🔧 Backend — Documentação Viva (GDE Mobile · MC656-2025-s2)

> **Escopo:** Este documento descreve a estrutura, decisões técnicas e progresso do **backend** do projeto GDE Mobile.
> O **crawler** exporta o banco de dados **SQLite**, e o backend realiza **leitura híbrida (geral e pessoal)** para expor **APIs REST** consumidas pelo app.
> Cada seção traz **checklists e subchecklists** para rastrear o progresso e manter a documentação viva.

---

1 Decisões de Arquitetura

- ☑ **Banco de Dados:** SQLite
  - ☑ `general.db` — dados públicos e históricos (read-only, cache in-memory)
  - ☑ `user.db` — dados pessoais por usuário (read/write local)
- ☑ **Ingestão:** Híbrida
  - ☑ Lê `general.db` (read-only)
  - ☑ Cria/usa `user.db` (dados locais do usuário)
- ☑ **Framework:** Node.js + NestJS
- ☑ **Contrato:** REST + OpenAPI (Swagger)
- ☑ **Versionamento:** Parâmetro `catalogYear` em todas as rotas
- ☑ **Cache:** TTL configurável (5–10 min)
- ☑ **Autenticação:** API-Key por ambiente
  - ☑ Rotas públicas → `general.db`
  - ☑ Rotas com dados pessoais → `user.db`
- ☑ **Paginação:** `limit/offset/sort` documentados
- ☑ **Observabilidade:** Logs estruturados e healthcheck
- ☑ **Deploy:** Docker Compose + volumes SQLite
- ☑ **Testes:** Contrato (OpenAPI) + Integração (SQLite snapshot)

---

2 Estrutura do Projeto (NestJS)

```
backend/
  src/
    main.ts
    app.module.ts
    common/
      interceptors/logging.interceptor.ts
      guards/api-key.guard.ts
      pipes/validation.pipe.ts
      filters/http-exception.filter.ts
    config/
      config.module.ts
      config.service.ts
    db/
```

```
      sqlite.module.ts
      general.repository.ts
      user.repository.ts
    modules/
      health/
      courses/
      offers/
      curriculum/
      schedule/
      attendance/
    swagger/
      swagger.ts
  .env.example
  Dockerfile
  docker-compose.yml
```

---

3 Banco de Dados

general.db

- [x] Estrutura
  - [x] `courses (code, name, type, credits, ...)`
  - [x] `offers (course_code, term, class_id, day, start, end, room, teacher, catalog_year)`
  - [x] `curriculum_nodes (course_code, category, recommended_semester, catalog_year)`
  - [x] `curriculum_edges (from_code, to_code, type, catalog_year)`
- [x] Índices
  - [x] `idx_courses_code`
  - [x] `idx_offers_year_term`
  - [x] `idx_curriculum_nodes_year`

user.db

- [x] Estrutura
  - [x] `user_plans (user_id, course_code, status, created_at)`
  - [x] `user_attendance (user_id, course_code, date, present)`
- [ ] Índices
  - [ ] `idx_user_plans_user`
  - [ ] `idx_user_attendance_course`

---

4 Rotas REST — Contrato e Checklists

4.1 Cursos

- [x] `GET /api/v1/courses`
  - [x] Params: `catalogYear, q, type, creditsMin, creditsMax, limit, offset, sort`

    - [x] Paginação e ordenação
    - [x] Cache TTL 10 min
- [x] `GET /api/v1/courses/:code`
    - [x] Detalhes por código
    - [ ] 404 se inexistente

4.2 Ofertas

- [x] `GET /api/v1/offers`
    - [x] Params: `catalogYear, term, courseCode?, day?, teacher?`
    - [ ] Filtros validados e sort whitelisted

4.3 Currículo

- [x] `GET /api/v1/curriculum`
    - [x] Params: `catalogYear, courseCode`
    - [x] Retorna grafo de integralização
- [x] `POST /api/v1/curriculum/progress`
    - [x] Params: `catalogYear`
    - [x] Body: `{ completed: [], planned: [] }`
    - [ ] Validação de cursos inexistentes

4.4 Comparação Geral × Pessoal

- [x] `GET /api/v1/curriculum/compare`
    - [x] Params: `catalogYear, userId`
    - [x] Retorna `{ missingInUserPlan, extraPlannedVsRecommended, creditGapByCategory }`
    - [ ] Detalhar formato por categoria

4.5 Grade / Conflitos

- [x] `POST /api/v1/schedule/conflicts`
    - [x] Detecta sobreposição de horários
    - [x] Retorna pares conflituosos
    - [ ] Regras para `end == start`

4.6 Faltas

- [x] `GET /api/v1/attendance/:courseCode`
- [x] `POST /api/v1/attendance/record`
    - [x] Body: `{ userId, courseCode, date, present }`
    - [ ] Garantir idempotência
- [x] `GET /api/v1/attendance/summary`
    - [x] Params: `userId, riskThreshold?`
    - [x] Default `riskThreshold = 25`

4.7 Saúde

- [x] `GET /health`
  - [x] Retorna `{ uptime, generalDbReadable, userDbWritable, version }`

---

5 Segurança e Acesso

- [x] Header `X-API-Key` obrigatório em rotas pessoais
- [x] `@Public()` define rotas abertas
- [x] API-Key validada via guard
- [ ] Rate limit (60 req/min)
- [ ] Sanitização de logs (remover API-Key)

---

6 Cache e Invalidação

- [x] TTL padrão: 10 min
- [x] Hash params como chave
- [ ] Invalidação ao trocar `general.db`
- [ ] Header `X-Force-Refresh` (modo dev)
- [ ] Logs de acerto/falha de cache

---

7 Logs e Observabilidade

- [x] LoggingInterceptor
  - [x] Rota, status, tempo
  - [x] request_id automático
- [x] Healthcheck ativo
- [x] Tratamento de erros padrão `{ error: { code, message } }`
- [ ] Métricas (contadores/latência)
- [ ] Dashboard local (Prometheus/Grafana)

---

8 Testes

- [x] Contrato (OpenAPI via Swagger)
- [x] Integração com snapshot SQLite
- [ ] Smoke test E2E: `buscar → montar grade → conflito`
- [ ] Mock API-Key nos testes

---

9 Docker e Deploy

- [x] Serviço `backend` via docker-compose
- [x] Volume `general.db` (ro)
- [x] Volume `user.db` (rw)
- [x] Porta `8080`
- [x] `.env.example`
  - [x] `API_KEY`

  - [x] `GENERAL_DB_PATH`
  - [x] `USER_DB_PATH`
  - [x] `CACHE_TTL_MS`
  - [x] `PORT`

🔟 Próximos Passos (ordenados por valor)

- [ ] Implementar `/courses` real (consulta SQLite paginada)
- [ ] Adicionar `/offers` com filtros + cache
- [ ] Implementar `/curriculum` (grafo + progress)
- [ ] Criar rota `/compare` (geral vs pessoal)
- [ ] Adicionar `/attendance` (user.db + API-Key)
- [ ] Padronizar erros e logs
- [ ] Adicionar métricas básicas
- [ ] Testes integrados e contrato
- [ ] Documentar OpenAPI final