Week5

Aim: To Implement JavaScript DOM Manipulation and Event Handling for Form Validation

Description:

This project implements a registration form with real-time validation using HTML, CSS, and JavaScript, demonstrating DOM manipulation and event handling for improved user interaction. The form includes fields for Full Name, Email, Password, and Phone Number, each validated through HTML5 attributes and JavaScript regular expressions. Dynamic feedback is provided using inline error messages and visual cues, such as red borders for invalid inputs and green borders for valid ones. The script ensures that emails follow the correct format, passwords meet strength requirements, and phone numbers contain exactly ten digits. Form submission is blocked until all inputs are valid, after which a success message is displayed. This approach enhances usability, reduces errors, and provides a clear example of client-side form validation techniques.

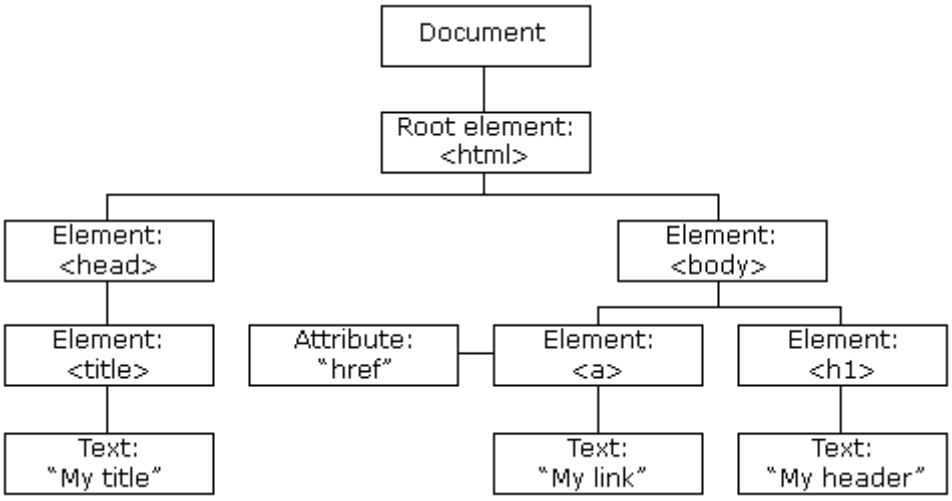
Tasks:

1. **Create an HTML form** with the following fields:
   - Full Name
   - Email
   - Password
   - Phone Number

   Ensure required fields use proper HTML5 validation attributes like required, pattern, and type.
2. **Use JavaScript to add event listeners** for each input field to:
   - Check if the input is empty.
   - Validate the email format using a regular expression.
   - Assess password strength (minimum length, use of special characters, etc.).
   - Validate the phone number format.
3. **Implement dynamic feedback** using DOM manipulation:
   - Display inline error messages for each invalid field.
   - Change the border color or background color to indicate valid/invalid inputs.
   - Show a success message when all validations pass.
4. **Handle form submission:**
   - Prevent submission if any field is invalid.
   - Display a success message dynamically when the form is submitted with valid inputs.

JavaScript DOM Manipulation refers to the process of using JavaScript to access, modify, and interact with elements in a webpage's Document Object Model (DOM), which represents the structure of an HTML document as a tree of nodes. Through DOM manipulation, developers can dynamically change the content, style, and structure of a webpage

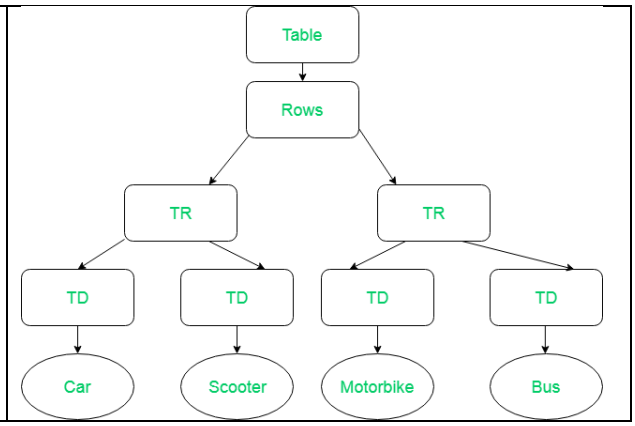
without reloading it. For example, JavaScript can update text inside an element (element.textContent), modify CSS classes (element.classList.add() or element.classList.remove()), or even create and remove elements (document.createElement(), element.appendChild(), element.remove()). Event handling, such as listening for clicks, inputs, or form submissions, also relies on DOM manipulation to trigger changes in real time. In the form validation project, DOM manipulation is used to display error messages, highlight invalid fields with red borders, mark valid fields in green, and show a success message when all conditions are met. This makes webpages interactive, responsive, and user-friendly.



Example : DOM Tree

```
<table>
   <ROWS>
      <tr>
         <td>Car</td>
         <td>Scooter</td>
      </tr>
      <tr>
         <td>MotorBike</td>
         <td>Bus</td>
      </tr>
   </ROWS>

</table>
```



Programs:

Form.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Form Validation with JS DOM</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <form id="myForm" novalidate>
    <h2>Registration Form</h2>

    <div class="form-group">
      <label for="fullname">Full Name</label>
      <input type="text" id="fullname" required>
      <div class="error" id="fullnameError"></div>
    </div>

    <div class="form-group">
      <label for="email">Email</label>
      <input type="email" id="email" required>
      <div class="error" id="emailError"></div>
    </div>

    <div class="form-group">
      <label for="password">Password</label>
      <input type="password" id="password" required>
      <div class="error" id="passwordError"></div>
    </div>

    <div class="form-group">
      <label for="phone">Phone Number</label>
      <input type="tel" id="phone" pattern="[0-9]{10}" required>
      <div class="error" id="phoneError"></div>
    </div>

    <button type="submit">Submit</button>
    <div class="success" id="successMessage"></div>
  </form>
  <script src="script.js"></script>
</body>
</html>
```

Explanation:

The **form.html** file defines the structure of the registration form and connects it with external CSS for styling and JavaScript for validation. It includes four input fields—**Full Name, Email, Password, and Phone Number**—each placed inside a for better organization. Labels are provided for clarity, and each input uses HTML5 attributes like required, type, and pattern to enforce basic validation rules. Below each input, a element is included to display inline error messages dynamically. The form also contains a **Submit** button and a area to show a success message once all validations pass. Finally, the HTML links to **style.css** for visual design and **script.js** for handling validation and DOM events.

style.css

```
body {
  font-family: Arial, sans-serif;
  background: #f4f6f9;
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
}
form {
  background: #fff;
  padding: 20px 30px;
  border-radius: 10px;
  box-shadow: 0 4px 8px rgba(0,0,0,0.2);
  width: 350px;
}
h2 {
  text-align: center;
  margin-bottom: 20px;
}
.form-group {
  margin-bottom: 15px;
}
label {
  font-weight: bold;
  display: block;
  margin-bottom: 5px;
}
input {
  width: 100%;
  padding: 8px;
  border: 2px solid #ccc;
  border-radius: 6px;
  outline: none;
```

```
  transition: border 0.3s;
}
input:focus {
  border-color: #007BFF;
}
.error {
  color: red;
  font-size: 0.9em;
  margin-top: 4px;
}
.success {
  color: green;
  font-weight: bold;
  margin-top: 15px;
  text-align: center;
}
.valid {
  border-color: green !important;
}
.invalid {
  border-color: red !important;
}
button {
  width: 100%;
  padding: 10px;
  background: #007BFF;
  border: none;
  color: white;
  border-radius: 6px;
  font-size: 1em;
  cursor: pointer;
}
button:hover {
  background: #0056b3;
}
```

Explanation:

The **style.css** file is responsible for designing and enhancing the appearance of the registration form, making it clean, user-friendly, and visually responsive. It applies a centered layout with a light background and styles the form with padding, rounded corners, and subtle shadows for a modern look. Each input field is given consistent spacing, borders, and focus effects, while labels and error messages are clearly styled for readability. The file also uses visual cues for validation by applying **green borders for valid inputs** and **red borders for invalid inputs**, helping users instantly recognize errors. Additionally, the **Submit button** is styled with a blue theme that changes shade on hover, and success messages are highlighted in green for clear confirmation. This ensures both aesthetic appeal and intuitive user interaction.

Script.js

```
const form = document.getElementById("myForm");
const fullname = document.getElementById("fullname");
const email = document.getElementById("email");
const password = document.getElementById("password");
const phone = document.getElementById("phone");

const fullnameError = document.getElementById("fullnameError");
const emailError = document.getElementById("emailError");
const passwordError = document.getElementById("passwordError");
const phoneError = document.getElementById("phoneError");
const successMessage = document.getElementById("successMessage");

// Regex patterns
const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
const passwordRegex = /^(?=.*[0-9])(?=.*[!@#$%^&*]).{6,}$/;
const phoneRegex = /^[0-9]{10}$/;

function validateFullname() {
  if (fullname.value.trim() === "") {
    fullnameError.textContent = "Full Name is required";
    fullname.classList.add("invalid");
    fullname.classList.remove("valid");
    return false;
  }
  fullnameError.textContent = "";
  fullname.classList.add("valid");
  fullname.classList.remove("invalid");
  return true;
}

function validateEmail() {
  if (!emailRegex.test(email.value)) {
    emailError.textContent = "Invalid email format";
    email.classList.add("invalid");
    email.classList.remove("valid");
    return false;
  }
  emailError.textContent = "";
  email.classList.add("valid");
  email.classList.remove("invalid");
  return true;
}
```

```
function validatePassword() {
  if (!passwordRegex.test(password.value)) {
    passwordError.textContent = "Password must be 6+ chars, include number & special char";
    password.classList.add("invalid");
    password.classList.remove("valid");
    return false;
  }
  passwordError.textContent = "";
  password.classList.add("valid");
  password.classList.remove("invalid");
  return true;
}

function validatePhone() {
  if (!phoneRegex.test(phone.value)) {
    phoneError.textContent = "Phone must be 10 digits";
    phone.classList.add("invalid");
    phone.classList.remove("valid");
    return false;
  }
  phoneError.textContent = "";
  phone.classList.add("valid");
  phone.classList.remove("invalid");
  return true;
}

// Add event listeners for real-time validation
fullname.addEventListener("input", validateFullname);
email.addEventListener("input", validateEmail);
password.addEventListener("input", validatePassword);
phone.addEventListener("input", validatePhone);

// Handle form submission
form.addEventListener("submit", function (e) {
  e.preventDefault();

  let isValid = validateFullname() & validateEmail() & validatePassword() & validatePhone();

  if (isValid) {
    successMessage.textContent = "Form submitted successfully!";
    form.reset();
    fullname.classList.remove("valid");
    email.classList.remove("valid");
    password.classList.remove("valid");
    phone.classList.remove("valid");
  } else {
```

```
    successMessage.textContent = "";
  }
});
```

Explanation:

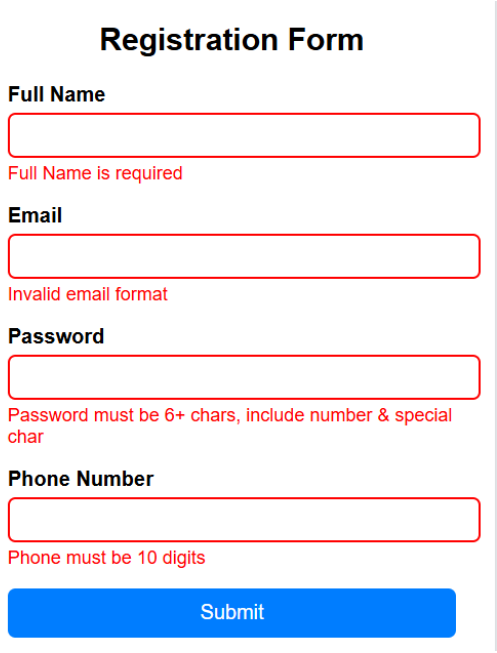
The **script.js** file contains the logic for validating user input and handling form submission through DOM manipulation and event listeners. It first selects all form fields and error message containers, then defines **regular expressions** to check email format, password strength, and phone number correctness. Separate validation functions are written for each field, which update the DOM by showing error messages and changing input border colors to green (valid) or red (invalid). Event listeners are added to each input field to trigger real-time validation as the user types. On form submission, the script prevents default submission with e.preventDefault(), checks all fields, and only displays a **success message** if every validation passes. If any field is invalid, the corresponding error message is shown, ensuring users cannot submit incorrect data.

Test cases

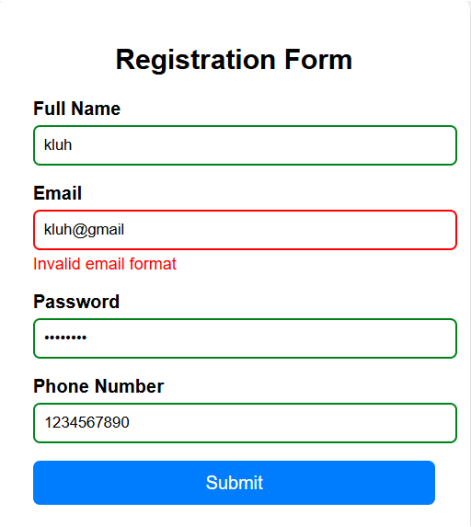
Test Case	Input Values	Expected Result
Empty Form	**Leave all fields empty and click Submit**	**Error messages for all fields: *"Full Name is required"*, *"Invalid email format"*, *"Password must be 6+ chars, include number & special char"*, *"Phone must be 10 digits"*.**
Invalid Email	**Name = "John", Email = "john@", Password = "abc!1", Phone = "1234567890"**	**Email shows error *"Invalid email format"*. Others valid.**
Weak Password	**Name = "John Doe", Email = "john@example.com", Password = "abc123", Phone = "9876543210"**	**Password shows error *"Password must be 6+ chars, include number & special char"*.**
Invalid Phone	**Name = "Alice", Email = "alice@test.com", Password = "Abc@123", Phone = "12345"**	**Phone shows error *"Phone must be 10 digits"*.**
Valid Form	**Name = "Alice Smith", Email = "alice@test.com", Password = "Abc@123", Phone = "9876543210"**	**Green borders on all fields + Success message *"Form submitted successfully!"* displayed.**

Outputs:

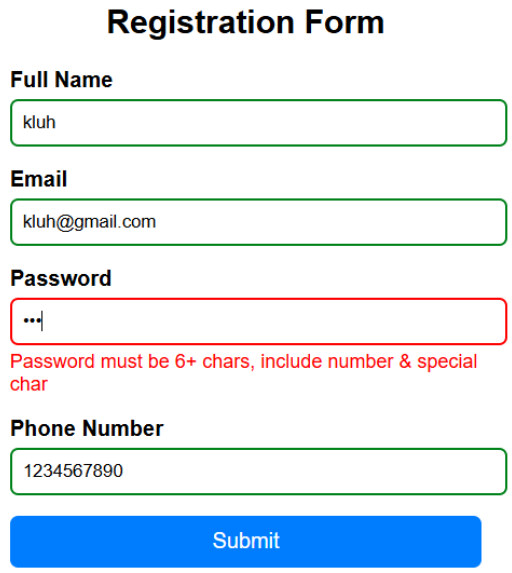
Empty Form	**Leave all fields empty and click Submit**	**Error messages for all fields: *"Full Name is required", "Invalid email format", "Password must be 6+ chars, include number & special char", "Phone must be 10 digits".***



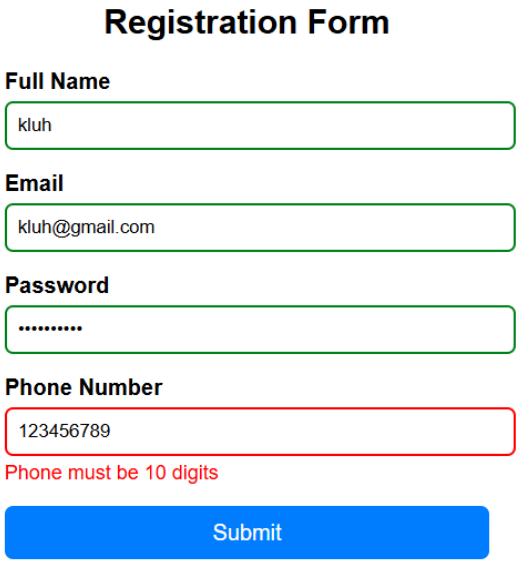
Invalid Email	**Name = "John", Email = "john@", Password = "abc!1", Phone = "1234567890"**	**Email shows error *"Invalid email format"*. Others valid.**



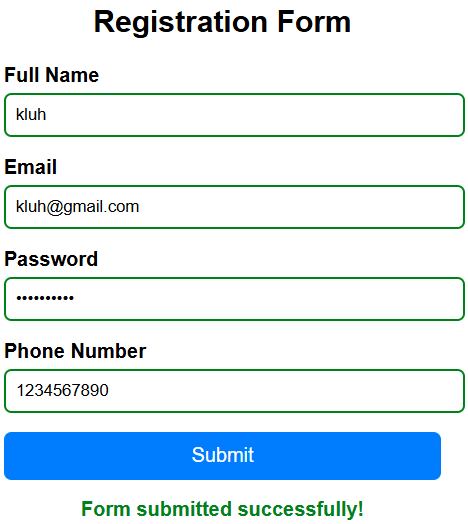
Weak Password	**Name = "John Doe", Email = "john@example.com", Password = "abc123", Phone = "9876543210"**	**Password shows error *"Password must be 6+ chars, include number & special char"*.**



Invalid Phone	**Name = "Alice", Email = "alice@test.com", Password = "Abc@123", Phone = "12345"**	**Phone shows error *"Phone must be 10 digits"*.**



Valid Form	**Name = "Alice Smith", Email = "alice@test.com", Password = "Abc@123", Phone = "9876543210"**	**Green borders on all fields + Success message *"Form submitted successfully!"* displayed.**



Results:

The combined results of the three programs (**form.html, style.css, and script.js**) produce a fully functional registration form that validates user input in real time using JavaScript DOM manipulation and event handling. The **form.html** provides the structure with input fields, error containers, and a success message area, while **style.css** enhances usability by styling the form with clear layouts, visual cues, and distinct colors for valid and invalid inputs. The **script.js** adds the logic to check each field against defined rules—ensuring the full name is not empty, the email matches a valid format, the password is strong enough, and the phone number has exactly ten digits. Errors are shown instantly with red borders and messages, while valid inputs are highlighted in green. On successful submission, the form displays a confirmation message and resets. Together, these programs demonstrate an interactive, user-friendly form with effective client-side validation.

Note : Zip File
https://github.com/subbusir/FEDF