|  |  |  |
| --- | --- | --- |
|  | **FACULTY OF COMPUTING, ENGINEERING and SCIENCE** | Final mark awarded:\_\_\_\_\_ |

**Assessment Cover Sheet and Feedback Form 2016/17**

|  |  |  |  |
| --- | --- | --- | --- |
| Module Code:  SE3S604 | Module Title:  Adv. Int. & Mob. Comp, | | Module Lecturer:  Keith Norris |
| Assessment Title and Tasks: JQuery and PHP PDOs (Set Tasks - not-time constrained 3) | | | Assessment No. 3 |
| No. of pages submitted in total including this page:  N/A | | | Word Count of submission  (if applicable): 2691 |
| Date Set:  5/12/2016 | | Submission Date:  30/1/2017 | Return Date:  28/2/2017 |

|  |  |
| --- | --- |
| ***Part A: Record of Submission (to be completed by Student)*** | |
| **Extenuating Circumstances**  If there are any exceptional circumstances that may have affected your ability to undertake or submit this assignment, make sure you contact the Advice Centre on your campus prior to your submission deadline. | |
| **Fit to sit policy**:  The University operates a fit to sit policy whereby you, in submitting or presenting yourself for an assessment, are declaring that you are fit to sit the assessment. You cannot subsequently claim that your performance in this assessment was affected by extenuating factors. | |
| **Plagiarism and Unfair Practice Declaration:**  By submitting this assessment, you declare that it is your own work and that the sources of information and material you have used (including the internet) have been fully identified and properly acknowledged as required (University Academic Integrity Regulations). Additionally, the work presented has not been submitted for any other assessment. You also understand that the Faculty reserves the right to investigate allegations of plagiarism or unfair practice which, if proven, could result in a fail in this assessment and may affect your progress. | |
| **Details of Submission:**  Note that all work handed in after the submission date and within 5 working days will be capped at 40%. No marks will be awarded if the assessment is submitted after the late submission date unless extenuating circumstances are applied for and accepted.  Work should be submitted as detailed in your student handbook. You are responsible for checking the method of submission. | |
| **You are required to acknowledge that you have read the above statements by writing your student number (s) in the box:** | Student Number(s):  13065858 |

**IT IS YOUR RESPONSIBILITY TO KEEP A RECORD OF ALL WORK SUBMITTED**

|  |
| --- |
| **Part B: Marking and Assessment** |
| This assignment will be marked out of 100  This assignment contributes to 25% of the total module marks.  This assignment is bonded: |
| **Assessment Task:**  See attached details |
| **Learning Outcomes to be assessed** (as specified in the validated module descriptor <http://icis.glam.ac.uk>):  To provide the theoretical and practical knowledge of the technologies associated with building web-based and mobile applications.  To develop skills to design, implement and critically evaluate such technologies and applications |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Grading Criteria:**   |  |  |  | | --- | --- | --- | | **Marking Scheme** | **Marks Available** | **Marks Awarded** | | Client & Server-side logic (PDO processing & JSON) to display suitable data tables | **25** |  | | Client & Server-side logic (PDO processing & JSON) to append new element values to the database | **25** |  | | Client-side logic (PDO processing & JSON) to search for specified contents | **25** |  | | Report | **25** |  | |

# ASSESSMENT - DETAILED REQUIREMENTS

**The following is an outline of the requirements of the task. You MUST clarify any uncertainties with module lecturers**

You are required to produce web based files to present a single html web page to a client browser demonstrating JavaScript/Ajax framework techniques associated with:

* PHP PDO processing;
* JQuery framework;
* JSON;
* CSS.

Assuming a number of database tables exist on your MySQL server, the html web page must provide the user with the means to choose a single database table and subsequently display this as a table on the same web page without refreshing the whole web page. A drop-down list box is to be used to present the list of available database tables to the user. The drop-down list box contents must reflect the database tables presently available on the MySQL server without further web development amendments i.e. available database table names must not be hardcoded into the web page. This information must be obtained when the document has been loaded and must be returned as JSON.

The web page layout must be similar to the following:

Search functionality (Only available when table is visible)

Footer

Drop-down box

Header

Table (Only available once a database table has been selected)

Insert functionality (Only available when table is visible)

The database tables must comprise an autoincrement numeric field and a string fields. No other data types may be employed and each table must comprise a different number of fields and fieldnames.

Upon receiving the returned JSON, the web page should construct appropriate html markup for rendering. You should not use TableSorter2 in this cw or similar components.

The drop-down list box and table must be populated as necessary using JQuery techniques. All Client to Server interactions must utilise getJSON and all data returned by the Server must be in a JSON format.

You may extend the framework component capabilities as necessary.

The user must be provided with the means to:

* view a table (including field headings) from the database, once a table has been selected;
* append new data to the selected database table. This will require form input fields and php script logic to insert the new detail into the relevant database table.
* search the currently selected database table for a specific field’s content so that only those records satisfying the selection are displayed in the table.

This involves additional input fields and logic to allow the user to select the field and field content.

A drop-down list box must be presented permitting the user to specify the relevant field.

The web page logic should be robust and cater for a variety of situations e.g.:

* No database tables being available.
* Table must only be displayed when a selection has been made.

These are just examples of robustness and do not represent a complete list for the available marks.

You must not utilise older:

* HTML dom selection methods such as getElementById. You must use JQuery’s selector features.
* Ajax techniques to retrieve the JSON returned by the php scripts. You must use JQuery’s getJSON feature.

Your submission must be supported by a report detailing your design, experiences and selection of content, including:

* Overall design documentation.
* Commentary on the selection of framework components used.
* Commentary on JavaScript functionality.
* Commentary on PDO processing.
* An explanation of how your implementation caters for prohibiting the possibility of attempts at SQL Injection together with supporting evidence showing your implementation actually probiting such attempts.
* Evidence of testing using different browsers.

You MUST present the material in a standard expected of level 6 studies i.e. must present arguments of critical evaluation demonstrating an understanding of the concepts investigated/evaluated.

Do not merely implement the overall solution and expect it to work. You are advised to test the individual components of your solution through the use of an appropriate browser’s url using parameters as necessary e.g.

* Reference the php script ( no parameters required ) to obtain the JSON detail representing the contents of the database table names drop-down list box. Should return expected JSON content depicting the database table names.

Knowing individual php functionality works allows you to implement a drop-down list box in an html file.

* Reference the php script (parameter specifying drop-down list selection) to obtain the JSON detail representing the contents of the specified database table. Should return expected JSON content representing the table’s details and contents.

You are required to demonstrate your implementation in the timetabled lab sessions during the two weeks immediately following the submission deadline.

Your submission must be a single zip file comprising:

* A Word version of your report
* All necessary implementation files

Contents

[ASSESSMENT - DETAILED REQUIREMENTS 3](#_Toc473583543)

[Overall Design and Documentation 7](#_Toc473583544)

[Commentary on the selection of framework components used 7](#_Toc473583545)

[PHP 7](#_Toc473583546)

[JavaScript, jQuery and Toastr 7](#_Toc473583547)

[Selectors 7](#_Toc473583548)

[$.getJSON 7](#_Toc473583549)

[$.ajax 8](#_Toc473583550)

[Each 8](#_Toc473583551)

[Toastr 8](#_Toc473583552)

[Append 8](#_Toc473583553)

[.before() 8](#_Toc473583554)

[SQL 9](#_Toc473583555)

[SHOW TABLES 9](#_Toc473583556)

[SELECT \* FROM ".$table 9](#_Toc473583557)

[Prepared Statement 9](#_Toc473583558)

[Insert 9](#_Toc473583559)

[Bootstrap 9](#_Toc473583560)

[Commentary on JavaScript functionality 9](#_Toc473583561)

[Prohibiting possible SQL Injection 15](#_Toc473583562)

[Testing in Different browsers 16](#_Toc473583563)

[Google Chrome 16](#_Toc473583564)

[Internet Explorer 18](#_Toc473583565)

[Microsoft Edge 20](#_Toc473583566)

# Overall Design and Documentation

The program features a robust design to cater the needs of a single page application using PHP and JavaScript techniques to asynchronously pass information from a client to a server. The application allows an end user to select a table from a database using a dropdown list. This will return the results to the end user in the form of a HTML table to view the data.

Once a table has been loaded, a user will be presented with another select menu which will provide the end user with the field related to the current loaded table. The end user can then type details into the search box (which is only displayed once the user has loaded a table). The search will update for each character inserted, and only search for information specific to the field selected from the select menu drop down.

# Commentary on the selection of framework components used

## PHP

The PHP file named ‘controller.php’ controls the logic between client and the server. The PHP file has a selection of functions needed to process data.

The controller.php has two classes defined called Controller and JSONSchema

The JSONSchema class serves the purpose of reporting information back to the developer and evaluating data processed between the client and server to build the JSON responses in a robust and specific approach.

The JSON schema will send a status report back to the client, send messages specific to an action, send the table information including its rows and it will send the data contained in the fields to the client.

The controller class contains the functions needed to process the required data requested by the client.

The default ‘\_\_construct’ function in the controller establishes a connection to the database using PHP Data Objects.

The ‘getTables’ function makes a request to the database using PHP PDO functions to get the names of each table in the database and store it in an array.

The ‘getTableContents’ function accesses the array of tables and loads the fields and values for a specified table passed as an argument in the function. This function will return the data as an associative array using PDO::FETCH\_ASSOC. The associative array will contain just the data of the fields and will not report back the number index’s.

The ‘searchTable’ function will take three parameters which is the table name, the column that is going to be searched and the value that is going to be searched against the column. This function uses a prepared statement to avoid attempts of SQL injection.

The ‘addRow’ function will loop through the table adding a field for each that exists in the database. When the field is added, an empty query is prepended to decorate the function. Once the loop has executed a statement can be generated as a URL which will add the data for each field and save it to the database. jQuery will post the URL back to the controller’s addRow function which will then be executed server side.

## JavaScript, jQuery and Toastr

### Selectors

To select a HTML element, this is done by using the selector “$”, followed by (“#”) for id element or (“.”) for a class followed by its name. An example of where this is used.

Var numberOfCoumns = $(‘#columns > option’).length;

This statement selects the list of options available in a select list identified by id of columns and returns its length.

### $.getJSON

$.getJSON loads JSON-encoded data from the server using a GET HTTP request. An example of using this in my application:

$.getJSON(“controller.php?action=getTables”, function(result) { … });

This will get the JSON the controller action the PHP file returns and populate the list of tables available in the array into a jQuery dropdown menu.

This is a short hand .ajax function.

### $.ajax

$.ajax performs an asynchronous HTTP request stands for asynchronous JavaScript and XML. Using $.ajax you can update a web page without reloading the page, request data from a server after the page has loaded, receive data from a server after the page has loaded and send data to a server in the background.

I have used $.ajax once in the assignment to show how it can be used to send data to a database.

This is done by building up a URL of ‘controller.php?action=addRow&table=’ and appending the selected table to it using a loop.

The full URL will look similar to ‘controller.php?action=addRow&table=client&first\_name=&last\_name=&email=&company=&country\_code=’ but with parameters after each ‘=’ symbol.

$.ajax({

url: saveQuery,

success: function() {

toastr.success('Record added to table');

},

error: function() {

toastr.error('Unable to save data to database');

}

});

Using this the URL is sent as a HTTP Post. You can use ajax fall backs to notify the user of success or failure.

### Each

jQuery’s .each function is an iterator that can loop through arrays and objects. This function has properties such as .length where you can return the size of an array or object.

$.each(result.data[0], function(column, columnsInRow) {

html = html + '<th>' + column + '</th>';

columnArray.push(column);

});

This will loop through the JSON object, loop through each data in the row of index 0 and build a header for the table of data that is specific to that individual table.

### Toastr

Toastr is a framework build in jQuery which has animated helper messages displayed on the screen in a pop up box. This assignment uses toastr.sucess, toastr.error and toastr.info with a string passed in as a parameter to notify the end user of relevant information.

### Append

jQuerys append method allows you to append dynamic HTML to already existing elements in the DOM.

$('#returnedData tbody .newDataRow').append('<td style="vertical-align:middle"><input id="' + currentColumn + '"placeholder="' + currentColumn + '"type="text" class="form-control" /></td>');

This will append a new row to the table with text boxes if the user wishes to add data to the dataset.

### .before()

Much like the append, this function will add dynamic markup to the page before the div that is selected.

## SQL

The SQL statements in this assignment are passed through using PHP objects. However, these are needed for basic ‘CRUD’ functionality. CRUD stands for Create, Read, Update and Delete.

The statements used in this assignment are:

### SHOW TABLES

Show tables is a basic script that is executed inside a specified database. It will return all the databases that exist.

### SELECT \* FROM ".$table

A basic statement that takes a variable as a parameter and builds an SQL statement. If the table selects the ‘users table from the client, PHP will take that variable and make the statement ‘SELECT \* FROM users’, which will return all data in that table.

### Prepared Statement

SELECT \* FROM $table WHERE $column LIKE :value

This will build a statement to select all data from a specified table where the column name is a value that cannot be injected through the client.

### Insert

INSERT INTO $table will generate an SQL statement that will take the variable $table as a parameter.

## Bootstrap

Bootstrap is an open source HTML, CSS and JavaScript front end framework developed by Twitter. It helps to build rich and robust interface components. I have used bootstrap for styling forms, wrapping the table in a bordered container and most importantly it’s helper classes. The helper classes are defined by ‘success’, ‘info’, ‘warning’, ‘danger’ and ‘primary’. These are becoming more relevant throughout web applications making their functionality more accessible.

# Commentary on JavaScript functionality

var url = ("controller.php?action=getTableContents&table=");

// function that checks for an error

function error(data) {

// if the status returned from the controller is 'OK' there is no error

if (data.status && data.status === "OK") {

return false;

}

else {

// there is an error

return true;

}

}

// function to display the errors in toastr popup boxes

function displayError(data) {

switch (data.status) {

case "ERROR":

// displays the error in a red popup box

toastr.error('<b>Status ' + data.status + ': </b>' + data.message);

break;

// displays the warning in a yellow popup box

case "WARNING":

toastr.warning(data.status + ": "+ data.message);

break;

default:

// displays messages that are not errors or warnings but may contain appropriate information

toastr.info(data.status + ": " + data.message);

}

}

// function to delete all table markup from the DOM

function clearHTML() {

// gets the table by its ID using a jQuery selector and calls the remove property

$('#returnedData').remove();

}

// hide contents of a table

function hideTableSection() {

// refactored to remove markup

// on the next version the method calls will go directly to the clearHTML function

clearHTML();

}

// closes and opened table

function closeTable() {

// gets a button by its ID using jQuery selector when it is clicked

$('#closeTable').click(function(){

// refactored to remove its contents from the DOM. On update this no longer needs to be a function calling another function

clearHTML();

});

}

// function for user to change the table

function changeTable() {

// gets the dropdown value using a jQuery selector when a user changes the dropdown

$("#dropdown").change(function() {

var selectedTable = $(this).val();

// calls the getTableFunction passing the new data as a parameter

getTableData(selectedTable);

});

}

// function to refresh table and it's contents

function refreshTable() {

// gets button by its ID using a jquery Selector when clicked

$('#refreshTable').click(function(){

// clears the table

clearHTML();

// variable to get the current option in the select list

var selectedTable = $('#dropdown option:selected').text();

// passes data back to the getTableData function

getTableData(selectedTable);

// clears the text in the search box

$('#search').val('');

// notifies the users if this has been successful using a toastr popup

toastr.success('<b>' + selectedTable + '</b>' + ' table has been refreshed');

});

}

// function to build the initial select list

// Todo, this needs renaming

function buildSelectList() {

// GET HTTP request to the controller->action getTables

$.getJSON("controller.php?action=getTables", function(result) {

// if there is no error

if (!error(result)) {

// generate a success message in a toastr success popup

toastr.success('<b>Status ' + result.status + ": </b>" + result.numOfRows + " tables listed successfully");

// store the selected object in a variable

var select = $('#dropdown');

// iterate though the dropdown

$.each(result.data.tables, function(key, val) {

// add each option to the select list

$(select)

.append($("<option></option>")

.attr("value", val)

.text(val));

});

}

else {

// displays error if failed

displayError(result);

}

});

}

// function to build the select list for the columns in chosen table

function buildColumnSelectList(columns) {

// selector for the select list

$("#columns")

.find('option')

.remove()

.end();

// iterate through the columns and generate the required markup

$.each(columns, function(index, value){

$('#columns').append($("<option></option>")

.attr("value",value)

.text(value));

});

}

// gets data and builds the table

function getTableData(table) {

// HTTP get request passing the table name into the action

$.getJSON(url + table, function(result){

// if there is no error

if(!error(result)) {

// check if there is data returned

// delete all previous markup (the page is truely dynamic)

clearHTML();

// checks the number of rows from the JSONSchema class

if(result.numOfRows > 0) {

// start building the table html

html = '<table id="returnedData" class="table table-striped table-bordered"> <thead> <tr>';

// init empty array for columns

var columnArray = [];

// loop through each column in the row and build table header

$.each(result.data[0], function(column) {

// markup for table head

html = html + '<th>' + column + '</th>';

// push the column into the array

columnArray.push(column);

});

// builds the select dropdown for the columns by passing in the columnArray as a parameter

buildColumnSelectList(columnArray);

// end table header

html = html + '</tr> </thead>';

// loop through each row in the table

$.each(result.data, function(index, row){

// build a new row for each index

html = html + '<tr>';

// loop through each column in row

$.each(row, function(item, value){

html = html + '<td>' + value + '</td>';

});

// end the row

html = html + '</tr>';

});

// close the table

html = html + '</table>';

$('#tableSection .container .panel-body').after(html);

}

else if (result.numOfRows < 1) {

$.getJSON('controller.php?action=getColumnsInTable&table=' + table, function(result){

html = '<table id="returnedData" class="table table-striped table-bordered"> <thead> <tr>';

var columnArray = [];

$.each(result.data.columns, function(index, colName) {

html = html + '<th>' + colName + '</th>';

columnArray.push(colName);

});

buildColumnSelectList(columnArray);

html = html + '</tr></thread>';

html = html + '<tbody><tr><td></td></tr></tbody>'

html = html + '</table>';

$('#tableSection .container .panel-body').after(html);

});

}

}

else {

clearHTML();

toastr.info('No rows in table');

}

});

}

// function to dymanically search through table

function searchTable(table, column, value) {

// log the url builder into the console for dev pupropses

console.log("controller.php?action=searchTable&table=" + table + "&column=" + column + "&value=" + value);

// passes the built URL (controll->action) into the GET HTTP request

$.getJSON("controller.php?action=searchTable&table=" + table + "&column=" + column + "&value=" + value, function(result){

if(!error(result)) {

// start building the table html

html = '<table id="returnedData" class="table table-striped table-bordered "> <thead> <tr>';

// loop through each column in the row and build table header

$.each(result.data[0], function(column, columnsInRow) {

html = html + '<th>' + column + '</th>';

});

// end table header

html = html + '</tr> </thead>';

html = html + '<tbody>';

// loop through each row in the table

$.each(result.data, function(index, row){

// build a new row for each index

html = html + '<tr>';

// loop through each column in row

$.each(row, function(item, value){

html = html + '<td>' + value + '</td>';

});

// end the row

html = html + '</tr>';

});

// close the table

html = html + '</tbody></table>';

$('#tableSection .container .panel-body').after(html);

}

});

}

// add dynamic row for user to input data

function newRow() {

// get the number of columns in current table

var numberOfColumns = $('#columns > option').length;

var html = '<tr class="newDataRow info"><td><button id="saveRow" class="btn btn-success btn-sm"><span class="glyphicon glyphicon-ok"></span></button> <br /> <br /> <button id="cancel" class="btn btn-danger btn-sm"><span class="glyphicon glyphicon-remove"></span></button></td>';

// first column will not need an input box as the field is auto incremented

$('#returnedData tbody tr:first').before(html);

// loop to add additional columns to capture data started at the next point

for (var i = 1; i < numberOfColumns; i++) {

var currentColumn = $('#columns option').eq(i).val();

$('#returnedData tbody .newDataRow').append('<td style="vertical-align:middle"><input attribute="' + currentColumn + '"placeholder="' + currentColumn + '"type="text" class="form-control" /></td>');

}

}

// function to save data entered by user

function saveUserInput() {

// build URL (controller->action) + (tablename selected in dropdown select list)

var saveQuery = 'controller.php?action=addRow&table=' + $('#dropdown :selected').val();

// itterate through each text box assigning it the correct attribute

$('.newDataRow input[type="text"]').each(function(){

saveQuery += "&" + $(this).attr('attribute') + "=" + $(this).val();

});

// generate a log to see the URL builder. This would be removed after production

console.log(saveQuery);

// jQuery's ajax function

$.ajax({

// passes the saveQuery variable to the url parameter

url: saveQuery,

// ajax success method

success: function() {

// success message for user in toastr popup box

toastr.success('Record added to table');

},

// ajax error method

error: function() {

// error message for user in toastr popup box

toastr.error('Unable to save data to database');

}

});

}

// when a variable changes in the search box (defined by its jQuery select #search)

$('#search').on('input', function(){

// deletes the table contents (to stop duplicated markup)

clearHTML();

// gets value of text in dropdown, value of text in columns drop down, and gets the value of text in the searchbox

// and passes these back as parameters to the searchTable function

searchTable($('#dropdown :selected').val(),$('#columns :selected').val(),$(this).val());

});

// detects a click event on the add row button

$('#addRow').click(function(){

// calls the method to add a new row

newRow();

});

// document on click to bind event handler to dynamically created button

$(document).on("click", "#saveRow", function(){

saveUserInput();

});

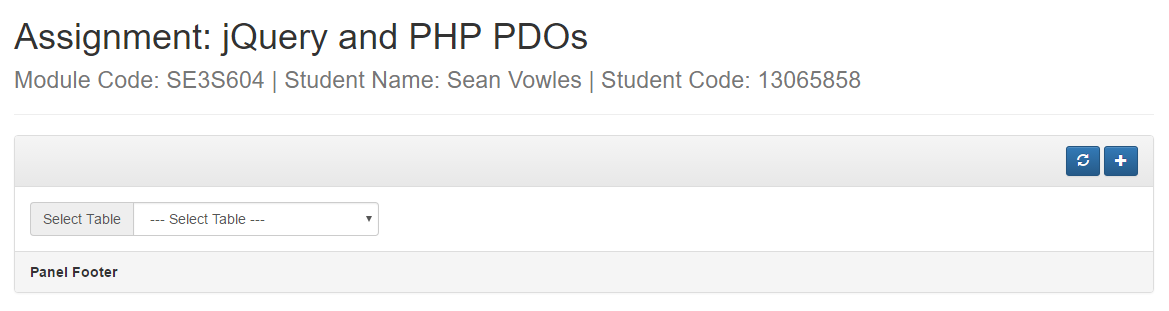
# Prohibiting possible SQL Injection

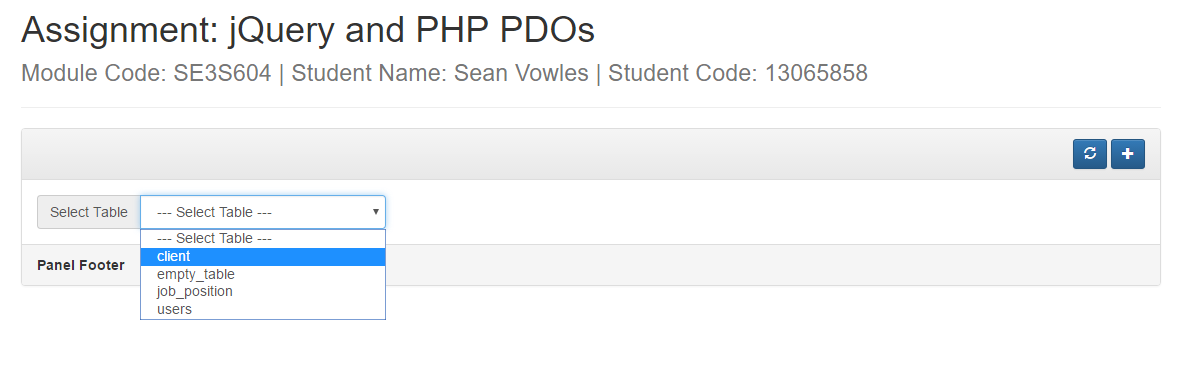
SELECT \* FROM $table WHERE $column LIKE :value

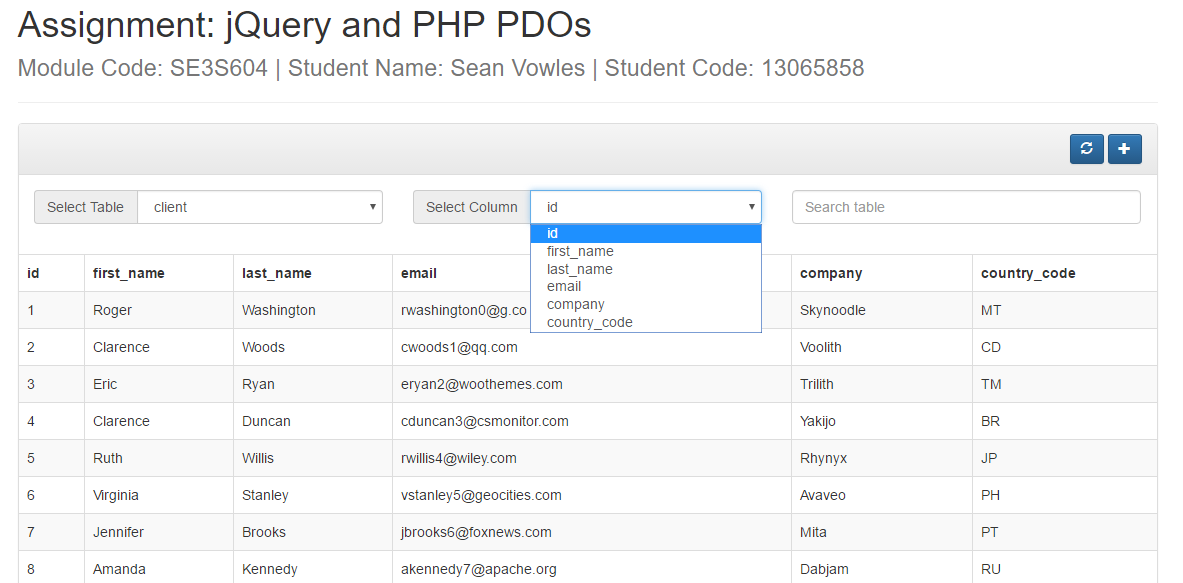
This will build a statement to select all data from a specified table where the column name is a value that cannot be SQL injected through the client.

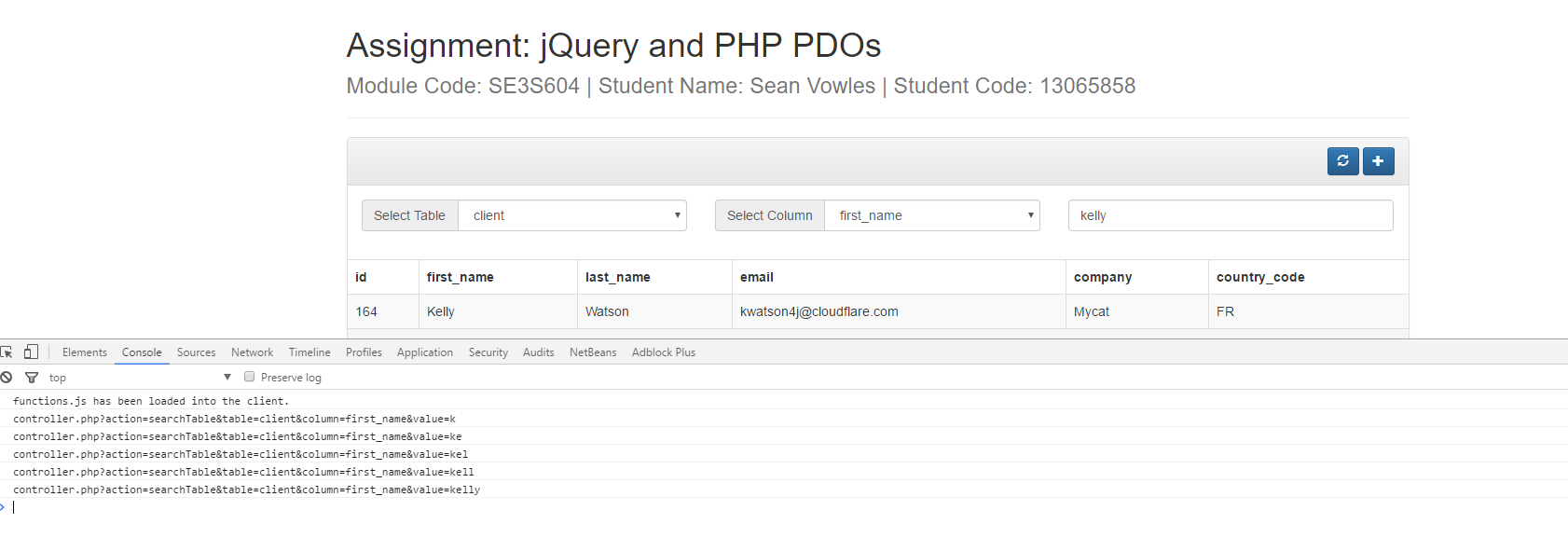
# Testing in Different browsers

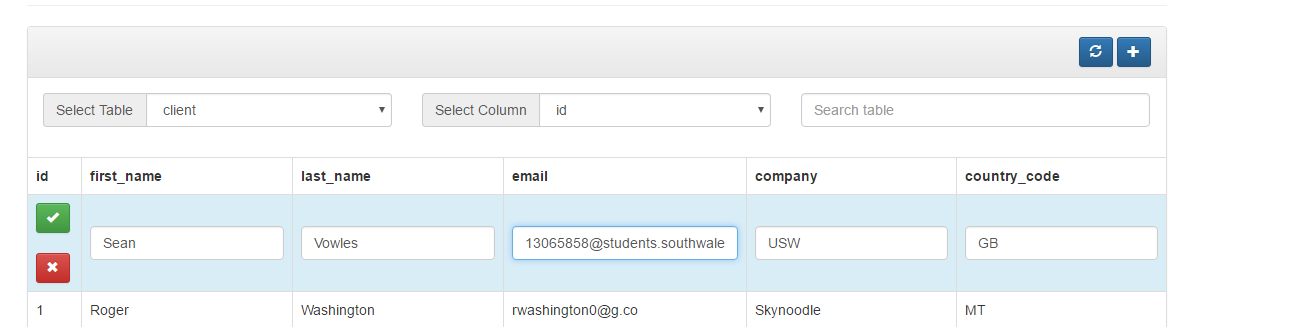
## Google Chrome

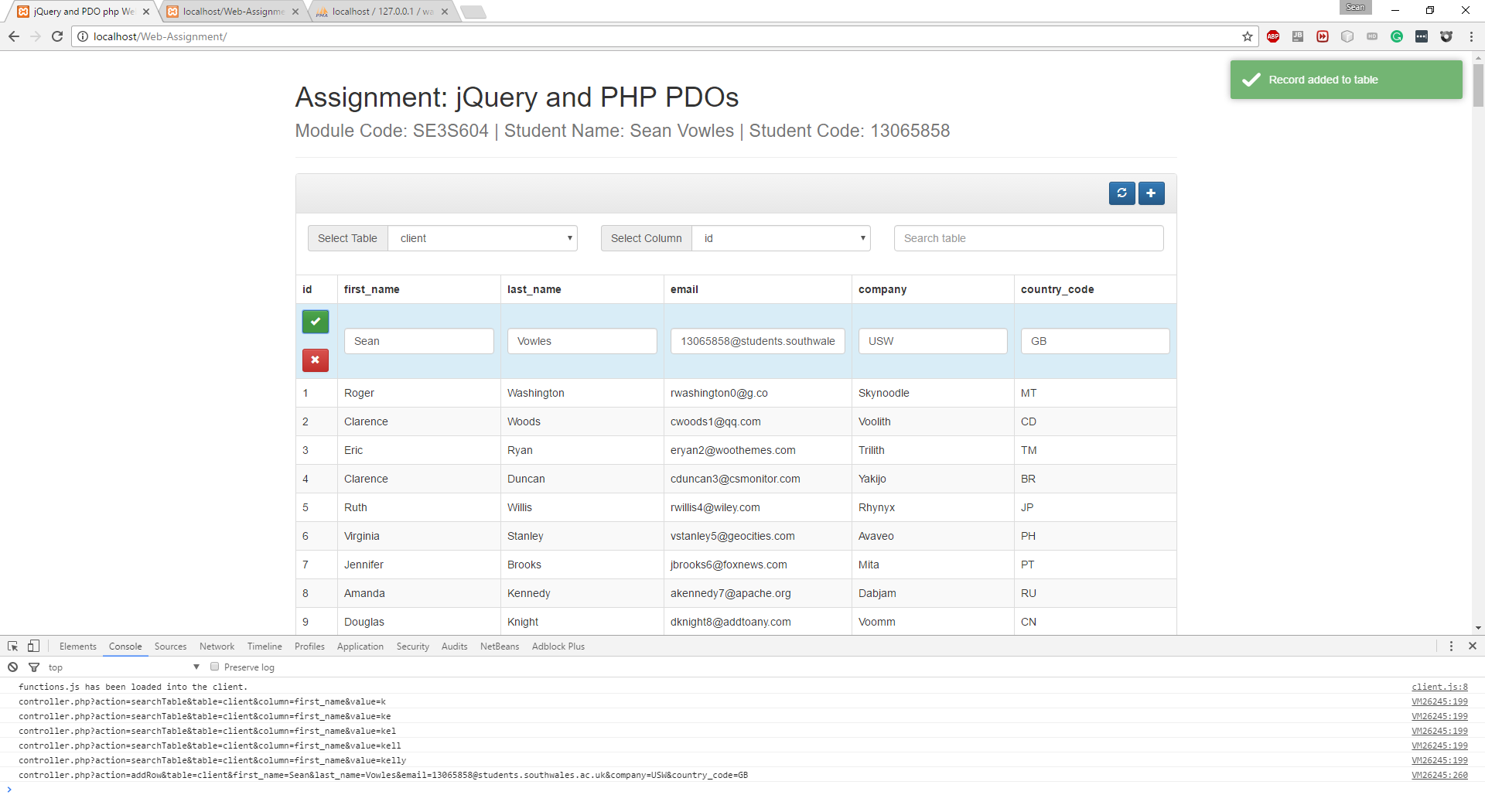


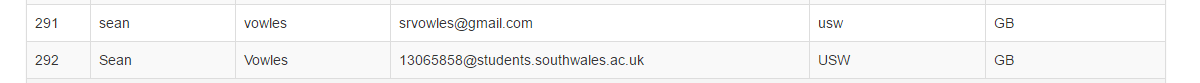




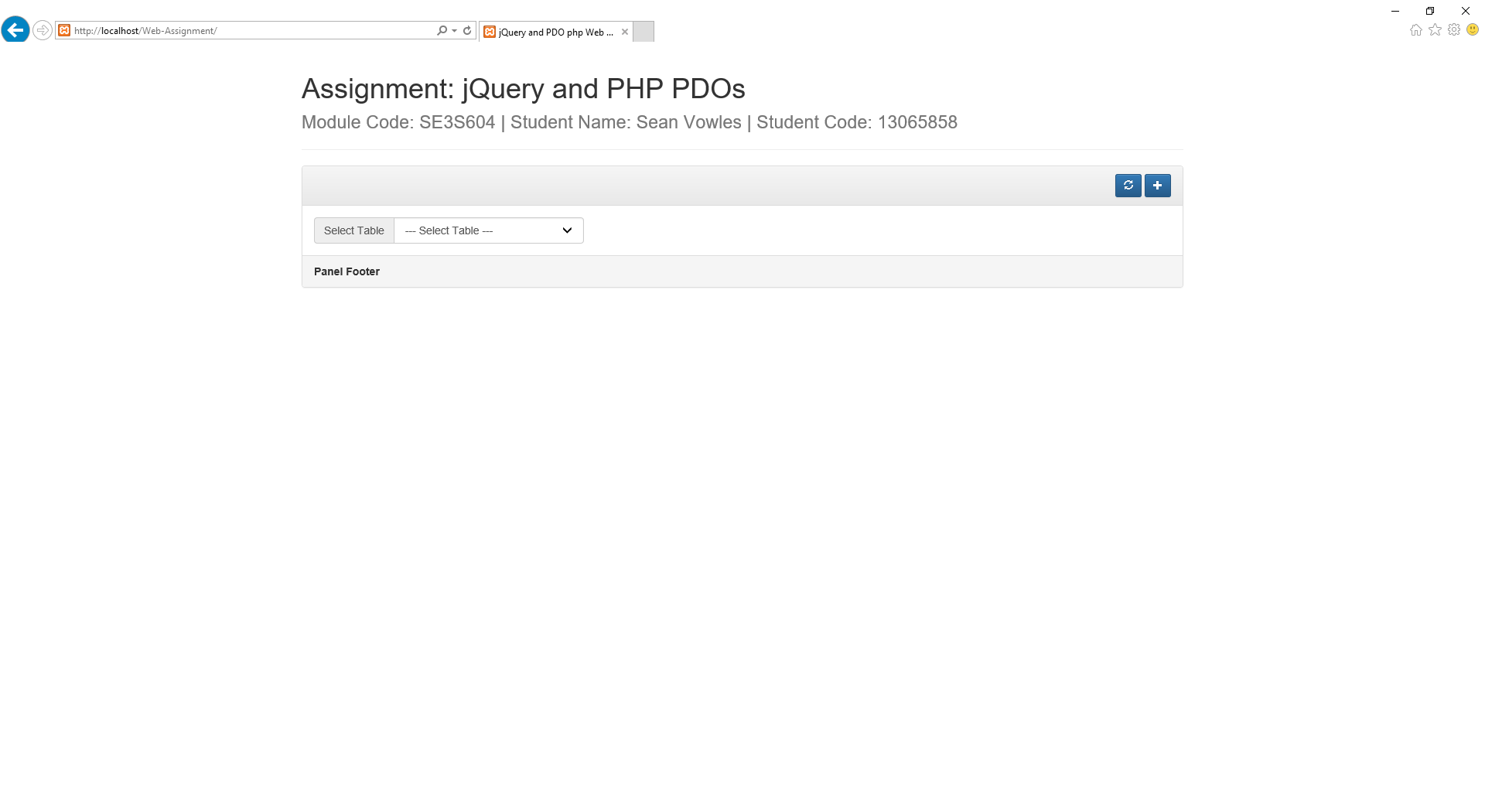


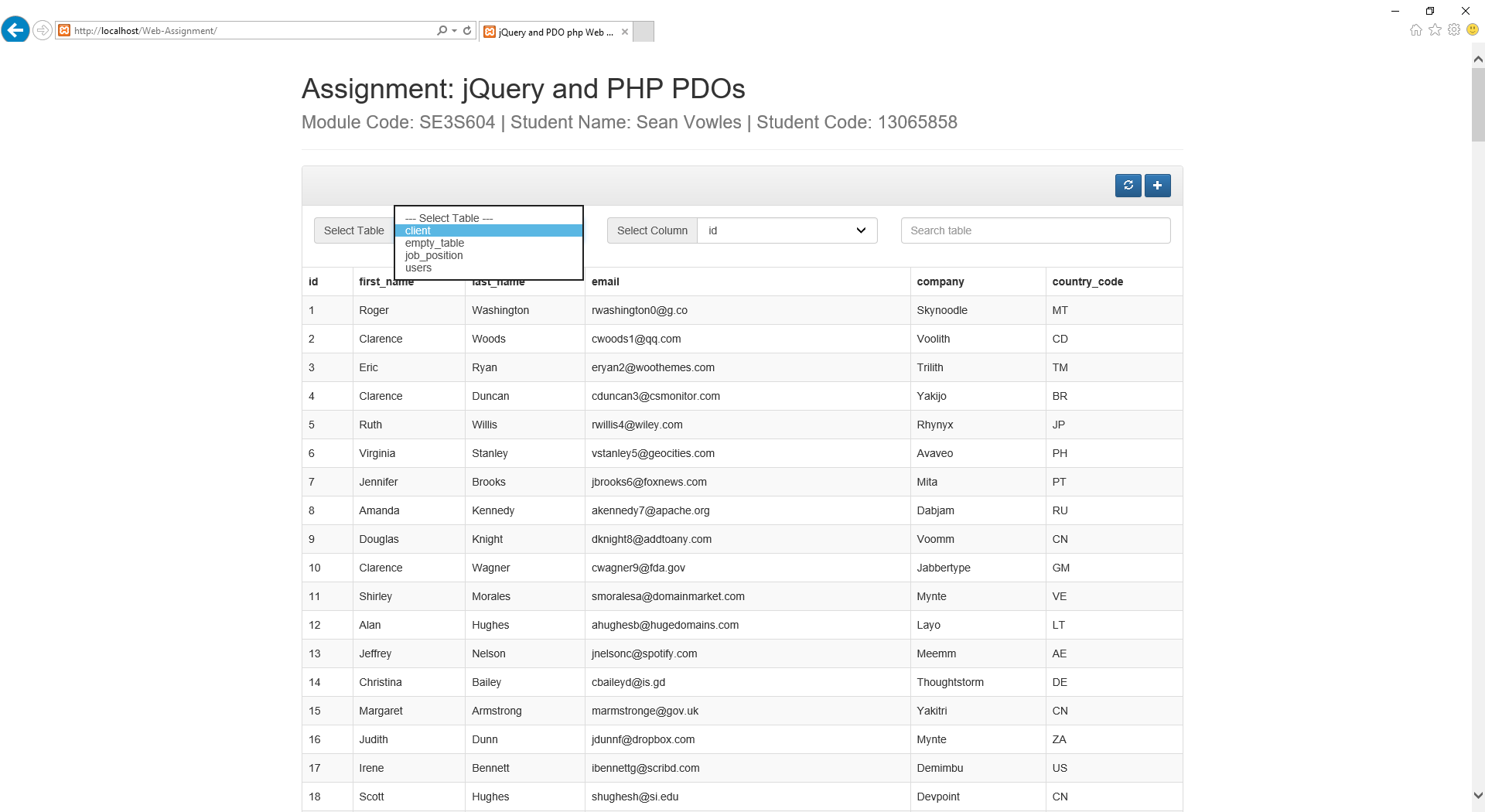


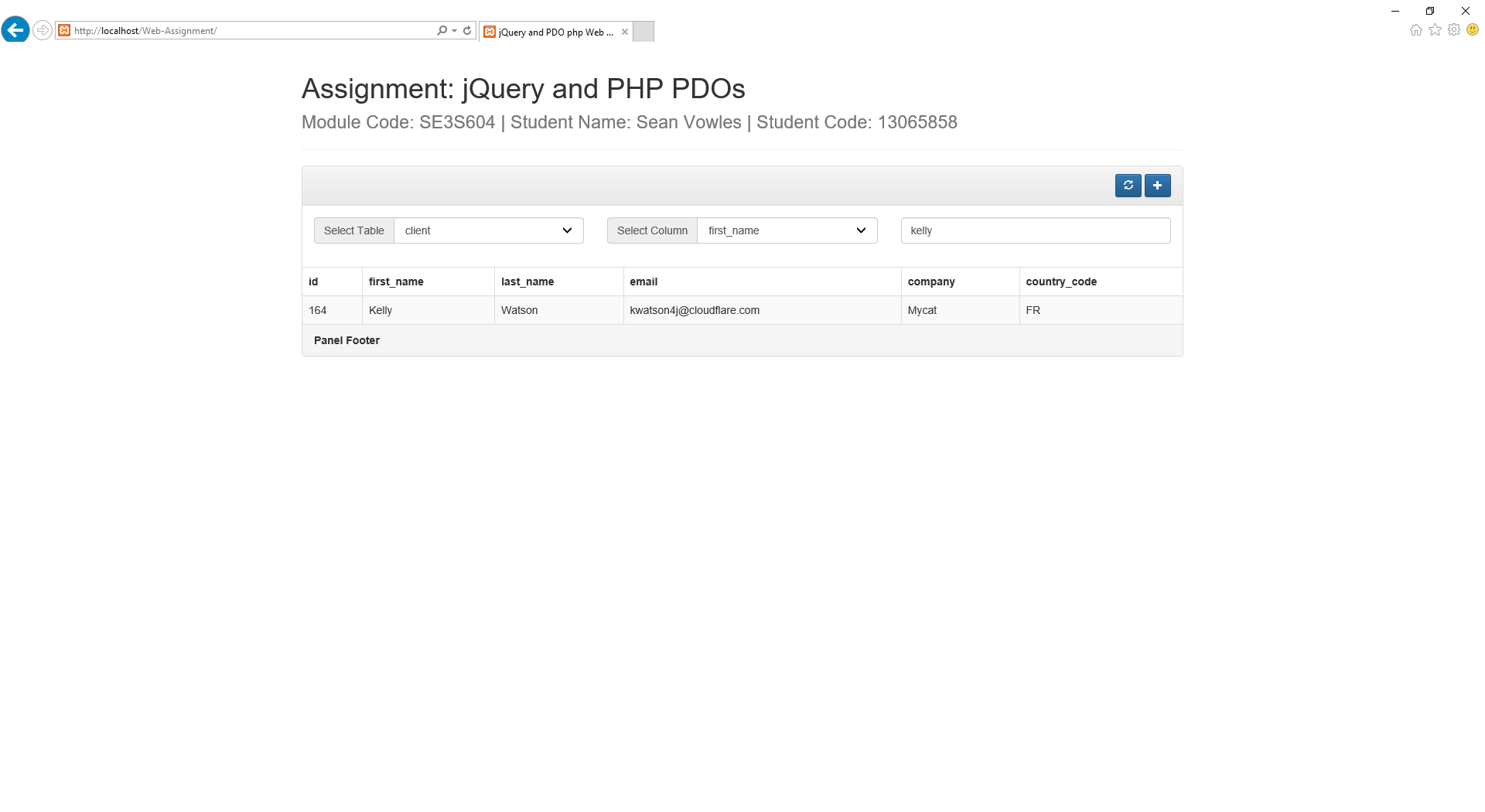
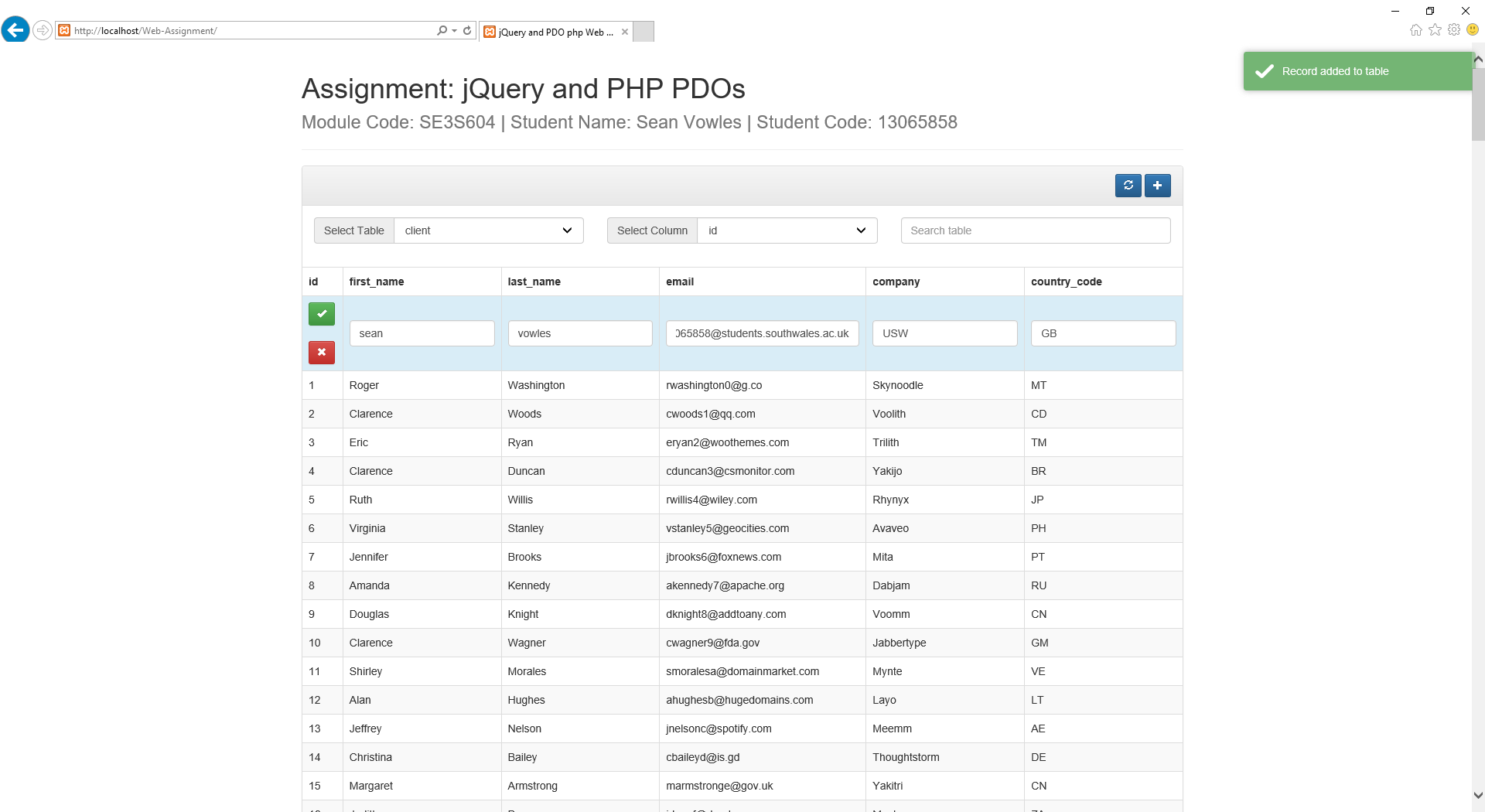


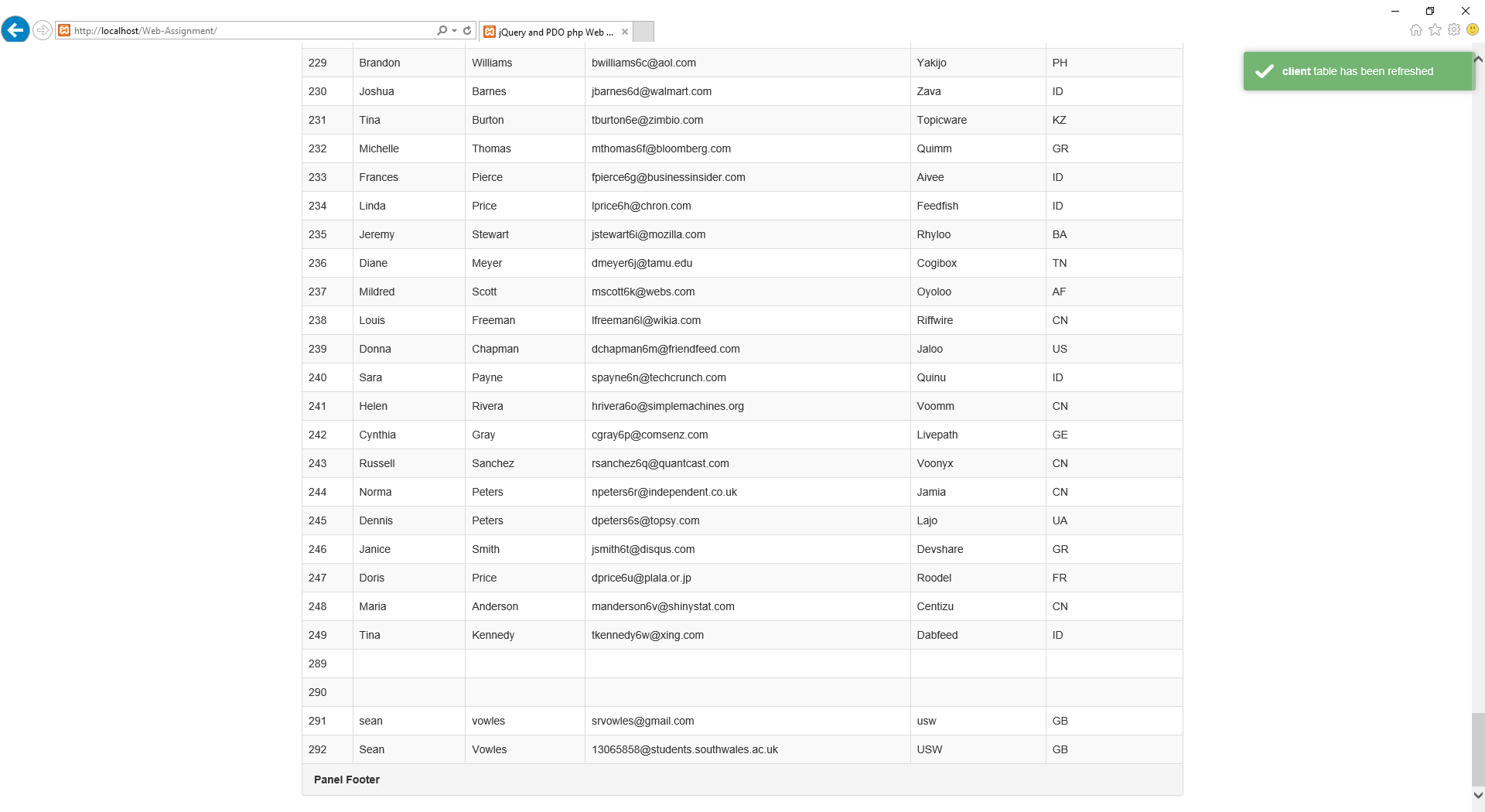


## Internet Explorer







## Microsoft Edge

