Введение

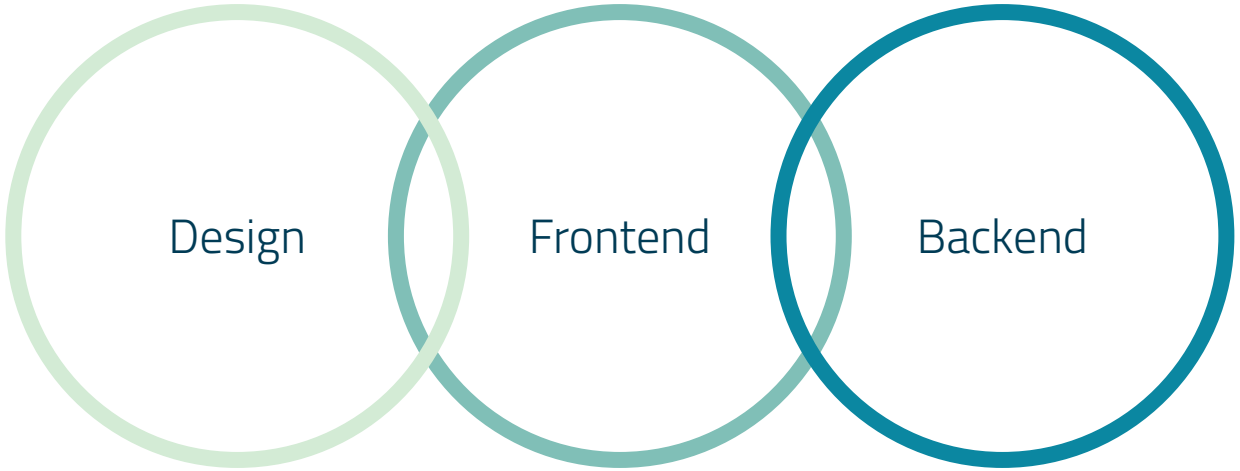
Проектирование и разработка веб-сервисов

Структура лекций

- Паттерн MVC.
- Маршрутизация.
- Серверная шаблонизация – Razor.
- Работа с базой данных – Entity Framework.
- Сервисы и Dependency Injection.
- Конфигурация приложения

Структура веб-приложения

Веб-приложение — клиент-серверное приложение, в котором клиент взаимодействует с сервером при помощи браузера, а за сервер отвечает — веб-сервер.



Структура веб-приложения

- Backend – серверная логика работы приложения (C#, ASP.NET Core MVC)
- Frontend – клиентская логика работы приложения (js, Jquery)
- Design – верстка, оформление(html, css)

Введение в ASP.NET Core

Платформа ASP.NET Core представляет технологию от компании Microsoft, предназначенную для создания различного рода веб-приложений: от небольших веб-сайтов до крупных веб-порталов и веб-сервисов.

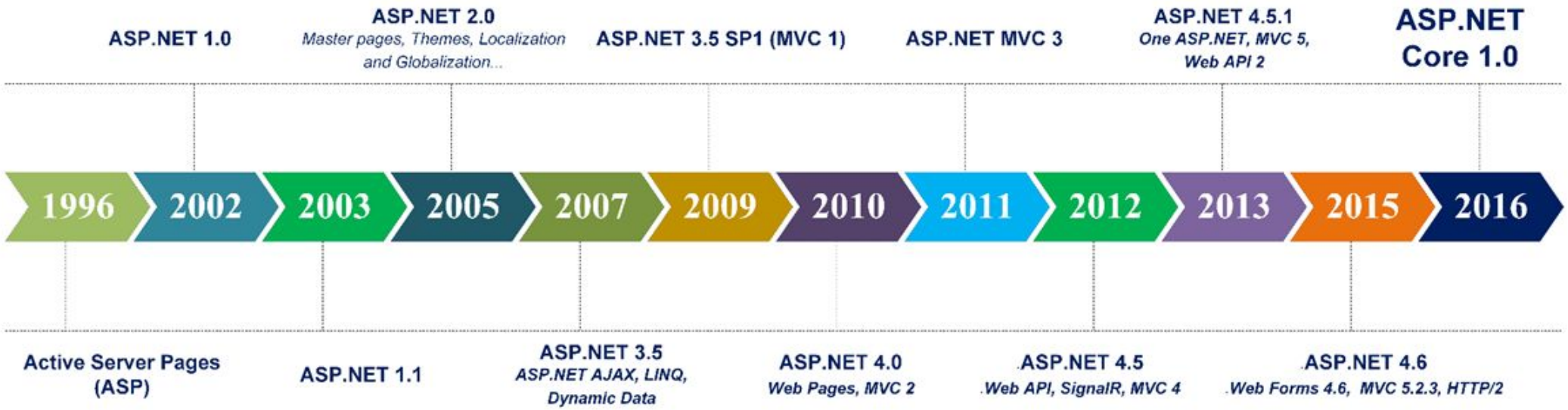
ASP.NET Core теперь полностью является opensource-фреймворком. Все исходные файлы фреймворка доступны на GitHub.

ASP.NET Core может работать поверх кросс-платформенной среды .NET Core, которая может быть развернута на основных популярных операционных системах: Windows, Mac OS X, Linux. И таким образом, с помощью ASP.NET Core мы можем создавать кросс-платформенные приложения.

Хотя ASP.NET Core преимущественно нацелено на использование .NET Core, но фреймворк также может работать и с полной версией фреймворка .NET.

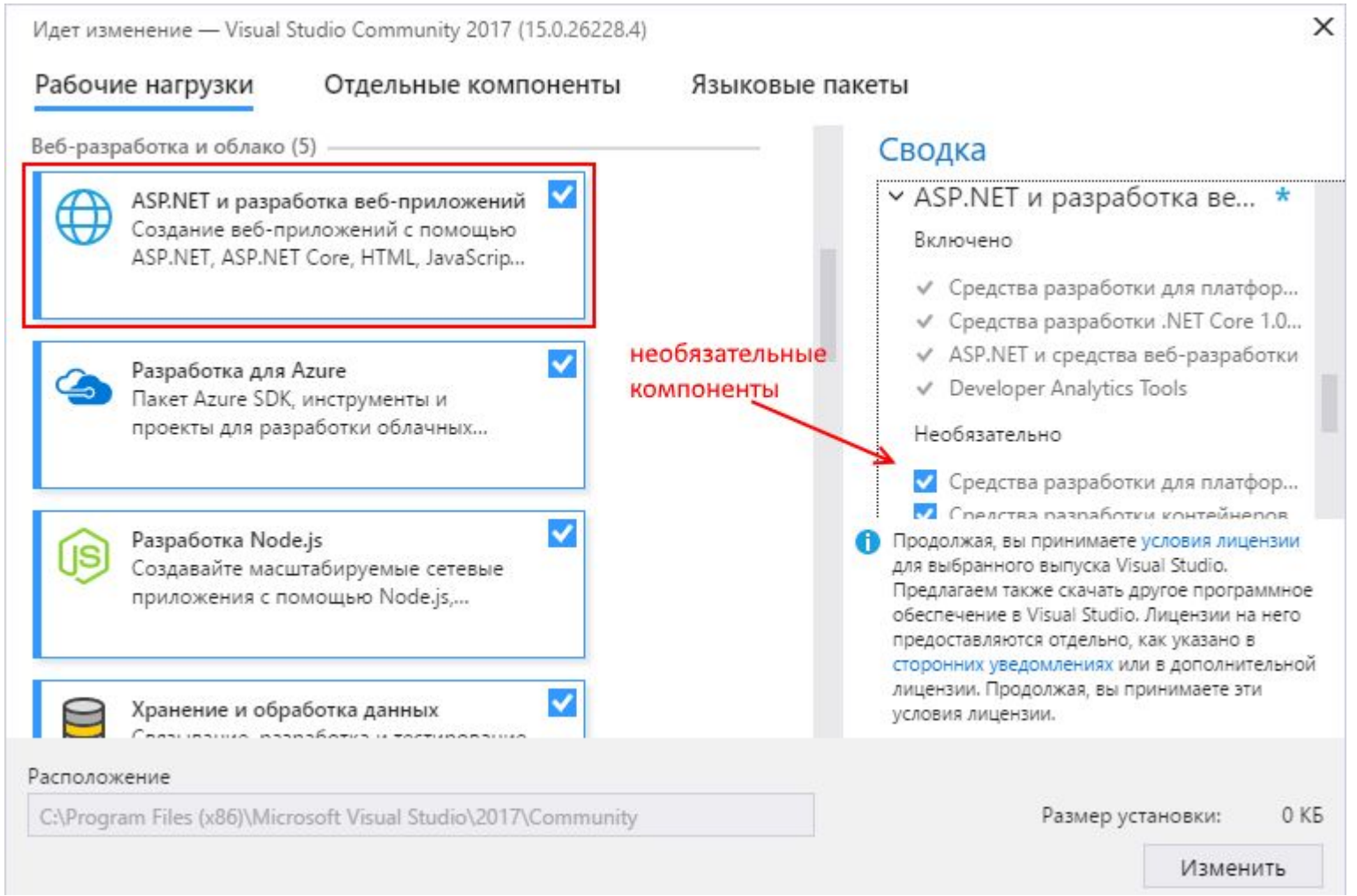
Благодаря модульности фреймворка все необходимые компоненты веб-приложения могут загружаться как отдельные модули через пакетный

История развития платформы ASP.NET



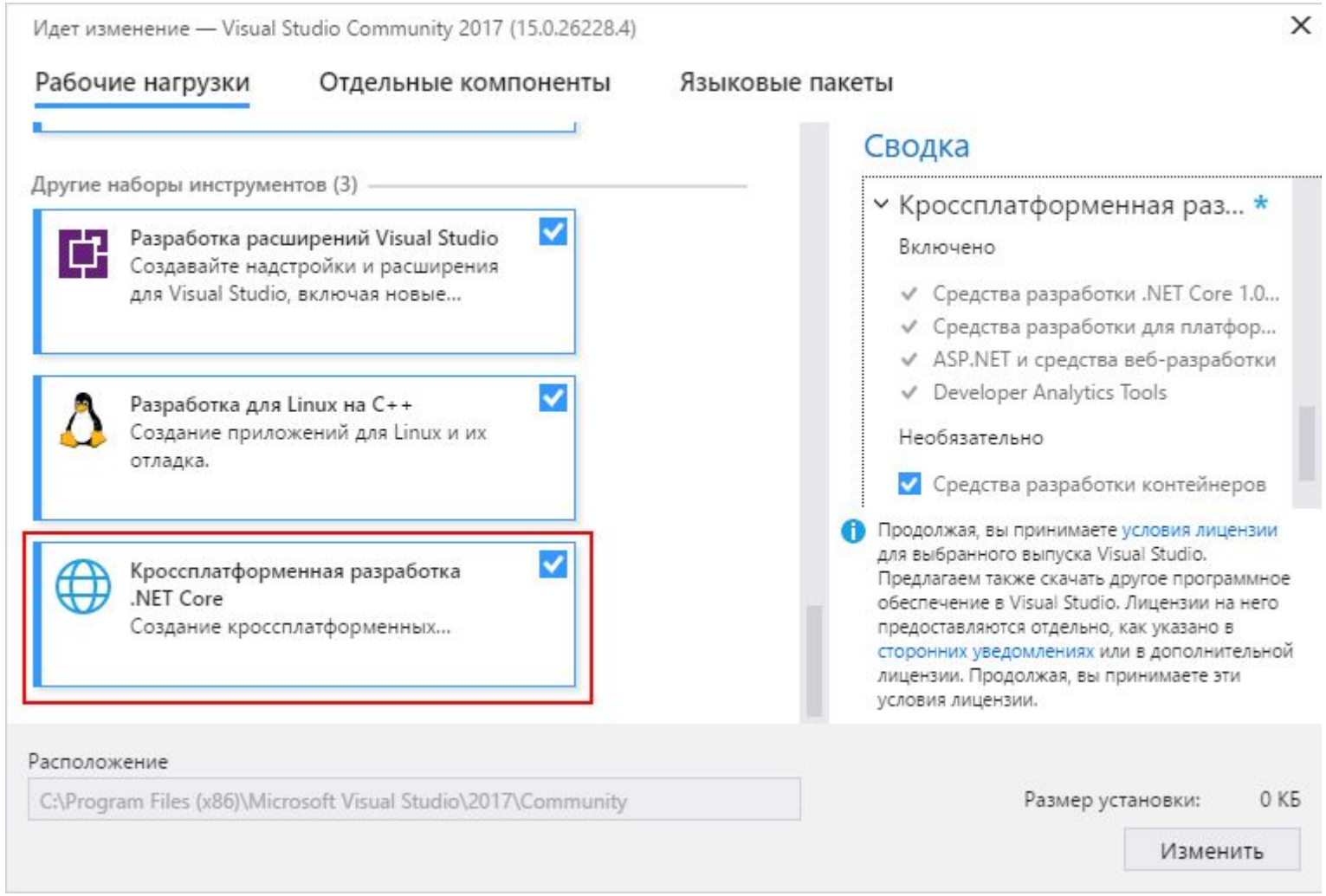
Среда разработки для ASP.NET Core

Для работы с ASP.NET Core при установке необходимо выбрать пункт «ASP.NET и разработка веб-приложений».

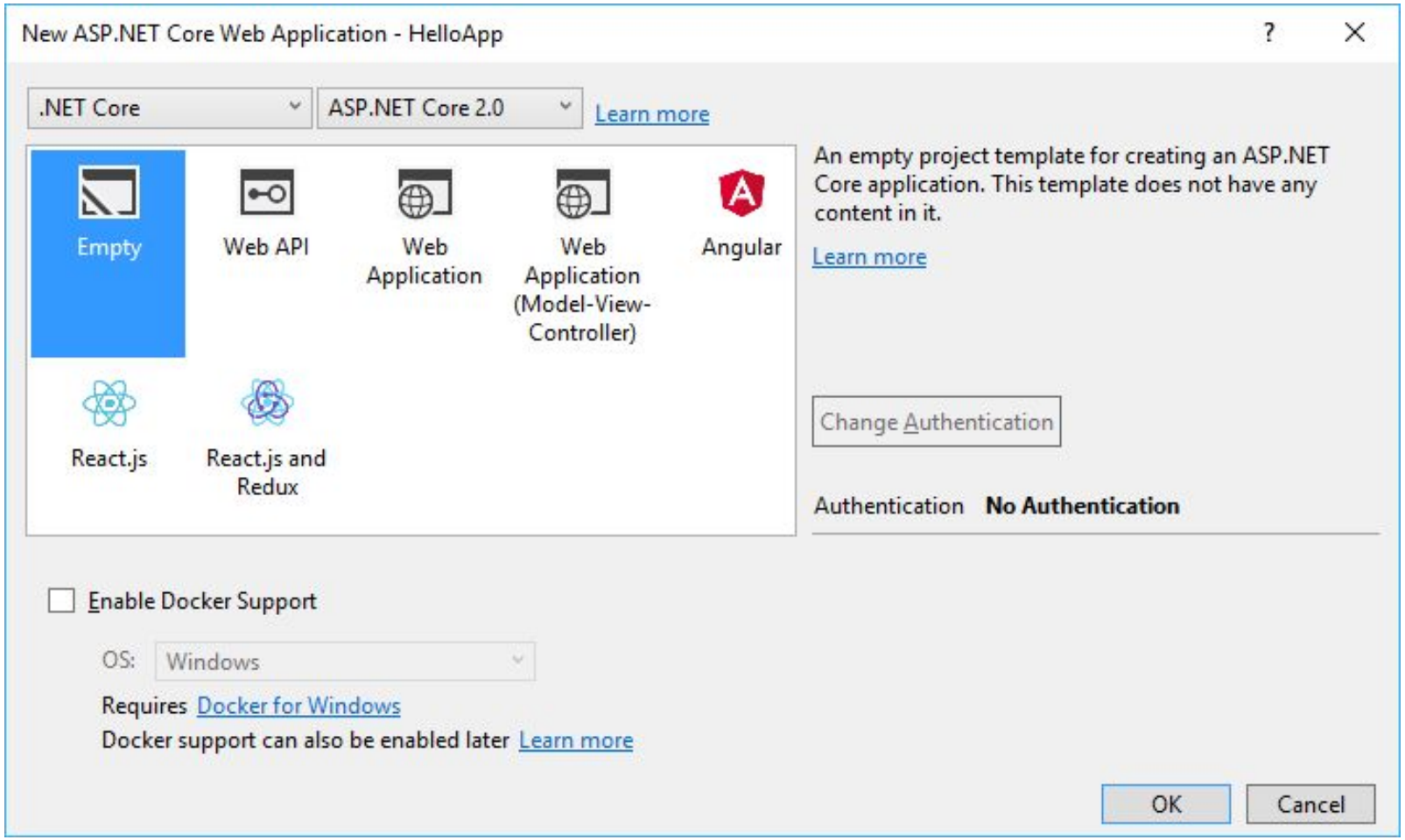


Среда разработки для ASP.NET Core

Кроме того, в программе установщика надо выбрать другой пункт «Кроссплатформенная разработка .NET Core».



Шаблоны приложения ASP.NET Core



Шаблоны приложения ASP.NET Core

- **Empty**: пустой шаблон с самой минимальной функциональностью для создания приложений с нуля
- **Web API**: проект веб-службы
- **Web Application**: проект, который для обработки запросов по умолчанию использует Razor Pages
- **Web Application(Model-View-Controller)**: проект, который использует архитектуру MVC
- **Angular**: проект, предназначенный специально для работы с Angular 2+.
- **React.js**: проект, который использует React.JS
- **React.js and Redux**: проект, который использует React.JS и Redux

Структура проекта ASP.NET Core

- **Connected Services**: подключенные сервисы из Azure
- **Dependencies**: все добавленные в проект пакеты и библиотеки, иначе говоря зависимости
- **wwwroot**: этот узел (на жестком диске ему соответствует одноименная папка) предназначен для хранения статических файлов - изображений, скриптов javascript, файлов css и т.д., которые используются приложением. Цель добавления этой папки в проект по сравнению с другими версиями ASP.NET, состоит в разграничении доступа к статическим файлам, к которым разрешен доступ со стороны клиента и к которым доступ запрещен (таким как project.json и т.д.).
- **Program.cs**: главный файл приложения, с которого и начинается его выполнение. Код этого файла настривает и запускает веб-хост, в рамках которого разворачивается приложение
- **Startup.cs**: файл, который определяет класс Startup и который содержит логику обработки входящих запросов

Класс Startup

Класс Startup является входной точкой в приложение ASP.NET Core. Этот класс производит конфигурацию приложения, настраивает сервисы, которые приложение будет использовать, устанавливает компоненты для обработки запроса или middleware.

```
public static void Main(string[] args)
{
    BuildWebHost(args).Run();
}

public static IWebHost BuildWebHost(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()    // установка класса Startup как стартового
        .Build();
}
```

Класс Startup

Класс Startup должен определять метод **Configure()**, и также опционально в Startup можно определить конструктор класса и метод **ConfigureServices()**.

При запуске приложения сначала срабатывает конструктор, затем метод **ConfigureServices()** и в конце метод **Configure()**. Эти методы вызываются средой выполнения ASP.NET.

Метод ConfigureServices

Необязательный метод ConfigureServices() регистрирует сервисы, которые используются приложением. В качестве параметра он принимает объект IServiceCollection, который и представляет коллекцию сервисов в приложении. С помощью методов расширений этого объекта производится конфигурация приложения для использования сервисов. Все методы имеют форму Add[название_сервиса].

Метод services.AddMvc() добавляет в коллекцию сервисов сервисы MVC. После добавления в коллекцию сервисов добавленные севисы становятся доступными для приложения.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
}
```

Метод Configure

Метод Configure устанавливает, как приложение будет обрабатывать запрос. Этот метод является обязательным. Для установки компонентов, которые обрабатывают запрос, используются методы объекта IApplicationBuilder. Объект IApplicationBuilder является обязательным параметром для метода Configure.

Кроме того, метод нередко принимает еще два необязательных параметра: IHostingEnvironment и ILoggerFactory:

- IHostingEnvironment: позволяет взаимодействовать со средой, в которой запускается приложение
- ILoggerFactory: предоставляет механизм логгирования в приложении

Метод Configure

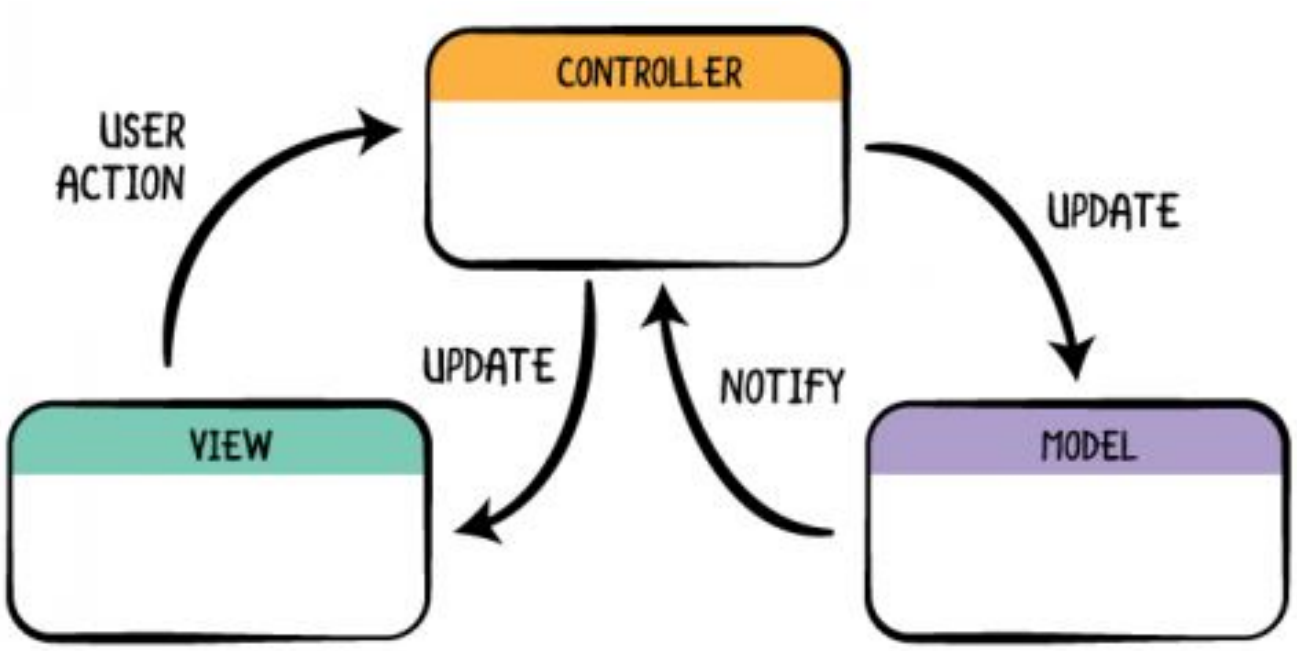
Большинство методов IApplicationBuilder имеют форму Use[название_сервиса]. Например, app.UseMvc() устанавливает компоненты MVC для обработки запроса и, в частности, настраивает систему маршрутизации в приложении.

```
app.UseStaticFiles();

app.UseMvc(routes =>
{
    routes.MapRoute(
        name: "default",
        template: "{controller}/{action=Index}/{id?}");
});
```

ASP.NET Core MVC

Ееверно отождествлять ASP.NET Core всецело с фреймворком ASP.NET Core MVC. Фреймворк ASP.NET Core MVC работает поверх платформы ASP.NET Core, и предназначен для того, чтобы упростить создание приложение.



Концепция паттерна MVC

Модель (model): описывает используемые в приложении данные, а также логику, которая связана непосредственно с данными, например, логику валидации данных. Как правило, объекты моделей хранятся в базе данных.

Модель может содержать данные, хранить логику управления этими данными. В то же время модель не должна содержать логику взаимодействия с пользователем и не должна определять механизм обработки запроса. Кроме того, модель не должна содержать логику отображения данных в представлении.

Концепция паттерна MVC

Представление (view): отвечают за визуальную часть или пользовательский интерфейс, нередко html-страница, через которую пользователь взаимодействует с приложением. Также представление может содержать логику, связанную с отображением данных. В то же время представление не должно содержать логику обработки запроса пользователя или управления данными.

Концепция паттерна MVC

Контроллер (controller): представляет центральный компонент MVC, который обеспечивает связь между пользователем и приложением, представлением и хранилищем данных. Он содержит логику обработки запроса пользователя. Контроллер получает вводимые пользователем данные и обрабатывает их. И в зависимости от результатов обработки отправляет пользователю определенный вывод, например, в виде представления, наполненного данными моделей.

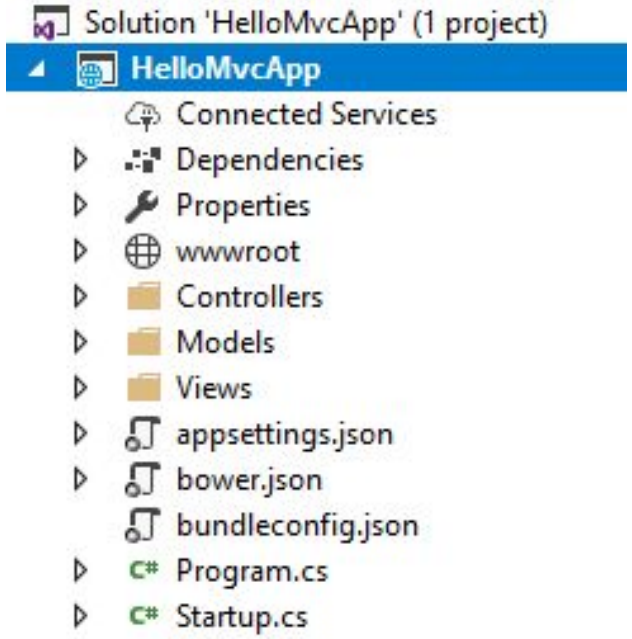
Концепция паттерна MVC

Отношения между компонентами паттерна можно описать следующей схемой:

Структура проекта ASP.NET Core MVC

Проект ASP.NET Core MVC имеет следующую структуру:

- **Controllers**: папка для хранения контроллеров, используемых приложением.
- **Models**: каталог для хранения моделей.
- **Views**: каталог для хранения представлений.
- **appsettings.json**: хранит конфигурацию приложения.
- **bower.json**: файл, который управляет клиентскими зависимостями (библиотеки javascript и css), которые подключаются через менеджер пакетов Bower.
- **bundleconfig.json**: файл, который содержит задачи по минификации используемых скриптов и стилей, которые выполняются при построении проекта



Структура проекта ASP.NET Core MVC

Проект ASP.NET Core MVC, создаваемый по умолчанию: