

EECS 368

Programming Language Paradigms

David O. Johnson
Fall 2022

Reminders

- Assignment 3 due: 11:59 PM, Monday, October 3
- Assignment 4 due: 11:59 PM, Monday, October 17

Any Questions?

In-Class Problem Solution

- 14-(9-26) In-Class Problem Solution.pptx

Any Questions?

Chapter 18 - HTTP and Forms

- The protocol
- Browsers and HTTP
- Fetch
 - Promises
- Security and HTTPS
- Form fields
- Focus
- Disabled fields
- The form as a whole
- Text fields
- Checkboxes and radio buttons
- Select fields
- File fields
- Storing data client-side

HTTP & TCP in the OSI Protocol Model

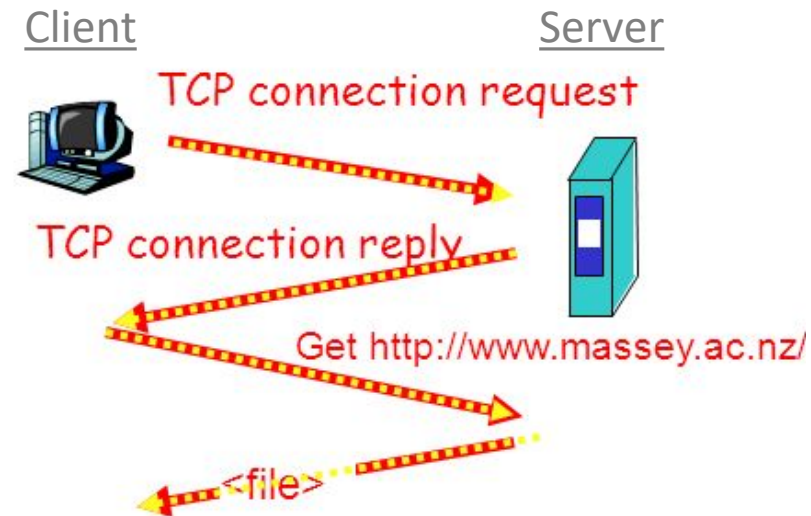
Recall:

- Transmission Control Protocol (TCP) is a protocol that ensures you can treat the network as a streamlike device into which you can put bits and have them arrive at the correct destination in the correct order.
- Hypertext Transfer Protocol (HTTP) is the mechanism through which data is requested and provided on the World Wide Web.

OSI model		
	Layer	Function
Host layers	7 Application	High-level APIs, including resource sharing, remote file access HTTP
	6 Presentation	Translation of data between a networking service and an application; including character encoding, data compression and encryption/decryption
	5 Session	Managing communication sessions, i.e., continuous exchange of information in the form of multiple back-and-forth transmissions between two nodes
	4 Transport	Reliable transmission of data segments between points on a network, including segmentation, acknowledgement and multiplexing TCP
Media layers	3 Network	Structuring and managing a multi-node network, including addressing, routing and traffic control
	2 Data link	Reliable transmission of data frames between two nodes connected by a physical layer
	1 Physical	Transmission and reception of raw bit streams over a physical medium

The Protocol

- For the next few lectures, we will look at the HTTP protocol itself in more detail and explain the way browser JavaScript has access to it.
- If you type: eloquentjavascript.net/18_http.html into your browser's address bar, ...
- the browser first looks up the address of the server associated with eloquentjavascript.net ...
- and tries to open a TCP connection to it on port 80.
- Port 80 is the default port for HTTP traffic.



- If the server exists and accepts the TCP connection request, the browser might send something like this:

Request from client

```
GET /18_http.html HTTP/1.1
Host: eloquentjavascript.net
User-Agent: Your browser's name
```


The Protocol

- Then the server responds, through that same connection:
- The browser takes the part of the response after the blank line, ...
- **its body, ...**
- (not to be confused with the HTML <body> tag) ...
- and displays it as an HTML document.

Request from client

```
GET /18_http.html HTTP/1.1
Host: eloquentjavascript.net
User-Agent: Your browser's name
```

Response from server

```
HTTP/1.1 200 OK
Content-Length: 65585
Content-Type: text/html
Last-Modified: Mon, 08 Jan 2018 10:29:45 GMT

<!doctype html>
... the rest of the document
```

Client's Request

- The information sent by the client is called the request.
- It starts with this line:
- `GET /18_http.html HTTP/1.1`
- The first word is the method of the request.
- `GET` means that we want to get the specified resource.
- Other common methods are:
 - DELETE to delete a resource
 - PUT to create or replace it
 - POST to send information to it
- Note that the server is not obliged to carry out every request it gets.
- If you walk up to a random website and tell it to DELETE its main page, it'll probably refuse.

Request from client

```
GET /18_http.html HTTP/1.1
Host: eloquentjavascript.net
User-Agent: Your browser's name
```

Response from server

```
HTTP/1.1 200 OK
Content-Length: 65585
Content-Type: text/html
Last-Modified: Mon, 08 Jan 2018 10:29:45 GMT

<!doctype html>
... the rest of the document
```

Client's Request

- The part after the method name is the path of the resource the request applies to.
- In the simplest case, a resource is simply a file on the server.
- But the protocol doesn't require it to be.
- A resource may be anything that can be transferred as if it is a file.
- Many servers generate the responses they produce on the fly.
- For example, if you open `https://github.com/marijnh`, ...
- the server looks in its database for a user named "marijnh", ...
- and if it finds one, it will generate a profile page for that user.

Request from client

```
GET /18_http.html HTTP/1.1
Host: eloquentjavascript.net
User-Agent: Your browser's name
```

Response from server

```
HTTP/1.1 200 OK
Content-Length: 65585
Content-Type: text/html
Last-Modified: Mon, 08 Jan 2018 10:29:45 GMT

<!doctype html>
... the rest of the document
```

Client's Request

- After the resource path, the first line of the request mentions **HTTP/1.1** to indicate the version of the HTTP protocol it is using.
- In practice, many sites use HTTP version 2, which supports the same concepts as version 1.1 ...
- but is a lot more complicated so that it can be faster.
- Browsers will automatically switch to the appropriate protocol version when talking to a given server, ...
- and the outcome of a request is the same regardless of which version is used.
- Because version 1.1 is more straightforward and easier to play around with, we'll focus on that.

Request from client

```
GET /18_http.html HTTP/1.1
Host: eloquentjavascript.net
User-Agent: Your browser's name
```

Response from server

```
HTTP/1.1 200 OK
Content-Length: 65585
Content-Type: text/html
Last-Modified: Mon, 08 Jan 2018 10:29:45 GMT

<!doctype html>
... the rest of the document
```

Server's Response

- The server's response will start with a version as well, followed by the status of the response, ...
 - first as a three-digit status code ...
 - and then as a human-readable string.
-
- Status codes starting with a 2 indicate that the request succeeded.
 - Codes starting with 4 mean there was something wrong with the request.
 - 404 is probably the most famous HTTP status code.
 - It means that the resource could not be found.
 - Codes that start with 5 mean an error happened on the server and the request is not to blame.

Request from client

```
GET /18_http.html HTTP/1.1
Host: eloquentjavascript.net
User-Agent: Your browser's name
```

Response from server

```
HTTP/1.1 200 OK
Content-Length: 65585
Content-Type: text/html
Last-Modified: Mon, 08 Jan 2018 10:29:45 GMT

<!doctype html>
... the rest of the document
```

Any Questions?

Request/Response Headers

- The first line of a request or response may be followed by any number of headers.
- These are lines in the form:
 name: value
- where value specifies extra information about the request or response.

Request from client

```
GET /18_http.html HTTP/1.1  
Host: eloquentjavascript.net  
User-Agent: Your browser's name
```

Response from server

```
HTTP/1.1 200 OK  
Content-Length: 65585  
Content-Type: text/html  
Last-Modified: Mon, 08 Jan 2018 10:29:45 GMT  
  
<!doctype html>  
... the rest of the document
```

Request/Response Headers

The headers from the example response:

Content-Length: 65585

- This tells us the size of the response document - 65,585 bytes

Content-Type: text/html

- This tells us the type of the response document - it is an HTML document

Last-Modified: Thu, 04 Jan 2018 14:05:30 GMT

- Tells us when that document was last modified

Request from client

```
GET /18_http.html HTTP/1.1
Host: eloquentjavascript.net
User-Agent: Your browser's name
```

Response from server

```
HTTP/1.1 200 OK
Content-Length: 65585
Content-Type: text/html
Last-Modified: Mon, 08 Jan 2018 10:29:45 GMT

<!doctype html>
... the rest of the document
```


Request/Response Headers

- For most headers, the client and server are free to decide whether to include them in a request or response.
- But a few are required.
- For example, the [Host](#) header, which specifies the hostname, should be included in a request.
- Because a server might be serving multiple hostnames on a single IP address ...
- without that header, ...
- the server won't know which hostname the client is trying to talk to.

Request from client

```
GET /18_http.html HTTP/1.1
Host: eloquentjavascript.net
User-Agent: Your browser's name
```

Response from server

```
HTTP/1.1 200 OK
Content-Length: 65585
Content-Type: text/html
Last-Modified: Mon, 08 Jan 2018 10:29:45 GMT

<!doctype html>
... the rest of the document
```

Request/Response Body

- After the headers, both requests and responses may include a ...
- blank line ...
- followed by a body, which contains the data being sent.
- GET and DELETE requests don't send along any data.
- PUT and POST requests do.
- Similarly, some response types, such as error responses, do not require a body.

Request from client

```
GET /18_http.html HTTP/1.1
Host: eloquentjavascript.net
User-Agent: Your browser's name
```

Response from server

```
HTTP/1.1 200 OK
Content-Length: 65585
Content-Type: text/html
Last-Modified: Mon, 08 Jan 2018 10:29:45 GMT

<!doctype html>
... the rest of the document
```

Any Questions?

Browsers and HTTP

- As we saw in the example, a browser will make a request when we enter a URL in its address bar.
- When the resulting HTML page references other files, such as images and JavaScript files, those are also retrieved.
- A moderately complicated website can easily include anywhere from 10 to 200 resources.
- To be able to fetch those quickly, ...
- browsers will make several GET requests simultaneously, ...
- rather than waiting for the responses one at a time.

Request from client

```
GET /18_http.html HTTP/1.1  
Host: eloquentjavascript.net  
User-Agent: Your browser's name
```

Response from server

```
HTTP/1.1 200 OK  
Content-Length: 65585  
Content-Type: text/html  
Last-Modified: Mon, 08 Jan 2018 10:29:45 GMT  
  
<!doctype html>  
... the rest of the document
```

HTML Forms (GET)

- HTML pages may include *forms*, which allow the user to fill out information and send it to the server.
- This is an example of a form:

```
<form method="GET" action="example/message.html">  
  <p>Name: <input type="text" name="name"></p>  
  <p>Message:<br><textarea name="message"></textarea></p>  
  <p><button type="submit">Send</button></p>  
</form>
```

- This HTML code describes a form with two fields:
 - a small one asking for a name and
 - a larger one to write a message in
- When you click the Send button, the form is *submitted*, ...
- meaning that the content of its field is packed into an HTTP request and ...
- the browser navigates to the result of that request.



Name:

Message:

Send

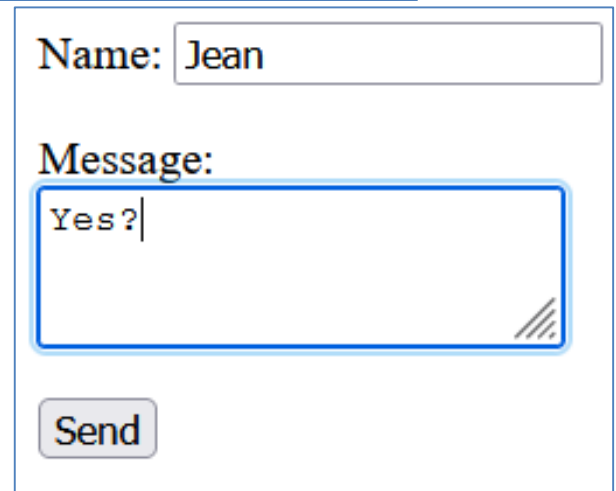
HTML Forms (GET)

```
<form method="GET" action="example/message.html">
  <p>Name: <input type="text" name="name"></p>
  <p>Message:<br><textarea name="message"></textarea></p>
  <p><button type="submit">Send</button></p>
</form>
```

- When the `<form>` element's method attribute is `GET` (or it is omitted), ...
- the information in the form is added to the end of the action URL as a query string.
- The browser might make a request to this URL:

```
GET /example/message.html?name=Jean&message=Yes%3F HTTP/1.1
```

- The question mark (?) indicates the end of the path part of the URL and the start of the query.
- It is followed by pairs of **names and values**, corresponding to the **name** attribute on the form field elements and the **content of those elements**, respectively.
- An ampersand character (&) is used to separate the pairs.



Name:


Message:

HTML Forms (GET)

```
<form method="GET" action="example/message.html">
  <p>Name: <input type="text" name="name"></p>
  <p>Message:<br><textarea name="message"></textarea></p>
  <p><button type="submit">Send</button></p>
</form>
```

GET /example/message.html?name=Jean&message=Yes%3F HTTP/1.1

- The actual message encoded in the URL is “Yes?”.
- But the question mark is replaced by a strange code.
- Some characters in query strings must be escaped.
- The question mark, represented as %3F, is one of those.
- There seems to be an unwritten rule that every format needs its own way of escaping characters.
- This one, called URL encoding, uses a percent sign followed by ...
- two hexadecimal (base 16) digits that encode the character code.
- In this case, 3F, which is 63 in decimal notation, ...
- is the code of a question mark character.



The image shows a web form with a light blue border. At the top, it says 'Name:' followed by a text input field containing the word 'Jean'. Below that, it says 'Message:' followed by a text area containing the text 'Yes?'. At the bottom of the form is a button labeled 'Send'.

HTML Forms (GET)

```
<form method="GET" action="example/message.html">
  <p>Name: <input type="text" name="name"></p>
  <p>Message:<br><textarea name="message"></textarea></p>
  <p><button type="submit">Send</button></p>
</form>
```

```
GET /example/message.html?name=Jean&message=Yes%3F HTTP/1.1
```

- JavaScript provides the `encodeURIComponent` and `decodeURIComponent` functions to encode and decode this format.

```
console.log(encodeURIComponent("Yes?"));
// → Yes%3F
console.log(decodeURIComponent("Yes%3F"));
// → Yes?
```

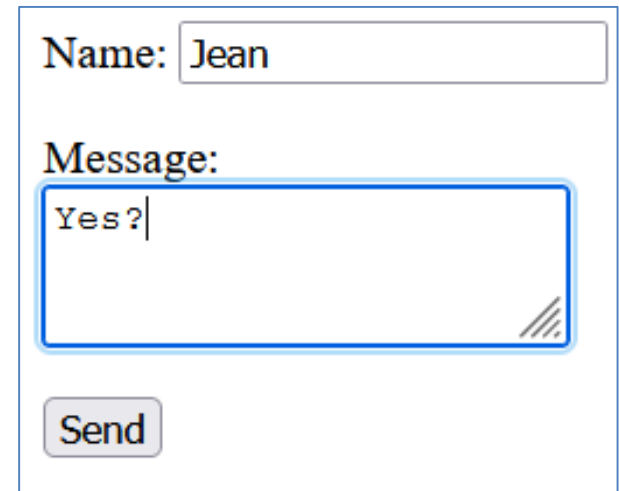

HTML Forms (POST)

```
<form method="POST" action="example/message.html">
  <p>Name: <input type="text" name="name"></p>
  <p>Message:<br><textarea name="message"></textarea></p>
  <p><button type="submit">Send</button></p>
</form>
```

- If we change the method attribute of the HTML form in the example we saw earlier to POST, ...
- the HTTP request made to submit the form will use the POST method and ...
- put the query string in the body of the request, rather than adding it to the URL.

```
POST /example/message.html HTTP/1.1
Content-length: 24
Content-type: application/x-www-form-urlencoded
```

```
name=Jean&message=Yes%3F
```



Name:

Message:

HTML Forms – GET? or POST?

```
<form method="GET" action="example/message.html">
  <p>Name: <input type="text" name="name"></p>
  <p>Message:<br><textarea name="message"></textarea></p>
  <p><button type="submit">Send</button></p>
</form>
```

GET /example/message.html?name=Jean&message=Yes%3F HTTP/1.1

```
<form method="POST" action="example/message.html">
  <p>Name: <input type="text" name="name"></p>
  <p>Message:<br><textarea name="message"></textarea></p>
  <p><button type="submit">Send</button></p>
</form>
```

POST /example/message.html HTTP/1.1
Content-length: 24
Content-type: application/x-www-form-urlencoded

name=Jean&message=Yes%3F

Name:

Message:

HTML Forms – GET? or POST?

- GET requests should be used for requests that do not have side effects but simply ask for information.
- Requests that change something on the server, ...
 - for example creating a new account or posting a message, ...
 - should be expressed with other methods, such as POST.
- Client-side software such as a browser knows that it shouldn't blindly make POST requests but ...
 - will often implicitly make GET requests.
 - For example to prefetch a resource it believes the user will soon need.
- We'll come back to forms and how to interact with them from JavaScript later.

Any Questions?

In-Class Problem

1. Create a request from a client to get the file “index.html” from the host “ku.edu” using HTTP version 1.1.
2. Create a response from the server for ku.edu which contains the index.html file. Assume the length of the file is 50,000 bytes and it was last modified February 23, 2022 at 9:00:00 GMT.