# Playbook — Turning the Path‑to‑Eagle Tracker into a Multi‑User Web App

This is a step‑by‑step build plan you can hand to your code assistant (e.g., GitHub Copilot / Codeium / “Codex”) to take the Excel tracker online for **multiple Scouts**, **leaders**, and **admins** with authentication, approvals, and permissions.

Goal: A secure web app where Scouts and authorized adult leaders can view/update rank progress; leaders can sign off requirements; an Admin (you) approves leaders’ access to specific Scouts. Print/export remains easy (Progress Report & Gantt).

## 0) TL;DR Implementation Path

1. **Scaffold** a modern stack (Next.js/React + TypeScript + Prisma + Postgres + NextAuth) with server‑side rendering for secure pages.
2. **Model data** for users/roles, scouts, leader links, requirement progress, merit badges, approvals, and audit trails.
3. **Auth** via email+password (and optional OAuth) + **email verification**; **RBAC** (Admin, Leader, Scout).
4. **Admin flow**: approve leaders and attach them to one or more Scouts.
5. **Leader flow**: sign off items; initials auto‑populate from profile; approvals are auditable.
6. **Scout flow**: view/edit own data; cannot self‑approve.
7. **Exports**: Generate print‑ready PDF and an Excel export mirroring the current workbook.
8. **Ship** to a managed platform (Vercel + Neon/Render/Cloud SQL).
9. **Hardening**: PII guardrails, rate limiting, logging, backups, and tests.

## 1) Tech Stack (Opinionated, but easy to swap)

* **Frontend/SSR**: Next.js 14 (App Router) + React + TypeScript
* **UI**: Tailwind CSS + shadcn/ui + Recharts (for Gantt‑like bars)
* **Auth**: NextAuth (email/password + email link; optional Google/Microsoft)
* **DB**: Postgres (Neon/Render/Cloud SQL). ORM: Prisma.
* **File Export**: exceljs for .xlsx; @react-pdf/renderer or playwright print‑to‑PDF for Progress Report.
* **Email**: Resend/SendGrid/SES (verification, notifications)
* **Infra**: Vercel (web) + Hosted Postgres; storage for generated files (S3/R2) if needed.
* **Observability**: Audit log table + console log shipping (e.g., Logtail/Datadog), Sentry for errors.

Alternative: Django + DRF + HTMX or Vue; or Rails. The core concepts and schema below still apply.

## 2) Data Model (Prisma‑style schema sketch)

Adjust names/fields to your style. Add createdAt/updatedAt everywhere (omitted for brevity). All IDs are UUID.

model User {  
 id String @id @default(uuid())  
 email String @unique  
 emailVerified DateTime?  
 passwordHash String? // null if using OAuth only  
 firstName String  
 lastName String  
 phone String?  
 role Role // 'ADMIN' | 'LEADER' | 'SCOUT'  
 bsaId String? // SID# for leaders or scouts  
 initials String? // leaders’ approval initials  
 // Relations  
 scoutProfile Scout? @relation(fields: [scoutProfileId], references: [id])  
 scoutProfileId String?  
 leaderLinks LeaderScoutLink[]  
 approvals Approval[]  
}  
  
enum Role { ADMIN LEADER SCOUT }  
  
model Scout {  
 id String @id @default(uuid())  
 userId String @unique // link to the User record with role=SCOUT  
 unit String?  
 council String?  
 dob DateTime?  
 currentRank Rank? // redundant convenience field  
 // Relations  
 progress RankProgress[]  
 badges MeritBadgeProgress[]  
 leaders LeaderScoutLink[]  
}  
  
enum Rank { SCOUT TENDERFOOT SECOND\_CLASS FIRST\_CLASS STAR LIFE EAGLE }  
  
model LeaderScoutLink { // Admin‑approved access for a leader to a scout  
 id String @id @default(uuid())  
 leaderId String  
 scoutId String  
 status LinkStatus // PENDING | APPROVED | REVOKED  
 approvedBy String?  
 approvedAt DateTime?  
 @@unique([leaderId, scoutId])  
}  
  
enum LinkStatus { PENDING APPROVED REVOKED }  
  
model Requirement { // Master catalog per rank (seed data)  
 id String @id @default(uuid())  
 code String // e.g., TF-6b  
 rank Rank  
 text String  
 durationDays Int? // e.g., 30 for TF-6b  
 durationMonths Int? // e.g., 4 for Star tenure  
 order Int  
}  
  
model RankProgress { // One row per requirement instance per Scout  
 id String @id @default(uuid())  
 scoutId String  
 requirementId String  
 startDate DateTime?  
 eligibleDate DateTime? // auto from duration + startDate  
 doneDate DateTime?  
 notes String?  
 approvedById String? // leader User.id who approved  
 approvedAt DateTime?  
 // denormalized leader initials for history  
 approvedByInitials String?  
}  
  
model MeritBadgeProgress {  
 id String @id @default(uuid())  
 scoutId String  
 badgeName String  
 isEagleReq Boolean  
 dateStarted DateTime?  
 dateDone DateTime?  
 notes String?  
}  
  
model BoardOfReview { // Timeline entries per rank  
 id String @id @default(uuid())  
 scoutId String  
 rank Rank  
 previousRankBORActual DateTime? // actual date of prior BOR  
 thisRankBORActual DateTime?  
 thisRankBORProjected DateTime? // elig + buffer  
}  
  
model Setting { // global inputs  
 id String @id @default(uuid())  
 key String @unique  
 value String  
}  
  
model AuditLog {  
 id String @id @default(uuid())  
 actorId String?  
 action String  
 entity String // e.g., RankProgress, LeaderScoutLink  
 entityId String  
 metadata Json?  
 at DateTime @default(now())  
}

**Seed data**: populate Requirement with the condensed list you used in Excel (codes, rank, text, durations). This powers forms and calculations.

## 3) AuthN & AuthZ (step‑by‑step)

1. **NextAuth** with credentials provider (email+password) **and** optional OAuth (Google/Microsoft).
2. **Email verification**: require verified email for all roles before access to protected routes.
3. **RBAC**: Roles in JWT/session (role: 'ADMIN'|'LEADER'|'SCOUT').
4. **Route guards**:
   * ADMIN: manage users, approve leader↔scout links, revoke access, view all.
   * LEADER: view/update Scouts they’re linked to with status=APPROVED.
   * SCOUT: view/update **own** data only; cannot approve.
5. **Session storage**: JWT + httpOnly secure cookies. Refresh as per NextAuth defaults.
6. **Rate limiting** on auth endpoints (e.g., @upstash/ratelimit).

## 4) Admin Approval Flow

**Happy path** 1. Leader signs up → role=LEADER (unprivileged).  
2. Leader requests access to a Scout (enter Scout’s email or Admin assigns from Admin UI).  
3. System creates LeaderScoutLink{status:PENDING}.  
4. Admin dashboard shows pending requests; Admin approves → status=APPROVED, sets approvedBy, approvedAt.  
5. Leader now sees and can sign off items for that Scout.

**Controls** - Admin can **revoke** a link (status=REVOKED).  
- All changes write to AuditLog.

## 5) Core Workflows & Pages

### 5.1 Onboarding

* **/signup (Scout/Leader)**: collect name, email, password; verify email. If Scout, optionally capture unit/council; if Leader, capture initials and BSA ID.
* **/admin/users**: view all users, set role, reset passwords (via email link), deactivate.

### 5.2 Scout Dashboard

* **/app/scout**: header panel (Profile fields), progress summary tiles, “Next 3 Steps”, and **rank cards** (Scout→Eagle).
* Each rank card links to **/app/scout/rank/[rank]** with requirement table: Start/Done, auto Eligible, Approved By (disabled for Scouts), Notes.
* **Print/Export**: button to generate **Progress Report (PDF)** and **Excel** export.

### 5.3 Leader Dashboard

* **/app/leader**: list of assigned Scouts (search/filter).
* Drill‑down to **/app/leader/scout/[id]** → same views as Scout but with **Approve** controls on each requirement row.
* Approval sets approvedById, approvedAt, and stores leader’s initials snapshot.

### 5.4 Admin Dashboard

* **/app/admin**:
  + **Pending approvals** (Leader↔Scout links) with Approve/Revoke.
  + Manage users/roles, reset MFA (if used), deactivate.
  + Global settings (BOR buffer days), backups, and audit log viewer.

## 6) Calculations & Business Rules (server‑side services)

Create a small service layer (e.g., /lib/services/progress.ts) with pure functions:

* eligibleDate(requirement, startDate) → date | null
  + durationDays → start + N days
  + durationMonths → EDATE(start, N) (use JS date add months with day clamping)
* rankEligibility(scoutId, rank) → max of all eligibleDate and doneDate for that rank’s requirements.
* projectedBOR(rankEligibility, bufferDays) → eligibility + bufferDays.
* eagleBadgeGates(scoutId) → {date14Required, date21Total}.
* eagleEligibility(scoutId) → max(active6mo, leadership6mo, date14Required, date21Total, project milestones).

Recompute upon change (webhooks or server actions). Cache per‑rank summaries for speed.

## 7) API (REST-ish examples; or use Next.js Server Actions)

POST /api/auth/signup  
POST /api/auth/signin  
POST /api/auth/verify-email  
  
GET /api/scouts/:id  
PATCH /api/scouts/:id // profile updates  
GET /api/scouts/:id/progress?rank=FIRST\_CLASS  
PATCH /api/progress/:progressId // start/done/notes  
POST /api/progress/:progressId/approve // leader only  
  
GET /api/badges/:scoutId  
PATCH /api/badges/:badgeId  
  
GET /api/bor/:scoutId  
PATCH /api/bor/:scoutId // actual BOR dates  
  
POST /api/links // leader requests access to a scout  
GET /api/links/pending // admin  
POST /api/links/:id/approve // admin  
POST /api/links/:id/revoke // admin  
  
GET /api/export/excel/:scoutId // returns xlsx  
GET /api/export/progress-pdf/:scoutId

Secure every route with role checks and link checks (LeaderScoutLink.APPROVED).

## 8) UI Components (shadcn/ui + Tailwind)

* **HeaderCard**: shows Name, SID#, Age, Phone, Unit, Council, Current Rank.
* **RankTable**: requirement rows (Start, Eligible, Done, Approved By, Notes). Disabled approve control for Scouts.
* **ProgressReport**: printable layout; share a route /app/scout/[id]/report with a print stylesheet.
* **Gantt**: Recharts Bar chart (Start→Projected BOR).
* **LeaderAccessModal**: request access to a Scout (by email or code).
* **AdminQueues**: Approvals list with filters and bulk actions.

## 9) Security & Privacy (must‑do)

* **PII**: minimize; don’t expose DOB or contact info to users without a valid link.
* **Authorization**: enforce on **server** (not just client UI).
* **Audit logging** of approvals and edits (who/when/what).
* **Email verification** required before any access.
* **Rate limit** sensitive endpoints; lockout after repeated failed logins.
* **Backups**: daily DB backups + test restore.
* **Logging**: centralize errors and security events; alerting for unusual activity.
* **Legal**: follow your organization’s youth protection guidelines; obtain guardian consent as required.

## 10) Exports & Printing

* **Excel (.xlsx)**: Use exceljs to mirror the Overview, Rank tables, and Badge gates. Optionally attach a watermark or footer timestamp.
* **PDF**: SSR a print view (Progress Report & Gantt) and use @react-pdf/renderer **or** playwright headless print‑to‑PDF.

## 11) Deployment

* **Web**: Vercel (preview deployments on PRs).
* **DB**: Neon/Render/Cloud SQL with IP allow‑listing and TLS.
* **Secrets**: Vercel env vars; rotate on admin password resets.
* **Domains**: custom domain + enforced HTTPS (HSTS).

## 12) Testing Checklist

* **Unit**: calculation functions for every duration scenario; RBAC guards.
* **Integration**: approve flow (pending→approved→revoked).
* **E2E**: Scout flow, Leader approval/sign‑off, Admin revocation.
* **Security**: attempt unauthorized access to another Scout’s data → must 403.

## 13) CI/CD

* GitHub Actions: lint, typecheck, unit tests on PR; deploy Preview on Vercel.
* Migration gate: run prisma migrate deploy on production only after backups succeed.

## 14) Build Order (done‑for‑you task list)

1. Scaffold Next.js + TS + Tailwind + Prisma + NextAuth + Postgres.
2. Implement Prisma schema; seed Requirement catalog (codes, durations).
3. Auth pages + email verification; add RBAC to session.
4. Admin dashboard: approve LeaderScoutLink requests.
5. Scout dashboard: Profile header, rank list, requirement table (read/write), calculations.
6. Leader dashboard: same, with **Approve** controls.
7. Badge pages + Eagle gates logic.
8. Overview page + Gantt chart.
9. Print view + PDF; Excel export endpoint.
10. Logging/Audit + rate limiting + backups.
11. E2E tests; security pass; deploy.

## 15) Nice‑to‑Haves

* **Notifications**: email when a Leader requests access; when a requirement is approved.
* **POR log**: fine‑grained leadership positions and durations.
* **Counselor fields** on badges (name/phone/email).
* **Bulk import** from existing spreadsheets.
* **Dark mode / theming** with unit colors.

### Snippet: Server‑side eligibility helpers (TypeScript)

export function addMonths(d: Date, months: number) {  
 const dt = new Date(d);  
 const day = dt.getDate();  
 dt.setMonth(dt.getMonth() + months);  
 // Clamp day for short months  
 if (dt.getDate() < day) dt.setDate(0);  
 return dt;  
}  
  
export function eligibleDate({ durationDays, durationMonths }: { durationDays?: number; durationMonths?: number }, start?: Date) {  
 if (!start) return undefined;  
 if (durationDays) return new Date(start.getTime() + durationDays \* 24 \* 60 \* 60 \* 1000);  
 if (durationMonths) return addMonths(start, durationMonths);  
 return undefined;  
}

### Snippet: API guard (Leader may update only assigned Scouts)

async function assertLeaderCanAccess(leaderId: string, scoutId: string, prisma: PrismaClient) {  
 const link = await prisma.leaderScoutLink.findUnique({  
 where: { leaderId\_scoutId: { leaderId, scoutId } },  
 });  
 if (!link || link.status !== 'APPROVED') throw new Error('Forbidden');  
}

## 16) Handoff Notes for Your Codex Assistant

* Maintain parity with Excel logic for durations (30‑day/4‑week streaks) and eligibility MAX rules.
* Keep an **immutable** audit trail for approvals (don’t overwrite initials—store snapshots).
* Enforce all permissions on the **server**.
* Expose exports behind auth and per‑Scout access checks.

**Done right, you’ll have:** a secure, auditable, role‑aware web app that mirrors—and improves—the Excel experience, scales to many Scouts, and keeps adult access under your explicit Admin control.