Software Design Document (SDD)

Mobile-First Expense Tracker

Prepared by: KOTLA ANUDEEP REDDY
Contact: anudeepreddy.kotla@gmail.con

August 11, 2025

Contents

1 Introduction

1.1 Purpose of this Document

This Software Design Document (SDD) describes the design and implementation details for the Mobile-First Expense Tracker ("ExpenseTracker") MVP. It translates the SRS into a concrete system design suitable for development and deployment, and provides guidance for implementation, testing, and operations.

1.2 Scope

This SDD covers:

- Architecture and component breakdown
- Detailed module-level design
- Data model, DDL and indexing strategy
- API contracts and example request/response bodies
- Background processing and sequence flows
- Security design and operational guidance
- Testing strategy and deployment guidelines

1.3 Audience

Developers, testers, DevOps, project stakeholders, and future maintainers.

1.4 Definitions and Acronyms

- **UPI** — Unified Payments Interface
- **AA** — Account Aggregator (RBI framework)
- **FCM** — Firebase Cloud Messaging
- **JWT** — JSON Web Token
- **DB** — Database
- **MVP** — Minimum Viable Product
- **SRS** — Software Requirements Specification
- **SDD** — Software Design Document

2 High-Level Architecture

2.1 Overview

The system is designed as a cloud-hosted, mobile-first solution with the following primary components:

1. **Flutter Mobile App** — cross-platform UI for Android/iOS, local cache, secure storage for refresh tokens, push token registration.

2. **FastAPI Backend** — REST API implementing business logic, AA orchestration, authentication, and notification orchestration.

3. **PostgreSQL** — primary persistent store (managed service recommended).

4. **Redis** — caching, Celery broker, ephemeral counters.

5. **Celery Workers** — background processes for ingestion, classification, budget checks, and notification tasks.

6. **Notification Gateways** — FCM for push, SendGrid/Mailgun for email, MSG91/Twilio for SMS.

7. **Secrets Management** — cloud provider secrets or Vault (recommended for production).

2.2 Architecture Diagram (placeholder)

Architecture diagram not found. Replace `architecture_diagram_placeholder.png` with final diagram.

3 Design Considerations and Constraints

3.1 Primary Design Drivers

- **AA-first data source**: Account Aggregator is the canonical transaction source because SMS/email notifications are unreliable.
- **Mobile-first UX**: Low-latency mobile experiences and offline-readiness for recent transactions.
- **Background continuity**: Transaction ingestion and alerts must operate regardless of user session validity.
- **Security and privacy**: Sensitive financial tokens and user data encrypted at rest and in transit.

3.2 Constraints

- RBI Account Aggregator availability and sandbox access needed for full integration.
- Initial primary bank example: State Bank of India (SBI).
- Budget enforcement (hard-block) at bank-level is out of scope for MVP; instead the system reserves allocations and recommends transfers.
- Mobile target devices: modern Android and iOS phones.

3.3 Assumptions

- User has at least one bank account that supports AA.
- Internet connectivity is available for sync; basic offline viewing is acceptable for recent cached data.
- Service will be deployed in a managed cloud provider (Render/DigitalOcean/Railway).

4 Module-Level Design

Each module description includes responsibilities, design decisions, data inputs/outputs, and API endpoints where applicable.

4.1 Authentication Module

Responsibilities:

- User registration, login, JWT issuance, refresh token handling
- OTP flows and social login (Google) support
- Enforce session TTL for UI while allowing background services to continue

Design:

- Use Argon2 (recommended) or bcrypt for password hashing.
- JWT short-lived access tokens (15 minutes) + refresh tokens (30 days) stored encrypted in DB.
- Refresh tokens revocable (blacklist or token versioning).
- Rate limit login attempts (Redis-based).

Endpoints:

- `POST /api/register` — register new user
- `POST /api/login` — login, returns access + refresh token
- `POST /api/token/refresh` — refresh access token
- `POST /api/auth/otp/request` — request OTP for phone
- `POST /api/auth/otp/verify` — verify OTP

4.2 Account Aggregator Module

Responsibilities:

- Orchestrate AA consent flow and callbacks
- Securely store AA consent IDs and tokens
- Provide sync interfaces for fetching transactions (incremental)

Design notes:

- Backend handles the AA redirect/callback, stores encrypted tokens in the ‘bank$_accounts‘table.Useserver-sidekeymanagement; rotatekeysperiodically.$

- Implement token refresh job (Celery scheduled job) before expiry.

Endpoints:

- `POST /api/aa/connect` — start AA consent
- `GET /api/aa/callback` — AA callback handler
- `POST /api/aa/sync` — admin/internal trigger to force sync

4.3 Transaction Module

Responsibilities:

- Store raw transactions and normalized fields
- Expose retrieval APIs with filtering and paging
- Allow manual add/edit of transactions

Design:

- Store raw AA payload in JSONB and normalized fields (amount, ts, merchant_norm, txn_type, category_id)

- Use ‘txn$_{i}d‘fromAAforidempotency$

Endpoints:

- `GET /api/transactions?from=&to=&category=&limit=&offset=`
- `POST /api/transactions/manual`

4.4 Categorization Module

Responsibilities:

- Auto-assign categories to transactions using rule-based + fuzzy matching
- Optional ML classifier fallback (deployed later)
- Provide UI for user corrections and update the per-user keyword map

Design:

- Per-user keyword dictionary stored in DB (JSONB) + normalized tokens index
- Fuzzy matching using trigram or token similarity (threshold configurable)
- Corrections update the dictionary and trigger model re-training / heuristic update

4.5 Budget & Allocation Module

Responsibilities:

- Allow user to define budgets and allocation rules
- Detect salary credits and allocate funds into Fixed/Save/Spend buckets
- Maintain allocation balances and history

Design:

- Budgets stored per category with ‘period‘ (monthly/weekly), ‘limit$_{a}mount‘,and‘threshold_{p}ct‘Allocations$
  $basedoramount - basedrules$

- For enforcement, the app will recommend transfers or maintain a simulation of reserved funds; hard bank-level enforcement is not in MVP

Endpoints:

- `POST /api/budgets` — create/update budget
- `GET /api/allocations` — get current allocation balances
- `POST /api/allocations/fund` — manual fund adjustment (simulation)

4.6 Alerting & Notification Module

Responsibilities:

- Generate alerts on threshold crossing or suspicious transactions
- Deliver alerts via push (FCM), email, and SMS
- Store alert history and support acknowledgement

Design:

- Templates for alert messages, severity levels (info/warn/critical)
- Event-driven: on classification completion, budget check runs and enqueues notifications if needed
- Retry with exponential backoff for notification delivery failures

Endpoints:

- `GET /api/alerts`
- `POST /api/alerts/ack`

4.7 Admin Module

Responsibilities:

- Admin-only endpoints for user support, forcing ingestion, viewing system health
- Stronger auditing and RBAC

5 Data Model and Database Design

PostgreSQL is the recommended DB for its relational features + JSONB support.

5.1 Logical Data Model (summary)

Entities:

- **users** — users and authentication metadata
- **bank_accounts** — AA-related info per bank account
- **transactions** — raw + normalized transaction data
- **categories** — expense categories, per-user
- **budgets** — category budgets
- **allocations** — Fixed/Spend/Save buckets
- **alerts** — notification records
- **audit_logs** — change history

5.2 Representative DDL

```
-- Users
CREATE TABLE users (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  name TEXT,
  email TEXT UNIQUE,
  phone TEXT UNIQUE,
  password_hash TEXT,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT now(),
  last_active TIMESTAMP WITH TIME ZONE
);

-- Bank Accounts
CREATE TABLE bank_accounts (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID REFERENCES users(id) ON DELETE CASCADE,
  bank_name TEXT,
  aa_consent_id TEXT,
  aa_token_enc BYTEA,
  meta JSONB,
  last_sync TIMESTAMP WITH TIME ZONE
);

-- Categories
CREATE TABLE categories (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID REFERENCES users(id) ON DELETE CASCADE,
  name TEXT,
  type TEXT CHECK (type IN (’fixed’,’variable’,’savings’)),
  parent_id UUID NULL REFERENCES categories(id)
);

-- Budgets
```

```
CREATE TABLE budgets (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID REFERENCES users(id) ON DELETE CASCADE,
  category_id UUID REFERENCES categories(id),
  period TEXT CHECK (period IN (’monthly’,’weekly’)),
  limit_amount NUMERIC,
  threshold_pct INTEGER DEFAULT 80
);

-- Transactions
CREATE TABLE transactions (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  account_id UUID REFERENCES bank_accounts(id) ON DELETE CASCADE,
  txn_id TEXT UNIQUE,
  amount NUMERIC,
  txn_type TEXT CHECK (txn_type IN (’debit’,’credit’)),
  ts TIMESTAMP WITH TIME ZONE,
  merchant_raw TEXT,
  merchant_norm TEXT,
  category_id UUID REFERENCES categories(id),
  raw_json JSONB,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT now()
);

-- Allocations
CREATE TABLE allocations (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID REFERENCES users(id) ON DELETE CASCADE,
  name TEXT,
  target_amount NUMERIC,
  current_amount NUMERIC,
  last_funded TIMESTAMP WITH TIME ZONE
);

-- Alerts
CREATE TABLE alerts (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID REFERENCES users(id) ON DELETE CASCADE,
  alert_type TEXT,
  message TEXT,
  sent_at TIMESTAMP WITH TIME ZONE,
  status TEXT
);

-- Audit Logs
CREATE TABLE audit_logs (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID,
  action TEXT,
  details JSONB,
  actor TEXT,
  ts TIMESTAMP WITH TIME ZONE DEFAULT now()
);
```

5.3 Indexes and Performance

- Index on transactions(txn$_id)foridempotency.Indexon$transactions(account$_id, ts)toacceleratetime$

- Redis counters for per-category running totals to reduce DB load on ingestion.

6 APIs (Representative)

The implementation will expose an OpenAPI spec (auto-generated by FastAPI). Below are representative endpoints with example request/response shapes.

6.1 Authentication

POST /api/register

```
Request:
{
  "name": "Anudeep",
  "email": "anudeep@example.com",
  "phone": "+919876543210",
  "password": "securePassword!"
}
Response: 201 Created
{
  "id": "uuid",
  "email": "anudeep@example.com"
}
```

POST /api/login

```
Request:
{
  "email": "anudeep@example.com",
  "password": "securePassword!"
}
Response:
{
  "access_token": "eyJ...",
  "refresh_token": "r_abc..."
}
```

6.2 AA / Bank

POST /api/aa/connect

```
Request: { "bank": "SBI", "account_nickname": "SBI Main" }
Response: { "consent_url": "https://aa.example/consent/..." }
```

6.3 Transactions

GET /api/transactions?from=2025-08-01to=2025-08-10

```
Response:
{
  "transactions": [
    {
      "id": "uuid",
      "txn_id": "aa-123",
      "amount": 250.00,
      "txn_type": "debit",
      "ts": "2025-08-10T07:31:00Z",
      "merchant_raw": "PHONEPE-12345",
      "merchant_norm": "PhonePe",
      "category_id": "uuid"
    },
    ...
  ],
  "meta": { "limit": 50, "offset": 0, "total": 102 }
}
```

6.4 Budgets

POST /api/budgets

```
Request:
{
  "category_id": "uuid",
  "period": "monthly",
  "limit_amount": 4000,
  "threshold_pct": 80
}
Response: 201 Created with budget resource
```

6.5 Alerts

GET /api/alerts

```
Response:
[
  { "id":"uuid", "alert_type":"budget_warn", "message":"Food at 82% of budget", "
      ↪ sent_at":"..." },
  ...
]
```

7 Sequence Diagrams and Flows (Textual)

7.1 AA Connect Flow

1. Mobile App requests `/api/aa/connect`.
2. Backend creates AA consent object and returns consent URL to client.
3. User completes consent via AA UI; AA calls backend callback `/api/aa/callback`.
4. Backend stores AA tokens encrypted and schedules an initial ingestion run.

7.2 Transaction Ingestion & Alert Flow

1. Celery worker polls AA (or webhook triggers) to fetch new transactions.
2. For each transaction:

- If `txn`$_idexistsinDB, skip.ElseinsertrawJSONB, normalizefields, andenqueueclassificationjob.$

3. Classification job assigns a category (rule-based or ML) and updates DB.
4. Budget checking job updates running totals (Redis + DB) and enqueues alerts if thresholds crossed.
5. Notification worker delivers alerts via FCM/email/SMS.

8 Algorithms and Business Rules

8.1 Idempotent Ingestion

- Use AA-provided unique transaction IDs to ensure idempotency.
- Lock or use upsert with unique constraint on `txn_id` to avoid duplicates.

8.2 Categorization (Hybrid Approach)

1. Normalize merchant string: lowercase, remove punctuation, strip tokens like `*UPI*`, trim multiple spaces.
2. Check per-user dictionary for exact/fuzzy matches (threshold 0.8).
3. If match found, assign mapped category.
4. Else optionally run ML text-classifier and accept if confidence ¿ 0.7.
5. Else mark as `uncategorized` for later user action.

8.3 Budget Check and Notification Rules

- Each budget has a `threshold_pct` (default 80%).
- On each debit transaction post-categorization, atomically update running sum.
- If running_sum ¿= threshold_pct * limit, create a `warning` alert.
- If running_sum ¿= 100% * limit, create a `critical` alert.
- Alerts are delivered immediately (goal: within 2 seconds) unless notification provider throttling occurs (exponential backoff applied).

8.4 Allocation on Salary Detection

- Detect salary credit via AA-provided txn_type and merchant information (expandable set of payroll keywords).
- Apply user-configured allocation rules (e.g., Fixed 40%, Save 20%, Spend 40%).
- Update `allocations.current_amount`. If user permits, provide guidance for standing instructions to move funds.

9 Security Architecture

9.1 Transport & Storage

- All endpoints use HTTPS/TLS 1.2+.
- AA tokens encrypted at rest (AES-256 with managed keys).
- Passwords hashed with Argon2 or bcrypt.
- Refresh tokens encrypted in DB and revocable.

9.2 Authentication & Authorization

- Role-based checks for admin endpoints.
- Short-lived access tokens (JWT) for API access; refresh tokens for session restoration.
- Use refresh token rotation (issue new refresh token on use and revoke old one).

9.3 Operational Security

- Store secrets in the provider secret manager or HashiCorp Vault.
- Audit logging for sensitive actions (category overrides, manual transaction edits, AA consent/rescissions).
- Regular security scanning (Snyk, Bandit) in CI pipeline.

10 Reliability, Availability & Scalability

- Background workers (Celery) horizontally scalable — run multiple worker replicas.
- Use managed Postgres for point-in-time recovery and backups.
- Redis for ephemeral counters to avoid DB hot-writes.
- Health checks and Prometheus metrics around ingestion latency, queue length, and failed tasks.
- Use horizontal scaling for API server behind load balancer as load increases.

11 Testing Strategy

11.1 Unit Testing

- Backend: pytest covering categorization rules, budget math, JWT flows, DB models (using test DB).
- Frontend: Flutter unit tests for UI logic, route guards, and local storage.

11.2 Integration Testing

- Mock AA provider to test consent and ingestion flows.
- End-to-end tests for user signup, connect AA, ingest sample transactions, alert generation.

11.3 Performance & Load Testing

- Simulate ingestion of high-volume transactions to validate worker scaling (target baseline: 10k txns/min in scale tests).
- Validate alert delivery latency under load.

11.4 Security Testing

- SAST tools (Bandit, Flake8), dependency scanning (Snyk).
- Periodic pen-testing for production systems.

12 Deployment and DevOps

12.1 Local Development

- Docker Compose for local stack: FastAPI, Postgres, Redis, Celery worker, Flower (optional).
- Provide ‘make‘ targets for common tasks (migrate, test, seed).

12.2 CI/CD

- GitHub Actions: lint, unit tests, build container image, push to registry.
- On merge: deploy to staging, run integration tests; manual promote to production.

12.3 Cloud Deployment (MVP)

- Use a reliable provider (Render / DigitalOcean / Railway) for ease of use and stability.
- Managed Postgres + Redis (managed or provider add-on).
- Containerized FastAPI + Celery workers (separate services).
- Secrets stored in provider secret store.

12.4 Cost Considerations

- Start with minimal managed instances for staging and upgrade as user base grows.
- Evaluate provider free tier and hobby tiers; adjust for AA provider sandbox/production access costs if any.

13 Operational Runbook (Summary)

13.1 Monitoring

- Monitor: ingestion latency, task queue size, failed tasks, DB connection pool exhaustion, AA token expiry.
- Set alerts to Slack/Email for critical incidents.

13.2 Common Recovery Steps

- If ingestion fails: check AA token validity and refresh; review logs and retry failed batches.
- If DB corruption: follow managed provider restore procedure (PITR if enabled).
- For notification failures: check provider quotas and credential validity.

13.3 Data Retention & Privacy

- Default retention: keep transactions for 2 years (configurable).
- Provide user export and deletion features to comply with data privacy requests.

14 Appendices

14.1 Appendix A: Example Categorization Keywords

```
{
  "zomato": "Food",
  "swiggy": "Food",
  "phonepe": "Payments",
  "metro": "Transport",
  "amazon": "Shopping"
}
```

14.2 Appendix B: Example Makefile / Commands

```
# Example commands
make build # build docker images
make dev-up # start compose stack
make migrate # run alembic migrations
make test # run pytest
make lint # run linters
```

14.3 Appendix C: Placeholder Diagrams

- architecture_diagram_placeholder.png
- sequence_aa_connect.png
- sequence_ingestion.png
- ui_mockup_placeholder.png

Prepared by: Kotla Anudeep Reddy
Contact: anudeepreddy.kotla@gmail.con
Version: 1.0