



Code.Hub

The first Hub for Developers

MVC

(Model-View-Controller)

Konstantinos Athanasoglou

Introduction

- Web vs Desktop (Develop, Deploy, Update, Debug)
- Web Sites vs Web Applications (Apps)

Terminology

- URLs – Resources

<https://google.com>, <ftp://ntua.gr/ubuntu>, <mailto:gpal@best.gr>

- HTTP – The *language*
- MVC – The *pattern*
- [ASP.NET Core MVC v2.1](#) – The *framework* (**.NET Core** is the *runtime*)

[Mozilla Developer Network \(MDN\)](#) is your friend

HTTP (1 / 4) - Introduction

Web Server: What do you think when you hear it?

- Physical Machine
- Application

Browser: What happens when you hit Enter?

- DNS Lookup, retrieve IP from Host Name
- Browser (*client*) opens connection to web server's address (*server*)
- Client sends request to Server (*HTTP Request*)
- Client receives response from Server (*HTTP Response*)
- Client retrieves content (*HTML*) + Renders it

HTTP (2/4) – HTTP Methods (Verbs)

HTTP Verbs convey *meaning / intention*

GET: Gets a resource. Most common verb. No state change.

POST: Post data to app -> Action taken (e.g. register account)

PUT: Update (replace). Replace current resource with payload

PATCH: Update (modify). Apply partial modification to resource

DELETE: Delete resource

Demo

POST <https://httpverbs.azurewebsites.net/tokens>

Set HTTP Header *X-Token* to value obtained.

GET <https://httpverbs.azurewebsites.net/posts>

GET <https://httpverbs.azurewebsites.net/posts/12>

GET <https://httpverbs.azurewebsites.net/posts/12/tags>

GET <https://httpverbs.azurewebsites.net/posts?authorId=33>

HTTP (3/4) – HTTP Status Codes

200 (OK): Request Succeeded.

201 (Created): Resource created (e.g. after POST)

301/308 (Moved *permanently*) 308 guarantees same verb

302/307 (Moved *temporarily*) 307 guarantees same verb

304 (Not Modified) Send header, but no content (e.g. cached)

400 (Bad Request): Server cannot understand request
(Domain validation, Bad syntax, Missing Params)

401 (Unauthorized): Missing or invalid auth headers.

403 (Forbidden): Authenticated, but has no permissions.

404 (Not Found): Resource not found (or masked 401/403)

500 (Internal Server Error): Code Exception.

502 (Bad gateway): Gateway got invalid response

503 (Service unavailable): Server down/overloaded

504 (Gateway timeout): Gateway got no response

2xx Successful Responses

3xx Redirection messages

4xx Client Errors

5xx Server Errors

HTTP (4/4) – HTTP Request Sample

POST / HTTP/1.1

Host: myserver.com

Content-Type: application/json

Content-Length: 56

```
{  
  "firstname": "Johnny",  
  "lastname": "Mnemonic"  
}
```

HTTP Verb

HTTP Headers

HTTP Body

MVC (Model – View – Controller)

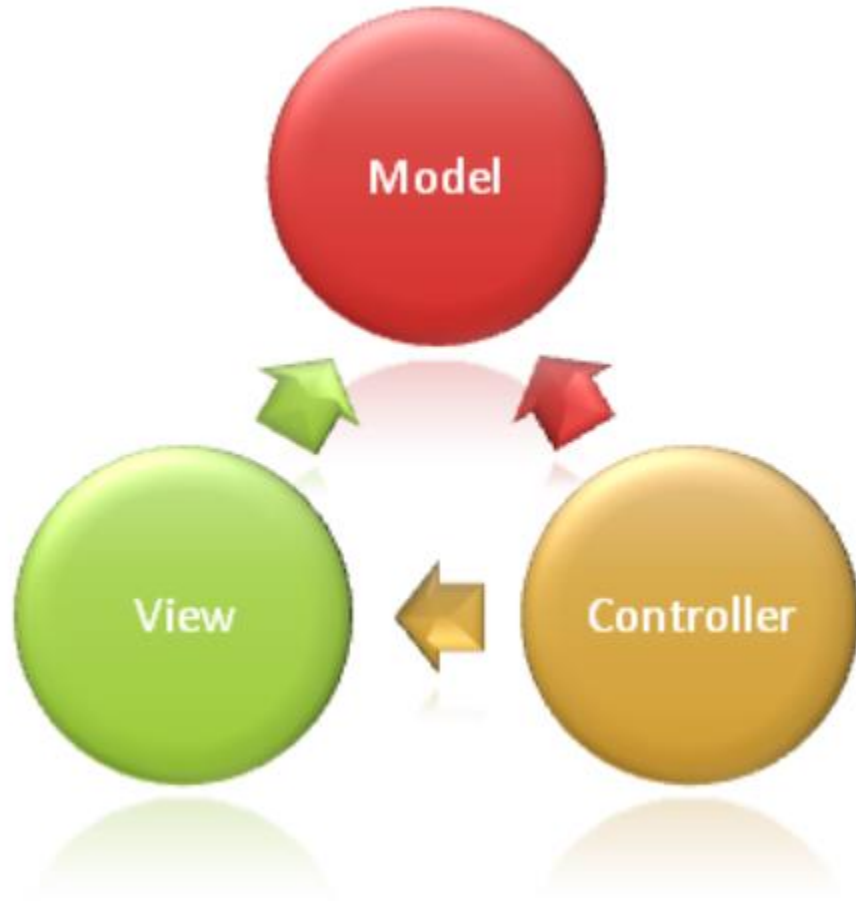
MVC: *Design Pattern* (Separation of Concerns). Popular in Web.

Model: A web app to be functional needs:
data and business logic.

Controllers: Web Apps interact with user through HTTP.
Way to map URLs to methods.

Views: Data needs to be *displayed*, in different formats
(e.g. web/mobile).

MVC (Model – View – Controller)

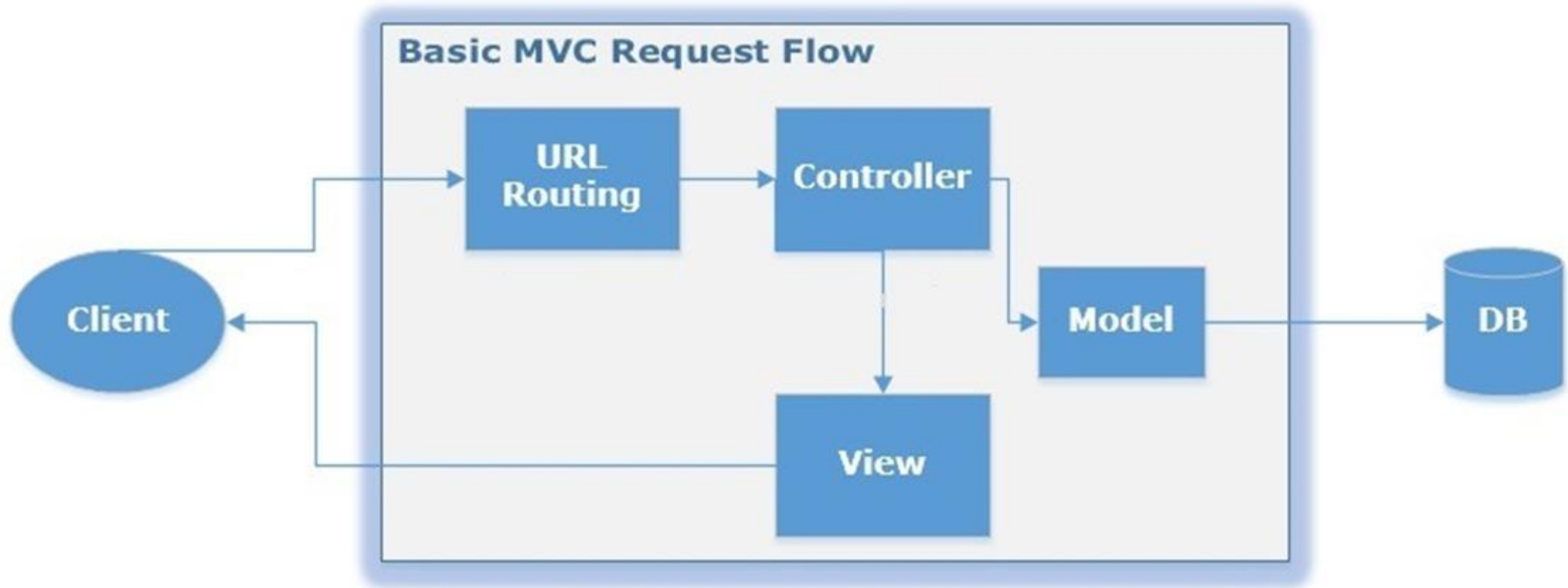


View, Controller *depend* on model

Model depends on **neither**

Controllers shouldn't be overly complicated. Mainly **routing**.

MVC (Model – View – Controller)



ASP.NET Core MVC – What is it?

- Microsoft's solution to MVC.
- Open Source (<https://github.com/aspnet/Home>)
- Cross-platform: Build on one platform. Run everywhere (Windows/Linux/Mac).
- Cloud-based: Easily deploy to Cloud (e.g. Azure)
- Small footprint (Package based), Very fast!!

ASP.NET Core MVC – Project Structure

- Project Structure
- Services & Middleware
- launchsettings.json / appsettings.json - Configuration
- IIS Express or Self-hosted
- Run!

ASP.NET Core MVC - Routing

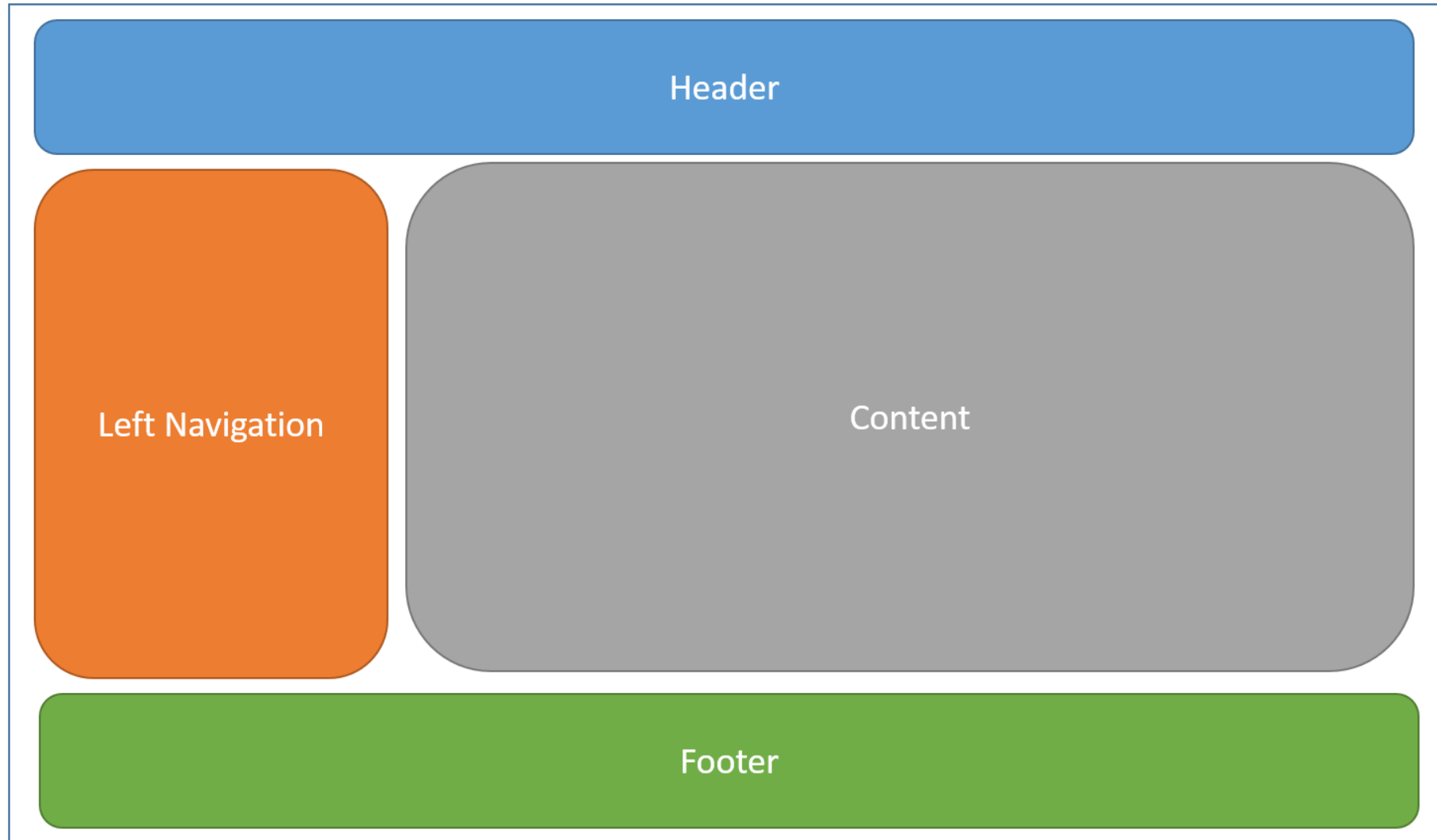
- **Conventional Routing:**
 - *Convention* for URLs: {controller}/{action}/{id}
 - Execute in order defined
 - Controller/Action name **do** play role.
- **Attribute Routing:**
 - Define routes closer to actions.
 - Greater flexibility/control (more code).
 - Controller/Action name **don't** play role.
 - [Route], [HttpGet], [HttpPost], ...
- **Mixed Routing:** Default conventional,
unless [Attribute] used (**overrides**)

ASP.NET Core MVC - Controllers

- **Controllers/** folder
- Inherit *Controller* (view) or *ControllerBase* (no view)
- Return *ActionResult* or *ActionResult<T>*
- return:
 - View(),
 - Content(),
 - Empty(),
 - File(),
 - Json(),
 - Redirect()
 - PartialView(), ViewComponent()

Should Validate Data?

ASP.NET Core MVC – Views (Layout Example)



ASP.NET Core MVC - Views

- **Views/** folder
- Display data, Interacts with User (quick validation)
- .cshtml files, (default: *Index.cshtml*)
- **_Layout**: Layout page: DRY, provides consistency
must: RenderBody(), may: RenderSection()
- **_ViewStart**: Code that runs before every page
- **_ViewImports**: Using directives / Tag Helpers /
DI for all pages

ASP.NET Core MVC – Passing data to Views

- Loosely-typed: ViewData[“Key”] or ViewBag.Key

Avoid! Use sparingly, e.g. for Title

- Strongly-typed: View Models

Recommended! More structured

ASP.NET Core MVC – Razor (C# in HTML)

- Can be mixed

```
<p>@DateTime.Now</p>
```

- Code blocks

```
@{  
    var quote = "The future depends on what you do today.";  
}  
<p>@quote</p>
```

- Control Logic

```
@foreach, @for, @while, @do  
@if, else, else if, @switch
```

- Reference Model from Controller

```
@model
```

ASP.NET Core MVC – Model

```
public class Movie
{
    public int Id { get; set; }

    public string Title { get; set; }

    public DateTime ReleaseDate { get; set; }

    public string Genre { get; set; }

    public decimal Price { get; set; }
}
```

ASP.NET Core MVC – Model Validation

Required: **[Required]**

Data Types: **[DataType(DataType.*)]**

Format:

- **[CreditCard]**, **[EmailAddress]**, **[Phone]**, **[Url]**
- **[Range]**: Validates value falls within given range.
- **[RegularExpression]**: Validates data matches specified RegEx.
- **[StringLength]**: Validates string has at most given max length.

Other:

- **[Compare]**: Validates two properties in a model match.
- **[Remote]**: Server-side call

ASP.NET Core MVC – Tag Helpers (Overview)

```
<form asp-controller="Demo" asp-action="Register" method="post">  
<!-- Input and Submit elements -->  
</form>
```

```
<label asp-for="Email"></label>  
<input asp-for="Email" /> <br />
```

```
<textarea asp-for="Description"></textarea>
```

```
<span asp-validation-for="Email"></span>  
<div asp-validation-summary="ModelOnly"></div>
```

```
// SelectList (string)  
<select asp-for="Country" asp-items="Model.Countries"></select>
```

```
// SelectList (Enum)  
<select asp-for="EnumCountry"  
    asp-items="Html.GetEnumSelectList<CountryEnum>()" />
```

ASP.NET Core MVC – Tag Helpers (Select from string)

```
<select asp-for="Country" asp-items="Model.Countries"></select>
```

```
public class CountryViewModel
{
    public string Country { get; set; }

    public List<SelectListItem> Countries { get; } = new List<SelectListItem> {
        new SelectListItem { Value = "MX", Text = "Mexico" },
        new SelectListItem { Value = "CA", Text = "Canada" },
        new SelectListItem { Value = "US", Text = "USA" },
    };
}

public IActionResult IndexOption(int id)
{
    var model = new CountryViewModel();
    model.Country = "CA";
    return View(model);
}
```

ASP.NET Core MVC – Tag Helpers (Select from Enum)

```
public class CountryEnumViewModel
{
    public CountryEnum EnumCountry { get; set; }
}
```

```
public enum CountryEnum
{
    [Display(Name = "United Mexican States")]
    Mexico,
    [Display(Name = "United States of America")]
    USA,
    Canada,
    France,
    Germany,
    Spain
}
```

```
<select asp-for="EnumCountry" asp-items="Html.GetEnumSelectList<CountryEnum>()"> >
</select>
```

ASP.NET Core MVC – Partial Views

1. Reusable code shared among many views.
2. More readable code (like methods).
3. Don't run `_ViewStart.cshtml` (like normal views)

```
<!-- name: required. use simple name for view discovery or explicit path -->  
<!-- for: use to pass model to partial view (use for or model) -->  
<partial name="Shared/_ProductPartial.cshtml" for="Product" />
```

```
<!-- model: use to pass model to partial (use model or for) -->  
<partial name="_ProductPartial" model='new Product {  
    Number = 1, Name = "Test product", Description = "This is a test" }' />
```

```
<!-- pass view data to partial -->  
<partial name="_ProductViewDataPartial" for="Product" view-data="@ViewData" />
```


Thank You!

Bonus: Create Model from Existing Db

// ef core for db provider you want

Install-Package Microsoft.EntityFrameworkCore.SqlServer

// create model from db

Install-Package Microsoft.EntityFrameworkCore.Tools

// asp.net core scaffolding tools

Install-Package Microsoft.VisualStudio.Web.CodeGeneration.Design

// reverse engineer model

Scaffold-DbContext

"Server=(localdb)\mssqllocaldb;Database=Movies;Trusted_Connection=True;"

Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models

Bonus: Create Model from Existing Db

Add DbContext in DI services (ConfigureServices):

```
var connection = @"Server=(localdb)\mssqllocaldb;  
    Database=Movies;  
    Trusted_Connection=True;  
    ConnectRetryCount=0";  
  
services.AddDbContext<MoviesContext>(  
    options => options.UseSqlServer(connection));
```

Fine-tune model using EF Core Fluent API (precedence over Attributes):

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