Chapter 10
Dependency Injection MVC and Web API

Service
It is a class which contains business logic such as business calculations and business validations that are specific to the domain of the client's business.

- Service is abstraction layer between presentation layer and data layer
- It makes the business logic separated from presentation layer and data layer
- It makes the business logic to be unit testable easily

Inversion of Control(IOC)
Its design pattern provides reusable solutions for common problems which suggests IOC containers for implementation of Dependency Inversion Principle. All dependencies should be added into IServiceCollections which act as IOC containers.

Dependency Injection
It is a technique for achieving inversion of control between clients and their dependencies. It allows you to inject concrete implementation objects of low level components into high level component.The client class receives the dependent object as a parameter either in constructor or in method.

Constructor Injection
This is process of injecting dependent object into constructor
Ex

```
public class HomeController
{
  private readonly ICitiesService _citiesService;

    //constructor
    public HomeController(ICitiesService citiesService)
    {
            _citiesService = citiesService;
    }

}
```

Method Injection
You can also inject required object through required method only

```
namespace DIExample.Controllers
{
  public class HomeController : Controller
  {
    [Route("/")]
    public IActionResult Index([FromServices]ICitiesService
_citiesService)
    {
      List<string> cities = _citiesService.GetCities();
      return View(cities);
    }
  }
}
```

Service Lifetime
Service lifetime indicates when a new object of service has to be created and destroyed by IOC container.

Transient - Every time a new object is created for each request. Service objects are destroyed at the end of scope, usually browser requests. When you want to create a short lived object means one time for one controller at that time you can use transient.

Scoped - For each scope each request is created, scope means browser request. Service objects are destroyed at the end of scope, usually browser requests. Database services such dbcontext should be registered using scoped service.

Singleton - Only one object created through the entire application lifetime. Service instances are disposed of at application shutdown. When you want to store data temporarily at that time you can use singleton.Like you want to create some in memory collection that scenario you can use it.

ViewInject
You can also inject services in views directly without injecting it into the controller method. For that you first need to import required services in import views and use inject directive

```
@inject ICitiesService citiesServiceInView
<h1>Cities</h1>
@{
    var citiesFromServiceInView = citiesServiceInView.GetCities();
}

<ul class="list">
    @foreach (string city in citiesFromServiceInView)
    {
        <li>@city</li>
    }
</ul>
```

Best Practice in DI

- **Global state in Services -** Avoid using static classes to store some data globally for all users. You may use singleton service for simple scenarios. Alternatively prefer to use Distributed Cache/Redis for significant amount of data or complex scenarios.
- **Request state in services -** Don't use scooped services to share data with the same request because that is not thread safe. Use HttpContext.Item instead.
- **Dispose Method -** Do not call dispose method manually for service injected via DI. IOC container automatically invokes it at end of its scope.
- **Avoid Using Captive Dependencies -** Do not call transient service inside singleton service because at the end transient service act as singleton in this case.

Autofac
It is another IOC container library for .Net Core it is alternative to built in IOC container of .Net

Microsoft DI vs Autofac

Microsoft DI	**Autofac**
Three lifetimes Singleton, Scoped, Transient	5 lifetimes InstancePerDependency equivalent to transient, InstanceperLifeitmeScope equivalent to Scoped, SigleIntance equivalent to Singleton, InstancePerOwned create instance based on relation, InstancePermatchingLifetimeScope instance based on matching name.
Metadata for services not supported	Metadata for services supported
Decorator not supported	Decorator supported

Questions for this assignment

1. Explain how dependency injection works in ASP.NET Core?
2. “ASP.NET Core has dependency injection to manage services; are you aware of the different lifetimes? What are they, and what does each mean?”
3. What are the benefits of Dependency Injection?
4. What is IoC (DI) Container?
5. What is Inversion of Control?
6. How do you create your own scopes in asp.net core?
7. How do you inject a service in view?
8. Why you prefer Autofac over built-in Microsoft DI?
9. What exception do you get when a specific service that you injected, can’t be found in the IoC container?