

Plano de Próximas Tarefas (Execução)

Este documento lista as próximas entregas priorizadas, com foco em: integração frontend↔backend, continuação do backend (NestJS/Mongoose), e preparação do Capacitor. Baseado em [docs/guidelines/synapse-feature-roadmap.md](#), [docs/guidelines/synapse-specs.md](#) e no estado atual do repo.

Contexto Atual

- Backend: NestJS base com conexão MongoDB, health check e Schemas para `users`, `decks`, `cards`, `classes`, `assignments`, `reviews`, `progress` (ver [apps/server/src/database/schemas/*](#)). Seed disponível: `npm run seed` (`apps/server`).
- Frontend: Páginas mock (Login, Dashboard, Decks, DeckNew, DeckDetail, Classes, ClassNew, ClassDetail, Study, Reports) em `apps/web/synapse-flash-study-main/`. Navegação pronta, dados via mocks em `src/lib/mockService.ts`.
- Docs: modelo de dados, requisitos não funcionais, roadmap por feature e comparativo do frontend já escritos.

Milestones Prioritários

1. Integração de Autenticação e Infra de API.
2. Users & RBAC básicos.
3. Decks & Cards (CRUD) reais.
4. Classes & Assignments.
5. Study (SRS) — fila e reviews.
6. Reports.
7. Observabilidade & Segurança (hardening).
8. CI/CD e DevOps.
9. Capacitor (preparo para app).
10. Documentação & QA.

1) Integração Auth e Infra de API

- Backend
 - Criar `AuthModule` : `POST /auth/login` (JWT), `POST /auth/logout`, `GET /users/me` .
 - Hash de senha com argon2; rate limit no login.
 - Guards: `JwtAuthGuard`, decorator `@Roles()` com `RolesGuard` (placeholders inicialmente).
 - Swagger com esquemas de auth e DTOs.
- Frontend
 - Adicionar `src/lib/api/client.ts` (axios) usando `import.meta.env.VITE_API_URL` .
 - Interceptors: anexa Bearer; trata 401 (planejar refresh, mesmo que stub).
 - `ProtectedRoute` / `RequireRole` e normalizar roles para `TEACHER|STUDENT|ADMIN` .
 - Trocar `AuthContext` para consumir `POST /auth/login` + `GET /users/me` (fallback: usar mock quando `VITE_API_URL` ausente).

2) Usuários & RBAC

- Backend
 - `POST /users` (ADMIN), `PATCH /users/:id` (self/ADMIN). DTOs + validação.
 - Logs/auditoria ao criar/editar (campo `userId` no contexto de request).
- Frontend
 - Tela Perfil (edição própria) — básico.
 - (Opcional MVP) Tela Admin Usuários (lista/criar/editar, filtro por role).

3) Decks & Cards (CRUD)

- Backend
 - `GET /decks?mine&query&tags&page&limit&sort` (paginação + busca por índice de texto).
 - `POST /decks`, `PATCH /decks/:id`, `DELETE /decks/:id` (owner/ADMIN).
 - `GET /decks/:id/cards`, `POST /decks/:id/cards`, `PATCH /cards/:id`, `DELETE /cards/:id`.
 - Atualização consistente de `cards_count` (middleware/service).
- Frontend
 - Substituir `mockService` por hooks de API (React Query) gradualmente: listagem de decks, detalhe e cards.
 - Formular criação/edição; estados loading/error/empty com skeletons.

4) Classes & Assignments

- Backend
 - `GET /classes` (por `teacher_id`), `POST /classes`, `PATCH /classes/:id`.
 - `POST /classes/:id/students` (add/remove em lote).
 - `POST /assignments { deckId, classId|studentId, dueDate? }` e `GET /assignments`.
 - Garantir índices únicos parciais para evitar duplicatas (já nos Schemas).
- Frontend
 - Trocar `ClassDetail` para usar API (alunos e atribuição de decks via modais).
 - Página "Atribuídos a mim" (aluno) — lista decks por assignments.

5) Study (SRS) — Fila, Reviews e Algoritmo

- Backend
 - `GET /study/queue?deckId=&size=` : retorna lote devido (ordem por `next_due_at`).
 - `POST /study/review { cardId, deckId, rating, elapsedMs }` : calcula próximo agendamento (`scheduleNext`).
 - `GET /study/progress?deckId=` : resumo por aluno/deck (ou materializar em `progress`).
 - Service SRS: implementar cálculo (FSRS simplificado) e idempotência.
 - Documento de especificação do algoritmo (ver `docs/algoritmo-srs.md`).

- Políticas: tamanho de lote diário, proporção entre novos e em revisão, limite de novos/dia, estratégia de "leech".
- Parâmetros por deck (opcional): permitir ajuste fino no futuro (feature flag/env).
- Frontend
 - Substituir Study mock para consumir fila e postar respostas; timer por card e atalhos.
 - Indicadores de progresso na sessão; estados de loading/empty/fim de sessão.

5.1 Algoritmo SRS — Tarefas detalhadas

- Definir modelo de estado por card/aluno: `stability (0..1)` , `difficulty (0..1)` , `interval (dias)` (implícito via `next_due_at`).
- Mapear ratings (`0..3` → `Again/Hard/Good/Easy`) para ajustes de `stability / difficulty` e multiplicador de intervalo.
- Função `scheduleNext(prevState, rating, elapsedMs, now)` retorna `{ next_due_at, stability, difficulty }` .
- Inicialização de novos cards (S/D defaults) e primeiras iterações (I1/I2).
- Regras de "clamp" (mín/máx de intervalo e dos parâmetros) e timezone (UTC na API).
- Testes determinísticos de unidade (casos de borda) + simulador simples (N reviews sintéticos) para validar crescimento/coerência.
- Métricas: taxa de acerto por rating, média de intervalo antes/depois, distribuição de due.

6) Reports

- Backend
 - GET `/reports/teacher/overview?classId?` — agregações por turma/deck.
 - GET `/reports/student/overview?deckId?` — agregações por deck/aluno.
 - Cache curto (Redis opcional) e ETag.
- Frontend
 - Substituir dados mock em `/reports` por chamadas reais; chart com loading.

7) Observabilidade & Segurança

- Backend
 - Logger estruturado (pino) com `request-id` + middleware de correlação.
 - GET `/ready` incluindo verificação de DB/filas.
 - OpenTelemetry (HTTP + Mongo) com amostragem configurável.
 - Helmet (já), CORS restrito por env (já), rate limit global.
 - Sanitização de inputs e escape seguro de markdown/HTML no render no frontend.
- Frontend
 - ErrorBoundary global e pages-level; aria-live nos toasts relevantes.

8) CI/CD & DevOps

- Repositório
 - Root `package.json` (workspaces) para scripts unificados (lint/test/build).
 - Pipeline CI: lint + testes (web/server) + build.

- Deploy
 - Web: Cloudflare Pages (envs VITE_API_URL e VITE_APP_VERSION).
 - API: Railway/Render/Cloud Run, variáveis de ambiente seguras.
- Banco & Migrações
 - Script idempotente de criação de índices (prod) e verificação.
 - Seeds por ambiente (dev/staging) controlados por flag.

9) Capacitor (Preparo)

- Frontend
 - Adicionar `capacitor.config.ts`; `npx cap init` (nome/id do app).
 - Abstrair armazenamento de sessão: web (cookie httpOnly) vs app (Secure Storage + header Bearer).
 - Validar rotas/paths e assets para build móvel.
 - (Futuro) Plugins: Local Notifications para lembretes de estudo.

10) Documentação & QA

- Documentação
 - `.env.example` (web e server) e README com setup local.
 - Atualizar Swagger conforme endpoints; linkar nas docs.
- Testes
 - Unit (services e utilitários), integração (controllers/repos), e2e básicos (login, criar deck, estudar 1 card).
 - Playwright/Cypress para fluxos críticos no web.

Itens Imediatos (1ª Semana)

- Backend: AuthModule (login/jwt), UsersModule básico (`/users/me`), guards e rate limit do login.
- Frontend: `api/client.ts` (axios + interceptors); `ProtectedRoute`; adaptar AuthContext para backend (fallback mock).
- Decks: GET `/decks` paginado e troca da lista em `/decks` para chamar API.
- Observabilidade: logger estruturado e health `/ready`.
- Dev: `.env.example` (web/server) + documentação rápida de setup.
- SRS: escrever `docs/algoritmo-srs.md` e protótipo da `scheduleNext` com testes básicos (sem expor endpoint ainda).

Dependências & Riscos

- Acesso ao MongoDB Atlas e variáveis de ambiente em staging/produção.
- Complexidade do SRS: começar com modelo simplificado e evoluir.
- Performance de agregações (reports): monitorar índices e avaliar cache.

Aceite por Milestone (resumo)

- Auth: rotas protegidas; login funcionando e `/users/me` retorna perfil.
- Decks/Cards: CRUD completo com paginação; `cards_count` consistente.

- Classes/Assignments: atribuição visível para alunos; sem duplicatas.
 - Study: fila respeita `next_due_at`; review calcula próximo agendamento.
 - Reports: agregações retornam em tempo adequado; gráficos mostram dados reais.
-

11) Cobertura dos Requisitos Funcionais Solicitados

Lista de requisitos a contemplar (alguns já parcialmente cobertos):

1. Cadastro de professor
2. Cadastro de aluno por convite
3. Login por e-mail/senha
4. Recuperação de senha
5. Edição de perfil
6. CRUD de turmas
7. Convite de alunos
8. CRUD de decks
9. CRUD de cards
10. Categorias (tags) de card
11. Importação de decks (CSV)
12. Exportação de decks (CSV/JSON)
13. Visualização de decks (listagem + detalhes)
14. Responder os decks (estudo)
15. Visualizar a nota (score / performance)
16. Marcar perguntas (flag/favorito/difícil)
17. Dashboard por turma
18. Relatório exportável
19. Publicar deck para turma
20. Retirar publicação
21. Lembrete diário por e-mail
22. Gestão básica (admin usuários)
23. Idiomas de UI (i18n)
24. Atalhos e leitura de tela (acessibilidade)
25. Logs de auditoria

11.1 Mapeamento para Milestones Existentes

Requisito	Milestone / Sessão	Observação
1,3	1 (Auth)	Cadastro professor via POST /users (role=TEACHER) + login
2,7	2 / 4	Fluxo convite gera token e cria aluno (CLASS join)
4	1 (Auth ext.)	Rotas: POST /auth/forgot, POST /auth/reset
5	2	PATCH /users/:id + tela Perfil
6	4	Classes CRUD
8,9,10	3	Decks + Cards + tags (índice texto)
11,12	3 (ext.)	Import/Export service + jobs se necessário
13	3	List & detail decks (paginação + filtros)

14,15	5	Study queue + cálculo de acertos e score por sessão
16	5	Campo flags ou is_flagged em review/card_state
17,18	6	Reports teacher + export (CSV)
19,20	4 / 3	Assignment = publicação; remoção = delete assignment
21	7 (ext.)	Scheduler (cron) + template e-mail (resumos due)
22	2	Admin Users UI + RBAC
23	Extra (UI infra)	i18n frontend (react-intl ou i18next) + fallback backend
24	7 / A11y	Atalhos (cmd+/ help), roles ARIA, foco, screen reader labels
25	7	AuditLog collection + middleware (userId, ação, entidade)

11.2 Novas Tarefas Detalhadas

A) Convites & Cadastro Aluno

Backend:

- POST /invites { email, role=STUDENT, classId? } (TEACHER/ADMIN)
- Envia token (UUID) armazenado em invites (status: pending/accepted/expired)
- POST /auth/accept-invite { token, name, password } cria aluno + adiciona à turma.
- Expiração automática (TTL index ou campo expiresAt + cleanup job). Frontend:
- Tela envio de convites (multi-email) + feedback
- Tela aceitar convite (form com validação) / rota pública

B) Recuperação de Senha

Backend:

- POST /auth/forgot { email } gera token single-use (collection password_resets)
- POST /auth/reset { token, password } invalida token e atualiza hash. Frontend:
- Form "Esqueci minha senha" + confirmação
- Form redefinição (token via query param)

C) Importação / Exportação de Decks

Backend:

- POST /decks/import (multipart CSV) -> parser (stream) -> valida -> cria deck + cards.
- GET /decks/:id/export?format=csv|json -> gera arquivo (stream) com cabeçalhos.
- Validação de colunas: front,back,difficulty?,tags?. Frontend:
- Modal Import (upload + preview erros)
- Botão Export (download; mostrar snackbars)

D) Publicação e Atribuição

- Publicar = criar assignment (deckId , classId).
- Retirar = deletar assignment.
- GET extended: /classes/:id/assignments . Frontend:

- Toggle publicar/retirar dentro de DeckDetail (aba "Publicação").

E) Flag de Perguntas / Dificuldade

Backend:

- Campo em `reviews` ou `card_state` : `is_flagged` , `difficulty_overrides` .
- Rotas: `POST /study/flag { cardId, deckId, flag: boolean }` . Frontend:
- Botão "Marcar" durante estudo + filtro "Somente marcadas".

F) Nota / Score & Relatório Exportável

Backend:

- Persistir em `progress` métricas agregadas: `total_reviews`, `correct`, `accuracy_pct`, `last_session_score` .
- `GET /reports/teacher/overview` inclui score médio por turma e por deck.
- `GET /reports/teacher/export?classId=&format=csv` . Frontend:
- Dashboard turma: cards (resumo decks, alunos top, média acertos).
- Botão export CSV.

G) Lembrete Diário por E-mail

Backend:

- Scheduler (cron @daily) lista alunos com `due_count > 0` e envia e-mail.
- Template básico (HTML + texto). Feature flag (`EMAIL_REMINDERS_ENABLED`).
- Endpoint admin: `POST /admin/reminders/test { userId }` . Frontend:
- Preferência usuário: ativar/desativar lembretes.

H) Internacionalização (UI Idiomas)

Frontend:

- Adicionar i18n lib (i18next) + namespaces: `common`, `auth`, `decks`, `study`, `reports` .
- Detector de idioma (navigator + fallback pt-BR).
- Script de extração de chaves (lint para chaves órfãs). Backend:
- Suporte a cabeçalho `Accept-Language` (para strings futuras em e-mails).

I) Acessibilidade / Atalhos

Frontend:

- Mapear atalhos: estudo (1..4 rating, F flip, M marcar, N próximo, ESC sair).
- Página de ajuda (modal) listando atalhos.
- Teste com axe-core em dev (lint opcional) + roles ARIA.

J) Logs de Auditoria

Backend:

- Middleware injeta `requestId` + `userId` no contexto.
- Service `AuditLogService.record({ actorId, action, entity, entityId, meta })` .
- Eventos: criar/editar/deletar deck/card, publicar/retirar deck, convite emitido, aluno aceitou convite, reset de senha realizado.

- Endpoint admin: GET /admin/audit-logs?entity=&action=&userId=&page= . Frontend:
- Tela simples admin (tabela paginada + filtros básicos).

11.3 Ajustes de Modelo de Dados Necessários

- Collection invites : { email, role, classId?, token, status, expiresAt, createdAt } (TTL index em expiresAt).
- Collection password_resets : { userId, token, expiresAt, usedAt? } (TTL index).
- Extensão cards : campo tags: string[] (já previsto via Deck tags? Manter separado para granularidade).
- Extensão progress : adicionar accuracy_pct , last_session_score , flagged_count .
- Collection audit_logs : { ts, actorId, action, entity, entityId, meta } (índices: actorId, entity+entityId, action+ts desc).

11.4 Critérios de Aceite Adicionais

- Convite expira corretamente e rejeita token inválido/expirado.
- Reset de senha só aceita tokens ativos (single-use).
- Import ignora linhas inválidas e retorna relatório de erros (linhas, motivo).
- Export reflete dados atualizados e inclui cabeçalho consistente.
- Publicar/Retirar reflete imediatamente na turma (assignment listado/ausente).
- Flag de card persiste e filtragem funcional em estudo.
- Score calculado (accuracy %) visível por deck e por turma.
- Lembrete diário enviado apenas a quem tem cards "due" e opt-in ativo.
- UI alterna idioma sem recarregar (hot swap de namespace).
- Atalhos documentados e operacionais; navegação por teclado sobre fluxos principais.
- Auditoria registra eventos sensíveis e retorna em ordem cronológica com filtros.

11.5 Sequência Recomendada (Incremental)

1. Auth extensão (convites + reset) → (A,B)
2. Decks tags + publish/unpublish + flagging → (C,D,E)
3. Reports + score + export → (F)
4. Email reminders + audit logs → (G,J)
5. i18n + acessibilidade/atalhos → (H,I)
6. Import/Export decks se não feito junto tags → (C)

11.6 Riscos & Mitigações

- Import CSV grande: usar streaming e limite de tamanho (validação antes do parse).
- Envio de e-mails: fila futura (BullMQ) se volume crescer; por agora direto + retry simples.
- Audit log crescimento: política de retenção (ex.: 180 dias) + índice composto.
- i18n aumento de overhead: lazy loading namespaces.

11.7 Métricas de Sucesso

- Taxa de convite aceito (>70%).
- Tempo médio entre convite e aceitação.
- Acurácia média por turma / melhoria após 2 semanas.
- % de alunos que usam lembrete e completam sessões diárias.
- Latência média export (<2s decks médios <2k cards).
- Cobertura de auditoria (100% eventos catalogados disparando log).

12) Integração Frontend ↔ Backend (Plano Detalhado)

Objetivo: migrar gradualmente de mocks para API real garantindo consistência de tipos, tratamento uniforme de erros, autenticação e observabilidade fim-a-fim.

12.1 Camadas e Responsabilidades

Camada	Local	Responsável	Notas
API Client	apps/web/src/lib/api/client.ts	Axios instância	BaseUrl, headers, retries leves
Auth Guard	Frontend (hooks + rotas)	Verificar sessão/roles	Redireciona p/ login 401/403
DTOs	Backend (classes) & gerados p/ FE	Contratos	Fonte da verdade: backend
Types Gerados	apps/web/src/types/api/*.d.ts	Script openapi	npm run gen:types
Normalizers	apps/web/src/lib/api/mappers	Ajuste camelCase	Evitar lógica na view
Query Hooks	apps/web/src/hooks/api/*	React Query	Cache/paginação/staleTim
Error Boundary	apps/web/src/components/app/ErrorBoundary	UX de falhas	Report centralizado
Telemetria	Interceptor + Nest Interceptor	IDs correlação	header x-request-id

12.2 Geração de Tipos (OpenAPI)

- Adicionar `@nestjs/swagger` decorators a todos DTOs.
- Exportar `/docs-json` (Nest).
- Script: `scripts/generate-types.sh` usando `openapi-typescript` → `apps/web/src/types/api/schema.d.ts`.
- Adicionar verificação em CI (diff detectado falha).

12.3 Interceptores & Tratamento de Erros

Backend:

- Filtro global padroniza erro: `{ statusCode, code, message, details? }`.
- Codes padronizados: `AUTH_INVALID_CREDENTIALS`, `DECK_NOT_FOUND`, etc. Frontend:
- Interceptor mapeia para `AppError { code, message, httpStatus }`.
- Toast automático para erros não tratados + fallback friendly.
- Erros de validação exibidos campo a campo (forms RHF).

12.4 Fluxo de Autenticação Integrado

1. Login: `POST /auth/login` (recebe access + refresh cookie `httpOnly` opcional).
2. Ao carregar app: `GET /users/me` popula store.
3. Refresh: stub (adiar) ou endpoint `POST /auth/refresh` futuro.
4. Logout: limpa store + chama backend (revoga refresh se houver).

5. Rota protegida: se store vazio tenta `me` ; falha → login.

12.5 Migração Gradual de Mocks

Etapa	Feature	Ação
1	Auth	Substituir contexto mock por chamadas reais
2	Decks list	GET /decks com paginação real (fallback mock se erro)
3	Deck detail/cards	Carregar cards reais; editar/criar via API
4	Classes & assignments	Trocar listagem e publish/unpublish
5	Study queue	Primeiro somente leitura; depois postar review
6	Reports	Substituir métricas agregadas
7	Import/Export	Integrar endpoints e remover parser client

Critério de conclusão de etapa: componente sem referências a `mockService` para domínio migrado.

12.6 Sincronização & Consistência

- Revalidação automática (React Query `invalidateQueries`) em mutações de deck, card, assignment, review.
- Política de cache: decks list `staleTime=30s`, study queue `no-cache`, user `∞` até logout.
- Conciliação pessimista: aguardar resposta antes de atualizar cache em operações de SRS.

12.7 Observabilidade Integrada

Backend:

- Interceptor adiciona `x-request-id` se ausente.
- Log de JSON por request (method, path, status, duration, userId?). Frontend:
- Geração de `x-request-id` (UUID v4) por request se não presente.
- Hook global para enviar erros JS não tratados a `/admin/log-client` (futuro / backlog).

12.8 Testes de Integração

- Postman / Insomnia collection exportada em `docs/api/collection.json`.
- Scripts `npm run test:api` (supertest) cobrindo Auth + Deck CRUD + Study review.
- E2E (Playwright) cenários: Login → Criar Deck → Add Card → Publicar → Aluno Estuda → Ver Score.

12.9 Variáveis de Ambiente Alinhadas

Nome	Web	Server	Uso
VITE_API_URL	✓	-	Base frontend
PORT	-	✓	Porta API
JWT_SECRET	-	✓	Assinatura tokens
EMAIL_REMINDERS_ENABLED	-	✓	Flag lembretes

SRS_MAX_NEW_PER_DAY	-	<input checked="" type="checkbox"/>	Política estudo
---------------------	---	-------------------------------------	-----------------

Adicionar `.env.example` sincronizado e validação (Zod) no bootstrap Nest.

12.10 Riscos & Mitigação

Risco	Mitigação
Divergência DTO ↔ Types gerados	CI compara hash do schema OpenAPI
Erros silenciosos no cliente	Interceptor central + console.warn em dev
Condições de corrida em SRS	Transação lógica (find+update atômico por filtro <code>_id + version</code>)
Latência alta em listas	Paginação + índices + compressão HTTP

12.11 Critérios de Aceite Integração

- 0 referências a mocks nos módulos migrados.
 - Tipos gerados atualizados sem diffs no CI.
 - Todos erros HTTP exibidos de forma consistente (toast + boundary).
 - Fluxo Login → Estudo → Report executável ponta a ponta sem refresh manual.
 - Publicar/Retirar deck reflete em até 2s no dashboard da turma (invalidação feita).
-