# 📝 Leave Management System (LMS)

A lightweight \*\*Leave Management System\*\* built with a clean architecture approach using C#. The system allows employees to apply for leave, and supports operations like view, approve, reject, update, and delete leave requests.

---

## 🏗️ Project Structure

```

Day5Proj2-LMS2/

├── Lms.Application/

│ └── Services/

│ └── LeaveService.cs

├── Lms.ConsoleApp/

│ └── Program.cs

├── Lms.Core/

│ ├── Entities/

│ │ └── Leave.cs

│ ├── Interfaces/

│ │ └── ILeaveRepository.cs

│ └── Program.cs

├── Lms.Infrastructure/

│ └── Repositories/

│ └── LeaveRepository.cs

```

---

## ⚙️ Technologies Used

- .NET 8.0

- C#

- Clean Architecture Principles

- Console App Interface

---

## 📦 Modules Overview

### 1. `Lms.Core`

Contains the \*\*domain layer\*\*:

- `Leave.cs`: The core entity representing a leave request.

- `ILeaveRepository.cs`: Interface defining the contract for leave-related operations.

### 2. `Lms.Infrastructure`

Implements the data layer:

- `LeaveRepository.cs`: In-memory implementation of `ILeaveRepository` using a `List<Leave>`.

### 3. `Lms.Application`

Holds the business logic:

- `LeaveService.cs`: Orchestrates the application flow — creating, fetching, approving, and rejecting leaves.

### 4. `Lms.ConsoleApp`

Acts as the UI (CLI):

- `Program.cs`: Entry point for the application.

---

## ✅ Features

- Apply for a leave

- View all leave applications

- Approve or reject a leave (simulated)

- Update leave status

- Delete a leave entry

---

## 🚀 How to Run

1. Clone or download the repository.

2. Open in Visual Studio.

3. Set `Lms.ConsoleApp` as the startup project.

4. Run the project (`Ctrl + F5` or `dotnet run`).

---

## 🧠 Core Classes Explained

### `Leave.cs` (Entity)

```

public class Leave {

public int Id { get; set; }

public required string NameOfEmployee { get; set; }

public required string TypeOfLeave { get; set; }

public required string Status { get; set; }

}

```

---

### `ILeaveRepository.cs` (Interface)

```

public interface ILeaveRepository {

void ApplyLeave(Leave leave);

void DeleteLeave(int id);

void UpdateLeave(int id, string status);

List<Leave> GetAll();

Leave GetById(int id);

}

```

---

### `LeaveRepository.cs` (In-memory Data Store)

Stores all leave data in a `List<Leave>` and implements `ILeaveRepository`.

---

### `LeaveService.cs` (Business Logic)

```

public class LeaveService {

private readonly ILeaveRepository \_repo;

public string Status { get; private set; }

public void CreateLeave(string noe, string type) { ... }

public List<Leave> GetAllLeaves() { ... }

public void ApproveLeave() => Status = "Approved";

public void RejectLeave() => Status = "Rejected";

}

```

---

## 📌 Example Output

```

Leave Applied: Casual

Fetching all Leave Details...

ID: 1, Name: Alice, Type: Sick, Status: Pending

Leave with ID 1 deleted.

```

---

## 📋 To-Do / Improvements

- [ ] Add persistence (e.g., SQLite or JSON file)

- [ ] Add validation and logging

- [ ] Add user role management (admin, employee)

- [ ] Create a simple GUI using Windows Forms or MAUI

---

## 👨‍💻 Author

Adhnan Jeff

💬 Feedback welcome!

---

## 📜 License

This project is open source and available under the [MIT License](LICENSE).