**MVC:** A popular software design pattern used for developing web applications and user interfaces.

**ASP.NET** is seen as a Web Application Framework within the broader .NET Architecture Ecosystem.

**MVC Architecture**

1. **Model**
   * Handles **data & business logic**.
   * Interacts with the **database**.
   * Example:

public class Person {

public int Id { get; set; }

public string Name { get; set; }

}

1. **View**
   * Defines the **UI using Razor syntax**.
   * Displays data from Model.
   * Sends user input to the Controller for processing.
   * Receives updates from the Controller
   * Example:

A screenshot of a computer screen

AI-generated content may be incorrect.@model List<Person>

@foreach(var person in Model) {

<p>@person.Name</p>

}

1. **Controller**
   * Handles **HTTP requests**.
   * Calls Model and passes data to View.
   * User actions trigger the Controller, which updates the Model if necessary, and the Model then usually notifies the View that a change has occurred
   * Example:

public class HomeController : Controller {

public ActionResult Index() {

var people = new List<Person>{ new Person { Name = "John" } };

return View(people);

}

}

**Structure of a MVC project:**

• **App\_Data:** Data files, such as database files or XML. • **App\_Start:** Classes that are executed on starts up. • **Content:** Static resources like CSS and images. • **Controllers:** Controller classes (HTTP requests, process input, output). • **fonts:** Font files (special or default). • **Models:** Classes that represent the data and business logic. • **Scripts:** JavaScript files. • **Views:** View templates that display the user interface. • **bin:** Contains compiled code and assemblies (hidden). • **obj:** Temporary files generated during build (hidden). • **Properties**: Application settings and metadata. • **packages:** Third-party libraries and dependencies (separate folder).

**A class** is used to create the model. Classes can be used in object-oriented programming, connecting and interfacing with data from a database and combining data from multiple sources. It functions asthe data manager. **The model** (class) contains a simple pre-fabricated pattern that needs to be updated based on the type of data that will be processed. We start with the expected, and enhance it (add new properties) as we progress. Think of an **ActionResult** as a click event procedure. The procedure **receives data from the view, processes data and the returns the result back to the view.**

This is the command where the page is rendered. The page is inserted into the container called “Render Section” This is also where globalscripts as stipulated.

**\_ViewStart.cshtml** — It is used to specify common settings for all the views under a folder and sub-folders where it is created.

The views which will be displayed in a placeholder RenderBody() are called child views.

**The CSS Box Model** is a fundamental concept that defines how elements are structured and displayed on a web page. Every element is treated as a rectangular box consisting of multiple layers that affect its overall size and spacing. Each HTML element is wrapped by a box. Everything in CSS has a box around it, and understanding these boxes is key to being able to create layouts with CSS, or to align items with other items. • Content - The content of the box, where text and images appear. • Padding - Clears an area around the content. The padding is transparent • Border - A border that goes around the padding and content • Margin - Clears an area outside the border. The margin is transparent. Default is 0. It is important to note that margin is the external space separating boxes.

**Total Width** = width + left padding + right padding + left border + right border + left margin + right margin

**Total Height** = height + top padding + bottom padding + top border + bottom border + top margin + bottom margin

**Box-Sizing Property:** **content-box (default):** The width and height only include the content. Padding and border are added *outside* the specified width and height, increasing the total element size. **padding-box:** The width and height include padding but *not* the border. **border-box:** The width and height include content, padding, and border. This makes layout calculations more predictable and is commonly used (\* { box-sizing: border-box; } ensures all elements follow this rule).

**Why border-box is useful:**

• It prevents the total size of an element from exceeding its specified dimensions. • It ensures a more intuitive design, especially for grid-based layouts. • By default, browsers use border-box for <table>, <select>, <button>, and <input> elements of type radio, checkbox, reset, button, submit, color, and search.

**Handling Overflow**

**visible (default):** Content overflows without being clipped. **hidden:** Extra content is clipped and not visible. **scroll: A** scrollbar appears even if the content fits within the element. **auto:** A scrollbar appears only when the content overflows.

**CSS Display Property**

The display property defines how elements are rendered in the layout.

**Inline Elements (display: inline):** Do not break onto a new line. • Only take up as much width as necessary. • Width and height properties are ignored. • Examples: <a>, <span>, <strong>, <em>.

**Inline Replaced Elements:** • Have an intrinsic width and height set outside CSS. • Content is usually external (e.g., images, inputs). • Examples: <img>, <input>.

**Block Elements (display: block):** • Always start on a new line. • Take up the full width of the parent container. • Width, height, padding, and margins are respected. • Examples: <p>, <div>, <h1>, <form>, <table>.

**Inline-Block Elements (display: inline-block)** • Behaves like inline, but allows setting width and height. • Top and bottom margins/padding are respected. • Elements appear next to each other without line breaks.

**CSS Position Property:** Determines how an element is positioned relative to the document flow. Possible values:

**Static (default):** • Elements are positioned according to the normal document flow. • top, left, right, bottom values have no effect.

**Relative:** • Elements remain in the normal document flow. • Can be offset using top, left, right, bottom (relative to its original position).

**Absolute:** • The element is removed from the normal document flow. • Positioned relative to the closest positioned ancestor (not static). • If no ancestor is positioned, it is relative to <html>.

**Fixed:** • Similar to absolute, but always positioned relative to <html>.• The element does not move when scrolling.

**Advanced CSS Positioning Concepts**

**1. Z-Index:** • z-index controls the **stacking order** of elements. • Higher values place elements **in front**, while lower values place elements **behind** others. • Only works on elements with position: relative, absolute, or fixed (not static).

Example:

.box1 {

position: absolute;

z-index: 2; /\* Will appear in front \*/ }

.box2 {

position: absolute;

z-index: 1; /\* Will appear behind \*/}

* A z-index of -1 places the element behind other elements.

**2. Float Property:** The float property is used to position elements within their container, often for wrapping text around images.

**Common Values:**• left: Element floats to the left of its container. • right: Element floats to the right of its container. • none: Default; element remains in normal flow. • inherit: Inherits the float value from its parent.

Example:

img {

float: left;

margin-right: 10px;}

This will make text wrap around an image positioned on the left.

**Clearing Floats**

When using float, other elements may be affected. Use clear to prevent layout issues:

.clearfix::after {

content: "";

display: block;

clear: both;}

Apply .clearfix to the parent container to fix float-related issues.

**3. Pseudo-Classes:** Pseudo-classes apply styles based on an element’s state.

**Common Pseudo-Classes:**

**User Interaction:**

:hover → Styles an element when hovered.

:focus → Styles an element when it receives focus.

:active → Styles an element when clicked.

Example:

button:hover {

background-color: blue;}

input:focus {

border: 2px solid red;}

**Link Styling:**

:link → Styles unvisited links.

:visited → Styles visited links.

Example:

a:link {

color: blue;}

a:visited {

color: purple;}

**Structural Pseudo-Classes:**

:first-child → Styles the first child of a parent.

:last-child → Styles the last child of a parent.

:nth-child(n) → Styles the nth child.

Example:

p:first-child {

color: red;}

p:nth-child(2) {

font-weight: bold;}

There are many more pseudo-classes, such as :not(), :checked, and :nth-of-type().

**Cascading Style Sheets (CSS)** is a style sheet language used to control the presentation of web documents written in HTML. It is one of the core technologies of the World Wide Web, alongside HTML and JavaScript. CSS is designed to separate content from design, enabling more flexible and maintainable web development.  
• Internal CSS\*\* - Internal styles are applied within a `<style>` tag inside an HTML document. <style> body {padding-top: 70px;padding-bottom: 20px;}</style>

• External CSS\*\*- External styles are stored in a separate `.css` file and linked to the HTML document using a `<link>` tag.

<link rel="stylesheet"href="/css/main.css">

•Inline CSS\*\*- Inline styles are applied directly to an individual HTML element using the `style` attribute.

<p style="height:182px !important;">This is a paragraph.</p>

**Responsive Web Design (RWD):** Responsive Web Design ensures that web pages are displayed properly on a variety of devices, from desktop computers to smartphones. This is achieved through the use of CSS techniques such as media queries.

**Media Queries:** The @media rule allows different styles to be applied based on the type of device or its screen properties.

Example: @media (max-width: 1100px) {

body {

background-color: lightgrey;}

}

Media queries help deliver a tailored style sheet depending on: •Screen size (mobile, tablet, desktop) •Print styles for printed documents•Accessibility features like screen readers

**Media Features:** Media features provide detailed conditions for applying styles based on specific device properties:

Example: @media (min-width: 600px) and (max-width: 1200px) {

body {

font-size: 18px;}}

This ensures that the font size is adjusted for screens within the specified range.

**Responsive Frameworks:** Responsive frameworks provide pre-built CSS and JavaScript components that make it easier to develop mobile-friendly websites.

**Popular Responsive Frameworks:** •Bootstrap•Foundation•Bulma•UIkit•Semantic UI•Materialize•Tailwind CSS•Skeleton•Milligram•Spectre•Primer

**Bootstrap - The Most Popular Framework:** Bootstrap is a widely used front-end framework that includes pre-styled components, layout grids, and JavaScript plugins to facilitate responsive web design.

**Features of Bootstrap**

**HTML and CSS templates** for typography, forms, buttons, tables, navigation, modals, and image carousels.

**JavaScript plugins** for interactivity.

**Grid system** for flexible layouts.

**Mobile-first design**, ensuring content scales properly across all devices.

**Advantages of Bootstrap:**

**Easy to use**: Beginners with basic HTML and CSS knowledge can start using it immediately.

**Responsive design**: Ensures a consistent look and feel across different devices.

**Mobile-first approach**: The framework prioritizes mobile-friendly designs by default.

**Cross-browser compatibility**: Works with all modern web browsers.

**Creating a bullseye**

The HTML file defines three nested div elements inside a container. Each div represents a different part of the dartboard:

The outermost div (green) represents the largest circle.

The middle div (yellow) represents the second layer.

The innermost div (red) represents the bullseye.

**HTML Code**

<body>

<div class="container">

<div class="green">

<div class="yellow">

<div class="red"></div>

</div>

</div>

</div>

</body>

Each div is wrapped inside another div, which helps in positioning the circles centrally.

Step 2: Styling with CSS

Each div is styled as a circle using CSS properties. Let's break it down:

1. Green Circle (Outer Layer)

border-radius: 50%; makes the div a circle.

height and width are set to 250px.

background: darkgreen; it gives it a green color.

margin: 0 auto; centers it horizontally.

display: flex; align-items: center; helps center the inner elements.

2. Yellow Circle (Middle Layer)

height and width are 150px, making it smaller than the green circle.

background: goldenrod; gives it a yellow color.

margin: 0 auto; keeps it centered.

3. Red Circle (Bullseye)

height and width are 50px, making it the smallest.

background: darkred; gives it a red color.

margin: 0 auto; centers it inside the yellow circle.

**CSS Code (styles.css)**

.container {

display: flex;

justify-content: center;

align-items: center;

}

.green {

display: flex;

justify-content: center;

align-items: center;

border-radius: 50%;

height: 250px;

width: 250px;

background: darkgreen;

border: 5px solid black;

}

.yellow {

display: flex;

justify-content: center;

align-items: center;

border-radius: 50%;

height: 150px;

width: 150px;

background: goldenrod;

border: 5px solid black;

}

.red {

border-radius: 50%;

height: 50px;

width: 50px;

background: darkred;

border: 5px solid black;

}

Step 3: Understanding Layout and Alignment

1. Flexbox for Centering

display: flex; align-items: center; justify-content: center; is used in .green and .yellow to center the child elements inside them.

2. Nesting for Structure

The .yellow div is inside .green, and the .red div is inside .yellow.

This ensures the circles appear stacked inside each other.

3. Using Margins for Centering

margin: 0 auto; centers each div horizontally.

Step 4: Creating an Oval Shape

An oval is similar to a circle but has a different width and height. Instead of keeping width and height equal, we make one dimension larger.

**CSS Code for an Oval**

.oval {

width: 200px; /\* Wider than the height \*/

height: 100px; /\* Shorter height \*/

background: blue;

border-radius: 50%; /\* Ensures the shape becomes an oval \*/

margin: 20px auto;

display: flex;

justify-content: center;

align-items: center;

border: 5px solid black;

}

**HTML Code for an Oval**

How It Works

border-radius: 50% makes the div rounded.

The width is greater than the height, making it an oval.

The background color is blue, but you can change it.

**Step 5: Rendering the Shapes**

The dartboard structure consists of three nested circles.

The oval is a standalone shape that follows the same principles but with different width and height.

**Why Do the Shapes Automatically Form?**

1. Circles

You don’t need to explicitly define a circle because:

A div is normally a rectangle.

When you set border-radius: 50%, the browser makes it a perfect circle as long as width and height are equal.

2. Ovals

If width and height are different, border-radius: 50% creates an oval instead of a circle.