Стартовые вопросы

Интернет

```
Всемирная компьютерная сеть, построенная на основе стека протоколов
TCP/IP.
```

Компоненты интернета

```
DNS, E-mail(STMP,POP3, IMAP), IRC(обмен сообщений в реальном времени),
FTP,Telnet (управление удаленным компьютером в терминальном режиме), WWW.
```

Клиент-серверная архитектура (какие есть типы взаимодействия)

(https://chat.deepseek.com/a/chat/s/956a4fbb-a6ae-4817-abb7-544cfccc5fcb)
клиент-серверное приложение = приложение (программа) с клиент-серверной архитектурой: приложение, состоящее из двух компонент – клиента и сервера; клиент и сервер взаимодействуют между собой в соответствии с заданными правилами (спецификациями, протоколами); для взаимодействия между клиентом и сервером в соответствии с правилами (спецификацией, протоколом) должно быть установлено соединение; инициатором соединения всегда является клиент**.

По модели коммуникации (кто инициирует обмен):

```
Запрос-Ответ (Request-Reply)
Очередь сообщений (Message Queue / Message Brokering)
Публикация-Подписка (Publish-Subscribe / Pub-Sub)
Потоковая передача данных (Streaming)
```

По режиму работы:

```
    Синхронное взаимодействие
    Асинхронное взаимодействие
```

Веб-приложение понятие

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)

Свойства http (тут также кратко про отличие версий)

- 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;
- поддерживается W3C, описан в нескольких RFC.

Что такое stateless протокол

Stateless (бессостоятельный) протокол — это такой протокол связи, в котором **сервер не хранит никакой информации о предыдущих запросах от одного и того же клиента**. Каждый запрос от клиента обрабатывается как совершенно новый и независимый.

Ключевая характеристика:

- Запрос от клиента должен содержать **всю необходимую информацию** для его обработки сервером.
- Серверу не нужно помнить, что было в прошлых запросах. После отправки ответа он "забывает" о клиенте.

HTTPS (процедура рукопожатия)

HTTPS (HTTP Secure) — это не отдельный протокол, а HTTP поверх шифрующих механизмов протоколов **SSL/TLS**. Его главная цель — обеспечить **конфиденциальность** (шифрование), **целостность** (защита от подмены) и **аутентификацию** (уверенность, что вы общаетесь с тем сервером, с которым нужно) данных.

Процесс установления безопасного соединения называется **"рукопожатие" (TLS Handshake)**. Вот его упрощенная схема:

Цели рукопожатия:

1. Согласовать версию TLS и алгоритмы шифрования.
2. Проверить подлинность сервера (с помощью SSL-сертификата).

3. Сгенерировать общие **сессионные ключи** для симметричного шифрования (которое гораздо быстрее асимметричного).

Шаги рукопожатия:

1. `ClientHello` (Приветствие от клиента)
   - Клиент (браузер) отправляет серверу:
     - Поддерживаемые версии TLS (например, TLS 1.2, TLS 1.3).
     - Список поддерживаемых шифров (ciphersuites).
     - Случайное число (`client random`).
2. `ServerHello` (Ответ сервера)
   - Сервер отвечает клиенту:
     - Выбранную версию TLS и шифр из предложенных клиентом.
     - Свой `server random` (случайное число).
     - **Свой SSL-сертификат**, содержащий его публичный ключ и подписанный доверенным центром сертификации (CA).
3. **Проверка сертификата**
   - **Клиент проверяет сертификат:**
     - Не истек ли срок его действия?
     - Выдан ли он доверенным Центром сертификации (CA), чьи корневые сертификаты есть в ОС/браузере клиента?
     - Соответствует ли доменное имя в сертификате тому домену, с которым устанавливается соединение?
   - Если проверка не пройдена, браузер покажет предупреждение о ненадежном соединении.
4. `ClientKeyExchange` (Обмен ключами)
   - Клиент генерирует еще одно случайное число, называемое `premaster secret`.
   - Он шифрует это число **публичным ключом сервера** (из сертификата) и отправляет его серверу.
   - **Только сервер** может расшифровать это сообщение своим **приватным ключом**.
5. **Генерация сессионных ключей**
   - И клиент, и сервер **независимо друг от друга** используют `client random`, `server random` и `premaster secret`, чтобы вычислить одни и те же **сессионные ключи** (общий секрет). Эти ключи будут использоваться для симметричного шифрования всего дальнейшего трафика.
6. `Finished` (Готово)
   - Обе стороны обмениваются сообщениями, зашифрованными уже новыми сессионными ключами, чтобы подтвердить, что handshake прошел успешно и соединение безопасно.

После handshake:

- Устанавливается **безопасное симметричное шифрование**.
- Начинается обмен данными защищенного HTTP — собственно, HTTPS. Все передаваемые данные (заголовки, cookies, содержимое страниц) шифруются сессионными ключами.

Итог: HTTPS через процесс handshake решает главную проблему безопасности в интернете — как безопасно обменяться ключами шифрования по незащищенному каналу, используя асимметричное шифрование для старта и быстрейшее симметричное — для основной передачи данных.

Структура запроса

1. **Стартовая строка (Request Line)**
   Всегда одна строка и состоит из трех элементов, разделенных пробелами:

   - **Метод HTTP (HTTP Method):** Определяет действие, которое нужно выполнить.
   - `GET`, `POST`, `PUT`, `DELETE`, etc.
   - **Путь к ресурсу (Request Target):** Путь до запрашиваемого ресурса на сервере (иногда включает параметры запроса).
   - `/articles/http-basics`

   Версия HTTP (HTTP Version): Используемая версия протокола.

   - `HTTP/1.1`, `HTTP/2` или `HTTP/3`.

   Пример: `GET /articles/http-basics HTTP/1.1`
2. **Заголовки (Headers)**
   Это набор пар "ключ: значение", которые передают серверу дополнительную информацию о запросе. Каждый заголовок — на новой строке.
   - **`Host`:** Обязательный заголовок в HTTP/1.1. Указывает доменное имя сервера. Позволяет одному серверу обслуживать несколько сайтов.
   - **`User-Agent`:** Описывает клиентское приложение (браузер, ОС).
   - **`Accept`**, `Accept-Language`, `Accept-Encoding`: Сообщают серверу, какие типы контента, языки и кодировки понимает клиент.
   - **`Content-Type`** (для тел запросов): Указывает тип данных, которые клиент отправляет в теле (например, `application/json`, `application/x-www-form-urlencoded`).
   - **`Content-Length`** (для тел запросов): Указывает размер тела запроса в байтах.
   - **`Authorization`:** Содержит учетные данные для доступа к защищенному ресурсу (например, токен или логин/пароль).
   - **`Cookie`:** Передает серверу данные, которые он ранее сохранил на клиенте.

   Заголовков может быть много. Их список завершается **пустой строкой**.
3. **Тело сообщения (Body)** — опционально, присутствует не всегда.
   - **Необязательная** часть запроса.

- Присутствует в запросах, которые отправляют данные на сервер: `POST`, `PUT`, etc.
- **Отсутствует** в запросах, которые только получают данные: `GET`, `HEAD`, `DELETE`.
- Формат тела описывается заголовком `Content-Type`.
  - `application/json`: `{"username": "john", "age": 30}`
  - `application/x-www-form-urlencoded`: `username=john&age=30`
  - `multipart/form-data`: Используется для загрузки файлов.

Группы заголовков

Заголовки HTTP сообщений (как запросов, так и ответов) делятся на несколько основных групп по их назначению и месту использования.

Группа заголовков	Описание	Примеры (наиболее распространенные)
General Headers (Общие заголовки)	Могут присутствовать как в запросах, так и в ответах. Они не относятся к самому телу сообщения, а описывают общие параметры передачи.	`Cache-Control`, `Connection`, `Date`, `Pragma`, `Transfe Encoding`, `Upgrade`, `Via`
Request Headers (Заголовки запроса)	Содержат информацию о запрашиваемом ресурсе и о самом клиенте. Используются только в HTTP-запросах.	`Accept`, `Accept-Encoding`, `Authorization`, `Host`, `User-Agent`, `Referer`, `Cookie`
Response Headers (Заголовки ответа)	Содержат дополнительную информацию о ответе сервера: его местоположение, о самом сервере и т.д. Используются	`Server`, `Set-Cookie`, `Location`, `WWW-Authenticate`, `ETag`

Группа заголовков	Описание	Примеры (наиболее распространенные)
	только в HTTP-ответах.	
Entity Headers (Заголовки сущности)	Описывают содержание (body) передаваемого сообщения: его тип, размер, время модификации. Могут быть в запросе (например, POST) и ответе.	`Content-Type`, `Content-Length`, `Content-Encoding`, `Last-Modified`, `Expires`
Representation Headers (Заголовки представления)	В HTTP/2 и выше описывают исходный формат данных (representation) ресурса, отделяя его от самого ресурса.	

Методы запроса

Методы HTTP (часто называемые "глаголы") определяют действие, которое клиент хочет выполнить с указанным ресурсом.

Метод	Описание	Идемпотентность*	Безопасность**
`GET`	**Запрашивает** представление указанного ресурса. Запросы с использованием GET должны только получать данные.	Да	Да
`POST`	**Отправляет** данные на сервер (например, данные формы, файл). Часто вызывает изменение состояния на сервере (создание новой записи).	Нет	Нет
`PUT`	**Заменяет** все текущие представления ресурса данными из запроса. Используется для обновления	Да	Нет

Метод	Описание	Идемпотентность*	Безопасность**
	или создания ресурса, если известен его URI.		
`DELETE`	**Удаляет** указанный ресурс.	Да	Нет
`PATCH`	**Частично изменяет** ресурс. Отправляет только те данные, которые необходимо изменить, а не всю сущность.	Нет	Нет
`HEAD`	Запрашивает ресурс так же, как `GET`, но **без тела ответа**. Используется для получения метаданных (заголовков) о ресурсе.	Да	Да
`OPTIONS`	Описывает **параметры соединения** с ресурсом (какие методы поддерживаются сервером для данного URL).	Да	Да
`CONNECT`	Используется для установления **туннеля** к серверу, определяемому по целевому ресурсу. Нужен для SSL-туннелирования через прокси.	Нет	Нет
`TRACE`	Выполняет **вызов возвратного сообщения** с самим запросом. Используется для диагностики.		

Структура uri

URI (Uniform Resource Identifier) — это строка, идентифицирующая ресурс. Его структура выглядит следующим образом:
scheme:[//[user[:password]@]host[:port]][/path][?query][#fragment]

Разберем компоненты на примере `https://john:pass@www.example.com:8080/docs/data.txt?lang=en&print=true#page=2`

Компонент	Описание	Пример из URL
`scheme` (Схема)	Определяет протокол, который будет использоваться для доступа к ресурсу.	`https`
`authority` (Авторитетная	Содержит информацию для аутентификации и	`//john:pass@www.example.com:8080`

Компонент	Описание	Пример из URL
часть)	адресации. Начинается с `//`.	
↳ `userinfo`	Необязательные данные пользователя для аутентификации (логин и пароль).	`john:pass`
↳ `host`	Доменное имя или IP-адрес сервера.	`www.example.com`
↳ `port`	Необязательный номер порта. По умолчанию используется стандартный для схемы (для HTTPS — 443).	`:8080`
`path` (Путь)	Иерархический путь к ресурсу на сервере.	`/docs/data.txt`
`query` (Запрос)	Необязательная строка запроса, содержащая параметры в виде пар `ключ=значение`, разделенных `&`. Начинается с `?`.	`?lang=en&print=true`
`fragment` (Фрагмент)	Необязательный идентификатор вторичного ресурса (например, якорь на странице). Не отправляется на сервер. Начинается с `#`.	

Важное отличие URI от URL: Все URL являются URI, но не наоборот.

- **URI** (Identifier) — более широкое понятие, это просто идентификатор.
- **URL** (Locator) — это URI, который не только идентифицирует ресурс, но и указывает его **местонахождение** и **способ его получения** (протокол).

Идемпотентность

- **Идемпотентность**: Означает, что многократное выполнение одного и того же запроса идентично однократному выполнению (например, `DELETE /api/users/123` — сколько раз ни выполни, результат будет тот же, что и после первого успешного вызова). `POST` не идемпотентен, так как два одинаковых запроса создадут две одинаковые сущности.
- **Безопасность**: Означает, что метод не изменяет состояние сервера. `GET`, `HEAD`, `OPTIONS` считаются безопасными и используются только для

чтения данных.

Структура ответа

1. **Строка статуса (Status Line)**
   Первая строка ответа. Состоит из трех частей:

   - **Версия HTTP (HTTP Version):** Версия протокола, которую использует сервер.
     - `HTTP/1.1`, `HTTP/2`
   - **Код состояния (Status Code):** Трехзначное число, которое кратко сообщает о результате обработки запроса.
     - `200` (Успех), `404` (Не найдено), `500` (Ошибка сервера)
   - **Пояснение к коду состояния (Reason Phrase):** Краткое текстовое описание кода.
     - `OK`, `Not Found`, `Internal Server Error`

   Пример: `HTTP/1.1 200 OK`
2. **Заголовки (Headers)**

   Аналогично запросу, это набор пар "ключ: значение", которые описывают ответ.

   - **`Server`:** Информация о программном обеспечении сервера.
   - **`Content-Type`:** MIME-тип данных, которые находятся в теле ответа.
     - `text/html`, `application/json`, `image/png`
   - **`Content-Length`:** Размер тела ответа в байтах.
   - **`Content-Encoding`:** Способ сжатия тела ответа (например, `gzip`).
   - **`Set-Cookie`:** Инструкция браузеру сохранить куки.
   - **`Cache-Control`:** Инструкции по кешированию ответа браузером.
   - **`Location`:** Используется для перенаправлений (редиректов) с кодами 3xx. Содержит новый URL.

   Заголовки завершаются **пустой строкой**.
3. **Тело сообщения (Body)** — опционально.
   - **Необязательная** часть ответа.
   - Содержит запрошенный ресурс (HTML-страницу, картинку, данные JSON) или сообщение об ошибке.
   - Его наличие и тип определяются заголовками `Content-Length` и `Content-Type`.

Коды состояния

- **1**xx: информационные сообщения;
- **2xx**: успешный ответ;
- **3**xx: переадресация;

- **4**xx: ошибка клиента;
- **5**xx: ошибка сервера.

Что такое сессия

серверный объект, хранящий информацию о соединении с клиентом, создается при первом обращении время жизни: **timeout** (системный параметр, обычно 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 Request принадлежит, какой-то сессии (имеет ссылку на объект Session или содержит Session ID). Обычно объект Session предоставляет приложению возможность хранить данные в формате ключ/значение.

Что такое куки

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.

Способы передачи параметров от клиента на сервер

В URL (Query String / Параметры запроса)

- **Как:** Параметры добавляются в конец URL после знака вопроса `?` в формате `ключ=значение`. Несколько параметров разделяются амперсандом `&`.
- **Синтаксис:**
  `http://example.com/page?param1=value1¶m2=value2
- **Как получить на сервере:** Через переменную `query string` или ее парсинг (например, в Node.js — `req.query`, в PHP — `$_GET`, в Python/Flask — `request.args`).
- **Плюсы:**
  - Можно закладмить URL с параметрами.
  - Видно пользователю (прозрачно).
  - Подходит для несекретных данных (фильтры, настройки сортировки, поисковые запросы).
- **Минусы:**
  - Ограниченная длина (максимальная длина URL зависит от браузера и сервера).
  - Параметры видны в истории браузера, логах, Referrer.

    - **Нельзя** передать файл.
- **Типичное использование:** GET-запросы для фильтрации, поиска, пагинации (например, `?page=2&sort=price`).

В теле запроса (Request Body)

- **Как:** Параметры передаются в невидимой части HTTP-запроса. Используется для методов `POST`, `PUT`, `PATCH`.
- **Основные форматы:**
    1. **`application/x-www-form-urlencoded`** (стандартный для HTML-форм):
        - Формат: `key1=value1&key2=value2` (как в URL, но в теле запроса).
    2. **`multipart/form-data`** (обязателен для загрузки файлов):
        - Разбивает тело на части (parts), каждая со своими заголовками. Позволяет передавать бинарные данные (файлы).
    3. **`application/json`** (популярен для REST API):
        - Формат: `{"key1": "value1", "key2": "value2"}`.
- **Плюсы:**
    - Нет ограничений на длину (разумных).
    - Можно передавать файлы и сложные структуры данных (JSON).
    - Более безопасно (не светится в URL, но все равно передается в открытом виде без HTTPS).
- **Минусы:**
    - URL нельзя закладмить.
    - Можно использовать только с методами, у которых есть тело запроса (не GET).
- **Типичное использование:** Отправка данных форм, авторизация, создание/изменение объектов через API.

В пути URL (URL Path / Route Parameters)

- **Как:** Параметры являются частью самого пути к ресурсу, а не строки запроса.
- **Синтаксис:**
  `http://example.com/users/**123**/posts/**456**`
  (Здесь `123` — это `userId`, `456` — это `postId`)
- **Как получить на сервере:** Фреймворки позволяют объявлять шаблоны путей и извлекать параметры (в Express.js — `req.params`, в Django — `path converters`, в Spring — `@PathVariable`).
- **Плюсы:**
    - Чистый, семантичный URL, который сам описывает ресурс.
    - Лучше для SEO (поисковые системы любят читаемые пути).
- **Минусы:**

    - Подходит только для идентификации ресурса, а не для передачи произвольных параметров.
- **Типичное использование:** RESTful API, где URL определяет ресурс (например, `/users/{id}`).

В заголовках (Headers)

- **Как:** Специальные HTTP-заголовки.
- **Примеры:**
    - `Authorization: Bearer <token>` — для передачи токена аутентификации.
    - `Cookie: name=value` — автоматически отправляются браузером.
    - Кастомные заголовки, например, `X-API-Key: your-api-key`.
- **Плюсы:** Идеально для служебной информации (аутентификация, метаданные).
- **Минусы:** Неудобно для передачи больших объемов данных или данных, напрямую связанных с бизнес-логикой.

Способы сохранить состояние в http

1. **Куки (Cookies)**: Это небольшие фрагменты данных, которые сервер отправляет клиенту, а клиент сохраняет и отправляет обратно с каждым запросом. Куки могут использоваться для хранения идентификаторов сессий, предпочтений пользователя и других данных.
2. **Сессии (Sessions)**: На сервере создаётся сессия для каждого пользователя, и её идентификатор хранится в куки на клиенте. Данные сессии хранятся на сервере, что позволяет сохранять большие объёмы данных.
3. **Локальное хранилище (Local Storage и Session Storage)**: Современные браузеры поддерживают локальное хранилище, которое позволяет сохранять данные на стороне клиента. Local Storage сохраняет данные до явного удаления, а Session Storage — только на время сессии (до закрытия вкладки).
4. **URL-параметры**: Состояние можно передавать через параметры URL, но это не очень безопасно и удобно для больших объёмов данных.
5. **Скрытые поля формы (Hidden Form Fields)**: Данные могут быть сохранены в скрытых полях формы и отправлены вместе с ней при следующем запросе.
6. **Web Storage API**: Это расширенный аналог локального хранилища, предоставляющий более гибкие возможности для хранения данных на клиенте.
7. **JWT (JSON Web Tokens)**: Токены, содержащие данные о пользователе, которые могут быть проверены сервером. Они могут храниться на клиенте и отправляться с каждым запросом.
8. **База данных**: Для долгосрочного хранения состояния можно использовать базу данных на сервере. Данные пользователя сохраняются и могут быть извлечены в любой момент.