```
<!-- 1. Open up:

        1) Past iTunes ajax lecture: http://303.itpwebdev.com/~nayeon/ajax-itunes/itunes.html
        2) Current Song app: http://303.itpwebdev.com/~nayeon/song_app/search_form.php
        3) Today's completed result: http://303.itpwebdev.com/~nayeon/ajax-php/frontend.html
        4) SPA example, open quiz: https://www.warbyparker.com/
        5) SPA example 2: http://www.thunderbirds.com/

        Remember AJAX? Back in JS days we used AJAX to use iTunes and Movie DB API and get results without refreshing the page. When AJAX came out this was huge because it feels faster (no reload) and smoother, like a desktop application.

        Show iTunes lecture.

        Then with PHP we used different pages to access the DB. Every time the DB had to do something, we refreshed the page to something else. This is typical web behavior. (Show song app).

        These modern days thanks to AJAX we don't need to refresh everytime we interact with the DB. A single-page application (SPA) is a web application or web site that interacts with the user by dynamically rewriting the current page rather than loading entire new pages from a server. Makes it a smoother interaction, kinda like a desktop application. Here are some examples of SPAs (warby parker and thunderbirds examples).

        This is why JS libraries and frameworks like React, Angular, Ember, etc are so popular these days -- b/c they make SPA easier to build.

        Building a SPA can take a lot of work. And notice that not ALL pages need to be SPA. It's typical of a web application where certain pieces of the pages are SPA (e.g. warby parker quiz) and the rest are traditional web pages.

        Today we'll make our own small SPA with AJAX and PHP using the same old song database. Show completed example.

        Show slide to show AJAX & PHP workflow. NEXT: scroll down to AJAX function.

-->
<!DOCTYPE html>
<html>
<head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <title>AJAX and PHP</title>
        <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta/css/bootstrap.min.css">
        <style>
                li {
                        margin-top: 5px;
                }
        </style>
```

```
 38   </head>
 39   <body>
 40
 41           <div class="container">
 42                   <div class="row">
 43                           <div id="test-output"></div>
 44                           <h1 class="col-12 mt-3">Song DB Search</h1>
 45                   </div> <!-- .row -->
 46
 47                   <div class="row">
 48                           <form action="" method="" class="form-inline col-12 mt-3">
 49                                   <div class="form-group">
 50                                           <label for="search-term-id" class="sr-only">Search:</label>
 51                                           <input type="text" class="form-control" id="search-term-id" placeholder="
Search..." name="search-term">
 52                                   </div>
 53                                   <button type="submit" class="btn btn-primary ml-3">Search</button>
 54                           </form>
 55                   </div> <!-- .row -->
 56
 57                   <div class="row">
 58
 59                           <h4 class="col-12 mt-4">Search Results:</h4>
 60
 61                           <div class="col-12">
 62                                   <ol id="search-results">
 63                                           <li>Track Name 1</li>
 64                                           <li>Track Name 2</li>
 65                                   </ol>
 66                           </div>
 67
 68                   </div> <!-- .row -->
 69           </div> <!-- .container -->
 70
 71           <script>
 72
 73                   // 3. Make a quick GET request to backend.php (no parameters yet). Return function is an
anonymous function that simply console.logs out the result. Open up backend.php to show that it has one array and
that's it. So ajax is calling the php file, and the code does get run because we see no errors. However, no way to know
that. NEXT: go to backend.php to return something to the
 74
 75                   // ajaxGet('backend.php', function(results){
 76                           // console.log(results);
 77
 78                           // 5. Write JS to display result in browser. Show that the data type also gets printed
out because var_dump does that. Not so useful. So what else can we try? NEXT: back to backend.php
 79
```

```
 80                         // document.querySelector("#test-output").innerHTML = results;
 81
 82                         // 11. Dot notation to try to grab values? Not gonna work since this is still a STRING.
 83
 84                         // console.log(results.first_name);
 85
 86                         // 12. Need to convert this string in to JS objects. Can do that here... but why not do
it from the ajax function so that we don't have to do the extra conversion. Go to the function declaration.
 87                         // results = JSON.parse(results);
 88
 89                         // 13. Display to browser using dot notation.
 90                         // document.querySelector("#test-output").innerHTML = results.first_name;
 91
 92                 //});
 93
 94                 // 14. Ok so now that we know how frontend/backend can talk to each other, let's talk about
sending data to the backend. In this search example, whatever the user searches for, we will need to send it to the
backend and the backend will have to run SQL, then return results. Let's add parameters. We can add as many as we'd
like. Send parameters using AJAX. Add parameters here using the ? and console log the results. NEXT: go to backend.php
to grab the parameters.
 95
 96                 // ajaxGet('backend.php?firstName=Tommy&lastName=Trojan', function(results){
 97
 98                 //      console.log(results);
 99                 //      document.querySelector("#test-output").innerHTML = results;
100                 //      console.log(results.first_name);
101                 //      document.querySelector("#test-output").innerHTML = results.first_name;
102
103                 // });
104
105
106                 // 20. Copy paste ajaxGet, change to ajaxPost, and set three parameters. NEXT: Go to backend.php
to validate $_POST is being set.
107
108                 // ajaxPost('backend.php', 'firstName=Tommy&lastName=Trojan', function(results){
109
110                 //      console.log(results);
111
112                 // });
113
114
115                 // 2. Review JS ajax. It will just return the responseText. Let's use this function make a quick
GET request to our backend.php
116                 function ajaxGet(endpointUrl, returnFunction){
117                         var xhr = new XMLHttpRequest();
118                         xhr.open('GET', endpointUrl, true);
119                         xhr.onreadystatechange = function(){
```

```
120                                     if (xhr.readyState == XMLHttpRequest.DONE) {
121                                             if (xhr.status == 200) {
122                                                     // returnFunction( xhr.responseText );
123
124                                                     // 12. JSON.parse() to parse the JSON string. Show on console
that now it's JS objects. We can grab properties now.
125                                                     // returnFunction( JSON.parse(xhr.responseText) );
126                                                     returnFunction( xhr.responseText );
127                                             } else {
128                                                     alert('AJAX Error.');
129                                                     console.log(xhr.status);
130                                             }
131                                     }
132                             }
133                             xhr.send();
134                     };
135
136
137                     // 16. Create separate function for POST requests. Copy/paste the GET function. With POST
requests, cant simply pass in parameters in the URL. We will need a third parameter just for the variables we want to
pass (postData).
138                     function ajaxPost(endpointUrl, postData, returnFunction){
139                             var xhr = new XMLHttpRequest();
140                             // 17. Specify POST request here
141                             xhr.open('POST', endpointUrl, true);
142
143                             // 18. POST request usually needs some extra header information. If sending from a Form,
use this content-type. Note there are other content-types. this one happens to be the most common. Note it says
urlencoded, which is the format of the variables. Send any other headers as neccesary.
144                             xhr.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');
145                             xhr.onreadystatechange = function(){
146                                     if (xhr.readyState == XMLHttpRequest.DONE) {
147                                             if (xhr.status == 200) {
148
149                                                     // returnFunction( xhr.responseText );
150                                                     returnFunction( JSON.parse(xhr.responseText) );
151
152                                             } else {
153                                                     alert('AJAX Error.');
154                                                     console.log(xhr.status);
155                                             }
156                                     }
157                             }
158                             // 19. Send the body of the post request. NEXT: Scroll back up and copy paste the
ajaxGet call to make ajaxPost. Comment out ajaxGet.
159                             xhr.send(postData);
160                     };
```

```
161
162
163
164
165                     // ---- Form handling
166                     // 22. Now we can send our search term and send it use it to search from our DB.
167                     document.querySelector('form').onsubmit = function() {
168
169                             // 23. Get user's search term
170                             var searchTerm = document.querySelector('#search-term-id').value.trim();
171
172                             // 24. Call ajax function, pass in the search term, log out results. NEXT: go to
backend.php to make SQL commands to DB.
173
174                             ajaxGet('backend.php?term=' + searchTerm, function(results) {
175
176                                     console.log(results);
177
178                                     results = JSON.parse(results);
179
180                                     // 29. Grab the list element
181                                     var resultsList = document.querySelector('#search-results');
182
183                                     // 30. Don't forget to clear all the previous elements upon every search. Now
you can keep searching without leaving the page.
184                                     while( resultsList.hasChildNodes()) {
185
186                                             resultsList.removeChild(resultsList.lastChild);
187                                     }
188
189                                     // 31. Run through the results and append them to resultsList.
190                                     for( var i = 0; i < results.length; i++) {
191                                             var li = document.createElement('li');
192                                             li.innerHTML = results[i].name;
193                                             resultsList.appendChild(li);
194                                     }
195                             })
196
197                             event.preventDefault();
198                             // return false;
199                     }
200
201             </script>
202
203     </body>
204     </html>
```

```php
<?php
        $php_array = [
                "first_name" => "Tommy",
                "last_name" => "Trojan",
                "age" => 21,
                "phone" => [
                        "cell" => "123-123-1234",

                        "home" => "456-456-4567"
                ],
        ];

        // 4. Three ways that we have learned so far on how to output information. First is var_dump(). Show on console - it shows data type as well. NEXT: display this on the browser, back to frontend.html.

        //var_dump("Hello world!");

        // 6. echo is also used to output HTML. Comment out var_dump() above. Refresh. Browser will show Hello World now.
        // echo "Hello World!";

        // 7. Classic return statement. Comment out echo above. Shows nothing :( the ajax call only returns string.
        // return "Hello World";

        // 8. However, with web applications we are not going to get back a simple string. It's gonna be a bunch of data, probably an associative array that looks like the one on top given to you. Try just echoing at first. Going to have errors b/c cant echo out arrays.

        // echo $php_array;

        // 9. Can't just display an array. Need to do some kind of conversion so that browser can somehow read it. Anyone guess what format we can convert to? To convert associative array to a JSON formatted string, use json_encode().

        // echo json_encode($php_array);

        // 10. That means we can use dot notation to grab values, right? NEXT: Go back to ajaxGet() in frontend.html and try dot notation.


        // 15. Show that we got the paramenters. So you can see where we are going with this. When user types in a search term, we can grab it using $_GET (an associative array) just like what we have done with PHP before. Then run a SQL query to grab the results we want. But before we do that, let's also cover POST requests. NEXT: back to frontend.html to create a POST request.
        // echo json_encode($_GET);

```

```php

        // 21. Verify POST is working. Now that those cases are covered, let's finish our demo here. This is a simple search, no sensitive data sent, so a GET request will be sufficient. NEXT: back to frontend.php to handle the form submission.
        // echo json_encode($_POST);


        // 25. Comment echo statements before. Connect to the database.
        $host = "303.itpwebdev.com";
        $user = 'nayeon_db_user_5';
        $pass = 'uscItp2018';
        $db = 'nayeon_song_db_default';

        $mysqli = new mysqli($host, $user, $pass, $db);
        // error checking here. omitted for time sake.
        if ( $mysqli->errno ) {
                echo $mysqli->error;
                exit();
        }

        // 26. SELECT statement, pass in the search term
        $sql = "SELECT * FROM tracks WHERE name LIKE '%" . $_GET['term'] . "%' LIMIT 10;";

        $results = $mysqli->query($sql);
        // check for errors.
        if ( !$results ) {
                echo $mysqli->error;
                exit();
        }

        $mysqli->close();


        // 27. At this point, we would run a while loop, use fetch_assoc() and etc. But we need to give this data back to the frontend, so let's store all this in an array.

        $results_array = [];

        while( $row = $results->fetch_assoc() ) {
                array_push( $results_array, $row );
        }

        // 28. Convert the assoc array into a json string. This string going to be converted to JSON objects. Back to frontend.html to see the results in console log. Try a couple more searches to show that no page refresh is required. NEXT: back to frontend.php to display the results.
        echo json_encode( $results_array );
```

```
78
79    ?>
```