Mobile Apps
Using any Mobile App (Android or iOS, however preference is on iOS) you have developed (or are developing) as a reference, answer the following questions in relation to the Mobile Apps.

Provide a brief description of the Mobile App you developed (or, are developing). Please include the name of the App and where it can currently be accessed from (if publically accessible/operational).

1. **App Name**: “Image Vault”
2. **Functionality Overview**:
    - **User Authentication**:
        - Users can sign up with their credentials (email/password or other authentication methods).
        - Firebase handles user authentication, ensuring that each user has a unique account.
        - Once signed up, users can log in securely.
    - **Image Management**:
        - Users can upload images to the app.
        - The app stores these images securely using Firebase Cloud Storage.
        - Users can organize their images by moving them around within the app.
        - Deletion functionality allows users to remove images they no longer need.
    - **Security Measures**:
        - Firebase Authentication ensures that users can’t sign up more than once.
        - Incorrect login credentials prevent unauthorized access.
    - **User Interface (UI)**:
        - The home page displays a list of uploaded images.
        - Users can interact with the images (e.g., view, move, delete) through intuitive UI elements.
3. **Firebase Integration**:
    - **Authentication**:
        - Firebase Authentication handles user sign-up, login, and session management.
        - It provides secure token-based authentication.
    - **Cloud Storage**:
        - Firebase Cloud Storage stores the uploaded images.

        - Images are accessible only to the authenticated user.
        - Security rules can be set to restrict access to specific users.
    - **Database**:
        - Firebase Realtime Database or Firestore is used to store additional metadata about the images (e.g., image titles, descriptions, timestamps).

What challenges is a developer likely to encounter during the development for:

a. Android devices

Fragmentation: Android has an extensive collection of devices with different screen sizes, resolutions, and operating system versions12. This can make it challenging to develop an app that works seamlessly on all Android devices.

Performance Optimization: Android apps often face performance issues due to memory leaks, inefficient resource usage, and slow network connections.

User Interface (UI) Design: Designing UIs that work well across different screen sizes and resolutions can take time and effort. Developers need to follow Android's UI design guidelines and use responsive layout techniques.

b. iOS devices:

App Store Approval: The App Store is known for its strict review process. Apps might be rejected during the App Store review process for various reasons, such as failing to meet App Store Review Guidelines, app performance issues, or incomplete information3.

Device Compatibility: Ensuring your app functions flawlessly across a wide range of Apple devices can be challenging. Apps need to adapt their layout and UI elements to fit and display correctly on various screen sizes and resolutions3.

Did you develop your app as a native, web or hybrid application?

Native application

What guided your decision to develop your app as native, web or hybrid in "2" above

Performance: Native apps are faster and more efficient as they work in tandem with the mobile device's operating system. Since they are designed for a specific platform, they can operate more quickly and effectively.
Leveraging device capabilities: Native apps can directly access the hardware of the device such as the GPS, camera, microphone, etc. This can help in enhancing the user experience by making use of the device's full potential.
User Experience: Native apps provide a better user experience. They are more intuitive and easier to navigate because they follow specific UI standards for each platform.
Quality Assurance: Each app store has its own app review process that checks for quality and safety. This can act as an additional layer of quality assurance.

What tools, frameworks and, or languages did you use when developing your app in “2” above

> **Language**: Dart , **Framework**: Flutter, **Libraries**: firebase_core, firebase_auth, cloud_firestore, firebase_storage, geolocator, image_picker, google_maps_flutter
> **Tools**:Android Studio

What are your thoughts on the use of Flutter or React Native Frameworks as opposed to the use of Native code like Swift , Java or Kotlin

> Flutter uses Dart language which is easy to understand for JavaScript or Java developers. Giving it an advantage over the rest.

What are 4 of the most valuable lessons you have learned during the testing of your mobile app(s)

> **Thorough Testing on Different Devices**: Mobile apps can behave differently on different devices due to variations in screen sizes, resolutions, operating system versions, and hardware capabilities. It’s important to test your app on a range of devices to ensure a consistent user experience.
>
> **Importance of Automated Testing**: Automated testing can save a lot of time and effort. Unit tests, integration tests, and UI tests can help catch issues early and reduce the amount of manual testing required.
>
> **Real-world User Scenarios**: It’s crucial to test the app under real-world scenarios, including poor network conditions, interruptions (like calls, notifications), and low battery conditions. This can help identify how the app behaves under such conditions and improve its robustness.
>
> **Performance Testing**: Users expect apps to be fast and responsive. Performance testing is crucial to ensure your app runs smoothly, without crashes or slowdowns. This includes testing the app’s speed, responsiveness, and stability under a workload.

In Android development why would you choose Kotlin over Java

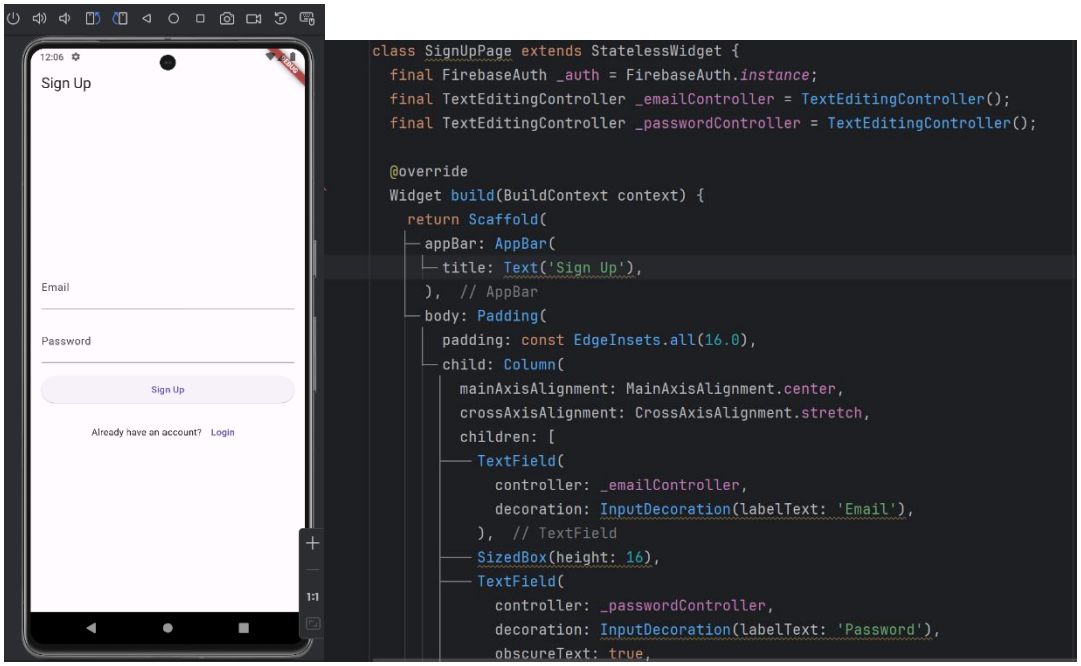
> **More Concise**: Kotlin has a more expressive syntax than Java, which leads to less boilerplate code. This makes the code easier to read and write

Between MVP and MVVM android patterns, which pattern would you prefer and why

> The choice between MVP (Model-View-Presenter) and MVVM (Model-View-ViewModel) patterns depends on the specific needs of the project.
>
> I would prefer MVP since i prefer straightforward one-to-one mapping between components and don’t plan to use data binding.

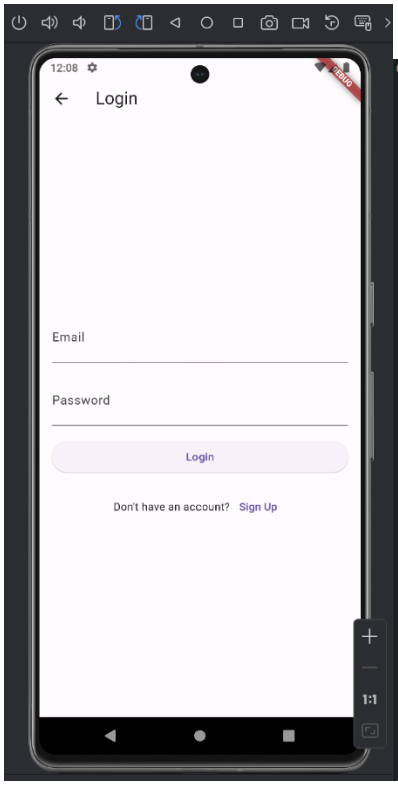
Provide at least 3 sample screenshots from your mobile app in “2” above, and for each screenshot, provide a brief explanation on what the interface does and a snippet of the code. Consider the illustration below as a guide

Signup Page



The SignUpPage class extends StatelessWidget, which means it describes part of the user interface which can depend on configuration information but doesn't hold mutable state. It uses Firebase Authentication (FirebaseAuth) to handle user registration. It has two TextEditingController objects to manage the text in the email and password fields. The build method returns a Scaffold widget, which provides a framework in which material design widgets can be arranged. It includes an AppBar with the title 'Sign Up' and a Column widget in the body. Inside the Column widget, there are two TextField widgets for the user to input their email and password, an ElevatedButton for the sign-up action, and a Row widget containing a Text and TextButton for users who already have an account to navigate to the login page. When the sign-up button is pressed, it first checks if the email and password fields are not empty. If they are, it shows a SnackBar with a message asking the user to fill in all fields. If the fields are not empty, it tries to create a new user account with the entered email and password using Firebase Authentication. If the user already exists or if there's any other error, it shows a SnackBar with the error message. If the account is created successfully, it navigates to the LoginPage.

Login Page.




```dart
class LoginPage extends StatelessWidget {
  final FirebaseAuth _auth = FirebaseAuth.instance;
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController = TextEditingController();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Login'),
      ),  // AppBar
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          crossAxisAlignment: CrossAxisAlignment.stretch,
          children: [
            TextField(
              controller: _emailController,
              decoration: InputDecoration(labelText: 'Email'),
            ),  // TextField
            SizedBox(height: 16),
            TextField(
              controller: _passwordController,
              decoration: InputDecoration(labelText: 'Password'),
              obscureText: true,
            ),  // TextField
            SizedBox(height: 16),
            ElevatedButton(
```

Users can enter their email and password, which are controlled by TextEditingControllers. Before attempting to log in, it checks if the email and password fields are not empty. If valid, it uses Firebase's signInWithEmailAndPassword method to authenticate the user. Upon successful login, it navigates to the HomePage. If the credentials are invalid, it displays an error message. If the user doesn't have an account, they can navigate to the SignUpPage to create one.

Home Page


```dart
class _HomePageState extends State<HomePage> {
  List<File> images = []; // Store uploaded image files

  Future<void> _pickImage() async {
    final pickedImage = await ImagePicker().pickImage(source: ImageSource.gallery);
    if (pickedImage != null) {
      setState(() {
        images.add(File(pickedImage.path));
      });
    }
  }

  void _deleteImage(int index) {
    setState(() {
      images.removeAt(index);
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Home'),
      ),  // AppBar
      body: ListView.builder(
        itemCount: images.length,
        itemBuilder: (context, index) {
```

Image Storage: It maintains a list of image files that users have uploaded.

Image Picker: Users can add images to the vault by selecting them from their device's gallery.

Image Display: The uploaded images are displayed in a scrollable list.

Image Deletion: Users can swipe to dismiss or press a delete icon to remove images from the vault.

SQL and Database

Which movie is preferred by most of the family members

```
SELECT PreferredMovie, COUNT(*) as PreferenceCount FROM PreferredMovies GROUP BY PreferredMovie ORDER BY PreferenceCount DESC LIMIT 1;
```

Which movie type (TypeOfMovie) is the least preferred within the family

```
SELECT TypeOfMovie, COUNT(*) as PreferenceCount FROM PreferredMovies GROUP BY TypeOfMovie ORDER BY PreferenceCount ASC LIMIT 1;
```

Top 2 popular movie types by family members older than 10 years

```
SELECT pm.TypeOfMovie, COUNT(*) as PreferenceCount FROM PreferredMovies pm JOIN FamilyMembers fm ON pm.userID = fm.id WHERE DATE(fm.DateOfBirth) <= DATE_SUB(CURDATE(), INTERVAL 10 YEAR) GROUP BY pm.TypeOfMovie ORDER BY PreferenceCount DESC LIMIT 2;
```

Top 2 popular movie types by female family members

```
SELECT pm.TypeOfMovie, COUNT(*) as PreferenceCount FROM PreferredMovies pm JOIN FamilyMembers fm ON pm.userID = fm.id WHERE fm.Gender = 'Female' GROUP BY pm.TypeOfMovie ORDER BY PreferenceCount DESC LIMIT 2;
```

Family members with no preferred movies

```
SELECT fm.id, fm.FirstName, fm.LastName FROM FamilyMembers fm LEFT JOIN PreferredMovies pm ON fm.id = pm.userID WHERE pm.id IS NULL;
```