零門檻 App 創作：對話式 Android APK 自動建構工具

專題編號：114-2-CSIE-S008
執行期限：114 年第 1 學期至 114 年第 2 學期
指導教授：陳彥霖
專題參與人員： 112340104 黃紹泓
112590005 陳宇威

一、摘要

本專題開發一款對話式 Android APK 自動建置系統，旨在解決大型語言模型（LLM）生成程式碼時的幻覺與編譯缺陷。系統設計多層次修復引擎：透過編排服務將需求轉化為 JSON 藍圖以解耦架構；結合 ktlint 進行語法清洗；動態擷取 Gradle 錯誤日誌，驅動本地 Gemma 模型執行自動修復循環（Auto-Repair Loop）；並導入 TDD 流程進行 JUnit 測試驗證。後端基於 FastAPI 與 Supabase 實現非同步建置與雲端託管。實驗證實，本系統具備優異的代碼自癒能力，能有效降低行動端開發門檻。

關鍵詞： 藍圖化架構、自動修復循環、測試驅動開發 (TDD)、FastAPI 非同步引擎、大型語言模型 (LLM)

二、緣由與目的

儘管現今 LLM 功能強大，但非資訊背景使用者仍難以克服行動端原生開發與編譯鏈的高門檻，且 AI 生成代碼時常因「幻覺」導致編譯失敗。為此，本專題打造「零門檻 APK 自動建置系統」，使用者僅需輸入自然語言需求，系統即全自動執行藍圖解耦、靜態清洗、修復循環與測試驗證，最終編譯打包並維護雲端生命週期。本專題旨在結合 AI 生成靈活性與軟體工程嚴謹驗證，實現端到端的零門檻行動開發。

三、研究報告內容

本研究範圍涵蓋自然語言處理、行動端架構編排、靜態代碼分析、自動化編譯與測試以及伺服器資源管理，主要分為以下三大核心範疇：

1. **自然語言藍圖化與解耦：** 設計結構化 JSON 藍圖規範，將模糊的語意描述收斂為確定的元件、邏輯與資料流配置。
2. **代碼語法清洗與自動修復：** 實作 Regex 語法過濾器，解決 LLM 常見的 Markdown 標記殘留、非預期字元等問題，並落實行動端靜態代碼規範。
3. **錯誤監控與動態自癒循環：** 攔截 Gradle 系統底層編譯日誌，將編譯錯誤（Compile-time Errors）精準回饋給大模型進行二次修復。

四、使用技術與工具說明

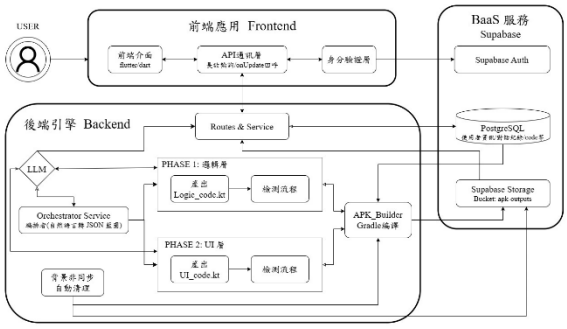
本系統採前後端分離架構，技術棧如下：

- **前端**：Flutter/Dar，以 StatefulWidget 結合長效輪詢與 onUpdate 回呼即時更新狀態，整合 supabase_flutter 處理驗證。
- **後端**：Python FastAPI，利用 asyncio 實作非同步佇列並以 UUID 追蹤，經 SSE/WebSocket 推送建置狀態。
- **LLM**：本地端 Gemma 4 26B A4B，專責語意理解、藍圖生成、程式碼生成和修復。
- **BaaS**：使用 Supabase，含 PostgreSQL（存對話與代碼快照）、 Storage（託管 APK 並提供時效性安全連結）與 Supabase

auth 認證使用者 gmail 帳號。

- **工具鏈**：整合 ktlint 語法修正、JUnit 單元測試，並配置共用 Gradle Cache 加速 Docker 編譯效能。

五、架構流程



圖一、系統的端到端運作流程

1. 編排與藍圖化（Orchestrator Stage）： 後端 Orchestrator 接收前端 Prompt，引導本地 Gemma 模型輸出 JSON 結構藍圖。若語意模糊則主動發起對話澄清。
2. 語法清洗與靜態修復（Static Cleaning）： QA & Parser 模組透過正規表達式過濾不必要代碼標記，並調用 ktlint 修正縮排與基本語法錯誤。
3. 測試驅動邏輯驗證（TDD & JUnit）： 系統同步生成邏輯代碼與單元測試代碼。在 Gradle 環境執行 JUnit 測試，確保核心邏輯 100% 正確後，才進入 UI 渲染階段。
4. 自動修復自癒循環（Auto-Repair Loop）： 若 Gradle 在 TDD 或最終打包階段編譯失敗，系統會透過 Regex 動態擷取 Error Log，並將錯誤代碼連同原程式碼作為 Context 重新餵給大模型，進行封閉式的自癒循環，直至編譯成功或觸及上限。
5. 雲端發布與通知： 打包完成的 APK 自動上傳至 Supabase Storage，後端變更狀態為 [DONE] 並釋出安全下載網址，觸發前端動態渲染下載按鈕。

六、實驗結果

本專題針對典型應用需求進行端到端功能性驗證，確認整體系統之可行性：

1. 代碼自癒功能驗證：測試證實系統能透過正規表達式精準擷取 Gradle 編譯錯誤訊息，並成功驅動本地 Gemma 模型進行二次修正，落實封閉式自癒修復循環並順利打包 APK。
2. 快取機制成效：於 Docker 伺服器環境下，配置共用 Gradle Cache 能成功複用依賴檔案，有效避免重複下載，優化後端引擎之建置速度與 I/O 損耗。
3. 資源釋放測試：經長期監控，全自動 Garbage Collector 能準確定位並移除閒置工作目錄（WORKSPACES_DIR）與暫存原始碼，成功將伺服器硬碟佔用率維持在安全水位。

七、結論

本專題成功開發「零門檻 APK 自動建置系統」，透過藍圖化解耦、自動修復循環與 TDD 測試機制，有效克服 LLM 生成代碼之幻覺缺陷，證實自然語言驅動原生行動端開發之可行性。未來將導入畫面即時預覽、圖片多模態審查，並將後端優化為輕量化多模型分工架構，以進一步提升編譯算力與產出效率。

八、參考文獻

[1] Google Gemma open models
https://deepmind.google/models/gemma/
[2] Pinterest ktlint GitHub
https://github.com/ktlint/ktlint
[3] JUnit
https://junit.org/
[4] FastAPI documentation
https://fastapi.tiangolo.com/
[5] Supabase
https://supabase.com/