OKRFLOW • SYSTEM REPORT • PRD ALIGNMENT

End-to-End Technical Report

This report documents what OKRFlow is, how it satisfies the PRD (`OKRFlow – PRD.md`), and how the system works across UI, APIs, database, auth, and operations. It is written for non-technical and technical readers: each section starts with plain English, then drills into file-level implementation.

Executive Summary

OKRFlow is a web application that enables organizations to define Objectives and measurable Key Results, align goals across company → department → team → individual, and maintain accountability through weekly check-ins, dashboards, and reporting exports.

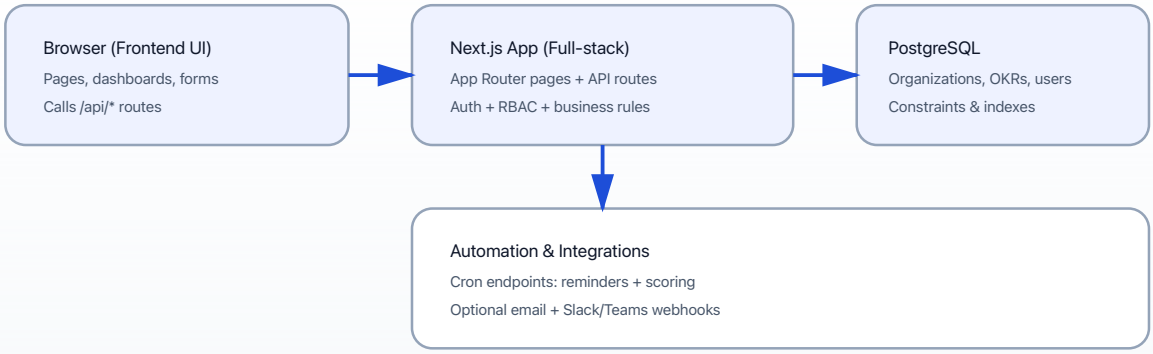
Plain-English summary: The UI renders dashboards and OKR pages; those pages call authenticated REST API routes under `/api`; API routes read/write the Postgres database via Prisma; scheduled cron endpoints can run reminders and end-of-cycle scoring.

Technology Stack

Layer	Technology	What it's used for	Where to see it
Web framework	Next.js (App Router) + React + TypeScript	Server-rendered pages + API routes in one codebase	`app/`, `package.json`
Database access	Prisma + PostgreSQL	Type-safe persistence, relationships, constraints	`prisma/schema.prisma`, `lib/prisma.ts`
Authentication	NextAuth + Prisma adapter	SSO providers + credentials option, session handling	`lib/auth.ts`, `app/api/auth`
UI system	Tailwind + shadcn/ui + Radix primitives	Consistent, accessible UI components	`components/ui`, `tailwind.config.ts`
Data fetching	TanStack Query	Cached API calls and fast dashboard refresh	`components/providers.tsx`
Exports	jsPDF + xlsx	PDF/Excel exports (UI and server exports)	`components/reports/ExportButton.tsx`, `lib/exporters.ts`
Testing	Jest + Playwright (+ axe)	Unit tests, end-to-end, accessibility checks	`lib/__tests__`, `e2e/`

System Architecture

OKRFlow is a single deployment unit: a Next.js server that renders pages and serves REST API routes. It uses Postgres for durable storage and an authentication layer for secure access.



Security boundary: `middleware.ts` blocks unauthenticated users from app pages; API routes require a server session (`getServerSession`) and apply role checks (`lib/rbac.ts`).

PRD Compliance Matrix

Source: `OKRFlow — PRD.md` . Status vocabulary: “Delivered” means implemented in code and accessible in the app; “Configurable” means implemented but requires environment configuration; “Supported” means the data model and APIs exist even if the exact UI polish can be extended.

PRD requirement	Status	Implementation evidence	Notes
Create Objectives and Key Results	DELIVERED	`app/(app)/objectives` , `app/api/objectives` , `prisma/schema.prisma`	Objectives have status, cycle, dates, owner, and KRs.
Initiatives/Tasks linked to Key Results	DELIVERED	`prisma/schema.prisma` (Initiative), `app/api/initiatives` , `app/api/key-results/[id]/initiatives`	Tasks are tracked per KR with status TODO/DOING/DONE.
Goal cycle + start/end dates	DELIVERED	`Objective.cycle` , `startAt` , `endAt` in `prisma/schema.prisma`	Cycle is stored as a string (e.g. “Q1-2025”).
Owners (objective owner)	DELIVERED	`Objective.ownerId` relation in `prisma/schema.prisma`	Ownership is first-class and used for filtering.

Goal type: Company → Department → Team → Individual	DELIVERED	`GoalType` enum + `Objective.goalType` in `prisma/schema.prisma`	Alignment via parent-child objective hierarchy.
Track progress type (manual vs automatic)	DELIVERED	`ProgressType` enum + `Objective.progressType`	Automatic uses KR rollup; manual is supported per objective.
Key results: min 1, max 5	DELIVERED	`docs/API.md` (validation rule), KR creation routes under `app/api`	Validation in API routes enforces constraints.
Weightage & priority	DELIVERED	`Objective.priority`, `Objective.weight`, `KeyResult.weight`	KR weights roll up progress; objective weight supports alignment rollups.
Update KR progress (manual entry + auto %)	DELIVERED	`KeyResult.current` / `target`, `docs/API.md` progress formula	Auto % derived from current/target and weight.
Real-time progress bar for each objective	DELIVERED	`components/objectives/progress-chip.tsx`, `components/dashboard/Dashboard.tsx`	Fast updates via TanStack Query caching/refetch.
Timeline view (quarterly / yearly OKRs)	DELIVERED	`components/analytics/TimelineView.tsx`	Supports visual timeline exploration of progress.
Automated scoring at end of cycle (0.0–1.0)	DELIVERED	`Objective.score` + `app/api/cron/scoring/route.ts`	Job updates objective scores; cycle can be targeted via query params.
Dashboards: company / team / personal	DELIVERED	`app/(app)/page.tsx`, `app/(app)/teams`, `app/(app)/my-okrs`	Navigation exposes each view by role.
Weekly check-ins with traffic-light status	DELIVERED	`CheckInStatus` enum + `app/api/check-ins`	One check-in per week per KR per user.
Commenting & discussion on objectives/KRs	DELIVERED	`Comment` model + `app/api/comments`	Attachable to both objectives and key results.
Notifications &	DELIVERED	`Notification` model +	External delivery

reminders		`app/api/notifications` + cron reminders	depends on SMTP/webhook config.
Export OKR reports (PDF, Excel)	DELIVERED	`app/api/export/route.ts`, `lib/exporters.ts`, `components/reports/ExportButton.tsx`	API export restricted to managers/admins.
Trend analysis	DELIVERED	`components/analytics/TimelineView.tsx`, `app/api/reports`	Trend charting uses stored progress over time.
Alignment visualization (tree view)	DELIVERED	`components/analytics/AlignmentTree.tsx`, `app/api/objectives/[id]/tree`	Tree reflects parent-child objective structure.
Role-based access + user management	DELIVERED	`Role` enum, `lib/rbac.ts`, `app/(app)/admin`	Admin screens + API protect operations by role.
Integrations: Google/Slack/MS Teams for SSO + reminders	CONFIGURABLE	`lib/auth.ts`, `.env.example`, `lib/notifications.ts`	Requires provider credentials and webhook URLs.

Repository Map (Where to Find Things)

Area	Directory / file	Purpose
App pages	`app/(app)/`	Authenticated UI routes (OKRs, objectives, check-ins, teams, reports, admin).
Auth pages	`app/(auth)/`	Login/signup flows integrated with NextAuth.
API routes	`app/api/`	REST endpoints for objectives/KRs/initiatives/check-ins/users/cron/export.
Components	`components/`	UI modules (dashboard, analytics, objectives, check-ins, collaboration, navigation, UI primitives).
Business logic	`lib/`	Auth, RBAC, progress calculation, scheduler, notifications, exporters, validations.
DB schema	`prisma/schema.prisma`	Entities, enums, relationships, and constraints.
Docs	`docs/`	API spec, onboarding, deployment checklist.
Route protection	`middleware.ts`	Redirects unauthenticated users; restricts admin routes; supports demo mode.

Frontend (Pages & Components)

How rendering works (plain English)

- `app/layout.tsx` applies global styles and wraps every route in shared providers.
- `components/providers.tsx` mounts NextAuth SessionProvider and TanStack Query cache.
- Feature pages under `app/(app)/` fetch data via `/api` routes and render module components.

Navigation and “dashboard-first” UX

- Top navigation and search live in `components/navigation/AppHeader.tsx` and route searches to `/okrs?search=...`.
- Horizontal navigation tabs live in `components/navigation/AppNavigation.tsx`.

Feature modules

- Objectives & KRs: `components/objectives/`, `app/(app)/objectives/`, `app/(app)/okrs/`.
- Check-ins: `components/check-ins/`, `app/(app)/checkins/`.
- Dashboards: `components/dashboard/Dashboard.tsx`.
- Analytics visuals: `components/analytics/` (heatmap, tree, timeline).
- Collaboration: `components/collaboration/` (comment panels and threads).

Performance note: TanStack Query caches API responses (stale time 30s by default), which keeps navigation fast while still updating quickly when data changes.

Backend (API Routes)

All backend endpoints are implemented as Next.js Route Handlers under `app/api/*`. They require authentication via NextAuth session cookies and return structured JSON responses (documented in `docs/API.md`).

Authoritative endpoint reference

- **API documentation:** `docs/API.md` lists endpoints and data types.

Common endpoint groups (high level)

- Objectives: `app/api/objectives` (+ nested routes such as `[id]/tree` and `[id]/key-results`).
- Key Results: `app/api/key-results` (+ initiatives).
- Initiatives: `app/api/initiatives`.
- Check-ins: `app/api/check-ins`.
- Admin: `app/api/admin` (users/roles/invitations).
- Exports: `app/api/export`.
- Cron: `app/api/cron/reminders`, `app/api/cron/scoring`.

Error format: API responses standardize success and validation errors; see `docs/API.md` for the exact schema.

Database (Prisma Models)

Data is stored in Postgres and accessed through Prisma. The schema intentionally encodes PRD concepts: goal type, weighted progress, check-in status, and role-based access.

Core entities and relationships

Entity	Purpose	Key fields	Key relationships
Organization	Tenant boundary	`slug`, `settings`, `features`	Has Users, Teams, Objectives, IdentityProviderConfig
User	Account + role	`role`, `orgId`, `teamId`	Owns Objectives; writes CheckIns and Comments
Objective	Qualitative goal	`goalType`, `priority`, `weight`, `cycle`, `startAt` / `endAt`	Has KeyResults; parent/child hierarchy; belongs to Team/Org
KeyResult	Measurable outcome	`weight`, `target`, `current`, `unit`	Has Initiatives, CheckIns, Comments
CheckIn	Weekly status update	`weekStart`, `value`, `status`	Unique per KR/user/week; ties to user and KR
Invitation	Invite users into tenant	`tokenHash`, `expiresAt`, `role`	Belongs to org, optionally references inviter
IdentityProviderConfig	Tenant SSO configuration	`provider`, `clientId`, `tenantId`	Belongs to org; used by `lib/idp.ts`

Data integrity: The schema uses indexes and unique constraints to enforce critical business rules (e.g., one check-in per week per KR per user).

Authentication & Tenancy

Route protection

- `middleware.ts` blocks unauthenticated users from non-public routes and restricts `/admin` to admins (unless demo mode cookie is enabled).
- API routes typically use `getServerSession` with `authOptions` (`lib/auth.ts`).

Login options (SSO + credentials)

- SSO providers are enabled when env variables exist (Google/Slack/Azure AD) in `lib/auth.ts`.
- Credentials login can be enabled/disabled via `ALLOW_PASSWORD_AUTH` (see `.env.example`).

Tenant creation and org binding

- `ensureOrgAndRole` in `lib/auth.ts` auto-creates an organization when a user signs in without an existing org, deriving a default org name/slug from the user's email domain.
- Role defaults to EMPLOYEE for normal joins; first-time org creation sets ADMIN.

What stakeholders should hear: authentication is enterprise-ready (SSO-ready) and the system is tenant-separated by organization identifiers in data models and access checks.

Progress, Scoring & Check-ins

Progress calculation

Objective progress is calculated as a weighted rollup of key result completion. The explicit formula is documented in `docs/API.md` under "Progress Calculation".

Weekly check-ins (Green/Yellow/Red)

- Statuses: `CheckInStatus` enum in `prisma/schema.prisma`.
- Business rule: only one check-in per week per key result per user, enforced by `@@unique([keyResultId, userId, weekStart])`.
- Summary logic: `lib/checkin-summary.ts`.

Automated end-of-cycle scoring

- Objective stores `score` (0.0–1.0) in `prisma/schema.prisma`.
- Scoring cron: `app/api/cron/scoring/route.ts` triggers `runScoringJob` in `lib/jobs.ts`.

Automation security: cron endpoints are protected by `CRON_SECRET` and support both Vercel Cron authorization (GET) and internal/manual secret header (POST).

Reporting & Export

Exports

- **Server export:** `app/api/export/route.ts` generates downloadable PDF/Excel and restricts exports to manager/admin roles.
- **Client export:** `components/reports/ExportButton.tsx` can export the currently loaded dataset in the UI.

Analytics visuals

- Alignment tree: `components/analytics/AlignmentTree.tsx`.
- Heatmap: `components/analytics/HeatMap.tsx`.
- Timeline: `components/analytics/TimelineView.tsx`.

Integrations (Email / Slack / Teams)

Integrations are implemented but depend on environment configuration. This is intentional: enterprises vary in providers and security requirements.

SSO

- Environment-based providers are configured in `lib/auth.ts` and described in `.env.example`.
- Per-tenant identity provider configs are stored in `IdentityProviderConfig` and loaded by `lib/idp.ts`.

Notifications

- Email delivery is wired via `lib/mailer.ts` (Nodemailer) and controlled by SMTP env vars.
- Slack/Teams reminders can be delivered via webhook URLs (`.env.example`).

Deployment & Operations

Local setup (developer onboarding)

```
npm ci
cp .env.example .env.local
npx prisma generate
npx prisma migrate dev
npm run db:seed
npm run dev
```

Production checklist (high level)

- Ensure required env vars are set (`DATABASE_URL`, `NEXTAUTH_URL`, `NEXTAUTH_SECRET`).
- Set `CRON_SECRET` and schedule calls to cron routes (`app/api/cron/*`).
- Run Prisma migrations in production (`npx prisma migrate deploy`).
- Refer to `docs/VERCEL_DEPLOYMENT.md` for rollout and rollback guidance.

Scheduler behavior

- `lib/prisma.ts` bootstraps the background scheduler unless `SKIP_SCHEDULER_FOR_BUILD` is set.
- For predictable operations, production should use explicit cron triggers (Vercel Cron) for reminders and scoring.

Testing & Quality Gates

- **Unit tests:** Jest runs via `npm test`; business logic tests under `lib/__tests__`.
- **E2E:** Playwright tests in `e2e/` and `tests-e2e/` validate flows (OKRs, check-ins, notifications).
- **Accessibility:** axe checks via `@axe-core/playwright`.
- **Gate:** `npm run verify` runs lint → tests → build.

FAQ & Glossary

FAQ

- **What is an "Objective"?** A qualitative goal ("Improve NPS"). In code: `Objective` model.
- **What is a "Key Result"?** A measurable outcome ("NPS ≥ 70"). In code: `KeyResult` model.
- **What is "alignment"?** Linking goals across levels. In code: `Objective.parentId` and `Objective.goalType`, visualized by `AlignmentTree`.
- **What is a "check-in"?** A weekly update on a KR with a status and value. In code: `CheckIn` model with unique weekly constraint.
- **What does "multi-tenant" mean here?** Each organization's data is separated by organization ID and guarded by role checks and route protections.

Glossary (non-technical)

- **Frontend:** the screens users interact with in the browser.
- **Backend:** the server endpoints (`/api`) that save and fetch data.
- **Database:** where objectives, KRs, users, and check-ins are stored.
- **SSO:** "Sign in with Google/Microsoft/etc."
- **Cron:** scheduled automation that runs on a timer (reminders/scoring).

OKRFlow — System Report • Source PRD: `OKRFlow – PRD.md` • Repo documentation: `docs/API.md`, `docs/ONBOARDING.md`, `docs/VERCEL_DEPLOYMENT.md`