Excellent. Let's break down **Practical 9**.

This practical, "Submit form data using PHP and store it in a text file," is an interesting case for your project. You have actually implemented a **more advanced and professional version** of this already. Your register.php script takes form data and stores it in a MySQL database, which is superior to storing it in a simple text file.

For your practical file, you have two options:

1. **Document your existing, superior method** (saving to the database) and explain that it fulfills the core requirements of this practical in a more robust way.
2. Create a simple, separate script *just for this practical* that saves to a .csv file (as mentioned in the supplementary problem).

I will guide you on how to document **both** so you can choose what fits best for your submission.

Practical 9: Submit Form Data and Store in a File

Aim: To create a PHP backend script to process data from an HTML form, validate it, and store the submitted information in a persistent file format (CSV).

1. Key Questions & Analysis

(This analysis applies to your existing register.php script, as it demonstrates all the core skills required.)

1.1. Is the form submitted using POST?

Yes. The frontend JavaScript (js/register.js) sends the form data using an asynchronous fetch request with method: 'POST'. The backend PHP script (register.php) is designed to only accept POST requests, which is verified using the $_SERVER['REQUEST_METHOD'] === 'POST' condition.

1.2. Is input validated/sanitized?

Yes. The register.php script performs crucial server-side validation and sanitization on all incoming data. This includes:

- Using trim() to remove extraneous whitespace.
- Using filter_var($email, FILTER_VALIDATE_EMAIL) to ensure the email format is correct.
- Checking for empty fields.
- Verifying that the password meets complexity requirements and that the confirmation password matches.

1.3. Is the confirmation message displayed?

Yes. Upon processing the data, the PHP script sends a structured JSON response back to the frontend. This response includes a success boolean and a message string (e.g., "Registration successful!" or "Email already registered."). The frontend JavaScript then uses this message to display a user-friendly notification popup.

2. Implementation: Storing Data in CSV Format (Supplementary Problem)

While your project uses a database, here is the code for a standalone PHP script (save_to_csv.php) that would fulfill the practical's requirement to save data to a CSV file. This demonstrates the file-writing skills asked for.

PHP

```php
<?php
// save_to_csv.php

header('Content-Type: application/json');
$response = ['success' => false, 'message' => ''];

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $first_name = trim($_POST['first_name'] ?? '');
    $last_name = trim($_POST['last_name'] ?? '');
    $email = trim($_POST['email'] ?? '');

    // --- Validation ---
    if (empty($first_name) || empty($last_name) || empty($email)) {
        $response['message'] = 'All fields are required.';
        echo json_encode($response);
        exit;
    }

    // --- Data Preparation ---
    $file_path = 'registrations.csv';
    $new_record = [$first_name, $last_name, $email, date('Y-m-d H:i:s')];

    // --- File Writing ---
    // Open the file in "append" mode
    $file_handle = fopen($file_path, 'a');

    if ($file_handle === false) {
        $response['message'] = 'Error: Could not open the data file.';
    } else {
        // Write the new record as a CSV row
        if (fputcsv($file_handle, $new_record)) {
            $response['success'] = true;
            $response['message'] = 'Data saved successfully!';
        } else {
            $response['message'] = 'Error: Could not write to the data file.';
        }
        // Close the file handle
        fclose($file_handle);
    }
} else {
    $response['message'] = 'Invalid request method.';
}

echo json_encode($response);
?>
```

Explanation:

1. The script receives POST data.
2. It defines a file path (registrations.csv).
3. fopen($file_path, 'a') opens the file for appending. If the file doesn't exist, it will be created.
4. fputcsv($file_handle, $new_record) formats the data array as a CSV line and writes it to the file.
5. fclose($file_handle) closes the file resource.

3. Test Cases (Post-Laboratory Work)

To demonstrate this practical, you would:

1. Temporarily change the fetch URL in js/register.js to point to save_to_csv.php.
2. Submit the registration form.

Test Case 1: Form Submission
[Insert a screenshot of the registration form filled with data.]
Test Case 2: Successful Response
[Insert a screenshot of the Network tab in your browser's developer tools, showing the JSON response {"success":true,"message":"Data saved successfully!"}.]
Test Case 3: Verifying the File Write
[Insert a screenshot of the registrations.csv file opened in a text editor or spreadsheet program, showing the newly added row of data.]
By documenting it this way, you show that you understand the specific file-writing requirement of Practical 9, even though your main project uses a more advanced database implementation.