DAY 3

Welcome back!

26 April 2017

Emil Craciun, Visma

Developing ASP.NET MVC Core 1.1 WEB Applications

Agenda for this day

- Introduction
- Git
- OWIN
- MVC

Show off time....

PYRAESSENCE

Microsoft
Specialist

Programming in HTML5 with JavaScript & CSS3

Microsoft
Specialist

Programming in C#

Microsoft
CERTIFIED
Solutions Developer

Web Applications

Microsoft
CERTIFIED
Solutions Associate

Web Applications

Microsoft
CERTIFIED
Professional

INTRODUCTION

This training gives you a **quick introduction** to **ASP.NET MVC Core 1.1** and some **Azure** Cloud technologies and shows you how to setup your **development environment** to build awesome ASP.NET Web applications.

The MVC Pattern

The MVC Pattern

Originally named **Thing-Model-View-Editor** in 1979.

Is a powerful and elegant means of separating concerns within an application.

- **The Model:** a set of classes that describes the data you're working with as well as the business rules for how the data can be changed and manipulated.
- **The View:** defines how the application's UI will be displayed.
- **The Controller:** a set of classes that handles communication from the user, overall application flow, and application-specific logic.

The MVC Pattern

With ASP.NET MVC, it's translated roughly as:

- **Models**: classes/domain objects that often encapsulate data stored in a database (Data Access Layer of some kind, using a tool like **Entity Framework or NHibernate** combined with custom code containing domain-specific logic).

- **View**: this is a template to dynamically generate **HTML**.

- **Controller**: a class that manages the relationship between the View and the Model. It responds to user input, talks to the Model, and decides which view to render (if any). In ASP.NET MVC, this class is conventionally denoted by the suffix ***Controller***.

Introduction to ASP.NET 5

The road to ASP.NET 5

History of ASP.NET MVC

ASP.NET MVC 1

Released on March 13, 2009

- HTML helpers
- Ajax helpers
- Routing
- Unit Testing

ASP.NET MVC 2

Released on March 10, 2010

- Templated helpers
- Client side Validation
- Asynchronous controllers

ASP.NET MVC 3

- Templates for HTML 5 and CSS 3
- Improved Model Validation
- Razor view Engine
- Support for Multiple View Engines
- Controller Improvements
- Unobtrusive JavaScript approach
- Improved Dependency Injection
- Partial page output caching

ASP.NET MVC 4

- ASP.NET Web API
- Adaptive rendering
- Look-n-feel improvements
- Empty Project Template
- Mobile Project Template
- Support for adding controller
- Task support for Async Controllers
- Bundling and Minification
- Support for Oauth and OpenID
- Support for Windows Azure SDK 1.6

ASP.NET MVC 5

- ASP.NET identity
- Authentication Filters
- Filter overrides
- Bootstrap
- Attribute Routing

History of ASP.NET MVC

- **ASP.NET MVC 1.0** was released on March 13, 2009
  Creator: Scott Guthrie, in 2007

- **ASP.NET MVC 2** – March, 2010
  - UI helpers with automatic scaffolding with customizable templates
  - Attribute-based model validation on both the client and server
  - Strongly typed HTML helpers
  - Improved Visual Studio tooling
  - A lot of API enhancements and “pro” features: areas, asynchronous controllers, Html.RenderAction, etc.

History of ASP.NET MVC

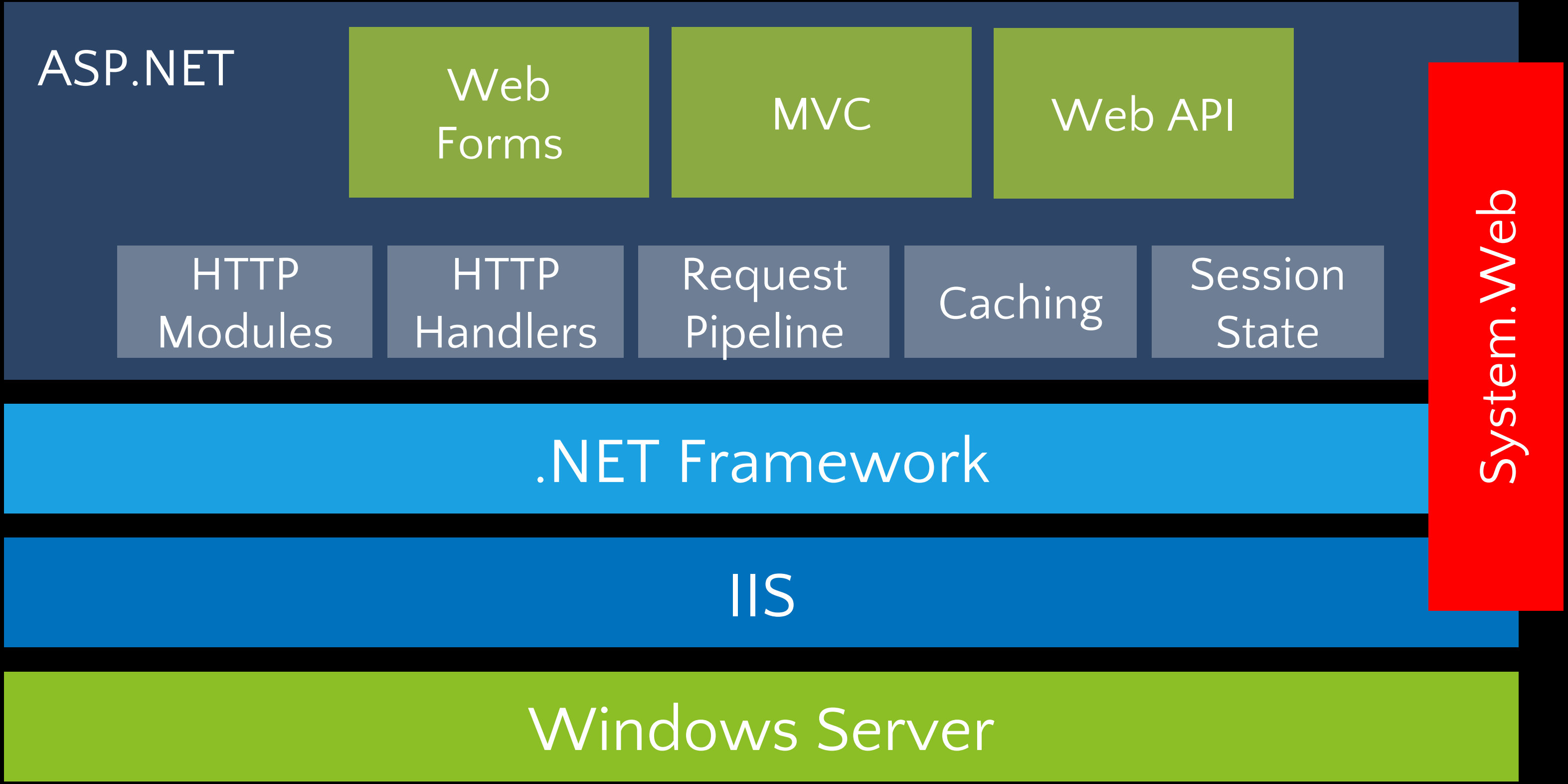
- **ASP.NET MVC 3** – January, 2011
  - The Razor view engine – previous MVC used WebForms
  - Support for .NET 4 Data Annotations, improved model validation
  - Support for dependency resolution and global action filters
  - Better JavaScript support with unobtrusive JavaScript, jQuery Validation, and JSON binding
  - Use of NuGet to deliver software and manage dependencies throughout the platform

- **ASP.NET MVC 4** – August, 2012
  - ASP.NET Web API (HTTP services)
  - Enhancements to default project templates
  - Mobile project template using jQuery Mobile
  - Display modes (support for .mobile.cshtml)
  - Task support for asynchronous controllers
  - Bundling and Minification

History of ASP.NET MVC

- **ASP.NET MVC 5** – October, 2013
  - One ASP.NET Experience
  - New Web Project Experience
  - ASP.NET Identity
  - Bootstrap templates
  - Attribute Routing
  - ASP.NET scaffolding
  - Authentication filters
  - Filter overrides

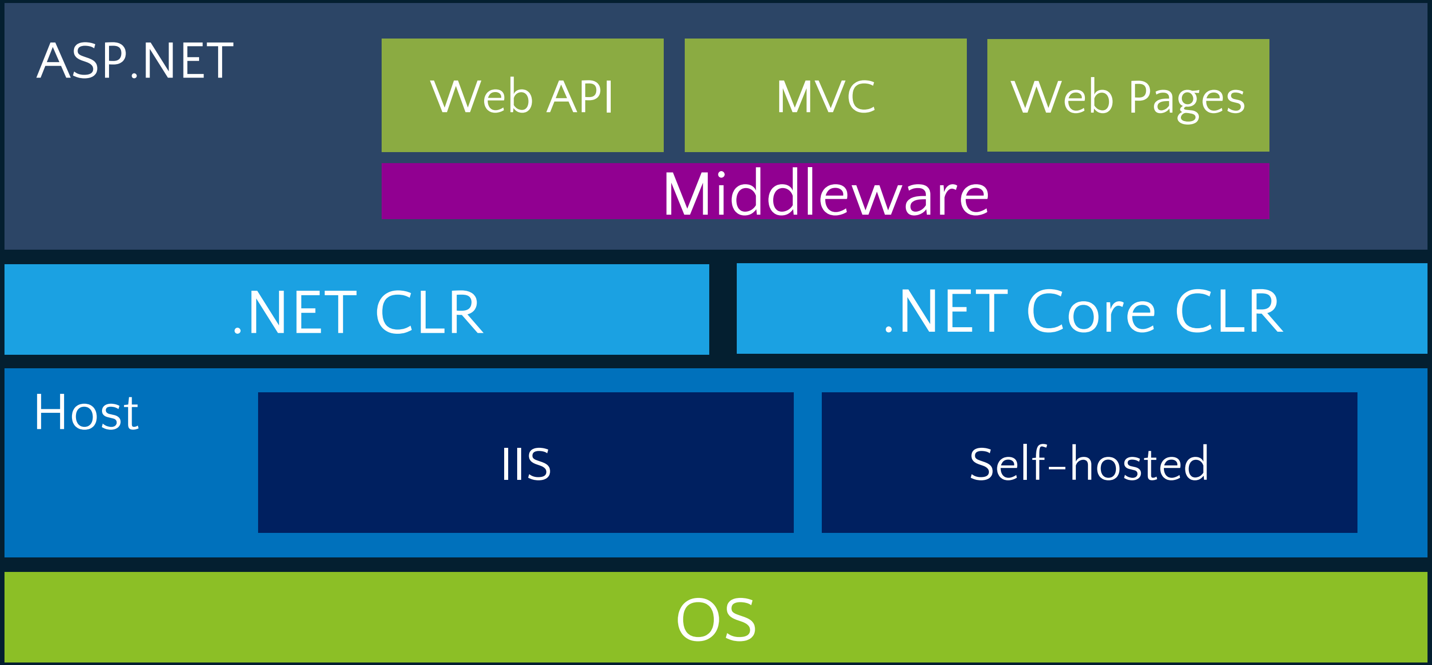
Current ASP.NET Stack



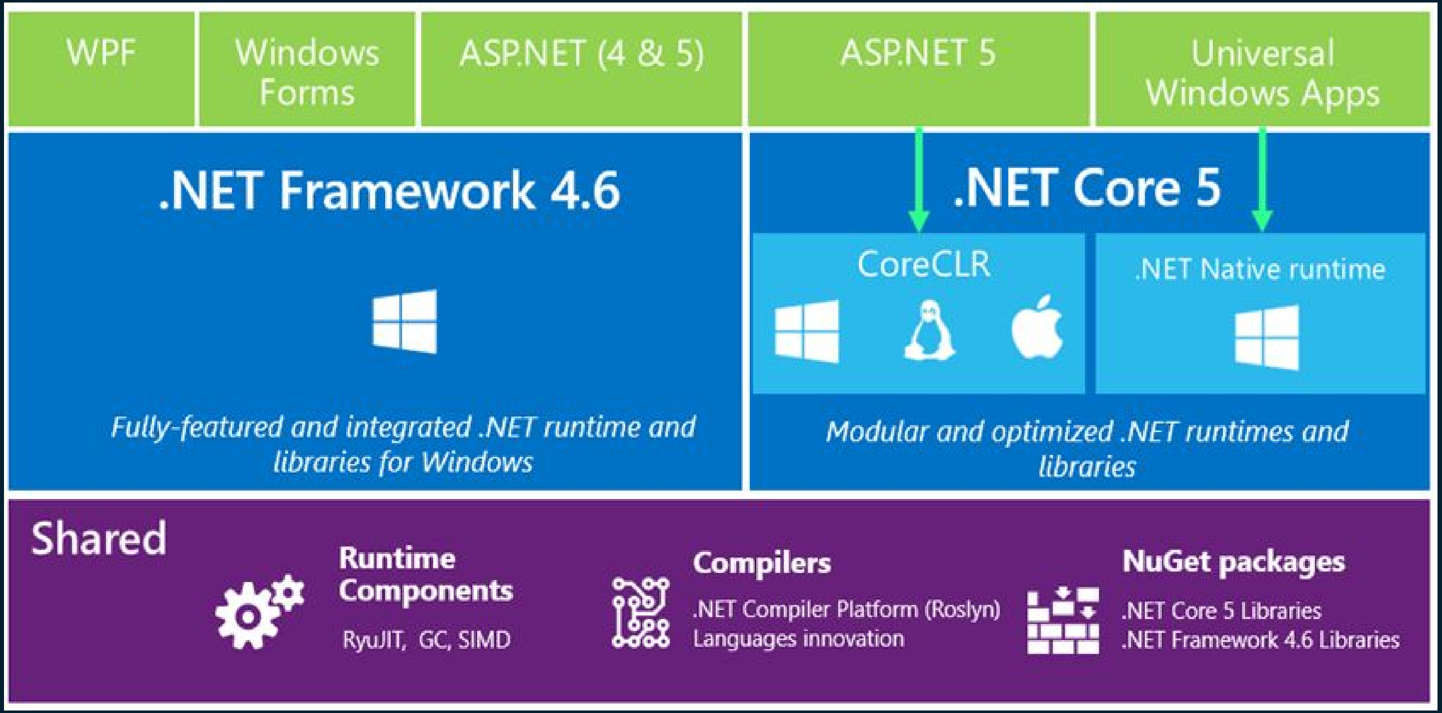
Problems with ASP.NET architecture

- Limited hosting possibilities (IIS only)
- Dependency on IIS environment (System.Web)
- Web evolves faster than .NET framework
- Requires full-blown .NET framework - resource intensive and not web-friendly
- Hard to optimize for lightweight high-performance apps

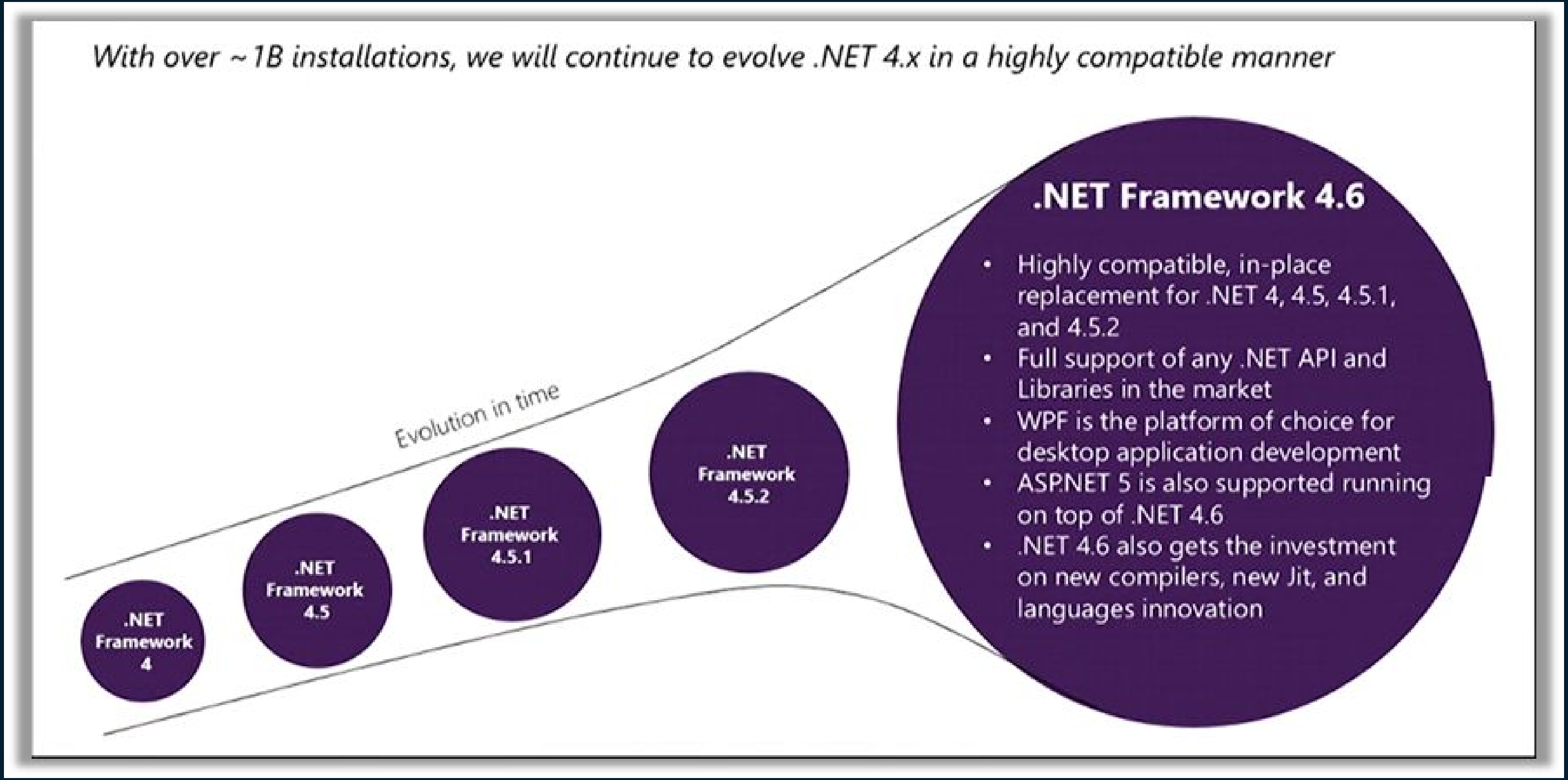
Introducing ASP.NET 5 stack



.NET 2015/6: High-Level Overview



.NET Framework 4.6



With over ~1B installations, we will continue to evolve .NET 4.x in a highly compatible manner

.NET Framework 4.6

- Highly compatible, in-place replacement for .NET 4, 4.5, 4.5.1, and 4.5.2
- Full support of any .NET API and Libraries in the market
- WPF is the platform of choice for desktop application development
- ASP.NET 5 is also supported running on top of .NET 4.6
- .NET 4.6 also gets the investment on new compilers, new Jit, and languages innovation

ASP.NET 5 – Agility

- **Faster Development Cycle**
  - Features are shipped as packages
  - Framework ships as part of the application
- **More Control**
  - Zero day security bugs patched by Microsoft
  - Same code runs in development and production
  - Developer opts into new versions, allowing breaking changes

ASP.NET 5 – Fast

- **Runtime Performance**
  - Faster startup times
  - Lower memory / higher density (> 90% reduction)
- **Modular, opt into just features needed**
  - Use a raw socket, framework or both
  - Development productivity and low friction
- **Edit code and refresh browser**
  - Flexibility of dynamic environment with the power of .NET
  - Develop with Visual Studio, third party and cloud editors

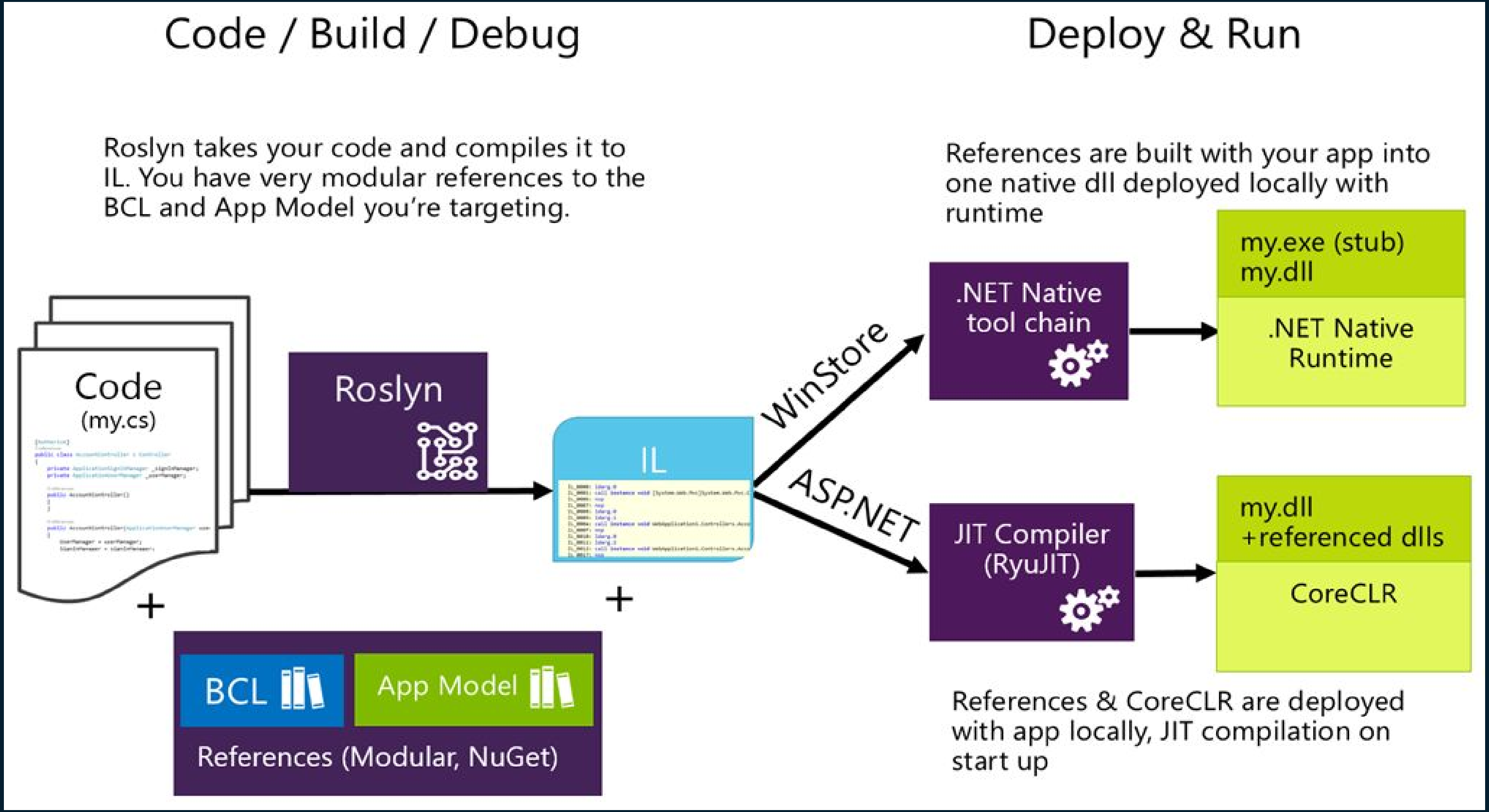
ASP.NET 5 – Cross platform

- **Runtime**
  - Windows, Mac, Linux
- **Editors**
  - Visual Studio + Code, Text, Cloud editors
  - OmniSharp – Sublime, Emacs, Vi, etc.
  - No editors (command line)
- **All Open Source with Contributions**

ASP.NET 5 – Features

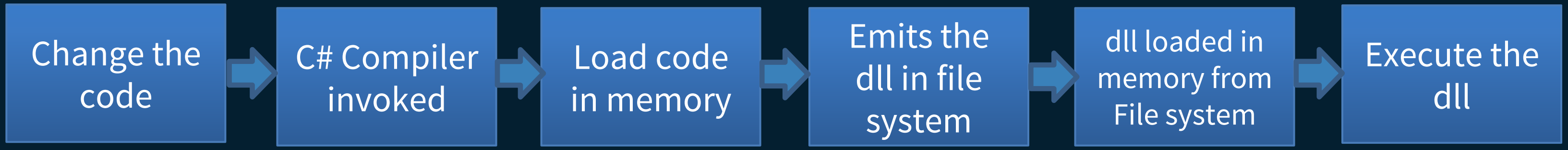
- New flexible and **cross-platform** runtime
- New **modular HTTP request** pipeline
- **Robust** environment configuration
- **Unified** programming model for MVC and API
- See changes **without re-building** the project
- **Side-by-side versioning** of the .NET Framework
- Built in **Dependency Injection**
- Ability to **self-host** or host on IIS
- **Open source** in GitHub

Compilation Process

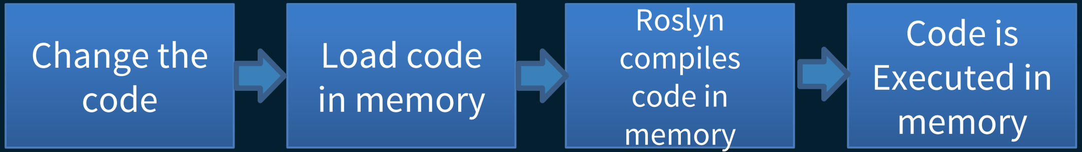


Code / Build / Debug
Deploy & Run
Roslyn takes your code and compiles it to IL. You have very modular references to the BCL and App Model you're targeting.
References are built with your app into one native dll deployed locally with runtime
Code
(my.cs)
Roslyn
IL
+
+
BCL
App Model
References (Modular, NuGet)
WinStore
ASP.NET
.NET Native tool chain
JIT Compiler (RyuJIT)
my.exe (stub)
my.dll
.NET Native Runtime
my.dll
+referenced dlls
CoreCLR
References & CoreCLR are deployed with app locally, JIT compilation on start up

Debugging without Roslyn



Debugging with Roslyn

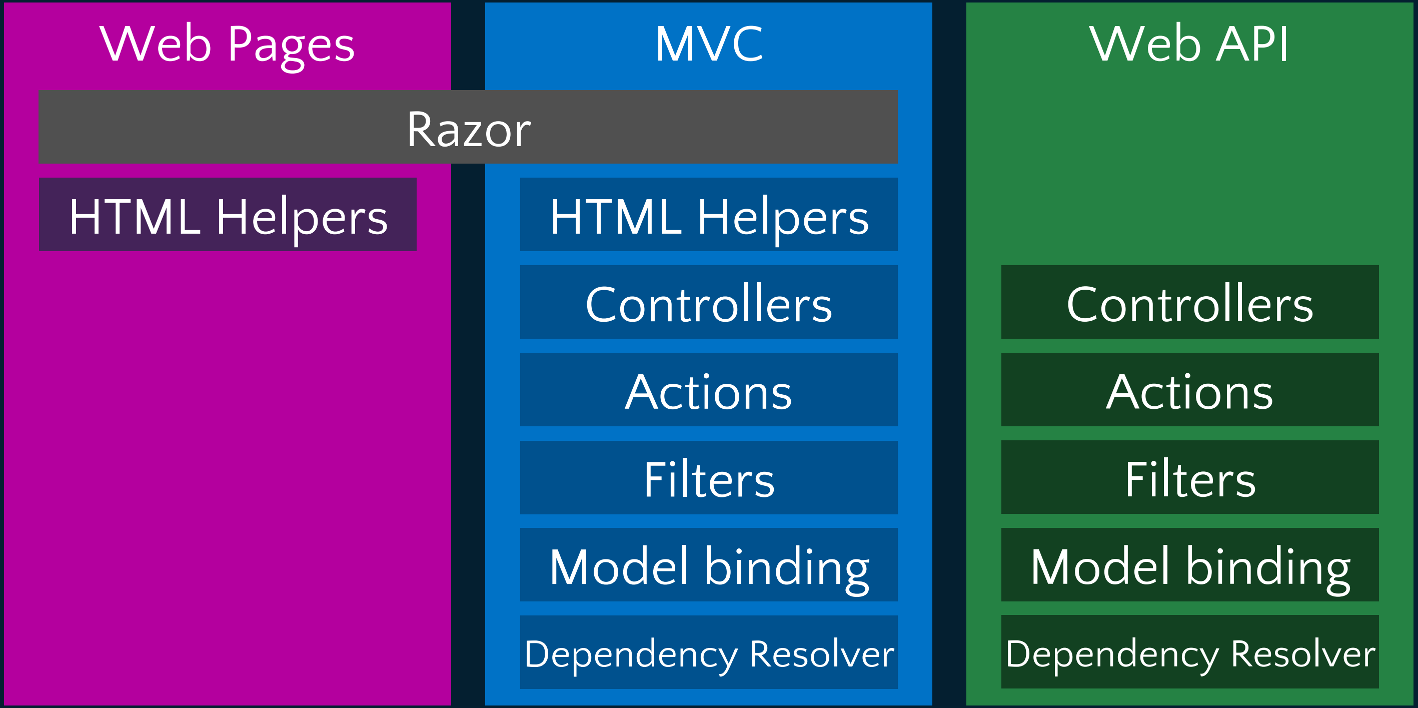


Time reduced from 7-8 second to 1-2 second

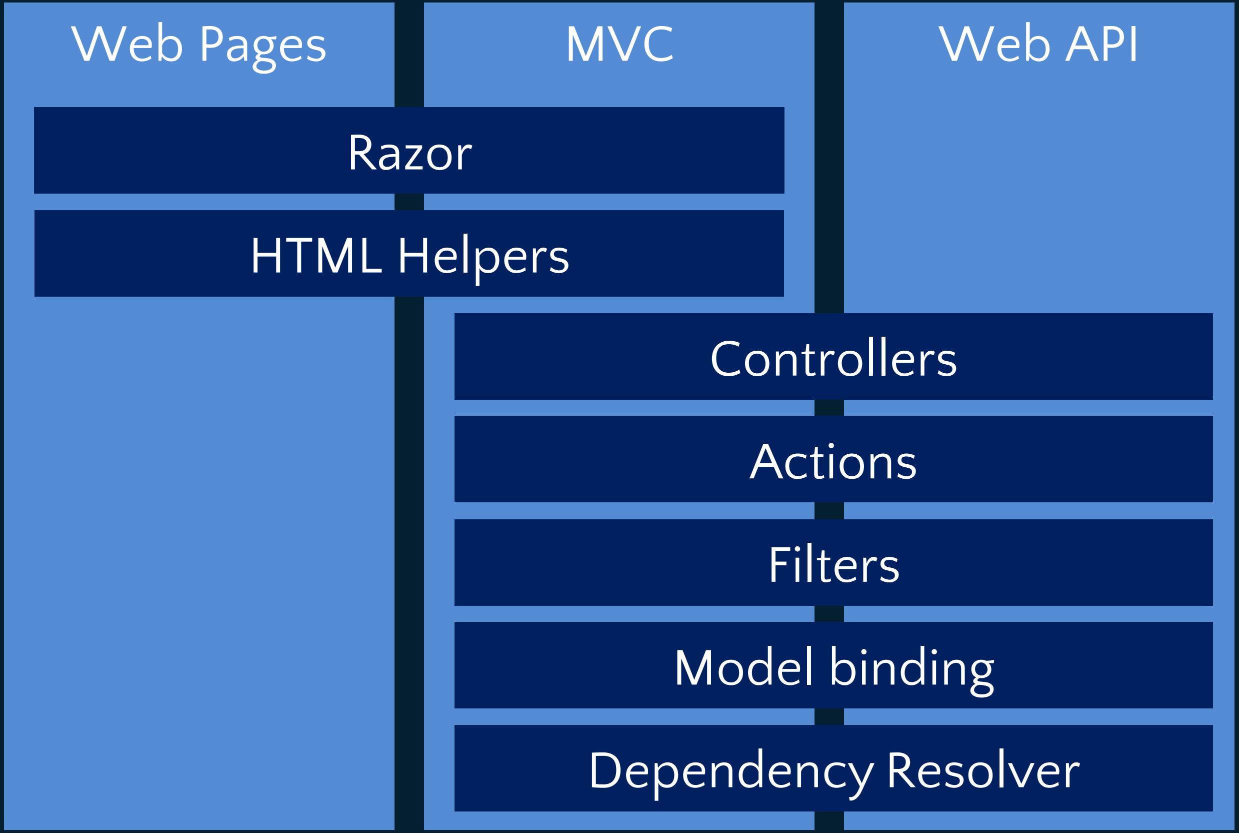
Design of ASP.NET MVC Core 1.1 Web Apps

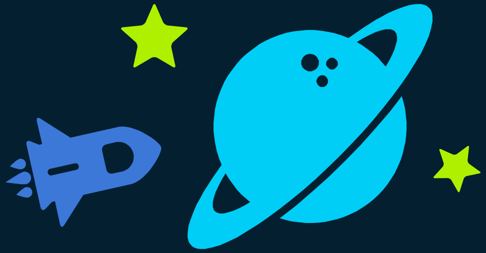
Reimagining Web Application Development in **.NET** …

ASP.NET < 5 Frameworks



ASP.NET 5 with MVC 6 (MVC Core 1.1)





MVC + Web API + Web Pages = ASP.NET MVC 6!

aka MVC Core 1.1

ASP.NET MVC Core 1.1

- No more duplication - one set of concepts
- Used for creating both UI and API
- Smooth transition from Web Pages to MVC
- Based on the new ASP.NET 5 pipeline
- Built DI first
- Runs on IIS or self-host

Ok enough talking…

(or not) Short detour -> GIT

What?

- most widely used modern version control system in the world today
- open source project originally developed in 2005 by Linus Torvalds
- distributed architecture

GIT

Where?

- https://git-scm.com/download/win

GIT

Why?

- Wouldn't you want to have a historical overview of your project? (and maybe revert a mistake)
- You'll be probably using it anyway pretty soon if not already
- To get used to a VCS

Getting started with MVC Core 1.1

The very first steps...

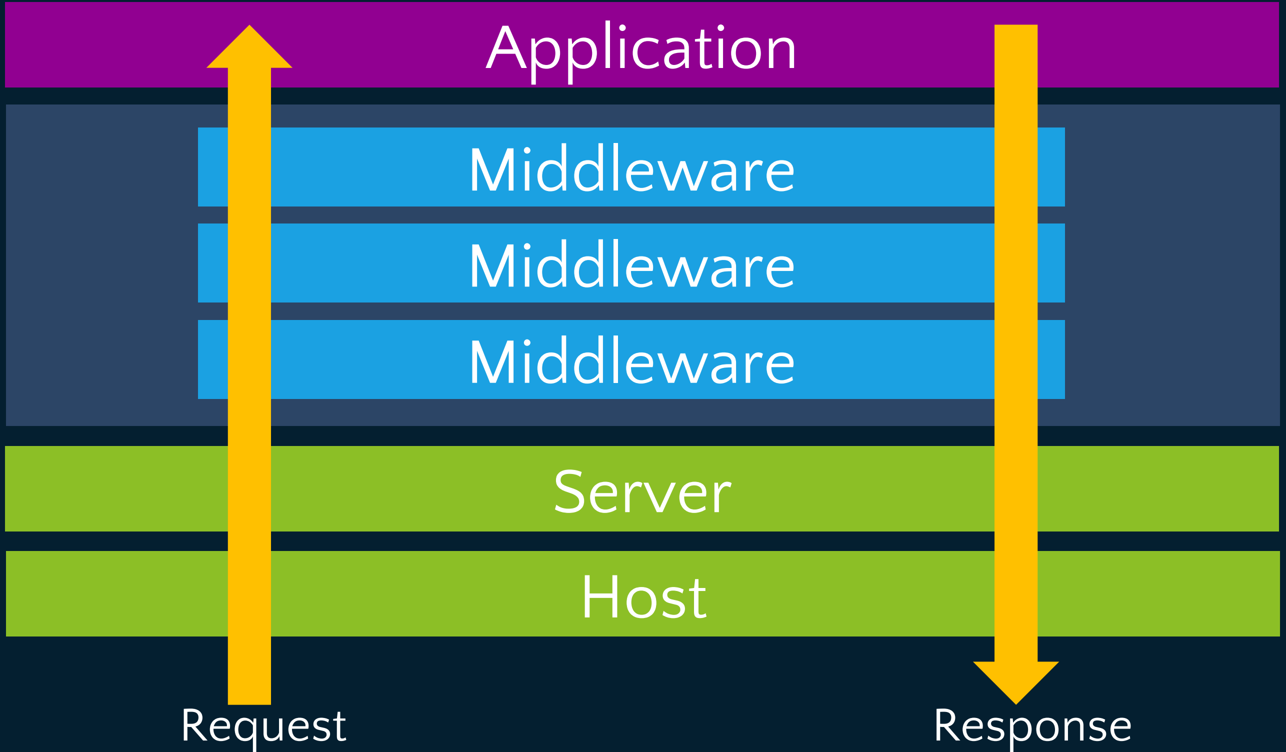
- Project structure
- Startup.cs
  https://docs.microsoft.com/en-us/aspnet/core/fundamentals/startup
- Configuration files
  https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration
- Project properties

! Don’t forget to set up git first :D

Let’s talk about OWIN

- **O**pen **W**eb **I**nterface for .**N**ET
- Community project (http://owin.org)
- Decouples application from server
- Enforces modularity of the server
- Stack of modules (**middlewares**) is processing the request from application to server
- Microsoft implementation of OWIN is "**Katana**"
- https://docs.microsoft.com/en-us/aspnet/core/fundamentals/owin

OWIN Implementation



Your code

Pass-through components stack

Manages sockets, delegates to middlewares

Startup, bootstrapping, process management

Middleware

Middleware is software that is assembled into an application pipeline to handle requests and responses.

https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware

ASP.NET 5 Middlewares

- **Improved HTTP performance**
  - New HTTP request pipeline that is **lean** and **fast**
    - The new pipeline also supports OWIN
- **You choose what to use in your application**
  - By registering middlewares

Dependency injection

A technique for achieving loose coupling between objects and their collaborators, or dependencies.

https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection

Environments

- improved support for controlling application behavior across multiple environments

https://docs.microsoft.com/en-us/aspnet/core/fundamentals/environments

Package Management

- ASP.NET 5 introduces a new, lightweight way to manage dependencies in your projects
  - *No more assembly references*
  - *Instead referencing NuGet packages*

Serving static files

Files served directly by ASP.NET Core apps, without any processing.

https://docs.microsoft.com/en-us/aspnet/core/fundamentals/static-files

Adding MVC

And so we begin :D

Routing - conventional & attribute

Map incoming requests to a request handler.

- Inline constraints
  - **Product/{ProductId:long}**
  - **Product/{ProductName:alpha}**
  - **Product/{ProductName:minlength(10)}**
- Optional parameters
  - **Product/{productId:long?}**
- Default values
  - **Product/{productId:long=1}**

https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/routing

https://docs.microsoft.com/en-us/aspnet/core/fundamentals/routing

Controllers

- They handle a request
- Get the model data and call view templates that return a response

https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/adding-controller

https://msdn.microsoft.com/en-us/library/dd410269(v=vs.98).aspx

https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/controller-methods-views

Understanding Controllers

- Every user input to a web application simply takes the form of a **request**; each request is routed (using **routing**) to a method on a controller (called an **action**).

- Controllers are responsible for responding to user input, often making changes to the model in response to user input. In this way, controllers in the MVC pattern <u>are concerned with the flow of the application</u>, working with data coming in, and providing data going out to the relevant view.

Actions

- The actual code that runs when a request is successfully mapped

https://msdn.microsoft.com/en-us/library/dd410269(v=vs.98).aspx#Action Methods

Controller Goodie - DI

- Dependency Injection and MVC
- ASP.NET 5 is **DI-friendly**
- Basic DI container available throughout the stack
- Out-of-the-box container supports
  - **Singleton** / Instance – single instance
  - **Transient** – new instance each time
  - **Scoped** – new instance per request

https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/dependency-injection

Action Results

- Basically returns the result of an action method (usually a view, but not always).

https://msdn.microsoft.com/en-us/library/dd410269(v=vs.98).aspx#ActionResult Return Type (actually go to link to show the types)

Views

- Encapsulates presentation logic and should not contain any application logic

https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/adding-view

https://msdn.microsoft.com/en-us/library/dd410123(v=vs.98).aspx

Understanding Views

The **view** is responsible for providing the user interface (UI) to the user.

↓

A view is always rendered by a **controller**, which provides the data the view will

render.

↓

More often, the controller needs to provide some information to the view, so it passes a data transfer object called a **model**.

↓

The **view** transforms that model into a format ready to be presented to the user.

Views Conventions

- There are some implicit **conventions** in the ASP.NET MVC Framework, which define the view selection logic.
- Notice that the project contains a Views directory structured in a very specific manner **\Views\[ControllerName].**
- Each controller folder contains <u>a view file for each action method</u>, and the file is named the same as the action method.
- The view selection logic looks for <u>a view with the same name as the action</u> within the /Views/ControllerName directory.
- You can override this convention:
  - return View(“ViewName”);
  - return View("~/Views/Example/Index.cshtml");

Adding Views

- **Create as a partial view:** Selecting this option indicates that the view you will create is not a full view, thus the Layout option is disabled. The resulting partial view looks much like a regular view, except you'll have no <html> tag or <head> tag at the top of the view.

- **Use a layout page:** This option determines whether or not the view you are creating references a layout or is a fully self-contained view. Specifying a layout is not necessary if you choose to use the default layout because the layout is already specified in the _ViewStart.cshtml file. However, you can use this option to override the default Layout file.

Razor syntax

- Syntax for embedding server based code into web pages
- Razor + C# + HTML

https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor

Understanding Models

- We discuss about models as the objects you use to send information to the database, perform business calculations, and even render in a view. In other words, these objects represent the domain the application focuses on, and the models are the objects you want to display, save, create, update, and delete (**CRUD**)
- <u>You need to talk to customers and business owners, and do some initial prototyping or test-driven development to start fleshing out a design</u>
- In a new MVC application, the folder “Models” stores the models that the application uses.
- In a complex application, usually the “**Models**” come from a different library project and the logic in handling those objects is stored in a “**Business**” layer project.

Models

https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/adding-model

Wrap up!

- Very basic Git intro
- OWIN
- Config files
- Static files
- Environments
- ASP.NET MVC -> the C, V and M

Key improvements in ASP.NET 5

- Totally **modular**
  - NuGet is first-class citizen in ASP.NET 5
  - Everything is a package
- **Lightweight** - you use minimum set of modules
- Faster startup, lower memory (>90%)
- Does not require .NET Framework installation - runtime environment (CLR) can be deployed with your application

Key improvements in ASP.NET 5

- Cross platform - can be hosted anywhere:

  IIS, self-hosted, Linux, Mac...

- Web Forms are left aside for now
- Better developer experience

  No-compile debugging with **Roslyn**, MVC **unified** programming model, basic **DI** out-of-the-box...

- Everything is **open-source**
- Architecture is **OWIN** based

THANK YOU!

Any questions?

You can find me at

emil.craciun@visma.com

DAY 4

Welcome back!

27 April 2016

Emil Craciun, Visma

Remember from last time?

GO!

Agenda for this day

- *HTML Helpers
- View models
- Data validation
- More about the views
- Tag helpers
- Entity Framework Core

*HTTP Verbs

- Talk about how the web works

https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/controller-methods-views

*HTML Helpers

- Classes that help you generate HTML

http://www.tutorialsteacher.com/mvc/html-helpers

View Models

- Don’t let the internals leak out into the view
- Aggregate multiple data types
- Limit access to fields
- What is stored or used internally by the application does not have to be 1:1 with what is displayed!

Data annotations

- They are basically attributes (eg. for **validation**)
- Set restrictions on fields
- Set a display name for a field
- Set data type

https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation

https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions/mvc-music-store/mvc-music-store-part-6

https://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations(v=vs.110).aspx

Layouts

- Layouts serve the same purpose as master pages.
- Used to define a common template for your site.
- There's a call to **@RenderBody** in the view, a placeholder that marks the location where views using this layout will have their main content rendered.
- A layout may have multiple sections. The view that uses the layout must implement all sections defined in the layout if required = true.

https://docs.microsoft.com/en-us/aspnet/core/mvc/views/layout

View Start

- If a set of views shares common settings, the **_ViewStart.cshtml** file is a useful place to consolidate these common view settings.
- The statements listed in _ViewStart.cshtml are run before every full view (not layouts, and not partial views).

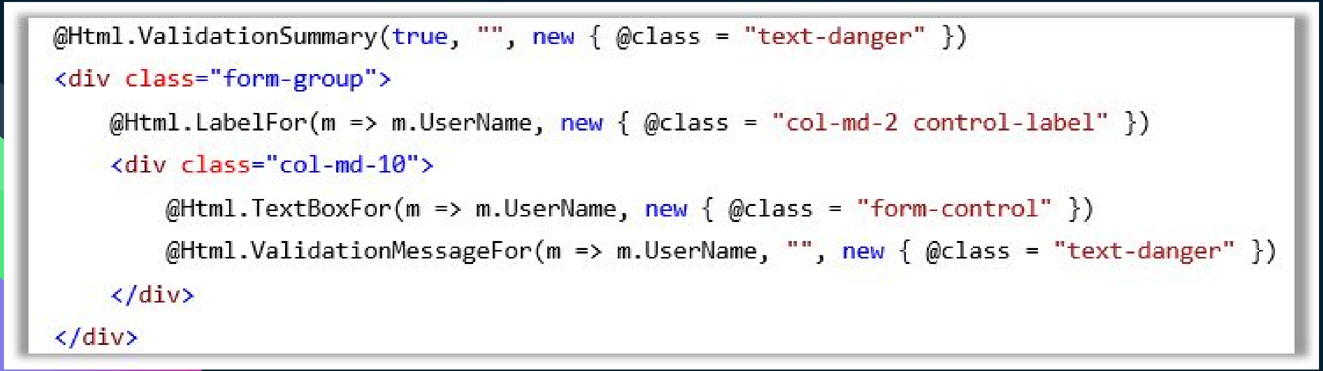
View Imports

- `@addTagHelper`
- `@removeTagHelper`
- `@tagHelperPrefix`
- `@using`
- `@model`
- `@inherits`
- `@inject`

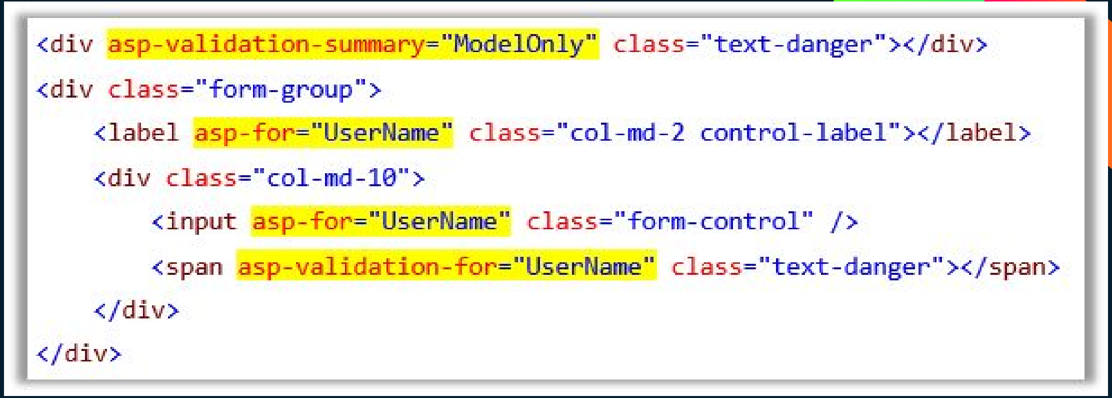
View Start & View Imports

- Both are hierarchical
- Typical location is in /Views folder, not /Views/Shared
- If there are some controller-associated view folder files, they will run **after** the default ones from /Views

Tag Helpers: Evolution of HTML Forms



```
@Html.ValidationSummary(true, "", new { @class = "text-danger" })
<div class="form-group">
    @Html.LabelFor(m => m.UserName, new { @class = "col-md-2 control-label" })
    <div class="col-md-10">
        @Html.TextBoxFor(m => m.UserName, new { @class = "form-control" })
        @Html.ValidationMessageFor(m => m.UserName, "", new { @class = "text-danger" })
    </div>
</div>
```



```
<div asp-validation-summary="ModelOnly" class="text-danger"></div>
<div class="form-group">
    <label asp-for="UserName" class="col-md-2 control-label"></label>
    <div class="col-md-10">
        <input asp-for="UserName" class="form-control" />
        <span asp-validation-for="UserName" class="text-danger"></span>
    </div>
</div>
```

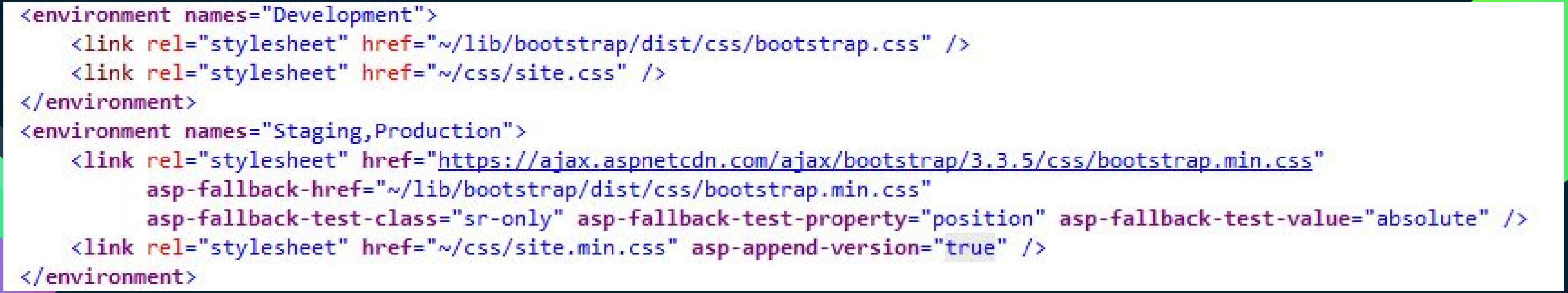
Tag helpers

- An HTML-friendly development experience
- Front-end designers conversant with HTML/CSS/JavaScript can edit Razor without learning C# Razor syntax.
- IntelliSense support
- Cleaner (Less Razor/HTML mess in the .cshtml file)

https://docs.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/intro

Environment Tag Helpers

- The Environment Tag Helper is used (typically in conjunction with the Link and Script Tag Helpers) to render different HTML in a **developer** vs. **test** vs. **production** environment.



```
<environment names="Development">
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />
    <link rel="stylesheet" href="~/css/site.css" />
</environment>
<environment names="Staging,Production">
    <link rel="stylesheet" href="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.5/css/bootstrap.min.css"
          asp-fallback-href="~/lib/bootstrap/dist/css/bootstrap.min.css"
          asp-fallback-test-class="sr-only" asp-fallback-test-property="position" asp-fallback-test-value="absolute" />
    <link rel="stylesheet" href="~/css/site.min.css" asp-append-version="true" />
</environment>
```

Partial View

- A view rendered in another view :)
- Breaking large views into smaller parts
- Reducing copy-pasting

https://docs.microsoft.com/en-us/aspnet/core/mvc/views/partial

View Components

- Similar to PartialViews
- Renders a chunk rather than a full response
- No model binding
- Most often used from a layout view
- Intended for rendering reusable logic that is too complex for PartialViews
- Views are located usually in /Views/Shared/Components/Foo/Default.cshtml

https://docs.microsoft.com/en-us/aspnet/core/mvc/views/view-components

View Components in MVC 6

```
//[ViewComponent(Name = "ProductDetails")]
public class ProductDetailsViewComponent : ViewComponent
{
private readonly IProductsRepository _repository;

public ProductDetailsViewComponent(IProductsRepository repository)
{
  _repository = repository;
}

public IViewComponentResult Invoke(int id)
{
  var product = _repository.GetProduct(id);
  return View(product);
}
```

```
@Component.Invoke("ProductDetails", 1)
// Or as async, if InvokeAsync is implemented in the view component
@await Component.InvokeAsync("ProductDetails", 1)
```

Uploading a file

- Forms must contain `enctype="multipart/form-data"`
- Inputs should be of type “file”
- And on the server side you should bind files to IFormFile type properties

https://docs.microsoft.com/en-us/aspnet/core/mvc/models/file-uploads

Entity Framework Core

- Lightweight, extensible, and cross-platform version of the popular Entity Framework data access technology
- Object-relational mapper (O/RM)

https://docs.microsoft.com/en-us/ef/core/

SQL Server localDB

- A lightweight version of the SQL Server Express Database Engine that is targeted for program development
- SQL Server Object Explorer (show)
- Dev command prompt -> sqllocaldb info

https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/working-with-sql

Code First

- Start storing and retrieving information in SQL Server without creating a database schema or opening a designer. Instead, you write plain C# classes and EF figures out how, and where, to store instances of those classes
- Easy when developing, and in the early, volatile stages

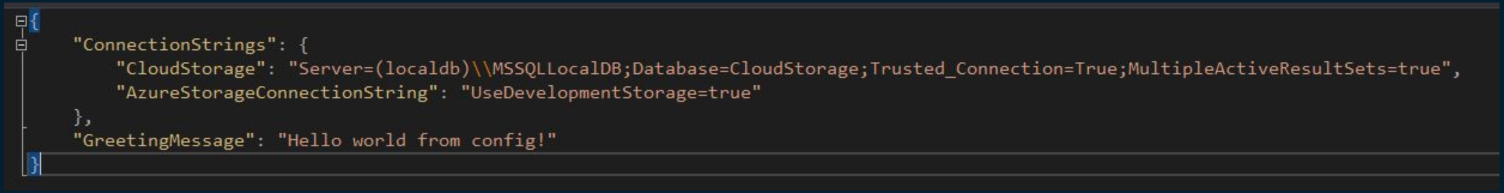
https://docs.microsoft.com/en-us/ef/core/get-started/aspnetcore/new-db

Data Context

- EF supports database-first, model-first and code-first styles of development; the MVC scaffolders use code-first style!
- The gateway to the database is a class derived from EF’s DbContext class.
- Main class that coordinates Entity Framework functionality for a given data model is the database context class

Configure connection to the database

- What database will **CloudStorage** use? Without a specific connection configured, EF tries to connect to a LocalDB instance of SQL Server and find a database with the same name as the DbContext derived class.



```json
{
    "ConnectionStrings": {
        "CloudStorage": "Server=(localdb)\\MSSQLLocalDB;Database=CloudStorage;Trusted_Connection=True;MultipleActiveResultSets=true",
        "AzureStorageConnectionString": "UseDevelopmentStorage=true"
    },
    "GreetingMessage": "Hello world from config!"
}
```

Migrations

- Keep the database in sync with the ever changing code
- Have historical view of the database structure
- You can upgrade / downgrade to specific versions

https://docs.microsoft.com/en-us/aspnet/core/data/ef-mvc/migrations

Seeding initial data

- Well it would be nice to have some initial data
- Insert manually some dummy data each time we startup the application if the database is empty

Wrap up!

- Separating internals from UI with ViewModels
- How to validate input data
- How can we customize the general look of our application using layouts, view start & imports
- The modern and clean way to write view files using tag helpers
- Separating presentation and business logic with partial views or view components
- Upload files from the browser
- Creating, maintaining, upgrading and seeding a new database from code using EF Core Code First

THANK YOU!

Any questions?

You can find me at

emil.craciun@visma.com

DAY 5

Welcome back!

2 May 2016

Emil Craciun, Visma

Remember from last time?

GO!

Agenda for this day

- EF (what was left from last time)
- Azure Blob Storage
- Filters
- ASP.NET Identity

EF (what was left from last time)

- Create DbContext class
- Hook up EF in Startup.cs
- Modify connection string
- Add initial migration
- Create the database
- Add seed data

Azure Storage Account

- Blob storage
- Queue storage
- Table storage
- File storage

https://docs.microsoft.com/en-us/azure/storage/storage-introduction

Azure Blob Storage

- Service for storing large amounts of unstructured data
- Accessed from anywhere in the world via HTTP or HTTPS
- Can be anything
- Storage blobs live in containers

https://docs.microsoft.com/en-us/azure/vs-azure-tools-connected-services-storage

https://docs.microsoft.com/en-us/azure/storage/vs-storage-aspnet5-getting-started-blobs

Azure Blob Storage

- Let’s look at the Azure Portal
- Azure Storage Emulator
- Azure Storage Explorer - http://storageexplorer.com/

Security

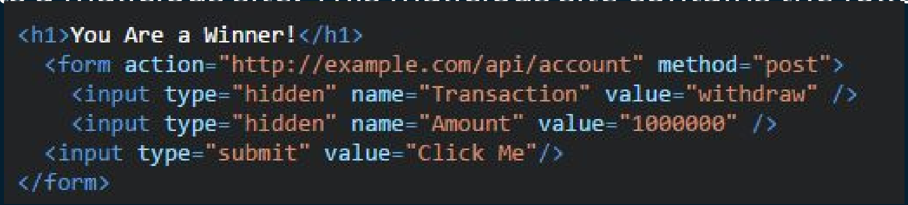
- Well we can talk for months about this subject, but let’s just have a tiny bit discussions about this very important subject

https://docs.microsoft.com/en-us/aspnet/core/security/

Anti-request-forgery

XSRF / CSRF

1. A user logs into www.example.com, using forms authentication.
2. The server authenticates the user and issues a response that includes an authentication cookie.
3. The user visits a malicious site. This malicious site contains the following HTML form:



```
<h1>You Are a Winner!</h1>
  <form action="http://example.com/api/account" method="post">
    <input type="hidden" name="Transaction" value="withdraw" />
    <input type="hidden" name="Amount" value="1000000" />
  <input type="submit" value="Click Me"/>
</form>
```

https://docs.microsoft.com/en-us/aspnet/core/security/anti-request-forgery

Anti-request-forgery

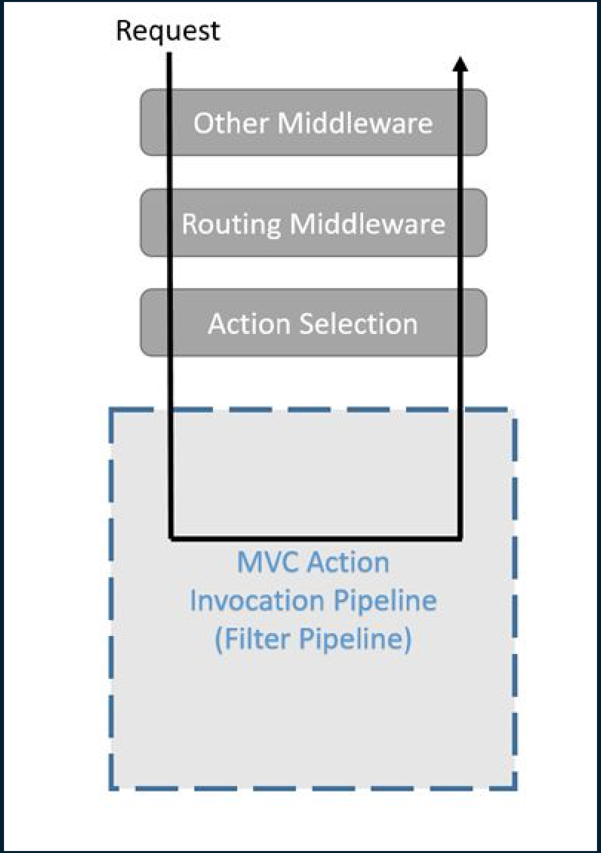
- Most common mitigation: synchronizer token pattern (STP)
  - The server sends a unique and unpredictable token associated with the current user's identity to the client
  - The client must send back the token to the server for verification
  - If the server receives a token that doesn't match the authenticated user's identity, the request should be rejected

Filters

- Filters in ASP.NET Core MVC allow you to run code before or after certain stages in the request processing pipeline.

https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/filters

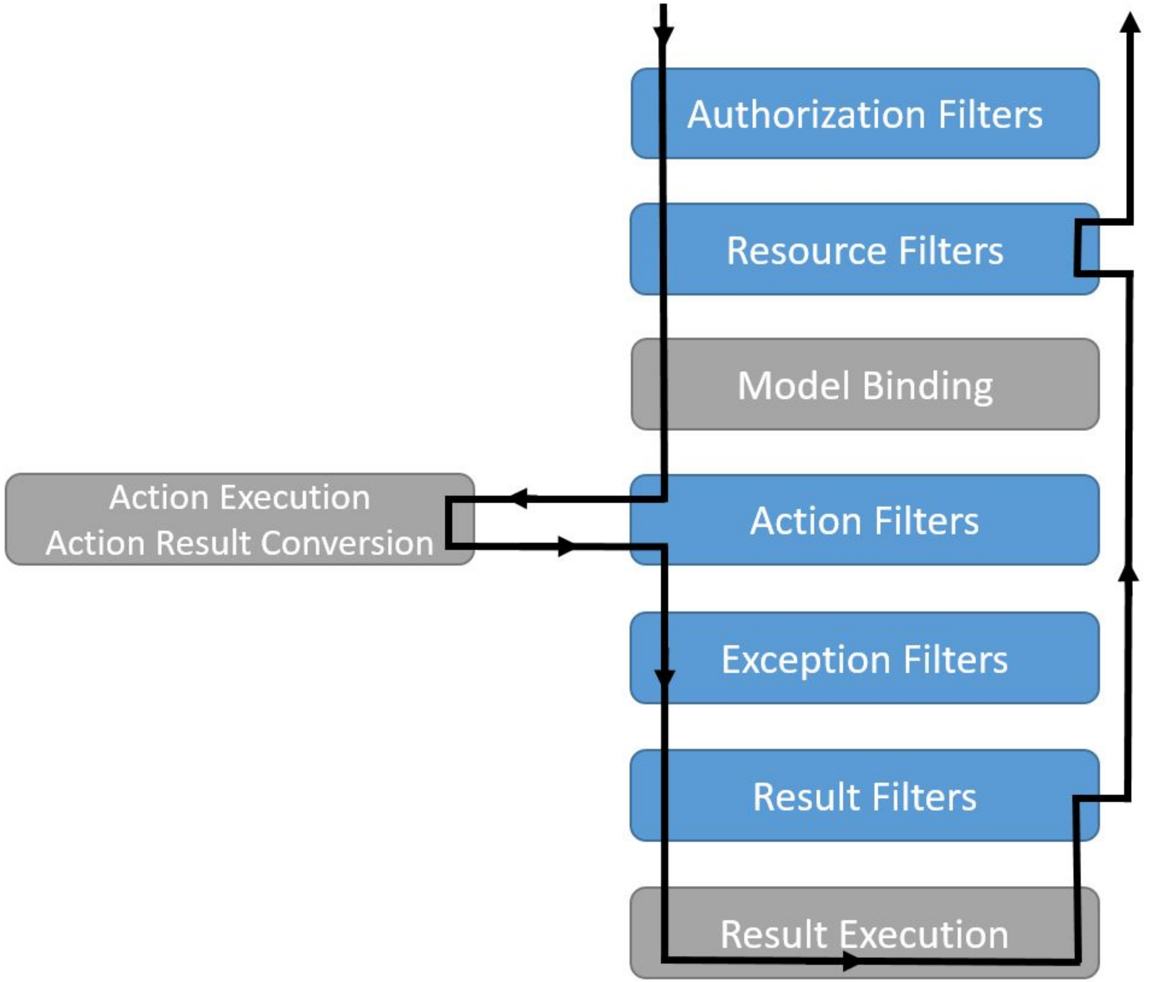
Filters


Request
Other Middleware
Routing Middleware
Action Selection
MVC Action
Invocation Pipeline
(Filter Pipeline)

Filters

Filter types:

- Authorization
- Resource
- Action
- Exception
- Result



ASP.NET Core Identity

- A membership system which allows you to add login functionality to your application

https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity

Authentication

- Determines who a user is
- Microsoft.AspNetCore.Identity.EntityFrameworkCore
- DbContext -> IdentityDbContext<User>
- And of course hook it up in Startup.cs
- (optional) Configure identity options

https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity

Authentication

- Everything starts with the [Authorize] attribute (actually this one belongs to the next part, but this is how we will see that the authentication is working)
- UserManager
- SignInManager

Wrap up!

- How to create a database from code with EF
- How we can do CRUD operations on blobs in Azure
- Enforce authentication and authorization rules in our application

THANK YOU!

Any questions?

You can find me at

emil.craciun@visma.com

DAY 6

Welcome back!

3 May 2016

Emil Craciun, Visma

Remember from last time?

GO!

Agenda for this day

- Authorization (continue from last day)
- Bower & UI 3rd party components
- Unit testing
- Service injection in views
- Logging
- SignalR

Authorization

- Determines what a user can do, once he is identified
- Independent from authentication

https://docs.microsoft.com/en-us/aspnet/core/security/authorization/introduction

Simple authorization

- This is the [Authorize] attribute we saw before
- Basically it only checks if a user is authenticated or not

https://docs.microsoft.com/en-us/aspnet/core/security/authorization/simple

Role based authorization

- A way of grouping multiple users and give them certain rights in the application
- A user can have to multiple roles
- This is the more “classical” way

https://docs.microsoft.com/en-us/aspnet/core/security/authorization/roles

Claims based authorization

- A claim is a key-value pair stating what a user **is**, and not what a user **can do**
- Claims are issued by a trusted party
- An identity can contain multiple claims with multiple values and can contain multiple claims of the same type

https://docs.microsoft.com/en-us/aspnet/core/security/authorization/claims

Bowr

- "A package manager for the web."
- Static content files -> client side libraries

https://docs.microsoft.com/en-us/aspnet/core/client-side/bower

Bootstrap

- most popular web framework for developing responsive web applications
- css + js

https://docs.microsoft.com/en-us/aspnet/core/client-side/bootstrap

Testing

Is an extremely important part of software development, and I, personally, think it should be mandatory.

Unit Testing

- The **smallest testable** parts of an application, called units, are **individually** and **independently** scrutinized for proper operation

https://docs.microsoft.com/en-us/dotnet/articles/core/testing/unit-testing-with-dotnet-test

Integration Testing

- Ensures that an application's components function correctly when assembled together
- ASP.NET Core supports integration testing using unit test frameworks and a built-in test web host that can be used to handle requests without network overhead
- They verify that different parts of an application work correctly together
- Frequently involve application infrastructure concerns, such as a database, file system, network resources, or web requests and responses

https://docs.microsoft.com/en-us/aspnet/core/testing/integration-testing

Testing Controllers

- Controllers in ASP.NET MVC apps should be small and focused on user-interface concerns
- Controller logic should be minimal and not be focused on business logic or infrastructure concerns
- Test controller logic, not the framework
- Test how the controller behaves based on valid or invalid inputs
- Test controller responses based on the result of the business operation it performs

https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/testing

Testing Controllers

- Verify ModelState.IsValid
- Return an error response if ModelState is invalid
- Retrieve a business entity from persistence
- Perform an action on the business entity
- Save the business entity to persistence
- Return an appropriate IActionResult

Injecting Services to Views

```
@inject MyApp.Services.IGreeter Greeter
<html>
     <body>
          <h3>Today’s greeting message is:</h3>
          @Greeter.GetGreeting()
     </body>
</html>
```

- Preferable than using static classes/methods
- Use interfaces instead of concrete types
- Register in IoC using different lifecycles

https://docs.microsoft.com/en-us/aspnet/core/mvc/views/dependency-injection

Logging

- ASP.NET Core supports a logging API that works with a variety of logging providers
- Built-in providers let you send logs to one or more destinations, and you can plug in a third-party logging framework
- Microsoft.Extensions.Logging.Console
- Microsoft.Extensions.Logging.Debug

https://docs.microsoft.com/en-us/aspnet/core/fundamentals/logging

WebSockets

- Two-way persistent communication protocol over TCP connection
- Adds real-time functionality in web applications

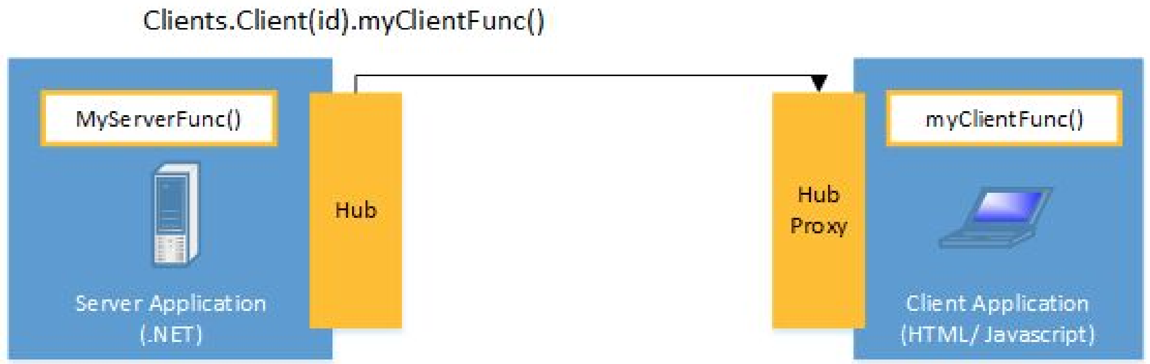
https://docs.microsoft.com/en-us/aspnet/core/fundamentals/websockets

SignalR

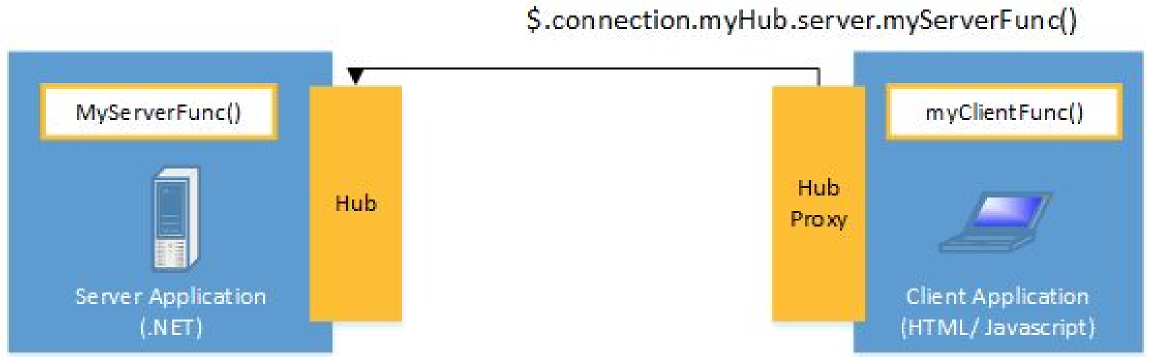
- Firstly add new nuget source:
  https://dotnet.myget.org/F/aspnetcore-ci-dev/api/v3/index.json
- Library that simplifies two-way communication

https://www.asp.net/signalr

https://docs.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr


Clients.Client(id).myClientFunc()
MyServerFunc()
Hub
Server Application (.NET)
Hub Proxy
myClientFunc()
Client Application (HTML/ Javascript)

Server invocation of client method myClientFunc()


$.connection.myHub.server.myServerFunc()
MyServerFunc()
Hub
Server Application (.NET)
Hub Proxy
myClientFunc()
Client Application (HTML/ Javascript)

Client invocation of server method MyServerFunc()

Wrap up!

- Authorization with roles and claims
- Client side package management with Bower, and easy styling with Bootstrap
- Why and how should we test our code
- How to inject services into views
- Short introduction into logging
- Two-way communication with SignalR

https://github.com/ecraciun/VismaACLabs

THANK YOU!

Any questions?

You can find me at

emil.craciun@visma.com