SplitLedger

Product Requirements & Technical Specification

Version: 2.1.0 | Author: Soumik Ghosh | 2026-02-21

Module	Status
Core (Groups, Users, Expenses, Balances, Settlements)	v1 — Included
Authentication (JWT register / login / refresh)	v1 — Included
Expense Editing (PATCH with INV-1 re-validation)	v1 — Included
Expense Deletion (soft-delete, balance-safe)	v1 — Included
Equal Split Mode (server-side computation)	v1 — Included
Expense Categories (enum, balance filter)	v1 — Included
Recurring Expenses	v2 — Extension Roadmap (Section 16)

1. Product Overview

SplitLedger is a group expense tracking and debt settlement web application. Authenticated users create groups, record shared expenses with categories and flexible split modes, edit or delete existing expenses, and track live balances. The system enforces strict financial invariants at every layer and surfaces the minimum-transaction debt simplification for settlement.

Technology Stack

Component	Technology
Backend	Python 3.11 + Flask + SQLAlchemy + Alembic
Frontend	React 18 (Vite) + TypeScript
Database	PostgreSQL 15
Auth	JWT (access token: 15 min, refresh token: 7 days)
Backend Validation	marshmallow
Frontend Validation	Zod
Backend Tests	pytest + pytest-cov
Frontend Tests	Vitest + React Testing Library

2. Goals & Non-Goals

Goals (v1)

- Authenticate users with JWT; protect all group and expense routes
- Create groups and manage membership
- Record expenses with explicit or equal-split modes and categories
- Edit expenses with full invariant re-validation
- Soft-delete expenses; exclude deleted records from balance computation
- Compute and display accurate real-time balances for every group member
- Record settlements between members with overpayment warnings
- Enforce all financial invariants at DB, service, and API layers
- Return structured, machine-readable errors for every failure case
- Be unit-testable at the service layer without a running database or auth context

Non-Goals (v1)

Feature	Reason Excluded
Recurring Expenses	Date math, cron scheduling, and is_active state machine are a 4-6 hour addition that proves nothing about correctness beyond what Expense Splits already prove. Deferred to v2.
Multi-currency	Requires FX rates; significantly changes balance computation
Email / push notifications	Infrastructure dependency; not in assessment scope

Feature	Reason Excluded
Receipt / file uploads	Storage dependency; out of scope
OAuth / social login	JWT username+password sufficient for v1; OAuth deferred
Group invitation links	Deferred to v2
Real-time websocket updates	Polling sufficient; deferred
CSV / PDF export	Additive; no logic impact; deferred
Mobile application	Web-only for v1

3. Glossary

Term	Definition
User	An authenticated person. Identified by JWT sub claim after login.
Group	A named collection of users who share expenses. Has an owner (creator).
Membership	The relationship between a User and a Group.
Expense	A payment by one member on behalf of participants. Has a split_mode, category, and optional soft-delete timestamp.
Split	The portion of an Expense for a specific member. All splits must sum exactly to expense.amount.
Split Mode	Either 'equal' (server computes even splits) or 'custom' (client sends explicit amounts).
Category	An enum label on an expense: food, transport, accommodation, entertainment, utilities, other.
Balance	Derived value: what a member is owed (+) or owes (−). Never stored; computed from source records.
Settlement	A direct payment from one member to another within a group, reducing outstanding debt.
Soft Delete	Marking an expense as deleted via deleted_at timestamp. Record stays in DB; excluded from balance computation.
Access Token	Short-lived JWT (15 min) used to authenticate API requests via Authorization header.
Refresh Token	Long-lived JWT (7 days) used to obtain a new access token without re-login.

4. Domain Model

Entities & Relationships

```
User
  id, username (unique), email (unique), password_hash, created_at

RefreshToken
  id, user_id (FK -> User CASCADE), token_hash (unique), expires_at, revoked

Group
  id, name, owner_user_id (FK -> User), created_at

Membership  [User <-> Group, junction]
```

```
  user_id (FK), group_id (FK)   UNIQUE(user_id, group_id)

Expense
  id, group_id (FK), paid_by_user_id (FK),
  description, amount NUMERIC(12,2),
  split_mode ENUM('equal','custom'),
  category   ENUM('food','transport','accommodation',
                  'entertainment','utilities','other'),
  created_at, updated_at, deleted_at  [NULL = active]

Split
  id, expense_id (FK CASCADE), user_id (FK), amount NUMERIC(12,2)
  UNIQUE(expense_id, user_id)

Settlement
  id, group_id (FK), paid_by_user_id (FK), paid_to_user_id (FK),
  amount NUMERIC(12,2), created_at
  CHECK(paid_by_user_id <> paid_to_user_id)
```

5. Core Invariants

> *Non-negotiable correctness rules. Enforced at DB constraint, service layer, and verified by automated tests. Never bypassed — not in migrations, scripts, or test utilities.*

INV-1	**Split Sum Integrity** For every active (non-deleted) Expense, sum(splits.amount) == expense.amount exactly. Tolerance: zero. Uses Decimal arithmetic. Applies to both CREATE and EDIT operations.
INV-2	**Group Balance Sum Zero** sum(net_balance(member)) == 0 for every group. Mathematical consequence of INV-1. Verified as integration test assertion. Deleted expenses are excluded from this computation.
INV-3	**Settlement Non-Negativity + Overpayment** amount > 0. If amount exceeds current debt, WARN in response but still record. Overpayment (pre-payment) is valid. Does NOT block the request.
INV-4	**No Self-Settlement** paid_by_user_id != paid_to_user_id. DB CHECK constraint + service hard block. Returns 422 SELF_SETTLEMENT.
INV-5	**Payer Must Be Group Member** paid_by_user_id must be a member of the group at time of recording. Applies to both Expense and Settlement.
INV-6	**Split Participant Must Be Group Member** Every split.user_id must be a member of the expense's group. Applies to CREATE and EDIT.
INV-7	**Amount Precision** All monetary amounts: NUMERIC(12,2). Input with more than 2 decimal places is REJECTED, not rounded. Applies everywhere amounts are accepted.

INV-8	**Soft-Delete Balance Exclusion** Expenses with deleted_at IS NOT NULL are excluded from balance computation. Deleting an expense is equivalent — from a balance perspective — to it never having existed.

INV-9	**Auth Resource Ownership** Only authenticated members of a group may read or write its expenses, settlements, or balances. Non-members receive 403 FORBIDDEN, not 404. Group owner may remove members; members may remove themselves.

6. Database Schema

```
-- Auth
CREATE TABLE users (
    id            SERIAL PRIMARY KEY,
    username      VARCHAR(50)  NOT NULL UNIQUE CHECK(LENGTH(TRIM(username))>0),
    email         VARCHAR(255) NOT NULL UNIQUE CHECK(email LIKE '%@%'),
    password_hash VARCHAR(255) NOT NULL,
    created_at    TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

CREATE TABLE refresh_tokens (
    id          SERIAL PRIMARY KEY,
    user_id     INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    token_hash  VARCHAR(255) NOT NULL UNIQUE,
    expires_at  TIMESTAMPTZ NOT NULL,
    revoked     BOOLEAN NOT NULL DEFAULT FALSE,
    created_at  TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- Groups
CREATE TABLE groups (
    id            SERIAL PRIMARY KEY,
    name          VARCHAR(100) NOT NULL CHECK(LENGTH(TRIM(name))>0),
    owner_user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE RESTRICT,
    created_at    TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

CREATE TABLE memberships (
    id        SERIAL PRIMARY KEY,
    user_id   INTEGER NOT NULL REFERENCES users(id)  ON DELETE RESTRICT,
    group_id  INTEGER NOT NULL REFERENCES groups(id) ON DELETE RESTRICT,
    joined_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    UNIQUE (user_id, group_id)
);

-- Enums
CREATE TYPE split_mode_enum AS ENUM ('equal', 'custom');
CREATE TYPE category_enum   AS ENUM ('food','transport','accommodation',
                                     'entertainment','utilities','other');

-- Expenses
CREATE TABLE expenses (
    id              SERIAL PRIMARY KEY,
    group_id        INTEGER NOT NULL REFERENCES groups(id) ON DELETE RESTRICT,
    paid_by_user_id INTEGER NOT NULL REFERENCES users(id)  ON DELETE RESTRICT,
    description     VARCHAR(255) NOT NULL CHECK(LENGTH(TRIM(description))>0),
    amount          NUMERIC(12,2) NOT NULL CHECK(amount > 0),
    split_mode      split_mode_enum NOT NULL DEFAULT 'custom',
    category        category_enum   NOT NULL DEFAULT 'other',
    created_at      TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at      TIMESTAMPTZ,
    deleted_at      TIMESTAMPTZ     -- NULL = active; NOT NULL = soft-deleted
```

```
);

CREATE TABLE splits (
    id         SERIAL PRIMARY KEY,
    expense_id INTEGER NOT NULL REFERENCES expenses(id) ON DELETE CASCADE,
    user_id    INTEGER NOT NULL REFERENCES users(id)    ON DELETE RESTRICT,
    amount     NUMERIC(12,2) NOT NULL CHECK(amount > 0),
    UNIQUE (expense_id, user_id)
);

-- Settlements
CREATE TABLE settlements (
    id              SERIAL PRIMARY KEY,
    group_id        INTEGER NOT NULL REFERENCES groups(id) ON DELETE RESTRICT,
    paid_by_user_id INTEGER NOT NULL REFERENCES users(id)  ON DELETE RESTRICT,
    paid_to_user_id INTEGER NOT NULL REFERENCES users(id)  ON DELETE RESTRICT,
    amount          NUMERIC(12,2) NOT NULL CHECK(amount > 0),
    created_at      TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    CHECK (paid_by_user_id <> paid_to_user_id)
);
```

Indexes

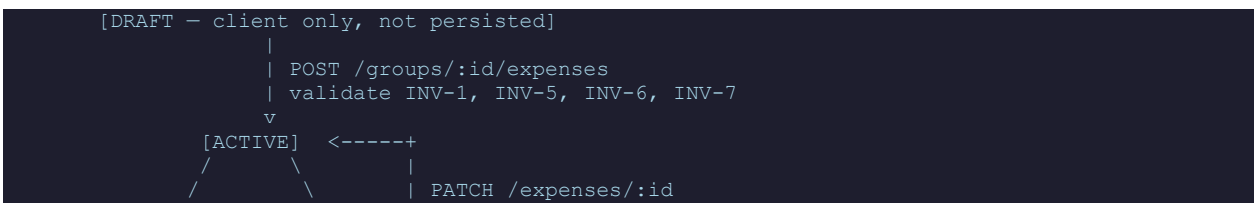
```
CREATE INDEX idx_refresh_tokens_user  ON refresh_tokens(user_id);
CREATE INDEX idx_memberships_group    ON memberships(group_id);
CREATE INDEX idx_memberships_user     ON memberships(user_id);
CREATE INDEX idx_expenses_group       ON expenses(group_id);
CREATE INDEX idx_expenses_active      ON expenses(group_id) WHERE deleted_at IS NULL;
CREATE INDEX idx_splits_expense       ON splits(expense_id);
CREATE INDEX idx_settlements_group    ON settlements(group_id);
```

ON DELETE Policy

Table	FK	Policy	Reason
refresh_tokens	user_id	CASCADE	Token is owned by user; delete together
memberships	user_id, group_id	RESTRICT	Cannot delete user/group with active memberships
expenses	group_id, paid_by_user_id	RESTRICT	Cannot delete group/user with associated expenses
splits	expense_id	CASCADE	Splits are owned by their expense
splits	user_id	RESTRICT	Cannot delete user with splits
settlements	group_id, user_ids	RESTRICT	Cannot delete group/user with settlements

7. State Machines

7.1 Expense Lifecycle



```
           /          \      | re-validate INV-1 atomically
          v            +-----+
      [DELETED]
      deleted_at = NOW()
      excluded from all balance queries (INV-8)
      splits remain for audit — not recoverable in v1
```

> *Hard (permanent) deletion is not exposed via API in v1. DELETE sets deleted_at only. All queries filter WHERE deleted_at IS NULL.*

7.2 Expense Edit Rules

- Only the original payer or group owner may edit an expense — others get 403 FORBIDDEN
- If amount OR splits are provided, BOTH must be present and INV-1 is re-validated atomically before any DB write
- Partial updates are allowed — updating description only does not re-validate splits
- If split_mode changes to 'equal', client must NOT send splits; server computes them fresh
- If split_mode changes to 'custom', client must send the new explicit splits array
- updated_at is set to NOW() on every successful PATCH
- A deleted expense cannot be edited — returns 422 EXPENSE_DELETED

7.3 Authentication Token Lifecycle

8. API Specification

Base URL: /api/v1

Auth: Bearer Token

All endpoints except POST /auth/register and POST /auth/login require:

```
Authorization: Bearer <access_token>
```

Missing / invalid token → 401. Expired → 401 TOKEN_EXPIRED. Authenticated non-member → 403 FORBIDDEN.

Response Envelope

```
// Success
{ "data": { ... }, "warnings": [] }

// Error
{ "error": { "code": "SPLIT_SUM_MISMATCH",
             "message": "Split amounts (90.00) do not equal expense amount (100.00)",
             "field": "splits" } }
```

8.1 Authentication

Method	Path	Description	Auth
POST	/auth/register	Create account; return tokens	No
POST	/auth/login	Authenticate; return tokens	No
POST	/auth/refresh	Exchange refresh token for new access token	No (RT in body)
POST	/auth/logout	Revoke refresh token	Yes
GET	/auth/me	Return current user profile	Yes

POST /auth/register — Field Rules

Field	Type	Rule
username	string	Required, 3–50 chars, alphanumeric + underscore, unique
email	string	Required, valid email format, unique
password	string	Required, min 8 chars, at least one letter and one digit

Response 201: { data: { user: {...}, access_token, refresh_token } }

8.2 Groups & Membership

Method	Path	Description
POST	/groups	Create group (caller becomes owner + member)
GET	/groups	List groups the current user belongs to
GET	/groups/:id	Get group with members (caller must be member)
POST	/groups/:id/members	Add user to group (owner only)
DELETE	/groups/:id/members/:uid	Remove member (owner removes anyone; member removes self)

8.3 Expenses

Method	Path	Description
POST	/groups/:id/expenses	Record a new expense
GET	/groups/:id/expenses	List active expenses (deleted_at IS NULL)
GET	/expenses/:id	Get expense detail with splits
PATCH	/expenses/:id	Edit expense (amount, description, splits, category, split_mode)

Method	Path	Description
DELETE	/expenses/:id	Soft-delete expense (sets deleted_at = NOW())

POST /groups/:id/expenses — Field Reference

Field	Type	Required	Rule
paid_by_user_id	integer	Yes	Must be a group member (INV-5)
description	string	Yes	Non-empty after trim, max 255 chars
amount	decimal string	Yes	Positive, max 2 decimal places (INV-7)
split_mode	enum	No	Default: 'custom'. Values: 'equal' | 'custom'
category	enum	No	Default: 'other'. See category enum in Section 11
splits	array	If custom	Required when split_mode='custom'. Must be absent when 'equal'.
splits[].user_id	integer	Yes (custom)	Must be a group member (INV-6)
splits[].amount	decimal string	Yes (custom)	Positive, max 2dp; sum must == amount (INV-1)

> *When split_mode='equal', the server divides amount evenly among all current group members. If the amount is not evenly divisible, the remainder (1 cent) is added to the payer's split. Client must NOT send a splits array in equal mode — returns 400 SPLITS_SENT_FOR_EQUAL_MODE if present.*

PATCH /expenses/:id — Partial Update Rules

- Any subset of fields may be sent. Only provided fields are updated.
- If amount OR splits are provided, BOTH must be present. INV-1 is re-validated atomically.
- If split_mode changes to 'equal': client must NOT send splits (server computes fresh).
- If split_mode changes to 'custom': client must send the new splits array.
- Cannot edit a deleted expense — 422 EXPENSE_DELETED.
- Only original payer or group owner may edit — others get 403 FORBIDDEN.

8.4 Balances

Method	Path	Description
GET	/groups/:id/balances	Compute balances for all members; return simplified debts
GET	/groups/:id/balances?category=food	Balances scoped to a single category (informational only)

```
{ "data": {
    "group_id": 1,
    "balances": [
      { "user_id": 1, "name": "Alice", "balance": "200.00" },
      { "user_id": 2, "name": "Bob",   "balance": "-100.00" },
      { "user_id": 3, "name": "Carol", "balance": "-100.00" }
    ],
    "simplified_debts": [
      { "from_user_id": 2, "from_name": "Bob",
        "to_user_id": 1,   "to_name": "Alice", "amount": "100.00" },
      { "from_user_id": 3, "from_name": "Carol",
        "to_user_id": 1,   "to_name": "Alice", "amount": "100.00" }
    ],
    "balance_sum": "0.00"
} }
```

> *balance_sum is always "0.00". Returned explicitly so clients can assert it. The backend also asserts this internally before responding — a non-zero sum surfaces as a 500 and is logged for investigation.*

8.5 Settlements

Method	Path	Description
POST	/groups/:id/settlements	Record a debt payment between two members
GET	/groups/:id/settlements	List all settlements for a group

9. Business Logic Rules

9.1 Balance Formula (Canonical)

> *This is the single source of truth for balance computation. It must not be reimplemented differently anywhere else. Any divergence is a bug.*

```
def compute_balances(group_id, db_session) -> dict[int, Decimal]:
    """
    Returns {user_id: net_balance} for all active members.
    Only considers expenses where deleted_at IS NULL  (INV-8).
    sum(return_value.values()) MUST equal Decimal("0.00").
    """
    balances = defaultdict(Decimal)

    # Credit payer for full expense amount
    for expense in get_active_expenses(group_id, db):  # WHERE deleted_at IS NULL
        balances[expense.paid_by_user_id] += expense.amount

    # Debit each participant their split
    for split in get_splits_for_active_expenses(group_id, db):
        balances[split.user_id] -= split.amount

    # Net settlements
    for s in get_settlements(group_id, db):
        balances[s.paid_by_user_id] += s.amount
        balances[s.paid_to_user_id] -= s.amount

    # Ensure all members appear even if balance is zero
    for member_id in get_member_ids(group_id, db):
        balances.setdefault(member_id, Decimal('0.00'))

    return dict(balances)
```

9.2 Equal Split Computation

```
def compute_equal_splits(
    amount: Decimal,
    participant_ids: list[int],
    payer_id: int
) -> list[dict]:
    """
    Divides amount evenly. Remainder (1 cent) goes to payer's split.
    Guarantees sum(splits) == amount  —  maintains INV-1.
    """
    n    = len(participant_ids)
    base = (amount / n).quantize(Decimal('0.01'), rounding=ROUND_DOWN)
```

```
    rem  = amount - (base * n)

    splits = [{'user_id': uid, 'amount': base} for uid in participant_ids]

    if rem > 0:
        payer_split = next((s for s in splits if s['user_id'] == payer_id),
                           splits[0])  # fallback: first participant
        payer_split['amount'] += rem

    assert sum(s['amount'] for s in splits) == amount  # INV-1 guarantee
    return splits
```

9.3 Debt Simplification (AI-Generated — Test-Verified)

> *Generated with AI assistance. MUST be verified by tests/unit/test_debt_simplification.py. The test suite is the source of truth, not this prose.*

Greedy minimum cash flow: repeatedly match the largest debtor with the largest creditor until all balances reach zero. For N members, produces at most N−1 transactions.

10. Error Handling Contract

Principles

- Never return 500 for user errors
- Error codes are machine-readable enums; messages are human-readable
- Return the first validation failure — do not aggregate
- Never leak stack traces; production 500s log the real error and return a generic message
- Auth: 401 = not authenticated; 403 = authenticated but not authorized (distinct, always)

Error Code Registry

Code	HTTP	When Used
MISSING_FIELD	400	Required field absent from request
INVALID_FIELD	400	Field fails format or range validation
INVALID_AMOUNT_PRECISION	400	Amount has more than 2 decimal places
INVALID_CATEGORY	400	category value not in allowed enum
INVALID_SPLIT_MODE	400	split_mode not 'equal' or 'custom'
SPLITS_SENT_FOR_EQUAL_MODE	400	Client sent splits array when split_mode='equal'
DUPLICATE_SPLIT_USER	400	Same user_id appears twice in splits array
DUPLICATE_EMAIL	409	Email already registered
DUPLICATE_USERNAME	409	Username already taken
ALREADY_MEMBER	409	User already in group
USER_NOT_FOUND	404	Referenced user_id does not exist
GROUP_NOT_FOUND	404	Referenced group_id does not exist
EXPENSE_NOT_FOUND	404	Referenced expense_id does not exist

Code	HTTP	When Used
PAYER_NOT_MEMBER	422	paid_by_user_id is not in the group (INV-5)
SPLIT_USER_NOT_MEMBER	422	A split user_id is not in the group (INV-6)
SPLIT_SUM_MISMATCH	422	sum(splits.amount) != expense.amount (INV-1)
RECIPIENT_NOT_MEMBER	422	paid_to_user_id is not in the group
SELF_SETTLEMENT	422	paid_by == paid_to in settlement (INV-4)
EXPENSE_DELETED	422	Attempt to edit a soft-deleted expense
INVALID_CREDENTIALS	401	Wrong username or password
TOKEN_MISSING	401	No Authorization header provided
TOKEN_INVALID	401	Token signature invalid or malformed
TOKEN_EXPIRED	401	Access token expired; client should use /auth/refresh
REFRESH_TOKEN_INVALID	401	Refresh token not found, revoked, or expired
FORBIDDEN	403	Authenticated but not a member of this group (INV-9)
INTERNAL_ERROR	500	Unexpected server-side error

Warning Code Registry

Code	When Used
OVERPAYMENT	Settlement amount exceeds current outstanding debt between the two parties (INV-3)

11. Validation Rules

Layer Responsibilities

Layer	Owns	Does NOT Own
marshmallow schema	Field types, formats, lengths, enums, decimal precision	Cross-entity rules, DB queries
service layer	INV-1 through INV-9, membership checks, edit permissions	HTTP context, request parsing, JWT
DB constraints	Last defense: CHECK, UNIQUE, FK RESTRICT, enum types	Business logic above schema level
Zod (frontend)	Client-side mirror of marshmallow rules	Server-side invariants

> *Never duplicate validation logic across layers. If a rule belongs in a schema, it must not also appear in the route handler or service. If it belongs in the service, it must not be re-implemented in the route.*

Category Enum Values

Value	Display Label
food	Food & Drink

Value	Display Label
transport	Transport
accommodation	Accommodation
entertainment	Entertainment
utilities	Utilities
other	Other (default)

12. Testing Requirements

Philosophy

Tests are not optional. Every invariant and business rule has a corresponding test. Tests are written before or alongside new code — never after. Coverage on core service modules is a hard requirement.

Unit Tests (no DB, no Flask, no auth context)

Location: backend/tests/unit/

- test_compute_balances.py — multiple expense/settlement combos; assert sum==0; assert deleted expenses excluded
- test_equal_split.py — even split, odd-cent remainder goes to payer, payer not in participants fallback
- test_debt_simplification.py — two-person, triangle reduces to two txns, all-zero returns empty, large group
- test_split_sum_invariant.py — INV-1 check in isolation for both CREATE and EDIT paths
- test_validation_schemas.py — valid and invalid inputs for every marshmallow schema

Integration Tests (real DB via fixture)

Location: backend/tests/integration/

- test_auth.py — register, login, refresh, logout, protected route without token, expired token
- test_expenses.py — happy path, INV-1 mismatch 422, payer not member 422, equal split mode, category
- test_expense_edit.py — edit amount+splits re-validates INV-1; edit deleted expense 422; non-owner 403
- test_expense_delete.py — soft-delete removes from balance computation; edit after delete 422
- test_balances.py — balance_sum==0 after expenses+settlements; deleted expense excluded from sum
- test_settlements.py — happy path; self-settlement 422; overpayment 201 with warning
- test_categories.py — category filter on balances; invalid category 400

Coverage Requirements

Module	Required Line Coverage
app/services/balance_service.py	>= 90%
app/services/expense_service.py	>= 90%
app/services/settlement_service.py	>= 90%
app/services/auth_service.py	>= 90%

Module	Required Line Coverage
app/schemas/	>= 90%

> *Coverage below 90% on these modules is a failing condition for the submission.*

13. Project Structure

```
splitledger/
├── backend/
│   ├── app/
│   │   ├── __init__.py              # Flask app factory
│   │   ├── models/
│   │   │   ├── user.py
│   │   │   ├── refresh_token.py
│   │   │   ├── group.py
│   │   │   ├── membership.py
│   │   │   ├── expense.py           # split_mode, category, deleted_at
│   │   │   ├── split.py
│   │   │   └── settlement.py
│   │   ├── schemas/                 # marshmallow — validation only
│   │   │   ├── auth_schema.py
│   │   │   ├── group_schema.py
│   │   │   ├── expense_schema.py    # split_mode, category, PATCH schema
│   │   │   └── settlement_schema.py
│   │   ├── services/                # all business logic — no HTTP awareness
│   │   │   ├── auth_service.py      # JWT create, password hash, revocation
│   │   │   ├── group_service.py
│   │   │   ├── expense_service.py   # create, edit, soft-delete, equal split
│   │   │   ├── balance_service.py   # compute_balances, simplify_debts
│   │   │   └── settlement_service.py
│   │   ├── routes/                  # Flask blueprints — HTTP only
│   │   │   ├── auth.py
│   │   │   ├── groups.py
│   │   │   ├── expenses.py
│   │   │   ├── balances.py
│   │   │   └── settlements.py
│   │   ├── middleware/
│   │   │   └── auth_middleware.py   # @require_auth decorator
│   │   ├── errors.py                # AppError base + error code constants
│   │   └── extensions.py            # SQLAlchemy, marshmallow init
│   ├── migrations/                  # Alembic — append-only
│   └── tests/
│       ├── unit/
│       └── integration/
│           └── conftest.py          # test DB, client, auth helper
│
├── frontend/
│   └── src/
│       ├── api/                     # typed fetch wrappers, auth header injection
│       ├── auth/                    # AuthContext, useAuth hook, token storage
│       ├── components/
│       ├── pages/
│       ├── types/                   # TypeScript interfaces matching API shapes
│       └── schemas/                 # Zod schemas (mirrors marshmallow)
│
├── GUIDE(AI & HUMAN).md             # Guidance for AI & Human also — read before any edit
├── ARCHITECTURE.md                  # system design, invariants, layer rationale
├── AI_USAGE.md                      # what was AI-generated, verified, and changed
├── README.md                        # quick start, decisions, known limitations
├── .gitignore
├── docker-compose.yml
└── .env.example
```

Architectural Boundaries

Layer	Allowed	Not Allowed
Route	Parse request, call schema, call service, return envelope	Business logic, DB queries, JWT operations
Schema	Field validation and deserialization	DB calls, service calls
Service	Business logic, invariant checks, DB via models	Import from routes, HTTP context, JWT
Model	SQLAlchemy table and relationship definitions	Any logic or computation
Middleware	Extract JWT, attach user_id to request context	Business logic

14. AI Guidance File (CLAUDE.md)

> *This file lives at the repo root. Any AI agent must read this in full before modifying any code. It defines enforceable rules, not aspirational guidance. Code that violates these rules will be rejected in review regardless of who or what produced it.*

```
# CLAUDE.md — AI Agent Guidance for SplitLedger

Read ARCHITECTURE.md before making any structural or service-layer changes.

## Hard Rules — No Exceptions

### Rule 1 — Never Bypass an Invariant
INV-1: sum(splits.amount) == expense.amount exactly. Use Decimal. Never float.
INV-4: paid_by_user_id != paid_to_user_id in any settlement.
INV-8: deleted_at IS NOT NULL expenses excluded from every balance computation.
INV-9: only authenticated group members may read or write group data.
If asked to write code that violates an invariant, refuse and explain why.

### Rule 2 — Always Use Decimal for Money
from decimal import Decimal   # ALWAYS
float(10.1) + float(20.2) = 30.299999999999997  # WRONG
Decimal('10.10') + Decimal('20.20') = Decimal('30.30')  # CORRECT
Applies to: service args, return values, test assertions, equal split math.

### Rule 3 — Layer Boundaries Are Sacred
Routes -> Schemas -> Services -> Models -> Database
Routes call services. Routes contain no logic.
Services contain all logic. Services have no Flask/HTTP knowledge.
Models define tables. Models contain no logic.
Middleware extracts JWT only. It does not perform business authorization.

### Rule 4 — Validation Lives in Schemas
Field rules (type, length, enum, precision) -> marshmallow schemas.
Cross-entity rules (membership, INV-1 check) -> service functions.
Never duplicate between layers.

### Rule 5 — Error Codes Are a Versioned Contract
Use only codes registered in errors.py and ARCHITECTURE.md Section 8.
New code requires: add to errors.py + update registry + add test.
Messages may change freely. Codes may not.

### Rule 6 — Tests Are Required for Business Logic
New service function = new unit test (tests/unit/).
New endpoint = new integration test covering happy path + error cases.
Coverage on core services must remain >= 90%.
```

```
### Rule 7 — Alembic Migrations Are Append-Only
Never edit or delete a migration file in migrations/versions/.
If a change is needed post-apply: create a new corrective migration.

### Rule 8 — Soft Delete = Excluded, Not Gone
deleted_at IS NOT NULL = excluded from all balance queries.
Always use get_active_expenses() helper — never query Expense directly.
Never hard-delete expense rows via the API.

## Common Mistakes to Avoid
1. Float creeping in: Decimal('0.10') / 3 coerces to float. Use quantize().
2. Missing deleted_at filter on new expense queries.
3. Logic in a route function (> ~15 lines = it belongs in a service).
4. New error code not registered in errors.py before use.
5. assert for invariant enforcement in production code (use explicit if + raise).
6. Conflating 401 (unknown identity) with 403 (known, unauthorized). Never swap.

## Out of Scope for v1 — Do Not Implement
Recurring expenses, multi-currency, email, file uploads, OAuth,
websockets, group invite links, CSV export, mobile.
Decline and reference ARCHITECTURE.md Section 10.

## When Unsure
Ask. Do not guess. Reference ARCHITECTURE.md.
Invariants -> Section 4  |  Error codes -> Section 8  |  Roadmap -> Section 10
```

15. Risks & Known Tradeoffs

Risk	Severity	Mitigation
JS float arithmetic on money display	Medium	Amounts transmitted as strings. Frontend never uses JS number type for monetary arithmetic.
Balance sum drift from future bugs	Medium	GET /balances asserts sum==0 before responding. Drift surfaces immediately as a 500, logged server-side.
Concurrent expense creation (single server)	Low	Balances computed at read time from source records. No denormalized balance to corrupt. Concurrent writes cause ordering ambiguity only.
Access token not revocable on logout	Low	Access tokens are 15-min short-lived. Refresh tokens are revocable in DB. Accepted tradeoff for stateless auth.
marshmallow <-> Zod schema drift	Low	Manual sync in v1. v2 should generate Zod from OpenAPI spec automatically.
Member removed mid-expense	Medium	Split user_ids remain in DB after member removal. Balance computation is unaffected — by design. Documented non-issue.

16. Extension Roadmap (v2+)

Feature	Why Deferred	Extension Point
Recurring Expenses	Date math (month-end edge cases), cron scheduling, and is_active state machine require 4–6 hours and prove nothing about correctness beyond what Expense Splits already demonstrate.	New RecurringExpense + RecurringSplit tables; recurring_service.py; frequency enum; cron or manual /generate trigger; next_due_date advancement logic

Feature	Why Deferred	Extension Point
OAuth / social login	JWT username+password sufficient for v1	Add oauth_provider to User; new auth flow alongside existing JWT
Multi-currency	Fundamentally changes balance computation	Add currency to Group, FX rate to Expense; rewrite balance formula
Cron-triggered recurring	Follows from Recurring Expenses	Background job reads next_due_date <= today; calls generate logic
Email notifications	Infrastructure dependency	Hook into expense/settlement service events
Group invitation links	Nice-to-have; not core	Add invite_token to Group; auth-aware join flow
Expense audit trail	Additive; no existing table changes	Add expense_history table on every PATCH; pure append
CSV / PDF export	Pure read; no invariant impact	GET /groups/:id/export; no logic changes
Pagination	Schema change only	Add ?page=&limit= query params to list endpoints

End of SplitLedger Product Specification v2.1.0

This document is the source of truth. Code that contradicts this document is wrong.