Excellent, let's move on to the backend. For **Practical 6**, the focus shifts from the frontend to the server. You need to document how your website receives data from the registration form and processes it using PHP. Your register.php script is the star of this section.

Here is a detailed guide on what to write in your practical file for Practical 6.

Practical 6: Store Submitted Registration Data

Aim: To create a backend form processor using PHP to securely receive, validate, and store user registration data submitted from the frontend form into a MySQL database.

1. Key Questions & Analysis

1.1. Are POST/GET methods used correctly?

Yes. The **POST** method is used correctly and securely to handle the form submission.

- **Frontend (js/register.js):** The JavaScript fetch API is configured to send the form data using the POST method. This is the correct approach because it sends the user's sensitive information (like their password) in the body of the HTTP request, rather than exposing it in the URL.
- **Backend (register.php):** The PHP script checks if the request method is POST using $_SERVER['REQUEST_METHOD'] === 'POST'. This ensures that the script only processes data that has been submitted from the form and rejects any other type of request.

Example Code from register.php:

PHP

```php
// From register.php
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $first_name = trim($_POST['first_name'] ?? '');
    $last_name = trim($_POST['last_name'] ?? '');
    $email = trim($_POST['email'] ?? '');
    $password = $_POST['password'] ?? '';
```

```
    // ... rest of the processing
}
```

1.2. How is data stored and displayed?

The data is stored securely in a MySQL database, not a simple PHP file. A success or error message is then displayed back to the user.

- **Database Storage:** Upon successful validation, the user's data is inserted into the users table in the bytebazaar_db database.
- **Data Flow:** The register.php script connects to the database using the credentials in db/db_config.php. It then executes a prepared SQL INSERT statement to add the new user record.
- **Response (Display):** The script does not display a new page. Instead, it sends a JSON response back to the JavaScript that called it. This response contains a success status (true/false) and a message (e.g., "Registration successful!"). The frontend JavaScript is then responsible for showing this message to the user as a notification.

1.3. Are user inputs sanitized?

Yes. The user inputs are properly sanitized and validated on the server-side to ensure data integrity and security.

- **Trimming Whitespace:** The trim() function is used on all incoming text fields to remove any accidental leading or trailing spaces.
- **Server-Side Validation:** The register.php script re-validates all the data that was already checked by JavaScript. This is a crucial security practice, as a malicious user can bypass frontend validation. The script checks for empty fields, valid email format, password length, and matching passwords.
- **Prepared Statements:** SQL injection is prevented by using **prepared statements** ($conn->prepare(...)). This method separates the SQL query from the user's data, ensuring that the data is treated as simple text and cannot be executed as a malicious command.

Example of Prepared Statement from register.php:

```
    PHP


// From register.php

// Hash password for security
$hashed_password = password_hash($password, PASSWORD_DEFAULT);
```

```
// Insert user using a prepared statement to prevent SQL injection
$stmt = $conn->prepare("INSERT INTO users (first_name, last_name, email, password) VALUES (?, ?, ?, ?)");
$stmt->bind_param("ssss", $first_name, $last_name, $email, $hashed_password);

if ($stmt->execute()) {
    $response['success'] = true;
    $response['message'] = 'Registration successful!';
} else {
    $response['message'] = 'Registration failed. Please try again.';
}
$stmt->close();
```

2. XAMPP Test Case (Post-Laboratory Work)

To demonstrate that the backend is working correctly, you should provide screenshots showing the data flow.

Test Case 1: Filling out the Registration Form
[Insert a screenshot of the register.html page with valid user data filled into the form fields.]
Test Case 2: Inspecting the Network Request
[Open the browser's Developer Tools (F12), go to the "Network" tab, and submit the form. Insert a screenshot showing the register.php request with a "200 OK" status and the JSON response {"success":true,"message":"Registration successful!"}.]
Test Case 3: Verifying Data in the Database
[Open phpMyAdmin, navigate to your users table, and take a screenshot showing the new user record that has just been inserted. Make sure to highlight the row with the data you submitted.]
This complete documentation will prove that you have successfully built a working and secure backend form processor, fulfilling all the requirements of Practical 6.