



Информатика

взаимодействие сервер-клиент,
ASP.NET Core

Ранее...

- ✓ Событийно-ориентированный подход
- ✓ Пользовательский интерфейс
- ✓ Клиент-серверное взаимодействие
- ✓ Request -> Response
- ✓ AJAX

Вспомним AJAX

```
<script type="text/javascript">
    function loadXMLDoc() {
        var xmlhttp; // Объект для совершения запроса
        if (window.XMLHttpRequest) {
            //for IE7+, Firefox, Chrome, Opera, Safari
            xmlhttp = new XMLHttpRequest();
        }
        else {
            // for IE6, IE5
            xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
        }
        // Отложенный вызов (callback) функции, когда меняется статус запроса
        xmlhttp.onreadystatechange = function () {
            /* readystate - статус запроса 0 - Unitialized, 1 - Loading,
            2 - Loaded, 3 - Interactive, 4 - Complete */
            if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
                document.getElementById("myDiv").innerHTML = xmlhttp.responseText;
            }
        }
        xmlhttp.open("GET", "http://localhost:8080/ajaxtest", true); // открывается соединение
        xmlhttp.send(""); // посылает запрос
    }
</script>

<button type="button" onclick="loadXMLDoc()">
    Get secret info from Server
</button> <p>Secret info is:<div id="myDiv"></div></p>
```

Ajaxtest HTTPListener

```
using System.Net;
public class Program
{
    public static void Main()
    {
        var listener = new HttpListener();
        listener.Prefixes.Add("http://localhost:8080/ajaxtest/");
        listener.Start();

        var context = listener.GetContext();

        context.Response.AppendHeader("Access-Control-Allow-Origin", "*");

        context.Response.ContentType = "text/xml";
        var secret = @"<date>42</date>";
        var bytes = System.Text.Encoding.UTF8.GetBytes(secret);
        context.Response.StatusCode = (int)HttpStatusCode.OK;
        context.Response.ContentLength64 = bytes.Length;
        context.Response.OutputStream.Write(bytes, 0, bytes.Length);

        listener.Stop();
    }
}
```

Пассивный сервер

- События генерируются клиентом
- Сервер только обрабатывает запросы

А что, если нужно обновлять данные на клиенте по событиям сервера?

Новое сообщение в переписке, новость в ленте...

Как быть?






- Можно постоянно опрашивать сервер на наличие обновлений
 - **Polling** (Google it!)
 - Опрос сервера на наличие событий с заданной периодичностью
 - **Long Polling** (Google it!)
 - Клиент посылает запрос, сервер держит соединение открытым, посылает данные, открывается новое соединение
 - В чём минусы?
- Как это должно работать, чтобы было хорошо?

Нужен активный сервер

- Сервер тоже должен иметь возможность генерации событий
- Пользователь должен слушать и обрабатывать события сервера

Server-sent events

- Инструмент **HTML5**
- Позволяет web-странице *автоматически* получать обновления с сервера
- Реализуется с помощью **EventSource** и события **onmessage**

| API |  |  |  |  |  |
|-----|--|--|--|--|--|
| SSE | 6.0 | Not supported | 6.0 | 5.0 | 11.5 |

Подписка на события сервера

```
var source = 'http://localhost:8080/';  
var loader = new EventSource(source);  
loader.onmessage = function (event) {  
    alert(event.data);  
};
```

Проверка поддержки sse браузером:

```
if (typeof (EventSource) !== "undefined") {  
    // Yes! Server-sent events support!  
} else {  
    // Sorry! No server-sent events support..  
}
```

Что на сервере?

- Сервер при подписке клиента на его события получает Request с заголовком **Accept: text/event-stream**
- Сервер должен генерировать ответы
 - с соответствующим ContentType
 - с кодом 200 Ok
 - без указания размеров передаваемых данных
 - не закрывать поток, использовать **Flush**

Что на сервере?

- Сервер держит соединение открытым!
- Генерирует ответ в текстовом формате
- Сервер может генерировать многострочный текст (используя \n)
- Для завершения строки пишется \n\n

Текущее время каждую секунду

```
static void Main(string[] args)
{
    var listener = new HttpListener();
    listener.Prefixes.Add("http://localhost:8080/");
    listener.Start();

    Task.Run(async () =>
    {
        while (true)
        {
            var ctx = await listener.GetContextAsync();

            if(ctx.Request.Headers["Accept"]=="text/event-stream")
                Task.Factory.StartNew(()=>SSEHandleAsync(ctx));
        }
    }).Wait();
}
```

Текущее время каждую секунду

```
public static async Task SSEHandleAsync(HttpContext context)
{
    var response = context.Response;
    response.StatusCode = (int)HttpStatusCode.OK;
    response.ContentType = "text/event-stream";
    response.AddHeader("Access-Control-Allow-Origin", "*");
    try
    {
        while (true)
        {
            var msg = $"data: {DateTime.Now}\n\n";
            var bytes = Encoding.UTF8.GetBytes(msg);
            response.OutputStream.Write(bytes, 0, bytes.Length);
            response.OutputStream.Flush();
            await Task.Delay(1000);
        }
    }
    catch (HttpListenerException e)
    {
        Console.WriteLine(e.Message);
    }
}
```

Всё вместе

- Пример серверного кода, генерирующего SSE с текущим временем
- Пример страницы с подпиской на SSE и обновлением содержимого

По ссылкам старая версия кода, в слайдах чуть новее

Чат на SSE

- **События SSE:** приход в онлайн, уход в оффлайн, сообщения
- **Схема работы:**
 - При входе в чат запрашивается имя
 - Клиент подключается к серверу, создаётся поток событий
 - При подключении/отключении клиента всем рассылается событие
 - Клиент отправляет сообщения по HTTP POST /message
 - Сервер обрабатывает сообщения и рассылает всем клиентам

[Статья с примером чата на JS. Переписать на C#](#)

SSE – однонаправленная связь

- Клиент слушает
- Сервер генерирует события
- Если клиент хочет отправить сообщения серверу, нужно посылать Request
- **А что, если сделать связь двусторонней?**

WebSocket

- Протокол *двунаправленной* связи поверх TCP для обмена сообщениями между сервером и клиентом в реальном времени
- Для установления соединения, участники по HTTP устанавливают соединение (рукопожатие - handshake)
- Google: WebSocket

WebSockets

- `var ws = new WebSocket("ws://localhost:8080");`
- `ws:` – схема URL для WebSockets (`wss:` `https`)

+ Обработчики для получения сведений о подключениях, сообщениях и ошибках (пример: `onopen`, `onmessage`, `onerror`, `onclose`)

Методы: `send(data)`, `close()`

А картинку можно отправить?

- С помощью WebSockets можно передавать как текстовые, так и бинарные данные
- Можно написать чат на стероидах
- Или крутое приложение для мониторинга биржевых котировок в реальном времени!

Пример WebSockets: всё вместе

- Сервер WebSocket, отправляющий обратно сообщения клиента
- Web-страница, с примером использования WebSocket

Итого:

WebSockets:

- Универсальные
- Быстрые и эффективные
(можно мониторить торговые площадки)
- Не ограничены по времени
(клиенту – хорошо, но нагрузка на сервер, нужен мультиплексор)
- Нет проблем с кросс-доменностью

Ок...

- Посмотрели много всего про принципы web-разработки, про использование `HttpListener` и реализацию этих принципов на нём
- А как оно во «взрослой» web-разработке?

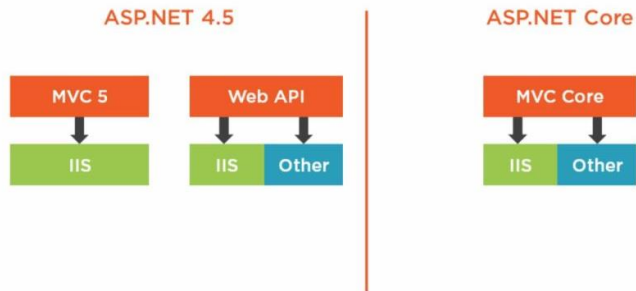
А давайте посмотрим...

- Очень коротко о том, как выглядит веб-разработка с использованием фреймворка
 - На примере ASP.NET Core MVC
 - Может какие идеи используете в семестровках
- А ASP.NET Core мы будем вплотную заниматься в следующем семестре

ASP.NET Core

- Кроссплатформенный фреймворк для создания web-приложений и API
- ASP.NET Core имеет открытый исходный код, доступный на GitHub ([ссылка](#))

Understanding MVC and Web API



Что было с ASP.NET

- Первый релиз **ASP.NET** состоялся 5 января 2002 года (**15 лет назад**)
- ASP.NET основывается на System.Web.dll и содержит много устаревшего кода, нет модульности (сразу подключается много всего)
- Разные подходы к созданию веб-приложений MVC и Web API

Почему ASP.NET Core

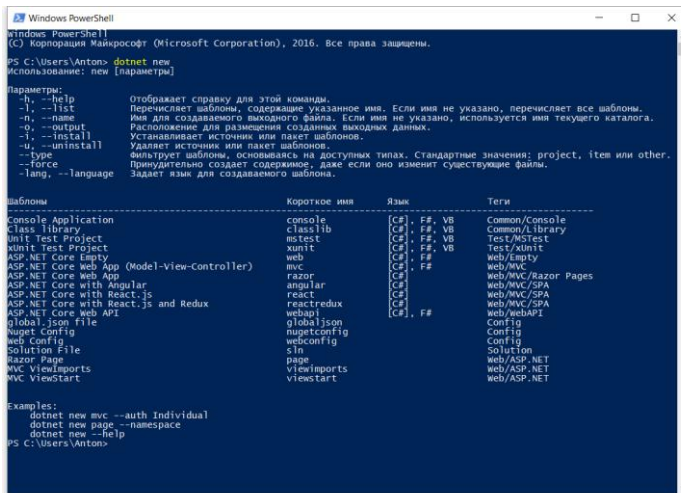
- Компактный, модульный фреймворк
- По умолчанию не устанавливается практически ничего (нужное докачивается через NuGet пакеты)
- Нет привязки к конкретному веб-серверу, в особенности к IIS, как было в ASP.NET

Улучшения

- Единый подход к Web UI и Web API
- Интеграция клиентских фреймворков
- Встроенное внедрение зависимостей
- Новый инструментарий
- Гибкие настройки хостинга

Фишки

- Инструменты для разработчиков .net core
 - Работа с Powershell и Developer's command prompt
 - Команды dotnet (new, run, help, watch)



```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation), 2016. Все права защищены.

PS C:\Users\Anton> dotnet new
Использование: new [параметры]

Параметры:
-h, --help          Отображает справку для этой команды.
-l, --list          Перечисляет шаблоны, содержащие указанное имя. Если имя не указано, перечисляет все шаблоны.
-o, --output        Имя для создаваемого выходного файла. Если имя не указано, используется имя текущего каталога.
-o, --output        Расположение для размещения созданных выходных данных.
-i, --install       Устанавливает источник или пакет шаблонов.
-u, --uninstall     Удаляет источник или пакет шаблонов.
--type             Вызывает шаблоны, основываясь на доступных типах. Стандартные значения: project, item или other.
--force            Принудительно создает содержимое, даже если оно изменит существующие файлы.
--lang, --language Задает язык для создаваемого шаблона.

Шаблоны:
-----
Шаблоны          Короткое имя  Язык          Тег
-----
Console Application  console      C#, F#, VB    Common/Console
Class Library       classlib     C#, F#, VB    Common/Library
Unit Test Project   unittest     C#, F#, VB    Test/UnitTest
ASP.NET Core Empty  empty        C#, F#         web/Empty
ASP.NET Core Web App (Model-View-Controller) mvc          C#            web/MVC/Razor Pages
ASP.NET Core Web App  razor        C#            web/MVC/SPA
ASP.NET Core with Angular angular       C#            web/MVC/SPA
ASP.NET Core with React.js and Redux reactredux   C#            web/MVC/SPA
ASP.NET Core Web API webapi       C#, F#        web/WebAPI
global.json file    globaljson
NuGet Config        nugetconfig
Web Config           webconfig
Solution File        sln
Razor Page           page
MVC ViewImports      viewimports
MVC ViewStart        viewstart

Примеры:
dotnet new mvc --auth Individual
dotnet new page --namespace
dotnet new --help
PS C:\Users\Anton>
```

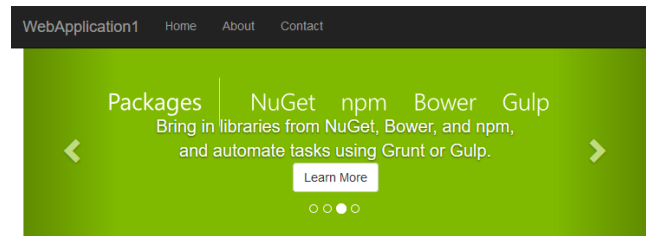
ASP.NET Core приложение

- По сути – просто консольное приложение, которое создаёт сервер в методе Main

```
using System.IO;  
using Microsoft.AspNetCore.Hosting;
```

```
namespace WebApplication1
```

```
{  
    public class Program  
    {  
        public static void Main(string[] args)  
        {  
            var host = new WebHostBuilder()  
                .UseKestrel()  
                .UseStartup<Startup>()  
                .Build();  
  
            host.Run();  
        }  
    }  
}
```



```
C:\Program Files\dotnet\dotnet.exe  
Hosting environment: Development  
Content root path: D:\Programming\Projects\WebApplication1\WebApplication1  
Now listening on: http://localhost:49672  
Application started. Press Ctrl+C to shut down.  
info: Microsoft.AspNetCore.Hosting.Internal.WebHost[1]  
      Request starting HTTP/1.1 GET http://localhost:49672/  
info: Microsoft.AspNetCore.Mvc.Internal.ControllerActionInvoker[1]  
      Executing action method WebApplication1.Controllers.HomeController.Index (WebApplication1) with arguments ((null)) - ModelState is Valid  
info: Microsoft.AspNetCore.Mvc.ViewFeatures.Internal.ViewResultExecutor[1]  
      Executing ViewResult, running view at path /Views/Home/Index.cshtml.  
info: Microsoft.Extensions.DependencyInjection.DataProtectionServices[0]  
      User profile is available. Using 'C:\Users\Anton\AppData\Local\ASP.NET\DataProtection-Keys' as key repository and Windows DPAPI to encrypt keys at rest.  
info: Microsoft.AspNetCore.Mvc.Internal.ControllerActionInvoker[2]  
      Executed action WebApplication1.Controllers.HomeController.Index (WebApplication1) in 253.89103ms  
info: Microsoft.AspNetCore.Hosting.Internal.WebHost[2]
```

Паттерн Builder

- *«Предоставляет способ создания составного объекта»*
- Позволяет изменять процесс конструирования, выбирать части
- **WebHostBuilder** развёртывает веб-приложение согласно этому паттерну
- Использует Fluent синтаксис (цепочки)

Startup

- С помощью UseStartup можно указать builder'у класс с настройками приложения
- Класс должен быть открытым
- Должны быть методы
 - **Configure** – настройки Middleware (связующего ПО)
 - **ConfigureServices** – настройки используемых сервисов (MVC, Entity, Identity, ...)

Configure

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env,
ILoggerFactory loggerFactory)
{
    loggerFactory.AddConsole(Configuration.GetSection("Logging"));
    loggerFactory.AddDebug();
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseBrowserLink();
    }
    else app.UseExceptionHandler("/Home/Error");

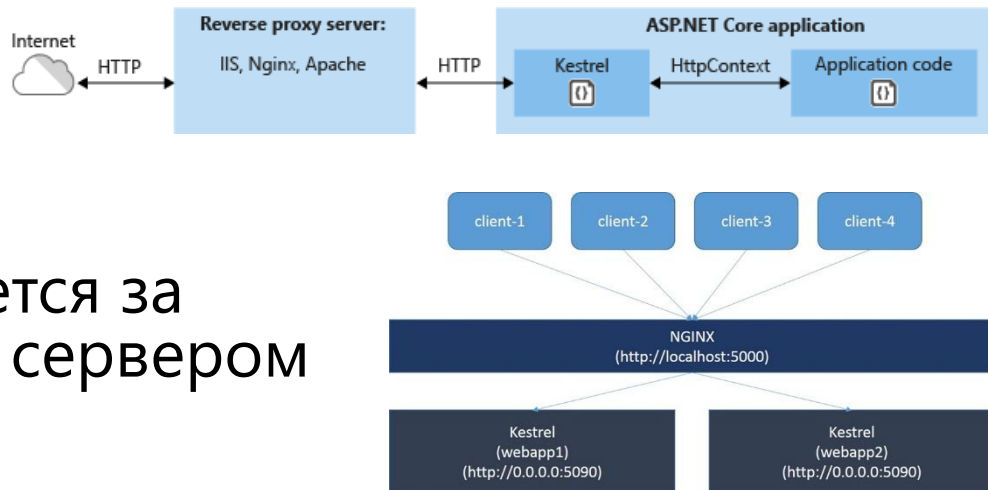
    app.UseStaticFiles();
    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
    });
}
```


Middleware – связующее ПО

- С помощью Use'ов подключается Middleware
- Middleware асинхронно обрабатывает HttpContext и либо вызывает следующего, либо сам обрабатывает запрос
- С ASP.NET Core используется любое связующее ПО, основанное на [OWIN](#)
 - Статические файлы, роутинг, аутентификация
 - Можно писать своё

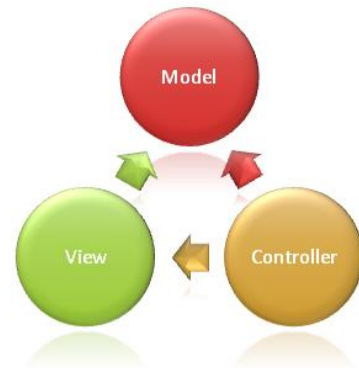
Серверы

- Приложение ASP.NET Core не слушает запросы напрямую, а полагается на веб-сервер, передающий ему запросы
- ASP.NET Core включает лёгкий кроссплатформенный веб-сервер Kestrel
 - http-сервер по сути
- Kestrel обычно запускается за производственным веб сервером (IIS, nginx, ...)



ASP.NET Core MVC

- Способ создания динамических веб-приложений на основе ASP.NET Core
- **Архитектура Model-View-Controller**
 - Разделение ответственностей
 - **Модель** – бизнес логика
 - **Представление** – отображение контекста в UI, использующее шаблонизатор Razor
 - **Контроллер** – обработка запросов, работа с моделью, выбор представления для отображения контента



Функционал ASP.NET Core MVC

- Роутинг
- Связывание и валидация моделей
- Внедрение зависимостей
- Шаблонизатор Razor
- Строго типизированные представления
- Scaffolding Visual Studio

Роутинг

- URL – Mapping, сопоставления пути с обработчиком
 - /Products/Details/17
 - {controller = Products, action = Details, id = 17}
- Позволяет создавать приложения с понятными и четкими URL (хорошие для SEO)

Настройки роутинга

- Глобальные в настройках связующего ПО

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

- Атрибутивный роутинг

– Указание роутинга в атрибутах (гораздо ближе к контроллеру и действиям)

```
[Route("api/[controller]")]
public class ProductsController : Controller
{
    [HttpGet("{id}")]
    public IActionResult GetProduct(int id)
    {
        ...
    }
}
```

Связывание и валидация моделей

- **Связывание моделей** – конвертация Request'a в объекты, которые может обработать контроллер и передача методам контроллера в виде параметров
- **Валидация моделей** – использование атрибутов-аннотаций для валидации данных на клиенте и на сервере до вызова методов контроллера

Внедрение зависимостей

```
public void ConfigureServices(IServiceCollection services)
{
    // Add framework services.
    services.AddDbContext<ApplicationDbContext>(
        options => options.UseSqlServer(
            Configuration.GetConnectionString("DefaultConnection")));

    services.AddIdentity<ApplicationUser, IdentityRole>()
        .AddEntityFrameworkStores<ApplicationDbContext>()
        .AddDefaultTokenProviders();

    services.AddMvc();

    // Add application services.
    services.AddTransient<IEmailSender, AuthMessageSender>();
    services.AddTransient<ISmsSender, AuthMessageSender>();
}
```


Внедрение зависимостей

- Dependency Injection
- Контроллеры могут запросить сервисы через конструкторы
- Принцип явных зависимостей (Google it!)

Razor

- Движок представления, шаблонизатор
- Язык разметки, использующий встраивание C# кода в HTML разметку для генерации динамического контента

```
<ul>  
  @for (int i = 0; i < 5; i++)  
  {  
    <li>List item @i</li>  
  }  
</ul>
```

Strongly typed views

Контроллер может передавать представлению строго типизированную модель, будет работать проверка типа и IntelliSense

```
@model IEnumerable<Product>
<ul>
  @foreach (Product p in Model)
  {
    <li> @p.Name </li>
  }
</ul>
```

Сессии

- Microsoft.AspNetCore.Session пакет NuGet
- Добавляется к сервисам AddSession()
- Регистрируется Middleware UseSession()
- Доступны в контроллерах через HttpContext.Session
- Можно настроить кэширование по умолчанию services.AddDistributedMemoryCache()

services.AddSqlServerCache(o =>

```
{o.ConnectionString = "Server=.;Database=ASPNET5SessionState;Trusted_Connection=True;";  
o.SchemaName = "dbo"; o.TableName = "Sessions";});
```

Scaffolding

- Генерация кода контроллеров и представлений
- На основе модели и спецификаций генерируется код для CRUD операций, а также представления
- Есть во многих фреймворках (вдохновлялись Ruby on Rails)

Server-Side Events в ASP.NET Core

Нужно реализовать

- Middleware, Controller и View
- [Пример со Stackoverflow](#)
- Или использовать SignalR
 - Библиотеку для коммуникаций реального времени (WebSockets, Long polls, SSE, ...)

Почитать

- [Документация Microsoft](#)
- [Обучалки по Html5 и CSS](#)
- [Открытая площадка с онлайн курсами от Microsoft \(aka Virtual Academy\)](#)



Вопросы?

e-mail: marchenko@it.kfu.ru