
POJO Entity List (Field Lists Only)

Here are the POJO class structures (field lists only) for the payroll system entities, indicating foreign keys and relationships:

1. **Employee**
    - id (Long, Primary Key)
    - code (String, Unique)
    - firstName (String)
    - lastName (String)
    - email (String, Unique)
    - password (String, Encrypted)
    - mobile (String)
    - dateOfBirth (LocalDate)
    - status (Enum: ACTIVE, DISABLED)
    - roles (Set<Role>, Many-to-Many relationship with Role entity)
2.
3. **Role** (Helper entity for Employee roles)
    - id (Long, Primary Key)
    - name (Enum: ROLE_MANAGER, ROLE_ADMIN, ROLE_EMPLOYEE, Unique)
4.
5. **Employment**
    - id (Long, Primary Key)
    - code (String, Unique, for the employment record itself)
    - employee (Employee, Many-to-One relationship, Foreign Key: employee_id)
    - department (String)
    - position (String)
    - baseSalary (BigDecimal)
    - status (Enum: ACTIVE, INACTIVE)
    - joiningDate (LocalDate)
6.
7. **Deductions**
    - id (Long, Primary Key)
    - code (String, Unique)
    - deductionName (String, Unique - e.g., "Employee Tax", "Pension")
    - percentage (BigDecimal - e.g., 0.06 for 6%)
8.
9. **Payslip**
    - id (Long, Primary Key)

- employment (Employment, Many-to-One relationship, Foreign Key: employment_id - links to the specific active employment record used for this payslip)
- calculatedBaseSalary (BigDecimal - base salary snapshotted for this calculation)
- housingAllowanceAmount (BigDecimal)
- transportAllowanceAmount (BigDecimal)
- employeeTaxAmount (BigDecimal)
- pensionAmount (BigDecimal)
- medicalInsuranceAmount (BigDecimal)
- otherDeductionsAmount (BigDecimal)
- grossSalary (BigDecimal)
- netSalary (BigDecimal)
- month (Integer - 1-12)
- year (Integer - e.g., 2025)
- status (Enum: PENDING, PAID)
- generatedAt (LocalDateTime)
- paidAt (LocalDateTime, nullable)
- *(Implicit Unique Constraint on: employment_id, month, year)*

10.
11. **Message**
    - id (Long, Primary Key)
    - employee (Employee, Many-to-One relationship, Foreign Key: employee_id)
    - messageContent (String/Text)
    - monthYearPayslip (String - e.g., "12-2025", for easy reference to the payslip period)
    - createdAt (LocalDateTime - when the message was generated by the system/trigger)
    - emailStatus (String/Enum - e.g., PENDING_SEND, SENT, FAILED)
12.

Technology Stack + Justification

For developing the backend of this ERP payroll system, I propose the following modern Java-based technology stack:

Core Stack:

1. **Java (Version 17+):**
    - **Justification:** As a long-term support (LTS) version, Java 17 (or newer LTS like 21) offers modern language features (records, pattern matching, sealed classes, etc.), performance improvements, and a vast, mature ecosystem. It's robust, scalable, and well-suited for enterprise applications. My familiarity and the project's requirement for a Java developer make this a natural choice.
2.

3. **Spring Boot (Latest Stable, e.g., 3.2.x):**
    - **Justification:** Spring Boot significantly accelerates application development by providing auto-configuration, embedded servers (like Tomcat), and simplified dependency management. Its "convention over configuration" approach reduces boilerplate, allowing me to focus on business logic. The large community support and integration with other Spring projects (like Spring Data, Spring Security) make it ideal for building RESTful APIs and a well-structured backend.
4.
5. **Spring Data JPA (with Hibernate as the provider):**
    - **Justification:** Spring Data JPA abstracts away much of the boilerplate code typically associated with JDBC and ORM interactions. It allows for easy creation of repository interfaces with powerful query derivation capabilities. Hibernate, as the underlying JPA provider, is a mature and feature-rich ORM tool that handles object-relational mapping, caching, and lazy loading effectively, simplifying database persistence logic.
6.
7. **Spring Security (with JWT for Authentication/Authorization):**
    - **Justification:** Security is paramount, especially for an ERP system handling sensitive employee and payroll data. Spring Security is a comprehensive and highly customizable framework. Using JSON Web Tokens (JWT) provides a stateless authentication mechanism suitable for RESTful APIs, allowing clients (like a future frontend or mobile app) to authenticate once and then include the token in subsequent requests. This approach simplifies scalability and session management. Role-based access control (RBAC) can be easily implemented to meet the specified access requirements for different user types (Manager, Admin, Employee).
8.
9. **PostgreSQL (Latest Stable):**
    - **Justification:** PostgreSQL is a powerful, open-source, and highly extensible object-relational database system. It's known for its reliability, data integrity features, and support for complex queries, stored procedures, and triggers (as required for the messaging feature). Its robust ACID compliance is crucial for financial data. It also handles JSON and other advanced data types well, offering flexibility for future enhancements. (MySQL would also be a viable alternative, but PostgreSQL's feature set, particularly around advanced SQL and extensibility, often gives it an edge for complex applications.)
10.

API Documentation & Developer Experience:

1. **Swagger UI (via Springdoc OpenAPI):**
    - **Justification:** Springdoc OpenAPI automatically generates OpenAPI 3.0 documentation from Spring Boot controllers. The integrated Swagger UI provides an interactive interface for developers (and potentially testers or frontend teams)

to explore, understand, and test the API endpoints without needing to write client code. This greatly improves API discoverability and reduces integration time.

2.
3. **Lombok:**
    - **Justification:** Lombok is a library that helps reduce boilerplate code in Java classes by providing annotations (like @Data, @Getter, @Setter, @NoArgsConstructor, @AllArgsConstructor). This makes the entity and DTO classes cleaner, more readable, and easier to maintain, speeding up development.
4.

Communication:

1. **JavaMailSender (Spring Boot Mail Starter):**
    - **Justification:** For sending email notifications (e.g., when payroll is paid), the spring-boot-starter-mail provides easy integration with JavaMailSender. It simplifies the configuration and sending of emails, abstracting away low-level SMTP details.
2.

Development & Build Tools:

1. **Maven:**
    - **Justification:** Maven is a robust build automation and dependency management tool. It standardizes the project structure, build lifecycle, and handling of external libraries. Its pom.xml provides a clear way to declare project dependencies and plugins. (Gradle is a strong alternative offering more flexibility with its Groovy/Kotlin DSL, but Maven's XML-based configuration is often considered more straightforward for many Java developers).
2.
3. **Flyway (or Liquibase) for Database Migrations:**
    - **Justification:** As the database schema evolves (e.g., adding new tables, columns, or modifying the trigger), managing these changes in a version-controlled and automated way is crucial, especially across different environments (dev, test, prod). Flyway (or Liquibase) allows for writing SQL (or XML/YAML/JSON for Liquibase) migration scripts that are applied sequentially, ensuring database schema consistency and making rollbacks or updates manageable. This is essential for team collaboration and CI/CD pipelines. *For this 5-hour project, I might initially rely on Hibernate's ddl-auto, but for a real-world scenario, Flyway/Liquibase is a must.*
4.

Self-Justification as the Developer:

My choice of this stack is driven by a balance of **modernity, robustness, developer productivity, and the specific requirements of the project.**

- **Productivity:** Spring Boot, Spring Data JPA, and Lombok are chosen to maximize development speed by minimizing boilerplate and providing sensible defaults. This is critical given the 5-hour timeframe for the initial development sprint.
- **Robustness & Scalability:** Java, Spring, and PostgreSQL form a foundation that is well-tested in enterprise environments, capable of handling the data volumes and transaction loads expected of an ERP system. Spring Security ensures that the critical data is protected.
- **Maintainability:** A clear, well-structured codebase using established Spring patterns, along with tools like Flyway for schema management and Swagger for API documentation, will make the system easier to understand, maintain, and extend in the future.
- **Requirement Fulfillment:** Each technology directly addresses specific project requirements: Spring Security with JWT for authentication/authorization, Spring Data JPA for database design, PostgreSQL for its trigger capabilities, JavaMailSender for notifications, and Swagger for API documentation.
- **My Expertise:** This stack aligns well with common Java backend development practices, allowing me to leverage existing knowledge and community resources effectively to deliver a quality solution efficiently.

By using these proven technologies, I am confident in my ability to develop a reliable, secure, and maintainable backend system for the ERP's Payroll and Employee Management modules.