jQuery and AJAX | Assignment

Introduction

For this assessment, you are going to use jQuery and AJAX to make a dynamic web page that communicates with two server-side scripts that we have developed for you.

- The ***first script*** uses data from the Food Standards Agency. The Food Standards Agency makes the results of restaurants' (and other businesses serving food) **hygiene inspections** publicly available (see here: https://ratings.food.gov.uk/), and we have developed a simple PHP script that will provide the results of the inspections for Maidstone in JSON format.
- The ***second script*** is uses data from Google Places. Google Places provides (among other data) **customer ratings** for restaurants and other businesses. We have developed a PHP script that will provide the customer ratings for some businesses in Maidstone in JSON format.

Requirements & constraints: You should contain all of your work on this assignment in a *single* HTML file. Use CSS embedded in the page's header for formatting, likewise for your JavaScript code. You must use the jQuery library (i.e., avoid plain JavaScript where jQuery can be used) and, for Part 3, the jQueryUI library (and its CSS if you wish). You cannot use any other jQuery plugins, external CSS, or any other JavaScript libraries. You should submit your single HTML file via Moodle before the deadline.

Part 1: Hygiene Ratings

Part 1 focuses on the hygiene ratings. The main challenges are: setting up AJAX calls and pagination.

The Server-Side Script

The server-side script is hosted at the following URL:

https://www.cs.kent.ac.uk/people/staff/yh/co539_a2_data/hygiene.php

The script can be manipulated using several GET parameters. Accessing the script without any parameters returns the first 10 businesses' ratings in JSON format for testing. The parameters accepted by the script are documented in the table below.

Parameter Name	Description
op	This is the most important parameter, controlling what type of information the script is to return. The op parameter can be set to one of the following values: `demo` Returns the first 10 businesses' ratings (default operation) `get` Returns a "page" of (20) results. Expects the ***page*** parameter `pages` Returns how many pages of results are available `search` Returns the first 20 records for businesses whose name contain the specified search term. Expects the ***`business`*** parameter.
page	Used only by the `get` operation. Expects an integer specifying which page of results to retrieve. For example, page=2 would return the 2nd page of results.
`business`	Used only by the `search` operation. Expects a string specifying the search term business names are to be matched against.

Examples:

- To retrieve the results on page 7:
- https://www.cs.kent.ac.uk/people/staff/yh/co539_a2_data/hygiene.php?op=get&page=7

- To find the businesses whose names contain the string "Eat":
- https://www.cs.kent.ac.uk/people/staff/yh/co539_a2_data/hygiene.php?op=search&business=Eat

JSON Format of Returned Data

The ***demo***, ***get***, and ***search*** operations all return data in the same JSON format. The format is an array of individual JavaScript objects, with each object having the following properties:

`business`	The name of the business inspected
`address`	The full address of the business
`type`	The type of the business, e.g. Restaurant/Cafe/Canteen
`rating`	The rating (out of 5) awarded to the business
`date`	The date on which the inspection was carried out

The ***pages*** operation returns a single JSON object, with a single "pages" property. This property contains the number of pages of results that are available.

Accessing the Script

The script outputs special HTTP headers, called Cross-Origin Resource Sharing (CORS, https://enablecors.org/), which will allow your browser to access the script using AJAX requests regardless of where your HTML file is, in spite of the same origin policy. This means that you can complete this work from home without needing a webserver or database. We have hosted the script on a publicly-accessible webserver, so you don't need to use the VPN either. Because of the way these headers are interpreted by your browser, you **must** use the form of the URL as given above. Leaving out the *`www.`* will cause the CORS to fail.

Task 1.1 – Retrieving the First Page of Results When the Page Loads (15 marks)

Write an HTML web page containing a title, some text explaining what the page does, and an empty table. Use jQuery and an AJAX request (you *must* use the jQuery function `$.get` to do this) to the server-side script to populate the table with the *first page* (page 1) of the results when your web page is loaded in a browser. It should display each business's name, address, type, the hygiene rating and the date when the inspection was carried out.

Task 1.2 – A Basic Paginator (15 marks)

Your page should now perform an additional AJAX request when the page loads (you *must* use the jQuery function `$.get` to do this). This request should find out the total number of pages of the inspection results that are available from the server-side script.

Once you have found this out, create a row of buttons, one for each page. Each button should be labelled with a page number. Clicking one of the page buttons should empty the results table, perform an AJAX request to the server-side script to fetch the results of the requested page, and display them in the table.

Part 2: Customer ratings – mashup

Part 2 focuses on merging information from more than one source: the script you used in Part 1 and another script we provide, with information on customer ratings. As before, the script outputs CORS headers, so you can develop your solution from home.

The Server-Side Script

The server-side script is hosted at the following URL:

https://www.cs.kent.ac.uk/people/staff/yh/co539_a2_data/rating.php

The script can be manipulated using several GET parameters, appended on to the end of the URL. The parameters accepted by the script are documented in the table below.

Parameter Name	Description
businessName	Expects a string specifying the search term business names are to be matched against. Returns all matching records for businesses whose name contains the specified search term.

Example:

- To find businesses whose names contain the string "kitchen":
  https://www.cs.kent.ac.uk/people/staff/yh/co539_a2_data/rating.php?businessName=kitchen

JSON Format of Returned Data

The data is returned in JSON format. The format is an array of individual JavaScript objects, with each object having the following properties:

`businessName:` the name of the business matched
`address:` the address of the business
`avgRating:` the average customer rating (out of 5)
`totalRatings` the total number of customer ratings

If no business matched the given name the script returns an empty array. Note that for some businesses there may not be an address or rating available (in which case key is associated to `null`).

Task 2.1 – Retrieving the Customer Rating when Requested by the User (20 marks)

Extend each row of the table with a new button that allows the user to retrieve the customers' ratings for the business in that row. Each button must trigger an AJAX request (you *must* use the jQuery function `$.get` to do this) directed at the server-side script `rating.php`. On clicking the 'Get rating' button for a selected business it should display in a pop up window the average customer rating and the total number of ratings for the business.

To help you testing your web application, you can view the complete data available on `rating.php`:

https://www.cs.kent.ac.uk/people/staff/yh/co539_a2_data/rating.php?html

Please notes

- The 'Get rating' buttons must be visible in any tables populated using the operations ***demo***, ***page,*** ***search***, or name search functionality from Part 1.
- The script `rating.php` may return more than one matching business. You need to find a way to match the business you are looking for with one (the right one!) of those returned by `rating.php`. You can, for example, try to match the address.
- If the customer rating is not available (e.g. `rating.php` returns an empty array when no matches are found, or the matching business has no rating) then you must notify the user using a pop up window.

Part 3: Search Functionality with Autocomplete

In this part you will use your knowledge of jQuery and AJAX, together with the available documentation of the jQueryUI API (https://jqueryui.com/autocomplete/), to learn to use a function that you may have not seen before: the autocomplete function of jQueryUI.

Task 3.1 – Add a search input with Autocomplete (15 marks)

You need to add a form at the bottom of your web page containing a text input and a search button. e.g.

When the user enters text in the text input and clicks the search button, your page should clear the results table, perform an AJAX request (you *must* use the jQuery function `$.get` to do this) to the server-side script for Part 1:

https://www.cs.kent.ac.uk/people/staff/yh/co539_a2_data/hygiene.php

After doing this, you need to apply the autocomplete function provided by jQueryUI to the input field. The autocomplete function will initially have a pre-defined array of *autocomplete tags*, which you have to define (initially, create an array containing *at most four* names of businesses at your choice). The autocomplete function should now suggest the four default names when focusing on the input field.

Next, you need to let the array of autocomplete tags be extended with previous search information. Consequently, every time the user clicks the search button, the names of the businesses returned by the script `hygiene.php` should be added to the array of autocomplete tags, avoiding duplicates. Recall that you need to import the jQueryUI libraries before using them (see URL below):
https://ajax.googleapis.com/ajax/libs/jqueryui/1.12.1/jquery-ui.min.js

Also, you may want to use the jQueryUI styles (see URL below):
https://code.jquery.com/ui/1.12.1/themes/smoothness/jquery-ui.css

General Correctness (15 marks)

There are a significant number of marks allocated for the correctness of the used JavaScript constructs. This refers to carefully following the instructions in this brief, and general correctness, for instance, avoiding any kind of bugs such as race conditions, loading-order and robustness issues.

Design & Code Quality (5 marks)

There are some marks available for the visual design of the page, which you should achieve using CSS. Pay particular attention to the table in terms of size, spacing and margins to maximise usability. You should also make sure that the paginator buttons are a suitable size. You should also ensure that your code is properly commented, organised and laid out.