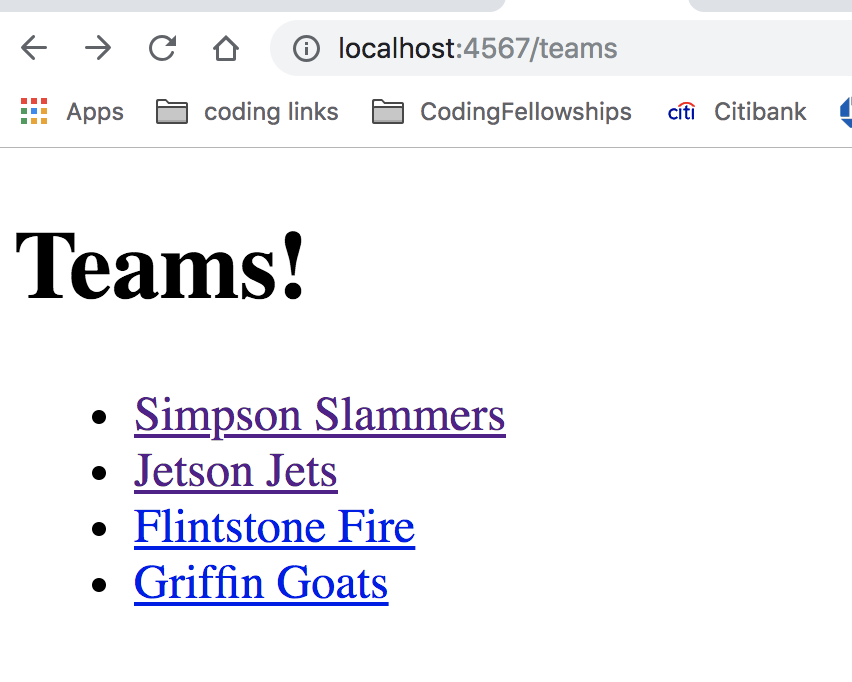
Sinatra is an open-source web app that allows you to make a ruby website easily. It’s a domain-specific language-which means, it’s especially made to serve a purpose within a particular stack or domain. That purpose is, to make website creation (and setting up that server-app connection) easy by setting up a series of template files in the particular order you need them in order to make magic happen.

I’ll run through a really simple Sinatra app that still uses most of the bells and whistles you’d ever need to throw your program up on the web. But first I’ll discuss the differences between the way Sinatra sets up your file structure, and just a plain ol’ app file structure that is not built for a browser, so you can see what Sinatra adds and why what it adds is important.

##Sinatra app file structure

So here is a very simple Sinatra app stripped down to bare bones, and I’ll tell you the 3 directories/files we need to worry about to get this thing running and those you can ignore In this tutorial.

This app helps you get the roster of a kickball team up on your browser and looks like this;



and it’s getting the information for this from the hash of data in our ‘leagues.rb’ file:

```

class League

ROSTER =

{

"Simpson Slammers" => {

"Catcher" => "Bart Simpson",

"Pitcher" => "Homer Simpson",

"1st Base" => "Marge Simpson",

"2nd Base" => "Lisa Simpson",

"3rd Base" => "Ned Flanders",

"Shortstop" => "Barney Grumble",

"Right Field" => "Monty Burns",

"Center Field" => "Waylon Smithers",

"Left Field" => "Clancy Wiggum"

},

"Jetson Jets" => {

"Catcher" => "George Jetson",

"Pitcher" => "Jane Jetson",

"1st Base" => "Judy Jetson",

"2nd Base" => "Elroy Jetson",

"3rd Base" => "Astro Jetson",

"Shortstop" => "Orbitty Jetson",

"Right Field" => "Cosmo Spacely",

"Center Field" => "Rosie Jetson",….etc, etc.

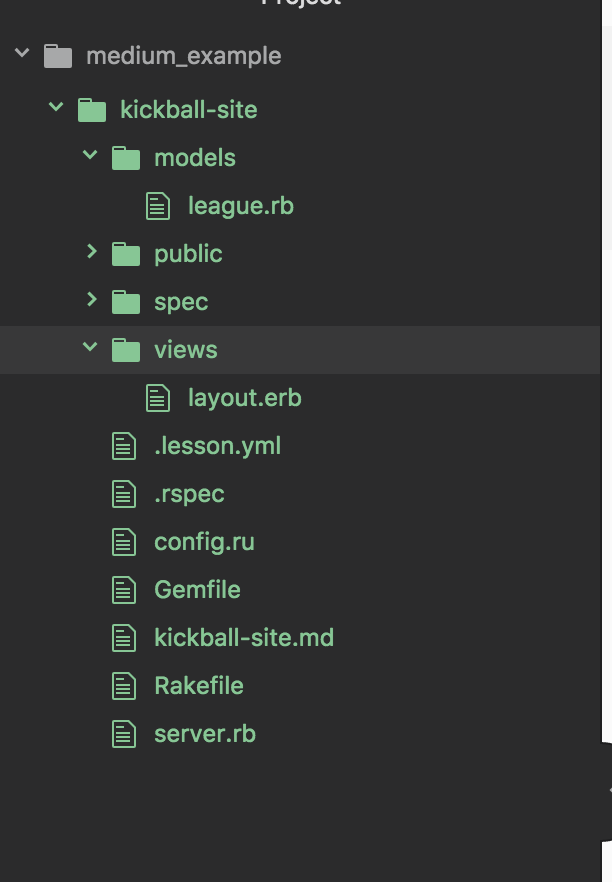
```

im only going to show you how to cycle through this file and throw the **names of the teams** up onto a webpage (forget the names of the players). **But let’s take a few steps back first and talk about the sinatra file structure so we can see how this fits into the larger picture.**

## Check out the bare-bones Sinatra file structure, identify what you need

1. Download my barebones sinatra app file structure from github (here)
2. Run ‘bundle install’ in terminal
3. Identify your model, view and controller files and only pay attention to those.

Open up the barebones app and here is what a Sinatra app basic file structure looks like:

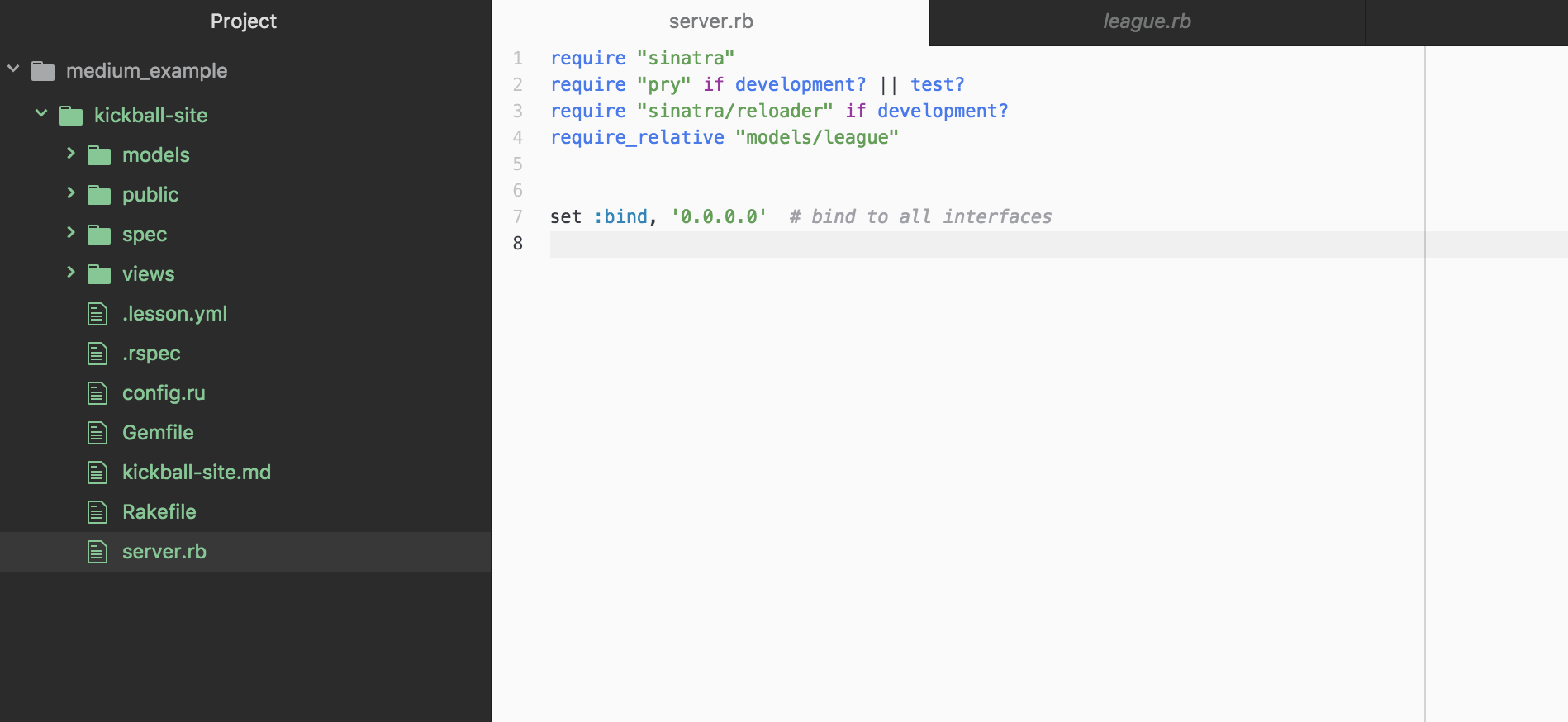


This is a good deal more complicated than the single folder with a ‘code.rb’ file from the snow shoveler example, isn’t it.

**For now, only worry about your models folder, your views folder** and your **server file** which is your **controller**. I’m going to discuss them each below.

**##The Controller - Server.rb**

The server file is translating between the **data** in your **models** and the **stuff people will see on the browser** in your **views**. That’s why this server file is called our ‘**controller**’.



```

require "sinatra"

require "pry" if development? || test?

require "sinatra/reloader" if development?

require\_relative "models/league"

set :bind, '0.0.0.0' # bind to all interfaces

```

lets look to see what else we may need to add to this boilerplate code to make a webpage.

‘Require Sinatra’ activates the Sinatra gem you installed. ‘Require pry’ installs the **pry library** which will enable **testing** if you are interested.

‘Sinatra reloader’ makes it so you don’t need to restart your server in terminal every time you make a change to the code- all you need to do is refresh your page in the browser.

And finally, our models are **‘required’**. Remember, our **models are our databases** . In this example you’ll see there is only one ‘model’ containing data; ‘league.rb’. If we had more data, we’d need to require “model/moredata.rb”- but we don’t. So we just have ‘model/league.rb’.

**##making a route in your server file (your controller);**

1. create a ‘/’ route for your homepage in your server
2. create ‘home.erb’ file in your views)

This is how you’d make yourself a homepage (or rather, this is the first step of many.)

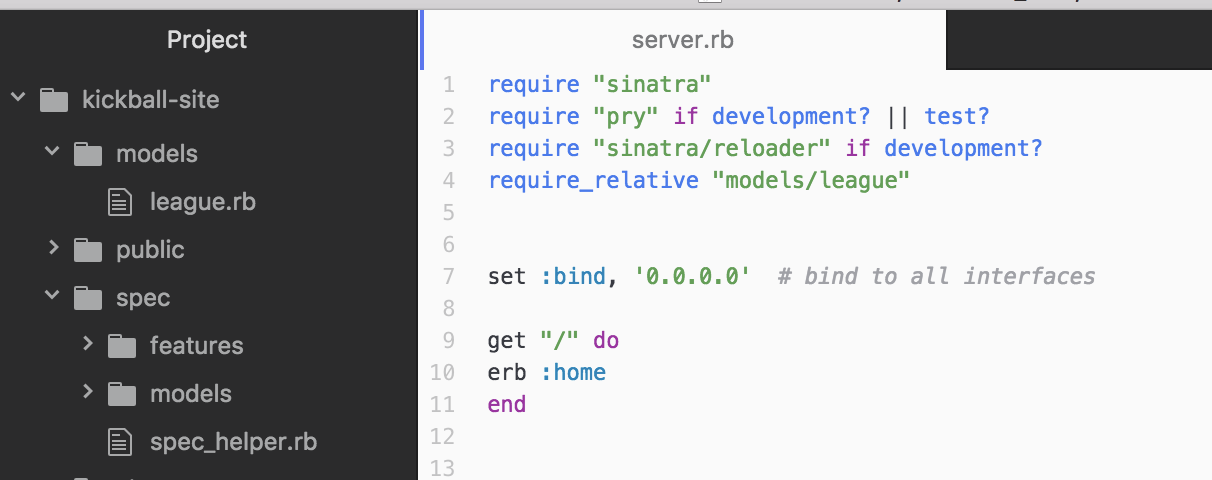
```

get ‘/’ do

erb :home

end

```



To explain what you’re seeing here, the ‘get’ is getting the url and erb :home **is rendering stuff from the corresponding ‘home.erb’ file you’ll need to make in your ‘views’ folder.**

##views folder- your ‘layout.erb’ and ‘home.erb’ files.

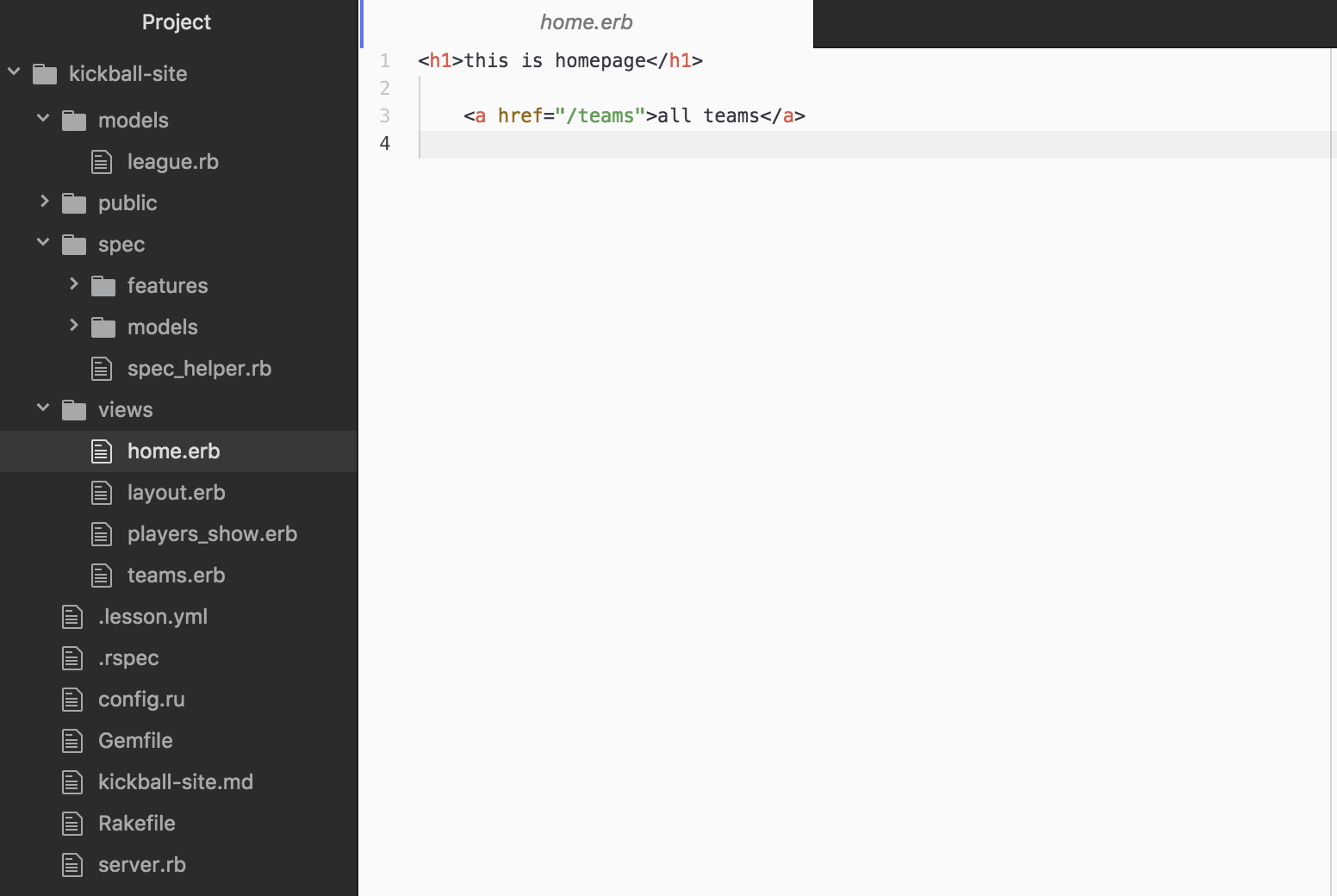
So create a file in your ‘views’ folder with a page called ‘home.erb’ and stick this in;

```

<h1>this is homepage</h1>

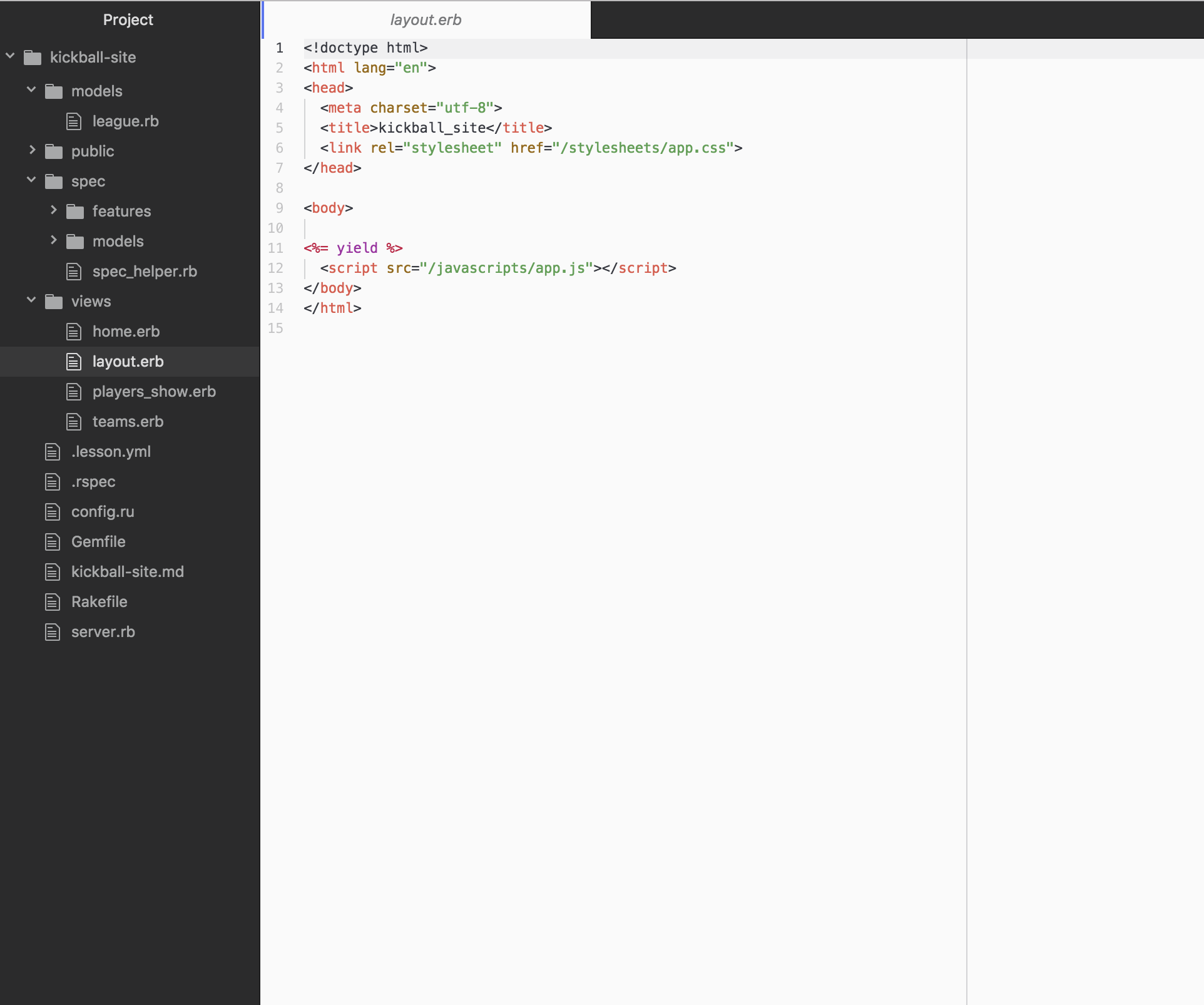
<a href="/teams">all teams</a>

```



Anytime you want to make another page for your site, and you need another url, you can set that url in your server.rb file, and then you need another corresponding erb file that you link in your server with ‘erb :another\_corresponding\_erb

Also, check out your ‘layout.erb’ file…



the html in your ‘home.erb’ file is actually what is showing up in the ‘layout.erb’ file where <%yield%> is. In Layout.erb, you’ll see some html-like stuff with a <%yield%> marker as a placeholder for where the body of your html will render. The ‘title’ in this layout file will actually be the title of your tab!

##**Models**

So we’ve talked about the views, and we’ve discussed the controller, and how you link those two up by setting a route in your server file and hooking It up to a corresponding erb file in your views folder.

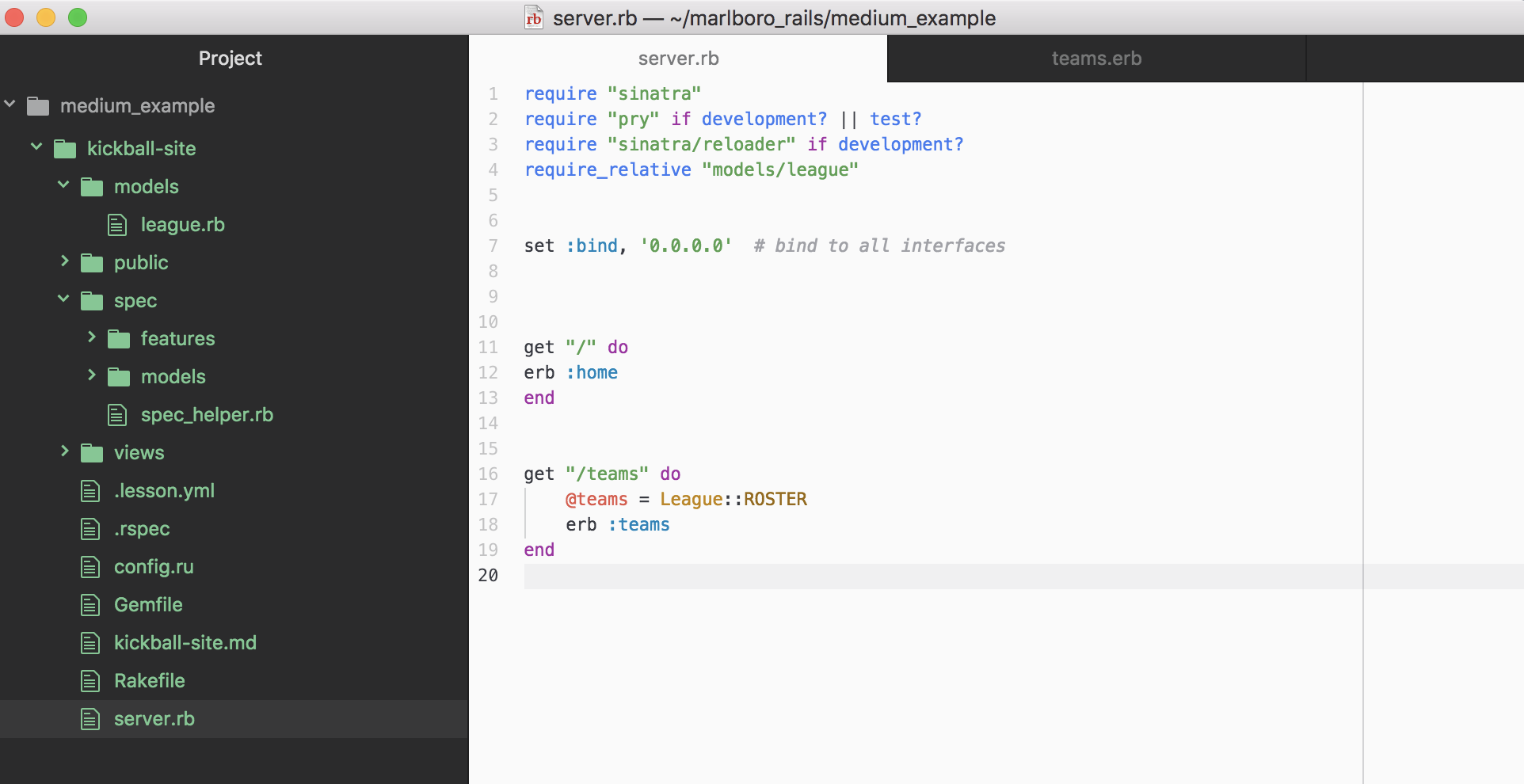
But how do we actually access data in our model (that ‘league.rb’ hash you first saw’) and get it onto the browser?

So its important to note that usually, a hash of data won’t just be kept right in models like it is in league.rb. Usually data will be stored in a database or csv file and you’d be using the model to access the csv file. But for this example we are getting this big piece of static data which is the roster and it is conveniently located in our league.rb file.

**##rinse and repeat to make another webpage**

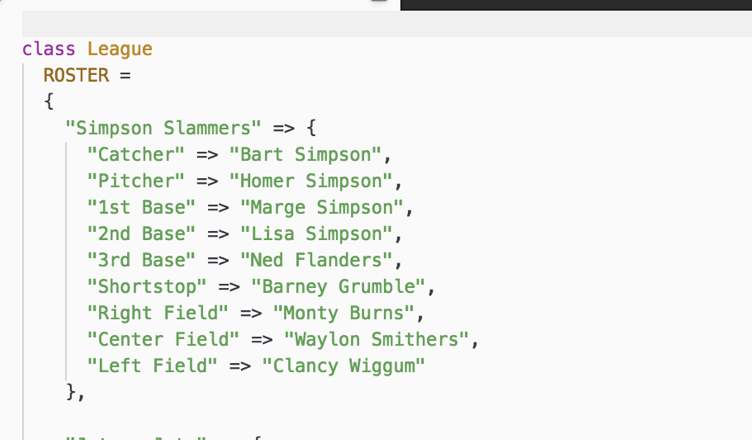
So, lets make a page that lists all the teams. We can repeat the steps we took to make a homepage- first setting a route in our server, then creating a ‘teams.erb’ file in our views folder.

1. Go to server.rb and set up the url for /teams and define @teams as your data and connect it to the data and link it with. Erb :teams
2. Go to teams .erb and follow hash convention to iterate through the teams in the data and put them up.



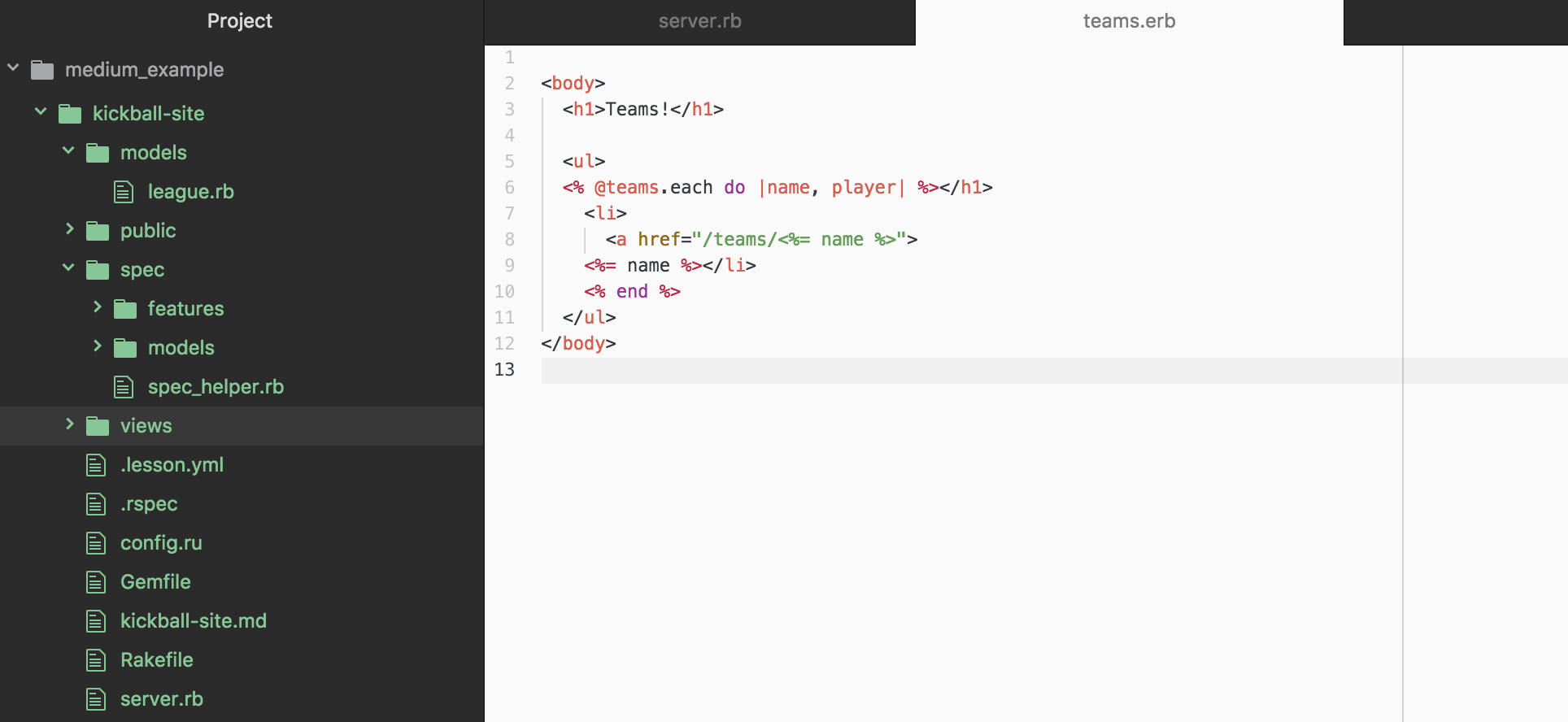
Here, setting the route in our server file, we are creating the instance variable @teams because we need to cycle through the league.rb hash with an .each do statement to spit out all the teams individually onto the browser. But don’t worry too much about that yet.

Don’t forget, though, to link your @teams variable with the database.

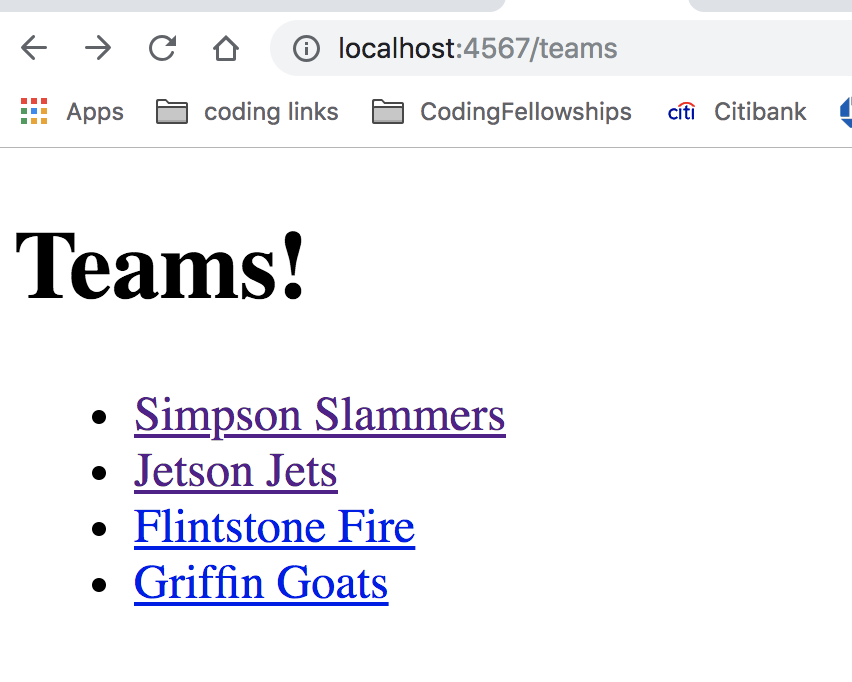


…@teams = League::ROSTER is saying, ‘ @teams contains the roster data within this league hash!

We’re going to iterate through it in our views folder, in our teams.erb file;



Since @teams is connected to both the team names and team players of the hash, you have access to both. Stick ‘em in an <a href> link and you have yourself a web browser that shows the team names only as a link!;



…Also check out that sweet ‘/teams’ route you set in your server/controller file and see how the URL is popping up with it here!

