ASP.NET Core Introduction

ASP.NET Core, MVC and ASP.NET with Databases

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Questions?

sli.do

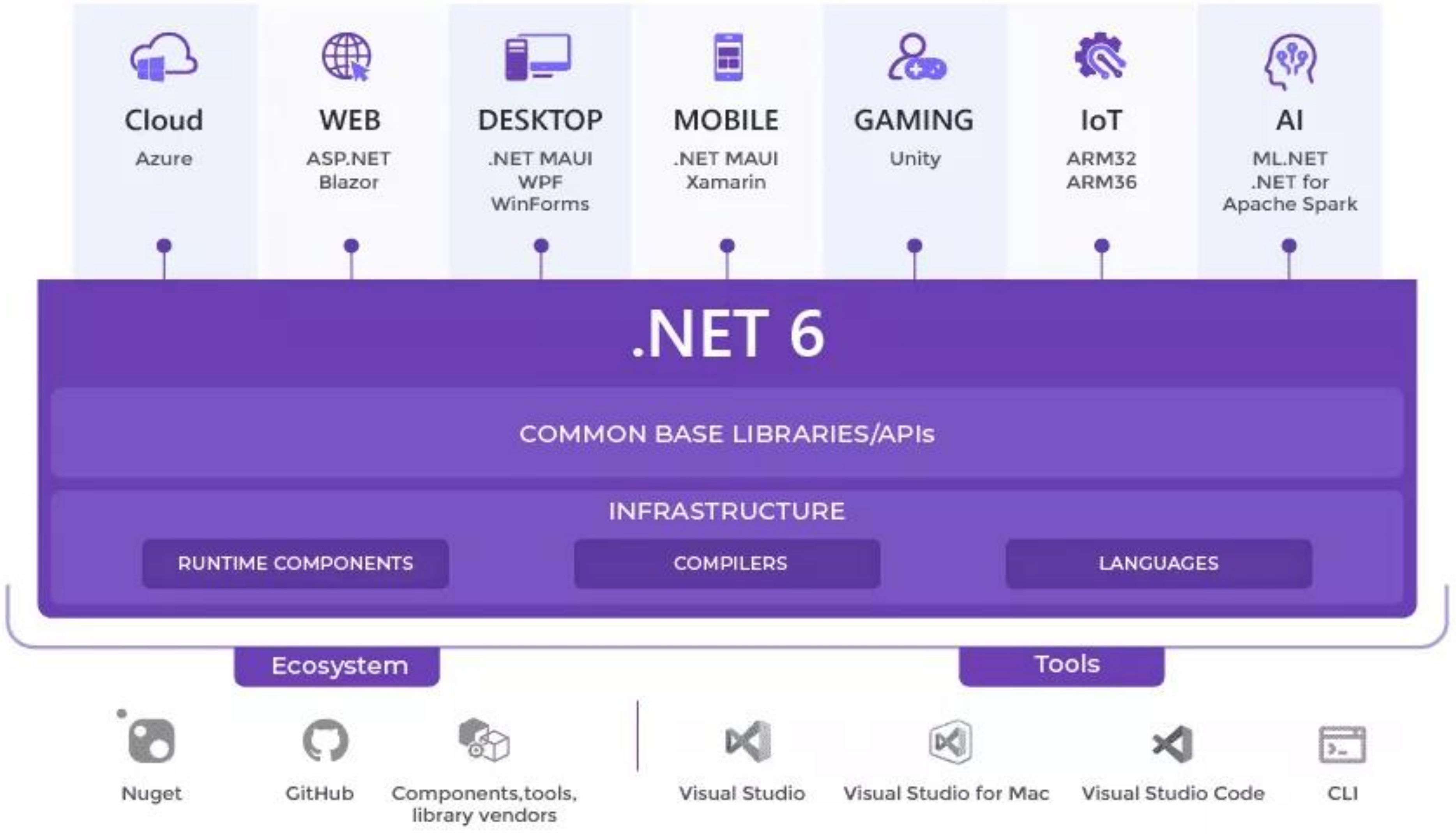
# QA-Auto-BackEnd

Table of Content

ASP.NET Core

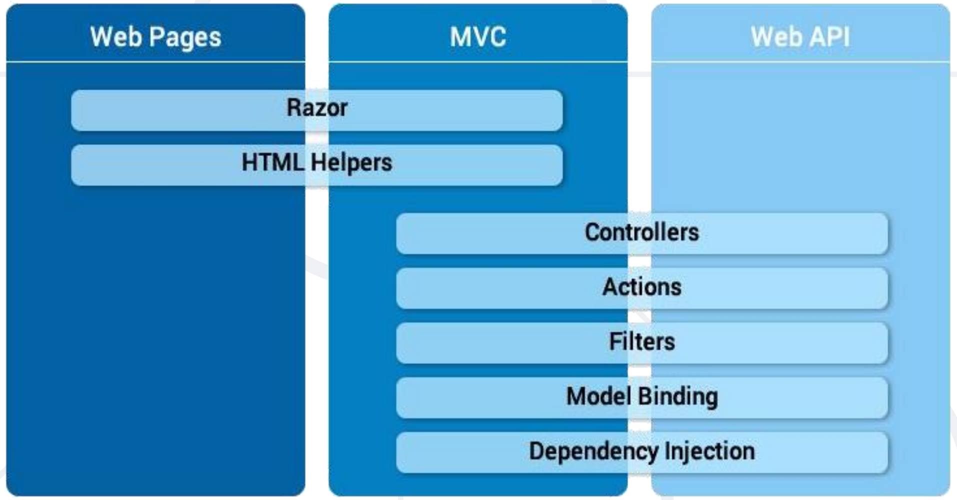
Overview

.NET Core: Bird's Eye View



ASP.NET Core Overview

- **ASP.NET Core** is a cross-platform open-source back-end development framework for C#



  - ASP.NET Core **Web Pages**: build simple Web apps
  - ASP.NET Core **MVC**: build server-side Web apps
  - ASP.NET Core **Web API**: build Web services and REST APIs

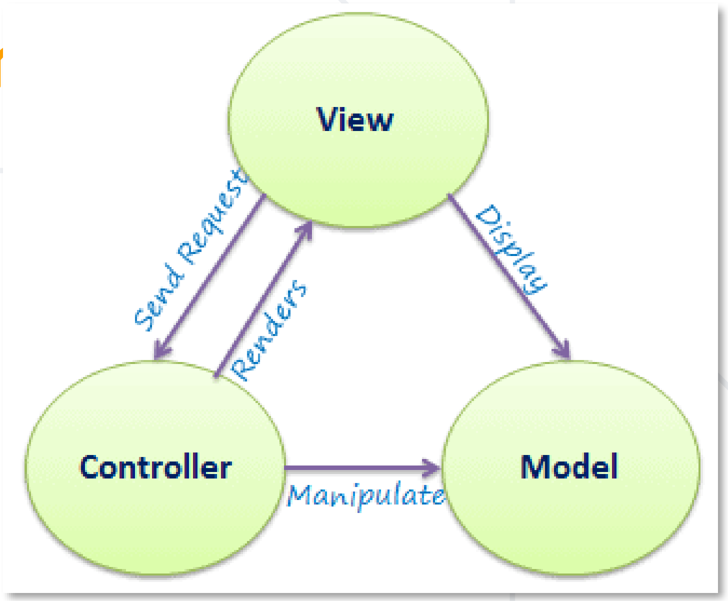
ASP.NET Core Overview

- Great documentation: https://docs.microsoft.com/en-us/aspnet
- **ASP.NET Core** provides
  - Integration of modern client-side frameworks (Angular, React, Blazor, etc.)
  - Development workflows (MVC, WebAPI, Razor Pages, SignalR)
- **ASP.NET Core** applications run both on **.NET Core** and **.NET Framework**

The MVC Pattern

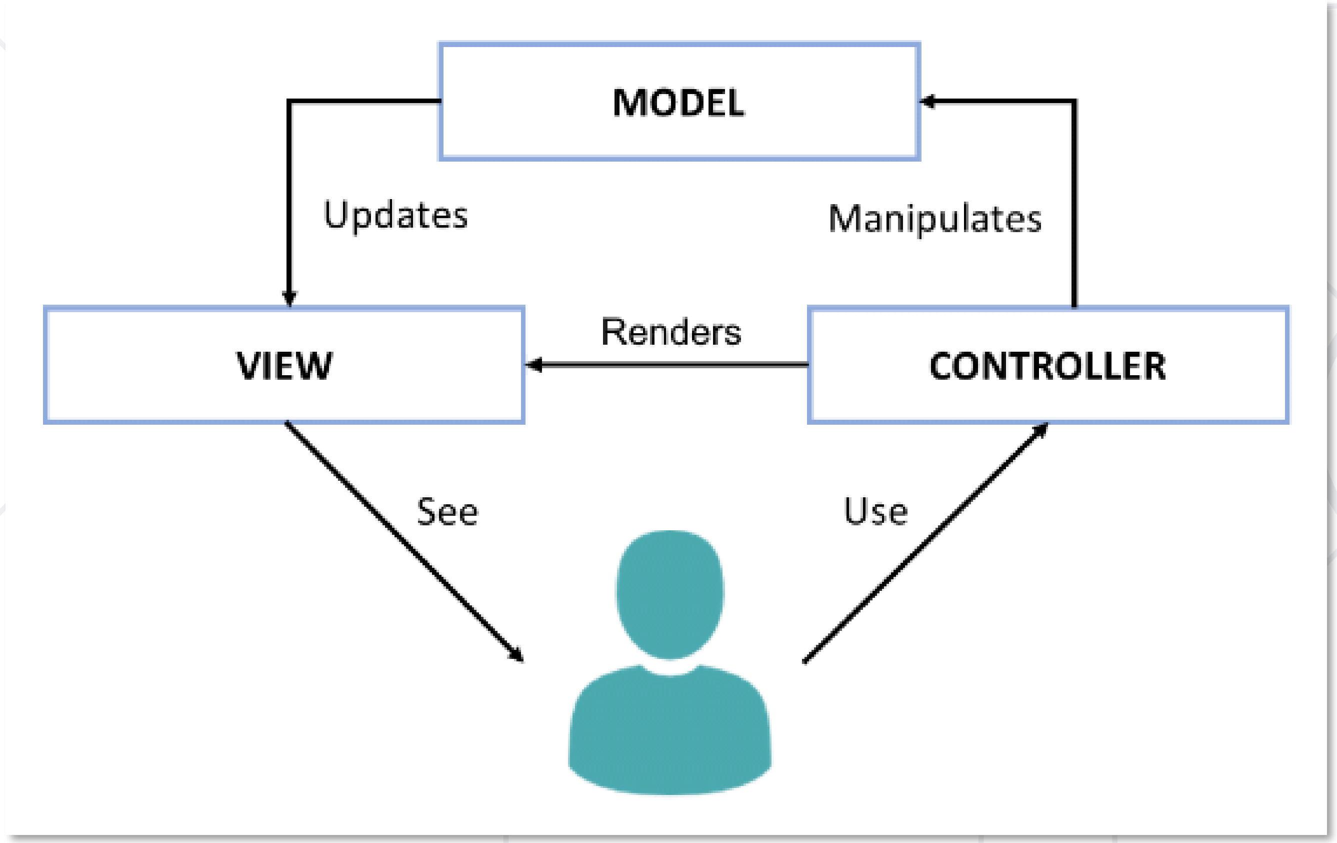
The MVC Pattern

- **Model–View–Controller** (MVC) is a **software architectural pattern**
- Originally formulated in the late 1970s by Trygve Reenskaug as part of the **Smalltalk** (object-oriented programming language)
- **Code reusability** and **separation of concern**
- Originally developed for **desktop**, then adapted for **internet applications**



The Model-View-Controller (MVC) Pattern

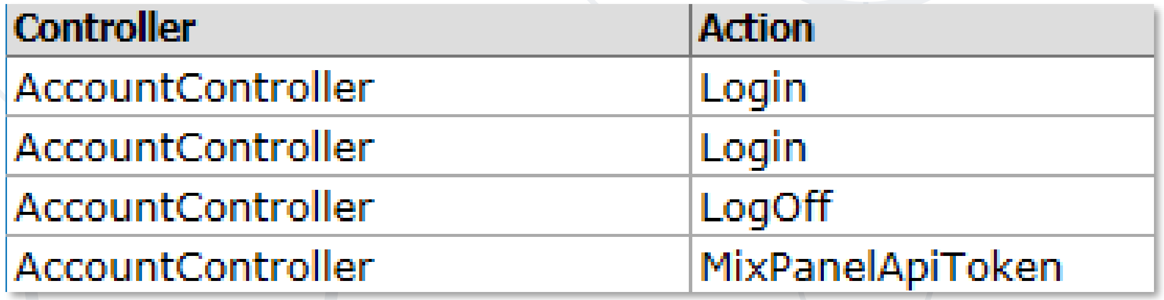
- The **Model-View-Controller** (**MVC**) pattern



- **Controller**
  - Handles user actions
  - Updates the model
  - Renders the view (UI)
- **Model**
  - Holds app data
- **View**
  - Displays the UI, based on the model data

Controller

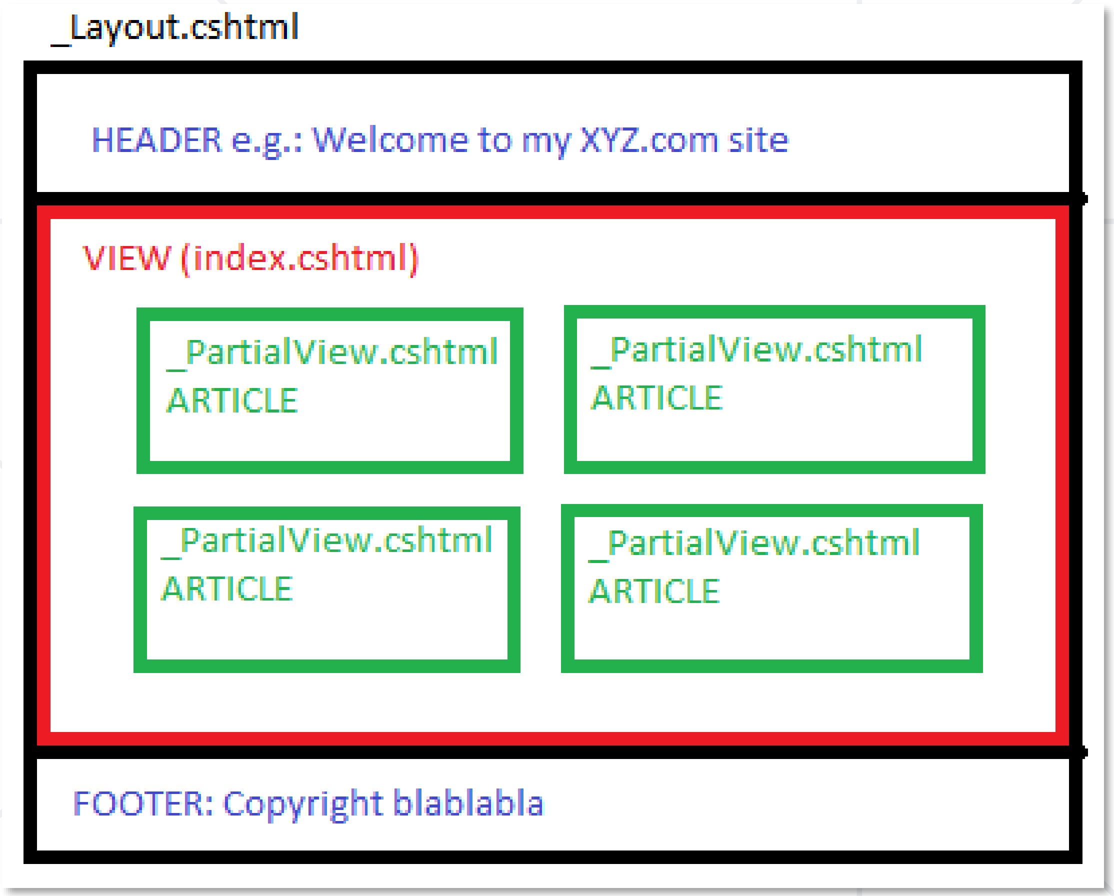
- The **Controller** in **MVC** represents
  - **Processes user's actions** and produces a response
  - Process the requests with the help of **Views** and **Models**
  - A set of classes that handles
    - Communication from the user
    - Overall application flow
    - Application-specific logic
  - Every **Controller** has one or more "**Actions**"



Controller	Action
AccountController	Login
AccountController	Login
AccountController	LogOff
AccountController	MixPanelApiToken

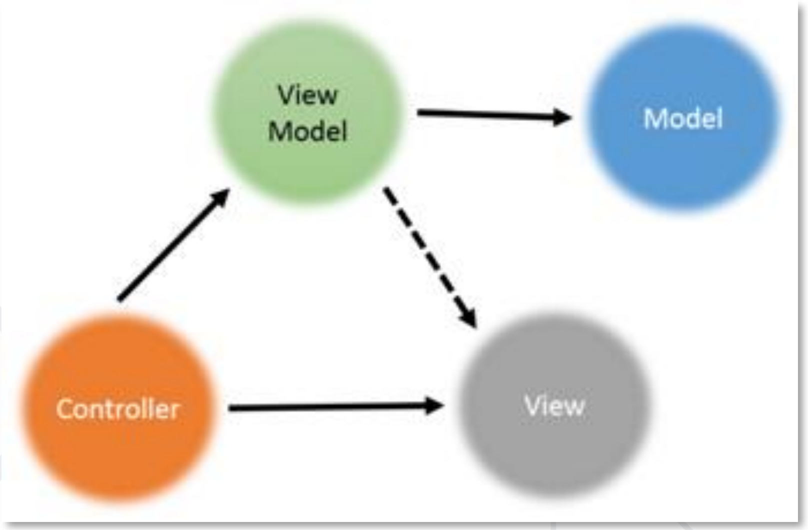
View

- The **View** in **MVC** represents
  - Defines how the application's user interface (**UI**) will be displayed
  - May support **Master Views** (**layouts**) and **Sub-Views** (**partial views** or controls)
  - In Web apps: template to dynamically generate HTML


_Layout.cshtml
HEADER e.g.: Welcome to my XYZ.com site
VIEW (index.cshtml)
_PartialView.cshtml ARTICLE
_PartialView.cshtml ARTICLE
_PartialView.cshtml ARTICLE
_PartialView.cshtml ARTICLE
FOOTER: Copyright blablabla

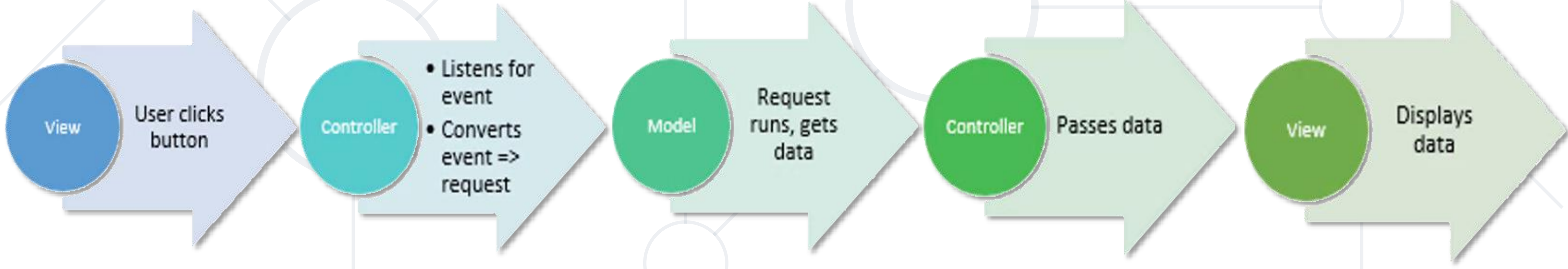
Model

- The **Model** in **MVC** represents
  - A **set of classes** that describes the **data** we display in the UI
  - May contain **data validation rules**
- Two types of models
  - **View model** / **binding model**
    - Maps the UI of the Web page to C# class
    - Part of the **MVC** architecture
  - **Database model** / **domain model**
    - Maps database table to C# class (using ORM)



MVC Steps

- Incoming **Request** routed to **Controller**
- **Controller** processes **Request** and creates a **Model** (view model)
  - Controller also selects **appropriate result** (for example: **View**)
- **Model** is passed to the **View**
- **The View** transforms **Model** into appropriate output format (HTML)
- **Response** is rendered (**HTTP Response**)



Web MVC Frameworks

- **Web MVC frameworks** are used to build Web applications
  - It provides the MVC **structure** and **engine** to build Web apps
  - **Controllers** handle HTTP GET / POST requests and render a view
  - **Views** display HTML + CSS, based on the models
  - **Models** hold app data for views, prepared by controllers
- Examples of Web MVC frameworks
  - **ASP.NET Core MVC** (C#), Spring MVC (Java), Express (JS), Django (Python), Laravel (PHP), Ruby on Rails (Ruby), Revel (Go), …

ASP.NET Core MVC

Overview

ASP.NET Core MVC Overview

- **ASP.NET Core MVC** provides features for building web APIs and web apps
  - Uses the **Model-View-Controller (MVC)** design pattern
  - Lightweight, open source, testable, good tooling
  - **Razor** markup for Razor Pages and MVC views
  - RESTful services with **ASP.NET Core Web API**
    - Built-in support for multiple data formats, content negotiation and CORS
  - Achieve high-quality architecture design, optimizing developer work
    - **Convention over Configuration**
  - **Model binding** automatically maps data from HTTP requests
  - **Model validation** with client-side and server-side validation
  - Often combined with **Entity Framework** for **ORM**

ASP.NET Core MVC Features

- **Routing** for mapping requests
- **Dependency injection** for injecting components at runtime
- Strongly-typed views with the **Razor view engine**
- **Model binding** automatically maps data from HTTP requests
- **Model validation** with client-side and server-side validation
- **Tag helpers** enable server-side code in HTML elements
- Filters, Areas, Middlewares
- Built-in security features
- **Identity** with users and roles
- And many more…

ASP.NET Core MVC Application

What's Inside?

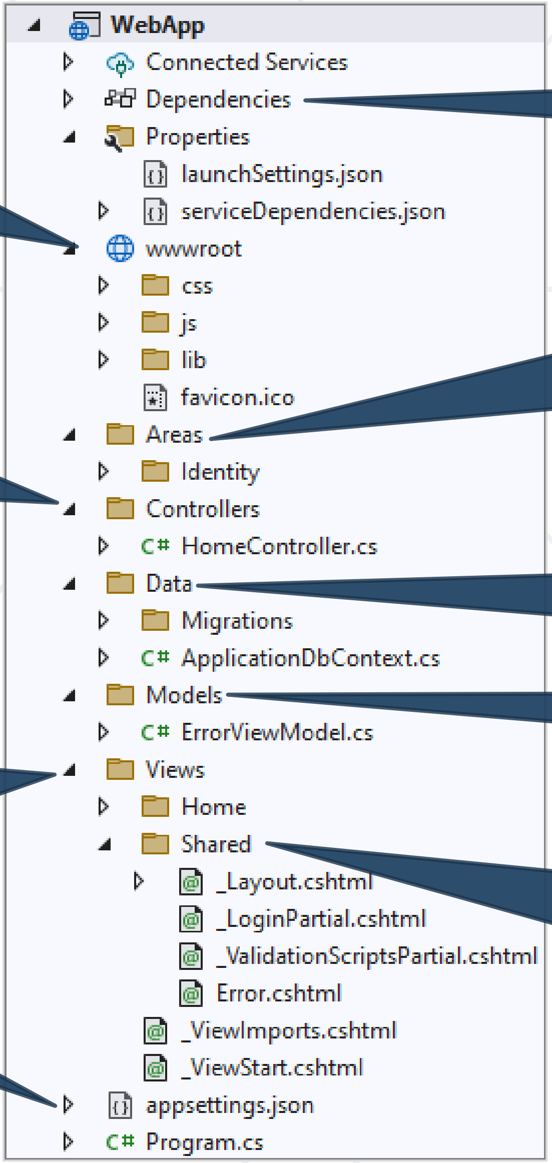
MVC App: What's Inside?

Static files: CSS styles. images, fonts, …

Controller classes holding actions

Views: HTML templates for the pages

App start files



```
◢ WebApp
  ▷ Connected Services
  ▷ Dependencies
  ◢ Properties
      launchSettings.json
    ▷ serviceDependencies.json
  ◢ wwwroot
    ▷ css
    ▷ js
    ▷ lib
      favicon.ico
  ◢ Areas
    ▷ Identity
  ◢ Controllers
    ▷ C# HomeController.cs
  ◢ Data
    ▷ Migrations
    ▷ C# ApplicationDbContext.cs
  ◢ Models
    ▷ C# ErrorViewModel.cs
  ◢ Views
    ▷ Home
    ◢ Shared
      ▷ _Layout.cshtml
        _LoginPartial.cshtml
        _ValidationScriptsPartial.cshtml
        Error.cshtml
      _ViewImports.cshtml
      _ViewStart.cshtml
  ▷ appsettings.json
  ▷ C# Program.cs
```

NuGet packages + Projects References

Areas: physically partition a web app in separate units

Data: EF models + DB context + migrations

Models: view models

Shared views: layout for all pages + partial views

ASP.NET Core MVC Controllers

- All controllers should be in the "**Controllers**" folder
- Controller naming standard should be **{name}Controller**
- Every controller should inherit the **Controller** class
  - Access to **Request**, **Response**, **HttpContext**, **RouteData**, **TempData**, etc.
- Routes select Controllers in every request

```
\Controllers\UsersController.cs
```

```
public class UsersController : Controller
{
  public IActionResult All() => View();
}
```

Mapped to URL "**/Users/All**"

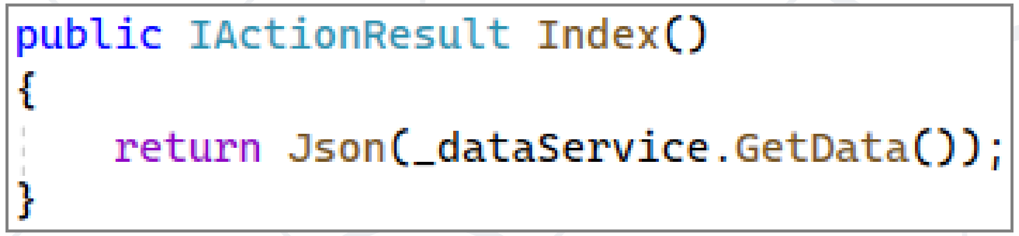
ASP.NET Core MVC Actions

- Actions are the ultimate `Request` destination
  - Public controller methods
  - Non-static
  - No return value restrictions
- Actions typically return an `IActionResult`

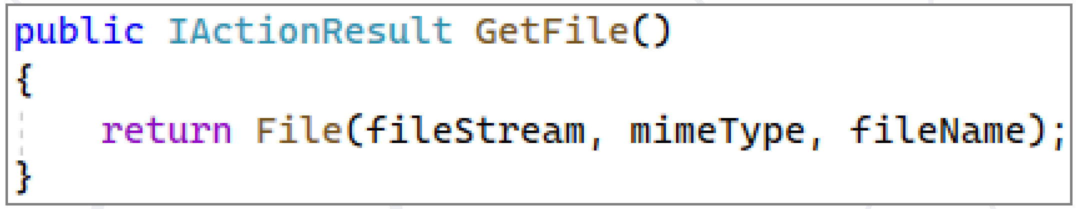
```
public IActionResult Details(int id)
{
    var viewModel = this.dataService.GetById(id).To<DetailsViewModel>();
    return this.View(viewModel);
}
```

Action Results

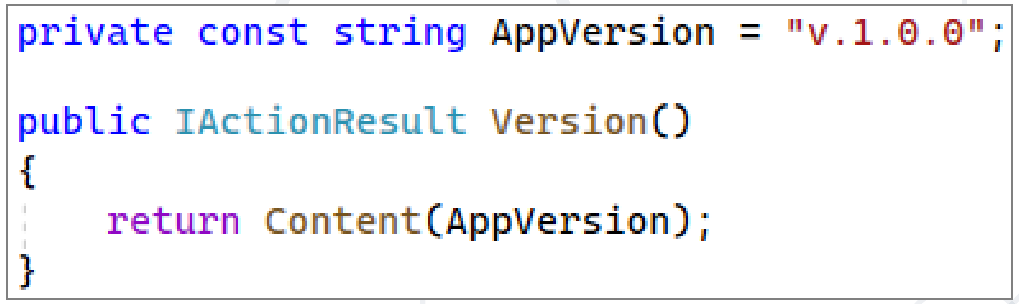
- **Action result** == controller's response to a browser request
  - Represent various **HTTP status codes**
- Inherit from the base **`ActionResult`** class



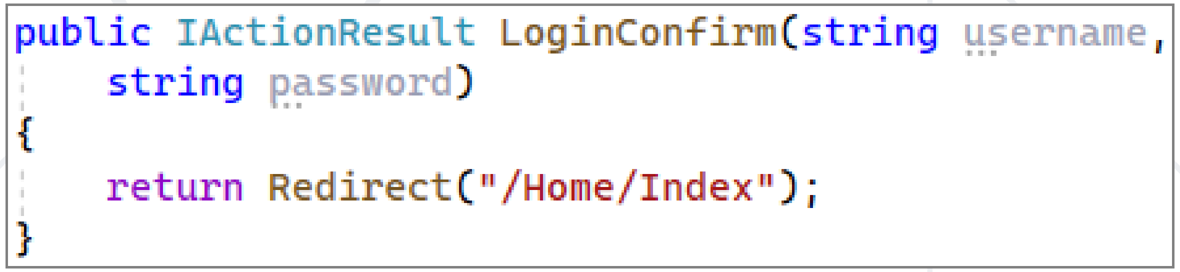
```
public IActionResult Index()
{
    return Json(_dataService.GetData());
}
```



```
public IActionResult GetFile()
{
    return File(fileStream, mimeType, fileName);
}
```



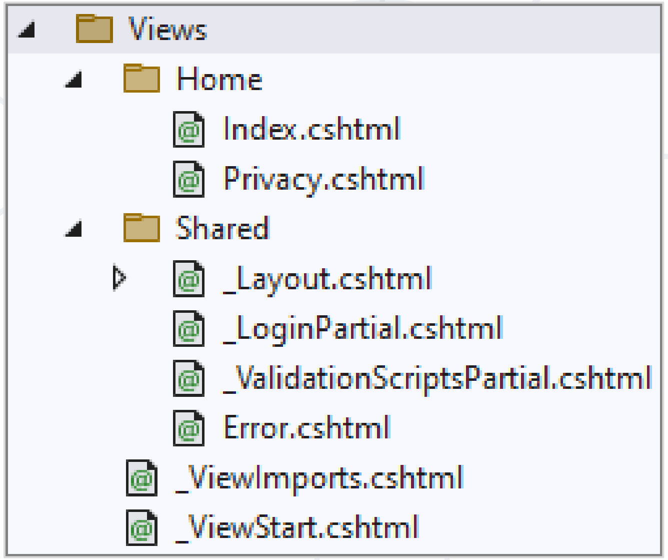
```
private const string AppVersion = "v.1.0.0";

public IActionResult Version()
{
    return Content(AppVersion);
}
```



```
public IActionResult LoginConfirm(string username,
    string password)
{
    return Redirect("/Home/Index");
}
```

ASP.NET Core MVC Views

- **Views** render the **HTML code** for the invoked action
- View naming standard is {**`ActionName`**}.**`cshtml`**
- Views should be placed in folder "**`/Views/{ControllerName}`**"
- A lot of **view engines** available
  - View engines execute code and provide HTML
  - Provide a lot of helpers to easily generate HTML
  - The most popular is **Razor View Engine**



```
Views
  Home
    Index.cshtml
    Privacy.cshtml
  Shared
    _Layout.cshtml
    _LoginPartial.cshtml
    _ValidationScriptsPartial.cshtml
    Error.cshtml
  _ViewImports.cshtml
  _ViewStart.cshtml
```

<%= DateTime.Now %>
@DateTime.Now
NEW RAZOR

ASP.NET Core MVC Models

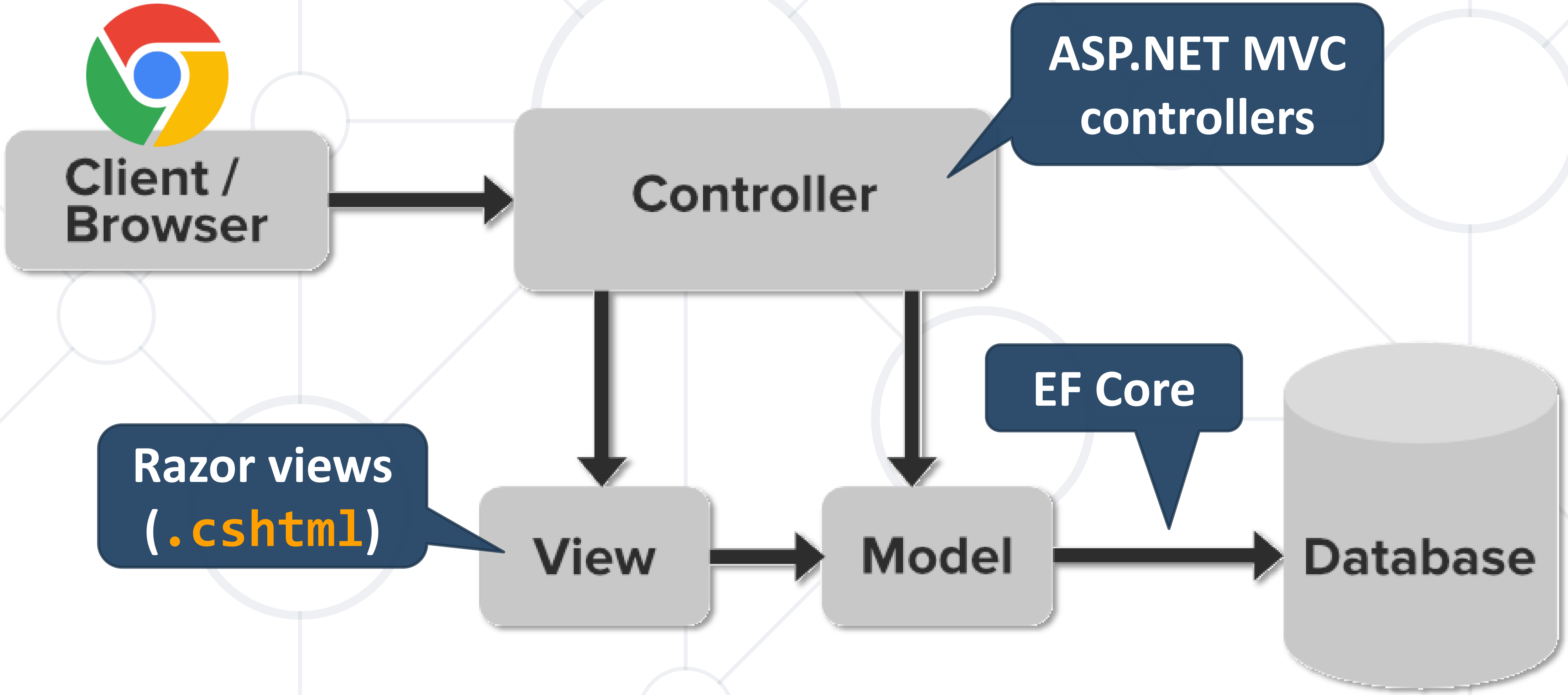
- Structured way to represent **application data and business logic**
- Usually, simple classes with properties
  - Correspond to the data that the application is working with
- Include methods to handle business logic, data validation, etc.
- Support complex data types
- Provide features to bind models to views
  - Easier to display and update data without requiring additional code
    - Streamlining the development process

ASP.NET Core MVC

&

Entity Framework Core

ASP.NET Core MVC + Entity Framework



How to Connect to SQL Server?

- In ASP.NET Core **connection string** is in the **appsettings.json** file and has the following **properties**

```
"ConnectionStrings": {
    "DefaultConnection": "Server=(localdb)\\mssqllocaldb;
         Database=ShoppingList;Trusted_Connection=True;
         MultipleActiveResultSets=true"}
```

How to Connect to SQL Server?

- Use the **DbContext** and tell it to use SQL with the **connection string** in in the **Program** class

```
var connectionString = builder
      .Configuration
      .GetConnectionString("DefaultConnection");

builder
      .Services
      .AddDbContext<ShoppingListDbContext>(
            x => x.UseSqlServer(connectionString));
```

Dependency Injection

Overview

What is Dependency Injection?

- **Dependency injection** injects objects at **runtime**
  - **Register** some service class in the **Program** class

```
services.AddTransient<DataService>();
```

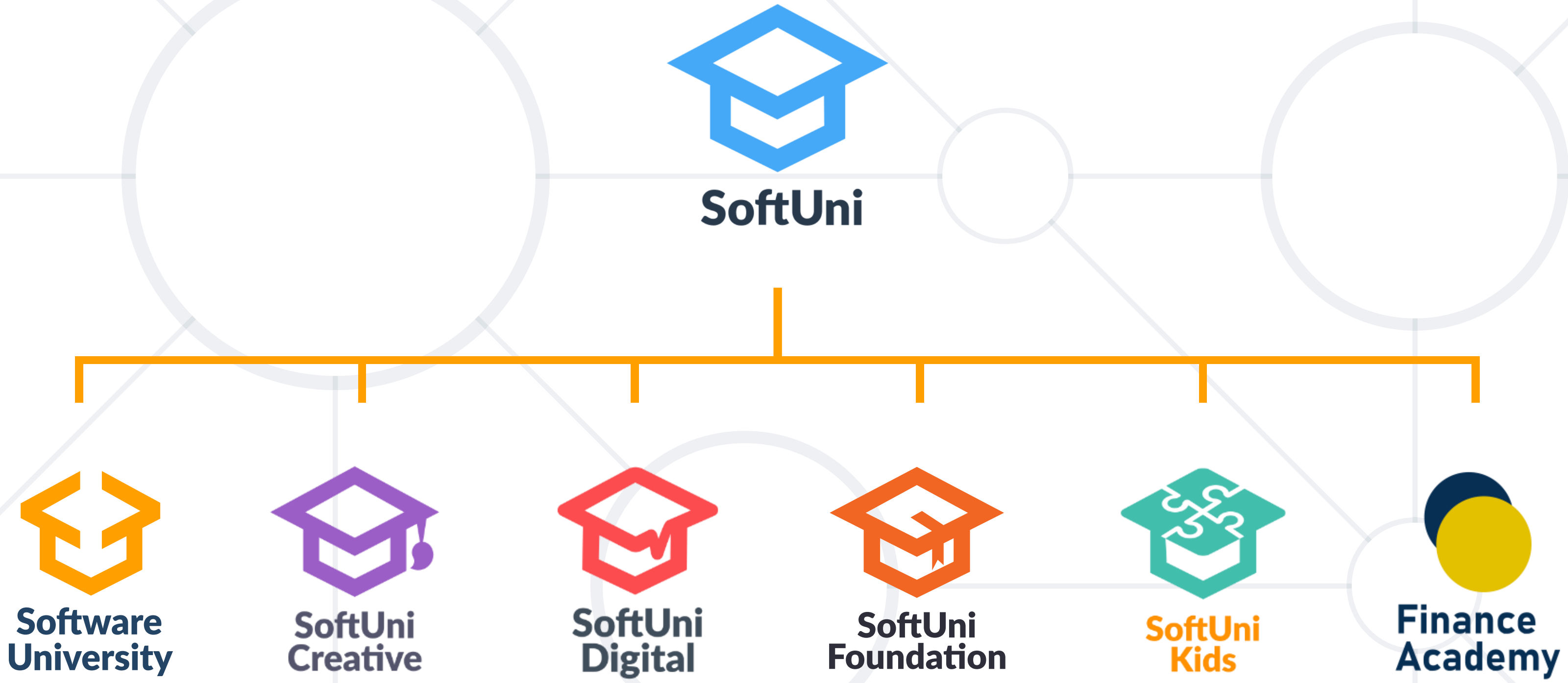
  - Later, **inject** the registered class in your controllers

```
public class ProductController : Controller
{
  public ProductController(DataService ds) {
    // Use the injected object "ds"
  }
}
```

Summary

- **ASP.NET Core** is a great platform for developing Web apps
- The **MVC** pattern is used in ASP.NET Core MVC
- Controllers, Views and Models overview
- How **ASP.NET** Core **MVC** works with **Entity Framework** Core

Questions?


SoftUni
Software University
SoftUni Creative
SoftUni Digital
SoftUni Foundation
SoftUni Kids
Finance Academy

SoftUni Diamond Partners

SmartIT

Coca-Cola HBC Bulgaria

SOFTWARE GROUP

DXC TECHNOLOGY

createX

Postbank

Решения за твоето утре

SUPER HOSTING .BG

DRAFT KINGS

VIVACOM

INDEAVR

Serving the high achievers

AMBITIONED

PHAR VISION

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg