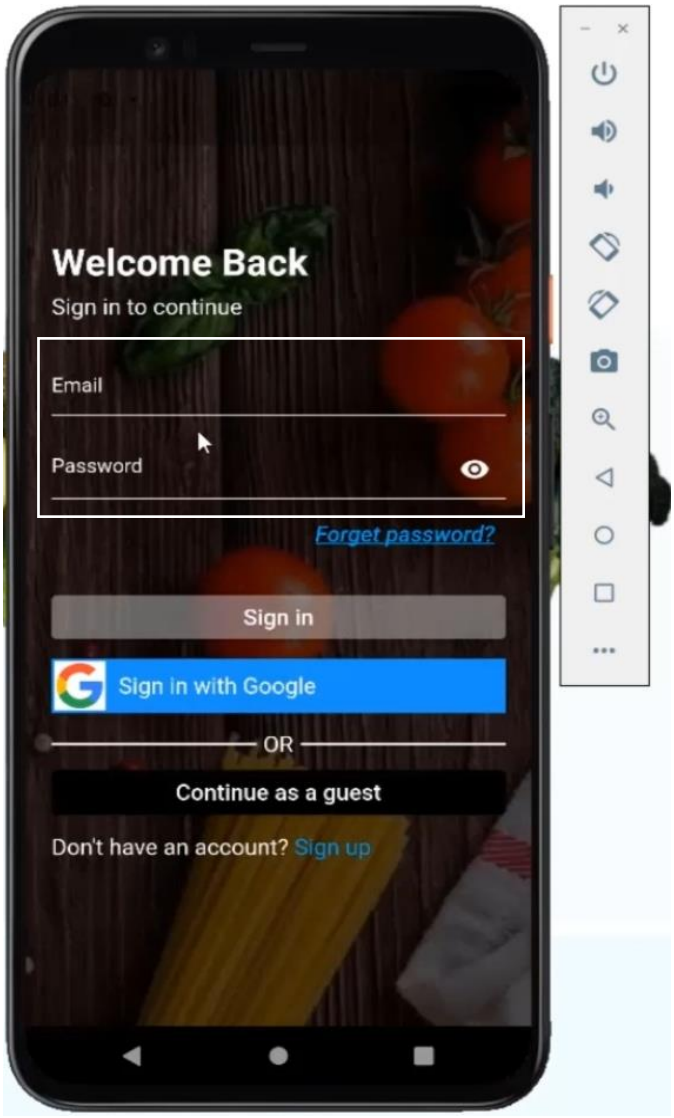
Experiment No.4
MAD & PWA LAB

- **Aim:** To create an interactive Form using a form widget.

- **Theory:**

Apps often require users to enter information into a Text Field. For example, you might require users to log in with an Email Address and Password combination. To make apps secure and easy to use, check whether the information the user has provided is valid. If the user has correctly filled out the form, process the information. If the user submits incorrect information, display a friendly error message letting them know what went wrong.

First, create a Form. The Form widget acts as a container for grouping and validating multiple form fields. When creating the form, provide a GlobalKey. This uniquely identifies the Form, and allows validation of the form in a later step



1. Purpose:

- Data Collection: Forms are primarily used for gathering user input, such as personal information, login credentials, preferences, and feedback.

- Data Validation: Forms provide mechanisms for validating user input to ensure it meets certain criteria or constraints, such as required fields, correct format, and valid ranges.

- User Interaction: Forms enable users to interact with the application by providing input fields and controls for entering data, making selections, and triggering actions.

2. Components:

- Input Fields: Text fields, checkboxes, radio buttons, dropdowns, sliders, date pickers, and other input controls are commonly used within forms to collect user data.

- Labels: Labels provide descriptive text for input fields, helping users understand the purpose of each field and providing context for their input.

- Error Messages: Error messages are displayed when user input fails validation, helping users understand what went wrong and how to correct it.

- Buttons: Buttons such as "Submit", "Cancel", or "Reset" are often included in forms to perform actions like submitting the form data, canceling the operation, or clearing the input fields.

- Validation Logic: Forms include validation logic to check the validity of user input against predefined rules and constraints, ensuring that the data submitted is accurate and reliable.

3. Form Validation:

- Built-in Validators: Flutter provides built-in validators for common validation tasks, such as required fields, email format, numeric values, and regular expressions.

- Custom Validators: Developers can implement custom validation logic by defining validator functions that validate specific aspects of user input, such as password strength, username uniqueness, or custom formatting rules.

- Error Handling: Forms handle validation errors by displaying error messages next to the corresponding input fields, highlighting the fields with invalid data, and preventing form submission until all errors are resolved.

4. State Management:

- Form State: In Flutter, forms are managed using the Form widget, which maintains the state of the form and its child input fields.

- Field State: Each input field within the form maintains its own state, including the current value, validation status, and error message.

- Form Submission: Form submission is typically handled by a callback function, which is invoked when the user submits the form. This callback can access the form data, validate it, and perform any necessary actions, such as sending the data to a server or updating local state.

5. Styling and Layout:

- Layout: Forms can be laid out using various layout widgets like Column, Row, ListView, or SingleChildScrollView to arrange input fields and controls vertically or horizontally.

- Styling: Forms can be styled using custom themes, colors, fonts, and decorations to match the overall visual design of the application and provide a consistent user experience.

Code:

```
import 'package:card_swiper/card_swiper.dart';

import 'package:firebase_auth/firebase_auth.dart';

import 'package:flutter/gestures.dart';

import 'package:flutter/material.dart';

import 'package:grocery_app/screens/auth/forget_pass.dart';

import 'package:grocery_app/screens/auth/register.dart';

import 'package:grocery_app/screens/btm_bar.dart';

import 'package:grocery_app/screens/loading_manager.dart';
```

```
import 'package:grocery_app/services/global_methods.dart';


import '../../consts/contss.dart';
import '../../consts/firebase_consts.dart';
import '../../fetch_screen.dart';
import '../../widgets/auth_button.dart';
import '../../widgets/google_button.dart';
import '../../widgets/text_widget.dart';


class LoginScreen extends StatefulWidget {
  static const routeName = '/LoginScreen';
  const LoginScreen({Key? key}) : super(key: key);


  @override
  State<LoginScreen> createState() => _LoginScreenState();
}


class _LoginScreenState extends State<LoginScreen> {
  final _emailTextController = TextEditingController();
  final _passTextController = TextEditingController();
  final _passFocusNode = FocusNode();
  final _formKey = GlobalKey<FormState>();
  var _obscureText = true;
  @override
```

```
 void dispose() {
  _emailTextController.dispose();
  _passTextController.dispose();
  _passFocusNode.dispose();
  super.dispose();
 }

 bool _isLoading = false;
 void _submitFormOnLogin() async {
  final isValid = _formKey.currentState!.validate();
  FocusScope.of(context).unfocus();

  if (isValid) {
   _formKey.currentState!.save();
   setState(() {
    _isLoading = true;
   });
   try {
    await authInstance.signInWithEmailAndPassword(
        email: _emailTextController.text.toLowerCase().trim(),
        password: _passTextController.text.trim());
    Navigator.of(context).pushReplacement(
     MaterialPageRoute(
      builder: (context) => const FetchScreen(),
```

```
        ),
      );
      print('Succefully logged in');
    } on FirebaseException catch (error) {
      GlobalMethods.errorDialog(
          subtitle: '${error.message}', context: context);
      setState(() {
        _isLoading = false;
      });
    } catch (error) {
      GlobalMethods.errorDialog(subtitle: '$error', context: context);
      setState(() {
        _isLoading = false;
      });
    } finally {
      setState(() {
        _isLoading = false;
      });
    }
  }
}

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

```
    return Scaffold(
      body: LoadingManager(
        isLoading: _isLoading,
        child: Stack(children: [
          Swiper(
            duration: 800,
            autoplayDelay: 8000,
            itemBuilder: (BuildContext context, int index) {
              return Image.asset(
                Constss.authImagesPaths[index],
                fit: BoxFit.cover,
              );
            },
            autoplay: true,
            itemCount: Constss.authImagesPaths.length,
          ),
          Container(
            color: Colors.black.withOpacity(0.7),
          ),
          SingleChildScrollView(
            child: Padding(
              padding: const EdgeInsets.all(20.0),
              child: Column(
                mainAxisAlignment: MainAxisAlignment.center,
```

```
crossAxisAlignment: CrossAxisAlignment.start,
mainAxisSize: MainAxisSize.max,
children: [
 const SizedBox(
  height: 120.0,
 ),
 TextWidget(
  text: 'Welcome Back',
  color: Colors.white,
  textSize: 30,
  isTitle: true,
 ),
 const SizedBox(
  height: 8,
 ),
 TextWidget(
  text: "Sign in to continue",
  color: Colors.white,
  textSize: 18,
  isTitle: false,
 ),
 const SizedBox(
  height: 30.0,
 ),
```

```
Form(
  key: _formKey,
  child: Column(
   children: [
    TextFormField(
     textInputAction: TextInputAction.next,
     onEditingComplete: () => FocusScope.of(context)
       .requestFocus(_passFocusNode),
     controller: _emailTextController,
     keyboardType: TextInputType.emailAddress,
     validator: (value) {
      if (value!.isEmpty || !value.contains('@')) {
       return 'Please enter a valid email address';
      } else {
       return null;
      }
     },
     style: const TextStyle(color: Colors.white),
     decoration: const InputDecoration(
      hintText: 'Email',
      hintStyle: TextStyle(color: Colors.white),
      enabledBorder: UnderlineInputBorder(
       borderSide: BorderSide(color: Colors.white),
      ),
```

```
    focusedBorder: UnderlineInputBorder(
      borderSide: BorderSide(color: Colors.white),
    ),
  ),
),
SizedBox(
  height: 12,
),
//Password

TextFormField(
  textInputAction: TextInputAction.done,
  onEditingComplete: () {
    _submitFormOnLogin();
  },
  controller: _passTextController,
  focusNode: _passFocusNode,
  obscureText: _obscureText,
  keyboardType: TextInputType.visiblePassword,
  validator: (value) {
    if (value!.isEmpty || value.length < 7) {
      return 'Please enter a valid password';
    } else {
      return null;
```

```
  }
},
style: const TextStyle(color: Colors.white),
decoration: InputDecoration(
  suffixIcon: GestureDetector(
    onTap: () {
      setState(() {
        _obscureText = !_obscureText;
      });
    },
    child: Icon(
      _obscureText
          ? Icons.visibility
          : Icons.visibility_off,
      color: Colors.white,
    )),
  hintText: 'Password',
  hintStyle: const TextStyle(color: Colors.white),
  enabledBorder: const UnderlineInputBorder(
    borderSide: BorderSide(color: Colors.white),
  ),
  focusedBorder: const UnderlineInputBorder(
    borderSide: BorderSide(color: Colors.white),
  ),
```

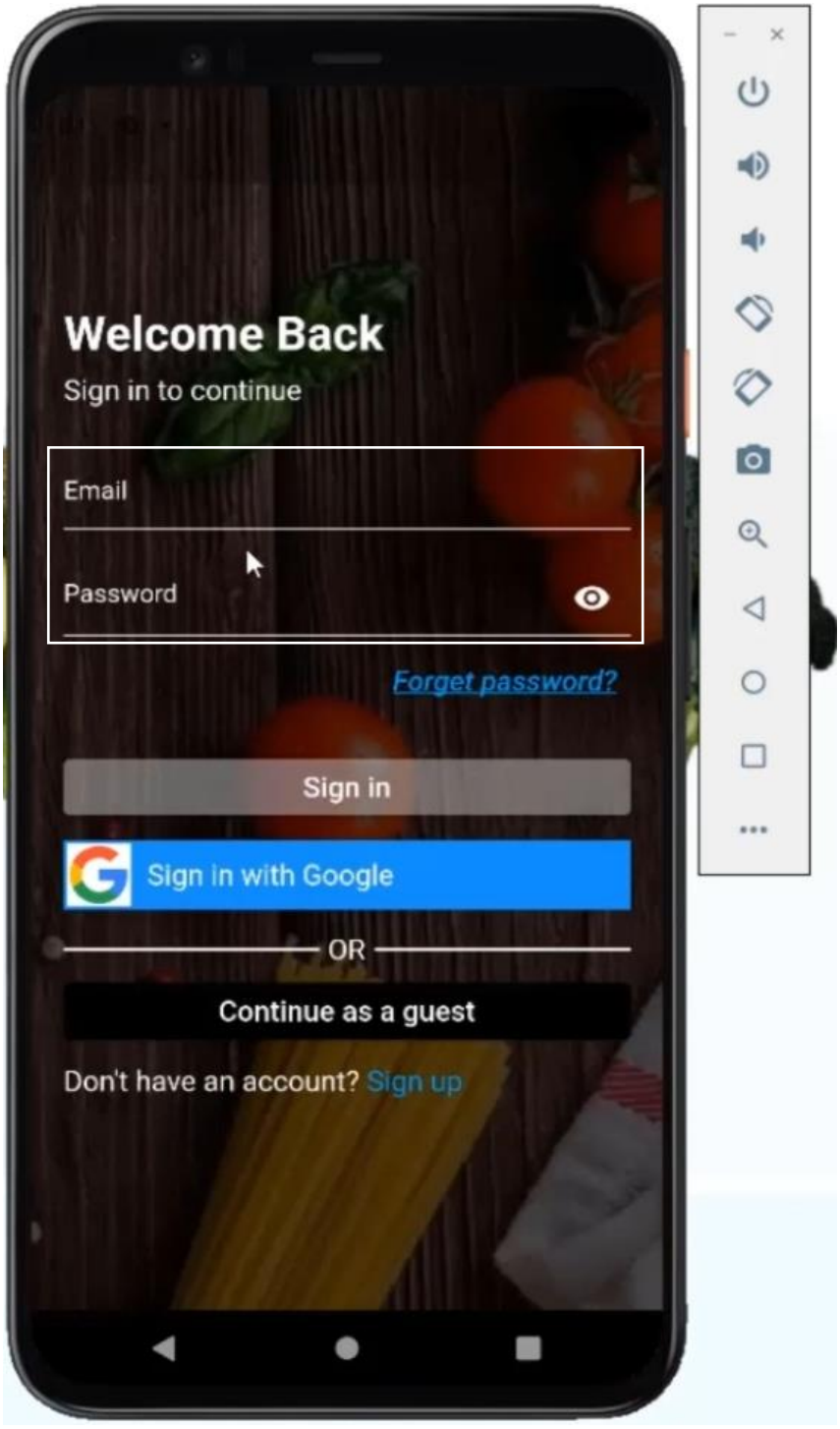
```
            ),
          ),
        ],
      )),
    const SizedBox(
     height: 10,
    ),
    Align(
     alignment: Alignment.topRight,
     child: TextButton(
       onPressed: () {
         GlobalMethods.navigateTo(
             ctx: context,
             routeName: ForgetPasswordScreen.routeName);
       },
       child: const Text(
         'Forget password?',
         maxLines: 1,
         style: TextStyle(
             color: Colors.lightBlue,
             fontSize: 18,
             decoration: TextDecoration.underline,
             fontStyle: FontStyle.italic),
       ),
```

```
  ),
),
const SizedBox(
  height: 10,
),
AuthButton(
  fct: _submitFormOnLogin,
  buttonText: 'Login',
),
const SizedBox(
  height: 10,
),
GoogleButton(),
const SizedBox(
  height: 10,
),
Row(
  children: [
    const Expanded(
      child: Divider(
        color: Colors.white,
        thickness: 2,
      ),
    ),
```

```
      const SizedBox(
        width: 5,
      ),
      TextWidget(
        text: 'OR',
        color: Colors.white,
        textSize: 18,
      ),
      const SizedBox(
        width: 5,
      ),
      const Expanded(
        child: Divider(
          color: Colors.white,
          thickness: 2,
        ),
      ),
    ],
  ),
  const SizedBox(
    height: 10,
  ),
  AuthButton(
    fct: () {
```

```
    Navigator.of(context).push(
      MaterialPageRoute(
        builder: (context) => const FetchScreen(),
      ),
    );
  },
  buttonText: 'Continue as a guest',
  primary: Colors.black,
),
const SizedBox(
  height: 10,
),
RichText(
    text: TextSpan(
        text: 'Don\'t have an account?',
        style: const TextStyle(
            color: Colors.white, fontSize: 18),
        children: [
      TextSpan(
          text: '  Sign up',
          style: const TextStyle(
              color: Colors.lightBlue,
              fontSize: 18,
              fontWeight: FontWeight.w600),
```

```
                    recognizer: TapGestureRecognizer()
                      ..onTap = () {
                        GlobalMethods.navigateTo(
                            ctx: context,
                            routeName: RegisterScreen.routeName);
                      }),
                ]))
            ],
          ),
        ),
      )
    ]),
  ),
 );
 }
}
```



- **Conclusion:**

Hence, we understood how to create an interactive Form using a form widget that has been used in making the Login Screen of our application (Grocery App).