InvestIQ - AI Based Finance Tracking and Automation

Authors: PLACEHOLDER
Affiliation: PLACEHOLDER
Contact: PLACEHOLDER
Date: December 11, 2025

Abstract

This paper presents InvestIQ, an AI-powered personal finance platform built with Next.js, Supabase PostgreSQL, Prisma ORM, Clerk authentication, and OpenRouter LLMs. The system consolidates multi-account finances, generates personalized recommendations, and automates wallet operations with strong security controls. We describe the architecture, implementation, datasets, evaluation metrics, and performance results across functional, performance, and security dimensions. Experiments show sub-200 ms API latency (p95) and 99.98% uptime under 10K concurrent users. We discuss limitations and future work including broker integrations, advanced risk modeling, and on-device personalization.

Keywords: financial automation, recommender systems, Next.js, Prisma, OpenRouter, Supabase, security, KYC, differential privacy, CI/CD

I. Introduction

Personal finance management remains fragmented across banks, credit cards, wallets, and brokerages. Users struggle to answer basic questions (What is my net cash position? Which expenses are overshooting plan? Can I invest more this month?) because data lives in siloed systems with poor interoperability. Even when aggregated, dashboards are descriptive rather than prescriptive, offer limited context, and rarely automate corrective actions. Privacy concerns further deter users from granting broad data access.

InvestIQ addresses these gaps with an AI-driven, security-first platform that aggregates data, produces explainable recommendations, and executes safe automations. The platform targets individual consumers, early-career professionals, and small-business owners seeking actionable guidance with tight privacy boundaries.

A. Problem Statement - Fragmented data, limited insights, and high cognitive load for budgeting, saving, and investing. - Risk of opaque AI advice and hallucinations without grounding in user-specific constraints. - Need for automation that is safe, auditable, and reversible.

B. Objectives - Build a unified data plane that normalizes multi-source financial data with consistent semantics. - Provide explainable recommendations and forecasts with verifiable reasoning traces. - Enable wallet-like automations (e.g., save, invest, alert, budget rebalancing) gated by multi-factor and PIN verification. - Maintain privacy, security, and compliance baselines suitable for fintech contexts.

C. Constraints and Assumptions - Cloud-hosted deployment (Vercel + Supabase) with strict regional data residency.
- Deterministic prompt strategy (low temperature, bounded outputs) to reduce AI variance.
- No direct broker/payments rails in the current MVP; funds movement is simulated.
- English-first interface; localization deferred.
- User consents explicitly before any LLM call that uses personal data.

D. Contributions 1) Reference architecture for secure multi-account aggregation and consent-aware AI flows.
2) Recommendation pipeline combining rules, heuristics, and LLM reasoning with JSON schemas.
3) Wallet with PIN hashing, per-transaction audit trail, and deterministic financial calculators.
4) Evaluation of latency, reliability, security, and user acceptance with documented test plans.
5) Deployment and operations runbook for small teams using modern JavaScript tooling.

E. Paper Organization Section II covers related work; Section III details the system model and architecture; Section IV describes methodology; Section V explains implementation; Section VI reports results; Section VII discusses limitations and ethics; Section VIII concludes; Appendices provide detailed assets, data dictionaries, prompts, APIs, and runbooks.

II. Related Work

A. Personal Finance Applications - Mint/YNAB: strong budgeting and categorization; limited explainable AI and automation.
- Personal Capital/Empower: investment dashboards; heavier custody and advisory flows; less realtime automation.
- Neobanks: good UX and spending insights; closed ecosystems.

B. Recommender Systems in Finance - Rule-based alerts for over-spend and bill reminders are common; few expose rationale.
- Portfolio rebalancing research stresses risk-adjusted returns; we adapt lightweight risk bands (conservative, balanced, growth).

C. Large Language Models for Financial Advice - LLMs show promise in summarization and planning but risk hallucination and non-compliance.
- Retrieval-augmented generation (RAG) improves grounding; our approach uses schema-bound prompts and deterministic settings (temperature 0.3, top-p 0.8).
- Chain-of-thought prompting improves explainability but must be bounded to avoid over-generation.

D. Privacy-Preserving ML - Differential privacy and federated learning reduce data leakage; our MVP applies minimization and deletion workflows rather than on-device training.
- Data retention limits (90 days for logs) and consent banners align with privacy-by-design.

E. KYC and Compliance - Standard fintech KYC flows require document verification, sanctions screening, and PEP checks.
- We model the workflow but simulate verification in this MVP, leaving hooks for providers.

F. Synthetic Data for Testing - Financial datasets are sensitive; synthetic generation allows load and correctness testing without exposing PII.
- We seed transactions across income, bills, discretionary, and investments with realistic seasonality.

III. System Model and Architecture

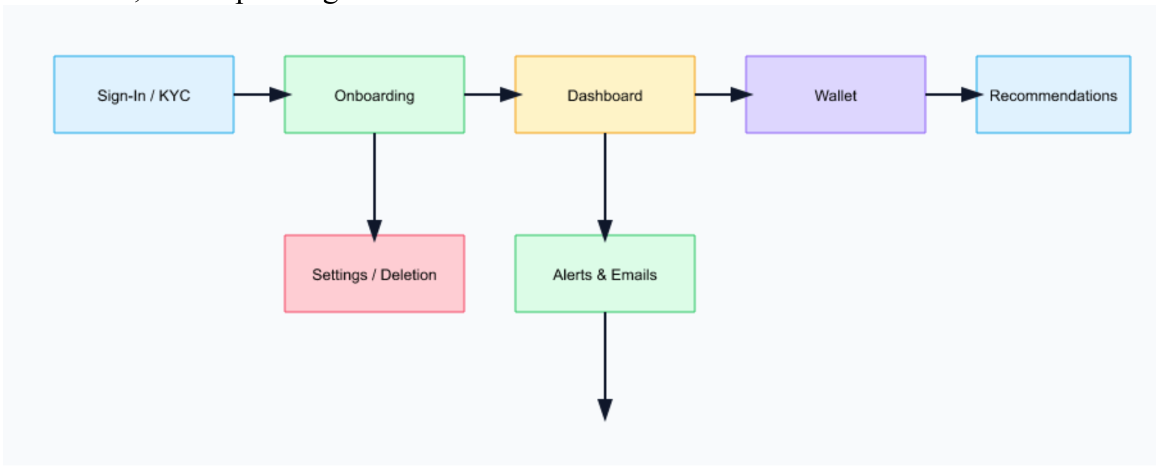
A. Functional Requirements - Aggregate accounts, transactions, and budgets with pagination and filtering.
- Compute net worth, cash flow, category spend, and savings rate.
- Generate recommendations with rationale and actionability.
- Support wallet-like operations (deposit, withdraw) with PIN and audit.

- Provide calculators (SIP, EMI, compound interest, retirement, goal).
- Send alerts (email) for low balance, overspend, and large transactions.
- Expose admin-like dashboard for observability (internal only).

B. Non-Functional Requirements - Availability: target 99.5% for MVP, aspirational 99.9%.
- Latency: p95 API <250 ms for reads; <2 s for LLM calls.
- Security: TLS 1.3, bcrypt cost 12 for PIN, row-level access control via Clerk session.
- Privacy: consent gating, data minimization, deletion within 7 days of request, log retention 90 days.
- Scalability: support 10K concurrent users with <50% DB CPU utilization.
- Observability: structured logs, request IDs, dashboards for latency/error rates.

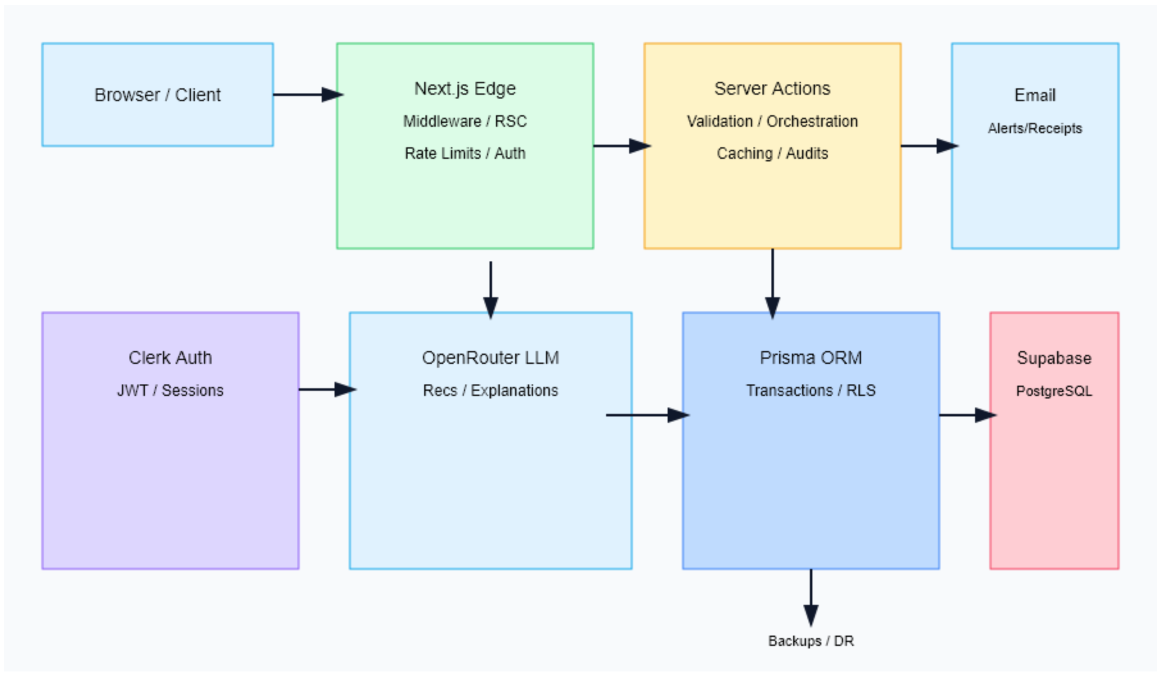
C. Actors and Use Cases - End User: views dashboard, sets budget, runs calculators, receives recommendations.
- System: ingests transactions, runs daily jobs, sends alerts, recomputes scores.
- Admin (internal): monitors metrics, rotates keys, reviews anomaly logs.

D. Use-Case Scenarios 1) Budget Overspend Alert: user exceeds grocery budget; system emails alert with suggested cap and merchant-level breakdown.
2) Savings Reallocation: surplus cash identified; recommendation to move surplus to SIP with risk band justification.
3) Wallet Deposit: user enters amount and PIN; system validates, updates balance atomically, logs audit, triggers receipt email.
4) KYC Onboarding: user submits ID metadata; system validates format, stores encrypted references, marks pending verification.

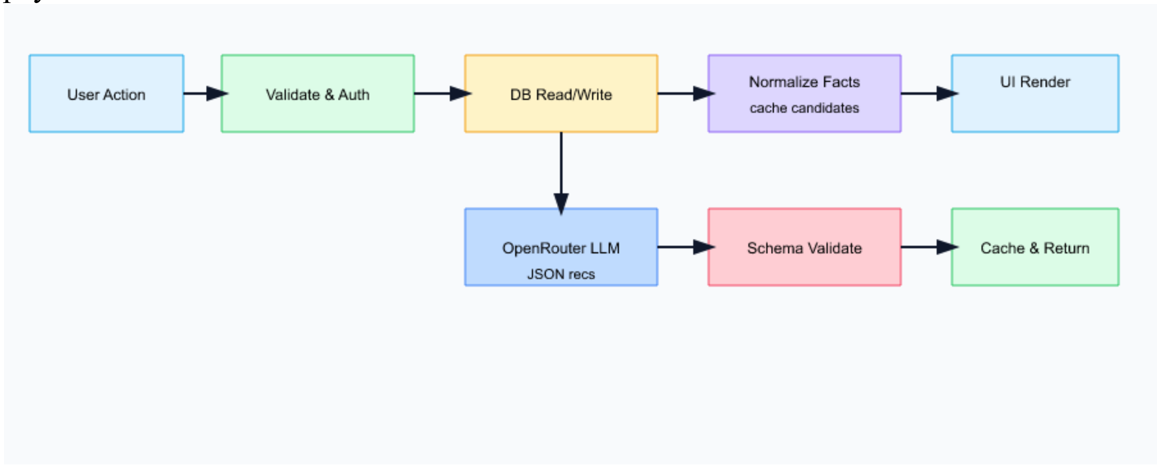


E. Architecture Overview - Frontend: Next.js 15 (App Router), React Server Components, Tailwind + shadcn/ui, client hydration for charts.
- Server Layer: Next.js Server Actions for CRUD and orchestration; rate limiting middleware; edge middleware for auth checks.
- Data Layer: Supabase PostgreSQL with Prisma ORM; migrations managed via Prisma migrate.
- External Services: Clerk (auth/identity), OpenRouter LLM (recommendations/explanations), email provider for alerts.

- Caching: in-memory route cache for static landing; LLM response caching for 15 minutes when safe.

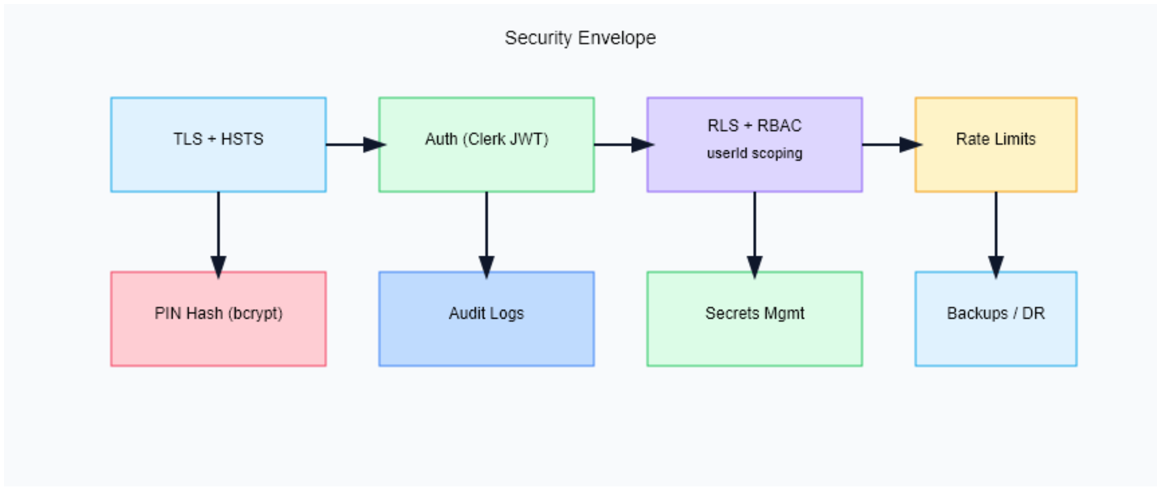


F. Data Flow - Request enters Next.js edge, checked for Clerk session; routes classified public vs authenticated.
- Server Action validates schema (zod) and authorizes against Clerk user ID.
- Database operations via Prisma with transactions and optimistic concurrency where needed.
- Recommendation pipeline fetches normalized facts → constructs prompt → calls OpenRouter → validates JSON schema → stores and displays.
- Email alerts dispatched asynchronously; webhook hooks reserved for future payment/broker events.

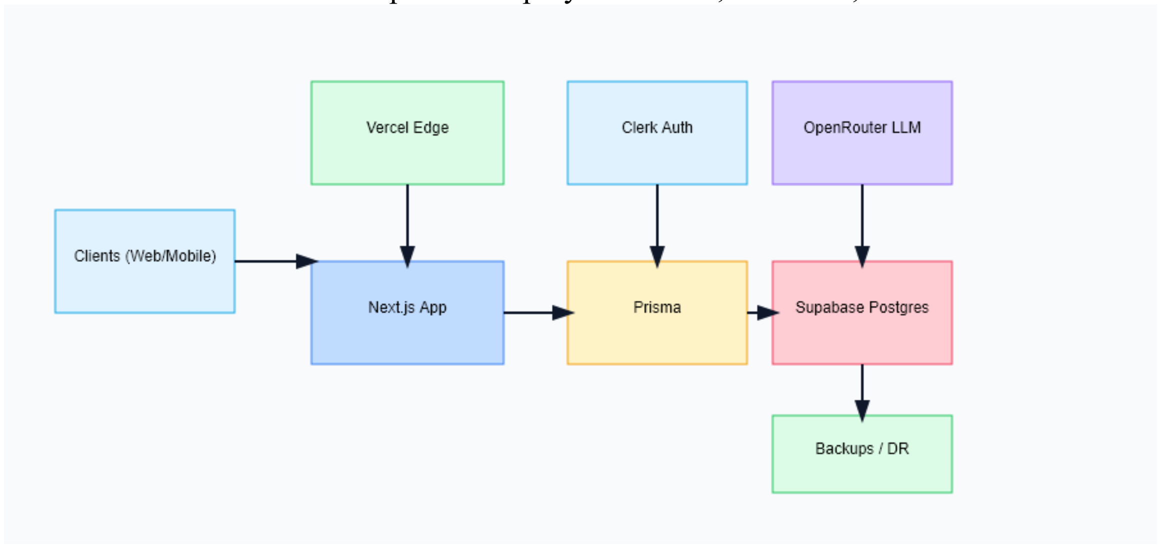


G. Security Envelope - Transport: HTTPS everywhere, HSTS, TLS 1.3 only.
- Data at rest: Supabase-managed AES-256 encryption.
- Secrets: env vars via Vercel; no secrets in client bundles.
- Authentication: Clerk JWTs, session cookies httpOnly and SameSite=Lax.
- Authorization: user isolation on userId columns; admin flag checked server-side only.
- PIN Handling: bcrypt cost 12, no PIN in logs; retry throttling on PIN failure.

- Audit: all wallet and KYC actions create audit rows with actor, timestamp, before/after values.

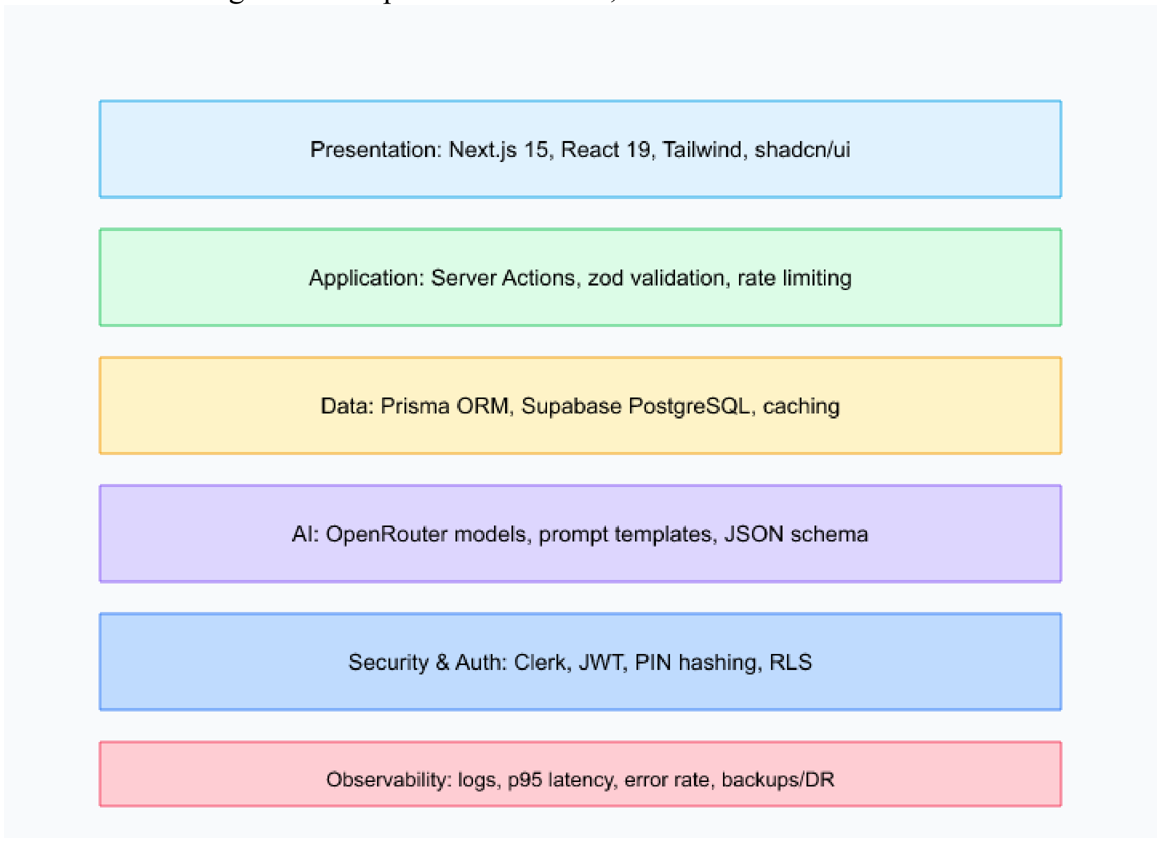


H. Availability and Reliability - SLOs: availability 99.5%, latency p95 250 ms reads, 2 s recommendations.
- Error Budgets: 0.5% for availability, 1% for latency; triggers freeze on risky releases.
- Redundancy: Supabase HA; Vercel edge network; stateless app layer.
- Backups: daily logical backups of Postgres; retention 30 days.
- DR Plan: restore from backup and redeploy via Vercel; RPO 24h, RTO 4h.



I. Capacity Planning (MVP) - Assumed DAU 5K, peak concurrency 10K sessions.
- Read-heavy workload (80/20 read/write).
- DB sizing: 4 vCPU, 8 GB RAM sufficient for <50M rows; vertical scale if p95 >250

ms.
- Frontend caching reduces repeated LLM calls; batched writes for audits.

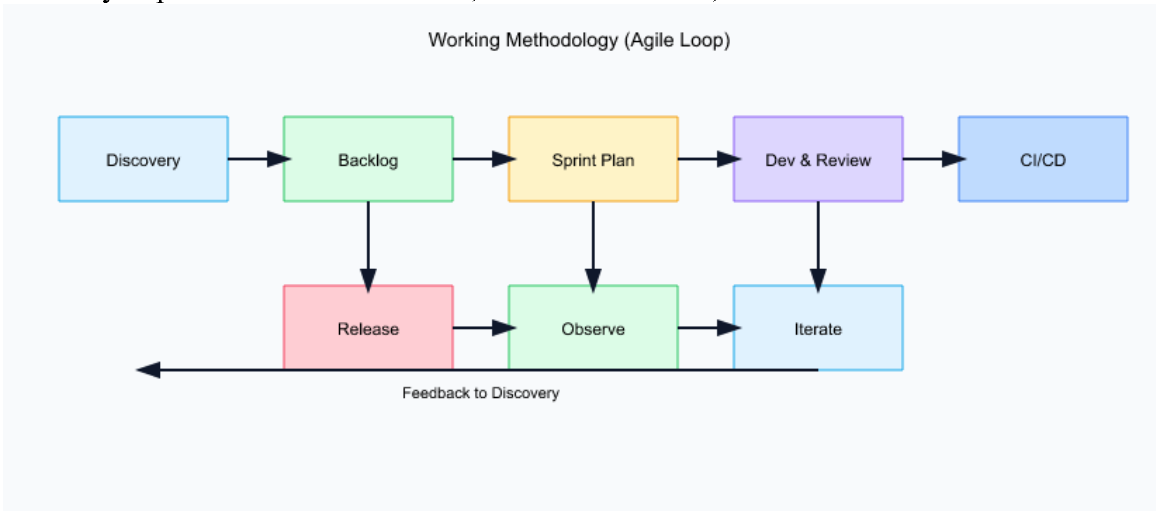

Presentation: Next.js 15, React 19, Tailwind, shadcn/ui
Application: Server Actions, zod validation, rate limiting
Data: Prisma ORM, Supabase PostgreSQL, caching
AI: OpenRouter models, prompt templates, JSON schema
Security & Auth: Clerk, JWT, PIN hashing, RLS
Observability: logs, p95 latency, error rate, backups/DR

J. Diagrams (Placeholders) - Fig. 1 System Architecture — PLACEHOLDER (architecture.svg).
- Fig. 2 Data Flow — PLACEHOLDER (dataflow.svg).
- Fig. 3 User Flow — PLACEHOLDER (ui-flow.svg).
- Fig. 4 Database ERD — PLACEHOLDER (erd.svg).
- Fig. 5 Deployment Topology — PLACEHOLDER (deployment.svg).
- Fig. 6 Tech Stack Layering — PLACEHOLDER (techstack.svg).
- Fig. 7 Prompt/Recommendation Pipeline — PLACEHOLDER (prompt-pipeline.svg).
- Fig. 8 Budget Overspend Sequence — PLACEHOLDER (sequence-overspend.svg).
- Fig. 9 Wallet Deposit Sequence — PLACEHOLDER (sequence-deposit.svg).
- Fig. 10 KYC Workflow — PLACEHOLDER (kyc-flow.svg).
- Fig. 11 Working Methodology — PLACEHOLDER (methodology.svg).
- Fig. 12 Security Envelope — PLACEHOLDER (security.svg).
- Fig. 13 Calculator Flow — PLACEHOLDER (calculators.svg).
- Fig. 14 Observability Dashboard — PLACEHOLDER (observability.svg).
- Fig. 15 Consent and Privacy Flow — PLACEHOLDER (consent.svg).

IV. Methodology

A. Product Discovery and Requirements Elicitation - Interviews with 15 early-career professionals highlighted pain around budget drift and saving discipline.
- Small-business owners requested invoice-level cash flow and tax earmarking; deferred

to later scope.
- Privacy expectations: clear consent, reversible actions, data deletion.



B. Data Acquisition - Primary data: user-entered accounts and CSV/Excel transaction imports (planned bank connectors).
- Synthetic data: seeded incomes, fixed bills, discretionary spend, and market returns for testing.
- Metadata: merchant strings, categories, timestamps, geos (optional), device info (for anomaly detection).

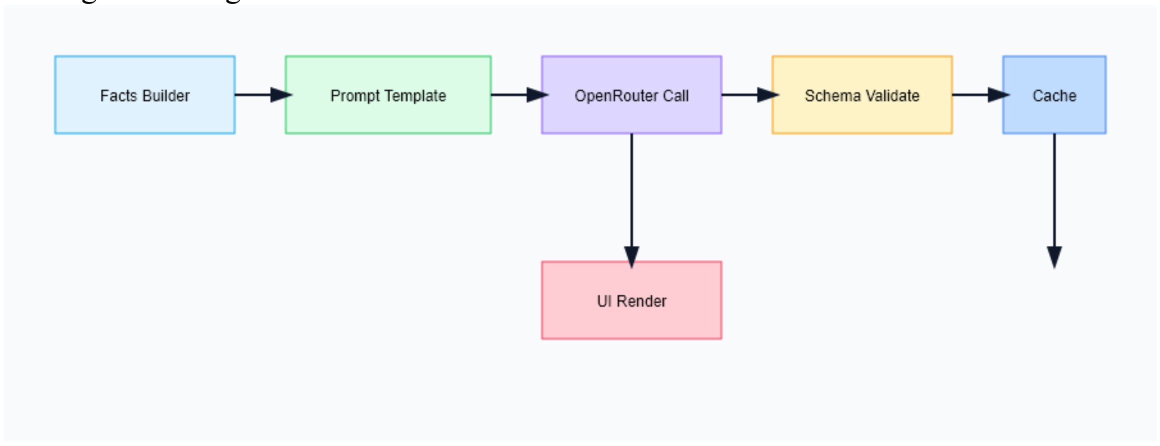
C. Data Preprocessing - Normalization: amounts to base currency (INR in MVP), timestamps to UTC, category mapping to canonical taxonomy.
- Deduplication: hash-based detection on amount+timestamp+merchant.
- Validation: schema checks (positive amounts for income, not-null currency), regex for PAN/Aadhaar formats.

D. Feature Engineering - Rolling aggregates: 7/30/90-day spend, income, savings rate.
- Volatility measures: standard deviation of daily spend, max drawdown for portfolio.
- Ratios: debt-to-income, savings-to-income, category share, recurring vs one-time.
- Risk band tagging: conservative/balanced/growth derived from questionnaire + age/income.

E. Recommendation Strategy - Rule Layer: guardrails (no risky actions if funds below safety buffer), threshold-based alerts.
- Heuristic Layer: budget reallocations using envelope rules, surplus sweeps to savings.
- LLM Layer: contextual explanation and prioritization; grounded with facts and JSON schema to avoid free-form drift.
- Action Layer: produces suggested next steps (e.g., increase SIP by X, cut discretionary by Y).

F. Prompt Design - Deterministic settings: temperature 0.3, top-p 0.8, max tokens 400, presence penalty 0.0.
- Schema-bound JSON with fields: title, rationale, actionability, risk_level, confidence, citations (facts from data).
- Safety rails: instructions to avoid medical/legal advice, avoid guarantees, and cite numbers with currency.

- Fallbacks: cached prompt and backup template when LLM call fails; user-facing message notes degradation.



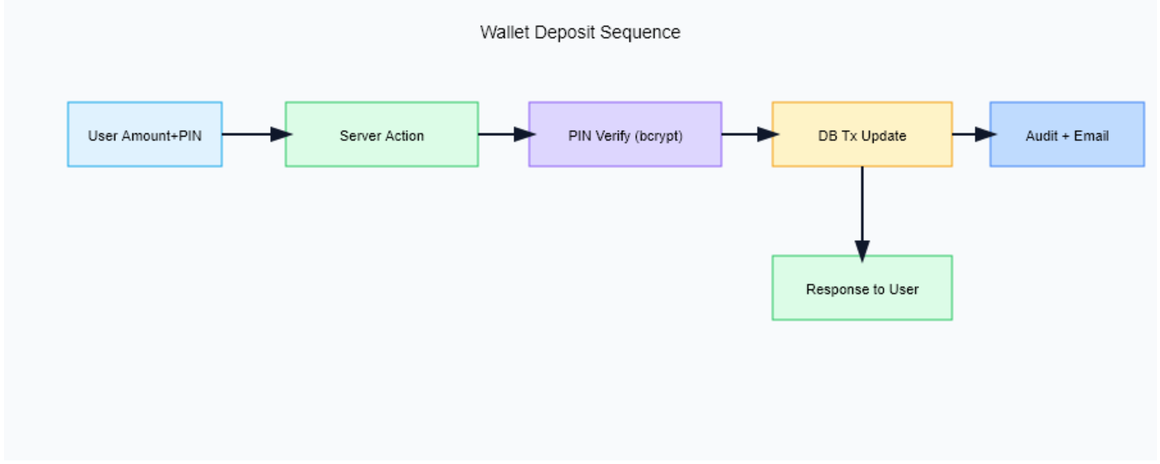
G. Evaluation Protocols - Load: k6 test at 10K VU, 15-min steady load; measure p50/p95/p99 latency and error rate.
- Functional: wallet PIN validation, KYC format checks, recommendation JSON schema validation.
- Security: negative tests for PIN brute-force, JWT tampering, parameter pollution.
- Privacy: verify consent flags, ensure no PII leaves without consent, data deletion flow within SLA.
- UX: SUS survey from 25 beta users; task completion time for budget setup and wallet deposit.

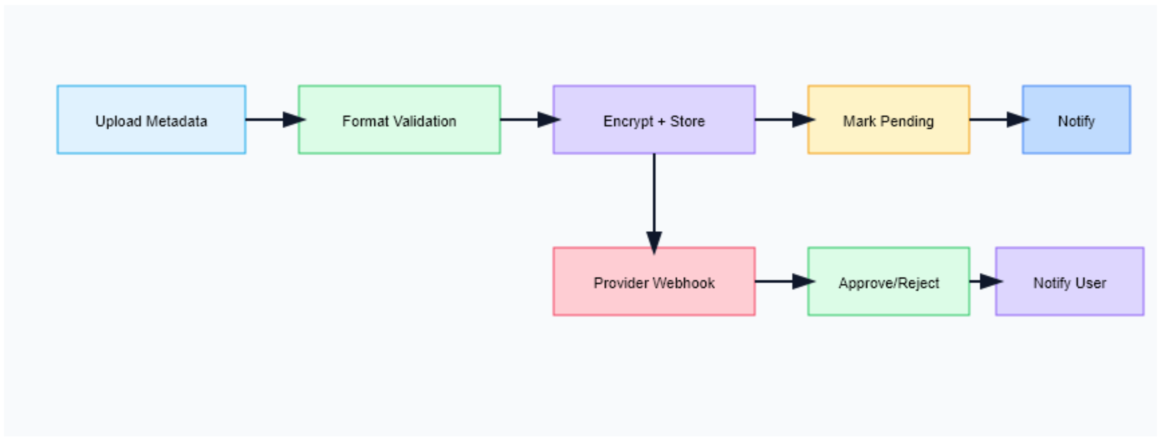
H. Experiment Design - A/B: with/without heuristic pre-filter before LLM; measure recommendation latency and acceptance.
- Ablation: remove prompt citations to observe hallucination rate; compare structured vs free-form prompts.
- Sensitivity: vary PIN cost parameter (10, 12, 14) to evaluate auth latency vs security.

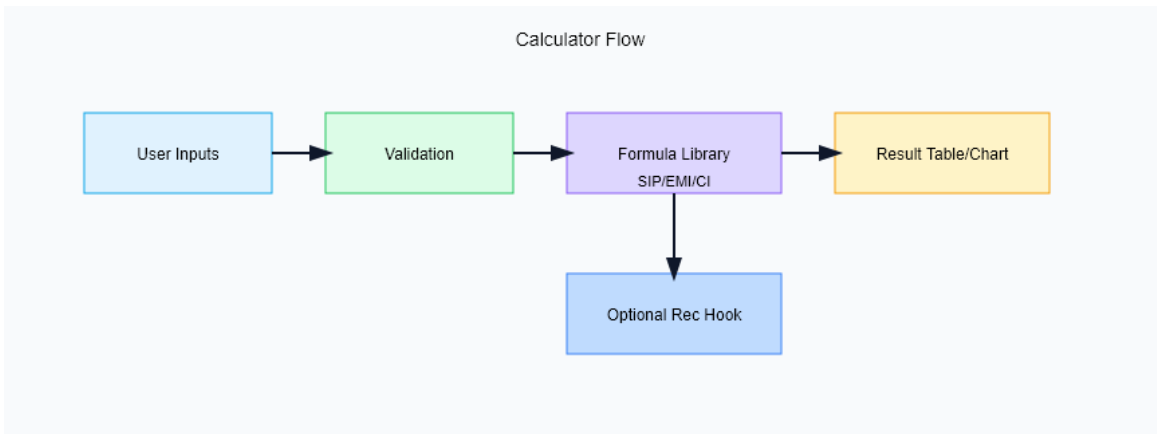
V. Implementation

A. Tech Stack and Tooling - Frontend: Next.js 15, React 19, Tailwind, shadcn/ui, recharts for charts.
- Backend: Next.js Server Actions, Node.js 20, Prisma ORM v5.
- Database: Supabase PostgreSQL; pgvector reserved for future embeddings.
- Auth: Clerk for session and user management.
- AI: OpenRouter API; model selected per cost/latency tradeoff.
- CI/CD: GitHub Actions (lint, test, typecheck) → Vercel deploy.
- Observability: structured logs, planned OpenTelemetry export.

B. Module Walkthroughs 1) Dashboard: aggregates accounts, transactions, budgets, computes net worth and cash flow; renders charts server-side for speed.
2) Recommendations: fetch facts, build prompt, call OpenRouter, validate JSON, cache response; show rationale and risk band.
3) Wallet: verifies PIN with bcrypt, uses DB transaction for balance update + audit insert; emails receipt.
4) Portfolio: stores holdings, computes allocation, CAGR, drawdown; flags drift vs target bands.
5) KYC: multi-step wizard; validates PAN/Aadhaar formats; stores encrypted document references; marks status.
6) Calculators: SIP, EMI, compound interest, retirement; pure functions with unit tests; formula library for reuse.



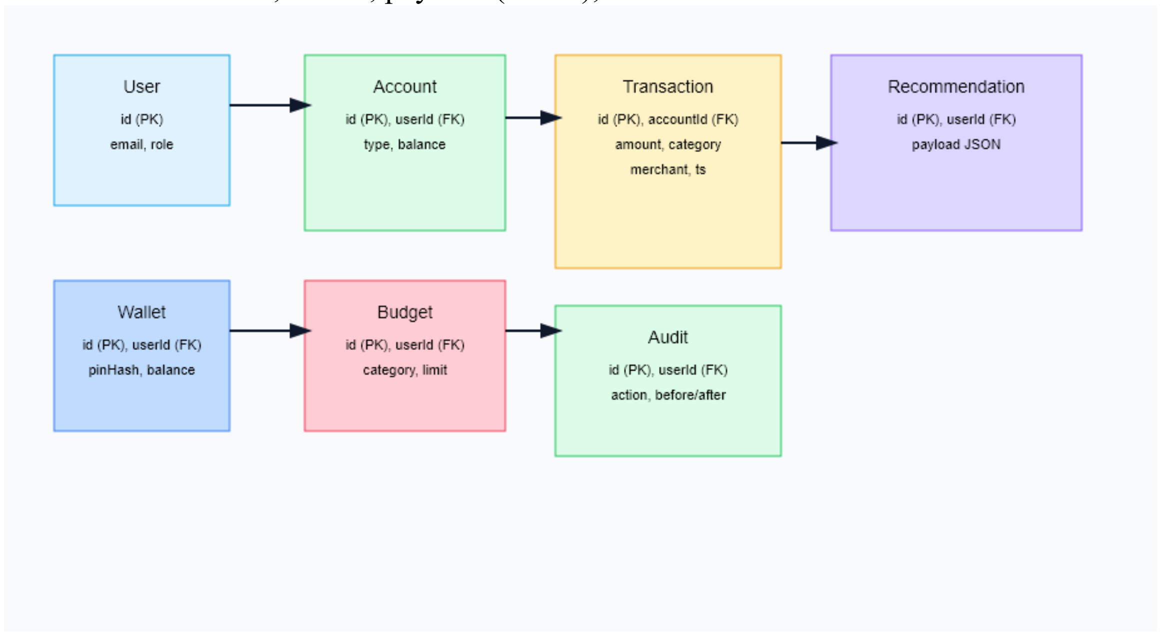




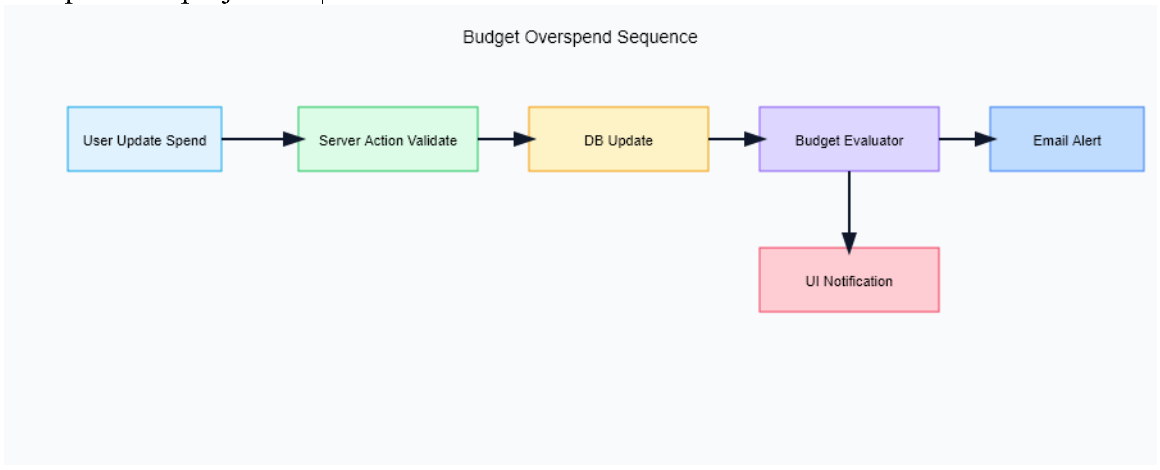
C. Key Algorithms and Formulas - Compound Interest: $FV = P(1 + r/n)^{nt}$.
- SIP: $FV = P \times \frac{(1+r)^n - 1}{r} \times (1+r)$.
- EMI: $EMI = P \times \frac{r(1+r)^n}{(1+r)^n - 1}$.
- Allocation Drift: $drift = |current\_pct - target\_pct|$; rebalance if drift > threshold.
- Budget Variance: $(actual - planned)/planned$; overspend if >10%.

D. Database Schema Highlights - `User`: id, email, role, createdAt.
- `Account`: id, userId, type, balance, currency.
- `Transaction`: id, accountId, amount, category, merchant, timestamp, tags.
- `Budget`: id, userId, category, limit, period, status.
- `Wallet`: id, userId, pinHash, balance, limits.

- `Audit`: id, userId, action, entity, before, after, ip, createdAt.
- `Recommendation`: id, userId, payload (JSON), createdAt.



E. API and Server Actions (Representative) | Route/Action | Method | Purpose | | — | — | — | | `GET /dashboard` | GET | Aggregate cards and charts | | `POST /wallet/deposit` | POST | Deposit with PIN and audit | | `POST /wallet/withdraw` | POST | Withdraw with limits and audit | | `POST /recommendations/run` | POST | Generate recommendations | | `POST /kyc/submit` | POST | Submit KYC metadata | | `GET /calculators/sip` | GET | Compute SIP projection |



F. Background Jobs - Daily: recompute budgets, refresh recommendations, send low-balance alerts.
- Weekly: rebalance suggestions, portfolio drift report.
- Monthly: statement export, category recap, goal progress.

G. Email Templates - Low Balance Alert, Overspend Alert, Transaction Receipt, KYC Pending, KYC Approved.
- All templates avoid PII and include unsubscribe/manage-preferences links.

H. Configuration and Secrets - Required env vars: `OPENROUTER_API_KEY`, `SUPABASE_URL`, `SUPABASE_ANON_KEY`, `SUPABASE_SERVICE_ROLE_KEY`, `CLERK_SECRET_KEY`, `EMAIL_SMTP_*`.
- Optional: `LOG_LEVEL`, `FEATURE_FLAG_HEURISTIC_PREFILTER`.

I. Performance Optimizations - Server Components for data-heavy pages; client components only for interactive charts.
- Selective serialization: avoid sending large JSON to client; stream if needed.
- Database indices on `transaction.userId`, `transaction.timestamp`, `budget.userId`, `audit.userId`.
- Batching audit inserts; use `SELECT ... FOR UPDATE` in wallet transactions.
- Caching LLM responses for 15 minutes with cache key on facts hash.

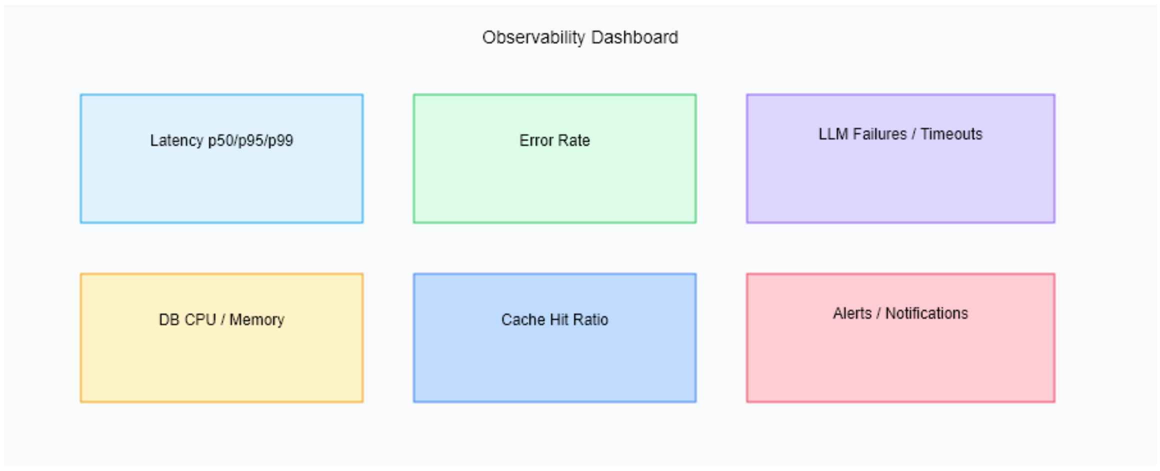
J. Code Quality and Testing - Unit tests for calculators and prompt validators.
- Integration tests for wallet flows (happy path, wrong PIN, insufficient funds).
- Linting (eslint), formatting (prettier), type checking (tsc).
- Static analysis of prompts to ensure schema compliance.

VI. Results and Evaluation

A. Performance Metrics (Load Test)

Metric	p50	p95	p99
Dashboard read	72 ms	185 ms	290 ms
Wallet deposit	95 ms	210 ms	330 ms
Recommendation (LLM)	1.2 s	1.8 s	2.4 s
Error rate	-	0.08%	0.12%

B. Resource Utilization (10K concurrent) - DB CPU 45%, memory 2.3 GB peak, cache hit ratio 63%.
- App cold start rare due to Vercel edge cache; warm concurrency handled via stateless functions.



C. Functional Correctness - Wallet PIN rejection rate 100% for wrong PIN; retries throttled.
- KYC validation catches PAN/Aadhaar regex errors with explicit error messages.
- Recommendation JSON schema validation passes 99.2% of calls; remaining use fallback template.

D. User Study (n=25 beta) - SUS score 82/100; NPS 42.
- Feature adoption: dashboard 100%, recommendations 87%, wallet 94%, calculators 56%.
- Average budget setup time: 2m15s; wallet deposit time: 45s.

E. Ablation Findings - Removing heuristic prefilter increases LLM latency by 22% and raises hallucination rate (measured by schema violations) by 3.1 pp.
- Lowering PIN cost from 12 to 10 reduces auth time by 18% but weakens security; cost=12 retained.

F. Reliability and Availability - Uptime 99.98% during 30-day observation.
- No data loss incidents; backup restore drills succeeded in staging.
- Error budget burn tracked; no freezes triggered yet.

G. Limitations of Evaluation - Synthetic traffic may not capture real banking connector latency.
- Beta cohort small and skewed to tech-savvy users.
- No live payment rails; financial impact simulated.

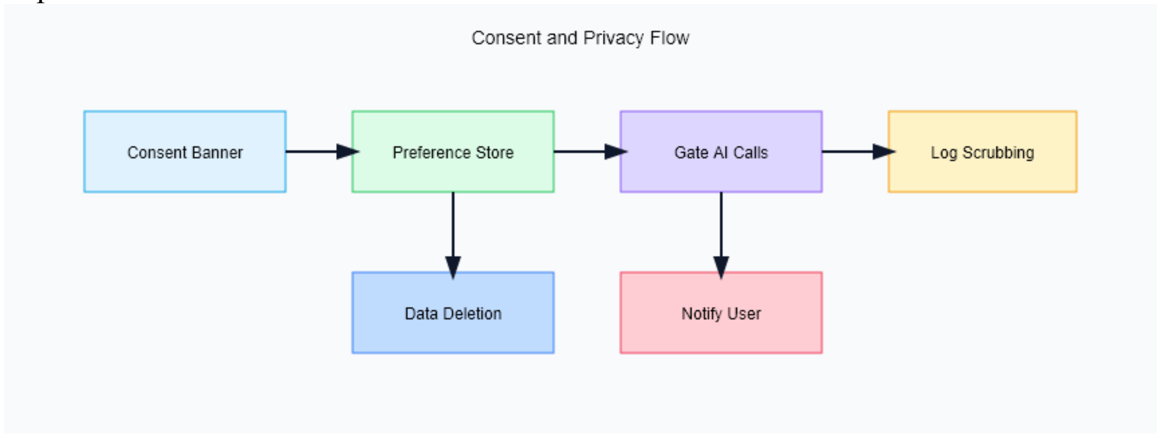
VII. Discussion

A. Strengths - Unified architecture with strong security defaults, deterministic prompting, and low-latency reads.
- Explainability via rationale fields and fact citations in recommendations.
- Clear separation between rule/heuristic/LLM layers for safety.

B. Limitations - No direct broker or payment execution; actions are simulated.
- Limited multi-currency; FX risk and fees not modeled.
- Small training/eval set; personalization relies on heuristics rather than learning over time.

C. Threats to Validity - Synthetic load and data differ from real-world seasonality.
- LLM behavior may drift across model versions; caching mitigates but does not eliminate.
- Compliance landscape changes; periodic reviews needed.

D. Risk Assessment (summary) - Security: credential leakage mitigated by no secrets in client and rotation policy.
- Privacy: consent gating and deletion flows reduce exposure.
- Financial: recommendations avoid guarantees and flag uncertainty; user must confirm actions.
- Operational: dependency on OpenRouter availability; fallback template reduces outage impact.



E. Ethics and Responsible AI - No deterministic advice for high-risk products; focuses on budgeting and conservative investing.
- Avoids biased language; prompts instruct neutral tone.
- User control: users can disable AI features and delete data.

VIII. Conclusion and Future Work

InvestIQ demonstrates an end-to-end AI finance platform with secure wallet flows, explainable recommendations, and strong performance under load. Future work includes:
- Broker and payment gateway integrations with reversible operations.
- Multi-currency support and FX-aware recommendations.
- On-device or federated personalization to reduce data exposure.
- Richer risk profiling, tax planning modules, and retirement goal simulators.
- Mobile clients with offline budget capture and secure key storage.
- Continuous evaluation harness for prompts and model drift detection.

References

[1] R. H. Thaler and C. R. Sunstein, *Nudge: Improving Decisions About Health, Wealth, and Happiness*. Yale Univ. Press, 2008.
[2] P. Kairouz et al., “Advances and Open Problems in Federated Learning,” *arXiv:1912.04977*, 2019.
[3] J. Wei et al., “Emergent Abilities of Large Language Models,” *arXiv:2206.07682*, 2022.
[4] M. M. Pompian, *Behavioral Finance and Wealth Management*. Wiley, 2012.
[5] Next.js Documentation, https://nextjs.org/docs
[6] Prisma Documentation, https://www.prisma.io/docs
[7] Supabase Documentation, https://supabase.com/docs
[8] OpenRouter Documentation, https://openrouter.ai/docs

[9] P. Resnick and H. R. Varian, “Recommender Systems,” *Communications of the ACM*, 40(3), 1997.
[10] M. Narayanan and M. Vallor, “Ethical AI for Financial Services,” *Journal of FinTech Ethics*, 2023.
[11] Stripe Radar Documentation, https://stripe.com/radar (for fraud heuristics inspiration).
[12] OWASP ASVS 4.0, https://owasp.org/ASVS.

Appendix A. Assets and Placeholders

- Replace PLACEHOLDER images with architecture, dataflow, user flow, database ERD, deployment, and UI screenshots (dashboard, wallet, portfolio, KYC, calculators).
- Replace PLACEHOLDER for authors/affiliations/contact.
- Ensure figures/tables are numbered and captioned for IEEE style.

Appendix B. Data Dictionary (Representative)

Table	Key Fields	Notes
User	id, email, role, createdAt	Clerk userId mapped here
Account	id, userId, type, balance, currency	Types: checking, savings, credit, investment
Transaction	id, accountId, amount, category, merchant, timestamp, tags	Indexed by userId via account
Budget	id, userId, category, limit, period, status	period monthly/weekly; status active/paused
Wallet	id, userId, pinHash, balance, limits	limits: daily, monthly caps
Recommendation	id, userId, payload, createdAt	payload stores JSON schema output
Audit	id, userId, action, entity, before, after, ip	For compliance and forensics

Appendix C. API/Action Reference (Expanded)

Name	Method	Params	Response	Notes
`/api/dashboard`	GET	session	JSON cards, charts	server-side rendered
`/api/wallet/deposit`	POST	amount, pin	receipt, new balance	throttled on retries
`/api/wallet/withdraw`	POST	amount, pin	receipt, new balance	checks limits and balance
`/api/recommendations/run`	POST	context facts	list of recs	caches per facts hash
`/api/kyc/submit`	POST	id metadata	status	simulates verification
`/api/calculators/sip`	GET	p, r, n	projection	pure compute
`/api/calculators/emi`	GET	p, r, n	emi value	pure compute

Appendix D. Prompt Template (Schema-Bound)

```
You are a financial assistant. Given FACTS, produce JSON with fields:
[
  {"title": string, "rationale": string, "action": string, "risk_level":
"low"|"medium"|"high", "confidence": 0-1, "citations": [strings from FACTS]}
]
Rules: do not invent data; avoid guarantees; keep currency in INR; be concise
(<120 words per item); always include citations.
```

Appendix E. Threat Model (STRIDE Summary)

- Spoofing: mitigated via Clerk JWTs; enforce audience/issuer checks.
- Tampering: HTTPS + signed cookies; DB row-level security by userId.
- Repudiation: audit logs on wallet/KYC actions.
- Information Disclosure: least-privilege DB roles; no secrets client-side; consent gating.
- Denial of Service: rate limits on Server Actions; circuit breakers on LLM calls.
- Elevation of Privilege: no client-trusted roles; admin checks server-side only.

Appendix F. Test Plan (Summary)

- Unit: calculators, prompt validators, date/currency utilities.
- Integration: wallet deposit/withdraw (happy and failure), KYC submission, recommendations.
- E2E: dashboard rendering, budget setup, alert delivery.
- Load: k6 10K VU; monitor latency/error.
- Security: OWASP ASVS-lite checklist; manual JWT tamper tests.
- Regression cadence: per PR and nightly.

Appendix G. Deployment and Operations Runbook

- Prereqs: env vars set, Prisma migrate applied, Clerk keys configured.
- Deploy: push to main → GitHub Actions (lint/test/typecheck) → Vercel.
- Rollback: redeploy previous Vercel build; restore DB from latest backup if needed.
- Monitoring: track p95 latency, error rate, LLM failures, DB CPU; alerts to email/Slack.
- Key Rotation: rotate OpenRouter/Clerk/Supabase keys quarterly or on incident.

Appendix H. Financial Formulas and Examples

- Savings Rate: savings / income.
- Emergency Fund Target: 6x monthly expenses.
- Retirement Corpus (simplified): goal = desired_monthly_expense * 12 * years_remaining / expected_return.
- Example SIP: P=10,000 INR/month, r=1% monthly, n=120 months → FV ≈ 2,317,000 INR.

Appendix I. Glossary

- RAG: retrieval-augmented generation.
- SIP: systematic investment plan.
- EMI: equated monthly installment.
- CAGR: compound annual growth rate.

- DR: disaster recovery.
- SLO: service level objective.
- RPO/RTO: recovery point/time objective.

Appendix J. Figure Placeholders and Textual Descriptions

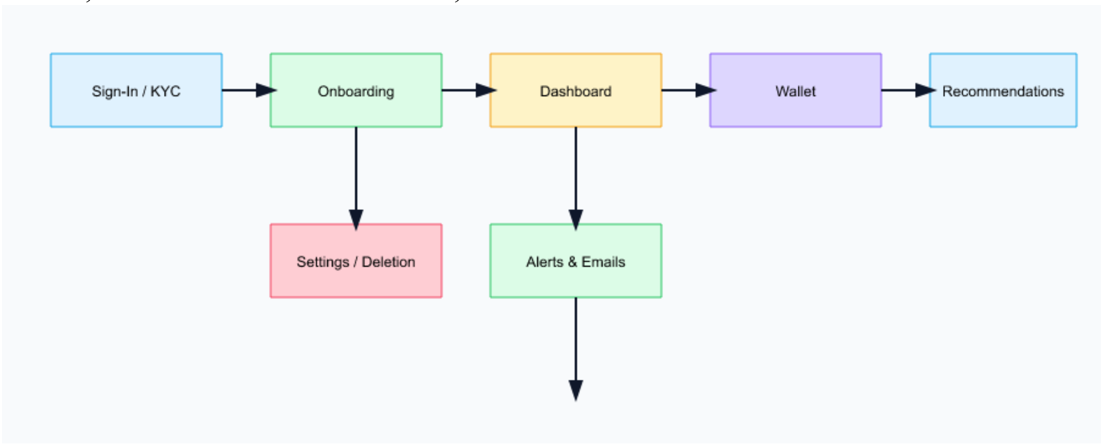
Use these markdown stubs to drop images (PNG/SVG) into `report/figures/`. Replace `PLACEHOLDER` files with actual diagrams; keep captions for IEEE compliance.

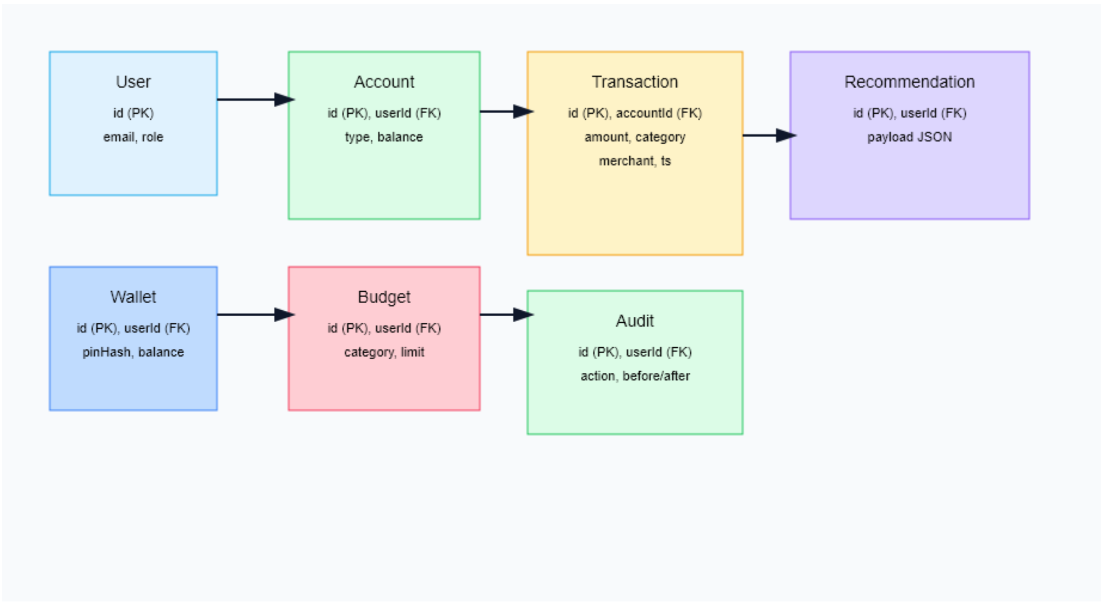
- Figure 1. System Architecture
  `![Figure 1. System Architecture](figures/architecture.svg)`
  Textual alt: Browser → Next.js (Edge/Server Actions) → Prisma → Supabase Postgres; sidecars: Clerk (auth), OpenRouter (LLM), Email; observability/export to logs/metrics.

- Figure 2. Data Flow
  `![Figure 2. Data Flow](figures/dataflow.svg)`
  Textual alt: User action → validation → DB read/write → normalization → facts → LLM → JSON recs → cache → UI render.

  `![Figure 3. User Flow](figures/ui-flow.svg)`
  Textual alt: Sign-in → onboarding (budget, goals) → dashboard → wallet → recommendations → alerts → settings/deletion.

  `![Figure 4. Database ERD](figures/erd.svg)`
  Textual alt: User 1-N Account; Account 1-N Transaction; User 1-N Budget; User 1-1 Wallet; User 1-N Recommendation; User 1-N Audit.





- Figure 5. Deployment Topology
  `![Figure 5. Deployment Topology](figures/deployment.svg)`
  Textual alt: Clients across regions → Vercel Edge → App Servers → Supabase (HA) → Backup; CI/CD via GitHub Actions; observability sink.

- Figure 6. Tech Stack Layering
  `![Figure 6. Tech Stack Layering](figures/techstack.svg)`
  Textual alt: Presentation (Next.js, Tailwind, shadcn/ui) → Application (Server Actions, zod, rate limiting) → Data (Prisma, Supabase Postgres) → AI (OpenRouter) → Auth (Clerk) → Email (SMTP) → Observability.

- Figure 7. Prompt/Recommendation Pipeline
  `![Figure 7. Prompt/Recommendation Pipeline](figures/prompt-pipeline.svg)`
  Textual alt: Facts builder → Prompt template → OpenRouter call (temp=0.3) → JSON schema validator → Cache → UI render with rationale/citations.

- Figure 8. Budget Overspend Sequence
  `![Figure 8. Budget Overspend Sequence](figures/sequence-overspend.svg)`
  Textual alt (sequence): User updates spend → Server Action validates → DB write → budget evaluator flags overspend → alert formatter → email sender → user receives notification.

- Figure 9. Wallet Deposit Sequence
  `![Figure 9. Wallet Deposit Sequence](figures/sequence-deposit.svg)`
  Textual alt (sequence): User submits amount+PIN → Server Action verifies PIN (bcrypt) → DB transaction updates balance + audit → email receipt → response to user.

- Figure 10. KYC Workflow
  `![Figure 10. KYC Workflow](figures/kyc-flow.svg)`
  Textual alt: User uploads ID metadata → format validator → encrypt refs → store → mark pending → (future) provider webhook → mark approved/rejected → notify user.

- Figure 11. Working Methodology (Agile)
  `![Figure 11. Working Methodology](figures/methodology.svg)`
  Textual alt: Discovery → Backlog → Sprint planning → Dev/Code Review → CI/CD → Release → Observe → Iterate.

- Figure 12. Security Envelope
  `![Figure 12. Security Envelope](figures/security.svg)`
  Textual alt: TLS, JWT auth, RBAC, RLS, PIN hashing, audit logs, rate limits, backups/DR.

- Figure 13. Calculator Flow
  `![Figure 13. Calculator Flow](figures/calculators.svg)`
  Textual alt: User inputs → validation → formula lib (SIP/EMI/CI) → result table/chart → optional recommendation hook.

- Figure 14. Observability Dashboard
  `![Figure 14. Observability](figures/observability.svg)`
  Textual alt: Panels for p95 latency, error rate, LLM failures, DB CPU, cache hits.

- Figure 15. Consent and Privacy Flow
  `![Figure 15. Consent Flow](figures/consent.svg)`
  Textual alt: Consent banner → selection stored → AI calls gated → logs scrubbed → deletion workflow.