เอกสารประกอบการสอน

ENGCE301 การออกแบบและพัฒนาซอฟต์แวร์

Software Design and Development

สัปดาห์ที่ 4

หัวข้อ: Software Design Principles & UX/UI + Responsive HTML/CSS

ชื่อบทเรียน

1. หลักการออกแบบซอฟต์แวร์เบื้องต้น (Software Design Principles: SOLID, Coupling & Cohesion)
2. การออกแบบที่เน้นผู้ใช้เป็นศูนย์กลาง (User-Centred Design, Personas & User Journey Map)
3. การออกแบบ Responsive Design ด้วย Flexbox และ Grid

วัตถุประสงค์การสอน

เมื่อจบบทเรียนนี้ นักศึกษาจะสามารถ:

บทเรียนที่ 1: หลักการออกแบบซอฟต์แวร์เบื้องต้น

- อธิบายหลักการ SOLID ในการออกแบบซอฟต์แวร์ได้
- เข้าใจแนวคิด Coupling และ Cohesion พร้อมประยุกต์ใช้ในการออกแบบได้
- ระบุปัญหาการออกแบบที่ไม่ดีและเสนอแนวทางแก้ไขได้

บทเรียนที่ 2: การออกแบบที่เน้นผู้ใช้เป็นศูนย์กลาง

- อธิบายหลักการ User-Centred Design (UCD) ได้
- สร้าง Personas และ User Journey Map สำหรับระบบของตนเองได้
- วิเคราะห์ความต้องการของผู้ใช้และนำมาใช้ในการออกแบบระบบได้

บทเรียนที่ 3: การออกแบบ Responsive Design

- ออกแบบและพัฒนา UI ที่รองรับหลายขนาดหน้าจอด้วย CSS Flexbox ได้
- ใช้ CSS Grid ในการจัดวาง Layout ที่ซับซ้อนได้
- ปรับแต่งหน้าเว็บของ Term Project ให้เป็น Responsive ได้

ทบทวนสัปดาห์ที่แล้ว

ในสัปดาห์ที่ 3 เราได้เรียนรู้เกี่ยวกับ:

- Agile Requirements และการเขียน User Stories ตามหลัก INVEST
- การสร้าง Use Case Diagram และการเขียน Use Case Scenario
- พื้นฐาน CSS: Box Model, Selectors และการจัด Layout
- Workshop: สร้าง Product Backlog และ Use Case Diagram สำหรับ Term Project

ในสัปดาห์นี้ เราจะต่อยอดความรู้เรื่องการออกแบบ โดยเน้นที่หลักการออกแบบซอฟต์แวร์ที่ดี การออกแบบที่คำนึงถึงผู้ใช้ และการสร้าง UI ที่ตอบสนองต่อขนาดหน้าจอต่างๆ เพื่อให้ Term Project ของเรามีคุณภาพและใช้งานได้ดีบนทุกอุปกรณ์

บทเรียนที่ 1: หลักการออกแบบซอฟต์แวร์เบื้องต้น

Software Design Principles: SOLID, Coupling & Cohesion

1.1 ความสำคัญของการออกแบบซอฟต์แวร์ที่ดี

การออกแบบซอฟต์แวร์ที่ดีเป็นรากฐานสำคัญของการพัฒนาซอฟต์แวร์ที่มีคุณภาพ ช่วยให้โค้ดมีความยืดหยุ่น บำรุงรักษาง่าย และสามารถขยายการทำงานได้ในอนาคต หลักการออกแบบที่ดีจะช่วยลดความซับซ้อน ลดข้อผิดพลาด และทำให้ทีมทำงานร่วมกันได้อย่างมีประสิทธิภาพ

คุณสมบัติของโค้ดที่ออกแบบมาอย่างดี:

- อ่านเข้าใจง่าย (Readable) - ผู้อื่นสามารถเข้าใจโค้ดได้โดยไม่ต้องใช้เวลามาก
- บำรุงรักษาง่าย (Maintainable) - แก้ไขและปรับปรุงได้สะดวก
- ขยายได้ (Extensible) - เพิ่มฟีเจอร์ใหม่ได้โดยไม่ต้องแก้ไขโค้ดเดิมมาก
- นำกลับมาใช้ได้ (Reusable) - ส่วนประกอบสามารถนำไปใช้ในโปรเจกต์อื่นได้
- ทดสอบได้ (Testable) - เขียน Unit Test ได้ง่าย

1.2 หลักการ SOLID

SOLID เป็นชุดหลักการออกแบบซอฟต์แวร์ 5 ข้อที่ช่วยให้โค้ดมีคุณภาพสูง ถูกพัฒนาโดย Robert C. Martin (Uncle Bob) ซึ่งเป็นพื้นฐานสำคัญของ Object-Oriented Programming และใช้ได้กับหลายภาษาโปรแกรม

ตารางสรุปหลักการ SOLID

แนวคิด	คำอธิบาย	ประโยชน์
S - Single Responsibility	คลาสหรือฟังก์ชันควรมีหน้าที่เดียว มีเหตุผลเดียวที่จะต้องเปลี่ยนแปลง	แก้ไขง่าย ไม่กระทบส่วนอื่น ทดสอบง่าย
O - Open/Closed	เปิดให้ขยาย (Extension) แต่ปิดไม่ให้แก้ไขโค้ดเดิม (Modification)	เพิ่มฟีเจอร์ใหม่โดยไม่กระทบโค้ดเดิม ลดความเสี่ยงในการแก้ไข

แนวคิด	คำอธิบาย	ประโยชน์
L - Liskov Substitution	คลาสลูกต้องสามารถแทนคลาสแม่ได้โดยไม่ทำให้โปรแกรมทำงานผิดพลาด	การ Inherit ที่ถูกต้อง ลดข้อผิดพลาดจาก Polymorphism
I - Interface Segregation	ไม่ควรบังคับให้คลาสต้อง Implement เมธอดที่ไม่ได้ใช้	Interface เล็กและเฉพาะเจาะจง ลดการพึ่งพาที่ไม่จำเป็น
D - Dependency Inversion	ส่วนที่ซับซ้อนไม่ควรพึ่งพาส่วนที่เรียบง่าย ทั้งสองควรพึ่งพา Abstraction	แยกส่วนได้ชัดเจน ทดสอบและแก้ไขง่าย เปลี่ยน Implementation ได้

ตัวอย่างการประยุกต์ใช้ SOLID:

ตัวอย่างที่ 1: Single Responsibility Principle (SRP)

❌ **แบบไม่ดี: คลาสทำหลายอย่าง**

```
class User {   constructor(name, email) {
this.name = name;     this.email = email;   }      //
หน้าที่ที่ 1: จัดการข้อมูล   save() { /* save to database */ }
// หน้าที่ที่ 2: ส่งอีเมล   sendEmail() { /* send welcome email
*/ }       // หน้าที่ที่ 3: สร้างรายงาน   generateReport() { /*
create user report */ } }
```

✅ **แบบดี: แยกหน้าที่ออกเป็นคลาสต่าง ๆ**

```
// คลาสจัดการข้อมูลผู้ใช้ class User {   constructor(name, email)
{     this.name = name;     this.email = email;   } }
// คลาสจัดการฐานข้อมูล class UserRepository {   save(user) {
/* save to database */ } }   // คลาสจัดการอีเมล class
EmailService {   sendWelcomeEmail(user) { /* send
email */ } }   // คลาสสร้างรายงาน class ReportGenerator {
generateUserReport(user) { /* create report */ } }
```

ตัวอย่างที่ 2: Open/Closed Principle (OCP)

❌ **แบบไม่ดี: แก้ไขโค้ดเดิมเมื่อเพิ่มประเภทใหม่**

```
class PaymentProcessor {   processPayment(method, amount) {     if (method === 'credit') {        //
โค้ดจ่ายด้วยบัตรเครดิต      } else if (method === 'paypal') {
// โค้ดจ่ายด้วย PayPal      }      // ถ้าเพิ่มวิธีใหม่ ต้องแก้ if-else นี้
} }
```

✅ **แบบดี: ขยายได้โดยไม่ต้องแก้โค้ดเดิม**

```
// สร้าง interface/base class class PaymentMethod {
process(amount) {      throw new Error('Must implement
process method');    } }   // แต่ละวิธีเป็นคลาสแยก class
CreditCardPayment extends PaymentMethod {
process(amount) { /* จ่ายด้วยบัตร */ } }  class
PayPalPayment extends PaymentMethod {
process(amount) { /* จ่ายด้วย PayPal */ } }   //
เพิ่มวิธีใหม่โดยไม่ต้องแก้โค้ดเดิม class CryptoPayment extends
PaymentMethod {   process(amount) { /* จ่ายด้วย Crypto */
} }   // Processor ใช้งาน class PaymentProcessor {
processPayment(paymentMethod, amount) {
paymentMethod.process(amount);    } }
```

1.3 Coupling และ Cohesion

Coupling และ Cohesion
เป็นแนวคิดสำคัญในการออกแบบซอฟต์แวร์ที่ช่วยให้เราประเมินคุณภาพของโครงสร้างโค้ด
เป้าหมายคือต้องการ Low Coupling (ความพึ่งพาต่ำ) และ High Cohesion (ความเกี่ยวข้องสูง)

ตารางเปรียบเทียบ Coupling และ Cohesion

Coupling (ความพึ่งพากัน)	Cohesion (ความเกี่ยวข้องกัน)
ความหมาย	ระดับที่โมดูลหรือคลาสพึ่งพากัน | ระดับที่ฟังก์ชันภายในโมดูลเดียวกันเกี่ยวข้องกัน
เป้าหมาย	ต้องการ Low Coupling (ต่ำ) | ต้องการ High Cohesion (สูง)
ข้อดี	แก้ไขส่วนหนึ่งไม่กระทบส่วนอื่น | ฟังก์ชันทำงานที่เกี่ยวข้องอยู่ด้วยกัน
ข้อดี (ต่อ)	ทดสอบแยกส่วนได้ | เข้าใจและดูแลรักษาง่าย

ตัวอย่างแย่	คลาส A เรียกใช้คลาส B, C, D โดยตรง | ฟังก์ชันในคลาสเดียวกันทำงานไม่เกี่ยวข้องกัน

ตัวอย่างการออกแบบ Low Coupling & High Cohesion:

❌ High Coupling & Low Cohesion: โมดูลพึ่งพากันมาก ฟังก์ชันทำงานไม่เกี่ยวข้อง

```
class OrderManager {   constructor() {     this.db = new Database(); // พึ่งพาโดยตรง     this.emailer = new EmailService(); // พึ่งพาโดยตรง     this.printer = new Printer(); // พึ่งพาโดยตรง    }     createOrder(data) { /* สร้างออเดอร์ */ }   sendEmail() { /* ส่งอีเมล */ }   printReport() { /* พิมพ์รายงาน */ }   calculateTax() { /* คำนวณภาษี */ }   // ฟังก์ชันทำงานไม่เกี่ยวข้องกัน (Low Cohesion) }
```

✅ Low Coupling & High Cohesion: แยกโมดูล ฟังก์ชันเกี่ยวข้องกัน

```
// แยกความรับผิดชอบชัดเจน แต่ละคลาสทำงานที่เกี่ยวข้องกัน (High Cohesion) class Order {   constructor(data) { this.data = data; }   validate() { /* ตรวจสอบข้อมูล */ }   calculate() { /* คำนวณราคา */ } }  class OrderRepository {   save(order) { /* บันทึกลง DB */ }   findById(id) { /* ค้นหา */ } } class OrderNotification {   sendConfirmation(order) { /* ส่งอีเมลยืนยัน */ } }  // ใช้ Dependency Injection ลด Coupling class OrderService {   constructor(repository, notification) {     this.repository = repository; // ไม่ต้อง new โดยตรง     this.notification = notification;   }   createOrder(data) {     const order = new Order(data);     order.validate();     order.calculate();     this.repository.save(order);     this.notification.sendConfirmation(order);   } }
```

1.4 การระบุและแก้ไขปัญหาการออกแบบที่ไม่ดี (Code Smells)

Code Smells คือสัญญาณเตือนที่บ่งบอกว่าโค้ดอาจมีปัญหาในการออกแบบ แม้โค้ดจะทำงานได้ แต่อาจทำให้บำรุงรักษายาก หรือเกิดข้อผิดพลาดในอนาคต การรู้จัก Code Smells ช่วยให้เราปรับปรุงโค้ดได้ตั้งแต่เนิ่นๆ

ตาราง Code Smells ที่พบบ่อยและวิธีแก้ไข

แนวคิด	คำอธิบาย	ประโยชน์
Long Method	ฟังก์ชันยาวเกินไป ทำหลายอย่างในที่เดียว	แยกเป็นฟังก์ชันย่อยๆ ตั้งชื่อให้สื่อความหมาย
Large Class	คลาสมี field และ method มากเกินไป	แยกเป็นหลายคลาสตาม responsibility
Duplicate Code	โค้ดซ้ำหลายที่ แก้ไขต้องแก้หลายจุด	สร้างฟังก์ชันหรือคลาสใช้ร่วมกัน
Dead Code	โค้ดที่ไม่มีการเรียกใช้แล้ว	ลบโค้ดเก่าออก ใช้ Git เก็บประวัติ
Magic Numbers	ใช้ตัวเลขโดยตรงโดยไม่อธิบาย	สร้าง constant ที่มีชื่อสื่อความหมาย

ตัวอย่างการแก้ไข Code Smell:

❌ Long Method & Magic Numbers

```
function processOrder(order) {   let total = 0;   for (let item of order.items) {     total += item.price * item.quantity;   }     if (total > 1000) {     total = total * 0.9; // ส่วนลด 10% (Magic Number)   }     let tax = total * 0.07; // ภาษี 7% (Magic Number)   total = total + tax;     // ส่งอีเมล sendEmail(order.customer.email, "ยืนยันออเดอร์...");     // บันทึกลง DB   database.save(order);     return total; // ฟังก์ชันยาว ทำหลายอย่าง มี Magic Numbers }
```

✅ Refactored: แยกฟังก์ชัน ใช้ Constants

```
// Constants const DISCOUNT_THRESHOLD = 1000; const DISCOUNT_RATE = 0.10; const TAX_RATE = 0.07;  // แยกการคำนวณ function calculateSubtotal(items) {   return items.reduce((sum, item) =>     sum + item.price * item.quantity, 0); }  function applyDiscount(amount) {   if (amount > DISCOUNT_THRESHOLD) {     return amount * (1 - DISCOUNT_RATE);   }   return amount; } function addTax(amount) {   return amount * (1 + TAX_RATE); }  // ฟังก์ชันหลักสั้นและชัดเจน function
```

```
processOrder(order, emailService, repository) {
const subtotal = calculateSubtotal(order.items);
const afterDiscount = applyDiscount(subtotal);
const total = addTax(afterDiscount);
emailService.sendConfirmation(order);
repository.save(order);      return total; }
```

💡 **สรุปบทเรียนที่ 1:**

- SOLID ช่วยให้โค้ดมีคุณภาพ ยืดหยุ่น และบำรุงรักษาง่าย
- Low Coupling & High Cohesion ทำให้โมดูลแยกส่วนชัดเจน
- Refactoring เป็นกระบวนการปรับปรุงโค้ดให้ดีขึ้นอย่างต่อเนื่อง

บทเรียนที่ 2: การออกแบบที่เน้นผู้ใช้เป็นศูนย์กลาง

User-Centred Design (UCD), Personas & User Journey Map

2.1 หลักการ User-Centred Design (UCD)

User-Centred Design (UCD)
เป็นแนวทางในการออกแบบที่เน้นความต้องการและประสบการณ์ของผู้ใช้เป็นหลัก
โดยมีผู้ใช้เข้ามามีส่วนร่วมตลอดกระบวนการออกแบบและพัฒนา
เพื่อให้มั่นใจว่าผลิตภัณฑ์สุดท้ายตอบโจทย์ผู้ใช้งานจริง

หลักการสำคัญของ UCD:

- เน้นผู้ใช้และงานที่ทำ (Focus on users and their tasks)
- วัดผลจากการใช้งานจริง (Empirical measurement)
- ออกแบบซ้ำและปรับปรุง (Iterative design)
- ทีมงานหลากหลายทักษะ (Multidisciplinary team)

กระบวนการ UCD

ขั้นตอน	กิจกรรม
1. เข้าใจบริบท	ศึกษาผู้ใช้ สภาพแวดล้อม และปัญหาที่ต้องแก้
2. กำหนดความต้องการ	สัมภาษณ์ สำรวจ สังเกตการใช้งาน สร้าง Personas
3. ออกแบบทางเลือก	สร้าง Wireframe, Mockup, Prototype หลายแบบ
4. ประเมินผล	Usability Testing กับผู้ใช้จริง รับ Feedback
5. ปรับปรุง	นำ Feedback มาแก้ไข และวนกลับไปขั้นตอนที่ 3

ทำไมต้องใช้ UCD?

- ลดความเสี่ยงในการพัฒนาผลิตภัณฑ์ที่ไม่ตรงความต้องการ
- เพิ่มความพึงพอใจและการใช้งานจริงของผู้ใช้
- ประหยัดเวลาและต้นทุนจากการแก้ไขในภายหลัง
- สร้างความได้เปรียบทางการแข่งขันจาก UX ที่ดี

2.2 Personas: การสร้างตัวแทนผู้ใช้

Personas คือตัวแทนผู้ใช้สมมติที่สร้างขึ้นจากข้อมูลจริงของผู้ใช้เป้าหมาย มีชื่อ อายุ อาชีพ เป้าหมาย และพฤติกรรม ช่วยให้ทีมพัฒนาเข้าใจผู้ใช้และตัดสินใจออกแบบได้ดีขึ้น โดยทั่วไปแต่ละโปรเจกต์จะมี 3-5 Personas ที่แตกต่างกัน

องค์ประกอบของ Persona ที่ดี:

- ข้อมูลพื้นฐาน: ชื่อ อายุ อาชีพ รูปภาพ
- พื้นหลัง: การศึกษา ประสบการณ์ ทักษะทางเทคโนโลยี
- เป้าหมาย: สิ่งที่ต้องการบรรลุจากการใช้ระบบ
- ปัญหา/อุปสรรค: จุดเจ็บ (Pain Points) ที่พบในปัจจุบัน
- พฤติกรรม: วิธีการทำงาน นิสัยการใช้เทคโนโลยี
- Quote: คำพูดที่สะท้อนทัศนคติของ Persona

ตัวอย่าง Persona สำหรับระบบจองห้องประชุม

Persona 1: นักธุรกิจที่ยุ่ง (Busy Executive)

ชื่อ	คุณอรุณ ใจดี, 42 ปี
อาชีพ	ผู้จัดการฝ่ายการตลาด
ทักษะเทคโนโลยี	ปานกลาง - ใช้สมาร์ทโฟนและอีเมลเป็นประจำ
เป้าหมาย	ต้องการจองห้องประชุมได้รวดเร็ว พร้อมดูความพร้อมของห้องแบบเรียลไทม์
ปัญหา	เวลาน้อย ต้องการจองผ่านมือถือ บ่อยครั้งห้องถูกจองซ้ำ
Quote	"ฉันต้องการระบบที่จองได้เร็วและแม่นยำ ไม่อยากเสียเวลาโทรศัพท์"

Persona 2: เจ้าหน้าที่ประสานงาน (Coordinator)

ชื่อ	คุณมานี รักงาน, 28 ปี
อาชีพ	เจ้าหน้าที่ธุรการ
ทักษะเทคโนโลยี	สูง - ใช้คอมพิวเตอร์และซอฟต์แวร์สำนักงานคล่องแคล่ว
เป้าหมาย	จัดการการจองห้องทั้งหมด ตรวจสอบสถานะ จัดอุปกรณ์
ปัญหา	จัดการหลายคำขอพร้อมกัน ต้องตรวจสอบความขัดแย้งของเวลา
Quote	"ฉันต้องเห็นภาพรวมการจองทั้งหมดและแก้ไขได้ง่าย"

ประโยชน์ของ Personas:

- ช่วยให้ทีมมีความเข้าใจผู้ใช้ตรงกัน ลดการตีความผิด
- ใช้ในการตัดสินใจออกแบบ เช่น "นักธุรกิจที่ยุ่งต้องการฟีเจอร์นี้หรือไม่?"
- จัดลำดับความสำคัญของฟีเจอร์ตามความต้องการของแต่ละ Persona
- ทดสอบ Usability โดยสวมบทบาทเป็น Persona

2.3 User Journey Map: การวาดเส้นทางผู้ใช้

User Journey Map เป็นการแสดงภาพเส้นทางที่ผู้ใช้ต้องผ่านเพื่อบรรลุเป้าหมาย โดยแสดงทุกจุดสัมผัส (Touchpoints) อารมณ์ ความคิด และ Pain Points ตลอดการใช้งาน
ช่วยให้เห็นโอกาสในการปรับปรุงประสบการณ์ผู้ใช้

องค์ประกอบของ User Journey Map:

- Persona: ระบุว่าใช้ Persona ไหนในการวาง Journey
- Scenario: สถานการณ์หรือเป้าหมายที่ผู้ใช้ต้องการบรรลุ
- Stages: ขั้นตอนต่างๆ ของการทำงาน (เช่น รับรู้ ⟶ พิจารณา ⟶ ตัดสินใจ ⟶ ใช้งาน ⟶ ประเมินผล)
- Actions: กิจกรรมที่ผู้ใช้ทำในแต่ละขั้นตอน
- Touchpoints: จุดที่ผู้ใช้ติดต่อกับระบบ (เว็บ แอป อีเมล)

- Emotions: อารมณ์/ความรู้สึกในแต่ละขั้นตอน (พอใจ/ผิดหวัง/สับสน)
- Pain Points & Opportunities: ปัญหาและโอกาสในการปรับปรุง

ตัวอย่าง User Journey Map: การจองห้องประชุม

Persona: คุณอรุณ (Busy Executive) | Scenario: ต้องการจองห้องประชุมสำหรับประชุมลูกค้าพรุ่งนี้

ขั้นตอน	รายละเอียด
1. ตระหนักถึงความต้องการ	Action: เตรียมประชุมลูกค้า | Emotion: กังวลว่าจะมีห้องว่างหรือไม่ | Pain Point: ไม่รู้ว่าห้องไหนว่าง
2. ค้นหาข้อมูล	Action: เปิดระบบจองห้อง | Touchpoint: เว็บไซต์/แอป | Emotion: หวังว่าจะใช้งานง่าย
3. เลือกห้อง	Action: ดูตารางว่างและเลือกห้อง | Emotion: ดีใจที่มีห้องว่าง | Opportunity: แสดงรูปและอุปกรณ์ห้อง
4. จอง	Action: กรอกข้อมูลและยืนยัน | Touchpoint: ฟอร์มจอง | Pain Point: ฟอร์มยาวเกินไป
5. รับการยืนยัน	Action: รับอีเมลยืนยัน | Touchpoint: อีเมล | Emotion: มั่นใจ | Opportunity: ส่ง Calendar invite
6. ใช้งานห้อง	Action: ไปใช้ห้องประชุม | Emotion: พอใจ/ผิดหวัง | Pain Point: ถ้าห้องไม่ตรงตามที่คาดหวัง
7. ประเมินผล	Action: ใช้ระบบต่อหรือไม่ | Opportunity: ขอ Feedback | Emotion: พอใจโดยรวม

ประโยชน์ของ User Journey Map:

- เห็นภาพรวมประสบการณ์ผู้ใช้ตั้งแต่ต้นจนจบ
- ระบุจุดที่ผู้ใช้มีปัญหา (Pain Points) ได้ชัดเจน

- หาโอกาสในการปรับปรุงและสร้างความแตกต่าง
- ทำให้ทีมเข้าใจผู้ใช้มากขึ้นและตัดสินใจออกแบบได้ดีขึ้น

💡 **สรุปบทเรียนที่ 2:**

- UCD เน้นออกแบบโดยมีผู้ใช้เป็นศูนย์กลาง
- Personas ช่วยให้ทีมเข้าใจและตัดสินใจเพื่อผู้ใช้ได้ดีขึ้น
- User Journey Map แสดงประสบการณ์และระบุโอกาสปรับปรุง

บทเรียนที่ 3: การออกแบบ Responsive Design

Responsive Web Design with Flexbox & CSS Grid

3.1 ความสำคัญของ Responsive Design

Responsive Design คือการออกแบบเว็บไซต์ให้แสดงผลได้ดีบนทุกขนาดหน้าจอ ตั้งแต่มือถือ แท็บเล็ต จนถึงคอมพิวเตอร์ โดยใช้เทคนิค CSS ที่ปรับ Layout อัตโนมัติตามขนาดหน้าจอ
ทำให้ผู้ใช้มีประสบการณ์ที่ดีไม่ว่าจะเข้าถึงจากอุปกรณ์ใด

หลักการสำคัญของ Responsive Design:

- Fluid Grid: ใช้หน่วยแบบ % แทน px เพื่อความยืดหยุ่น
- Flexible Images: รูปภาพปรับขนาดตามพื้นที่ (max-width: 100%)
- Media Queries: กำหนด CSS สำหรับขนาดหน้าจอต่างๆ
- Mobile-First Approach: เริ่มออกแบบจากมือถือก่อน
- Progressive Enhancement: ค่อยๆ เพิ่มฟีเจอร์สำหรับจอใหญ่

Breakpoints ที่แนะนำ

อุปกรณ์	Breakpoint (Media Query)
Mobile (Portrait)	Default (ไม่ต้อง media query) หรือ max-width: 575px
Mobile (Landscape)	@media (min-width: 576px)
Tablet	@media (min-width: 768px)
Desktop	@media (min-width: 992px)
Large Desktop	@media (min-width: 1200px)

3.2 CSS Flexbox: การจัดวางแบบยืดหยุ่น

Flexbox เป็นระบบ Layout แบบมิติเดียว (one-dimensional) ที่จัดองค์ประกอบในแถว (row) หรือคอลัมน์ (column) ได้อย่างยืดหยุ่น เหมาะสำหรับการจัดวาง Navigation, Card Layout, หรือการจัดองค์ประกอบในแนวเดียวกัน

คุณสมบัติหลักของ Flexbox:

- display: flex - เปิดใช้งาน Flexbox container
- flex-direction - กำหนดทิศทาง (row, column, row-reverse, column-reverse)
- justify-content - จัดแนวนอน (flex-start, center, flex-end, space-between, space-around)
- align-items - จัดแนวตั้ง (flex-start, center, flex-end, stretch)
- flex-wrap - อนุญาตให้ขึ้นบรรทัดใหม่ (nowrap, wrap, wrap-reverse)
- gap - ระยะห่างระหว่างองค์ประกอบ (gap: 20px)

ตัวอย่างที่ 1: Navigation Bar ด้วย Flexbox

HTML:

```
<nav class="navbar">   <div class="logo">MyApp</div>
<ul class="nav-links">     <li><a
href="#">Home</a></li>     <li><a
href="#">About</a></li>     <li><a
href="#">Services</a></li>     <li><a
href="#">Contact</a></li>   </ul> </nav>
```

CSS:

```
.navbar {   display: flex;   justify-content: space-
between;   align-items: center;   padding: 1rem 2rem;
background: #2E75B6;   color: white; }  .nav-links {
display: flex;   list-style: none;   gap: 2rem; /*
ระยะห่างระหว่าง item */ }  /* Responsive: เปลี่ยนเป็นแนวตั้งในมือถือ */
@media (max-width: 768px) {   .navbar {     flex-
direction: column;     gap: 1rem;   }     .nav-links
{     flex-direction: column;     gap: 0.5rem;   } }
```

ตัวอย่างที่ 2: Card Layout แบบ Responsive

CSS:

```
.card-container {   display: flex;   flex-wrap: wrap;
/* อนุญาตให้ขึ้นบรรทัดใหม่ */   gap: 1.5rem;   padding: 2rem; }
.card {   flex: 1 1 300px; /* grow shrink basis */
/* flex-basis: 300px = ความกว้างขั้นต่ำ */   /* flex-grow: 1 =
ขยายเต็มพื้นที่ว่าง */   /* flex-shrink: 1 = หดได้ถ้าจำเป็น */
padding: 1.5rem;   background: white;   border-
radius: 8px;   box-shadow: 0 2px 8px rgba(0,0,0,0.1);
}  /* Mobile: 1 card ต่อแถว */ @media (max-width: 768px)
{   .card {     flex: 1 1 100%;   } }
```

3.3 CSS Grid: การจัดวางแบบสองมิติ

CSS Grid เป็นระบบ Layout แบบสองมิติ (two-dimensional) ที่จัดการทั้งแถว (rows) และคอลัมน์ (columns) พร้อมกัน เหมาะสำหรับ Layout ที่ซับซ้อน เช่น Dashboard, Gallery, หรือหน้าเว็บโดยรวม ให้การควบคุมที่แม่นยำกว่า Flexbox

คุณสมบัติหลักของ CSS Grid:

- display: grid - เปิดใช้งาน Grid container
- grid-template-columns - กำหนดคอลัมน์ (เช่น 1fr 1fr 1fr หรือ repeat(3, 1fr))
- grid-template-rows - กำหนดแถว
- gap - ระยะห่างระหว่าง cells (grid-gap เดิม)
- grid-column / grid-row - กำหนดตำแหน่ง item (เช่น grid-column: 1 / 3)
- grid-template-areas - ตั้งชื่อพื้นที่และจัดวาง

ตัวอย่างที่ 1: Page Layout ด้วย Grid Areas

HTML:

```
<div class="page-layout">   <header class="header">Header</header>   <aside class="sidebar">Sidebar</aside>   <main class="content">Main Content</main>   <footer class="footer">Footer</footer> </div>
```

CSS:

```
.page-layout {   display: grid;   grid-template-areas:     "header header"     "sidebar content" "footer footer";   grid-template-columns: 250px 1fr; grid-template-rows: auto 1fr auto;   min-height: 100vh;   gap: 1rem; }  .header { grid-area: header; } .sidebar { grid-area: sidebar; } .content { grid-area: content; } .footer { grid-area: footer; }  /* Responsive: Sidebar อยู่บน Content ในมือถือ */ @media (max-width: 768px) {   .page-layout {     grid-template-areas:       "header"       "sidebar"       "content" "footer";     grid-template-columns: 1fr;   } }
```

ตัวอย่างที่ 2: Photo Gallery แบบ Responsive

CSS:

```
.gallery {   display: grid;   grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));   /* auto-fit = สร้างคอลัมน์อัตโนมัติ */   /* minmax(250px, 1fr) = ขั้นต่ำ 250px, ขยายเต็มพื้นที่ */   gap: 1rem;   padding: 1rem; }  .gallery-item {   aspect-ratio: 4 / 3; /* รักษาสัดส่วน */   overflow: hidden;   border-radius: 8px; }  .gallery-item img {   width: 100%;   height: 100%;   object-fit: cover; /* ครอบภาพให้เต็มพื้นที่ */   transition: transform 0.3s; }  .gallery-item img:hover {   transform: scale(1.05); /* ซูมเมื่อ hover */ }  /* Featured item ใหญ่ขึ้น */ .gallery-item.featured {   grid-column: span 2;   grid-row: span 2; }
```

3.4 เปรียบเทียบ Flexbox vs Grid

ตารางเปรียบเทียบ

แนวคิด	คำอธิบาย	ประโยชน์
มิติ	Flexbox: มิติเดียว (แนวเดียว) | Grid: สองมิติ (แถว + คอลัมน์)	Flexbox สำหรับ component เดี่ยว | Grid สำหรับ layout รวม
ใช้งาน	Flexbox: Navigation, Toolbar, Card content | Grid: Page layout, Dashboard, Gallery	เลือกตามความซับซ้อนของ layout
ควบคุม	Flexbox: ควบคุมจากแนว (main axis) | Grid: ควบคุมแบบ precise positioning	Grid ให้การควบคุมที่แม่นยำกว่า
Content	Flexbox: ขึ้นกับขนาด content | Grid: กำหนดพื้นที่ล่วงหน้า	Flexbox ยืดหยุ่นตาม content
Browser Support	ทั้งสองรองรับเบราว์เซอร์สมัยใหม่ดี	ใช้ร่วมกันได้ (Grid layout + Flexbox components)

เมื่อไหร่ควรใช้อันไหน?

- ใช้ Flexbox เมื่อ: จัดองค์ประกอบในแนวเดียว, Navigation bars, Card layouts, Button groups
- ใช้ Grid เมื่อ: Layout ทั้งหน้า, Dashboard, Gallery, Form layouts ที่ซับซ้อน
- ใช้ร่วมกัน: Grid สำหรับโครงสร้างหลัก + Flexbox สำหรับ components ย่อย

💡 **สรุปบทเรียนที่ 3:**

- Responsive Design สำคัญมากในยุคที่ใช้มือถือเยอะ
- Flexbox เหมาะสำหรับ layout มิติเดียว เช่น navbar, cards
- Grid เหมาะสำหรับ layout ซับซ้อน เช่น page layout, dashboard
- ใช้ Media Queries เพื่อปรับ layout ตามขนาดหน้าจอ

ในสัปดาห์ถัดไป (สัปดาห์ที่ 5)

เราจะเรียนรู้เกี่ยวกับ:

- UI Design & Prototyping ด้วย Figma
- Usability Heuristics (Nielsen's 10 Heuristics)
- JavaScript เบื้องต้น: DOM Manipulation และ Event Handling
- Form Validation และการจัดการข้อมูล Input

Workshop & Lab Activities

กิจกรรมปฏิบัติการ 3 ชั่วโมง

Workshop 1: วิเคราะห์และออกแบบ UX (60 นาที)

วัตถุประสงค์:

- สร้าง Personas สำหรับ Term Project
- วาด User Journey Map สำหรับ scenario หลัก

ขั้นตอนการทำ:

4. แบ่งกลุ่ม (3-4 คน) ใช้ Term Project เดียวกัน (15 นาที)
5. กำหนดผู้ใช้เป้าหมาย และสร้าง 2 Personas ที่แตกต่างกัน (25 นาที)
6. เลือก 1 Persona และวาด User Journey Map สำหรับ 1 scenario หลัก (เช่น การลงทะเบียน, การทำธุรกรรม) (20 นาที)

✅ Deliverable (ส่งหลังเรียน):

- เอกสาร Personas 2 คน (พร้อมข้อมูลครบตามที่เรียน)
- User Journey Map 1 scenario (อาจเขียนมือหรือใช้เครื่องมือออนไลน์)

Workshop 2: ปรับปรุง HTML/CSS ให้เป็น Responsive (90 นาที)

วัตถุประสงค์:

- ปรับหน้า HTML/CSS ของ Term Project ให้เป็น Responsive
- ฝึกใช้ Flexbox และ Grid

ขั้นตอนการทำ:

7. ทบทวนหน้า HTML ที่มีอยู่แล้วจากสัปดาห์ที่ 3 (10 นาที)
8. สร้าง Navigation Bar ด้วย Flexbox ที่เปลี่ยนเป็นแนวตั้งในมือถือ (30 นาที)
9. ใช้ CSS Grid สำหรับ Main Content Layout (เช่น 2-column layout: sidebar + content) (30 นาที)
10. เพิ่ม Media Queries สำหรับ Mobile (max-width: 768px) และทดสอบ (20 นาที)

💡 เทคนิค:

- ใช้ Browser DevTools (F12) เปิด Responsive Mode เพื่อทดสอบ

- เริ่มจาก Mobile-First แล้วค่อย scale up
- ใช้ max-width: 100% สำหรับรูปภาพทั้งหมด

✅ Deliverable (ส่งก่อนเลิกเรียน):

- Push โค้ด HTML/CSS ที่เป็น Responsive ขึ้น GitHub
- แคปหน้าจอทดสอบใน 3 ขนาด: Mobile (375px), Tablet (768px), Desktop (1200px)
- ส่ง Git commit link ใน Google Classroom

Workshop 3: Code Review และปรับปรุงการออกแบบ (30 นาที)

วัตถุประสงค์:

- ฝึกการให้ Feedback ด้าน UX และ Code Quality
- ปรับปรุงโค้ดตาม Feedback

ขั้นตอนการทำ:

11. แลกเปลี่ยนกลุ่มกับกลุ่มข้างๆ ให้ดูหน้าเว็บของกันและกัน (10 นาที)
12. ให้ Feedback อย่างสร้างสรรค์ ตรวจสอบ: SOLID principles, Responsive design, UX issues (10 นาที)
13. ปรับปรุงโค้ดตาม Feedback ที่ได้รับ (10 นาที)

Lab Assignment (ทำที่บ้าน)

ส่งภายใน 1 สัปดาห์ (ก่อนเรียนสัปดาห์ถัดไป)

งานที่ 1: ปรับปรุง Personas และ User Journey (20 คะแนน)

- เพิ่มรายละเอียดให้ Personas ครบถ้วนตามที่เรียน (ชื่อ, อายุ, อาชีพ, เป้าหมาย, Pain Points, Quote)
- ขยาย User Journey Map ให้มี 7 stages ครบ (Awareness ⟶ Evaluation ⟶ Purchase ⟶ Usage ⟶ Support)
- ระบุ Pain Points และ Opportunities ในแต่ละ stage อย่างน้อย 3 จุด
- สร้างเป็น Document หรือ Slide ที่ดูง่าย สวยงาม

งานที่ 2: ปรับปรุง Responsive Design (30 คะแนน)

- สร้างหน้าเพิ่มอย่างน้อย 2 หน้า (เช่น About, Services, Product List) ที่เป็น Responsive
- ใช้ Flexbox สำหรับ Navigation, Card layouts, Footer
- ใช้ Grid สำหรับ Main layout ของหน้า
- กำหนด Media Queries สำหรับ 3 breakpoints: Mobile (< 768px), Tablet (768px-992px), Desktop (> 992px)
- ทดสอบและแคปหน้าจอในแต่ละขนาดหน้าจอ

งานที่ 3: Code Refactoring ตามหลัก SOLID (20 คะแนน)

- ทบทวนโค้ด HTML/CSS ที่เขียนไว้ และระบุ Code Smells (ถ้ามี)
- แยก CSS เป็นไฟล์ต่างหาก และจัดกลุ่ม (variables, layout, components, responsive)
- ใช้ CSS Variables (Custom Properties) สำหรับสี และขนาดที่ใช้บ่อย
- สร้าง Reusable CSS Classes (เช่น .btn, .card, .container)
- เขียน Comments อธิบายส่วนสำคัญของโค้ด

งานที่ 4: เขียนรายงาน Reflection (10 คะแนน)

เขียนรายงานสั้นๆ (1-2 หน้า) ตอบคำถามต่อไปนี้:

- หลักการ SOLID ข้อไหนที่คุณคิดว่าสำคัญที่สุดสำหรับโครงงานของคุณ? เพราะอะไร?
- Personas และ User Journey ช่วยให้คุณเข้าใจผู้ใช้ดีขึ้นอย่างไร?

- ความท้าทายที่พบเมื่อทำ Responsive Design และแก้ไขอย่างไร?
- สิ่งที่เรียนรู้ใหม่ในสัปดาห์นี้ที่คิดว่าจะนำไปใช้ได้จริง?

การส่งงาน

- Push โค้ดทั้งหมดขึ้น GitHub Repository ของกลุ่ม
- สร้างโฟลเดอร์ week4 แยกงานแต่ละส่วน
- ส่ง Link ของ Git Repository พร้อม Screenshot ผลงานใน Google Classroom
- **กำหนดส่ง: ก่อนเรียนสัปดาห์ที่ 5 (วันที่ _____)**

⚠️ **หมายเหตุ:**

- งานที่ส่งช้าจะถูกหัก 10% ต่อวัน
- ต้องมี Git commit history ที่สม่ำเสมอ (ไม่ใช่ commit เดียวตอนสุดท้าย)
- แต่ละคนในกลุ่มต้องมีส่วนร่วมในการ commit (ตรวจสอบได้จาก Git History)

Checklist สำหรับนักศึกษา

ทฤษฎี (2 ชั่วโมง)

- ☐ เข้าใจหลักการ SOLID และสามารถอธิบายได้
- ☐ เข้าใจความแตกต่างระหว่าง Coupling และ Cohesion
- ☐ เข้าใจหลักการ User-Centred Design
- ☐ รู้จัก Personas และ User Journey Map
- ☐ เข้าใจหลักการ Responsive Design
- ☐ รู้จักและใช้ Flexbox และ Grid ได้

ปฏิบัติ (3 ชั่วโมง)

- ☐ สร้าง Personas 2 คนสำหรับ Term Project
- ☐ วาด User Journey Map อย่างน้อย 1 scenario
- ☐ สร้าง Navigation Bar ด้วย Flexbox
- ☐ ใช้ Grid สำหรับ Page Layout
- ☐ เพิ่ม Media Queries สำหรับ Mobile
- ☐ ทดสอบ Responsive ใน 3 ขนาดหน้าจอ
- ☐ Push โค้ดขึ้น GitHub

Lab Homework

- ☐ ปรับปรุง Personas และ User Journey ให้สมบูรณ์
- ☐ สร้างหน้าเว็บเพิ่ม 2 หน้าที่เป็น Responsive
- ☐ Refactor CSS ตามหลัก SOLID
- ☐ เขียนรายงาน Reflection
- ☐ ส่งงานตามกำหนดใน Google Classroom

Good luck with your Term Project! 🚀

หากมีคำถามสามารถสอบถามได้ทาง Google Classroom หรือในชั่วโมงปรึกษา