API действует как интерфейс между двумя программными приложениями и позволяет им связываться друг с другом на оговоренных правилах и не залезая в реализацию предоставляемых функций.

Порты можно просканировать с помощью- netcat и Nmap

Frontend testing - это тип тестирования, который проверяет уровень представления 3-уровневой архитектуры. С точки зрения непрофессионала, вы проверяете GUI - все, что видно на экране, на стороне клиента. Для веб-приложения интерфейсное тестирование будет включать проверку функциональных возможностей, таких как формы, графики, меню, отчеты и т. д., а также связанный Javascript. Frontend testing - это термин, охватывающий различные стратегии тестирования, включая оценку производительности фронтенда, которая является хорошей практикой перед тестированием приложения с высокими пользовательскими нагрузками. Тестировщик должен хорошо понимать бизнес-требования для выполнения этого типа тестирования. Ранее оптимизация производительности означала оптимизацию на стороне сервера. Это было связано с тем, что большинство веб-сайтов были в основном статичными, а большая часть обработки выполнялась на стороне сервера. Однако сегодня веб-приложения становятся более динамичными и в результате код на стороне клиента нередко становится причиной низкой производительности. Тестирование клиентской части невозможно в некоторых случаях: бэкенд разрабатывают быстрее, чем фронтенд; очевидно, если клиентская часть отсутствует в принципе ( самодостаточное приложение, терминальная команда). Backend testing - это тип тестирования, который проверяет уровень приложений и базы данных 3-уровневой архитектуры. В сложном программном приложении, таком как ERP, внутреннее тестирование повлечет за собой проверку бизнес-логики на уровне приложений. Для более простых приложений бэкэнд-тестирование проверяет серверную часть или базу данных. Это означает, что данные, введенные в интерфейс, будут проверены в базе данных. Базы данных проверяются на наличие свойств ACID, операций CRUD, их схемы, соответствия бизнес-правилам. Базы данных также проверяются на безопасность и производительность. Производится проверка целостности данных, Проверка достоверности данных, Тестирование функций, процедур и триггеров. При внутреннем тестировании нет необходимости использовать графический интерфейс. Вы можете напрямую передавать данные с помощью браузера с параметрами, необходимыми для функции, чтобы получить ответ в некотором формате по умолчанию. Например, XML или JSON. Вы также подключаетесь к базе данных напрямую и проверяете данные с помощью SQL-запросов.

**http(протокол передачи гипер текста) hyper text transfer protocol**

**Работает на 7 уровне- прикладном.**

Протокол HTTP предполагает использование клиент-серверной структуры передачи данных. Клиентское приложение формирует запрос и отправляет его на сервер, после чего серверное ПО обрабатывает данный запрос, формирует ответ и передает его обратно клиенту. То есть этот протокол не только устанавливает правила обмена информацией, но и служит транспортом для передачи данных — с его помощью браузер загружает содержимое сайта на ваш компьютер или смартфон.

**Гипертекст-** это система текстовых страниц, соединённых между собой ссылками. Так же означает текст, созданный с помощью языка разметки и рассчитанный на использование гиперссылок.

Гипертекстом можно назвать любой http документ. Работает в режиме- запрос- ответ. Работает на TCP. Сейчас в ходу версия- 1.1. В 2015 появилась версия 2.0.

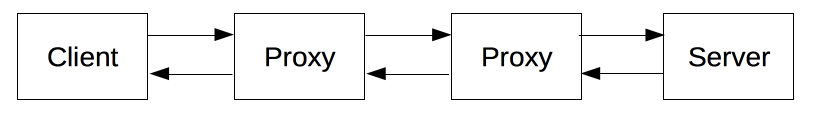
**http сообщение**- это обмен данными между сервером и клиентом.

Есть 2 типа сообщений: запросы- отправляемые клиентом, что бы инициировать реакцию со стороны сервера и ответы от сервера.

**http caching**- процесс сохранения часто запрашиваемых документов на прокси серверах или машине пользователя с целью предотвращения их постоянной загрузки с сервера.

**Как это работает?**

Каждый request отправляется серверу, который обрабатывает его и возвращает response. Между этими запросами и ответами как правило существуют многочисленные посредники, называемые proxy, которые выполняют различные операции и работают как шлюзы или кэш, фильтрация, сканирование антивирусом, родительский контроль, аунтефикация, протоколизация(разрешение на хранение истории операций).



Все сообщения http состоят в тестовой информации в кодировке SACII.

Для установки безопасного соединения используется протокол HTTPS с поддержкой шифрования. Защиту данных в HTTPS обеспечивает криптографический протокол SSL/TLS, который шифрует передаваемую информацию. По сути этот протокол является оберткой для HTTP. Протокол SSL/TLS хорош тем, что позволяет двум незнакомым между собой участникам сети установить защищенное соединение через незащищенный канал. При установке безопасного соединения по HTTPS ваш компьютер и сервер сначала выбирают общий секретный ключ, а затем обмениваются информацией, шифруя ее с помощью этого ключа. Общий секретный ключ генерируется заново для каждого сеанса связи. Его нельзя перехватить и практически невозможно подобрать — обычно это число длиной более 100 знаков. Этот одноразовый секретный ключ и используется для шифрования всего общения браузера и сервера. Для этой гарантии существует сертификат. Вам как пользователю сертификат не нужен, но любой сервер (сайт), который хочет установить безопасное соединение с вами, должен его иметь. Сертификат подтверждает две вещи: 1) Лицо, которому он выдан, действительно существует и 2) Оно управляет сервером, который указан в сертификате. Выдачей сертификатов занимаются центры сертификации. Как и в паспорте, в сертификате содержатся данные о его владельце, в том числе имя (или название организации), а также подпись, удостоверяющая подлинность сертификата. Проверка подлинности сертификата — первое, что делает браузер при установке безопасного HTTPS-соединения. Обмен данными начинается только в том случае, если проверка прошла успешно.

**Компоненты HTTP**

**Примеры HTTP запросов:**

**HTTP определяет следующую структуру запроса (request):**

стартовая строка (starting line) — определяет тип сообщения, имеет вид Метод URI HTTP/Версия протокола, например GET /web-programming/index.html HTTP/1.1

заголовки запроса (header fields) — характеризуют тело сообщения, параметры передачи и прочие сведения

тело сообщения (body) — необязательное

* HTTP-метод, обычно глагол подобно GET, POST или существительное, как OPTIONS или HEAD, определяющее операцию, которую клиент хочет выполнить. Обычно, клиент хочет получить ресурс (используя GET) или передать значения HTML-формы (используя POST), хотя другие операция могут быть необходимы в других случаях.
* Путь к ресурсу: URL ресурсы лишены элементов, которые очевидны из контекста, например без протокола (http://), домена (здесь developer.mozilla.org), или TCP порта (здесь 80).
* Версию HTTP-протокола.
* Заголовки (опционально), предоставляющие дополнительную информацию для сервера.
* Или тело, для некоторых методов, таких как POST, которое содержит отправленный ресурс.



**HTTP определяет следующую структуру ответного сообщения (response):**

строка состояния (status line), включающая код состояния и сообщение о причине

поля заголовка ответа (header fields)

дополнительное тело сообщения (body)

* Версию HTTP-протокола.
* HTTP код состояния, сообщающий об успешности запроса или причине неудачи.
* Сообщение состояния — краткое описание кода состояния.
* HTTP заголовки, подобно заголовкам в запросах.
* Опционально: тело, содержащее пересылаемый ресурс.



**Методы http запросов.**

Два обязательных метода, GET и HEAD, никогда не должны быть деактивированы и не должны возвращать этот код ошибки.

Метод, используемый в HTTP-запросе, указывает, какое действие вы хотите выполнить с этим запросом.

**GET-** получение ресурса, извлечение данных. Не имеет тела. Могу только извлекать данные.

**POST**- для передачи пользовательских данных к ресурсу. Сообщение ответа сервера не выполнение метода. post не кешируется. Создание ресурса

**PUT**- обновление ресурса, перезапись уже имеющейся на сервере инфы. Обновление/перезапись

**DELETE**- удаляяет указанный ресурс

**PATCH**- частично изменяет ресурс. Может иметь побочные эффекты на другие ресурсы. Больше подходит для обновления ресурсов. Изменяет отдельные поля ресурса. Не идемпотентный. Можно обновить и через PUT одно поле, но метод PUT будет проходить все поля объекта и искать необходимое, в отличие от patch, который не обходит объект целиком.

**HEAD**- метод запрашивает только информацию о файле и ресурсе, нечего кроме заголовков не возвращает. Используется когда нужно найти информацию о документе, не получая его. Без тела ответа.

**Trace**- отслеживать изменения.

**Options**- узнать методы поддерживаемые для ресурса

**Cennect**- подключение к веб серверу через прокси

**Безопасные:** get, head, options, trace

**Небезопасные:** post, put, delete

**Идемпотентные:** get, head, put, delete, options, trace

**Кешируемые:** get, head. Post, patch- если установлен заголовок(Content- location(en-US)

**Некешируемые:** put, delete

**Безопасный метод**- является безопасным, если он не меняет состояние сервера. Проводит операцию только чтение. Можно выполнять в любой последовательности.

**Идемпотентность(идентичное действие).** Если повторный идентичный запрос, сделанный один или несколько раз подряд имеет один и тот же эфект, не изменяет состояние сервера.

Так же все безопасные методы являются идемпотентными. Одно и тоже действие с одинаковыми входными параметрами получается одинаковый результат. Для идемпотентности нужно рассматривать только изменения фактического внутреннего состояния сервера, а возврающиеся запросами статус коды могут отличаться. Первый вызов DELETE вернёт код 200, в то время как последующие вызовы вернут код 404.

**OSI(Open System Interconnection)**

**PDU(protocol data unit)**

7. Application layer(Прикладной). Обеспечивает взаимодействие сети и пользователей. Задача- использовать свои протоколы, что бы пользователь увидел данные в понятном ему виде. Доступ к сетевым службам. HTTP, FTP, POP3, WebSocket

6. Presentation layer(Представления). Представление и шифрование данных.

Представляет данные в понятном для человека виде и машине. Занимается шифрованием. Отвечает за переобразование протоколов в кодирований/декодирования данных. Когда мы пересылаем картинку- уровень представления разбивает её на отдельные составляющие биты и когда мы получаем то из этих битов на этом уровне она составляется в готовую. Использует TLS/SSl

5. Session layer(Сеансовый )- организует сеанс связи между ПК. Отвечает за поддержание сеанса связи. Определяет как будет передача инфы: полудуплексно или дуплексно( и приём и передача).

4. Transport layer(Транспортный) – PDU: Сегменты (segment) /Дейтаграммы (datagram). Обеспечивается надёжность передачи информации.

**UDP(User datagram protocol)-** Без установки соединения и подтверждения доставки данных и не делает повтор. При передачи по UDP- пакеты данных делятся на датаграмы.

**TCP(transmission control protocol)-** отвечает за передачу данных по средствам установления соединения, подтверждает передачу, при необходимости делает повтор, гарантирует целостность и правильность загружаемых данных. Данные делятся на сегменты. Сегмент- это часть пакета. Когда приходит пакет данных, который привышает пропускную способность сети, пакет делится на сегменты допустимого размера. Появляется понятие портов. OSI можно разделить на 2 группы. Media(l1,l2,l3) layer и host layer(4,5,6,7)- используются непосредственно на устройвах, пк, мобилы. Media layer- занимается передачей информации по кабелю или безпроводным сетям.

3. Network layer(Сетевой). PDU: Пакеты (packet). Предназначен для определения пути передачи данных. Определяет кротчайшие маршруты, коммутацию и маршрутизацию. На этом уровне работает маршрутизатор. Активно используется ARP(address resolution protocol). С его помощью 64- битные MAC адреса преобразуется в 32 битные ip адреса и наоборот и обеспечивает инкапсуляцию и декапсуляцию данных. Наиболее распространённым протоколом является ip. ipv4, ipv6.

2. Data link layer(Канальный уровень)- PDU: Биты (bit)/ Кадры (frame). На этом уровне появляется адресация, мак адрес. Канальный отвечает за доставку кадров адресанту и их целостность. Работает протокол ARP- для определения соотвествия между ip(логический адрес) и физическим адресом устройвам(MAC). Используются протоколы PPP, Ethernet.

1. Physical layer(Физический уровень). PDU: биты(bit). Отвечает за обмен физическими сигналами между физическими устройствами. Основная задача- представить 0 и 1 в качестве сигналов.

**TCP/IP** — сетевая модель передачи данных, представленных в цифровом виде. Модель описывает способ передачи данных от источника информации к получателю.

**TCP/IP —** сетевая модель передачи данных, представленных в цифровом виде. Модель описывает способ передачи данных от источника информации к получатНабор интернет-протоколов обеспечивает сквозную передачу данных, определяющую, как данные должны пакетироваться, обрабатываться, передаваться, маршрутизироваться и приниматься. Эта функциональность организована в четыре слоя абстракции, которые классифицируют все связанные протоколы в соответствии с объемом задействованных сетей. От самого низкого до самого высокого уровня:

уровень связи, содержащий методы связи для данных, которые остаются в пределах одного сегмента сети;

интернет-уровень, обеспечивающий межсетевое взаимодействие между независимыми сетями;

транспортный уровень, обрабатывающий связь между хостами;

прикладной уровень, который обеспечивает обмен данными между процессами для приложений.

**Инкапсуляция**- это включение сообщения вышестоящего уровня в сообщение нижестоящего уровня. Сообщения состоит из заголовка, данных и концевика. Когда передающая сторона готовит данные к передаче- это процесс инкапсуляции и наоборот когда принимающая сторона начинает брабатывать входную последовательность бит и формирует из неё сообщение- декапсуляция.

**Инкапсуляция**- это процесс в котором некоторая дополнительная инфа добавляется к элементу данных, что бы добавить к нему некоторые фукнции. Инкапслуяция данных добавляет к данным информацию протокола, что бы передача данных могла происходить надлежащим образом.

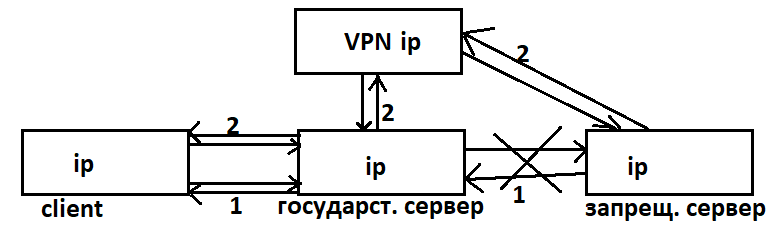
Сниферы- это проги, которые перехватывают весь сетевой трафик. Снифферы полезны для диагностики сети.

**VPN(virtual private network)-** шифрует трафик. Tracert

Это совокупность технологий, позволяющих обеспечить одно или несколько сетевых соединений(логическую сеть) поверх другой(физической)

Соединение VPN- это туннель между пк юзера и компьютеров- сервером

Каждый узел шифрует данные до их попадания в туннель.



**NAT(network address translation)-** переводит приватные адреса в общедоступные. Позволяет устройствам с частным адресом обращаться к ресурсам за пределами его частной сети.

**Proxy-** это удалённый компьютер посредник для выхода пользователя в сеть. Его задача состоит в том, что бы все запросы от пользователя предавать в сеть Интернет через него, а полученные ответы транслировать обратно. Есть публичные, пакетные, приватные, совместные прокси.

**Proxy server-** промежуточный сервер в компьютерных сетях, выполняющий роль посредника между пользователем и целевым сервером, позволяющий клиентам как выполнять косвенные запросы к другим сетевым службам, так и получать ответы.

**Сетевой шлюз-** аппаратный маршрутизатор или программное обеспечение для сопряжения компьютерных сетей, использующих разные протоколы. Служит мостом между локальной и глобальной сетью.

**Ethernet-** это стандарт, который относится только к построению LAN. Это набор описаний способов физической передачи сигналов(l1) и формирование кадров на(l2).

**Ethernet** — протокол, определяющий стандарты сети на физическом и канальном уровнях. остается легко читаемым, даже при больших массивах информации.

**Ip(internet prtotocol)-** основная задача- адерсация, перенаправление пакетов от хоста отправителя к хосту получателя. Protocol ip- не подтверждает доставку данных, не контролирует данные. Предназначен для определения адерсата и доставки ему информации. Адресация в сети Internet представляет собой 32 битную последовательность 0 и 1- называющихся ip адресами

**Маска подсети позволяет** выполнить разбиение одной большой подсети на несколько маленьких и тем самым упросить маршрутизацию. Маска не является частью ip- пакета. Помогает маршрутизатору понять, как и куда предавать пакеты. Протокол ip использует маршрутизацию что бы определить какой подсети пренадлежит получатель.

**Маршрутизация-** определяет лучший путь добавки пакета информации получателю. Есть 2 вида- статическая и динамическая. Состоит из 2 этапов- на 1 этапе происходит изучение сети, какие подсети есть в этой сети, какие маршрутизаторы и как они объеденены между собой. На втором когда сеть уже изучена и на маршрутизатор поступил пакет, для этого пакета нужно определить куда именно его отправить.

**Протокол передачи данных**- набор правил который регламентирует порядок и особености передачи информации, между 2 и более включёнными в сеть пк.

**Redirect-** перенаправление на другой ip, другой сревер или другой порт.

**Websocket** - протокол в основании которого лижет Http протокол. Позваоляет работать с двух направленным потоком данных. Обеспечиывает возможность обмена данных между клиентом и сервером через постоянное соединение. Достаточно 1 раз выполнить запрос и ждать отклика. Для того что бы получать ответы от сервера, не нужно с клиента запрашивать по каждому запросу- получать ответ. Использует TCP. Websocket- протокол связи поверх TCP

**Комбинация ip и порта- называется сокетом.**

**Порт применяет** для идентификации программы или процесса для обмена пакетами данных в рамках 1 ip адреса. Идентификатор. От 0 до 65535.

**Протоколы:**

**FTP (File Transfer Protocol)** — это протокол передачи файлов со специального файлового сервера на компьютер пользователя. FTP дает возможность абоненту обмениваться двоичными и текстовыми файлами с любым компьютером сети. Установив связь с удаленным компьютером, пользователь может скопировать файл с удаленного компьютера на свой или скопировать файл со своего компьютера на удаленный.

**POP3 (Post Office Protocol**) — это стандартный протокол почтового соединения. Серверы POP обрабатывают входящую почту, а протокол POP предназначен для обработки запросов на получение почты от клиентских почтовых программ.

**SMTP (Simple Mail Transfer Protocol)** — протокол, который задает набор правил для передачи почты. Сервер SMTP возвращает либо подтверждение о приеме, либо сообщение об ошибке, либо запрашивает дополнительную информацию.

**TELNET** — это протокол удаленного доступа. TELNET дает возможность абоненту работать на любой ЭВМ находящейся с ним в одной сети, как на своей собственной, то есть запускать программы, менять режим работы и так далее. На практике возможности ограничиваются тем уровнем доступа, который задан администратором удаленной машины.

**Локаль-** это набор наддных указывающих на региональные стандарты.

**UTM-** метки- фрагмент кода, которые добавляются в Url. Каждый параметр в utm метке состоит из 2 частей- название параметра и его значение.

Utm метки помогают узнать, из какого источника, канала и компании пользователи переходят на наш сайт. Можно оценить количество трафика на странице и откуда он идёт. Сколько пользователи провели времени на странице и куда перешли с енё. Какой элемент страницы- лучше конвертирует.

**Логи. Есть 3 вида- логи приложения, логи сервера, логи системы.**

Уровни логирования:

1/ off(не записываются)

2 Fatal(блокирующие)

3 Error(ошибка но система работает)

4 Warm (предупреждение)

5 Info( создание/удаление)

6 Debug(отладка, вкобчает все предведущие уровни)

7 Trace access logs- запрос пользователей, дополнительные хедеры, юзер агент, код ответа, тайминги.

**Cache — это временное хранилище для данных (перечень определен создателем сайта) с посещенного сайта.**

**Cache- делится на Private(кеш браузера), Общий(shader),**

Общий кэш сохраняет ответы, что бы их потом могли использовать разные пользователи. Это может быть proxy сервер.

Private cache- позволяет сохранить страницы не обращаясь лишний раз к серверу.

Управление кешированием происходит при помощи заголовков

Срок действия кеширования исчисляется в секундах

Cache- control: max-age=31536000(срок действия expiration)

При первом запросе отправляем запрос и он проходит через наш кэш, понимаем что кэш пустой и отправляем на сервер, сервер отправляет нам ответ и пишет что срок кэша такой то( 100с)

Если в ответе не присутствует валидатор(E-tag или last- modified заголовок) и он не содержит никакой явной информации о свежести- не кэшируется

**Зачем чистить кеш?** Веб страницы могут отображаться некоректно в связи с тем, что в них были внесены изменения, а браузер продолжает использовать устаревшие данные из кеша. С накопление кеша место на пк, незначительно уменьшается.

**Cookies.** Файл cookie HTTP (файл cookie Интернета, файл cookie браузера) представляет собой небольшой фрагмент данных (часть http заголовка), который веб-сервер хранит в текстовом файле на жестком диске пользователя (клиента). Эта часть информации затем отправляется обратно на сервер каждый раз, когда браузер запрашивает страницу с сервера. Обычно cookie-файлы содержат персонализированные пользовательские данные или информацию, которые используются для определения того, поступили ли два запроса от одного и того же браузера - например, для входа пользователя в систему или для связи между различными веб-страницами.

**Куки в основном используются для трех целей:**

* Управление сессиями: Логины, корзины покупок, результаты игр и все, что сервер должен запомнить
* Пользовательские настройки, темы и другие настройки
* Запись и анализ поведения пользователя

**Куки состоят в основном из трех вещей:**

* Имя сервера, с которого был отправлен куки
* Время жизни (Cookies Lifetime)
* Случайно сгенерированный уникальный номер

**Виды кук:**

**Сессионные cookie**, также известные как временные cookie, существуют только во временной памяти, пока пользователь находится на странице веб-сайта. Браузеры обычно удаляют сессионные cookie после того, как пользователь закрывает окно браузер. В отличие от других типов cookie, сессионные cookie не имеют истечения срока действия, и поэтому браузеры понимают их как сессионные. Атрибут Expires или max-age для него не задаются.

**Постоянные cookie**-файлы удаляются в определенную дату или через определенный промежуток времени. Это означает, что информация о cookie будет передаваться на сервер каждый раз, когда пользователь посещает веб-сайт, которому эти cookie принадлежат. По этой причине постоянные cookie иногда называются следящие cookie, поскольку они могут использоваться рекламодателями для записи о предпочтениях пользователя в течение длительного периода времени. Однако, они также могут использоваться и в «мирных» целях, например, чтобы избежать повторного ввода данных при каждом посещении сайта. Удаляются при наступлении определённой даты(атрибут expires) или после определённого интервала времени( атрибут max-age).

Поскольку cookie можно очень легко удалить из браузера, программисты ищут способы идентифицировать пользователей даже после полной очистки истории браузера. Одним из таких решений **являются зомби-cookie (или evercookie, или persistent cookie)** — не удаляемые или трудно удаляемые cookie, которые можно восстановить в браузере с помощью JavaScript. Это возможно потому, что для хранения куков сайт одновременно использует все доступные хранилища браузера.

**Как тестировать cookies.**

1. Убедиться что в куки нет конфиденциальной инфы
2. Проверить количество используемых куки(лучше меньше)
3. Убедиться что приложение сохраняет свою работоспособность при отключённом либо выборочном приёме куки
4. Убедиться что приложение адеквтано реагирует на пудаление куки в ручную
5. Проверить реакцию на повреждённые куки
6. Уебидться что куки записываются и удалюятся именно на тех страницах на которых это ожидаемо и необходимо
7. Проверить что куки корректно работают во всех браузерах с которыми будет использоваться приложуха
8. Проверить что доступ к различным частям приложения не может быть осуществлён в обход куки используемых для аунтификации или авторизации.

**Примеры Test case для Cookie testing:**

* Отключение файлов cookie: отключите все файлы cookie и попытайтесь использовать основные функции сайта
* Поврежденные файлы cookie: вручную отредактируйте файл cookie в блокноте и измените параметры на несколько случайных значений
* Шифрование куки: конфиденциальная информация, такая как пароли и имена пользователей, должна быть зашифрована
* Тестирование файлов cookie в нескольких браузерах. Убедитесь, что с вашего веб-сайта правильно записываются cookie в разных браузерах
* Проверка удаления куки с веб-сайта
* Удаление файлов cookie: удалите все файлы cookie для веб-сайтов и посмотрите, как веб-сайт среагирует
* Доступ к файлам cookie: файлы cookie, написанные одним сайтом, не должны быть доступны другим
* Не допускайте чрезмерного использования файлов cookie: если тестируемое приложение является общедоступным веб-сайтом, не следует злоупотреблять файлами cookie.
* Тестирование с другими настройками. Тестирование должно выполняться правильно, чтобы убедиться, что веб-сайт работает хорошо с другими настройками файлов cookie.
* Категоризируйте куки отдельно: куки не должны храниться в той же категории вирусов, спама или шпионских программ

**Зачем чистить куки?** Воспользовавшить пк, посторонний чел(Севун- Сисьпензия какая-то) может посещать формумы и сайты под именем владельца пк. Боль Я ЗАЕБАЛСЯ большое количество сохраняемых кук ведет к сокращению места на жёстком диске. Пользуясь куки, можно проследить историю перемещения в сети, что нежелательно с точки зреня конфиденциальности.

**Web Storage.** Интернет-хранилище представляет собой постоянное хранилище данных, похожее на куки, но со значительно расширенной емкостью и без хранения информации в заголовке запроса HTTP. Существуют два основных типа веб-хранилища: локальное хранилище (localStorage) и сессионное хранилище (sessionStorage), ведущие себя аналогично постоянным и сессионным кукам соответственно.

**Локальное хранилище- это объект Local storage**, который позволяет хранить данные для каждого домена в течении неограниченного времени(после закрытия вкладки браузера, самого браузера, перезагрузки пк), а точнее пока пользователь самостоятельно не удалит данные. local storage нужен для хранения данных объекта между сессиями клиента.

**Сессионное хранилище- это объект session storage**, который позволяет хранить данные для каждого домена, доступные на протяжении сессии, пока открыт браузер и даже в том случае, если была осуществлена перезагрузка страницы. Однако закрытие вкладки браузера, самого браузера или перезагрузка пк приводит к удалению данных из session storage.

Объект session storage используется гораздо реже, чем Local storage.

**URI(uniform resource identifier) унифицированый идентификатор ресурса.** Последовательность символов, которая индетифицирует, кокой то ресурс по его адресу и по его имени. Содержит URL и URN

**URL(uniform resource locator)- унифицированый определитель местонахождения ресурса**. По его адресу и способ обращения к нему

**URN(uniform resource name)- для идентификации ресурса, но только по его имени( не важно где находится ресурс).** Определяет только название ресурса но не говорит как к нему подключится и не укзаывает его место нахождения.

URL: вся ссылка

URL: <https://www.google.com:80/> до якоря

URN: /catalog/product?27... / до якоря

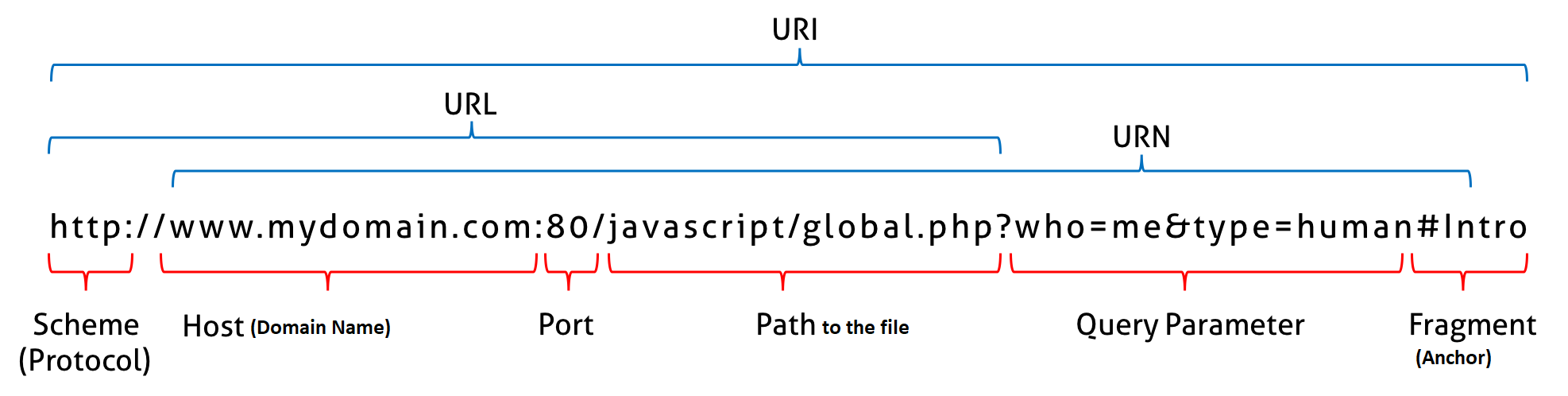
**URI содержит в себе следующие части:**

**Схема (scheme)** - показывает на то, как обращаться к ресурсу, чаще всего это сетевой протокол (http, ftp, ldap)

**Иерархическая часть (hier-part)** - данные, необходимые для идентификации ресурса (например, адрес сайта)

**Запрос (query)** - необязательные дополнительные данные ресурса (например, поисковой запрос)

**Фрагмент (fragment)** – необязательный компонент для идентификации вторичного ресурса ресурса (например, место на странице)



**Идентификация( id фикация)**- процесс распознавания пользователя по его идентификатору. Login

**Аутентификация**- процедура проверки подлинности, доказательство что юзер именно тот за кого себя выдаёт. Например проверка подлинности пользователя путём сравления введённого им пароля с паролем, сохранённым в базе данных. Есть 1-2 уровневая, много уровневая. Password. Запускается один раз для текущей сессии.

**Авторизация**- предоставление определённых прав на выполнение действий или доступа. Происходит при попытке совершения любых действий пользователем.

**API(application programming interface) програмный интерфейс приложения.** Составляющая часть сервера, которая получает запросы и отправляет ответы. Как ко мне, к моей системе можно обратиться? API- интерфейс совокупности методов и правил взаимодействия. Что бы фронт мог стучаться в бэк и получать ответы. Способ взаимодействия 1 системы с другой. Это программный интерфейс, который позволяет двум приложениям взаимодействовать друг с другом без какого-либо вмешаетльства пользователя. API предоставляет продукт или услугу для связи с другими продуктами и услугами, не зная, как они реализованы. Как один фрагмент кода разговаривает с другим. В веб разработке API часто относится к способу, которым мы получаем информацию из онлайн- сервиса.

Интерфейс- это прослойка между приложением А и приложением Б. В них происходят процессы, которые позволяют 2 программа обмениваться информацией и выполнять функции связанные с обеими сторонами. Инкапсуляция сокрытие части функций ради упрощения работы в целом и минимализации участков программного обеспечения, где один из разработчиков мог бы допустить ошибку.

Программные интерфейсы используются для того что бы делать программы надёжней, для масштабируемости, упрощение настройки между разными серверами и программами.

Веб сервис(служба)- программа, которая организовывает взаимодействие между сайтами. Информация с одного портала передаётся на другой. Упрощают обмен информацией между разными площадками. Задача веб сервисов- помогают интергрировать процессы, без участия людей. Если в компании используются корпоративные программы, то веб сервис поможет настроить их совместную работу.

Главное отличие веб-сервиса от других способов передачи данных: стандартизированность. Приняв решение использовать веб-сервисы, можно сразу переходить к структуре данных и доступным функциям.

Web- API Веб сервис.- Набор протоколов и стандартов, со скрытым исходным кодом, который используется для обмена данными между системами или приложениями. Все веб сервисы являются API. Он поддерживает XML. Он может использоваться любым клиентом, который понимает XML. Веб сервис использует три стиля: REST, SOAP и XML-RPC для связи. Они обеспечивает поддержку только для протокола HTTP. Поддерживает только протокол http.

Все веб сервисы- являются API интерфейсами, но не все API являются веб сервисами.

API

**Включает в себя:**

* - Саму операцию, которую мы можем выполнить
* - данные которые поступают на вход
* - данные которые оказываются на выходе

Есть 2 типа API- local and remote api.

remote api- SOAP(протокол) и REST(архитектурный стиль)

API появляется раньше gui.

**Как работает API:**

* 1 вызов операции- методом get,post
* 2 входные параметры- http request
* 3 выходные данные- http response

**Способы вызова API:**

* вызов функции системой
* вызов метода другой системы
* вызов метода человеком
* gui > api

**У API есть endpoint.** Смысл в том, что сайт, написанный на любом языке, поддерживающем HTTP запросы, не посылает на сервер никаких PHP/C/Python команд, а общается ним с помощью запросов, описанных в API. Адрес, на который посылаются сообщения называется endpoint. Обычно это URI (например, название сайта) и порт.

Если я хочу создать веб сервис на порту 8080, Endpoint будет выглядеть так: http://vladislaveremeev.ru:8080 Если моему Web сервису нужно будет отвечать на различные сообщения я создам сразу несколько URL (interfaces) по которым к сервису можно будет обратиться. Например:

https://vladislaveremeev.ru:8080 /resource1/status

https://vladislaveremeev.ru:8080 /resource1/getserviceInfo

https://vladislaveremeev.ru:8080 /resource1/putID

http://vladislaveremeev.ru:8080 /resource1/eventslist

https://vladislaveremeev.ru:8080 /resource2/putID

Итак, простейший запрос состоит из метода и Endpoint Request = Method + Endpoint Ресурс — это ключевая абстракция, на которой концентрируется протокол HTTP. Ресурс — это все, что вы хотите показать внешнему миру через ваше приложение

Выполняет конкретную задачу, принимает параметры и возвращает данные клиенту. Адрес функции. Способ взаимодействия 1 системы с другой.

Каждый endpoint- может быть отдельным серверов(зависит от архитектуры)

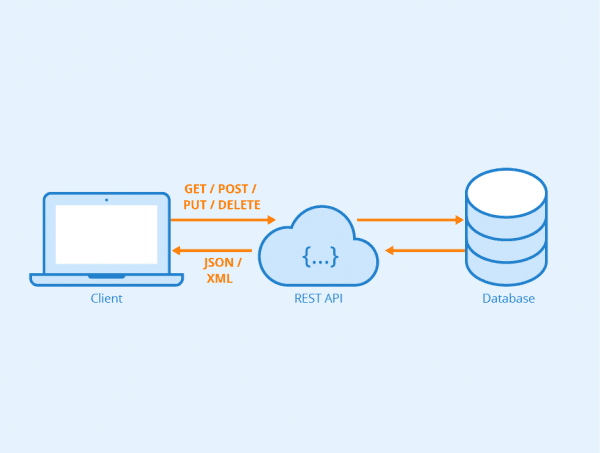
**Клиент серверная архитектура в связке с API**

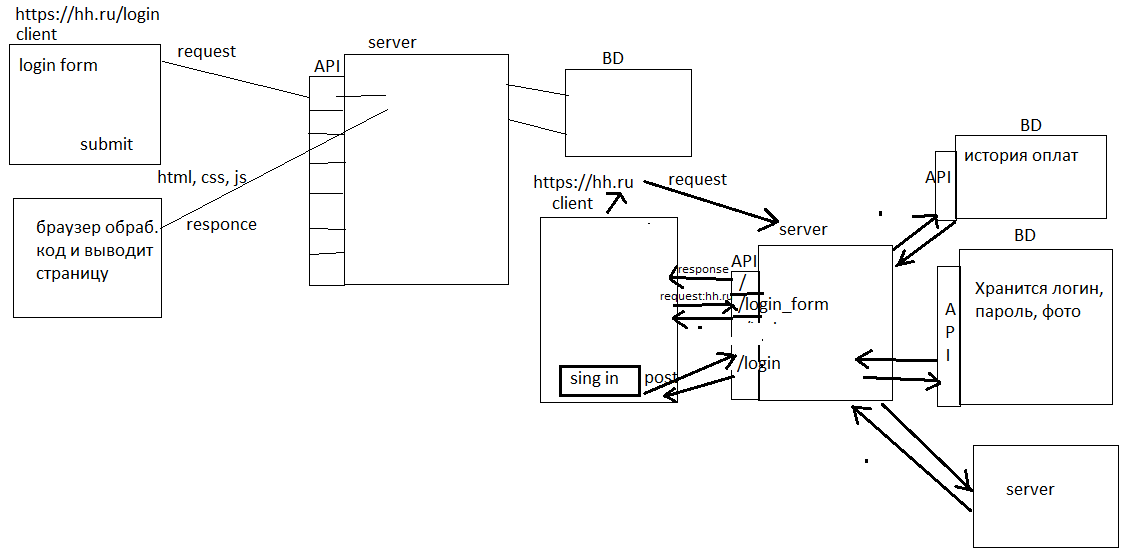
**Клиент**- браузер, локальный пк на стороне пользователя который выполняет отправку запроса к серверу и получает его.

**Сервер**- всё у чего есть сетовой интерфейс и к чему можно обратиться. Апаратный или облачный сервер, который принимает запрос и выполняет его и отправляет обратно ответ.

**Работает так:**

* Клиент отправляет запрос к серверу- response
* Сервер принимает обращение с требованием выполнить определённые действия и выполняет поставленную задачу
* Сервер отправляет клиенту результат выполненной работы- response





**SOAP (Simple Object Access Protocol)** — стандартный протокол обмена структурированными сообщениями в распределенной вычислительной среде. Данные передаются в XML. Используется для обмена произвольными сообщениями в формате XML. Способ взаимодействия информационной системы через web с другими информационными системам. Тоесть мы должны куда то передать XML и получить назад так же XML. Для передачи нужны протоколы: smtp, ftp, http. Пример реализации SOAP: XML поверх SOAP через HTTP. Формат обмена сообщениями происходит с использованием SOAP-XML. SOAP — это формат обмена сообщениями. SOAP использует WSDL (Web Services Description Language) — язык описания веб-сервисов и доступа к ним, основанный на языке XML. SOAP не накладывает никаких ограничений на тип транспортного протокола. Вы можете использовать либо Web протокол HTTP, либо MQ.

Специфика SOAP — это формат обмена данными. С SOAP это всегда SOAP-XML, который представляет собой XML, включающий:

— Envelope (конверт) – корневой элемент, который определяет сообщение и пространство имен, использованное в документе,

— Header (заголовок) – содержит атрибуты сообщения, например: информация о безопасности или о сетевой маршрутизации,

— Body (тело) – содержит сообщение, которым обмениваются приложения,

— Fault – необязательный элемент, который предоставляет информацию об ошибках, которые произошли при обработке сообщений. И запрос, и ответ должны соответствовать структуре SOAP.

Пример SOAP запроса:



Пример SOAP ответа:

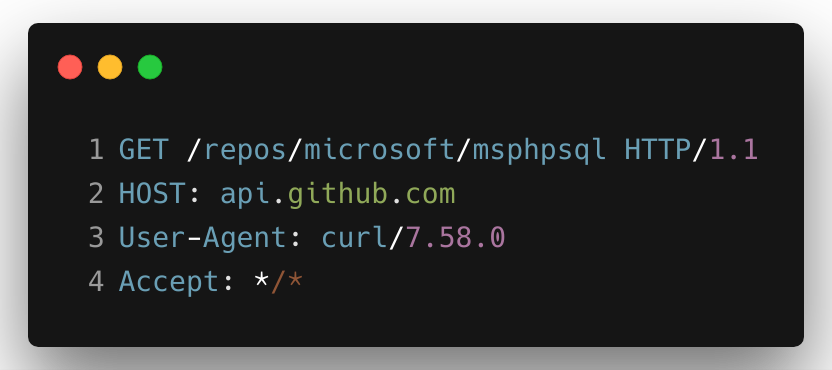


**REST (Representational State Transfer)** — архитектурный стиль взаимодействия компьютерных систем в сети основанный на методах протокола HTTP. Данные по умолчанию передаются в JSON. REST — это архитектурный подход, ориентированный на использование HTTP в качестве транспортного протокола. REST подразумевает наилучшее использование транспортного протокола HTTP

REST не использует конвертацию данных при передаче, данные передаются в исходном виде — это снижает нагрузку на клиент веб-сервиса, но увеличивает нагрузку на сеть. Управление данными происходит с помощью методов HTTP

**REST (Representational State Transfer)** — архитектурный стиль взаимодействия компьютерных систем в сети основанный на метоОсновное различие между SOAP и REST заключается в степени связи между реализациями клиента и сервера. Клиент SOAP работает как пользовательское настольное приложение, тесно связанное с сервером. Между клиентом и сервером существует жесткое соглашение, и ожидается, что все сломается, если какая-либо из сторон что-то изменит. Вам нужно постоянное обновление после любого изменения, но легче определить, выполняется ли контракт. REST-клиент больше похож на браузер. Это универсальный клиент, который знает, как использовать протокол и стандартизированные методы, и приложение должно соответствовать этому. дах протокола HTTP. Данные по умолчанию передаются в JSON. REST- это не протокол и не стандарт, а архитектурный стиль. RESR- это набор rules/standards/guidelines для создания веб-API. Важным компонентом для REST являются ресурсы.

Пример REST запроса:



Пример REST ответа:



Ресурс- это ключевая абстракция, но которой концентрируется протокол http.

Ресурс- это всё, что вы хотите показать внешнему миру через ваше приложение.

Действительно, если любым объектам вашего сервиса не нужны более сложные взаимоотношения, кроме: «Создать», «Прочитать», «Изменить», «Удалить» (как правило — в 99% случаев этого достаточно), возможно, именно REST станет правильным выбором. Кроме того, REST по сравнению с SOAP, может оказаться и более производительным, так как не требует затрат на разбор сложных XML команд на сервере (выполняются обычные HTTP запросы — PUT, GET, POST, DELETE). Хотя SOAP, в свою очередь, более надежен и безопасен.

SOAP работает с различными протоколами. REST только с http. SOAP нельзя кэшировать, а REST- можно. REST= скорость, разширяемость, поддержка многих форматов. SOAP больше возможностей по безопасности, так же больше возможности провести тестирование качественно, так как сразу в SOAP Будут WSDL, где будут прописаны образцы запросов. REST позволяет записывать информацию в более удобном формате, который занимает меньше места и повышает производительность системы.

Если REST- архитектурный стиль, с помощью которого у нас описывается структура передачи информации у веб сервисов, какие то правила- как эта инфа будет передаваться, то RESTFUL- это характеристика веб сервисов. Тоесть это такие web сервисы, которые полностью отвечают требованиям REST, которые предьявляет этот архитектурный стиль. Тоесть REST описывает структуру нашего сервиса, а RESTFUL- это уже сам вебсервис, который отвечает требованиям REST.

**JSON (JavaScript Object Notation - обозначение объектов JavaScript**) - текстовый формат обмена данными, основанный на JavaScript (но он не зависит от языка). Объект JSON- это формат данных- ключ значение.

Масивы Json- являются упорядоченными колекциями и могут содержать значения различных типов данных.

JSON syntax

“:” отделить имя от значения

“,” разделять пары имя- значение

“{“ и “}” для объектов

“[“ и “]” для массивов

Основные типы данных JSON

Простые типы данных

Строка(String)- любая последовательность символов, вставленная между “”(двойными кавычками).

{

“color” : “Purple”

}

Число(Number):

1. Представлены в десятичной системе без предшетствующих нулей
2. Могут быть отрицательными
3. Или с плавающей точкой
4. Представлены в виде десятичной степени

{

“number\_1” : 210,

“number\_2” : -210,

“number\_3” : 21.05,

“number\_4” : 1.0E+2 (1 умножено на 10 во второй степени)

}

Boolean- логическое значение, может быть истинным или ложным

{

“visibility” : true

}

Пустое значение(Null)- Отсутствие значения, рассматривается как null

{

“visibility” : true,

“popularity” : , //пусто

“id” : 210

}

Сложные типы данных

Объект(Object)- представляет из себя неупорядоченный набор пар имя/значение, вставленный между {}(фигурными скобками).

Объект может содержать пустое значение(null) или другие пары имя/значение.

Несколько пар имя/значение разделены запятыми.

{

“visibility” : true,

“popularity” : “immense”,

“id” : 210

}

Массив(Array)

Упорядоченная коллекция значений

Заключены между квадратными скобками

Значение разделены запятыми

{

“ids” : [“1”,”2”,”3”]

}

// или

{

“ids” : [

{“id” : 1},

{“id” : 2},

{“id” : 3}

]

}

Схема Json(JSON Schema)- для семантики элементов json.

JSON схема это словарь, который позволяет аннотирвать, валидировать и описывать json документы.

JSON схема сама является JSON Объектом

JSON схема предлагает документацию читаемую равно и человеком и машиной

Официальные ресурс для поддержания JSON схемы- <http://json-schema.org>

Json Object

1 JSON объекты имеют сходство с Javascript объектами.

Можно получить доступ к значениям объекта двумя способами: Через точку и доступ через скобки

dot(.)notation “Доступ через точку”

var author = {

“name” : “Jon Doe”,

“age” : 35,

“rights” : [ “admin”, “editor”, “contributior” ]

}

Console.log( author.name );

// Результат:

Jon Doe

Bracket([])notation “Доступ через скобки”

Var author = {

“name” : “Jon Doe”,

“age” : 35,

“rights” : [ “admin”, “editor”, contributor” ]

}

Console.log( author [ “name” ] );

Console.log( author [ “age” ] );

// Результат:

Jon Doe

35

Так же перебирать значение объекта можно, используя цикл. Тем же способом, если бы мы итерировали по массиву.

Итерация по значениям объекта

Var author = {

“name” : “Jon Doe”,

“age” : 35,

“rights” : [ “admin”, “editor”, “contributor” ]

}

For (x in author)

{

Console.log(x + “-“ + (author[x]));

// Output

Name – Jon Doe

Age – 35

Right- admin, editor, contributor

2 JSON объекты заключены в фигурные скобки {}

3 JSON объекты записываются в виде пары: ключ- значение, отделенные друг от друга двоеточием.

4 Ключи должны быть типа String, то есть строка, а значения должны быть одним из шести разрешённых типов данных JSON(строка, цифра, объект, массив, Булевое значение, ничто(null)

5 Пары ключ- значение отделены друг от друга запятой

JSON массив

1 Json массив формируется при помощи использования квадратных скобок []

2 Json массив это упорядоченный список значений

3 JSON массив может хранить сразу несколько типов значения: строку, число, Boolean, Объект или даже другой массив

4 в Json массиве, значения перечисляются через запятую

Доступ к элементам массива по индексу

Var author = {

“name” : “Jon Doe”,

“age” : 35,

“rights” : [ “admin”, “editor”, “contributor” ]

}

X = author.rights[0];

//Output

Name – admin

Многомерный массив

Var author = {

“name” : “Jon Doe”,

“age” : “35,

“rights” : [ “admin”, [1,2,3]],

[ “editor”,[4,5,6]]

}

for (i in author.rights) {

for (j in author.rights [i]) {

X = author.rights[i] [j];

Console.log(x);

};

//Output

1 2 3 4 5 6

JSON.parse(): Преобразует строковое представление Json в javascript Значение, иными словами в Json object. При желании можем использовать функцию reviver, для трансформации объекта до его возвращения. JSON.parse(text[, reviver])

“Конвертация”

JSON.parse() без reviver

Var jsonString = ‘{ “foo” : 1}’;

Var obj = JSON.parse( jsonString );

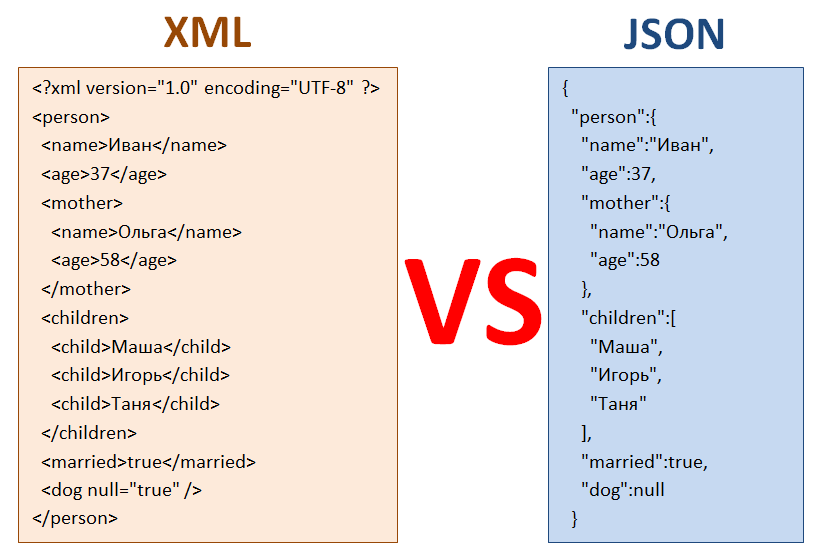
Var obj = JSON.parse(‘{ “x”: 5, “y”: 6 }’);

//Output

{foo: 1}

{x: 5, y: 6}

XML (eXtensible Markup Language — расширяемый язык разметки) - это язык разметки. Является выбором по умолчанию для обмена данными,Какие еще бывают протоколы?



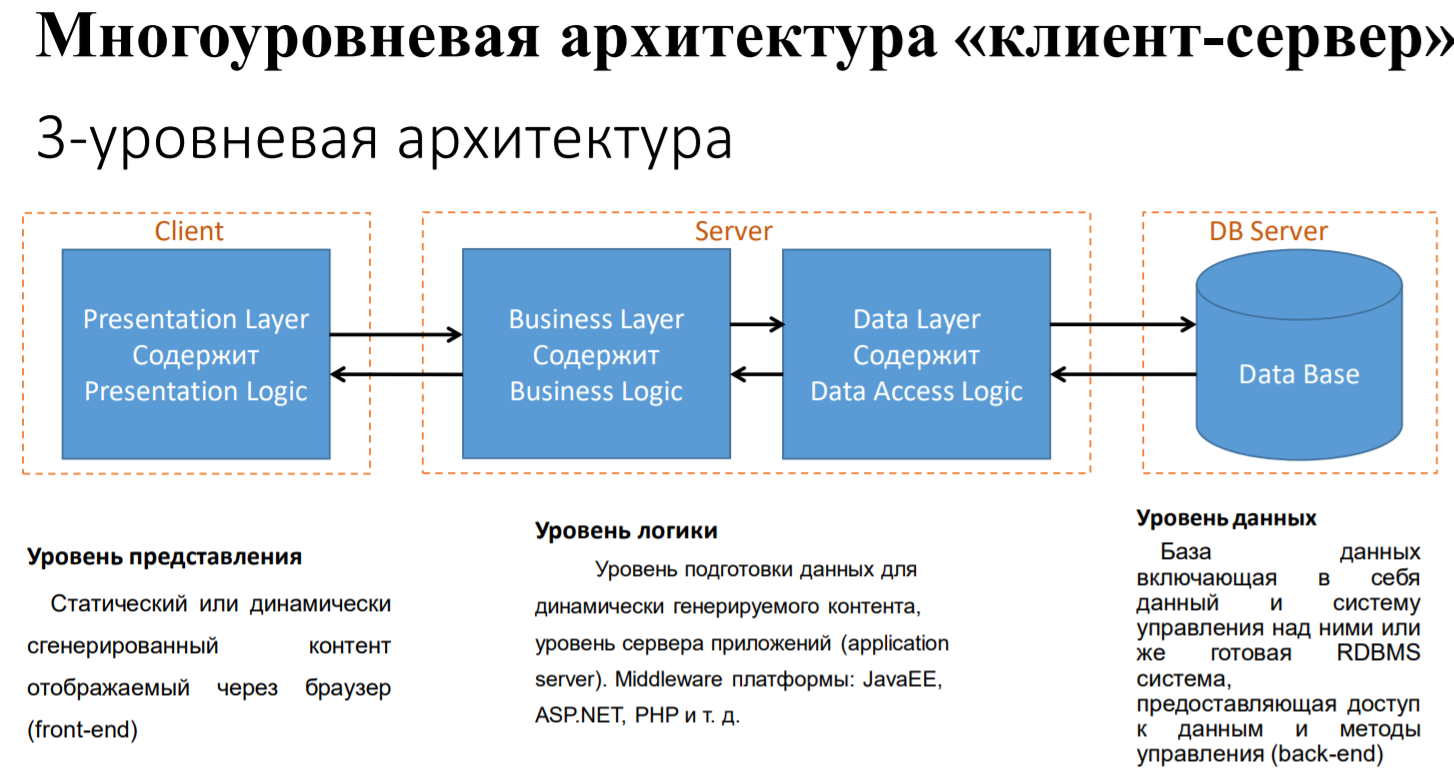
**Клиент - серверная архитектура состоит из:**

**Сервер** – логический процесс, который обеспечивает некоторый сервис по запросу от клиента. Обычно сервер не только выполняет запрос, но и управляет очередностью запросов, буферами обмена, извещает своих клиентов о выполнении запроса и т. д.

**Клиент** – процесс, который запрашивает обслуживание от сервера. Процесс не является клиентом по каким-то параметрам своей структуры, он является клиентом только по отношению к серверу.

**Сеть, протоколы** – третий компонент, который обеспечивает обмен информацией между клиентом и сервером.

Многоуровневая архитектура «клиент-сервер»: Разновидность архитектуры клиент-сервер, в которой функция обработки данных вынесена на один или несколько отдельных серверов. Это позволяет разделить функции хранения, обработки и представления данных для более эффективного использования возможностей серверов и клиентов.



**Ajax ( Asynchronous Javascript and XML — «асинхронный JavaScript и XML»)** — подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером. В результате при обновлении данных веб-страница не перезагружается полностью, и веб-приложения становятся быстрее и удобнее. По-русски иногда произносится транслитом как «аякс». У аббревиатуры AJAX нет устоявшегося аналога на кириллице.

**В классической модели веб-приложения:**

* Пользователь заходит на веб-страницу и нажимает на какой-нибудь ее элемент.
* Браузер формирует и отправляет запрос серверу.
* В ответ сервер генерирует совершенно новую веб-страницу и отправляет ее браузеру и т. д. , после чего браузер полностью перезагружает всю страницу.
* При использовании AJAX:
* Пользователь заходит на веб-страницу и нажимает на какой-нибудь ее элемент.
* JavaScript определяет, какая информация необходима для обновления страницы.
* Браузер отправляет соответствующий запрос на сервер.
* Сервер возвращает только ту часть документа, на которую пришел запрос.
* Скрипт вносит изменения с учетом полученной информации (без полной перезагрузки страницы).

**Как работает авторизация/аутентификация? Как сайт понимает, что ты залогинен?**

**Классический вариант** – регистрация по логину/почте и паролю. При входе и введении правильных данных, если данные совпадают с таковыми в базе, вы получаете доступ на сайт.

* Т.к. протокол HTTP не отслеживает состояния, нельзя достоверно знать, что человек, залогинившийся до этого по почте и паролю остается тем же человеком. И тогда изобрели аутентификацию на основе сессий/кук, на основе которых реализовано отслеживание состояний (stateful), то есть аутентификационная запись или сессия хранятся как на сервере, так и на клиенте. Сервер отслеживает открытые сессии в базе данных или в оперативной памяти, в свою очередь на фронтенде создаются cookies, в которых хранится идентификатор сессии. Процедура аутентификации на основе сессий:
* Пользователь вводит в браузере свое имя и пароль, после чего клиентское приложение отправляет на сервер запрос.
* Сервер проверяет пользователя, аутентифицирует его, шлет приложению уникальный пользовательский токен (сохранив его в памяти или базе данных).
* Клиентское приложение сохраняет токены в куках и отправляет их при каждом последующем запросе.
* Сервер получает каждый запрос, требующий аутентификации, с помощью токена аутентифицирует пользователя и возвращает запрошенные данные клиентскому приложению.
* Когда пользователь выходит, клиентское приложение удаляет его токен, поэтому все последующие запросы от этого клиента становятся неаутентифицированными.

У этого метода есть и недостатки:

* При каждой аутентификации пользователя сервер должен создавать у себя запись. Обычно она хранится в памяти, и при большом количестве пользователей есть вероятность слишком высокой нагрузки на сервер.
* Поскольку сессии хранятся в памяти, масштабировать не так просто. Если вы многократно реплицируете сервер, то на все новые серверы придется реплицировать и все пользовательские сессии. Это усложняет масштабирование.

**Аутентификация на основе токенов** в последние годы стала очень популярна из-за распространения одностраничных приложений (SPA), веб-API и интернета вещей. Чаще всего в качестве токенов используются Json Web Tokens (JWT). Хотя реализации бывают разные, но токены JWT превратились в стандарт де-факто. При аутентификации на основе токенов состояния не отслеживаются (stateless). Мы не будем хранить информацию о пользователе на сервере или в сессии и даже не будем хранить JWT, использованные для клиентов. Процедура аутентификации на основе токенов:

* Пользователь вводит имя и пароль.
* Сервер проверяет их и возвращает токен (JWT), который может содержать метаданные вроде user\_id, разрешений и т. д.
* Токен хранится на клиентской стороне, чаще всего в локальном хранилище, но может лежать и в хранилище сессий или кук.
* Последующие запросы к серверу обычно содержат этот токен в качестве дополнительного заголовка авторизации в виде Bearer {JWT}. Еще токен может пересылаться в теле POST-запроса и даже как параметр запроса.
* Сервер расшифровывает JWT, если токен верный, сервер обрабатывает запрос.
* Когда пользователь выходит из системы, токен на клиентской стороне уничтожается, с сервером взаимодействовать не нужно.

**Статус коды.**

Информационные (100-105)

Успешные (200-226)

Перенаправление (300-307)

Ошибка клиента (400-499)

Ошибка сервера (500-510)

100- промежуточный ответ, продолжить. Указывает что запрос успешно принят и клиент может продолжить отсылать запросы. Continue- Продолжить.

102- в обработке, сервер получил запрос и обрабатывает его, но обработка ещё не завершена. Proccessing

101- Switching protocol- переключение протокола

103- ранние подсказки, ресурсы которые могут быть загружены, пока сервер будет подготавливать основной ответ

Успешные

200- Ок. Запрос успешно обработан

201- created- Обычно присылается на запрос PUT. Запрос успешно выполнен и в результате был создан ресурс “Создано”

202- Accepted “Принято”. Запрос принят но не ещё не обработан

204- No content- Нет содержимого. Нет содержимого для ответа на запрос, но заголовки которые могут быть полезными посылаются

Сообщения о перенаправлении

301- Moved Permanently. Перемещён на постоянной основе. Значит что URI ресурса был изменён. Возможно новый URI Будет предоставлен в ответе

302- Found.

Клиентские ошибки:

400- Bad request. "Плохой запрос". Этот ответ означает, что сервер не понимает запрос из-за неверного синтаксиса.

401- Unauthorized. "Неавторизовано". Для получения запрашиваемого ответа нужна аутентификация. Статус похож на статус 403, но, в этом случае, аутентификация возможна.

403- Forbidden. "Запрещено". У клиента нет прав доступа к содержимому, поэтому сервер отказывается дать надлежащий ответ.

404- Not found. "Не найден". Сервер не может найти запрашиваемый ресурс.

405- Method Not Allowed. "Метод не разрешен". Сервер знает о запрашиваемом методе, но он был деактивирован и не может быть использован

409- Conflict. Этот ответ отсылается, когда запрос конфликтует с текущим состоянием сервера.

410- Gone. Этот ответ отсылается, когда запрашиваемый контент удален с сервера.

412- precondition failed- клиент указал в заголовке условия, которые сервер не может выполнить

413- URI Запрашиваемый клиентом слишком длинный, для того что-бы сервер мог его обтаботать.

500- internal server error- "Внутренняя ошибка сервера". Сервер столкнулся с ситуацией, которую он не знает, как обработать.

501- Not Implemented. "Не выполнено". Метод запроса не поддерживается сервером и не может быть обработан.

502- Bad Gateway. "Плохой шлюз". Эта ошибка означает что сервер, во время работы в качестве шлюза для получения ответа, нужного для обработки запроса, получил недействительный (недопустимый) ответ.

503- Service Unavailable. "Сервис недоступен". Сервер не готов обрабатывать запрос. Зачастую причинами являются отключение сервера или то, что он перегружен.

504- getaway timeout. Этот ответ об ошибке предоставляется, когда сервер действует как шлюз и не может получить ответ вовремя.

505- httpversion not supported- версия Http не поддерживается сервером

**Адаптивный веб-дизайн и Отзывчивый веб-дизайн (Adaptive Vs. Responsive)**

**Адаптивный**- это когда меняем размер окна браузера и на определённой точке- картинка подстраивается под наши параметры.

**Респонсивный**- это когда подстраивание происходит на каждом пикселе.

**User agent-** Когда вы отправляете HTTP-запрос, он содержит в себе заголовки (headers) с различной информацией. Одним из них является User-Agent. Он сообщает: браузер, его версию и язык, движок браузера, версию движка, операционную систему.

UX — это User Experience (дословно: «опыт пользователя»). То есть это то, какой опыт/впечатление получает пользователь от работы с вашим интерфейсом. Удается ли ему достичь цели и на сколько просто или сложно это сделать.

А UI — это User Interface (дословно «пользовательский интерфейс») — то, как выглядит интерфейс и то, какие физические характеристики приобретает. Определяет, какого цвета будет ваше «изделие», удобно ли будет человеку попадать пальцем в кнопочки, читабельным ли будет текст и тому подобное…

UX/UI дизайн — это проектирование любых пользовательских интерфейсов в которых удобство использования так же важно как и внешний вид.

Разница между UX и UI в том, что UX дизайнер планирует то, как вы будете взаимодействовать с интерфейсом и какие шаги вам нужно предпринять, чтобы сделать что-то. А UI дизайнер придумывает, как каждый из этих шагов будет выглядеть.

В рабочем процессе сначала делается UX, а потом UI.

**Отличие stateless и stateful?**

stateful — модель, при которой объект содержит информацию о своем состоянии, все методы работают в контексте его состояния stateless не предоставляют эту информацию. Все методы объекта работают вне какого-либо контекста или локального состояния объекта, которого в этом случае просто нет. Не делается предположений о состоянии сессии, все изменения атомарны, нет каких-то сессионных переменных на сервере, помнящих результат предыдущего запроса. Они каждый раз дают один и тот же неизменный ответ на один и тот же запрос, функцию или вызов метода. HTTP не имеет состояния в необработанном виде - если вы выполняете GET для определенного URL, вы получаете (теоретически) один и тот же ответ каждый раз.