Web Development with ASP.NET Core 6

EXPERT NETWORK

The Sessions

1. Data Access
2. Concepts and Techniques
3. ASP.NET Core Introduction
4. ASP.NET Core Advanced
5. Deploy in the Cloud

*Note that each session builds upon the previous one.

EXPERT NETWORK

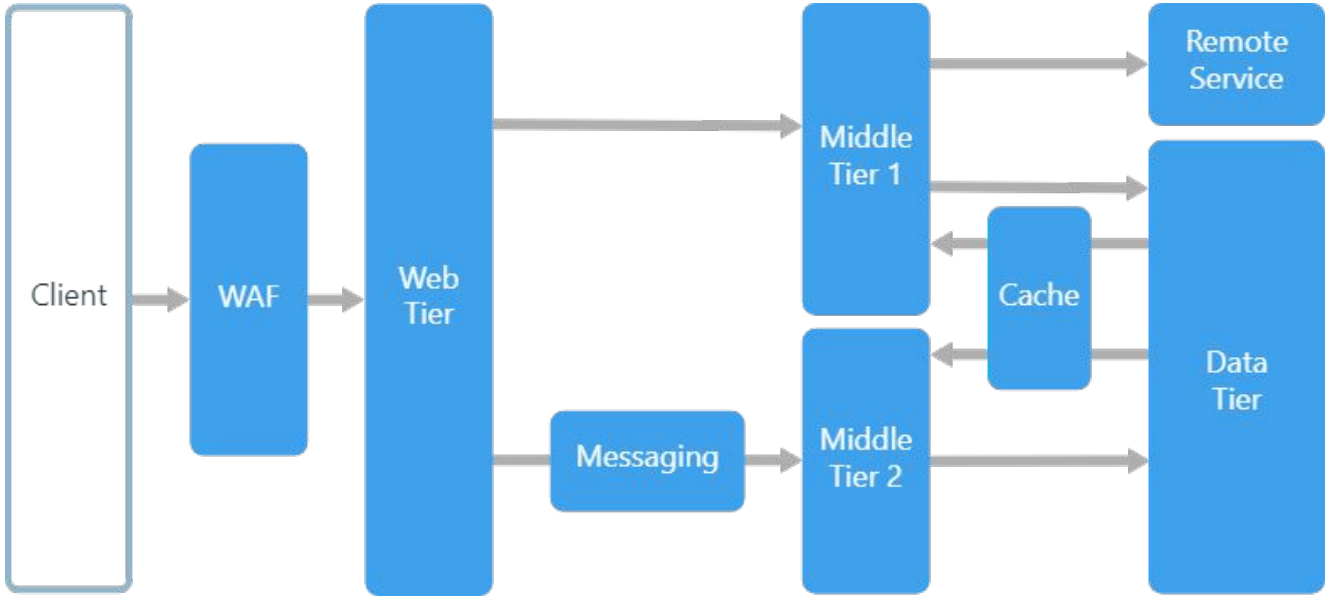
For this session

- N-tier architecture
- Services
- Repository
- Inversion of Control
- Dependency Injection

*We will start from https://github.com/AlexandruCristianStan/FII-Practic-EXN-2022.

EXPERT NETWORK

N-tier architecture



https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/n-tier

EXPERT NETWORK

N-tier architecture

An N-tier architecture divides an application into **logical layers** and **physical tiers**.

Layers are a way to separate responsibilities and manage dependencies. Each layer has a specific responsibility. **A higher layer can use services in a lower layer, but not the other way around.**

Tiers are physically separated, running on separate machines. Although each layer might be hosted in its own tier, that's not required. **Several layers might be hosted on the same tier.**

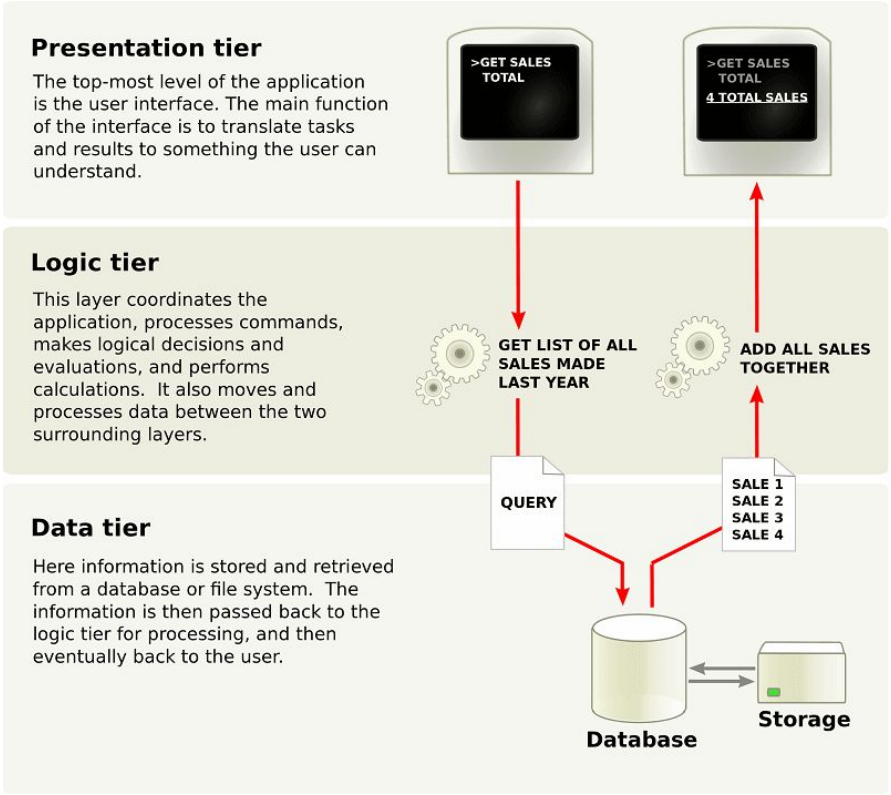
N-tier architecture

Benefits	Challenges
Less learning curve	It's easy to end up with a middle tier that just does CRUD
Easy to manage	Monolithic design prevents independent deployment of features
Should allow any of the three tiers to be upgraded or replaced independently	Sometimes you need to write a lot of code for simple operations

*This architecture can be used for simple web applications.

EXPERT NETWORK

Three-tier architecture



EXPERT NETWORK

Exercise 1: Setup Solution

1. Create a new Solution (*ExnCars*)
2. Add the Data, Services and Presentation projects
   ExnCars.Data - .net standard
   ExnCars.Services - .net standard
   ExnCars.TestConsole - console .net 6
3. Copy Entities and Context from the previous session
4. Make sure that Project References are set up

EXPERT NETWORK

Repository Pattern

- Is intended to create an abstraction layer between the data access layer and the business logic layer of an application.
- Help insulate your application from changes in the data store and can facilitate automated Unit Testing or test-driven development (TDD).

https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application

Repository Example

```
3 references | 0 changes | 0 authors, 0 changes
public interface IRepository<T>
    where T : class
{
    2 references | 0 changes | 0 authors, 0 changes
    void Add(T entity);
    1 reference | 0 changes | 0 authors, 0 changes
    void Update(T entity);
    1 reference | 0 changes | 0 authors, 0 changes
    void Delete(T entity);
    1 reference | 0 changes | 0 authors, 0 changes
    T? GetById(int id);
    3 references | 0 changes | 0 authors, 0 changes
    IQueryable<T> Query(Expression<Func<T,bool>> expression);
}
```

EXPERT NETWORK

Unit of Work

- The unit of work (UOW) class serves one purpose: to make sure that when you use multiple repositories, they share a single database context.

- When a unit of work is complete you can call the SaveChanges method on that instance of the context and be assured that all related changes will be coordinated.

https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application

Services

- Public Classes in which we write our business logic in order to reduce duplication (DRY principle) and promote reuse.
- They group methods that are used in the same feature (example: user management)
- Always declare Interfaces for services so that they we may reduce the coupling between them and the Presentation.
- These interfaces should have method that return or accept DTOs and only internally work with repositories.

DTO

A **Data Transfer Object** is an object that is used to encapsulate data, and send it from one subsystem of an application to another.

DTOs are most commonly used by the Services layer in an N-Tier application to transfer data between itself and the UI layer.

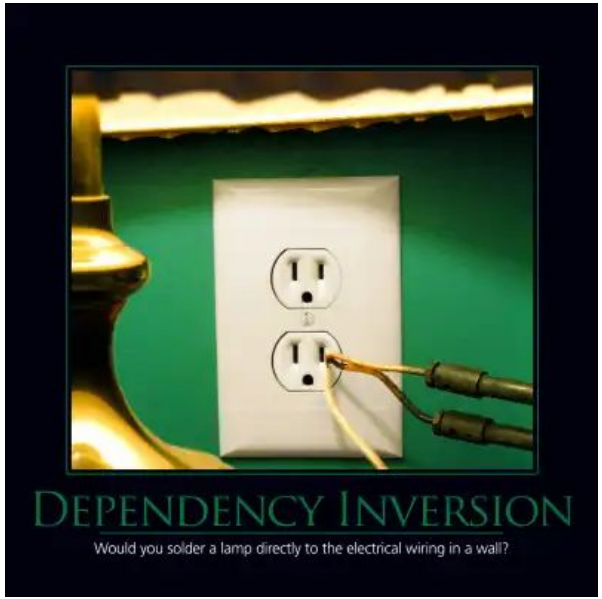
DTOs are simple objects that should not contain any business logic!

Exercise 2: Repository & UOW

1. Create a Generic Repository
2. Create Unit of Work class
3. Create a UserService that allows us to:
    - Get a user by email address
    - Register a user
    - Update user’s first name and last

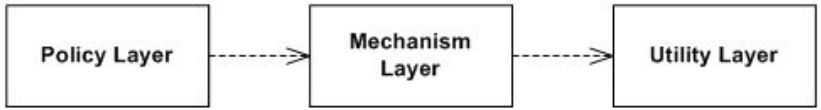
Dependency Inversion

The Dependency Inversion Principle (DIP) states that high level modules should not depend on low level modules; both should depend on abstractions. Abstractions should not depend on details. Details should depend upon abstractions.

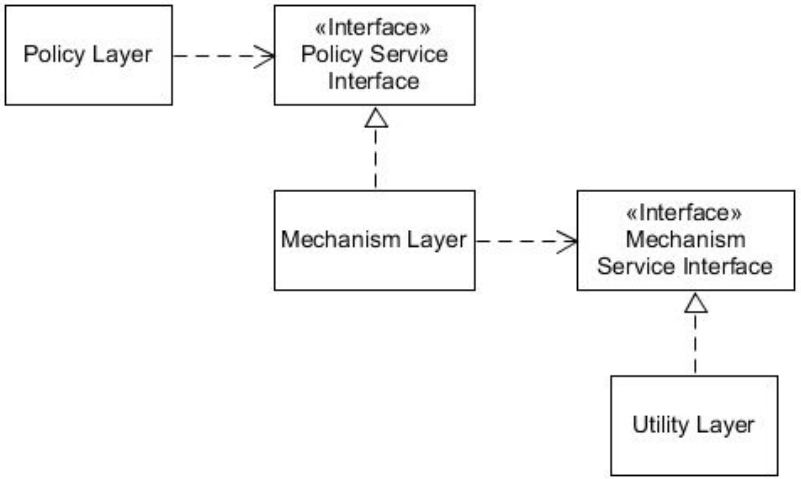


EXPERT NETWORK

Dependency Inversion

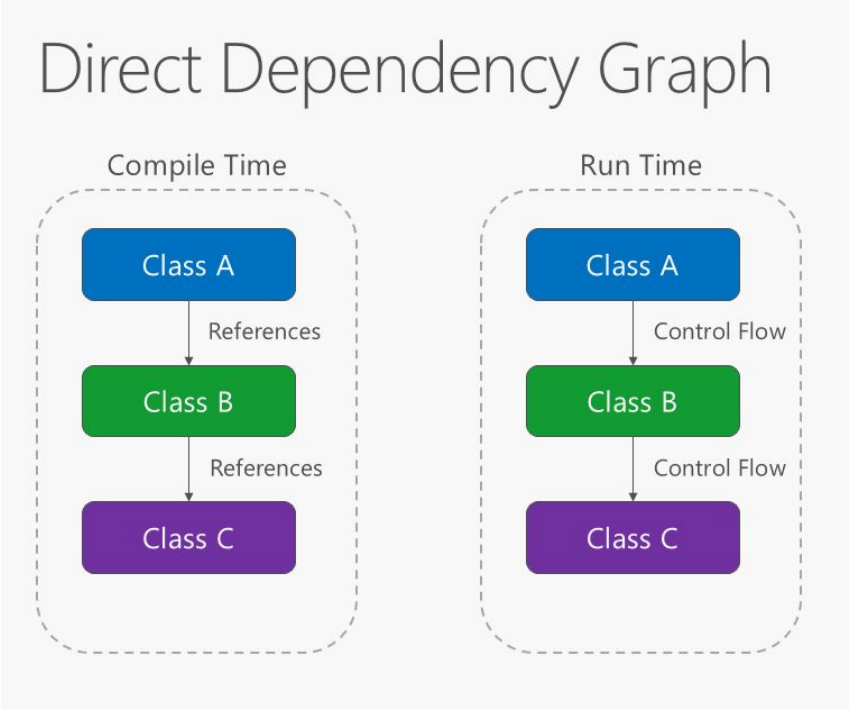


VS



EXPERT NETWORK

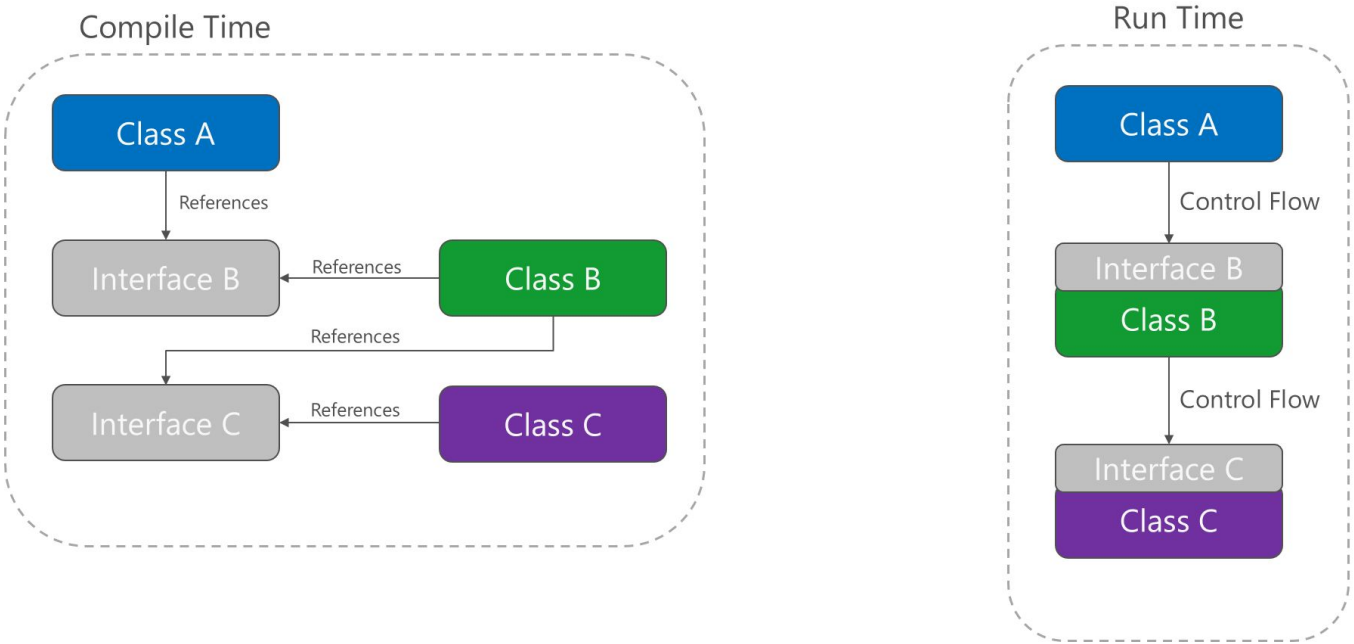
Dependency Inversion



EXPERT NETWORK

Dependency Inversion

Inverted Dependency Graph



EXPERT NETWORK

Dependency Inversion

- Avoid high coupling
- Encourage reusability of higher layers
- Simplify Unit Testing
- Makes your code easier to maintain

Dependency Injection

- The dependency injection (DI) software design pattern, which is a technique for achieving Inversion of Control (IoC) between classes and their dependencies.
- A dependency is an object that another object depends on.
- You can read more on how to configure DI in .NET by following the link: https://docs.microsoft.com/en-us/dotnet/core/extensions/dependency-injection

https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-6.0

Exercise 3: Setup DI

1. We will define a ServiceCollection
2. From it we will create a ServiceProvider
3. We will use the provider to get an instance of type IUserService
4. We will call the methods that we built