**Leave Management System – LLD documentation**

**1 — High-level microservice decomposition**

1. **identity-service** (Keycloak or thin wrapper) — OAuth2/OIDC (recommended: Keycloak). Handles auth, user sync (optional).
2. **employee-service** — CRUD for employee profiles, entitlements, accrual rules. Source of truth for employee metadata.
3. **leave-service** — Core domain: create leave requests, validate balances, state transitions (PENDING→APPROVED/REJECTED/CANCELLED), audit. Enforces transactional rules.
4. **balance-service** (optional; can be part of employee-service) — Maintains per-employee leave balances, accruals, year rollovers. If separate, it owns balance updates and publishes events.
5. **notification-service** — Sends emails/SMS/push on leave events; consumes Kafka.
6. **reporting-service** (optional) — Aggregations, CSV export, heavy queries, scheduled reports (can read from materialized views).
7. **integration-service (camel)** — Calendar sync, payroll export, third-party connectors; Apache Camel routes to integrate external systems.
8. **api-gateway** — routing, rate-limiting, token verification, simple request enrichment.
9. **audit-service** (or centralized logging) — persistent audit store or use central DB/table per service plus ELK.

Notes: Small teams may combine employee + balance into one service. For simplicity in code generation I'll treat **employee-service** as owning entitlement & balance logic.

**2 — Bounded contexts & responsibilities**

**leave-service**

* Validate requests (date ranges, overlaps, policy checks).
* Persist LeaveRequest(s) and manage lifecycle.
* Coordinate balance deduction upon approval.
* Publish domain events: leave.requested, leave.approved, leave.rejected, leave.cancelled.
* Expose REST API and accept commands from managers/employees.

**employee-service**

* Manage employee profile fields (id, name, dept, managerId, employmentDate).
* Manage entitlement & accrual config per employee (entitlement/year, accrualFrequency, accrualAmount).
* Provide APIs for balance inquiries and updates.
* Runs scheduled accrual job if it owns balances.

**balance semantics**

* Yearly entitlement (e.g., 20 days/year) stored per employee.
* Balance record per year (employee\_id, leave\_type, entitlement, used, available).
* Accrual: monthly (entitlement / 12) or custom rate.
* Carryover rules and expiry implemented in employee-service.

**3 — Key domain models (entities)**

Common data types and names used across services:

**LeaveRequest (leave-service)**

* id: UUID
* tenant\_id: varchar
* employee\_id: UUID
* leave\_type: ENUM (ANNUAL, SICK, CASUAL, UNPAID)
* start\_date: date
* end\_date: date
* days: numeric(6,2)
* reason: text
* status: ENUM (PENDING, APPROVED, REJECTED, CANCELLED, WITHDRAWN)
* requested\_by: UUID
* requested\_at: timestamptz
* decided\_by: UUID (nullable)
* decided\_at: timestamptz (nullable)
* version: bigint (optimistic locking)
* metadata: jsonb

**LeaveBalance (employee-service)**

* id: BIGSERIAL
* employee\_id: UUID
* leave\_type: ENUM
* year: INT
* entitlement: numeric(6,2)
* used: numeric(6,2)
* available: numeric(6,2) GENERATED ALWAYS AS (entitlement - used) STORED
* updated\_at: timestamptz

**Employee (employee-service)**

* id: UUID
* employee\_number: varchar
* first\_name, last\_name, email
* manager\_id: UUID
* joined\_at: date
* active: boolean
* accrual\_rate: numeric(6,4) — days per month (or fraction)
* carryover\_allowed: boolean
* created\_at: timestamptz

**AuditLog (shared pattern)**

* id: UUID
* service: varchar
* entity\_type: varchar
* entity\_id: uuid
* action: varchar
* performed\_by: UUID
* payload: jsonb
* created\_at: timestamptz

**4 — Database DDL (PostgreSQL) — core tables**

sql

CopyEdit

-- Leave types (static)

CREATE TYPE leave\_type AS ENUM ('ANNUAL','SICK','CASUAL','UNPAID');

-- Leave status

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

-- leave\_service.leave\_request

CREATE TABLE leave\_request (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

tenant\_id VARCHAR(64) DEFAULT 'default',

employee\_id UUID NOT NULL,

leave\_type leave\_type NOT NULL,

start\_date DATE NOT NULL,

end\_date DATE NOT NULL,

days NUMERIC(6,2) NOT NULL,

reason TEXT,

status leave\_status NOT NULL DEFAULT 'PENDING',

requested\_by UUID NOT NULL,

requested\_at TIMESTAMPTZ NOT NULL DEFAULT now(),

decided\_by UUID,

decided\_at TIMESTAMPTZ,

metadata JSONB,

version BIGINT DEFAULT 0

);

CREATE INDEX idx\_leave\_employee ON leave\_request(employee\_id);

CREATE INDEX idx\_leave\_status ON leave\_request(status);

-- employee\_service.leave\_balance

CREATE TABLE leave\_balance (

id BIGSERIAL PRIMARY KEY,

employee\_id UUID NOT NULL,

leave\_type leave\_type NOT NULL,

year INT NOT NULL,

entitlement NUMERIC(6,2) NOT NULL,

used NUMERIC(6,2) NOT NULL DEFAULT 0,

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

updated\_at TIMESTAMPTZ NOT NULL DEFAULT now(),

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

);

-- employee\_service.employee

CREATE TABLE employee (

id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

employee\_number VARCHAR(64) UNIQUE,

first\_name VARCHAR(100) NOT NULL,

last\_name VARCHAR(100) NOT NULL,

email VARCHAR(150) UNIQUE,

manager\_id UUID,

joined\_at DATE,

active BOOLEAN DEFAULT TRUE,

accrual\_rate NUMERIC(6,4) DEFAULT 0, -- days per month

carryover\_allowed BOOLEAN DEFAULT FALSE,

created\_at TIMESTAMPTZ DEFAULT now()

);

Notes:

* Use pgcrypto or uuid-ossp extension for UUID generation (gen\_random\_uuid()).
* Use partitioning on leave\_request if scale demands (by year or tenant).

**5 — REST API contracts (representative)**

**leave-service (core endpoints)**

* POST /api/v1/leaves — create request  
  Request:

json

CopyEdit

{ "employeeId":"uuid", "leaveType":"ANNUAL", "startDate":"2025-09-01", "endDate":"2025-09-03", "reason":"vacation" }

Response: 201 Created with { "requestId":"uuid", "status":"PENDING" }

* GET /api/v1/leaves/{id} — get request
* GET /api/v1/employees/{employeeId}/leaves — list leaves (filter params: year, status)
* POST /api/v1/leaves/{id}/approve — approve (manager only)  
  Body: { "approverId":"uuid", "comment": "ok" }
* POST /api/v1/leaves/{id}/reject — reject
* POST /api/v1/leaves/{id}/cancel — cancel by employee

**employee-service**

* GET /api/v1/employees/{id} — employee profile
* GET /api/v1/employees/{id}/balances — list balances per year & type
* PATCH /api/v1/employees/{id}/balances/{year}/{leaveType} — admin change balance
* POST /api/v1/employees — create employee

Security: JWT Bearer token (Authorization: Bearer ...) validated at API gateway or at service using Spring Security with OIDC.

**6 — Kafka topics & sample messages**

Topics:

* leave.requested — produced by leave-service on successful creation
* leave.status.changed — produced on any status change (approved/rejected/cancelled)
* leave.balance.updated — produced when balance change occurs
* employee.updated — employee changes propagated

Sample leave.requested payload (JSON):

json

CopyEdit

{

"eventType": "leave.requested",

"occurredAt": "2025-08-12T10:15:30Z",

"data": {

"requestId": "uuid",

"employeeId": "uuid",

"leaveType": "ANNUAL",

"startDate": "2025-09-01",

"endDate": "2025-09-03",

"days": 3.0,

"status": "PENDING"

}

}

Use JSON Schema or Avro for strong typing. Use compacted topics for idempotency if needed.

**7 — Sequence flows (textual diagrams)**

**A — Employee requests leave (synchronous validation + async notification)**

1. Frontend -> API Gateway (POST /api/v1/leaves with JWT)
2. API Gateway -> leave-service (auth info forwarded)
3. leave-service:
   * fetch employee profile/balance from employee-service (GET /balances)
   * validate overlap, entitlement: if insufficient -> 400 return
   * persist leave\_request (DB transaction)
   * publish leave.requested to Kafka
   * return 201 Created
4. notification-service (Kafka consumer) => email manager

**B — Manager approves leave (transactional)**

1. Manager -> API Gateway -> leave-service POST /leaves/{id}/approve
2. leave-service: begin DB tx
   * verify manager is authorized (managerId match via employee-service)
   * check optimistic lock/version on leave\_request
   * update leave\_request.status = APPROVED, set decided\_by, decided\_at
   * call employee-service (or balance-service) to deduct balance: POST /balances/deduct (synchronous REST or via Kafka request-reply). Prefer in-process two-phase pattern:
     + Option 1 (synchronous): call employee-service to deduct in same flow (two service calls inside same transaction boundary is impossible across services) — so use local transaction + idempotent compensation or event driven:
     + Option 2 (recommended): update leave\_request in DB, publish leave.status.changed event. balance-service consumes the event and performs balance deduction with its own transaction. If balance-service fails (rare), it publishes a compensating event or manual intervention. Use a retry + DLQ.
   * publish leave.approved event
   * commit tx
3. Notification + payroll listeners consume event

Note: Avoid distributed transactions (XA). Use eventual consistency with well-defined compensations.

**8 — Concurrency & consistency patterns**

* Use **optimistic locking** (@Version / version column) on leave\_request and optionally on leave\_balance.
* Enforce **database constraints**: unique constraints for overlapping leaves (if business requires), CHECK (start\_date <= end\_date).
* For balance enforcement:
  + On leave creation: either **reserving** (create request and reserve balance entry) OR only deduct on approval.
  + Recommended: deduct only on approval to avoid complex reservation expirations.
  + If you need reservation (to show available balance while multiple requests pending), implement balance\_reservation table with TTL and background cleanup.
* Idempotency: include requestId client-provided or server generated and do idempotent inserts (use unique constraint on externalRef or id).
* Retries: use exponential backoff and dead-letter topics for Kafka consumers.

**9 — Apache Camel integration ideas**

Use integration-service with Camel routes for:

* From Kafka leave.approved -> smtp component to send email
* From Kafka leave.approved -> ical/Google Calendar API connector (route with OAuth2)
* From REST POST /payroll/export -> transform leaves -> send to SFTP or call payroll API

Example simple Camel route (Spring Boot):

java

CopyEdit

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

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

.to("bean:notificationService?method=sendLeaveApproved");

Camel supports retry, dead-letter, and outbox patterns.

**10 — Accrual design (monthly job)**

Two options:

**A. employee-service owns accruals (recommended)**

* Maintain accrual\_rate (days/month) per employee or global policy per leave\_type.
* Monthly scheduled job (Spring @Scheduled(cron = "...") or Kubernetes CronJob) calls a service method:
  + For each employee and leave\_type for current year:
    - UPDATE leave\_balance SET entitlement = entitlement + accrual\_amount, updated\_at = now()
    - Publish leave.balance.updated event.
* For new joiners: pro-rate accrual until joining month.

**B. scheduler-service triggers accrual via events**

Edge-cases:

* Carryover expiration: run yearly job to move or expire balance.
* Backfills: provide admin API to run accrual for missed months.

**11 — Security & RBAC**

* Use **Keycloak** / OIDC for authentication. Configure API Gateway to validate tokens and inject X-User-Id header.
* Roles:
  + ROLE\_EMPLOYEE
  + ROLE\_MANAGER
  + ROLE\_HR
  + ROLE\_ADMIN
* leave-service endpoints:
  + Employees can create/cancel their own leaves and view their history.
  + Managers can approve/reject for direct reports.
  + HR/Admin can adjust balances and view organization-wide reports.
* Input validation: all endpoints validate payloads and return 400 on invalid.

**12 — Observability & monitoring**

* **Tracing**: OpenTelemetry instrumentation (spring-boot-opentelemetry) and Jaeger.
* **Metrics**: Micrometer + Prometheus (expose actuator/metrics).
* **Logs**: JSON structured logging shipped to ELK/EFK or Loki.
* **Health**: Spring Boot Actuator endpoints (/actuator/health, /actuator/metrics), Kubernetes liveness/readiness.

**13 — Testing strategy**

* **Unit tests**: JUnit + Mockito for services and domain logic (balance calc, accrual).
* **Integration tests**: Spring Boot test slices with Testcontainers (Postgres, Kafka).
* **Contract tests**: Pact or Spring Cloud Contract for async messages (Kafka) and RESTs between services.
* **E2E**: using a staging stack (K8s or docker-compose) to validate flows (request→approve→balance update).

**14 — Deployment & infra (suggested)**

* Containerize each Spring Boot service (distroless JDK image). Use Maven/Gradle build and multi-stage Dockerfile.
* Local dev: Docker Compose with Postgres, Kafka (Confluent or Redpanda), Zookeeper (if needed), Keycloak, and API Gateway (e.g., Kong or Spring Cloud Gateway).
* Production: Kubernetes (EKS/GKE/AKS). Use Helm charts or Kustomize.
* Use persistent storage for Postgres (PVC), configure database backups.
* Use managed Kafka (Confluent Cloud, MSK) if possible.

Example Docker Compose (local dev) to include later if you want.

**15 — Sample code snippets (Spring idioms)**

**Entity (LeaveRequest.java)** — include @Version:

java

CopyEdit

@Entity

@Table(name="leave\_request")

public class LeaveRequest {

@Id

private UUID id = UUID.randomUUID();

private UUID employeeId;

@Enumerated(EnumType.STRING) private LeaveType leaveType;

private LocalDate startDate;

private LocalDate endDate;

private BigDecimal days;

private String reason;

@Enumerated(EnumType.STRING) private LeaveStatus status = LeaveStatus.PENDING;

private UUID requestedBy;

private Instant requestedAt = Instant.now();

@Version

private Long version;

}

**Service (LeaveService.java)** (simplified)

java

CopyEdit

@Transactional

public LeaveRequestDto createLeave(CreateLeaveCmd cmd) {

// validate dates, compute days

// check overlap with repository

// TODO: decide reserve vs deduct later policy

LeaveRequest entity = repo.save(mapper.toEntity(cmd));

kafkaTemplate.send("leave.requested", mapper.toEvent(entity));

return mapper.toDto(entity);

}

**Consumer (BalanceService)** listens to leave.status.changed:

java

CopyEdit

@KafkaListener(topics="leave.status.changed")

public void handle(LeaveStatusEvent ev) {

if(ev.getStatus().equals("APPROVED")) {

// deduct balance in a transaction

balanceRepository.deduct(ev.getEmployeeId(), ev.getLeaveType(), ev.getDays());

}

}

**16 — Error handling, retries & DLQ**

* Kafka consumers: configure retry backoff (~exponential) and DLQ (dead-letter topic) for poison messages.
* For crucial async processing (balance deduction), implement idempotency keys (requestId) and at-least-once semantics with idempotent DB operations.
* For failed approvals where balance deduction fails, raise alert and create manual intervention tickets. Optionally implement compensating transactions (reverse approval) if necessary.

**17 — Migration & data model evolution**

* Use Flyway or Liquibase for DB migrations.
* Version Kafka message schemas with backward/forward compatibility (use Avro/Schema Registry).

**18 — Example operational flows & edge cases to design for**

* Employee applies overlapping leave: reject or merge based on policy.
* Concurrent approvals: optimistic lock on leave request, first committer wins.
* Manager approves but balance insufficient (due to another approval): either prevent approval with check or allow and trigger compensation; prefer check and prevent approval with user-visible error.
* Accrual after entitlement change: recalc future entitlements or run reconciliation job.

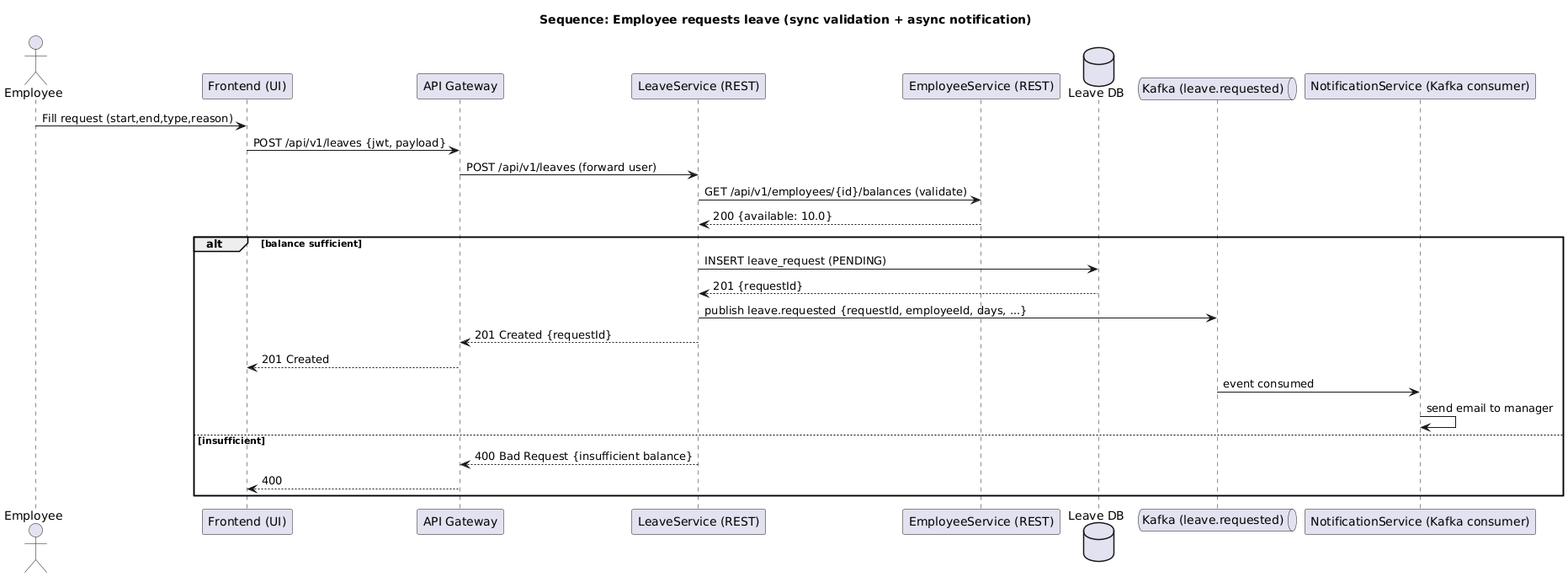
**19 — Data model & API summary table (quick)**

* LeaveRequest (LeaveService) — create/read/update lifecycle
* LeaveBalance (EmployeeService) — read/update balances
* Employee (EmployeeService) — profile and entitlement
* Kafka events: leave.requested, leave.status.changed, leave.balance.updated, employee.updated

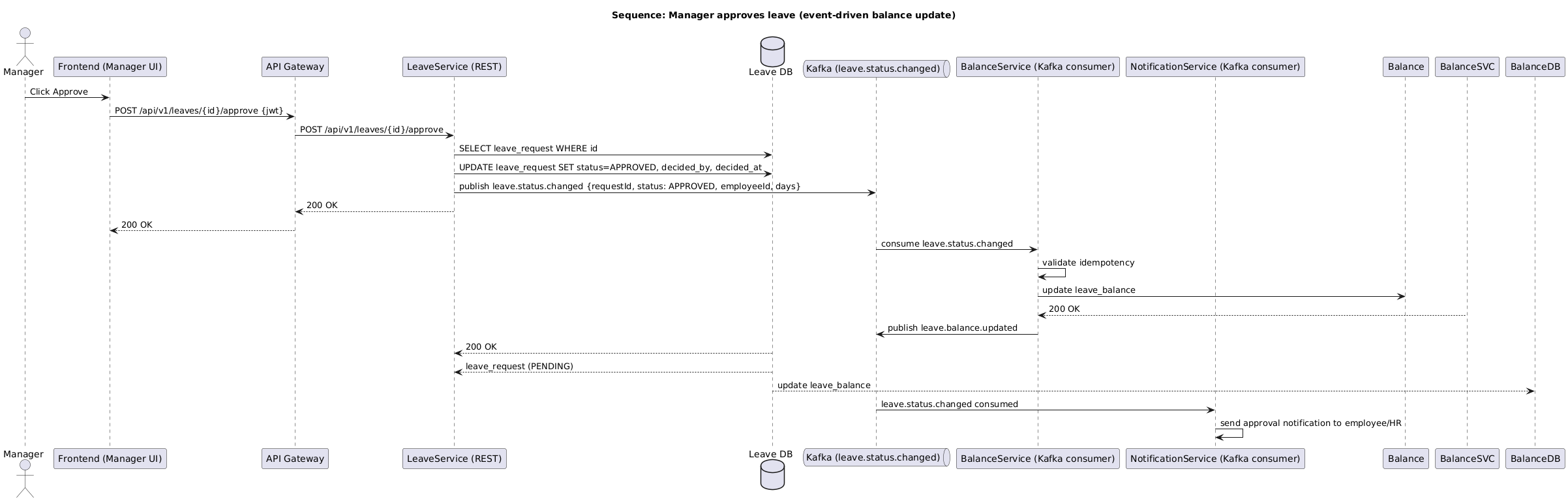
**20 — Deliverables I can produce next (pick one or more)**

* Full **Spring Boot microservice skeletons** (maven) for leave-service, employee-service, notification-service with entities, repositories, DTOs, OpenAPI YAML and Kafka producers/consumers.
* **Docker Compose** for local dev: Postgres + Kafka + Zookeeper + Keycloak + services.
* **Kubernetes Helm chart** example for the services.
* **UML diagrams**: sequence diagrams and class diagrams in PNG.
* **Detailed OpenAI** specs and example Postman collection.
* **Test containers-based integration test examples**.

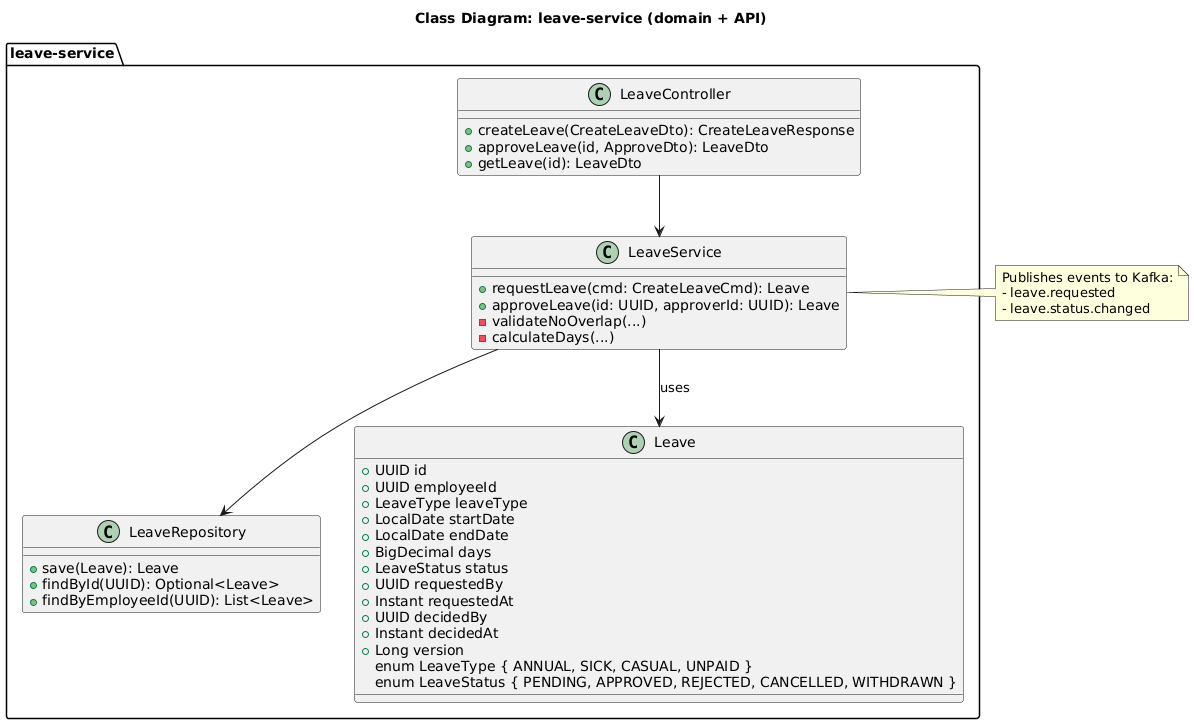
**Sequence Diagram: Employee requests leave**



Sequence diagram — Manager approves leave



Class diagram — leave-service (domain & API layer)



Class diagram — employee-service (profile & balance)

A screenshot of a computer

AI-generated content may be incorrect.

Component diagram — LMS microservice landscape

A diagram of a computer

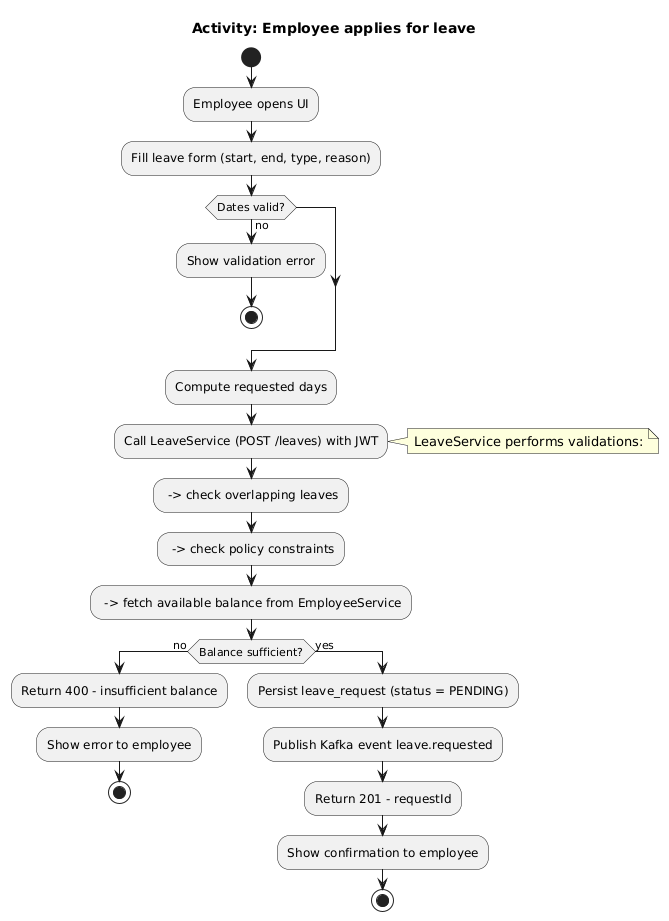
AI-generated content may be incorrect.

Sequence diagram — Monthly accrual job

A screenshot of a computer

AI-generated content may be incorrect.

Activity diagram — Employee applies for leave

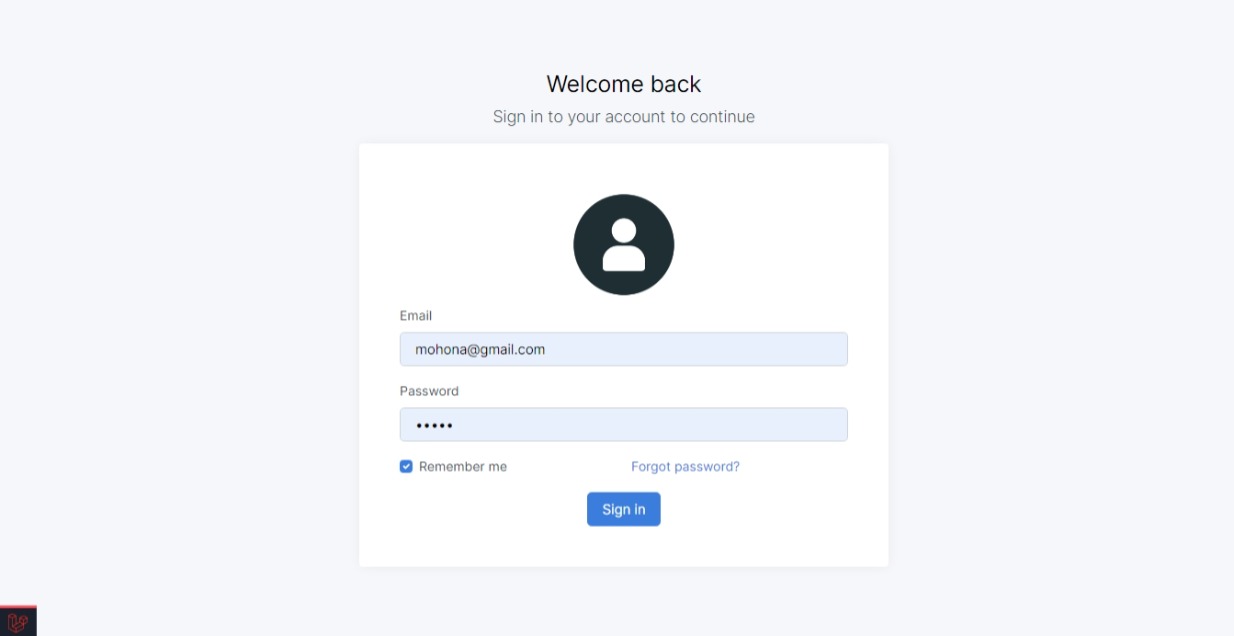


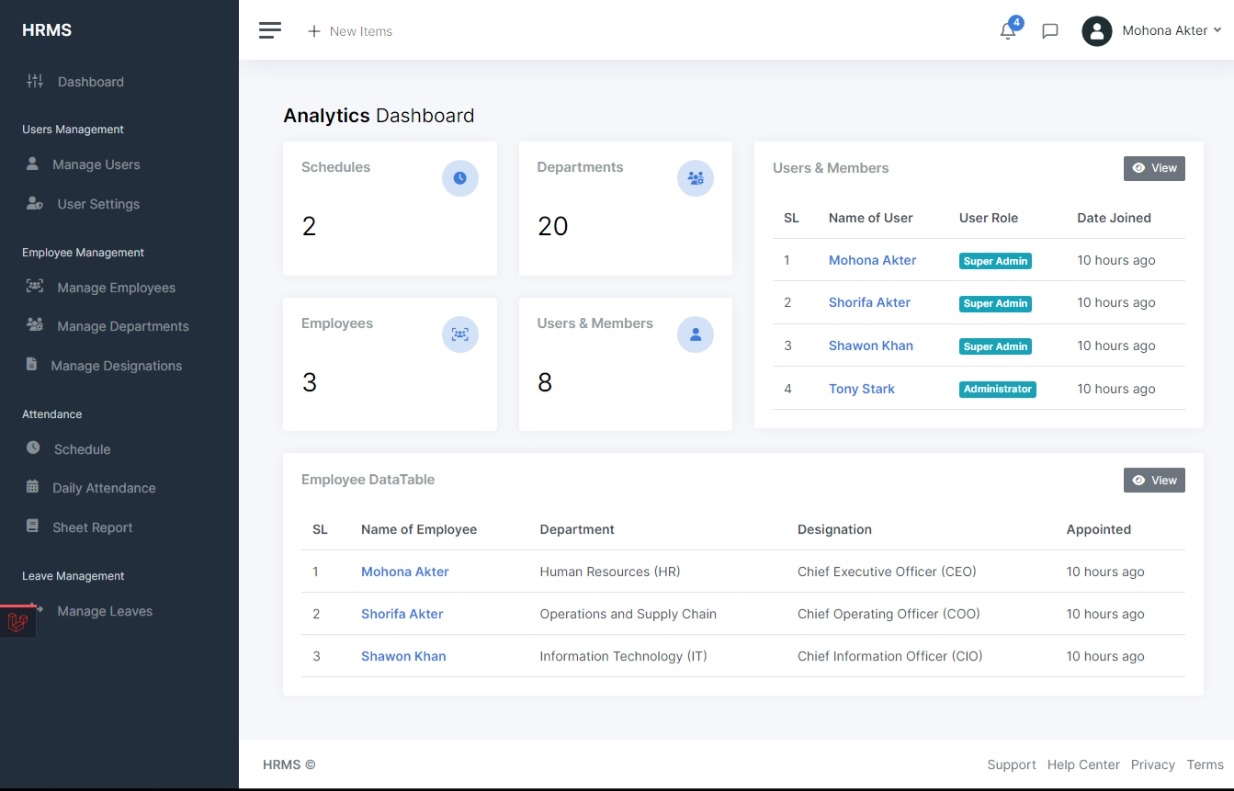
Activity diagram — Manager approves leave

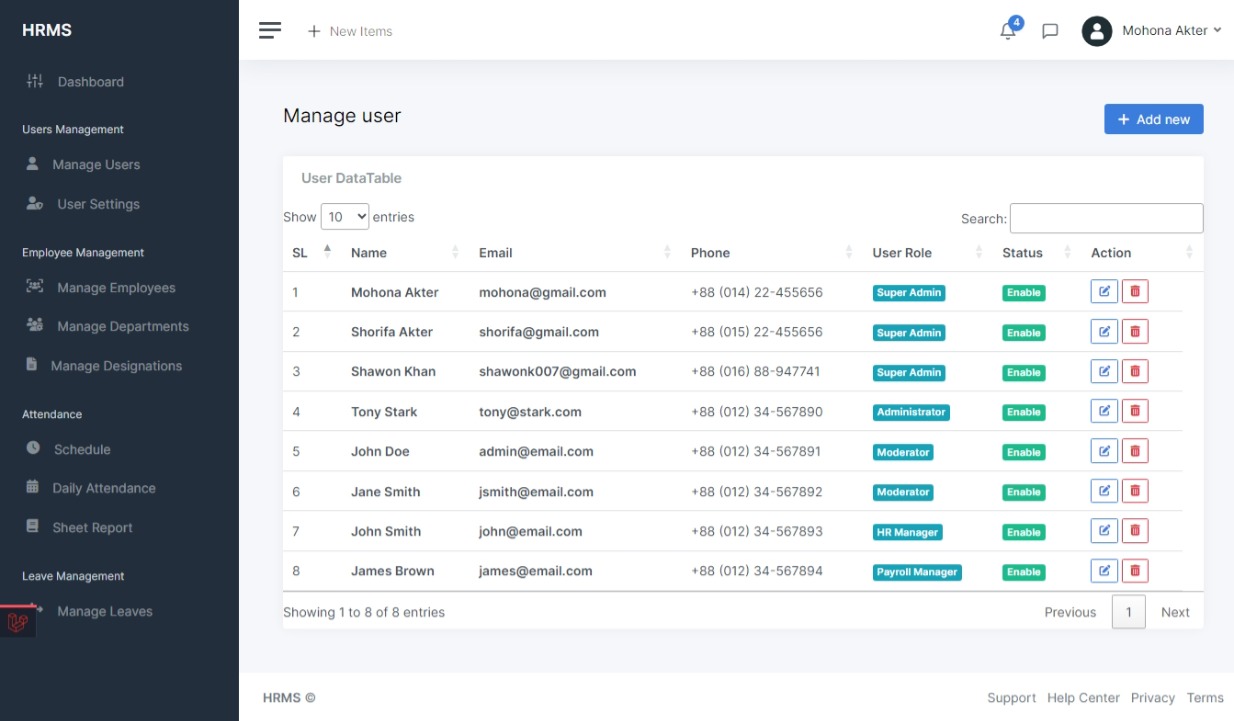
A diagram of a flowchart

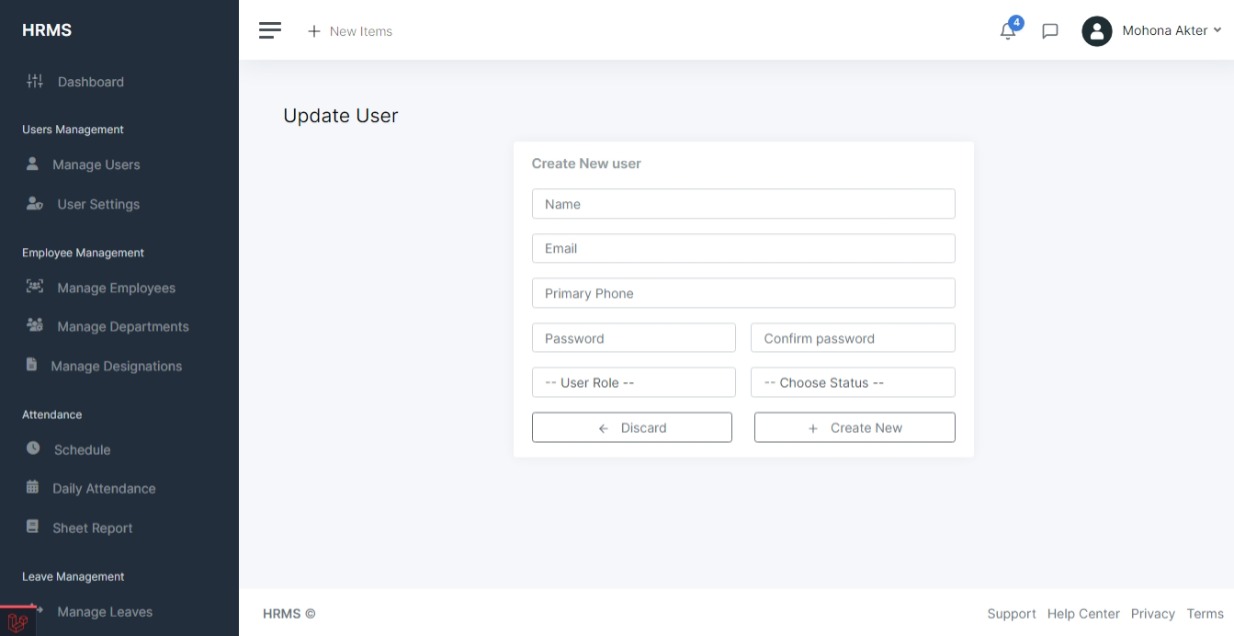
AI-generated content may be incorrect.

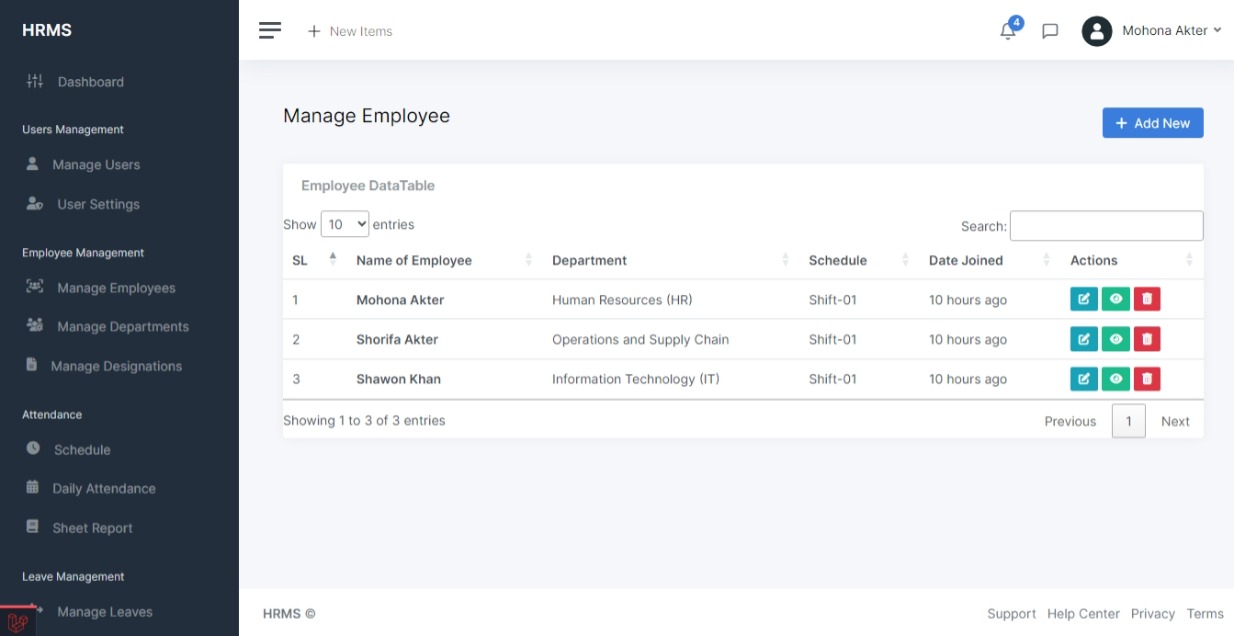
UI Design Information

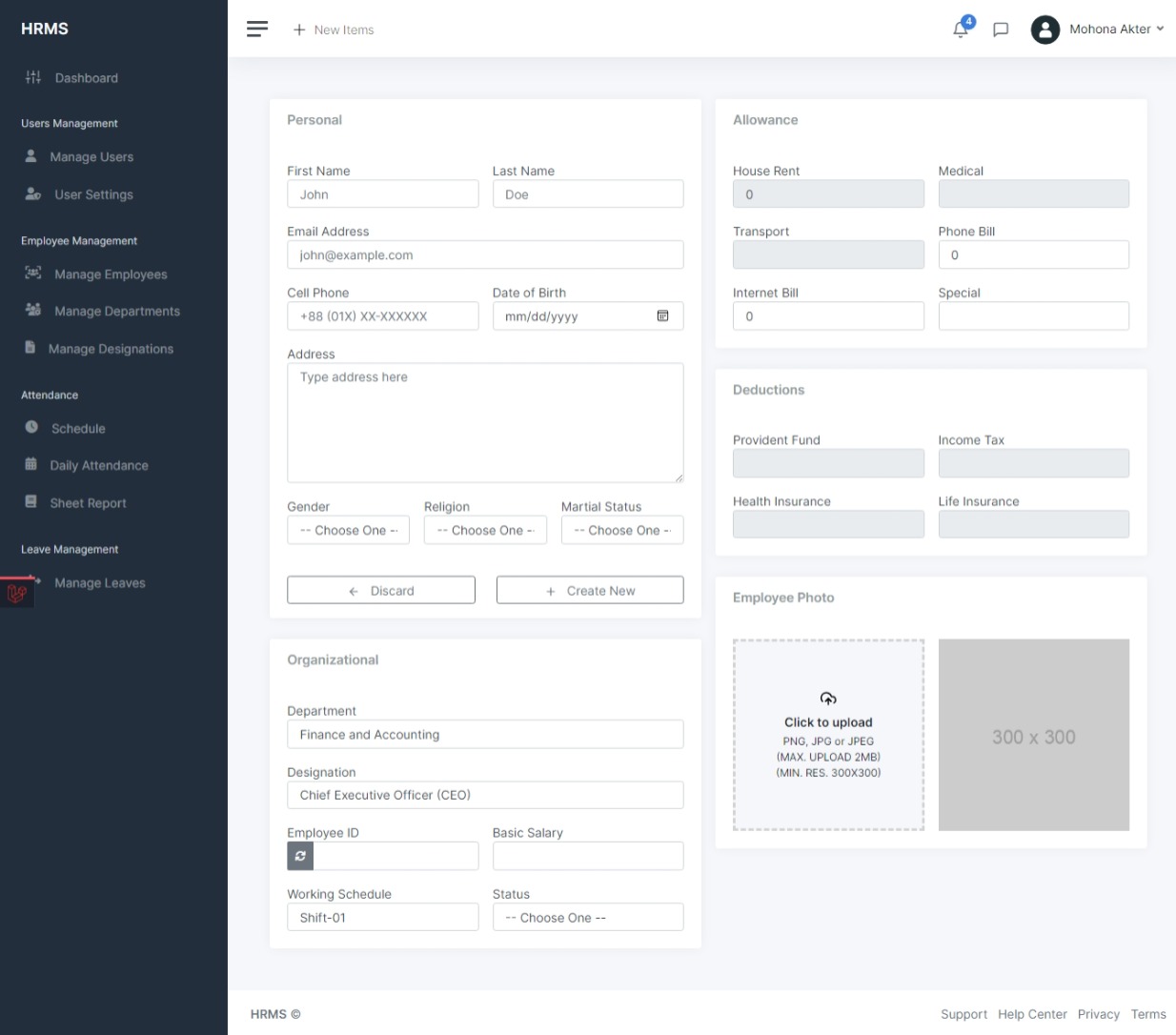
[](https://github.com/MOHONA678/employee-management-system/blob/main/public/img/screenshots/Login.jpeg)

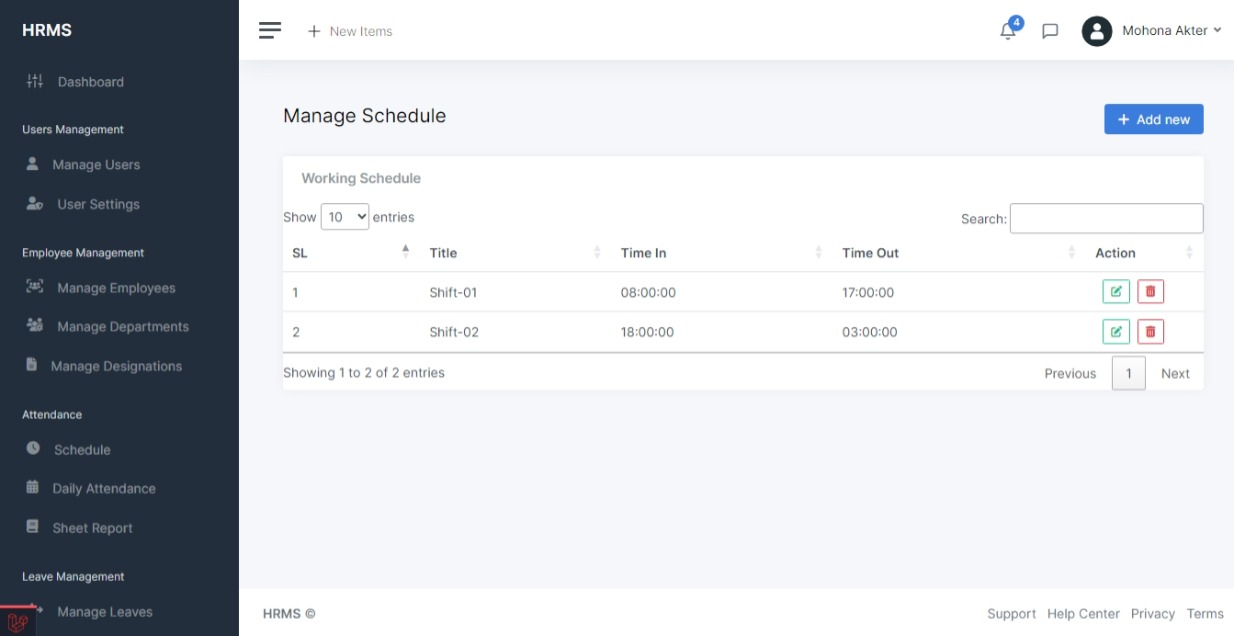
[](https://github.com/MOHONA678/employee-management-system/blob/main/public/img/screenshots/Dashboard.jpeg)

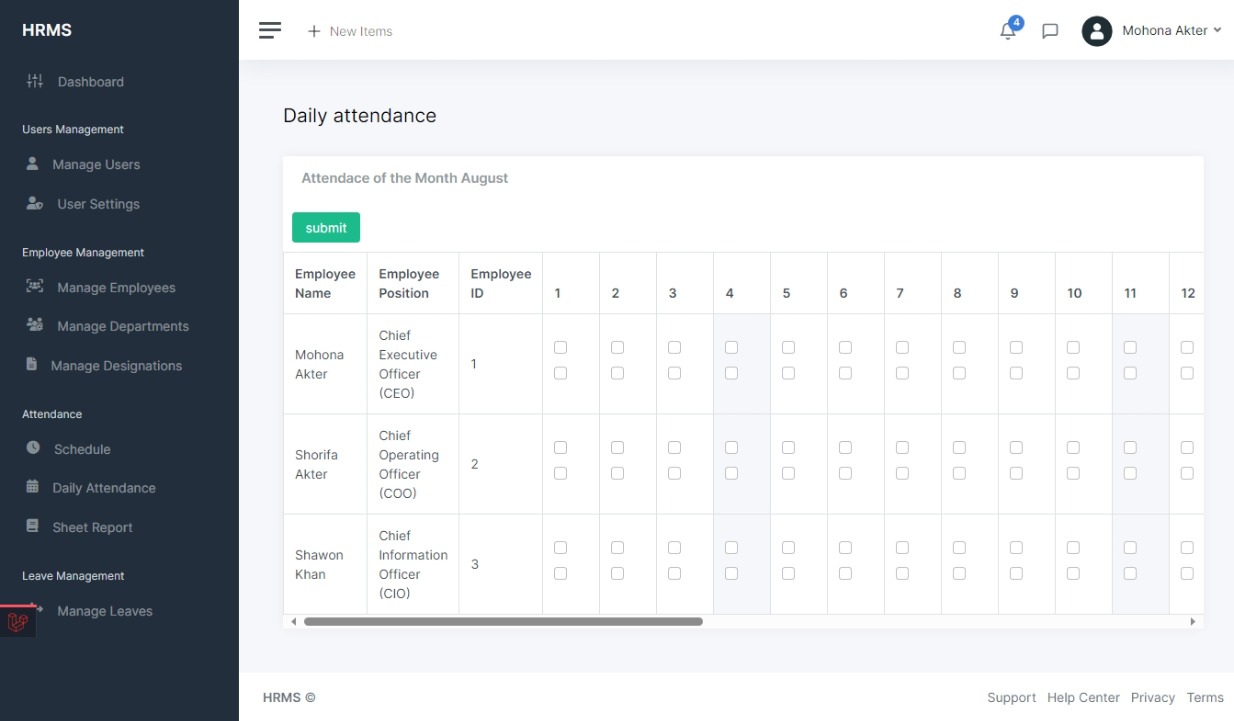
[](https://github.com/MOHONA678/employee-management-system/blob/main/public/img/screenshots/Users.jpeg)

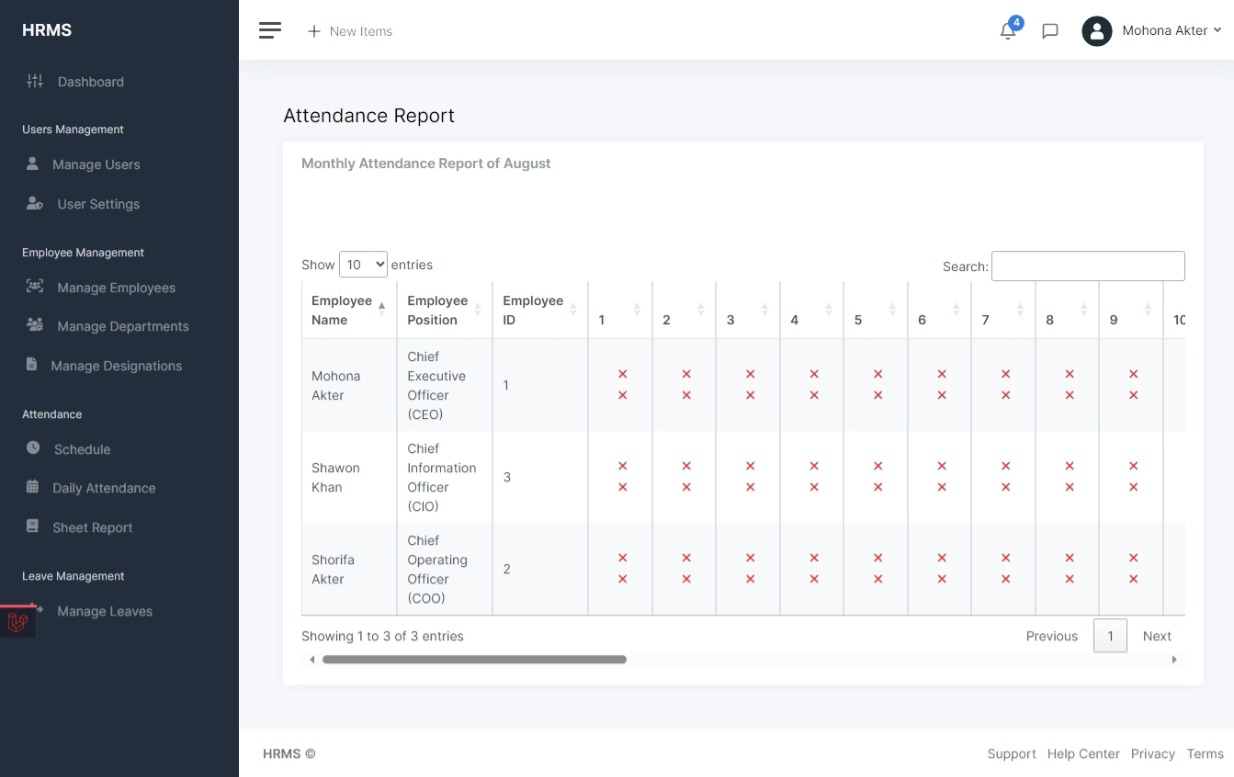
[](https://github.com/MOHONA678/employee-management-system/blob/main/public/img/screenshots/New_User.jpeg)

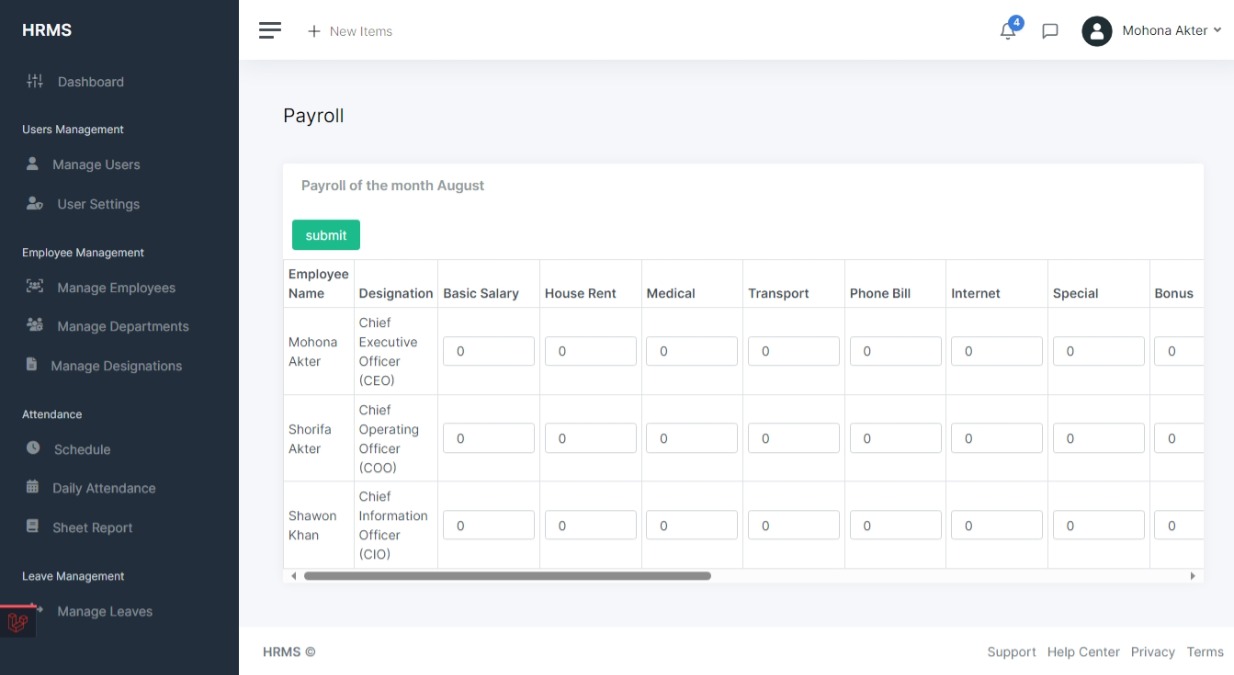
[](https://github.com/MOHONA678/employee-management-system/blob/main/public/img/screenshots/Employees.jpeg)

[](https://github.com/MOHONA678/employee-management-system/blob/main/public/img/screenshots/New_Employee.jpeg)

[](https://github.com/MOHONA678/employee-management-system/blob/main/public/img/screenshots/Schedule.jpeg)

[](https://github.com/MOHONA678/employee-management-system/blob/main/public/img/screenshots/Daily_Attendance.jpeg)

[](https://github.com/MOHONA678/employee-management-system/blob/main/public/img/screenshots/Attendance_Report.jpeg)

[](https://github.com/MOHONA678/employee-management-system/blob/main/public/img/screenshots/Payroll.jpeg)

**Technologies Used**