The webpage URL is: ***https://www.geos.ed.ac.uk/~s1757431/web_mapping.html***
This webpage has the best functionality on *Google Chrome* and *Safari* browsers.

Description of the web mapping project:

A FIELDS-FINDS archeological database is available. An interactive interface to this database is required. To this end, we need python codes to 1) query the Oracle database to retrieve the data in fields and finds tables. 2) do necessary computation and 3) display the results on a web browser.

Solution:

I used 1 html file, 1 java script file, and 7 python files to design a webpage with three different tabs and fill them with relevant information from database.

Web_mapping.html: This html file is the main file, which makes the general view for the each one of the following three tabs, also contains the address of related python files for these three tabs.

The first tab (Chart): shows the "*archeological finds map*". This map has the possibility of zooming in and out, also moving around when zoomed. Moreover, by clicking on each finds the relevant information will pop up in a proper window. By re-clicking that find the window will be closed.[1] Depending on the clicking position, I used if statements to position the windows dynamically to avoid them cutting the page's borders.

The second tab (Fields Table): shows the "*fields table*". This table is a copy of the fields table in the database. Using 'x' and 'add field' buttons on the page, user can remove an existing row or add a new row to the table. Any changes made by user will be saved on the copy version table (named fields2 on the codes) and will be shown by automatically refreshing the page. If user presses the reset button the primary values for the table will be again copied from the field (unchanged table) and the table will contain the default values. Domains for entering each value is set, in case of entering inappropriate input values by user, a message pops out saying the right domain values for the cell.

The third tab (Finds Table): shows the "*finds table*". This table has all the features explained in the second tab, except it is for finds table.

Python files:

Map.py
This file contains the content of the 'Chart' tab.
It consists of four functions:
get_text_tagStr: This function is responsible for creation of any text (string or number) in the grid tab.
get_field_tagStr: This function creates rectangles with the use of 'rect' SVG tag. It draws fields and the container of their id numbers in the middle of each field. get_text_tagStr is also called to fill the container with the corresponding id number.

[1] To make the ability of comparison possible I avoided to set time out for closing the windows, but the code for this purpose is commented in the map.py file.

get_finding_tagStr: It uses SVG circle tag to point the location of the finds on the grid tab. It also calls get_text_tagStr to allocate appropriate id number to each find.
get_legend_tagStr: It uses the rect and circle SVG tag to draw the legend for the map. Also calls get_text_tagStr to enter desired texts to legend section.
The needed input arguments for functions comes from the connection to the database.

Fields.py
It contains the content of fields tab. The get_field_row_table function makes one row of the table, using for loop for every column in the fields table. I connect to the database and by calling the get_field_row_table function for each table row, the whole rows for the html table is created.

Finds.py
It has the same procedure as the Fields.py except for the finds table values.

Credentials.py
In this file, I made a dictionary to save my real username and password as values for 'user' and 'password' keys for security purposes. Wherever I needed to login to the database in python codes, I imported the credentials library and used the values of 'user' and 'password' instead of the real ones. I also changed the mode for this file not to be accessible for others and groups. I did not include this file in my prints.

Delete.py
In this file I made two variables (fid, tab). Based on which tab(le) we are in (Finds or Fields), a database query is used to delete the row with the specific field/find id (fid). This python file will be called by ui.js file, when the user presses the 'x' button at the first of each row.

Insert.py
This file gets the new values from the user by post method. It uses insert query to put a new row with the corresponding values into the database, then reloads the page and shows the new row. The unique id is allocated to the row by finding the maximum of id(s) and increasing its value by one. This File is called by the ui.js file, when the 'add value' button in Finds or Fields tab is pressed.

Reset.py
This file does the automatic refresh of the webpage, when it is called in fields.py or finds.py. It connects to the database and deletes the content of copied version of fields or finds tables (fields2 or finds2) and inserts the default value amounts from untouched tables into copy ones.

Ui.js
It Contains java scripts, and as it was mentioned before, its functions are used in different python files.

How to improve to a full-blown web mapping system:

For future improvement, each user could have a username and password for logging into the system; this helps us with the controlling of the level of accessibility for each user. A real topographical map for the background of the Grid tab could be used. To prevent loading massive volume of data a default map (limit in the extent of the x and y coordinates) for each user could be allocated based on the user's interest and business status. The zooming and spanning ability could be improved, with consideration to load the neighborhood data on the map when spanning is stopped to prevent delays. The shape of the fields, which are assumed rectangles or squares,

could include more shapes like hexagonal (for specific needs). To make my webpage more user friendly I could allow users to drag two points to the table showing the diameter of the fields (instead of entering the coordinates manually). A full blown web mapping system should take the coordinates in one or more well-known coordinate systems and check if the inserted fields have overlap with other fields or not. More related tables from database could be joined to give users more information. Not every user should be able to add to the database or remove from it. Guidance notes should be added to help users working with the web page. More meticulous constraints and domains should be applied to the entering fields. The codes could be object oriented with classes for fields, finds, users, maps, tables, etc. Photos could be used to makes the webpage more colorful, and users could drag (for example) crops' image to the map instead of entering their names in the table. The pop up window on the map could have a close button on its self.