Developer How-To Guide

Oxford House Expense Tracker

Version 1.0 | January 2026

Overview

This guide summarizes the developer workflow, system architecture, and the core files used to maintain the Oxford House Expense Tracker. It is designed as a quick reference and points to deeper documentation stored in `docs/developer/`.

> **Quick Start (Local Dev)**
>
> - Backend: `cd admin-web/backend && npm start` (http://localhost:3002)
> - Web portal: `cd admin-web && npm start` (http://localhost:3000)
> - Mobile (Expo): `npm start`

Primary Architecture References

Document	Purpose
`docs/developer/ARCHITECTURE_CLEAN.md`	Clean architecture overview, data flow, and sync model
`docs/developer/PROJECT_STRUCTURE.md`	Full repository map with folder-by-folder description
`docs/developer/DATABASE_MANAGEMENT_GUIDE.md`	Database schema and migration references
`docs/developer/PRODUCTION_GO_LIVE_PLAN.md`	Production rollout checklist and validation steps

Repository Map (High Level)

- `src/` - Mobile app (Expo)
- `admin-web/` - Web portal (React)
- `admin-web/backend/` - Backend API (Express + SQLite)
- `docs/` - Documentation and guides
- `assets/` - Brand assets and images

For a full directory walkthrough, see `docs/developer/PROJECT_STRUCTURE.md`.

System Components

Mobile App (React Native / Expo)

- Source: `src/`
- Core entry: `App.tsx`
- Local data: `src/services/database.ts`
- API base URL: `src/config/api.ts`

Admin Web Portal

- Frontend: `admin-web/src/`
- Theme and branding: `admin-web/src/App.tsx`

Backend (Render)

- Server entry: `admin-web/backend/server.js`
- Routes: `admin-web/backend/routes/`
- Database: SQLite (backend server)
- WebSocket: real-time updates to web portal

Data & Sync

Database Architecture

Mobile App (SQLite)	Backend (SQLite)
employees, mileage_entries, receipts, time_tracking, daily_descriptions, daily_odometer_readings, saved_addresses, per_diem_rules, monthly_reports, weekly_reports, biweekly_reports, cost_center_summaries, current_employee	employees, cost_centers, per_diem_rules, monthly_reports, weekly_reports, biweekly_reports, report_status (legacy)

The mobile app is the source of truth for mileage entries, receipts, and hours. The backend stores employee/admin data and report workflow metadata.

Mobile Local DB

The mobile app stores data in SQLite for offline-first behavior. Sync services push changes to the backend and pull updates back into the local store.

Backend Sync

The backend provides REST endpoints and WebSocket updates for real-time sync. See `docs/developer/ARCHITECTURE_CLEAN.md` for a full flow diagram.

Sync Flow (Write)

```
Mobile action
  → Save to local DB
  → SyncIntegrationService queue
  → Auto-sync every 5 seconds
```

```
  → POST to backend API
  → WebSocket broadcast to web portal
```

Sync Flow (Read)

```
App launch
  → ApiSyncService.syncFromBackend()
  → Fetch employees / time / descriptions / per diem rules
  → Store in local DB
  → UI refresh
```

What Syncs

From Mobile to Backend	From Backend to Mobile
mileage entries, receipts, time tracking, daily descriptions, employee updates	employees list, time tracking updates, daily descriptions, per diem rules

Web Portal Data Access

- Staff portal reads mobile data via API and submits monthly reports
- Supervisor portal reviews and approves/rejects reports
- Admin portal manages employees, cost centers, per diem rules

> **Tip**
>
> When testing sync behavior, verify both the local DB changes and the backend response logs. This catches silent failures early.

Data Integrity Rules

- Preserve IDs when syncing; generate new IDs only for brand-new entries.
- Always validate employeeId before saving records.
- Use YYYY-MM-DD date format and avoid timezone shifts.
- Check for duplicates by ID and data signature before insert.

Best Practices

Do

- Save to local DB first, then rely on auto-sync.
- Use Promise.all for parallel DB queries.
- Refresh screens via useFocusEffect without re-syncing.
- Log important operations in backend routes and sync services.

Don't

- Call backend APIs directly from screens for write operations.
- Sync on every screen focus (avoid duplicate writes).
- Store the same data in multiple places.

Monitoring & Debugging

Mobile Log Markers

✅ success • ❌ error • 🔄 syncing • 📥 downloading • 📤 uploading • 💾 saving

Backend Observability

- Check server logs for sync queue activity and WebSocket broadcasts.
- Inspect SQLite with a viewer if data appears missing in portals.

Release & Deployment

- Expo OTA updates: `eas update --branch main`
- Android internal testing: EAS build + Play Console upload
- iOS TestFlight: Requires Apple Developer account

This document is part of the Oxford House Expense Tracker documentation suite.

For technical support, please contact your system administrator.

Document Version: 1.0 | Last Updated: January 2026