Vivekanand Education Society's Institute of Technology

An Autonomous Institute Affiliated to University of Mumbai
Hashu Advani Memorial Complex, Collector Colony, Chembur East, Mumbai - 400074.

Since 1962

Department of Information Technology

CERTIFICATE

This is to certify that **Prajakta Upadhye** of **D15A** semester **VI,** have successfully completed necessary experiments in the **MAD & PWA Lab** under my supervision in **VES Institute of Technology** during the academic year **2023-2024.**

Lab Assistant

Subject Teacher
Mrs. Kajal Joseph

Principal

Head of Department
Dr. Mrs. Shalu Chopra

Name of the Course : MAD & PWA Lab **Course Code :** ITL604

Year/Sem/Class : D15A/D15B **A.Y.:** 23-24

Faculty Incharge : Mrs. Kajal Joseph.

Lab Teachers : Mrs. Kajal Jewani.

Email : kajal.jewani@ves.ac.in

Programme Outcomes: The graduate will be able to:

PO1) Basic Engineering knowledge: An ability to apply the fundamental knowledge in mathematics, science and engineering to solve problems in Computer engineering.

PO2) Problem Analysis: Identify, formulate, research literature and analyze computer engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and computer engineering and sciences.

PO3) Design/ Development of Solutions: Design solutions for complex computer engineering problems and design system components or processes that meet specified needs with appropriate consideration for public health and safety, cultural, societal and environmental considerations.

PO4) Conduct investigations of complex engineering problems using research-based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of information to provide valid conclusions.

PO5) Modern Tool Usage: Create, select and apply appropriate techniques, resources and modern computer engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6) The Engineer and Society: Apply reasoning informed by contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to computer engineering practice.

PO7) Environment and Sustainability: Understand the impact of professional computer engineering solutions in societal and environmental contexts and demonstrate knowledge of and need for sustainable development.

PO8) Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of computer engineering practice.

PO9) Individual and Team Work: Function effectively as an individual, and as a member or leader in diverse teams and in multidisciplinary settings.

PO10) Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations and give and receive clear instructions.

PO11) Project Management and Finance: Demonstrate knowledge and understanding of computer engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12) Life-long Learning: Recognize the need for and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change.

Program specific Outcomes

PSO1) An ability to manage and analyze data / information effectively for making better decisions.

PSO2) Demonstrate the ability to use state of the art technologies and tools including Free and Open Source Software (FOSS) tools in developing software.

Lab Objectives:

Sr. No.	Lab Objectives
The Lab experiments aims:	
1	Learn the basics of the Flutter framework.
2	Develop the App UI by incorporating widgets, layouts, gestures and animation
3	Create a production ready Flutter App by including files and firebase backend service.
4	Learn the Essential technologies, and Concepts of PWAs to get started as quickly and efficiently as possible
5	Develop responsive web applications by combining AJAX development techniques with the jQuery JavaScript library.
6	Understand how service workers operate and also learn to Test and Deploy PWA.

Lab Outcomes:

Sr. No.	Lab Outcomes	Cognitive levels of attainment as per Bloom's Taxonomy
On Completion of the course the learner/student should be able to:		
1	Understand cross platform mobile application development using Flutter framework	L1, L2
2	Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation	L3
3	Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS	L3, L4
4	Understand various PWA frameworks and their requirements	L1, L2
5	Design and Develop a responsive User Interface by applying PWA Design techniques	L3
6	Develop and Analyze PWA Features and deploy it over app hosting solutions	L3, L4

Index

Sr. No	Experiment Title	LO	DOP	DOS	Grade
1.	To install and configure the Flutter Environment	LO1	19/01	02/02	15
2.	To design Flutter UI by including common widgets.	LO2	26/01	02/02	15
3.	To include icons, images, fonts in Flutter app	LO2	02/02	09/02	15
4.	To create an interactive Form using form widget	LO2	09/02	16/02	15
5.	To apply navigation, routing and gestures in Flutter App	LO2	16/02	23/02	15
6.	To Connect Flutter UI with fireBase database	LO3	23/02	08/03	15
7.	To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”.	LO4	08/03	22/03	15
8.	To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA	LO5	15/03	22/03	15
9.	To implement Service worker events like fetch, sync and push for E-commerce PWA	LO5	22/03	01/04	15
10.	To study and implement deployment of Ecommerce PWA to GitHub Pages.	LO5	29/03	01/04	15
11.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.	LO6	29/03	01/04	15
12.	Assignment-1	LO1,LO2 ,LO3	28/01	05/02	5
13.	Assignment-2	LO4,LO5 ,LO6	14/03	21/03	4

MAD & PWA Lab Journal

Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	65
Name	PRAJAKTA UPADHYE
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	15

MAD PWA LAb 1

NAME: Prajakta Upadhye **Batch :** C

Class : D15A **Roll No.** : 65

Aim: To install flutter and Android Studio and create a 'Hello World App' using Flutter

Step 1: Download the installation bundle of the Flutter Software Development Kit for windows. To download Flutter SDK, Go to its official website https://docs.flutter.dev/get-started/install , you will get the following screen.

Step 2: Next, to download the latest Flutter SDK, click on the Windows **icon**. Here, you will find the download link for SDK.

Step 3: When your download is complete, extract the **zip** file and place it in the desired installation folder or location, for example, C: /Flutter.

Step 4: To run the Flutter command in regular windows console, you need to update the system path to include the flutter bin directory. The following steps are required to do this:

Step 4.1: Go to MyComputer properties -> advanced tab -> environment variables. You will get the following screen.

Step 4.3: In the above window, click on New->write path of Flutter bin folder in variable value - > ok -> ok -> ok.

Step 5: Now, run the $ **flutter** command in command prompt.

```
C:\Users\acer>flutter --version

┌─────────────────────────────────────────────────────────┐
│ A new version of Flutter is available!                  │
│                                                         │
│ To update to the latest version, run "flutter upgrade". │
└─────────────────────────────────────────────────────────┘
Flutter 3.16.7 • channel stable • https://github.com/flutter/flutter.git
Framework • revision ef1af02aea (8 days ago) • 2024-01-11 15:19:26 -0600
Engine • revision 4a585b7929
Tools • Dart 3.2.4 • DevTools 2.28.5
```

```
C:\Users\acer>flutter devices
Found 3 connected devices:
  Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22621.3007]
  Chrome (web)      • chrome  • web-javascript • Google Chrome 120.0.6099.225
  Edge (web)        • edge    • web-javascript • Microsoft Edge 120.0.2210.121

Run "flutter emulators" to list and start any available device emulators.

If you expected another device to be detected, please run "flutter doctor" to diagnose potenti
try increasing the time to wait for connected devices with the "--device-timeout" flag. Visit
for troubleshooting tips.
```

Now, run the $ **flutter doctor** command. This command checks for all the requirements of Flutter app development and displays a report of the status of your Flutter installation.

Step 6: When you run the above command, it will analyze the system and show its report, as shown in the below image. Here, you will find the details of all missing tools, which required to run Flutter as well as the development tools that are available but not connected with the device.

Step 7: Install the Android SDK. If the flutter doctor command does not find the Android SDK tool in your system, then you need first to install the Android Studio IDE. To install Android Studio IDE, do the following steps.

Step 7.1: Download the latest Android Studio executable or zip file from the official site.

Step 7.2: When the download is complete, open the **.exe** file and run it. You will get the following dialog box.

Step 7.3: Follow the steps of the installation wizard. Once the installation wizard completes, you will get the following screen.

Step 7.4: In the above screen, click Next-> Finish. Once the Finish button is clicked,

you need to choose the 'Don't import Settings option' and click OK. It will start the Android Studio.

Step 7.5 run the $ **flutter doctor** command and Run flutter doctor --android-licenses command.

Step 8: Next, you need to set up an Android emulator. It is responsible for running and testing the Flutter application.

Step 8.1: To set an Android emulator, go to Android Studio > Tools > Android > AVD Manager and select Create Virtual Device. Or, go to Help->Find Action->Type Emulator in the search box. You will get the following screen.

Step 8.2: Choose your device definition and click on Next.

Step 8.3: Select the system image for the latest Android version and click on Next.

Step 8.4: Now, verify the all AVD configuration. If it is correct, click on Finish. The following screen appears.

Step 8.5: Last, click on the icon pointed into the red color rectangle. The Android emulator displayed as below screen.

Step 9: Now, install Flutter and Dart plugin for building Flutter application in Android Studio. These plugins provide a template to create a Flutter application, give an option to run and debug Flutter application in the Android Studio itself. Do the following steps to install these plugins.

Step 9.1: Open the Android Studio and then go to File->Settings->Plugins.

Step 9.2: Now, search the Flutter plugin. If found, select Flutter plugin and click install. When you click on install, it will ask you to install Dart plugin as below screen. Click yes to proceed.

Step 9.3: Restart the Android Studio.

CODE:

```
import 'package:flutter/material.dart';
void main() {
runApp(const MyApp());
}
class MyApp extends StatelessWidget {
const MyApp({Key? key}) : super(key: key);
@override
Widget build(BuildContext context) {
return MaterialApp(
title: 'Welcome to Flutter',
home: Scaffold(
appBar: AppBar(
title: const Text('Welcome Prajakta Upadhye'),
),
body: const Center(
child: Text('Hello World'),
),
),
);
}
}
```

Conclusion: Thus,we installed flutter and android studio and created our first flutter application.

MAD & PWA Lab Journal

Experiment No.	02
Experiment Title.	To design Flutter UI by including common widgets.
Roll No.	65
Name	PRAJAKTA UPADHYE
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

MAD PWA LAb 2

NAME: Prajakta Upadhye **Batch :** C

Class : D15A **Roll No.** : 65

Aim: To design Flutter UI by including common widgets.

Widgets:

1. Column:
    a. Column is a widget that arranges its children in a vertical array.
    b. It is commonly used to create vertical layouts, stacking widgets on top of each other.
2. Row:
    a. Row is a widget that arranges its children in a horizontal array.
    b. It is frequently used to create horizontal layouts, aligning widgets side by side.
3. Stack:
    a. Stack is a widget that allows children to be positioned on top of each other.
    b. It's useful for creating layered UIs, where widgets can overlap and be visually stacked.
4. Container:
    a. Container is a versatile box model widget that can contain other widgets.
    b. It allows customization of its appearance, such as background color, padding, margin, and more.
5. Padding:
    a. Padding is a widget used to add padding around its child widget.
    b. It is commonly used to create space or margins around widgets within a Column, Row, or other layout widgets.
6. SizedBox:
    a. SizedBox is a box model widget used to constrain the size of its child widget.
    b. It can be used to set a fixed width, height, or both, providing control over the dimensions of a widget.
7. TextField:
    a. TextField is a widget used to accept user input in the form of text.
    b. It provides a text input field where users can type or paste text.
8. RichText:

a. RichText is a widget that allows text to be styled with multiple styles within the same text widget.
b. It is useful for creating text with different fonts, colors, or other styles.

Code:

```
import 'package:flutter/material.dart';
import 'dart:ui' as ui;
void main() {
 runApp(const MyApp());
}

class MyApp extends StatelessWidget {
 const MyApp({Key? key}) : super(key: key);

 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     home: HomePage(),
   );
 }
}

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

 @override
 State<HomePage> createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
 @override
 Widget build(BuildContext context) {
   return Scaffold(body: LoginScreen());
 }
}

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

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

class _LoginScreenState extends State<LoginScreen> {
 @override
 Widget build(BuildContext context) {
   return Padding(
     padding: const EdgeInsets.all(5.0),
     child: Stack(
       children: [
         Column(
           mainAxisAlignment: MainAxisAlignment.start,
           crossAxisAlignment: CrossAxisAlignment.start,
           children: [
             Stack(
               children: [
                 Container(
                   color: Colors.blueAccent,
                   height: 40.0,
                   width: double.infinity,
                   margin: const EdgeInsets.only(top: 50.0),
                 ),
                 Positioned(
                   left: 150.0,
                   top: 50.0 + (30.0 - 16.0) / 2,
                   child: Text(
                     "FlipKart",
                     style: TextStyle(
                       color: Colors.white,
                       fontSize: 16.0,
                       fontWeight: FontWeight.bold,
                     ),
                   ),

                 ),
                 SizedBox(width: 8.0), // Add some space between text and image
                 Positioned(
                   top: 53.0,
                   left: 215.0,
                   child: Image.asset(
                     'assets/flipkartlogo.jpg', // Replace with your image path
                     height: 30.0,
                     width: 30.0,
```

```
        // color: Colors.white,
      ),
    ),
  ],
),
Padding(
  padding: const EdgeInsets.only(left: 10.0, top: 15.0),
  child: Text(
    "Log in for the best experience",
    style: TextStyle(
      color: Colors.black,
      fontSize: 20.0,
      fontWeight: FontWeight.bold,
    ),
  ),
),
Padding(
  padding: const EdgeInsets.only(left: 10.0, top: 1.0),
  child: Text(
    "Enter your phone number to continue",
    style: TextStyle(
      color: Colors.grey,
      fontSize: 14.0,
    ),
  ),
),
Padding(
  padding: const EdgeInsets.only(left: 20.0, top: 30.0, right: 20.0),
  child: Container(
    decoration: BoxDecoration(
      border: Border.all(color: Colors.blueAccent),
      borderRadius: BorderRadius.circular(8.0),
    ),
    child: TextField(
      decoration: InputDecoration(
        hintText: 'Mobile Number',
        hintStyle: TextStyle(color: Colors.grey),
        border: InputBorder.none,
        contentPadding: EdgeInsets.all(15.0),
      ),
    ),
  ),
),
```

```
Padding(
  padding: const EdgeInsets.only(left: 260.0, top: 5),
  child: Text(
    "Use Email-ID",
    style: TextStyle(
      color: Colors.blue[800],
      fontSize: 15.0,
      fontWeight: FontWeight.bold,
    ),
  ),
),
Padding(
  padding: const EdgeInsets.only(left: 20.0, top: 30.0, right: 20.0),
  child: Container(
    decoration: BoxDecoration(
      border: Border.all(color: Colors.blueAccent),
      borderRadius: BorderRadius.circular(8.0),
    ),
    child: TextField(
      decoration: InputDecoration(
        hintText: 'First Name',
        hintStyle: TextStyle(color: Colors.grey),
        border: InputBorder.none,
        contentPadding: EdgeInsets.all(15.0),
      ),
    ),
  ),
),
Padding(
  padding: const EdgeInsets.only(left: 20.0, top: 10.0, right: 20.0),
  child: Container(
    decoration: BoxDecoration(
      border: Border.all(color: Colors.blueAccent),
      borderRadius: BorderRadius.circular(8.0),
    ),
    child: TextField(
      decoration: InputDecoration(
        hintText: 'Last Name',
        hintStyle: TextStyle(color: Colors.grey),
        border: InputBorder.none,
        contentPadding: EdgeInsets.all(15.0),
      ),
    ),
  ),
```

```
            ),
            Padding(
              padding: const EdgeInsets.only(left: 20.0, top: 20.0, right: 20.0),
              child: RichText(
                text: TextSpan(
                  text: 'By continuing, you agree to FlipKart`s ',
                  style: TextStyle(color: Colors.grey),
                  children: [
                    TextSpan(
                      text: 'Terms of Use and Privacy Policy',
                      style: TextStyle(color: Colors.blue),
                    ),
                  ],
                ),
              ),
            ),
          ],
        ),
        Positioned(
          left: 0,
          right: 0,
          bottom: 5,
          child: Container(
            color: Colors.blue,
            height: 50.0,
            child: Center(
              child: Text(
                'Continue',
                style: TextStyle(
                  color: Colors.white,
                  fontSize: 18.0,
                  fontWeight: FontWeight.bold,
                ),
              ),
            ),
          ),
        ),
      ],
    ),
  );
}
}
```

Output:

Conclusion:

Thus we studied how to design UI using common widgets.

MAD & PWA Lab Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	65
Name	PRAJAKTA UPADHYE
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

MAD PWA LAb 3

NAME: Prajakta Upadhye **Batch :** C

Class : D15A **Roll No.** : 65

Aim: Exploring Flutter Widgets

Theory:

Images:

- Flutter supports the integration of various image formats, such as PNG, JPEG, GIF, and WebP.
- Images can be displayed using the Image widget, and different aspects like width, height, and alignment can be customized.
- Asset images are commonly used and are declared in the pubspec.yaml file under the flutter section. The Image.asset constructor is then used to load these assets

Fonts:

- Fonts in Flutter can be customized by specifying the TextStyle for text widgets.
- Custom fonts, including TrueType and OpenType fonts, can be used by declaring them in the pubspec.yaml file and specifying them in the TextStyle with the fontFamily property.

Icons:

- Flutter provides a wide range of icons through the Icons class, including Material Icons and Cupertino Icons.
- The Icon widget is used to display icons, and you can customize their appearance using properties like color, size, and opacity.

Button Types:

- Flutter provides various button types, including ElevatedButton, TextButton, OutlinedButton, and more.
- These buttons offer different visual styles, allowing developers to choose the one that best fits the application's design.

Button Properties:

- Buttons can be customized using properties such as onPressed (to define the action when the button is pressed), style (to modify the button's appearance), and child (to set the content inside the button).

Code:

```
import 'package:flutter/material.dart';
import 'dart:ui' as ui;
void main() {
 runApp(const MyApp());
}

class MyApp extends StatelessWidget {
 const MyApp({Key? key}) : super(key: key);

 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     home: HomePage(),
   );
 }
}

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

 @override
 State<HomePage> createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
 @override
 Widget build(BuildContext context) {
   return Scaffold(body: LoginScreen());
 }
}

void _navigateToItemDescription(BuildContext context, String imagePath) {
  Navigator.of(context).push(
    MaterialPageRoute(
      builder: (context) => ItemDescription(imagePath: imagePath),
```

```
    ),
  );
}

class ItemDescription extends StatelessWidget {
  final String imagePath;

  ItemDescription({required this.imagePath});
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      // appBar: AppBar(
      //   title: Text('Item Description'),
      // ),
      body: Padding(
        padding: const EdgeInsets.all(5.0),
        child: Column(
          children: [
            // Search bar above the watch image
            Padding(
              padding: const EdgeInsets.only(
                  left: 20.0, top: 80.0, right: 20.0, bottom: 20.0),
              child: Container(
                decoration: BoxDecoration(
                  border: Border.all(color: Colors.black12),
                  color: Colors.transparent,
                  borderRadius: BorderRadius.circular(8.0),
                ),
                child: Stack(
                  alignment: Alignment.centerLeft,
                  children: [
                    Padding(
                      padding: const EdgeInsets.only(left: 40.0),
                      child: TextField(
                        decoration: InputDecoration(
                          hintText: 'Search',
                          hintStyle: TextStyle(color: Colors.black),
                          border: InputBorder.none,
                          contentPadding: EdgeInsets.all(15.0),
                        ),
                      ),
                    ),
                    Padding(
                      padding: const EdgeInsets.only(left: 15.0),
```

```
          child: Icon(
            Icons.search,
            color: Colors.black,
          ),
        ),
        Padding(
          padding: const EdgeInsets.only(left: 315.0),
          child: Icon(
            Icons.mic,
            color: Colors.black,
          ),
        ),
      ],
    ),
  ),
),
// Watch image
Positioned(
  top: 220, // Adjust the distance from the top as needed
  left: 20.0,
  right: 20.0,
  child: Image.asset(
    imagePath,
    // 'assets/watch.png',
    width: 350.0, // Adjust the width as needed
    height: 350.0, // Adjust the height as needed
  ),
),
Padding(
  padding: const EdgeInsets.only(top: 15.0, right: 200.0),
  child: Text(
    "Select Variant",
    style: TextStyle(
      color: Colors.black,
      fontSize: 20.0,
      fontWeight: FontWeight.bold,
    ),
  ),
),
Divider(
  color: Colors.black,
  thickness: 1.0,
),
Padding(
```

```
      padding: const EdgeInsets.only(top: 10.0, right: 100.0),
      child: Text(
        "Strap Color: Cherry Blossom",
        style: TextStyle(
          color: Colors.black,
          fontSize: 18.0,
        ),
      ),
    ),
    Divider(
      color: Colors.black,
      thickness: 1.0,
    ),
    Padding(
      padding:
          const EdgeInsets.only(top: 10.0, right: 150.0, bottom: 20.0),
      child: Text(
        "Display Size: 46.4 mm",
        style: TextStyle(
          color: Colors.black,
          fontSize: 18.0,
        ),
      ),
    ),
    Expanded(
      child: Align(
        alignment: Alignment.bottomCenter,
        child: Padding(
          padding: const EdgeInsets.only(bottom: 40.0),
          child: Row(
            children: [
              Expanded(
                child: Container(
                  height: 50.0,
                  child: ElevatedButton(
                    onPressed: () {
                      // Handle first button press
                    },
                    style: ElevatedButton.styleFrom(
                      shape: RoundedRectangleBorder(
                        borderRadius: BorderRadius.circular(0.0),
                      ),
                      primary: Colors.white,
                    ),
```

```
                child: Text(
                  'Add to Cart',
                  style: TextStyle(
                    fontSize: 25.0,
                    color: Colors.black,
                  ),
                ),
              ),
            ),
          ),
          SizedBox(width: 5.0),
          Expanded(
            child: Container(
              height: 50.0,
              child: ElevatedButton(
                onPressed: () {
                  // Handle second button press
                },
                style: ElevatedButton.styleFrom(
                  shape: RoundedRectangleBorder(
                    borderRadius: BorderRadius.circular(0.0),
                  ),
                  primary: Colors.yellow,
                ),
                child: Text(
                  'Buy Now',
                  style: TextStyle(
                    fontSize: 25.0,
                    color: Colors.black,
                  ),
                ),
              ),
            ),
          ),
        ],
      ),
    ),
  ),
  ),
  );
}}
```

Conclusion:

Thus we studied how to design UI using various icons ,images,buttons.

MAD & PWA Lab Journal

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget
Roll No.	65
Name	PRAJAKTA UPADHYE
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

MAD PWA LAb 4

NAME: Prajakta Upadhye **Batch :** C

Class : D15A **Roll No.** : 65

Aim: To create interactive Form using form widget

Theory:

- The Form widget in Flutter is a container for grouping and validating multiple form-related widgets.

2. **Usage:**

- Wrap your form elements, such as text fields, checkboxes, and buttons, within the Form widget.

3. **Key Properties:**

- key: Assign a unique key to the Form to identify and manipulate it, if necessary.

TextFormField:

1. **Definition:**

- TextFormField is a form field that takes care of the most common use case of a single-line text input.

2. **Usage:**

- Embed TextFormField widgets within the Form to collect text input.

3. **Properties:**

- controller: Use a TextEditingController to control and manipulate the text entered.
- validator: Implement a function to validate the input based on specific criteria.

Form Validation:

1. **Form Validation:**

- Utilize the validator property in TextFormField or other form fields to perform validation.
- The validator function should return an error message if validation fails or null if it succeeds.

2. **Form Submission:**

- Use a FormState object to handle form submission.
- Typically, this involves calling a function when the user presses a submit button.

Form Submission:

1. **Submit Button:**

- Create a button (e.g., ElevatedButton) that triggers the form submission.
- Wrap the submission logic in a function (e.g., _submitForm).

2. **Submission Logic:**

- Inside the submission function, access the form data using controllers or the FormState.
- Perform any necessary actions, such as sending data to a server or updating the UI.

CODE:

```
class RegisterScreen extends StatelessWidget {
  void _navigateToLoginScreen(BuildContext context) {
    Navigator.of(context).pushReplacement(
      MaterialPageRoute(
        builder: (context) => LoginScreen(),
      ),
    );
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Padding(
        padding: const EdgeInsets.only(top: 50.0),
        child: Stack(
          children: [
            Column(
```

```
mainAxisAlignment: MainAxisAlignment.start,
crossAxisAlignment: CrossAxisAlignment.start,
children: [
  Stack(
    children: [
      Container(
        color: Colors.blueAccent,
        height: 40.0,
        width: double.infinity,
        margin: const EdgeInsets.only(top: 50.0),
      ),
      Positioned(
        left: 150.0,
        top: 50.0 + (30.0 - 16.0) / 2,
        child: Text(
          "FlipKart",
          style: TextStyle(
            color: Colors.white,
            fontSize: 16.0,
            fontWeight: FontWeight.bold,
          ),
        ),
      ),
      SizedBox(
          width: 8.0), // Add some space between text and image
      Positioned(
        top: 53.0,
        left: 215.0,
        child: Image.asset(
          'assets/flipkartlogo.jpg', // Replace with your image path
          height: 30.0,
          width: 30.0,

          // color: Colors.white,
        ),
      ),
    ],
  ),
  Padding(
    padding: const EdgeInsets.only(left: 10.0, top: 15.0),
    child: Text(
      "Register for the best experience",
      style: TextStyle(
        color: Colors.black,
```

```
      fontSize: 20.0,
      fontWeight: FontWeight.bold,
    ),
  ),
),
Padding(
  padding: const EdgeInsets.only(left: 10.0, top: 1.0),
  child: Text(
    "Enter your phone number to continue",
    style: TextStyle(
      color: Colors.grey,
      fontSize: 14.0,
    ),
  ),
),
Padding(
  padding:
      const EdgeInsets.only(left: 20.0, top: 30.0, right: 20.0),
  child: Container(
    decoration: BoxDecoration(
      border: Border.all(color: Colors.blueAccent),
      borderRadius: BorderRadius.circular(8.0),
    ),
    child: TextField(
      decoration: InputDecoration(
        hintText: 'Mobile Number',
        hintStyle: TextStyle(color: Colors.grey),
        border: InputBorder.none,
        contentPadding: EdgeInsets.all(15.0),
      ),
    ),
  ),
),
Padding(
  padding: const EdgeInsets.only(left: 260.0, top: 5),
  child: Text(
    "Use Email-ID",
    style: TextStyle(
      color: Colors.blue[800],
      fontSize: 15.0,
      fontWeight: FontWeight.bold,
    ),
  ),
),
```

```
Padding(
  padding:
    const EdgeInsets.only(left: 20.0, top: 30.0, right: 20.0),
  child: Container(
    decoration: BoxDecoration(
      border: Border.all(color: Colors.blueAccent),
      borderRadius: BorderRadius.circular(8.0),
    ),
    child: TextField(
      decoration: InputDecoration(
        hintText: 'First Name',
        hintStyle: TextStyle(color: Colors.grey),
        border: InputBorder.none,
        contentPadding: EdgeInsets.all(15.0),
      ),
    ),
  ),
),
Padding(
  padding:
    const EdgeInsets.only(left: 20.0, top: 10.0, right: 20.0),
  child: Container(
    decoration: BoxDecoration(
      border: Border.all(color: Colors.blueAccent),
      borderRadius: BorderRadius.circular(8.0),
    ),
    child: TextField(
      decoration: InputDecoration(
        hintText: 'Last Name',
        hintStyle: TextStyle(color: Colors.grey),
        border: InputBorder.none,
        contentPadding: EdgeInsets.all(15.0),
      ),
    ),
  ),
),
Padding(
  padding:
    const EdgeInsets.only(left: 20.0, top: 10.0, right: 20.0),
  child: Container(
    decoration: BoxDecoration(
      border: Border.all(color: Colors.blueAccent),
      borderRadius: BorderRadius.circular(8.0),
    ),
```

```
        child: TextField(
          decoration: InputDecoration(
            hintText: 'Password',
            hintStyle: TextStyle(color: Colors.grey),
            border: InputBorder.none,
            contentPadding: EdgeInsets.all(15.0),
          ),
        ),
      ),
    ),
    Padding(
      padding:
          const EdgeInsets.only(left: 20.0, top: 20.0, right: 20.0),
      child: RichText(
        text: TextSpan(
          text: 'By continuing, you agree to FlipKart`s ',
          style: TextStyle(color: Colors.grey),
          children: [
            TextSpan(
              text: 'Terms of Use and Privacy Policy',
              style: TextStyle(color: Colors.blue),
            ),
          ],
        ),
      ),
    ),
    Padding(
      padding: const EdgeInsets.only(top: 20.0),
      child: ElevatedButton(
        onPressed: () => _navigateToLoginScreen(context),
        style: ElevatedButton.styleFrom(
          primary: Colors.blue, // Set the button color here
        ),
        child: Text(
          'Log In',
          style: TextStyle(
            color: Colors.white,
            fontSize: 18.0,
            fontWeight: FontWeight.bold,
          ),
        ),
      ),
    ),
  ],
```

```
            ),
          ],
        ),
      ),
    );
  }
}
```

11:45
DEBUG
FlipKart
Register for the best experience
Enter your phone number to continue
Mobile Number
Use Email-ID
First Name
Last Name
Password
By continuing, you agree to FlipKart`s Terms of Use and Privacy Policy
Log In

CONCLUSION:

Thus we created interactive Form using form widget

MAD & PWA Lab Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	65
Name	PRAJAKTA UPADHYE
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

MAD PWA LAb 5

NAME: Prajakta Upadhye **Batch :** C

Class : D15A **Roll No.** : 65

Aim: To apply navigation,routing and gestures features in Flutter Application.

Theory:

Navigation and Routing:

1. **Navigator:**

- Definition: In Flutter, the Navigator class is responsible for managing a stack of Route objects, where each route represents a screen or page in the app.
- Purpose: It facilitates navigation between different parts of the app, managing transitions and maintaining a hierarchical structure of screens.

2. **MaterialPageRoute:**

- Definition: MaterialPageRoute is a commonly used route in Flutter that provides a standard Android/iOS page transition when navigating to a new screen.
- Usage: It is initiated by calling Navigator.of(context).push(MaterialPageRoute(builder: (context) => NextScreen()));.
- Benefit: Offers a consistent and platform-specific user experience.

3. **Named Routes:**

- Definition: Named routes provide a way to organize routes using strings instead of widget constructors.
- Usage: Define named routes in the routes parameter of MaterialApp.
- Navigation: Navigate to a named route using Navigator.pushNamed(context, '/next');.

Gestures:

1. **GestureDetector:**

- Definition: GestureDetector is a widget that enables gesture recognition in a subtree of the widget tree.
- Gestures: It supports various gestures like onTap, onDoubleTap, onLongPress, onPanUpdate, etc.
- Usage: Wrap the desired widget with GestureDetector and handle the required gestures.

2. **InkWell:**

- Definition: InkWell is a material design widget that provides a visual ink splash on tap.
- Usage: It simplifies the implementation of common tap gestures and adds a visual touch feedback.

3. **Draggable:**

- Definition: Draggable is a widget that allows dragging of its child within a designated area.
- Usage: It is useful for creating draggable components, like draggable tiles or cards.
- Callbacks: You can specify callbacks like onDraggableCanceled to handle events when dragging is canceled.

CODE:

```
class CartScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('My Cart'),
      ),
      body: Padding(
        padding: const EdgeInsets.all(10.0),
        child: SingleChildScrollView(
          scrollDirection: Axis.vertical,
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.center,
            children: [
              // Product 1
              _buildCartItem(
```

```
  imagePath: 'assets/watch.png',
  productName: 'boAt Watch Neo',
  stars: 4,
  discountPercentage: 75,
  originalPrice: '6499',
  discountedPrice: '1,599',
  offerText: '1 offer available',
),
// Divider between products
Divider(
  color: Colors.black,
  thickness: 1.0,
),
// Image below the divider
SizedBox(
  height: 250.0,
  width: 400.0,
  child: Padding(
    padding: EdgeInsets.only(top: 10.0, bottom: 5.0),
    child: Container(
      decoration: BoxDecoration(
        borderRadius: BorderRadius.circular(20.0),
        border: Border.all(
          color: Colors.black,
          width: 2.0,
        ),
      ),
      child: ClipRRect(
        borderRadius: BorderRadius.circular(18.0),
        child: Image.asset(
          'assets/warranty.jpeg',
          fit: BoxFit.fill,
        ),
      ),
    ),
  ),
),

SizedBox(height: 6.0),
// Price details
Padding(
  padding: EdgeInsets.only(bottom: 5.0),
  child: Text(
    'Price Details',
```

```
            style: TextStyle(
              fontSize: 24.0,
              color: Colors.black,
              fontWeight: FontWeight.bold,
            ),
          ),
        ),

        Row(
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: [
            Text(
              'Original Price:',
              style: TextStyle(
                fontSize: 18.0,
                color: Colors.black,
              ),
            ),
            Text(
              'Rs.6499',
              style: TextStyle(
                fontSize: 18.0,
                color: Colors.black,
              ),
            ),
          ],
        ),
        SizedBox(height: 10.0),
        // Calculating discounted price
        _buildDiscountedPrice(6499, 75),
        Divider(
          color: Colors.black,
          thickness: 1.0,
        ),
      ],
    ),
  ),
),
// Bottom Navigation Bar
bottomNavigationBar: SizedBox(
  height: 80.0,
  child: Container(
    margin: EdgeInsets.only(top: 21.0),
    child: BottomNavigationBar(
```

```
          items: const <BottomNavigationBarItem>[
            BottomNavigationBarItem(
              icon: Icon(Icons.shopping_cart),
              label: 'Cart',
            ),
            BottomNavigationBarItem(
              icon: Icon(Icons.home),
              label: 'Home',
            ),
            BottomNavigationBarItem(
              icon: Icon(Icons.account_circle),
              label: 'Account',
            ),
          ],
          selectedItemColor: Colors.blue,
          onTap: (index) {
            if (index == 2) {
              // Check if 'Account' is tapped
              Navigator.of(context).push(
                MaterialPageRoute(
                  builder: (context) => AccountScreen(),
                ),
              );
            }
            if (index == 1) {
              Navigator.of(context).push(
                MaterialPageRoute(
                  builder: (context) => HomeScreen(),
                ),
              ); // Go back to the HomeScreen
            }
          },
        ),
      ),
    ),
  );
}

Widget _buildDiscountedPrice(double originalPrice, int discountPercentage) {
  double discountedPrice = (discountPercentage / 100) * originalPrice;

  return Column(
    children: [
      Row(
```

```
    mainAxisAlignment: MainAxisAlignment.spaceBetween,
    children: [
      Text(
        'You Save:',
        style: TextStyle(
          fontSize: 18.0,
          color: Colors.black,
        ),
      ),
      Text(
        'Rs.${discountedPrice.toStringAsFixed(0)}',
        style: TextStyle(
          fontSize: 18.0,
          color: Colors.black,
        ),
      ),
    ],
  ),
  SizedBox(height: 10.0),
  // Display Discounted Price
  Row(
    mainAxisAlignment: MainAxisAlignment.spaceBetween,
    children: [
      Text(
        ' Discounted Price:',
        style: TextStyle(
          fontSize: 18.0,
          color: Colors.black,
        ),
      ),
      Text(
        'Rs.${discountedPrice.toStringAsFixed(0)}',
        style: TextStyle(
          fontSize: 18.0,
          color: Colors.black,
        ),
      ),
      Container(
        margin: EdgeInsets.symmetric(vertical: 0.0),
        child: ElevatedButton(
          onPressed: () {
            // Add functionality for placing an order
          },
          style: ElevatedButton.styleFrom(
```

```
                primary: Colors.yellow[700], // Yellow color
                padding:
                    EdgeInsets.symmetric(horizontal: 20.0, vertical: 10.0),
              ),
              child: Text(
                'Place Order',
                style: TextStyle(
                  fontSize: 18.0,
                  color: Colors.black,
                ),
              ),
            ),
          ),
        ],
      ),
    ],
  );
}

Widget _buildCartItem({
  required String imagePath,
  required String productName,
  required int stars,
  required int discountPercentage,
  required String originalPrice,
  required String discountedPrice,
  required String offerText,
}) {
  double discountedPriceValue =
      double.parse(originalPrice.replaceAll('', '')) -
          (discountPercentage / 100) *
              double.parse(originalPrice.replaceAll('', ''));

  return Container(
    margin: EdgeInsets.only(right: 16.0),
    child: Row(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        // Image on the left
        Image.asset(
          imagePath,
          height: 80.0,
          width: 80.0,
        ),
```

```
SizedBox(width: 20.0),
// Text on the right
Column(
  crossAxisAlignment: CrossAxisAlignment.start,
  children: [
    // Box below the image with "Qty: 1" inside
    SizedBox(height: 8.0),
    Text(
      productName,
      style: TextStyle(fontSize: 20.0),
    ),
    SizedBox(height: 8.0),
    // Star rating
    Row(
      children: List.generate(
        5,
        (index) => Icon(
          index < stars ? Icons.star : Icons.star_border,
          color: Colors.yellow[700],
        ),
      ),
    ),
    SizedBox(height: 8.0),
    // Discount details
    Row(
      children: [
        // Percentage text
        Text(
          '$discountPercentage%',
          style: TextStyle(
            fontSize: 20.0,
            color: Colors.yellow[700],
          ),
        ),
        SizedBox(width: 15.0),
        // Original price
        Text(
          'Rs.$originalPrice ',
          style: TextStyle(
            fontSize: 20.0,
            color: Colors.grey[600],
          ),
        ),
        SizedBox(width: 10.0),
```

```
                // Discounted price
                Text(
                  'Rs.${discountedPriceValue.toStringAsFixed(0)}',
                  style: TextStyle(
                    fontSize: 20.0,
                    color: Colors.black,
                  ),
                ),
              ],
            ),
            // Offer text
            Text(
              offerText,
              style: TextStyle(
                fontSize: 16.0,
                color: Colors.green,
              ),
            ),
          ],
        ),
      ],
    ),
  );
}
}
```

CONCLUSION:

Thus we studied navigation,routing and gesture features in Flutter Application.

MAD & PWA Lab Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	65
Name	PRAJAKTA UPADHYE
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	15

MAD PWA LAb 6

NAME: Prajakta Upadhye **Batch :** C

Class : D15A **Roll No.** : 65

Aim: To connect flutter with the firebase database.

To connect Firebase with Flutter, follow these steps:

1. Create a Firebase Project:
   - Go to the [Firebase Console](https://console.firebase.google.com/).
   - Click on "Add project" and follow the setup instructions.

2. Add Your App to Firebase Project:
   - After creating the project, click on "Add app" and select the appropriate platform (iOS/Android).
   - Follow the setup instructions to download the configuration files.

3. Add Dependencies in `pubspec.yaml`:
   - Open your `pubspec.yaml` file and add the following dependencies:

   ```yaml
   dependencies:
     firebase_core: ^latest_version
     firebase_auth: ^latest_version  # If you need authentication
     cloud_firestore: ^latest_version  # If you need Firestore
     firebase_storage: ^latest_version  # If you need Firebase Storage
   ```

   - Run `flutter pub get` in your terminal to download the dependencies.

4. Initialize Firebase in Your App:
   - In your main Dart file (usually `main.dart`), initialize Firebase in the `main` function:

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

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(MyApp());
}
```

5. Use Firebase Services:
   - Now, you can use Firebase services in your Flutter app. For example, if you're using Firestore:

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

final FirebaseFirestore _firestore = FirebaseFirestore.instance;

// Example: Reading data from Firestore
Future<void> fetchData() async {
  QuerySnapshot snapshot = await
_firestore.collection('your_collection').get();
  // Process snapshot data
}
```

6. Configure Firebase Authentication (Optional):
   - If you're using Firebase Authentication, follow the documentation to set up authentication methods.

7. Configure Firebase Storage (Optional):
   - If you're using Firebase Storage, follow the documentation to upload/download files.

Remember to check for the latest versions of Firebase packages on pub.dev and update your dependencies accordingly.

CODE:

```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter_application_1/firebase_options.dart';
import 'package:flutter_application_1/user_auth/firebase_auth_services.dart';
//import 'dart:ui' as ui;

// void main() {
//   runApp(const MyApp());
// }
Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(options: DefaultFirebaseOptions.currentPlatform);
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        body: HomePage(),
      ),
    );
  }
}

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

  @override
  State<HomePage> createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(body: LoginScreen());
  }
```

```
}

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

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

class _LoginScreenState extends State<LoginScreen> {
  final FirebaseAuthServices _auth = FirebaseAuthServices();
  // TextEditingController _mobilenoContoller = TextEditingController();
  // TextEditingController _usernameContoller = TextEditingController();
  TextEditingController _emailContoller = TextEditingController();
  TextEditingController _passwordContoller = TextEditingController();
  bool _passwordVisible = false;

  @override
  void dispose() {
    // TODO: implement dispose
    // _mobilenoContoller.dispose();
    // _usernameContoller.dispose();
    _emailContoller.dispose();
    _passwordContoller.dispose();
    super.dispose();
  }

  void _navigateToHomeScreen() {
    Navigator.of(context).push(
      MaterialPageRoute(
        builder: (context) => HomeScreen(),
      ),
    );
  }

  void _navigateToRegisterScreen() {
    Navigator.of(context).push(
      MaterialPageRoute(
        builder: (context) => RegisterScreen(),
      ),
    );
  }

  @override
```

```
Widget build(BuildContext context) {
  return Padding(
    padding: const EdgeInsets.all(5.0),
    child: ListView(
      children: [
        Stack(
          children: [
            Container(
              color: Colors.blueAccent,
              height: 40.0,
              width: double.infinity,
              margin: const EdgeInsets.only(top: 40.0),
            ),
            Positioned(
              left: 150.0,
              top: 40.0 + (30.0 - 16.0) / 2,
              child: Text(
                "FlipKart",
                style: TextStyle(
                  color: Colors.white,
                  fontSize: 16.0,
                  fontWeight: FontWeight.bold,
                ),
              ),
            ),
            SizedBox(width: 8.0),
            Positioned(
              top: 43.0,
              left: 215.0,
              child: Image.asset(
                'assets/flipkartlogo.jpg',
                height: 30.0,
                width: 30.0,
              ),
            ),
          ],
        ),
        Padding(
          padding: const EdgeInsets.only(left: 10.0, top: 25.0),
          child: Text(
            "Log in for the best experience",
            style: TextStyle(
              color: Colors.black,
              fontSize: 20.0,
```

```
        fontWeight: FontWeight.bold,
      ),
    ),
  ),
  Padding(
    padding: const EdgeInsets.only(left: 20.0, top: 50.0, right: 20.0),
    child: Container(
      decoration: BoxDecoration(
        border: Border.all(color: Colors.blueAccent),
        borderRadius: BorderRadius.circular(8.0),
      ),
      child: TextField(
        controller: _emailContoller,
        decoration: InputDecoration(
          hintText: 'Email',
          hintStyle: TextStyle(color: Colors.grey),
          border: InputBorder.none,
          contentPadding: EdgeInsets.all(15.0),
        ),
      ),
    ),
  ),
  // Padding(
  //   padding: const EdgeInsets.only(left: 20.0, top: 30.0, right: 20.0),
  //   child: Container(
  //     decoration: BoxDecoration(
  //       border: Border.all(color: Colors.blueAccent),
  //       borderRadius: BorderRadius.circular(8.0),
  //     ),
  //     child: TextField(
  //       controller:  _passwordContoller,
  //       decoration: InputDecoration(
  //         hintText: 'Password',
  //         hintStyle: TextStyle(color: Colors.grey),
  //         border: InputBorder.none,
  //         contentPadding: EdgeInsets.all(15.0),
  //       ),
  //     ),
  //   ),
  // ),
  Padding(
    padding: const EdgeInsets.only(left: 20.0, top: 30.0, right: 20.0),
    child: Container(
      decoration: BoxDecoration(
```

```
        border: Border.all(color: Colors.blueAccent),
        borderRadius: BorderRadius.circular(8.0),
      ),
      child: TextField(
        controller: _passwordContoller,
        obscureText: !_passwordVisible, // Obfuscate the password
        decoration: InputDecoration(
          hintText: 'Password',
          hintStyle: TextStyle(color: Colors.grey),
          border: InputBorder.none,
          contentPadding: EdgeInsets.all(15.0),
          suffixIcon: IconButton(
            icon: Icon(
              _passwordVisible
                  ? Icons.visibility
                  : Icons.visibility_off,
            ),
            onPressed: () {
              setState(() {
                _passwordVisible = !_passwordVisible;
              });
            },
          ),
        ),
      ),
    ),
  ),
),
Padding(
  padding: const EdgeInsets.only(left: 20.0, top: 30.0, right: 20.0),
  child: RichText(
    text: TextSpan(
      text: 'By continuing, you agree to FlipKart`s ',
      style: TextStyle(color: Colors.grey),
      children: [
        TextSpan(
          text: 'Terms of Use and Privacy Policy',
          style: TextStyle(color: Colors.blue),
        ),
      ],
    ),
  ),
),
Padding(
  padding: const EdgeInsets.only(top: 20.0),
```

```
            child: ElevatedButton(
              onPressed: () => _navigateToRegisterScreen(),
              style: ElevatedButton.styleFrom(
                primary: Colors.blue,
              ),
              child: Text(
                'Register',
                style: TextStyle(
                  color: Colors.white,
                  fontSize: 18.0,
                  fontWeight: FontWeight.bold,
                ),
              ),
            ),
          ),
          Padding(
            padding: const EdgeInsets.only(top: 10.0),
            child: ElevatedButton(
              onPressed: _signIn,
              style: ElevatedButton.styleFrom(
                primary: Colors.blue,
              ),
              child: Text(
                'Continue',
                style: TextStyle(
                  color: Colors.white,
                  fontSize: 18.0,
                  fontWeight: FontWeight.bold,
                ),
              ),
            ),
          ),
        ],
      ),
    );
  }

  void _signIn() async {
    // String username = _usernameContoller.text;
    // String mobileno = _mobilenoContoller.text;
    String email = _emailContoller.text;
    String password = _passwordContoller.text;

    User? user = await _auth.signInWithEmailandPassword(email, password);
```

```
    if (user != null) {
      print("User succesfully Signed in");
      _showDialog("Success", "User successfully signed in!");
      Navigator.of(context).pushReplacement(
        MaterialPageRoute(
          builder: (context) => HomeScreen(),
        ),
      );
    } else {
      print("Error in registeration");
      _showDialog("Error", "Invalid email or password. Please try again.");
    }
  }

  void _showDialog(String title, String content) {
    showDialog(
      context: context,
      builder: (BuildContext context) {
        return AlertDialog(
          title: Text(title),
          content: Text(content),
          actions: [
            TextButton(
              onPressed: () {
                Navigator.of(context).pop(); // Close the dialog
              },
              child: Text('OK'),
            ),
          ],
        );
      },
    );
  }
}

class RegisterScreen extends StatefulWidget {
  @override
  _RegisterScreenState createState() => _RegisterScreenState();
}

class _RegisterScreenState extends State<RegisterScreen> {
  final FirebaseAuthServices _auth = FirebaseAuthServices();
  TextEditingController _mobilenoContoller = TextEditingController();
```

```
TextEditingController _usernameContoller = TextEditingController();
TextEditingController _emailContoller = TextEditingController();
TextEditingController _passwordContoller = TextEditingController();
bool _passwordVisible = false;

@override
void dispose() {
  // TODO: implement dispose
  _mobilenoContoller.dispose();
  _usernameContoller.dispose();
  _emailContoller.dispose();
  _passwordContoller.dispose();
  super.dispose();
}

@override
Widget build(BuildContext context) {
  return Material(
    child: Scaffold(
      appBar: AppBar(),
      body: SingleChildScrollView(
        // Wrap with SingleChildScrollView
        physics: BouncingScrollPhysics(), // Enable scrolling
        child: Padding(
          padding: const EdgeInsets.only(top: 0.0),
          child: Stack(
            children: [
              Column(
                mainAxisAlignment: MainAxisAlignment.start,
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  Stack(
                    children: [
                      Container(
                        color: Colors.blueAccent,
                        height: 40.0,
                        width: double.infinity,
                        margin: const EdgeInsets.only(top: 10.0),
                      ),
                      Positioned(
                        left: 150.0,
                        top: 10.0 + (30.0 - 16.0) / 2,
                        child: Text(
                          "FlipKart",
```

```
          style: TextStyle(
            color: Colors.white,
            fontSize: 16.0,
            fontWeight: FontWeight.bold,
          ),
        ),
      ),
      SizedBox(
        width: 8.0, // Add some space between text and image
      ),
      Positioned(
        top: 13.0,
        left: 215.0,
        child: Image.asset(
          'assets/flipkartlogo.jpg',
          height: 30.0,
          width: 30.0,
        ),
      ),
    ],
  ),
  Padding(
    padding: const EdgeInsets.only(left: 10.0, top: 15.0),
    child: Text(
      "Register for the best experience",
      style: TextStyle(
        color: Colors.black,
        fontSize: 20.0,
        fontWeight: FontWeight.bold,
      ),
    ),
  ),
  Padding(
    padding: const EdgeInsets.only(left: 10.0, top: 1.0),
    child: Text(
      "Enter your phone number to continue",
      style: TextStyle(
        color: Colors.grey,
        fontSize: 14.0,
      ),
    ),
  ),
  Padding(
    padding: const EdgeInsets.only(
```

```
      left: 20.0, top: 30.0, right: 20.0),
    child: Container(
      decoration: BoxDecoration(
        border: Border.all(color: Colors.blueAccent),
        borderRadius: BorderRadius.circular(8.0),
      ),
      child: TextField(
        controller: _mobilenoContoller,
        decoration: InputDecoration(
          hintText: 'Mobile Number',
          hintStyle: TextStyle(color: Colors.grey),
          border: InputBorder.none,
          contentPadding: EdgeInsets.all(15.0),
        ),
      ),
    ),
  ),
  Padding(
    padding: const EdgeInsets.only(left: 260.0, top: 5),
    child: Text(
      "Use Email-ID",
      style: TextStyle(
        color: Colors.blue[800],
        fontSize: 15.0,
        fontWeight: FontWeight.bold,
      ),
    ),
  ),
  Padding(
    padding: const EdgeInsets.only(
        left: 20.0, top: 30.0, right: 20.0),
    child: Container(
      decoration: BoxDecoration(
        border: Border.all(color: Colors.blueAccent),
        borderRadius: BorderRadius.circular(8.0),
      ),
      child: TextField(
        controller: _usernameContoller,
        decoration: InputDecoration(
          hintText: 'UserName',
          hintStyle: TextStyle(color: Colors.grey),
          border: InputBorder.none,
          contentPadding: EdgeInsets.all(15.0),
        ),
```

```
      ),
    ),
  ),
  Padding(
    padding: const EdgeInsets.only(
        left: 20.0, top: 10.0, right: 20.0),
    child: Container(
      decoration: BoxDecoration(
        border: Border.all(color: Colors.blueAccent),
        borderRadius: BorderRadius.circular(8.0),
      ),
      child: TextField(
        controller: _emailContoller,
        decoration: InputDecoration(
          hintText: 'Email',
          hintStyle: TextStyle(color: Colors.grey),
          border: InputBorder.none,
          contentPadding: EdgeInsets.all(15.0),
        ),
      ),
    ),
  ),
  Padding(
    padding: const EdgeInsets.only(
        left: 20.0, top: 10.0, right: 20.0),
    child: Container(
      decoration: BoxDecoration(
        border: Border.all(color: Colors.blueAccent),
        borderRadius: BorderRadius.circular(8.0),
      ),
      child: TextField(
        controller: _passwordContoller,
        obscureText:
            !_passwordVisible, // Obfuscate the password
        decoration: InputDecoration(
          hintText: 'Password',
          hintStyle: TextStyle(color: Colors.grey),
          border: InputBorder.none,
          contentPadding: EdgeInsets.all(15.0),
          suffixIcon: IconButton(
            icon: Icon(
              _passwordVisible
                  ? Icons.visibility
                  : Icons.visibility_off,
```

```
          ),
          onPressed: () {
            setState(() {
              _passwordVisible = !_passwordVisible;
            });
          },
        ),
      ),
    ),
  ),
),
Padding(
  padding: const EdgeInsets.only(
      left: 20.0, top: 20.0, right: 20.0),
  child: RichText(
    text: TextSpan(
      text: 'By continuing, you agree to FlipKart`s ',
      style: TextStyle(color: Colors.grey),
      children: [
        TextSpan(
          text: 'Terms of Use and Privacy Policy',
          style: TextStyle(color: Colors.blue),
        ),
      ],
    ),
  ),
),
Padding(
  padding: const EdgeInsets.only(top: 40.0, left: 140.0),
  child: ElevatedButton(
    onPressed: () {
      _register();
      String mobileNumber = _mobilenoContoller.text;
      String userName = _usernameContoller.text;
      String email = _emailContoller.text;
      String password = _passwordContoller.text;

      // Print the values for testing (you can replace this with your logic)
      print('Mobile Number: $mobileNumber');
      print('User Name: $userName');
      print('Email: $email');
      print('Password: $password');

      //  Navigator.of(context).pushReplacement(
```

```
            //   MaterialPageRoute(
            //     builder: (context) => HomeScreen(),
            //   ),
            // );
            // Add the action for the blue button here
            // For example, you can navigate to another screen
            print('Blue button pressed!');
          },
          style: ElevatedButton.styleFrom(
            primary: Colors.blue,
          ),
          child: Text(
            'Submit',
            style: TextStyle(
              color: Colors.white,
              fontSize: 18.0,
              fontWeight: FontWeight.bold,
            ),
          ),
        ),
      ),

      // Padding(
      //   padding: const EdgeInsets.only(top: 0.0, left: 100.0),
      //   child: TextButton(
      //     onPressed: () {
      //       // Add the action for the text button here
      //       // For example, you can navigate to the login screen
      //       print('Text button pressed!');
      //     },
      //     child: Text(
      //       'Already have an account? Login',
      //       style: TextStyle(
      //         color: Colors.blue,
      //         fontSize: 16.0,
      //         fontWeight: FontWeight.bold,
      //       ),
      //     ),
      //   ),
      // ),
    ],
  ),
],
),
```

```
          ),
        ),
      ),
    );
  }

  void _register() async {
  String username = _usernameContoller.text;
  String mobileno = _mobilenoContoller.text;
  String email = _emailContoller.text;
  String password = _passwordContoller.text;

  try {
    User? user = await _auth.registerWithEmailandPassword(email, password);

    if (user != null) {
      print("User successfully registered");
      Navigator.of(context).pushReplacement(
        MaterialPageRoute(
          builder: (context) => HomeScreen(),
        ),
      );
      addUserDetails(
        _usernameContoller.text.trim(),
        _mobilenoContoller.text.trim(),
        _emailContoller.text.trim(),
        _passwordContoller.text.trim(),
      );
    } else {
      print("Error in registration");
    }
  } catch (e) {
    print("Error during registration: $e");
    // Display a pop-up indicating that the user already exists
    _showErrorDialog("User Already Exists",
        "The provided email is already registered. Please use a different email or login.");
  }
}

void _showErrorDialog(String title, String message) {
  showDialog(
    context: context,
    builder: (BuildContext context) {
      return AlertDialog(
```

```
        title: Text(title),
        content: Text(message),
        actions: [
          TextButton(
            onPressed: () {
              Navigator.of(context).pop(); // Close the dialog
            },
            child: Text('OK'),
          ),
        ],
      );
    },
  );
}


  Future addUserDetails(
      String username, String email, String mobileno, String password) async {
    await FirebaseFirestore.instance.collection('accountinfo').add({
      'username': username,
      'email': email,
      'mobileno': mobileno,
      'password': password,
      'address': '',
    });
  }
}
```

Project Title: Flutter:Flipkart Clone/ PWA:To do List **Roll No. 65**

CONCLUSION:

Thus we connected firebase to flutter.

MAD & PWA Lab Journal

Experiment No.	07
Experiment Title.	To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”.
Roll No.	65
Name	PRAJAKTA UPADHYE
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	15

MAD PWA LAb 7

NAME: Prajakta Upadhye **Batch :** C

Class : D15A **Roll No.** : 65

AIM: To write meta data of your Ecommerce PWA in a Web app manifest file to enable "add to homescreen feature"

THEORY:

Regular Web App

A regular web app is a website that is designed to be accessible on all mobile devices such that the content gets fit as per the device screen. It is designed using a web technology stack (HTML, CSS, JavaScript, Ruby, etc.) and operates via a browser. They offer various native-device features and functionalities. However, it entirely depends on the browser the user is using. In other words, it might be possible that you can access a native-device feature on Chrome but not on Safari or Mozilla Firefox because the browsers are incompatible with that feature

Progressive Web App

Progressive Web App (PWA) is a regular web app, but some extras enable it to deliver an excellent user experience. It is a perfect blend of desktop and mobile application experience to give both platforms to the end-users.

manifest.json

```
{
   "short_name": "React App",
   "name": "Create React App Sample",
   "icons": [
     {
       "src": "favicon.ico",
       "sizes": "64x64 32x32 24x24 16x16",
       "type": "image/x-icon"
     },
     {
       "src": "logo192.png",
       "type": "image/png",
       "sizes": "192x192"
     },
     {
       "src": "logo512.png",
       "type": "image/png",
       "sizes": "512x512"
     }
   ],
   "start_url": ".",
   "display": "standalone",
   "theme_color": "#000000",
   "background_color": "#ffffff"
 }
```

serviceworker.js

```
var staticCacheName = "pwa";

self.addEventListener("install", function (e) {
e.waitUntil(
   caches.open(staticCacheName).then(function (cache) {
   return cache.addAll(["/"]);
   })
```

```
);
});

self.addEventListener("fetch", function (event) {
console.log(event.request.url);

event.respondWith(
   caches.match(event.request).then(function (response) {
   return response || fetch(event.request);
   })
);
});
```

public/index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8" />
  <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <meta name="theme-color" content="#000000" />
  <meta name="description" content="Web site created using create-react-app" />
  <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
  <!--
     manifest.json provides metadata used when your web app is installed on a
     user's mobile device or desktop. See
https://developers.google.com/web/fundamentals/web-app-manifest/
   -->
  <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
  <!--
     Notice the use of %PUBLIC_URL% in the tags above.
     It will be replaced with the URL of the `public` folder during the build.
     Only files inside the `public` folder can be referenced from the HTML.

     Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
     work correctly both with client-side routing and a non-root public URL.
     Learn how to configure a non-root public URL by running `npm run build`.
   -->

  <!-- Bootstrap CSS -->
```

```
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.rtl.min.css"

integrity="sha384-PJsj/BTMqILvmcej7ulplguok8ag4xFTPryRq8xevL7eBYSmpXKcbNVuy+P0
RMgq" crossorigin="anonymous">

  <title>iNoteBook</title>
</head>

<body>
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root"></div>
  <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run `npm start` or `yarn start`.
      To create a production bundle, use `npm run build` or `yarn build`.
    -->

  <!-- Option 1: Bootstrap Bundle with Popper -->
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"

integrity="sha384-geWF76RCwLtnZ8qwWowPQNguL3RmwHVBC9FhGdlKrxdiJJigb/j/68SIy
3Te4Bkz"
    crossorigin="anonymous"></script>

    <script src="https://kit.fontawesome.com/e7a7193d30.js" crossorigin="anonymous"></script>

    <script>
        window.addEventListener("load", () => {
          registerSW();
        });

        // Register the Service Worker
        async function registerSW() {
          if ("serviceWorker" in navigator) {
```

```
        try {
          await navigator;
          ServiceWorker;
          register("serviceworker.js");
        } catch (e) {
          console.log("SW Registration Failed.");
        }
      }
    }
  </script>
</body>
</html>
```

COMMAND: **npm run start**

```
You can now view inotebook in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.0.104:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

Open localhost:3000 on the browser

Click on 3 dots on the top right corner of the browser → go to **Save and Share** → Click on **Install Create React App Sample**

You will get the following pop-up:

Click on **Install**.

App Icon will appear at the bottom:

CONCLUSION:

Hence, we learnt how to write a metadata of our PWA in a Web App Manifest File to enable add to homescreen feature.

MAD & PWA Lab Journal

Experiment No.	08
Experiment Title.	To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA
Roll No.	65
Name	PRAJAKTA UPADHYE
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

MAD PWA LAb 8

NAME: Prajakta Upadhye **Batch :** C

Class : D15A **Roll No.** : 65

Aim: To code and register a service worker, and complete the install and activation process for a new service worker for a PWA

Theory:

Service Worker
Service Worker is a script that works on browser background without user interaction independently. Also, It resembles a proxy that works on the user side. With this script, you can track network traffic of the page, manage push notifications and develop “offline first” web applications with Cache API.

Things to note about Service Worker:

- A service worker is a programmable network proxy that lets you control how network requests from your page are handled.
- Service workers only run over HTTPS. Because service workers can intercept network requests and modify responses, "man-in-the-middle" attacks could be very bad.
- The service worker becomes idle when not in use and restarts when it's next needed. You cannot rely on a global state persisting between events. If there is information that you need to persist and reuse across restarts, you can use IndexedDB databases.

What can we do with Service Workers?

- You can dominate **Network Traffic**
  You can manage all network traffic of the page and do any manipulations. For example, when the page requests a CSS file, you can send plain text as a response or when the page requests an HTML file, you can send a png file as a response. You can also send a true response too.

- You can **Cache**
  You can cache any request/response pair with Service Worker and Cache API and you can access these offline content anytime.

- You can manage **Push Notifications**
  You can manage push notifications with Service Worker and show any information message to the user.
- You can **Continue**
  Although Internet connection is broken, you can start any process with Background Sync of Service Worker.

What can't we do with Service Workers?

- You can't access the **Window**
  You can't access the window, therefore, You can't manipulate DOM elements. But, you can communicate to the window through post Message and manage processes that you want.

- You can't work it on **80 Port**
  Service Worker just can work on HTTPS protocol. But you can work on localhost during development.

Service Worker Cycle

A service worker goes through three steps in its life cycle:

- Registration
- Installation
- Activation

My Project's folder structure:

Registration

To install a service worker, you need to register it in your main JavaScript code. Registration tells the browser where your service worker is located, and to start installing it in the background.

CODE:

public/index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop. See
https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.
```

```
    Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
    work correctly both with client-side routing and a non-root public URL.
    Learn how to configure a non-root public URL by running `npm run build`.
  -->

  <!-- Bootstrap CSS -->
  <link
    rel="stylesheet"
    href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.rtl.min.css"

integrity="sha384-PJsj/BTMqILvmcej7ulplguok8ag4xFTPryRq8xevL7eBYSmpXKcbNVuy+P0
RMgq"
    crossorigin="anonymous"
  />

  <title>iNoteBook</title>
 </head>

 <body>
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root"></div>
  <!--
    This HTML file is a template.
    If you open it directly in the browser, you will see an empty page.

    You can add webfonts, meta tags, or analytics to this file.
    The build step will place the bundled scripts into the <body> tag.

    To begin the development, run `npm start` or `yarn start`.
    To create a production bundle, use `npm run build` or `yarn build`.
  -->

  <!-- Option 1: Bootstrap Bundle with Popper -->
  <script
    src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"

integrity="sha384-geWF76RCwLtnZ8qwWowPQNguL3RmwHVBC9FhGdlKrxdiJJigb/j/68SIy
3Te4Bkz"
    crossorigin="anonymous"
  ></script>
```

```
  <script
    src="https://kit.fontawesome.com/e7a7193d30.js"
    crossorigin="anonymous"
  ></script>

  <script>
    window.addEventListener("load", () => {
      registerSW();
    });

    // Register the Service Worker
    async function registerSW() {
      if ("serviceWorker" in navigator) {
        try {
          const registration = await navigator.serviceWorker.register(
            "serviceworker.js"
          );
          console.log("Registered Service Worker:", registration);
        } catch (error) {
          console.log("SW Registration Failed:", error);
        }
      }
    }
  </script>
 </body>
</html>
```

Installation

Once the browser registers a service worker, installation can be attempted. This occurs if the service worker is considered to be new by the browser, either because the site currently doesn't have a registered service worker, or because there is a byte difference between the new service worker and the previously installed one.

A service worker installation triggers an install event in the installing service worker. We can include an install event listener in the service worker to perform some task when the service worker installs. For instance, during the install, service workers can precache parts of a web app so that it loads instantly the next time a user opens it (see caching the application shell). So, after that first load, you're going to benefit from instant repeat loads and your time to

interactivity is going to be even better in those cases. An example of an installation event listener looks like this:

```
self.addEventListener("install", function (e) {
  e.waitUntil(
    caches.open(staticCacheName).then(function (cache) {
      return cache.addAll(["/"]);
    })
  );
});
```

~This is done in serviceworker.js

Activation

Once a service worker has successfully installed, it transitions into the activation stage. If there are any open pages controlled by the previous service worker, the new service worker enters a waiting state. The new service worker only activates when there are no longer any pages loaded that are still using the old service worker. This ensures that only one version of the service worker is running at any given time.

When the new service worker activates, an activate event is triggered in the activating service worker. This event listener is a good place to clean up outdated caches.

```
self.addEventListener('activate', event => {
  event.waitUntil(
    caches.keys().then(cacheNames => {
      return Promise.all(
        cacheNames.filter(name => {
          return name !== staticCacheName;
        }).map(name => {
          return caches.delete(name);
        })
      );
    })
  );
});
```

~This is done in serviceworker.js

<u>Therefore, serviceworker.js finally looks like this:</u>

```
var staticCacheName = "pwa";
```

```
self.addEventListener("install", function (e) {
  e.waitUntil(
    caches.open(staticCacheName).then(function (cache) {
      return cache.addAll(["/"]);
    })
  );
});

self.addEventListener('activate', event => {
    event.waitUntil(
        caches.keys().then(cacheNames => {
            return Promise.all(
                cacheNames.filter(name => {
                    return name !== staticCacheName;
                }).map(name => {
                    return caches.delete(name);
                })
            );
        })
    );
});

self.addEventListener("fetch", function (event) {
  console.log(event.request.url);

  event.respondWith(
    caches.match(event.request).then(function (response) {
      return response || fetch(event.request);
    })
  );
});
```

We now run the app using the following command:
> npm run start

```
You can now view inotebook in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.0.104:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

On the browser:

Conclusion:

The aim was to implement a service worker for aProgressive Web Application (PWA), enabling offline functionality and improving performance. This involved coding and registering the service worker (serviceworker.js) to intercept network requests, cache essential resources, and enhance the offline experience. By completing the install and activation process, the PWA gained features like offline access and faster page loads, enhancing user experience and resilience to network issues.

MAD & PWA Lab Journal

Experiment No.	09
Experiment Title.	To implement Service worker events like fetch, sync and push for E-commerce PWA
Roll No.	65
Name	PRAJAKTA UPADHYE
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

MAD PWA LAb 9

NAME: Prajakta Upadhye **Batch :** C

Class : D15A **Roll No.** : 65

Aim: To implement Service worker events like fetch, sync and push for E-commerce PWA.

Theory:
Service Worker
Service Worker is a script that works on browser background without user interaction independently. Also, It resembles a proxy that works on the user side. With this script, you can track network traffic of the page, manage push notifications and develop "offline first" web applications with Cache API.

Things to note about Service Worker:

- A service worker is a programmable network proxy that lets you control how network requests from your page are handled.
- Service workers only run over HTTPS. Because service workers can intercept network requests and modify responses, "man-in-the-middle" attacks could be very bad.
- The service worker becomes idle when not in use and restarts when it's next needed. You cannot rely on a global state persisting between events. If there is information that you need to persist and reuse across restarts, you can use IndexedDB databases.
- Service workers make extensive use of promises, so if you're new to promises, then you should stop reading this and check out Promises, an introduction.

Fetch Event
You can track and manage page network traffic with this event. You can check existing cache, manage "cache first" and "network first" requests and return a response that you want.

Of course, you can use many different methods but you can find in the following example a "cache first" and "network first" approach. In this example, if the request's and current location's origin are the same (Static content is requested.), this is called

"cacheFirst" but if you request a targeted external URL, this is called "networkFirst".

- **CacheFirst** - In this function, if the received request has cached before, the cached response is returned to the page. But if not, a new response requested from the network.
- **NetworkFirst** - In this function, firstly we can try getting an updated response from the network, if this process completed successfully, the new response will be cached and returned. But if this process fails, we check whether the request has been cached before or not. If a cache exists, it is returned to the page, but if not, this is up to you. You can return dummy content or information messages to the page.

Sync Event

Background Sync is a Web API that is used to delay a process until the Internet connection is stable. We can adapt this definition to the real world; there is an e-mail client application that works on the browser and we want to send an email with this tool. Internet connection is broken while we are writing e-mail content and we didn't realize it. When completing the writing, we click the send button.

Here is a job for the Background Sync.
The following view shows the classical process of sending email to us. If the Internet Connection is broken, we can't send any content to Mail Server.

1. When we click the "send" button, email content will be saved to IndexedDB.
2. Background Sync registration.
3. **If the Internet connection is available**, all email content will be read and sent to Mail Server.

   If the Internet connection is unavailable, the service worker waits until the connection is available even though the window is closed. When it is available, email content will be sent to Mail Server.

   You can see the working process within the following code block.

Push Event

This is the event that handles push notifications that are received from the server. You can apply any method with received data.

We can check in the following example.
"Notification.requestPermission();" is the necessary line to show notification to the

user. If you don't want to show any notification, you don't need this line.

In the following code block is in sw.js file. You can handle push notifications with this event. In this example, I kept it simple. We send an object that has "method" and "message" properties. If the method value is "pushMessage", we open the information notification with the "message" property.

serviceworker.js

```
var staticCacheName = "pwa";

self.addEventListener("install", function (event) {
  event.waitUntil(preLoad());
});

self.addEventListener("fetch", function (event) {
  event.respondWith(
    checkResponse(event.request).catch(function () {
      console.log("Fetch from cache successful!");
      return returnFromCache(event.request);
    })
  );
  console.log("Fetch successful!");
  event.waitUntil(addToCache(event.request));
});

self.addEventListener("sync", (event) => {
  if (event.tag === "syncMessage") {
    console.log("Sync successful!");
  }
});

self.addEventListener("push", function (event) {
    if (event && event.data) {
      var data = event.data.json();
      if (data.method == "pushMessage") {
        console.log("Push notification requested");

        if (Notification.permission === "granted") {
          event.waitUntil(
            self.registration.showNotification("To Do List", {
              body: data.message,
```

```
          }).catch(function(error) {
            console.error("Error showing notification:", error);
          })
        );
      } else if (Notification.permission === "denied") {
        console.error("Notification permission denied.");
        // Handle the case where notification permission is denied, such as showing a message to
the user.
      } else {
        Notification.requestPermission().then(function(permission) {
          if (permission === "granted") {
            console.log("Notification permission granted, showing notification");
            self.registration.showNotification("To Do List", {
              body: data.message,
            });
          } else {
            console.error("Notification permission denied.");
            // Handle the case where notification permission is denied after requesting.
          }
        });
      }
    }
  }
});


var filesToCache = ["/login", "/Home", "/About", "/index.html"];

var preLoad = function () {
  return caches.open("index").then(function (cache) {
    return cache.addAll(filesToCache);
  });
};

var checkResponse = function (request) {
  return new Promise(function (fulfill, reject) {
    fetch(request)
      .then(function (response) {
        if (response.status !== 404) {
          fulfill(response);
        } else {
          reject();
        }
      })
```

```
      .catch(reject);
  });
};

var addToCache = function (request) {
  return caches.open("index").then(function (cache) {
    return fetch(request).then(function (response) {
      return cache.put(request, response);
    });
  });
};

var returnFromCache = function (request) {
  return caches.open("index").then(function (cache) {
    return cache.match(request).then(function (matching) {
      if (!matching || matching.status == 404) {
        return cache.match("index.html");
      } else {
        return matching;
      }
    });
  });
};

self.addEventListener("activate", (event) => {
  event.waitUntil(
    caches.keys().then((cacheNames) => {
      return Promise.all(
        cacheNames
          .filter((name) => {
            return name !== staticCacheName;
          })
          .map((name) => {
            return caches.delete(name);
          })
      );
    })
  );
});
```

Fetch Event :

Sync Event :

Push Event:

```
Push notification requested                                  serviceworker.js:29
▼Notification permission denied.                             serviceworker.js:42
  (anonymous) @ serviceworker.js:42
```

Go to Site Settings and allow Notifications:

Now, we can see the desired result:

CONCLUSION:

- The aim is to enhance the functionality and user experience of a Progressive Web App (PWA) by implementing service worker events such as fetch, sync, and push.
- These events enable the app to efficiently cache resources for offline access, synchronize data in the background, and deliver timely notifications to users, thereby improving performance, reliability, and engagement.

MAD & PWA Lab Journal

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	65
Name	PRAJAKTA UPADHYE
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

MAD PWA LAb 10

NAME: Prajakta Upadhye **Batch :** C

Class : D15A **Roll No.** : 65

AIM:
To study and implement deployment of Ecommerce PWA to GitHub Pages.

THEORY:

GitHub Pages
Public web pages are freely hosted and easily published. Public webpages hosted directly from your GitHub repository. Just edit, push, and your changes are live.

GitHub Pages provides the following key features:
1. Blogging with Jekyll
2. Custom URL
3. Automatic Page Generator

Reasons for favoring this over Firebase:
1. Free to use
2. Right out of github
3. Quick to set up

GitHub Pages is used by Lyft, CircleCI, and HubSpot.
GitHub Pages is listed in 775 company stacks and 4401 developer stacks.

Pros
1. Very familiar interface if you are already using GitHub for your projects.
2. Easy to set up. Just push your static website to the gh-pages branch and your website is ready.
3. Supports Jekyll out of the box.
4. Supports custom domains. Just add a file called CNAME to the root of your site, add an A record in the site's DNS configuration, and you are done.

Cons
1. The code of your website will be public, unless you pay for a private repository.
2. Currently, there is no support for HTTPS for custom domains. It's probably coming soon though.
3. Although Jekyll is supported, plug-in support is rather spotty.

Firebase

The Realtime App Platform. Firebase is a cloud service designed to power real-time, collaborative applications. Simply add the Firebase library to your application to gain access to a shared data structure; any changes you make to that data are automatically synchronized with the Firebase cloud and with other clients within milliseconds.

Some of the features offered by Firebase are:

1. Add the Firebase library to your app and get access to a shared data structure. Any changes made to that data are automatically synchronized with the Firebase cloud and with other clients within milliseconds.
2. Firebase apps can be written entirely with client-side code, update in real-time out-of-the-box, interoperate well with existing services, scale automatically, and provide strong data security.
3. Data Accessibility- Data is stored as JSON in Firebase. Every piece of data has its own URL which can be used in Firebase's client libraries and as a REST endpoint. These URLs can also be entered into a browser to view the data and watch it update in real-time.

Reasons for favoring over GitHub Pages:

1. Realtime backend made easy
2. Fast and responsive

Instacart, 9GAG, and Twitch are some of the popular companies that use Firebase
Firebase has a broader approval, being mentioned in 1215 company stacks & 4651 developers stacks

Pros

1. Hosted by Google. Enough said.
2. Authentication, Cloud Messaging, and a whole lot of other handy services will be available to you.
3. A real-time database will be available to you, which can store 1 GB of data.
4. You'll also have access to a blob store, which can store another 1 GB of data.
5. Support for HTTPS. A free certificate will be provisioned for your custom domain within 24 hours.

Cons

1. Only 10 GB of data transfer is allowed per month. But this is not really a big problem, if you use a CDN or AMP.

2. Command-line interface only.
3. No in-built support for any static site generator.

Link to our GitHub repository: [susmitasanti/inote (github.com)]

HOSTED LINK:
[iNoteBook (susmitasanti.github.io)]

HOSTED IMAGE:

Project Title: Flutter:Flipkart Clone/ PWA:To do List **Roll No. 65**

MAD & PWA Lab Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	65
Name	PRAJAKTA UPADHYE
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	15

MAD PWA LAb 11

NAME: Prajakta Upadhye **Batch :** C

Class : D15A **Roll No.** : 65

AIM : To use google Lighthouse PWA Analysis Tool to test the PWA functioning.

THEORY:

Google Lighthouse :

Google Lighthouse is a tool that lets you audit your web application based on a number of parameters including (but not limited to) performance, based on a number of metrics, mobile compatibility, Progressive Web App (PWA) implementations, etc. All you have to do is run it on a page or pass it a URL, sit back for a couple of minutes and get a very elaborate report, not much short of one that a professional auditor would have compiled in about a week.

The best part is that you have to set up almost nothing to get started. Let's begin by looking at some of the top features and audit criteria used by Lighthouse.

Key Features and Audit Metrics

Google Lighthouse has the option of running the Audit for Desktop as well as mobile version of your page(s). The top metrics that will be measured in the Audit are:

1. **Performance:** This score is an aggregation of how the page fared in aspects such as (but not limited to) loading speed, time taken for loading for basic frame(s), displaying meaningful content to the user, etc. To a layman, this score is indicative of how decently the site performs, with a score of 100 meaning that you figure in the 98th percentile, 50 meaning that you figure in the 75th percentile and so on.
2. **PWA Score (Mobile):** Thanks to the rise of Service Workers, app manifests, etc., a lot of modern web applications are moving towards the PWA paradigm, where the objective is to make the application behave as close as possible to native mobile applications. Scoring points are based on the Baseline PWA checklist laid down by Google which includes Service Worker implementation(s), viewport handling, offline functionality, performance in script-disabled environments, etc.
3. **Accessibility:** As you might have guessed, this metric is a measure of how accessible your website is, across a plethora of accessibility

features that can be implemented in your page (such as the 'aria-' attributes like aria-required, audio captions, button names, etc.). Unlike the other metrics though, Accessibility metrics score on a pass/fail basis i.e. if all possible elements of the page are not screen-reader friendly (HTML5 introduced features that would make pages easy to interpret for screen readers used by visually challenged people like tag names, tags such as <section>, <article>, etc.), you get a 0 on that score. The aggregate of these scores is your Accessibility metric score.

4. **Best Practices:** As any developer would know, there are a number of practices that have been deemed 'best' based on empirical data. This metric is an aggregation of many such points, including but not limited to:Use of HTTPS
   Avoiding the use of deprecated code elements like tags, directives, libraries, etc. Password input with paste-into disabled
   Geo-Location and cookie usage alerts on load, etc.

Click on the 3 dots on the top-right corner of the page → Click on **More Tools** → Click on **Developer Tools**

Click on **Analyze page load**

Project Title: Flutter:Flipkart Clone/ PWA:To do List **Roll No. 65**

Project Title: Flutter:Flipkart Clone/ PWA:To do List **Roll No. 65**

Made the following changes in **manifest.json**

```
{
  "short_name": "React App",
  "name": "Create React App Sample",
  "icons": [
    {
      "src": "favicon.ico",
      "sizes": "64x64 32x32 24x24 16x16",
      "type": "image/x-icon"
    },
    {
      "src": "logo192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
    {
      "src": "logo512.png",
      "type": "image/png",
      "sizes": "512x512",
      "purpose":"maskable"
    },
```

```
    {
      "src": "download.png",
      "type": "image/png",
      "sizes": "144x144",
      "purpose": "any"
    }
  ],
  "start_url": ".",
  "display": "standalone",
  "theme_color": "#000000",
  "background_color": "#ffffff"
}
```

Conclusion: Thus we successfully used google Lighthouse PWA Analysis Tool for testing the PWA functioning.

MAD & PWA Lab Journal

Experiment No.	Assignment-1
Assignment 1 Questions	1. Flutter Overview:Explain the key features and advantages of using Flutter for mobile app development. Discuss how the Flutter framework differs from traditional approaches and why it has gained popularity in the developer community. 2. Widget Tree and Composition: Describe the concept of the widget tree in Flutter. Explain how widget composition is used to build complex user interfaces. Provide examples of commonly used widgets and their roles in creating a widget tree. 3. State Management in Flutter: Discuss the importance of state management in Flutter applications. Compare and contrast the different state management approaches available in Flutter, such as setState, Provider, and Riverpod. Provide scenarios where each approach is suitable. 4. Firebase Integration in Flutter: Explain the process of integrating Firebase with a Flutter application. Discuss the benefits of using Firebase as a backend solution. Highlight the Firebase services commonly used in Flutter development and provide a brief overview of how data synchronization is achieved.
Roll No.	65
Name	PRAJAKTA UPADHYE
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	5

Prajakta . K . Upadhye
D15A – Batch C
Roll No – 65

Assignment – 1

Q.1. Flutter overview: Explain key features & advantages of using flutter for mobile app development. Discuss how it differs from traditional framework & why it has gained popularity in developer community.

ans. 1) a) Features.

i) Single codebase for multiple platforms:
Allows developers to write code & deploy it on both iOS & Android platforms.

ii) Hot Reload
This enables developers to instantly see the results of the code changes they make.

iii) Expresive UI
Flexiblity to create expressive & flexible UIs

iv) Integration with other tools:

b) Advantages:

1. Faster development as it uses single codebase for multiple platforms.
2. widgets provide consistent look & feel across different platforms.
3. cost efficient for developing & maintaining single codebase for iOS & Android – reduces cost.

c) Difference from traditional approach.

1) Traditional approach uses heirarchial structure whereas a widget based approach is used by flutter
2) Flutter complies to native ARM code providing performances comparable to native applications.

3) Hot Reload allows to see changes made instantly
Flutter popularity is driven by increased productivity, growing community, flexibility in UI.

Q.2) Describe widget tree. Explain how widget composition is used to build complex user interface. Provides examples of commonly used widgets & their roles in creating a widget tree.

ans.) widget Tree:
1. ~~widget~~
i) widget tree is a heirarchial structure of widgets that defines the user interface of an application.
ii) Every visual element from simple components to complex layouts is representated by widget.
iii widget can be categorized in 2 types:
a) Stateless : immutable. , eg: images, text
b) Statefull : can change over time , eg: buttons, form

2. Widget composition:
i widget composition in flutter involves combining multiple single widgets to create more complex & compound widgets.
ii This composability is a powerful concept that allows developers to build sophisticated UI by nesting widgets with each other.

(a) Container:
A box model for padding, margin & decoration.

(b) Column & row:
Layout widget for arranging children vertically or horizontally.

(c) Stack:
Overlapping widgets, allowing them to be layered on top of each other.

(d) Listview:
A scrollable list of widgets

(e) Grid view:
A scrollable grid of widgets.

(f) Appbar:
A material design app bar typically at top of screen.

(g) Textfield:
An input field for users to enter text.

(h) Button widgets
Interactive buttons for user widgets

Q3)

ans. State management is crucial in flutter applications because it involves managing the data that can change over time.
Flutter is reactive, UI builds when data changes.

	setState	Provider	Riverpod
1.	Built-in flutter method	External package named 'provider'	External package riverpod.
2.	Local state within widget	Global state within widget tree.	Global state with additional features.
3.	Limited scalability for large apps.	Suitable for medium sized apps.	Designed for large & complex apps.
4.	May lead to code redundancy	Balances simplicity & readability	Emphasizes readability & clean future.

Scenario where each is applicable:

① setState:
For small to moderately complex applications.
manages localState within widget.
eg: forms,

② Provider:
- For medium to large sized applications.
- When centralized state is needed accessible by multiple widgets

eg: managing user authentication, theme change.

③ Riverpod:
- For large & complex applications.
- When testability & maintainability are top priorities.

eg: complex application with multiple feature.

Q4) Integration:
1) Go to Firebase console & create a new project.
2) Add firebase SDK by including dependencies in pubspec.yaml

```
dependencies:
    firebase_core : ^version
    firebase_auth : ^version
    cloud_firestore : ^version
```

3) Run flutter pub get.
4) Initialize firebase by calling Firebase Initialise App() : in main

→ Benefits:
1) It offers real-time NoSQL database
2) Provides secure & easy to implement solution for user authentication.

3) Provides a secure & easy to implement scalable NoSQL database that allows to store & sync data in real time.

4) Provides simple & efficient way to deploy & host web applications.

Data synchronization:

1. Real When data changes on one client, it triggers events that automatically update data on other clients.
2. It notifies clients when data changes allowing for seamless real time updates.
3. If user sign in or out on one device, the authentication state is automatically reflected on other devices.

MAD & PWA Lab Journal

Experiment No.	Assignment-2
Assignment 2 Questions	1. Define Progressive Web App (PWA) and explain its significance in modern web development.Discuss the key characteristics that differentiate PWAs from traditional mobile apps 2. Define responsive web design and explain its importance in the context of Progressive Web Apps.Compare and contrast responsive, fluid, and adaptive web design approaches. 3. Describe the lifecycle of Service Workers, including registration, installation, and activation phases. 4. Explain the use of IndexedDB in the Service Worker for data storage.
Roll No.	65
Name	PRAJAKTA UPADHYE
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO4:Understand various PWA frameworks and their requirements LO5: Design and Develop a responsive User Interface by applying PWA Design techniques LO6:Develop and Analyze PWA Features and deploy it over app hosting solutions
Grade:	4

Prajakta. K. Upadhye
D15A
65

1) Define Progressive web App (PWA) & explain its significance in modern web development. Discuss key characteristics that differentiate PWA from traditional mobile apps.

ans.1. PWA is a type of web application that leverages modern web capabilities to provide users with an app like experience directly through web browsers. They are built using web technologies such as HTML, CSS, & Javascript & are intended to offer a seamless & engaging user experience.

4

significance:

1. Cross platform compatibility:
PWA can run on any device with a compatible browser, eliminating the need to develop seperate versions for different platforms like ios & Android.

2. Offline Functionality:
PWA can work offline or with a poor internet connection thanks to technologies like service workers that cache essential resources.

3. App like experience:
They mimic look & feel of native mobile apps offering features such as push notification home screen installation & smooth animations.

4. Improved Performance:
They are optimized for performance, resulting in faster load times & smoother interactions leading to better user engagement & retention.

Key characteristics:

1. Platform independent:
   PWA are accessed through web browser & can run on any platform whereas traditional mobile apps are platform specific.
2. Installation:
   PWA can be installed on a device's home screen like native app.
3. Updates:
   can be automatically updated whenever user accesses them.
4. Offline functionality:
   Can function offline or with limited connectivity.

Q2) Define responsive web designs & explain its importance in context of Progressive Web Apps. Compare & contract responsive, fluid & adaptive web designs approaches.

ans. Responsive web design is an approach to web development that ensures web pages render well on various devices & window or screen sizes. It involves using flexible layouts & grids, responsive images & CSS media queries to adapt layout & design of a website to viewing environment providing an optimal user.

Importance:

1. Responsive Web Design: Uses fluid grids, flexible images & media queries to adapt the layout & design of a website to different screen sizes & devices dynamically.

2. Fluid web Design:
Similar to responsive design, fluid design uses percentages for widths & heights instead of fixed units, allowing elements to resize smoothly as viewport changes.

3. Adaptive web Designs:
Involves creating multiple fixed layout sizes designed for specific system / screen sizes or device categories. The website detects the user's device & serves the appropriate layout optimizing device experience for that specific device.

Q3) Describe the lifecycle of Service workers, including registeration, installation & activation phases.

ans-1. Services workers are scripts that run in the background of web application, seperate from main browser thread. They enable features such as push notifications, background sync & offline functionality in PWA.

Registration:
The service worker script is registered with the browser by web application. This usually occurs in main Javascript file of application.

Installation:
The browser downloads & caches the service worker script.

Activation:
After installation, service workers enters activation phase. It takes control of pages it was registered to handle, allowing it to intercept network requests, manage caching & perform other tasks.

Q4) Explain the use of IndexedDB in service worker for data storage.

ans-1. IndexedDB is a low-level API for client side storage of significant amounts of structured data, including files & blobs. Service workers can utilize IndexedDB to store data locally, enabling PWAs to work offline & provide a seamless user experience even with limited or no internet connectivity.

2. Service workers can intercept network requests & cache responses in IndexedDB, allowing the application to serve content from the cache when offline or to improve performance by serving cached content when the network is slow.

3. IndexedDB provides a persistant storage solution that allows PWAs to store data locally across sessions, ensuring that users can access content even when they are not connected to the internet.