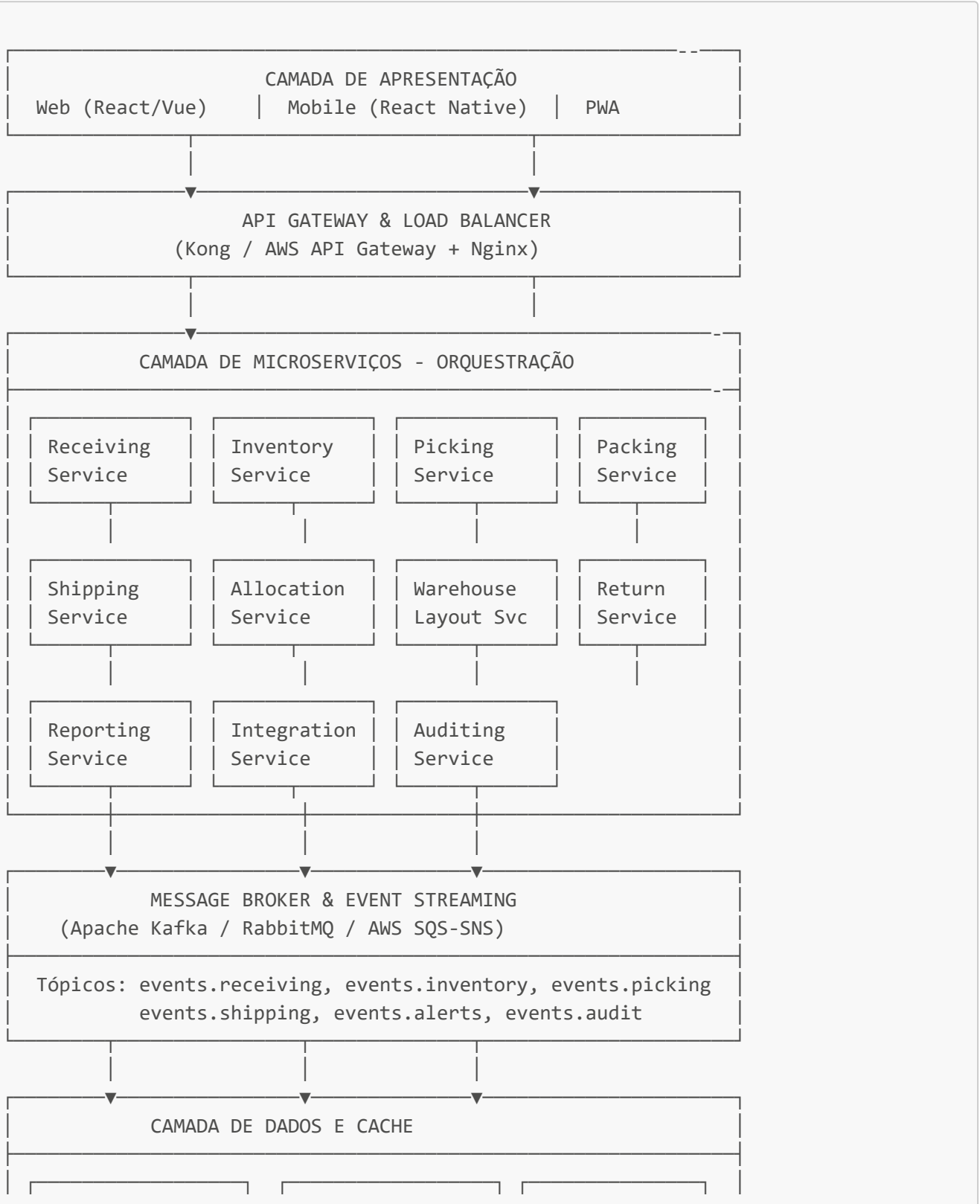
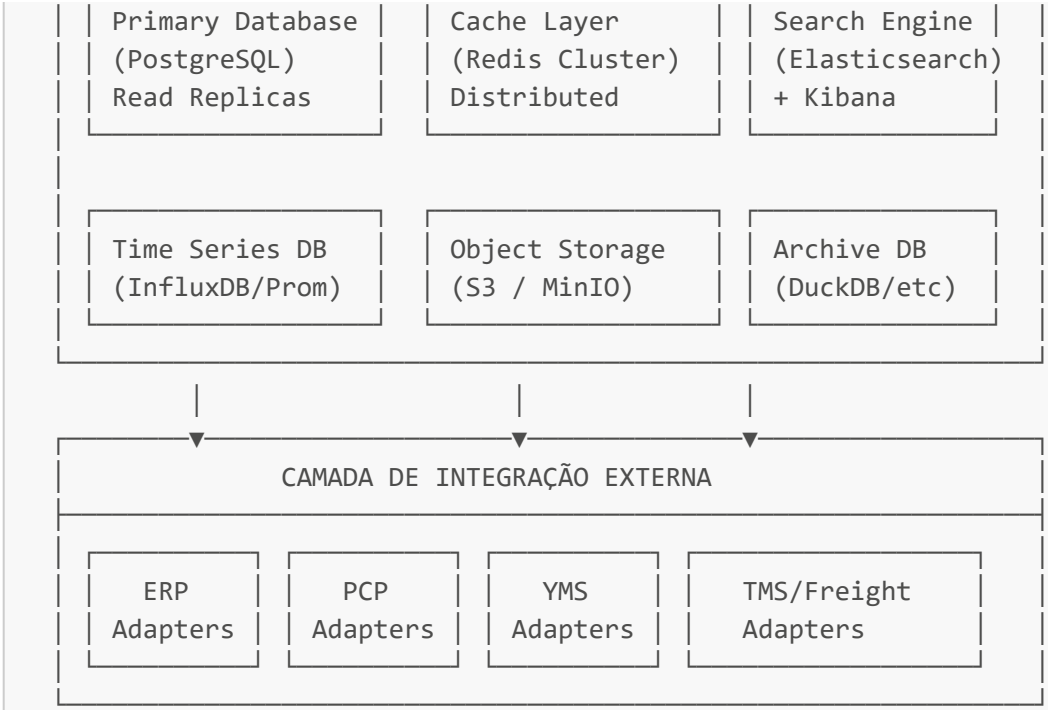
ARQUITETURA DO WMS ENTERPRISE

1. Visão Geral Arquitetural

1.1 Padrão Arquitetural

O WMS Enterprise utiliza arquitetura **microserviços com event-driven** combinada com **CQRS (Command Query Responsibility Segregation)** para máxima escalabilidade, manutenibilidade e performance.





---

2. Componentes de Negócio (Microserviços)

2.1 Receiving Service (Serviço de Recebimento)

Responsabilidades:

- Gestão de avisos de chegada (ASN)
- Validação de NF contra PO
- Gerenciamento de operações de descarregamento
- Inspeção de qualidade
- Conferência de quantidades
- Movimentação inicial para armazenagem

Dependências:

- `allocation-service`: Para alocação de localização
- `inventory-service`: Para atualização de stock
- `audit-service`: Para registro de todas as operações
- Message Broker: Para publicar eventos

API Endpoints:

```
POST   /api/v1/inbound/asn                - Criar ASN
GET    /api/v1/inbound/asn/{asnId}        - Obter detalhes
PUT    /api/v1/inbound/asn/{asnId}        - Atualizar ASN
POST   /api/v1/inbound/{asnId}/receive    - Iniciar recebimento
POST   /api/v1/inbound/{asnId}/inspect    - Inspeção de qualidade
POST   /api/v1/inbound/{asnId}/confirm    - Confirmar recebimento
```

Eventos Publicados:

- `inbound.asn.created`
- `inbound.asn.received`
- `inbound.asn.confirmed`
- `inbound.quality.failed`
- `inbound.discrepancy.detected`

2.2 Inventory Service (Serviço de Inventário)

Responsabilidades:

- Manutenção do estado de inventário em tempo real
- Controle de stock por SKU, localização, lote
- Reserva de itens para pedidos
- Ajustes de inventário
- Contagem cíclica
- Alertas de stock mínimo/máximo

Padrão CQRS:

- **Commands (Escrita):** Reserve, Release, Adjust, Transfer
- **Queries (Leitura):** GetInventory, GetAvailability, GetHistory

Estrutura de Dados:

```
TABLE inventory_snapshot (
    id UUID PRIMARY KEY,
    sku_id UUID,
    location_id UUID,
    batch_id UUID,
    quantity_on_hand INT,
    quantity_reserved INT,
    quantity_available INT,
    warehouse_id UUID,
    tenant_id UUID,
    updated_at TIMESTAMP,
    version INT  -- Para otimistic locking
)

TABLE inventory_transaction_log (
    id UUID PRIMARY KEY,
    inventory_id UUID,
    transaction_type ENUM (RECEIVE, PICK, ADJUST, TRANSFER),
    quantity_before INT,
    quantity_after INT,
    reason TEXT,
    user_id UUID,
    timestamp TIMESTAMP
)
```

Eventos Publicados:

- `inventory.reserved`
- `inventory.released`
- `inventory.low_stock_alert`
- `inventory.overstock_alert`
- `inventory.expired_alert`

2.3 Allocation Service (Serviço de Alocação)

Responsabilidades:

- Algoritmos de alocação inteligente
- Otimização de localização para produtos
- Rebalanceamento de inventário
- Cálculo de rota de picking
- Sugestão de consolidação

Algoritmos Implementados:

1. **ABC Analysis** - Classifica produtos por giro
2. **Correlation Analysis** - Produtos vendidos juntos
3. **Wave Optimization** - Agrupa pedidos similares
4. **Route Optimization** - Minimiza distância/tempo
5. **ML-based Forecasting** - Predição de demanda

API Endpoints:

```
POST   /api/v1/allocation/suggest          - Sugerir alocação
POST   /api/v1/allocation/optimize         - Otimizar layout
GET    /api/v1/allocation/location/{skuId} - Localização sugerida
```

2.4 Picking Service (Serviço de Picking)

Responsabilidades:

- Criação de ordens de picking
- Otimização de rota de picking
- Rastreamento de progresso
- Validação de itens coletados
- Consolidação em ponto de coleta

Modos de Operação:

- Single line picking
- Batch picking

- Zone picking
- Wave picking
- Pick-to-light
- Voice picking
- Mobile picking

Estrutura:

```
TABLE picking_order (
    id UUID PRIMARY KEY,
    wave_id UUID,
    operator_id UUID,
    status ENUM (CREATED, IN_PROGRESS, COMPLETED, FAILED),
    created_at TIMESTAMP,
    completed_at TIMESTAMP,
    location_sequence TEXT  -- JSON com rota otimizada
)

TABLE picking_line (
    id UUID PRIMARY KEY,
    picking_order_id UUID,
    sku_id UUID,
    location_id UUID,
    quantity_required INT,
    quantity_picked INT,
    sequence INT,  -- Ordem da rota
    completed_at TIMESTAMP
)
```

Eventos Publicados:

- `picking.order.created`
- `picking.order.started`
- `picking.line.completed`
- `picking.order.completed`
- `picking.quality.issue`

2.5 Packing Service (Serviço de Embalagem)

Responsabilidades:

- Gestão de processos de embalagem
- Seleção de embalagem apropriada
- Geração de etiquetas
- Pesagem e medição
- Consolidação para expedição

Características:

- Integração com balança eletrônica
- Impressora de etiquetas em tempo real
- Validação de peso vs. esperado
- Sugestão de tamanho de embalagem

2.6 Shipping Service (Serviço de Expedição)

Responsabilidades:

- Gestão de remessas para expedição
- Integração com TMS (Transport Management System)
- Geração de documentação fiscal
- Consolidação de cargas
- Rastreamento até cliente

Integrações:

- TMS para roteirização
- SEFAZ para NF-e
- Transportadoras para tracking
- Sistema de notificação ao cliente

2.7 Reporting Service (Serviço de Relatórios)

Responsabilidades:

- Agregação de dados operacionais
- Geração de relatórios sob demanda
- Dashboard em tempo real
- Alertas e notificações
- Business Intelligence

Data Pipeline:

```
Raw Events (Kafka)
      ↓
[Stream Processing - Kafka Streams]
      ↓
[Data Warehouse - Dimension Tables]
      ↓
[OLAP Cube / Star Schema]
      ↓
[Visualization Layer - Grafana/Tableau]
```

---

3. Camada de Apresentação

3.1 Web Application (Desktop/Tablet)

Stack Tecnológico:

- **Framework:** React.js 18+ / Vue.js 3+
- **State Management:** Redux Toolkit / Pinia
- **UI Components:** Material-UI / Element UI
- **Build Tool:** Vite
- **Testing:** Vitest + React Testing Library

Funcionalidades:

- Dashboard em tempo real
- CRUD de entidades principais
- Relatórios e analytics
- Gestão de usuários
- Configurações de sistema

3.2 Mobile Application

Stack Tecnológico:

- **Framework:** React Native / Flutter
- **Offline Sync:** Redux Offline / Workbox
- **Camera:** Para leitura de QR code
- **GPS:** Para geolocalização

Funcionalidades:

- Recebimento de mercadorias
- Picking com rota otimizada
- Packing
- Expedição
- Inspeção de qualidade

3.3 Progressive Web App (PWA)

- Funciona offline
- Sincronização em background
- Install como app nativo
- Notificações push

---

4. Camada de Dados

4.1 PostgreSQL (Banco Relacional Principal)

Características:

- Multi-tenancy com Row Level Security (RLS)
- JSONB para flexibilidade

- Particionamento por data
- Read replicas para queries pesadas
- Backup automático e PITR

Tabelas Principais:

```
tenants
users / roles / permissions
warehouses / locations / aisles / racks
skus / products / categories
storage_types / handling_rules
suppliers / customers / transporters
inbound_asn / receiving_documents
inventory_master / inventory_transactions
orders / order_lines / picking_orders
shipments / packages
returns / claims
batch_operations / audit_log
```

4.2 Redis (Cache & Sessions)

Uso:

- Cache de consultas frequentes (Inventory state)
- Session storage com TTL
- Rate limiting
- Pub/Sub para notificações em tempo real
- Distributed locks para operações críticas

Topology:

- Redis Cluster (HA)
- Persistência com RDB + AOF
- Master-Replica replication

4.3 Elasticsearch (Search & Analytics)

Uso:

- Busca em tempo real de pedidos, shipments
- Análise de logs
- Auditoria com full-text search

Índices:

- `orders-*` (time-series)
- `shipments-*`
- `audit-logs-*`
- `inventory-history-*`

4.4 Time Series Database (InfluxDB/Prometheus)

Métricas Armazenadas:

- Performance de operadores (pedidos/hora)
- Tempos de processamento
- Utilização de localização
- Alertas do sistema

---

5. Message Broker (Event Streaming)

5.1 Apache Kafka / AWS Kinesis

Topologia:

```
inbound.* events
├── inbound.asn.created
├── inbound.asn.received
├── inbound.quality.failed
└── inbound.confirmed

inventory.* events
├── inventory.reserved
├── inventory.released
├── inventory.low_stock
└── inventory.transfer

picking.* events
├── picking.order.created
├── picking.order.completed
└── picking.quality.issue

shipping.* events
├── shipping.prepared
├── shipping.manifested
└── shipping.dispatched

alerts.* events
├── alerts.system_error
├── alerts.sla_breach
└── alerts.capacity_alert

audit.* events
└── audit.operation
```

Características:

- Retenção: 7-30 dias
- Replication factor: 3
- Particionamento por tenant_id ou location_id

- Consumer groups para cada serviço

---

6. Padrões de Design e Integração

6.1 Service-to-Service Communication

Síncrono (gRPC/REST):

- Para chamadas que precisam de resposta imediata
- Timeout: 5-30 segundos
- Circuit breaker ativo

Assíncrono (Event Streaming):

- Para operações que podem ser eventual-consistent
- Replay de eventos para recuperação
- Idempotência garantida

6.2 Multi-tenancy

- Isolamento no banco de dados via tenant_id
- Row Level Security (RLS) no PostgreSQL
- Isolamento de filas por tenant
- Quotas por tenant (resources)

6.3 Resiliência

Padrões:

- Circuit Breaker (Resilience4j / Polly)
- Retry com Exponential Backoff
- Bulkhead (isolamento de recursos)
- Fallback strategies
- Health checks periódicos

---

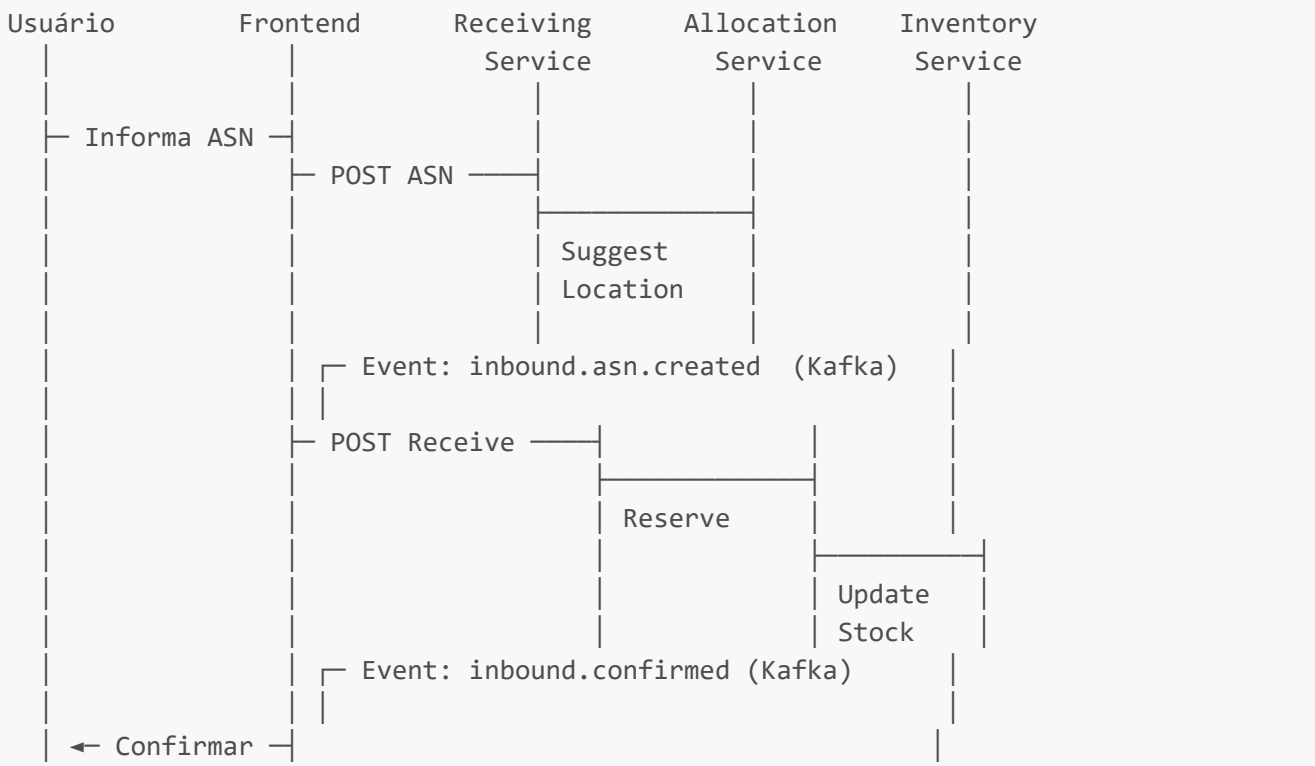
7. Tecnologia Stack Recomendado

Camada	Componente	Tecnologia	Alternativa
Backend	Core	Go / Rust	Python + FastAPI
	API Gateway	Kong / AWS API Gateway	NGINX + OpenResty
	Message Broker	Kafka	RabbitMQ / AWS SQS
Banco de Dados	OLTP	PostgreSQL	MySQL 8+
	Cache	Redis Cluster	Memcached
	Search	Elasticsearch	OpenSearch

Camada	Componente	Tecnologia	Alternativa
	Time Series	InfluxDB	Prometheus
Frontend	Web	React.js	Vue.js 3
	Mobile	React Native	Flutter
	UI Components	Material-UI	Ant Design
Infrastructure	Orquestração	Kubernetes	Docker Swarm
	Logs	ELK Stack	Grafana Loki
	Monitoring	Prometheus + Grafana	New Relic
	CI/CD	GitLab CI / GitHub Actions	Jenkins

---

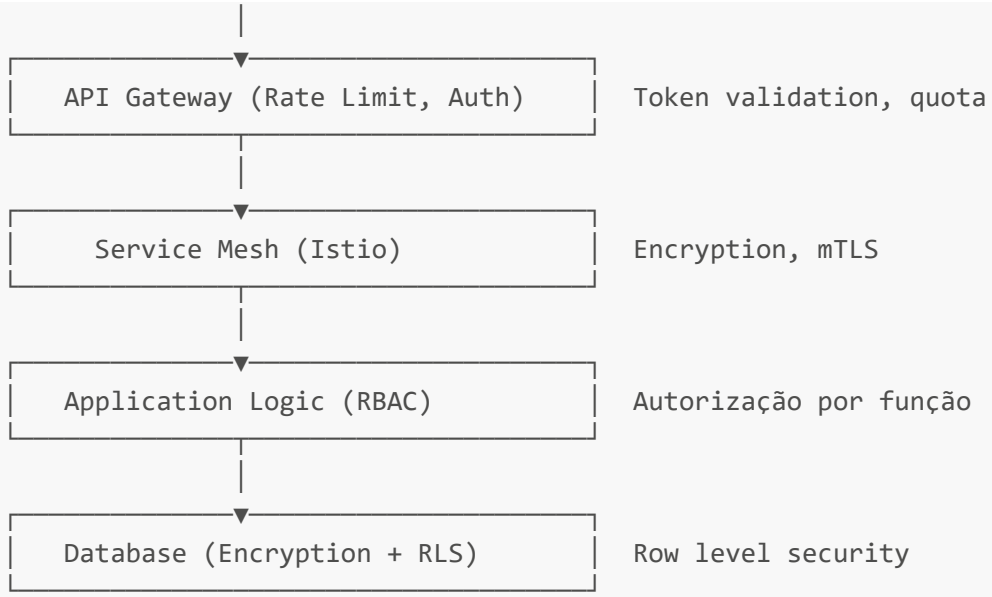
8. Diagrama de Sequência - Recebimento



---

9. Segurança Arquitetural

9.1 Defense in Depth



9.2 Encriptação

- **Em Trânsito:** TLS 1.3 obrigatório
- **Em Repouso:** AES-256 para dados sensíveis
- **Chaves:** Vault (HashiCorp Vault) para gerenciamento

---

10. Deployment & Infrastructure

10.1 Kubernetes Cluster

```
Namespaces:
  - wms-prod (production)
  - wms-staging (staging)
  - wms-dev (development)

Deployments:
  - API Services (replicas: 3+)
  - Message Broker (StatefulSet: 3 nodes)
  - Databases (StatefulSet: primary + replicas)
  - Cache (StatefulSet: cluster mode)
```

10.2 High Availability

- Multi-region deployment
- Auto-scaling baseado em CPU/memory
- Load balancing com health checks
- Disaster recovery com RTO < 1 hora

---

11. Performance Targets

Métrica	Alvo	Medição
API Response Time (P95)	< 500ms	APM (Application Performance Monitoring)
Database Query (P95)	< 100ms	Query execution time
Message Latency	< 100ms	Kafka broker lag
Cache Hit Rate	> 90%	Redis stats
System Availability	99.95%	Uptime monitoring
Throughput	50.000 tx/sec	Load testing
---	---	---

Documento Versão: 1.0
Status: Arquitetura Proposta
Próximo Passo: Validação com Tech Council