

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO
ĐỒ ÁN CUỐI KỲ

Môn học

Phát triển ứng dụng cho thiết bị di động nâng cao

Giáo viên hướng dẫn

Thầy Phạm Hoàng Hải

Thầy Trương Phước Lộc

Thầy Trần Duy Quang

Lớp

CQ2021/3

Sinh viên thực hiện

21120126 – Nguyễn Tấn Hoàng Sa

21120144 – Nguyễn Phúc Thuần

21120186 – Lê Hữu Trí

5 tháng 1 năm 2025

LỜI CẢM ƠN

Em xin gửi lời cảm ơn chân thành và sâu sắc nhất đến thầy Phạm Hoàng Hải, thầy Trương Phước Lộc, thầy Trần Duy Quang.

Nhờ có sự tận tình chỉ bảo, hướng dẫn và động viên của các thầy, em đã có thể hoàn thành tốt công việc/học tập của mình. Sự tận tụy và kiến thức chuyên môn của các thầy đã giúp em không chỉ hiểu rõ hơn về chuyên môn mà còn tiếp thu được nhiều bài học quý giá trong cuộc sống.

Em xin chân thành cảm ơn các thầy vì sự đồng hành và chỉ bảo trong suốt thời gian qua. Kính chúc các thầy luôn mạnh khỏe, hạnh phúc và tiếp tục gặt hái nhiều thành công trong sự nghiệp trồng người.

MỤC LỤC

<i>LỜI CẢM ƠN</i>	2
I. THÔNG TIN DỰ ÁN	5
1. Thông tin nhà phát triển:	5
2. Phân công công việc:	5
3. Tên ứng dụng:	7
4. Mô tả ứng dụng:	7
5. Dependencies:.....	8
II. CHỨC NĂNG	9
1. Xác thực và phân quyền	10
1.1. Giao diện:.....	10
1.2. Kĩ thuật:.....	11
2. AI Chat.....	11
2.1. Giao diện:.....	11
2.2. Kĩ thuật:.....	13
3. Tạo và quản lý AI BOT.....	18
3.1. Giao diện:.....	18
3.2. Kĩ thuật:.....	19
4. Tạo bộ dữ liệu tri thức Knowledge và Units trong Knowledge	19
4.1. Giao diện:.....	19
4.2. Kĩ thuật:.....	24
- Sử dụng Provider để quản lí trạng thái statement của các trạng thái state:.....	24
o Các notifier của Knowledge bao gồm KnowlegeNotifier để quản lí các trạng thái của Knowledge và AddKnowlegeDialog để quản lí các trạng thái của Dialog Add Knowlege.....	25
5. Quản lý và sử dụng Prompt	28
5.1. Giao diện:.....	28
5.2. Kĩ thuật:.....	29
6. Nâng cấp tài khoản lên Pro & Monetization.....	32
6.1. Giao diện:.....	32
6.2. Kĩ thuật:.....	33
7. Soạn Email với AI.....	35
7.1. Giao diện:.....	35
7.2. Kĩ thuật:.....	37
8. Preview chat với personal assistant.....	39
8.1. Giao diện:.....	39
8.2. Kĩ thuật:.....	41

I. THÔNG TIN DỰ ÁN

1. Thông tin nhà phát triển:

STT	MSSV	Họ tên	Mức độ hoàn thành
1	21120126	Nguyễn Tấn Hoàng Sa	100%
2	21120144	Phạm Phúc Thuần	100%
3	21120186	Lê Hữu Trí	100%

2. Phân công công việc:

Tính năng	Phân công	Tiến độ
1. Xác thực và phân quyền		
1 Trang chủ	Thuần	Đã xong
2 Đăng ký tài khoản	Thuần	Đã xong
3 Kiểm tra các ràng buộc về tên đăng nhập, mật khẩu nhập lại, ...	Thuần	Đã xong
4 Kích hoạt tài khoản bằng email	Thuần	Đã xong
5 Đăng nhập hệ thống với tài khoản đã tạo	Thuần	Đã xong
6 Đăng nhập Google	-	-
7 Quên mật khẩu và làm mới mật khẩu bằng email	Thuần	Chưa làm
8 Đăng xuất tài khoản	Thuần	Đã xong
2. Xác thực và phân quyền		
9 Hiển thị nội dung chat	Sa	Đã xong
10 Chat với AI Chat bot	Sa	Đã xong
11 Giảm số lượng token khi chat	Sa	Đã xong
12 Hỗ trợ thay đổi AI Agent	Sa	Đã xong
13 Tạo thread chat mới	Sa	Đã xong
14 Xem danh sách lịch sử thread chat	Sa	Đã xong
15 Mở lịch sử chat	Trí	Đã xong
6. Quản lý và sử dụng prompt		
16 Hiển thị và tìm kiếm public prompt	Trí	Đã xong
17 Lọc prompt theo category	Trí	Đã xong
18 Thêm prompt vào favourite và xem danh sách favourite	Trí	Đã xong
19 Tạo mới 1 private prompt	Trí	Đã xong

20	Hiển thị và tìm kiếm private prompt	Trí	Đã xong
21	Cập nhật và xóa private prompt	Trí	Đã xong
22	Sử dụng Prompt trong library	Trí	Đã xong
23	Sử dụng nhanh prompt trong Chat với slash (/)	Thuần	Đã xong
7. Tạo và quản lý AI BOT			
24	Tạo AI BOT	Thuần	Đã xong
25	Hiển thị/tìm kiếm AI BOT	Thuần	Đã xong
26	Cập nhật và xoá AI BOT	Thuần	Đã xong
27	Cập nhật prompt cho AI BOT	Thuần	Đã xong
28	Giao tiếp với AI BOT đã tạo qua Chat widget	Thuần	Đã xong
29	Thêm/xóa dữ liệu tri thức vào AI BOT	Thuần	Đã xong
30	Preview và chat với AI BOT	Thuần	Đã xong
31	Publish AI Chat ra Slack, Telegram, Messenger	Thuần	Đã xong
8. Tạo bộ dữ liệu tri thức			
32	Thêm bộ dữ liệu tri thức	Sa	Đã xong
33	Hiển thị/tìm kiếm bộ dữ liệu tri thức	Sa	Đã xong
34	Disable/delete nguồn dữ liệu	Sa	Đã xong
35	Nạp dữ liệu từ file	Sa	Đã xong
36	Nạp dữ liệu từ URL website	Sa	Đã xong
37	Nạp dữ liệu từ Google Drive	Sa	Chưa làm
38	Nạp dữ liệu từ Slack	Sa	Đã xong
39	Nạp dữ liệu từ Confluence	Sa	Đã xong
7. Nâng cấp tài khoản lên Pro & Monetization			
40	Nâng cấp tài khoản	Trí	Đã xong
41	Hiển thị thông tin tài khoản Pro và cập nhật số token thành unlimited	Trí	Đã xong
42	Gắn code quảng cáo và kiếm tiền qua quảng cáo	Trí	Đã xong
8. Hỏi đáp trên ảnh			
43	Upload ảnh để chat	Trí	Đã xong
44	Chụp ảnh và chat với ảnh đã chụp	Trí	Đã xong
45	Screenshot và chat với screenshot	-	-
9. Soạn email với AI			
46	Tạo tab riêng cho soạn theo email	Trí	Đã xong
47	Thêm các AI actions để tạo draft email (Thanks, Sorry, Yes, No, Follow Up, Request for more information)	Trí	Đã xong

3. Tên ứng dụng:

- Tên ứng dụng: STEP AI - System for Task Efficiency and Productivity.
- Github: <https://github.com/LeHuuTri-186/STEP-AI>

4. Mô tả ứng dụng:

- Dự án STEP AI được phát triển nhằm cung cấp một nền tảng tích hợp trí tuệ nhân tạo (AI) để hỗ trợ tối ưu hóa quy trình làm việc và nâng cao hiệu quả cho người dùng. Với các tính năng chính như tạo và quản lý BOT AI, trò chuyện tự nhiên với AI, xây dựng cơ sở tri thức tùy chỉnh, quản lý và sử dụng lời nhắc, cũng như soạn thảo email thông minh, STEP AI hứa hẹn mang đến trải nghiệm tối ưu trong việc tự động hóa và cải thiện giao tiếp.
- Dự án còn bao gồm việc triển khai các gói nâng cấp Pro, cho phép người dùng tiếp cận thêm các tính năng cao cấp và các giải pháp AI. STEP AI không chỉ là một công cụ hỗ trợ công việc mà còn là bước đệm để thúc đẩy sáng tạo và đổi mới trong môi trường làm việc hiện đại.

5. Dependencies:

```
dependencies:
  google_fonts: ^6.2.1
  font_awesome_flutter: ^10.7.0
  flutter:
    sdk: flutter

  freezed_annotation: ^2.0.3
  json_annotation: ^4.9.0
  sentry_flutter: ^8.12.0

  http: ^1.2.2
  provider: ^6.1.2
  flutter_secure_storage: ^9.2.2
  shared_preferences: ^2.3.3
  get_it: ^8.0.2
  flutter_highlight: ^0.7.0

  cupertino_icons: ^1.0.8
  flutter_markdown: ^0.7.4+1
  dio: ^5.7.0
  loading_animation_widget: ^1.3.0
  file_picker: ^8.1.6
  http_parser: ^4.0.2
```

```
cupertino_icons: ^1.0.8
flutter_markdown: ^0.7.4+1
dio: ^5.7.0
loading_animation_widget: ^1.3.0
file_picker: ^8.1.6
http_parser: ^4.0.2
url_launcher: ^6.3.1

image_picker: ^0.8.7+4
connectivity_plus: ^6.1.1
google_mobile_ads: ^5.2.0

dev_dependencies:
  flutter_native_splash: ^2.2.10
  flutter_test:
    sdk: flutter
  freezed: ^2.4.2
  build_runner: ^2.3.3
  json_serializable: ^6.6.2

  flutter_lints: ^4.0.0

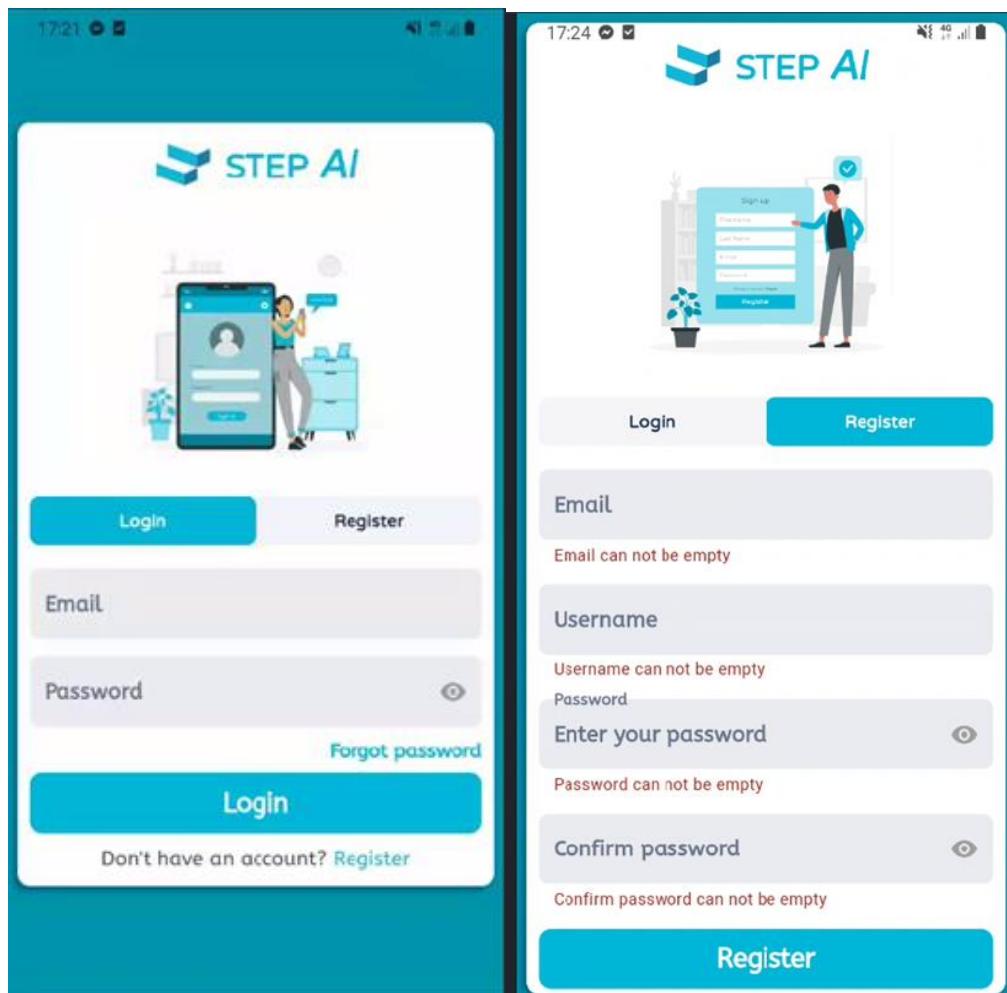
flutter:
  ...
  flutter:
    uses-material-design: true
    assets:
      - lib/core/assets/imgs/
      - lib/core/assets/source_unit_images/
  flutter_native_splash:
    color: "#ffffff" # Background color of the splash screen
    image: lib/core/assets/imgs/step-ai-logo.png # Path to your splash image
    android: true
    ios: true
    web: false
```

II. CHỨC NĂNG

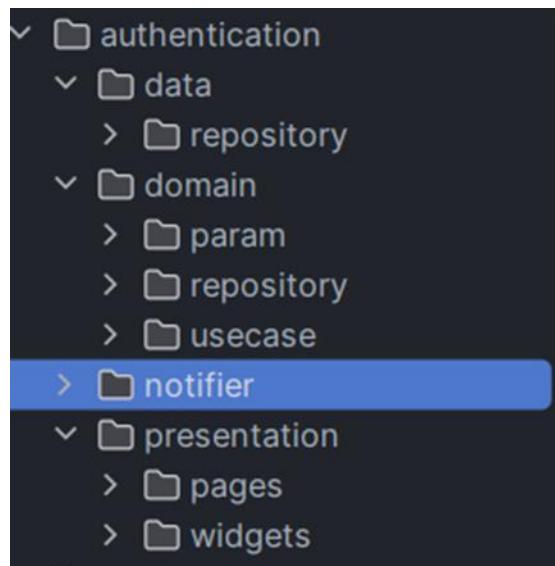
1. Xác thực và phân quyền

1.1. Giao diện:

- Gồm giao diện login và register, việc xác thực nội dung hợp lệ được thực hiện trực tiếp ở front-end và được hiển thị qua errorMessage của TextFormField.



1.2. Kĩ thuật:

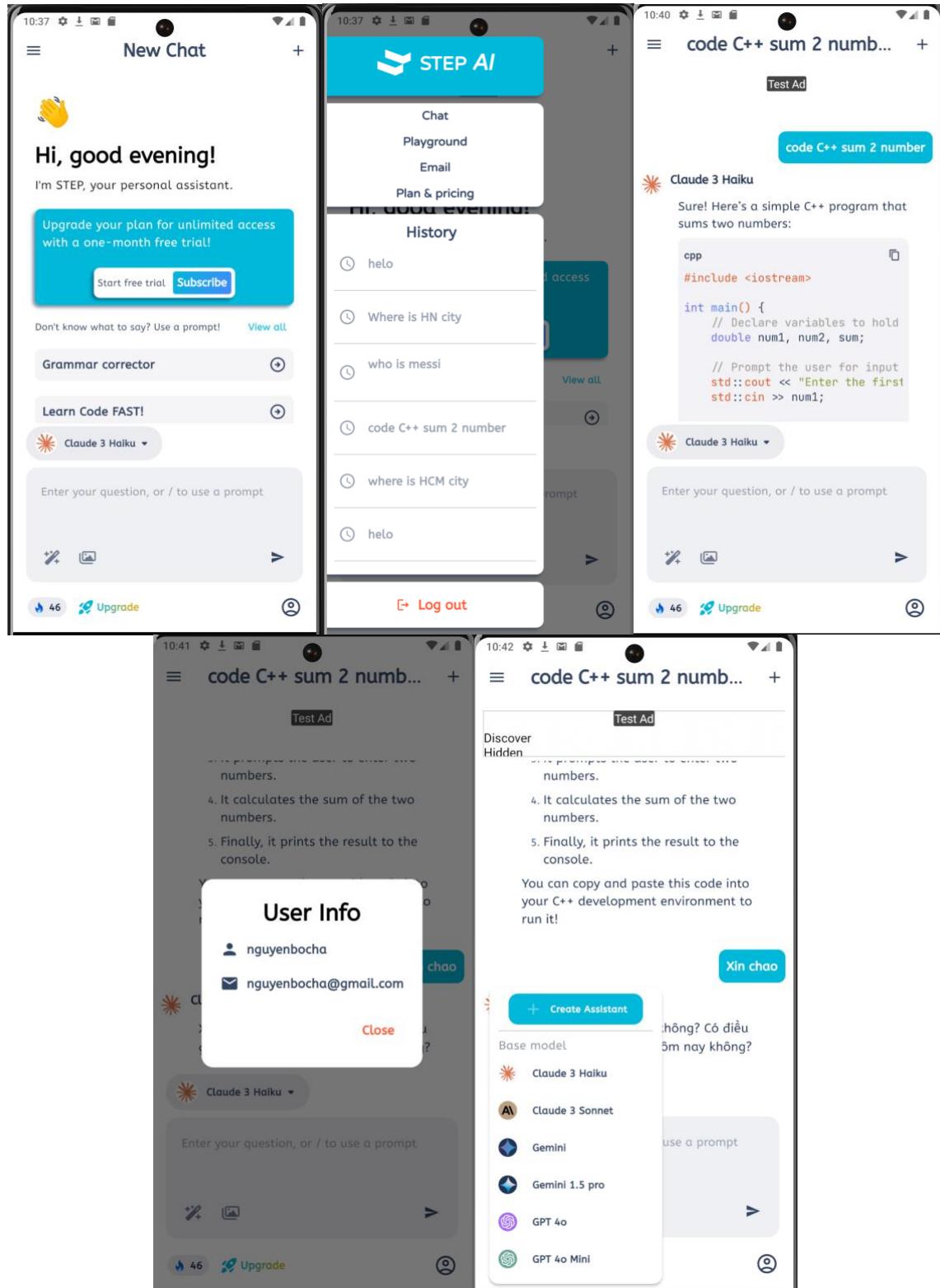


- Tuân theo kiến trúc clean architecture với tầng data, domain và presentation (notifier chứa các ChangeNotifier thuộc presentation)

2. AI Chat

2.1. Giao diện:

- Giao diện bao gồm các màn hình chat trước khi chat, màn hình lịch sử chat và các chức năng khác, màn hình sau khi chat, màn hình thông tin và màn hình chứa các AI Chatbot



2.2. Kĩ thuật:

- Sử dụng thư viện Dio để có thể gọi API có truy cập thông qua các Interceptor ở Request, Response và gọi lại các API khi access token hết hạn.

```
class ApiClientChat {  
    final Dio _dio = Dio();  
    final secureStorageHelper = getIt<SecureStorageHelper>();  
    final ApiService _apiService = ApiService(Constant.apiUrl);  
    // RefreshTokenUseCase refreshTokenUseCase = getIt<RefreshTokenUseCase>();  
    String? accessToken;  
    String? refreshToken;  
  
    ApiClientChat() {  
        _dio.options.baseUrl = Constant.apiUrl; // Base URL API  
  
        _dio.interceptors.add(InterceptorsWrapper(  
            onRequest: (options, handler) async {  
                // print(  
                //     "\n-----OnRequest-----"  
                await _initializeTokens();  
            },  
            onResponse: (response, handler) {  
                // print("-----OnResponse-----");  
                return response; // Return the response as is  
            },  
            onError: (error, handler) {  
                // print("-----OnError-----");  
                return error; // Return the error as is  
            }  
        );  
    }  
}  
extension Dio on Dio {  
    Future<Response> interceptorsPreHandle(DioInterceptorHandler handler) {  
        return super.interceptorsPreHandle(handler);  
    }  
}
```

- Sử dụng Provider để quản lí trạng thái statement của các trạng thái state:
 - o ChatNotifier để quản lí các trạng thái tin nhắn ở màn hình chat. quản lí các bot hỗ trợ AssitantNotifier:

```
class ChatNotifier with ChangeNotifier {  
    bool isLoading = false;  
    //number of rest token  
    int _numberRestToken = 0;  
    int get numberRestToken => _numberRestToken;  
    set numberRestToken(int numberRestToken) {  
        _numberRestToken = numberRestToken;  
        notifyListeners();  
    }  
  
    //isLoading detailed conversation  
    final GetMessagesByConversationIdUsecase _getMessagesByConversationIdUsecase;  
    bool _isLoadingDetailedConversation = false;  
    bool get isLoadingDetailedConversation => _isLoadingDetailedConversation;
```

- Quản lý trạng thái lịch sử histories ở drawer sử dụng HistoryConversationListNotifier.

```
import 'package:flutter/material.dart';

import '../domain/entity/assistant.dart';

class AssistantNotifier with ChangeNotifier{
>   final List<Assistant> _assistants = [];
    late String _currentAssistantId;
    AssistantNotifier() {
        _currentAssistantId = _assistants[0].id!;
    }

    String get currentAssistantId=> _currentAssistantId;

    void setCurrentAssistantId(String id) {
        _currentAssistantId = id;
        notifyListeners();
    }
}
```

- AssitantNotifier để quản lí trạng thái thay đổi của AI Bot cố định, mặc định của dự án:

```
import 'package:flutter/material.dart';

import '../domain/entity/assistant.dart';

class AssistantNotifier with ChangeNotifier{
    final List<Assistant> _assistants = [...];
    late String _currentAssistantId;
    AssistantNotifier() {
        _currentAssistantId = _assistants[0].id!;
    }

    String get currentAssistantId=> _currentAssistantId;

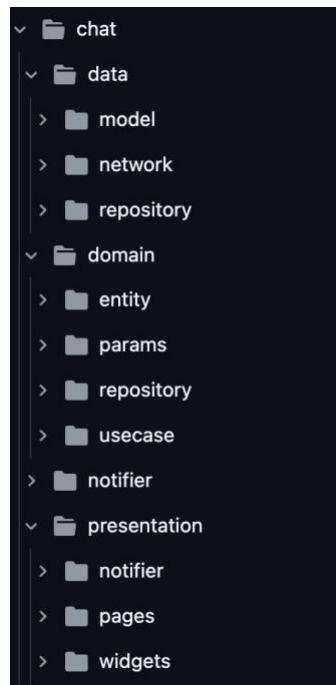
    void setCurrentAssistantId(String id) {
        _currentAssistantId = id;
        notifyListeners();
    }
}
```

- Sử dụng Dependency Injection bằng thư viện getIt để tiêm các chức năng phụ thuộc vào lẫn nhau. Kết hợp với đó là sử dụng Singleton Pattern để có thể đăng kí và tạo ra các đối tượng state duy nhất cho từng chức năng cần thiết.

```
//HistoryConversationListNotifier:-----
getIt.registerSingleton<HistoryConversationListNotifier>(
    HistoryConversationListNotifier(
        getIt<GetHistoryConversationListUsecase>(),
    );
//ChatNotifier:-----
getIt.registerSingleton<ChatNotifier>(
    ChatNotifier(
        getIt<SendMessageUsecase>(),
        getIt<AssistantNotifier>(),
        getIt<HistoryConversationListNotifier>(),
        getIt<GetUsageTokenUsecase>(),
        getIt<GetMessagesByConversationIdUsecase>(),
        getIt<PersonalAssistantNotifier>(),
        getIt<CreateThreadUseCase>(),
        getIt<AskBotUseCase>(),
        getIt<LogoutUseCase>(),
        getIt<GetCurrentUserUsecase>(),
    ),
);
```

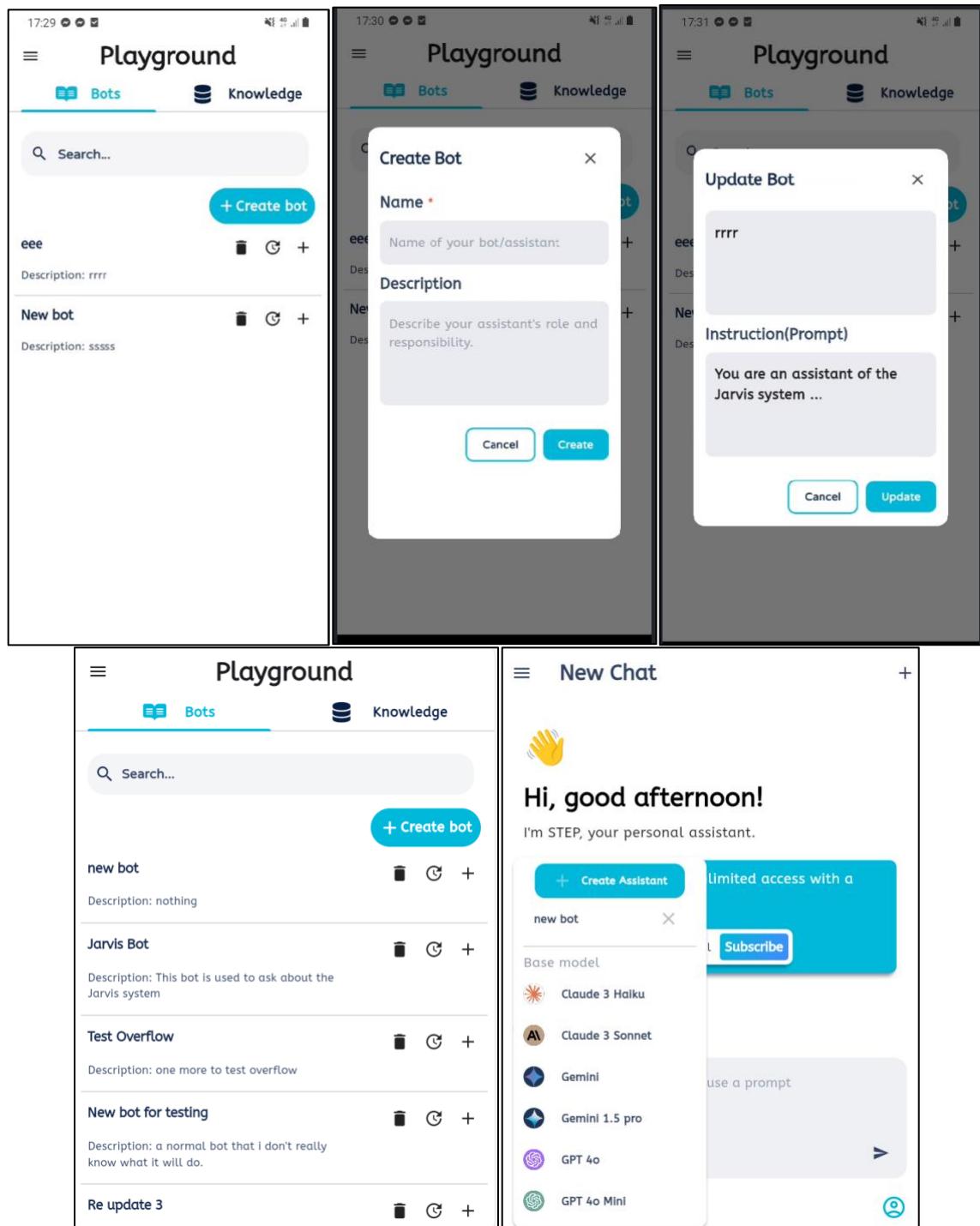
```
//Chat:-----
getIt.registerSingleton<SendMessageUsecase>
    SendMessageUsecase(
        getIt<ConversationRepository>(),
    ),
);
getIt.registerSingleton<GetMessagesByConversationIdUsecase>(
    GetMessagesByConversationIdUsecase(
        getIt<ConversationRepository>(),
    ),
);
getIt.registerSingleton<GetHistoryConversationListUsecase>(
    GetHistoryConversationListUsecase(
        getIt<ConversationRepository>(),
    ),
);
getIt.registerSingleton<GetUsageTokenUsecase>(
    GetUsageTokenUsecase(
        getIt<ConversationRepository>(),
    ),
);
getIt.registerSingleton<GetCurrentUserUsecase>
```

- Áp dụng tính năng dynamic load lịch sử cuộc trò chuyện khi người dùng kéo xuống dưới các lịch sử cũ hơn ở drawer.
- Có bắt các sự kiện nếu không có internet thì tin nhắn khi gửi sẽ thông báo là sẽ không có internet.
- Cấu trúc tổng quan thư mục của tính năng chat tuân theo clean architecture, các lời gọi API chỉ xử lý ở tầng data và các thành phần giao diện chỉ sử dụng ở tầng presentation, cũng như ở tầng domain chứa các cấu trúc mẫu và danh sách các usecase chính của chat.



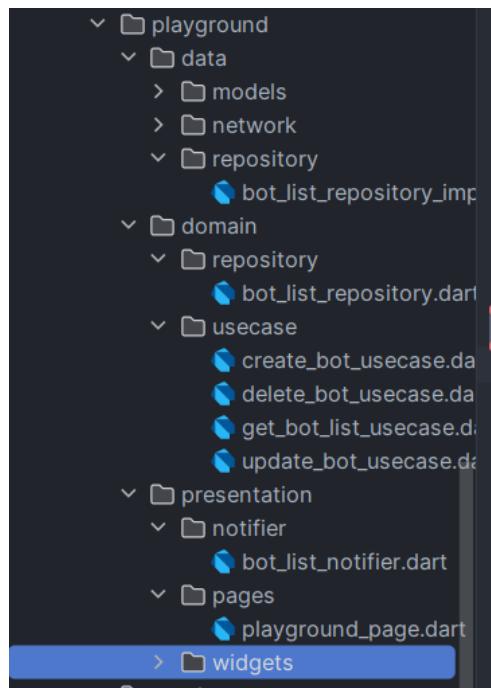
3. Tạo và quản lý AI BOT

3.1. Giao diện:



- Giao diện gồm giao diện cơ bản cho Bots (Assistant), thêm bot vào chat widget, thêm, update, xóa bot, lấy danh sách full và tìm kiếm.

3.2. Kĩ thuật:

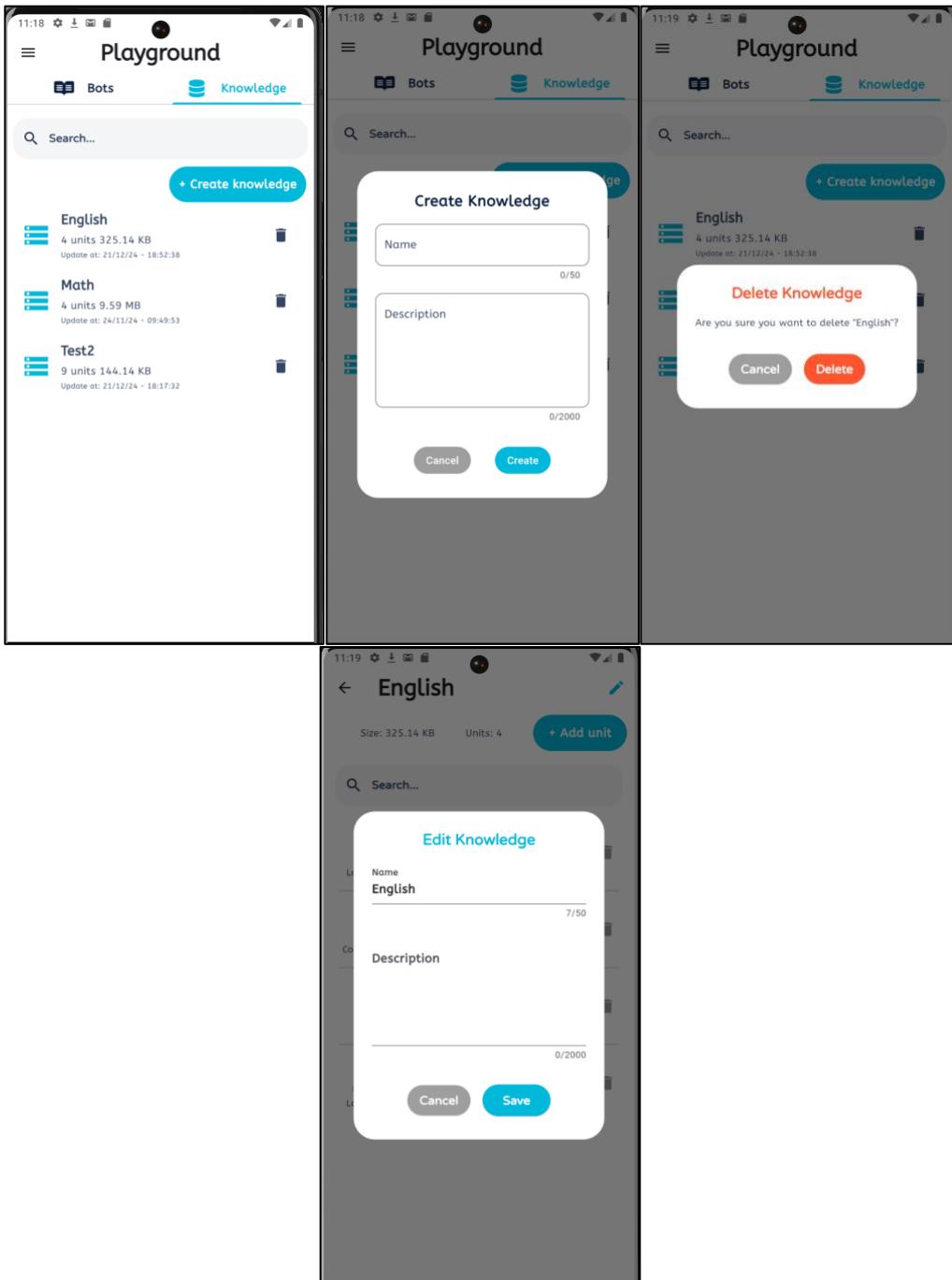


- Tuân theo kiến trúc clean architecture với việc giao tiếp với backend qua API chỉ diễn ra ở tầng Data.
- Tương tự với các tính năng khác khi cũng áp dụng getIt để inject các dependency đã được khai báo là Singleton để tránh lãng phí tài nguyên.

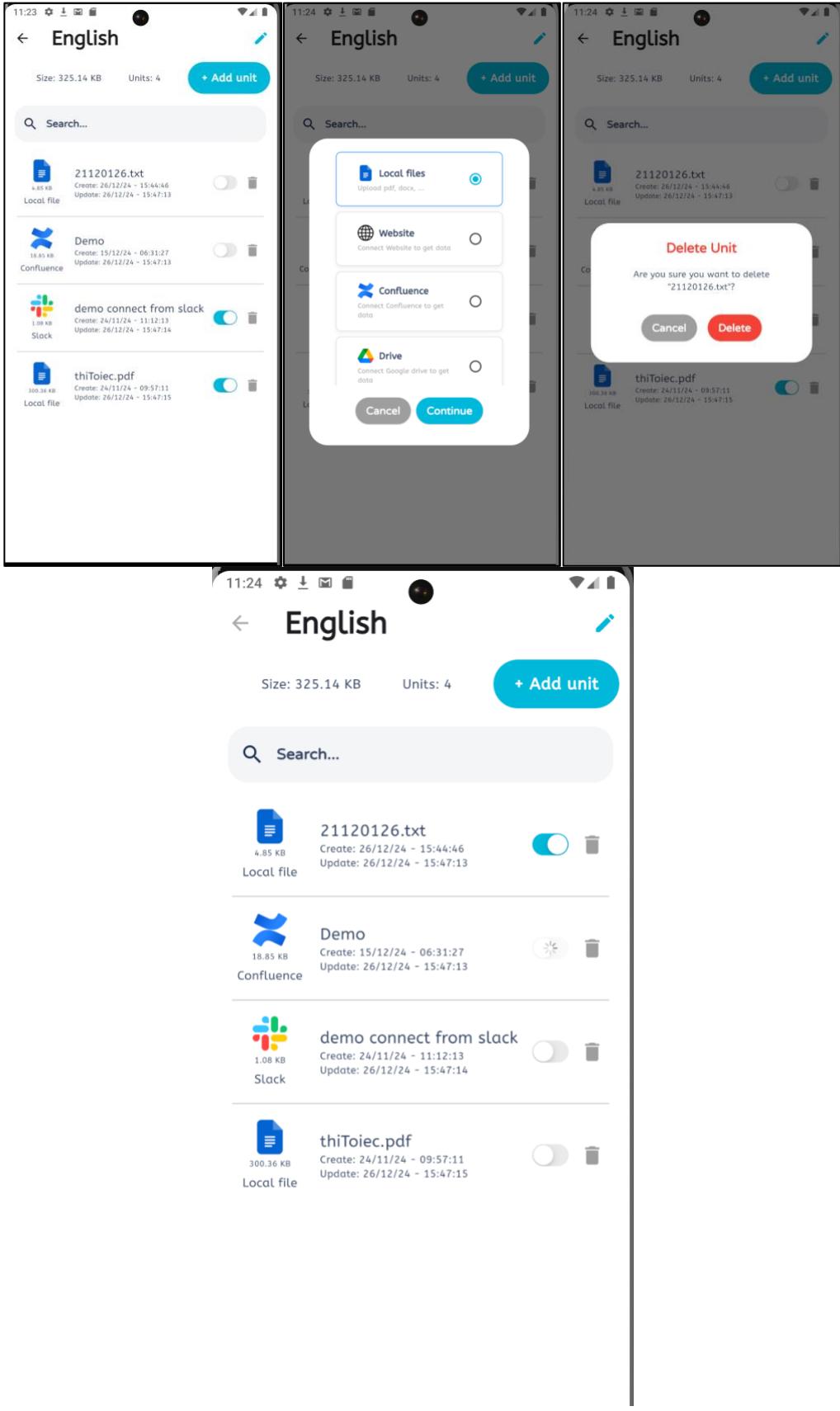
4. Tạo bộ dữ liệu tri thức Knowledge và Units trong Knowledge

4.1. Giao diện:

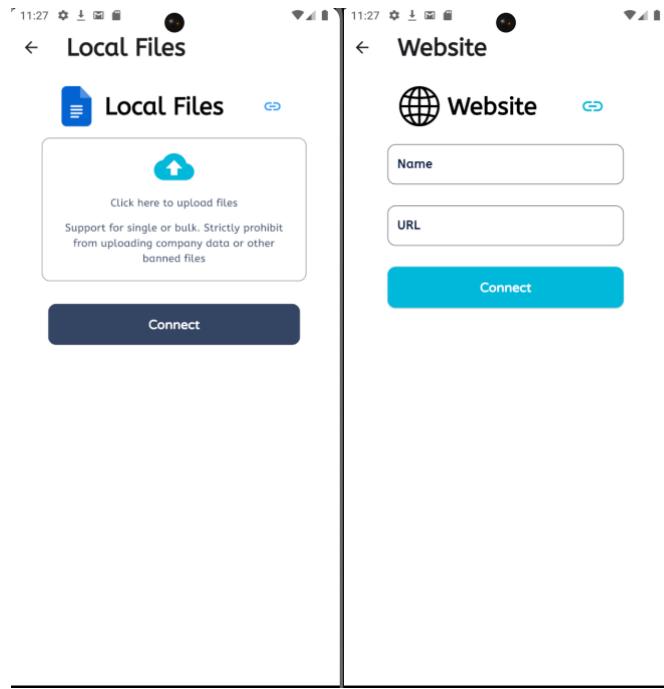
- Giao diện Knowledge gồm các màn hình liên quan tới danh sách Knowledge, tạo mới Knowledge, Xóa Knowledge và chỉnh sửa các thông tin liên quan tới Knowledge

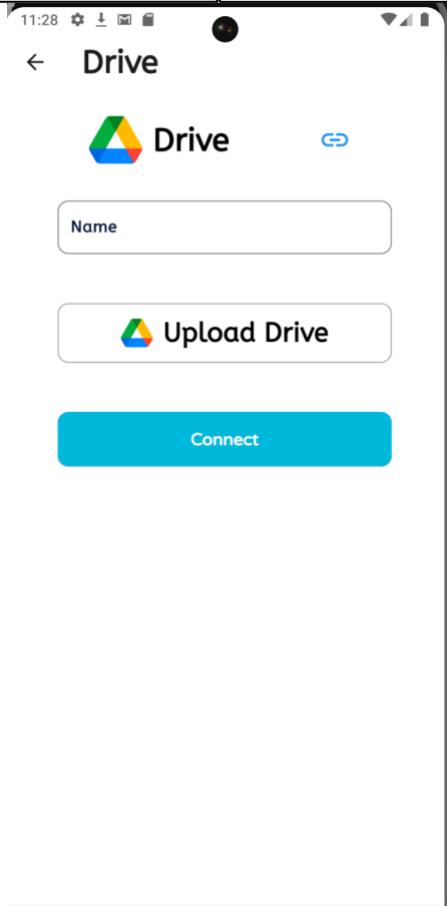
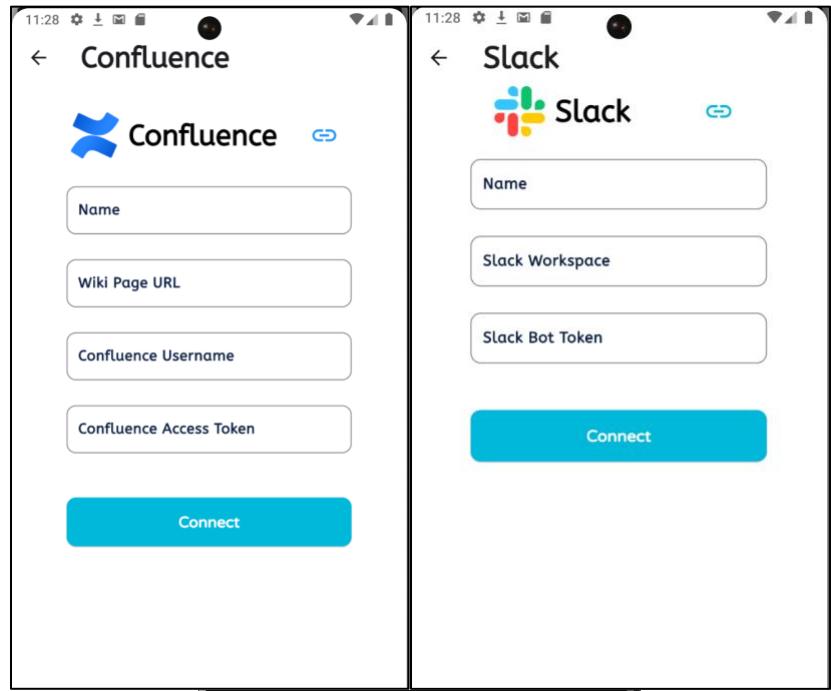


- Giao phần Unit trong Knowledge bao gồm các màn hình danh sách unit, Tạo mới Unit, Xóa Unit, màn hình chuyển status của các unit.



- Màn hình chi tiết để tạo các Unit ở bên trong Knowledge: bao gồm các màn lấy dữ liệu từ local file, nạp dữ liệu từ web, nạp dữ liệu từ Confluence, nạp dữ liệu từ Slack, và nộp dữ liệu từ Drive (chức năng này hiện tại đang pending vì chưa có API)





4.2. Kĩ thuật:

- Sử dụng thư viện Dio để có thể gọi API có truy cập thông qua các Interceptor ở Request, Response và gọi lại các API khi access token hết hạn.

```
class KnowledgeApi {  
    final Dio _dio = Dio();  
    // final secureStorageHelper = getIt<SecureStorageHelper>();  
    // final ApiService _apiService = ApiService(Constant.apiBaseUrl);  
    // RefreshTokenUseCase refreshTokenUseCase = getIt<RefreshTokenUseCase>();  
    String? accessKnowledgeToken;  
    String? refreshKnowledgeToken;  
    SecureStorageHelper secureStorageHelper;  
  
    KnowledgeApi(this.secureStorageHelper) {  
        _dio.options.baseUrl = Constant.kbApiUrl; // Base URL API  
  
        _dio.interceptors.add(InterceptorsWrapper(  
            onRequest: (options, handler) async {  
                // print(  
                //     "\n-----OnRequest-----1");  
                await _initializeTokens();  
                options.headers['x-jarvis-guid'] = '';  
                if (accessKnowledgeToken != null) {  
                    options.headers['Authorization'] = 'Bearer $accessKnowledgeToken';  
                }  
            }  
        );  
    }  
  
    class UnitApi {  
        final Dio _dio = Dio();  
        // final secureStorageHelper = getIt<SecureStorageHelper>();  
        // final ApiService _apiService = ApiService(Constant.apiBaseUrl);  
        // RefreshTokenUseCase refreshTokenUseCase = getIt<RefreshTokenUseCase>();  
        String? accessKnowledgeToken;  
        String? refreshKnowledgeToken;  
        SecureStorageHelper secureStorageHelper;  
  
        UnitApi(this.secureStorageHelper) {  
            _dio.options.baseUrl = Constant.kbApiUrl; // Base URL API  
  
            _dio.interceptors.add(InterceptorsWrapper(  
                onRequest: (options, handler) async {  
                    // print(  
                }  
            );  
        }  
    }  
}
```

- Sử dụng Provider để quản lý trạng thái statement của các trạng thái state:

- Các notifier của Knowledge bao gồm KnowlegeNotifier để quản lí các trạng thái của Knowledge và AddKnowlegeDialog để quản lí các trạng thái của Dialog Add Knowlege

```

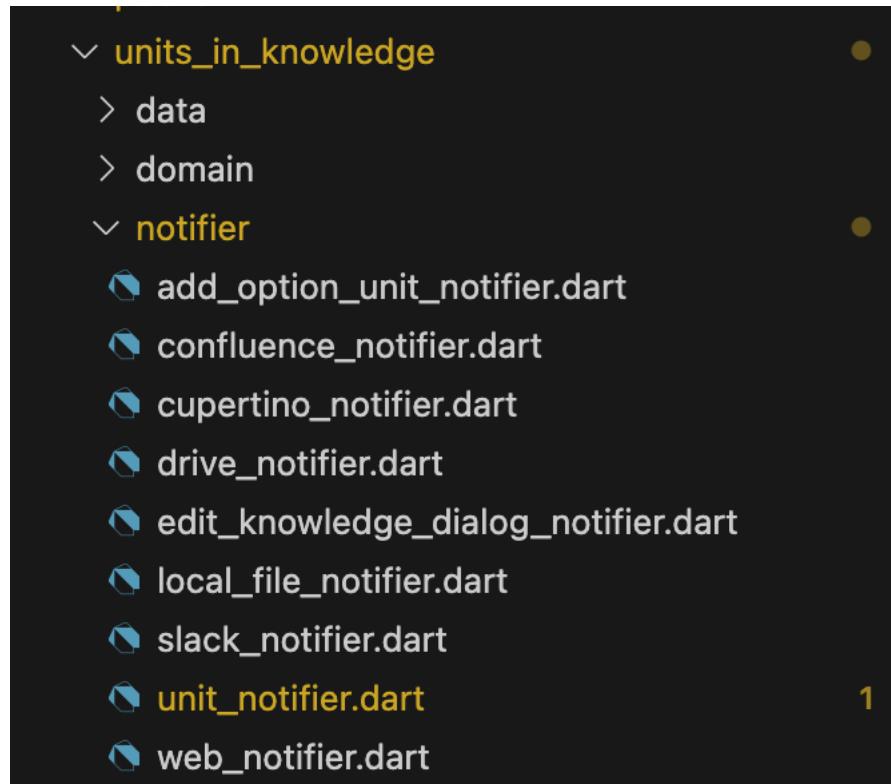
class KnowledgeNotifier with ChangeNotifier {
  GetKnowledgeListUsecase _getKnowledgeListUsecase;      The private field _getKnowledgeListUsecase
  AddKnowledgeUsecase _addKnowledgeUsecase;      The private field _addKnowledgeUsecase
  DeleteKnowledgeUsecase _deleteKnowledgeUsecase;      The private field _deleteKnowledgeUsecase
  EditKnowledgeUsecase _editKnowledgeUsecase;      The private field _editKnowledgeUsecase
  KnowledgeNotifier(this._getKnowledgeListUsecase, this._addKnowledgeUsecase,
    this._deleteKnowledgeUsecase, this._editKnowledgeUsecase);
  String errorString = "";
  bool isLoadingKnowledgeList = false;
  KnowledgeList? knowledgeList;
  int limit = 0;
  bool hasNext = false;

  class AddKnowledgeDialogNotifier extends ChangeNotifier {
    bool isLoadingWhenCreateNewKnowledge = false;
    String errorDisplayWhenNameIsUsed = "";
    void updateNumberTextInTextFiled() {
      notifyListeners();
    }

    void setIsLoadingWhenCreateNewKnowledge(bool isLoading) {
      isLoadingWhenCreateNewKnowledge = isLoading;
      notifyListeners();
    }
    void setErrorDisplayWhenNameIsUsed(String error) {
      errorDisplayWhenNameIsUsed = error;
      notifyListeners();
    }
  }
}

```

- Các Notifier ở Unit gồm nhiều Notifier khác nhau để quản lý các trạng thái tương ứng của các page unit khác nhau như Confluence, Slack, Drive, LocalFile, Web code tương tự như trong thư mục ở git, ở đây chỉ chụp hình các thư mục Notifier.



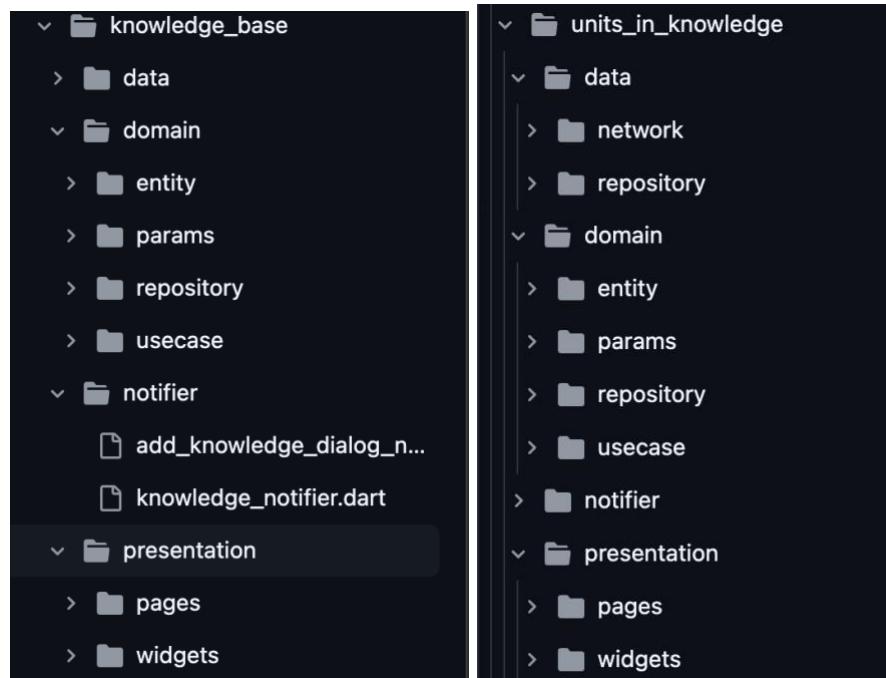
- Sử dụng Dependency Injection bằng thư viện getIt để tiêm các chức năng phụ thuộc vào lẫn nhau. Kết hợp với đó là sử dụng Singleton Pattern để có thể đăng kí và tạo ra các đối tượng state duy nhất cho từng chức năng cần thiết. Và đăng kí Factory Pattern để đăng kí quản lí status của unit.

```

//Units in knowledge:-----
getIt.registerSingleton<UnitNotifier>(UnitNotifier(
    getIt<GetUnitListUsecase>(),
    getIt<DeleteUnitUsecase>(),
    getIt<UpdateStatusUnitUsecase>(),
    getIt<UploadLocalFileUsecase>(),
    getIt<UploadWebUsecase>(),
    getIt<UploadSlackUsecase>(),
    getIt<UploadDriveUsecase>(), "Usecase": Unknown word.
    getIt<UploadConfluenceUsecase>()));
getIt.registerSingleton<AddOptionUnitNotifier>(AddOptionUnitNotifier());
getIt.registerSingleton<EditKnowledgeDialogNotifier>(
    EditKnowledgeDialogNotifier());
getIt.registerFactory<CupertinoNotifier>(() => CupertinoNotifier());
//Unit options:-----
getIt.registerSingleton<LocalFileNotifier>(LocalFileNotifier());
getIt.registerSingleton<WebNotifier>(WebNotifier());
getIt.registerSingleton<DriveNotifier>(DriveNotifier());
getIt.registerSingleton<ConfluenceNotifier>(ConfluenceNotifier());
getIt.registerSingleton<SlackNotifier>(SlackNotifier());

```

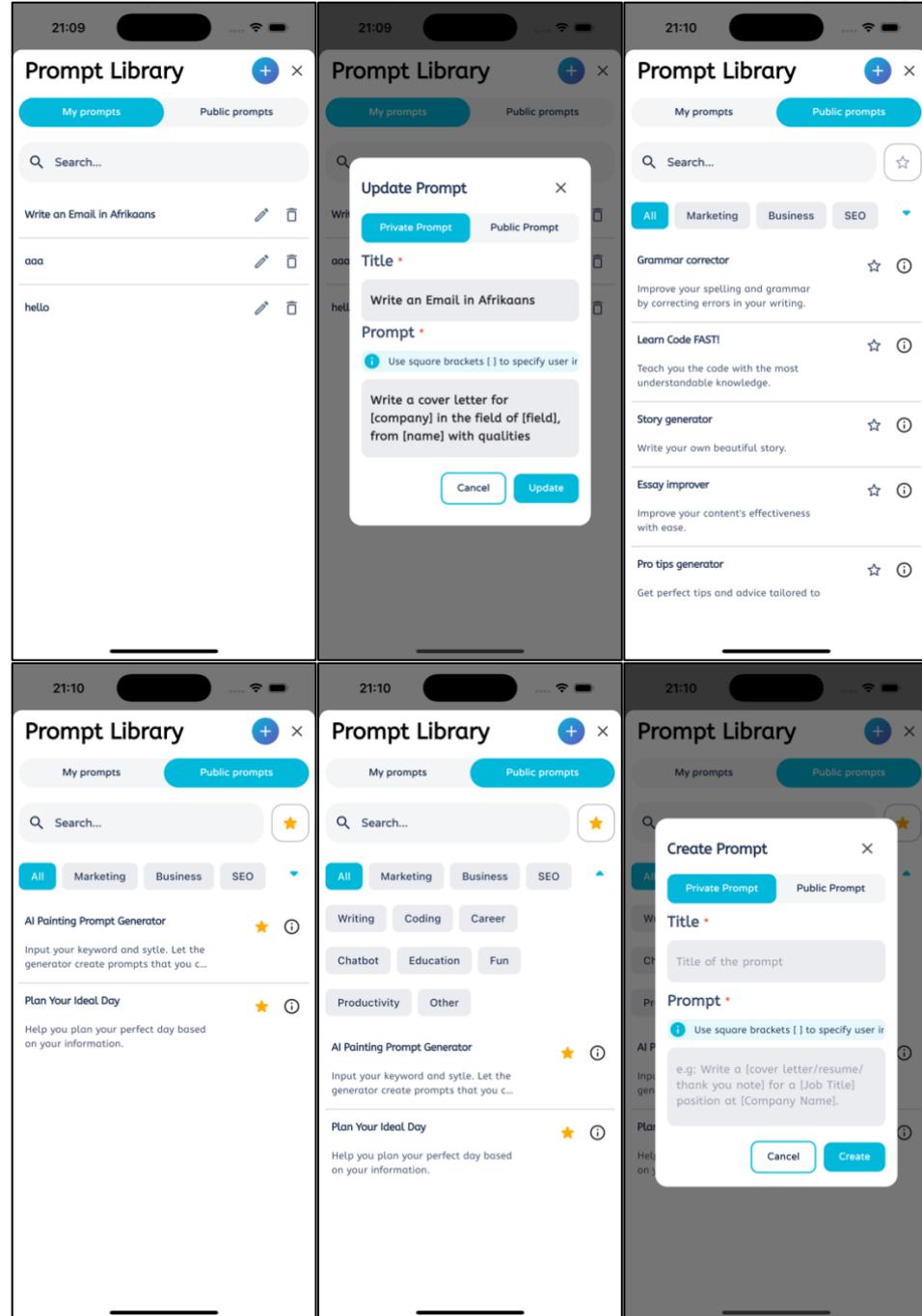
- Cấu trúc tổng quan thư mục của tính năng knowledges và unit tuân theo clean architecture, các lời gọi API chỉ xử lí ở tầng data và các thành phần giao diện chỉ xử lí ở tầng presentation, cũng như ở tầng domain chứa các cấu trúc mẫu và danh sách các usecase chính của knowledge và unit.

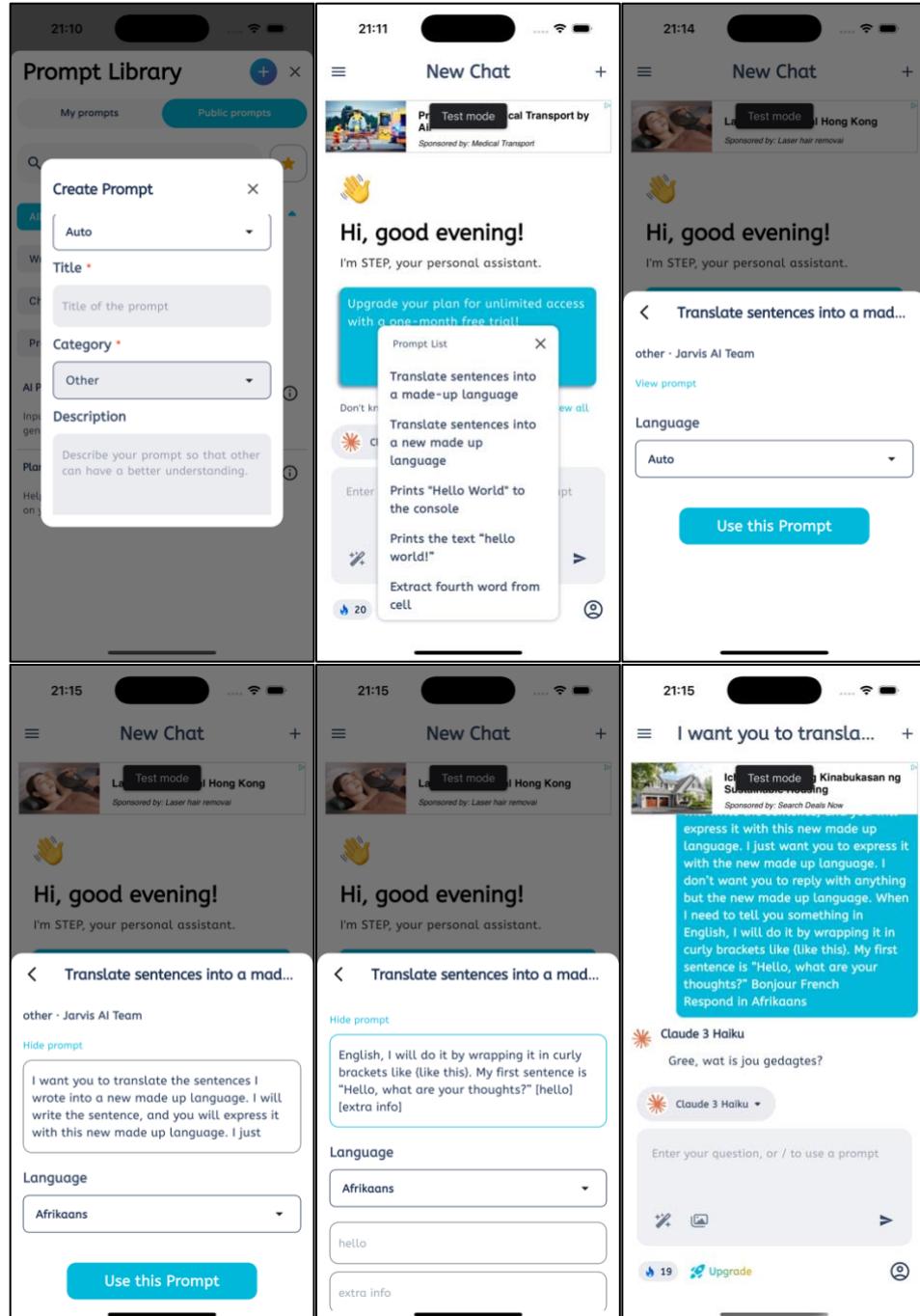


5. Quản lý và sử dụng Prompt

5.1. Giao diện:

- Giao diện bao gồm các tính năng như tạo, xoá, sửa và sử dụng Prompt.





5.2. Kĩ thuật:

- Áp dụng Repository Pattern và thư viện dio: Để liên kết đến Server, lấy và cập nhập dữ liệu liên quan đến Prompt. Kết hợp với đó là sử dụng Singleton Pattern để có thể đăng kí và tạo ra các đối tượng state duy nhất cho từng chức năng cần thiết.
- Áp dụng Dependency Injection sử dụng thư viện getIt để tiêm các phu thuộc của các

```

final getIt = GetIt.instance;

Future<void> initPromptData() async {
    PromptApi promptApi = PromptApi();
    PromptRepository promptRepository = PromptRepositoryImpl(promptApi);

    getIt.registerSingleton<PromptRepository>(promptRepository);
}

final getIt = GetIt.instance;

Future<void> initPromptPresentation() async {
    PrivateFilterState privateFilterState = PrivateFilterState();
    PublicFilterState publicFilterState = PublicFilterState();
    FormModel formModel = FormModel();
    PrivateViewState privateViewState = PrivateViewState(
        repository: getIt<PromptRepository>(),
        filterState: privateFilterState);
    PublicViewState publicViewState = PublicViewState(
        repository: getIt<PromptRepository>(),
        filterState: publicFilterState);
    PromptViewState promptViewState = PromptViewState(
        publicViewState: publicViewState,
        privateViewState: privateViewState);

    getIt.registerSingleton<PublicViewState>(publicViewState);
    getIt.registerSingleton<PrivateViewState>(privateViewState);
    getIt.registerSingleton<PublicFilterState>(publicFilterState);
    getIt.registerSingleton<PrivateFilterState>(privateFilterState);
    getIt.registerSingleton<FormModel>(formModel);
    getIt.registerSingleton<PromptViewState>(promptViewState);
}

```

- Sử dụng Provider để xử lý logic của giao diện:
 - o Phân tách các notifier tuỳ thuộc vào tính năng như:
 - Filter cho Public Prompt.

```

import 'dart:async';
💡
import 'package:flutter/material.dart';

class PublicFilterState extends ChangeNotifier {
    bool _isFavorite = false;
    String _category = 'All';
    String? _searchQuery;
}

```

- Filter cho Private Prompt.

```
class PrivateFilterState extends ChangeNotifier {
    String? _searchQuery;

    Timer? _debounce;

    String? get searchQuery => _searchQuery;

    void setSearchQuery(String query) {
        if (query == _searchQuery) {
            return;
        }

        _debounce?.cancel();
        _debounce = Timer(const Duration(milliseconds: 500), () {
            _searchQuery = query;
            notifyListeners();
       }); // Timer
    }
}
```

- Quản lý Private Prompt.

```
class PrivateViewState extends ChangeNotifier {
    final PromptRepository repository;
    final PrivateFilterState filterState;

    List<PromptModel> _prompts = [];
    bool _isLoading = false;
    bool _isFetchingMore = false;
    final int _itemsRetrieve = 10;
    bool _hasMore = true;
    bool _isAdding = false;
```

- Quản lý Public Prompt.

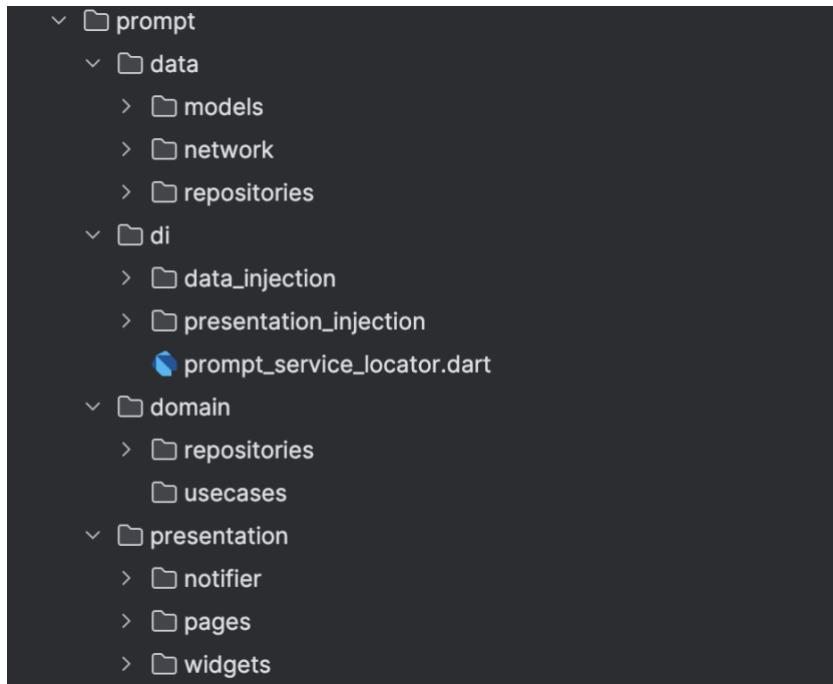
```

class PublicViewState extends ChangeNotifier {
    final PromptRepository repository;
    final PublicFilterState filterState;

    List<PromptModel> _prompts = [];
    bool _isLoading = false;
    bool _isAdding = false;
    bool _isFetchingMore = false;
    final int _itemsRetrieve = 10;
    bool _hasMore = true;
    bool _isExpandedChips = false;
}

```

- Áp dụng tính năng dynamic load khi người dùng kéo xuống cuối để tự động tải thêm Prompt về từ Server.
- Cài đặt debounce time để hạn chế truy vấn tìm kiếm thưa thải.

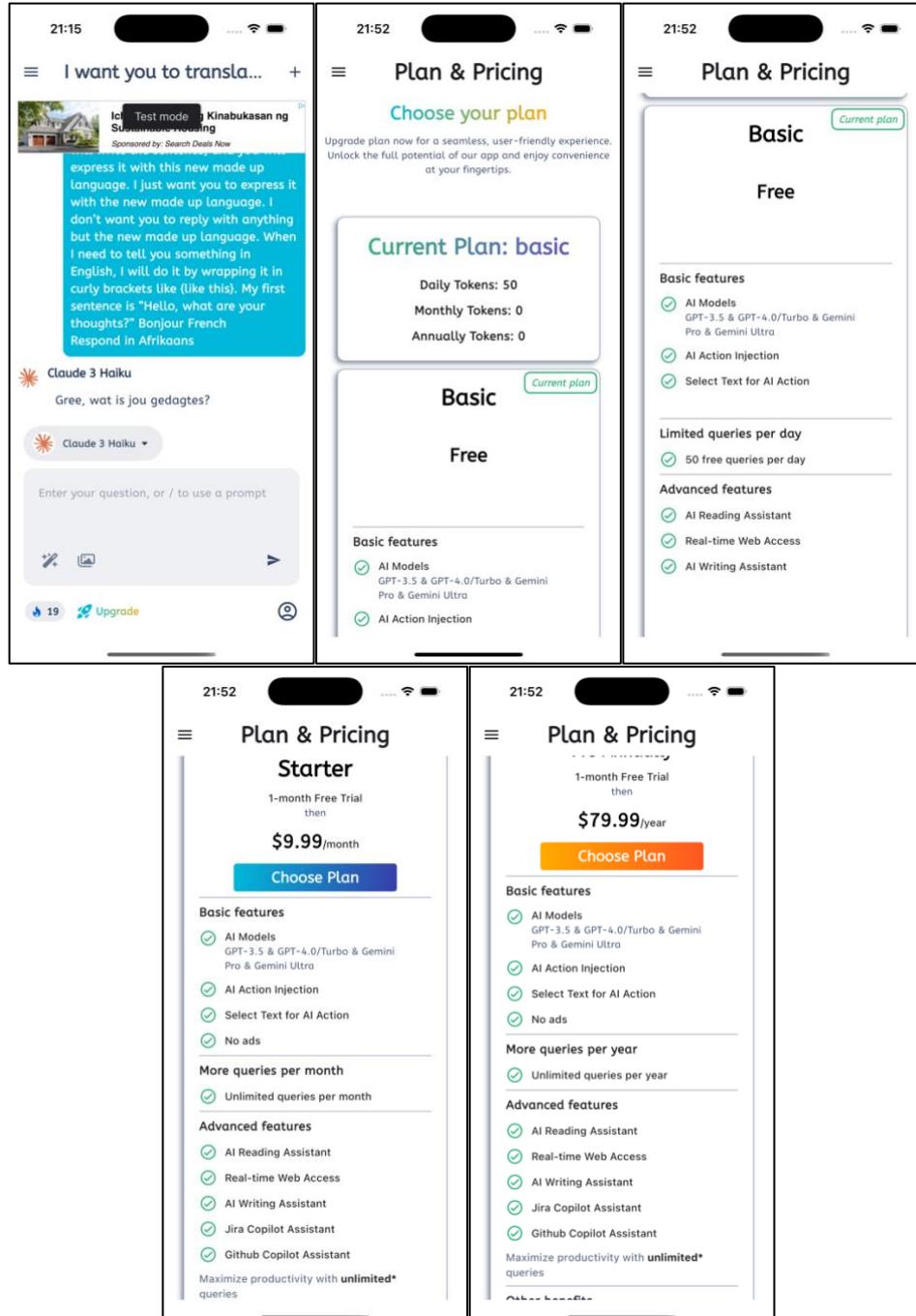


6. Nâng cấp tài khoản lên Pro & Monetization

6.1. Giao diện:

- Giao diện quảng cáo sử dụng Google Admod.

- Giao diện Plan & Pricing thể hiện gói hiện tại của người dùng, các gói lựa chọn như Basic, Starter, Pro.
- Nút đăng ký và feature kèm theo.



6.2. Kỹ thuật:

- Áp dụng các kỹ thuật như Repository kết hợp với thư viện dio để lấy dữ liệu về gói hiện tại của người dùng. Thể hiện gói đó lên trên giao diện sử dụng notifier của thư viện Provider.

```

abstract class SubscriptionRepository {
    Future<Plan> getCurrentPlan();
}

class SubscriptionRepositoryImpl extends SubscriptionRepository {
    final ApiSubscription _apiSubscription;
    SubscriptionRepositoryImpl(this._apiSubscription);
    @override
    Future<Plan> getCurrentPlan() async {
        late Response<dynamic> response;
        try {
            response = await _apiSubscription.getUsage(Constant.usageEndpoint);
        } catch (e) {
            throw("Error getting usage");
        }

        return PlanModel.fromJson(response.data).toPlanEntity();
    }
}

class SubscriptionNotifier extends ChangeNotifier {
    Plan? plan;
    bool isLoading = false;
    bool hasError = false;

    final LogoutUseCase _logoutUseCase;
    final GetSubscriptionUsecase _subscriptionUsecase;
    SubscriptionNotifier(this._logoutUseCase, this._subscriptionUsecase);

    Future<void> getPlan() async {
        isLoading = true;
        hasError = false;
        notifyListeners();
        try {
            plan = await _subscriptionUsecase.call();
        } catch (e) {
            hasError = true;
            await _logoutUseCase.call(params: null);
        } finally {
            isLoading = false;
            notifyListeners();
        }
    }
}

```

- Sử dụng Google Admob để cài quảng cáo lên giao diện, ta tạo quản cáo trên website và lấy khoá. Sau đó truyền khoá vào build của ứng dụng và sử dụng thư viện tương ứng để thực hiện việc hiển thị quảng cáo lên giao diện.

```

class AdModService {
    static String? getBannerAdUnitId() {
        if (Platform.isAndroid) {
            return 'code for android';
        } else if (Platform.isIOS) {
            return 'code for ios';
        }
        return null;
    }

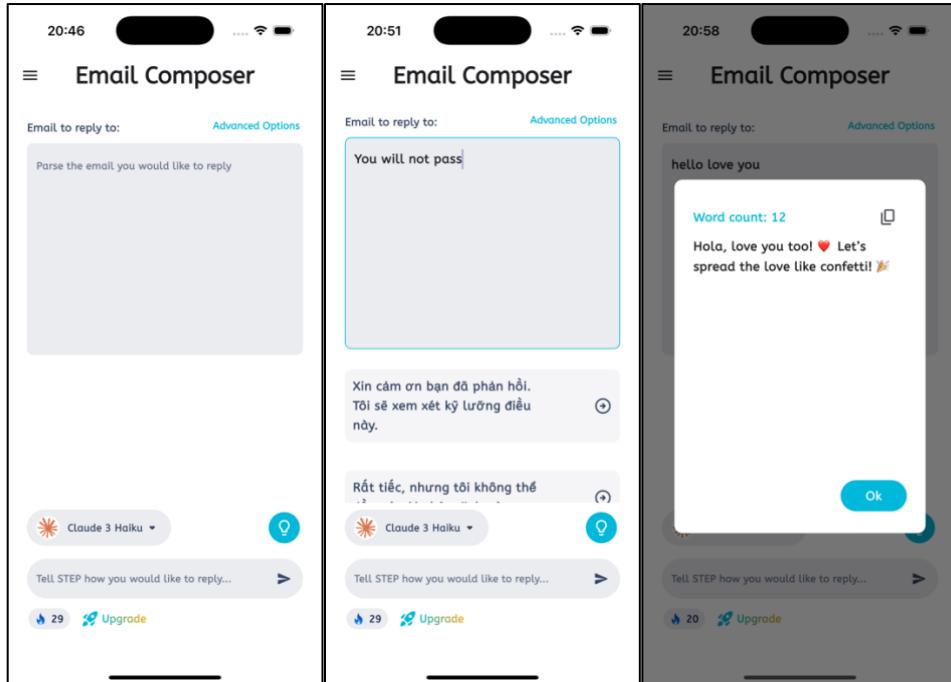
    static BannerAdListener listener = BannerAdListener(
        onAdLoaded: (ad) => {},
        onAdFailedToLoad: (ad, error) {
            ad.dispose();
        },
        onAdOpened: (ad) => {},
        onAdClicked: (ad) => {},
    );
}

```

7. Soạn Email với AI

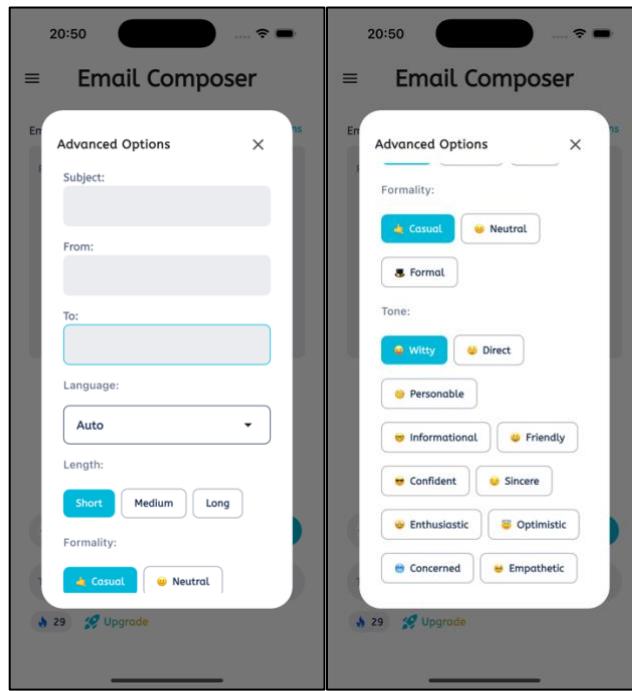
7.1. Giao diện:

- Phản hồi email:

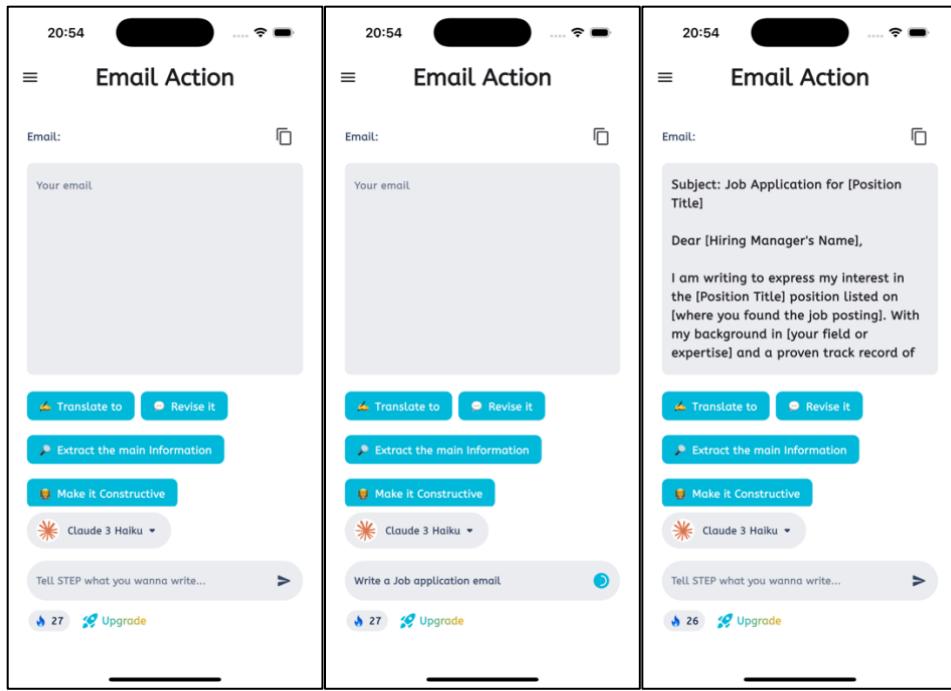


- Vùng Text field giúp người dùng nhập Email mong muốn phản hồi vào.
- Nút bóng đèn giúp người dùng khởi tạo ý tưởng để phản hồi Email.

- Drop box cho phép người dùng chọn các loại model.
- Chat bar cho phép người dùng gửi hành động đến Server.
- Hiển thị số Token của người dùng và nút nâng cấp Gói dịch vụ.
- Nút tuỳ chỉnh nâng cao:



- Tiêu đề Email.
 - Người gửi.
 - Người nhận.
 - Formality, Tone, Length.
 - Ngôn ngữ.
- Viết email:



- Cho phép người dùng nhập Email mong muốn.
- Thực hiện các tác vụ như: Dịch, Xem lại, Tách thông tin, Viết lại,...
- Chọn mô hình người dùng mong muốn.
- Hiển thị số Token của người dùng.
- Chat bar cho phép người dùng tạo Email theo yêu cầu.

7.2. Kĩ thuật:

- Áp dụng Repository Pattern, Interceptor và thư viện dio để quản lý việc khởi tạo, gửi và phản hồi Email.

```
abstract class ResponseEmailRepository {
    Future<ResponseEmail> generateResponseEmail({required AiEmail aiEmail});
    Future<List<String>> generateIdeasEmail({required AiEmail aiEmail});
    Future<ResponseEmail> composeEmail({required ComposeEmail composeEmail});
}
```

```

class ApiResponseEmail {
    final Dio _dio = Dio();
    final secureStorageHelper = getIt<SecureStorageHelper>();
    final ApiService _apiService = ApiService(Constant.apiUrl);
    // RefreshTokenUseCase refreshTokenUseCase = getIt<RefreshTokenUseCase>()
    String? accessToken;
    String? refreshToken;

    ApiResponseEmail() {
        _dio.options.baseUrl = Constant.apiUrl; // Base URL API

        _dio.interceptors.add(InterceptorsWrapper(
            onRequest: (options, handler) async {

                await _initializeTokens();
                options.headers['x-jarvis-guid'] = '';
                if (accessToken != null) {
                    options.headers['Authorization'] = 'Bearer $accessToken';
                }

                return handler.next(options);
            },
            onResponse: (response, handler) {

```

...

```

        class ResponseEmailRepositoryImpl extends ResponseEmailRepository {
            final ApiResponseEmail _apiResponseEmail;

            ResponseEmailRepositoryImpl(this._apiResponseEmail);

            @override
            Future<ResponseEmail> generateResponseEmail({required AiEmail aiEmail}) async {

```

- Áp dụng Dependency Injection sử dụng thư viện getIt:

```

final getIt = GetIt.instance;

Future<void> initEmailComposerData() async {
    getIt.registerSingleton<ApiResponseEmail>(ApiResponseEmail());
    getIt.registerSingleton<ApiUsage>(<ApiUsage>());
    getIt.registerSingleton<ResponseEmailRepository>(<ResponseEmailRepository>(getIt<ApiResponseEmail>()));
    getIt.registerSingleton<UsageRepository>(<UsageRepository>(apiUsage: getIt<ApiUsage>()));
}

final getIt = GetIt.instance;

Future<void> initEmailComposerDomain() async {
    getIt.registerSingleton<ComposeEmailUsecase>(
        ComposeEmailUsecase(repository: getIt<ResponseEmailRepository>()));
    getIt.registerSingleton<GenerateResponseEmailUsecase>(
        GenerateResponseEmailUsecase(
            repository: getIt<ResponseEmailRepository>()));
    getIt.registerSingleton<GenerateIdeaUsecase>(
        GenerateIdeaUsecase(repository: getIt<ResponseEmailRepository>()));
    getIt.registerSingleton<GetUsageUsecase>(
        GetUsageUsecase(repo: getIt<UsageRepository>()));
}

Future<void> initEmailComposerServices() async {
    await initEmailComposerData();
    await initEmailComposerDomain();
    await initEmailComposerPresentation();
}

```

```

final GetIt getIt = GetIt.instance;

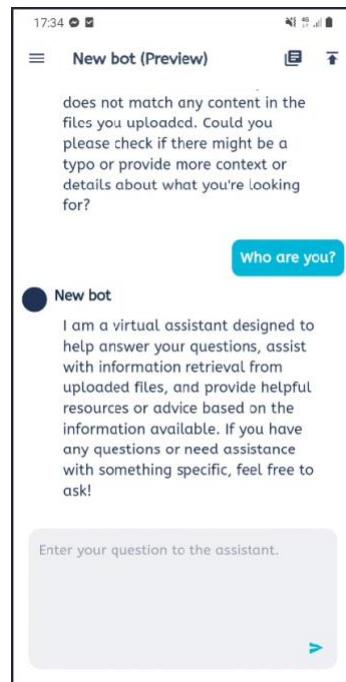
Future<void> initEmailComposerPresentation() async {
    getIt.registerSingleton<EmailComposerNotifier>(EmailComposerNotifier(
        generateResponseEmailUsecase: getIt<GenerateResponseEmailUsecase>(),
        generateIdeaUsecase: getIt<GenerateIdeaUsecase>()));
    getIt.registerSingleton<UsageTokenNotifier>(
        UsageTokenNotifier(usageTokenUsecase: getIt<GetUsageUsecase>()));
    getIt.registerSingleton<AiActionNotifier>(
        AiActionNotifier(composeEmailUsecase: getIt<ComposeEmailUsecase>()));
}

```

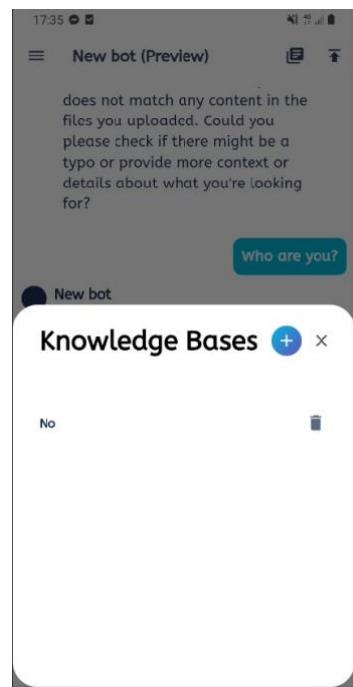
8. Preview chat với personal assistant

8.1. Giao diện:

- Preview chat:

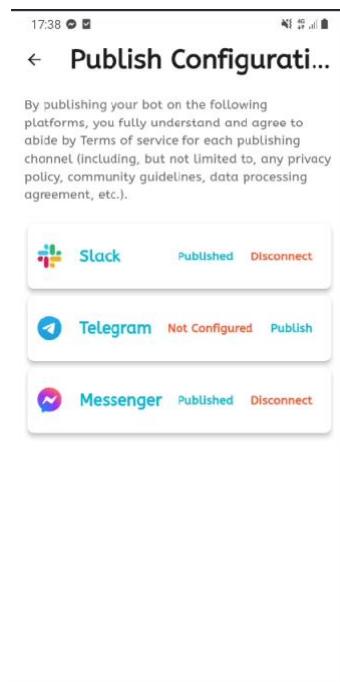


- Lịch sử cuộc trò chuyện preview này sẽ được lưu lại ứng với từng bot. Nhưng lịch sử này sẽ hoàn toàn thuộc một thread khác với thread khi bạn chat với bot trong chat widget (sau khi đã thêm)
- Hai nút bấm ở góc trên là giao diện quản lý knowledge-base của con assistant, và publish assistant:
- Quản ký KB:

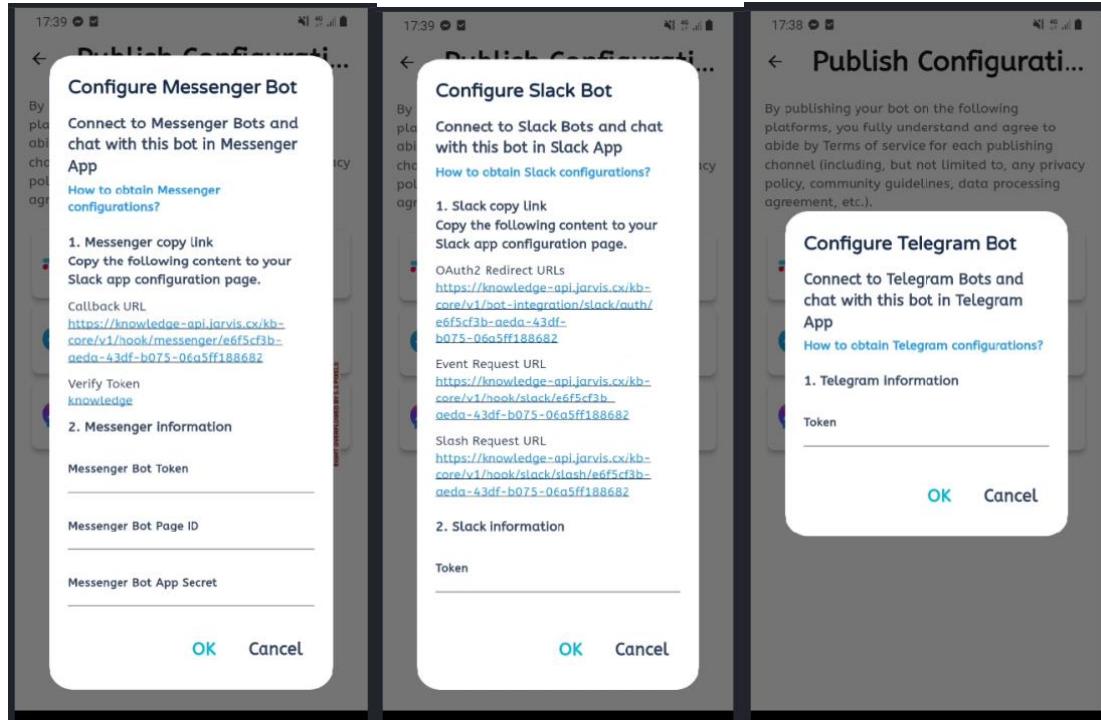


(Bạn có thể thêm mới hoặc xóa KB ra khỏi dữ liệu của Assistant.)

- Publish:

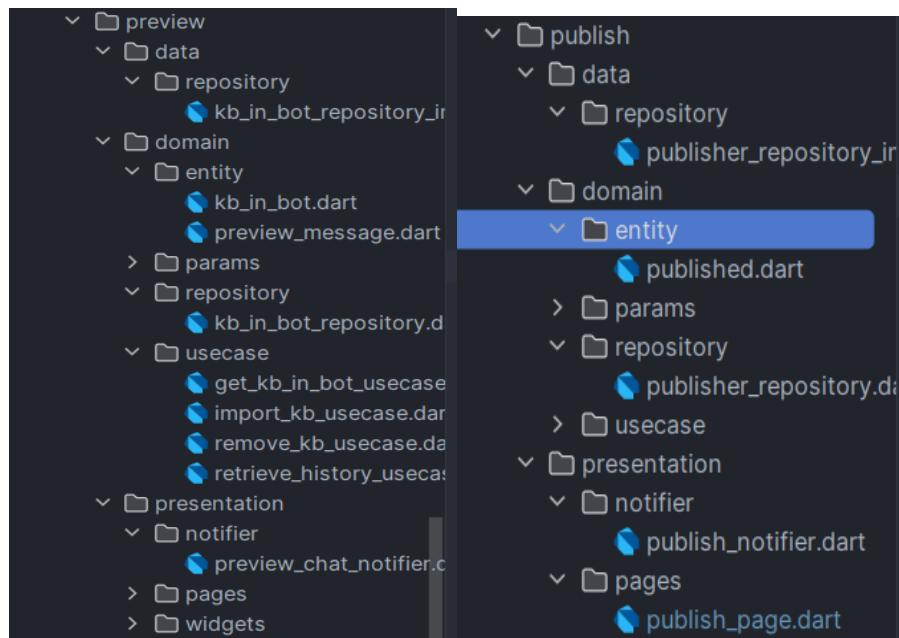


- Giao diện publish của Messenger, Telegram, Slack:



- Giao diện publish chứa đường dẫn tới hướng dẫn chính thức của Jarvis.
- Cần thực hiện đúng theo hướng dẫn để có các Token thực hiện Publish. Tại đây bên FE sẽ gọi API Validate và Publish cùng lúc để giảm thao tác cho người dùng.

8.2. Kĩ thuật:



- Áp dụng Repository Pattern, tuân theo clean architecture với cấu trúc 3 tầng.
- Áp dụng Dependency Injection sử dụng thư viện getIt:
- Áp dụng REST để thực hiện các dịch vụ API với phía Backend. Việc giao tiếp chỉ diễn ra ở tầng data, tại Impl của Repo:

```

class BotListRepositoryImpl extends BotListRepository {
  final ApiService _restClient = ApiService(Constant.kbApiUrl);
  final SecureStorageHelper _secureStorageHelper;

  BotListRepositoryImpl(this._secureStorageHelper);
  @override
  Future<BotListResDto?> getBotList(String? query) async{
    String? kbAccessToken = await _secureStorageHelper.kbAccessToken;
    if (kbAccessToken == null) {
      throw -1;
    }

    var headers = {
      'x-jarvis-guid': '',
      'Authorization': 'Bearer $kbAccessToken'
    };

    String finalQuery = query?? "";
    var request = await _restClient.get(
      "${Constant.botGetEndpoint}=$finalQuery${Constant.botGetOrderSet}${Constant.botOffset}${Constant.botLimit}",
      headers: headers
    );
    String stream = await request.stream.bytesToString();
    print(stream); Don't invoke 'print' in production code.
    if (request.statusCode == 200) {
      BotListResDto bots = BotListResDto.fromJson(
        jsonDecode(stream)
      );
      return bots;
    }
  }
}

```

40:19 - CPLE - 1

- Một ví dụ nhỏ ở getBotList, sử dụng REST gọi API đến BE đổi với Endpoint GetAssistants.