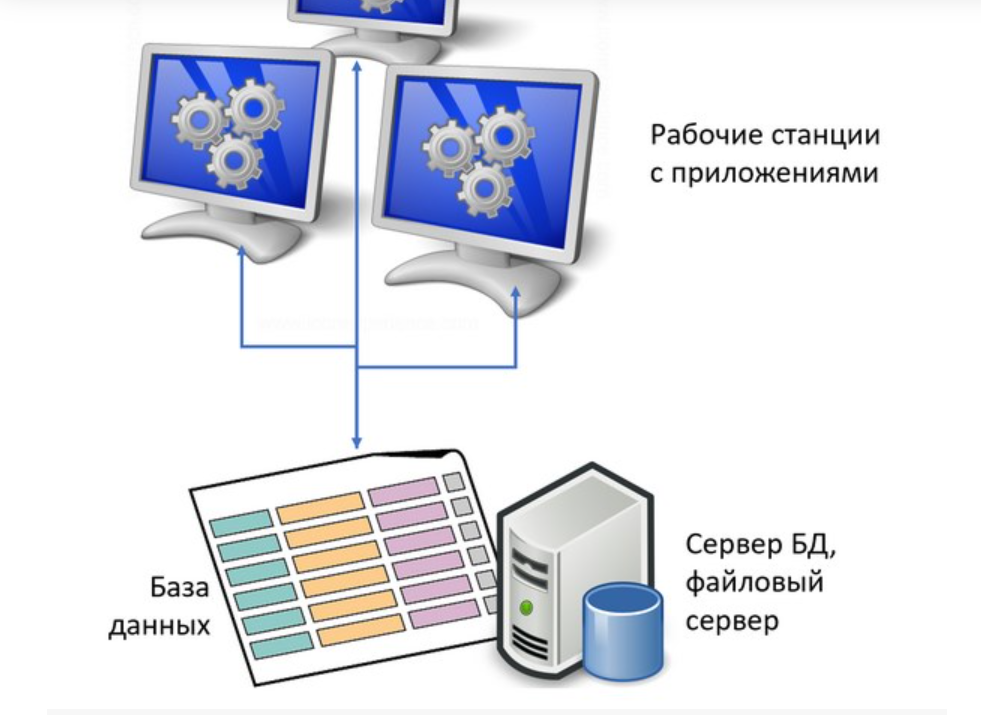
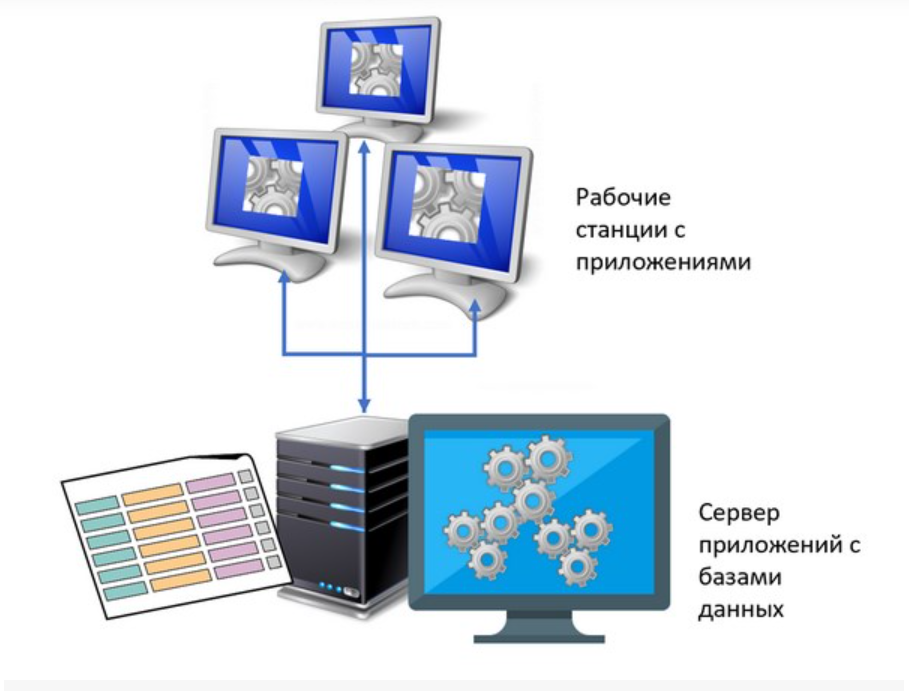
**1)Какие бывают уровни клиент серверной архитектуры?**

**Одноуровневая архитектура «клиент-сервер»** (1-Tier) - такая, где все прикладные программы рассредоточены по рабочим станциям, которые обращаются к общему серверу баз данных или к общему файловому серверу. Никаких прикладных программ сервер при этом не исполняет, только предоставляет данные.

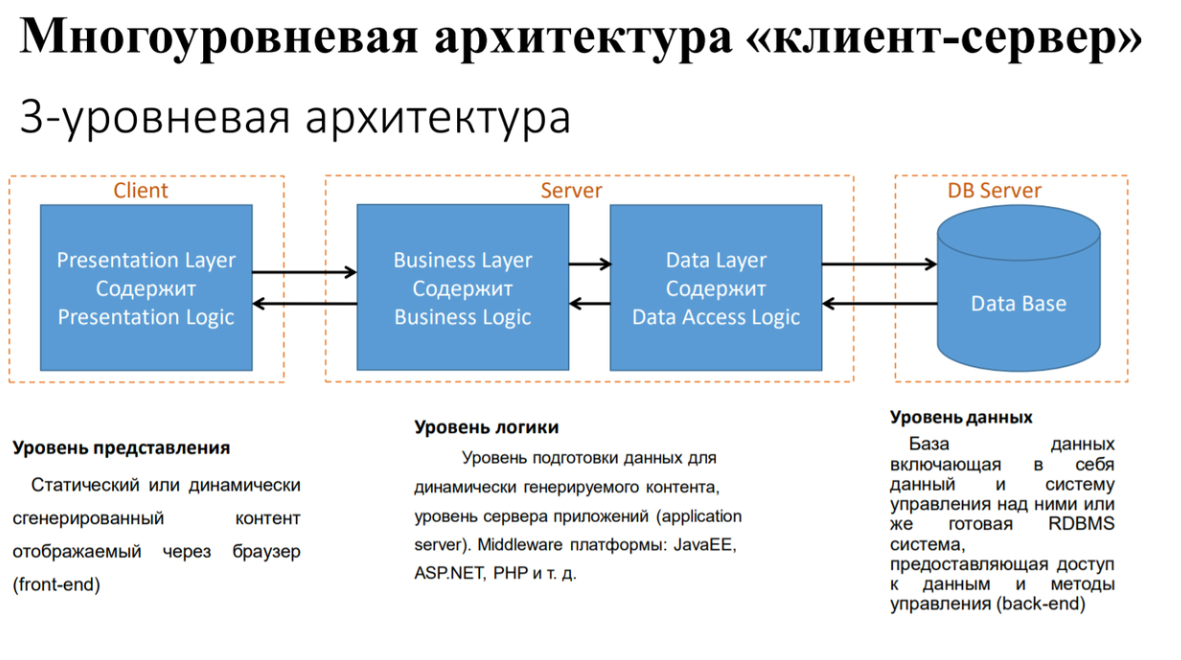


**К двухуровневой архитектуре «клиент-сервер**» следует относить такую, в которой прикладные программы сосредоточены на сервере приложений (Application Server), например, сервере 1С или сервере CRM, а в рабочих станциях находятся программы-клиенты, которые предоставляют для пользователей интерфейс для работы с приложениями на общем сервере.



Такая архитектура представляется наиболее логичной для архитектуры «клиент-сервер». В ней, однако, можно выделить два варианта. Когда общие данные хранятся на сервере, а логика их обработки и бизнес-данные хранятся на клиентской машине, то такая архитектура носит название “fat client thin server” (толстый клиент, тонкий сервер). Когда не только данные, но и логика их обработки и бизнес-данные хранятся на сервере, то это называется “thin client fat server” (тонкий клиент, толстый сервер). Такая архитектура послужила прообразом облачных вычислений (Cloud Computing).

**В трехуровневой архитектуре** сервер баз данных, файловый сервер и другие представляют собой отдельный уровень, результаты работы которого использует сервер приложений. Логика данных и бизнес-логика находятся в сервере приложений. Все обращения клиентов к базе данных происходят через промежуточное программное обеспечение (middleware), которое находится на сервере приложений. Вследствие этого, повышается гибкость работы и производительность.

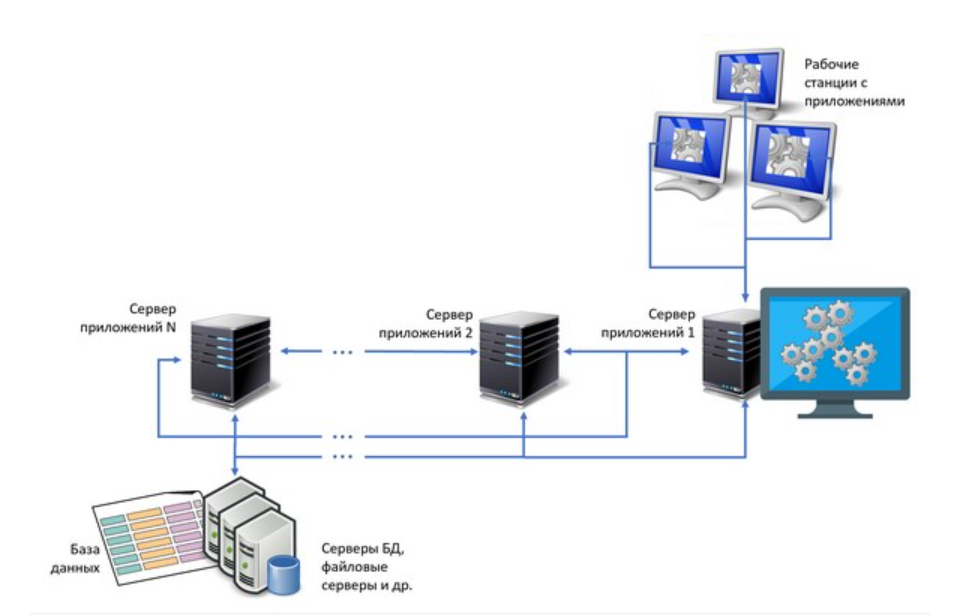


**Многоуровневая архитектура (N-Tier)**

В отдельный класс архитектуры «клиент-сервер» можно вынести многоуровневую архитектуру, в которой несколько серверов приложений используют результаты работы друг друга, а также данные от различных серверов баз данных, файловых серверов и других видов серверов.

По сути, предыдущий вариант, трехуровневая архитектура - не более, чем частный случай многоуровневой архитектуры.

Преимуществом многоуровневой архитектуры является гибкость предоставления услуг, которые могут являться комбинацией работы различных приложений серверов разных уровней и элементов этих приложений.



Очевидным недостатком является сложность, многокомпонентность такой архитектуры.

**Слой клиента** — отвечает за взаимодействие с пользователем. Этот слой не должен содержать бизнес-логики и не должен хранить критически важные данные.

Однако какая-то логика тут все же есть. Во-первых, это взаимодействие с пользователем через интерфейс, валидация вводимых им данных, работа с локальными файлами. Еще сюда можно отнести все, что касается авторизации пользователя и шифрование данных при работе с сервером.

Во-вторых, это несложная бизнес логика. Например, прислал интернет-магазин список товаров, на стороне клиента мы их можем отсортировать, отфильтровать. И примитивное хранение данных тут тоже есть: кэширование, куки залогиненого пользователя и тому подобное.

**Слой бизнес-логики** располагается на втором уровне, на нем сосредоточена бо́льшая часть бизнес-логики. Вне его остаются только фрагменты, экспортируемые на клиента, а также элементы логики, погруженные в базу данных (хранимые процедуры и триггеры).

**Слой данных** обеспечивает хранение данных и выносится на отдельный уровень, реализуется, как правило, средствами систем управления базами данных (СУБД), подключение к этому компоненту обеспечивается только с уровня сервера приложений.

**2)Горячий резерв серверов? холодный резерв серверов? Балансировщики**

[Существует](http://iaas-blog.it-grad.ru/katastrofoustoychivoe-reshenie-s-ispolzovaniem-iaas/) так называемое катастрофоустойчивое решение – допустимый масштаб аварии. DRS-системы могут восстановиться после крупного отказа сразу нескольких дата-центров, не парализовав работу сервисов на длительное время.

**«Холодный» резерв** – это низко-производительные серверы, которые заказываются и настраиваются уже после возникновения аварии.   «Подъем» этой инфраструктуры может тянуться несколько дней или недель и зависит от поставщиков, транспорта и навыков персонала.

**«Теплый резерв»** – это серверы в минимально необходимом количестве для запуска критически важных систем; они подключены, активированы и всегда готовы к переводу нагрузки. Для запуска такой системы требуется не более одного дня.

**«Горячий» резерв** – это серверы, производительность которых соответствует серверам основной площадки; в этом случае все данные реплицируются регулярно и на постоянной основе. Запуск такой системы происходит в пределах одного часа (часто меньше).

**Балансировщик нагрузки** – это устройство, которое находится между клиентом и группой серверов, который обеспечивает одинаковое использование всех серверов, предотвращает перегрузки.

**3) Прокси-сервер** — 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.

Позволяет скрывать реальные данные пользователя, например, IP-адрес и местоположение, подменять их своими. Обходить блокировки по IP-адресу или по географическому положению.

**4) API (Application Programming Interface** - интерфейс программных приложений) — это установка функций и правил позволяющая взаимодействовать между программным обеспечением, которое предоставляет API и другими программными компонентами.

В Веб разработке, под API обычно подразумевают набор стандартных методов, свойств, событий и URL ссылок для взаимодействия с Веб контентом.

Представляет собой передачу данных по определенному запросу со стороны клиента или другого приложения. Например: гугл календарь, погодное приложение, кнопки авторизации с помощью соц.сетей.

Простыми словами, API – это посредник среди программ, задающий правила «составления» (например схема Клієнт кафе ― програма 1, кухар ― програма 2, офіціант ― API)

API обмениваются данными и функциями, а для этого требуются четкие протоколы и архитектуры – правила, по которым будет работать API. Смысл игры не меняется, но структура разнится: RPC, REST и SOAP. Их можно назвать "форматами", каждый из которых имеет уникальные характеристики и используется для разных целей.

**5) Что такое REST API (RESTful)**

А[рхитектурный стиль](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) взаимодействия компонентов распределённого приложения в [сети](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C). Другими словами, REST — это набор правил того, как программисту организовать написание кода серверного приложения, чтобы все системы легко обменивались данными и приложение можно было масштабировать.

REST используются для быстрого обмена простыми параметрами, в том числе теми, из которых состоят базы данных. Поэтому REST API отлично подходит для взаимодействия огромных баз данных. Эти характеристики делают REST популярным для публичных API, например для мобильных приложений.

Model:

**Client** -- (method URI HTTP+headers, body)**>> API (REST API)** –(http request)>>**SERVER**

**Client** << (JSON, XML, HTML) **-- API (REST API)** << (http response)--**SERVER**

Технологию REST API применяют везде, где пользователю сайта или веб-приложения нужно предоставить данные с сервера. Например, при нажатии иконки с видео на видеохостинге REST API проводит операции и запускает ролик с сервера в браузере.

**6) Требования к архитектуре Rest**

**- отделение клиента от сервера** ( код запросов остается на стороне клиента, а код для доступа к данным остается на стороне сервера )

**- отсутствие записи состояния клиента (Stateless).**Сервер не должен хранить информацию о состоянии клиента (проведенных операций). Каждый запрос от клиента должен содержать только ту информацию, которая нужна для получения данных от сервера.

**- кэшируемость.** В данных запроса должно быть указано, нужно ли кэшировать данные (сохранять в специальном буфере для частых запросов). Если такое указание есть, клиент получит право обращаться к этому буферу при необходимости.

**- единство интерфейса (Uniform Interface).** Все данные должны запрашиваться через один URL-адрес стандартными протоколами, например, HTTP. Это упрощает архитектуру сайта или приложения и делает взаимодействие с сервером понятнее.

**- многоуровневость системы (Layered System).** Сервера могут располагаться на разных уровнях, при этом каждый сервер взаимодействует только с ближайшими уровнями и не связан запросами с другими.

**- предоставление кода по запросу (Code on Demand).**Серверы могут отправлять клиенту код (например, скрипт для запуска видео). Так общий код приложения или сайта становится сложнее только при необходимости.

**- начало от нуля (Starting with the Null Style).**Клиент знает только одну точку входа на сервер. Дальнейшие возможности по взаимодействию обеспечиваются сервером.

**7) Что такое SOAP**

это протокол, по которому веб-сервисы взаимодействуют друг с другом или с клиентами в формате XML.

SOAP более безопасный чем REST.

Сейчас протокол используется для обмена произвольными сообщениями в формате [XML](https://ru.wikipedia.org/wiki/XML), а не только для вызова процедур.

Чаще всего SOAP используется поверх HTTP.

SOAP является одним из стандартов, на которых базируются технологии [веб-служб](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%BB%D1%83%D0%B6%D0%B1%D0%B0).

* Использование SOAP для передачи сообщений увеличивает их объём и снижает скорость обработки. В системах, где скорость важна, чаще используется пересылка XML-документов через HTTP напрямую, где параметры запроса передаются как обычные HTTP-параметры.

**8) Чем REST отличается от SOAP**

**REST** - **это архитектурный стиль** где REST предоставляет набор требований, которым нужно следовать, чтобы предоставить веб-службу RESTful. REST поддерживает несколько форматов файлов XML, JSON, TXT, CSV, HTML, но только с одним протоколом http/https.

Архитектурный стиль для операций по работе с сервером, который взаимодействует при помощи http методов (GET, POST, PUT, DELETE).

Считается более простой альтернативой SOAP. Каждая единица информации для этого вида API ― уникальный URL-адрес, который можно запросить.

1) rest использует разные форматы файла, а soap только xml

2) rest работает только с протоколами http/https, а soap работает с разными протоколами - может использоваться с протоколами SMTP, FTP, [HTTP](https://blog.skillfactory.ru/glossary/http/), HTTPS

3) rest это архитектурный стиль, но в котором нет огромного количества правил, есть только рекомендации по созданию приложений, а soap это протокол который сильно ограничен теми правилами которые ему предъявляется

4) rest может быть закэширован, а soap нет

5) в случае rest это простота, скорость, расширяемость, нет привязки к правилам

6) в случае soap больше безопасности, и проще для тестирования, разработка дольше и энергозатратна

7) Soap проще тестировать потому что есть строгие правила создания приложений по soap-WSDL документ-общедоступен, в случае rest из-за рекомендаций, а не правил создания ресурсов-нет чётких правил, по этому документация и наполнение приложений/ресурсов -индивидуальны, документация может быть даже не в общем доступе.

**9) Что такое HTTP, HTTPS**

**Hyper text transfer protocol** - протокол прикладного уровня, который предназначен для передачи файлов между клиентом и сервером. По правилам этого протокола, сервер может отвечать только после того, как получит запрос от клиента.

**HTTPS -** это безопасный протокол передачи данных, который поддерживает шифрование посредством криптографических протоколов SSL и TLS, и является расширенной версией протокола HTTP работает поверх протоколов TCP/IP.

**10) Какие методы HTTP**

**- GET запрос информации (безопасный)**

**- HEAD запрашивает информацию как GET, только без тела ответа (безопасный)**

- **POST отправка данных на сервер (не безопасный)**

- **PUT обновление ранее отправленных данных (не безопасный)**

- **DELETE удаление данных (не безопасный)**

- PATCH частичное изменение данных (не безопасный)

- TRACE вызов возвращаемого тестового сообщения

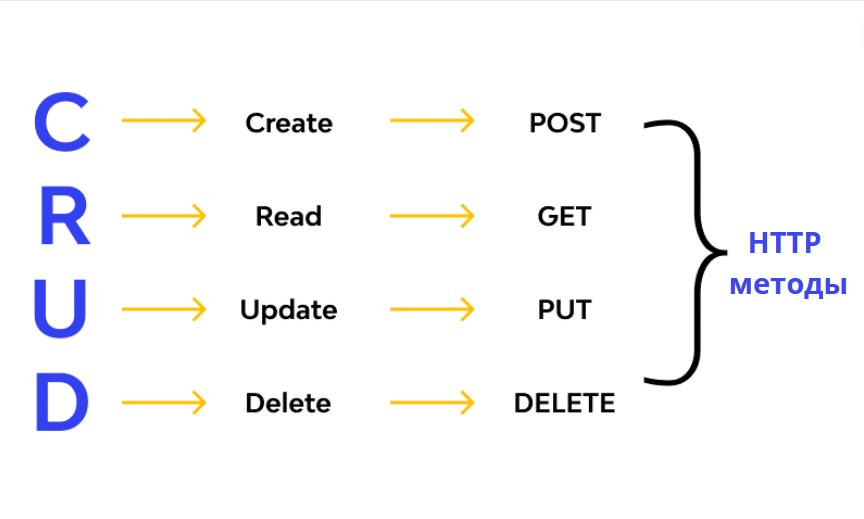
**-** OPTIONS описание параметров соединения с ресурсом (безопасный)

- CONNECT установление связи с сервером

**11) Что такое CRUD**

Акроним, обозначающий четыре базовые функции, используемые при работе с базами данных: создание (англ. create), чтение (read), модификация (update), удаление (delete).

Если утрировать и свести к частному, то CRUD - это набор функций для работы с базой данных. Чтобы при каждом обращении к базе не писать громоздкие и неудобные всякие селекты на SQL, используется CRUD, который уже есть в современных фреймворках. При работе с базой как раз и применяются эти четыре операции - создание (create), чтение (read), обновление (update), удаление (delete).



**12) Чем GET отличается от POST**

Get - выдача сервером какой-то информации клиенту. Для примера, сервер отдаёт какую-то информацию, допустим, имя и логин пользователя, который зашёл на сайт, для того, что бы отобразить её где-то там. Такая информация передаётся как HttpGet. GetПередаётся в URL. Если вам нужно просто запросить информацию с сервера, выбирайте метод GET.  
Дополню, что GET запросы кешируются. POST – нет

* Могут кэшироваться
* Остаются в истории браузера
* Могут быть/стать "закладкой"
* Не должны использоваться для передачи паролей и всего такого
* Имеют ограничение по длине (URL и в некоторых браузерах свои заморочки)

Post - это отправка данных на сервер. Любая. Когда вы регистрируетесь на сайте, ваши данные отправляются как HttpPost. Когда вы делаете изменения личной информации, обновлённые данные отправляются как HttpPost. Т.е. абсолютно любая отправка пакета данных на сервер это HttpPost.

Если вам нужно обеспечить безопасность данных и предоставить возможность отправки больших объемов данных, выбирайте метод POST.

* Никогда не кэшируются
* Не остаются в истории браузера
* Не могут быть/стать "закладкой"
* Не имеют таких ограничений по длине (обычно в браузерах и на web серверах есть ограничение по умолчанию)

**13) Чем POST отличается от PUT**

POST это отправка новых данных на сервер.

PUT вносит изменения в уже имеющуюся на сервере информацию.

**14) Чем PUT отличается от PUTCH**

GET – для получения (чтения) данных (например, списка пользователей).

PUT / PATCH – используется для создания или для обновления данных.

**15) Какие методы бывают идемпотентные**

Метод запроса считается «идемпотентным», если предполагаемое воздействие на сервер нескольких идентичных запросов с помощью этого метода такое же, как и эффект от одного такого запроса. Из методов запроса, определенных этой спецификацией, идемпотентными являются методы PUT, DELETE и безопасный запрос.

Также все безопасные методы являются идемпотентными.

**\*Идемпоте́нтность** ([лат.](https://ru.wikipedia.org/wiki/%D0%9B%D0%B0%D1%82%D0%B8%D0%BD%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) [idem](https://ru.wiktionary.org/wiki/idem#%D0%9B%D0%B0%D1%82%D0%B8%D0%BD%D1%81%D0%BA%D0%B8%D0%B9) «тот же самый» + [potens](https://ru.wiktionary.org/wiki/potens#potens_II) «способный») — свойство объекта или операции при повторном применении операции к объекту [давать тот же результат](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B2%D0%B0%D1%80%D0%B8%D0%B0%D0%BD%D1%82_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), что и при первом.

**16) Какие методы есть безопасные**

Это методы, которые не изменяют состояние в базе данных. Безопасными методами являются: GET, HEAD и OPTIONS.

Методы запроса считаются «безопасными», если их определенная семантика по существу доступна только для чтения; т. е. клиент не запрашивает и не ожидает какого-либо изменения состояния исходного сервера в результате применения безопасного метода к целевому ресурсу. Аналогичным образом, разумное использование безопасного метода не должно причинить никакого вреда, потери имущества или необычной нагрузки на исходный сервер.

Это определение безопасных методов не препятствует включению в реализацию поведения, которое потенциально вредно, не полностью доступно только для чтения или вызывает побочные эффекты при вызове безопасного метода. Однако важно то, что клиент не требовал такого дополнительного поведения и не может нести за него ответственность. Например, большинство серверов добавляют информацию о запросе для доступа к файлам журналов при завершении каждого ответа, независимо от метода, и это считается безопасным, даже если хранилище журналов может переполниться и привести к сбою сервера. Аналогичным образом, безопасный запрос, инициированный выбором рекламного объявления в Интернете, часто имеет побочный эффект в виде списания средств с рекламного счета.

https://datatracker.ietf.org/doc/html/rfc7231#section-4.2.2

**17) В чем разница между безопасностью и идемпатентностью?**

Отличие идемпотентных методов от безопасных заключается в том, что безопасные методы не меняют состояние базы данных, в то время как идемпотентные методы могут внести изменения при первом запросе, но последующие идентичные запросы уже не будут менять состояние в базе данных.

**GET /pageX HTTP/1.1 идемпотентен. Вызвавший несколько раз подряд этот запрос, клиент получит тот же результат:**

**GET /pageX HTTP/1.1**

**GET /pageX HTTP/1.1**

**GET /pageX HTTP/1.1**

**GET /pageX HTTP/1.1**

**POST /add\_row HTTP/1.1 не идемпотентен; если его вызвать несколько раз, то он добавит несколько строк:**

**POST /add\_row HTTP/1.1**

**POST /add\_row HTTP/1.1 -> Adds a 2nd row**

**POST /add\_row HTTP/1.1 -> Adds a 3rd row**



**18) Можно ли в POST передать данные и через URL и через Body**

Вообще по стандарту В POST-запросе данные отправляются в теле запроса, а не в URL. Но при желании все можно, если так программист захочет и запрограммирует.

Таким образом, любые важные данные — логины, пароли, данные карты, персональные данные — лучше передавать с помощью метода POST . Также метод POST поддерживает тип кодирования данных multipart/form-data , что позволяет передавать файлы.

**19) Можно ли с помощью URLa передать данные на сервер?** - Можно

Какое ограничение есть у URLa по количеству символов

**20) В чем разница между URI, URL, URN ?**

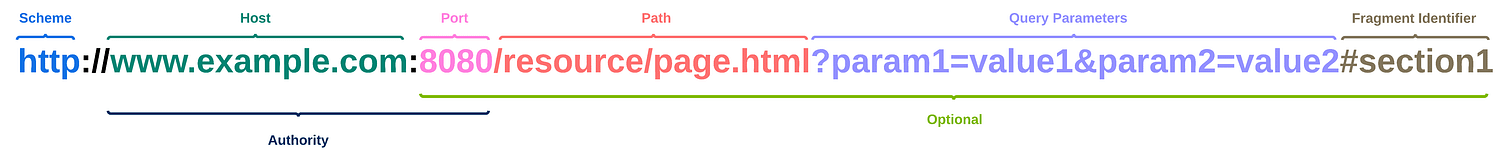
URL (uniform resource locator) – строка символов, которая используется для идентификации ресурса на сервере, но только по его адресу, по местоположению.

URN (uniform resource name) – строка символов, которая используется для идентификации ресурса на сервере, но только по его имени. Предоставляет постоянный идентификатор, который остается неизменным даже в случае перемещения ресурса или изменения его местоположения.

URI (Uniform Resource Identifier) - универсальный идентификатор ресурса. Это строка символов, идентифицирующая ресурс, и по имени, и по адресу.

Таким образом, хотя URI — это общий термин, охватывающий все типы идентификаторов ресурсов, URL и URN являются конкретными подтипами URI.





**Хост** : указывает доменное имя или IP-адрес сервера, на котором размещен ресурс.

**Порт** : **(необязательно)** представляет конкретный номер порта на хосте, через который осуществляется доступ к ресурсу. Если порт не указан, по умолчанию используется стандартный порт для данной схемы. Порт по умолчанию для HTTP — 80, а для HTTPS — 443.

**Путь** : **(необязательно)** . Обозначает конкретное местоположение или каталог на сервере, где находится ресурс. Он может включать каталоги и имена файлов

**Параметры запроса** : **(необязательно)** . Это дополнительные параметры, передаваемые ресурсу, которые обычно используются в динамических веб-приложениях. Они появляются после пути и разделяются символом ?. Параметры запроса имеют форму пар ключ-значение.

**Идентификатор фрагмента** : **(необязательно).** Он относится к определенному разделу или привязке на веб-странице. Идентификатор фрагмента обозначается символом #, за которым следует идентификатор. Обычно он используется для ссылки на определенный раздел веб-страницы.



**URN:** указывает, что идентификатор является унифицированным именем ресурса.

**Пространство имен:** представляет уникальное пространство имен или систему идентификаторов, которая определяет и управляет URN. Он предоставляет контекст и гарантирует уникальность идентификатора.

**Идентификатор:** это строка символов, которая однозначно идентифицирует ресурс в указанном пространстве имен. Сам идентификатор не передает никакой информации о местоположении или методе доступа к ресурсу.

Однако, хотя URL-адреса позволяют вам найти ресурс, URI просто идентифицирует ресурс. Это означает, что URI не обязательно предназначен для использования в качестве адреса для получения ресурса. Он предназначен просто как идентификатор.

Если вы говорите, что ели в единственном итальянском ресторане в своем городе, вы не указываете, как туда добраться. Однако эта информация идентифицирует итальянский ресторан среди других в вашем городе.

С другой стороны, URL-адрес — это URI. Помимо того, что он использует тот же синтаксис URI, он также идентифицирует ресурс через адрес. Другими словами, URL-адрес — это идентификатор, который позволяет идентифицировать ресурс и в то же время указывает направление доступа к нему.

**21) Из чего состоит путь запроса?**

Из чего состоит запрос HTTP – реквест

-http метод (описывает что должно быть сделано с ресурсом)

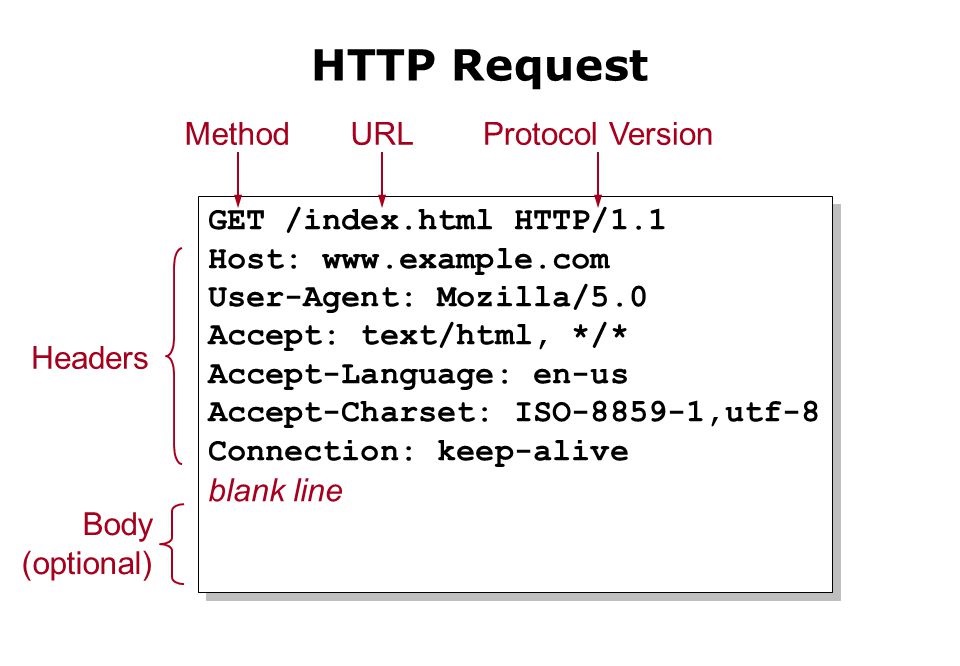
-endpoint (содержит URI который указывает где и как найти ресурс в интернете)

-header (содержит в себе информацию относящуюся к клиенту и к серверу. предоставляет API ключ, название или IP адрес компьютера на котором установлен сервер, информация о формате ответа)

-body (для дополнительной информации, условно то что хотим добавить/изменить)

\*Традиционно запросы обладают структурой следующего вида:

1. **Стартовая строка**. Необходима для описания запроса или статуса. В ней сообщается версия протокола и другие сведения. Это может быть запрашиваемый ресурс или код ответа (например, ошибки). Длина — ровно одна строчка.
2. **HTTP-заголовок**. Выглядит как несколько строчек текста, в которых уточняется запрос или описывается содержимое тела сообщения.
3. **Пустая строка**. Она необходима, чтобы сообщить об успешной отправке метаданных конкретного запроса.
4. **Тело запроса**. В нём хранится информация о запросе или документ, который отправляется в ответе на запрос.



**22) Из чего состоят ответы HTTP –респонс**

- стартовая строка ответа http (версия протокола http/1.1, статус код, статус текст)

- header

- body

**23) Что такое JSON и XML**

**XML** это расширяемый язык разметки, используемый для создания логической структуры данных, их хранения и передачи в виде, удобном человеку и компьютеру. Синтаксис состоит из тегов.

SOAP API используется только XML формат, а вот REST API может использовать не только его.

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

**Запись в** **JSON** — это неупорядоченное множество пар **ключ**:**значение**, заключённое в фигурные скобки

*«*{ }*»*.

Ключ описывается **строкой**, между ним и значением стоит символ *«*:*»*. Пары *ключ-значение* отделяются друг от друга запятыми.

Ключом может быть только «стринга»,

JSON это открытый формат обмена данными, который могут читать как люди, так и машины.

JSON не зависит ни от какого языка программирования и является общим API, используемым в самых разных приложениях.

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.

**24) Каким форматом данных могут быть ключи в JSON**

Ключом может быть только строка ([регистрозависимость](https://ru.wikipedia.org/wiki/%D0%A7%D1%83%D0%B2%D1%81%D1%82%D0%B2%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%BA_%D1%80%D0%B5%D0%B3%D0%B8%D1%81%D1%82%D1%80%D1%83_%D1%81%D0%B8%D0%BC%D0%B2%D0%BE%D0%BB%D0%BE%D0%B2" \o "Чувствительность к регистру символов) не регулируется стандартом, это остаётся на усмотрение программного обеспечения. Как правило, регистр учитывается программами — имена с буквами в разных регистрах считаются разными, например[[5]](https://ru.wikipedia.org/wiki/JSON#cite_note-5)),

**25) Каким форматом данных могут быть значения в JSON**

Значением может быть любая форма. Повторяющиеся имена ключей допустимы, но не рекомендуются стандартом; обработка таких ситуаций происходит на усмотрение программного обеспечения, возможные варианты — учитывать только первый такой ключ, учитывать только последний такой ключ, генерировать ошибку.

**26) В качестве значений в JSON могут быть использованы:**

* **запись** — это неупорядоченное множество пар **ключ**:**значение**, заключённое в фигурные скобки *«*{ }*»*.
* **массив** (одномерный) — это упорядоченное множество **значений**. Массив заключается в квадратные скобки *«*[ ]*»*.
* **число** (целое или вещественное).
* **литералы** *true* ([логическое значение](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D1%82%D0%B8%D0%BF) «истина»), *false* ([логическое значение](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D1%82%D0%B8%D0%BF) «ложь») и *null*.
* **строка** — это упорядоченное множество из нуля или более символов [юникода](https://ru.wikipedia.org/wiki/%D0%AE%D0%BD%D0%B8%D0%BA%D0%BE%D0%B4), заключённое в двойные кавычки.

<https://developer.mozilla.org/ru/docs/Learn/JavaScript/Objects/JSON>

**27) Что такое статус код сервера**

Это трёхзначное число, с которого начинается любой ответ сервера на запрос по протоколу HTTP. Код кратко сообщает суть ответа — был ли выполнен запрос или возникла ошибка.

**- 1\*\* информационные**

100 продолжить (клиент может продолжать отправлять запросы)

101 переключение протокола

**- 2\*\* успешные**

200 ОК

201 создано

202 принято (запрос принят, но еще не обработан)

**- 3\*\* переадресация**

300 множественный выбор

301 перемещен на постоянной основе (URI был изменен)

**- 4\*\* ошибка на стороне клиента**

400 плохой запрос (bad request )

403 запрещено (нет права доступа)

404 ресурс не найден (Not Found-возможно не верный адрес)

405 метод не разрешён (Method Not Allowed -метод запроса известен серверу, но был отключён и не может)

быть использован.

**- 5\*\* ошибка на стороне сервера**

500 внутренняя ошибка сервера

502- сервер запросил информацию с доп. ресурса и получил не корректный ответ (Bad Gateway)

503- сервис не доступен (сервер не готов обработать данный запрос. Часто причиной этого оказывается закрытие сервера для технических работ или его перегрузка)

504-Gateway Time Out, возникает, когда сервер не может получить ответ от сервера-посредника в установленный срок(если например на сервере заложено время ответа-1 минута, и по истечению минуты, если ответ не найден, сервер возвращает эту ошибку клиенту).

**28) Что такое IP адрес**

Идентификатор любого устройства, подключенного к интернету. Он всегда уникален и никогда не повторяется.

IP – это адрес, по которому можно найти сайт или компьютер. Он состоит из четырех чисел, которые разделены точками, например, 217.69.139.202.

**29) Что такое порт**

Идентификатор пакета, передаваемого по протоколу TCP/IP. Номер порта используется для маршрутизации данных внутри одного хоста и обозначает процесс, которому необходимо доставить конкретный пакет.

Сетевой порт – это некое виртуальное расширение, дополнение к сетевому адресу (IP-адресу). Для большего понимания можно сказать, что без сетевого порта информация с другого устройства вряд ли дойдет до нашего компьютера, так как она будет идти только по IP. Компьютер просто не поймёт, как обработать её, с помощью какого приложения.

- порт для http 80(иногда 8080)

- порт для https 443

**30) Что такое хост?**

Хост IP-адрес — это уникальный числовой идентификатор, присваиваемый устройству или компьютеру, подключенному к сети. Он используется для обмена информацией между устройствами в сети Интернет.

Хостинг — это пространство на сервере, которое позволяет создавать, публиковать и поддерживать сайт в интернете. Чтобы после создания ваш сайт был доступен другим людям, вам нужно загрузить его на хостинг.

**31) Чем web service отличается от web server?**

Веб-сервер - это сервер реализующий http протокол. Cлужит для доставки пакетов

Веб-сервис - это технология для взаимодействия между системами. REST подход подразумевает передачу представления ресурсов, это может быть HTML, JSP, XML. Служит для осмысленного стандартизированного общения между клиентом и сервером

**32) Что такое толстый клиент, тонкий клиент, теплый клиент, холодный клиент**

**- толстый клиент** - это когда вся логика приложения происходит на клиенте, и он может обратиться только в базу данных за данными, клиент который берет на себя все вычеслительные функции (1С-программа яркий пример толстого клиента)

**- тонкий клиент —** клиент который имеет возможность только получать готовый результат от сервера, а логика и все вычислительные функции приложения выполняется на сервере.

**- холодный клиент** — это клиент, про которого сервер ничего не знает, на стороне клиента не хранится никакой информации про него (обычно это пользователь, который первый раз зашел на сайт, на нём еще нет куков и заполненного кэша от этого ресурса, для того чтобы оставаться «холодным» можно включить режим инкогнито).

**- горячий клиент** — это клиент, про которого есть информация о его предпочтениях, настройках и так далее (пользователь, который уже был на этом ресурсе, у него сохранена информация в куках и кэше).

**32) Что такое куки**

Это файлы создаваемые веб сайтами. В ней хранится информация об вашей аутентификации, о предпочтениях и настройках юзера.

Браузер может сохранить этот фрагмент у себя и отправлять на сервер с каждым последующим запросом. Это, в частности, позволяет узнать, с одного ли браузера пришли несколько запросов (например, для аутентификации пользователя). С помощью кук можно сохранить любую информацию о состоянии, HTTP-протокол сам по себе этого делать не умеет.

*Куки часто используются для:*

* *Управления сеансом (логины, корзины для виртуальных покупок)*
* *Персонализации (пользовательские предпочтения)*
* *Трекинга (отслеживания поведения пользователей)*

До недавнего времени куки использовались в качестве хранилища информации на стороне пользователя.

**33) Что такое кеш**

Кеш (cache) браузера — это папка с копиями некоторых данных со страниц, которые вы посещали. Обычно в кеше сохраняются те элементы страницы, которые вряд ли успели измениться за промежуток времени между двумя запросами, — музыка, видео, изображения, стили, скрипты.

**34) Что такое WSDL**

Язык описания веб-сервисов и доступа к ним, основанный на языке XML.- описание операций в soap что умеет какой-то ресурс в SOAP.

**35) Что такое WADL**

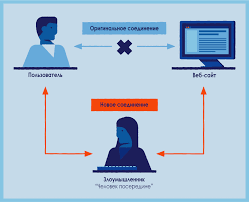
Машинно-читаемое [XML](https://ru.wikipedia.org/wiki/XML)-описание для [web](https://ru.wikipedia.org/wiki/World_Wide_Web" \o "World Wide Web)-приложений [HTTP](https://ru.wikipedia.org/wiki/Hypertext_Transfer_Protocol) (как правило, [веб-сервисы](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B5%D1%80%D0%B2%D0%B8%D1%81) [REST](https://ru.wikipedia.org/wiki/Representational_State_Transfer)). Аналог [WSDL](https://ru.wikipedia.org/wiki/WSDL) для SOAP.

**36) Что такое Ad point**

Конечная точка (endpoint) — адрес, по которому отправляется запрос. Один и тот же объект (ресурс) может иметь несколько конечных точек.

**37) Что такое атака MAN in the Midl**

Атака типа «человек посередине» (MITM) — это кибератака, при которой киберпреступник перехватывает данные, пересылаемые между двумя организациями или людьми. Целью перехвата является кража, прослушивание или изменение данных в какой-либо злонамеренной цели, такой как вымогательство денег.



**38) Что такое и какая разница, Идентификация, Аутентификация, Авторизация**

Аутентификация — это процедура проверки подлинности заявленного пользователя, процесса или устройства. Данная процедура позволяет достоверно убедиться, что пользователь (процесс или устройство) является именно тем, кем себя выдает.

Авторизация — процесс предоставления пользователю или группе пользователей определенных прав доступа.

Идентификация – это процедура распознавания субъекта по его идентификатору (проще говоря, это определение имени, логина или номера). Идентификация выполняется при попытке войти в какую-либо систему.

**39) Что такое валидация и верификация**

Верификация – соответствие продукта требованиям (спецификации).

Валидация – соответствие продукта потребностям пользователей.

**40) В чем разница между сешин сторедж и локал сторедж?**

Хранилище сессии (sessionStorage) обслуживает область хранения данных для каждого домена, доступное на протяжении сессии. (пока браузер открыт, даже в случае перезагрузки страницы)

Локальное хранилище (localStorage) делает то же самое, но сохраняет данные даже в случае, если переоткрыть браузер.

Разница между ними состоит во времени хранения данных. В localStorage данные хранятся постоянно до их «явного» удаления (ни перезагрузка страницы, ни ее закрытие не приводят к удалению данных). Время хранения данных в sessionStorage, как следует из названия, ограничено сессией браузера.

**41) Какую структуру имеет веб-страница?**

Типичная веб-страница представляет собой текстовый файл в формате [HTML](https://ru.wikipedia.org/wiki/HTML), который может содержать ссылки на файлы в других форматах ([текст](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BA%D1%81%D1%82), [графические изображения](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0), [видео](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D0%B4%D0%B5%D0%BE), [аудио](https://ru.wikipedia.org/wiki/%D0%90%D1%83%D0%B4%D0%B8%D0%BE), [мультимедиа](https://ru.wikipedia.org/wiki/%D0%9C%D1%83%D0%BB%D1%8C%D1%82%D0%B8%D0%BC%D0%B5%D0%B4%D0%B8%D0%B0), [апплеты](https://ru.wikipedia.org/wiki/%D0%90%D0%BF%D0%BF%D0%BB%D0%B5%D1%82), [прикладные программы](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), [базы данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), [веб-службы](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%BB%D1%83%D0%B6%D0%B1%D0%B0) и прочее), а также [гиперссылки](https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%BF%D0%B5%D1%80%D1%81%D1%81%D1%8B%D0%BB%D0%BA%D0%B0) для быстрого перехода на другие веб-страницы или доступа к ссылочным файлам.

**42) Что такое HTML/CSS/JavaScript?**

HTML — стандартизированный язык гипертекстовой разметки документов для просмотра веб-страниц в браузере. HTML определяет структуру содержимого страницы. Например, именно в HTML задаются заголовки и абзацы, изображения и т. д.

CSS - язык таблиц стилей, который позволяет прикреплять стиль (например, шрифты и цвет) к структурированным документам (например, к вышеупомянутым документам HTML).

JavaScript – это язык программирования, который используют разработчики для создания интерактивных веб-страниц. Функции JavaScript могут улучшить удобство взаимодействия пользователя с веб-сайтом: от обновления ленты новостей в социальных сетях и до отображения анимации и интерактивных карт.

**43) Зачем чистить кэш?**

Когда вы используете какой-либо браузер, например Chrome, некоторая информация о посещенных сайтах сохраняется в файлах cookie и кеш-памяти. Удаление этих данных помогает устранить различные проблемы с сайтами (в том числе с загрузкой и отображением веб-страниц).

Если кэш-память будет переполнена, то производительность программ снизится. Вторая — старые файлы из кэша могут привести к ошибкам в программах. Например, браузер может хранить в кэше старые скрипты сайта.

**44) Для чего в веб-страницах используют JavaScript?**

JavaScript-функцию или другой JavaScript-код, используемый несколькими HTML-файлами, можно держать в одном файле и считывать при необходимости. Это уменьшает объем занимаемой дисковой памяти и намного облегчает поддержку программного кода, т. к. отпадает необходимость править каждый HTML-файл при изменении кода.

**45) Что такое AJAX?**

AJAX <(Asynhronous JS and XML) (технология обращения к серверу без перезагрузки страницы) подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в фоновом обмене данными браузера и веб-сервера. объект может как отправлять так и получать информацию в различных форматах, XML, HTML, TXT. самое важное в нем то, что он работает асинхронно. с помощью этой технологии можно осуществлять взаимодействие с сервером без необходимости перезагрузки страницы. это позволяет обновлять содержимое страницы частично , в зависимости от действий пользователя.

Обобщим достоинства AJAX:

* Возможность создания удобного Web-интерфейса
* Активное взаимодействие с пользователем
* Частичная перезагрузка страницы, вместо полной
* Удобство использования

(пример аякса: фильтр в онлайн магазине, или когда мы со страницы каталога добавляем объект в корзину, не заходя на страничку товара)

**46) Что такое адаптивная и респонсивная верстка?**

Адаптивная верстка — это формирование структуры документа на базе HTML-разметки, предусматривающей автоматическое изменение страницы в зависимости от размера экрана пользователя. Простыми словами это макет страницы, автоматически подстраивающийся под размер экрана пользователя.

Респонсивная верстка - когда элементы тянутся на всех возможных разрешениях, а верстка подстраивается под breakpoint’ы.

В ней используются как относительные единицы измерения, так и медиа-запросы CSS. Таким образом, это некий симбиоз резиновой и адаптивной верстки.

В реализации такая верстка самая трудозатратная, но этого того стоит — сайт в результате будет адекватно подстраиваться под любое разрешение.

**47) Как выполнить Debug страницы в браузере?**

Debug — Программа-отладчик, которую используют для проверки и отладки выполняемых файлов.

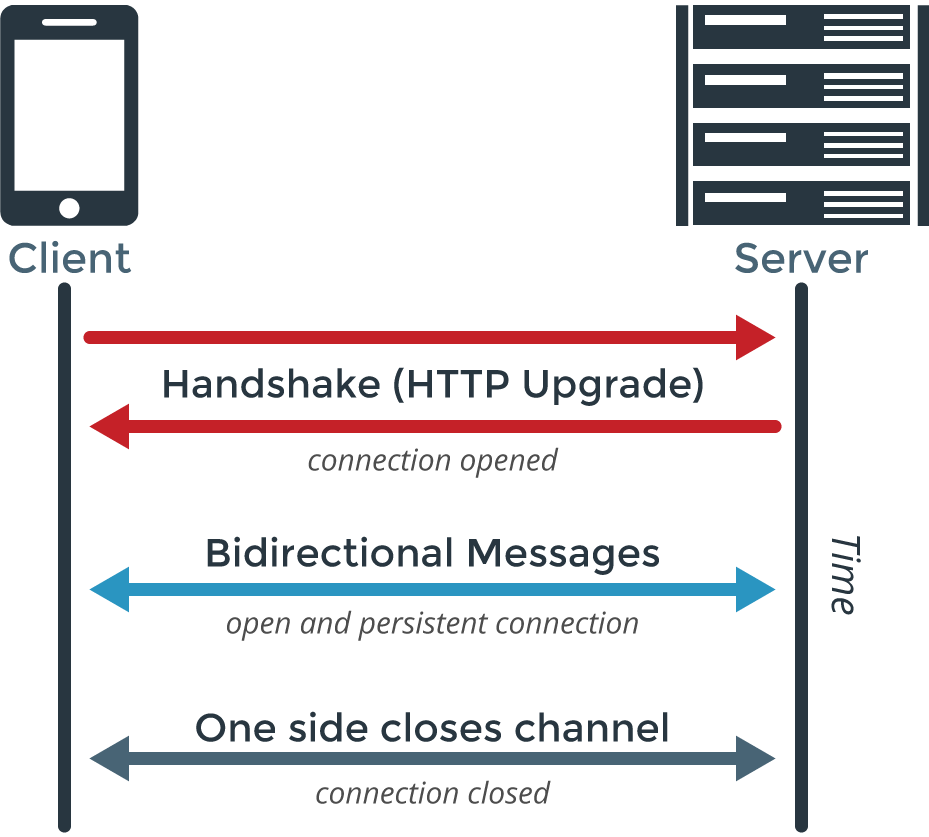
Работая в Chrome, откройте тестовую страницу. Включите инструменты разработчика, нажав F12 (Mac: Cmd+Opt+I). Щёлкните по панели sources («исходный код»). При первом запуске получаем следующее: Кнопка-переключатель откроет вкладку со списком файлов.

Кликните на неё и выберите hello.js. Вот что появится: Интерфейс состоит из трёх зон: 1. В зоне Resources (Ресурсы) показаны файлы HTML, JavaScript, CSS, включая изображения, используемые на странице. Здесь также могут быть файлы различных расширений Chrome. 2. Зона Source показывает исходный код. 3. Наконец, зона Information and control (Сведения и контроль) отведена для отладки, вскоре мы к ней вернёмся. Чтобы скрыть список ресурсов и освободить экранное место для исходного кода, щёлкните по тому же переключателю

https://www.kaznu.kz/content/files/pages/folder24337/%D0%9B%D0%B5%D0%BA%D1%86%D0%B8%D1%8F%204%20java.pdf

**48) Что такое WebSocket и как проверить обрыв соединения?**

WebSocket — протокол связи поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и веб-сервером, используя постоянное соединение.



**49) SSL и TLS - криптографические протоколы.**

TLS – это безопасный протокол связи, обеспечивающий шифрование и аутентификацию, и это относилось к SSL до того, как он устарел. Как TLS, так и SSL используют цифровые сертификаты, которые облегчают подтверждение подключения (рукопожатия) и обеспечивают зашифрованную связь между браузером и веб-сервером(\* Сейчас все пользуются TLS, SSL устарел, но по старой памяти все называли тлс ссл-ом, и по этому иногда их называют “SSL/TLS” -произносят вместе)

Использование в HTTPS

HTTP – это протокол или набор правил взаимодействия между клиентом и сервером в любой сети. HTTPS – это практика установления защищенного протокола SSL/TLS на незащищенном HTTP-соединении.

Перед подключением к веб-сайту ваш браузер использует протокол TLS для проверки сертификата TLS или SSL веб-сайта. Они показывают, что сервер соответствует действующим стандартам безопасности. Сведения о сертификате указаны в адресной строке браузера. В подлинном зашифрованном соединении вместо *http://* отображается *https://*. Дополнительная *буква s* означает *безопасность(секьюрность)*.

**50) TCP/IP**

Сетевая модель передачи данных, представленных в цифровом виде. Модель описывает способ передачи данных от источника информации к получателю. В модели предполагается прохождение информации через четыре уровня, каждый из которых описывается правилом (протоколом передачи).

TCP/IP в настоящий момент самый популярный набор протоколов TCP/IP, который задает стандарты связи между компьютерами и содержит подробные соглашения о маршрутизации и межсетевом взаимодействии.

Уровни TCP/IP

Стек протоколов TCP/IP включает в себя четыре уровня:

* [Прикладной уровень](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D0%B9_%D1%83%D1%80%D0%BE%D0%B2%D0%B5%D0%BD%D1%8C) (Application Layer),
* [Транспортный уровень](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BF%D0%BE%D1%80%D1%82%D0%BD%D1%8B%D0%B9_%D1%83%D1%80%D0%BE%D0%B2%D0%B5%D0%BD%D1%8C) (Transport Layer),
* [Межсетевой уровень](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%BE%D0%B9_%D1%83%D1%80%D0%BE%D0%B2%D0%B5%D0%BD%D1%8C) (Сетевой уровень[[7]](https://ru.wikipedia.org/wiki/TCP/IP#cite_note-:1-7)) (Internet Layer),
* [Канальный уровень](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D1%83%D1%80%D0%BE%D0%B2%D0%B5%D0%BD%D1%8C) (Network Access Layer).

**51) Уровни OSI**

OSI - уровни модели взаимодействия открытых систем (Open Systems Interconnection, OSI) объединяют все типы сетевых коммуникаций между программными и аппаратными компонентами.

Описывает как работают сетевые устройства. она необходима для понимания как работают компьютерные сети. другими словами это процесс передачи данных который разделен на 7 уровней:

1 уровень. физический, на нем происходит момент передачи физических сигналов,токов,импульсов. проще говоря это уровень проводов(блютуз,вайфай) на этом уровне работают концетраторы и репиторы.

2 уровень. канальный. на этом уровне проверяется и исправляются ошибки передачи(информация об отправителе и получателе).

3 уровень. сетевой. на этом уровне происходит маршрутизация трафика. на этом уровне живет маршрутизатор

4 уровень. транспортный обеспечивает передачу данных по сети. он предназначен для надежной передачи информации от отправителя к получателю. тут обитают протоколы TCP and UDP. трафик чувствительный к потерям - TCP. если немного потеряем данных и это не страшно – UDP(поток неприрывный).

5 уровень. сеансовый. управляет соединением сеансами.

6 уровень. представления. происходит кодирование или сжатие файлов. преобразование.

7 уровень. приложения. этот уровень нужен чтобы приложения имели доступ к сетевым службам. доступ к файлам, пересылкам почты и запросам к базам данным.

**52) Что такое FTP?**

FTP (File Transfer Protocol) — это стандартный протокол, используемый для передачи файлов между клиентом и сервером через Интернет. FTP позволяет пользователям загружать, скачивать и удалять файлы с удаленных серверов, а также управлять файлами на сервере.

**53) Что происходит при переходе на веб страницу**

Как только мы установили соединение с веб-сервером, браузер обращается к DNS серверу и отправляет инициирующий HTTP GET запрос от имени пользователя (найти реальный адрес сервера на котором живет сайт). Чаще всего запрашивается HTML файл. В момент, когда сервер получает запрос, он начинает ответ с посылки заголовков ответа и содержимым HTML-файла.

Какие ты знаешь Headers в Request

Какие ты знаешь Headers в Responce

**GDPR** - общий регламент по защите данных, Общий регламент защиты персональных данных, Генеральный регламент о защите персональных данных — постановление Европейского союза, с помощью которого Европейский парламент, Совет Европейского союза и Европейская комиссия усиливают и унифицируют защиту персональных данных всех лиц. GDPR направлен прежде всего на то, чтобы дать гражданам контроль над собственными персональными данными, и на упрощение нормативной базы для международных экономических отношений путём унификации регулирования в рамках ЕС.