Full Stack MidSem1 Lab evaluation

Experiment:-

3(a). Create an HTML page with: an input box, an empty list (<ul>), and two buttons: Add Item and Delete Item. When the Add Item button is clicked, the text entered in the input box should be added to the list. When the Delete Item button is clicked, the corresponding item (if present) should be removed from the list. Also, display the current count of items in the list.

Lab 3 (b). Create a form with the following fields: Name Email Date of Birth (DOB) Pincode Mobile Number IP Address Validate each field on keypress (i.e., while the user is typing), without using a separate Validate button.

Submitted by:-

Saransh Gupta

123103002

Submitted to:-

Prof. Chandresh Kumar

Codes :-

3a)

Working example:-

Dynamic List Manager

Add and remove items from your list dynamically. See the current count of items below.

Enter an item to add to the list...

Add Item | Delete Item

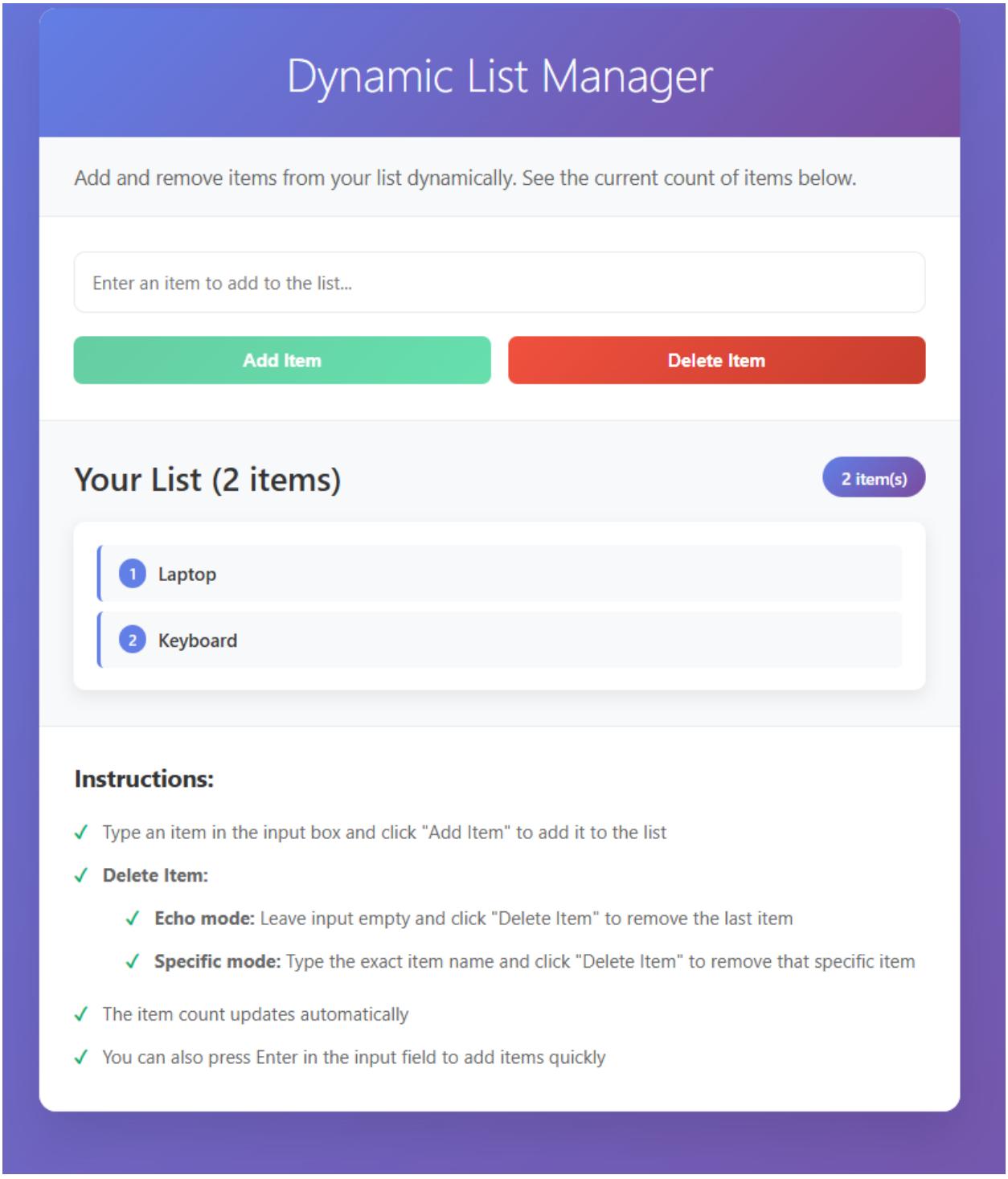
Your List (3 items)

3 item(s)

1. Laptop
2. Mouse
3. Keyboard

Instructions:

- ✓ Type an item in the input box and click "Add Item" to add it to the list
- ✓ **Delete Item:**
  - ✓ **Echo mode:** Leave input empty and click "Delete Item" to remove the last item
  - ✓ **Specific mode:** Type the exact item name and click "Delete Item" to remove that specific item
- ✓ The item count updates automatically
- ✓ You can also press Enter in the input field to add items quickly



Index.html

```
<!DOCTYPE html>

<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Dynamic List Manager</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="container">
        <h1>Dynamic List Manager</h1>
        <p class="description">Add and remove items from your list dynamically. See the current count of items below.</p>

        <div class="input-section">
            <div class="input-group">
                <input type="text" id="itemInput" placeholder="Enter an item to add to the list...">
                <div class="button-group">
                    <button id="addBtn">Add Item</button>
                    <button id="deleteBtn">Delete Item</button>
                </div>
            </div>
        </div>

        <div class="list-section">
            <div class="list-header">
                <h2>Your List</h2>
                <div class="item-count">
                    <span id="itemCount">0</span> item(s)
                </div>
```

```
        </div>
        <ul id="itemList" class="item-list">
            <!-- Items will be added here dynamically -->
        </ul>
    </div>

    <div class="instructions">
        <h3>Instructions:</h3>
        <ul>
            <li>Type an item in the input box and click "Add Item" to add it to the list</li>
            <li><strong>Delete Item:</strong>
                <ul style="margin-top: 8px; margin-left: 20px;">
                    <li><strong>Echo mode:</strong> Leave input empty and click "Delete Item" to remove the last item</li>
                    <li><strong>Specific mode:</strong> Type the exact item name and click "Delete Item" to remove that specific item</li>
                </ul>
            </li>
            <li>The item count updates automatically</li>
            <li>You can also press Enter in the input field to add items quickly</li>
        </ul>
    </div>
</div>

<script src="script.js"></script>
</body>
</html>
```

Style.css

```
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
    min-height: 100vh;
    padding: 20px;
}

.container {
    max-width: 800px;
    margin: 0 auto;
    background: white;
    border-radius: 15px;
    box-shadow: 0 20px 40px rgba(0, 0, 0, 0.1);
    overflow: hidden;
}

h1 {
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
    color: white;
    text-align: center;
    padding: 30px;
    font-size: 2.5em;
    font-weight: 300;
```

```
    margin: 0;
}

.description {
    padding: 20px 30px;
    background: #f8f9fa;
    color: #666;
    font-size: 1.1em;
    line-height: 1.6;
    border-bottom: 1px solid #e9ecef;
}

.input-section {
    padding: 30px;
    background: white;
}

.input-group {
    display: flex;
    flex-direction: column;
    gap: 20px;
}

#itemInput {
    width: 100%;
    padding: 15px;
    border: 2px solid #e9ecef;
    border-radius: 10px;
    font-size: 16px;
    font-family: inherit;
    transition: border-color 0.3s ease;
}

#itemInput:focus {
```

```
    outline: none;
    border-color: #667eea;
    box-shadow: 0 0 0 3px rgba(102, 126, 234, 0.1);
}

.button-group {
    display: flex;
    gap: 15px;
    flex-wrap: wrap;
}

button {
    padding: 12px 24px;
    border: none;
    border-radius: 8px;
    cursor: pointer;
    font-size: 1em;
    font-weight: 600;
    transition: all 0.3s ease;
    min-width: 120px;
    flex: 1;
}

#addBtn {
    background: linear-gradient(135deg, #27ae60 0%, #2ecc71 100%);
    color: white;
}

#addBtn:hover {
    transform: translateY(-2px);
    box-shadow: 0 8px 20px rgba(39, 174, 96, 0.3);
}

#deleteBtn {
```

```
    background: linear-gradient(135deg, #e74c3c 0%, #c0392b 100%);
    color: white;
}

#deleteBtn:hover {
    transform: translateY(-2px);
    box-shadow: 0 8px 20px rgba(231, 76, 60, 0.3);
}

button:active {
    transform: translateY(0);
}

button:disabled {
    opacity: 0.6;
    cursor: not-allowed;
    transform: none;
}

.list-section {
    padding: 30px;
    background: #f8f9fa;
    border-top: 1px solid #e9ecef;
}

.list-header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 20px;
    flex-wrap: wrap;
    gap: 10px;
}
```

```
.list-header h2 {
    color: #333;
    font-size: 1.8em;
    font-weight: 500;
}

.item-count {
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
    color: white;
    padding: 8px 16px;
    border-radius: 20px;
    font-weight: 600;
    font-size: 0.9em;
}

.item-list {
    list-style: none;
    background: white;
    border-radius: 10px;
    padding: 20px;
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.08);
    min-height: 100px;
}

.item-list:empty::before {
    content: "No items in the list yet. Add some items to get started!";
    display: block;
    text-align: center;
    color: #999;
    font-style: italic;
    padding: 40px 20px;
}

.list-item {
```

```
    display: flex;
    align-items: center;
    justify-content: space-between;
    padding: 12px 16px;
    margin-bottom: 8px;
    background: #f8f9fa;
    border-left: 4px solid #667eea;
    border-radius: 6px;
    transition: all 0.3s ease;
    animation: slideIn 0.3s ease;
}

.list-item:last-child {
    margin-bottom: 0;
}

.list-item:hover {
    background: #e9ecef;
    transform: translateX(5px);
}

.item-text {
    flex: 1;
    color: #333;
    font-weight: 500;
}

.item-number {
    background: #667eea;
    color: white;
    padding: 4px 8px;
    border-radius: 12px;
    font-size: 0.8em;
    font-weight: 600;
```

```
    margin-right: 10px;
    min-width: 24px;
    text-align: center;
}

.instructions {
    padding: 30px;
    background: white;
    border-top: 1px solid #e9ecef;
}

.instructions h3 {
    color: #333;
    margin-bottom: 15px;
    font-size: 1.3em;
}

.instructions ul {
    list-style: none;
    padding-left: 0;
}

.instructions li {
    padding: 8px 0;
    color: #666;
    position: relative;
    padding-left: 25px;
}

.instructions li::before {
    content: "✓";
    position: absolute;
    left: 0;
```

```
    color: #27ae60;
    font-weight: bold;
}

@keyframes slideIn {
    from {
        opacity: 0;
        transform: translateX(-20px);
    }
    to {
        opacity: 1;
        transform: translateX(0);
    }
}

@keyframes slideOut {
    from {
        opacity: 1;
        transform: translateX(0);
    }
    to {
        opacity: 0;
        transform: translateX(20px);
    }
}

.list-item.removing {
    animation: slideOut 0.3s ease forwards;
}

.list-item.deleting {
    background: linear-gradient(135deg, #e74c3c 0%, #c0392b 100%) !important;
    color: white !important;
    border-left-color: #c0392b !important;
```

```
    transform: scale(1.02);
    box-shadow: 0 4px 12px rgba(231, 76, 60, 0.3);
}

.list-item.deleting .item-text {
    color: white !important;
}

.list-item.deleting .item-number {
    background: rgba(255, 255, 255, 0.2) !important;
    color: white !important;
}

/* Responsive design */
@media (max-width: 768px) {
    .container {
        margin: 10px;
        border-radius: 10px;
    }

    h1 {
        font-size: 2em;
        padding: 20px;
    }

    .description,
    .input-section,
    .list-section,
    .instructions {
        padding: 20px;
    }

    .button-group {
        flex-direction: column;
```

```
    }

    button {
        width: 100%;
    }

    .list-header {
        flex-direction: column;
        align-items: flex-start;
    }

    .item-count {
        align-self: flex-end;
    }
}
```

Script.js

```
// Dynamic List Manager
class ListManager {
    constructor() {
        this.items = [];
        this.itemCounter = 0;
        this.initializeElements();
        this.attachEventListeners();
        this.updateDisplay();
    }

    initializeElements() {
        this.itemInput = document.getElementById('itemInput');
```

```
    this.addBtn = document.getElementById('addBtn');
    this.deleteBtn = document.getElementById('deleteBtn');
    this.itemList = document.getElementById('itemList');
    this.itemCount = document.getElementById('itemCount');
}

attachEventListeners() {
    // Add button click
    this.addBtn.addEventListener('click', () => this.addItem());

    // Delete button click
    this.deleteBtn.addEventListener('click', () => this.deleteItem());

    // Enter key press in input
    this.itemInput.addEventListener('keypress', (e) => {
        if (e.key === 'Enter') {
            this.addItem();
        }
    });

    // Input validation
    this.itemInput.addEventListener('input', () => {
        this.validateInput();
    });
}

addItem() {
    const inputValue = this.itemInput.value.trim();

    if (inputValue === '') {
        this.showMessage('Please enter an item to add!', 'error');
        this.itemInput.focus();
        return;
    }
```

```
    // Check for duplicate items
    if (this.items.includes(inputValue)) {
        this.showMessage('This item already exists in the list!', 'warning');
        this.itemInput.focus();
        return;
    }

    // Add item to array
    this.items.push(inputValue);
    this.itemCounter++;

    // Create list item element
    const listItem = this.createListItem(inputValue, this.itemCounter);

    // Add to DOM with animation
    this.itemList.appendChild(listItem);

    // Clear input and update display
    this.itemInput.value = '';
    this.updateDisplay();
    this.renumberItems();
    this.validateInput();

    // Show success message
    this.showMessage(`"${inputValue}" added to the list!`, 'success');

    // Focus back to input
    this.itemInput.focus();
}

deleteItem() {
    if (this.items.length === 0) {
        this.showMessage('No items to delete!', 'warning');
```

```
    return;
}

const inputValue = this.itemInput.value.trim();
let itemToDelete;
let itemIndex = -1;
let listItemToRemove;

if (inputValue === '') {
    // Echo mode: Delete the last item
    itemToDelete = this.items.pop();
    listItemToRemove = this.itemList.lastElementChild;
    this.showMessage(`"${itemToDelete}" removed from the list!`, 'success');
} else {
    // Specific item mode: Delete the specified item if it exists
    itemIndex = this.items.indexOf(inputValue);
    if (itemIndex !== -1) {
        itemToDelete = this.items.splice(itemIndex, 1)[0];
        // Find the corresponding DOM element
        const listItems = this.itemList.children;
        for (let i = 0; i < listItems.length; i++) {
            const itemText = listItems[i].querySelector('.item-text').textContent;
            if (itemText === itemToDelete) {
                listItemToRemove = listItems[i];
                break;
            }
        }
        this.showMessage(`"${itemToDelete}" removed from the list!`, 'success');
        // Clear the input after successful deletion
        this.itemInput.value = '';
    } else {
        this.showMessage(`"${inputValue}" not found in the list!`, 'warning');
        return;
    }
```

```
    }

    // Add visual feedback and removing animation
    if (listItemToRemove) {

        listItemToRemove.classList.add('deleting');

        // After a brief moment, start the removal animation
        setTimeout(() => {
            listItemToRemove.classList.remove('deleting');
            listItemToRemove.classList.add('removing');

            // Remove from DOM after animation
            setTimeout(() => {
                if (listItemToRemove.parentNode) {
                    listItemToRemove.remove();
                    // Renumber items after DOM removal
                    this.renumberItems();
                }
            }, 300);
        }, 200);
    }

    // Update display
    this.updateDisplay();
    this.validateInput();
}

createListItem(text, number) {
    const li = document.createElement('li');
    li.className = 'list-item';
    li.innerHTML = `
        <span class="item-number">${number}</span>
        <span class="item-text">${this.escapeHtml(text)}</span>
```

```
    `;
    return li;
}

updateDisplay() {
    // Update item count
    this.itemCount.textContent = this.items.length;

    // Update button states
    this.deleteBtn.disabled = this.items.length === 0;

    // Update list header
    const listHeader = document.querySelector('.list-header h2');
    if (this.items.length === 0) {
        listHeader.textContent = 'Your List';
    } else if (this.items.length === 1) {
        listHeader.textContent = 'Your List (1 item)';
    } else {
        listHeader.textContent = `Your List (${this.items.length} items)`;
    }
}

renumberItems() {
    // Renumber all remaining items sequentially starting from 1
    const listItems = this.itemList.children;
    for (let i = 0; i < listItems.length; i++) {
        const itemNumber = listItems[i].querySelector('.item-number');
        if (itemNumber) {
            itemNumber.textContent = i + 1;
        }
    }
}

validateInput() {
```

```
    const inputValue = this.itemInput.value.trim();
    const isEmpty = inputValue === '';
    const isDuplicate = this.items.includes(inputValue);

    // Update add button state
    this.addBtn.disabled = isEmpty || isDuplicate;

    // Update input styling
    if (isDuplicate && !isEmpty) {
        this.itemInput.style.borderColor = '#f39c12';
        this.itemInput.style.backgroundColor = '#fef9e7';
    } else {
        this.itemInput.style.borderColor = '#e9ecef';
        this.itemInput.style.backgroundColor = 'white';
    }
}

showMessage(message, type = 'info') {
    // Remove existing message
    const existingMessage = document.querySelector('.message');
    if (existingMessage) {
        existingMessage.remove();
    }

    // Create new message
    const messageDiv = document.createElement('div');
    messageDiv.className = `message message-${type}`;
    messageDiv.textContent = message;

    // Style the message
    messageDiv.style.cssText = `
        position: fixed;
        top: 20px;
        right: 20px;
```

```
    padding: 12px 20px;
    border-radius: 8px;
    color: white;
    font-weight: 600;
    z-index: 1000;
    animation: slideInRight 0.3s ease;
    max-width: 300px;
    word-wrap: break-word;
`;

// Set background color based on type
switch (type) {
    case 'success':
        messageDiv.style.background = 'linear-gradient(135deg, #27ae60 0%, #2ecc71 100%)';
        break;
    case 'error':
        messageDiv.style.background = 'linear-gradient(135deg, #e74c3c 0%, #c0392b 100%)';
        break;
    case 'warning':
        messageDiv.style.background = 'linear-gradient(135deg, #f39c12 0%, #e67e22 100%)';
        break;
    default:
        messageDiv.style.background = 'linear-gradient(135deg, #667eea 0%, #764ba2 100%)';
}

// Add to document
document.body.appendChild(messageDiv);

// Auto remove after 3 seconds
setTimeout(() => {
    if (messageDiv.parentNode) {
        messageDiv.style.animation = 'slideOutRight 0.3s ease';
        setTimeout(() => {
            if (messageDiv.parentNode) {
```

```
                messageDiv.remove();
            }
        }, 300);
    }
}, 3000);
}

escapeHtml(text) {
    const div = document.createElement('div');
    div.textContent = text;
    return div.innerHTML;
}

// Public methods for external access
getItems() {
    return [...this.items]; // Return a copy
}

getItemCount() {
    return this.items.length;
}

clearAll() {
    this.items = [];
    this.itemList.innerHTML = '';
    this.updateDisplay();
    this.renumberItems();
    this.showMessage('All items cleared!', 'info');
}
}

// Add CSS animations
const style = document.createElement('style');
style.textContent = `
```

```
    @keyframes slideInRight {
        from {
            transform: translateX(100%);
            opacity: 0;
        }
        to {
            transform: translateX(0);
            opacity: 1;
        }
    }

    @keyframes slideOutRight {
        from {
            transform: translateX(0);
            opacity: 1;
        }
        to {
            transform: translateX(100%);
            opacity: 0;
        }
    }
`;
document.head.appendChild(style);

// Initialize the list manager when DOM is loaded
document.addEventListener('DOMContentLoaded', function() {
    const listManager = new ListManager();

    // Add keyboard shortcuts
    document.addEventListener('keydown', function(e) {
        // Ctrl/Cmd + Enter to add item
        if ((e.ctrlKey || e.metaKey) && e.key === 'Enter') {
            e.preventDefault();
            listManager.addItem();
```

```
        }

        // Ctrl/Cmd + Delete to delete item
        if ((e.ctrlKey || e.metaKey) && e.key === 'Delete') {
            e.preventDefault();
            listManager.deleteItem();
        }
    });

    // Add some demo functionality
    console.log('Dynamic List Manager initialized!');
    console.log('Keyboard shortcuts:');
    console.log('- Enter: Add item');
    console.log('- Ctrl/Cmd + Enter: Add item');
    console.log('- Ctrl/Cmd + Delete: Delete last item');
});
```

3b

Working example:-

Real-time Form Validation

Fill out the form below. All fields are validated in real-time as you type, without needing a separate validate button.

Name *

saransh gupta

✓ Valid

Email *

123103002@nitkkr.ac.in

✓ Valid

Date of Birth *

03/02/2004

✓ Valid

Pincode *

244001

✓ Valid

Mobile Number *

8979448556

✓ Valid

IP Address *

1.34.134

Please enter a valid IP address (e.g., 192.168.1.1)

Clear Form

Submit Form

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Real-time Form Validation</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <div class="container">
    <h1>Real-time Form Validation</h1>
    <p class="description">Fill out the form below. All fields are validated in real-time as you type, without needing a separate validate button.</p>

    <form id="validationForm" class="form">
      <div class="form-group">
        <label for="name">Name *</label>
        <input type="text" id="name" name="name" placeholder="Enter your full name">
        <div class="validation-message" id="nameMessage"></div>
      </div>

      <div class="form-group">
        <label for="email">Email *</label>
        <input type="email" id="email" name="email" placeholder="Enter your email address">
        <div class="validation-message" id="emailMessage"></div>
      </div>

      <div class="form-group">
        <label for="dob">Date of Birth *</label>
        <input type="date" id="dob" name="dob">
```

```
            <div class="validation-message" id="dobMessage"></div>
        </div>

        <div class="form-group">
            <label for="pincode">Pincode *</label>
            <input type="text" id="pincode" name="pincode" placeholder="Enter 6-digit pincode" maxlength="6">
            <div class="validation-message" id="pincodeMessage"></div>
        </div>

        <div class="form-group">
            <label for="mobile">Mobile Number *</label>
            <input type="tel" id="mobile" name="mobile" placeholder="Enter 10-digit mobile number" maxlength="10">
            <div class="validation-message" id="mobileMessage"></div>
        </div>

        <div class="form-group">
            <label for="ip">IP Address *</label>
            <input type="text" id="ip" name="ip" placeholder="Enter IP address (e.g., 192.168.1.1)">
            <div class="validation-message" id="ipMessage"></div>
        </div>

        <div class="form-actions">
            <button type="button" id="clearBtn">Clear Form</button>
            <button type="submit" id="submitBtn" disabled>Submit Form</button>
        </div>
    </form>

    <div class="validation-summary">
        <h3>Validation Summary</h3>
        <div id="summaryContent">
            <p>Start filling the form to see real-time validation results.</p>
        </div>
    </div>
</div>
```

```
    <script src="script.js"></script>
</body>
</html>
```

Style.css

```
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    background: linear-gradient(135deg, #2c3e50 0%, #3498db 100%);
    min-height: 100vh;
    padding: 20px;
}

.container {
    max-width: 600px;
    margin: 0 auto;
    background: white;
    border-radius: 15px;
    box-shadow: 0 20px 40px rgba(0, 0, 0, 0.1);
    overflow: hidden;
}

h1 {
    background: linear-gradient(135deg, #2c3e50 0%, #3498db 100%);
```

```
    color: white;
    text-align: center;
    padding: 30px;
    font-size: 2.5em;
    font-weight: 300;
    margin: 0;
}

.description {
    padding: 20px 30px;
    background: #ecf0f1;
    color: #2c3e50;
    font-size: 1.1em;
    line-height: 1.6;
    border-bottom: 1px solid #bdc3c7;
}

.form {
    padding: 30px;
    background: white;
}

.form-group {
    margin-bottom: 25px;
}

.form-group label {
    display: block;
    margin-bottom: 8px;
    font-weight: 600;
    color: #2c3e50;
    font-size: 1.1em;
}
```

```
.form-group input {
    width: 100%;
    padding: 12px 15px;
    border: 2px solid #bdc3c7;
    border-radius: 8px;
    font-size: 16px;
    font-family: inherit;
    transition: all 0.3s ease;
    background: #f8f9fa;
}

.form-group input:focus {
    outline: none;
    border-color: #3498db;
    box-shadow: 0 0 0 3px rgba(52, 152, 219, 0.1);
    background: white;
}

.form-group input.valid {
    border-color: #27ae60;
    background: #d5f4e6;
}

.form-group input.invalid {
    border-color: #e74c3c;
    background: #fadbd8;
}

.form-group input.warning {
    border-color: #f39c12;
    background: #fef9e7;
}

.validation-message {
```

```
    margin-top: 5px;
    font-size: 0.9em;
    font-weight: 500;
    min-height: 20px;
    transition: all 0.3s ease;
}

.validation-message.valid {
    color: #27ae60;
}

.validation-message.invalid {
    color: #e74c3c;
}

.validation-message.warning {
    color: #f39c12;
}

.validation-message.info {
    color: #3498db;
}

.form-actions {
    display: flex;
    gap: 15px;
    margin-top: 30px;
    flex-wrap: wrap;
}

.form-actions button {
    padding: 12px 24px;
    border: none;
    border-radius: 8px;
```

```
    cursor: pointer;
    font-size: 1em;
    font-weight: 600;
    transition: all 0.3s ease;
    flex: 1;
    min-width: 120px;
}

#clearBtn {
    background: linear-gradient(135deg, #95a5a6 0%, #7f8c8d 100%);
    color: white;
}

#clearBtn:hover {
    transform: translateY(-2px);
    box-shadow: 0 8px 20px rgba(149, 165, 166, 0.3);
}

#submitBtn {
    background: linear-gradient(135deg, #27ae60 0%, #2ecc71 100%);
    color: white;
}

#submitBtn:hover:not(:disabled) {
    transform: translateY(-2px);
    box-shadow: 0 8px 20px rgba(39, 174, 96, 0.3);
}

#submitBtn:disabled {
    background: #bdc3c7;
    cursor: not-allowed;
    transform: none;
}
```

```
button:active {
    transform: translateY(0);
}

.validation-summary {
    padding: 30px;
    background: #f8f9fa;
    border-top: 1px solid #bdc3c7;
}

.validation-summary h3 {
    color: #2c3e50;
    margin-bottom: 15px;
    font-size: 1.5em;
    font-weight: 500;
}

#summaryContent {
    background: white;
    border-radius: 8px;
    padding: 20px;
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.05);
}

.summary-stats {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(150px, 1fr));
    gap: 15px;
    margin-bottom: 20px;
}

.stat-item {
    text-align: center;
    padding: 15px;
```

```
    border-radius: 8px;
    font-weight: 600;
}

.stat-item.valid {
    background: #d5f4e6;
    color: #27ae60;
}

.stat-item.invalid {
    background: #fadbd8;
    color: #e74c3c;
}

.stat-item.warning {
    background: #fef9e7;
    color: #f39c12;
}

.stat-number {
    display: block;
    font-size: 1.5em;
    margin-bottom: 5px;
}

.stat-label {
    font-size: 0.9em;
}

.field-status {
    display: flex;
    align-items: center;
    gap: 10px;
    padding: 8px 0;
```

```
    border-bottom: 1px solid #ecf0f1;
}

.field-status:last-child {
    border-bottom: none;
}

.field-name {
    font-weight: 600;
    min-width: 120px;
    color: #2c3e50;
}

.field-status-icon {
    width: 20px;
    height: 20px;
    border-radius: 50%;
    display: flex;
    align-items: center;
    justify-content: center;
    font-size: 0.8em;
    font-weight: bold;
}

.field-status-icon.valid {
    background: #27ae60;
    color: white;
}

.field-status-icon.invalid {
    background: #e74c3c;
    color: white;
}
```

```
.field-status-icon.warning {
    background: #f39c12;
    color: white;
}

.field-status-icon.pending {
    background: #bdc3c7;
    color: white;
}

.field-message {
    flex: 1;
    font-size: 0.9em;
    color: #7f8c8d;
}

/* Responsive design */
@media (max-width: 768px) {
    .container {
        margin: 10px;
        border-radius: 10px;
    }

    h1 {
        font-size: 2em;
        padding: 20px;
    }

    .description,
    .form,
    .validation-summary {
        padding: 20px;
    }
```

```
    .form-actions {
        flex-direction: column;
    }

    .form-actions button {
        width: 100%;
    }

    .summary-stats {
        grid-template-columns: 1fr;
    }

    .field-status {
        flex-direction: column;
        align-items: flex-start;
        gap: 5px;
    }

    .field-name {
        min-width: auto;
    }
}

/* Animation for validation messages */
@keyframes slideDown {
    from {
        opacity: 0;
        transform: translateY(-10px);
    }
    to {
        opacity: 1;
        transform: translateY(0);
    }
}
```

```
.validation-message {
    animation: slideDown 0.3s ease;
}

/* Loading animation for real-time validation */
.form-group input.validating {
    position: relative;
}

.form-group input.validating::after {
    content: '';
    position: absolute;
    right: 10px;
    top: 50%;
    transform: translateY(-50%);
    width: 16px;
    height: 16px;
    border: 2px solid #3498db;
    border-top: 2px solid transparent;
    border-radius: 50%;
    animation: spin 1s linear infinite;
}

@keyframes spin {
    0% { transform: translateY(-50%) rotate(0deg); }
    100% { transform: translateY(-50%) rotate(360deg); }
}
```

Script.js

```
// Real-time Form Validation System
class FormValidator {
    constructor() {
        this.fields = {
            name: { required: true, pattern: /^[a-zA-Z\s]{2,50}$/, message: 'Name must be 2-50 characters, letters and spaces only' },
            email: { required: true, pattern: /^[^\s@]+@[^\s@]+\.[^\s@]+$/, message: 'Please enter a valid email address' },
            dob: { required: true, validator: this.validateDOB, message: 'Please enter a valid date of birth' },
            pincode: { required: true, pattern: /^\d{6}$/, message: 'Pincode must be exactly 6 digits' },
            mobile: { required: true, pattern: /^\d{10}$/, message: 'Mobile number must be exactly 10 digits' },
            ip: { required: true, validator: this.validateIP, message: 'Please enter a valid IP address' }
        };

        this.validationState = {
            name: { isValid: false, message: '', isTouched: false },
            email: { isValid: false, message: '', isTouched: false },
            dob: { isValid: false, message: '', isTouched: false },
            pincode: { isValid: false, message: '', isTouched: false },
            mobile: { isValid: false, message: '', isTouched: false },
            ip: { isValid: false, message: '', isTouched: false }
        };

        this.initializeElements();
        this.attachEventListeners();
        this.updateSubmitButton();
    }

    initializeElements() {
        this.form = document.getElementById('validationForm');
        this.submitBtn = document.getElementById('submitBtn');
        this.clearBtn = document.getElementById('clearBtn');
        this.summaryContent = document.getElementById('summaryContent');
    }

    attachEventListeners() {
```

```
// Add event listeners for each field
Object.keys(this.fields).forEach(fieldName => {
    const input = document.getElementById(fieldName);
    const messageElement = document.getElementById(`${fieldName}Message`);

    // Real-time validation on input
    input.addEventListener('input', (e) => {
        this.validateField(fieldName, e.target.value);
        this.updateFieldDisplay(fieldName);
        this.updateSubmitButton();
        this.updateSummary();
    });

    // Validation on blur (when user leaves field)
    input.addEventListener('blur', (e) => {
        this.validationState[fieldName].isTouched = true;
        this.validateField(fieldName, e.target.value);
        this.updateFieldDisplay(fieldName);
        this.updateSubmitButton();
        this.updateSummary();
    });

    // Clear validation state on focus
    input.addEventListener('focus', () => {
        this.clearFieldValidation(fieldName);
    });
});

// Form submission
this.form.addEventListener('submit', (e) => {
    e.preventDefault();
    this.handleFormSubmission();
});
```

```
    // Clear form
    this.clearBtn.addEventListener('click', () => {
        this.clearForm();
    });
}

validateField(fieldName, value) {
    const field = this.fields[fieldName];
    const state = this.validationState[fieldName];

    // Reset state
    state.isValid = false;
    state.message = '';

    // Check if field is empty
    if (field.required && (!value || value.trim() === '')) {
        state.message = `${this.getFieldDisplayName(fieldName)} is required`;
        return;
    }

    // Skip validation if field is empty and not required
    if (!value || value.trim() === '') {
        state.isValid = true;
        return;
    }

    // Custom validator
    if (field.validator) {
        const result = field.validator(value);
        if (result !== true) {
            state.message = result || field.message;
            return;
        }
    }
```

```
    // Pattern validation
    if (field.pattern && !field.pattern.test(value)) {
      state.message = field.message;
      return;
    }

    // Field is valid
    state.isValid = true;
  }

  validateDOB(value) {
    const dob = new Date(value);
    const today = new Date();
    const age = today.getFullYear() - dob.getFullYear();
    const monthDiff = today.getMonth() - dob.getMonth();

    // Check if date is valid
    if (isNaN(dob.getTime())) {
      return 'Please enter a valid date';
    }

    // Check if date is in the future
    if (dob > today) {
      return 'Date of birth cannot be in the future';
    }

    // Check if age is reasonable (between 0 and 150)
    if (age < 0 || (age === 0 && monthDiff < 0) || age > 150) {
      return 'Please enter a valid date of birth';
    }

    // Check minimum age (13 years)
    if (age < 13 || (age === 13 && monthDiff < 0)) {
```

```
        return 'You must be at least 13 years old';
    }

    return true;
}

validateIP(value) {
    const ipRegex = /^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$/;

    if (!ipRegex.test(value)) {
        return 'Please enter a valid IP address (e.g., 192.168.1.1)';
    }

    return true;
}

updateFieldDisplay(fieldName) {
    const input = document.getElementById(fieldName);
    const messageElement = document.getElementById(`${fieldName}Message`);
    const state = this.validationState[fieldName];

    // Remove all validation classes
    input.classList.remove('valid', 'invalid', 'warning');
    messageElement.classList.remove('valid', 'invalid', 'warning', 'info');

    if (state.isTouched || input.value.trim() !== '') {
        if (state.isValid) {
            input.classList.add('valid');
            messageElement.classList.add('valid');
            messageElement.textContent = '✓ Valid';
        } else {
            input.classList.add('invalid');
            messageElement.classList.add('invalid');
```

```
            messageElement.textContent = state.message;
        }
    } else {
        messageElement.textContent = '';
    }
}

clearFieldValidation(fieldName) {
    const input = document.getElementById(fieldName);
    const messageElement = document.getElementById(`${fieldName}Message`);

    input.classList.remove('valid', 'invalid', 'warning');
    messageElement.classList.remove('valid', 'invalid', 'warning', 'info');
    messageElement.textContent = '';
}

updateSubmitButton() {
    const allFieldsValid = Object.values(this.validationState).every(state => state.isValid);
    const hasRequiredFields = Object.values(this.validationState).some(state => state.isTouched);

    this.submitBtn.disabled = !allFieldsValid || !hasRequiredFields;
}

updateSummary() {
    const validCount = Object.values(this.validationState).filter(state => state.isValid).length;
    const invalidCount = Object.values(this.validationState).filter(state => !state.isValid && state.isTouched).length;
    const pendingCount = Object.values(this.validationState).filter(state => !state.isTouched).length;
    const totalFields = Object.keys(this.fields).length;

    let summaryHTML = `
        <div class="summary-stats">
            <div class="stat-item valid">
                <span class="stat-number">${validCount}</span>
                <span class="stat-label">Valid</span>
```

```
            </div>
            <div class="stat-item invalid">
                <span class="stat-number">${invalidCount}</span>
                <span class="stat-label">Invalid</span>
            </div>
            <div class="stat-item warning">
                <span class="stat-number">${pendingCount}</span>
                <span class="stat-label">Pending</span>
            </div>
        </div>
        <div class="field-statuses">
    `;

    Object.keys(this.fields).forEach(fieldName => {
        const state = this.validationState[fieldName];
        const displayName = this.getFieldDisplayName(fieldName);

        let statusClass = 'pending';
        let statusIcon = '?';
        let statusMessage = 'Not validated yet';

        if (state.isTouched) {
            if (state.isValid) {
                statusClass = 'valid';
                statusIcon = '✓';
                statusMessage = 'Valid';
            } else {
                statusClass = 'invalid';
                statusIcon = '✗';
                statusMessage = state.message;
            }
        }
```

```
        summaryHTML += `
            <div class="field-status">
                <span class="field-name">${displayName}:</span>
                <span class="field-status-icon ${statusClass}">${statusIcon}</span>
                <span class="field-message">${statusMessage}</span>
            </div>
        `;
    });

    summaryHTML += '</div>';
    this.summaryContent.innerHTML = summaryHTML;
}

getFieldDisplayName(fieldName) {
    const displayNames = {
        name: 'Name',
        email: 'Email',
        dob: 'Date of Birth',
        pincode: 'Pincode',
        mobile: 'Mobile Number',
        ip: 'IP Address'
    };
    return displayNames[fieldName] || fieldName;
}

clearForm() {
    // Clear all input values
    Object.keys(this.fields).forEach(fieldName => {
        const input = document.getElementById(fieldName);
        input.value = '';
        this.validationState[fieldName] = { isValid: false, message: '', isTouched: false };
        this.clearFieldValidation(fieldName);
    });
```

```
    this.updateSubmitButton();
    this.updateSummary();

    // Show success message
    this.showMessage('Form cleared successfully!', 'success');
}

handleFormSubmission() {
    // Double-check all fields are valid
    const allValid = Object.values(this.validationState).every(state => state.isValid);

    if (!allValid) {
        this.showMessage('Please fix all validation errors before submitting.', 'error');
        return;
    }

    // Collect form data
    const formData = {};
    Object.keys(this.fields).forEach(fieldName => {
        formData[fieldName] = document.getElementById(fieldName).value;
    });

    // Simulate form submission
    this.showMessage('Form submitted successfully!', 'success');
    console.log('Form Data:', formData);

    // You can add actual form submission logic here
    // For example: send data to server, redirect, etc.
}

showMessage(message, type = 'info') {
    // Remove existing message
    const existingMessage = document.querySelector('.form-message');
    if (existingMessage) {
```

```
    existingMessage.remove();
}

// Create new message
const messageDiv = document.createElement('div');
messageDiv.className = `form-message form-message-${type}`;
messageDiv.textContent = message;

// Style the message
messageDiv.style.cssText = `
    position: fixed;
    top: 20px;
    right: 20px;
    padding: 12px 20px;
    border-radius: 8px;
    color: white;
    font-weight: 600;
    z-index: 1000;
    animation: slideInRight 0.3s ease;
    max-width: 300px;
    word-wrap: break-word;
`;

// Set background color based on type
switch (type) {
    case 'success':
        messageDiv.style.background = 'linear-gradient(135deg, #27ae60 0%, #2ecc71 100%)';
        break;
    case 'error':
        messageDiv.style.background = 'linear-gradient(135deg, #e74c3c 0%, #c0392b 100%)';
        break;
    case 'warning':
        messageDiv.style.background = 'linear-gradient(135deg, #f39c12 0%, #e67e22 100%)';
        break;
```

```
            default:
                messageDiv.style.background = 'linear-gradient(135deg, #3498db 0%, #2980b9 100%)';
        }

        // Add to document
        document.body.appendChild(messageDiv);

        // Auto remove after 3 seconds
        setTimeout(() => {
            if (messageDiv.parentNode) {
                messageDiv.style.animation = 'slideOutRight 0.3s ease';
                setTimeout(() => {
                    if (messageDiv.parentNode) {
                        messageDiv.remove();
                    }
                }, 300);
            }
        }, 3000);
    }
}

// Add CSS animations
const style = document.createElement('style');
style.textContent = `
    @keyframes slideInRight {
        from {
            transform: translateX(100%);
            opacity: 0;
        }
        to {
            transform: translateX(0);
            opacity: 1;
        }
    }
```

```
    @keyframes slideOutRight {
        from {
            transform: translateX(0);
            opacity: 1;
        }
        to {
            transform: translateX(100%);
            opacity: 0;
        }
    }
`;
document.head.appendChild(style);

// Initialize the form validator when DOM is loaded
document.addEventListener('DOMContentLoaded', function() {
    const formValidator = new FormValidator();

    console.log('Real-time Form Validation initialized!');
    console.log('Features:');
    console.log('- Real-time validation on input');
    console.log('- Visual feedback with colors and icons');
    console.log('- Comprehensive validation rules');
    console.log('- Form submission only when all fields are valid');
});
```