, finalizado etc.).
     3. Acompanha métricas de vendas.

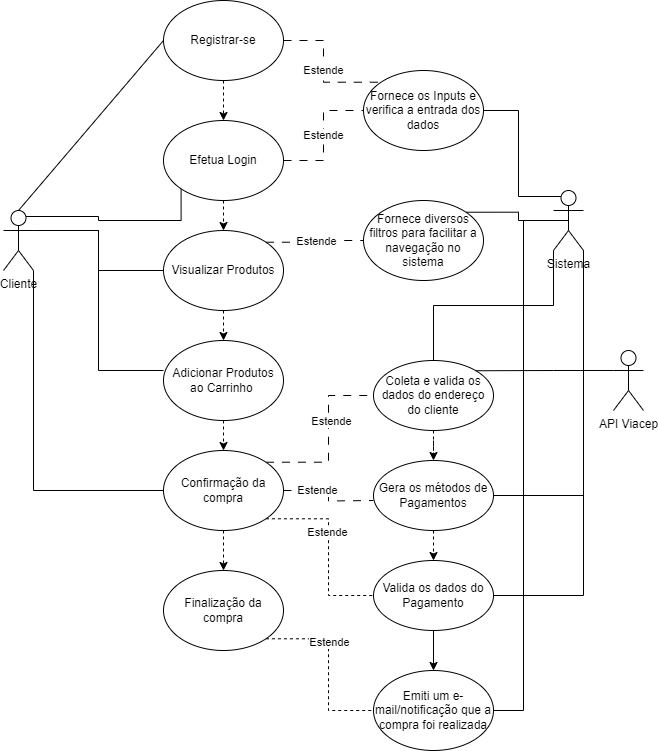
**Para o Vendedor**

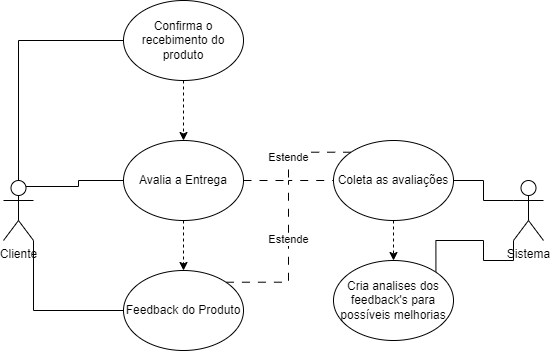
1. **Cadastrar Produtos**
   * **Descrição**: O vendedor adiciona novos produtos ao catálogo.
   * **Fluxo Principal**:
     1. O vendedor acessa o painel de gerenciamento.
     2. Preenche as informações do produto (nome, descrição, preço etc.).
     3. Envia o produto para aprovação do administrador.
2. **Gerenciar Estoque**
   * **Descrição**: O vendedor atualiza informações de estoque.
   * **Fluxo Principal**:
     1. O vendedor acessa a lista de produtos.
     2. Atualiza a quantidade em estoque.
     3. Salva as alterações.
3. **Visualizar Relatórios de Vendas**
   * **Descrição**: O vendedor analisa o desempenho de seus produtos.
   * **Fluxo Principal**:
     1. O vendedor acessa a área de relatórios.
     2. Filtra por período ou produto.
     3. Visualiza métricas como vendas totais e avaliações.

**UML dos Principais Fluxos do Sistema**

Diagrama

Descrição gerada automaticamente





**Escolha da Arquitetura (Microsserviços)**

A arquitetura de microserviços se destaca como a escolha ideal para o cenário apresentado devido à sua capacidade de atender aos requisitos de escalabilidade, flexibilidade, segurança e eficiência necessários para uma startup de e-commerce em crescimento. Abaixo estão as principais razões:

**1. Escalabilidade e Alta Disponibilidade**

O setor de e-commerce, especialmente com foco em produtos sustentáveis, pode vivenciar picos de tráfego devido a campanhas de marketing e sazonalidade.

* **Escalabilidade horizontal:** Microserviços permitem escalar individualmente apenas os componentes mais requisitados, como o serviço de carrinho de compras ou pagamentos. Isso otimiza o uso de recursos na infraestrutura em nuvem.
* **Resiliência:** Cada microserviço opera de forma independente. Assim, se um serviço falhar (por exemplo, envio de notificações), o restante do sistema continua funcionando.

**2. Agilidade no Desenvolvimento e Entrega Contínua**

Com a necessidade de entregas rápidas e contínuas, microserviços facilitam a implementação de novas funcionalidades.

* **Times independentes:** Cada equipe pode trabalhar em um microserviço específico sem interferir nos outros, acelerando o ciclo de desenvolvimento.
* **Implementação gradual:** Novas features podem ser lançadas e integradas sem impactar o restante do sistema, reduzindo riscos.

**3. Integração com Infraestrutura Baseada em Cloud**

O uso de containers (Docker, Kubernetes) combina perfeitamente com microserviços, permitindo:

* Gerenciamento eficiente de cada serviço em contêineres isolados.
* Escalabilidade automatizada com orquestradores como Kubernetes.
* Integração com serviços de terceiros (como gateways de pagamento ou APIs de logística) por meio de interfaces bem definidas.

**4. Boas Práticas de Segurança e Conformidade**

A gestão de dados pessoais e transações financeiras requer uma abordagem robusta para segurança e conformidade com LGPD/GDPR.

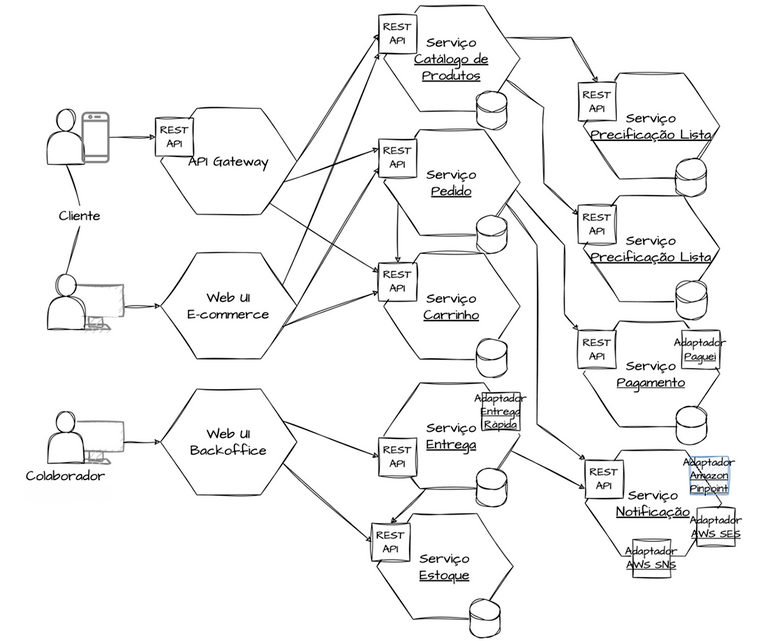
* **Isolamento de dados:** Microserviços podem ser configurados para armazenar dados sensíveis em serviços dedicados, com criptografia e controle de acesso avançado.
* **Auditoria e conformidade:** A separação de responsabilidades entre microserviços facilita auditorias e adequação a regulamentos.

**5. Suporte ao Crescimento Futuro**

Com a expansão da equipe e do sistema, a arquitetura de microserviços:

* Permite a fácil inclusão de novos desenvolvedores, já que cada microserviço tem escopo reduzido e documentação própria.
* Suporta o crescimento modular do sistema, adicionando novos serviços sem reescrever a base existente.

**Desenho da Arquitetura**

****

**Diagrama de Componentes**

**Diagrama

Descrição gerada automaticamente**

**Diagrama de Sequência**

**Linha do tempo

Descrição gerada automaticamente**

**Diagrama de Classes**

**Desenho técnico

Descrição gerada automaticamente com confiança baixa**

**Detalhamento de Classes e Funções**

**class Customer**

**Attributes**

**ID: Integer PK**

**Name: String**

**Birthday: Date**

**CPF: String**

**AddressID: Address FK**

**Methods**

**GetAge**

**return CurrentDate.Year - Birthday.Year**

**class Address**

**Attributes**

**ID: Integer PK**

**CEP: String**

**Logradouro: String**

**Complemento: String**

**Bairro: String**

**Localidade: String**

**UF: String**

**Estado: String**

**Regiao: String**

**class Product**

**Attributes**

**ID: Integer PK**

**Name: String**

**Quantity: Integer**

**Price: Double**

**CategoryID: Category FK**

**SellerID: Seller FK**

**class Order**

**Attributes**

**ID: Integer PK**

**CustomerID: Customer FK**

**Products[]: Array<Product>**

**TotalValue: Float**

**PaymentID: Payment FK**

**Methods**

**CalculateTotalValue**

**if (!Products.empty)**

**for (Product in Products)**

**TotalValue += Product.Price \* Product.Quantity**

**else**

**return 0**

**class Cart**

**Attributes**

**ID: Integer PK**

**CustomerID: Customer FK**

**Products[]: Array<Product>**

**Methods**

**AddProduct(product: Product)**

**Products.append(product)**

**RemoveProduct(productID: Integer)**

**Products.removeIf(product => product.ID == productID)**

**GetCartTotal**

**total = 0**

**for (Product in Products)**

**total += Product.Price \* Product.Quantity**

**return total**

**class Payment**

**Attributes**

**ID: Integer PK**

**Method: String**

**Status: String**

**PaymentDate: Date**

**Amount: Float**

**class Category**

**Attributes**

**Id: Integer PK,**

**Name: String**

**class Seller**

**Attributes**

**ID: Integer PK,**

**Name: String,**

**CNPJ: String,**

**Products[]: Array<Product>**

**AddressID: Address FK**

**class Notification**

**Attributes**

**ID: Integer PK**

**Message: String**

**SentDate: Date**

**CustomerID: Customer FK**

**Estratégia de Escalabilidade e Segurança**

**Estratégias de Escalabilidade**

**1. Escalabilidade Horizontal**

* Containers e Orquestração: Utilização do Docker para empacotar microserviços e orquestradores como Kubernetes para gerenciar escalabilidade automática. Isso permitirá replicar serviços críticos (ex.: Catálogo e Carrinho) durante picos de tráfego.
* Cluster de Banco de Dados: Configuração dos bancos de dados como PostgreSQL ou MongoDB em clusters para dividir a carga de leitura/escrita, usando réplicas.
* CDN (Content Delivery Network): Uso de uma CDN como Cloudflare ou AWS CloudFront para distribuir conteúdo estático (imagens, CSS, JavaScript), reduzindo a latência.

**2. Cacheamento**

* Cache para Dados Frequentes: Adoção de ferramentas como Redis ou Memcached para armazenar dados de acesso frequente, como informações de produtos e carrinhos de compras.
* API Gateway Cache: Configuração de caches nas respostas de APIs frequentemente requisitadas, como buscas no catálogo.

**3. Desacoplamento com Mensageria**

* Filas de Mensagem: Utilização de sistemas como RabbitMQ ou Apache Kafka para desacoplar serviços e gerenciar tarefas assíncronas (ex.: processamento de pagamentos e notificações).
* Event-Driven Architecture: Eventos como confirmação de pagamento, disparem serviços relevantes independentemente.

**4. Autoescala e Balanceamento de Carga**

* Auto Scaling Groups: Configuração do AWS Auto Scaling ou equivalente para aumentar ou diminuir o número de instâncias de acordo com a demanda.
* Balanceadores de Carga: Uso de um balanceador de carga como AWS ALB/ELB ou NGINX para distribuir solicitações entre instâncias de backend.

**5. Separação de Bancos de Dados**

* Shard por Domínio: Separação de bancos por domínio funcional (ex.: banco de clientes, produtos e pedidos).
* Particionamento por Região: Para atender usuários de diferentes regiões, configuração dos bancos de dados particionado geograficamente.

**Estratégias de Segurança**

**1. Proteção de Dados**

* Criptografia em Descanso e em Trânsito: Uso de AES-256 para criptografar dados em repouso e TLS 1.3 para tráfego entre clientes e servidores.
* Tokenização: Substituição de dados sensíveis, como números de CPF, por tokens armazenados em um local seguro.
* Controle de Acesso a Bancos de Dados: Uso de privilégios mínimos para usuários de banco de dados, separando funções de leitura/escrita.

**2. Autenticação e Autorização**

* OAuth 2.0: Implementação do OAuth 2.0 para autenticação segura, especialmente em integrações com terceiros.
* Multi-Factor Authentication (MFA): MFA para acessos administrativos e, opcionalmente, para usuários finais.
* RBAC (Role-Based Access Control): Controle baseado em papéis para restringir acessos por tipo de usuário (cliente, vendedor, administrador).

**3. Proteção Contra Ataques**

* Firewall de Aplicação Web (WAF): Utilização WAFs, como o AWS WAF, para proteger contra ataques como SQL Injection, Cross-Site Scripting (XSS) e outros.
* Prevenção de DDoS: Proteção contra DDoS com ferramentas como AWS Shield ou Cloudflare.
* Rate Limiting: Limitação do número de requisições por cliente para evitar abuso de APIs.

**4. Monitoramento e Auditoria**

* Registro de Logs: Centralização de logs de eventos críticos (ex.: login, transações) usando ferramentas como ELK Stack ou Datadog.
* Auditoria Periódica: Auditorias regulares para verificar conformidade com LGPD/GDPR e identificar vulnerabilidades.

**5. Conformidade com LGPD/GDPR**

* Consentimento do Usuário: O consentimento do usuário deve ser coletado de forma explícita para o armazenamento e processamento de dados pessoais.
* Direito ao Esquecimento: Implementação de APIs para exclusão permanente de dados, quando solicitado pelo usuário.
* Data Protection Officer (DPO): Contratação de um profissional responsável pela governança dos dados.

**6. Proteção Contra Vazamentos**

* Segurança em APIs: Autenticação via tokens JWT, validação de entradas e políticas de tempo de expiração para chaves.
* Proteção de Armazenamento: Para serviços em nuvem como AWS S3, uso de criptografia de objetos e buckets privados.