สร้างเว็บแอป To-Do List ฉบับเข้าใจง่าย!

สรุปครบ จบในที่เดียว: จาก Requirements สู่ Database (Flask + SQLite)

วางแผนก่อนโค้ด: การจัดการ Requirements

ไอเดียเยอะ
แต่ไม่รู้จะเริ่มตรงไหน?

Functional Requirements:
สิ่งที่ระบบ ‘ต้องทำได้’

MoSCoW Method

- เน้นทำ Must Have (สำคัญมาก ขาดไม่ได้ เช่น Login, Add/Edit Task)
- จัดลำดับสิ่งที่ ควรมี (Should Have)

Non-Functional
คุณภาพที่ ‘ซ่อนอยู่’

Speed Security

- โหลดหน้าเว็บไว ภายใน 2 วินาที
- ความปลอดภัยสูง (เข้ารหัส Password ด้วย bcrypt)

User Stories: มองแอปจากมุมมองผู้ใช้

As a
[ใคร]

I want to
[ทำอะไร]

So that
[เพื่ออะไร]

ในฐานะ **ผู้ใช้ใหม่**
ฉันต้องการ **สมัครสมาชิก** เพื่อ
สร้างบัญชีของตัวเอง

ในฐานะ **สมาชิก**
ฉันต้องการ **เพิ่ม Task** เพื่อ
จดบันทึกสิ่งที่ต้องทำ

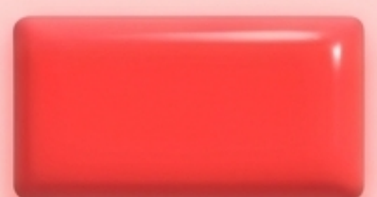
⭐ **Acceptance Criteria = เงื่อนไขที่บอกว่าฟีเจอร์นี้ ‘เสร็จสมบูรณ์’ จริงๆ**

Flowchart 101: สัญลักษณ์แผนผังระบบ

วงรี (Terminal):
จุดเริ่มต้น / สิ้นสุดเสมอ

สี่เหลี่ยมสีเขียว (CRUD):
การกระทำกับข้อมูล
(Add/Edit/Toggle Task)

สี่เหลี่ยมสีฟ้า (Process):
ขั้นตอนการทำงาน / หน้าเว็บ
(เช่น Login, Dashboard)



สี่เหลี่ยมสีแดง
(Destructive): ลบข้อมูล
(Delete Task)

ข้าวหลามตัดสีเหลือง
(Decision): เงื่อนไขที่มี 2
ทางออกเสมอ (Yes/No)

Pro-Tip:
จำไว้! ข้าวหลามตัด (Decision)
ต้องมี 2 เส้นออกเสมอ
ถ้ามีเส้นเดียวแปลว่า Logic ยังไม่ครบ!

The Master Plan: Flowchart ของระบบ To-Do List

เริ่ม

หน้า Login

รหัสถูกไหม?

ไม่ใช่

ใช่

เข้าสู่หน้าหลัก (Dashboard)

Add Task

Edit Task

Toggle Done

Delete Task

Logout

สิ้นสุด

การวาด Flowchart ทำให้เราเห็นภาพรวม ก่อนเขียนโค้ด ป้องกันการลืม Logic สำคัญ!

จากไอเดียสู่หน้าจอ: UX vs UI Design

Step 1: Sketch (UX - โครงสร้าง)

กระดาษ + ปากกา
เน้นการจัดวางและการใช้งานพื้นฐาน

Step 2: Wireframe (โครงร่างดิจิทัล)

จัดลำดับความสำคัญของข้อมูลและ
ปุ่มต่างๆ (Boxes & Text)

Step 3: UI Prototype (หน้าตาที่สมบูรณ์)

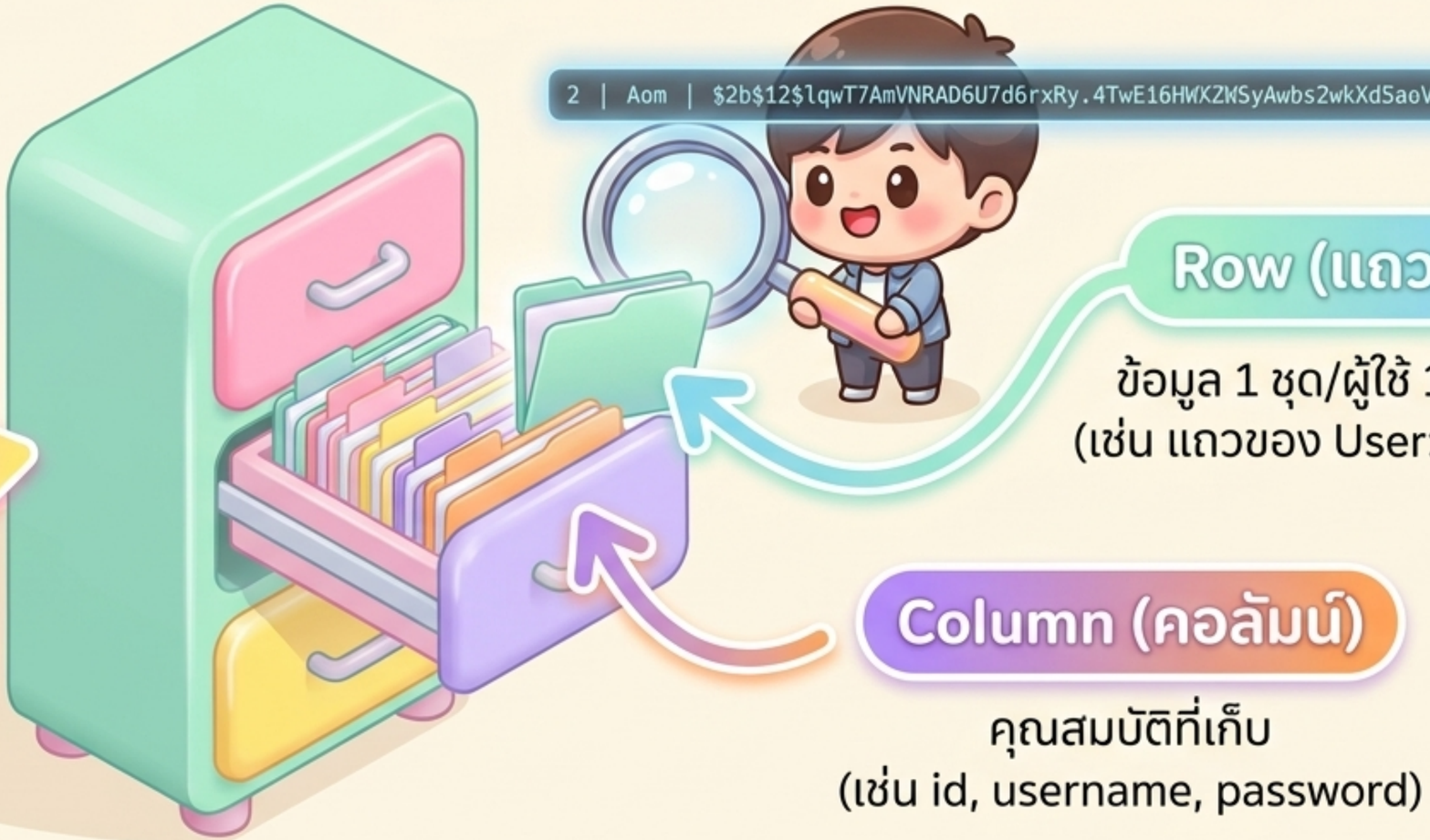
ใส่สีสัน Font และ Interaction
ให้สวยงามพร้อมใช้งาน

Database 101: ห้องเก็บข้อมูลของแอป

ฐานข้อมูล (Database) คือที่เก็บข้อมูลที่มีโครงสร้าง คล้ายตาราง Excel แต่ค้นหาและเชื่อมโยงได้เร็วกว่า

Table (ตาราง)

ตู้เอกสารทั้งตู้
(เช่น ตาราง users)



Row (แถว)

ข้อมูล 1 ชุด/ผู้ใช้ 1 คน
(เช่น แถวของ User: Aom)

Column (คอลัมน์)

คุณสมบัติที่เก็บ
(เช่น id, username, password)

Data Types & Constraints: กฎกติกาของข้อมูล

INTEGER

ตัวเลขจำนวนเต็ม (เช่น 1, 42)

TEXT

ข้อความ (เช่น ‘ทำตารางงาน’)

BOOLEAN

จริง/เท็จ (SQLite ใช้ 0/1)

Constraints (กฎกติกา)

PRIMARY KEY 🔑:
รหัสประจำตัว
ต้องไม่ซ้ำกัน (ID)

FOREIGN KEY 🔗:
กุญแจเชื่อมโยงอ้างอิง
ข้อมูลตารางอื่น

UNIQUE 🌟:
ห้ามซ้ำกันเด็ดขาด
(เช่น Username)

NOT NULL 🚫:
ช่องนี้ห้ามปล่อยว่าง

สติ๊กเกอร์กฎเหล่านี้เอาไว้แปะบนบล็อกข้อมูลเพื่อกำหนดกฎ

ER Diagram: เชื่อมโยงข้อมูลตารางเข้าด้วยกัน

User 1 คน มีได้หลาย Tasks
แต่ละ Task เป็นของ User คนเดียวเท่านั้น
(เชื่อมกันด้วย user_id)

SQL (Structured Query Language): เวทมนตร์สั่งการ Database

CREATE:
สร้างตาราง
ข้อมูลใหม่

INSERT:
เพิ่มข้อมูลใหม่
ลงในตาราง

SELECT:
ดึง/ค้นหา
ข้อมูลออกมาดู

UPDATE:
แก้ไขอัปเดต
ข้อมูลที่มีอยู่แล้ว

DELETE:
ลบข้อมูลทิ้ง

WHERE:
กรองเงื่อนไข
เฉพาะที่ต้องการ

โครงสร้างแอป: Frontend vs Backend

การสื่อสารผ่าน HTTP: GET vs POST

GET Request (ขอข้อมูล)

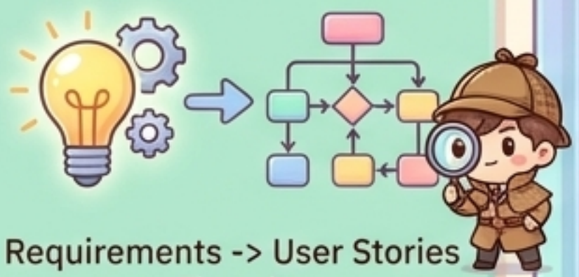
Browser ทำการขอหน้าเว็บจาก Server -> Server ส่งไฟล์ HTML กลับมาให้แสดงผล

POST Request (ส่งข้อมูล)

Browser ส่งข้อมูลที่ User กรอก (เช่น ฟอร์ม Login/Add Task) ไปให้ Server -> Server ประมวลผลและบันทึกลง Database

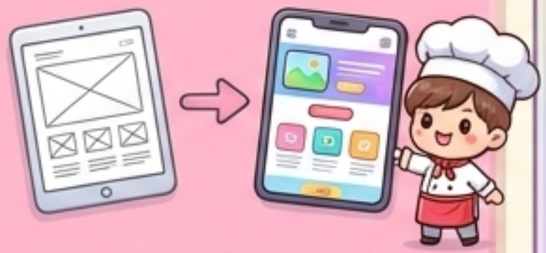
สรุปภาพรวม: The Developer's Journey

Idea & Logic



Requirements -> User Stories
-> Flowchart
(สัญลักษณ์เชื่อมโยงระบบ)

Frontend



Sketch -> Wireframe -> UI
(หน้าร้านที่ผู้ใช้มองเห็น)

Backend & Data

Flask (ครัว) เชื่อมต่อกับ
SQLite (ตู้เอกสาร) ผ่านคำสั่ง SQL

แอปพลิเคชันที่สมบูรณ์เกิดจากการผสานกันระหว่าง
ประสบการณ์ที่ดีของผู้ใช้ (UX/UI) และ
โครงสร้างข้อมูลที่แข็งแรง (Database & Logic)!