WiCyS: Intro to Web Security Part II

Recap from last week

- Briefly glossed over HTML/CSS syntax
- Wrote a basic TCP client and server in Javascript
- Slides available here:

https://wicysillinois.wordpress.com/resources/

- Code available here:

https://github.com/danielazieba/wicys-uiuc-fa20

Setting up a simple server in Node.js

```
const net = require('net')
const server = net.createServer(function
  (socket) {
    // handle connections here
    socket.end("hello!!!")
});
server.listen(8000)
```

Testing your server, by building a client

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

var client = new net.Socket(); // create client
// connect, specifying port, IP address to connect to
client.connect(process.argv[2], '127.0.0.1', function() {
   console.log('Connected to server at port 8000');
   client.write('Hello server!!');
});
// print data when we receive it
client.on('data', function(data) {
   console.log('Data received from the server: ' + data);
});
// print when server connection closed
client.on('close', function() {
   console.log('Connection closed');
});
```

Testing server/client

In one terminal, run **node my-server.js**

In another terminal, run **node my-client.js 8000**

Goals for today's workshop

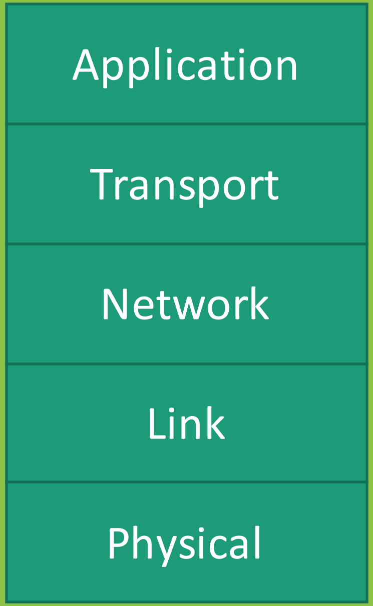
- Learn a bit more about what TCP actually is
- Learn basic networking
- Continue building a client-server with Javascript/Node.js
    - Make a server that serves the current time
    - Make a server that returns JSON
    - Make a server that returns an HTML web page

Why do we care about networking?

- The Internet, and all web pages, require networks
- Can be a major source of security vulnerabilities
- Useful to know about for network engineering positions, or even general software engineering
  - Using an API will frequently require capturing and debugging network traffic
  - Building an app will need confirmation that things are secure/working as intended

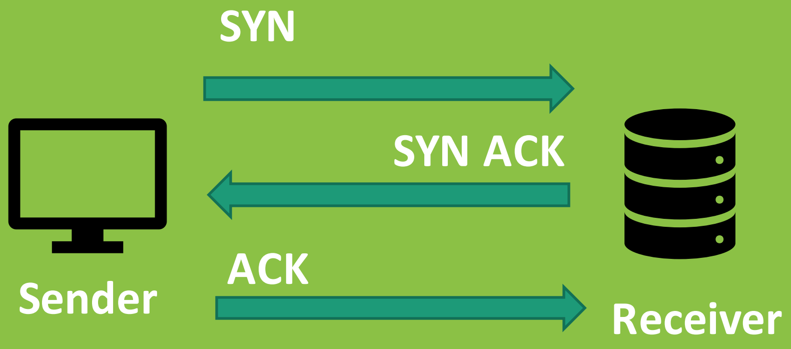
Source: CS/ECE438 FA2020 slides

How does the Internet work?

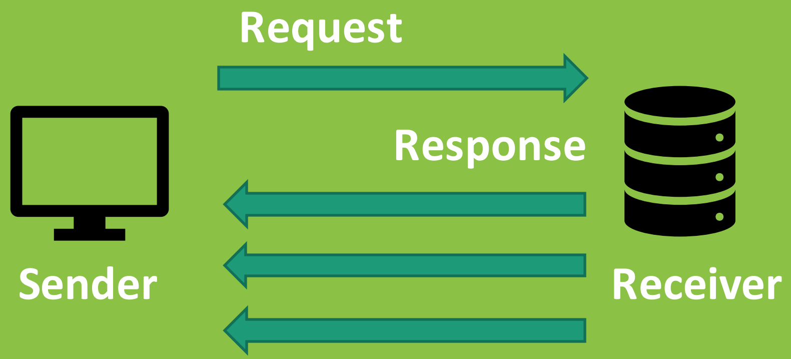


- A lot of things put together – especially cables, routers, networks of networks
- Specifically, has a stack of network protocols
  - Application layer: supporting network applications
    - FTP, SMTP, HTTP, DNS
  - Transport: host-host data transfer
    - TCP, UDP, ...
  - Network: routing of datagrams from source to destination
    - IP, routing protocols, etc.
  - Link: data transfer between neighboring network elements
    - Ethernet, WiFi, Bluetooth
  - Physical: bits "on the wire"
    - OFDM, DSSS, CDMA, Coding, ...

What are TCP and UDP?



TCP (top) vs. UDP (bottom)



- TCP = Transmission Control Protocol
  - Requires a "handshake" to establish connection
  - Prioritizes accuracy of sent data
- UDP = User Datagram Protocol
  - No handshaking
  - Unreliable, may drop packets, but faster
- Other protocols like HTTP are built on top of TCP

What are TCP and UDP?

- Other protocols like HTTP are built on top of TCP
- We could define our own protocols for TCP data sent between our servers
  - For example: our protocol could be such that every string sent starts with the number of bytes in the rest of the message
  - We could look at devices' network traffic, if unencrypted figure out the request structure needed to brew a cup of coffee [0]

Our theoretical protocol (most will have more complexity than this):

32 bits	Some other quantity of bits
Number of bytes	Actual message content here

[0] https://decoded.avast.io/martinhron/the-fresh-smell-of-ransomed-coffee/

Setting up a simple server in Node.js

```
const net = require('net')
const server = net.createServer(function
  (socket) {
   // handle connections here
   socket.end("hello!!!")
});
server.listen(8000)
```

How can we write the current time?

Setting up a simple server in Node.js

```
const net = require('net')
const server = net.createServer(function
  (socket) {
   // handle connections here
   socket.end("hello!!!")
});
server.listen(8000)
```

To get the date,

```
var curr_date = new Date();
```

Setting up a simple server in Node.js

We want to display the time in format "YYYY-mm-dd hh:mm" and add a new line at the end.

Setting up a simple server in Node.js

We want to display the time in format "YYYY-mm-dd hh:mm" and add a new line at the end.

Some functions that might be useful:

```
date.getFullYear()
date.getMonth() // starts at 0
date.getDate() // returns the day of month
date.getHours()
date.getMinutes()
```

Setting up a simple server in Node.js

Concatenating strings in Javascript?

Can be done like Python -

```
var five = 5;
var ten = 10;
var my_string = "five plus ten is " + (five + ten)
  + ".";
```

my_string will be "five plus ten is 15."

Setting up a simple server in Node.js

```
var curr_date = new Date();
var curr_timestamp = curr_date.getFullYear()
+ "-"
    + curr_date.getMonth() + "-"
    + curr_date.getDate() + " "
    + curr_date.getHours + ":"
    + curr_date.getMinutes() + "\n";
```

Setting up a simple server in Node.js

```
const net = require('net')
const server = net.createServer(function  (socket) {
    // handle connections here
    var curr_date = new Date();
    var curr_timestamp = curr_date.getFullYear() + "-"
        + curr_date.getMonth() + "-"
        + curr_date.getDate() + " "
        + curr_date.getHours + ":"
        + curr_date.getMinutes() + "\n";
    socket.end(curr_timestamp)
});
server.listen(8000)
```

Setting up a simple server in Node.js

```
const net = require('net')
const server = net.createServer(function (socket) {
  // handle connections here
  var curr_date = new Date();
  var curr_timestamp = curr_date.getFullYear() + "-"
    + curr_date.getMonth() + "-"
    + curr_date.getDate() + " "
    + curr_date.getHours + ":"
    + curr_date.getMinutes() + "\n";
  socket.end(curr_timestamp)
});
server.listen(8000)
```

Problem: if month, hour, etc. is a number like 7, our output doesn't follow the desired format with two digits

Setting up a simple server in Node.js

```
const net = require('net')
const server = net.createServer(function (socket) {
  // handle connections here
  var curr_date = new Date();
  var curr_timestamp = curr_date.getFullYear() + "-"
    + curr_date.getMonth() + "-"
    + curr_date.getDate() + " "
    + curr_date.getHours + ":"
    + curr_date.getMinutes() + "\n";
  socket.end(curr_timestamp)
});
server.listen(8000)
```

Problem: if month, hour, etc. is a number like 7, our output doesn't follow the desired format with two digits

Problem: getMonth() starts at 0, so returning 7 for the month of August isn't necessarily what we expected?

Setting up a simple server in Node.js

```
const net = require('net')
const server = net.createServer(function (socket) {
   // handle connections here
   var curr_date = new Date();
   var curr_timestamp = curr_date.getFullYear() + "-"
      + (curr_date.getMonth() + 1) + "-"
      + curr_date.getDate() + " "
      + curr_date.getHours + ":"
      + curr_date.getMinutes() + "\n";
   socket.end(curr_timestamp)
});
server.listen(8000)
```

Problem: if month, hour, etc. is a number like 7, our output doesn't follow the desired format with two digits

Solution: add one to the return value of getMonth

Setting up a simple server in Node.js

```
const net = require('net')
const server = net.createServer(function (socket) {
  // handle connections here
  var curr_date = new Date();
  var curr_timestamp = curr_date.getFullYear() + "-"
    + (curr_date.getMonth() + 1) + "-"
    + curr_date.getDate() + " "
    + curr_date.getHours + ":"
    + curr_date.getMinutes() + "\n";
  socket.end(curr_timestamp)
});
server.listen(8000)
```

Solution: check each output for number of digits

Solution: add one to the return value of getMonth

Setting up a simple server in Node.js

```
var curr_month = (curr_date.getMonth() + 1) < 10 ? // is month < 10?
                                 "0" + (curr_date.getMonth() + 1) : // if so, pad with a 0
                                 (curr_date.getMonth() + 1); // else, no need to pad
```

Setting up a simple server in Node.js

```
var curr_month = (curr_date.getMonth() + 1) < 10 ? // is month < 10?
    "0" + (curr_date.getMonth() + 1) : // if so, pad with a 0
    (curr_date.getMonth() + 1); // else, no need to pad
```

This is logically equivalent if you prefer if statement for readability:

```
var curr_month = curr_date.getMonth() + 1;
if (curr_month < 10) {
    curr_month = "0" + curr_date.getMonth() + 1;
}
```

Setting up a simple server in Node.js

```
const net = require('net')
const server = net.createServer(function (socket) {
  // handle connections here
  var curr_date = new Date();
  var curr_month = (curr_date.getMonth() + 1) < 10 ? "0" + (curr_date.getMonth() + 1) : (curr_date.getMonth() + 1);
  var curr_hours = (curr_date.getHours() < 10) ? "0" + curr_date.getHours() : curr_date.getHours();
  var curr_minutes = (curr_date.getMinutes() < 10) ? "0" + curr_date.getMinutes() : curr_date.getMinutes();
  var curr_timestamp = curr_date.getFullYear() + "-"
    + (curr_date.getMonth() + 1) + "-"
    + curr_date.getDate() + " "
    + curr_date.getHours + ":"
    + curr_date.getMinutes() + "\n";
  socket.end(curr_timestamp)
});
server.listen(8000)
```

Setting up a simple server in Node.js

```
const net = require('net')
const server = net.createServer(function (socket) {
   // handle connections here
   var curr_date = new Date();
   var curr_month = (curr_date.getMonth() + 1) < 10 ? "0" + (curr_date.getMonth() + 1) : (curr_date.getMonth() + 1);
   var curr_hours = (curr_date.getHours() < 10) ? "0" + curr_date.getHours() : curr_date.getHours();
   var curr_minutes = (curr_date.getMinutes() < 10) ? "0" + curr_date.getMinutes() : curr_date.getMinutes();
   var curr_timestamp = curr_date.getFullYear() + "-"
      + (curr_date.getMonth() + 1) + "-"
      + curr_date.getDate() + " "
      + curr_date.getHours + ":"
      + curr_date.getMinutes() + "\n";
   socket.end(curr_timestamp)
});
server.listen(8000)
```

Problem: we keep repeating the same code

Setting up a simple server in Node.js

```
const net = require('net')
const server = net.createServer(function (socket) {
    // handle connections here
    var curr_date = new Date();
    var curr_month = (curr_date.getMonth() + 1) < 10 ? "0" + (curr_date.getMonth() + 1) : (curr_date.getMonth() + 1);
    var curr_hours = (curr_date.getHours() < 10) ? "0" + curr_date.getHours() : curr_date.getHours();
    var curr_minutes = (curr_date.getMinutes() < 10) ? "0" + curr_date.getMinutes() : curr_date.getMinutes();
    var curr_timestamp = curr_date.getFullYear() + "-"
        + (curr_date.getMonth() + 1) + "-"
        + curr_date.getDate() + " "
        + curr_date.getHours + ":"
        + curr_date.getMinutes() + "\n";
    socket.end(curr_timestamp)
});
server.listen(8000)
```

Solution: refactor!!

Setting up a simple server in Node.js

```
const net = require('net')
function pad_zero (date_value) {
   return date_value < 10 ? "0" + date_value : date_value;
}
const server = net.createServer(function (socket) {
   // socket handling logic
   var curr_date = new Date();
   var date_output = curr_date.getFullYear() + "-"
      + pad_zero(curr_date.getMonth() + 1) + "-"
      + pad_zero(curr_date.getDate()) + " "
      + pad_zero(curr_date.getHours()) + ":"
      + pad_zero(curr_date.getMinutes()) + "\n";
   socket.end(date_output);
});
server.listen(8000)
```

Solution: refactor!!

Setting up a simple server in Node.js

```
const net = require('net')
function pad_zero (date_value) {
   return date_value < 10 ? "0" + date_value : date_value;
}
const server = net.createServer(function (socket) {
   // socket handling logic
   var curr_date = new Date();
   var date_output = curr_date.getFullYear() + "-"
      + pad_zero(curr_date.getMonth() + 1) + "-"
      + pad_zero(curr_date.getDate()) + " "
      + pad_zero(curr_date.getHours()) + ":"
      + pad_zero(curr_date.getMinutes()) + "\n";
   socket.end(date_output);
});
server.listen(8000)
```

Final problem: taking in a command-line argument?

Command-line arguments side note

- In general, command-line arguments can be passed into any program
- Convention calls these arguments "argv", usually an array
- To access command-line arguments in Node.js:

```
process.argv
```

Command-line arguments side note

- In general, command-line arguments can be passed into any program
- Convention calls these arguments "argv", usually an array
- To access command-line arguments in Node.js:

  process.argv

node time-server.js 8000

argv[0] = node

argv[1] = time-server.js

argv[2] = 8000

Command-line arguments side note

- In general, command-line arguments can be passed into any program
- Convention calls these arguments "argv", usually an array
- To access command-line arguments in Node.js:

  process.argv

node time-server.js 8000

argv[0] = node

argv[1] = time-server.js

argv[2] = 8000

- Can also replace the port number with argv call in your client!

Setting up a simple server in Node.js

```
const net = require('net')
function pad_zero (date_value) {
    return date_value < 10 ? "0" + date_value : date_value;
}
const server = net.createServer(function (socket) {
    // socket handling logic
    var curr_date = new Date();
    var date_output = curr_date.getFullYear() + "-"
        + pad_zero(curr_date.getMonth() + 1) + "-"
        + pad_zero(curr_date.getDate()) + " "
        + pad_zero(curr_date.getHours()) + ":"
        + pad_zero(curr_date.getMinutes()) + "\n";
    socket.end(date_output);
});
server.listen(process.argv[2])
```

Final solution!

Setting up a server in Node.js with web page

- Need to handle HTTP requests instead of TCP!
- First, a toy HTTP request-handling Node server to introduce you to making and handling HTTP requests
- Then, we're going to go higher-level with a React app; not for security reasons, just to use React (if we get to this, maybe a future workshop)
    - And, we will try to fix our security problems there

What is HTTP?

- "**HTTP** is a protocol which allows the fetching of resources, such as HTML documents. It is the foundation of any data exchange on the Web and it is a client-server protocol, which means requests are initiated by the recipient, usually the Web browser." [0]
  - As previously noted, built on top of TCP and other networking layers

- [0]: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

Setting up a server in Node.js with web page

- How do we handle HTTP requests instead of TCP?

Setting up a server in Node.js with web page

- How do we handle HTTP requests instead of TCP?

- Answer: we can use the http module instead of net!

Setting up a server in Node.js with web page

- How do we handle HTTP requests instead of TCP?

- Answer: we can use the http module instead of net!

```
const http = require('http');
const server = http.createServer(function (req, res) {
    // handle connections here!
    // notice we have a req input, unlike our TCP server
});
```

Setting up a server in Node.js with web page

- First, let's see how to handle GET requests and play with JSON. Then, we'll get to the web page.
- Goal: build a server that takes GET request input at two possible URLS
    - /api/parsetime expects a parameter called iso, an ISO-formatted timestamp, and will return JSON with fields hour, minute, second
    - /api/unixtime expects the same iso parameter, will return JSON with field unixtime (and value.. UNIX time)

(Idea credit: Feross Aboukhadijeh)

Setting up a server in Node.js with web page

A brief side note on HTTP requests....

- You can send two different types of requests called GET and POST.
- GET request data is visible in a URL
  - Ex: make a request at URL "/api/unixtime?iso=2013-08-10T12:10:15.474Z"
- POST request data is not visible in a URL
  - Ex: make a request at URL "/api/unixtime" and upload data in a form such as {unixtime: "2013-08-10T12:10:15.474Z"}

Generally, best practice for things such as logins is using POST requests.

Setting up a server in Node.js with web page

Here's what can happen if you don't use POST for forms:

- Say you put someone's user ID into a GET request.
- Your URL would look something like, "/home/userid=27"
- Anybody could change the userid and gain access to other accounts

Setting up a server in Node.js with web page

Here's what can happen if you don't use POST for forms:

- Say you put someone's user ID into a GET request.

- Your URL would look something like, "/home/userid=27"

- Anybody could change the userid and gain access to other accounts

Another difference between GET and POST: GET requests can be cached, POST is not.

Setting up a server in Node.js with web page

Here's what can happen if you don't use POST for forms:

- Say you put someone's user ID into a GET request.

- Your URL would look something like, "/home/userid=27"

- Anybody could change the userid and gain access to other accounts

Another difference between GET and POST: GET requests can be cached, POST is not.

- So, a userid could be recommended/stored by your browser

Setting up a server in Node.js with web page

- Goal: build a server that takes GET request input at two possible URLS
- But, how can we know which URL is being requested?

```
const http = require('http');
const server = http.createServer(function (req, res) {
    // handle connections here!
    // notice we have a req input, unlike our TCP server
});
```

Setting up a server in Node.js with web page

- Goal: build a server that takes GET request input at two possible URLS

(If you're curious, feel free to print request and URL to see their format)

```
const http = require('http');
const url = require('url');
const server = http.createServer(function (req, res) {
   // handle connections here!
  var url_parts = url.parse(req.url, true); // parsing a URL
  var query = url_parts.query; // this grabs any query a user made
});
```

Setting up a server in Node.js with web page

How can we check which URL was requested?

```
const http = require('http');
const url = require('url');

const server = http.createServer(function (req, res) {
    // handle connections here!
   var url_parts = url.parse(req.url, true); // parsing a URL
   var query = url_parts.query; // this grabs any query a user made

});
```

Setting up a server in Node.js with web page

```
const http = require('http');
const url = require('url');
const server = http.createServer(function (req, res) {
   // handle connections here!
  var url_parts = url.parse(req.url, true); // parsing a URL
  var query = url_parts.query; // this grabs any query a user made
  if (url_parts.pathname === "/api/parsetime") {
      // handle parsetime URL
  } else if (url_parts.pathname === "/api/unixtime") {
      // handle unixtime URL
  } else {
      // handle any other URL
}});
```

Setting up a server in Node.js with web page

Another side note about HTTP...

- There are different codes that designate the status of a request.
- Some mean that something worked, some mean something went wrong, and different numbers are used to state what exactly went wrong.

- All we need to know here is that we're going to set our status as 200 because everything is fine.

Setting up a server in Node.js with web page

```
  var url_parts = url.parse(req.url, true); // parsing a URL
  var query = url_parts.query; // this grabs any query a user made
  if (url_parts.pathname === "/api/parsetime") {
    // set status, specify that we're sending over some JSON
    res.writeHead(200, { 'Content-Type': 'application/json'});
  } else if (url_parts.pathname === "/api/unixtime") {
    res.writeHead(200, { 'Content-Type': 'application/json'});
  } else {
    // handle any other URL
}
```

Setting up a server in Node.js with web page

```
  var url_parts = url.parse(req.url, true); // parsing a URL
  var query = url_parts.query; // this grabs any query a user made
  if (url_parts.pathname === "/api/parsetime") {
     // set status, specify that we're sending over some JSON
     res.writeHead(200, { 'Content-Type': 'application/json'});
  } else if (url_parts.pathname === "/api/unixtime") {
     res.writeHead(200, { 'Content-Type': 'application/json'});
  } else {
     // handle any other URL
}
```

Setting up a server in Node.js with web page

```
  var url_parts = url.parse(req.url, true); // parsing a URL
  var query = url_parts.query; // this grabs any query a user made
  if (url_parts.pathname === "/api/parsetime") {
     // set status, specify that we're sending over some JSON
     res.writeHead(200, { 'Content-Type': 'application/json'});
     // create our JSON, play a little more with the Date functionality of JS
     var given_iso_time = new Date(query.iso);
     var parsed_time_json = { hour: given_iso_time.getHours(), minute: given_iso_time.getMinutes(), second: given_iso_time.getSeconds() };
     res.end(JSON.stringify(parsed_time_json)); // we need to convert this to a string when sending, else will get an error!
  } else if (url_parts.pathname === "/api/unixtime") {
     res.writeHead(200, { 'Content-Type': 'application/json'});
  } else {
     // handle any other URL
}
```

Setting up a server in Node.js with web page

```
  var url_parts = url.parse(req.url, true); // parsing a URL
  var query = url_parts.query; // this grabs any query a user made
  if (url_parts.pathname === "/api/parsetime") {
     // this code removed from this slide for readability
  } else if (url_parts.pathname === "/api/unixtime") {
     // more JSON parsing here
     var given_iso_time = new Date(query.iso);
     var unixtime_json = { unixtime: given_iso_time.getTime() };
     res.writeHead(200, { 'Content-Type': 'application/json'});
     res.end(JSON.stringify(unixtime_json));
  } else {
     // handle any other URL
}
```

Setting up a server in Node.js with web page

How can we test this?

- node http-server.js 8000 (or whatever other name you gave to this server) to run it in one terminal

Make some GET requests (install curl if you don't have it):

- curl localhost:8000/api/parsetime?iso=2013-08-10T12:10:15.474Z
- curl localhost:8000/api/unixtime?iso=2013-08-10T12:10:15.474Z
  - Might not work on Windows
  - Can also go to localhost:8000/api/parsetime?iso=2013-08-10T12:10:15.474Z in your web browser

Setting up a server in Node.js with web page

Ok, so, we finally made a TCP server, and then an HTTP server returning JSON.

Setting up a server in Node.js with web page

Ok, so, we finally made a TCP server, and then an HTTP server returning JSON.

Time for that web page that was promised 30 slides ago!

Setting up a server in Node.js with web page

How can we go from serving JSON to HTML?

Setting up a server in Node.js with web page

How can we go from serving JSON to HTML?

If you made the proper GET request on the previous server, you would have already been able to view the JSON in your web browser. So, now all we have to do is replace the JSON with HTML!

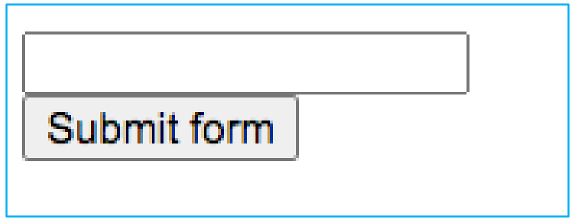
Setting up a server in Node.js with web page

```
const http = require('http');
var url = require('url');
const server = http.createServer(function (req, res) {
    res.end(`
    <!doctype html>
        <html>
        <body>
            <form action="/" method="post">
                <input type="text" name="test_text" /><br />
                <button>Submit form</button>
            </form>
        </body>
     </html> `); });
server.listen(process.argv[2])
```

Setting up a server in Node.js with web page

```
const http = require('http');
var url = require('url');
const server = http.createServer(function (req, res) {
    res.end(`
    <!doctype html>
        <html>
        <body>
            <form action="/" method="post">
                <input type="text" name="test_text" /><br />
                <button>Submit form</button>
            </form>
        </body>
     </html> `); });
server.listen(process.argv[2])
```

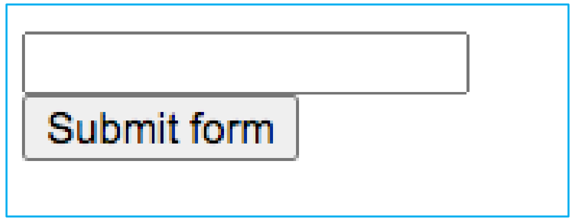
What you'll see if you go to localhost:8000 (or any other port you chose) in your browser:



Setting up a server in Node.js with web page

```
const http = require('http');
var url = require('url');
const server = http.createServer(function (req, res) {
  res.end(`
  <!doctype html>
    <html>
    <body>
      <form action="/" method="post">
        <input type="text" name="test_text" /><br />
        <button>Submit form</button>
      </form>
    </body>
   </html> `); });
server.listen(process.argv[2])
```

What you'll see if you go to localhost:8000 (or any other port you chose) in your browser:



Next – let's actually handle this form!

Setting up a server in Node.js with web page

```
const http = require('http');
var url = require('url');
const server = http.createServer(function (req, res) {
    if (req.method === 'POST') {
    // handle POST req here
} else {
res.end(`
<!doctype html> <html> <body> <form action="/" method="post">
<input type="text" name="test_text" /><br />
<button>Submit form</button>
</form> </body> </html>
`); } });
server.listen(process.argv[2])
```

Setting up a server in Node.js with web page

```
if (req.method === 'POST') {
   // handle POST req here
   let body = '';
   req.on('data', chunk => {
       body += chunk.toString();
   });
   req.on('end', () => {
   var url_parts = url.parse("/?" + body, true);
   res.end('<p>I received a post request with the text input: ' + url_parts.query.test_text + '</p>');
   });
} else {
   ….
}
```

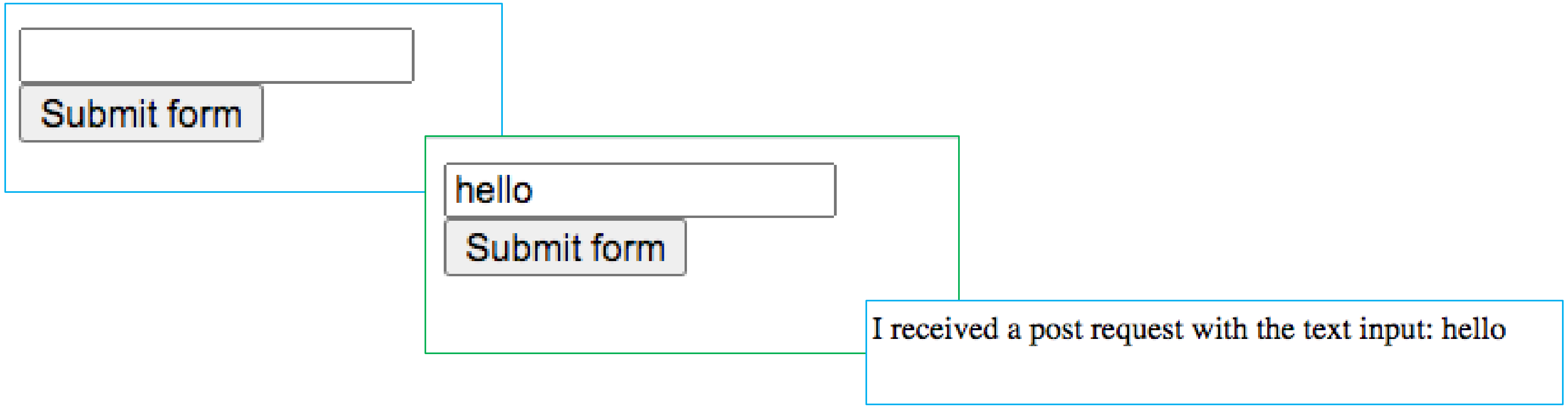
Setting up a server in Node.js with web page

```
if (req.method === 'POST') {
    // handle POST req here
    let body = '';
    req.on('data', chunk => {
        body += chunk.toString();
    });
    req.on('end', () => {
    var url_parts = url.parse("/?" + body, true);
    res.end('<p>I received a post request with the text input: ' + url_parts.query.test_text + '</p>');
    });
} else {
    ….
}
```

Warning: probably not best practice for parsing a POST request but it works for now.

Setting up a server in Node.js with web page

End result:



```
I received a post request with the text input: hello
```

Can also make POST requests using curl:

```
curl --request POST --data 'test_text=hello' localhost:8000
```

Final thoughts

Was this interesting at all? Hopefully even a little?

Final CS241 MP is a basic TCP client and server written in C

- If you wanted to test your code with this Javascript client, as long as you follow the given protocol, you're probably able to

If you liked this workshop or CS241, CS/ECE438 (Intro to Computer Networks) might be a good class to take

- First 438 MP is also building a client/server

Final thoughts

Was this interesting at all? Hopefully even a little?

- Much more complexity needed to handle cases such as many clients at once, slow connections, errors, etc.

Final CS241 MP is a basic TCP client and server written in C

- If you wanted to test your code with this Javascript client, as long as you follow the given protocol, you're probably able to

If you liked this workshop or CS241, CS/ECE438 (Intro to Computer Networks) might be a good class to take

- First 438 MP is also building a client/server