Of course. For **Practical 3**, the goal is to document how you created the user registration form and implemented client-side validation using HTML5 and JavaScript. Your register.html and js/register.js files are the perfect evidence for this.

Here is a guide on what to write in your practical file for Practical 3.

Practical 3: User Registration with Frontend Validation

Aim: To create a user registration page for the ByteBazaar website with effective, user-friendly frontend validation using HTML5 attributes and JavaScript.

1. Key Questions & Analysis

1.1. Are all input types correctly used?

Yes. The registration form (register.html) utilizes appropriate HTML5 input types to ensure a better user experience and leverage built-in browser validation.

- type="text" is used for the First Name and Last Name fields.
- type="email" is used for the Email Address field, which prompts mobile users with an email-specific keyboard.
- type="password" is used for the Password and Confirm Password fields, which automatically masks the input.
- type="checkbox" is used for the "Terms of Service" agreement.

All fields also include the required attribute to prevent form submission if they are left empty.

Example Code Snippet from register.html:

HTML

```
<div class="form-group">
    <label for="register-email">Email Address</label>
    <div class="input-group">
        <i class="fas fa-envelope"></i>
```

```
        <input type="email" id="register-email" placeholder="Enter your email" required>
    </div>
</div>

<div class="form-group">
    <label for="register-password">Password</label>
    <div class="input-group">
        <i class="fas fa-lock"></i>
        <input type="password" id="register-password" placeholder="Create a password" required>
    </div>
</div>
```

1.2. Is JavaScript validation effective and user-friendly?

Yes. The js/register.js script adds a layer of dynamic, client-side validation that is both effective and user-friendly.

- **Real-time Feedback:** Validation logic runs on the blur event (when a user clicks out of a field), providing immediate feedback.
- **Specific Error Messages:** Custom error messages are dynamically created and displayed directly below the invalid field, clearly explaining the issue (e.g., "Passwords do not match," "Please enter a valid email address").
- **Comprehensive Checks:** The script validates multiple conditions:
    - Checks for empty required fields.
    - Validates the email format using a regular expression.
    - Enforces password strength rules (minimum length, uppercase, lowercase, and a number).
    - Ensures that the "Password" and "Confirm Password" fields match.

1.3. Are errors appropriately handled?

Yes. Errors are handled gracefully without interrupting the user's flow.

- **Visual Cues:** Invalid fields are highlighted with a red border, and a descriptive error message appears below them.
- **Submission Prevention:** The form submission is prevented (e.preventDefault()) if any validation rule fails, ensuring that only valid data can be sent to the server.
- **Error Clearing:** As the user corrects an invalid entry, the error message and red border are automatically removed, providing positive reinforcement.

Example JavaScript Validation Logic from js/register.js:

JavaScript

```
// From js/register.js

// Password strength validation
function isValidPassword(password) {
    const passwordRegex = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)[a-zA-Z\d@$!%*?&]{8,}$/;
    return passwordRegex.test(password);
}

// Register Form Submission Handler
registerForm.addEventListener('submit', function (e) {
    e.preventDefault();
    let isValid = true;

    // ... validation checks for all fields ...

    // Validate password
    if (registerPassword.value.trim() === '') {
        showError(registerPassword, 'Password is required');
        isValid = false;
    } else if (!isValidPassword(registerPassword.value)) {
        showError(registerPassword, 'Password must be at least 8 characters with uppercase,
lowercase, and number');
        isValid = false;
    } else {
        clearError(registerPassword);
    }

    // ... more checks ...
});
```

2. Password Strength Meter (Supplementary Problem)

(Note: If you haven't implemented this, you can skip this section. If you want to add it, it would involve creating a small UI element that updates as the user types their password, indicating whether it's "Weak," "Medium," or "Strong.")

A password strength meter was implemented to provide real-time feedback to the user, enhancing security and usability. The meter checks for length, character types (uppercase,

lowercase, numbers, symbols), and updates its visual state accordingly.

3. Test Cases (Post-Laboratory Work)

To demonstrate the effectiveness of the validation, you should include screenshots of the form showing different validation errors.

Test Case 1: Empty Field Error
[Insert a screenshot of the registration form with an empty "First Name" field and its corresponding error message]
Test Case 2: Invalid Email Format
[Insert a screenshot of the form with an invalid email (e.g., "test@test") and its error message]
Test Case 3: Passwords Do Not Match
[Insert a screenshot of the form with mismatching passwords and the resulting error message]
This detailed documentation, supported by code snippets and screenshots, will clearly show that you have successfully completed all requirements for Practical 3.