**Multiple Form Inputs:**

In the quiz input form, I faced this problem. Here the admin can enter Multiple quizzes without returning to the admin index.

The problems,

* After entering a quiz or returning to admin index page he can go back and enter the same quiz again resulting in same quiz for 2 or more days.

The solution,

The qid in quiz table is AUTO\_INCREMENT, so we need to enter the quiz only, qid is generated automatically.

Now I try to achieve 2 things,

1. Whenever a new quiz input form opens the input fields should be empty.
2. Refreshing the quiz input page must not resubmit the form again.

To achieve 1. I need to redirect once to the same page each time the form opens so that we get a clean form each time we open it.

To achieve 2. I need to redirect to the same page each time the form is submitted so that the form that is loaded after successful submission is not the form that processed the form input, rather it's a new form.

So, we structure of our quiz input form is as follows,

(Start)

|

V

[Session validation]

|

V

[Form input processing and database interaction]

|

V

[Redirect Once]

|

V

[HTML form input elements (front end)]

|

V

(End)

/\*----------------------- Redirect once code -----------------------\*/

if(!isset($\_SESSION['redirect\_once']))

{

$\_SESSION['redirect\_once'] = true;

header("location: quiz input.php");

}

else

{

unset($\_SESSION['redirect\_once']);

}

This structure ensures Whenever the quiz input page is loaded, we get a brand-new input form.

At the very end, I also provided a delete last quiz button and an update quiz button in quiz display page.

**Primary session security:**

While creating session we hash the combination of client IP address and client's browser user agent string and store it into the session which is checked while entering the session to prevent session hijacking.

Each time we enter a session we change session id to prevent session fixation.

**Security Concerns:**

Admin area must not be accessible from outside, only way to enter there should be through successful admin login.

Steps taken to ensure this,

* Home and Admin login page destroys any existing session.
* While creating admin session we create a unique session element, to indicate that it's admin session.
* All admin pages will check this element, if not found opens home page.

**Form input:**

* Here we have several forms for registration, login, quiz input and update etc.
  + Input entered in those forms will be sanitized and trimmed before adding them to SQL query to prevent SQL or HTML injection (sanitization).
  + Passwords are hashed and the hash codes are stored in the database so they are not sanitize.
  + Emails are checked for syntax and valid domain.
  + Apart from User or Admin name and password case of the input characters is not important.

**Form resubmission warning:**

“Confirm form resubmission" is shown when we visit a page that required you to submit a web form to visit.

If we get this warning while reloading a form that means,

* The file containing the form processes the form inputs, i.e., the inputs given to the form is sent to the same file upon submission.
* We have submitted this form several times and got the outputs in the same page (above or below the form), hence the contents of the form page are dependent on previous submission of the same form.

When such webpages are reloaded the previous form inputs are reprocessed which may cause some updates in the server (like, re-entering same record twice in database); that’s why the warning is shown, we wouldn't want to accidentally update something twice. If you know for sure that nothing will get updated, it's OK to resubmit the form.

Solution,  
In the program, we have made the forms so that, this warning is never displayed.

* The trick is to redirect to the target webpage (the form itself, if the same page processes form inputs) and send GET variables, while redirecting, to indicate success or failure or to display any other message. This makes the browser to load a new copy of the target webpage (with all necessary output messages) rather than the one which received the form outputs.

Thus, when such a target page is reloaded no form input is accessed or processed and we don’t get any warning.

**How to load the Pokémon names list faster for input forms:**

I was asked to provide the list containing names of all the Pokémons in the forms where user needs to input a Pokémon name, so that the user can select the proper Pokémon name.

Problem:

* My initial approach was to load them directly from poke API into those forms but this makes loading time of those forms longer.

Solution:

* Now I plan to keep this list in the server and attach it to the forms.
* When Index or Home is opened (or generated) it checks if the list of Names of the Pokémon are present in the server, if not it loads it. (the function to load the names was written by Bard)
* This list is in the form of a HTML <datalist> that can be attached to an input tag to get input suggestion.
* This list is stored in a .php file as simple HTML script, and we use “include” of PHP to add it to the proper place in the form as needed.

**Update and delete anomaly in quiz display:**

Previously I did update and delete based on the qid provided. But this has a problem any one can change the code of quiz display page and delete unwanted things.

Solution:

* Updated the quiz delete code so that it always deletes the last quiz regardless the qid.
* Updated the quiz update so that it doesn’t accept update request if qid is not >= current qid.

**How I got the nice shadow:**

Previously I wanted to give a nice soft shadow effect to the forms. Hence, I added this box-shadow to the form element in “form styles.css” file, which is shared by all forms.

Now I faced 2 problems,

* In some pages I was using a list of forms, there this shadow effect doesn’t look good so, I had to make shadow none and apply new shadow styles.
* Some non-form elements would look good with this shadow effect, but I don’t want to write the same effect multiple times for multiple elements, hence, I couldn’t apply this effect on them.

The solution,

* I removed the form element totally from all the CSS files.
* Made a new class “nice\_shadow” that contains the code for the nice box-shadow effect and this property is also made important so that, any element that uses this class will get this effect.
* Put the CSS code for this class into “general styles.css” that all the webpages include.

Now I just add this class to any element to get this nice shadow effect behind it.

**Stopping unwanted scroll and Shrinking “main\_box” in index:**

I used the following structure

<body>

<header> … </header>

<main>

<ul class = “main\_box”> … </ul>

</main>

<footer> … </footer>

</body>

“main” element here is a flex box with “align-items: *center*; justify-content: *center*;” to keep “ul” in the center, flex direction is row.

* **When the width of the elements were smaller than the “main\_box”, it started to shrink.** Previously I set the width of “main\_box” (a class generally used with “ul” to create main box of the form) as “max-width: 27em;” in “form styles.css” file. My aim was to get the box to “27em” width and shrink it when the screen is narrower, and set width elements (flex-items as “main-box” is a flex box with flex direction column) inside the “main\_box” to 100% so that they spread horizontally all the way across the “main-box” element. But I faced this problem.

**Solution**, I added a new property to “main-box”, “width: 50em”, this tried to increase the width of “main-box” beyond previously set max width “27em” but the width remained constant at “27em” though it could shrink below “27em” if necessary and the elements inside spread horizontally all the way across the “main-box” element.

* **I failed to set the height of main tag properly it was shooting out of body**, I tried the following and failed to bound it within body,
  + Set the height of body to 100%.
  + Adjust padding and margin of main and body elements.
  + Make the body flex and then set size of header and main.

**The solution**, I set the height of main to "min-height: 90dvh;" if header is not present.

If header is present height of header is "min-height: 3.5em;" and height of main is "min-height: calc(90dvh - 3.5em)".

**Destroying session in login pages (failure of session destroy function):**

According to my plan I destroyed any existing session while entering a login page. The function I used for this clears the session variables, destroys the cookie containing session name and destroys the session.

The problem,

* I found this function quite unreliable as after 1 or more failed login attempts, if the user logs in successfully and then goes back to the login page, this function fails to do anything.

Solution,

* I removed this function, instead I started or resumed current session at the beginning of the login files and checked if there are any session variables. If any session variables found I redirect to index or home page that destroys the session once and for all.

For now, this works.

**How we decide which quiz to display on which day:**

* We keep a starting date (date of the day on which our first quiz will be aired) in a PHP file.
* This file also calculates the no. of days between the “start date” and “current date”.
* In quiz each quiz has a “qid” assigned to it, which starts from 1 (quiz with qid 1 is displayed on “start date”).
* The qid of the current quiz = [no. of days between the “start date” and “current date” + 1].
* Advantages of this approach,
  + We don’t need to store dates in the database, saving some space (a date as string takes 8 bytes but smallint takes 2bytes).

**Disadvantages and Advantages of using Auto Increment:**

If a field is set to AUTO INCREMENT its value starts with 1 and increases by 1, i.e., while inserting new records into this table we don’t have to provide any value it will be added automatically and the first record will have the value as 1 and the value for every subsequent record will increase by 1.

**Advantage:**

Usually, key fields are set to AUTO INCREMENT so that, while inserting records into the table we don’t have to worry about providing unique key to each record.

**Disadvantage:**

If we delete the last record in the table; the value of AUTO INCREMENT doesn’t update automatically. Hence, if we append record after deleting the last one the value automatically assigned to its AUTO INCREMENT field will not be incremented value of previous tuple. To fix it, after deleting one or more tuples from the end we need to set AUTO INCREMENT value to next expected value i.e., (AUTO INCREMENT field value in last tuple in the table + 1).

**Quiz input anomalies:**

While admin inputs the quizzes, there can be three cases,

1. The current quiz already exists in the quiz table, there are several more quizzes for future. In this case, we append a new quiz to the quiz table and qid gets auto incremented.
2. Start date is in future, in this case, we append a new quiz to the quiz table and qid gets auto incremented.
3. Start date was in past and current quiz doesn’t exist (admin forgot to enter quiz for several days). In this case, we calculate current qid, set it as auto increment value and append a new quiz to the quiz table. Thus, we enter the current quiz, skipping the previous unentered quizzes.

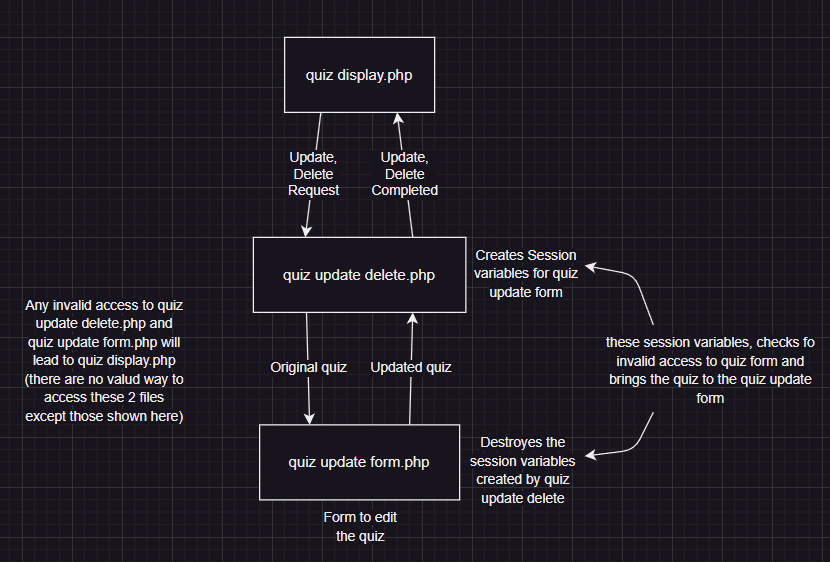
**Auto-commit mode in SQL:**

Auto-commit mode means that when a statement is completed, the method commit is called on that statement automatically. Auto-commit in effect makes every SQL statement a transaction. The commit occurs when the statement completes or the next statement is executed, whichever comes first.

In SQL Auto-commit mode is on by default.

**Detailed Interaction diagrams:**

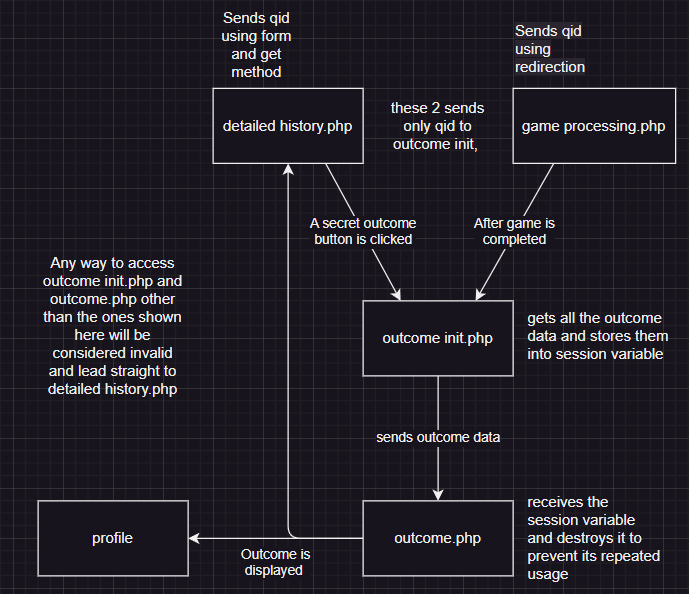
Quiz Display-Update-Delete:



Game Game-Processing:

Any processing regarding the game is done in “game processing.php”, after processing it directs to either “game.php” or “outcome init.php”. Any invalid access attempt is redirected to “game.php”.

Outcome and Outcome-init:



**Flow charts of some pages:**

Quiz input:

Already shown.

Login:

(Start)

|

V

[Destroy any existing session]

|

V

[Process login credentials if they are submitted]

|

V

[Successful login] -> [Start session] -> [Admin or User Index]

|

V

[Failed Login] -> [Same file, with GET variable indicating failure]

|

V

[HTML form input elements (front end)]

|

V

(End)

Register:

(Start)

|

V

[Form submitted]

| |

| true

| |

| V

| [Check email, match passwords] - failure -> [Redirect to same page and send error message via GET]

| |

| V

| [Insert input data into login table]

| |

| V

| [Redirect to same page and send success message via GET]

|

false

|

V

[HTML form input elements (front end)]

|

V

(End)

**What we learned from this project:**

* Difference between GET and POST.
* Sending data to a file via GET while redirecting to a file.
* Session in PHP.
* Cookies in PHP.
* Input sanitization, SQL, HTML injection.
* File handling in PHP “file\_get\_contents()”, “file\_put\_contents()”.
* Auto increment in SQL.
* How to fix form resubmission warning.
* How to clear a form while loading.
* Nice box-shadow in CSS.
* Some tricks about height width adjustment in CSS, which I have mentioned above.
* Where should I look for the list of Pokémon names (<https://pokeapi.co/api/v2/pokemon/?offset=0&limit=1025>) [here we get data about all the Pokémons including their names as jeson file], and images (<https://img.pokemondb.net/artwork/large/pokemon_name.jpg>).