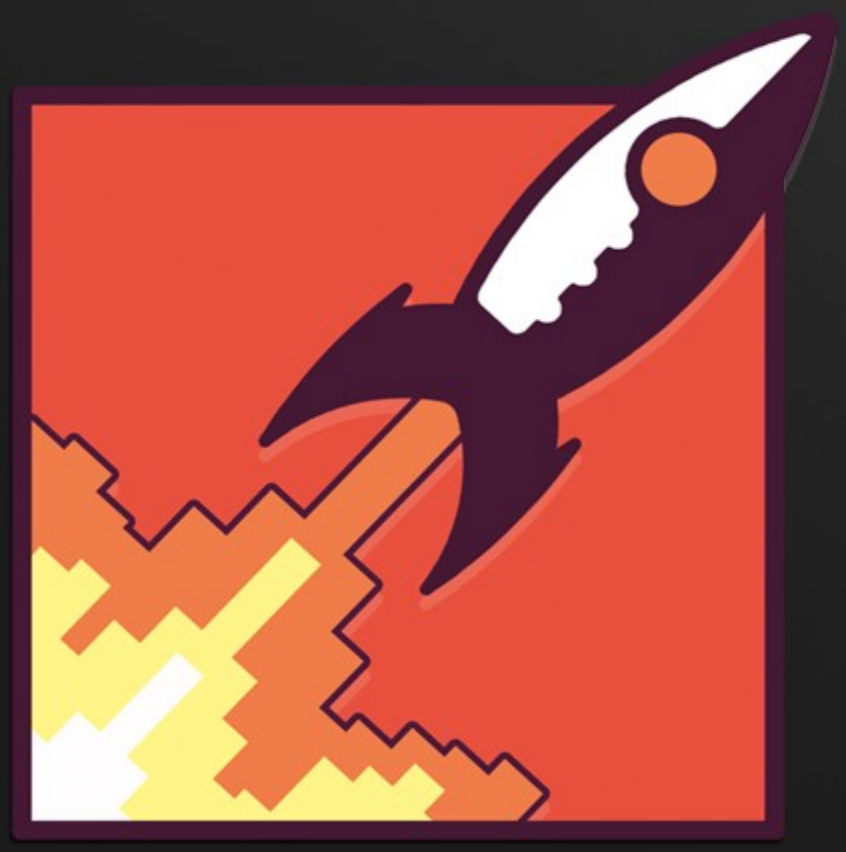
WebDev Roadmap

THE COMPLETE GUIDE TO BECOMING A WEB DEVELOPER



1

MOTIVATION

When I was 19, I started sending my CV to various companies in Berlin, just for fun. Basically I just wanted to practice my English. Until that time, I had never spoken any real English to anyone so the first few interviews on Skype were not a great success and I have not been hired.

But then something amazing happened. After finishing up one of the many job interviews (the fourth in a row) and a test, I was offered a job. Just like that: I was given a contract and told that I was waited for in Berlin. And while I was busy with the visa process I was offered another contract—to work remotely.

I could not believe my eyes and ears when I realized that even working under freelance contract I got twice the amount of an average wage of junior programmers

in Perm. More than that, I guess I was paid as much as most of the developers are getting now in my native town. And the sum they started to pay me on my arrival in Berlin was big enough to drive me crazy: finally I could afford to rent my own flat, eat out every day in cafés and restaurants, and even travel all around Europe. And the most important thing I'm proud of is that I have stopped leeching off my parents way before my graduation.

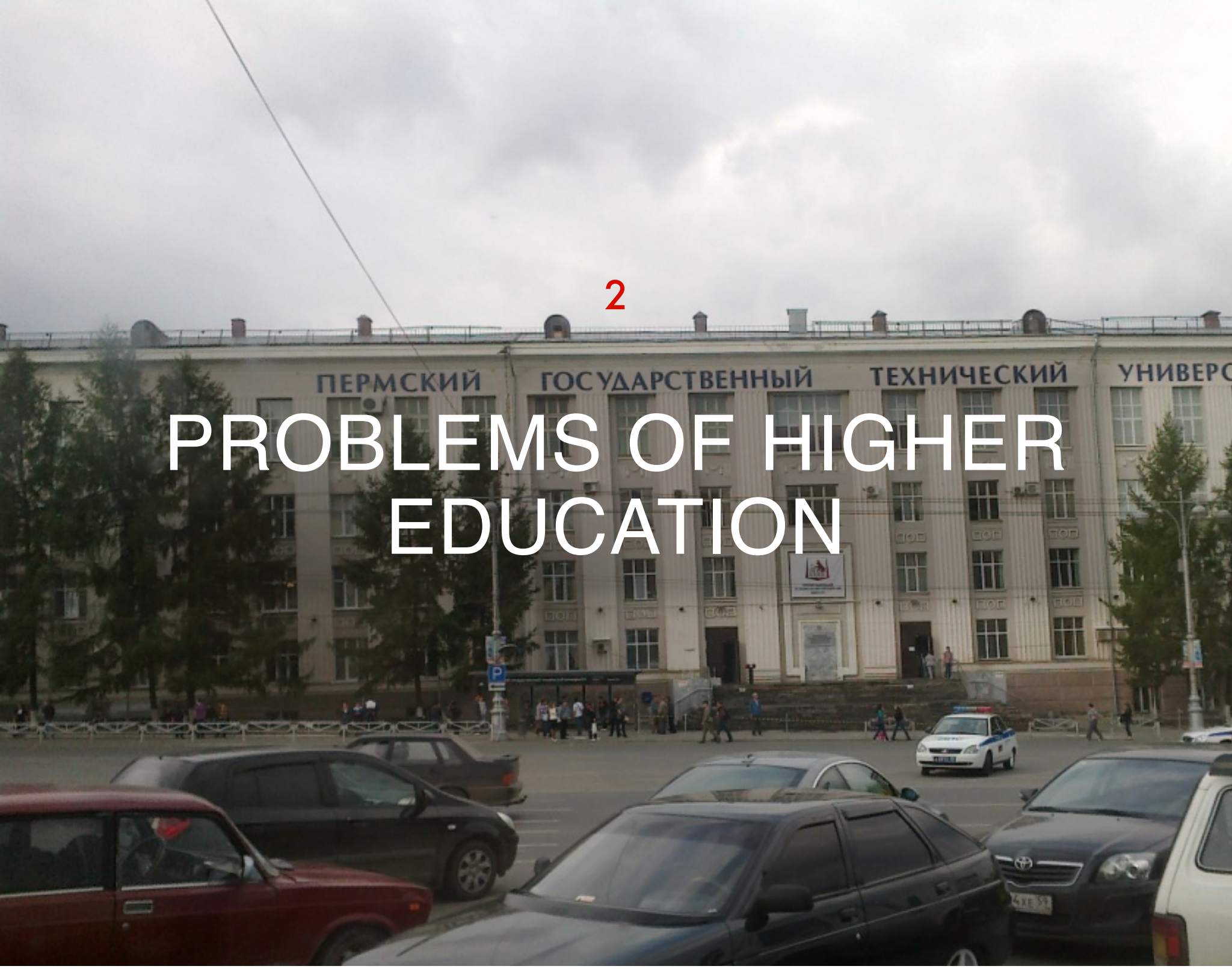
It wasn't only that. After one year working for that company I left it and began to look for a new job. Only two weeks later Babbel.com, one of the leaders of the on-line language learning market, offered me a job as a Ruby on Rails developer, and I was going to make a quarter more than that at my previous job.

So what conclusions can we draw from that story? One might say "what a lucky boy", or even "what a young prodigy/genius/wunderkind". But actually my story is just a proof of two important things:

1. There is a great pressing demand for good programmers throughout the world;
2. You don't need any degree to make that good programmer.

I've been actively learning Ruby on Rails during the summer 2011, and in August, 2012, I've got a freelance contract to work in Berlin. So let's not be under any illusions: if a 19-year-old lazy student addicted to video games and TV-series managed to find a well-paid job in the European Silicon Valley, it means everyone who decides to take up web application development can manage it, too.

This book is for those who have always dreamed of designing applications but didn't know what to begin with. It is for those who have lost their enthusiasm for learning somewhere between launching Linux and setting up RVM. It is for those who are thinking about trying a new field but not sure if they have enough time. The book may also be useful for those who believe that programming is difficult and that one needs college education to be a programmer (I'll be tackling that myth in the next chapter).



PROBLEMS OF HIGHER EDUCATION

Let’s admit it — higher education system faces a number of problems. I am not going to dwell on the disadvantages of higher education system of any specific country. I'll try to describe some of the weak points of any given educational establishment.

To begin with, these several (five on average) years of studying do not give you any guarantee of employment. Why is it so? Because there is no guarantee you'll learn anything by the end of these five years. At the moment I am writing this guide, my former group mates are fiercely writing their graduation papers. We have studied together for three and a half years and I can say that at least one third of them are not ready for doing the real work. With the skills they’ve got, the only thing they can hope for is a position of junior programmer in a stagnating local IT company where they

surely won't have any exciting projects or challenging tasks. Only boredom and working with out-of-date technologies.

Why is it happening? It is because in most cases, a student doesn't have to know anything to graduate from the university. You can just “hack” most of the subjects, buy ready-made papers, cheat on the exam, and learn just 10% of the material to make it to “C”. I know it from my own experience because I've done things like that myself in numerous subjects during my study. Social engineering, eloquence and other skills learned from such “hacking” are definitely helpful in real life. But they won't help you to learn something really important for your future.

It seems to be an obvious thing: if you want to learn how to do something you should work hard on certain academic subjects. But this only makes sense if your academic program meets some necessary requirements. I will try to formulate questions the answers to which every student wants to know:

1. Why do you need this subject in your real life?
2. In what way is theoretic knowledge of particular relevance to real applications?
3. How close are these practical exercises to your real tasks?

For many (too many) subjects that I have been learning at the university I was never able to answer these questions. For the rest of the subjects (mostly related to my majors), I found the answers long after my graduation, not in university but in my life experience.

There can hardly be anybody to blame for that. Historically, university academic program includes a dozen of subjects each semester and nobody has enough time to really get into them and, most importantly, to find any practical use for them.

It may look like nonsense, but the so-called ‘practice’ in my case was organized only after my third year (after three years of studying) and it had nothing to do with what I wanted to be engaged in and what I was dreaming about (developing cool web- and mobile applications).

To sum it up, higher educational system leaves no single chance for enthusiasm and passion, and gives no explanation to why and in what way a particular piece of knowledge might be useful in real life. Having lost their enthusiasm somewhere between the second and the third semester, students tend to be very disappointed with the whole educational system. They try hard on getting “at least C” or an enhanced scholarship and, as a result, waste five years of their life.

If you are a student and have already faced some of the above problems, don't lose your hope: the pessimistic part of the book is over, and there is a whole world of happiness, challenging tasks and money that awaits you (if you work hard enough).

If you are fresh from school and are now thinking about going to college and whether to do it at all, this chapter will help you to make the right decision.

Anyway, it's high time we move on to the very core of this book – how to develop web-applications.

As you could have already guessed from the size of this book, I am not going to bother you with long descriptions, boring listings or long speculation on a particular technology. Instead, each chapter will be built in the following way:

What is it?

A brief summary of the technology.

What is it for?

How and what for the technology is used in real projects by skillful web developers.

Basics

The necessary basics that you need to know to use the technology regularly and efficiently. You don't have to know all aspects and features of the Version Control System to start using it. In some chapters, the basic theoretical knowledge is described, and in others this section looks more like a list of exercises. In both cases, everything mentioned in this part of a chapter is really "must learn" material.

Resources

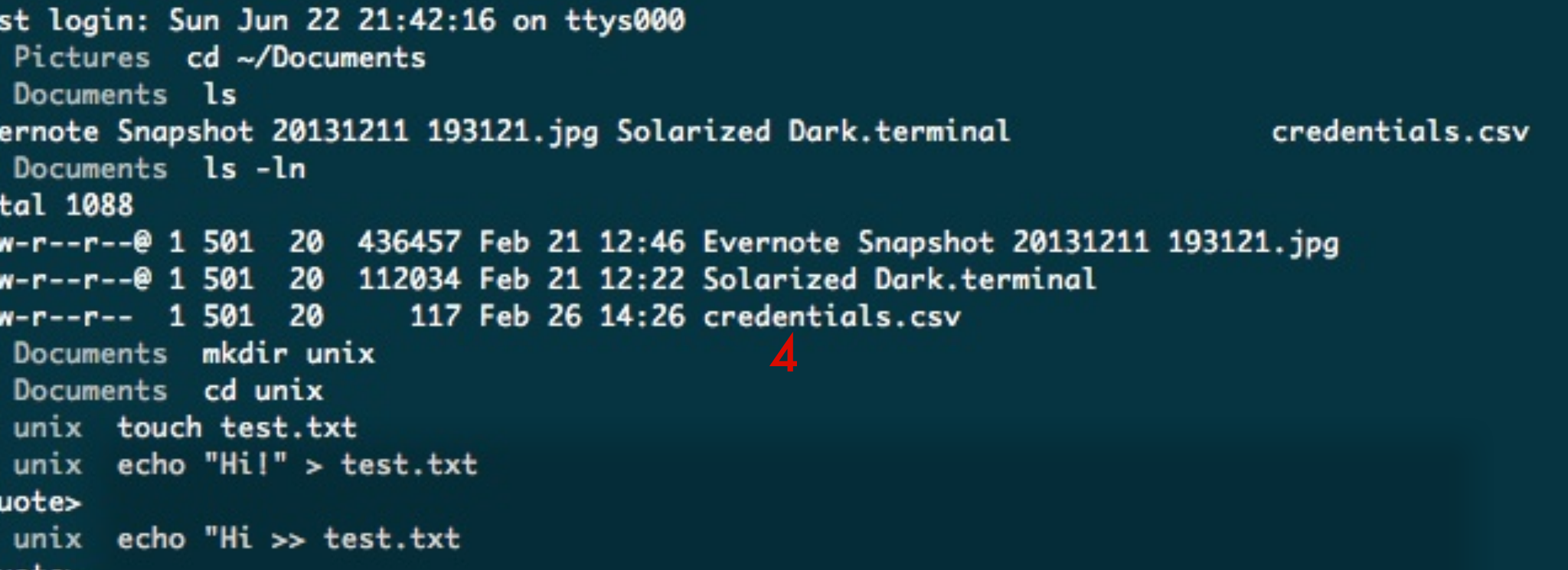
It's a selection of resources that may help you study the technology, and start using it as soon as possible. Here you can find everything from books and articles to online courses, etc. If you still have any questions after reading Basics, you'll definitely find the necessary information in Resources.

Thus the aim of every chapter is to introduce the reader to a new technology, show how to start using it straight away and give an exhaustive list of resources that you can use to develop your skills afterwards.

We are going to begin with common tools you may need just to start doing something. Some later chapters are dedicated to front-end technologies. As soon as we realize that we've learned the ropes of the chaotic world of front-end, we'll get down immediately to back-end, and then study how to deploy ready applications on servers.

The most important thing is that the book is not a detailed textbook and you are not going to see walls of text and explanations rich in details. You can read through the book in just an hour. However, I'd recommend you to get back to it time after time, as you move forward in your training. This is your guide-book, a rough map to

the enchanting world of web development. With this book, your self-education will be more focused and specific, but it'll still be self-education.



What is it?

By Unix-like systems I mean, in the first instance, Linux and Mac OSX. UNIX systems have their own philosophy and approach to organization of everything. Here are some peculiarities:

- Everything is a file. For example, physical devices and storage of system settings are represented as files. Generally, everything that can be a file is made as a file.
- Programs should perform only one task and should do it well, i.e. finding a file or counting lines in a given file.

- These simplest programs may and should be combined into a pipelines where the result of performing one program goes to another. According to UNIX philosophy, this is way better than solid software that performs thousands tasks at once.
- Very wide use of the command line.

What is it for?

Many beginner developers have a perilous inclination to continue their work on Windows. Generally, this is because they are afraid of changing their working/entertaining environment. I have to admit I did the same thing. I used Windows when I was studying front-end technologies. There weren't many pros to using Linux, and I didn't have any money to buy MacBook.

Anyway, sooner or later you'll realize that you can't live like that anymore. Web development on Windows is possible but it has its issues. Especially if you are going to use the technologies mentioned below. For example, if you use Git (I'll describe it in the next chapter) on Windows, you'll get some troubles trying to install it, while on Linux/MacOs you'll only have to enter one command into your console. Furthermore, if you decide to use Windows for developing with Ruby on Rails, you'll definitely face issues that are specific for this OS only. And it is very unlikely that someone could help you because the vast majority of developers use only Linux or MacOS. Another pro argument is that every normal server has UNIX-like system. Trust me, the closer your working environment configuration to the real server on which your cool application will eventually operate, the better. Oh, by the way, Ruby itself is made precisely for UNIX systems.

Basics

The most important thing you'll have to get used to is the console. Keep it open all the time. Learn to navigate between folders using **`cd`** command, to copy files using **`cp`** and delete things with the help of **`rm`**, search through files using **`grep`** and output the list of files with **`ls`**. Above all things you should understand how UNIX file access permissions work, and how to manage users and groups. Create a few groups, and practice switching back and forth and making new folders and files. Be sure to read about 755 and 644: what these numbers are, and what the difference between them is.

Also, learn how ssh works, and customize it for yourself. Some more commands being helpful that I, for one, use every day are **`cat`**, **`tail`**, and **`pgrep`**.

I'd be happy to tell you in more detail about all the commands but then I'll definitely digress from the topic of this book. However, all these commands are easy to come to grips with, and you'll need them on a regular basis.

Resources

Limitations in running Ruby/Rails on windows — a powerful argument against the use of Windows for web development with Ruby on Rails;

Learn UNIX in 10 minutes — a great and brief introductory course to basic UNIX programs;

Unix Toolbox – a collection of Unix/Linux/BSD commands and tasks which are useful for IT work or for advanced users;

The Unix Programming Environment, Brian W. Kernighan, Rob Pike — the only book you need to begin understanding UNIX systems.

What is it?

Git is a version control system. For example, if you have a file on which you've been working on and reworking for a long time, all the versions of it are saved in Git, and you can easily get back to every version.

What is it for?

The version control system has the following benefits:

• You have access to all versions of all files in Git repository at any time, it's almost impossible to lose any part of a code.

• Multiple developers can work on one project at the same time without interfering with each other, and without fear of losing any changes made by a colleague. In Git, the possibilities of collaborative work are unlimited.

You will have to use Git every day, and this is a tool you should have a perfect command of.

Basics

To create a repository you've got to run **`git init`** in a project folder. To add files to it, first use **`git add file_name`** (or **`git add .`** to add all the files at once), and then **`git commit –m 'description_of_changes_made'`**.

Any further changes in the files can also be done with **`git add`**, and then you do a commit. You can consider creating a commit to be the same thing as saving a version of the file.

Git has branches. You can work in a separate branch after creating it on the basis of the current one. By default the main branch is the 'master'. It is believed to be a good practice for big projects to develop a new feature in an individual branch, and then when it's done merge changes into the main branch with the help of **`git merge`** command.

Git repository may have a remote original, e.g. a repository on GitHub. You can send commits there using **`git push repository_name branch_name`**, and get them back by **`git pull repository_name branch_name`**. This is how developers work on their computers and synchronize all the changes using remote repository. To add remote repository use **`git remote add repository_name repository_link`**.

Be sure to sign up on GitHub or bitbucket. These services offer great manuals on how to start working with remote repositories. I like bitbucket for the possibility of creating unlimited number of private repositories. However, in terms of all other characteristics, GitHub is way ahead of butbucket, and has turned into the standard and Mecca of the whole open source community.

Resources

Try Git — an interactive introduction course on Git. If you study it closely and thoroughly you'll know enough to perform usual tasks;

Version control: best practices — an article on the best ways of using Git and Git-nub;

A Git Workflow for Agile Teams — provides insight into the organization of work with Git repository for a team of three or more developers.

6

EDITOR

Be especially careful while choosing a code editor. As for me, for a long time I've been using only two: Vim and Sublime Text. I like Sublime Text for its simplicity and friendliness. By the way, you can enable vintage mode that allows you to edit your text using Vim commands. The best of the two worlds.

It may be worthwhile taking a closer look at RubyMine but I think beginners need not use it at all. Automation of routine tasks and lots of useful functions will be good for a skillful developer. If you are new to programming, you will be much better off doing everything by yourself for some time, just to understand the structure of the project, what files are responsible for what, etc. Many beginners lay themselves out at once, creating plenty of code automatically, and as a result they get a myriad of files and have no idea what each file is responsible for, and why it has been created.

Now I'd say a few more words about Vim. When you edit content on a server (one day it'll happen) you won't have access to new editors like Sublime Text. But you'll always have access to the old-fashioned Vim (and some more similar editors like nano). That's why you got to know the basics of using Vim: it's going to come in handy later.

Anyway, try different editors, make settings as you like and find the solution that suits you most. And then you are free to troll other developers' choice of editor. That's what we do all the time ;-)

The next few chapters are dedicated to front-end technologies. Broadly speaking, front-end is everything the end user sees and works with. I think it's best to start studying web development with front-end. This way you can create something tangible straight away, something that you can view in a browser, click the links and show to your friends.

7

HTML/CSS

What is it?

HTML is a markup language that describes the structure of the website content with the help of tags.

CSS, cascading stylesheets or just stylesheets describe actual appearance of the website by assigning various properties to HTML tags, from the text color to arrangement of all the website parts on the screen.

What is it for?

In 2008, when I've just decided to ‘make websites’, I started precisely with HTML/CSS. This is the basis of web development which everyone needs to know. The first websites were nothing but sets of .html and .css files. In each and every web application you will, sooner or later, have to correct the files responsible for its physical appearance. And these are sure to be HTML and CSS. Or their improved versions (mentioned a few lines below).

While I would never rush anyone to become a professional front-end developer (though this is what I started with myself) but I believe that knowing how to make up a simple website of a couple of columns is a good thing for everyone. Eventually, all that your user sees in the end is just HTML, CSS and JavaScript code.

Basics

At a certain moment of my career, I realized that there are no strict rules in writing HTML and CSS. There are a lot of approaches and recommendations concerning layouts each of which has its own particular use. I, for one, have developed the following rules and I try to follow them:

- Always separate apples from oranges. CSS describes physical appearance while HTML describes structure. In 95% cases, it is a big mistake to use a style attribute or out-of-date HTML attributes to describe appearance of a page.
- If there is a chance that a certain set of styles would be used over again, this set should be taken to CSS class. There is nothing wrong in a large number of classes. If they are widely and properly used, your CSS will never depend on HTML structure. Even if you change the arrangement of elements or the elements themselves, the appearance will remain unchanged.
- Individual elements of the page should be made in such a way that they could be used in any part of the application without breaking their physical appearance. For example, appearance of a button should be described once, and should not depend on where this button is.

Basically, for each project one should try to write CSS in such a way that it would turn into a front-end framework unique for this project, so that one could easily add new elements and build new pages in no time.

And now a few words about current tools that facilitate your work with HTML and CSS. Firstly, there are CSS preprocessors. They considerably enhance the capabilities of stylesheets but in fact they just generate a very common CSS supported by all browsers. For example, SCSS supports variables, loops, and nested styles. All these allow arranging CSS code much more efficiently, and save a lot of time and effort.

There are some more handy options for HTML, too. HAML and Slim, for example, are aimed at reducing the time spent on writing the HTML code by providing alternative syntax using indentation instead of closing tags.

Such tools have long ago become a standard in any serious project, they help boost the development process, and, in case of CSS, lends more flexibility and structure to code writing. It is essential to know how to use these things. SCSS and Slim are my personal favorites.

Resources

Codeacademy HTML & CSS – interactive HTML and CSS course on Codeacademy (in general good resource to learn basics of few technologies);

Sass и Slim — improved CSS and HTML. You may not need them now but keep in mind that pretty soon you'll have to have a closer look at them;

What is it?

A tool embedded in each browser, for checking and editing HTML/CSS without reloading a page, for analysis of web requests sent by the browser, for debugging of JavaScript, working with cookies, etc.

What is it for?

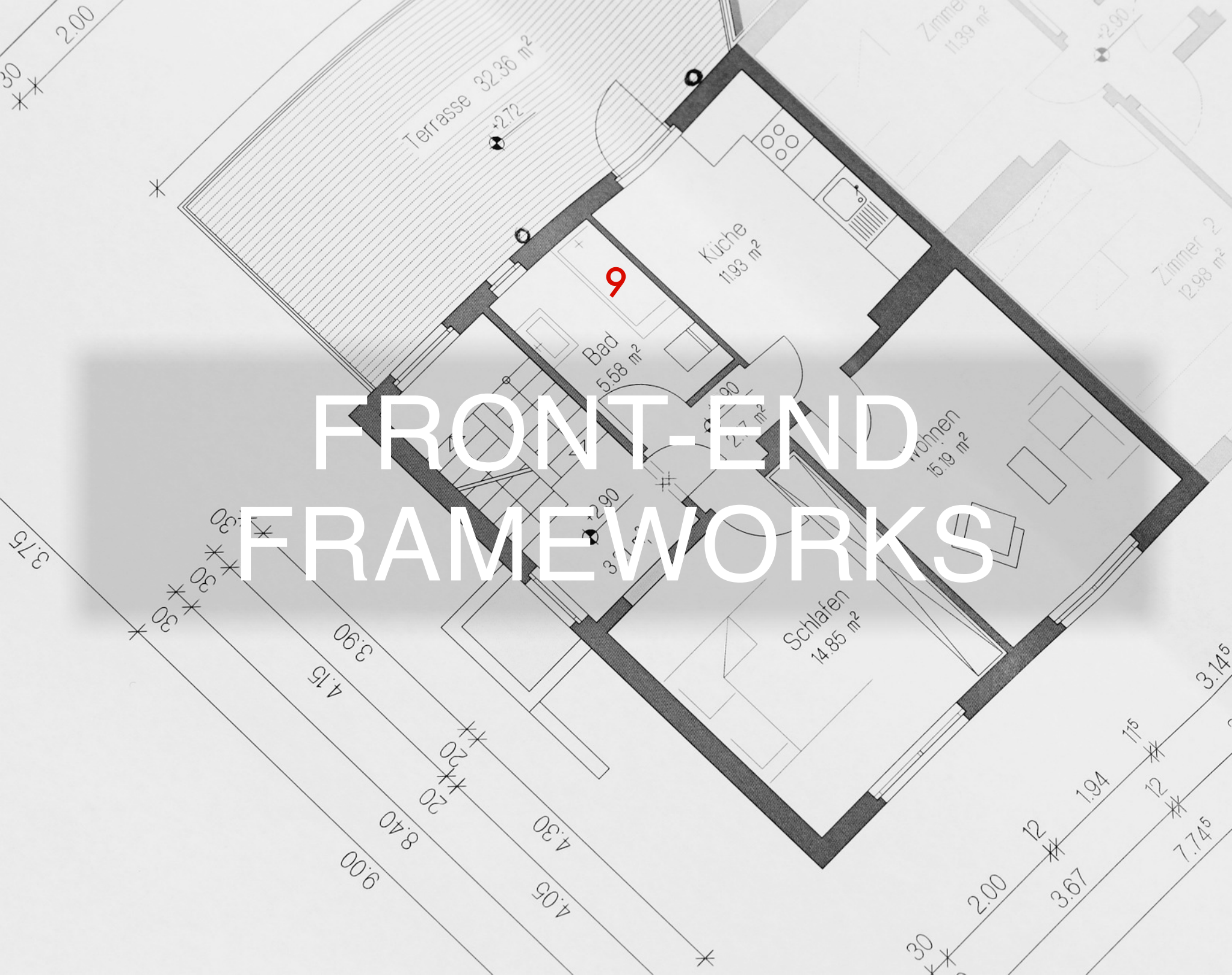
Web-inspector is an indispensable tool of front-end development. This is the best way to quickly find out why CSS does not change appearance of an element properly, due to what requests a page is loading slowly and, what is more important, in most cases this is the only tool for fast debugging of a JavaScript code.

Basics

Many people think that the most convenient and fully-featured web-inspector is the one embedded in Chrome. In spite of the vigorous attempts of Chrome developers to make the interface of the inspector more intricate and complicated, it is still easy to understand. At the same time, it contains a lot of advanced functions such as analysis of the page rendering and simulation of mobile devices. When catching the most strange CSS and JavaScript errors one has to spend hours in the inspector, so it is very important to know how to use it.

Resources

Chrome DevTools — official documentation for the web inspector in Chrome.



FRONT-END FRAMEWORKS

What is it?

It is a set of CSS and JavaScript files. Most popular examples are Twitter Bootstrap, and Zurb Foundation.

What is it for?

At a certain moment, front-end developers became fed up with writing HTML/CSS from scratch for each project. In some way or another, any application has a layout determining the arrangement of blocks on the page, and there are forms, buttons, lists, navigation elements, heads, etc. Instead of writing CSS for all these elements each time, they invented front-end frameworks. Basically, this is a large CSS file (often

also a JS file) describing the elements frequently used in web projects. Instead of writing for each button hundreds of lines of CSS code responsible for its physical appearance and various states (active/inactive), you just connect a CSS framework and then add a couple of classes to the necessary HTML element.

The use of frameworks considerably decreases the time of layout building. Moreover, most developers no longer need to study CSS in detail or to rack their brains to create an application design.

Basics

In spite of the fact that the essence of front-end frameworks is utterly simple, back in their day, they made a small revolution in web development. Programmers no longer needed a designer to make a project attractive. By using Twitter Bootstrap they can build a visually-appealing web interface themselves in no time. As such frameworks usually allow changing their default properties with the help of scss/less variables, the applications created in such a way are not necessarily as like as two peas.

It should be understood that, at a certain moment, you will probably need the help of a real designer to draw a logo, to select fonts, and to arrange all elements correctly on the screen. But for startups and small applications with a limited budget and time a front-end framework is more of a necessity than a nice-to-have. It is needless to say that to create admin panels of websites for which individual design is almost never ordered, there is nothing better than these frameworks.

One more useful tip: try to figure out what exactly the front-end framework you use includes. In most cases, it is safe to remove a part of CSS and especially JS. The size of these files plays a significant role in the website load speed which, in turn, is very important for the end user.

Resources

Bootswatch — free themes for Twitter Bootstrap. There you can also find links to some paid (but rather cheap) themes that are well worth showing to anyone in the world;

Why Bootstrap might be *very* important – some more details about the reasons why the advent of such technologies as Bootstrap is very important.

10

JAVASCRIPT

What is it?

By a coincidence, JavaScript is the most popular programming language in the world. Surely, this statement gives room for argument as different sources praise different top languages. Yet the global popularity of JS is beyond doubt.

What is it for?

Sooner or later — most likely sooner — your application will need the functionality that cannot be implemented by standard HTML and CSS means. For example, a calendar or a dynamic form, or pop-up tips. Or, the most amazing thing — when you

need to update a part of the page without fully reloading it — an AJAX request. This is the moment when you have to study JavaScript.

Basics

As I have already said above, HTML is responsible for web page structure, and CSS is responsible for its appearance on the screen. As for JavaScript, it is responsible for user behavior. Most of interactive elements require writing a JS code.

Unfortunately, JavaScript is far from being the best programming language out there. In fact, writing a code in JS may easily become the worst experience in your life. In many aspects this language is broken. In many others it is very strange. Be sure to see 'Wat?' video listed in the resources section. And the worst thing is that it is as easy to learn, as to start using it for writing an erratic and dangerous code. Fortunately, many of these problems have already been solved, and one just has to select proper tools.

One of them is jQuery. This is the most popular JavaScript library used on 70% of all websites worldwide. It greatly facilitates writing a client code. Read documentation for jQuery, then learn how to use it to manage DOM and make sure you know how to use numerous plug-ins (I recommend starting with jQuery UI). This library is extremely easy to understand, and the resources provided cover 80% of the needs unless you write a single page app for which you may need one of JS MVC frameworks that I will describe in more detail in the bonus chapter.

CoffeeScript is the second tool to be mastered. Remember CSS preprocessors? In its core it is just a JS preprocessor. You write a code in CS and get a correct (this is important) JS code in the browser. Basically CS has two principal advantages: it has a beautiful and visually appealing code resembling Ruby in its simplicity and it helps avoid major mistakes of entry-level JS programmers (e.g. messing with the global namespace scope the idea of which you will get from the links in the resources section). By the way, in Ruby on Rails CoffeeScript is used by default.

Resources

Codecademy – interactive courses on JavaScript. An easy way to get the idea of the language syntax, and learn how to write some simple programs;

WAT – a hilarious but so true lecture on the weirdness of JavaScript;

What You Need To Know About JavaScript Scope – this is about namespace scope in JavaScript, may help you to avoid many errors in the future;

jQuery Essential Training – if you have problems with documentation for jQuery, this free video course will help you to figure out everything;

CoffeeScript – the official CoffeeScript website.

11

RUBY

What is it?

Ruby is an object-oriented programming language created in Japan and known for its beauty, ease of learning and rich features. It became especially popular with the advent of the Ruby on Rails framework.

What is it for?

Why Ruby? There are a number of reasons. I will try to indicate only some of them, especially important for those who doubt it's a good choice for beginners.

Firstly, Ruby is very easy to learn. In fact, I can't name any easier language to study. Such a low entry barrier can be explained by a simple and intuitive syntax, easy access to some of the best learning resources, online courses and tests, and very friendly community.

Secondly, there is a high demand for Ruby, mostly thanks to Ruby on Rails, of course. It would be more accurate to say that there is a high demand for developers who use this framework. Moreover, the average salary for a Ruby programmer is higher than, for example, that for PHP-programmers. One of the possible reasons is that the number of Ruby developers is kind of limited and the number of really cool developers using this language is even fewer. However the percentage of good programmers using Ruby is higher than that of good programmers using PHP. A simple proof of the high demand for good Ruby developers can be myself: aged 19, having no higher education I have landed a job in Berlin just because of my programming skills in general and in particular for Ruby.

Thirdly, Ruby is a full-fledged language. It is supported by a vast, well-developed and sustainable ecosystem with a number of libraries using which you can solve any problem. In some programming languages, for some tasks you will have to reinvent the wheel. But because in Ruby we have a very friendly and helpful open source community, Ruby guys hardly ever experience problems like that. I'm not even talking about the fact that, with the standard means of language available, one often has no need to consult any extra libraries.

Fourthly, writing your code in Ruby is always fun and exciting experience. It is difficult to substantiate such a claim, but you will know what I mean when you first try it yourself.

Basics

I'm now in a tricky position: a minimum required knowledge of any programming language will take several dozens of pages, which is outside the scope of this guidebook. Nevertheless, I will try to give a few tips that are relevant for most cases.

Most beginners looking to learn Ruby and Rails are faced with a choice: learn Ruby first, or go directly to the Rails. I started immediately with Rails, but before that

I had spent a couple of hours reading about the basic features of Ruby. The fact is there are not so many challenging tasks for developers new to Ruby, and there's hardly any use in getting through the whole list of Ruby methods and classes. Just try not to focus too much on Ruby theory and apply it to practice that in this case means writing Ruby on Rails web applications.

On the other hand, one can think of a quick task for plain Ruby application. For example, try, by using Nokogiri, to grab the headlines of the last five articles from your favorite news website. If you can do this in Ruby, you are probably ready to move on to Rails.

Another useful thing is to study and follow different styleguides. A styleguide is a document that contains the rules on how to write a code using particular programming language. You will find the most popular Ruby styleguides in the list of useful resources at the end of this chapter. Just read them carefully and try to follow each of them. There's a lot of little things that you think wouldn't matter, but in fact they really do.

Best Practices are a bit less sensitive documents. They represent a number of recommendations on how to, according to the developer community, solve a particular problem the best way possible. Unlike the styleguide, Best Practices have no strict rules, just some common and the most accurate ways to write a particular piece of code. However, it is one of the most important things, especially for a beginner.

Resources

GitHub Ruby Styleguide – if there is someone who knows how to write a pretty awesome code, these are the guys from GitHub;

Ruby – the official website that serves as an excellent introductory course to Ruby programming;

TryRuby – a free online course on some Ruby basics (very cool design). A must for all newbies to Ruby;

Ruby Koans – a set of exercises where you have to write code that passes the tests;

Confident Ruby – a relatively new book about writing an awesome clear code in Ruby;

Codemy – free English screencasts about Ruby and Rails;

RubyTapas – English screencasts about Ruby, some episodes come free of charge;

CodeSchool – online courses that teach Ruby and a number of other technologies. Free of charge to follow but after a certain stage you would certainly have to pay.

What is it?

Gems is the name of Ruby libraries. Bundler is a gem to handle dependencies between different gems in Ruby projects.

What is it for?

Any library in programming is used to not solve problems that have already been solved many times by other programmers. Sooner or later, a programmer realizes that his code may be useful to other people, and he goes on to put this code into a library and give free access to it to everyone who may need it. Then other programmers can improve the library's code and thus, over time, the community of people using this pro-

gramming language will obtain a reliable, time-tested solution to a specific problem. This process reveals a great power of open source projects.

Open source Ruby Community is one of the most vibrant communities, it has created so many libraries — gems — that sometimes writing an application is as easy as "collect them all" game.

Since there are very many gems available for a variety of tasks, it is not uncommon situation when one gem depends on another, and even on its particular version. Furthermore, your application may require only one specific version of gem and your task is to make sure that you are using the version you need. For such cases people invented a gem called Bundler that manages dependencies for gems.

Basics

Let me start with a tip: try not to use gems too often, at the beginning. Writing a code should never turn into a magic process of using ready-made solutions. First try to write a solution yourself, then, after it starts working OK, check out how others have solved the problem using gems.

There is nothing in particular to advise on Bundler. For each Ruby application you create a *Gemfile* where you specify the gems necessary for the project and their version, and use **`bundle install`** command to install them. Then a *Gemfile.lock* file is generated, showing all the gems used with their versions. Why do you need two files? That's simple: if you, for example, mentioned a gem without its version, the first installation of gems will use the latest version, and the second one will take the version downloaded last time, without automatic update, and thereby reduce the chances of adding a library incompatible with your code.

Using Gemfile in Rails does not cause any difficulties at all. For non-rails Ruby projects you will have to make a little more effort, but, again, there is nothing too complicated.

Resources

Gemfile – official Bundler documentation, the best source of information about Gemfile and handling dependencies between gems.

What is it?

They are managers for Ruby versions.

What is it for?

It is very unlikely that a developer easily makes do with one installed version of Ruby, especially considering that new versions are released regularly. One project may rely on the most recent Ruby 2.1, while the other may still work only with Ruby 1.9.3. Both versions should be available on the developer's machine, and make sure you update all the gems for both versions every time you need them (some gems might also work well only with a specific version of Ruby).

Without RVM (that is more popular) or rbenv you'll have a lot of trouble using multiple versions of Ruby simultaneously, and, to be quite honest, it makes no sense. With these tools all this becomes a matter of one console command.

Basics

I should have probably mentioned this in the beginning, but do get either RVM or rbenv as quickly as possible and use them while developing your apps.

What is it?

The best framework to develop web applications.

What is it for?

There are two options for developing web-applications: with and without a framework. In some cases, it makes sense to build an app from scratch, but most often your application needs a certain structure and a set of ready-made solutions for most common web development tasks. This is what you get when you use Ruby on Rails — a structure and a wide range of ready to use solutions.

If you want examples from real life, imagine that you are going to build a house. You would hardly have an idea to get a hammer and shovel and go build a house on a vacant lot nearby. For sure, it's better to follow standard practices: start with the foundation, make walls vertical, and ceilings horizontal. Many people before you have built millions of houses and wrote thousands of manuals on how to build them properly.

And that's the same way with Rails. It directly tells you how to handle forms, how the links in the browser are related to the internal code, how to connect this code to your database, how to scatter files in folders, how to generate HTML pages and so on. You just need to follow the conventions to get a well-organized application, open and comprehensible to any other developer in this framework. Moreover, by following these conventions you manage to develop web applications much faster than without using Rails or by using frameworks in other languages. Ruby on Rails is Apple of web development world, a system in itself, with its own rules, atmosphere and a categorical set of opinions.

Basics

It's not easy to start developing applications in Ruby on Rails. And it's not just about the complexity of the framework — on the contrary, it was created to remove all pain and unnecessary movements from the web development process. The problem is the barriers to entry: apart from the framework itself, you need to know an impressively wide range of web technologies: from HTML, CSS, JavaScript, Ruby, Git, SQL to SCSS, CoffeeScript, NoSQL, and third-party APIs.

I am talking about this not in order to intimidate you. On the contrary, I want to once again remind you that the purpose of this book is to help you get started, and not to be scared of all these rampant technologies, to provide an overview of all necessary tools and essential minimum of knowledge being sufficient to start developing applications. You have already made it halfway through the book, there are only four unknown tools that you may need. In fact, pretty soon I'll give you a detailed action plan, following which you simply can't help learning how to develop web applications.

As for Ruby on Rails, here again I am going to give you just a few tips that can really help in understanding of what is happening within this framework.

Firstly, always look back at your logs. In any unclear situation, with any weird error, look at the logs. Pay close attention to what parameters you pass to the controller, and what it does after the receipt of those parameters. In fact, Rails only takes parameters from HTTP-requests, passes them to the controller specified by a given link, and then returns some kind of data (for example, another application page).

Secondly, read the official documentation carefully. Very carefully!!! There are technologies that come with some obscure and somewhat cryptic documentation (e.g., Angular.js). Luckily, Rails has no such problem. Actually you won't even need any books and/or guidance, such well-written documentation RoR has. My students keep making the same mistake: they don't take the time to gain a grasp of Rails docs. Sometimes I end up copying and pasting direct answers to their questions.

Resources

In addition to the links below, check out the resources from the chapter about Ruby, you'll find some excellent self-education books on Rails.

Ruby on Rails Guides – official Ruby on Rails documentation;

Rails Tutorial – one of the most popular books on Rails, written by Michael Hart and translated into Russian;

Railscasts – iconic (in certain communities) Ruby on Rails screencasts. If you experience any problem, it is very likely to have been solved in Railscasts. Unfortunately, the project had been abandoned by the author, but its 417 issues are still available online and are highly recommended for the review (as and when necessary);

Rails Best Practices – he most popular collection of Best Practices in Rails. This resource is a must for all beginners;

PHP: a fractal of bad design – an article for those of you who are confused about choosing between Rails and PHP. After reading this article the choice will be obvious;

Upcase – a checklist of the necessary knowledge made by Thoughtbot, a company being well known in the Rails community

15

TESTING

What is it?

It is a set of methods to verify the accuracy of the written code.

What is it for?

The larger the project, the more difficult it is to make sure that all parts of the system are working, and that they are working correctly. The number of features increases and so does the number of bugs. In compiled programming languages, a part of the problems is solved by the compiler: if something goes wrong you will most likely learn about the problem during the compilation. In interpreted languages, you should always double-check everything you do.

To be on the safe side and to boost your confidence in consistency of the whole application, you need tests. Tests are files with ‘if I do A, the result should be B’ type code. Surely, nobody writes tests without using special libraries (in the Ruby world, these are RSpec, MiniTest) that largely turn writing tests into an organized and easy process. But one shouldn't forget that, at the most basic level tests are just a set of code pieces checking that ‘if A, then B’ holds true.

Basics

If you have just started to study programming or Rails in particular, it's too early for you to use tests. It is very likely that experienced developers who may be reading this book (what for?) would not tolerate such statements and would accuse me of heresy. It is true that in the Ruby world tests are sacred and form an integral part of the program writing process.

Nevertheless, I think that there is no point in writing tests before you learn to write something useful. You just won’t understand their value and will find yourself doing things that are pretty useless given your current level of skills. As your web application grows bigger and you incidentally bump into some parts that have suddenly stopped working, then (and only then) you will understand that it would be nice to have some tool to help you make sure that those previously broken parts are still working properly. It is at that moment when it makes sense to write your first test.

I would give you another tip: go easy on your tests. 100% coverage with tests (when absolutely each line of the code is checked) is a good but somewhat utopic goal that has nothing to do with reality. Sure enough, many things does not require test at all. Alas, at the beginning it is difficult to sort out what needs to be covered by the tests, and what does not. If you are new to this and it's an extremely hard job, I know. In due time you will get the grip of what you should do but before that I'll come up with some more insights that work for me:

1. If you have come across a bug in the application then, apart from repair, be sure to write a test checking that the bug no longer exists.

2. Focus on the tests checking the work of the whole system (in RSpec these are so called feature specs). The most important thing to be checked by tests is that

your end user faces no problems, even with most complex scenarios of the application use.

Resources

TDD is dead. Long live testing. – the father of Rails has stirred up much controversy by claiming that the development through testing is dead.

The best thing is to learn how to write tests within some of the existing open source projects. See how gems and applications available on GitHub are being tested. Take Spree source code, CMS for web stores on Rails as an example. Figuring that out is not the easiest thing in the world but it is well worth the effort.

What is it?

It is a relational database management system.

What is it for?

To store a major part of the important data in your application.

Basics

Perhaps I should have named this chapter “Database Management Systems” to avoid imposing my personal preferences so much. Of course, one may have the question ‘why not MySQL?’ which I will not answer here. However, in the resources section I will provide links to excellent material which explains, with reason, why MySQL should never be used in new projects.

Besides, I purposefully make no mention of non-relational DBMS, the notorious NoSQL. However among the resources given below you will also find a couple of excellent articles on this subject.

The data in relational DB management systems are represented in the form of tables with each column having a data type (line, number, date — the types of data vary from one DBMS to another). They are called relational because they provide an easy and clear way to show relations between the tables. For example, “Book” table has “Name of Author” column in which the number of a line of “Authors” table is stored. Thus, it is very easy to find information about the book author by means of a simple SQL query. Of course, if we assume that the book may have several authors, the task becomes a bit more complicated. Strange as it may seem, the official documentation on associations between models in Rails may serve as very good reference material on links between the tables.

You need to know at least two things: how to write SQL queries, and how to create indexes. For me writing SQL queries presupposes an understanding of how the interrelations between tables in a database are arranged, and how to use JOIN and other SQL structures. As for indexes, it is necessary to have an idea of what they are, otherwise you may very soon face performance problems.

To start using Rails you don't need to write SQL queries yourself. But this is a very useful and important skill. Keep in mind that a correct SQL query will always cope with the task of complex search of data better and faster that a Ruby-based code executing a huge array of data.

Resources

Do not pass this way again – a superb article explaining, in simple and clear language, why you should never use MySQL;

Why you should never use MongoDB – another detailed and interesting article this time telling why you should never use MongoDB;

PostgreSQL awesomeness for Rails developers – now a few things about what is so good in PostgreSQL, especially for Rails developers;

Postgres Guide – a guide for newbies both to PostgreSQL and SQL in general.;

How Databases Work – incredible article explaining how databases work;

Intro To SQL – free interactive course on basics of SQL; must do if you are unfamiliar with SQL.

What is it?

It is a process of putting the web application on the server.

What is it for?

When you finish developing your web application you wish to share it with your friends, to begin attracting users, and to earn your first millions in cash. So now it's time to deploy.

Basics

As it often happens in the world of Ruby/Rails development, proper and useful tools to deploy applications have already been created for you. Having acquired, in its third version, one of the best documentation and its own website, Capistrano is probably the best choice.

In addition to the ability to write Capistrano recipes for deploy (there is nothing difficult in that, there are plenty of ready-to-use samples on the web), you will need to learn how to configure VPS. Apart from Ruby, you will need to put on the server such things as a web server (Nginx), DBMS (PostgreSQL), application server (Unicorn), and a bunch of system packages to get all these things working together. Moreover, you will need to know how to manage users' rights in Linux. This is what the security of your application largely depends on, and there is a good reason I said that you need to learn it in the chapter about UNIX-like systems. The point is that each component of the system should be managed on behalf of an individual user, and have no chance to affect other components of the system. Thus, if your web server gets hacked the database will remain intact.

The deploy process goes roughly as follows:

1. A new code version (use a git repository) is downloaded to the server;
2. On the server you install your gems, compile static files, and some other things you may need, it depends on what you specify in your code for the deploy;
3. After that the application is launched by the specified command e.g. by the command to launch Unicorn.

Communication with the server is usually maintained by means of login/password or SSH. For reasons of safety it is preferable to use SSH. It is based on the following principle: two files with keys (a whole lot of odd symbols and letters encrypted in a special way), a public and a private one, are kept on your computer, and the list of permitted public keys is kept on the server. When you try to reach the server, it verifies whether your public key is on the list of permitted ones, and then the private key is checked to make sure that it belongs to you. Thus, only the people or devices whose keys have been added on the server have access to it.

First off, you don't need to look at all nuances of deploy process and configuration of servers, you can simply follow the detailed guides. Believe me, there are so many things about this part of development that they would take several years to learn. But our idea is to start with some small steps, to learn the necessary minimum, and then to constantly improve our skills. Remember: the more time passes between the start of your training and the time you get to do something that feels true to life, the bigger are the chances you will exhaust your initial enthusiasm and fall into procrastination.

Resources

Heroku – the simplest way to deploy applications, and a free one (up to a certain time). The whole application deploy process is about the use of **`git push`** command, and no additional knowledge is required;

Digital Ocean – one of the cheapest and most convenient VPS hostings;

AWS Free Tier – here Amazon provides free limited access to its web services during a year;

Capistrano – the most popular tool to deploy application. In its last versions it has acquired many new functions but it is still fairly easy to use.

18

THE ONLY WAY TO LEARN HOW TO DEVELOP WEB APPLICATIONS

Now, when you know all you need to start developing web applications, there is just one question left: how to get the self-education process going?

I believe that the only way to learn how to develop web applications is to be engaged in the web application development.

At the beginning of my career, I tried to follow everything written in the book called “Agile Web Development with Rails”, and I built the likes of those web stores all the readers of this book may have built in their time. I can't remember a single thing from that book, and it didn't help me to solve any interesting problem. Worse yet, I had nothing to show to the world either. It was a waste of time, just like any other applications written by following examples from a book.

A couple of months after reading that book I made another attempt, this time only relying on myself: I started coding an engine for my friend to manage a detailed encyclopedia of rock music. This appeared to be more useful experience for two reasons.

The first one: I was building something on my own, newly from scratch. There was no detailed development plan for any particular boring sample application

.

The second one: this application has posed a wide range of challenges that are not included in the books. I had to use Google a lot, tried different approaches, and wrestled with the problems on my own. Moreover, it was then when I really started communicating with other people in English. The Russian community was in its nascent stage, while on the official IRC channel of Rails there were dozens of people from all over the world ready to help.

Then I went on to develop another application, now just for myself. It was the manager of personal finance FJMoney. Having written it then, in 2010, I still use it every day, and keep developing it for my own needs. Please don't Google it and don't use it, because in that case I will have to support it.

Another thing to do to learn how to develop web applications is to find a mentor. Try to find a person who is a pro and has a considerable experience in web development. Try to offer him or her a mutually beneficial cooperation. For example, you may help your mentor with his/her projects in exchange for assistance in learning web development.

I myself regularly take up to 10 students per month to train and help them find the right way. Some of them have already found a job and launched their projects, and others have helped me greatly in development of my own services. At a certain moment, I got so involved in teaching that and mentoring that I decided to launch mkdev.me project in June 2014. I won't go into too much detail, but I hope you would click the link and see how we make online self-education better and more efficient.

Today, there is no shortage in web development courses but they all usually cost a pretty penny for a short period of time. What you need is your personal coach, your

Master Yoda who will always push you in the right direction, help you to find errors in your code, and answer all tricky questions or show you where to look for answers.

Try to post announcements on various forums and mail groups.There is a constant demand for assistants or junior developers. Your skills may be of some use there.

However, there is just a slight hitch: mentors aren't normally interested in working with complete beginners, particularly because beginners don’t need a mentor as everything they need to know at that stage they can easily learn by themselves. Therefore, try to get as far as possible on your own—to do it you only need some motivation and free time. As soon as you are ready, find a person who will help you to make your first money developing web applications.

I've taken a leap forward in my personal self-education thanks to my mentor, a great man for whom I had been working for about a year and who has actually taught me how to work and get new information.

Thus, having used my own ideas, writing real applications rather than useless samples, with the help of an experienced and seasoned developer, I have become Ruby on Rails web developer, and got sufficient skills to land my first full-time job, after which I moved to Berlin.

This is exactly what makes me believe that the best way to learn how to develop web (and any other) applications is the development per se. Don’t waste your time on boring tutorials. I am sure you already have or will easily find a couple of ideas for some small but cool applications. Why don't you go and try to develop an app of your own? Don’t be afraid of any difficulties. The web development training process should be fun and useful in real life. There is no other way to learn it. You don’t have to wait five years to start doing something real. Go for it.

BONUS #1

ACTION PLAN

Development of web applications is a multilayer process. You have a whole zoo of technologies and programming languages at your disposal, and understanding the way they all are linked together is one of most important tasks for anyone who pursues his or her self-education in this field.

I hope that after reading this guide you have got an idea of what it takes to get things done and in what direction you want to move.

And now, I take the liberty to offer anyone who is still tormented by the question "where to start?" a detailed action plan following which you will be able to develop your first application and to master (in due time) everything you need for develop-

ment of web applications. As always, I omit the details that you have to learn on your own. Let's roll:

1. Come up with an idea of your app.

Jot down several ideas for web applications. Choose the idea that that inspires you the most. It is desirable that this should be an application which you will regularly use yourself, after you develop it.

2. Install Linux or buy a Mac

There are no other alternatives here. See Chapter 4.

3. Sketch layouts of the principal pages

You don't have to be a designer. Just take a pencil and an A4 sheet and draw what your application will look like.

4. Download Twitter Bootstrap and use it to build your main pages

No Ruby, no Rails. Learn how to make simple web pages using a front-end framework. You need to be able to click the links and see various parts of the application.

4.1. Install git to track the history of changes

While building the pages learn how to use git. You may need to use Try-Git first. Write meaningful comments on your commits. Once you have created a page, make a commit. Another page — another commit. Had you CSS corrected — one more commit.

5. Configure Ruby, Rails and PostgreSQL and generate a new application

Everything is simple: install RVM and then use it to get the latest version of Ruby and Rails. Then create a new Rails application with git repository in it.

6. Decide on the entities in the application

Determine the necessary data to be used. Draw a small scheme showing interrelations between different elements of data (the guide on associations in Rails may be of some help, see resources to Chapter 15). Simply write a list of the necessary Rails models. For a blog, these will include Post, Category, User, Tag. For a finance management system — Transaction, Account. Don't try to grasp everything at the moment, you only need the entities without which it is impossible to built your application. Then think of the fields for these models.

7. Do it!

All you have to do now is to develop your web application using Rails. The resources from the previous chapters contain a whole lot of information and examples. And this is just a small portion of the things you will be able to find on your own. Don’t think of tests now, try to make a working application as soon as possible. Look at the existing open source applications to understand how everything should be written.

8. Show your application to the world

See the chapter on deploy. Start with Heroku to deploy the prototype. Be sure to learn how to deploy on VPS. No need to pay for using a real server, you may learn technologies like Vagrant and virtual machines, and deploy your app to the virtual machine working on your computer.

9. Prepare your CV and start looking for a job

As soon as you have built your first application, share it on GitHub and write a CV specifying all the technologies that you have become familiar with, and include a link to your code. Then start looking for a job, send your CV to as many companies as you can to get invited to an interview. I am sure you will be able to find a junior developer job. Good luck!

10. Be ready to learn

Now let me be brutally honest with you: if you start the self education of a web developer, you will never finish it. There will always be too many things to learn. Technologies will become obsolete and replaced by the new ones. New databases, new languages and new versions of old languages, new tools will appear. And you will have to keep your knowledge up to date.

There are also complete fields of development not cover in this book and that you will encounter for sure. After learning how to develop web applications you might decided to switch to Big Data field, or DevOps field, or mobile apps field. And then there will be a whole new world to absorb.

Good thing is, the more you learn the easier it will be to learn more.

BONUS #2

ADDITIONAL RESOURCES

Below are some more links you may find useful.

Teach yourself to code – a selection of links to the best guides and training articles on programming;

Hacker Newsletter – a weekly newsletter containing some of the best articles on startups and programming;

Ruby Weekly, PostgreSQL Weekly и т.д. – a vast array of newsletters. Once a week you will receive a selection of the best articles;

10 Articles Every Programmer Must Read – ten articles every programmer must read;

Understanding Computation – aimed at everyone who has majored in subjects other than Computer Science and feel they lack some theoretical background. This ex-

cellent book gives detailed and clear explanations of some of the most difficult aspects of Computer Science;

Step-by-Step Guide to Building Your First Ruby Gem – this will help you to create your own gem;

A Day to Love Postgres – another article explaining how cool PostgreSQL is;

Must Have Gems for Development Machine in Ruby on Rails – a selection of gems to facilitate Ruby app development;

Sysadmin Casts – some brief and interesting screencasts for system administrators. The best thing about them is that they contain very useful information on UNIX tools that are essential to know;

Awesome Ruby – great links to different libraries, frameworks, and Ruby programs. This is where you can find many good libraries you may need in Ruby development;

Thoughtbot Trails – checklists of essential knowledge of various technologies from Thoughtbot, a company being well-known in Rails community.

BONUS #3

JS MVC

At some point you may decide to write a web application where everything works on AJAX requests, the page never fully reloads and the interface is similar to a full-fledged desktop application, rather than to a conventional website. You will try to do everything using simple JavaScript and dozens of jQuery plugins and pretty soon the code you have written will be impossible to support, read and edit. To avoid this peril, you need a tool that provides rigid code structure and agreements on writing the applications as well as ready solutions to some well-known issues (e.g. for two-way binding of code variable and a text on a page based on the value of that variable). This tool is JavaScript MVC framework (or MVC prototype).

It's very likely that you will not need to use JS framework in the coming months and/or years. The area of its application is rather specific. Even where the task is to

write a single page application, it doesn't always make sense to use a full-fledged JavaScript framework.

However, the popularity of such tools is higher now than ever and it is good at least to know about their existence. It is not possible to speak about all versions available, for this reason I will only tell you about two of them which I have used myself in real projects.

One of the best known frameworks is Backbone.js. This is arguably the simplest, most stable and most flexible one. It is not even a framework to all intents and purposes, as it doesn't impose any specific application structure, and because of that there exists a typical issue for JavaScript world: dozens of methods for creating the same thing, and none of them recognized as a standard one. Behind the simplicity of Backbone.js there is a hidden need to reinvent, for many tasks, the wheel which you will have to support afterwards. It is not the best solution if you want to get rid of headache while arranging a JavaScript code in a large application. I dare tell you that the time of Backbone.js is over. For small applications it is not better than bare JS, as for large applications it creates more problems than it solves.

At the moment of writing this book I give preference to Angular.js. In my opinion, Angular.js has great combinations of low barriers to entry and a rich set of functions.

It will take you about an hour to start writing small applications with it.

However, it will take you some months to study and use all of its features. I have not had so far any fronted task that I have not managed to solve using this framework.

I have used it for a cross-platform mobile application with a wide functionality, for small widgets and even for a video editor.

Nevertheless, Angular.js is not perfect. I think that its major disadvantages are horrible official documentation and at times excessive academicism. To understand better what I mean, try a Google search for "angularjs service vs factory vs provider" or read the article "You Have Ruined JavaScript". However, it is the best tool for solving a certain task, as compared with all others.

And this is the most important thing: to use JavaScript frameworks for solving a certain task. I conjure you not to ever try new frameworks just because they seem to be really cool. It is especially dangerous in JavaScript world since the number of technologies is increasing every day, and it is important to understand that every single of them was initially written to solve a specific problem but not as a silver bullet to eliminate all fronted problems. In addition, several of them were written not for solving any problems but just for the reason that someone wanted to write one more framework.

Choose your favorite one and learn to use it properly. Then use it until you feel that it is necessary to change something about it.

Teamwork shows itself in practice. Labor organization, task distribution etc. vary from company to company. However, there are some common things for many projects.

Firstly, in any team, a particular tool of task control is used. A physical board is sometimes used to draw columns, and small pieces of paper with task descriptions are stuck onto different columns. Much more often, this is a major web application helping to monitor the task progress status, discuss its details together in a team, and distribute these tasks among web developers. The following is far from being a full list of such tools (more popular ones):

1. PivotalTracker
2. Basecamp
3. Trello
4. Asana
5. Github Issues (it is in each project on Github)
6. Redmine

Apart from the task manager, communication software is often used among web developers. HipChat and Slack are most demonstrative examples.

Secondly, in successful large teams some version of Scrum (most popular) or Kanban (less popular) methodologies is used. Basically, it is a set of rules on the development process for new features, teamwork organization etc. As a matter of practice, every company has its own “scrum”, that is why it does not make any sense to go into details.

No one requires of you knowledge of the methodology from memory but sometimes employers may ask you whether you have an experience of working in a scrum team. Anyway, I advise you to search on the Internet for the words “scrum”, “agile”, “kanban”, to click on the links connected with these terms, and simply to know that such things exist and sooner or later you will have to work with them.