padrino

Matthias Guenther

padrino

Contents

1

Introduction

Why another book about how to develop an application in Rails? But hold, this book should give you a basic introduction how to develop a web application with Padrino[1]. Padrino is a "The Elgant Ruby Web Framework". Is is based on Rails[2] and has some really nice features which makes it easier to create web application with less code and more fun than to with it's father Rails. To say it with words of the Padrino webpage: "Padrino is a full-stack ruby framework built upon Sinatra[3]".

1.1 Planning the application

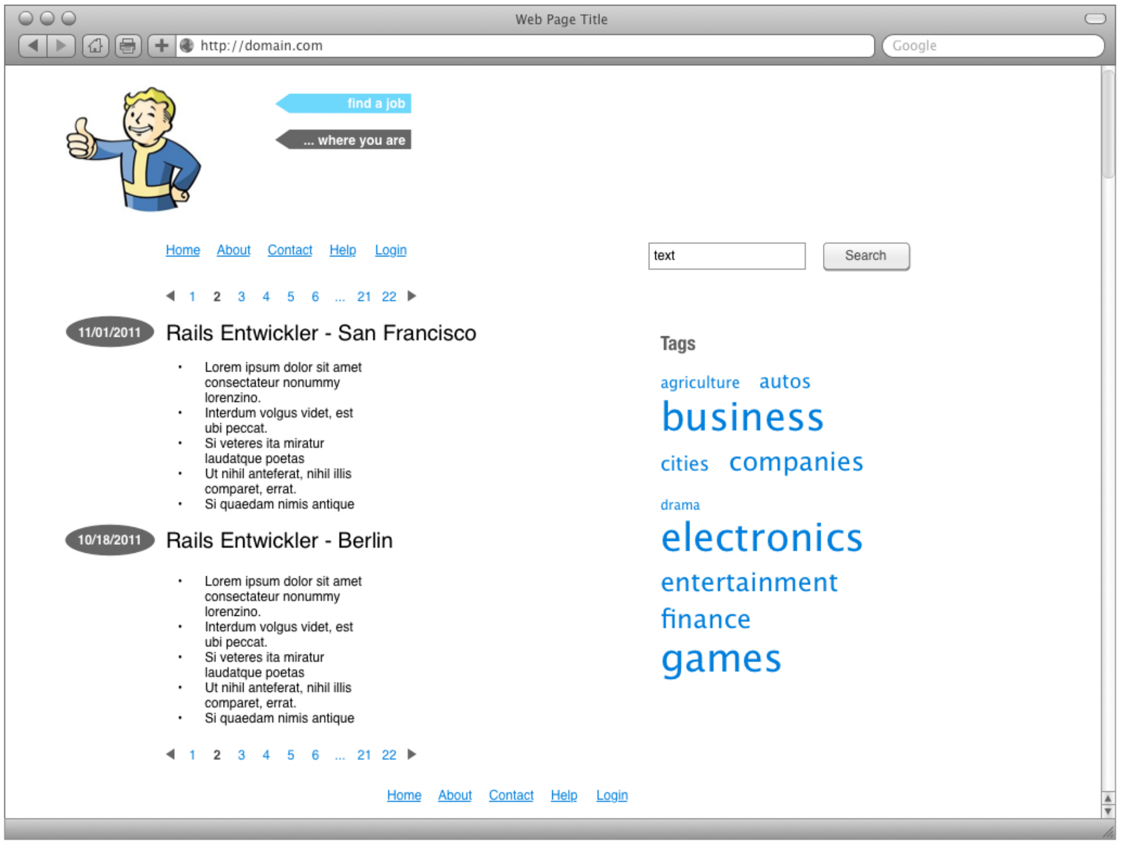
On the following image you can image you can see the basic image of our application[4]:

[1]http://www.padrinorb.com/

[2]http://rubyonrails.org/

[3]http://www.sinatrarb.com/

[4]You can use a classical stencil and paper to create mockups. But my handwriting is



11/01/2011 Rails Entwickler - San Francisco

- Lorem ipsum dolor sit amet consectateur nonummy lorenzino.
- Interdum volgus videt, est ubi peccat.
- Si veteres ita miratur laudatque poetas
- Ut nihil anteferat, nihil illis comparet, errat.
- Si quaedam nimis antique

10/18/2011 Rails Entwickler - Berlin

- Lorem ipsum dolor sit amet consectateur nonummy lorenzino.
- Interdum volgus videt, est ubi peccat.
- Si veteres ita miratur laudatque poetas
- Ut nihil anteferat, nihil illis comparet, errat.
- Si quaedam nimis antique

Figure 1-1. Start page of the application

so bad that I used Omnigraffle[5] with the stencil extensions by konigi[6] for writing wireframes.

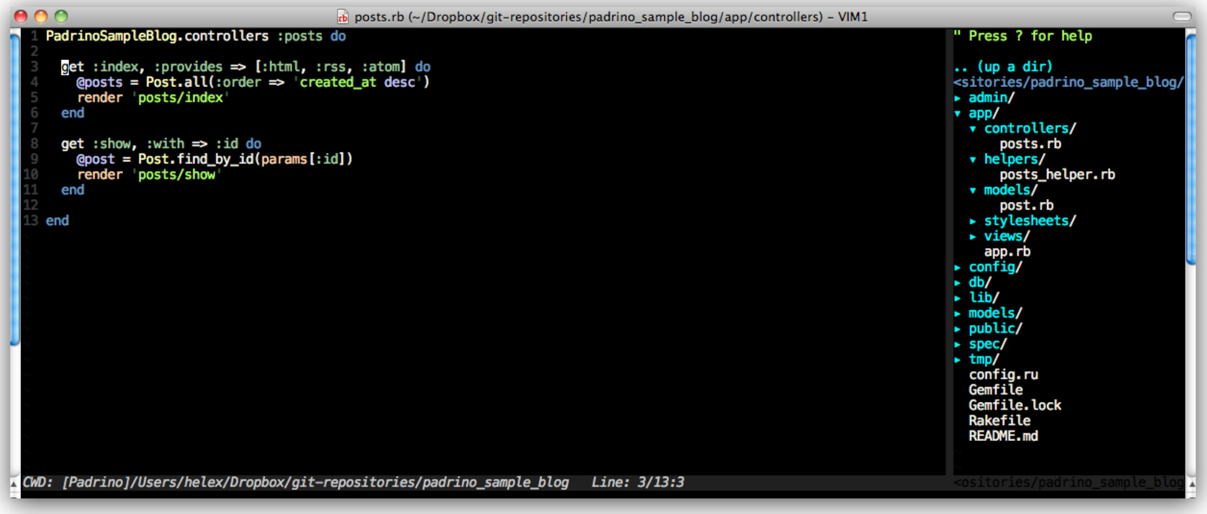
1.2 Your environment

Nowadays there are a bunch of IDEs out there: RubyMine by JetBrains[7] (commercial) [Aptana RadRails](http://www.aptana.com/products/radrails "Aptana RadRails") (free). Or you can switch to some text editors Textmate[8] (commercial for Mac only), Vim[9], and Emacs[10], which just run on

[5]http://www.omnigroup.com/products/omnigraffle/
[6]http://konigi.com/tools/omnigraffle-wireframe-stencils
[7]http://www.jetbrains.com/ruby/
[8]http://macromates.com/
[9]http://www.vim.org/
[10]http://www.gnu.org/s/emacs/

every server and under every terminal.

They all have there strength and weak sides. You have to experiment with them and take what fits your needs. Due to the fact that I will do a bunch of really great command-line, I prefer to use a "classical" **text editor** (see my vim-settings repository on github[11] if you want to see which tool I use every day). In the end you have to decide what you want to take. The main goal is that you are comfortable with it because you will mostly spend a lot of time with it.



```
posts.rb (~/Dropbox/git-repositories/padrino_sample_blog/app/controllers) - VIM1

PadrinoSampleBlog.controllers :posts do

  get :index, :provides => [:html, :rss, :atom] do
    @posts = Post.all(:order => 'created_at desc')
    render 'posts/index'
  end

  get :show, :with => :id do
    @post = Post.find_by_id(params[:id])
    render 'posts/show'
  end

end
```

```
" Press ? for help

.. (up a dir)
<sitories/padrino_sample_blog/
▸ admin/
▾ app/
  ▾ controllers/
      posts.rb
  ▾ helpers/
      posts_helper.rb
  ▾ models/
      post.rb
  ▸ stylesheets/
  ▸ views/
    app.rb
▸ config/
▸ db/
▸ lib/
▸ models/
▸ public/
▸ spec/
▸ tmp/
  config.ru
  Gemfile
  Gemfile.lock
  Rakefile
  README.md
```

```
CWD: [Padrino]/Users/helex/Dropbox/git-repositories/padrino_sample_blog    Line: 3/13:3
<ositories/padrino_sample_blog
```

Figure 1-2. Picture of Vim with NERDTree

1.3 Your browser

Here it is the as with the editors: There are many of them with great plugins out there for web development. The mostly used browsers by Rails developer are:

- Firefox[12]: Has a tons of plugins, is free, and with the magnificent Firebug[13] plugin, which let you inspect HTML document, measure the loading time of certain parts

[11]https://github.com/matthias-guenther/vim-settings
[12]http://www.mozilla.org/en-US/firefox/new/
[13]http://getfirebug.com/

- Chrome[14]: This browser is from Google (and I know many complains about data privacy). It feels very fast and is shipped with *inspect element*[15] to search the DOM[16]. It integrates the ability to debug JavaScript
- Opera[17]: Never used it, and don't know why I should (instead you tell my why and I will quote your words if they convince me)

There are more outside this main domain, like [Safari by Apple](http://www.apple.com/safari/ "Safari by Apple"), Lynx webbrowser[18] (quite an experience to use just a plain text browser) and the well known Internet Explorer[19]. Got out, grab the thing you want, and then gets your hands dirty.

1.4 Your system

"War… War never changes." this quote from my all-time favorite video game series Fallout[20] describes the battle of the different operating systems. I will briefly give you an overview what you can take

- Microsoft[21]: This commercial operating system is installed on a huge range of computers. Much software and games are available for this operation system. Updates comes regularly, and everything is mostly intuitive. One big problem of Windows is it's mouse affinity: There are shortcuts and handy tips available. It is good for development, but lacks some comfort and design issues (you may not know, when you haven't worked with Linux).
- Unix/Linux[22]: Highly configurable, Linux is shipped with a bunch of free software, and is Open Source. If you come from the Windows world, your brain will be burn like hell because everything is other. The program calls are other, you can install new software via *console*, the firewall is safe, and you can run most commands in the terminal (no annying mouse clicks anymore if you avoid graphical work). It is great for development because the great community are passionate about building software, and most software is Open Source so you don't have to pay for it.

[14]http://www.google.com/chrome

[15]This is like a Firebug pendant of Firefox.

[16]stands for *Document Object Model* and is a tree-like representation of the HTML page.

[17]http://www.opera.com/

[18]http://lynx.isc.org/

[19]http://windows.microsoft.com/en-US/internet-explorer/downloads/ie

[20]http://en.wikipedia.org/wiki/Fallout_(video_game)

[21]http://www.microsoft.com/

[22]http://en.wikipedia.org/wiki/Linux

- Mac[23]: The core under it's has Unix, so you can reuse your Linux knowledge. Many people say that products from Apple are very expensive. In the contrast they provide you with an extreme stable, high configurable, and reliable system, which has some really nice software for it. If you haven't paid for software in your life, some piece of software will make you want to pay for it because. In mine eyes, Mac is the gap between Windows and Linux: It can be changed in any any way you want (Linux) and has some really good software for it (Microsoft). Development on this machine is extreme good, because it underlines the needs of developers.

There is a war between these three operation systems, and which you chose is a matter of taste. As you can see on this pictures, I'm using a Mac. I like and love (and yeah, I had to pay a lot of patience and learning to come to the point where I can say, that it was worth paying much money for the hardware).

1.5 Other tools

This sections contains a list of several tools you need today for running the project.

Ruby

Padrino is implemented in Ruby. It is flexible, rich, and make it possible for you to turn all the crazy ideas you have from books, conferences, and blogs into reality. Most ruby programmers (like Dave Thomas (calld PragDave)[24], or Chad Fowler (the passionate programmer)[25]). This book will not give you an introduction to ruby. I recommend the following list of books (for newbies and veterans):

- Programming ruby[26]
- why's (poignant) Guide to Ruby[27] - written by the nebulous programmer why the lucky stiff[28] in a entertaining and educational way.

In this project I will explain you difficult language constructs in ruby - but don't assume that I will explain them in every way.

[23]http://www.apple.com/mac/
[24]http://pragdave.pragprog.com/
[25]http://chadfowler.com/
[26]http://pragprog.com/book/ruby3/programming-ruby-1-9
[27]http://www.scribd.com/doc/8545174/Whys-Poignant-Guide-to-Ruby
[28]http://en.wikipedia.org/wiki/Why_the_lucky_stiff

git - put your code under version control

git helps you to keep track of the changes in your code. You can switch between certain versions in your code, create branches to experiment with code, and easily manage your code in distributed teams

Recommended books:

- Pro Git[29] - this free online book explains you the basic internals of git, and how the workflow can be like
- gitref[30] - page with the basic commands you have to use to be productive when working with git

Recommended tools:

- gitk[31] - gives you a tree-like overview of your git repository (it works on every platform)
- gitx[32] - branching, merging, committing, you can even stage different changes in your code with this tool. It looks like a diamond and checking in your code will make you happy
- fugitive[33] - git wrapper for Vim, very good for command-line guys who don't want to leave the editor. Works good, if you are the only person working on a project (like I doing writing this book)

In this book I will explain the commands of *git* when they first occur, and repeat the commands every time I did during the development. Repeating is very important when learning something new.

heroku

The heroku cloud application platform[34] enables you to deploy your Rails application on the heroku platform. It manage the database creation, installation of the gems - difficult configurations tasks are handled by this platform. Heroku is so attractive that even the creator of ruby, Yukihiro Matsumoto[35], works as *Chief Architect of Ruby* on this platform.

[29]http://progit.org/
[30]http://gitref.org/
[31]http://gitk.sourceforge.net/
[32]http://gitx.frim.nl/
[33]https://github.com/tpope/vim-fugitive/
[34]http://www.heroku.com/
[35]http://blog.heroku.com/archives/2011/7/12/matz_joins_heroku/

Hello world

You know this sections from several tutorials which makes you comfortable with your first program in a new programming language. Get your hands dirty and start coding. First of all we need to install the gem with:

```
$ gem install padrino
```

This will install all necessary dependencies and makes you ready to create your web applications. Now we will generate a fresh new Padrino project:

```
$ padrino generate project hello-world
```

We will go through each part:

- `padrino generate` - tells Padrino to perform the generator with the specified options. The generate options can be used to create other *components* for your application like a mailing system or a nice admin panel to manage your database entries. A shortcut for generate is `g`
- `project` - says padrino to generate a new application.
- `hello-wolrd` - the name of the new application and this is also the folder name.

The console output should looks like the following:

```
  create
  create  .gitignore
  create  config.ru
  create  config/apps.rb
  create  config/boot.rb
  create  public/favicon.ico
  create  public/images
  create  public/javascripts
  create  public/stylesheets
  create  tmp
  create  .components
  create  app
  create  app/app.rb
  create  app/controllers
  create  app/helpers
  create  app/views
  create  app/views/layouts
  create  Gemfile
skipping  orm component...
skipping  test component...
skipping  mock component...
skipping  script component...
applying  haml (renderer)...
   apply  renderers/haml
  insert  Gemfile
skipping  stylesheet component...
identical  .components

=================================================================
brand_new is ready for development!
=================================================================
$ cd ./brand_new
$ bundle install
=================================================================
```

The last line in the console output tells you the next steps you have to perform. Going in the application folder and starting the application:

```
$ cd hello-world
$ bundle install
```

The command `bundle install` will install with the bundler[36] all the necessary gem dependencies for your project which are declared in your *GemFile*.

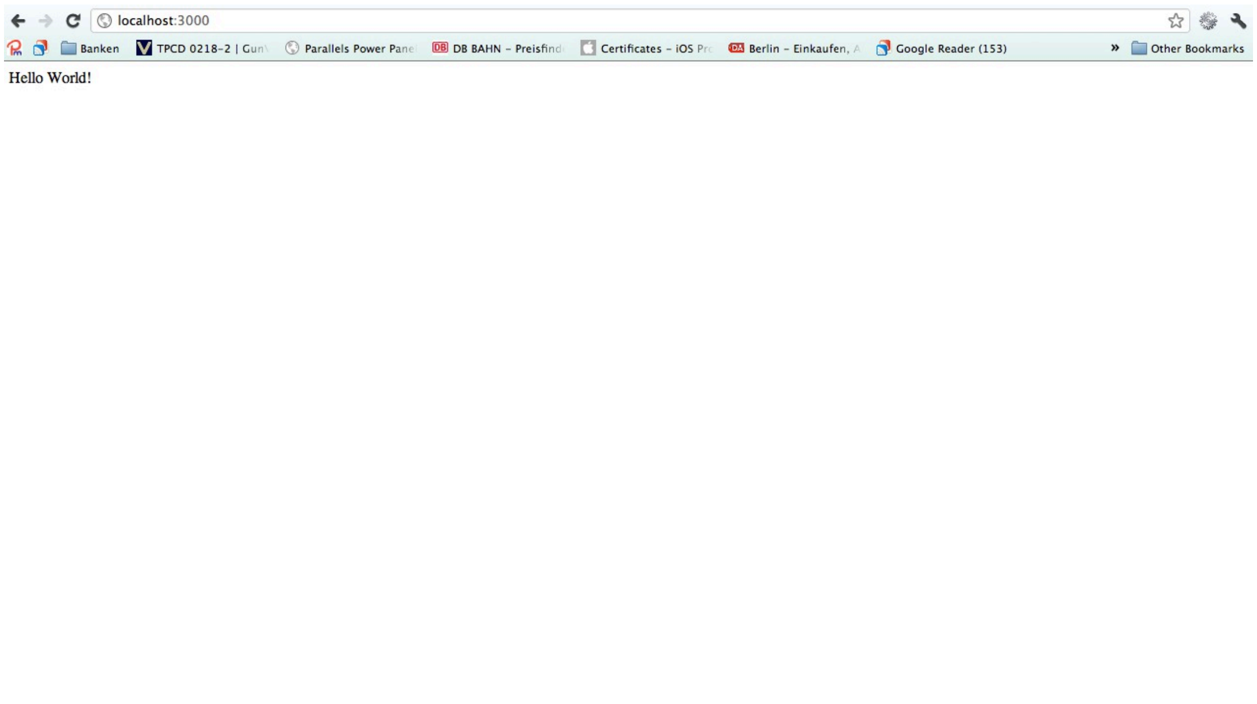
Let's open the file *app/app.rb* (this is like the root controller) and write in the following:

Now run:

```
$ padrino start
```

and fire up your browser with the URL *http://localhost:3000*. Be happy with the following pictures:

[36]http://gembundler.com/



Hello World!

Figure 1-3. Hello world in your browser

You can say, you have build your first Padrino application in less than five minutes.

Wait, there is more - the file structure

Navigating through the various parts of a project is essentials. Thus we will go through the basic file structure of the *hello-world* project:

```
|-- Gemfile
|-- app
|    |-- app.rb
|    |-- controllers
|    |-- helpers
|    `-- views
|        `-- layouts
|-- config
|    |-- apps.rb
|    `-- boot.rb
|-- config.ru
|-- public
|    |-- favicon.ico
|    |-- images
|   |-- javascripts
|   `-- stylesheets
`-- tmp
```

We will go through each part.

- **Gemfile**: The place where you put all the necessary *gems* for your project.
- **app**: Contains the "executable" files of your project with controllers, helpers, and views for displaying the contents of your application.
- **config**: General settings for the application, that means which hooks should be performed before or after the application is loaded, setting the environment (e.g. production, development, test), mounting other application within the existing application under different subdomains.
- **config.ru**: If you want to start Padrino from the console, this file tells to start the application from the command line.
- **public**: Place where you put global available files to be available for the public audience of your page like images folder, JavaScript files, or style sheets
- **tmp**: d'oh still don't know it

1.6 Git

You will ask: "Why the hack should I use version control?", and I will say: "Why should you want to shot a bullet through your head by yourself?". This metaphor is harsh but express exactly what I think - without version control you are lost. Git helps to detect changes in your code base, to better collaborate with other people, and to get out of the pit when you break something badly. You can experiment in *branches* to create running code, or to build a prototype to test design and technical limitation.

Except from Git, there are other version control system like cvs[37] (*concurrent version system*), svn[38] (*subversion*), or Mercurial[39] - just to name a few. Git has many topics, and this book should not focus how to apply every command of it in this project - I will just explain the things which are important for this Rails projects. I highly recommend Scott Chacon[40] book Pro git[41] book - it covers

[37]http://en.wikipedia.org/wiki/Concurrent_Versions_System

[38]http://en.wikipedia.org/wiki/Apache_Subversion

[39]http://mercurial.selenic.com/

[40]http://scottchacon.com/

[41]http://progit.org/book/

a lot of grass about git, and explaining its internals and fundamentals with beautiful images (like my images in this book created with omnigraffle).

Installation plus least configuration

We want to get the least possible thing to be working. Follow the installation section[42] of the **Pro git** book to install git on your local machine. local machine. After that you need to setup your email address (so that other can see who blames the last lines of code) and your user name:

```
$ git config --global user.name "wikimatze"
$ git config --global user.email "matthias.guenther@wikimatze.de"
```

It is possible to define git aliases[43] for comment commands so that you have to hack less chars. But this is the first time you are using git, repeating commands is the best way to learn, and everyone has a different opinion what aliases should be used, I will not create further configurations.

Get started with 'git init'

Git can keep track of every file you have in a certain directory. Let's assume that you have a brand new Padrino application in the directory *padrino_brand_new*. First you need to initialize git in the project folder:

```
$ cd ~/padrino_brand_new
$ git init
> Initialized empty Git repository in ~/padrino_brand_new/.git/
```

Now that you started a new repository and it is time to add new files. If you type in `git status` you will see a list of all *untracked*, *modified*, or *changed* files in your folder. Again, git gives you a nice console output what you can do in the next steps:

```
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
# .components
```

[42]http://progit.org/book/ch1-4.html
[43]http://gitready.com/intermediate/2009/02/06/helpful-command-aliases.html

```
# .gitignore
# Gemfile
# app/
# config.ru
# config/
# public/
nothing added to commit but untracked files present (use "git add" to track)
```

If you are new to git (and want to do the same work again and again), you have to type in *git add <filename* for each part of the files. But git is at least as smart as you are and has handy command to add all files simultaneously in each step:

```
$ git add .
```

The '.' stays for the current directory and git then knows that it should track all files and components (which even stay in subdirectories) recursively. During this stadium, we have just *staged* the files and not committed them in the repository. Let's do it:

```
$ git commit -m "initial commit (Padrino and git rocks)"
```

The '-m' options tells git to take a message for the commit. I'm a big fan of Tim Popes[44] Vim plugins. With the fugitive[45] he has written a fully integrated git into the Vim environment. According to Popes philosophy, a commit message should not be longer than 80 chars - anything that is longer is an indicator that you are writing to much, or have a to big commit with too much changes.

gitignore

This file let's you specify which files should not be put under versions control. It is useful to add in this file the names of files/directories which changes automatically like log files. When you start your Padrino application with `padrino start` it will start to track requests you can see in the console in the *log/* directory. You don't want to have this in your commit history, so you have to write in your *.gitignore* files

```
log/**/*
```

This tells git to ignore all changes in all subdirectories of the *log* directory. You are lucky to work with Padrino, and it will automatically a *.gitignore file* when you start a new project, with all entries you need:

[44] http://tpo.pe/
[45] https://github.com/tpope/vim-fugitive

```
.DS_Store
log/**/*
tmp/**/*
bin/*
vendor/gems/*
!vendor/gems/cache/
.sass-cache/*
```

Just keep your mind on committing small changes for your project.

1.7 Conclusion

We have covered a lot of stuff in this chapter: installing the Padrino gem, finding the right tools for the job, and get used to version control with git. Now it is time to jump into a real project!

2

Job Board Application

There are more IT jobs out there than people are available. It would be great if we have the possibility to offer a platform where users can easily post new jobs offer (of course ruby and rails jobs) to get enough people for your company.

This is the outline scenario - it's a real world example, so when your chef ask you if you know how to create such piece of software raise your hand and say that you have some bleeding-edge technology which fits perfect for the job.

The outline is easy, and as a developer you have implementation details with many good and cool features in your mind. But step back from the thoughts to get started with switching your fingertips on the keyboard. Clear your mind and take a piece of paper (or your favorite paint program) and outline a scheme of what you want to create. Make small steps, test your code, integrate your code often, and deploy your code. Don't worry, I will show you how you to create a healthy and motivating work flow with many neat and small hints. Sounds difficult in your ears? Well, actually it is, but just start, and don't let your mind gets filled with fears and issues to hinder your daily progress.

2.1 Creation of a user data model

There are many different ways how to develop a user for your system. In this way, we will go conform the standard of nearly each application. A user in our system will have an *unique* identification number **id** (useful for indexing and entry in our database), a **name** and an **email** each a type string.

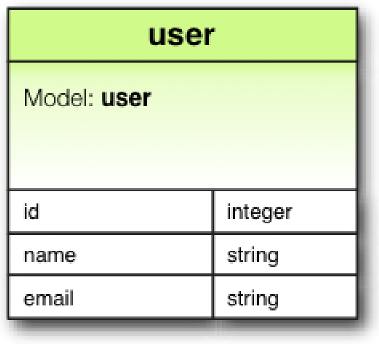


Figure 2-1. user data model

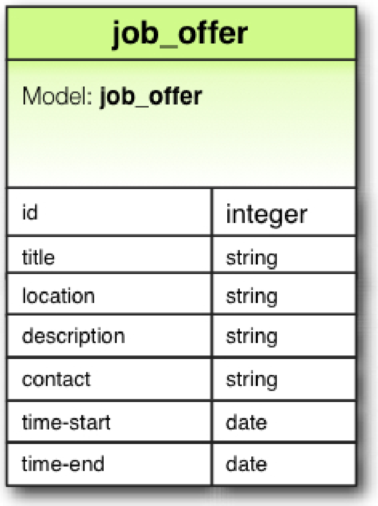
2.2 Creation of a job offer data model

I like the **K.I.S.S**[1] principle, so we will keep up the easy design. A job offer consists of the following attributes:

- title: the concrete
- location: where the will be places
- description: what is important
- contact: an email address is sufficant - they should call you instead of vice versa
- time-start: what is the earliest date when you can start
- time-end: nothing lives forever - even a brittle job offer

Under normal circumstances it would be nice to upload an image, but we will limit this opportunity to link to an existing image (most likely on flickr or some other image provider).

[1]Is an acronym for *Keep it simple and stupid.*



job_offer

Model: **job_offer**

id	integer
title	string
location	string
description	string
contact	string
time-start	date
time-end	date

Figure 2-2. job offer data model

2.3 Basic coding of the app

After making some thoughts about the data models of our application it is time to put our dream into reality.

In a first attempt we will start with generating a new project with the normal `padrino` command (see section \ref{section 'Hello world'}) but this time it has a bunch of new options:

```
$ cd ~/padrino_projects
$ padrino g project job_app -t rspec -a sqlite -e haml -c sass -s jquery
```

Explanation of the new fields:

- **g**: is shortcut for generate (who does not love shortcut to keep you save you keystrokes)
- **-t rspec**: using the RSpec[2] testing framework (a later explanation about this will follow)
- **-a sqlite**: specifying the orm[3] database adapter is sqlite[4] - easy to maintainer and easy to inspect because all entries are saved in a plain text file

[2]https://github.com/dchelimsky/rspec/wiki/get-in-touch

[3]stands for *object relational mapper*

[4]http://www.sqlite.org/

- **-e haml**: using Haml[5][6] markup as a *renderer* to describe HTML in better and faster way
- **-c sass**: using Sass[7][8] markup for describing the CSS[9] of the application
- **-s jquery**: defining the script library we are using - for this app will be using the famous jQuery[10] library (other possible libraries are)

If this commands works, you have a nice green playground with all the Next, we need to specify the used *gem* in the *Gemfile* with your favored text editor (cause I'm a big Vim fan boy `vim Gemfile`):

```
source :rubygems

# Project requirements
gem 'rake'
gem 'sinatra-flash', :require => 'sinatra/f

# Component requirements
gem 'sass'
gem 'haml'

# Test requirements
gem 'rspec', :group => "test"
gem 'rack-test', :require => "rack/test", :

# Padrino Stable Gem
gem 'padrino', '0.10.5'
```

We are using the last stable version of Padrino (during the release of this book it is version **0.10.5**). Let's include the gems for our project (later when *time has come*, we will add other gems) with bundler[11]:

```
$ bundle install
```

Recall from section (\ref{section 'git - put your code under version control'}) that we need to put our achievements under version control:

[5]http://haml-lang.com/
[6]stands for ??? (couldn't find it out)
[7]http://sass-lang.com/
[8]stands for *Syntactical Awesome Style Sheets*
[9]stands for *Cascading Style Sheets*
[10]http://jquery.com/
[11]recall that bundler is service to install all the required gems for a certain project

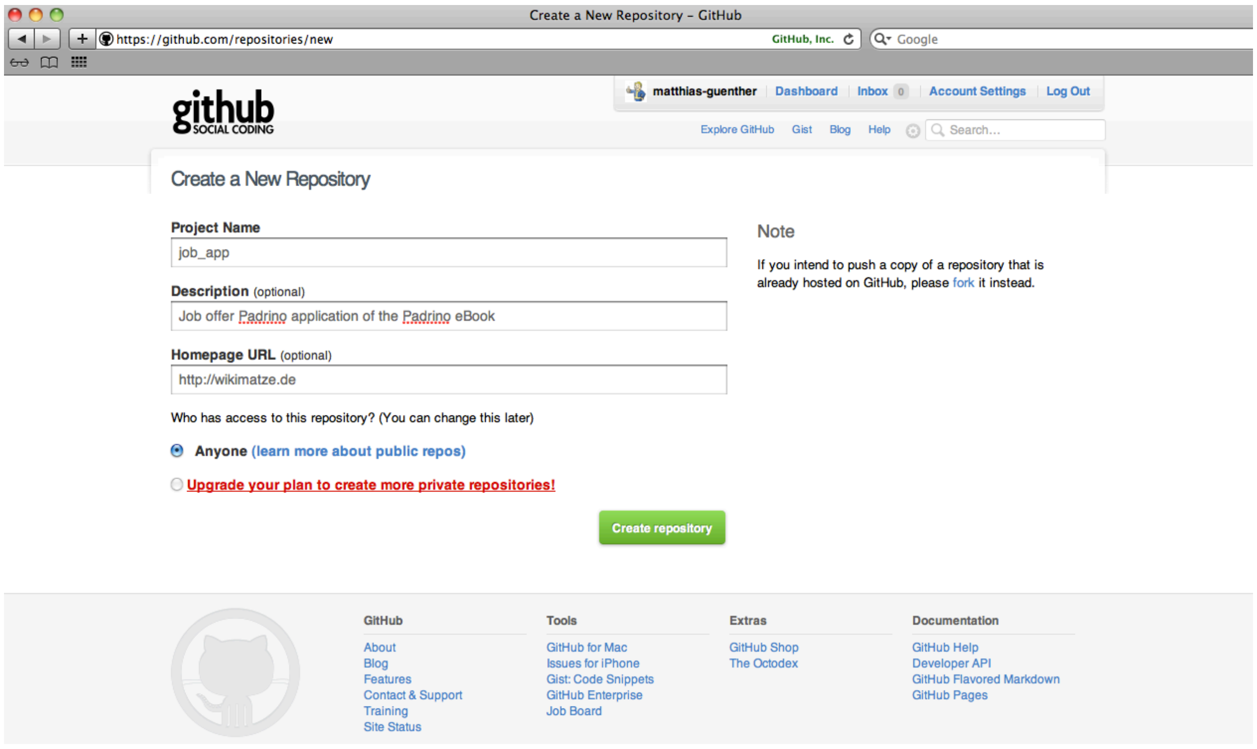
```
$ git init
$ git add .
$ git commit -m 'first commit of a marvelous padrino application'
```

Can you remember what the git commands are doing. If yes or no, just read the following explanation to refresh your memory:

- `git init` - initialize a new git repository
- `git add .` - add recursively all files to staging
- `git commit -m` - check in your changes in the repository

Because we are hosting our application on github[12] we need to push the project on the platform. (TODO: installation explanation of github, maybe just a link)

```
$ git remote add origin git@github.com:matthias-guenther/job_off_app.git
$ git push origin master
```



Create a New Repository – GitHub

https://github.com/repositories/new

matthias-guenther | Dashboard | Inbox 0 | Account Settings | Log Out

Explore GitHub Gist Blog Help Search...

Create a New Repository

Project Name: job_app

Description (optional): Job offer Padrino application of the Padrino eBook

Homepage URL (optional): http://wikimatze.de

Who has access to this repository? (You can change this later)

● Anyone (learn more about public repos)

○ Upgrade your plan to create more private repositories!

Create repository

Note

If you intend to push a copy of a repository that is already hosted on GitHub, please fork it instead.

Figure 2-3. creating a new project on github

12

Instead of *matthias-guenther* you have to replace this phrase with your personal github account name. That's it, now project is online and everyone can see it - even potential head-hunters which want to hire you. We want to give extra credit for reader so that they can see what this project is about. So we will add a README.md to the project

```
$ git add README.md
$ git commit -m 'add README'
$ git push
```

If you want to check how it has to be, just checkout the sources[13].

[13]https://github.com/matthias-guenther/job_app