So here is a working example of adding a `POST` handler to a `{shiny}` app, using `{brochure}`.

```
library(shiny)
library(brochure)
# We'll start by defining a home page
home <- function(
  httr_code
){
  page(
    href = "/",
    # Simple UI, no server side
    ui = tagList(
      tags$h1("Hello world!"),
      tags$p("Open a new R console and run:"),
      tags$pre(
        httr_code
      )
    )
  )
}

postpage <- function(){
  page(
    href = "/post",
    # We'll handle POST requests via a request handler
    req_handlers = list(
      function(req){
        # This is where the magic happens
        # Our req object contains a `REQUEST_METHOD`
        # entry that contains the HTTP verb used
        # to perform the request
        if( req$REQUEST_METHOD == "POST" ){
          print("In POST!")
          # Because we want the HTTP request to be
          # completed here, we return an httpResponse object here.
          # httpResponse() is exported since {shiny} 1.6.0,
          # otherwise you'll have to ::: (shiny:::httpResponse)
          return(
            httpResponse(
              # 201 is the HTTP code you'll send back when
              # you have created a resource on the server
              status = 201,
              content = "ok"
            )
          )
        }
        # Whenever we're not in a POST, we'll simply return
        # req, which will the move to standard {shiny} handling,
        # i.e. calling ui and server.
        return(req)
```

```
      }
    ),
    ui = tagList(
      tags$p("Hello from /post!")
    )
  )
}
# For the sake of reproducibility:
options(shiny.port = 2811)

brochureApp(
  home(
    httr_code = "httr::POST('http://127.0.0.1:2811/post')"
  ),
  postpage()
)
```

If you open another terminal, you can run:

```
> httr::POST("http://127.0.0.1:2811/post")

Response [http://127.0.0.1:2811/post]
  Date: 2021-02-28 21:47
  Status: 201
  Content-Type: text/html; charset=UTF-8
  Size: 2 B
```

And your R console running the `{shiny}` app should print a message:

```
Listening on http://127.0.0.1:2811
[1] "In POST!"
```

Of course, what's interesting with `POST` is that you can actually send a `body` along the http request. Using `{brochure}`, you can access it inside the request handler, via `req$.bodyData`. It's an external pointer to a file, so we can read it with `readLines`.

```
str(req$.bodyData)
 'file' int 3
 - attr(*, "conn_id")=
```

Let's add this to our app:

```
postpage <- function(){
  page(
    href = "/post",
    req_handlers = list(
      function(req){
        if( req$REQUEST_METHOD == "POST" ){
          # reading the req$.bodyData.
          body <- readLines(
            req$.bodyData,
            # The file will have no final EOL
            # and we don't want a message to be
```

```
            # printed to the console
            warn = FALSE
          )
          # Here, you can define your own handling
          # logic of the body.
          # We'll simply print it to the console
          # in this example.
          print(body)
          return(
            httpResponse(
              status = 201,
              content = "Created"
            )
          )
        }
        return(req)
      }
    ),
    ui = tagList(
      tags$p("Hello from /post!")
    )
  )
}
brochureApp(
  home(
    httr_code = "httr::POST('http://127.0.0.1:2811/post', body = 'plop')"
  ),
  postpage()
)
```

Now inside another console, send:

```
> (resp <- httr::POST("http://127.0.0.1:2811/post", body = "plop"))
Response [http://127.0.0.1:2811/post]
  Date: 2021-02-28 21:57
  Status: 201
  Content-Type: text/html; charset=UTF-8
  Size: 7 B
```

The R console running the `{shiny}` app should print a message:

```
Listening on http://127.0.0.1:2811
[1] "plop"
```

Back to the other console, the `content()` of the `{httr}` `resp` will be the content defined in the `httpResponse`:

```
> httr::content(resp)
{html_document}

[1]
```

Created

Hope this helps!