

COM 1001

# INTRODUCTION TO SOFTWARE ENGINEERING

**Professor Phil McMinn**

An Introduction to  
Web Application Development  
Sinatra Basics: Routes, Blocks, and HTTP

# Sinatra

A **Domain Specific Language (DSL)** for writing web applications in Ruby

It is meant to be easy to learn

It is designed so that developers can write web applications quickly

It has been used commercially by several organisations from Apple, to LinkedIn, to GitHub, to the NSA.

Website and documentation: <http://sinatrarb.com>





# Running a Hello World example in Sinatra

(You can see this again after the lecture by watching the video  
on the “Getting Started” started page on Blackboard.)

# Hello, World!

Sinatra involves writing blocks of Ruby code that determine which **HTTP Requests** the web application will respond to. These involve specifying a **HTTP method** and a **URL-matching pattern**, which together are called a **route**.

```
require "sinatra"  
  
get "/hello-world" do  
  "Hello, World!"  
end
```

We're using the sinatra gem

getting\_started/hello\_world.rb

The **URL-matching pattern** is *almost* the same thing as the **resource identifier** of a HTTP request, except the pattern can actually be used to specify more than one resource. (More on this later in the module.) In this case, the pattern maps to exactly one resource – “/hello-world” so the pattern and the resource identifier are essentially the same in this instance.

Ruby uses underscores as filename separators, but **URL-matching patterns should always use hyphens**. This might be confusing at first, but it means that the resulting URLs have hyphens rather than underscores too. This is important because it's hard to distinguish underscores from spaces when a URL is underlined. Google recommends hyphens as separators in URLs for this reason. This is known as “**kebab-case**”.

# The Base URL

The **Base URL** forms the initial part of every URL of your web application.

It is made up of the **HTTP protocol** being used, the **domain name** of your web server, and the **port number** it is using for your web application, as follows:

`http://localhost:4567`

This how it usually looks, if  
you were developing on  
your own machine

`https://box-name-4567.codio.io`

if you're developing with  
Codio (recommended)

You don't need to supply the port number if the default HTTP port (80) is being used. Since we tend to use port 80 for *deployed* web applications (as opposed to those being *developed*), you often don't see the port number in normal web usage.

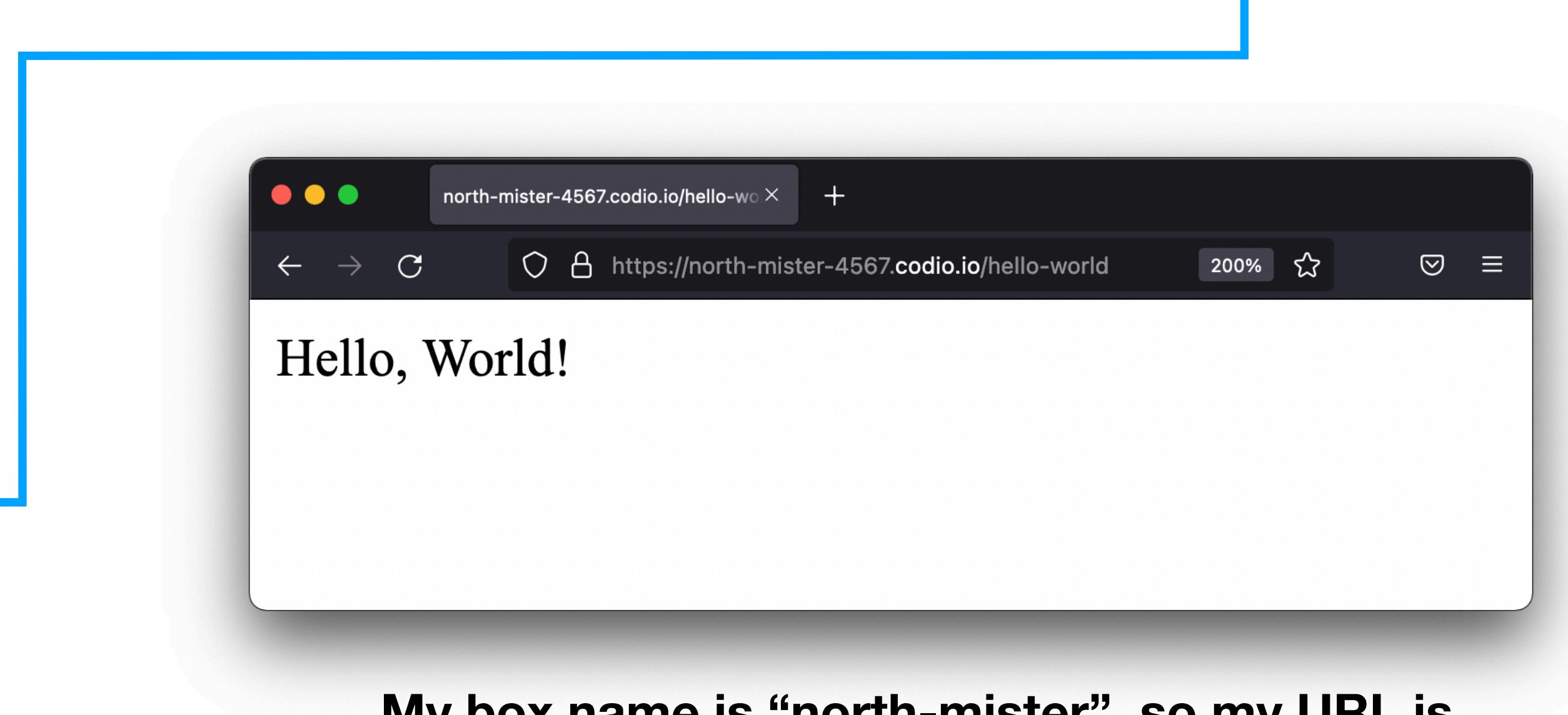
# Viewing the Hello World Example

To invoke a Sinatra `get` route, we need to suffix the **Base URL** with a string that matches the route's pattern, then type the whole URL into a browser

For the Hello World example, we need to add [/hello-world](#)

```
require "sinatra"

get "/hello-world" do
  "Hello, World!"
end
```



My box name is “north-mister”, so my URL is  
<https://north-mister-4567.codio.io/hello-world>

Your box will have a different name, so your URL will be different of course!

# The **HTTP Request** from My Browser

```
GET /hello-world HTTP/2
Host: north-mister-4567.codio.io
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.16; rv:84.0) Gecko/20100101 Firefox/84.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
...
...
```

# The **HTTP Response** from the Web Server

```
HTTP/2 200 OK
content-type: text/html; charset=utf-8
content-length: 13
...
...
```

# Sinatra Blocks

Sinatra **blocks** contain Ruby code specific to your web application

The block should return a string. The string forms the **content** of the **HTTP response** (i.e., a HTML web page) for the **HTTP request** that invokes the verb and route.  
As with Ruby methods, the **return** keyword is optional on the last line of the block.

```
require "sinatra"

TIMES_TABLE = 3
LIMIT = 10

get "/times-table" do
  title = "#{TIMES_TABLE} times table"
  output = "<html><head><title>#{title}</title></head>"
  output += "<body><h1>#{title}</h1><ul>"
  (1..LIMIT).each do |i|
    result = i * TIMES_TABLE
    text = "#{i} times #{TIMES_TABLE} = #{result}"
    output += "\n\t\t<li>#{text}</li>"
  end
  output += "</ul></body></html>"
  output
end
```

# Multiple Routes and Blocks

```
require "sinatra"

get "/first-route" do
  "This code is run when first-route is invoked"
end

get "/second-route" do
  "This code is run when second-route is invoked"
end
```

basics/multiple\_routes.rb

Suppose the base URL of your web application is <https://your-box-4567.codio.io>  
The two blocks above would be executed by the following URLs:

<https://your-box-4567.codio.io/first-route>  
<https://your-box-4567.codio.io/second-route>

# The Effect of Using puts

The `puts` method can still be used  
– it outputs strings to the console.

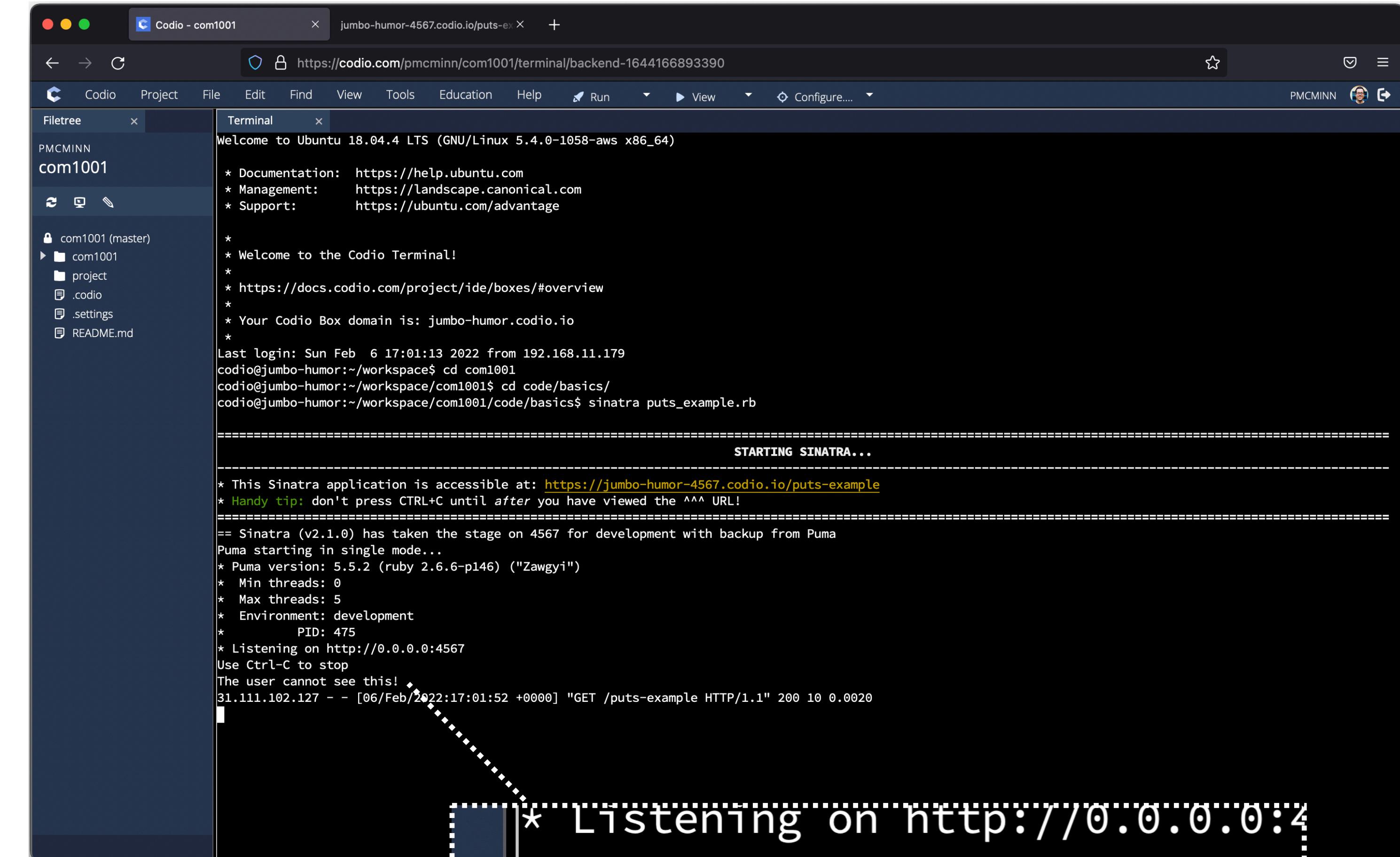
The user of the web page cannot  
see this output of course, as this  
output exists on the server.

It is therefore useful for **debugging**  
and other **status messages**.

```
require "sinatra"

get "/puts-example" do
  puts "The user cannot see this!"
  "Hi, folks!"
end
```

basics/puts\_example.rb



A screenshot of a Codio terminal window titled "Terminal". The window shows the output of a Sinatra application. The logs include:

- Welcome to Ubuntu 18.04.4 LTS (GNU/Linux 5.4.0-1058-aws x86\_64)
- \* Documentation: <https://help.ubuntu.com>
- \* Management: <https://landscape.canonical.com>
- \* Support: <https://ubuntu.com/advantage>
- \*
- \* Welcome to the Codio Terminal!
- \*
- \* <https://docs.codio.com/project/ide/boxes/#overview>
- \*
- \* Your Codio Box domain is: jumbo-humor.codio.io
- \*
- Last login: Sun Feb 6 17:01:13 2022 from 192.168.11.179
- codio@jumbo-humor:~/workspace\$ cd com1001
- codio@jumbo-humor:~/workspace/com1001\$ cd code/basics/
- codio@jumbo-humor:~/workspace/com1001/code/basics\$ sinatra puts\_example.rb
- =====
- STARTING SINATRA...
- =====
- \* This Sinatra application is accessible at: <https://jumbo-humor-4567.codio.io/puts-example>
- \* Handy tip: don't press CTRL+C until after you have viewed the ^^^ URL!
- =====
- Sinatra (v2.1.0) has taken the stage on 4567 for development with backup from Puma
- Puma starting in single mode...
- \* Puma version: 5.5.2 (ruby 2.6.6-p146) ("Zawgyi")
- \* Min threads: 0
- \* Max threads: 5
- \* Environment: development
- \* PID: 475
- \* Listening on http://0.0.0.0:4567
- Use Ctrl-C to stop
- The user cannot see this!
- 31.111.102.127 - - [06/Feb/2022:17:01:52 +0000] "GET /puts-example HTTP/1.1" 200 10 0.0020

\* Listening on http://0.0.0.0:4567  
Use Ctrl-C to stop  
The user cannot see this!  
31.111.102.127 - - [06/Feb/2022:17:01:52 +0000] "GET /puts-example HTTP/1.1" 200 10 0.0020

# HTTP Methods

HTTP methods prefix the route. We will only be using the `get` and `post` verbs with Sinatra.

```
require "sinatra"

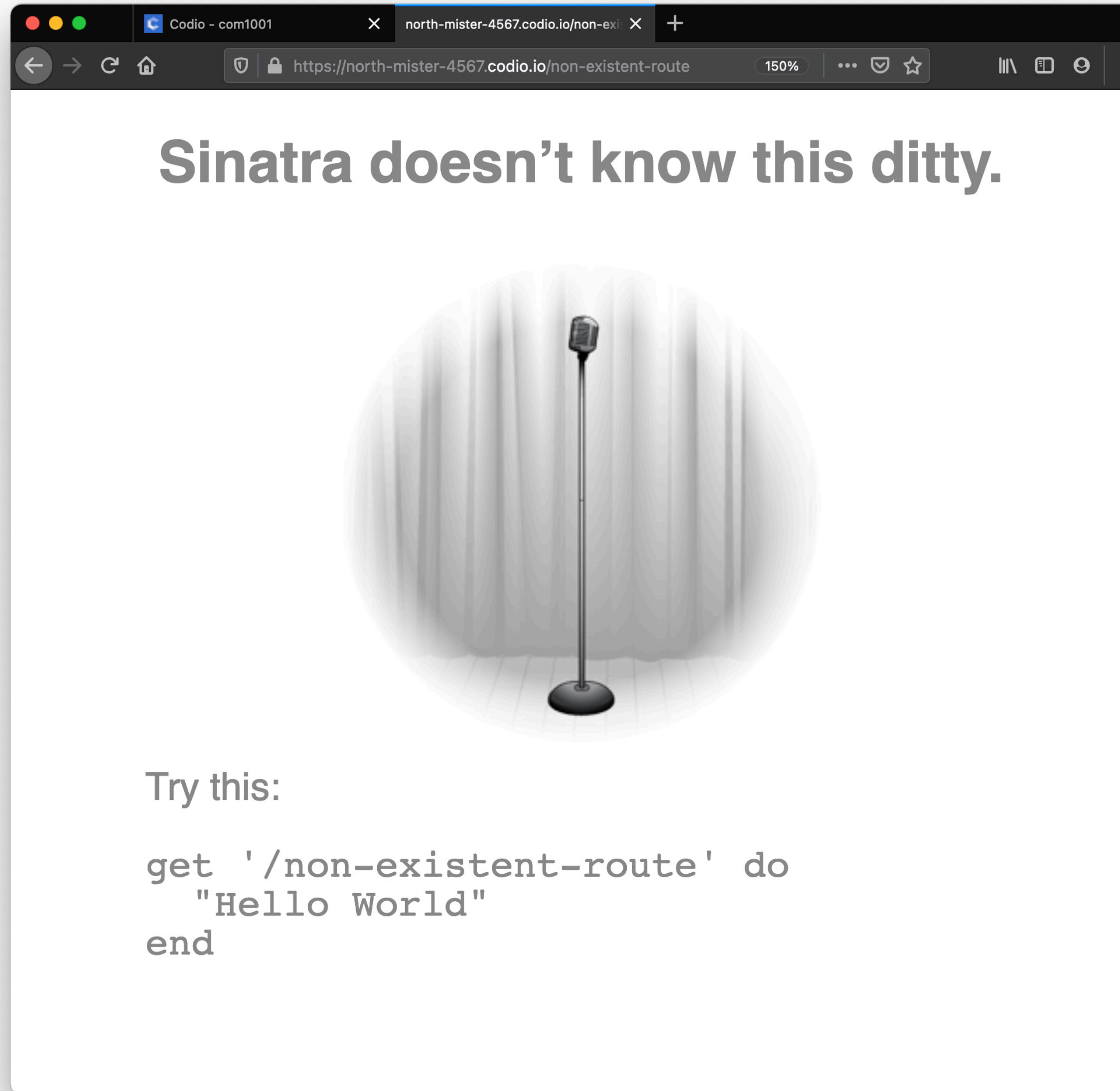
get "/get-route" do
  "This route is accessed using the HTTP GET method"
end

post "/post-route" do
  "This route is accessed using the HTTP POST method. " \
  "It is inaccessible by typing the URL into your browser."
end
```

`basics/http_methods.rb`

We can only access `get` routes by typing URLs into our browser. We will discuss how to use `post` in later in the module.

# When a HTTP Request Doesn't Map to a Route



If Sinatra cannot map the HTTP request to a block with a matching route, it triggers a **HTTP 404 Not Found** status and returns this page.

This might be due to a problem with our Ruby code, or it could be that a user mistyped the URL into their browser.

Either way, we would want them to see something more user friendly that does not reveal underlying development details. We will learn how to do this later in the module.

# When the Application Contains a Fatal Bug

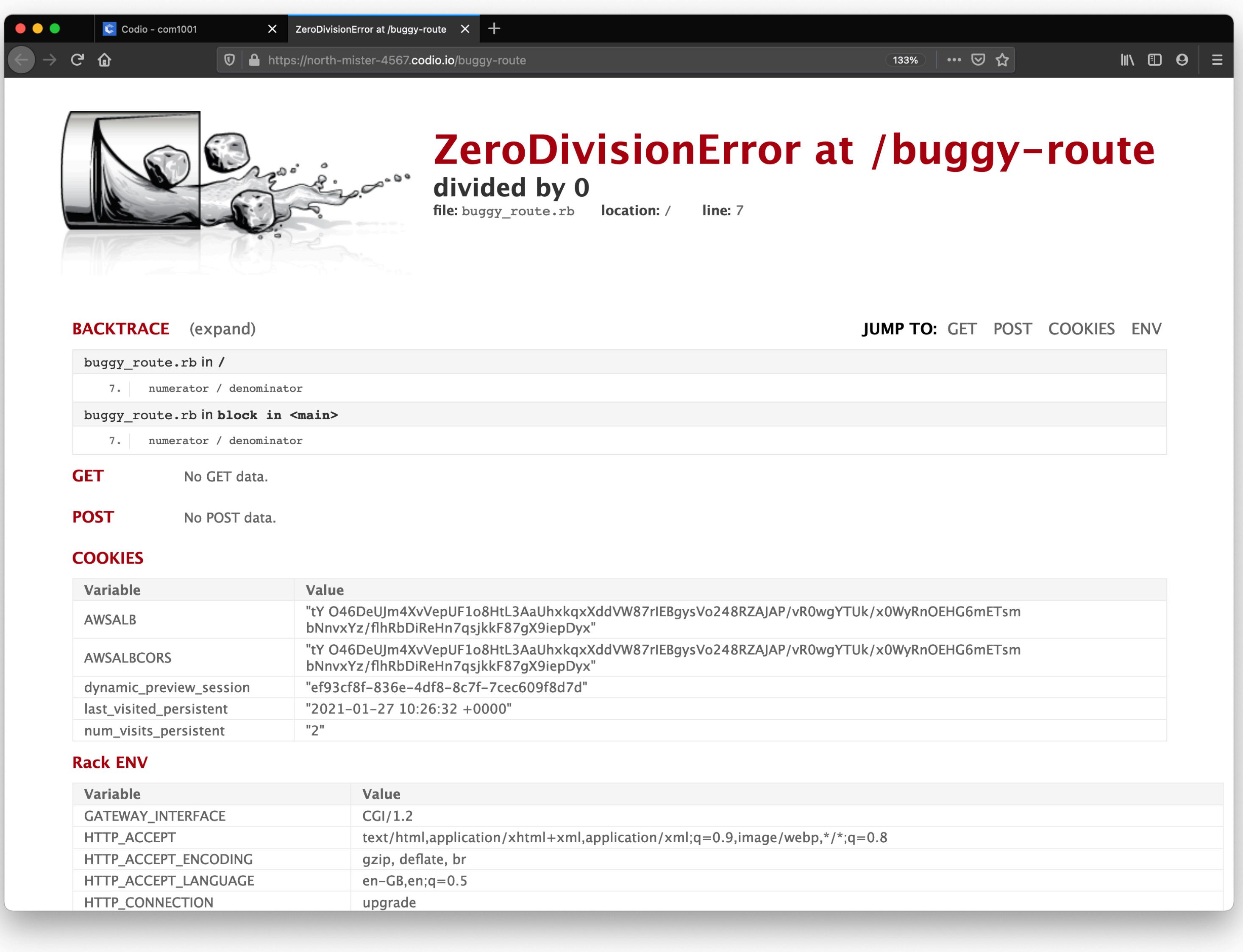
```
require "sinatra"
set :bind, "0.0.0.0"

get "/buggy-route" do
  numerator = 5
  denominator = 0
  numerator / denominator
end
```

basics/buggy\_route.rb

When this route is  
executed, a  
**ZeroDivisionError**  
will occur

# When the Application Contains a Fatal Bug



The screenshot shows a web browser window with the title "ZeroDivisionError at /buggy-route". The error message is "ZeroDivisionError at /buggy-route divided by 0" with "file: buggy\_route.rb" and "location: / line: 7". Below the error message is a cartoon illustration of a hand holding a knife, with liquid splashing out. A "BACKTRACE" section shows the stack trace:

```
buggy_route.rb in /
  7. |   numerator / denominator
buggy_route.rb in block in <main>
  7. |   numerator / denominator
```

Under "JUMP TO:", there are links for GET, POST, COOKIES, and ENV. Below the backtrace, there are sections for "GET" (No GET data), "POST" (No POST data), and "COOKIES". The "COOKIES" table lists variables and their values:

Variable	Value
AWSALB	"tY O46DeUJm4XvVepUF1o8HtL3AaUhxkqxXddVW87rIEBgysVo248RZAJAP/vR0wgYTUk/x0WyRnOEHG6mETsm bNnvxYz/flhRbDiReHn7qsjkkF87gX9iepDyx"
AWSALBCORS	"tY O46DeUJm4XvVepUF1o8HtL3AaUhxkqxXddVW87rIEBgysVo248RZAJAP/vR0wgYTUk/x0WyRnOEHG6mETsm bNnvxYz/flhRbDiReHn7qsjkkF87gX9iepDyx"
dynamic_preview_session	"ef93cf8f-836e-4df8-8c7f-7cec609f8d7d"
last_visited_persistent	"2021-01-27 10:26:32 +0000"
num_visits_persistent	"2"

Finally, there is a "Rack ENV" table:

Variable	Value
GATEWAY_INTERFACE	CGI/1.2
HTTP_ACCEPT	text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
HTTP_ACCEPT_ENCODING	gzip, deflate, br
HTTP_ACCEPT_LANGUAGE	en-GB,en;q=0.5
HTTP_CONNECTION	upgrade

Sinatra encounters an error and so triggers a **HTTP 500 Internal Server Error** status message. In this instance, Sinatra usefully presents us with diagnostic information that may help with debugging.

Again, we would not want the user to see this kind of information once we've deployed the app. We'll return to this later in the module.

# Stopping and Starting...

**When you change your code, the changes won't take effect until you stop and restart the web server.**

This is a bit of pain during the course of normal development. Using `sinatra/reloader` means you don't have to stop and restart the web server after every change. It listens for changes and reloads the application on the fly.

All you need to do is include this line of code in your app, and ensure the `sinatra-contrib` gem is installed.

```
require "sinatra"
require "sinatra/reloader"

get "/reloader-example" do
  "Change me, save, and reload the web page"
end
```

`basics/reloader-example.rb`

```
source "https://rubygems.org" do
  # ...
  gem "sinatra"
  gem "sinatra-contrib"
end
```

`basics/Gemfile`