I) OBJECTIVE

Build a simple dashboard-style backend system for managing employee leave requests. The system should support basic CRUD operations and a workflow for submitting and approving leave.

To build a simple dashboard-style backend system for managing employee leave requests with CRUD and approval workflow- using Laravel, you can follow these steps.

Requirements Summary

1. Core features:

1. Employee and Admin roles
2. Employee can submit leave requests
3. Admin can approve/reject leave
4. Basic CRUD (Create, Read, Update, Delete) operations
5. Status workflow: Pending → Approved/Rejected
6. Dashboard showing leave statistics

2. System Design

Entities:

1. User: (id, name, email, password, role [employee/admin])
2. LeaveRequest: (id, user\_id, leave\_type, start\_date, end\_date, reason, status [pending/approved/rejected], admin\_comment).

3. Implementation (Laravel Example) –Using Bash or API Endpoints

Step 1: Setup Laravel Project

composer create-project laravel/laravel leave-management

cd leave-management

php artisan migrate

Step 2: Authentication & Roles

composer require laravel/breeze --dev

php artisan breeze: install

npm install && npm run dev

php artisan migrate

Add a role column to users:

migration $table->string('role')->default('employee'); or 'admin'

Step 3: Create Models & Migrations

php artisan make:model LeaveRequest -m

Leave\_requests table: Using PHP

Schema::create('leave\_requests', function (Blueprint $table) {

$table->id();

$table->foreignId('user\_id')->constrained()->onDelete('cascade');

$table->string('leave\_type');

$table->date('start\_date');

$table->date('end\_date');

$table->text('reason')->nullable();

$table->string('status')->default('pending');

$table->text('admin\_comment')->nullable();

$table->timestamps();

});

Then: using bash

php artisan migrate

Step 4: Create Controller & Routes- Using Bash

php artisan make:controller LeaveRequestController

Routes (web.php):

Route::middleware(['auth'])->group(function () {

Route::resource('leaves', LeaveRequestController::class);

Route::post('leaves/{id}/approve',

[LeaveRequestController::class, 'approve']);

Route::post('leaves/{id}/reject',

[LeaveRequestController::class, 'reject']); });

Step 5: Logic in Controller

public function store(Request $request) {

LeaveRequest::create([

'user\_id' => auth()->id(),

'leave\_type' =>

$request->leave\_type, 'start\_date' =>

$request->start\_date, 'end\_date' => $request->end\_date,

'reason' => $request->reason,

'status' => 'pending',

]); }

public function approve($id) {

$leave = LeaveRequest::findOrFail($id);

$leave->update(['status' => 'approved']);

}

public function reject(Request $request, $id) {

$leave = LeaveRequest::findOrFail($id);

$leave->update(['status' => 'rejected', 'admin\_comment' => $request->admin\_comment]);

}

4. Workflow / Dashboard Logic

For Manager;

a) Submit: Employee submits a request → status = pending

b) Review: Manager views pending requests

c) Action: Manager approves/rejects → status changes

For Employee:

1. Form to submit leave
2. List of personal leave history
3. Optional Enhancements
4. Notifications/email when leave is approved/rejected
5. Export leave reports to Excel
6. Leave quota system
7. REST API endpoints

II) CORE FEATURES

\* Employees can submit leave requests.

\* Admins or managers can view, approve, or reject requests.

\* Each leave request should have a status (e.g., Pending, Approved, and Rejected).

\* Include any supporting entities you believe are necessary (e.g., users, roles, departments).

The core features for the leave management system based on my latest input:

A) CORE FEATURES:

1. Leave Request Management

1. Submit Leave: Employees can create leave requests.
2. Leave Status: Each request has a status field:
   * 1. Pending (default on submission)
     2. Approved (set by manager/admin)
     3. Rejected (set by manager/admin)
     4. Setup models, migration, and relationships.
     5. Add role-based access ( employee and admin )
     6. Build the logic for leave submission, status updates, and view filtering.

2. Approval Workflow

1. View Requests: Admins/Managers can view all leave requests.
2. Approve/Reject: Admins/Managers can change the status of any request.

B) SUPPORTING ENTITIES

1- User:

id: Unique identifier

name: Full name

email: Login/notification purposes

role: Enum (Employee, Manager, Admin)

department\_id: FK to Department (optional but useful)

2- Leave Request

id: Unique identifier

user\_id: FK to User (who requested)

start\_date: Start of leave

end\_date: End of leave

leave\_type: Enum (Vacation, Sick, Unpaid, etc.)

reason: Optional text

status: Enum (Pending, Approved, Rejected)

created\_at: Timestamp

3- Department (Optional, useful for filtering)

id: Unique identifier

name: "Engineering", "HR", "Finance".

C) Optional Enhancements

Audit Log: Track who approved/rejected and when.

Leave Balance Tracking: Track remaining leave days per employee.

Notifications: Email or dashboard alerts on request status changes.

III) DELIVERABLES

\* A working Laravel-based application.

\* Authenticated access with basic role separation (e.g., employee vs admin).

\* Seeded data for testing.

\* A README file that includes:

o Setup instructions

o Architectural choices

o Any assumptions or trade-offs made

To meet the deliverables for my Laravel-based leave management system, follow the below structure:

1. Working Laravel Application;

Ensure your Laravel app has the following:

* Core features: leave requests CRUD, approval workflow
* Dashboard for admin and employee
* Proper routing and controllers
* Working forms/UI (Blade, Vue, etc.)
* Use: php artisan serve to run it locally.

2. Authenticated Access with Role Separation

You should already have this:

* Laravel Breeze or Laravel UI for login/register.
* Install Breeze for authentication:

composer require laravel/breeze --dev

php artisan breeze:install

npm install && npm run dev

php artisan migrate

* Role-based access (Employee, Admin) via middleware or policies.
* Add role column to users table: 'employee' or 'admin'
* Use middleware or conditionals in controllers/blades:

if (auth()->user()->role === 'admin') { // admin logic }

Optionally, create middleware IsAdmin and use it in routes.

* Leave Request System (with various statuses):
* CRUD operations for leave requests (employees).
* Approval/rejection system (admin).

LeaveRequest::create ([

'user\_id' => 2, // Ahmad Omari

'leave\_type' => 'Sick',

'start\_date' => '2025-06-01',

'end\_date' => '2025-06-03',

'reason' => 'Flu', ]);

* User Roles & Management:
* Seeder creates default users with roles.
* Dashboard View:
* Admin: View/filter all requests.
* Employee: View own requests and statuses.

3. Seeded Data for Testing;

Create seeders for:

1. Departments (if included);

DB::table('departments')->insert ([

['name' => 'IT'],

['name' => 'HR'],

]);

1. Users (employees + at least one admin)- Using PHP code

User::create(

[ 'name' => 'Admin User',

'email' => 'admin@ limitless-tech.ai,

'password' => Hash::make('password'),

'role' => 'admin', ]);

User::create([ 'name' => 'Ahmad Omari',

'email' => 'Ahmad@limitless-tech.ai,

'password' => Hash::make('password'),

'role' => 'employee',

]);

3. README.md

Should include:

1. Setup Instructions
   * Requirements (PHP, Composer, MySQL)

* .env config
* Commands to run: Using Bash

composer install

php artisan migrate --seed

php artisan serve

Users for Testing

* Admin: admin@ limitless-tech.ai / password
* Employee: 'Ahmad@limitless-tech.ai / password

Architectural Choices

* Laravel Breeze for lightweight auth
* Role-based access via middleware vs policies
* Blade templates for simplicity
* Eloquent relationships between users and leave requests
* Role string in users table to avoid full RBAC complexity

Assumptions & Trade-offs

* Only two roles (admin, employee) for simplicity
* Leave quota or overlap checks not yet implemented
* UI kept minimal to focus on backend logic

IV) GIT & COMMIT REQUIREMENTS

\* Use Git from the start of the project.

\* All work should be committed in logical steps with clear, descriptive commit messages.

\* We will review your Git history as part of the assessment.

To meet the Git & Commit Requirements, following the below steps for beginning the project:

Git & Commit Requirements:

Version Control:

Use Git as the version control system from the very beginning of the project. Ensure all files are properly tracked and committed.

1. Initialize Git from the Start;

In the Laravel project root – Using Bash : git init

Then create a .gitignore file (Laravel includes one by default). If not:

/vendor/ /node\_modules/ /.env /public/storage /storage/\*.key

2. Connect to GitHub (Optional);

If pushing to GitHub:

Git remote add origin

<https://github.com/Laravel411/Leave-management.git>

3. Make Logical Commits with Clear Messages and avoid vague messages like fixed stuff or update;

Follow this commit strategy:

* Initial Setup

Git add

Git commit -m "Initial Laravel setup with Breeze and auth"

* Database Design

Git add database/migrations

Git commit -m "Add migrations for departments, users (with roles), and leave\_requests"

* Seeders

Git add database/seeders

Git commit -m "Add seeders for departments, users, and leave requests"

* Core Features

Git add app/Models app/Http/Controllers

Git commit -m "Implement LeaveRequest model, controller, and basic CRUD logic"

* Role-Based Access

Git commit -m "Add role checks for admin vs employee in LeaveRequestController"

* UI/Views

Git commit -m "Add Blade templates for submitting and managing leave requests"

* Testing

Git commit -m "Add feature tests for leave submission and approval"

4. Tips for Good Git History;

* Commit after each logical block of work
* Avoid large, sweeping commits
* Use clear verbs: Add, Fix, Update, Refactor, and Implement, And Remove.

5. Push to GitHub (If applicable);

Git branch -M main

Git push -u origin main

V) TIME LIMIT

This is a time-boxed task. Please aim to spend no more than 6-8 hours. It is okay if some areas are incomplete - just note them in your README.

To effectively manage a time-boxed task (6–8 hours) for your Laravel leave management system, follow this structured plan:

Step-by-Step Breakdown (Time Management Guide):

A) Hour 0–1: Project Setup

* Initialize Laravel project
* Install auth (Laravel Breeze or Jetstream)
* Set up .env and DB connection
* Initialize Git and first commit: Git commit -m "Initial Laravel project setup with auth"

B) Hour 1–2: Data Modelling

* Create migrations:
* departments
* Add role to users
* leave\_requests
* Define Eloquent relationships
* Commit:
* Git commit -m "Seed test data and add basic UI structure"

C) Hour 3–5: Core Functionality

* Employees: create leave requests
* Admins: view and approve/reject requests
* Leave status flow
* Role-based access control
* Commit:

Git commit -m "Implement leave request creation and approval"

Git commit -m "Add admin-only access for approval routes"

D) Hour 5–6: UI Polishing + Clean-up

* Basic Blade templates or Livewire/Vue components
* Add feedback messages (success/error)
* Sort/filter leave requests in dashboard
* Commit: git commit -m "Improve leave request views and add feedback UI"

Hour 6–8: Final Touches

* Add README.md:
* Setup instructions
* Users for testing
* Architectural choices
* Final bug fixes or UI polishing
* Commit: Git commit -m "Add README and finalize project"

README Note for Incomplete Areas

If time runs out:

Known Limitations / Incomplete Areas - Leave quota system is not implemented - No email notifications on approval/rejection –feature that are stubbed and planned but not coded - Validation and unit tests are basic/minimal – UI/UX design is functional but not styled.

E) Final Tip

Use a timer and stick to the plan Prioritize:

* Working core features
* Clean role separation
* Good Git history
* Clear README

VI) WHAT WE ARE LOOKING FOR

\* Laravel project structure and organization

\* Use of Laravel-specific features

\* Database design and model relationships

\* Code quality and maintainability

\* Git usage and commit clarity

\* (Bonus) Testing, policies, and optional UI implementation

During the review of this project, we will assess the following criteria:

1. Laravel Project Structure & Organization

What to Do:

* Stick to Laravel conventions:
* Models in app/Models
* Controllers in app/Http/Controllers
* Views in resources/views
* Routes in routes/web.php
* Keep your folder structure clean and modular, controllers, requests , policy (avoid bloated controllers)
* Use resource routes and controller methods logically (index, store, update)

2. Use of Laravel-Specific Features

What we Use:

* Auth: Laravel Breeze or Jetstream
* Validation: Use FormRequest classes

(php artisan make:request StoreLeaveRequest )

* Eloquent Relationships: example, User hasMany LeaveRequest, department has many users.
* Blade Components (if using Blade templates)
* Route model binding ( Model factories and seeders for test data) - using php

public function show (LeaveRequest $leaveRequest)

3. Database Design and Model Relationships

The Expectation is:

* Normalized DB structure by using the migrations to define clear, normalized structure.
* Foreign keys & constraints
* Proper use of hasMany, belongsTo
* Clear seeder data for testing, include timestamps , softdeletes if useful

Example:

User.php public function leaveRequests() { return $this->hasMany(LeaveRequest::class); }

Ensure:

leave\_requests.user\_id is a foreign key

Seeding is repeatable and minimal

4. Code Quality and Maintainability

What to Focus On:

* Clear, meaningful method and variable names
* DRY (Don’t Repeat Yourself)
* Small, testable methods
* Comments only where necessary (code should be self-explanatory)
* Use custom fromRequest classes for cleaner controllers.

Suggested Tools:

* PHPStan or Laravel Pint (for static analysis and formatting)
* php artisan route:list to check route cleanliness

5. Git Usage and Commit Clarity

How to Proceed:

* Commit after every meaningful change
* Use clear, active voice messages: Add leave request form for employees Restrict approval routes to admins only Refactor LeaveRequestController into service class
* Push to GitHub with a clean commit history

Avoid:

* Huge single commits
* Messages like “final fix” or “wip”

6. (Bonus) Testing, Policies, UI

Optional Enhancements:

* Testing:

php artisan make:test LeaveRequestTest

Write a basic test for submitting a leave request and checking status.

* Authorization Policies:

php artisan make:policy LeaveRequestPolicy --model=LeaveRequest

Control who can approve, reject, etc.

* Optional UI:
* Use Tailwind CSS if using Breeze
* Simple and clean dashboard
* Flash messages for feedback

VII) SUBMISSION

Please share a GitHub repository link along with setup instructions in your README file. Provide the required through [hello@limitless-tech.ai](mailto:hello@limitless-tech.ai)

The Final submission:

1. Push Your Laravel Project to GitHub

We need to Initialize – Using Bash

Git init

Git remote add origin

<https://github.com/Laravel411/Leave-management.git>

Git add

Git commit -m "Initial commit with leave management system"

Git branch -M main

Git push -u origin main

We should to make sure our repository is public, or invite reviewers if it's private.

2. Ensure our README Includes Setup Instructions

Minimum Checklist in README.md:

Laravel Leave Management System

Setup Instructions

1. Clone the repository – using bash

Git clone

https://github.com/myname-username/leave-management.git

cd leave-management

1. Install dependencies:

composer install

npm install && npm run dev

1. Set up environment - Bash

cp .env.example .env

php artisan key:generate

1. Configure .env with your DB credentials
2. Run migrations and seeders: Bash

php artisan migrate --seed

1. Serve the application: bash

php artisan serve

3- Email Submission and Admin Credentials:

Email: admin@limitless-tech.ai

Password: password

[hello@limitless-tech.ai](mailto:hello@limitless-tech.ai)

Subject: Leave Management System

Submission – Ahmad Omari

Hello,

I am submitting my Laravel-based Leave Management System as requested.

GitHub Repository:

<https://github.com/Laravel411/Leave-management.git>

The project supports employee leave requests with admin approvals, basic authentication, and role separation. Setup instructions and credentials are in the README file.

Thank you!

Best regards,

Ahmad Omari