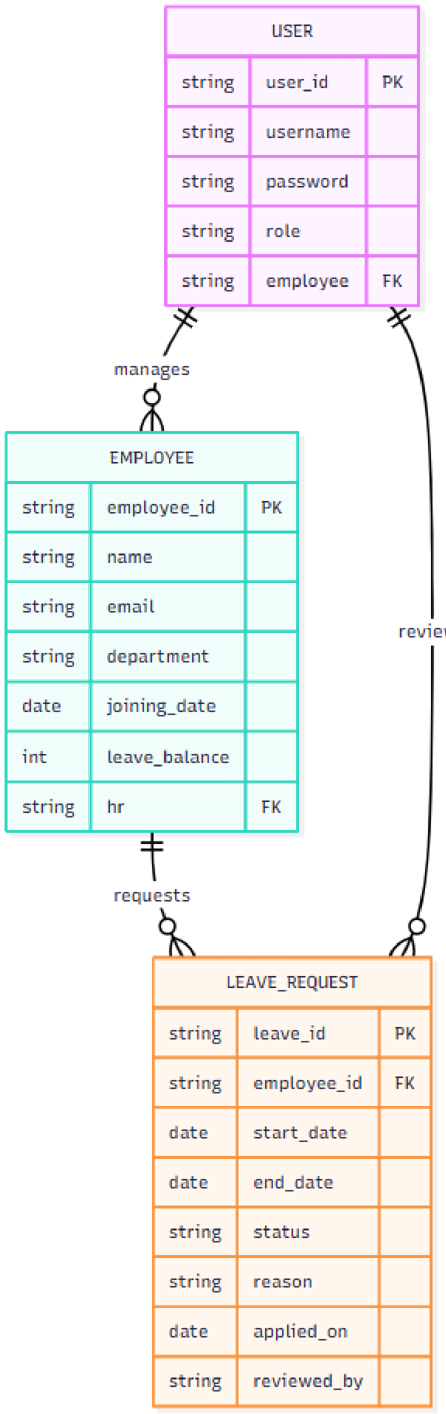
Part 1 — Data Modelling

● Create a basic ER diagram for the system.



Entities & Attributes

User

- _id (PK)
- username (unique)
- password
- role ('hr' or 'employee')
- employee (FK, nullable, references Employee._id)

Employee

- _id (PK)
- name
- email (unique)
- department
- joining_date
- leave_balance
- hr (FK, references User._id where role='hr')

LeaveRequest

- _id (PK)
- employee_id (FK, references Employee._id)
- start_date
- end_date
- status ('pending', 'approved', 'rejected')
- reason
- applied_on
- reviewed_by (FK, references User._id where role='hr', nullable)

Relationships

- One **HR** (User, role='hr') manages many **Employees** (Employee.hr).
- One **Employee** can have one **User** account (User.employee).
- One **Employee** can have many **LeaveRequests**.
- Each **LeaveRequest** is reviewed by one HR (optional, for audit).

● Define database tables for employees, leaves, and leave transactions.

HRs (users collection, role: 'hr')

Field	**Type**	**Key**	**Description**
_id	ObjectId	PK	HR unique ID
username	String	Unique	HR login
password	String		Hashed password
role	String		'hr'

Employees

Field	**Type**	**Key**	**Description**
_id	ObjectId	PK	Employee unique ID
name	String		Employee name
email	String	Unique	Employee email
department	String		Department
joining_date	Date		Joining date
leave_balance	Number		Leave balance
hr	ObjectId	FK → HR	Managed by HR

LeaveRequests

Field	**Type**	**Key**	**Description**
_id	ObjectId	PK	Leave request ID
employee_id	ObjectId	FK → Employee	Employee
start_date	Date		Leave start date
end_date	Date		Leave end date
status	String		pending/approved/rejected
reason	String		Reason for leave
applied_on	Date		Date applied

● Specify keys, relationships, and indexes.

- **Primary Keys:**
    - _id for each collection.
- **Foreign Keys:**
    - hr in Employee → references HR (users collection, role: 'hr').
    - employee_id in LeaveRequest → references Employee.
- **Indexes:**
    - email in Employee: unique index.
    - username in User: unique index.
    - employee_id in LeaveRequest: index for fast lookup.
    - hr in Employee: index for HR’s employees.

Part 2 — Low Level System Design

● API contracts (request/response format).

POST /leaves/apply

- **Request:**

```
{
  "employee_id": "ObjectId",
  "start_date": "YYYY-MM-DD",
  "end_date": "YYYY-MM-DD",
  "reason": "string"
}
```

- **Response:**

```
{
  "_id": "ObjectId",
  "employee_id": "ObjectId",
  "start_date": "YYYY-MM-DD",
  "end_date": "YYYY-MM-DD",
  "status": "pending",
  "reason": "string",
  "applied_on": "YYYY-MM-DD"
}
```

POST /leaves/:id/approve

- **Request:** (HR only, path param: leave ID)
- **Response:**

```
{ "message": "Leave approved." }
```

GET /leaves/my

- **Response:**

```
[
 {
   "_id": "ObjectId",
   "start_date": "YYYY-MM-DD",
   "end_date": "YYYY-MM-DD",
   "status": "pending",
   "reason": "string"
 }
]
```

● Class/module design (e.g., LeaveService, EmployeeService).

EmployeeService

- createEmployee(data)
- getEmployeeById(id)
- listEmployeesByHR(hrId)

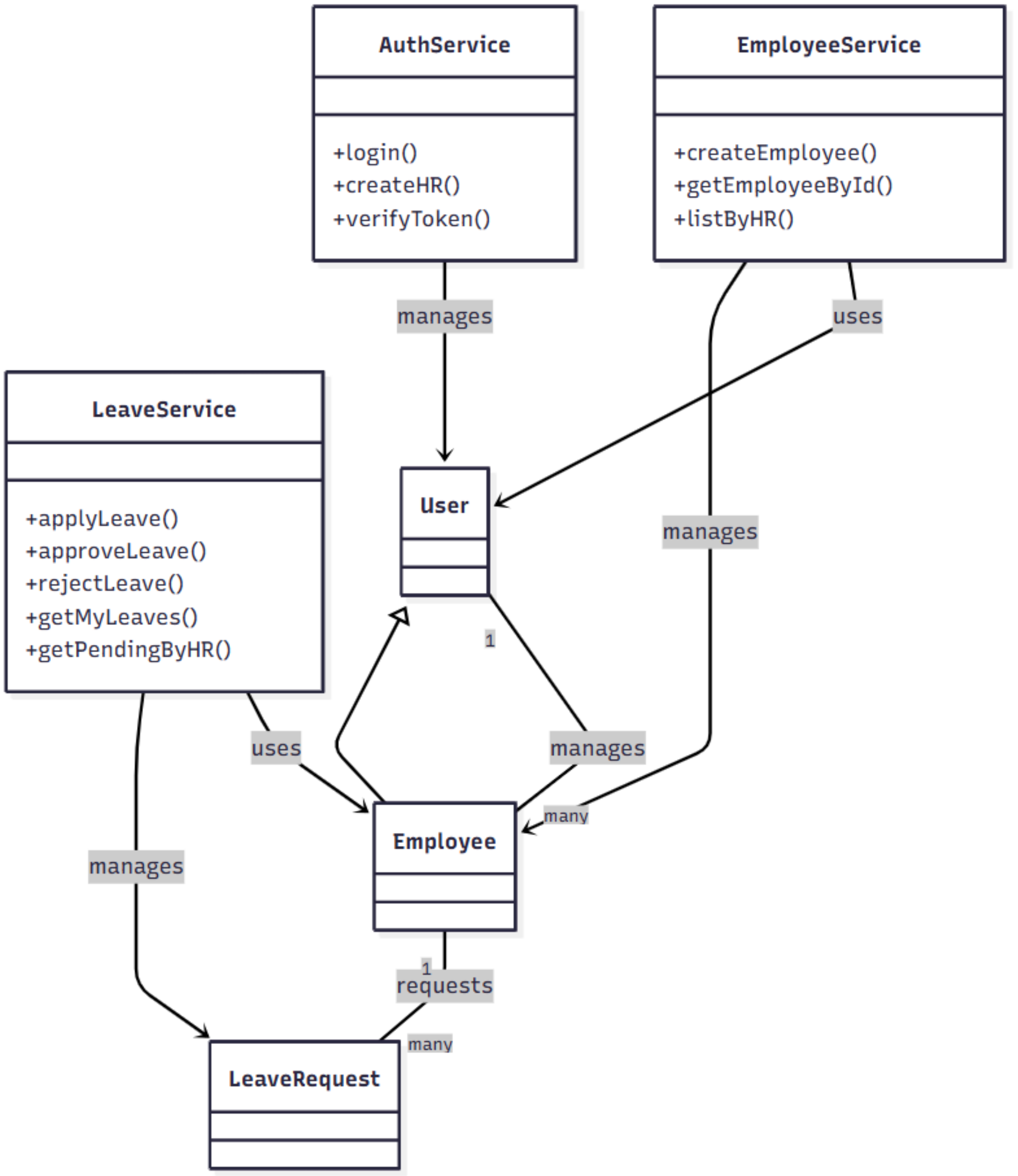
LeaveService

- applyLeave(employeeId, start, end, reason)
- approveLeave(leaveId, hrId)
- rejectLeave(leaveId, hrId)
- getLeavesByEmployee(employeeId)
- getPendingLeavesByHR(hrId)

AuthService

- login(username, password)

- createHR(data)



● Pseudocode for leave approval logic.

```
function approveLeave(leaveId, hrId):
   leave = LeaveRequest.findById(leaveId)
   if not leave or leave.status != 'pending':
       return error("Invalid leave request")
   employee = Employee.findById(leave.employee_id)
   if employee.hr != hrId:
       return error("Not authorized")
   if leave duration > employee.leave_balance:
       return error("Insufficient balance")
   leave.status = 'approved'
   leave.save()
   employee.leave_balance -= leave duration
   employee.save()
   return success("Leave approved")
```