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

Lab Assistant

Subject Teacher

Mrs. Kajal Joseph

Principal

Head of Department

Dr. Mrs. Shalu Chopra

Project Title: Roll No.

Name of the Course : MAD & PWA Lab **Course Code :** ITL604 **Year/Sem/Class :**

D15A/D15B **A.Y.:** 24-25 **Faculty Incharge** : Mrs. Kajal Joseph.

Lab Teachers : Mrs. Kajal Joseph.

Email : kajal.jewani@ves.ac.in

Programme Outcomes: The graduate will be able to:

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

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

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

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

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

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

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

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

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

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

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

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

Program specific Outcomes

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

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

Project Title: Roll No.

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 hostingsolutions	L3, L4

Project Title: Roll No.

Index

Experiment Title	LO	Grade
To install and configure the Flutter Environment	LO1	
To design Flutter UI by including common widgets.	LO2	
To include icons, images, fonts in Flutter app	LO2	
To create an interactive Form using form widget	LO2	
To apply navigation, routing and gestures in Flutter App	LO2	
To Connect Flutter UI with fireBase database	LO3	
To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”.	LO4	
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	
To implement Service worker events like fetch, sync and push for E-commerce PWA	LO5	
To study and implement deployment of Ecommerce PWA to GitHub Pages.	LO5	
To use google Lighthouse PWA Analysis Tool to test the PWA functioning.	LO6	
Assignment-1	LO1,LO2 ,LO3	
Assignment-2	LO4,LO5 ,LO6	

MAD & PWA Lab

Journal

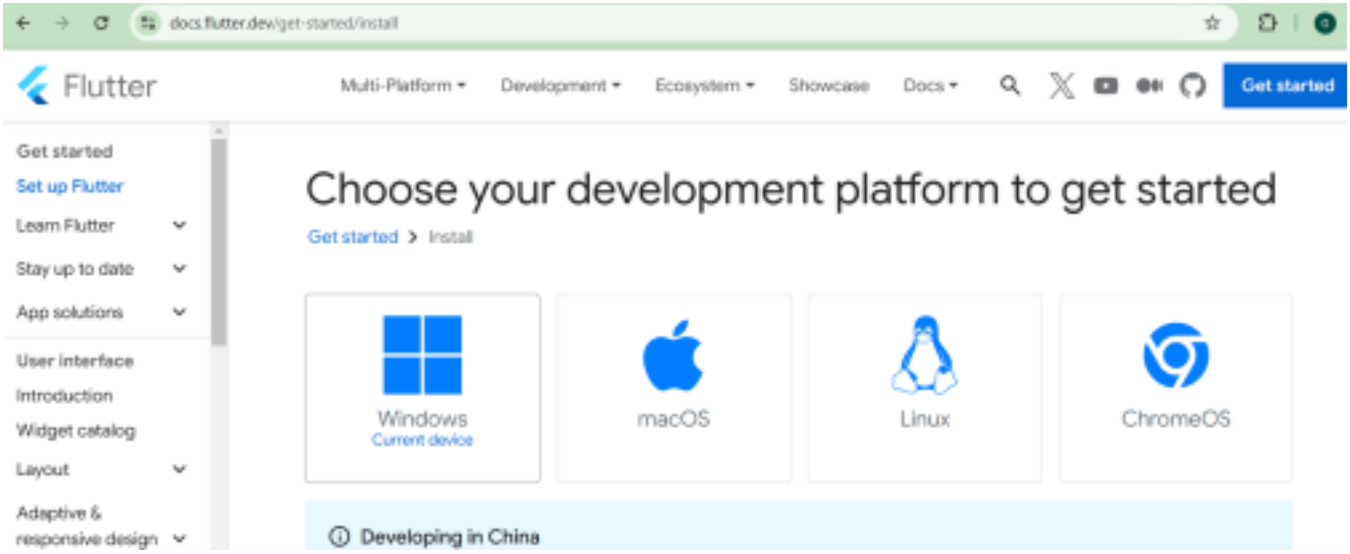
Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	03
Name	Kartik Bhat
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	

Name: kartik bhat Roll No: 03 Div:D15A

EXP 1: Installation and Configuration of Flutter Environment.

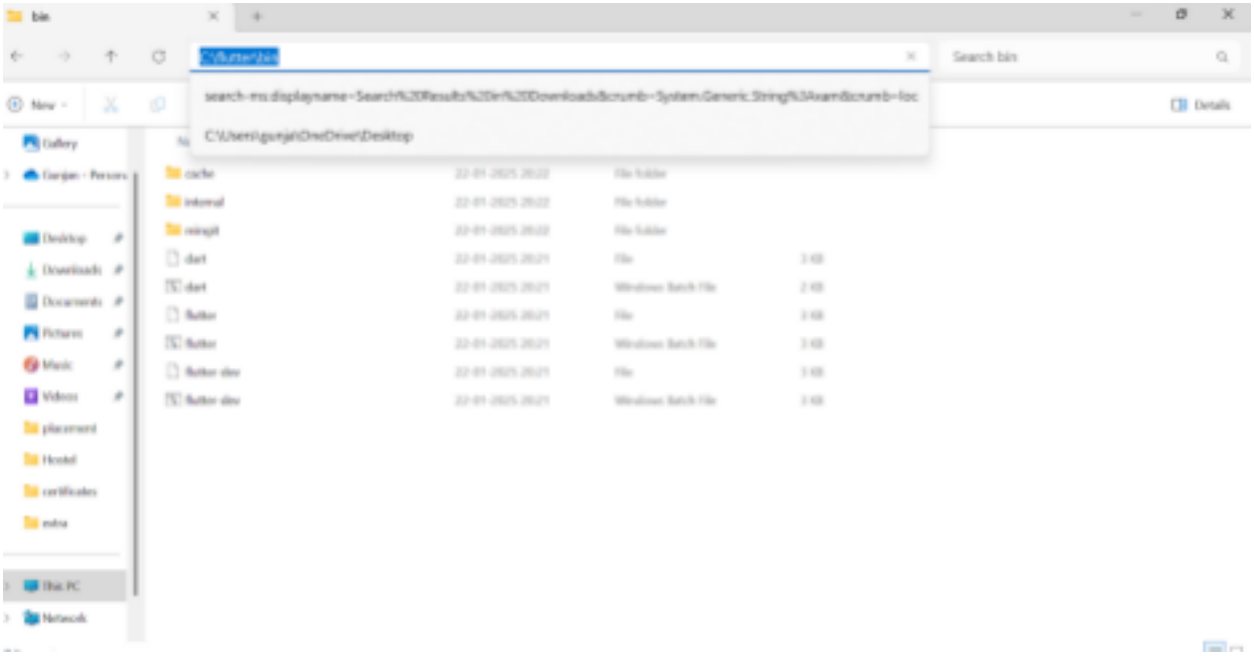
Install the Flutter SDK

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



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

Coping Bin path:

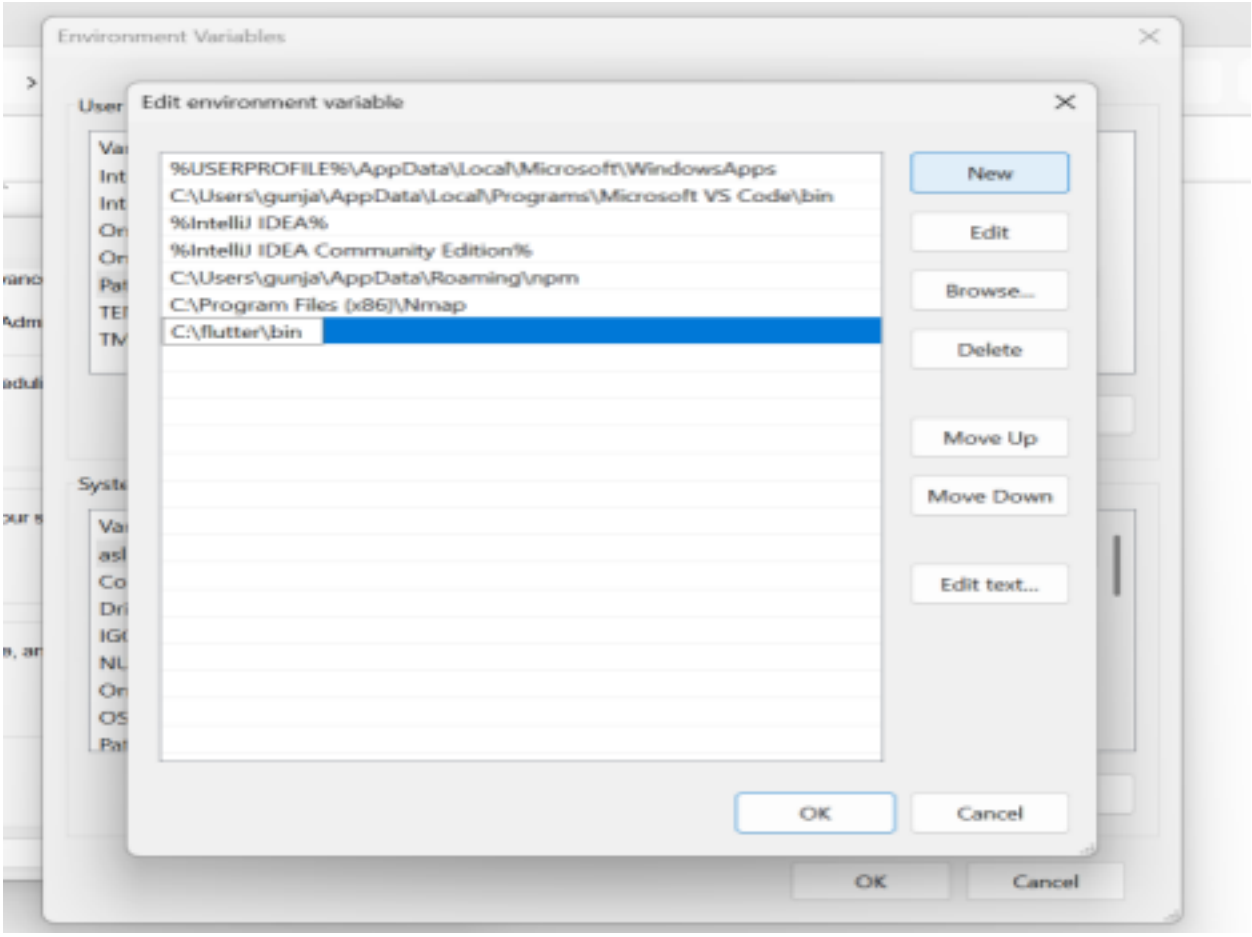


Step 3: When your download is complete, extract the **zip** file and place it in the desired installation folder or location, for example, C: /Flutter.

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

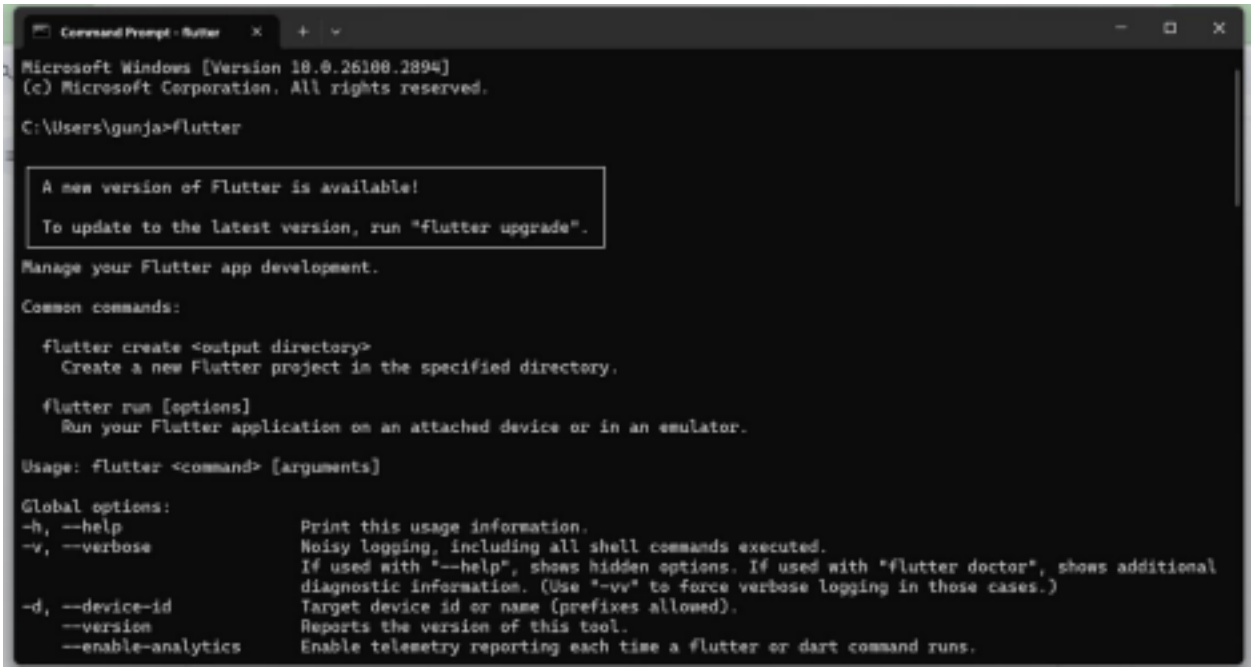
Step 4.1: Go to MyComputer properties -> advanced tab -> environment variables. You will get

the following screen.

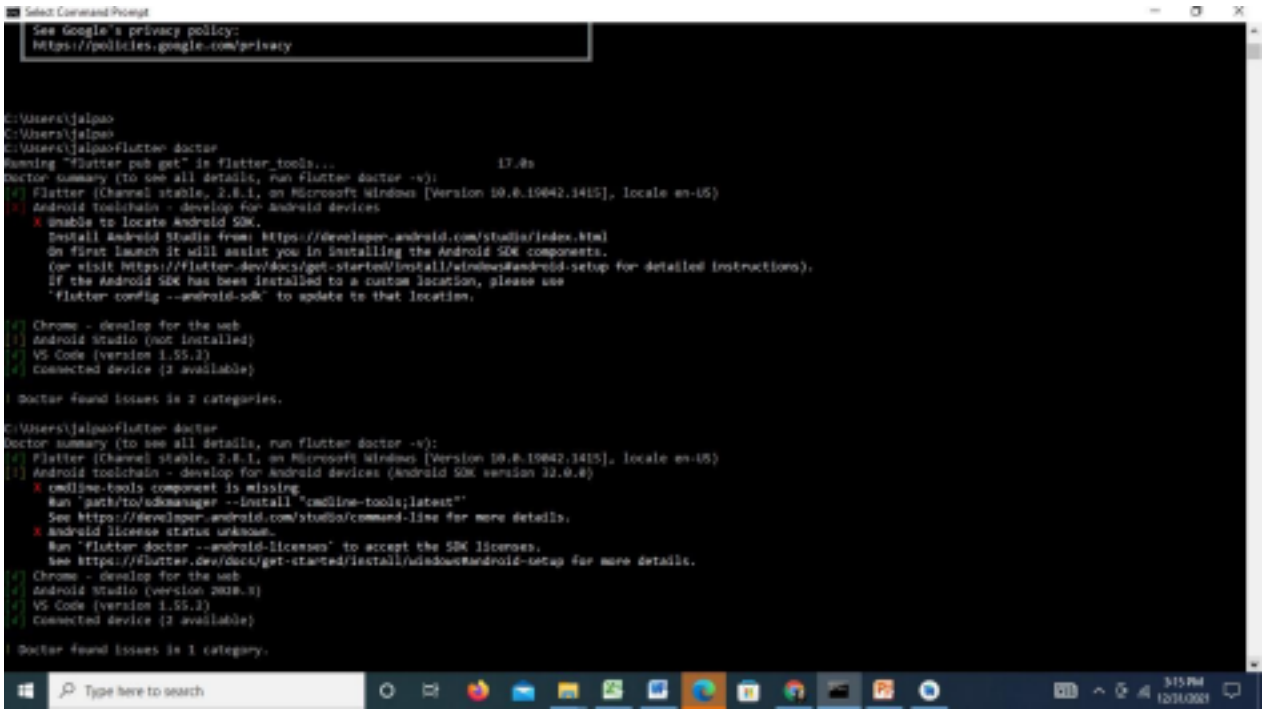


Step 4.3: In the above window, click on New->write path of Flutter bin folder in variable value -> ok -> ok -> ok.

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



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



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

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

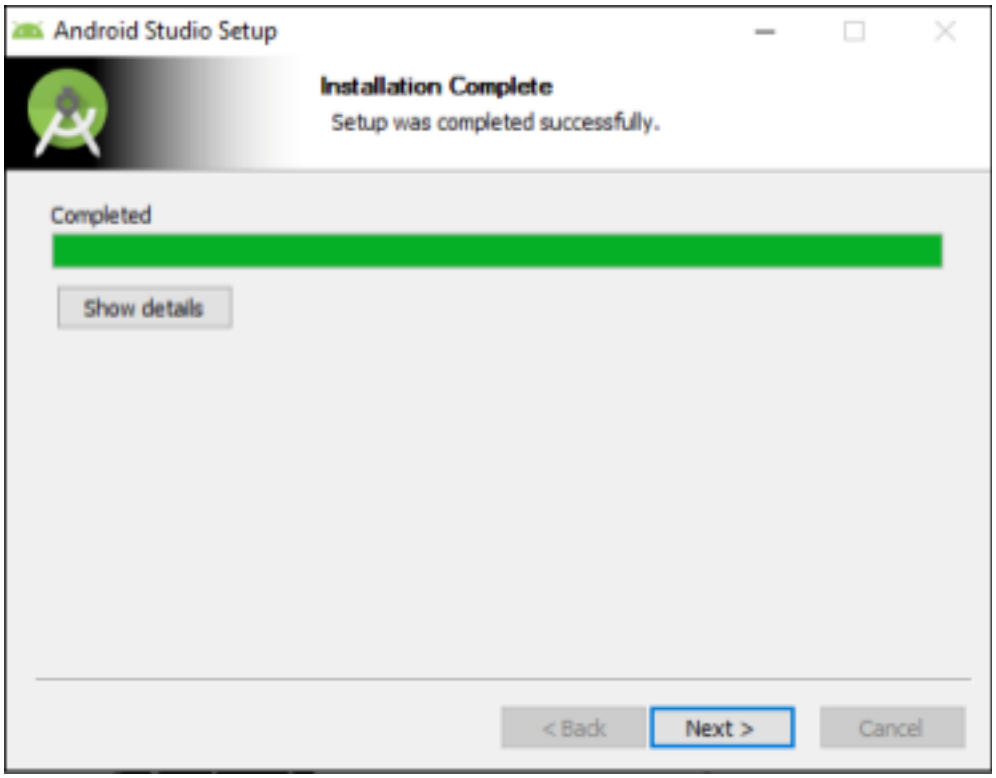
Step 7.1: Download the latest Android Studio executable or zip file from the official site. **Step**

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

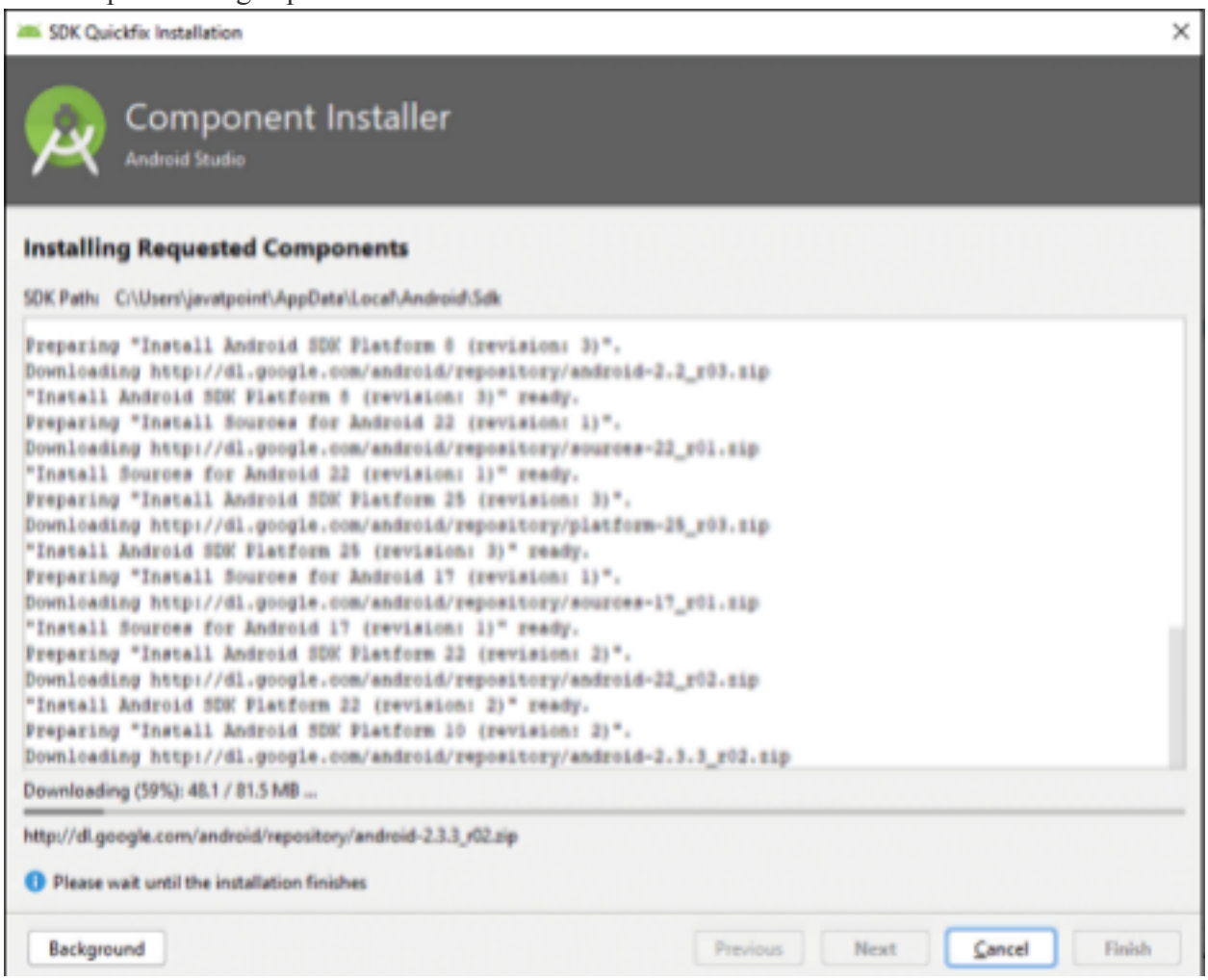
following dialog box.

Step 7.3: Follow

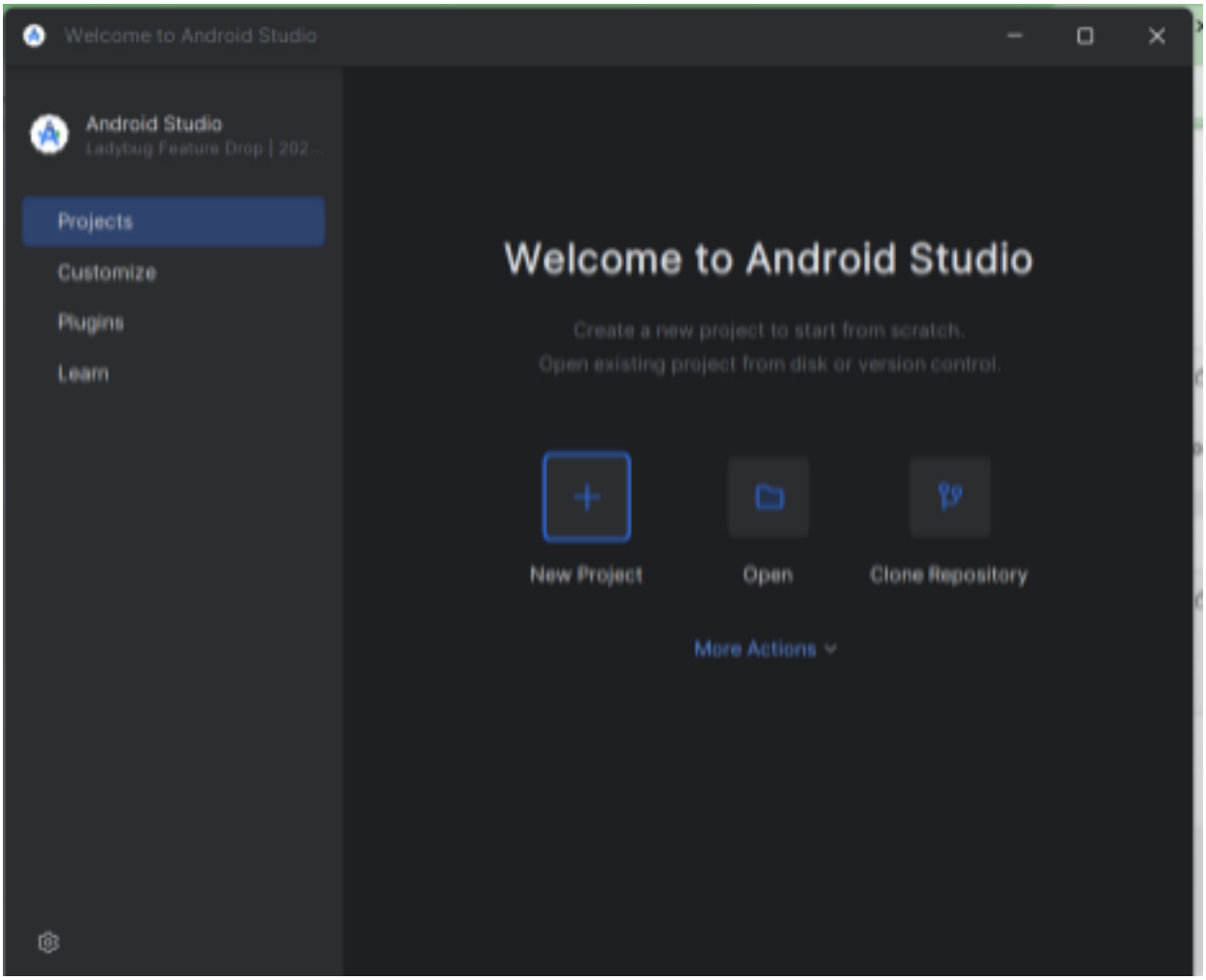
the steps of the installation wizard. Once the installation wizard completes, you will get the following screen.



Step 7.4: In the above screen, click Next-> Finish. Once the Finish button is clicked, you need to choose the 'Don't import Settings option' and click OK. It will start the Android Studio.

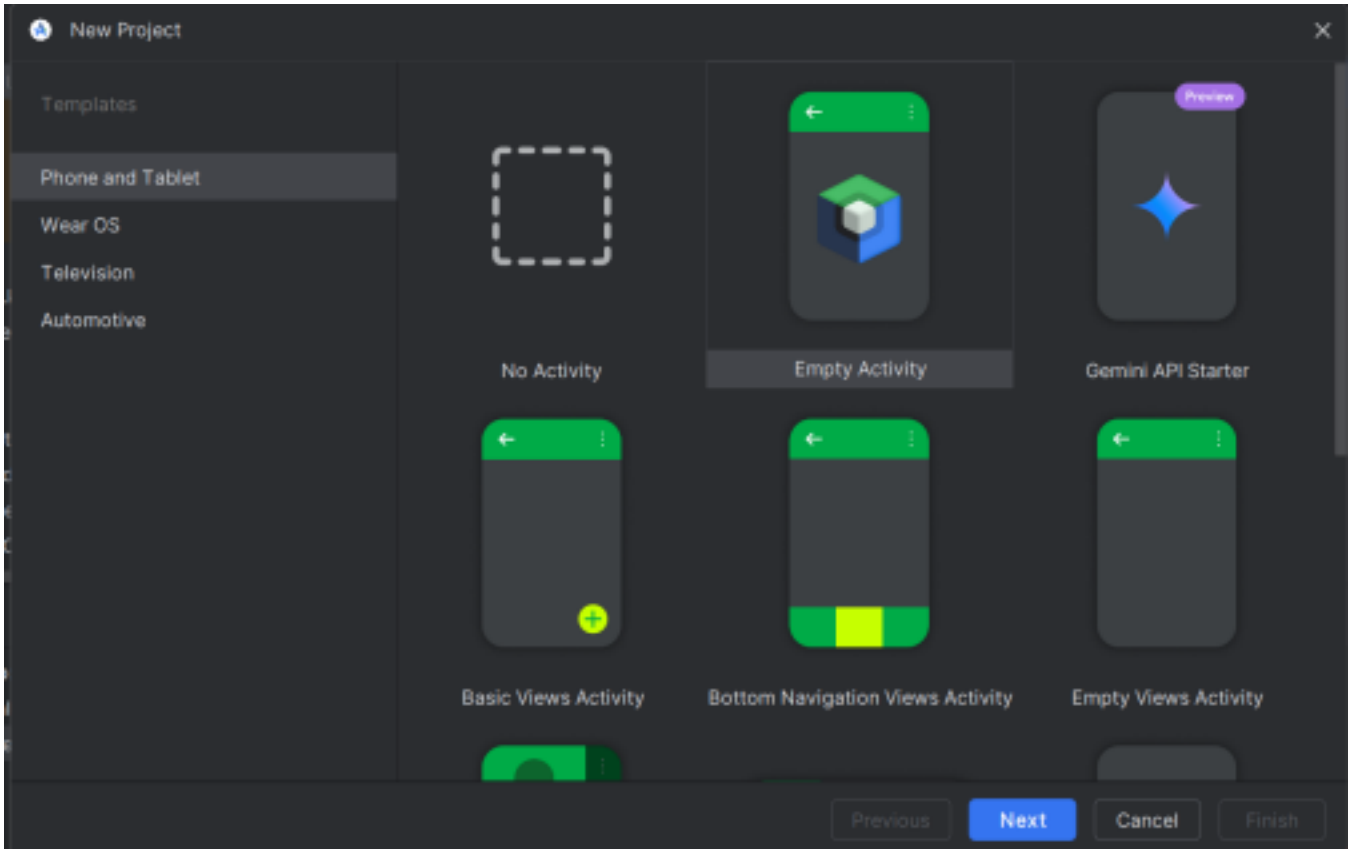


Step 7.5 run the $ **flutter doctor** command and Run flutter doctor --android-licenses command.



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

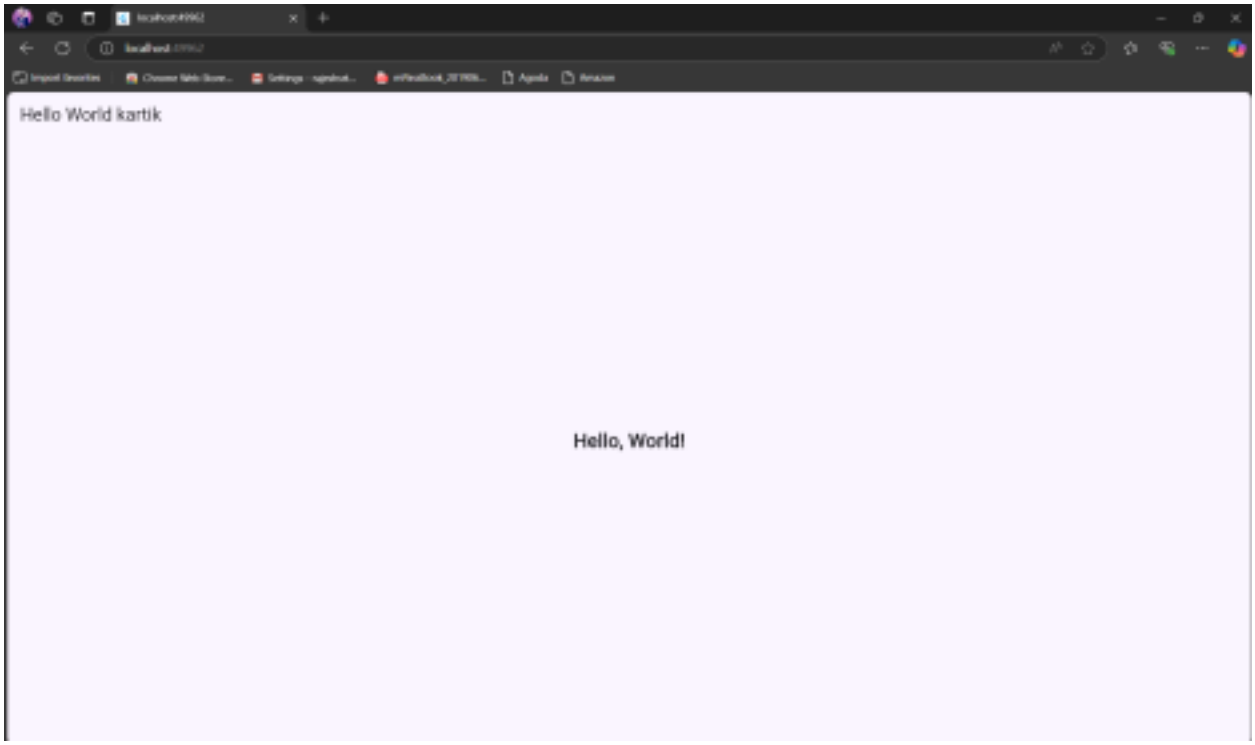
Step 8.1: To set an Android emulator, go to Android Studio > Tools > Android > AVD Manager and select Create Virtual Device. Or, go to Help->Find Action->Type Emulator in the search box. You will get the following screen.

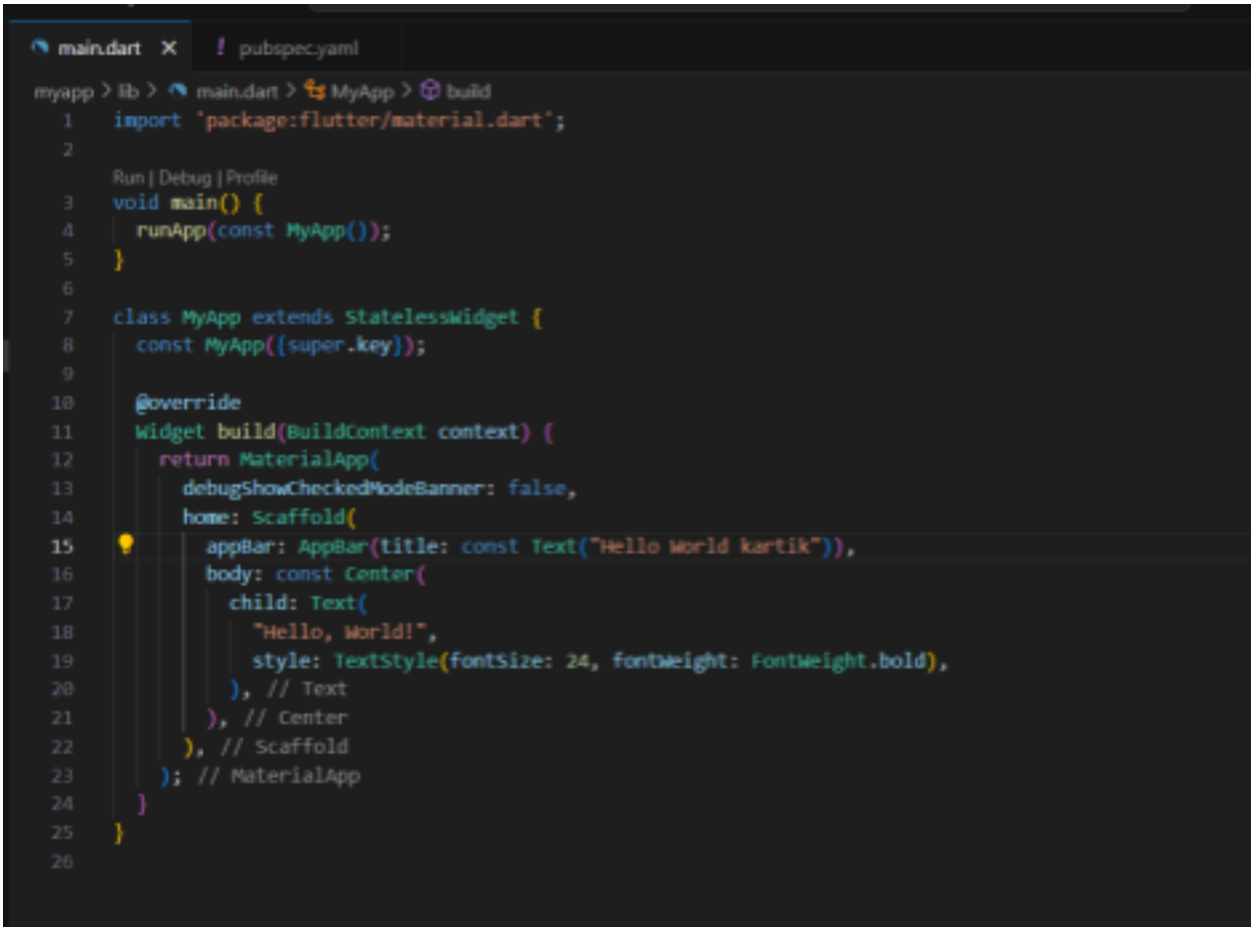


Step 8.2: Choose your device definition and click on Next.

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

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




main.dart
pubspec.yaml
myapp > lib > main.dart > MyApp > build
import 'package:flutter/material.dart';
Run | Debug | Profile
void main() {
runApp(const MyApp());
}
class MyApp extends StatelessWidget {
const MyApp({super.key});
@override
Widget build(BuildContext context) {
return MaterialApp(
debugShowCheckedModeBanner: false,
home: Scaffold(
appBar: AppBar(title: const Text("Hello World kartik")),
body: const Center(
child: Text(
"Hello, World!",
style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold),
), // Text
), // Center
), // Scaffold
); // MaterialApp
}
}

Project Title: Roll No.

MAD & PWA Lab

Journal

Experiment No.	02
Experiment Title.	To design Flutter UI by including common widgets.
Roll No.	03
Name	Kartik bhat
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

EXPERIMENT NO 2

NAME-Kartik Bhat

CLASS-D15A

ROLL NO-03

AIM-To design Flutter UI by including common widgets

THEORY

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

Some of the common widgets

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

SYNTAX

AppBar creates a top navigator bar with title and icons.

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

Scaffold creates the basic layout structure of the app.

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

TabBar creates tabs for switching views

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

Drawer a side menu that slides in from left

```
Drawer(
 child: ListView(
 children: [
 DrawerHeader(
 decoration: BoxDecoration(color: Colors.blue),
 child: Text("Header"),
 ),
 ListTile(
 leading: Icon(Icons.star),
 title: Text("Menu Item"),
 onTap: () {},
 ),
 ],
 ),
)
```

Listview creates a scrollable list
dynamically ListView.builder(
itemCount: items.length,
itemBuilder: (context, index) {
return ListTile(
title: Text(items[index]),
);
},
)

Widget Properties

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

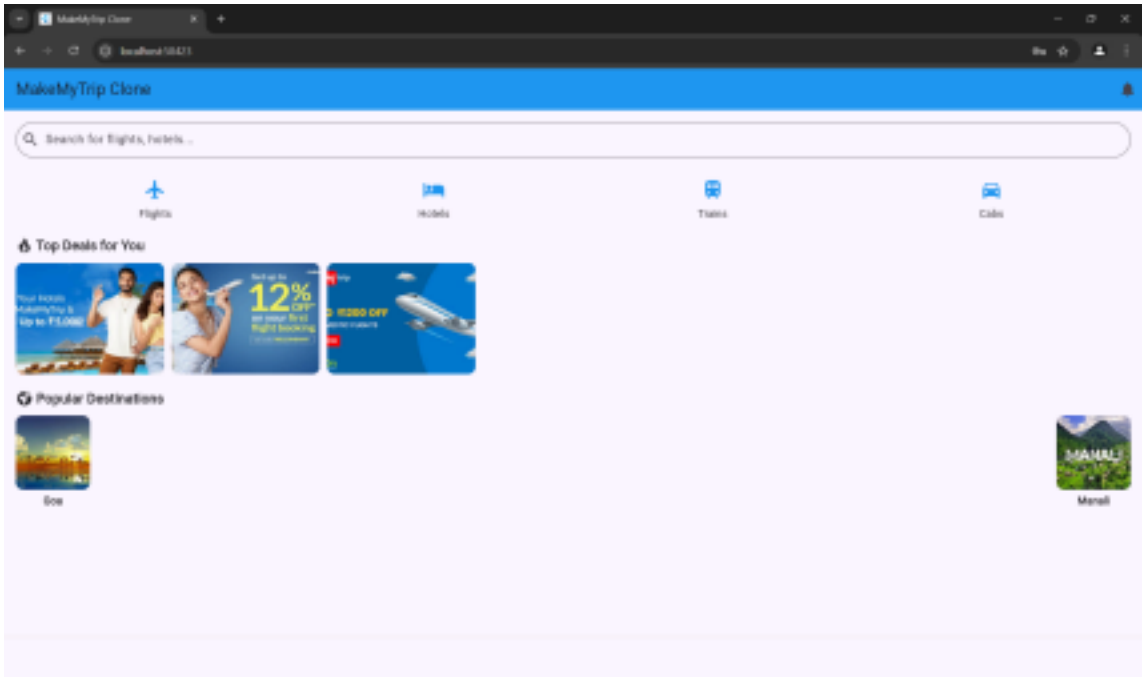
AppBar
title → Sets a title or an icon
leading → Adds an icon or button on the left
backgroundColor → Changes the background
color elevation → Controls the shadow effect
centerTitle → Aligns the title in the
center bottom → Adds a TabBar

Drawer
child → Contains a list of menu items
ListView → Displays menu options in a scrollable list
ListView
padding → Controls spacing around the list
children → Contains multiple widgets inside the
list
ListTile
leading → Adds an icon on the left
title → The main text of the item
trailing → Adds an icon on the right
onTap → Defines what happens when tapped
CODE

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

import 'package:firebase_core/firebase_core.dart' show Firebase;
import 'package:makemytrip_clone/firebase_options.dart' show
```

```
DefaultFirebaseOptions;

import 'package:makemytrip_clone/screens/splash_screen.dart';

// ignore: unused_import

import 'package:makemytrip_clone/screens/homepagescreen.dart';

import 'firebase_options.dart';



void main() async {

 WidgetsFlutterBinding.ensureInitialized();

 await Firebase.initializeApp(options:
DefaultFirebaseOptions.currentPlatform);

 runApp(const MyApp());

 }
class MyApp extends StatelessWidget {

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



 @override

 Widget build(BuildContext context) {

 return MaterialApp(

 debugShowCheckedModeBanner: false,

 title: 'MakeMyTrip Clone',

 theme: ThemeData(

 fontFamily: 'Roboto',

 primarySwatch: Colors.blue,
 ),

 home: const SplashScreen(), // ✅ Start with SplashScreen );

  }

 }
```

OUTPUT



User Profile page

when clicked on User Icon on top right corner

CODE

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


import
'package:makemytrip_clone/screens/booking_confirmation_screen.dar t ' ;
import 'package:makemytrip_clone/screens/payment_screen.dart'; class




ProfileScreen extends StatelessWidget {



    const ProfileScreen({Key? key}) : super(key: key);
```

```
@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: const Text("My Profile"),

centerTitle: true,

backgroundColor: Colors.blue,

iconTheme: const IconThemeData(color: Colors.white),
titleTextStyle: const TextStyle(color: Colors.white,
fontSize: 20),

),
body: Padding(

padding: const EdgeInsets.all(20.0),

child: Column(
crossAxisAlignment: CrossAxisAlignment.center,

children: [
```

```
const CircleAvatar(

radius: 50,

backgroundImage: AssetImage('assets/user.jpg'), ),

const SizedBox(height: 16),
const Text(

'johnny depp',

style: TextStyle(

fontSize: 22,

fontWeight: FontWeight.bold, ),

),
const Text(

'johndepp@email.com',

style: TextStyle(color: Colors.grey), ),
const SizedBox(height: 30), const

Divider(),

ProfileOption(
```

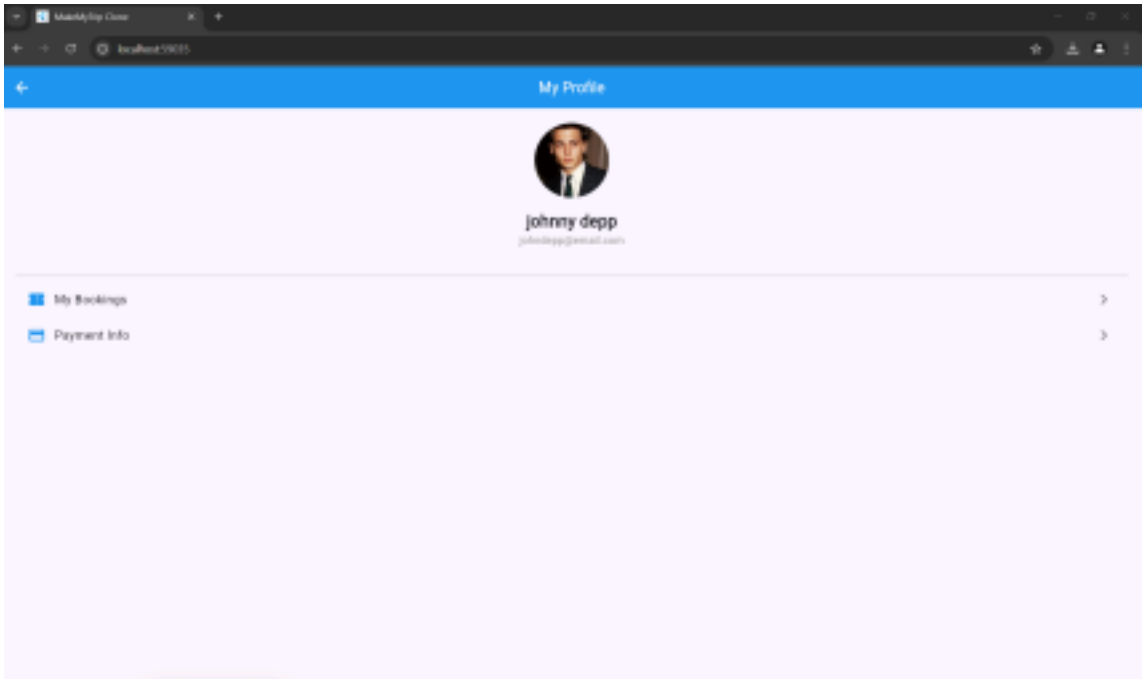
```
 icon: Icons.confirmation_num, title: 'My

Bookings',



 onTap: () {

 Navigator.push(

 context,

 MaterialPageRoute(builder: (context) =>
BookingConfirmationScreen(bookingType: '',)),

 );

 },
 ),

 ProfileOption(

 icon: Icons.payment,
 title: 'Payment Info',

 onTap: () {
 Navigator.push(

 context,

 MaterialPageRoute(builder: (context) =>
PaymentScreen(bookingType: '',)),
```

```
        );

      },

    ),

  ],

),

),

),

);
 }
}
```

```
class ProfileOption extends StatelessWidget {

final IconData icon;

 final String title;
 final VoidCallback onTap;

 const ProfileOption({

 super.key,
```

```
required this.icon,

required this.title,

required this.onTap,
});




@override

Widget build(BuildContext context) {
return ListTile(

leading: Icon(icon, color: Colors.blue),

title: Text(title),

trailing: const Icon(Icons.arrow_forward_ios, size: 16),

onTap: onTap,

);

}

}
```

OUTPUT


My Profile
johnny depp
My Bookings
Payment Info

Project Title: Roll No.

MAD & PWA Lab

Journal

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

EXPERIMENT 3

NAME-kartik bhat

CLASS-D15A

ROLL NO-03

AIM-To include images, fonts in flutter app.

THEORY

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

A) Local images can be stored in the project directory and loaded into the app. Steps to add Local images

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

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

B) Network Images

Flutter allows displaying images from the internet using

Image.network():

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

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

SYNTAX

1)Create an assets folder for Local images.

Declare assets in pubspec.yaml file.

flutter:

assets:
- assets/image1.png

- assets/images/image2.jpg

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

2)If using network Images

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

) 3)Install fontawesome package in flutter

Add this dependency in pubspec.yaml file

dependencies:

font_awesome_flutter: ^10.7.0

Run flutter pub get

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

Widget properties

1)image

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

Example

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

width: 100,
height: 100,

fit: BoxFit.contain,

loadingBuilder: (context, child, progress) {

return progress == null ? child :

CircularProgressIndicator(); },

errorBuilder: (context, error, stackTrace) {
return Icon(Icons.error);

},

)

2)font

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

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

ListTile(leading: Icon(Icons.star), title: Text("Premium"), onTap: () {}),

ListTile(leading: Icon(Icons.bookmark), title: Text("Bookmarks"),
onTap: () {}),

ListTile(leading: Icon(Icons.list), title: Text("Lists"), onTap: () {}),
ListTile(

leading: Icon(Icons.logout, color: Colors.red), title:
Text("Logout"), onTap: _logout),

],

OUTPUT
CODE

```
@override
Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: const Text("My Profile"),
```

```
centerTitle: true,

backgroundColor: Colors.blue,

iconTheme: const IconThemeData(color: Colors.white),
titleTextStyle: const TextStyle(color: Colors.white,
fontSize: 20),

),

body: Padding(

padding: const EdgeInsets.all(20.0),

child: Column(
crossAxisAlignment: CrossAxisAlignment.center,

children: [

const CircleAvatar(

radius: 50,

backgroundImage: AssetImage('assets/user.jpg'), ),

const SizedBox(height: 16),

const Text(

'johnny depp',

style: TextStyle(

fontSize: 22,

fontWeight: FontWeight.bold,

),

),

const Text(

'johndepp@email.com',
style: TextStyle(color: Colors.grey), ),

const SizedBox(height: 30),

const Divider(),

ProfileOption(
```

```
 icon: Icons.confirmation_num,

 title: 'My Bookings',

 onTap: () {
 Navigator.push(
 context,

 MaterialPageRoute(builder: (context) =>
BookingConfirmationScreen(bookingType: '',)),

 );

 },

 ),

 ProfileOption(

 icon: Icons.payment,

 title: 'Payment Info',

 onTap: () {

 Navigator.push(

 context,

 MaterialPageRoute(builder: (context) =>
PaymentScreen(bookingType: '',)),

 );

 },

 ),

 ],

 ),

 ),

 );
 }

 }
class ProfileOption extends StatelessWidget {

 final IconData icon;

 final String title;
```

```
final VoidCallback onTap;


const ProfileOption({
super.key,

required this.icon,

required this.title,

required this.onTap,

});


@override

Widget build(BuildContext context) {

return ListTile(

leading: Icon(icon, color: Colors.blue),

title: Text(title),

trailing: const Icon(Icons.arrow_forward_ios, size: 16),

onTap: onTap,

);

}

}
```

OUTPUT

Project Title: Roll No.

MAD & PWA Lab

Journal

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget
Roll No.	03
Name	Kartik bhat
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

EXPERIMENT NO 4

NAME-kartik bhat

CLASS-D15A

ROLL NO-03

AIM-To create an interactive form using form widget

THEORY

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

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

Key Features of Interactive Forms

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

Components of Form Widget

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

SYNTAX

```
Form(
```

```
 key: formKey, // Unique key to manage form
state child: Column(
 children: [
 TextFormField(
 decoration: InputDecoration(labelText: "Enter your name"),
validator: (value) {
 if (value == null || value.isEmpty) {
 return "This field cannot be empty";
 }
 return null;
 },
 ),
 SizedBox(height: 10),
 ElevatedButton(
 onPressed: () {
 if (formKey.currentState!.validate()) {
 // Perform form submission action
 }
 },
 child: Text("Submit"),
 ),
 ],
 ),
 )
```

Widget Properties

1)key

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

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

```
Form(
 key: _formKey,
 child: Column(
```

```
children: [ /* Form fields go here */ ],
 ),
);
```

2)child

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

Example:

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

3)onchanged

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

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

CODE

The introduction page

```dart
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
// ignore: unused_import
import 'home_screen.dart';
// ignore: unused_import
import 'signup_screen.dart';

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

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

class _LoginScreenState extends State<LoginScreen> {
final TextEditingController _emailController =
TextEditingController();
 final TextEditingController _passwordController =
TextEditingController();
 String? errorMessage;

 Future<void> _login() async {
 try {
 await FirebaseAuth.instance.signInWithEmailAndPassword(
email: _emailController.text.trim(),
 password: _passwordController.text.trim(),
 );
 Navigator.pushReplacementNamed(context, '/home'); }
on FirebaseAuthException catch (e) {
 setState(() {
 errorMessage = e.message;
  });
  }
  }

 @override
 Widget build(BuildContext context) {
 return Scaffold(
 body: Stack(
 children: [
 // ?? Background Image
 Positioned.fill(
 child: Image.asset(
```

```
 "assets/mmt_background.jpg",
 fit: BoxFit.cover,
 ),
 ),

 // ?? Dark overlay
 Positioned.fill(
  child: Container(color: Colors.black.withOpacity(0.5)), ),

 // ?? Login Form
 Center(
 child: SingleChildScrollView(
 padding: const EdgeInsets.all(16.0),
 child: Column(
 mainAxisSize: MainAxisSize.min,
 children: [
 Image.asset("assets/mmt_logo.png", height: 80), const
SizedBox(height: 20),

 TextField(
 controller: _emailController, keyboardType:
TextInputType.emailAddress, decoration: InputDecoration(
 labelText: 'Email',
 filled: true,
 fillColor: Colors.white.withOpacity(0.8), border: const
OutlineInputBorder(), ),
 ),
 const SizedBox(height: 10),

 TextField(
 controller: _passwordController, obscureText:
true,
 decoration: InputDecoration(
 labelText: 'Password',
 filled: true,
 fillColor: Colors.white.withOpacity(0.8), border: const
OutlineInputBorder(), ),
 ),
 const SizedBox(height: 10),

 if (errorMessage != null)
 Padding(
```

```
 padding: const EdgeInsets.only(bottom: 10), child: Text(
 errorMessage!,
                          style: const TextStyle(color: Colors.red,
fontWeight: FontWeight.w600),
 ),
 ),
 ElevatedButton(
 onPressed: _login,
 style: ElevatedButton.styleFrom( backgroundColor:
Colors.blueAccent, foregroundColor: Colors.white,
padding: const
EdgeInsets.symmetric(horizontal: 50, vertical: 12),
 ),
 child: const Text("Login"), ),
 const SizedBox(height: 10),

 TextButton(
 onPressed: () {
 Navigator.pushNamed(context, '/signup'); },
 child: const Text(
 "Don't have an account? Sign up", style: TextStyle(color:
Colors.white), ),
 ),
 ],
 ),
 ),
),
],
),
);
}
}
```

OUTPUT

Create Account Page

CODE

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

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

'package:cloud_firestore/cloud_firestore.dart'; // ignore: unused_import

import 'login_screen.dart';


class SignUpScreen extends StatefulWidget { const SignUpScreen({Key? key})

: super(key: key);


@override

State<SignUpScreen> createState() =>
_SignUpScreenState();

}


class _SignUpScreenState extends State<SignUpScreen> { final

TextEditingController _emailController =

    TextEditingController();

     final TextEditingController _passwordController =
    TextEditingController();

     String? errorMessage;


     Future<void> _signup() async {

     try {
```

```
 final userCredential = await
FirebaseAuth.instance.createUserWithEmailAndPassword(
email: _emailController.text.trim(), password:
_passwordController.text.trim(), );


 // Store extra user info in Firestore (optional but
useful)
 await FirebaseFirestore.instance
  .collection('users')
  .doc(userCredential.user!.uid) .set({ 'email':
 _emailController.text.trim(), 'name': '', // you
 can ask for this in form later 'photoUrl': '',
 // default empty });


 // After signup, go to login screen
 Navigator.pushReplacementNamed(context,
 '/login');
  } on FirebaseAuthException catch (e) {
setState(() {
 errorMessage = e.message;
  });
  }
  }
 @override
 Widget build(BuildContext context) {
 return Scaffold(
 appBar: AppBar(title: const Text("Sign Up")),
body: Padding(
```

```
 padding: const EdgeInsets.all(16.0),

child: Column(

 children: [

 TextField(

 controller: _emailController,

 decoration: const
InputDecoration(labelText: 'Email'),
 keyboardType:
TextInputType.emailAddress,

 ),

 const SizedBox(height: 10),

TextField(

 controller: _passwordController,
 decoration: const
InputDecoration(labelText: 'Password'),

obscureText: true,


 ),

 const SizedBox(height: 20), if

 (errorMessage != null)

 Text(errorMessage!, style: const
TextStyle(color: Colors.red)),

 ElevatedButton(

 onPressed: _signup,
 child: const Text("Sign Up"), ),

 ],

 ),

 ),

 );

  }

 }
```

OUTPUT
T

Project Title: Roll No.

MAD & PWA Lab

Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	03
Name	Kartik bhat
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

EXPERIMENT NO 5

NAME-kartik bhat

CLASS-D15A

ROLL NO-03

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

THEORY

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

Types of Navigation

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

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

Types of Routing

1. Direct Route Navigation (MaterialPageRoute)-Used for simple page-to page navigation.
2. Named Routes (Predefined Routes in main.dart)-Defined in the MaterialApp widget and used throughout the app. Flutter uses the GestureDetector widget to detect user interactions like taps, swipes, pinches, and long presses. This is essential for making an app interactive.

Common Gestures & Their Uses:

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

for zooming.

SYNTAX

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

Routing

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

Gestures

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

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

Widget Properties

Navigator

context → The current build context for navigation.

MaterialPageRoute → Creates a transition animation between pages. builder → Defines the widget to navigate to.

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

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

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

Routing

initialRoute → Sets the first screen when the app starts. routes → Defines a map of route names and corresponding widgets.

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

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

onDoubleTap → Detects a double tap.

onLongPress → Detects when the user presses and holds.

onHorizontalDragStart → Detects when a horizontal drag begins.

onHorizontalDragUpdate → Detects movement during a horizontal drag. onHorizontalDragEnd → Detects when a

horizontal drag stops.

CODE

```
// Navigate to Home Page
 Navigator.pushReplacement(
 context,
 MaterialPageRoute(builder: (context) => const
TwitterHomePage()), );
 }
```

OUTPUT

CODE

To go the user profile

```
onTap: () {
 Navigator.push(
 context,
 MaterialPageRoute(builder: (context) => const UserProfilePage()), );
 },
```

OUTPUT

Project Title: Roll No.

MAD & PWA Lab

Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	03
Name	Kartik bhat
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:	

EXPERIMENT NO 6

NAME-kartik bhat

CLASS-D15A

ROLL NO-03

AIM-To connect Flutter UI with firebase.

THEORY

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

Steps to Add firebase to our Flutter app using

Firebase CLI 1.Install the Firebase CLI and log in

(run firebase login)

2.From any directory, run this command:

- dart pub global activate flutterfire_cli

3.Then, at the root of your Flutter project directory, run this

command: flutterfire configure --project=questitnextjs

4.This automatically registers your per-platform apps with Firebase and adds a lib/firebase_options.dart configuration file to your Flutter project.

5.To initialise Firebase, call Firebase.initializeApp from the firebase_core package with the configuration from your new firebase_options.dart file:

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

await Firebase.initializeApp(
 options:

DefaultFirebaseOptions.currentPlatform, );
```

6.Add the dependencies in the

pubspec.yaml file Firebase_core : ^version

Firebase_auth : ^version

SYNTAX

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


Future<void> signInUser(String email, String password)
async { try {
 await
FirebaseAuth.instance.signInWithEmailAndPassword(
email: email,
 password: password,
 );
 print("User Signed In Successfully!");
} catch (e) {
 print("Error: $e");
}
}
```

Widget Properties

1)Firebase_Auth

- currentUser → Returns the currently signed-in user.
- signInWithEmailAndPassword(email, password) → Logs in a user.
- createUserWithEmailAndPassword(email, password) → Registers a new user.
- signOut() → Logs out the current user.

2)FirebaseFirestore

- collection("name") → Accesses a Firestore collection.
- doc("id") → Refers to a specific document.
- set(Map<String, dynamic> data) → Adds or updates data.
- get() → Fetches document data.

🗑 delete() → Deletes a document.

CODE

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

import 'package:cloud_firestore/cloud_firestore.dart';



class AuthService {

  final FirebaseAuth _auth = FirebaseAuth.instance; final

FirebaseFirestore _firestore = FirebaseFirestore.instance;



 Future<User?> signUp(String email, String password, String name,
String photoUrl) async {

 try {

 UserCredential userCred = await
_auth.createUserWithEmailAndPassword(

 email: email, password: password);



 await
_firestore.collection('users').doc(userCred.user!.uid).set({
 'name': name,

  'email': email,

  'photoUrl': photoUrl,

  });



 return userCred.user;

  } catch (e) {

 print(e);

 return null;
  }
```

```
  }


 Future<User?> login(String email, String password) async {
try {
 UserCredential userCred = await
_auth.signInWithEmailAndPassword(
 email: email, password: password);
 return userCred.user;
  } catch (e) {
 print(e);
 return null;
  }
  }


 User? getCurrentUser() {
return _auth.currentUser; }


 Future<void> signOut() async {
await _auth.signOut(); }
 }
```

OUTPUT

Project Title: Roll No.

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.	03
Name	Kartik bhat
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	

EXPERIMENT 7

Name-kartik bhat

Class-D15A

Roll no-03

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

Theory

Regular Web App

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

Progressive Web App

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

Difference between PWAs vs. Regular Web Apps:

A Progressive Web is different and better than a Regular Web app with features like:

1. Native Experience

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

2. Ease of Access

Unlike other mobile apps, PWAs do not demand longer download time and make memory space available for installing the applications. The

PWAs can be shared and installed by a link, which cuts down the number of steps to install and use. These applications can easily keep an app icon on the user's home screen, making the app easily accessible to the users and helps the brands remain in the users' minds, and improving the chances of interaction.

3. Faster Services

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

4. Engaging Approach

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

5. Updated Real-Time Data Access

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

6. Discoverable

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

7. Lower Development Cost

Progressive web apps can be installed on the user device like a native device, but it does not demand submission on an App Store. This makes it far more

cost-effective than native mobile applications while offering the same set of functionalities.

Pros and cons of the Progressive Web App

The main features are:

Progressive — They work for every user, regardless of the browser chosen because they are built at the base with progressive improvement principles.

Responsive — They adapt to the various screen sizes: desktop, mobile, tablet, or dimensions that can later become available.

App-like — They behave with the user as if they were native apps, in terms of interaction and navigation.

Updated — Information is always up-to-date thanks to the data update process offered by service workers.

Secure — Exposed over HTTPS protocol to prevent the connection from displaying information or altering the contents.

Searchable — They are identified as “applications” and are indexed by search engines.

Reactivable — Make it easy to reactivate the application thanks to capabilities such as web notifications.

Installable — They allow the user to “save” the apps that he considers most useful with the corresponding icon on the screen of his mobile terminal (home screen) without having to face all the steps and problems related to the use of the app store.

Linkable — Easily shared via URL without complex installations.

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

IOS support from version 11.3 onwards;

Greater use of the device battery;

Not all devices support the full range of PWA features (same speech for iOS and Android operating systems);

It is not possible to establish a strong re-engagement for iOS users (URL scheme, standard web notifications);

Support for offline execution is however limited;

Lack of presence on the stores (there is no possibility to acquire traffic from that channel);

There is no “body” of control (like the stores) and an approval process; Limited access to some hardware components of the

devices; Little flexibility regarding “special” content for users (eg loyalty programs, loyalty, etc.).

Code: -

//Manifest.json:

```
{

 "short_name": "CollegePortal",

 "name": "My College Portal PWA",

 "description": "A visually rich and interactive college portal
built as a Progressive Web App.",

 "start_url": ".",

 "display": "standalone",

 "theme_color": "#121212",
```

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.	03
Name	Kartik bhat
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

Experiment No. 8

Name-kartik bhat
Class-D15A
Roll No-03

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

Theory:

Service Worker

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

Things to note about Service Worker:

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

What can we do with Service Workers?

- You can dominate Network Traffic

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

- You can Cache

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

- You can manage Push Notifications

You can manage push notifications with Service Worker and show any information message to the user.

- You can Continue

Although Internet connection is broken, you can start any process with Background Sync of Service Worker.

What can’t we do with Service Workers?

You can’t access the Window

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

- You can’t work it on 80 Port

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

Service Worker Cycle

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

- Registration
- Installation
- Activation

Registration
To install a service worker, you need to register it in your main JavaScript code. Registration tells the browser where your service worker is located, and to start installing it in the background. Let's look at an example:
main.js

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

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

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

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

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

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

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

main.js

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

Installation

Once the browser registers a service worker, installation can be

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

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

service-worker.js

```
// Listen for install event, set callback
self.addEventListener('install', function(event) {
// Perform some task
});
```

Activation

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

When the new service worker activates, an activate event is triggered in the activating service worker. This event listener is a good place to clean up outdated caches (see the Offline Cookbook for an example).

service-worker.js

```
self.addEventListener('activate', function(event) {// Perform
some task });
```

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

CODE

Index.html

```
<!DOCTYPE html>
<html lang="en">
 <head>
 <meta charset="utf-8" />
 <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
<meta name="viewport" content="width=device-width,
initial-scale=1" />
 <meta name="theme-color" content="#121212" />
 <meta
 name="description"
 content="My College Portal - Access your portal with a fast,
installable Progressive Web App."
 />
 <link rel="apple-touch-icon"
href="%PUBLIC_URL%/icons/icon-192x192.png" />
 <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />

 <!-- SEO & Social Tags -->
 <meta property="og:title" content="My College Portal PWA" />
<meta property="og:description" content="Modern, installable,
offline-capable college portal." />
 <meta property="og:type" content="website" />
   <meta name="twitter:card" content="summary_large_image" />
 <title>My College Portal</title>
 </head>
 <body>
 <noscript>You need to enable JavaScript to run this
app.</noscript>
 <div id="root"></div>
 </body>
</html>
```

OUTPUT

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.	03
Name	Kartik bhat
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

EXPERIMENT 9

Name-kartik bhat

Class-D15A

Roll No-03

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

Theory:

Service Worker

Service Worker is a script that works on browser background without user interaction independently. Also, It resembles a proxy that works on the user side. With this script, you can track network traffic of the page, manage push notifications and develop “offline first” web applications with Cache API. Things to note about Service Worker: ● A service worker is a programmable network proxy that lets you control how network requests from your page are handled. ● Service workers only run over HTTPS. Because service workers can intercept network requests and modify responses, "man-in-the-middle" attacks could be very bad.

● The service worker becomes idle when not in use and restarts when it's next needed. You cannot rely on a global state persisting between events. If there is information that you need to persist and reuse across restarts, you can use IndexedDB databases.

● Service workers make extensive use of promises, so if you're new to promises, then you should stop reading this and check out Promises, an introduction.

Fetch Event

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

Of course, you can use many different methods but you can find in the following example a “cache first” and “network first” approach. In this example, if the request’s and current location’s origin are the same (Static content is requested.), this is called “cacheFirst” but if you request a targeted external URL, this is called “networkFirst”. ● CacheFirst - In this function, if the received request has cached before, the cached response is returned to the page. But if not, a new response requested from the network.

● NetworkFirst - In this function, firstly we can try getting an updated response from the network, if this process completed successfully,

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

Sync Event

Background Sync is a Web API that is used to delay a process until the Internet connection is stable. We can adapt this definition to the real world; there is an e mail client application that works on the browser and we want to send an email with this tool. Internet connection is broken while we are writing e-mail content
and we didn't realize it. When completing the writing, we click the send button. Here is a job for the Background Sync.
The following view shows the classical process of sending email to us. If the Internet Connection is broken, we can't send any content to Mail Server.

Push Event

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

CODE

```
const CACHE_NAME = 'college-portal-cache-v1';

const urlsToCache = [
  '/',

  '/index.html',

  '/manifest.json',

  '/icons/icon-192x192.png',

  '/icons/icon-512x512.png',

];
```

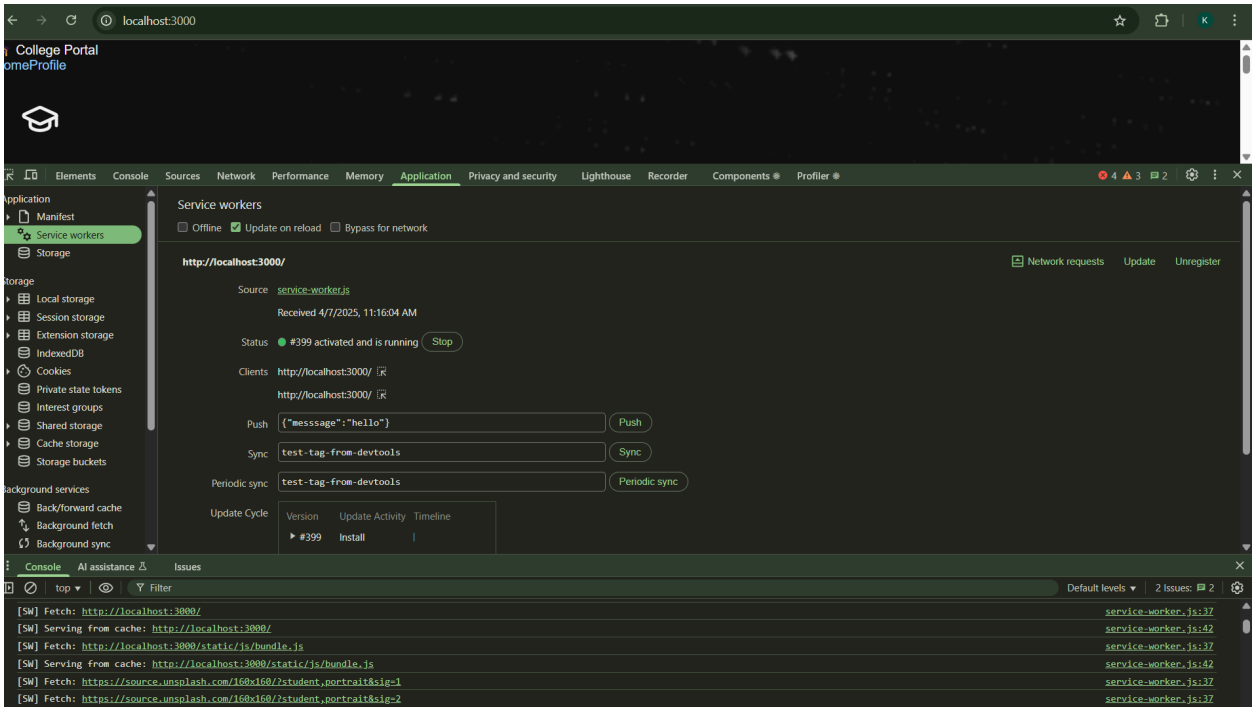
```
// ✅ INSTALL: Pre-cache
self.addEventListener('install', (event) => {
event.waitUntil(
 caches.open(CACHE_NAME).then((cache) =>
cache.addAll(urlsToCache))
 );
 self.skipWaiting();
 });

// ✅ ACTIVATE: Clear old caches
self.addEventListener('activate', (event) => {
event.waitUntil(
 caches.keys().then((keys) =>
 Promise.all(keys.map((key) => key !== CACHE_NAME &&
caches.delete(key)))
 )
 );
 self.clients.claim();
 });

// ✅ FETCH: Cache-first
self.addEventListener('fetch', (event) => {
 if (event.request.method !== 'GET') return;

 event.respondWith(
 caches.match(event.request).then((cached) => {
return (
```

```
 cached ||

 fetch(event.request)
 .then((response) => {

 return caches.open(CACHE_NAME).then((cache) => {

cache.put(event.request, response.clone()); return

response;

 });

 })

 .catch(() => new Response('Offline', { status: 503 })) );

 })

 );

});
// ✅ BACKGROUND SYNC

self.addEventListener('sync', (event) => {

 if (event.tag === 'sync-data') {

 event.waitUntil(

 fetch('/sync-endpoint', {

 method: 'POST',

 body: JSON.stringify({ message: 'Synced after reconnecting' }),

headers: { 'Content-Type': 'application/json' }, }).then((res)

=> console.log('[SW] Sync complete:', res)) ); }

 });


// ✅ PUSH NOTIFICATIONS

self.addEventListener('push', (event) => {

 let data = { title: 'New Message', message: 'You have a new

notification' };


 try {
```

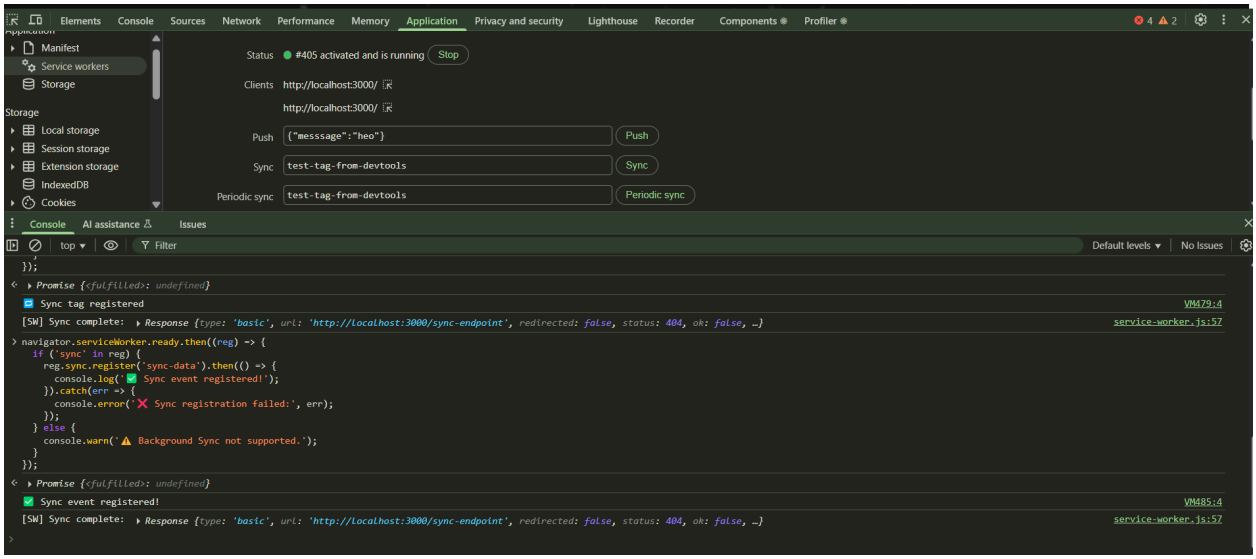
```
if (event.data) {
const json = JSON.parse(event.data.text());
data = { ...data, ...json };
}
} catch (e) {
console.warn('[SW] Invalid push JSON, using default'); }

const options = {
body: data.message,
icon: '/icons/icon-192x192.png',
badge: '/icons/icon-192x192.png',
};

event.waitUntil(
self.registration.showNotification(data.title, options)
);
});
```

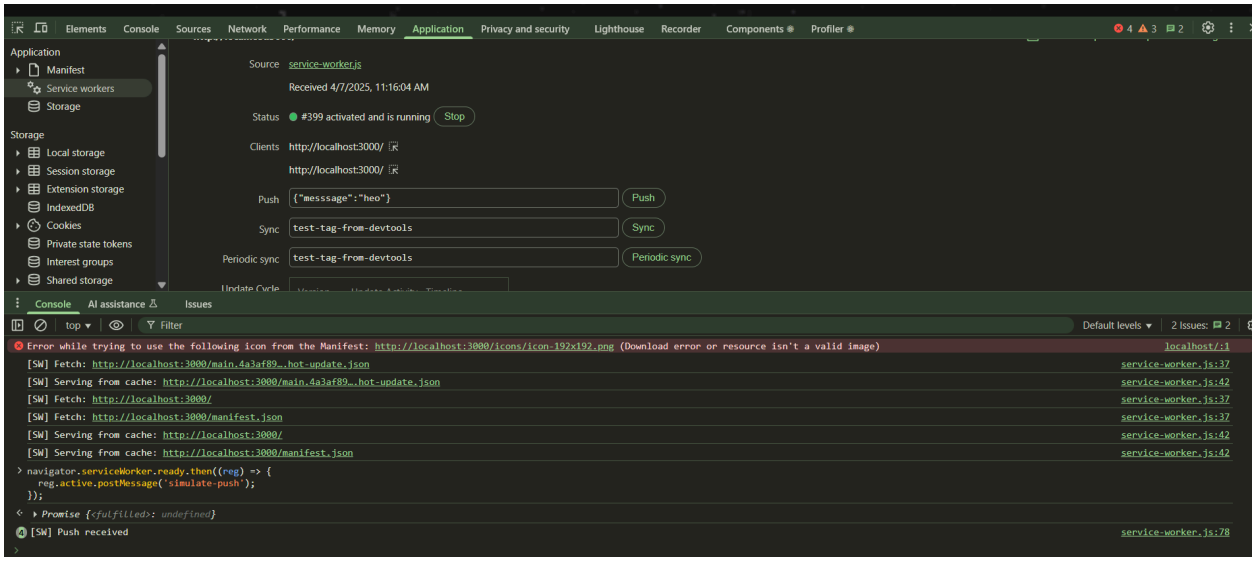
OUTPUT

Service worker activated
Fetch Event


localhost:3000
College Portal
omeProfile
Elements
Console
Sources
Network
Performance
Memory
Application
Privacy and security
Lighthouse
Recorder
Components
Profiler
Application
Manifest
Service workers
Storage
Storage
Local storage
Session storage
Extension storage
IndexedDB
Cookies
Private state tokens
Interest groups
Shared storage
Cache storage
Storage buckets
Background services
Back/forward cache
Background fetch
Background sync
Service workers
Offline
Update on reload
Bypass for network
http://localhost:3000/
Network requests
Update
Unregister
Source
service-worker.js
Received 4/7/2025, 11:16:04 AM
Status
#399 activated and is running
Stop
Clients
http://localhost:3000/
http://localhost:3000/
Push
{"messsage":"hello"}
Push
Sync
test-tag-from-devtools
Sync
Periodic sync
test-tag-from-devtools
Periodic sync
Update Cycle
Version
Update Activity
Timeline
#399
Install
Console
AI assistance
Issues
top
Filter
Default levels
2 Issues:
[SW] Fetch: http://localhost:3000/
service-worker.js:37
[SW] Serving from cache: http://localhost:3000/
service-worker.js:42
[SW] Fetch: http://localhost:3000/static/js/bundle.js
service-worker.js:37
[SW] Serving from cache: http://localhost:3000/static/js/bundle.js
service-worker.js:42
[SW] Fetch: https://source.unsplash.com/160x160/?student,portrait&sig=1
service-worker.js:37
[SW] Fetch: https://source.unsplash.com/160x160/?student,portrait&sig=2
service-worker.js:37

Sync Event


Elements
Console
Sources
Network
Performance
Memory
Application
Privacy and security
Lighthouse
Recorder
Components
Profiler
Manifest
Service workers
Storage
Storage
Local storage
Session storage
Extension storage
IndexedDB
Cookies
Status
#405 activated and is running
Stop
Clients
http://localhost:3000/
http://localhost:3000/
Push
{"messsage":"heo"}
Push
Sync
test-tag-from-devtools
Sync
Periodic sync
test-tag-from-devtools
Periodic sync
Console
AI assistance
Issues
top
Filter
Default levels
No Issues
});
Promise {<fulfilled>: undefined}
Sync tag registered
VM479:4
[SW] Sync complete:
Response {type: 'basic', url: 'http://localhost:3000/sync-endpoint', redirected: false, status: 404, ok: false, …}
service-worker.js:57
navigator.serviceWorker.ready.then((reg) => {
if ('sync' in reg) {
reg.sync.register('sync-data').then(() => {
console.log('✅ Sync event registered!');
}).catch(err => {
console.error('❌ Sync registration failed:', err);
});
} else {
console.warn('⚠️ Background Sync not supported.');
}
});
Promise {<fulfilled>: undefined}
Sync event registered!
VM485:4
[SW] Sync complete:
Response {type: 'basic', url: 'http://localhost:3000/sync-endpoint', redirected: false, status: 404, ok: false, …}
service-worker.js:57


Elements
Console
Sources
Network
Performance
Memory
Application
Privacy and security
Lighthouse
Recorder
Components
Profiler
Application
Manifest
Service workers
Storage
Storage
Local storage
Session storage
Extension storage
IndexedDB
Cookies
Private state tokens
Interest groups
Shared storage
Source service-worker.js
Received 4/7/2025, 11:16:04 AM
Status #399 activated and is running Stop
Clients http://localhost:3000/
http://localhost:3000/
Push {"messsage":"heo"} Push
Sync test-tag-from-devtools Sync
Periodic sync test-tag-from-devtools Periodic sync
Update Cycle
Console
AI assistance
Issues
top
Filter
Default levels
2 Issues: 2
Error while trying to use the following icon from the Manifest: http://localhost:3000/icons/icon-192x192.png (Download error or resource isn't a valid image) localhost/:1
[SW] Fetch: http://localhost:3000/main.4a3af89….hot-update.json service-worker.js:37
[SW] Serving from cache: http://localhost:3000/main.4a3af89….hot-update.json service-worker.js:42
[SW] Fetch: http://localhost:3000/ service-worker.js:37
[SW] Fetch: http://localhost:3000/manifest.json service-worker.js:37
[SW] Serving from cache: http://localhost:3000/ service-worker.js:42
[SW] Serving from cache: http://localhost:3000/manifest.json service-worker.js:42
navigator.serviceWorker.ready.then((reg) => {
reg.active.postMessage('simulate-push');
});
Promise {<fulfilled>: undefined}
[SW] Push received service-worker.js:78

push event

MAD & PWA Lab

Journal

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	03
Name	Kartik bhat
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

EXPERIMENT NO 10

Name-kartik bhat

Class-D15A

Roll No-03

Aim:

To study and implement deployment of Ecommerce PWA to GitHub

Pages. Theory:

GitHub Pages

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

GitHub Pages provides the following key features:

1. Blogging with Jekyll
2. Custom URL
3. Automatic Page Generator

Reasons for favoring this over Firebase:

1. Free to use
2. Right out of github
3. Quick to set up

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

Pros

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

Cons

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

spotty. Firebase

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

Some of the features offered by Firebase are:

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

Reasons for favoring over GitHub Pages:

1. Realtime backend made easy
2. Fast and responsive

Instacart, 9GAG, and Twitch are some of the popular companies that use Firebase

Firebase has a broader approval, being mentioned in 1215 company stacks & 4651 developers stacks

Pros

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

Cons

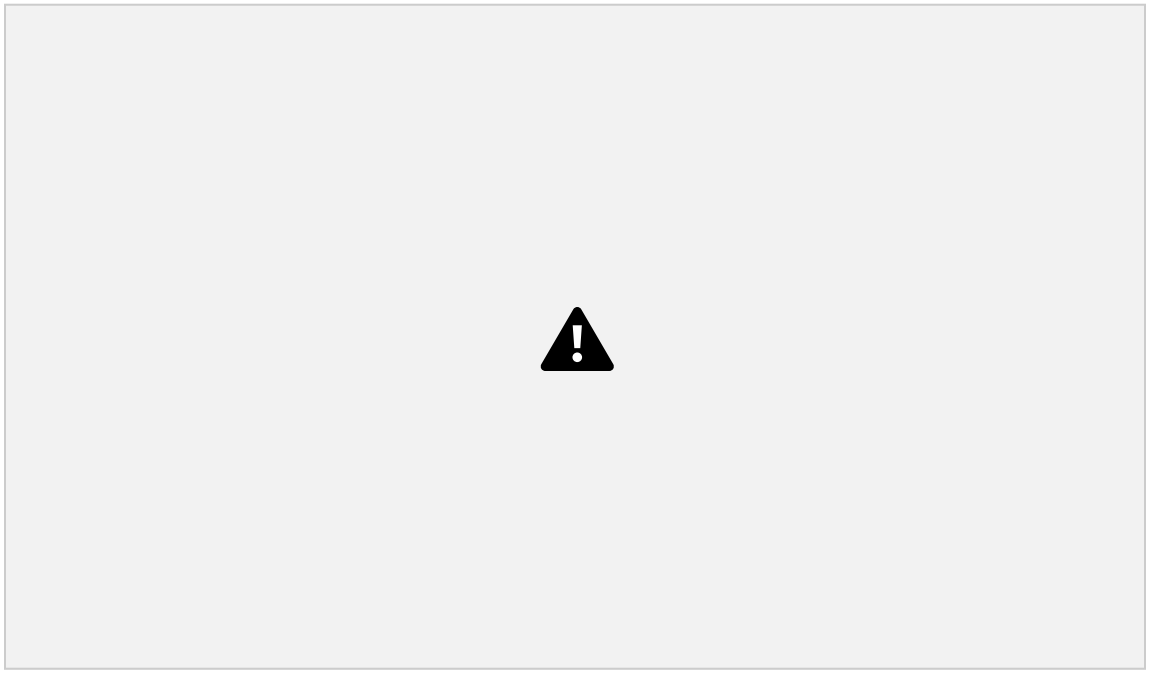
1. Only 10 GB of data transfer is allowed per month. But this

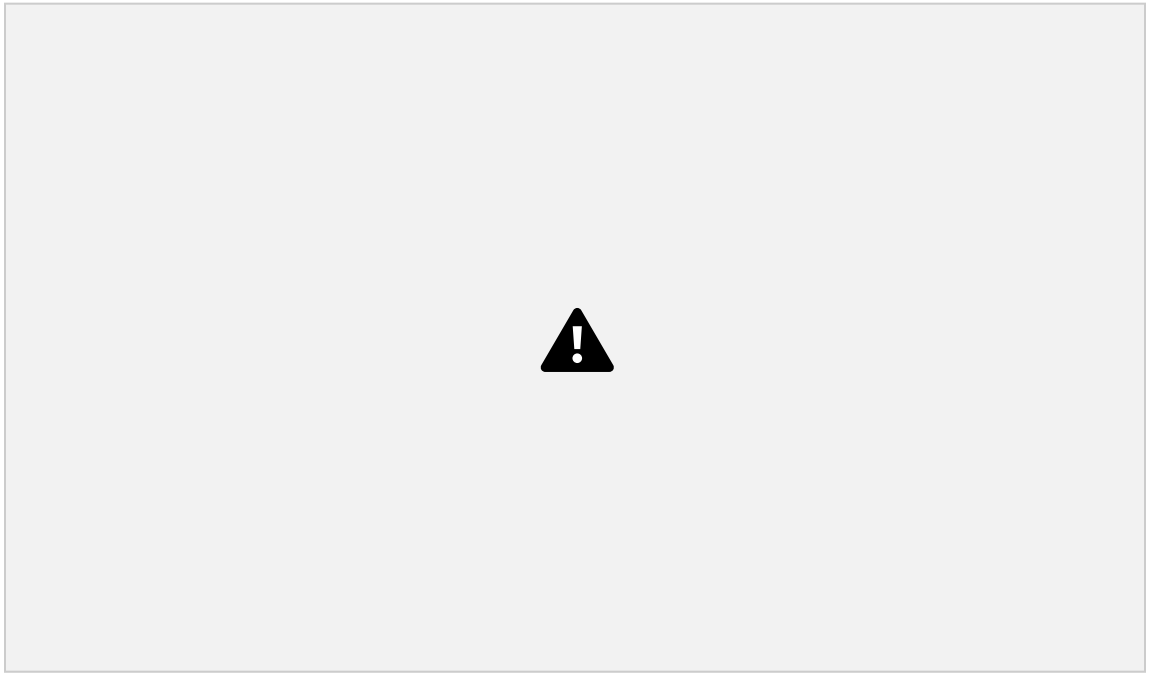
is not really a big problem, if you use a CDN or AMP.
2. Command-line interface only.
3. No in-built support for any static site generator.

Link to Github Repository

https://github.com/kartikbhat78/my-react-app

OUTPUT-





MAD & PWA Lab

Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	03
Name	Kartik bhat
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	

Experiment 11

Name-Kartik Bhat
Class-D15A
Roll no-03

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

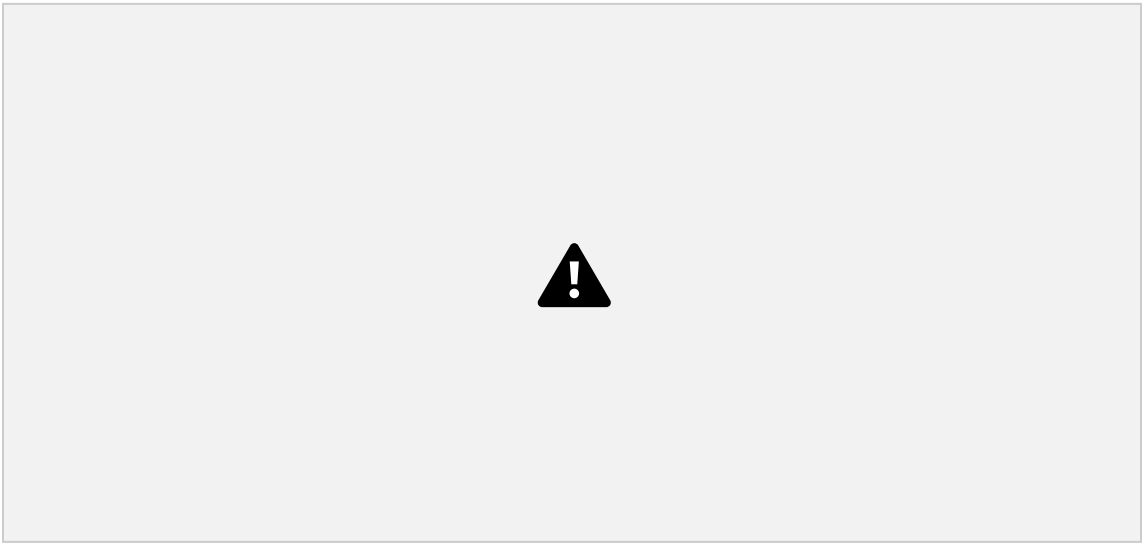
Google Lighthouse has the option of running the Audit for Desktop as well as mobile version of your page(s). The top metrics that will be measured in the Audit are:

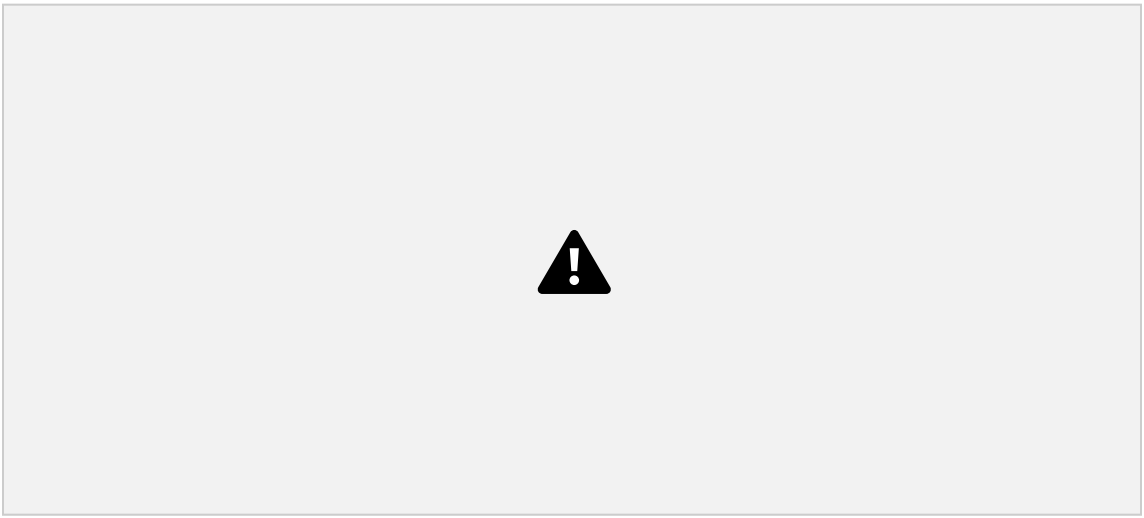
1. Performance: This score is an aggregation of how the page fared in aspects such as (but not limited to) loading speed, time taken for loading for basic frame(s), displaying meaningful content to the user, etc. To a layman, this score is indicative of how decently the site performs, with a score of 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 applicaϴons are moving towards the PWA paradigm, where the objecϴve is to make the applicaϴon behave as close as possible to naϴve mobile applicaϴons. Scoring points are based on the Baseline PWA checklist laid down by Google which includes Service Worker implementaϴon(s), viewport handling, offline funcϴonality, 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-' aΣributes like aria-required, audio capϴons, buΣon 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 <secϴon>, <arϴcle>, etc.), you get a 0 on that score. The aggregate of these scores is your Accessibility metric score. 4. Best PracϴcesAs any developer would know, there are a number of pracϴces that have been deemed ‘best’ based on empirical data. This metric is an aggregaϴon of many such points, including but not limited to:Use of HTTPS Avoiding the use of deprecated code elements like tags, direcϴves, libraries, etc.
Password input with paste-into disabled
Geo-LocaϴonAlerts on load, etc.

OUTPUT:





CONCLUSION

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