Muhammad Zohaib Aslam

**Documentation Payroll Management System**

Contents

[**1. What is This System?** 3](#_Toc212398948)

[**Key Points:** 3](#_Toc212398949)

[**Technologies Used:** 3](#_Toc212398950)

[**2. Why We Built This** 3](#_Toc212398951)

[**Educational Purpose:** 3](#_Toc212398952)

[**Real-World Application:** 3](#_Toc212398953)

[**3. What You Can Do** 4](#_Toc212398954)

[**Admin Features:** 4](#_Toc212398955)

[**Employee Features:** 4](#_Toc212398956)

[**4. OOP Concepts in the Project** 4](#_Toc212398957)

[**Encapsulation:** 4](#_Toc212398958)

[**Inheritance:** 4](#_Toc212398959)

[**Polymorphism:** 4](#_Toc212398960)

[**Abstraction:** 4](#_Toc212398961)

[**5. How to Install and Run** 5](#_Toc212398962)

[**Step 1: Download the Code** 5](#_Toc212398963)

[**Step 2: Install Requirements** 5](#_Toc212398964)

[**Step 3: Run the App** 5](#_Toc212398965)

[**Step 4: Access in Browser** 5](#_Toc212398966)

[**6. How the System Works** 5](#_Toc212398967)

[**Admin Workflow** 5](#_Toc212398968)

[**Employee Workflow** 5](#_Toc212398969)

[**7. Understanding the Code** 6](#_Toc212398970)

[**Backend Files Explained:** 6](#_Toc212398971)

[**8. File Structure Explained** 6](#_Toc212398972)

[**Why This Organization?** 6](#_Toc212398973)

[**9. Features in Detail** 7](#_Toc212398974)

[**Authentication:** 7](#_Toc212398975)

[**Admin Management:** 7](#_Toc212398976)

[**Employee Portal:** 7](#_Toc212398977)

[**Database Integration:** 7](#_Toc212398978)

[**10. Troubleshooting** 7](#_Toc212398979)

[**11. Future Improvements** 8](#_Toc212398980)

[**12. Summary** 8](#_Toc212398981)

[**Key Takeaways:** 8](#_Toc212398982)

**1. What is This System?**

The **Payroll Management System** is a web-based application that helps companies manage employee records, salaries, and attendance automatically.  
It reduces manual work by providing a digital system where **Admins** can manage employees, and **Employees** can view their salary and details securely.

**Key Points:**

✅ Admin can log in securely  
✅ Create and manage employee accounts  
✅ View all employee salary data  
✅ Employees can log in and see their payroll info  
✅ Secure login system using JWT tokens  
✅ Follows all 4 OOP principles

**Technologies Used:**

* **Backend:** FastAPI (Python web framework)
* **Database:** SQLite
* **ORM:** SQLAlchemy
* **Validation:** Pydantic
* **Authentication:** JWT (Access and Refresh Tokens)
* **Environment:** Visual Studio Code

**2. Why We Built This**

**Educational Purpose:**

This project shows how to apply **Object-Oriented Programming (OOP)** in real applications.  
It teaches how concepts like *Encapsulation*, *Inheritance*, *Polymorphism*, and *Abstraction* help build scalable systems.

**Real-World Application:**

* Automates employee payroll operations
* Provides secure authentication
* Reduces manual record errors
* Demonstrates practical OOP and database integration

**3. What You Can Do**

**Admin Features:**

* Login with admin credentials
* Create and manage employee accounts
* View all employee salary details
* Block or unblock users
* Manage payroll and access controls

**Employee Features:**

* Login with email and password
* View salary and personal profile
* Update password securely
* Access profile via token-based authentication

**4. OOP Concepts in the Project**

The Payroll System strongly follows OOP principles:

**Encapsulation:**

Sensitive data like passwords and tokens are hidden inside classes and functions.  
Only authorized methods can access or modify them.  
Example: Password hashing and verification are done through service classes, not directly by the route.

**Inheritance:**

Common models or base classes are inherited by other components.  
Example: Database models inherit from SQLAlchemy Base for common fields and methods.

**Polymorphism:**

Functions handle different user roles (Admin or Employee) with the same interface.  
Example: The same login function can validate different types of users depending on input.

**Abstraction:**

Complex logic (database sessions, authentication) is hidden inside services.  
Routes interact only with simple function calls — hiding unnecessary details from the user.

**5. How to Install and Run**

**Step 1: Download the Code**

Create a folder named payrollsystem  
Copy the backend files into it.

**Step 2: Install Requirements**

Open terminal inside the backend folder and run:

pip install -r requirements.txt

**Step 3: Run the App**

Run the server:

python run.py

**Step 4: Access in Browser**

Open your browser and visit:

http://127.0.0.1:8000

You will see the main system running.

**6. How the System Works**

**Admin Workflow**

1. Admin logs in using secure credentials
2. Admin creates employee accounts
3. Admin views or updates salary information
4. Admin can block or unblock employees

**Employee Workflow**

1. Employee logs in using email and password
2. System verifies user using JWT authentication
3. Employee can view profile and payroll details
4. Employee can reset password or update info

**7. Understanding the Code**

**Backend Files Explained:**

**main.py** – Application entry point.  
Handles FastAPI setup, routing, and server startup.

**config.py** – Stores environment variables and secret keys.

**database.py** – Creates database connection and session.

**models/** – Defines data tables (Employee, Admin).  
Each model maps directly to a database table.

**schemas/** – Defines input/output data structures using Pydantic.

**routes/** – Contains all API endpoints for Admin and Employee.  
(Example: /admin/login, /user/profile)

**services/** – Handles main logic like authentication, password hashing, and user management.

**8. File Structure Explained**

payrollsystem/

├── backend/

│ ├── run.py ← Starts the FastAPI server

│ ├── requirements.txt ← Dependencies

│ └── app/

│ ├── main.py ← FastAPI setup

│ ├── config.py ← Configuration and secrets

│ ├── database.py ← Database engine and session

│ ├── models/ ← Database models

│ ├── routes/ ← API routes for admin and employee

│ ├── schemas/ ← Data validation schemas

│ └── services/ ← Business logic and utilities

**Why This Organization?**

* Clear separation of logic (models, routes, services)
* Easy to maintain and extend
* Each file handles one clear responsibility

**9. Features in Detail**

**Authentication:**

* Secure JWT-based login
* Access and refresh token system
* Email verification for registration

**Admin Management:**

* Create, view, block, or unblock employees
* Access all payroll data

**Employee Portal:**

* Secure login and access
* View salary and personal info
* Update password using old or verification code

**Database Integration:**

* SQLite used for storing user details and payroll records
* SQLAlchemy ORM used for table mapping

**10. Troubleshooting**

**Problem:** ModuleNotFoundError or import issues  
**Solution:** Make sure you are in the backend directory before running the app.

**Problem:** Database not found  
**Solution:** The database (payroll.db) is created automatically. Ensure write permission in the folder.

**Problem:** JWT token expired  
**Solution:** Use the refresh token endpoint to get a new access token.

**Problem:** “Admin not authenticated”  
**Solution:** Ensure you’re using the correct admin access token in the request header.

**11. Future Improvements**

**Easy Additions:**

* Add attendance management system
* Salary slip generation in PDF
* Employee performance reports

**Advanced Features:**

* Multi-role dashboard (Super Admin, Admin, User)
* Cloud database integration
* Email notifications for salary updates

**Technical Improvements:**

* Use Docker for deployment
* Add unit testing and CI/CD
* Integrate frontend using React or Bootstrap UI

**12. Summary**

This Payroll Management System is a complete example of combining **FastAPI**, **SQLite**, and **OOP principles** in one real-world application.  
It’s designed for learning, clarity, and scalability — showing how clean architecture makes development easier.

**Key Takeaways:**

* **Encapsulation:** Securely hide data and logic
* **Inheritance:** Reuse base models and methods
* **Polymorphism:** Single functions serve multiple user roles
* **Abstraction:** Hide complexity behind service layers
* **Clean Design:** Easy to understand, extend, and maintain