**Filter with AJAX**

Here is a brief coverage of filtering with AJAX for the website assignment.

**Prerequisites**

JQuery, JQuery UI, Ajax

If you don't know any of the above languages, do not fret. This will be covered very briefly.

**Objectives**

Creating a filter with a control UI and live updating div, to adjust number of weeks shown.

**Filter**

Suppose we want to create a page with live updating of contents. This requires AJAX. Here, we create two divisions:

1. Sidebar with control UI - Here, we use range to control number of weeks
2. Main container - Here, we update the contents

Graphical user interface

Description automatically generated

CONTROL UI

We will be using a slider. Without JQuery UI, the slider we use only has one thumb (red filled circle above). We want two sliders.

Attach the cdn links into the <head> </head> as below:

|  |
| --- |
| <head>      <title>Courses</title>      <meta name="viewport" content="width=device-width">        <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/jqueryui/1.12.1/themes/cupertino/jquery-ui.min.css" />      <script src="https://code.jquery.com/jquery-3.6.0.js"></script>      <script src="https://code.jquery.com/ui/1.13.0/jquery-ui.js"></script>  </head> |

The stylesheet stylises the UI, so we need that.

We then want a filter container, with the following:

* Two inputs, hidden, containing default values
* One input, text, to show the user what they are adjusting
* One div, for slider

|  |
| --- |
| <input type="hidden" id="hidden\_minimum\_week" value="<?php echo $weekno; ?>"/>  <input type="hidden" id="hidden\_maximum\_week" value="<?php echo $weekno + 5; ?>" />  <input type="text" id="filter-weeks" readonly class="slider-text" value="<?php echo 'Week '. $weekno .' - Week '.($weekno + 5).''; ?>" />  <div id="slider-range"></div> |

The slider UI will be created automatically with JQuery. Place this as a script below every element (but still inside body).

$(document).ready(function() { });

The above statement runs all code inside { } when the document first loads.

|  |
| --- |
| <script>  $(document).ready(function() {      $( "#slider-range" ).slider({          range: true,          min: 1,          max: 54,          values: [ 45, 50 ],          slide: function( event, ui ) {              $( "#filter-weeks" ).val( "Week " + ui.values[ 0 ] + " - Week " + ui.values[ 1 ] );              $('#hidden\_minimum\_week').val(ui.values[0]);              $('#hidden\_maximum\_week').val(ui.values[1]);          }      });  });  </script> |

The slider has the following parameters:

* Range - should there be a range? (True / False)
* Min - set min value
* Max - set max value
* Values - set values that you want user to see when they first open the page
* Step - how much increment should the thumbs do?

Then, create a function that will:

1. Update the text when the slider moves (controlled by event and ui)
2. Append new values into the hidden inputs (after the thumb is moved)

The control UI is done.

**Updating Content with AJAX**

We need some JQuery first. Replace the above script with the following below:

|  |
| --- |
| <script>      $(document).ready(function() {          filter\_data();          function filter\_data() {              var action = 'week';              var minimum\_week = $('#hidden\_minimum\_week').val();              var maximum\_week = $('#hidden\_maximum\_week').val();              $.ajax({                  url: "week.php",                  method: "POST",                  data: {action:action, minimum\_week:minimum\_week, maximum\_week:maximum\_week},                  success:function(data) {                      $('#load-weeks').html(data);                  }              });          }          $( "#slider-range" ).slider({              range: true,              min: 1,              max: 54,              values: [ 45, 50 ],              slide: function( event, ui ) {                  $( "#filter-weeks" ).val( "Week " + ui.values[ 0 ] + " - Week " + ui.values[ 1 ] );                  $('#hidden\_minimum\_week').val(ui.values[0]);                  $('#hidden\_maximum\_week').val(ui.values[1]);                  filter\_data();              }          });      });  </script> |

We have a function, filter\_data which actively filters out data.

fitler\_data() has the following parameters:

1. Store an array of objects 'week' under action ([Reference](https://stackoverflow.com/questions/55727468/how-to-define-an-ajax-action/55727761))
2. Store the values of hidden inputs inside each variable

Then, the Ajax action starts here:

1. url - what should ajax do? change the contents based on the php file
2. method - always POST for any data submission
3. data - encapsulate every parameter to be sent to the url

If successful, the data should be returned and updated accordingly.

$('#load-weeks').html(data)

#load-weeks here is the place where the filtered data should appear.

**Database Prerequisites**

*Ignore this and the following part if you are not interested in how the PHP works.*

We assume at least a table stem0301 initialised in the database. This table contains all the files descriptions and associated week number.

**PHP Action**

As in the above code, we have a php file, called week.php.

* Remember to always include db.php and auth.php for authorisation

Once submitted from the Jquery Ajax, this php file will look through if the action is submitted.

if(isset($\_POST["action"])

PARAMETERS

The parameters brought from Ajax is then stored into local variables here. (This is for easy reuse, so we do not need to use $\_POST every time we want to reuse)

* $start - store starting week (left thumb of slider)
* $end - store ending week (right thumb)
* $course - store course id (for querying table later on)

|  |
| --- |
| $start = $\_POST['minimum\_week'];  $end = $\_POST['maximum\_week'];  $course = strtolower($\_SESSION['current\_course']); |

SQL WITH INJECTION PREVENTION

Doing $query = 'SELECT …'; enables a threat: SQL injection. This is where other users can inject and modify SQL to be sent.

To prevent this, we will use the prepare keyword.

$query\_string = "SELECT \* FROM {$course} WHERE week = ?;"

The above is optional, as this can be placed directly inside prepare later. ? means the data to be binded later in bindparam(). We will be binding $start, and then print every week (by using a while loop, and incrementing $start).

* Note: If '" .$course. "' is used, you will need to modify. What this means is that if it is used, our query\_string will actually be: SELECT \* FROM 'stem0301'
* Reference: <https://stackoverflow.com/tags/pdo/info> (PDO Prepared statements and identifiers)

$query = $conn->prepare($query\_string);

This prepares the string for query later. It prevents SQL injection.

Note: if query\_string is wrong, an error will show related to bindparam(). This is not the fault of bindparam(), so check that the string $query is exactly what SQL requires.

If the database does not have the table, the querying fails. So, we use a try, catch statement.

|  |
| --- |
| try {      // prevent SQL injection      $queryweeks = $con->prepare($query\_string);      // if table does not exist      if($queryweeks === false) {          throw new Exception("Course does not exist.");      }  } catch(Exception $e) {      echo $e->getMessage();  } |

If stem0301 is not found, then it prints Course does not exit.

Then, we will do the binding of parameters if the table does exist.

|  |
| --- |
| try {      // prevent SQL injection      $queryweeks = $con->prepare($query\_string);      // if table does not exist      if($queryweeks === false) {          throw new Exception("Course does not exist.");      } else {          // bind start parameter to queryweeks          $queryweeks->bind\_param("s", $start);  } |

Here, $start will take the position of "?" in our query\_string.

Suppose the user only wants to see from week 45 - 50. Then, $start = 45, $end = 50. For every week, we echo a section. The snippet below is placed below $queryweeks->bind\_param().

|  |
| --- |
| // create week sections  while($start <= $end) {  $queryweeks->execute();    $weeklists = $queryweeks->get\_result();  $week = mysqli\_fetch\_array($weeklists);  } |

$queryweeks->execute()

$weeklists = $queryweeks->get\_result();

This executes the statement, and does the same as $mysqli\_query().

$week = mysqli\_fetch\_array($weeklists);

This fetches weeklists array. $week["column\_name"] is how we access each record's data. Here, we get an array of records with the week number of $start.

For example, if there are 3 records with week = 3, then $week has 3 records.

To access each record, we have to loop across each.

|  |
| --- |
| while($week = mysqli\_fetch\_array($weeklists)) |

The rest are self-explanatory.