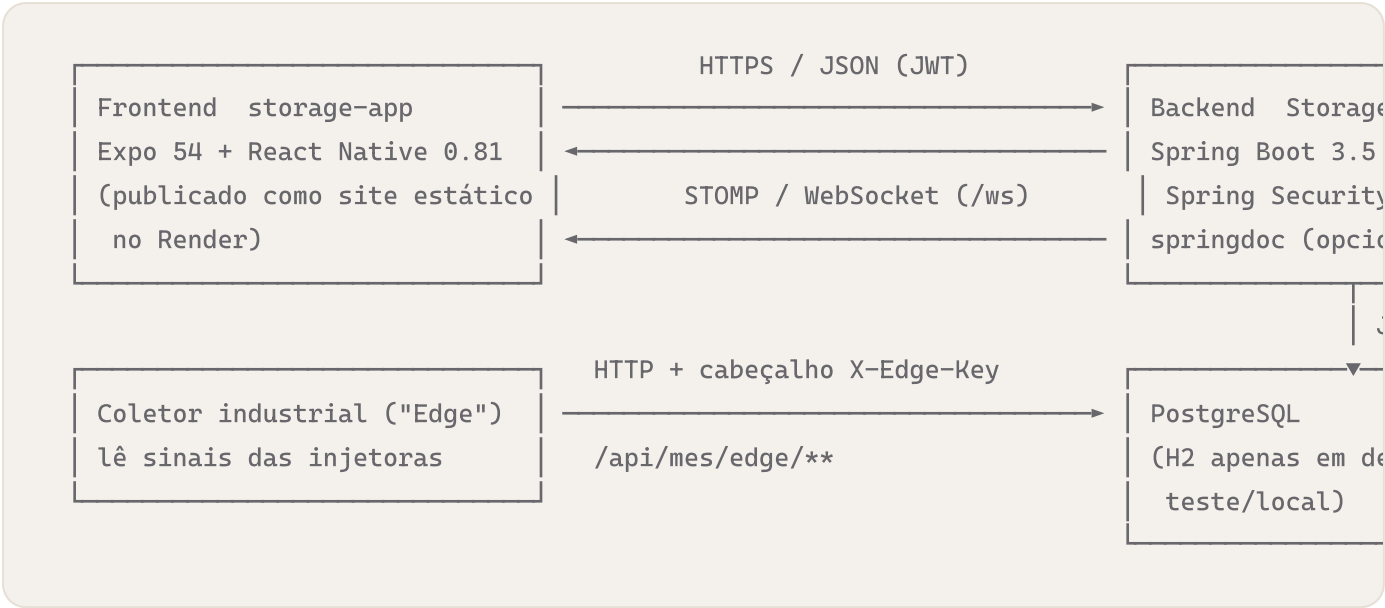


Arquitetura

← Índice da documentação da API

Escopo: o que está comprovado no código dos dois projetos — frontend `storage-app` (Expo / React Native Web) e backend `Storage_Wester` (Spring Boot).

1. Visão geral



CAMADA	TECNOLOGIA (COMPROVADA EM <code>POM.XML</code> / <code>PACKAGE.JSON</code>)
Frontend	Expo <code>^54</code> , React <code>19.1</code> , React Native <code>0.81</code> , React Navigation (Drawer), React Native Paper, Axios, <code>@stomp/stompjs</code> , Three.js + <code>@react-three/fiber</code> (visão 3D), <code>@dnd-kit</code> (arrastar e soltar)
Backend	Spring Boot <code>3.5.0</code> , Java <code>21</code> , Spring Web, Spring Security (JWT com <code>jjwt</code>), Spring Data JPA, Bean Validation, Spring WebSocket (STOMP), Actuator, springdoc-openapi, Apache PDFBox <code>3.0.3</code>
Banco	PostgreSQL (driver e dialeto <code>PostgreSQLDialect</code>), migrações Flyway em <code>src/main/resources/db/migration</code>
Implantação	Frontend: <code>render.yaml</code> (site estático, <code>npm run build:web</code> , reescreve <code>/*</code> para <code>/index.html</code>); backend: <code>Dockerfile</code> e <code>Dockerfile.hardened</code> , <code>docker-compose.yaml</code>

2. Backend: camadas e módulos

O backend segue o padrão **controller** → **service** → **repository** → **entidade**, com DTOs de entrada/saída em `dto/`:

PACOTE	CONTEÚDO
<code>controller</code>	21 controllers REST + <code>exception/GlobalExceptionHandler</code>
<code>mes</code>	módulo de produção (MES/PCP): <code>MesController</code> , <code>MesEdgeController</code> , <code>MesService</code> , <code>MesRules</code> , regras de acesso, leitura de PDF (<code>MesOrderPdfParser</code> , <code>MesOrderPdfOcr</code>)
<code>service</code>	24 classes de serviço e apoio com as regras de negócio
<code>repository</code>	repositórios JPA (alguns com consultas customizadas)
<code>model</code>	17 entidades JPA e 3 enums (<code>MaterialType</code> , <code>TipoAcessoPerfil</code> , <code>TipoEstruturaArea</code>)
<code>config</code>	segurança, JWT, CORS, limite de taxa, OpenAPI
<code>realtime</code>	WebSocket/STOMP, publicação de eventos após commit

2.1 Módulos funcionais da API

A API foi agrupada em **16 módulos** (ver endpoints.md):

1. Acesso, usuários e perfis
2. Produtos
3. Categorias de produtos
4. Setores e tipos de material
5. Corredores, blocos e níveis (estrutura física)
6. Itens e transferências de estoque
7. Moldes
8. Materiais fixos e slots legados
9. Listas de materiais ("encerto")
10. Consultas de estoque e mapa

- .1. Histórico e auditoria
- .2. Movimentações operacionais
- .3. Alertas de estoque
- .4. Relatórios de estoque
- .5. PCP e produção (MES)
- .6. Integração industrial Edge

3. Modelo de dados e relacionamentos

Os campos de cada DTO estão em modelos.md. A relação entre as entidades persistidas é:

```

AreaEstoque (setor) -1—N- Fileira (corredor/estante) -1—N- Grade (bloco) -1—N- Nivel (
    |
    | 1—N
    ▼
AreaMaterialTag (tipos de material aceitos no setor: MOLDE | PRODUTO | FERRAMENTA)
AreaEstoque -1—N- TabelaEncerto -1—N- TabelaEncertoItem -N—1- Produto (list
HistoricoMovimentacao — referência opcional a Usuario, Produto, ItemEstoque, Nivel (atu
                        e ids avulsos (área, corredor, bloco, slot, molde, entidade)
Usuario —(texto `perfil` = `Perfil.codigo`, sem chave estrangeira)— Perfil

Módulo MES (tabelas criadas em V10): máquina · ordem de serviço · execução · evento · c
apontamento de qualidade · parada
  
```

Observações confirmadas no código:

- **Dois tipos de setor** (`TipoEstruturaArea`): `DINAMICO` (níveis podem ser criados, movidos e resequenciados) e `FIXO` (posições criadas na configuração e editadas pontualmente). O frontend mostra uma tela diferente para cada um (`Warehouse2DView` para dinâmico, `WarehouseFixedView` para fixo) e uma visão 3D para os dois.
- **Rótulos configuráveis por setor**: `labelEstruturaPrimaria` , `labelEstruturaSecundaria` , `labelNivel` e `labelSlot` — por padrão o frontend usa "Corredor", "Bloco" e "Ordem/Gaiola"; nos setores fixos podem aparecer "Estante", "Posição" etc.
- **Controle de concorrência otimista** (`@Version`) em `Fileira` , `Grade` , `Nivel` e `ItemEstoque` .

- **Perfil do usuário não é chave estrangeira:** o usuário guarda o código do perfil como texto; se o perfil for inativado ou apagado, o usuário fica sem permissões (ver `autenticacao.md`).
 - **Auditoria:** o `HistoricoMovimentacao` é gravado na **mesma transação** da alteração de negócio (exigido por `Propagation.MANDATORY` em `registrarEventoGenericoNaTransacaoAtual`): se a auditoria falhar, a mudança é desfeita.
 - **Migrações Flyway:** 19 arquivos `V*__*.sql` (V1–V16, V21–V23; não existem V17–V20 no repositório). `ddl-auto` é `update` no perfil padrão e `validate` no perfil `prod`.
-

4. Tempo real

`WebSocketConfig` publica STOMP em `/ws` (SockJS) e `/ws-native`, com corretor simples em `/topic` e prefixo de aplicação `/app`. `EntityChangeListener` publica um `RealtimeEvent` **depois do commit** para as entidades que alteram estoque, estrutura, histórico, cadastro etc. Tópicos: `/topic/estoque`, `/topic/area/{areaId}` e `/topic/entidade/{entidade}` (detalhes em `integracoes.md`). O frontend usa isso para atualizar listas e mapa sem recarregar a página.

5. Frontend: organização

PASTA (<code>SRC/</code>)	FUNÇÃO
<code>navigation/</code>	menu lateral (Drawer), rotas e <i>deep links</i> (<code>linking.ts</code>)
<code>auth/</code>	contexto de autenticação (<code>AuthContext</code>), proteção de rotas (<code>RequireAuth</code>), retorno pós-login
<code>security/permissions.ts</code> <code>x</code>	mapeia o perfil para telas e ações; mostra "Sem permissão" quando não liberado
<code>screens/</code>	telas: login/cadastro, Dashboard, Armazém, Setores, Produtos, Moldes, Perfis, Histórico, Usuários, PCP (<code>mes/</code>) e as telas desativadas Alertas, Movimentações, Relatórios
<code>components/</code> , <code>components/warehouse2d</code> <code>/</code>	componentes do armazém 2D, setor fixo, mapa V4, modais
<code>twin/</code>	"Wester Twin" — visão 3D do armazém
<code>services/</code>	clientes HTTP por módulo; <code>services/http/</code> trata erros
<code>realtime/</code> , <code>services/realtime/</code>	assinatura dos tópicos STOMP
<code>axios.ts</code> , <code>config/apiConfig.ts</code>	cliente HTTP, anexo do token e URL da API

- **URL da API:** `EXPO_PUBLIC_API_BASE_URL` (ou `VITE_API_BASE_URL`); se ausente, `http://localhost:8080 / 127.0.0.1` quando a página roda em localhost, senão a URL de produção definida em `apiConfig.ts` . Barras finais e um sufixo `/api` são normalizados.
- **Sessão:** token e dados do usuário ficam em `AsyncStorage` (no navegador, armazenamento local). Um `401` com token anexado limpa a sessão e leva ao Login.
- **Temas:** Light, Dark e Console (seletor no cabeçalho de toda tela).
- **3D:** disponível apenas no navegador (WebGL); a opção não é oferecida em build nativo.

6. Pontos de extensão e restrições conhecidas

- A API tem **alias** duplicados em português e inglês (`/api/areas` = `/api/setores` , `/api/produtos` = `/api/products` ...). Eles contam como rotas, mas são a mesma operação (211 combinações método+rota para 145 operações).
- Alertas, Movimentações e Relatórios têm **backend e código de tela prontos, mas estão desativados no menu atual** (comentário em `DrawerNavigator.tsx`). Ver documento interno de lacunas.
- Swagger/OpenAPI da aplicação só liga com `SWAGGER_ENABLED=true` ; o `openapi.yaml` deste diretório é uma análise estática independente, com exemplos sintéticos.