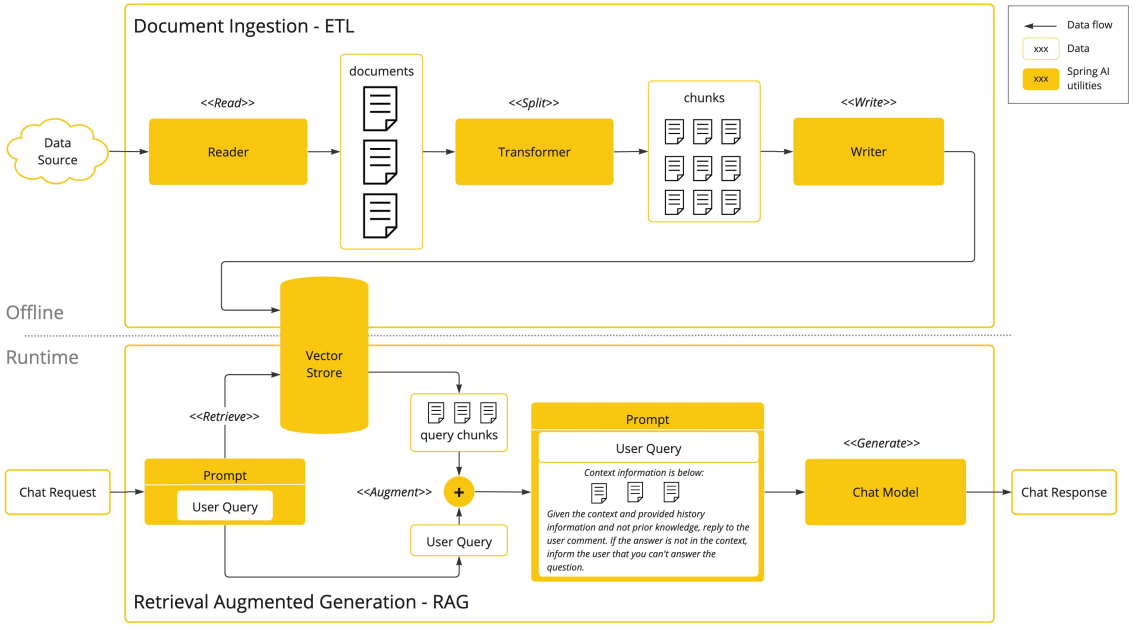
Day29 - RAG(增強查詢)

所有的準備都是為了最後的演出

前面已詳述過 RAG 最主要的目的，就是提供 LLM 未知的資訊，而這些資料會使用 ETL 技術或是由企業資料庫中取得，經過 embedding 計算後存於向量資料庫，而詢問內容同樣也經過 embedding 後在去向量資料庫中比對，所得近似資料連同問題再一起提供給 LLM，利用 In-Context 的特性將資訊整理後將最終結果呈現出來



所以 RAG 最後的動作其實包含了查詢與內容生成，生成的部分其實與前面說的提示詞應用都大同小異，就是將內容附加在提示詞中，再由 LLM 分析產生最後的內容，RAG 最關鍵的部分就是如何讓查詢更為精準

昨天講的 Transformer 使用了關鍵字跟摘要讓內容更為精準，而在查詢時也有許多手法能加強搜尋結果，這些方法還不斷在增加中，凱文大叔就自己所知列出於下，若有遺漏也請各位大大前輩分享

- 排除近似查詢分數較低的結果：正所謂垃圾進垃圾出，若提供的分塊內容偏離主題太多，LLM 的回答就會有所偏離，此時可設定一個分數基準，低於分數的內容就屏棄不用，避免干擾 LLM 生成資料
- 關鍵字篩選：昨天的關鍵字除了能強化 embedding 分數外，在近似查詢前後也能使用關鍵字過濾內容讓得到的結果更進一步的精練

- 重複驗證內容：避免 LLM 胡謅一通可以生成資料後再請 LLM 驗證，可以使用不同的 LLM 來進行複驗讓資料更有公信力
- 建立上下關聯：同一份資料拆解前都有一定的相關性，分塊後將原始文件的資訊紀錄在 metadata 中，進行 embedding 時可以加強各維度的分數
- 使用知識圖譜：這也是最近幾個月才由微軟提出的論文，將 ETL 後的資料以及詢問的內容都先經過資料梳理成多個 Entities，並將 Entity 之漸漸立 Relationship，再對這些 Entities 進行 embedding，查詢時透過 Graph 特性可以找到所有相關的 Entity，並將 Entity 與相關內容整理後交由 LLM 生成結果，如此可以將有關係的內容也一併找出，不像一般的 RAG 只能關注在分塊後的資料

程式實作

如同上面所說，使用 RAG 查詢前需要先將資料匯入向量資料庫(使用ETL或是從其他資料庫匯入)，資料就拿前幾天匯入向量資料庫的文章(這次鐵人賽的內容)作為範例

為了區隔 **ETL** 以及 **查詢**，將 **查尋** 的部分獨立出來，分別寫一隻 Service 類別以及 Controller 類別，其他的 Config 類別以及 application.yml 設定都沿用前幾天的程式

`SearchService.java`

```
@Data
@Service
public class SearchService {
    private final VectorStore vectorStore;  //Spring AI自動配置
    private final ChatClient chatClient;    //使用Config類別建立

    public String search(String query) throws IOException {
        Path pathResult = Paths.get("./result.md");
        String result = chatClient.prompt().advisors(
                                                new Quest
                                        ).user(query).call().
        Files.writeString(pathResult, result);
    Resource resource = new UrlResource(pathResult.toUri());
        return resource;
    }
}
```

> 為了呈現程式碼的效果，特別轉成md檔案匯出

API 也分離出來

`SearchController.java`

```
@RestController
@RequestMapping("/ai/search")
@RequiredArgsConstructor
public class SearchController {
    private final SearchService searchService;

    @GetMapping("/rag")
    public ResponseEntity<Resource> search(String query) {
        Resource resource;
        try {
            resource = searchService.search(query);
        } catch (IOException e) {
            e.printStackTrace();
            return ResponseEntity.internalServerError().build
        }
        return ResponseEntity.ok()
                .header(HttpHeaders.CONTENT_DISPOSITION, "att
                .body(resource);
    }
}
```