List of POJOs

The following Java classes are used as entities to represent the database tables:

1. **Employee**
    - Properties: code (Long, PK), firstName (String), lastName (String), email (String), password (String), roles (String), mobile (String), dateOfBirth (LocalDate), status (String)
    - Relationships: One-to-Many with Employment, Payslip, and Message
2. **Employment**
    - Properties: code (Long, PK), employee (Employee, FK), department (String), position (String), baseSalary (Double), status (String), joiningDate (LocalDate)
    - Relationships: Many-to-One with Employee
3. **Deduction**
    - Properties: code (Long, PK), deductionName (String), percentage (Double)
    - Relationships: None (standalone, used in Payslip calculations)
4. **Payslip**
    - Properties: id (Long, PK), employee (Employee, FK), houseAmount (Double), transportAmount (Double), employeeTaxedAmount (Double), pensionAmount (Double), medicalInsuranceAmount (Double), otherTaxedAmount (Double), grossSalary (Double), netSalary (Double), month (Integer), year (Integer), status (String)
    - Relationships: Many-to-One with Employee
5. **Message**
    - Properties: id (Long, PK), employee (Employee, FK), message (String), monthYear (String)
    - Relationships: Many-to-One with Employee

Justification of Technology Stack for ERP System

The technology stack for the ERP system was chosen to meet the requirements of a secure, scalable, and maintainable backend for managing employee data, payroll, deductions, and messaging for the Government of Rwanda. The stack includes Spring Boot, Spring Data JPA, Spring Security with JWT, MySQL, Swagger, and Lombok. Below is a detailed justification for each technology, updated to reflect the use of MySQL instead of H2.

1. Spring Boot

- **Why Chosen**:
    - **Rapid Development**: Spring Boot's auto-configuration and embedded server (Tomcat) streamline REST API development, enabling implementation of employee management, deductions, and payroll endpoints within the 5-hour timeline.
    - **Scalability**: Its microservices-friendly architecture supports scaling for additional ERP modules, suitable for a government system with thousands of employees.
    - **Ecosystem**: Seamlessly integrates with Spring Data JPA, Spring Security, and MySQL, providing a cohesive framework.
    - **Community and Support**: Extensive documentation and community support ensure maintainability.
- **Relevance to Project**: Handles REST API endpoints (e.g., `/api/employees`, `/api/payroll/generate`) for payroll and employee management, meeting the project's functional requirements.

2. Spring Data JPA

- **Why Chosen**:
    - **Simplified Database Operations**: Abstracts database access through repositories, reducing boilerplate code for CRUD operations on entities like `Employee`, `Employment`, `Deductions`, `Payslip`, and `Message`.
    - **ORM Capabilities**: Maps Java POJOs to MySQL tables, supporting complex relationships (e.g., One-to-Many between `Employee` and `Payslip`) as defined in the ERD.
    - **Database Agnostic**: Works seamlessly with MySQL via the MySQL Connector/J, ensuring compatibility and ease of configuration.
- **Relevance to Project**: Enables efficient database design and querying for payroll calculations and employee management, fulfilling the requirement to use Spring Data JPA for database configuration.

3. Spring Security with JWT

- **Why Chosen**:

    - **Role-Based Security**: Supports fine-grained access control for `ROLE_MANAGER`, `ROLE_ADMIN`, and `ROLE_EMPLOYEE`, ensuring only authorized users access specific endpoints (e.g., admins approving payroll).
    - **JWT for Authentication**: Provides stateless, secure authentication using email and password, as required for employee logins.
    - **Scalability**: JWT's stateless nature improves performance for a system with many users.
- **Relevance to Project**: Secures APIs, preventing unauthorized actions and meeting the authentication/authorization requirements with JWT.

4. MySQL

- **Why Chosen**:
    - **Relational Database**: MySQL is a robust, widely-used relational database that supports complex queries and relationships, ideal for managing `Employee`, `Employment`, `Deductions`, `Payslip`, and `Message` tables.
    - **Trigger Support**: MySQL supports triggers, enabling the messaging feature where a database trigger generates messages when payslips are approved (status changes from `pending` to `paid`).
    - **Scalability and Performance**: Handles large datasets efficiently, suitable for a government system with thousands of employees and payslips.
    - **Production-Ready**: Unlike H2, MySQL is suitable for both development and production, ensuring continuity as the system scales.
    - **Community and Tools**: Offers extensive tooling (e.g., MySQL Workbench) for manual data entry and inspection, aligning with the requirement for data addition via the DBMS.
- **Relevance to Project**: Provides a reliable database for storing and querying ERP data, supports the trigger-based messaging feature, and meets the need for manual data entry via tools like MySQL Workbench or Postman.

5. Swagger (Springdoc OpenAPI)

- **Why Chosen**:
    - **API Documentation**: Generates interactive API documentation for endpoints like `/api/employees`, `/api/deductions`, and `/api/payroll`, accessible via `/swagger-ui.html`.
    - **Developer Productivity**: Allows testing APIs directly in the browser, supporting the requirement for Postman/Swagger-based testing.
    - **Standardization**: Follows OpenAPI standards, ensuring modern API practices.
- **Relevance to Project**: Fulfills the requirement to document APIs using Swagger UI, enabling easy testing and validation.

6. Lombok

- **Why Chosen**:
  - **Code Reduction**: Annotations like `@Getter` and `@Setter` reduce boilerplate code in POJOs, speeding up development of entities like `Employee` and `Payslip`.
  - **Maintainability**: Improves code readability and maintenance.
- **Relevance to Project**: Accelerates development within the 5-hour timeframe by minimizing repetitive code.

Conclusion

The stack—Spring Boot, Spring Data JPA, Spring Security with JWT, MySQL, Swagger, and Lombok—ensures a secure, scalable, and maintainable backend that meets all project requirements, including role-based access, payroll calculations, messaging via triggers, and API documentation. MySQL's robustness and trigger support make it a superior choice over H2 for this system, supporting both development and production environments.