Aim: - To create an interactive Form using the **Form widget** in Flutter.

Theory:

A **Form** is a container for grouping and validating multiple form fields. It provides an easy way to manage user input, validate it, and handle form submission.

Flutter offers two key widgets for creating forms:

1. **Form Widget** – a container that holds multiple form fields and tracks their state.
2. **TextFormField Widget** – a specialized input field that integrates with the Form widget for validation and state tracking.

To use a form:

- We first create a GlobalKey<FormState> to uniquely identify the Form and allow validation.
- Inside the Form widget, we include one or more TextFormField widgets.
- Each TextFormField can have built-in validation using the validator parameter.
- We use a RaisedButton or ElevatedButton to trigger form submission and validation.

Important Components:

- GlobalKey<FormState>: A key to access the form state.
- Form: Widget to group multiple form fields.
- TextFormField: Input field with built-in validation support.
- validator: A function that returns an error string if validation fails or null if valid.
- formKey.currentState!.validate(): Triggers validation for all fields.

Form Validation Process:

1. User fills in data.
2. On submit, formKey.currentState!.validate() is called.
3. Each TextFormField runs its validator.
4. If all return null, form is valid.
5. Data can then be saved or processed.

Use Cases of Form Widget in the Crime Alert App

1. **Crime Report Submission**
   Used to collect structured information from users reporting a crime, including crime type, location, and description. The form ensures input validation before submission to Firestore.
2. **Send Feedback**
   Allows users to send feedback through a form, validating that the message field is not empty before sending it as an email
3. **Profile Update**
   Used to update user details like name,contact, state, city and contact information, with built-in validation for each field.
4. **Login/Signup**: For authentication and validation of users

Code Snippet:

1. **Import Required Packages**: Required for image picking, Firebase, and user data handling.

```
import 'package:flutter/foundation.dart' show kIsWeb;
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:image_picker/image_picker.dart';
```

2. **Define a StatefulWidget for the Edit Profile Form**: Allows dynamic updates as the user edits their profile.

```
class EditProfileScreen extends StatefulWidget {
 final String userId;
 const EditProfileScreen({required this.userId, Key? key}) : super(key: key);
 @override
 _EditProfileScreenState createState() => _EditProfileScreenState();
}
```

3. **Create a GlobalKey for the Form:**

```
final _formKey = GlobalKey<FormState>();
```

This key is used to uniquely identify the form and access its state, such as validation and saving user input

4. **Initialize Form State and Controllers:**

```
late UserModel _user;
bool _isLoading = true;
bool _hasChanges = false;
dynamic _pickedImage;
bool _hasNewImage = false;
final FirebaseAuth _auth = FirebaseAuth.instance;
```

Tracks loading, form change status, and selected image.

5. **Build UI with Form and Fields:**

```
Form(
  key: _formKey,
  child: Column(
    children: [
      GestureDetector(onTap: _pickImage, child: CircleAvatar(...)),
      _buildTextFormField(...), // For each field
      ElevatedButton(
        onPressed: _hasChanges ? _updateProfile : null,
        child: const Text('Save Changes'),
      )
    ],
  ),
)
```

Editable form with save button and profile image picker.

6. **Add Form Field Validators:**

```
validator: (val) => val!.trim().isEmpty ? 'Required' : null,
```

Ensures required fields like name/email are filled.

7. **Handle Profile Image Upload:**

```
  try {
    // Handle image upload if changed
    if (_hasNewImage) {
      String? imageUrl;

      if (kIsWeb) {
        imageUrl = await CloudinaryService.uploadImage(
            _pickedImage as Uint8List, "profiles/${_user.uid}");
      } else {
        imageUrl = await CloudinaryService.uploadImage(
            _pickedImage as File, "profiles/${_user.uid}");
      }

      if (imageUrl != null) {
        _user = _user.copyWith(imageURL: imageUrl);
      }
    }
```

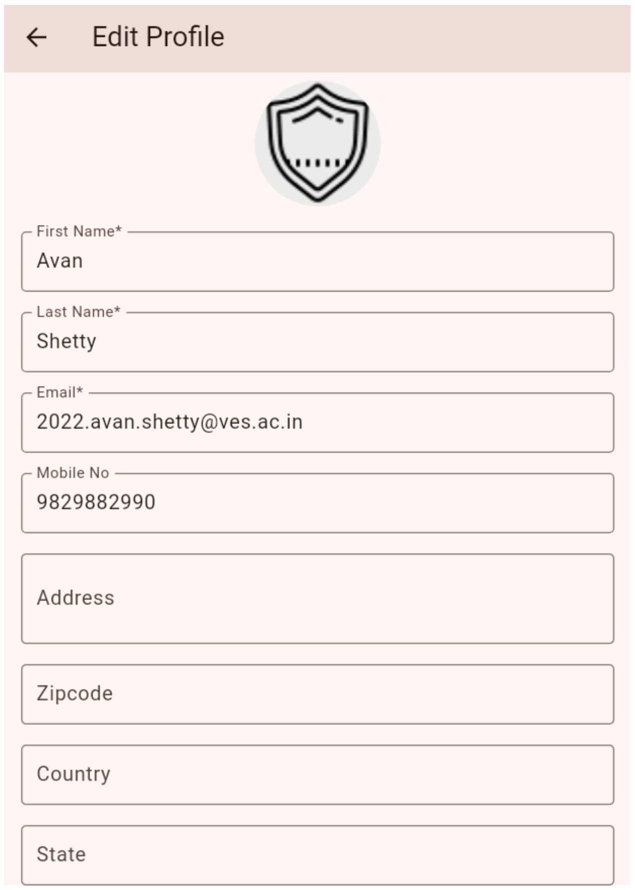
1. If the user selected a new profile picture (_hasNewImage), this uploads it to Cloudinary.
2. It handles both Web and Mobile:
3. Web needs Uint8List, Mobile uses File.
4. After uploading, it gets the image URL and updates the local _user object with the new imageURL.

8. **Submit and Save Profile Changes**:

```
Future<void> _updateProfile() async {
  if (!_formKey.currentState!.validate() || !_hasChanges) return;
  ...
}
```

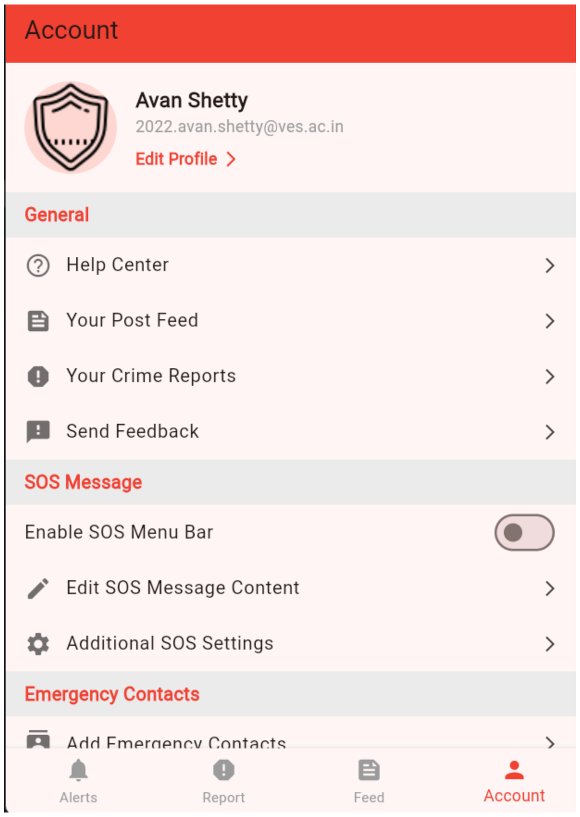
Validates form, uploads image, updates Firestore and auth email.

Output :



First Name*
Avan
Last Name*
Shetty
Email*
2022.avan.shetty@ves.ac.in
Mobile No
9829882990
Address
Zipcode
Country
State

After editing profile/account screen:


Account

Avan Shetty
2022.avan.shetty@ves.ac.in
Edit Profile >

General
Help Center
Your Post Feed
Your Crime Reports
Send Feedback

SOS Message
Enable SOS Menu Bar
Edit SOS Message Content
Additional SOS Settings

Emergency Contacts
Add Emergency Contacts

Conclusion:

In this experiment, we successfully created an interactive form using Flutter's Form and TextFormField widgets. We learned how to validate user input using validators and how to manage form state with a GlobalKey. This allows us to build structured, responsive, and user-friendly form-based interfaces in mobile applications.