FINDER- MATCH AND CHAT

I. INTRODUCTION

- FINDER - MEET NEW PEOPLE, EXPLORE LOVE: is an app designed to help people get to know and connect emotionally with each other through personal information based on the popular Finder platform in social media. It is a groundbreaking dating app that allows you to meet unique and interesting individuals around you. With a user-friendly interface similar to Finder, this app offers an enjoyable and intimate experience, helping you find that special someone for your life.
- Contact support: hongduyle.it@gmail.com

**** How to run this source (Finder - Match and Chat) ****

Step 1. Open source code folder in Android Studio.

Step 2. Run the following command in Android Studio's terminal:
- flutter pub get
- flutter gen-l10n

Step 3. When source is already running in mobile device, you must configure your own firebase console to sign up / login app and use all app's functions
-> Take a look at the full documentation below.

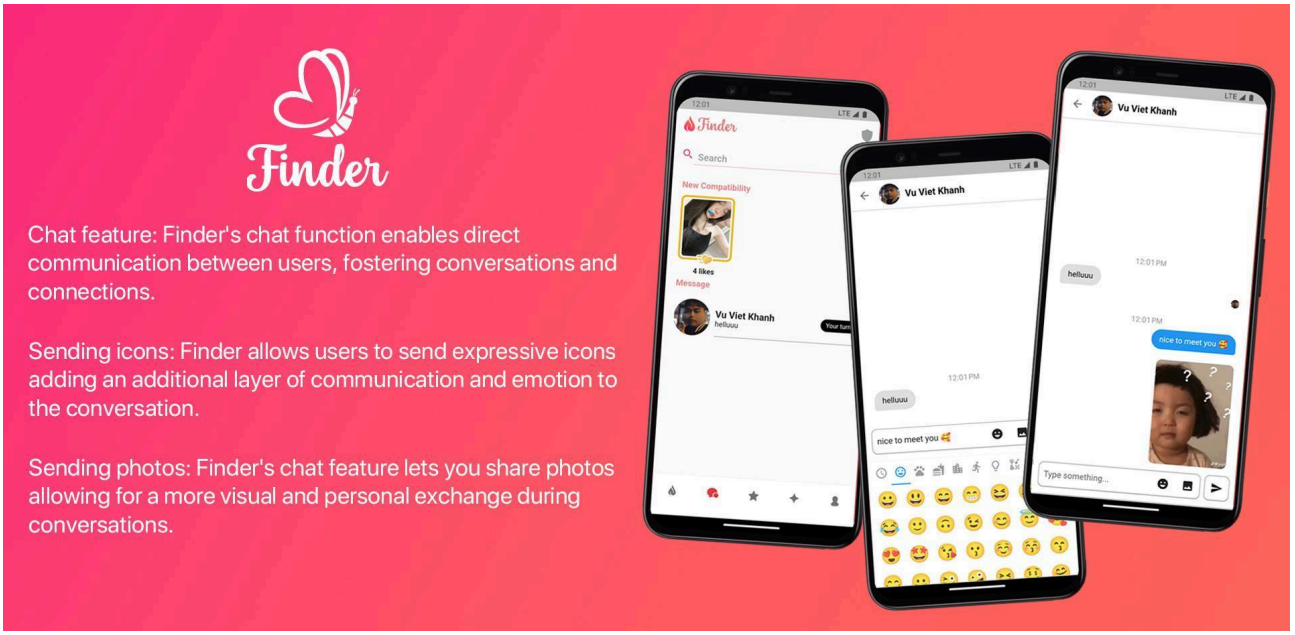
II. BASIC DESCRIPTION

1. MEET AND CONNECT

- With Finder you can browse through thousands of user profiles and choose the people you want to connect with. The first glances, shared interests, and nearby locations provide opportunities for us to get closer to each other. When two people like each other, a new chance unfolds!

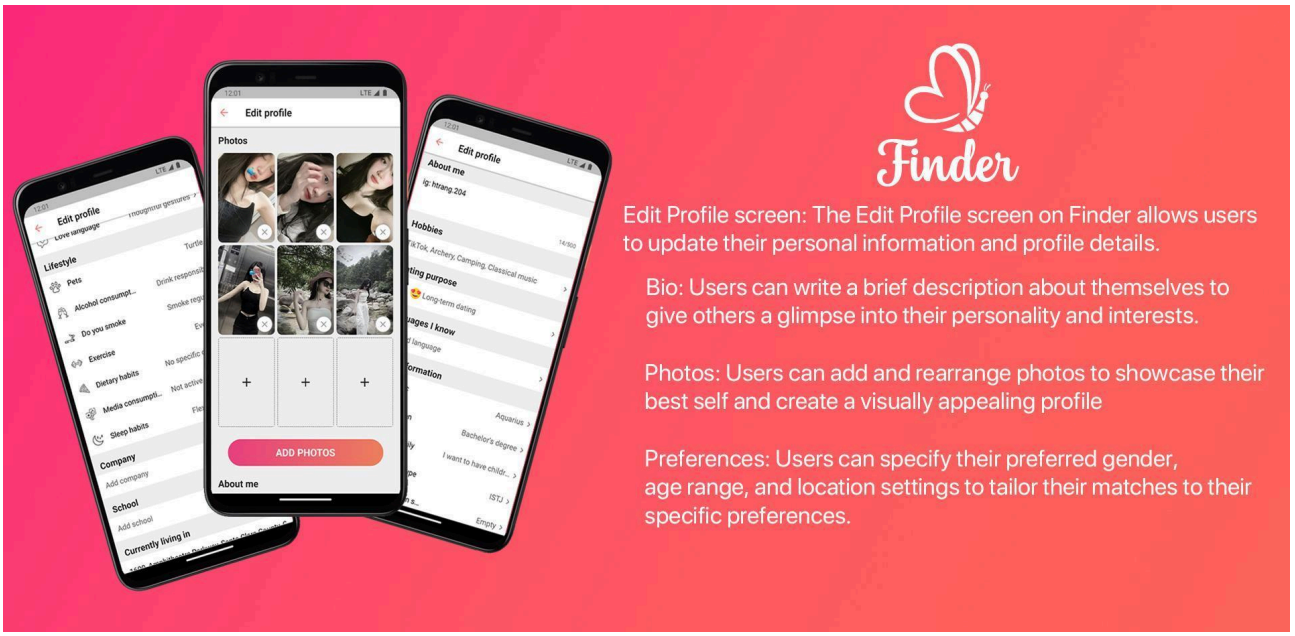
2. MESSAGING

- You and your match can chat right within the app through the integrated messaging feature. Get to know each other better and create meaningful connections.



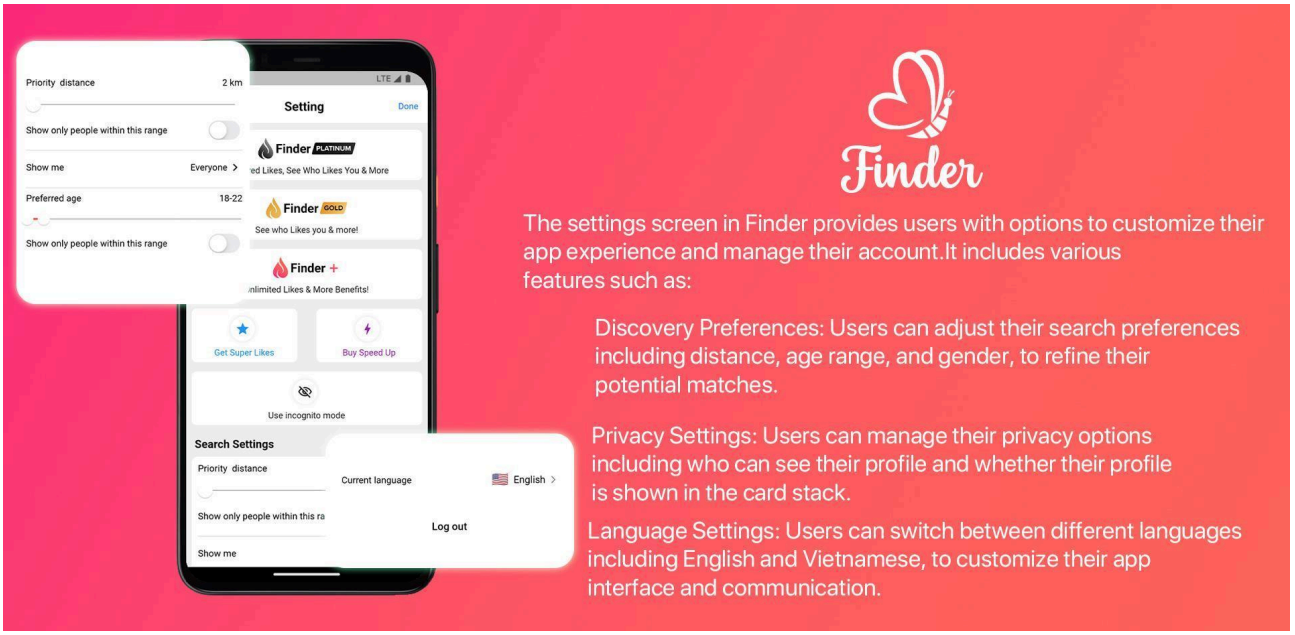
3. PHOTOS AND PERSONAL INFORMATION

- Present yourself uniquely through photos and personal information. Introduce yourself and attract attention from people with similar interests.



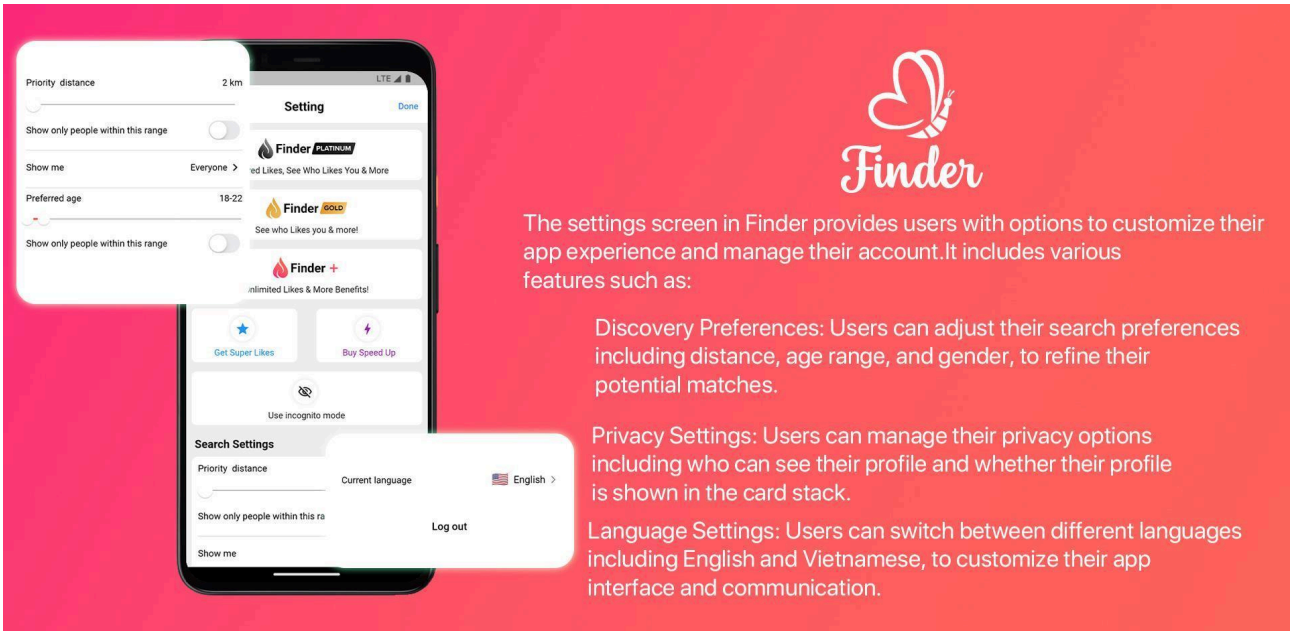
4. DISCOVERY SETTINGS

- Customize your search criteria to find people who match what you're looking for. Easily adjust age, location, interests, and much more.



5. HIGHLIGHTS PROFILE

- Make your profile stand out with unique features and questions. Impress everyone you meet with your creativity and personality.



III. FUNCTIONS, TOOLS, AND TECHNOLOGY

1. FUNCTIONS

1.1, SIGN UP, LOGIN

- Users can create a new account and login into the app using username and password or OTP for social media account integration (Google, Facebook, etc.)

1.2, SEARCH AND VIEW USERS:

- Users can browse through a list of other users and view their profile information, including pictures, name, age, etc.

1.3, CONNECTING VÀ MATCHING:

- Users can connect with people they are interested in by swiping left or right, depending on their interest in that person. If two users swipe right on each other, they will be connected and form a “Match”.

1.4, CHATTING:

- After having a “Match”, users can start chatting with each other through the in-app messaging system

1.5, INTEGRATED LOCATION SERVICES:

- The app can utilize integrated location services to locate users in nearby areas and suggest suitable matches.

1.6, CUSTOMIZE PROFILES:

- Users can customize their profiles by adding photos, self-descriptions, age, gender, interests, and more.

1.7, NOTIFICATIONS AND SETTINGS:

- The app can send notifications to users about activities related to their accounts. Users can customize app settings such as notifications, privacy, language, etc.

1.8, ACCOUNT MANAGEMENT AND LOGOUT:

- Users can manage their accounts, including logging out of the app when necessary.

2. TOOLS AND TECHNOLOGY

1.1. Framework: Flutter
1.2. Technologies applied in the app: Firebase, Navigator 2.0, GoRouter, Bloc, Provider, etc.
1.3. State Management Architecture: MVVM, Clean Architecture .

IV. CODE DESCRIPTION

1 . INTEGRATE ADMOB ADVERTISING (TEST VERSION)

- Tích hợp quảng cáo bản thử (test) từ AdMob vào ứng dụng Finder giúp bạn kiểm tra việc hiển thị và hoạt động của quảng cáo mà không gây ảnh hưởng đến người dùng thật sự. Dưới đây là mô tả sơ qua về việc tích hợp AdMob quảng cáo bản thử trong ứng dụng.

- Installation instructions can be found ***here.***

- Description of code in the project:
    - First, we have a Provider to manage AdMob called *‘AdMobProvider’* with initialization functions for banner ads and ad IDs (Here we use test IDs as an example).

```dart
class AdMobProvider with ChangeNotifier {
  BannerAd? _bannerAd;
  bool _isLoaded = false;

  BannerAd? get bannerAd => _bannerAd;
  bool get isLoaded => _isLoaded;

  Future<void> loadBannerAd(BuildContext context) async {
    final AnchoredAdaptiveBannerAdSize? size =
    await AdSize.getCurrentOrientationAnchoredAdaptiveBannerAdSize(
        MediaQuery.of(context).size.width.truncate());

    if (size == null) {
      print('Unable to get height of anchored banner.');
      return;
    }

    _bannerAd = BannerAd(
      // TODO: replace these test ad units with your own ad unit.
      adUnitId: Platform.isAndroid
          ? 'ca-app-pub-3940256099942544/6300978111'
          : 'ca-app-pub-3940256099942544/2934735716',
      size: size,
      request: AdRequest(),
      listener: BannerAdListener(
        onAdLoaded: (Ad ad) {
          print('$ad loaded: ${ad.responseInfo}');
          _isLoaded = true;
          notifyListeners();
        },
        onAdFailedToLoad: (Ad ad, LoadAdError error) {
          print('Anchored adaptive banner failedToLoad: $error');
          ad.dispose();
        },
      ),
    );

    await _bannerAd!.load();
  }

  @override
  void dispose() {
    _bannerAd?.dispose();
    super.dispose();
  }
}
```

- When you want to use real ads in the application, you need to register with Google AdMob and replace the IDs in the method below to generate revenue from ads:

```
adUnitId: Platform.isAndroid
          ? 'ca-app-pub-3940256099942544/6300978111'
          : 'ca-app-pub-3940256099942544/2934735716',
```

- To display the banner ad, we can call the Provider as follows (Note that you need to declare *AdMobProvider* in the main.dart function):

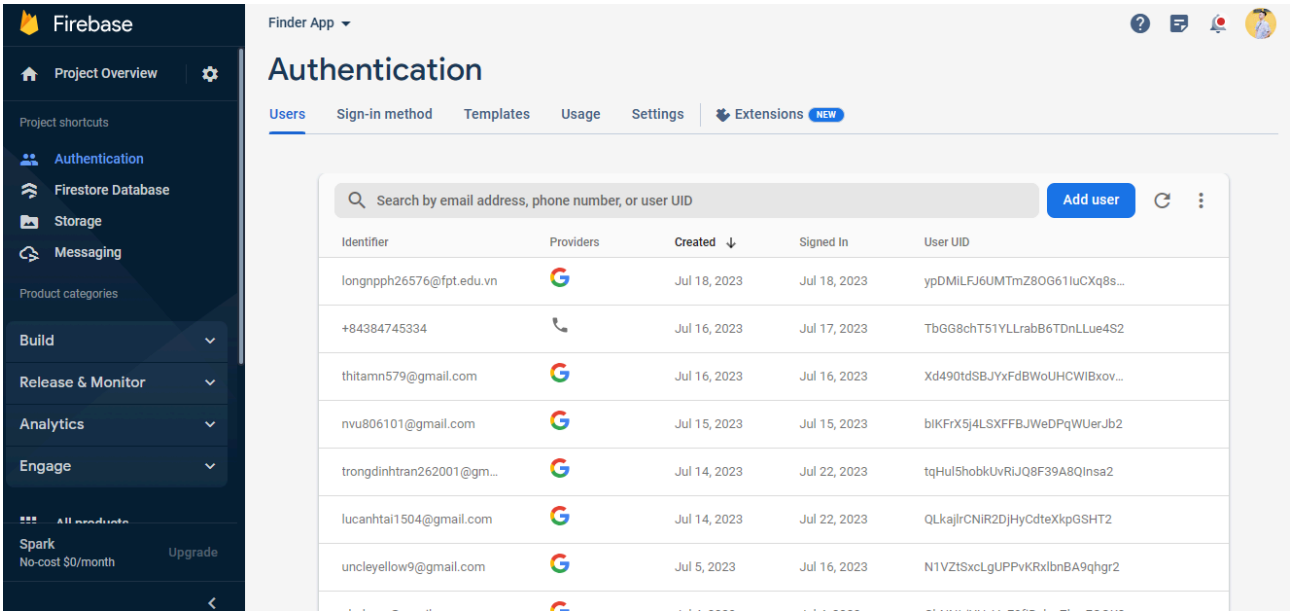
```
@override
Widget build(BuildContext context) {
  final appLocal = AppLocalizations.of(context);
  final adProvider = Provider.of<AdMobProvider>(context);

  return Scaffold(
    body: isLoading
        ? Center(...)  // Center
        : Container(
      decoration: BoxDecoration(...),  // BoxDecoration
      child: Stack(
        children:[
          Positioned(...),  // Positioned
          Column(...),  // Column
          (adProvider.isLoaded && adProvider.bannerAd != null) ?
          Align(
            alignment: Alignment.bottomCenter,
            child: SizedBox(
              width: adProvider.bannerAd?.size.width.toDouble(),
              height: adProvider.bannerAd?.size.height.toDouble(),
              child: AdWidget(ad: adProvider.bannerAd!),
            ),  // SizedBox
          ) : SizedBox.shrink(),  // Align
        ]
      ),  // Stack
    ),  // Container
  );  // Scaffold
}
```

2. FIREBASE AND DESCRIPTION

2.1, Firebase connection in Flutter:See detailed instructions on how to connect *here* , *Link Firebase* of Finder App

2.2, Authentication

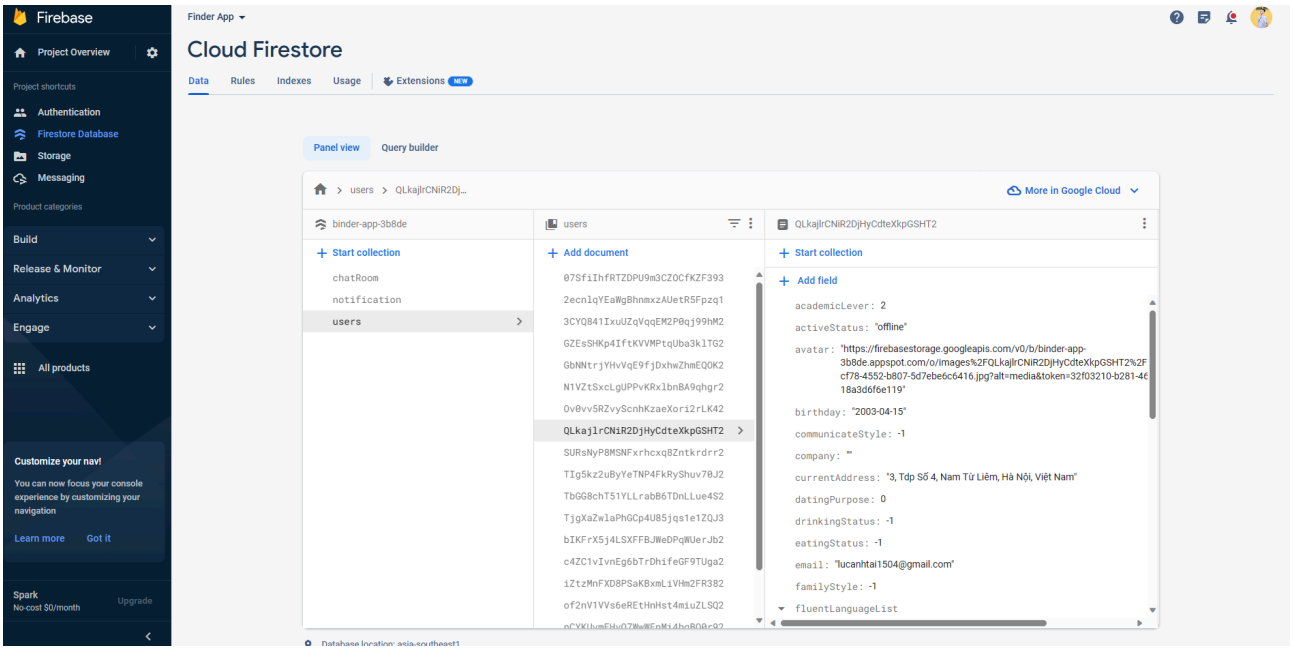
- In the Finder app, we utilize two authentication methods to allow users to access and enjoy the app's features: Registration with a phone number and Login with a Google Firebase account.
- Registration with a phone number: Users can register for an account in the Finder app using their phone number. When they press the "Register with a phone number" button, users will be prompted to enter their mobile phone number. After entering the phone number, they will receive a verification code via SMS to verify their phone number. Users will input the verification code into the app to complete the registration process.
- Login with a Google Firebase account: In addition to registering with a phone number, the Finder app also supports logging with a Google Firebase account. Users can choose the "Login with Google" option to sign in to the app.If they have previously logged into Google on their device, the app will prompt them to authenticate the account once again to access the Finder app. After successful authentication, users will be directed to the app's main screen and start enjoying its functionalities.



2.3, Cloud Firestore

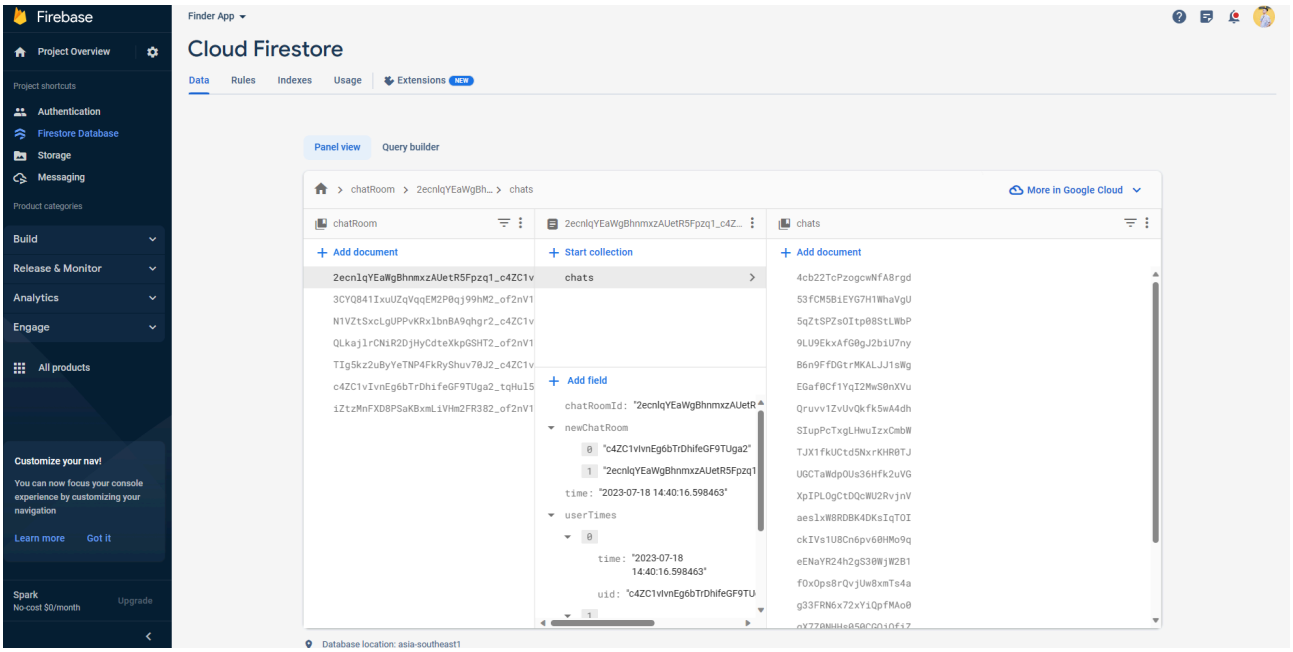
2.3.1, User:

- The user field contains information about users in the Finder app. Each record in the "users" collection represents a specific user with basic information fields..



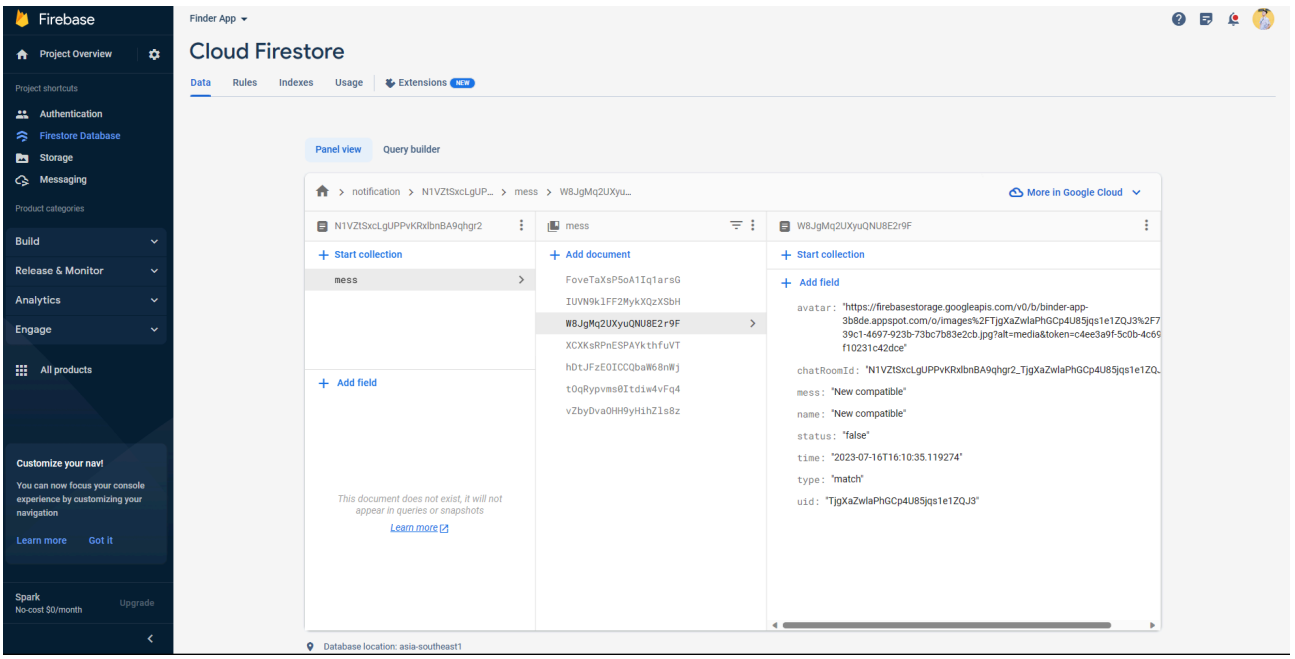
2.3.2, `ChatRoom` (Message Management)

- The ChatRoom field is used to store messages between two users who have formed a “Match”. Each record in the “chatRoom” collection represents a specific conversation between two users.



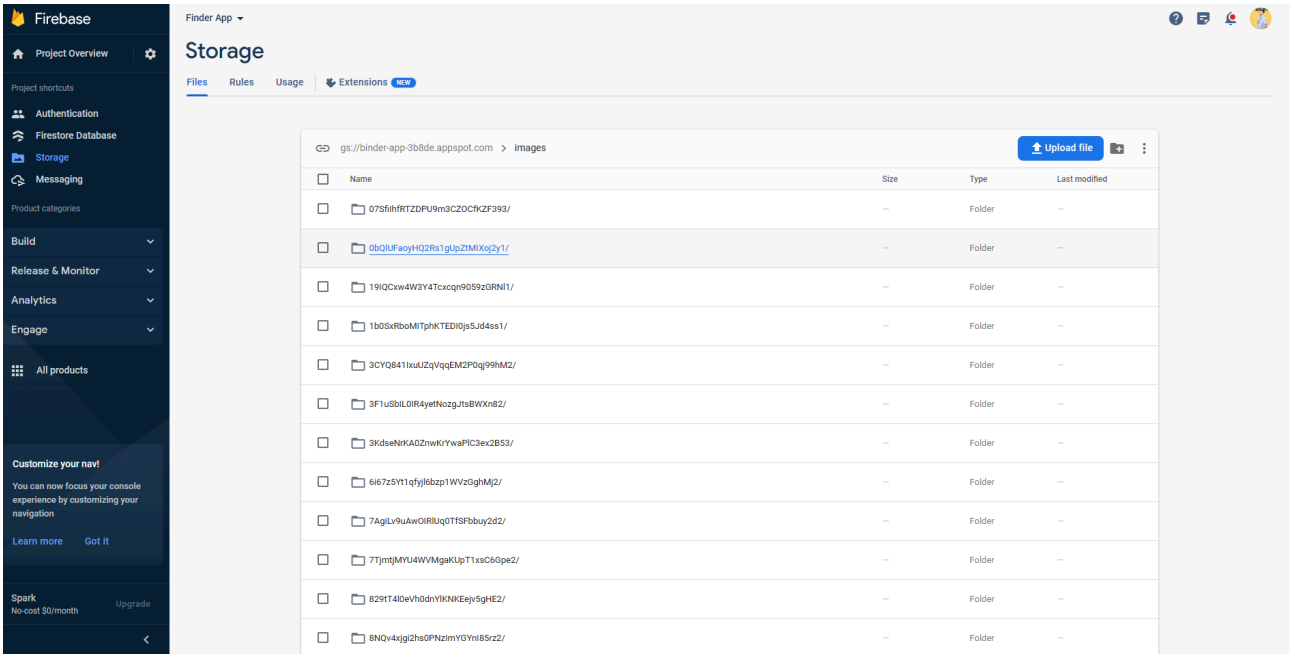
2.3.3, `Notification`(Thông báo đẩy):

- The Notification field is used to store notifications and updates for users in the Finder app. Each record in the “notification” collection represents a specific notification.



2.4, Storage

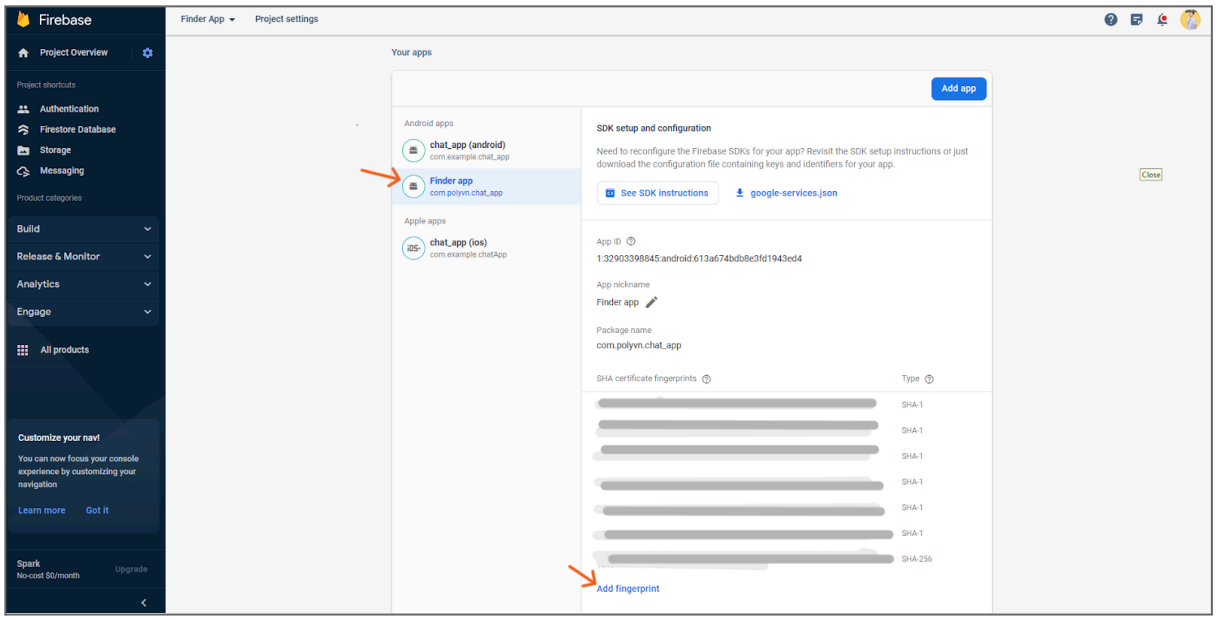
- When users create an account or update their profiles (e.g: adding a profile picture), the images are uploaded from the user's device to Firebase Cloud Storage. Firebase will generate a unique path for the uploaded image, such as "gs://binder-app-3b8de.appspot.com/images/07SfiIhfRTZDPU9m3CZOCfKZF393". This path is stored in the Cloud Firestore database in the "images" field of the user's account.



2.5,**Note**: when running the application in Debug mode, it is necessary to assign the SHA-1 code of the personal machine to Firebase as follows:

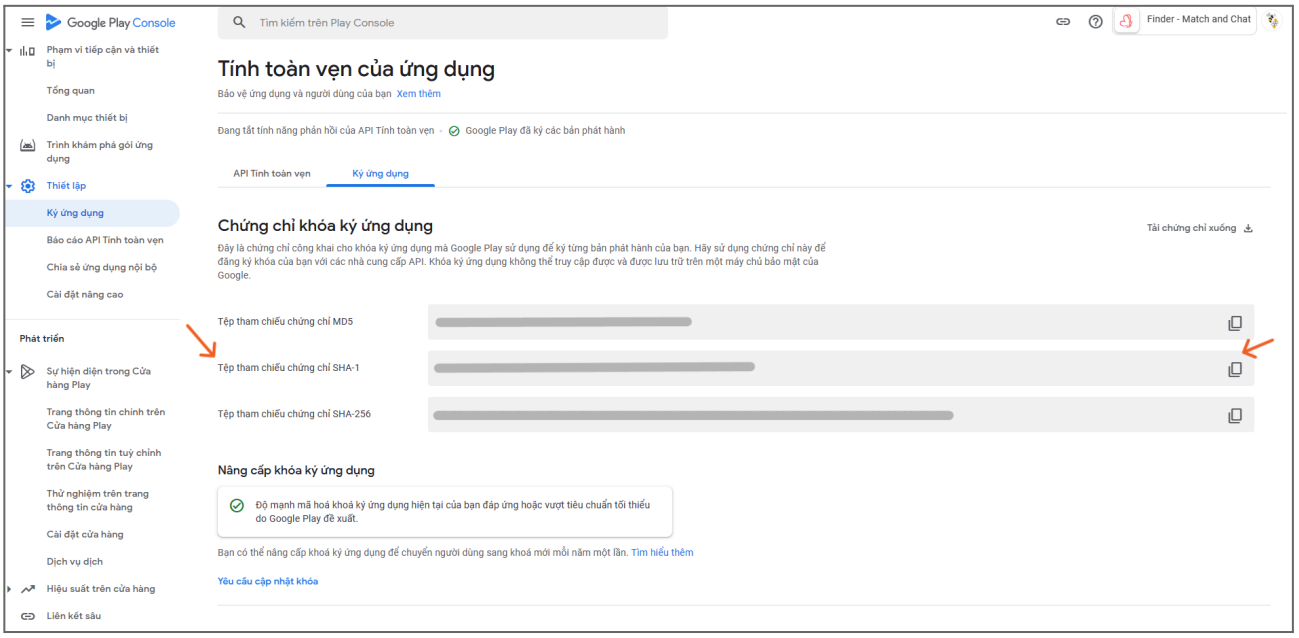
- How to get the SHA-1 code in your Flutter project ***HEAR***

- After we have the SHA-1 code, we go to Firebase according to the link assigned in 2.1 and add it as follows:

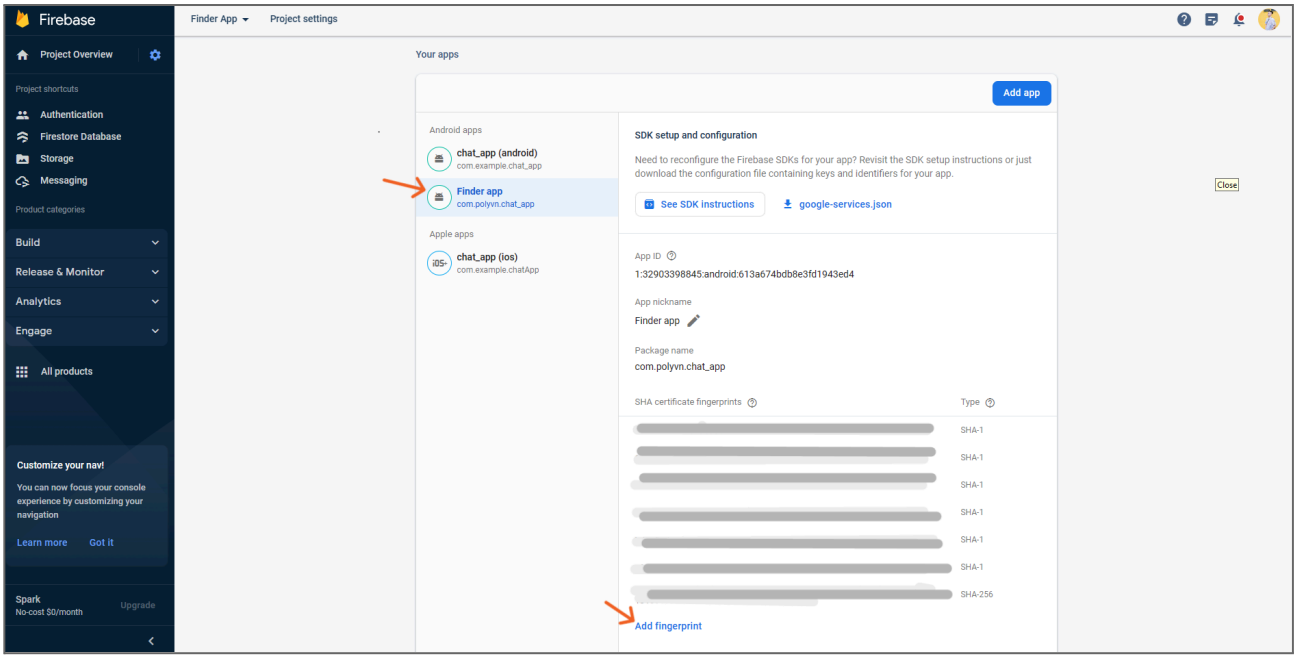


2.6, ***Note:*** When publishing the app on Google Store, you need to retrieve the SHA-1 code of your project from the Google Play Console and input it into the **`SDK setup and configuration`** section on Firebase

- To get the SHA-1 code from Google Play Console:



- Add code into Firebase:



3. SCREENSHOT AND DETAILED CODE OF EACH SCREEN (*SOURCE CODE*)

3.1, REGISTRATION AND LOGIN

3.1.1, Registration and Login with Google

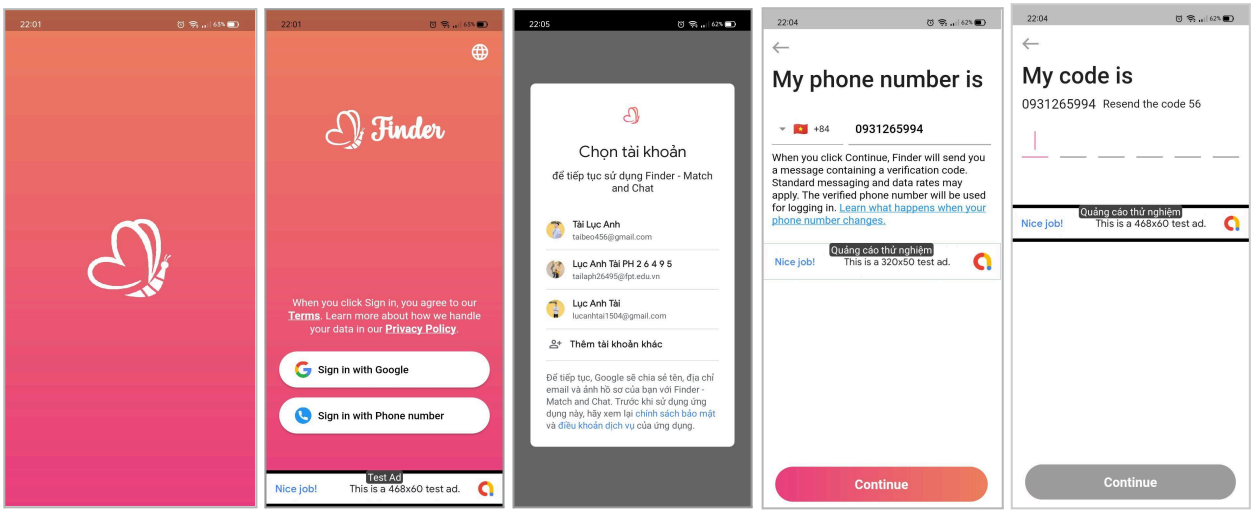
- This feature allows users to register and login to the Finder app using their Google account.
  a) Registration: When users access the app for the first time, the Finder app requests their permission to access their Google account information (e.g: name, profile picture). After successful authentication, the app retrieves the user's Google account information and creates a new account in the Firebase database. The user is then redirected to the app's home screen with the registered account.
  b) Login: Users who have registered with their Google accounts can log in. If they are already logged into Google on their device, the Finder app automatically authenticates the user's account and logs them in. If not already logged in, the app prompts the user to authenticate their Google account again. After successful authentication, the user is redirected to the app's home screen with the logged-in account.

3.1.1, Registration and Login with Phone Number

- This feature allows users to register and log in to the Finder app using their mobile phone numbers..
  c) Registration: When users access the app for the first time, the Finder app sends a verification code via SMS to the provided phone number. Users enter the verification code from the message to complete the account registration process. After successfully entering the verification code, the user's account is created and stored in the Firebase database. The user is then redirected to the app's home screen with the registered account.
  d) Login: Users who have registered with their phone numbers can log in. The user enters the registered mobile phone number into the app. The Finder app

sends a verification code via SMS to the provided phone number. Users enter the verification code from the message to log in to their account. After successfully entering the verification code, the user is redirected to the app's home screen with the logged-in account.

- This is the screenshots:



- Related Code Sections (Referenced in each heading's link):

a) Registration and Login Screen: This screen displays functions to start using the App, including the following sections:

- WithGoogle:

+, Choose to register and use the app with Google

+, Provider manages usage with Google

- With Phone number

+, Phone number registration screen

```dart
class LoginWithPhoneNumber extends StatelessWidget {

  @override
  Widget build(BuildContext context) {
    final appLocal = AppLocalizations.of(context);
    final adProvider = Provider.of<AdMobProvider>(context,listen: false);
    adProvider.loadBannerAd(context);

    return Consumer<LoginPhoneProvider>(builder: (context, myProvider, _) {
      return Scaffold(
          extendBody: true,
          backgroundColor: Colors.white,
          appBar: AppBar(
            elevation: 0,
            backgroundColor: Colors.white,
            leading: IconButton(
              icon: const Icon(
                Icons.west,
                color: Colors.grey,
                size: 30,
              ),
              onPressed: () {
                context.pop();
              },
            ),
          ),
          bottomNavigationBar: BottomAppBar(
            elevation: 0,
            color: Colors.transparent,
            child: ButtonSubmitPageView(text:appLocal.textNextButton,
                color: !myProvider.isTextFieldEmpty ? Colors.transparent : Colors.grey,
                onPressed: () async{
                  !myProvider.isTextFieldEmpty ?  await myProvider.onSubmitPhone(context) : null;
                }),
          ),
          body: SingleChildScrollView(
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
              Container(
                padding: EdgeInsets.symmetric(horizontal: 15),
                  child: Text(appLocal.titleEnterPhoneNumber,
                    style: TextStyle(
                        fontWeight: FontWeight.w500,
                        fontSize: 35
                    ),),
                ),
                const SizedBox(
                  height: 30,
                ),
                Padding(
                  padding: EdgeInsets.symmetric(horizontal: 15),
                  child: Row(
                    children: [
                    Expanded(
                        flex: 2,
                        child: IntlPhoneField(
                          decoration: const InputDecoration(
                            counterText: '',
                          ),
                          initialCountryCode: 'VN',
                          onCountryChanged: (phone) {
                            myProvider.selectedCountry('+${phone.fullCountryCode!.toString()}');
                          },
                        )
                    ),
                    const SizedBox(width: 8,),
                    Expanded(
                      flex: 4,
                      child: TextField(
                        controller: context
                            .watch<LoginPhoneProvider>()
                            .textEditingController,
                        keyboardType: TextInputType.number,
                        cursorColor: Color.fromRGBO(234, 64, 128, 100,),
                        style: TextStyle(color: Colors.black, fontWeight: FontWeight.w500,fontSize: 18),
                        decoration: InputDecoration(
                            contentPadding: EdgeInsets.symmetric(horizontal: 10) ,
                            hintText: appLocal.hintTextPhone,
                            hintStyle: TextStyle(color: Colors.grey,fontSize: 14),
                            enabledBorder: UnderlineInputBorder(
                              borderSide: BorderSide(color: Colors.grey,width: 1.5),
                            ),
                            focusedBorder: UnderlineInputBorder(
                              borderSide: BorderSide(color:Color.fromRGBO(234, 64, 128, 100,),width: 2),
                            ),
                        ),
                        onChanged: (value) {myProvider.onTextFieldChanged();},
                      ),
                    ),

                    ],
                  ),
                ),
```

The code snippet above represents the login or registration interface based on the phone number, with an input field for the phone number and a country selection. Users can enter their phone number and choose the country before pressing the "Continue" button to proceed with login or registration. The Consumer from the Provider package is used to listen to changes and rebuild related child widgets accordingly.

+, The Code Verification Screen:

```dart
@override
Widget build(BuildContext context) {
  final appLocal = AppLocalizations.of(context);
  final adProvider = Provider.of<AdMobProvider>(context,listen: false);
  adProvider.loadBannerAd(context);

  return Scaffold(
      extendBody: true,
      backgroundColor: Colors.white,
      appBar: AppBar(
        elevation: 0,
        backgroundColor: Colors.white,
        leading: IconButton(
          icon: const Icon(
            Icons.west,
            color: Colors.grey,
            size: 30,
          ),
          onPressed: () {
            context.pop();
          },
        ),
      ),
      bottomNavigationBar: BottomAppBar(
        elevation: 0,
        color: Colors.transparent,
        child: ButtonSubmitPageView(text:appLocal.textNextButton,
            color:  Provider.of<LoginPhoneProvider>(context, listen: false).smsCode.length == 6
                ? Colors.transparent : Colors.grey,
            onPressed: () async{
              await Provider.of<LoginPhoneProvider>(context, listen: false).verify(context);
            }),
      ),
      body: SingleChildScrollView(
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Padding(
              padding: const EdgeInsets.symmetric(horizontal: 15),
              child: Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                   Text(appLocal.titleEnterOTP,
                    style: TextStyle(
                        fontWeight: FontWeight.w500,
                        fontSize: 35
                    ),),
                  const SizedBox(
                    height: 10,
                  ),
                  Row(
                    children: [
                       Text('${Provider.of<LoginPhoneProvider>(context, listen: false)
                           .textEditingController.text}',
                        style: TextStyle(
                            fontWeight: FontWeight.w400,
                            fontSize: 20
                        ),),
                      const SizedBox(
                        width: 10,
                      ),
                      Center(
                          child: context
                              .watch<LoginPhoneProvider>()
                              .resend ? resend(context) : countDown(context)
                      ),
                    ],
                  ),
                  const SizedBox(
                    height: 20,
                  ),
                  PinCodeTextField(
                      appContext: context,
                      cursorColor: const Color.fromRGBO(234, 64, 128, 100,),
                      length: 6,
                      keyboardType: TextInputType.number,
                      inputFormatters: [FilteringTextInputFormatter.digitsOnly],
                      pinTheme: PinTheme(
                        borderWidth: 2,
                        shape: PinCodeFieldShape.underline,
                        borderRadius: BorderRadius.circular(10),
                        inactiveColor: Colors.grey,
                        selectedColor: const Color.fromRGBO(234, 64, 128, 100,),
                      ),
                      onChanged: (value) {
                        Provider.of<LoginPhoneProvider>(context, listen: false)
                            .inputCode(value);
                      }),
                ],
```

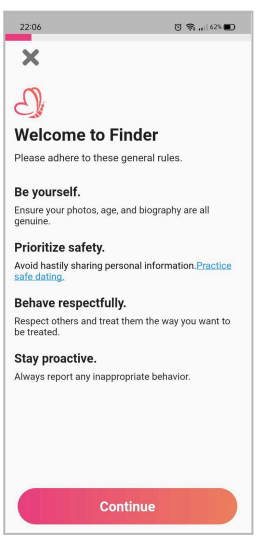
- The above code snippet is the OTP verification input interface for the login or registration process in the app. It allows users to enter the OTP code and verify it to complete the login or registration process.

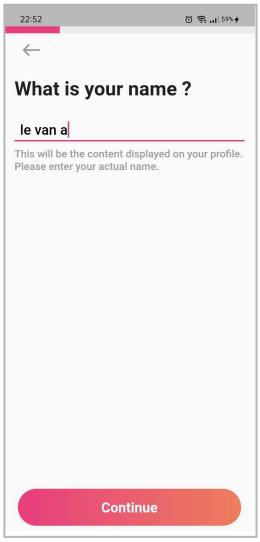
+, Provider manages usage with Phone Number

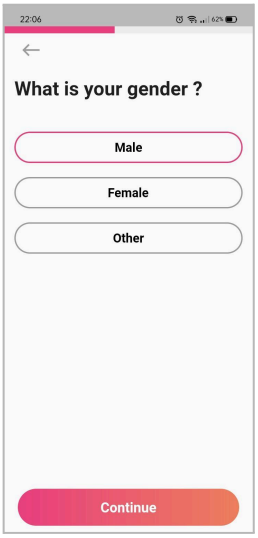
```dart
class LoginPhoneProvider extends ChangeNotifier {
  final FirebaseAuth auth = FirebaseAuth.instance;
  TextEditingController textEditingController = TextEditingController();
  bool isTextFieldEmpty = true;
  bool isErrorText = false;
  bool isErrorSms = false;
  String country = '+84';
  String codeVerify = '';
  String smsCode = '';
  bool resend = false;

  void onTextFieldChanged() {
    isTextFieldEmpty = textEditingController.text.isEmpty;
    notifyListeners();
  }

  void onTextFieldError() {
    isErrorText = !isErrorText;
    notifyListeners();
  }

  void selectedCountry(String newValue) {
    country = newValue;
    notifyListeners();
  }

  void inputCode(String newValue) {
    smsCode = newValue;
    notifyListeners();
  }


  void Resend(bool value) {
    resend = value;
    notifyListeners();
  }
  void smsError(bool value){
    isErrorSms = value;
    notifyListeners();
  }

  Future<void> onSubmitPhone(BuildContext context) async {
    if(!HelpersUserAndValidators.isValidPhoneNumber(textEditingController.text)){
      onTextFieldError();
      isTextFieldEmpty = false;
      textEditingController.clear();
    }else {
      await FirebaseAuth.instance.verifyPhoneNumber(
          phoneNumber: '$country ${textEditingController.text}',
          verificationCompleted: (PhoneAuthCredential credential) {},
          verificationFailed: (FirebaseAuthException e) {
            print('${e.message}');
          },
          codeSent: (String verificationId, int? resendToken) {
            codeVerify = verificationId;
            context.go('/login-home-screen/loginPhone/verify_otp');
          },
          codeAutoRetrievalTimeout: (String verificationId) {});
    }
  }

  Future<void> verify(BuildContext context) async {
    try {
      PhoneAuthCredential credential = PhoneAuthProvider.credential(
        verificationId: codeVerify,
        smsCode: smsCode,
      );
      UserCredential userCredential = await auth.signInWithCredential(credential);

      String? uid = userCredential.user?.uid;
      bool check = await DatabaseMethods().checkUserExists(uid);
      if(check == true){
        textEditingController.text = '';
        await HelpersFunctions.saveIdUserSharedPreference(uid!);
        context.go('/home');
      }else{
        textEditingController.text = '';
        context.go('/login-home-screen/confirm-screen');
      }
    } catch (e) {
      smsError(true);
      print('Lỗi xác minh số điện thoại: $e');
    }
  }
}
```

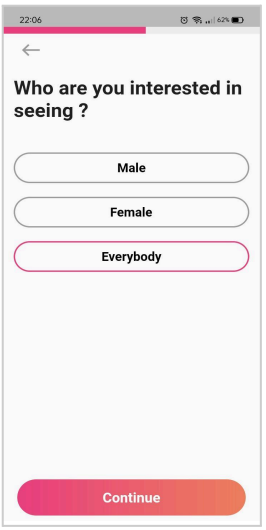
3.2, Confirmation and User Information Registration:

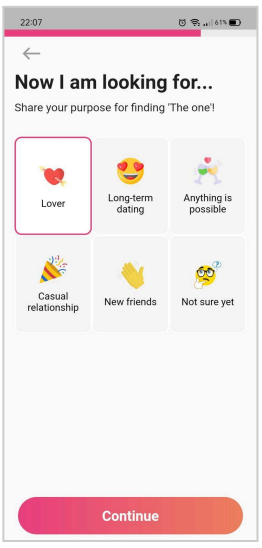
- This feature allows users to register basic information about themselves to be displayed when using the App. After completing the required steps, the user's database is created, and they are directed to the main screen of the app to start using it.
- This is the screenshots

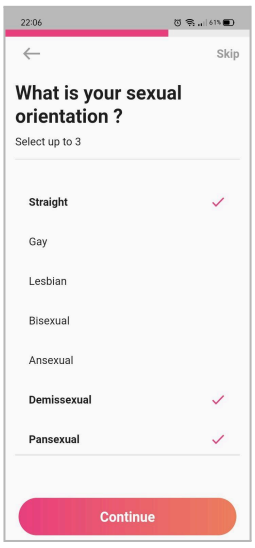


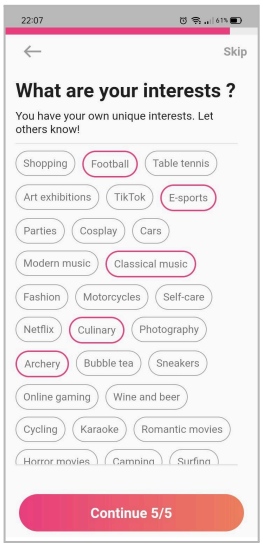


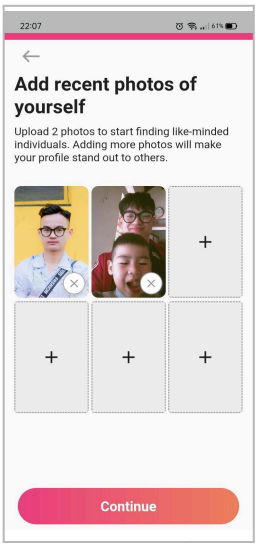












- Related code items.(In the link attached to each title) :

The user verification screen after completing registration will display various pages for entering basic information such as name, date of birth, gender, the person they want to meet, sexual orientation, interests, purposes, and uploaded photos…

```dart
class ConfirmProfile extends StatelessWidget {
  const ConfirmProfile({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    final pageProvider = Provider.of<PageDataConfirmProfileProvider>(context);
    return WillPopScope(
        onWillPop: () async {
          if (pageProvider.currentPageIndex > 0) {
            pageProvider.previousPage();
            return false;
          } else {
            pageProvider.showCustomDialog(context);
            return true;
          }
        },
        child: Scaffold(
          body: Container(
                padding: EdgeInsets.only(top: MediaQuery.of(context).padding.top),
                child: Column(
                  children: [
                    SizedBox(
                      height: 10,
                      child: LinearProgressIndicator(
                        valueColor: AlwaysStoppedAnimation<Color>(Color.fromRGBO(234, 64, 128, 1)),
                        backgroundColor: Colors.grey.shade200,
                        value: (pageProvider.currentPageIndex + 1) / 9,
                      ),
                    ),
                    Expanded(
                      child: IndexedStack(
                        index: pageProvider.currentPageIndex,
                        children: [
                          RulesPageSection(),
                          AddNamePageSection(),
                          AddBirthdayPageSection(),
                          AddGenderPageSection(),
                          AddRequestToShowPageSection(),
                          AddSexualOrientationListPageSection(),
                          AddDatingPurposePageSection(),
                          AddInterestsListPageSection(),
                          AddPhotoListPageSection(),
                        ],
                      ),
                    ),
                  ],
                ),
              ),
        ),
      );
    }
}
```

The "ConfirmProfile" screen utilizes an "IndexedStack" to sequentially display each widget for user information input on the screen, managed by the "currentPageIndex" variable of "PageDataConfirmProfileProvider."

The information entered into the input widgets is managed through the state provided by PageDataConfirmProfileProvider. The input data will be stored in variables within the provider. The input fields (TextFields) will be controlled using controllers created in the provider. Once all the information is entered into the widgets, including the widget for adding an image, when the user presses the confirmation button, the data will be saved through a function within the PageDataConfirmProfileProvider provider.

```dart
Future<void> confirmUser(BuildContext context) async {
    if(FirebaseApi.enablePermission){
      await save(context);
      context.go('/home');
    }else{
      context.goNamed('location-screen');
    }

  }
  Future<void> save(BuildContext context) async {
    isLoading = true;
    print('Số lượng ảnh: ${photosList.length}');
    final signUp = context.read<CallDataProvider>();
    signUp.confirmProfile(
        nameController.text,
        selectedGender,
        selectedRequestToShow,
        birthday,
        newInterestsList,
        newDatingPurpose!,
        photosList,
        newSexualOrientationList,
        ['21.07302', '105.7703283'])
        .whenComplete(() =>  isLoading = false);
  }
```

If the user grants permission to access their location, they will be redirected to the home screen; otherwise, they will be redirected to the LocationScreen to input their address.

To view the detailed code, please access the following links.

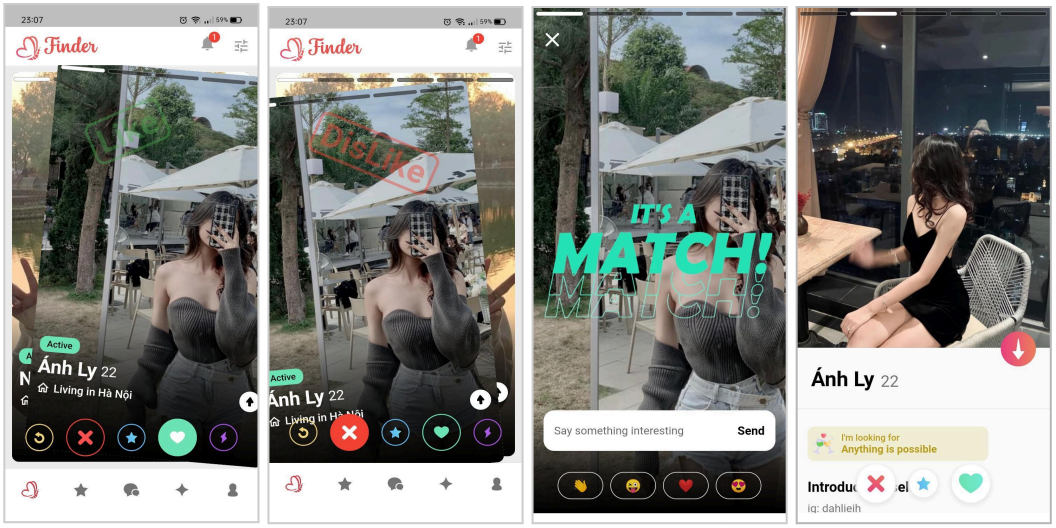
- , Main screen and confirm information pages

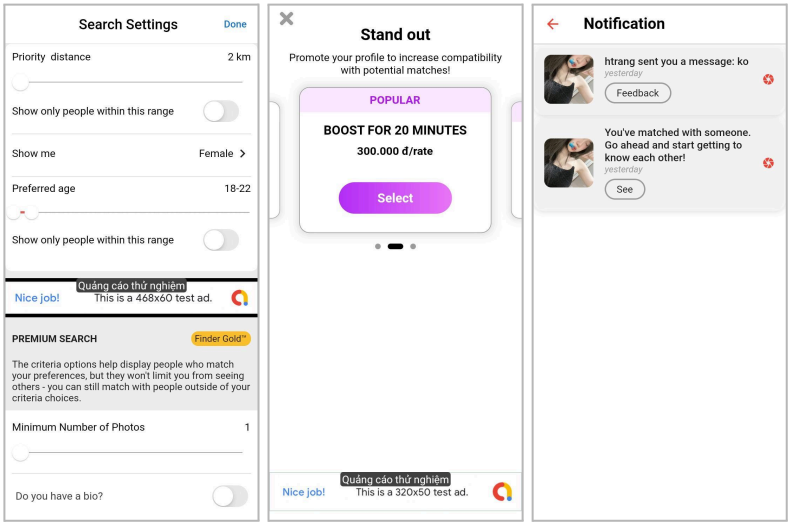
-, Constructor Widgets

-, Verification Management Provider

3.3, Main Screen of the Application

- This function includes a user interface featuring cards of other users, along with basic information such as their name, interests, and a list of display pictures. Users can swipe to send a "Match" to the other person, view detailed profiles of others, and highlight their own profile to increase compatibility and make friends. Additionally, the page also includes sections such as a notification screen to display updates on new matches, incoming messages, and app-related notifications. There is also a filtering screen to sort users' data based on location, age, and gender to cater to users' specific preferences.
- Screenshots





- Related code items.(In the link attached to each title) :
    a) The screen displays user cards for sending Match invitations, notifications when a Match is successful, viewing detailed user profiles, and highlighting.

- , [BinderPage main screen]

The main content of the BinderPage screen is a FutureBuilder, which renders the interface based on the value of the future obtained from the allUserBinder function of the BinderWatch provider. The parameters passed to the allUserBinder function are the filtering conditions, which are also received from the BinderWatch provider.

Specifically, the BinderPage screen will display data related to user binders (such as information or a list) based on the filtering conditions received from the BinderWatch provider. These filtering conditions will be processed on the filter settings screen.

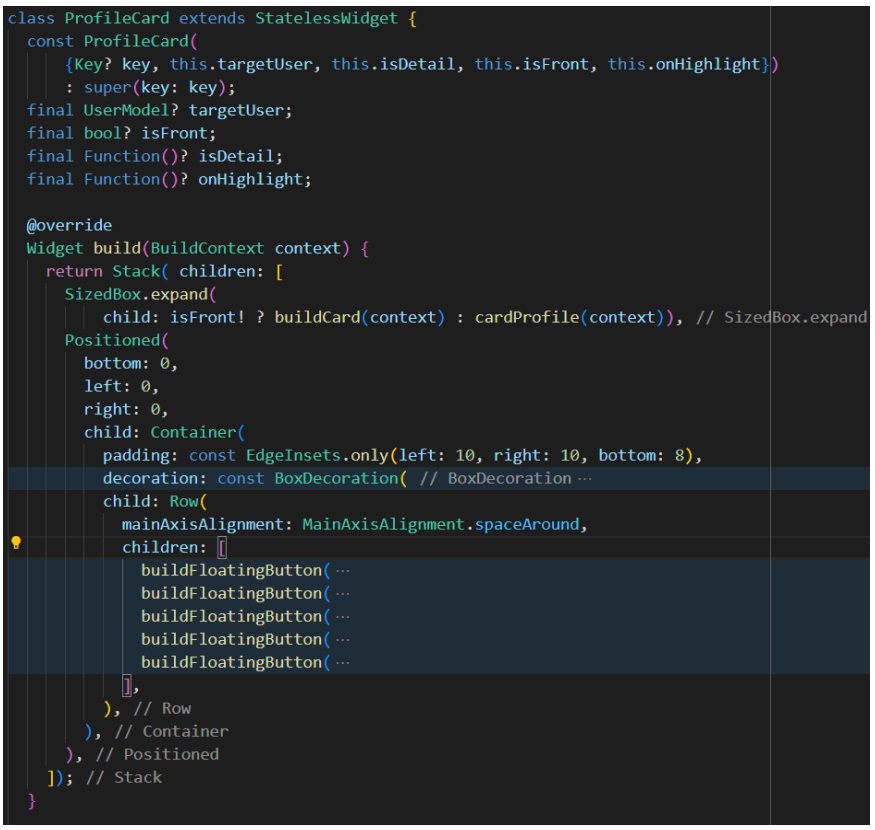
```
FutureBuilder(
      future: context
          .read<BinderWatch>()
          .allUserBinder(context, gender, age, isInDistanceRange, kilometres),
      builder: (context, snapshot) => snapshot.hasData
          ? Container(
              padding: const EdgeInsets.all(10),
              decoration: BoxDecoration(
                boxShadow: [
                  BoxShadow(
                    color: Colors.grey.shade200,
                    spreadRadius: 3,
                    blurRadius: 3,
                    offset: Offset(0, 3),
                  ),
                ],
              ),
              child: Stack(
                  alignment: Alignment.center,
                  children: context
                      .watch<BinderWatch>()
                      .listCard
                      .reversed
                      .map((e) => ProfileCard(
                            targetUser: e,
                            isDetail: () => context.goNamed(
                                'home-detail-others',
                                queryParameters: {
                                  'uid': e.uid.toString(),
                                }),
                            onHighlight: () => context.goNamed(
                                'home-highlight-page',
                                queryParameters: {
                                  'currentUserID': context
                                      .read<ProfileWatch>()
                                      .currentUser
                                      .uid
                                      .toString(),
                                  'targetUserID': e.uid.toString(),
                                }),
                            isFront:
                                context.watch<BinderWatch>().listCard.first ==
                                    e,
                          ))
                      .toList()),
            )
          : Center(
              child: LoadingAnimationWidget.dotsTriangle(
                color: const Color.fromRGBO(234, 64, 128, 1),
                size: 70,
              ),
            ),
    );
```

allUserBinder function of provider BinderWatch:

```dart
Future<List<UserModel>> allUserBinder(BuildContext context, int gender,
      List<double> age, bool isInDistanceRange, double kilometres) async {
    try {
      final uid =
      await HelpersFunctions().getUserIdUserSharedPreference() as String;
      final List<UserModel> users;
      if (isInDistanceRange) {
        users = await DatabaseMethods()
            .getUserHasFilterKm(uid, gender, [age.first, age.last], kilometres);
      } else {
        users = await DatabaseMethods()
            .getUserHasFilter(uid, gender, [age.first, age.last]);
      }
      _listCard = users;
      shuffleUsers(_listCard);
      await Provider.of<HighlightUserNotify>(context, listen: false)
          .sortUsers(_listCard);

      return _listCard;
    } catch (e) {
      throw Exception(e);
    }
  }
```

The function takes parameters as filter conditions and a query to retrieve a list of users who meet the filtering conditions through the getUserHasFilterKm or getUserHasFilter function from the DatabaseMethods class. After obtaining the list of users, it uses the shuffleUsers function to shuffle the positions of the users.

The obtained list of users will be displayed on the BinderPage screen as a stack of ProfileCard widgets. The ProfileCard widget displays user information in the form of a card. If it's the first card, it returns the buildCard function; otherwise, it returns the cardProfile function. Additionally, there are functional buttons created from the buildFloatingButton function:

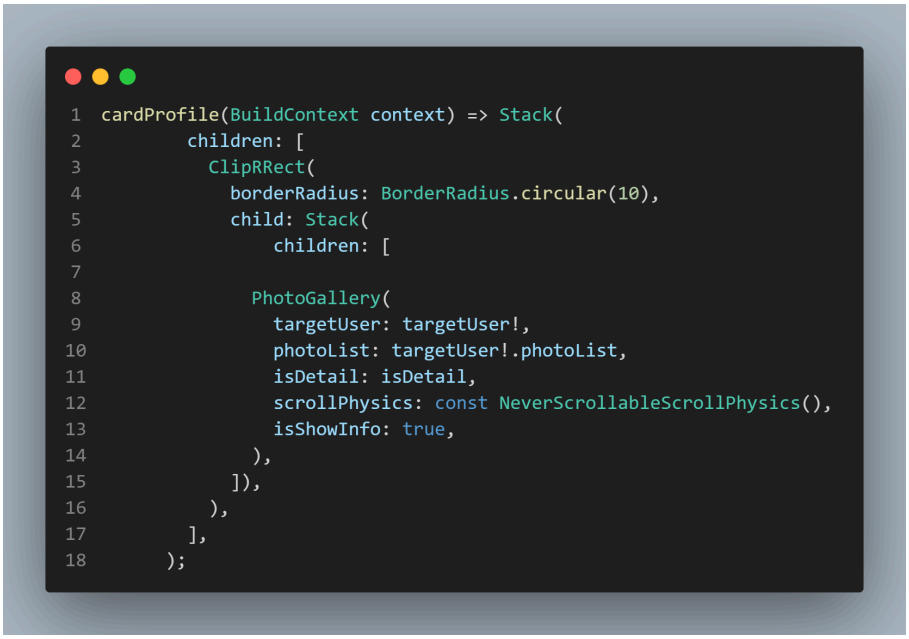


```dart
class ProfileCard extends StatelessWidget {
  const ProfileCard(
      {Key? key, this.targetUser, this.isDetail, this.isFront, this.onHighlight})
      : super(key: key);
  final UserModel? targetUser;
  final bool? isFront;
  final Function()? isDetail;
  final Function()? onHighlight;

  @override
  Widget build(BuildContext context) {
    return Stack( children: [
      SizedBox.expand(
          child: isFront! ? buildCard(context) : cardProfile(context)), // SizedBox.expand
      Positioned(
        bottom: 0,
        left: 0,
        right: 0,
        child: Container(
          padding: const EdgeInsets.only(left: 10, right: 10, bottom: 8),
          decoration: const BoxDecoration( // BoxDecoration ···
          child: Row(
            mainAxisAlignment: MainAxisAlignment.spaceAround,
            children: [
              buildFloatingButton(···
              buildFloatingButton(···
              buildFloatingButton(···
              buildFloatingButton(···
              buildFloatingButton(···
            ],
          ), // Row
        ), // Container
      ), // Positioned
    ]); // Stack
  }
```

The function buildCard contains a GestureDetector widget and a LayoutBuilder to handle swipe events when users swipe. When a user swipes, it will call functions from the BinderWatch provider to handle the events.

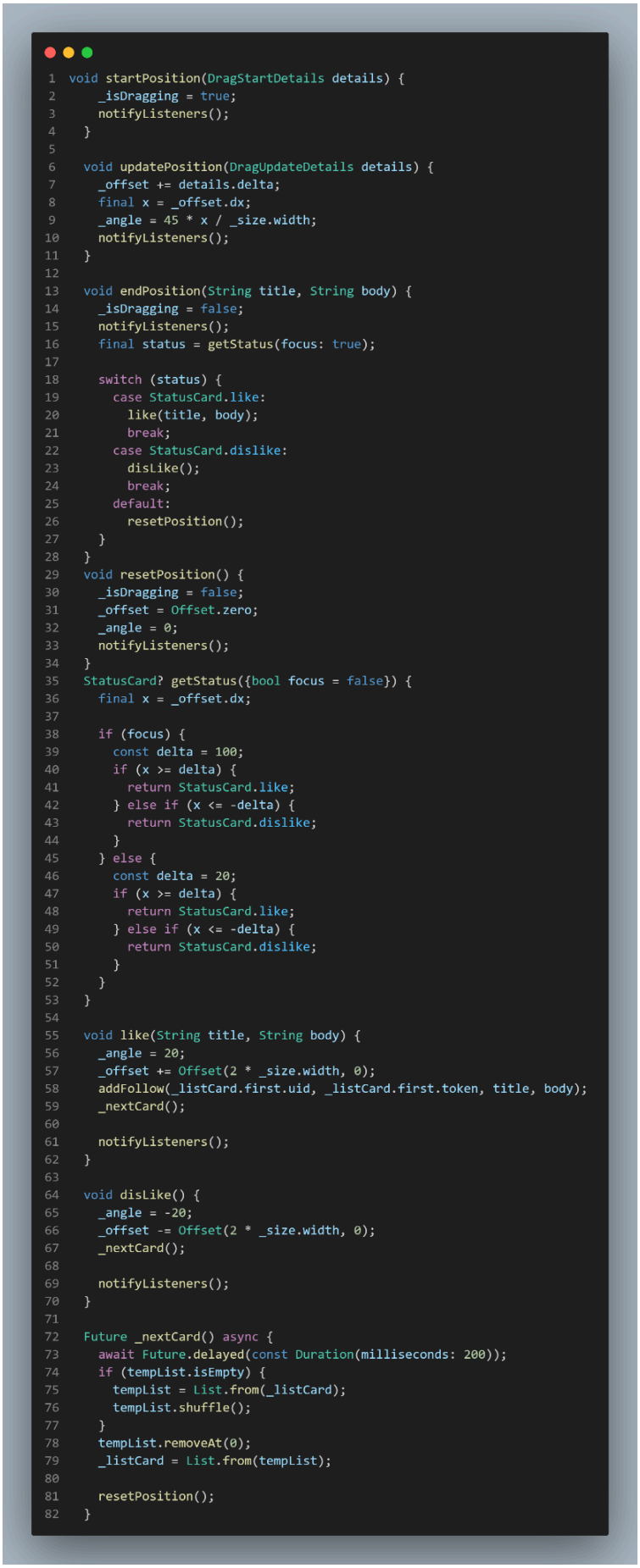
```dart
Widget buildCard(context) => GestureDetector(
        onPanStart: (details) {
          final provider = Provider.of<BinderWatch>(context, listen: false);
          provider.startPosition(details);
        },
        onPanUpdate: (details) {
          final provider = Provider.of<BinderWatch>(context, listen: false);
          provider.updatePosition(details);
        },
        onPanEnd: (details) {
          final provider = Provider.of<BinderWatch>(context, listen: false);
          provider.endPosition(AppLocalizations.of(context)
          .notificationScreenTitle2, AppLocalizations.of(context)
          .notificationScreenContent);
        },
        child: LayoutBuilder(builder: (context, constraints) {
          final provider = Provider.of<BinderWatch>(context);
          final position = provider.position;
          final milliseconds = provider.isDragging ? 0 : 400;

          final center = constraints.smallest.center(Offset.zero);
          final angle = provider.angle * pi / 180;
          final rotatedMatrix = Matrix4.identity()
            ..translate(center.dx, center.dy)
            ..rotateZ(angle)
            ..translate(-center.dx, -center.dy);

          return AnimatedContainer(
            curve: Curves.easeInOut,
            transform: rotatedMatrix..translate(position.dx, position.dy),
            duration: Duration(milliseconds: milliseconds),
            child: Stack(
              children: [
                cardProfile(context),
                buildStamps(context),
              ],
            ),
          );
        }),
      );
```

The function cardProfile displays the user information of the card.:



```
cardProfile(BuildContext context) => Stack(
        children: [
          ClipRRect(
            borderRadius: BorderRadius.circular(10),
            child: Stack(
                children: [

              PhotoGallery(
                targetUser: targetUser!,
                photoList: targetUser!.photoList,
                isDetail: isDetail,
                scrollPhysics: const NeverScrollableScrollPhysics(),
                isShowInfo: true,
              ),
            ]),
          ),
        ],
      );
```

The event handling functions in the provider will process based on the direction of the user's swipe. If the swipe is to the left, the getStatus function will return StatusCard.dislike, and if it's to the right, it will return StatusCard.like. After the swipe, it will trigger the endPosition function, which will check the status returned by the getStatus function. If it's a like, it will call the like function; if it's a dislike, it will call the dislike function.



```dart
void startPosition(DragStartDetails details) {
    _isDragging = true;
    notifyListeners();
  }

  void updatePosition(DragUpdateDetails details) {
    _offset += details.delta;
    final x = _offset.dx;
    _angle = 45 * x / _size.width;
    notifyListeners();
  }

  void endPosition(String title, String body) {
    _isDragging = false;
    notifyListeners();
    final status = getStatus(focus: true);

    switch (status) {
      case StatusCard.like:
        like(title, body);
        break;
      case StatusCard.dislike:
        disLike();
        break;
      default:
        resetPosition();
    }
  }
  void resetPosition() {
    _isDragging = false;
    _offset = Offset.zero;
    _angle = 0;
    notifyListeners();
  }
  StatusCard? getStatus({bool focus = false}) {
    final x = _offset.dx;

    if (focus) {
      const delta = 100;
      if (x >= delta) {
        return StatusCard.like;
      } else if (x <= -delta) {
        return StatusCard.dislike;
      }
    } else {
      const delta = 20;
      if (x >= delta) {
        return StatusCard.like;
      } else if (x <= -delta) {
        return StatusCard.dislike;
      }
    }
  }

  void like(String title, String body) {
    _angle = 20;
    _offset += Offset(2 * _size.width, 0);
    addFollow(_listCard.first.uid, _listCard.first.token, title, body);
    _nextCard();

    notifyListeners();
  }

  void disLike() {
    _angle = -20;
    _offset -= Offset(2 * _size.width, 0);
    _nextCard();

    notifyListeners();
  }

  Future _nextCard() async {
    await Future.delayed(const Duration(milliseconds: 200));
    if (tempList.isEmpty) {
      tempList = List.from(_listCard);
      tempList.shuffle();
    }
    tempList.removeAt(0);
    _listCard = List.from(tempList);

    resetPosition();
  }
```

-, Highlight Page Screen : Push your profile to the top of someone else's.

```dart
@override
  Widget build(BuildContext context) {
    final EdgeInsets padding = MediaQuery.of(context).padding;
    final adProvider = Provider.of<AdMobProvider>(context);
    final appLocal = AppLocalizations.of(context);

    return Scaffold(
      extendBody: true,
      backgroundColor: Colors.white,
      body: Container(
        padding: EdgeInsets.only(top: padding.top),
        decoration: BoxDecoration(color: Colors.white),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.start,
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Padding(
              padding: const EdgeInsets.only(top: 10.0, left: 15),
              child: InkWell(
                onTap: () => Navigator.pop(context),
                child: SvgPicture.asset(
                  AppAssets.iconDelete,
                  width: 20,
                  colorFilter: ColorFilter.mode(Colors.grey, BlendMode.srcIn),
                ),
              ),
            ),
            Container(
              alignment: Alignment.center,
              padding: EdgeInsets.symmetric(horizontal: 25),
              child: Column(
                crossAxisAlignment: CrossAxisAlignment.center,
                children: [
                  Text(
                    appLocal.highlightPageTitle,
                    style: TextStyle(
                      color: Colors.black,
                      fontSize: 23,
                      fontWeight: FontWeight.w700,
                    ),
                    textAlign: TextAlign.center,
                  ),
                  const SizedBox(
                    height: 10,
                  ),
                  Text(
                    appLocal.highlightPageContent,
                    style: TextStyle(
                      color: Colors.black,
                      fontSize: 15,
                    ),
                    textAlign: TextAlign.center,
                  ),
                ],
              ),
            ),
            Expanded(
              flex: 3,
              child: PageView.builder(
                  controller: PageController(viewportFraction: 0.76),
                  onPageChanged: (index) {
                    setState(() {
                      _selectedIndex = index;
                    });
                  },
                  scrollDirection: Axis.horizontal,
                  itemCount: HelpersUserAndValidators.highlightPriceList.length,
                  itemBuilder: (context, index) {
                    var _scale = _selectedIndex == index ? 1.0 : 0.8;

                    return TweenAnimationBuilder(
                      duration: const Duration(milliseconds: 350),
                      tween: Tween(begin: _scale, end: _scale),
                      curve: Curves.ease,
                      child: buildItemPageView(
                          HelpersUserAndValidators.highlightAppbarTitleList(context)[index].toUpperCase(),
                          HelpersUserAndValidators.highlightTitleTimeList(context)[index].toUpperCase(),
                          HelpersUserAndValidators.highlightPriceList[index],
                          HelpersUserAndValidators.highlightTimeList[index]),
                      builder: (context, value, child) {
                        return Transform.scale(
                          scale: value,
                          child: child,
                        );
                      },
                    );
                  }),
            ),
            Row(
              crossAxisAlignment: CrossAxisAlignment.center,
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                ...List.generate(
                    HelpersUserAndValidators.highlightPriceList.length,
                    (index) => Indicator(
                        isActive: _selectedIndex == index ? true : false)),
              ],
            ),
            Expanded(flex: 4, child: Container()),

            adProvider.isLoaded && adProvider.bannerAd != null ?
                Align(
                  alignment: Alignment.bottomCenter,
                  child: SizedBox(
                    width: adProvider.bannerAd?.size.width.toDouble(),
                    height: adProvider.bannerAd?.size.height.toDouble(),
                    child: AdWidget(ad: adProvider.bannerAd!),
                  ),
                ) : SizedBox.shrink()
```

```dart
class HighlightUserNotify extends ChangeNotifier {

  bool  _highlightedUser = false;
  late Timer _countdownTimer;
  String idHighlightedDelete = '';

  bool get highlightedUser => _highlightedUser;

  set highlightedUser(bool value) {
    _highlightedUser = value;
  }
  @override
  void dispose() {
    super.dispose();
  }

  Timer get countdownTimer => _countdownTimer;

  set countdownTimer(Timer value) {
    _countdownTimer = value;
  }

  void startHighlight(UserModel currentUser, UserModel targetUser, int time) {
    activateHighlight(currentUser, targetUser);
    idHighlightedDelete = currentUser.uid;
    print("Id xóa: $idHighlightedDelete");
    _highlightedUser = currentUser.isHighlighted;
    _countdownTimer = Timer(Duration(minutes: time), () async {
      await resetHighlight(currentUser);
      cancelHighlight();
    });
  }
  void cancelHighlight() {
    _highlightedUser = false;
    _countdownTimer.cancel();
    notifyListeners();
  }
  Future<void> activateHighlight(UserModel currentUser, UserModel targetUser) async {
    final currentTime = DateTime.now();
    currentUser.isHighlighted = true;
    currentUser.highlightTime = currentTime.toString();
    await FirebaseFirestore.instance.collection('users').doc(currentUser.uid).update({
      'isHighlighted': true,
      'highlightTime': currentTime.toString(),
    });
    await
      FirebaseFirestore.instance.collection('users').doc(targetUser.uid).collection('highlights').doc(currentUser.uid).set({
      'highlightTime': currentTime.toString(),
    });
    notifyListeners();
  }
  List<UserModel> sortUsers(List<UserModel> users) {
    users.sort((a, b) {
      if (a.isHighlighted && !b.isHighlighted) {
        return -1;
      } else if (!a.isHighlighted && b.isHighlighted) {
        return 1;
      } else if (a.isHighlighted && b.isHighlighted) {
        return b.highlightTime.compareTo(a.highlightTime);
      } else {
        return 0;
      }
    });
    return users;
  }
  Future<void> resetHighlight(UserModel currentUser) async {
    if (currentUser.isHighlighted && currentUser.highlightTime != null) {

        currentUser.isHighlighted = false;
        currentUser.highlightTime = '';

        await FirebaseFirestore.instance.collection('users').doc(currentUser.uid).update({
          'isHighlighted': false,
          'highlightTime': '',
        });

        await FirebaseFirestore.instance.collection('users').doc(currentUser.uid).collection('highlights').doc(idHighlightedDelete).delete();
        notifyListeners();
    }

  }
}

```

The code snippet above displays a list of popular packages over time and allows users to view and select packages for use in an application. It also includes a Provider to store and display the popular packages.

-, Profile Screen

```dart
class DetailProfileOthersScreen extends StatefulWidget {
  final String? uid;
  const DetailProfileOthersScreen({Key? key, this.uid}) : super(key: key);

  @override
  State<DetailProfileOthersScreen> createState() => _DetailProfileOthersScreenState();
}
class _DetailProfileOthersScreenState extends State<DetailProfileOthersScreen> {
  int tappedButtonIndex = -1;
  bool showFullList = false;
  Future<void> _handleTap(int buttonIndex,Function() onTap) async {
    setState(() {
      tappedButtonIndex = buttonIndex;
    });
    await Future.delayed(const Duration(milliseconds: 100));
    setState(() {
      tappedButtonIndex = -1;
    });
    onTap();
  }
  @override
  void initState() {
    super.initState();
    Provider.of<ProfileWatch>(context, listen: false).getUser();
  }
  @override
  Widget build(BuildContext context) {
    final appLocal = AppLocalizations.of(context);
    final adProvider = Provider.of<AdMobProvider>(context,listen: false);
    adProvider.loadBannerAd(context);
    final EdgeInsets padding = MediaQuery.of(context).padding;
    return Scaffold(
      extendBody: true,
        body: FutureBuilder<UserModel>(
            future: context.read<ProfileWatch>().getDetailOthers(widget.uid),
            builder: (context, snapshot) {
              return snapshot.hasData
                  ? SingleChildScrollView(
                    child: Padding( // Padding ···
                  ) // SingleChildScrollView
                  : Container();
            }), // FutureBuilder
      bottomNavigationBar: BottomAppBar( // BottomAppBar ···
    ); // Scaffold
  }
```

More details about the code, please see the links.

-, Children Widgets

-, Providers and Data Connections management.

b) The screen displays user filtering settings based on criteria such as location, gender, and age.

```dart
class DiscoverySetting extends StatelessWidget {
  const DiscoverySetting({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    final provider = context.read<BinderWatch>();
    final appLocal = AppLocalizations.of(context);
    final adProvider = Provider.of<AdMobProvider>(context,listen: false);
    adProvider.loadBannerAd(context);
    return  Scaffold(
          appBar: AppBar(
            backgroundColor: Colors.white,
            elevation: 0.4,
            title: Center(
              child: Text(
                appLocal.settingPageSearchText,
                style: TextStyle(color: Colors.black),
              ),
            ),
            actions: [
              TextButton(
                onPressed: () async {
                  await provider.updateRequestToShow();
                  context.pop("refresh");
                },
                child: Text(
                  appLocal.discoverySettingConfirmText,
                  style: TextStyle(color: Colors.blue[700]),
                ),
              ),
            ],
          ),
          backgroundColor: Colors.grey[200],
          body: SingleChildScrollView(
            padding: EdgeInsets.only(bottom:20),
            child: Column(
              children: [
                BodyDiscoverySetting(isGlobal: false),

                (adProvider.isLoaded && adProvider.bannerAd != null) ?
                Container(
                  width: MediaQuery.of(context).size.width,
                  margin: EdgeInsets.symmetric(vertical: 10),
                  child: SizedBox(
                    width: adProvider.bannerAd?.size.width.toDouble(),
                    height: adProvider.bannerAd?.size.height.toDouble(),
                    child: AdWidget(ad: adProvider.bannerAd!),
                  ),
                ) : SizedBox.shrink(),
                BodyHighSearch()
              ],
            ),
          ),

    );
  }
}
```

- , The main filtering screen

-, Providers and Data Connections management.

c) The notification screen is designed to display a list of push notifications whenever there is a new update or change.

Notification main screen and widgets

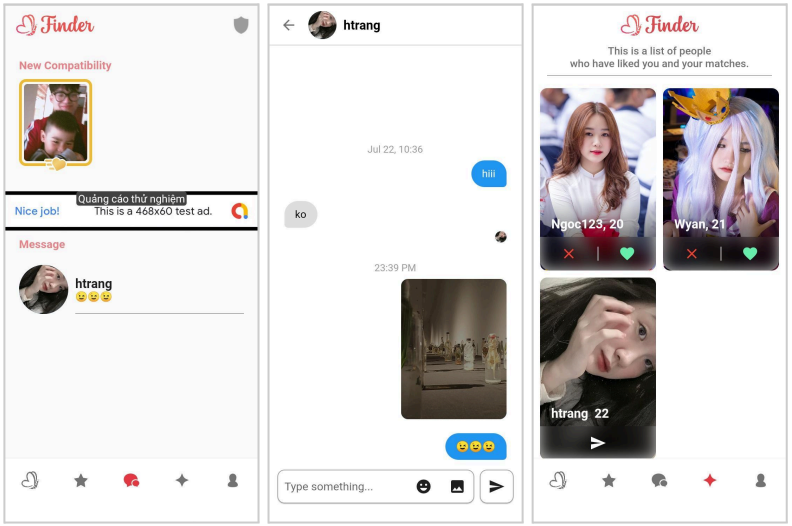
```dart
class NotificationScreen extends StatefulWidget {
  const NotificationScreen({super.key});

  @override
  State<NotificationScreen> createState() => _NotificationScreenState();
}

class _NotificationScreenState extends State<NotificationScreen> {
  @override
  void initState() {
    // TODO: implement initState
    super.initState();
    context.read<NotificationWatch>().getNotification();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        elevation: 1,
        backgroundColor: Colors.white,
        title: Text(
          AppLocalizations.of(context).notificationScreenTitle,
          style: TextStyle(
              color: Colors.black,
              fontWeight: FontWeight.bold,
              fontSize: 22),
        ),
        leading: IconButton(
            onPressed: () => context.pop(),
            icon: const Icon(Icons.arrow_back_outlined, color: Colors.red,)),
      ),
        backgroundColor: Colors.white,
        primary: true,
        body: FutureBuilder(
          future: context.watch<NotificationWatch>().getNotification(),
          builder: (context, snapshot) => snapshot.hasData
              ? snapshot.connectionState == ConnectionState.waiting
              ? ListView.builder(
            physics: const ScrollPhysics(),
            shrinkWrap: true,
            reverse: true,
            itemCount: context
                .watch<NotificationWatch>()
                .listNotification
                .length,
            itemBuilder: (context, index) {
              return ItemNotification(
                title: snapshot.data[index]['type'],
                mess: snapshot.data[index]['mess'],
                imageUrl: snapshot.data[index]['avatar'],
                idUser: snapshot.data[index]['uid'],
                status: snapshot.data[index]['status'],
                time: snapshot.data[index]['time'],
                chatRoomId: snapshot.data[index]['chatRoomId'],
                name: snapshot.data[index]['name'],
              );
            },
          )
              :  Center(
            child: CircularProgressIndicator(),
          )
              : Center(
            child: CircularProgressIndicator(),
          ),
        ));
  }
}
```

- , [Notification main screen and widgets]

-, [Providers and Data Connections management.]

3.4, Messages Screen, who like you

- This feature includes the first interface which is a list of users when both parties have matched each other. Below that, there is a message list where users can exchange messages, send text, images, or emojis. The chat interface allows users to have conversations with each other.
- The second interface displays a list of people who have sent Match requests to the user. The user can choose to accept or decline these requests. If they accept, they can start sending messages to each other as usual.
- Screenshots



- Related code items.(In the link attached to each title) :
    a) The screen displaying "Who Likes You" and the functionalities to accept or reject.
    - Who like you main screen
    - Providers and Data Connections management.
    - Below is the general layout of the "Who Like You" screen.

```dart
class WhoLikePage extends StatefulWidget {
  const WhoLikePage({Key? key}) : super(key: key);

  @override
  State<WhoLikePage> createState() => _WhoLikePageState();
}

class _WhoLikePageState extends State<WhoLikePage> {
  @override
  void initState() {
    // context.read<FollowNotify>().userFollowYou();
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    final appLocal = AppLocalizations.of(context);

    return Scaffold(
      appBar: AppBar( // AppBar ···
      backgroundColor: Colors.white,
      body: SingleChildScrollView(
        child: Column( // Column ···
      ), // SingleChildScrollView
    ); // Scaffold
  }
}
```

Inside the Column widget, there are several widgets that configure the interface and widgets that contain the main content:

```dart
FutureBuilder(
    future: context.watch<FollowNotify>().userFollowYou(),
    builder: (context, AsyncSnapshot<List<UserModel>> snapshot) {
      if (snapshot.data == null) {
        return Container();
      } else {
        return snapshot.hasData
            ? GridView.builder(
                itemCount: snapshot.data!.length,
                shrinkWrap: true,
                addAutomaticKeepAlives: false,
                physics: const ScrollPhysics(),
                padding: const EdgeInsets.all( 10),
                gridDelegate:
                    const SliverGridDelegateWithFixedCrossAxisCount(
                        crossAxisCount: 2,
                        mainAxisSpacing: 10,
                        crossAxisSpacing: 10,
                        mainAxisExtent: 260),
                itemBuilder: (context, index) {
                  UserModel user = snapshot.data![index];
                  return LikedUserCard(
                    user: user,
                  );
                },
              )
            : const Center(
                child: CircularProgressIndicator(),
              );
      }
    }),
```

The code uses FutureBuilder to handle a future obtained from the userFollowYou function of the FollowNotify provider. Based on the result of the future, it displays a GridView containing a list of LikedUserCards or a circular progress indicator (CircularProgressIndicator) while loading the data.

Provider FollowNotify:

```dart
class FollowNotify extends ChangeNotifier {
  Future<List<UserModel>> userFollowYou() async {
    try {
      String uid =
          await HelpersFunctions().getUserIdUserSharedPreference() as String;
      List<UserModel> userFollow = await DatabaseMethods().getUserFollow(uid);
      notifyListeners();
      return userFollow;
    } catch (e) {
      throw Exception(e);
    }
  }
}
```

Widget LikedUserCard :

```dart
class LikedUserCard extends StatelessWidget {
  const LikedUserCard({super.key, this.user});

  final UserModel? user;

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: () {
        context
            .goNamed('home-detail-others', queryParameters: {'uid': user!.uid});
      },
      child: Container(
        decoration: BoxDecoration(
          borderRadius: BorderRadius.circular(15),
        ), // BoxDecoration
        child: Stack(children: [
          Container(···
              decoration: BoxDecoration( // BoxDecoration // Container ···
          Container(
            decoration: BoxDecoration( // BoxDecoration ···
          ), // Container
          FutureBuilder(
              future: context
                  .watch<LikedUserCardProvider>()
                  .checkMatched(user!.uid),
              builder: (context, snapshot) {
                return snapshot.hasData
                    ? Positioned( // Positioned ···
                    : Positioned( // Positioned ···
              }) // FutureBuilder
        ]), // Stack
      ), // Container
    ); // GestureDetector
  }
}
```

The code utilizes GestureDetector and FutureBuilder to handle futures and display user information in the form of a card with an image and options to send messages or perform other actions. Based on the future obtained from the checkMatched function of the

LikeUserCardProvider provider, it checks for matches to determine which functional buttons should be displayed. When tapping on a LikedUserCard item, it navigates to another user's profile information (DetailProfileOthersScreen) using the context.goNamed method.

Provider LikeUserCardProvider:

```dart
class LikedUserCardProvider extends ChangeNotifier {
  Future<ChatRoom?> checkMatched(String otherUid) async {
    try {
      final uid =
          await HelpersFunctions().getUserIdUserSharedPreference() as String;
      String firstCheckString =
          await DatabaseMethods().checkFollow(uid, otherUid);
      if (firstCheckString == "follow") {
        ChatRoom? chatRoom = await DatabaseMethods().getChatRoom(otherUid);
        notifyListeners();
        return chatRoom;
      }
      return null;
    } catch (e) {
      throw Exception(e);
    }
  }

  Future addFollow(String followId) async {
    try {
      final uid =
          await HelpersFunctions().getUserIdUserSharedPreference() as String;
      await DatabaseMethods().addFollow(uid, followId);
      String check = await DatabaseMethods().checkFollow(uid, followId);
      DateTime currentTime = await NTP.now();
      String time = currentTime.toString();
      if (check == "follow") {
        List<String> users = [uid, followId];
        String chatRoomId = getChatRoomId(uid, followId);
        List<dynamic> userTimes = [UserTime(uid: uid, time: time).toJson(),
          UserTime(uid: followId, time: time).toJson()];
        Map<String, dynamic> chatRoom = {
          "users": users,
          "chatRoomId": chatRoomId,
          "userTimes": userTimes,
          "time": time,
          "newChatRoom": []
        };
        await DatabaseMethods().addChatRoom(chatRoom, chatRoomId);
        notifyListeners();
      }
    } catch (e) {
      throw Exception(e);
    }
  }

  Future removeFollow(String followId) async {
    try {
      final uid =
          await HelpersFunctions().getUserIdUserSharedPreference() as String;
      await DatabaseMethods().removeFollow(uid, followId);
      // notifyListeners();
    } catch (e) {
      throw Exception(e);
    }
  }

  getChatRoomId(String a, String b) {
    if (a.substring(0, 1).codeUnitAt(0) > b.substring(0, 1).codeUnitAt(0)) {
      return "${b}_$a";
    } else {
      return "${a}_$b";
    }
  }
}
```

The functions "addFollow" and "removeFollow" are used for feature buttons when there is no match yet. The function "getChatRoomId" is used to retrieve the chat room ID for use when pressing feature buttons after a match has been made.

b) The main messaging screen and the chat screen for conversing with each other.

- , Main Messages Screen, Message details

- Below is the general layout of the conversation list screen:

```dart
class MyMessageScreen extends StatelessWidget {
  const MyMessageScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.transparent,
        elevation: 0,
        title: Row(
          children: [
            SvgPicture.asset(
              AppAssets.iconTinder,
              width: 30,
              height: 30,
              fit: BoxFit.cover,
            ),
            const SizedBox(
              width: 5,
            ),
            const Text(
              "Finder",
              style: TextStyle(
                fontFamily: 'Grandista',
                fontSize: 24,
                color: Color.fromRGBO(223, 54, 64, 100),
              ),
            ),
          ],
        ),
        actions: [
          IconButton(
            onPressed: () {},
            icon: Image.asset(
              AppAssets.iconShield,
              color: Colors.grey,
              width: 25,
            ),
          ),
        ],
      ),
      body: _buildBody(context),
    );
  }

```

```dart
_buildBody(BuildContext context) {
    final adProvider = Provider.of<AdMobProvider>(context,listen: false);
    adProvider.loadBannerAd(context);
    return BlocBuilder<MessageBloc, MessageState>(
      builder: (context, state) {
        if (state is ChatRoomsLoading) {
          return const Center(child: CircularProgressIndicator());
        }
        if (state is ChatRoomsLoaded) {
          return SingleChildScrollView(
            child: Container(
              color: Colors.transparent,
              child: Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  //search(context),
                  Container(
                    margin: const EdgeInsets.only(left: 20, top: 20, bottom: 10),
                    child:  Text(
                      AppLocalizations.of(context).messageScreenTitle1,
                      style: TextStyle(
                          fontSize: 16,
                          fontWeight: FontWeight.bold,
                          color: Color.fromRGBO(229, 58, 69, 100)),
                    ),
                  ),
                  SizedBox(
                    height: 160,
                    child: _buildNewChatRoomsList(state.newChatRoomsStream,
                        state.currentUserId, state.user),
                  ),
                  (adProvider.isLoaded && adProvider.bannerAd != null) ?
                  Container(
                    width: MediaQuery.of(context).size.width,
                    child: SizedBox(
                      width: adProvider.bannerAd?.size.width.toDouble(),
                      height: adProvider.bannerAd?.size.height.toDouble(),
                      child: AdWidget(ad: adProvider.bannerAd!),
                    ),
                  ) : SizedBox.shrink(),
                  Container(
                    margin: const EdgeInsets.only(left: 20, bottom: 20,top: 10),
                    child:  Text(
                      AppLocalizations.of(context).messageScreenTitle2,
                      style: TextStyle(
                          fontSize: 16,
                          fontWeight: FontWeight.bold,
                          color: Color.fromRGBO(229, 58, 69, 100)),
                    ),
                  ),
                  SingleChildScrollView(
                    child: _buildChatRoomsList(
                        state.chatRoomsStream, state.currentUserId, state.user),
                  )
                ],
              ),
            ),
          );
        }
        return const SizedBox();
      },
    );
  }
```

This screen utilizes "BlocBuilder" to manage the state and build the interface based on events and states provided by "MessageBloc," "MessageEvent," and "MessageState." If the state is "ChatRoomsLoading," it will display a centered header with a "CircularProgressIndicator" to let the user know that data is being loaded. If the state is "ChatRoomsLoaded," it will display the interface with a list of new conversations and a list of previously joined user conversations.

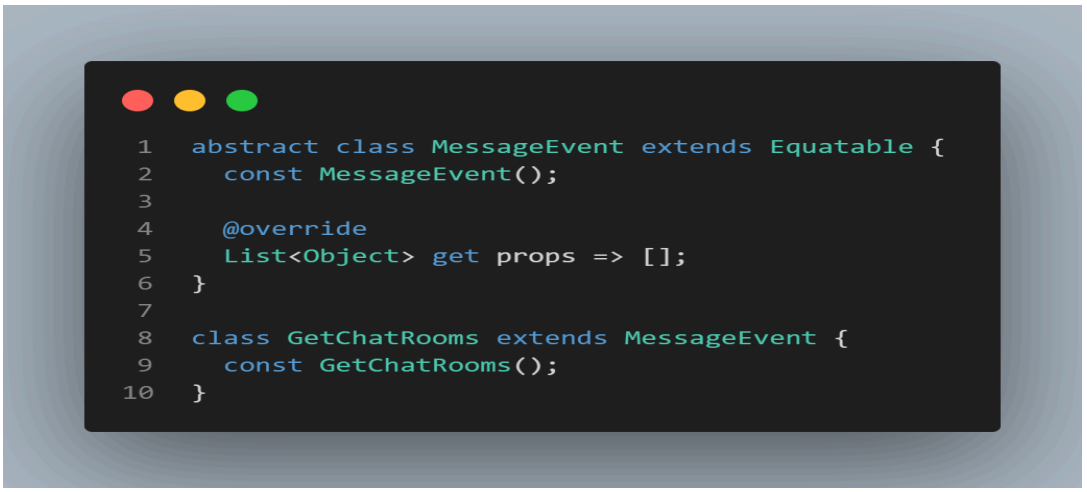
- Message Bloc

```dart
class MessageBloc extends Bloc<MessageEvent, MessageState> {

  final GetChatRoomsUseCase _getChatRoomsUseCase;
  final GetMyInfoUseCase _getMyInfoUseCase;
  final GetNewChatRoomsUseCase _getNewChatRoomsUseCase;

  MessageBloc(this._getChatRoomsUseCase,this._getMyInfoUseCase,this._getNewChatRoomsUseCase) : super(const MessageInitial()) {
    emit(const ChatRoomsLoading());
    on<GetChatRooms>(getChatRooms);
  }

  FutureOr<void> getChatRooms(GetChatRooms event, Emitter<MessageState> emit) async {
    String? uid = await HelpersFunctions().getUserIdUserSharedPreference();
    UserEntity user = await _getMyInfoUseCase(uid!);
    emit(ChatRoomsLoaded(_getChatRoomsUseCase(uid!),uid!,user,_getNewChatRoomsUseCase(uid!)));
  }
}
```

-Message State

```dart
abstract class MessageState extends Equatable {
  const MessageState();
  @override
  List<Object> get props => [];
}

class MessageInitial extends MessageState  {
  const MessageInitial();
}

class ChatRoomsLoading extends MessageState {
  const ChatRoomsLoading();
}

class ChatRoomsLoaded extends MessageState {
  final String currentUserId;
  final UserEntity user;
  final Stream<List<ChatRoomEntity>> chatRoomsStream;
  final Stream<List<ChatRoomEntity>> newChatRoomsStream;
  const ChatRoomsLoaded(this.chatRoomsStream, this.currentUserId,this.user,this.newChatRoomsStream);
}

```

- Message Event



```dart
abstract class MessageEvent extends Equatable {
  const MessageEvent();

  @override
  List<Object> get props => [];
}

class GetChatRooms extends MessageEvent {
  const GetChatRooms();
}
```

- The list of conversations is divided into two parts: the section of new conversations and the section of previously joined user conversations.

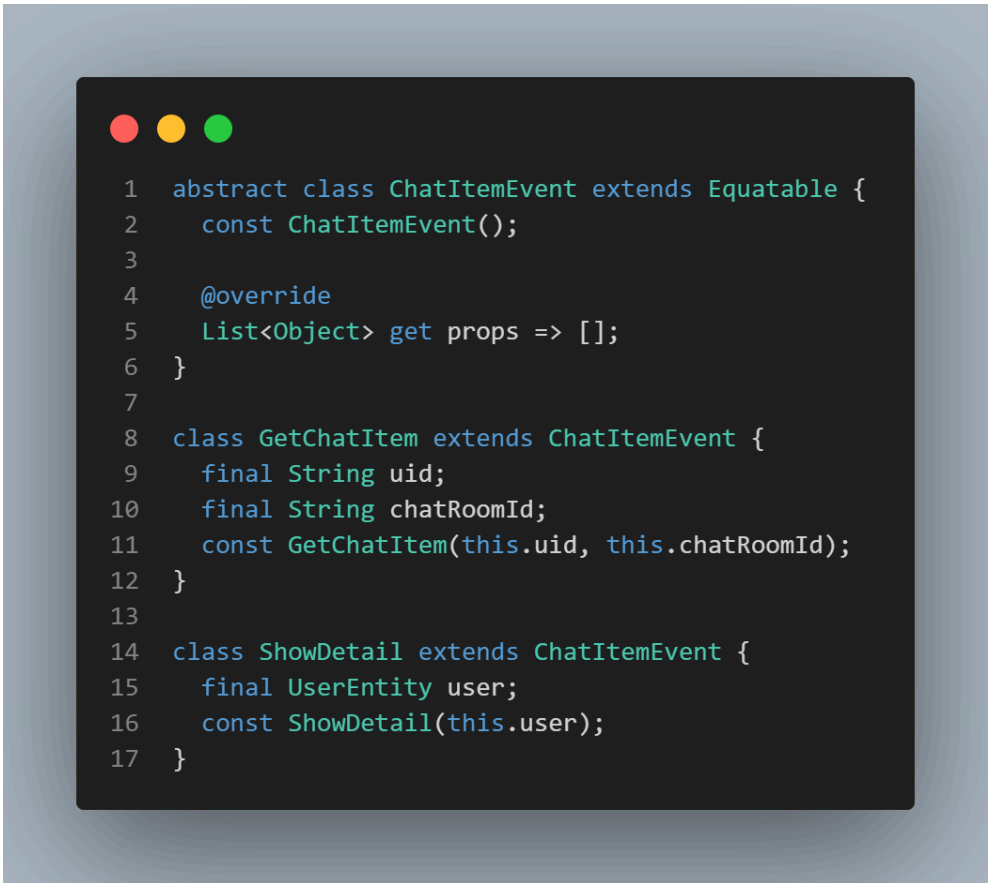
```dart
class CircleMessageItem extends StatelessWidget {
  const CircleMessageItem(
      {super.key, required this.uid, required this.chatRoomId});

  final String uid;
  final String chatRoomId;

  @override
  Widget build(BuildContext context) {
    return BlocConsumer<ChatItemBloc, ChatItemState>(
      listenWhen: (previous, current) => current is ChatItemActionState,
      buildWhen: (previous, current) => current is! ChatItemActionState,
      listener: (context, state) {
        if (state is ChatItemClicked) {
          context.goNamed('detail-message',queryParameters: {
            'uid': uid,
            'chatRoomId':chatRoomId,
            'name': state.user.fullName,
            'avatar': state.user.avatar,
            'token': state.user.token
          });
          BlocProvider.of<ChatItemBloc>(context)
              .add(GetChatItem(uid!, chatRoomId!));
        }
      },
      builder: (context, state) {
        if (state is ChatItemLoaded) {
          return GestureDetector(
            onTap: () {
              BlocProvider.of<ChatItemBloc>(context)
                  .add(ShowDetail(state.user));
            },
            child: Container(
              margin: const EdgeInsets.only(left: 20),
              child: Column(
                children: [
                  newChatRoom(state.user.avatar!, AppAssets.iconStar2, state.isNewChatRoom),
                  Container(
                      margin: const EdgeInsets.only(top: 5),
                      child: Text(
                        state.user.fullName ?? "",
                        style: const TextStyle(
                          fontSize: 14,
                          fontWeight: FontWeight.bold,
                        ),
                        maxLines: 1,
                        overflow: TextOverflow.ellipsis,
                      ))
                ],
              ),
            ),
          );
        }
        return const SizedBox();
      },
    );
  }
```

- The code above is used to display information about each new conversation, including the avatar and name. It utilizes "BlocConsumer" to manage the state and build the interface based on events and states provided by "ChatItemBloc," "ChatItemState," and "ChatItemEvent."

- ChatItemState

```dart
abstract class ChatItemState extends Equatable {
  const ChatItemState();
  @override
  List<Object> get props => [];
}

class ChatItemActionState extends ChatItemState {
  const ChatItemActionState();
}

class ChatItemInitial extends ChatItemState {
  const ChatItemInitial();
}

class ChatItemLoaded extends ChatItemState {
  final UserEntity user;
  final bool isNewChatRoom;
  final Stream<ChatMessageEntity> lastMessageStream;
  const ChatItemLoaded(this.user, this.lastMessageStream,this.isNewChatRoom);
}

class ChatItemClicked extends ChatItemActionState {
  final UserEntity user;
  const ChatItemClicked(this.user);
}
```

- ChatItemEvent



```dart
abstract class ChatItemEvent extends Equatable {
  const ChatItemEvent();

  @override
  List<Object> get props => [];
}

class GetChatItem extends ChatItemEvent {
  final String uid;
  final String chatRoomId;
  const GetChatItem(this.uid, this.chatRoomId);
}

class ShowDetail extends ChatItemEvent {
  final UserEntity user;
  const ShowDetail(this.user);
}
```

- ChatItemBloc

```dart
class ChatItemBloc extends Bloc<ChatItemEvent, ChatItemState> {
  final GetLastMessageUseCase _getLastMessageUseCase;
  final GetUserInformationUserCase _getUserInformationUserCase;
  final GetNewChatRoomUseCase _getNewChatRoomUseCase;
  ChatItemBloc(this._getLastMessageUseCase, this._getUserInformationUserCase,this._getNewChatRoomUseCase)
      : super(const ChatItemInitial()) {
    on<GetChatItem>(getChatItem);
    on<ShowDetail>(showDetail);
  }

  FutureOr<void> getChatItem(GetChatItem event, Emitter<ChatItemState> emit) async {
    UserEntity user = await _getUserInformationUserCase(event.uid);
    Stream<ChatMessageEntity> lastMessageStream = _getLastMessageUseCase(event.chatRoomId);
    ChatRoomEntity chatRoom = await _getNewChatRoomUseCase(event.chatRoomId);
    List<String>? newUsers = chatRoom.newChatRoom;
    bool isNewChatRoom = true;
    String? myUid = await HelpersFunctions().getUserIdUserSharedPreference();
    for(var index in newUsers!){
      if(index == myUid){
        isNewChatRoom = false;
      }
    }
    emit(ChatItemLoaded(user, lastMessageStream, isNewChatRoom));
  }

  FutureOr<void> showDetail(ShowDetail event, Emitter<ChatItemState> emit) {
    emit(ChatItemClicked(event.user));
  }
}
```

```dart
class MyItemMessage extends StatelessWidget {
  final String? uid;
  final String? chatRoomId;

  const MyItemMessage({super.key, this.uid, this.chatRoomId});

  @override
  Widget build(BuildContext context) {
    return FutureBuilder(
        future: Provider.of<ItemMessageProvider>(context,listen: false).getUserInformation(uid!),
        builder: (BuildContext context, AsyncSnapshot<dynamic> snapshot) {
          if (!snapshot.hasData) return const SizedBox.shrink();
          UserEntity user = snapshot.data;
          return snapshot.hasData
              ? GestureDetector(
                  onTap: () {
                    context.goNamed('detail-message', queryParameters: {
                      'uid': user.uid,
                      'chatRoomId': chatRoomId,
                      'name': user.fullName,
                      'avatar': user.avatar,
                    });
                  },
                  child: Container(
                    color: Colors.transparent,
                    margin: const EdgeInsets.only(left: 20, bottom: 20),
                    width: 70,
                    height: 70,
                    child: Row(
                      children: [
                        CircleAvatar(
                          radius: 35,
                          backgroundImage: NetworkImage(user?.avatar ?? ""),
                          child: user?.activeStatus == "online"
                              ? Stack(children: [
                                  const Align(
                                    alignment: Alignment.bottomRight,
                                    child: CircleAvatar(
                                      radius: 10,
                                      backgroundColor: Colors.white,
                                    ),
                                  ),
                                  Align(
                                      alignment: Alignment.bottomRight,
                                      child: Container(
                                        width: 20,
                                        height: 20,
                                        decoration: BoxDecoration(
                                          shape: BoxShape.circle,
                                          border: Border.all(
                                            color: Colors.white,
                                            width: 2,
                                          ),
                                        ),
                                        child: const CircleAvatar(
                                          backgroundColor: Colors.green,
                                        ),
                                      )),
                                ])
                              : null,
                        ),
                        Expanded(
                          child: Container(
                            margin: const EdgeInsets.only(left: 10, right: 20),
                            decoration: const BoxDecoration(
                              border: Border(
                                bottom: BorderSide(
                                  color: Colors.grey,
                                  width: 1.0,
                                ),
                              ),
                            ),
                          ),
```

The code above is used to display information about the conversation between a user and another user in a list of conversations. This includes the avatar, username, and the content of

the last message. The code utilizes a FutureBuilder to handle a future obtained from the getUserInformation function in ItemMessageProvider. If the future contains data, it displays the information on the screen.

ItemMessageProvider

```dart
class ItemMessageProvider extends ChangeNotifier{
  final GetLastMessageUseCase _getLastMessageUseCase;
  final GetUserInformationUserCase _getUserInformationUserCase;

  ItemMessageProvider(this._getLastMessageUseCase, this._getUserInformationUserCase);

  getUserInformation(String uid) async {
    UserEntity? user = await _getUserInformationUserCase(uid);
    return user;
  }
  getLassMessage(String chatRoomId){
    Stream<ChatMessageEntity> lastMessageStream = _getLastMessageUseCase(chatRoomId);
    return lastMessageStream;
  }
}
```

- When users tap on a conversation, they will be taken to the messaging screen. Below is an overview of the messaging screen used to display detailed conversations between two users in a chat application. This widget allows users to view and send text and image messages. It utilizes "BlocBuilder" to manage the state and build the interface based on events and states provided by "DetailMessageBloc," "DetailMessageState," and "DetailMessageEvent."

```dart
class DetailMessage extends StatefulWidget {
  const DetailMessage(
      {super.key, this.uid, this.chatRoomId, this.name, this.avatar});

  final String? uid;
  final String? chatRoomId;
  final String? name;
  final String? avatar;

  @override
  State<DetailMessage> createState() => _DetailMessageState();
}

class _DetailMessageState extends State<DetailMessage> {
  final TextEditingController messageController = TextEditingController();
  var keyUid = '';
  var isShowEmoji = false;
  var watchTime = '';

  Future<void> getKeyUid() async {
    keyUid = await HelpersFunctions().getUserIdUserSharedPreference() as String;
  }

  @override
  Widget build(BuildContext context) {
    getKeyUid();
    return BlocConsumer<DetailMessageBloc, DetailMessageState>(
      listener: (context, state) {},
      builder: (context, state) {
        return Scaffold(
          appBar: AppBar(
            titleSpacing: 0,
            elevation: 1,
            backgroundColor: Colors.white,
            title: state is MessageListLoaded
                ? head(state.user)
                : const CircularProgressIndicator(),
            leading: InkWell(
              onTap: () {
                Navigator.pop(context);
              },
              child: const Icon(
                Icons.arrow_back,
                color: Colors.black54,
              ),
            ),
          ),
          backgroundColor: Colors.transparent,
          body: Container(
            padding: const EdgeInsets.symmetric(horizontal: 8),
            decoration: const BoxDecoration(color: Colors.white),
            child: Builder(builder: (context) {
              if (state is MessageListLoaded) {
                return Column(
                  children: [
                    Expanded(
                        child: listMessage(state.messagesList, state.chatRoom,
                            state.watchTime)),
                    controlMessage(context, state.showEmoji, state.image),
                    state.showEmoji == true
                        ? emoji(context)
                        : const SizedBox.shrink()
                  ],
                );
              }
              return const SizedBox.shrink();
            }),
          ),
        );
      },
    );
  }
```

- DetailMessageBloc

```dart
class DetailMessageBloc extends Bloc<DetailMessageEvent, DetailMessageState> {
  final GetMessagesUseCase _getMessagesUseCase;
  final AddMessageUseCase _addMessageUseCase;
  final CompareUserTimeUseCase _compareUserTimeUseCase;
  final GetChatRoomUseCase _getChatRoomUserCase;
  final GetInfoUserUseCase _getInfoUserUseCase;

  DetailMessageBloc(
      this._getMessagesUseCase,
      this._addMessageUseCase,
      this._compareUserTimeUseCase,
      this._getChatRoomUserCase,
      this._getInfoUserUseCase)
      : super(const DetailMessageInitial()) {
    on<GetMessageList>(getMessageList);
    on<AddMessage>(addMessage);
    on<CompareUserTime>(compareUserTime);
  }

  FutureOr<void> getMessageList(
      GetMessageList event, Emitter<DetailMessageState> emit) async {
    emit(const MessageListLoading());
    emit(MessageListLoaded(
      _getChatRoomUserCase(event.uid,event.chatRoomId),
      _getMessagesUseCase(event.chatRoomId),
      _getInfoUserUseCase(event.uid),
      event.showEmoji,
      event.image,
      event.watchTime));
  }

  FutureOr<void> addMessage(
      AddMessage event, Emitter<DetailMessageState> emit) {
    _addMessageUseCase(event.uid, event.chatRoomId, event.content,
        event.image, event.avatar, event.name);

  }

  FutureOr<void> compareUserTime(
      CompareUserTime event, Emitter<DetailMessageState> emit) {
    _compareUserTimeUseCase(event.uid, event.chatRoomId);
  }

}

```

- DetailMessageState

```dart
abstract class DetailMessageState extends Equatable {
  const DetailMessageState();

  @override
  List<Object> get props => [];
}

class DetailMessageActionState extends DetailMessageState {
  const DetailMessageActionState();
}

class DetailMessageInitial extends DetailMessageState {
  const DetailMessageInitial();
}

class MessageListLoading extends DetailMessageState {
  const MessageListLoading();
}

class MessageListLoaded extends DetailMessageState {
  final bool showEmoji;
  final String watchTime;
  final File? image;
  final Stream<List<ChatMessageEntity>> messagesList;
  final Stream<ChatRoomEntity> chatRoom;
  final Stream<UserEntity> user;
  const MessageListLoaded(this.chatRoom,this.messagesList,this.user,this.showEmoji,this.image,this.watchTime);
}
```

- DetailMessageEvent

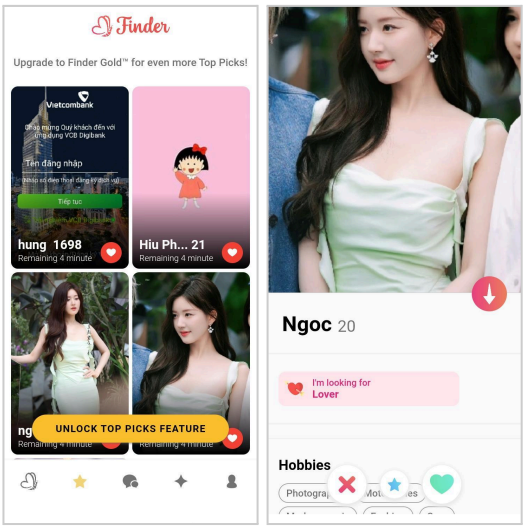
```dart
abstract class DetailMessageEvent extends Equatable {
  const DetailMessageEvent();

  @override
  List<Object> get props => [];
}

class GetMessageList extends DetailMessageEvent {
  final String watchTime;
  final bool showEmoji;
  final File? image;
  final String uid;
  final String chatRoomId;
  const GetMessageList(this.uid ,this.chatRoomId,this.showEmoji,this.image,this.watchTime);
}

class AddMessage extends DetailMessageEvent {
  final String uid;
  final String chatRoomId;
  final String content;
  final File? image;
  final String avatar;
  final String name;

  const AddMessage(this.uid, this.chatRoomId, this.content, this.image,
      this.avatar, this.name);
}

class CompareUserTime extends DetailMessageEvent {
  final String uid;
  final String chatRoomId;

  const CompareUserTime(this.uid, this.chatRoomId);
}
```

-

-, [Providers and Data Connections management.]

3.5, Top Picks Display Screen

- This feature includes a user interface that shows a list of users filtered according to various criteria such as most active, recently online, highly prominent, etc. When clicking on a user, it will display detailed information about that user, while still sending a match request as usual.
- ScreenShot



- Related code items.(In the link attached to each title) :
  a) Top Picks Display Screen

  - , MainScreen

  -, Providers and Data Connections management.

Screenshots:

BinderSelection.dart

```dart
class BinderSelection extends StatelessWidget {
  const BinderSelection({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {

    return Scaffold(
      backgroundColor: Colors.white,
      appBar: AppBar(
        backgroundColor: Colors.white,
        elevation: 0,
        title: Align(
          alignment: Alignment.center,
          child: Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              SvgPicture.asset(
                AppAssets.iconTinder,
                width: 30,
                height: 30,
                fit: BoxFit.cover,
              ),
              const SizedBox(
                width: 5,
              ),
              const Text(
                "Finder",
                style: TextStyle(
                  fontFamily: 'Grandista',
                  fontSize: 24,
                  color: Color.fromRGBO(225, 72, 80, 1.0),
                ),
              ),
            ],
          ),
        ),
      ),
      body: BodySelection(),
    );
  }
}
```

The Body widget of this code snippet has been extracted into a separate file to improve readability and maintainability.

Body.dart

```dart

class BodySelection extends StatelessWidget {
  final _scrollController = ScrollController();

  BodySelection({Key? key}) : super(key: key);


  void _scrollListener(BuildContext context) {
    if (_scrollController.offset >=
        _scrollController.position.maxScrollExtent) {
      _showBottomModal(
        context: context,
        color: Colors.yellow,
        title: "Binder",
        isHaveIcon: false,
        packageModel: packageBinderGoldList(context),
        assetsBanner: AppAssets.iconTinderGoldBanner,
        assetsIcon: AppAssets.iconTinderGold,
        isHaveColor: true,
        subTitle: AppLocalizations.of(context).selectionPageSubTitle,
      );
    }
  }

  void _showBottomModal({
    required BuildContext context,
    required Color color,
    Color? iconColor,
    bool? isHaveColor,
    bool? isHaveIcon,
    bool? isSuperLike,
    IconData? iconData,
    String? assetsBanner,
    String? assetsIcon,
    List<PackageModel>? packageModel,
    required String subTitle,
    required String title,
  }) {

    showModalBottomSheet(
      context: context,
      isScrollControlled: true,
      isDismissible: true,
      builder: (BuildContext context) {
        return BottomModalFullScreen(
          packageModel: packageModel,
          color: color,
          assetsBanner: assetsBanner,
          assetsIcon: assetsIcon,
          title: title,
          subTitle: subTitle,
          iconColor: iconColor,
          isHaveColor: isHaveColor ?? false,
          isHaveIcon: isHaveIcon ?? false,
          iconData: iconData,
          isSuperLike: isSuperLike ?? false,
        );
      },
    );
  }


```

In this code snippet, there are two private functions to display modals and one function to handle the event when the user scrolls the screen. When the user reaches the bottom, the _showBottomModal function will be called to display the modal. Now, let's continue with the content of the code inside the build function:

```dart
@override
 Widget build(BuildContext context) {
   _scrollController.addListener(() => _scrollListener(context));
   final appLocal = AppLocalizations.of(context);
   return Scaffold(
     backgroundColor: Colors.white,
     body: Stack(
       children: [
         SingleChildScrollView(
           controller: _scrollController,
           child: Padding(
             padding: EdgeInsets.only(top: 15),
             child: Column(
               mainAxisAlignment: MainAxisAlignment.start,
               crossAxisAlignment: CrossAxisAlignment.stretch,
               children: [
                 Text(
                   appLocal.selectionPageContent1,
                   textAlign: TextAlign.center,
                   style: TextStyle(
                     fontSize: 15,
                     fontWeight: FontWeight.w500,
                     color: Colors.grey[600],
                   ),
                 ),
                 SizedBox(
                   height: 15,
                 ),

                 getBody(context),
               ],
             ),
           ),
         ),
         Align(
           alignment: Alignment.bottomCenter,
           child: Padding(
             padding: const EdgeInsets.only(bottom: 20.0),
             child: ElevatedButton(
               onPressed: () => _showBottomModal(
                 context: context,
                 color: Colors.yellow,
                 isHaveIcon: false,
                 assetsBanner: AppAssets.iconTinderGoldBanner,
                 assetsIcon: AppAssets.iconTinderGold,
                 title: "Binder",
                 isHaveColor: true,
                 iconData: null,
                 packageModel: packageBinderGoldList(context),
                 subTitle: appLocal.selectionPageContent2,
               ),
               style: ElevatedButton.styleFrom(
                 shadowColor: Colors.grey,
                 fixedSize: Size(280, 40),
                 backgroundColor: Colors.yellow[700],
                 shape: RoundedRectangleBorder(
                   borderRadius: BorderRadius.circular(30),
                 ),
               ),
               child: Text(
                 appLocal.selectionPageContent3,
                 style: TextStyle(
                   color: Colors.black,
                   fontWeight: FontWeight.bold,
                   letterSpacing: 1,
                 ),
               ),
             ),
           ),
         ),
       ],
     ),
   );
 }
}
```

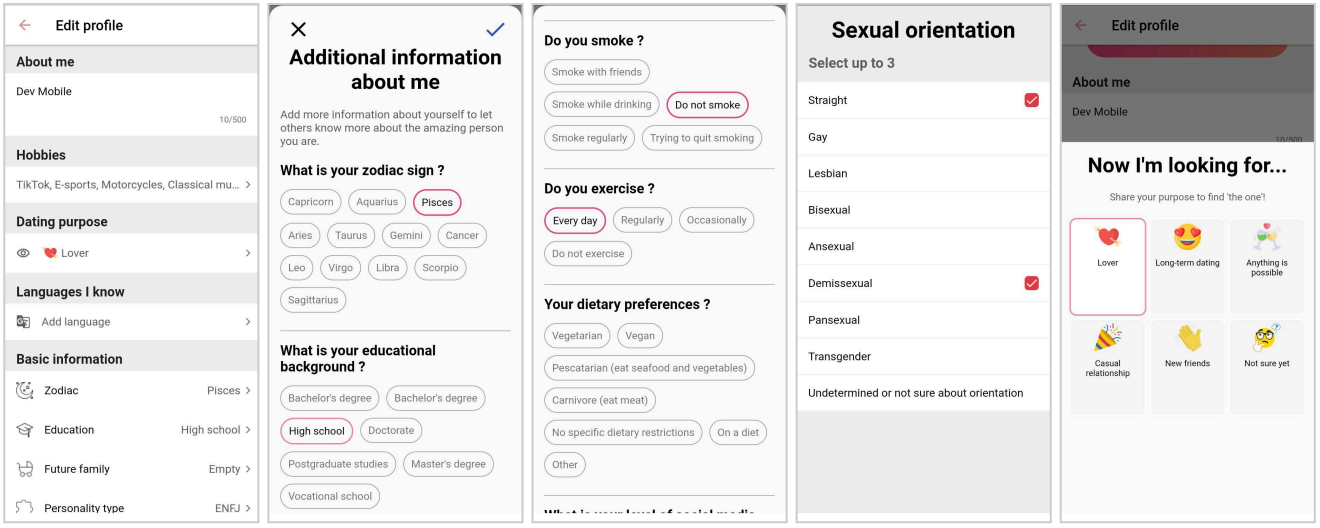
```dart
Widget getBody(BuildContext context) {
  return FutureBuilder(
    future: context.read<BinderWatch>().allBinderSelectionUser(),
    builder: (context, snapshot) =>
      snapshot.hasData ?
        Padding(
        padding: const EdgeInsets.all(10),
        child: GridView.builder(
          shrinkWrap: true,
          physics: NeverScrollableScrollPhysics(),
          gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
            crossAxisCount: 2,
            crossAxisSpacing: 5,
            mainAxisSpacing: 5,
            mainAxisExtent: 260,
          ),
          itemCount: context.watch<BinderWatch>().listCard.length,
          itemBuilder: (context, index) {
            return ItemSelectionCard(
              onTap: () {
                Provider.of<LikedUserCardProvider>(context, listen: false)
                    .addFollow(
                    context.read<BinderWatch>().listCard[index].uid);
                context.read<BinderWatch>().removeCardAtIndex(index);
              },
              user: context.read<BinderWatch>().listCard[index],
              isDetail: () => context.goNamed(
                'home-detail-others',
                queryParameters: {
                  'uid': context
                      .read<BinderWatch>()
                      .listCard[index]
                      .uid
                      .toString()
                },
              ),
            );
          },
        ),
      ): Center(

        child: LoadingAnimationWidget.fallingDot(
          color: Color.fromRGBO(234, 64, 128, 100),
          size: 60,
        ),
      ),
  );
}
```

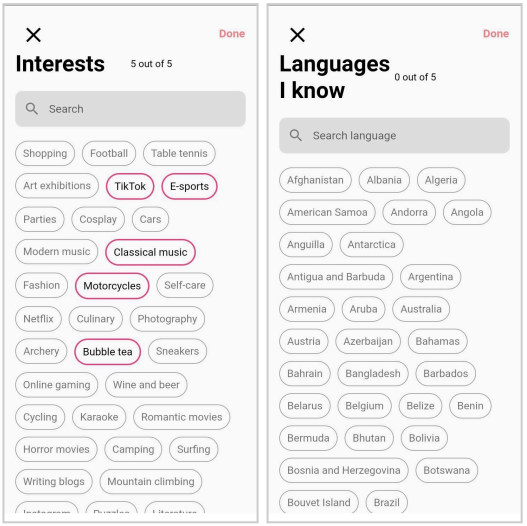
After initializing the interface, the allUserSelectionBinder() function will be used to filter out individuals with the same interests and gender. The filtered results will be displayed on a GridView based on the ItemSelectionCard.

You can access the following link to view this code snippet: [item_selection_card]

3.6, The profile display screen

- This function includes a user interface displaying personal information and editable details, with multiple diverse fields to customize and make the profile more appealing and outstanding.
- Screenshots:





- Related code items(In the link attached to each title.) :
  a) The screen displays the details of one's personal profile, and the tabs to update personal information such as photos, biography, interests, lifestyle, objectives, language, and location.

The general layout of the screen for displaying and updating user information would be implemented as a StatefulWidget, using the provider UpdateNotify to manage the state.

```dart
class UpdateProfileScreen extends StatefulWidget {
  const UpdateProfileScreen({super.key});
  @override
  State<UpdateProfileScreen> createState() => _UpdateProfileScreenState();
}
class _UpdateProfileScreenState extends State<UpdateProfileScreen> {
  @override
  void initState() {
    super.initState();
    Provider.of<UpdateNotify>(context, listen: false).getUser(true);
  }
  @override
  Widget build(BuildContext context) {
    var size = MediaQuery.of(context).size;
    final double itemHeight = (size.height - 210) / 2;
    final double itemWidth = size.width / 2;
    final updateProvider = Provider.of<UpdateNotify>(context);
    final appLocal = AppLocalizations.of(context);
    return WillPopScope(
      onWillPop: () async {
        await Future.delayed(const Duration(seconds: 3)).then((value) async {···
        return true;
      },
      child: Scaffold(
          backgroundColor: Colors.white,
          appBar: AppBar( // AppBar ···
          body: Stack(children: [
            SingleChildScrollView(
              child: Column( // Column ···
            ), // SingleChildScrollView
            if (updateProvider.isLoading)
              Positioned.fill( // Positioned.fill ···
          ])), // Stack // Scaffold
    ); // WillPopScope
  }
}
```

Use the WillPopScope widget to handle the event when the user presses the back button to save the changed information. The Stack widget will contain two child widgets: one is SingleChildScrollView to display the main content, and the other is a widget to display loading while data is being loaded (when the isLoading variable of the UpdateNotify provider is true).

The main content is contained within a Column widget, which includes various widgets to display and edit user information.

User information is retrieved from the getUser function of the UpdateNotify provider:

```dart
Future<void> getUser(bool initTextController) async {
    userLoaded = true;
    uid = await HelpersFunctions().getUserIdUserSharedPreference();
    UserModel user = await DatabaseServices(uid).getUserInfo();
    gender = user.gender;
    datingPurpose = user.datingPurpose;
    photoList = user.photoList;
    interestsList = user.interestsList;
    fluentLanguageList = user.fluentLanguageList;
    sexualOrientationList = user.sexualOrientationList;

    //BasicInfoUser
    zodiac = user.zodiac!;
    academicLever = user.academicLever!;
    communicateStyle = user.communicateStyle!;
    languageOfLove = user.languageOfLove!;
    familyStyle = user.familyStyle!;
    personalityType = user.personalityType!;

    //StyleOfLifeUser
    myPet = user.myPet!;
    drinkingStatus = user.drinkingStatus!;
    smokingStatus = user.smokingStatus!;
    sportsStatus = user.sportsStatus!;
    eatingStatus = user.eatingStatus!;
    socialNetworkStatus = user.socialNetworkStatus!;
    sleepingHabits = user.sleepingHabits!;
    if (initTextController) {
      introduceYourselfController.text = user.introduceYourself!;
      companyController.text = user.company!;
      schoolController.text = user.school!;
      currentAddressController.text = user.currentAddress!;
    }
    isLoading = false;
    isInterestSearching = false;
    await fetchLanguages();
    notifyListeners();
  }
```

The information is stored in variables within UpdateNotify and is utilized on the UpdateProfile screen.

Widget to update a list of images:

```dart

UpdateImage(
  itemWidth: itemWidth,
  itemHeight: itemHeight,
),

```

```dart
class UpdateImage extends StatelessWidget {
  const UpdateImage({
    super.key,
    required this.itemWidth,
    required this.itemHeight,
  });
  final double itemWidth;
  final double itemHeight;
  @override
  Widget build(BuildContext context) {
    List<String> photoList = context.watch<UpdateNotify>().photoList!;
    return Container(
      color: Colors.grey.shade200,
      padding: const EdgeInsets.symmetric(horizontal: 16.0, vertical: 8.0),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          GridView.builder(
            physics: const NeverScrollableScrollPhysics(),
            shrinkWrap: true,
            gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
              crossAxisCount: 3,
              mainAxisSpacing: 5,
              crossAxisSpacing: 5,
              childAspectRatio: itemWidth / itemHeight,
            ), // SliverGridDelegateWithFixedCrossAxisCount
            itemCount: 9,
            itemBuilder: (BuildContext context, int index) {
              if (index < photoList.length) {
                return Stack(children: [
                  ClipRRect( // ClipRRect ···
                  Positioned( // Positioned ···
                ]); // Stack
              } else {
                return InkWell(
                  onTap: () {
                    Provider.of<UpdateNotify>(context, listen: false)
                        .pickImages();
                  },
                  child: ClipRRect( // ClipRRect ···
                ); // InkWell
              }
            },
          ), // GridView.builder
          const SizedBox( // SizedBox ···
          Align( ···
              child: InkWell( // InkWell // Align ···
        ],
      ), // Column
    ); // Container
  }
}
```

The Widget UpdateImage retrieves a list of images from the photoList variable of the UpdateNotify provider and displays them in a GridView, limited to 9 items. For items that already have an image, the image will be shown. However, for elements without an image, a

replacement widget will be displayed instead. When the user taps on this replacement widget, it will call the pickImages function of UpdateNotify to select images.

The functions pickImages and _cropImage:

```dart
Future<void> pickImages() async {
    List<XFile>? resultList = await ImagePicker().pickMultiImage();
    List<File> selectedImages =
        resultList.map((xFile) => File(xFile.path)).toList();
    if (photoList!.length + selectedImages.length <= 9) {
      for (var imageFile in selectedImages) {
        await _cropImage(imageFile: imageFile);
      }
      notifyListeners();
    } else {
      // Xử lý khi vượt quá số lượng ảnh tối đa
    }
  }

  Future<void> _cropImage({required File imageFile}) async {
    CroppedFile? croppedFile = await ImageCropper().cropImage(
      sourcePath: imageFile.path,
      aspectRatioPresets: [
        CropAspectRatioPreset.square,
        CropAspectRatioPreset.ratio3x2,
        CropAspectRatioPreset.original,
        CropAspectRatioPreset.ratio4x3,
        CropAspectRatioPreset.ratio16x9
      ],
      compressQuality: 100,
      maxWidth: 500,
      maxHeight: 500,
      uiSettings: [
        AndroidUiSettings(
          toolbarTitle: 'Cắt ảnh',
          toolbarColor: Colors.blue,
          toolbarWidgetColor: Colors.white,
          statusBarColor: Colors.blue,
          backgroundColor: Colors.white,
        ),
        IOSUiSettings(
          title: 'Cắt ảnh',
        ),
      ],
    );

    if (croppedFile != null) {
      File? croppedImage = File(croppedFile.path);
      if (croppedImage.existsSync()) {
        String fileUrl = await DatabaseMethods().pushImage(croppedImage, uid!);
        photoList!.add(fileUrl);
        notifyListeners();
        updatePhotoList();
      }
    }
  }
```

The "ImagePicker" library is used to allow users to select images from their device's memory or gallery. After selecting an image, the "ImageCropper" library comes into play, allowing users to edit or crop the selected image as desired.

After selecting and editing, the data will be passed to the UpdatePhotoList function to save the information to Firestore. The function will handle the process of storing the edited image data in the Firestore database.

```
void updatePhotoList() {
    FirebaseFirestore.instance
        .collection("users")
        .doc(uid)
        .update({"photoList": photoList});
  }
```

Next up are the widgets for editing descriptions, company names, school names, and addresses. These widgets use TextFields for data entry:

```

Container(
  padding: const EdgeInsets.only(
      right: 16.0, left: 16.0, bottom: 16.0),
  color: Colors.white,
  child: Column(
    children: [
      TextField(
        controller:
            updateProvider.introduceYourselfController,
        keyboardType: TextInputType.multiline,
        maxLength: 500,
        textInputAction: TextInputAction.newline,
        maxLines: null,
        decoration: InputDecoration(
          contentPadding:
              EdgeInsets.symmetric(vertical: 16.0),
          border: InputBorder.none,
          hintText: appLocal.updateProfileAboutMeText,
        ),
        style: const TextStyle(
            fontSize: 16,
            color: Colors.black,
            fontWeight: FontWeight.normal),
      ),
    ],
  ),
),
```

A TextField has a controller that is managed by a provider. The information entered will be saved when the user exits the update screen by calling a function in the provider:

```dart
Future<void> updateInputField() async {
  userLoaded = false;
  await FirebaseFirestore.instance.collection("users").doc(uid).update({
    "introduceYourself": introduceYourselfController.text,
    "company": companyController.text,
    "school": schoolController.text,
    "currentAddress": currentAddressController.text,
  });
  notifyListeners();
}
```

The remaining widgets follow a similar logic. They will display user information, and when the user taps on them, a BottomSheet will open for editing. After the editing is completed, the updated information will be saved using functions provided by the Provider.

Example, widget of gender information:

```
InkWell(
  splashColor: const Color.fromRGBO(229, 58, 69, 100),
  onTap: () {
    showModalBottomSheet(
      isScrollControlled: true,
      isDismissible: true,
      useSafeArea: true,
      context: context,
      builder: (context) {
        return const GenderBottomSheet();
      },
    ).whenComplete(() {
      updateProvider.getUser(false);
    });
  },
  child: Container(
    width: MediaQuery.of(context).size.width,
    padding: const EdgeInsets.all(16.0),
    decoration: const BoxDecoration(),
    child: Text(
      HelpersUserAndValidators.getItemFromIndex(
          context,
          HelpersUserAndValidators.genderList(context),
          updateProvider.gender!),
      style: const TextStyle(
          fontSize: 16,
          color: Colors.black,
          fontWeight: FontWeight.normal),
    ),
  ),
),
```

Save Gender Function:

```
Future<void> updateGender() async {
    await FirebaseFirestore.instance.collection("users").doc(uid).update({
      "gender": gender,
    });
  }
```

Detailed information about the remaining widgets and BottomSheet can be found at the link below:

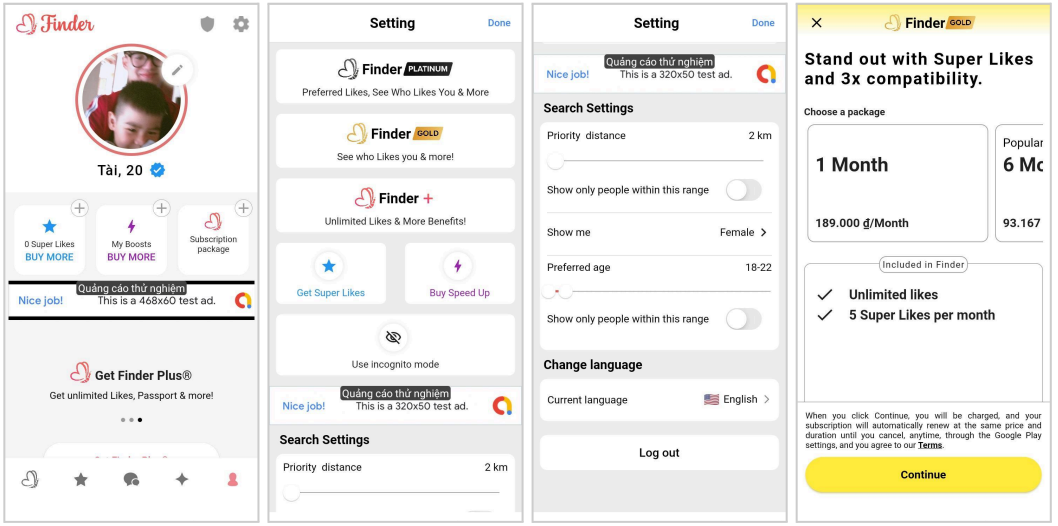
- , Main screen, profile update, and sub-widgets.

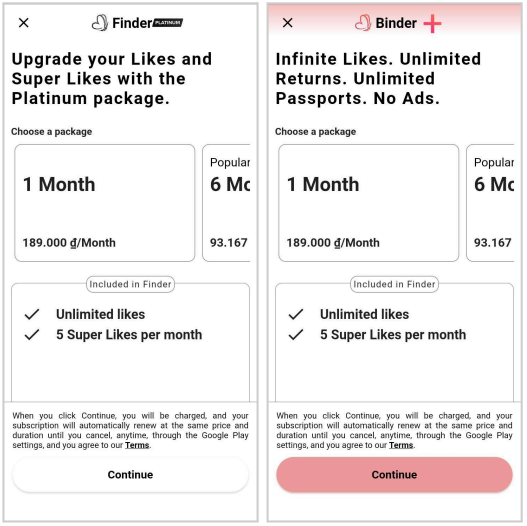
-, Provider and data connection management.

3.7, The settings screen and advanced paid pages.

- This functionality includes the general app settings interface such as search settings, user filtering, language preferences, and logout option.

- Screenshots





- Related code items.(In the link attached to each title) :
  a) The main settings screen with an avatar and navigation buttons, as well as advertisements.

- , Main Screen

```dart
class ProfileScreen extends StatelessWidget {
  const ProfileScreen({Key? key}) : super(key: key);
  void _showBottomModal(BuildContext context) {
    showModalBottomSheet(
      context: context,
      isScrollControlled: true,
      builder: (BuildContext context) {
        return const SettingScreen();
      },
    );
  }
  @override
  Widget build(BuildContext context) {
    Provider.of<ProfileWatch>(context, listen: false).getUser();
    return SafeArea(
      child: Scaffold(
        appBar: AppBar( // AppBar ···
        backgroundColor: Colors.white,
        body: getBody(context),
      ), // Scaffold
    ); // SafeArea
  }
  Widget getBody(BuildContext context) {
    return StreamBuilder<UserModel>(
        stream: context.watch<ProfileWatch>().getUserStream(),
        builder: (context, snapshot) {
          if (snapshot.hasData) {
            String fullName = snapshot.data!.fullName;
            List<String> splitName = fullName.split(" ");
            return SingleChildScrollView(

              child: Column( // Column ···
            ); // SingleChildScrollView
          } else {
            return Container();
          }
        }); // StreamBuilder
  }
}
```

The ProfileScreen contains a primary content that is a StreamBuilder listening to a stream obtained from the context.watch<ProfileWatch>().getUserStream() function. It holds information about the current user to display their name, age, and avatar in the ProfileScreen. Additionally, the screen includes several UI elements for premium features, which you can find more details about in the link above.

The widget displaying the avatar is interactive and triggers a navigation event to switch to the UpdateProfileScreen when tapped.

b) The SettingScreen includes various functionalities as described:

```dart
class SettingScreen extends StatelessWidget {
  const SettingScreen({super.key});

  @override
  Widget build(BuildContext context) {
    final provider = context.read<BinderWatch>();
    final padding =  MediaQuery.of(context).padding;
    return  Container(
      color:  Colors.white,
      padding: EdgeInsets.only(top: 30),
      child: Scaffold(
          appBar: AppBar(
            elevation: 0.3,
            backgroundColor: Colors.white,
            title: Center(
              child: Text(
                AppLocalizations.of(context).settingPageSubTitleAppBar,
                style: TextStyle(color: Colors.black, fontWeight: FontWeight.bold),
              ),
            ),
          actions: [
            TextButton(
                onPressed:() async{
                  await provider.updateRequestToShow();
                  context.pop();} ,
                child: Text(
                  AppLocalizations.of(context).settingPageSubDoneText,
                  style: TextStyle(color: Colors.blueAccent),
                ))
          ],
          automaticallyImplyLeading: false,
          leading: Text(""),
        ),
        backgroundColor: Colors.grey[200],
        body: Body()

      ),
    );
  }
}
```

- , The main settings screen

- , The data for changing the language.

-, The providers and data connections manage the settings and logout functionalities.

V. GUIDE TO APP INTERFACE CUSTOMIZATION (RESKIN APP)

1 . Change the app's background color.

1.1 Greetings:

Change background color:

- Open the file named "welcome.dart".
- Find the "build" widget as shown in the image below.
- Modify the LinearGradient in BoxDecoration with the desired color.

```dart
@override
 Widget build(BuildContext context) {
   final checkLogin = context.watch<CallDataProvider>();
   return StreamBuilder<User?>(
     stream: checkLogin.user,
     builder: (context, AsyncSnapshot<User?> snapshot) {
       Timer(Duration(seconds: 3), () {
         snapshot.data == null ? context.go('/login-home-screen') : context.go('/home');
       });
       if (snapshot.connectionState == ConnectionState.active) {
         return Scaffold(
           body: Container(
             decoration: BoxDecoration(
               gradient: const LinearGradient(
                   begin: Alignment.topCenter,
                   end: Alignment.bottomCenter,
                   colors: [ const Color.fromRGBO(238, 128, 95, 1) ,const Color.fromRGBO(234, 64, 128, 1)],),
             ),
```

1.2 Login screen:

Change background color:

- Open the file named "login_home_screen.dart".
- Find the "build" widget as shown in the image below.
- Modify the LinearGradient in BoxDecoration with the desired color.

```dart
@override
 Widget build(BuildContext context) {
   final appLocal = AppLocalizations.of(context);
   final adProvider = Provider.of<AdMobProvider>(context);

   return Scaffold(
     body: isLoading
         ? Center(
       child: LoadingAnimationWidget.threeArchedCircle(
         color: Color.fromRGBO(234, 64, 128, 1),
         size: 100,
       ),
     )
         : Container(
       decoration: BoxDecoration(
         gradient: const LinearGradient(
           begin: Alignment.topCenter,
           end: Alignment.bottomCenter,
           colors: [
             const Color.fromRGBO(238, 128, 95, 1),
             const Color.fromRGBO(234, 64, 128, 1)
           ],
         ),
       ),
```

1.3 Change the color of the Bottom Navigation:

- Open the file named "home.dart".
- Find the widget named "BottomNavigationBar" as shown in the image below.
- Change the "backgroundColor" to the color you want.

```dart
BottomNavigationBar _bottomNavigation() {
    return BottomNavigationBar(
      backgroundColor: Colors.white,
      elevation: 2,
```

1.4 Swipe user screen:

Change the color of the AppBar:

- Open the file named "binder_page.dart".
- Find the "build" widget as shown in the image below.
- Change the "backgroundColor" in AppBar to the color you want.

```dart
@override
  Widget build(BuildContext context) {
    final provider = context.read<BinderWatch>();
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.white,
        elevation: 0,
```

Change the background color while loading data:

- Open the file named "binder_page.dart".
- Find the "build" widget as shown in the image below.
- Change the "backgroundColor" in Scaffold to the color you want.

```dart
@override
Widget build(BuildContext context) {
  final provider = context.read<BinderWatch>();
  return Scaffold(
    appBar: AppBar(...),  // AppBar
    backgroundColor: Colors.white,
    body: getBody(context, provider.selectedOption, provider.currentAgeValue,
        provider.showPeopleInRangeDistance, provider.distancePreference),
  );  // Scaffold
}
```

Change the background color after having data:

- Open the file named "binder_page.dart".

- Find the "getBody" widget as shown in the image below.
- Change the "color" in BoxShadow to the color you want.

```dart
Widget getBody(BuildContext context, int gender, List<double> age,
      bool isInDistanceRange, double kilometres) {
    return FutureBuilder(
      future: context.read<BinderWatch>().allUserBinder(context,gender, age, isInDistanceRange, kilometres),
      builder: (context, snapshot) => snapshot.hasData
          ? Container(
              padding: const EdgeInsets.all(10),
              decoration: BoxDecoration(
                boxShadow: [
                  BoxShadow(
                    color: Colors.grey.shade200,
                    spreadRadius: 3,
                    blurRadius: 3,
                    offset: Offset(0, 3),
                  ),
                ],
```

1.5 Top selection screens:

Change the color of the AppBar:

- Open the file named "binder_selection.dart".
- Find the "build" widget as shown in the image below.
- Change the "backgroundColor" in AppBar to the color you want.

```dart
@override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.white,
      appBar: AppBar(
        backgroundColor: Colors.white,
        elevation: 0,
```

Change the background color:

- Open the file named "body.dart".
- Find the "build" widget as shown in the image below.
- Change the "backgroundColor" in Scaffold to the color you want.

```dart
@override
  Widget build(BuildContext context) {
    _scrollController.addListener(() => _scrollListener(context));
    final appLocal = AppLocalizations.of(context);
    return Scaffold(
      backgroundColor: Colors.white,
      body: Stack(
```

1.6 Chat list screen:

Change the color of the AppBar:

- Open the file named "message_screen.dart".
- Find the "build" widget as shown in the image below.
- Change the "backgroundColor" in AppBar to the color you want.

```dart
@override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.transparent,
        elevation: 0,
```

Change the background color:

- Open the file named "message_screen.dart".
- Find the "_buildBody" widget as shown in the image below.
- Change the "color" in Container to the color you want.

```dart
_buildBody(BuildContext context) {
  final adProvider = Provider.of<AdMobProvider>(context,listen: false);
  adProvider.loadBannerAd(context);
  return BlocBuilder<MessageBloc, MessageState>(
    builder: (context, state) {
      if (state is ChatRoomsLoading) {
        return const Center(child: CircularProgressIndicator());
      }
      if (state is ChatRoomsLoaded) {
        return SingleChildScrollView(
          child: Container(
            color: Colors.transparent,
            child: Column(
```

1.7 Chat screen:

Change the color of the AppBar:

- Open the file named "detail_message.dart".
- Find the "build" widget as shown in the image below.
- Change the "backgroundColor" in AppBar to the color you want.

```dart
@override
Widget build(BuildContext context) {
  getKeyUid();
  return BlocConsumer<DetailMessageBloc, DetailMessageState>(
    listener: (context, state) {},
    builder: (context, state) {
      return Scaffold(
        appBar: AppBar(
          titleSpacing: 0,
          elevation: 1,
          backgroundColor: Colors.white,
```

Change the background color:

- Open the file named "detail_message.dart".
- Find the "build" widget as shown in the image below.
- Change the "color" in BoxDecoration to the color you want.

```dart
@override
Widget build(BuildContext context) {
  getKeyUid();
  return BlocConsumer<DetailMessageBloc, DetailMessageState>(
    listener: (context, state) {},
    builder: (context, state) {
      return Scaffold(
        appBar: AppBar(...),  // AppBar
        backgroundColor: Colors.transparent,
        body: Container(
          padding: const EdgeInsets.symmetric(horizontal: 8),
          decoration: const BoxDecoration(color: Colors.blue),
          child: Builder(builder: (context) {
            if (state is MessageListLoaded) {...}
            return const SizedBox.shrink();
          }),  // Builder
        ),  // Container
      );  // Scaffold
    },
  );  // BlocConsumer
```

- Then find the "listMessage" widget and change the color of the container to transparent.

```
Widget listMessage(Stream<List<ChatMessageEntity>> messageListStream,
    Stream<ChatRoomEntity> chatRoom, String watchTime12345) {
  DateFormat timeFormat = DateFormat('HH:mm a');
  DateFormat dateFormat = DateFormat('MMM dd, HH:mm');
  String userTime = '';
  bool checkTime(DateTime dateTime) {...}

  bool checkDuration(DateTime dateTime1, DateTime dateTime2) {...}

  return StreamBuilder(
      stream: chatRoom,
      builder: (context, snapshot) {
        if (snapshot.hasData) {
          var userTimes = snapshot.data?.userTimes as List<dynamic>;
          for (var index in userTimes) {
            if (index.uid == widget.uid) {
              userTime = index.time!.toString();
            }
          }
        }
      return StreamBuilder(
          stream: messageListStream,
          builder: (context, snapshot) {
            BlocProvider.of<DetailMessageBloc>(context)
                .add(CompareUserTime(keyUid, widget.chatRoomId!));
            return snapshot.hasData
            ? ListView.builder(
                itemCount: snapshot.data?.length,
                shrinkWrap: true,
                scrollDirection: Axis.vertical,
                reverse: true,
                padding: const EdgeInsets.only(top: 10),
                physics: const ScrollPhysics(),
                itemBuilder: (context, index) {
                  DateTime time = DateTime.parse(
                      snapshot.data?[index].time.toString() ?? "");
                return Container(
                  color: Colors.white,
                  padding: const EdgeInsets.only(
```

1.8 See who likes you screen:

Change the color of the AppBar:

- Open the file named "who_like_page.dart".
- Find the "build" widget as shown in the image below.
- Change the "backgroundColor" in AppBar to the color you want.

Change the background color:

- Open the file named "who_like_page.dart".
- Find the "build" widget as shown in the image below.
- Change the "backgroundColor" in Scaffold to the color you want.

```
@override
Widget build(BuildContext context) {
  final appLocal = AppLocalizations.of(context);

  return Scaffold(
    appBar: AppBar(
      backgroundColor: Colors.white,
      elevation: 0,
      title: Align(...),  // Align
    ),  // AppBar
    backgroundColor: Colors.white,
    body: SingleChildScrollView(...),  // SingleChildScrollVie
  );  // Scaffold
}
```

1.9 Profile screens:

Change the color of the AppBar:

- Open the file named "profile.dart".
- Find the "build" widget as shown in the image below.
- Change the "backgroundColor" in AppBar to the color you want.

Change the background color:

- Open the file named "profile.dart".
- Find the "build" widget as shown in the image below.
- Change the "backgroundColor" in Scaffold to the color you want.

```dart
@override
Widget build(BuildContext context) {
  Provider.of<ProfileWatch>(context, listen: false).getUser();
  return SafeArea(
    child: Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.white,
        elevation: 0,
        title: Row(...),  // Row
        actions: [
          IconButton(...),  // Image.asset, IconButton
          IconButton(...),  // Image.asset, IconButton
        ],
      ),  // AppBar
      backgroundColor: Colors.white,
      body: getBody(context),
    ),  // Scaffold
  );  // SafeArea
}
```

- Then, open the file "BodyBuyPremium.dart".
- Find the "build" widget as shown in the image below.
- Change the "color" in Container to the color you want.

```dart
@override
 Widget build(BuildContext context) {
   final appLocal = AppLocalizations.of(context);
   final adProvider = Provider.of<AdMobProvider>(context,listen: false);
   adProvider.loadBannerAd(context);

   return Container(
     padding: const EdgeInsets.only(top: 10, left: 5, right: 5, bottom: 70),
     color: Colors.grey[100],
```

1.10 Settings screen:

Change the color of the AppBar:

- Open the file named "setting_screen.dart".

- Find the "build" widget as shown in the image below.
- Change the "backgroundColor" in AppBar to the color you want.

Change the background color:

- Open the file named "setting_screen.dart".
- Find the "build" widget as shown in the image below.
- Change the "backgroundColor" in Scaffold to the color you want.

```
@override
Widget build(BuildContext context) {
  final provider = context.read<BinderWatch>();
  final padding =  MediaQuery.of(context).padding;
  return  Container(
    color:  Colors.white,
    padding: EdgeInsets.only(top: 30),
    child: Scaffold(
      appBar: AppBar(
        elevation: 0.3,
        backgroundColor: Colors.white,
        title: Center(...),  // Center
        actions: [...],
        automaticallyImplyLeading: false,
        leading: Text(""),
      ),  // AppBar
      backgroundColor: Colors.grey[200],
      body: Body()
    ),  // Scaffold
  );  // Container
}
```

1.11 Notification screens:

Change the color of the AppBar:

- Open the file named "notification_screen.dart".
- Find the "build" widget as shown in the image below.
- Change the "backgroundColor" in AppBar to the color you want.

Change the background color:

- Open the file named "notification_screen.dart".
- Find the "build" widget as shown in the image below.
- Change the "backgroundColor" in Scaffold to the color you want.

```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      elevation: 1,
      backgroundColor: Colors.white,
      title: Text(...),  // Text
      leading: IconButton(
          onPressed: () => context.pop(),
          icon: const Icon(Icons.arrow_back_outlined, color: Colors.red,)),
    ),  // AppBar
      backgroundColor: Colors.white,
      primary: true,
```

1.12 Other user profile screen:

Change the background color:

- Open the file named "detail_profile_others.dart".
- Find the "build" widget as shown in the image below.
- Change the "backgroundColor" in Scaffold to the color you want.

```dart
@override
  Widget build(BuildContext context) {
    final appLocal = AppLocalizations.of(context);
    final adProvider = Provider.of<AdMobProvider>(context,listen: false);
    adProvider.loadBannerAd(context);

    final EdgeInsets padding = MediaQuery.of(context).padding;
    return Scaffold(
      backgroundColor: Colors.white,
      extendBody: true,
```

1.13 Enter personal information screen:

Change the background color:

- Open the files in the pageConfirm directory, such as add_birthday_page.dart.
- Find the "build" widget as shown in the image below.

- Add or modify the "backgroundColor" property in Scaffold to the color you want.

```
@override
  Widget build(BuildContext context) {
    final pageProvider = Provider.of<PageDataConfirmProfileProvider>(context);
    final appLocal = AppLocalizations.of(context);

    return Scaffold(
      backgroundColor: Colors.white,
      extendBody: true,
```

1.14 Search settings screen:

Change the color of the AppBar:

- Open the file named "discovery_setting.dart".
- Find the "build" widget as shown in the image below.
- Change the "backgroundColor" in AppBar to the color you want.

Change the background color:

- Open the file named "discovery_setting.dart".
- Find the "build" widget as shown in the image below.
- Change the "backgroundColor" in Scaffold to the color you want.

```dart
@override
  Widget build(BuildContext context) {
    final provider = context.read<BinderWatch>();
    final appLocal = AppLocalizations.of(context);
    final adProvider = Provider.of<AdMobProvider>(context,listen: false);
    adProvider.loadBannerAd(context);
    return  Scaffold(
          appBar: AppBar(
            backgroundColor: Colors.white,
            elevation: 0.4,
            title: Center(
              child: Text(
                appLocal.settingPageSearchText,
                style: TextStyle(color: Colors.black),
              ),
            ),
            actions: [
              TextButton(
                onPressed: () async {
                  await provider.updateRequestToShow();
                  context.pop("refresh");
                },
                child: Text(
                  appLocal.discoverySettingConfirmText,
                  style: TextStyle(color: Colors.blue[700]),
                ),
              ),
            ],
          ),
          backgroundColor: Colors.grey[200],
```

1.15 Tinder Premium screen:

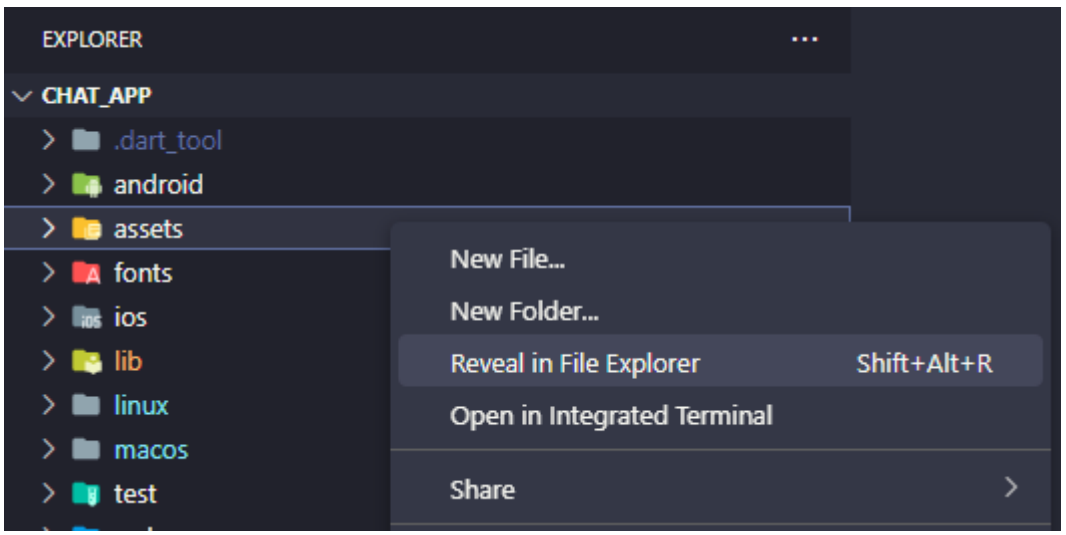
Change the background color:

- Open the file named "bottom_modal_fullscreen.dart".
- Find the "build" widget as shown in the image below.
- Modify the LinearGradient in BoxDecoration with the desired color.

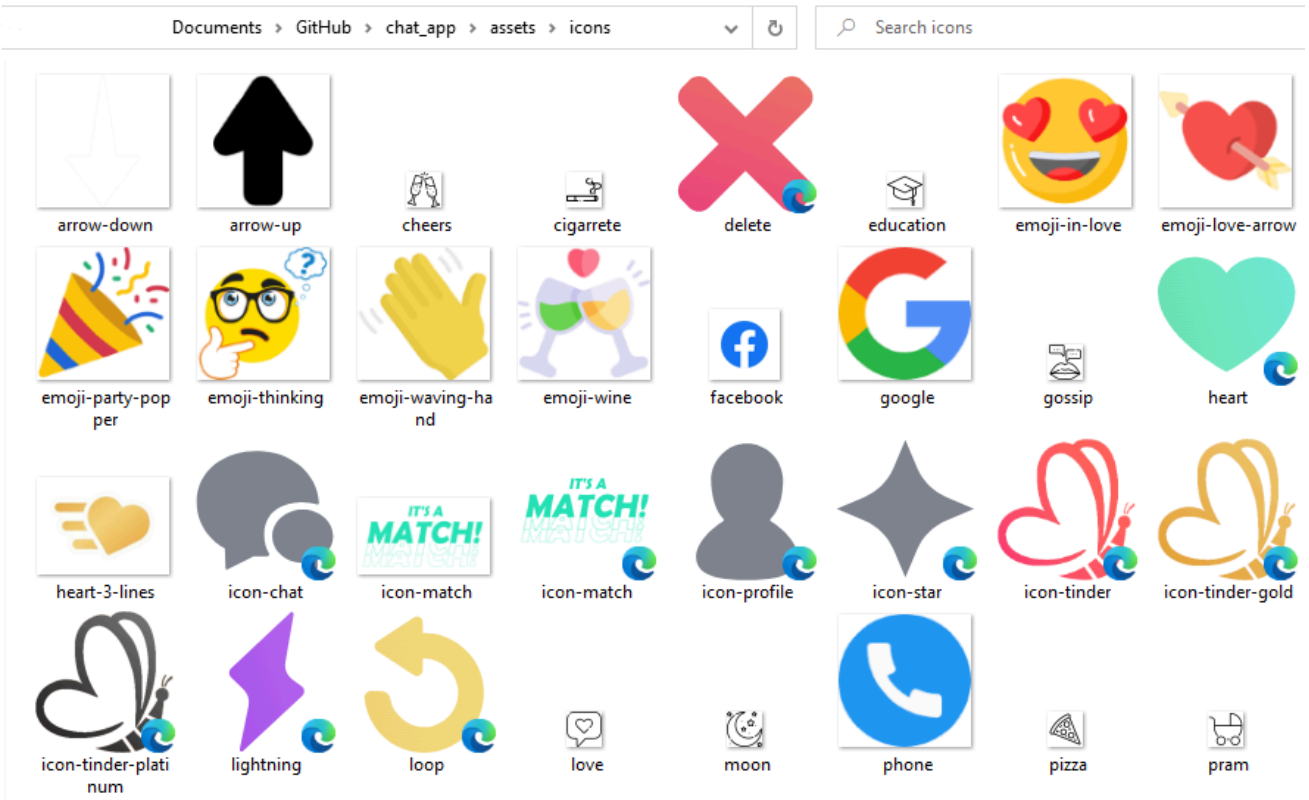
```dart
@override
 Widget build(BuildContext context)

 {
   final height = MediaQuery.of(context).size.height;
   final styleTextSpan = TextStyle(color: Colors.black, fontSize: 12);
   final appLocal = AppLocalizations.of(context);
   return Container(
     padding: EdgeInsets.only(top: 30),
     decoration: isHaveColor
         ? BoxDecoration(
       gradient: LinearGradient(
         begin: Alignment.topCenter,
         end: Alignment.bottomCenter,
         colors: [color, Colors.white],
         stops: [0.01, 0.1],
       ),
     )
```

2 . CHANGE THE APP'S ICON.

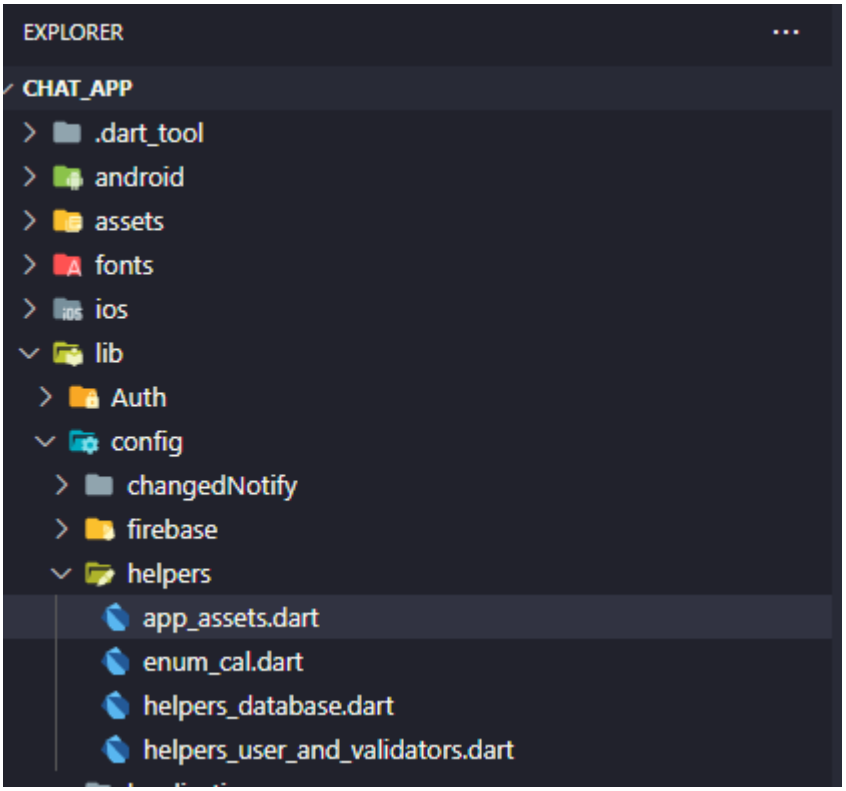
Open the “**assets”** folder from the root directory of your project, then right- click and select “**Reveal in File Explorer**”



At this point, your computer will automatically open that folder, and what you need to do is click on the "**assets**" folder, then continue clicking on the "**icons**" folder. This is where all the resources of the application are located



You can modify resources as you wish. After that, open the Integrated Development Environment (IDE) and find the file "**app_assets.dart**" located in the folder *lib\config\helpers*



After that, configure the resources for your application. Then, rebuild the application.

// Example of configuring a static resource constant:

// static const String iconExample = '$iconPath + your-resource-icon.png'

```dart
class AppAssets {
  static const String iconPath = 'assets/icons/';

  //example
  // static const String iconExample = '$iconPath+your-resource-icon.png';

  static const String iconTinder = iconPath + 'icon-tinder.svg';
  static const String iconTinderGold = iconPath + 'icon-tinder-gold.svg';
  static const String iconTinderGoldBanner =iconPath + 'tinder-gold-banner.svg';
  static const String iconTinderPlatinum =  iconPath + 'icon-tinder-platinum.svg';
  static const String iconTinderPlatinumBanner =  iconPath + 'tinder-platinum-banner.svg';
  static const String iconTinderPlusBanner =  iconPath + 'tinder-plus-banner.svg';
  static const String iconChatBottomBar = iconPath + 'icon-chat.svg';
  static const String iconLoveBottomBar = iconPath + 'icon-star.svg';
  static const String iconProfileBottomBar = iconPath + 'icon-profile.svg';
  static const String iconMatch = iconPath + 'icon-match.png';

  static const String iconGG = iconPath + 'google.png';
  static const String iconShield = iconPath + 'shield.png';
  static const String iconSetting = iconPath + 'settings.png';
  static const String iconPhone = iconPath + 'phone.png'
```

If you want to use the icon in a particular file, navigate to that file and write ***Appassets.iconExample*** to reference the iconExample constant from the Appassets class.

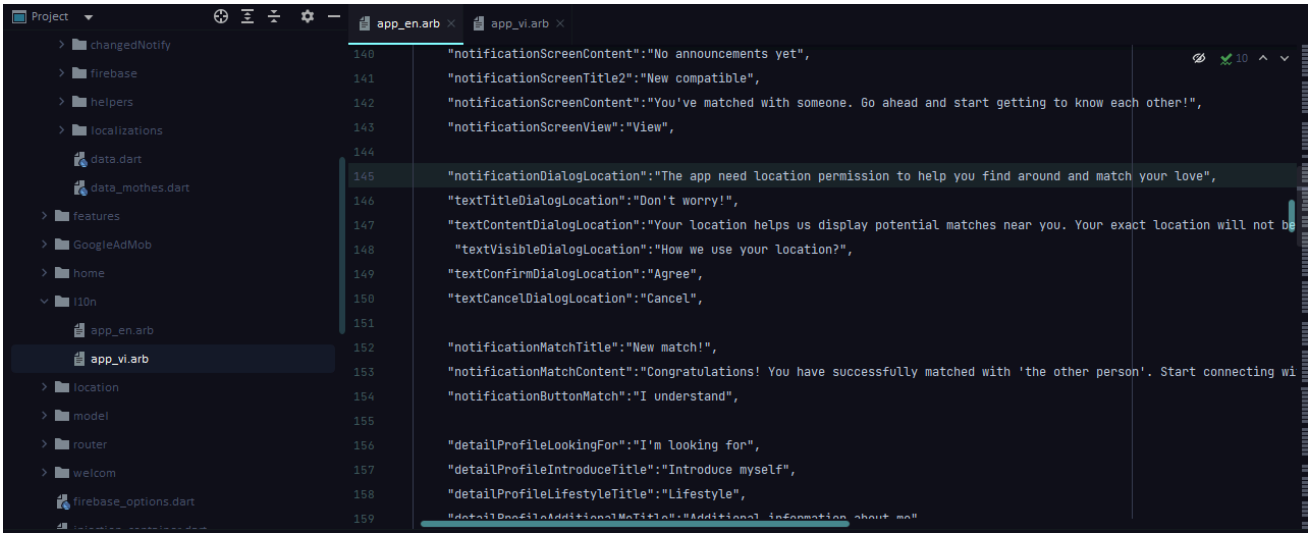
```dart
SvgPicture.asset(
//change resource here
  AppAssets.iconExample,
  width: 120,
  height: 120,
  fit: BoxFit.cover,
  colorFilter: ColorFilter.mode(Colors.white, BlendMode.srcIn),
),
```

Change icon in welcome.dart

If you want to add image resources, you can do the same as you did for icons.

3 . Changing the app's language content

- When you want to add a new word or add a new language in the app, you can use f***lutter localizations***, which can be further explored ***here.***
- After reading and understanding how to handle multiple languages, you can add, modify, or delete new phrases or words here:



where the files in the l10n directory represent different languages customized with the ***'key':'value'*** pairs. The **key** must match accurately across all language files.

- After adding the language values, run the command ***'flutter gen-l10n'*** in the project's terminal to synchronize the values and use them. In the main.dart function, you need to initialize as follows to handle multiple languages.

```dart
class _MyAppState extends State<MyApp> {
  Locale? _locale;
  setLocale(Locale locale) {
    setState(() {
      _locale = locale;
    });
  }
  @override
  void didChangeDependencies() {
    getLocale().then((locale) => {setLocale(local
e)});
    super.didChangeDependencies();
  }
  @override
  Widget build(BuildContext context) {
    return MaterialApp.router(
      title: 'Finder',
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      locale: _locale,
      localizationsDelegates: AppLocalizations.loc
alizationsDelegates, // Add this line,
      supportedLocales: AppLocalizations.supported
Locales,
      routerConfig: router,
    );
  }
}
```

- Usage is as follows:
    - In the file where you want to use a different language, import ***'package:flutter_gen/gen_l10n/app_localizations.dart';*** then initialize **'final appLocal = AppLocalizations.of(context);'** inside the build method of each Widget.

- Finally, to access the text values created above, you just need to use **'appLocal.key'** to initialize the multilingual support for the app when switching between languages.
- When you want to use the language switching feature, you can do it as follows:

```
DropdownButton<Language>(
                  underline: const SizedBox(),
                  icon:  const Icon(Icons.language,color: Colors.white,size: 30, ),
                  onChanged:(Language? language) async {
                    if (language != null) {
                      Locale _locale = await setLocale(language.languageCode);
                      MyApp.setLocale(context, _locale);
                    }
                  },
                  items: Language.languageList().map<DropdownMenuItem<Language>>(
                          (e) => DropdownMenuItem<Language>(
                          value: e,
                          child: Row(
                            mainAxisAlignment: MainAxisAlignment.start,
                            children: <Widget>[
                              Text( e.flag,style: const TextStyle(fontSize: 30),),
                              const SizedBox(width: 5,),
                              Text(e.name)
                            ],
                          )
                      )
                  ).toList(),
                )
```