Leave Management System — Part 2

Goal: Provide a robust, maintainable architecture for the Leave Management MVP that supports the current 50 employees and can scale to 500+ employees. Deliverables include architecture diagram, DB schema, API flows, scaling plan, deployment checklist, and README content for submission.

Primary actors: HR admin, Employee, System (cron jobs / maintenance)

Functional requirements:

- Add employee (name, email, department, joining_date)
- Apply for leave
- Approve / reject leave
- Fetch leave balance (include/exclude pending)

Non-functional requirements:

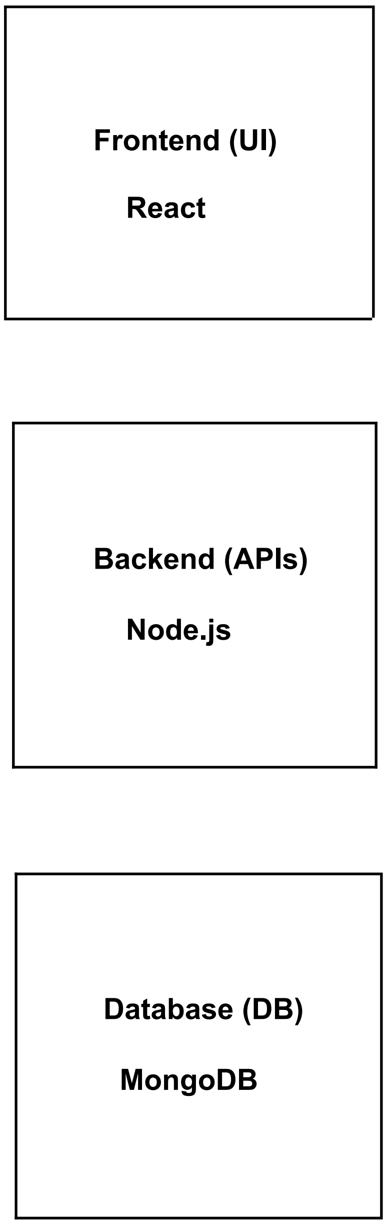
- Secure API access (auth tokens)
- Consistent, date-safe calculations (timezone-agnostic storage)
- Audit trail for changes (who approved/rejected and when)
- Reasonable performance and reliability for 50→500 employees

1. **High-Level Architecture Diagram:**

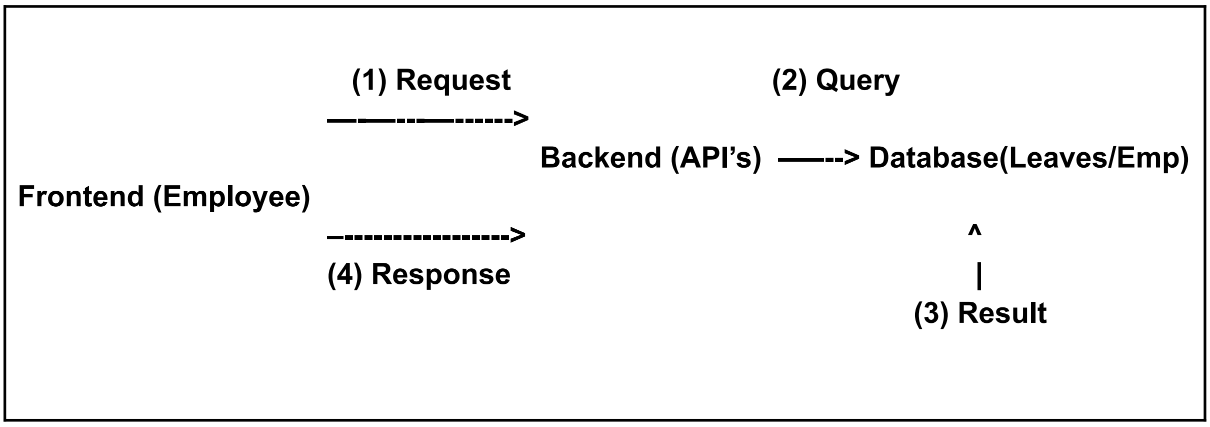
The Frontend (React) provides the user interface for employees and managers to interact with the system.

The Backend (Node.js APIs) handles business logic, authentication, and leave approval workflows.

The Database (MongoDB) stores employee details, leave records, and role-based access information.

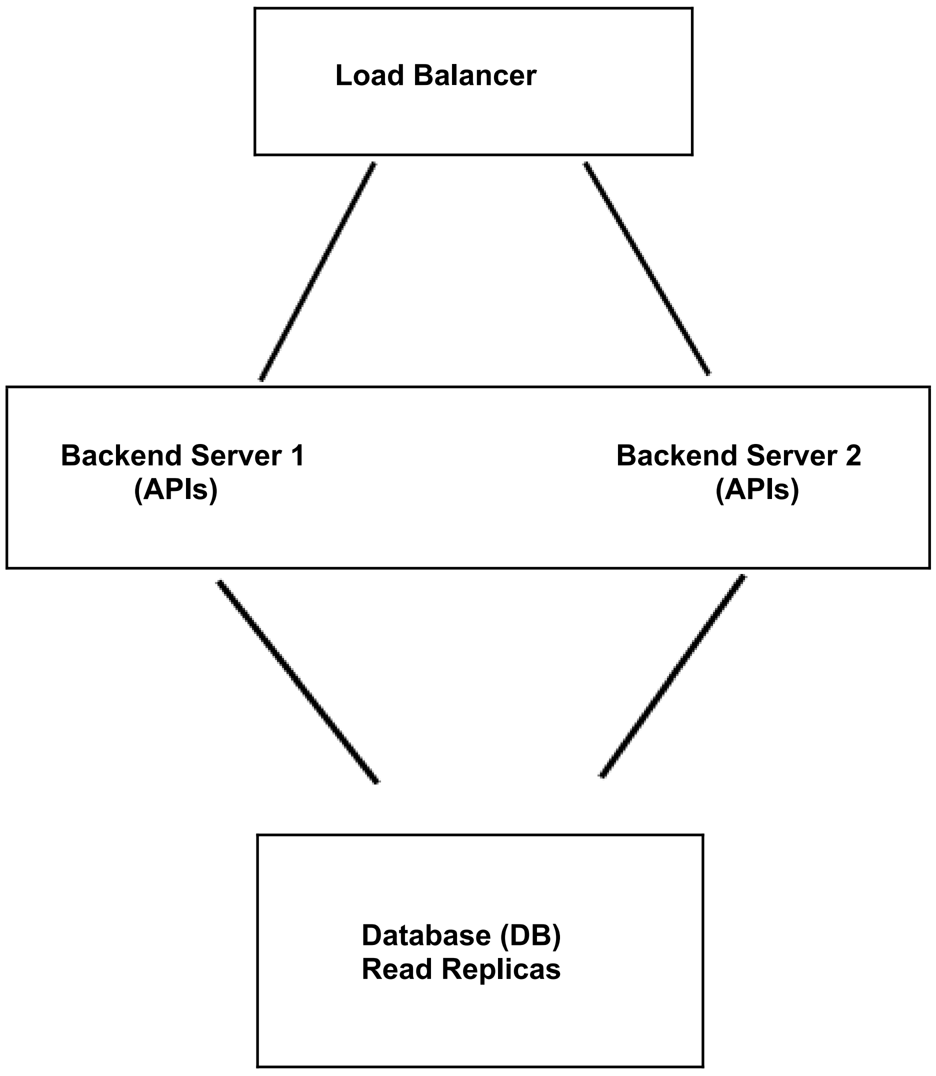


2. API & Database Interaction Flow :



Step (1): The employee frontend sends a leave request through the UI.
Step (2): The backend (Node.js APIs) converts this request into a query to MongoDB.
Step (3): The database returns results (e.g., employee's leave balance, approvals).
Step (4): The backend sends the final response back to the frontend for display.

3. Scaling from 50 → 500 Employees :



The Load Balancer evenly distributes incoming traffic across multiple backend servers.
Multiple Backend Servers (Node.js) ensure fault tolerance and high availability.
The MongoDB Database handles primary data storage, while read replicas are used to improve read performance and scalability as the number of employees grows.
This setup allows the system to scale from 50 employees to 500+ without performance degradation.