Phase 1 — Configuration & App bootstrap

1) core/settings.py

Purpose: Centralize configuration pulled from env vars.

Do: Define names for DATABASE\_URL, PRICE\_SERVICE\_BASE\_URL, SECRET\_KEY, ALGORITHM, token TTLs, APP\_ENV, LOG\_LEVEL, PORT.

Done when: You can print settings in the next step without errors.

2) .env.example

Purpose: Developer template of required env vars.

Do: Add keys from settings.py with placeholder values.

Done when: Copying it to .env makes the app read values in dev.

3) app/main.py

Purpose: FastAPI application object + router registration.

Do: Create the app, include routers.auth, routers.portfolios, routers.transactions. Add a simple /health endpoint if you like.

Done when: uvicorn app.main:app starts and /docs loads.

Phase 2 — Database plumbing & models

4) app/database.py

Purpose: All DB plumbing lives here.

Do: Create the async engine (from DATABASE\_URL), sessionmaker, get\_db() dependency, and the declarative Base.

Done when: You can import get\_db and open/close a session (you’ll use it in routers).

5) app/models.py

Purpose: Source of truth for tables.

Do (MVP):

User (id, email unique, password\_hash, created\_at).

Portfolio (id, user\_id FK→User, name nullable, created\_at/updated\_at).

Transaction (id, portfolio\_id FK→Portfolio, btc\_amount (8 dp), timestamp\_original (tz-aware), timestamp\_hour\_rounded (UTC), price\_at\_purchase (2 dp), created\_at).

Done when: Base.metadata shows all three tables and FKs.

Migrations: you’ll generate them later—just keep models stable for now.

Phase 3 — API contracts (schemas)

6) app/schemas/users.py

Purpose: Request/response shapes for auth.

Do: UserCreate, UserRead, LoginRequest, Token (access[, refresh] if you want it now).

Done when: The shapes line up with your planned endpoints.

7) app/schemas/portfolios.py

Purpose: Contracts for portfolio reads (and later creates if you add).

Do: PortfolioRead (id, name, created\_at, updated\_at), PortfolioCreate (optional, for later).

Done when: Models ↔ schemas fields match.

8) app/schemas/transactions.py

Purpose: Contracts for transactions.

Do: TransactionCreate (portfolio\_id, btc\_amount, timestamp) and TransactionRead (full view).

Done when: You’re clear on inputs/outputs your routes will accept/return.

Phase 4 — Security & auth

9) core/security.py

Purpose: Authentication/authorization primitives.

Do:

Password hashing/verify (bcrypt).

Access token (JWT) create/verify.

Dependency get\_current\_user reading Authorization: Bearer <token>, loading user from DB, 401 on failure.

Done when: You can call get\_current\_user in a protected route to retrieve the user.

10) app/routers/auth.py

Purpose: Auth endpoints.

Do:

POST /auth/register (email+password → create user).

POST /auth/login (email+password → return access token [and refresh later if you want]).

(Optional now / later) POST /auth/refresh, POST /auth/logout.

Done when: You can register & login and receive a valid token you can use on protected endpoints.

Phase 5 — Domain endpoints (ownership enforced)

11) app/routers/portfolios.py

Purpose: Portfolio read/list, scoped to the authenticated user.

Do:

Add Depends(get\_current\_user) on the endpoints.

GET /portfolios/{id} → return only if portfolio.user\_id == current\_user.id.

GET /portfolios?limit=&offset= → list only the user’s portfolios.

Done when: A logged-in user can only see their own portfolios. Others are 404/empty.

12) app/routers/transactions.py

Purpose: Transaction create/read/list, price lookup.

Do:

Add Depends(get\_current\_user) on all endpoints.

Create (POST /transactions) flow:

Validate payload.

Confirm portfolio\_id belongs to current\_user (else 404/403).

Round timestamp to nearest hour (UTC; ≥30 min rounds up).

Call price\_service (use PRICE\_SERVICE\_BASE\_URL) to fetch price for rounded hour.

Persist transaction with price.

Return created transaction.

GET /transactions/{id} → return only if it belongs to a portfolio owned by current\_user.

GET /transactions?portfolio\_id=&limit=&offset= → list user’s transactions (filter by portfolio if provided and owned).

Done when: You can create a transaction end‑to‑end with a valid token, and listings/reads are owner‑scoped.

For the price lookup, keep the small HTTP call inside this router for now (since you don’t want to add files). Use a short timeout and translate failure to 424 (no price) or 502 (service/network error).

Phase 6 — Tests (minimum confidence)

13) tests/ (you choose filenames)

Purpose: Lock behavior while you iterate.

Do (minimum set):

Auth: register/login success; bad password → 401.

Ownership: user A can’t see user B’s portfolios/transactions; creating a transaction for someone else’s portfolio → 404/403.

Transactions: happy path create (mock price\_service call in test), list filtering by portfolio, get by id.

Rounding: a couple of asserts that 10:29:59 rounds down and 10:30:00 rounds up (UTC).

Done when: Tests cover your core invariants and pass locally.

Phase 7 — Migrations & run

14) Migrations (later, when models are stable)

Purpose: Real DB schema.

Do: Initialize/upgrade DB to create users, portfolios, transactions.

Done when: The service runs connected to Postgres and endpoints work against it.

What each file is “for” (quick reference)

core/settings.py — all environment-driven config in one place.

core/security.py — password hashing, token functions, get\_current\_user.

app/database.py — async engine, sessionmaker, Base, get\_db dependency.

app/models.py — ORM tables & relationships (User, Portfolio, Transaction).

app/schemas/users.py — DTOs for register/login/tokens/public user.

app/schemas/portfolios.py — DTOs for portfolio responses (and future creates).

app/schemas/transactions.py — DTOs for transaction create/response.

app/routers/auth.py — register/login[/refresh/logout].

app/routers/portfolios.py — read/list portfolios (owner-only).

app/routers/transactions.py — create/read/list transactions (owner-only + price lookup).

app/main.py — app bootstrap & router includes.

tests/ — safety net for auth, ownership, rounding, CRUD.

.env.example — template of required environment variables.

requirements.txt — installable deps for local/Docker.

Suggested implementation order (checklist)

settings → 2) database → 3) models → 4) main → 5) schemas (users/portfolios/transactions) →

security → 7) auth router → 8) portfolios router (owner scope) →

transactions router (rounding + price lookup + create + reads) →

tests → 11) migrations.

When you’re ready to start coding the first file, say which step you want to tackle (e.g., “settings” or “database”), and I’ll walk you through the exact fields and edge cases to cover.