Working with Services and Dependency Injection

Kevin Dockx
Architect

@KevinDockx | www.kevindockx.com

Coming Up

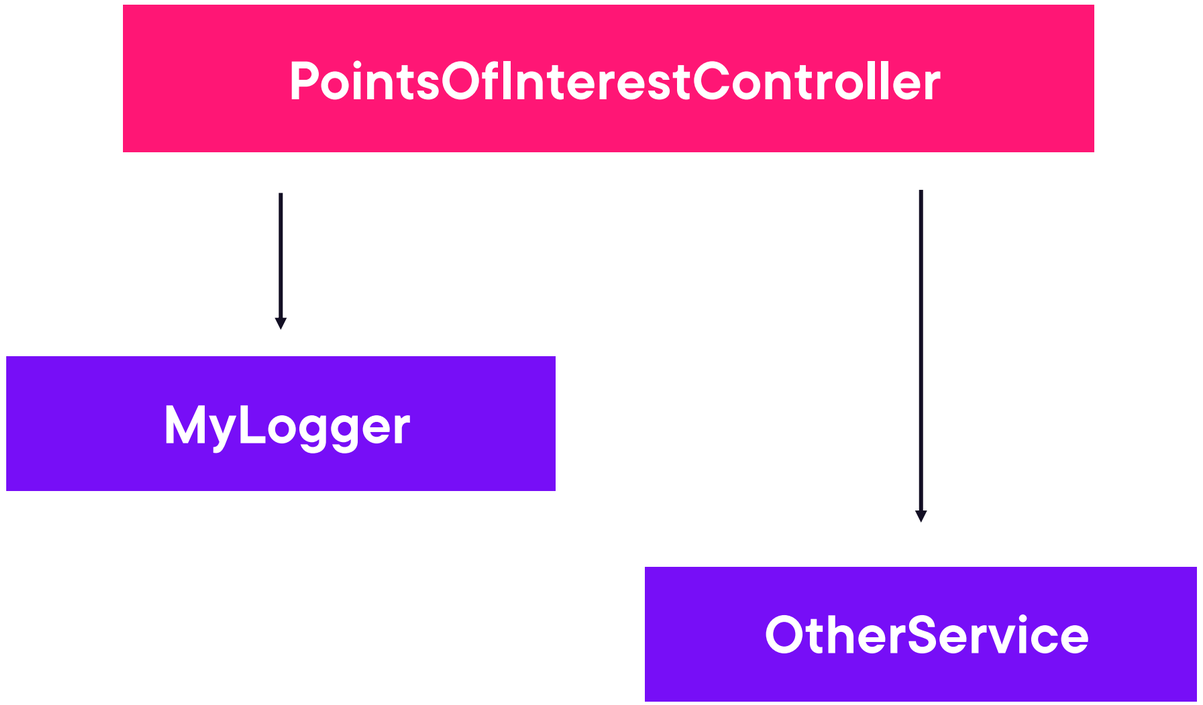
Inversion of control and dependency injection

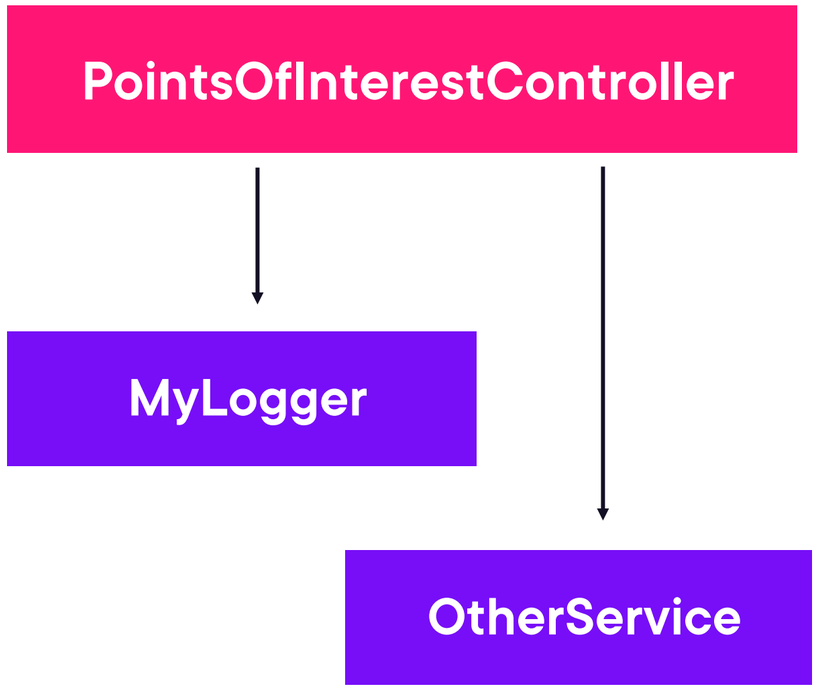
Logging in ASP.NET Core

Creating and using custom services

Working with configuration files and scoping them to environments

Inversion of Control and Dependency Injection





Issues arise...

- Class implementation has to change when a dependency changes
- Difficult to test
- Class manages the lifetime of the dependency

This is tight coupling

Inversion of Control

IoC delegates the function of selecting a concrete implementation type for a class's dependencies to an external component

Dependency Injection

A specialization of the Inversion of Control pattern which uses an object - the container - to initialize objects and provide the required dependencies to the object

Inversion of Control and Dependency Injection

Services are registered on the container

- The container becomes responsible for providing instances when needed: it manages the service lifetime

```
public class PointsOfInterestController :
Controller
{

    private
    ILogger<PointsOfInterestController>
    _logger;

    public PointsOfInterestController(
      ILogger<PointsOfInterestController>
      logger)
    {
        _logger = logger;
    }

...

}
```

- **Interface, not concrete implementation**
- **Constructor injection**

Inversion of Control and Dependency Injection

Class is decoupled from the concrete type

- Dependencies can easily be replaced
- Class becomes easier to test

Inversion of Control and Dependency Injection

Dependency injection is built into ASP.NET Core

Register services on the built-in container in your `Program` class

Demo

Injecting and using a logger

Demo

Handling and logging exceptions

Demo

Globally handling and logging exceptions

Demo

Replacing the default logger and logging to a file

Demo

Implementing and using a custom service

Demo

Registering a service by interface

Demo

Working with configuration files

Demo

Scoping configuration to environments

Summary

Dependency injection

- Specialization of inversion of control
  - Loose coupling
  - Less code changes
  - Better testability

Summary

Custom services are registered on the built-in container

- Transient
- Scoped
- Singleton

Use configuration files for configuration data, scoped to a specific environment

Up Next:

Getting Acquainted with Entity Framework Core