Introduction to .NET

Florin Olariu
“Alexandru Ioan Cuza”, University of Iași
Department of Computer Science

Agenda

- Building products - hints
- KATA MVC Core Scaffolding using EF Core(Tips&Tricks)
- Using Dependency Injection with ASP.NET Core 2.1
- ASP.NET MVC Core 2.1
  - Models, views and controllers – an MVC refresher
  - Understanding MVC

Building products - hints

Building products - hints

- “Build more. Code less.”
- “Software is eating the world. – We are just little vegetables floating in software soup.”

KATA MVC Core Scaffolding using EF Core(Tips&Tricks)

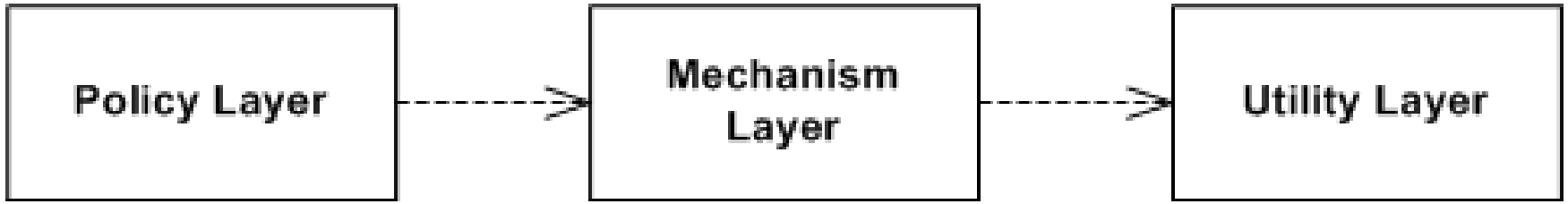
- Kata - demo

Using Dependency Injection with ASP.NET

- IoC/DI mechanisms

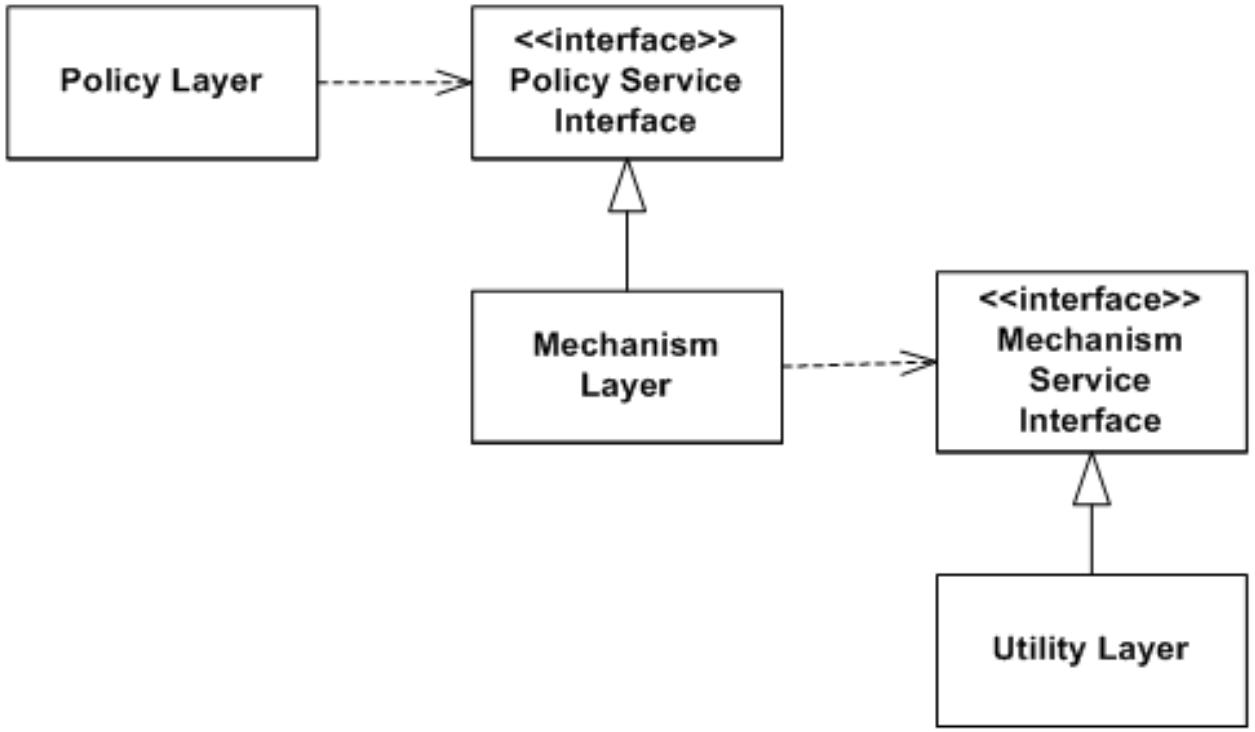
Using Dependency Injection with ASP.NET

- IoC/DI mechanisms



Using Dependency Injection with ASP.NET

- IoC/DI mechanisms



Using Dependency Injection with ASP.NET

- IoC/DI mechanisms

PROS	CONS
Helps with adhering to the Dependency Inversion Principle (DIP)	DI introduces a learning curve for some developers
Allows objects to be easily swapped with replacements	DI may require a significant overhaul of existing projects
Facilitates the use of the Strategy Design Pattern (SDP)	
Improves the testability of applications	
Enables loose coupling of software components	

Using Dependency Injection with ASP.NET

- IoC/DI mechanisms

Using Dependency Injection with ASP.NET

- IoC/DI mechanisms
- Transient

Using Dependency Injection with ASP.NET

- IoC/DI mechanisms

- Transient
- Scoped

Using Dependency Injection with ASP.NET

- IoC/DI mechanisms

- Transient
- Scoped
- Singleton

Using Dependency Injection with ASP.NET

- IoC/DI mechanisms

- Transient
- Scoped
- Singleton
- Instance (special case of Singleton) – AddSingleton()

Using Dependency Injection with ASP.NET

- IoC/DI mechanisms

- Transient
- Scoped
- Singleton
- Instance (special case of Singleton)

Using Dependency Injection with ASP.NET

- IoC/DI mechanisms

- Transient
- Scoped
- Singleton
- Instance (special case of Singleton)

- Constructor injection

Using Dependency Injection with ASP.NET

- IoC/DI mechanisms

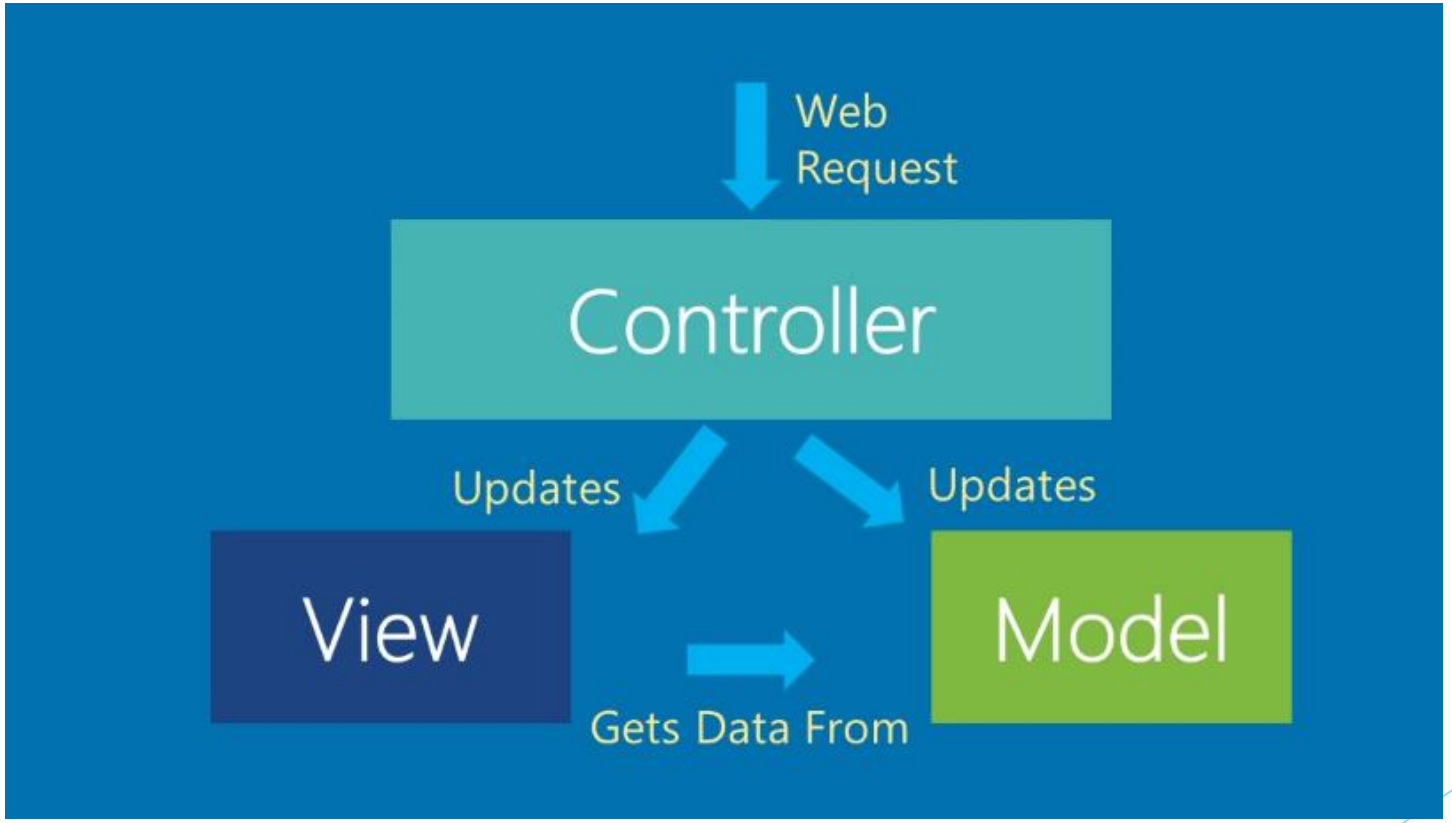
- Transient
- Scoped
- Singleton
- Instance (special case of Singleton)

- Constructor injection
- Action injection

ASP.NET MVC Core 2.1

Models, views and controllers – an MVC refresher

Models, views and controllers – an MVC refresher



Models, views and controllers – an MVC refresher

- Controllers

Models, views and controllers – an MVC refresher

- Controllers
  - Is a subclass of the base class **Controller**

Models, views and controllers – an MVC refresher

- Controllers
  - Is a subclass of the base class Controller
  - Lives into the following namespace: **Microsoft.AspNetCore.Mvc**

Models, views and controllers – an MVC refresher

- Controllers
  - Is a subclass of the base class Controller
  - Lives into the following namespace: **Microsoft.AspNetCore.Mvc**
  - Typically a controller returns an **IActionResult** from its action methods

Models, views and controllers – an MVC refresher

- **Models**

Models, views and controllers – an MVC refresher

- **Models**
  - **Typically a model is DTO class (properties)**

Models, views and controllers – an MVC refresher

- **Models**
  - **Typically a model is DTO class (properties)**
  - **For a cleaner architecture, you can use a view-specific model (or ViewModel) to bind to a view.**

Models, views and controllers – an MVC refresher

- **Views**

Models, views and controllers – an MVC refresher

- **Views**
  - **Views are stored in *.cshtml* files**

Models, views and controllers – an MVC refresher

- **Views**
  - **Views are stored in *.cshtml* files**
  - **ViewBag** allows you to store your own properties and display them in the view.

Models, views and controllers – an MVC refresher

- **Views**
  - **Views are stored in *.cshtml* files**
  - **ViewBag** allows you to store your own properties and display them in the view.
  - We can use tag helpers in your views for smoother syntax.

Understanding MVC

Understanding MVC

- Implementing controllers
  - **HttpGet**: Uses the HTTP GET method with optional querystring parameters
  - **HttpPost**: Uses the HTTP POST method for form submissions to create an entity

Understanding MVC

- Implementing controllers
  - **HttpGet**: Uses the HTTP GET method with optional querystring parameters
  - **HttpPost**: Uses the HTTP POST method for form submissions to create an entity
  - **HttpPut**: Uses the HTTP PUT method to edit an existing entity
  - **HttpDelete**: Uses the HTTP DELETE method to delete an existing entity
  - **HttpPatch**: Allows partial model updates instead of a full PUT request
  - **AcceptVerbs**: Allows multiple action verbs to be specified

Understanding MVC

- Implementing views

Understanding MVC

- Implementing views
  - ViewData, ViewBag, and TempData

Understanding MVC

- Implementing views
  - ViewData, ViewBag, and TempData

```
ViewData[" PatientId"] = id;
ViewBag.PatientData = "someData";
TempData[" UserToken"] = userTokenData;
```

```
@{
        ViewData["Title"] = "Patient Details";
}
        <h2> Patient Details </h2>
        <ul>
                <li> ID: @ViewData[" PatientId"] </li>
                <li> Name: @ViewData[" PatientName"] </li>
        </ul>
```

Understanding MVC

- Implementing views
  - ViewData, ViewBag, and TempData

```
@{
        ViewData[" Title"] = "Patient Index";
}
<h2> Patient Index, with Tag Helpers </h2>
 <ul> @for (int i=0; i<10; i++) {
        <li>
        <a asp-controller =" Patient"
        asp-action ="Details" asp-route-id ="@i"
        asp-route-name ="Patient @i" >
        Patient # @i </a></li>
        }
</ul>
```

Understanding MVC

- Implementing models and ViewModels

Understanding MVC

- Implementing models and ViewModels
  - A model is just a class file with a .cs file extension.

Understanding MVC

- Implementing models and ViewModels
  - A model is just a class file with a .cs file extension.

```
@using (Html.BeginForm("action", "controller", FormMethod.Post, new {}))
{
    @Html.LabelFor(m => m.Field1)
    @Html.TextBoxFor(m => m.Field1)

    @Html.LabelFor(m => m.Field2)
    @Html.TextBoxFor(m => m.Field2)

    <input type='Submit' value='Submit' />
}
```

Understanding MVC

- Implementing models and ViewModels
  - A model is just a class file with a .cs file extension.

```
<form asp-controller="controller" asp-action="action" method="post">
    <label asp-for="Field1"></label>
    <input asp-for="Field1" />
    <label asp-for="Field2"></label>
    <input asp-for="Field2" />
    <input type="submit" value="Submit" />
</form>
```

Understanding MVC

- Implementing models and ViewModels
  - A model is just a class file with a .cs file extension.

```
2 references
public class VerifyCodeViewModel
{
    [Required]
    2 references
    public string Provider { get; set; }

    [Required]
    1 reference
    public string Code { get; set; }

    2 references
    public string ReturnUrl { get; set; }

    [Display(Name = "Remember this browser?")]
    1 reference
    public bool RememberBrowser { get; set; }

    [Display(Name = "Remember me?")]
    2 references
    public bool RememberMe { get; set; }
}
```

One more thing…(1/2)

One more thing…(2/2)

“Truth can only be found in one place: the code.”

by Robert C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship

Bibliography

- https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api- By Mike Wasson and Rick Anderson
- Chowdhuri, Shahed. ASP.NET Core Essentials
- Price, Mark J.. C# 7 and .NET Core: Modern Cross-Platform Development

Questions

- Do you have any other questions?

Thanks!

See you next time! ☺