WealthNest

A Fintech Investment Platform

Definition :

WealthNest is a digital investment and portfolio management platform that enables users to securely buy and track stocks, mutual funds, and other assets. It provides wallet management, financial analytics, and real-time prices, along with gamified investing features to enhance user engagement. The platform also includes a dedicated admin panel for compliance, monitoring, and financial oversight, ensuring security and transparency at every level.

2. Proposed System

2.1 Scope

WealthNest is a digital investment platform that allows users to securely invest in stocks, mutual funds, and other assets. It provides real-time market tracking, portfolio management, and analytics to help users make informed decisions. The platform supports secure wallet management and digital transactions. Administrators can monitor activity, ensure compliance, and manage user accounts. A points system with a leaderboard encourages user engagement. Overall, WealthNest simplifies investing while maintaining security, transparency, and ease of use.

2.2 Objectives

- Enable users to create and manage investment portfolios efficiently.
- Provide real-time tracking of stocks, mutual funds, and other assets.
- Facilitate secure wallet management and digital transactions.
- Offer analytics and reports for better investment decision-making.
- Implement a points system with a leaderboard to enhance user engagement.
- Provide an admin panel for compliance, monitoring, and management.

2.3 Constraints

2.3.1 Hardware Constraints

- Requires devices with stable internet for real-time updates.
- Performance may be limited on low-end devices.
- Sufficient storage is needed for historical transaction and portfolio data.

2.3.2 Software Constraints

- Requires updated browsers or mobile apps for full functionality.
- Integration with third-party financial APIs may have limitations.
- Security depends on correct implementation of encryption and authentication protocols.

2.4 Advantages

- Simplifies investment and portfolio management for users.
- Provides real-time analytics and portfolio tracking.
- Enhances security and reliability of financial transactions.
- Points and leaderboard systems encourage engagement without complicating the platform.
- Admin panel allows for compliance and monitoring.

2.5 Limitations

- Offline access to real-time financial data is not available.
- Advanced AI-driven investment recommendations are not implemented.
- Multi-market or international asset support is not included.
- Payment integration is not implemented in the current version.
- Some custom integrations with external financial platforms are not available

---

Features

⇒ Product Features

User Panel:

- ○ Signup/Login with email verification.
- ○ Wallet management (deposit, withdraw, view balance).
- ○ Portfolio management (view holdings, P&L, charts).
- ○ Buy assets (stocks, mutual funds, lumpsum or SIP).
- ○ View activity history.
- ○ Achievements, points, leaderboard.

Admin Panel:

- ○ Admin login with role-based access.
- ○ View and manage users.
- ○ Monitor transactions and system funds.
- ○ Ban/unban users, handle flagged accounts.
- ○ Manage asset listings and reconcile prices.
- ○ Generate reports and review audit logs.

Requirements

Functional Requirements

User Panel

- Authentication
  - FR-1: User shall be able to register with email & password.
  - FR-2: System shall send a verification email with link.
  - FR-3: User shall be able to log in after verification.
- Wallet Management
  - FR-4: User shall be able to deposit funds via payment gateway.
  - FR-5: User shall be able to withdraw funds.
  - FR-6: User shall view wallet balance and transaction ledger.
- Portfolio Management

- FR-7: User shall view all holdings with quantity, average price, market price, and P&L.
    - FR-8: User shall buy assets (stocks, mutual funds) via lumpsum or SIP.
    - FR-9: User shall see charts and statistics of portfolio performance.
- Activity & Engagement
    - FR-10: User shall view transaction history with filters.
    - FR-11: System shall award achievements for milestones.
    - FR-12: System shall maintain a leaderboard of points.

Admin Panel

- Authentication & Roles
    - FR-13: Admins shall log in via secure credentials.
    - FR-14: Admin roles: Superadmin, Employee.
- User Management
    - FR-15: Admins shall view and filter user accounts.
    - FR-16: Admins shall ban/unban users.
- Transactions & Compliance
    - FR-17: Admins shall view and reconcile transactions.
    - FR-18: Admins shall manage flagged transactions.
- Assets & Financial Oversight
    - FR-19: Admins shall add/edit assets.
    - FR-20: Admins shall view and reconcile system financials.
- Audit & Reporting
    - FR-21: System shall maintain an immutable Admin Audit Log.
    - FR-22: Admins shall export reports in CSV/PDF format.

Non-Functional Requirements

- Performance:
    - NFR-1: Dashboard shall load in ≤3 seconds for 95% of users.
    - NFR-2: Transaction operations shall complete in ≤2 seconds (excluding payment gateway delays).
- Security:

    - NFR-3: All sensitive data (passwords, OTPs) shall be stored with encryption.
    - NFR-4: Multi-factor authentication for admin logins.
    - NFR-5: Role-based access control enforced.
- Reliability & Availability:
    - NFR-6: System uptime ≥ 99.5%.
    - NFR-7: Data backups taken daily.
- Usability:
    - NFR-8: Mobile UI must be responsive and intuitive.
    - NFR-9: Provide simple onboarding guide.
- Maintainability & Extensibility:
    - NFR-10: Code shall follow modular architecture for easy enhancements.

Database Design

1. Users

Column Name	Data Type	Description
user_id (PK)	INT	Unique identifier for each user
name	VARCHAR(100)	Full name of the user
email	VARCHAR(150)	Unique email (login credential)
phone	VARCHAR(15)	Contact number
password_hash	VARCHAR(255)	Encrypted password
birthdate	DATE	User’s date of birth
address	TEXT	Residential address

photo_url	VARCHAR(255)	Profile photo (optional)
created_at	DATETIME	When the user registered
status	ENUM(active, banned)	Account status

---

2. Wallets

Column Name	Data Type	Description
wallet_id (PK)	INT	Unique wallet ID
user_id (FK)	INT	References Users
balance	DECIMAL(12,2)	Current balance
currency	VARCHAR(10)	Currency type (INR, USD, etc.)
updated_at	DATETIME	Last update

---

3. Assets

Column Name	Data Type	Description
asset_id (PK)	INT	Unique asset ID
asset_type	ENUM(stock, mutual_fund, bond, crypto)	Type of asset
symbol	VARCHAR(20)	Ticker symbol (e.g., RELIANCE, BTC)
name	VARCHAR(100)	Asset full name
current_price	DECIMAL(12,2)	Market price
updated_at	DATETIME	Last price update

4. Portfolios

Column Name	Data Type	Description
portfolio_id (PK)	INT	Unique portfolio ID
user_id (FK)	INT	References Users
created_at	DATETIME	When portfolio was created
last_updated	DATETIME	Last modified

---

5. Portfolio_Holdings

Column Name	Data Type	Description
holding_id (PK)	INT	Unique holding ID
portfolio_id (FK)	INT	References Portfolios
asset_id (FK)	INT	References Assets
quantity	DECIMAL(12,4)	Number of units held
average_price	DECIMAL(12,2)	Average buy price per unit
created_at	DATETIME	When asset was added

6. Transactions

Column Name	Data Type	Description
transaction_id (PK)	INT	Unique transaction ID
user_id (FK)	INT	References Users
wallet_id (FK)	INT	References Wallets
asset_id (FK, nullable)	INT	References Assets if buy/sell
transaction_type	ENUM(buy, sell, deposit, withdraw)	Type
amount	DECIMAL(12,2)	Total amount of transaction
quantity (nullable)	DECIMAL(12,4)	Quantity of asset

status	ENUM(pending, completed, failed)	State
created_at	DATETIME	Timestamp

---

7. Payment_Methods

Column Name	Data Type	Description
method_id (PK)	INT	Unique payment method ID
user_id (FK)	INT	References Users
type	ENUM(card, UPI, bank)	Method type
masked_details	VARCHAR(100)	Last 4 digits, masked info
created_at	DATETIME	When added

---

8. OTP_Log

Column Name	Data Type	Description
otp_id (PK)	INT	Unique OTP ID
user_id (FK)	INT	References Users
otp_code	VARCHAR(255)	Hashed OTP
purpose	ENUM(login, buy, withdraw)	OTP use
expires_at	DATETIME	Expiry time
used	BOOLEAN	Whether OTP used

9. Achievements

Column Name	**Data Type**	**Description**
achievement_id (PK)	INT	Unique ID
title	VARCHAR(100)	Name of badge/achievement
description	TEXT	Details
points	INT	Points awarded

10. User_Achievements

Column Name	Data Type	Description
user_achievement_id (PK)	INT	Unique ID
user_id (FK)	INT	References Users
achievement_id (FK)	INT	References Achievements
earned_at	DATETIME	When earned

---

11. Leaderboard

Column Name	**Data Type**	**Description**
leaderboard_id (PK)	INT	Unique ID
user_id (FK)	INT	References Users
points_total	INT	Total points accumulated
rank	INT	Leaderboard position

---

12. Feedback

Column Name	Data Type	Description
feedback_id (PK)	INT	Unique feedback ID
user_id (FK)	INT	References Users
message	TEXT	Feedback message
status	ENUM(open, closed, in-progress)	Status
created_at	DATETIME	Timestamp

---

13. Admins

Column Name	Data Type	Description
admin_id (PK)	INT	Unique admin ID
name	VARCHAR(100)	Admin name
email	VARCHAR(150)	Unique email
password_hash	VARCHAR(255)	Encrypted password
role	ENUM(superadmin, employee)	Role
created_at	DATETIME	Added time

---

14. Admin_Audit_Log

Column Name	Data Type	Description
log_id (PK)	INT	Unique log ID
admin_id (FK)	INT	References Admins
action	VARCHAR(100)	Performed action
details	TEXT	Details

timestamp	DATETIME	When it happened

---

15. System_Financials

Column Name	Data Type	Description
record_id (PK)	INT	Unique record
total_client_funds	DECIMAL(14,2)	Total funds in system
last_updated	DATETIME	Last update

---

16. Client_Fund_Ledger

Column Name	Data Type	Description

ledger_id (PK)	INT	Unique ledger ID
user_id (FK)	INT	References Users
transaction_id (FK)	INT	References Transactions
entry_type	ENUM(credit, debit)	Fund flow
amount	DECIMAL(12,2)	Transaction amount
balance_after	DECIMAL(12,2)	User’s balance after txn
fund_account_balance	DECIMAL(14,2)	System-wide balance after txn
timestamp	DATETIME	When it occurred

Application Flow

User

1 — Open app / Landing

1. UI: user opens app or website → sees landing / login / signup buttons.
2. UI -> Server/Auth: Supabase Authentication services.

---

2 — Signup (email verification)

1. UI: user fills Name + Email + Password → taps **Sign up**.
2. UI -> Auth (Supabase): signUp(email, password) → creates auth user (status unverified).
3. Server/Auth -> DB: insert users row (status = unverified).
4. Server -> Email service: send verification link or code.
5. USER: clicks link → UI loads verification token → UI -> Auth verifies token.
6. Auth -> DB: set users.status = active.
7. UI: show success and prompt Login.

---

3 — Login

1. UI: user enters email + password and taps **Login**.
2. UI -> Auth: signIn(email, password)
3. Auth -> DB: verify credentials & users.status must be active.
4. On success: return session token; UI stores session (cookie / local storage).
5. On failure: show error and "forgot password".

---

4 — Dashboard (landing after login)

1. UI -> Server/API: call GET /dashboard with session token.
2. Server -> DB:
    - fetch wallets (balance),
    - fetch top portfolio summary (total value, P&L),

    - fetch achievements/leaderboard snippet.
3. Server -> MarketAPI: may request latest prices (or use cached price columns).
4. Server -> UI: render Dashboard.

---

5 — View Wallet & Transactions

1. UI: user clicks **Wallet**.
2. UI -> Server: GET /wallets/:user_id
3. Server -> DB: return wallet balance, client fund ledger, recent transactions.
4. UI: show ledger and deposit/withdraw buttons.

---

6 — Deposit (Add Funds) — Payment flow

1. UI: user chooses amount → taps **Deposit**.
2. UI -> Payment Gateway: opens checkout (card/UPI).
3. User completes payment inputs; gateway returns success or failure.
4. Gateway -> Server (webhook): notifies about payment result.
5. Server -> DB:
    - create transactions row type=deposit, status=pending → on webhook update to completed/failed,
    - if completed: update wallets.balance += amount and insert Client_Fund_Ledger entry,
    - update System_Financials if applicable.
6. Server -> Notif: email/notification receipt to user.
7. UI: poll or get realtime update (websocket / supabase realtime) to show updated balance.

Failure: payment failed → transaction marked failed, user notified.

---

7 — Buy Asset — Lumpsum (Stocks or Mutual Funds)

Precondition: user has enough wallet balance.

1. UI: user selects asset (stock or mutual fund) and chooses **Lumpsum** (enter amount or qty) → taps **Buy**.
2. UI -> Server: POST /transactions with asset_id, quantity or amount, transaction_type=buy.

3. Server -> MarketAPI: fetch latest price (stocks) or NAV (mutual funds).
4. Server -> DB:
    - check wallets.balance >= required_amount,
    - if insufficient → return fail to UI.
5. If sufficient:
    - Server -> DB: create transaction row (status = processing),
    - deduct wallet balance (or place reserve): wallets.balance -= amount,
    - insert Client_Fund_Ledger (debit),
    - update portfolio_holdings: if holding exists, recalc avg_price and quantity, else insert new row,
    - update System_Financials if needed.
6. Server -> DB: set transaction status = completed.
7. Server -> Notif: send buy confirmation.
8. UI: show new holdings and updated wallet.

---

8 — Buy Asset — SIP (if supported)

1. UI: user selects **SIP**, sets amount & frequency.
2. UI -> Server: POST /sip_plans (schedule).
3. Server: store SIP schedule and create job (cron / serverless scheduler).
4. On each SIP date:
    - run scheduled job → fetch NAV → perform buy flow (similar to lumpsum) subject to balance check.
5. Notify user after each execution.

---

9 — View Portfolio, Holdings & P&L (Profit & Loss)

1. UI: user clicks **Portfolio**.
2. UI -> Server: GET /portfolio/:user_id
3. Server -> DB: query portfolio_holdings for user and assets current_price.
4. Server: calculate per-asset:
    - market_value = quantity * current_price,
    - unrealized_P&L = market_value - (quantity * avg_price).
    - total portfolio value = sum of market_value.
5. Server -> UI: send holdings, market values, P&L, charts.
6. UI: show gains/losses, totals, and help tips.

---

10 — View Activity / Transaction History

1. UI -> Server: GET /transactions?user_id=...&limit=...
2. Server -> DB: return list of transactions with asset names (join assets).
3. UI: show timeline with filters (date/type/status).

---

11 — Achievements & Leaderboard

1. Server detects events (first buy, milestone portfolio growth) → award achievements and increment leaderboard.
2. DB: insert into User_Achievements and update Leaderboard.points_total.
3. UI -> Server: GET /leaderboard → show ranks.

---

12 — Logout

1. UI: user logs out → clear local session and call Auth.signOut().

Admin

1 — Admin Login & Auth

1. Admin UI: enters admin credentials.
2. Admin UI -> Auth: signInAdmin(email, password) (server validates role).
3. Server -> DB: check admins table and role.
4. On success: create admin session. On fail: deny.

---

2 — Admin Dashboard (monitoring)

1. Admin UI -> Server: request dashboard data.
2. Server -> DB: query users, transactions (pending/flagged), system_financials, admin_audit_log (recent).
3. Server -> UI: return data and visual charts.

---

3 — View & Inspect User

1. Admin selects a user → Admin UI -> Server: GET /admin/users/:id.
2. Server -> DB: fetch users, wallets, portfolio_holdings, transactions, feedback.
3. Admin views profile, KYC flags, recent activities.

---

4 — Ban / Suspend / Unban User

1. Admin clicks **Ban user**.
2. UI -> Server: POST /admin/users/:id/ban (with reason).
3. Server -> DB: update users.status = banned (and optionally block auth).
4. Server -> DB: insert Admin_Audit_Log(action='ban', admin_id, details).
5. Server -> Notif: send suspension email.
6. UI: show success.

Rollback: Admin can unban → update status and log.

---

5 — Review / Flag Transactions

1. Admin views flagged or large transactions → inspects payment IDs and ledgers.
2. Admin may mark transaction as in_review, resolved, reconciled, or escalate to finance.
3. DB: update transactions.status and add audit log entry.

---

6 — Manage Assets & Prices

1. Admin can add/update assets (symbol, name, asset_type) in assets table.
2. For manual price updates: Admin updates assets.current_price (logged). Automatic price source is preferred via MarketAPI.

---

7 — Reconcile System Financials

1. Admin runs reconciliation:
    - compare sum(wallet balances) with Client_Fund_Ledger + System_Financials.
2. If mismatch → investigate specific transactions, mark adjustments, log audit.

---

8 — View Audit Logs

1. Admin can view Admin_Audit_Log for traceability of actions (who banned, who changed price, etc.).
2. Logs are immutable (append-only) for compliance.

---

9 — Reports & Exports

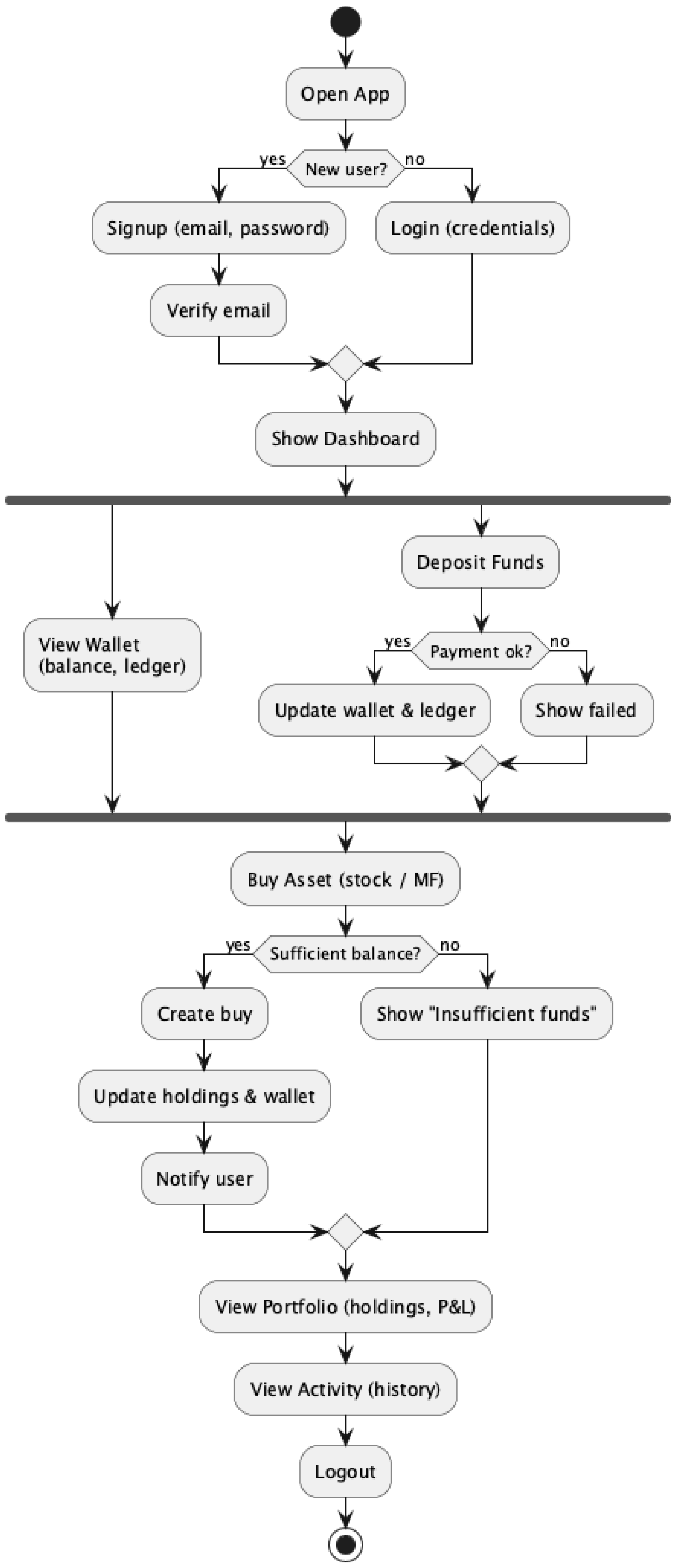
1. Admin can export CSV reports of transactions, user lists, or ledgers for accounting.

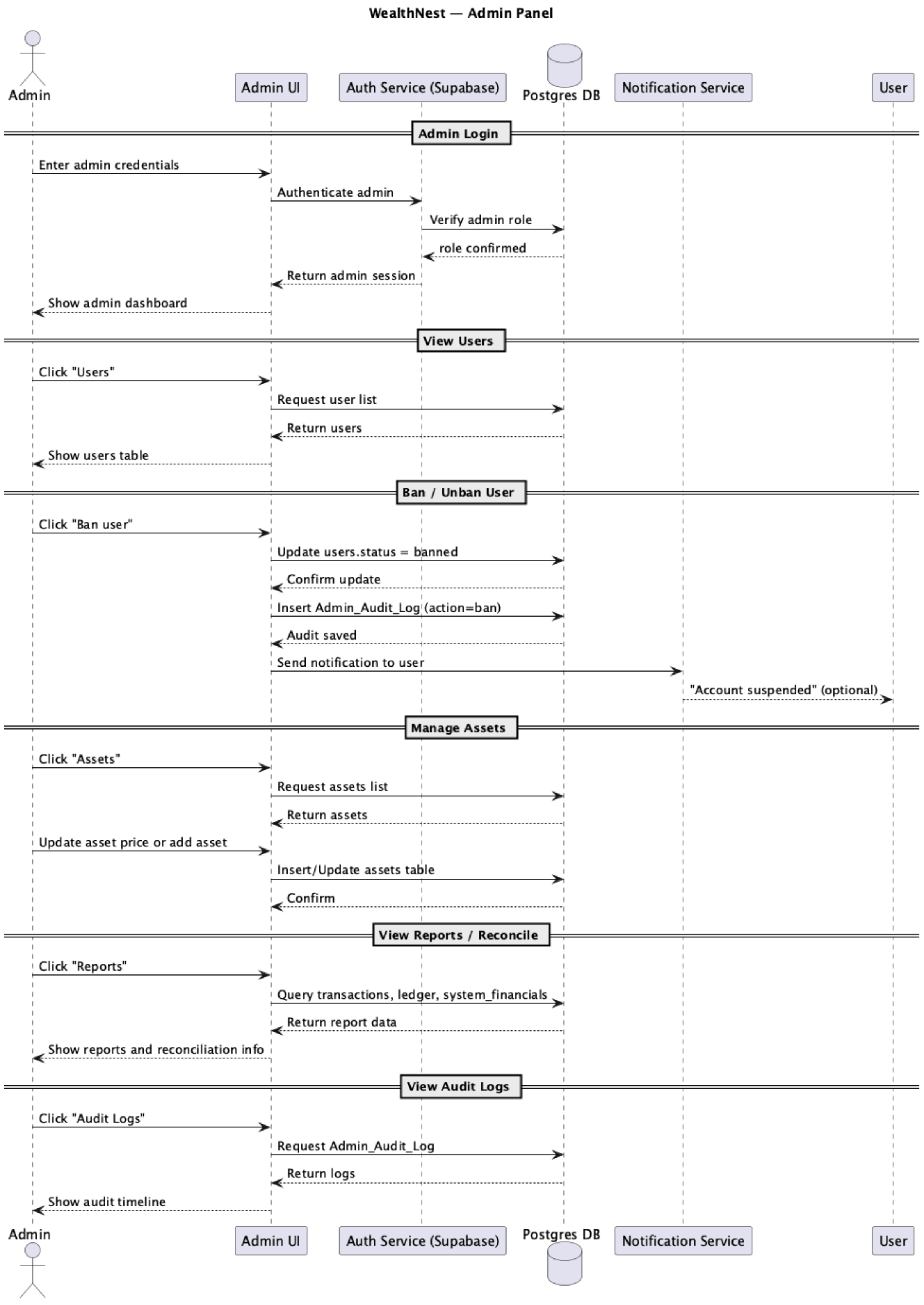
---

10 — Admin Logout

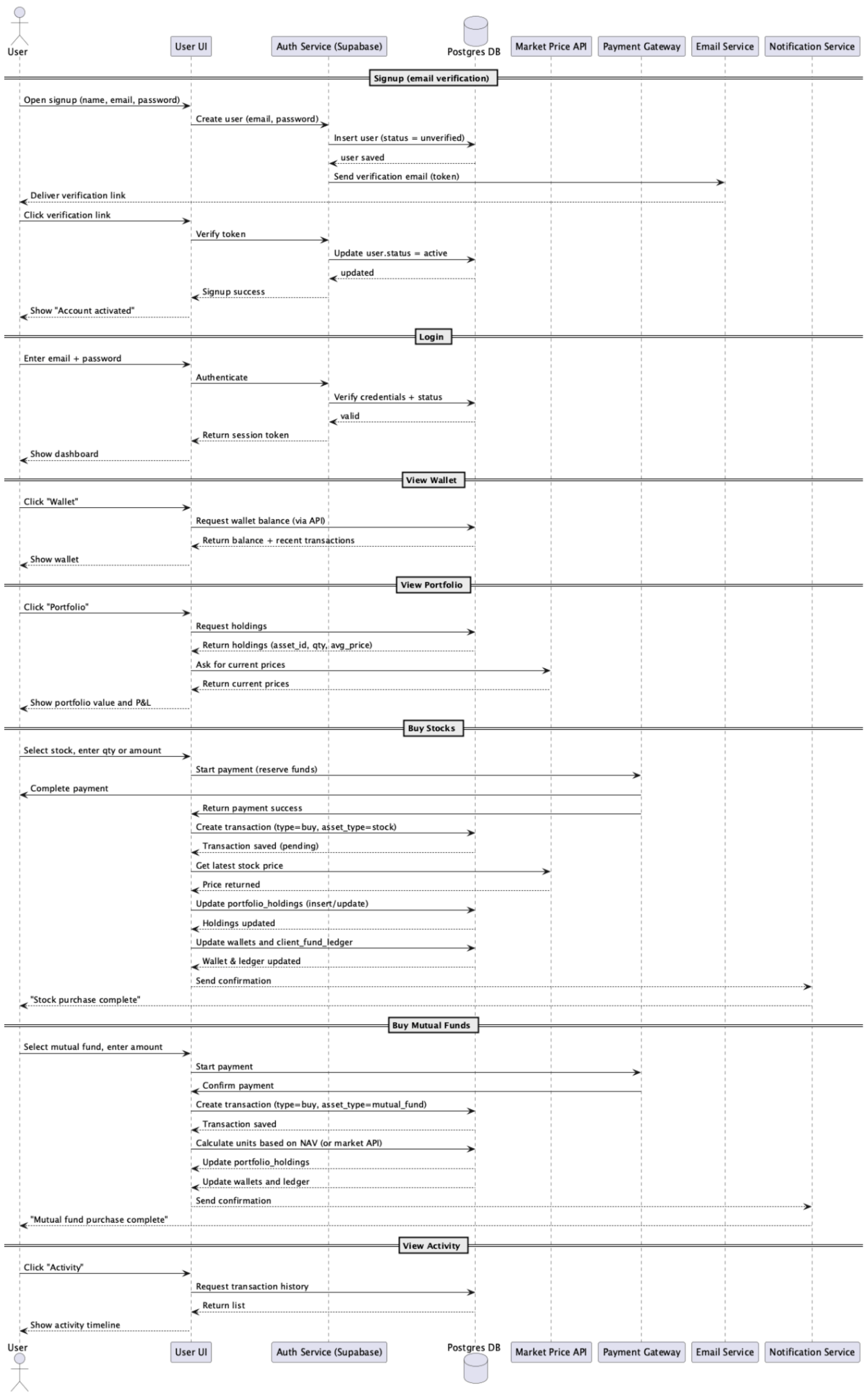
1. Admin signs out; server invalidates admin session tokens.

WealthNest — User Activity Flow




WealthNest — Admin Panel
Admin
Admin UI
Auth Service (Supabase)
Postgres DB
Notification Service
User
Admin Login
Enter admin credentials
Authenticate admin
Verify admin role
role confirmed
Return admin session
Show admin dashboard
View Users
Click "Users"
Request user list
Return users
Show users table
Ban / Unban User
Click "Ban user"
Update users.status = banned
Confirm update
Insert Admin_Audit_Log (action=ban)
Audit saved
Send notification to user
"Account suspended" (optional)
Manage Assets
Click "Assets"
Request assets list
Return assets
Update asset price or add asset
Insert/Update assets table
Confirm
View Reports / Reconcile
Click "Reports"
Query transactions, ledger, system_financials
Return report data
Show reports and reconciliation info
View Audit Logs
Click "Audit Logs"
Request Admin_Audit_Log
Return logs
Show audit timeline

WealthNest — User Panel


User
User UI
Auth Service (Supabase)
Postgres DB
Market Price API
Payment Gateway
Email Service
Notification Service
Signup (email verification)
Open signup (name, email, password)
Create user (email, password)
Insert user (status = unverified)
user saved
Send verification email (token)
Deliver verification link
Click verification link
Verify token
Update user.status = active
updated
Signup success
Show "Account activated"
Login
Enter email + password
Authenticate
Verify credentials + status
valid
Return session token
Show dashboard
View Wallet
Click "Wallet"
Request wallet balance (via API)
Return balance + recent transactions
Show wallet
View Portfolio
Click "Portfolio"
Request holdings
Return holdings (asset_id, qty, avg_price)
Ask for current prices
Return current prices
Show portfolio value and P&L
Buy Stocks
Select stock, enter qty or amount
Start payment (reserve funds)
Complete payment
Return payment success
Create transaction (type=buy, asset_type=stock)
Transaction saved (pending)
Get latest stock price
Price returned
Update portfolio_holdings (insert/update)
Holdings updated
Update wallets and client_fund_ledger
Wallet & ledger updated
Send confirmation
"Stock purchase complete"
Buy Mutual Funds
Select mutual fund, enter amount
Start payment
Confirm payment
Create transaction (type=buy, asset_type=mutual_fund)
Transaction saved
Calculate units based on NAV (or market API)
Update portfolio_holdings
Update wallets and ledger
Send confirmation
"Mutual fund purchase complete"
View Activity
Click "Activity"
Request transaction history
Return list
Show activity timeline
User
User UI
Auth Service (Supabase)
Postgres DB
Market Price API
Payment Gateway
Email Service
Notification Service