COMP 3015

Server Side Web Application Development

Today

- More OOP
- HTTP: status codes, verbs, headers
- Accepting text input from HTML forms
- Uploading files
- Redirections
- Input Validation and Regex

Hypertext Transfer Protocol (HTTP)

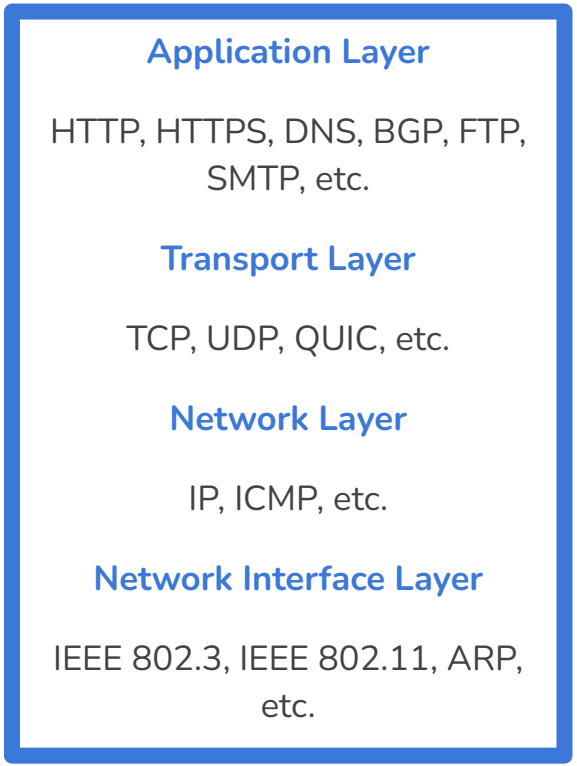
HTTP: Hypertext Transfer Protocol

“The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems.” source: https://www.ietf.org/rfc/rfc2616.txt

What is a protocol?

- **Protocol** (data communications): a set of rules for communication between systems.

HTTP: an application-level protocol?



Protocols are grouped into conceptual layers in a communication system.

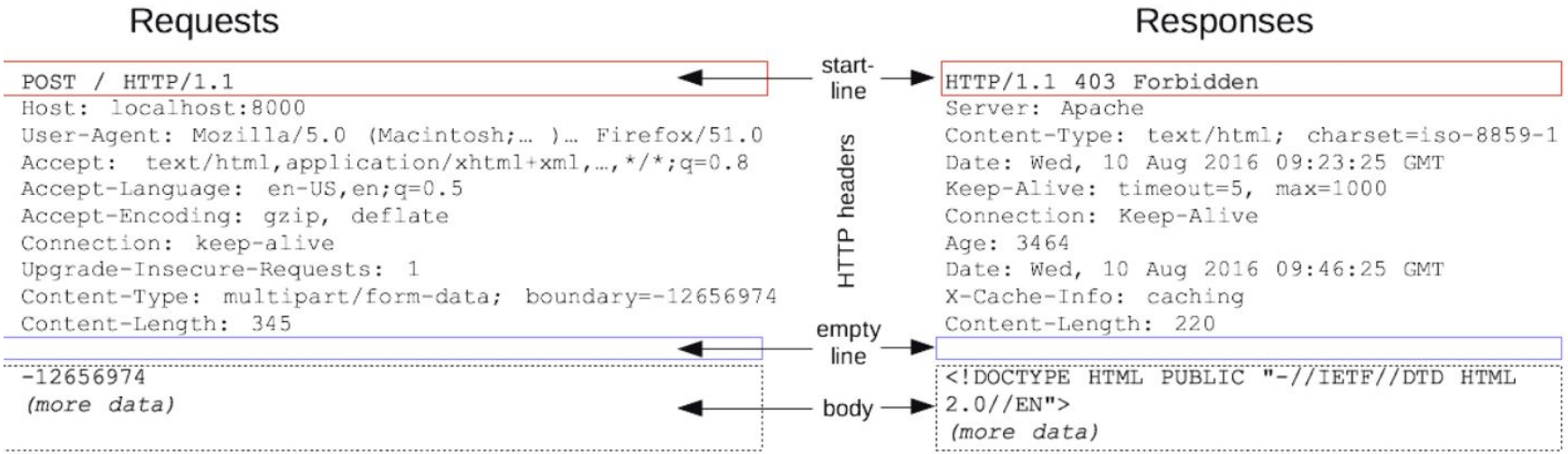
HTTP/1.0, HTTP/1.1 and HTTP/2 use TCP as the underlying transport protocol, HTTP/3 will use QUIC as the underlying transport protocol.

HTTP: Hypertext Transfer Protocol

- Foundation of the World Wide Web
- HTTP has request types including: GET, POST, PUT, PATCH, DELETE

- Each request type is meant for something different and functions slightly differently.
    - In this course we will mostly be working with GET and POST requests (when we get to RESTful APIs we'll use more types)

HTTP: Message Format



The start line (red), contains the request method type, the target (in this case, "/" or the "index" page), and the HTTP version (in this case HTTP/1.1). Source: https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages

Try running on the terminal: **$ curl -v example.com** ← do you see the HTTP headers, empty line, body?
Note: you might not have cURL installed by default

HTTP: Status Codes

https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

- Informational responses 100-199
  - e.g. Switching Protocols – HTTP is the main protocol but we can switch to others such as Web Sockets
- Success responses 200-299
  - 200 OK
  - 201 Created
- Redirection responses 300-399
  - 301 Moved Permanently
  - 302 Moved Temporarily
- Client error responses 400-499
  - 404 Not Found
- Server error responses 500-599
  - 500 Internal Server Error
    - Thrown in cases such as your PHP code having invalid syntax

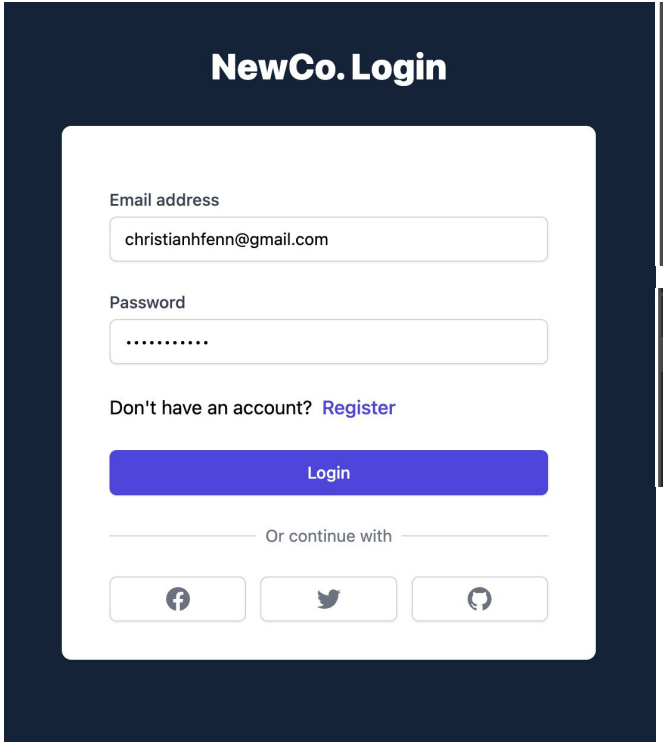
HTTP: GET Requests

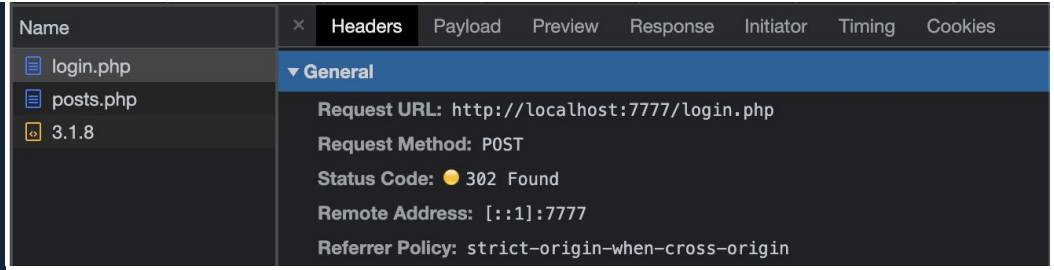
- HTTP GET request data is put in the URL

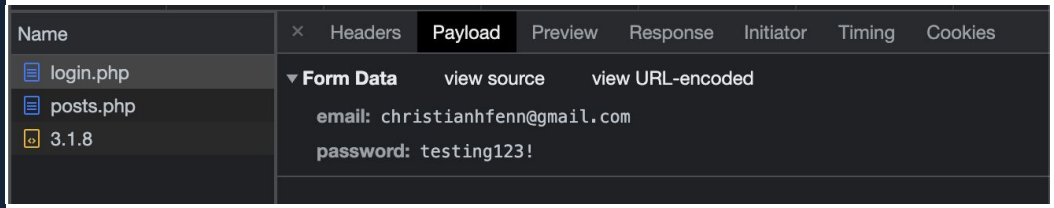
- Examples:
  - https://www.google.com/search?q=cats
  - https://www.google.com/maps/@49.251537,-123.003726,3a,75y,154.4h,96.48t/data=!3m6!1e1!3m4!1sQuh2zERbL1nLxd7ReXX3iA!2e0!7i16384!8i8192?entry=ttu

- What kind of requests would using a HTTP GET be bad for?

HTTP: POST Requests







Additional Notes

For now we only need to know about HTTP GET and POST requests. Later in the course we'll cover PUT, PATCH, DELETE and OPTIONS requests. When building RESTful APIs these become essential.

Difference between GET and POST

- HTTP GET requests append data to the URL
- HTTP POST requests put data in the body of the HTTP request

Common industry interview question (and quiz/exam question).

Accepting User Input

Superglobals in PHP we'll be using today

- $_POST
  - Associative array containing data sent to the script using a HTTP POST request
  - https://www.php.net/manual/en/reserved.variables.post
- $_GET
  - Associative array containing data sent to the script using a HTTP GET request
  - https://www.php.net/manual/en/reserved.variables.get.php

Superglobals in PHP we'll be using today

- $_SERVER
  - Server and execution environment information
    - REQUEST_METHOD tells us what type of HTTP request was made to get to the current page
  - https://www.php.net/manual/en/reserved.variables.server.php
- $_FILES
  - Used when files are uploaded using HTTP (we'll see that later today!)
  - https://www.php.net/manual/en/reserved.variables.files.php

Additional PHP superglobals

https://www.php.net/manual/en/language.variables.superglobals.php

We'll see **$_COOKIE**, **$_ENV** later in the course

Accessing POST data using $_POST global

```
<form action="register.php" method="post">
    <div>
        <label>
            Name:
            <input type="text" name="name">
        </label>
    </div>
    <div>
        <label>
            Email:
            <input type="text" name="email">
        </label>
    </div>
```

Wherever the form is processed (in this case, the form **action** points to register.php):

```
$name = $_POST['name'];
$email = $_POST['email'];
```

There is a $_GET global for HTTP GET data (see D2L FormsExample for usage)

Accepting user input

```
<form action="register.php" method="post">
    <label>
        Email:
        <input type="email" name="email">
    </label>
    <input type="submit" value="Submit Email">
</form>
```

“method” specifies how to send form data:

- post: perform an HTTP POST request (data is sent in the body of the HTTP request)
- get: perform an HTTP GET request (form data is appended to the URL)

“action”: The URL that processes the form submission

Forms Example on D2L

Uploading Files: Encoding

HTML forms have 3 encoding types:

- **application/x-www-form-urlencoded**
  - This is the default. It's fine for any text inputs
- **multipart/form-data**
  - Must be used for file uploads (if a <form> contains an input of type='file')
- **text/plain**
  - Human readable payload format. Can be useful for debugging purposes

For more information, see:
https://developer.mozilla.org/en-US/docs/Web/HTML/Element/form#attr-enctype

Uploading Files: Idea

- When a file is uploaded, it will be placed in a temporary location on the server

- We can use the **$_FILES** superglobal to access the temporary name (path) of the file on the server, as well as additional information such as the file size, original file name, file type, etc.

- This is easier to work with than receiving a steam of binary data (representing an image, PDF, etc.) would be
  - That's what you would do in TCP socket programming

Uploading Files: Example

The form specifies that an HTTP POST request will be made to “file_upload.php”, with an encoding type of “multipart/form-data”.

```
<form action="file_upload.php" method="post" enctype="multipart/form-data">
    <input type="file" name="profile_picture">
    <input type="submit" value="Upload Picture">
</form>
```

In **file_upload.php** we handle the request:

```
<?php
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $file = $_FILES['profile_picture'];
    $temporaryPath = $file['tmp_name'];
    $originalFileName = $file['name'];
    move_uploaded_file($temporaryPath, __DIR__ . "/images/$originalFileName");
}
?>
```

Redirection

The Location response header can be used to redirect a user.

Using the header function does not stop script execution, so in most cases you will want to call exit after using it.

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Location

```
header( header: "Location: welcome.php?from=register&email=$email");
exit();
```

Validation

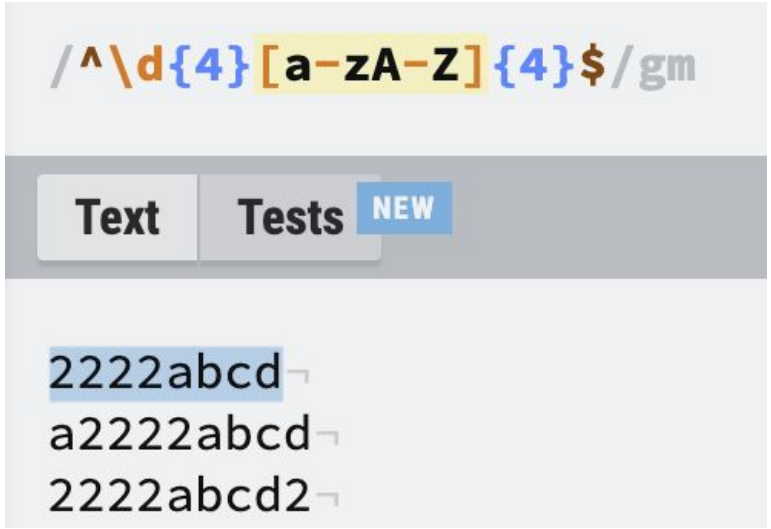
Validation: Regular Expressions (Regex)

Lots of inputs follow common formats. We can often use regular expressions to check if a given input string matches an expected pattern.

Let’s imagine we want to match a pattern of any 4 numeric digits, followed by any 4 letters.

Go to: https://regexr.com/ and we can try it out.

Regex Solution



Explained from left to right

- ^ means match the start of the string
- **\d** is any digit
- **{4}** is a quantifier meaning “match 4 of the preceding token”
- **[a-zA-Z]** means any letter (uppercase or lowercase)
- **{4}** quantifier, saying 4 of the previous characters in the range [a-zA-Z] (any lowercase or uppercase letter)
- **$** means match the end of the string

For understanding ^ and $, look at the inputs that aren’t highlighted in the image on the left.

Regex in PHP

preg_match: https://www.php.net/manual/en/function.preg-match.php

```php
<?php
$pattern = "/^\d{4}[a-zA-Z]{4}$/";
$subject = "1234ABCD";
if (preg_match($pattern, $subject)) {
    echo "$subject matches \"$pattern\"";
}
```

Regex (cont.)

- If we validate data such as an email or phone number, this only tells us that the input string conforms to the pattern we're looking for
    - It doesn't tell us that the email or phone number belongs to the user that entered it
    - We still need additional validation for this kind of data
        - Verify email address/phone num by clicking a link

Validation with filters

- Data including email addresses can be very difficult to validate with regex.
- PHP provides filters that we can use instead of regex in some cases
- See: https://www.php.net/manual/en/book.filter.php for filters

Example:

```php
if (!filter_var($email, filter: FILTER_VALIDATE_EMAIL)) {
    echo "The email: \"$email\" is not valid<br/>";
} else {
    echo "The email: \"$email\" is valid<br/>";
}
```

See:
https://stackoverflow.com/questions/201323/how-can-i-validate-an-email-address-using-a-regular-expression for the complexities of using regular expressions to validate email addresses

Validation continued: XSS

Cross Site Scripting (XSS)

Whenever you want to output something in HTML that is from a user (keeping in mind that no user input should be trusted), we need to ensure the data is “sanitized”.

htmlspecialchars can be used for converting HTML characters such as <, >, &, etc. to their HTML encodings

e.g. “&” → “&”

“<” → “<”

XSS Prevention: Key Idea

- Use the PHP function htmlspecialchars, which will convert special characters to HTML entities.

- These encoded values won't be processed by the browser as standard HTML. (e.g. in the case of someone using **<script>alert('test')</script>** as input to a form or query param).

Assignment 1 is on D2L

Work for the week

- Assignment 1
  - Due in week 5 (2 weeks from now)
- Review from today
  - Ensure you know the difference between HTTP GET and POST requests
  - How to accept user input using HTML forms
    - Text inputs and file uploads
    - Review code examples posted on D2L
    - Needed for Assignment 1