

فهرست

2	مقدمه
2	ساختار پروژه
3	فایل pubspec.yaml
4	پوشه asset
5	ماژول common
5	پوشه ui
5	پوشه custom_bottom_navigation
5	پوشه util
5	پوشه constant
5	پوشه base
10	ماژول data
10	ماژول local
10	ماژول model
11	ماژول remote
11	ماژول repository & domain
11	ماژول features
11	ماژول navigation
12	فایل app_bindings
12	فایل app_router
12	فایل main

مقدمه

در این نوشتار فرض بر این گرفته شده است شما یک توسعه دهنده فلاتر با زبان برنامه نویسی دارت هستید. در نتیجه با توجه به شفاف بودن کد پروژه از توضیحات اضافه پرهیز شده است.

ساختار پروژه

پروژه با ساختار MVVM طراحی شده است و شامل ساختار درختی زیر است:

```
super_app_base_flutter 📁  
├── android 📂 └── ↗  
├── ios 📂 └── ↗  
└── lib 📂 └── ↗  
    ├── common 📂 └── ↗  
    ├── constant 📂 └── ↗  
    └── ui 📂 └── ↗  
        ├── custom_bottom_navigation 📂 └── ↗  
        └── util 📂 └── ↗  
    ├── features 📂 └── ↗  
    ├── detail 📂 └── ↗  
    ├── home 📂 └── ↗  
    └── main.dart 📄 └── ↗  
    ├── app_router.dart 📄 └── ↗  
    └── app_bindings.dart 📄 └── ↗  
└── test 📂 └── ↗  
    └── pubspec.yaml 📄 └── ↗
```

پروژه براساس معماری MVVM بنا شده است و استوار بر اصول SOLID و Clean Architecture می باشد.
اگر بر این مفاهیم اشراف ندارید ابتدا تسلط پیدا کنید). تا از این طریق توسعه و نگهداری پروژه آسان تر گردد
و هر کس بتواند مازول یا فیچر خود را با یک زیر ساخت مشترک توسعه دهد. پس اگر بر مفاهیم ذکر شده
تسلط ندارید ابتدا آنها را بررسی و مطالعه کنید. حال به ساختار پروژه اشاره می کنیم و هر زمان که نیاز به
توضیح مفاهیم باشد آن ها تشریح خواهیم کرد.

فایل pubspec.yaml

در این فایل لایبرری ها و کانفیگ ریسورس برنامه قرار دارد:

```
name: super_app_base_flutter
description: "A base Project"
publish_to: 'none'

version: 1.0.0+1

environment:
  sdk: ^3.6.0

dependencies:
  flutter:
    sdk: flutter
  dio: ^5.7.0
  get: ^4.6.6
  connectivity_plus: ^6.1.0
  flutter_secure_storage: ^9.2.4
  cached_network_image: ^3.4.1
  smooth_page_indicator: ^1.2.0+3
  go_router: ^14.8.0
  flutter_inappwebview: ^6.1.5
  cupertino_icons: ^1.0.8

dev_dependencies:
  flutter_test:
    sdk: flutter

  flutter_lints: ^5.0.0

flutter:
  uses-material-design: true

  assets:
    - assets/images/

  fonts:
    - family: IranYekan
```

```
fonts:  
  - asset: assets/fonts/iran_yekan_xfanum_regular.ttf  
- family: Modam  
  fonts:  
    - asset: assets/fonts/modam_fa_num_regular.ttf
```

همانطور که مشاهده می کنید علاوه بر تنظیمات مربوط به نام و ورژن برنامه و نسخه زبان دارد، بعد از آنها لایبرری هایی را که در برنامه استفاده شده است می بینیم. به ترتیب علت استفاده از این دیپندنسی ها:

dio : برای ارتباط RESTFul با سرور

get : برای تزریق وابستگی

connectivity_plus : برای بررسی اتصال کاربر یا دستگاه به اینترنت

flutter_secure_storage : برای ذخیره اطلاعات به صورت امن در حافظه محلی

cached_network_image : برای نمایش تصاویر اینترنتی

smooth_page_indicator : برای نمایش نشانگر یا مشخصه های زیر اسلایدر

go_router : برای مسیر دهی و روتینگ برنامه برای همه پلتفرم ها

flutter_inappwebview : برای نمایش برنامه هایی که به صورت وب هستند و باید در برنامه در قالب موبایل یا وب نمایش داده شوند.

برای مطالعه کامل مستندات هر کدام از دیپندنسی ها می توانید به سایت pub.dev به عنوان مرجعه و ریپازیتوری اصلی لایبرری های فلاتر و دارت مراجعه کنید.

در ادامه نیز تصاویر و فونت های مورد استفاده به برنامه شناسانده شده است.

پوشه asset

فولدر asset شامل resource برنامه می شود. از قبیل فونت ها، تصاویر و

کل کدهای یک پروژه فلاتر در فولدر lib پیاده سازی می شود.

ماژول common

در فolder common آنچه که در اپ به صورت عمومی استفاده می شود، قرار داده شده است.

پوشه ui

رابط های کاربری شخصی سازی شده که در فolder ui قرار دارند.

پوشه custom_bottom_navigation

این پوشه شامل باتم نویگیشنی است که در برنامه استفاده شده است و به صورت عمومی قرار داده شده تا اگر نیاز است در جای دیگر از آن استفاده شود. در غیر این صورت متفاوت بودن نوع و رابط کاربری باتم نویگیشن توسعه دهنده‌گان می توانند از باتم نویگیشن مورد علاقه خود استفاده کنند.

پوشه util

فولدر util شامل کلاس‌ها و متدهایی است که به صورت کاربردی استفاده می شوند و می توانند در جای جای برنامه استفاده داشته باشند. به طور مثال بررسی تلفن همراه یا اعتبار سنجی رمز عبور که شرایط لازم را داشته باشد.

پوشه constant

در این پوشه مقادیر ثابتی که در برنامه استفاده می شوند قرار دارند. از قبیل رشته‌ها، رنگ‌ها، مسیر تصاویر، فونت‌ها و ... در فولدر extension نیز کدهایی برای تسهیل استفاده از هر کانسپتی وجود دارد. به طور مثال اکستنشنی که بر روی کلاس استرینگ نوشته شده بر قابلیت‌ها و فانکشن‌های آن افزوده است.

پوشه base

اصلی‌ترین پوشه‌ای که در اینجا داریم که به طور عمومی در همه برنامه استفاده می شود. از آنجا که ما از معماری MVVM استفاده می کنیم. سعی بر آن شده با توجه بر اصول SOLID وظایف هر کلاس یا فانکشن یا

کامپوننت مشخص شده باشد. ما با دو کلاس حیاتی در برنامه موجه هستیم با نام های `base_widget` و `.base_view_model`

در این کلاس ها وظایفی که بین هر صفحه در برنامه به صورت عمومی و مشترک هستند قرار داده شده اند.
کلاس `BaseViewModel` به عنوان ویومدل پایه استفاده می شود و همه ویو مدل ها در برنامه از آن ارث بری می کنند.

```
import 'package:get/get.dart';
import '../navigation/navigation_command.dart';

abstract class BaseViewModel extends GetxController {
    final _navigationCommand = Rx<NavigationCommand?>(null);
    final _snackBarMessage = Rx<String?>(null);

    Stream<NavigationCommand?> get navigationCommand => _navigationCommand.stream;
    Stream<String?> get snackBarMessage => _snackBarMessage.stream;

    void navigateTo(String route, {dynamic arguments}) {
        _navigationCommand.value = To(route, arguments: arguments);
    }

    void navigateBack() {
        _navigationCommand.value = Back();
    }

    void navigateAndClearStack(String route, {dynamic arguments}) {
        _navigationCommand.value = ClearAndNavigate(route, arguments: arguments);
    }

    void showSnackBar(String message) {
        _snackBarMessage.value = message;
        // Clear the message immediately after emitting
        Future.microtask(() => _snackBarMessage.value = null);
    }

    void deleteBindingIfRegistered<T>() {
        if (Get.isRegistered<T>()) {
            Get.delete<T>();
        }
    }

    @override
    void onClose() {
        _navigationCommand.close();
        _snackBarMessage.close();
        super.onClose();
    }
}
```

در این کلاس وظیفه واسط نویگیشن و روتینگ برنامه را برعهده دارد. با استفاده از استریم ها و ری اکتیو پروگرمینگ با کلاس `BaseWidget` ارتباط گرفته و نویگیشن اصلی با کتابخانه `GoRouter` انجام می شود. همین منوال برای نشان دادن اسنک بار و پیام ها وجود دارد.

از طرفی برای اینکه از `GoRouter` به عنوان روتینگ برنامه استفاده کنیم تا برنامه بتواند از وب نیز پشتیبانی کند نمی توانیم از روتینگ کتابخانه `GetX` استفاده کنیم. اما برای استفاده از قابلیت تزریق وابستگی یا `Binding dependency injection` ، بایستی مدیریت حافظه را برعهده بگیریم. این کار را برای حذف `usecase` ها یا وابستگی های دیگر متدهای `deleteBindingIfRegistered` ایجاد شده و باید در ویو مدل های فرزند از آن استفاده شود.

در کلاس `base widget` همه آنچه به صورت مشترک بین صفحات برنامه است، قرار گرفته است.

```
import 'dart:async';
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:go_router/go_router.dart';
import '../navigation/navigation_command.dart';
import '../constant/app_colors.dart';
import '../ui/custom_app_bar.dart';
import '../util/utility.dart';
import 'base_view_model.dart';

class BaseWidget<T extends BaseViewModel> extends StatefulWidget {
    final T viewModel;
    final String title;
    final bool? showTopBar;
    final VoidCallback? onBackPressed;
    final bool showBackArrow;
    final Widget? iconButtonAppBar;
    final List<Widget>? actions;
    final Widget Function(BuildContext, EdgeInsets) content;
    final Widget? bottomBar;
    final bool activePadding;
    final Color backgroundColor;
    final Widget? drawer;
    final Widget? endDrawer;
    final bool useSpace;
    final Widget? floatingActionButton;

    BaseWidget({
        super.key,
        required this.viewModel,
        required this.content,
        this.title = '',
        bool? showTopBar,
        this.onBackPressed,
        this.showBackArrow = false,
        this.iconButtonAppBar,
```

```

        this.actions,
        this.bottomBar,
        this.activePadding = true,
        this.backgroundColor = AppColors.whiteColor,
        this.drawer,
        this.endDrawer,
        this.useSpace = false,
        this.floatingActionButton,
    )): showTopBar = showTopBar ?? (!Utility.isWeb() ? true : false);

    @override    BaseWidgetState<T> createState() => BaseWidgetState<T>();
}

class BaseWidgetState<T extends BaseViewModel> extends State<BaseWidget<T>> {
    late final StreamSubscription _snackSub;
    late final StreamSubscription _navSub;
    double? _previousWidth;
    bool preventClose = false;
    Timer? _resizeTimer;
    @override    void initState() {
        super.initState();
        // Subscribe once to the snackBar stream.
        _snackSub = widget.viewModel.snackBarMessage.listen((message) {
            if (message != null) {
                WidgetsBinding.instance.addPostFrameCallback((_) {
                    Utility.showTopSnackBar(context, message, showTopBar : widget.showTopBar! );
                });
            }
        });
        // Subscribe once to the navigation commands.
        _navSub = widget.viewModel.navigationCommand.listen((command) {

            if (command != null /*&& mounted*/) {
                final router = Get.find<GoRouter>();
                if (command is To) {
                    Utility.isWeb() ? router.go(command.route, extra: command.arguments):
                        router.push(command.route, extra: command.arguments);
                } else if (command is Back) {
                    router.pop();
                } else if (command is ClearAndNavigate) {
                    router.go(command.route, extra: command.arguments);
                }
            }
        });
    }

    @override
    void dispose() {
        if(preventClose) return;

        _snackSub.cancel();
        _navSub.cancel();
        // Delete the binding when this widget is popped.
        if (Get.isRegistered<T>()) {
            Get.delete<T>();
        }

        super.dispose();
    }

    @override
    Widget build(BuildContext context) {
        final padding = const EdgeInsets.all(16);

```

```

Widget contentWidget = GestureDetector(
  onTap: () => FocusManager.instance.primaryFocus?.unfocus(),
  child: Scaffold(
    extendBody: true,
    resizeToAvoidBottomInset: !Utility.isWebMobile(),
    backgroundColor: widget.backgroundColor,
    appBar: widget.showTopBar!
      ? CustomAppBar(
          title: widget.title,
          iconButtonAppBar: widget.iconButtonAppBar,
          showBackArrow: widget.showBackArrow,
          actions: widget.actions,
          onBackPressed: widget.onBackPressed,
        )
      : null,
    drawer: widget.drawer,
    endDrawer: widget.endDrawer,
    body: Directionality(
      textDirection: TextDirection rtl,
      child: Padding(
        padding: widget.activePadding
          ? EdgeInsets.only(
              left: padding.left,
              right: padding.right,
              top: padding.top,
              bottom: widget.bottomBar != null ? padding.bottom : 0,
            )
          : EdgeInsets.zero,
        child: Row(
          children: [
            if (widget.useSpace) const Expanded(child: SizedBox()),
            Expanded(flex: 4, child: widget.content(context, padding)),
            if (widget.useSpace) const Expanded(child: SizedBox()),
          ],
        ),
      ),
    ),
    floatingActionButton: widget.floatingActionButton,
    bottomNavigationBar: Row(children: [
      if (widget.useSpace) const Expanded(child: SizedBox()),
      if (widget.bottomBar != null) Expanded(child: widget.bottomBar!),
      if (widget.useSpace) const Expanded(child: SizedBox()),
    ]),
  ),
);

// **Only wrap with LayoutBuilder for non-mobile & non-web mobile**
if (Utility.isWebDesktop() || Utility.isDesktop()) {
  return LayoutBuilder(builder: (context, constraints) {
    if (constraints.biggest.width != _previousWidth) {
      preventClose = true;
      _previousWidth = constraints.biggest.width;
      _resizeTimer?.cancel();

      // Reset preventClose after delay
      _resizeTimer = Timer(Duration(milliseconds: 500), () {
        if (mounted) {
          preventClose = false;
        }
      });
    }
    return contentWidget;
  });
} else {
  return contentWidget;
}

```

}

تمام ساختار اولیه و مشترک یک صفحه در این ساختار قرار دارد. **Scaffold** و ساختار آن در اینجا پیاده سازی شده تا در صفحات ویجت ها در این ویجت پایه پیچیده یا **wrap** شود. از طرفی نویگیشن یا روتینگ و نمایش پیام ها را این ویجت پایه انجام می دهد. با چک کردن نوع رسانه ای که اپ در آن اجرا می شود، به صورت عمومی مسئله سایز بندی یا روتینگ انجام می شود.

ماژول **data**

در پوشه یا ماژول دیتا تمام آنچه مربوط به دیتا در برنامه می شود قرار دارد و هر فیچر پوشه مربوط به خود را در زیر ماژول ها خواهد داشت.

ماژول **local**

در این پوشه ابزار کار با دیتایی که به صورت لوکال در برنامه ذخیره می شود، قرار داده شده است. دیتا ها به صورت کلید و مقدار یا **key/value** ذخیره و بازیابی می شوند. شاید عادی ترین مثال آن بحث توکن دسترسی و توکن بازیابی باشد. از دیتابیس محلی استفاده نشده است چون نیاز به آن نبوده است اما اگر نیاز به دیتابیس محلی باشد باید کلاس های رابط مربوط به آن را در این ماژول پیاده سازی کرد تا در سطح برنامه به صورت عمومی استفاده شود. در پوشه **di** این ماژول نیز تزریق و استنگی آن را می بینیم.

ماژول **model**

در این فolder بر اساس فیچر یا اپ هایی که داخل سوپر اپ پیاده سازی می شوند. مدل های آن برای ارتباط رست یا لوکال در این ماژول قرار می گیرد.

ماژول remote

ارتباط با بک اند و استفاده از REST API از طریق این ماژول انجام می شود.

ماژول repository & domain

لایه ریپازیتوری و دامین به صورت تنگ استفاده می شوند و براساس هر فیچر یا اپ باید کلاس های مربوطه ایجاد شوند و نمونه آن را در کد می بینید. لایه ریپازیتوری استفاده کننده از سه لایه گفته شده در بالا هست و پیاده کننده ریپازیتوری های ایجاد شده در ماژول دامین است. در ساب ماژول use case منطق برنامه بر اساس فیچر پیاده سازی می شود و در ساب ماژول util قالب ریسپانش برنامه را می بینیم تا بتواند استیت های مختلف را هندل کند و بر اساس آن رابط کاربری برنامه مدیریت می گردد.

ماژول features

در اینجا رابط کاربری برنامه پیاده سازی می شود و شما با استی فیچر یا ماژول پروژه خود را براساس معماری MVVM پیاده سازی کنید و از ویجت پایه و ویو مدل پایه برای این منظور استفاده کنید و یوزکیس هایی که در لایه دامین پیاده سازی کرده اید که شامل منطق برنامه شما می شود، در اینجا فراخوانی کنید.

ماژول navigation

در اینجا انواع نویگیشن یا روتینگ را می بینیم که با To به صورت مستقیم مسیر بعدی باز می شود، با نوع Back به مسیر قبل باز می گردیم و با ClearAndNavigate مسیر جدید جایگزین مسیر قبلی در استک می شود.

فایل app_bindings

در این کلاس وابستگی هایی که در ماثول های مختلف ایجاد شده از ساب ماثول های دیتا نظری لوکال، ریموت و ریپازیتوری و فیچر هایی که پیاده سازی شده اند در اینجا قرار می گیرند تا در طول برنامه مورد استفاده قرار گیرند.

فایل app_router

در این فایل روتینگ برنامه شناسانده می شود و مسیر اولیه ای که برنامه بر روی آن شروع می شود مشخص می شود.

فایل main

همانطور که می دانید برنامه از این کلاس آغاز می شود و کانفیگ های اولیه در این کلاس انجام می شود. همچنان در توضیحات قبلی در مورد روتینگ برنامه گفتیم از آنجا که از کتابخانه GoRouter برای مسیریابی استفاده می کنیم، برای استفاده از کتابخانه GetX برای تزریق وابستگی ها باید آن را به صورت دستی مقدار دهی و مدیریت کنیم. سپس تنظیمات مربوط به تم و رنگ بندی برنامه و ترنزیشن بین صفحات و روتینگ را مقدار دهی می کنیم.