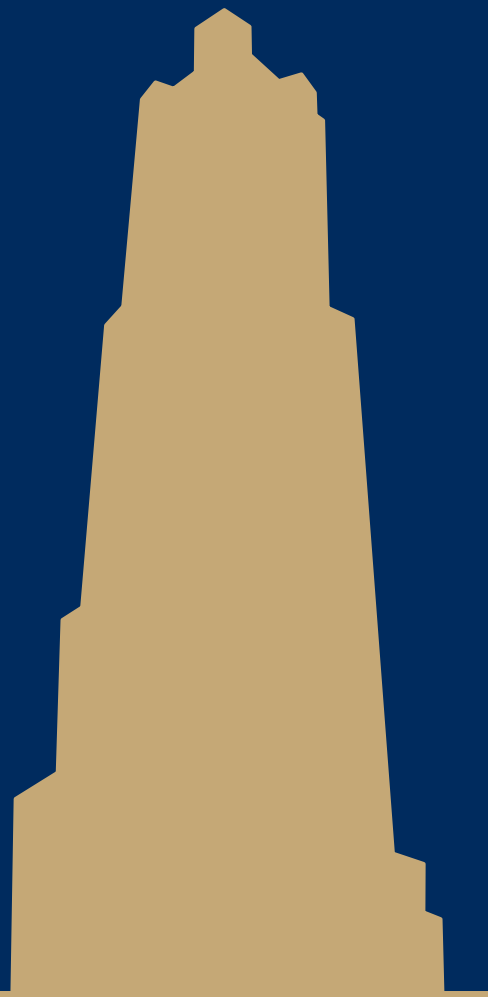
CS/COE 1520

pitt.edu/~ach54/cs1520

AJAX



Where we stand so far

- With JavaScript, we said that we wanted to build *dynamic* web *applications*
  - E.g., your battleship game
- With Flask, we started to utilize client/server interactions
  - First true website of the class
    - Even if it is all running on the same machine…
- However, to get new data from our model, we needed to reload the entire page
  - This isn't very dynamic…
  - How do we build the dynamic applications we started off talking about?

AJAX

- We use JavaScript to create dynamic client-side applications
  - Edit the DOM
    - Causing the page to be re-rendered
  - But how can we use it to fetch new data from the server?
    - Through the use of the `XMLHttpRequest` object
      - The backbone of AJAX

XMLHttpRequest.open()

- `open(method, url, async)`
  - *`method`* is an HTTP method
  - *`url`* is the location of the server
  - *`async`* is a boolean to determine if the transfer is to be done asynchronously or not
    - Defaults to true

XMLHttpRequest.send()

- `send(`*`data`*`)`
  - Issues the specified HTTP request to the server
  - *`data`* is the (optional) information to be sent to the server
    - Can be formatted in various ways, with different encodings
      - E.g., *`var=value`* pair *query string*
    - If data is sent to the server, the content type must be set
    - E.g., for a query string:

```
req.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
```

      - Where `req` is an `XMLHttpRequest` object

XMLHttpRequest.readyState

- Attribute that stores the current state of the object
- `Changes throughout the execution:`
  - `0`
    - `XMLHttpRequest.UNSENT`
  - `1`
    - `XMLHttpRequest.OPENED`
  - `2`
    - `XMLHttpRequest.HEADERS_RECEIVED`
  - `3`
    - `XMLHttpRequest.LOADING`
  - `4`
    - `XMLHttpRequest.DONE`

XMLHttpRequest.status

- Stores the HTTP status code of the response to the request
  - E.g.,
    - `200`
    - `404`
    - `500`
    - etc.
  - Before the request completes, will have a value of `0`

XMLHttpRequest.response

- Holds the data returned from the server
  - Type is determined via `XMLHttpRequest.responseType`
- Response data can also be accessed via:
  - `XMLHttpRequest.responseText`
  - `XMLHttpRequest.responseURL`
  - `XMLHttpRequest.responseXML`

XMLHttpRequest.onreadystatechange

- Attribute to which we assign an event listener
  - This will associate the function with the occurrence of the `readystatechange` event
    - This event fires in several places throughout the the execution (each time the state changes)
    - We can check the `XMLHttpRequest.readyState` to see what, if anything, we will do to handle the event
- Note that this attribute should be set before starting the request

Seems rather onerous to parse responseText

- This example is rather simple, what if we wanted complex data back from the server?
  - E.g., data to populate multiple cells of multiple rows of a table?
  - This is where the X in AJAX comes in
    - Asynchronous JavaScript and XML

XML

- Extensible Markup Language
- Data representation format
  - RSS is built on top of XML
- Uses tags in a very similar manner to HTML
  - Can similarly be traversed using the DOM!

XML Example

```
<person>
    <name>John Smith</name>
    <age>25</age>
    <address>
        <streetAddress>21 2nd Street</streetAddress>
        <city>New York</city>
        <state>NY</state>
        <postalCode>10021-3100</postalCode>
    </address>
    <phoneNumbers>
        <phoneNumber>
            <type>mobile</type>
            <number>123 456-7890</number>
        </phoneNumber>
    </phoneNumbers>
    <children></children>
    <spouse></spouse>
</person>
```

...

- Seems a bit unwieldy
  - Very verbose
    - Both to represent data
    - And to parse it with the DOM
- Really, it would be nice to just send Objects back and forth from client to server

Marshalling

- The process of transforming an in-memory representation of an object into a format that can be stored or transferred
- Similar to *serialization*
  - In some languages, the terms will be used interchangeably
  - In others, marshalling and serialization will carry different meanings

Javascript objects are rather simple

- Basically key/value stores
  - This is exploited to bring about a simplified approach to marshalling for use in exchanging JavaScript objects
    - JSON
      - JavaScript Object Notation
      - Uses human-readable text to transmit objects as key value pairs
      - Used implement AJAJ
        - Asynchronous JavaScript and JSON

Basic JSON data types

- Basic data types:
  - Number
    - Signed
    - Can have fractional component
  - String
    - Double-quoted
  - Boolean
    - `true` or `false`
  - Array
    - Enclosed in square brackets
  - Objects
    - key: value pairs in curly braces
  - `null`

JSON Example

```
{   "name": "John Smith",
    "age": 25,
    "address": { "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021-3100"
    },
    "phoneNumbers": [ { "type": "mobile",
            "number": "123 456-7890"
        },
        { "type": "office",
            "number": "646 555-4567"
        },
    ],
  "children": [],
  "spouse": null
}
```

So we can have the page update itself

- … in response to user actions
  - When else would we want the page to update itself *without* some prompting user action?
    - What about if new information is available on the server?

Polling

- Periodically request updates from the server
- How to accomplish this?
  - A loop with a call to a sleep function?
    - Not very event-driven…

JavaScript Timers

- `window.setTimeout()`
- `window.setInterval()`
- `window.clearTimeout()`
- `window.clearInterval()`

If we're polling data from the server anyway...

- Should we even bother populating a page to send on the server side?