Practical Test: Leave Management System

Objective:

Build a Leave Management System using Laravel (backend with REST API), React (frontend), and MySQL. Include features for both Employee and Admin with web and API access.

Features to Implement

User Roles

1. Employee
   - Register/Login
   - Apply for leave
   - View leave status

2. Admin
   - Login
   - View all leave requests
   - Approve/Reject leave
   - View users & leave statistics

API Endpoints (Laravel)

- POST /api/login

- POST /api/register

- GET /api/leaves

- POST /api/leaves

- PUT /api/leaves/{id}

- GET /api/user

Web Views (React)

- Login/Register Page

- Employee Dashboard: Leave application form, Leave status list

- Admin Dashboard: Leave approval list, Approve/Reject buttons, Users overview

Tech Stack

Backend

- Laravel 10+
- MySQL
- Laravel Passport

Frontend

- React (Vite or CRA)
- Axios
- React Router
- Tailwind CSS (optional)

Extra Requirements

- Use Laravel Form Requests for validation

- Include meaningful validation messages

- Use Laravel Seeders to pre-populate data

- Include error handling and loading indicators in React

- Add role-based access control (middleware/auth guards)

Deliverables

Upload the entire project to a public GitHub repository with the following structure and README files.

README.md must include:

- Project overview

- Setup instructions (backend and frontend)

- API documentation (list of endpoints + request/response examples)

- Database credentials (localhost config)

- Seeder instructions (php artisan db:seed)

- Default credentials:

Admin:
Email: admin@example.com
Password: password

Employee:
Email: employee1@example.com
Password: password

Folder Structure (Recommended)

```
leave-management-system/
├── backend/ (Laravel)
│   └── README.md
├── frontend/ (React)
│   └── README.md
└── README.md (master - summary)
```

Evaluation Criteria

Functionality: All features working as expected

UI/UX: Clean and attractive frontend (extra for Tailwind or similar)

Security: Auth, roles, route protection

Validations: Proper backend + frontend validations

Code Quality: Organized, commented code

Documentation: Clear README with setup & usage instructions

REST API: Clean and well-documented API usage

Bonus: Responsive design, dashboard charts, etc.