

models.py

โค้ดนี้ใช้ **dataclass** และ **Enum** เพื่อสร้างโครงสร้างข้อมูลที่ใช้ในระบบ ได้แก่:

- **QuestionType (Enum)**: กำหนดประเภทของคำถาม ได้แก่
 - **SCIENCE** (วิทยาศาสตร์)
 - **GENERAL** (ความรู้ทั่วไป)
 - **EMOTIONAL** (คำถามเชิงอารมณ์)
- **Question (dataclass)**: เก็บข้อมูลของคำถามแต่ละข้อ ประกอบด้วย:
 - **id** (str): รหัสคำถาม
 - **type** (QuestionType): ประเภทของคำถาม
 - **text** (str): เนื้อหาคำถาม
 - **answer** (str): คำตอบของคำถาม
- **ResponseLog (dataclass)**: บันทึกการตอบของผู้ใช้แต่ละครั้ง ประกอบด้วย:
 - **question_id** (str): รหัสของคำถามที่ถูกลบ
 - **question_type** (QuestionType): ประเภทของคำถาม
 - **answer** (str): คำตอบของผู้ใช้
 - **emotion_level** (float): ระดับอารมณ์ของผู้ใช้
 - **timestamp** (datetime): เวลาที่เกิดการตอบ

2. EmotionCalculator (ตัวคำนวณอารมณ์)

คลาสนี้ใช้เพื่อติดตามระดับอารมณ์ของผู้ใช้และคำนวณอารมณ์จากประเภทของคำถาม

- **__init__**
 - กำหนดค่าเริ่มต้นของ **previous_emotion** เป็น 100
 - ใช้ **previous_type** เพื่อติดตามประเภทของคำถามล่าสุด
 - ใช้ **emotional_streak** เพื่อนับจำนวนครั้งที่มีการถามคำถามประเภท "อารมณ์" ติดต่อกัน
- **calculate_science_emotion**
 - หากคำถามก่อนหน้านี้เป็นประเภท **EMOTIONAL** และอารมณ์ต่ำกว่า 30 ให้สั้ค่าอารมณ์ระหว่าง 10-40
 - ถ้าไม่ใช่ ให้สั้ค่าอารมณ์ระหว่าง 50-80
- **calculate_general_emotion**
 - หากคำถามก่อนหน้านี้เป็น **SCIENCE** และอารมณ์ต่ำกว่า 60 ให้สั้ค่าอารมณ์ระหว่าง 30-60
 - ถ้าไม่ใช่ = ให้สั้ค่าอารมณ์ระหว่าง 70-100
- **calculate_emotional_emotion**
 - หากผู้ใช้ตอบคำถามอารมณ์ติดต่อกัน 3 ครั้งขึ้นไป จะสั้ค่าอารมณ์ระหว่าง 20-50
 - ถ้าไม่มีค่าอารมณ์ก่อนหน้านี้ จะเริ่มต้นที่ 100
 - หากมีค่าอารมณ์ก่อนหน้านี้ จะมีการเปลี่ยนแปลงแบบสั้ระหว่าง -10 ถึง +10
- **get_emotion**
 - ตรวจสอบประเภทของคำถามและคำนวณค่าอารมณ์ที่เหมาะสม
 - อัปเดตค่า **previous_emotion** และ **previous_type**

3. ChatbotDatabase (ฐานข้อมูลของแชทบอท)

ใช้สำหรับจัดเก็บและเรียกใช้คำถามที่บันทึกไว้ในไฟล์ JSON

- **__init__**
 - โหลดข้อมูลจากไฟล์ questions.json และเก็บไว้ใน self.questions
 - เก็บ self.logs เป็นรายการบันทึกการตอบของผู้ใช้
- **load_data**
 - โหลดข้อมูลจาก questions.json หากมีไฟล์อยู่
 - ถ้าไม่มีไฟล์ จะสร้างตัวอย่างข้อมูลใหม่และบันทึกลงไฟล์
- **save_data**
 - บันทึกคำถามทั้งหมดลง questions.json ในรูปแบบ JSON
- **get_random_question**
 - เลือกคำถามแบบสุ่มจากประเภทที่ระบุ
- **log_response**
 - เพิ่มบันทึกการตอบของผู้ใช้ลงใน self.logs
 - เรียก ChatLogger.log_chat เพื่อบันทึกลงไฟล์
- **get_average_emotions**
 - คำนวณค่าเฉลี่ยของระดับอารมณ์จากบันทึกการตอบของผู้ใช้

4. ChatLogger

บันทึกข้อมูล Chat ไปที่ chat_log.txt

- **log_chat**
 - เปิดไฟล์ chat_log.txt ในโหมด append
 - เขียนข้อมูลที่รวม timestamp, question_type, question_id และ emotion_level

view.py

1. การตั้งค่า UI เบื้องต้นใน Tkinter



โค้ดนี้ใช้ **Tkinter** ในการสร้าง GUI สำหรับแสดงแชทบอท โดยมีการจัดการกับหลายองค์ประกอบใน UI เช่น โครงสร้างของข้อความ คำถามประเภทต่างๆ และสถิติต่างๆ

- ตั้งชื่อหน้าต่างเป็น "AI Chatbot Birb-png"
- ตั้งขนาดหน้าต่างเป็น 800x600 และพื้นหลังเป็น #f0f2f5
- ใช้ **ttk.Frame** สำหรับทุกคอนเทนเนอร์หลักใน UI

2. ส่วนประกอบ UI

- พื้นที่แชท:
แบ่งออกเป็น 2 ส่วนหลักคือ ข้อความ (คำถามและคำตอบ) และ สถิติ
 - ข้อความ จะแสดงใน scrollable frame ด้วยการใช้ tk.Canvas เพื่อให้สามารถเลื่อนดูข้อความที่มีมากๆ ได้
 - สถิติ จะแสดงในแถบด้านขวาของหน้าจอ ประกอบด้วยค่าอารมณ์ปัจจุบันและค่าเฉลี่ยของคำถามแต่ละประเภท

3. สร้างและแสดงข้อความในบับเบิ้ล

- ข้อความที่แสดงในบับเบิ้ล:
 - USER: จะแสดงทางขวาโดยใช้ 
 - BOT: จะแสดงทางซ้ายโดยใช้  และมีการใช้ #e4e6eb เป็นสีพื้นหลัง
- **Timestamp** จะแสดงเวลาในรูปแบบ HH:MM ตรงข้ามกับข้อความในแต่ละบับเบิ้ล
- การเลื่อน Scrollbar: ใช้ self.canvas.yview_moveto(1) เพื่อเลื่อนแถบเลื่อน (Scrollbar) ไปที่ด้านล่างสุดอัตโนมัติเมื่อมีข้อความใหม่

4. ฟังก์ชันหลักและการตอบกลับ

- ฟังก์ชัน **on_generate_click()**:
เมื่อผู้ใช้คลิกที่ปุ่ม "ส่งคำถาม", ฟังก์ชันนี้จะเรียก **callback function** ที่ถูกกำหนดไว้จากภายนอก ซึ่งอาจจะเป็นฟังก์ชันที่ดึงข้อมูลคำถามและคำตอบ
- ฟังก์ชัน **update_response()**:
อัปเดตข้อความแชทในหน้าจอเมื่อคำถามและคำตอบใหม่ได้รับการส่ง โดยแสดงข้อความของผู้ใช้และบอท รวมถึงการอัปเดตค่าระดับอารมณ์ที่ได้หลังจากการส่งคำถาม
- ฟังก์ชัน **update_stats()**:
อัปเดตสถิติการตอบคำถามตามประเภทต่างๆ เช่น คำรวม, วิทยาศาสตร์, ความรู้ทั่วไป และคำถามเชิงอารมณ์
 - ใช้ self.stats_vars เพื่อแสดงผลข้อมูลสถิติที่อัปเดตในแถบด้านขวาของ UI

5. การเลือกประเภทคำถาม

- ฟังก์ชัน **get_selected_type()**:
ฟังก์ชันนี้จะคืนค่าประเภทคำถามที่ผู้ใช้เลือกจากตัวเลือกใน UI โดยใช้ tk.StringVar() สำหรับเก็บประเภทคำถามที่เลือก

6. สร้างปุ่มและการเลือกประเภทคำถาม

- ปุ่ม "สุ่มคำถาม":

เมื่อผู้ใช้คลิกที่ปุ่มนี้จะทำการสุ่มคำถามจากประเภทที่เลือกไว้ โดยการใช้ `ttk.Button` และสไตล์ `Modern.TButton` สำหรับปุ่มที่ทันสมัย

- ตัวเลือกประเภทคำถาม:

ใช้ `ttk.Radiobutton` สำหรับให้ผู้ใช้เลือกประเภทคำถามจากสามประเภทที่กำหนด: วิทยาศาสตร์, ความรู้ทั่วไป และคำถามเชิงอารมณ์

controller.py

ทำหน้าที่เป็นตัวกลาง (Controller) สำหรับการควบคุมแอปพลิเคชัน **AI Chatbot** โดยมีการเชื่อมต่อระหว่าง **View** (UI) และ **Model** (การจัดการข้อมูล)

1. การกำหนดค่าเริ่มต้น (__init__ method)

เมื่อโปรแกรมเริ่มทำงาน ChatbotController จะสร้างอินสแตนซ์ของฐานข้อมูลและเครื่องมือคำนวณอารมณ์ รวมถึงตั้งค่าหน้าต่าง GUI โดยมีรายละเอียดดังนี้

รับค่าประเภทคำถามที่เลือกจาก UI

```
question_type = self.view.get_selected_type()
```

ได้ค่าหมวดหมู่ของคำถามที่ผู้ใช้เลือก (เช่น วิทยาศาสตร์, ความรู้ทั่วไป ฯลฯ)

ดึงคำถามจากฐานข้อมูล

```
question = self.db.get_random_question(question_type)
```

ค้นหาคำถามแบบสุ่มจากฐานข้อมูลตามประเภทที่เลือก

คำนวณระดับอารมณ์

```
emotion = self.emotion_calc.get_emotion(question_type)
```

ใช้ EmotionCalculator คำนวณระดับอารมณ์ที่เกี่ยวข้องกับประเภทของคำถาม

2. ฟังก์ชัน generate_response()

ทำหน้าที่ดึงคำถามจากฐานข้อมูลและคำนวณระดับอารมณ์ ก่อนอัปเดต UI และบันทึกข้อมูลลงฐานข้อมูล

กระบวนการทำงาน

1. รับค่าหมวดหมู่คำถามที่ผู้ใช้เลือกจาก UI
2. ดึงคำถามแบบสุ่มจากฐานข้อมูลตามหมวดหมู่ที่เลือก
3. ใช้เครื่องมือคำนวณอารมณ์เพื่อกำหนดระดับอารมณ์ที่เกี่ยวข้อง
4. บันทึกข้อมูลการตอบกลับลงในฐานข้อมูล โดยเก็บหมายเลขคำถาม, คำตอบ, ระดับอารมณ์ และเวลาที่ตอบ
5. อัปเดต UI เพื่อแสดงคำถาม, คำตอบ และระดับอารมณ์ของคำถามนั้น
6. คำนวณค่าเฉลี่ยของอารมณ์ทั้งหมดที่ถูกบันทึก และอัปเดต UI

3. ฟังก์ชัน run()

เป็นฟังก์ชันหลักที่ทำหน้าที่เปิดหน้าต่าง GUI และรอรับ **OUTPUT** จากผู้ใช้

4. เงื่อนไข if __name__ == "__main__":

ใช้ตรวจสอบว่าไฟล์ถูกเรียกใช้งานโดยตรงหรือไม่ หากใช่ จะสร้างอินสแตนซ์ของ ChatbotController และเรียกใช้ฟังก์ชัน run() เพื่อเริ่มทำงาน

GenerateData.py

โค้ดนี้ใช้ API จาก Open Trivia Database เพื่อดึงคำถาม โดยแต่ละประเภทของคำถามจะมีรหัส

"วิทยาศาสตร์" → รหัส 17 (Science & Nature)

"ความรู้ทั่วไป" → รหัส 9 (General Knowledge)

"คำถามเชิงอารมณ์" → รหัส 25 (Art, สื่อถึงอารมณ์ 😊)

2. ฟังก์ชัน generate_question_id()

ใช้สร้าง รหัสคำถามแบบ 5 หลัก โดยที่ตัวแรกต้องไม่เป็นศูนย์ เพื่อให้รหัสมีลักษณะคล้ายกับที่ใช้ในระบบฐานข้อมูลจริง

กระบวนการทำงาน:

- ใช้ random.randint(1, 9) เพื่อสร้างตัวเลขหลักแรกที่ไม่เป็น 0
- ใช้ random.randint(0, 9) อีก 4 ตัว เพื่อสร้างรหัสที่เหลือ

3. ฟังก์ชัน get_questions_from_api(amount, category_id)

ใช้ส่งคำขอ (requests.get) ไปยัง Open Trivia Database API เพื่อดึงคำถามจากประเภทที่ระบุ

กระบวนการทำงาน:

- สร้าง URL สำหรับ API ด้วยจำนวนคำถาม (amount) และรหัสหมวดหมู่ (category_id)
- ใช้ requests.get(url) เพื่อดึงข้อมูล
- ใช้ response.raise_for_status() เพื่อตรวจสอบว่า API ตอบกลับปกติ
- ค้นหา data.get("results", []) ซึ่งเป็นรายการคำถามที่ได้จาก API
- หากเกิดข้อผิดพลาด เช่น ไม่มีการเชื่อมต่ออินเทอร์เน็ต จะคืนค่า []

4. ฟังก์ชัน format_questions(raw_questions, category_name)

ใช้จัดรูปแบบ API ให้อยู่ JSON วนลูปผ่านคำถามที่ได้จาก API

- สร้างโครงสร้างใหม่ที่มี:
 - id → ใช้ generate_question_id()
 - type → ใช้ชื่อหมวดหมู่ที่รับเข้ามา (category_name)
 - text → เนื้อหาของคำถาม (q["question"])
 - answer → คำตอบที่ถูกต้อง (q["correct_answer"])
- RETURN คืนค่ารายการของคำถามที่จัดรูปแบบแล้ว

5. ฟังก์ชัน main()

ดึงคำถามจาก API และเก็บในไฟล์ questions.json

1. แสดงข้อความ "Loading..."
2. สร้างตัวแปร all_questions ที่เก็บคำถามทั้งหมดในรูปแบบ JSON
3. ดึงคำถามสำหรับแต่ละหมวดหมู่
 - วนลูปผ่าน QUESTION_CATEGORIES
 - ใช้ **get_questions_from_api()** เพื่อดึงคำถาม (สูงสุด **"input"** ข้อ)
 - หาก API ส่งกลับ [] (ไม่มีคำถาม) จะลองใหม่ไม่เกิน 3 ครั้ง โดยรอ 2 วินาที
 - ใช้ **format_questions()** เพื่อจัดรูปแบบข้อมูล
 - เพิ่มลงใน all_questions["questions"]
4. ตรวจสอบว่าแต่ละหมวดหมู่มีคำถามอย่างน้อย 10 ข้อ
 - หากไม่ครบ จะพิมพ์คำเตือน **"Warning: Only X questions found!"**
5. บันทึกคำถามลงไฟล์ questions.json
 - ใช้ json.dump() เพื่อเขียนข้อมูลเป็นไฟล์ JSON ที่อ่านง่าย (indent=2)
6. แสดงข้อความ **"finish"**

6. การเรียกใช้งาน (if __name__ == "__main__": main())

- ใช้เงื่อนไข if __name__ == "__main__": เพื่อให้โค้ดสามารถนำไปใช้เป็น โมดูล ได้
- หากรันไฟล์นี้โดยตรง จะเรียก main() เพื่อทำงาน