Интернет – глобальная сеть позволяющая обмениваться данными по протоколу tcp/ip

Компоненты: **службы** (DNS, DHCP), **протоколы**, **организации** (IETF, ICAAN, RFC), **безопасность, документы**.

Клиент-серверная архитектура – архитектура, в которой клиент отправляет запрос серверу, который обрабатывает их и возвращает ответ. Инициатор клиент

Типы взаимодействия клиента и сервера:

1. Симплексная – данные передаются только в одном направлении
2. Полудуплексная – данные могут передаваться в двух направлениях, но не одновременно
3. Дуплексная – данные могут передаваться в двух направлениях одновременно
4. подписчик/издатель

Веб-приложение – приложение, основанное на клиент-серверной архитектуре работающее по протоколу http.

Свойства HTTP:

1. На прикладном уровне iso/osi
2. Клиент-серверная арх-ра
3. Без состояния
4. Ассиметричный (запрос-ответ)
5. Кэширование
6. Безопасность
7. Не шифруется
8. Работает на tcp/ip
9. Полудуплексный
10. Поддерживает передачу различных форматов данных
11. 80- http; 443- https

- Версии:

- HTTP/1.0: Первая версия, без поддержки постоянных соединений.

- HTTP/1.1: Поддерживает keep-alive, что позволяет сохранять соединение.

- HTTP/2: Использует бинарный формат, поддерживает мультиплексирование и дуплексный обмен.

- HTTP/3: Основан на протоколе QUIC (UDP), обеспечивающий быструю передачу данных.

Stateless протокол - тип протокола, который не сохраняет информацию о взаимодействиях клиента и сервера. Каждый запрос рассматривается как отдельная и независимая операция.

HTTPS - (Hypertext Transfer Protocol Secure) — это расширение протокола HTTP, которое обеспечивает безопасную передачу данных между клиентом и сервером.

Структура запроса: метод, URI, версия http, заголовки, тело

Группы заголовков:

1. Запросов
2. Ответов
3. Entity
4. General
5. Пользовательские

Методы запроса:

GET, POST, PUT, DELETE,

HEAD (Запрашивает заголовки ресурса, без тела),

OPTIONS (Запрашивает информацию о доступных HTTP-методах для указанного ресурса),

PATCH (частичное обновление)

Структура URI:

Схема – Домен – Путь – Запрос + фрагмент

Идемпотентность - свойство операций, при котором повторное выполнение одной и той же операции не изменяет результат после первого выполнения (GET, PUT, DELETE, HEAD, OPTIONS)

Структура ответа: версия http, статус-код, статус-сообщение, заголовки, тело

Коды состояния:

100-ые – информационные

200-ые успешные

300-ые перенаправления

400-ые ошибки клиента

500-ые ошибки сервера

Сессия - способ хранения информации о пользователе во время его взаимодействия с веб-приложением

Куки - небольшие текстовые файлы, которые веб-сервер отправляет на устройство пользователя и сохраняет в браузере. Они используются для хранения информации о пользователе и его взаимодействии с веб-сайтом

клиентские куки хранятся на стороне клиента и отправляются с каждым новом запросом

серверные куки создаются и управляются сервером для сохранения состояния сессии.

Способы передачи параметров от клиента на сервер:

1. URL-параметры
2. Тело
3. Заголовки
4. Куки
5. Формы

Способы передачи параметров от сервера:

1. Пуллинг - это метод, при котором клиент регулярно отправляет HTTP-запросы на сервер для проверки наличия новых данных.
2. Вебсокет - это протокол, который обеспечивает двустороннюю связь между клиентом и сервером по постоянному соединению.

Способы сохранить состояние в http:

1. Куки
2. Сессии
3. Local Storage (Хранит данные в браузере на стороне клиента.)
4. Session Storage (данные хранятся только в течение сессии и удаляются при закрытии вкладки или браузера)
5. Url параметры
6. Формы

- Схема обработки:

Обработка исключений обычно осуществляется с использованием middleware, который перехватывает ошибки и предоставляет соответствующий ответ (например, 500 Internal Server Error).

Рукопожатие

Клиент обращается к серверу с просьбой установить безопасное соединение и предлагает набор шифров, которые «понимает», а также совместимую версию SSL/TLS. Сервер проверяет присланный шифронабор, сравнивает со своим, и отсылает ответ клиенту с файлом сертификата и открытым ключом.

**Процесс установления соединения (СЕРТИФИКАТ И СЕССИОННЫЙ КЛЮЧ)**

1. **Инициация**:
   * Клиент отправляет ClientHello с параметрами соединения.
2. **Ответ сервера**:
   * Сервер отвечает ServerHello, затем отправляет свой сертификат клиенту.
3. **Проверка сертификата**:
   * Клиент проверяет сертификат сервера, удостоверяясь, что он действителен и подписан надежным CA.
4. **Генерация сессионного ключа**:
   * Клиент генерирует сессионный ключ и шифрует его с использованием публичного ключа, содержащегося в сертификате сервера.
5. **Отправка сессионного ключа**:
   * Клиент отправляет зашифрованный сессионный ключ на сервер.
6. **Дешифрование**:
   * Сервер использует свой приватный ключ для дешифрования сессионного ключа.
7. **Шифрование данных**:
   * После успешного обмена сессионным ключом обе стороны используют его для шифрования и дешифрования данных во время сеанса.

Конвейер:

По конвейеру проходить будем пока не дойдем до middleware

Можно остановиться и обратно пойти

Middleware- компонент для обработки запросов

**15. Назначение паттерна Builder.**

Паттерн **Builder** (Строитель) используется для пошагового построения сложных объектов, абстрагируя процесс создания от конкретных деталей реализации. Он позволяет создавать объекты с различными конфигурациями, избегая перегруженных конструкторов.  
В **ASPA (ASP.NET Core)** этот паттерн применяется, например, при конфигурировании **WebApplicationBuilder** для пошаговой настройки компонентов приложения (сервисов, Middleware и маршрутизации).

**16. Назначение паттерна Chain of Responsibility.**

Паттерн **Chain of Responsibility** (Цепочка обязанностей) позволяет передавать запрос по цепочке обработчиков, пока один из них не обработает его. Это помогает разделить ответственность и упростить поддержку кода.  
В **ASPA** этот паттерн реализуется в **Middleware** — каждый Middleware-компонент либо обрабатывает запрос, либо передаёт его дальше по конвейеру обработки.

**17. Понятие Middleware.**  
**Middleware** – это программный компонент, который обрабатывает HTTP-запросы и ответы в ASP.NET Core. Он может выполнять аутентификацию, логирование, обработку ошибок, маршрутизацию и многое другое. Middleware компоненты выстраиваются в конвейер обработки запросов, где каждый компонент может изменять запрос или ответ, либо передавать управление следующему Middleware.

**18. Понятие статического файла.**  
Статический файл – это файл, который не требует серверной обработки и передаётся клиенту в неизменённом виде. Примеры: **HTML, CSS, JavaScript, изображения, видео**. В ASP.NET Core они обслуживаются специальным Middleware.

**19. Отличительный признак наименования Middleware-компонента.**  
Общепринятый признак наименования Middleware-компонента — использование суффикса **Middleware** (например, **UseRoutingMiddleware, UseAuthenticationMiddleware**). При регистрации компонентов в конвейере запросов применяются методы **UseXXX()**, например, app.UseRouting();.

**20. Встроенные Middleware-компоненты в ASP.NET Core:**

* **UseRouting** – определяет маршрутизацию.
* **UseEndpoints** – обрабатывает конечные точки маршрутов.
* **UseAuthentication** – выполняет аутентификацию.
* **UseAuthorization** – проверяет права доступа.
* **UseStaticFiles** – обслуживает статические файлы.
* **UseExceptionHandler** – обрабатывает исключения.
* **UseCors** – включает поддержку CORS.
* **UseSession** – поддержка сессий.
* **UseHsts** – включает защиту HTTPS.

**21. Директория для хранения статических файлов по умолчанию в ASPA:**  
**wwwroot**.

**22. Кто такой Джон фон Нейман?**  
**Джон фон Нейман (1903–1957)** — венгерско-американский математик, один из основателей теории игр, квантовой механики и создатель **архитектуры фон Неймана**, на основе которой работают современные компьютеры. В его модели программный код и данные хранятся в одной памяти, а процессор выполняет инструкции последовательно.

 **Блоки try и catch**:

* Код, который может вызвать исключение, помещается в блок try.
* Если возникает исключение, управление передается в соответствующий блок catch, где можно обработать ошибку.

 **Middleware для обработки исключений**:

* В ASP.NET можно использовать middleware для обработки исключений на уровне приложения. Это позволяет централизовать логику обработки ошибок и возвращать стандартизированные ответы клиенту.
* Например, можно создать middleware, который будет перехватывать исключения и возвращать JSON-ответ с информацией об ошибке, включая статус и сообщение [[1]](https://stackoverflow.com/questions/38630076/asp-net-core-web-api-exception-handling).