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.

Department of Information Technology

CERTIFICATE

This is to certify that **Harshita Dubey** 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.

Project Title: Flutter : Thread Clone/ PWA : Scribble **RollNo. 14**

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			15
2.	To design Flutter UI by including common widgets.	LO2			15
3.	To include icons, images, fonts in Flutter app	LO2			15
4.	To create an interactive Form using form widget	LO2			15
5.	To apply navigation, routing and gestures in Flutter App	LO2			15
6.	To Connect Flutter UI with fireBase database	LO3			15
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

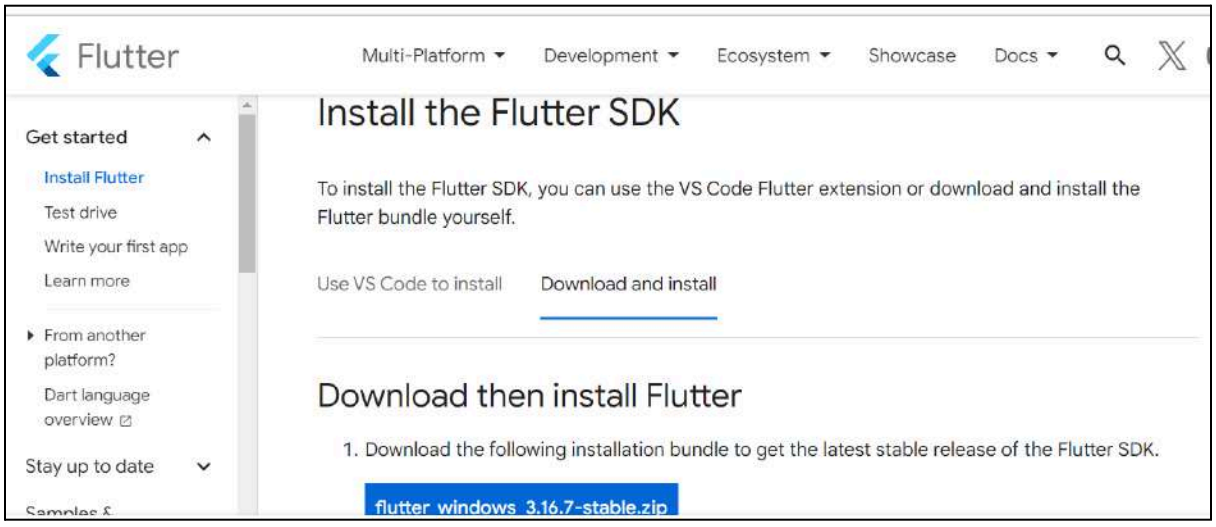
Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	14
Name	Harshita Dubey
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	15

MPL Experiment-01

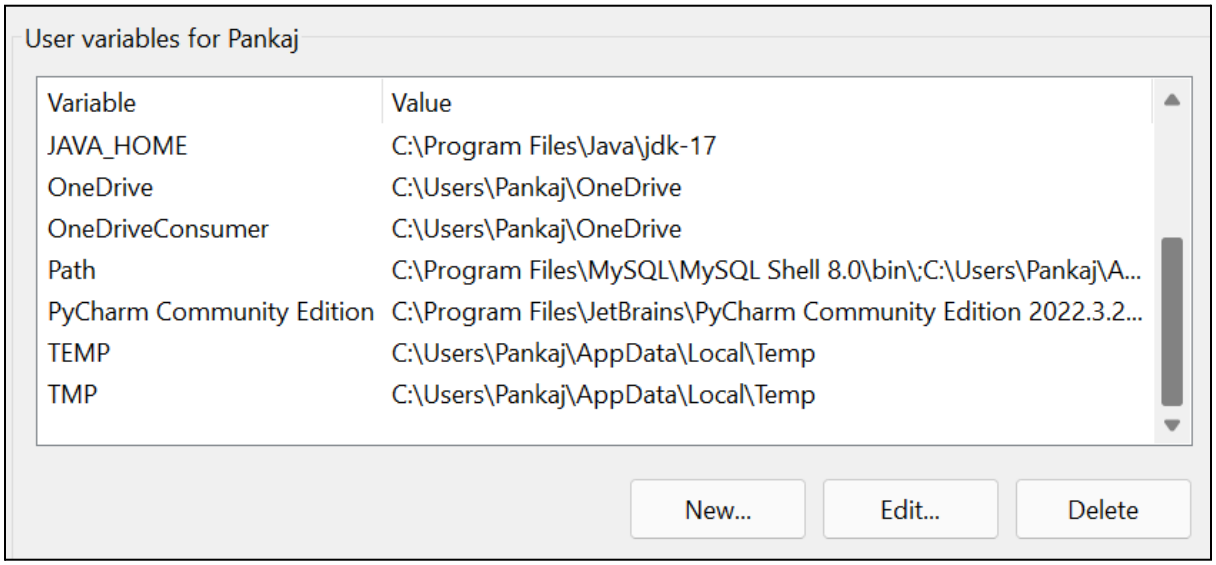
Name: Harshita Dubey Roll no:14 Batch:A/D15A

Aim : Create a 'Hello World' App using Flutter

Step 1: Install Flutter SDK



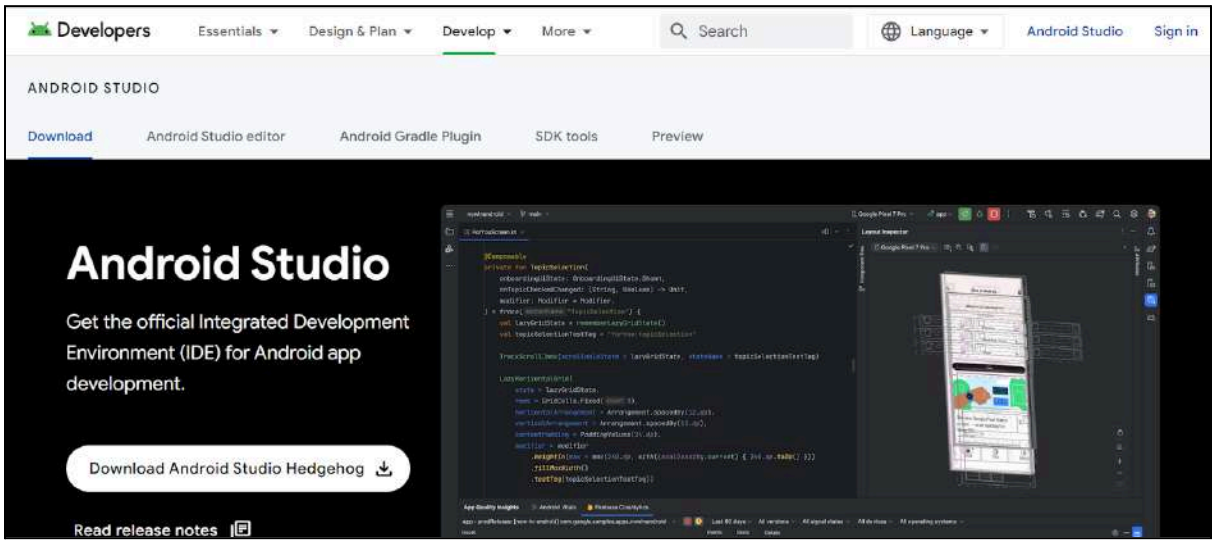
Step 2: Set up the Environment Variables

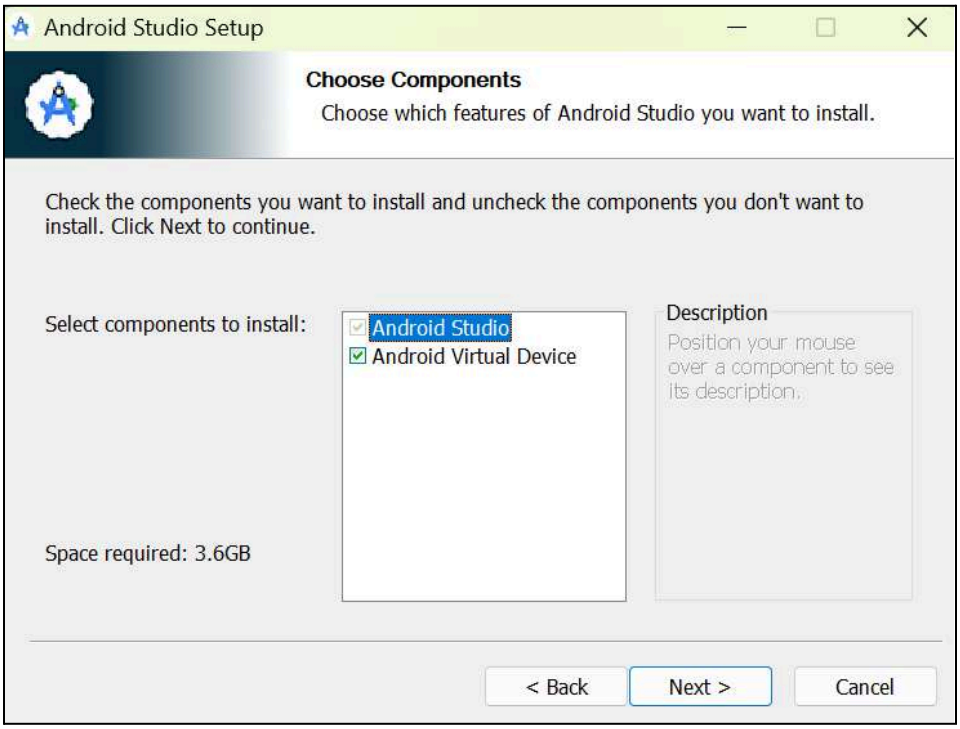


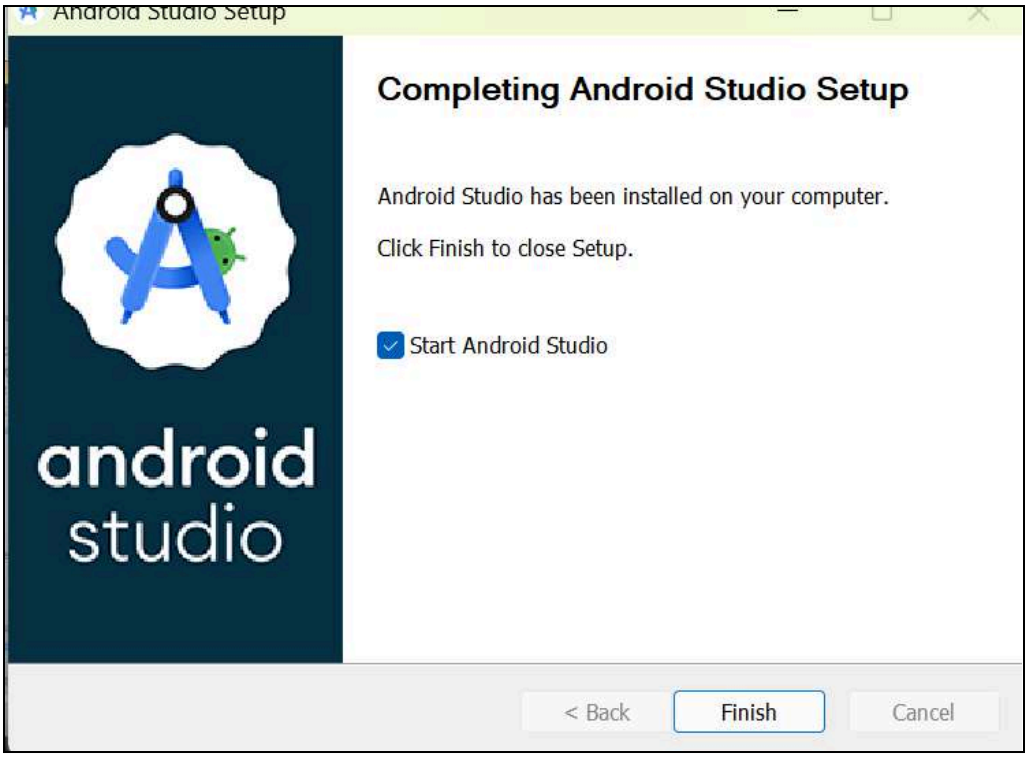
Step 3: Check if Flutter is installed correctly

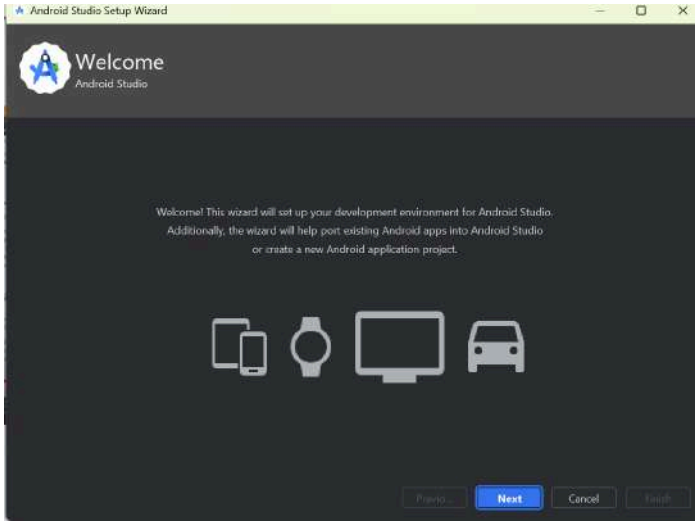
```
PS C:\Users\Pankaj> flutter doctor

                Welcome to Flutter! - https://flutter.dev

The Flutter tool uses Google Analytics to anonymously report feature usage
statistics and basic crash reports. This data is used to help improve
Flutter tools over time.

Flutter tool analytics are not sent on the very first run. To disable
reporting, type 'flutter config --no-analytics'. To display the current
setting, type 'flutter config'. If you opt out of analytics, an opt-out
event will be sent, and then no further information will be sent by the
Flutter tool.

By downloading the Flutter SDK, you agree to the Google Terms of Service.
The Google Privacy Policy describes how data is handled in this service.

Moreover, Flutter includes the Dart SDK, which may send usage metrics and
crash reports to Google.

Read about data we send with crash reports:
https://flutter.dev/docs/reference/crash-reporting

See Google's privacy policy:
https://policies.google.com/privacy

To disable animations in this tool, use 'flutter config --no-animations'.
```

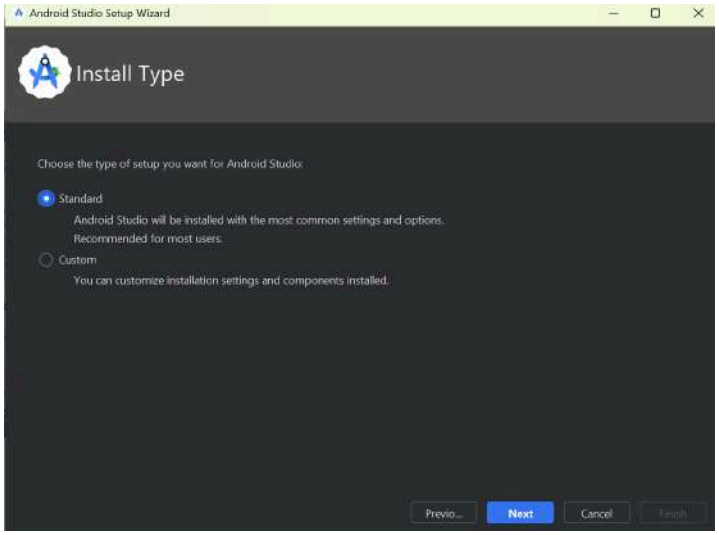
Step 4: Download the Android Studio and click on next for installation

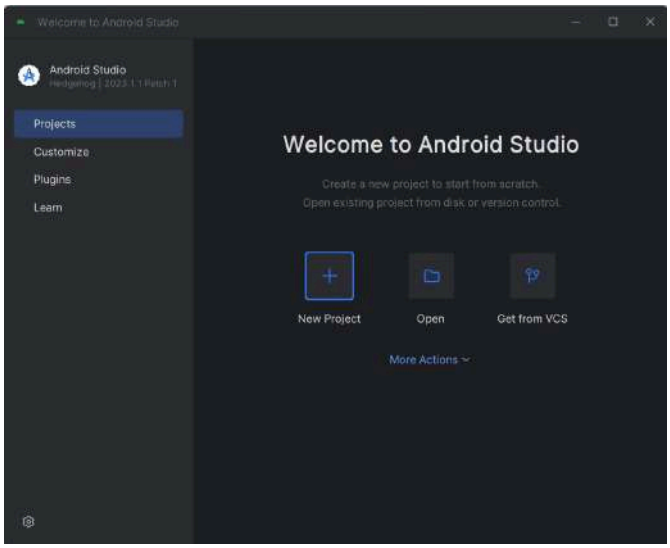



Android Studio Setup
Choose Components
Choose which features of Android Studio you want to install.
Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.
Select components to install:
Android Studio
Android Virtual Device
Description
Position your mouse over a component to see its description.
Space required: 3.6GB
< Back
Next >
Cancel

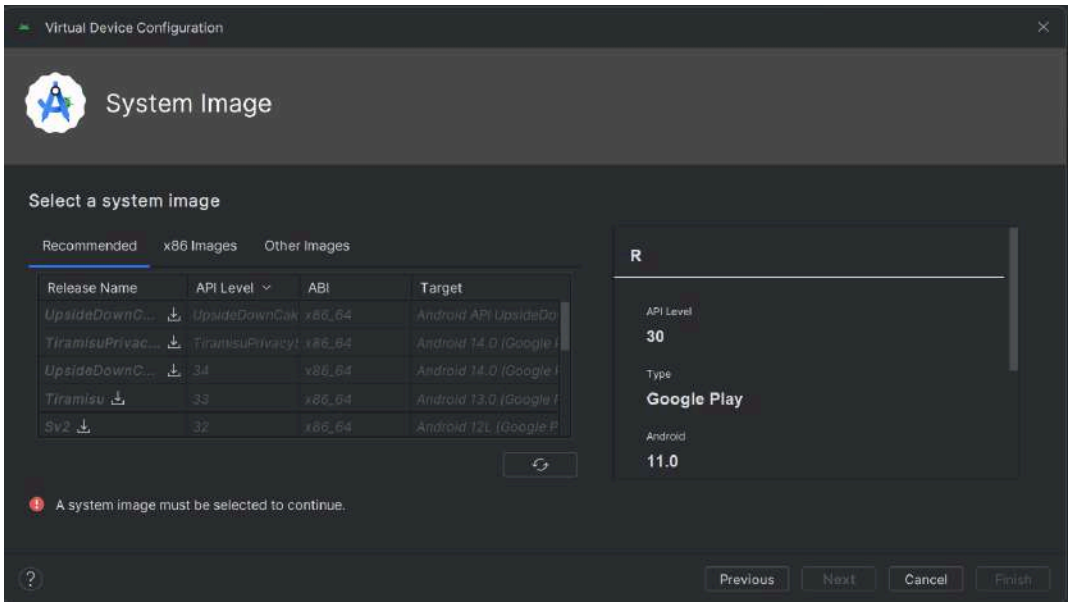

Android Studio Setup
Completing Android Studio Setup
Android Studio has been installed on your computer.
Click Finish to close Setup.
Start Android Studio
android studio
< Back
Finish
Cancel

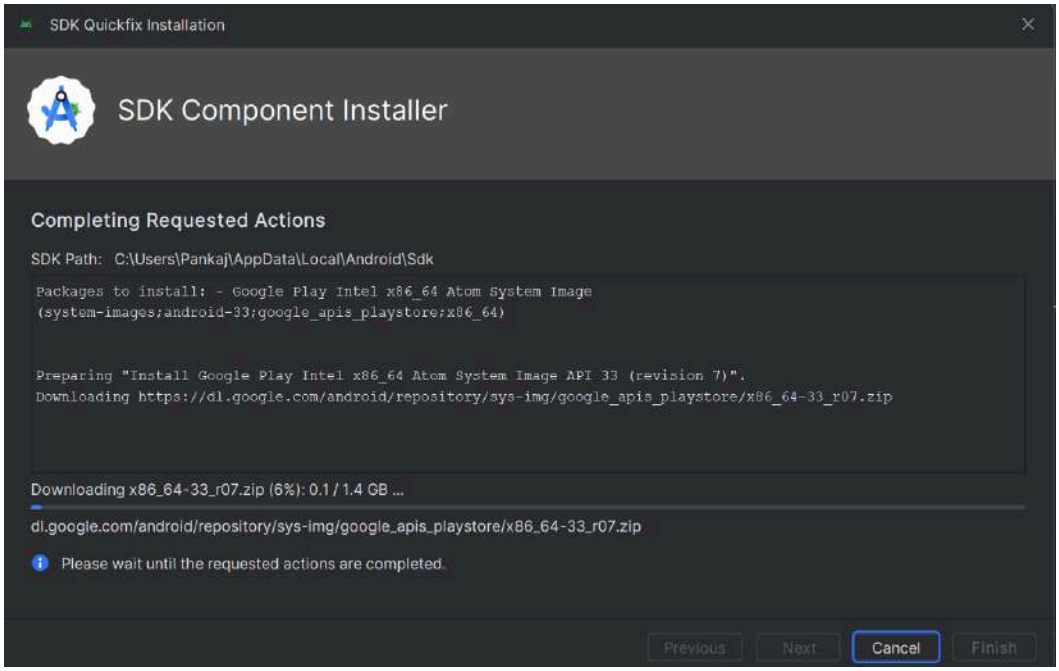

Android Studio Setup Wizard
Welcome
Android Studio
Welcome! This wizard will set up your development environment for Android Studio.
Additionally, the wizard will help port existing Android apps into Android Studio
or create a new Android application project.
Next
Cancel

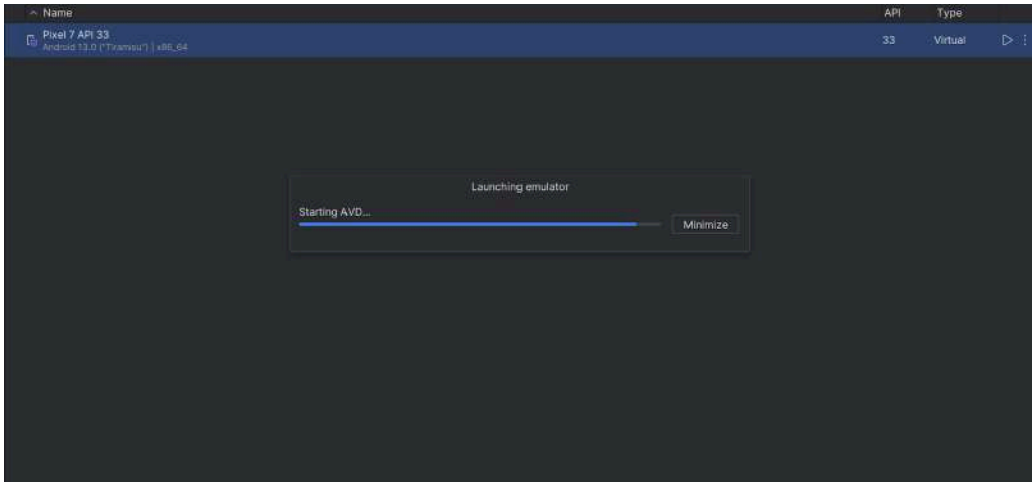

Android Studio Setup Wizard
Install Type
Choose the type of setup you want for Android Studio:
Standard
Android Studio will be installed with the most common settings and options.
Recommended for most users.
Custom
You can customize installation settings and components installed.
Previo...
Next
Cancel

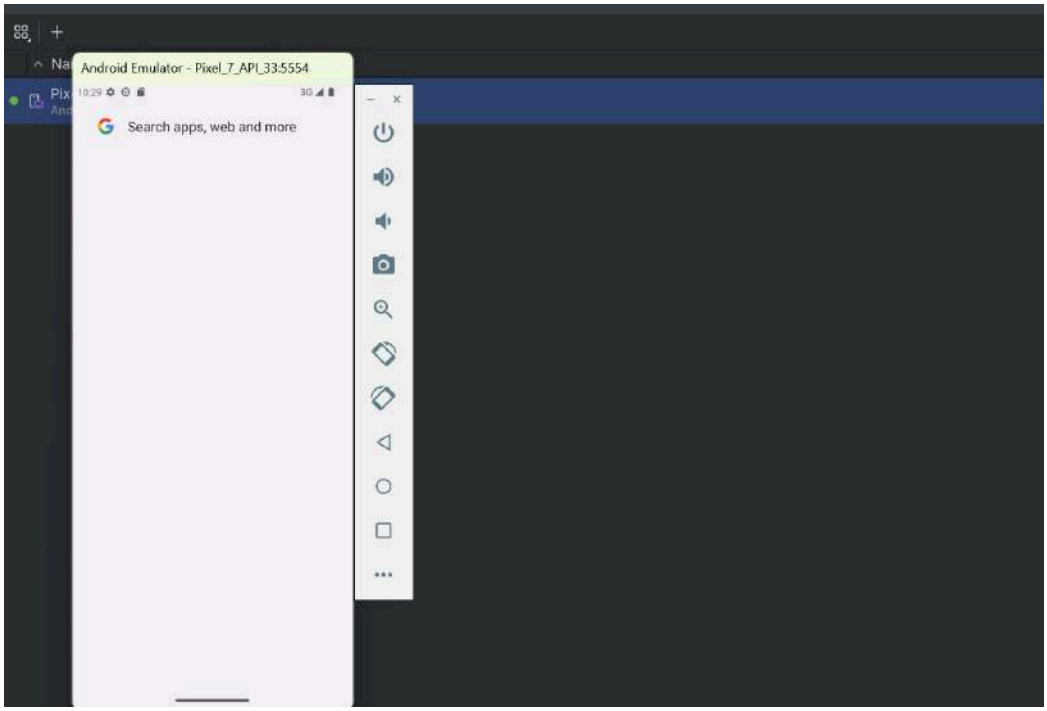

Welcome to Android Studio
Android Studio
Projects
Customize
Plugins
Learn
Welcome to Android Studio
Create a new project to start from scratch.
Open existing project from disk or version control.
New Project
Open
Get from VCS
More Actions

Step 5: Open Virtual Device Manager and select any system image(eg; Tiramisu) and install it….
You are good to go. Select the device(Eg: Pixel 7 API 33) and you have successfully setup the Emulator









Step 6: Open VSCode and install Extension 'Flutter'

Step 7: Hit Ctrl+Shift+P and select 'Flutter New Project' and then Appication. Open the folder you want to built project in..

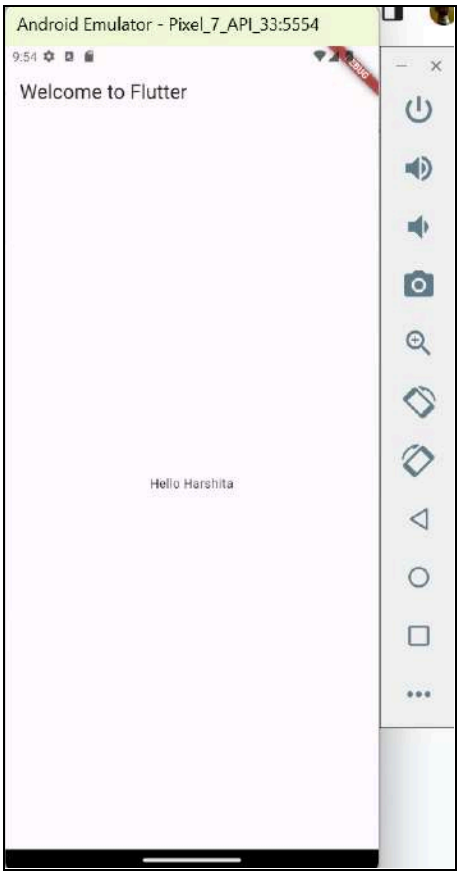
Step 8: Open the lib/main.dart file

Step 9: Write the code there

```dart
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 Harshita's Application",
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Welcome to Flutter'),
        ),
        body: const Center(
          child: Text('Hello Harshita'),
        ),
      ),
    );
  }
}
```

Step 8: Open the terminal and enter ‘run Flutter’.

Step 9: In Emulator, it will be displayed as


Android Emulator - Pixel_7_API_33:5554
9:54
Welcome to Flutter
Hello Harshita

MAD & PWA Lab
Journal

Experiment No.	02
Experiment Title.	To design Flutter UI by including common widgets.
Roll No.	14
Name	Harshita Dubey
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

Aim: To design Flutter UI by including common widgets.

Theory:

Designing a Flutter UI involves combining and customizing a variety of common widgets to create a visually appealing and functional user interface.

Widgets

In Flutter, widgets are the building blocks of the UI. Widgets can be either stateless or stateful and can be combined to create complex UIs.

1. StatelessWidget:

● A basic building block in Flutter.
● Represents part of the user interface that can be described by a configuration that cannot change over time. Example: Container, Icon, Text.

2. StatefulWidget:

● Represents part of the user interface that can change dynamically.
● Has mutable state that affects its appearance. Example: TextField, Checkbox, Radio.

BASIC WIDGETS

- Container: Box model for layout and styling.
- Row and Column: Horizontal and vertical layout.
- Stack: Stacks widgets on top of each other. ListView: Scrollable list of widgets. GridView: Scrollable grid layout.
- Card: Material design card for grouping info.
- AppBar: Top app bar with title and actions.
- TextField: User text input field.
- Buttons: ElevatedButton, TextButton, OutlinedButton.
- Icon: Displays material design icons.
- Image: Displays images with various sources.
- Divider: Creates horizontal or vertical lines.
- SizedBox: Box with specified width and height.

- Expanded: Takes up remaining space.
- Flexible: Adjusts flex factor in a Flex widget.
- PageRouteBuilder: Customizable page transitions.
- ClipRRect and ClipOval: Clipping with rounded corners or oval shape.
- Sliver Widgets: Advanced scrolling in CustomScrollView.

Code:

Make a folder screens in the lib folder. Under screens folder make a home.dart,feed.dart,post_screen.dart,favorite_screen.dart,post_screen.dart ,sear ch.dart,profile_screen.dart

Main.dart

```
import 'package:flutter/material.dart';
 import 'screens/login.dart'; void main() {
 runApp(const MyApp());

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

@override

Widget build(BuildContext context) {
return MaterialApp( title: 'Flutter Demo', theme: ThemeData(), home:const
LoginScreen(), );

} }
```

Home.dart

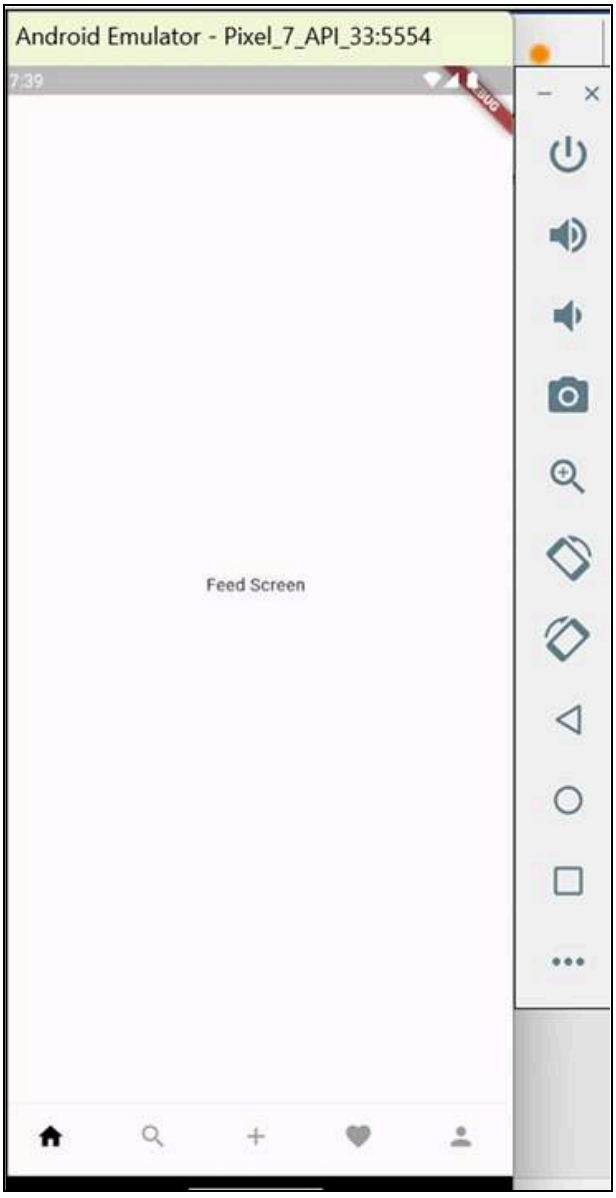
```
import 'package:flutter/material.dart';
import 'package:my_app/screens/favorite_screen.dart';
 import 'package:my_app/screens/feed.dart';
```

```
import 'package:my_app/screens/post_screen.dart';
import 'package:my_app/screens/profile_screen.dart';
import 'package:my_app/screens/search.dart';

class Home extends StatefulWidget {

const Home({Key? key}) : super(key: key);

@override

State createState() => _HomeState();
}

class _HomeState extends State {

int selectedIndex = 0; List pages = [ const FeedScreen(),
const SearchScreen(),
const PostScreen(),
const FavoriteScreen(),
const ProfileScreen(),
]; // You need to populate this list with your pages

@override

Widget build(BuildContext context) {

return Scaffold( body: pages[selectedIndex],
bottomNavigationBar: BottomNavigationBar(
currentIndex: selectedIndex, selectedItemColor: Colors.black,
unselectedItemColor: Colors.grey, showSelectedLabels: false,
showUnselectedLabels: false, onTap: (index){

 setState(() { selectedIndex=index; }); }, type: BottomNavigationBarType.fixed,
items: const [ BottomNavigationBarItem(icon: Icon(Icons.home),label: 'Feed',),
BottomNavigationBarItem(icon: Icon(Icons.search),label: 'Search',),
BottomNavigationBarItem(icon: Icon(Icons.add),label: 'post',),
```

```
BottomNavigationBarItem(icon: Icon(Icons.favorite),label: 'favorite',),
BottomNavigationBarItem(icon: Icon(Icons.person),label: 'profile',), ], ), );
 }
 }
```

feed.dart

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

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

@override

State createState() => _FeedScreenState();

}
class _FeedScreenState extends State {

@override

Widget build(BuildContext context) {
 return const Scaffold( body: Center( child: Text("Feed Screen"),
),
);
 }
}
```



search.dart

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:thread_clone_flutter/model/user.dart';

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

  @override
```

```
  State<SearchScreen> createState() => _SearchScreenState();
}

class _SearchScreenState extends State<SearchScreen> {
  final CollectionReference userCollection =
      FirebaseFirestore.instance.collection('users');

  final userId = FirebaseAuth.instance.currentUser!.uid;

  String searchQuery = "";
  final searchController = TextEditingController();
  List<UserModel> searchUsers(List<UserModel> users, String query) {
    return users.where((user) {
      return user.username.toLowerCase().contains(query.toLowerCase());
    }).toList();
  }

  Future<void> followUser(UserModel user) async {
    await userCollection.doc(userId).update({
      'following': FieldValue.arrayUnion([user.id])
    });
    await userCollection.doc(user.id).update({
      'followers': FieldValue.arrayUnion([userId])
    });
  }

  Future<void> unFollowUser(UserModel user) async {
    await userCollection.doc(userId).update({
      'following': FieldValue.arrayRemove([user.id])
    });
    await userCollection.doc(user.id).update({
      'followers': FieldValue.arrayRemove([userId])
    });
  }

  @override
  void initState() {
    searchController.addListener(() {
      setState(() {
        searchQuery = searchController.text;
```

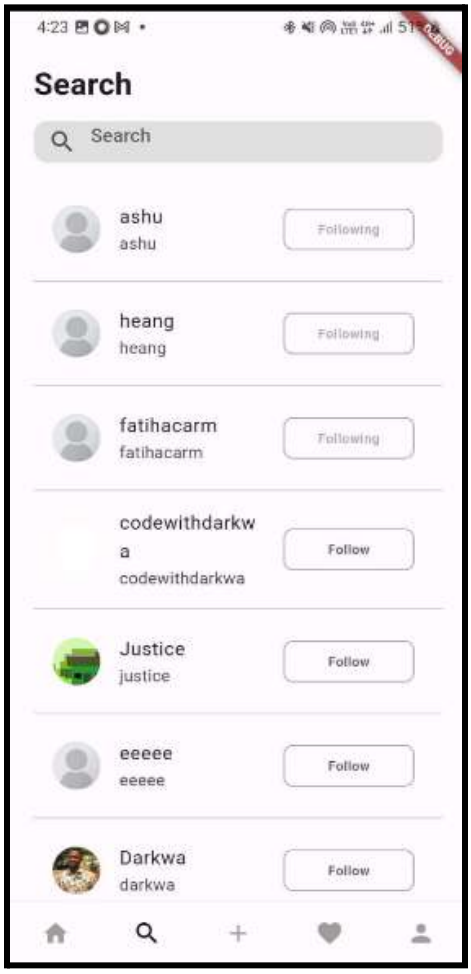
```
    });
  });
  super.initState();
 }

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   body: SafeArea(
    child: SingleChildScrollView(
     child: Padding(
      padding: const EdgeInsets.all(20.0),
      child: Column(
       crossAxisAlignment: CrossAxisAlignment.start,
       children: [
        const Text(
         'Search',
         style: TextStyle(fontSize: 26, fontWeight: FontWeight.bold),
        ),
        Padding(
         padding: const EdgeInsets.only(top: 12.0),
         child: Container(
          width: double.infinity,
          height: 35,
          decoration: BoxDecoration(
           color: Colors.grey[300],
           borderRadius: BorderRadius.circular(12),
          ),
          child: TextFormField(
           controller: searchController,
           decoration: const InputDecoration(
            hintText: 'Search',
            border: InputBorder.none,
            prefixIcon: Icon(Icons.search),
           ),
          ),
         ),
        ),
        const SizedBox(height: 20),
        StreamBuilder(
```

```
stream: userCollection
    .where('id', isNotEqualTo: userId)
    .snapshots(),
builder: (context, snapshot) {
 if (!snapshot.hasData) {
   return const Center(
     child: CircularProgressIndicator(),
   );
 } else if (snapshot.hasError) {
   return Center(
     child: Text('Error: ${snapshot.error}'),
   );
 }
 final users = snapshot.data!.docs;

 final allUsers = users.map((doc) {
   final user = doc.data() as Map<String, dynamic>;
   return UserModel(
     id: user['id'],
     username: user['username'],
     profileImageUrl: user['profileImageUrl'],
     name: user['name'],
     followers: [],
     following: [],
   );
 }).toList();
 final filteredUsers = searchUsers(allUsers, searchQuery);
 return ListView.builder(
   shrinkWrap: true,
   itemCount: filteredUsers.length,
   itemBuilder: (contex, index) {
     final user = filteredUsers[index];

     return SuggestedFollowerWidget(
       user: user,
       follow: () => followUser(user),
       unFollow: () => unFollowUser(user),
     );
   },
 );
```

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

class SuggestedFollowerWidget extends StatefulWidget {
  const SuggestedFollowerWidget({
    super.key,
    required this.user,
    required this.follow,
    required this.unFollow,
  });

  final UserModel user;
  final VoidCallback follow;
  final VoidCallback unFollow;

  @override
  State<SuggestedFollowerWidget> createState() =>
      _SuggestedFollowerWidgetState();
}

class _SuggestedFollowerWidgetState extends State<SuggestedFollowerWidget> {
  @override
  Widget build(BuildContext context) {
    final userId = FirebaseAuth.instance.currentUser!.uid;
    return Column(
      children: [
        ListTile(
          leading: CircleAvatar(
            backgroundImage: NetworkImage(widget.user.profileImageUrl ??

"https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRz8cLf8-P2P8GZ0-KiQ-OXp
ZQ4bebpa3K3Dw&usqp=CAU"),
```

```
  backgroundColor: Colors.white,
),
title: Text(widget.user.username),
subtitle: Text(widget.user.username.toLowerCase()),
trailing: StreamBuilder(
   stream: FirebaseFirestore.instance
      .collection('users')
      .doc(userId)
      .snapshots(),
   builder: (context, snapshot) {
    if (!snapshot.hasData) {
     return const SizedBox.shrink();
    }
    final currentUser = UserModel.fromMap(
       snapshot.data!.data() as Map<String, dynamic>);
    final isFollowing =
       currentUser.following.contains(widget.user.id);
    return Row(
     mainAxisSize: MainAxisSize.min,
     children: [
      InkWell(
       onTap: isFollowing ? widget.unFollow : widget.follow,
       child: Container(
        alignment: Alignment.center,
        width: 110,
        height: 35,
        decoration: BoxDecoration(
         border: Border.all(color: Colors.grey),
         borderRadius: BorderRadius.circular(8),
        ),
        child: isFollowing
           ? const Text(
              'Following',
              style: TextStyle(color: Colors.grey),
             )
           : const Text('Follow'),
       ),
      )
     ],
    );
```

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



CONCLUSION

In conclusion, the process of designing a Flutter UI by incorporating common widgets has proven to be effective and user-friendly.I learned many new concepts in flutter widgets.

MAD & PWA Lab
Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	14
Name	Harshita Dubey
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

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

Theory:

Icons:

Icons in Flutter are primarily used to represent actions or features in the user interface. Flutter provides a wide range of built-in icons through the Icons class. These icons cover common actions, such as navigation, file operations, communication, and more.

To use icons in your Flutter app, follow these steps:

- Import the material.dart package if you haven't already, as it contains the Icons class.
- Use the Icon widget to display an icon. You can set the icon using the Icons class directly or by specifying a custom icon.Optionally, wrap the Icon widget with other widgets like IconButton to add interactivity.

Example:

```
import 'package:flutter/material.dart';
        // Using a built-in icon
Icon(Icons.home)

// Using a custom icon
Icon(
 IconData(0xe801, fontFamily: 'MyCustomIcons'), // Replace 0xe801 with your icon's code point
 size: 24.0,
 color: Colors.red,
)
```

- Images:
  Images play a crucial role in enhancing the visual appeal of your Flutter app. Flutter supports various image formats, including JPEG, PNG, GIF, WebP, and animated WebP.
  Here's how to use images in your Flutter app:
  - Store your image files in the project directory. Typically, images are stored under the assets folder.
  - Declare the image assets in the pubspec.yaml file to make them accessible within your app.
  - Use the Image widget to load and display the images. You can use either Image.asset for images stored in the assets folder or Image.network for images fetched from the internet.

Example:

Flutter:
assets:
- assets/images/my_image.png

Image.asset('assets/images/my_image.png')

- Fonts:
  Custom fonts allow you to give your Flutter app a unique typographic identity. You can use custom fonts for text styling throughout your app.
  To include custom fonts in your Flutter app:
  - Add the font files (usually .ttf or .otf files) to your project directory.
  - Declare the font files in the pubspec.yaml file, specifying the font family name and the file paths.
  - Apply the custom font using the fontFamily property of the TextStyle widget when styling text.
- Example:

flutter:
 fonts:
 - family: MyCustomFont
 fonts:
 - asset: assets/fonts/my_font.ttf

dart
Copy code
Text(
 'Custom Font Text',
 style: TextStyle(
 fontFamily: 'MyCustomFont',
 fontSize: 20.0,
 ),
)

Code:

Make a folder screens in the lib folder. Under screens folder make a home.dart,feed.dart,post_screen.dart,favorite_screen.dart,post_screen.dart ,sear ch.dart,profile_screen.dart

Main.dart

```
import 'package:flutter/material.dart';
 import 'screens/login.dart'; void main() {
 runApp(const MyApp());

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

@override

Widget build(BuildContext context) {
return MaterialApp( title: 'Flutter Demo', theme: ThemeData(), home:const
LoginScreen(), );

} }
```

Favorite_Screen.dart

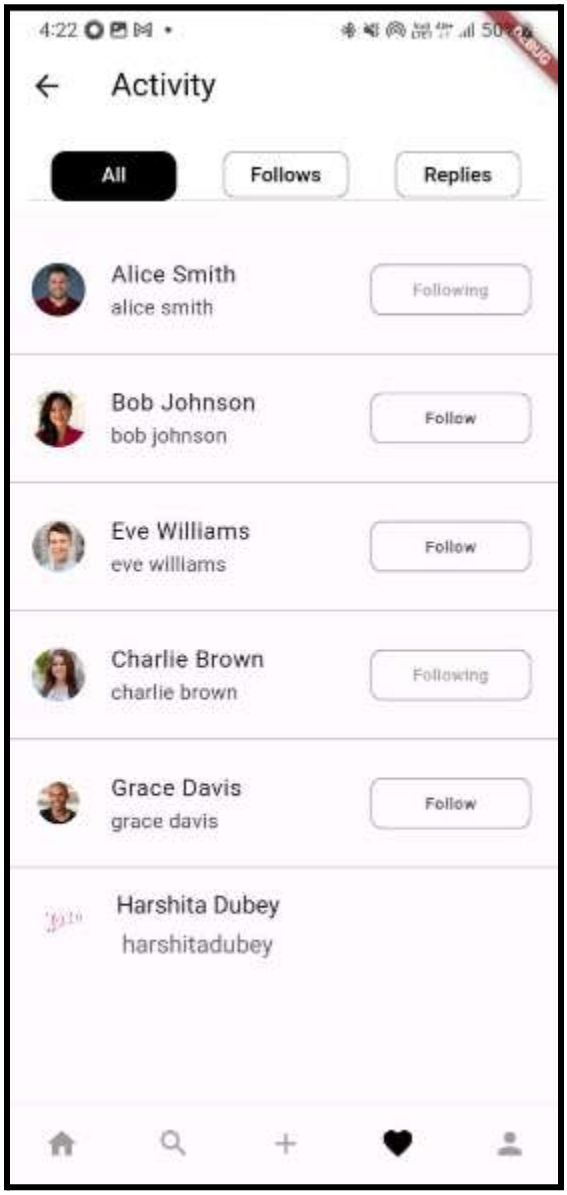
```
import 'package:flutter/material.dart';

import '../model/suggested_follower.dart';

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

  @override
  State<FavoriteScreen> createState() => _FavoriteScreenState();
}

class _FavoriteScreenState extends State<FavoriteScreen> {
  int _currentIndexTab = 0;

  @override
```

```
Widget build(BuildContext context) {
  return DefaultTabController(
    length: 3,
    child: Scaffold(
      appBar: AppBar(
        elevation: 0,
        backgroundColor: Colors.white,
        title: const Text(
          'Activity',
          style: TextStyle(color: Colors.black),
        ),
        bottom: PreferredSize(
          preferredSize: Size.fromHeight(AppBar().preferredSize.height),
          child: Container(
            height: 44,
            padding: const EdgeInsets.symmetric(horizontal: 15, vertical: 5),
            child: TabBar(
              labelColor: Colors.white,
              unselectedLabelColor: Colors.black,
              indicator: BoxDecoration(
                  borderRadius: BorderRadius.circular(10),
                  color: Colors.black),
              onTap: (index) => setState(() => _currentIndexTab = index),
              tabs: [
                Container(
                  width: double.infinity,
                  decoration: BoxDecoration(
                    border: Border.all(
                        color: _currentIndexTab == 0
                            ? Colors.transparent
                            : Colors.grey),
                    borderRadius: BorderRadius.circular(8),
                  ),
                  child: const Tab(text: 'All'),
                ),
                Container(
                  width: double.infinity,
                  decoration: BoxDecoration(
                    border: Border.all(
                        color: _currentIndexTab == 1
```

```
                    ? Colors.transparent
                    : Colors.grey),
                borderRadius: BorderRadius.circular(8),
              ),
              child: const Tab(text: 'Follows'),
            ),
            Container(
              width: double.infinity,
              decoration: BoxDecoration(
                border: Border.all(
                    color: _currentIndexTab == 2
                    ? Colors.transparent
                    : Colors.grey),
                borderRadius: BorderRadius.circular(8),
              ),
              child: const Tab(text: 'Replies'),
            ),
          ],
        ),
      ),
    ),
  ),
  body: Padding(
    padding: const EdgeInsets.only(top: 20.0),
    child: TabBarView(
      children: [
        Column(
          children: [
            ...suggestedFollowers.map((follower) {
              return SuggestedFollowerWidget(follower: follower);
            }).toList()
          ],
        ),
        const Center(child: Text('Nothing to see here yet')),
        const Center(child: Text('Nothing to see here yet')),
      ],
    ),
  ),
 ),
);
```

```
  }
}

class SuggestedFollowerWidget extends StatelessWidget {
  const SuggestedFollowerWidget({super.key, required this.follower});

  final SuggestedFollower follower;
  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        ListTile(
          leading: CircleAvatar(
            backgroundImage: AssetImage(follower.profileImageUrl),
            backgroundColor: Colors.white,
          ),
          title: Text(follower.username),
          subtitle: Text(follower.username.toLowerCase()),
          trailing: Row(
            mainAxisSize: MainAxisSize.min,
            children: [
              Container(
                alignment: Alignment.center,
                width: 110,
                height: 35,
                decoration: BoxDecoration(
                  border: Border.all(color: Colors.grey),
                  borderRadius: BorderRadius.circular(8),
                ),
                child: follower.isFollowing
                    ? const Text(
                        'Following',
                        style: TextStyle(color: Colors.grey),
                      )
                    : const Text('Follow'),
              )
            ],
          ),
        ),
        const Divider(),
```

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



CONCLUSION

In conclusion, integrating icons, images, and custom fonts into my Flutter app enhances its visual appeal, usability, and brand identity.

MAD & PWA Lab
Journal

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget
Roll No.	14
Name	Harshita Dubey
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

Aim: To create an interactive Form using form widget.

THEORY:

In Flutter, a form is a collection of interactive widgets that allows users to input data and submit it for processing. The Form widget is a crucial component for handling user input and validation efficiently.

Form Widget:
The Form widget is the container for form elements in Flutter.
It provides a way to organize and manage form fields, as well as handle their validation and submission.

FormField Widget:
The FormField widget is a wrapper for individual form fields within a Form.
It encapsulates the logic for managing the state of the form field, handling user input, and triggering validation.

TextFormField:
The TextFormField widget is a specialized form field for handling text input.
It comes with built-in features for managing text editing, validation, and error handling.

FormState:
Each Form has an associated FormState that holds the current state of the form.
The FormState is automatically created by the framework and can be used to interact with and manage the form programmatically.

Validation:
Flutter provides a robust validation mechanism through the Form and FormField widgets.
You can define validation logic for each form field, and the framework takes care of displaying error messages when necessary.

Form Submission:
Form submission is often triggered by a button press, such as a "Submit" button.

You can use the onPressed callback of a button to trigger the form submission, and within that callback, you can access the FormState to check if the form is valid before proceeding with submission.

Handling Form Reset:
To reset the form to its initial state, you can use the reset method provided by the FormState.

CODE:

Steps:

1. Main.dart

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

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


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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(),
      home:const LoginScreen(),
    );
  }
}
```

Step 2 : In the Folder screens made previously, add new file login.dart

```
import 'package:flutter/material.dart';
import 'package:my_app/screens/home.dart';
import 'signup.dart'; // Import the file where SignupScreen is defined

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

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

class _LoginScreenState extends State<LoginScreen> {
  final emailController = TextEditingController();
  final passwordController = TextEditingController();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: SafeArea(
        child: Padding(
          padding: const EdgeInsets.all(20.0),
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Center(
                child: Image.asset(
                  'assets/thread_logo.png',
                  width: 80,
                ),
              ),
              Padding(
                padding: const EdgeInsets.only(top: 20.0),
                child: TextFormField(
                  controller: emailController,
                  decoration: const InputDecoration(
                    contentPadding: EdgeInsets.all(8),
                    hintText: 'Enter your email',
                    border: OutlineInputBorder(
                      borderSide: BorderSide(
                        color: Colors.grey,
                      ),
                    ),
                    focusedBorder: OutlineInputBorder(
                      borderSide: BorderSide(
```

```
            color: Colors.grey,
          ),
        ),
      ),
    ),
  ),
),
Padding(
  padding: const EdgeInsets.only(top: 20.0),
  child: TextFormField(
    controller: passwordController,
    decoration: const InputDecoration(
      contentPadding: EdgeInsets.all(8),
      hintText: 'Enter your password',
      border: OutlineInputBorder(
        borderSide: BorderSide(
          color: Colors.grey,
        ),
      ),
      focusedBorder: OutlineInputBorder(
        borderSide: BorderSide(
          color: Colors.grey,
        ),
      ),
    ),
  ),
),
Row(
  mainAxisAlignment: MainAxisAlignment.end,
  children: [
    TextButton(
      onPressed: () {},
      child: const Text(
        'Forgot password',
        style: TextStyle(
          color: Colors.black,
        ),
      ),
    ),
  ],
),
SizedBox(
  width: double.infinity,
  height: 42,
  child: ElevatedButton(
```

```
              onPressed: () {
                Navigator.push(context, MaterialPageRoute(builder: (context)=> const Home()));
              },
              style: ElevatedButton.styleFrom(backgroundColor: Colors.black),
              child: const Text("Login"),
            ),
          ),
          const Divider(),
          const Spacer(),
          Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              const Text("Don't have an account yet? "),
              TextButton(
                onPressed: () {
                  // Use SignupScreen() constructor to create an instance
                  Navigator.push(
                    context,
                    MaterialPageRoute(
                      builder: (context) => SignupScreen(),
                    ),
                  );
                },
                child: const Text(
                  "Sign up",
                  style: TextStyle(
                    fontWeight: FontWeight.bold,
                    color: Colors.black,
                  ),
                ),
              ),
            ],
          ),
        ],
      ),
    ),
  ),
 );
 }
}
```

Step 3: make a folder named assets and add the image threads_logo.png. Now add the path of the image file in the pubspec.yaml and then run it using flutter run.

Step 4: make a file in the screens folder named signup.dart

```
import 'package:flutter/material.dart';
import 'package:my_app/screens/login.dart';
// import 'signup_screen.dart'; // Import the file where SignupScreen is defined

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

  @override
  State<SignupScreen> createState() => _SignupScreenState();
}

class _SignupScreenState extends State<SignupScreen> {
  final emailController = TextEditingController();
  final passwordController = TextEditingController();
  final nameController = TextEditingController();
  final usernameController = TextEditingController();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: SafeArea(
        child: Padding(
          padding: const EdgeInsets.all(20.0),
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Center(
                child: Image.asset(
                  'assets/thread_logo.png',
                  width: 80,
                ),
              ),
              Padding(
                padding: const EdgeInsets.only(top: 20.0),
```

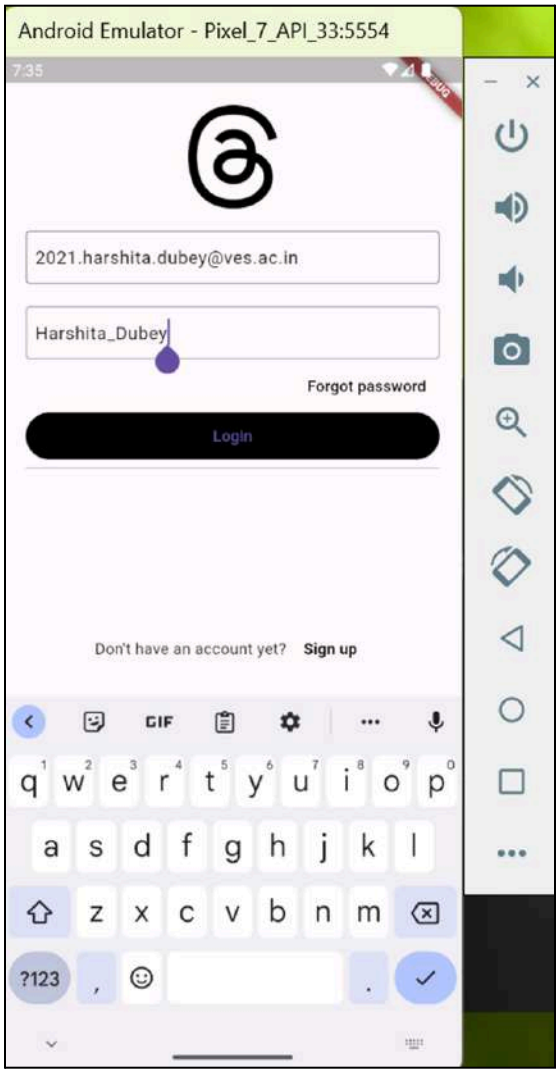
```
  child: TextFormField(
    controller: emailController,
    decoration: const InputDecoration(
      contentPadding: EdgeInsets.all(8),
      hintText: 'Enter your email',
      border: OutlineInputBorder(
        borderSide: BorderSide(
          color: Colors.grey,
        ),
      ),
      focusedBorder: OutlineInputBorder(
        borderSide: BorderSide(
          color: Colors.grey,
        ),
      ),
    ),
  ),
),
Padding(
  padding: const EdgeInsets.only(top: 20.0),
  child: TextFormField(
    controller: passwordController,
    decoration: const InputDecoration(
      contentPadding: EdgeInsets.all(8),
      hintText: 'Enter your password',
      border: OutlineInputBorder(
        borderSide: BorderSide(
          color: Colors.grey,
        ),
      ),
      focusedBorder: OutlineInputBorder(
        borderSide: BorderSide(
          color: Colors.grey,
        ),
      ),
    ),
  ),
),
          Padding(
padding: const EdgeInsets.only(top: 20.0),
child: TextFormField(
  controller: nameController,
  decoration: const InputDecoration(
    contentPadding: EdgeInsets.all(8),
```

```
        hintText: 'Enter your full name',
        border: OutlineInputBorder(
          borderSide: BorderSide(
            color: Colors.grey,
          ),
        ),
        focusedBorder: OutlineInputBorder(
          borderSide: BorderSide(
            color: Colors.grey,
          ),
        ),
      ),
    ),
  ),
  Padding(
    padding: const EdgeInsets.only(top: 20.0),
    child: TextFormField(
      controller: usernameController,
      decoration: const InputDecoration(
        contentPadding: EdgeInsets.all(8),
        hintText: 'Enter your username',
        border: OutlineInputBorder(
          borderSide: BorderSide(
            color: Colors.grey,
          ),
        ),
        focusedBorder: OutlineInputBorder(
          borderSide: BorderSide(
            color: Colors.grey,
          ),
        ),
      ),
    ),
  ),
  Padding(
    padding: const EdgeInsets.all(20.0),
    child: SizedBox(
        width: double.infinity,
        height: 42,
        child: ElevatedButton(
          onPressed: () {},
          style: ElevatedButton.styleFrom(backgroundColor: Colors.black),
          child: const Text("Sign Up"),
        ),
```

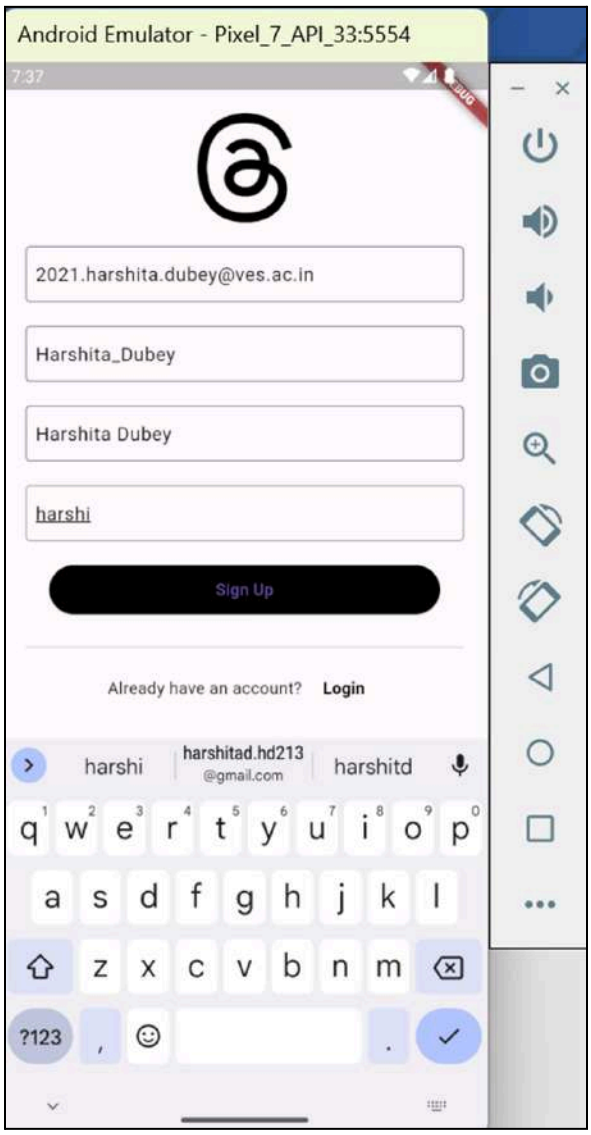
```
            ),
          ),
          const Divider(),
          const Spacer(),
          Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              const Text("Already have an account? "),
              TextButton(
                onPressed: () {
                  // Use SignupScreen() constructor to create an instance
                  Navigator.push(
                    context,
                    MaterialPageRoute(
                      builder: (context) => const LoginScreen(),
                    ),
                  );
                },
                child: const Text(
                  "Login",
                  style: TextStyle(
                    fontWeight: FontWeight.bold,
                    color: Colors.black,
                  ),
                ),
              ),
            ],
          ),
        ],
      ),
    ),
  );
 }
}
```

OUTPUT:

Login page



Signup page



CONCLUSION:

I have completed with making signup and login page from which I gained useful insights to how to use form widgets like TextFormField , Onpressed etc.
I have also used ElevatedButton.

MAD & PWA Lab
Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	14
Name	Harshita Dubey
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

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

Theory:

Navigation:

Navigation is the process of moving between different screens or views within your app. It's crucial for providing users with a seamless way to explore your app's features and content. In Flutter, navigation is managed using the Navigator class, which maintains a stack of routes representing the app's navigation history.

- Implicit navigation: This involves directly pushing new routes onto the navigation stack using Navigator.push() and popping routes off the stack using Navigator.pop(). It's suitable for simple navigation flows.
- Named routes: With named routes, you define a route table in your app's main.dart file, mapping route names to corresponding widgets. This approach makes navigation more declarative and easier to manage, especially in larger apps with multiple screens. You can then navigate to named routes using Navigator.pushNamed() and Navigator.popAndPushNamed().
- Nested navigation: Sometimes, you may need to manage navigation independently within different parts of your app, such as tab bars, bottom navigation bars, or drawers. Flutter supports nested navigation by using multiple instances of the Navigator widget. Each nested navigator maintains its own navigation stack, allowing you to handle navigation within a specific context without affecting the rest of the app.

- Routing:
  Routing involves defining the routes available in your app and handling navigation requests to those routes. Key concepts include:
  - Route management: You define a route table in your app's main.dart file, which maps route names to corresponding widget builders or constructors. This table serves as a central registry for all the screens in your app.

    - Route transitions: Flutter provides various route transition animations out-of-the-box, such as slide transitions, fade transitions, and scale transitions. You can customize these animations or create your own by subclassing the PageRoute class and implementing custom transition effects.
    - Route parameters: Routes can accept parameters or data that influence the behavior or content of the destination screen. You can pass parameters to named routes using the arguments parameter of Navigator.pushNamed(), allowing you to customize the destination screen based on the context of the navigation request.
- Gestures:
  Gestures enable users to interact with your app through touch input, making the user experience more intuitive and engaging. Key concepts include:
    - Gesture recognizers: Flutter provides a rich set of gesture recognizers that you can attach to widgets to detect and handle various types of user interactions. For example, GestureDetector allows you to detect taps, double taps, long presses, drags, and more.
    - Gesture callbacks: Gesture recognizers trigger callbacks when specific gestures are detected. For example, you can use the onTap, onDoubleTap, onLongPress, onHorizontalDragUpdate, and onVerticalDragUpdate callbacks to respond to user input and implement interactive behaviors.
    - Gesture widgets: Flutter provides specialized gesture recognizer widgets like InkWell, Draggable, Dismissible, and GestureDetector. These widgets encapsulate common gesture handling patterns and make it easier to add interactivity to your app's UI components.

Code:

Make a folder screens in the lib folder. Under screens folder make a home.dart,feed.dart,post_screen.dart,favorite_screen.dart,post_screen.dart ,sear ch.dart,profile_screen.dart

Main.dart

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

```
import 'package:flutter/material.dart';
import 'package:thread_clone_flutter/firebase_options.dart';
import 'package:thread_clone_flutter/screens/home.dart';

import 'screens/login.dart';

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(),
      home: StreamBuilder<User?>(
          stream: FirebaseAuth.instance.authStateChanges(),
          builder: (context, snapshot) {
            if (snapshot.hasData && snapshot.data != null) {
              return const Home();
            } else {
              return const LoginScreen();
            }
          }),
    );
  }
}
```

Home.dart

```
import 'package:flutter/material.dart';
import 'package:sliding_up_panel/sliding_up_panel.dart';
import 'package:thread_clone_flutter/screens/post_screen.dart';
import 'package:thread_clone_flutter/screens/profile_screen.dart';

import 'favorite_screen.dart';
import 'feed.dart';
```

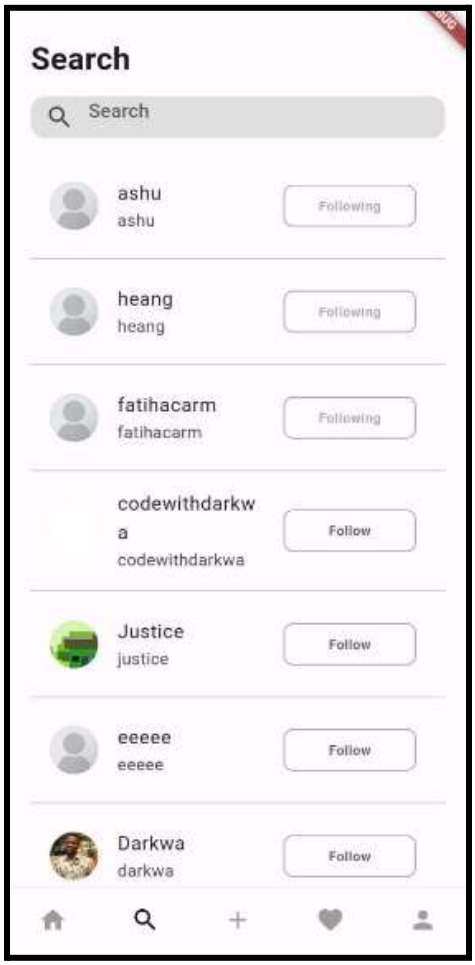
```
import 'search.dart';

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

  @override
  State<Home> createState() => _HomeState();
}

class _HomeState extends State<Home> {
  int selectedIndex = 0;

  List<Widget> pages = [];
  PanelController panelController = PanelController();

  @override
  void initState() {
    pages = [
      const FeedScreen(),
      const SearchScreen(),
      PostScreen(panelController: panelController),
      const FavoriteScreen(),
      const ProfileScreen(),
    ];
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: SlidingUpPanel(
        controller: panelController,
        minHeight: 0,
        maxHeight: MediaQuery.of(context).size.height * 0.8,
        borderRadius: const BorderRadius.only(
          topLeft: Radius.circular(25),
          topRight: Radius.circular(25),
        ),
        panelBuilder: (ScrollController sc) {
          return PostScreen(panelController: panelController);
        },
        body: pages[selectedIndex],
      ),
      bottomNavigationBar: BottomNavigationBar(
```

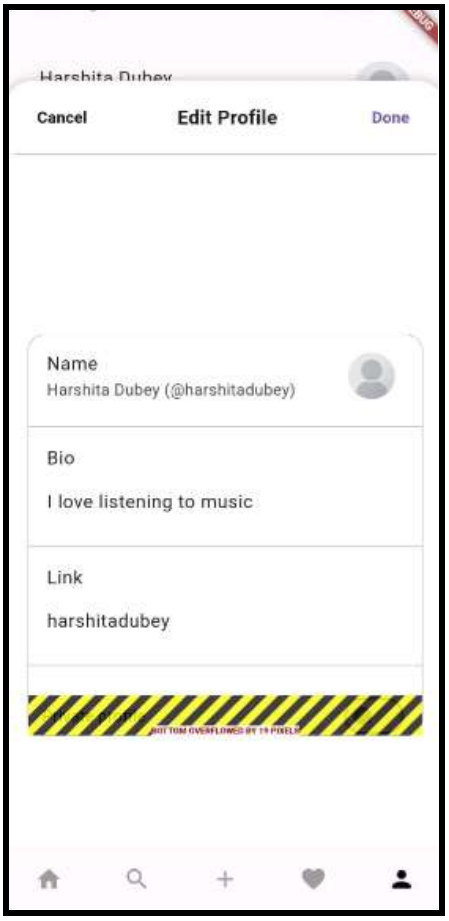
```
      currentIndex: selectedIndex,
      selectedItemColor: Colors.black,
      unselectedItemColor: Colors.grey,
      showSelectedLabels: false,
      showUnselectedLabels: false,
      onTap: (index) {
        if (index == 2) {
          panelController.isPanelOpen
              ? panelController.close()
              : panelController.open();
        } else {
          panelController.close();
          setState(() {
            selectedIndex = index;
          });
        }
      },
      type: BottomNavigationBarType.fixed,
      items: const [
        BottomNavigationBarItem(icon: Icon(Icons.home), label: 'Feed'),
        BottomNavigationBarItem(icon: Icon(Icons.search), label: 'search'),
        BottomNavigationBarItem(icon: Icon(Icons.add), label: 'post'),
        BottomNavigationBarItem(
            icon: Icon(Icons.favorite), label: 'favorite'),
        BottomNavigationBarItem(icon: Icon(Icons.person), label: 'profile'),
      ],
    ),
  );
 }
}
```

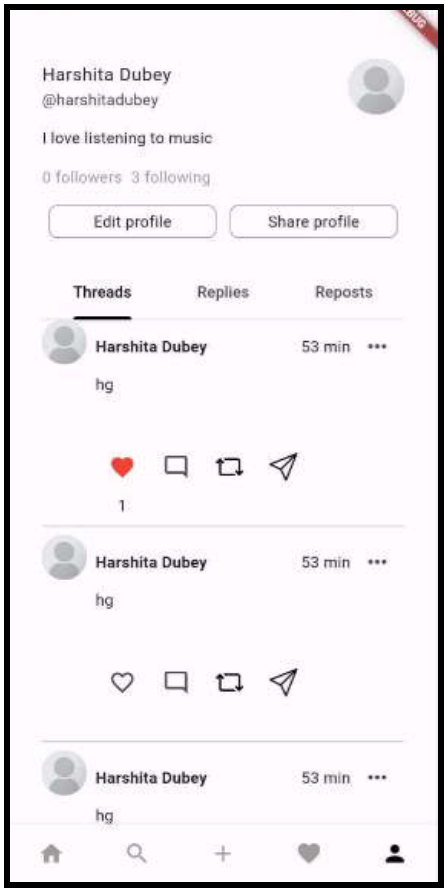
feed.dart

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

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

@override
```

```
State createState() => _FeedScreenState();

}
class _FeedScreenState extends State {

@override

Widget build(BuildContext context) {
 return const Scaffold( body: Center( child: Text("Feed Screen"),
),
);
 }
}
```




Harshita Dubey
Cancel
Edit Profile
Done
Name
Harshita Dubey (@harshitadubey)
Bio
I love listening to music
Link
harshitadubey



CONCLUSION

In conclusion navigation, routing, and gestures are fundamental components of building a user-friendly and interactive Flutter app. By mastering these concepts, I have created seamless and engaging user experience that facilitates effortless navigation, smooth transitions between screens, and intuitive touch interactions.

MAD & PWA Lab

Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	14
Name	Harshita
Class	D15A
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

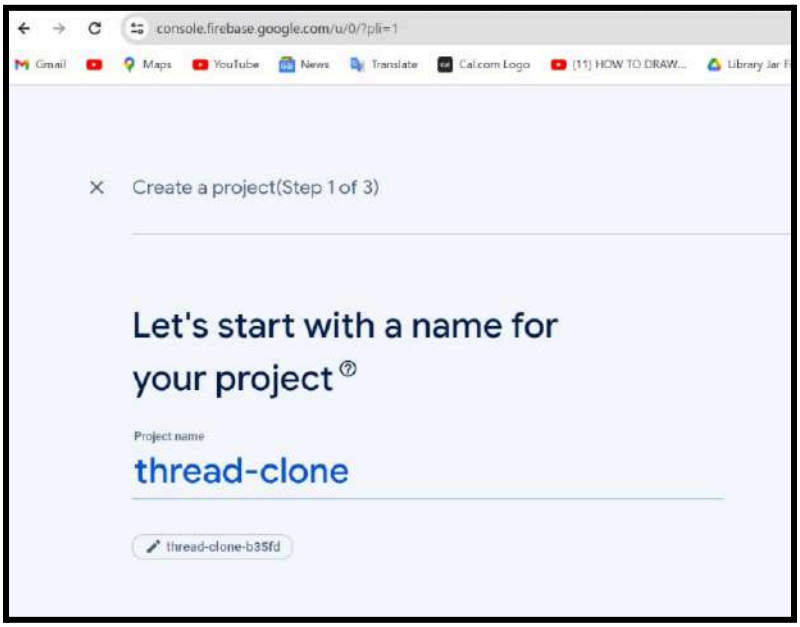
Experiment 6

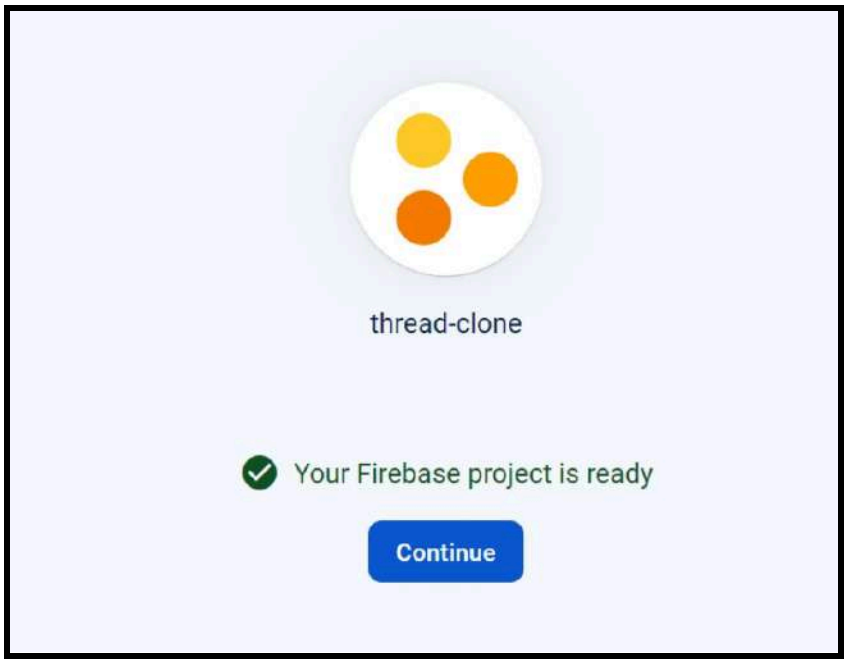
Aim: To Connect Flutter UI with fireBase database

Theory:

- Create a Firebase Project:
  - Go to the Firebase Console (https://console.firebase.google.com/) and create a new project.

  Add Your App to Firebase:
  - After creating your Firebase project, click on the "Add app" button.
  - Choose the platform (iOS or Android) for which you want to add the app.

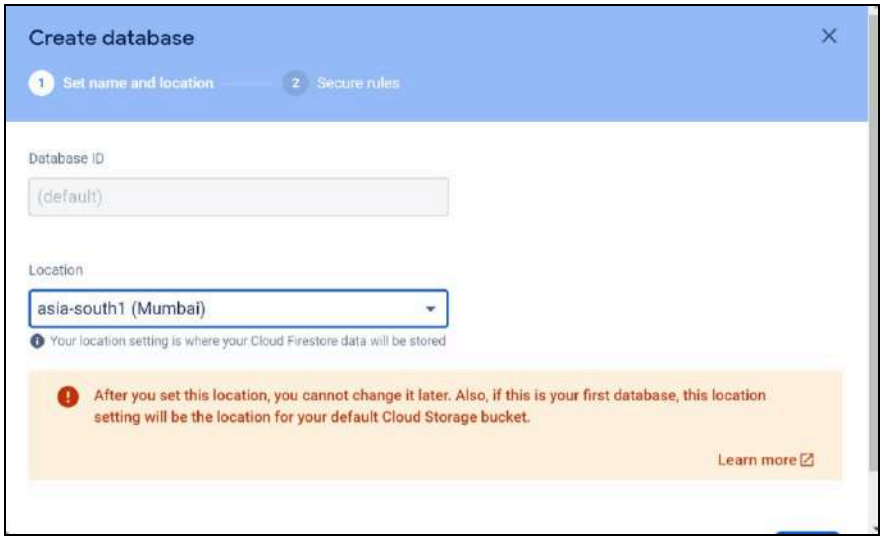




Go to the Terminal:

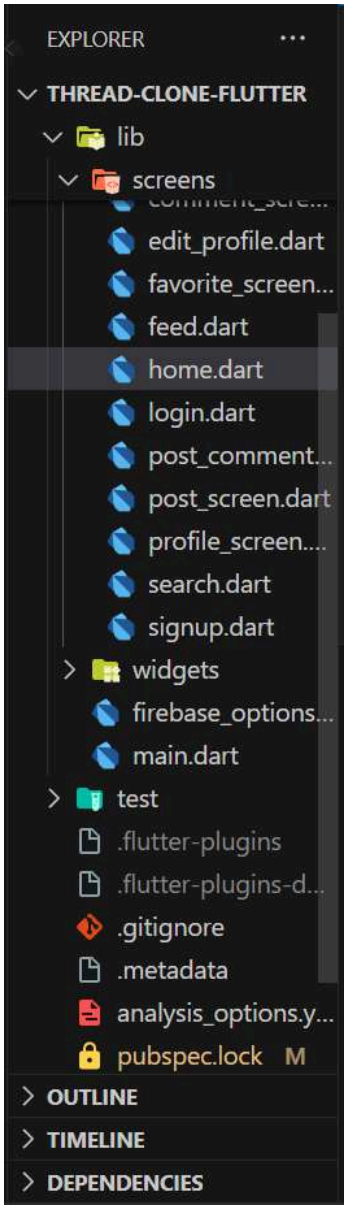
```
Visit this URL on this device to log in:
https://accounts.google.com/o/oauth2/auth?client_id=563584335869-fgrhgmd47bqnekij5i8b5pr03ho849e6.apps.googleusercontent.c
ail%20openid%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcloudplatformprojects.readonly%20https%3A%2F%2Fwww.googleapis.com
irebase%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcloud-platform&response_type=code&state=177746748&redirect_uri=http%3A
host%3A9005

Waiting for authentication...

+  Success! Logged in as harshitad.hd213@gmail.com
PS E:\Flutter Projects\my_app> firebase login
PS E:\Flutter Projects\my_app> flutterfire configure
i Found 1 Firebase projects.
✓ Select a Firebase project to configure your Flutter application with · thread-clone-b35fd (thread-clone)
✓ Which platforms should your configuration support (use arrow keys & space to select)? · android, ios, macos, web
i Firebase android app com.example.my_app is not registered on Firebase project thread-clone-b35fd.
i Registered a new Firebase android app on Firebase project thread-clone-b35fd.
i Firebase ios app com.example.myApp is not registered on Firebase project thread-clone-b35fd.
i Registered a new Firebase ios app on Firebase project thread-clone-b35fd.
```

To create Database

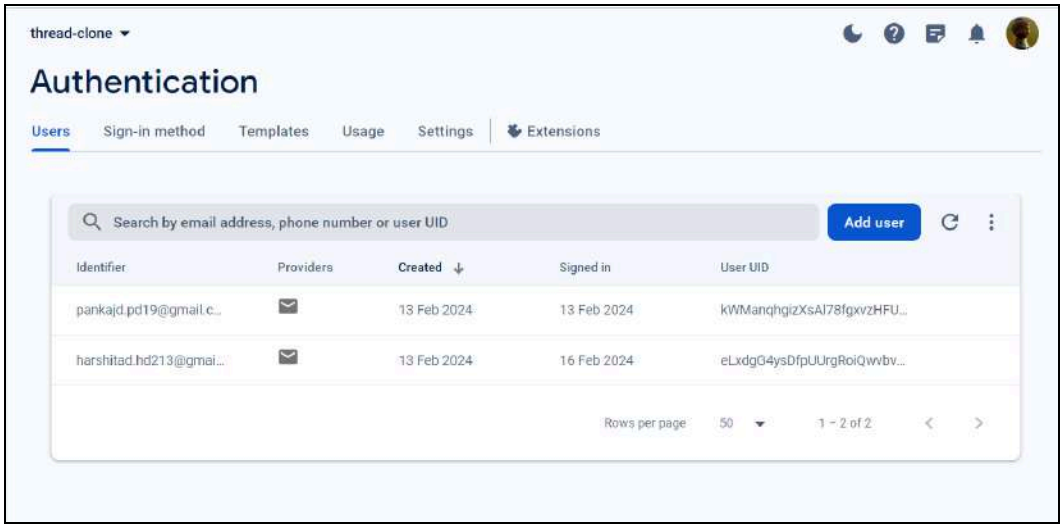

Create database
1 Set name and location
2 Secure rules
Database ID
(default)
Location
asia-south1 (Mumbai)
Your location setting is where your Cloud Firestore data will be stored
After you set this location, you cannot change it later. Also, if this is your first database, this location setting will be the location for your default Cloud Storage bucket.
Learn more

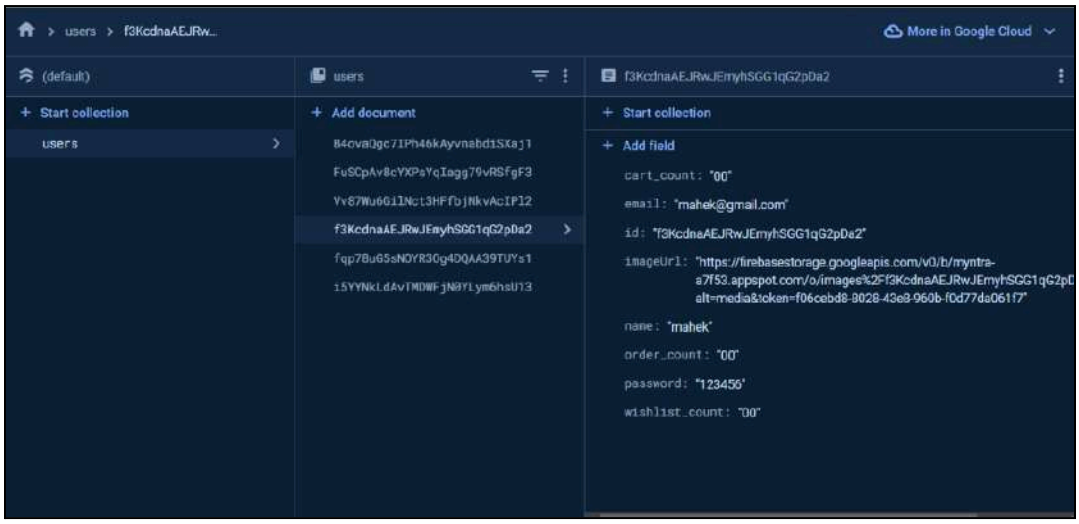

Set name and location
2 Secure rules
After you define your data structure, you will need to write rules to secure your data.
Learn more
Start in production mode
Your data is private by default. Client read/write access will only be granted as specified by your security rules.
Start in Test mode
Your data is open by default to enable quick setup. However, you must update your security rules within 30 days to enable long-term client read/write access.
rules_version = '2';
service cloud.firestore {
match /databases/{database}/documents {
match /{document=**} {
allow read, write: if
request.time < timestamp.date(2024, 3, 13);
}
}
}
The default security rules for test mode allow anyone with your database reference to view, edit and delete all data in your database for the next 30 days
Cancel
Create

Folder Structure:



Authentication of User





CONCLUSION

I have successfully implemented my application with the firebase

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.	14
Name	Harshita Dubey
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	15

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.

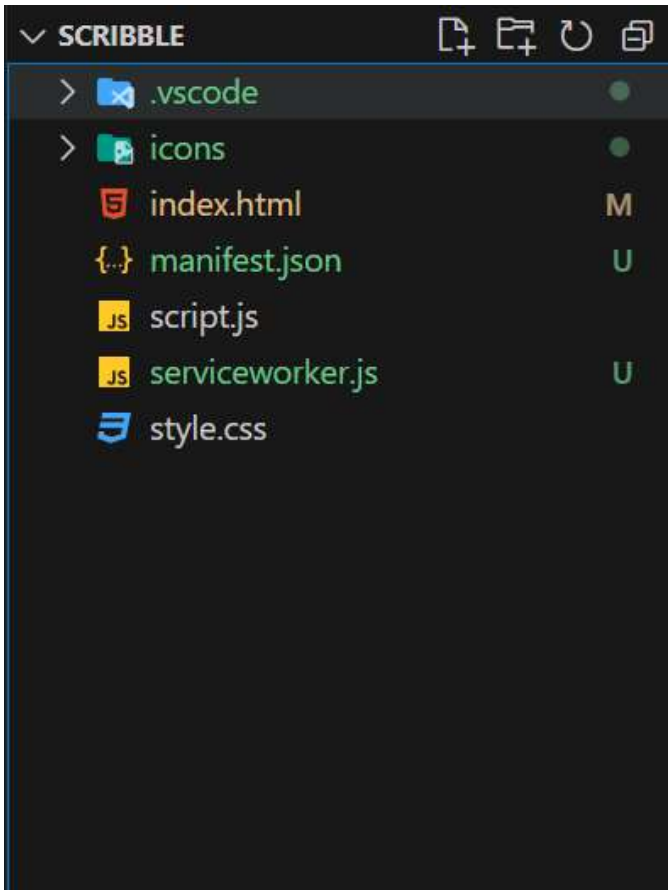
7. Lower Development Cost

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.

IMPLEMENTATION

STEP 1: Open any web application through any Text Editor eg:VSCODE.

STEP 2: Add few files to this. Folder structure of my application is:



CODE:

manifest.json

```
{
   "name":"Scribble",
   "short_name":"PWA",
   "start_url":"index.html",
   "display":"standalone",
   "background_color":"#5900b3",
   "theme_color":"black",
   "scope": ".",
   "description":"This is a PWA tutorial.",
   "icons":[
   {
   "src":"icons/ma.png",
   "sizes":"192x192",
   "type":"image/png"
   },
   {
   "src":"icons/twitter.png",
   "sizes":"512x512",
   "type":"image/png"
   }
 ]
}
```

serviceworker.js

```
var staticCacheName = "pwa";

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

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

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

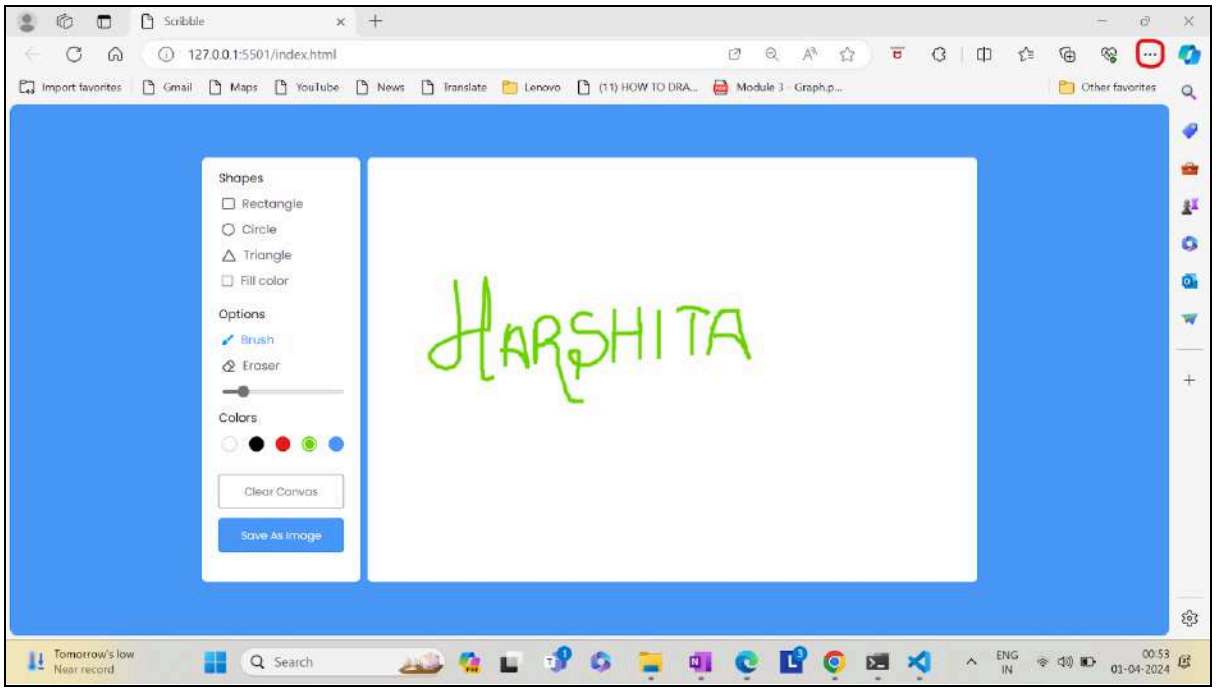
index.html

```
<!DOCTYPE html>

<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Scribble</title>
    <link rel="stylesheet" href="style.css">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script src="script.js" defer></script>
    <!-- Manifest File link -->
<link rel="manifest"
href="manifest.json">
  </head>
  <body>
    <div class="container">
      <section class="tools-board">
        <div class="row">
          <label class="title">Shapes</label>
          <ul class="options">
            <li class="option tool" id="rectangle">
              <img src="icons/rectangle.svg" alt="">
              <span>Rectangle</span>
            </li>
            <li class="option tool" id="circle">
              <img src="icons/circle.svg" alt="">
              <span>Circle</span>
            </li>
            <li class="option tool" id="triangle">
              <img src="icons/triangle.svg" alt="">
              <span>Triangle</span>
            </li>
```

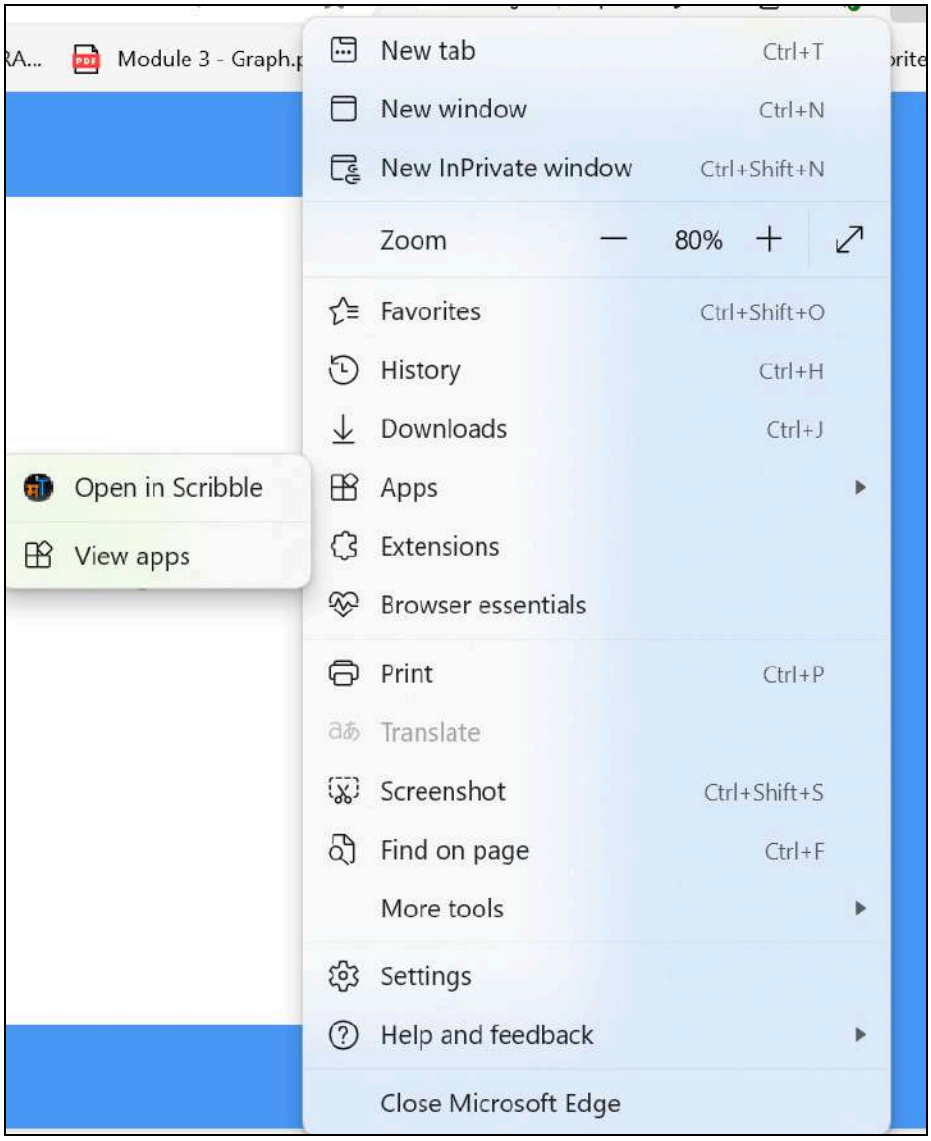
```
        <li class="option">
          <input type="checkbox" id="fill-color">
          <label for="fill-color">Fill color</label>
        </li>
      </ul>
    </div>
    <div class="row">
      <label class="title">Options</label>
      <ul class="options">
        <li class="option active tool" id="brush">
          <img src="icons/brush.svg" alt="">
          <span>Brush</span>
        </li>
        <li class="option tool" id="eraser">
          <img src="icons/eraser.svg" alt="">
          <span>Eraser</span>
        </li>
        <li class="option">
          <input type="range" id="size-slider" min="1" max="30" value="5">
        </li>
      </ul>
    </div>
    <div class="row colors">
      <label class="title">Colors</label>
      <ul class="options">
        <li class="option"></li>
        <li class="option selected"></li>
        <li class="option"></li>
        <li class="option"></li>
        <li class="option">
          <input type="color" id="color-picker" value="#4A98F7">
        </li>
      </ul>
    </div>
    <div class="row buttons">
      <button class="clear-canvas">Clear Canvas</button>
      <button class="save-img">Save As Image</button>
    </div>
  </section>
  <section class="drawing-board">
```

```
      <canvas></canvas>
    </section>
  </div>
  <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>

 </body>
</html>
```

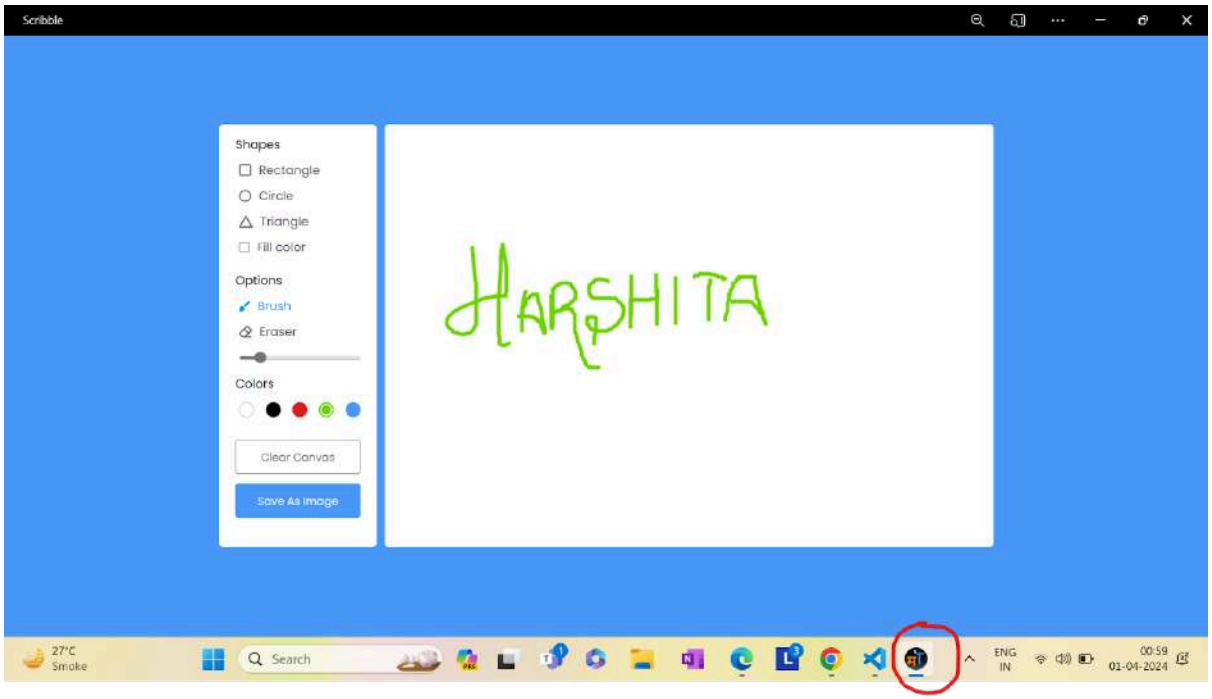
STEP 4 : Open Live Server and http://127.0.0.1:5501/index.html will be opened



STEP 5: Click on 3 dots on the top right corner of the browser → go to Apps → Click on Install Scribble.


RA...
Module 3 - Graph.p
rite
New tab Ctrl+T
New window Ctrl+N
New InPrivate window Ctrl+Shift+N
Zoom 80%
Favorites Ctrl+Shift+O
History Ctrl+H
Downloads Ctrl+J
Apps
Extensions
Browser essentials
Print Ctrl+P
Translate
Screenshot Ctrl+Shift+S
Find on page Ctrl+F
More tools
Settings
Help and feedback
Close Microsoft Edge
Open in Scribble
View apps

App Icon will appear at the bottom:



CONCLUSION:

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

MAD & PWA Lab
Journal

Experiment No.	08
Experiment Title.	To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA
Roll No.	14
Name	Harshita Dubey
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

NAME: HARSHITA DUBEY ROLL NO.:14
SUBJECT: MAD LAB CLASS:D15A/BATCH A

EXP 8

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

THEORY:

Service Worker

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

Things to note about Service Worker:

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

What can we do with Service Workers?

- You can dominate **Network Traffic**

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

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

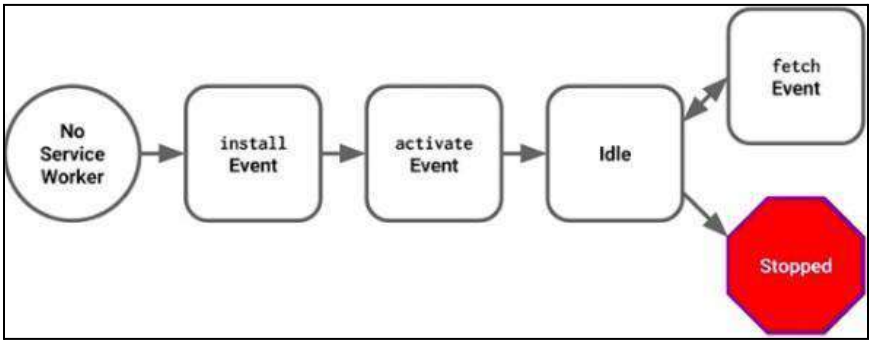
- You can manage **Push Notifications**
  You can manage push notifications with Service Worker and show any information message to the user.
- You can **Continue**
  Although Internet connection is broken, you can start any process with Background Sync of Service Worker.

What can't we do with Service Workers?

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

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

Service Worker Cycle



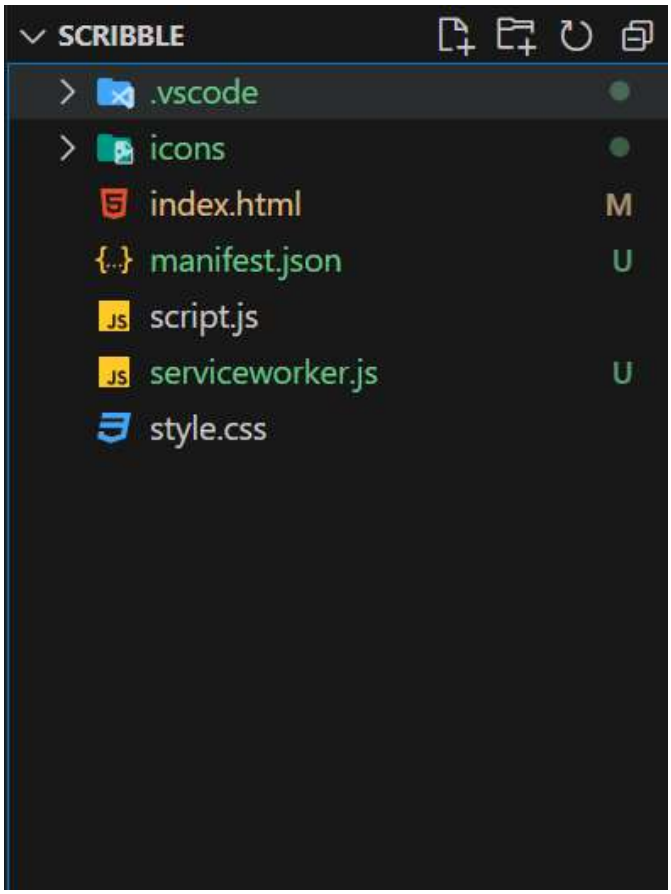
A service worker goes through three steps in its life cycle:

- Registration
- Installation
- Activation

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.

My Folder Structure



CODE:

index.html

```
<!DOCTYPE html>
```

```
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Scribble</title>
    <link rel="stylesheet" href="style.css">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script src="script.js" defer></script>
    <!-- Manifest File link -->
<link rel="manifest"
href="manifest.json">
  </head>
  <body>
    <div class="container">
      <section class="tools-board">
        <div class="row">
          <label class="title">Shapes</label>
          <ul class="options">
            <li class="option tool" id="rectangle">
              <img src="icons/rectangle.svg" alt="">
              <span>Rectangle</span>
            </li>
            <li class="option tool" id="circle">
              <img src="icons/circle.svg" alt="">
              <span>Circle</span>
            </li>
            <li class="option tool" id="triangle">
              <img src="icons/triangle.svg" alt="">
              <span>Triangle</span>
            </li>
            <li class="option">
              <input type="checkbox" id="fill-color">
              <label for="fill-color">Fill color</label>
            </li>
          </ul>
        </div>
        <div class="row">
          <label class="title">Options</label>
          <ul class="options">
            <li class="option active tool" id="brush">
```

```
          <img src="icons/brush.svg" alt="">
          <span>Brush</span>
        </li>
        <li class="option tool" id="eraser">
          <img src="icons/eraser.svg" alt="">
          <span>Eraser</span>
        </li>
        <li class="option">
          <input type="range" id="size-slider" min="1" max="30" value="5">
        </li>
      </ul>
    </div>
    <div class="row colors">
      <label class="title">Colors</label>
      <ul class="options">
        <li class="option"></li>
        <li class="option selected"></li>
        <li class="option"></li>
        <li class="option"></li>
        <li class="option">
          <input type="color" id="color-picker" value="#4A98F7">
        </li>
      </ul>
    </div>
    <div class="row buttons">
      <button class="clear-canvas">Clear Canvas</button>
      <button class="save-img">Save As Image</button>
    </div>
  </section>
  <section class="drawing-board">
    <canvas></canvas>
  </section>
</div>
<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>

 </body>
</html>
```

Installation

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

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

```
var staticCacheName = "pwa";
self.addEventListener("install", function (e) {
e.waitUntil(
caches.open(staticCacheName).then(function (cache) {
```

```
return cache.addAll(["/"]);
})
);
});
```

-Step 1 of service worker installation.

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).

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

-Step 2 of service worker activation.

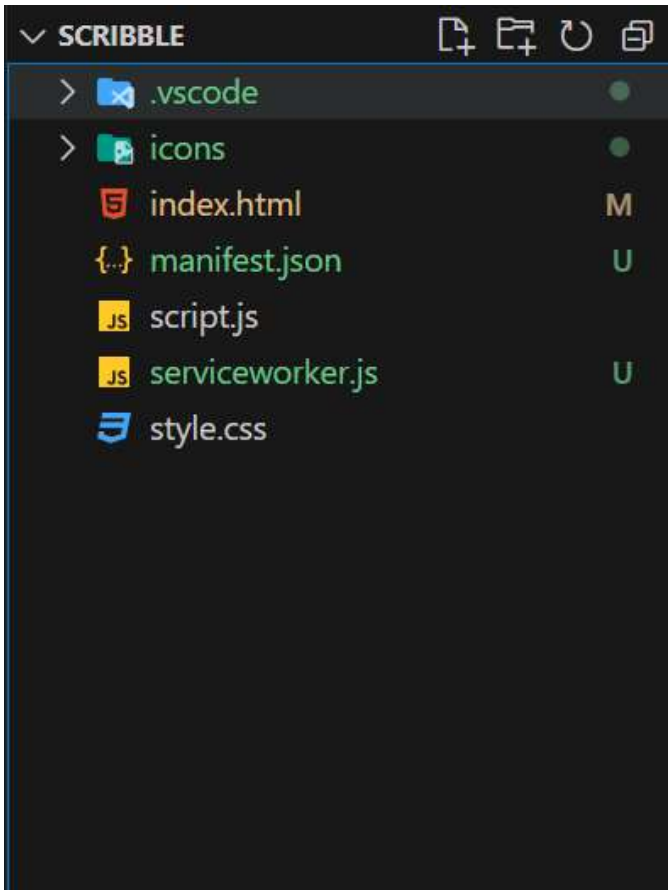
Thus the **serviceworker.js** will look like this

```
var staticCacheName = "pwa";
self.addEventListener("install", function (e) {
e.waitUntil(
caches.open(staticCacheName).then(function (cache) {
return cache.addAll(["/"]);
})
);
});
self.addEventListener('activate', event => {
event.waitUntil(
caches.keys().then(cacheNames => {
return Promise.all(
cacheNames.filter(name => {
return name !== staticCacheName;
}).map(name => {
return caches.delete(name);
})
);
})
);
});
self.addEventListener("fetch", function (event) {
console.log(event.request.url);
event.respondWith(
caches.match(event.request).then(function (response) {
return response || fetch(event.request);
})
);
});
```

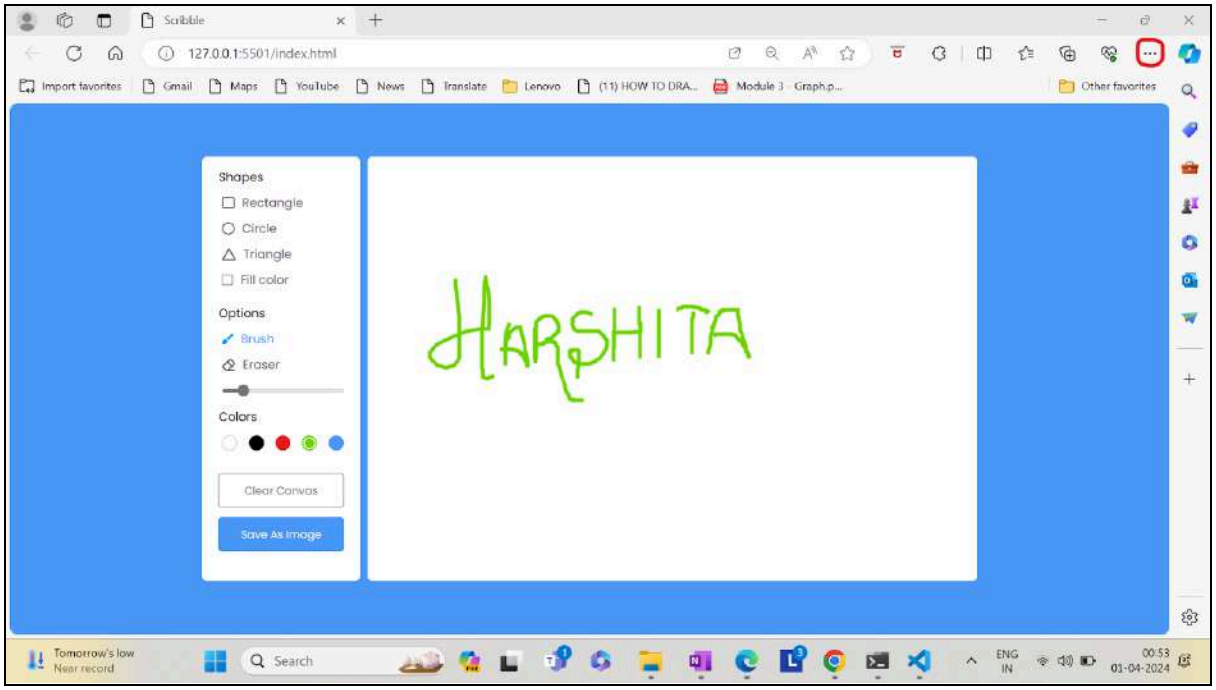
IMPLEMENTATION

STEP 1: Open any web application through any Text Editor eg:VSCODE.

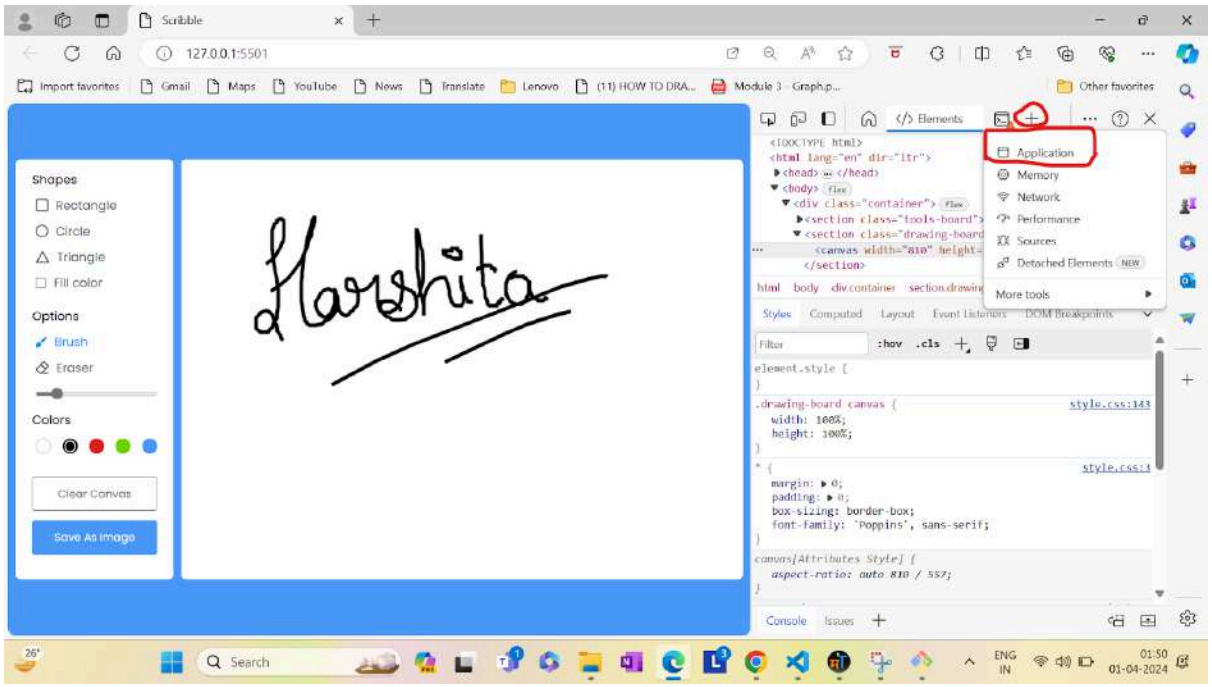
STEP 2: Add few files to this. Folder structure of my application is:

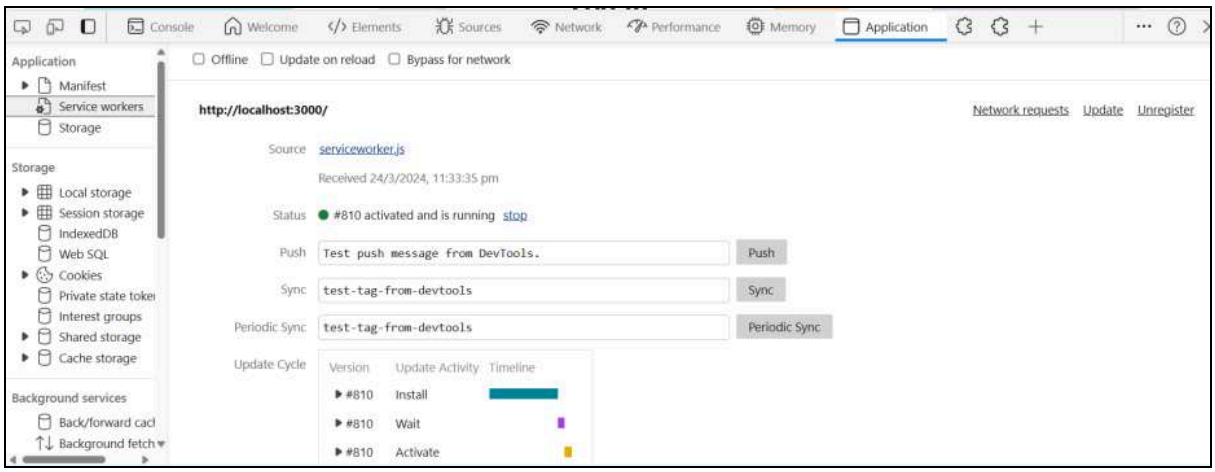


STEP 4 : Open Live Server and http://127.0.0.1:5501/index.html will be opened



STEP 4: Right Click on the browser and click on inspect —-> Plus icon —-> Application





Conclusion:

To enhance the user experience and resilience of a Progressive Web Application (PWA), a service worker (serviceworker.js) was implemented and registered. This service worker intercepts network requests, caches critical resources, and facilitates offline functionality and improved performance. Upon completion of the installation and activation process, the PWA gains features such as offline access and accelerated page loads. This strategic integration of a service worker not only elevates user experience but also fortifies the application against network disruptions, ensuring seamless functionality even in challenging connectivity conditions.

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.	14
Name	Harshita Dubey
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

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

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

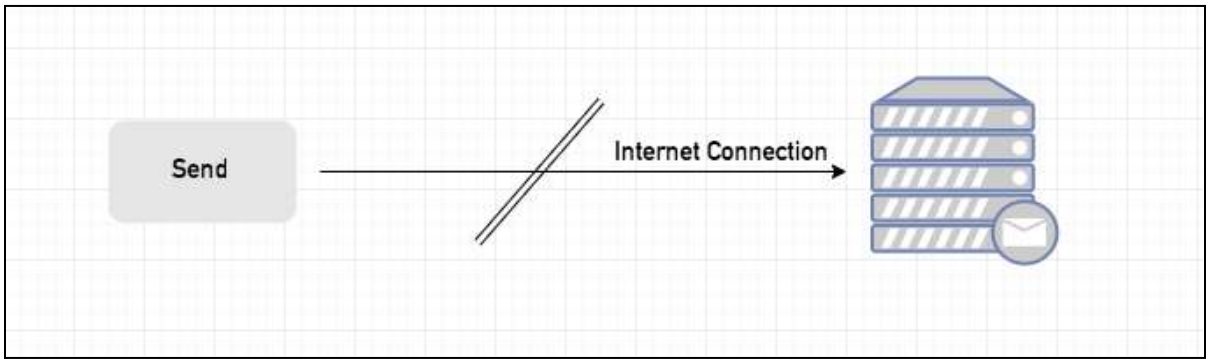
returned. But if this process fails, we check whether the request has been cached before or not. If a cache exists, it is returned to the page, but if not, this is up to you. You can return dummy content or information messages to the page.

Sync Event

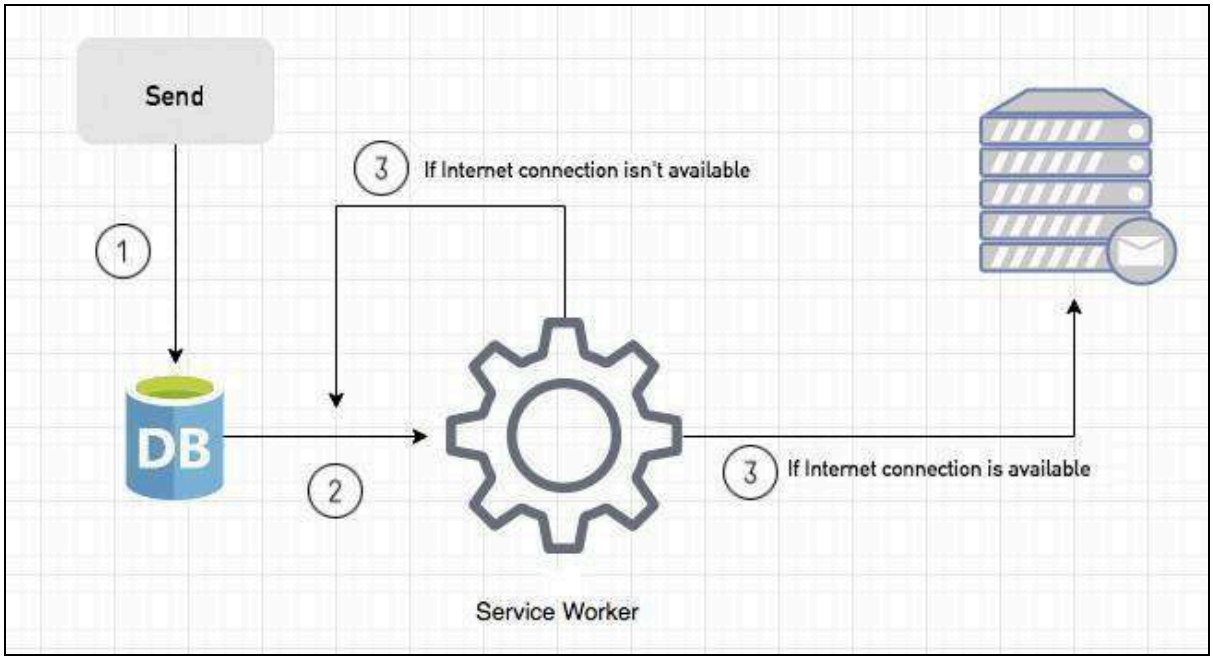
Background Sync is a Web API that is used to delay a process until the Internet connection is stable. We can adapt this definition to the real world; there is an e-mail client application that works on the browser and we want to send an email with this tool. Internet connection is broken while we are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



Here, you can create any scenario for yourself. 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.

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

Event Listener for Background Sync Registration

```
document.querySelector("button").addEventListener("click", async () => {
    var swRegistration = await navigator.serviceWorker.register("sw.js");
    swRegistration.sync.register("helloSync").then(function () {
        console.log("helloSync success [main.js]");
    });
});
```

Event Listener for sw.js

```
self.addEventListener('sync', event => {
    if (event.tag == 'helloSync') {
        console.log("helloSync [sw.js]");
    }
});
```

Push Event

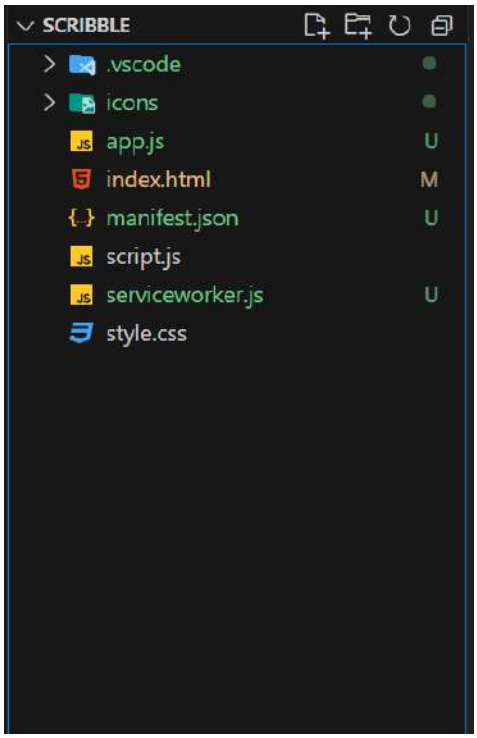
This is the event that handles push notifications that are received from the server. You can apply any method with received data.

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 :

Folder Structure



serviceworker.js

```
self.addEventListener("install", function (event) {
 event.waitUntil(preLoad());
 });
 self.addEventListener("fetch", function (event) {
 event.respondWith(
 checkResponse(event.request).catch(function () {
 console.log("Fetch from cache successful!");
 return returnFromCache(event.request);
 })
 );
 console.log("Fetch successful!");
 event.waitUntil(addToCache(event.requesst));
 });
 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("Scribble", {
 body: data.message,
 });
 }
 } catch (error) {
 console.error("Error parsing push data:", error);
 }
 }
 });
 var preLoad = function () {
 return caches.open("offline").then(function (cache) {
 // caching index and important routes
 return cache.addAll([
 '/',
```

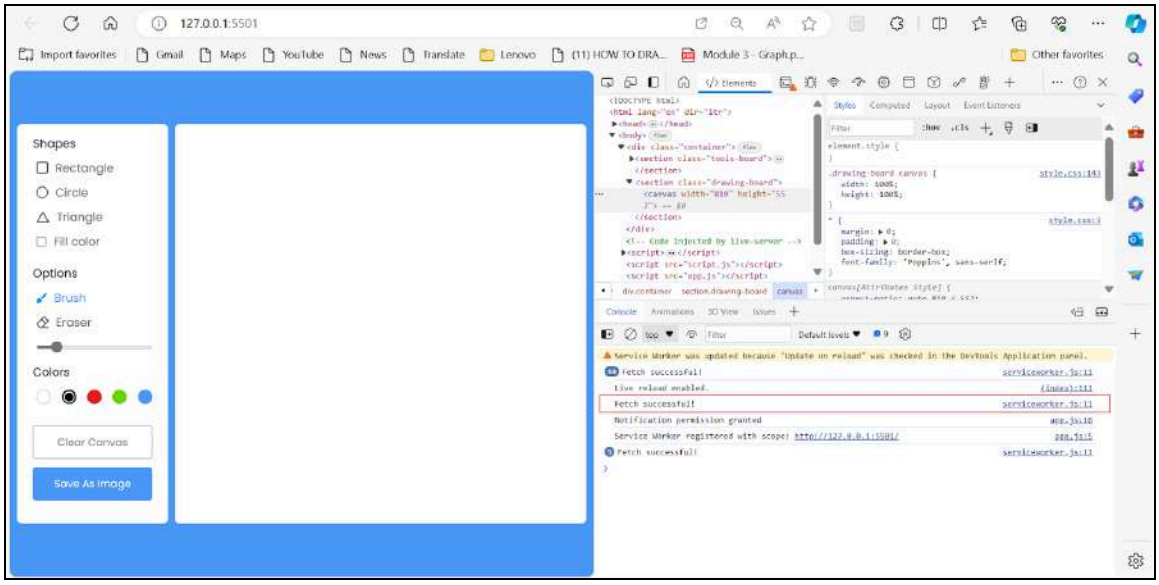
```
'/index.html',
'/style.css',
'/script.js'

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

```
});
};
```

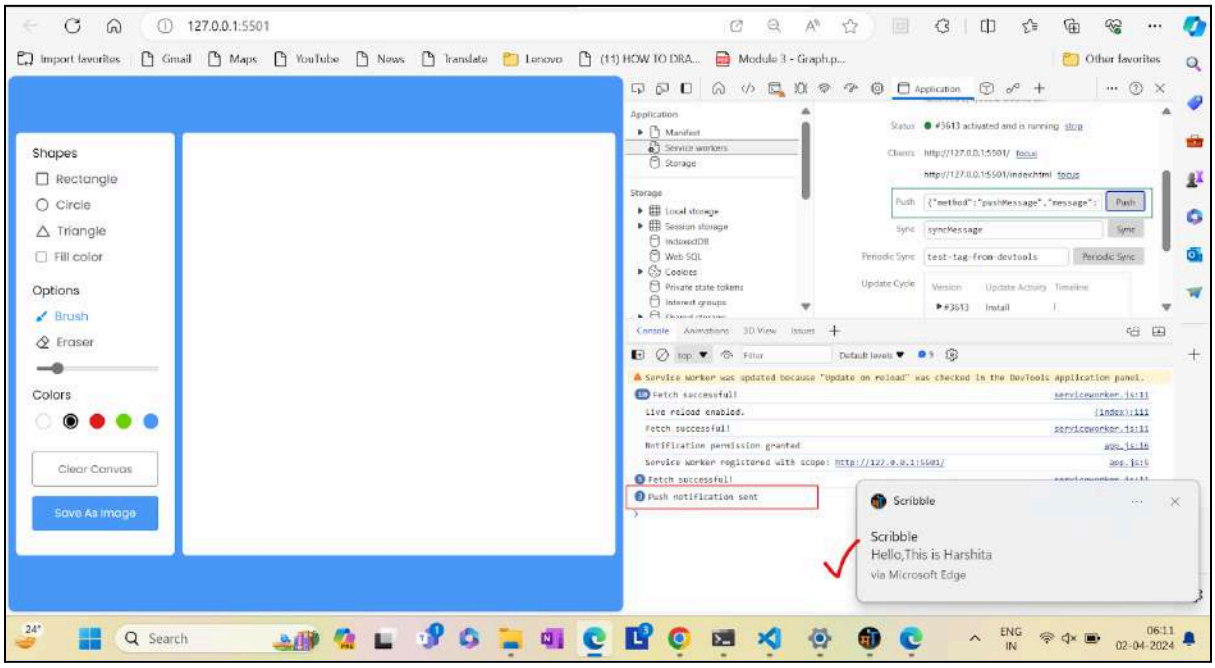
app.js

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

  if ('Notification' in window) {
  Notification.requestPermission().then(function (result) {
  if (result === 'granted') {
  console.log('Notification permission granted');
  } else {
  console.warn('Notification permission denied');
  }
  });
  }
```

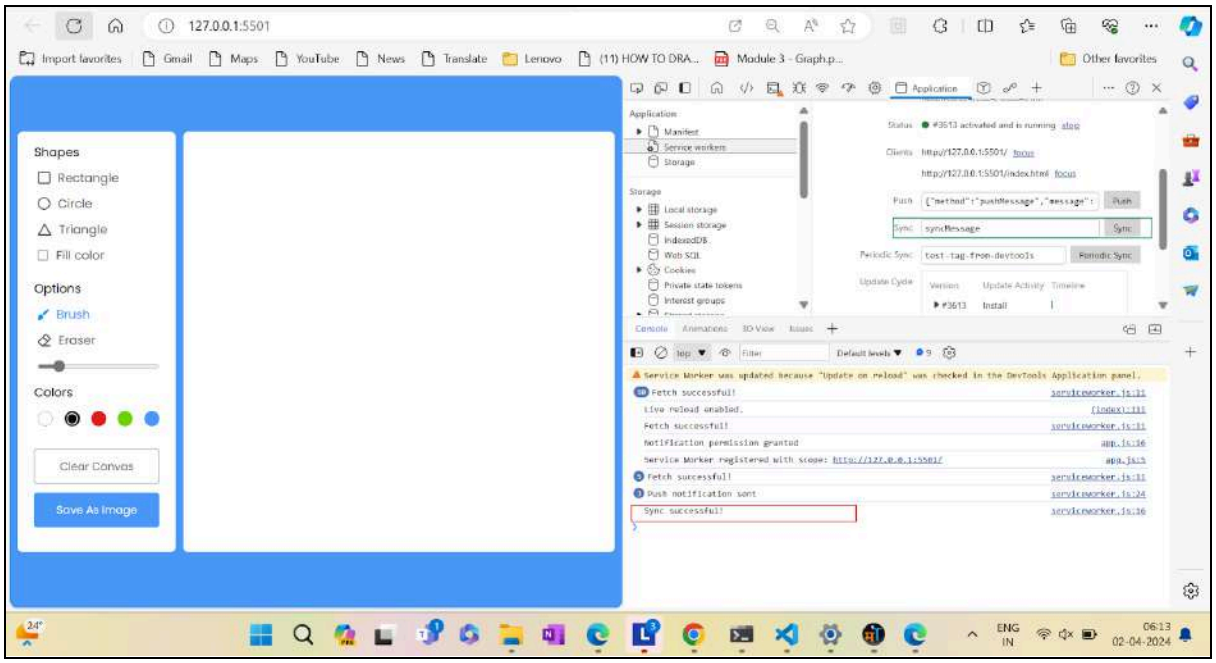
1). Fetch Event



2). Push Event



3) Sync Event



Conclusion:

In this experiment, we have successfully implemented service worker events like fetch, sync and push for my Scribble E-commerce PWA and found out output for above implementation.

MAD & PWA Lab

Journal

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	14
Name	Harshita Dubey
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

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.

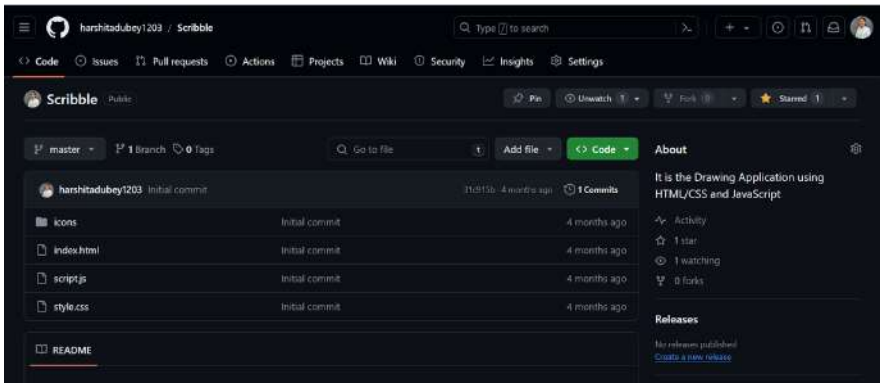
Cons

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:

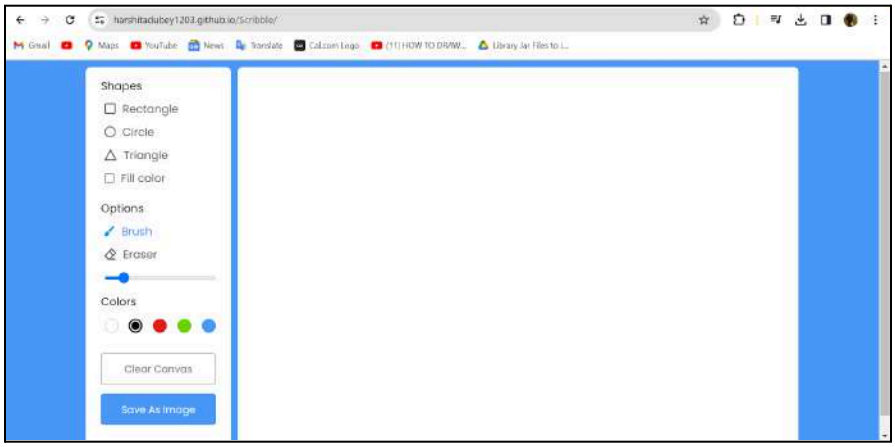
harshitadubey1203/Scribble

Github Screenshot:



HOSTED LINK

Scribble(github.io)



CONCLUSION

Hosted Scribble webpage on github page.

MAD & PWA Lab
Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	14
Name	Harshita Dubey
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	15

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

Theory :

Google Lighthouse :

Google Lighthouse is a tool that lets you audit your web application based on a number of parameters including (but not limited to) performance, based on a number of metrics, mobile compatibility, Progressive Web App (PWA) implementations, etc. All you have to do is run it on a page or pass it a URL, sit back for a couple of minutes and get a very elaborate report, not much short of one that a professional auditor would have compiled in about a week.
The best part is that you have to set up almost nothing to get started. Let's begin by looking at some of the top features and audit criteria used by Lighthouse.

Key Features and Audit Metrics

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

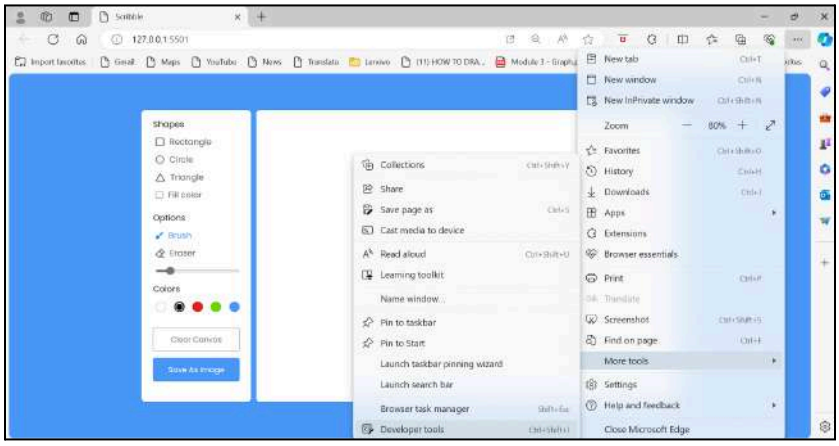
1. Performance: This score is an aggregation of how the page fared in aspects such as (but not limited to) loading speed, time taken for loading for basic frame(s), displaying meaningful content to the user, etc. To a layman, this score is indicative of how decently the site performs, with a score of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

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

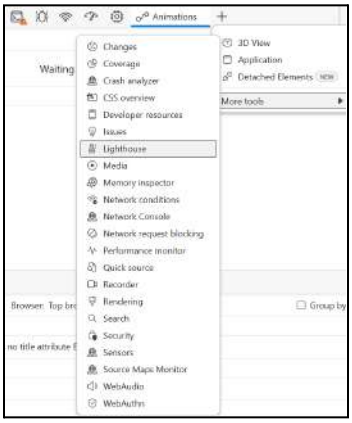
4. Best Practices: As any developer would know, there are a number of practices that have been deemed 'best' based on empirical data. This metric is an aggregation of many such points, including but not limited to:Use of HTTPS

Avoiding the use of deprecated code elements like tags, directives, libraries, etc.
Password input with paste-into disabled
Geo-Location and cookie usage alerts on load, etc.

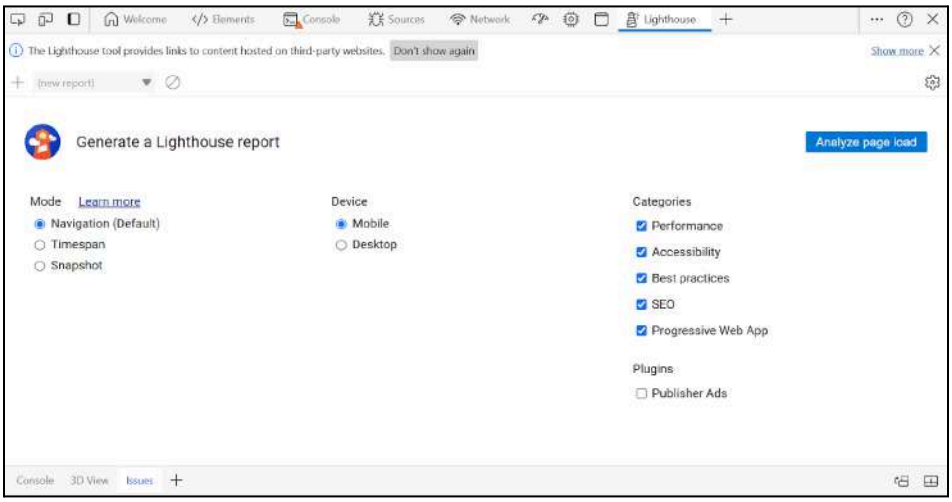
Right Click on the Browser —-> More tools —-> Developer tools

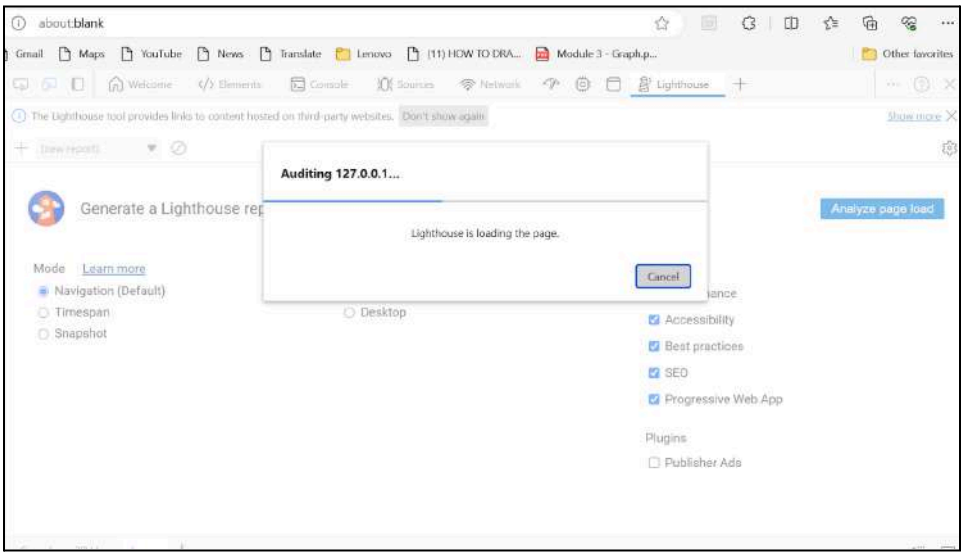


Click on Plus icon —-> Google Lighthouse

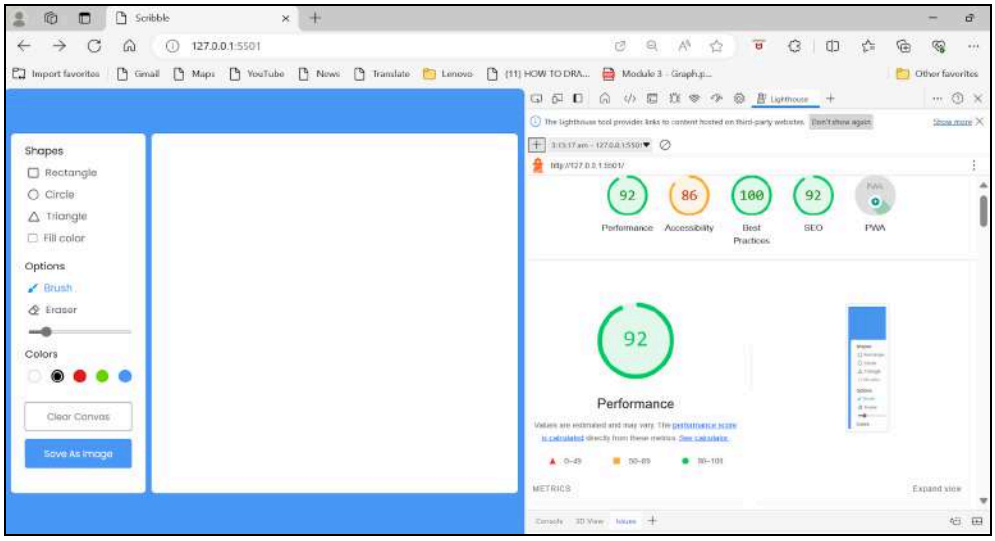


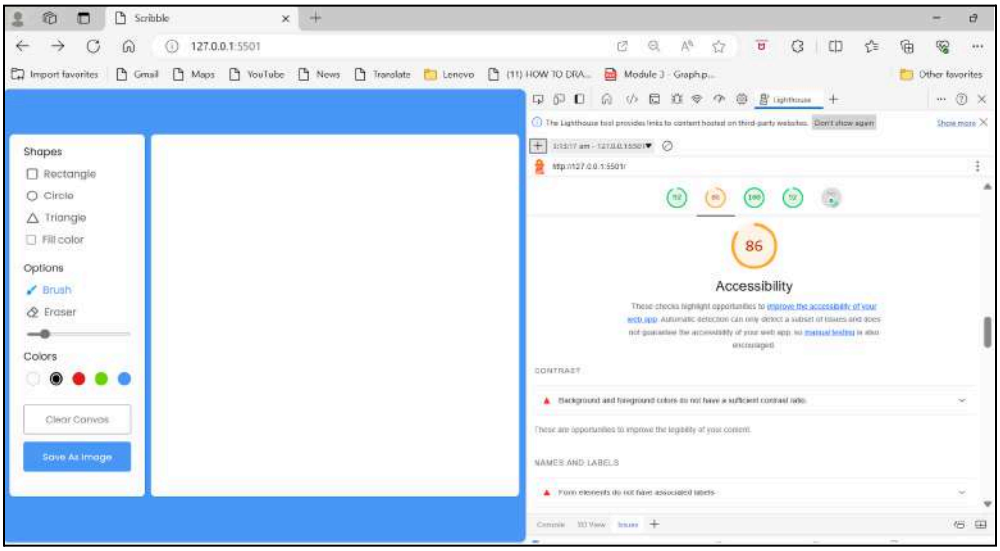
Click on Analyze page load



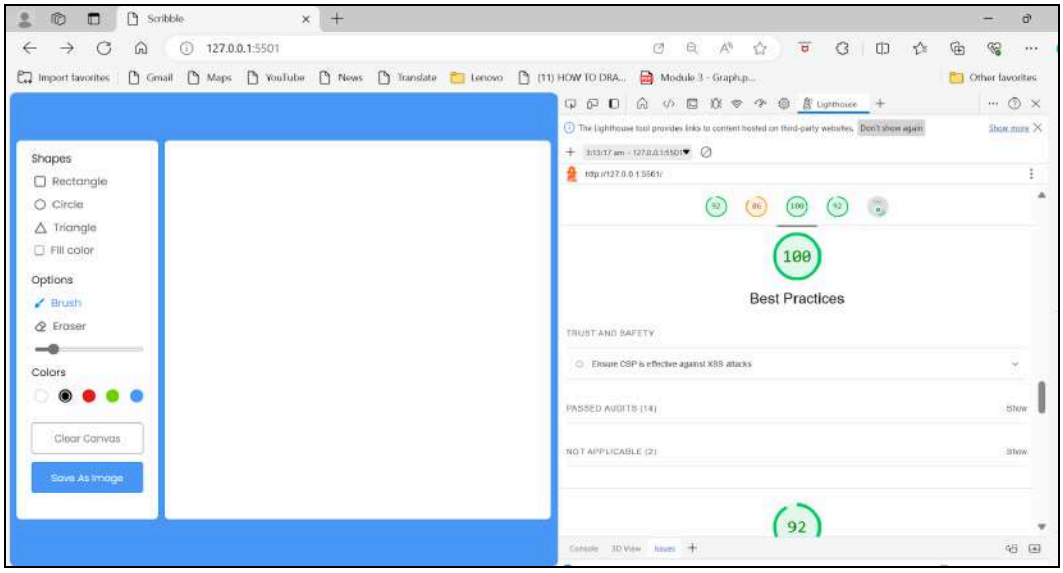


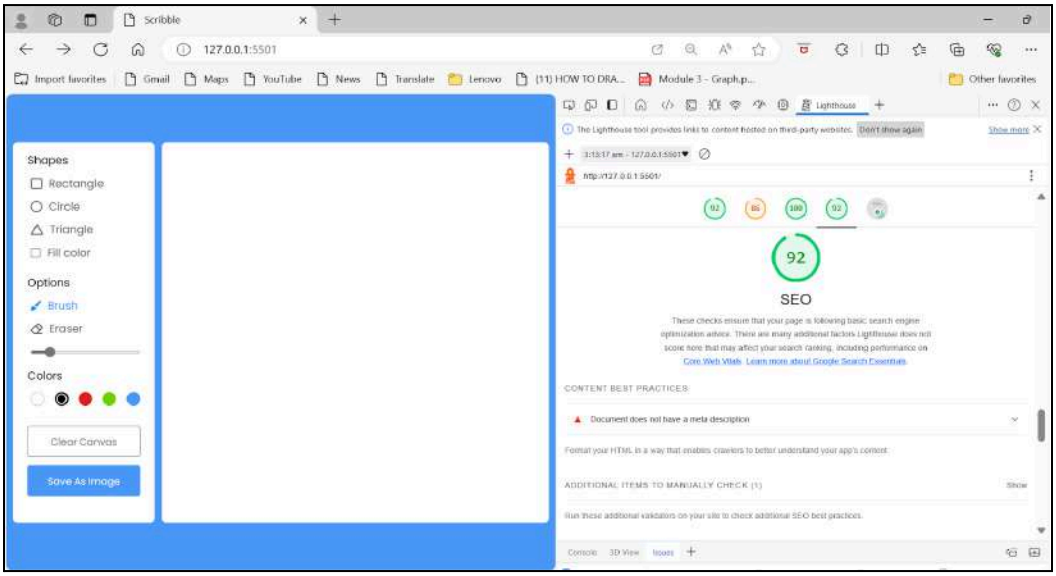
Project Title: Flutter : Thread Clone/ PWA : Scribble **RollNo. 14**

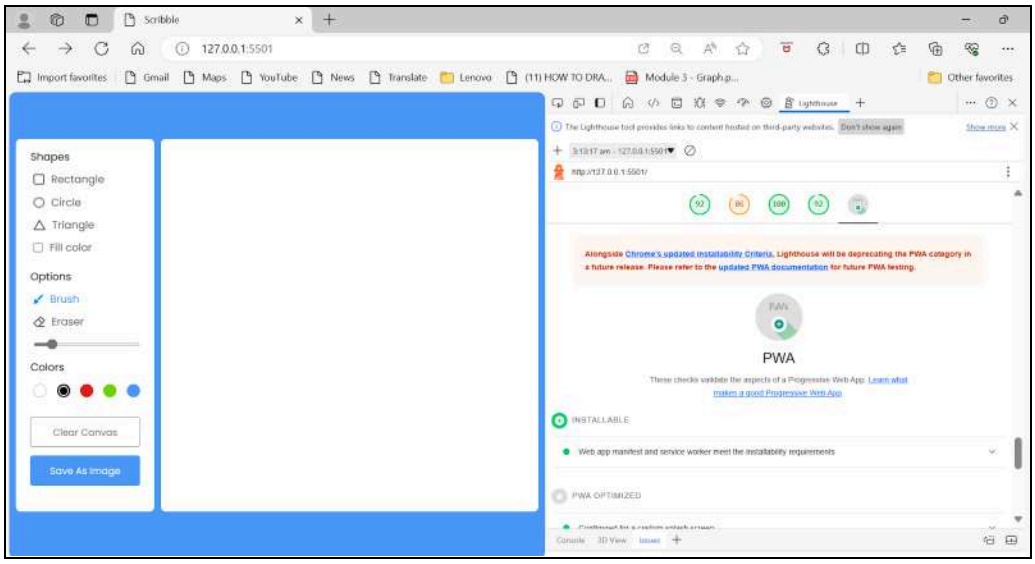




Project Title: Flutter : Thread Clone/ PWA : Scribble **RollNo. 14**







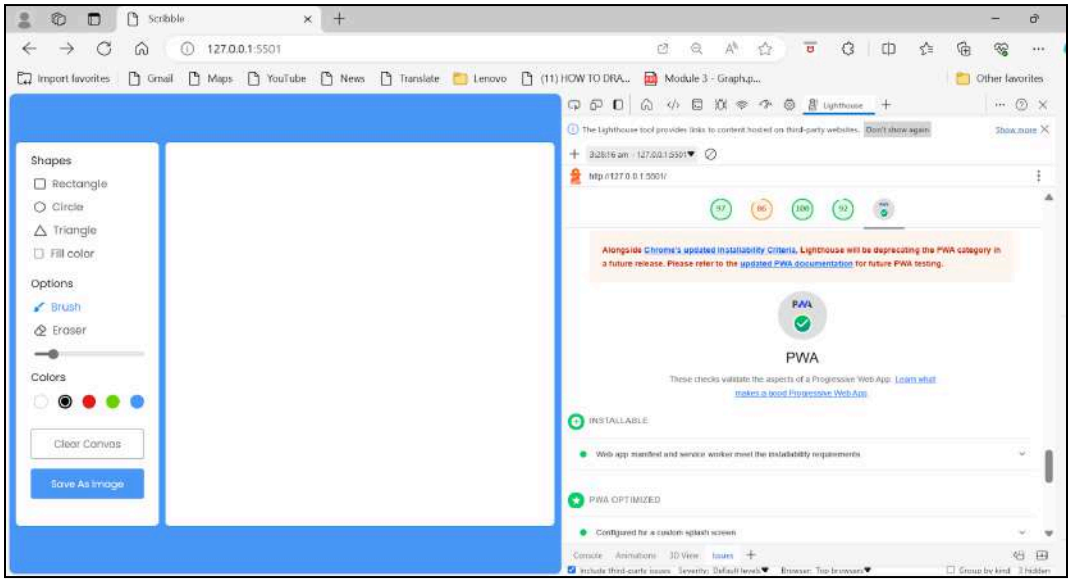
Changes in the index.html

```
<meta name="theme-color" content="#4285f4">
<link rel="apple-touch-icon" href="">
```

Changes made in the manifest.json

```
{
   "name":"Scribble",
   "short_name":"PWA",
   "start_url":"index.html",
   "display":"standalone",
   "background_color":"#5900b3",
   "theme_color":"black",
   "scope": ".",
   "description":"This is a PWA tutorial.",
   "icons":[
   {
   "src":"icons/ma.png",
   "sizes":"192x192",
   "type":"image/png"
   },
   {
   "src":"icons/twitter.png",
   "sizes":"512x512",
   "type":"image/png",
   "purpose":"any maskable"
   }
```

```
  ]
}
```



CONCLUSION

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

MAD & PWA Lab
Journal

Experiment No.	Assignment-1
Assignment 1 Questions	1. Flutter Overview:Explain the key features and advantages of using Flutter for mobile app development. Discuss how the Flutter framework differs from traditional approaches and why it has gained popularity in the developer community. 2. Widget Tree and Composition: Describe the concept of the widget tree in Flutter. Explain how widget composition is used to build complex user interfaces. Provide examples of commonly used widgets and their roles in creating a widget tree. 3. State Management in Flutter: Discuss the importance of state management in Flutter applications. Compare and contrast the different state management approaches available in Flutter, such as setState, Provider, and Riverpod. Provide scenarios where each approach is suitable. 4. Firebase Integration in Flutter: Explain the process of integrating Firebase with a Flutter application. Discuss the benefits of using Firebase as a backend solution. Highlight the Firebase services commonly used in Flutter development and provide a brief overview of how data synchronization is achieved.
Roll No.	14
Name	Harshita Dubey
Class	D15A
Subject	MAD
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

Harshita Dubey
D15A (14)

05/02/2024

MAD - PWA LAB

Assignment - 01

Q1) Flutter Overview: Explain the key features and advantages of using flutter for mobile app developement. Discuss how the flutter framework differs from traditional approaches and why it has gained popularity in the developer community.

Ans Flutter, developed by Google, is a UI toolkit that enables the creation of natively compiled applications for mobile, web, and desktop from the single codebase. Its feature include:-

1. Single Codebase: Flutters allows developers to write one codebase for both iOS and Android platforms, reducing development time and effort.
2. Hot Reload: This feature enables developers to instantly see the impact of code changes, enhancing the development and debugging process.
3. Widgets: Flutter uses a widget-base architecture, providing a rich set of customizable and extensible Widgets for building complex UIs easily.
4. Performance: flutter's compiled code is highly optimized, delivering near-native performance, and it supports hardware-accelerated graphics for smooth animations and transitions.
5. Platform Integration: Flutter allows seamless integration with platform-specific features and APIs, providing a native look and feel for the app on both iOS and Android

Teacher's Sign.: _____

Flutter differs from traditional approaches, such as native development or cross-platform solutions, in several ways :-

(i) Performance : Flutter compiles to native code, ensuring high performance comparable to apps developed using native languages.

ii) Consistent UI : Flutter's widget-based approach allows developers to maintain a consistent UI across platform without compromising on native look and feel.

(iii) Productivity : Hot Reload enhances developer productivity by enabling real-time code changes, speeding up the development and testing cycle.

(iv) No Bridge : Unlike some cross-platform solutions that rely on a bridge to communicate with native components, flutter directly renders UI components, reducing performance overhead.

Flutter has gained popularity in the developer community due to its ability to address common pain points associated with mobile app development. Its focus on a single codebase performance, and ease of use has attracted developers, making it a prefferred choice for building cross-platform.

Teacher's Sign.: _____

Q2). 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 widgets tree.

Ans. Widget Tree

- The Widget Tree is a hierarchical architecture of widgets that defines the user interface of an application.
- Every visual element, from simple components to complex layouts, is represented by a widget.
- Widget can be categorized in 2 types
  - Stateless Widget.

It is immutable and it cannot change over time. Eg: Images.

- Stateful Widget.

Widget that can change its state over time Eg: buttons.

Widget Composition.

- Widget composition in flutter involves combining multiple simple widgets to create more.
- This composibility is a powerful concept that allows developers to build sophisticated user Interfaces by nesting widgets within each other

Commonly Used Widgets

(a) Container.

- A box model for padding, margin and decoration.

(b) Column & Row

- Layout widgets for arranging children vertically or horizontal

Teacher's Sign.:

© Stack.
Overlapping widgets, allowing them to be layered on top of each other

ⓓ ListView.
A Scrollable ~~grid~~ list of widgets.

ⓔ GridView
A Scrollable grid of widgets.

ⓕ AppBar
A material design app bar typically at top of screen.

ⓖ Text field.
An input field for users to enter text.

ⓗ Button Widgets.
Interactive Buttons for user actions.

Q3). State Management in Flutter
2nd Discuss the Importance of State Management in Flutter applications. Compare and Contrast the different state management approaches available in flutter, such as setState, Provider, Riverpod. Provide scenario where each approach is suitable.

Ans → • ~~Statement~~ statemanagement is crucial in flutter applications because it invdves managing the data that can change over time.

• Flutter is reactive, meaning the UI rebuilds when the underlying data changes.

Teacher's Sign.: _____

Set State	Provider	Riverpod
① Built-in flutter Method	① External package named ('provider')	① External package ('riverpod')
② local state within a widget	② Global state within widget tree	② Global state with additional features
③ Limited scalability for large apps	③ Suitable for medium sized apps	③ Designed for large and complex apps
④ May lead to code redundancy	④ Balance simplicity & readability	④ Emphasize readability clean syntax

→ Scenario where each is applicable.

① Set State
- For small to moderately complex applications.
- When managing local state within a widget.

Eg. Simple forms, UI components etc.

② Provider
- For small medium to large-sized applications
- When a centerized state is needed.

Eg - Managing user authentication

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

Teacher's Sign.: _____

Q4) Firebase Integration in flutter. Explain the process of integrating firebase with flutter application. Discuss the benefits of using firebase as a backend solution. Highlight the services of firebase commonly used in Flutter development and provide a brief overview of how data synchronization is achieved.

Ans Integration.

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

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

3. Run flutter pub get.
4. Initialise firebase by calling `firebase.initialiseApp()` in main.

```
import 'package.firebase_core/firebase_core.dart'
void main() async{
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp()
  runApp(MyApp());
}
```

Benefits of Using firebase as Backend.

① Real-time Database.
- firebase offers a real-time NoSQL database.

② Authentication
- Provides a secure and easy to implementation solution user authentication

③ Cloud · firestore.
firebase's Cloud firestore provides a secure and scalable NOO NoSQL database that allows you to store & sync data in real-time.

④ Hosting.
firebase Hosting provides a simple and efficient way to deploy and host web application.

Data Synchronisation.

① Real time Database.
- When data changes on one client, it triggers events that automatically update data on other client.

② Cloud Firestore.
- It notifies clients when data changes, allowing for seamless real-time updates.

③ Authentication.
- If user sign in or out on one device, the authentication that is automatically reflected on other devices.

Teacher's Sign.:

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.	Harshita Dubey
Name	14
Class	D15A
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

Harshita Dubey
DISA (14)

22/03/2024.

Date
Page

PWA Assignment - 2

Q1] Define Progressive Web App (PWA) & explain its significance in modern web development. Discuss the key characteristics that differentiate PWA from traditional web Apps.

→ • A progressive Web App (PWA) is a type of web application that uses modern web technologies to provide a native app-like experience to users.
• They are designed to work on any platform that uses a standard compliant browser, such as desktop and mobile devices.
• key Characteristics that differentiate PWAs :
[i] Cross-Platform compatibility.
PWAs work across different devices & platforms.
[ii] Responsive Design.
It is built with responsive design principle.
[iii] Offline functionality.
It can work offline or with poor internet connection.
[iv] App-like experience.
It offers features such as push notification, homescreen installation, full-screen mode making them feel like native-app.
[v] fast Performance
[v] Discoverability.
It is discoverable through search engines & can be shared easily.
• Overall, PWAs combine the best feature of web & mobile apps, offering a compelling alternative to traditional native app development.

Teacher's Sign.: