Core – программная платформа, разработанная майкрософт и предназначенная для разработки веб-приложений. Является развитием owin и .net core альтернатива. Своя реализация clr (coreclr) и библиотеками. OpenSource

Runtime Components – собственно ОС. Над ней есть .net core library. Над ней - .net core 2.0. Работу над линуксом, маком, виндой. ASP. NET Core залезает на территорию ASP .NET 4.6, можно с ним работать

Теперь всё устанавливается через nuget. Это все для обеспечения кроссплатформенности.

Какие приложения:

- MVC

- WebApi

- WebPages

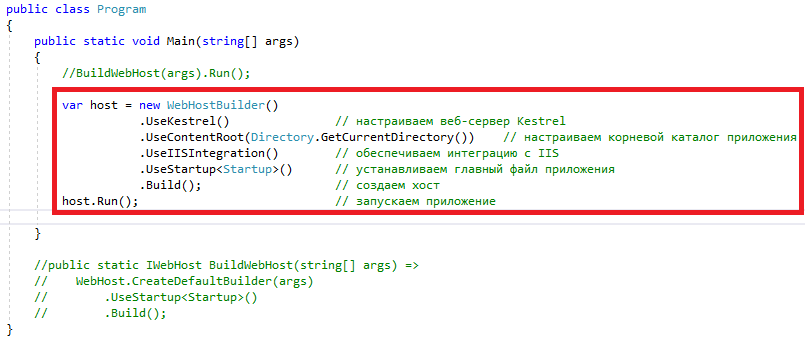
- WebForms не поддерживаются

Обыкновенный main, как у консольного приложения. Здесь создается BuildWebHost со стрелочной функцией. Он запускает цепочку функций, которые создают хост, который создает стандартный хттп сервер.

Класс Startup – это главный класс. Класс, который является главным классом в приложении. Он включает в себя 2 метода: ConfigureServices и Configure. Обработка сообщений, которые приходят по протоколу хттп – в run, там формируется response. Все запросы, что приходят – обрабатываются здесь.

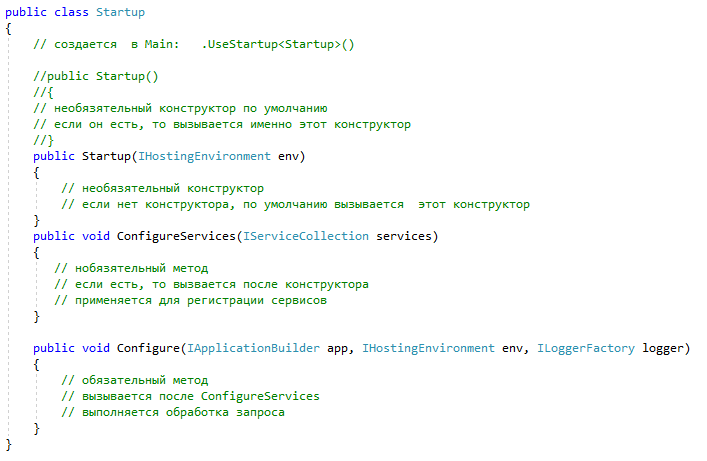
Можно стартовать в 2 режимах: как консольное приложение, или как IIS.

Что происходит в host.

****

Создается webhostbuilder. Настраиваем Kestrel. Указываем текущий директорий. Интеграция с IIS. Установка главного файла приложения.

Startup – он может иметь необязательный конструктор. Если есть – вызывается и передается объект. Если необязательный метод – ConfigureServices. Обязательный метод Configure – вызывается в конце в нем выполняется обработка запроса.

****

Структура приложения:

Wwwroot – предназначена для сохранения статической информации. Все то, что подтягиваем на html.

Для отображения контекста нужно подключить aspnetcore.all в нугете.

UseStaticFiles – если хотим юзать статический ресурс

UseDefaultPages – стартовая страница по умолчанию

DefaultFilesOptions – добавить свой элемент в статик пэйдж

UseDirectoryBrowser – файлы для скачивания

Добавление заголовков: в первом UseStaticFiles используем OnPrepareResponse => Context.Response.Headers.Append(,)

MVC:

Для того, чтобы использовать mvc, нужно в ConfigureServices указываем AddMvc, а в методе Configure используем UseMvc, определяем маршрутизатор. Можем сразу указать контроллер с дефолтным значением, экшн с дефолтным значением.

Контроллеры разрабатываются так же, как и в framework. В asp .net core и в webapi у контроллеров одинаковые наследуемые контроллеры, а также одинаковые маршрутизаторы.

Можно как и прежде располагать вью в директориях. View так же будет вызывать ту вью по имени action.

Для того, чтобы выводить в журнал консоли, можно добавить addConsole

Есть встроенное внедрение зависимости.

Параметры передаются так же, как и раньше. По имени параметра

Если хотим сделать маршрутизацию без таблиц маршрутизации – можно сделать через Route. Описывать маршруты явно для каждого action.

Есть множественные маршруты. Можно к одному action привязать много маршрутов. А можно указать сложный маршрут (с параметрами)

Можно включить журналирование. Если использовать CreateDefaultBuilder – автоматом ключается протоколирование.

Контроллер – программный класс, наследуемые от Controller или ControllerBase. КонтроллерБэйз не поддерживает View

Контроллер, некоторые особенности по поиску, reflection-поиск признаки:

- неабстрактный public

- не generic

- не вложени

- не декорирован

- наследуется от Control

Контекст контроллера – у него есть свойство ControllerContext – в нем есть ряд свойств. ModelState – спец объект, предназначенный для валидации данных.

Есть возможность создавать POCO-контроллеры – не наследники Controller, могут находиться в отдельной сборке.

Для использования сессии - UseSession

Требования к action:

* Нестатический puplic
* Не является generic
* Не помечен атрибутом
* Перегруженные методы могут быть отмечены атрибутами
* Нельзя ref и out
* Возвращает IActionResult

Можно использовать асинхронный вызов ExecuteResultAsync

OnAction – специальный обработчик, который можно использовать для контроллера. Можем сделать специальные обработчики до action или после action.

DependencyInjection – встроенный в кор. Для внедрения зависимости используется AddTrancient – для скопа. В данном случае, если хотим добавить какой-то объект. AddSingleton – синглтон скоп.