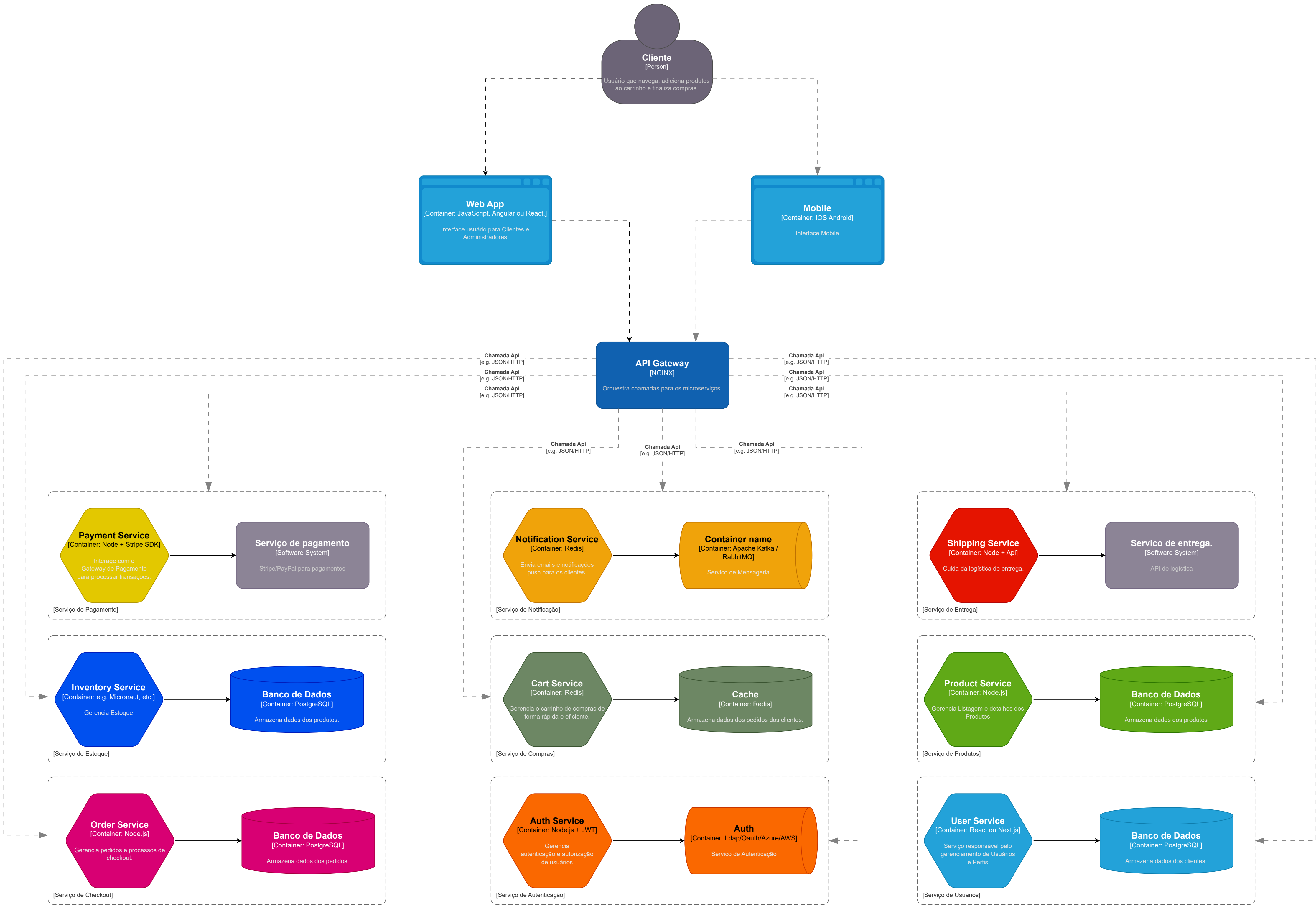


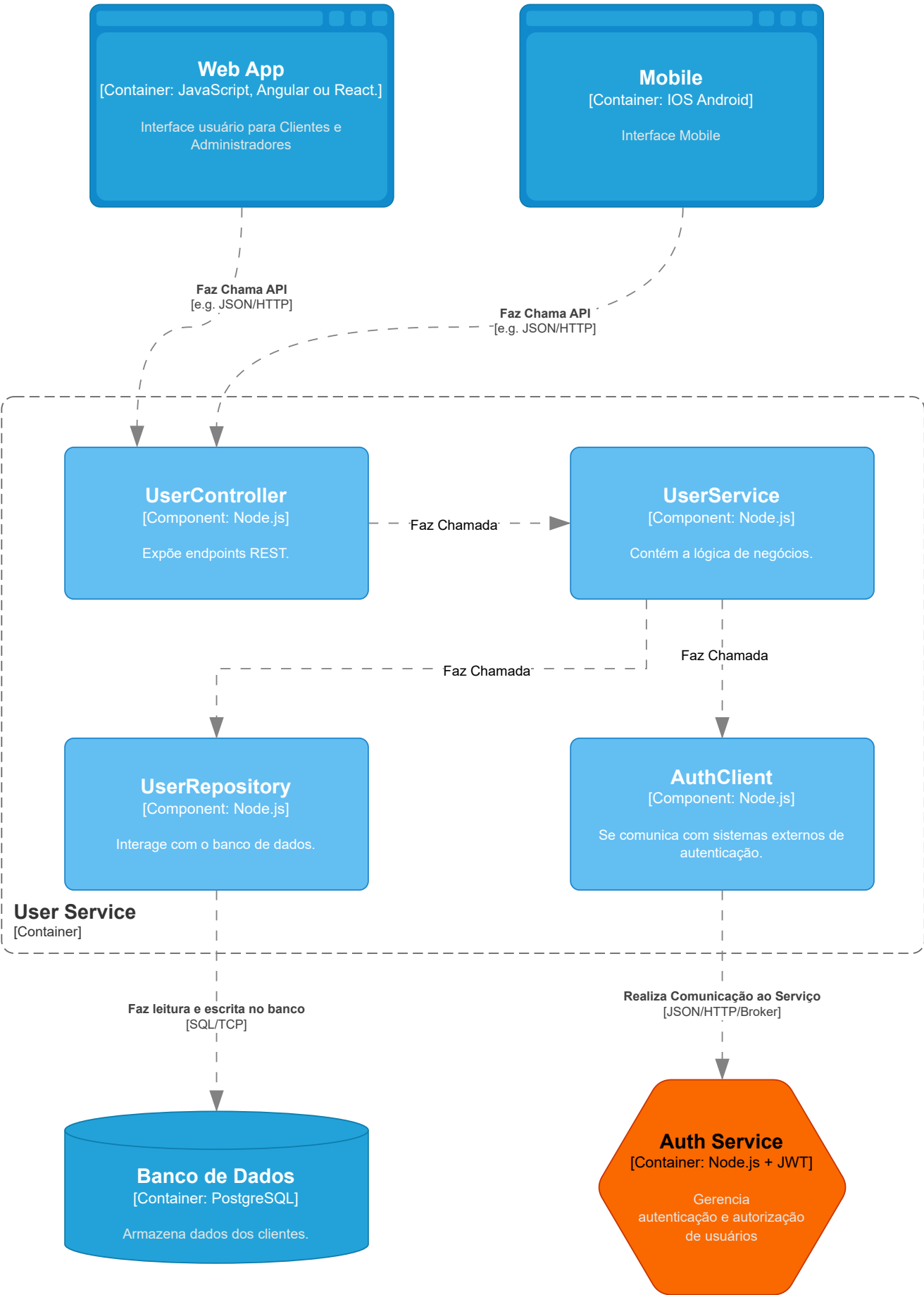
### Nível 1 [Diagrama Contexto]

Diagrama de Contexto para uma plataforma de e-Commerce baseada em Microserviços.

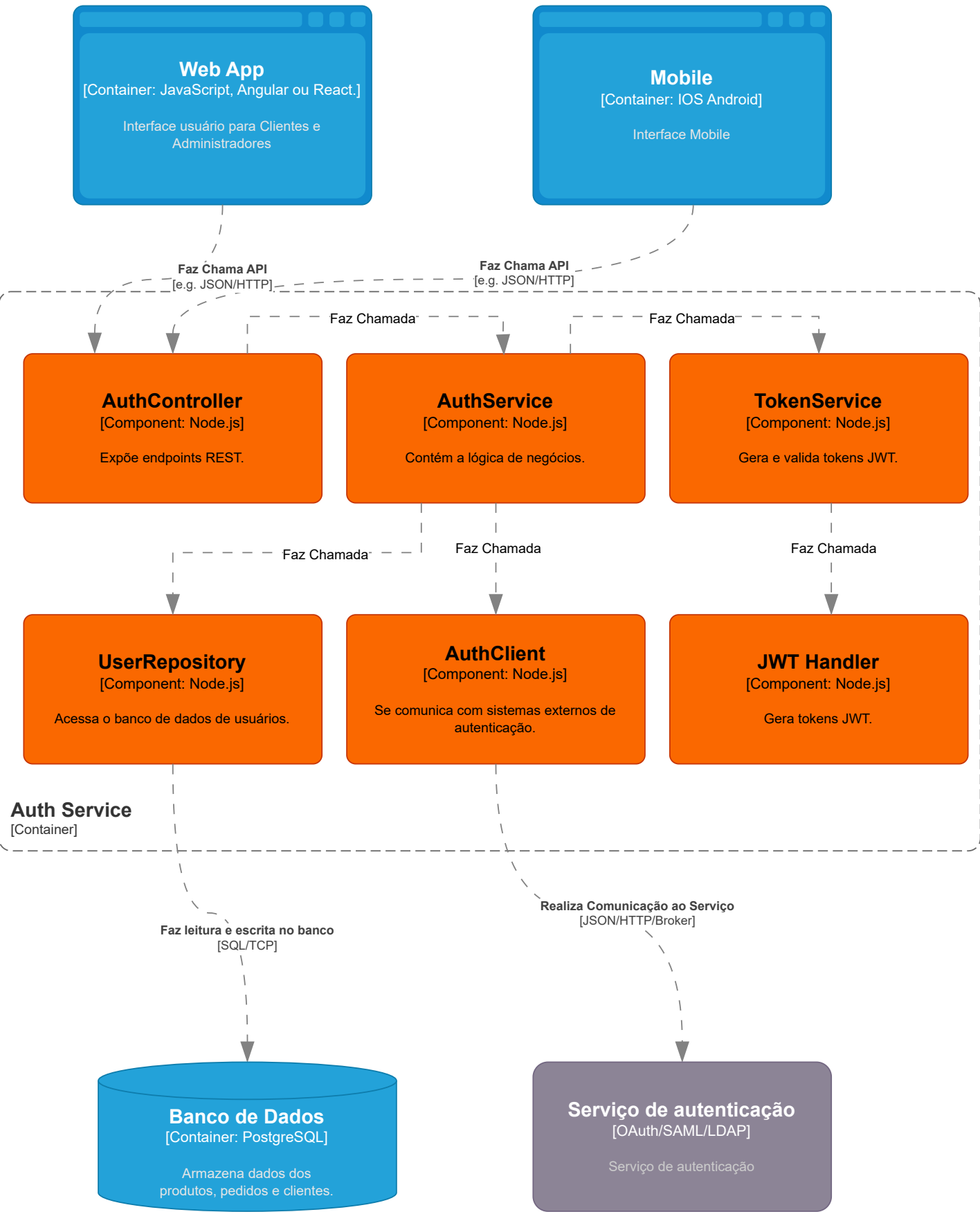


Nível 2 [Diagrama de Containers]  
Diagram de container para e-commerce

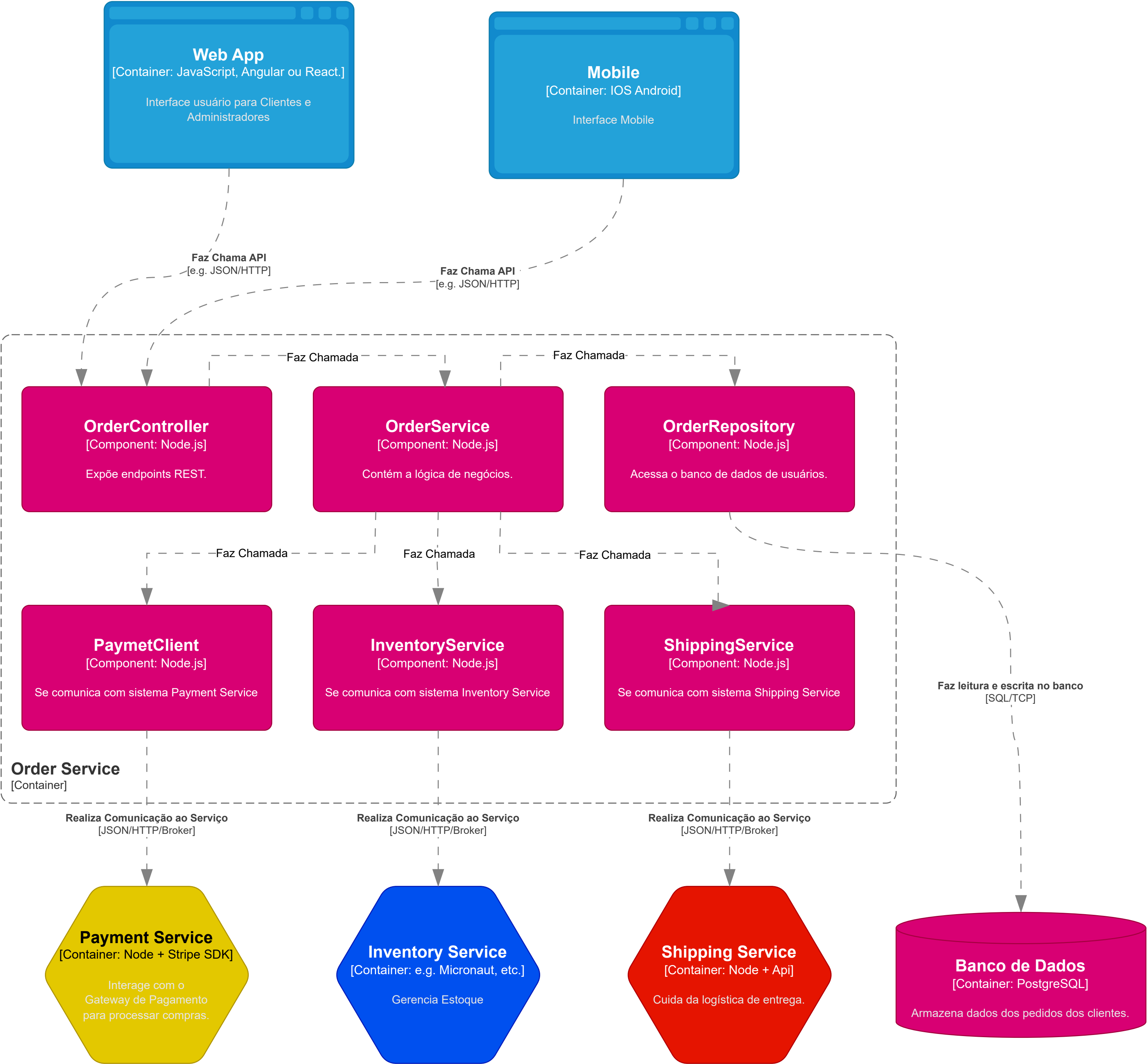
**User Service**  
Componente para serviço de usuários e perfis



**Auth Service**  
Componente para serviço de autenticação



**Order Service**  
Componente Para Gerenciar Pedidos



# ADR: Arquitetura Baseada em Microserviços para um E-commerce

**Data:** 22/02/2025

**Status:** Aprovado

**Autor:** Clever Santoro Lopes

## 1. Contexto

Nossa startup está construindo uma plataforma de e-commerce que precisa ser **escalável, resiliente e modular**. A arquitetura deve permitir que cada parte do sistema evolua independentemente e suporte alto volume de transações.

Decidimos seguir o **Modelo C4** para documentar a arquitetura e garantir um design claro e compreensível para todos os stakeholders.

### Requisitos-chave

- ✓ Alta disponibilidade e escalabilidade.
- ✓ Modularidade para facilitar manutenção e evolução.
- ✓ Facilidade para integrar novos serviços, como novos gateways de pagamento.
- ✓ Observabilidade para monitoramento e depuração.

## 2. Decisão

Optamos por uma **arquitetura baseada em microserviços** com os seguintes elementos:

### Camadas da Arquitetura

- Frontend Web (React/Next.js)**
- API Gateway (NestJS ou Spring Cloud Gateway)**
- Microserviços Independentes**
  - User Service:** Gerenciamento de usuários.
  - Order Service:** Processamento de pedidos.
  - Payment Service:** Integração com gateways de pagamento.
  - Inventory Service:** Controle de estoque.
  - Shipping Service:** Gerenciamento de entregas.
- Bancos de Dados**
  - PostgreSQL** para dados transacionais.
  - Redis** para cache e filas assíncronas.
- Observabilidade**
  - Log Aggregation** (Elastic Stack, Loki).
  - Tracing** (OpenTelemetry, Jaeger).
  - Metrics** (Prometheus, Grafana).

### Tecnologias Escolhidas

Componente	Tecnologia	Justificativa
Frontend	Next.js / React	SEO, SSR/SSG, UX moderna

Componente	Tecnologia	Justificativa
API Gateway	NestJS / Spring Gateway	Gerenciamento de rotas, segurança
Microserviços	Node.js (NestJS) ou Java (Spring Boot)	Performance, escalabilidade
Banco Relacional	PostgreSQL	Consistência de dados
Cache	Redis	Melhor resposta e escalabilidade
Mensageria	Kafka / RabbitMQ	Comunicação assíncrona entre serviços
Observabilidade	Prometheus, Grafana, Jaeger	Monitoramento proativo

## 3. Alternativas Consideradas

### 3.1 Monólito

- **Rejeitado** – Não atende à escalabilidade e modularidade exigidas.

### 3.2 Microserviços sem API Gateway

- **Rejeitado** – Difícil gerenciar comunicação direta entre os serviços, aumento da complexidade.

### 3.3 Arquitetura Serverless

- **Parcialmente Considerado** – Possível no futuro para algumas funções, mas alto acoplamento com provedores cloud pode limitar flexibilidade.

## 4. Consequências

### ✅ Vantagens

- Maior **escalabilidade e resiliência**.
- Manutenção facilitada com equipes independentes para cada serviço.
- Possibilidade de **desenvolvimento paralelo** sem grandes dependências.
- Uso de filas assíncronas para comunicação eficiente entre serviços.

### ⚠️ Desafios

- Requer um **maior investimento inicial** em automação e infraestrutura.
- **Gerenciamento de consistência** entre bancos de dados distribuídos.
- A complexidade de **monitoramento e debugging** é maior do que em um monólito.

## 5. Próximos Passos

- 🚀 Configuração de **CI/CD** para automação de deploys.
- 🔍 Implementação de **observabilidade completa** (logging, tracing e métricas).
- 🧪 Testes de carga para garantir a **resiliência e escalabilidade**.

## 🎯 Conclusão

Essa decisão de arquitetura possibilita uma plataforma **robusta, flexível e escalável** para nosso e-commerce, garantindo a evolução da startup sem gargalos tecnológicos.

# ADR: Arquitetura do User Service

- 📌 **Data:** 22/02/2025
- 📌 **Status:** Aprovado
- 📌 **Autor:** Clever Santoro Lopes

## 1. Contexto

O **User Service** é responsável por gerenciar os dados de usuários na plataforma de e-commerce. Ele deve garantir:

- ✅ **Cadastro, atualização e exclusão de usuários.**
- ✅ **Gerenciamento de perfis e permissões.**
- ✅ **Integração com o Auth Service para autenticação.**
- ✅ **Alta disponibilidade e segurança dos dados.**

## 2. Decisão

Optamos por um **serviço independente para gerenciamento de usuários**, garantindo escalabilidade e segurança.

### 📌 Tecnologias Escolhidas

Componente	Tecnologia	Justificativa
Linguagem	Node.js (NestJS) ou GoLang	Performance e suporte a APIs
Banco de Dados	PostgreSQL	Armazena usuários e permissões
Cache	Redis	Armazena dados temporários e sessões
Autenticação	Integração com Auth Service	Segurança centralizada
Mensageria	Kafka / RabbitMQ	Notificação de eventos de usuários
Monitoramento	Prometheus + Grafana	Logs e métricas

### 📌 Arquitetura do User Service

- 1 **API Gateway** → Direciona requisições para o User Service.
- 2 **User Service** → Gerencia operações de CRUD de usuários.
- 3 **Banco de Dados** → Armazena perfis e permissões.
- 4 **Integração com Auth Service** → Autenticação centralizada.
- 5 **Mensageria** → Envia eventos de alteração de usuários para outros serviços.

## 3. Alternativas Consideradas

### 3.1 Armazenar Usuários no Auth Service

- ❌ **Rejeitado** – Aumenta acoplamento entre serviços e dificulta evolução independente.

### 3.2 Monólito com Usuários e Autenticação juntos

- ❌ **Rejeitado** – Menos escalável e limita flexibilidade para futuros ajustes.



### 3.3 User Service Independente

✓ **Aprovado** – Permite escalabilidade e integração modular com outros serviços.

---

## 4. Consequências

### ✓ Benefícios

- ✓ **Modularidade** – Evolução independente do Auth Service.
- ✓ **Escalabilidade** – Capacidade de distribuir carga entre instâncias.
- ✓ **Segurança** – Proteção de dados de usuários separada da autenticação.
- ✓ **Integração facilitada** – Notificação de eventos via mensageria.

### ⚠ Desafios

- ⚠ **Sincronização com Auth Service** → Necessário garantir consistência de dados.
  - ⚠ **Gerenciamento de Permissões** → Pode aumentar a complexidade inicial da implementação.
- 

## 5. Próximos Passos

- 🚀 Implementação de **controle de acesso granular (RBAC/ABAC)**.
  - 🔍 **Monitoramento** com OpenTelemetry para rastrear acessos e alterações de usuários.
  - 📊 **Testes de carga e segurança** para validar resiliência e performance.
- 

### 🎯 Conclusão

O **User Service** fornecerá um gerenciamento de usuários seguro, escalável e modular, garantindo flexibilidade na evolução da plataforma.

# ADR: Arquitetura do Auth Service

- ✦ **Data:** 22/02/2025
- ✦ **Status:** Aprovado
- ✦ **Autor:** Clever Santoro Lopes

## 1. Contexto

O **Auth Service** é responsável por autenticação e autorização de usuários no e-commerce. Ele deve garantir:

- ✓ **Autenticação segura** (tokens JWT ou OAuth 2.0).
- ✓ **Autorização baseada em papéis (RBAC)** para controle de acesso.
- ✓ **Integração com provedores externos** (Google, Facebook, Apple).
- ✓ **Alta disponibilidade e resiliência contra ataques** (ex: força bruta).

## 2. Decisão

Optamos por um **serviço de autenticação centralizado** utilizando **OAuth 2.0 + JWT**, garantindo escalabilidade e segurança.

### ✦ Tecnologias Escolhidas

Componente	Tecnologia	Justificativa
Linguagem	Node.js (NestJS) ou GoLang	Performance e suporte a APIs
Banco de Dados	PostgreSQL	Armazena usuários e permissões
Cache	Redis	Armazena tokens para revogação rápida
Autenticação	JWT + OAuth 2.0	Segurança e suporte a terceiros
Mensageria	Kafka / RabbitMQ	Eventos de login/logout
Monitoramento	Prometheus + Grafana	Logs e métricas de segurança

### ✦ Arquitetura do Auth Service

- 1 API Gateway** → Direciona requisições para o Auth Service.
- 2 Auth Service** → Gera tokens JWT, valida credenciais e gerencia permissões.
- 3 Banco de Dados** → Armazena informações de usuários e sessões.
- 4 Integração com OAuth** → Permite login com Google, Facebook, Apple.
- 5 Monitoramento e Segurança** → Protege contra ataques e acessos suspeitos.

## 3. Alternativas Consideradas

### 3.1 Autenticação com Sessões

● **Rejeitado** – Depende de sticky sessions e não escala bem em múltiplas instâncias.

### 3.2 OpenID Connect com Keycloak/Auth0

● **Parcialmente Considerado** – Boa opção para reduzir complexidade, mas pode gerar custos adicionais.

### 3.3 OAuth 2.0 com JWT

✓ **Aprovado** – Permite autenticação distribuída, segura e escalável.

---

## 4. Consequências

### ✓ Benefícios

- ✓ **Autenticação segura** com tokens assinados.
- ✓ **Escalabilidade** sem depender de sessões no servidor.
- ✓ **Integração com provedores externos** (SSO).
- ✓ **Monitoramento e rastreabilidade** de logins e acessos.

### ⚠ Desafios

- ⚠ **Revogação de tokens** → Necessário cache distribuído (Redis) para blacklisting.
  - ⚠ **Gerenciamento de OAuth** → Pode aumentar a complexidade inicial da implementação.
- 

## 5. Próximos Passos

- 🚀 Implementação de **MFA (autenticação multifator)** para maior segurança.
  - 🔍 **Monitoramento** com OpenTelemetry para rastrear tentativas de login.
  - 📊 **Testes de carga e segurança** para validar resistência contra ataques.
- 

### 🎯 Conclusão

O **Auth Service** fornecerá autenticação segura, escalável e integrada com provedores externos, garantindo a segurança do e-commerce.

# ADR: Arquitetura do Order Service

- 📅 **Data:** 22/02/2025
- ✅ **Status:** Aprovado
- 👤 **Autor:** Clever Santoro Lopes

## 1. Contexto

O **Order Service** é um dos principais microserviços do e-commerce, responsável pelo processamento de pedidos. Ele precisa garantir:

- ✅ Consistência transacional entre pagamento, estoque e envio.
- ✅ Alta disponibilidade e escalabilidade para lidar com picos de tráfego.
- ✅ Baixa latência para proporcionar uma boa experiência ao usuário.
- ✅ Tolerância a falhas para evitar pedidos incorretos ou duplicados.

## 2. Decisão

Optamos por uma arquitetura baseada em microserviços utilizando as seguintes tecnologias e padrões:

### Tecnologias

Componente	Tecnologia	Justificativa
Linguagem	Node.js (NestJS) ou Java (Spring Boot)	Performance e escalabilidade
Banco de Dados	PostgreSQL	Consistência transacional
Cache	Redis	Melhor tempo de resposta
Mensageria	Kafka / RabbitMQ	Comunicação assíncrona
Monitoramento	Prometheus + Grafana	Observabilidade e métricas

### Arquitetura do Order Service

O serviço segue um design orientado a eventos, garantindo que pedidos sejam processados de forma assíncrona e resiliente.

- API Gateway → Recebe requisições de pedidos.
- Order Service → Processa pedidos e gerencia a lógica de negócios.
- Integração com Pagamentos → Usa Payment Client para confirmar o pagamento.
- Validação de Estoque → Usa Inventory Client para verificar disponibilidade.
- Geração de Eventos → Usa Kafka/RabbitMQ para processar pedidos assincronamente.

## 3. Alternativas Consideradas


### 3.1 Processamento Síncrono

- ❌ Rejeitado – Aumentaria a latência e dependeria de respostas imediatas do Payment Service e Inventory Service.

### 3.2 Banco de Dados Centralizado




-  Rejeitado – Criaria um ponto único de falha e dificultaria a escalabilidade.

### 3.3 Arquitetura Event-Driven



-  Aprovado – Permite processamento assíncrono, escalabilidade e maior resiliência.
- 

## 4. Consequências




### Benefícios

-  Maior escalabilidade → O uso de eventos desacopla os serviços.
-  Menor tempo de resposta → Redis e filas assíncronas otimizam a performance.
-  Melhor tolerância a falhas → Se um serviço falhar, o pedido pode ser reprocessado.

### Desafios

-  Maior complexidade operacional → Requer monitoramento avançado e mecanismos de retry.
  -  Gestão de consistência eventual → Sincronização de eventos pode ser desafiadora.
- 

## 5. Próximos Passos

-  Implementação de SAGA Pattern para garantir consistência entre serviços.
  -  Monitoramento com OpenTelemetry para rastrear transações distribuídas.
  -  Testes de carga para validar a escalabilidade do serviço.
- 

## Conclusão

Essa arquitetura garante que o Order Service seja confiável, escalável e resiliente, suportando o crescimento do e-commerce da startup.