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 **NITISH BHOSLE** of **D15A/D15B** semester **VI,** have successfully completed necessary experiments in the **MAD & PWA Lab** under my supervision in **VES Institute of Technology** during the academic year **2024-2025.**

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

Project Title: Friends Making app **Roll No. 04**

Year/Sem/Class **:** D15A/D15B **A.Y.:** 24-25

Faculty Incharge : Mrs. Kajal Joseph.

Lab Teachers **:** Mrs. Kajal Joseph.

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.

Project Title: Friends Making app **Roll No. 04**

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

Project Title: Friends Making app **Roll No. 04**

Index

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

MAD & PWA Lab Journal

Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	04
Name	Nitish Bhosle
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross-platform mobile application development using Flutter framework
Grade:	

Project Title: Friends Making app **Roll No. 04**

AIM: Installation and Configuration of Flutter Environment.

CODE:

Install Flutter

Setting the Environment Variable

Running Flutter command

```
C:\Users\HOME\spandan>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 "flut
                            diagnostic information. (Use "-vv" to force verbose logging in
-d, --device-id             Target device id or name (prefixes allowed).
    --version               Reports the version of this tool.
    --enable-analytics      Enable telemetry reporting each time a flutter or dart command
    --disable-analytics     Disable telemetry reporting each time a flutter or dart command
                            re-enabled.
    --suppress-analytics    Suppress analytics reporting for the current CLI invocation.

Available commands:
```

Running flutter doctor

```
C:\Users\HOME\spandan>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 3.27.3, on Microsoft Windows [Version 10.0.22631.4751], locale
[✓] Windows Version (Installed version of Windows is version 10 or higher)
[X] 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/to/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.
```

Downloading Android SDK

Downloaded command line tools of Android SDK

There are 2 issues according flutter doctor

```
C:\Users\HOME>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 3.27.3, on Microsoft Windows [Version 10.0.22631.4751], locale en-IN)
[✓] Windows Version (Installed version of Windows is version 10 or higher)
[!] Android toolchain - develop for Android devices (Android SDK version 35.0.1)
    ! Some Android licenses not accepted. To resolve this, run: flutter doctor --android-licenses
[✓] Chrome - develop for the web
[X] Visual Studio - develop Windows apps
    X Visual Studio not installed; this is necessary to develop Windows apps.
      Download at https://visualstudio.microsoft.com/downloads/.
      Please install the "Desktop development with C++" workload, including all of its default component
[✓] Android Studio (version 2024.2)
[✓] VS Code (version 1.96.2)
[✓] Connected device (3 available)
[✓] Network resources

! Doctor found issues in 2 categories.
```

Accepted the Android licenses

```
C:\Users\HOME>flutter doctor --android-licenses
Warning: This version only understands SDK XML versions up to 3 but an SDK XML file of version 4 was e
you use versions of Android Studio and the command-line tools that were released at different times.
                                                                                Warning: Errors during
Warning: Additionally, the fallback loader failed to parse the XML.ry...
Warning: Errors during XML parse:       ] 66% Fetch remote repository...
Warning: Additionally, the fallback loader failed to parse the XML.
[=======================================] 100% Computing updates...
6 of 7 SDK package licenses not accepted.
Review licenses that have not been accepted (y/N)? y

1/6: License android-googletv-license:
---------------------------------------
Terms and Conditions

This is the Google TV Add-on for the Android Software Development Kit License Agreement.

1. Introduction

1.1 The Google TV Add-on for the Android Software Development Kit (referred to in this License Agreeme
d specifically including the Android system files, packaged APIs, and Google APIs add-ons) is licensed
```

Installed Visual Studio

Successful Running of flutter doctor command

```
C:\Users\HOME\spandan>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 3.27.3, on Microsoft Windows [Version 10.0.22631.4751], locale en-IN)
[✓] Windows Version (Installed version of Windows is version 10 or higher)
[✓] Android toolchain - develop for Android devices (Android SDK version 35.0.1)
[✓] Chrome - develop for the web
[✓] Visual Studio - develop Windows apps (Visual Studio Community 2022 17.12.4)
[✓] Android Studio (version 2024.2)
[✓] VS Code (version 1.96.2)
[✓] Connected device (3 available)
[✓] Network resources

• No issues found!
```

Code

```
import 'package:flutter/material.dart';


void main() {
  runApp(const MyApp());
}


class MyApp extends StatelessWidget {
  const MyApp({super.key});


  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: Scaffold(
        appBar: AppBar(title: const Text('Hello App')),
        body: const Center(
          child: Text(
            'Hello, Nitish!',
            style: TextStyle(fontSize: 24, fontWeight:
FontWeight.bold),
          ),
        ),
      ),
    );
  }
}
```

MAD & PWA Lab Journal

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

AIM- To design Flutter UI by including common widgets

THEORY-

Flutter is an open-source UI framework by Google that allows developers to build natively compiled applications for mobile, web, and desktop using a single codebase. It uses the Dart programming language and follows a widget- based architecture. In Flutter, everything is a widget, from layout components to UI elements.

Some of the common widgets

- **Container** – A flexible box that can hold other widgets and be styled with padding, margins, borders, and background colors. It is often used for layout structuring.
- **Text** – Used to display text with various styles, such as font size, color, weight, and alignment.
- **Image** – Loads and displays images from different sources like assets, networks, and memory.
- **Row & Column** – These layout widgets help arrange child widgets horizontally (Row) or vertically (Column). They are essential for structuring UI components.
- **Scaffold** – Provides a basic page structure, including an AppBar, body, floating action button, and drawer. It is the foundation of most Flutter screens.
- **AppBar** – A top navigation bar that usually contains a title, icons, and action buttons.
- **ListView** – A scrollable list widget that efficiently displays multiple items, often used for dynamic content like messages or product lists.
- **Text Field** – Allows users to input text, commonly used in forms and search fields.

SYNTAX-

AppBar creates a top navigator bar with title and icons.

```
AppBar (
 title:
 Text("Title"),
 leading:
 IconButton( icon:
 Icon(Icons.menu),
 onPressed: () {},
 ),
 bottom: TabBar(),
)
```

Scaffold creates the basic layout structure of the app.

```
Scaffold (
 appBar:
 AppBar(), body:
 Widget(),
 drawer:
 Drawer(),
 bottomNavigationBar: BottomNavigationBar(),
)
```

TabBar creates tabs for switching views

```
TabBar(
 tabs: [
  Tab(text: "Tab 1"),
  Tab(text: "Tab 2"),
 ],
)
```

Drawer a side menu that slides in from left

```
Drawer(
 child: ListView(
  children: [
  DrawerHeader(
    decoration: BoxDecoration(color:
    Colors.blue), child: Text("Header"),
   ),
   ListTile(
    leading: Icon(Icons.star),
```

```
    title: Text("Menu
    Item"), onTap: ()
    {},
   ),
  ],
 ),
)
```

Listview creates a scrollable list dynamically

```
ListView.builder(
 itemCount: items.length,
 itemBuilder: (context,
 index) { return ListTile(
   title: Text(items[index]),

  );
 },
)
```

Widget Properties

Scaffold

key → Used to manage state
appBar → Adds a top navigation bar
body → The main content of the screen
drawer → A slide-out menu on the left
bottomNavigationBar → A navigation bar at the bottom

AppBar

title → Sets a title or an icon
leading → Adds an icon or button on the left
backgroundColor → Changes the background color
elevation → Controls the shadow effect
centerTitle → Aligns the title in the center
bottom → Adds a TabBar

Drawer

child → Contains a list of menu items
ListView → Displays menu options in a scrollable list

ListView

padding → Controls spacing around the list children
→ Contains multiple widgets inside the list

ListTile
leading → Adds an icon on the left
title → The main text of the item
trailing → Adds an icon on the right
onTap → Defines what happens when tapped

CODE

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:tinder_clone/component/profile_feature.dart';
import 'package:tinder_clone/component/profile_setting.dart';
import 'package:tinder_clone/component/profile_overall.dart';

class ProfileScreen extends StatefulWidget {
  const ProfileScreen({super.key});

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

class _ProfileScreenState extends State<ProfileScreen> {
  final FirebaseAuth _auth = FirebaseAuth.instance;
  final FirebaseFirestore _firestore = FirebaseFirestore.instance;
  String name = "User";
  String profilePic = "";
  int age = 20;
  String bio = "Add your bio!";

  @override
  void initState() {
    super.initState();
    _getUserData
```

```
  }

  void _getUserData() async {
    User? user = _auth.currentUser;
    if (user != null) {
      DocumentSnapshot userData =
          await _firestore.collection('users').doc(user.uid).get();
      setState(() {
        name = userData['name'] ?? "User";
        profilePic = userData['profilePic'] ?? "";
        age = userData['age'] ?? 20;
        bio = userData['bio'] ?? "Add your bio!";
      });
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text("Profile"),
        actions: [
          IconButton(
            icon: Icon(Icons.settings),
            onPressed: () {
              Navigator.push(
                context,
                MaterialPageRoute(builder: (context) =>
ProfileSetting()),
              );
            },
          ),
        ],
      ),
      body: Column(
        crossAxisAlignment: CrossAxisAlignment.center,
        children: [
          ProfileOverall(
```

```
            imageUrl: profilePic,
            userName: name,
            userAge: age,
            completionPercentage: 90,
          ),
          SizedBox(height: 20),
          Text(
            bio,
            style: TextStyle(fontSize: 16, color: Colors.grey),
          ),
          SizedBox(height: 20),
          ProfileFeature(),
        ],
      ),
  );
}
```

OUTPUT

tinder_clone
tinder
Nitish, 20
Super Like
0
Boost
want more?
Membership
Upgrade
Tinder Gold Subscription
Upgrade
See who likes you
Top Picks
Unlimited Super Likes
Home
Chat
Profile

```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:tinder_clone/component/swipecard.dart';
import 'package:tinder_clone/screen/login_screen.dart';

class HomeScreen extends StatelessWidget {
  final FirebaseAuth _auth = FirebaseAuth.instance;

  void _logout(BuildContext context) async {
    await _auth.signOut();
    Navigator.pushReplacement(
      context,
      MaterialPageRoute(builder: (context) => LoginScreen()),
    );
  }

  @override
  Widget build(BuildContext context) {
    User? user = _auth.currentUser;

    return Scaffold(
      appBar: AppBar(
        title: Row(
          children: [
            Icon(Icons.local_fire_department, color: Colors.red),
            SizedBox(width: 8),
            Text(
              "tinder",
              style: TextStyle(color: Colors.red, fontWeight:
FontWeight.bold),
            ),
          ],
        ),
        actions: [
          if (user != null)
            CircleAvatar(
              backgroundImage: NetworkImage(user.photoURL ?? ""),
            ),
          IconButton(
            icon: Icon(Icons.logout, color: Colors.pink),
            onPressed: () => _logout(context),
          ),
        ],
```

```
      ),
      body: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Expanded(
            child: TinderSwipeCard(),
          ),
        ],
      ),
    );
  }
}
```

MAD & PWA Lab Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	04
Name	Nitish Bhosle
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

AIM-To include images, fonts in flutter app.

THEORY-

Images are an essential part of UI design, and Flutter supports adding both local and network images.

A) Local images can be stored in the project directory and loaded into the

app. Steps to add Local images

- Create an assets folder in the root directory.
- Store images inside the assets folder.
- Declare assets in pubspec.yaml under the flutter section:

```
flutter:
  assets:
    - assets/image1.png
    - assets/images/image2.jpg
```

B) Network Images

Flutter allows displaying images from the internet using Image.network():

Image.network('https://example.com/image.jpg')

Font Awesome provides a vast collection of scalable vector icons that behave like fonts. These icons can be used in Flutter via the font_awesome_flutter package, which integrates Font Awesome's font-based icons seamlessly into the app.

SYNTAX

1) Create an assets folder for Local images. Declare assets in pubspec.yaml file. flutter:

assets:

- assets/image1.png
- assets/images/image2.jpg

Image.asset('assets/image1.png')

2) If using network Images

Image.network('https://example.com/image.jpg')

3) Install fontawesome package in flutter Add this dependency in pubspec.yaml file dependencies:

font_awesome_flutter: ^10.7.0

Run flutter pub get

FaIcon(FontAwesomeIcons.heart, size: 50, color: Colors.red)

Widget properties

1) image

- width: Sets image width.
- height: Sets image height.
- fit: Controls how image fits (e.g., BoxFit.cover, BoxFit.fill).
- alignment: Aligns the image inside the container.
- color: Applies a color filter.
- opacity: Controls image transparency.
- loadingBuilder: Handles loading states.
- errorBuilder: Handles image load errors.

Example

```
Image.network(
  'https://example.com/image.jpg',
  width: 100,
  height: 100,
  fit: BoxFit.contain,
  loadingBuilder: (context, child, progress) {
    return progress == null ? child : CircularProgressIndicator();
  },
  errorBuilder: (context, error, stackTrace) {
    return Icon(Icons.error);
  },
)
```

2) font

- size: Adjusts icon size.
- color: Sets icon color.
- semanticLabel: Adds an accessibility label for screen readers.

Example:

```
FaIcon(
  FontAwesomeIcons.heart,
  size: 50,         // Sets icon
  size
  color: Colors.red, // Sets icon color
  semanticLabel: 'Heart Icon', // Provides accessibility label
)
```

CODE

```
// Create a list of UserProfile instances
List<UserProfile> userProfiles = [
  UserProfile(
    name: "Mark Lee",
    age: 25,
    description: "You can mark me in your heart.",
    imageUrls: [
      "assets/images/mark-1.jpg",
      "assets/images/mark-2.jpg",
    ],
  ),
  UserProfile(
      name: "real_pcy",
      age: 28,
      description: "nan",
      imageUrls: [
        "assets/images/cy1.jpg",
        "assets/images/cy2.jpg",
        "assets/images/cy3.jpg",
        "assets/images/cy4.jpg"
      ]),
  UserProfile(
      name: "beakhyunee_",
      age: 28,
      description: "nan",
      imageUrls: [
        "assets/images/baek4.jpg",
        "assets/images/baek2.jpg",
        "assets/images/baek1.jpg"
      ]),
  UserProfile(
      name: "Anton Lee",
      age: 20,
      description: "make the impossible possible",
      imageUrls: [
        "assets/images/anton1.jpg",
      ]),
  UserProfile(
      name: "Park Wonbin",
      age: 22,
      description: "Chrome Heart lover",
      imageUrls: [
        "assets/images/wonbin1.jpg",
```

```
      ]),
  UserProfile(
      name: "jung Sungchan",
      age: 28,
      description: "bog bog",
      imageUrls: [
        "assets/images/jss1.jpg",
      ]),
  UserProfile(
      name: "mille",
      age: 28,
      description: "Action = Rection ",
      imageUrls: [
        "assets/images/hj1.jpg",
      ]),
  UserProfile(
    name: "JUJU",
    age: 28,
    description: "nan",
    imageUrls: [
    "assets/images/juyeon1.jpg",
  ]),
];
```

OUTPUT

tinder_clone
tinder
Mark Lee 25
You can mark me in your heart.
Home
Chat
Profile

MAD & PWA Lab Journal

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

AIM-To create an interactive form using form widget

THEORY-

Forms are essential components of web and mobile applications, allowing users to input and submit data. An interactive form enhances user experience by providing real-time validation, user-friendly input fields, and seamless data handling.

A **Form Widget** is a structured way to manage user input, validate data, and handle submissions efficiently. It provides an interactive interface for users to enter and modify information.

Key Features of Interactive Forms

- User Input Fields: Text fields, dropdowns, checkboxes, radio buttons, and other input elements.
- Real-time Validation: Ensures correct data format before submission.
- Error Handling: Displays messages for invalid inputs.
- Data Submission: Sends user input to a backend or local storage for further processing.
- Dynamic Updates: Auto-fills or adjusts form fields based on user selections.

Components of Form Widget

- Form Container: Wraps all input fields.
- Input Fields: Text fields, number fields, password inputs, email inputs, etc.
- Buttons: Submit and reset buttons to process or clear input.
- Validation Mechanisms: Ensures valid input before submission.

SYNTAX

```
Form(

 key: formKey, // Unique key to
 manage form state child: Column(
  children:
   [
   TextFormF
   ield(
    decoration: InputDecoration(labelText:
    "Enter your name"), validator: (value) {
     if (value == null ||
      value.isEmpty) { return
      "This field cannot be
      empty";
     }

     return null;

    },

   ),

   SizedBox(heig
   ht: 10),
   ElevatedButto
   n( onPressed:
   () {
     if (formKey.currentState!.validate()) {

      // Perform form submission action

     }

    },

    child: Text("Submit"),

   ),

  ],

 ),

)
```

Widget Properties

1) key
   - Used to uniquely identify the Form widget.
   - Typically assigned a GlobalKey<FormState> to manage validation and submissions.

   Example, final _formKey = GlobalKey<FormState>();

```
Form(
 key:
 _formKey,
 child:
 Column(
  children: [ /* Form fields go here */ ],
 ),
);
```

2) child
   - Defines the content inside the Form, usually containing form fields like TextFormField, DropdownButtonFormField, etc.

Example:

```
Form(
 child:
  Column(
  children: [
  TextFormFiel
  d(),
   ElevatedButton(onPressed: () {}, child: Text("Submit")),
  ],
 ),
);
```

3) onchanged
   - A callback function that gets triggered when any field inside the form changes.
   - Can be used to update state based on form input.

Example:

```
Form(
 onChanged:
 () {
  print("Form data changed!");
 },
 child: TextFormField(),
);
```

CODE

```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:google_sign_in/google_sign_in.dart';
import 'package:tinder_clone/screen/homescreen.dart';

class LoginWithScreen extends StatefulWidget {
  @override
_LoginWithScreenState createState() => _LoginWithScreenState();
}

class _LoginWithScreenState extends State<LoginWithScreen> {
  final FirebaseAuth _auth = FirebaseAuth.instance;

  Future<UserCredential?> _signInWithGoogle() async {
    try {
      final GoogleSignInAccount? googleUser = await
GoogleSignIn().signIn();
      final GoogleSignInAuthentication googleAuth =
          await googleUser!.authentication;

      final AuthCredential credential =
GoogleAuthProvider.credential(
accessToken: googleAuth.accessToken, idToken:
        googleAuth.idToken,
      );

UserCredential userCredential =
          await _auth.signInWithCredential(credential);
      return userCredential
```

```
} catch (e) {
print("Google Sign-In Error: $e"); return
      null;
    }
  }

void _handleSignIn() async {
UserCredential? userCredential = await _signInWithGoogle(); if
    (userCredential != null) {
      Navigator.pushReplacement(
        context,
MaterialPageRoute(builder: (context) => HomeScreen()),
      );
    }
  }

@override
  Widget build(BuildContext context) {
    return Scaffold(
body: Center(
child: ElevatedButton(
onPressed: _handleSignIn,
child: Text("Sign in with Google"),
        ),
      ),
    );
  }
}
```

```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:google_sign_in/google_sign_in.dart';
import 'package:tinder_clone/screen/homescreen.dart';

class LoginScreen extends StatelessWidget {
final FirebaseAuth _auth = FirebaseAuth.instance;
final FirebaseFirestore _firestore = FirebaseFirestore.instance;

  LoginScreen({super.key}); // Ensure a proper constructor

  Future<void> _signInWithGoogle(BuildContext context) async {
    try {
```

```
      final GoogleSignInAccount? googleUser = await
GoogleSignIn().signIn();
      if (googleUser == null) return; // User canceled login

      final GoogleSignInAuthentication googleAuth =
          await googleUser.authentication;

      final AuthCredential credential =
GoogleAuthProvider.credential(
        accessToken: googleAuth.accessToken,
        idToken: googleAuth.idToken,
      );

      UserCredential userCredential =
          await _auth.signInWithCredential(credential);
      User? user = userCredential.user;

      if (user != null) {
        await _firestore.collection('users').doc(user.uid).set({
          'uid': user.uid,
          'name': user.displayName ?? 'No Name',
          'email': user.email ?? 'No Email',
          'profilePic': user.photoURL ?? '',
          'age': 20, // Default age, you can update it later
          'bio': 'Add your bio here!',
          'createdAt': FieldValue.serverTimestamp(),
        }, SetOptions(merge: true));
      }

      Navigator.pushReplacement(
        context,
        MaterialPageRoute(builder: (context) => HomeScreen()),
      );
    } catch (e) {
      print("Google Sign-In Error: $e");
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: ElevatedButton(
          onPressed: () => _signInWithGoogle(context),
```

```
child: Text("Sign in with Google"),
        ),
      ),
    );
  }
}
```

OUTPUT

MAD & PWA Lab Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	04
Name	Nitish Bhosle
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

AIM-To apply navigation, routing and gestures in Flutter App

THEORY-

Navigation in Flutter allows users to move between different screens (or pages) in the app. Flutter uses the Navigator widget to handle navigation between routes (screens).

Types of Navigation

- Push Navigation (Forward Navigation) → Moves to a new screen.
- Pop Navigation (Backward Navigation) → Moves back to the previous screen.
- PushReplacement → Replaces the current screen with a new one.
- PushAndRemoveUntil → Moves to a new screen and removes previous screens from the stack.

Routing in Flutter manages different screens in the app. It helps organize and structure navigation efficiently.

Types of Routing

1. Direct Route Navigation (MaterialPageRoute)-Used for simple page-to-page navigation.
2. Named Routes (Predefined Routes in main.dart)-Defined in the MaterialApp widget and used throughout the app.

Flutter uses the GestureDetector widget to detect user interactions like taps, swipes, pinches, and long presses. This is essential for making an app interactive.

Common Gestures & Their Uses:

- Tap → Detects simple taps on a widget.
- Double Tap → Recognizes double-clicking.
- Long Press → Triggers an action when the user presses and holds.
- Swipe (Drag) → Detects horizontal or vertical dragging.
- Pinch (Zoom In/Out) → Detects two-finger pinch for zooming.

SYNTAX

Navigator

```
Navigator.push
( context,
 MaterialPageRoute(builder: (context) => SecondPage()),
);
Navigator.pushReplacement(
 context,
 MaterialPageRoute(builder: (context) => NewPage()),
);
```

Routing

```
void main() {
 runApp(MaterialApp
 ( initialRoute:
 '/', routes: {
   '/': (context) => HomePage(),
   '/profile': (context) =>
   ProfilePage(),
  },
 ));
}
```

Gestures

```
GestureDetector(
onTap: () {
  print("Widget Tapped!");
 },
```

```
 child:
  Container(
  width: 100,
  height: 100,
  color: Colors.blue,
 ),
);
```

Widget Properties

Navigator

context → The current build context for navigation. MaterialPageRoute → Creates a transition animation between pages. builder → Defines the widget to navigate to.

Navigator.push() → Pushes a new screen on top of the stack.

Navigator.pop() → Removes the top screen and goes back.

Navigator.pushReplacement() → Replaces the current screen with a new one.

Routing

initialRoute → Sets the first screen when the app starts.

routes → Defines a map of route names and corresponding widgets.

Navigator.pushNamed() → Navigates using a predefined route.

Navigator.pop() → Closes the current screen and returns to the previous one.

Gestures

onDoubleTap → Detects a double tap.

onLongPress → Detects when the user presses and holds. onHorizontalDragStart → Detects when a horizontal drag begins. onHorizontalDragUpdate → Detects movement during a horizontal drag. onHorizontalDragEnd → Detects when a horizontal drag stops.

CODE

```
import 'dart:async';


import 'package:flutter/material.dart';
import 'package:flutter_screenutil/flutter_screenutil.dart';


import 'package:tinder_clone/screen/login_screen.dart';



class SplashScreen extends StatefulWidget {


  @override
  _SplashScreenState createState() => _SplashScreenState();
}
```

```
class _SplashScreenState extends State<SplashScreen> {

  @override

  void initState() {

    super.initState();

    Timer(new Duration(seconds: 2),(){

      Navigator.pop(context);

      Navigator.push(context, MaterialPageRoute(builder: (context)
=> LoginScreen())) ;

    });

  }



  @override

  Widget build(BuildContext context) {

    ScreenUtil.init(context) ;



    return Scaffold(

      backgroundColor: Colors.white,

      body: new Center(

        child: new Image(

          width: ScreenUtil().setWidth(100.0),

          height: ScreenUtil().setHeight(100.0),

          image: new AssetImage('assets/images/tinder.png'),

        ),

      ),

    );

  }

}
```

OUTPUT

tinder_clone
tinder
Swipe Right To Explore
Create Account
Login

tinder_clone
tinder
Mark Lee 25
You can mark me in your heart.
Home
Chat
Profile

MAD & PWA Lab Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	04
Name	Nitish Bhosle
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:	

AIM- To connect Flutter UI with firebase.

THEORY-

Firebase helps developers to manage their mobile app easily. It is a service provided by Google. Firebase has various functionalities available to help developers manage and grow their mobile apps.

Steps to Add firebase to our Flutter app using Firebase CLI

1. Install the Firebase CLI and log in (run firebase login)
2. From any directory, run this command:
    - dart pub global activate flutterfire_cli
3. Then, at the root of your Flutter project directory, run this command:
    - flutterfire configure --project=questitnextjs
4. This automatically registers your per-platform apps with Firebase and adds a lib/firebase_options.dart configuration file to your Flutter project.
5. To initialise Firebase, call Firebase.initializeApp from the firebase_core package with the configuration from your new firebase_options.dart file:

import 'package:firebase_core/firebase_core.dart';

import 'firebase_options. dart';

await Firebase.initializeApp(

options: DefaultFirebaseOptions.currentPlatform,

);

6. Add the dependencies in the

pubspec.yaml file Firebase_core : ^version

Firebase_auth : ^version

SYNTAX

```
1. Firebase Core


Purpose Serves as the foundation for all Firebase services in
your Flutter application.


Implementation:

-    Initialized in `main.dart` using
`Firebase.initializeApp()` to set up the Firebase SDK before
app startup.

-    Ensures Firebase services are properly configured and
ready to use throughout the app.


Reference Code

dart

void main() async {

  WidgetsFlutterBinding.ensureInitialized()

  ; await Firebase.initializeApp();

  runApp(MyApp());

}

2. Firebase Authentication


Purpose: Manages user authentication, allowing users to sign
```

in/sign up and maintaining authentication state.

Implementation:

- Provides multiple authentication methods via `AuthService` including:
  - Email/password authentication
  - Google Sign-In integration
- Tracks authentication state changes with `authStateChanges()` stream
- Handles user sign-out across multiple auth providers

Reference Code:

dart

```
// Email/password auth

Future<User?> signInWithEmail(String email, String password)
async {

  try {

    final userCredential = await
_auth.signInWithEmailAndPassword

      ( email: email,

      password: password,

    );

    return userCredential.user;

  } catch (e) {

    print(e);

    return null;

  }

}


// Google Sign-In

Future<User?> signInWithGoogle() async {

  try {

    final googleUser = await _googleSignIn.signIn();

    if (googleUser == null) return null;

    final googleAuth = await googleUser.authentication;
```

```
    final credential = GoogleAuthProvider.credential(
      accessToken: googleAuth.accessToken,
      idToken: googleAuth.idToken,
    );
    final userCredential = await
_auth.signInWithCredential(credential);
    return userCredential.user;
  } catch (e) {
    print(e);
    return null;
  }
}
```

3. Cloud Firestore

Purpose: Provides a NoSQL database for storing and synchronizing application data in real-time.

Implementation:

- User Profiles: Stores user information in the `users` collection
- Matching System: Records user swipes in the `swipes` collection
- Conversations: Manages user matches and messages in the `conversations` collection
- Real-time Data: Uses streams to provide live updates for messages and matches

Data Structure:

- users: Stores user profiles with personal information, preferences, and interests
- swipes: Records user swiping behavior (like/dislike) with timestamps
- conversations: Contains messaging between matched users

```
 - Each conversation has a subcollection of `messages`


Reference Code:

dart

// Creating/updating user profile

Future<void> setUserProfile(UserModel user)

{

  return
_db.collection('users').doc(user.uid).set(user.toMap());

}



// Recording swipe action

Future<void> recordSwipe(String swiperUID, String swipedUID,
String action) {

  return
_db.collection('swipes').doc('${swiperUID}_$swipedUID').set({

    'swiperUID': swiperUID,

    'swipedUID': swipedUID,

    'action': action,

    'timestamp': FieldValue.serverTimestamp(),

  });

}



// Creating a match conversation when both users like each
other

Future<void> checkForMatch(String user1UID, String user2UID)
async {

  final doc = await
_db.collection('swipes').doc('${user2UID}_$user1UID').get()

  ; if (doc.exists && doc['action'] == 'like') {

    final conversationID = user1UID.compareTo(user2UID) < 0

        ? '${user1UID}_$user2UID'

        : '${user2UID}_$user1UID';


    await
```

```
_db.collection('conversations').doc(conversationID).set({
```

```
      'users': [user1UID, user2UID],
      'createdAt':
      FieldValue.serverTimestamp(),
    });
  }
}
```

4. Real-time Messaging

Purpose: Enables instant messaging between matched users.

Implementation:

- Uses Firestore for real-time message delivery
- Stores messages in subcollections within conversation documents
- Efficiently retrieves and displays messages with StreamBuilder
- Handles message sending with server timestamps

Reference Code:

```dart
// Sending a message
Future<void> sendMessage(String conversationID, String senderUID, String message) {
  return _db.collection('conversations').doc(conversationID).collection('messages').add({
    'sender': senderUID,
    'message': message,
    'timestamp': FieldValue.serverTimestamp(),
  });
}
```

```
// Retrieving messages in real-time
```

```
Stream<List<Map<String, dynamic>>> getMessages(String
conversationID) {

  return _db

      .collection('conversations')

      .doc(conversationID)

      .collection('messages')

      .orderBy('timestamp')

      .snapshots()

      .map((snapshot) => snapshot.docs.map((doc) =>
doc.data()).toList());

}
`
```

5. State Management with Firebase

Purpose: Integrates Firebase data with the app's state management system.

Implementation:

- Uses Provider pattern to make Firebase services accessible throughout the app
- Manages authenticated user state with UserProvider
- Efficiently updates UI based on real-time Firebase data changes
- Implements proper error handling for Firebase operations

Reference Code:

```dart
// App wrapper that handles authentication state

return StreamBuilder<User?>(

  stream: FirebaseAuth.instance.authStateChanges(),

  builder: (context, snapshot) {

    if (snapshot.connectionState == ConnectionState.waiting) {

      return Scaffold(body: Center(child:
CircularProgressIndicator()));
```

```
      }
      final user = snapshot.data;
      if (user == null) {
        return LoginScreen();
      } else {
        // User is authenticated, fetch and provide user data
        // ...
      }
    }
);
```

Summary

Your application is a dating/matching app that effectively uses Firebase for:

1. User Management: Authentication with multiple sign-in methods and user profile storage
2. Matching Algorithm: Records and processes user preferences and swipe actions
3. Real-time Communication: Facilitates instant messaging between matched users
4. State Synchronization: Keeps UI in sync with backend data changes

OUTPUT

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.	04
Name	Nitish Bhosle
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	

Aim-To write meta data of your Ecommerce PWA in a Web app manifest file to enable add to home screen 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.

Pros and cons of the Progressive Web App

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.

Weaknesses refer to:
IOS support from version 11.3 onwards;
Greater use of the device battery;
Not all devices support the full range of PWA features (same speech for iOS and Android operating systems);
It is not possible to establish a strong re-engagement for iOS users (URL scheme, standard web notifications);
Support for offline execution is however limited;
Lack of presence on the stores (there is no possibility to acquire traffic from that channel);
There is no “body” of control (like the stores) and an approval process;
Limited access to some hardware components of the devices;
Little flexibility regarding “special” content for users (eg loyalty programs, loyalty, etc.).

Code: -

//Manifest.json:

```
{
  “short_name”: “Movie Rec”,
  “name”: “Movie Recommendation App”,
  “icons”: [
    {
      “src”: “favicon.ico”,
      “sizes”: “64x64 32x32 24x24 16x16”,
      “type”: “image/x-icon”
    },
    {
```

```
      "src": "logo192.png",
      "type": "image/png",
      "sizes": "192x192",
      "purpose": "any maskable"
    },
    {
      "src": "logo512.png",
      "type": "image/png",
      "sizes": "512x512"
    }
  ],
  "start_url": ".",
  "display": "standalone",
  "theme_color": "#000000",
  "background_color": "#ffffff",
  "description": "Find your next favorite movie",
  "orientation": "portrait-primary",
  "categories": ["entertainment", "movies"]
}
```

//Serviceworker.js

```
// Import workbox from CDN (you can also use the
workbox-webpack-plugin)

importScripts('https://storage.googleapis.com/workbox-cdn/rele
ases/6.4.1/workbox-sw.js');

// Precache and route setup

workbox.routing.registerRoute(

  ({request}) => request.destination === 'image',

  new workbox.strategies.CacheFirst({

    cacheName: 'images',

    plugins: [

      new workbox.expiration.ExpirationPlugin({

        maxEntries: 60,

        maxAgeSeconds: 30 * 24 * 60 * 60, // 30 Days

      }),

    ],
```

```
  })
);

// Cache CSS and JavaScript Files
workbox.routing.registerRoute(
  ({request}) => request.destination === 'script' ||
request.destination === 'style',
  new workbox.strategies.StaleWhileRevalidate({
    cacheName: 'static-resources',
  })
);
```

//index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width,
initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Movie recommendations app - find your next
favorite film"
    />
    <link rel="apple-touch-icon"
href="%PUBLIC_URL%/logo192.png" />
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <title>Movie Recommendations</title>
  </head>
```

```
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
  </body>
</html>
```

Output

//Open folder in VS Code and run the website

```
HOME@LAPTOP-9JIMM8I3 MINGW64 /e/New folder/Movie-Recommendation-App (main)
$ npm start

Compiled successfully!

You can now view movie-recommendation-app in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.162.1:3000
```

//open developers tools -> Applications

// Install the app by following this route Click on three dots on top right corner of the browser from app option ->Cast,share,save->Install the app

// This is the 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.	04
Name	Nitish Bhosle
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

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.

What can't we do with Service Workers?

You can't access the **Window**

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

- You can't work it on **80 Port**

Service Worker just can work on HTTPS protocol. But you can work on localhost during development.

Service Worker Cycle

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

- Registration
- Installation
- Activation

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. Let's look at an example:

main.js

```
if ('serviceWorker' in navigator) { navigator.serviceWorker.register('/service-worker.js')
.then(function(registration) {
console.log('Registration successful, scope is:', registration.scope);
})
.catch(function(error) {
console.log('Service worker registration failed, error:', error);
});
}
```

This code starts by checking for browser support by examining **navigator.serviceWorker**. The service worker is then registered with navigator.serviceWorker.register, which returns a promise that resolves when the service worker has been successfully registered. The scope of the service worker is then logged with registration.scope. If the service worker is already installed, navigator.serviceWorker.register returns the registration object of the currently active service worker.

The scope of the service worker determines which files the service worker controls, in other words, from which path the service worker will intercept requests. The default scope is the location of the service worker file, and extends to all directories below. So if service-worker.js is located in the root directory, the service worker will control requests from all files at this domain.

You can also set an arbitrary scope by passing in an additional parameter when registering. For example: main.js

```
navigator.serviceWorker.register('/service-worker.js', { scope: '/app/'

});
```

In this case we are setting the scope of the service worker to /app/, which means the service worker will control requests from pages like /app/, /app/lower/ and /app/lower/lower, but not from pages like /app or /, which are higher.

If you want the service worker to control higher pages e.g. /app (without the trailing slash) you can indeed change the scope option, but you'll also need to set the Service-Worker-Allowed HTTP Header in your server config for the request serving the service worker script.

main.js

navigator.serviceWorker.register('/app/service-worker.js', { scope: '/app'

});

Installation

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

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

```
service-worker.js

// Listen for install event, set callback

self.addEventListener('install', function(event) {

// Perform some task

});
```

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 (see the Offline Cookbook for an example).

service-worker.js

self.addEventListener('activate', function(event) {*// Perform some task*

});

Once activated, the service worker controls all pages that load within its scope, and starts listening for events from those pages. However, pages in your app that were loaded before the service worker activation will not be under service worker control. The new service worker will only take over when you close and reopen your app, or if the service worker calls **clients.claim()**. Until then, requests from this page will not be intercepted by the new service worker. This is intentional as a way to ensure consistency in your site.

CODE-

```
Index.html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
           <meta   name="viewport"   content="width=device-width,
initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
          content="Movie  recommendations  app  -  find  your  next
favorite film"
    />
     <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png"
/>
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <title>Movie Recommendations</title>
  </head>
  <body>
       <noscript>You  need  to  enable  JavaScript  to  run  this
app.</noscript>
    <div id="root"></div>
  </body>
</html>
```

```
service-worker.js
importScripts('https://storage.googleapis.com/workbox-cdn/releas
es/6.4.1/workbox-sw.js');

if (workbox) {
  console.log('✅ Workbox loaded');
```

```
const CACHE_NAME = 'pwa-cache-v1';
const PRECACHE_ASSETS = [
  '/',
  '/index.html',
  '/favicon.ico',
  '/logo192.png',
  '/logo512.png',
  '/manifest.json',
  '/static/css/main.css',
  '/static/js/main.js'
];

// Precache assets manually
workbox.precaching.precacheAndRoute(PRECACHE_ASSETS);

// Cache images
workbox.routing.registerRoute(
  ({ request }) => request.destination === 'image',
  new workbox.strategies.CacheFirst({
    cacheName: 'images-cache',
    plugins: [
      new workbox.expiration.ExpirationPlugin({
        maxEntries: 60,
        maxAgeSeconds: 30 * 24 * 60 * 60, // 30 Days
      }),
    ],
  })
);

// Cache CSS & JS
workbox.routing.registerRoute(
      ({ request }) => request.destination === 'script' ||
request.destination === 'style',
  new workbox.strategies.StaleWhileRevalidate({
    cacheName: 'static-resources',
  })
);

// Debug Fetch Requests
self.addEventListener('fetch', (event) => {
                console.log('[Service Worker] Fetching:',
event.request.url);
  event.respondWith(
    caches.match(event.request).then((cachedResponse) => {
      return cachedResponse || fetch(event.request);
    })
  );
});
```

```
  // Install Event
  self.addEventListener('install', (event) => {
    console.log('[Service Worker] Installing...');
    event.waitUntil(
      caches.open(CACHE_NAME).then((cache) => {
        console.log('[Service Worker] Caching assets');
        return cache.addAll(PRECACHE_ASSETS);
      })
    );
    self.skipWaiting();
  });

  // Activate Event
  self.addEventListener('activate', (event) => {
    console.log('[Service Worker] Activating...');
    event.waitUntil(
      caches.keys().then((cacheNames) => {
        return Promise.all(
          cacheNames.map((cache) => {
            if (cache !== CACHE_NAME) {
                console.log('[Service Worker] Deleting old
cache:', cache);
              return caches.delete(cache);
            }
          })
        );
      })
    );
    self.clients.claim();
  });

} else {
  console.log('❌ Workbox failed to load');
}
```

OUTPUT

Dimensions: Responsive 1312 x 928 50% No throttling
MovieHouse
Carjackers
Cleaner
The Quiet Ones
A Working Man
Captain America: Brave New World
Elements Console Sources Network Application
Application
Manifest
Service workers
Storage
Storage
Local storage
Session storage
Extension storage
IndexedDB
Cookies
Private state tokens
Interest groups
Service workers
Offline Update on reload Bypass for network
http://localhost:3000/ Network requests Update
Source service-worker.js
Received 4/4/2025, 1:32:33 PM
Status #6500 activated and is starting
Stop
Clients http://localhost:3000/
Console What's new AI assistance Autofill
Service Worker registered with scope: http://localhost:3000/ index.js:17

Dimensions: Responsive 1312 x 928 50% No throttling
MovieHouse
Carjackers
Cleaner
The Quiet Ones
A Working Man
Captain America: Brave New World
Elements Console Sources Network Application
Source service-worker.js
Received 4/4/2025, 1:32:33 PM
Status #6500 activated and is starting Stop
Clients http://localhost:3000/
http://localhost:3000/
Push Test push message from DevTools. Push
Sync test-tag-from-devtools Sync
Periodic sync test-tag-from-devtools Periodic sync
Update Cycle
Version Update Activity Timeline
#6500 Install
#6500 Wait
#6500 Activate
Service workers from other origins
See all registrations
Console What's new AI assistance Autofill

Elements Console Sources Application
IndexedDB
Cookies
Private state ...
Interest gro...
Shared stora...
Cache storage
pwa-cach...
static-reso...
Storage buc...
Background services
Back/forwar...
Background ...
Background ...
Bounce trac...
Notifications
Payment ha...
Periodic bac...
Speculative l...
Push messa...
Reporting API
Filter by path
http://localhost:3000
Origin http://localhost:3000
Bucket name default
Is persistent No
Durability relaxed
Quota 0 B
Expiration None
Name Res... Cont... Con... Tim... Vary...
0 /static/js/bundle.js basic appl... 0 4/4/... Acce...
1 /css2?family=Anta&family=... opa... text/... 0 4/4/... Sec-...
Total entries: 2

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.	04
Name	Nitish Bhosle
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

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

Theory:

Service Worker

Service Worker is a script that works on browser background without user interaction independently. Also, It resembles a proxy that works on the user side. With this script, you can track network traffic of the page, manage push notifications and develop “offline first” web applications with Cache API.
Things to note about Service Worker:
● A service worker is a programmable network proxy that lets you control how network requests from your page are handled.
● Service workers only run over HTTPS. Because service workers can intercept network requests and modify responses, "man-in-the-middle" attacks could be very bad.
● The service worker becomes idle when not in use and restarts when it's next needed. You cannot rely on a global state persisting between events. If there is information that you need to persist and reuse across restarts, you can use IndexedDB databases.
● Service workers make extensive use of promises, so if you're new to promises, then you should stop reading this and check out Promises, an introduction.

Fetch Event

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

Of course, you can use many different methods but you can find in the following example a “cache first” and “network first” approach. In this example, if the request’s and current location’s origin are the same (Static content is requested.), this is called “cacheFirst” but if you request a targeted external URL, this is called “networkFirst”.
● **CacheFirst** - In this function, if the received request has cached before, the cached response is returned to the page. But if not, a new response requested from the network.
● **NetworkFirst** - In this function, firstly we can try getting an updated response from the network, if this process completed successfully, the new response will be cached and returned. But if this process fails, we check whether the request has been cached before or not. If a cache exists, it is returned to the page, but if not, this is up to you. You can return dummy content or information messages to the page.

Sync Event

Background Sync is a Web API that is used to delay a process until the Internet connection is stable. We can adapt this definition to the real world; there is an e-mail client application that works on the browser and we want to send an email with this tool.

Internet connection is broken while we are writing e-mail content and we didn't realize it. When completing the writing, we click the send button.
Here is a job for the Background Sync.
The following view shows the classical process of sending email to us. If the Internet Connection is broken, we can't send any content to Mail Server.

Push Event

This is the event that handles push notifications that are received from the server.You can apply any method with received data.
We can check in the following example.
"Notification.requestPermission();" is the necessary line to show notification to the user. If you don't want to show any notification, you don't need this line.
In the following code block is in sw.js file. You can handle push notifications with this event. In this example, I kept it simple. We send an object that has "method" and "message" properties. If the method value is "pushMessage", we open the information notification with the "message" property.

CODE-

```
Service-worker.js

//  Import Workbox

importScripts('https://storage.googleapis.com/workbox-cdn/releas
es/6.4.1/workbox-sw.js');


if (workbox) {

  console.log(' Workbox Loaded Successfully');


  // Precache assets

  workbox.precaching.precacheAndRoute(self.__WB_MANIFEST || []);


  //  Cache Images (Cache-First Strategy)

  workbox.routing.registerRoute(

    ({ request }) => request.destination === 'image',
```

```
    new workbox.strategies.CacheFirst({
      cacheName: 'images-cache',
      plugins: [
        new workbox.expiration.ExpirationPlugin({
          maxEntries: 60,
          maxAgeSeconds: 30 * 24 * 60 * 60, // 30 Days
        }),
      ],
    })
  );

  //  Cache CSS & JS (Stale-While-Revalidate Strategy)
  workbox.routing.registerRoute(
    ({ request }) => request.destination === 'script' ||
request.destination === 'style',
    new workbox.strategies.StaleWhileRevalidate({
      cacheName: 'static-resources',
    })
  );

  // Cache API Responses (Network-First Strategy)
  workbox.routing.registerRoute(
    ({ url }) => url.origin.includes('api.themoviedb.org'),
    new workbox.strategies.NetworkFirst({
      cacheName: 'api-cache',
      plugins: [
        new workbox.expiration.ExpirationPlugin({
          maxEntries: 50,
```

```
          maxAgeSeconds: 5 * 60, // 5 minutes
        }),
      ],
    })
  );

}

//  Fetch Event Logging (Works outside Workbox)
self.addEventListener('fetch', (event) => {
    console.log(` Fetch event detected: ${event.request.url}`);

    if (event.request.url.includes('api.themoviedb.org')) {
      console.log(`Intercepting API Request:
${event.request.url}`);

      event.respondWith(
        caches.match(event.request).then((cachedResponse) => {
          return cachedResponse ||
fetch(event.request).then((response) => {
            console.log(` API Response Fetched:
${event.request.url}`);
            return response;
          }).catch((err) => {
            console.error(` API Fetch Failed:
${event.request.url}`, err);
            return new Response('API fetch failed', { status:
500 });
          });
```

```
        })
      );
    }
  });

//  Background Sync Event (Syncing Watchlist)
self.addEventListener('sync', (event) => {
  if (event.tag === 'sync-watchlist') {
    console.log(' Sync event triggered: sync-watchlist');
    event.waitUntil(
      syncWatchlist().then(() => {
        console.log(' Sync successful');
      }).catch((err) => {
        console.error(' Sync failed:', err);
      })
    );
  }
});

//  Example Function to Sync Watchlist
async function syncWatchlist() {
  console.log(' Syncing watchlist data...');
  return fetch('/sync-watchlist', { method: 'POST' })
    .then(() => console.log(' Sync request sent successfully!'))
    .catch(() => console.log(' Sync request failed, retrying
later.'));
}
```

```
//  Push Notification Event
self.addEventListener('push', (event) => {
  console.log('Push notification received');

  const notificationData = event.data ? event.data.text() ;
  console.log(`📨 Push payload: ${notificationData}`);

  const options = {
    body: notificationData,
    icon: '/logo192.png',
    badge: '/logo192.png',
  };

  event.waitUntil(
    self.registration.showNotification(' Movie House', options)
      .then(() => console.log(' Push notification sent
successfully'))
      .catch((err) => console.error(' Push notification
failed:', err))
  );
});

//  Activate event - Cleanup old caches
self.addEventListener('activate', (event) => {
  console.log(' Service Worker activated');
  event.waitUntil(
    caches.keys().then((cacheNames) =>
      Promise.all(
```

```
        cacheNames.map((cache) => {
          if (!['images-cache', 'static-resources',
'api-cache'].includes(cache)) {
            console.log('🗑️ Deleting old cache:', cache);
            return caches.delete(cache);
          }
        })
      )
    )
  );
  self.clients.claim();
});
```

OUTPUT

Service worker activated

Project Title: Friends Making app **Roll No. 04**

Fetch Event

Sync Event

Push Event

MAD & PWA Lab Journal

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	04
Name	Nitish Bhosle
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

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 Github Repository-

[NitiSX27/PWA]

OUTPUT-

MAD & PWA Lab Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	04
Name	Nitish Bhosle
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	

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

Theory:

Google Lighthouse is an open-source tool that audits web applications based on multiple key parameters, including performance, accessibility, Progressive Web App (PWA) implementation, and best practices. It provides a detailed, automated report that helps developers optimize their websites efficiently. Unlike traditional manual audits, which can take days or even weeks, Lighthouse generates insights within minutes.
One of the key advantages of Lighthouse is its ease of use—no complex setup is required. Simply run it on a webpage or provide a URL, and it will generate an extensive performance report.

Key Features and Audit Metrics
Lighthouse can audit both desktop and mobile versions of a webpage. The core evaluation criteria include:
1. Performance
This metric measures how efficiently a webpage loads and displays content. Key factors influencing the performance score include:

- Page load speed – How quickly the page becomes visible to the user.
- First Contentful Paint (FCP) – The time taken for the first piece of content to appear.
- Largest Contentful Paint (LCP) – The time taken for the main content to fully load.
- Cumulative Layout Shift (CLS) – Measures how visually stable a page is (i.e., avoiding unexpected shifts in content).
- Time to Interactive (TTI) – The time it takes for the page to

become fully functional. Lighthouse assigns a score from 0 to 100 based on percentile rankings, where:

- 100 → Top 2% of websites (98th percentile)
- 50 → Around the 75th percentile
- Lower scores → Indicate areas that need optimization

2. Progressive Web App (PWA) Score (Mobile)
With the rise of PWAs, modern web applications aim to provide a native app-like experience. Lighthouse evaluates the PWA implementation based on Google's Baseline PWA Checklist, which includes:

- Service Worker implementation – Ensuring offline support and background synchronization.
- App Manifest compliance – Providing metadata for better mobile integration.
- Viewport configuration – Optimizing mobile responsiveness.
- Performance in script-disabled environments – Ensuring the page functions even when JavaScript is disabled.

A high PWA score indicates that the application meets essential PWA criteria and provides an app-like user experience.

3. Accessibility

Accessibility ensures that web applications are usable by individuals with disabilities. Lighthouse audits a webpage based on:

- ARIA attributes – Enhancing accessibility through attributes like aria-required.
- Text alternatives for media – Ensuring audio and visual content is accessible.
- Semantic HTML – Proper use of <section>, <article>, <button>, and other elements that improve screen-reader compatibility.

Unlike other metrics, accessibility checks follow a pass/fail approach—if a necessary feature is missing, it significantly impacts the score. A higher accessibility score ensures inclusivity for users with visual or cognitive impairments.

4. Best Practices

This metric evaluates whether the website follows modern web development best practices, including:

- Use of HTTPS – Ensuring secure data transmission.
- Avoiding deprecated code – Preventing the use of outdated elements, directives, and libraries.
- Secure password inputs – Disabling paste-into fields to mitigate credential theft risks.
- User security alerts – Prompting users about geo-location access

and cookie usage on load. A high score indicates that the website follows industry standards, improving security, usability, and maintainability.

Manifest.json

```
{
 "id": "/",
 "name": "TOURLY - Travel
 Ecommerce", "short_name": "TLY",
 "description": "A travel
ecommerce platform for booking
tours and experiences.",
"start_url": "/index.html",
 "scope": "/",
 "display":
 "standalone",
 "background_color":
```

```
 "#FFFFFF",
 "theme_color":
 "#000000",
 "orientation":
 "portrait",
 "lang": "en",
 "categories": ["travel",
"shopping", "ecommerce"],
 "icons": [
  {
   "src":
   "assets/images/logo_tourly_192
   .png", "sizes": "192x192",
   "type": "image/png",
   "purpose": "any
   maskable"

},
```

```
 {
  "src":
  "assets/images/logo_tourly_5
  12.png", "sizes": "512x512",
  "type":
  "image/png",
  "purpose": "any
  maskable"
 },
 {

 "src":
  "assets/images/logo_tourly_192.
  webp", "sizes": "192x192",
  "type":
  "image/webp",
  "purpose": "any
  maskable"
 },
 {
  "src":
"assets/images/logo_tourly_512
.webp",
  "sizes":
  "512x512",
  "type":
  "image/webp",
  "purpose": "any
  maskable"
 }
],
"screenshots": [
 {
```

```
    "src":
     "assets/screenshots/homepage.
     png", "sizes": "1080x1920",
     "type": "image/png"
    },
    {
     "src":
     "assets/screenshots/booking.p
     ng", "sizes": "1080x1920",
     "type": "image/png"
    }
   ],
   "shortcuts": [
    {
"name": "Book a Tour",
"short_name": "Book",
     "description": "Go directly to
     tour booking", "url":
     "/book.html",
     "icons": [
      {
       "src":
       "images/shortcut-icon.pn
       g", "sizes": "96x96",
       "type": "image/png"

      }
     ]
    },
    {
     "name": "View Popular
     Tours", "short_name":
     "Popular", "description":
     "Explore popular tours",
     "url": "/popular.html",
     "icons": [
      {
       "src":
   "images/popular-shortcut-
   icon.png",
       "sizes":
       "96x96",
       "type":
       "image/png"
      }
     ]
    }
   ],
    "iarc_rating_id":
   "e10c6b5e-3b3c-4d3a-9f3b-
   1e5f3e2b5f3c",
    "dir": "ltr",
    "related_appl
    ications": [
     {
      "platform
      ":
      "play",
      "url":
   "https://play.google.com/store/
   apps/details?id=c
   om.tourly.app",
      "id": "com.tourly.app"
     },

     "platform":
     "itunes",
     "url":
   "https://apps.apple.com/app/id123
   4567890"
     }
    ],
    "prefer_related_applications":
    false
   }
```

Project Title: Friends Making app **Roll No. 04**

Project Title: Friends Making app **Roll No. 04**

Project Title: Friends Making app **Roll No. 04**

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.	
Name	
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:	

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.	
Name	
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:	