FRONTEND: ВЕРСТКА, ИНСТРУМЕНТЫ И ФРЕЙМВОРКИ

Оглавление

- Верстка (HTML + CSS)
- Другие инструменты для разработки фронтенда
- Основные фреймворки для JS и CSS
- Инструменты для сборки и автоматизации фронтенда
  - Преобразование современного кода JavaScript
  - Сборка проекта на JavaScript
  - Таск-менеджеры

ВЕРСТКА (HTML + CSS)

Фактически, страница сайта — это такой же документ, как, например, документ Microsoft Word, только другого формата и приспособленный для того, чтобы его просматривать в браузере. Основа этого документа — это HTML, язык разметки гипертекста. Это не язык программирования, а именно разметка документа, которая говорит, что должно быть на странице, и наполняет страницу контентом. В начале 90-х HTML был единственной веб-технологией, и с его помощью создавались макеты (что и где должно располагаться на странице) и дизайн (как это должно выглядеть) сайта.

Однако в современном вебе задачу дизайна выполняет CSS, также CSS отвечает за расположение элементов. Задача HTML сейчас заключается в том, чтобы создать логическую структуру страницы, наполнить ее контентом и указать браузеру (и другим программам), каков тип данного контента (например: является ли картинка на сайте фотографией, пунктом меню, логотипом сайта или фоном).

HTML представляет из себя так называемые теги, которые бывают открывающими и закрывающими. Весь код HTML-страницы заключается между

тегов <html>/<html>. Этот тег говорит браузеру, что это веб-страница, и ее нужно отображать как сайт. Также существуют теги заголовка, то есть такие теги, которые содержат разную служебную информацию для браузера, например — название вкладки.

Большинство тегов что-то делают, результат их работы визуально виден пользователю. Например, тег помещает на страницу блок текста, или заголовки, или создаёт форму.

HTML — это иерархическая структура документов. Это значит, что все служебные теги будут находиться в теге <head> </head>, весь контент — внутри тега <body> </body>, и так далее. Теги как бы вкладываются один в другой, поэтому у них есть тег верхнего уровня — родитель, и теги, вложенные в него — потомки.

На данный момент (декабрь 2019) актуальная версия HTML — 5.2. Фактически, сейчас HTML стал достаточно самостоятельным. Например, можно осуществлять перетаскивание элементов на странице, воспроизводить музыку и видео, рисовать различные фигуры. Раньше все это реализовывалось с помощью других технологий, что приводило к разнообразным проблемам. Теперь же все отображается одинаково во всех браузерах и избавляет разработчиков от лишней головной боли.

Как уже было сказано выше, HTML используется для того, чтобы указать, что должно быть на странице, а вот для того, чтобы указать, как именно это “что” должно выглядеть и где располагаться, используется CSS.

CSS также не является языком программирования, это язык описания стилей документов. Конечно, внешний вид страницы можно изменять при помощи нативного HTML, однако CSS обладает большими возможностями, а также это более правильно с точки зрения архитектуры веб-страницы (с ним просто удобнее работать и поддерживать продукт). CSS использует привязку к тегам HTML. Например, можно указать, что для всех тегов <h1> </h1> цвет текста должен быть красным, а для всех тегов <h2> </h2> — зеленым. Последняя версия CSS — 3.

Связка HTML + CSS образует так называемую верстку страницы. Верстка — это что, как и где и где расположено на странице (по аналогии с версткой журналов и книг). Людей, которые занимаются версткой страниц, называют верстальщиками. Они не являются программистами, потому что не работают с языками программирования. Зачастую верстальщики переводят макет, созданный дизайнером (например, в Photoshop, в формате .psd) в веб-форму (HTML+CSS), хотя могут и делать сайты с нуля (выполнять роль дизайнера, самостоятельно

делая макет). На сегодняшний день верстальщики, которые не работают с JavaScript, встречаются редко и только в небольших агентствах по созданию простеньких сайтов. Более того, грань между вёрсткой и фронтенд-разработкой (читай, JS-разработкой) довольно нечеткая. Верстальщикам надо хотя бы базово знать JS, а фронтендерам — уметь верстать. Более того, обычно этим занимается один и тот же человек. Обычно фронты начинают с большого количества вёрстки в своей работе и приходят к бОльшему количеству разработки по мере профессионального роста. Периодически среди фронтов встречается нежелание заниматься вёрсткой.

Всего есть несколько видов верстки сайтов. Самая большая проблема, которая есть у верстальщиков, заключается вот в чем: сделать так, чтобы сайт одинаково хорошо отображался в разных браузерах (кроссбраузерная верстка) и на разных устройствах (адаптивная верстка). Последнее является отдельной головной болью, из-за разных размеров экрана на разных устройствах (на мобильных телефонах — 320 , а на экране компьютера — обычно 1980 px). В результате, если мы делаем сайт без учета этого факта (это называется “фиксированная верстка”), то на мобильных телефонах он будет отображаться не полностью, так как попросту не влезет в экран. Поэтому придумали так называемую “резиновую верстку” — когда все элементы на странице изменяются пропорционально размеру экрана. На маленьком экране они пропорционально уменьшаются, на большом — увеличиваются.

Но здесь появляется еще одна проблема: на маленьком экране все становится очень мелким и неудобным для чтения. Поэтому были придуманы принципы адаптивной верстки (или адаптивной разметки). Под адаптивной версткой подразумевается, что у сайта будет разный внешний вид на мобильном устройстве и на компьютерном мониторе. В CSS есть специальные инструменты, с помощью которых можно определять размер экрана, а под него внешний вид сайта будет изменяться соответственно.

Чтобы это реализовать, нужно разделить всю страницу на определенные элементы — блоки. Например: блок меню, блок текста, блок рекламы и так далее. Соответственно, такая верстка называется блочной. Адаптивная верстка реализуется посредством блочной верстки. В спецификации CSS есть две современные реализации блочной верстки (вам, рекрутерам, нужно просто знать термины): flexbox и grid layout.

Иногда также можно встретить понятие так называемой семантической верстки. Дело в том, что в HTML есть возможность напрямую указывать, что за контент будет на странице. Например, есть тег <blockquote> </blockquote>, который говорит браузеру, что это цитата, взятая из внешних источников. Семантические теги помогают размечать страницу для различных вспомогательных устройств: например, это может быть ридер с функцией прочтения текста голосом (используется слепыми или имеющими плохое зрение). Если на странице есть тег , то ридер сделает акцент на этом теге и прочтет обрамленный этим тегом абзац с выделением интонации. А вот если этот же абзац будет обрамлен тегом , то интонационного выделения не произойдет, хотя визуально оба тега дают один и тот же результат: выделяют **полужирным** шрифтом.

ДРУГИЕ ИНСТРУМЕНТЫ ДЛЯ РАЗРАБОТКИ ФРОНТЕНДА

Начнем с так называемых препроцессоров.

Для начала небольшое отступление. Любая программа работает по следующему принципу: она принимает какие-либо данные, обрабатывает их и выдает результат. Обычно говорят так: программа получила что-то на “входе”, обработала это и передала на “выход”.

Например, программист пишет код на CSS, который передается на вход интерпретатору CSS. Интерпретатор обрабатывает этот код и передает результат на вход браузеру.

Так вот, суть препроцессора в том, что на вход он получает упрощенный и видоизмененный код CSS, который он обрабатывает, и на выходе получается обычный нативный CSS, который уже обрабатывается интерпретатором.

Иными словами, верстальщик пишет код на синтаксисе препроцессора, а уже сам препроцессор преобразует этот код в обычный CSS, чтобы его мог обработать браузер.

Для чего это нужно? Во-первых, препроцессор добавляет “синтаксический сахар”: немного изменяет синтаксис кода, что упрощает его написание. Например,

программисту нужно написать десять строк кода, а с синтаксическим сахаром будет достаточно пяти.

Второе применение — это добавление полноценных переменных и функций, которых не хватает в нативном CSS. Опять-таки, эти переменные и функции существуют только лишь для для удобства разработчика, браузер принимает обычный код CSS. Они помогают автоматизировать процесс верстки множества файлов CSS: в CSS-файлах обычно есть много мест, где применяется однотипный код. Например: {color: red;}. Вместо такой конструкции мы можем создать переменную $red и использовать её. И тогда, если мы поменяем цвет на голубой, нам нужно будет указать это изменение только в одном месте, а не искать все вхождения во всех файлах. Самые популярные препроцессоры: Sass, Less, Stylus.

Наряду с препроцессорами существуют так называемые постпроцессоры. Как понятно из названия, постпроцессор принимает на вход код CSS, преобразовывает его и передает уже непосредственно интерпретатору. От препроцессора отличается тем, что препроцессор обрабатывает код, который верстальщик написал на синтаксисе этого препроцессора и переводит в обычный код CSS, а постпроцессор обрабатывает обычный код CSS, дополняя его.

Соответственно, постпроцессор логичнее использовать, чтобы что-то автоматически добавлять в код, если этого не хватает. Например, для обеспечения кроссбраузерности, у некоторых CSS свойств нужно писать специальные ключевые слова под каждый браузер, чтобы они (свойства) корректно работали. И есть специальный постпроцессор, который называется Autoprefixer, который автоматически добавляет недостающие префиксы. Собственно, это самый популярный постпроцессор, который стоит знать.

ОСНОВНЫЕ ФРЕЙМВОРКИ ДЛЯ JS И CSS

Существует несколько очень популярных технологий для разработки фронтенда, которые чаще всего встречаются у разработчиков:

Во-первых, это jQuery, одна из самых популярных библиотек для JS. Фактически, в течение многих лет jQuery была основной библиотекой для JS, ещё до появления большого количества фреймворков. Многие вещи при помощи этой библиотеки реализовываются очень просто, например, когда вы на сайте

нажимаете на ссылку, а она раскрывается в текст, а также работа с AJAX. Последняя версия — 3.2.1.

Во-вторых, Angular. Это современный фреймворк от Google, который использует TypeScript для написания кода. Angular — это один из основных фреймворков для построения одностраничных приложений (то есть SPA).

По состоянию на сентябрь 2018 уже появился Angular 6. Обычно между опытом на Angular 2-6 не делают различия, но Angular 1, он же Angular.js, отличается от последующих версий довольно сильно.. С большой вероятностью, если в резюме у кандидата указан Angular.js, то имеется в виду первый Angular. Несмотря на то, что Angular удерживает первую в мире позицию по используемости, зачастую нужны люди с React, затем с Vue, а Angular уже на третьем месте.

В-третьих, React.js. Технически, это библиотека, но очень часто его называют фреймворком. Это не совсем верно, так как React не предоставляет полного функционала фреймворка. React — это работа с той частью вашего сайта, который видит пользователь. Написать полноценное SPA на React невозможно, нужно использовать какие-то дополнительные библиотеки или фреймворки (в отличие от самодостаточного Angular).

Так, для работы с AJAX React использует jQuery, а для WebSocket — Socket.io. В комбинации с React мы часто встречаем Redux. По сути, это библиотека, которая специальным образом структурирует взаимодействие между частями React-приложения, а также это дополнительный инструмент для разработчика, с помощью которого можно повысить качество и производительность React-приложения. Аналоги: Vuex, MobX.

React не зря называется так. Фактически, React — это реализация принципов реактивного программирования. Чтобы понять, что это такое, рассмотрим пример. Мы знаем, что JavaScript работает с DOM и изменяет его при необходимости. Но DOM не был создан для постоянного динамического изменения. В итоге, если нам нужно изменить сразу много узлов, браузеру придется фактически перерисовывать DOM-дерево заново, а это будет занимать огромное (по меркам Интернета) количество времени. Для решения этой проблемы React использует так называемый “Virtual DOM”. Это виртуальная копия DOM-дерева страницы, и React работает именно с ней. Когда в Virtual DOM происходят изменения, реальный DOM синхронизируется с Virtual DOM, и перерисовываются только те узлы, которые были затронуты изменениями. Эта синхронизация происходит по принципу

наблюдателя: “Я ничего не трогаю, как только состояние Virtual DOM изменилось, я сразу проверяю, надо ли синхронизироваться с реальным DOM”. В данном случае работает принцип WebSocket и реактивного программирования в целом: “Как только изменяется состояние в одном месте, я сразу меняю что-то в другом месте без запросов”.

В-четвертых, это Bootstrap. Это первый и самый популярный фреймворк для верстки (CSS-фреймворк). Он позволяет без особых проблем верстать адаптивные приложения и используется во многих проектах (часто в совокупности с пре- и постпроцессорами, фреймворками для JS и другими инструментами для фронтенд-разработки). Разработан и поддерживается Twitter. Последняя версия 4.2 (на декабрь 2019).

Вот ещё несколько популярных фреймворков и библиотек для JS:

- Vue.js — набирает популярность, представляется как более простая и легкая замена Angular.js. Очень перспективный фреймворк с большими возможностями.
- Backbone.js. Это скорее библиотека, позиционирующаяся как крупный фреймворк, и один из первых MVC фреймворков. Используется, например, в Trello.
- Ember.js. Достаточно крупный и сложный фреймворк. В последнее время уступает Angular и Vue.
- Knockout.js. Один из старейших фреймворков. Поддерживает очень большую совместимость с браузерами (даже очень старыми), но из-за этого достаточно медленный.
- Lodash и Underscore. Достаточно популярные и легкие в использовании библиотеки, которые расширяют возможности стандартного JS.
- Polymer. Набирающий популярность фреймворк, который позволяет делать легковесные приложения. По функциональности отстает от того же Vue.
- D3.js. Библиотека для визуализации графических объектов: например, для построения графиков и диаграмм.

ИНСТРУМЕНТЫ ДЛЯ СБОРКИ И АВТОМАТИЗАЦИИ ФРОНТЕНДА

При разработке большого проекта нам часто нужно автоматизировать какие-либо рутинные действия. Пример: мы внесли изменения в код, нам нужно, чтобы эти изменения автоматически попали в систему контроля версий, были прогнаны по тестам и, в случае успеха, попали в прод. Или же у нас весь проект состоит из сотен файлов, и их нужно как-то объединить. Для всего этого используются различные инструменты, про которые речь пойдет далее:

ПРЕОБРАЗОВАНИЕ СОВРЕМЕННОГО КОДА JAVASCRIPT

Развитие JavaScript идет в последнее время достаточно быстрым шагами, и новые версии языка появляются практически каждый год. Из-за этого разработчики браузеров могут не успевать за новыми изменениями и не поддерживать новые стандарты. Нужно понимать, что новые стандарты делают язык лучше и проще для разработки, а также добавляют ему функциональности. Поэтому, чтобы код, написанный по новым стандартам, работал в старых или не поддерживающих конкретные стандарты браузерах, используются специальные программы-транспиляторы. Самый известный транспилятор для JavaScript — Babel. Его задача — обеспечить поддержку старых браузеров. Как это работает: он перекомпилирует (транспилирует) текущий код, написанный по новым стандартам (последний ES9 на декабрь 2019), в один из предыдущих стандартов (ES5). Фактически, он вставляет некие “костыли” для добавления необходимой функциональности в старые версии языка или просто переводит новые синтаксические конструкции в старые. Например:

let func = () => a + b станет после транспиляции: var func = function(a,b) {return a+b;}

Как мы видим, Babel заметно увеличивает размер кода, который получает браузер и нередко такой код работает не так оптимально, как хотелось бы. Поэтому программисты не всегда отдают предпочтение Babel, несмотря на то, что он сильно

упрощает жизнь. Иногда лучше писать код в рамках старых стандартов и сразу думать об оптимизации.

СБОРКА ПРОЕКТА НА JAVASCRIPT

В современных проектах один код на JavaScript разбивается на много маленьких обособленных файлов, каждый из которых отвечает за свою часть работы программы. Соответственно, эти файлы надо как-то собирать вместе и подключать к HTML-странице. Теоретически, если у нас есть 10 файлов JavaScript, то мы можем на HTML странице прописать все эти 10 файлов.

Но это было бы слишком громоздко, да и файлов обычно гораздо больше. На помощь приходят специальные программы-сборщики, которые в режиме реального времени собирают все файлы в один. Это удобно ещё и потому, что мы сразу увидим в браузере результат внесенных в код изменений (обычно нужно каждый раз обновлять страницу, на которой выполняется скрипт).

Одним из самых популярных сборщиков для JS является Webpack. Также набирает популярность Browserify. От последнего Webpack отличается тем, что он предоставляет большую гибкость и возможность настройки сборки, а также автоматизирует часть других задач (подробнее про такую автоматизацию ниже). Зато Browserify более простой в настройке, и поэтому его гораздо быстрее развернуть на проекте.

Для сборки проектов также часто используются Gulp и Grunt, но они скорее являются таск-менеджерами, а не сборщиками.

ТАСК-МЕНЕДЖЕРЫ

В процессе разработки фронтенда нам часто нужно как-то автоматизировать часть рабочих процессов. Есть, например, такая вещь, как минификация кода. Когда разработчик пишет код, он делает его легко читаемым. Другими словами, он расставляет пробелы, отступы, лишние знаки препинания (в некоторых случаях они не нужны) и так далее. Все это увеличивает размер файла (даже пробелы), и для оптимизации загрузки этого файла клиентом нам нужно из него все лишнее удалить. Это и называется минификация. Чтобы не выполнять её вручную, мы

можем использовать таск-менеджеры. Одними из самых популярных таск-менеджеров являются Grunt и Gulp. Понятно, что с помощью них можно и выполнять сборку, и автоматизировать другие процессы, связанные с фронтендом. Но все-таки для сборки лучше использовать специальные инструменты-сборщики, потому что лучше использовать узкоспециализированные инструмент для каждой задачи, нежели один многофункциональный нож.