✅ Lab Exercise: Implementing Dependency Injection in ASP.NET Core Web API

🎯 Objective:

Learn how to apply dependency injection in a Web API Core project using real-world services like logging, notification, and repository layers.

🧪 Real-World Scenario:

You're developing an API for a **Parcel Delivery Tracking System**. The system must:

- Track parcel status
- Notify customers via email or SMS (simulated)
- Log delivery activities

The project must use **Dependency Injection** to decouple services like logging, notification, and data access.

🧰 Tech Stack:

- .NET 6 or later
- ASP.NET Core Web API
- In-Memory data for simplicity

☐ Lab Steps:

🔷 Step 1: Create a New Web API Project

```
dotnet new webapi -n ParcelTrackingAPI
cd ParcelTrackingAPI
```

🔷 Step 2: Define the Domain Models

Create a folder `Models` and add:

Parcel.cs

```
public class Parcel
{
    public int Id { get; set; }
    public string TrackingNumber { get; set; }
    public string Status { get; set; }
    public string CustomerEmail { get; set; }
}
```

🔷 Step 3: Create Interfaces for Services

Interfaces/INotificationService.cs

```
public interface INotificationService
{
    void Notify(string to, string message);
}
```

Interfaces/ILoggerService.cs

```
public interface ILoggerService
{
    void Log(string message);
}
```

Interfaces/IParcelRepository.cs

```
public interface IParcelRepository
{
    IEnumerable<Parcel> GetAll();
    Parcel GetById(int id);
    void UpdateStatus(int id, string status);
}
```

🔷 Step 4: Implement the Services

Create a folder `Services`.

Services/EmailNotificationService.cs

```
public class EmailNotificationService : INotificationService
{
    public void Notify(string to, string message)
    {
        Console.WriteLine($"[EMAIL to {to}] - {message}");
    }
}
```

Services/ConsoleLoggerService.cs

```
public class ConsoleLoggerService : ILoggerService
{
    public void Log(string message)
    {
        Console.WriteLine($"[LOG] - {message}");
    }
}
```

Services/InMemoryParcelRepository.cs

```
public class InMemoryParcelRepository : IParcelRepository
{
    private readonly List<Parcel> _parcels = new()
    {
        new Parcel { Id = 1, TrackingNumber = "T123", Status = "Shipped", 
CustomerEmail = "john@example.com" },
        new Parcel { Id = 2, TrackingNumber = "T124", Status = "In 
Transit", CustomerEmail = "jane@example.com" }
    };

    public IEnumerable<Parcel> GetAll() => _parcels;

    public Parcel GetById(int id) => _parcels.FirstOrDefault(p => p.Id == 
id);

    public void UpdateStatus(int id, string status)
    {
        var parcel = GetById(id);
        if (parcel != null) parcel.Status = status;
    }
}
```

🔷 Step 5: Register Services in DI Container

Update `Program.cs`:

```
builder.Services.AddScoped<INotificationService, 
EmailNotificationService>();
builder.Services.AddSingleton<ILoggerService, ConsoleLoggerService>();
builder.Services.AddSingleton<IParcelRepository, 
InMemoryParcelRepository>();
```

🔷 Step 6: Create the API Controller

Controllers/ParcelController.cs

```
[ApiController]
[Route("api/[controller]")]
public class ParcelController : ControllerBase
{
    private readonly IParcelRepository _repo;
    private readonly INotificationService _notifier;
    private readonly ILoggerService _logger;

    public ParcelController(IParcelRepository repo, INotificationService
notifier, ILoggerService logger)
    {
        _repo = repo;
        _notifier = notifier;
        _logger = logger;
    }

    [HttpGet]
    public IActionResult GetAll()
    {
        return Ok(_repo.GetAll());
    }

    [HttpPut("{id}/status")]
    public IActionResult UpdateStatus(int id, [FromQuery] string status)
    {
        var parcel = _repo.GetById(id);
        if (parcel == null)
            return NotFound();

        _repo.UpdateStatus(id, status);
        _logger.Log($"Parcel {id} status updated to {status}");
        _notifier.Notify(parcel.CustomerEmail, $"Your parcel status is now:
{status}");

        return Ok(parcel);
    }
}
```

🔷 Step 7: Run & Test

- Run the app

```
dotnet run
```

- Test with Swagger or Postman:
  - `GET /api/parcel`
  - `PUT /api/parcel/1/status?status=Delivered`

You should see logs and notifications printed in the console.

🧠 Learning Points Recap

- Registered and injected services using **DI Container**
- Followed **SOLID principles** by depending on abstractions
- Separated **concerns** between controller, data, logging, and notifications
- Used **scoped**, **singleton**, and **transient** appropriately

---

🧩 Bonus Challenges

1. Replace `EmailNotificationService` with an `SmsNotificationService` using `IConfiguration` to pick mode.
2. Implement `ILoggerService` using a file logger.
3. Make `IParcelRepository` fetch from a real DB using EF Core.