Architectural Design of Human Resource Management System (HRMS) using the 4+1 View Model

Course: B.Tech Computer Engineering – Semester 5
Subject: Software Engineering
Submitted to: Prof. Sumit Kalra
Submitted by:

· Dhruvkumar Mulani – Roll No. 231040011006
· Shyam Copda – Roll No. 231040011028
· Naman Singh – Roll No. 231040012009

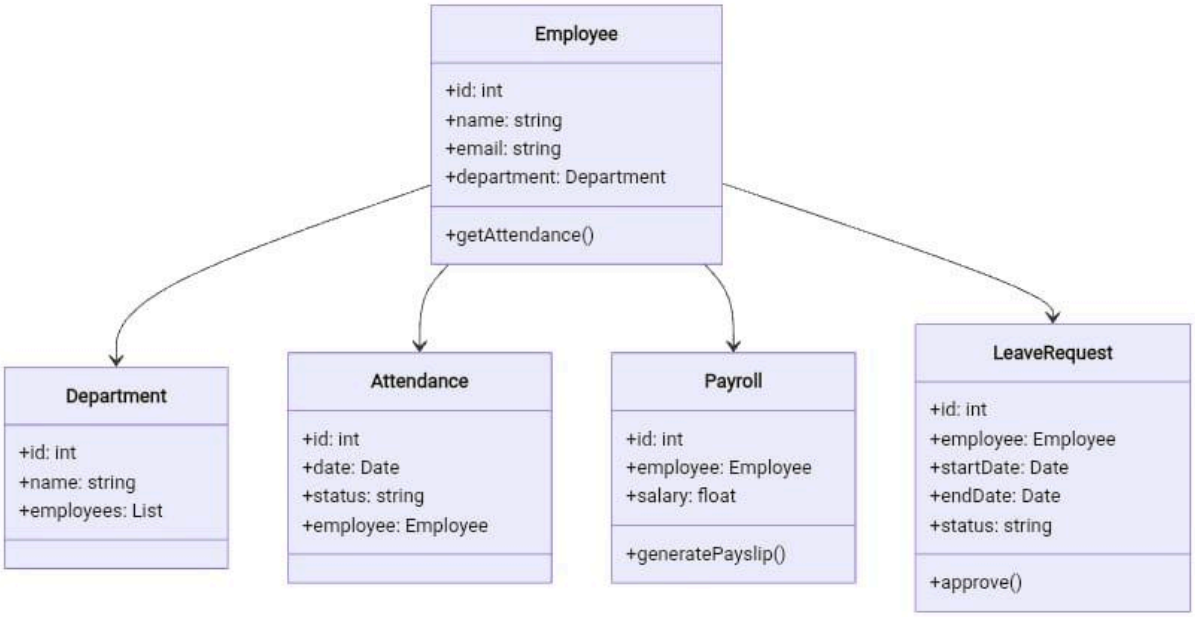
Date: 17th October 2025

---

1. Logical View

Description: The Logical View represents the functional requirements of the system through object-oriented design. It focuses on the key classes, their attributes, methods, and relationships that form the core business entities of the HRMS.

Diagram:


Employee
+id: int
+name: string
+email: string
+department: Department
+getAttendance()
Department
+id: int
+name: string
+employees: List
Attendance
+id: int
+date: Date
+status: string
+employee: Employee
Payroll
+id: int
+employee: Employee
+salary: float
+generatePayslip()
LeaveRequest
+id: int
+employee: Employee
+startDate: Date
+endDate: Date
+status: string
+approve()

```
# Employee
+id: int
+name: string
+email: string
+department: Department
+getAttendance()

# Attendance
+id: int
+date: Date
+status: string
+employee: Employee

# Payroll
+id: int
+employee: Employee
+salary: float
+generatePayslip()

# LeaveRequest
+id: int
+employee: Employee
+startDate: Date
+endDate: Date
+status: string
+approve()
```

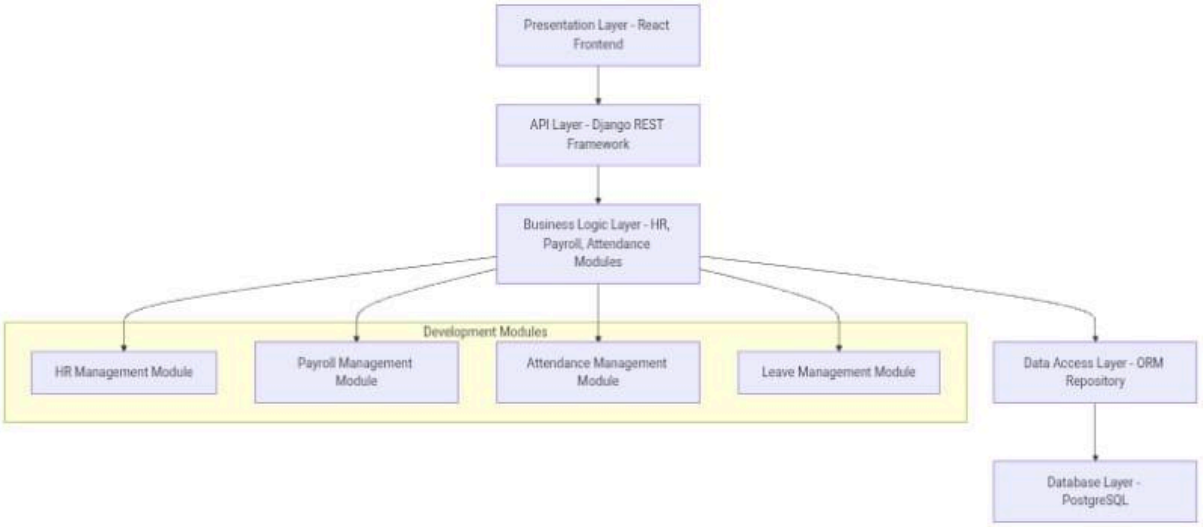
Explanation: The logical view showcases four main entities: Employee, Attendance, Payroll, and LeaveRequest. Employee serves as the central class with associations to other entities. Attendance tracks employee presence, Payroll manages salary information and payslip generation, while LeaveRequest handles leave applications. This design follows object-oriented principles with clear relationships between entities, enabling modular development and maintenance. The attributes and methods defined support core HR operations including attendance tracking, payroll processing, and leave management.

---

2. Development View

Description: The Development View illustrates the system's organization in terms of software modules, layers, and development environment structure, focusing on how the system is decomposed into manageable components.

Diagram:


Presentation Layer - React Frontend
API Layer - Django REST Framework
Business Logic Layer - HR, Payroll, Attendance Modules
Development Modules
HR Management Module
Payroll Management Module
Attendance Management Module
Leave Management Module
Data Access Layer - ORM Repository
Database Layer - PostgreSQL

```
Presentation Layer - React
|
API Layer - Django REST Framework
|
Business Logic Layer - HR, Payroll, Attendance Modules
|
Development Modules
|-- HR Management Module
|-- Payroll Management Module
|-- Attendance Management Module
|-- Leave Management Module
|
Data Access Layer - ORM Repository
|
Database Layer - PostgreSQL
```

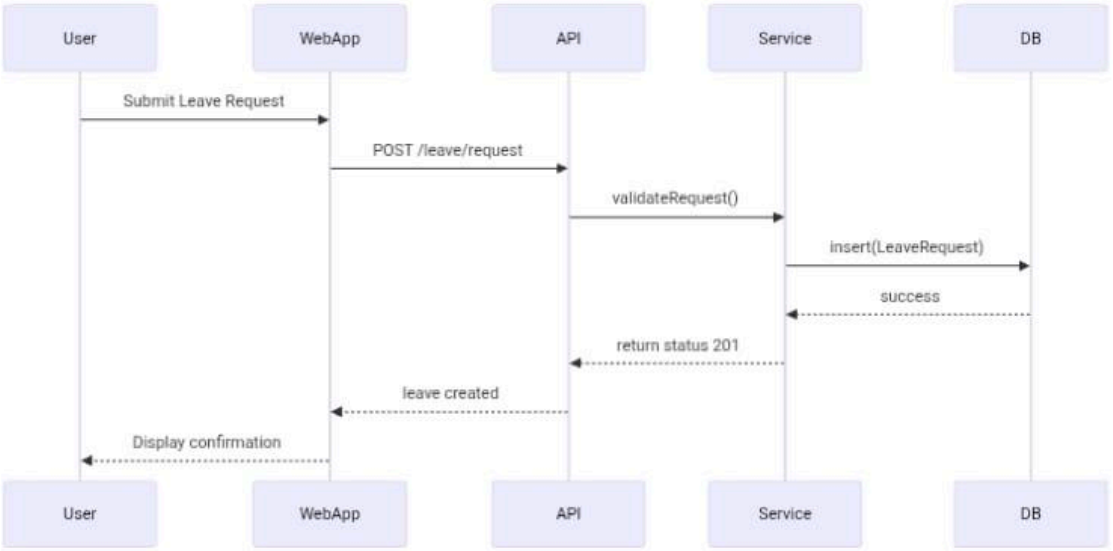
Explanation: The development architecture follows a layered pattern with clear separation of concerns. The Presentation Layer uses React for user interface, while the API Layer employs Django REST Framework for handling HTTP requests. Business logic is modularized into HR, Payroll, Attendance, and Leave Management components. The Data Access Layer uses ORM for database abstraction, and PostgreSQL serves as the persistent storage. This modular approach enables parallel development, easier testing, and maintainability. Each layer has distinct responsibilities, promoting code reusability and scalability.

---

3. Process View

Description: The Process View captures the dynamic behavior of the system, focusing on process interactions, data flow, and concurrency aspects during runtime execution.

Diagram:


User
WebApp
API
Service
DB
Submit Leave Request
POST /leave/request
validateRequest()
insert(LeaveRequest)
success
return status 201
leave created
Display confirmation
User
WebApp
API
Service
DB

```
User → WebApp → API → Service → DB

Submit Leave Request Sequence:
User → WebApp → POST /leave/request → validateRequest() → insert(LeaveRequest) → success

Response Flow:
DB → Service → API → WebApp → User
Leave created → return status 201
```

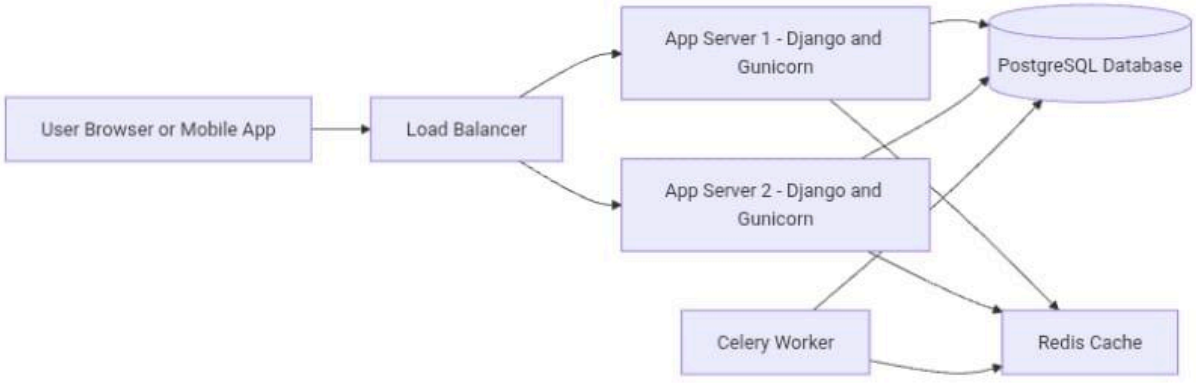
Explanation: The process view demonstrates the leave request submission workflow. When a user submits a leave request through the web application, it triggers a POST API call to /leave/request. The system validates the request, inserts a new LeaveRequest record into the database, and returns a 201 status upon successful creation. This asynchronous flow ensures non-blocking operations and better user experience. The process handles concurrent requests through Django's request-response cycle and utilizes database transactions to maintain data consistency during leave record creation.

---

4. Physical View

Description: The Physical View describes the deployment architecture, showing how software components are mapped to hardware nodes and infrastructure elements.

Diagram:


User Browser or Mobile App
Load Balancer
App Server 1 - Django and Gunicorn
App Server 2 - Django and Gunicorn
PostgreSQL Database
Celery Worker
Redis Cache

```
User Browser/Mobile App
        ↓
   Load Balancer
    ↙      ↘
App Server 1      App Server 2
(Django + Gunicorn) (Django + Gunicorn)
        ↖      ↗
   PostgreSQL Database
        ↑
    Redis Cache
        ↑
  Celery Worker
```

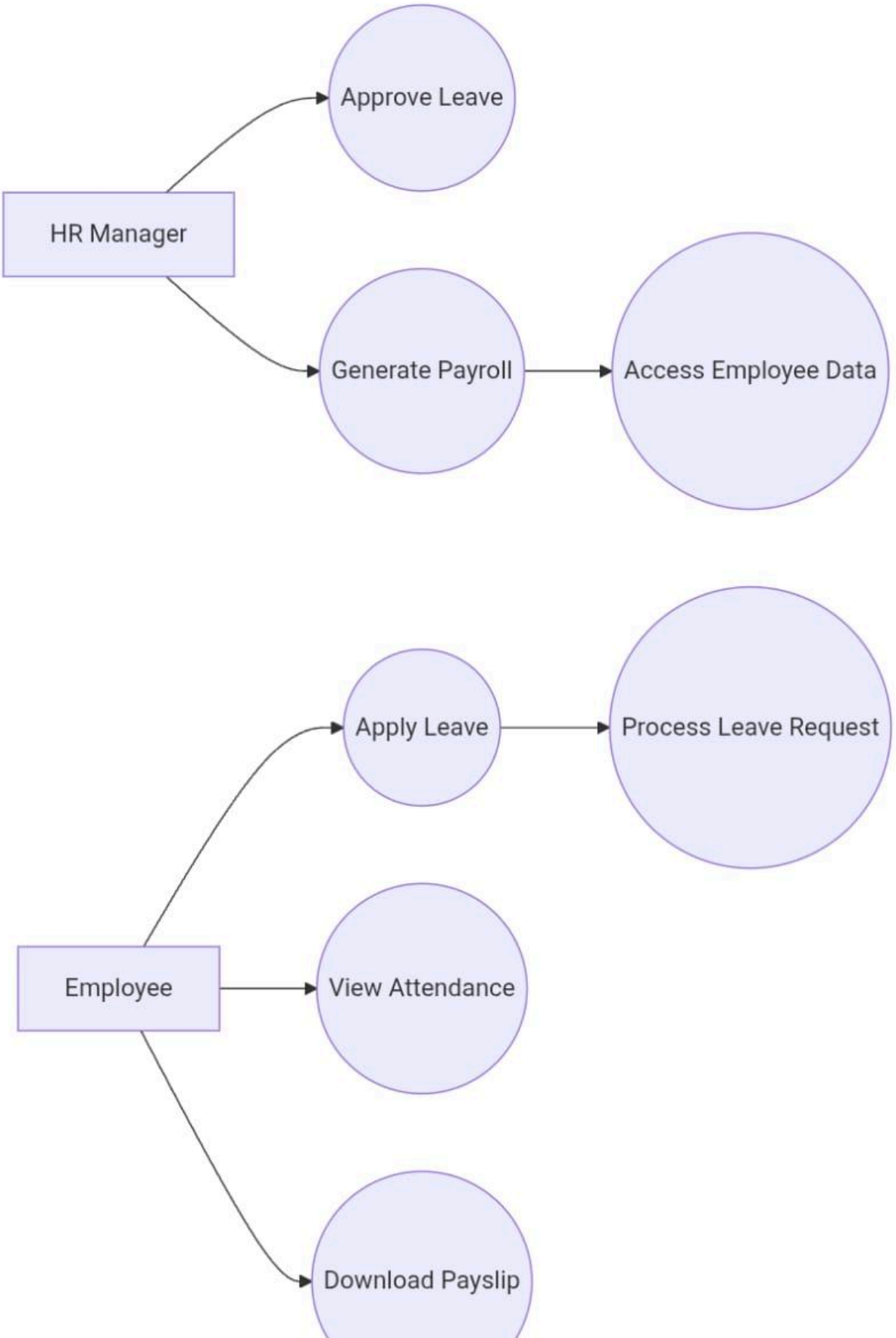
Explanation: The physical deployment uses a scalable architecture with multiple application servers behind a load balancer for handling high traffic. Each app server runs Django with Gunicorn as the WSGI server. PostgreSQL serves as the primary database for persistent storage, while Redis provides caching for improved performance. Celery workers handle background tasks like payroll processing and email notifications. This distributed setup ensures high availability, load distribution, and fault tolerance. The architecture supports horizontal scaling by adding more app servers and utilizes optimized resource allocation for different components.

---

5. Scenarios View (+1)

Description: The Scenarios View demonstrates how the system supports key use cases by showing the collaboration between different architectural components.

Use Case Diagram:


HR Manager
Approve Leave
Generate Payroll
Access Employee Data
Employee
Apply Leave
Process Leave Request
View Attendance
Download Payslip

```
Actors: Employee, HR Manager

Employee Use Cases:
- Apply Leave
- View Attendance
- Download Payslip

HR Manager Use Cases:
- Approve Leave
- Generate Payroll
- Access Employee Data
- Process Leave Request
```

Key Scenarios:

1. Leave Application Scenario: Employee submits leave request → System validates availability → HR manager approves/rejects → Notification sent to employee.
2. Payroll Generation Scenario: HR manager initiates payroll → System calculates salaries → Generates payslips → Employees can download payslips.

Explanation: The scenarios view integrates all architectural perspectives. For leave management, the logical entities (LeaveRequest, Employee) interact through process flows involving presentation layer (React UI), business logic (validation), and data persistence. The payroll scenario demonstrates how physical components (Celery workers, database) collaborate with development modules (Payroll Management) to generate payslips. These scenarios validate that all views work cohesively to support business requirements, ensuring the architecture meets functional needs while maintaining performance, scalability, and usability standards.

Conclusion

The 4+1 View Model provides a comprehensive framework for designing and documenting the HRMS architecture. Each view addresses specific stakeholder concerns: logical view for developers, development view for project managers, process view for integrators, physical view for system administrators, and scenarios for end-users. This structured approach ensures that the HRMS meets all functional requirements while maintaining scalability, maintainability, and performance standards across all architectural dimensions.