* поясните понятие «программная платформа»;

Набор библиотек и инструментов для разработки приложений на основе этих библиотек.

* что такое ASP.NET?

Программная платформа для разработки веб-приложений, состоящее из клиента и сервера, взаимодействующее по протоколу http. Обеспечивается кроссплатформенность.

* что такое http-обработчик?

это компонент, который обрабатывает HTTP-запросы и генерирует HTTP-ответы. В контексте веб-разработки, HTTP-обработчик часто относится к функции или методу, которое вызывается, когда определенный URL или маршрут запрашивается клиентом.

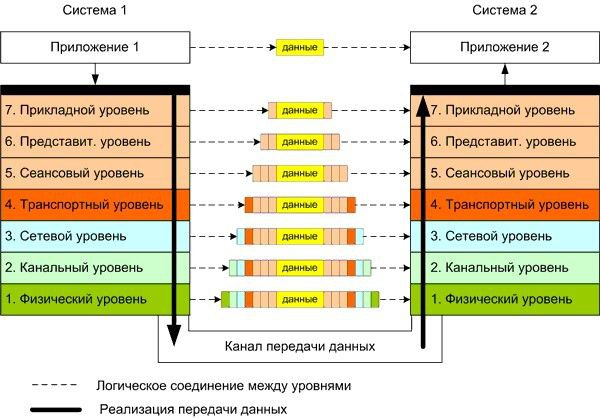
* каким образом запросы маршрутизируются к http-обработчику?

обычно осуществляется с помощью маршрутизатора, который является частью веб-сервера или веб-фреймворка. Маршрутизатор определяет, какой обработчик должен быть вызван на основе URL-запроса, метода HTTP (GET, POST, PUT, DELETE и т.д.) и, возможно, других аспектов запроса. Маршруты обычно определяются в коде приложения или конфигурации веб-сервера.

**Интернет:** Всемирная компьютерная сеть, построенная на основе стека протоколов TCP/IP.







**Структура:** службы и услуги.

**Internet-служба**: другое название Internet-сервис, один из видов Internet-ресурса, имеющий специальное назначение (DNS, WWW, E-mail, FTP, ICQ, Telnet).

**Internet-ресурс:** сущность в сети Internet, имеющая адрес (опубликованная в Internet сущность).

**Службы (сервисы) Интернет**:

* DNS – система для преобразования доменных имен в IP-адрес,
* E-mail (STMP, POP3, IMAP);
* IRC (обмен сообщений в реальном времени);
* FTP;
* Telnet (управление удаленным компьютером в терминальном режиме);
* WWW.

**ISOC:** Internet Society – международная организация, занимающаяся развитием сети Internet. ISOC владелец RFC-стандартов. ISOC обеспечивает правовую поддержку и финансирует все другие организации, связанные с деятельностью Internet.

**IETF:** Internet Engineering Task Force - рабочая группа проектирования Internet. Публикует RFC (Request for). Задачи IETF описаны в RFC 4677.

**RFC:** Жизненный цикл описан в RFC 2026.

**Этапы ЖЦ:**

* Интернет-проект (Internet Draft): Предложение без официального статуса, удаляется через 6 месяцев после последнего изменения.
* Предложенный стандарт (Proposed Standard): Проект признан успешным, получает номер RFC. Наличие программной реализации желательно, но не обязательно.
* Проект стандарта (Draft Standard): Принят сообществом, существуют две независимые совместимые реализации. Считается стабильным и рекомендуемым для реализации, допускаются мелкие правки.
* Стандарт Интернета (Internet Standard): Высший уровень, с успешным опытом применения. Имеет собственную нумерацию STD, список представлен в документе STD 1 (RFC 5000). Достигли этого уровня только несколько десятков из более чем трех тысяч RFC.
* Исторические документы (Historic): Устаревшие RFC, замененные новыми версиями или вышедшие из употребления.

RFC может содержать не только стандарты, но и концепции (Experimental – результаты экспериментов, Informational – информационные, Best Current Practice – опыт применения).

**IAB:** Internet Architecture Board - совет по архитектуре Internet, одна из комиссий IETF, имеет консультативный статус при ISOC.

**ICANN: Internet Corporation for Assigned Names and Numbers** – корпорация по управлению доменными именами и IP-адресами.

**IANA: Internet Assigned Numbers Authority –** Администрация адресного пространства Internet. Под контролем ICANN. Кроме того, регистрирует типы данных **MIME**.

**MIME**: **Multipurpose Internet Mail Extensions** - многоцелевые расширения Internet-почты. Используется и как стандарт кодирования Internet-сообщений.

**W3C: World Wide Web Consortium –** организация разрабатывающая и внедряющая web-стандарты.

**WHATWG:**Web Hypertext Application Technology Working Group – рабочая группа по развитию web, отделился от W3C, разработка HTML5 и Browser API.

**Программирование в Internet:** разработка распределенного (сетевого) приложения, представляющего собой один или несколько связанных Internet-ресурсов. Сайт – частный случай приложения, Вычислительное облако – частный случай приложения.

**Технологии программирования в Internet:**

* LAMP (Linux, Apache, MySQL, PHP/Perl, кроссплатформенная технология);
* **Java EE (кроссплатформенная технология на основе Java);**
* **ASP.NET** (ASP.NET CORE – кроссплатформенная версия);
* **NODE.JS** (кроссплатформенная технология, на основе JavaScript);
* Ruby on Rails (кроссплатформенная технология на основе Ruby и фреймворка MVC для web-разработки);
* Python Django (кроссплатформенная технология на основе Python и фреймворка MVC для web-разработки).

**Программирование в Internet:** в большинстве случаев длясвязи между компонентами приложения применяются протоколы HTTP, WebSocket, SMTP, POP3, IMAP, WebDav.

**Web-приложение:** клиент-серверное приложение, применяющее для обмена данными протокол HTTP; может быть просто web-приложением (HTML+HTTP) или web-службой (API, HTTP-транспорт, формат XML, JSON).

**Архитектура веб-приложения:** клиент и сервер.

**Клиент-серверная архитектура -** модель организации вычислительных систем, в которой задачи распределены между клиентами и серверами.

**Режимы передачи данных:**

Симплексный режим: В симплексном режиме отправитель может отправлять данные, но не может их получать.

Дуплекс: Двусторонняя связь, клиент и сервер могут отправлять и получать данные одновременно.

Полудуплекс: Односторонняя связь, данные могут передаваться в одном направлении в каждый момент времени.

СПОСОБ ПЕРЕДАЧИ?? В душе не ебу что там требуется, возможно правильным будет перечислить протоколы:

* HTTP (Hypertext Transfer Protocol): Протокол передачи гипертекста, используемый для передачи данных в вебе.
* HTTPS (Hypertext Transfer Protocol Secure): Защищенная версия HTTP с использованием шифрования SSL/TLS для безопасной передачи данных.
* FTP (File Transfer Protocol): Протокол передачи файлов, используемый для обмена файлами между компьютерами в сети.
* SMTP (Simple Mail Transfer Protocol): Протокол передачи почты, применяемый для отправки электронной почты между серверами электронной почты.
* POP3 (Post Office Protocol 3): Протокол получения почты, используемый для извлечения сообщений с почтового сервера.
* IMAP (Internet Message Access Protocol): Протокол доступа к почтовому ящику в Интернете, предоставляющий более расширенные функции, чем POP3.
* TCP (Transmission Control Protocol): Протокол управления передачей, обеспечивающий надежную и упорядоченную передачу данных в сети.
* UDP (User Datagram Protocol): Протокол пользовательских датаграмм, предоставляющий передачу данных без гарантии доставки или упорядочивания.
* DNS (Domain Name System): Протокол системы доменных имен, используемый для преобразования доменных имен в IP-адреса.
* SSH (Secure Shell): Протокол для безопасной удаленной работы с устройствами через зашифрованное соединение.
* Telnet: Протокол удаленного доступа, использующий небезопасное текстовое соединение.
* SNMP (Simple Network Management Protocol): Протокол для управления и мониторинга сетевых устройств.
* ICMP (Internet Control Message Protocol): Протокол для передачи сообщений об ошибках и управляющей информации в сети IP.
* WebSockets: Протокол для полнодуплексной связи между клиентом и сервером через веб.
* MQTT (Message Queuing Telemetry Transport): Протокол для передачи сообщений между устройствами в условиях низкой пропускной способности сети.

**TCP/IP (Transmission Control Protocol/Internet Protocol)** — это набор протоколов, обеспечивающий передачу данных в интернете и локальных сетях. Он состоит из нескольких уровней: IP отвечает за адресацию и маршрутизацию пакетов, TCP обеспечивает надежную, ориентированную на соединение передачу данных, а другие протоколы, такие как UDP, HTTP и FTP, выполняют специфические функции для различных приложений. TCP/IP масштабируем и универсален, что делает его основой современных сетевых коммуникаций.

**HTTP (Hypertext Transfer Protocol):** Протокол передачи гипертекста, используемый для передачи данных в вебе.

**HTTP:** основные свойства

* версии HTTP/1.1 – действующий (текстовый), HTTP/2 черновой (не распространен, бинарный);
* два типа абонентов: клиент и сервер;
* два типа сообщений: request и response;
* от клиента к серверу – request;
* от сервера к клиенту – response;
* на один request всегда один response, иначе ошибка;
* одному response всегда один request, иначе ошибка;
* TCP-порты: 80, 443;
* для адресации используется URI или URN;
* стейтлесс-протокол (взаимодействие не зависит от предыдущих);

**Запрос:** серверный объект, который образуется в результате обработки сервером http-запроса, поступающего от клиента и передается серверному программному коду для обработки. Содержит: всю информацию из http-запроса: метод, коллекция заголовков, коллекция параметров, поток данных. Обычно объект Request предоставляет возможность хранить данные в формате ключ/значение.

**Ответ:** серверныйобъект, который автоматически формируется сервером, при получении http-запроса (одновременно с объектом Request), заполняется данными серверными программным кодом, преобразуется в http-ответ и отправляется клиенту. Содержит: всю информацию, которая должна быть помещена в http-ответ: статус, коллекция заголовков, поток данных.

**Структура запроса:**

* метод;
* URI;
* версия протокола (HTTP/1.1);
* заголовки (пары: имя/заголовок);
* параметры (пары: имя/заголовок);
* расширение.

**Структура ответа:**

* версия протокола (HTTP/1.1);
* код состояния (1xx, 2xx, 3xx, 4xx, 5xx);
* пояснение к коду состояния;
* заголовки (пары: имя/заголовок);
* расширение.

**HTTPS** — это защищенная версия HTTP, использующая SSL/TLS для шифрования данных, что обеспечивает защиту конфиденциальности и целостности информации.

**Методы HTTP:**

* + - 1. **GET**: Запрос данных с сервера. Используется для получения информации.
      2. **POST**: Отправка данных на сервер. Обычно используется для создания новых ресурсов.
      3. **PUT**: Обновление существующих данных на сервере. Заменяет весь ресурс.
      4. **DELETE**: Удаление данных с сервера.
      5. **PATCH**: Частичное обновление существующего ресурса.
      6. **OPTIONS**: Запрос информации о доступных методах и параметрах для конкретного ресурса.
      7. **HEAD**: Запрос заголовков ответа без тела. Полезен для проверки доступности ресурса.
      8. **TRACE**: Отладочный метод, который возвращает запрос, полученный сервером, что позволяет проследить за маршрутом.
      9. **CONNECT:** используется для установления туннельного соединения с сервером через прокси.

**Типы заголовков:**

* **General**: общие заголовки, используются в запросах и ответах;
* **Request**: используются только в запросах;
* **Response:** используются только в ответах;
* **Entity**: для сущности в ответах и запросах.
* **Пользовательский**

**Response: Код состояния:**

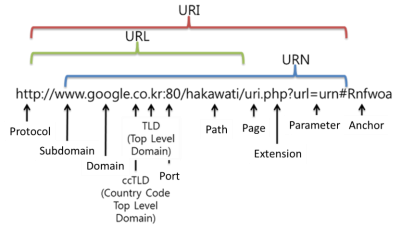
* **1xx**: информационные сообщения;
* **2xx**: успешный ответ;
* **3xx**: переадресация;
* **4xx**: ошибка клиента;
* **5xx**: ошибка сервера.

**URI: Uniform Resource Identifier** – унифицированный идентификатор ресурса (документ, изображение, файл, служба, электронная почта).

**URL: Uniform Resource Location** - унифицированный локатор ресурса, содержащий местонахождение ресурса и способ обращения (протокол) к ресурса, описывает множество URI.

**URN: Uniform Resource Name** -унифицированное имя ресурса – URI, имя ресурса, не содержащее месторасположение и способ доступа к ресурсу. В будущем URN должен заменить URL (для решения проблем с перемещением ресурсов в Internet).

**URI, URL, URN** –рекомендуется использовать термин URI**.**



**PURPL**: **Persistent Uniform Resource Locator** – постоянный унифицированный локатор ресурса.

**SSL/TLS Handshake** – «рукопожатие» между сервером и клиентом. Проще говоря – идентификация друг друга. Происходит во время HTTPS-соединения внутри зашифрованного туннеля SSL/TLS, который гарантирует безопасность как серверу, так и клиенту. После успешной идентификации генерируется секретный сеансовый ключ, который обеспечивает защищенную связь – он служит одновременно как для шифрования, так и для дешифрования передаваемых данных.

**Как происходит хэндшейк:**

Если представить SSL-рукопожатие как диалог между сервером и клиентом, то процесс будет выглядеть следующим образом:

1. Клиент обращается к серверу с просьбой установить безопасное соединение и предлагает набор шифров, которые «понимает», а также совместимую версию SSL/TLS.
2. Сервер проверяет присланный шифронабор, сравнивает со своим, и отсылает ответ клиенту с файлом сертификата и открытым ключом.
3. Клиент проверяет сертификат и, если всё в порядке, предлагает проверить закрытый ключ. Для этого он его генерирует и шифрует общий секретный ключ с помощью присланного ранее открытого ключа сервера.
4. Сервер принимает ключ, проверяет его своим закрытым ключом. Далее он создает главный секрет, который и будет использоваться для шифрования обмениваемой информации.

После этого клиент отправляет серверу тестовое сообщение, зашифрованное по продуманному ранее методу, а тот его расшифровывает и анализирует. На этом SSL/TLS Handshake завершается, и клиент с сервером могут спокойно обмениваться информацией дальше.

Если сеанс будет завершен, и через какое-то время клиент снова обратится к серверу, проходить заново процедуру «рукопожатия» не потребуется – все сгенерированные ранее данные и главный секрет сохранят свою актуальность. Весь этот процесс занимает считанные секунды и проходит абсолютно незаметно для пользователя.

**Сессия (Session):** серверный объект, хранящий информацию о соединении с клиентом, создается при первом обращении.

**ЖЦ сессии:** **timeout** (равен 10 – 30 минутам) – максимальное время между запросами клиента. Если **timeout** превышен, то Session разрушается и при следующем запросе создается новый экземпляр. Каждая сессия имеет собственный идентификатор (**Session ID**, 16 или более байт). Каждый Requestпринадлежит, какой-то сессии (имеет ссылку на объект Session или содержит Session ID). Обычно объект Session предоставляет приложению возможность хранить данные в формате ключ/значение.

**Cookie**: фрагмент данных, оправленный web-сервером и хранимый web-клиентом. Используется для аутентификации, хранения пользовательских предпочтений, статистики, **информации о сеансе** (обычно Session ID). Обычно имеет имя, содержащее URL, может иметь срок действия. Для создания и пересылки Cookie применяются заголовки.

**Кэш (Cache):** серверный объект, предназначенный для временного хранения данных с целью ускорения выполнения запроса. Кэширование – процессы записи и извлечения данных в/из Cache.

**Фильтр** — это серверный объект, который служит препроцессором запроса, предназначенный для предварительной обработки объекта Request.

**Слушатели событий (Lister):** серверные объекты – для обработки событий жизненного цикла web-приложения.

**Синхронный http-запрос:** запрос, при котором поток, выдавший http-запрос, блокируется до поступления запроса.

**Асинхронные запросы:** запрос, при котором поток, выдавший http-запрос, не блокируется до поступления запроса; для обработки ответа применяется функция обратного вызова.

**AJAX:** Asynchronous JavaScript and XML – асинхронный JavaScript and XML – методология (подход) построения динамических приложений, при которых не осуществляется полная перезагрузка html-страниц. AJAX: XMLHTTPRequest, DOM, формат: XML и JSON.

**Конвейер обработки запросов:**

Обработка запросов в ASP.NET Core устроена по **принципу конвейера.**

**Middleware** (промежуточное или связующее программное обеспечение) — это компонент (фрагмент кода в конвейере приложения), используемый для обработки запросов и ответов.

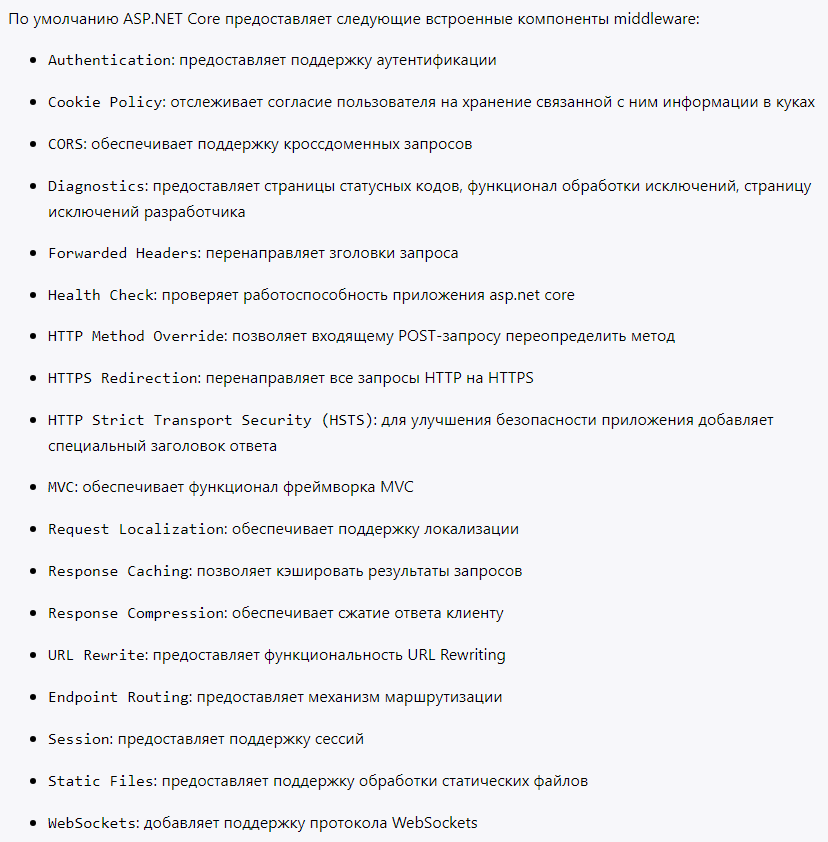
Компоненты middleware конфигурируются с помощью методов расширений Run, Map и Use объекта IApplicationBuilder, который передается в метод Configure() класса Startup.

Метод Configure выстраивает конвейер обработки запросов в ASP.NET Core приложении. Он состоит из последовательности делегатов запросов, вызываемых один за другим.

Цепочка у middleware следующая: обработать – передать следующему. Если не вызывать следующую часть, то произойдет замыкание, что позволяет избежать ненужной работы.

**app.Use()**

Этот метод используется для конфигурирования нескольких middleware. В отличие от app.Run(), мы можем включить в него параметр next, который вызывает следующий делегат запроса в конвейере. Мы также можем замкнуть (завершить) конвейер, не вызывая параметр next.



**app.Map()**

Этот метод расширения используются как условное обозначение для ветвления конвейера. Map разветвляет конвейер запросов на основе пути запроса. Если путь запроса начинается с указанного пути, ветвь выполняется.

**app.Run()**

Этот метод добавляет middleware-компонент в виде Run[Middleware], который выполнится в конце конвейера. Как правило, он действует как замыкающее middleware и добавляется в конце конвейера запросов, поскольку не может вызывать следующий middleware-компонент. Run — в соглашениях описывается так, что этим методом стоит пользоваться, только тогда, когда мы хотим добавить наш middleware в конец обработки запроса(pipeline), соответственно после Run ничего не будет вызвано.

**ЖЦ:**

Метод Configure выполняется один раз при создании объекта класса Startup, и компоненты middleware создаются один раз и живут в течение всего жизненного цикла приложения. То есть для последующей обработки запросов используются одни и те же компоненты.

**OWIN или Open Web Interface for .NET** представляет собой спецификацию, определяющую взаимодействие между веб-приложением и веб-сервером. Современным веб-приложениям требуется больше гибкости. И ответом на эти вызовы стало появление спецификации OWIN.

Каждое OWIN приложение должно иметь Startup класс, в котором будет находится определение наших компонентов.

**Katana** — это набор компонентов для создания и запуска веб-приложений на hosting abstraction, реализация спецификации owin. Katana позволяет создавать приложение, использующее различные веб-технологии, и запускать его, где угодно.

**Архитектура Katana:** приложение – промежуточное по(middleware) – сервер – хост.