**HRMS-Payroll-System LLD**

**1) Module Decomposition**

* **Payroll Core**
  + PayrollRunService, PayrollCalcEngine, EarningsService, DeductionsService, TaxEngine, ProrationService, RetroService
* **Master Data**
  + EmployeeService, OrgService, CompensationService, BankAccountService, BenefitService
* **Time Data Integrations**
  + AttendanceAdapter (REST/SFTP), LeaveAdapter, OvertimeAdapter
* **Approvals & Workflow**
  + PayrollApprovalService (multi-step approvals, re-run rules)
* **Disbursement**
  + PaymentFileService (bank file), BankApiClient (API), ReconciliationService
* **Document & Reporting**
  + PayslipService (PDF), ComplianceReportService (PF/ESI/Tax), LedgerExportService
* **Schedulers & Batches**
  + PayrollScheduler, CutoffScheduler, ComplianceScheduler
* **Messaging**
  + Kafka producers/consumers for employee-events, attendance-events, payroll-events, payment-events
* **Security**
  + Keycloak/OAuth2 resource server, RBAC with ROLE\_HR, ROLE\_PAYROLL, ROLE\_FINANCE, ROLE\_EMPLOYEE
* **Observability**
  + Metrics (Micrometer), structured logs, audit trail

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

@startuml

class Employee {

UUID id

String empCode

String firstName

String lastName

LocalDate doj

String employmentType // FT, PT, Contractor

UUID orgUnitId

UUID compPackageId

UUID bankAccountId

boolean active

}

class CompensationPackage {

UUID id

Money basePayMonthly

List<Component> components // HRA, Bonus, Allowances

String payFrequency // MONTHLY, BIWEEKLY

LocalDate effectiveFrom

LocalDate effectiveTo

}

class Component {

UUID id

String code

String name

ComponentType type // EARNING, DEDUCTION, BENEFIT, TAX

CalcRule rule

}

class CalcRule {

UUID id

String expression // DSL/SpEL e.g. "0.4 \* BASE" for HRA

String basis // BASE, GROSS, CUSTOM

RoundingMode roundingMode

}

class AttendanceSummary {

UUID id

UUID employeeId

YearMonth period

int workingDays

int presentDays

int unpaidLeaveDays

int paidLeaveDays

double overtimeHours

}

class PayrollRun {

UUID id

YearMonth period

PayrollRunType type // REGULAR, OFFCYCLE, BONUS, RETRO

PayrollRunStatus status // DRAFT, READY, APPROVAL\_PENDING, APPROVED, CALCULATING, COMPLETED, FAILED

LocalDateTime createdAt

LocalDateTime closedAt

}

class PayrollLineItem {

UUID id

UUID runId

UUID employeeId

Map<String, Money> earnings // key=componentCode

Map<String, Money> deductions

Money gross

Money taxableIncome

Money taxes

Money netPay

boolean locked

}

class PaymentInstruction {

UUID id

UUID runId

UUID employeeId

Money amount

String currency

String mode // NEFT, RTGS, ACH

String bankRef

PaymentStatus status // PENDING, SENT, CONFIRMED, FAILED

}

class Payslip {

UUID id

UUID runId

UUID employeeId

String pdfPath

String hash

}

Employee "1" -- "1" BankAccount

Employee "1" -- "1" CompensationPackage

PayrollRun "1" -- "many" PayrollLineItem

PayrollLineItem "1" -- "1" AttendanceSummary

PayrollRun "1" -- "many" PaymentInstruction

PayrollLineItem "1" -- "1" Payslip

@enduml

**3) Service Classes & Method Signatures (Java/Spring Boot style)**

interface PayrollRunService {

PayrollRun createRun(YearMonth period, PayrollRunType type, List<UUID> employeeIds);

void lockRun(UUID runId);

void submitForApproval(UUID runId);

void approve(UUID runId, UUID approverId);

void calculate(UUID runId); // orchestrates engine

void complete(UUID runId);

}

interface PayrollCalcEngine {

PayrollLineItem calculateForEmployee(UUID runId, UUID employeeId, YearMonth period);

}

interface EarningsService {

Map<String, Money> computeEarnings(Employee emp, CompensationPackage pkg, AttendanceSummary att, YearMonth period);

}

interface DeductionsService {

Map<String, Money> computeDeductions(Employee emp, AttendanceSummary att, YearMonth period);

}

interface TaxEngine {

Money computeTaxes(Employee emp, Map<String, Money> earnings, Map<String, Money> deductions, YearMonth period);

}

interface ProrationService {

Money prorate(Money monthlyAmount, int presentDays, int workingDays);

}

interface RetroService {

PayrollLineItem computeRetro(UUID employeeId, YearMonth from, YearMonth to);

}

interface PaymentFileService {

Path generateBankFile(UUID runId, String bankFormat); // e.g., XML/CSV

}

interface BankApiClient {

void sendBatch(UUID runId, Path bankFile);

PaymentStatus queryStatus(String bankBatchRef);

}

interface PayslipService {

Path generatePayslip(UUID runId, UUID employeeId);

}

interface AttendanceAdapter {

AttendanceSummary fetch(UUID employeeId, YearMonth period);

}

**4) Calculation Logic (Deterministic Order of Steps)**

1. **Collect Inputs**: Employee, Compensation, AttendanceSummary, Prior YTD values.
2. **Earnings Calculation** (base, allowances, OT, bonus):
   * Apply proration for joiners/leavers mid-period.
   * Apply component rules (CalcRule.expression) with a deterministic eval engine (DSL).
3. **Pre-Tax Deductions** (benefits, retirement contributions).
4. **Taxable Income** = Earnings − Pre-Tax Deductions ± Retro adjustments.
5. **Tax Calculation**:
   * Slab-based progressive rates.
   * Standard deductions, exemptions, cess.
   * Consider YTD to avoid over/under-withholding.
6. **Post-Tax Deductions** (loans, garnishments).
7. **Net Pay** = Earnings − Deductions − Taxes.
8. **Rounding** as per policy (per-component or final).
9. **Lock Line Item** and persist audit trail.

**Pseudo-code**

for emp in run.employees:

att = AttendanceAdapter.fetch(emp.id, period)

pkg = CompensationService.getPackage(emp.id, period)

earnings = EarningsService.compute(emp, pkg, att, period)

pretax = DeductionsService.computePreTax(emp, att, period)

taxable = sum(earnings) - sum(pretax) + RetroService.delta(emp, period)

taxes = TaxEngine.compute(emp, earnings, pretax, period)

posttax = DeductionsService.computePostTax(emp, period)

net = sum(earnings) - sum(pretax) - taxes - sum(posttax)

line = persistLineItem(earnings, pretax, posttax, taxes, net)

PayslipService.queue(line)

**5) API Design (REST)**

**Auth**: OAuth2 (Bearer), scopes by role.

* POST /api/payroll/runs
  + body: { "period":"2025-07", "type":"REGULAR", "employeeIds":[...] }
  + resp: 201 { "runId": "<uuid>", "status": "DRAFT" }
* POST /api/payroll/runs/{runId}/submit
* POST /api/payroll/runs/{runId}/approve
* POST /api/payroll/runs/{runId}/calculate
* GET /api/payroll/runs/{runId}
* GET /api/payroll/runs/{runId}/lines?empId=&page=&size=
* GET /api/payroll/runs/{runId}/summary
* POST /api/payroll/runs/{runId}/disburse?mode=FILE|API&bankFormat=CSV|XML
* GET /api/payroll/runs/{runId}/payments
* GET /api/employees/{id}/payslips?from=YYYY-MM&to=YYYY-MM
* GET /api/payslips/{payslipId}/download

**Error Model**

{

"timestamp": "2025-08-14T10:00:00Z",

"errorCode": "ATTENDANCE\_NOT\_FOUND",

"message": "No attendance for employee 123 in 2025-07",

"traceId": "c7e7-..."

}

**6) Database Schema (PostgreSQL DDL)**

CREATE TABLE employees (

id UUID PRIMARY KEY,

emp\_code VARCHAR(32) UNIQUE NOT NULL,

first\_name VARCHAR(64) NOT NULL,

last\_name VARCHAR(64),

doj DATE NOT NULL,

employment\_type VARCHAR(16) NOT NULL,

org\_unit\_id UUID,

comp\_package\_id UUID,

bank\_account\_id UUID,

active BOOLEAN DEFAULT TRUE

);

CREATE TABLE compensation\_packages (

id UUID PRIMARY KEY,

employee\_id UUID NOT NULL REFERENCES employees(id),

base\_pay\_monthly NUMERIC(12,2) NOT NULL,

pay\_frequency VARCHAR(16) NOT NULL,

effective\_from DATE NOT NULL,

effective\_to DATE

);

CREATE TABLE components (

id UUID PRIMARY KEY,

code VARCHAR(32) UNIQUE NOT NULL,

name VARCHAR(64) NOT NULL,

type VARCHAR(16) NOT NULL, -- EARNING/DEDUCTION/BENEFIT/TAX

rule\_expression TEXT NOT NULL,

basis VARCHAR(16),

rounding\_mode VARCHAR(16)

);

CREATE TABLE package\_components (

package\_id UUID REFERENCES compensation\_packages(id),

component\_id UUID REFERENCES components(id),

PRIMARY KEY (package\_id, component\_id)

);

CREATE TABLE attendance\_summaries (

id UUID PRIMARY KEY,

employee\_id UUID NOT NULL REFERENCES employees(id),

period CHAR(7) NOT NULL, -- YYYY-MM

working\_days INT NOT NULL,

present\_days INT NOT NULL,

unpaid\_leave\_days INT NOT NULL,

paid\_leave\_days INT NOT NULL,

overtime\_hours NUMERIC(8,2) DEFAULT 0,

UNIQUE (employee\_id, period)

);

CREATE TABLE payroll\_runs (

id UUID PRIMARY KEY,

period CHAR(7) NOT NULL,

type VARCHAR(16) NOT NULL,

status VARCHAR(24) NOT NULL,

created\_at TIMESTAMP NOT NULL DEFAULT now(),

closed\_at TIMESTAMP

);

CREATE TABLE payroll\_line\_items (

id UUID PRIMARY KEY,

run\_id UUID NOT NULL REFERENCES payroll\_runs(id) ON DELETE CASCADE,

employee\_id UUID NOT NULL REFERENCES employees(id),

earnings JSONB NOT NULL,

deductions JSONB NOT NULL,

gross NUMERIC(12,2) NOT NULL,

taxable\_income NUMERIC(12,2) NOT NULL,

taxes NUMERIC(12,2) NOT NULL,

net\_pay NUMERIC(12,2) NOT NULL,

locked BOOLEAN DEFAULT FALSE

);

CREATE INDEX idx\_line\_items\_run ON payroll\_line\_items(run\_id);

CREATE INDEX idx\_line\_items\_emp ON payroll\_line\_items(employee\_id);

CREATE TABLE payment\_instructions (

id UUID PRIMARY KEY,

run\_id UUID NOT NULL REFERENCES payroll\_runs(id) ON DELETE CASCADE,

employee\_id UUID NOT NULL REFERENCES employees(id),

amount NUMERIC(12,2) NOT NULL,

currency CHAR(3) NOT NULL DEFAULT 'INR',

mode VARCHAR(8) NOT NULL,

bank\_ref VARCHAR(64),

status VARCHAR(16) NOT NULL DEFAULT 'PENDING'

);

CREATE TABLE payslips (

id UUID PRIMARY KEY,

run\_id UUID NOT NULL REFERENCES payroll\_runs(id) ON DELETE CASCADE,

employee\_id UUID NOT NULL REFERENCES employees(id),

pdf\_path TEXT NOT NULL,

sha256\_hash CHAR(64) NOT NULL

);

CREATE TABLE audit\_logs (

id BIGSERIAL PRIMARY KEY,

actor UUID,

action VARCHAR(64),

entity VARCHAR(64),

entity\_id UUID,

details JSONB,

created\_at TIMESTAMP DEFAULT now()

);

**7) Sequence Diagrams**

**Regular Payroll Run**

@startuml

actor PayrollAdmin

participant "PayrollRunService" as PRS

participant "PayrollCalcEngine" as PCE

participant "AttendanceAdapter" as ATT

database "DB" as DB

participant "PayslipService" as PS

PayrollAdmin -> PRS: createRun(period, type, empIds)

PRS -> DB: insert payroll\_runs + assignments

PayrollAdmin -> PRS: submitForApproval(runId)

PRS -> DB: update status=APPROVAL\_PENDING

PayrollAdmin -> PRS: approve(runId)

PRS -> DB: status=APPROVED

PayrollAdmin -> PRS: calculate(runId)

loop for each employee

PRS -> ATT: fetch(empId, period)

PRS -> PCE: calculateForEmployee(runId, empId, period)

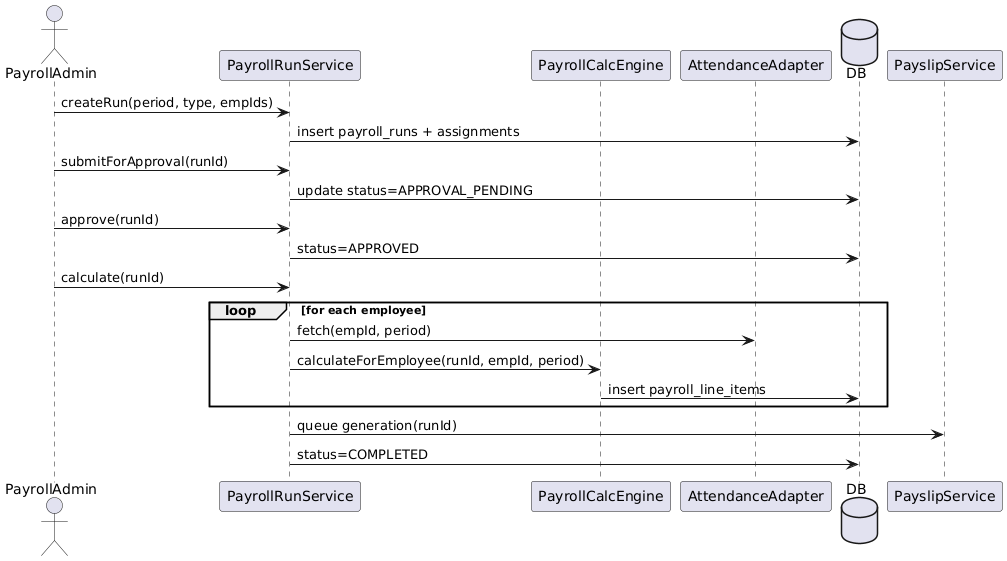
PCE -> DB: insert payroll\_line\_items

end

PRS -> PS: queue generation(runId)

PRS -> DB: status=COMPLETED

@enduml



**Disbursement (Bank File)**

@startuml

participant "PaymentFileService" as PFS

database DB

participant "BankApiClient" as BAC

PFS -> DB: fetch net pays by runId

PFS -> PFS: build CSV/XML per bank format

PFS -> BAC: sendBatch(file)

BAC -> PFS: bankBatchRef

PFS -> DB: insert payment\_instructions(status=SENT)

@enduml

A diagram of a service

AI-generated content may be incorrect.

**8) Rules & DSL**

* Store rule expressions as text (SpEL or MVEL or custom).
* Example HRA rule:
  + expression = "min(0.4 \* BASE, ACTUAL\_RENT\_PAID - 0.1 \* BASE)"
* Maintain **RuleVersion** with effective\_from, effective\_to.
* Provide RuleContext inputs: BASE, PRESENT\_DAYS, WORKING\_DAYS, OT\_HOURS, YTD\_TAXABLE.

**Evaluation Safety**

* Whitelist functions: min, max, round.
* Sandboxed evaluation, timeouts, numeric bounds.

**9) Messaging (Kafka)**

* Topics:
  + employee-events (created/updated/terminated)
  + attendance-events (period-closed)
  + payroll-events (run-created, run-completed, payslip-ready)
  + payment-events (batch-sent, payment-confirmed, payment-failed)
* **Retry/DLQ**:
  + Consumers use exponential backoff.
  + DLQ topics: <topic>.dlq with headers: errorCode, stacktraceHash, payloadHash.

**10) Validation & Edge Cases**

* **Joiners/Leavers mid-month**: proration by present/working days.
* **Unpaid Leaves**: reduce base proportionally.
* **Negative Net Pay**: cap deductions or carry-forward negative balance.
* **Retro Adjustments**: detect component/rule changes after prior run → compute delta.
* **Rounding**: per-component vs final-pay rounding (bank-compliant).
* **Currency**: multi-currency with FX locking at cut-off (if multi-country).
* **Duplicate Disbursement**: idempotency keys per run + employee.

**11) Security**

* OAuth2 Resource Server (Keycloak).
* Scopes:
  + payroll:run:\* (HR/Payroll), payroll:view, finance:disburse, employee:self.
* Record every action in audit\_logs.
* Encrypt PII columns at rest (pgcrypto or app-level).
* Mask sensitive fields in logs.

**12) Observability**

* **Metrics**:
  + payroll\_run\_duration\_seconds, line\_calc\_latency\_ms, payslips\_generated\_total,
  + bank\_batch\_failures\_total, tax\_engine\_errors\_total.
* **Tracing**: traceId per request; propagate into Kafka headers.
* **Logs**: JSON, include runId, empId, period.

**13) Performance & Concurrency**

* Parallel calculation with thread pool / Reactor:
  + Batch employees in chunks (e.g., 500).
* DB tuning:
  + Use COPY for bulk inserts of line items.
  + Partition payroll\_line\_items by period (optional).
* Cache static rules and components.

**14) Test Strategy (Outline)**

* **Unit**:
  + Rule evaluation (boundary values).
  + Proration logic.
  + Tax slabs (table-driven tests).
* **Integration**:
  + Attendance adapter (mock APIs).
  + Bank client (sandbox).
  + PDF generation (checksum/hash).
* **E2E**:
  + Create→Approve→Calculate→Disburse→Payslip.
* **Property-Based**:
  + Randomized compensation/attendance to ensure invariants (gross >= net, etc.).
* **Regression**:
  + Golden-master payslip snapshots per employee archetype.

**15) Sample Component Rules (Table)**

| **Code** | **Type** | **Basis** | **Expression** | **Rounding** |
| --- | --- | --- | --- | --- |
| BASE | EARNING | N/A | fixed monthly | NEAREST |
| HRA | EARNING | BASE | min(0.4 \* BASE, RENT - 0.1 \* BASE) | DOWN |
| PF | DEDUCTION | BASE | min(0.12 \* BASE, 1800) | DOWN |
| PTAX | TAX | GROSS | slab(GROSS) | NEAREST |

**16) Example Payslip Layout (Fields)**

* Employee header (Name, EmpCode, PAN), Period.
* Earnings breakdown (amount, YTD).
* Deductions breakdown (amount, YTD).
* Taxes, Net Pay (words + number), Bank details (masked).
* QR or hash for integrity.

**17) Configuration & Feature Flags**

* cutoff.day=25
* rounding.mode=NEAREST
* bank.format=CSV|XML
* calc.parallelism=8
* Feature flags: enable.retro, enable.bank.api, enable.multi.currency

**18) Failure Handling**

* **Calc Errors**: mark line FAILED, continue; report summary.
* **Bank API Failure**: retry with backoff; on persistent failure → switch to file mode.
* **PDF Generation Failure**: queue retry, DLQ after N attempts.