Курс по знакомству с веб-технологиями

Лекция 1

Веб-технологии: вчера, сегодня, завтра

В рамках этой лекции рассмотрим историю создания и развития Web, универсальные адреса ресурсов URL URI URN, протоколы взаимодействия http и https, разберемся, как отправляются HTTP-запросы.

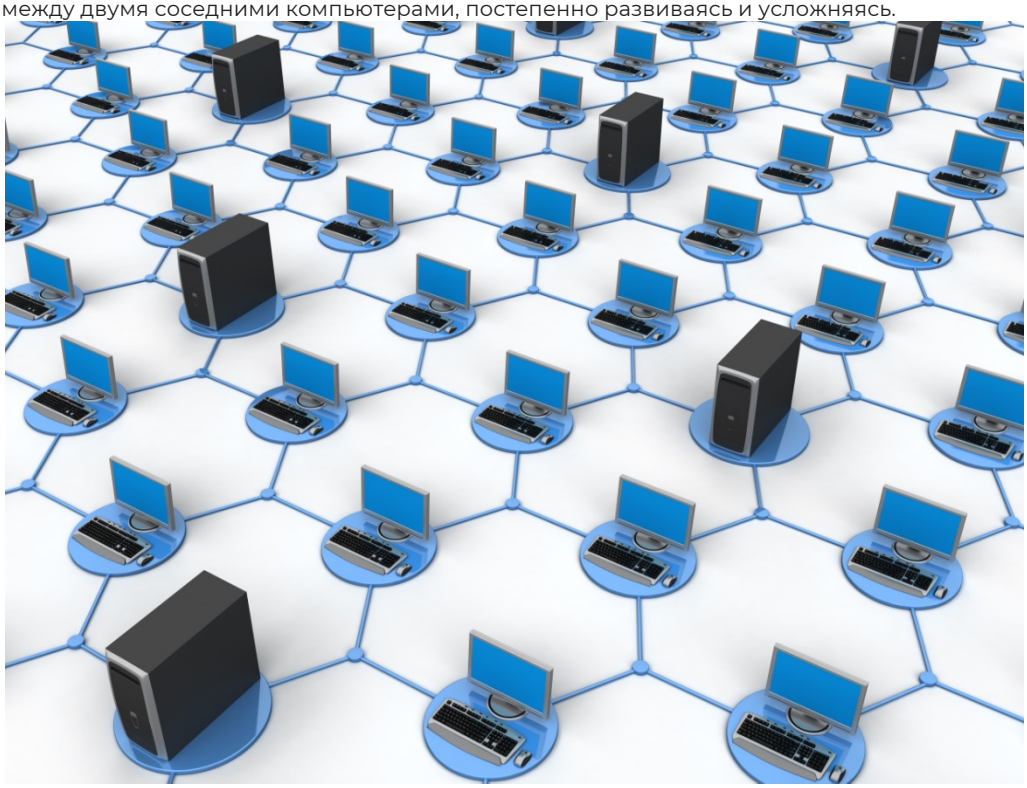
Интернет – это то, чем пользуются почти все. Ежедневно по сети пересылается около 247 миллиардов электронных писем, каждую минуту около 15 часов видеоконтента загружаются в YouTube по всему миру, а создатель Интернета посвящен за свое изобретение в рыцари королевой Англии. И как бы любители постапокалиптики не прочили Интернету скорую смерть, эта технология подобно живому организму, развиваясь из микро-изобретения во всемирную паутину, приобрела иммунитет к различным сбоям и атакам.
Ну что? Давайте, наконец, разберемся со всеми техническими особенностями этой чудо технологии.

Короткий инструктаж

Первое, что нам нужно знать, это то, что история порой шутит с человеческими изобретениями: далеко не всегда задуманное становилось реальностью, очень часто реальностью становилось то, что задуманным не было. Похоже, вся история веба и соответственно веб-разработки — то, с чего оно все начиналось, как развивалось, куда направлялось и где оказалось теперь, — яркий пример этого утверждения.

Путешествие в прошлое

Наше путешествие во времени по технологиям немного напоминает экскурсию в музей Ч. Дарвина. Интернет, подобно живому организму начал свой эволюционный путь от простых связей в виде кабеля между двумя соседними компьютерами, постепенно развиваясь и усложняясь.



Первые шаги в направлении создания всемирной компьютерной сети были предприняты в 1962 году Джозефом Ликлайдером. В своей работе «Галактическая Сеть», он описал первую подробную концепцию компьютерной сети. В том же году Пол Бэран из RAND Corporation в докладе «О Распределенных Коммуникационных Сетях» предложил создать децентрализованную систему связанных между собою компьютеров. Устройства в такой сети были бы равноправны, что позволяло бы сохранить ее информацию и работоспособность даже при частичном уничтожении.

Еще тогда в условиях «Холодной войны» перспективы живучести системы произвели впечатление на власти США, ведь появлялась возможность в разы повысить вероятность сохранения контроля за действиями войск

даже после нанесения противником ядерного удара. Правительство Америки выделило финансирование на исследования в данной области.

Как видите, это еще не было похоже на современный Интернет и задумывалось совсем для других целей, но эта технология считается прародительницей глобальной сети.

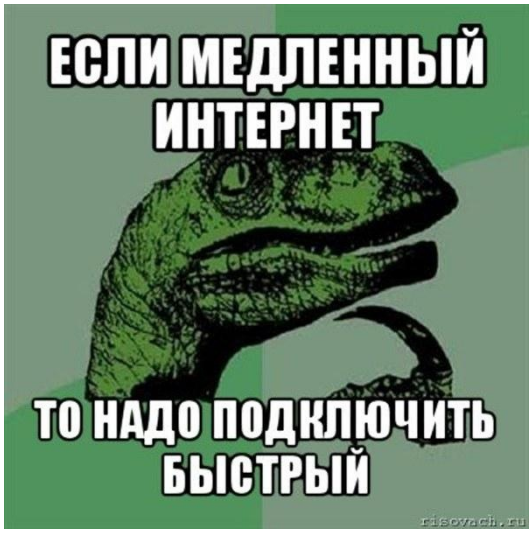
Интернет развился из множества локальных сетей, постепенно охватывая собой все больше территорий. Из межвузовского проекта ARPANet технология превратилась в мировую сеть, в основе которой лежала простая идея возможности просматривать содержимое файлов, которые находятся на другом компьютере.

При этом файлы связывали между собой ссылками. То есть когда мы нажимаем на часть документа в одном файле, то просто увидим содержимое другого файла, местоположение которого указано в этой ссылке. Физически сам файл может находиться на другом компьютере.
Впоследствии, этот компьютер стал множеством жестких дисков, объединенных в один компьютер (сервер), к которому любой пользователь получает доступ и имеет возможность посмотреть содержимое файлов у себя на компьютере (клиент). Также с развитием технологии само понятие компьютера, как ключевого звена сети, стало применяться в расширительном значении.

Доступом в сеть сейчас обладают такие устройства как телефоны, планшеты и, в некоторых случаях, даже бытовая техника — стиральные машины, холодильники, которые создали отдельное понятие — интернет вещей.

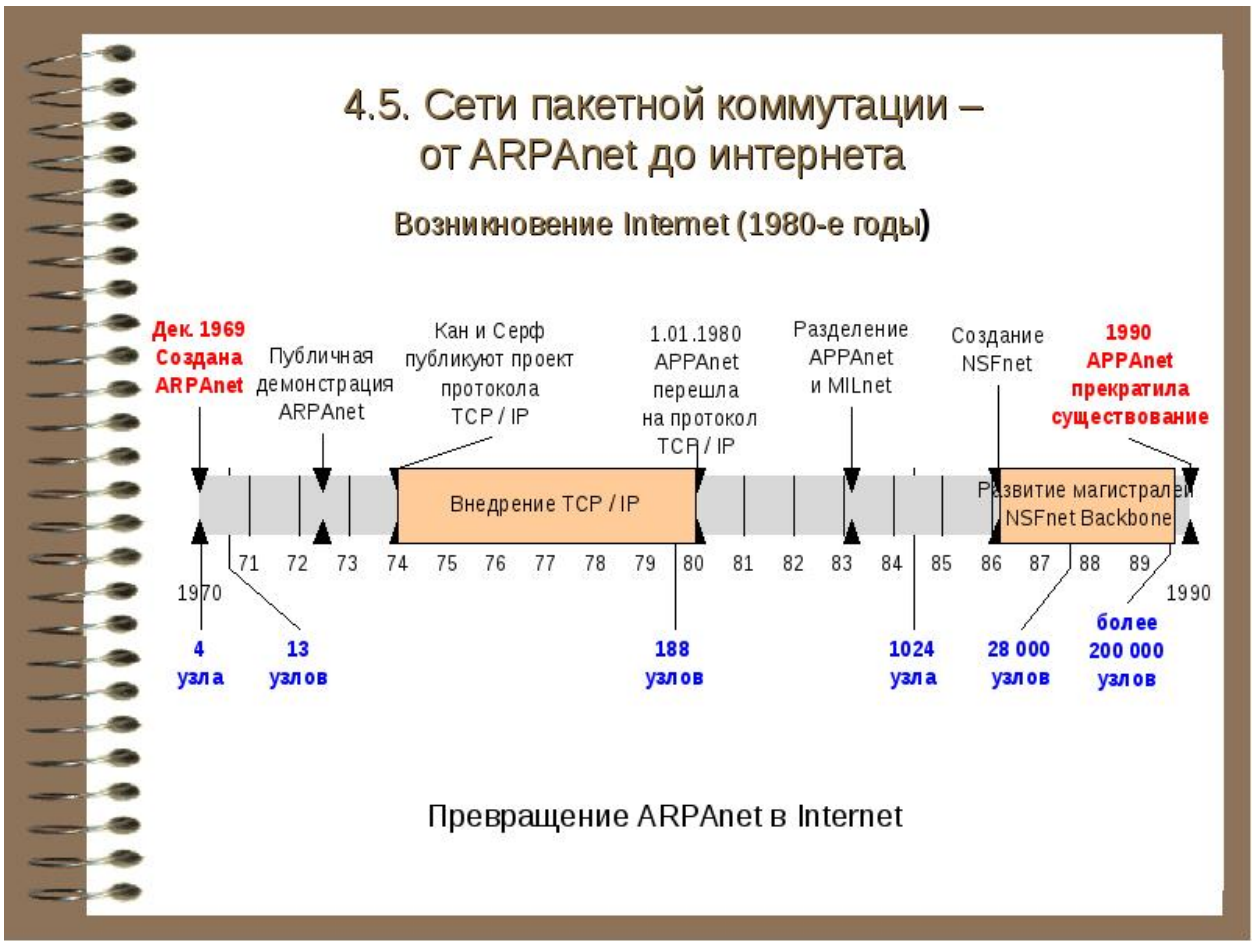
Т.о. простыми словами Интернет — это объединенные по всему миру в сеть компьютеры и иные схожие устройства, осуществляющие процессы обмена и хранения информации, благодаря которым люди могут общаться, обучаться, просмотр фильмы и слушать музыку не учитывая границы и расстояния.

За время при постепенном переходе от Web 1.0 до Web 4.0многое изменилось. Интернет ускорился, стал включать разноплановый динамичный, интерактивный контент, используя огромное количество новых технологий. Первоначальная идея, основанная на гиперссылках трансформировалась до неузнаваемости, немало удивив своих создателей. Интернет прошел путь от «полнейшей анархии» (на компьютерах было различное ПО), до унифицированных стандартов, параллельно опираясь на развитие других технологий (проложенный трансатлантический телефонный кабель в 1973 г. положил начало эре глобальной сети).



Вакцина для сети: HTTP

Продолжая нашу аналогию «Интернет, как биологическое существо» – на определенном этапе созревания, у него появились подростковые проблемы. Пришло время кризиса и потребности в создании новых порядков.

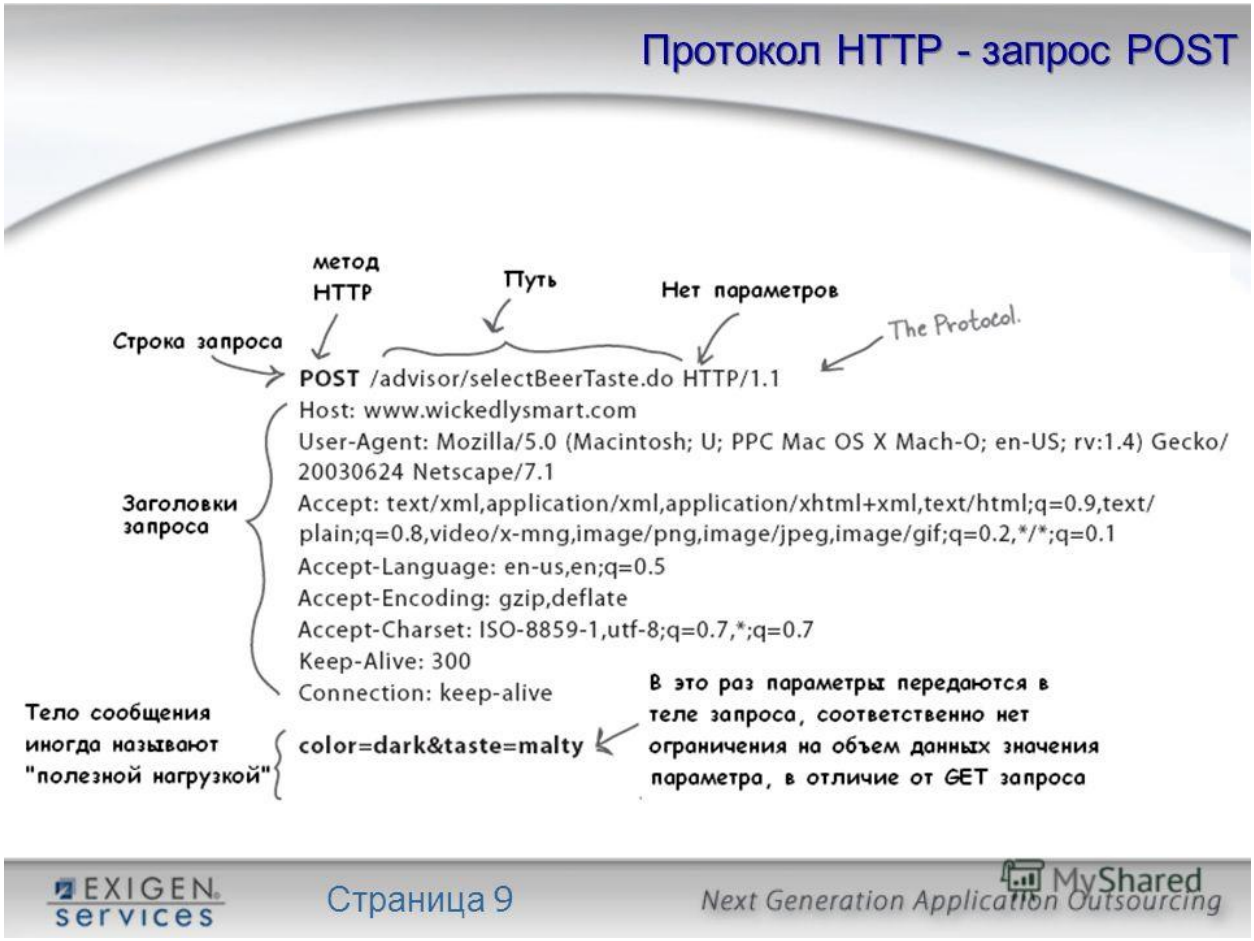


Чтобы компьютеры понимали друг друга и без проблем показывали файлы, лежащие где угодно, нужно было создать набор правил – протоколы. Сейчас самый популярный из них HTTP (HyperText Transfer Protocol). Но этот эволюционный путь был полон сложностей. Развитие различных протоколов передачи данных значительно затрудняло объединение и взаимодействие сетей построенных на различных стандартах. Возможность компаний придумывать свои правила передачи данных приводила к ошибкам.
Например, у нас в России принято представляться в светской беседе вначале по фамилии, затем называть имя – «Меня зовут Шаталова Алевтина!» А в Европе наоборот вначале называют имя: «I am Will Smith». В такой ситуации, можно подумать, что родители в день моего рождения выбрали мне имя – Шаталова, также и с протоколами. По сети мы передаем текст, а вот как он будет оформлен – это уже вопрос создателей сети. И если правила не совпадают, то компьютеры из разных сетей не смогут общаться друг с другом. Поэтому пришлось договориться об общих правилах и универсальных протоколах передачи данных.
Так через несколько итераций был создан HTTP (HyperText Transfer Protocol).

HTTP — широко распространённый протокол передачи данных, изначально предназначенный для передачи гипертекстовых документов (то есть документов, которые могут содержать ссылки, позволяющие организовать переход к другим документам).

Ну, что? Пришло время поближе познакомиться со взрослой и окрепшей «особью» Интернета.

Задача, которая традиционно решается с помощью протокола HTTP в современном Интернете — обмен данными между пользовательским приложением, осуществляющим доступ к веб-ресурсам (обычно это веб-браузер) и веб-сервером. Например, между вашим компьютером и сервером чаще всего необходимо делать следующие операции – запросить что-то с сервера (содержимое страницы, например) или отправить туда какие-то свои данные. Соответственно, для них используются 2 команды в протоколе (GET и POST). Есть еще ряд операций, но эти встречаются чаще всего. Кроме того, надо знать, куда отправлять, что отправлять и т.п.
Например, запрос от браузера может выглядеть вот так:



К слову сказать, большая часть того, что передается по сети – это текст. Браузер отправляет текст на сервер, в ответ ему приходит тоже текст. Другое дело, что надо договориться о том, как его писать: необходимо ввести некоторый синтаксис. Вот сначала указывается глагол POST как на картинке, потом путь, затем тело сообщения.

На данный момент именно благодаря протоколу HTTP обеспечивается работа Всемирной паутины.

HTTPS

Наверняка в браузере теперь вы чаще встречаете HTTPS вместо HTTP. Это еще один признак «взрослого» Интернета.
Сам по себе протокол HTTP не предполагает использование шифрования для передачи информации. По сути передается текст, который может прочитать кто угодно. А сигнал проходит между браузером и сервером (и обратном пути) огромное количество инстанций. Поэтому для HTTP есть распространённое расширение, которое реализует упаковку передаваемых данных в криптографический протокол. Название этого расширения — HTTPS (HyperText Transfer Protocol Secure). HTTPS широко используется для защиты информации от перехвата, а также, как правило, обеспечивает защиту от атак — в том случае, если сертификат проверяется на компьютере пользователя (клиенте), и при этом приватный ключ сертификата не был скомпрометирован, пользователь не подтверждал использование неподписанного сертификата, и на компьютере пользователя не были внедрены сертификаты центра сертификации злоумышленника.

На данный момент HTTPS поддерживается всеми популярными веб-браузерами.

URI, URL, URN

Теперь, когда в Интернет – это глобальная паутина, становится острым вопрос идентификации. Сложно не потеряться в этих «разросшихся» за годы «ветвях». И здесь инженеры создали опознавательные знаки – это верхняя строчка в адресной строке вашего браузера.

Вы когда-нибудь обращали внимание на то, что там написано? Что это? URI, URL или URN? Многие из нас не делают различий между URI, URL, URN, а кое-кто даже и не слышал терминов URI и URN, все просто пользуются термином URL.

Давайте вместе попытаемся разобраться в этом.

URL - Uniform Resource Locator (унифицированный определитель местонахождения ресурса)

URN - Unifrorm Resource Name (унифицированное имя ресурса)

URI - Uniform Resource Identifier (унифицированный идентификатор ресурса)

URL: Исторически возник самым первым из понятий и закрепился как синоним термина веб-адрес. URL определяет местонахождение ресурса в сети и способ его (ресурса) извлечения. Это позволяет нам полностью узнать: как, кому и где можно достать требуемый ресурс, вводя понятия схемы, данных авторизации и местонахождения.

URN: Неизменяемая последовательность символов определяющая только имя некоторого ресурса. Смысл URN в том, что им единоразово и уникально именуется какая-либо сущность в рамках конкретного пространства имен (контекста), либо без пространства имен, в общем (что не желательно). Таким образом, URN способен преодолеть недостаток URL связанный с возможным будущим изменением и перемещением ссылок, однако, теперь для того, чтобы знать местонахождение URN ресурса необходимо обращаться к системе разрешения имен URN, в которой он должен быть зарегистрирован.

URI: Это лишь обобщенное понятие (множество) идентификации ресурса, включающее в нашем случае как URL, так и URN, как по отдельности, так и совместно. Т.е. мы можем считать, что: URI = URL или URI = URN или URI = URL + URN

HTML

Теперь самое время присмотреться к деталям – сайты – каким образом они создаются? Основной технологией здесь является HTML (HyperText Markup Language). И он также не задумывался таким образом, как мы используем его сегодня.

Языков гипертекстовой разметки существует множество и все они созданы, чтобы устранить очевидные отличия человека от машины. Наш мозг лучше воспринимает сгруппированную (отформатированную на заголовки, лиды, подзаголовки, основной текст) информацию, а не однотипные «простыни», в случае с компьютером.

Изначально в связанных файлах физически находящихся на разных компьютерах не было форматирования (разметки). Так появился язык гипертекстовой разметки HTML. Теперь файлы стали структурированными (появились таблицы, картинки и т.п.), но такие файлы надо открывать в специальных программах (браузерах), которые могут их преобразовать в нужный вид.

```
<!DOCTYPE html>

<html>

  <head>

    <meta charset="utf-8">

    <title>Моя тестовая страница</title>
```

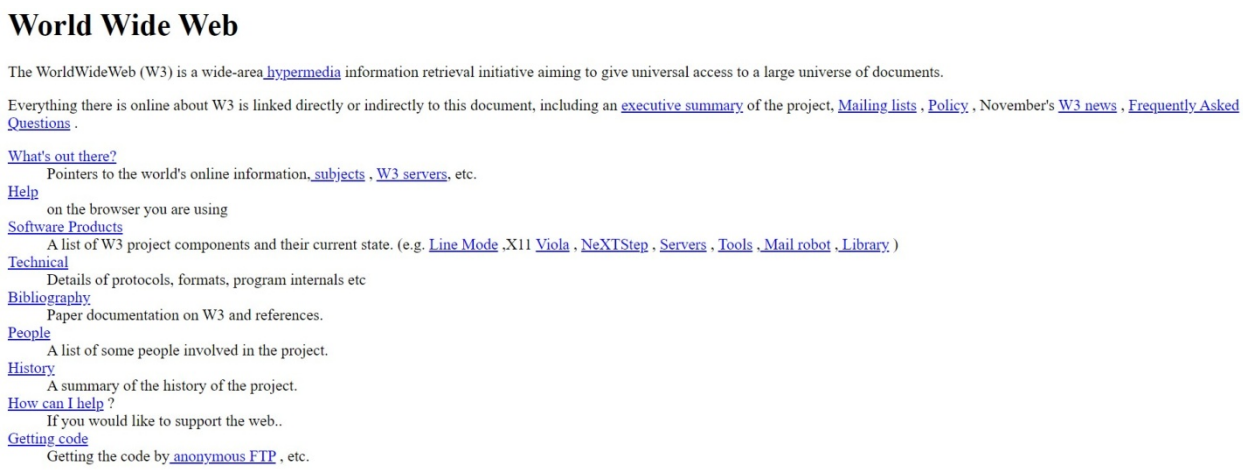
```
  </head>

  <body>

    <img src=" https://avatars.mds.yandex.net/get-
zen_doc/4387796/pub_602a9e4e20b7ca228db53232_602a9f3020b7ca228db7591c/scale_1200 "
alt="Моё тестовое изображение">

  </body>

</html>
```

Чтобы браузер понял, как отображать текст, необходимо все разметить тегами. HTML тег — это специальный код (элемент или команда), который указывает браузеру, как интерпретировать документ для пользователя.

```
<strong>eget aliquet</strong>
```

eget aliquet

Первый сайт увидел свет 6 августа 1991 года. Это был набор примитивных веб-страниц, которые, собственно, и презентовали всемирную паутину — World Wide Web до сих пор доступен по тому же адресу, что и почти три десятилетия назад.



World Wide Web

The WorldWideWeb (W3) is a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an executive summary of the project, Mailing lists , Policy , November's W3 news , Frequently Asked Questions .

What's out there?
Pointers to the world's online information, subjects , W3 servers, etc.
Help
on the browser you are using
Software Products
A list of W3 project components and their current state. (e.g. Line Mode ,X11 Viola , NeXTStep , Servers , Tools , Mail robot , Library)
Technical
Details of protocols, formats, program internals etc
Bibliography
Paper documentation on W3 and references.
People
A list of some people involved in the project.
History
A summary of the history of the project.
How can I help ?
If you would like to support the web..
Getting code
Getting the code by anonymous FTP , etc.

Если вы перейдете на этот сайт и заглянете в его код, у вас может случиться когнитивный диссонанс: несмотря на то, что сайт открывается и отображается вполне нормально в современных браузерах, его код лишь отдаленно напоминает тот, который мы пишем сегодня.

Сам отец интернета — физик-контрактор CERN Тим Бернерс-Ли представлял себе web как библиотеку документов, связанных между собой гиперссылками. Сам термин «Гиперссылка» изобрел еще в 1963 году Тед Нельсон, работавший над проектом Xanadu — попыткой переосмыслить понятие документа и работы с информацией вообще.

Если говорить о веб-страницах в целом и HTML частности, то стоит отметить, что формат сам по себе является не чем иным, как формой представления информации в виде структурированного документа, который может содержать текст, информацию в виде списков и таблиц, изображения, аудио и видео, а также, конечно, гиперссылки. Все это предназначено только для того, чтобы показать информацию пользователю, еще и в достаточно статической форме, подобно странице книги, газеты или журнала. Интерактивность в HTML реализована с помощью элементов формы, задача которых — «взять» информацию у пользователя и отправить ее на сервер. Но это все равно статические файлы на другом компьютере, хоть и уже как-то отформатированные/структурированные.

С развитием интернета этого стало недостаточно. Стало необходимо применять более активно форматирование (оформление) текста. Появилась технология каскадной таблицы стилей.

CSS

Считается, что автором CSS является норвежец Хокон Ли (Håkon Wium Lie), который предложил идею Тиму Бернерсу-Ли в октябре 1994 года, а первая версия стандарта вышла два года спустя. До CSS разделения на представление и контент в HTML не существовало (а теперь это очень частая практика).

Несмотря на кучу различной функциональности в CSS, поддержки целостного и продуманного подхода к размещению элементов на странице долго не было. Это делалось комбинацией каких-то трюков и хаков, понять которые неподготовленному человеку было крайне сложно.

Оформление всех элементов в CSS можно задавать, выбирая их из общего контента с помощью селекторов. Селектор определяет, к какому элементу применять то или иное CSS-правило.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Это заголовок тайтл </title>
        <style>
            p {
                color: red;
            }
        </style>
    </head>
    <body>
        <p>
            Это абзац с текстом.
        </p>
    </body>
</html>
```

CSS — настолько мощный и гибкий инструмент, что его часто применяют не по назначению, например, исправляя недостатки HTML. Например, в HTML есть тег, позволяющий сделать горизонтальную линию, и логично было бы иметь тег, делающий вертикальную. Но его нет, и чтобы нарисовать вертикальную линию, приходится креативить, многие делают это по-разному и используют при этом CSS.

Теперь мы можем создать любое оформление на странице, так или иначе. Но как страницу сделать «живой» – чтобы на ней были интерактивные элементы – реагирующие на наши касания и сохраняющие введенные нами данные? Для этого поможет язык программирования Java Script.

JavaScript

Переходя к, вероятно, самой интересной составляющей фронтенда (именно так называется набор тех инструментов, которые мы используем для отображения сайта в браузере пользователя), стоит отметить, что в 1990-х человека, который занимался сайтом, очень часто называли веб-мастер, и далеко не всегда это был человек с навыками программирования. Как-то на эту тему пошутил Крис Коэр, основатель css-tricks.com: «Два фронтенд-разработчика сидят рядом в баре. Им не о чем говорить». И это не тот случай, когда инструментов настолько много, что просто разработчики не знают их все, а выбирают что-то одно. Это скорее о том, что один — типичный программист с алгоритмическим мышлением, которому удобнее сгенерировать весь контент динамично и императивно (последовательно), достаточно лишь получить контейнер, а второй, наоборот, человек, которому ближе семантика элементов, структура контента, стили, анимации и т.д., то есть декларативный подход.

Собственно, сам Тим Бернерс-Ли никакого языка программирования внутри браузера не предполагал, поэтому неудивительно, что история его возникновения напоминает детектив. Компания Netscape,

основанная в апреле 1994 года, выпустила первую публичную версию браузера Netscape Navigator с номером 0.9 в ноябре того же года. Браузер стремительно начал набирать популярность, и уже через четыре месяца занимал три четверти всего рынка.

Основатель компании Марк Андрессен решил, что HTML не хватает именно легковесного скриптового языка программирования, код которого можно было бы писать прямо в тексте веб-страниц. Поэтому в апреле 1995 года он нанял Брэндона Айка, который был тогда известным специалистом по языкам программирования, для того чтобы встроить язык программирования в браузер Netscape Navigator. Компания тогда активно продвигала Java (язык программирования), поэтому одним из требований, которые выдвинул Брэндон Айк к новому языку, была синтаксическое сходство с Java. Среди требований также была достаточная простота языка и отсутствие классов, поэтому для Брэндона это стало своеобразным челленджем — он решил реализовать в языке ООП (объектно-ориентированное программирование), но без классов. Класс – это описание множества объектов программирования (объектов) и выполняемых над ними действий. Класс можно сравнить с чертежом, согласно которому создаются объекты. Например, мы создадим объект «Кнопка», которая может перекрашивать текст на сайте в другой цвет. В ее классе мы укажем и функцию, которая будет заниматься перекрашиванием.

В Netscape принято было работать очень быстро, поэтому Брэндон Айк разработал прототип языка за 10 рабочих дней в мае 1995 года. Это действительно очень мало для того, чтобы создать собственный язык программирования с нуля, но, похоже, вполне достаточно, если базироваться на уже готовых решениях.

```
<html>
<body>
    <h1>Считаем кроликов</h1>

    <script type="text/javascript">
        for(var i=1; i<=3; i++) {
            alert("Из шляпы достали "+i+" кролика!")
        }
    </script>
    <h1>...Посчитали</h1>
</body>
</html>
```

Также интересной особенностью языка была встроенная устойчивость к ошибкам — синтаксическим и ошибкам во время выполнения кода. Это давало бы возможность пользоваться языком непрофессиональным разработчикам. Поэтому, например, в JS точка с запятой для разделения утверждений является опциональной, а арифметические операции вообще не прерывают ход выполнения кода — можно смело делить на ноль или извлекать квадратный корень из отрицательного числа. Попытка прочитать значение из массива за его пределами просто вернет undefined, а если забыть задекларировать переменную и присвоить ранее незадекларированному идентификатору какое-то значение, то ошибки не возникнет, переменная создастся автоматически.

На тот момент такая гибкость и устойчивость к ошибкам в коде казалась важным достижением, она должна была привлечь к программированию людей, которые до того не имели к нему отношения. Но время показало, что профессиональным разработчикам, привыкшим к строгости и точности в программировании, JavaScript казался недостаточно серьезным языком.

В 1995 году и представить было нельзя, что однажды JavaScript станет самым популярным языком программирования в мире. Теперь же это так.

Теперь мы узнали из чего строятся веб-документы и каким образом они появляются в браузере, а также разобрались за счет каких инструментов мы можем «оживить» страницы и даже сохранить на пользователь компьютера пароль для ввода, чтобы не вводить его каждый раз. Но это еще не все...
Например, у нас есть энциклопедия, в которой мы хотим хранить разные статьи, но помимо самой статьи у нас есть различные блоки (меню слева, шапка сверху, внизу допоплнительная информация и т.п.) и было бы супер, если мы могли не хранить каждую статью в полностью собранном виде (с меню, шапкой и прочими одинаковыми частями), а под каждый запрос формировать страницу на сервере. Это сильно сэкономит нам память, а также оптимизирует работу (проще вносить изменения и т.п.). Так появляются веб-приложения.

Вместо статических страниц у нас теперь страница формируется после запроса. Кроме того, можно часть действий выполнять без обращения к серверу.

Веб-приложения

Например, когда вы заходите на AliExpress и набираете в поиске название товара, обновляется только выдача, которая динамично подгружается из базы данных на сервере, при этом навигация, шапка, баннеры остаются без изменений. Это реализовано веб-приложениями.

Веб-приложения можно разделить на несколько типов, в зависимости от разных сочетаний его основных составляющих:

1. Backend (бэкенд или серверная часть приложения) работает на удаленном компьютере, который может находиться где угодно. Она может быть написана на разных языках программирования. Если создавать приложение используя только серверную часть, то в результате любых переходов между разделами, отправок форм, обновления данных, сервером будет генерироваться новый HTML-файл и страница в браузере будет перезагружаться.
2. Frontend (фронтенд или клиентская часть приложения) выполняется в браузере пользователя. Эта часть написана на языке программирования Javascript. Приложение может состоять только из клиентской части, если не требуется хранить данные пользователя дольше одной сессии. Это могут быть, например, фоторедакторы или простые игрушки.
3. Single page application (SPA или одностраничное приложение). Более интересный вариант, когда используются и бэкенд и фронтенд. С помощью их взаимодействия можно создать приложение, которое будет работать совсем без перезагрузок страницы в браузере. Или в упрощенном варианте, когда переходы между разделами вызывают перезагрузки, но любые действия в разделе обходятся без них.

«Фронтенд» и «бэкенд»

Вы наверняка уже слышали эти модные в сфере программирования слова «фронтенд» и «бэкенд». Давайте в них разберемся подробнее.
Фронтенд — все, что браузер может читать, выводить на экран и / или запускать. То есть это HTML, CSS и JavaScript.
Бэкенд — все, что работает на сервере, то есть «не в браузере» или «на компьютере, подсоединенном к сети (обычно к Интернету), который отвечает на сообщения от других компьютеров».

Для бэкенда вы можете использовать любые инструменты, доступные на вашем сервере (который, по сути, является просто компьютером, настроенным для ответов на сообщения). Это означает, что вы можете использовать любой универсальный язык программирования: Ruby, PHP, Python, Java, JavaScript / Node, bash. Это также означает, что вы можете использовать системы управления базами данных, такие как MySQL, PostgreSQL, MongoDB, Cassandra, Redis, Memcached.

Сегодня существует несколько основных архитектур, определяющих, как будут взаимодействовать ваши бэкенд и фронтенд.

Серверные приложения. В этом случае HTTP-запросы отправляются напрямую на сервер приложения, а сервер отвечает HTML-страницей.

Между получением запроса и ответом сервер обычно ищет по запросу информацию в базе данных и встраивает ее в шаблон (ERB, Blade, EJS, Handlebars).

Когда страница загружена в браузере, HTML определяет, что будет показано, CSS — как это будет выглядеть, а JS — всякие особые взаимодействия.

Связь с использованием AJAX. Другой тип архитектуры использует для связи AJAX (Asynchronous JavaScript and XML). Это означает, что JavaScript, загруженный в браузере, отправляет HTTP-запрос (XHR, XML HTTP Request) изнутри страницы и (так сложилось исторически) получает XML-ответ (файл с расширением xml, написанный на языке программирования для создания логической структуры данных, их хранения и передачи в виде, удобном и для компьютера, и для человека). Сейчас для ответов также можно использовать формат JSON (более компактный и удобный вариант, по сравнению с XML). Это значит, что у вашего сервера должна быть конечная точка, которая отвечает на запросы JSON- или XML-кодом. Два примера протоколов, используемых для этого — REST и SOAP.

Клиентские (одностраничные) приложения. AJAX позволяет вам загружать данные без обновления страницы. Больше всего это используется в таких фреймворках, как Angular и Ember. После сборки такие приложения отправляются в браузер, и любой последующий рендеринг выполняется на стороне клиента (в браузере). Такой фронтенд общается с бэкендом через HTTP-протокол, используя JSON- или XML-ответы.

Универсальные/изоморфные приложения. Некоторые библиотеки и фреймворки, например, React и Ember, позволяют вам исполнять приложения, как на сервере, так и в клиенте (на компьютере пользователя). В этом случае для связи фронтенда с бэкендом приложение использует и AJAX, и обрабатываемый на сервере HTML.

Прогрессивные веб-приложения загружаются лишь один раз и работают (почти) всегда. Вы можете хранить базу данных в браузере. В некоторых случаях вашим приложениям нужен бэкенд только при первой загрузке, а затем лишь для синхронизации / защиты данных. Такой уровень постоянства означает, что большая часть логики приложения находится непосредственно в клиенте.

Бэкенд, в свою очередь, становится легче и легче. Такие технологии, как хранилища документов и графовые базы данных, приводят к сокращению количества обращений к бэкенду для повторного агрегирования данных. Задача клиента — уточнить, какие данные ему нужны (базы данных графов), или извлечь все различные фрагменты данных, которые ему нужны (REST API).

Сейчас можно создавать бэкенд-сервисы, которые работают не постоянно, а только тогда, когда они нужны, благодаря бессерверным архитектурам, таким как AWS Lambda.

Вычислительные задачи теперь можно перемещать между фронтендом и бэкендом. В зависимости от вида приложения можно сделать так, чтобы вычисления производились либо в клиенте, либо на сервере.

Каждый из вариантов имеет свои плюсы и минусы. Сервер — среда более стабильная, имеет меньше неизвестных, но ему постоянно нужно подключение к Сети. Некоторые пользователи используют последние версии браузеров, и им выгоднее использовать клиентские приложения, которые и делают большую часть работы, и могут похвастаться красивым интерфейсом, но тогда вы оттолкнёте пользователей, которые не используют новейшие браузеры и высокоскоростное подключение к Интернету.

В любом случае, хорошо, что есть, из чего выбирать. Главное — выбирать именно то, что лучше всего подходит для конкретной задачи. Надеюсь, у вас появилось больше понимания о том, в каком состоянии сегодня находится веб-разработка.

Кстати, программирование на бэкенд пришло значительно раньше, чем на фронтенд, там его место казалось логичным. Первой такой технологией, позволявшей динамически генерировать HTML, была технология CGI (Common Gateway Interface), созданная в 1993-м, всего через два года после запуска первого сайта. Реализована она была очень примитивно: HTTP-запрос от клиента направлялся на скрипт на сервере, который получал параметры запроса и генерировал ответ, направляя его на стандартный вывод, что, собственно, и получал браузер в ответ. CGI позволяла писать код на любом языке, достаточно было запустить его на сервере — хоть Perl, хоть C, однако в целом это делалось довольно трудоемко.

PHP

Настоящим прорывом в веб-разработке стало появление PHP (*Hypertext Preprocessor*) 1995 года — этот язык был создан специально для интернета и позволял органически соединить HTML и синтаксически близкий к C язык программирования. Идея оказалась удивительно удачной и послужила прообразом для Active Server

Pages от Microsoft (1996) и многих других технологий, которые заняли свою нишу рынка, однако не смогли приблизиться к популярности PHP даже четверть века спустя.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Пример</title>
    </head>
    <body>

        <?php
        echo "Привет, я - скрипт PHP!";
        ?>

    </body>
</html>
```

В определенной степени своим успехом PHP обязан тому, что браузер тогда не воспринимался как серьезная платформа для программирования. Возможности JavaScript в работе со страницей были очень ограничены, к тому же во второй половине 1990-х разгорелись браузерные войны: Microsoft выпустила свой браузер, в котором имплементировала собственную интерпретацию JavaScript под названием JScript, и особенно проблемно было создавать кросс-браузерный код.

Фреймворки

Если бы фреймворков не существовало, то создание сайта длилось бы долго. По сути это такие «заготовочки» - шаблоны, чтобы ускорить процесс разработки. Например, когда мы заказываем еду в MacDonald's или KFS мы очень быстро получаем свой заказ, разумеется потому, что не ждем когда для нашего ланча почистят картошку и замесят тесто для булок. Здесь используются заранее заготовленные полуфабрикаты.

Любой сайт обладает очень схожим функционалом. Можно авторизоваться, если это магазин то есть корзина с соответствующим функционалом, подгрузка файлов и многое-многое другое. И это надо каждый раз делать для каждого сайта. Это очень долго, и, самое главное, каждый раз одно и то же с минимальными изменениями. Фреймворки позволяют вместо написания функционала заново сделать настройки (те самый минимальные изменения) и получить готовый продукт.

Также фреймворки дают возможность подключаться к различным типам СУБД (Систе́ма управле́ния ба́зами да́нных) без погружения в специфику организации инфраструктуры, создавать элементы интерфейса и ускорять процесс верстки. В нем есть готовые решения для работы с файловой системой, инструменты для оптимизации и ускорения работы приложения.

JavaScript-фреймворки это всегда горячая тема. Программисты шутят: «День проходит впустую, если мир не увидел новый фреймворк». Хотя это едва ли не самая любимая тема для новичков, опытные разработчики редко поддаются на провокации и просто наблюдают за трендами и изучают то, на что есть спрос.

Выводы

Интернет превратился из сайтов, доступных только для чтения, в сегодняшний многофункциональный Semantic Web. Совсем скоро он сменится Web 4.0, обеспечивающим прочную связь между человеком и машиной для создания интерактивных данных. Блокчейн-проекты, такие как Free TON, ChainLink, Polkadot способствуют более быстрому развитию интернета. Криптовалюты и блокчейн принесли очень много новшеств в интернет, например, децентрализованные финансы (DeFi), децентрализованные приложения (dApps), невзаимозаменяемые токены (NFT).

Использование новых технологий в повседневной деятельности приведет к симбиозу между человеком и машиной. Большие данные будут все чаще использоваться в виртуальной реальности. Внедрение RFID-меток наладит связь между машинами. Эти устройства и приложения будут чувствовать потребности своих пользователей благодаря искусственному интеллекту. И это будет новая эволюционная веха инернет-технологий.