**1. Какой уровень TCP/IP заменяет порт**

В **модели TCP/IP** порты работают на **транспортном уровне**. Этот уровень отвечает за установление соединений и передачу данных между приложениями. Он использует такие протоколы, как:

* **TCP (Transmission Control Protocol)** — надёжный протокол, который устанавливает соединение с помощью трёхстороннего рукопожатия и гарантирует доставку данных.
* **UDP (User Datagram Protocol)** — менее надёжный, но быстрый протокол, который не гарантирует доставку данных.

**Порт** — это число, которое используется для того, чтобы различать приложения на одном и том же IP-адресе. Примеры портов:

* 80 — HTTP
* 443 — HTTPS
* 25 — SMTP (почта)

**2. HTTP: что это и зачем он нужен**

**HTTP (Hypertext Transfer Protocol)** — это протокол прикладного уровня, который используется для передачи данных в сети. Основное его применение — это передача веб-страниц между серверами и клиентами (например, браузерами). Основные особенности:

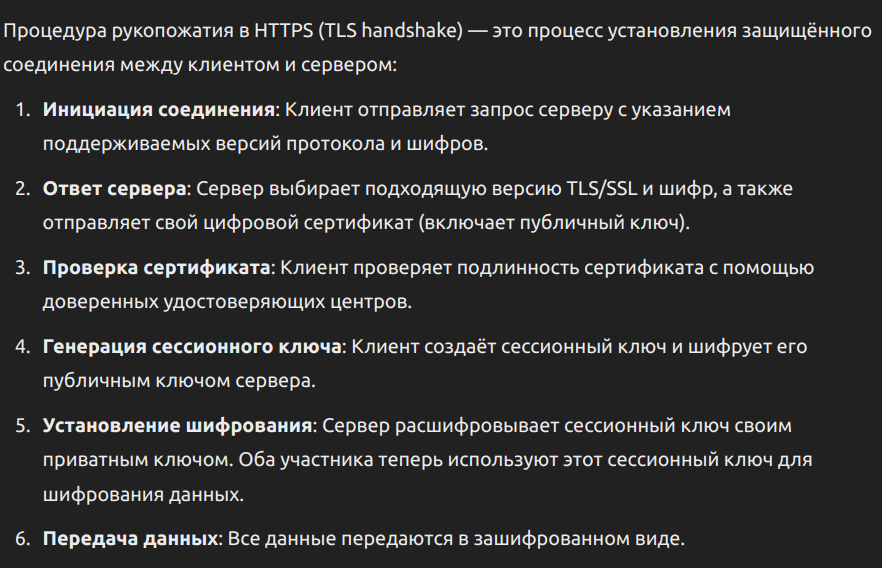
* Без сохранения состояния: HTTP не отслеживает сессии, для этого применяются механизмы, такие как куки или session ID.
* Структура состоит из запросов и ответов, передаваемых по сети.
* Работает поверх транспортного протокола TCP, хотя существуют расширения, использующие UDP (например, HTTP/3).

**3. Процедура рукопожатия**

**Трёхстороннее TCP-рукопожатие (three-way handshake)** — это процесс установления соединения между клиентом и сервером. Он состоит из следующих шагов:

1. **SYN**: Клиент посылает серверу запрос на установление соединения, посылая сегмент SYN.
2. **SYN-ACK**: Сервер подтверждает, что получил запрос, и отправляет клиенту ответ с сегментом SYN-ACK.
3. **ACK**: Клиент подтверждает получение ответа от сервера, отправив сегмент ACK, и соединение устанавливается.

Этот процесс необходим для того, чтобы оба участника (клиент и сервер) убедились в готовности передавать данные.



**4. Сеансовый ключ**

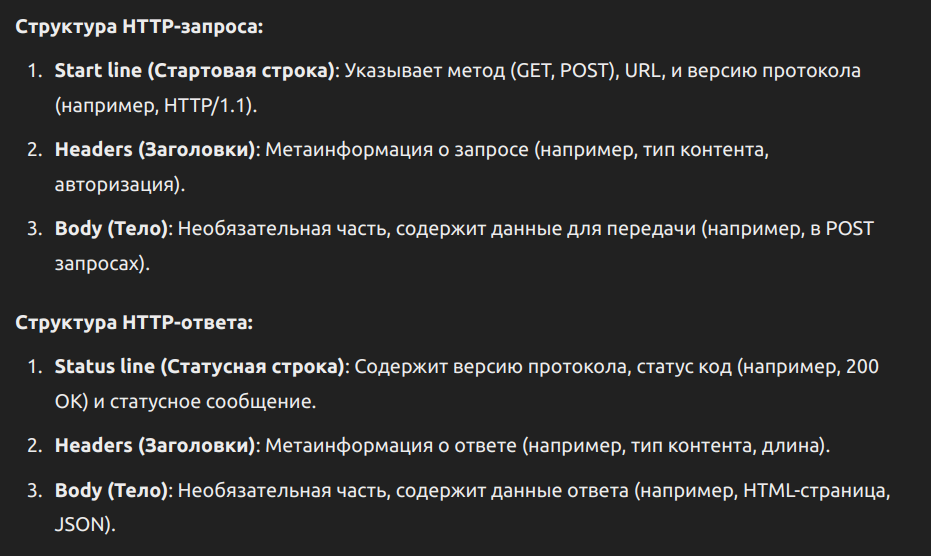
**Сеансовый ключ** — это временный ключ, который используется для шифрования данных между клиентом и сервером в течение одной сессии. Применяется в протоколах вроде **HTTPS** (который использует TLS) для обеспечения безопасной передачи данных. Особенности:

* Временный: сессия длится ограниченное время, после чего ключ больше не используется.
* Генерируется заново при каждом новом сеансе.

**5. Типы запросов (HTTP методы)**

В HTTP есть несколько методов для взаимодействия с ресурсами на сервере:

* **GET**: Используется для запроса данных. Пример: получение веб-страницы.
* **POST**: Используется для отправки данных на сервер (например, отправка формы).
* **PUT**: Обновляет существующие данные. Если ресурс не существует, то может быть создан.
* **DELETE**: Удаляет ресурс.
* **PATCH**: Выполняет частичное обновление ресурса.
* **HEAD**: Аналогичен GET, но возвращает только заголовки ответа, без тела.
* **OPTIONS**: Запрашивает поддерживаемые методы для ресурса.



**Идемпотентность** — это свойство, при котором повторное выполнение одного и того же запроса приводит к одному и тому же результату, без изменения состояния на сервере.

Идемпотентные HTTP-методы:

* GET
* PUT
* DELETE
* HEAD
* OPTIONS

Повторные запросы этих методов не изменят результат.

**6. Отличие URI от URL**

* **URI (Uniform Resource Identifier)** — это общий термин, который идентифицирует ресурс. URI может быть именем или адресом.
* **URL (Uniform Resource Locator)** — это тип URI, который указывает на местоположение ресурса и способ его получения (например, протокол). URL всегда включает в себя схему (например, http://) и путь к ресурсу.

Пример:

* URI: urn:isbn:0451450523 (это уникальный идентификатор книги)
* URL: https://example.com/index.html (это адрес веб-страницы)

**7. Заголовки HTTP**

**HTTP-заголовки** — это метаданные, которые передаются вместе с запросами и ответами для предоставления дополнительной информации. Есть несколько категорий заголовков:

1. **General headers** (Общие заголовки): могут использоваться как в запросах, так и в ответах. Примеры:
   * Date: Дата и время отправки сообщения.
   * Connection: Указывает, будет ли соединение закрыто после завершения передачи.
2. **Request headers** (Заголовки запросов): информация о запросе и клиенте. Примеры:
   * User-Agent: Информация о клиенте (браузере).
   * Accept: Какие форматы данных клиент может принимать.
3. **Response headers** (Заголовки ответа): информация о сервере и ответе. Примеры:
   * Server: Информация о сервере.
   * Set-Cookie: Устанавливает куки для клиента.
4. **Entity headers** (Заголовки сущности): информация о теле сообщения. Примеры:
   * Content-Type: Тип данных (например, application/json).
   * Content-Length: Длина тела ответа.

**Entity заголовки** в заголовках HTTP запросов могут отсутствовать, если запрос не содержит тела. В ответах же эти заголовки используются, если есть тело ответа.

**8. Передача параметров между клиентом и сервером**

Есть несколько способов передать данные между клиентом и сервером:

* **GET-запросы**: Параметры передаются через строку запроса в URL. Например, ?name=value. Эти параметры видны в адресной строке и ограничены по длине.
* **POST-запросы**: Параметры передаются в теле запроса (в формате JSON, FormData, XML). Этот метод предпочтителен для передачи большого количества данных, так как данные не видны в URL.
* Ответ от сервера возвращается в виде тела ответа (чаще всего в формате JSON).

**9. Состояния (states) и их сохранение**

**Состояние** — это информация о текущем взаимодействии клиента и сервера. Сохранение состояния важно для отслеживания данных между запросами (например, авторизация пользователя). Способы сохранения состояния:

* **Cookies**: Маленькие текстовые файлы, которые хранятся на клиенте и передаются с каждым запросом.
* **Session Storage** и **Local Storage**: Хранилище браузера для сохранения данных.
* **JWT (JSON Web Token)**: Токен, который хранится на клиенте и передаётся с каждым запросом для авторизации.

**10. Сессия и её жизненный цикл**

* **Сессия** — это временное взаимодействие между клиентом и сервером. Она отслеживается с помощью session ID, который хранится на клиенте, чаще всего в куках.
* **Жизненный цикл сессии**:
  1. Клиент отправляет запрос на сервер.
  2. Сервер проверяет наличие session ID в запросе.
  3. Если session ID нет, создаётся новая сессия, и сервер отправляет session ID клиенту.
  4. Клиент сохраняет session ID для использования в следующих запросах.
* Если сессия заканчивается, пользователь будет вынужден начать новую сессию (например, войти заново).

**11. HTTP статус-коды**

**HTTP статус-коды** указывают на результат выполнения запроса. Они делятся на несколько категорий:

**1xx: Информационные**

* **100 Continue**: Клиент может продолжать отправлять запрос.
* **101 Switching Protocols**: Сервер переключается на другой протокол, указанный клиентом.

**2xx: Успешные запросы**

* **200 OK**: Запрос выполнен успешно, сервер вернул запрашиваемый ресурс.
* **201 Created**: Запрос выполнен успешно, и был создан новый ресурс.
* **202 Accepted**: Запрос принят, но обработка ещё не завершена.
* **204 No Content**: Успешный запрос, но в ответе нет содержимого.

**3xx: Перенаправления**

* **301 Moved Permanently**: Ресурс был перемещён на новый URL.
* **302 Found**: Ресурс временно перемещён на другой URL.
* **304 Not Modified**: Ресурс не изменялся с момента последнего запроса.

**4xx: Ошибки клиента**

* **400 Bad Request**: Неверный запрос, сервер не может его обработать.
* **401 Unauthorized**: Требуется авторизация для доступа к ресурсу.
* **403 Forbidden**: Доступ запрещён, даже при наличии авторизации.
* **404 Not Found**: Ресурс не найден.
* **405 Method Not Allowed**: Метод, указанный в запросе, не поддерживается сервером.

**5xx: Ошибки сервера**

* **500 Internal Server Error**: Внутренняя ошибка сервера.
* **502 Bad Gateway**: Сервер, выступающий в роли шлюза, получил недействительный ответ.
* **503 Service Unavailable**: Сервер временно недоступен из-за перегрузки или технических работ.
* **504 Gateway Timeout**: Сервер-шлюз не дождался ответа от другого сервера.

**12. Кеширование**

**Кеширование** — это процесс сохранения данных для последующего использования с целью ускорить загрузку страницы и уменьшить нагрузку на сервер. Важные аспекты кеширования:

* **HTTP-заголовки для кеширования**:
  + Cache-Control: Указывает, как долго можно хранить кеш.
  + Expires: Дата, после которой кеш считается недействительным.
  + ETag: Идентификатор версии ресурса, помогающий определить, был ли изменён ресурс.
* **Типы кеширования**:
  + **Клиентское**: Хранение данных на стороне клиента.
  + **Серверное**: Хранение часто запрашиваемых данных на сервере.

**1. JWT (JSON Web Token)**

**JWT (JSON Web Token)** — это компактный и безопасный способ передачи информации между участниками в виде JSON-объекта. JWT используется для аутентификации и авторизации в веб-приложениях.

**Структура JWT**

JWT состоит из трёх частей, разделённых точками:

1. **Header** (заголовок): Содержит информацию о типе токена и алгоритме подписи.
   * Пример: {"alg": "HS256", "typ": "JWT"}
2. **Payload** (полезная нагрузка): Содержит утверждения (claims) — данные о пользователе или других сущностях.
   * Пример: {"sub": "1234567890", "name": "John Doe", "admin": true}
3. **Signature** (подпись): Используется для подтверждения подлинности токена и защиты данных от подделки. Подпись формируется на основе header, payload и секретного ключа.
   * Пример: HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)

#### Использование JWT

JWT часто используется для:

* **Аутентификации**: После успешного входа пользователь получает JWT, который затем отправляется с каждым запросом, чтобы подтвердить его личность.
* **Авторизации**: JWT может содержать права доступа пользователя, и на основе этого сервер решает, какие ресурсы могут быть предоставлены пользователю.

#### Преимущества JWT:

* **Быстрота**: Серверу не нужно хранить токен. Вся информация о пользователе содержится в самом токене.
* **Мобильность**: JWT может использоваться в разных сервисах, API и мобильных приложениях.

#### Недостатки:

* JWT не может быть отозван (если не использовать сторонние механизмы, такие как чёрные списки).
* Токен может занимать больше места, чем session ID, из-за необходимости хранения всей информации в нем.

### 2. ****Сессии****

**Сессия** — это период взаимодействия между клиентом и сервером, при котором сервер сохраняет информацию о состоянии взаимодействия с клиентом (например, состояние авторизации).

#### Как работают сессии:

1. **Создание сессии**: Когда пользователь впервые заходит на сайт, сервер создаёт сессию и присваивает ей уникальный идентификатор (session ID).
2. **Хранение session ID**: Этот идентификатор передаётся клиенту, обычно через куки. Клиент сохраняет этот session ID и отправляет его с каждым последующим запросом.
3. **Поддержка сессии**: Сервер хранит состояние сессии в памяти или в базе данных, и при каждом запросе с session ID сервер проверяет состояние сессии (например, проверяет, авторизован ли пользователь).
4. **Завершение сессии**: Сессия может завершиться вручную (например, при выходе пользователя из системы) или автоматически (по истечении времени неактивности).

#### Преимущества сессий:

* **Безопасность**: Информация о состоянии хранится на сервере, и клиенту передаётся только session ID.
* **Гибкость**: Сервер может легко завершить сессию, удалив её состояние из памяти.

#### Недостатки:

* Требует хранения данных на сервере, что может приводить к увеличению нагрузки при большом количестве сессий.

#### Применение:

Сессии часто используются для отслеживания пользователей на веб-сайтах, например, чтобы сохранить состояние авторизации.

### 3. ****Версионирование HTTP****

**HTTP** — это протокол, используемый для передачи данных по сети. На сегодняшний день существует несколько версий HTTP, каждая из которых улучшала производительность и безопасность по сравнению с предыдущими.

#### ****HTTP/1.0****

* Введён в 1996 году.
* Поддерживает только **один запрос на соединение**: каждое соединение закрывается после отправки одного запроса и получения одного ответа.
* Нет явной поддержки **кеширования**.

#### ****HTTP/1.1****

* Введён в 1999 году.
* Поддерживает **постоянные соединения** (keep-alive), что позволяет отправлять несколько запросов по одному соединению.
* Внедрены **заголовки для кеширования** (Cache-Control), что улучшает работу с кешем.
* Поддержка **чёткой обработки ошибок** с введением статус-кодов.

#### ****HTTP/2****

* Введён в 2015 году.
* Поддерживает **мультиплексирование**: несколько запросов и ответов могут передаваться одновременно через одно соединение, что снижает задержки.
* Поддержка **сжатия заголовков**, что уменьшает объём передаваемых данных.
* Сервер может отправлять клиенту **push-уведомления**, что ускоряет загрузку веб-страниц.

#### ****HTTP/3****

* Работает поверх протокола **QUIC** (UDP вместо TCP), что увеличивает скорость передачи данных и снижает задержки.
* Улучшена поддержка мобильных устройств, так как HTTP/3 лучше работает с изменениями IP-адреса при переключении сетей (например, между Wi-Fi и сотовой связью).

### 4. ****Типы HTTP-запросов****

#### ****GET****

* **Описание**: Запрашивает данные с сервера, не изменяя их.
* **Применение**: Используется для получения ресурсов (например, HTML-страниц, изображений).
* **Пример**: GET /products

#### ****POST****

* **Описание**: Отправляет данные на сервер для создания нового ресурса.
* **Применение**: Используется для отправки форм, загрузки файлов, создания новых записей в базе данных.
* **Пример**: POST /products с телом запроса: {"name": "Product 1", "price": 100}

#### ****PUT****

* **Описание**: Полностью заменяет существующий ресурс на сервере. Если ресурс не существует, он может быть создан.
* **Применение**: Используется для обновления данных.
* **Пример**: PUT /products/1 с телом запроса: {"name": "Updated Product", "price": 150}

#### ****PATCH****

* **Описание**: Выполняет частичное обновление ресурса.
* **Применение**: Используется, когда нужно обновить только часть данных ресурса.
* **Пример**: PATCH /products/1 с телом запроса: {"price": 120}

#### ****DELETE****

* **Описание**: Удаляет ресурс с сервера.
* **Применение**: Используется для удаления записей из базы данных или файлов.
* **Пример**: DELETE /products/1

#### ****HEAD****

* **Описание**: Аналогичен GET, но возвращает только заголовки ответа без тела.
* **Применение**: Используется для проверки наличия ресурса или получения метаданных.
* **Пример**: HEAD /products

#### ****OPTIONS****

* **Описание**: Запрашивает информацию о поддерживаемых методах для указанного ресурса.
* **Применение**: Используется для получения информации о возможностях сервера.
* **Пример**: OPTIONS /products

#### ****CONNECT****

* **Описание**: Используется для установки туннеля к серверу (обычно для SSL-шифрования).
* **Применение**: Используется прокси-серверами.
* **Пример**: CONNECT example.com:443

#### ****TRACE****

* **Описание**: Выполняет тестовый запрос, который возвращает путь запроса через серверы.
* **Применение**: Используется для диагностики маршрута запроса.
* **Пример**: TRACE /

Веб-сервис — это программный компонент, доступный через интернет, который позволяет взаимодействовать с другими приложениями или сервисами посредством стандартных веб-протоколов, таких как HTTP, SOAP, или REST. Веб-сервис предоставляет функциональность, которую другие приложения могут использовать удалённо, обмениваясь данными в формате XML, JSON или другом формате.

### Отличия RESTful от REST API:

1. REST (Representational State Transfer):
   * Архитектурный стиль для создания веб-сервисов.
   * Основные принципы REST:
     + Отсутствие состояния (stateless).
     + Идентификация ресурсов через URI.
     + Использование стандартных HTTP-методов (GET, POST, PUT, DELETE).
     + Кеширование данных.
     + Поддержка разных форматов данных (JSON, XML и т.д.).
2. RESTful:
   * Описание веб-сервиса, который следует принципам REST.
   * RESTful сервисы используют четко структурированные URL для ресурсов и соблюдают правила REST.
3. REST API:
   * Это интерфейс (API), который позволяет клиентам взаимодействовать с RESTful веб-сервисом.
   * REST API предоставляет методы для работы с ресурсами через HTTP-запросы, такие как GET, POST, PUT, DELETE, но не всегда строго соблюдает все принципы REST.

### Отличия:

* REST — это общая архитектурная концепция.
* RESTful — реализация веб-сервиса, полностью соответствующая принципам REST.
* REST API — конкретная реализация интерфейса для доступа к RESTful сервисам, которая может не строго следовать всем REST-принципам.

Основные принципы REST, которые могут не соблюдаться в некоторых API:

1. **Статусность (statelessness)**: Состояние между запросами не должно сохраняться на сервере. Клиент должен отправлять всю необходимую информацию в каждом запросе (например, аутентификацию).
2. **Правильное использование методов HTTP**: Например, использовать GET только для получения данных, POST для создания, PUT для обновления и DELETE для удаления. Некоторые API могут нарушать это правило.
3. **Уникальные URI для ресурсов**: Каждому ресурсу должен соответствовать свой уникальный URI. API может нарушать это, используя URI неправильно (например, запрашивая действия вместо ресурсов).
4. **Без кэширования**: REST предполагает возможность кэширования ответов, но если API не реализует это, оно не будет полностью RESTful.
5. **Единообразие интерфейса**: RESTful API должны предоставлять единообразные структуры ответов и запросов, что иногда нарушается, если API не стандартизирован.

Всемирная компьютерная сеть на основе TCP/IP, организации ISO ioan, типы взаимодействий: дуплекс полудуплекс симплекс