Session 4

Client-side Development Using ASP.NET Core MVC

Session Overview

- Describe layouts in ASP.NET Core MVC
- Explain implementing styles in ASP.NET Core MVC applications
- Explain Data Annotations
- Describe routing
- Explain dependency injection
- Identify the process to create a Single Page Application
- Describe client side validation and server side validation

Layout View in ASP.NET Core MVC

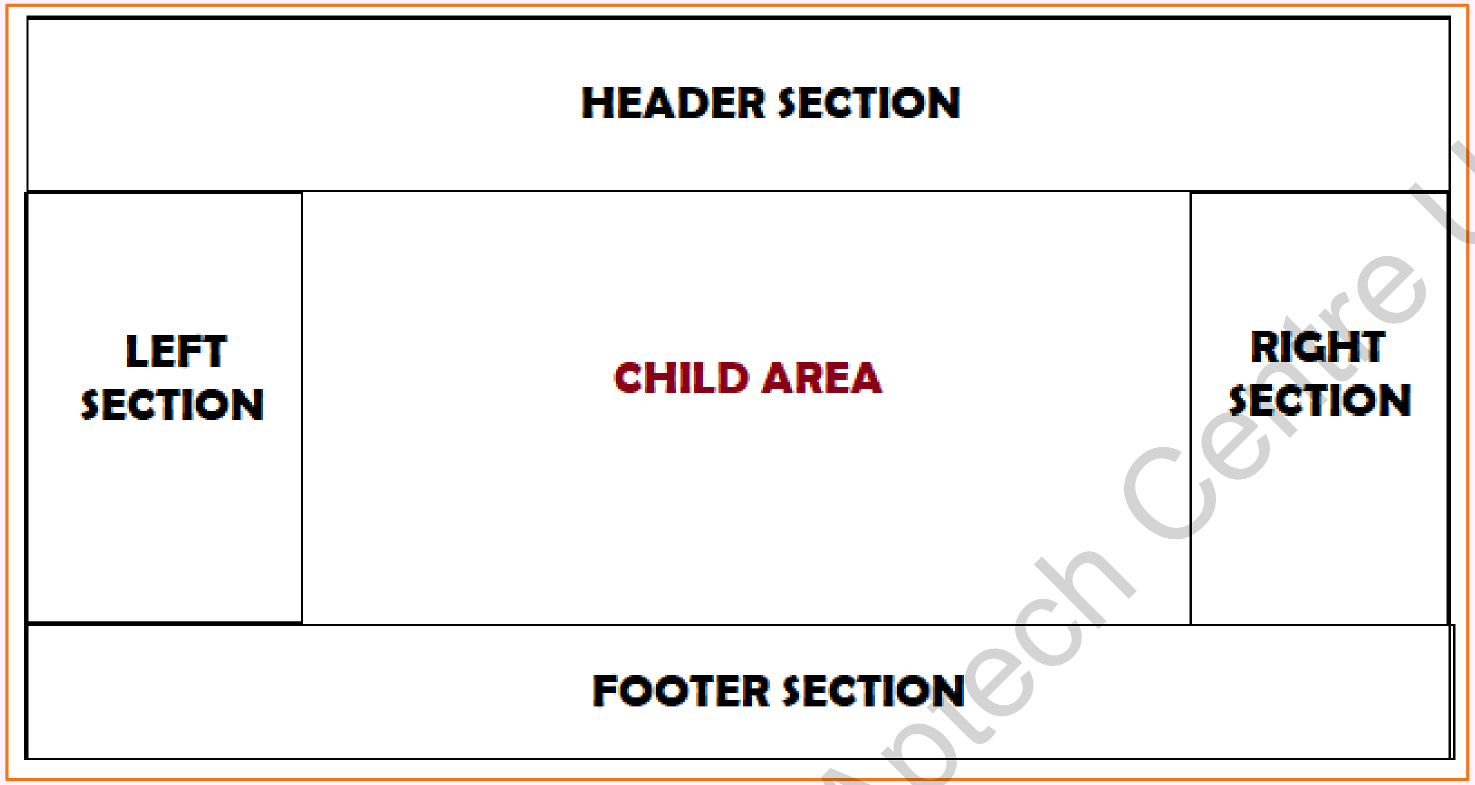
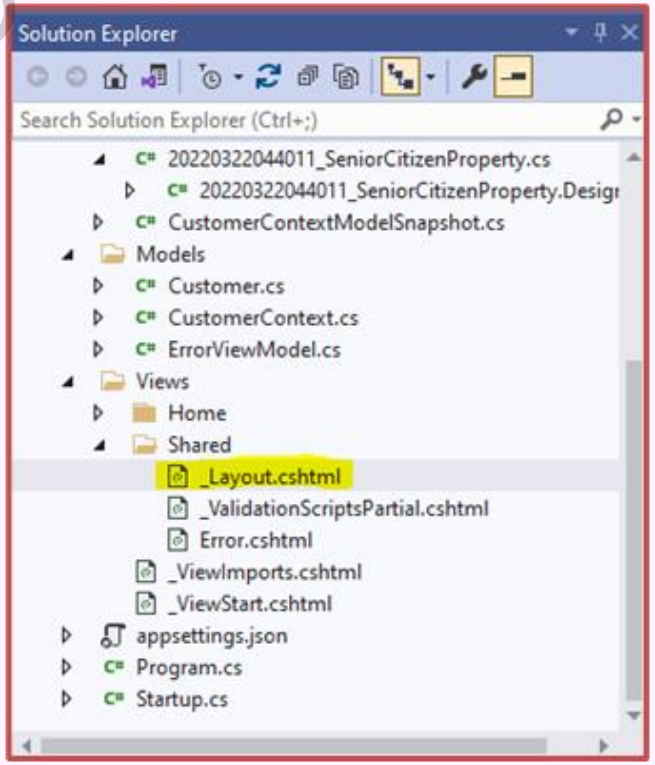


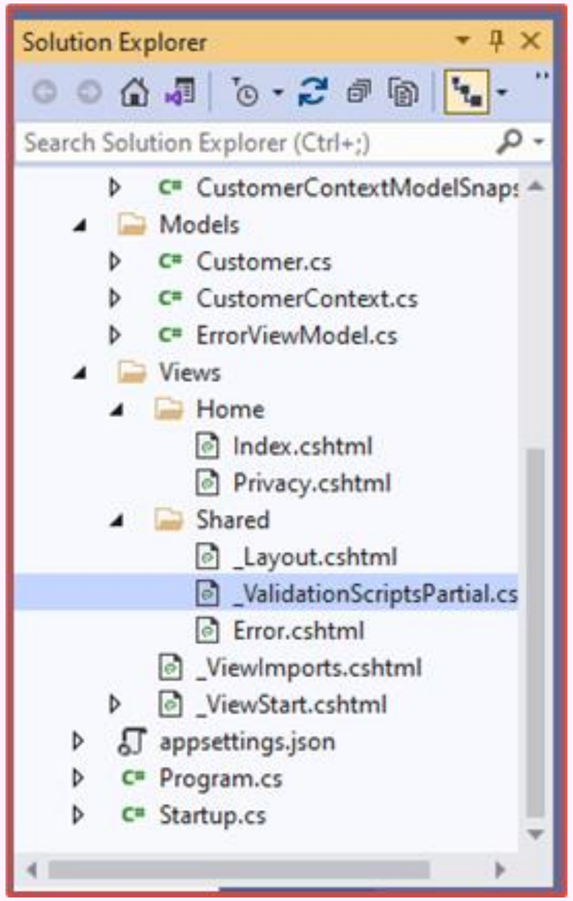
Figure 4.1: UI Sections



```
20220322044011_SeniorCitizenProperty.cs
  20220322044011_SeniorCitizenProperty.Desigr
CustomerContextModelSnapshot.cs
Models
  Customer.cs
  CustomerContext.cs
  ErrorViewModel.cs
Views
  Home
  Shared
    _Layout.cshtml
    _ValidationScriptsPartial.cshtml
    Error.cshtml
  _ViewImports.cshtml
  _ViewStart.cshtml
appsettings.json
Program.cs
Startup.cs
```

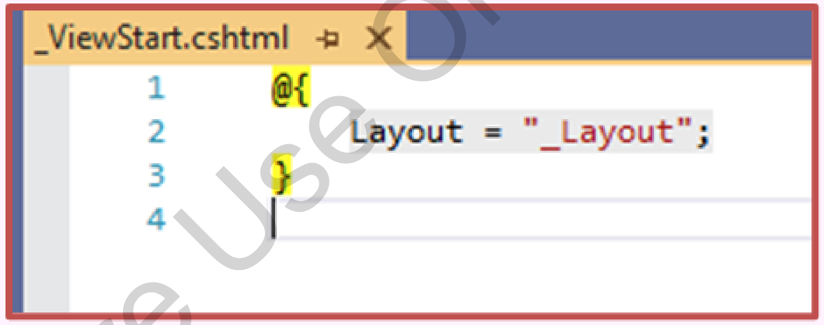
Figure 4.2: _Layout_cshtml File

Using Layout View (1-2)



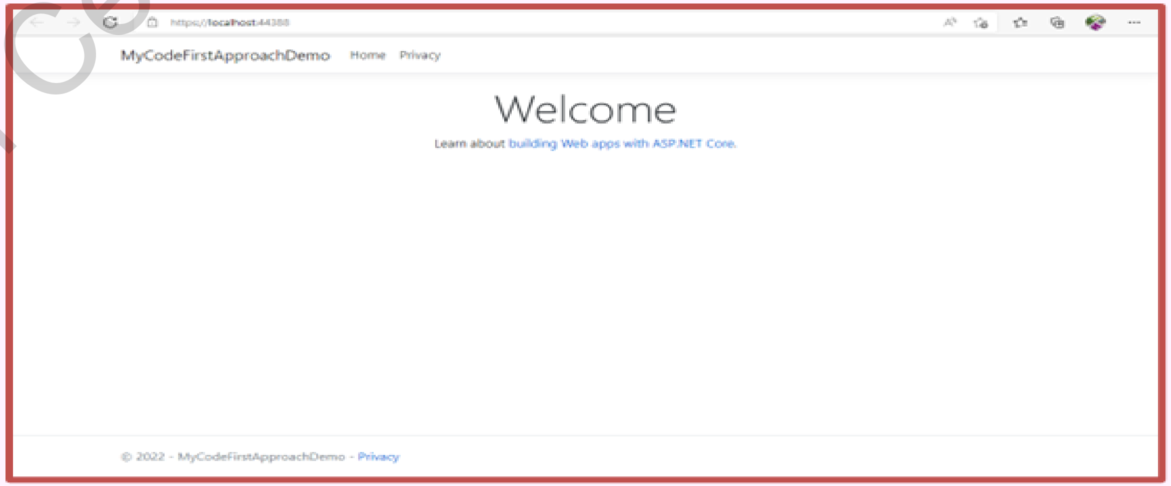
```
    CustomerContextModelSnaps
  Models
    Customer.cs
    CustomerContext.cs
    ErrorViewModel.cs
  Views
    Home
      Index.cshtml
      Privacy.cshtml
    Shared
      _Layout.cshtml
      _ValidationScriptsPartial.cs
      Error.cshtml
    _ViewImports.cshtml
    _ViewStart.cshtml
  appsettings.json
  Program.cs
  Startup.cs
```

Figure 4.3: Views Folder



```
@{
    Layout = "_Layout";
}
```

Figure 4.4: Default Layout



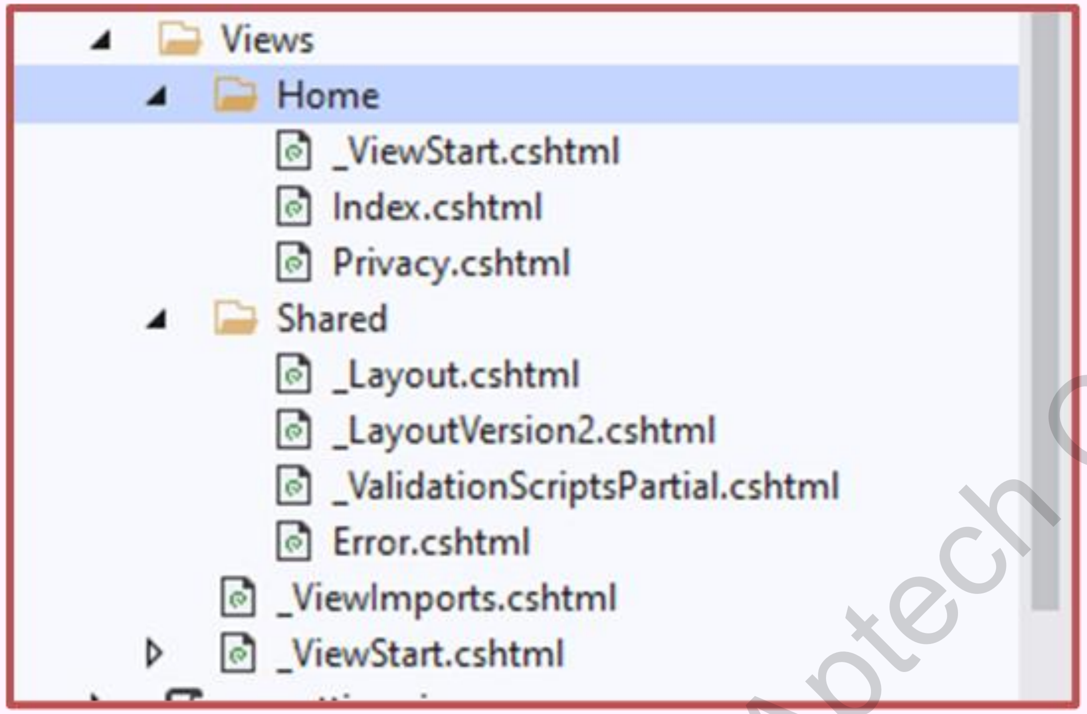
Welcome

Learn about building Web apps with ASP.NET Core.

© 2022 - MyCodeFirstApproachDemo - Privacy

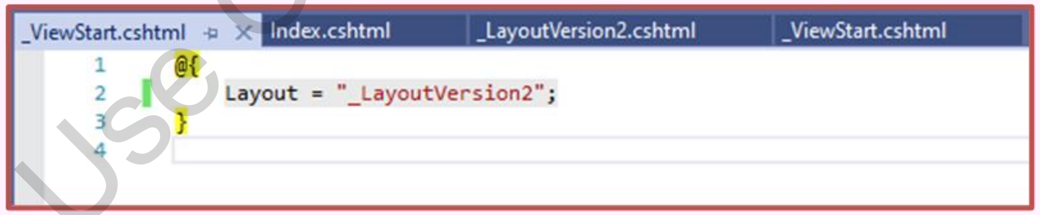
Figure 4.5 Layout Output

Using Layout View(2-2)



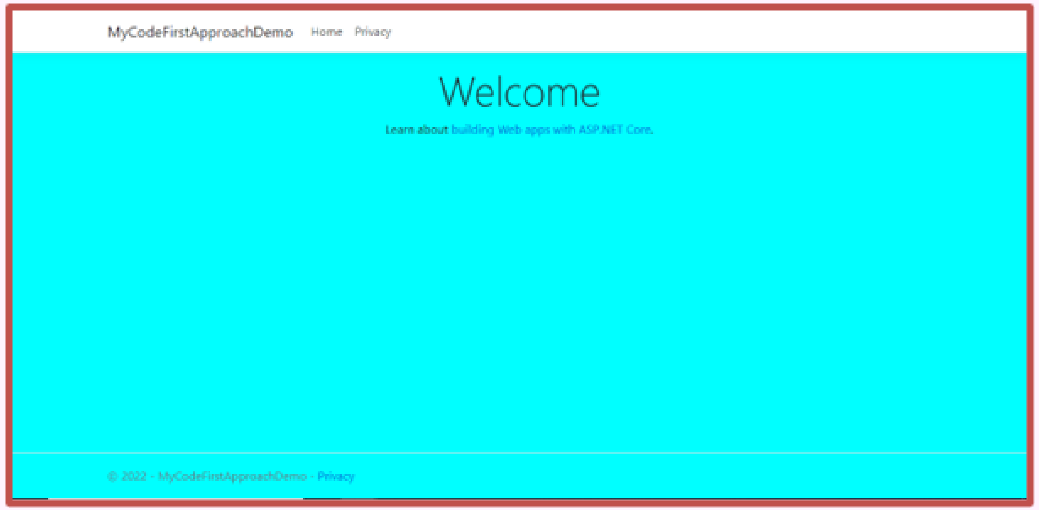
```
◢ Views
   ◢ Home
        _ViewStart.cshtml
        Index.cshtml
        Privacy.cshtml
   ◢ Shared
        _Layout.cshtml
        _LayoutVersion2.cshtml
        _ValidationScriptsPartial.cshtml
        Error.cshtml
     _ViewImports.cshtml
   ▷ _ViewStart.cshtml
```

Figure 4.6: Subfolders of Views



```
@{
    Layout = "_LayoutVersion2";
}
```

Figure 4.7: _ViewStart.html Layout



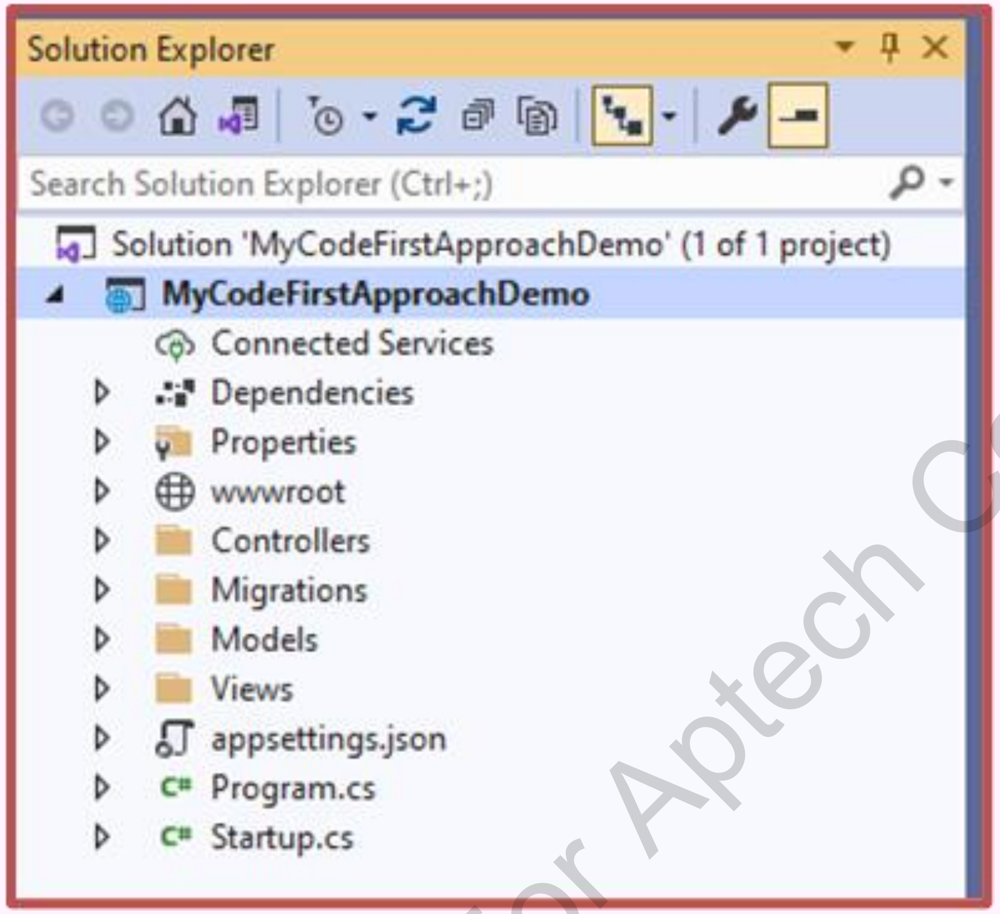
Welcome

Learn about building Web apps with ASP.NET Core.

© 2022 - MyCodeFirstApproachDemo - Privacy

Figure 4.8: Layout with Changed Background Color

Implementing Styles (1-2)


Solution Explorer

Search Solution Explorer (Ctrl+;)

Solution 'MyCodeFirstApproachDemo' (1 of 1 project)
- MyCodeFirstApproachDemo
  - Connected Services
  - Dependencies
  - Properties
  - wwwroot
  - Controllers
  - Migrations
  - Models
  - Views
  - appsettings.json
  - Program.cs
  - Startup.cs

Figure 4.9: Solution Explorer

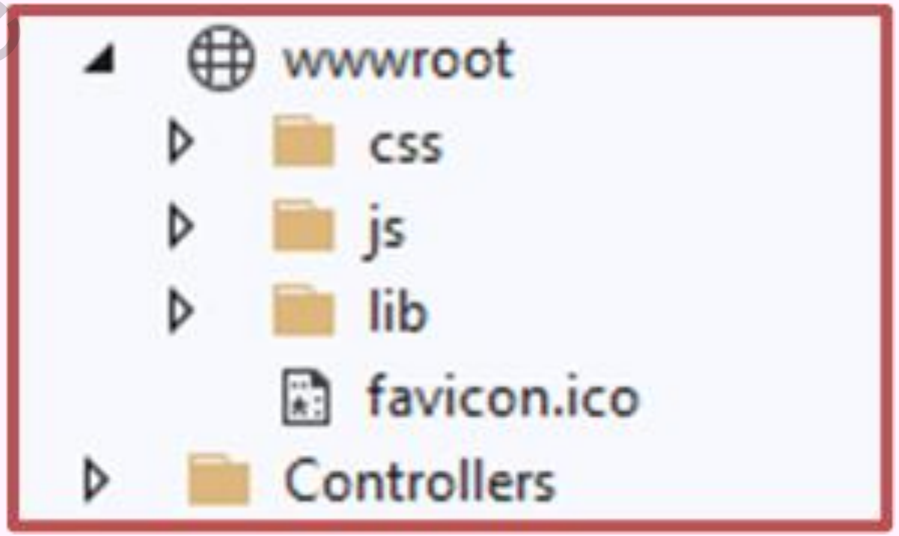
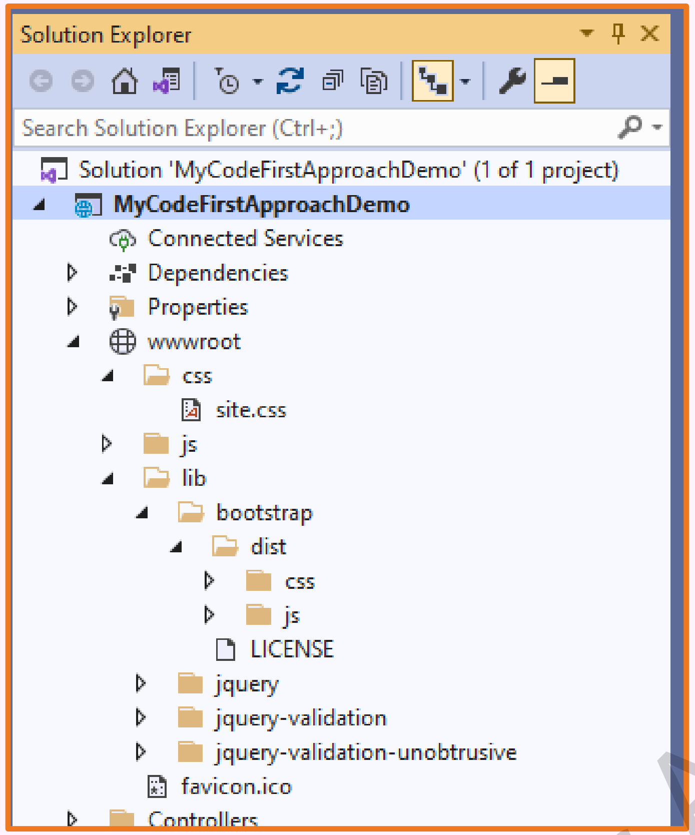

- wwwroot
  - css
  - js
  - lib
  - favicon.ico
- Controllers

Figure 4.10: wwwroot Folder

Implementing Styles (2-2)

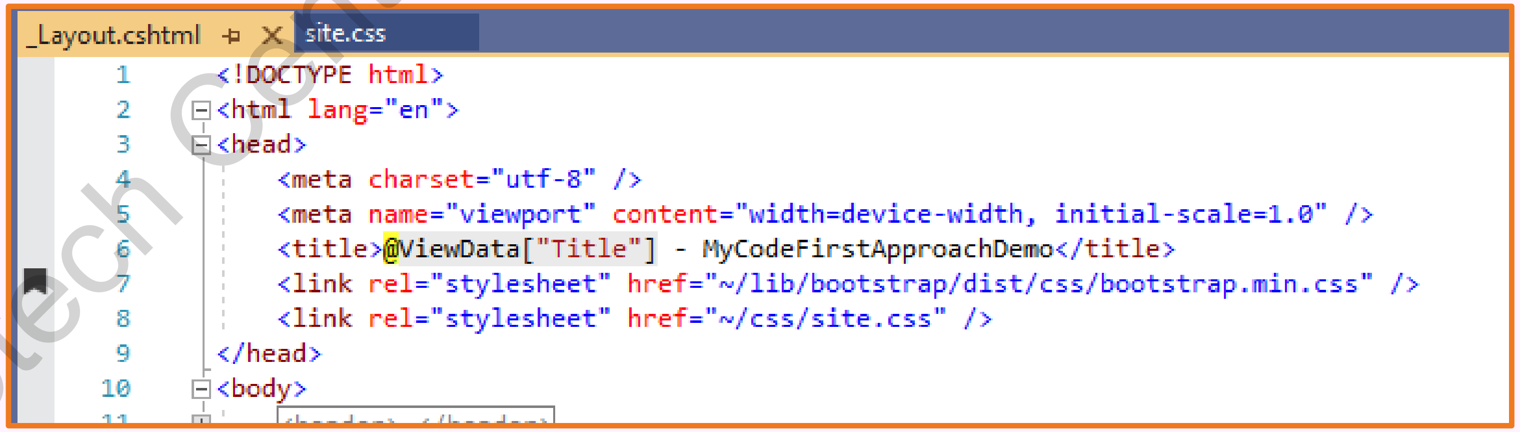


Solution Explorer

Search Solution Explorer (Ctrl+;)

```
Solution 'MyCodeFirstApproachDemo' (1 of 1 project)
  MyCodeFirstApproachDemo
    Connected Services
    Dependencies
    Properties
    wwwroot
      css
        site.css
      js
      lib
        bootstrap
          dist
            css
            js
            LICENSE
        jquery
        jquery-validation
        jquery-validation-unobtrusive
      favicon.ico
    Controllers
```

Figure 4.11: Bootstrap Folder



_Layout.cshtml   site.css

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - MyCodeFirstApproachDemo</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" />
</head>
<body>
```

Figure 4.12: site.css

Data Annotations

RequiredAttribute	This attribute indicates that a field value is to be entered by a user.
RangeAttribute	This attribute indicates the numeric range in which data should fall.
PhoneAttribute	It indicates that the value in the field must be in a specified phone number format.
UrlAttribute	It allows the developer the ability to enforce specified url types. We can also make it such that users can only enter https URLs.
EmailAddressAttribute	This attribute is widely used to limit users to specify email addresses.

Routing

1

- It bridges the gap between incoming HTTP requests and forwarding them to the executable endpoints allowing the required action to be completed.

2

- It can also produce URLs that map to the endpoints using endpoint information from the app.

3

- It is a strategy that tracks requests and then, maps them to controllers and their action methods.

4

- For a single application, numerous routes can be configured.

Dependency Injection

Dependency Injection (DI)

Inversion of Control (IoC)

Dependency Inversion Principle (DIP)

Both high-level and low-level modules should depend on abstractions.

Details should be dependent on abstractions and not vice versa. When using this principle, ensure to make use of the interfaces.

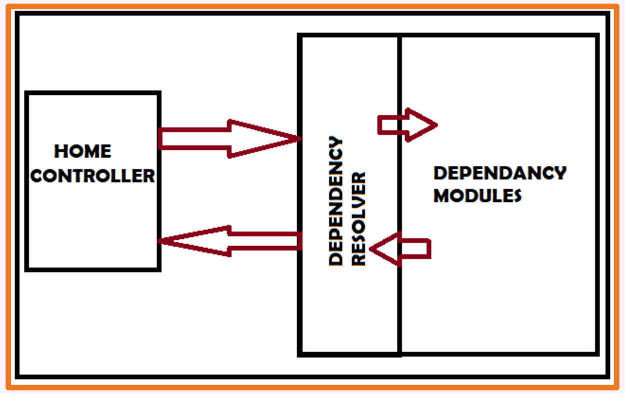


Figure 4.13: Dependency Resolver

Creating Single Page Applications (1-4)

Throughout the lifespan of an application, most resources are loaded only once. It is only the data that is transmitted back and forth. This makes SPA faster.

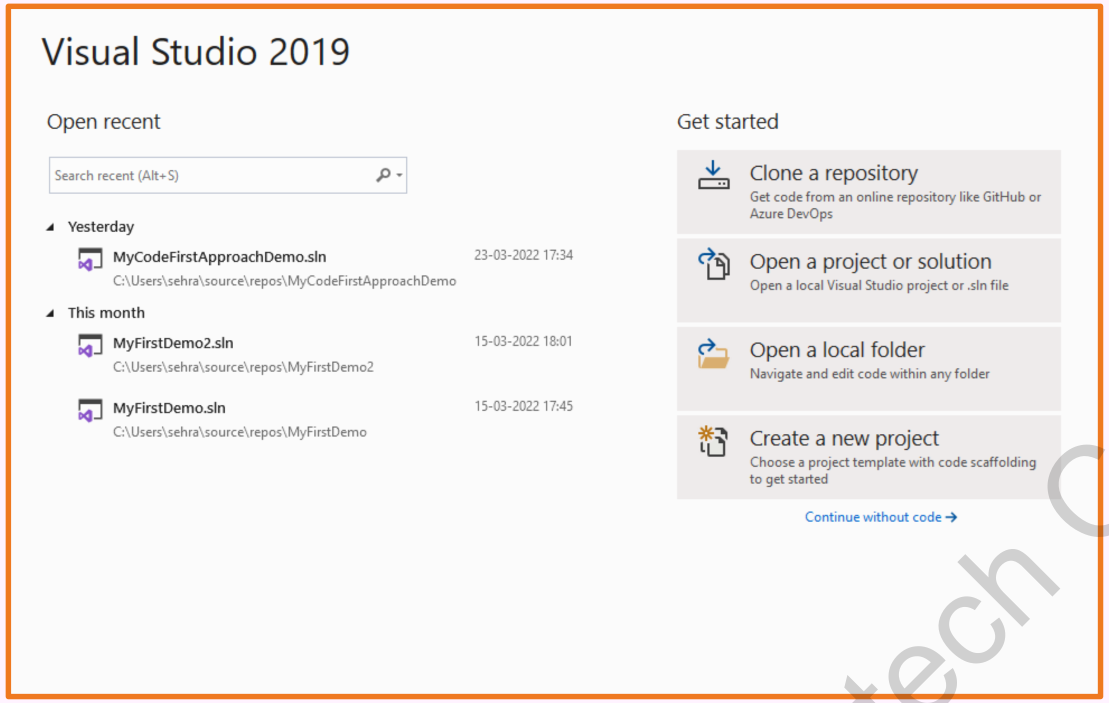
It offers simplified and streamlined development.

Monitoring network operations, investigating page elements, and data associated with it makes SPAs easy to debug with Chrome.

The same backend code can be used for Web application and native mobile application.

Local storage can be cached effectively.

Creating Single Page Applications (2-4)


Visual Studio 2019

Open recent

Search recent (Alt+S)

Yesterday

MyCodeFirstApproachDemo.sln 23-03-2022 17:34
C:\Users\sehra\source\repos\MyCodeFirstApproachDemo

This month

MyFirstDemo2.sln 15-03-2022 18:01
C:\Users\sehra\source\repos\MyFirstDemo2

MyFirstDemo.sln 15-03-2022 17:45
C:\Users\sehra\source\repos\MyFirstDemo

Get started

Clone a repository
Get code from an online repository like GitHub or Azure DevOps

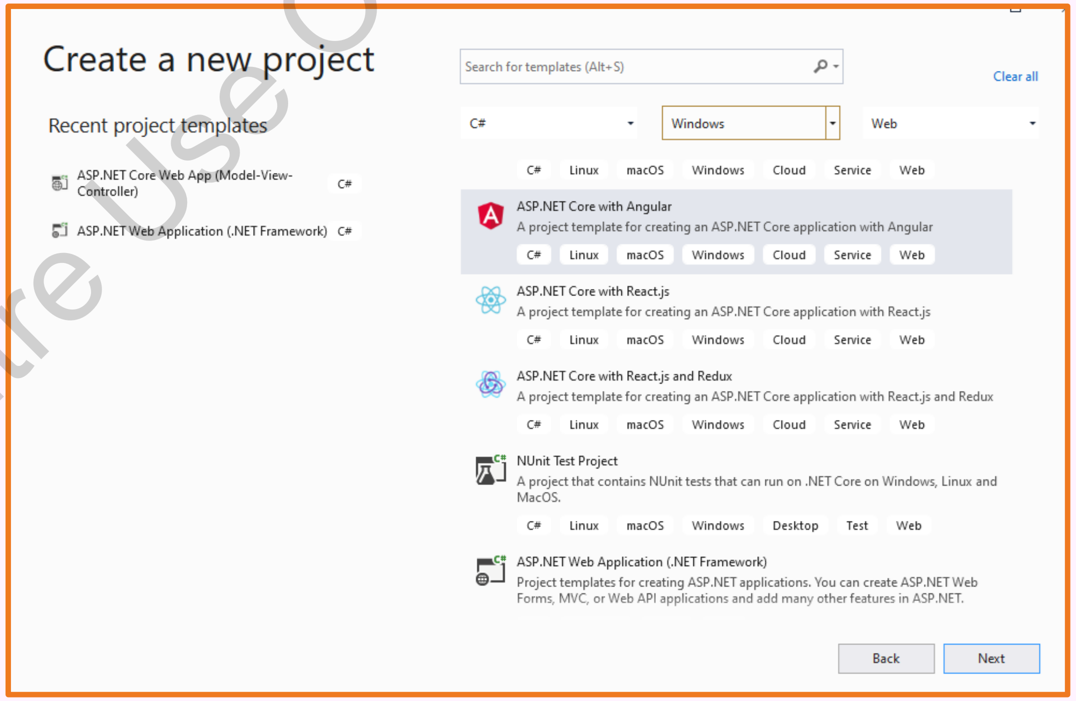
Open a project or solution
Open a local Visual Studio project or .sln file

Open a local folder
Navigate and edit code within any folder

Create a new project
Choose a project template with code scaffolding to get started

Continue without code →

Figure 4.14: Create a New Project


Create a new project

Search for templates (Alt+S) Clear all

C# | Windows | Web

Recent project templates

ASP.NET Core Web App (Model-View-Controller) C#

ASP.NET Web Application (.NET Framework) C#

C# Linux macOS Windows Cloud Service Web

ASP.NET Core with Angular
A project template for creating an ASP.NET Core application with Angular
C# Linux macOS Windows Cloud Service Web

ASP.NET Core with React.js
A project template for creating an ASP.NET Core application with React.js
C# Linux macOS Windows Cloud Service Web

ASP.NET Core with React.js and Redux
A project template for creating an ASP.NET Core application with React.js and Redux
C# Linux macOS Windows Cloud Service Web

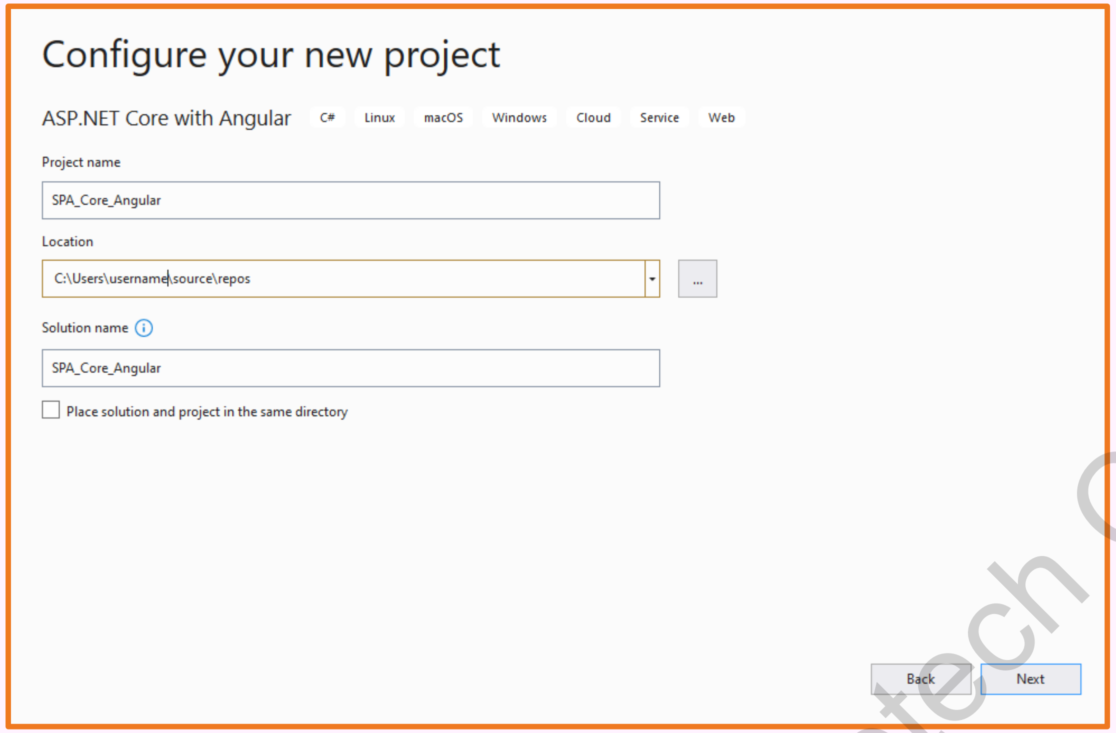
NUnit Test Project
A project that contains NUnit tests that can run on .NET Core on Windows, Linux and MacOS.
C# Linux macOS Windows Desktop Test Web

ASP.NET Web Application (.NET Framework)
Project templates for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Back Next

Figure 4.15: ASP.NET Core with Angular Option

Creating Single Page Applications (3-4)



Configure your new project

ASP.NET Core with Angular C# Linux macOS Windows Cloud Service Web

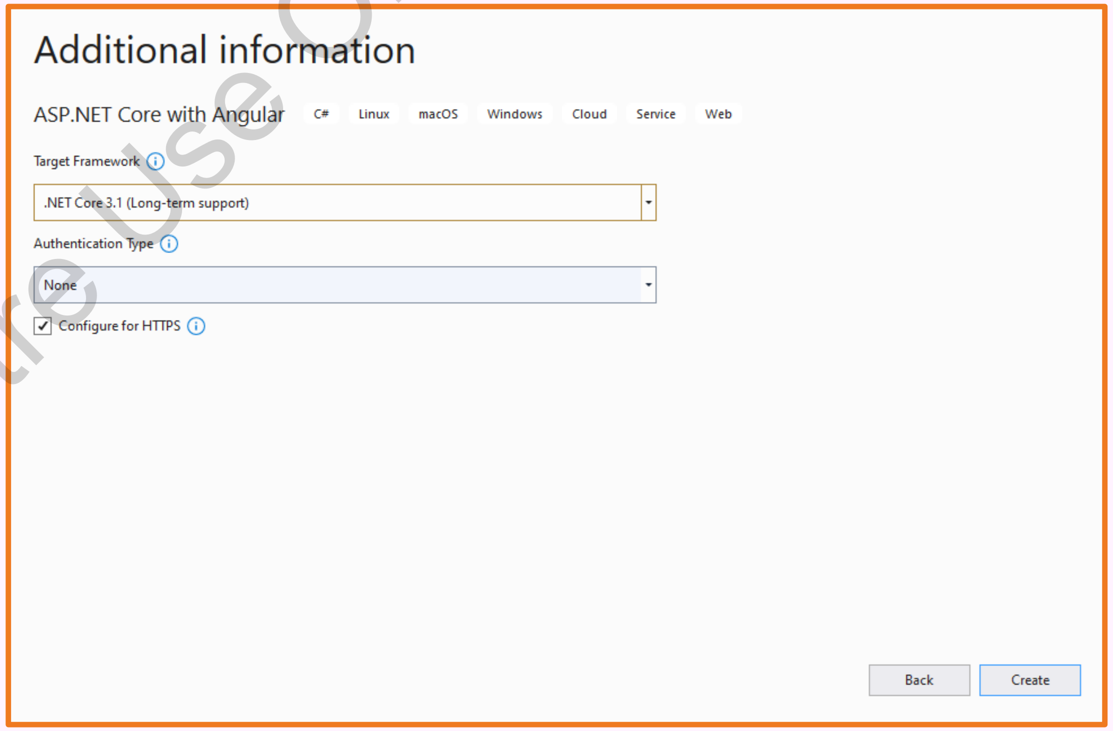
Project name: SPA_Core_Angular

Location: C:\Users\username\source\repos

Solution name: SPA_Core_Angular

☐ Place solution and project in the same directory

Figure 4.16: Name and Location of Project



Additional information

ASP.NET Core with Angular C# Linux macOS Windows Cloud Service Web

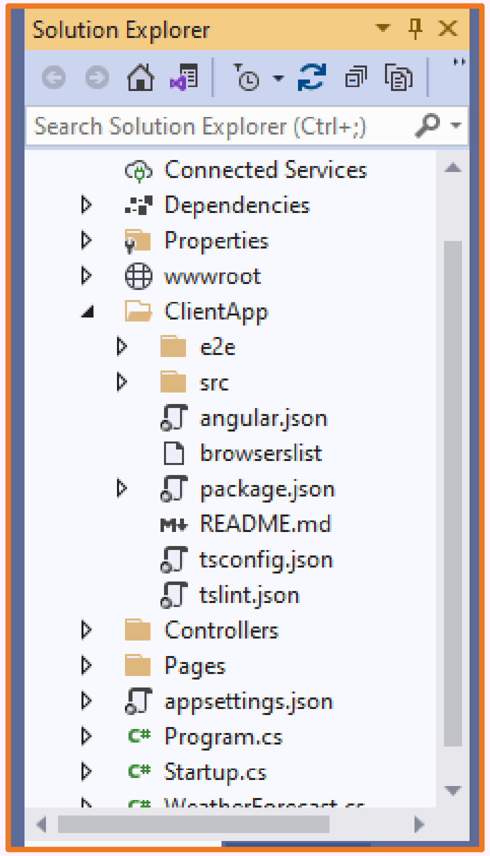
Target Framework: .NET Core 3.1 (Long-term support)

Authentication Type: None

☑ Configure for HTTPS

Figure 4.17: Additional Information

Creating Single Page Applications (4-4)

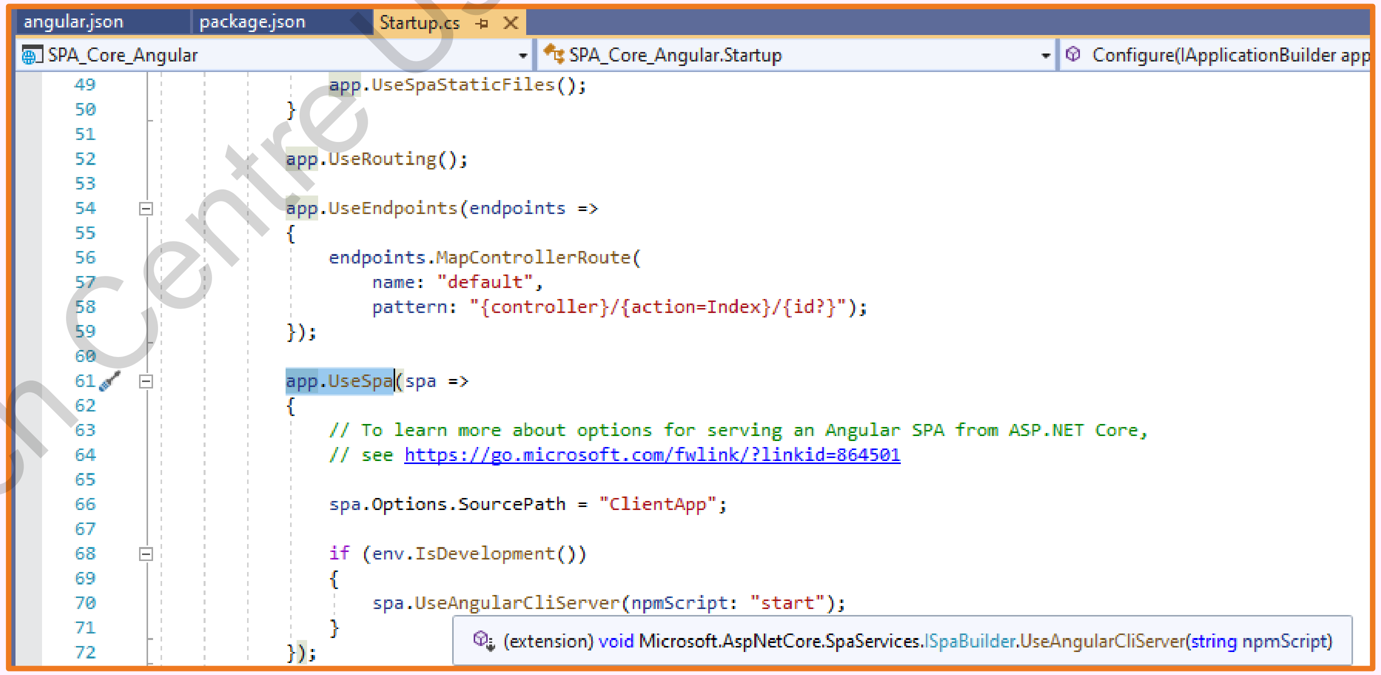


Solution Explorer

Search Solution Explorer (Ctrl+;)

```
   Connected Services
 ▷ Dependencies
 ▷ Properties
 ▷ wwwroot
 ◢ ClientApp
    ▷ e2e
    ▷ src
      angular.json
      browserslist
    ▷ package.json
      README.md
      tsconfig.json
      tslint.json
 ▷ Controllers
 ▷ Pages
 ▷ appsettings.json
 ▷ Program.cs
 ▷ Startup.cs
 ▷ WeatherForecast.cs
```

Figure 4.18: Solution Explorer



angular.json | package.json | Startup.cs

SPA_Core_Angular | SPA_Core_Angular.Startup | Configure(IApplicationBuilder app

```
        app.UseSpaStaticFiles();
    }

    app.UseRouting();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllerRoute(
            name: "default",
            pattern: "{controller}/{action=Index}/{id?}");
    });

    app.UseSpa(spa =>
    {
        // To learn more about options for serving an Angular SPA from ASP.NET Core,
        // see https://go.microsoft.com/fwlink/?linkid=864501

        spa.Options.SourcePath = "ClientApp";

        if (env.IsDevelopment())
        {
            spa.UseAngularCliServer(npmScript: "start");
        }
    });
```

(extension) void Microsoft.AspNetCore.SpaServices.ISpaBuilder.UseAngularCliServer(string npmScript)

Figure 4.19: Startup.cs

Client-Side and Server-Side Validations (1-4)

Changes in the form: To access the HTML element to display the HTML error message, add the id attribute to all the span tags. A JavaScript function is called to validate the input data when the form is submitted.

The script HTML element is included and a JavaScript function is created to validate the input data.

Client-side validation provides better user experiences.

Client-side validation is performed by Script languages, such as JavaScript, VBScript

JavaScript is a high-level, interpreted language. If the user turns it OFF, dangerous input can be bypassed and submitted to the server.

Client-Side and Server-Side Validations (2-4)

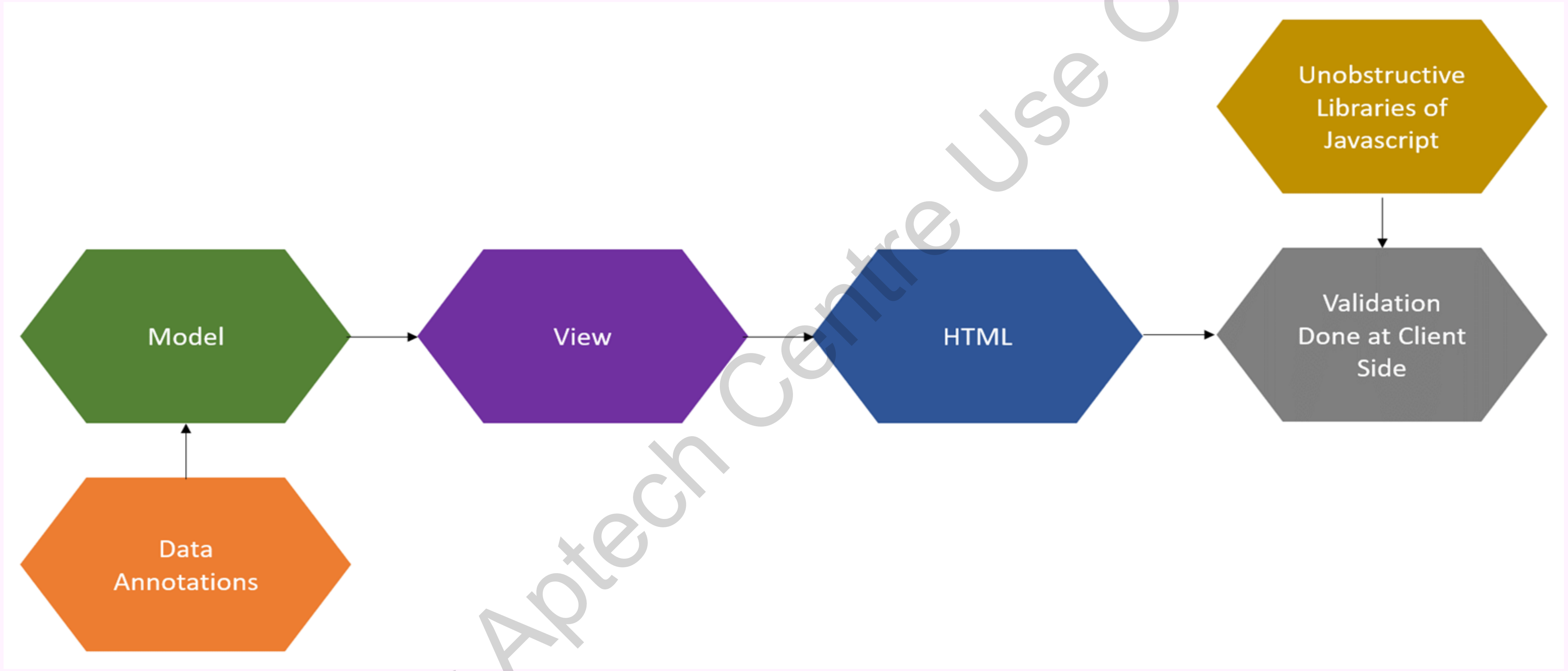
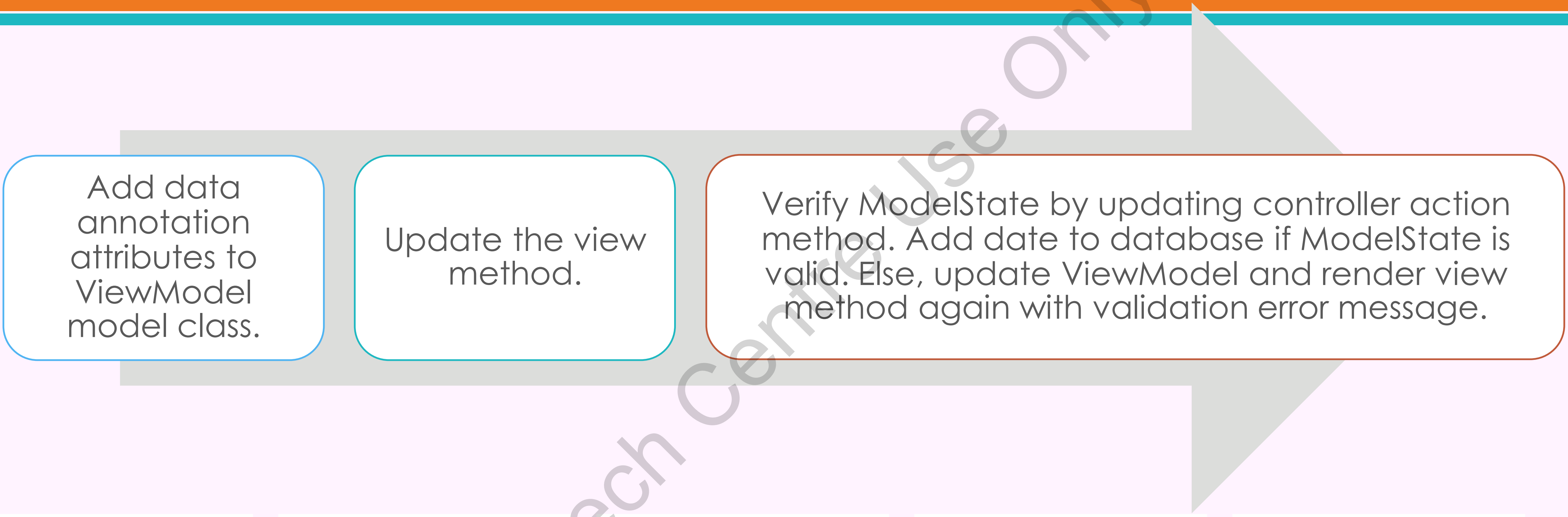

Unobstructive Libraries of Javascript
Model
View
HTML
Validation Done at Client Side
Data Annotations

Figure 4.21: Client-side Validation

Client-Side and Server-Side Validations (3-4)

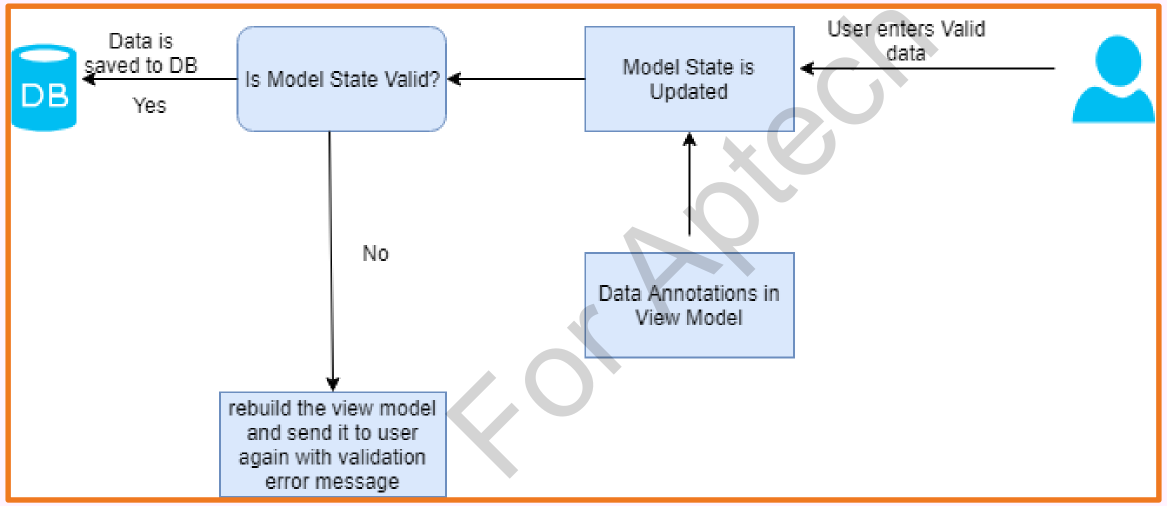

Add data annotation attributes to ViewModel model class.
Update the view method.
Verify ModelState by updating controller action method. Add date to database if ModelState is valid. Else, update ViewModel and render view method again with validation error message.

Required | **Range** | **MinLength** | **MaxLength** | **RegularExpression**

Client-Side and Server-Side Validations (4-4)

High-level sequence of events in the server-side validation:

1. Enters the invalid data
2. Updates the ModelState based on the data annotation attribute in the View model
3. Verifies the ModelState in the controller's action method
4. Saves the entered data to the database when the ModelState is valid
5. Renders View model again with validation error message when the ModelState is not valid



User enters Valid data
Model State is Updated
Data Annotations in View Model
Is Model State Valid?
Yes
Data is saved to DB
DB
No
rebuild the view model and send it to user again with validation error message

Figure 4.22: Server-side Validation

Summary

- In Web applications, a specific part of the user interface is the same across all pages.
- The common sections are the header section, footer section, and left and right navigation.
- Data validation is an important part of development, especially when dealing with big data.
- Routing in ASP.Net MVC refers to bridging the gap between incoming HTTP requests and forwarding them to the app's executable endpoints.
- ASP.NET MVC has a built-in namespace System.ComponentModel.DataAnnotations that has classes for data validation.
- In server-side validation, the server validates the input submitted by the user. Post validation, a dynamically generated new Web page sends the feedback back to the client.
- In client-side validation, all user inputs are validated in the user's browser itself.
- A design pattern where the dependencies of one object is provided by another object is DI.
- A Web application that fits on a single Web page is a Single Page Application (SPA).