HRMS + Payroll Management — Low Level Design (LLD)

## **Tech Stack**

Java 21, Spring Boot 3.x, Spring Data JPA, Spring Validation, Spring Security (Resource Server, OIDC), Keycloak, PostgreSQL, Apache Kafka, Apache Camel, MapStruct, Flyway, OpenAPI/Swagger, Testcontainers, JUnit 5.

# 1. Overall Architecture & Packaging

## **Architecture style**

Modular microservices with synchronous REST for CRUD and async events via Kafka. Each service has its own DB schema (per-service database).

**Services**

* **auth-gateway** (Keycloak, API Gateway/Ingress)
* **employee-service** (Core HR, profiles, departments)
* **attendance-service** (device ingestion, timesheets)
* **leave-service** (entitlements, accruals, approvals)
* **payroll-service** (salary structure, payroll engine, payslips)
* **notification-service** (email/SMS/WhatsApp provider integration)
* **reporting-service** (PDF/Excel aggregation)

**Common libraries**

* **hrms-common**: DTOs (shared), error model, event contracts, security utils.

**Recommended package structure (per service)**

com.acme.hrms.<service>

├─ api # controllers + request/response models

├─ app # application config, runners

├─ domain # JPA entities, domain services, repositories

│ ├─ entity

│ ├─ repository

│ └─ service

├─ integration # kafka producers/consumers, camel routes, external clients

├─ mapper # MapStruct mappers

├─ security # resource server config, method security

├─ util

└─ validation # custom validators

**Database per service:** PostgreSQL schemas employee, attendance, leave, payroll, notification.

**2. Domain Model (Key Entities)**

**2.1 Employee & Org**

* **Department**(id, code, name, description)
* **Employee**(id, empCode, firstName, lastName, email, phone, doj, status, deptId, managerId, grade, band)
* **EmploymentDetails**(id, employeeId, designation, location, costCenter)

**2.2 Attendance & Leave**

* **AttendanceRecord**(id, employeeId, date, inTime, outTime, workHours, status)
* **LeavePolicy**(id, type, accrualRatePerMonth, carryForwardLimit, encashable)
* **LeaveBalance**(id, employeeId, type, opening, accrued, consumed, closing)
* **LeaveRequest**(id, employeeId, type, startDate, endDate, days, status, approverId, remarks)

**2.3 Payroll**

* **PayComponent**(id, code, name, type=EARNING/DEDUCTION, calcMethod=FLAT/PERCENT/FORMULA, taxExemptFlag)
* **SalaryStructure**(id, employeeId, effectiveFrom, effectiveTo, components: List)
* **SalaryComponentLine**(id, structureId, componentId, amount, percentOfCode, priority)
* **PayrollRun**(id, periodMonth, periodYear, status, scheduledBy, runAt, closedAt)
* **Payslip**(id, employeeId, payrollRunId, gross, totalDeductions, net, currency)
* **PayslipLine**(id, payslipId, componentCode, componentName, type, amount)
* **TaxSlab**(id, country, fyStart, fyEnd, slabMin, slabMax, rate, cessRate)
* **BankInstruction**(id, payrollRunId, bankFilePath, status)

**3. Class Diagrams**

**A screenshot of a computer

AI-generated content may be incorrect.**

**4. Persistence Layer (Spring Data JPA)**

**A screenshot of a computer

AI-generated content may be incorrect.**

**5. Example JPA Entities (payroll-service)**

@Entity @Table(name = "pay\_component")

public class PayComponent {

@Id @GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable=false, unique=true) private String code;

@Column(nullable=false) private String name;

@Enumerated(EnumType.STRING) private ComponentType type;

@Enumerated(EnumType.STRING) private CalcMethod calcMethod;

private boolean taxExempt;

}

@Entity @Table(name = "salary\_structure")

public class SalaryStructure {

@Id @GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable=false) private Long employeeId;

private LocalDate effectiveFrom;

private LocalDate effectiveTo;

@OneToMany(mappedBy="structure", cascade=CascadeType.ALL, orphanRemoval=true)

private List<SalaryComponentLine> lines = new ArrayList<>();

}

@Entity @Table(name = "salary\_component\_line")

public class SalaryComponentLine {

@Id @GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name="structure\_id")

private SalaryStructure structure;

private String componentCode;

private BigDecimal amount;

private String percentOfCode; // for PERCENT calc

private int priority; // evaluation order

}

**Repositories**

public interface PayComponentRepository extends JpaRepository<PayComponent, Long> {

Optional<PayComponent> findByCode(String code);

}

public interface SalaryStructureRepository extends JpaRepository<SalaryStructure, Long> {

Optional<SalaryStructure> findFirstByEmployeeIdAndEffectiveFromLessThanEqualAndEffectiveToGreaterThanEqual(Long empId, LocalDate from, LocalDate to);

}

public interface PayrollRunRepository extends JpaRepository<PayrollRun, Long> {

Optional<PayrollRun> findByYearAndMonth(int year, int month);

}

**6. Flyway migration (snippet)**

CREATE TABLE pay\_component (

id BIGSERIAL PRIMARY KEY,

code VARCHAR(50) UNIQUE NOT NULL,

name VARCHAR(100) NOT NULL,

type VARCHAR(20) NOT NULL,

calc\_method VARCHAR(20) NOT NULL,

tax\_exempt BOOLEAN DEFAULT FALSE

);

**5. DTOs & Mapping (MapStruct)**

@Mapper(componentModel = "spring")

public interface PayComponentMapper {

PayComponentDto toDto(PayComponent entity);

PayComponent toEntity(PayComponentDto dto);

}

**Example DTO**

public record PayComponentDto(Long id, String code, String name, String type, String calcMethod, boolean taxExempt) {}

**6. Application Services (Domain Logic)**

**6.1 Payroll Engine**

**Responsibilities**

* Resolve employee salary structure and components (effective date within pay period)
* Pull attendance and approved leave from respective services
* Evaluate components in priority order
* Calculate taxes from slabs and exemptions
* Generate payslip + lines, persist, and publish events

**Interfaces**

public interface PayrollEngine {

Payslip computeForEmployee(Long employeeId, YearMonth period);

PayrollRunResult runForPeriod(YearMonth period, Set<Long> employeeIds);

}

**Computation Strategy**

* CalcMethod.FLAT: amount as-is
* CalcMethod.PERCENT: amount = percentOfCode.amount \* (amount/100)
* CalcMethod.FORMULA: evaluate via expression engine (MVEL / JEXL) with context {basic, hra, lta, ...}

**Service Skeleton**

@Service

@RequiredArgsConstructor

public class PayrollEngineImpl implements PayrollEngine {

private final SalaryStructureRepository structureRepo;

private final AttendanceClient attendanceClient;

private final LeaveClient leaveClient;

private final TaxService taxService;

private final PayslipService payslipService;

@Transactional

public Payslip computeForEmployee(Long empId, YearMonth period) {

var structure = structureRepo.findActive(empId, period);

var context = PayrollContext.load(structure, attendanceClient, leaveClient, empId, period);

var lines = ComponentEvaluator.evaluateAll(context);

var tax = taxService.computeTax(empId, period, lines);

return payslipService.persist(empId, period, lines, tax);

}

}

**6.2 Tax Service**

public interface TaxService { BigDecimal computeTax(Long empId, YearMonth period, List<PayslipLine> lines); }

Implementation can be country-specific via Spring profiles (tax.in, tax.us).

**7. Integrations**

**7.1 Kafka Topics (Event Contracts)**

* employee.v1.created
* employee.v1.updated
* attendance.v1.daily\_aggregated
* leave.v1.approved
* payroll.v1.payslip\_generated
* payroll.v1.run\_completed

**Event Example (JSON Schema)**

{

"$schema": "http://json-schema.org/draft-07/schema#",

"title": "PayslipGenerated",

"type": "object",

"properties": {

"eventId": {"type": "string"},

"occurredAt": {"type": "string", "format": "date-time"},

"employeeId": {"type": "integer"},

"payrollRunId": {"type": "integer"},

"period": {"type": "string", "pattern": "^\\d{4}-\\d{2}$"},

"net": {"type": "number"}

},

"required": ["eventId","occurredAt","employeeId","payrollRunId","period","net"]

}

**7.2 Apache Camel Routes (attendance ingestion)**

from("kafka:attendance.raw")

.routeId("attendance-normalize")

.unmarshal().json(JsonLibrary.Jackson, RawAttendance.class)

.process(new AttendanceNormalizer())

.to("direct:aggregate-daily");

from("direct:aggregate-daily")

.bean(AttendanceAggregator.class, "aggregateDaily")

.marshal().json()

.to("kafka:attendance.v1.daily\_aggregated");

7.3 Swagger

package com.example.hrms.config;

import io.swagger.v3.oas.models.OpenAPI;

import io.swagger.v3.oas.models.info.Info;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class OpenApiConfig {

@Bean

public OpenAPI hrmsOpenAPI() {

return new OpenAPI()

.info(new Info()

.title("HRMS & Payroll API")

.version("1.0.0")

.description("Human Resource Management System with Payroll features"));

}

}

EmployeeController.java:

package com.example.hrms.controller;

import com.example.hrms.dto.EmployeeDto;

import com.example.hrms.service.EmployeeService;

import io.swagger.v3.oas.annotations.Operation;

import io.swagger.v3.oas.annotations.tags.Tag;

import lombok.RequiredArgsConstructor;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api/v1/employees")

@RequiredArgsConstructor

@Tag(name = "Employee", description = "Employee management APIs")

public class EmployeeController {

private final EmployeeService employeeService;

@GetMapping

@Operation(summary = "List all employees")

public ResponseEntity<List<EmployeeDto>> getAllEmployees() {

return ResponseEntity.ok(employeeService.getAllEmployees());

}

@PostMapping

@Operation(summary = "Create a new employee")

public ResponseEntity<EmployeeDto> createEmployee(@RequestBody EmployeeDto dto) {

return ResponseEntity.ok(employeeService.createEmployee(dto));

}

}

OpenApiConfig.java

package com.example.hrms.config;

import io.swagger.v3.oas.models.OpenAPI;

import io.swagger.v3.oas.models.info.Info;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class OpenApiConfig {

@Bean

public OpenAPI hrmsOpenAPI() {

return new OpenAPI()

.info(new Info()

.title("HRMS & Payroll API")

.version("1.0.0")

.description("Human Resource Management System with Payroll features"));

}

}

**10. Operational & non-functional notes**

* Batching: Process employees in chunks (e.g., 500) to bound memory usage.
* Parallelism: Use a bounded thread-pool to process chunks concurrently.
* Observability: Emit metrics for pay slips\_ generated, errors, time per employee.
* Retries: Retry transient failures; on persistent failures mark employee payslip status=FAILED and continue.
* Backups & audit: Keep payslip historical records immutable; store audit trail for run operations.

**11. Testing checklist**

* Unit tests for Payroll Engine (edge cases: zero basic, percent loops, formula errors).
* Integration tests using Test containers (Postgres + Kafka).
* Contract tests for event schemas.
* Load test a payroll run of 10k employees.

**12. Next steps (deliverables)**

* Provide starter Spring Boot module for payroll-service (skeleton project).
* Flyway migrations for critical tables.
* Example Payroll Engine tests and sample payslip JSON.

Review

|  |  |  |
| --- | --- | --- |
| Kaushik Nandy | created | V1.0 |
|  |  |  |
|  |  |  |