Project Overview

This document summarizes the backend development steps and outlines requirements for integrating the UI/UX layer with the HR Management backend system. The backend is built with Python using FastAPI, utilizes SQLAlchemy for ORM, and is designed for compatible SQL databases. It provides secure, modular, and well-documented REST APIs for employee management, attendance, and leave tracking.

1. Backend Development Steps Completed

A. Project Setup & Structure

- **Framework:** FastAPI (Python) selected for its speed, asynchronous support, and automatic OpenAPI documentation.
- **Database:** SQLAlchemy ORM configured for PostgreSQL (can be adapted to other SQL databases).
- **Environment Management:** Sensitive configuration (such as database URLs and secret keys) is loaded from `.env` files using python-dotenv.

B. Authentication & Security

- **User Registration & Login:** Endpoints for user signup and login are implemented.
- **Password Security:** Passwords are hashed using bcrypt before storage.
- **JWT Authentication:** Secure, stateless authentication is implemented using JSON Web Tokens (JWT). Tokens are required for all protected endpoints.
- **Role Management:** Currently, all authenticated users have the same access. Role-based access can be added if needed.

C. Core Features Implemented

- **Employee Management:** CRUD API for employee records, linked to authenticated users.
- **Attendance Tracking:** CRUD API for employee attendance, including calculation of work hours from login and logout times.
- **Leave Management:** CRUD API for leave requests, supporting status and reason fields.
- **Data Validation:** All request and response data is validated using Pydantic schemas.
- **Error Handling:** Consistent HTTP error responses are provided for invalid data, authentication failures, and not found errors.

D. API Documentation & Integration

- **OpenAPI/Swagger:** Full OpenAPI schema is auto-generated and available at `/openapi.json` and `/docs` endpoints for easy API exploration and integration.
- **Modular Routing:** Each feature (employees, attendance, leaves) has its own router for clarity and maintainability.

E. Code Quality

- **Refactoring & Comments:** Code has been reviewed, cleaned, and commented for clarity and maintainability.
- **Separation of Concerns:** Authentication, database models, schemas, and business logic are separated into distinct modules and files.

2. API Endpoints Overview

- All endpoints except `/register` and `/token` require a valid JWT token in the `Authorization: Bearer <token>` header.

3. Requirements for UI/UX Integration

A. Technical Requirements

- **API Consumption:** UI must authenticate users via `/token` and include the JWT in all subsequent requests.
- **Error Handling:** UI should handle HTTP error codes (400, 401, 404, etc.) and display user-friendly messages.
- **Data Validation:** UI should validate forms according to the backend schemas (see `/openapi.json` for field requirements).
- **Pagination:** List endpoints support `skip` and `limit` query parameters for pagination.

B. API Documentation

- **Swagger UI:** Available at `/docs` for interactive API testing.
- **OpenAPI Spec:** Downloadable at `/openapi.json` for use with API tools or code generators.

C. Environment & Dependencies

- **Backend Requirements:** FastAPI, SQLAlchemy, bcrypt, python-jose, python-dotenv, and a PostgreSQL-compatible driver (such as psycopg2 or pymssql).

- **Environment Variables:** The UI/UX team does not need to manage backend `.env` files but should be aware that endpoints may differ in development, staging, or production environments.

4. Next Steps & Recommendations

- Use `/docs` for API reference and testing.
- Start with implementing the authentication flow (register/login).
- Integrate employee, attendance, and leave features as per the API documentation.
- Use `/openapi.json` for code generation or API client scaffolding.