Awesome — let’s slice your Spotify Track Explorer into tidy, portfolio-ready deliverables. Each item below has a clear goal, acceptance criteria (DoD), and suggested tech. We’ll start local (cheap + simple), then layer in AWS once the core works.

**Phase 0 — Repo + Foundations (Day-zero hygiene)**

**Goal:** Clean, professional, minimal starter you can push immediately.

* **Deliverables**
  + Monorepo (or two repos) with:
    - backend/ Spring Boot 3 + Java 21 (Maven), modular package structure:

com.yourname.trackexplorer.{domain,application,infra,web,config}

* + - frontend/ Next.js 15 (App Router, TypeScript, Tailwind + daisyUI)
    - Shared docs: README.md, ARCHITECTURE.md, CONTRIBUTING.md
  + Dev tooling: EditorConfig, Prettier, ESLint, Spotless, Checkstyle.
  + CI: GitHub Actions (build + unit tests on PR).
* **DoD**
  + mvn test and pnpm test pass in CI.
  + One-click local run instructions work on a fresh machine.

**Phase 1 — Backend Skeleton (OOP-first)**

**Goal:** Solid Spring Boot service with clean boundaries.

* **Deliverables**
  + Hexagonal architecture:
    - **Domain**: entities (User, AccountLink, Track, Playlist), value objects, domain services.
    - **Application**: use cases (LinkSpotifyAccount, GetMyPlaylists, GetMyTopTracks).
    - **Adapters**: REST controllers, persistence (JPA), Spotify client (HTTP).
    - **Infra**: JPA/Hibernate with Postgres (or H2 dev), Flyway migrations, MapStruct for DTO mapping.
  + Error handling: problem+json responses.
* **DoD**
  + Health endpoints (/actuator/health) and a dummy /api/ping.
  + Unit tests for domain + application layers; controller slice tests.

**Phase 2 — App Auth (users of**

**your**

**app)**

**Goal:** Users sign up/sign in to **your** app (not via Spotify yet).

* **Deliverables**
  + Option A (fast + AWS-ready later): **Spring Security + JWT** (password flow or magic-link).
  + Option B (AWS showpiece, minimal code): **Amazon Cognito** user pool; backend validates JWTs.
* **DoD**
  + Protected endpoint /api/me returns current user profile when authorised.
  + Postman collection (or HTTPie scripts) demonstrating login → call protected route.
  + Security tests for anonymous vs. authenticated access.

**Phase 3 — Spotify Account Linking (OAuth 2.0 Authorization Code + PKCE)**

**Goal:** A logged-in app user can **link** their Spotify account and grant scopes.

* **Deliverables**
  + OAuth flow: Next.js starts PKCE, Spring Boot callback exchanges code → stores **encrypted** tokens (access + refresh) with scopes granted (e.g., user-read-email, playlist-read-private, user-top-read).
  + Token refresh service + scheduled renewal.
  + Domain aggregate AccountLink(provider=SPOTIFY, scopes, status).
* **DoD**
  + Link/unlink endpoints + state machine (LINKED, REVOKED).
  + Integration tests with Spotify sandbox/mocked server.
  + Sensitive secrets in **.env.local** only; README explains setup.

**Phase 4 — Data Contracts + Fetch Use Cases**

**Goal:** Pull useful, portfolio-worthy views of a user’s library.

* **Deliverables**
  + Use cases:
    - GetMyPlaylists (paged, owner flags, public/private).
    - GetMyTopTracks (short/medium/long term).
    - GetAudioFeaturesForTracks (tempo, energy, valence, danceability).
  + DTOs shaped for UI (minimal, stable).
  + Caching: In-memory (Caffeine) + ETag/If-None-Match pass-through to Spotify when possible.
* **DoD**
  + Contract tests for Spotify adapter (WireMock).
  + Observed 200/304 behaviour where applicable (logged).

**Phase 5 — Frontend MVP (clean, modern, accessible)**

**Goal:** A crisp Next.js app that proves full-stack chops.

* **Deliverables**
  + Pages:
    - / Landing (explain app + “Connect Spotify” CTA).
    - /dashboard (user summary: total playlists, top artists, quick stats).
    - /playlists (grid/list with search, pagination).
    - /top-tracks (time-range filter; show audio features chips).
  + UI kit: Tailwind + daisyUI; responsive + keyboard accessible.
  + Client auth: store app JWT (or Cognito) securely; SSR where sensible; API route proxy for same-origin calls → backend.
* **DoD**
  + Lighthouse ≥ 90 (Performance/Accessibility/Best Practices).
  + Robust empty/error/loading states.
  + E2E smoke (Playwright) for login → link → see data.

**Phase 6 — Polishing the Backend (quality + scale)**

**Goal:** Production-ish behaviours without overkill.

* **Deliverables**
  + Request logging (structured JSON), correlation IDs.
  + Rate limiting (Bucket4j) on public routes.
  + Metrics: Micrometer + Prometheus endpoints; basic dashboards (locally via Docker Compose).
  + API versioning (/api/v1), OpenAPI 3 (springdoc), Swagger UI (dev only).
* **DoD**
  + OpenAPI JSON generated in CI artifact.
  + Unit test coverage threshold (e.g., 75% for app/domain).

**Phase 7 — Minimal AWS Footprint (show your cloud skills, watch costs)**

**Goal:** Deploy a lean, low-cost, professional setup.

* **Deliverables**
  + **Backend**: Containerised (Docker). Deploy to **AWS App Runner** or **ECS Fargate** (single service).
  + **DB**: **Amazon RDS PostgreSQL** (or **Aurora Serverless v2** if desired). For dev/demo: **RDS Free Tier** or **Neon**/**Render** (mention trade-offs in README).
  + **Frontend**: Deploy to **Vercel** or **AWS Amplify Hosting** (pick one and explain).
  + **Secrets**: **AWS Secrets Manager** (Spotify client secret, DB creds, JWT secret).
  + **Config**: **AWS Parameter Store** for non-secret config.
  + **Networking**: Public service with HTTPS via **ALB/ACM** (if ECS) or built-in TLS (App Runner).
  + **IaC**: **Terraform** (or CDK) for repeatable infra (separate infra/ folder).
* **DoD**
  + One command (terraform apply / “Deploy” workflow) builds the stack.
  + Smoke test URL + health check verified post-deploy.
  + Cost section in README: *how we keep it low* (min capacity, scale-to-zero where possible).

**Phase 8 — Ops & Observability on AWS**

**Goal:** Prove you can run and support it.

* **Deliverables**
  + **CloudWatch** logs + metrics; alarm for error rate/latency; SNS email notification.
  + **OpenTelemetry** exporter (optional): traces to AWS X-Ray.
  + Basic runbook in RUNBOOK.md (rotate keys, re-link Spotify, common failures).
* **DoD**
  + Intentionally induce a handled 4xx/5xx and see alarms/logs fire.

**Phase 9 — Nice-to-Haves (pick 1–2 for flair)**

**Options:**

* Batch sync job to snapshot user playlists/tracks daily (Spring @Scheduled) → show “historical changes” chart.
* Recommendation toy: simple content-based recs using audio features + cosine similarity.
* Shareable summary card (OG image) built server-side in Next.js.

**OOP & Code Quality guardrails (apply throughout)**

* **Entities**: rich domain models (no setters), invariants enforced in constructors/factories.
* **Use cases (application services)**: orchestrate, no HTTP/JPA code.
* **Ports/Adapters**: interfaces in app layer; Spotify/JPA implementations in infra.
* **Mapping**: MapStruct from domain ↔ DTOs.
* **Validation**: Bean Validation on DTOs + domain factories.
* **Testing Pyramid**: fast unit (domain), slice tests (web/data), a few end-to-end (wiremock + testcontainers).

**Suggested folder layout**

/backend

/src/main/java/com/you/trackexplorer

/config /web /application /domain /infra/{persistence,spotify,security}

/src/test/java/... (unit + slices)

pom.xml

/frontend

/src/app (Next.js App Router)

/src/components /lib /styles

tailwind.config.ts daisyui.config.ts package.json

/infra

terraform/ (or cdk/)

/docs

README.md ARCHITECTURE.md RUNBOOK.md API.md POSTMAN.json

**Milestone checklist (copy into your GitHub Projects)**

1. ✅ Foundations (repos, tooling, CI)
2. ✅ Backend skeleton + health
3. 🔐 App auth (JWT or Cognito)
4. 🎧 Spotify linking (PKCE, tokens encrypted)
5. 🧩 Fetch playlists/top tracks/audio features + cache
6. 🖥️ Next.js MVP UI (dash, playlists, top tracks)
7. 📈 API docs, metrics, rate limiting, tests ≥ 75%
8. ☁️ AWS deploy (App Runner/ECS, RDS, Secrets Manager)
9. 🔍 CloudWatch/X-Ray + alarms + runbook
10. ✨ One nice-to-have (pick)

If you want, I can drop starter templates for Phase 0–1 (pom.xml, package structure, Next.js scaffold, CI yml) in a canvas next.