University of Sheffield

COM1001 SPRING SEMESTER
Professor Phil McMinn
p.mcminn@sheffield.ac.uk

The `post` HTTP method – and when to use it instead of `get`

The `post` Form Submission Method

```
<html>
  <head>
    <title>POST Form Example</title>
    <link rel="stylesheet" href="style/style.css">
  </head>
  <body>
    <form method="post" action="/process-post-form">
      <p>
        Add some text into this box and press "submit": <br />
        <input type="text" name="text_field" />
      </p>
      <p><input type="submit" value="Submit"></p>
    </form>
  </body>
</html>
```

The `post` form submission method is an alternative to `get` that does not expose form data as part of the submission URL.

To use the `post` method, we set it as the `method` attribute in the form

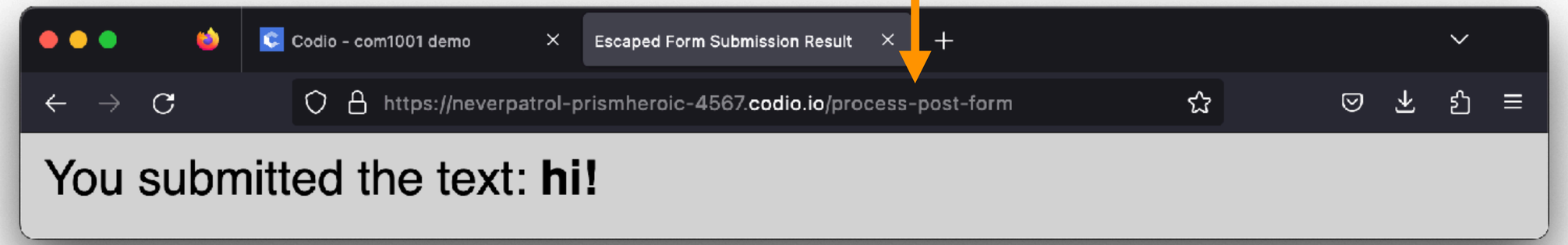
The `post` HTTP method

In order to handle the form submission sent using the `post` method, we need to use the `post` verb to prefix the route in our Sinatra app.

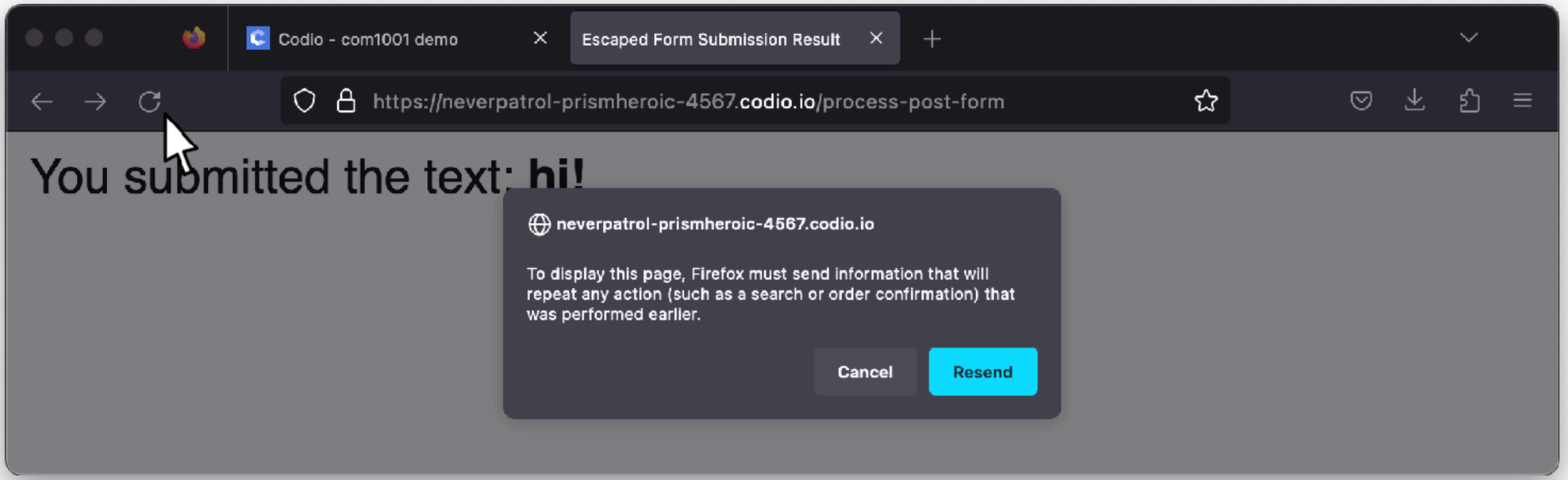
```
get "/post-form" do
  erb :post_form
end

post "/process-post-form" do
  @submitted_text_field_value = params["text_field"]
  erb :escaped_form_submission
end
```

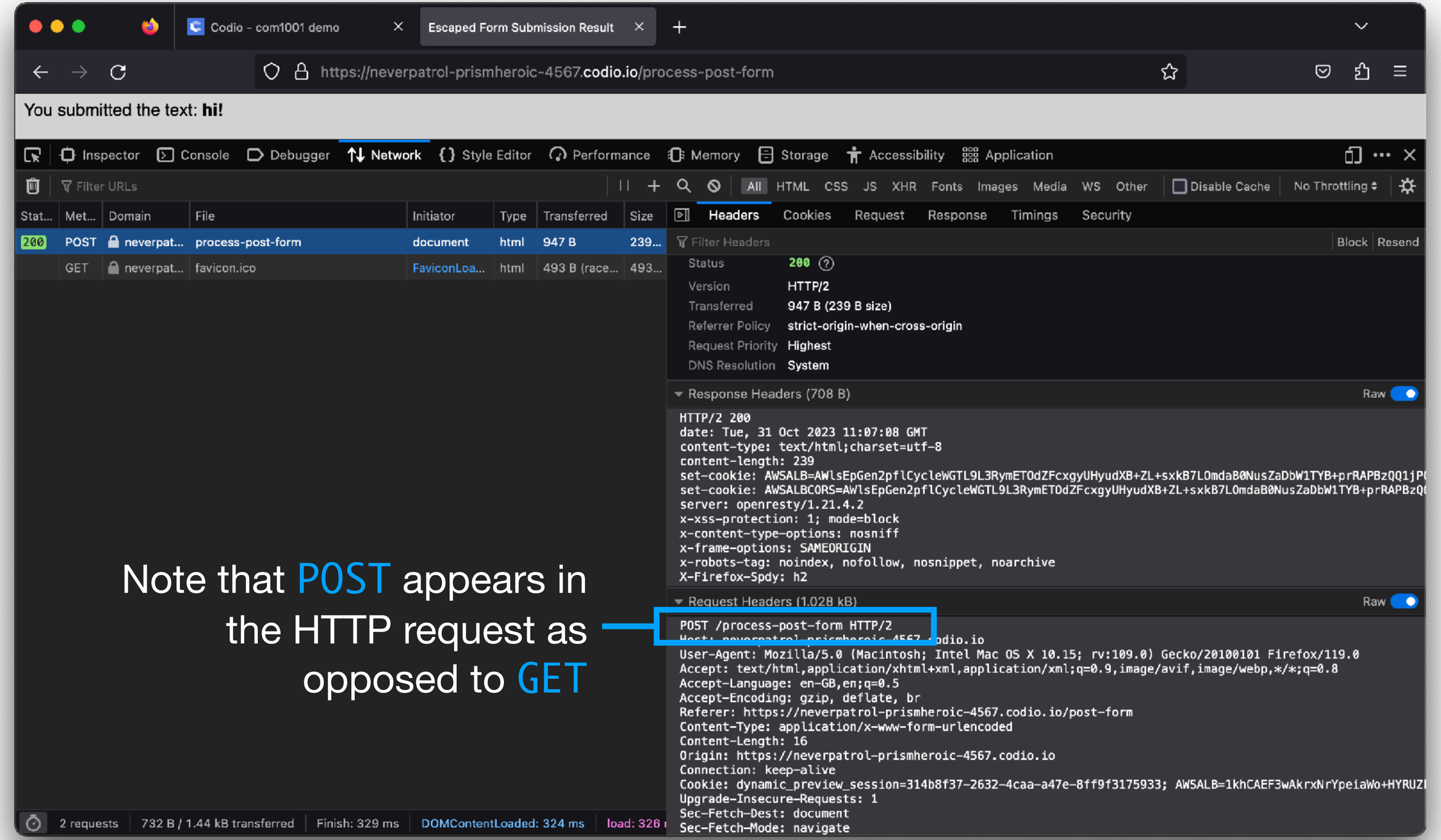
This route is inaccessible by typing the URL into your browser, since the browser will only generate a `get` request, which will not match this route, since it uses `post` instead.

The `post` form submission method does not expose form data as part of the query in the URL



Users cannot resubmit the form without the browser specially asking them whether they wish to repeat the action:




Codio - com1001 demo
Escaped Form Submission Result
https://neverpatrol-prismheroic-4567.codio.io/process-post-form
You submitted the text: hi!
Inspector
Console
Debugger
Network
Style Editor
Performance
Memory
Storage
Accessibility
Application
Filter URLs
All HTML CSS JS XHR Fonts Images Media WS Other
Disable Cache
No Throttling
Stat... Met... Domain File Initiator Type Transferred Size
200 POST neverpat... process-post-form document html 947 B 239...
GET neverpat... favicon.ico FaviconLoa... html 493 B (race... 493...
Headers Cookies Request Response Timings Security
Filter Headers
Block Resend
Status 200
Version HTTP/2
Transferred 947 B (239 B size)
Referrer Policy strict-origin-when-cross-origin
Request Priority Highest
DNS Resolution System
Response Headers (708 B)
Raw
HTTP/2 200
date: Tue, 31 Oct 2023 11:07:08 GMT
content-type: text/html;charset=utf-8
content-length: 239
server: openresty/1.21.4.2
x-xss-protection: 1; mode=block
x-content-type-options: nosniff
x-frame-options: SAMEORIGIN
x-robots-tag: noindex, nofollow, nosnippet, noarchive
X-Firefox-Spdy: h2
Request Headers (1.028 kB)
Raw
POST /process-post-form HTTP/2
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0) Gecko/20100101 Firefox/119.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://neverpatrol-prismheroic-4567.codio.io/post-form
Content-Type: application/x-www-form-urlencoded
Content-Length: 16
Origin: https://neverpatrol-prismheroic-4567.codio.io
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
2 requests
732 B / 1.44 kB transferred
Finish: 329 ms
DOMContentLoaded: 324 ms
Note that POST appears in the HTTP request as opposed to GET

Characteristics of a `get` Form Submission

Since the data is displayed in the query part of the URL, the form submission…

- Can be bookmarked (useful for repeating search queries)
- Remains in the browser's history
- Can be cached by the browser (no need to request the page repeatedly)

But:

- Data is restricted to **text only** (ASCII)
- **Security** is very weak – data can be stored in server logs etc.

Characteristics of a `post` Form Submission

Since the data is *not* displayed in the URL, the form submission…

… cannot be bookmarked (useful for repeating search queries)

… does not remain in the browser's history

… cannot be cached by the browser

However:

- Data is can be **binary** (allows for document uploads…)
- **Security** is stronger – especially when connections are encrypted with SSL – useful for logging in with confidential credentials.

Crafting a `get` Query String

```ruby
require "uri"

get "/construct-querystring" do
  person = { "name" => "Phil McMinn",
             "job" => "Professor of Software Engineering",
             "address" => "Regent Court",
             "age" => "That would be telling..." }

  @querystring = URI.encode_www_form(person)

  erb :construct_querystring
end
```

Sometimes it's useful to pass values from one page to another via query in the URLs in `<a href="...">` tags. We can turn any arbitrary hash into a query string using the `URI.encode_www_form` method

```
<a href="/process-querystring?<%= h @querystring %>">Click me!</a>
```

We always need to escape the query string in the view, since "&" (the key-value separator in a query string) is a HTML special character. This will only work properly if we've not already escaped any key-value pairs.

The value of `@querystring` is:

`/process-querystring?name=Phil+McMinn&job=Professor+of+Software+Engineering&address=Regent+Court&age=That+would+be+telling...`

```ruby
get "/process-querystring" do
  @name = params["name"]
  @job = params["job"]
  @address = params["address"]
  @age = params["age"]

  erb :process_querystring
end
```

The route corresponding to the URL we sent the query string with can then unpack the `params` hash and get the values we originally sent.

Note this code is no different to that had the values had been submitted by a form – the receiving route has no idea of the context, or how the data got in the `params` hash – just that it is there!

get or post?

	get	post
Cachable by Browser?	Yes	No
Remain in Browser History?	Yes	No
Bookmarkable?	Yes	No
Restriction on Length?	Yes	No
Restriction on Data?	ASCII only	Binary allowed
Data displayed in query of URL?	Yes	No
Security	Weak – data part of URL, can be cached, bookmarked, stored in web server logs etc.	Stronger – especially when connections are encrypted with SSL

How to Decide Whether to Use `get` or `post`

`get` works well for search queries on insensitive data:

- queries can be bookmarked
- direct URLs can be constructed for linking to specific search results (queries can be added to the URLs of `<a href="...">` links)

`post` works best when a user needs to:

- submit sensitive data (e.g. logging into a system)
- or is providing one-time information (e.g., job application data) or performing a one-time action (e.g., deleting some data)