

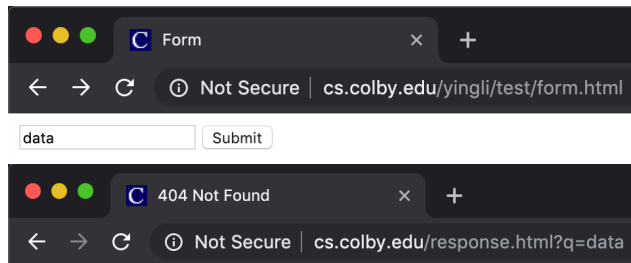
Application Layer (V)

HTTP POST Method

- POST
 - client informs server about data to be delivered
 - client sends data
 - e.g., submit an online form

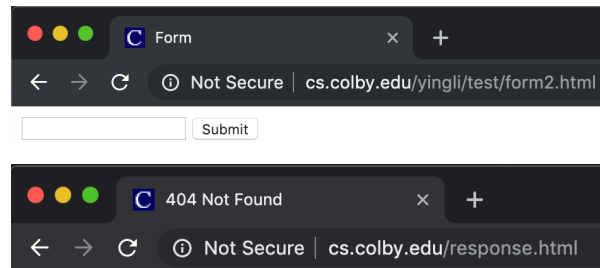
Compare POST with GET

- request form1.html and form2.html on the CS department web server from a web browser and show the difference on the URL.
 - form1: input data is in the URL
 - form2: input data is not in the URL



Not Found

The requested URL /response.html was not found on this server.



Not Found

The requested URL /response.html was not found on this server.

- show form1.html and form2.html

```

form.html
< > form.html > No Selection
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Form</title>
5 </head>
6 <body>
7   <form action="/response.html" method="GET">
8     <input type="text" name="q">
9     <input type="submit">
10  </form>
11 </body>
12 </html>

```

```

form2.html > No Selection
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Form</title>
5 </head>
6 <body>
7   <form action="/response.html" method="POST">
8     <input type="text" name="q">
9     <input type="submit">
10 </body>
11 </html>

```

- form.html uses GET, so the submitted data was in the URL.
- form2.html uses POST, so the submitted data was sent to the server implicitly (not in the URL).

- But both requests couldn't get appropriate responses, as the CS department web server was not set up to response these requests.
- So how can we set up a server?

Server Architecture

- Generally speaking, the server accepts and process the incoming requests while listening at a port.
- Typically, there are two ways to process the accepted requests, sequentially and in parallel.
- The web servers in real life must accept concurrent requests. So, processing the requests in parallel is a more feasible way.
 - The server can allocate a process or thread to handle a request, the main process and thread of the server deals with accepting the incoming requests.
- Event-driven is another approach to handle concurrent requests.
 - The server does a **quick bit of processing** and **schedule a callback when** (typically) **I/O is done**.
 - Node.js (<https://nodejs.org/en/>) provides a framework to implement an event-driven server.
 - **Not for CPU intensive application**.

Node.js Example 1

- Show the `server1.js` code, run it, and request `localhost:8000` on web browser.
- `server1.js` is a very simple server that displays a string and nothing more than that.

```
var http = require('http');

http.createServer(
  function (req, resp) {
    resp.writeHead(200, {'Content-Type': 'text/plain'});
    resp.end('Hello World\n');
  }
).listen(8000);

console.log('Server running at http://localhost:8000/');
```

- To use the HTTP server and client one **must require('http')**
- **`http.createServer(requestHandler)`** returns a new instance of `http.Server`.
 - The `requestHandler` is a function that is called once a HTTP request is made against that server. Node calls the handler with two objects, request and response.
 - The request object has a `url` attribute that contains the pathname and the query.
 - The response object includes the HTTP status code, the response header, and the response content (e.g., the requested file).

- `writeHead(status code, header)` send a response header to the request. **Must be called before `resp.end()`.**
- `resp.end(data)` signals to the server that all of the response headers and body have been sent. **Must be called on each response.**
- Reference: <https://nodejs.org/en/docs/guides/anatomy-of-an-http-transaction/>
- Let's understand how both sides exchange information over HTTP using Wireshark.
 - open Wireshark and listen at Loopback
 - run `server1.js` locally
 - open a browser and type `http://localhost:8000/`
 - GET request is the packet with GET / HTTP/1.1
 - GET response is the packet with HTTP/1.1 200 OK

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	::1	::1	TCP	88	58994 → 8000 [SYN] Seq=2628888301 Win=65535 Len=0 MSS=16324 WS=64 TSval=3...
2	0.000249	::1	::1	TCP	88	8000 → 58994 [SYN, ACK] Seq=584812095 Ack=2628888302 Win=65535 Len=0 MSS=...
3	0.000292	::1	::1	TCP	76	58994 → 8000 [ACK] Seq=2628888302 Ack=584812096 Win=407744 Len=0 TSval=30...
4	0.000310	::1	::1	TCP	76	[TCP Window Update] 8000 → 58994 [ACK] Seq=584812096 Ack=2628888302 Win=4...
5	0.032829	::1	::1	HTTP	513	GET / HTTP/1.1
6	0.032921	::1	::1	TCP	76	8000 → 58994 [ACK] Seq=584812096 Ack=2628888739 Win=407360 Len=0 TSval=34...
7	0.033725	::1	::1	HTTP	255	HTTP/1.1 200 OK (text/plain)
8	0.033753	::1	::1	TCP	76	58994 → 8000 [ACK] Seq=2628888739 Ack=584812275 Win=407616 Len=0 TSval=30...
9	5.035291	::1	::1	TCP	76	8000 → 58994 [FIN, ACK] Seq=584812275 Ack=2628888739 Win=407360 Len=0 TSv...
10	5.035395	::1	::1	TCP	76	58994 → 8000 [ACK] Seq=2628888739 Ack=584812276 Win=407616 Len=0 TSval=30...
11	5.035544	::1	::1	TCP	76	58994 → 8000 [FIN, ACK] Seq=2628888739 Ack=584812276 Win=407616 Len=0 TSv...
12	5.035622	::1	::1	TCP	76	8000 → 58994 [ACK] Seq=584812276 Ack=2628888740 Win=407360 Len=0 TSval=34...

Node.js Example 2

- The server should listen to the client request and act according if we want some interaction like sending a form and getting some response.
- show the code of `server2.js`
 - the server listens at port 8000
 - response to the request from clients after parsing the request (pathname) and send the requested file back to the client.

```

var http = require('http');
var fs = require('fs');
var url = require('url');

// Create a server
http.createServer( function (req, resp) {
  // Parse the request containing file name
  var pathname = url.parse(req.url).pathname;
  // Print the name of the file for which request is made
  console.log("Request for " + pathname + " received.");

  // Read the requested file content from file system
  fs.readFile(pathname.substr(1), function (err, data) {
    if (err) {
      console.log(err);
      // HTTP Status: 404 : NOT FOUND
      // Content Type: text/plain
      resp.writeHead(404, {'Content-Type': 'text/html'});
    }
  })
})

```

```

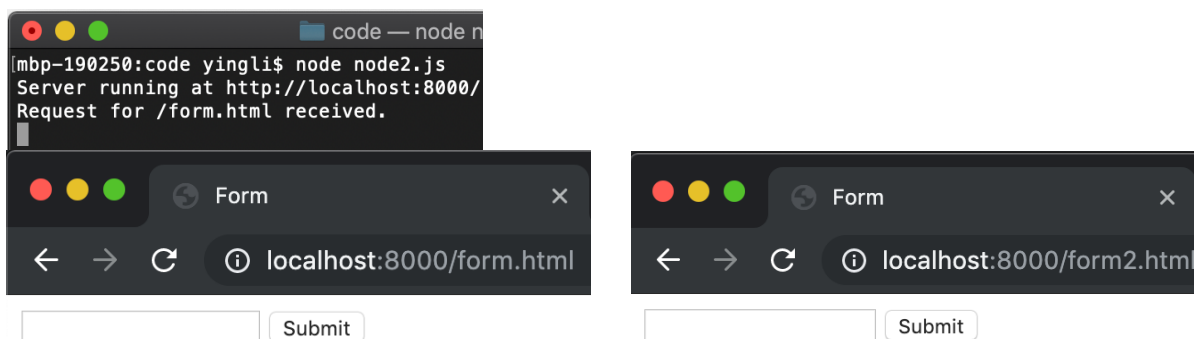
else {
  // Page found
  // HTTP Status: 200 : OK
  // Content Type: text/plain
  resp.writeHead(200, {'Content-Type': 'text/html'});

  // Write the content of the file to response body
  resp.write(data.toString());
}
// Send the response body
resp.end();
});
}.listen(8000);

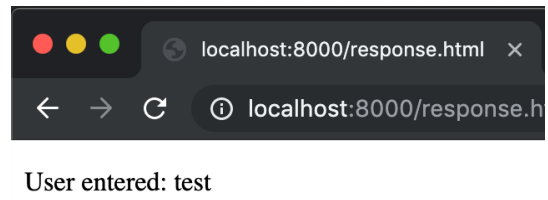
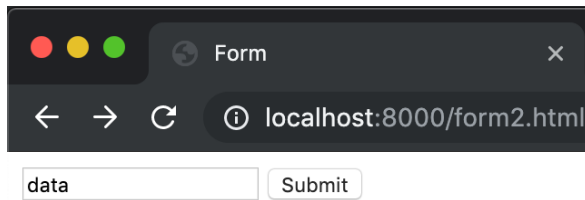
// Console will print the message
console.log('Server running at http://localhost:8000/');

```

- The module `fs` provides a variety of file system methods.
- The easiest way to read the entire file is using `readFile` method:
 - `fs.readFile(file, [encoding], [callback]);`
 - `file` is a string indicating the path of the file to read
 - `encoding` is an optional parameter that specifies the type of encoding to read the file, such as `'ascii'`, `'utf8'`, and `'base64'`. The default is `null` if no encoding is provided.
 - `callback` is a function to call when the file has been read and the contents are ready. It has two parameters, `error` and `data`. If there is no error, `error` will be `null` and `data` will contain the file content. Otherwise, `error` will contain the error message.
- References: <https://nodejs.org/en/knowledge/file-system/how-to-read-files-in-nodejs/>
- run `server2.js` and request `form1.html` and `form2.html` from web browser



- The `server2.js` server can send the responses to the web browser according to the requests.
- However, the response page cannot reflect the submitted data correctly. Submitted “data” and return “test”.



- Because the action field of the `<form>` in `form1.html` and `form2.html` is a statical html page, which cannot reflect the data submitted by the client.
- action defines the response after submission

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Form</title>
5 </head>
6 <body>
7   <form action="/response.html" method="POST">
8     <input type="text" name="q">
9     <input type="submit">
10 </body>
11 </html>

```

```

1 <!DOCTYPE html>
2 <html>
3 <p>User entered: test</p>
4 </html>

```

- To make an interactive webpage, there should be a way to transmit data between the client and server back and forth, even better, without impaction to the current page display (no need to reload the page for every submission).

Node.js Example 3

- Client side AJAX
 - send and retrieve data from a server asynchronously without interfering with the display and behavior of the existing page (open form3.html using browser to illustrate).
 - It's common to use GET in AJAX unless sending a sensitive data or calling script processing data on the server (open form3.html using a text editor).

```

<script>
  $("form").submit(function(f) {
    f.preventDefault();
    $.ajax({
      url: "/request",
      data: "q="+$("#q").val(),
      type: "GET",
      success: function(data) {
        var results = JSON.parse(data);
        $("#result").html(results["result"]);
      },
      error: function(e) {
        alert("Error: "+e.status+": "+e.statusText);
      }
    });
  });
</script>

```

GET /request?q=test HTTP/1.1

{"result": "server response"}