專案架構說明

使用 Provider(狀態管理) , MVVM（design pattern）, get_it（依賴注入、Service Locator）

```
├── core
│   ├── models（存放資料類型）
│   │   └── user.dart
│   ├── services（發 api）
│   │   ├── api_service.dart
│   └── utils（共用方法）
│       └── utils.dart
├── locator.dart（ViewModel , Services 註冊）
├── main.dart（程式進入點）
├── routers.dart（路由定義）
├── ui
│   ├── views
│   │   ├── base_view.dart（每個 view 須 return 此 widget）
│   └── widgets（view 裡面的局部元件）
└── viewmodels
    ├── base_view_model.dart（所有 ViewModel 須繼承此類）
    ├── global_view_model.dart（存放全域變數）
```

基本規則

1. 所有 view 須 return BaseView() 並綁定 ViewModel
2. 所有 ViewModel 須 extends BaseViewModel
3. 全域變數寫在 global_view_model.dart
4. api 一律寫在 services
5. view 須使用 api 資料時，一律透過 ViewModel（使用 locator 注入 services）
6. ViewModel , Services 須在 locator 進行註冊
7. 路由定義在 routers.dart

範例

創建一個頁面

1.在 **viewmodels** 建立 **[檔案名稱]_view_model.dart**，並 extends BaseViewModel：

```
class NewViewModel extends BaseViewModel {
  bool viewModelVariable = false; // 在 model 設置變數
}
```

2.在 locator.dart 綁定剛剛建立的 ViewModel

```
void setupLocator() {
  locator.registerLazySingleton(() => BaseViewModel());
  locator.registerLazySingleton(() => GlobalViewModel());
  locator.registerLazySingleton(() => NewViewModel()); // 綁定
}
```

3.在 **views** 建立 **[檔案名稱]_view_model.dart**，並 return BaseView ，設定回傳的 widget 以及綁定的 ViewModel：

```
class NewView extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return BaseView(
        builder: (context, model, child) {
          print(model.viewModelVariable); // 使用 ViewModel 的變數
          return Scaffold(); // 回傳 widget
        },
        modelProvider: () => NewViewModel(),
    );
```

```
  }
}
```

建立 API Services

1.在 services 建立 **[檔案名稱]_service.dart**，並撰寫相關程式碼：

```
class ApiService {
  Future<String?> getToken() async {
    var url =
Uri.parse('https://tpfishdev.hamastar.com.tw/connect/token');
    var response = await http.post(
      url,
      headers: {
        'Content-Type': 'application/x-www-form-urlencoded',
      },
      body: {
        'grant_type': 'password',
        'client_id': 'TaipeiFishTradingSystem_Dev',
        'client_secret': '1q2w3e*',
        'username': 'admin',
        'password': '1q2w3E*',
        'scope': 'TaipeiFishTradingSystem',
      },
    );

    if (response.statusCode == 200) {
      // 成功
```

```
      print('success:${response.body}');
      var responseData = jsonDecode(response.body);
      return responseData['access_token'];
    } else {
      // 失敗
      print('fail');
      return null;
    }
  }
}
```

2. 在 locator.dart 註冊 service

```
void setupLocator() {
  locator.registerLazySingleton(() => BaseViewModel());
  locator.registerLazySingleton(() => GlobalViewModel());

  locator.registerLazySingleton(() => ApiService()); // 註冊
}
```

3. 在 ViewModel 使用 locator 注入，並使用 api

```
class HomeViewModel extends BaseViewModel {
  // 使用 locator 注入 api service
  final ApiService _apiService = locator<ApiService>();

  // 發 Api
  void deleteTrades() async {
    setBusy(true); // 若 widget 須更新，請設置 setBusy()

    // 使用 api service
    String? accessToken = await _apiService.getToken();
    setBusy(false);
  }
}
```

注意：

1.若 View 須使用 api 資料，一律透過 ViewModel 注入 api service 後進行存取
2.若 View 在取得完資料後須更新頁面，請設置 setBusy()

全域變數

1.在 global_view_model.dart 設置全域變數、getter 、setter

```
class GlobalViewModel extends ChangeNotifier {
  bool _globalVariable = false;
  bool get globalVariable => _globalVariable;
  void set setGlobalVariable(bool value) {
    this._globalVariable = value;
  }
}
```

2-1.在 View 中使用

```
class NewView extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return BaseView(
      builder: (context, model, child) {
        print(model.globalViewModel.globalVariable);// 使用全域變數
        return Scaffold();
      },
      modelProvider: () => NewViewModel(),
    );
  }
}
```

2-2.在 ViewModel 中使用

```
class NewViewModel extends BaseViewModel {
  void printGlobalVariable() {
    print(globalViewModel.globalVariable); // 使用全域變數
```

```
    }
}
```