HTML

Привет, я Исаков Евгений, работаю в компании EPAM. Сегодня я хотел бы поговорить с вами на тему ошибок, которые возникают у подавляющего большинства верстальщиков, то есть у нас с вами. У меня за плечами 7 лет опыта работы верстальщиком, а опыта использования HTML/CSS/JS больше 12, больше 100 проектов за плечами. Я прошел через все те ошибки, о которых сегодня пойдет речь, давно или не так давно, и хочу поделиться с вами тем опытом, который в итоге набрал, с той целью, чтобы вы не наступали на мои грабли и учились на чужих ошибках, чтобы не приходилось учиться на своих.

Сейчас развитие веб-технологий идет семимильными шагами, и так же быстро шагает развитие фронтенда. С нами уже пятая версия стандарта HTML, а от создания первых версий спецификаций HTML и CSS прошло уже больше 20 лет, только вдумайтесь. Я тогда только пошел в первый класс.

Знаете, я не думал, что те проблемы, о которых я буду говорить, настолько распространены. Ведь это всё – довольно очевидные вещи, о которых можно без проблем прочесть в сети, многому из этого учат на большинстве, нынче распространенных, курсов верстки.

Тем не менее, эти проблемы есть, это ошибки повторяются снова и снова, и, несмотря на то, что на дворе уже 2015й год, они все так же популярны.

Прошлой осенью я принимал несколько десятков собеседований, и практически каждый кандидат имел те или иные ошибки в своих проектах, которые он представлял, как свое портфолио. Я подумал, что пора бы положить этому конец, или хотя бы попытаться уменьшить количество этих ошибок.

Собственно, давайте подумаем, какой код мы пишем. Мы стараемся следить за стандартами, за выходом новинок, развиваемся и улучшаем свой код. Но в реальности существует множество не зависящих от нас проблем. Это смещение сроков, неправильно или неточно составленные ТЗ, изменения архитектуры, дизайна проекта, дедлайны. И так далее. Все эти факторы, очевидно, негативно влияют на разработку продукта. Потому в угоду готовой, так сказать, картинке для заказчика, мы забиваем на более изящные решения, которые заняли бы больше времени, на оформление и ревью кода, и, естественно, не возвращаемся после сдачи проекта к этим мелочам, потому что впереди нас ждет еще один проект, с которым будет все точно так же.

Давайте поближе рассмотрим, что именно мы часто делаем неправильно или не так, как нам хотелось бы.

1. Доктайп HTML5

Нынче на вопрос "с чего начинается верстка страницы" все дружно отвечают "указанием доктайпа". К сожалению, так было не всегда. Сейчас ситуация исправляется, если еще не исправилась, но, тем не менее, самое первое и самое главное правило верстальщика - указание доктайпа. Доктайп - это указание на ту или иную версию стандартов, на которую следует ориентироваться браузеру при рендеринге страницы. Многие из вас, думаю, помнят, как раньше указывался доктайп. Это была длиннющая строка, которую никто не знал и не запоминал. Сейчас уже заканчивается та эра, в которой существовало несколько режимов отображения страниц - стандартный, и так называемый режим совместимости, который особенно был выражен в IE и представлял собой поддержку стандартов на уровне IE версии 5.5. Более того, многие сайты не имели указания доктайпа вообще.

Если раньше сайты пестрили указанием разнообразия различных версий доктайпов версий HTML 4.01 или XHTML Transitional, то с выходом новой версии спецификации ситуация значительно упростилась, - теперь вместо размышлений и выбора между зоопарком доктайпов есть одно решение, которое устроит всех. Новый стандарт был призван сделать жизнь разработчиков проще, и начали ребята с доктайпа. И, к слову, в случае с доктайпом у них это получилось.

Потому сейчас нет смысла использовать что-либо кроме <!doctype html>, более того, для самих браузеров и их парсеров уже давно не существует другого стандарта, кроме HTML5. Есть еще исключения вроде IE8, но эта ситуация скоро исчезнет сама собой.

Потому на каждом вашем сайте первая строка должна быть указанием доктайпа, и этот доктайп должен быть именно <!doctype html>

2. Около доктайпа

Бывало, я натыкался на резюме того или иного кандидата, где указывалось, что «человек верстает исключительно с использованием стандарта HTML5», все по фен-шую. И тут же в резюме ссылка на сайт. Открываешь сайт, смотришь в исходный код, в самый верх - <!doctype html> - о, круто! Чувак юзает современный доктайп. А затем смотришь чуть ниже – и видишь примерно такое – старое объявление кодировки и использование устаревших атрибутов для подключения стилей и скриптов. Многие атрибуты в новом стандарте стали устаревшими или были откинуты за ненадобностью, и больше не нужны. Хватит дописывать их просто так, «по привычке», пора их менять. То же самое относится к переосмысливанию назначения различных тегов – тег <center> наконец стал deprecated и канул в лету, тег <b> - тоже стал исключительно декоративным, окончательно потеряв смысловое назначение, которое теперь осталось для <strong>. Те же атрибуты cellpadding и cellspacing для тега <table> тоже больше не нужны, и, больше того, их вообще нет в новой спецификации.

Собственно, не используйте устаревших тегов и атрибутов, если уж вы решили «верстать, как по спеке».

3. Диватоз и новые HTML5 теги

Лет 7-8 назад было очень распространено такое понятие, как «дивная верстка», вы наверняка слышали о таком. Во всех вакансиях и требованиях к верстке обязательно присутствовал пункт «верстка без использования таблиц», «умение верстать сайты без использования таблиц», и другое в таком же духе. Собственно, это привело к тому, что люди верстали дивами всё подряд, даже таблицы. И, каюсь, даже у меня был один такой проект, я уже не помню, что именно за сайт это был, это было около 6 лет назад, но там была парочка довольно внушительных таблиц. Вот я тогда их сверстал дивами, и ходил гордился собой. Как говорят сейчас, «его потом в дурку забрали, конечно».

Так вот, тот период ознаменовался тем, что верстальщики забыли о существовании других тегов. В итоге даже сейчас часто можно увидеть на большинстве сайтов, особенно трехлетней давности и больше, жуткую кашу из одних дивов. Картину немного разбавляют разве что теги <h1>, <h2>, которые понатыкали в какой-нибудь SEOшной конторе, «в целях поисковой оптимизации». Я, кстати, тоже таким занимался, это было 7 лет назад, на первом моем месте работы, было такое дело.

И тут, внезапно, в этот «диватоз» врывается стандарт HTML5 со своими новыми тегами.

Вообще, есть две "группировки", развивающие стандарты, W3C и WHATWG. W3C следует некоей верификации и при этом ужасно тормозит с их выпуском, при том что WHATWG развивает "живую" версию. У этих огранизаций имеются некоторые разхождения во взглядах на развитие спецификаций, но это отдельная и длинная история. Так вот, официально последняя "живая" версия HTML - HTML5.1, и в ней вроде бы последний введенный тег - это тег <main>. Он указывает на основное содержимое страницы, главную его часть. Это самый новый тег, и нынче решается судьба еще нескольких семантических тегов. Если раньше вы создавали между шапкой и футером сайта div с классом "content" или каким-то похожим, то теперь для этого имеется отдельный тег, предназначенный именно для этого.

За последние пару лет эти теги перестали быть чем-то вроде диковинок, к ним привыкли у уже вовсю используют. О них появилось достаточно информации о том, в каких именно ситуациях следует использовать тот или иной тег. Если вы по каким-то причинам еще не используете их, начните это делать прямо сейчас. Это как широко известные <article>, <section>, <header>, <footer>, <nav>, так и менее известные, вроде <details>, <summary>, <picture> или <progress>.

Для чего они вообще, эти новые теги. Они призваны сделать страницы еще более читабельными и доступными - в первую очередь, для поисковых роботов и различных устройств, особенно предназначенных для людей с ограниченными возможностями. Если с первым все уже хорошо, и с каждым днем становится лучше, то со второй частью - с адаптацией сайтов для людей с ограниченными возможностями, - пока все довольно плохо.

Это пока слабо развитая отрасль веба, точнее, она слабо развита именно на уровне поддержки этих стандартов именно разработчиками. Сами стандарты уже давно написаны и только и ждут, когда же их начнут внедрять на страницы. С их внедрением есть несколько очевидных проблем, по которым подавляющее большинство веба еще не оптимизировано практически никак для людей с ограниченными возможностями. В первую очередь, это неочевидность кейсов и проблема адекватного тестирования таких оптимизаций, к тому же для людей с различными проблемами эти оптимизации могут быть совсем разными. Многие не готовы пока вкладывать деньги в подобную оптимизацию, хотя число таких людей в интернете постоянно возрастает. Вообще, на тему есть отличный доклад Романа Лютикова "Доступный веб для всех", он не так давно прозвучал на Web Standards Days в декабре.

http://webstandardsdays.ru/2014/12/06/pres/web-a11y/

Следование новым стандартам, пусть хоть и немного, но все же сделает ваш сайт доступнее. В будущем, я уверен, это будет иметь огромное значение как для результатов выдачи и получения роботами нужной информации со страниц, так и для тех пользователей сайтов, которые в этом нуждаются.

4. Единообразие в написании HTML

Правда, в этой "свободе синтаксиса", которую принесла с собой новая версия HTML, ее сильные стороны при некоем усилии могут стать слабыми сторонами. Так, новая версия позволяет по-разному записывать атрибуты тегов, написание самих тегов и т.д. Например, как известно, браузеру плевать, капсом будет написано имя тега или нет. Помимо этого, у многих людей еще со времен знакомства с XHTML осталась привычка записывать логические атрибуты именно как «название - значение», плюс к этому, одно и то же на выходе может быть получено разными способами, например, checked=”checked” или checked=”true” на выходе даст одно и то же. Потому, если в команде разные люди пишут код по-разному, да еще и в одном файле, то со временем появляется желание кого-то убить. HTML5 решает эту проблему

https://twitter.com/pepelsbey/status/561178362237157376

Намного более остро стоит проблема использования кавычек. Одинарные или двойные, а чаще всего это адская смесь и того и другого. Это касается не только HTML, и, может быть, наиболее остро проблема стоит не в HTML, но она существует, и это просто адский ад. Дополняет этот хаос еще и копипаст. Хуже всего то, что проблема не решается, например, постпроцессорами, потому что изменения нужно производить не в конечном файле, а в исходнике. Вроде в этом и нет проблем в плане распознавания браузерами, главное только, чтобы на одном уровне открывающая и закрывающая кавычки были одинаковыми, это понятно. Тем не менее, эта каша очевидно не делает ваш код лучше и читаемее.

Плюс, расширились правила для названий классов и id элементов. Использовать их - это не очень хорошая идея, но тем не менее. Мы еще подымем этот вопрос чуть дальше.

5. HTML5 формы

Стандарт HTML5 существенно расширил возможности для более удобного взаимодействия пользователя со страницей, как то введение новых видов элементов форм. Например, если взять пример выше, и заметить, что в одном из полей, судя по всему, требуется ввод именно номера телефона. И, если вы, например, будете делать это с девайса - с телефона или планшета, вам будет не очень удобно это делать, поскольку вы будете видеть перед собой обычную клавиатуру. Определение типа этого поля как email даст браузеру девайса понять, что от польхователя требуется ввести номер телефона, и в этом случае ему будет вместо огроменной дефолтной клавиатуры предложен более компактный, и, главное, более удобный вариант, на котором будет лишь ввод цифр. То же самое относится и к других похожим типам элементов форм.

Более того, появились и интерактивные типы элементов, например, <input type=”color”>, <input type=”range”>. Пока, конечно, существует проблема с их кроссбраузерностью, а также поддаванию стилевого изменения, но в этом плане все впереди.

Плюсы их использования очевидны – это, во первых, более удобная работа с сайтом, и нативная поддержка, без кучи js-плагинов вроде тех же слайдеров, имитирующих <input type=”range”>.

Да, и еще одно – теперь у элементов форм есть атрибут pattern, в котором можно задать (с помощью регулярных выражений) список допустимых символов, их количество и порядок – то есть вся сила регулярок прямо из коробки. А в объединении с не такими уже и новыми, но все же, CSS3-селекторами вроде :valid, :invalid и других, появилась возможность валидировать формы прямо из коробки, средствами только HTML и CSS, не прибегая к помощи JS, для которого сейчас хватает других задач. Естественно, это не отменяет того факта, что пришедшие данные нужно проверять еще средствами сервера, но это уже хотя бы освобождает нагрузку на браузер, и позволяет отказаться от всяких Form Validation плагинов, хотя бы в тех браузерах, которые это поддерживают.

6. Использование тегов не по назначению.

Здесь даже особо и нечего говорить. Это очень распространенная проблема многих верстальщиков – объединение понятий кнопки и ссылки, и замены одного элемента другим в ситуациях, где это не нужно. Например, у вас на странице есть попап, у которого в углу есть элемент для его закрытия. Почему бы не сделать его ссылкой, не правда ли? Или есть форма и есть кнопка, по нажатию на которую нужно отправить данные на сервер. Ссылка? Why not? Или есть на странице какой-нибудь текст, который вы спрятали за спойлер. Его внешний вид очень похож на ссылку, давайте сделаем его ссылкой? Все это – неправильные выводы. «Ссылка» должна быть ссылкой только в том, случае, если при клике на нее должен происходить переход на другую страницу. Если речь идет о взаимодействии с интерфейсом, оставаясь при этом на странице, то это что-угодно другое, но не ссылка. Это и так называемые псевдоссылки, и кнопки, но не ссылки. Если по клику на «ссылку» должна произойти отправка данных на сервер, то логичнее это делать не средствами JS (при которых блокируется дефолтное поведение элемента и перехватывается созданное им событие), а дефолтными возможностями браузера, которые будут работать вне зависимости от включенного JS и любых других факторов.

На эту тему есть уже старенький доклад Вадима Макеева, который называется «Жми сюда!», в котором речь идет как раз о неправильном использовании этих элементов.

7. Микроданные

Отдельная возможность улучшить доступность сайта - внедрение разметки микроданных. Микроданные - это способ семантически размечать сведения о разнообразных сущностях (событиях, организациях, людях, товарах и так далее) на страницах, используя стандартные элементы языка HTML. Изначально он был создан компаниями Google, Bing и Yahoo в мае 2011 года, - тогда они заявили о поддержке микроданных и вместе создали сайт schema.org. На этом сайте собран перечень всех поддерживаемых сущностей, с примерами использования.

8. Data-атрибуты

Также стандарт HTML5 привнес с собой возможность записи любой нужной информации на странице в специальные data-атрибуты. На самом деле, судя по тестам, особого практического профита в использовании в JS-коде именно data-атрибутов, а не обычных кастомных, нету, но их быстродействие в любом случае - это шестизначное число операций в секунду, потому, думаю, это не приниципально. Суть в другом - такие атрибуты валидны, и просто-напросто они удобнее. При виде таких атрибутов в коде из можно сразу выделить среди остальных служебных, дефолтных, и, соответственно, лучше понимать код.

CSS

9. Общие проблемы CSS

В 2014м году распределение количества HTML и CSS кода составляло примерно 15% к 85% - то есть фактически все время мы проводим не за написанием HTML, а за украшательством и оформлением его в CSS. Потому и говорить о CSS можно больше и дольше, но начнем, пожалуй, с самых общих вещей, касающихся CSS.

Во-первых, CSS Reset. Это что-то вроде аналога доктайпа для HTML, это то, что должно быть на каждой странице. Дело в том, что разные браузеры (и разные их версии) по-разному отображают многие элементы, применяя к ним разные свойства вроде шрифтов, их размеров, отступов или фонового цвета. Эти наборы правил называются useragent stylesheets и являются встроенными в браузер, с тем, чтобы браузер хоть как-то, по-дефолтному, отобразил страницу, если на ней по каким-то причинам не подключились стили. Так вот, CSS Reset – это набор стилей, в котором указаны правила для «сброса» этих правил с каким-то общим начальным значениям. Существует несколько их видов, о которых вы все слышали.

Первый – global White-Space Reset. Все очень просто, одно правило, ясно и понятно. Проблема одна – он проходится абсолютно по всем элементам, что сказывается на производительности, хотя и не сильно (пару лет назад пробегала цифра в 5%). Есть еще и вторая, но она затрагивает еще несколько других вариантов.

Следующий – Eric Meyer’s CSS Reset (//meyerweb.com/eric/tools/css/reset/reset.css)

В нем перечисляется список тегов, для которых следует сбросить правила. Лучше, но все равно не то. Плохо в нем то, что для некоторых тегов, которые пользователи привыкли видеть в их дефолтном виде, происходит сброс этих настроек, и, соответственно, для того, чтобы в необходимых местах вернуть элементам им изначальных, родной вид, требуется заново его назначать. Немного глупо.

Следующим поклением стал Normalize.css, написанный Николасом Галлахером почти 4 года назад. Качественное отличие его подхода от других – normalize.css, как понятно из названия, не просто сбрасывает стили элементов, а приводит их к неким нормам, неким негласным правилам, некоторые из которых берут свое начало еще из типографии. Сейчас это де-факто стандарт для обнуления, а точнее, нормализации стилей, и на данный момент существует уже третья версия. Помимо очевидного качественного прогресса, это совершенно открытый продукт, проект находится на гитхабе, он отлично задокументирован и протестирован большим количеством разработчиков, следовательно, потенциальный шанс словить какое-то неправильное поведение того или иного элемента при использовании normalize.css стремится к нулю.

Нынче использовать normalize.css не то, чтобы модно – это уже норма. Конечно, если у вас имеется совсем неболшая страница, которой достаточно global white-space reset’a, будет логичнее его и использовать.

Отдельным пунктом стоят так называемые «самописные» резеты. Обычно это Мейеровский резет, в конец которого дописаны еще несколько правил «по нужде». Последнее слово, конечно, за вами, но как по мне, это уже какое-то наследние прошлого - быть самому наедине со своим резетом. Я в свое время тоже через это прошел, у меня ровно такая же история, но однажды я словил какой-то баг в связи с выходом новой версии FF – то ли это была какая-то версия 3.6, то ли 4я. Суть в том, что мне пришлось тогда убирать одно правило по нескольким проектам, которое я считал «вечным», думал, что его поведение и использование никогда не поменяется.

Кроме резетов, есть одна довольно забавная общая вещь. Это количество подключаемых CSS-файлов и количество CSS-селекторов в одном файле. Дело в том, что IE9 и ниже на одной странице может подключить только 31 файл (через <link> и @import), а также один файл может содержать не более 4095 селекторов. Если будут превышения лимитов, остальщные правила или файлы просто не будут применены. Это, казалось бы, что-то совершенно экзотическое, но я сам с этим столкнулся только три месяца назад. На проекте после очередного спринта добавилась очередная порция CSS-кода, который на страницах собирается средствами сервера в один CSS-файл в целях оптимизации, и тестировщики обнаружили странный баг, связанный с тем, что часть страниц начали неправильно отображаться в IE (мы поддерживаем на данный момент IE8+). Мы полдня думали, в чем проблема, пока мне не стукнуло в голову сосчитать количество селекторов в результирующей CSS, их оказалось чуть больше 4100. Мы разделили финальный файл на два, и проблема исчезла. Так что иногда такие знания тоже могут спасти вам часы дебагов и нервы.

10. CSS-селекторы

Теперь перейдем уже к самому написанию CSS и ошибкам, связанным с этим. Основное, что мы делаем в CSS – мы используем те или иные селекторы. Существует уже четвертая версия специцикации CSS Selectors, которая пока находится в стадии Working Draft. Нам уже доступны селекторы целых трех версий модуля. С каждым разом нам предлагают все более крутые возможности управления теми или иными элементами, которые уже обросли логикой взаимодействия страницы с пользователем и различными фильтрами.

CSS1 – простейшие селекторы вроде селекторов по тегу, по классу и по ID, а также каскад.

CSS2 – селекторы посложнее – по атрибуту, первая фильтрация – так называемые псевдоселекторы - :first-child, а также крутой каскад – селектор непосредственного потомка. Ну и общий селектор \*, и sibling селекторы.

CSS3 – появляются псевдоэлементы, а также крутые селекторы вроде :not(), :enabled, :disabled для проверки состояния элементов форм, и различные селекторы для определения селектора или группы одинаковых селекторов внутри какой-то большей группы, как то :nth-child(), :nth-of-type() и другие.

И четвертая версия модуля подарит нам еще больше возможностей для взаимодействия с элементами форм, а также долгожданный селектор потомка, который правда претерпел некоторые изменения, - раньше указывался через восклицательный знак, теперь же, похоже, позаимствовал синтаксис и принцип работы из JS.

Парадокс всех этих нововведений заключается в том, что браузеры с разной скоростью обрабатывают те или иные селекторы, и хотя сейчас скорость обработки селекторов, пожалуй, достигла небес, тем не менее,самыми быстрыми являются селекторы по ID и по классу. Разработчики браузеров, несомнено, работают над оптимизацией обработки новых селекторов.

Помимо этого, многие псевдоселекторы жество задают тот или иной порядок следования элементов в коде, а порой даже жестко привязываются к тому или иному тегу, что негативно влияет на переносимость и переиспользуемость кода, - чтобы поменять порядок элементов, добавить или убрать ненужное, приходится помимо правок в HTML вносить правки еще и в CSS, причем не всегда это можно сделать легко.

Единственный селектор, ставший очевидно быстрее похожего на него более «старого» селектора – это селектор непосредственного потомка. Фактически всё остальное – это только утяжеления CSS, как плата взамен на более который или элегантный код.

Использование чрезмерного каскада также, как вы знаете, обладает негативным эффектом, ведь считываются селекторы справа налево, потому любой каскад будет медленнее такого же селектора, записанного без каскада.

Потому, я часто вижу две крайности – это либо супер-точечные селекторы, которые работают «только здесь и только сейчас», и которые к тому же при необходимости будет еще труднее переопределить, либо слишком общие селекторы, особенно это касается селекторов по тегу. Такие селекторы обладают плохой переиспользуемостью и переносимостью, - почти всегда при переиспользовании приходится вносить правки в CSS – дополнение непосредственно селекторов или переопределение правил внутри.

11. Поддержка legacy-кода

Ну, во-первых, его, конечно, можно просто не поддерживать. Но пока большинство сайтов по карйней мере на территории СНГ требуют поддержки IE8, потому что Win XP еще жива, несмотря на уже официальное окончание поддержки ее самой Microsoft, то есть MS больше не будет выпускать патчей и латать дыры в безопасности. Сейчас доля IE8 в уа-нете равна примерно 1%, чуть меньше. Много это или мало – решать вам.

Сейчас уже и IE9 можно считать устаревшим браузером, но в этом плане MS можно даже похвалить, поскольку они уже проапгрейдили IE9 до IE10, а затем и до IE11, который уже довольно неплох и в плане быстродействия, и в плане поддержки стандартов, а в сравнении с тем же IE8 это просто небо и земля.

Тем не менее, иногда необходимо подправить что-то для старых версий IE или хоть как-то оформить критически важные элементы на странице. Я говорю сейчас только об IE, потому что понятно, что проблема с устаревшими версиями для других браузеров не стоит так остро, как для IE.

В эпоху господства IE дело с кроссбраузерностью обстояло значительно хуже, и то и дело приходилось использовать так называемые CSS-хаки, специальным образом записанные CSS-правила или селекторы, которые воспринимались и обрабатывались только определенной версией IE.

Они были разные и их было много, и слава богу что их больше нет. Тем не менее, только спустя несколько лет после их повсеместного употребления, появилась методика, как это делать понятнее и правильнее. Речь пойдет о conditional comments, которые поддерживаются только в старых версиях IE, включая IE9. IE10 уже не поддерживает их. Этот способ гораздо понятнее и читабельнее любого CSS-хака. Плюс к тому, CSS-хаки потому и названы хаками, что они по сути являются незадокументированным и неправильным поведением браузера. Соответственно, в будущем это поведени может стать ожидаемым для тех или иных действий.

Иногда это делается с помощью тех же CC, только происходит подключение еще одного внешнего файла. Ну мы же с вами хотим загрузить все как можно быстрее, правда? Так зачем нагружать сервер лишними запросами, если к нам пришли со старого IE, можно перевесить эту нагрузку (которая, к слову, будет минимальной) на браузер.

12. Progressive enhancement и graceful degradation

В продолжение темы поддержки legacy-кода, существует несколько принципов, которые определяют, по каким критериям можно построить уровень поддержки различных браузеров. Это graceful degradation и progressive enhancement. В общем-то, это два названия одного и того же, с той лишь разницей, что одно оглядывается назад, другое – бежит вперед. Самый простой пример первого – это элемент, у которого в качестве фона указан градиент. В IE8 его не будет, и по принципу graceful degradation можно либо указать ему в качестве фона просто цвет, либо картинку. Пример второго – скажем, использование директивы @supports.

13. Вендорные префиксы

Эпоха вендорных префиксов уже, хочется надеяться, прошла свой пик и сейчас испытывает закат, тем не менее, разрабатываются новые и новые стандарты, и чтобы понять, достойную ли штуку придумали эти «теоретики», эту штуку нужно обкатать, а также пофиксить возможные баги. Собственно, для этого создатели браузеров еще на этапе только создания чернового варианта спецификации внедряют тот или иной стандарт, но прячут его за префиксом, как бы намекая, что эта вещь пока экспериментальная, и использовать ее можно на свой страх и риск. В следующей версии браузера ее поведение может быть изменено или она вообще может оказаться выпиленной.

Тем не менее, сейчас еще довольно большое количество правил, которые уже широко используются при верстке, требуют наличия тех или иных вендорных правил. Раньеш ситуация была еще плачевнее, огромное количество верстальщиков к каждому новому правилу, где встречалась хоть одна вендорная версия, дописывали и все остальные правила, которые были им знакомы, даже если таких правил вовсе не существовало. Самый яркий пример – это border-radius, который уже поддерживается абсолютно всеми браузерами без вендорных префиксов.

Также, в сети существует сайт caniuse.com, на котором собран список правил и стандартов с указанием уровня их поддержки в том или ином браузере (и их версиях), а также статистика по распространению той или иной версии браузера.

В последнее время написание этих вендорных правил, во первых, упростилось с помощью использования препроцессоров, где в служебных миксинах можно один раз собрать все необходимые версии записи того или иного правила с нужными префиксами, и затем использовать по всему проекту, а также с помощью постпроцессоров, пожалуй, самого известного среди них – autoprefixer, который, в зависимости от настроек, сам анализирует код и проставляет нужные префиксы в заранее подготовленном файле.

Тем не менее, иногда возникают ситуации, когда приходится иметь дело с чистым CSS, где не внедрен ни препроцессор, ни постпроцессор. Здесь я склоняюсь к тому, что использование инструментов – это, конечно, здорово, только следует понимать, зачем они используются и какая работа ими осуществляется. Иначе, оказавшись в один день наедине с проблемой, можно попасть впросак.

14. Общие правила записи CSS-свойств

15. Пространство имен классов

16. Препроцессоры и постпроцессоры

Препроцессоры существуют с нами уже больше 5 лет, и собственно причины их возникновения кроются в том, что сам по себе CSS и развитие его стандартов не давали разработчикам всей необходимой для разработки мощи. Сейчас они значительно ускоряют процесс разработки и облегчают поддержку проекта.

Постпроцессоры с нами не так давно, но они уже прочно заняли свою нишу в инструментах для разработки сайтов. Это принципиально новая идея, плюс безграничные возможности для написания любого интрументария. Это всё работает на Node.js, и к вашим услугам вся мощь языка javascript.

17. Методология

Надеюсь, вы еще не уснули.

На самом деле можно о многом еще поговорить – я почти не затронул тему, например, мобильных устройств и их поддержки, поддержки ретины, использования смежных технологий вроде SVG, и многое другое. Разговаривать об этом можно еще долго.  
  
Завершая свой воркшоп, я хочу вам вот что сказать.   
  
Всегда считайте, что на вашем сайте есть пользователь, у которого имеются все возможные ограничения и лишения, которые вы можете себе представить. Он сидит из-под IE6-7, политиками безопасности у него отключен JS, возникают проблемы с загрузкой картинок и шрифтов, а еще куча пользовательских настроек, каких-нибудь флагов или еще чего-нибудь. Он очень любит ресайзить окно браузера, и постоянно меняет масштаб страницы. Это не значит, что нужно малейшую мелочь вычищать до блеска – естественно, это просто-напросто бред. Да и число таких РЕАЛЬНЫХ пользователей уменьшается день ото дня. Но можно хотя бы при внедрении какой-нибудь фичи с использованием какого-нибудь ES6 или написании раскладки на флексбоксе стоит просто задать себе вопрос «а что будет, если это не будет работать, и каков возможный процент таких людей?». Разрабатывая продукт, я считаю, вы должны знать, под какие браузеры и системы вы его делаете. И должно быть понимание того, что, отбрасывая те или иные версии браузеров или платформ, вы потенциально можете потерять некий процент клиентов. Притом, оно должно быть у всех, начиная от владельца продукта или заказчика. Если в вашей компании за это отвечаете вы, вам нужно отслеживать тенденции изменения динамики пользователей, используемых ими браузеров и платформ. Расскажу на своем примере, когда я где-то 3 года назад пришел в один из крупнейших порталов уа-нета на должность лида по верстке, передо мной была сразу поставлена задача оценить процент пользователей, полная адаптация продуктов портала для которых является сложнее или дольше определенного значения, с тем, чтобы просто-напросто не работать себе в убыток – и это нормально, нет смысла гнаться за полной поддержкой верстки, например, под IE6, который имеет 0.4% пользователей, когда у вас тут кругом CSS3. Собственно, тогда я и решил сразу отказаться от поддержки IE6 (которая до моего прихода осуществлялась), а где-то чуть больше года назад мы отказались и от поддержки IE7, у которого на тот момент тоже было примерно 0.5% пользователей. То есть, новые продукты по требованиям не должны были иметь полностью такой же внешний вид, как, скажем, последняя версия Chrome. Должна лишь соблюдаться работоспособность критических составляющих продукта, вроде возможности логина, или возможности прочесть необходимые тексты, нажать на необходимые кнопки, - пусть они будут без подключенных веб-шрифтов, без скруглений и остальных украшательств, но они будут доступными.  
  
Ну и в завершение процитирую одного умного человека, который сказал: «Пишите код так, как будто сопровождать его будет склонный к насилию психопат, который знает, где вы живёте».

18. И помните, самый важный инструмент в разработке – это то, что находится у вас на плечах.

Спасибо.