# Dart Professional

## Programming Language:

# Professional

## (Integrative-Generative AI Edition)



Student Price Book Center

# คำนำ

การพัฒนาแอปพลิเคชันสมัยใหม่ไม่เพียงแต่ต้องการโค้ดที่ทำงานได้เท่านั้น แต่ยังต้องให้ความสำคัญกับความเสถียร ประสิทธิภาพ และความสามารถในการขยายตัวของระบบ ภาษา Dart ได้รับการออกแบบมาเพื่อรองรับความต้องการเหล่านี้ ด้วยความเรียบง่าย ทันสมัย และยืดหยุ่นในการใช้งาน ทั้งในฝั่ง client, server และ web ทำให้ Dart กลายเป็นเครื่องมือที่นักพัฒนามืออาชีพเลือกใช้เมื่อสร้างระบบที่มีความซับซ้อนและต้องการประสิทธิภาพสูง หนังสือเล่มนี้จึงถูกเขียนขึ้นเพื่อให้ผู้อ่านเข้าใจ Dart ในระดับ Professional ครอบคลุมตั้งแต่การจัดการโปรเจ็กต์ การใช้ package และ library การทดสอบ การจัดการหน่วยความจำและประสิทธิภาพ ไปจนถึงการนำไปใช้งานจริงใน production environment

ในส่วนของ **บทที่ 17 การจัดการโปรเจ็กต์ Dart (Dart Project Management)** ผู้อ่านจะได้เรียนรู้หลักการจัดการโปรเจ็กต์อย่างเป็นระบบ ตั้งแต่การสร้างโปรเจ็กต์ด้วยคำสั่ง dart create การจัดโครงสร้างไฟล์และโฟลเดอร์ให้มีมาตรฐาน การรันโปรเจ็กต์ด้วย dart run และการคอมไพล์ด้วย dart compile ไปจนถึงการสร้าง script utility เพื่อช่วยลดงานซ้ำซ้อน การเข้าใจแนวทางเหล่านี้จะช่วยให้นักพัฒนาสามารถเริ่มต้นโปรเจ็กต์ Dart ได้อย่างมั่นใจและมีระบบ

ต่อมาที่ **บทที่ 18 Package & Library** จะเน้นการใช้งานแพ็กเกจและไลบรารีอย่างมืออาชีพ ทั้งการค้นหาและติดตั้งแพ็กเกจจาก pub.dev การเพิ่ม dependency ในไฟล์ pubspec.yaml การ import และ export library และการสร้าง package ของตนเอง พร้อมตัวอย่างที่บูรณาการเพื่อให้ผู้อ่านเห็นภาพการใช้งานจริง การทำความเข้าใจ package และ library ช่วยให้การพัฒนาโค้ดสามารถนำกลับมาใช้ซ้ำ ลดความซับซ้อน และเพิ่มประสิทธิภาพในการทำงานร่วมกันเป็นทีม

**บทที่ 19 Testing** เน้นการสร้างคุณภาพของซอฟต์แวร์ผ่านการทดสอบ ตั้งแต่ Unit Testing ด้วย test package การเขียน Test Case ที่ครอบคลุมทุกสถานการณ์ การใช้ Mocking เพื่อลดการพึ่งพา dependency ภายนอก และการรันทดสอบอัตโนมัติ (Automated Testing) ทั้งหมดนี้มาพร้อมตัวอย่างที่บูรณาการเพื่อให้ผู้อ่านสามารถนำแนวทางไปประยุกต์ใช้กับโปรเจ็กต์จริงได้ การทดสอบโค้ดไม่เพียงลดข้อผิดพลาด แต่ยังช่วยสร้างความมั่นใจในการพัฒนาฟีเจอร์ใหม่และการ refactor โค้ด

สำหรับ **บทที่ 20 Memory & Performance** ผู้อ่านจะได้เจาะลึกกลไกการจัดการหน่วยความจำและการปรับปรุงประสิทธิภาพใน Dart เริ่มจาก Garbage Collection การจัดการ Memory Leak และเทคนิคการ Optimize โค้ด Dart เพื่อให้ทำงานได้รวดเร็ว ใช้ทรัพยากรอย่างคุ้มค่า พร้อมตัวอย่างที่บูรณาการ การเข้าใจประเด็นเหล่านี้จะช่วยให้นักพัฒนาสามารถสร้างซอฟต์แวร์ที่เสถียร มีประสิทธิภาพ และตอบสนองความต้องการผู้ใช้ได้อย่างเต็มที่

สุดท้ายใน **บทที่ 21 Dart สำหรับ Production** จะครอบคลุมการเตรียมโค้ด Dart ให้พร้อมใช้งานจริง ทั้งการ Compile เป็น Native Code เพื่อประสิทธิภาพสูงสุด การ Compile เป็น JavaScript สำหรับเว็บแอป การใช้งาน Dart บน Server ด้วย Dart Frog หรือ Shelf และการพัฒนาเว็บแอปด้วย Dart พร้อมตัวอย่างที่บูรณาการ ความรู้เหล่านี้ช่วยให้ผู้อ่านสามารถ deploy แอปพลิเคชันใน production environment ได้อย่างมั่นใจ ปลอดภัย และมีประสิทธิภาพ

หนังสือเล่มนี้ถูกออกแบบให้เป็นคู่มือสำหรับนักพัฒนาที่ต้องการยกระดับความสามารถด้าน Dart ให้เป็นระดับ Professional ด้วยการเรียนรู้แบบครบวงจร ทั้งทฤษฎี เทคนิค และตัวอย่างการใช้งานจริง ผู้อ่านจะได้เข้าใจทั้งแนวคิดพื้นฐานและแนวปฏิบัติขั้นสูง เพื่อให้สามารถสร้างระบบซอฟต์แวร์ที่เสถียร มีประสิทธิภาพ และพร้อมรองรับการเติบโตในอนาคต

ด้วยรักและปรารถนาดี

**ศูนย์หนังสือราคานักเรียน**

# สารบัญ

หน้า

# บทที่ 17
# การจัดการโปรเจ็กต์ Dart
# (Dart Project Management)

**เนื้อหา**

- การจัดการโปรเจ็กต์ Dart
- เจาะลึกรายละเอียดของ การจัดการโปรเจกต์ Dart
- การสร้าง Project ด้วย dart create
- การจัดโครงสร้างไฟล์และโฟลเดอร์
- dart run และ dart compile
- การสร้าง Script utility

---

**บทนำ บทที่ 17: การจัดการโปรเจ็กต์ Dart**

ในการพัฒนาแอปพลิเคชันด้วยภาษา Dart สิ่งสำคัญประการแรกคือการจัดการโปรเจ็กต์อย่างเป็นระบบ เนื่องจากโปรเจ็กต์ทำหน้าที่เป็นศูนย์รวมของโค้ด ไลบรารี ทรัพยากร และการตั้งค่าต่าง ๆ ที่เกี่ยวข้อง หากขาดการจัดการที่ดี การพัฒนาจะเกิดความซับซ้อน เพิ่มความเสี่ยงต่อการเกิดข้อผิดพลาด และทำให้การดูแลรักษาระยะยาวเป็นเรื่องยาก บทนี้จึงมุ่งเน้นไปที่การจัดการโปรเจ็กต์ Dart ตั้งแต่การสร้างโครงเริ่มต้นไปจนถึงการใช้เครื่องมือต่าง ๆ ที่ช่วยสนับสนุนกระบวนการพัฒนา

หัวข้อแรกคือ **การสร้าง Project ด้วย dart create** ซึ่งเป็นคำสั่งมาตรฐานที่ช่วยให้ผู้พัฒนาสามารถเริ่มต้นโปรเจ็กต์ใหม่ได้อย่างรวดเร็ว โดยไม่จำเป็นต้องสร้างไฟล์และโฟลเดอร์เองทั้งหมด คำสั่งนี้จะกำหนดโครงสร้างพื้นฐานอัตโนมัติ เช่น โฟลเดอร์ lib สำหรับเก็บโค้ดหลัก และไฟล์ pubspec.yaml สำหรับกำหนด dependency ส่งผลให้การเริ่มต้นมีรูปแบบที่เป็นมาตรฐานและสอดคล้องกัน

ถัดมาคือ **การจัดโครงสร้างไฟล์และโฟลเดอร์** ซึ่งเป็นหัวใจสำคัญในการรักษาความเป็นระเบียบและความเข้าใจในระบบ หากจัดการได้อย่างเหมาะสม จะทำให้การขยายระบบหรือการบำรุงรักษาในอนาคตเป็นไปอย่างราบรื่น นักพัฒนาสามารถแบ่งโฟลเดอร์และไฟล์ตามฟีเจอร์หรือเลเยอร์ เช่น models, services, และ utils เพื่อสร้างความชัดเจน ลดความสับสน และสนับสนุนการทำงานร่วมกันในทีม

อีกประเด็นหนึ่งที่ควรให้ความสำคัญคือ **การใช้ dart run และ dart compile** ซึ่งเป็นเครื่องมือหลักในการทดสอบและเตรียมโค้ดสำหรับการใช้งานจริง dart run ช่วยให้ผู้พัฒนาสามารถรันแอปพลิเคชันได้ทันทีจากไฟล์ต้นฉบับ ส่วน dart compile มีบทบาทในการคอมไพล์โค้ด Dart ให้กลายเป็น

Executable ที่สามารถทำงานได้โดยไม่จำเป็นต้องติดตั้ง Dart SDK บนเครื่องปลายทาง ทำให้การแจกจ่ายและการใช้งานสะดวกยิ่งขึ้น

นอกจากนี้ **การสร้าง Script utility** ถือเป็นอีกแนวทางหนึ่งที่ช่วยให้การพัฒนามีความคล่องตัวมากขึ้น โดยการเขียนสคริปต์สำหรับงานซ้ำ ๆ เช่น การ build แอปพลิเคชัน การ run test หรือการ deploy ไปยัง environment ต่าง ๆ การใช้สคริปต์ช่วยลดข้อผิดพลาดจากการทำงานด้วยมือ และทำให้กระบวนการทำงานเป็นไปอย่างอัตโนมัติและมีประสิทธิภาพ

การเรียนรู้การจัดการโปรเจ็กต์ Dart ไม่ได้มีเพียงแค่การใช้คำสั่งหรือเครื่องมือเท่านั้น แต่ยังรวมถึงการฝึกคิดเชิงโครงสร้าง การออกแบบระบบ และการสร้างมาตรฐานสำหรับการทำงานเป็นทีม เมื่อโค้ดขยายตัวมากขึ้น ความสามารถในการบริหารจัดการโปรเจ็กต์อย่างเป็นระบบจะยิ่งทวีความสำคัญ เพื่อรองรับทั้งการปรับปรุงและการพัฒนาต่อยอดในอนาคต

ดังนั้น บทที่ 17 จึงมีเป้าหมายในการปูพื้นฐานและชี้แนะแนวทางการจัดการโปรเจ็กต์ Dart อย่างมีมาตรฐาน ครอบคลุมตั้งแต่การสร้างโปรเจ็กต์เบื้องต้น การวางโครงสร้างไฟล์และโฟลเดอร์ การใช้คำสั่งที่จำเป็นสำหรับการรันและคอมไพล์ ตลอดจนการสร้างสคริปต์เพื่อเพิ่มประสิทธิภาพการทำงาน หากผู้อ่านสามารถประยุกต์ใช้ความรู้เหล่านี้ได้ ก็จะทำให้การพัฒนาโปรเจ็กต์ด้วย Dart เป็นไปอย่างราบรื่น มีระเบียบ และพร้อมต่อการขยายตัวในอนาคต

---

## การจัดการโปรเจ็กต์ Dart

- การสร้าง Project ด้วย dart create
- การจัดโครงสร้างไฟล์และโฟลเดอร์
- การใช้ dart run และ dart compile
- การสร้าง Script utility

---

**1. การสร้าง Project ด้วย dart create**

dart create เป็นคำสั่งหลักที่ใช้สำหรับสร้างโปรเจกต์ Dart ขึ้นมา โดยมี Template ให้เลือกใช้งานหลายแบบ

- **Console app:** โปรเจกต์สำหรับ Command-line, ใช้สำหรับสร้างโปรแกรมที่รันบน Terminal เป็นหลัก
- **Web app:** โปรเจกต์สำหรับสร้างเว็บแอปพลิเคชัน
- **Server app:** โปรเจกต์สำหรับสร้าง Web server (เช่น REST API)

**วิธีการใช้งาน:**

Bash

```
dart create -t console my_app
```

คำสั่งนี้จะสร้างโฟลเดอร์ชื่อ my_app พร้อมโครงสร้างไฟล์ที่จำเป็นสำหรับโปรเจกต์ Console

**2. การจัดโครงสร้างไฟล์และโฟลเดอร์**

โปรเจกต์ Dart จะมีโครงสร้างไฟล์มาตรฐานเพื่อให้ง่ายต่อการจัดการและทำงานร่วมกับผู้อื่น

- **lib/**: โฟลเดอร์หลักสำหรับเก็บไฟล์โค้ดของโปรเจกต์
  - lib/my_app.dart: ไฟล์หลักของ Library หรือ Module
- **bin/**: โฟลเดอร์สำหรับเก็บไฟล์ที่สามารถรันได้ (Executable)
  - bin/main.dart: ไฟล์หลักของโปรแกรม ที่มีฟังก์ชัน main()
- **test/**: โฟลเดอร์สำหรับเขียน Unit tests
- **.gitignore**: ไฟล์สำหรับกำหนดไฟล์ที่ไม่ต้องการให้ Git จัดการ
- **pubspec.yaml**: ไฟล์สำหรับจัดการ Dependency และข้อมูลของโปรเจกต์ (สำคัญมาก)
  - name: ชื่อโปรเจกต์
  - version: เวอร์ชั่นของโปรเจกต์
  - environment: เวอร์ชั่นของ Dart ที่โปรเจกต์รองรับ
  - dependencies: แพ็กเกจที่โปรเจกต์ต้องใช้

**3. การใช้ dart run และ dart compile**

หลังจากเขียนโค้ดเสร็จแล้ว เรามีคำสั่งหลักที่ใช้รันและคอมไพล์โปรแกรม

- **dart run**: ใช้สำหรับรันโปรแกรม Dart โดยตรงจากไฟล์ต้นฉบับ
  - **ข้อดี:** สะดวกและรวดเร็วสำหรับการพัฒนา
  - **วิธีการใช้งาน:** dart run หรือ dart run bin/main.dart
- **dart compile**: ใช้สำหรับคอมไพล์โค้ด Dart ให้เป็นไฟล์ Executable
  - **ข้อดี:** ไฟล์ที่ได้สามารถนำไปรันบนระบบอื่นได้ทันทีโดยไม่ต้องติดตั้ง Dart SDK และทำงานได้รวดเร็วกว่า
  - **วิธีการใช้งาน:** dart compile exe bin/main.dart -o my_app.exe
  - คำสั่งนี้จะสร้างไฟล์ my_app.exe ที่สามารถรันได้

**4. การสร้าง Script utility**

ในบางครั้งเราต้องการสร้างโปรแกรมขนาดเล็กเพื่อทำงานเฉพาะอย่าง เช่น การแปลงไฟล์, การประมวลผลข้อมูล ซึ่ง Dart เหมาะมากสำหรับการทำ Script

- **วิธีการสร้าง:**

1. สร้างไฟล์ .dart ขึ้นมาในโฟลเดอร์ที่ต้องการ (มักจะเก็บไว้ใน bin/ หรือ tool/)
2. เขียนโค้ดในฟังก์ชัน main()
3. รันด้วยคำสั่ง dart run path/to/my_script.dart

**ตัวอย่าง:**

Dart

```
// bin/gen_report.dart
void main(List<String> arguments) {
  if (arguments.isEmpty) {
    print('กรุณาระบุชื่อไฟล์!');
```

```
    return;
  }
  final filename = arguments[0];
  print('กำลังสร้างรายงานสำหรับไฟล์: $filename...');
  // โค้ดสำหรับสร้างรายงาน
  print('สร้างรายงานสำเร็จ!');
}
```

**วิธีรัน:**

Bash

```
dart run bin/gen_report.dart sales_data.csv
```

คำสั่งนี้ทำให้ Dart กลายเป็นเครื่องมือที่มีประสิทธิภาพสำหรับการสร้าง Script Automation ในชีวิตประจำวันได้ครับ

## เจาะลึกรายละเอียดของ การจัดการโปรเจกต์ Dart

---

**1. การสร้าง Project ด้วย dart create (เชิงลึก)**

dart create ไม่ใช่แค่การสร้างไฟล์ แต่เป็นการตั้งค่าสภาพแวดล้อมการพัฒนาที่สมบูรณ์แบบ

- **Templates (-t flag):**
  - -t console: สร้างโปรเจกต์สำหรับรันบน Command-line. โครงสร้างจะประกอบด้วย bin/main.dart สำหรับโค้ดที่รันได้, lib/ สำหรับไลบรารีที่โค้ดส่วนอื่นสามารถเรียกใช้ได้, และ pubspec.yaml สำหรับจัดการแพ็กเกจ
  - -t web: สร้างโปรเจกต์สำหรับ Web app โดยใช้แพ็กเกจ dart:html ซึ่งโค้ด Dart จะถูกคอมไพล์เป็น JavaScript เพื่อรันในเว็บเบราว์เซอร์
  - -t server: สร้างโปรเจกต์สำหรับ Web server ที่ใช้แพ็กเกจอย่าง shelf หรือ http เพื่อรับ-ส่ง HTTP request/response
- **เหตุผลเบื้องหลัง:** การมี Template ที่แตกต่างกันช่วยให้นักพัฒนาไม่ต้องตั้งค่าเริ่มต้นเอง และมั่นใจได้ว่าโครงสร้างไฟล์เป็นไปตามมาตรฐานที่ชุมชนกำหนด

---

**2. การจัดโครงสร้างไฟล์และโฟลเดอร์ (เชิงลึก)**

โครงสร้างไฟล์มาตรฐานใน Dart ไม่ใช่แค่การจัดระเบียบ แต่มีผลต่อการทำงานของระบบจัดการแพ็กเกจ (pub) และการ Import

- **lib/ folder:**
  - นี่คือหัวใจของโปรเจกต์ โค้ดทั้งหมดที่ไม่ได้เป็นโปรแกรมที่รันได้ (Executable) ควรอยู่ในโฟลเดอร์นี้

- **ไฟล์หลักของ Library:** มักจะเป็นไฟล์เดียวที่มี export คำสั่งเพื่อรวมไฟล์ย่อยทั้งหมดเข้าด้วยกัน ทำให้การ Import จากภายนอกทำได้ง่าย เช่น import 'package:my_app/my_app.dart';

- **bin/ folder:**
- โฟลเดอร์นี้มีไว้สำหรับไฟล์ที่มีฟังก์ชัน main() เท่านั้น
- เมื่อคุณรันคำสั่ง dart run, Dart จะมองหาไฟล์ที่รันได้ในโฟลเดอร์นี้เป็นอันดับแรก
- **pubspec.yaml (ไฟล์ Manifest):**
- **name:** ชื่อนี้ต้องไม่ซ้ำกันบน pub.dev หากคุณต้องการเผยแพร่
- **dependencies:**
  - ^1.2.3: นี่คือ **Caret Syntax** ซึ่งหมายความว่า Dart จะใช้เวอร์ชัน 1.2.3 หรือสูงกว่า แต่ต้องไม่ถึงเวอร์ชัน 2.0.0
  - **dev_dependencies:** สำหรับแพ็กเกจที่ใช้ในการพัฒนาเท่านั้น เช่น แพ็กเกจสำหรับ Test (test) หรือสำหรับสร้างไฟล์ (build_runner) ซึ่งจะไม่ถูกรวมอยู่ในตอนคอมไพล์จริง
- **.gitignore:** มีไว้เพื่อป้องกันไม่ให้ไฟล์ที่สร้างขึ้นอัตโนมัติ (เช่น .dart_tool/, build/, *.dart.js) ถูก Commit เข้าไปใน Git Repository

---

**3. การใช้ dart run และ dart compile (เชิงลึก)**

การทำความเข้าใจความแตกต่างของสองคำสั่งนี้สำคัญต่อการเลือกใช้ในแต่ละขั้นตอน

- **dart run (Just-In-Time Compilation):**
- เมื่อคุณรันคำสั่งนี้, Dart จะใช้ **JIT (Just-In-Time) Compiler** เพื่อคอมไพล์โค้ดในระหว่างที่โปรแกรมกำลังทำงาน
- **ข้อดี:** JIT Compiler มีความเร็วในการรันครั้งแรกที่สูงมาก เหมาะสำหรับการพัฒนาและ Debug เพราะสามารถทำการ Hot Reload ได้
- **กระบวนการ:** dart run จะมองหาไฟล์ bin/main.dart ในโปรเจกต์ปัจจุบัน ถ้าไม่เจอจะต้องระบุพาธของไฟล์ด้วยตัวเอง

- **dart compile (Ahead-Of-Time Compilation):**
- เมื่อคุณรันคำสั่งนี้, Dart จะใช้ **AOT (Ahead-Of-Time) Compiler** เพื่อคอมไพล์โค้ดทั้งหมดให้เป็นโค้ดเครื่อง (Machine Code) ก่อนที่จะรันโปรแกรม
- **ข้อดี:** ไฟล์ Executable ที่ได้มีขนาดเล็กและรันได้เร็วมาก เพราะไม่ต้องมีการคอมไพล์ใดๆ เพิ่มเติม
- **กระบวนการ:** dart compile exe path/to/main.dart จะสร้างไฟล์ Executable ที่ไม่มี Dependency กับ Dart SDK ทำให้สามารถแจกจ่ายไฟล์นี้ให้ผู้อื่นนำไปรันได้ทันที

---

**4. การสร้าง Script utility (เชิงลึก)**

การใช้ Dart เป็นภาษาสำหรับ Scripting เป็นแนวทางปฏิบัติที่ดี เพราะช่วยให้โค้ดมีความน่าเชื่อถือและจัดการได้ง่ายกว่าภาษา Scripting อื่นๆ

- **ใช้ bin/ หรือ tool/:** ไฟล์ Script ที่รันได้ควรอยู่ในโฟลเดอร์เหล่านี้เพื่อให้แยกจากโค้ดของ Library หลัก
- **การจัดการ Argument:** ฟังก์ชัน main(List<String> arguments) ช่วยให้ Script ของคุณสามารถรับค่าจาก Command-line ได้อย่างมีประสิทธิภาพ
- **ความน่าเชื่อถือของโค้ด:** เนื่องจาก Dart เป็นภาษาที่แข็งแรงในเรื่อง Type Safety (Strongly Typed) ทำให้คุณสามารถตรวจจับข้อผิดพลาดในโค้ดได้ตั้งแต่ตอน Compile
- **ข้อดีของการใช้ Dart สำหรับ Script:**

1. **Type Safety:** ลดโอกาสเกิดข้อผิดพลาดจากประเภทข้อมูลที่ไม่ถูกต้อง
2. **ประสิทธิภาพ:** แม้จะรันแบบ JIT ก็ยังทำงานได้เร็วพอสำหรับงาน Script ส่วนใหญ่
3. **เครื่องมือครบครัน:** สามารถใช้แพ็กเกจจาก pub.dev มาช่วยในการทำงาน Script ได้อย่างง่ายดาย

## การสร้าง Project ด้วย dart create

dart create เป็นคำสั่งพื้นฐานที่ใช้ในการสร้างโปรเจกต์ Dart ขึ้นมาอย่างรวดเร็ว โดยจะสร้างโครงสร้างไฟล์และโฟลเดอร์ที่จำเป็นให้คุณพร้อมใช้งานทันที

---

***หลักการทำงาน***

คำสั่ง dart create จะสร้างโปรเจกต์จาก **Template** ที่มีให้เลือกใช้ ซึ่งจะกำหนดโครงสร้างของโปรเจกต์ให้เหมาะสมกับประเภทของแอปพลิเคชันที่คุณต้องการพัฒนา

- **Console application**: ใช้สำหรับสร้างโปรแกรมที่รันบน Command-line
- **Web application**: ใช้สำหรับสร้างโปรแกรมที่รันบน Web browser
- **Server application**: ใช้สำหรับสร้าง Web server หรือ REST API

***วิธีการใช้งาน***

**1. เปิด Terminal หรือ Command Prompt**

**2. รันคำสั่ง dart create**

คำสั่งพื้นฐานมีดังนี้:

```bash
dart create [ชื่อโปรเจกต์]
```

**ตัวอย่าง: การสร้างโปรเจกต์ Console**

หากต้องการสร้างโปรเจกต์ชื่อ `my_console_app` ให้รันคำสั่ง:

```bash
dart create my_console_app
```

จากนั้น Dart จะสร้างโฟลเดอร์ `my_console_app` พร้อมโครงสร้างไฟล์ดังนี้:

```
my_console_app/
├── .gitignore
├── analysis_options.yaml
├── pubspec.yaml
├── README.md
├── bin/
│   └── my_console_app.dart  # ไฟล์หลักของโปรแกรม
├── lib/
│   └── my_console_app.dart  # ไฟล์ Library ของโปรเจกต์
└── test/
    └── my_console_app_test.dart
```

**ตัวอย่าง: การเลือก Template อื่นๆ**

หากต้องการระบุประเภทของโปรเจกต์ ให้ใช้ Flag `-t` (หรือ `--template`)

```bash
# สร้างโปรเจกต์ Web
dart create -t web my_web_app

# สร้างโปรเจกต์ Server
dart create -t server my_server_app
```

**3. เข้าสู่โฟลเดอร์โปรเจกต์และรันโปรแกรม**

หลังจากสร้างโปรเจกต์แล้ว ให้เข้าไปในโฟลเดอร์ที่สร้างขึ้น:

Bash

```
cd my_console_app
```

แล้วใช้คำสั่ง dart run เพื่อรันโปรแกรม:

Bash

```
dart run
```

เมื่อรันคำสั่งนี้ โปรแกรมจะรันไฟล์หลักที่อยู่ใน bin/ และแสดงผลลัพธ์บนหน้าจอ

---

dart create ช่วยให้คุณเริ่มต้นโปรเจกต์ใหม่ได้อย่างรวดเร็วและเป็นมาตรฐาน ทำให้คุณสามารถโฟกัสกับการเขียนโค้ดได้ทันทีโดยไม่ต้องเสียเวลาไปกับการตั้งค่าเบื้องต้นครับ

จากเนื้อหาเรื่อง **dart create** ที่เราได้เรียนรู้กัน ได้เตรียมตัวอย่างโปรแกรมทั้งแบบพื้นฐานและแบบประยุกต์มาให้ เพื่อให้เห็นภาพการสร้างโปรเจกต์และใช้งานจริง

---

**1. ตัวอย่างโปรแกรมพื้นฐาน**

โปรแกรมเหล่านี้จะแสดงการสร้างโปรเจกต์ด้วยคำสั่ง dart create และการรันโปรแกรมพื้นฐานที่สร้างขึ้นมาให้

***โปรแกรมที่ 1: การสร้างและรันโปรเจกต์ Console App พื้นฐาน***

**คำสั่ง:**

Bash

```
dart create -t console my_first_app
```

โครงสร้างไฟล์:

เมื่อรันคำสั่งด้านบน Dart จะสร้างโฟลเดอร์ my_first_app พร้อมโครงสร้างดังนี้

```
my_first_app/
├── bin/
│   └── my_first_app.dart  # ไฟล์หลักของโปรแกรม
└── pubspec.yaml
└── ...
```

โค้ดเต็มไฟล์ (bin/my_first_app.dart):

Dart จะสร้างโค้ดเริ่มต้นมาให้ ซึ่งเป็นโปรแกรมที่รับค่าจากอาร์กิวเมนต์แล้วแสดงผล

Dart

```
import 'package:my_first_app/my_first_app.dart' as my_first_app;
```

```
void main(List<String> arguments) {
  print('Hello world: ${my_first_app.calculate()}!');
}
```

**คำอธิบายโค้ด:**

- **import**: นำเข้าไฟล์ Library ที่อยู่ในโฟลเดอร์ lib/ เพื่อนำฟังก์ชัน calculate() มาใช้
- **main**: ฟังก์ชันหลักที่โปรแกรมจะเริ่มทำงาน
- **arguments**: ตัวแปรสำหรับรับค่าจาก Command-line (ถ้ามี)
- **print**: แสดงผลลัพธ์ออกทางหน้าจอ

**การรัน:**

Bash

```
cd my_first_app
dart run
```

**ผลการรัน:**

```
Hello world: 42!
```

---

***โปรแกรมที่ 2: การสร้างและรันโปรเจกต์ Web App***

**คำสั่ง:**

Bash

```
dart create -t web my_web_app
```

โครงสร้างไฟล์:

เมื่อรันคำสั่งด้านบน Dart จะสร้างโฟลเดอร์ my_web_app พร้อมโครงสร้างที่เหมาะสมกับการพัฒนาเว็บ

```
my_web_app/
├── web/
│   ├── index.html       # ไฟล์ HTML หลัก
│   └── main.dart        # ไฟล์ Dart หลักของเว็บ
└── pubspec.yaml
└── ...
```

โค้ดเต็มไฟล์ (web/main.dart):

Dart จะสร้างโค้ดเริ่มต้นที่จัดการกับการคลิกปุ่มบนหน้าเว็บ

Dart

```
import 'dart:html';

void main() {
```

```
  querySelector('#output')?.text = 'Your Dart app is running.';
}
```

**คำอธิบายโค้ด:**

- **import 'dart:html'**: นำเข้า Library สำหรับจัดการ DOM (Document Object Model) ของ HTML
- **main**: ฟังก์ชันหลักสำหรับเว็บแอป
- **querySelector('#output')**: ค้นหา Element HTML ที่มี ID เป็น "output"
- **.text = ...**: เปลี่ยนข้อความของ Element นั้น

**การรัน:**

```
Bash
cd my_web_app
dart run
```

**ผลการรัน:**

- คำสั่ง dart run จะเปิด Development server ขึ้นมา
- คุณสามารถเปิดเบราว์เซอร์ไปที่ **http://localhost:8080** เพื่อดูผลลัพธ์
- หน้าเว็บจะแสดงข้อความ "Your Dart app is running."

---

***โปรแกรมที่ 3: การสร้างและรันโปรเจกต์ Server***

**คำสั่ง:**

```
Bash
dart create -t server my_server_app
```

โครงสร้างไฟล์:

เมื่อรันคำสั่งด้านบน Dart จะสร้างโฟลเดอร์ my_server_app พร้อมโครงสร้างที่เหมาะสมสำหรับการพัฒนา Server

```
my_server_app/
├── bin/
│   └── server.dart      # ไฟล์หลักของ Web server
└── pubspec.yaml
└── ...
```

โค้ดเต็มไฟล์ (bin/server.dart):

Dart จะสร้างโค้ดเริ่มต้นที่ใช้แพ็กเกจ shelf สำหรับสร้าง Web server ที่รับ HTTP Request

```
Dart
import 'dart:io';
import 'package:shelf/shelf.dart';
import 'package:shelf/shelf_io.dart';
```

```
void main(List<String> args) async {
  final ip = InternetAddress.anyIPv4;
  final handler = const Pipeline().addMiddleware(logRequests()).addHandler(_echoRequest);
  final port = int.parse(Platform.environment['PORT'] ?? '8080');
  final server = await serve(handler, ip, port);

  print('Server listening on http://${server.address.host}:${server.port}');
}

Response _echoRequest(Request request) => Response.ok('Request for "${request.url}"');
```

**คำอธิบายโค้ด:**

- **import**: นำเข้า Library สำหรับจัดการ IO (dart:io) และ Web server (shelf)
- **main**: สร้าง Pipeline เพื่อรับและจัดการ Request จากเบราว์เซอร์
- **serve**: เริ่มต้น Server ที่ localhost:8080
- **_echoRequest**: ฟังก์ชันสำหรับตอบกลับ Request ด้วยข้อความ "Request for..."

**การรัน:**

```
Bash
cd my_server_app
dart run
```

**ผลการรัน:**

- คำสั่ง dart run จะเปิด Server ขึ้นมา
- Terminal จะแสดงข้อความ "Server listening on http://0.0.0.0:8080"
- เมื่อเปิดเบราว์เซอร์ไปที่ **http://localhost:8080** จะเห็นข้อความ "Request for "/"
- หากเปิดเบราว์เซอร์ไปที่ **http://localhost:8080/hello** จะเห็นข้อความ "Request for "/hello"

---

**2. ตัวอย่างโปรแกรมประยุกต์**

โปรแกรมเหล่านี้จะแสดงการใช้โปรเจกต์ที่สร้างด้วย dart create เพื่อแก้ปัญหาในชีวิตจริง

***โปรแกรมที่ 4: การสร้าง CLI (Command-Line Interface) สำหรับแปลงหน่วย***

**คำสั่ง:**

```
Bash
dart create -t console unit_converter
```

**โค้ดเต็มไฟล์ (bin/unit_converter.dart):**

```
Dart
```

```dart
import 'dart:io';

void main(List<String> arguments) {
  if (arguments.length != 2) {
    print('การใช้งาน: dart run <จำนวน> <หน่วย>');
    print('ตัวอย่าง: dart run 10 km');
    return;
  }

  final value = double.tryParse(arguments[0]);
  final unit = arguments[1].toLowerCase();

  if (value == null) {
    print('กรุณาใส่ตัวเลขที่ถูกต้อง');
    return;
  }

  switch (unit) {
    case 'km':
      print('$value กิโลเมตร เท่ากับ ${value * 1000} เมตร');
      break;
    case 'm':
      print('$value เมตร เท่ากับ ${value / 1000} กิโลเมตร');
      break;
    default:
      print('ไม่รองรับหน่วย "$unit"');
  }
}
```

**คำอธิบายโค้ด:**

- โปรแกรมจะตรวจสอบว่ามีอาร์กิวเมนต์ 2 ตัวหรือไม่ (จำนวนและหน่วย)
- ใช้ double.tryParse เพื่อแปลง String เป็นตัวเลขอย่างปลอดภัย
- ใช้ switch เพื่อแปลงหน่วยตามที่ผู้ใช้ระบุ

**การรัน:**

Bash

```
cd unit_converter
dart run bin/unit_converter.dart 5 km
dart run bin/unit_converter.dart 2000 m
dart run bin/unit_converter.dart 10 inch
```

**ผลการรัน:**

```
5.0 กิโลเมตร เท่ากับ 5000.0 เมตร
2000.0 เมตร เท่ากับ 2.0 กิโลเมตร
ไม่รองรับหน่วย "inch"
```

---

***โปรแกรมที่ 5: การสร้าง Server สำหรับ API ข้อมูลสินค้า***

**คำสั่ง:**

```
Bash
dart create -t server product_api
```

**โค้ดเต็มไฟล์ (bin/server.dart):**

```dart
import 'dart:io';
import 'dart:convert';
import 'package:shelf/shelf.dart';
import 'package:shelf/shelf_io.dart';
import 'package:shelf_router/shelf_router.dart';

// Router สำหรับจัดการ Endpoint
final _router = Router()
  ..get('/', _rootHandler)
  ..get('/products', _productsHandler);

Response _rootHandler(Request request) => Response.ok('Product API is running.');

Response _productsHandler(Request request) {
  final products = [
    {'id': 1, 'name': 'Laptop', 'price': 25000},
    {'id': 2, 'name': 'Mouse', 'price': 500},
    {'id': 3, 'name': 'Keyboard', 'price': 1200},
  ];
```

```
  return Response.ok(jsonEncode(products), headers: {'Content-Type': 'application/json'});
}

void main(List<String> args) async {
  final ip = InternetAddress.anyIPv4;
  final handler = const Pipeline().addMiddleware(logRequests()).addHandler(_router);
  final port = int.parse(Platform.environment['PORT'] ?? '8080');
  final server = await serve(handler, ip, port);

  print('Server listening on http://${server.address.host}:${server.port}');
}
```

**คำอธิบายโค้ด:**

- โปรเจกต์นี้ใช้แพ็กเกจ shelf_router เพื่อจัดการ Routing
- **_productsHandler**: เป็น Endpoint ที่ตอบกลับด้วยข้อมูลสินค้าในรูปแบบ JSON
- ใช้ jsonEncode เพื่อแปลง List ของ Map ให้เป็น String JSON
- กำหนด Content-Type ใน Header เพื่อบอกเบราว์เซอร์ว่าข้อมูลเป็น JSON

**การรัน:**

```
Bash
cd product_api
# เพิ่ม shelf_router ใน pubspec.yaml ก่อน
# แล้ว dart pub get
dart run bin/server.dart
```

**ผลการรัน:**

- เมื่อเปิดเบราว์เซอร์ไปที่ **http://localhost:8080/products**
- จะได้รับข้อมูล JSON ดังนี้:

```
[{"id":1,"name":"Laptop","price":25000},{"id":2,"name":"Mouse","price":500},{"id":3,"name":"Keyboard","price":1200}]
```

## การจัดโครงสร้างไฟล์และโฟลเดอร์

การจัดโครงสร้างไฟล์และโฟลเดอร์ในโปรเจกต์ Dart มีความสำคัญอย่างยิ่งในการทำให้โค้ดเป็นระเบียบ, จัดการง่าย, และทำงานร่วมกับผู้อื่นได้อย่างมีประสิทธิภาพ โครงสร้างมาตรฐานที่สร้างโดย dart create เป็นแนวทางปฏิบัติที่ดีที่สุด (Best Practices) ที่ชุมชน Dart ยอมรับกัน

***โครงสร้างมาตรฐานของโปรเจกต์ Console***

โปรเจกต์ที่สร้างด้วย dart create -t console จะมีโครงสร้างดังนี้:

```
my_app/
├── bin/
│   └── main.dart              # ไฟล์หลักของโปรแกรม (Executable)
├── lib/
│   └── my_app.dart            # ไฟล์ Library หลักของโปรเจกต์
├── test/
│   └── my_app_test.dart       # ไฟล์สำหรับ Unit tests
├── pubspec.yaml               # ไฟล์จัดการ Dependency และข้อมูลโปรเจกต์
├── analysis_options.yaml      # ไฟล์สำหรับตั้งค่า Code Analyzer
└── README.md                  # ข้อมูลทั่วไปของโปรเจกต์
```

***หน้าที่ของแต่ละโฟลเดอร์และไฟล์***

1. **bin/**:
   - โฟลเดอร์นี้มีไว้สำหรับเก็บ **ไฟล์ที่สามารถรันได้** (Executable) เท่านั้น
   - ไฟล์ในโฟลเดอร์นี้มักจะมีฟังก์ชัน main() ซึ่งเป็นจุดเริ่มต้นของโปรแกรม
   - เมื่อคุณรันคำสั่ง dart run, Dart จะมองหาไฟล์ในโฟลเดอร์นี้เพื่อรัน
2. **lib/**:
   - นี่คือหัวใจของโปรเจกต์ โฟลเดอร์นี้ใช้สำหรับเก็บ **โค้ด Library** ของคุณ
   - โค้ดทั้งหมดที่ไม่ได้อยู่ใน main() ควรอยู่ในโฟลเดอร์นี้เพื่อสามารถนำไปใช้ซ้ำได้
   - ไฟล์ my_app.dart ในโฟลเดอร์นี้มักจะทำหน้าที่เป็น "ไฟล์หลัก" ของ Library เพื่อ export หรือ import ไฟล์ย่อยอื่นๆ
   - การแบ่งโค้ดเป็นไฟล์ย่อยๆ ใน lib/ (เช่น lib/src/, lib/models/) ช่วยให้โค้ดของคุณเป็นระเบียบมากขึ้น
3. **test/**:
   - โฟลเดอร์นี้มีไว้สำหรับเก็บ **Unit tests** ซึ่งเป็นโค้ดที่ใช้ตรวจสอบการทำงานของโค้ดในโฟลเดอร์ lib/
   - การเขียน Test ช่วยให้คุณมั่นใจว่าโค้ดทำงานถูกต้องตามที่คาดหวัง
4. **pubspec.yaml**:
   - ไฟล์สำคัญที่ทำหน้าที่เป็น **ไฟล์ Manifest** ของโปรเจกต์
   - ใช้สำหรับกำหนดข้อมูลสำคัญของโปรเจกต์ เช่น ชื่อ, เวอร์ชั่น, และแพ็กเกจที่ต้องการใช้ (Dependencies)
   - เมื่อคุณเพิ่มแพ็กเกจใหม่ในไฟล์นี้ เช่น http: ^0.13.0, คุณจะต้องรันคำสั่ง dart pub get เพื่อดาวน์โหลดแพ็กเกจนั้น
5. **analysis_options.yaml**:

- ไฟล์นี้ใช้สำหรับตั้งค่า **Code Analyzer** เพื่อกำหนดกฎในการเขียนโค้ด
- คุณสามารถกำหนด Linter rules ที่ต้องการเพื่อให้โค้ดของคุณมีคุณภาพสูงและเป็นไปตามมาตรฐาน
- ตัวอย่าง: กำหนดให้โปรแกรมแจ้งเตือนเมื่อมีการใช้ตัวแปรที่ไม่ได้ใช้

จากเนื้อหาเรื่อง **การจัดโครงสร้างไฟล์และโฟลเดอร์** ที่เราได้เรียนรู้กัน ได้เตรียมตัวอย่างโปรแกรมทั้งแบบพื้นฐานและแบบประยุกต์มาให้ เพื่อให้เห็นภาพการจัดระเบียบโค้ดในโปรเจกต์จริง

---

**1. ตัวอย่างโปรแกรมพื้นฐาน**

โปรแกรมเหล่านี้จะแสดงการจัดโครงสร้างไฟล์และโฟลเดอร์ตามมาตรฐานของ Dart พร้อมคำอธิบายว่าไฟล์แต่ละส่วนมีหน้าที่อะไร

***โปรแกรมที่ 1: โปรเจกต์ Console App พื้นฐาน***

โครงสร้างไฟล์:

โปรเจกต์นี้จะสร้างโดยใช้คำสั่ง dart create -t console my_app_basic

```
my_app_basic/
├── bin/
│   └── main.dart              # จุดเริ่มต้นของโปรแกรม
├── lib/
│   └── calculator.dart        # โค้ดสำหรับคำนวณ
├── test/
│   └── calculator_test.dart   # โค้ดสำหรับทดสอบ
└── pubspec.yaml
```

**โค้ดเต็มไฟล์:**

- **lib/calculator.dart**:

```dart
int add(int a, int b) {
  return a + b;
}
```

- **bin/main.dart**:

```dart
import 'package:my_app_basic/calculator.dart';

void main(List<String> arguments) {
  final result = add(5, 3);
  print('ผลรวม: $result');
```

```
}
```

**คำอธิบายโค้ด:**

- **lib/calculator.dart**: ไฟล์นี้ทำหน้าที่เป็น **Library** โดยเก็บฟังก์ชัน add ที่สามารถนำไปใช้ซ้ำได้
- **bin/main.dart**: ไฟล์นี้คือ **Executable** ซึ่งเป็นจุดเริ่มต้นของโปรแกรม มีหน้าที่เรียกใช้ฟังก์ชัน add จากไฟล์ lib/ เพื่อคำนวณและแสดงผล

**การรัน:**

Bash

```
cd my_app_basic
dart run
```

**ผลการรัน:**

```
ผลรวม: 8
```

---

***โปรแกรมที่ 2: การใช้ Class ในไฟล์แยก***

โครงสร้างไฟล์:

โปรเจกต์นี้จะสร้างโดยใช้คำสั่ง dart create -t console user_profile_app

```
user_profile_app/
├── bin/
│   └── main.dart
├── lib/
│   └── models/
│       └── user.dart        # Class สำหรับข้อมูลผู้ใช้
│   └── user_profile_app.dart # ไฟล์หลักของ Library
└── pubspec.yaml
```

**โค้ดเต็มไฟล์:**

- **lib/models/user.dart**:

Dart

```
class User {
  String name;
  int age;

  User({required this.name, required this.age});
}
```

- **lib/user_profile_app.dart**:

Dart

```
export 'models/user.dart'; // Export เพื่อให้ไฟล์อื่น Import ได้ง่าย
```

- **bin/main.dart**:

Dart

```
import 'package:user_profile_app/user_profile_app.dart';

void main(List<String> arguments) {
  final user = User(name: 'Alice', age: 30);
  print('ชื่อผู้ใช้: ${user.name}, อายุ: ${user.age}');
}
```

**คำอธิบายโค้ด:**

- **lib/models/user.dart**: ไฟล์นี้เก็บ Class ที่เป็น Model ของข้อมูลผู้ใช้
- **lib/user_profile_app.dart**: ไฟล์นี้ใช้คำสั่ง export เพื่อรวมไฟล์ user.dart เข้าด้วยกัน ทำให้ไฟล์ main.dart สามารถ import ได้ง่ายขึ้น
- **bin/main.dart**: ไฟล์หลักที่สร้าง Object จาก Class User ที่ถูก Import มาใช้งาน

**การรัน:**

Bash

```
cd user_profile_app
dart run
```

**ผลการรัน:**

```
ชื่อผู้ใช้: Alice, อายุ: 30
```

---

***โปรแกรมที่ 3: การจัดการ Configuration File***

โครงสร้างไฟล์:

โปรเจกต์นี้จะสร้างโดยใช้คำสั่ง dart create -t console config_app

```
config_app/
├── bin/
│   └── main.dart
├── lib/
│   └── config.dart          # โค้ดสำหรับจัดการ Configuration
└── pubspec.yaml
```

**โค้ดเต็มไฟล์:**

- **lib/config.dart**:

Dart

```
class AppConfig {
  static const String appName = 'My App';
  static const int version = 1;
}
```

- **bin/main.dart**:

```
Dart
import 'package:config_app/config.dart';

void main(List<String> arguments) {
  print('ชื่อโปรแกรม: ${AppConfig.appName}');
  print('เวอร์ชัน: ${AppConfig.version}');
}
```

**คำอธิบายโค้ด:**

- **lib/config.dart**: ไฟล์นี้ใช้เก็บค่าคงที่ (Constants) ของโปรแกรม ทำให้สามารถจัดการค่าเหล่านี้ได้จากไฟล์เดียว
- **bin/main.dart**: ไฟล์หลักที่เรียกใช้ค่าคงที่จาก lib/config.dart

**การรัน:**

```
Bash
cd config_app
dart run
```

**ผลการรัน:**

```
ชื่อโปรแกรม: My App
เวอร์ชัน: 1
```

---

**2. ตัวอย่างโปรแกรมประยุกต์**

โปรแกรมเหล่านี้จะแสดงการจัดโครงสร้างไฟล์ในโปรเจกต์ที่มีความซับซ้อนขึ้น โดยมีการแบ่งโค้ดเป็นส่วนๆ เพื่อให้ง่ายต่อการจัดการ

***โปรแกรมที่ 4: ระบบจัดการสินค้าขนาดเล็ก***

โครงสร้างไฟล์:

โปรเจกต์นี้จะสร้างโดยใช้คำสั่ง dart create -t console product_manager

```
product_manager/
├── bin/
│   └── main.dart
├── lib/
```