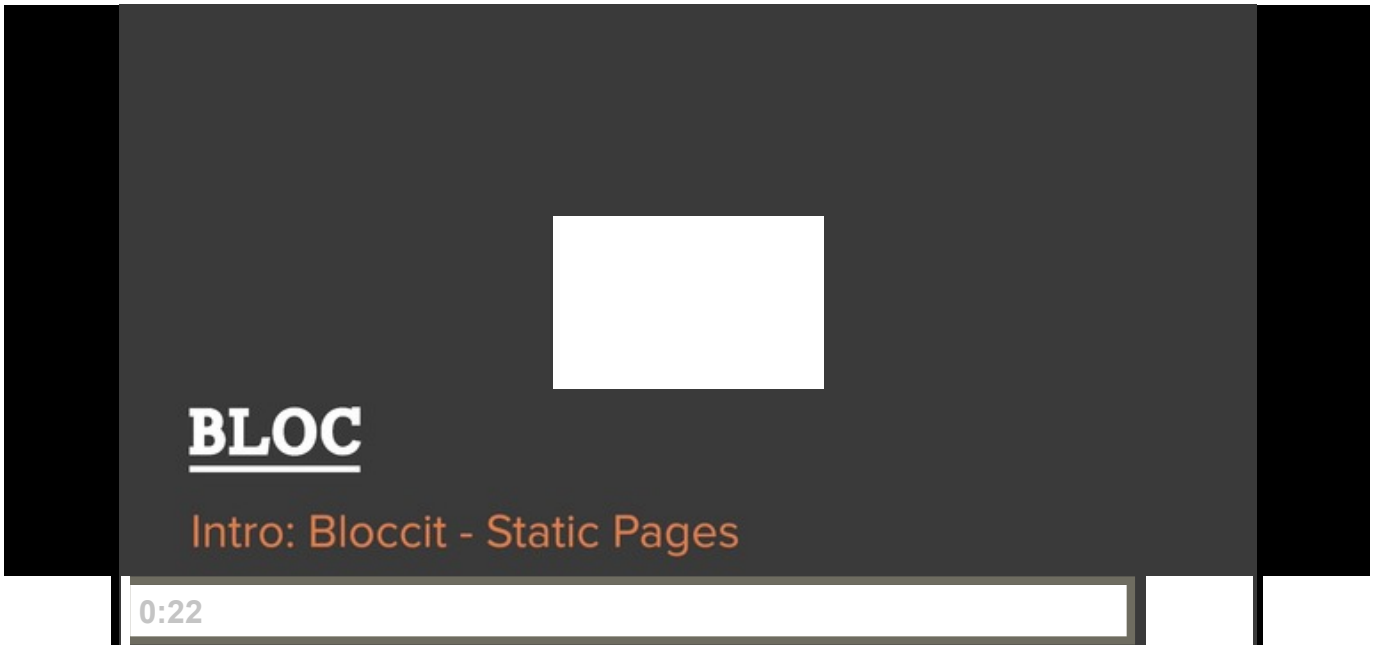
25 Rails: Static Pages

> “I make static art, not dynamic art. That's what I do.”
>
> — **Michael Heizer**

Static Views

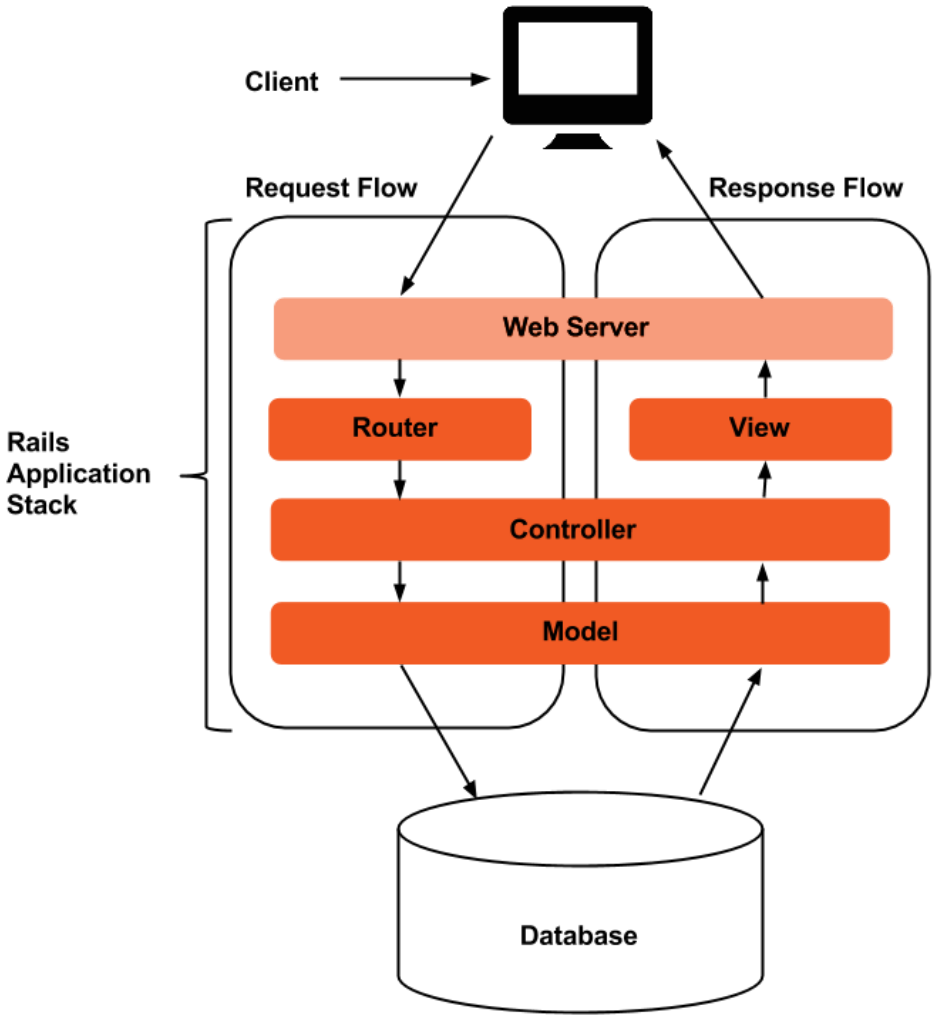


We have a working Rails app, but other than the default index page there's not much to show. The purpose of this checkpoint is to build static views, and in the process we'll learn the fundamentals of MVC architecture.

MVC Architecture

MVC, which is an acronym for "Model View Controller", is the basic architectural pattern that guides the creation of all Rails applications. You worked with basic MVC when you built Address Bloc. In this checkpoint, we'll focus on views and controllers and learn about models later.

A view is equivalent to a web page, and a controller determines what view should be shown. Consider the diagram below and focus on the flow of the request and response, as they pertain to views and controllers.



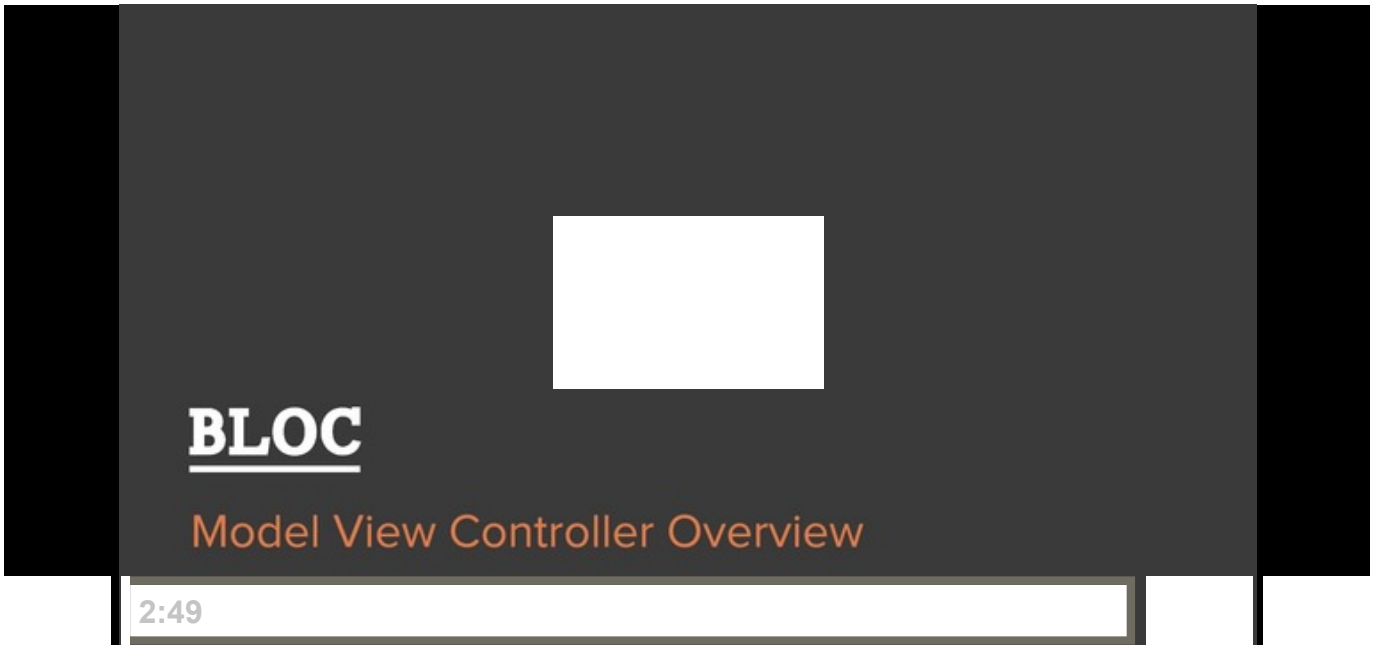
When you visit a website, you initiate a chain of actions. In an MVC application, a request is handled by a controller, which receives information from the model layer, and then uses that information to display a view.

MVC architecture is analogous to the basic function in a restaurant:

1. A customer (user) places an order with the waiter (controller).
2. The waiter informs the kitchen (model) of the order.
3. After the kitchen makes the order, the waiter serves the dish (view) to a customer.

> The waiter doesn't need to know how the order will be prepared, or how it will be consumed, and that's just fine. Controllers, like waiters, should only be concerned with passing things to other parties.

We review MVC components and examples of their corresponding code in the next video:



Git

Create a new Git feature branch for this checkpoint. See **Git Checkpoint Workflow: Before Each Checkpoint** for details.

Generating a Controller and Views

The best way to understand the relationship between controllers and views is to create them. We could create controller and view files manually, but Rails provides a handy generator which ensures that *all* necessary files are generated for a given controller. To generate a controller and its views, type the following on your command line in your project's directory:

Terminal

```
$ rails generate controller welcome index about
```

The output should look like this:

Terminal

```
      create  app/controllers/welcome_controller.rb
       route  get "welcome/about"
       route  get "welcome/index"
      invoke  erb
      create    app/views/welcome
      create    app/views/welcome/index.html.erb
      create    app/views/welcome/about.html.erb
      invoke  helper
      create    app/helpers/welcome_helper.rb
      invoke  assets
      invoke    coffee
      create      app/assets/javascripts/welcome.coffee
      invoke    scss
      create      app/assets/stylesheets/welcome.scss
```

We passed three arguments to the `rails generate` command. The first argument represents the controller name, which is `welcome`. The next two arguments (`index` and `about`) represent views corresponding with the `welcome` controller. We could've named the controller and views anything, but the names should correspond with their primary function, as a best practice.

Exploring Controllers and Views

Open your project in your text editor. You should see a file named `welcome_controller.rb` in `app/controllers/`. You should also see the two views you created in `app/views/welcome/`. The generator created some code:

app/controllers/welcome_controller.rb

```
class WelcomeController < ApplicationController
  def index
  end

  def about
  end
end
```

`WelcomeController` is a Ruby class, and contains two empty methods that correspond to view names. These identically named methods and views are an example of a Rails

convention called **default rendering**. When a controller method's purpose is to invoke a view, *it must be named with respect to the view*. The `index` method in the `WelcomeController` will invoke the **index** view inside the `app/views/welcome` directory.

Open the **index** and **about** views and read the placeholder code:

app/views/welcome/index.html.erb

```
<h1>Welcome#index</h1>
<p>Find me in app/views/welcome/index.html.erb</p>
```

app/views/welcome/about.html.erb

```
<h1>Welcome#about</h1>
<p>Find me in app/views/welcome/about.html.erb</p>
```

Start the Rails server from your command line:

Terminal

```
$ rails s
```

Visit **localhost:3000/welcome/index** and **localhost:3000/welcome/about** to view the HTML code that was created by the controller generator.

Routing in Rails

The controller generator created the basic code needed for the `WelcomeController` and its views, and it also created code in the `config/routes.rb` file:

config/routes.rb

```
Rails.application.routes.draw do
  get "welcome/index"

  get "welcome/about"
  ...
end
```

This code creates HTTP `GET` routes for the **index** and **about** views. HTTP is the protocol that the Internet uses to communicate with websites. The `get` action corresponds to the HTTP `GET` verb. `GET` requests are used to retrieve information identified by the URL.

> The HTTP protocol has other actions which we'll explore later.

If `routes.rb` doesn't specify a `GET` action, the view will not be served because the application won't know what to `get` when a user sends a request. Test this by commenting out these lines:

config/routes.rb

```
  # get "welcome/index"

  # get "welcome/about"
```

Restart the server and visit **localhost:3000/welcome/index**. We'll see a Rails "Routing Error" page. This error occurs when our app doesn't understand what we're requesting, because there is no corresponding `get` action.

Uncomment those two lines and **delete all the other commented lines in the file**. Add a `root` path to the `routes.rb` file:

config/routes.rb

```
  Rails.application.routes.draw do
    get "welcome/index"

    get "welcome/about"

+   root 'welcome#index'
  ...
  end
```

The `root` method allows us to declare the default page the app loads when we navigate to the home page URL. Test it by going to `localhost:3000`. You should see the welcome **index** view by default.

> `root` is a method that takes a hash as an argument, here using the "implied hash" syntax. The

> line could be rewritten without using an implied hash as: `root({to: 'welcome#index'})`. You'll see implied hashes frequently in Rails because they enhance readability.

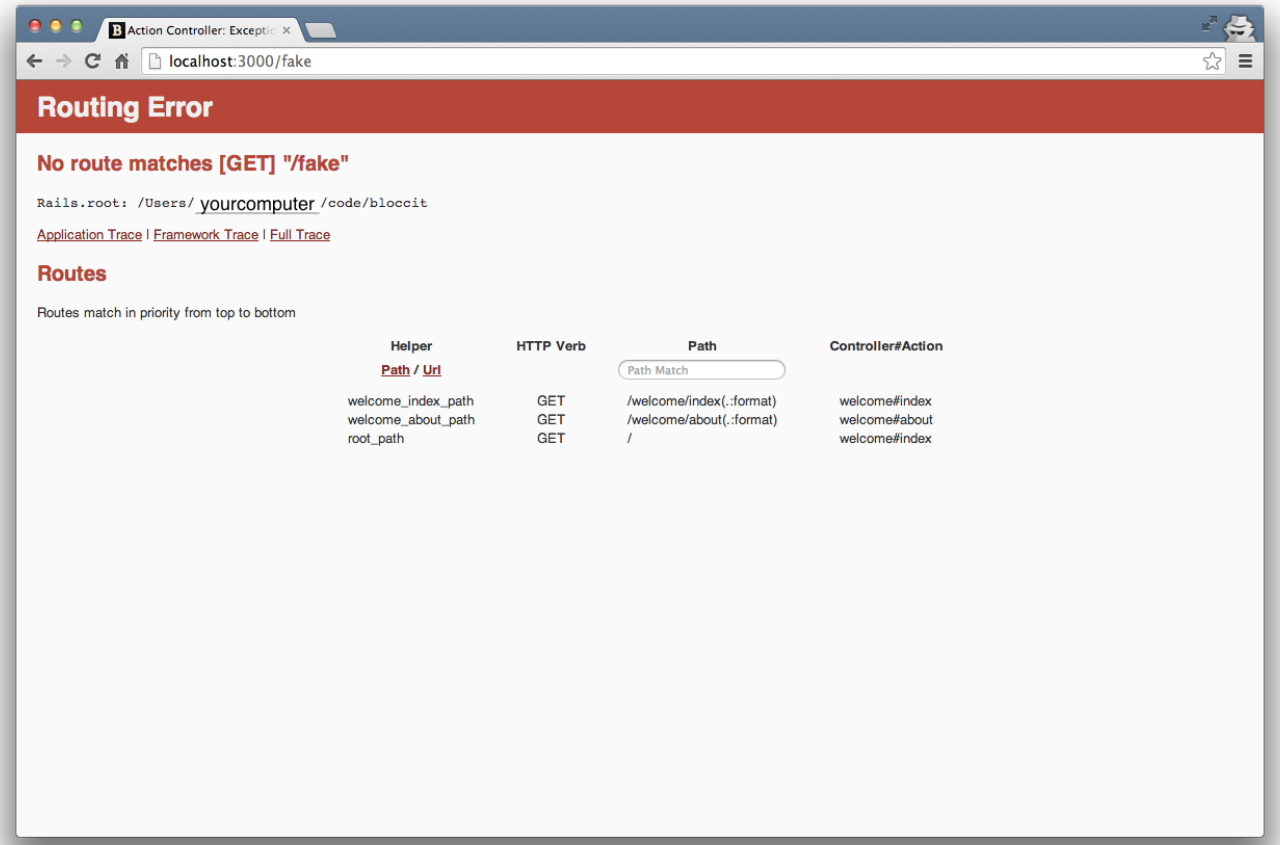
View your app's available routes by typing `rake routes` from the command line. Stop the Rails server (`CTRL+C`) and give it a try. You should see the following output:

Terminal

```
$ rake routes
       Prefix Verb URI Pattern                Controller#Action
welcome_index GET /welcome/index(.:format) welcome#index
welcome_about GET /welcome/about(.:format) welcome#about
         root GET /                        welcome#index
```

- The first column represents the route name: `welcome_index`
- The second column represents the HTTP action associated with the route: `GET`
- The third column represents the URI pattern, which is the URL used to request the view: `/welcome/index`
- The fourth column represents the route destination, which translates to the controller and associated view: `welcome#index`

> By default, Rails will present a searchable list of valid routes if an invalid route is requested. This is handy for troubleshooting large applications with many routes, and is also a nice fail-safe. Try it on `localhost`:



Git

Commit your checkpoint work in Git. See **Git Checkpoint Workflow: After Each Checkpoint** for details. Then deploy to Heroku.

Recap

Concept	Description
MVC	MVC (Model–view–controller) is an architectural pattern that divides a given application into three interconnected parts with distinct responsibilities.
Git	Diverges the master branch, so that you can work on new features without affecting the master branch. Git branches require little

Branching	memory or disk space, making branching operations nearly instantaneous.
`rails generate`	The `rails generate` command creates controllers from templates. The `generate` command can also generate controller actions and their corresponding views.
Controller	Controllers are represented by the C in MVC. Controllers process requests and produce the appropriate output. Controllers communicate with the database and perform CRUD actions where necessary, via models.
Views	Views are responsible for rendering templates. View templates are written using embedded Ruby in tags and integrated with HTML.
`rake routes`	The `rake routes` command lists all routes, in the same order as `routes.rb`.
localhost	**localhost** is a hostname that represents "this computer".

Assignment | Discussion | Submission

Create a new Git feature branch for this assignment. See **Git Checkpoint Workflow: Before Each Assignment** for details.

Use what you learned in this checkpoint to create a Contact page, do not use `rails generate`:

1. Manually create `app/views/welcome/contact.html.erb`.

2. Manually create the route to your new page in `routes.rb`.
3. Add a `contact` action to `WelcomeController`.

Commit your assignment in Git. See **Git Checkpoint Workflow: After Each Assignment** for details. Submit your commit to your mentor.

Solution

Do not watch this video until after you've attempted to complete the assignment. If you struggle to complete the assignment, submit your best effort to your mentor *before watching a solution video*.

Static Pages Solution

assignment completed

?