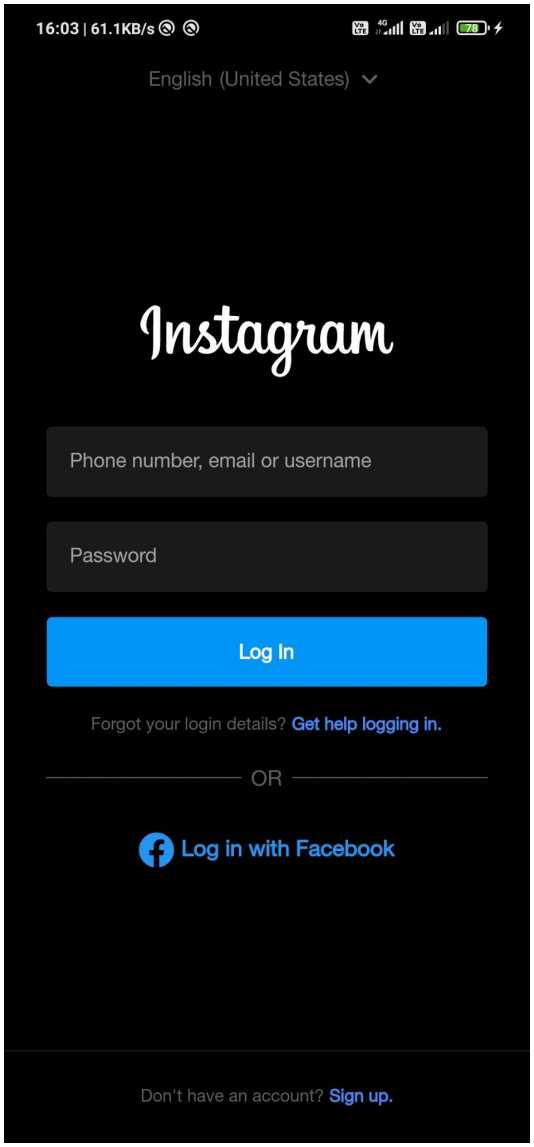
EXPERIMENT NO. 2
MAD and PWA Lab

Aim: To design Flutter UI by including common widgets.

Theory:

Flutter widgets are built using a modern framework that takes inspiration from React. Widgets describe what their view should look like given their current configuration and state. When a widget's state changes, the widget rebuilds its description, which the framework differs against the previous description in order to determine the minimal changes needed in the underlying render tree to transition from one state to the next.



Flutter comes with a suite of powerful widgets, of which the following are used making the **Login Screen** of our application (Instagram) :

- Stateful
- Scaffold
- Column
- SizedBox
- Row
- Text
- Container
- TextField
- Progress Bar
- Center
- Inkwell

A **Stateful widget** is a widget that describes part of the user interface by building a constellation of other widgets that describe the user interface more concretely. They are useful when the part of the user interface you are describing can change dynamically, e.g. due to having an internal clock-driven state, or depending on some system state. Following is an example of the widget I have used in the Login Screen.

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

  @override
  _LoginScreenState createState() => _LoginScreenState();
}

class _LoginScreenState extends State<LoginScreen> {
  @override
  Widget build(BuildContext context) {
    return Container(

    );
  }
}
```

The examples of the StatefulWidget are Checkbox, Radio, Slider, InkWell, Form, and TextField.

The **Scaffold** is a widget in Flutter used to implement the basic material design visual layout structure. It is quick enough to create a general-purpose mobile application and contains almost everything we need to create a functional and responsive Flutter app. This widget is able to occupy the whole device screen. Following is an example of the widget I have used in the Login Screen.

```
return Scaffold(
      resizeToAvoidBottomInset: false,
      body: SingleChildScrollView(
        child: Column(
          children: [],
        ),
      ),
    );
```

The following are the properties and description of the Scaffold widget class.

Property	Description
appBar	It is a horizontal bar that is mainly displayed at the top of the Scaffold widget.
body	It is the other primary and required property of this widget, which will display the main content in the Scaffold.
drawer	It is a slider panel that is displayed at the side of the body.
floatingActionButton	It is a button displayed at the bottom right corner and floating above the body.
backgroundColor	This property is used to set the background color of the whole Scaffold widget.
primary	It is used to tell whether the Scaffold will be displayed at the top of the screen or not.
persistentFooterButton	It is a list of buttons that are displayed at the bottom of the Scaffold widget.
bottomNavigationBar	This property is like a menu that displays a navigation bar at the bottom of the Scaffold.
endDrawer	It is similar to a drawer property, but they are displayed at the right side of the screen by default.
resizeToAvoidBottomInset	Scaffold's floating widgets adjust their size themselves to avoid the onscreen keyboard.
floatingActionButtonLocation	By default, it is positioned at the bottom right corner of the screen.

The **Column widget** arranges its children in a vertical direction on the screen. In other words, it will expect a vertical array of children widgets. A column widget does not appear scrollable because it displays the widgets within the visible view. Flutter column widget has several other properties like mainAxisSize, textDirection, verticalDirection, etc. These are **mainAxisAlignment** and **crossAxisAlignment** properties.

Property	Description
start	It will place the children from the starting of the main axis.
end	It will place the children at the end of the main axis.
center	It will place the children in the middle of the main axis.

spaceBetween	It will place the free space between the children evenly.
spaceAround	It will place the free space between the children evenly and half of that space before and after the first and last children widget.
spaceEvenly	It will place the free space between the children evenly and before and after the first and last children widget.

We can also control how a column widget aligns its children using the property **mainAxisAlignment** and **crossAxisAlignment**. The column's cross-axis will run horizontally, and the main axis will run vertically. Following is an example of the widget I have used in the Login Screen.

```
child: Column(
      children: [],
),
```

The **SizedBox widget** is a box with a specified size. If given a child, this widget forces it to have a specific width and/or height. These values will be ignored if this widget's parent does not permit them.

In the Login Screen, I have used the SizedBox as a vertical space between the widgets which are placed in the screen. Following is an example of the widget I have used in the Login Screen.

```
SizedBox(
      height: 44,
),
```

The following are the properties and description of the SizedBox widget class.

Property	**Description**
width	If non-null, requires the child to have exactly this width.
height	If non-null, requires the child to have exactly this height.

The **Row widget** arranges its children in a horizontal direction on the screen. A row widget does not appear scrollable because it displays the widgets within the visible view. Flutter row widget has several other properties like mainAxisSize, textDirection, verticalDirection, etc. These are **mainAxisAlignment** and **crossAxisAlignment** properties.

Property	Description
start	It will place the children from the starting of the main axis.
end	It will place the children at the end of the main axis.
center	It will place the children in the middle of the main axis.
spaceBetween	It will place the free space between the children evenly.
spaceAround	It will place the free space between the children evenly and half of that space before and after the first and last children widget.
spaceEvenly	It will place the free space between the children evenly and before and after the first and last children widget.

We can control how a row widget aligns its children based on our choice using the property crossAxisAlignment and mainAxisAlignment. The row's cross-axis will run vertically, and the main axis will run horizontally. Following is an example of the widget I have used in the Login Screen.

```
Row(
      mainAxisAlignment: MainAxisAlignment.center,
      children: const <Widget>[
            Text("English",
                  style: TextStyle(color: Color(0xff616161), fontSize: 16.0)),
            SizedBox(
                  width: 5.0,
            ),
            Text("(United States)",
                  style: TextStyle(color: Color(0xff616161), fontSize: 16.0)),
            SizedBox(
                  width: 2.0,
            ),
            Icon(
                  Icons.expand_more,
                  color: Color(0xff616161),
            ),
      ],
),
```

I have used the Row widget for every widget placement in the screen like the TextField, Button etc.

A **Text widget** in Flutter that allows us to display a string of text with a single line in our application. If we do not specify any styling to the text widget, it will use the closest DefaultTextStyle class style. Following is an example of the widget I have used in the Login Screen.

```
Text("Log in with Facebook",
      style: TextStyle(
            color: Colors.blue,
            fontWeight: FontWeight.bold,
            fontSize: 18,
      ),
),
```

The following are the essential properties of the Text widget used in our application :

Property	**Description**
fontWeight	It determines the thickness of the text.
fontSize	It determines the size of the text.
fontFamily	It is used to specify the typeface for the font.
fontStyle	It is used to style the font either in bold or italic form.
Color	It is used to determine the color of the text.

I have used numerous Text widgets in the Login Screen such as “English (United States)”, “Log In”, “Forgot your login details? Get help logging in.”, “Don’t have an account? Sign up.” etc.

The **Container widget** in Flutter is a parent widget that can contain multiple child widgets and manage them efficiently through width, height, padding, background color, etc. It is a widget that combines common painting, positioning, and sizing of the child widgets. It is also a class to store one or more widgets and position them on the screen according to our needs. Following is an example of the widget I have used in the Login Screen.

```
Container(
      child: const Text(
            "Don't have an account? ",
            style: TextStyle(
                  color: Color(0xff616161),
            ),
```

```
      ),
      padding: const EdgeInsets.symmetric(
            vertical: 8,
      ),
),
```

The following are the essential properties of the Container widget :

Property	**Description**
child	This property is used to store the child widget of the container.
color	This property is used to set the background color of the text.
height and width	This property is used to set the container's height and width according to our needs.
margin	This property is used to surround the empty space around the container.
padding	This property is used to set the distance between the border of the container (all four directions) and its child widget.
alignment	This property is used to set the position of the child within the container.
decoration	This property allows the developer to add decoration on the widget.
transform	The transform property allows developers to rotate the container.
constraints	This property is used when we want to add additional constraints to the child.

I have used numerous Container widgets in the Login Screen to keep Text widget, Inkwell widget, Icon widget, TextField widget etc.

A **TextField widget** is an input element which holds the alphanumeric data, such as name, password, address, etc. It is a GUI control element that enables the user to enter text information using a programmable code. It can be of a single-line text field or multiple-line text field.

TextField in Flutter is the most commonly used text input widget that allows users to collect inputs from the keyboard into an app. We can use the TextField widget in building forms, sending messages, creating search experiences, and many more. By default, Flutter decorated the TextField with an underline.

```
TextField(
      cursorColor: Colors.white,
      cursorWidth: 1,
      controller: textEditingController,
      decoration: InputDecoration(
            hintText: hintText,
            border: inputBorder,
            focusedBorder: inputBorder,
            enabledBorder: inputBorder,
            filled: true,
            contentPadding: const EdgeInsets.all(17),
      ),
      keyboardType: textInputType,
      obscureText: isPass,
);
```

The following are the essential properties of the TextField widget used in our application :

Property	**Description**
cursorColor	It is used to show the color of the cursor that indicates the current location of the text insertion point in the field.
cursorWidth	It is used to show how thick the cursor will be.
controller	It is a popular method to retrieve text field value using TextEditingController. It will be attached to the TextField widget and then listen to change and control the widget's text value.
decoration	It is used to show the decoration around TextField.
hintText	It is used to show the hint text inside TextField.
keyboardType	It is used to show the type of keyboard to use for editing the text.
obscureText	It is used to hide the text being edited. When this is set to true, all the characters in the text field are replaced by *.

I have used the TextField widget for the input of Email ID and Password fields in the Login Screen. For the password field, I have kept the obscureText true to hide the text which is to be entered.

A **Progress Bar widget** is a graphical control element used to show the progress of a task such as downloading, uploading, installation, file transfer, etc. I have used **CircularProgressIndicator** for Log In button when clicked.

It is a widget, which spins to indicate the waiting process in your application. It shows the progress of a task in a circular shape. It also displays the progress bar in two ways :

1. A **determinate** progress bar is used when we want to show the progress of ongoing tasks such as the percentage of downloading or uploading files, etc. We can show the progress by specifying the value between 0.0 and 1.0.
2. An **indeterminate** progress bar is used when we do not want to know the percentage of an ongoing process. By default, CircularProgressIndicator shows the indeterminate progress bar.

```
child: _isLoading
      ? const Center(
            child: CircularProgressIndicator(
            color: primaryColor,
            ),
      )
      : const Text('Log In',
            style: TextStyle(
                  fontSize: 16.0, fontWeight: FontWeight.bold)),
```

Over here, I have used the indeterminate progress bar, which will end once the user logs in and the screen changes from Login Screen to Home Screen.

The **Center widget** will center its child within itself. This widget will be as big as possible if its dimensions are constrained and **widthFactor** and **heightFactor** are null. If a dimension is unconstrained and the corresponding size factor is null then the widget will match its child's size in that dimension.

```
Center(
      child: CircularProgressIndicator(
      color: primaryColor,
),
```

The following are the properties of the Center widget :

Property	Description
alignment	It is used to show how to align the child.
widthFactor	If non-null, sets its width to the child's width multiplied by this factor.
heightFactor	If non-null, sets its height to the child's width multiplied by this factor.

I have used the Center widget to center the progress bar used in the Log In button of the Login Screen.

Buttons are the graphical control element that provides a user to trigger an event such as taking actions, making choices, searching things, and many more. They can be placed anywhere in our UI like dialogs, forms, cards, toolbars, etc.

InkWell widget button is a material design concept, which is used for touch response. This widget comes under the Material widget where the ink reactions are actually painted. It creates the app UI interactive by adding gesture feedback. It is mainly used for adding a splash ripple effect.

```
child: InkWell(
      onTap: loginUser,
      child: Container(
            child: _isLoading
            ? const Center(
                  child: CircularProgressIndicator(
                        color: primaryColor,
                  ),
            )
            : const Text('Log In',
                  style: TextStyle(
                        fontSize: 16.0, fontWeight: FontWeight.bold)),
            width: double.infinity,
            alignment: Alignment.center,
            padding: const EdgeInsets.symmetric(vertical: 17),
            decoration: const ShapeDecoration(
            shape: RoundedRectangleBorder(
                  borderRadius: BorderRadius.all(Radius.circular(4)),
            ),
            color: blueColor),
      ),
),
```

I have used the Inkwell widget to make the Log In button in the Login Screen.

Conclusion: Hence, we understood how to design Flutter UI by including the widgets that have been used in making the **Login Screen** of our application (Instagram).