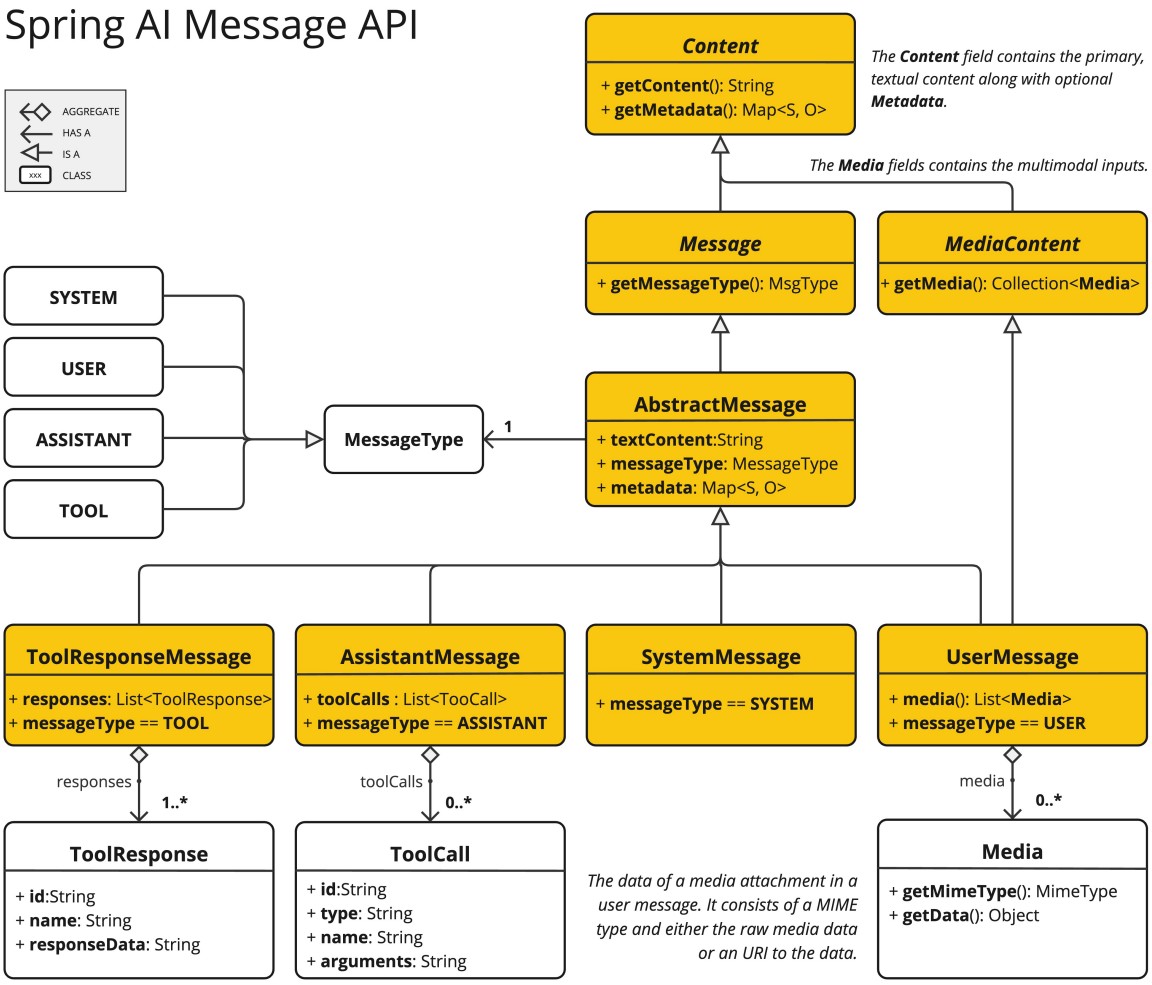
Day7 - 如何跟 ChatGPT 一樣處理多模態資料

還記得 Message 這張 UML 圖片嗎？在 UserMessage 中其實是可以包含多媒體檔案的，這也是最近 AI 強調的多模態（**Multimodality**）



目前有支援多模態的模型如下，其他模型送出多媒體檔案時可是會出現錯誤的，前面提到的 Groq 是不支援多媒體檔案的喔，不過還好便宜的 GPT-4o mini 能支援

- Open AI - (GPT-4-Vision and GPT-4o models)
- Ollama - (LlaVa and Baklava models)
- Vertex AI Gemini - (gemini-pro-vision model)
- Anthropic Claude 3

- AWS Bedrock Anthropic Claude 3

下表是不同模型支援的輸入輸出格式（這些資料都會隨著模型升級後有所變化，大家還是以官方最新資料為參考）

Input	Output	Examples
Language/Code/Images (Multi-Modal)	Language/Code	GPT4 - OpenAI, Google Gemini
Language/Code	Language/Code	GPT 3.5 - OpenAI-Azure OpenAI, Google Bard, Meta Llama
Language	Image	Dall-E - OpenAI + Azure, Deep AI
Language/Image	Image	Midjourney, Stable Diffusion, RunwayML
Language	Audio	OpenAI, Azure OpenAI
Audio	Language	OpenAI, Azure OpenAI
Text	Numbers	Many (AKA embeddings)

除了文字以外，圖片、影片以及聲音都屬於多媒體檔案

程式

接下來我們寫一隻程式模擬 ChatGPT 上傳檔案並詢問圖片在說明甚麼內容

```
@RestController
@RequiredArgsConstructor
public class AiController {
    private final ChatModel chatModel;

    @PostMapping(value = "/imagequery")
    public String imageQuery(@RequestParam MultipartFile file
        UserMessage userMessage = new UserMessage(
                message,
                new Media(
                    MimeTypeUtils.parseMimeType(file.getConte
                    file.getResource())
                );
        return chatModel.call(userMessage);
```

```
    }
}
```

重點說明

- 首先是檔案上傳的部分，之前為了方便測試，都使用 GetMapping，不過 Get 不支援上傳檔案，必須改為 PostMapping

```
@PostMapping(value = "/imagequery")
```

- 傳入的參數使用 Spring 封裝的 MultipartFile

```
public String imageQuery(@RequestParam MultipartFile file, @R
```

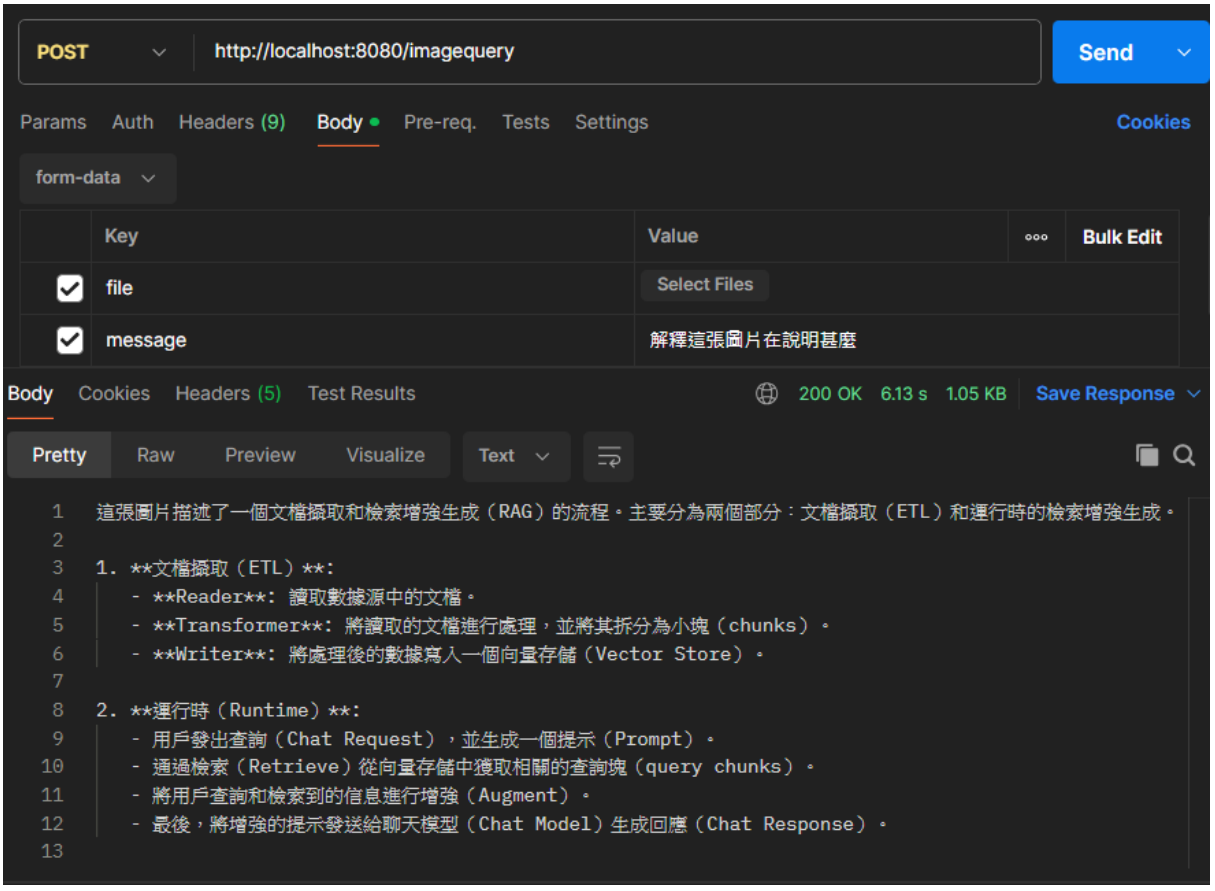
- Media 需要傳入 MimeType 以及 Resource，都可以從 MultipartFile 取得

```
new Media(
                    MimeTypeUtils.parseMimeType(file.getConte
                    file.getResource())
                );
```

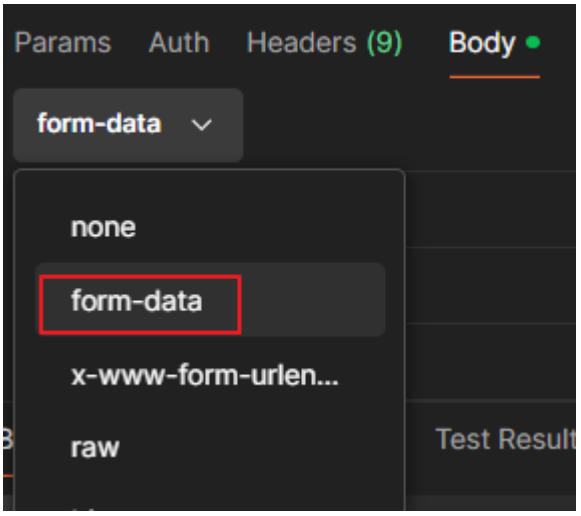
UserMessage 有好幾種建構方式，若有多個檔案可使用 List 傳入

```
public class UserMessage extends AbstractMessage {

    public UserMessage(String message) {
        super(MessageType.USER, message);
    }

    public UserMessage(Resource resource) {
        super(MessageType.USER, resource);
    }

    public UserMessage(String textContent, List<Media> mediaList) {
        super(MessageType.USER, textContent, mediaList);
    }

    public UserMessage(String textContent, Media... media) {
        this(textContent, Arrays.asList(media));
    }

    public UserMessage(String textContent, Collection<Media> mediaList, Map<String, Object> metadata) {
        super(MessageType.USER, textContent, mediaList, metadata);
    }
```

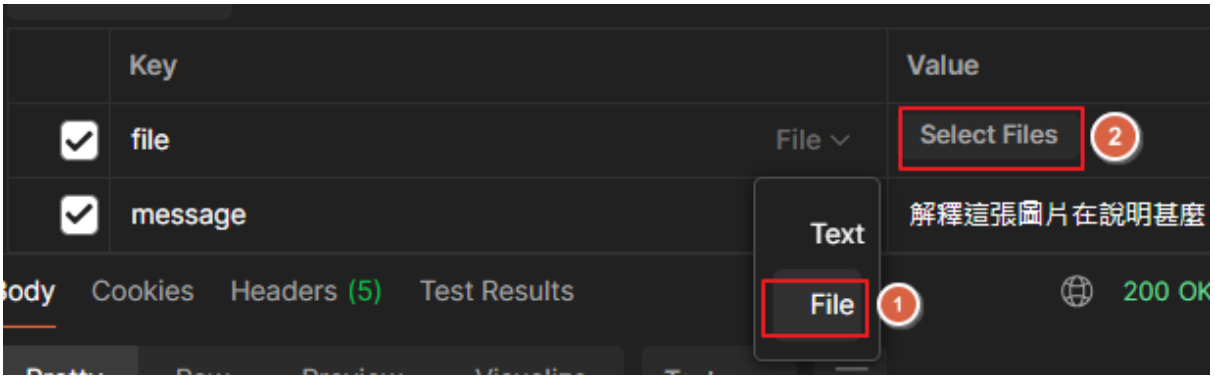
測試



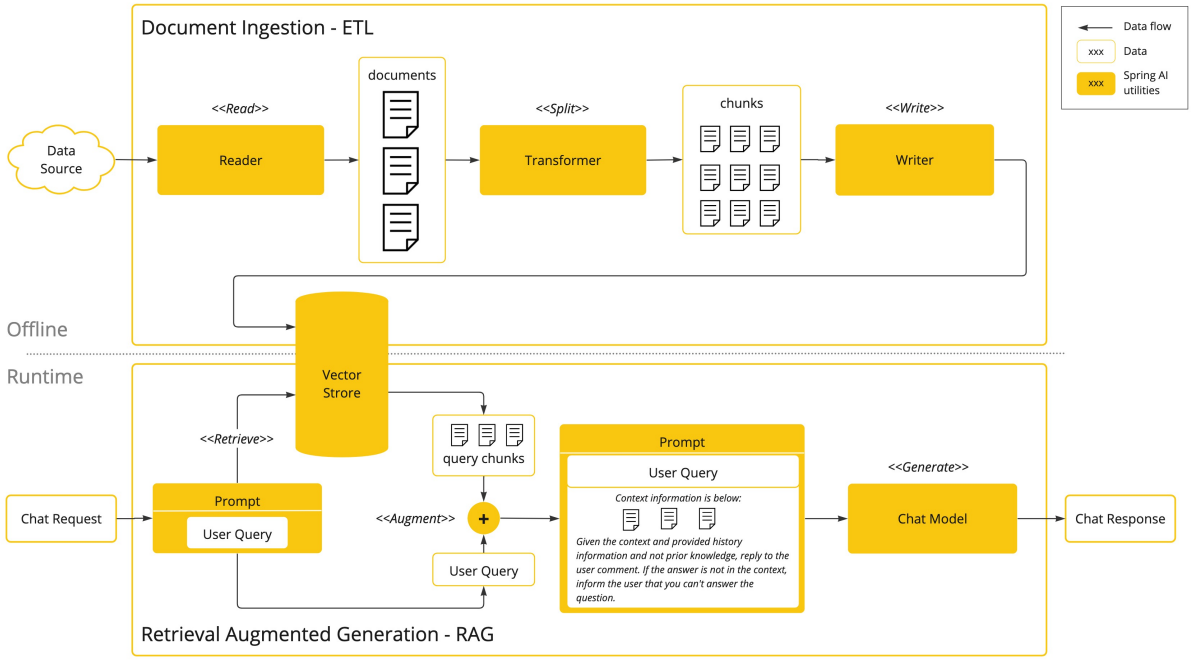
- PostMan 支援多種 Http Method，測試時記得先將方法改為 Post
- 在 Body 上要將類型改為 form-data



- key 的部分要傳入 file 及 message（程式傳入的參數名），file 需要將類別改為 File 才能傳送檔案



完成後就能按 Send 測試結果了，下面這張圖是在介紹 RAG 的流程



來看看 AI 的回答吧

> 這張圖片描述了一個文檔攝取和檢索增強生成（RAG）的流程。主要分為兩個部分：文檔攝取（ETL）和運行時的檢索增強生成。
>
> 1. **文檔攝取（ETL）**：
>
>   - **Reader**: 讀取數據源中的文檔。
>   - **Transformer**: 將讀取的文檔進行處理，並將其拆分為小塊（chunks）。

> - **Writer**: 將處理後的數據寫入一個向量存儲（Vector Store）。
>
> 2. **運行時（Runtime）**：
>
>   - 用戶發出查詢（Chat Request），並生成一個提示（Prompt）。
>   - 通過檢索（Retrieve）從向量存儲中獲取相關的查詢塊（query chunks）。
>   - 將用戶查詢和檢索到的信息進行增強（Augment）。
>   - 最後，將增強的提示發送給聊天模型（Chat Model）生成回應（Chat Response）。
>
> 整個過程展示了如何從數據源提取信息並使用該信息來增強用戶查詢的回應。

回顧一下今天學到甚麼:

1. 如何透過 Spring MVC 上傳檔案
2. 如何將檔案轉為 Media 並加入 UserMessage
3. 在 PostMan 測試檔案上傳並詢問 AI 內容