Bitly: Building Our First Web App

Time	~ 3 - 4 hours
Learning Goals	• Understand how the web works through the HTTP protocol • Understand how HTML forms work • Understand the difference between the various HTTP methods • Deeper understanding of routes and the MVC relationship • Learn about ActiveRecord::Callbacks • Debugging

We are going to build a basic version of Bit.ly, a url shortener! Do this well, because this is going to be one of your portfolios when you graduate! It helps with getting freelance jobs, employment and a lot more! You will show the world how far you have come by sharing this web app you build online, accessible by anyone through a web browser!

Set Up Instructions

1. Fork the repo from: https://github.com/NextAcademy/bitly-clone
2. Clone the repo to your computer: `$ git clone <repo-link>`
3. `cd` into the app folder. Run `$ bundle install` to install the necessary gems.
4. Launch Your App Server by typing `$ shotgun config.ru` in the command line. You should now be able to visit the skeleton app at http://localhost:9393.

Note: The `localhost` always refers to "the current machine," so you actually have a tiny web server running on your own computer!

Sinatra, like Rails, is a Rack-based framework, which means the main point of entry is this config.ru file. The ru stands for "rackup."

Note that this application uses Postgres for its database, not SQLite. If there's a database-related error at any point grab a staff member to make sure the machine is configured correctly and Postgres is running.

ps: at this juncture, you may wonder what's the difference between SQLite and Postgres SQL. Well, we are glad you asked! Check out this link to see what it has to say about the different SQL databases!

Objectives

Bitly: Building Our First Web App

Explore Bit.ly

Check out the Bit.ly web app and familiarise yourself with the way it works. The main goal of Bit.ly is for a user to submit a long url link, which is then converted into a shorter url that can be more easily shared.

See MVC in action

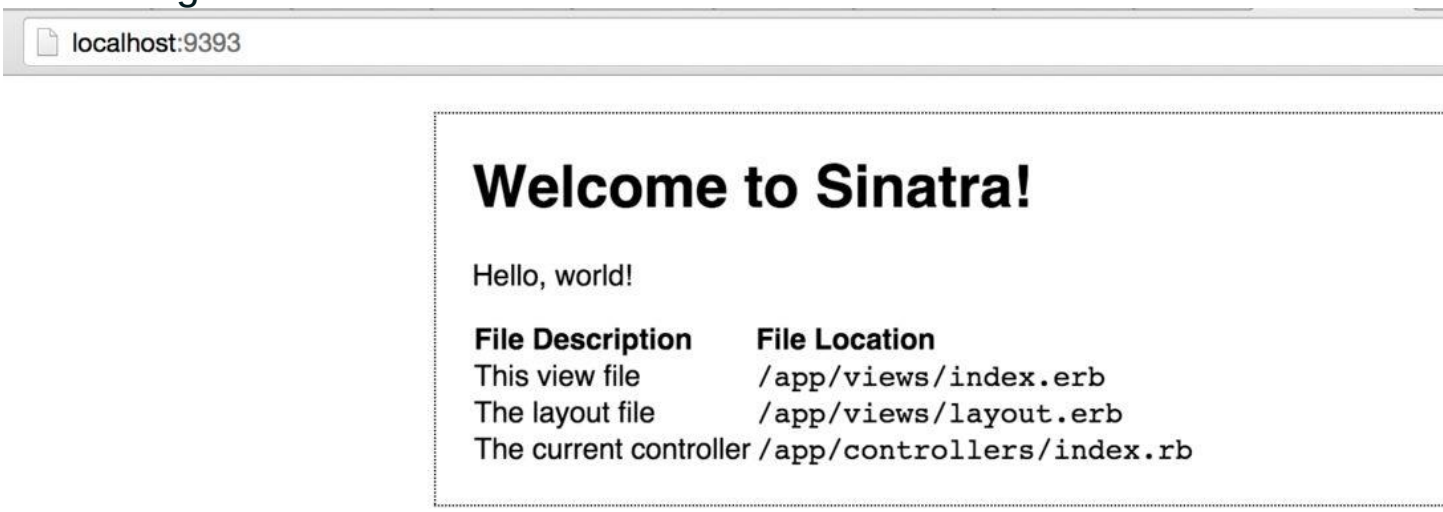
Jumping from one file of code to many files to an entire folder of files is scary. But trust us, it's way better to have many files sorted by type then one giant file with 20 - 100 thousand lines of code.

MVC as we know by now is just a way to organize code. Now let's see it in in action in the app's source code. Explore the directory structure and read through the files in each folder. Please spend some time here so that we can start familiarising ourselves with the MVC structure.

- Controllers are in app/controller
- Views are in app/views
- Models will be in app/models

Front-end: Creating a form

Currently, the homepage of your skeleton app would look something like this:



As you may have observed, the View for the file is located at index.erb.

Bitly: Building Our First Web App

Let's start by creating a simple form that will take in a url input. We can replace the contents of index.erb with a basic form like this:

```
<form action="#" method="">
  Enter URL:
  <input type="text" name="long_url">
  <input type="submit" value="Submit">
</form>
```

We'll spruce it up with HTML & CSS later. For now, keep it simple.

Back-end: Creating Models and Migration

Did you try submitting this form? Hmm, nothing happens. That is because we have not told the form what to do with the input. Let's start building the back-end for our url shortener.

- First, you would need to create a method to convert the long urls to shortened urls (put this method inside the model).
- Second, you would need to store the urls that have been submitted by users in a database (migration).

To do so, we would need a Url model and a create_urls migration. You can generate them by running the following from the command line inside the application root directory:

```
$ rake generate:model NAME=Url
$ rake generate:migration NAME=create_urls
```

These are custom rake tasks. Look in the Rakefile to see how they work, if you are curious.

After you have run these rake tasks, you are now able to access your model in app/models and see your migration file in db/migrate.

Creating a Method

Add two fields to store the original long url and shortened url in a migration table:

```
class CreateUrls < ActiveRecord::Migration
  def change
```

Bitly: Building Our First Web App

```
  # Write code that will create a URLs table and its needed fields. Think of your database like a giant Excel Spreadsheet.
 end
end
```

Define a method on your Url model that will return a shortened url string. This can be a string of random alpha-numeric characters.

```
class Url < ActiveRecord::Base
 def shorten
  # Write a method here
 end
end
```

Create Controllers

Let's build the controller. We have one resource: Urls. For our controllers, we need a URL that lists all our Url objects and another URL that, when POSTed to, creates a Url object.

We'll also need a URL that redirects us to the full (unshortened) URL. If you've never used Bit.ly, use it now to get a feel for how it works.

The controller methods should look like this:

```
get '/' do
 # let user create new short URL, display a list of shortened URLs
end


post '/urls' do
 # create a new Url
end
```

Bitly: Building Our First Web App

```
# i.e. /q6bda
get '/:short_url' do
  # redirect to appropriate "long" URL
end
```

Use a Callback

Active Record has a set of methods called callbacks that gets called when an object is “created, saved, updated, deleted, validated or loaded from the database”. In the Url model, use a before_create callback in the Url model to generate the short URL (i.e. before the URL is saved, run the shorten method on the long URL).

ps: for a “plain English” explanation of callbacks, check out this stack overflow!

Debugging

**Method 1: Printing to Log

**In the event that you’re stuck in the controller code, you can always use puts like this:

```
get '/' do
  puts "[LOG] Getting /"
  puts "[LOG] Params: #{params.inspect}"
  erb :index
end
```

Method 2: Byebug

If you have not$gem install byebug, it’s time to do it now! Remember to require 'byebug' in the right file(s). If you are stuck, kindly ask your mentor to give you a demo on how to use it!

Using byebug in the right controller, see what happens if you change the input name elements of the form to:

Bitly: Building Our First Web App

```
<input type="text" name="url[long]">
```