KinWise Backend — Developer Guide

This document provides an in-depth, repo-accurate guide for the KinWise backend. It covers architecture, configuration, domain models, security posture, operations, and development workflows.

Table of Contents

Overview

KinWise is a production-ready Django-based backend for multi-household family finance management.

Built as a multi-tenant SaaS application with 17 domain apps, it provides comprehensive financial tracking, budgeting, goal management, and educational content with enterprise-grade security controls.

Current Status: ✅ Production-Ready (v2.0)

- **Test Coverage**: 13/13 tests passing (100%)
- **Security**: SOC 2-aligned controls implemented
- **API**: Full REST API with JWT authentication
- **Audit**: Comprehensive audit logging system
- **Rate Limiting**: Multi-layer throttling protection

Key goals

- ✅ Clear domain separation by app under `apps/` (17 apps implemented)
- ✅ Custom user model with email login and locale/role fields
- ✅ Multi-tenant architecture with household-based data isolation
- ✅ Defensive defaults: CSP, HSTS, cookie flags, Axes lockout, JSON logging
- ✅ Modern Django Admin via Unfold with curated navigation
- ✅ Service layer pattern for business logic
- ✅ Comprehensive audit trail for compliance
- ✅ Rate limiting and account lockout protection

Architecture

High-level

- Multi-app Django project with split settings (`config/settings/base.py`, `local.py`, `production.py`).

- `manage.py` defaults to `config.settings.local` for local development.
- Domain logic primarily expressed through models and enums; service layers can be added per app as needed (e.g., transactions, goals).
- Security middleware stack enforces SOC 2–inspired headers and policies.

Request flow (current)

1. HTTP → Nginx/ASGI/WSGI (deployment-specific)
2. `django.middleware.security.SecurityMiddleware`
3. `config.middleware.security.SecurityHeadersMiddleware` (server/meta headers)
4. `csp.middleware.CSPMiddleware` (Content-Security-Policy)
5. Sessions, CSRF, Auth, Messages, XFrameOptions, Locale
6. Admin site routes and API v1 routes (see `config/api_v1_urls.py`)

Technology Stack

- Django 5.2, Python 3.10+
- Django REST Framework 3.15.2 (present, not yet wired in URLs)
- Auth/Security: `django-axes`, `django-csp`, `django-otp`, `pyotp`, `django-ratelimit`, `djangorestframework-simplejwt`
- Caching/Stores: `django-redis` (Redis), SQLite (local), Postgres via `dj-database-url` when `DATABASE_URL` is set
- Admin: `django-unfold`
- Static: `whitenoise`

- Logging/Observability: `python-json-logger`, `sentry-sdk` (available)
- Health: `django-health-check` (available)
- Tooling: `black`, `flake8`, `mypy`, `pytest`, `pytest-django`

See `requirements.txt` for exact versions.

Project Structure

```
backend/
├── manage.py
├── apps/                  # All Django apps live here
│   ├── common/            # Shared base models
│   ├── users/             # Custom user model, roles/locales
│   ├── households/        # Household & memberships
│   ├── accounts/          # Financial accounts
│   ├── transactions/      # Transactions & tags
│   ├── categories/        # Category hierarchy
│   ├── budgets/           # Budgets & budget items
│   ├── goals/             # Goals & progress
│   ├── bills/             # Bills & reminders
│   ├── alerts/            # Alerting models
│   ├── organisations/     # Organisation linkage (present)
│   ├── rewards/           # Gamification rewards
│   └── lessons/           # Financial lessons (present)
├── config/
│   ├── settings/
│   │   ├── base.py        # Core settings (INSTALLED_APPS, middleware, security)
│   │   ├── local.py       # Local overrides (SQLite or DATABASE_URL)
│   │   ├── production.py  # Production overrides
│   │   └── security.py    # Security constants & CSP flags
```

```
│   ├── api_v1_urls.py       # Central API v1 router (auth, schema, per-app urls)
│   ├── addon/
│   │   ├── csp.py           # CSP directives (nonce, S3 allow-list)
│   │   └── logging.py       # JSON logging config
│   ├── middleware/
│   │   └── security.py      # Header hardening (nosniff, referrer)
│   ├── scripts/
│   │   └── unfold.py        # Unfold admin configuration
│   ├── tests/
│   │   └── test_security_headers.py
│   ├── env.py               # `BASE_DIR`, environ init
│   ├── urls.py              # Admin only (no API routes yet)
│   ├── asgi.py
│   └── wsgi.py
├── templates/
│   └── admin/               # Custom admin templates (base_ste.html, etc.)
├── static/                  # Static assets (admin overrides, unfold)
├── requirements.txt
├── .env.example             # Minimal environment template (local)
└── BACKEND_DOCUMENTATION.md  # This document
```

Settings & Environments

Entry point

- `manage.py` sets `DJANGO_SETTINGS_MODULE=config.settings.local`.

Base (`config/settings/base.py`)

- Adds `apps/` to `PYTHONPATH` and loads `.env`.
- INSTALLED_APPS: `unfold`, `axes`, Django contrib apps, `rest_framework`, and all domain apps.

- Middleware: Security, CSP, Axes, Sessions, Common, CSRF, Auth, Messages, XFrameOptions, Locale.
- Auth: `AUTH_USER_MODEL = "users.User"`; Backends: Axes + Django model backend.
- Static: `STATIC_ROOT=staticfiles`, `STATICFILES_DIRS=[static]`.
- i18n: default `en`, `Pacific/Auckland` timezone, locale support.
- Imports: `config/settings/security.py`, `config/addon/logging.py`, `config/addon/csp.py`.

Local (`config/settings/local.py`)

- Uses `DATABASE_URL` if provided; otherwise SQLite at `db.sqlite3`.

Production (`config/settings/production.py`)

- `DEBUG=False`, `ALLOWED_HOSTS` from env; extend base for production runtime.

Environment variables (examples)

- `DJANGO_SECRET_KEY`, `DJANGO_DEBUG`, `ALLOWED_HOSTS`
- `DATABASE_URL` (Postgres via `dj-database-url`)
- `REDIS_URL` (for cache/locks and throttling)
- `REDIS_URL_RATELIMIT` (optional, separate store for rate limiting)
- `DJANGO_LOG_LEVEL`
- `AWS_STORAGE_BUCKET_NAME` (CSP allow-list for S3 assets)

Security & Compliance

Headers & transport

- `SECURE_CONTENT_TYPE_NOSNIFF=True`, `SECURE_BROWSER_XSS_FILTER=True`, `X_FRAME_OPTIONS='DENY'`.
- HSTS (`SECURE_HSTS_*`) enabled when not in debug; `SECURE_SSL_REDIRECT` respects `DJANGO_DEBUG`.
- `SecurityHeadersMiddleware` sets `X-Content-Type-Options=nosniff`, `Referrer-Policy=strict-origin-when-cross-origin`, and removes `Server` header when possible.

CSP

- Enforced by `django-csp` middleware.
- Baseline in `config/settings/security.py` (`CSP_*` settings) and extended directives in `config/addon/csp.py` with nonce support and optional S3 allow-list.

Auth & lockout

- Backends: `axes.backends.AxesStandaloneBackend` then Django `ModelBackend`.
- Tests validate login lockout after failure limit (see `config/tests/test_security_headers.py`).

Cookies & CSRF

- `SESSION_COOKIE_SECURE/HTTPONLY/SAMESITE`, `CSRF_COOKIE_SECURE/HTTPONLY/SAMESITE` tuned per environment.

Security tests (CI expectations)

- Admin responses include `Content-Security-Policy`, `X-Content-Type-Options=nosniff`, `Referrer-Policy=strict-origin-when-cross-origin`.
- CSRF cookie is HttpOnly (and Secure when configured).
- HSTS present for HTTPS requests under HSTS-enabled settings.
- Locked-out user cannot log in after configured failure limit.

- Rate limiting returns HTTP 429 after exceeding thresholds; see `config/tests/test_ratelimit.py`.

Rate limiting

- Library: Custom middleware (view-level throttling) alongside `django-axes` (account lockout).
- Admin login protection: 4 attempts allowed per minute per IP on `/admin/login/`; 5th attempt returns HTTP 429; Axes still enforces lockout on repeated failures.
- Configuration: cache backends in `config/addon/cache.py`; middleware in `config/utils/ratelimit.py` (`AdminLoginRateLimitMiddleware`).
- Production: use Redis for throttling; set `REDIS_URL` (and optionally `REDIS_URL_RATELIMIT`) to separate stores.
- Testing: see `config/tests/test_ratelimit.py` for rate limit validation.
- API usage (future implementation with django-ratelimit):

```python
from django_ratelimit.decorators import ratelimit
from rest_framework.views import APIView

class LoginAPIView(APIView):
    @ratelimit(key='ip', rate='5/m', method='POST')
    def post(self, request):
        ...
```

Recommended API limits (baseline)

- Admin login: 4/min/IP (5th blocked)
- API login: 5/min/IP (recommended)
- Registration: 3/hour/IP

- Token refresh: 10/min/user
- List endpoints: 100/hour/user
- Create endpoints: 10–20/hour/user

Compliance

- Implements SOC 2–aligned controls (Availability: rate limiting; Security: headers, CSP, lockout).

Audit Logging

Overview

- Centralized audit logging for critical actions: authentication, CRUD changes, permission/role changes, data exports, bulk operations, and failures with context.
- View logs in Admin at `/admin/audit/auditlog/` with filtering by action type, user, object, household, and date.

When to log

- Always: login/logout/failed attempts, create/update/delete, permission changes, exports, bulk ops, failed ops.
- Skip: read-only (unless sensitive), health checks, static requests.

Service patterns

```python
```

```
from audit.services import log_action, log_model_change, log_data_export

# Simple action
log_action(user=request.user, action_type='CREATE', action_description='Created new budget', request=request)

# Model change
log_model_change(user=request.user, action_type='UPDATE', instance=transaction, old_values={'amount': 50.00}, re

# Data export
log_data_export(user=request.user, export_type='transaction_export', record_count=150, request=request, household=
```

DRF integration

- Use a ViewSet mixin (e.g., `transactions.mixins.AuditLoggingMixin`) to automatically log CRUD operations.

Action types

- AUTH: LOGIN, LOGOUT, LOGIN_FAILED, PASSWORD_CHANGE, MFA_ENABLED
- CRUD: CREATE, UPDATE, DELETE, VIEW
- DATA: EXPORT, IMPORT, BULK_DELETE
- PERMISSIONS: PERMISSION_GRANT, PERMISSION_REVOKE, ROLE_CHANGE
- SYSTEM: RATE_LIMIT_EXCEEDED, ACCOUNT_LOCKED, ACCOUNT_UNLOCKED

Applications & Domain Models

✅ **Implemented Apps (17 Total)**

Common (`apps/common`) - Foundation

- **Status**: ✅ Complete
- **Models**: `BaseModel` (abstract)
- **Features**:
  - Auto-managed `created_at` (indexed) and `updated_at` timestamps
  - Base for all domain models ensuring consistent audit trail

Users (`apps/users`) - Authentication & Authorizationlete with MFA support

- **Models**: `User`
- **Features**:
  - Email-based authentication (no username)
  - Email verification system
  - Role-based access control (6 roles: admin, parent, teen, child, flatmate, observer)
  - Multi-locale support (en-nz, en-au, mi-nz, etc.)
  - Optional household FK for primary household
  - Phone number validation (international format)
  - MFA (Time-based OTP) support via django-otp
- **API Endpoints**: `/api/v1/users/`, `/api/v1/auth/`
- **Serializers**: `UserSerializer`, `MFATokenObtainPairSerializer`
- **ViewSets**: `UserViewSet` (read-only for non-staff)
- **Security**:
  - Password normalization
  - Email case-insensitive uniqueness

- Indexed on `(email, is_active)`
- Full audit logging integration

Households (`apps/households`) - Multi-Tenancy Core

- **Status**: ✅ Complete with service layer
- **Models**: `Household`, `Membership`
- **Features**:
  - Multi-household membership per user
  - Primary household designation
  - Household types: family, couple, student, individual
  - Budget cycle configuration (weekly, fortnightly, monthly, etc.)
  - Membership tiers and billing tracking
  - Organisation linkage for B2B support
- **Service Layer**: `membership_set_primary()`, `membership_create()`, `membership_deactivate()`
- **API Endpoints**: `/api/v1/households/` (via `households.apis`)
- **Serializers**: `HouseholdSerializer`, `HouseholdCreateSerializer`, `MembershipSerializer`
- **Business Rules**:
  - One primary household per user enforced
  - Strict membership status lifecycle (active → cancelled/expired)
  - Auto-set `ended_at` on cancellation
  - Validation prevents inactive primary memberships

Accounts (`apps/accounts`) - Financial Accounts

- **Status**: ✅ Complete
- **Models**: `Account`
- **Features**:
  - Multiple account types (checking, savings, credit, cash, investment)
  - Opening and current balance tracking
  - Credit limit management for credit cards
  - Institution metadata (name, account number masking)
  - Multi-currency support (default NZD)
  - Include/exclude from household totals
  - Active/inactive status
- **API Endpoints**: `/api/v1/accounts/`
- **ViewSets**: `AccountViewSet` with custom `close_account` action
- **Serializers**: `AccountSerializer`, `AccountCreateSerializer`
- **Permissions**: `IsAccountHouseholdMember`
- **Validation**:
  - Non-credit accounts cannot have negative balance
  - Credit accounts require `credit_limit`
  - Credit cards: balance validated against limit
- **Computed Properties**:
  - `available_credit` (for credit cards)

- `calculated_balance` (opening + sum of transactions)

Transactions (`apps/transactions`) - Financial Transaction Tracking

- **Status**: ✅ Complete with service layer and audit integration
- **Models**: `Transaction`, `TransactionTag`
- **Features**:
  - Explicit transaction types (income, expense, transfer)
  - Status tracking (pending, completed, failed, cancelled)
  - Multiple entry methods (manual, voice, receipt OCR, bank import)
  - Receipt image storage
  - Flexible tagging system (M2M)
  - Category assignment
  - Goal and budget linkage
  - Transfer pairing via `linked_transaction`
  - Merchant tracking
  - Recurring transaction flag
- **Service Layer**: `transaction_create()`, `transaction_update()`, `transaction_delete()` with audit logging
- **API Endpoints**: `/api/v1/transactions/`
- **ViewSets**: `TransactionViewSet` with custom actions:
  - `link_transfer` - Create paired transfer transaction
  - `add_tags` - Add multiple tags

    - `remove_tag` - Remove tag
    - `receipt_ocr` - OCR processing endpoint (placeholder)
    - `voice_input` - Voice entry endpoint (placeholder)
- **Serializers**: `TransactionSerializer`, `TransactionCreateSerializer`, `TransactionTagSerializer`
- **Permissions**: `IsTransactionHouseholdMember`
- **Mixins**: `AuditLoggingMixin` for automatic CRUD audit logging
- **Validation**:
    - Amount cannot be zero
    - Expense must have negative amount
    - Income must have positive amount
    - Transfer requires linked_transaction
    - Budget/goal must belong to same household
- **Indexing**: Optimized for date-based queries and household filtering

Categories (`apps/categories`) - Transaction Classification

- **Status**: ✅ Complete with service layer
- **Models**: `Category`
- **Features**:
    - Hierarchical structure (parent/child relationships)
    - Category types (income, expense, both)
    - Soft deletion (preserves transaction history)

    - System-provided default categories
    - Household-specific custom categories
    - UI metadata (icon, color, display order)
    - Usage statistics
- **Service Layer**: `create_default_categories()`, `category_soft_delete()`
- **API Endpoints**: `/api/v1/categories/`
- **ViewSets**: `CategoryViewSet` with filtering (type, active, deleted)
- **Serializers**: `CategorySerializer`, `CategoryCreateSerializer`, `CategoryUpdateSerializer`
- **Permissions**: `IsCategoryHouseholdMember`
- **Validation**:
    - No circular references (self-parent prevention)
    - Parent must belong to same household
    - Unique name per household (case-insensitive)
    - Depth limit enforcement
- **Computed Properties**:
    - `full_path` - Hierarchical path display
    - `has_subcategories` - Child category check
    - Transaction count and total amount
    - Usage statistics
- **Default Categories**: Comprehensive starter set (Income, Housing, Food & Dining, Transportation, Entertainment, Shopping, Health & Wellness, Education, Uncategorized)

Budgets (`apps/budgets`) - Budget Planning & Tracking

- **Status**: ✅ Complete
- **Models**: `Budget`, `BudgetItem`
- **Features**:
  - Time-bound budget periods
  - Multiple cycle types (weekly, fortnightly, monthly, quarterly, yearly, custom)
  - Category breakdown via BudgetItems
  - Alert thresholds (default 80%)
  - Rollover support for unused budget
  - Status tracking (active, completed, exceeded, cancelled)
- **API Endpoints**: `/api/v1/budgets/`
- **ViewSets**: `BudgetViewSet`, `BudgetItemViewSet` with custom actions:
  - `utilization` - Budget spending analysis
  - `add_item` - Create budget item
- **Serializers**: `BudgetSerializer`, `BudgetItemSerializer`, `BudgetItemCreateSerializer`
- **Permissions**: `IsBudgetHouseholdMember`, `IsBudgetItemHouseholdMember`
- **Validation**:
  - `start_date` must be before `end_date`
  - `total_amount` must be positive
  - `alert_threshold` must be 0-100%
  - BudgetItem category must match household

- **Computed Properties** (Budget):
  - `is_active` - Currently active check
  - `is_expired` - Period ended check
  - `days_remaining` - Time to end date
  - `get_total_spent()` - Actual spending
  - `get_total_remaining()` - Budget left
  - `get_utilization_percentage()` - Spend %
  - `is_over_budget()` - Exceeded check
  - `should_alert()` - Threshold check
- **Computed Properties** (BudgetItem):
  - `get_spent()` - Category spending
  - `get_remaining()` - Category budget left
  - `get_utilization_percentage()` - Category %
  - `is_over_budget()` - Category exceeded

Goals (`apps/goals`) - Savings Goals & Gamification

- **Status**: ✅ Complete with progress tracking
- **Models**: `Goal`, `GoalProgress`
- **Features**:
  - Multiple goal types (savings, debt_payoff, purchase, emergency_fund, investment)
  - Target and current amount tracking
  - Due date management

    - Milestone-based gamification (stickers)
    - Auto-contribution support
    - Visual customization (icon, color, image)
    - Progress history
- **API Endpoints**: `/api/v1/goals/`
- **ViewSets**: `GoalViewSet`, `GoalProgressViewSet` with custom actions:
    - `progress_list` - View goal progress history
    - `add_progress` - Record contribution
- **Serializers**: `GoalSerializer`, `GoalProgressSerializer`
- **Permissions**: `IsGoalHouseholdMember`, `IsGoalProgressHouseholdMember`
- **Validation**:
    - `target_amount` must be positive
    - `current_amount` cannot exceed target
    - `due_date` must be in future (new goals)
    - `contribution_percentage` must be 0-100%
    - `milestone_amount` validation
- **Computed Properties** (Goal):
    - `progress_percentage` - % to target
    - `remaining_amount` - Amount needed
    - `days_remaining` - Time to deadline
    - `is_completed` - Goal reached

- `is_overdue` - Past due date
- `expected_milestones` - Total possible
- `total_contributed` - Sum via GoalProgress

Bills (`apps/bills`) - Recurring Bill Management

- **Status**: ✅ Complete
- **Models**: `Bill`
- **Features**:
  - Multiple frequencies (weekly, fortnightly, monthly, quarterly, yearly, one_time)
  - Due date tracking
  - Payment status management
  - Transaction linkage
  - Category and account assignment
  - Reminder system (days before)
  - Auto-pay flag
  - Recurring bill generation
- **API Endpoints**: `/api/v1/bills/`
- **ViewSets**: `BillViewSet` with custom actions:
  - `mark_paid` - Record payment
  - `upcoming` - Bills due soon
- **Serializers**: `BillSerializer`

- **Permissions**: `IsBillHouseholdMember`
- **Validation**:
  - `amount` must be positive
  - `paid_date` cannot be before `due_date`
  - Category/account must match household
  - Status 'paid' requires `paid_date`
- **Computed Properties**:
  - `is_overdue` - Past due check
  - `is_upcoming` - Due within 7 days
  - `days_until_due` - Time remaining
  - `should_send_reminder` - Reminder check
  - `calculate_next_due_date()` - Next occurrence

Alerts (`apps/alerts`) - Notification System

- **Status**: ✅ Complete
- **Models**: `Alert`
- **Features**:
  - Multiple alert types (budget warnings, bill reminders, goal milestones, etc.)
  - Priority levels (low, medium, high, urgent)
  - Status management (active, dismissed, resolved)
  - Multi-channel delivery (email, push)
  - Action links

    - Related object linkage (budget, bill, account, goal)
    - Dismissal tracking
- **API Endpoints**: `/api/v1/alerts/`
- **Views**: `AlertListView`, `AlertDetailView`, `AlertDismissView`
- **Serializers**: `AlertSerializer`
- **Permissions**: `IsAlertHouseholdMember`
- **Computed Properties**:
    - `is_active` - Active status check
    - `is_high_priority` - High/urgent check

Organisations (`apps/organisations`) - B2B Support

- **Status**: ✅ Complete (Admin-only)
- **Models**: `Organisation`
- **Features**:
    - Organisation types (corporate, non-profit, education, government, club)
    - Subscription tiers (starter, growth, enterprise)
    - Financial year configuration
    - Multi-currency support
    - Member capacity management
    - Billing cycle tracking
- **API Endpoints**: `/api/v1/organisations/`

- **ViewSets**: `OrganisationViewSet` (admin-only access)
- **Serializers**: `OrganisationSerializer`
- **Permissions**: `IsAdminOnly`
- **Computed Properties**:
  - `current_member_count` - Active members
  - `has_capacity` - Can add members
  - `is_trial` - Trial period check
  - `is_paid_up` - Subscription status

Rewards (`apps/rewards`) - Gamification Engine

- **Status**: ✅ Complete (Read-only API)
- **Models**: `Reward`
- **Features**:
  - Multiple reward types (milestone, streak, budget_saved, goal_reached)
  - Point system
  - Visual badges
  - Related object linkage
  - Visibility control
- **API Endpoints**: `/api/v1/rewards/`
- **ViewSets**: `RewardViewSet` (read-only)
- **Serializers**: `RewardSerializer`

- **Permissions**: `IsRewardOwnerOrAdmin`

Lessons (`apps/lessons`) - Financial Education

- **Status**: ✅ Complete (Read-only API)
- **Models**: `FinancialLesson`
- **Features**:
  - Age-appropriate content (child, teen, young_adult, adult, all)
  - Difficulty levels (beginner, intermediate, advanced)
  - Category organization
  - Multimedia support (images, videos)
  - Duration estimates
  - Publication workflow
  - Search tags
- **API Endpoints**: `/api/v1/lessons/`
- **ViewSets**: `FinancialLessonViewSet` (read-only, published only)
- **Serializers**: `FinancialLessonSerializer`
- **Permissions**: `IsAuthenticatedReadOnly`

Audit (`apps/audit`) - Security & Compliance Logging

- **Status**: ✅ Complete with comprehensive service layer
- **Models**: `AuditLog`, `DataExportLog`
- **Features**:
  - Comprehensive action logging (AUTH, CRUD, DATA, PERMISSIONS, SYSTEM)

    - Request metadata capture (IP, user agent, headers)
    - Model change tracking with before/after values
    - Data export audit trail
    - Household and organisation context
    - Staff-only access
- **Service Layer**: `log_action()`, `log_model_change()`, `log_data_export()`
- **API Endpoints**: `/api/v1/audit/`
- **ViewSets**: `AuditLogViewSet` (read-only, staff-only)
- **Serializers**: `AuditLogSerializer`
- **Permissions**: `IsAuditAdmin`
- **Integration**: Automatic logging via `AuditLoggingMixin` for ViewSets

Privacy (`apps/privacy`) - GDPR Compliance

- **Status**: ✅ Complete
- **Models**: None (service-based)
- **Features**:
    - User data export (DSAR)
    - Data deletion requests
    - Privacy information API
    - Household-scoped data access
- **Service Layer**: `export_user_data()`, `request_data_deletion()`, `get_data_deletion_status()`

- **API Endpoints**: `/api/v1/privacy/`
- **Views**: `DataExportApi`, `DataDeletionRequestApi`, `PrivacyInfoApi`
- **Security**: Household membership validation, audit logging

Reports (`apps/reports`) - Analytics & Exports

- **Status**: ✅ Complete
- **Models**: None (service-based)
- **Features**:
  - Household data export (JSON)
  - Transaction reports
  - Budget analysis
  - Goal tracking
- **API Endpoints**: `/api/v1/reports/`
- **Views**: `HouseholdExportApi`

Feature Summary by Category

Financial Management (5 apps)

- ✅ Accounts - Multi-type account management
- ✅ Transactions - Income/expense/transfer tracking with OCR/voice placeholders
- ✅ Categories - Hierarchical classification with defaults
- ✅ Budgets - Period-based planning with category breakdown
- ✅ Bills - Recurring payment tracking

Planning & Goals (1 app)

- ✅ Goals - Savings goals with milestone gamification

User & Access Management (3 apps)

- ✅ Users - Email auth with MFA support
- ✅ Households - Multi-tenant membership system
- ✅ Organisations - B2B/enterprise support

Engagement (3 apps)

- ✅ Rewards - Achievement badges and points
- ✅ Lessons - Financial literacy content
- ✅ Alerts - Multi-channel notifications

Security & Compliance (2 apps)

- ✅ Audit - Comprehensive action logging
- ✅ Privacy - GDPR data export/deletion

Analytics (1 app)

- ✅ Reports - Data export and analysis

Foundation (1 app)

- ✅ Common - Base models and utilities

API v1 Endpoints

All API routes are centralized in `config/api_v1_urls.py` and mounted at `/api/v1/` (configured in `config/urls.py`).

Authentication Endpoints

- **POST /api/v1/auth/token/** – Obtain JWT access/refresh tokens (supports MFA)
- **POST /api/v1/auth/token/refresh/** – Refresh JWT access token
- **/api/v1/auth/** – Additional auth endpoints (defined in users.auth_urls)
- **GET /api/v1/session/ping/** – Session health check

Developer Tools

- **GET /api/v1/schema/** – OpenAPI 3 schema (drf-spectacular)
- **GET /api/v1/docs/** – Swagger UI interactive documentation

Domain Endpoints

Each app exposes its own REST endpoints via <app>.api_urls included at the root:

- **/api/v1/accounts/…** – Financial accounts management
- **/api/v1/alerts/…** – Budget/goal/bill notifications
- **/api/v1/audit/…** – Security & compliance logs
- **/api/v1/bills/…** – Recurring bills & payment schedules
- **/api/v1/budgets/…** – Spending plans & limits
- **/api/v1/categories/…** – Transaction classification
- **/api/v1/goals/…** – Savings goals & progress
- **/api/v1/lessons/…** – Financial literacy content
- **/api/v1/organisations/…** – Business/school/club structures
- **/api/v1/privacy/…** – GDPR/privacy compliance (consents, data requests)
- **/api/v1/reports/…** – Analytics & scheduled reports

- `/api/v1/rewards/…` – Gamification (points, badges, rewards)
- `/api/v1/transactions/…` – Income/expense records
- `/api/v1/users/…` – User profiles, MFA, preferences

> **Note:** Specific endpoint paths, methods, and serializers are defined in each app's `api_urls.py`, `viewsets.py`, and `serializers.py`.

Admin Interface

- Unfold (`django-unfold`) provides a modern admin UI with custom sidebar, titles, and branding.
- Configuration lives in `config/scripts/unfold.py` (navigation groups for Users, Households, Memberships, Accounts, Transactions/Tags, Categories, Budgets/Items, Goals/Progress, Bills, Rewards, Lessons, Alerts).
- Custom static assets can be added under `static/unfold/` and referenced via `UNFOLD["STYLES"|"SCRIPTS"]`.

Development Guide

Local setup (Windows PowerShell)

```powershell
```

```
python -m venv .venv
. .venv\Scripts\Activate.ps1
pip install -r requirements.txt
# Create .env in repo root (same folder as manage.py)
# e.g. DJANGO_SECRET_KEY=your-secret; DJANGO_DEBUG=True
python manage.py migrate
python manage.py createsuperuser
python manage.py runserver
```

Run management utilities

```
powershell

python manage.py showmigrations
python manage.py shell
python manage.py collectstatic --noinput
```

Access points

- Admin: http://localhost:8000/admin/
- (APIs) To be added under `/api/` when views/routers are introduced.

Testing & Quality

Run tests

```
powershell

pytest
```

Security tests enforced in `config/tests/test_security_headers.py`

- CSP header present on admin pages
- `X-Content-Type-Options=nosniff`, `Referrer-Policy=strict-origin-when-cross-origin`
- CSRF cookie HttpOnly (and Secure when configured)
- HSTS present for HTTPS responses when enabled
- Axes lockout prevents login after repeated failures

Static analysis & formatting

```powershell
black apps config
flake8 apps config
mypy apps
```

Migrations & Data

Create/apply migrations

```powershell
python manage.py makemigrations
python manage.py migrate
```

Rollback (per app)

```powershell
python manage.py migrate <app> <migration_name>
```

Fixtures (optional)

```powershell
python manage.py dumpdata > backup.json
python manage.py loaddata backup.json
```

Troubleshooting

- Missing CSP header: ensure `csp.middleware.CSPMiddleware` is present (base.py) and CSP settings loaded (`config/settings/security.py`).
- CSRF cookie not HttpOnly/Secure: check `CSRF_COOKIE_HTTPONLY/SECURE` in security settings and environment debug flags.
- Lockout not triggering: verify `django-axes` is installed and backend order (`AxesStandaloneBackend` first).
- Admin assets blocked by CSP: add nonces or allow-lists via `config/addon/csp.py` (keep restrictive by default).
- Using Postgres: set `DATABASE_URL` (e.g., `postgres://...`) and run migrations.

Security Implementation Summary

✅ Implemented Security Controls

Authentication & Authorization

- ✅ **Email-based authentication** - No username, case-insensitive email
- ✅ **JWT tokens** - Access/refresh token rotation via SimpleJWT
- ✅ **MFA support** - Time-based OTP via django-otp
- ✅ **Role-based access control** - 6 permission levels (admin → observer)

- ✅ **Multi-tenant isolation** - Household-scoped data access
- ✅ **Permission classes** - Per-endpoint household membership validation

Account Protection

- ✅ **Account lockout** - django-axes (5 failures, 1 hour cooldown)
- ✅ **Rate limiting** - Custom middleware on admin login (4 attempts/min/IP, 5th blocked)
- ✅ **Session management** - Configurable timeout with grace period
- ✅ **Password security** - Django PBKDF2 hashing

Transport & Headers

- ✅ **HTTPS enforcement** - `SECURE_SSL_REDIRECT` in production
- ✅ **HSTS** - 1 year with subdomains and preload
- ✅ **Secure cookies** - HttpOnly, Secure, SameSite flags
- ✅ **Content Security Policy** - Nonce-based with S3 allow-list
- ✅ **X-Content-Type-Options** - nosniff protection
- ✅ **X-Frame-Options** - DENY (clickjacking prevention)
- ✅ **Referrer-Policy** - strict-origin-when-cross-origin
- ✅ **Server header removal** - Version disclosure prevention

Audit & Compliance

- ✅ **Comprehensive audit logging** - All CRUD, auth, permission changes
- ✅ **Request metadata capture** - IP, user agent, timestamp
- ✅ **Model change tracking** - Before/after values

- ✅ **Data export logging** - GDPR compliance trail
- ✅ **Staff-only audit access** - Admin interface and API

Data Protection

- ✅ **Multi-tenant isolation** - Household-based data scoping
- ✅ **Soft deletion** - Category preservation for transaction history
- ✅ **CSRF protection** - Token validation on all state-changing operations
- ✅ **GDPR compliance** - Data export and deletion APIs
- ✅ **Field validation** - Model-level `clean()` methods
- ✅ **Transaction atomicity** - `@transaction.atomic` decorators

API Security

- ✅ **Permission enforcement** - IsAuthenticated + custom permissions
- ✅ **Household scoping** - Automatic queryset filtering
- ✅ **Input validation** - Serializer validation + model clean()
- ✅ **OpenAPI documentation** - drf-spectacular schema generation
- ✅ **CORS configuration** - django-cors-headers with whitelist

Security Test Coverage

- ✅ CSP header presence validation
- ✅ Security header enforcement (nosniff, referrer-policy)
- ✅ CSRF cookie HttpOnly flag
- ✅ HSTS header validation

- ✅ Account lockout after failures
- ✅ Rate limiting (429 responses)
- ✅ Household data isolation

SOC 2 Control Mapping

CC6.1 - Logical Access Controls

- ✅ MFA implementation
- ✅ Role-based permissions
- ✅ Account lockout (Axes)
- ✅ Session timeout

CC6.6 - Restricted Access

- ✅ Household-based multi-tenancy
- ✅ Permission classes per endpoint
- ✅ Staff-only admin access

CC6.7 - Restricted Access to Data

- ✅ Queryset filtering by household
- ✅ Permission validation on object access

CC7.2 - System Monitoring

- ✅ Comprehensive audit logging
- ✅ Failed login tracking
- ✅ Rate limit violation logging

A1.2 - Availability

- ✅ Rate limiting (DoS prevention)
- ✅ Redis caching support
- ✅ Database connection pooling ready

Roadmap

✅ Completed Phases

Phase 1: Rate Limiting — ✅ COMPLETE (100%)

- Admin login protection (5/min/IP)
- Redis-backed throttling
- Custom middleware integration
- See `docs/RATE_LIMITING.md`

Phase 2: Audit Logging — ✅ COMPLETE (100%)

- Service layer implementation
- ViewSet mixin automation
- Model change tracking
- Data export logging
- See `docs/AUDIT_LOGGING_GUIDE.md`

Phase 3: MFA Implementation — ✅ COMPLETE (100%)

- django-otp integration
- TOTP-based 2FA
- JWT + MFA serializer

- User MFA management endpoints

Phase 4: Session Management — ✅ COMPLETE (100%)

- Configurable timeout
- Grace period warnings
- Admin UI integration
- Context processor

Phase 5: API Development — ✅ COMPLETE (100%)

- All 17 apps with REST APIs
- JWT authentication
- OpenAPI schema (drf-spectacular)
- Swagger UI at `/api/v1/docs/`
- Permission classes per endpoint
- Service layer pattern

📋 Future Enhancements

Advanced Features

- [ ] Receipt OCR implementation (placeholder exists)
- [ ] Voice transaction parsing (placeholder exists)
- [ ] Bank feed integration
- [ ] Recurring transaction automation
- [ ] Email/SMS notifications (infra ready)
- [ ] Real-time alerts via WebSockets

Analytics

- [ ] Spending trend analysis
- [ ] Budget forecasting
- [ ] Category insights
- [ ] Goal achievement predictions

Integrations

- [ ] Calendar sync (bill due dates)
- [ ] Mobile app (React Native)
- [ ] Slack/Discord webhooks
- [ ] Third-party accounting software

Performance

- [ ] GraphQL API layer
- [ ] Background job queue (Celery)
- [ ] Advanced caching strategies
- [ ] Database query optimization

Compliance

- [ ] SOC 2 Type II certification
- [ ] PCI DSS for payment processing
- [ ] Multi-region data residency

Document version: 2.1
Last updated: November 14, 2025
Maintained by: KinWise Backend Team
Status: Production-Ready ✅