เอกสารประกอบโปรเจกต์: Toolsmith

1. ภาพรวมและวัตถุประสงค์ (Project Overview)

Toolsmith คือเว็บแอปพลิเคชันที่ถูกสร้างขึ้นเพื่อปฏิวัติกระบวนการสร้างโจทย์ปัญหาสำหรับการแข่งขันเขียนโปรแกรม (Competitive Programming) โดยใช้ประโยชน์จาก AI (Google Gemini) ในการสร้างเนื้อหาโจทย์, โค้ดเฉลย, และชุดข้อมูลทดสอบ (Test Cases) ทั้งหมดโดยอัตโนมัติ

ผู้ใช้สามารถกรอกเพียงหัวข้อและรายละเอียดเบื้องต้นของโจทย์ที่ต้องการ จากนั้นระบบจะทำการสร้างไฟล์ทั้งหมดที่จำเป็น, แสดงตัวอย่างให้ผู้ใช้ตรวจสอบ, และอัปโหลดไฟล์เหล่านั้นไปยัง Supabase Storage เพื่อจัดเก็บอย่างเป็นระบบตามโครงสร้างที่กำหนดไว้ล่วงหน้า

2. เทคโนโลยีหลักที่ใช้ (Core Technologies)

- **Backend Framework**: **FastAPI** - สำหรับการสร้าง API ที่รวดเร็วและมีประสิทธิภาพ
- **AI Orchestration**: **LangGraph** - สำหรับการสร้างและจัดการ Workflow การทำงานของ AI อย่างเป็นขั้นตอนและมีสถานะ (Stateful)
- **AI Model**: **Google Gemini** - เป็นแกนหลักในการสร้างสรรค์เนื้อหาทั้งหมด
- **File Storage**: **Supabase Storage** - สำหรับการจัดเก็บไฟล์โจทย์ที่สร้างเสร็จแล้วในรูปแบบของ Bucket ที่มีโครงสร้างชัดเจน
- **Configuration**: **Pydantic** - สำหรับการจัดการการตั้งค่าและตัวแปรสภาพแวดล้อม (Environment Variables)
- **Frontend**: **HTML, CSS, JavaScript** (Vanilla) - สำหรับสร้างหน้าจอการใช้งานที่เรียบง่ายและตอบสนองได้ดี

---

3. สถาปัตยกรรมและโครงสร้างโปรเจกต์ (Architecture & Project Structure)

โปรเจกต์นี้ถูกออกแบบโดยแบ่งความรับผิดชอบของแต่ละส่วนออกจากกันอย่างชัดเจน เพื่อให้ง่ายต่อการบำรุงรักษาและพัฒนาต่อยอดในอนาคต

```
toolsmith_project/
├── app/
│   └── main.py              # จัดการ API Endpoints และเป็นจุดเริ่มต้นของเว็บเซิร์ฟเวอร์
│
├── core/
│   ├── graphs/
│   │   ├── generation_graph.py  # กำหนด State และ Workflow ของ LangGraph
│   │   └── nodes/               # โฟลเดอร์สำหรับเก็บ Logic ของแต่ละ Node ในกราฟ
│   │       ├── generation.py
│   │       └── file_creation.py
│   │
│   ├── models/
│   │   ├── pydantic_models.py # กำหนด Data Models สำหรับ Request
│   │   └── graph_models.py    # กำหนด State Model สำหรับ LangGraph
│   │
│   ├── services/
│   │   ├── config.py          # จัดการ Environment Variables
│   │   └── storage_service.py # จัดการการเชื่อมต่อและอัปโหลดไฟล์ไปยัง Supabase Storage
│   │
│   └── prompts/
│       └── task_generation_prompt.txt # เก็บ Prompt ที่ใช้สั่งงาน AI
│
├── static/
│   └── favicon.ico          # เก็บไฟล์ไอคอนสำหรับหน้าเว็บ
│
```

```
├── frontend/
│   ├── index.html          # โครงสร้างหน้าเว็บ
│   └── app.js              # Logic การทำงานของหน้าเว็บ
│
├── .env                    # ไฟล์เก็บข้อมูลสำคัญ (API Keys, URLs)
└── README.md               # เอกสารสรุปและวิธีการติดตั้ง
```

---

4. ขั้นตอนการทำงานของระบบ (Workflow)

ระบบมีขั้นตอนการทำงานที่ชัดเจนตั้งแต่ผู้ใช้เริ่มกรอกข้อมูลจนถึงการอัปโหลดไฟล์สำเร็จ

1. **ผู้ใช้กรอกข้อมูล (User Input)**: ผู้ใช้เปิดหน้า `index.html` และกรอกข้อมูล 3 ส่วน คือ "หัวข้อโจทย์", "จำนวนเทสเคส", และ "รายละเอียดเพิ่มเติม"
2. **ส่งคำขอสร้างตัวอย่าง (Generate Preview)**: เมื่อกดปุ่ม "Generate Problem" สคริปต์ `app.js` จะส่ง `POST` request ไปยัง endpoint `/generate-preview` ของ FastAPI
3. **AI ทำงานผ่าน LangGraph**:
   - `app/main.py` รับคำขอและส่งต่อไปยัง `GraphManager`
   - **Node 1: `generation.py`** : สร้าง Prompt จากข้อมูลที่ผู้ใช้กรอก แล้วส่งไปให้ Google Gemini เพื่อสร้างเนื้อหาทั้งหมดกลับมาเป็นข้อความยาวๆ หนึ่งก้อน
   - **Node 2: `file_creation.py`** : รับข้อความจาก AI, ทำความสะอาด (ลบ ```, ชื่อไฟล์, Prefix ที่ไม่ต้องการ), แยกเนื้อหาออกเป็นไฟล์ย่อยๆ, และทำการรันโค้ด `generate.py` เพื่อสร้างเทสเคสจริงๆ ขึ้นมา จากนั้นจึงจัดโครงสร้างไฟล์ทั้งหมดในรูปแบบที่พร้อมใช้งาน
4. **ส่งผลลัพธ์กลับไปที่ Frontend**: `app/main.py` นำข้อมูลไฟล์ที่ได้มาสร้างเป็นไฟล์ `.zip` ในหน่วยความจำ แล้วส่งกลับไปให้ `app.js`
5. **แสดงตัวอย่าง (Preview)**: `app.js` ใช้ไลบรารี `JSZip` ในการอ่านไฟล์ `.zip` ที่ได้รับมา และแสดงรายชื่อไฟล์ทั้งหมดให้ผู้ใช้เลือกดูเนื้อหาได้
6. **ผู้ใช้อนุมัติและอัปโหลด (Approve & Upload)**: เมื่อผู้ใช้กดปุ่ม "Approve and Upload to Database" สคริปต์ `app.js` จะส่ง `POST` request ไปยัง endpoint `/upload-task` พร้อมข้อมูลที่ผู้ใช้กรอกไว้ในตอนแรก
7. **อัปโหลดไปยัง Supabase Storage**:
   - `app/main.py` รับคำขอและสั่งให้ `GraphManager` **รันกระบวนการสร้างไฟล์ทั้งหมดใหม่อีกครั้ง** เพื่อความถูกต้องของข้อมูล
   - จากนั้นจึงส่งข้อมูลไฟล์ทั้งหมด (ชื่อ, path, เนื้อหา) ไปให้ `StorageService`
   - `storage_service.py` ทำการวนลูปอัปโหลดไฟล์แต่ละไฟล์ขึ้นไปบน Supabase Storage Bucket ที่ชื่อ `problems` โดยสร้างโฟลเดอร์ย่อยตามชื่อโจทย์

---

5. คำอธิบายส่วนประกอบหลัก (Component Deep Dive)

Frontend (`index.html` & `app.js`)

- เป็น Single Page Application (SPA) ที่เรียบง่าย
- **`index.html`** : ใช้ HTML และ CSS ในการสร้าง UI ที่สวยงามและใช้งานง่าย มีส่วนของฟอร์ม, ส่วนแสดงสถานะ, และส่วนแสดงตัวอย่างไฟล์ที่ซ่อนไว้
- **`app.js`** :
  - จัดการ Event Listener ของปุ่มต่างๆ
  - ใช้ `fetch` API ในการสื่อสารกับ Backend
  - ใช้ `JSZip` เพื่ออ่านและแสดงข้อมูลจากไฟล์ ZIP ที่ได้รับมาโดยไม่ต้องให้ผู้ใช้ดาวน์โหลดก่อน
  - มี Logic ในการเก็บข้อมูลที่ผู้ใช้กรอก (`lastRequestData`) ไว้เพื่อใช้ในการส่งคำขออัปโหลดในภายหลัง

Backend (`app/main.py`)

- ทำหน้าที่เป็น Gateway หลักของระบบ
- **`/generate-preview`** : Endpoint สำหรับการสร้างไฟล์ตัวอย่าง ส่งคืนเป็นไฟล์ ZIP

- **`/upload-task`** : Endpoint สำหรับการทำงานจริง จะรันกระบวนการสร้างไฟล์ใหม่ทั้งหมดแล้วอัปโหลดไปยัง Storage Bucket
- ใช้ Dependency Injection (`Depends`) ของ FastAPI ในการเรียกใช้ Service ต่างๆ

LangGraph Workflow (`core/graphs/`)

- เป็นหัวใจของการประมวลผล
- **`graph_models.py`** : กำหนด `GraphState` ซึ่งเปรียบเสมือน "หน่วยความจำ" ที่ส่งต่อระหว่าง Node
- **`generation_graph.py`** : กำหนดลำดับการทำงานของกราฟ (`generator` -> `file_creator`)
- **`nodes/`** :
  - **`generation.py`** : รับผิดชอบการสื่อสารกับ LLM เท่านั้น
  - **`file_creation.py`** : รับผิดชอบการประมวลผลข้อมูลที่ได้จาก LLM, การทำความสะอาดข้อมูล, และการรันโค้ดเพื่อสร้างเทสเคส

Services (`core/services/`)

- **`storage_service.py`** : เป็น Abstraction Layer สำหรับจัดการการอัปโหลดไฟล์ไปยัง Supabase โดยเฉพาะ ทำให้โค้ดส่วนอื่นไม่จำเป็นต้องรับรู้รายละเอียดการทำงานของ Supabase Storage
- **`config.py`** : ใช้ `pydantic-settings` ในการโหลดค่าจากไฟล์ `.env` มาใช้งานอย่างปลอดภัยและเป็นระบบ

---

6. การติดตั้งและใช้งาน (Setup & Run)

สามารถทำตามขั้นตอนทั้งหมดที่ระบุไว้ในไฟล์ **README.md** ซึ่งเป็นคู่มือฉบับย่อสำหรับผู้ใช้งานทั่วไปได้ทันที โดยมีขั้นตอนหลักคือการตั้งค่า Environment Variables, การสร้าง Storage Bucket บน Supabase และการรันเซิร์ฟเวอร์ด้วย `uvicorn`