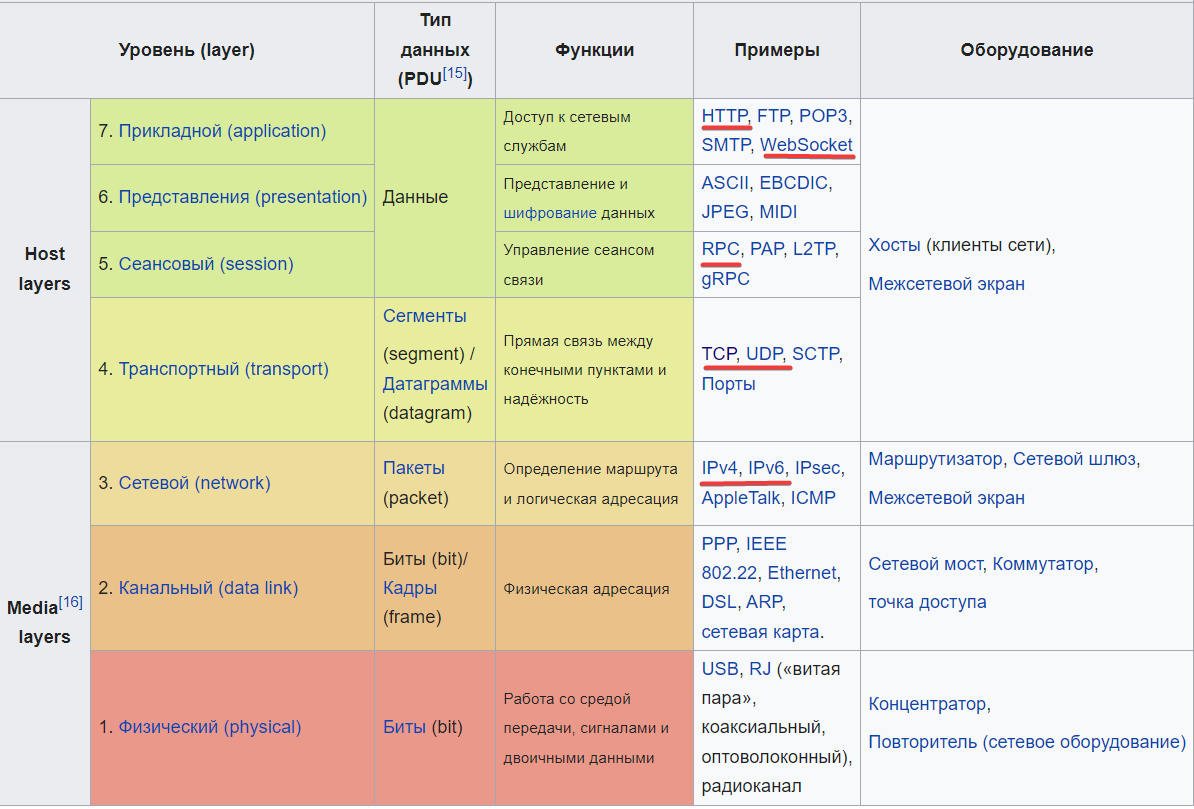
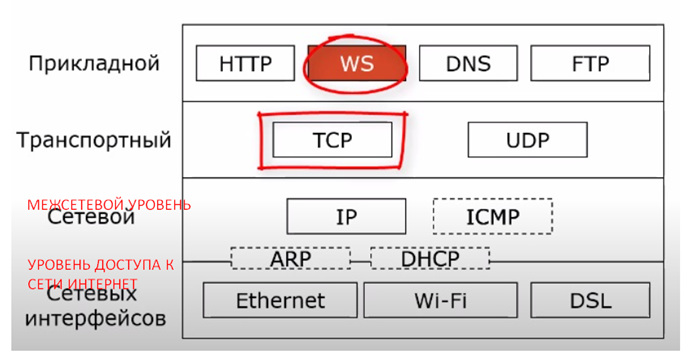
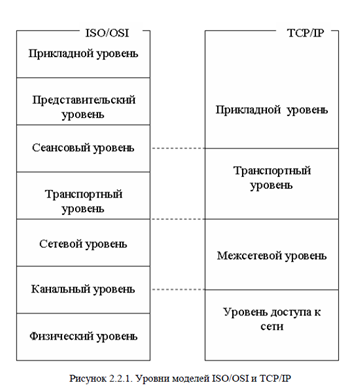
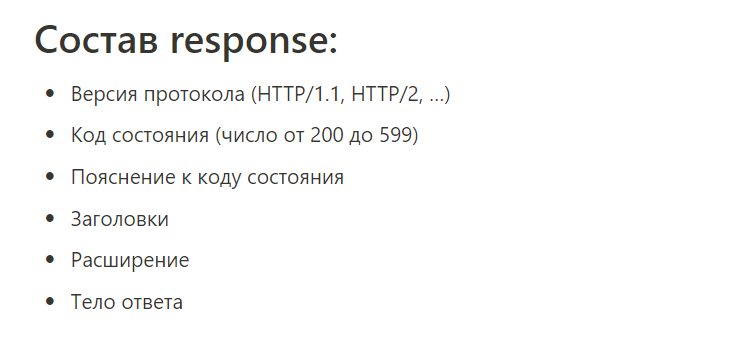
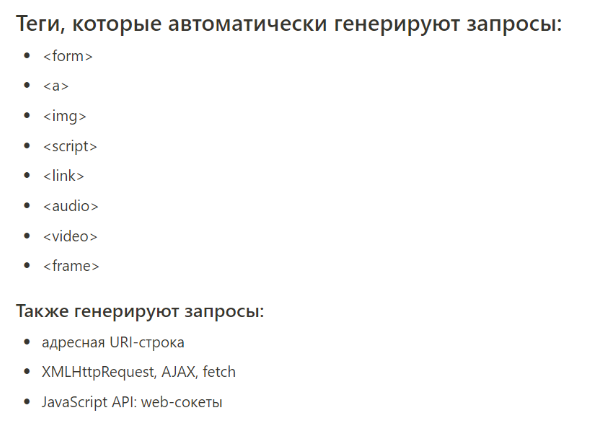
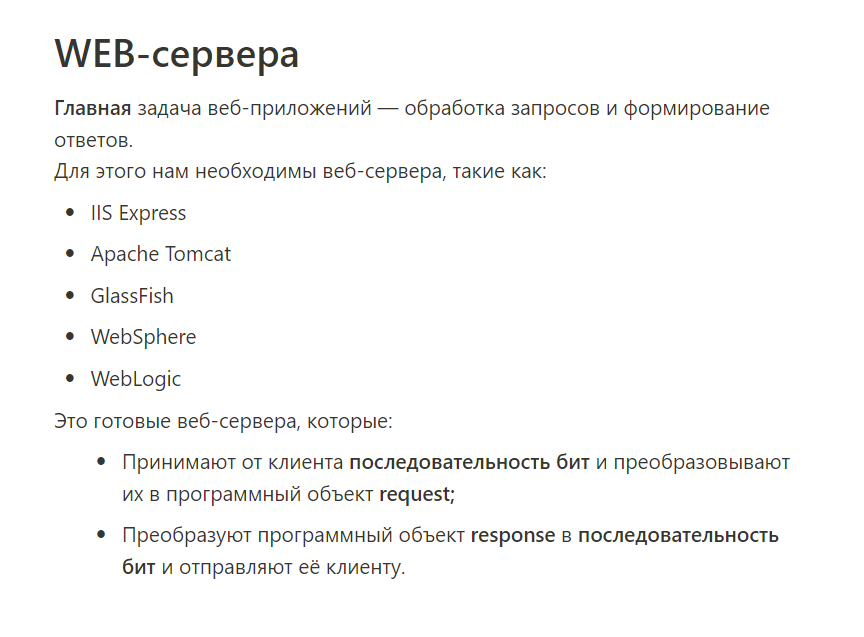
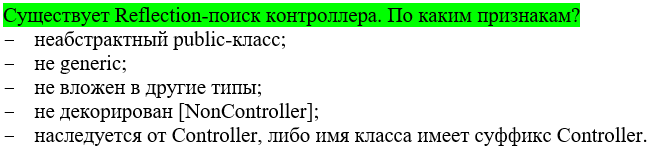
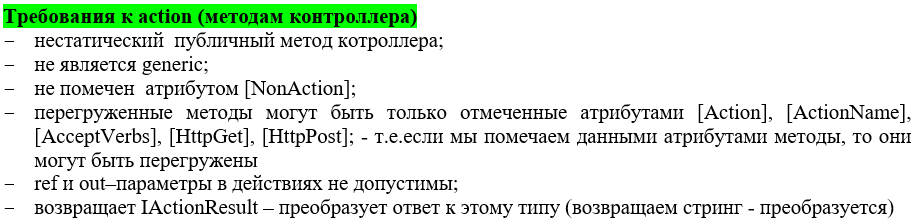
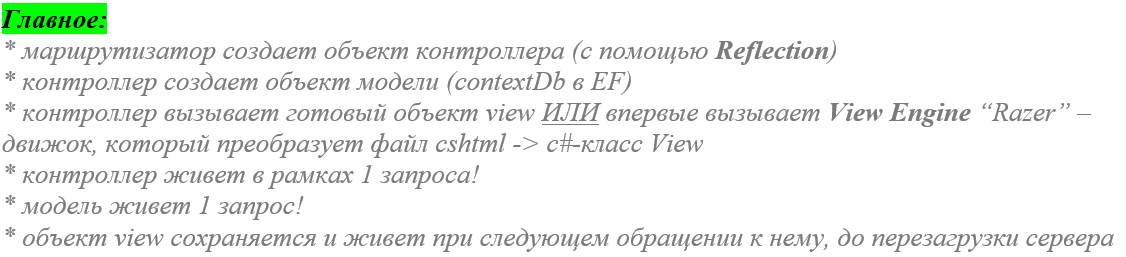
.

# SMELOV THE BEST

1. **Сервер** – программное обеспечение, которое принимает запросы.   
   (и, вероятнее всего, обрабатывает их).
2. Интернет – всемирная сеть, построенная на стеке протоколов TCP/IP.
3. **Интернет** – 4 компонента:
   1. Стек протоколов TCP-IP — это основа Интернета.
   2. Интернет службы (DNS, DHCP, SMTP, POP3 и т.д.).
   3. Документация в формате RFC и STD.
   4. Система организаций, которые поддерживают сеть (*в порядке убывания важности, как сказал Смелов*):
      * ISOC (развитие, внедрение и распространение новых интернет-технологий)
      * IETF (разрабатывает стандарты интернета в RFC)
      * IAB (занимается архитектурой)
      * ICANN (распространяет доменные имена)
      * IANA (регистрирует MIME)
      * W3C (WWW Consortium, некоммерческая организация для согласования стандартов) — согласовывают *всё*, что касается веб-программирования (HTTP, CSS, SVG, URI/URL, XML, PNG, JPEG, …). Также они изготавливают стандарты и регистрируют их в IETF.
4. **Интернет-служба** – это программа, система, предоставляющая услуги клиентам. Сервер + протокол. Стандартные серверы, которые прослушивают стандартные порты (*от 0 до 1024*).
5. **Веб-приложение** – клиент-серверное приложение, в котором клиент и сервер взаимодействуют с помощью протокола HTTP.
6. **HTTP** – *полудуплексный* протокол *прикладного* уровня.
7. Создатель HTTP и HTML – Тим Бёрнерс-Ли (Также он глава W3C).
8. Версии HTTP:
   1. *0.9* (Работает поверх TCP. Только гипертекст (нельзя передавать файлы). Нет заголовков, status codes, полного URL. Methods: GET)
   2. *1.0* (Есть кеширование, аутентификация, заголовки (e.g. Content-Type), можно передавать медиа (фото, видео, аудио), есть коды ответа.   
      Нет постоянного соединения. Methods: GET, HEAD, POST)
   3. *1.1* (Действующая версия. Имеет постоянное соединение (Connection: keep-alive), есть SSL/TLS, chunked responses, content negotiation.)
   4. *2.0* (Не распространен. Есть параллельные запросы в рамках одного соединения и сжатие одинаковых заголовков. *Бинарный*: (это значит, что в 2.0 всё изначально закодировано и сжато, а в 1.1 изначально всё в строковом значении, а только потом сжимается))
   5. *3.0* (Не распространен. Главная особенность – универсальный транспортный протокол QUIC, работающий поверх UDP. Замена TCP)  
      <https://habr.com/ru/companies/sbermarket/articles/703048/>  
      https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics\_of\_HTTP/Evolution\_of\_HTTP
9. Два типа сообщений – request и response.
10. Два типа абонентов – клиент и сервер. (клиент – всегда инициатор связи)
11. **HTTPS** (HTTP + SSL/TLS) – специальный протокол, обеспечивающий шифрование и аутентификацию.   
    SSL – Secure Socket Layer, был заменен в 2014 на TLS  
    TLS – Transport Layer Security
12. В каких версиях STD описаны URI и HTTP?  
     *URI* – STD 66  
     *HTTP* – RFC 2616 + RFC 7230-7235
13. Последние версии: HTML 5.3 (2018), CSS 3.
14. HTTP – порт *80*,  
     HTTPS – порт *443*.
15. **Интернет-ресурс** – *сущности*, которые имеют адрес в сети Интернет.  
     Бывают:  
     1) статические (html)  
     2) динамические (js)  
     3) полудинамические (html + js)  
     (адрес = IP + порт. например, 192.168.0.1:8000).  
     socket = IP + port (example: 192.168.0.1:8000)
16. Виды каналов связи:
    1. симплексный – передача данных только в одну сторону (радио)
    2. полудуплексный – обе стороны могут передавать и принимать данные, но не одновременно. Когда один говорит, второй только слушает (например, рация)
    3. дуплексный – данные передаются и принимаются в обе стороны одновременно (e.g. телефон)
17. 
18. 
19. **Типы заголовков:**
    1. General – общие для запросов и ответов (напр. Connection, Date)
    2. Request (напр. Accept, Cookie, Host, Authorization)
    3. Response (напр. Set-Cookie, Server)
    4. Entity – для сущностей в запросах и ответах (напр. Content-Type, Content-Length, Content-Encoding: все, что начинается с “Content”)
    5. User – заголовок, который не описан в протоколе HTTP   
       (должен начинаться с “x-header\_name”)
20. **Уровни TCP/IP:**  
     1) Прикладной   
     2) Транспортный   
     3) Межсетевой   
     4) Уровень доступа к сети
21. **Уровни ISO/OSI:** 1) Прикладной  
     2) Представительский  
     3) Сеансовый  
     4) Транспортный  
     5) Сетевой  
     6) Канальный  
     7) Физический
22. 
23. 
24. **Протокол** – набор согласованных правил, которые устанавливают порядок установки соединения и обмена данными между пользователями сети Интернет.
25. Важные для нас протоколы каждого уровня:
    1. Прикладной: HTTP, WebSocket, DNS, DHCP
    2. Транспортный: TCP, UDP, ***Сокеты***
    3. Сетевой: IP
26. **URI** (Uniform Resource Identifier, унифицированный айди ресурса) – строка, позволяющая идентифицировать некоторый ресурс.
27. **Основные методы запроса:**
    1. GET
    2. POST
    3. PUT
    4. DELETE
    5. HEAD
    6. OPTIONS
    7. CONNECT
    8. TRACE
    9. PATCH

*Также не забываем, что существует протокол WebDAV, который добавляет еще* ***7*** *методов запроса: PROPFIND, PROPPATCH, MKCOL, COPY, MOVE, LOCK, UNLOCK).*

1. **Stateless протокол** – который *не сохраняет* состояние   
   между запросом и ответом (например, HTTP или UDP).   
   Это значит, что запросы никак не связаны между собой, и данные из одного запроса в другой запрос никак нельзя получить.
2. **Stateful протокол** – который *сохраняет* состояние между запросом и  
    ответом (например, TCP).
3. **Способы сохранения состояния** (state):
   1. Cookie (родной для HTTP способ)
   2. Длинная переменная в URI, которая хранит состояние
   3. Скрытые поля в форме, в которых пересылается доп. информация
4. **Узел Интернет** – это устройство, имеющее IP-адрес и подключенное к сети Интернет (как правило, к сети Интернет-провайдера).  
    Каждый узел характеризуется своей программно-аппаратной платформой (аппаратура + ОС).
5. Клиент – это ПО, которое переносит большую часть функционала по обработке информации на сервер. В качестве клиента зачастую выступает веб-браузер, через который с сервером взаимодействуют конечные пользователи.
   1. Для отображения контента используется HTML5
   2. Инициирует соединение (отправляет запрос) через протокол HTTP
   3. Использует DOM модель для представления HTML-документа, а JavaScript – для доступа к содержимому HTML-документал
   4. Использует CSS (Cascading Style Sheets) для добавления стилей к html документу
   5. Web-engine – веб движок, который преобразует HTML-страницу во внутреннее представление веб-браузера в соответствии с моделью DOM. В хроме это Blink, написан на C++
6. **WebSocket** – протокол *прикладного* уровня, основанный на TCP,   
    использующий *дуплексный* канал связи.
7. **Как устанавливается соединение WebSocket?** 
   1. Клиент посылает обычный HTTP-запрос, называемый *рукопожатием*, с заголовками:  
       Connection: Upgrade  
       Upgrade: Websocket
   2. Сервер решает, устанавливать WebSocket-соединение или нет. Если да, то сервер пересылает те же заголовки с кодом ответа 101 Switching Protocols.
   3. HTTP-соединение заменяется на WebSocket-соединение, данные можно передавать.
8. **Кеширование** – процесс перемещения данныхиз медленно-действующей памятив более быстродействующую для повышения производительности.  
    Протокол HTTP *НЕ ПРЕДУСМАТРИВАЕТ* кеширование на стороне сервера, только на стороне клиента !!!  
    *На стороне клиента:* Для этого используется заголовок Cache-Control. Это обязывает клиента в соответствии с протоколом кешировать страницу. При следующем обращении (request), клиент будет брать данные из кэша.   
    *На стороне сервера:* нужно кешировать response, поэтому надо его куда-нибудь записать и, когда приходит request, то, если response не устарел, то клиенту можно отдать закешированный сохранённый response.
9. **Cookie:***Фрагмент данных*, который запоминается клиентом по указанию сервера. 1. Сервер отправляет заголовок *Set-Cookie* со значением Cookie внутри  
   2. Клиент обязан в последующих запросах отправлять заголовок *Cookie* Также не забываем, что в Cookie есть *path*, который указывает, на какие эндпоинты распространяется этот куки.
10. **Что такое Redirect? Как он работает?**  
     Клиент отправляет запрос на сервер на доступ к ресурсу, который был перемещён. Сервер отправляет клиенту заголовок *Location*, в котором хранится URL, на который надо редиректнуться, а также один из статус-кодов: 301, 302, 307 или 308. https://developer.mozilla.org/en-US/docs/Web/HTTP/Redirections
    1. Temporary Redirect – ***302 и 307***
    2. Permanent Redirect – ***301 и 308***
    3. Special Redirect - ***300, 304***
11. **В чем разница между 301, 302, 307 и 308 status codes?**  
     Во-первых, 301 и 302 считаются устаревшими. В спецификации HTTP/1.1 им на замену пришли аналогичные 307 и 308.   
     Вся разница в том, что 301 и 302 могли изменить метод запроса: например, делается запрос *POST* /api/users, возвращается 302 status code, и делается новый запрос *GET* /api/users. В 307 и 308 такой проблемы нет, и метод запроса всегда будет сохраняться при редиректе.
12. Как происходит адресация на канальном уровне?  
     С помощью MAC-адреса (48-бит).
13. Что значит заголовок Content-Type?  
     Это MIME-тип ресурса в request body или response body. *(Entity header)*
14. Какие методы запросов можно отправлять через тег <form>?  
     Только GET и POST.
15. **Middleware** (или фильтр):  
     Это конвейер обработки запросов и ответов.
16. **Фильтр** – препроцессор запроса, предназначенный для предварительной обработки запроса. Является серверным объектом. Если есть несколько фильтров, то они могут “перефутболивать” запрос друг другу.
17. **Как пересылаются данные через форму?***В виде пар ключ:значение.*  
    **Дополнительная инфа:**   
    У тега <form> задается атрибут enctype, равный либо   
     x-www-form-urlencoded, либо multipart/form-data, либо text/plain.
    1. *x-www-form-urlencoded:*Данные похожи на то, как кодируются параметры в URL. Отправляются, как пары ключ-значение. Эти пары разделяются через «&», пробелы заменяются на «+», а русские символы заменяются их шестнадцатеричным представлением (%D0%90%D0%BD%D1%8F). Ключи берутся из атрибута name, а значения – из атрибута value в теге input. (<input name=”someKey” value=”someValue”>).   
       Как это выглядит на практике: text1=hello+world&text2=a%CF%89b&file=a.txt
    2. *multipart/form-data:*  
       Используется в основном для пересылки файлов через форму. Данные отправляются в теле запроса и разделяются некоторой границей – boundary.
    3. *text/plain:* Не используется.
18. **Session (сессия, сеанс)** – серверный объект, хранящийся на стороне сервера и имеющий идентификатор Session ID (**128**-битное число).   
    Задача сессии – *идентифицировать серию запросов. (сохранить состояние)*  
    Необходима для последовательных запросов, хранения промежуточных данных, e.g. не спрашивать у клиента логин и пароль при каждом запросе.   
     Относится к ***сеансовому*** уровню OSI/ISO: на этом уровне выполняется шифрование, и генерируется сеансовый ключ (Session ID), который актуален в течение всей сессии.
19. **Когда и как сессия создается и завершается?** 
    1. Сервер создает сессию, когда на него приходит первый request:
       * без заголовка Cookie;
       * в заголовке Cookie не записан Session ID;
       * в заголовке Cookie записан неправильный или несуществующий Session ID.
    2. Сервер создаёт объект сессии, генерирует Session ID, и отправляет клиенту заголовок *Set-Cookie* со значением Session ID;
    3. Клиент, получивший заголовок *Set-Cookie*, обязан запомнить значение Session ID;
    4. Во все последующие запросы клиент будет добавлять заголовок *Cookie* со значением Session ID;
    5. Сам сервер никогда не отправляет заголовок *Cookie*, он только обрабатывает их от клиента.
20. **Session Timeout** – это максимальное время между двумя запросами.   
    Если время превышено, то сессия будет уничтожена, и будет создана новая.
21. **Три уровня памяти:**
    1. Запрос — хранится в рамках одного запроса (пока не пришел ответ);
    2. Сессия — существует в рамках одной сессии;
    3. Контекст — в рамках контекста; может существовать условно бесконечно (пока работает сервер), можно передавать данные между сессиями. *Создается на основании config-файла.*
22. Контекст – создается на основе config-файла при старте приложения.
23. Как сохранить данные между сессиями?  
     1. В объекте process (Express)  
     2. HttpContext (ASP.NET, Java)  
     То есть, данные между сессиями храним данные в *контексте*.
24. Где можно хранить данные в течение одного запроса?  
     1. ViewBag/ViewData  
     2. В объекте запроса (там есть хранилище, можно как в JS к свойству объекта обратиться и записать внутрь данные)
25. Кроссплатформенное приложение – это приложение, способное работать на более чем одной программно-аппаратной (аппаратура + операционная система) платформе.
26. **Класс** – шаблон создания объекта.
27. **Объект** – специализированный объект времени исполнения.   
     Также объект – это экземпляр класса.
28. **Интерфейс** – поименованный набор сигнатур*.*  
    (У смелова понятие “поименованный набор *методов*”, но в интерфейсе могут быть методы, поля, свойства и т.д., поэтому мы говорим “поименованный набор сигнатур”)
29. **Платформа** – совокупность инструментов для разработки и библиотек   
     (ИЛИ совокупность инструментов и среды исполнения)
30. Рефлексия – выявления типа во время RunTime, метаданные. Применяется в Router для маппинга маршрутов, поиска классов контроллеров в сборке и handler методом рефлексии.
31. AJAX – методология программирования, построенная на асинхронных запросах. Эта методология подразумевает такой подход к построению динамических приложений, при которых не осуществляется полная перезагрузка HTML-страниц. *(XmlHttpRequest, fetch)*
32. **CLI** **(Common Language Infrastructure)** – спецификация общеязыковой инфраструктуры. Определяет архитектуру исполнительной системы и набор предоставляемых сервисов. Состоит из шести пунктов:
    1. Концепция и архитектура:
       * CTS (Common Type System)
       * VES (Virtual Execution System)
       * CLS (Common Language Specification)
    2. Метаданные
    3. Инструкции CIL (Common Intermediate Language)
    4. Библиотеки:
       * BCL (Base Class Library)
    5. Формат взаимодействия с отладчиком
    6. Приложения
33. **CLR (Common Language Runtime)** – общеязыковая среда исполнения – виртуальная машина, на которой исполняются все приложения, работающие в среде .NET. Является реализацией концепции VES от компании Microsoft. Содержит в себе JIT-компилятор(Just-In-Time).
34. *VES* (Virtual Execution System) — концепция виртуальной компиляции и запуска. VES – это только идея, концепция, спецификация, у которой есть конкретные реализации. Реализация VES на .NET – это CLR, на Java – это виртуальная Java-машина (JVM) и т.д.
35. *FCL* (Framework Class Library) — по аналогии, в то время как BCL – это просто библиотека стандартных (Basic) классов, то FCL – это конкретная реализация BCL для .NET Framework.   
     Можно рассматривать, как API CLR.
36. *MSIL* (Microsoft IL) – реализация CIL (Common Intermediate Language) компанией Microsoft.
37. *NET Framework* – программная технология программирования.   
    Это совокупность: CLR + FCL. Является реализацией CLI-спецификации.
38. **ASP.NET** – набор технологий (и библиотек) для создания Web-приложений и Web-сервисов от компании Microsoft на основе платформы .NET Framework.
39. **Типы приложений ASP.NET** (т. е. какие проекты можно создать):
    1. HTTP-handler
    2. Web Form
    3. MVC
    4. Web API
    5. ASMX-сервисы
    6. WCF-сервисы (отдельно).
40. Http-handler – это простейшее приложение ASP.NET. Представлен классом, который реализует интерфейс *IHttpHandler* и имеет свойство *IsReusable* и метод *ProcessRequest(HttpContext httpContext)*. Внутри объекта HttpContext хранятся request, response и т.д.
41. **IIS** – Internet Information Server – набор серверов для Internet-служб компании Microsoft. Поддерживает протоколы: HTTP, HTTPS, FTP, SMTP, POP3.
42. Режимы публикации приложения:
    1. ручной
    2. автоматический (через VS Express или IIS)
43. **Параметры при публикации IIS:**
    1. имя сайта
    2. физический путь
    3. протокол, IP и port
44. **Model-View-Controller** – паттерн проектирования, в котором приложение состоит из 4 взаимодействующих компонентов: модель, представление, контроллер, роутер. *Они относительно независимы.*Каждый компонент имеет свою зону ответственности:
    1. модель – данные;
    2. представление – динамическое формирование разметки для отправки клиенту;
    3. контроллер – обработка запроса, формирование экземпляра модели, вызов Razor Engine;
    4. роутер – маппинг URI.
45. **Какие требования должен соблюдать MVC контроллер?**  
     1) должен иметь в названии слово “Controller”  
     2) должен наследоваться от класса Controller (не обязательно)
46. Разница между классами Controller и ControllerBase:  
     Controller наследуется от ControllerBase и добавляет функционал для поддержки представлений ( *ViewResult View(object model)* ). Таким образом, в MVC надо наследоваться от Controller, а в Web API – от ControllerBase.
47. **Как контроллер взаимодействует с моделью?**  
     *В рамках контроллера создается объект модели.*
48. Типы моделей:
    1. MVC-модель *(модель Action)* – модель из набора параметров, которые приходят на action в контроллере, например Get([FromBody] User user)
    2. DB-модель *(модель данных)* – модель, которая создается с помощью Entity Framework (или другой ORM) и соответствует сущностям базы данных.
    3. View-модель – то, что передается в параметрах в данном коде:   
       *return View(object model)*. Таким образом, view-моделью может быть что угодно – от объекта или коллекции до строкового литерала.
49. Строго типизированные представления – представления, привязанные к некоторой модели. В них используется ключевое слово @model в начале представления, которое говорит, к какой модели она привязана.
50. Способы передачи данных на View:  
     1) ViewData (глобальный key-value словарь типа ViewData[“Foo”])   
     2) ViewBag (как ViewData, но использует свойства типа ViewBag.Foo)  
     3) ViewModel (специальный класс для передачи данных)  
     4) Передача модели через конструктор– для адекватных людей
51. Где регистрируется таблица маршрутов?   
     В файле Global.asax.
52. По какому принципу Controller находит View?
    1. сначала ищет в Views/Controller\_name/Action\_name.cshtml
    2. потом в Views/Shared
53. **Веб-сервис** – веб-приложение, предоставляющее программный интерфейс для других веб-приложений.
54. Что позволяет сделать tag-helper в ASP.NET Core?  
     Позволяет создать собственный тег, а также редактировать существующий, добавляя в него контент, который будет отображаться.
55. В чем польза HATEOAS?  
     *Даёт возможность отвязать клиента от ссылок сервисов.*   
     Например, есть страница, на которой есть таблица со списком студентов, и кнопки “оценки” и “рейтинг” рядом с каждым из них. Без HATEOAS пришлось бы “зашивать” (хардкодить) ссылки на оценки и рейтинг внутрь кода. При наличии HATEOAS, можно получить ссылки по их названию.  
     По большей части необходимо только в том случае, если инфа получается из нескольких ресурсов.
56. Структура SOAP-сообщения:
    1. Envelope – корневой элемент, определяет сообщение и пространство имен, используемое в документе
    2. Header – содержит атрибуты сообщения, например: информация о безопасности или о сетевой маршрутизации
    3. Body – содержит сообщение, которым обмениваются приложения
    4. Fault – необязательный элемент, который предоставляет информацию об ошибках, которые произошли при обработке сообщений
57. 
58. 
59. 
60. 
61. SMX – веб-приложение, является частым случаем DAP.CV, один из типов приложения
62. Все веб-сервисы можно разбить 2 группы:

1) Интерфейс RPC (в виде программного объекта, у которого есть методы, которые можно вызывать удаленно)

2) Интерфейс REST (методы запросы)

1. ASMX – это веб-сервис, работает на SOAP, встраивается в HTTP протокол, представляет из себя xml сообщение, он может присутствует в body
2. WSDL – ? для сервис-ориентированную архитектуру для приложений
3. WSDL? – описывает сервис, информация о типах данных, типы сообщений, какие конечные точки процедур и их удаленные имена, есть информация о связи транспорта, который переносит сообщение, а также физический адрес, каждый программист может вызвать удаленную процедуру на другом сервисе
4. MVC – графический интерфейс поддерживаю,
5. WEB-API – это сервис, которые поддерживаю интерфейс REST
6. DAP.CV – для разработки приложеней сервис-ориентированных архитектур, могут работать на разных протоколах, STMP, HTTP, NamedPipe
7. Handler, MVC, ASMX, WEB-API, DAP.CV