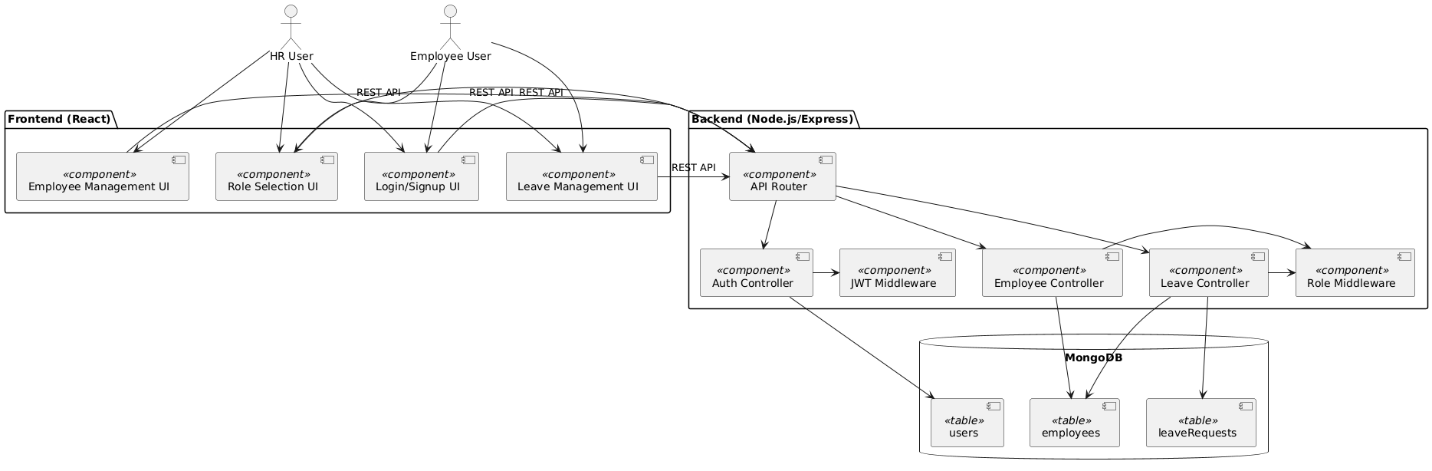
Part 2 — High Level System Design

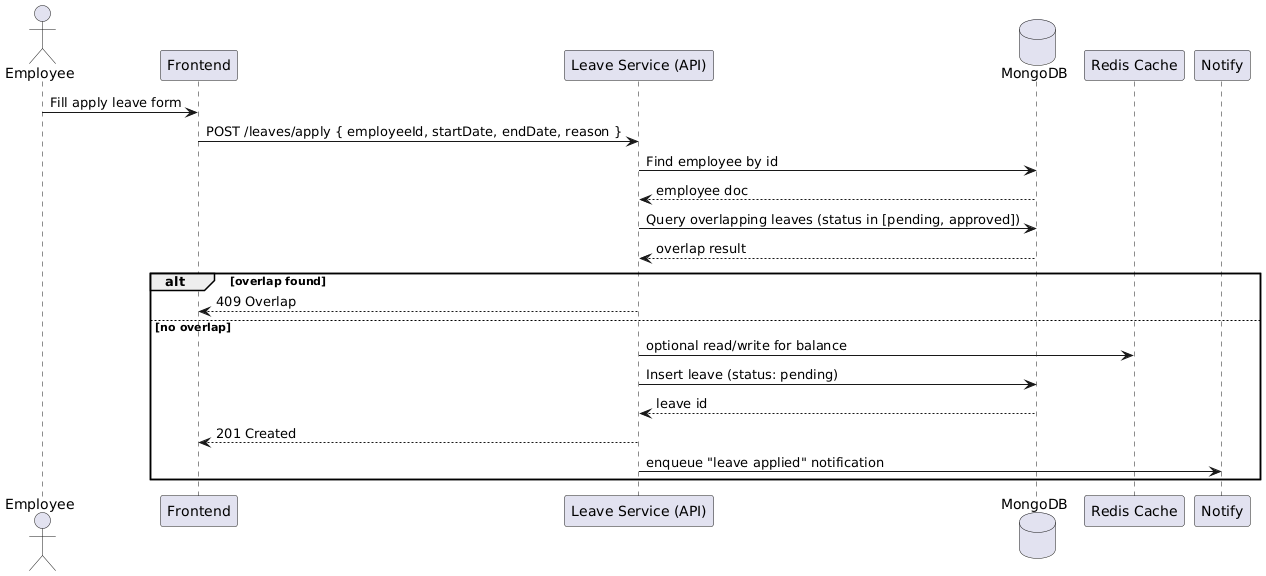
● Architecture diagram (Frontend, Backend, Database).



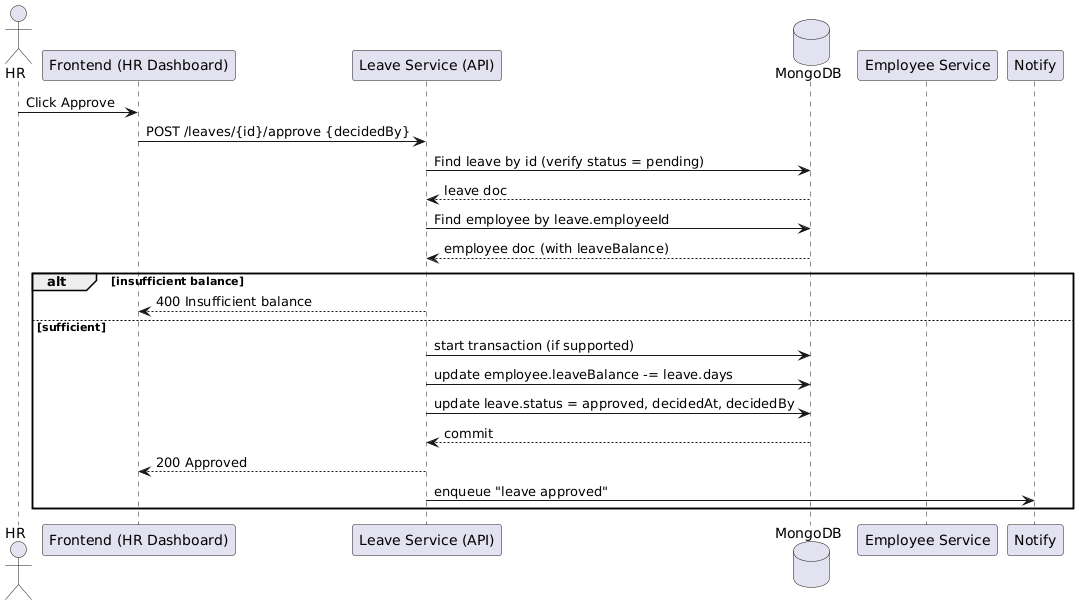
● How APIs and DB interact.

* The React frontend sends HTTP requests (using fetch) to the Node.js/Express backend REST API.
* The backend authenticates requests (JWT), checks user roles, and processes business logic.
* For data storage and retrieval, the backend uses Mongoose to interact with MongoDB collections (users, employees, leaveRequests).
* Example: When an employee applies for leave, the frontend sends a POST request to /leaves/apply, the backend validates and stores the request in the leaveRequests collection.

**Apply Leave API**: frontend POSTs to /leaves/apply. Backend validates dates, joining date, checks overlaps (query leaves with date range + statuses), checks balance, and inserts a pending leave document. Notifications queued asynchronously.



**Approve Leave API**: HR calls /leaves/{id}/approve. Backend re-reads leave & employee (to avoid stale checks), performs a transactional update: deduct employee’s leaveBalance and set leave status=approved. If DB supports transactions (Mongo replica set / Atlas), wrap updates in a transaction to keep balance and leave atomically consistent.



● How scaling will be handled if the company grows from 50 → 500 employees.

**1. Frontend (React):**

* Serve as static files via **CDN (Content Delivery Network)** or platforms like **Vercel/Netlify**.
* Auto-scalable, no state management required on servers.

**2. Backend (Node.js/Express):**

* **Containerized with Docker**, deployed behind a **Load Balancer** (AWS ELB / Nginx).
* **Horizontal Scaling** → Run multiple backend instances in parallel to handle growing requests.
* **Stateless** (authentication with JWT), so any instance can serve any request.

**3. Database (MongoDB):**

* Start with a **replica set** for high availability.
* **Vertical scaling**: Increase CPU/RAM as initial growth happens.
* **Horizontal scaling**: Move to **MongoDB sharding** when employee and leave data grow large.
* Prefer **MongoDB Atlas** for automated scaling, backups, and monitoring.

**4. Caching (Redis):**

* Cache frequently accessed data (e.g., leave balances, employee profiles).
* Reduces load on the database during peak times.

**5. Monitoring & Auto-Scaling:**

* Use **PM2** for process management.
* Integrate monitoring tools (**Datadog, New Relic, AWS CloudWatch**) for real-time performance tracking.
* Configure **auto-scaling rules** (e.g., spin up new backend containers when CPU > 70%).