1.    Принцип работы Веб-приложения. Основные отличия работы

веб-приложения и настольного приложения.

Работа веб-приложения заключается в том, что клиент отправляет запросы на сервер и в зависимости от реализации того, какой это тип веб-приложения. Из наиболее используемых типов веб-приложений можно выделить следующие:

* SPA (Single Patch Application) Клиентские приложения
* Server-Side Rendering Application
* Progressive Web Application

SPA (клиентское) приложение, это тип веб-приложения, в котором создаётся один шаблон веб страницы, а его наполнение изменяется динамически. То есть у нас может быть какой-то элемент, вариации которого будут описаны фреймворками javascript’a (ReactJS, VueJS, AngularJS и тд.)

SSR приложение в свою очередь по запросу генерирует и отправляет на клиент новую HTML страницу, которая отображается в браузере. Такая реализация подразумевает «перезагрузку» страницы браузера каждый раз, когда выполняется запрос.

Прогрессивные веб приложения в свою очередь могут пользоваться преимуществами десктопных и веб-приложений. Основные преимущества это возможность использования приложения без подключения к интернету, а также возможность отправки уведомлений.

Основные отличия в работе настольных приложений от веб в том, что само приложение хранится у пользователя на запоминающем устройстве, при его запуске генерируется окно с соответствующими компонентами, и работа приложений построена на прерываниях, результат которых отображается на экране.

В веб-приложении клиент отправляет запрос на сервер и получает ответ в одном из вышеперечисленных форматов.

2.    Протокол HTTP. Доступ к элементам Http запроса из кода.

HTTP – Hypertext Transfer Protocol. Буквально можно сказать, что это протокол для передачи гипертекста.

Состоит из 3х частей:

1. Стартовая строка – тип сообщения
2. Заголовки – тело сообщения, параметры передачи и прочие сведения.
3. Тело сообщения – непосредственно данные сообщения.

Доступ к элементам HTTP получает сервер Kestrel, который передаёт параметры в свойства объекта HttpContext.

HttpContext.Session

HttpContext.Request

HttpContext.Response

HttpContext.Server

HttpContext.User

3.    Язык разметки HTML. Назначение, синтаксис.

HTML – hypertext markup language. Язык разметки гипертекста, который используется для отображения документов.

Синтаксис выглядит следующим образом:

<html> - тег, открывающий html документ

<head> - заголовок документа

</head>

<body> - тело документа

</body>

</html>

Значит html для размещения составных частей использует теги с параметрами, которые поддерживают kebab-style.

Тег div есть блочный элемент, предназначенный для выделения и изменения документа посредством изменения стиля.

ul – создаёт маркированный список с элементами <li>

ol – нумерованный список, где элементы списка начинаются с <li>

a – создаёт ссылки или якори, в зависимости от того, какой атрибут будет указан внутри. Если name, то создаётся якорь для закладки в внутри страницы.

1. Стили, CSS. Применение стилей и CSS на странице.

CSS – Cascading Style Sheets – это механизм управления представлением отдельных документов или их наборов. Стиль включает в себя все типы элементов дизайна: шрифт, фон, текст, цвета ссылок, поля и расположение объектов на  странице.

selector {prop1: value1;

prop2: value2;}

Селекторы класса могут выглядеть следующим образом

*p.warning* – этот селектор имеет отношение к первому элементу вида

*<p class=«warning»>*

.warning – будет иметь отношение к первому элементу, чей класс имеет имя warning.

*#left\_panel{fontweight: bold;}*

Данный селектор имеет отношение к первому элементу, чей id будет равен id=left\_panel

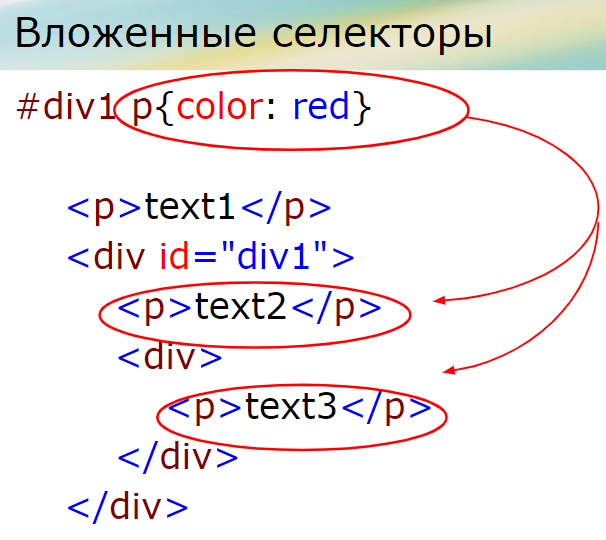
Селектор атрибута

[required]{fontweight: bold;}

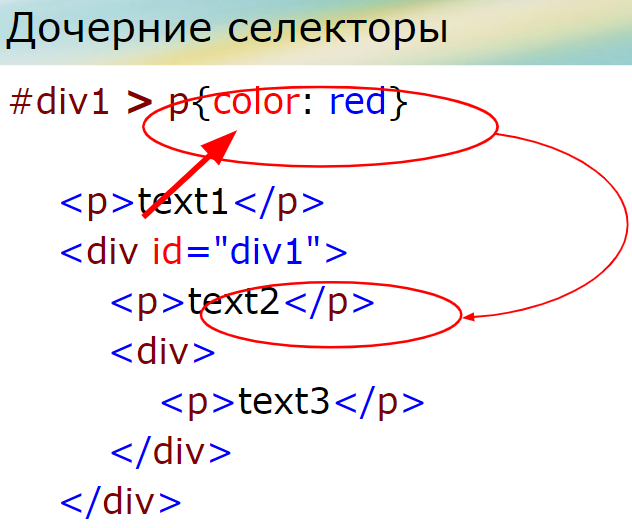
В разметке: <div id= «ala» required> </div>

Отношения между селекторами:

* Вложенность:



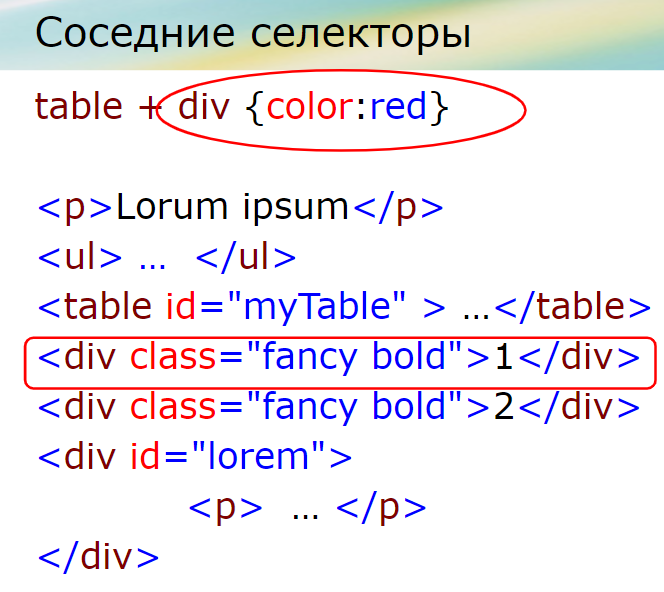
* Дочерние селекторы



* Родственные селекторы



* Соседние селекторы



1. Формы HTML, синтаксис, назначение. Методы Get и Post

Формы предназначены для обмена данными между пользователем и сервером. В документе может быть несколько форм, но на сервер будет отправлена только одна. Программа, которая будет обрабатывать данные с отправленной формы. Тип запроса указывается в аттрибуте method. Также форма отправляется на сервер с помощью кнопки Submit.

Метод GET используется для передачи коротких сообщений и переменных, которые передаются в явном виде через строку браузера.

Метод POST предназначается для передачи данных, заключённых в теле сообщения.

1. Структура каталогов приложения ASP.NET Core. Правила размещения файлов в проекте. Правила формирования имен файлов.

В структуру приложения ASP.NET Core входят **Content Root**, который представляет собой основной путь приложения, откуда доступны все реализованные в нём элементы. Также есть **WEB Root** для таких открытых статических ресурсов как файлы css, js и файлов изображений.

**Connected Services** – сервисы, используемые в данном приложении. Такие как Azure, сервис для подключения к базе данных.

**Dependencies** – используемые библиотеки или проекты.

**Wwwroot** – узел для хранения статических файлов.

В ASP.net есть определённые правила по наименованию файлов в проекте.

1. Имена файлов соответствуют CamelCase стилю.
2. Также если мы описываем контроллеры, они должны описываться в соответствующей директории Controllers, а сами файлы в названии также должны иметь суффикс Controller.
3. Если мы хотим описать различные представления или области, то для них мы создаём отдельные папки, где уже создаём необходимые нам файлы.
4. Если мы хотим создать общие представления, то для них мы создаём папку Shared, в которой и будем размещать соответствующие файлы.
5. Файлы appsettings.json, \_ViewStart.cshtml и \_ViewImports.cshtml. Назначение, использование.

В файле appsettings json как правило описываются параметры, которые мы потом используем либо в конструкторе класса Startup либо Program.cs, в зависимости от того, какую версию asp.net мы используем. Как правило в себе он хранит параметры для подключения к базе данных, а также параметры конфигурации логгирования. Например уровни логгирования, а также разрешённые хосты.

1. Класс Startup. Назначение. Методы Configure и ConfigureServices.

Класс Startup предназначен для конфигурации конвейера обработки запросов, а также занимается конфигурацией сервисов, используемых в разрабатываемом приложении.

Соответственно методы Configure определяет конвейер для обработки запросов.

А метод ConfigureService определяет сервисы, используемые приложением. ASP.NET MVC Core framework, Entity Framework Core, Identity

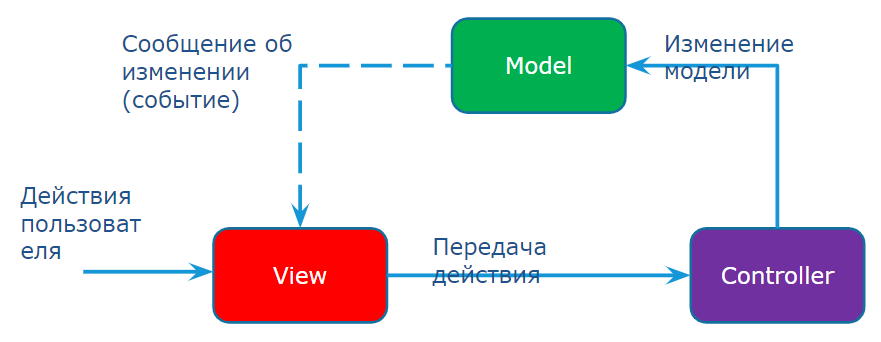
1. Шаблон проектирования MVC. Реализация MVC в ASP.NET Core.

MVC – шаблон проектирования model view controller в которых модель связи с базой данных, представление данных и взаимодействие с пользователем разделены на три компонента таким образом, что один компонент не зависит от двух других.

Собственно классы Моделей отвечают за представление и управление данными.

Представления отвечают за визуальное отображение модели.

Контроллер управляет логикой приложений, другими словами является связующим звеном между моделями данных и представлениями.



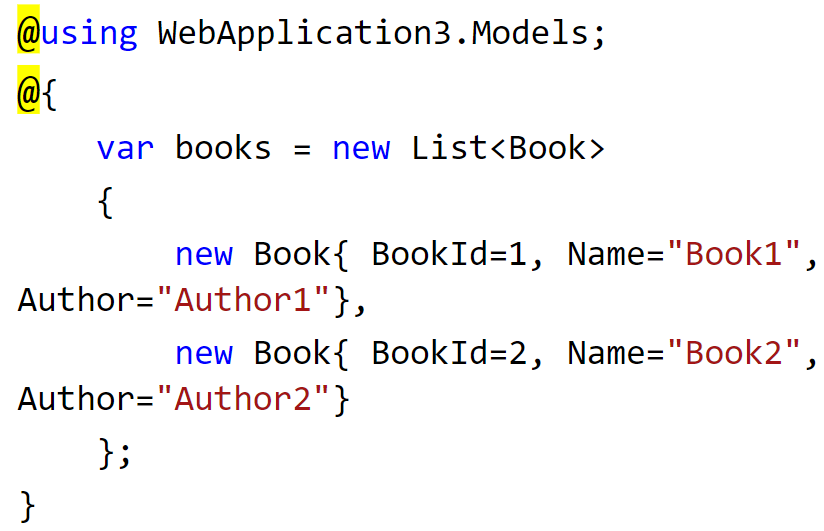
1. Представление ASP.NET. Язык Razor.

Представления в ASP.NET нееобходимы для отображения пользовательского интерфейса приложения, файлы которых имеют расширение cshtml. Что говорит о том, что отображение может производиться языком HTML со вставками языка Razor.

Файлы представлений располагаются в папке Views, а далее в зависимости от того, какие представления мы реализуем, то либо мы создаём в этой папке папку с названием контроллера и реализуем всё там, либо создаём папку Shared если представление может использоваться несколькими контроллерами.

Собственно, Razor язык, который позволяет делать вставки из C# и HTML. Есть несколько видов выражения Razor:

* Неявные выражения Razor - код, который идёт без пробелов, идущий после @. В таком коде нельзя использовать угловые скобки, потому что они будут восприняты как разметка HTML
* Явные выражения Razor – код, вид которого это @(code). Здесь уже можно использовать угловые скобки.
* Кодовые блоки – начинаются с @ и далее идут фигурные скобки с выражением.



1. Частичные представления. Методы для вызова частичных представлений.

Частичные представления позволяют нам изменять пользовательский интерфейс, без изменения основного layout а. Их особенностью является то, что при описании cshtml файла данного представления мы не используем мастер-страницу. То есть они отлично подходят для случаев, где нужно сделать изменение страниц без изменения основной. Отлично использовать для создания AJAX запросов, что обеспечивает работу со странице без её постоянной перезагрузки. Достигается это либо с помощью метода <partial name=”\_PartialView” model=@mod (если требуется)>. Либо же с помощью метода Html.PartialAsync(”\_PartialView”, model). В методе же контроллера мы используем метод PartialView(”\_PartialView”)

1. Страницы-макеты. Объявление секций на странице-макете. Использование страницы-макета в представлении.

Страницы макеты относятся к понятию мастер-страниц, которые в свою очередь могут быть необходимы в местах, где мы прописываем много раз одинаковые участки кода. Далее в мастер-страницах мы можем использовать методы @RenderBody() и @RenderSection(). Которые отвечают за отображения из подчинённых страниц. Чтобы определить, какую мастер-страницу будет использовать данный файл для представления, существует свойство Layout которое переопределяется в Razor секции. В случае если есть несколько мастер-страниц, автоматически выбирается та, которую находится в этом, либо в ближайшем каталоге. Значит, чтобы использовать какую-то конкретную мастер-страницу, нужно указывать полный путь к ней.

Что касается секций, то мы используем метод RenderSection(). Но в случае если мы определяем секцию, но не реализуем, то нам выдаст ошибку. Значит мы должны определить эту секцию видом

@section Footer {

Lalalala

}

Также данную секцию можно определить как необязательную к реализации, путём добавления @RenderSection(«Название», false).

1. Компоненты представлений. Использование компонента в представлении.

Компоненты представлений необходимы нам как альтернатива использованию частичных представлений, потому что компоненты представлений могут реализовывать более сложную логику, нежели частичные представления. Компоненты представлений используют класс C# и соответственно cshtml файл который рендерит наше представление.

Вызывается методом @ await Component.InvokeAsync(«Name») либо, если метод несинхронный, то Component.Invoke(«Name»).

Сам класс C# реализуется 3мя способами:

1 Создание класса с суффиксом ViewComponent

2 Cоздание класса-наследника от класса ViewComponent

3 Добавление перед классом аттрибута [ViewComponent]

В случае 2-3 в названии класса добавлять суффикс ViewComponent необязательно.

Аргументы передаются путём создания анонимных объектов, свойства которых мы определяем внутри new {Name = “Alex”, Surname = “Smith”}. Либо в определения компонента как тег-хелпера, то мы передаём как аттрибуты, описанные kebab-case’ом. Был аттрибут UserInfo – стал user-info.

<p>С секундами: <br />

@await Component.InvokeAsync("Timer", new { includeSeconds=true})

</p>

<p>Без секунд: <br />

<vc:timer include-seconds="false"></vc:timer>

</p>

14. Отображение данных в представлении. Свойства ViewBag и ViewData. Строго типизированные представления.

15. Вспомогательные (расширяющие) методы Html для генерации разметки.

16. Вспомогательные (расширяющие) методы Html для передачи управления и генерации частичных представлений.

17. Вспомогательные (расширяющие) методы Url. Метод Url.Content и свойство Server.MapPath.

18. Тэг-хелперы. Встроенные тэг-хелперы для формы.

19. Тэг-хелперы. Встроенные тэг-хелперы для передачи управления.

20. Шаблон проектирования «Внедрение зависимостей» (Dependency injection). IoC-ontainer.

21. Реализация шаблона проектирования Dependency injection в ASP.NET Core.

22. Контроллер ASP.NET. Назначение. Типы данных, возвращаемые методами контроллера. Вспомогательные методы класса Controller для возврата значения из метода контроллера.

23. Передача данных методам контроллера. Механизм привязки данных. Передача данных от контроллера представлению.

24. Фильтры, применяемые в контроллерах.

25. Аннотация и валидация данных. Проверка результата валидации. Вывод сообщений об ошибках.

26. Механизм маршрутизации ASP.NET Core. Свойства маршрута. Регистрация маршрутов.

27. Механизм маршрутизации ASP.NET Core. Маршрутизация с помощью атрибутов.

28. Генерирование URL c помощью вспомогательных методов Html и Url. Класс LinkGenerator. Чтение значений сегментов маршрута.

29. Передача файлов от клиента на сервер. Метод формы для передачи файла. Элемент формы для передачи файла.

30. Передача файлов от сервера к клиенту. Класс FileResult. Вспомогательный метод File.

31. Работа со статическими файлами.

32. Технология Ajax. Назначение, принцип работы. Формирование Ajax-запросов с помощью JQuery.

33. JSON. Создание JSON-запросов на Веб-странице. Формирование ответов контроллера в формате JSON.

34. Модульное тестирование в ASP.NET Core. Основные принципы модульного тестирования. Структура теста – шаблон ААА.

35. Модульное тестирование в ASP.NET. Библиотека Moq. Методы класса Assert. Тестирование контроллера.

36. Система аутентификации ASP.NET. Назначение. Принцип работы.

37. Система аутентификации ASP.NET Identity. Класс IdentityUser.

38. Основные методы классов UserManager, RoleManager и SignInManager. Проверка результата аутентификации и получение данных о текущем пользователе.

39. Атрибуты ограничения доступа к контроллеру или методу контроллера.

40. REST API контроллеры. Принцип обработки запросов

41. REST API контроллеры. типы данных, возвращаемых методами API контроллера.

42. Компоненты MiddleWare. Регистрация компонентов.

43. Сессии в ASP.NET Core.

44. Области (Areas). Назначение. Использование областей с контроллерами и страницами.

45. Razor pages. Обработка запросов при обращении к страницам

46. Razor pages. Привязка данных между страницей и кодом модели.

47. Razor pages. Маршрутизация страниц.

48. SignalR. Принцип работы.

49. Фреймворк Blazor. Принцип работы Blazor Server

50. Фреймворк Blazor. Структура проекта Blazor Server

51. Фреймворк Blazor. Принцип работы Blazor WebAssembly

52. Фреймворк Blazor. Структура проекта Blazor WebAssembly

53. Фреймворк Blazor. Компоненты Razor.

54. Фреймворк Blazor. Маршрутизация страниц.

55. Фреймворк Blazor. Привязка данных к параметрам маршрута

56. Фреймворк Blazor. Привязка данных внутри компонента

57. Фреймворк Blazor. Привязка данных от родительского компонента к дочернему

58. Фреймворк Blazor. Привязка данных от дочернего компонента к родительскому.

59. Фреймворк Blazor. Обработка событий

60. Фреймворк Blazor. Генерация ссылок в разметке и в коде. Отправка Http-запросов.