

Bill Organizer (BILLZ) - Buyer Documentation

Shipped version: 1.0.61 (versionCode 184) - Android, live on Google Play.

This document describes what you received. For step-by-step setup, see BUYER-SETUP.md.

1. Overview

Bill Organizer is a full-stack personal-finance app that answers one question: will my money last until payday? Users track bills they pay (payable) and money they are owed (receivable), import bank statements, keep a transaction ledger, set budgets, goals and debt payoff plans, read spending reports and LLM-written spending insights, and see what is due on a calendar. The app is deliberately bank-free: there is no account linking - you enter bills by hand or import a statement.

You have the complete, unencrypted source for the mobile app, the backend API and the admin portal, and you can rebrand and relaunch it as your own product.

2. What is included in the package

- app/, src/, constants/, modules/, assets/, shims/ - the Expo / React Native app
- backend/ - FastAPI + MongoDB API (bills, ledger, statement import, insights, Google Play IAP verification, admin portal, email)
- android/ - the native Android project
- scripts/, mock/ - helper scripts and sample bank statements for demos/tests
- package.json, app.json, eas.json, metro.config.js - build configuration
- BUYER-SETUP.md - step-by-step deployment and rebrand guide
- PACKAGE-MANIFEST.txt - exactly what was included and excluded

3. Technology stack

Frontend

- Expo SDK 54, React Native 0.81, React 19, Expo Router 6, TypeScript
- Charts via react-native-gifted-charts / react-native-svg
- Local native modules: modules/exact-alarms (exact-alarm scheduling bridge) and modules/widget-bridge

Backend

- FastAPI (Python 3.12), MongoDB via the async motor driver, JWT auth
- bcrypt/passlib password hashing, pydantic models
- Statement parsing: CSV / QIF / OFX and PDF text extraction (pypdf), with OCR for scanned PDFs

Integrations

- Google Play in-app purchase verification via the Android Publisher API
- AdMob (banner + rewarded) for the free tier
- AI insights via a multi-provider chain: OpenRouter / DeepSeek primary, Google Vertex AI (Gemini) fallback, DeepSeek direct as a second fallback
- Gmail SMTP for password-reset and signup-code email

Build / release

- EAS build and OTA updates (expo-updates); Kotlin 2.2.20; Android target SDK 35

4. Architecture and main user flows

Architecture

- The React Native app talks to the FastAPI backend over a JWT-authenticated REST API. MongoDB is the datastore.
- Every record is owner-scoped by user_id and device_id. A middleware reads the client's local date (X-Local-Date) so server-side "today" and report windows match the user's phone rather than UTC.
- Computed figures (net worth, spending reports, ledger summary) are produced by server-side aggregations so they cannot drift between screens.

Main user flows

1. Sign in / register (email + code), optional app lock (biometrics).
2. Create or edit bills, set recurrence and reminders; pay or skip an occurrence.
3. Import a statement: parse -> review detected items -> apply to the ledger.
4. Dashboard shows to-pay vs to-receive, overdue alerts, Payday Runway and AI insights.
5. Ledger: assign rows to accounts, filter, bulk delete, view summary.
6. Budgets, Goals, Debt payoff planner.
7. Reports: spending by window, by account, period comparison; Charts tab.
8. Calendar view of due dates; export to the device calendar (reversible).
9. Settings: preferences, exact-alarm access, backup/restore, CSV export.

5. Data model (MongoDB collections)

- users, user_sessions - accounts and sessions
- bills - payable/receivable items with recurrence, paid status, trials
- transactions - the ledger (statement/imported rows)
- categories - user categories
- budgets, goals, debts, accounts, splits - planning and money features
- attachments - document vault (receipts / warranties / docs), stored server-side
- households - shared-bill membership
- events, feedback - product analytics and in-app feedback
- app_config - backend-served app-version gate and settings

6. API surface (selected endpoints)

- GET /api/healthz, GET /api/ (health and discovery)
- Auth: /api/auth/* (register, login, code, forgot-password, sessions)
- Bills: GET/POST /api/bills, GET/PUT/DELETE /api/bills/{id},
POST /api/bills/{id}/skip
- GET /api/dashboard/summary, GET /api/pro/status
- Import: POST /api/import/parse, POST /api/import/apply
- Ledger: GET /api/ledger/summary, GET /api/ledger/by-account,
POST /api/ledger/assign-account, DELETE /api/ledger
- Reports: GET /api/reports/spending, GET /api/reports/period-compare

- GET /api/net-worth
- Household: /api/household (create, join, leave, bills)
- Attachments: POST/GET /api/attachments, GET/DELETE /api/attachments/{id}
- POST /api/digest/send, POST /api/ocr/receipt
- GET/PUT /api/settings, GET/PUT /api/app/version
- Backup: GET /api/backup/export, POST /api/backup/import
- Admin: /api/admin/* (users, overview, payments, ban/unban, pro restore/revoke, refunds, audit, feedback, re-engagement)

7. Integrations

- Google Play IAP: purchases verified server-side via the Android Publisher API using a service-account JSON you provide.
- AdMob: banner (collapsible) and rewarded units; free tier only.
- AI insights: provider chain with automatic fallback on outage, rate limit or credit exhaustion. Health: GET /api/insights/health.
- Email: Gmail SMTP for transactional email.

8. Deployment (summary)

- Backend: copy backend/.env.template to backend/.env, fill in your MongoDB URL, JWT secret, admin credentials, AI provider keys, Gmail credentials and the path to your Google Play service-account JSON. Deploy backend/ to any container host (Cloud Run, Render, Fly, or the included Dockerfile).
- Frontend: npm install, set your production API URL in src/lib/backend-url.ts, set your AdMob IDs, then build with EAS.
- Full details are in BUYER-SETUP.md.

9. Rebrand checklist

- app.json: name, slug, scheme, android.package, ios.bundleIdentifier, version, owner, extra.eas.projectId, updates.url.
- Replace icons and splash in assets/images/.
- Search-and-replace the product name and package id across app/ and src/.
- Decide monetisation: keep the one-time Pro product or change the product id; keep or change the AdMob unit IDs.
- Provide your own google-services.json (Firebase), Play service-account JSON, signing keystore and MongoDB connection string.

10. Support

This package is sold with 30 days of support from the date of purchase (deployment and rebrand questions). The original seller's accounts, keys and signing material are deliberately excluded - you run everything on your own infrastructure.