Documentação Arquitetural 101-Estoques (arc42)

Alunos: Alexandre G. Dominski, Henrique S. Etzel, Julio Cesar R. Batista, Victor Eugênio F. Rosa

Versão do template: 8.2 PT, Outubro de 2025

---

1. Introdução e Objetivos

1.1. Visão Geral

O **101-Estoques** é uma plataforma digital para gerenciar operações de locação de espaços físicos, intermediando locadores e locatários. Oferece funcionalidades de controle de estoque, gestão de contratos, processamento de pagamentos e relatórios.

Público-alvo:

- **Locadores:** Pessoas que desejam alugar seus espaços físicos
- **Locatários:** Pessoas que precisam de espaço físico adicional
- **Administradores:** Gestores da plataforma

1.2. Requisitos Principais

- Gerenciar locadores, locatários e espaços físicos
- Controle financeiro com múltiplas formas de pagamento
- Sistema de controle de estoque (Premium)
- Processamento assíncrono de eventos
- Relatórios operacionais e dashboards

1.3. Metas de Qualidade

Característica	Objetivo	Justificativa
Disponibilidade	99.5% uptime	Sistema crítico para operações diárias
Performance	< 2s para 95% das operações	Garantir produtividade
Segurança	Conformidade LGPD + JWT	Proteção de dados sensíveis
Escalabilidade	10.000+ usuários simultâneos	Crescimento do negócio
Manutenibilidade	Cobertura de testes > 50%	Facilitar evolução

1.4. Stakeholders

Função	Expectativas
Locador/Locatário	Facilitar processos de aluguel
Product Owner	Automatizar 80% dos processos, ROI em 12 meses
Arquiteto	Arquitetura escalável e modular
DevOps	Sistema estável com recuperação automática

---

2. Restrições Arquiteturais

2.1. Restrições Técnicas

Restrição	Descrição
Backend	Node.js 18+ com Express.js

Restrição	Descrição
Frontend	React + Vite + Material-UI
Bancos	MongoDB Atlas (Estoques), Azure SQL/SQLite (Aluguéis)
Cloud	Azure (Functions, SQL)
Containers	Docker
CI/CD	GitHub Actions

2.2. Restrições Organizacionais e Legais

- Desenvolvimento ágil (sprints de 2 semanas)
- Time de 4 desenvolvedores
- Conformidade LGPD, NF-e/NFSe, PCI-DSS
- MVP em 6 meses

---

3. Contexto e Escopo

3.1. Diagrama de Contexto (C4 - Nível 1)

```
graph TB
    Usuario["<b>Usuário</b><br/>[Person]<br/>Operador do sistema"]
    Admin["<b>Administrador</b><br/>[Person]<br/>Gestor da plataforma"]

    SofI["<b>101-Estoques</b><br/>[Software System]<br/>Gestão de estoques e aluguéis"]

    Frontend["<b>Microfrontend</b><br/>[External]<br/>Interface web"]
    MongoDB["<b>MongoDB Atlas</b><br/>[External]<br/>Banco NoSQL"]
    AzureSQL["<b>Azure SQL</b><br/>[External]<br/>Banco relacional"]
    Functions["<b>Azure Functions</b><br/>[External]<br/>Serverless"]
    Notifier["<b>SMTP</b><br/>[External]<br/>Notificações"]

    Usuario -->|Usa| Frontend
    Admin -->|Monitora| Frontend
    Frontend -->|APIs REST| SofI
    SofI -->|Persiste estoques| MongoDB
    SofI -->|Persiste aluguéis| AzureSQL
    SofI -->|Eventos| Functions
    SofI -->|Notifica| Notifier
    Functions -->|Lê eventos| MongoDB
```

3.2. Interfaces Externas

Canal	Entrada	Saída
Frontend	Cadastros, consultas, ações	Dashboards, relatórios, confirmações
Sistema Fiscal	Dados transacionais	NF-e, cálculos tributários
Gateway Pagamento	Dados de pagamento	Confirmação, status
Serviço Email/SMS	Disparos de notificação	Emails, SMS

---

4. Estratégia de Solução

4.1. Visão Geral

Arquitetura de **microserviços** com **BFF (Backend-for-Frontend)** como API Gateway, responsável por roteamento, agregação de dados e autenticação.

4.2. Componentes Principais

1. **microservice-estoques:** Gerenciamento de estoques (MongoDB)
2. **microservice-alugueis:** Gerenciamento de aluguéis (Azure SQL/SQLite)
3. **function-eventos:** Processamento assíncrono (Azure Functions)
4. **bff-node:** Agregação de APIs e proxy
5. **microfrontend:** Interface React responsiva

4.3. Padrões Arquiteturais

- **Microserviços:** Separação de responsabilidades
- **Event-Driven:** Processamento assíncrono
- **BFF:** Agregação de dados para frontend
- **REST APIs:** Comunicação HTTP/JSON

---

5. Visão de Blocos de Construção

5.1. Diagrama de Containers (C4 - Nível 2)

```
graph TB
    subgraph System["101-Estoques"]
        MFE["<b>Microfrontend</b><br/>React + Vite"]
        BFF["<b>BFF</b><br/>Node.js + Express"]
        MSE["<b>MS-Estoques</b><br/>Node.js + MongoDB"]
        MSA["<b>MS-Aluguéis</b><br/>Node.js + SQL"]
        FE["<b>Function-Eventos</b><br/>Azure Functions"]
    end

    MFE -->|API calls| BFF
    BFF -->|CRUD| MSE
    BFF -->|CRUD| MSA
    BFF -->|Events| FE
    FE -->|Updates| MSE
```

5.2. Componentes do Microservice Estoques (C4 - Nível 3)

```
graph TB
    subgraph MSEstoques["Microservice - Estoques"]
        API["API Controller<br/>Express Router"]
        Auth["Auth Middleware<br/>JWT Validation"]
        Valid["Validator<br/>Schema Validation"]
        Service["Estoque Service<br/>Business Logic"]
        Repo["Repository<br/>Data Access"]
        Event["Event Publisher<br/>Domain Events"]
        Error["Error Handler<br/>Centralized"]
    end

    API --> Auth
    API --> Valid
    API --> Service
    Service --> Repo
    Service --> Event
    API --> Error
```

Responsabilidades:

- **Controller:** Endpoints REST
- **Middleware:** Autenticação JWT
- **Validator:** Validação de schemas
- **Service:** Lógica de negócio
- **Repository:** Acesso a dados MongoDB
- **Event Publisher:** Eventos de domínio
- **Error Handler:** Tratamento centralizado

5.3. APIs REST Principais

BFF:

```
GET  /api/dashboard              # Dados agregados
GET  /api/estoque/{id}/detalhes  # Estoque + aluguéis
```

Microservice Estoques:

```
GET    /estoques      # Listar
POST   /estoques      # Criar
GET    /estoques/{id} # Buscar
PUT    /estoques/{id} # Atualizar
DELETE /estoques/{id} # Deletar
```

Microservice Aluguéis:

```
GET    /alugueis          # Listar (filtro por status)
POST   /alugueis          # Criar
GET    /alugueis/{id}     # Buscar
PUT    /alugueis/{id}     # Atualizar
DELETE /alugueis/{id}     # Cancelar
```

5.4. Schemas de Dados

Estoque (MongoDB):

```
{
  "_id": "ObjectId",
  "codigo": "string (obrigatório)",
  "nome": "string (obrigatório)",
  "tamanho_m2": "number (obrigatório)",
  "preco_mensal": "number (obrigatório)",
  "localizacao": "string (obrigatório)",
  "status": "enum [disponivel, ocupado, manutencao]",
  "created_at": "datetime"
}
```

Aluguel (Azure SQL):

```
{
  "id": "integer (PK)",
  "estoque_id": "string (FK)",
  "cliente_nome": "string (obrigatório)",
  "cliente_email": "string (obrigatório)",
  "data_inicio": "date (obrigatório)",
  "data_fim": "date (nullable)",
  "valor_mensal": "number (obrigatório)",
  "status": "enum [ativo, finalizado, cancelado]",
  "created_at": "datetime"
}
```

---

6. Visão de Tempo de Execução

6.1. Cenário: Criação de Aluguel

Fluxo:

1. Usuário autentica via JWT no frontend
2. Frontend → BFF: `POST /api/alugueis`
3. BFF valida token e roteia para MS-Alugueis
4. MS-Alugueis valida disponibilidade com MS-Estoques
5. MS-Alugueis cria registro e publica evento
6. BFF envia evento para Function-Eventos
7. Function atualiza status do estoque para "ocupado"
8. Function envia notificação por email

9. Frontend recebe confirmação

Aspectos:

- Comunicação assíncrona via eventos
- Consistência eventual
- Escalabilidade via Azure Functions

6.2. Cenário: Dashboard Agregado

Fluxo:

1. Frontend → BFF: `GET /api/dashboard`
2. BFF executa agregação paralela:
    - MS-Estoques: estoques disponíveis
    - MS-Alugueis: aluguéis ativos
3. BFF combina respostas
4. Frontend renderiza dashboard

---

7. Visão de Implantação

7.1. Infraestrutura Azure

Componente	Tecnologia	Ambiente
Frontend	React + Vite	Azure Static Web Apps (CDN)
BFF	Node.js	Azure App Service (Container)
MS-Estoques	Node.js	Azure App Service (Container)
MS-Aluguéis	Node.js	Azure App Service (Container)
Function-Eventos	Azure Functions	Serverless
MongoDB	MongoDB Atlas	Cloud gerenciado
Azure SQL	Azure SQL Database	Cloud gerenciado
CI/CD	GitHub Actions	Pipeline automatizado

7.2. Ambientes

Desenvolvimento:

- SQLite local, MongoDB local/Atlas free tier
- Docker Compose

Produção:

- Azure App Services (Premium)
- MongoDB Atlas (cluster replicado)
- Azure SQL (Standard/Premium)
- Auto-scaling habilitado

7.3. Pipeline CI/CD

```
1. Commit → GitHub
2. GitHub Actions:
   - Build containers
   - Testes (unit + integration)
   - Push to Azure Container Registry
3. Deploy Staging (automático)
4. Approval Gate
5. Deploy Production (manual)
```

8. Conceitos Transversais

8.1. Segurança

- **Autenticação:** JWT com expiração
- **Autorização:** RBAC por endpoint
- **Criptografia:** TLS/HTTPS, AES-256 em repouso
- **Conformidade:** LGPD, PCI-DSS
- **Secrets:** Azure Key Vault

8.2. Tratamento de Erros

- Middleware centralizado
- Códigos HTTP apropriados (400, 404, 500)
- Mensagens padronizadas
- Logs estruturados de exceções

8.3. Observabilidade

Ferramentas:

- Azure Application Insights (métricas, traces)
- Azure Monitor (infraestrutura, alertas)
- Azure Log Analytics (logs centralizados)

Métricas:

- Tempo de resposta < 2s
- Taxa de erro < 1%
- Disponibilidade > 99.5%
- CPU/Memória

8.4. Resiliência

- **Retry Pattern:** Tentativas automáticas
- **Circuit Breaker:** Interrupção de chamadas falhas
- **Timeout:** Limites de tempo
- **Fallback:** Respostas alternativas

---

9. Decisões Arquiteturais

9.1. Microserviços vs Monolito

Decisão: Microserviços

Justificativa:

- Escalabilidade independente
- Isolamento de falhas
- Tecnologias diferentes (MongoDB + SQL)

Trade-offs:

- Maior complexidade operacional
- Consistência eventual

9.2. MongoDB para Estoques

Decisão: MongoDB Atlas

Justificativa:

- Flexibilidade de schema
- Escalabilidade horizontal
- Managed service

9.3. Azure SQL para Aluguéis

Decisão: Azure SQL (SQLite em dev)

Justificativa:

- ACID para dados financeiros
- Integridade referencial
- Familiaridade com SQL

9.4. Azure Functions para Eventos

Decisão: Serverless

Justificativa:

- Escalabilidade automática
- Pay-per-execution
- Ideal para processamento assíncrono

9.5. BFF Pattern

Decisão: Backend-for-Frontend

Justificativa:

- Simplifica cliente
- Agrega múltiplos serviços
- Reduz chamadas do frontend

---

10. Requisitos de Qualidade

10.1. Árvore de Qualidade

```
Sistema 101-Estoques
├── Funcionalidade (CRUD estoques/aluguéis, eventos)
├── Confiabilidade (99.5% disponibilidade, retry, backup)
├── Usabilidade (interface intuitiva, < 30min aprendizado)
├── Eficiência (< 2s resposta, 10k usuários)
├── Manutenibilidade (modular, 50% cobertura testes)
├── Portabilidade (Docker, cloud-agnostic)
└── Segurança (JWT, RBAC, TLS, LGPD)
```

10.2. Cenários de Avaliação

Cenário	Medida	Resposta
Disponibilidade	Uptime 99.5%/mês	Auto-healing, replicas
Performance	95% req < 2s	Cache, otimização, scaling
Segurança	Auditoria trimestral	JWT, HTTPS, RBAC
Manutenibilidade	Nova feature < 2 sprints	Testes automatizados, CI/CD

---

11. Riscos e Débitos Técnicos

11.1. Riscos

Risco	Impacto	Prob.	Mitigação
Falha MongoDB	Alto	Baixo	Backup, replicação
Falha Azure SQL	Alto	Baixo	Failover automático
Sobrecarga eventos	Médio	Médio	Rate limiting, auto-scaling
Inconsistência entre serviços	Médio	Médio	Event sourcing, idempotência
Vulnerabilidade segurança	Alto	Baixo	Auditorias, updates
Vendor lock-in Azure	Médio	Médio	Design cloud-agnostic

Risco	Impacto	Prob.	Mitigação

11.2. Débitos Técnicos

Débito	Impacto	Plano
Ausência testes E2E	Médio	Implementar Cypress
Cobertura < 50%	Médio	Meta 70% em 3 meses
Falta circuit breaker	Alto	Implementar em 2 sprints
Secrets hardcoded	Alto	Migrar para Key Vault

---

12. Glossário

Termo	Definição
101-Estoques	Plataforma de gestão de estoques e aluguéis
Locador	Proprietário de espaços físicos
Locatário	Pessoa que aluga espaços
BFF	Backend-for-Frontend
JWT	JSON Web Token
RBAC	Role-Based Access Control
Event-Driven	Arquitetura baseada em eventos
LGPD	Lei Geral de Proteção de Dados
Circuit Breaker	Padrão de interrupção de chamadas falhas
Idempotência	Operação com mesmo resultado em múltiplas execuções

---

Referências

- **arc42:** https://arc42.org
- **C4 Model:** https://c4model.com
- **Swagger/OpenAPI:** Documentação completa em `swagger.yaml`
- **Microservices Patterns:** https://microservices.io/patterns/
- **Azure Docs:** https://docs.microsoft.com/azure/

Versão: 2.0
Data: Outubro 2025
Status: Aprovado