📘 Employee Management System (EMS)

Version 1.0.0 Status: Production-Ready

> ***Lead Developer Note:*** *This system is built with a focus on asynchronous performance, strict Type-Safety (Python 3.12+), and a granular Role-Based Access Control (RBAC) model.*

Executive Summary

The Employee Management System (EMS) is a high-performance, RESTful API designed to manage organizational hierarchies and employee lifecycles. Unlike a standard CRUD app, this project implements enterprise-grade patterns:

01. **Asynchronous I/O:** Fully non-blocking database operations using SQLAlchemy Async and FastAPI.
02. **Strict RBAC:** Granular middleware-level permission enforcement for Admin and User roles.
03. **Data Integrity:** Robust schema validation via Pydantic and database-level unique constraints.

📑 Table of Contents

1. Architectural Design

The system follows a **Layered Architecture** to ensure strict separation of concerns.

```
Client ↔ FastAPI Router ↔ Security Middleware (JWT) ↔ Service Layer ↔
SQLAlchemy (Async) ↔ PostgreSQL
```

Key Design Decisions

- **Statelessness:** No server-side sessions; all data encapsulated in JWT.
- **Dependency Injection:** Modular database and auth injection using FastAPI `Depends`.
- **Async-First:** Native support for high-concurrency workloads.

2. Technology Stack & Rationale

COMPONENT	TECHNOLOGY	RATIONALE
Framework	FastAPI	ASGI performance and auto-OpenAPI generation.
Language	Python 3.12	Advanced generics and improved asyncio.
ORM	SQLAlchemy 2.0	Industry standard for complex async queries.
Migrations	Alembic	Reliable schema versioning.
Auth	JWT / Passlib	Secure bcrypt hashing and token-based access.

3. Project Structure

```
emp/

📂 alembic/
# Database migration scripts

📂 app/
```

```
📂 api/routes/
    🐍 auth.py  # Authentication logic
    🐍 employees.py  # CRUD logic
📂 core/
    config.py  # Env variables
    database.py  # Async Engine
    security.py  # JWT Hashing
📂 models/ # SQLAlchemy definitions
📂 schemas/ # Pydantic validators
🐍 main.py  # Application Entry
📄 requirements.txt  # Dependencies
⚙️ .env.example  # Env Template
```

4. Data Modeling & Persistence

🧑 User Table (Auth Domain)

FIELD	TYPE	CONSTRAINT
id	Integer	Primary Key
username	String	Unique, Indexed
hashed_password	String	Not Null
role	String	Enum: admin | user

👥 Employee Table (Business Domain)

FIELD	TYPE	CONSTRAINT
id	Integer	Primary Key
name	String	Not Null
email	String	Unique, Indexed
department	String	Optional
role	String	Not Null
date_joined	Date	Not Null

5. Security & Authentication

ENDPOINT	METHOD	ADMIN	USER
`/auth/login`	POST	✅	✅
`/auth/me`	GET	✅	✅
`/employees`	GET	✅	✅
`/employees/{id}`	GET	✅	✅
`/employees`	POST	✅	❌
`/employees/{id}`	PUT	✅	❌
`/employees/{id}`	DELETE	✅	❌

Note: Passwords are salted and hashed using **bcrypt** with a cost factor of 12.

---

6. API Reference

🌐 Global API Setup

ENVIRONMENT CONFIGURATION

Base URL	`http://127.0.0.1:8000`
Auth Header	`Authorization: Bearer <token>`

STANDARD STATUS CODES

- **200/201** Success/Created
- **401** Unauthorized
- **403** Forbidden (RBAC)
- **404** Not Found
- **422** Validation Error
- **204** No Content

🔐 Authentication Module

POST `/auth/login` — Endpoint Name: Login User

Full URL	`http://127.0.0.1:8000/auth/login`
Description	Authenticates user and returns JWT.
Authentication	Not Required
Roles Allowed	Admin, User
Request Body	`{"username": "string", "password": "string"}`
Response Body	`{"access_token": "...", "token_type": "bearer"}`

GET `/auth/me` — Endpoint Name: Current Profile

Full URL	`http://127.0.0.1:8000/auth/me`
Description	Fetches identity details for the authenticated requester.
Authentication	Required (JWT)
Headers	`Authorization: Bearer {{token}}`

👥 Employees Module

GET `/employees` — Endpoint Name: List Employees

Full URL	`http://127.0.0.1:8000/employees`
Description	Retrieves paginated employees with dynamic filters.

Query Params	`page (int), page_size (int), department (str), role (str)`
Authentication	Required (JWT)

GET /employees/{id}

Endpoint Name: Get Employee by ID

Full URL	`http://127.0.0.1:8000/employees/{id}`
Description	Retrieves a single employee record by their ID.
Status Codes	200 (Success), 404 (Not Found)

POST /employees

ADMIN ONLY

Full URL	`http://127.0.0.1:8000/employees`
Description	Creates a new employee record.
Request Body	`{"name": "str", "email": "str", "department": "str", "role": "str"}`
Status Codes	**201 (Created), 400 (Duplicate Email)**

PUT /employees/{id}

ADMIN ONLY

Full URL	`http://127.0.0.1:8000/employees/{id}`
Description	Updates an existing record. Partial updates allowed.
Request Body	`{"name": "str", "email": "str", "department": "str", "role": "str"}`

Response Body	`{"id": 1, "name": "...", "email": "...", "role": "...", ...}`
Roles Allowed	**Admin Only**

DELETE `/employees/{id}` **ADMIN ONLY**

Full URL	`http://127.0.0.1:8000/employees/{id}`
Description	Permanently deletes an employee record.
Status Codes	**204 (Deleted), 404 (Not Found)**

7. Testing & Quality Assurance

We maintain **90%+ test coverage** using an automated async test suite.

```
pytest -v --disable-warnings
```

- Isolated PostgreSQL test containers or in-memory SQLite for CI.
- ASGITransport for testing FastAPI endpoints without actual server boot.

8. Development Workflow

Quick Setup

```
# Install & Migrations
pip install -r requirements.txt
alembic upgrade head

# Run Server
uvicorn app.main:app --reload
```

Error Handling Standards

All API errors return a standard JSON structure for frontend consistency:

```
{
  "detail": "Descriptive error message",
  "error_code": "RESOURCE_NOT_FOUND"
}
```