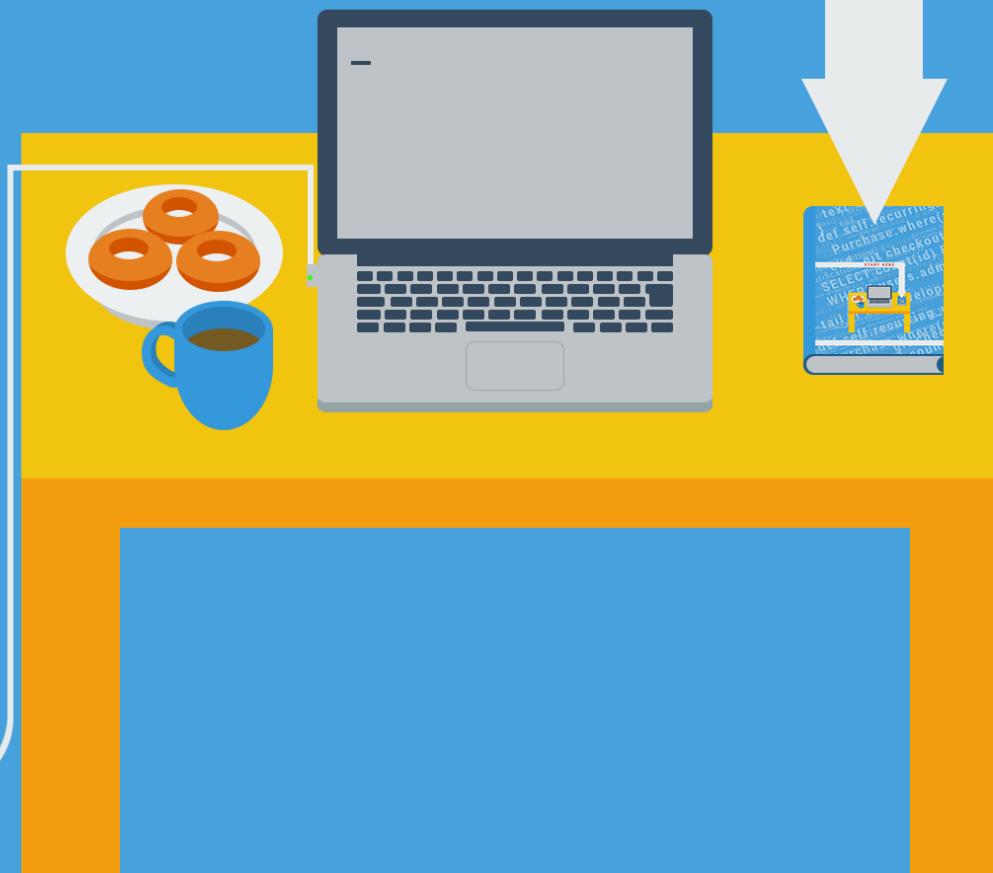


Кирилл Ширинкин

самообразование веб-разработчика

откуда начать

и куда двигаться



МОТИВАЦИЯ



Когда мне было 19 лет, я начал ради интереса рассыпать своё резюме в различные компании в Берлине. В основном, ради того, чтобы попрактиковать свои знания в английском. На тот момент я ещё ни разу не разговаривал ни с кем на английском по-настоящему, поэтому первые несколько собеседований по скайпу прошли не лучшим образом, и меня никто не взял на работу.

Но потом произошло кое-что удивительное. После прохождения очередного собеседования (четвёртого по счёту) и выполнения тестового задания мне предложили работу. Вот так просто: мне дали контракт и сказали, что ждут меня в Берлине. А пока оформляется виза, предложили ещё один контракт — на удалённую работу.

Я не мог поверить своим глазам и ушам: даже по удалённому контракту мне платили в два раза больше средней зарплаты младших программистов в Перми. Да чего там, подозреваю, что мне платили столько же, сколько сейчас получает большинство разработчиков в родном крае. А то количество денег, которое мне стали платить по приезду в Берлин и вовсе могло свести с ума: я наконец-то смог позволить себе снимать собственное жильё, питаться каждый день в кафе и ресторанах и путешествовать по всей Европе. А самое важное, чем я сильно горжусь, я слез с шеи родителей гораздо раньше, чем получил диплом.

Но на этом всё не остановилось. Спустя год после работы в этой компании я ушёл оттуда и начал искать новую работу. И спустя всего две недели поисков лидеры рынка онлайн-изучения языков Babbel.com предложили мне работу Ruby on Rails разработчиком, с зарплатой выше предыдущей на четверть.

Какие выводы можно сделать из этой истории? Скорее всего, в первую очередь, вам придут в голову мысли вроде: “вот же везунчик” или, что ещё хуже, “вот ведь юный умник/гений/вундеркинд”. Но, на самом деле, выводы должны быть следующими:

- Во всём мире огромный, чудовищный спрос на хороших разработчиков
- Не нужен никакой диплом, чтобы стать этим самым хорошим разработчиком

Я начал активно изучать Ruby on Rails летом 2011-го года, а в августе 2012-го мне выслали контракт на удалённую работу в Берлине. Так что давайте взглянем на вещи трезво: если 19-ти летний ленивый студент, сильно привязанный к видеоиграм и сериалам, смог найти высокооплачиваемую работу в Кремниевой долине Европы, то это по плечу любому, кто решит серьёзно заняться разработкой веб-приложений.

Эта книга для тех, кто всегда мечтал разрабатывать приложения, но не знал, с чего начать. Для тех, кто потерял мотивацию к изучению где-то между установкой линукса и настройкой RVM. Для тех, кто только раздумывает о том, чтобы сменить область деятельности, но не уверен, что найдёт на это время. Для всех тех, кто считает, что стать программистом, это сложно и требует высшего образования (в следующей главе я развею этот миф).

ПРОБЛЕМЫ ВЫСШЕГО ОБРАЗОВАНИЯ



Признаем честно, у высшего образования множество проблем. Я не буду акцентировать внимание на недостатках системы ВО в конкретных странах, а постараюсь указать общие недостатки любого учреждения.

Начнём с того, что пять (в среднем) лет учёбы не дают никаких гарантий трудоустройства. Почему? Потому что пять лет учёбы не дают никаких гарантий, что по их истечении вы действительно будете хоть что-либо знать. На момент написания этого руководства мои бывшие одногруппники яростно пишут дипломы. Я проучился с ними три с половиной года и с уверенностью могу сказать, что как минимум треть из них совершенно не готова к настоящей работе. С полученными знаниями максимум, на что они могут рассчитывать, – позиция младшего программиста в стагнирующей локальной “IT” компании. В

которой, конечно же, нет ни интересных проектов, ни интересных задач. Только скука и работа с устаревшими технологиями.

Почему так происходит? Дело в том, что в большинстве случаев студенту и не нужно ничего знать, чтобы закончить ВУЗ. Многие предметы можно успешно “хакнуть”: купить готовые работы, списать на экзамене, выучить лишь 10% материала и тем самым дотянуть до тройки. Я знаю это по личному опыту, потому что проделывал подобное со многими, очень многими предметами во время своего обучения. Социальная инженерия, красноречие и прочие навыки, приобретаемые в процессе такого вот “хакерства” процесса обучения безусловно, нужные в жизни вещи. Но они не помогут научиться делать хоть что-либо важное.

Казалось бы, очевидная вещь: если хочешь научиться что-то делать, то нужно прикладывать усилия к изучению предметов учебной программы. Но только если учебная программа соответствует некоторым требованиям. Попробую сформулировать вопросы, на которые хочет знать ответы каждый студент:

- Зачем нужен этот предмет в реальной жизни?
- Как теоретические знания связаны с реальными приложениями?
- Насколько близки заданные практические упражнения к реальным задачам?

Для многих (слишком многих) предметов во время обучения в университете я так и не нашёл ответов на эти вопросы. Для остальных (к ним относятся большинство профильных предметов) ответы были даны гораздо позже и не университетом, а жизненным опытом.

Сложно обвинить в этом кого-либо. Так уж сложилось, что учебная программа в ВУЗах состоит из десятка предметов каждый семестр, и ни у кого не остаётся времени на то, чтобы действительно понять эти предметы и, самое главное, применить их на практике.

Прозвучит абсурдно, но так называемая “практика” в моём случае проводилась впервые только после третьего курса (спустя три года после начала

обучения) и не имела ничего общего с тем, чем я хотел заниматься и о чём мечтал (разрабатывать крутые веб- и мобильные приложения).

Итого: система высшего образования не оставляет ни единого шанса на выживание энтузиазму и любви к профессии и не даёт никаких объяснений, зачем, как и что пригодится в реальных условиях. Студенты, утратив запал где-то между вторым и третьим семестром, разочаровываются во всём процессе обучения и начинают бесцельно выбивать “хотя бы тройку” или повышенную стипендию и, как результат, тратят пять лет своей жизни впустую.

Если вы студент и перечисленные выше проблемы вам знакомы, то не отчаивайтесь: депрессивная часть этой книги закончилась, и впереди вас ждёт мир, полный счастья, интересных задач и денег (если будете стараться).

Если вы закончили школу и думаете, куда поступать и поступать ли вообще, то, надеюсь, эта глава поможет вам сделать правильный выбор.

В любом случае, пора переходить к сути этого произведения: как начать разрабатывать веб-приложения.

A close-up photograph of a white ceramic cup filled with dark coffee, resting on a matching saucer. The saucer features a decorative pattern of concentric circles in various colors including blue, green, yellow, and red. The cup has a handle visible on the right side. The entire setup is placed on a dark wooden surface.

3

САМООБРАЗОВАНИЕ ВЕБ-РАЗРАБОТЧИКА

Как вы уже могли догадаться по объему этой книги, я не буду мучить вас длинными описаниями, скучными листингами и пространными рассуждениями о той или иной технологии. Вместо этого каждая глава будет построена следующим образом:

Что это

Краткое описание технологии.

Зачем это

Как и зачем эта технология используется в реальных проектах опытными разработчиками.

Минимум

Необходимый минимум, который необходимо знать, чтобы регулярно и эффективно пользоваться этой технологией. Вам не нужно знать все тонкости и особенности использования системы контроля версий, чтобы начать её использовать. В некоторых главах будут даны основные теоретические знания, в других эта секция будет больше похожа на список упражнений. В обоих случаях всё, что упоминается в этой части главы, обязательно к изучению.

Ресурсы

Подборка материалов, которые помогут изучить технологию и начать использовать ее как можно раньше. Книги, статьи, онлайн курсы и прочее – всё здесь. Если у вас возникнут вопросы по Минимуму, то в Ресурсах обязательно найдутся материалы, которые ответят на эти вопросы. Важно: большинство материалов будут на английском языке. Знание английского – *обязательный* навык любого программиста. Не запирайте себя в Рунете, рано или поздно это станет для вас труднопреодолимым препятствием.

Таким образом, цель каждой главы — ознакомить дорогого читателя с новой технологией, помочь начать ее использовать уже сегодня и дать исчерпывающий список материалов, с помощью которых впоследствии можно совершенствовать свои навыки.

Мы начнем с общих инструментов для разработки, необходимых, чтобы делать хоть что-то. Затем посвятим с десяток глав фронтенд-технологиям. Попробовав, что мы ориентируемся в хаотичном мире фронтеда, мы

немедленно перейдем к бэкенду, а после научимся деплоить уже готовые приложения на сервера.

И самое важное: это не подробный учебник и здесь не будет стен текста и подробных объяснений. Всю книгу реально осилить за час. Тем не менее я рекомендую регулярно к ней возвращаться по мере продвижения в вашем обучении. Это ваш путеводитель, грубая карта в волшебный мир разработки веб-приложений, с которым самообразование будет более направленным и конкретным, но при этом по-прежнему останется самообразованием.

```
st login: Sun Jun 22 21:42:16 on ttys000
Pictures cd ~/Documents
Documents ls
ernote Snapshot 20131211 193121.jpg Solarized Dark.terminal
Documents ls -ln
tal 1088
w-r--r--@ 1 501 20 436457 Feb 21 12:46 Evernote Snapshot 20131211 193121.jpg
w-r--r--@ 1 501 20 112034 Feb 21 12:22 Solarized Dark.terminal
w-r--r-- 1 501 20 117 Feb 26 14:26 credentials.csv
Documents mkdir unix
Documents cd unix
unix touch test.txt
unix echo "Hi" > test.txt
uote>
unix echo "Hi" >> test.txt
uote>
unix echo "Hi" >> test.txt
unix ls
st.txt
unix cat test.txt
t: test.txt: No such file or directory
unix cat test.txt

unix
```

4

UNIX-подобные системы

Что это

Под UNIX-подобными системами я в первую очередь понимаю Linux и Mac OSX. UNIX системы отличает своя философия и подход к организации всего. Из некоторых особенностей:

- Всё – файл. Например, физические устройства представлены в виде файлов. И хранение настроек системы. И вообще всё, что можно было представить файлом, сделали файлом.
- Программы должны выполнять одну конкретную задачу и выполнять её хорошо. Например, поиск по файлу. Или подсчёт количества строк в файле.

- Эти самые простые программы можно и нужно объединять в конвейеры, где результат выполнения одной программы передаётся в следующую. Это, согласно философии UNIX, гораздо лучше, чем монолитные программы, выполняющие сразу тысячи функций.
- Активное использование командной строки. Очень активное.

Зачем это

У многих начинающих разработчиков возникает желание придерживаться своих пагубных привычек и продолжать работать на Windows. В основном, потому, что они боятся резко сменить всю свою рабочую/развлекательную среду. Признаюсь, я тоже был таким. Во время изучения фронтенд-технологий я всё ещё использовал Windows. Аргументов в пользу использования линукса было мало, а денег на MacBook не было в принципе.

Тем не менее, рано или поздно придёт осознание, что дальше так жить нельзя. Разработка на Windows возможна, но проблематична. Особенно, если вы будете использовать те технологии, о которых пойдёт речь дальше. К примеру, использование Git, о котором я расскажу в следующей главе, на Windows затруднено уже этапом установки, в то время как на Linux/MacOS нужно лишь вбить одну команду в консоль. Более того, если вы решите использовать Windows для разработки на Ruby on Rails, то обязательно столкнётесь с проблемами, специфичными только для этой ОС. И тут вам вряд ли смогут помочь, потому что подавляющее большинство разработчиков используют для работы либо Linux, либо MacOS. И ещё один аргумент: на любом нормальном сервере будет стоять именно UNIX-подобная система. И, поверьте мне, чем ближе конфигурация вашего рабочего окружения к реальному серверу, на котором будет в итоге работать ваше крутое приложение, тем лучше. О, кстати, сам Ruby заточен именно под UNIX системы.

Минимум

Самое важное, к чему вам необходимо будет привыкнуть, это консоль. Держите её всегда открытой. Научитесь передвигаться по папкам командой `cd`. Копировать файлы командой `cp`. Удалять командой `rm`. Искать по файлам при помощи `grep`. Вывести список файлов командой `ls`. Самое важное, – узнайте

как работают права доступа в UNIX, как управлять пользователями и группами. Создайте несколько и потренируйтесь, переключаясь между ними и создавая папки и файлы. Обязательно прочитайте, чем отличается 755 от 644, и что это вообще за цифры. Ну, и не забудьте узнать, как работает *ssh*, и настроить его у себя. Ещё несколько полезных команд, которые лично я использую каждый день: *cat*, *tail*, *pgrep*.

Я был бы рад рассказать о всех командах подробнее, но тогда я легко выйду за пределы темы этой книги. Тем не менее, все эти программы просты в освоении и нужны регулярно.

Ресурсы

[Limitations in running Ruby/Rails on windows](#) — дополнительная аргументация против Windows для веб-разработки на Ruby on Rails;

[Learn UNIX in 10 minutes](#) — отличный и короткий вводный курс в основные программы UNIX;

[The Unix Programming Environment, Brian W. Kernighan, Rob Pike](#) — единственная книга, которая вам нужна, чтобы начать разбираться в UNIX системах.



Что это

Git – это система контроля версий. Это означает, что если, например, у вас есть файл, над которым вы долго работали и перерабатывали, то все версии этого файла будут сохранены в git и у вас будет возможность вернуться к каждой из них.

Зачем это

Система контроля версий дает следующие преимущества:

- Все версии всех файлов в git репозитории доступны в любой момент, очень тяжело потерять какую либо часть кода

- Разработчики могут работать над одним проектом одновременно, не мешая другу другу и не боясь затереть изменения друг друга. Предоставляемые git возможности совместной работы безграничны.

Git вам придется использовать ежедневно. Это тот инструмент, которым необходимо владеть идеально.

Минимум

Чтобы создать репозиторий, нужно в папке с проектом выполнить *git init*. Чтобы добавить файлы в него нужно сначала сделать *git add имя_файла* (можно сделать *git add .* что бы добавить все файлы сразу) а затем *git commit -m 'описание_сделанных_изменений'*.

Каждые последующие изменения в файлах также добавляются командой *git add*, а затем создается коммит. Считайте, что сделать коммит, это то же самое, что сохранить версию файла.

В git есть ветки. Вы можете вести работу в отдельной ветке, создав её на основе текущей. Самая главная ветка по-умолчанию — master. Считается хорошей практикой в больших проектах вести разработку новой фичи в отдельной ветке, а потом, когда она готова, 'мержить' изменения в основную ветку командой *git merge*.

У git репозитория может быть удаленный оригинал. Например, репозиторий на GitHub. Коммиты в него отправляются командой *git push название_репозитория название_ветки*, вытягиваются *git pull название_репозитория название_ветки*. Таким образом, разработчики ведут разработку на своих машинах и синхронизируют все изменения через удаленный репозиторий. Добавить удаленный репозиторий можно командой *git remote add название_репозитория ссылка_на_репозиторий*.

Обязательно создайте аккаунт на GitHub или bitbucket. У этих сервисов отличная документация по началу работы с удаленными репозиториями. Мне нравится bitbucket возможностью создавать неограниченное число приватных

репозиториев. Тем не менее по всем остальным параметрам GitHub далеко впереди и давно превратился в стандарт и Мекку всего опен сурс сообщества.

Ресурсы

[Try Git](#) — интерактивный вводный курс по Git. Вдумчиво и тщательно пройдя его, вы будете знать о Git достаточно для большинства повседневных задач;

[Version control: best practices](#) — статья о лучших способах использования Git и Github;

[A Git Workflow for Agile Teams](#) — разумный взгляд на организацию работы с git репозиторием внутри команды из трёх и более разработчиков.

6

РЕДАКТОР

Отнеситесь серьезно к выбору редактора кода. Сам я продолжительное время пользовался только двумя: Vim и Sublime Text. Я люблю Sublime Text за простоту и человечность. А ещё в нем можно включить vintage mode, который позволяет редактировать текст, используя команды Vim. Лучшее от обоих миров.

Возможно, стоит приглядеться к RubyMine. Но я считаю, что новичкам его точно не нужно использовать. Автоматизация рутинных задач и куча удобных функций пригодятся опытному разработчику. Если вы только начинаете, то гораздо полезнее будет делать все самостоятельно какое-то время, чтобы понять структуру проекта, какие файлы за что отвечают и т. п. Многие сразу же пускаются во все тяжкие, генерируя тонны кода автоматически, и в итоге

получают огромное количество файлов и не имеют ни малейшего понятия, какой за что отвечает и почему он был создан.

И еще, вернусь на секунду к Vim. Когда вам понадобится редактировать файлы на сервере (а это рано или поздно случится), то у вас не будет доступа к новым редакторам вроде Sublime Text. Зато всегда будет доступ к старому добруму Vim (и ещё некоторым аналогичным редакторам, например nano). Поэтому знать, как им пользоваться на базовом уровне, так или иначе пригодится.

В любом случае, пробуйте разные редакторы, настраивайте их под себя и найдите наиболее эффективное для себя решение. А затем троллите выбор редактора других разработчиков, это неотъемлемая часть нашей культуры ;-)

Следующие несколько глав будут посвящены фронтенд технологиям. Фронтенд это, грубо говоря, все то, что видит и с чем взаимодействует конечный пользователь. Я считаю необходимым начать изучение веб-разработки с фронтенда. Так вы сможете сразу написать что-то осязаемое, что-то, что можно посмотреть в браузере, покликать по ссылкам, показать друзьям.



HTML/CSS

7

Что это

HTML – это язык разметки. Он описывает структуру содержимого сайта при помощи тегов.

CSS, или таблицы стилей, или просто стили, описывают внешний вид сайта, путем выставления HTML тегам различных свойств, от цвета текста до расположения всех частей сайта на экране.

Зачем это

5 лет назад, когда я только решил, что хочу 'делать сайты', я начал именно с HTML/CSS. Это та основа веб-разработки, которую необходимо знать каждому. Первые сайты были ничем иным как набором .html и .css файлов. Абсолютно в каждом веб-приложении рано или поздно придётся идти и править файлы, отвечающие за его внешний вид. И это обязательно будут HTML и CSS. Или их улучшенные версии (о них чуть ниже).

Я не призываю становиться профессиональными „верстальщиками“ (хотя сам начал именно с этого), но умение сверстать простенький сайт из пары колонок считаю обязательным. В конце концов, всё, что в итоге видит ваш пользователь, это именно HTML, CSS и JavaScript код.

Минимум

В определённый момент своей карьеры я осознал, что каких либо строгих правил в написании HTML и CSS не существует. Есть множество подходов и рекомендаций к вёрстке, каждый из которых имеет право на жизнь. Лично для себя я выработал следующие правила, которых стараюсь придерживаться:

- Всегда отделять мух от котлет. CSS описывает внешний вид, HTML структуру. В 95% случаев использовать атрибут style или устаревшие HTML-атрибуты для описания внешнего вида страницы недопустимо.
- Если есть шанс, что какой-то набор стилей будет использован заново, нужно выносить его в CSS класс. Нет ничего плохого в большом количестве классов: при их грамотном и активном использовании ваш CSS не будет зависеть от HTML структуры. Даже поменяв расположение элементов или сами элементы, внешний вид останется без изменений.
- Отдельные элементы страницы должны быть свёрstanы так, чтобы их можно было использовать в любой части приложения, не сломав их внешний вид. Например, внешний вид кнопки должен быть описан один раз и не зависеть от того, где эта кнопка находится.

В целом для каждого проекта нужно стараться писать CSS так, чтобы он превратился в уникальный для этого проекта фронтенд фреймворк, позволяющий легко добавлять новые элементы и верстать новые страницы в кратчайшие сроки.

Пара слов о современных инструментах, облегчающих работу с HTML и CSS. В первую очередь это CSS препроцессоры. Они значительно расширяют стандартные возможности таблиц стилей, но в конечном счёте генерируют самый обычновенный CSS, поддерживаемый всеми браузерами. Например, в SCSS есть поддержка переменных, циклов и вложенных стилей. Всё это позволяет организовывать CSS код гораздо эффективней и сэкономить кучу времени и нервов.

Для HTML тоже существуют чуть более удобные альтернативы. HAML и Slim, например, нацелены на сокращение времени написания HTML кода, предоставляя альтернативный синтаксис, в котором вместо закрывающих тегов используются отступы от начала строки.

Подобные инструменты давно стали стандартом в любом серьёзном проекте, их использование сокращает время разработки и, в случае с CSS, даёт больше гибкости и структуры в написании кода. Знать, как пользоваться этими вещами, чрезвычайно полезно. Мои персональные фавориты – SCSS и Slim.

Ресурсы

[Учебник Ивана Сагалаева](#), — на самом деле, никакой не учебник, а 10 статей о самых важных моментах вёрстки. Для меня это своеобразная Библия, стартовая точка моей карьеры, самые необходимые знания по HTML и CSS;

[Семантическая вёрстка](#) — две статьи о том, как правильней с точки зрения семантики верстать страницы;

[Sass](#) и [Slim](#) — улучшенные CSS и HTML. Пока что они вам не нужны, но не забывайте, что совсем скоро с ними надо будет познакомиться;

[HTML Academy](#) — интерактивные онлайн курсы по HTML и CSS.



WEB-ИНСПЕКТОР

Что это

Встроенный в каждый браузер инструмент для проверки и редактирования HTML/CSS без перезагрузки страницы, анализа отправляемых браузером веб-запросов, дебаггинга JavaScript, работы с cookies и т.д.

Зачем это

Веб-инспектор – незаменимый инструмент фронтенд разработки. Это самый лучший способ быстро отследить, почему CSS не меняет внешний вид элемента как следует, из-за каких запросов страница медленно загружается и,

самое важное, это в большинстве случаев единственный инструмент для быстрого дебаггинга JavaScript кода.

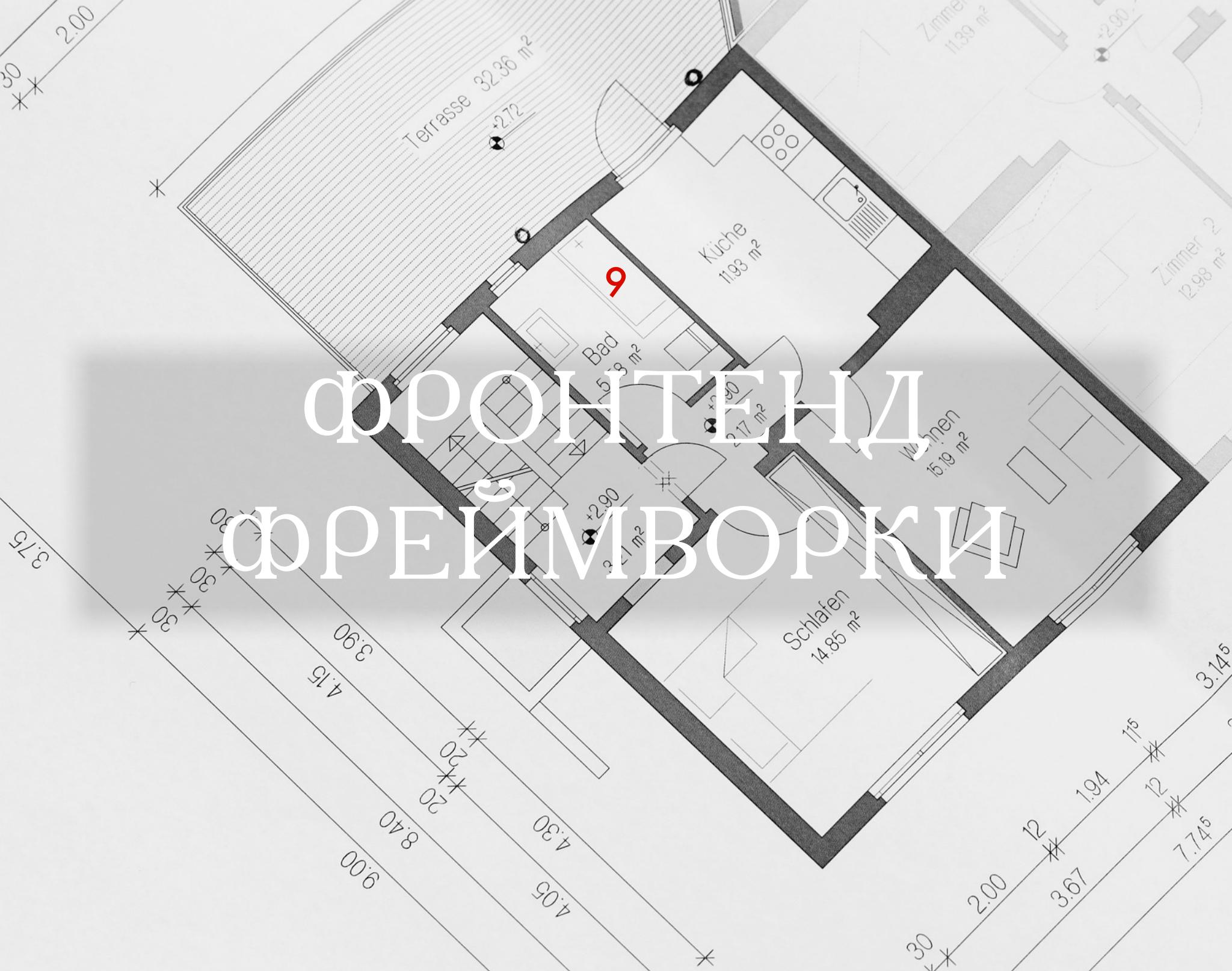
Минимум

Самым удобным и полноценным web inspector'ом многие считают тот, что встроен в Chrome. Не смотря на отчаянные попытки разработчиков Chrome сделать интерфейс инспектора более запутанным и сложным, в нём по-прежнему легко разобраться. При этом там содержится множество продвинутых функций, таких как анализ рендеринга страницы и симуляция мобильных устройств. Порой при отлове самых странных CSS и JavaScript ошибок приходится проводить в инспекторе целые часы, поэтому уметь им пользоваться очень важно.

Ресурсы

[Chrome DevTools](#) — официальная документация к веб-инспектору в Chrome.

фронтенд фреймворки



Что это

Набор CSS и JavaScript файлов. Самые популярные примеры: Twitter Bootstrap, Zurb Foundation.

Зачем это

В какой-то момент фронтенд-разработчикам надоело писать HTML/CSS с нуля для каждого проекта. Так или иначе, в любом веб-приложении есть сетка, определяющая расположение блоков на странице, есть формы, кнопки, списки, элементы навигации, шапки и т.п. Вместо того чтобы каждый раз писать для всех этих элементов CSS, придумали фронтенд фреймворки. По сути, это

большой CSS файл (а зачастую ещё и JS), описывающий часто используемые в веб-проектах элементы. Вместо того чтобы для каждой кнопки писать сто строчек CSS кода, отвечающих за её внешний вид и различные состояния (нажатая, неактивная), вы просто подключаете CSS фреймворк, а потом добавляете к нужному HTML элементу пару классов.

Использование фреймворков значительно снижает время вёрстки. Более того, большинству разработчиков больше нет необходимости детально изучать CSS или ломать голову над дизайном приложения.

Минимум

Несмотря на то, что суть фронтенд фреймворков предельно проста, в своё время они совершили маленькую революцию в веб-разработке. Программистам больше не нужно искать дизайнера, чтобы сделать проект привлекательным. Используя Twitter Bootstrap, они сами могут в кратчайшие сроки написать приятный глазу интерфейс. Так как подобные фреймворки обычно позволяют изменить свои свойства по-умолчанию при помощи scss/less переменных, то созданные таким образом веб-приложения не обязательно похожи друг на друга как две капли воды.

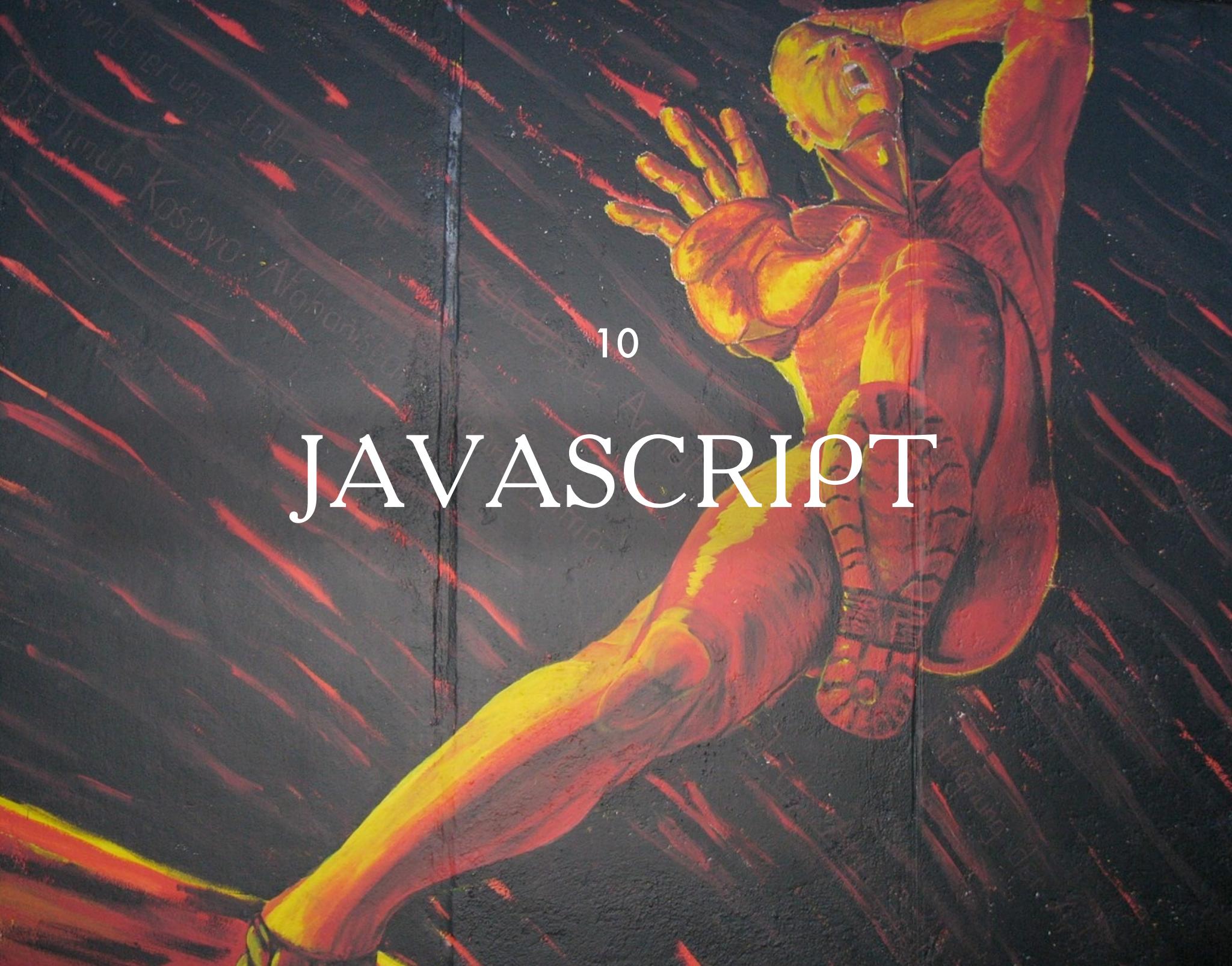
Стоит понимать, что в какой-то момент помочь настоящего дизайнера скорее всего понадобится: нарисовать логотип, подобрать шрифты, правильно расположить все элементы на экране. Но для стартапов и небольших приложений, в условиях ограниченных материальных и временных ресурсов, фронтенд фреймворк скорее необходимость, чем приятное дополнение. Не стоит и говорить, что для создания админок сайтов, для которых отдельный дизайн почти никогда не заказывают, ничего лучше подобных фреймворков не придумать.

Небольшой совет: постарайтесь разобраться, что именно включает в себя используемый фронтенд фреймворк. В большинстве случаев часть CSS и особенно JS можно смело удалить. Размер этих файлов играет решающую роль в скорости загрузки сайта, что, в свою очередь, очень важно для конечного пользователя.

Ресурсы

Bootswatch — бесплатные темы для Twitter Bootstrap. Там же можно найти ссылки на платные (весьма дешёвые) темы, которые совсем не стыдно показать миру;

Why Bootstrap might be **very important** — немного подробнее о том, почему появление таких технологий, как Bootstrap, очень важно.



10

JAVASCRIPT

Что это

По случайному стечению обстоятельств JavaScript является самым популярным языком программирования в мире. С этим утверждением, конечно, можно поспорить, топ языков варьируется от источника к источнику. Тем не менее, безумная популярность JS не подлежит сомнению.

Зачем это

Рано или поздно (скорее всего рано) для вашего приложения понадобится функциональность, не реализуемая стандартными средствами HTML и CSS. Календарик, например. Или динамическая форма. Или всплывающие

подсказки. Или, самое замечательное, обновление части страницы без полной ее перезагрузки – AJAX запрос. И вот в этот самый момент придется учить JavaScript.

Минимум

Ранее было сказано, что HTML отвечает за структуру веб-страницы, а CSS – за ее отображение на экране. JavaScript же отвечает за поведение пользователя. Большинство интерактивных элементов требуют написания JS кода.

К сожалению, JavaScript – далеко не самый лучший язык программирования. На самом деле, зачастую написание кода на нем может стать худшим опытом в вашей жизни. Во многих аспектах этот язык сломан. Во многих других – очень странен. Обязательно посмотрите видео Wat? из ресурсов. И самое грустное – его очень просто изучить и так же просто начать писать на нем неправильный, опасный код. К счастью, многие из этих проблем уже решены, нужно лишь выбрать правильные инструменты.

Первым из них станет jQuery. Это самая популярная JavaScript библиотека, используется на 70% всех сайтов в мире и значительно облегчает написание клиентского кода. Прочитайте документацию к jQuery, научитесь с его помощью управлять DOM, и изучите как использовать бесчисленные плагины (рекомендую начать с jQuery UI). Порог вхождения в эту библиотеку предельно низок, а предоставляемые ею возможности покрывают 80% нужд. Если только вы не пишите single page app, для которого уже потребуется один из JS MVC фреймворков, о которых я подробнее расскажу в бонусной главе.

А вторым обязательным к овладению инструментом станет CoffeeScript. Помните про CSS препроцессоры? Грубо говоря, это JS препроцессор. Пишете код на CS, получаете правильный (и это важно) JS код в браузере. Вообще, у CS два основных преимущества: красивый, приятный глазу, напоминающий Ruby своей простотой код и возможность избежать ужасных ошибок начинающих JS программистов (например, засорение глобальной зоны видимости, о чем вы узнаете из ссылок в секции Ресурсы). Кстати, в Ruby on Rails CoffeeScript используется по умолчанию.

Ресурсы

Codecademy – интерактивные курсы по JavaScript. Подойдут для ознакомления с синтаксисом языка и написанием простых программ;

WAT – невероятно смешное и правдивое выступление о странностях JavaScript;

What You Need To Know About JavaScript Scope – о зонах видимости в JavaScript, поможет вам избежать множества ошибок в будущем;

jQuery Essential Training – если возникают проблемы с документацией к jQuery, то этот бесплатный видео курс поможет во всём разобраться;

CoffeeScript – официальный сайт CoffeeScript.

A scenic view of a rugged coastline. In the foreground, several tall, thin evergreen trees stand on a grassy hillside. Below them is a wide, sandy beach. Large, smooth driftwood logs are scattered across the sand. In the middle ground, a prominent, dark rock formation juts out from the water. The ocean extends to the horizon under a clear blue sky.

11

RUBY

Что это

Объектно ориентированный язык программирования, созданный в Японии и известный своей красотой, легкостью изучения и богатыми возможностями. Стал особенно популярен с появлением фреймворка Ruby on Rails.

Зачем это

Почему именно Ruby? На это есть множество причин, я постараюсь указать лишь некоторые из них, особенно важные для сомневающихся в правильности выбора новичков.

Во-первых, Ruby очень лёгок в изучении. На самом деле, я сомневаюсь, что есть более лёгкий в изучении язык программирования. Такой низкий порог вхождения обусловлен простым и понятным синтаксисом, наличием потрясающих обучающих ресурсов, онлайн курсов и тестов и очень дружелюбным сообществом.

Во-вторых, Ruby востребован. В основном, конечно, благодаря Ruby on Rails. Корректнее будет сказать, что востребованы разработчики, владеющие этим фреймворком. Более того, в среднем зарплаты Ruby программистов выше, чем, например, PHP-программистов, что объясняется, возможно, меньшим количеством разработчиков на этом языке и ещё меньшим количеством по-настоящему хороших. При этом процент хороших программистов на Ruby выше, чем хороших программистов на PHP. Простым примером востребованности хороших Ruby разработчиков могу быть я сам: в 19 лет, без высшего образования, меня позвали работать в Берлин только благодаря моим навыкам программирования в целом и Ruby в частности.

В-третьих, Ruby полноценен. Вокруг этого языка сложилась крупная и стабильная экосистема, с огромным количеством библиотек, с помощью которых можно решить любую задачу. В некоторых языках программирования для решения многих задач придётся изобретать свой велосипед. Благодаря открытости и активности open source сообщества Ruby, с этим языком подобные проблемы встречаются крайне редко. Я даже не говорю о том, что стандартные средства языка зачастую позволяют не прибегать к сторонним библиотекам.

В-четвёртых, писать код на Ruby – это всегда приятный и весёлый опыт. Сложно аргументировать подобное утверждение, но вы поймёте о чём я, когда сами попробуете.

Минимум

Я нахожусь в неловком положении: минимум знаний по любому языку программирования займёт несколько десятков страниц, а значит, не вписывается в формат этого руководства. Тем не менее я постараюсь дать несколько советов, которые считаю важными.

Большинство начинающих изучать Ruby и Rails сталкиваются с выбором: выучить сначала Ruby или сразу перейти к Rails. Я начал сразу с Rails, предварительно потратив пару часов на чтение про основные особенности Ruby. Дело в том, что существует не так уж много интересных задач, которыми может заняться новичок в Ruby. А зубрить список существующих в языке методов и классов сложно назвать продуктивным времяпрепровождением. Так что постарайтесь не зацикливаться на одном только Ruby, переходите сразу к практике, которая в данном случае будет написанием веб-приложений на Ruby on Rails.

С другой стороны, можно и на чистом Ruby придумать небольшое упражнение. Например, попробуйте с использованием Nokogiri получить с любимого новостного сайта заголовки пяти последних статей. Если вы сможете сделать это на Ruby, то уже точно стоит переходить к Rails.

Ещё будет полезным изучить и следовать различным styleguide'ам. Styleguide – это документ с правилами написания кода на том или ином языке программирования. Я укажу в ресурсах ссылки на самые популярные стайлгайды для Ruby. Внимательно их прочитайте и по возможности следуйте каждому правилу из них. Там много мелочей, которые кажутся не важными, но тем не менее являются таковыми.

Чуть менее строгими документами являются Best Practices. Это набор рекомендаций, как, согласно сообществу разработчиков, наилучшим образом решить ту или иную проблему. В отличие от styleguide, лучшие практики – это не строгие правила, а общепринятые, наиболее корректные способы написать конкретный кусок кода. Тоже очень важная вещь для новичков.

Ресурсы

GitHub Ruby Styleguide – если кто-то и знает, как писать отличный код, то это разработчики из GitHub;

Ruby – официальный сайт будет отличным вводным курсом в программирование на Ruby;

TryRuby – бесплатный интерактивный (и очень классно оформленный) курс по основам Ruby. Обязателен к ознакомлению всеми новичками;

Ruby Koans – набор упражнений, в которых нужно написать код так, чтобы тесты на этот код прошли успешно;

Confident Ruby – относительно свежая книжка про написание красивого, аккуратного кода на Ruby. Чтобы увидеть некоторые из паттернов из книги, прочитайте мою статью “[Несколько паттернов для написания надежного Ruby кода](#)”;

Объектная модель Ruby – потрясающая подробная статья об объектной модели в Ruby. Понятным языком рассказано, почему в Ruby всё – объект и как это работает;

hasBrains – русскоязычные скринкасты о Ruby;

Codemy – бесплатные англоязычные скринкасты о Ruby и Rails;

RubyTapas – англоязычные скринкасты о Ruby, периодически появляются бесплатные выпуски;

CodeSchool – онлайн-курсы по Ruby и ряду других технологий. До определённого момента бесплатные.

GEMS И BUNDLER

Что это

Gems (гемы) – название библиотек в Ruby. Bundler – гем для управления зависимостями между гемами в проектах на Ruby.

Зачем это

Любые библиотеки в программировании нужны для того, чтобы не решать заново задачу, решенную сотни раз другими программистами. Рано или поздно кто-нибудь решит, что его код будет полезен другим людям, оформит этот код в библиотеку и даст всем доступ к нему. Потом другие программисты могут улучшить код библиотеки, и таким образом со временем все сообщество

разработчиков на этом языке программирования получит надежное, протестированное решение конкретной проблемы. В этом процессе кроется великая сила проектов с открытым исходным кодом (open source). Open source сообщество языка Ruby одно из самых активных, и создало столько библиотек – gems, гемов, – что порой написание приложения превращается в игру “собери их всех”.

Так как гемов очень много и для самых разных задач, то не редки ситуации, когда один гем зависит от другого, да ещё и от конкретной его версии. Более того, ваше приложение может нуждаться только в конкретной версии гема, и нужно как-то убедиться, что вы используете именно её. Для таких задач придумали Bundler – менеджер зависимостей для гемов.

Минимум

Сразу же совет: постарайтесь поначалу не злоупотреблять гемами. Написание кода никогда не должно превращаться для вас в мистический процесс использования готовых решений. Сначала попробуйте написать решение сами, затем, когда оно работает, ищите, как проблему решили другие разработчики и при помощи гемов.

По поводу Bundler посоветовать особенно и нечего. К каждому Ruby приложению создаётся Gemfile, в нём вы указываете нужные для проекта гемы и их версии, а затем командой *bundle install* устанавливаете их, после чего генерируется файл Gemfile.lock, в котором указаны все используемые гемы с их версиями. Зачем нужно два файла? Всё просто: если вы, например, указали какой-то гем без версии, то первая установка гемов вытянет самую свежую версию, а повторная установка будет использовать скачанную в предыдущий раз, не сделав автоматического обновления и тем самым уменьшив шансы подключить несовместимую с вашим кодом библиотеку.

В Rails использование Gemfile не вызывает вообще никаких трудностей. Для не-rails проектов на Ruby придётся проделать чуть больше телодвижений, но, опять же, ничего сложного.

Ресурсы

Gemfile – официальная документация Bundler – является лучшим источником информации о Gemfile и управлении зависимостями между гемами.

A close-up photograph of a large pile of ripe, shiny red cranberries. They are densely packed, filling the frame. Some cranberries have small brown stems or "eyes" visible.

13

RVM, RBENV

Что это

Менеджеры установленных версий Ruby.

Зачем это

Редко бывает, что разработчику достаточно одной установленной версии Ruby, особенно учитывая, что новые версии выходят регулярно. Один проект может полагаться на самый свежий Ruby 2.1, другой всё ещё работает только с Ruby 1.9.3. И та, и другая версии должны быть на машине разработчика, и для обеих версий все гемы нужно скачивать по-новой (ведь возможно какой-то гем тоже работает только с конкретной версией Ruby).

Без использования RVM (более популярен) или rbenv использовать сразу несколько версий Ruby крайне проблематично и, чего уж там, не имеет смысла. С использованием этих инструментов всё это становится делом одной команды в консоли.

Минимум

Возможно, стоило начать с этого раньше, но ставьте либо RVM, либо rbenv как можно быстрее и используйте их при разработке.



14

RAILS

Что это

Самый лучший фреймворк для разработки веб-приложений.

Зачем это

Есть два варианта написания веб-приложений: с использованием фреймворка и без. В некоторых случаях писать с чистого листа имеет смысл, но чаще всего вашему приложению нужна будет структура и набор готовых решений для самых распространенных в мире веб-разработки задач. Именно это и даёт Ruby on Rails – структуру и широкий спектр готовых к использованию решений.

Если проводить аналогию с реальной жизнью, то представьте, что вам нужно построить дом. Вряд ли у вас мелькнёт мысль, вооружившись молотком и лопатой, идти строить жилище на ближайшем пустыре. Нет, конечно же, лучше будет следовать стандартам, начать с фундамента, стены делать вертикальными, а потолки – горизонтальными. До вас построили миллионы домов и написали тысячи руководств о том, как их правильно строить.

Вот и с Rails так же. Rails прямо говорит вам, как обрабатывать формы, как связаны ссылки в приложении с внутренним кодом, как связывать этот код с базой данных, как раскидывать файлы по папкам, как генерировать HTML страницы и так далее. Вам лишь нужно следовать соглашениям, и вы получите красиво организованное приложение, понятное любому другому разработчику на этом фреймворке. Более того, следуя этим соглашениям, у вас получится разрабатывать веб-приложения гораздо быстрее, чем без использования Rails или с использованием фреймворков на других языках. Ruby on Rails – это Apple от мира веб-разработки. Со своими правилами, атмосферой и категоричным набором мнений.

Минимум

Начать разрабатывать приложения на Ruby on Rails не просто. И дело не в сложности самого фреймворка – напротив, он был создан с целью убрать всю боль и ненужные телодвижения из процесса веб-разработки. Проблема в пороге входления: помимо самого фреймворка вам нужно будет знать потрясающе большой спектр веб-технологий. HTML, CSS, JavaScript, Ruby, Git, SQL, а дальше ещё хуже: SCSS, CoffeeScript, NoSQL, Third-party API.

Я это говорю не с целью вас запугать, скорее, наоборот, лишний раз напомнить, что цель этой книги как раз в том, чтобы помочь вам начать, не испугаться всего буйства технологий, дать обзор всех необходимых инструментов и важный минимум знаний, которого будет достаточно, чтобы начать разработку приложений. Мы уже прошли половину пути, осталось лишь четыре неизведанных инструмента, которые нам понадобятся. Более того, скоро я вам дам подробный план действий, следуя которому не научиться разрабатывать веб-приложения невозможно.

Что касается Ruby on Rails, то здесь я вновь ограничусь парой советов, которые сильно облегчат понимание всего происходящего внутри этого фреймворка.

Во-первых, всегда смотрите логи. В любой непонятной ситуации, непонятной ошибки – смотрите в логи. Внимательно следите за тем, какие параметры вы передаёте в контроллер и что он делает после получения этих параметров. По сути, всё, что делает Rails, это принимает параметры с HTTP-запросов, передаёт их в соответствующий ссылке контроллер, а затем что-нибудь возвращает (например, другую страницу приложения).

Во-вторых, внимательно, **ВНИМАТЕЛЬНО** читайте официальную документацию. Есть технологии, документация к которым туманна и загадочна (например, Angular.js). У Rails такой проблемы нет. Вам, в принципе, даже не нужны никакие книжки и советы, настолько у RoR всё хорошо с документацией. Мои ученики часто совершают одну и ту же ошибку – ленятся вчитаться в Rails доки. Порой доходит до того, что приходится самому копипастить им прямой ответ на возникшую у них проблему.

Ресурсы

Помимо ссылок ниже, проверьте ресурсы из главы про Ruby, там найдётся пара отличных курсов по Rails.

Ruby on Rails Guides – официальная документация Ruby on Rails;

Rails Tutorial – один из самых популярных курсов по Rails, написанный Майклом Хартом и переведённый на русский язык;

Иерархия контроллеров – одна из моих любимых статей по Rails, описывающая один из способов организации кода в Rails;

Railscasts – легендарные (в определённых кругах) скринкасты по Rails. Если у вас возникла какая-то проблема, то скорее всего для неё уже снят рэйлскаст. К сожалению, проект заброшен автором, но 417 выпусков по-прежнему доступны и обязательны к ознакомлению (по мере необходимости);

Rails Best Practices – самый популярный набор Best Practices в Rails. Необходим всем новичкам;

PHP: фрактал плохого дизайна – статья для всех тех, кто раздумывает о выборе между Rails и PHP. После её прочтения выбор становится очевидным;

FJCorp – я регулярно пишу статьи о Rails, в том числе рассчитанные на новичков. Более того, я занимаюсь менторством, поэтому, если нужна помощь в изучении записывайтесь на консультации.



15

ТЕСТИРОВАНИЕ

Что это

Набор методик проверки корректности написанного кода.

Зачем это

Чем больше проект, тем сложнее оставаться уверенным в том, что все части системы работают и работают правильно. Фич становится больше, периодически вылезают баги. В компилируемых языках программирования часть проблем решается компилятором: если что-то не в порядке, то есть большая вероятность, что вы узнаете о проблеме во время компиляции. В интерпретируемых языках нужно быть вдвойне осторожным.

Чтобы обезопасить себя и повысить уверенность в работоспособности всего приложения, нужны тесты. Тесты – это просто файлы с кодом вида: “если я сделаю А, то результат должен быть Б”. Конечно, никто не пишет тесты без использования специальных библиотек (в мире Ruby это RSpec, MiniTest), пытающихся превратить написание тестов в структурированный и удобный процесс. Но стоит помнить, что на самом базовом уровне – это просто набор кусков кода, проверяющих, что “если А, то Б” выполняется.

Минимум

Если вы только начали изучать программирование, ну, или только начали изучать Rails, то на время забудьте о тестах. Опытные разработчики, которые, возможно, читают эту книжку (зачем?), скорее всего не выдержат таких заявлений и начнут обвинять меня в еретизме. И правда, в Ruby-мире тесты священны и являются неотъемлемой частью процесса написания программ.

Тем не менее, я считаю, что учиться писать тесты до того, как вы научитесь писать что-то полезное, бессмысленно. Вы просто не поймёте их ценность, будете заставлять себя делать то, что ещё совсем не нужно делать при текущем уровне знаний. Когда ваше веб-приложение станет чуть более крупным и вы периодически будете натыкаться на внезапно переставшие работать его части, тогда (и только тогда) вы поймёте, что было бы неплохо, если бы какой-нибудь инструмент сам проверял, что эти сломавшиеся части всё ещё работают, как нужно. И вот в этот самый момент имеет смысл написать ваш первый тест.

Дам ещё один совет: не увлекайтесь. Стопроцентное покрытие тестами (когда абсолютно каждая строчка кода проверена) это хорошая и утопическая цель, к сожалению, не имеющая прямого отношения к реальности. Многие вещи, действительно, не имеет смысла тестировать. Увы, сложно по-началу разобраться, что нужно крыть тестами, а что нет. Для новичков это титанический труд, я знаю. Но со временем вы набьёте руку, а до того дам вам ещё два совета, которые работают для меня лично:

1. Если вы наткнулись на баг в приложении, то, помимо починки, обязательно напишите тест, проверяющий, что бага больше нет.

2. Сконцентрируйтесь на тестах, проверяющих работу всей системы в целом (в RSpec это так называемые request specs). Самое важное, что должны проверять тесты, что ваш конечный пользователь не сталкивается ни с какими проблемами, даже при самых сложных сценариях использования приложения.

Ресурсы

TDD is dead. Long live testing. – создатель Rails недавно наделал много шума, объявив разработку через тестирование мёртвой.

Лучше всего учиться писать тесты по существующим open source проектам. Посмотрите, как тестируются гемы и приложения, доступные на GitHub. Например, [исходный код Spree](#), CMS для интернет-магазинов на Rails. В нём не так просто разобраться, но оно того стоит.

POSTGRESQL

16

Что это

Реляционная система управления базой данных.

Зачем это

Для хранения большей части важных данных в вашем приложении

Минимум

Пожалуй, стоило назвать главу “Системы управления базами данных”, а не навязывать столь явно мои предпочтения. Конечно, у вас может возникнуть вопрос, ‘почему не mysql’, на который я не буду здесь отвечать. Однако, в ресурсах я укажу ссылки на отличный материал, аргументированно объясняющий, почему в новых проектах никогда не нужно использовать MySQL.

Так же я сознательно не упоминаю не-реляционные СУБД, знаменитый nosql. Хотя в ресурсах и на эту тему вас ждет парочка отличных статей.

Данные в реляционных системах управления БД представлены в виде таблиц, у каждой колонки есть тип данных (строка, число, дата – типы данных вариируются от одной СУБД к другой). Реляционными они называются потому, что позволяют очень легко и понятно указывать отношения между таблицами. К примеру, в таблице Книга есть колонка Номер автора, в которой хранится номер строки таблицы Авторы. Таким образом, очень легко найти информацию об авторе книги, при помощи простого SQL запроса. Конечно, если предположить, что у книги может быть несколько авторов, то задача немного усложняется. Как ни странно, но очень хорошим материалом по связям между таблицами может послужить официальная документация по ассоциациям между моделями в Rails.

Как минимум, вам нужно знать две вещи: как писать SQL запросы и как создавать индексы. К написанию SQL запросов я также отношу понимание того, как устроены взаимосвязи между таблицами в базе данных, как использовать JOIN и другие SQL конструкции. А про индексы знать необходимо потому, что иначе вы быстро столкнетесь с проблемами с производительностью.

При использовании Rails поначалу не будет необходимости писать SQL запросы самому. Тем не менее это необходимый и важный навык. Постоянно держите в памяти, что правильный SQL запрос всегда справится с задачей сложного поиска данных лучше и быстрее, чем код на Ruby, выполняющий операции на огромном массиве объектов.

Ресурсы

[Do not pass this way again](#) – великолепная статья, объясняющая простым и понятным языком, почему вам никогда не стоит использовать MySQL;

[Why you should never use MongoDB](#) – ещё одна подробная и интересная статья, на этот раз о том, почему вам никогда не стоит использовать MongoDB;

[PostgreSQL awesomeness for Rails developers](#) – а теперь немного о том, чем же так хорошо именно PostgreSQL, особенно для Rails разработчиков;

[Postgres Guide](#) – руководство для новичков как в PostgreSQL, так и в SQL в целом.



ДЕПЛОЙ

Что это

Процесс "выкладывания" веб-приложения на сервер.

Зачем это

Когда вы закончите разрабатывать ваше веб-приложение, вам захочется поделиться им с друзьями, начать привлекать пользователей, заработать первые миллионы денег. А значит, настанет время деплоить.

Минимум

Как это обычно и бывает в мире Ruby/Rails разработки, для вас уже создали правильные и удобные инструменты для деплоя приложений. Доминирует среди них Capistrano, к третьей версии обзаведшийся шикарной документацией и отдельным сайтом.

Помимо умения писать рецепты деплоя на Capistrano (ничего сложного, интернет переполнен готовыми примерами, а в ресурсах я приведу ссылку на тот, который обычно использую сам), вам нужно будет научиться настраивать VPS. Кроме Ruby, на сервер нужно будет поставить такие вещи, как веб-сервер (Nginx), СУБД (PostgreSQL), сервер приложения (Unicorn) и пачку системных пакетов, чтобы все эти штуки заработали. Более того, нужно будет уметь управлять правами пользователей в Linux – от этого зависит безопасность вашего приложения, я не зря упомянул необходимость изучения этой темы еще в главе про UNIX-подобные системы. Суть вот в чем: каждый компонент системы должен управляться от имени отдельного пользователя и не иметь возможности навредить остальным компонентам системы. Таким образом, если взломают ваш веб-сервер, база данных останется неприкасновенной.

Процесс деплоя происходит примерно следующим образом:

1. Новая версия кода (используем git репозиторий) скачивается на сервер;
2. На сервере выполняется установка гемов, компиляция статических файлов и что угодно еще: зависит от того, что вы укажете в вашем коде для деплоя;
3. После этого приложение запускается указанной вами командой, например, командой для запуска Unicorn.

Связь с сервером обычно происходит при помощи логина/пароля или SSH. Предпочтительней использовать SSH, это в разы безопаснее. Работает он по следующему принципу: на вашем компьютере хранятся два файла с ключами (большая куча случайных символов и букв, зашифрованных специальным образом): публичный и приватный, на сервере хранится список разрешенных публичных ключей. Когда вы пытаетесь подсоединиться к серверу, тот проверяет, есть ли ваш публичный ключ в списке разрешенных, а затем

проходит проверка приватного ключа, чтобы убедиться, что это именно вы. Таким образом, только люди или устройства, ключи которых добавили на сервер, имеют доступ к нему.

На первых порах можно слишком много не думать обо всех тонкостях деплоя и настройки серверов и просто следовать подробным руководствам. Поверьте, в этой области разработки столько всего, что хватит на несколько лет изучения. А нашей первоочередной задачей стоит начать с малого, научиться делать необходимый минимум, а затем постоянно (постоянно) совершенствовать свои навыки. Помните, чем больше промежуток между началом обучения и возможностью что-либо пощупать вживую, тем больше шансов растерять энтузиазм и скатиться в прокрастинацию.

Ресурсы

Heroku – самый простой способ задеплоить приложения, да ещё и бесплатный (до поры до времени). Весь процесс развёртывания приложения заключается в использовании команды `git push`, никаких дополнительных знаний не требуется;

Digital Ocean – один из самых дешёвых и удобных VPS хостингов;

AWS Free Tier – Amazon предоставляет бесплатный ограниченный доступ к своим веб-сервисам в течении года;

Capistrano – самый популярный инструмент для развёртывания приложений. К последним версиям обзавёлся множеством новых функций, но не растерял в простоте применения;

Невероятный Chef – рассказ о Chef, более продвинутом и интересном способе настраивать сервера. Позволяет одной командой настраивать свой сервер до состояния полной боеготовности.

Единственный способ научиться разработке

Теперь, когда вы знаете все, что нужно узнать, чтобы начать разрабатывать веб-приложения, остается лишь один вопрос: как наладить процесс самообучения?

Я верю, что единственный способ научиться разрабатывать веб-приложения, это заняться разработкой веб-приложения.

В начале своего пути я попробовал следовать книжке "Гибкая разработка веб-приложений на Rails" и написал тот самый интернет-магазин, который пишут все читатели этой книги. Я не запомнил ничего из той книги и не решил ни одной интересной проблемы. Хуже того, показать что-либо миру мне тоже

было нечего. Пустая трата времени, как и любые другие приложения, которые пишутся по примерам из книжки.

Спустя пару месяцев после прочтения той книжки я предпринял вторую попытку, на этот раз полностью самостоятельную: я начал писать для своего друга движок для управления подробной энциклопедией рок-музыки. Это оказалось более полезным опытом по двум причинам.

Первая: я писал что-то свое, новое, с нуля. Не было детального плана разработки конкретного скучного приложения-примера.

Вторая: в этом приложении был широкий спектр задач, которые не были решены в книжках. Приходилось очень много гуглить, пробовать разные подходы, биться самому над решением проблем. Более того, именно тогда я впервые начал активно общаться с другими людьми на английском. Русское сообщество было не сильно развито, в то время как на официальном IRC канале Rails сидели десятки готовых помочь иностранцев.

Позже я взялся за разработку другого приложения, уже лично для себя – менеджером личных финансов [FJMoney](#). Написав его тогда, в 2010, я все еще пользуюсь им каждый день и развиваю для своих нужд.

Ещё один фактор, который может сыграть решающую роль в вашем изучении разработки веб-приложений, это наличие ментора. Постарайтесь найти человека, имеющего значительный опыт в мастерстве веб-разработки, и предложите ему взаимовыгодное сотрудничество. Например, вы можете помогать ему с его проектами взамен на помощь в обучении. Я сам регулярно беру на обучение нескольких человек, которым помогаю не сбиться с пути. Некоторые уже запустили свои проекты, другие значительно помогли мне в разработке моих сервисов. Также я даю консультации по скайпу, подробнее о которых читайте на [моём сайте](#).

Сейчас существует немало курсов по веб-разработке, но все они обычно стоят денег и делятся короткий период времени. Вам же нужен ваш постоянный наставник, ваш мастер Йода, который всегда пнёт в нужном направлении,

поможет найти ошибки в вашем коде и ответит на все интересные вопросы или укажет, где искать ответы на неинтересные вопросы.

Попробуйте разместить объявление на различных форумах и почтовых группах. В [google-группе ror2ru](#) постоянно ищут помощников или младших разработчиков. Возможно, и вы там кому-нибудь пригодитесь.

Есть одно но: работать с полными новичками менторам обычно неинтересно. В основном, потому, что полным новичкам и не нужен ментор, так как всё, что им нужно знать на данном этапе, легко изучается самостоятельно. Поэтому постарайтесь продвинуться самостоятельно как можно дальше, для этого действительно нужна лишь сила воли и некоторое количество свободного времени. А как будете готовы, найдите своего наставника, вместе с которым вы дойдёте до первых заработанных на разработке веб-приложений денег.

Значительный скачок в моём личном самообразовании сыграло наличие ментора, потрясающего человека, на которого я около года работал и который, собственно, научил меня работать и учиться.

Таким образом, благодаря реализации своих идей, написания настоящих приложений, а не синтетических примеров, и помощи опытного сильного разработчика я научился веб-разработке на Ruby on Rails и обладал достаточным количеством навыков, чтобы найти первую работу и в скором времени после этого переехать в Берлин.

Именно это позволяет мне считать, что самый лучший способ научиться разработке веб-приложений (да и любых других) – это непосредственно разработка. Не мучайтесь над скучными туториалами. Я уверен, у вас уже есть или легко найдется парочка идей для небольших и классных приложений. Возьмитесь за их разработку. Не бойтесь сложностей. Обучение должно быть полезным и интересным во всех отношениях. Вам не нужно ждать пять лет, чтобы начать делать что-то настоящее. Дерзайте.



БОНУС #1

ПЛАН ДЕЙСТВИЙ

ONE WAY

Разработка приложений – многослойный процесс. Целый зоопарк технологий и языков программирования ждет вас, и понять, как они все связаны, вместе является одной из важнейших задач любого занявшегося самообразованием в этой сфере.

Надеюсь, после прочтения этого руководства в вашей голове появилось понимание, что и как устроено и в каком направлении двигаться.

А теперь, для тех, кого все еще мучает вопрос 'с чего начать?' я осмелюсь предложить поэтапный план действий, следуя которому вы сможете разработать свое первое приложение и походу дела освоить все, что нужно для разработки

веб-приложений. Как и всегда, я опускаю те детали, которые вы обязаны узнать сами. Погнали:

1. Придумайте идею для приложения.

Набросайте несколько идей для веб-приложений. Выберите ту, которая вызывает наибольший интерес. Желательно, чтобы это было приложение, которым вы сами регулярно будете пользоваться после того, как разработаете его.

2. Установите Linux или купите Mac

Тут без вариантов, см. главу 4.

3. Набросайте макеты основных страниц

Не нужно быть дизайнером. Просто нарисуйте карандашом на листе А4 как будет примерно выглядеть ваше приложение.

4. Скачайте Twitter Bootstrap и сверстайте основные страницы с его помощью

Без Ruby, без Rails. Научитесь верстать простые HTML странички при помощи фронтенд фреймворка. Так, чтобы можно было покликать по ссылкам, увидеть разные части приложения.

4.1. Установите git и ведите историю изменений

Пока будете верстать, учитесь пользоваться git'ом. Предварительно стоит пройти trygit. Пишите осмысленные сообщения к коммитам на английском языке. Сверстали страничку – коммит. Ещё одну – коммит. Поправили CSS – ещё коммит.

5. Настройте Ruby, Rails и PostgreSQL и сгенерируйте новое приложение

Всё просто: ставите RVM, затем с его помощью свежий Ruby и Rails. А дальше создайте новое Rails приложение и инициализируйте в нём git репозиторий.

6. Определитесь с сущностями в приложении

Определите необходимые данные, которые будут использоваться. Нарисуйте схемку, отображающую их взаимосвязи (в этом поможет руководство по ассоциациям в Rails, см. ресурсы к главе 15). Грубо говоря, напишите список необходимых Rails моделей. Для блога это будет Post, Category, User, Tag. Для системы учёта финансов Transaction, Account. Вам не нужно учесть всё, только те сущности, без которых приложение невозможно создать. Дополнительно продумайте поля для этих моделей.

7. Сделайте это!

Всё, что вам осталось теперь, – разработать ваше веб-приложение на Rails. Ресурсы из предыдущих глав содержат в себе массу информации и примеров. И это лишь малая толика того, что вы сможете найти сами. Не думайте сейчас о тестах, старайтесь сделать функционирующее приложение как можно быстрей. Смотрите на существующие open source приложения, чтобы понять, как всё должно быть написано.

8. Покажите приложение миру

См. главу про деплой. Начните с Heroku, чтобы задеплоить прототип. Обязательно научитесь деплоить на VPS. Не обязательно платить за настоящий сервер, можно изучить такие технологии, как Vagrant и виртуальные машины, и деплоить на виртуальную машину,ирующую на вашем компьютере.

9. Составьте резюме и ищите работу

Как только написали первое приложение, выложите его на GitHub, напишите резюме, указав в нём все технологии, с которыми успели ознакомиться, и ссылку на ваш код. А затем ищите работу, отправляйте это резюме как можно чаще, проходите собеседования. Я уверен, что вы сможете найти работу junior разработчиком. Удачи.

A close-up photograph of a wooden treasure chest overflowing with various European Union coins. The coins are stacked in piles, showing different denominations and designs. A single four-leaf clover is visible on the left side of the chest.

БОНУС #2

ДОПОЛНИТЕЛЬНЫЕ РЕСУРСЫ

Я собрал ещё несколько ссылок, которые вам пригодятся.

[Teach yourself to code](#) – подборка ссылок на лучшие руководства и обучающие статьи по программированию;

[Hacker Newsletter](#) – еженедельная рассылка лучших статей о стартапах и программировании;

[Ruby Weekly, PostgreSQL Weekly](#) и т.д. – целая армия отличных новостных рассылок. Раз в неделю вам будет приходить подборка лучших материалов;

[10 Articles Every Programmer Must Read](#) – 10 статей, которые должен прочитать каждый программист;

Understanding Computation – для всех тех, кто не учился в университете на программиста и чувствует недостаток теоретических знаний. Великолепная книга с понятными объяснениями самых сложных тем информатики;

Step-by-Step Guide to Building Your First Ruby Gem – о том, как написать свой gem;

A Day to Love Postges – ещё одна статья о том, как крут PostgreSQL;

Справочник “Паттерн проектирования” (<http://design-pattern.ru/patterns>) – о способах организации вашего кода и разбиения его на логические части;

Must Have Gems for Development Machine in Ruby on Rails – подборка гемов, облегчающих разработку Ruby-проектов;

Быстрое вступление в rack – статья о том, что такое rack и зачем он нужен;

Бесплатные книги – огромный список бесплатных книг по программированию.



В какой-то момент вы решаете написать веб-приложение, в котором абсолютно все работает на AJAX запросах, страница никогда не перезагружается, а интерфейс напоминает полноценное настольное приложение, а не традиционный сайт. Вы попытаетесь сделать все при помощи простого JavaScript и пары десятков jQuery плагинов, и очень быстро написанный код станет невозможно поддерживать, невозможно читать и невозможно править. И чтобы избежать такой катастрофической ситуации, вам понадобится инструмент, предоставляющий жесткую структуру кода и соглашений о написании приложений, а также готовые решения для известных проблем (например, для двусторонней связи переменной в коде и текста на странице, выводящего значение этой переменной). Этот инструмент – JavaScript MVC (или аналоги MVC) фреймворки.

Есть большая вероятность, что вам не придётся использовать JS фреймворки ближайшие месяцы и\или годы. Область их применения довольно специфична. Даже в тех случаях, когда стоит задача написать одностраничное приложение, не всегда имеет смысл использовать полноценный JavaScript фреймворк.

Тем не менее популярность подобных инструментов как никогда высока, и, как минимум, знать об их существовании стоит. Рассказать обо всех доступных вариантах невозможно, поэтому я лишь расскажу о тех двух, которые попробовал лично сам в реальных проектах.

Одним из самых известных фреймворком является Backbone.js. Он, пожалуй, самый простой, самый стабильный и самый гибкий. Это даже не фреймворк на самом деле, потому что не навязывает никакой конкретной структуры приложения. Отсюда типичная для мира JavaScript проблема: десятки способов сделать одну и ту же вещь, ни один из которых не признан стандартом. За простотой Backbone.js кроется необходимость изобретать для многих задач велосипеды, которые впоследствии приходится поддерживать. Не самое лучшее решение, если хочется избавить себя от головной боли при организации JavaScript кода в крупном приложении. Осмелюсь сказать, что время Backbone.js прошло. Для небольших приложений он ничем не лучше голого JS, а для крупных создаёт больше проблем чем решает.

Мой фаворит на момент написания этой книги – Angular.js. С моей точки зрения, Angular.js стал потрясающей комбинацией низкого порога вхождения и богатого набора функций. Чтобы начать писать на нём небольшие приложения, понадобится около часа свободного времени. При этом на то, чтобы изучить и использовать все его фичи, понадобятся месяцы. Пока что не было ни одной фронтенд задачи, которую я не смог бы решить при помощи этого фреймворка. А использовал я его для кроссплатформенного мобильного приложения с богатой функциональностью, для небольших виджетов и даже для редактора видео.

Впрочем, Angular.js не идеален. Основными его недостатками я считаю ужасную официальную документацию и порой чрезмерную академичность.

Чтобы понять, о чём я, вбейте в гугл „angularjs service vs factory vs provider“ или прочитайте статью [You have ruined JavaScript](#). И всё же как инструмент для решения конкретной задачи самым быстрым образом он подходит лучше всех прочих.

И это самое важное: использовать JavaScript фреймворки для решения конкретной задачи. Заклинаю: никогда не пробуйте новые фреймворки просто потому, что они кажутся крутыми. В мире JavaScript это особенно опасно, потому что количество технологий ежедневно растёт и важно понимать, что каждая из них изначально была написана для решения специфичной проблемы, а не как серебряная пуля для убийства всех фронтенд-проблем. Более того, некоторые из них были написаны даже не для решения каких-то проблем, а просто потому, что кто-то захотел написать ещё один фреймворк.

Выберите одного любимца и по-настоящему научитесь им пользоваться. А затем используйте его до тех пор, пока не станет понятно, что нужно что-то менять.

КОНТАКТЫ

Если у вас возникли вопросы или предложения, связанные с этой книгой, пишите на fodojyko@gmail.com.

Я регулярно публикую отличные материалы по веб-разработке на fodoj.com. Там же вы можете узнать о платных консультациях, которые помогут сделать скачок в вашем самообразовании как программиста.

Помимо fodoj.com, я активно веду [twitter \(@fodoj\)](#), где делаюсь самым интересным и полезным контентом по веб-разработке и стартапам.

Я всегда рад помочь людям, отважно взявшимся за самообразование и готовых учиться. Ничто так не привлекает в людях, как тяга к саморазвитию и движению вперёд.

БЛАГОДАРНОСТИ

Спасибо!

Вячеславу Роганову за здравую критику и поддержку.

Леониду Сущеву (<https://www.elance.com/s/ratatat/portfolio/>), нарисовавшему для этого произведения потрясающую обложку.

Алексею Пильщиковой за помощь и советы.

Татьяне Николаевне Рогановой за проверку и редактуру текста.

Моим ученикам, которые помогли мне понять, с какими проблемами сталкиваются начинающие разработчики.

Дмитрию Васильцу за то, что научил меня работать и учиться.

Евгению Янусову за то, что помог не растерять интерес к веб-разработке в далёком 2008-ом году.

Родителям за веру в меня и поддержку во всех моих начинаниях.

ЛИЦЕНЗИЯ

Произведение «Самообразование веб-разработчика» созданное автором по имени Кирилл Ширинкин, публикуется на условиях лицензии Creative Commons «Attribution-NonCommercial-NoDerivatives» («Атрибуция — Некоммерческое использование — Без производных произведений») 4.0 Всемирная.

Разрешения, выходящие за рамки данной лицензии, могут быть доступны на странице fodoj.com.