Visão geral

- Velocidade no MVP: um único repositório facilita subir o pipeline ponta-a-ponta com `docker-compose` (web + api + worker + db + minio).
- Tipos e esquemas compartilhados: o projeto usa um BPMN_JSON interno (edição robusta) que depois é convertido para XML; manter esse schema em um pacote compartilhado evita divergências entre Web/Backend/Worker.
- UX/IA acopladas ao editor: a sincronização Editor↔IA (eventBus/commandStack.changed), linter BPMN e auto-layout ELK.js ficam mais fáceis de versionar juntos.

Árvore de diretórios (proposta inicial)

```
bpmappr/
├─ apps/
│  ├─ web/                                  # Next.js + TS + Tailwind +
shadcn/ui + bpmn-js
│  │  ├─ src/
│  │  │  ├─ app/                             # Rotas/Layouts do Next.js
(App Router)
│  │  │  ├─ features/
│  │  │  │  ├─ bpmn/
│  │  │  │  │  ├─ editor/                       # Integração bpmn-js
(modeler/viewer, context pad, palette)
│  │  │  │  │  ├─ layout/                        # Adaptador ELK.js
(auto-layout, pins, reaplicar layout)
│  │  │  │  │  ├─ linting/                    # bpmnlint + regras mínimas
(no-disconnected, etc.)
│  │  │  │  │  ├─ io/                          # Import/Export (.bpmn,
.png/.svg)
│  │  │  │  │  └─ sync/                          # Ponte Editor↔IA
(EventBus/commandStack.changed)
│  │  │  │  ├─ copiloto/                        # Chat + tool-calls
aplicados ao canvas
│  │  │  │  └─ citations/                     # Painel de evidências
(trechos PDF/imagens)
│  │  │  ├─ lib/
│  │  │  │  ├─ byok/                          # BYOK no navegador
(armazenar chaves localmente)
│  │  │  │  ├─ api-client/                    # Cliente TS gerado do
OpenAPI do FastAPI
│  │  │  │  └─ i18n/                         # PT-BR/EN (mínimo)
```

```
|  |  |  ├─ components/                    # shadcn/ui + componentes
do Studio
|  |  |  └─ styles/
|  |  ├─ public/
|  |  └─ tests/                                  # Playwright E2E
(drag-and-drop, import/export)
|  |
|  ├─ api/                                 # FastAPI (REST) + Alembic
+ serviços
|  |  ├─ app/
|  |  |  ├─ main.py
|  |  |  ├─ api/
|  |  |  |  └─ v1/
|  |  |  |     ├─ ingest.py              # POST /ingest
|  |  |  |     ├─ generate.py            # POST /generate
|  |  |  |     ├─ edit.py                # POST /edit
|  |  |  |     └─ export.py              # GET /export
|  |  |  ├─ core/                          # config, segurança, rate
limiting, logging/OTel
|  |  |  ├─ db/                                  # models, schemas,
migrations (Alembic)
|  |  |  ├─ services/
|  |  |  |  ├─ ingestion/                                  #
Docling/Unstructured/PyMuPDF + OCR + normalização
|  |  |  |  ├─ rag/                           # Indexador/Retriever
(pgvector), citações
|  |  |  |  ├─ agents/                     # LangGraph (síntese→JSON,
linter, layout, supervisor)
|  |  |  |  ├─ bpmn/                        # Conversores JSON↔XML,
bpmnlint server-side, GED/RGED
|  |  |  |  ├─ versioning/                  # Versionamento, diff
visual (nós/arestas), trilhas
|  |  |  |  └─ storage/                     # MinIO/S3, uploads e
exports
|  |  |  └─ worker/                         # Tarefas assíncronas
(Celery/RQ) e filas
|  |  └─ tests/                                          # Pytest
(unitário/integrado)
|  |
|  └─ eval/                                # Harness de avaliação
(GED/RGED) e relatórios
```

```
|     ├─ datasets/
|     ├─ runners/
|     └─ reports/
|
├─ packages/
|   ├─ shared-schemas/                    # BPMN_JSON schema + tipos
comuns (TS/Pydantic)
|   ├─ ui-kit/                                  # Design system
compartilhado (shadcn/ui wrappers)
|   ├─ prompts/                             # Prompts base (síntese,
linter, supervisor)
|   ├─ clients/                                 # SDKs: /python e
/typescript (consumo da API)
|   └─ telemetry/                              # Utilitários de
logs/OTel/metrics compartilhados
|
├─ infra/
|   ├─ compose/
|   |   ├─ docker-compose.yml                 # api, web, worker,
postgres, minio
|   |   ├─ postgres/                        # init.sql, configurações
|   |   └─ minio/                             # políticas/buckets
(artifacts, exports)
|   ├─ k8s/                                 # (futuro) manifests/helm
charts
|   └─ ci/                                      # pipelines (GitHub
Actions), segurança, lint
|
├─ data/                                    # Amostras locais p/ dev
(PDFs, imagens, .bpmn)
├─ docs/
|   ├─ PRD.md                                 # Cópia em texto do PRD
para consulta
|   ├─ ADRs/                                # Decisões arquiteturais
|   ├─ API/openapi.yaml                       # Esquema da API (gera
clients)
|   ├─ guides/ (setup, BYOK, segurança, avaliação)
|   └─ rules.md                               # Regras operacionais do
anexo
|
```

```
├─ scripts/                                  # Utilitários: seed,
migrations, jobs locais
├─ .env.example                             # Variáveis de ambiente
(NUNCA subir segredos)
├─ Makefile                                  # Tarefas comuns (dev,
test, lint, compose up)
└─ README.md
```

O propósito de cada bloco — guia didático

`apps/web/` — Studio + Copiloto com bpmn-js

Onde vive o editor visual inteligente (drag-and-drop, palette, context pad, properties panel), o copiloto com comandos naturais e o painel de citações.

- `features/bpmn/editor/`: embed do bpmn-js (modeler/viewer), atalhos (N/A), undo/redo, seleção múltipla, properties panel, minimap. Mapeia diretamente os RF-D e critérios de aceite do editor.
- `features/bpmn/layout/`: integração com ELK.js (layered, lanes/pools, message flows), com suporte a pins e reaplicar layout.
  `features/bpmn/linting/`: configuração bpmnlint (start-event-required, no-disconnected, no-complex-gateway) e avisos em tempo real.
- `features/bpmn/io/`: importar/exportar BPMN XML + PNG/SVG preservando layout pós-edição.
- `features/bpmn/sync/`: ponte Editor↔IA; a cada edição no canvas, salva XML, atualiza o BPMN_JSON "vivo" e versiona.
- `features/citactions/`: exibe trechos de PDF/frames que deram origem a cada elemento (traçabilidade).
  `lib/byok/`: BYOK no navegador (armazenamento de chaves, nunca no servidor).
- Stack: Next.js + TypeScript + Tailwind + shadcn/ui, conforme PRD.

`apps/api/` — FastAPI + serviços de domínio

Implementa o diagrama do PRD: REST → ingestão → vetores (pgvector) → multiagente (LangGraph) → layout → retorno do XML e persistência (modelos/versões/artefatos/métricas).

- `api/v1/*.py`: endpoints do rascunho (`/ingest`, `/generate`, `/edit`, `/export`).
  `core/`: config (env), rate limiting para rotas públicas (copiloto), logs estruturados e tracing (OpenTelemetry), seguindo as notas de segurança.
- `db/`: modelos para ProcessModel, ModelVersion, Artifact, EmbeddingChunk e AuditEntry + migrations Alembic.
- `services/ingestion/`: wrappers de Docling/Unstructured/PyMuPDF + OCR e normalização multimodal (texto limpo + metadados + referências).

- `services/rag/`: indexação/consulta semântica em PostgreSQL + pgvector, com filtros por tipo e citações (arquivo/página).

- `services/agents/`: LangGraph com agentes de síntese BPMN (JSON), linter/validador, layout/visualização e supervisor (human-in-the-loop).
- `services/bpmn/`: conversores JSON↔XML (edita no JSON e converte no final), linter server-side e métricas GED/RGED.
- `services/versioning/`: versionamento "tipo git" + diff visual por nós/arestas + trilhas de prompts/ações.
  `services/storage/`: integração MinIO/S3 para uploads (artefatos) e exports (bpmn/png/json).
- `worker/`: filas e tarefas (ex.: "gerar de artifacts", "aplicar edição do copiloto", "reaplicar layout"), para cumprir P95 ≤ 60 s.

`apps/eval/` — Avaliação & métricas embutidas

Harness para **GED/RGED**, testes de edição e **telemetria** de custo/latência, como pede o PRD (KPIs e Epic F).

`packages/` — Reaproveitamento sem acoplamento

- `shared-schemas/`: o schema do BPMN_JSON (JSON Schema + Pydantic + tipos TS). É o coração da edição robusta citada no PRD.
- `ui-kit/`: componentes baseados em shadcn/ui para Web e páginas administrativas.
- `prompts/`: prompts base do Agente de Síntese/Linter/Supervisor (fáceis de versionar e testar).
- `clients/`: SDKs (TS/Python) gerados a partir do OpenAPI — simplifica automações e testes.
- `telemetry/`: convenções de logs/metrics/tracing compartilhadas entre API/Worker/Web.

`infra/` — Execução e operação

- `compose/docker-compose.yml`: sobe Next.js, FastAPI, Worker, PostgreSQL+pgvector e MinIO; combina BYOK no front com serviços no back.
- `k8s/` (futuro): manifestos/Helm, caso saia de Compose.
- `ci/`: pipelines (lint/test/build), scans e políticas de segurança.

`docs/` — Fonte única de verdade

- `PRD.md`: cópia textual do PRD para consulta no repo.
- `rules.md`: regras operacionais do anexo (ex.: "NUNCA logar segredos", "rate limit em rotas públicas").
- `guides/`: BYOK, segurança/privacidade, fluxo de layout, avaliação.
- code_architecture: arquitetura de pastas e diretórios do código

Como isso cobre os EPICs do PRD

- EPIC A — Ingestão Multimodal → `apps/api/services/ingestion/` + `apps/api/api/v1/ingest.py` (upload PDF/DOCX/TXT/PNG/JPG, parsing Docling/Unstructured/PyMuPDF, normalização em Artifact).
- EPIC B — RAG Corporativo → `apps/api/services/rag/` (indexação/consulta em pgvector com citações exibidas no Studio).
- EPIC C — Orquestração Multiagente (LangGraph) → `apps/api/services/agents/` (síntese BPMN_JSON, linter/validador, layout, supervisor; estados persistidos e HIL).
- EPIC D — Editor Visual Inteligente (bpmn-js) → `apps/web/features/bpmn/*` + `features/citactions/` + `features/copiloto/` (drag-and-drop, comandos naturais, linting e import/export).
- EPIC E — Versionamento & Auditoria → `apps/api/services/versioning/` + UI de Diff no Studio; guarda ModelVersion e AuditEntry.
- EPIC F — Avaliação & Métricas → `apps/eval/` (runner GED/RGED, testes, telemetria).
- EPIC G — Entrega & Operação → `infra/compose/` (BYOK no front, Docker Compose, observabilidade/logs/alerts de custo).

Fluxo ponta-a-ponta (onde cada etapa mora)

1. Upload (PDF/DOCX/TXT/Imagem) → `apps/web` chama `POST /ingest` → `apps/api/services/ingestion` salva Artifact (MinIO) e metadados.
2. Indexação → worker roda `services/rag/indexer.py` e grava EmbeddingChunk em Postgres+pgvector.
3. Geração → `POST /generate` aciona LangGraph (síntese BPMN_JSON → conversão para BPMN XML → layout ELK.js) e cria ModelVersion.
4. Edição guiada → Studio (bpmn-js) + copiloto (tool-calls); sincroniza XML↔JSON, valida com bpmnlint e atualiza versão.
5. Exportação → `GET /export` retorna `.bpmn` + `.png` + `.json`.
6. Avaliação → `apps/eval` calcula GED/RGED, alimenta painel de qualidade e auditoria.

Convenções e detalhes úteis

- BYOK no navegador (`apps/web/src/lib/byok/`): documentação de uso (docs/guides/byok.md). O back-end nunca persiste chaves, conforme o PRD.
- Segurança (`apps/api/app/core/`): TLS, mascaramento leve de PII, rate limiting nas rotas do copiloto e logs sem segredos (reforçado em `docs/rules.md`).
- Dados (`apps/api/app/db/`): modelos exatamente como o Modelo de Dados essencial (ProcessModel, ModelVersion, Artifact, EmbeddingChunk, AuditEntry) + migrações Alembic.

- Linter BPMN (`apps/web/features/bpmn/linting/` e `apps/api/services/bpmn/`): regras base e erros críticos (ex.: diagrama desconectado).
- Layout (`apps/web/features/bpmn/layout/`): ELK.js com fallback e pins para respeitar ajustes manuais; considerar cache por hash do grafo.
- Observabilidade (`packages/telemetry/` + `infra/compose/`): logs estruturados, tracing e métricas de custo/latência — atendendo KPIs e "G".
- i18n e A11y: suporte PT-BR/EN, contraste AA, atalhos e tooltips descritivas, como requisitado.

Mapeamento rápido (PRD → pastas)

Item do PRD	**Pasta(s) chave**
Next.js + TS + Tailwind + shadcn/ui	`apps/web/` + `packages/ui-kit/`
bpmn-js + properties/linting + import/export	`apps/web/features/bpmn/*`
ELK.js (layout em pools/lanes)	`apps/web/features/bpmn/layout/`
FastAPI + Worker + Postgres+pgvector + MinIO	`apps/api/*` + `infra/compose/`
LangGraph (multiagente)	`apps/api/services/agents/`
JSON interno ↔ BPMN XML	`packages/shared-schemas/` + `apps/api/services/bpmn/`
Versionamento + Diff + Auditoria	`apps/api/services/versioning/` + UI em `apps/web`
Avaliação (GED/RGED)	`apps/eval/` + `apps/api/services/bpmn/metrics/`
BYOK	`apps/web/src/lib/byok/` + `docs/guides/byok.md`

Próximos passos práticos (em cima dessa estrutura)

1. Definir o `BPMN_JSON schema` em `packages/shared-schemas/` e gerar tipos TS/Pydantic (base do fluxo JSON→XML).
2. Publicar OpenAPI em `docs/API/openapi.yaml` para gerar `packages/clients/`.

Subir Compose de `infra/compose/` (api, web, worker, postgres/pgvector, minio) e validar o fluxo “Do arquivo ao BPMN” com um PDF de exemplo.

3. Configurar bpmnlint + ELK.js no Studio e a ponte Editor↔IA (salvar XML → atualizar JSON “vivo” → versionar).

Objetivo documento - Em uma frase

Esta arquitetura deixa cada responsabilidade no seu lugar, mas compartilha o que é crítico (schemas/prompts/telemetria) para acelerar o MVP descrito no PRD — da ingestão multimodal e RAG ao editor inteligente, versionamento e avaliação.
Se quiser, eu já posso adaptar esse esqueleto ao gerenciador de pacotes/stack que você preferir (pnpm/Yarn/Turborepo, Celery ou RQ, etc.).