Why Ruby on Rails?

Rails is a full-stack web framework, meaning that it comes with the capabilities to query the database, render templates, and exchange information with the web server.

Pros

- easy to get simple apps running quickly
- comes with all the basics you need: database management, templating, built in testing framework, nice MVC separation, asset precompilation
- active ruby community --> lots of useful gems (Ruby packages)

Cons

- Rails and its community are constantly evolving, so version incompatibility issues between gems are common.
- The official Rails and Ruby documentations are awful.
- Ruby is a messy, inconsistent language, and many inconsistencies carry over into the Rails mindset. (This is not a huge con, as building a Rails app does not proportionally necessitate that much Ruby coding.)
- Black magic. A lot of complexity is hidden from you. Again, this makes it easy to set up a simple app, but difficult when you want to get into the guts of the framework and customize advanced stuff, like how your cache and database work.
- Rails apps are difficult to scale. The built in database management is not very performant, and your app will be difficult to optimize if it grows substantially.

In essence, Rails is fantastic for building small apps that don't require particularly specialized technologies. This makes it perfect for things like 6.470 projects, personal portfolios, and small business websites. Rails would be a terrible choice for building a multimillion-user social network or a search engine. Use your judgment.

Setup

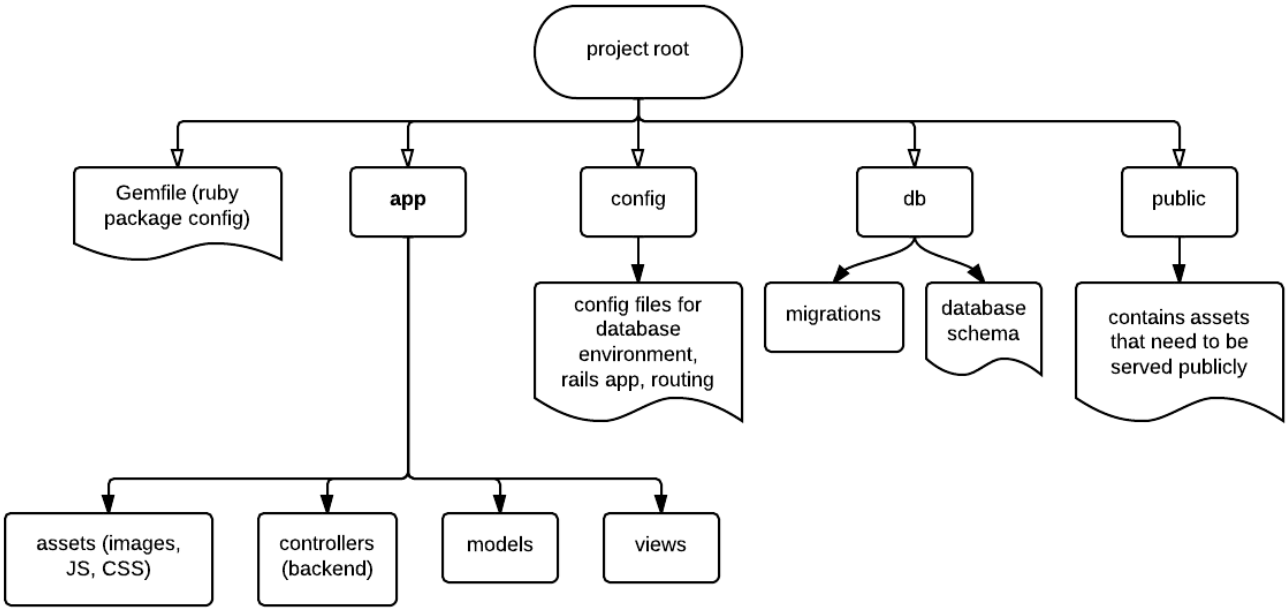
0. Get **rvm (ruby version manager)**

1. Download Ruby. To work with the latest version of Rails, you probably want 1.9.3. If you already have multiple versions of Ruby installed, you should put '*rvm use 1.9.3*' (or whatever version) in your bash profile.

2. Read/follow this guide until you have a website:
http://guides.rubyonrails.org/getting_started.html

help I lost my HTML

Rails can be confusing to use for the first time because everything is stored in a seemingly arbitrary but very particular place. Here is a cheat sheet for the files and folders you'll care about while getting started.



Rails tools

Scaffolding: use it to automatically generate templates in the right places when you want to create a new model/view/controller. It's very helpful when you're getting started and still figuring out where everything is. You'll probably never use it again after section 5 of the getting started guide.

Console/ActiveRecord: Run 'rails console' in a terminal to query ActiveRecord, which is Rails's ORM (Object-Relational Mapping) implementation. This means that it's a middleman between the backend and the database, which can still be whatever you want (MySQL, Postgres, sqlite, etc). It's basically a more readable way to interact with the database.

When you start setting up your database schema, make sure to have this guide on ActiveRecord *associations* (the Rails word for database relations) handy.

Rake: the Ruby analogy to Unix *make*. Commonly used to manipulate the database schema (*rake db:migrate*).

Asset pipeline: Rails magically preprocesses your assets (images, Javascript, CSS) and puts them in your public/assets folder. You shouldn't need to wrangle with it much; I'm just letting you know in case you run into issues with assets, because it's turned on by default as of Rails 3.1.

Quirks to watch out for

Functionally, Ruby is very similar to Python. However, the Ruby syntax and standard library naming scheme are notoriously inconsistent, and some Ruby library functions contain weird characters. Also, someone thought it was a good idea to delineate class and instance variables by prefixing the variable names with multiple @ symbols. Just keep the documentation on hand, and don't make assumptions.

When you define a model object, Rails will require its name to be pluralized in some places (such as when defining database relations), and not in others (such as when querying that model's table in the database). There isn't too much reason to it; you just have to follow the lead of whatever the Rails guide does in certain cases.