**LMS Documentation high level**

**1 — Summary / Purpose**

A Leave Management System (LMS) lets employees request leave, managers approve/reject, HR track balances and reports, and the system handle accruals, validations, notifications and audit logs.

**2 — Key features**

* Employee portal: request leave, view balances & history
* Manager portal: approve/reject requests, team view
* HR/Admin: configure leave types, policies, accrual rules, reports
* Notifications: email/SMS/Slack on request/decision
* Audit & history: immutable log of actions
* Integrations: LDAP/Keycloak for auth, payroll export, calendar sync
* Offline/async: events via Kafka for notifications/audit/export

**3 — Non-functional & constraints**

* Tech stack: Java 21, Spring Boot, PostgreSQL, Apache Camel (for integration), Kafka (events), Keycloak (auth), Docker Compose for local dev
* Scalability: horizontally scale API nodes, Kafka for async
* Security: OAuth2/OIDC via Keycloak, RBAC (employee/manager/hr/admin)
* Consistency: transactional for leave balance updates; use DB transactions + optimistic locking

**4 — High-level architecture**

Components:

* **Frontend** (React/Vue) — calls API Gateway
* **API Service** (Spring Boot) — REST + GraphQL optional
* **Auth** (Keycloak) — OAuth2/OIDC
* **DB** (Postgres) — primary data store
* **Kafka** — events (leave.requested, leave.approved, leave.rejected, leave.cancelled)
* **Integration** (Apache Camel routes) — sends notifications, calendar updates, payroll export
* **Notification Service** — SMS/Email worker (subscribes to Kafka)
* **Scheduler** — daily accrual jobs, reminders (implements using Spring Scheduler or Quartz)

Flow (example): Employee submits request → API writes request & updates balances (transaction) → publish leave.requested to Kafka → Notification Service consumes and emails manager → Manager approves via API → update status, publish leave.approved → Notification & payroll listeners react.

**5 — Database schema (core tables) — PostgreSQL DDL**

sql

CopyEdit

-- Leave types (e.g., Annual, Sick, Casual)

CREATE TABLE leave\_type (

id SERIAL PRIMARY KEY,

code VARCHAR(50) UNIQUE NOT NULL,

name VARCHAR(100) NOT NULL,

description TEXT,

default\_entitlement NUMERIC(5,2) NOT NULL, -- days per year

accrual BOOLEAN DEFAULT FALSE,

carryover\_allowed BOOLEAN DEFAULT FALSE,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT now()

);

-- Employee basic info (link to identity provider via external\_id)

CREATE TABLE employee (

id BIGSERIAL PRIMARY KEY,

external\_id VARCHAR(128) UNIQUE, -- e.g., Keycloak id

employee\_number VARCHAR(64) UNIQUE,

first\_name VARCHAR(100) NOT NULL,

last\_name VARCHAR(100) NOT NULL,

email VARCHAR(150) UNIQUE NOT NULL,

manager\_id BIGINT REFERENCES employee(id),

joined\_at DATE,

active BOOLEAN DEFAULT TRUE,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT now()

);

-- Leave request

CREATE TYPE leave\_status AS ENUM ('PENDING','APPROVED','REJECTED','CANCELLED','WITHDRAWN');

CREATE TABLE leave\_request (

id BIGSERIAL PRIMARY KEY,

employee\_id BIGINT NOT NULL REFERENCES employee(id),

leave\_type\_id INT NOT NULL REFERENCES leave\_type(id),

start\_date DATE NOT NULL,

end\_date DATE NOT NULL,

start\_half BOOLEAN DEFAULT FALSE, -- morning/afternoon example

end\_half BOOLEAN DEFAULT FALSE,

days NUMERIC(6,2) NOT NULL,

reason TEXT,

status leave\_status DEFAULT 'PENDING',

requested\_at TIMESTAMP WITH TIME ZONE DEFAULT now(),

decided\_by BIGINT REFERENCES employee(id),

decided\_at TIMESTAMP WITH TIME ZONE,

decision\_comment TEXT,

tenant\_id VARCHAR(64) DEFAULT 'default', -- optional multi-tenant

version BIGINT DEFAULT 0, -- optimistic locking

CONSTRAINT chk\_dates CHECK (start\_date <= end\_date)

);

-- Employee leave balance (for each leave type, year)

CREATE TABLE leave\_balance (

id BIGSERIAL PRIMARY KEY,

employee\_id BIGINT NOT NULL REFERENCES employee(id),

leave\_type\_id INT NOT NULL REFERENCES leave\_type(id),

year INT NOT NULL,

entitlement NUMERIC(6,2) NOT NULL,

used NUMERIC(6,2) DEFAULT 0,

available NUMERIC(6,2) GENERATED ALWAYS AS (entitlement - used) STORED,

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT now(),

UNIQUE (employee\_id, leave\_type\_id, year)

);

-- Audit log

CREATE TABLE audit\_log (

id BIGSERIAL PRIMARY KEY,

entity\_type VARCHAR(80),

entity\_id BIGINT,

action VARCHAR(80),

performed\_by BIGINT REFERENCES employee(id),

payload JSONB,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT now()

);

**6 — Events (Kafka topics)**

* leave.requested — publish when employee requests leave
* leave.updated — changes to leave requests (approved/rejected/etc)
* leave.balance.updated — when balance changed (accrual or usage)
* notification.email — for notifications (or let Notification Service handle)  
  Event payloads should include requestId, employeeId, leaveType, startDate, endDate, days, status, timestamp, meta.

**7 — Minimal OpenAPI (YAML) — core endpoints**

yaml

CopyEdit

openapi: 3.0.3

info:

title: Leave Management API

version: 1.0.0

servers:

- url: /api

paths:

/leaves:

post:

summary: Request leave

security:

- oauth2: [openid, profile]

requestBody:

required: true

content:

application/json:

schema:

type: object

required: [leaveTypeId, startDate, endDate]

properties:

leaveTypeId: { type: integer }

startDate: { type: string, format: date }

endDate: { type: string, format: date }

reason: { type: string }

responses:

'201': { description: Created }

/leaves/{id}/approve:

post:

summary: Approve leave (manager)

parameters:

- name: id

in: path

required: true

schema: { type: integer }

requestBody:

content:

application/json:

schema:

properties:

comment: { type: string }

responses:

'200': { description: OK }

/employees/{id}/balances:

get:

summary: Get leave balances for employee

parameters:

- name: id

in: path

required: true

schema: { type: integer }

responses:

'200': { description: OK }

components:

securitySchemes:

oauth2:

type: oauth2

flows:

authorizationCode:

authorizationUrl: https://auth.example.com/auth

tokenUrl: https://auth.example.com/token

scopes: {}

**8 — Sample Spring Boot snippets**

**Entity (LeaveRequest.java)**

java

CopyEdit

@Entity

@Table(name = "leave\_request")

public class LeaveRequest {

@Id @GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@ManyToOne(fetch = FetchType.LAZY)

private Employee employee;

@ManyToOne

private LeaveType leaveType;

private LocalDate startDate;

private LocalDate endDate;

private BigDecimal days;

@Enumerated(EnumType.STRING)

private LeaveStatus status = LeaveStatus.PENDING;

private Instant requestedAt = Instant.now();

@Version

private Long version;

// getters/setters

}

**Repository**

java

CopyEdit

public interface LeaveRequestRepository extends JpaRepository<LeaveRequest, Long> {

List<LeaveRequest> findByEmployeeId(Long employeeId);

List<LeaveRequest> findByStatusAndEmployeeManagerId(LeaveStatus status, Long managerId);

}

**Service (transactional)**

java

CopyEdit

@Service

public class LeaveService {

@Autowired LeaveRequestRepository repo;

@Autowired LeaveBalanceRepository balanceRepo;

@Autowired KafkaTemplate<String, LeaveEvent> kafka;

@Transactional

public LeaveRequest requestLeave(Long empId, Long leaveTypeId, LocalDate start, LocalDate end, String reason){

BigDecimal days = calculateDays(start, end);

// validate entitlement

LeaveBalance bal = balanceRepo.findByEmployeeIdAndLeaveTypeIdAndYear(empId, leaveTypeId, Year.now().getValue())

.orElseThrow(...);

if (bal.getAvailable().compareTo(days) < 0) throw new InsufficientBalanceException();

// create request

LeaveRequest r = new LeaveRequest(...);

repo.save(r);

// optimistic: do NOT deduct until approved OR deduct on request depending on rules

kafka.send("leave.requested", new LeaveEvent(r));

return r;

}

@Transactional

public LeaveRequest approve(Long requestId, Long approverId, String comment){

LeaveRequest r = repo.findById(requestId).orElseThrow();

r.setStatus(LeaveStatus.APPROVED);

r.setDecidedBy(approverId);

r.setDecidedAt(Instant.now());

repo.save(r);

// deduct balance

LeaveBalance bal = balanceRepo.findByEmployeeIdAndLeaveTypeIdAndYear(...).orElseThrow();

bal.setUsed(bal.getUsed().add(r.getDays()));

balanceRepo.save(bal);

kafka.send("leave.approved", new LeaveEvent(r));

return r;

}

}

**Controller (minimal)**

java

CopyEdit

@RestController

@RequestMapping("/api/leaves")

public class LeaveController {

@Autowired LeaveService svc;

@PostMapping

public ResponseEntity<?> request(@RequestBody RequestDto dto, Principal p){

// map principal to employee id; call svc.requestLeave(...)

var r = svc.requestLeave(...);

return ResponseEntity.status(HttpStatus.CREATED).body(r.getId());

}

@PostMapping("/{id}/approve")

public ResponseEntity<?> approve(@PathVariable Long id, @RequestBody ApproveDto dto){

var r = svc.approve(id, /\*approverId\*/ dto.getApproverId(), dto.getComment());

return ResponseEntity.ok(r);

}

}

**9 — Apache Camel integration examples (concept)**

* **Route**: on leave.approved topic → route to email component smtp or push to notification.queue.
* Example Camel route in Spring:

java

CopyEdit

from("kafka:leave.approved?brokers={{kafka.bootstrap}}")

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

.to("bean:notificationService?method=sendLeaveApproved(${body})");

**10 — Concurrency & retry patterns**

* Use optimistic locking (@Version) on leave\_request to avoid lost updates.
* DB transactions for balance updates.
* If third-party integration fails (notification), push to a retry queue with DLQ. Use Kafka retries/compacted DLQ or Camel RedeliveryPolicy.

**11 — Test cases (core)**

1. **Request leave with sufficient balance** → created, event published
2. **Request leave with insufficient balance** → 4xx error
3. **Manager approves** → status=APPROVED, balance reduced
4. **Concurrent approvals** → one success, others fail/rollback due to optimistic lock
5. **Cancel before approval** → status=CANCELLED, no balance change
6. **Accrual job** → balances incremented correctly on Jan 1 (or monthly)
7. **Audit log entries** → present for every action
8. **Notification flow** → message published and consumed

**12 — Reporting & exports**

* Provide endpoints & scheduled jobs to export leave summaries (CSV/Excel) for payroll.
* Consider materialized views for heavy aggregation.

**13 — Audit, compliance & GDPR**

* Keep minimal PII in logs; store external\_id for identity provider.
* Add retention policy for audit logs and exports.

**14 — Example roadmap / next steps (pick one)**

* I can scaffold a Spring Boot project with Maven/Gradle + entities/repositories/controllers + Docker Compose for Postgres + Keycloak + Kafka.
* Or: generate full OpenAPI YAML and Postman collection.
* Or: produce UML diagrams (component, sequence, class) and PNGs.  
  Tell me which and I’ll generate it immediately.