AJAX

SWE 432, Fall 2016
Design and Implementation of Software for the Web
Today’s Objectives

• Learn how to interact with remote hosts from a web page using AJAX
• Learn how to use Firebase web service to persist and synchronize data in realtime
AJAX: Asynchronous JavaScript and XML

- Set of technologies to send and receive data from server asynchronously without interfering with behavior of page
  - HTML & CSS
  - DOM Manipulation
  - JSON or XML for data interchange
  - XMLHttpRequest for asynchronous communication
  - JavaScript
  - Originally defined for XML. But representation independent, and now used mostly for JSON.
History

• 1998: Microsoft Outlook Web App implements first XMLHttpRequest script
• 2004: Google releases Gmail with AJAX
• 2005: Google Maps with AJAX
• 2006: W3C releases draft of XMLHttpRequest standard

Synchronous vs. Asynchronous Requests

- Classic web apps require user to wait for response to server
- Asynchronous requests enable user to continue to interact with app
Example - Lazy Content Loading

- User changes visible viewport
  - JS code renders new area of map based on updated viewport
- Check tile cache
  - If in cache, load tile from cache
  - If not in cache,
    - request tile from Google Maps Server
Lazy Content Loading

• Advantages:
  • Can have *vast* dataset that the user feels as if they are interacting with in real time
  • Only need to download content that user actually needs
  • Can (sometimes) do computation on client with really simple server that just fetches appropriate part of large data set
Some Uses for AJAX

- Lazily load content only when requested
  - e.g., FB newsfeed, Google Maps tile loading
- Load parts of web page from different hosts
  - e.g., advertisements, embedded Twitter widget, …
- Persist user data
  - In some cases, can do all computation client side
  - Enables building web app *without dedicated backend*
- Submit form data to server
Single Page Application Site

Browser

HTTP Request

Web Server

HTML

Domain logic tier

Persistence tier

Database

HTML elements

Javascript

events

HTTP Response (JSON)
Single Page Application (SPA)

- Client-side logic sends messages to server, receives response
- Logic is associated with a single HTML page, written in Javascript
- HTML elements dynamically added and removed through DOM manipulation
- Processing that does not require server may occur entirely client side, dramatically increasing responsiveness & reducing needed server resources

```
Projects:
<ol id="new-projects"></ol>

<script>
$( "#new-projects" ).load( "/resources/load.html #projects li" );
</script>
</body>
</html>
```

- Classic example: Gmail
Example: Weather

• Let’s use a Web Service API to get the current weather.
• Will use the WeatherUnderground API.
  • [https://www.wunderground.com/weather/api/d/docs?MR=1](https://www.wunderground.com/weather/api/d/docs?MR=1)
Recall: HTTP

HTTP Request

HTTP GET http://api.wunderground.com/api/3bee87321900cf14/conditions/q/VA/Fairfax.json

HTTP Response

HTTP/1.1 200 OK
Server: Apache/2.2.15 (CentOS)
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
X-CreationTime: 0.134
Last-Modified: Mon, 19 Sep 2016 17:37:52 GMT
Content-Type: application/json; charset=UTF-8
Expires: Mon, 19 Sep 2016 17:38:42 GMT
Cache-Control: max-age=0, no-cache
Pragma: no-cache
Date: Mon, 19 Sep 2016 17:38:42 GMT
Content-Length: 2589
Connection: keep-alive

{
  "response": {
    "version": "0.1",
    "termsOfService": "http://www.wunderground.com/weather/api/d/terms.html",
    "features": {
      "conditions": 1
    }
  },
  "current_observation": {
    "image": {
      "url": "http://icons.wxug.com/graphics/wu2/logo_130x80.png",
      "title": "Weather Underground"
    }
  }
}
Recall: HTTP

HTTP Request

HTTP GET http://api.wunderground.com/api/3bee87321900cf14/conditions/q/VA/Fairfax.json

HTTP Response

HTTP/1.1 200 OK
Server: Apache/2.2.15 (CentOS)
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
X-CreationTime: 0.134
Last-Modified: Mon, 19 Sep 2016 17:37:52 GMT
Content-Type: application/json; charset=UTF-8
Expires: Mon, 19 Sep 2016 17:38:42 GMT
Cache-Control: max-age=0, no-cache
Pragma: no-cache
Date: Mon, 19 Sep 2016 17:38:42 GMT
Content-Length: 2589
Connection: keep-alive

JSON format response

```json
{
"response": {
"version":"0.1",
"termsOfService":"http://www.wunderground.com/weather/api/d/terms.html",
"features": {
"conditions": 1
}
},
"current_observation": {
"image": {
"url":"http://icons.wxug.com/graphics/wu2/logo_130x80.png",
"title":"Weather Underground"
}
}
}
```

JSON data
function reqListener () {
  console.log(JSON.parse(this.responseText));
}

var oReq = new XMLHttpRequest();
oReq.addEventListener("load", reqListener);
oReq.open("GET", "http://api.wunderground.com/api/3bee87321900cf14/conditions/q/VA/Fairfax.json");
oReq.send();
jQuery AJAX

```javascript
var jqxhr = $.ajax(
    "http://api.wunderground.com/api/3bee87321900cf14/conditions/q/VA/Fairfax.json"
).done(function() {
    console.log( jqxhr.responseJSON );  // JSON of response data
});
```

- Convenience wrapper for making HTTP requests using jQuery
- Defaults to GET method
  - Can specify post with type: “POST” or using $.post(…)
- Looks at content-type response header to determine parser
  - Can override with data-type (e.g., data-type: “JSON”)

http://api.jquery.com/jquery.ajax/
AJAX w/ jQuery

• Can use AJAX to load or send resources of any type
  • HTML, CSS, JSON, text, JS, XML, images, …
  • Content-Type HTTP header describes format
• Load JS file and run it
  • $.load(…)
• Combine DOM manipulation w/ HTTP request
  • $("#result").load( "ajax/test.html" );
  • Requests data from server, updates selected elements with returned HTML
Demo: Sentiment Analysis Web Service

Very similar to the news demo I posted online

But instead we use POST + AJAX

Tricks: Host static files using express
Troubleshooting AJAX

- Make sure that the data sent to server is a JSON object
- Not an array, String, primitive, etc.
- Check for errors (will look at this next time)
- Inspect HTTP requests through the Network tab on the Developer tools
API keys

- Problem: limiting request to a web service
  - Handling requests costs time and money
  - But still want to let developers play with service for free
  - And want to bill heavy users based on usage
- Solution: API key
  - e.g., AIzaSyDPyU6iHOovjYYONyFxixl9NRYQKlxfR0A
  - Generate a unique key for each user
  - Require all calls to API to provide key
  - Monitor use & bill based on key provided
  - DO NOT want to let these API keys be released publicly!
Packaging web services

• Web services are sometimes packaged as library.
  • Include an external library through a <script> include
  • Library itself makes AJAX calls to remote host.

• Advantages
  • Makes possible higher level abstractions for interacting with web service
    • Don’t have to worry about individual HTTP requests
  • Can implement caching of data received from server
  • Can maintain local state of data when server may not be available
Example: Firebase Realtime Database

- Google web service
  - https://firebase.google.com/docs/database/
- Realtime database
  - Data stored to remote web service
  - Data synchronized to clients in real time
- Simple API
  - Offers library wrapping AJAX calls
  - Handles synchronization of data
- Can build web apps with persistence without backend
- Magically makes it look like all data is local
Firebase data model: JSON

- JSON format data
  - Hierarchic tree of key/value pairs
- Can access arbitrary node in tree
  - Just like JSON object...
- Offers realtime console
  - Shows data values in realtime
  - Edit data values
Firebase data model: JSON

Your App Code

Updates objects

Firebase API

Sends callbacks

AJAX Magic

Firebase remote servers
function writeUserData(userId, name, email, imageUrl) {
  firebase.database().ref('users/' + userId).set(
    {
      username: name,
      email: email,
      profile_picture: imageUrl
    }
  );
}

“On the active firebase database”
Must be initialized first (coming soon....).

“Get the users/[userID] node”
Arbitrary nodes in the tree can be addressed by their path.

“Set value”
Sets the value to specified JSON.
Storing Data: Push

```javascript
var key = firebase.database().ref().child('posts').push(
    { author: username, uid: uid, body: body, title: title });
```

- What about storing collections?
  - Use push to create key automatically
  - All data MUST have a key so it can be uniquely referenced
    - Arrays given index keys
  - You really should not ever make your own keys
  - Should never have multiple clients synchronizing an array
    - Local indexes could get of sync with remote keys
    - Instead, use JSON object with number as key
Storing Data: Delete

```javascript
firebase.database().ref().child('posts').remove();

Removes the ‘posts’ subtree.
```

- Can delete a subtree by setting value to null or by calling remove on ref
Listening to data changes

```javascript
var starCountRef = firebase.database().ref('posts/' + postId + '/starCount');
starCountRef.on('value', function(snapshot) {
    updateStarCount(postElement, snapshot.val());
});
```

“When values changes, invoke function”
Specify a subtree by creating a reference to a path. Listen to one or more events by using `on(eventName, eventHandlerFunction(snapshot))`.

• Read data by *listening* to changes to specific subtrees
• Events will be generated for initial values and then for each subsequent update
Data Update Events

• Types of events
  • value: entire contents of a path
  • child_added
  • child_changed
  • child_removed

• Can listen to events on any part of subtree
  • Could have subtrees that correspond to different collections of data
  • Should always listen to lowest subtree of interest to minimize extraneous communication

• Can read data exactly one time (and not get updates) using once
Ordering data

• Data is by, default, ordered by key in ascending order
  • e.g., numeric index keys are ordered from 0…n
  • e.g., alphanumeric keys are ordered in alphanumeric order

• Can get only first (or last) n elements
  • e.g., get n most recent news items

```javascript
var recentPostsRef = firebase.database().ref('posts').limitToLast(100);
recentPostsRef.once('value', function(snapshot) {
    displayPost(snapshot.val());
});
```
Setting up Firebase

- Detailed instructions to create project, get API key
  - [https://firebase.google.com/docs/web/setup](https://firebase.google.com/docs/web/setup)
  - You should run through web server, not localhost

```html
<script src="https://www.gstatic.com/firebasejs/3.4.0/firebase.js"></script>
<script>
// Initialize Firebase
// TODO: Replace with your project's customized code snippet
var config = {
  apiKey: "<API_KEY>",
  authDomain: "<PROJECT_ID>.firebaseapp.com",
  databaseURL: "https://<DATABASE_NAME>.firebaseio.com",
  storageBucket: "<BUCKET>.appspot.com",
};
firebase.initializeApp(config);
</script>
```
Permissions

• By default, Firebase *requires* authentication
  • All unauthenticated requests will be refused
  • Do not want anyone with your URL to steal, destroy your production data
  • Will look at authentication in later lecture
  • For development, ok to allow anonymous access
Firebase Demo