High-Level Design (HLD)

System Overview

- Native Android (Kotlin + MVVM)
- Backend: Spring Boot
- AI Service: ChatGPT API (future: custom LLM)
- Deployment: Docker + Kubernetes

Architecture Diagram

[Android App (Kotlin + MVVM)] | | REST API calls (HTTPS) v [Spring Boot Backend (REST APIs)] | | Connects v [Database (MySQL / MongoDB)] | | Optional: Logging/Analytics v [AI Service (ChatGPT API / Custom LLM)]

Key Modules

Frontend (Android)

- User Authentication
- Expense Management
- Budget Management
- Analytics
- AI Integration
- Offline-first (WorkManager)

Backend (Spring Boot)

- REST APIs
- Database layer
- AI Service Integration
- Security
- Logging & Monitoring

Database

- Users table
- Expenses table
- Categories table
- Budget table
- AI Logs table (optional)

AI Service

- ChatGPT API: Categorization, insights, suggestions
- Future LLM: Host on backend, consume via API

Deployment

- Dockerize backend
- Host on Kubernetes cluster
- Optional: CI/CD pipeline

Low-Level Design (LLD)

Database Design (Tables)

Users

- user_id (UUID, PK)
- name (VARCHAR)
- email (VARCHAR, unique)
- password (VARCHAR, hashed)
- created_at (TIMESTAMP)

Expenses

- expense_id (UUID, PK)
- user_id (UUID, FK → Users)
- title (VARCHAR)
- amount (DECIMAL)
- category (VARCHAR)
- date (DATE)
- created_at (TIMESTAMP)

Categories

- category_id (UUID, PK)
- name (VARCHAR)

Budgets

- budget_id (UUID, PK)
- user_id (UUID, FK → Users)
- category (VARCHAR)
- limit_amount (DECIMAL)
- start_date (DATE)
- end_date (DATE)

AI Logs (Optional)

- log_id (UUID, PK)
- user_id (UUID, FK → Users)
- query (TEXT)
- response (TEXT)
- timestamp (TIMESTAMP)

API Endpoints

User - POST `/api/auth/signup` - POST `/api/auth/login` - GET `/api/users/{id}`

Expenses - POST `/api/expenses` - PUT `/api/expenses/{id}` - DELETE `/api/expenses/{id}` - GET `/api/expenses`

Budgets - POST `/api/budgets` - GET `/api/budgets` - PUT `/api/budgets/{id}`

Analytics / AI - GET `/api/analytics/summary` - POST `/api/ai/query`

Android App Module Structure

```
com.example.expensetracker
├─ data
│   ├─ model
│   ├─ repository
│   └─ local (Room)
├─ network
│   └─ ApiService.kt
├─ ui
│   ├─ auth
│   ├─ dashboard
│   ├─ expense
│   └─ ai
├─ viewmodel
└─ utils
```

- MVVM: ViewModel → Repository → Local DB
- WorkManager: Offline sync
- LiveData / StateFlow: Reactive UI

AI Integration

- Backend `/ai/query` → ChatGPT API → response → AI Logs → Android UI
- Future: Custom LLM hosted on backend

Deployment / Kubernetes

- Dockerize backend
- MySQL/MongoDB in containers
- Kubernetes: Deployment + Service + Ingress + ConfigMaps + Secrets + HPA

Sequence Flow: Adding Expense

1. User enters expense
2. Store locally (Room)
3. WorkManager syncs → POST `/api/expenses`
4. Backend stores in DB
5. Analytics updated
6. UI updated via LiveData

Tech Stack

Layer	Technology
Frontend	Native Android (Kotlin, MVVM, Jetpack)
Backend	Spring Boot, REST APIs, JWT, MySQL/MongoDB
AI	ChatGPT API / Custom LLM
Deployment	Docker + Kubernetes
Offline sync	WorkManager + Room
State management	LiveData / StateFlow
Notifications	Android Notifications / WorkManager

End of HLD & LLD