High Level Design Document

Introduction

This High Level Design (HLD) document outlines the architecture and core components for **FinLedger - Simple Expense Tracker API**. The project delivers a backend service for personal expense tracking, supporting user registration, authentication, and CRUD operations on expenses, built with Node.js, Express, and MySQL.

---

1. System Architecture Overview

Architecture Description:
FinLedger is a RESTful API backend structured in a modular fashion. Clients interact via HTTP(S) requests. The API server handles authentication, business logic, and data persistence using a MySQL database.

Main System Components:

Component	Description
API Server	Node.js/Express app handling HTTP requests and routing
Auth Module	Manages user registration, login, and JWT-based authentication
Expense Module	Handles CRUD operations for user expenses
Database Layer	MySQL database for persistent storage of users and expenses

---

2. Component Interactions

Sequence Step	Interaction Description
1	Client sends HTTP request to API Server (e.g., register, login, expense CRUD)
2	API Server routes request to appropriate module (Auth or Expense)
3	Auth Module verifies credentials or JWT token as needed
4	Expense Module processes business logic and interacts with Database Layer
5	Database Layer executes SQL queries and returns results
6	API Server formats and returns HTTP response to client

---

3. Data Flow Overview

Data Flow	Source	Destination	Purpose
User Registration/Login	Client	Auth Module	Create user or authenticate credentials

Auth Token Validation	Client	Auth Module	Verify JWT for protected endpoints
Expense CRUD Operations	Client	Expense Module	Create, read, update, delete expenses
Data Persistence	Modules	Database	Store/retrieve user and expense data

---

4. Technology Stack

Layer/Function	Technology/Framework
API Server	Node.js, Express
Authentication	JWT (JSON Web Tokens)
Database	MySQL
ORM/Query Builder	Sequelize or Knex.js (optional)
API Documentation	OpenAPI/Swagger (optional)

---

5. Scalability & Reliability

- **Stateless API:** Enables horizontal scaling by deploying multiple API server instances behind a load balancer.
- **Database Reliability:** Use MySQL with regular backups; consider read replicas for scaling reads.
- **Security:** All sensitive endpoints require JWT authentication; passwords are hashed (e.g., bcrypt).
- **Error Handling:** Consistent error responses and input validation to ensure API robustness.

End of Document