Documento de Projeto de Sistema

Bookd

Vitória, ES

2026

Contribuíram para esta especificação:

Nome	Contato	Contribuições
Ricardo Magalhães Santos Filho	ricardo.santos.48@edu.ufes.br	Escrita do documento.

Registro de alterações:

<https://github.com/rickymagal/Bookd>

1 Introdução

Este documento apresenta o projeto (*design*) do sistema *Bookd*.

O escopo funcional abrange a construção e manutenção de uma biblioteca pessoal a partir de diferentes provedores de metadados, a organização por tags com uma camada híbrida (tags do usuário e um conjunto de sugestões "top 7", incluindo a marcação especial de "clássico"), a produção de anotações e destaques com anexos de mídia cuja transmissão é resiliente (fila local, retomada e confirmação), o acompanhamento de metas por período e por tag (livros, páginas e *streak*), além de métricas agregadas, retrospectivas e exportação em formatos abertos. As recomendações expõem sinais e razões legíveis, permitindo que o usuário compreenda por que determinado título foi sugerido.

Do ponto de vista arquitetural, o projeto adota armazenamento local e sincronização eventual com versionamento por entidade e políticas de resolução de conflitos por campo, recorrendo a estratégias de união para coleções quando aplicável. A integração com provedores externos de metadados e mídia é modularizada por *drivers* com *fallback*, limites de taxa e mecanismos de contenção de falhas. A privacidade é tratada como padrão do sistema, com criptografia em trânsito e opção de criptografia local para conteúdos sensíveis. Observabilidade e testabilidade são contempladas por telemetria focada nos fluxos de sincronização e envio de mídia, além de testes de consistência e degradação controlada. O alvo inicial é Android, com camadas de acesso a dados e casos de uso isolando a interface das políticas de persistência e rede.

Além desta introdução, este documento está organizado da seguinte forma: a Seção 2 apresenta a plataforma de software utilizada na implementação do sistema; a Seção 3 apresenta a especificação dos requisitos não funcionais (atributos de qualidade), definindo as táticas e o tratamento a serem dados aos atributos de qualidade considerados condutores da arquitetura; a Seção 4 apresenta a arquitetura de software; por fim, a Seção 5 apresenta o projeto dos componentes da arquitetura.

Este documento foi produzido por:

- Ricardo Magalhães Santos Filho

2 Plataforma de Desenvolvimento

A plataforma adota JavaScript no cliente e no servidor, com *app* móvel em React Native (foco inicial em Android) operando *offline-first* sobre SQLite e camada de sincronização por API REST em Node.js/Express com PostgreSQL no servidor. A recomendação é implementada como microserviço independente em Python (FastAPI) utilizando rede neural em PyTorch, expondo endpoints internos para scoring e explicabilidade. Armazenamento de mídia é feito em serviço S3-compatível (MinIO em desenvolvimento e S3 em produção), com *uploads* autenticados por *pre-signed URLs*. Cache e filas assíncronas (envio de mídia, reconciliação) utilizam Redis.

Na Tabela 1 são listadas as tecnologias utilizadas no desenvolvimento da ferramenta, bem como o propósito de sua utilização.

Tabela 1 – Plataforma de Desenvolvimento e Tecnologias Utilizadas.

Tecnologia	**Versão**	**Descrição**	**Propósito**
React Native (Expo)	0.76.x / SDK	Framework móvel em JavaScript	Implementação do app Android, UI e integrações nativas.
React Navigation	6.x	Biblioteca de navegação	Fluxos de telas, pilhas e abas.
Zustand	4.x	Gerenciador de estado leve	Estado local previsível e simples.
Axios	1.x	Cliente HTTP	Consumo da API REST e do microserviço de IA.
WatermelonDB	0.26+	Banco local *local-first* sobre SQLite	Persistência offline e sincronização incremental.
SQLite (expo-sqlite)	3.x	Banco de dados embarcado	Armazenamento local de entidades e filas.
Zod	3.x	Validação de esquemas	Validação de dados na borda do app.
Node.js	22 LTS	Runtime JavaScript no servidor	Backend REST, *tooling* e scripts.
Express.js	4.x	*Framework* web minimalista	Exposição de API REST, autenticação e *routing*.
Sequelize	6.x	ORM para Node.js	Modelagem, migrações e acesso ao PostgreSQL.
PostgreSQL	16.x	Banco de dados relacional	Persistência servidor; busca textual (`pg_trgm`).
Redis	7.x	Armazenamento em memória	Cache e filas assíncronas (uploads/sync).
MinIO (S3)	2025.x	Armazenamento de objetos	Mídia via *pre-signed URLs* (dev); S3 em produção.
OpenAPI (Swagger)	3.1	Especificação de APIs	Contrato da API e documentação interativa.
JWT	2.x	Padrão de tokens	Autenticação e autorização.
Nginx	1.26.x	Servidor HTTP / *proxy*	TLS, *rate limiting* e roteamento.

Tecnologia	Versão	Descrição	Propósito
Python	3.11	Linguagem para IA	Serviço de recomendação (rede neural) e pré-processamento.
FastAPI	0.11x	*Framework* ASGI	Microserviço de recomendação com baixo acoplamento.
PyTorch	2.x	*Framework* de *DL*	Treinamento/inferência do modelo de recomendação.
Pandas	2.x	Biblioteca de dados	Engenharia de atributos e análises.
Uvicorn	0.x	Servidor ASGI	Execução do microserviço FastAPI.
Jest	29.x	*Test runner* JavaScript	Testes unitários no cliente e servidor.
Supertest	6.x	Testes de API	Testes de integração REST.

Na Tabela 2 vemos os softwares que apoiaram o desenvolvimento de documentos e também do código-fonte.

Tabela 2 – Softwares de Apoio ao Desenvolvimento do Projeto

Tecnologia	Versão	Descrição	Propósito
Git	2.x	Sistema de controle de versão	Versionamento, *branching* e revisão de código.
GitHub	2025.x	Hospedagem de repositórios	Issues, PRs e CI/CD (Actions).
VS Code	1.x	Editor de código	Desenvolvimento do cliente/servidor e depuração.
Android Studio	2025.x	IDE Android	*Build*, emulação e perfilamento do app.
Postman	11.x	Ferramenta de APIs	Teste e documentação dos endpoints REST.
DBeaver	24.x	Cliente de banco de dados	Inspeção do PostgreSQL e execução de consultas.
Docker / Compose	26.x/2.x	Contêineres e orquestração local	Padronização de ambiente e serviços (DB, IA).
LaTeX (TeX Live)	2024/25	Sistema de composição tipográfica	Produção dos documentos técnicos.
diagrams.net	24.x	Ferramenta de diagramas	Modelagem C4 e fluxos arquiteturais.
Figma	2025.x	Ferramenta de design	Protótipos de UI (apoio ao capítulo de Interface).

3 Requisitos Não Funcionais

A Tabela 3 apresenta a especificação dos requisitos não funcionais identificados no Documento de Especificação de Requisitos, considerados condutores da arquitetura.

Tabela 3 – Especificação de Requisitos Não Funcionais.

RNF-1 – Operação *offline-first* com sincronização resiliente de dados do usuário.	
Categoria:	Disponibilidade e Confiabilidade.
Tática / Tratamento:	Persistência local (SQLite/WatermelonDB) com fila de mudanças (*write-ahead log*) que sobrevive a reinícios; sincronização em *background* com *retry* exponencial e detecção de duplicidade via IDs idempotentes; reconciliação por campo (*field-wise LWW*) e união para coleções; APIs idempotentes no servidor; telemetria de estados (queued/-running/success/error).
Medida:	Após retorno da conectividade, 95% das mudanças locais sincronizam em até 60s (p95) e 99% em até 5min (p99); zero perda de dados em interrupções simuladas de energia/rede durante criação/edição.
Critério de Aceitação:	Em teste automatizado com 1.000 operações offline ao longo de 24h e três falhas de rede controladas, o sistema deve: (i) restaurar a fila após reinício, (ii) concluir a sincronização com taxa de sucesso $\geq$ 99%, (iii) não apresentar registros corrompidos ou duplicados.

RNF-2 – Privacidade por padrão e controle de visibilidade por campo.	
Categoria:	Segurança e Usabilidade.
Tática / Tratamento:	*Privacy by default* (opt-in para publicação); escopo de visibilidade por entidade/campo; autenticação por JWT e trânsito protegido (TLS 1.3); minimização de dados; opção de criptografia local de notas sensíveis; testes de segurança (OWASP ZAP) em *pipeline*.
Medida:	100% das novas leituras, metas e anotações criadas permanecem privadas até ação explícita do usuário; 100% das requisições autenticadas trafegam via HTTPS; nenhuma vulnerabilidade de severidade “Alta” em varredura ZAP.
Critério de Aceitação:	Suite de testes valida que o *default* de visibilidade é privado; auditoria manual confirma que alterações de visibilidade impactam apenas os campos selecionados; relatório ZAP sem achados “Altos” e no máximo 2 “Médios”, todos com plano de mitigação.

RNF-3 – Integração com múltiplos provedores de metadados com *fallback* e cache.	
Categoria:	Interoperabilidade, Disponibilidade e Eficiência.
Tática / Tratamento:	*Drivers* por provedor (ISBN/DOI) com *circuit breaker*, *rate-limit awareness*, *backoff* e *fallback* ordenado; cache com expiração e invalidação por chave; normalização de versões por ISBN e ordenação por popularidade para escolha do usuário.
Medida:	Taxa de sucesso de obtenção de metadados $\geq$ 99% com ao menos um provedor; *hit rate* de cache $\geq$ 60%; latência de busca p95 $\leq$ 1.5s sob 10 RPS e 20% de falhas induzidas no provedor primário.

Critério de Aceitação:	Em teste de caos com bloqueio do provedor primário por 10 minutos, o sistema mantém respostas válidas via provedores alternativos; logs mostram abertura e fechamento corretos do *circuit breaker*; métricas de cache e latência atendem aos limiares definidos.

RNF-4 – Envio de mídia resiliente com verificação de integridade e retomada.	
Categoria:	Confiabilidade e Eficiência.
Tática / Tratamento:	*Uploads* multipart/chunked com URLs pré-assinadas (S3), *checksums* por parte, registro de *offset* e retomada; publicação condicionada à confirmação do armazenamento; fila local com *retry* e *backoff*; indicador de estado na UI.
Medida:	Capacidade de retomar *uploads* após até 3 interrupções consecutivas sem perda de dados; verificação byte a byte (ETag/MD5) em 100% dos arquivos; p95 de conclusão $\leq$ 2x o tempo de uma transferência contínua sob perda de 5% de pacotes.
Critério de Aceitação:	Em ambiente de teste com injeção de falhas, 30 *uploads* de 50 MB cada concluem com integridade comprovada; nenhum post é publicado antes da confirmação do objeto final; telemetria registra tentativas e sucesso final.

RNF-5 – Recomendações explicáveis com indicadores de rigor em não ficção.	
Categoria:	Usabilidade e Manutenibilidade.
Tática / Tratamento:	Microserviço de IA (Python/FastAPI) com *features* de uso/afinidade e *features* de rigor (ex.: citações/impacto quando disponíveis); para cada item recomendado, persistir *Signals* (fonte e peso) e *Reasons* (tipo e detalhe) expostos à UI; versão do modelo versionada e auditável.
Medida:	95% das recomendações exibem ao menos 1 razão legível; para títulos de não ficção, presença de ao menos 1 *signal* de rigor em 90% dos casos; latência p95 do endpoint de recomendação $\leq$ 300 ms com *cache* quente.
Critério de Aceitação:	Testes automatizados validam a existência e o formato de *Reasons/Signals* em listas recomendadas; amostra manual de 100 itens de não ficção confirma o uso de sinais de rigor; *benchmark* controlado atesta latência dentro do p95 definido.

4 Arquitetura de Software

A Figura 1 apresenta a arquitetura do sistema *Bookd*. O desenho combina um cliente móvel *offline-first* com persistência local e um núcleo de serviços organizado em estilo Camada de Serviço (Fowler, 2002), além de um microserviço independente para recomendações. O objetivo principal é permitir que leitura, anotações, curadoria e metas funcionem integralmente sem rede, conciliando depois, de forma determinística, as alterações realizadas em múltiplos dispositivos, ao mesmo tempo em que integra metadados de fontes externas e publica mídia com retomada e verificação de integridade. A privacidade é tratada como padrão, a explicabilidade das recomendações é parte do contrato funcional e a infraestrutura é preparada para escalar horizontalmente sem acoplamentos desnecessários.

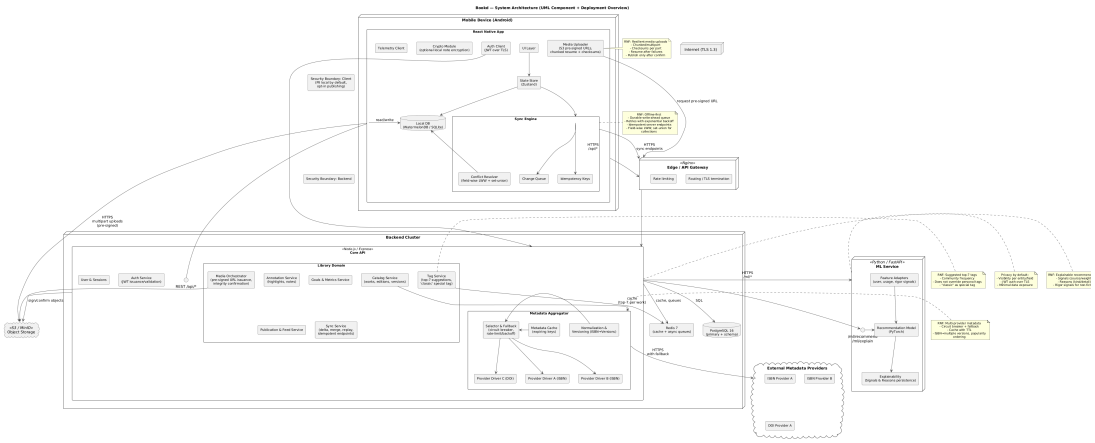

Figura 1 – Arquitetura do sistema.

No cliente Android, a aplicação em React Native mantém um repositório local baseado em WatermelonDB/SQLite, onde cada entidade de domínio (obra, versão de ISBN, leitura, anotação, tag, meta, publicação) é versionada com campos de controle (*id*, *updatedAt*, *originDeviceId*, *version*). Toda operação do usuário gera um registro na fila de mudanças persistida localmente, o que garante durabilidade mesmo em encerramentos abruptos do aplicativo. A sincronização ocorre em segundo plano quando há conectividade: o cliente empacota deltas de estado com chaves idempotentes e o servidor aplica uma política de reconciliação por campo (*field-wise* LWW) e união para coleções (por exemplo, conjuntos de tags e conjuntos de trechos anotados), retornando o estado consolidado para atualização do banco local. Esse mecanismo atende ao RNF de operação *offline-first* com sincronização resiliente (RNF RNF-1), pois evita perda de dados, tolera interrupções de rede e resolve conflitos de maneira determinística, com telemetria dos estados da fila (*queued*, *running*, *success*, *error*) visível para depuração.

O envio de mídia é tratado como um fluxo distinto e resiliente. Antes de publicar um post que contenha imagem, o cliente solicita ao servidor uma URL pré-assinada no *object storage* (S3/MinIO), realiza o *upload* em partes, calcula e envia *checksums* por

parte e registra o *offset* localmente para suportar retomada em caso de falhas. Somente após a confirmação de integridade do objeto final pelo *storage* é que o servidor autoriza a publicação do conteúdo associado. Esse desenho cumpre o RNF de mídia com verificação e retomada (RNF RNF-4) e, por descarregar a transferência diretamente para o *storage*, reduz o acoplamento e a carga do núcleo transacional.

A API do núcleo é implementada em Node.js/Express e expõe contratos REST documentados em OpenAPI. Internamente, serviços de domínio coesos encapsulam regras e integrações: Catálogo mantém obras, edições e o relacionamento "ISBN → múltiplas versões" com ordenação por popularidade; Anotações gerencia destaques e notas, inclusive anexos; Tags reúne a camada pessoal do usuário e as "7 tags sugeridas" derivadas de frequência comunitária, incluindo a tag especial "clássico", sem jamais sobrescrever as escolhas do leitor; Metas e Métricas consolida progresso por período e por tag, além de gerar agregações para retrospectivas; Publicações/Feed trata da visibilidade e do fluxo social; Sincronização oferece endpoints idempotentes para ingestão de deltas e *replay*; Orquestração de Mídia emite URLs pré-assinadas e confirma a integridade no *storage*. O estado sistêmico é persistido em PostgreSQL, com índices adequados (por exemplo, suporte a busca textual via `pg_trgm` para obras e autores), e Redis é utilizado tanto para *cache* de consultas e dados derivados (como "top 7" por obra) quanto para filas assíncronas de tarefas que não exigem resposta síncrona ao usuário.

A integração com provedores externos de metadados é encapsulada em um agregador com *drivers* especializados. Um seletor aplica *rate-limit awareness*, *circuit breaker* e *fallback* ordenado entre fontes (por exemplo, alternando entre dois provedores de ISBN e um provedor de DOI) e normaliza as respostas para o modelo interno. As versões de metadados associadas a um mesmo ISBN são consolidadas e ordenadas por popularidade para a decisão do usuário na hora de vincular sua leitura. Um *cache* com expiração controla a pressão sobre as fontes externas e melhora a latência percebida nas buscas. Essa abordagem cumpre o RNF de multi-provedor com *fallback* e *cache* (RNF RNF-3) e sustenta o requisito de múltiplas versões por ISBN com apresentação ordenada por popularidade.

O subsistema de recomendação é implementado como microserviço independente em Python/FastAPI, com modelo em PyTorch e *feature adapters* que combinam sinais de afinidade/uso (histórico de leitura, ritmo, temas preferidos) com indicadores de rigor para não ficção quando disponíveis. As respostas incluem não apenas a lista de itens recomendados, mas também estruturas de explicabilidade persistidas e exibíveis na interface: *Signals* (fonte e peso) e *Reasons* (tipo e detalhe) justificam cada sugestão. A separação em processo próprio permite que a equipe evolua e escale o motor de IA sem impactar a estabilidade do núcleo transacional, atendendo ao RNF de recomendações explicáveis (RNF RNF-5) e mantendo baixo acoplamento.

A segurança e a privacidade atravessam todas as camadas. Por padrão, leituras, metas e anotações são privadas; qualquer publicação exige ação explícita do usuário, e a visibilidade é configurável por entidade e por campo. A autenticação utiliza JWT e todo o tráfego é protegido por TLS 1.3 com terminação no *reverse proxy*. Na ponta do cliente, há a opção de criptografia local para notas sensíveis, o que impede que conteúdos privados deixem o dispositivo em texto claro. Varreduras automatizadas de segurança integram o ciclo de *build*, e políticas mínimas de exposição de dados são aplicadas em todos os endpoints. Esse conjunto de medidas materializa o RNF de privacidade por padrão com controle de visibilidade (RNF RNF-2).

A observabilidade foi planejada para apoiar diagnóstico e evolução. Requisições recebem correlação de *trace* de ponta a ponta (cliente, API, IA), logs são estruturados e capturam decisões de reconciliação de conflitos, abertura e fechamento de *circuit breakers* e resultados de *uploads* de mídia, enquanto métricas de latência, taxa e erro são expostas para *dashboards*. Testes automatizados cobrem contratos de API, cenários de sincronização com injeção de falhas de rede, retomada de *upload*, *chaos testing* de provedores externos e verificação de explicabilidade nas respostas do microserviço de IA. O objetivo é garantir que os critérios de aceitação definidos para cada RNF possam ser aferidos continuamente em integração e em produção.

Do ponto de vista de implantação, o *reverse proxy* Nginx realiza terminação TLS e *rate limiting* e encaminha tráfego para instâncias sem estado da API, o que permite escalar horizontalmente conforme demanda. O PostgreSQL armazena o estado transacional com migrações versionadas; o Redis suporta *cache* e filas; o *object storage* S3/MinIO hospeda mídias e verifica a integridade dos objetos antes de qualquer publicação. O microserviço de IA escala separadamente, dado que sua carga e seu perfil de latência diferem do núcleo transacional. Em desenvolvimento, um conjunto de contêineres Docker/Compose reproduz os serviços de apoio; em produção, recomenda-se S3 e bancos gerenciados, além de *autoscaling* controlado por métricas.

O conjunto de decisões arquiteturais resulta de compromissos explícitos. A adoção de um núcleo modular com um único ponto de verdade relacional simplifica consistência, relatórios e governança de dados, evitando a complexidade prematura de um mosaico de microserviços. A separação do motor de recomendações, por sua vez, absorve a variabilidade de modelos, bibliotecas e *tooling* de ML sem contaminar o ciclo de vida do núcleo. A opção por eventual consistência entre cliente e servidor é um corolário da estratégia *offline-first*: favorece disponibilidade no dispositivo e, com uma política de *merge* clara, preserva a previsibilidade. A orquestração de mídia via URL pré-assinada é uma defesa em profundidade tanto para desempenho quanto para confiabilidade. Por fim, a presença de um agregador de metadados com *fallback* garante resiliência a falhas e mudanças de política dos provedores externos, preservando a experiência do usuário quando fontes

individuais se tornam indisponíveis.

Em síntese, a arquitetura implementa, de forma coesa, os requisitos não funcionais eleitos como condutores: operação *offline-first* com sincronização resiliente (RNF RNF-1), privacidade por padrão e visibilidade granular (RNF RNF-2), integração com múltiplos provedores com *fallback* e *cache* (RNF RNF-3), mídia com verificação e retomada (RNF RNF-4) e recomendações explicáveis com sinais de rigor para não ficção (RNF RNF-5). O resultado é um sistema robusto, observável e extensível, capaz de operar bem em conectividade intermitente e de evoluir sem rupturas na experiência do leitor.

5 Projeto dos Componentes da Arquitetura

Conforme mostrado na Figura 1, o sistema está particionado em cinco subsistemas coesos: o cliente móvel, que concentra a apresentação e a orquestração local de dados sob a estratégia *offline-first*; o núcleo de serviços, que encapsula as regras de negócio e expõe contratos REST estáveis; o agregador de metadados, que integra e normaliza informações de múltiplos provedores de ISBN/DOI; o microserviço de recomendação, responsável pelo cálculo e pela explicabilidade das sugestões; e a infraestrutura de suporte, que provê terminação segura, persistência, cache, filas, armazenamento de objetos e observabilidade. A decomposição em subsistemas, camadas e componentes segue princípios consolidados de arquitetura de software e de aplicações corporativas (Bass; Clements; Kazman, 2021; Fowler, 2002; Souza, 2020), combinados a práticas contemporâneas de microsserviços (Newman, 2021; Nygard, 2018). Ao longo desta seção, descrevemos para cada subsistema suas camadas de apresentação (quando aplicável), de negócio e de acesso a dados, indicando contratos, invariantes e o atendimento aos RNFs condutores (RNF RNF-1, RNF RNF-2, RNF RNF-3, RNF RNF-4, RNF RNF-5). As Figuras 2 a 6 apresentam os componentes por subsistema, e as Figuras 7, 8, 9 e 10 ilustram os fluxos críticos.

5.1 Subsistema SS01 — Cliente Móvel

O cliente Android em React Native permite que leitura, anotações, curadoria por tags, metas, publicações e retrospectivas operem integralmente no dispositivo, sem dependência imediata de rede, com posterior reconciliação determinística. A aplicação conversa apenas com um *state store* que orquestra casos de uso, persiste entidades em WatermelonDB/SQLite e aciona o mecanismo de sincronização. Conflitos são resolvidos com *last-writer-wins* por campo e união para coleções, garantindo determinismo mesmo em reentregas e *replays*, em linha com técnicas de replicação otimista e resolução de conflitos em sistemas intensivos em dados (Kleppmann, 2017). A autenticação por JWT é gerida por um cliente dedicado com renovação silenciosa; o envio de mídia solicita URLs pré-assinadas, transmite partes com verificação de integridade e retoma a partir do último *offset* confirmado; notas sensíveis podem ser cifradas localmente, respeitando *privacy by default* (RNF RNF-2). A Figura 2 mostra os componentes e suas relações internas, destacando fila durável, idempotência e *merge* determinístico como elementos-chave do RNF RNF-1.

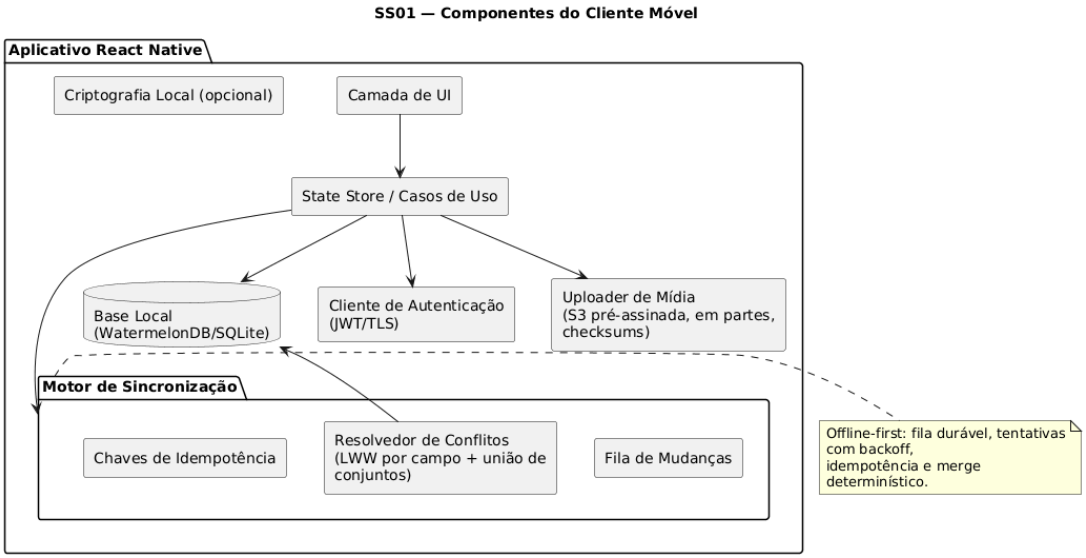

Figura 2 – Componentes do SS01 (Cliente Móvel): UI e *state store* sobre base local (WatermelonDB/SQLite), motor de sincronização com fila durável, chaves idempotentes e resolvedor de conflitos, cliente de autenticação JWT, *uploader* de mídia com retomada e módulo opcional de criptografia local.

5.1.1 Camada de Negócio

Os casos de uso operam sobre entidades locais versionadas com metadados (*id*, *updatedAt*, *originDeviceId*, *version*). Cada ação gera um evento na fila durável identificado por chave idempotente; o sincronizador agrupa deltas, agenda tentativas com *backoff* exponencial e envia lotes sob TLS. A reconciliação aplica LWW por campo e união para

coleções, devolvendo ao cliente o estado consolidado, materializando o RNF RNF-1 e seguindo recomendações para sistemas distribuídos orientados a logs de eventos (Kleppmann, 2017).

5.1.2 Camada de Apresentação

As telas são compostas exclusivamente pelo *state store*, sem dependências diretas de rede. Jornadas como leitura, criação/edição de anotações, curadoria de tags, definição de metas e preparação de retrospectivas permanecem responsivas em conectividade intermitente e apresentam, quando cabível, as *Reasons* e *Signals* das recomendações (RNF RNF-5), em linha com boas práticas de separação entre lógica de apresentação e de domínio em aplicações corporativas (Fowler, 2002).

5.1.3 Camada de Acesso a Dados

O repositório local mantém o grafo de entidades, a fila de mudanças e marcadores de progresso de *upload*, com transações que sobrevivem a encerramentos abruptos. Índices locais favorecem buscas por título, autor e tags; visões derivadas alimentam métricas e retrospectivas. Invariantes garantem *ids* únicos, monotonicidade de *updatedAt* por origem e integridade referencial entre leituras, anotações, tags e publicações, conforme preconizado por arquiteturas orientadas a domínio e catálogos de padrões de persistência (Fowler, 2002).

5.2 Subsistema SS02 — Núcleo de Serviços

O núcleo em Node.js/Express expõe contratos REST e encapsula regras de negócio: Catálogo, Anotações, Tags, Metas e Métricas, Publicações/Feed, Sincronização e Orquestração de Mídia. O Catálogo mantém obras, edições e múltiplas versões de metadados por ISBN, ordenadas por popularidade; Anotações trata destaques, notas e anexos sob a mesma matriz de visibilidade de Publicações/Feed, em consonância com *privacy by default* (RNF RNF-2); Tags integra preferências do leitor com sugestões automáticas; Metas e Métricas calcula progresso por período e por tema e prepara dados para retrospectivas; Sincronização aceita deltas idempotentes, aplica *merge* por campo e união para coleções e devolve estados consolidados (RNF RNF-1); Orquestração de Mídia emite URLs pré-assinadas, valida confirmações do *object storage* e só autoriza publicação após integridade comprovada (RNF RNF-4). A organização em serviços coesos e contratos bem definidos segue princípios de microsserviços e integração robusta (Newman, 2021; Richardson; Amundsen; Ruby, 2013). A Figura 3 delineia esses serviços e suas dependências.

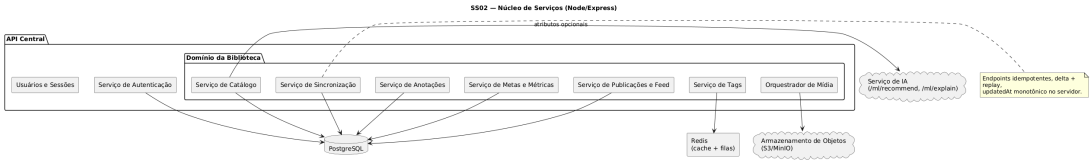

Figura 3 – Componentes do SS02 (Núcleo de Serviços): Catálogo, Anotações, Tags, Metas e Métricas, Publicações/Feed, Sincronização e Orquestração de Mídia sobre PostgreSQL e Redis, com integração ao serviço de IA para recomendações e explicações.

5.2.1 Camada de Negócio

As regras asseguram consistência referencial, idempotência em rotas críticas, monotonicidade de *updatedAt* como fonte de verdade e aplicação de visibilidade por campo (RNF RNF-2). Contratos estáveis são documentados em OpenAPI, com semântica explícita de erro e de reentrega, em conformidade com recomendações para APIs REST evolutivas e de longo ciclo de vida (Richardson; Amundsen; Ruby, 2013; Bass; Clements; Kazman, 2021).

5.2.2 Camada de Apresentação

A apresentação é feita por endpoints REST versionados com autenticação por JWT e minimização de dados (RNF RNF-2). As respostas podem incluir *Reasons* e *Signals* do microserviço de IA, prontos para exibição (RNF RNF-5), mantendo o núcleo desacoplado de detalhes de UI, em linha com abordagens em camadas para aplicações Web (Souza, 2020).

5.2.3 Camada de Acesso a Dados

PostgreSQL armazena o estado transacional com índices e restrições adequados e, quando aplicável, busca textual; Redis funciona como cache de leitura e mecanismo de filas assíncronas. Repositórios encapsulam SQL e transações, e migrações versionadas garantem reprodutibilidade, alinhando-se às práticas de projetos intensivos em dados descritas em (Kleppmann, 2017).

5.3 Subsistema SS03 — Agregador de Metadados

O agregador integra provedores ISBN/DOI por meio de *drivers* individuais, seleção com *circuit breaker*, consciência de limites e *fallback*, normalização para o modelo interno e cache com expiração. Versões múltiplas de um mesmo ISBN são consolidadas e ordenadas por popularidade. A composição atende disponibilidade, eficiência e interoperabilidade (RNF RNF-3). O uso de *circuit breakers*, *backoff* e cache segue padrões de resiliência

amplamente descritos na literatura de sistemas em produção (Nygard, 2018; Bass; Clements; Kazman, 2021). A Figura 4 descreve a composição desses componentes.

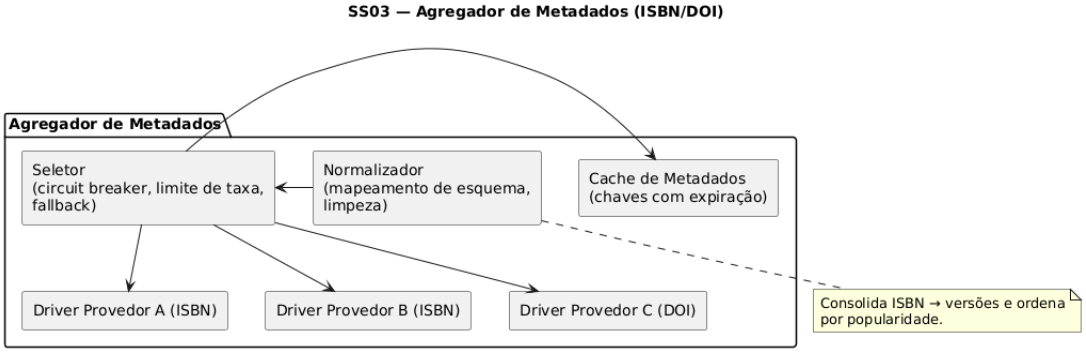

Figura 4 – Componentes do SS03 (Agregador de Metadados): seletor com *circuit breaker*, *fallback* e *rate-limit awareness*, *drivers* ISBN/DOI, normalizador para o modelo interno e cache com expiração.

5.3.1 Camada de Negócio

A lógica decide a ordem de consulta, ativa *fallback* diante de falhas ou limites, normaliza autores, editoras e edições e entrega resultados prontos para o Catálogo. Telemetria de disponibilidade e eventos de abertura/fechamento de *circuit breaker* reforçam o RNF RNF-3, em linha com recomendações de observação e proteção de integrações externas (Nygard, 2018; Sigelman; Barroso, 2020).

5.3.2 Camada de Apresentação

Não há API pública; apenas contratos internos consumidos pelo núcleo, com estruturas já compatíveis com a ordenação por popularidade.

5.3.3 Camada de Acesso a Dados

Redis guarda respostas normalizadas com expiração coerente com os SLAs dos provedores externos; logs de proveniência podem ser armazenados para auditoria, seguindo princípios de rastreabilidade e diagnóstico em arquiteturas distribuídas (Sigelman; Barroso, 2020).

5.4 Subsistema SS04 — Microserviço de Recomendação

O microserviço de recomendação em Python/FastAPI calcula listas personalizadas, com explicabilidade via estruturas de *Signals* e *Reasons*. O modelo em PyTorch combina *features* de afinidade/uso e indicadores de rigor em não ficção (RNF RNF-5), mantendo baixo acoplamento com o núcleo. A abordagem segue princípios de sistemas de recomen-

dação baseados em múltiplos sinais de usuário e de item (Aggarwal, 2016). A Figura 5 detalha esses componentes.

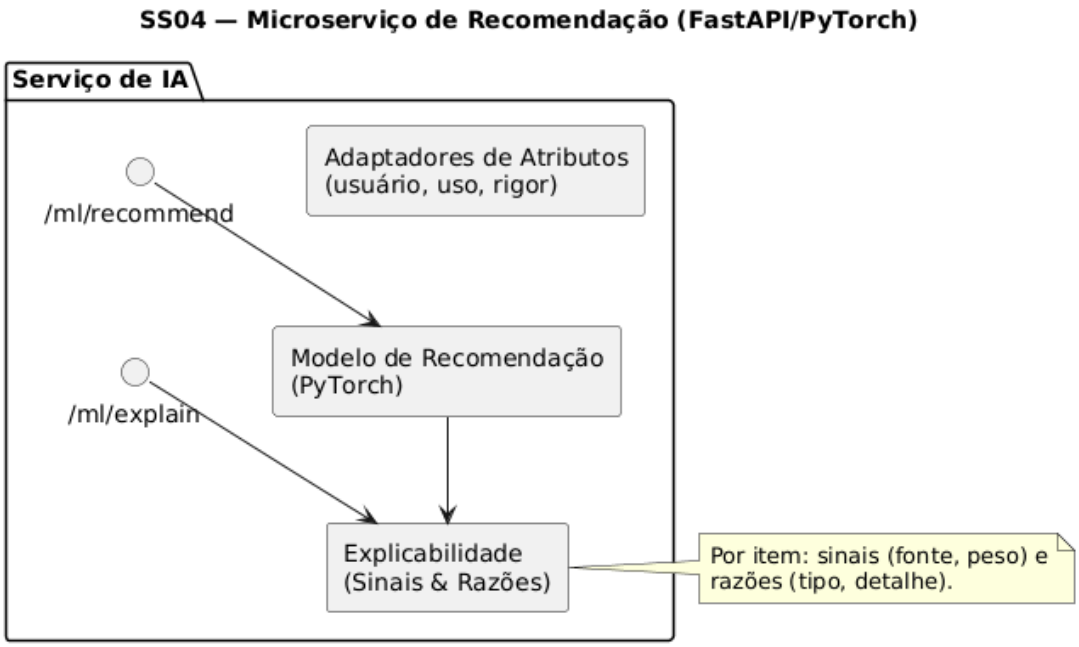

Figura 5 – Componentes do SS04 (Microserviço de Recomendação): adaptadores de *features*, modelo de recomendação em PyTorch e módulo de explicabilidade, expostos por endpoints internos `/ml/recommend` e `/ml/explain`.

5.4.1 Camada de Negócio

A camada combina sinais de gosto e de rigor, produz escores e listas ordenadas e gera explicações legíveis pela UI. Versionamento de modelos e *pipelines* garante reprodutibilidade e auditoria; testes *A/B* e *canary* mitigam riscos em trocas, como recomendado em projetos de sistemas de recomendação em produção (Aggarwal, 2016; Bass; Clements; Kazman, 2021).

5.4.2 Camada de Apresentação

A apresentação ocorre por endpoints internos consumidos pelo núcleo; as respostas incluem campos imediatamente exibíveis para o usuário, incluindo *Reasons* e *Signals* (RNF RNF-5), mantendo o microserviço focado em lógica algorítmica e deixando a orquestração para o núcleo de serviços (Newman, 2021).

5.4.3 Camada de Acesso a Dados

O microserviço não guarda listas permanentemente; sinais e razões podem ser resumidos no repositório do núcleo para auditoria. Métricas de latência e acurácia percebida

orientam *tuning* e capacidade, em consonância com práticas de monitoramento de serviços de ML em produção (Sigelman; Barroso, 2020).

5.5 Subsistema SS05 — Infraestrutura de Suporte

A infraestrutura garante segurança, resiliência e observabilidade: o *reverse proxy* termina TLS e aplica *rate limiting*; a API é *stateless* e escala horizontalmente; PostgreSQL mantém o estado transacional; Redis provê cache e filas; S3/MinIO realiza armazenamento de mídia com verificação de integridade (RNF RNF-4); a *stack* de observabilidade agrega logs, métricas e *traces* com correlação ponta a ponta. O uso de armazenamento de objetos com *multipart upload*, URLs pré-assinadas e checagem de integridade segue as recomendações dos provedores de nuvem (Amazon Web Services, 2024), enquanto a composição geral da infraestrutura reflete práticas consolidadas de disponibilidade e operação em produção (Bass; Clements; Kazman, 2021; Nygard, 2018; Sigelman; Barroso, 2020). Garantias de trânsito cifrado e de *privacy by default* reforçam o RNF RNF-2. A Figura 6 sintetiza essa composição.

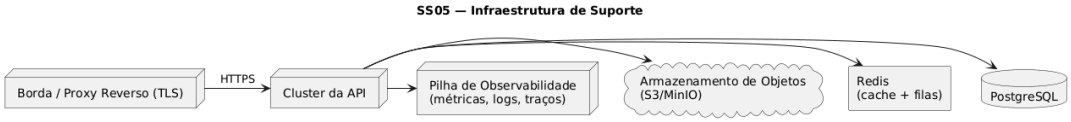

Figura 6 – Componentes do SS05 (Infraestrutura): *reverse proxy* com TLS e *rate limiting*, cluster da API, PostgreSQL, Redis, S3/MinIO e *stack* de observabilidade.

5.6 Cenários e Fluxos Arquiteturais

Os fluxos críticos reforçam os RNFs condutores e ligam os subsistemas. No fluxo *offline-first* (Figura 7), ações locais são registradas, enfileiradas e sincronizadas com idempotência e *merge* determinístico (RNF RNF-1), alinhados a padrões de replicação otimista (Kleppmann, 2017). No fluxo de mídia (Figura 8), *uploads* multipart com verificação de integridade e retomada só resultam em publicação após confirmação do objeto no *object storage* (RNF RNF-4), conforme documentação de serviços de armazenamento de objetos (Amazon Web Services, 2024). No fluxo de metadados (Figura 9), o agregador aplica *fallback* e normalização entre provedores externos, garantindo disponibilidade e eficiência (RNF RNF-3) com o apoio de *circuit breakers* e cache (Nygard, 2018). No fluxo de recomendação (Figura 10), a API coordena `/ml/recommend` e `/ml/explain`, devolvendo itens com *Reasons* e *Signals* exibíveis, em conformidade com RNF RNF-5 e práticas de explicabilidade em sistemas de recomendação (Aggarwal, 2016).

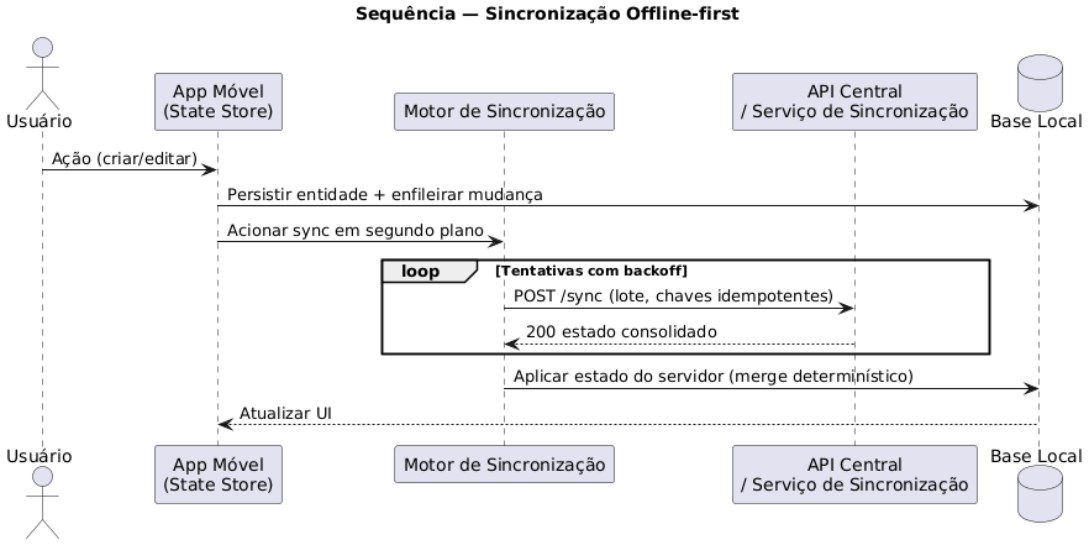

Figura 7 – Sequência do fluxo de sincronização *offline-first*: operações locais são enfileiradas com chaves idempotentes, enviadas em lotes sob TLS e consolidadas com reconciliação determinística por campo.

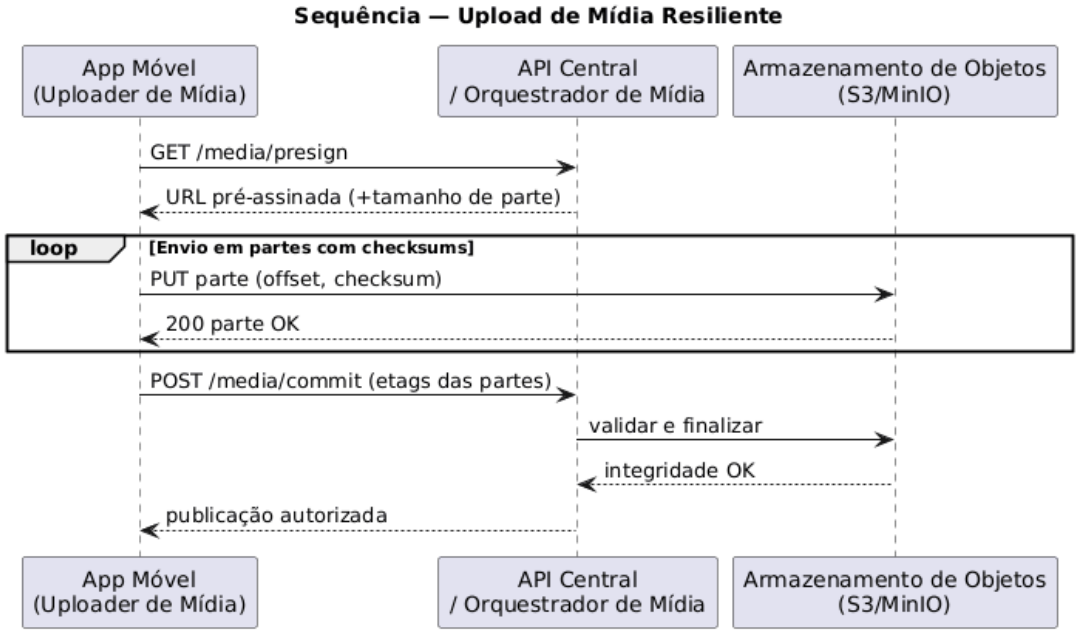

Figura 8 – Sequência do fluxo de *upload* de mídia resiliente: emissão de URL pré-assinada, envio multipart com *checksums*, confirmação de integridade no *object storage* e publicação condicionada.

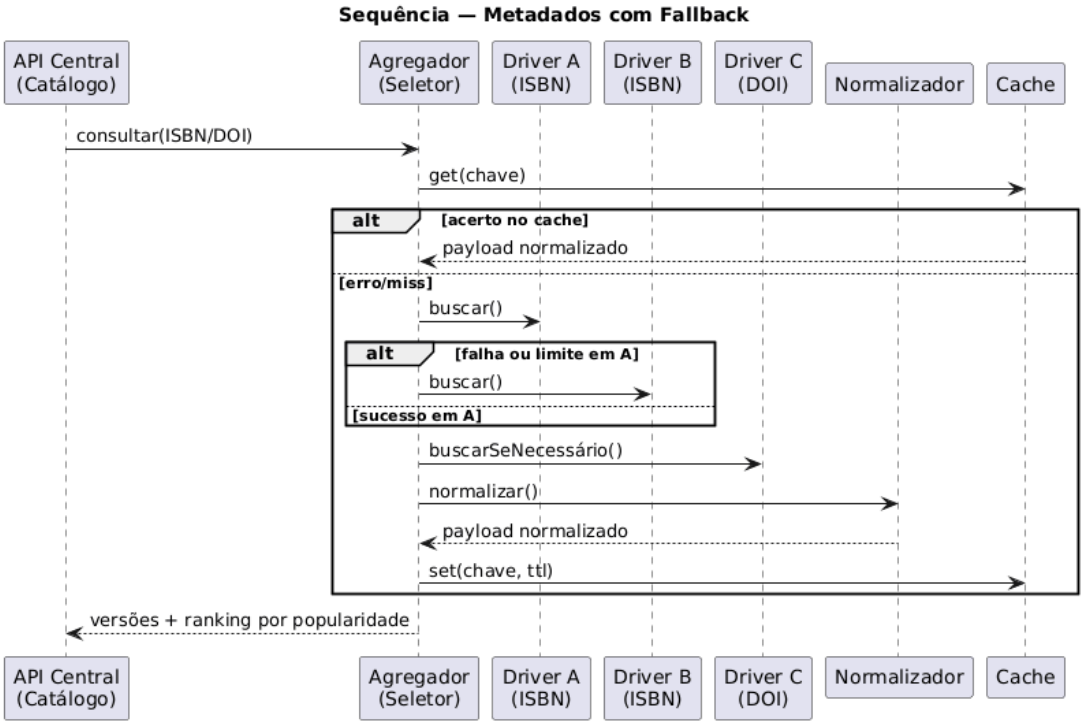

Figura 9 – Sequência do fluxo de metadados com *fallback*: seletor consulta cache, aplica *circuit breaker*, escolhe provedores ISBN/DOI, normaliza para o modelo interno e armazena resultados em cache com expiração para eficiência.

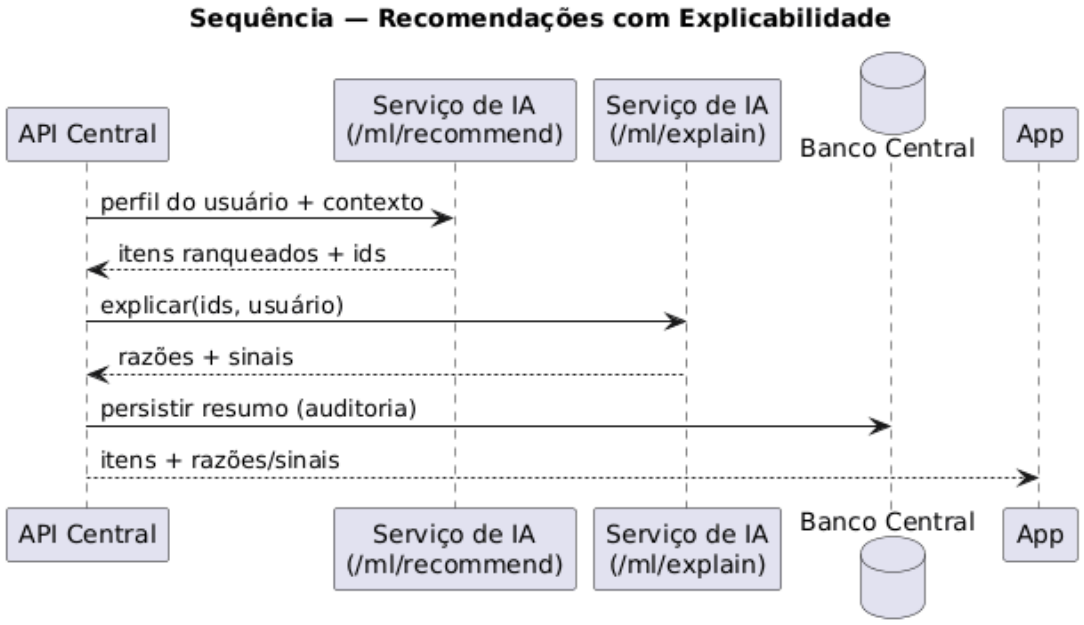

Figura 10 – Sequência do fluxo de recomendações com explicabilidade: a API orquestra `/ml/recommend` e `/ml/explain`, persiste resumos para auditoria e devolve itens com *Reasons* e *Signals* exibíveis.

Referências

AGGARWAL, C. C. *Recommender Systems: The Textbook*. [S.l.]: Springer, 2016. Citado 2 vezes nas páginas 15 e 16.

Amazon Web Services. *Amazon Simple Storage Service Documentation*. 2024. <https://docs.aws.amazon.com/s3/index.html>. Citado na página 16.

BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software Architecture in Practice*. 4. ed. [S.l.]: Addison-Wesley, 2021. Citado 5 vezes nas páginas 10, 13, 14, 15 e 16.

FOWLER, M. *Patterns of Enterprise Application Architecture*. 1. ed. [S.l.]: Addison-Wesley, 2002. ISBN 9780321127426. Citado 3 vezes nas páginas 7, 10 e 12.

KLEPPMANN, M. *Designing Data-Intensive Applications*. [S.l.]: O'Reilly Media, 2017. Citado 4 vezes nas páginas 11, 12, 13 e 16.

NEWMAN, S. *Building Microservices*. 2. ed. [S.l.]: O'Reilly Media, 2021. Citado 3 vezes nas páginas 10, 12 e 15.

NYGARD, M. T. *Release It!: Design and Deploy Production-Ready Software*. 2. ed. [S.l.]: Pragmatic Bookshelf, 2018. Citado 3 vezes nas páginas 10, 14 e 16.

RICHARDSON, L.; AMUNDSEN, M.; RUBY, S. *RESTful Web APIs*. [S.l.]: O'Reilly Media, 2013. Citado 2 vezes nas páginas 12 e 13.

SIGELMAN, B.; BARROSO, L. A. *Distributed Tracing in Practice*. [S.l.]: O'Reilly Media, 2020. Citado 2 vezes nas páginas 14 e 16.

SOUZA, V. E. S. The FrameWeb Approach to Web Engineering: Past, Present and Future. In: ALMEIDA, J. P. A.; GUIZZARDI, G. (Ed.). *Engineering Ontologies and Ontologies for Engineering*. 1. ed. Vitória, ES, Brazil: NEMO, 2020. cap. 8, p. 100–124. ISBN 9781393963035. Available on: <http://purl.org/nemo/celebratingfalbo>. Citado 2 vezes nas páginas 10 e 13.