***Клиент-серверная архитектура***

1. **Разница между HTTP & HTTPS**

**HTTP (от англ. HyperText Transfer Protocol — протокол передачи гипертекста)** — это прикладной протокол передачи данных в сети. На текущий момент используется для получения информации с веб-сайтов. Протокол HTTP основан на использовании технологии «клиент-сервер»: клиент, отправляющий запрос, является инициатором соединения; сервер, получающий запрос, выполняет его и отправляет клиенту результат.

**HTTPS (от англ. HyperText Transfer Protocol Secure — безопасный протокол передачи гипертекста)** — это расширение протокола HTTP, **поддерживающее шифрование посредством криптографических протоколов SSL и TLS.**

Чем отличаются HTTP от HTTPS

- HTTPS не является отдельным протоколом передачи данных, а представляет собой **расширение** протокола HTTP с надстройкой **шифрования**;

- передаваемые по протоколу HTTP данные не защищены, HTTPS обеспечивает конфиденциальность информации путем ее шифрования;

**- HTTP использует порт 80, HTTPS — порт 443.**

***Структура HTTP-запроса***

**1. Строка запроса**

Он состоит из 3 частей: **метод запроса, URL и версия протокола, разделенная пробелами**

К методам запроса относятся GET, HEAD, PUT, POST, TRACE, OPTIONS, DELETE и методы расширения. Конечно, не все серверы реализуют все методы. Даже если некоторые методы поддерживаются, они недоступны по соображениям безопасности.

Формат версии протокола: HTTP / основной номер версии. Дополнительный номер версии, обычно используемые HTTP / 1.0 и HTTP / 1.1.

**2. Заголовок запроса.**

Заголовок запроса добавляет некоторую дополнительную информацию к сообщению запроса, которое состоит из пар «имя / значение», по одной паре на строку, а имя и значение разделяются двоеточием.

**Общие заголовки запросов следующие:**

Host : Адрес сервера, принимающего запрос, который может быть IP: номером порта или доменным именем.

User-Agent : Название приложения, отправившего запрос

Connection : Укажите свойства, связанные с подключением, например Connection: Keep-Alive

Accept-Charset : Сообщите формат кодировки, который сервер может отправлять

Accept-Encoding : Уведомить формат сжатия данных, который сервер может отправить

Accept-Language : Сообщите язык, который сервер может отправлять

**3. Тело запроса**

Необязательная часть, например запрос GET без тела запроса

Сервер использует тело сообщения для доставки информации обратно клиенту. Тело сообщения содержит информацию, строку запроса, заголовки, пустую строку и необязательное тело сообщения. Хотя не у всех запросов есть тело, те, у которых оно есть, часто используют POST для доставки полезной нагрузки.

***Структура ОТВЕТА HTTP***

**1. Строка запроса**

Он состоит из 3 частей: **метод запроса, URL и версия протокола, разделенная пробелами**

К методам запроса относятся GET, HEAD, PUT, POST, TRACE, OPTIONS, DELETE и методы расширения. Конечно, не все серверы реализуют все методы. Даже если некоторые методы поддерживаются, они недоступны по соображениям безопасности.

Формат версии протокола: HTTP / основной номер версии. Дополнительный номер версии, обычно используемые HTTP / 1.0 и HTTP / 1.1.

**2. Заголовок запроса.**

Заголовок запроса добавляет некоторую дополнительную информацию к сообщению запроса, которое состоит из пар «имя / значение», по одной паре на строку, а имя и значение разделяются двоеточием.

**Общие заголовки запросов следующие:**

1. **access-control-allow-credentials:** true
2. **access-control-allow-origin:** \*
3. **Connection:** keep-alive
4. **Content-Encoding:** gzip
5. **Content-Type:** application/json
6. **Date:** Tue, 27 Sep 2022 15:16:08 GMT
7. **Server:** nginx
8. **Strict-Transport-Security:** max-age=31536000
9. **Transfer-Encoding:** chunked
10. **Vary:** Accept-Encoding

**Альтернатива:**

**cache-control**

**content-encoding**

**content-type**

**date**

**expires**

**pragma**

**server**

**set-cookie**

**3. Тело запроса**

Необязательная часть, например запрос GET без тела запроса

Сервер использует тело сообщения для доставки информации обратно клиенту. Тело сообщения содержит информацию, строку запроса, заголовки, пустую строку и необязательное тело сообщения. Хотя не у всех запросов есть тело, те, у которых оно есть, часто используют POST для доставки полезной нагрузки.

***Методы HTTP-запроса***

**GET** - получение каких-то данных с сервера

**HEAD** - запрашивает ресурс так же, как и метод GET, но без тела ответа

**POST** - создать новый объект или внести какие-либо изменения

**CONNECT** - установить "туннель" к серверу, определенному по ресурсу

**PUT** - внести изменения в существующий объект

**OPTIONS** - используется для описания параметров соединения с ресурсом

**DELETE** - удалить объект

**PATCH** - используется для частичного изменения ресурса

**TRACE** - выполняет вызов возвращаемого тестового сообщения с ресурса

\_\_\_\_\_\_\_\_\_\_\_\_\_

**Идемпотентный метод**- метод при котором ,при повторном идентичным запросом,имеет один и тот же результат,не меняющий состояние сервера.К таким запросам относятся методы:

1. GET
2. PUT
3. DELETE
4. HEAD
5. OPTIONS
6. TRACE

**то есть один раз просил удалить один файл, если ещё раз попросишь удалить тот же - он новый файл не станет удалять**

**\_\_\_\_\_\_**

**Безопасные методы:**

1. GET
2. HEAD
3. TRACE
4. OPTIONS

Все безопасные методы идемпотентны

\_\_\_\_\_\_\_\_\_\_\_

***XML (eXtensible Markup Language)*** - расширяемый язык разметки. Используется для хранения и передачи данных.

Передача данных - это запросы и ответы в API-методах. **При отправлении SOAP-запроса данные передаются только в XML.** В REST возможны варианты - или XML, или JSON.

Хранение данных - это когда XML встречается внутри кода. Его легко понимает как машина, так и человек. В формате XML можно описывать правила, которые будут применяться к данным.

***JSON***  — текстовый формат обмена данными, основанный на JavaScript. Но при этом формат независим от JS и может использоваться в любом языке программирования.

Данные в JSON записываются парами **"Ключ":"Значение".**

Формат {"ключ": "значение"},[{"такой формат":{"поддерживает безмерную": {"вложенность одного формата ключ значения в другой":{" включая массивы"}}}}]

JavaScript ***(JSON - JavaScript Object Notation)***

***Отличия XML от JSON***

1) XML - язык разметки. JSON - формат, написанный на JavaScript

2) XML - данные хранятся в виде древовидной структуры. JSON - данные хранятся в формате "ключ: значение"

3) XML - Может выполнять обработку и форматирование документов и объектов. JSON - Не выполняет никакой обработки или вычислений

4) XML - Громоздкий и медленный синтаксический анализ. JSON - Очень быстро, поскольку размер файла значительно мал

5) XML - Напрямую не поддерживает массивы. JSON - Поддерживает массив

6) XML -Поддерживает множество сложных типов данных, включая диаграммы, изображения и другие непримитивные типы данных. JSON - поддерживает только строки, числа, массивы, логические значения и объекты. Объекты могут содержать только примитивные типы.

7) XML - поддерживает кодировки UTF-8 и UTF-16. JSON - поддерживает кодировки UTF, а также ASCII.

8) **В SOAP можно применять только XML**, там без вариантов.

В REST можно применять как XML, так и JSON. Разработчики отдают предпочтение json-формату, потому что он проще воспринимается и меньше весит. В XML есть лишняя обвязка, название полей повторяется дважды (открывающий и закрывающий тег).

XML

<req>

<surname>Иванов</surname>

<name>Иван</name>

</req>

JSON

{

"surname": "Иванов",

"name": "Иван",

}

За счет того, что мы не дублируем название поля каждый раз «surname – surname», читать JSON проще. И за счет этого же запрос меньше весит, что при плохом интернете бывает важно. Или при большой нагрузке.

9) **JSON менее безопасен.**

10) **JSON не поддерживает комментарии**. XML поддерживает

**SOAP**

**SOAP** — это протокол, по которому веб-сервисы взаимодействуют друг с другом или с клиентами. Название происходит от сокращения **Simple Object Access Protocol** («простой протокол доступа к объектам»). **SOAP API** — это веб-сервис, использующий протокол SOAP для обмена сообщениями между серверами и клиентами. При этом **сообщения** должны быть написаны на **языке XML** в соответствии со строгими стандартами, иначе сервер вернет ошибку.

SOAP - протокол обмена структурированными сообщениями в распределенной вычислительной среде. Данные передаются в формате XML.

**Всегда имеет документы**:

1) **XSD (XML Schema Definition)** - описывает структуру XML документа и типы данных, которые там могут храниться

2) **WSDL (Webservice Description Language)** - описывает сообщения, заголовки, события, которые свойственны для веб-сервиса

**Отличия между SOAP и REST**

**REST** — это архитектурный стиль. **SOAP** — это формат обмена сообщениями (протокол). Специфика **SOAP** — это формат обмена данными. С SOAP это всегда SOAP-XML. Специфика REST — использование HTTP в качестве транспортного протокола.

**В SOAP** вы используете формат SOAP XML для запросов и ответов.

**В REST** такого фиксированного формата нет. Можно обмениваться сообщениями на основе XML, JSON или любого другого удобного формата.

**REST** обычно использует JSON, который легче анализировать и обрабатывать. В дополнение к этому, REST не требует наличия определения службы для предоставления веб-службы.

Однако в случае **SOAP** вам необходимо определить свой сервис с использованием WSDL, и при обработке и анализе сообщений SOAP-XML возникают большие накладные расходы.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **SOAP = XML = протокол**

**SOAP не может использовать REST**

**Но REST может использовать SOAP**

SOAP *более безопасен* и надежен/ больше весит/ менее производительный *(медленнее).*

SOAP проще тестировать и обязательно наличие файла WSDL с документацией!

1. **REST (Swagger) = JSON (чаще всего) = архитектурный стиль**

*REST* более производителен (*быстрее*)/ *менее безопасен*

Сложнее тестировать, может не быть документации

**Session storage**

**sessionStorage** — один из способов хранить данные в браузере.

Это объект, хранящийся в window, который позволяет сохранять данные в браузере на время сессии. Этот тип хранилища очень похож на localStorage и работает как хранилище данных в формате ключ-значение. При сохранении данных мы указываем имя поля, в которое должны быть сохранены данные, и затем используем это имя для их получения.

Сессия страницы создаётся при открытии новой вкладки браузера. *Сессия остаётся активной до тех пор, пока открыта вкладка, а состояние сессии сохраняется между перезагрузками*. Открытие новой вкладки с таким же адресом приведёт к созданию новой сессии.

Значения хранятся в виде строк. При попытке сохранения других типов данных, они будут приведены к строке. Например, если записать число, то при чтении нам вернётся число, записанное в строку.

Максимальный объем данных ограничен размером 5MB.

Данные **продолжают существовать после перезагрузки страницы**, но не после закрытия/открытия вкладки.\

**Объём данных больше чем в Cookie.**

Хранит данные только для сеанса, что означает, что **данные хранятся до закрытия браузера (или вкладки).**

***Данные никогда не передаются на сервер.***

**Cookies**

**Cookies** - небольшой фрагмент данных, отправленный веб-сервером и хранимый на компьютере пользователя.

Проще говоря, когда пользователь совершает на сайте какое-то действие, например добавляет товар в корзину или вводит логин и пароль для входа в аккаунт, сервер записывает эту информацию в куки и отправляет браузеру вместе со страницей. Эти и другие сведения о различных действиях собираются в текстовом документе **cookies.txt.**

Благодаря кукам при каждом посещении того или иного ресурса не нужно вводить регистрационные данные — браузер их подгружает самостоятельно. Поэтому нагрузка не сервер не возрастает и повышается скорость открытия веб-страниц.

**Виды куков:**

1. **Постоянные файлы cookie**: могут сохранять данные в течение длительного периода времени. Это файлы cookie, которые позволяют веб-сайтам хранить информацию об имени пользователя и пароле для пользователей.
2. **Сторонние файлы cookie**: сторонние файлы cookie ищут данные о вашей онлайн-активности, чтобы отправить их владельцам веб-сайтов, которые хотят улучшить рекламу.

1. **Сессионные куки**: удаляются сразу после закрытия браузера (session storage). Они наиболее известны тем, что позволяют вам сохранять товары в корзине даже после перехода на другую страницу.

**Cookie могут быть защищёнными, в этом случае их содержимое нельзя получить на стороне клиента. Это важно для аутентификации при хранении пользовательских токенов.**

**Cash Storage**

Браузер будет кэшировать веб-сайты, которые часто посещает пользователь. Таким образом, в следующий раз, когда пользователь посетит определенный веб-сайт, его содержимое будет загружено из кэша, а не с сервера.

CacheStorage (кеш-память) - это браузерный механизм для хранения и получения сетевых запросов и ответов. Он хранит пары объектов: запрос и ответ, запрос в качестве ключа и ответ в качестве значения.