**Для того чтобы отдавать вёрстку клиенту, достаточно соблюдения первых 4 пунктов. Для выдачи в продакшн — первых 6.**

0. Соответствие макету

1. Кроссбраузерность, кодировка и DOCTYPE
2. Валидность, доступность, микроформаты
3. Независимость блоков в CSS: минимизация каскада, использование техник БЭМ/MCSS/SMACSS
4. Сайт должен нормально смотреться во всех стандартных разрешениях от 1024 и выше, не иметь горизонтального скролла и вписываться в экран мобильных устройств
5. Корректная работа при вбивании реального текста, надёжность вёрстки
6. Проверка и оптимизация скорости загрузки
7. Наличие Win/Mac/Linux-аналогов шрифтов
8. Доступность при выключенных (загружающихся) картинках
9. HTML5 формы, линковка, валидация
10. Семантичность. Отсутствие глупостей в html и css, единообразие, аккуратность
11. Правильная структура заголовков (H1,H2,… и т.д. и TITLE)
12. Работоспособность при выключенном JavaScript
13. Работоспособность при выключенном Flash
14. Отсутствие багов при увеличенном шрифте
15. И последний пункт – мелкие проверки (ниже подробней)

**0. Соответствие макету**

Pixel perfectРасположение блоков должно быть 1:1 по сравнению с макетом. Допускается расхождение до 5px для текста. Разрешены и даже приветствуются правки размеров и расположения криво нарисованных блоков (разница размерах в 1-2px на разных страницах).  
  
При изменениях контента размеры блоков могут меняться (по высоте, например), это нормально. Pixel Perfect используется для проверки, что все базовые блоки находятся на своих местах, их размеры и отступы соответствуют макету.  
  
Проверяется в Firefox через плагин [Pixel Perfect](https://addons.mozilla.org/ru/firefox/addon/pixel-perfect/). Для проверки в других браузерах используйте [ModularGrid](https://github.com/aishek/modulargrid), но в принципе достаточно просто взглянуть намётанным глазом, если расхождения заметные — они будут бросаться в глаза.

**1. Кроссбраузерность, кодировка и DOCTYPE**

* **Кодировка: UTF-8**  
  Как проверяется: в FF Инструменты→Информация о странице, в появившемся окне должно быть написано «Кодировка: UTF8». Эта кодировка должна использоваться для всех файлов: html, css, js (если файлы в разных кодировках, могут быть проблемы)
* **DOCTYPE: HTML5**  
  HTML5 — это современный стандарт, в нём можно писать и в строгом XHTML-синтаксисе. [Аргументация для сомневающихся](http://vimeo.com/19072506).  
  Проверка: открываем исходный код страницы, первая строка должны быть <!DOCTYPE HTML>.
* **Кроссбраузерность:**
  + Firefox (последний)
  + Chrome (последний)
  + Safari (последний), и если у вас нет Mac, чтобы проверить «размытые Mac'овские» шрифты (они чуть большего размера, из-за этого бывает, что вылазят баги), то установите в Preferences→Appearance, «Font Smoothing» в Medium (по дефолту там «Windows Standart»).
  + iPhone (смотрим в landscape и portrait режимах, т.е. вертикально и горизонтально) + Android. Для этого нужно указать [правильный viewport](https://vimeo.com/33180097).
  + Opera (последняя). Имеет смысл также посмотреть на 12-й версии с движком Presto, если там есть баги в отображении — это признак потенциальных проблем в качестве кода
  + IE7+ (для IE6 выводится уведомление о неподдержке и предложении скачать другой браузер или установить Google Frame, но с возможностью всё-таки просмотреть сайт!)
  + Проверка в IE7 делается переключением IE8 в режим IE7 (F12→Режим обозревателя→Internet Explorer 7).
  + В IE6 можно посмотреть на [ipinfo.info/netrenderer/](http://ipinfo.info/netrenderer/) или на виртуальной машине (удобно использовать Windows XP Mode в Win7).
  + Opera Mini (проверяется в [Opera Developer Tools](http://www.opera.com/developer/tools/)→[Opera Mini Simulator](http://www.opera.com/mobile/demo/), нужно установить Java плагин к браузеру, или в крайнем случае: Opera 9.64→Вид-Маленький экран, но в 9.64 JS будет работать полноценно в отличие от настоящей Opera Mini, это нужно учитывать)

#### 2. Валидность, доступность, микроформаты

1. **Не должно быть js-ошибок!**
2. **Титульная страница должна быть валидна в любом случае.** Ошибки на внутренних страницах простительны в следующих случаях:
   * Вставлены произвольные тексты с произвольным форматированием;
   * Программисту необходимы какие-то кастомные атрибуты (хотя для этого в HTML5 ввели специальные пользовательские атрибуты «data-\*»).
3. **CSS валидируется по версии 3.0, его валидность не требуется**, а вот синтаксические ошибки исправлять нужно.
4. **Микроформаты должны быть.** Как минимум – hCard (<http://microformats.org/wiki/hcard>). Желательно также hCalendar, XFN, hAtom. Микроформаты не только полезны для SEO, но и здорово упорядочивают код. Не нужно полчаса думать как назвать новый блок. Выбирай из существующих стандартных имён! Микроформаты постепенно вытесняются [microdata](http://habrahabr.ru/post/141216/), стоит использовать и то, и другое.
5. Проверка статическими анализаторами качества кода: [CSSLint](http://csslint.net/) и JSHint

Ошибки js проверяются просмотром сайта в IE – в левом нижнем углу не должно быть значка «есть js-ошибки».

Валидность проверяется онлайн-валидаторами:

* HTML: [validator.w3.org/](http://validator.w3.org/) (или [Web Developer](https://addons.mozilla.org/ru/firefox/addon/web-developer/) →Инструменты →Проверить HTML)
* CSS: [jigsaw.w3.org/css-validator/](http://jigsaw.w3.org/css-validator/) (или [Web Developer](https://addons.mozilla.org/ru/firefox/addon/web-developer/) →Инструменты →Проверить CSS)
* Наличие микроформатов проверяется плагинами [Operator](https://addons.mozilla.org/ru/firefox/addon/operator/) и [Tails Export](https://addons.mozilla.org/ru/firefox/addon/tails-export/).  
  Валидаторы микроформатов:
  + [microformatique.com/optimus/](http://microformatique.com/optimus/)
  + [www.google.com/webmasters/tools/richsnippets](http://www.google.com/webmasters/tools/richsnippets)
  + [webmaster.yandex.ru/microtest.xml](http://webmaster.yandex.ru/microtest.xml)
  + [hcard.geekhood.net/](http://hcard.geekhood.net/)

**В идеале вёрстка должна соответствовать стандарту доступности: WCAG.** Он имеет три уровня сложности, если проходит хотя бы WCAG1 A (Priority 1) – уже хорошо. Идеальный вариант – WCAG2 Priority 3 (AAA). Самый простой способ проверить, что, вероятно, WCAG1 Priority A соблюдён — [www.cynthiasays.com/](http://www.cynthiasays.com/) (или [Web Developer](https://addons.mozilla.org/ru/firefox/addon/web-developer/) →Инструменты →Проверить WAI). Почему «вероятно»? Некоторые требования WCAG невозможно проверить автоматически. Вот ещё несколько скриптов помощников:

* [Check a Site: scan web sites for over 450 quality problems](http://try.powermapper.com/demo/sortsite.aspx)
* [Total Validator: (X)HTML validator, an accessibility validator, a spell checker, and a broken links checker](http://www.totalvalidator.com/tools/index.html)
* [Validación de accesibilidad de acuerdo a las WCAG 2.0 con PISTA](http://olgacarreras.blogspot.com/2009/03/validacion-de-accesibilidad-de-acuerdo.html)

Чеклист WCAG2:

* [www.w3.org/TR/WCAG20/#guidelines](http://www.w3.org/TR/WCAG20/" \l "guidelines)
* и детальнее: [www.w3.org/WAI/WCAG20/quickref/](http://www.w3.org/WAI/WCAG20/quickref/)

##### Некоторые ошибки в валидации допустимы.

###### HTML

Стандарт HTML5 находится в активной разработке. Валидатор HTML5 меняет правила проверки в соответствии с этим.  
То, что было валидным вчера, сегодня может выдавать ошибку, например такая ситуация сейчас с [apple-touch-icon](http://habrahabr.ru/qa/8128/) и [XFN](http://habrahabr.ru/qa/8349/).  
В отличии от HTML4 и XHTML, [официальной кнопки «Valid HTML 5» не существует](http://www.w3.org/QA/Tools/Icons), поэтому валидатор не выдаст вам код для её вставки, даже если он считает документ валидным.  
[Люди сами рисуют свои варианты кнопочек](http://notepad.patheticcockroach.com/1733/a-valid-html-5-button-plus-a-valid-css-one/), вы можете использовать любые, но рекомендованным вариантом на сегодняшний день является [добавление на сайт официального HTML5 badge с лентой используемых технологий](http://www.w3.org/html/logo/).

###### CSS

* 1. **По-умолчанию валидатор CSS проверяет код согласно стандарту 2.1, а не 3.**
  2. **Валидатор считает ошибкой указание** [**вендорных префиксов**](http://web-standards.ru/articles/prefix-or-posthack/)
  3. **Раньше проприентарные свойства IE было рекомендовано выносить в отдельный CSS. Сейчас стоит использовать HTML5 Boilerplate и фильтровать в style.css с помощью html.oldie, html.ie7 и т.д.**

#### Настройки CSSLint:

Выключить опции:   
— Beware of broken box sizing  
— Disallow adjoining classes  
— Disallow empty rules

#### Настройки JSHint:

Выключить:   
— When code is not in strict mode  
  
Включить:   
+ jQuery

#### 3. Независимость блоков в CSS: минимизация каскада, использование техник БЭМ/MCSS/SMACSS

Проверяется в FF через плагин [Firebug](https://addons.mozilla.org/ru/firefox/addon/firebug/)  
При наведении на любой блок, в его стилях не должно быть множество перечёркнутых правил (следствие длинного каскада).  
Для минимизации каскада и построения надёжной, современной, масштабируемой вёрстки сейчас применяют следующие техники: [БЭМ](http://ru.bem.info/), [MCSS](https://github.com/operatino/MCSS) и [SMACSS](http://smacss.com/).

#### 4. Сайт должен нормально смотреться во всех стандартных разрешениях от 1024 и выше, не иметь горизонтального скролла и вписываться в экран мобильных устройств

Проверяется в FF через плагин [Web Developer](https://addons.mozilla.org/ru/firefox/addon/web-developer/)→Resize  
Список классических разрешений:

* 1024x600
* 1024x768
* 1152x864
* 1280x800
* 1280x1024
* 1440x900
* 1680x1050
* 1920x1080

На больших разрешениях нужно обращать внимание что не пропадает фоновая графика по краям сайта (что размера картинки хватило на большое разрешение).  
  
Вписывание в экран мобильных устройств лучше проверять на самих мобильных устройствах. Множество проблем решается указанием верного Viewport.

#### 5. Корректная работа при вбивании реального текста, надёжность вёрстки [для верстки, которая будет натягиваться на CMS]

Вёрстка должна тянутся, не разваливаться и не терять дизайнерский вид при изменении контента на странице. Его может быть больше или меньше, чем на макете, он может быть обёрнут во всякие <p> из визига и т.п.  
Обязательно нужно проверять удаление заголовков! Бывает, что отступы между блоками после этого схлопываются, это частая ошибка, причина — что отступы были заданы не для блоков, а для внутренних элементов — заголовоков.

1. **Проверка ввода и удаления данных.**  
   На странице с контентом пробуем добавлять и удалять содержимое – «что будет, когда текста много/мало?».   
   Обязательно пробовать менять расположение элементов, чтобы после того, как блоки поменялись местами, не развалилось оформление (из-за каскада).
2. **Проверка корректности работы стилей.**  
   На страницы с контентом вбиваем текст с абзацами и без абзацев (т.к. стили могут быть прописаны только для абзацев, но не для текста), со списками и картинками, таблицами и заголовками разных уровней.

Это нужно, чтобы на живом сайте потом не вылезли проблемы при заполнении реальными данными.  
Правки содержимого страницы делаются в FF через плагин [Firebug](https://addons.mozilla.org/ru/firefox/addon/firebug/): HTML→Edit – меняем/добавляем/удаляем текст. Хороший пример проверочного текста находится в [normalize.css](https://github.com/necolas/normalize.css/blob/v2.1.1/test.html) в test.html между <body> и </body>.  
  
Хорошо **использовать html5-тэги для разметки**: header, footer, aside, nav, section, article и т.д. Кроме того, что это семантично, также повышается надёжность, «пуленепробиваемость» вёрстки. Лишний открытый или закрытый div легко может поломать вёрстку. Но когда каркас сайта — атомарные и редко повторяющиеся html5-тэги, то «поломка» локализуется в пределах html5-тэга.

#### 6. Использование препроцессоров и систем сборки

CSS должен быть написан с использованием препроцессоров (LESS/Sass/Stylus). Проверяется поиском файлов с расширениями вида: .less, .sass, .scss, .styl.

Желательно использование систем сборки (Grunt/Gulp/Webpack) и построцессоров (PostCSS/Autoprefixer). Проверяется поиском файлов Gruntfile.js или Gulpfile.js

#### 7. Оптимизация скорости загрузки

* для мелких картинок, связанных логически, должны использоваться CSS-спрайты (например, набор буллетов или вариации отображения пункта меню: дефолтный, активный)
* Base64-encode для мелких отдельных картинок;
* CSS3 border-radius, gradient, box-shadow, text-shadow вместо использования графики;
* Оптимизация PNG и JPG файлов;
* JS максимально должен быть вынесен во внешние файлы, внешние js-файлы разумно объединены для уменьшения количества запросов. JS-библиотеки, например - jQuery, нужно грузить с Google CDN. Постарайтесь включить отложенную загрузку для большинства JS.

Зачем это нужно:

скорость загрузки оказывает ключевое влияние на доступность сайта (больше психологическую, чем фактическую), активность пользователей на сайте (медленными сайтами люди предпочитают не пользоваться) и его конверсию (медленным сайтам не доверяют).

В целом скорость загрузки проверяется так:

* по панели Net в [Firebug](https://addons.mozilla.org/ru/firefox/addon/firebug/)  
  Необходимо проверять, [как отображается страница при загрузке на малых скоростях](http://www.uselessapplications.com/en/Application/FirefoxThrottle.aspx) (хотя бы 64 КБ). Очень часто в такие моменты пользователь видит разъехавшуюся верстку.
* Сервисами [PageSpeed Insights](https://developers.google.com/speed/pagespeed/insights) и [WebPageTest](http://www.webpagetest.org/), основанными на [Google PageSpeed Service](http://code.google.com/speed/pss/)
* [Page speed](http://code.google.com/intl/ru-RU/speed/page-speed/) и [YSlow](http://developer.yahoo.com/yslow/) add-ons в Firebug (учитывайте что значительная часть рекомендаций: включение сжатия, установка определённых headers, minifying кода – относится к серверным работам, а не вёрстке)
* Наличие css-спрайтов проверяется поиском по коду блоков объявлений вида:  
  … {background-position: 0 -30px}  
  … {background-position: 0 -60px}  
  … {background-position: 0 -90px}
* Наличие base64-encode проверяется поиском по коду блоков объявлений вида url(… )
* border-radius, gradient, box-shadow, text-shadow проверяются поиском этих слов в css
* Самый простой способ проверить оптимизацию png/jpg – запустить [готовые скрипты оптимизации графики](http://delka.name/download/IMGoptimize.rar) для картинок вашей вёрстки и сравнить результат с исходными файлами. Можно использовать инструмент Yandex для этих целей — [imgo](http://clubs.ya.ru/ui/replies.xml?item_no=211)
* Если js не объединены в один файл, то Page Speed сообщит об этом: “Combine external JavaScript”.

Разумеется, нужно не забывать очевидные вещи: правильно выбирать тип картинки для сохранения JPG или PNG, выставлять тип Progressive для JPG, не использовать тяжелые (больше 200-300kb картинки).  
Необходимо учитывать что спрайтов, base64 encode и css3-стилей может и не быть по причине ненужности (если макет очень простой).

#### 8. Наличие Win/Mac/Linux-аналогов шрифтов

Часто забывают прописывать альтернативы для Myriad Pro, Arial Narrow.  
Проверяется поиском по тексту css названий “Helvetica”,“Liberation”, “DejaVu”,”Meera”,”Monaco”, “ Century Schoolbook L”,” Nimbus Mono L”, “URW”. Хотя бы два из них должны быть.  
Исправляется указанием готовых наборов шрифтов (css font stack) с <http://cssfontstack.com/>

Также стоит ознакомиться с тем, какие шрифты стандартны для разных OS:

* [Complete Guide to Pre-Installed Fonts in Linux, Mac, and Windows](http://www.apaddedcell.com/web-fonts)
* [CSS font matching: Windows, Mac and Linux](http://my.opera.com/area42/blog/css-font-matching-windows-mac-and-linux)
* [Codestyle: Combined font survey results](http://www.ampsoft.net/webdesign-l/WindowsMacFonts.html)
* [Common fonts to all versions of Windows & Mac equivalents](http://www.codestyle.org/css/font-family/sampler-CombinedResults.shtml)

#### 9. Доступность при отключенных (загружающихся) картинках

Надписи (особенно логотип и главное меню сайта) должны оставаться читабельными, у всех информационных картинок должны быть подписи аккуратным небольшим серым шрифтом (да, для img можно задавать font – это внешний вид alt-текста, что выводится вместо картинки).  
Картинки часто отключают при использовании дорогого интернета, тарифицируемого по траффику (GPRS, роуминг).  
Проверяется в FF через плагин [Web Developer](https://addons.mozilla.org/ru/firefox/addon/web-developer/)→Images→Replace Images With Alt Attributes.

**10. HTML5 формы, линковка, валидация**

1. **Label и input/select должны быть слинкованы.**  
   Это нужно для удобства пользователей. Также это очень облегчает жизнь пользователям с ограниченными физическими возможностями. Проверяется кликом по label – должен активироваться соответствующий ему элемент ввода.
2. **HTML5 валидация формы.**  
   Практическая ценность пока что невелика, ведь серверная проверка ввода данных тоже может быть реализована без перезагрузки страницы (через ajax), но это говорит о профессиональном уровне исполнителя — у редкого числа пользователей современных браузеров с отключенным javascript, проверка ввода данных произойдёт средствами браузера, а не сервера.  
   Проверяется в Opera: выключаем javascript, не заполняем форму, жмём Submit – должны появиться уведомления о необходимости заполнить поля.
3. **JS-валидация формы.**  
   Это ожидаемое поведение. Пользователи привыкли, что если они неправильно заполнят форму, им не дадут её отправить, а укажут на ошибки.  
   Проверяется в Opera/Safari/Chrome: включаем javascript, не заполняем форму, жмём Submit – должны появиться уведомления о необходимости заполнить поля.
4. **Если проверяем frontend в целом — должна быть серверная валидация формы.**  
   Проверяется в Firefox 3.6: выключаем javascript, не заполняем форму, жмём Submit – должны появиться уведомления о необходимости заполнить поля.
5. **Правильные input type=”email/url/tel”.**  
   Пока что практическая ценность для пользователя лишь в том, что на iPhone будет показываться клавиатура, соответствующая формату поля ввода.  
   Проверяется на iPhone — в зависимости от типа поля ввода, он должен показывать различную клавиатуру: стандартную/цифровую/для набора web/email-адресов.

Уведомления об ошибках должны быть текстом в дизайне сайта! Всё оформление в формах должно быть повешено на классы, id — только для линковки с label (иначе потом программисты прикручивают, пишут свои id и ломается внешний вид).

#### 11 Семантичность. Отсутствие глупостей в html и css, единообразие, аккуратность

Пожалуй, единственный пункт, где нельзя дать чётких критериев. Про то, что такое плохо, можно почитать в моей статье «[Вредная вёрстка](http://habrahabr.ru/blogs/webdev/25680/)» (делая скидку, что она написана в 2008 году). Важно понимать, что семантика может быть не только в используемых элементах, но и в именах классов. И БЭМ-иерархия классов — это новый уровень семантики.

Порядок атрибутов

Для удобства чтения HTML-атрибуты должны быть указаны именно в этом порядке:

class

id, name

data-\*

src, for, type, href

title, alt

aria-\*, role

Как ориентир – наиболее частые ошибки:

* Не указывать значение логических атрибутов HTML (checked, disabled, required, selected, readonly, …)
* Для ссылок без адреса вместо # нужно вставлять javascript:void(0);, чтобы страницу не скроллило вверх
* Для внешних ссылок нужно добавлять атрибуты target="\_blank" для открытия в новой вкладке и rel="nofollow" для запрета отслеживания.
* **Самое страшное — float: left для всех блоков.** Проверяется: Web Developer Outline → Float elements, если всё в красных блоках, вёрстку нужно переделывать.
* Отступы между блоками на сайте должны быть за счёт margin у блоков, а не выпирающих наружу margin у содержимого блоков.
* Плохо — отсутствие title.
* Плохо — отсутствие alt у картинок.
* Плохо — стили для IE внутри main.css без фильтров. Т.е. когда просто пишем {zoom: 1;} — это очень плохо, т.к. будет применяться ко всем IE, а не только к тому, в котором проблема. Нужно фильтровать: HTML5 Boilerplate-стили (html.ie7, html.ie8, ...), использовать Conditional Comments, ну или на худой конец (\* html, \*+html и т.д.).
* Плохо – не проверять tabindex в формах.
* Плохо — писать стили, не думая о логике размещения элементов. Например, если элемент всегда находится сверху, у него должен быть большой z-index, нельзя надеяться на то, что сейчас всё нормально смотрится – стили должны быть железобетонными. Если элемент всегда должен находиться на каком-то месте, вне зависимости от окружающих его элементов — это position: absolute; а не float и т.д.  
  Блоки, не зависящие друг от друга, не должны быть внутри одного блока (например, телефон компании и поиск по сайту). Блоки, не зависящие по расположению друг от друга, должны быть position absolute, а не float.
* Очень плохо — презентационные классы (right, red).
* Также вёрстка не должна содержать пустых блоков (для презентационных целей)
* Плохо – когда нет базовых стилей у стандартных элементов. Т.е. просто h1, h2, ul, table, etc. без классов должны смотреться красиво и органично.
* Плохо – когда нет постепенного уточнения стилей, а стиль выписывается для каждого элемента отдельно, а за его пределами — внешний вид может быть кардинально другой. Речь о ситуации, когда, например, текст, написанный без абзацев, имеет вообще не тот стиль что у всех элементов в блоке. Надо вести стили снизу вверх, постепенно уточняя их.
* Ещё хуже – чересчур детализированные глобальные стили. Например: a {font: italic 10px Tahoma;} Потом приходится переопределять стиль ссылок для каждого блока.
* Размеры и позиционирование элемента должны указываться в одних единицах измерения. Для текстовых блоков это в 90% случаев — em. Line-height — как правило, коэффициентом (1/1.2/1.4… т.е. без указания единицы измерения — это цифра, на которую умножается font-size и получится межстрочный интервал). Т.е. задали font-size/height в em — значит задаём и margin/padding тоже в em. Классический пример: список dl-dt-dd, где dt и dd ставятся друг напротив друга с помощью подтягивания dd отрицательным margin вверх. Или — выделили padding'ом место под position: absolute какого-то текстового блока. У текстовых элементов (абзацей, ячеек таблиц) задаём padding и height в em (чтоб корректно увеличивать размер шрифта).
* Плохо вешать стили на стандартные тэги, без классов. Т.е., допустим, пишем что-то типа h2 a span {какая-то крепкая штука, типа картинки с графикой, что закрывает текст}, а потом клиент в визиге внезапно вбивает такое сочетание! Все стили элементов внутри #content обязательно должны навешиваться на элементы с классом.
* Плохо — напрямую задавать визуальное отображение элементов через js: $(’elementid’).css(styleName,’something’). Это должно делаться через установку/смену классов.

Примеры хорошего тона:

* [**БЭМ**](http://bem.github.com/bem-method/pages/beginning/beginning.ru.html)! Важно понимать, что это методология, а не инструменты. Для большинства сайтов достаточно вёрстки в АНБ-нотации, без использования библиотеки блоков и bem-tools. Также почитайте про [MCSS](https://github.com/operatino/MCSS) и [SMACSS](http://smacss.com/). Нужно обязательно уходить от каскада, а БЭМ — это один из отличных вариантов решения.
* Хорошо — структурировать код в блоки, описывающие логику документа. Т.е. создавать div даже там, где он для презентационных целей не нужен. И наоборот — стараться не ставить лишний div там, где структура этого не требует, а это нужно лишь для визуальных эффектов.
* [**HTML5 Boilerplate**](http://html5boilerplate.com/) — замечательный стартовый шаблон от «отцов». Используйте и присоединяйтесь к разработке, добавляйте свои велосипеды туда!   
  Раньше у нас был свой самописный framework-стартовый html, теперь используем Boilerplate как основу, а в него уже добавляем старые наработки.
* Используйте наработанные решения, чужие модули, jQuery-плагины, не изобретайте велосипедов, а если изобретаете — поддерживайте их, ведите библиотеку кода (после каждого нового проекта добавляйте туда код, обновляйте старый).
* Для текстовых блоков, что редактируются в админке, должна обеспечиваться атомарность текстовых стилей. Т.е. есть у нас страничка с каким-то текстовым блоком, примерно такой структуры:

.content-text h1

.content-text .entry

.content-text .entry .somenamedblock

То .somenamedblock должен получать текстовые стили непосредственно — .somenamedblock {font: ...; color: ...;}, т.к. иначе в визиге админки мы не сможем их застайлить.

* одинаковый html-код для блоков на морде и внутряках, с идентичным контентом, но разным визуальным представлением данных.
* Помещайте медиавыражения настолько близко к соответствующим наборам правил, насколько это возможно.

Организация кода

Разделите CSS-код на несколько файлов со своими составляющими:

- Сброс стилей

- Шрифты

- Базовые стили

- Общие стили:

header

footer

- Блоки

- Компоненты

Ориентируйтесь на блоки и компоненты, а не на страницы, которые могут быть переработаны, а блоки и компоненты могут перемещаться с одной страницы на другую.

Порядок объявления

Объявления логически связанных свойств должны быть сгруппированы в следующем порядке:

- Позиционирование - может удалить элемент из нормального потока документа и переопределить блочную модель связанных стилей.

- Расположение - задаются внешние и внутренние отступы.

- Блочная модель - диктует размеры

- Типографика - шрифты, стилизация текста

- Отображение - цвет, фон и т.п.

- Прочее - трансформации, поведение и т.п.

Пример:

.declaration-order {

/\* Позиционирование \*/

position: absolute;

z-index: 100;

top: 0;

right: 0;

left: 0;

bottom: 0;

/\* Расположение \*/

margin-top: 20px;

margin-left: auto;

margin-right: auto;

padding: 8px 16px;

/\* Блочная модель \*/

display: block;

float: left;

width: 100px;

height: 100px;

/\* Типографика \*/

font-family: 'Helvetica Neue', sans-serif;

font-size: 14px;

line-height: 24px;

text-align: center;

/\* Отображение \*/

color: #333;

background-color: #f5f5f5;

border: 1px solid #e5e5e5;

border-radius: 3px;

opacity: 1;

/\* Прочее \*/

transition: .25s ease-out;

transform: scale(.75);

}

"position",

"z-index",

"top",

"right",

"bottom",

"left",

"margin",

"margin-top",

"margin-right",

"margin-bottom",

"margin-left",

"padding",

"padding-top",

"padding-right",

"padding-bottom",

"padding-left",

"display",

"visibility",

"float",

"clear",

"overflow",

"overflow-x",

"overflow-y",

"-ms-overflow-x",

"-ms-overflow-y",

"clip",

"zoom",

"flex-direction",

"flex-order",

"flex-pack",

"flex-align",

"-webkit-box-sizing",

"-moz-box-sizing",

"box-sizing",

"width",

"min-width",

"max-width",

"height",

"min-height",

"max-height",

"table-layout",

"empty-cells",

"caption-side",

"border-spacing",

"border-collapse",

"list-style",

"list-style-position",

"list-style-type",

"list-style-image",

"font",

"font-family",

"font-size",

"font-weight",

"font-style",

"font-variant",

"font-size-adjust",

"font-stretch",

"font-effect",

"font-emphasize",

"font-emphasize-position",

"font-emphasize-style",

"font-smooth",

"-moz-osx-font-smoothing",

"-webkit-font-smoothing",

"line-height",

"content",

"quotes",

"counter-reset",

"counter-increment",

"resize",

"cursor",

"-webkit-user-select",

"-moz-user-select",

"-ms-user-select",

"user-select",

"nav-index",

"nav-up",

"nav-right",

"nav-down",

"nav-left",

"text-align",

"-webkit-text-align-last",

"-moz-text-align-last",

"-ms-text-align-last",

"text-align-last",

"vertical-align",

"white-space",

"text-decoration",

"text-emphasis",

"text-emphasis-color",

"text-emphasis-style",

"text-emphasis-position",

"text-indent",

"-ms-text-justify",

"text-justify",

"letter-spacing",

"word-spacing",

"-ms-writing-mode",

"text-outline",

"text-transform",

"text-wrap",

"text-overflow",

"-ms-text-overflow",

"text-overflow-ellipsis",

"text-overflow-mode",

"-ms-word-wrap",

"word-wrap",

"word-break",

"-ms-word-break",

"-moz-tab-size",

"-o-tab-size",

"tab-size",

"-webkit-hyphens",

"-moz-hyphens",

"-ms-text-size-adjust",

"-webkit-text-size-adjust",

"hyphens",

"pointer-events",

"-webkit-appearance",

"color",

"filter:progid:DXImageTransform.Microsoft.Alpha(Opacity",

"-ms-filter:\\'progid:DXImageTransform.Microsoft.Alpha",

"-ms-interpolation-mode",

"-webkit-focus-ring-color",

"-webkit-tap-highlight-color",

"background",

"filter:progid:DXImageTransform.Microsoft.AlphaImageLoader",

"background-color",

"background-image",

"background-repeat",

"background-attachment",

"background-position",

"background-position-x",

"-ms-background-position-x",

"background-position-y",

"-ms-background-position-y",

"-webkit-background-clip",

"-moz-background-clip",

"background-clip",

"background-origin",

"-webkit-background-size",

"-moz-background-size",

"-o-background-size",

"background-size",

"border",

"border-width",

"border-style",

"border-color",

"border-top",

"border-top-width",

"border-top-style",

"border-top-color",

"border-right",

"border-right-width",

"border-right-style",

"border-right-color",

"border-bottom",

"border-bottom-width",

"border-bottom-style",

"border-bottom-color",

"border-left",

"border-left-width",

"border-left-style",

"border-left-color",

"-webkit-border-radius",

"-moz-border-radius",

"border-radius",

"-webkit-border-top-left-radius",

"-moz-border-radius-topleft",

"border-top-left-radius",

"-webkit-border-top-right-radius",

"-moz-border-radius-topright",

"border-top-right-radius",

"-webkit-border-bottom-right-radius",

"-moz-border-radius-bottomright",

"border-bottom-right-radius",

"-webkit-border-bottom-left-radius",

"-moz-border-radius-bottomleft",

"border-bottom-left-radius",

"-webkit-border-image",

"-moz-border-image",

"-o-border-image",

"border-image",

"-webkit-border-image-source",

"-moz-border-image-source",

"-o-border-image-source",

"border-image-source",

"-webkit-border-image-slice",

"-moz-border-image-slice",

"-o-border-image-slice",

"border-image-slice",

"-webkit-border-image-width",

"-moz-border-image-width",

"-o-border-image-width",

"border-image-width",

"-webkit-border-image-outset",

"-moz-border-image-outset",

"-o-border-image-outset",

"border-image-outset",

"-webkit-border-image-repeat",

"-moz-border-image-repeat",

"-o-border-image-repeat",

"border-image-repeat",

"outline",

"outline-width",

"outline-style",

"outline-color",

"outline-offset",

"box-decoration-break",

"-webkit-box-shadow",

"-moz-box-shadow",

"box-shadow",

"opacity",

"filter:progid:DXImageTransform.Microsoft.gradient",

"-ms-filter:\\'progid:DXImageTransform.Microsoft.gradient",

"text-shadow",

"-webkit-transition",

"-moz-transition",

"-ms-transition",

"-o-transition",

"transition",

"-webkit-transition-delay",

"-moz-transition-delay",

"-ms-transition-delay",

"-o-transition-delay",

"transition-delay",

"-webkit-transition-timing-function",

"-moz-transition-timing-function",

"-ms-transition-timing-function",

"-o-transition-timing-function",

"transition-timing-function",

"-webkit-transition-duration",

"-moz-transition-duration",

"-ms-transition-duration",

"-o-transition-duration",

"transition-duration",

"-webkit-transition-property",

"-moz-transition-property",

"-ms-transition-property",

"-o-transition-property",

"transition-property",

"-webkit-transform",

"-moz-transform",

"-ms-transform",

"-o-transform",

"transform",

"-webkit-transform-origin",

"-moz-transform-origin",

"-ms-transform-origin",

"-o-transform-origin",

"transform-origin",

"-webkit-animation",

"-moz-animation",

"-ms-animation",

"-o-animation",

"animation",

"-webkit-animation-name",

"-moz-animation-name",

"-ms-animation-name",

"-o-animation-name",

"animation-name",

"-webkit-animation-duration",

"-moz-animation-duration",

"-ms-animation-duration",

"-o-animation-duration",

"animation-duration",

"-webkit-animation-play-state",

"-moz-animation-play-state",

"-ms-animation-play-state",

"-o-animation-play-state",

"animation-play-state",

"-webkit-animation-timing-function",

"-moz-animation-timing-function",

"-ms-animation-timing-function",

"-o-animation-timing-function",

"animation-timing-function",

"-webkit-animation-delay",

"-moz-animation-delay",

"-ms-animation-delay",

"-o-animation-delay",

"animation-delay",

"-webkit-animation-iteration-count",

"-moz-animation-iteration-count",

"-ms-animation-iteration-count",

"-o-animation-iteration-count",

"animation-iteration-count",

"-webkit-animation-direction",

"-moz-animation-direction",

"-ms-animation-direction",

"-o-animation-direction",

"animation-direction",

"will-change"

Для всех элементов, которые могут содержать текст различной длины, который должен быть вписан в одну строку (например, для кнопок или заголовков, если в дизайне не предусмотрено, что они могут занимать больше одной строки), обязательно задавайте CSS свойство **white-space:nowrap**.

Если вы делаете обработку событий при нажатии на ссылки, следите за тем, чтобы обработчики событий возвращали false, или же используйте href='javascript:void(0)' вместо популярного href='#', чтобы страница не скроллилась вверх.

**Анимация**

Во имя всего святого, не пытайтесь сделать всё ваше приложение анимированным. Бо́льшая часть CSS анимаций задействует перерисовку всего макета; лучше ограничьте себя настолько, насколько возможно, в использовании transform, opacity, transition.

Избегайте ленивых вычислений параметра transition, если вы используете его, то используйте отдельные атрибуты (например, transition: opacity 250ms ease-in вместо transition: all 250ms ease-in).