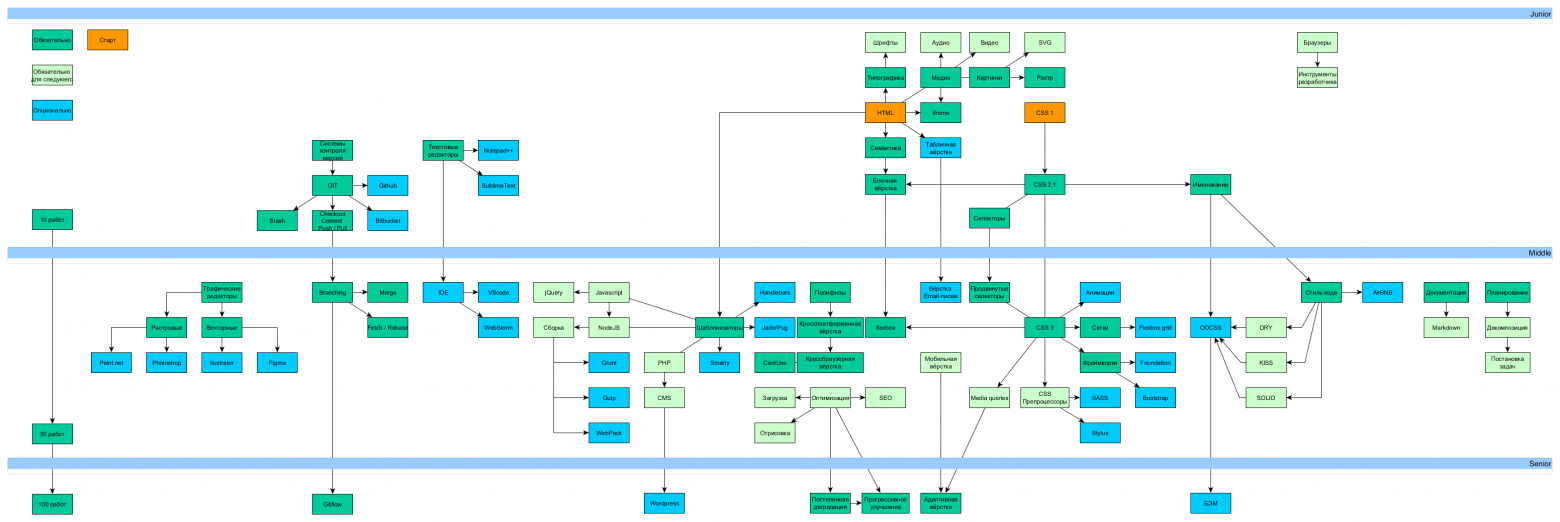
Путь верстальщика: с нуля до сеньора

[Разработка веб-сайтов \*](https://habr.com/ru/hub/webdev/)[CSS \*](https://habr.com/ru/hub/css/)[Совершенный код \*](https://habr.com/ru/hub/complete_code/)[HTML \*](https://habr.com/ru/hub/html5/)[Учебный процесс в IT](https://habr.com/ru/hub/study/)

Здравствуйте, меня зовут [Александр Зеленин](http://zelenin1.moikrug.ru/), и я веб-разработчик.  
Многократно я слышал мнение, что верстка — удел начинающих frontend’еров. Хотя фактически это важнейшая часть любого (почти) веб-проекта. Это то, что пользователи видят в первую очередь. На текущий момент качественная вёрстка (особенно проектирование блоков) в крупном проекте требует большого количества различных навыков.

В данной статье представляю схему развития верстальщика

[](https://habrastorage.org/webt/lg/d9/qe/lgd9qe8etwf96cpiwsbo7j2moa8.png)  
[большая по клику]  
Само собой, это не всеобъемлющая и единственно верная схема. Есть ещё целая гора связанных навыков, релевантных технологий и так далее. Градация является субъективной.

Хочу сразу добавить, что конкретных ссылок на учебные материалы в статье не будет. Буду рад дополнениям в комментариях.

Чего мы ожидаем?

В первую очередь необходимо понять, чего мы ожидаем от человека на каждом из уровней. Ваши ожидания могут отличаться.  
**Junior** — знает основные теги, может подправить контент на готовом html сайте, умеет грамотно форматировать текст, без проблем вставит встраиваемый элемент (ролик с ютуба, яндекс карту), ведёт работу через систему контроля версий только для себя (master, 1 контрибьютор). Может сверстать что-то несложное с нуля, и оно даже не поедет (лично у него). В целом не является самостоятельной единицей и требует руководства свыше.  
**Middle** — более-менее автономная единица (либо полностью автономная для малых и средних проектов). Может хорошо сверстать сайт средних размеров, и он будет корректно отображаться во всех актуальных версиях браузеров. Понимает, как работают шаблонизаторы, и может их использовать (при условии предоставления остального кода). Умеет планировать и документировать свою работу и оценивать сроки. Понимает важность поддержания стиля кода. Понимает, зачем есть грид-системы и css фреймворки. Умеет брать всю необходимую информацию из макетов от дизайнера. Может взаимодействовать с небольшой командой, создавать ветки и пулл-реквесты.  
**Senior** — может спроектировать систему блоков для крупного проекта. Знает, как избежать повторов и проблемных мест при использовании его кода другими разработчиками. Умеет декомпозировать сложные задачи и грамотно формулировать задачи. Умеет применять минимум одну методологию разработки (например, БЭМ). Может сделать так, чтобы проект открывался максимально быстро. Хорошо понимает несколько популярных шаблонизаторов. Способен написать сборщики и автоматизировать процесс, с этим связанный. Может проводить код ревью и курировать других верстальщиков.  
Рассмотрим теперь чуть конкретнее, что включает в себя каждый из блоков на каждом уровне.

Junior

**HTML** — знание основных тегов, аттрибутов. Понимание, как его писать в принципе.  
**Типографика** — умение отформатировать текст. Текст — основа практически любого проекта. Вставка неразрывных пробелов где надо, выделение жирным, курсивом, abbr и так далее. Можно использовать типограф или схожий сервис, но уметь понимать результат.  
**Семантика** — понимание, что для определённых задач есть определённые теги. Уметь выбрать нужный тег.  
**Медиа** — какие виды медиа можно встраивать в страницу.  
**Iframe** — встраивание сторонних виджетов (видео, аудио, карты и т.п.).  
**Аудио, Видео** — можно отложить изучение, т.к. частично решаются с помощью iframe. Понимать, какие форматы умеет воспроизводить браузер, как оформить плеер и так далее.  
**Картинки** — какие графические форматы и в каком виде воспринимает браузер. Плюсы и минусы использование тех или иных форматов.  
**Растр** — jpg, png, gif. Понимать различие форматов и уметь применять что и где надо.  
**SVG** — можно отложить изучение, т.к. применяется реже, чем хотелось бы. Понимать плюсы, минусы, ограничения и т.п.  
**Шрифты** — можно отложить изучение. На самом деле довольно сложная тема, и в общем случае я бы советовал начинающим использовать системные шрифты. Уметь подгрузить шрифты, оптимизировать отображение, минимизировать лаг отрисовки с кастомными шрифтами.  
**Табличная вёрстка** — опционально. Для желающих верстать качественные email-рассылки в будущем.  
**CSS 1** — шрифты, цвета, выравнивание, размеры.  
**CSS 2.1** — управление поведением блоков, позиционирование, полноценное оформление.  
**Селекторы** — простые селекторы на тег, класс, вложенный элемент. Простые псевдоселекторы типа :hover.  
**Именование** — как называть классы, чтобы не было мучительно больно.  
**Блочная вёрстка** — разбить картинку на осознанные блоки, воплотить блоки в HTML, оформить с CSS.  
**Браузеры** — можно отложить изучение. Какие браузеры существуют, в чём их различие.  
**Инструменты разработчика** — можно отложить изучение. Использовать инструменты браузеров, чтобы разбираться в проблемах отображения.  
**Текстовые редакторы** — какие текстовые редакторы для разработчиков есть и зачем. SublimeText и Notepad++ приведены для примера, как хорошо мне знакомые. Уметь настраивать в них базовые вещи, такие как отступы, перенос строки и так далее.  
**Системы контроля версий** — я лично считаю умение их использовать хотя бы индивидуально, лично для себя, очень важным. Понимать, для чего эти системы созданы и какие бывают.  
**Git** — понимать в общих чертах задачу и принципы самой популярной системы контроля версий.  
**Github / bitbucket** — уметь использовать одну из популярных площадок для git.  
**Checkout / commit / push / pull** — базовые операции для личного использования.  
**Stash** — для временного сохранения ненужных в данный момент данных.  
**10 работ** — сделать минимум 10 работ в различном дизайне. Можно тестовых, это не важно. Важно, чтобы полноценных в рамках текущих знаний.

Middle

**CSS 3** — градации, тени, сглаживание, фильтры, трансформации.  
**Продвинутые селекторы** — элементы, следующие за опделённым (+), определённый по счёту (nth-child), shadow-dom, before/after и так далее.  
**Анимации** — опционально. transition и animation. Плавные переходы, анимации. Понимать ограничения и минусы.  
**Сетки** — зачем существуют, как выстраивать, какие есть готовые решения. Для примера можно посмотреть Flexbox grid или любой другой, который найдёте.  
**Фреймворки (CSS)** — зачем нужны, как применять. Желательно научиться хорошо использовать минимум один. Очень пригождается для прототипирования. Значительно повышает качество проекта при отсутствии бюджета на дизайн (не уникально, но юзабельно).  
**CSS препроцессоры** — можно отложить изучение. Оптимизация работы, более красивый и читабельный код. Переменные, миксины и т.п. Поработать с одним или несколькими популярными препроцессорами типа SASS, LESS, Stylus.  
**Media queries** — можно отложить изучение. Отображение нужных стилей в зависимости от условий (устройство, размер экрана, плотность пикселей, версия для печати и т.п.).  
**Стиль кода** — понять, для чего есть соглашения по стилям, изучить и начать применять любой (рекомендую от AirBNB).  
**DRY / KISS / SOLID** — можно отложить изучение. Понять важные принципы разработки, ощутимо упрощающие дальнейшее сопровождение проекта.  
**OOCSS** — опционально. Понять, что есть объектно-ориентированный css и для чего он. В том или ином виде используется во многих проектах (хоть и без понимания, что это он). В идеале, научиться проектировать. Может отлично зайти для крупных проектов.  
**Документация** — понять, что и как документировать. Документировать. Можно отложить, но в будущем обязательно изучить разметку markdown.  
**Планирование** — научиться оценивать сроки по картинке и определять последовательность работ.  
**Декомпозиция** — можно отложить изучение. Научиться разбивать задачу на подзадачи. Это сложнее, чем кажется :-)  
**Постановка задач** — можно отложить изучение. Научиться чётко описывать задачи текстом так, чтобы другие разработчики, в том числе с меньшей квалификацией, достаточно однозначно понимали, что необходимо сделать для их выполнения.  
**Flexbox** — понимание модели, умение применять полноценно.  
**Вёрстка писем** — опционально. В целом навык не лишний. Понимать особенности почтовых систем, сохранить хороший внешний вид и не упасть в спам (если это не спам).  
**Полифилы** — разобраться, как использовать самые актуальные возможности разработки, сохраняя обратную совместимость. Понять минусы и плюсы от такого подхода.  
**Кроссплатформенная вёрстка** — понимать, что нужно делать, чтобы проект хорошо выглядел не только под Windows, Linux и Mac, но и под SmartTV или PS.  
**Кроссбраузерная вёрстка** — понимать разницу рендера браузеров и заставлять их показывать одинаково. Сайт CanIUse очень помогает в этом.  
**Мобильная вёрстка** — можно отложить изучение. Понимать, какие ограничения несут в себе мобильные платформы. Использовать лимитированное пространство грамотно.  
**Оптимизация** — можно отложить изучение. Разобраться в «цене» тех или иных приёмов. Понять, из каких фаз состоит отображение сайта пользователю.  
**Загрузка** — можно отложить изучение. Оптимизации, связанные с размером, кешем, сжатием, объединением ресурсов и т.п.  
**Отрисовка** — можно отложить изучение. Оптимизации, связанные со скоростью рендера после загрузки.  
**SEO** — можно отложить изучение. Хотя бы базовое понимание работы поисковых систем. Умение «помочь» поисковой системе разобраться, куда смотреть и что важно.  
**Шаблонизаторы** — разобраться, каким образом можно переиспользовать код, группировать элементы и компоновать страницы. Очень желательно изучить как серверный рендеринг, так и клиентский. Сюда же входят шаблоны на «чистом» языке (типа простых PHP вставок). Работа «до» (dataflow) шаблонизатора нас не интересует.  
**PHP** — можно отложить изучение. Разобраться в базовом синтаксисе и уметь внести мелкие правки, связанные с оформлением страницы.  
**CMS** — можно отложить изучение. Узнать, какие CMS существуют и для чего они были созданы. Научиться писать шаблоны хотя бы для одной (рекомендую Wordpress).  
**Javascript** — можно отложить изучение. Изучить базовый синтаксис, понять, как вешать простейшие обработчики и проводить простую работу с DOM.  
**jQuery** — можно отложить изучение. Научиться экономить кучу времени для решения довольно типовых задач с использованием плагинов для самой популярной js библиотеки (после vanilla.js, конечно).  
**NodeJS** — можно отложить изучение. Разобраться, как запустить простейший сервер, раздавать статику и рендерить на стороне сервера. Можно использовать Express или любой другой фреймворк.  
**Сборка** — опционально. Научиться собирать проект из кучи CSS / HTML файлов в то, что нужно. Рекомендую ознакомиться хотя бы с grunt и gulp, как представителями «разных» лагерей.  
**IDE** — опционально. Изучить, зачем нужны IDE и как их использовать. Перейти на использование какого-либо IDE для экономии времени. Важно: изучение IDE сравнимо с полноценным изучением языка программирования, и вкладывание большого количества времени в это может не оправдаться. Лично я пользуюсь текстовыми редакторами (и только для ооочень крупных проектов включаю IDE).  
**Branching** — научиться управлять ветками в git.  
**Merge** — научиться сливать ветки с разрешением конфликтов.  
**Fetch / Rebase** — разобраться, зачем они, когда их применять, и начать применять по необходимости.  
**Графические редакторы** — разобраться, какие бывают, для чего. Чем векторные отличаются от растровых. Важно понимать редактор хотя бы на уровне «чтения» макета от дизайнера. Выбрать правильный шрифт, размер, цвет и т.п. Чтобы не на глаз, а точно. Рекомендую поработать минимум с 1 растровым (Photoshop) и 1 векторным (Figma).  
**50 работ** — к концу этапа у вас порядка 50 различных работ, демонстрирующих навыки из изученных областей.

Senior

Хоть на схеме и кажется маленькой эта группа, но фактически она самая большая. Т.к. к этому моменту необходимо изучить всё то, что откладывалось.  
**Адаптивная / Отзывчивая вёрстка** — разобраться в высшей ступени и скомбинировать все полученные до этого знания. Проект должен выглядеть хорошо везде и на всём (в рамках разумного).  
**Постепенная деградация / Прогрессивное улучшение** — понять, что это и зачем. Использовать.  
**Gitflow** — уметь объяснить другим разработчикам, как создавать ветки, куда их вливать, как, уметь провести код ревью (вёрстки, само собой, не кода).  
**БЭМ** — опционально. Разобраться в методологии, позволяющей создавать безлимитно крупные проекты так, чтобы при минимальной синхронизации различные команды могли использовать блоки друг друга. Есть и другие методологии, дающие не худший результат. К этому моменту о них вы так или иначе будете знать и сможете, при желании, изучить.  
**100 работ** — иметь суммарно сотню работ, демонстрирующих различные полученные навыки. На самом деле, это всё условно. Можно иметь одну работу (состоящую из различных частей) в портфолио, которая уже покажет, что вам ничего не страшно.

Заключение

Фактически многие из этих тем небольшие. Однако держать кучу различных факторов в голове очень сложно. За 15 лет я общался менее чем с десятью верстальщиками (да и вообще программистами), которых отнёс бы к категории Senior (верстальщиков, программистов хороших было сильно больше).  
Навскидку – на полноценное изучение и практику только вёрстки с нуля вполне можно затратить год (и более, если уходить в проектирование компонентов).

Надеюсь, тем, кто находится в начале пути, схема подскажет, куда можно ещё двигаться, и что не обязательно сразу (а можно и вообще не двигать в него) бежать в программирование.

Также планирую создать полноценный бесплатный курс с подробным рассмотрением каждой из тем-прямоугольников со схемы. Буду рад предложением, отзывам, пожеланиям и так далее.  
Курс, вероятно, будет в формате видеоуроков с полным текстовым и графическим дублированием. Если дело пойдёт, приложу в будущем ссылки в комментарии.

Skillbox Layout