Blog (pt. 1)
Setting Rails Up

Ruby on Rails - Hack Club

Note: This is not the only thing you can do with Ruby on Rails, though it might be one of the simplest. In fact, throughout this mini-project, you could notice how much you could do with Ruby on Rails.

Blog

Review:

What is Ruby?

- High-level, general purpose programming language
- Useful in web development via Ruby on Rails, Sinatra, etc. (look them up!)

Why use Ruby on Rails?

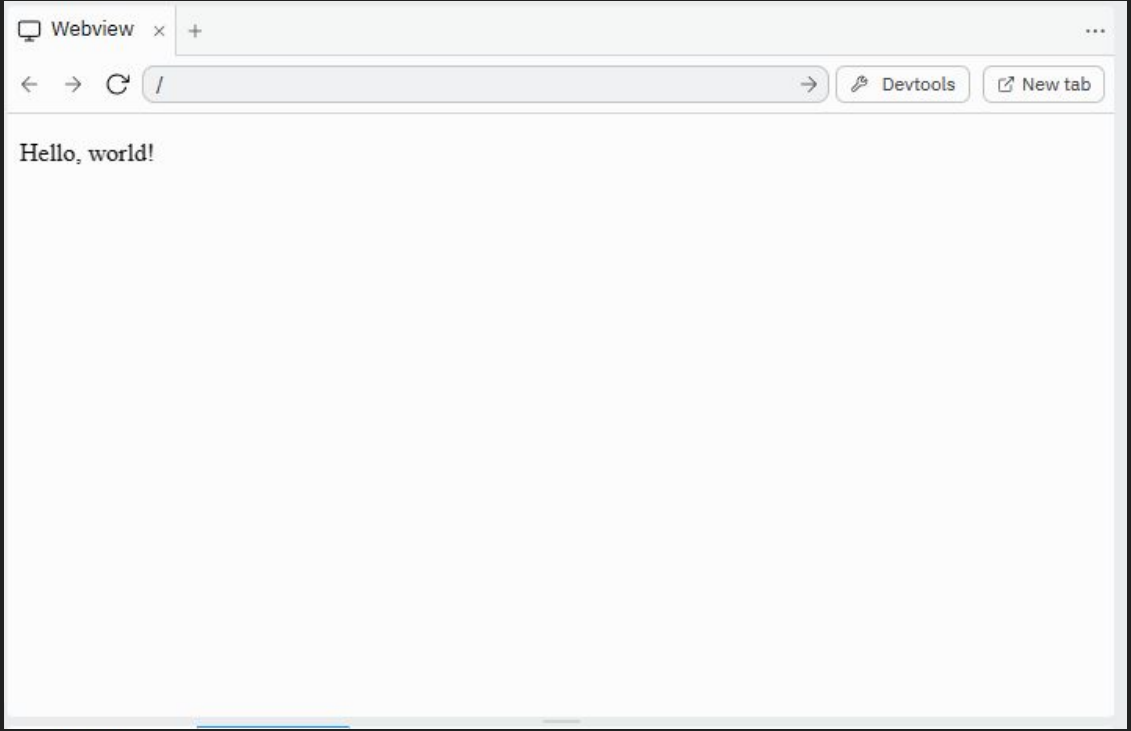
- Organized, efficient way of storing data with cleanly designed web pages and simple routing
- Gems - Ruby libraries (at least one exists to make what you want to create on a Ruby on Rails website a lot more convenient)

Blog

Goal:

(Not a blog yet)

(Setting up the Ruby project)

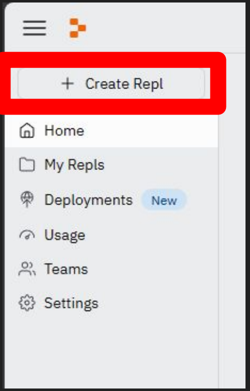


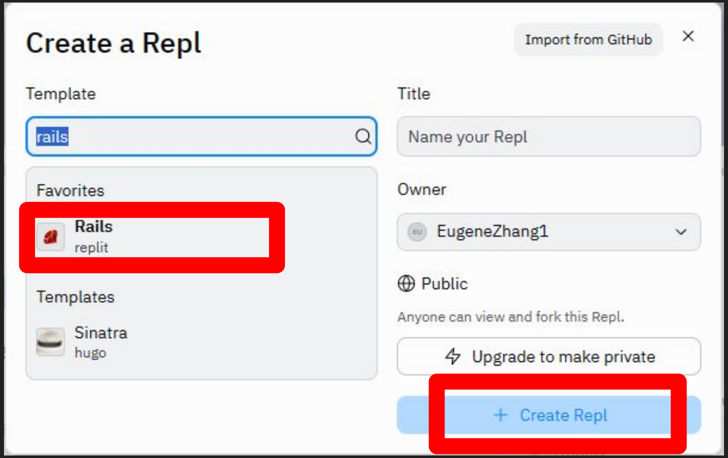
Hello, world!

Blog

Create a new Replit

https://replit.com/~





Blog

Theoretical Stuff

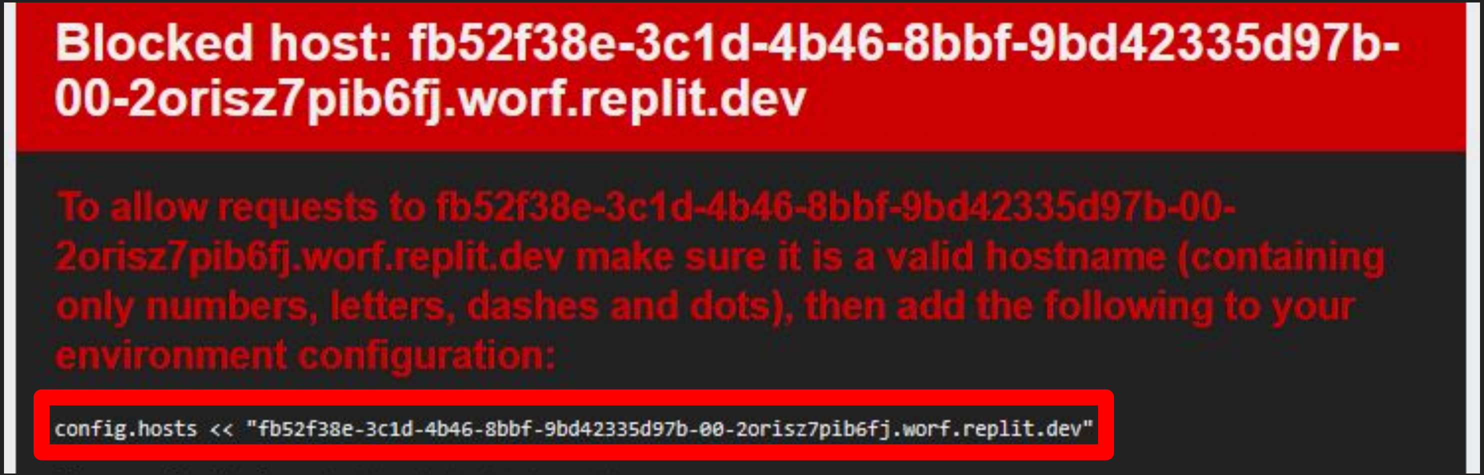
Bare minimum for functionality:

(1) Controller
  (a) Action
(2) View
(3) Route (URL)
(4) (Replit) Authentication “bypass”

Blog

Starting a Server

Right after a Rails Project is created, a server can usually be started. However, because Replit is stupid, it doesn’t work right away. Click “Run”. Below should be something like what the output should be.



Blocked host: fb52f38e-3c1d-4b46-8bbf-9bd42335d97b-00-2orisz7pib6fj.worf.replit.dev

To allow requests to fb52f38e-3c1d-4b46-8bbf-9bd42335d97b-00-2orisz7pib6fj.worf.replit.dev make sure it is a valid hostname (containing only numbers, letters, dashes and dots), then add the following to your environment configuration:

```
config.hosts << "fb52f38e-3c1d-4b46-8bbf-9bd42335d97b-00-2orisz7pib6fj.worf.replit.dev"
```

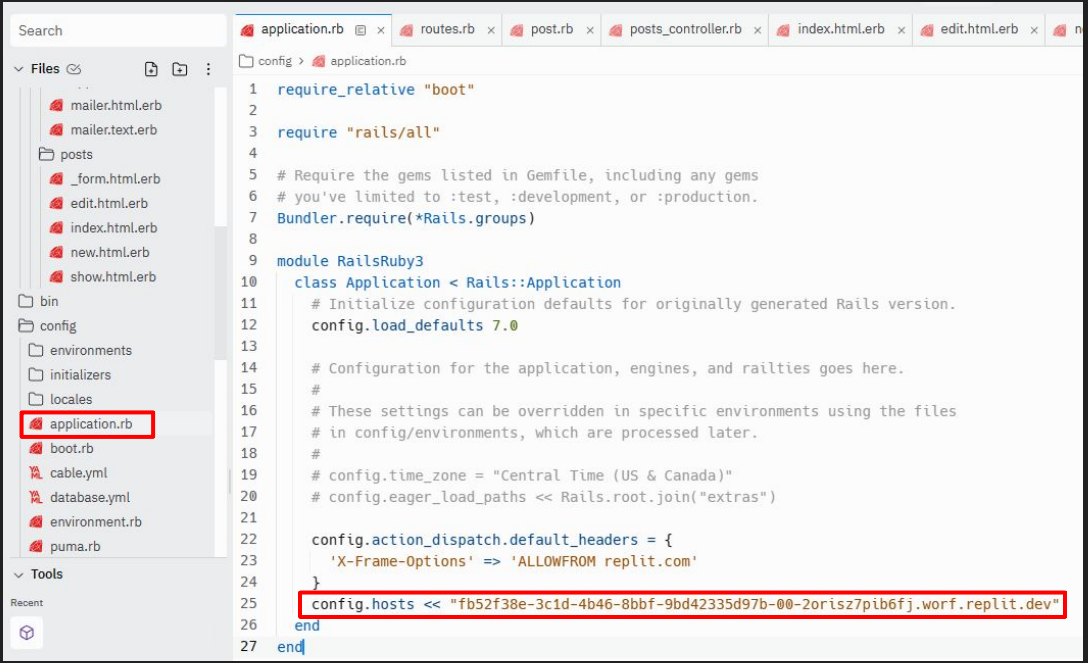
Copy (Ctrl + C) the output line outlined by red.

Blog

Starting a Server

Paste (Ctrl + V)

Open `config/application.rb`, and paste the line right below line 24.

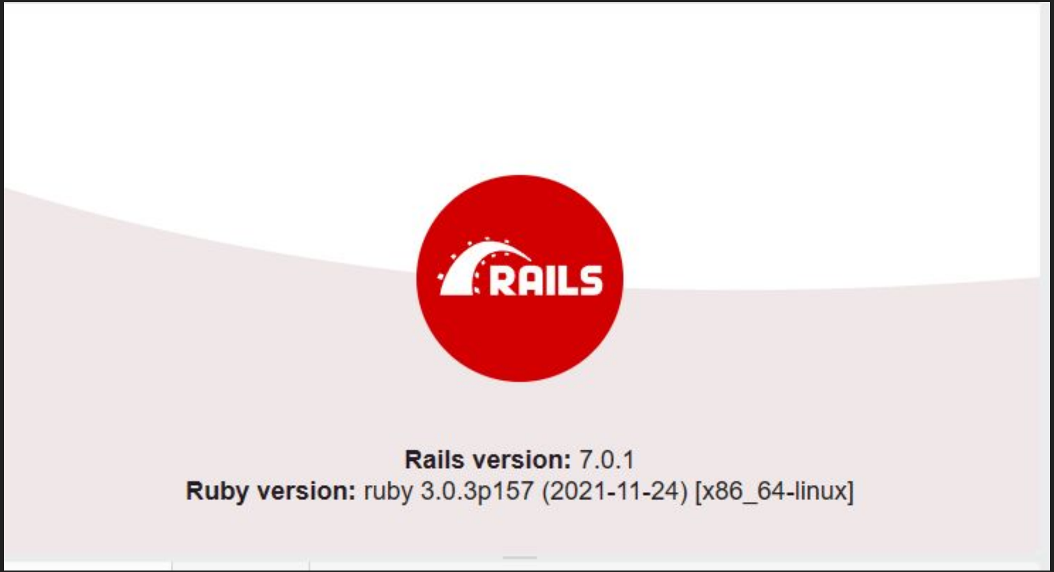


```ruby
1  require_relative "boot"
2
3  require "rails/all"
4
5  # Require the gems listed in Gemfile, including any gems
6  # you've limited to :test, :development, or :production.
7  Bundler.require(*Rails.groups)
8
9  module RailsRuby3
10   class Application < Rails::Application
11     # Initialize configuration defaults for originally generated Rails version.
12     config.load_defaults 7.0
13
14     # Configuration for the application, engines, and railties goes here.
15     #
16     # These settings can be overridden in specific environments using the files
17     # in config/environments, which are processed later.
18     #
19     # config.time_zone = "Central Time (US & Canada)"
20     # config.eager_load_paths << Rails.root.join("extras")
21
22     config.action_dispatch.default_headers = {
23       'X-Frame-Options' => 'ALLOWFROM replit.com'
24     }
25     config.hosts << "fb52f38e-3c1d-4b46-8bbf-9bd42335d97b-00-2orisz7pib6fj.worf.replit.dev"
26   end
27 end
```

Blog

Starting a Server

Click “Run” again, and it should look like this.



Rails version: 7.0.1
Ruby version: ruby 3.0.3p157 (2021-11-24) [x86_64-linux]

Note: If it takes a while to re-run the server, it is most likely due to Replit’s lack of bandwidth and editor quality.

Blog

Resources Theoretical Stuff

Now that we got the server to successfully start, we won't need to re-run it again.

We do want to create something to show when we reload the page though. We will use something called resources.

Resources are an object created using Ruby on Rails that can be stored in a database (an organized structure of data) and are a fundamental way of storing and transmitting information on a Ruby on Rails website.

Example: Articles → could be stored in a database, contains attributes (e.g. name, author, description, etc.)

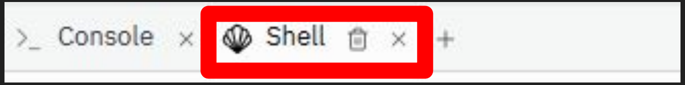
You might know what we're going to do with resources. Can you guess?

Blog

Create a Resource: Post

Question: What is the difference between backend and frontend?

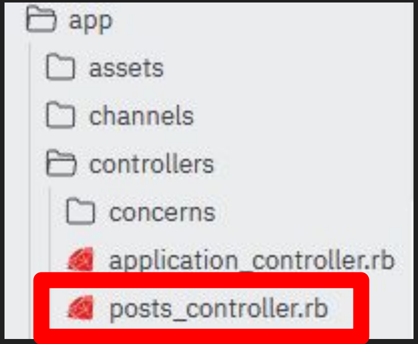
Shell time! To create a resource, we need to use a `rails` command to generate a controller, the resource's backend. Click the shell tab on the right-side window and type the following command:

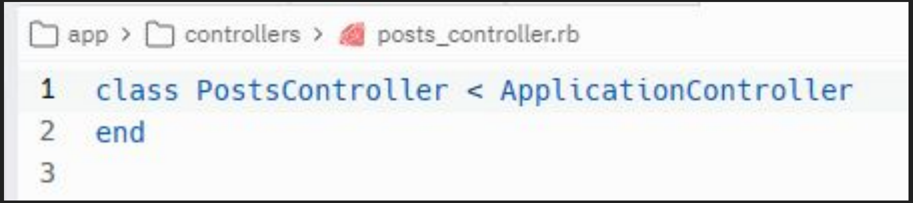


```
bin/rails generate controller Posts
```

This will actually also generate a frontend (view) file which we will look at later. We will first take a look at the controller file by navigating to

```
app/controllers/posts_controller.rb
```





```ruby
class PostsController < ApplicationController
end
```

Blog

Pages Theoretical Stuff

The first page we will create will correspond to the index of the post resource, which conventionally means the listing of all of its instances. An example of this is a user profile on Instagram, where all of their post instances are listed. However, this week, our index page will just say “Hello, world!”

We can create several pages that correspond to actions of the post resource, like creating a new instance, reading (or showing) a specific instance, updating a specific instance, or destroying a specific instance. These four actions make up the acronym CRUD, the four fundamental operations of almost every web application. Think of the analogy of Instagram and how these operations connect to what you do on it.

Blog

Create an Index Page

Setup:

(1) Define the `index` action in the controller and define data to send. This function *(backend)* sends the web page *(frontend)* the necessary information to list all the posts when triggered by a request from the server (whenever someone goes onto the website). This week, we won’t send anything yet.

(2) Create the `index.html.erb` page. It is the frontend code that formats the actual page. It is an embedded Ruby (erb) file, which means syntax exists to implement Ruby.

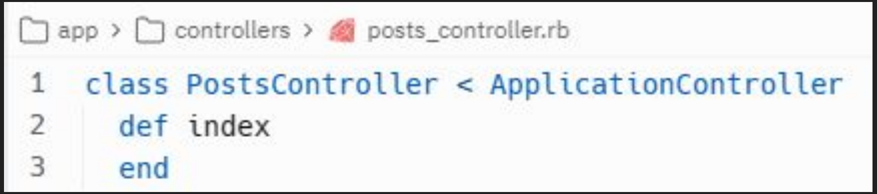
Blog

Create an Index Page

(1) Defining `index`

In the controller ruby file, right after the first line, type in:

```
def index
end
```



app › controllers › posts_controller.rb

```
class PostsController < ApplicationController
  def index
  end
```

Since we aren't going to send any post data to the view yet, `index` won't contain anything.

Blog

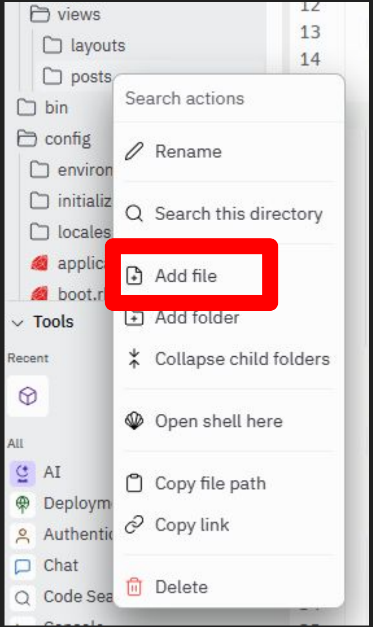
Create an Index Page

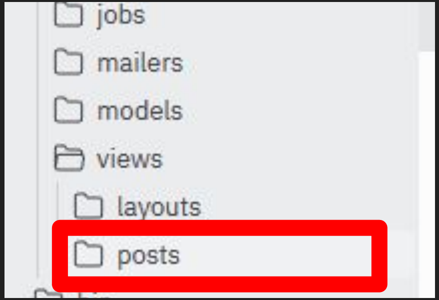
We will also take a look at the `views` folder. In it will include a `posts` folder that was created after generating the posts controller.

Right click the folder, click “Add file”, and name it `index.html.erb`.

Click into the file and type in

```
<p>Hello world!</p>
```





Blog

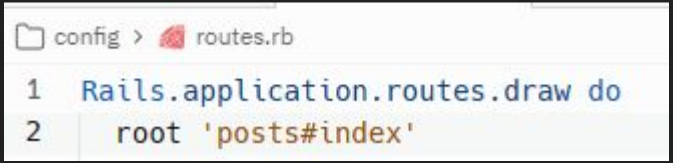
Routes

Routes are important because they allow customized links/URLs of the website.

We need to set up the root of the website (homepage). We will probably want to set it to the posts index page, so that in the future it will show all the posts (even though it just shows “Hello World” right now).

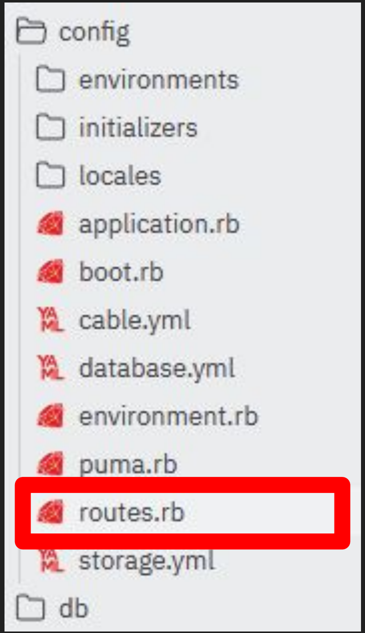
Head over to the `config/routes.rb` file and, after the first line, type in

```
root ‘posts#index’
```



config › routes.rb

```
Rails.application.routes.draw do
  root 'posts#index'
```

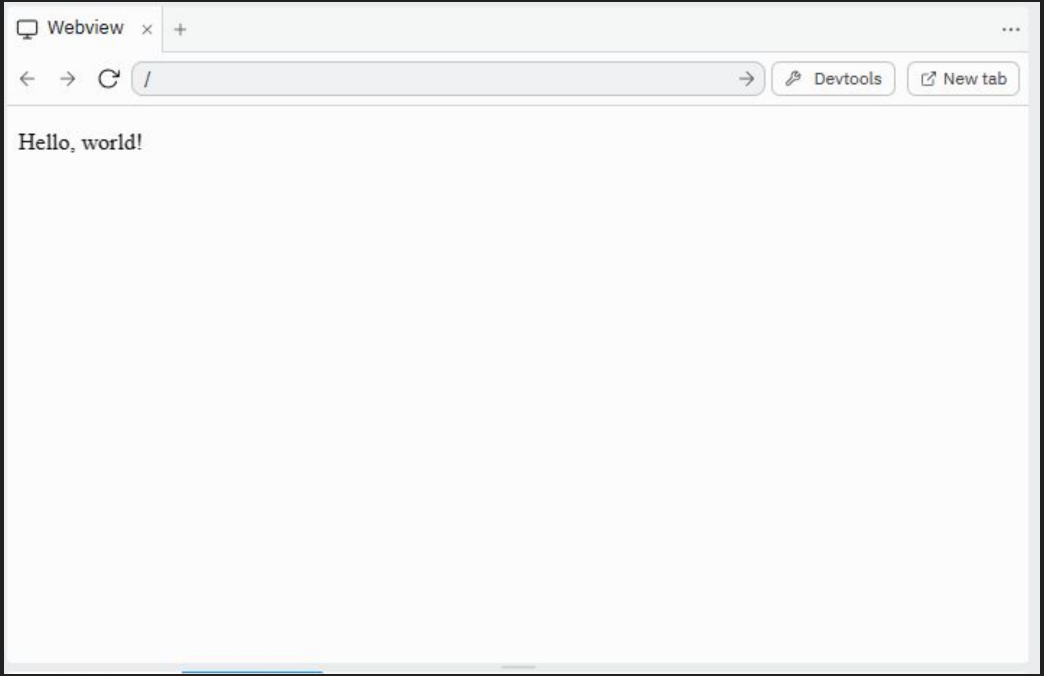


```
config
  environments
  initializers
  locales
  application.rb
  boot.rb
  cable.yml
  database.yml
  environment.rb
  puma.rb
  routes.rb
  storage.yml
db
```

Blog

Run

Click ‘Run’ again and this is what you should see:



```
Hello, world!
```

Blog

See you Next Time!

In the next part of this project (which will be uploaded onto the Hack Club website), we will cover showing and creating actual posts, frontend and backend.

Visit the Hack Club Website at: https://wlhackclub.github.io/