**Оглавление**

[1. Протокол HTTP: клиент-сервер; типы сообщений, структура запроса, структура отвта, статус (серии значений), методы, заголовки, параметры. Понятие stateless-протокола. 2](#_Toc136779448)

[2. Протокол HTTPS: TLS, шифронаборы, сертификаты, процедура рукопожатия. 4](#_Toc136779449)

[3. HTML. Структура HTML-страницы. Каскадные таблицы стилей (CSS). Модель DOM. Пример. 6](#_Toc136779450)

[4. Протокол WebSockets: принципы работы и применения. Пример 7](#_Toc136779451)

[5. JavaScript. Основные стандарты. Типы данных. Программные структуры. Принцип применения. Пример. 8](#_Toc136779452)

[6. Методология Ajax. Структура Ajax-приложения, принципы разработки и применения. Пример. 10](#_Toc136779453)

[7. Web-приложение. Архитектура web-приложения. Особенности реализации web-приложения. Web-сервер и web-клиент. Пример. 11](#_Toc136779454)

[8. ASP.NET: публикация ASP.NET-приложения, структура и параметры узла IIS, реальный и виртуальный каталоги, процедура настройки web-узла. 12](#_Toc136779455)

[9. ASP.NET: http-обработчики, порядок разработки, http-обработчик для взаимодействия с клиентом по протоколу WebSockets. Пример. 13](#_Toc136779456)

[10. ASP.NET: ASMX-сервисы, WSDL, SOAP, прокси, порядок разработки, принципы применения. Пример. 14](#_Toc136779457)

[11. ASP.NET: MVC-приложение, структура MVC(R)-приложения, назначение основных компонентов приложения, маршрутизация. Пример. 16](#_Toc136779458)

[12. ASP.NET CORE: программная платформа, принципы работы, архитектура. Пример. 19](#_Toc136779459)

1. Протокол HTTP: клиент-сервер; типы сообщений, структура запроса, структура ответа, статус (серии значений), методы, заголовки, параметры. Понятие stateless-протокола.

**ИНТЕРНЕТ** –сеть построенная на стеке протоколов tcp/ip

**Web-приложение** – это клиент-серверное приложение, взаимодействие в котором осуществляется по протоколу http

**HTTP** – это протокол прикладного уровня, работающий поверх tcp (начиная с версии http3 поверх udp) для передачи данных между клиентом и сервером, характеризуется полудуплексным каналом связи

**Полудуплексный канал связи** – это канал в котором в один момент времени данные могут идти в одном направлении, либо от клиента к серверу, либо от сервера к клиенту

**Типы сообщений**

* request
* response

**Структура request и response** сообщений одинаковая, включает в себя:  
- \*стартовую строку  
- заголовки  
- \*пустую строку  
- и тело

**request и response отличаются стартовой строкой.  
Стартовая строка request** состоит из:  
- метода  
- URI  
- и версии протокола Http  
**Стартовая строка response** состоит из:  
- версии протокола  
- кода состояния  
- и пояснения к коду состояния

**Заголовки –** это пары ключ-значение, которые хранят мета-информацию о сообщении(пакете)

**Типы заголовков:**

* **General** – используются в запросах и ответах (примеры: Cache-Control, Upgrade, Connection)
* **Request** – используются в запросах (примеры: Accept, Host, User-Agent)
* **Response** – используются в ответах (примеры: WWW-Authenticate, Location)
* **Entity** – используются для описания тела сообщения (примеры: Content-Type, Content-Language, Content-Length)

**Коды состояний**

* 1xx – информационные сообщения
* 2xx – успешные
* 3xx – переадресация
* 4xx – ошибка клиента
* 5xx – ошибка сервера

**http-методы**

* **Get** – используется для получения данных с сервера
* **Post** – для добавления данных
* **Put** – для обновления данных
* **Patch** – для частичного обновления данных
* **Delete** – для удаления данных
* **Options** – для описания параметров соединения с целевым ресурсом

**Понятие Stateless протокола**

Данное понятие говорит о том, что сервер не сохраняет никаких данных клиента между запросами, каждый запрос является изолированным друг от друга.

1. Протокол HTTPS: TLS, шифронаборы, сертификаты, процедура рукопожатия.

**HTTPS** – это тот же http, который использует протокол tls для шифрования сообщений (его предшественник - протокол ssl). Шифрование проиходит непосредственно над транспортным уровнем (tcp)

Таким образом мы получаем защищенный канал связи

по умолчанию https протокол использует 443 tсp-порт

**TLS** – это криптографический протокол, обеспечивающий защищенную передачу данных по сети

**Шифронаборы**

В шифронабор входит:

* алгоритмы шифрования и сжатия
* а также используемая хеш-функция

**Сертификаты**

ключевым аспектом реализации tls протокола является использование tls-сертификатов (стандарт X.509)

У центра сертификации есть свой сертификат, которым он подписывает сертификаты на ресурсы

Сертификат X.509 внутри себя содержит следующую информацию:

* доменное имя, на которое оформлен сертификат
* имя владельца сертификата
* имя издателя
* срок действия
* открытый ключ
* подпись сертификата

**Процедура рукопожатия**

* клиент обращается к серверу со списком алгоритмов шифрования, сжатия и хеш-функций
* сервер высылает сертификат на ресурс и также выбирает подходящий шифронабор
* клиент проверяет сертификат, у клиента есть список центров сертификации, которым он доверяет
* клиент генерирует сеансовый ключ для симметричного шифрования, зашифровывает его открытым ключом и передает серверу
* сервер расшифровывает сеансовый ключ с помощью приватного ключа

для того чтобы проверить валидность сертификата на ресурс полученного от сервера

необходимо проверить ЭЦП, которая есть внутри сертификата, данную подпись поставил один из центров сертификации, у каждого клиента есть список центров сертификации, которым он доверяет

1. HTML. Структура HTML-страницы. Каскадные таблицы стилей (CSS). Модель DOM. Пример.

**HTML** – язык гипертекстовой разметки, который может интерпретироваться браузерами

**Структура Html**

* тег html – определяет начало html документа
* тег head – в котором хранится мета информация об html документе, а также подключение дополнительных файлов, таких как css и html
* тег body – в котором храниться весь контент html документа

**CSS** – каскадная таблица стилей, которая описывает внешний вид html-документа

**Селектор** – css-правило, позволяющее обратиться к группе элементов, к которым будет применен стиль

**Стиль** – совокупность правил, определяющая внешний вид элемента

**DOM** – это модель представления браузером html-документа, у которой есть интерфейс для работы с этой моделью

document.querySelector(“.class”)

1. Протокол WebSockets: принципы работы и применения. Пример

**WebSocket** – это протокол, работающий на прикладном уровне поверх tcp, обеспечивает возможность обмена данными между клиентом и сервером через постоянной соединение, характеризуется дуплексным каналом связи

**Дуплексный канал связи** – канал, в котором в один момент времени данные могут идти в две стороны независимо друг от друга

**WebSocket API**

**Методы:**

* socket.send(data)
* socket.close([code], [reason])

**События:**

* open
* message
* error
* close

**Рукопожатие**

* клиент обращается к серверу с двумя заголовками по протоколу http  
  **Connection: Upgrade  
  Upgrade: WebSocket**
* если сервер поддерживает протокол ws, он отправляет ответ со статус кодом 101
* после чего устанавливается соединение по протоколу ws.

1. JavaScript. Основные стандарты. Типы данных. Программные структуры. Принцип применения. Пример.

**JavaScript** – это язык программирования с динамической типизацией. Применяется для решения различных задач, в частности для придания интерактивности веб-страницам.

**TypeScript** – это строготипизируемый язык программирования, который компилируется в JavaScript

**ECMAScript** – это стандарт, а JavaScript самая популярная его реализация  
новый стандарт выходит ежегодно и именуется ECMAScript2015 либо ES6

**Типы данных (7 примитивов и один ссылочный тип)**

Примитивы:

* number
* bigInt
* string
* null
* undefined
* symbol
* boolean

И один ссылочный тип – object

**Структуры данных**

* **Array**
* **Map**
* **Set**

**Применение**

JavaScript применяется для придания интерактивности веб-страницам

**Promise (промис)** – это механизм, который позволяет избавится от глубокой вложенности при выполнение асинхронных операций

Раньше для обработки результат асинхронной функции последним параметром передавалась callback-функция обработчик

Теперь у промиса можно построить цепочку обработки асинхронных функций благодаря методу **then**

**Понятие асинхронности**

Операция называется асинхронной, если ее выполнение осуществляется в две фазы:

* заявка на выполнение
* и получение результата

При этом участвуют два механизма

* первый механизм формирует заявку и ожидает результат
* второй механизм получает заявку от первого, исполняет ее и отправляет результат

В промежутке между заявкой и получение результата первый механизм может выполнять другие операции

TypeScript - это язык программирования, разработанный как надмножество JavaScript, что означает, что он предоставляет все возможности JavaScript и добавляет дополнительные функции и синтаксические конструкции. Вот некоторые преимущества TypeScript по сравнению с JavaScript:

1. Статическая типизация: Одно из основных преимуществ TypeScript - это статическая типизация. TypeScript позволяет определять типы переменных, параметров функций, возвращаемых значений и других элементов кода. Это помогает выявлять ошибки на этапе компиляции, а также предоставляет более надежную документацию и инструменты автодополнения кода.
2. Улучшенная разработка: Благодаря статической типизации и другим дополнительным возможностям TypeScript предлагает улучшенный опыт разработки. Редакторы кода, такие как Visual Studio Code, могут предлагать подсказки и предупреждения о потенциальных ошибках, облегчая процесс разработки и уменьшая количество ошибок в коде.
3. Обратная совместимость с JavaScript: TypeScript является надмножеством JavaScript, поэтому любой действительный код JavaScript также является действительным кодом TypeScript. Вы можете постепенно внедрять TypeScript в существующий проект, добавляя типы постепенно и получая все преимущества языка, сохраняя при этом совместимость с JavaScript.
4. Поддержка новых функций ECMAScript: TypeScript активно поддерживает новые функции и синтаксические конструкции из стандарта ECMAScript. Это означает, что вы можете использовать новые возможности JavaScript, даже если они еще не полностью поддерживаются в браузерах, благодаря компиляции TypeScript в старую версию JavaScript.
5. Лучшая поддержка инструментов и экосистемы: TypeScript имеет развитую экосистему инструментов, включая мощный компилятор, среды разработки с подсветкой синтаксиса и интеллектуальным анализом кода, а также множество сторонних библиотек и фреймворков, которые предоставляют дополнительные возможности для разработчиков.
6. Более безопасный код: Благодаря статической типизации и другим функциям TypeScript помогает предотвращать ошибки во время выполнения, такие как неправильное обращение к свойствам или методам объектов или передача неправильных аргументов функциям. Это повышает надежность кода и упрощает его поддержку.

В целом, TypeScript предоставляет разработчикам больше инструментов и функций для написания более безопасного, надежного и масштабируемого кода по сравнению с JavaScript. Однако выбор между TypeScript и JavaScript зависит от конкретных требований проекта и предпочтений разработчиков.

1. Методология Ajax. Структура Ajax-приложения, принципы разработки и применения. Пример.

**AJAX** – методология по построению интерактивных пользовательских интерфейсов, которая позволяет асинхронно подгружать данные без перезагрузки страницы по инициативе клиента

В основе Ajax лежит XMLHttpRequest

**XMLHttpRequest** – это встроенный в браузер объект, который даёт возможность делать HTTP-запросы к серверу без перезагрузки страницы.   
На каждый запрос создается свой экземпляр класса XMLHttpRequest

**fetch** – это надстройка над XMLHttpRequest

**Структура Ajax-приложения:**

* Html
* JavaScript
* DOM
* HTTP
* JSON
* XMLHttpRequest

1. Web-приложение. Архитектура web-приложения. Особенности реализации web-приложения. Web-сервер и web-клиент. Пример.

**Web-приложение** – клиент-серверное приложение, в котором клиент взаимодействует с сервером по протоколу HTTP

**Клиент-серверная архитектура** – это архитектура, которая подразумевает две компоненты клиент и сервер.   
**Клиент является инициатором соединения.**

1. ASP.NET: публикация ASP.NET-приложения, структура и параметры узла IIS, реальный и виртуальный каталоги, процедура настройки web-узла.

**ASP.NET** – это фреймворк, позволяющий разрабатывать web-приложения на платформе .NET Framework

**Платформа** – это набор библиотек и инструментов

**Типы ASP.NET приложений**

* http-handler
* mvc
* webApi
* asmx

**Публикация ASP.NET приложения**

Подразумевает под собой, настройку web-узла на web-сервере,   
для узла мы должны указать **протокол, port, ip**, так как на одном компьютере может быть несколько сетевых карт, а соответственно несколько ip адресов  
Также мы должны указать физический путь к web-приложению

**Реальный каталог** – физическое расположение приложения.

**Виртуальный каталог** – это псевдоним, под которым будет скрываться физическое расположение приложения

**Локальный виртуальный каталог** физически расположен на том же ПК

**Сетевой виртуальный каталог** физически расположен на удаленном ПК

1. ASP.NET: http-обработчики, порядок разработки, http-обработчик для взаимодействия с клиентом по протоколу WebSockets. Пример.

**http-обработчик** – это простейшее ASP.NET приложение  
Задача http-обработчика – принять запрос и сформировать ответ

Обработчики бывают двух типов:

* системный – входящий в состав asp.net
* А также обработчик asp.net приложения, так как любое asp.net приложение имеет свой http-обработчик

1. ASP.NET: ASMX-сервисы, WSDL, SOAP, прокси, порядок разработки, принципы применения. Пример.

**ASMX-сервис** – это web-сервис, построенный в соответствии с RPC взаимодействие происходит по протоколу SOAP

**Web-сервис** – это web-приложение, которое предназначено для использования другими web-приложениями

**RPC** – это технология, которая позволяет представить сервер в качестве объекта, у которого мы можем вызывать удаленные процедуры

**Особенности ASMX-сервиса**

* в качестве транспорта используется http (полудуплексный канал связи)
* одна точка доступа
* сервис построен в соответствии с RPC
* в качестве web-сервера может использоваться только IIS web-сервер

**WCF-сервис** – это web-сервис, который может быть построен в соответствии с RPC или REST

**Особенности WCF-сервиса**

* транспорт: http, tcp, named pipe, smtp (соответственно дуплексный, полудуплексный, симплексный и потоковый каналы связи)
* может быть несколько точек доступа (за счет разных транспортов)
* сервис может быть построен в соответствии с RPC или REST
* поддерживается self-hosting

**WCF-сервис**, представляет собой класс; этот класс не может существовать самостоятельно, а должен находиться под управлением некоторого процесса Windows, называемого хостовым процессом.

В качестве хоста может выступать консольное или графическое NET-приложение (автохостинг), IIS.

**WSDL** – это язык описания web-сервисов  
**Структура WSDL**

WSDL-документ можно разбить на три части:

* определение типов данных
* список операций
* способ, которым сообщение будет доставлено

**SOAP** – это протокол, по которому web-сервисы взаимодействуют друг с другом в формате xml  
**SOAP сообщение состоит из:**

*envelope {  
- header  
- body  
- fault (необязательный компонент)  
}*

**Proxy** – класс, упрощающий обращению к web-сервису  
**Задача proxy** – создать иллюзию локальной работы с сервисом  
proxy принимает параметры преобразует их в xml формат генерирует envelope, по сути формирует soap сообщение, отправляте его на сервер, затем принимает soap-сообщение ответа, распаковывает его и выдает ответ

**proxy создается на стороне клиента**

**proxy класс создается на основе WSDL**

на стороне сервера есть аналог proxy называемый **stub**

1. ASP.NET: MVC-приложение, структура MVC(R)-приложения, назначение основных компонентов приложения, маршрутизация. Пример.

**MVC-приложение** – это еще один тип asp.net приложений, который построен по архитектурному паттерну MVC

**MVC** – это архитектурный паттерн, который позволяет разделить логику приложения на три части:

* Модель
* Представление
* Контроллер

**Назначение основных компонентов приложения**

* Модель – отвечает за работу с данными, наиболее часто в качестве модели используется ORM Entity Framework
* Представление – отвечает за визуальное отображение данных
* Контроллер – обеспечивает связь между представлением и хранилищем данных

**Маршрутизация** – это процесс перенаправления HTTP-запроса на контроллерЗадача маршрутизатора – с помощью таблицы маршрутизации *(cама таблица заполняется при старте приложения в соответствии с указанными роутам)* найти подходящий класс контроллера, создать его экземпляр и вызвать необходимой action  
**Создание экземпляра и вызов action производится с помощью Reflection**

**Существует два способа маршрутизации**

* на основе паттерна
* на основе атрибутов

Маршрутизация с помощью атрибутов по умолчанию отключена. Включается она посредством вызова метода **MapMvcAttributeRoutes()**, который вызывается на объекте RouteCollection, передаваемом в качестве аргумента статическому методу RegisterRoutes() в RouterConfig:

**Контроллер –** класс, обеспечивающий связь между представлением и хранилищем данных

**Признаки контроллера**

* суффикс «Controller» в названии класса
* класс контроллера должен быть наследником класса Controller

**Жизненный цикл Контроллера**На каждый запрос создается новый контроллер

**Взаимодействие с моделью и представлением**

Существуют различные способы передачи данных из контроллера в представление:

* ViewData (? он это не упоминал)
* ViewBag
* Модель представления

**Представление**

**Жизненный цикл**

Razor Engine отрабатывает один раз при первом обращении к View и преобразует cshtml в объект

а при всех последующих обращениях к данному View будут вызываться методы ранее созданного объекта

Так работает до тех пор, как мы не перезагрузим сервер

**Модель**Наиболее часто в качестве модели используется ORM Entity Framework

**Entity Framework** – это одна из реализаций ORM

**ORM** – это механизм, который позволяет отобразить сущности бд в объекты используемого языка программирования

* базе данных соответствует - contextDB
* таблице - коллекция объктов
* строке таблицы - объект
* структуре таблицы – класс

**DI (внедрение зависимостей)** – это одна из реализаций IoC

**IoC** – это принцип по написанию слабосвязного кода, это возможно благодаря тому, что поток управления переходит от программиста к IoC контейнеру  
**IoC** – это принцип по написанию слабосвязного кода, при котором мы делегируем создание экземпляров зависимостей IoC контейнеру

**А для внедряемых объектов мы указываем их жизненный цикл**

* transient – создается новый объект на каждое обращение
* requset – создается новый объект на каждый запрос
* thread – создается новый объект, для нового потока
* singleton – создается один раз

**Способы внедрения**

* через конструктор
* через свойство (требует аннотации [Inject])
* через метод setter (требует аннотации [Inject])
* а также напрямую ручным способом

**WebAPI-сервисы** – это сервисы, построенные в соответствии архитектурой REST

**REST** – это архитектурный стиль по написанию распределенного приложения, которое имеет ресурсо-ориентированную архитектура

требования к REST приложению:

* взаимодействие клиент-сервер
* отсутствие состояния (сервер не сохраняет данные клиента между запросами)
* единообразный интерфейс: у каждого ресурса есть свой URI  
  а для указания действия над ресурсом используются http методы: GET, POST, PUT, DELETE, PATCH
* кеширование: клиент, а также промежуточный узлы могут выполнять кеширование ответов
* слоистость
* код по требованию

Контроллер WebApi должен наследоваться от класса ApiController

1. ASP.NET CORE: программная платформа, принципы работы, архитектура. Пример.

**ASP.NET CORE** – это фреймворк, который позволяет разрабатывать web-приложения на платформе .NET Core

**Платформа** – набор библиотек и инструментов

**Отличия asp.net framework от asp.net core**

* кроссплатформенность (возможно благодаря разработанным под каждую операционную систему CLR и FCL)
* поддерживается self-hosting (необязательно деплоить на IIS)
* open-source
* реализация соответствует интерфейсу OWIN
* имеется встроенная DI (директива inject может использоваться внутри cshtml для внедрения зависимостей)
* распространение пакетов происходит через пакетный менеджер nuget

**OWIN** – это интерфейс между web-сервером и web-приложением, позволяющие разрабатывать отдельно web-сервер и web-приложение  
представляет из себя конвеер, включающий в себя:

* host
* web-сервер
* middleware
* web-приложение

**web-сервер не может существовать самостоятельно**, а должен находиться под управлением некоторого процесса Windows, называемого хостовым процессом.

**Katana** – OWIN-совместимый host

**Хост** – процесс операционной системы управляющий жизненным циклом web-сервера  
**Kestrel** – web-сервер реализующий интерфейс OWIN

ASP.NET Core приложения могут работать под IIS (Windows) или под web-сервером **Kestrel**

**Middleware** – промежуточный слой, который обрабатывает запрос  
цепочка middleware представляет собой конвеер обработки, при отправке ответа формируется обратная петля, и middleware-ы могут изменять response  
каждый middleware может заблокировать дальнейшую обработку и отправить ответ

**По умолчанию asp.net core приложение не поддерживает доступ к статическим файлам**  
Для того чтобы это исправить необходимо в Startup классу в методе Configure воспользоваться методом UseDefaultFiles

**Session** – серверный объект, хранящий информацию о соединении с клиентом, создается при первом обращении (SessionID передается в куки для идентификация клиента)

**Куки (Cookies)** – данные, которые хранит клиент в виде ключ-значение, создается по инициативе сервера (заголовок Set-Cookie), пересылается клиентом (заголовок Cookie)

**при помощи директивы inherits мы можем переопределить базовый класс для представлений, но при этом все равно должен быть производным от RazorPage  
по умолчанию базовый класс razorPage**

***У IIS-сервера есть возможность загрузки новой версии приложения без его остановки  
при этом старые запросы будут обработаны старой версией приложения, а новые запросы будут обрабатываться свежей версией приложения  
после того как отключиться последний клиент старой версии приложения, данная версия удалится  
По сути IIS сервер позволяет вносить правки в hot-режиме (горячий способ обновления приложений)***

**CLR** – программа интерпретатор, является COM компонентом, имеет JIT компилятор

**FCL** – библиотека классов

**Каждый узел** – это отдельное web-приложение

**Объект router создается один раз**Маршрутизатор использует механизм рефлексии с помощью которого происходит поиск в dll нужного класса контроллера и создает объект этого класса, у которого вызывается подходящий метод.

При первом обращении view вызвается **Razor Engine**. После вызывается тот экземпляр, который сформировал **Razor Engine.**

**Identity** – это фреймворк, который позволяет произвести токен аутентификацию

**Переадресация**

**301 и 308 – постоянные перенаправления**

**302, 303 и 307 – временные перенаправления**

**307 и 308 не изменяют метод и тело запроса**