Create a new application 3

1. Create an account at Nitrous.io 3

2. On the Nitrous website, create a new Rails project called pinterest\_clone 3

3. Create a new Rails app 3

4. Run the Rails server 4

5. Preview the app 4

Commit with Git 4

1. Open a new console tab 4

2. Initialize Git 5

Create a home page 5

1. Create a new page 5

3. Update text on your new page 5

Direct users to home page 6

1. Show the home page of your app 6

Create more pages 7

1. Add a new action in the controller 7

2. Create the HTML in your view 7

3. Add your route 7

What’s embedded Ruby? 7

Create navigation links 8

Add navigation links 8

Install the Bootstrap gem 9

1. Add the Bootstrap gem 9

2. Bundle install to install the new gem 9

3. What does the *application.css* file do? 9

4. Create a new SCSS file 10

5. Restart the server 10

Add Bootstrap styling 10

1. Add a container 10

2. Create a partial template for the header 11

3. Create a link to that partial template 11

4. Fill out the nav bar partial template 11

5. Require Bootstrap's JavaScript 12

6. Add viewport to set width and scale 12

7. Add a jumbotron 12

Generate Pins Scaffold 13

2. Delete the default scaffold CSS 14

Pins Controller 14

Pins Views 15

1. Delete these files 15

2. The partial template for a form 15

3. Commit to git 16

Upload images using the Paperclip gem 16

1. Install the paperclip gem 16

2. Generate a paperclip migration 17

3. Restart server 18

4. Add image upload field to pin form 18

5. Update the Pins Controller for strong parameters 19

6. Update the pins show view 19

7. Update the pins index 19

8. Add some pins 20

Change the Root Route 20

1. Change the routes so that the home pages is the Pins index 20

2. Commit to Git 21

Order Pins Chronologically 21

1. Edit Pins controller 21

2. Commit to Git 22

\*\*\*\*\*BONUS SECTION\*\*\*\*\* 23

BONUS: Use jQuery Masonry to add transitions 23

BONUS: Create a page to show only one pin 23

BONUS: Add icons with Glyphicons 23

BONUS: Add pagination 23

BONUS: Add pin validations 23

# Create a new application

### 1. Create an account at Nitrous.io

### 2. On the Nitrous website, create a new Rails project called pinterest\_clone

[Nitrous.io](https://www.nitrous.io/) provides a hosted development environment. Basically, you get access to a remote computer from your web browser. The Nitrous computer runs Ubuntu Linux, which is a popular operating system for Rails development.

Let’s take a look at what this README file says.

First it says “To see which version of Rails is installed, run: rails –v” so let’s do that here in the black box at the bottom, which is called a console. We’re running Rails 4.2.5.1 which is important to know because if you run into trouble later on and need to ask questions about how to fix a bug, in your question it’s always helpful to include the version number of the tools you’re using.

Scrolling down on the README we can also see that Nitrous included a sample app, which we can delete by right-clicking on it.

### 3. Create a new Rails app

So now let’s build our pinterest app. To do that type this in the black box so that we can get situated inside the right folder:

➜  cd ~/code

➜  rails new pinterest\_clone –d postgresql

You can see that a ton of files were just created and a lot of things were just installed. Let’s switch from the code folder into the directory we just created with the command:

➜  cd pinterest\_clone

### 4. Run the Rails server

Now let’s launch the rails server with this command:

➜  rails s -b 0.0.0.0

A web server is a computer that can be accessed through the internet. When you visit www.google.com, 3 things happen:

1. Your browser sends out a request for a web page
2. A web server receives your request and puts together the right parts
3. The web server sends you back a web page

It’s like ordering food at a restaurant. You tell them what you want, then they make it and they deliver it to your table.

### 5. Preview the app

Now let’s see what our pinterest\_clone looks like by going to the navigation bar at the top and clicking on PREVIEW and then Port 3000(default).

So there is our first Rails application. Very exciting!

# **Commit with Git**

Before we go any further, let’s use Git to save our progress.

### 1. Open a new console tab

Let’s go back to Nitrous. The console tab we have open -- the black box at the bottom -- is for the server to run, and we can’t type anything in here unless we close down the server by typing control-C.

To open a new tab, click on the plus-sign at the top of the console. Once inside that tab, navigate into the pinterest clone folder with these commands:

➜  cd ~/code

➜  cd pinterest\_clone

### 2. Initialize Git

We only do this once in the beginning of every new project with the command:

console

➜  git [init](https://onemonth.com/courses/one-month-rails/steps/commit-with-git)

➜  git config --global user.email "youremail@example.com"

➜  git config --global user.name "Your Name"

➜  git status #shows files you’re tracking (or not tracking)

➜  git add .

➜  git commit –am “Initial Commit”

# **Create a home page**

To create pages in your app, create a pages controller.

### 1. Create a new page

console

➜  rails generate controller pages home

In your preview tab, tack this text onto to the URL:

[/pages/home](localhost:3000/pages/home" \t "_blank)

and see the new blank homepage you just created.

### 3. Update text on your new page

On the menu bar on the left side of the screen, open app🡪 views🡪pages🡪home.html.erb and type this code in that file:

<h1>This is my app!</h1>

<p>We’re learning to build with SpeakCode</p>

Save the file by typing:

Mac: Command + S

or

PC: Ctrl + S

Now we can refresh the browser preview tab and see our changes under pages/home

# **Direct users to home page**

You can direct people to a certain page by setting a route that points to that page.

### 1. Show the home page of your app

Open config/routes.rb

Replace the line

get "pages/home"

with:

root "pages#home"

In the other browser tab, now if we delete the pages/home part of our URL we see that when we go to the home page, we’re getting redirected to pages and then home.html.erb.

# **Create more pages**

You’ll probably need more than just a "Home” page on your app. To do that, we’ll add a new view and set a route for it.

### 1. Add a new action in the controller

Open app/controllers/pages\_controller.rb

def about

end

### 2. Create the HTML in your view

app/views/pages/about.html.erb

<h1>About Us</h1>

<p>We’re building an app with SpeakCode</p>

### 3. Add your route

config/routes.rb

get "about" => "pages#about"

# **What’s embedded Ruby?**

Embedded Ruby (erb) lets you add dynamic content to static HTML pages.

1. Links in HTML

Remember that in HTML, a link looks like this:

<a href="#">here</a>

In a Rails app, we create links using embedded Ruby tags instead. Inside these tags we can write code in the Ruby language. The tags are like saying, heads up, we’re not going to be speaking HTML for a second—instead we’ll be speaking Ruby—so please listen for Ruby instead of HTML. This is what Ruby tags look like:

<%= %>

2. Add an embedded Ruby link to the home page:

Let’s write a full Ruby link into the *home.html.erb* file:

*Open app/views/pages/home.html.erb*

<%= link\_to "here", "#" %>

In Ruby on Rails, a full link will look like that (that’s Ruby code inside the tags).

# **Create navigation links**

In Rails, layouts let you create elements that show up on every page in your app--like a navigation bar that always appears that the top.

### Add navigation links

Open apps/views/layouts/application.html.erb and add this code above the line that says <%= yield %>:

<%= link\_to "Home", root\_path %> <%= link\_to "About", about\_path %>

Now when we refresh our other browser tab, we see these nav links.

# **Install the Bootstrap gem**

We can install Bootstrap with a Ruby gem. Ruby gems are time-saving libraries of code that you can download and plug into your application. We save a list of our app’s Ruby gems in the Gemfile.

### 1. Add the Bootstrap gem

Open /Gemfile and add this code under the line that says  
 “gem 'jbuilder', '~> 2.0'”

..  
gem 'bootstrap-sass'  
..

### 2. Bundle install to install the new gem

Every time we add a Ruby gem to our list of gems in the Gemfile, we need to tell Rails to grab those files from the internet, drop them into our app, and install them with this command in the console:

console:

➜  bundle install

### 3. What does the *application.css* file do?

Open app/assets/stylesheets/application.css

Application.css takes all the other files in your /stylesheets directory and combines them when you run your app.

### 4. Create a new SCSS file

SCSS is a **precompiler** for CSS. It helps you write CSS quicker. (This will make more sense soon).

Create a new SCSS file by right clicking on app/assets/stylesheets and scrolling to “New File”

Type this inside the new file

@import 'bootstrap';

Use command+S or CTRL+S to save the file as: bootstrap\_custom.css.scss

### 5. Restart the server

Restart your server whenever you add a new gem.

In console, switch over to the first tab and shut off the Rails server:

CONTROL + C

Then use the up-arrow key to display your previous command, and press enter:

➜  rails s -b 0.0.0.0

Now when we refresh our home page we can see that some pretty Bootstrap styling has been applied.

# **Add Bootstrap styling**

### 1. Add a container

views/layouts/application.html.erb

<%= link\_to "Home", root\_path %> <%= link\_to "About", about\_path %>

<div class="container">

<%= yield %>

</div>

### 2. Create a partial template for the header

If you have a big chunk of code that you’re going to use over and over in an app—in different parts of your app—then a partial template is a good way to streamline. The partial template is where you store that big chunk of code so that you’re not copy and pasting the same lines over and over.

Let’s create a new file called “\_header.html.erb” and save it in app/views/layouts.

### 3. Create a link to that partial template

Instead of copying and pasting the same lines of code when you want all of those lines of code to be executed, you simply call the partial template using this line of code:

app/views/layouts/application.html.erb:

replace

<%= link\_to "Home", root\_path %> <%= link\_to "About", about\_path %>

with

<%= render 'layouts/header' %>

### 4. Fill out the nav bar partial template

app/views/layouts/\_header.html.erb

<nav class="navbar navbar-default" role="navigation">

<div class="navbar-header">

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-ex1-collapse">

<span class="sr-only">Toggle navigation</span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

<a class="navbar-brand" href="#">My Pinterest Clone</a>

</div>

<div class="collapse navbar-collapse navbar-ex1-collapse">

<ul class="nav navbar-nav navbar-right">

<li><%= link\_to "Home", root\_path %></li>

<li><%= link\_to "About", about\_path %></li>

</ul>

</div>

</nav>

### 5. Require Bootstrap's JavaScript

app/assets/javascripts/application.js

...

//= require jquery

//= require jquery\_ujs

//= require bootstrap

//= require turbolinks

//= require\_tree .

### 6. Add viewport to set width and scale

A viewport meta tag sets the width and initial scale of the viewport. Below the line that says <%= csrf\_meta\_tags %>, write the below code:

views/layouts/application.html.erb

<meta name="viewport" content="width=device-width, initial-scale=1.0">

### 7. Add a jumbotron

views/pages/home.html.erb

<div class="jumbotron">

<h1>Welcome to my app!</h1>

Click <%= link\_to "here", "#" %> to visit a signup link that we're not implementing today.

</div>

# **Generate Pins Scaffold**

Generating a scaffold lets us quickly add new pins to our app.

In this Rails app we have pins, which we call “resources” in this context. For pins, we could:

1. create a pins **model** (to define what data we have in our app, the relationships between that data, and how we will use it)
2. create a pins **controller** (to connect the View and the Model to the database where pin info is stored), and ferry information between all three
3. create **views** in which the user can for Creating, Reading (viewing), Updating (editing), and Destroying (deleting) pins. Another way of saying this is that it’s a CRUD—Create/Read/Update/Destroy—resource.

Or, instead of creating all of these files ourselves, we can use a simple command:

➜  rails generate scaffold pin description:string

In that command we’re tellng Rails to create all those files for a resource called a Pin, and each pin will have a description that’s a string.

Now, since we’ve added a form of data—Pins—to our app, we need to give Rails a heads up that we want to use the version of the database that has an empty table in which we’ll save data for our pins.

➜  rake db:migrate

### 2. Delete the default scaffold CSS

This file is also created with scaffold. It messes with the rest of your CSS, so delete it

*(to view this new file in Nitrous you might need to click “Refresh Tree”, which you can access by clicking on the three dots at the top of the tree, next to Pinterest\_clone)*

app/assets/stylesheets/scaffolds.css.scss

# **Pins Controller**

Let’s look at the controller to better understand its actions. Open this file that we just created and delete the commented, gray text that starts a # symbol in:

*app/controllers/pins\_controller.rb*

You can see in this file the actions that can be taken on each pin:

* we can display an INDEX of all pins
* we can SHOW a single pin
* we can create a NEW pin that won’t be saved
* we can EDIT an existing pin (this sends the user a form in their browser view, in which they can make edits)
* we can CREATE a pin, which both creates and saves a new pin to the database
* we can UPDATE a pin, which the action that happens when a user submits a form to EDIT the pin
* we can DESTROY the pin, or delete it from the database

Then we have some private actions at the bottom. They’re called “private” because these methods—“set\_pin” and “pin\_params” won’t be applied to anything other than a pin.

So, again, these are actions that can be taken on each pin resource. If we were making a Twitter clone, we’d take these actions on resources called Tweets. If we were making a Facebook clone, we’d take these actions on resources we’d call Statuses.

# **Pins Views**

Let's better understand the views we just created

### 1. Delete these files

We won't be using the JSON files, so we can get rid of them

app/views/pins/index.json.jbuilder

app/views/pins/show.json.jbuilder

### 2. The partial template for a form

You can see that in apps/views/pins/new.html.erb, we’re already telling Rails to render a form with this code:

<%= render 'form' %>

Let’s look at the form it’s rendering. That form lives in this file:

apps/views/pins/\_form.html.erb

At the top of this form we can see that we have a block that starts with

<%= form\_for(@pin) do |f| %>

Up top there is a block for messages that will get sent if there are any user errors that prevent a pin from being saved.

Below that there is a label—description, and then a text field—also a description.

Underneath that, there’s a submit button.

We can check out what this form looks like by tacking on:

/pins/new

to our preview URL in the other browser tab

### 3. Commit to git

Now would be a good time to commit the changes we’ve made to Git:

➜  git status

➜  git add .

➜  git commit –am “Added styling, pins resource”

# **Upload images using the Paperclip gem**

We have a form for pins, but there’s no field in that form for us to upload images. Paperclip is a gem that allows us to add that form field.

### 1. Install the paperclip gem

<https://rubygems.org/gems/paperclip/versions/4.3.6>

Type this into your Gemfile to add the paperclip gem to your list of gems, right under bootstrap:

/Gemfile

➜  gem 'paperclip', '~> 4.3', '>= 4.3.6'

Then run:

console

➜ bundle install

Now we need to tell explain, inside the pin model, that each pin has an attached image—and we need to explain what KIND of image we will allow users to upload. So let’s open up the pin model and add this code:

/app/models/pin.rb

class Pin < ActiveRecord::Base

has\_attached\_file :image, :styles => { :medium => "300x300>", :thumb => "100x100>" }

validates\_attachment\_content\_type :image, :content\_type => ["image/jpg", "image/jpeg", "image/png", "image/gif"]

end

### 2. Generate a paperclip migration

Now we need to tell database, *look for an image for each pin that you’re storing*, with this command:

console

➜ rails generate paperclip pin image

Then we run the migration:

console

➜ rake db:migrate

And we check to see the migrations that we’ve generated:

console

➜ rake db:migrate:status

So you can see that the first migration covered the creation of pins, and the second one, which we just created, was to add attachment image to pins.

### 3. Restart server

We just added a gem, so we need to restart the server:

console

➜ CTRL+C

Then arrow up one step and press enter to restart the server.

### 4. Add image upload field to pin form

Our form still looks like it did before, which means our users can’t upload any images even though we now have the capability within out app to do that. So let’s change the user’s view of the form by adding a field to upload images—add the text in red

/app/views/pins/\_form.html.erb

**<%= form\_for @pin**, html: { multipart: true } **do |f| %>**

**<% if @pin.errors.any? %>**

**<div id="error\_explanation">**

**<h2><%= pluralize(@pin.errors.count, "error") %> prohibited this pin from being saved:</h2>**

**<ul>**

**<% @pin.errors.full\_messages.each do |msg| %>**

**<li><%= msg %></li>**

**<% end %>**

**</ul>**

**</div>**

**<% end %>**

<div class="form-group">

<%= f.label :image %>

<%= f.file\_field :image, class: "form-control" %>

</div>

**<div class="form-group">**

**<%= f.label :description %>**

**<%= f.text\_field :description, class: "form-control" %>**

**</div>**

**<div class="form-group">**

**<%= f.submit** class: "btn btn-primary" **%>**

**</div>**

**<% end %>**

### 5. Update the Pins Controller for strong parameters

Open /app/controllers/concerns/pins\_controller.rb

Parameters are basically pieces of data that a user supplies. In our pins controller here at the bottom, we are permitting one piece of data that’s tied to the pin—the description. So let’s add the pin’s image to that list.

. . .

def pin\_params

params.require(:pin).permit(:description**,** **:image**)

end . . .

### 6. Update the pins show view

Now we need to update the view of the pin so that includes the image that the user uploaded. So let’s add some Ruby code right above the line that says <strong>Description:</strong> … and since we want the image size to be medium, we’ll include that too.

/app/views/pins/show.html.erb

<%= image\_tag @pin.image.url(:medium) %> . . .

### 7. Update the pins index

Add the text in red so that the image for each pin shows up in the index of all pins:

/app/views/pins/index.html.erb

<p id="notice"><%= notice %></p>

<h1>**My** Pins</h1>

<table>

<thead>

<tr>

**<th>Image</th>**

<th>Description</th>

<th colspan="3"></th>

</tr>

</thead>

<tbody>

<% @pins.each do |pin| %>

<tr>

**<td><%= image\_tag pin.image.url(:medium) %></td>**

<td><%= pin.description %></td>

<td><%= link\_to 'Show', pin %></td>

<td><%= link\_to 'Edit', edit\_pin\_path(pin) %></td>

<td><%= link\_to 'Destroy', pin, method: :delete, data: { confirm: 'Are you sure?' } %></td>

</tr>

<% end %>

</tbody>

</table>

<br>

<%= link\_to 'New Pin', new\_pin\_path %>

### 8. Add some pins

I’m going to upload 3 pins so you can see what they’ll look like in the index and show views.

# **Change the Root Route**

Let’s change a route in the routes.rb files so that all of our pins display on the homepage.

### 1. Change the ­­­­routes so that the home pages is the Pins index

/config/routes.rb

replace

root 'pages#home'

with

root "pins#index"

### 2. Commit to Git

➜  git status

➜  git add .

➜  git commit –am “Add image upload with Paperclip and change root path”

# **Order Pins Chronologically**

Right now the Pin that I most recently uploaded is showing up at the bottom of the screen, but it would be nice if it showed up at the top instead. With some changes to the code in the pins controller, we can make pins show up reverse-chronologically.

### 1. Edit Pins controller

app/controllers/pins\_controller.rb

replace

def index

@pins = Pin.all

end

with

def index

@pins = Pin.all.order("created\_at DESC")

end

### 2. Commit to Git

➜  git status

➜  git add .

➜  git commit –am “Made pins show up reverse-chronologically”

# **\*\*\*\*\*BONUS SECTION\*\*\*\*\***

# **BONUS: Use jQuery Masonry to add transitions**

# **BONUS: Create a page to show only one pin**

# **BONUS: Add icons with Glyphicons**

# **BONUS: Add pagination**

# **BONUS: Add pin validations**