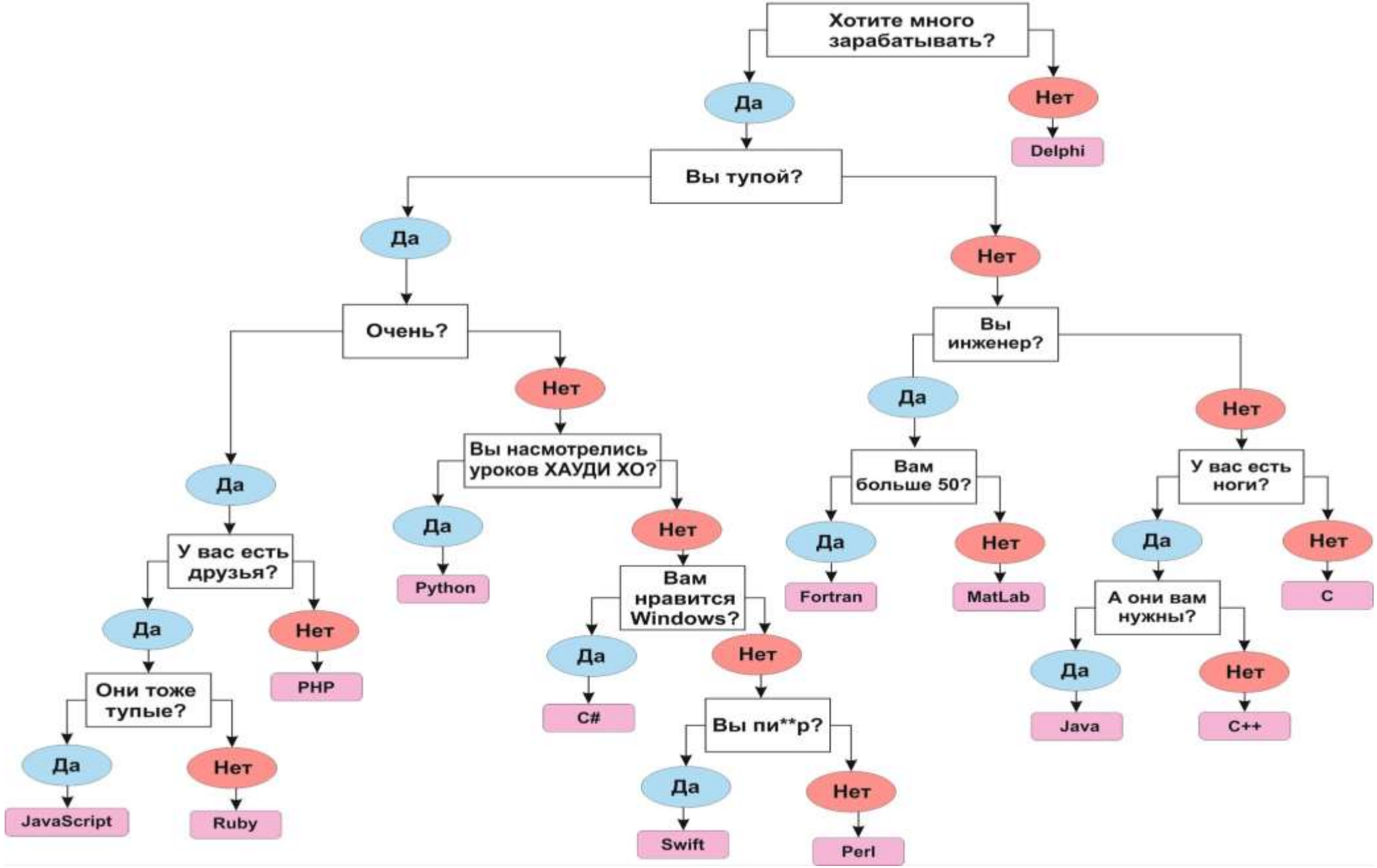
Частное учреждение профессионального образования

«Высшая школа предпринимательства (колледж)»

(ЧУПО «ВШП»)

Кафедра информационных технологий

Инструментальные средства разработки программного обеспечения


Хотите много зарабатывать?
Да
Нет
Delphi
Вы тупой?
Да
Нет
Очень?
Да
Нет
Вы инженер?
Да
Нет
Вы насмотрелись уроков ХАУДИ ХО?
Да
Нет
Вам больше 50?
Да
Нет
У вас есть ноги?
Да
Нет
Python
Вам нравится Windows?
Да
Нет
Fortran
MatLab
А они вам нужны?
Да
Нет
C
У вас есть друзья?
Да
Нет
Вы пи**р?
Да
Нет
Они тоже тупые?
Да
Нет
C#
Java
C++
PHP
JavaScript
Ruby
Swift
Perl

Повторим основные команды Git

git init
git status
git add
git commit
git log

git branch

git checkout

git merge

Типичная работа с Гитхабом

1. Форк оригинального удаленного репозитория
2. Клонируем локально форк
3. Создаем свою ветку для работы
4. Работаем, коммитим
5. Пушим изменения в форк
6. Делаем пуллреквест из форка в оригинал
7. Подтягиваем изменения из оригинального удаленного репозитория в локальный репозиторий

Рабочий процесс в Gitflow

1. Создается репозиторий
2. Репозиторий инициализируется
3. Начинается работа на ветке develop
4. Возникает необходимость опробовать новую штуку – создается feature-ветка и делаются коммиты
5. Закончив работу на feature-ветке, вы сливаете ее с develop
6. Если вы довольны текущей версией, но хотите продолжить работу, создается ветка release, куда перемещается текущая версия. Правка багов будет происходить на этой же ветке.
7. Когда с веткой release покончено, время слить ее в master и продолжить работу с develop

Редакторы кода

- Подсветка синтаксиса
- Автоматические отступы
- Автодополнение
- Разделение рабочей области
- Мини-карта
- Проекты
- Плагины (Emmet)
- Консоль
- Система контроля версий
- Дебаггер
- Горячие клавиши

Выбор очевиден.

Что происходит, когда мы открываем сайт в браузере?

Поиск сервера в интернете

Каждый сайт в сети физически хранится на каком-то сервере. Как только браузер от нас получил адрес сайта, он должен понять, к какому серверу обратиться за данными. Но то, что мы называем адресом, на самом деле не адрес, а **доменное имя**.

Проще говоря, когда вы садитесь в такси и говорите «Мне в „Рио“», вы назвали водителю не адрес, а доменное имя. Водитель уже сам должен знать, где «Рио».

Теперь задача браузера — определить по доменному имени адрес, на который отправлять запрос. Этот адрес называется IP-адресом. Он выглядит, например, так:

31:184:208:243

Поиск сервера в интернете

Чтобы понять, какой именно **IP-адрес** у сервера с нашим сайтом, браузер делает так:

1. Сначала смотрит, посещали мы этот сайт раньше или нет. Если посещали — **возьмёт IP-адрес из истории**. Так же, как водитель, который тысячу раз ездил в «Рио».
2. Если не посещали — посмотрит в **конфигурационных файлах операционной системы**. Иногда для ускорения работы некоторые IP-адреса можно прописать в конфигурации компьютера, чтобы он сразу знал, куда обращаться.
3. Если в настройках такого нет, браузер смотрит **недавние адреса в роутере**, через который компьютер подключается к интернету.
4. Если и там нет, то браузер отправляет запрос на **DNS-сервер**. Там точно всё есть, но результат получится медленнее, чем в остальных способах.

Нюансы IP

Числа и точки — это то же самое, что части обычного почтового адреса. Только в почтовом адресе у нас страна, город, улица и дом, а в интернете это просто узлы связи и магистральные роутеры.

В теории, если вы знаете IP-адрес компьютера и можете сформулировать ему запрос, вы можете «позвонить» на любой компьютер, подключённый к интернету. Например, если вы запустили на своём домашнем компьютере файловый сервер и знаете IP-адрес домашнего компьютера, вы можете зайти на свой сервер из отпуска и залить на него отпускные фотографии, находясь в другой стране. Между вами и вашим домашним железом могут быть тысячи километров, но с помощью IP-адреса вы сможете получить доступ.

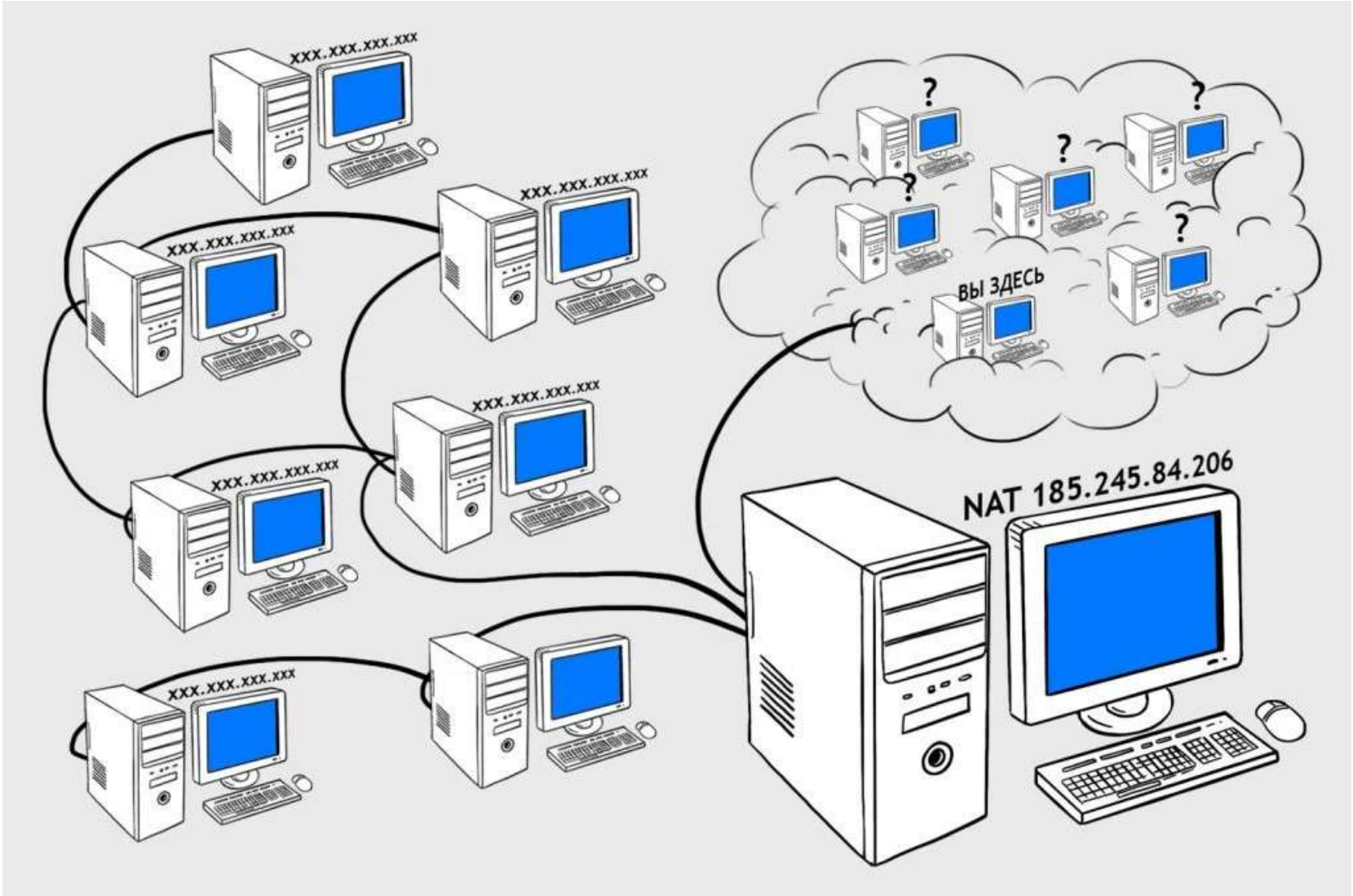
Это если в теории и очень упрощённо. В жизни есть несколько нюансов.

Нюансы IP

Когда вы выходите в интернет, иногда у вас может не быть персонального IP-адреса. Ваши запросы будут уходить с какого-то адреса, но он будет принадлежать не только вам, но и множеству других абонентов. Между вами и интернетом будет узел, который от вашего имени принимает и отправляет запросы. Такой узел называют NAT — Network Address Translator. Из интернета виден один NAT, из которого прут миллионы запросов. Что находится за этим NAT — интернет не знает.

Некоторые провайдеры домашнего интернета выделяют абонентам индивидуальные IP-адреса (без NAT), но даже тогда вам нужно будет настроить свой домашний роутер, чтобы запрос «загрузи фоточки» он отправлял именно на ваш файловый сервер, а не на умный чайник.

Нюансы IP


xxx.xxx.xxx.xxx
xxx.xxx.xxx.xxx
xxx.xxx.xxx.xxx
xxx.xxx.xxx.xxx
xxx.xxx.xxx.xxx
xxx.xxx.xxx.xxx
xxx.xxx.xxx.xxx
?
?
?
?
?
?
ВЫ ЗДЕСЬ
NAT 185.245.84.206

Можно ли вычислить по IP

Максимум, что можно узнать по одному лишь IP, — из какого вы города и какой у вас провайдер. Если вы выходите в интернет с работы или из института — при определённых условиях можно вычислить и их, но не более того.

Полиция имеет полномочия и инструменты, чтобы узнать ваш адрес через интернет-провайдера: они делают запрос с вашим IP, а провайдер смотрит по своей базе данных, кому и когда этот IP был выдан. По закону они обязаны выдать эти сведения полиции, и вот она уже может приехать.

Отправка запроса

Браузер нашёл IP-адрес сервера, на котором располагается наш сайт, и отправляет по этому адресу запрос типа **«Я знаю, что у тебя есть вот такой домен. Мне нужна вот такая страница с этого домена с такими-то параметрами. Дай, пожалуйста».**

Чтобы всё было безопасно и данные никто не перехватил по пути, браузер и сервер договариваются шифровать все сообщения друг другу. Как только все формальности соблюдены, сервер отвечает **«Да, конечно, сейчас всё отправлю»**. Иногда в адресе бывают ошибки, и сервер не может у себя найти нужную страницу. Тогда он отвечает **«А у меня нет нужной страницы, ничем не могу помочь»**, и браузер показывает ошибку.

Сервер думает

Когда сервер получает запрос от браузера и с адресом всё в порядке, он начинает готовить данные к отправке. Для этого он смотрит, какие серверные программы отвечают за этот домен, и говорит им «**Соберите мне вот эту страницу, чтобы я её отправил в браузер»**.

Например, на сервере может стоять CMS или обработчик, который на лету собирает страницу из разных фрагментов кода.

Отправка данных в браузер

Как только сервер получил от своих внутренних программ всё, что ему нужно, он отправляет результат в браузер.

Для этого он нарезает все данные на мелкие пакеты данных по 8 килобайт, нумерует их и отправляет браузеру. Так делается для того, чтобы одновременно передавать много пакетов — в этом случае загрузка идёт быстрее. Нумерация нужна для того, чтобы браузер потом собрал все пакеты в одно целое и получил исходный документ. Если по пути пакет потерялся, браузер говорит серверу **«У меня потерялись такие-то пакеты, отправь их ещё раз»**. И так до тех пор, пока браузер не соберёт все пакеты.

Браузер думает

Когда все пакеты собраны, браузер разбирает документ на составляющие:

1. **HTML;**
2. **CSS;**
3. **JavaScript;**
4. **прочий код, который браузер может выполнить.**

Это нужно для того, чтобы браузер построил **DOM-модель** страницы. Такая модель содержит:

1. **все элементы, которые есть на странице;**
2. **связи между ними;**
3. **как они взаимодействуют между собой;**
4. **что умеют и как реагируют на действия пользователя.**

На основе **DOM-модели** браузер в итоге будет рисовать страницу на экране.

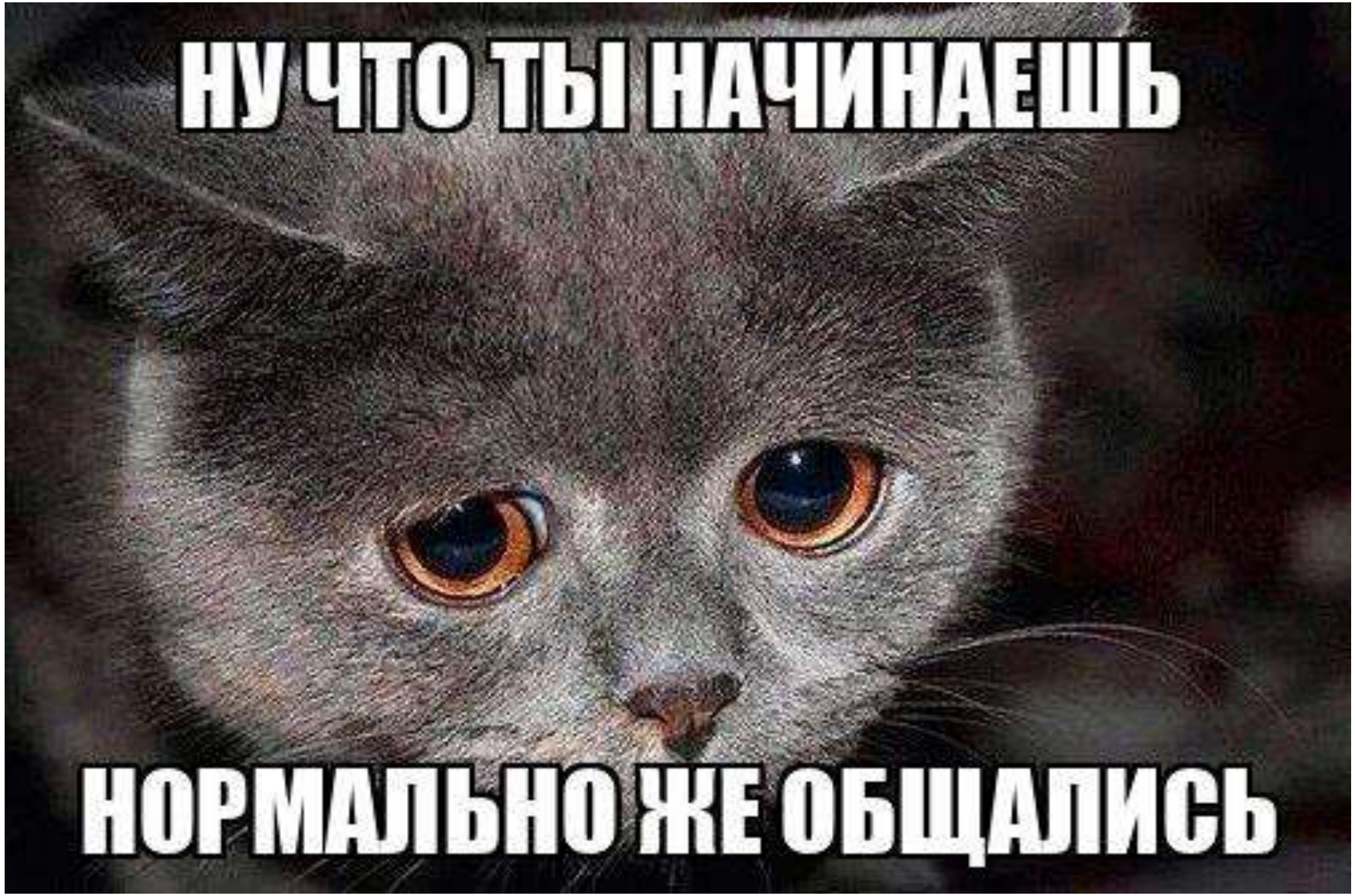
Всё готово

Когда страница загрузилась и браузер всё нарисовал, мы видим готовый результат. Но даже сейчас браузер может продолжать работать над страницей:

1. выполнять JS-скрипт;
2. подгружать в фоне музыку или видео;
3. подгружать страницы, на которые переходят с этого сайта чаще всего, чтобы создать эффект моментальной загрузки новых страниц;
4. записывать что-то в куки или в локальное хранилище;
5. собирать данные о том, что вы делаете на странице;
6. и что угодно ещё, что предусмотрели программисты.

Клиент-серверная архитектура.

Веб-сайт, веб-приложение и веб-сервис


НУ ЧТО ТЫ НАЧИНАЕШЬ
НОРМАЛЬНО ЖЕ ОБЩАЛИСЬ

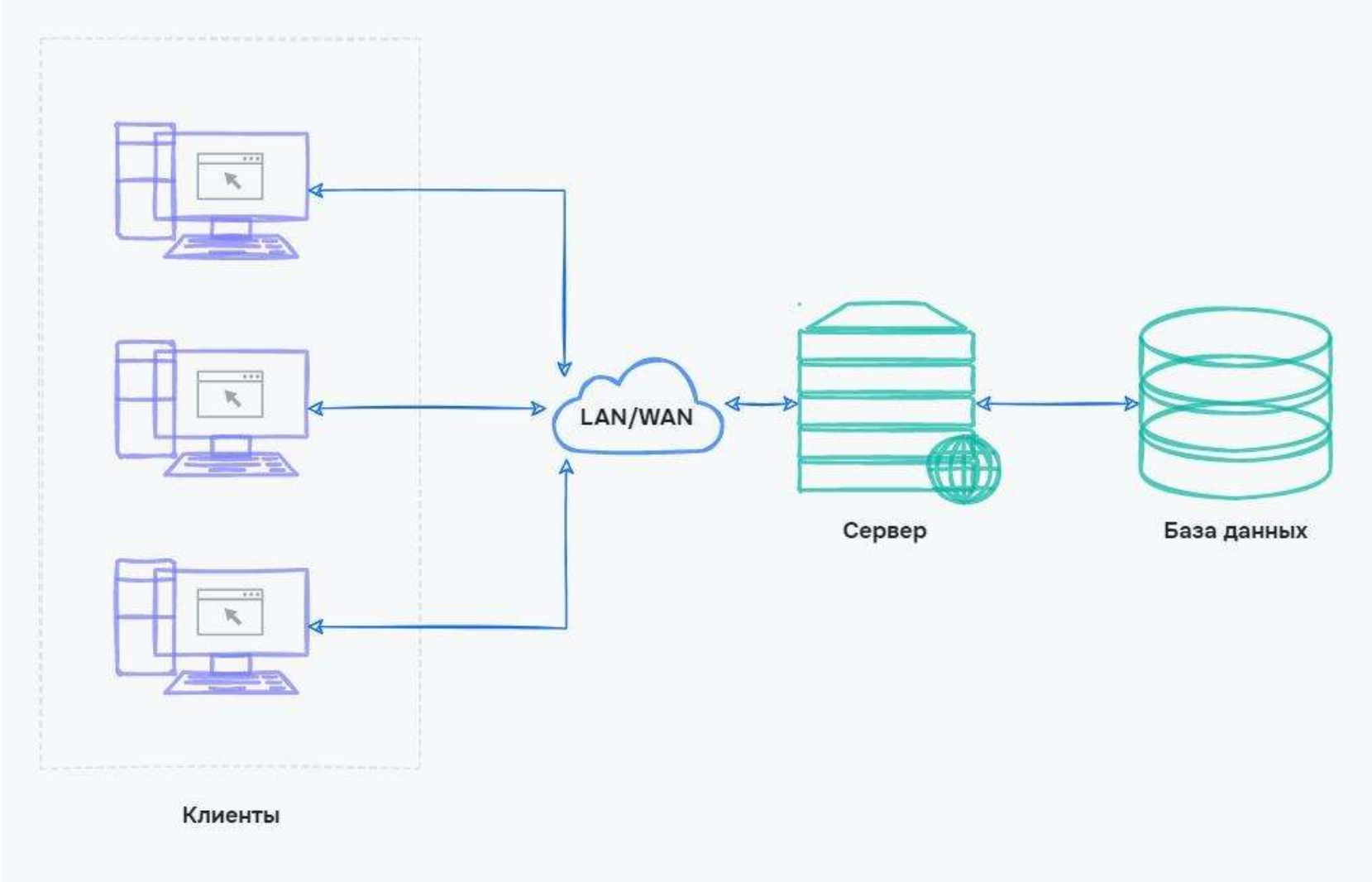
Клиент-серверная архитектура

Что такое клиент-серверная архитектура?

Это такая архитектура, в которой сетевая нагрузка распределяется между поставщиками услуг, которые называются серверами, и заказчиками услуг, которые называются клиентами.

Фактически, клиент и сервер - это некое программное обеспечение. Обычно эти программы расположены на разных вычислительных машинах и взаимодействуют между собой через сетевые протоколы, но также клиент и сервер могут располагаться на одной машине. Самый распространенный протокол это HTTP. Клиент-серверная архитектура построена, в основном, на взаимодействии через данный вид протокола.

Клиент-серверная архитектура



Клиент-серверная архитектура. Тонкий клиент

Тонкий клиент - это компьютер, либо же какая-то программа-клиент в сетях с **клиент-серверной архитектурой**, которая переносит все или большую часть задач по обработке информации на сервер.

Примером такого тонкого клиента может быть **браузер**, который используется для работы с приложением. Вся основная бизнес-логика, все вычислительные мощности у нас расположены на сервере. По сути, это какое-то простое программное обеспечение, которое помогает нам отправлять запросы на сервер.

Клиент-серверная архитектура. Толстый клиент

Толстый клиент - это такое приложение, которое обеспечивает расширенную функциональность независимо от центрального сервера. Часто сервер в этом случае является лишь каким-то хранилищем данных, а вся работа по обработке и представлению этих данных переносится на машину клиента.

Например, 1с бухгалтерия, т.е. в нем содержится вся основная бизнес-логика, на сервер у нас передаются только те данные, которые необходимо сохранить в базе данных, либо же когда нам будет необходимо их получить обратно, отправить какой-то запрос и получить из базы данных ответ. А всё остальное основное находится именно вот на этом толстом клиенте в 1с. Также к такому толстому клиенту можно отнести все онлайн игры.

Веб-сайты

Веб-сайты (веб-страницы) обычно носят какой-то информационный характер, т.е. они состоят из неких веб-страниц, объединённых друг с другом в единый ресурс, имеют какую-то простую архитектуру на основе **html-кода**.

Такие вот веб-сайты, по сути, служат в качестве платформы для предоставления контента для посетителей, они содержат какие-то текстовые файлы, изображения, возможно музыку. Сайты не предоставляют возможности взаимодействия с нашей программой, т.е. пользователи не имеют доступа к размещению своей информации,кроме как заполнение формы для получения, к примеру, подписки.

Наиболее яркими примерами типичных сайтов могут быть новостные, кулинарные сайты, прогнозы погоды.

Веб-приложения

Веб-приложения. В отличие от веб-сайта, веб-приложения - это интерактивные приложения, которые специально позволяют пользователям вводить, получать и манипулировать данными с помощью взаимодействия. Такие программы имеют очень тесную связь с **сервером** и отправляют на него очень много запросов.

Такие приложения могут быть встроены в страницы, либо же сами веб-страницы могут являться приложениями.

И еще одно отличие от веб-сайтов то, что многие веб-приложения могут не иметь реального информативного содержания. Т.е. они используются для выполнения каких-то дополнительных задач - какие-то интернет переводчики, мессенджеры, конвертеры файлов, конвертеры валют.

Веб-сервисы

Веб-сервисы. Прежде всего, **веб-сервисы (или веб-службы)** — это технология. И как и любая другая технология, они имеют довольно четко очерченную среду применения.

Интернет — разнороден, т. е. различные приложения на различных узлах сети функционируют на разных аппаратно-программных платформах, и используют различные технологии и языки.

Чтобы связать все это и предоставить возможность одним приложениям обмениваться данными с другими, и были придуманы веб-сервисы.

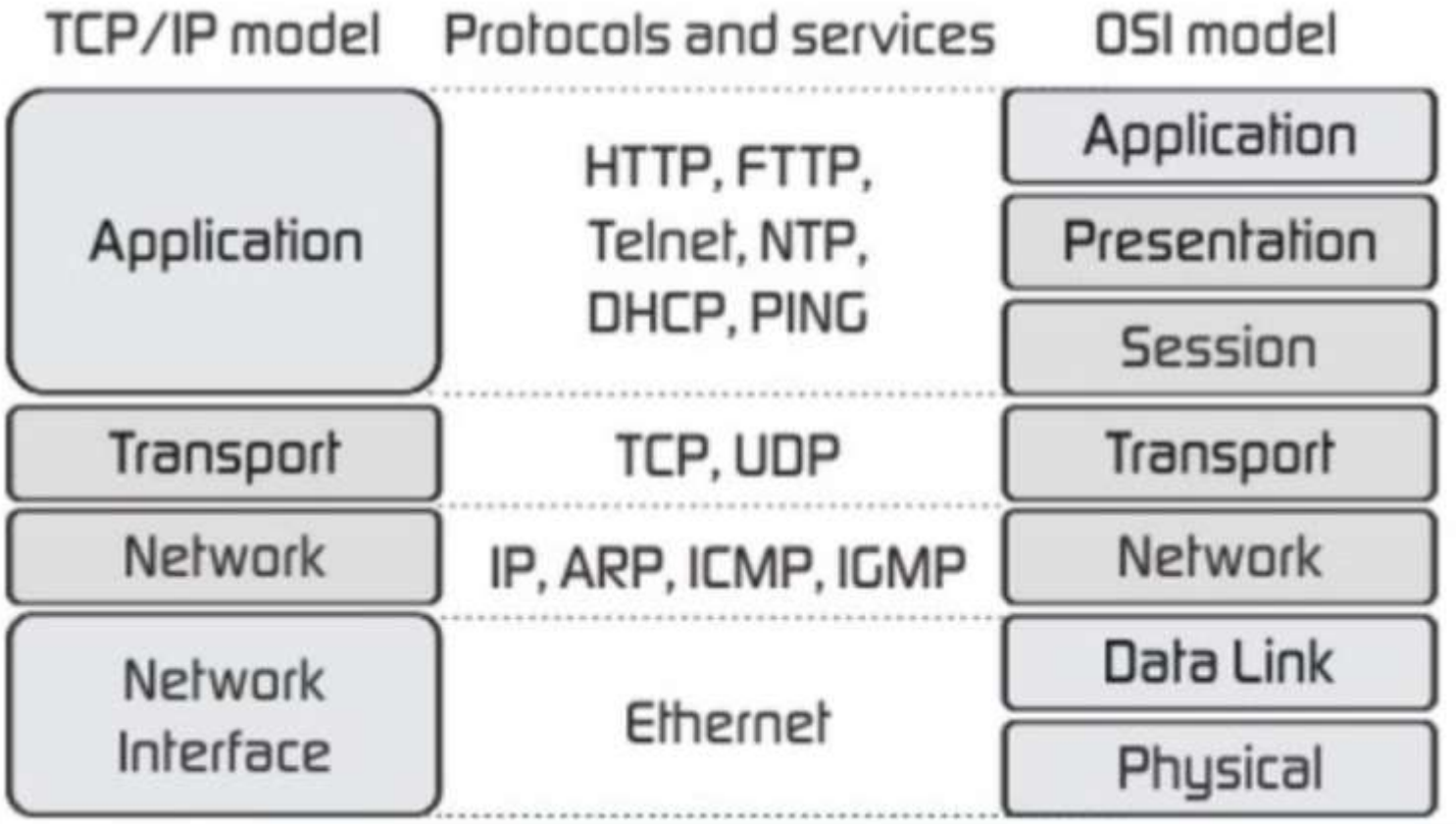
По сути, веб-сервисы — это реализация абсолютно четких интерфейсов обмена данными между различными приложениями, которые написаны не только на разных языках, но и распределены на разных узлах сети.

HTTP

HTTPS

Сеть

Существует несколько моделей сетей - **OSI** и модель **TCP/IP**, наиболее современная модель, которая сейчас повсеместно применяется в сетях интернет.



Сеть

Первый уровень - это уровень сетевых интерфейсов, т.е. в данном случае передаются какие-то физические импульсы

Второй уровень - это уровень сетевой, т.е. здесь уже происходит передача физических сигналов в виде битов или байтов, здесь уже, например, можно выделить такой протокол, как IP.

Третий уровень - транспортный уровень, здесь уже происходят какие-то транспортные взаимодействия в нашей сети, здесь выделяют два вида протоколов - TCP и UDP.

Четвертый уровень - это application (прикладной) уровень. Это специфический уровень для нашего приложения.

TCP vs UDP

TCP протокол - это транспортный протокол, в результате которого при передаче файлов происходит гарантия того, что информация доходит до нашего клиента. Если же в рамках передачи информации происходит то, что информация не проходит, то происходит повторная отправка информации. **Пример TCP протокола** - почтовый сервис, мы должны убедиться, что действительно наш клиент получил какое-то письмо.

UDP - здесь не нужно убеждаться в том, что наша информация дошла, т.е. это происходит непрерывным потоком. Информация передается и нет никаких механизмов, которые говорили бы о том, что гарантированно наша информация попала к клиенту, либо серверу. **Пример UDP протокола** - онлайн-игры или стриминг, где не нужно убеждаться в том, что информация передалась от клиента к серверу.

HTTP

Данные между клиентом и сервером в рамках работы протокола передаются с помощью HTTP-сообщений. Они бывают двух видов:

Запросы (HTTP Requests) — сообщения, которые отправляются клиентом на сервер, чтобы вызвать выполнение некоторых действий. Основой запроса является HTTP-заголовок.

Ответы (HTTP Responses) — сообщения, которые сервер отправляет в ответ на клиентский запрос. Само по себе сообщение представляет собой информацию в текстовом виде, записанную в несколько строчек.

HTTP

В целом, как запросы HTTP, так и ответы имеют следующую структуру:

Стартовая строка (start line) — используется для описания версии используемого протокола и другой информации — вроде запрашиваемого ресурса или кода ответа. Как можно понять из названия, ее содержимое занимает ровно одну строчку.

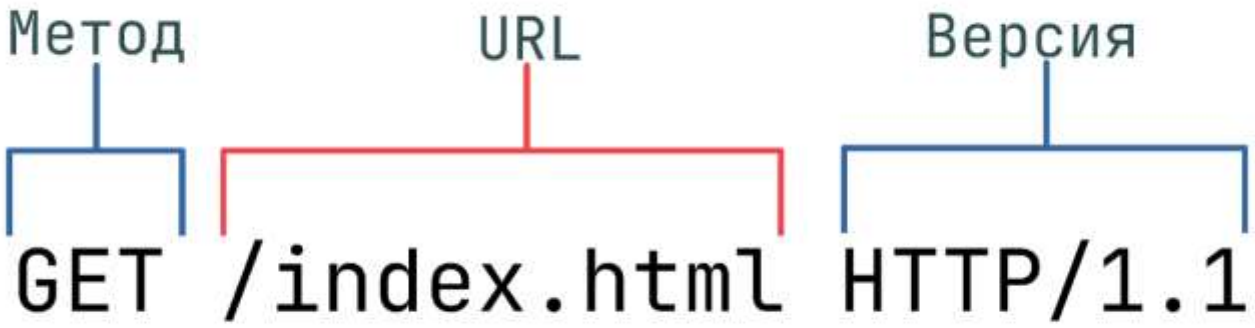
HTTP-заголовки (HTTP Headers) — несколько строчек текста в определенном формате, которые либо уточняют запрос, либо описывают содержимое тела сообщения.

Пустая строка, которая сообщает, что все метаданные для конкретного запроса или ответа были отправлены.

Опциональное тело сообщения, которое содержит данные, связанные с запросом, либо документ (например HTML-страницу), передаваемый в ответе.

Стартовая строка HTTP-запроса

1. **Метод HTTP-запроса** (method, реже используется термин verb). Обычно это короткое слово на английском, которое указывает, что конкретно нужно сделать с запрашиваемым ресурсом. Например, метод GET сообщает серверу, что пользователь хочет получить некоторые данные, а POST — что некоторые данные должны быть помещены на сервер.
2. **Цель запроса.** Представлена указателем ресурса URL, который состоит из протокола, доменного имени (или IP-адреса), пути к конкретному ресурсу на сервере. Дополнительно может содержать указание порта, несколько параметров HTTP-запроса и еще ряд опциональных элементов.
3. **Версия используемого протокола** (либо HTTP/1.1, либо HTTP/2), которая определяет структуру следующих за стартовой строкой данных.



Основные методы HTTP-запроса

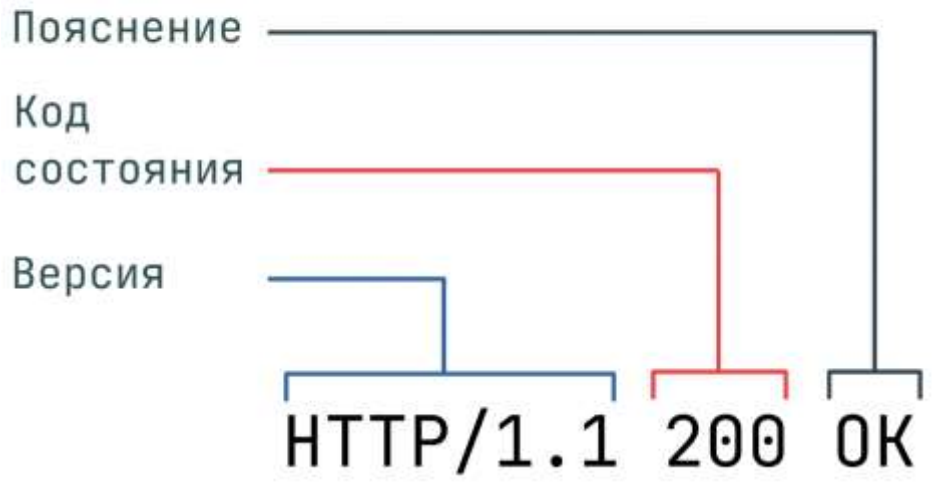
GET	Позволяет запросить некоторый конкретный ресурс. Дополнительные данные могут быть переданы через строку запроса (Query String) в составе URL (например ?param=value).О составляющих URL мы поговорим чуть позже.
POST	Позволяет отправить данные на сервер. Поддерживает отправку различных типов файлов, среди которых текст, PDF-документы и другие типы данных в двоичном виде. Обычно метод POST используется при отправке информации (например, заполненной формы логина) и загрузке данных на веб-сайт, таких как изображения и документы.
PUT	Используется для создания (размещения) новых ресурсов на сервере. Если на сервере данный метод разрешен без надлежащего контроля, то это может привести к серьезным проблемам безопасности.
DELETE	Позволяет удалить существующие ресурсы на сервере. Если использование данного метода настроено некорректно, то это может привести к атаке типа «Отказ в обслуживании» (Denial of Service, DoS) из-за удаления критически важных файлов сервера.

Ответы HTTP

HTTP-ответ является сообщением, которое сервер отправляет клиенту в ответ на его запрос. Его структура равна структуре HTTP-запроса: стартовая строка, заголовки и тело.

Стартовая строка HTTP-ответа называется строкой статуса (status line). На ней располагаются следующие элементы:

1. Уже известная нам по стартовой строке запроса версия протокола (HTTP/2 или HTTP/1.1).
2. Код состояния, который указывает, насколько успешно завершилась обработка запроса.
3. Пояснение — короткое текстовое описание к коду состояния. Используется исключительно для того, чтобы упростить понимание и восприятие человека при просмотре ответа.



Коды состояния

1xx	Коды из данной категории носят исключительно информативный характер и никак не влияют на обработку запроса.
2xx	Коды состояния из этой категории возвращаются в случае успешной обработки клиентского запроса.
3xx	Эта категория содержит коды, которые возвращаются, если серверу нужно перенаправить клиента.
4xx	Коды данной категории означают, что на стороне клиента был отправлен некорректный запрос. Например, клиент в запросе указал не поддерживаемый метод или обратился к ресурсу, к которому у него нет доступа.
5xx	Ответ с кодами из этой категории приходит, если на стороне сервера возникла ошибка.

Коды состояния

Категория	Описание
200 OK	Возвращается в случае успешной обработки запроса, при этом тело ответа обычно содержит запрошенный ресурс.
302 Found	Перенаправляет клиента на другой URL. Например, данный код может прийти, если клиент успешно прошел процедуру аутентификации и теперь может перейти на страницу своей учетной записи.
400 Bad Request	Данный код можно увидеть, если запрос был сформирован с ошибками. Например, в нем отсутствовали символы завершения строки.
403 Forbidden	Означает, что клиент не обладает достаточными правами доступа к запрошенному ресурсу. Также данный код можно встретить, если сервер обнаружил вредоносные данные, отправленные клиентом в запросе.
404 Not Found	Каждый из нас, так или иначе, сталкивался с этим кодом ошибки. Данный код можно увидеть, если запросить у сервера ресурс, которого не существует на сервере.
500 Internal Error	Данный код возвращается сервером, когда он не может по определенным причинам обработать запрос.

HTTP 1.1 vs HTTP 2.0

HTTP 1.1

- использует текстовый формат, из-за чего намного больше места занимает и дольше передается, это влияет на производительность.
- Если мы изучим различные реквесты и респонсы, то там часто встречаются сдваивания заголовков.
- Каждый наш запрос, будь то отправка html-кода, отправка css, отправка скрипта, они проходят по отдельному TCP соединению. Все запросы идут друг за дружкой, т.е., например, сначала мы получаем какую-то текстовую информацию html, затем мы получаем информацию о стилях css, затем мы получаем информацию по javascript по какому-то динамическому обновлению нашей страницы, и вот эти все соединения проходят друг за другом.

HTTP 1.1 vs HTTP 2.0

HTTP 2.0

● использует бинарный формат, т.е. нули и единицы. Как вы понимаете, использование нулей и единиц намного уменьшает количество той информации, которую необходимо передавать на сервер и серверу передавать обратно на клиент.
● Сокращение информации за счёт сжатия заголовков.
● Все запросы проходят по одному единственному TCP соединению, т.е. нет такого различия, какую информацию передают они на сервер, они всегда идут по одному соединению.
● Мультиплексирование. Есть одно соединение и информация передается одновременно, как со стороны запроса, так и со стороны ответа от сервера. Всё происходит одномоментно

HTTPS

Чтобы решить проблему незашифрованных данных, в 2000 году придумали **HTTPS — HyperText Transfer Protocol Secure**, безопасный протокол передачи гипертекста. Внутри себя он работает как обычный HTTP, но снаружи шифрует весь свой трафик. Даже если кто-то вклинится посередине, он увидит только шифр, который не получится разобрать.

За шифрование страниц в **HTTPS** отвечает протокол **SSL — Secure Sockets Layer**, уровень защищённых сокетов.
Сокеты — это такие виртуальные соединения между компьютерами. Защищённый сокет означает, что данные, которые идут внутри от одного компьютера к другому, — в безопасности. Если браузер открывает страницу по такому протоколу, то он перед отправкой на сервер шифрует всё, что вы делаете или вводите на сайте.
Самое то, если нужно отправить данные платёжной карты или логин с паролем от сервиса.

Как работает SSL-протокол

1. После ввода адреса браузер отправляется к нужному серверу и запрашивает у него страницу по протоколу HTTPS. Если страницы работают с HTTPS, то всё отлично, переходим к шагу 2. Если на сервере используется ещё старый протокол HTTP, то он отдаст браузеру страницу по этому незащищённому протоколу и никакого шифрования дальше не будет.
2. Сервер отправляет браузеру копию своего SSL-сертификата, чтобы браузер убедился, что всё в порядке. В таком сертификате написано, что это за домен, кто выдал сертификат и информация о владельце.
3. Браузер проверяет подлинность сертификата и, если всё хорошо, — извлекает из сертификата ключ шифрования и с его помощью безопасно договаривается с сервером о том, каким образом они будут кодировать всё дальнейшее общение.
4. Сервер шифрует страницу и отправляет её браузеру.
5. Браузер расшифровывает страницу, показывает её пользователю, а серверу сообщает, что всё в порядке, работаем дальше. Все данные шифруются и можно отправлять на сервер что угодно.

Что не так с безопасным соединением

Значок замка и надпись «Защищено» означает лишь то, что браузер установил с сервером защищённое соединение. Если кто-то попытается перехватить трафик, то он всё равно не сможет его расшифровать. Но если злоумышленник поставит себе на сайт SSL-сертификат и будет принимать информацию о платёжных картах, то они попадут к нему в руки.

Никто не может помешать преступнику получить сертификат, установить на свой сайт и сделать вид, что это безопасная страница. SSL-сертификат гарантирует безопасную передачу данных, но не отвечает за то, куда они отправляются. Поэтому перед тем как вводить на сайте секретную информацию, убедитесь в том, что сайт принадлежит нужной компании. Иногда бывает так: злоумышленники меняют в адресе сайта одну букву, получают сертификат и делают точно такой же дизайн, как в оригинале. Кто-то заходит на такой сайт за покупками, вводит реквизиты карты и… Будьте внимательны.

Учебная программа от EPAM по фронтенду

rs.school/js/

старт: 05 ноября

Дополнительный материал

docs.github.com/ru

ru.stackoverflow.com

habr.com

ya.ru