Week 7: DOM Manipulation & Events - Lab

วัตถุประสงค์: นิสิตสามารถ

- ใช้ querySelector เลือก elements
- แก้ไข DOM โดยใช้ textContent, classList, attributes
- สร้าง event listeners และจัดการ events
- ประยุกต์ event delegation สำหรับ dynamic content
- สร้าง interactive UIs ที่ responsive

ACTIVITY 1: Interactive Toggle Menu

1.0 อ่านก่อน

- "TODO: select #menu-toggle" หมายความว่า: ใช้ document.querySelector() เพื่อเลือก element ที่มี id="menu-toggle"
- "classList" = JavaScript object ที่ใช้เพิ่ม/ลบ/เปลี่ยน CSS classes
- "toggle" = สลับสถานะ (มี ⟶ ไม่มี, ไม่มี ⟶ มี)

1.1 วัตถุประสงค์: สร้าง responsive navigation menu ที่:

- แสดง/ซ่อน menu เมื่อคลิกปุ่ม hamburger
- เปลี่ยน styling (สี, ขอบ) ตามสถานะ
- ปิด menu เมื่อคลิก link
- Responsive (mobile + desktop)

1.2 HTML Structure

นิสิต **เขียนเองทั้งหมด**

```
<!DOCTYPE html>
<html>
  <head>
    <title>Interactive Menu</title>
    <style>
      /* นิสิตเขียนเอง */
    </style>
  </head>
  <body>
    <!-- Header with toggle button -->
    <header>
      <button id="menu-toggle">☰ Menu</button>
      <nav class="menu-nav">
        <a href="#home">Home</a>
        <a href="#about">About</a>
        <a href="#services">Services</a>
        <a href="#blog">Blog</a>
        <a href="#contact">Contact</a>
      </nav>
    </header>

    <!-- Main content -->
    <main>
      <h1>Welcome</h1>
      <p>Your content here...</p>
    </main>
```

```
    <script>
      // นิสิตเขียน JavaScript ที่นี่
    </script>
  </body>
</html>
```

1.3 Requirements

HTML Elements ที่ต้องมี

- 1 button#menu-toggle (hamburger icon หรือ "Menu" text)
- 1 nav.menu-nav
  - 5+ menu links (a tags)
- 1 main content area

CSS ที่ต้องมี

- IMPORTANT: ซ่อน menu ด้วย .hidden class

  .menu-nav.hidden { display: none; }

- Base styling:
  - nav: padding, background-color, width
  - links (a): hover effect, padding, display
  - button: cursor: pointer, padding, background-color

- Responsive (mobile @media max-width 768px):
- nav should toggle visibility with .hidden class

JavaScript ที่ต้องมี

1. Select button & nav elements using querySelector()
2. Add click event listener to button
3. Inside the listener:
   - Use nav.classList.toggle("hidden") to show/hide
   - (Optional) Change button text to "Open Menu" or "Close Menu"
   - (Optional) Close menu when user clicks a link

1.4 Code Example

```
// 1. Select the button and nav elements
const button = document.querySelector("#menu-toggle");
const nav = document.querySelector(".menu-nav");

// 2. Add click event listener to the button
button.addEventListener("click", () => {
  // 3. Toggle the 'hidden' class on nav
  nav.classList.toggle("hidden");

  // 4. Optional: Change button text based on state
  if (nav.classList.contains("hidden")) {
    button.textContent = "☰ Open Menu";
  } else {
    button.textContent = "✕ Close Menu";
  }
});

// 5. Optional: Close menu when a link is clicked
const links = document.querySelectorAll(".menu-nav a");
links.forEach((link) => {
  link.addEventListener("click", () => {
    nav.classList.add("hidden");
```

```
    button.textContent = "☰ Open Menu"; // Reset button
  });
});
```

1.5 ขั้นตอนการทำ

Step 1: เขียน HTML

- Header ที่มี button + nav
- 5 links ใน nav (Home, About, Services, Blog, Contact)
- Main content area

Step 2: เขียน CSS

- nav styling: width, background, position
- button styling: padding, background, cursor
- .hidden { display: none; } class
- Responsive breakpoint (768px)

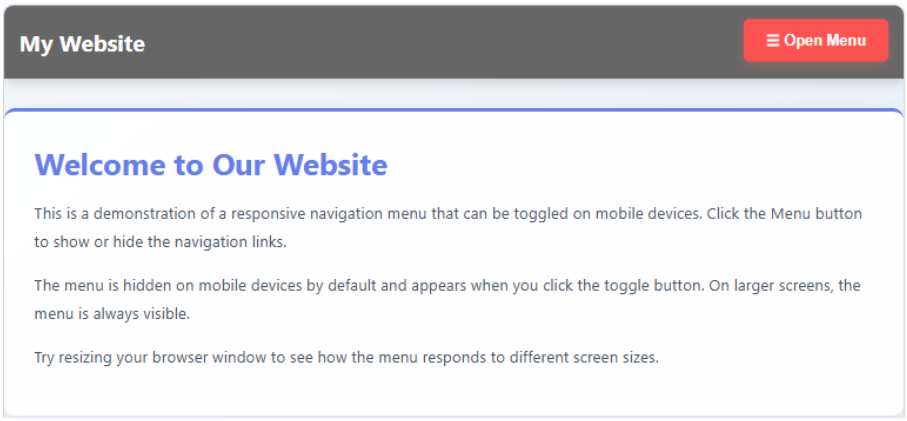
Step 3: เขียน JavaScript

- querySelector สำหรับ button & nav
- addEventListener for click
- classList.toggle()
- ทดสอบ: click button, menu ปรากฏ/หายไป

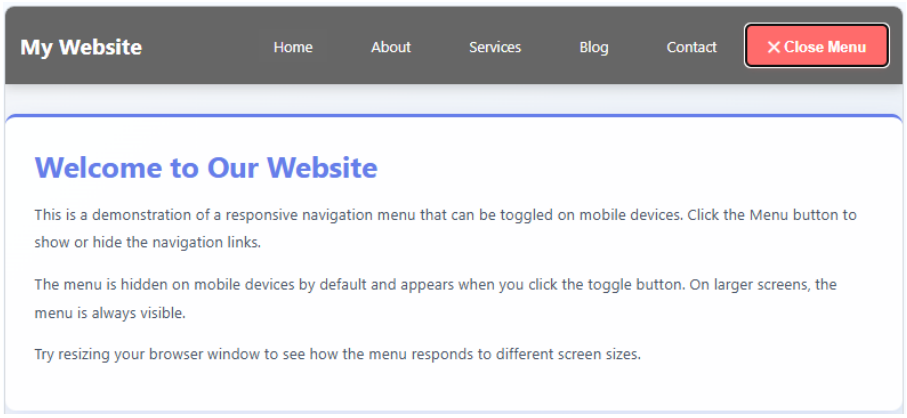
Step 4: ทดสอบ + Refine

- Click button ระหว่าง menu ปรากฏ/หายไป
- Menu ปิดเมื่อคลิก link
- Mobile responsive

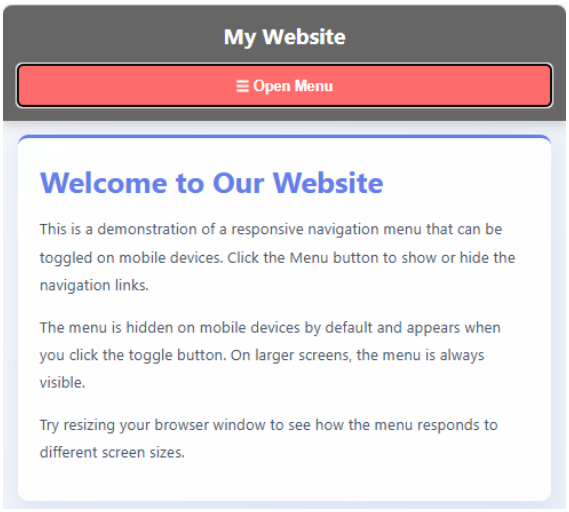
ตัวอย่างหน้าเว็บที่ต้องการ



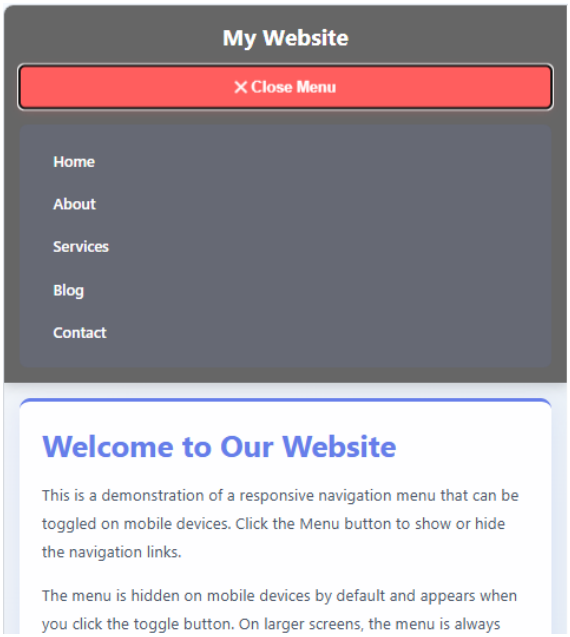
คลิกปุ่ม Open Menu



ย่อหน้าต่างเป็นแนวตั้ง



คลิกปุ่ม Open Menu



1.6 ตอบคำถามต่อไปนี้เพื่อวัดความเข้าใจ:

1. **querySelector vs querySelectorAll**: อะไรคือความแตกต่าง
    - querySelector() ใช้เมื่อไร
    - querySelectorAll() ใช้เมื่อไร
2. **classList.toggle()**: nav.classList.toggle('hidden') ทำงานอย่างไร
    - ถ้า nav มี class 'hidden' แล้ว toggle() จะทำอะไร
    - ถ้า nav ไม่มี class 'hidden' แล้ว toggle() จะทำอะไร
3. **event.target**: เมื่อคลิกปุ่ม event.target จะชี้ไปที่อะไร
4. **remove() vs toggle()**: เมื่อคลิก link ควร

    ใช้ nav.classList.remove('hidden') หรือ nav.classList.toggle('hidden')? ทำไม
5. **display: none**: .hidden { display: none; } ทำไมจึงจำเป็น? วิธีอื่นเพื่อซ่อน element ได้หรือไม่

ACTIVITY 2: Todo List with Event Delegation

2.1 วัตถุประสงค์ สร้าง todo list ที่

- เพิ่ม todo จากการพิมพ์ + กด Add
- ลบ todo ด้วยการคลิกปุ่ม delete
- Mark complete (toggle class) ด้วย checkbox
- ใช้ event delegation (เพราะ items เป็น dynamic)

2.2 HTML Structure

```
<!DOCTYPE html>
<html>
  <head>
    <title>Todo List</title>
    <style>
      /* นิสิตเขียนเอง */
    </style>
  </head>
  <body>
    <div class="todo-container">
      <!-- Input field -->
      <!-- Add button -->
      <!-- Todo list (ul/ol) -->
    </div>

    <script>
      // นิสิตเขียน JavaScript
    </script>
  </body>
</html>
```

2.3 Requirements

HTML Elements

- 1 input#new-todo (placeholder: "Enter a new todo...")
- 1 button#add-btn (text: "Add")
- 1 ul#todo-list
  - 2-3 initial li.todo-item
  - Each li contains:
    - span with todo text
    - button.delete-btn (text: "✕" or "Delete")

CSS

- todo-container: width, max-width, margin
- input: padding, border, border-radius
- button: padding, background, cursor, hover
- li: padding, border-bottom, flex display
- .completed li: text-decoration: line-through, opacity: 0.6
- .delete-btn: small, red background, hover effect

JavaScript

1. Select input, button, list
2. addEventListener on button
   - Get input value
   - Create li element
   - Add to list
   - Clear input
3. Event delegation on ul

- Check if clicked target is delete button
- Remove parent li

2.4 Code Example

```
// 1. Select elements
const input = // TODO: #new-todo
const addBtn = // TODO: #add-btn
const todoList = // TODO: #todo-list

// 2. Add event listener on button
addBtn.addEventListener('click', () => {
  const text = input.value.trim();

  // Validate: not empty
  if (!text) return;

  // Create new li
  const li = document.createElement('li');
  li.classList.add('todo-item');
  li.innerHTML = `
    <span>${text}</span>
    <button class="delete-btn">X</button>
  `;

  // Add to list
  todoList.appendChild(li);

  // Clear input
  input.value = '';
  input.focus();
});

// 3. Event delegation on list
todoList.addEventListener('click', (event) => {
  // Check if delete button clicked
  if (event.target.classList.contains('delete-btn')) {
    const li = event.target.parentElement;
    li.remove();
  }
});

// 4. Optional: Mark complete
todoList.addEventListener('change', (event) => {
  if (event.target.type === 'checkbox') {
    const li = event.target.parentElement;
    li.classList.toggle('completed');
  }
});
```

2.5 ขั้นตอนการทำ

Step 1: เขียน HTML

- Input field + Add button
- ul#todo-list
- 2-3 initial todos
- Each todo: span + delete button

Step 2: เขียน CSS

- Container styling
- Input/button styling

- Todo item styling (flex, padding)
- Delete button styling
- .completed state

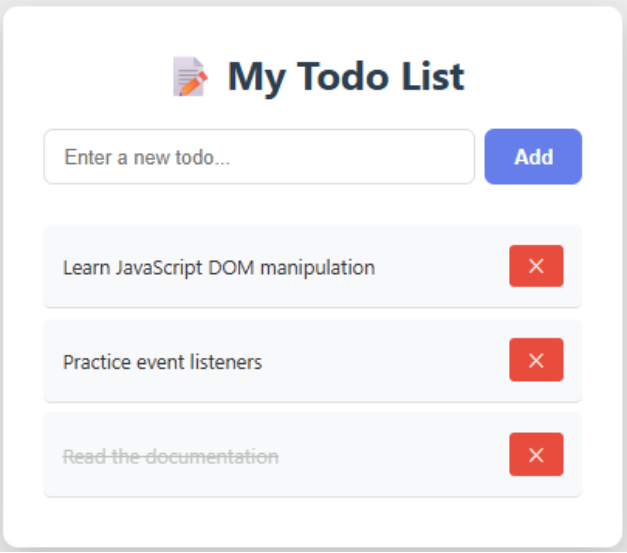
Step 3: เขียน JavaScript

- Select elements
- addEventListener on button → create + append li
- Event delegation on ul → handle delete
- Validate input (not empty)
- Clear input after add

Step 4: ทดสอบ

- Type + click Add → todo appears
- Click delete → todo removed
- Click todo item to show/remove strikethrough

คลิกปุ่ม **Open Menu**



2.6 ตอบคำถามต่อไปนี้เพื่อวัดความเข้าใจ

1. **Event Delegation**: ทำไมเราต้องใช้ event delegation สำหรับ todo list? (เคล็ดลับ: todos เป็น dynamic elements)
2. **event.target**: ใน event delegation event.target ชี้ไปที่อะไร? element ที่คลิก หรือ element ที่มี listener
3. **innerHTML**: ทำไมต้องใช้ li.innerHTML = '...<button>...</button>' แทน li.textContent = 'text'
4. **New Elements**: ถ้าเพิ่ม todo ใหม่หลังจาก page โหลด delete button ยังใช้งานได้ไหม ทำไม
5. **if (!text) return**: Code นี้ทำอะไร? ถ้าไม่มี code นี้จะเกิดอะไร
6. **trim()**: ทำไมต้องใช้ .trim() ก่อนตรวจสอบว่าว่างไหม

ACTIVITY 3: Form Validation & Feedback

3.1 วัตถุประสงค์: สร้าง registration form ที่

- Validate เข้ามาแบบ real-time (ขณะพิมพ์)
- แสดง error messages ชัดเจน
- Visual feedback (สีแดง/เขียว)
- Disable submit ถ้า invalid

3.2 HTML Structure

```
<!DOCTYPE html>
<html>
  <head>
    <title>Registration Form</title>
    <style>
      /* นิสิตเขียนเอง */
    </style>
  </head>
  <body>
    <form id="register-form">
      <!-- Name field -->
      <!-- Email field -->
      <!-- Password field -->
      <!-- Confirm password field -->
      <!-- Submit button -->

      <!-- Error messages (initially hidden) -->
    </form>

    <script>
      // นิสิตเขียน JavaScript
    </script>
  </body>
</html>
```

3.3 Requirements

HTML Elements

- 1 form#register-form
- input[name="name"] + span.error-name
- input[name="email"] + span.error-email
- input[name="password"] + span.error-password
- input[name="confirm-password"] + span.error-confirm
- button[type="submit"] (initially disabled)

CSS

- Form styling: width: 400px, centered
- Input styling: padding, border, margin
- .error class: border-color: red, background: #ffe0e0
- .valid class: border-color: green, background: #e0ffe0
- .error-message: color: red, font-size: small, display: none
- .error-message.show: display: block
- button[disabled]: opacity: 0.5, cursor: not-allowed

JavaScript

1. Select all inputs + error spans
2. Validation functions:
   - validateName(value) → min 3 chars
   - validateEmail(value) → includes @

- validatePassword(value) → min 8 chars
- validateConfirmPassword(pwd, confirm) → match

3. Event listeners on inputs:
- input event: validate on change
- Update UI (add/remove error class)
- Show/hide error messages
- Enable/disable submit button

4. Submit handler:
- event.preventDefault()
- Validate all fields
- If valid: show success message or submit

3.4 Code Example

```
// 1. Select elements
const form = // TODO: #register-form
const nameInput = form.querySelector('input[name="name"]');
const emailInput = form.querySelector('input[name="email"]');
const passwordInput = form.querySelector('input[name="password"]');
const confirmInput = form.querySelector('input[name="confirm-password"]');
const submitBtn = form.querySelector('button[type="submit"]');

const nameError = form.querySelector('.error-name');
const emailError = form.querySelector('.error-email');
const passwordError = form.querySelector('.error-password');
const confirmError = form.querySelector('.error-confirm');

// 2. Validation functions
function validateName(value) {
  return value.trim().length >= 3;
}

function validateEmail(value) {
  return value.includes('@') && value.includes('.');
}

function validatePassword(value) {
  return value.length >= 8;
}

function validateConfirmPassword(pwd, confirm) {
  return pwd === confirm;
}

// 3. Check if form is valid
function isFormValid() {
  return (
    validateName(nameInput.value) &&
    validateEmail(emailInput.value) &&
    validatePassword(passwordInput.value) &&
    validateConfirmPassword(passwordInput.value, confirmInput.value)
  );
}

// 4. Update UI
function updateUI() {
  // Name validation
  if (validateName(nameInput.value)) {
```

```
    nameInput.classList.remove('error');
    nameInput.classList.add('valid');
    nameError.classList.remove('show');
  } else {
    nameInput.classList.add('error');
    nameInput.classList.remove('valid');
    nameError.classList.add('show');
    nameError.textContent = 'Name must be at least 3 characters';
  }

  // Email validation
  if (validateEmail(emailInput.value)) {
    emailInput.classList.remove('error');
    emailInput.classList.add('valid');
    emailError.classList.remove('show');
  } else {
    emailInput.classList.add('error');
    emailInput.classList.remove('valid');
    emailError.classList.add('show');
    emailError.textContent = 'Please enter valid email';
  }

  // Password validation
  if (validatePassword(passwordInput.value)) {
    passwordInput.classList.remove('error');
    passwordInput.classList.add('valid');
    passwordError.classList.remove('show');
  } else {
    passwordInput.classList.add('error');
    passwordInput.classList.remove('valid');
    passwordError.classList.add('show');
    passwordError.textContent = 'Password must be at least 8 characters';
  }

  // Confirm password validation
  if (validateConfirmPassword(passwordInput.value, confirmInput.value)) {
    confirmInput.classList.remove('error');
    confirmInput.classList.add('valid');
    confirmError.classList.remove('show');
  } else {
    confirmInput.classList.add('error');
    confirmInput.classList.remove('valid');
    confirmError.classList.add('show');
    confirmError.textContent = 'Passwords do not match';
  }

  // Enable/disable submit button
  submitBtn.disabled = !isFormValid();
}

// 5. Event listeners on inputs
nameInput.addEventListener('input', updateUI);
emailInput.addEventListener('input', updateUI);
passwordInput.addEventListener('input', updateUI);
confirmInput.addEventListener('input', updateUI);

// 6. Form submit
form.addEventListener('submit', (event) => {
  event.preventDefault();

  if (isFormValid()) {
```

```
    alert('Form submitted successfully!');
    form.reset();
    updateUI();
  }
});

// 7. Initial state
updateUI();
```

3.5 ขั้นตอนการทำ

Step 1: เขียน HTML

- Form ที่มี 4 inputs (name, email, password, confirm-password)
- Error message span สำหรับแต่ละ field
- Submit button (initially disabled)

Step 2: เขียน CSS

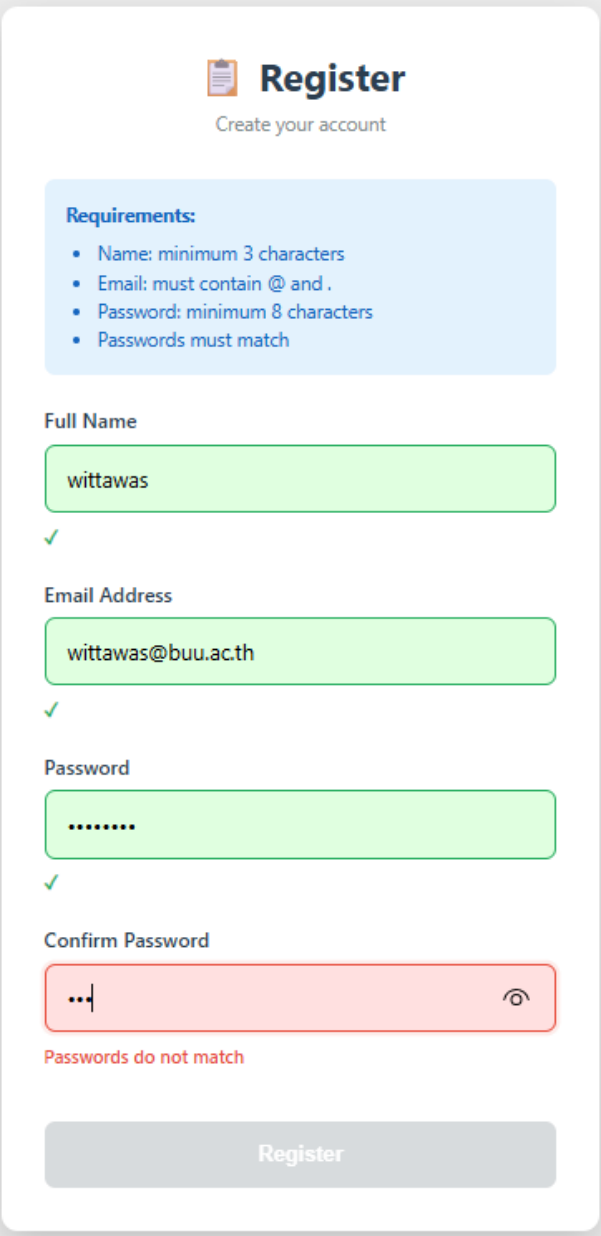
- Form styling: centered, readable
- Input styling: padding, border
- Error state: red border + light red background
- Valid state: green border + light green background
- Error messages: red text, initially hidden

Step 3: เขียน JavaScript

- Validation functions (name, email, password, confirm)
- isFormValid() function
- updateUI() function (update classes + messages)
- Event listeners on inputs → call updateUI()
- Submit handler + preventDefault()

Step 4: ทดสอบ + Refine

- Type name → error/valid state ปรากฏ
- Type email → validate @ .
- Password: min 8, confirm match
- Submit disabled ถ้า invalid, enabled if valid



3.6 ตอบคำถามต่อไปนี้เพื่อวัดความเข้าใจ

1. **'input' vs 'change' event**: ต่างกันอย่างไร? ทำไมต้องใช้ 'input' สำหรับ real-time validation
2. **preventDefault()**: ทำอะไร? ถ้าไม่มี code นี้จะเกิดอะไร
3. **classList operations**: ทำไมต้อง remove('error') และ add('valid') ทั้งคู่? ไม่ได้มี add('valid') คนเดียวไหม
4. **disabled attribute**: submitBtn.disabled = true หมายความว่าอะไร? สามารถจัดการผ่าน JavaScript ได้ไหม
5. **Email validation**: value.includes('@') && value.includes('.') กรณีไหนจะผิด
    - "user@gmail..com" (2 dots) - ผ่านหรือไม่
    - "user.gmail.com" (ไม่มี @) - ผ่านหรือไม่
6. **DRY principle**: ทำไม code ที่เพิ่ม listener 4 ครั้ง (สำหรับ 4 inputs) ทำให้ code ยาว
7. **confirm-password field**: ทำไมต้องมี? ถ้าลบออกจะเกิดปัญหาไหม

📚 **Resources**

- MDN: querySelector
- MDN: addEventListener

- MDN: classList
- MDN: Events
- JavaScript.info

Summary
นิสิต learned & practiced:

- querySelector สำหรับเลือก elements
- classList สำหรับการเปลี่ยน styling
- addEventListener สำหรับ events
- event.target สำหรับเข้าถึง clicked element
- Event delegation สำหรับ dynamic content
- Real-time validation & visual feedback
- Interactive UI patterns (toggle, list, form)

O]O