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 **Preeti Vinayak Khamkar** 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 Analyse 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	17/01	24/01	12
2.	To design Flutter UI by including common widgets.	LO2	24/01	31/01	15
3.	To include icons, images, fonts in Flutter app	LO2	31/01	07/02	15
4.	To create an interactive Form using form widget	LO2	07/02	14/02	15
5.	To apply navigation, routing and gestures in Flutter App	LO2	14/02	21/02	15
6.	To Connect Flutter UI with fireBase database	LO3	21/02	06/03	15
7.	To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”.	LO4	06/03	13/03	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	13/03	20/03	15
9.	To implement Service worker events like fetch, sync and push for E-commerce PWA	LO5	20/03	27/03	15
10.	To study and implement deployment of Ecommerce PWA to GitHub Pages.	LO5	27/03	29/03	15
11.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.	LO6	27/03	29/03	15
12.	Assignment-1	LO1,LO2 ,LO3	02/02	05/02	5
13.	Assignment-2	LO4,LO5 ,LO6	19/03	21/03	4

MAD & PWA Lab Journal

Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	28
Name	Preeti Khamkar
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	12

Name: Preeti Khamkar **Div:** D15A **Roll No.:** 28

Aim: Installation and Configuration of Flutter Environment.
Create a 'Hello World App' using Flutter.

Installation Steps:

1. Install Flutter SDK for Windows

2. Set Environmental Variables

3. $flutter command

```
Command Prompt
Microsoft Windows [Version 10.0.19042.1415]
(c) Microsoft Corporation. All rights reserved.

C:\Users\jalpa>flutter
Manage your Flutter app development.

Common commands:

  flutter create <output directory>
    Create a new Flutter project in the specified directory.

  flutter run [options]
    Run your Flutter application on an attached device or in an emulator.

Usage: flutter <command> [arguments]

Global options:
-h, --help                  Print this usage information.
-v, --verbose               Noisy logging, including all shell commands executed.
                            If used with "--help", shows hidden options. If used with "flutter doctor", shows additional                    diagnostic informati
on. (Use "-vv" to force verbose logging in those cases.)
-d, --device-id             Target device id or name (prefixes allowed).
    --version               Reports the version of this tool.
    --suppress-analytics    Suppress analytics reporting when this command runs.

Available commands:

Flutter SDK
  bash-completion   Output command line shell completion setup scripts.
  channel           List or switch Flutter channels.
  config            Configure Flutter settings.
  doctor            Show information about the installed tooling.
  downgrade         Downgrade Flutter to the last active version for the current channel.
  precache          Populate the Flutter tool's cache of binary artifacts.
  upgrade           Upgrade your copy of Flutter.

Project
  analyze           Analyze the project's Dart code.
  assemble          Assemble and build Flutter resources.
  build             Build an executable app or install bundle.
  clean             Delete the build/ and .dart_tool/ directories.
  create            Create a new Flutter project.
  drive             Run integration tests for the project on an attached device or emulator.
  format            Format one or more Dart files.
```

4. $flutter doctor

```
\Users\jalpa>
\Users\jalpa>
\Users\jalpa>flutter doctor
nning "flutter pub get" in flutter_tools...                    17.0s
ctor summary (to see all details, run flutter doctor -v):
] Flutter (Channel stable, 2.8.1, on Microsoft Windows [Version 10.0.19042.1415], locale en-US)
] Android toolchain - develop for Android devices
  X Unable to locate Android SDK.
    Install Android Studio from: https://developer.android.com/studio/index.html
    On first launch it will assist you in installing the Android SDK components.
    (or visit https://flutter.dev/docs/get-started/install/windows#android-setup for detailed instructions).
    If the Android SDK has been installed to a custom location, please use
    `flutter config --android-sdk` to update to that location.

] Chrome - develop for the web
] Android Studio (not installed)
] VS Code (version 1.55.2)
] Connected device (2 available)

Doctor found issues in 2 categories.

\Users\jalpa>flutter doctor
ctor summary (to see all details, run flutter doctor -v):
] Flutter (Channel stable, 2.8.1, on Microsoft Windows [Version 10.0.19042.1415], locale en-US)
] Android toolchain - develop for Android devices (Android SDK version 32.0.0)
  X cmdline-tools component is missing
    Run `path/to/sdkmanager --install "cmdline-tools;latest"`
    See https://developer.android.com/studio/command-line for more details.
  X Android license status unknown.
    Run `flutter doctor --android-licenses` to accept the SDK licenses.
    See https://flutter.dev/docs/get-started/install/windows#android-setup for more details.
] Chrome - develop for the web
] Android Studio (version 2020.3)
] VS Code (version 1.55.2)
] Connected device (2 available)

Doctor found issues in 1 category.
```

5. Install Android Studio and Android SDK

6. Create Virtual Device

Code:

Main.dart file

```
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 to Flutter'),
      ),
      body: const Center(
        child: Text('Hello, Preeti Khamkar!'),
      ),
    ),
  );
}
}
```

Output:

MAD & PWA Lab Journal

Experiment No.	02
Experiment Title.	To design Flutter UI by including common widgets.
Roll No.	28
Name	Preeti Khamkar
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

Experiment - 2

Name: Preeti Khamkar **Division:** D15A **Roll no. :** 28

Aim: To design Flutter UI by including common widgets.

Theory:
In Flutter, there are several common widgets that are used to build user interfaces. These widgets serve as the building blocks for creating various UI elements. Here are some of the most commonly used ones:

1. Container: A versatile widget that can contain other widgets and provides properties for styling, padding, margin, alignment, and more.
2. Text: Displays a string of text with properties for styling such as font size, color, alignment, and font weight.
3. Image: Displays an image from various sources such as assets, the internet, or memory.
4. Row: Arranges its children widgets horizontally in a row.
5. Column: Arranges its children widgets vertically in a column.
6. Stack: Stacks its children widgets on top of each other, allowing for overlapping UI elements.
7. ListView: Displays a scrollable list of children widgets.
8. GridView: Displays a scrollable grid of children widgets arranged in rows and columns.
9. AppBar: A material design app bar that typically contains a title, actions, and/or navigation elements.
10. Scaffold: Implements the basic material design visual layout structure, providing a framework for the app's UI.
11. IconButton: A button widget that displays an icon and triggers an action when pressed.
12. TextField: Allows users to input text and provides properties for customization such as keyboard type, input validation, and styling.
13. RaisedButton and FlatButton: Button widgets that trigger an action when pressed, with raised and flat styles respectively.
14. AlertDialog: Displays a modal dialog box with a message and optional actions.
15. BottomNavigationBar: Provides navigation between top-level views in the app, typically displayed at the bottom of the screen.

These are just a few examples of the many widgets available in Flutter. You can combine these widgets and customize their properties to create complex and beautiful user interfaces for your Flutter applications.

Syntax:

1. Container Widget

```
Container(
  // Properties

  // Child widget
  child: Text(),
)
```

2. Text Widget

```
Text(
  style: TextStyle(
  ),
)
```

3. RaisedButton Widget

```
RaisedButton(
  onPressed: () {
    // Action to perform when the button is pressed
  },
)
```

4. Scaffold Widget

```
Scaffold(
  appBar: AppBar(
  ),
)
```

Widget and Properties:

TextButton, TetField, AppBar, Container, Scaffold are the widgets used in this experiment.

Code and Output:

```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:telegram/screen/chatPage.dart';
import '../contacts.dart';
import '../search.dart';
import 'drawer/settings.dart';
import 'package:intl/intl.dart';

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

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

 // Make the method public
 static String constructChatRoomId(String phoneNumber1, String phoneNumber2) {
   List<String> phoneNumbers = [phoneNumber1, phoneNumber2];
   phoneNumbers.sort();
   return phoneNumbers.join("");
 }
}

class _HomePageState extends State<HomePage> {
 final FirebaseAuth _auth = FirebaseAuth.instance;

 void signOut() async {
   await _auth.signOut();
   // Navigate to login screen
   Navigator.pushReplacementNamed(context, '/login');
 }

 @override
 Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(
       title: Text('Telegram', style: TextStyle(color: Colors.white)),
       backgroundColor: Color(0xFF588FBA),
       actions: [
         IconButton(
           icon: Icon(Icons.search),
           onPressed: () {
             // Navigate to search screen
             Navigator.push(
```

```
          context,
          MaterialPageRoute(
            builder: (context) => Search(),
          ),
        );
      },
    ),
  ],
  iconTheme: IconThemeData(color: Colors.white), // Set the color of drawer icon to white
),

drawer: Drawer(
  child: StreamBuilder<DocumentSnapshot>(
    stream: FirebaseFirestore.instance
        .collection('users')
        .doc(_auth.currentUser!.uid)
        .snapshots(),
    builder: (context, snapshot) {
      if (snapshot.hasError) {
        return Text('Error: ${snapshot.error}');
      }
      if (snapshot.connectionState == ConnectionState.waiting) {
        return Center(child: CircularProgressIndicator());
      }

      var userData = snapshot.data!.data() as Map<String, dynamic>;
      var profileImageUrl = userData['profileImageUrl'] ?? '';
      var username = userData['username'] ?? '';
      var phoneNumber = userData['phone'] ?? '';

      return ListView(
        padding: EdgeInsets.zero,
        children: [
          DrawerHeader(
            decoration: BoxDecoration(color: Color(0xFF588FBA)), // Changed to #588FBA
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                CircleAvatar(
                  backgroundImage: profileImageUrl.isNotEmpty
                      ? NetworkImage(profileImageUrl)
                      : null,
                  backgroundColor:
                  Colors.orange, // Default background color
```

```
        radius: 30, // Adjust as needed
        child: profileImageUrl.isEmpty
            ? Text(
          username[0].toUpperCase(), // Display the first letter of the username
          style: TextStyle(
            color: Colors.white,
            fontSize: 24, // Adjust the font size as needed
            fontWeight: FontWeight.bold, // Make the font bold
          ),
        )
            : null,
      ),
      SizedBox(height: 8),
      Row(
        children: [
          Expanded(
            child: Text(
              username,
              style:
              TextStyle(color: Colors.white, fontSize: 18),
            ),
          ),
          Icon(Icons.keyboard_arrow_down,
              color: Colors.white), // Arrow color set to white
        ],
      ),
      Text(
        phoneNumber,
        style: TextStyle(color: Colors.white),
      ),
    ],
  ),
),
ListTile(
  leading:
  Icon(Icons.group, color: Colors.grey), // Changed icon color
  title: Text('New Group'),
  onTap: () {
    // Navigate to new group screen
  },
),
ListTile(
  leading: Icon(Icons.contacts,
      color: Colors.grey), // Changed icon color
```

```
  title: Text('Contacts'),
  onTap: () {
    // Navigate to Contacts screen
  },
),
ListTile(
  leading: Icon(Icons.call,
      color: Colors.grey), // Changed icon color
  title: Text('Call'),
  onTap: () {
    // Navigate to Contacts screen
  },
),
ListTile(
  leading: Icon(Icons.emoji_people_outlined,
      color: Colors.grey), // Changed icon color
  title: Text('People Nearby'),
  onTap: () {
    // Navigate to People Nearby screen
  },
),
ListTile(
  leading: Icon(Icons.save,
      color: Colors.grey), // Changed icon color
  title: Text('Saved Messages'),
  onTap: () {
    // Navigate to Saved Messages screen
  },
),
ListTile(
  leading: Icon(Icons.settings,
      color: Colors.grey), // Changed icon color
  title: Text('Settings'),
  onTap: () {
    // Navigate to settings screen
    Navigator.push(
        context,
        MaterialPageRoute(
            builder: (context) => ProfilePage(
                userId: _auth.currentUser!.uid)));
  },
),
Divider(
  color: Colors.grey[300], // Added a light grey line separator
```

```
          thickness: 1,
        ),
        ListTile(
          leading: Icon(Icons.person_add_outlined,
              color: Colors.grey), // Changed icon color
          title: Text('Invite Friends'),
          onTap: () {
            // Invite Friends
          },
        ),
        ListTile(
          leading: Icon(Icons.help_outline,
              color: Colors.grey), // Changed icon color
          title: Text('Telegram Features'),
          onTap: () {
            signOut();
          },
        ),
        ListTile(
          leading: Icon(Icons.logout,
              color: Colors.grey), // Changed icon color
          title: Text('Logout'),
          onTap: () {
            signOut();
          },
        ),
      ],
    );
  },
),
),
body: _buildUserList(),
floatingActionButton: FloatingActionButton(
  onPressed: () {
    // Navigate to ContactsPage
    Navigator.push(
      context,
      MaterialPageRoute(
        builder: (context) => ContactsPage(), // Replace ContactsPage with the actual page
      ),
    );
  },
  child: Icon(Icons.edit, color: Colors.white,),
  backgroundColor: Color(0xFF3BACEB),
```

```
      shape: CircleBorder(),// Adjust the color to match the theme
    ),
    floatingActionButtonLocation: FloatingActionButtonLocation.endFloat,
  );
}


// build list of users except the current user
Widget _buildUserList() {
  return StreamBuilder<QuerySnapshot>(
    stream: FirebaseFirestore.instance.collection('users').snapshots(),
    builder: (context, snapshot) {
      if (snapshot.hasError) {
        return Text('Error: ${snapshot.error}');
      }
      if (snapshot.connectionState == ConnectionState.waiting) {
        return Center(child: CircularProgressIndicator());
      }

      return ListView(
        children: snapshot.data!.docs
            .map<Widget>((doc) => _buildUserListItem(doc))
            .toList(),
      );
    },
  );
}

// build individual user list items
Widget _buildUserListItem(DocumentSnapshot document) {
  Map<String, dynamic>? data = document.data() as Map<String, dynamic>?;

  if (_auth.currentUser != null &&
      _auth.currentUser!.phoneNumber != null &&
      data != null) {
    if (data['phone'] != null &&
        _auth.currentUser!.phoneNumber != data['phone']) {
      String displayName = data['username'] ?? data['phone'];

      return StreamBuilder<QuerySnapshot>(
        stream: FirebaseFirestore.instance
            .collection('chat_rooms')
            .doc(HomePage.constructChatRoomId(_auth.currentUser!.phoneNumber.toString(),
data['phone']))
```

```
    .collection('messages')
    .orderBy('timestamp', descending: true)
    .limit(1)
    .snapshots(),
  builder: (context, snapshot) {
   if (snapshot.hasData &&
      snapshot.data!.docs.isNotEmpty) {
    var lastMessage = snapshot.data!.docs.first;
    var timestamp = lastMessage['timestamp'];
    var messageTimestamp = DateTime.fromMillisecondsSinceEpoch(timestamp.seconds * 1000);
    var formattedTimestamp = DateFormat.jm().format(messageTimestamp);

    return Column(
      children: [
        ListTile(
          title: Text(
            displayName,
            style: TextStyle(fontWeight: FontWeight.bold),
          ), // Make the username bold
          subtitle: Text(
            lastMessage['message'],
            maxLines: 1,
            overflow: TextOverflow.ellipsis,
            style: TextStyle(color: Colors.grey), // Set text color to grey
          ),
          trailing: Text(
            formattedTimestamp,
            style: TextStyle(color: Colors.grey), // Set text color to grey
          ),
          leading: CircleAvatar(
            backgroundImage: data['profileImageUrl'] != null
                ? NetworkImage(data['profileImageUrl'])
                : null,
            backgroundColor: Colors.orange, // Default background color
            radius: 40, // Increased the radius to make the profile icon bigger
            child: data['profileImageUrl'] == null
                ? Text(
              displayName[0].toUpperCase(), // Display the first letter of the username
              style: TextStyle(
                color: Colors.white,
                fontSize: 24, // Adjust the font size as needed
                fontWeight: FontWeight.bold, // Make the font bold
              ),
            )
```

```
                : null, // Show the first letter only when no profile image is available
              ),
              onTap: () {
                Navigator.push(
                  context,
                  MaterialPageRoute(
                    builder: (context) => ChatPage(
                      receivedUserPhoneNumber: data['phone'] ?? '',
                    ),
                  ),
                );
              },
            ),
            Divider(
              color: Colors.grey[300], // Added a light grey line separator
              thickness: 1,
            ),
          ],
        );
      } else {
        return SizedBox.shrink(); // Return an empty SizedBox if there are no messages
      }
    },
  );
 }
}
return SizedBox.shrink(); // Return an empty SizedBox if user data is invalid
}
}
```

Telegram
P
Preeti
hello
9:52 PM
A
Annabeth
good
10:02 PM

MAD & PWA Lab Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	28
Name	Preeti Khamkar
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

Experiment - 3

Name: Preeti Khamkar **Division:** D15A **Roll no. :** 28

Aim: To include icons, images, fonts in Flutter app

Theory:
In Flutter, you can easily incorporate icons, images, and custom fonts into your app to enhance its visual appeal and user experience.

- Icons:
  Icons are small graphical symbols used to represent actions, objects, or concepts in a graphical user interface. In Flutter, icons can be represented using the Icon widget, which allows developers to easily include built-in icons provided by the Flutter framework. These icons are accessed through the Icons class, which offers a wide range of commonly used symbols.
- Images:
  Images are visual representations that can enhance the visual appeal and functionality of a Flutter app. In Flutter, images can be displayed using the Image widget, which supports various image sources, including assets, network URLs, and memory. To use images in a Flutter app, developers typically include image files in the assets directory of their project and specify them in the pubspec.yaml file. This allows Flutter to bundle the images with the app and access them programmatically. Images can be loaded asynchronously for improved performance.
- Fonts:
  Fonts play a crucial role in shaping the visual appearance and readability of text in a Flutter app. In addition to the default system fonts, Flutter allows developers to use custom fonts to achieve a unique and consistent typography style. Custom fonts are included in a Flutter app by adding font files (typically TrueType or OpenType) to the assets directory and specifying them in the pubspec.yaml file. Once added, developers can apply custom fonts to text widgets using the TextStyle class, which provides properties for font family, size, weight, and other typographic attributes. Custom fonts help maintain brand identity, improve readability, and enhance the overall design aesthetic of a Flutter app.

Syntax:

1. Icon-
   Icon(Icons.favorite); // Using a built-in icon
   For custom icons import them as assets.

2. Image-
   Image.asset()
   //include your image files in the assets section of your pubspec.yaml file.

3. Font-
   TextStyle()

Add the necessary configurations in your pubspec.yaml file to include assets and fonts:

```
flutter:
  assets:
    - assets/images/
    - assets/fonts/
  fonts:
    - family:
      fonts:
        - asset: assets/fonts/fontName.ttf
          weight: <weight>
```

Widget and Properties:
In-built Phone icon, telegram logo image are used.

Code and Output:

```
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'login/phone_verify_screen.dart';

class LoginScreen extends StatefulWidget {
 @override
 State<StatefulWidget> createState() => LoginScreenState();
}

class LoginScreenState extends State<LoginScreen>{
 var phoneCt = TextEditingController();
 @override
```

```
 Widget build(BuildContext context){
  return Scaffold(
    backgroundColor: Colors.white,
    appBar: PreferredSize(
      preferredSize: const Size.fromHeight(0),
      child: Container(),
    ),
    body: SizedBox(
      width: double.infinity,
      height: double.infinity,
      child: Center(
        child: Column(
          mainAxisSize: MainAxisSize.min,
          children: <Widget>[
            Align(alignment: Alignment.topRight, child:
            TextButton(onPressed: (){
              Navigator.push(
                  context,
                  MaterialPageRoute(
                      builder: (context) =>
                          PhoneVerifyScreen()));
            },
              child: const Text("Next", style: TextStyle(fontWeight: FontWeight.bold), textDirection:
TextDirection.ltr,),

            ),),
            const Spacer(),
            SizedBox(
              width: 150,
              child: Image.asset('assets/images/telegram_logo.png'),
            ),
            Container(height: 30,),
            const Text("Your Phone", style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold,
color: Colors.black),textDirection: TextDirection.ltr, ),
            Container(height: 15,),
            const Text('Please confirm your country code \nand enter your phone number.',
textAlign: TextAlign.center, textDirection: TextDirection.ltr,),
            Container(height: 30,),
            Container(
              margin: const EdgeInsets.only(left: 25, right: 25),
```

```
            child: TextField(

              textAlign: TextAlign.center,
              controller: phoneCt,
              maxLength: 12,
              maxLines: 1,
              keyboardType: TextInputType.phone,
              decoration: const InputDecoration(
                prefixIcon: Icon(Icons.phone),
                hintText: 'Your phone number'
              ),
            ),
          ),

          const Spacer(),
        ],
      ),
    ),
  ),
);
}
}
```

MAD & PWA Lab
Journal

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget
Roll No.	28
Name	Preeti Khamkar
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

Experiment - 4

Name: Preeti Khamkar **Division:** D15A **Roll no. :** 28

Aim: To create an interactive Form using form widget

Theory:
In Flutter, forms are essential for capturing user input and interacting with various widgets. Forms help in validation, submission, and overall management of user data input. They are typically composed of form fields, which are widgets that allow users to enter data, and often include buttons for submitting or resetting the form.

1. Form Widget: The Form widget is the container for form elements. It maintains the form state, which includes validation, error messages, and values of the form fields.
2. Form Field Widget: Form fields represent input fields like text fields, checkboxes, radio buttons, etc. They are managed by the Form widget and handle validation and state changes internally.
3. Validation: Forms typically include validation logic to ensure that the data entered by the user is correct. Flutter provides built-in validation mechanisms and allows you to define custom validators.
4. Handling Form Submission: Once the user fills out the form and submits it, you need to handle the submitted data. This can involve processing the data, performing further actions, or sending it to a server.

Syntax:

```
Form(
  key: _formKey, // GlobalKey<FormState>
  autovalidateMode: AutovalidateMode.onUserInteraction,
  child: Column(
    children: <Widget>[
      // Form fields go here
    ],
  ),
)
```

Widget and Properties:
In-built Phone icon, telegram logo image are used.

Code and Output:

```
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'login/phone_verify_screen.dart';

class LoginScreen extends StatefulWidget {
 @override
 State<StatefulWidget> createState() => LoginScreenState();
}

class LoginScreenState extends State<LoginScreen>{
 var phoneCt = TextEditingController();
 @override
 Widget build(BuildContext context){
  return Scaffold(
    backgroundColor: Colors.white,
    appBar: PreferredSize(
      preferredSize: const Size.fromHeight(0),
      child: Container(),
    ),
    body: SizedBox(
      width: double.infinity,
      height: double.infinity,
      child: Center(
        child: Column(
          mainAxisSize: MainAxisSize.min,
          children: <Widget>[
            Align(alignment: Alignment.topRight, child:
            TextButton(onPressed: (){
              Navigator.push(
                  context,
                  MaterialPageRoute(
                      builder: (context) =>
                          PhoneVerifyScreen()));
            },
              child: const Text("Next", style: TextStyle(fontWeight: FontWeight.bold), textDirection:
TextDirection.ltr,),
```

```
          ),),
          const Spacer(),
          SizedBox(
            width: 150,
            child: Image.asset('assets/images/telegram_logo.png'),
          ),
          Container(height: 30,),
          const Text("Your Phone", style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold,
color: Colors.black),textDirection: TextDirection.ltr, ),
          Container(height: 15,),
          const Text('Please confirm your country code \nand enter your phone number.',
textAlign: TextAlign.center, textDirection: TextDirection.ltr,),
          Container(height: 30,),
          Container(
            margin: const EdgeInsets.only(left: 25, right: 25),
            child: TextField(

              textAlign: TextAlign.center,
              controller: phoneCt,
              maxLength: 12,
              maxLines: 1,
              keyboardType: TextInputType.phone,
              decoration: const InputDecoration(
                  prefixIcon: Icon(Icons.phone),
                  hintText: 'Your phone number'
              ),
            ),
          ),

          const Spacer(),
         ],
        ),
      ),
    ),
  );
}
}
```

localhost:59121
Back
Next
DEBUG
+911234567890
We just sent you an SMS with the code.
Code
0/12
+Haven't received the code?

localhost:59121
Next
DEBUG
Your Phone
Please confirm your country code
and enter your phone number.
Your phone number
0/12

MAD & PWA Lab Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	28
Name	Preeti Khamkar
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

Experiment - 5

Name: Preeti Khamkar **Division:** D15A **Roll no. :** 28

Aim: To apply navigation, routing and gestures in Flutter App

Theory:
Navigation and routing are fundamental concepts in Flutter for managing the flow of screens or "routes" within an application. They allow users to move between different parts of the app, such as navigating from a home screen to a settings screen, or from a product list to a product detail page.

1. Navigation: Navigation refers to the action of moving between different screens or routes in an application. This can be triggered by user interactions like tapping on a button, or programmatically in response to certain events.
2. Routes: In Flutter, a route represents a distinct screen or page within the application. Each route has its own widget tree and can be pushed onto a navigation stack or popped off it.
3. Navigator: The Navigator widget manages a stack of routes. It allows you to push new routes onto the stack, pop existing routes off the stack, or replace routes with new ones. The Navigator typically resides at the root of the widget tree and facilitates navigation within the app.
4. Routing: Routing refers to the process of defining and managing the mapping between route names (or route objects) and their corresponding widgets in the application. Flutter provides mechanisms for both named routing and dynamic routing.

Syntax:

```
Navigator.push(
        context,
        MaterialPageRoute(builder: (context) => const SecondRoute()),
      );
```

Widget and Properties:

1. Drawer: The Drawer widget provides a panel that slides in from the left side of the screen and contains navigation options and user information. It includes user

details like profile image, username, phone number, and options like creating a new group, accessing contacts, settings, etc.

2. StreamBuilder: It's used to listen to changes in Firestore collections and update the UI accordingly. In this code, it's used to fetch user data and display a list of users.
3. ListView: Displays a list of users fetched from Firestore.
4. Navigation: Navigation between screens is handled using Navigator.push() and MaterialPageRoute.

Code and Output:

```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:telegram/screen/chatPage.dart';
import '../contacts.dart';
import '../search.dart';
import 'drawer/settings.dart';
import 'package:intl/intl.dart';

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

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

 // Make the method public
 static String constructChatRoomId(String phoneNumber1, String phoneNumber2) {
   List<String> phoneNumbers = [phoneNumber1, phoneNumber2];
   phoneNumbers.sort();
   return phoneNumbers.join("");
 }
}

class _HomePageState extends State<HomePage> {
 final FirebaseAuth _auth = FirebaseAuth.instance;

 void signOut() async {
   await _auth.signOut();
   // Navigate to login screen
   Navigator.pushReplacementNamed(context, '/login');
 }
```

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Telegram', style: TextStyle(color: Colors.white)),
      backgroundColor: Color(0xFF588FBA),
      actions: [
        IconButton(
          icon: Icon(Icons.search),
          onPressed: () {
            // Navigate to search screen
            Navigator.push(
              context,
              MaterialPageRoute(
                builder: (context) => Search(),
              ),
            );
          },
        ),
      ],
      iconTheme: IconThemeData(color: Colors.white), // Set the color of drawer icon to white
    ),

    drawer: Drawer(
      child: StreamBuilder<DocumentSnapshot>(
        stream: FirebaseFirestore.instance
            .collection('users')
            .doc(_auth.currentUser!.uid)
            .snapshots(),
        builder: (context, snapshot) {
          if (snapshot.hasError) {
            return Text('Error: ${snapshot.error}');
          }
          if (snapshot.connectionState == ConnectionState.waiting) {
            return Center(child: CircularProgressIndicator());
          }

          var userData = snapshot.data!.data() as Map<String, dynamic>;
          var profileImageUrl = userData['profileImageUrl'] ?? '';
          var username = userData['username'] ?? '';
          var phoneNumber = userData['phone'] ?? '';

          return ListView(
            padding: EdgeInsets.zero,
```

```
children: [
  DrawerHeader(
    decoration: BoxDecoration(color: Color(0xFF588FBA)), // Changed to #588FBA
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        CircleAvatar(
          backgroundImage: profileImageUrl.isNotEmpty
              ? NetworkImage(profileImageUrl)
              : null,
          backgroundColor:
          Colors.orange, // Default background color
          radius: 30, // Adjust as needed
          child: profileImageUrl.isEmpty
              ? Text(
            username[0].toUpperCase(), // Display the first letter of the username
            style: TextStyle(
              color: Colors.white,
              fontSize: 24, // Adjust the font size as needed
              fontWeight: FontWeight.bold, // Make the font bold
            ),
          )
              : null,
        ),
        SizedBox(height: 8),
        Row(
          children: [
            Expanded(
              child: Text(
                username,
                style:
                TextStyle(color: Colors.white, fontSize: 18),
              ),
            ),
            Icon(Icons.keyboard_arrow_down,
                color: Colors.white), // Arrow color set to white
          ],
        ),
        Text(
          phoneNumber,
          style: TextStyle(color: Colors.white),
        ),
      ],
    ),
```

```
),
ListTile(
  leading:
  Icon(Icons.group, color: Colors.grey), // Changed icon color
  title: Text('New Group'),
  onTap: () {
    // Navigate to new group screen
  },
),
ListTile(
  leading: Icon(Icons.contacts,
      color: Colors.grey), // Changed icon color
  title: Text('Contacts'),
  onTap: () {
    // Navigate to Contacts screen
  },
),
ListTile(
  leading: Icon(Icons.call,
      color: Colors.grey), // Changed icon color
  title: Text('Call'),
  onTap: () {
    // Navigate to Contacts screen
  },
),
ListTile(
  leading: Icon(Icons.emoji_people_outlined,
      color: Colors.grey), // Changed icon color
  title: Text('People Nearby'),
  onTap: () {
    // Navigate to People Nearby screen
  },
),
ListTile(
  leading: Icon(Icons.save,
      color: Colors.grey), // Changed icon color
  title: Text('Saved Messages'),
  onTap: () {
    // Navigate to Saved Messages screen
  },
),
ListTile(
  leading: Icon(Icons.settings,
      color: Colors.grey), // Changed icon color
```

```
          title: Text('Settings'),
          onTap: () {
            // Navigate to settings screen
            Navigator.push(
                context,
                MaterialPageRoute(
                    builder: (context) => ProfilePage(
                        userId: _auth.currentUser!.uid)));
          },
        ),
        Divider(
          color: Colors.grey[300], // Added a light grey line separator
          thickness: 1,
        ),
        ListTile(
          leading: Icon(Icons.person_add_outlined,
              color: Colors.grey), // Changed icon color
          title: Text('Invite Friends'),
          onTap: () {
            // Invite Friends
          },
        ),
        ListTile(
          leading: Icon(Icons.help_outline,
              color: Colors.grey), // Changed icon color
          title: Text('Telegram Features'),
          onTap: () {
            signOut();
          },
        ),
        ListTile(
          leading: Icon(Icons.logout,
              color: Colors.grey), // Changed icon color
          title: Text('Logout'),
          onTap: () {
            signOut();
          },
        ),
      ],
    );
  },
 ),
),
body: _buildUserList(),
```

```
    floatingActionButton: FloatingActionButton(
      onPressed: () {
        // Navigate to ContactsPage
        Navigator.push(
          context,
          MaterialPageRoute(
            builder: (context) => ContactsPage(), // Replace ContactsPage with the actual page
          ),
        );
      },
      child: Icon(Icons.edit, color: Colors.white,),
      backgroundColor: Color(0xFF3BACEB),
      shape: CircleBorder(),// Adjust the color to match the theme
    ),
    floatingActionButtonLocation: FloatingActionButtonLocation.endFloat,
  );
}


// build list of users except the current user
Widget _buildUserList() {
  return StreamBuilder<QuerySnapshot>(
    stream: FirebaseFirestore.instance.collection('users').snapshots(),
    builder: (context, snapshot) {
      if (snapshot.hasError) {
        return Text('Error: ${snapshot.error}');
      }
      if (snapshot.connectionState == ConnectionState.waiting) {
        return Center(child: CircularProgressIndicator());
      }

      return ListView(
        children: snapshot.data!.docs
            .map<Widget>((doc) => _buildUserListItem(doc))
            .toList(),
      );
    },
  );
}

// build individual user list items
Widget _buildUserListItem(DocumentSnapshot document) {
  Map<String, dynamic>? data = document.data() as Map<String, dynamic>?;
```

```
  if (_auth.currentUser != null &&
      _auth.currentUser!.phoneNumber != null &&
      data != null) {
    if (data['phone'] != null &&
        _auth.currentUser!.phoneNumber != data['phone']) {
      String displayName = data['username'] ?? data['phone'];

      return StreamBuilder<QuerySnapshot>(
        stream: FirebaseFirestore.instance
            .collection('chat_rooms')
            .doc(HomePage.constructChatRoomId(_auth.currentUser!.phoneNumber.toString(),
data['phone']))
            .collection('messages')
            .orderBy('timestamp', descending: true)
            .limit(1)
            .snapshots(),
        builder: (context, snapshot) {
          if (snapshot.hasData &&
              snapshot.data!.docs.isNotEmpty) {
            var lastMessage = snapshot.data!.docs.first;
            var timestamp = lastMessage['timestamp'];
            var messageTimestamp = DateTime.fromMillisecondsSinceEpoch(timestamp.seconds * 1000);
            var formattedTimestamp = DateFormat.jm().format(messageTimestamp);

            return Column(
              children: [
                ListTile(
                  title: Text(
                    displayName,
                    style: TextStyle(fontWeight: FontWeight.bold),
                  ), // Make the username bold
                  subtitle: Text(
                    lastMessage['message'],
                    maxLines: 1,
                    overflow: TextOverflow.ellipsis,
                    style: TextStyle(color: Colors.grey), // Set text color to grey
                  ),
                  trailing: Text(
                    formattedTimestamp,
                    style: TextStyle(color: Colors.grey), // Set text color to grey
                  ),
                  leading: CircleAvatar(
                    backgroundImage: data['profileImageUrl'] != null
                        ? NetworkImage(data['profileImageUrl'])
```

```
                : null,
              backgroundColor: Colors.orange, // Default background color
              radius: 40, // Increased the radius to make the profile icon bigger
              child: data['profileImageUrl'] == null
                ? Text(
                displayName[0].toUpperCase(), // Display the first letter of the username
                style: TextStyle(
                  color: Colors.white,
                  fontSize: 24, // Adjust the font size as needed
                  fontWeight: FontWeight.bold, // Make the font bold
                ),
              )
                : null, // Show the first letter only when no profile image is available
            ),
            onTap: () {
              Navigator.push(
                context,
                MaterialPageRoute(
                  builder: (context) => ChatPage(
                    receivedUserPhoneNumber: data['phone'] ?? '',
                  ),
                ),
              );
            },
          ),
          Divider(
            color: Colors.grey[300], // Added a light grey line separator
            thickness: 1,
          ),
        ],
      );
    } else {
      return SizedBox.shrink(); // Return an empty SizedBox if there are no messages
    }
  },
);
}
}
return SizedBox.shrink(); // Return an empty SizedBox if user data is invalid
}
}
```

P
Percy
+911234567890
New Group
Contacts
Call
People Nearby
Saved Messages
Settings
Invite Friends
Telegram Features
Logout

Contacts
New Group
New Secret Chat
New Channel
Sorting order: Ascending
A
Annabeth
Online
G
Gargi
Offline
P
Preeti
Offline
P
Priya
Offline

MAD & PWA Lab Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	28
Name	Preeti Khamkar
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

Experiment - 6

Name: Preeti Khamkar **Division:** D15A **Roll no. :** 28

Aim: To Connect Flutter UI with fireBase database

Theory:
Connecting a Flutter app to Firebase involves several steps, including setting up a Firebase project, configuring the Flutter app to use Firebase, and integrating Firebase SDKs into your Flutter code.

1. Firebase Setup: First, you need to create a Firebase project in the Firebase console (https://console.firebase.google.com/). This project will host your app's data and services.
2. Add Firebase to Flutter App: You'll need to add the Firebase SDK to your Flutter app. This involves adding Firebase configuration files to your project and updating your Flutter app's dependencies to include Firebase SDKs.
3. Initialize Firebase: In your Flutter app, you'll initialize Firebase by calling Firebase.initializeApp(). This should typically be done at the beginning of your app's lifecycle, such as in the main() function or in the initState() method of your main widget.
4. Use Firebase Services: Once Firebase is initialized, you can use various Firebase services in your Flutter app, such as Authentication, Firestore, Realtime Database, Cloud Storage, Cloud Messaging, and more.

Syntax:
```
WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
```

And import necessary packages. Add firebase_core and firebase_auth packages in pubsec.yaml.

Widget and Properties:
TextButton, TextField, AppBar, Container, Scaffold are the widgets used in this experiment.

Code and Output:
```
import 'dart:io';
```

```
import 'package:firebase_storage/firebase_storage.dart';
import 'package:flutter/material.dart';
import 'package:image_picker/image_picker.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:telegram/model/message.dart';
import 'package:telegram/services/chat/chat_service.dart';
import 'package:intl/intl.dart';
import 'package:telegram/mediaScreen.dart';

class ChatPage extends StatefulWidget {
 final String receivedUserPhoneNumber;
 const ChatPage({
   Key? key,
   required this.receivedUserPhoneNumber,
 }) : super(key: key);

 // Hardcoded set of usernames for receiver user's phone number
 static Map<String, String> hardcodedUsernames = {
   '+911234567890': 'Percy',
   '+919867506341': 'Preeti',
   '+911234567891': 'Annabeth',
   '+911987654321': 'Priya',
   '+911987654321': 'Gargi'
   // Add more mappings as needed
 };

 @override
 State<ChatPage> createState() => _ChatPageState();
}

class _ChatPageState extends State<ChatPage> {
 final TextEditingController _messageController = TextEditingController();
 final ChatService _chatService = ChatService();
 final FirebaseAuth _firebaseAuth = FirebaseAuth.instance;

 void sendMessage() async {
   if (_messageController.text.isNotEmpty) {
     await _chatService.sendMessage(
       widget.receivedUserPhoneNumber,
       _messageController.text,
     );
     _messageController.clear();
   }
```

```
}

Future<void> _selectImageFromGallery() async {
  final pickedFile = await ImagePicker().pickImage(source: ImageSource.gallery);
  if (pickedFile != null) {
    String imageUrl = await _uploadImage(File(pickedFile.path));
    await _chatService.sendMessage(
      widget.receivedUserPhoneNumber,
      '',
      imageUrl: imageUrl,
    );
  }
}

Future<String> _uploadImage(File imageFile) async {
  Reference storageReference =
FirebaseStorage.instance.ref().child('chat_images/${DateTime.now().millisecondsSinceEpoch}');
  UploadTask uploadTask = storageReference.putFile(imageFile);
  TaskSnapshot snapshot = await uploadTask;
  String downloadUrl = await snapshot.ref.getDownloadURL();
  return downloadUrl;
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      backgroundColor: const Color(0xFF588FBA),
      title: Row(
        children: [
          GestureDetector(
            onTap: () {
              // Navigate to Media Screen
              Navigator.push(
                context,
                MaterialPageRoute(builder: (context) => MediaScreen(userPhoneNumber:
widget.receivedUserPhoneNumber,)),
              );
            },
            child: Row(
              children: [
                // Add profile icon here, for example:
                CircleAvatar(
                  radius: 20,
```

```
                backgroundColor: Colors.orange, // Default background color
                child: Text(
                  ChatPage.hardcodedUsernames[widget.receivedUserPhoneNumber]?.substring(0, 1) ?? '', //
Display the first letter of the username
                  style: TextStyle(
                    color: Colors.white,
                    fontWeight: FontWeight.bold,
                  ),
                ),
              ),
              SizedBox(width: 13), // Add some spacing between icon and text
              // Add username text here
              Text(
                ChatPage.hardcodedUsernames[widget.receivedUserPhoneNumber] ?? 'Preeti',
                style: TextStyle(color: Colors.white),
              ),
            ],
          ),
        ),
        SizedBox(width: 110),
        Icon(Icons.call, color: Colors.white),
        SizedBox(width: 15),
        Icon(Icons.more_vert, color: Colors.white)
      ],
    ),
    iconTheme: IconThemeData(color: Colors.white), // Change back arrow color to white
  ),
  body: Container(
    decoration: BoxDecoration(
      image: DecorationImage(
        image: AssetImage("assets/images/chatBg.png"), // Replace "assets/background_image.jpg" with
your image asset path
        fit: BoxFit.cover,
      ),
    ),
    child: Column(
      children: [
        Expanded(
          child: _buildMessageList(),
        ),
        _buildMessageInput()
      ],
    ),
  ),
```

```
  );
}



Widget _buildMessageList() {
  return StreamBuilder(
    stream: _chatService.getMessages(widget.receivedUserPhoneNumber,
_firebaseAuth.currentUser!.phoneNumber.toString()),
    builder: (context, snapshots) {
      if (snapshots.hasError) {
        print("Error fetching messages: ${snapshots.error}");
        return Text('Error: ${snapshots.error}');
      }
      if (snapshots.connectionState == ConnectionState.waiting) {
        print("Waiting for messages...");
        return const Text('Loading...');
      }
      if (!snapshots.hasData) {
        print("No message data available.");
        return const Text('No messages yet.');
      }
      return ListView(
        children: snapshots.data!.docs
            .map((document) => _buildMessageItem(document))
            .toList(),
      );
    },
  );
}

Widget _buildMessageItem(DocumentSnapshot document) {
  Map<String, dynamic> data = document.data() as Map<String, dynamic>;

  var alignment = (data['senderPhoneNumber'] == _firebaseAuth.currentUser!.phoneNumber)
      ? Alignment.centerRight
      : Alignment.centerLeft;

  if (data['imageUrl'] != null && data['imageUrl'] != '') {
    return Container(
      alignment: alignment,
      child: Padding(
        padding: const EdgeInsets.all(8.0),
        child: Column(
```

```
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          FutureBuilder(
            future: _getUserDisplayName(data['senderPhoneNumber']),
            builder: (context, snapshot) {
              if (snapshot.connectionState == ConnectionState.waiting) {
                return CircularProgressIndicator();
              } else {
                return Text(snapshot.data.toString());
              }
            },
          ),
          Container(
            constraints: BoxConstraints(maxWidth: 250.0), // Adjust the max width as needed
            child: ClipRRect(
              borderRadius: BorderRadius.circular(8.0),
              child: Image.network(
                data['imageUrl'],
                width: 250.0, // Adjust the width as needed
                height: 200.0, // Adjust the height as needed
                fit: BoxFit.cover,
              ),
            ),
          ),
          Text(
            _formatTimestamp(data['timestamp']),
            style: TextStyle(color: Colors.grey),
          ), // Add this line to display the timestamp
        ],
      ),
    ),
  );
} else {
  return Container(
    alignment: alignment,
    child: Padding(
      padding: const EdgeInsets.all(8.0),
      child: Stack(
        children: [
          Container(
            padding: EdgeInsets.symmetric(vertical: 10, horizontal: 15),
            decoration: BoxDecoration(
              color: alignment == Alignment.centerRight ? Color(0xFFE1FFC6) : Colors.grey[100],
              borderRadius: BorderRadius.circular(20),
```

```
            ),
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Text(
                  data['message'],
                  style: TextStyle(fontSize: 16),
                ),
                SizedBox(height: 5),
                Text(
                  _formatTimestamp(data['timestamp']),
                  style: TextStyle(color: Colors.grey),
                ), // Add this line to display the timestamp
              ],
            ),
          ),
          Positioned(
            bottom: -10,
            right: alignment == Alignment.centerRight ? -8 : null,
            left: alignment == Alignment.centerLeft ? -8 : null,
            child: Icon(
              Icons.arrow_drop_up,
              color: alignment == Alignment.centerRight ? Color(0xFFE1FFC6) : Colors.grey[100],
              size: 30,
            ),
          ),
        ],
      ),
    ),
  );
 }
}

String _formatTimestamp(Timestamp timestamp) {
  DateTime dateTime = timestamp.toDate();
  String formattedTime = DateFormat.jm().format(dateTime); // Format the time as desired
  return formattedTime;
}

Widget _buildMessageInput() {
  return Container(
    color: Colors.white, // Set background color to white
    child: Row(
      children: [
```

```
        Expanded(
          child: TextField(
            controller: _messageController,
            decoration: const InputDecoration(
              hintText: 'Enter message',
            ),
            obscureText: false,
          ),
        ),
        IconButton(
          onPressed: sendMessage,
          icon: const Icon(
            Icons.send,
            size: 30,
            color: Colors.grey,
          ),
        ),
        IconButton(
          onPressed: _selectImageFromGallery,
          icon: const Icon(
            Icons.file_present_outlined,
            size: 30,
            color: Colors.grey,
          ),
        )
      ],
    ),
  );
}


Future<String> _getUserDisplayName(String phoneNumber) async {
  var snapshot = await FirebaseFirestore.instance.collection('users').doc(phoneNumber).get();
  var userData = snapshot.data();
  if (userData != null) {
    return userData['username'] ?? userData['phone'];
  }
  return phoneNumber;
}
}
```

Preeti
6:19 PM
hi
7:10 PM
hey
7:12 PM
hi
8:18 PM
+911234567890
8:18 PM
hello
9:52 PM
Enter message

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.	28
Name	Preeti Khamkar
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	15

Practical 7

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.

Difference between PWAs vs. Regular Web Apps:
A Progressive Web is different and better than a Regular Web app with features like:
1. **Native Experience**
Though a PWA runs on web technologies (HTML, CSS, JavaScript) like a Regular web app, it gives user experience like a native mobile application. It can use most native device features, including push notifications, without relying on the browser or any other entity. It offers a seamless and integrated user experience that it is quite tough for one to differentiate between a PWA and a Native application by considering its look and feel.

2. **Ease of Access**
Unlike other mobile apps, PWAs do not demand longer download time and make memory space available for installing the applications. The PWAs can be shared and installed by a link, which cuts down the number of steps to install and use. These applications can easily keep an app icon on the user's home screen, making the app easily accessible to the users and helps the brands remain in the users' minds, and improving the chances of interaction.

3. **Faster Services**
PWAs can cache the data and serve the user with text stylesheets, images, and other web content even before the page loads completely. This lowers the waiting time for the end-users and helps the brands improve the user engagement and retention rate, which eventually adds value to their business.

4. **Engaging Approach**

As already shared, the PWAs can employ push notifications and other native device features more efficiently. Their interaction does not depend on the browser user uses. This eventually improves the chances of notifying the user regarding your services, offers, and other options related to your brand and keeping them hooked to your brand. In simpler words, PWAs let you maintain the user engagement and retention rate.

5. **Updated Real-Time Data Access**
Another plus point of PWAs is that these apps get updated on their own. They do not demand the end-users to go to the App Store or other such platforms to download the update and wait until installed.
In this app type, the web app developers can push the live update from the server, which reaches the apps residing on the user's devices automatically. Therefore, it is easier for the mobile app developer to provide the best of the updated functionalities and services to the end-users without forcing them to update their app.

6. **Discoverable**
PWAs reside in web browsers. This implies higher chances of optimizing them as per the Search Engine Optimization (SEO) criteria and improving the Google rankings like that in websites and other web apps.

7. **Lower Development Cost**
Progressive web apps can be installed on the user device like a native device, but it does not demand submission on an App Store. This makes it far more cost-effective than native mobile applications while offering the same set of functionalities.

The main features are:
Progressive — They work for every user, regardless of the browser chosen because they are built at the base with progressive improvement principles.
Responsive — They adapt to the various screen sizes: desktop, mobile, tablet, or dimensions that can later become available.
App-like — They behave with the user as if they were native apps, in terms of interaction and navigation.
Updated — Information is always up-to-date thanks to the data update process offered by service workers.
Secure — Exposed over HTTPS protocol to prevent the connection from displaying information or altering the contents.
Searchable — They are identified as “applications” and are indexed by search engines.
Reactivable — Make it easy to reactivate the application thanks to capabilities such as web notifications.
Installable — They allow the user to “save” the apps that he considers most useful with the corresponding icon on the screen of his mobile terminal (home screen) without having to face all the steps and problems related to the use of the app store.
Linkable — Easily shared via URL without complex installations.

Offline — Once more it is about putting the user before everything, avoiding the usual error message in case of weak or no connection. The PWA are based on two particularities: first of all the 'skeleton' of the app, which recalls the page structure, even if its contents do not respond and its elements include the header, the page layout, as well as an illustration that signals that the page is loading.

Code and output:

Adding the manifest and serviceworker files:

Inserting the code:

//manifest.json

```
{
    "name":"KiddoShield",
  "short_name":"KS",
  "start_url":"index.html",
  "display":"standalone",
  "background_color":"#5900b3",
  "theme_color":"black",
  "scope": "./",
  "description":"This is a PWA tutorial.",
  }
```

//serviceworker.js

```
var staticCacheName = "Kiddoshield";

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

//index.html

The header includes:

```
<!-- Meta Tags required for
   Progressive Web App -->
<meta name=
"apple-mobile-web-app-status-bar"
       content="#aa7700">
<meta name="theme-color"
       content="black">

       <!-- Manifest File link -->
<link rel="manifest"
href="manifest.json">
```

The body includes:

```
<script>
       // Add event listener to execute code when page loads
       window.addEventListener('load', () => {
           // Call registerSW function when page loads
           registerSW();
       });

       // Register the Service Worker
       async function registerSW() {
           // Check if browser supports Service Worker
           if ('serviceWorker' in navigator) {
               try {
                   // Register the Service Worker named 'serviceworker.js'
                   await navigator.serviceworker.register('serviceworker.js');
               }
               catch (e) {
                   // Log error message if registration fails
```

```
            console.log('SW registration failed');
        }
    }
}
</script>
```

Installing live server:

Click on “Go live” in the bottom right corner:

Click on the three dots on the top right corner-> more tools-> developer tools:

Go to application:

Click on the three dots-> Apps-> Install this site as an app

Conclusion:

Thus, the “add to homescreen feature was successfully enabled by adding a manifest file to our web app.

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.	28
Name	Preeti Khamkar
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

Practical 8

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

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.

Service Worker Cycle

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

- Registration
- Installation
- Activation

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.

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.

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

Code:

//serviceworker.js

```
var staticCacheName = "pwa";

self.addEventListener("install", function (e) {
e.waitUntil(
  caches.open(staticCacheName).then(function (cache) {
  return cache.addAll(["/", "/index.html", "/about.html", "/blog.html", "/contact.html",
"/services.html","/img/slider-img4.jpg", "/css/main.css"]);
  })
);
});

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

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

JS in index.html:

```
<script>
    if ('serviceWorker' in navigator) {
    window.addEventListener('load', () => {
        navigator.serviceWorker.register('/serviceworker.js')
            .then(registration => {
                console.log('Service Worker registered with scope:', registration.scope);
            })
            .catch(error => {
                console.error('Service Worker registration failed:', error);
            });
    });
}
    </script>
```

Output:

Click on the three dots-> More tools-> Developer tools-> Application-> Service workers

The service worker is active and running.

Go to Cache storage:

Conclusion: The service worker was successfully registered and activated.

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.	28
Name	Preeti Khamkar
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

Practical 9

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 Workers:

- 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.

A sample is in the following for this case.

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.

Push Event

This is the event that handles push notifications that are received from the server. You can apply any method with received data. "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. You can use Application Tab from Chrome Developer Tools for testing push notification.

Code:

```
//serviceworker
const cacheName = 'ecommerce-pwa-v1';
var filesToCache = [
    '/',
    '/index.html',
    '/app.js',
    '/serviceworker.js'
    // Add more files and assets here as needed
];
self.addEventListener('install', 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) {
        try {
            var data = event.data.json();
            if (data && data.method === "pushMessage") {
                console.log("Push notification sent");
                self.registration.showNotification("Kiddoshield", { body: data.message });
            }
        } catch (error) {
            console.error("Error parsing push data:", error);
        }
    }
});
var preLoad = function(){
    return caches.open("offline").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(new Error("Response not found"));
                }
            })
            .catch(function (error) {
                reject(error);
            });
    });
};
var returnFromCache = function(request){
    return caches.open("offline").then(function(cache){
        return cache.match(request).then(function(matching){
            if(!matching || matching.status == 404){
                return cache.match("offline.html");
            }
            else{
                return matching;
            }
        });
    });
};
var addToCache = function(request) {
    return caches.open("offline").then(function(cache) {
        return fetch(request).then(function(response) {
            return cache.put(request, response.clone()).then(function() {
                return response;
            });
        });
    });
};
```

Output:

Fetch:

Sync:

Push:

Conclusion: Implemented service worker events like fetch, sync, and push for E-commerce PWA.

MAD & PWA Lab Journal

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	28
Name	Preeti Khamkar
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

Practical
10

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:

Pages link: p0203k.github.io
Github repo: https://github.com/P0203K/KS

Github Screenshot:

MAD & PWA Lab Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	28
Name	Preeti Khamkar
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	15

Practical 11

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

Original manifest.js:

```
{
   "name":"PWA Tutorial",
   "short_name":"PWA",
   "start_url":"index.html",
   "display":"standalone",
   "background_color":"#FFFFFF",
   "theme_color":"white",
   "scope": ".",
   "description":"This is a PWA tutorial."
}
```

Changes in manifest.js:

```
{
   "name":"PWA Tutorial",
   "short_name":"PWA",
   "start_url":"index.html",
   "display":"standalone",
   "background_color":"#FFFFFF",
   "theme_color":"white",
   "scope": ".",
   "description":"This is a PWA tutorial.",
   "icons":[
      {
      "src":"images/cropped.png",
      "sizes":"192x192",
      "type":"image/png",
      "purpose": "any"
      },
      {
      "src":"images/icon-512x512.png",
      "sizes":"512x512",
      "type":"image/png",
      "purpose": "maskable"
      }
   ]
}
```

Add the lighthouse tool to the activity bar by: Clicking on the three dots on the upper right corner -> more tools -> developer tools -> plus sign -> lighthoue

(Note: These steps are applicable for the Microsoft Edge browser)

Output:

Desktop-

Mobile-

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.	28
Name	Preeti Khamkar
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

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.	28
Name	Preeti Khamkar
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

Preeti Khamkar
DISA-28

1) Progressive Web App (PWA) -

- A progressive web app (PWA) is a set of mobile web application development techniques that entails building apps that feel and look like native ones
- Using a web stack, PWA combines a rich functionality and smooth user experience associated with native apps.
- Significance:
  - Cross platform compatibility (can run on any device)
  - Improved user experience (similar to native apps, with fast loading times, smooth animations and offline functionality)
  - offline functionality (allows users to access when offline)
  - Push notifications (It can send push notification to users, keeping them engaged and informed even when the app is not actively in use, similar to native apps)
  - Installation free (PWAs don't need to be downloaded and installed. These can be simply added ~~to~~ to the home screen)
- Key characteristics that differentiate PWA
  - Cross-platform compatibility.
  - No installation required
  - offline functionality.
  - Automatic updates
  - Low device storage usage

8 / 4

- PWAs offer a modern approach to web development by combining the best features of web and mobile applications, providing a cross-platform, user-friendly, and efficient solution for delivering engaging experiences to users.

Q.2) Responsive web design:

Ans: Responsive web design refers to a web design approach that enables your website content to 'respond' to and adapt to the screen and window sizes of its accessed devices.

Importance:

- Consistent user experience (regardless of screen size or resolution. crucial for maintaining usability and engagement across various platforms).
- Accessibility (Due to responsiveness, accessible to a broader range of users)
- Improved SEO (tend to ~~prefer~~ perform better in search engine rankings because search engines prioritize mobile-friendly websites)
- Cost efficiency (Create a single codebase that adapts to various orientations. This approach is cost-effective and easier to maintain.

Contd

Aspect	Responsive web design	Fluid web design	Adaptive web design
Layout adjustment	Base on viewport size	elements are sized proportionally	switches between predefined layouts
Control mechanism	Used media queries for layout changes	Relies on proportional sizing	Used predefined breakpoints
Flexibility	Continous adaptation	Continous adaptation	Control at breakpoints
Development complexity	Moderate	Moderate	Needs more effort
User experience	Consistent	Consistent	May vary
SEO impact	Positive	Positive	May vary
Cost effectiveness	Yes. due to single codebase	Yes due to proportional sizing	No, may require more resources initially

Q.3) Lifecycle of service workers

Ans. A serviceworker is a script that ~~uses~~ runs independently in the browser background.

1) Registeration-

- Involves the initial setup of the service worker
- Developers define a js file that acts as a service worker for web application
- It is ~~the~~ ~~very~~ registered using 'navigator service

worker.register()' method.

- During registration, the scope is specified which determines which URLs to control; the scope restricts the service worker's access to resources and defines the context in which it operates.
- Once registered, SW is installed in the background (by browser).

2) Installation -

- The browser then downloads the SW script.
- Browser fires 'install' event which triggers the execution of SW's 'install' even handler.
- This can be used to cache static assets and other resources need for offline functionality / performance optimization.
- Provides an opportunity to set up the initial state of SW and prepare it for activation.

3) Activation -

- Browser fires 'activate' event
- During activation developers can perform tasks like cleaning up outdated caches, managing versioning or initializing service-specific logic.
- Here, SW takes control of clients within its scope, intercepting network requested and handling events.
- Remains active until explicitly unregistered or replaced by newer version.

Q.4) Index DB in SW -

Ans - IndexDB is a NoSQL DB which is a key-value datab
that stores the data in the user's browser.
- It helps store large data, that is structured, locall
- Use within SW -
- (i) Caching resources: By storing cache resources locally the application can load and serve them more quickly, especially when the user is offline/ on slow network connection.
- Offline Support: SW can interrupt network requests and serve cached data from IndexDB when the network is unavailable, thus allowing offline access.
- Dynamic data storage: Provides a flexible and efficien way to store dynamic data generated by the web app SW can use IndexedDB to store user preferences, session data, application state, etc. that needs to persis across session / browser restarts.
- Background Synchronization: Conjunction of IndexedDB with background synchronisation techniques to queue a store data while offline and synchronize it later on so as to maintain data consistency
- Efficient data management: offers advanced features lik indexed, transactions and cursors which allow SW e efficiently query, update, retrieve and delete stored data.