

AJAX & Polling

User Interaction

- Our goal is to add more interactivity to our site
- How to have users interact with each other?
 - Form to submit data
 - Page reloads after submission
 - How does a user get updates when someone submits a form? Reload the page?
- We want our sites to update without a refresh

Goal: Chat App

- We want to build a simple chat app
 - Users can send messages
 - All other users can see those messages without taking any action
- We'll need
 - A form to accept chat messages and send them to a path on the server
 - A path to serve the chat history
 - A way to send GET/POST requests without a refresh

AJAX

Asynchronous JavaScript [And XML]

A way to make HTTP requests using JavaScript *after* the page loads

Can make HTTP GET and POST requests

AJAX - HTTP GET Request

```
var request = new XMLHttpRequest();
request.onreadystatechange = function(){
    if (this.readyState === 4 && this.status === 200){
        console.log(this.response);
        // Do something with the response
    }
};
request.open("GET", "/path");
request.send();
```

- Use JavaScript to make an AJAX request
- Create an XMLHttpRequest object
- Call "open" to set the request type and path for the request
- Call send to make the request

AJAX - HTTP GET Request

```
var request = new XMLHttpRequest();
request.onreadystatechange = function(){
    if (this.readyState === 4 && this.status === 200){
        console.log(this.response);
        // Do something with the response
    }
};
request.open("GET", "/path");
request.send();
```

- Set onreadystatechange to a function that will be called whenever the ready state changes
- A ready state of 4 means a response has been fully received
 - In this example, when the ready state changes to 4 and the response code is 200 the response is printed to the console
 - This is where the response would be processed

AJAX - HTTP POST Request

```
var request = new XMLHttpRequest();
request.onreadystatechange = function(){
    if (this.readyState === 4 && this.status === 200){
        console.log(this.response);
        // Do something with the response
    }
};
request.open("POST", "/path");
let data = { 'username': "Jesse", 'message': "Welcome" }
request.send(JSON.stringify(data));
```

- To make a post request:
 - Change the method to POST
 - Add the body of your request as an argument to the send method

AJAX - Uses

- We can now make HTTP requests without reloading the page

But why?

AJAX - Uses

Faster page loads

- HTML contains the main structure of the page and very little content
- Any content that takes longer to process is requested via AJAX
 - Request may require database lookups and complex algorithms to generate content
 - Typical when a server is deciding which ad to load
- User sees the page quickly and the content populates as the AJAX responses are received

AJAX - Uses

Improved user experience (UX)

- Can be disruptive if the page reloads every time you interact with the server
 - Uses bandwidth to repeatedly request all the content
 - Can experience flicker, or worse, when the page reloads
 - Clutters browser history
- Allows streaming content

Encodings - Multipart

- We have choices for the format when sending the data of the AJAX request
- We can use multipart formatting by changing the attributes of our forms
- Add an onsubmit attribute that calls your JavaScript function
 - Add "return false" to block the page reload

```
<form enctype="multipart/form-data" id="myForm" onsubmit="sendMessageWithForm(); return false;">
  <label for="form-chat">Chat: </label>
  <input id="form-chat" type="text" name="message"><br/>
  <input type="submit" value="Submit">
</form>
```

Encodings - Multipart

- In JavaScript, create a FormData object using your form element
- Send the FormData object
- Provide the same formatting as submitting the form

```
function sendMessageWithForm() {  
    const formElement = document.getElementById("myForm");  
    const formData = new FormData(formElement);  
  
    const request = new XMLHttpRequest();  
    // onreadystatechange removed for slide  
  
    request.open("POST", "send-message-form");  
    request.send(formData);  
}
```

Encodings - JSON

- Another option: Manually format the data using JSON
- Don't use the form element
- Create a button instead of a submit input

```
<label for="chatInput">Chat: </label>  
<input id="chatInput" type="text" name="message"><br/>  
<button onclick="sendMessage()">Send</button>
```

Encodings - JSON

- Manually read the values of any inputs
- Add the values into a JavaScript object or array
- Convert the data to JSON before sending

```
function sendMessage() {  
  const chatBox = document.getElementById("chatInput");  
  const message = chatBox.value;  
  const request = new XMLHttpRequest();  
  
  // onreadystatechange removed for slide  
  
  request.open("POST", "send-message");  
  const messageObject = {"message": message};  
  request.send(JSON.stringify(messageObject));  
}
```

Rendering Content

Rendering Content

- There's another new design decision
 - When do we render the content?
 - When do we convert raw data into HTML to be added to a page?

Rendering Content

- We've rendered HTML templates on the server before sending a request
 - Client only sees the final HTML
 - Uses the server CPU to render content
- This can be restrictive in certain situations
 - What if you want to add a mobile app that doesn't display HTML?

Rendering Content

- Alternative
 - Serve raw data
 - Render it client-side using JavaScript
- Uses client CPU
 - Increased load times
- Server functionality shifts from hosting the whole web app to hosting an API
 - Serve JSON strings at most paths

Polling

Making it Live

- What if someone chats after you load the page?
 - Have to refresh or send a new AJAX call to get the new data
 - AJAX is preferred, but what triggers the AJAX request?
- Polling
 - Keep sending AJAX requests at fixed intervals to refresh the data

Polling

```
setInterval(getMessages, 1000)
```

- Browser sends requests for updates at regular intervals
- Use `setInterval`
 - Takes a function to be called
 - Takes the number of milliseconds to wait between function calls
- This example calls `getMessages()` every second
 - `getMessages()` makes the AJAX call to get the most recent data from the server and render it on the page

Polling

```
setInterval(getMessages, 1000)
```

- Easy to implement
 - Assuming the AJAX calls are already setup
 - Just telling the browser to keep making requests to the server
- Limitations
 - Users wait upto an entire interval to get new content
 - Lowering the interval length increases server load and bandwidth

Long-Polling

- Server hangs on requests (Intentionally)
- Client makes a long-poll request to get the most current data
 - If there's new data, the server responds just like polling
 - When the response is received, client makes another long-poll request
- If there's no new data, the server does not send a response
- Server waits until there is new data to be sent, then responds
- Timeouts
 - If there's no new data after ~10-20 seconds, server responds with no new data
 - Client gets the response and sends a new long-poll request

Long-Polling

- End result
 - The client always has a request waiting at the server
 - Whenever the server has data to send to the client, it responds to the waiting request
 - Real-time updates!
 - Minimal delays between users without excess server load
 - *If designed properly. This is not true if each requests requires its own thread
- We'll reach this same goal with WebSockets
 - More modern solution

Long-Polling

- Even though WebSockets is a more modern solution, many major site still use long-polling
 - I.e. You may still encounter this in your career
- Long-polling only uses HTTP
 - Compatible with very old browsers!