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 **Om Gaikwad** 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 **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: Upwork Clone and e-comm site **Roll no. 16**

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	16/1	23/1	15
2.	To design Flutter UI by including common widgets.	LO2	23/1	30/1	15
3.	To include icons, images, fonts in Flutter app	LO2	30/1	6/2	15
4.	To create an interactive Form using form widget	LO2	6/2	13/2	15
5.	To apply navigation, routing and gestures in Flutter App	LO2	13/2	20/2	13
6.	To Connect Flutter UI with fireBase database	LO3	20/2	5/3	13
7.	To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”.	LO4	5/3	12/3	15
8.	To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA	LO5	12/3	19/3	15
9.	To implement Service worker events like fetch, sync and push for E-commerce PWA	LO5	19/3	26/3	15
10.	To study and implement deployment of Ecommerce PWA to GitHub Pages.	LO5	26/3	2/4	15
11.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.	LO6	5/3	12/3	15
12.	Assignment-1	LO1,LO2 ,LO3	2/2	5/2	5
13.	Assignment-2	LO4,LO5 ,LO6	19/3	21/3	3

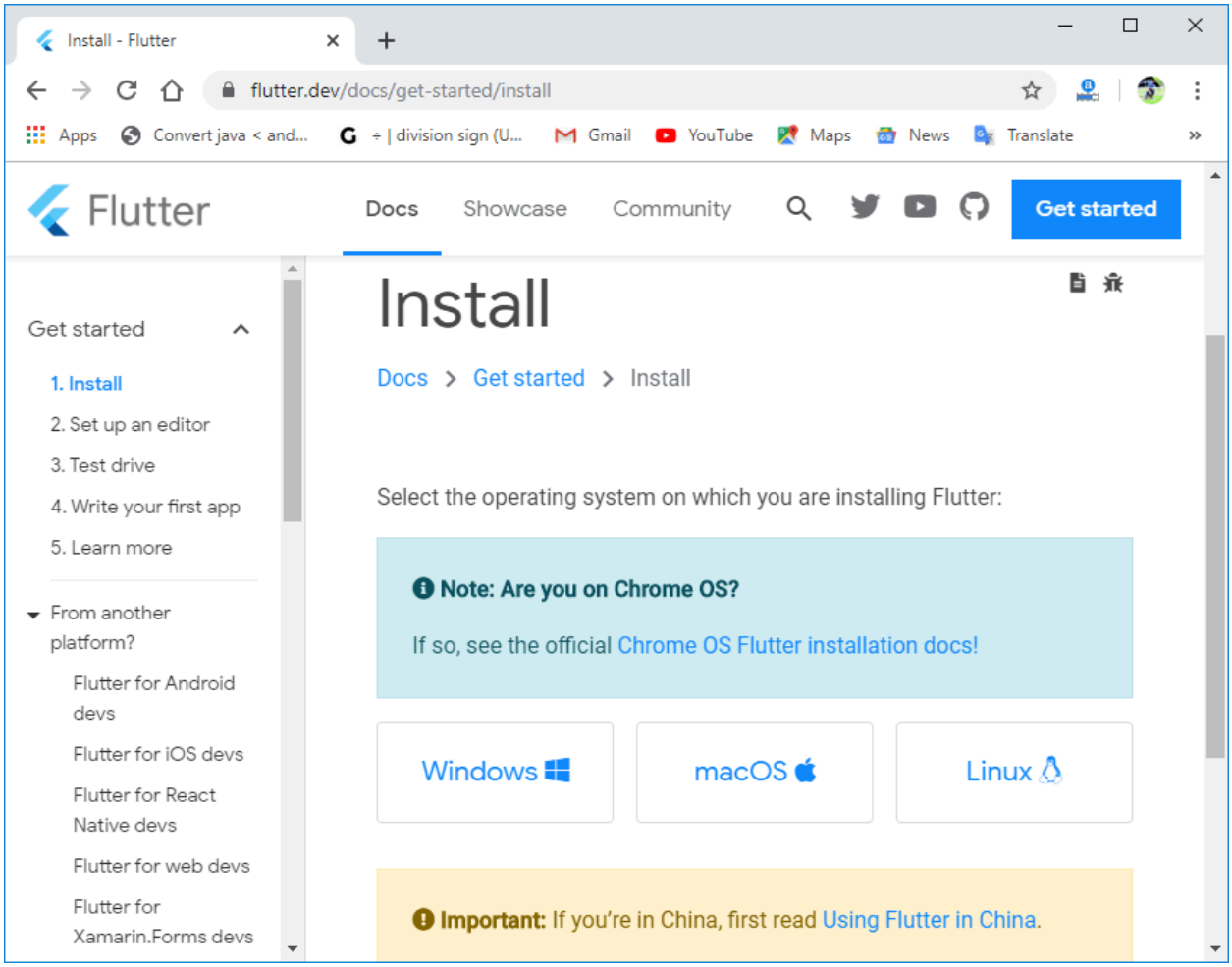
MAD & PWA Lab
Journal

Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	16
Name	Om Gaikwad
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	15

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.

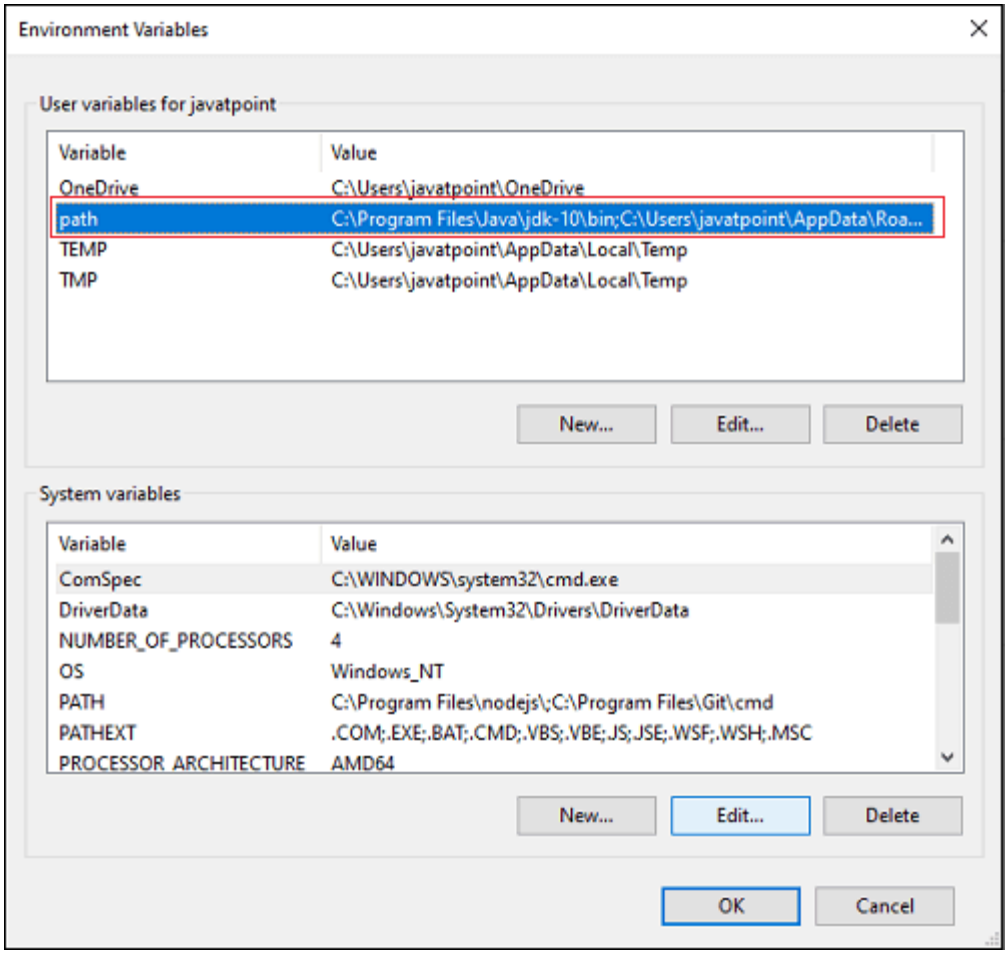


Step 2: Next, to download the latest Flutter SDK, click on the Windows **icon**. Here, you will find the download link for SDK.

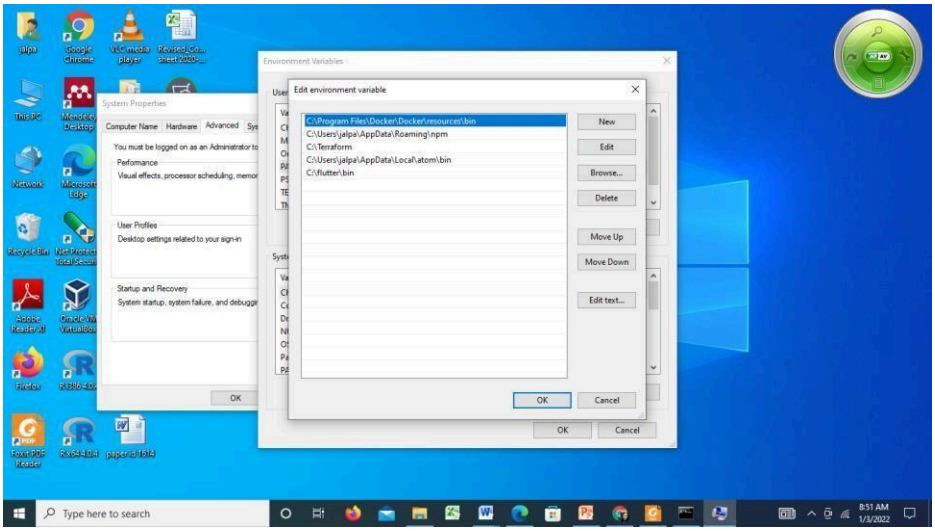
Step 3: When your download is complete, extract the **zip** file and place it in the desired installation folder or location, for example, C: /Flutter.

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.

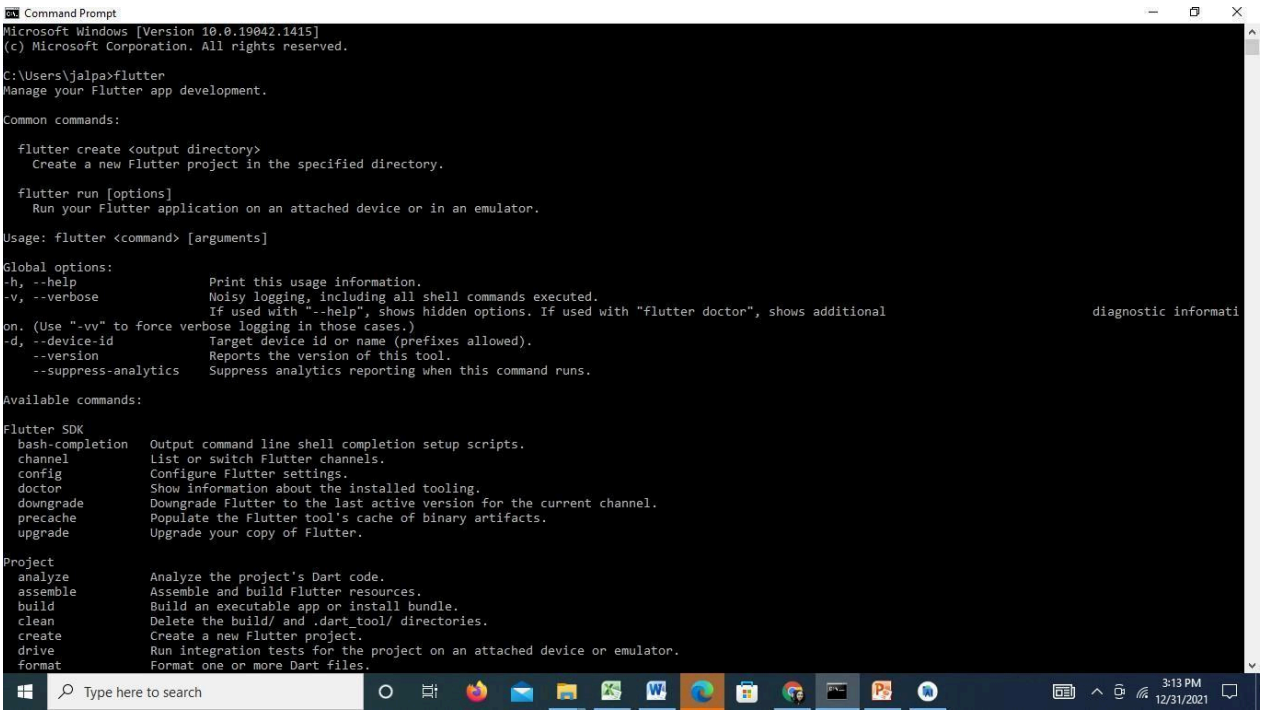


Step 4.2: Now, select path -> click on edit. The following screen appears

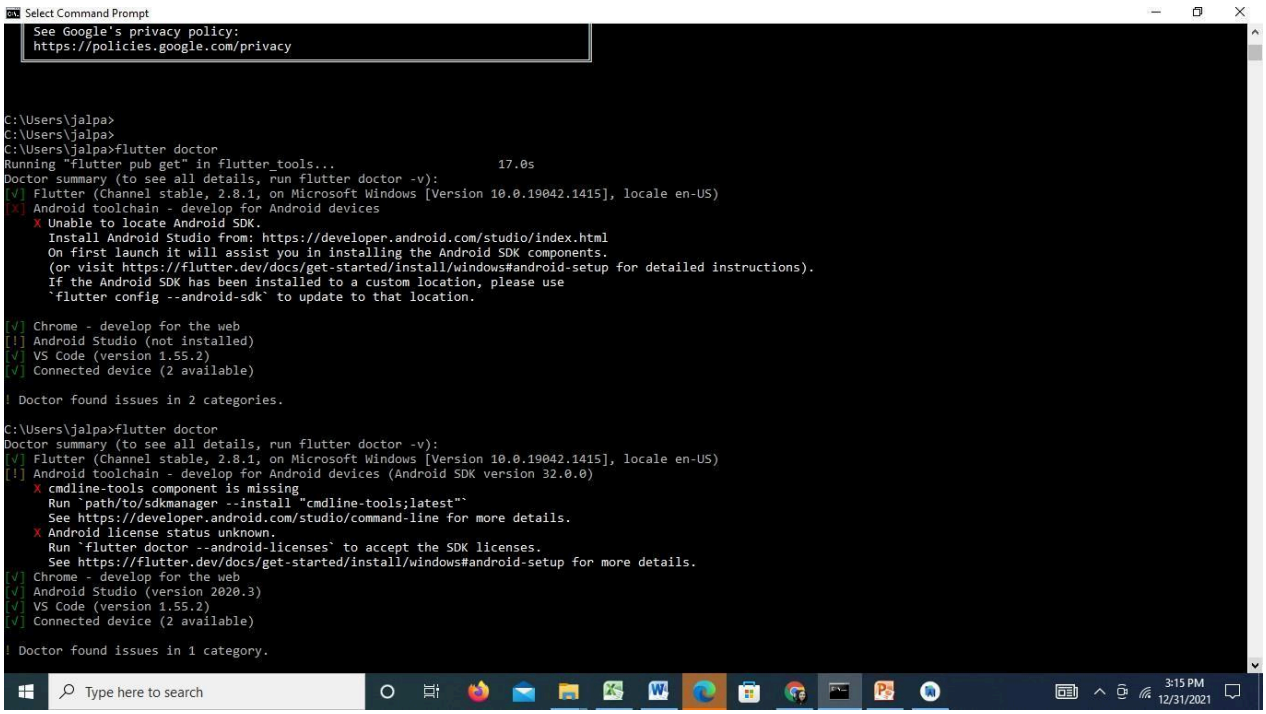


Step 4.3: In the above window, click on New->write path of Flutter bin folder in variable value -> ok -> ok -> ok.

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.



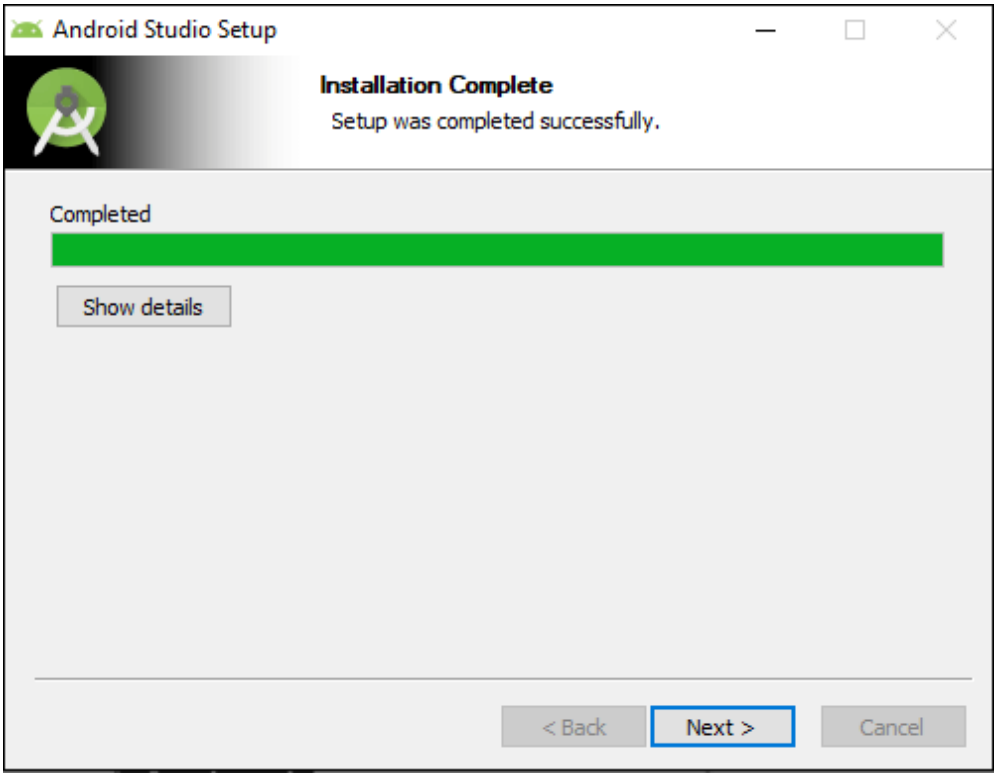
Step 6: When you run the above command, it will analyze the system and show its report, as shown in the below image. Here, you will find the details of all missing tools, which required to run Flutter as well as the development tools that are available but not connected with the device.

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.

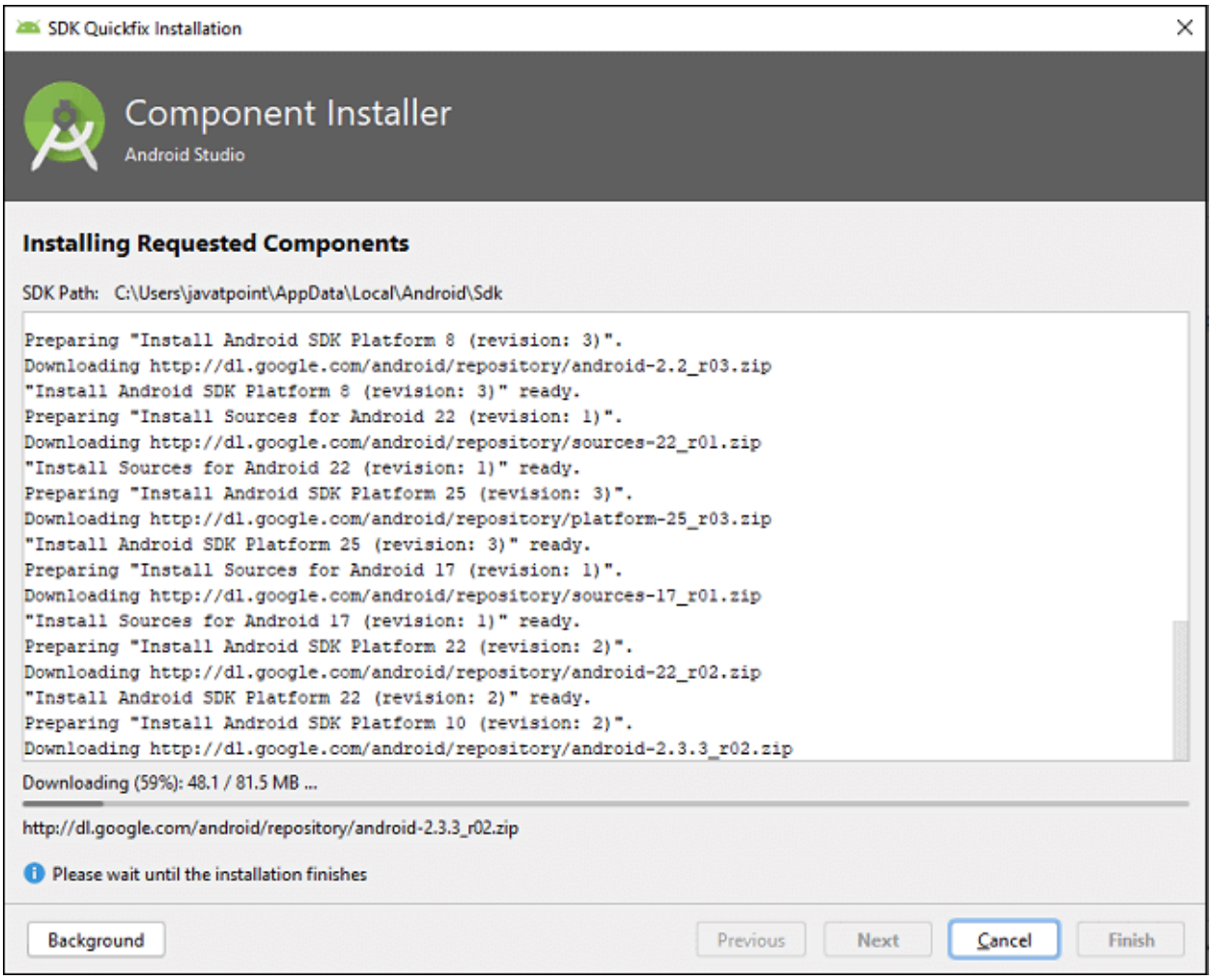
Step 7.1: Download the latest Android Studio executable or zip file from the official site.

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

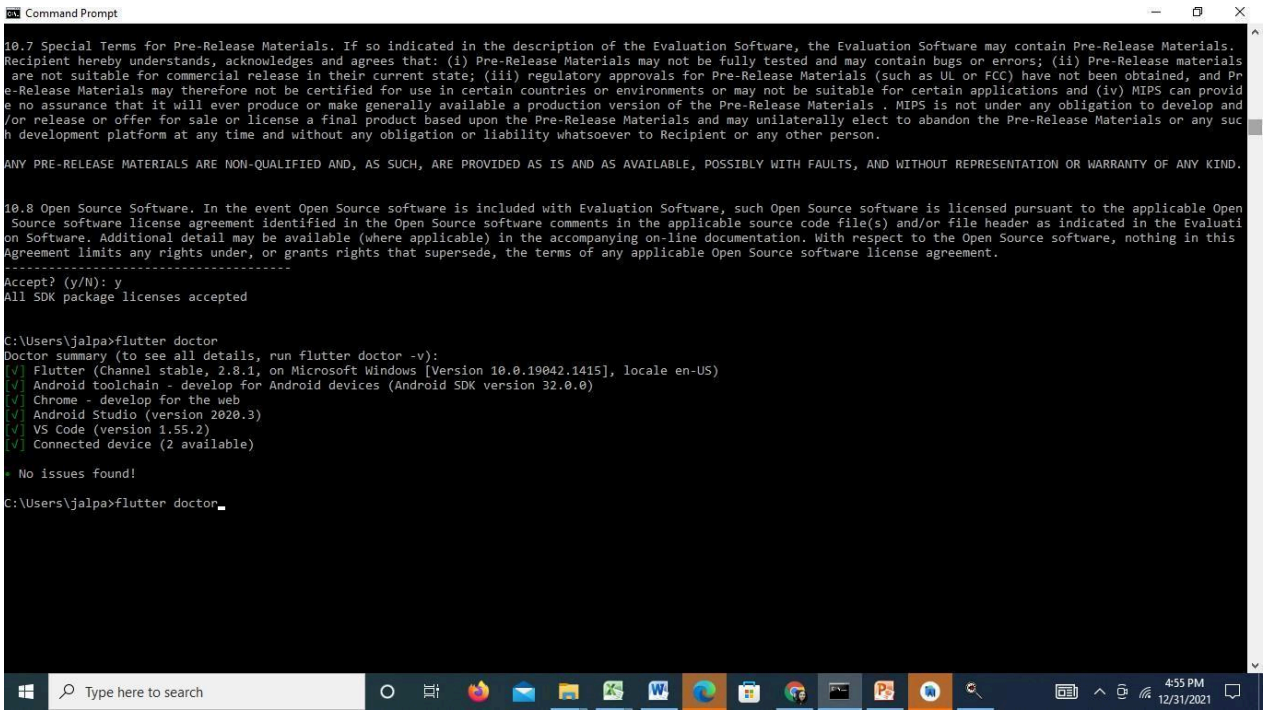
Step 7.3: Follow the steps of the installation wizard. Once the installation wizard completes, you will get the following screen.



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.

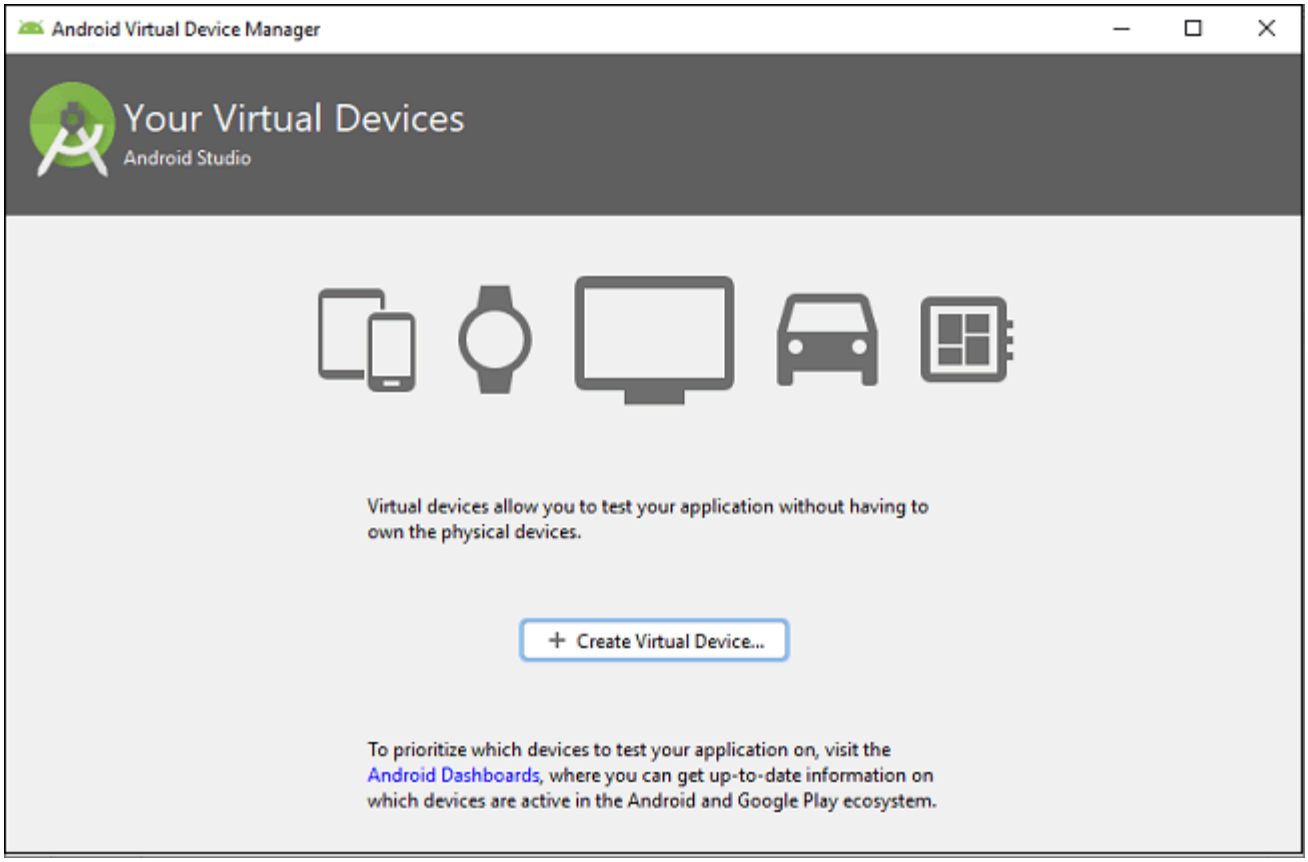


Step 7.5 run the $ **flutter doctor** command and Run flutter doctor --android-licenses command.



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

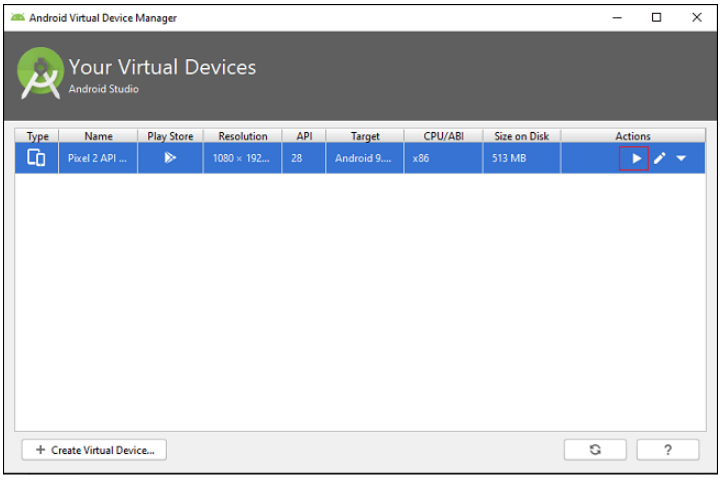
Step 8.1: To set an Android emulator, go to Android Studio > Tools > Android > AVD Manager and select Create Virtual Device. Or, go to Help->Find Action->Type Emulator in the search box. You will get the following screen.



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.

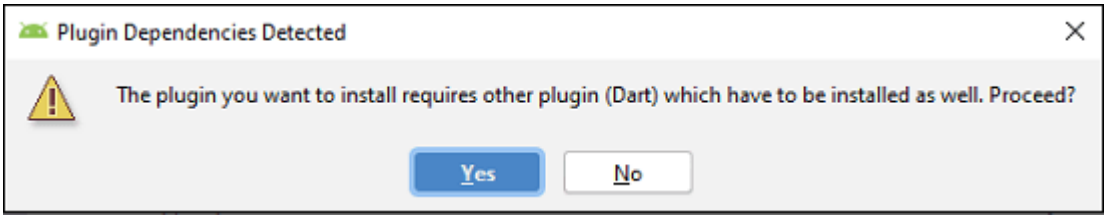


Step 8.5: Last, click on the icon pointed into the red color rectangle. The Android emulator displayed as below screen.

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

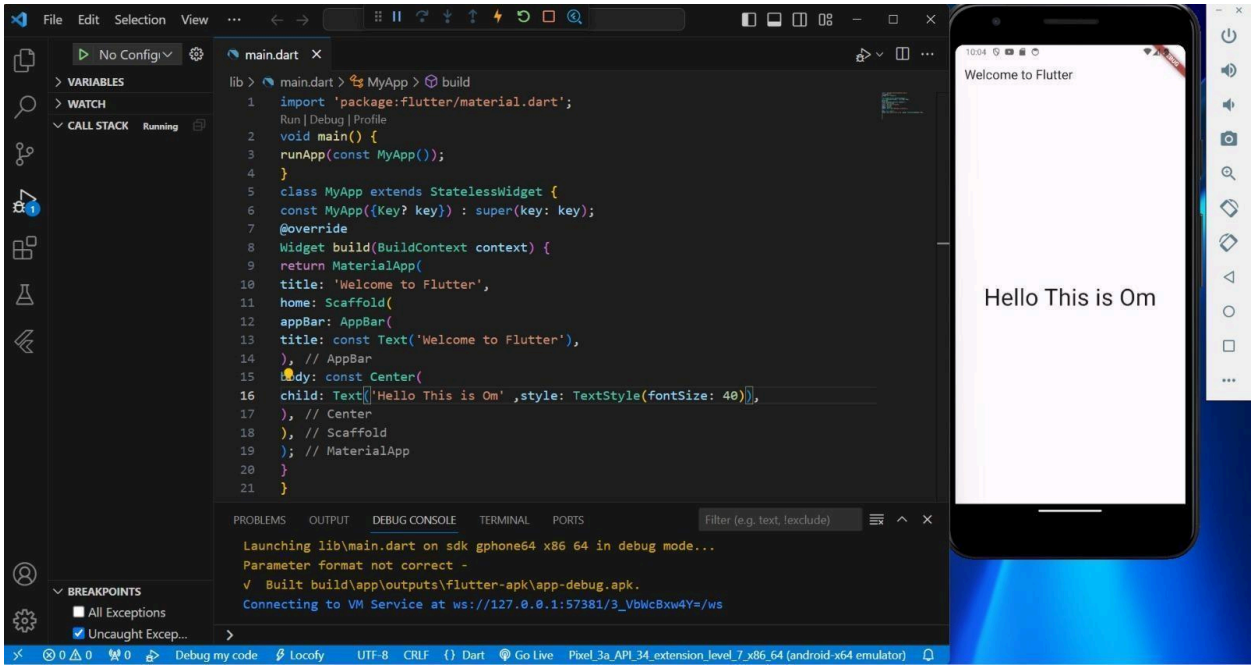
Step 9.1: Open the Android Studio and then go to File->Settings->Plugins.

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



Step 9.3: Restart the Android Studio.

Om Gaikwad D15A
16



import 'package:flutter/material.dart'; void main() {

runApp(const MyApp()); }

class MyApp extends StatelessWidget { const MyApp({Key? key}) : super(key: key); @override Widget build(BuildContext context) {

return MaterialApp(
title: 'Welcome to Flutter', home: Scaffold(

appBar: AppBar(
title: const Text('Welcome to Flutter'),

),
body: const Center(

child: Text('Hello This is Om'),

),

),

);

```
 }
}
```

MAD & PWA Lab
Journal

Experiment No.	02
Experiment Title.	To design Flutter UI by including common widgets.
Roll No.	16
Name	Om Gaikwad
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

MAD PWA
EXP2

Aim: To design flutter by including common widgets

Theory:-

Introduction to the App:

- The splash screen is often the first visual element users encounter when launching the app. It provides a quick introduction to the application, setting the tone for the user experience.

Branding and Logo Display:

- Incorporating the app's logo on the splash screen reinforces the brand identity. In the provided experiment, the Upwork logo is prominently displayed, creating brand recognition and familiarity.

Welcome Message:

- Including a welcome message, such as "Welcome to Upwork," on the splash screen adds a personal touch. It can create a sense of friendliness and make users feel more connected to the app.

Duration and Transition:

- The splash screen is typically displayed for a short duration, usually a few seconds. In this experiment, a `Timer` is used to delay the transition to the login page for 5 seconds, allowing users to briefly engage with the welcoming content.

Navigation to Main Content:

- After the designated duration, the splash screen transitions to the login page, where users can access the main functionality of the application. This seamless transition ensures a smooth and uninterrupted user flow.

Code:-

Splash_screen.dart

```
// lib/screens/splash_screen.dart
import 'dart:async';
```

```
import '../screens/login/login_page.dart';

class SplashScreen extends StatefulWidget {
@override
_SplashScreenState createState() => _SplashScreenState();
}

class _SplashScreenState extends State<SplashScreen> {
@override
void initState() {
super.initState();
// Delay for 5 seconds and then navigate to the login page
Timer(Duration(seconds: 5), () {
Navigator.pushReplacementNamed(context, '/login');
});
}

@override
Widget build(BuildContext context) {
return Scaffold(
body: Center(
child: Column(
mainAxisAlignment: MainAxisAlignment.center,
children: [
Image.asset(
'images/2021-upwork-new-logo-design.png', // Replace with the actual path to your Upwork logo
height: 150.0,
),
SizedBox(height:
16.0), Text(
'Welcome to Upwork',
style: TextStyle(fontSize: 20.0, fontWeight: FontWeight.bold),
),
],
),
),
);
}
}
```

Main.dart :

```
// lib/main.dart
```

```
import 'screens/login/login_page.dart';
import 'screens/login/signup_page.dart';
import 'screens/jobs/jobs_page.dart';
import 'screens/splash_screen.dart';

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

class MyApp extends StatelessWidget {
@override
Widget build(BuildContext context) {
return MaterialApp(
title: 'Upwork',
theme: ThemeData(
// primarySwatch: Colors.blue,
),
initialRoute: '/',
routes: {
'/': (context) => SplashScreen(),
'/login': (context) => LoginPage(),
'/signup': (context) => SignupPage(),
'/jobs': (context) => JobsPage(),
},
);
}
}
```


DEBUG
upwork
Welcome to Upwork

MAD & PWA Lab
Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	16
Name	Om Gaikwad
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

MAD PWA
EXP 3

AIM:-

To include icons, images, fonts in Flutter App.

Widgets and Components:

Scaffold:
- The `Scaffold` widget provides the basic structure for the page, including the app bar, drawer, and body.

AppBar:
- The `AppBar` widget represents the top app bar that typically contains the page title, in this case, "Jobs."

Drawer:
- The `Drawer` widget creates a side menu that can be accessed by swiping from the left or tapping a menu icon. It contains a `UserAccountsDrawerHeader` for user information and a `ListTile` for navigation.

UserAccountsDrawerHeader:
- This widget displays user account information, including the account name, email, and a profile picture. It is often used in combination with the `Drawer` widget.

ListTile:
- The `ListTile` widget represents an item in a list. In this case, it is used for the "Profile" item in the drawer. It is configured to navigate to the `ProfilePage` when tapped.

Column:
- The `Column` widget is used to arrange its children in a vertical column. It is employed to structure the layout, grouping the logo, search bar, and job listings vertically.

Image.asset:
- The `Image.asset` widget displays an image loaded from the asset bundle. It is used to show the Upwork logo at the top of the screen. Make sure to replace the placeholder image path with the actual path to the Upwork logo.

TextField:
- The `TextField` widget is an input field for text. It is used to

implement a search bar where users can input keywords to search for jobs.

ListView.builder:

- The `ListView.builder` widget is used to create a scrollable list of job listings. It efficiently creates and recycles widgets as the user scrolls

through the list.

ListTile in ListView.builder:

- Each job listing is represented by a `ListTile` within the `ListView.builder`. It displays the job title, company name, and potentially more details.

Container:

- The `Container` widget is used to wrap the row of elevated buttons (Jobs, Messages, Contracts, Alerts) at the bottom of the screen. It provides a structured layout for these buttons.

ElevatedButton:

- The `ElevatedButton` widget is used to create buttons with a raised appearance. Each button corresponds to a specific functionality (Jobs, Messages, Contracts, Alerts).

Icon:

- The `Icon` widget is used to display icons within the buttons. Icons provide a visual representation of each button's purpose. For example, the search bar has a search icon (`Icons.search`), and each elevated button has an associated icon.

Theory:

- **User Interaction:**
  - The `JobsPage` provides a user-friendly interface for browsing job listings on the Upwork platform. Users can utilize the search bar to find specific jobs and navigate through the list.
- **Navigation:**
  - The drawer allows users to access additional features such as viewing their profile (`ProfilePage`). The bottom row of buttons serves as a quick navigation bar to switch between different sections of the app.
- **Branding:**
  - The Upwork logo at the top reinforces brand identity, creating a sense of familiarity for users.
- **Visual Hierarchy:**
  - The layout follows a logical visual hierarchy, with the Upwork logo at the top, followed by the search bar, job listings, and finally, the navigation buttons at the bottom.
- **Consistency:**
  - The use of consistent colors, fonts, and the Upwork logo contributes to a cohesive and professional appearance.
- **Interactivity:**
  - Buttons and icons are interactive elements that provide feedback to users, enhancing the overall user experience.
- **Efficiency:**
  - The use of `ListView.builder` ensures efficient rendering of job listings, optimizing performance even for a large number of items.

jobs_page.dart :

```
// lib/screens/jobs/jobs_page.dart
import 'package:flutter/material.dart';
import '../profile/profile_page.dart';

class JobsPage extends StatelessWidget {
@override
Widget build(BuildContext context) {
return Scaffold(
appBar: AppBar(
title: Text('Jobs'),
),
drawer:
Drawer( child:
ListView(
padding: EdgeInsets.zero,
children: [
UserAccountsDrawerHeader(
accountName: Text('John Doe'), // Replace with actual name
accountEmail: Text('john.doe@example.com'), // Replace with actual email
currentAccountPicture: CircleAvatar(
// Replace with an actual profile picture
backgroundImage: AssetImage('images/profile_picture.jpg'),
),
),
ListTile(
title: Text('Profile'),
onTap: () {
Navigator.push(
context,
MaterialPageRoute(builder: (context) => ProfilePage()),
);
},
),
// Add more drawer items as needed
],
),
),
body: Column(
children: [
Padding(
padding: const EdgeInsets.all(16.0),
child: Column(
children: [
Image.asset(
'images/2021-upwork-new-logo-design.png', // Replace with the actual path to your Upwork logo
height: 50.0,
),
SizedBox(height:
16.0), TextField(
```

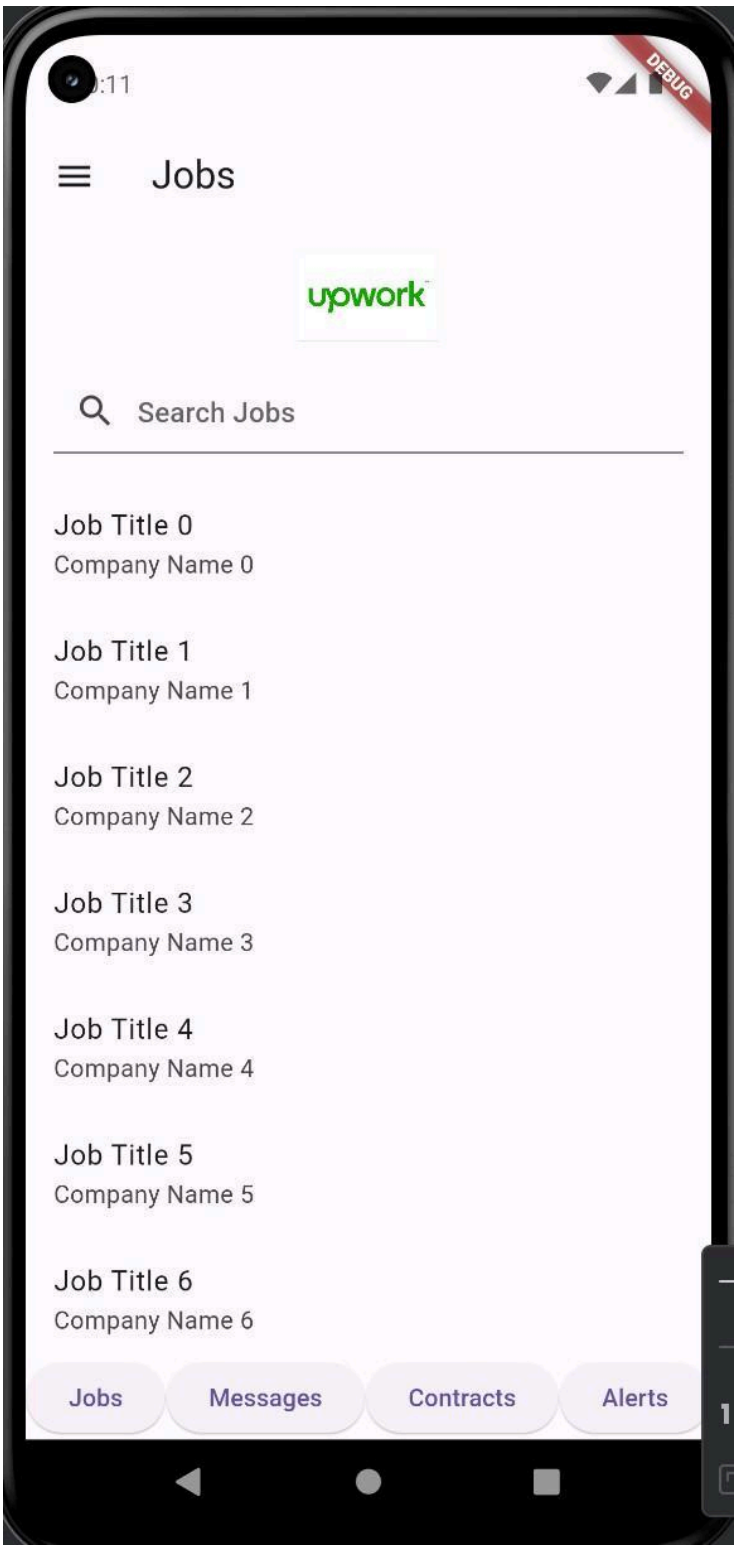
```
decoration: InputDecoration(
hintText: 'Search Jobs',
prefixIcon: Icon(Icons.search),
```

```
),
),
],
),
),
// Job listings can go here
// Use ListView.builder or other widgets to display job listings
Expanded(
child: ListView.builder(
itemCount: 10, // Replace with your actual job listings
itemBuilder: (context, index) {
// Replace the ListTile with your job listing widget
return ListTile(
title: Text('Job Title $index'),
subtitle: Text('Company Name $index'),
// Add more details as needed
);
},
),
),
Container(
// color: Colors.blue, // Background color for the bar
child: Row(
mainAxisAlignment: MainAxisAlignment.spaceAround,
children: [
ElevatedButton
( onPressed: ()
{
// Implement jobs button functionality
},
child: Text('Jobs'),
),
ElevatedButton
( onPressed: ()
{
// Implement messages button functionality
},
child: Text('Messages'),
),
ElevatedButton
( onPressed: ()
{
// Implement contracts button functionality
},
child: Text('Contracts'),
),
ElevatedButton
( onPressed: ()
{
// Implement alerts button functionality
},
```

```
child: Text('Alerts'),
),
],
),
),
],
```

main.dart :

```
// lib/main.dart
import 'package:flutter/material.dart';
import 'screens/login/login_page.dart';
import 'screens/login/signup_page.dart';
import 'screens/jobs/jobs_page.dart';
import 'screens/splash_screen.dart';

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

class MyApp extends StatelessWidget {
@override
Widget build(BuildContext context) {
return MaterialApp(
title: 'Upwork',
theme: ThemeData(
// primarySwatch: Colors.blue,
),
initialRoute: '/',
routes: {
'/': (context) => SplashScreen(),
'/login': (context) => LoginPage(),
'/signup': (context) => SignupPage(),
'/jobs': (context) => JobsPage(),
},
);
}
}
```



Conclusion:- The experiment successfully enhanced the Flutter UI by integrating icons, images, and custom fonts, contributing to a visually appealing and engaging user interface

MAD & PWA Lab
Journal

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget
Roll No.	16
Name	Om Gaikwad
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

MAD and PWA Lab

Name: Om Gaikwad **Class: D15A**
Roll no:16

EXP 4

Aim : To create an interactive Form using form widget

Theory:

Form Widgets:

Form widgets are essential components of interactive forms, offering a range of input elements such as text fields, checkboxes, radio buttons, dropdown menus, and more. These widgets empower developers to design forms that cater to specific data input requirements. The flexibility of form widgets allows for the creation of dynamic and user-friendly interfaces, ensuring that the form adapts to the user's needs.

Form Inputs:

Text Fields:

Purpose: Allow users to input general text information.
Attributes: May include specifications such as maximum length, placeholder text, and input type (e.g., email, password).

Checkboxes:

Purpose: Enable users to make multiple selections from a list of

options. Attributes: Each checkbox typically represents a distinct option, and users can choose multiple checkboxes simultaneously.

Radio Buttons:

Purpose: Provide users with exclusive choices within a group.
Attributes: Users can select only one option from the group, making radio buttons suitable for mutually exclusive selections.

Dropdown Menus:

Purpose: Offer a space-efficient way to present a list of options for selection.
Attributes: Users click on a dropdown menu to reveal a list of choices, selecting one option from the list.

Textareas:

Purpose: Allow users to input multiline text, suitable for longer responses or comments.
Attributes: Can include settings for the number of rows and columns to determine the size of the textarea.

Date Pickers:

Purpose: Facilitate the selection of dates.
Attributes: Users can choose a specific date from a calendar interface, helping to ensure accurate date input.

File Upload:

Purpose: Enable users to submit files (e.g., images, documents).
Attributes: May include file type restrictions, maximum file size, and a browse button for users to locate and upload files from their device.

Code

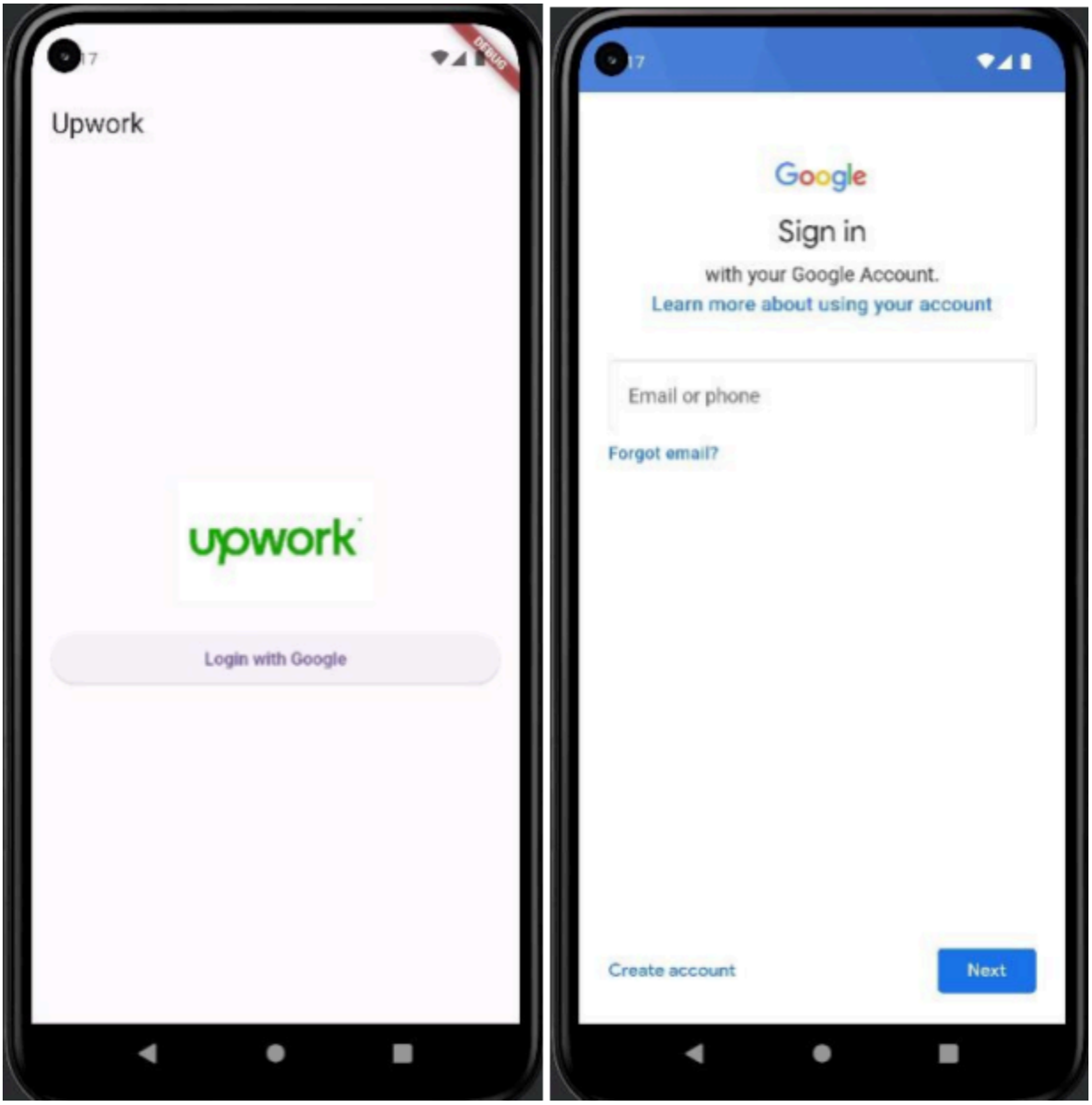
login_page.dart

```
// lib/screens/login/login_page.dart
import
'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:google_sign_in/google_sign_in.dart';
import '../jobs/jobs_page.dart'; // Import the JobsPage if not already imported
import 'signup_page.dart'; // Import the SignupPage if not already imported

class LoginPage extends StatelessWidget {
@override
Widget build(BuildContext context) {
return Scaffold(
appBar: AppBar(
title: Text('Upwork'),
),
body: Padding(
padding:
EdgeInsets.all(16.0), child:
Column(
mainAxisAlignment: MainAxisAlignment.center,
crossAxisAlignment: CrossAxisAlignment.stretch,
children: [
Image.asset(
'images/2021-upwork-new-logo-design.png',
height: 100.0,
),
// SizedBox(height: 20.0),

// SizedBox(height: 20.0),
// TextField(
// decoration: InputDecoration(labelText: 'Email'),
// ),
// SizedBox(height: 12.0),
// TextField(
// obscureText: true,
// decoration: InputDecoration(labelText: 'Password'),
// ),
SizedBox(height:
20.0), ElevatedButton(
onPressed: () {
signInWithGoogle();
Navigator.pushNamed(context, '/jobs');
```

```
},
child: Text('Login with Google'),
```

```
),
SizedBox(height: 12.0),
// TextButton(
// onPressed: () {
// Navigator.pushNamed(context, '/signup');
// },
// child: Text('Don\'t have an account? Sign up'),
// ),
],
),
),
);
}

signInWithGoogle() async {
GoogleSignInAccount? googleUser= await GoogleSignIn().signIn();
GoogleSignInAuthentication? googleAuth=await googleUser?.authentication;
AuthCredential credential= GoogleAuthProvider.credential(
accessToken: googleAuth?.accessToken,
idToken: googleAuth?.idToken
);

UserCredential userCredential=await FirebaseAuth.instance.signInWithCredential(credential);

print(userCredential.user?.displayName);

FirebaseAuth.instance.signInWithCredential(credential);

}

}
```

Output:



Conclusion: In this experiment , we have successfully created form using form widget and create a login page for my clone application, various properties of form are implemented successfully in the above experiment.

MAD & PWA Lab
Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	16
Name	Om Gaikwad
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	13

Exp 5 To apply navigation, routing and gestures in Flutter App

//Navigation and gestures
Reference :
https://docs.flutter.dev/cookbook/navigation/navigation-basics
Steps:

1. Create two routes. (In Flutter, *screens* and *pages* are called *routes*.)
2. Navigate to the second route using Navigator.push().
3. Return to the first route using Navigator.pop().

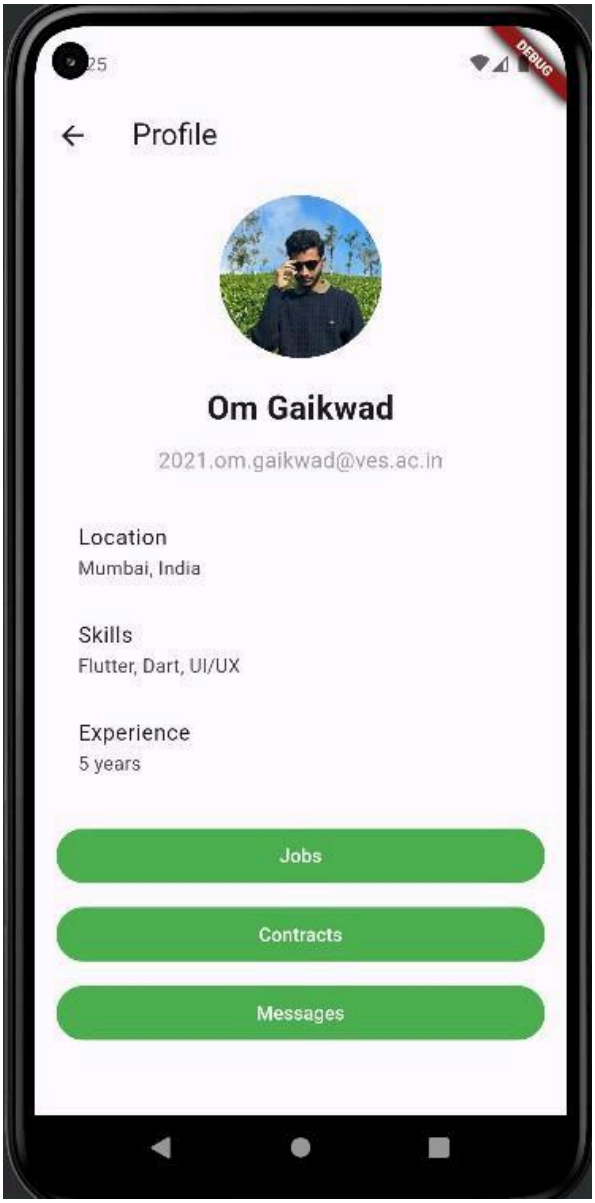
Profile_page.dart

```
import 'package:flutter/material.dart';
import 'package:trial1/screens/jobs/Messages.dart'; import
'package:trial1/screens/jobs/contracts_page.dart'; import
'package:trial1/screens/jobs/jobs_page.dart';

class ProfilePage extends StatelessWidget {
@override
Widget build(BuildContext context) { return
Scaffold(
appBar: AppBar(
title: Text('Profile'),
),
body: Padding(
padding: const EdgeInsets.all(16.0), child:
Column(
crossAxisAlignment: CrossAxisAlignment.stretch,
children: [
CircleAvatar( radius:
60.0,
// Replace with an actual profile picture
backgroundImage: AssetImage('images/WhatsApp Image 2024-02-20 at 00.07.49.jpeg'),
),
SizedBox(height: 20.0),
Text(
'Om Gaikwad', // Replace with actual name
style: TextStyle(fontSize: 24.0, fontWeight: FontWeight.bold),
textAlign: TextAlign.center,
),
SizedBox(height: 10.0),
Text(
'2021.om.gaikwad@ves.ac.in', // Replace with actual email style:
```

```
TextStyle(fontSize: 16.0, color: Colors.grey),
```

```
textAlign: TextAlign.center,
),
SizedBox(height:
20.0), ListTile(
title: Text('Location'),
subtitle: Text('Mumbai, India'), // Replace with actual location
),
ListTile(
title: Text('Skills'),
subtitle: Text('Flutter, Dart, UI/UX'), // Replace with actual skills
),
ListTile(
title: Text('Experience'),
subtitle: Text('5 years'), // Replace with actual experience
),
// Additional details or user settings can go here
// Customize based on your requirements
SizedBox(height: 20.0),
TextButton(
onPressed: () {
Navigator.push(
context,
MaterialPageRoute(builder: (context) => JobsPage()),
);
// Implement messages button functionality
},
child:
Text(
'Jobs',
style: TextStyle(color: Colors.white),
),
style: ButtonStyle(
backgroundColor: MaterialStateProperty.all<Color>(Colors.green),
),
),
SizedBox(height:
10.0), TextButton(
onPressed: () {
Navigator.push(
context,
MaterialPageRoute(builder: (context) => ContractsPage()),
);
// Implement messages button functionality
},
child: Text(
```

```
'Contracts',
style: TextStyle(color: Colors.white),
),
style: ButtonStyle(
backgroundColor: MaterialStateProperty.all<Color>(Colors.green),
),
),
SizedBox(height:
10.0), TextButton(
onPressed: () {
Navigator.push(
context,
MaterialPageRoute(builder: (context) => MessagesPage()),
);
// Implement messages button functionality
},
child: Text(
'Messages',
style: TextStyle(color: Colors.white),
),
style: ButtonStyle(
backgroundColor: MaterialStateProperty.all<Color>(Colors.green),
),
),
],
),
),
);
}
}
```


DEBUG
Profile
Om Gaikwad
2021.om.gaikwad@ves.ac.in
Location
Mumbai, India
Skills
Flutter, Dart, UI/UX
Experience
5 years
Jobs
Contracts
Messages

MAD & PWA Lab
Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	16
Name	Om Gaikwad
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:	13

MAD and PWA Lab

Name: Om Gaikwad **Class: D15A**
Roll no:16

Experiment - 6

Aim: To Connect Flutter UI with FireBase database

Theory:

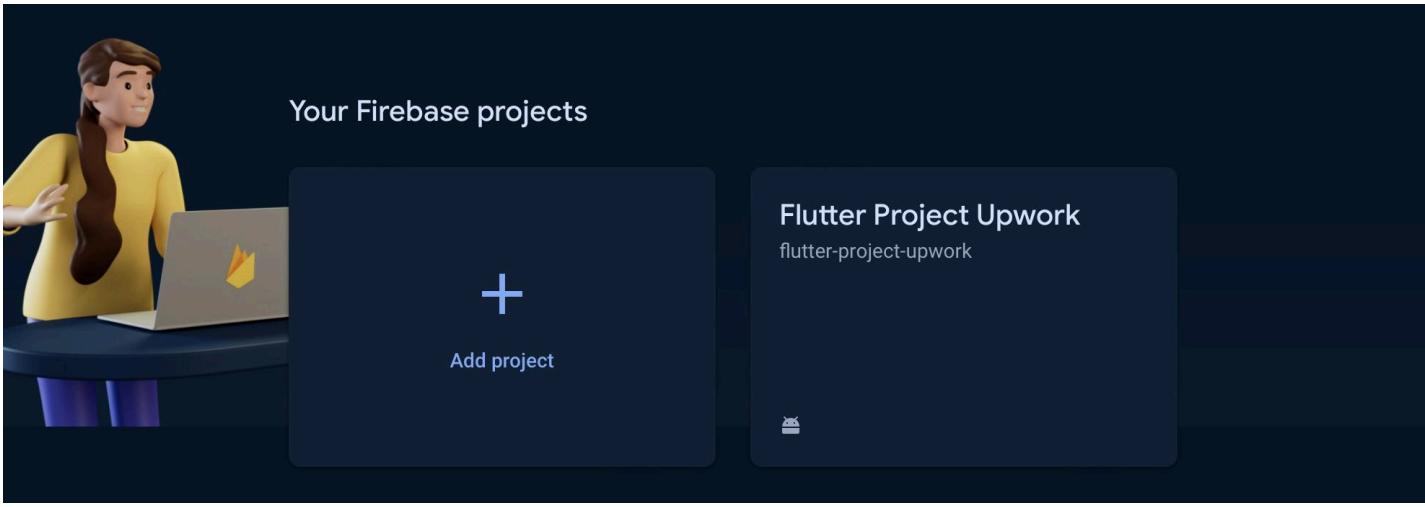
Prerequisites

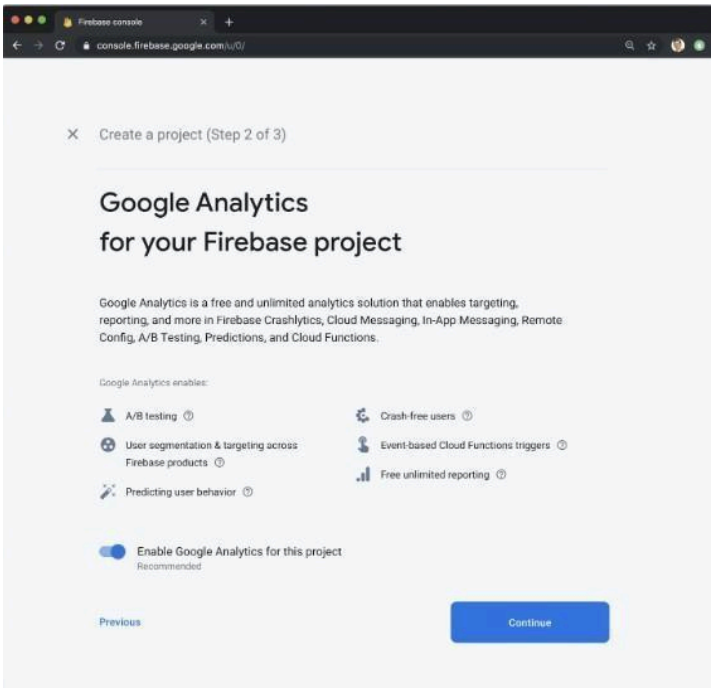
To complete this tutorial, you will need:

- A Google account to use Firebase.
- Developing for iOS will require XCode.
- To download and install Flutter.
- To download and install Android Studio and Visual Studio Code.
- It is recommended to install plugins for your code editor:
    - Flutter and Dart plugins installed for Android Studio.
    - Flutter extension installed for Visual Studio Code.

Create a Firebase Project:

First, log in with your Google account to manage your Firebase projects. From within the Firebase dashboard, select the Create new project button and give it a name:





Go to the Firebase Console and create a new project.
Add your Flutter app to the Firebase project:

Register your app in the Firebase project, and follow the instructions to download the configuration files (google-services.json for Android, GoogleService-Info.plist for iOS).

The most important thing here is to match up the Android package name that you choose here with the one inside of our application.

The structure consists of at least two segments. A common pattern is to use a domain name, a company name, and the application name:

com.example.flutterfirebaseexample

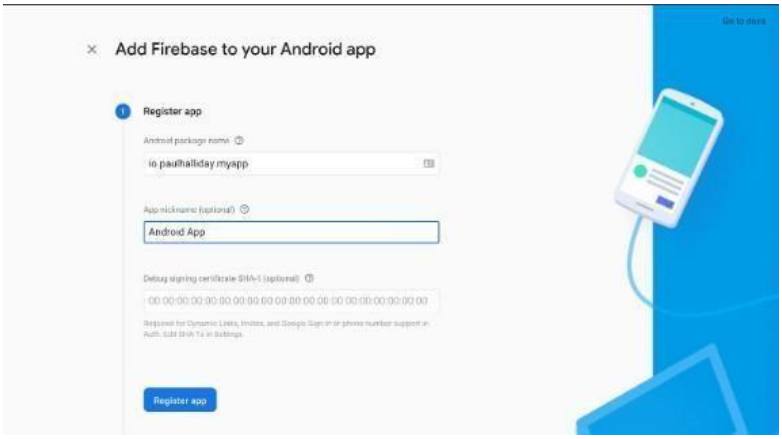
Once you've decided on a name, open android/app/build.gradle in your code editor and update the applicationId to match the Android package name:
android/app/build.gradle

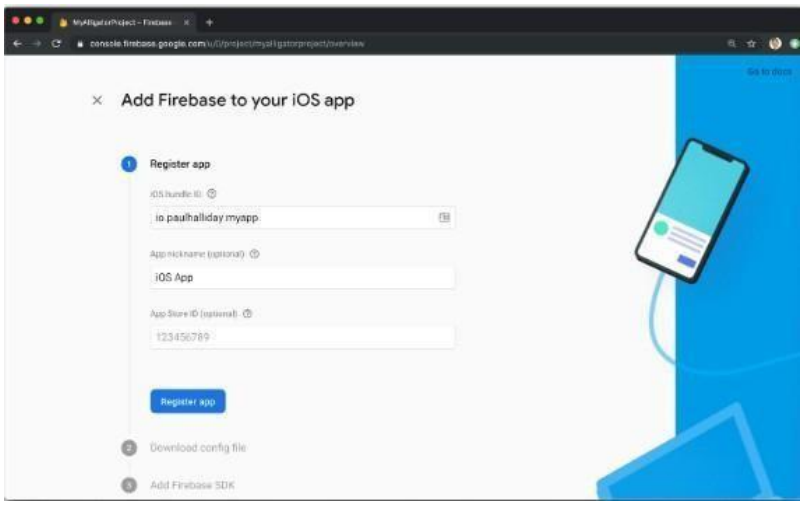
```
...
defaultConfig {
    // TODO: Specify your own unique Application ID (https://developer.android.com/studio/build/application-id.html).
    applicationId 'com.example.flutterfirebaseexample'
    ...
}
...
```

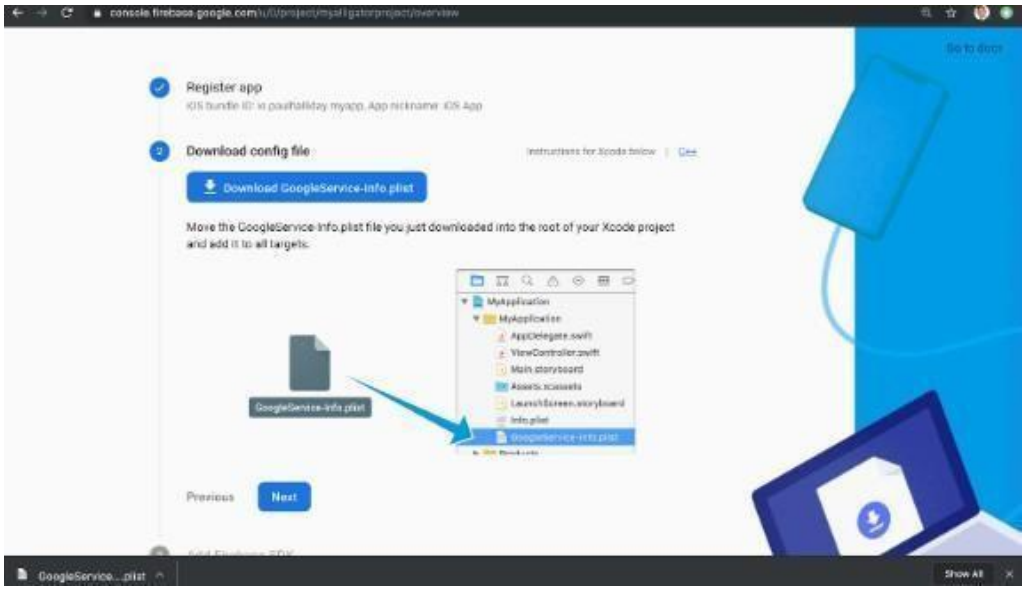
Downloading the Config File

The next step is to add the Firebase configuration file into our Flutter project. This is important as it contains the API keys and other critical information for Firebase to use.

Select Download `google-services.json` from this page:



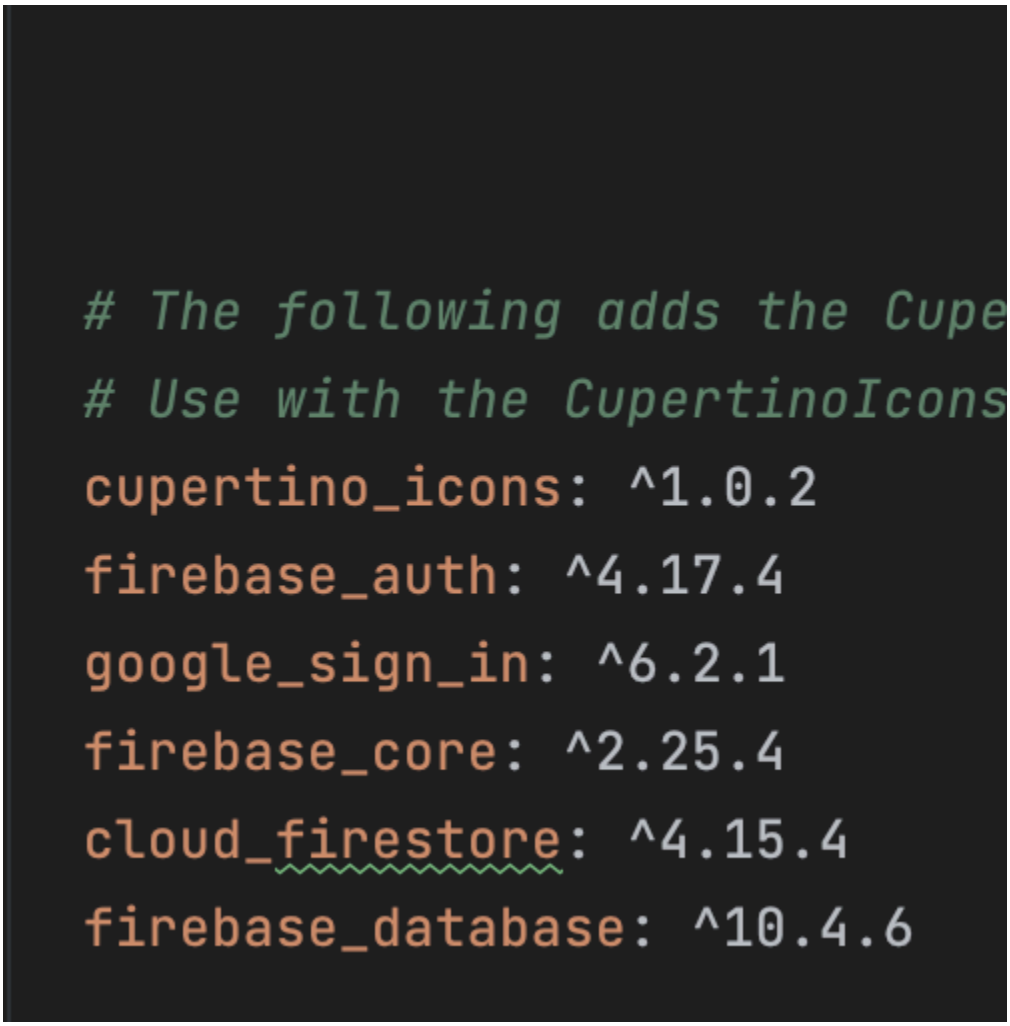




2. Add Firebase to your Flutter project:
Add Dependencies:

Open your pubspec.yaml file and add the necessary dependencies:

yaml



Code:

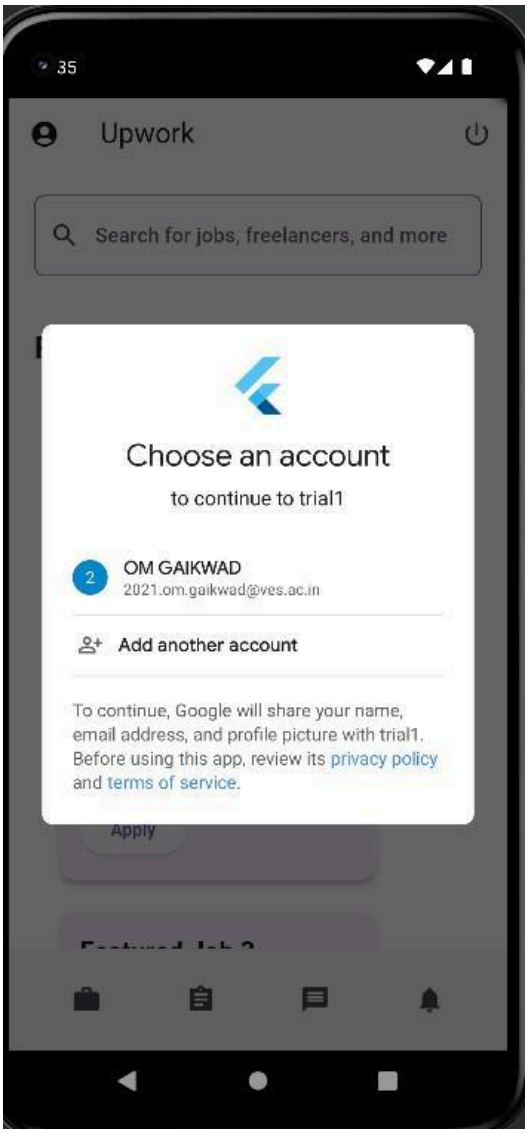
main.dart

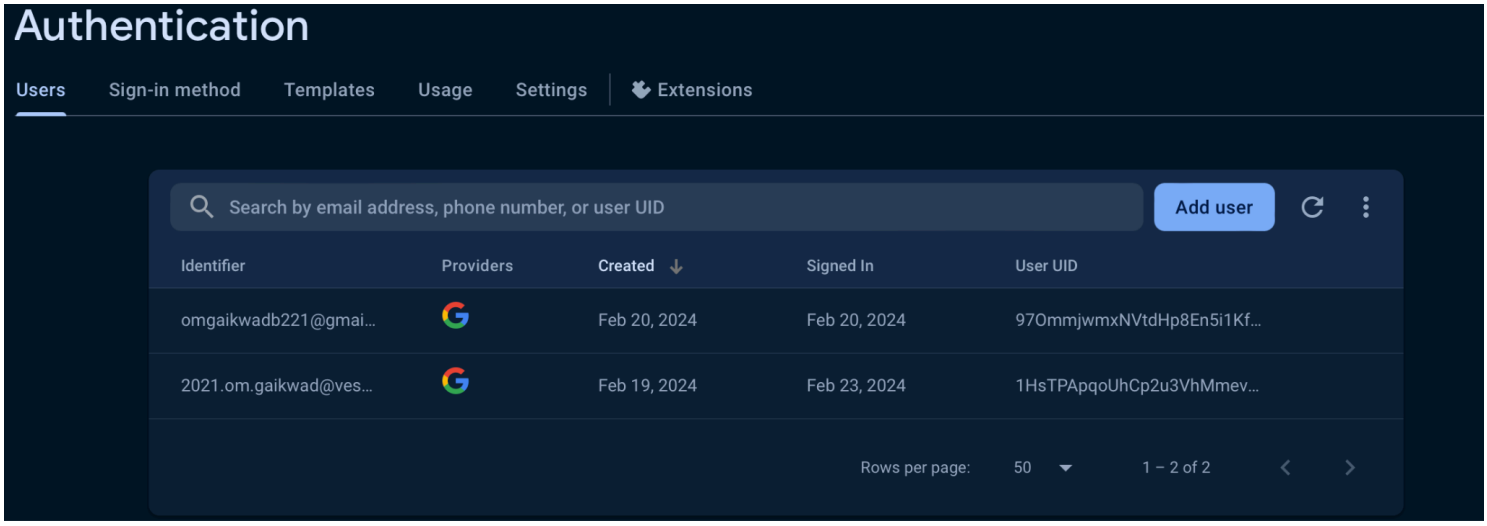
```
// lib/main.dart
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';
import 'screens/login/login_page.dart';
import 'screens/login/signup_page.dart';
import 'screens/jobs/jobs_page.dart';
import 'screens/splash_screen.dart';
import 'package:firebase_auth/firebase_auth.dart';

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

class MyApp extends StatelessWidget {
@override
```

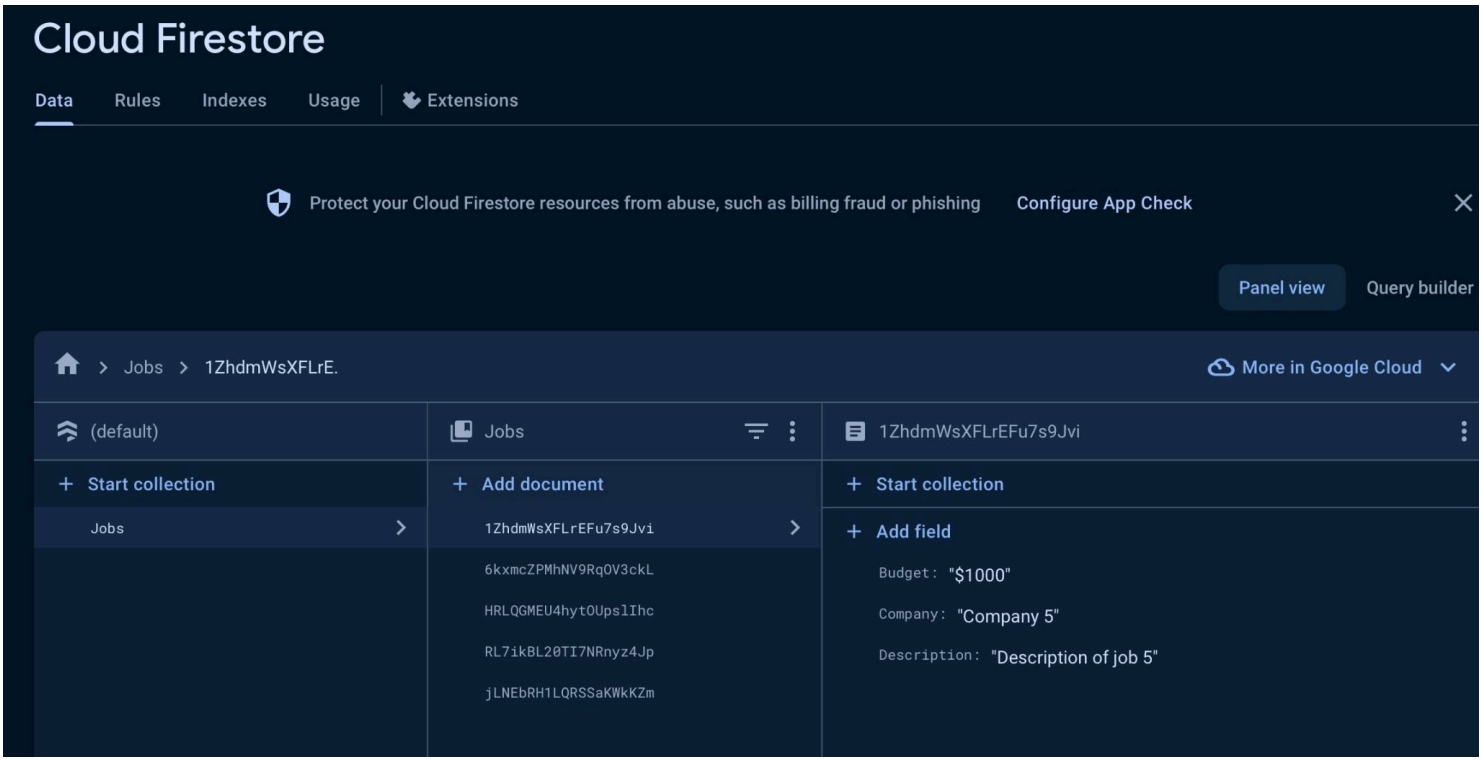
```
Widget build(BuildContext context) {
return MaterialApp(
title: 'Upwork',
theme:
ThemeData(
// primarySwatch: Colors.blue,
),
// home: StreamBuilder<User?>(
// stream: FirebaseAuth.instance.authStateChanges(),
// builder: (BuildContext context ,AsyncSnapshot snapshot){
// if(snapshot.hasError)
// {
// return Text(snapshot.error.toString());
// }
//
// if(snapshot.connectionState==ConnectionState.active)
// {
// if(snapshot.data==null)
// {
// return LoginPage();
// }
//
// else
// {
// return JobsPage();
// }
// }
// return Center(child: CircularProgressIndicator());
// },
// ),




initialRoute:
'/', routes: {
'/': (context) => SplashScreen(),
'/login': (context) => LoginPage(),
'/signup': (context) => SignupPage(),
'/jobs': (context) => JobsPage(),

},
);
}
}
```

Output:

Project Title: Upwork Clone and e-comm site **Roll no. 16**







Conclusion:

In this experiment, we have successfully connected firebase database and authenticated using google signin within the flutter application successfully.

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.	16
Name	Om Gaikwad
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	15

Experiment No:7

Aim: To write meta data of your Ecommerce PWA

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:

1. Progressive — They work for every user, regardless of the browser chosen because they are built at the base with progressive improvement principles.
2. Responsive — They adapt to the various screen sizes: desktop, mobile, tablet, or dimensions that can later become available.
3. App-like — They behave with the user as if they were native apps, in terms of interaction and navigation.
4. Updated — Information is always up-to-date thanks to the data update process offered by service workers.
5. Secure — Exposed over HTTPS protocol to prevent the connection from displaying information or altering the contents.
6. Searchable — They are identified as “applications” and are indexed by search engines.

7. Reactivable — Make it easy to reactivate the application thanks to capabilities such as web notifications.
8. 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.

9. Linkable — Easily shared via URL without complex installations.
10. 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.).

Output:

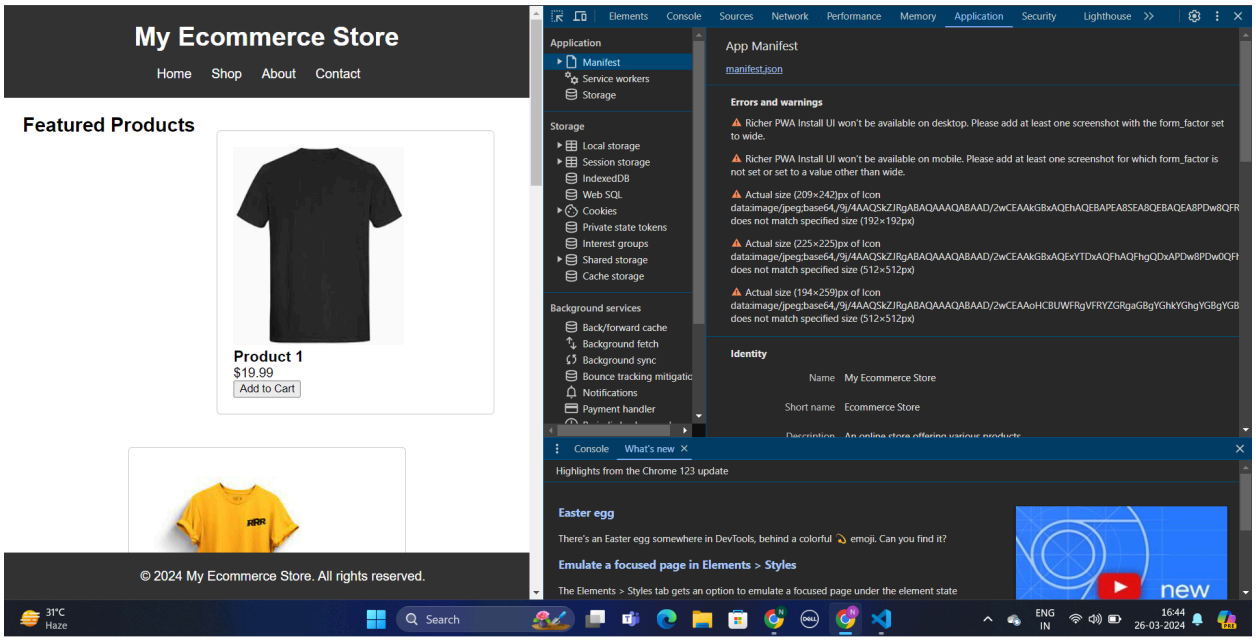
Manifest.json:

```
{
  "name": "My Ecommerce Store",
  "short_name": "Ecommerce Store",
  "description": "An online store offering various products.",
  "icons": [
    {
      "src": "/image.jpg",
      "sizes": "192x192",
      "type": "image/png"
```

```
},
{
    "src": "/image2.jpg",
    "sizes": "512x512",
    "type": "image/png"
},
{
    "src": "/image3.jpg",
    "sizes": "512x512",
    "type": "image/png"
```

```
        }
    ],
    "start_url": "/index.html",
    "display": "standalone",
    "background_color": "#ffffff",
    "theme_color": "#3367D6",
    "orientation": "portrait"
}
```

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>My Ecommerce Store</title>
    <link rel="manifest" href="manifest.json" />
    <link rel="stylesheet" href="styles.css" />
  </head>
```



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.	16
Name	Om Gaikwad
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

Experiment No:8

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

Theory:

Service Worker

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

Things to note about Service Worker:

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

What can we do with Service Workers?

- You can dominate Network Traffic

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

- You can Cache

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

- You can manage Push Notifications

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

- You can Continue

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

What can’t we do with Service Workers?

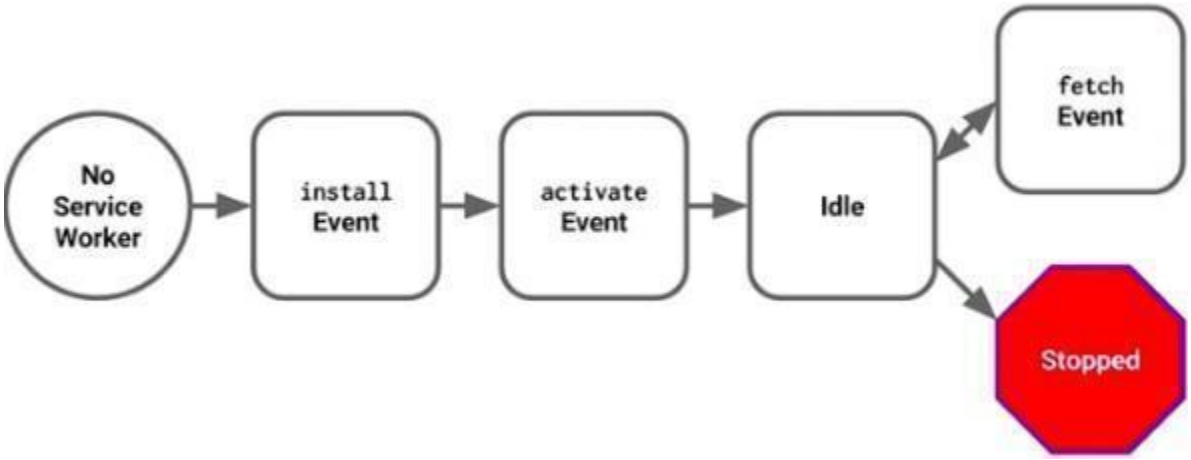
- You can’t access the Window

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

1.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:

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

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.

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

});

self.addEventListener("fetch", function (event)
  { event.respondWith(

    checkResponse(event.request).catch(function () {

      console.log("Fetch from cache successful!");
      return returnFromCache(event.request);
    })

  );

  console.log("Fetch successful!");

  event.waitUntil(addToCache(event.request)); });
```

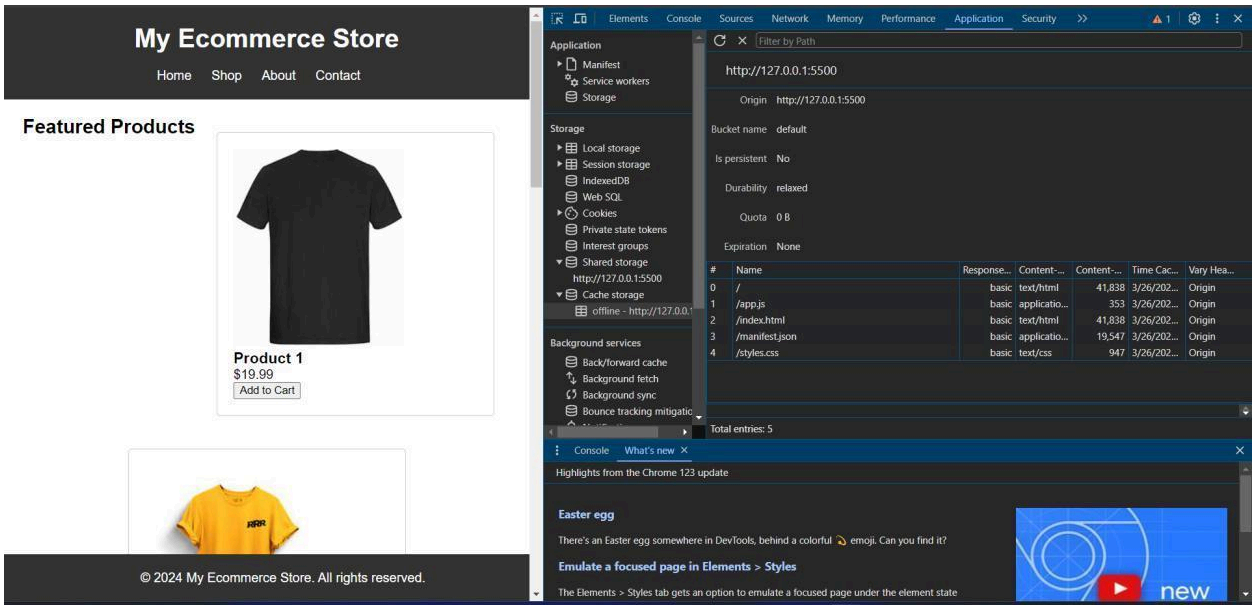
```
self.addEventListener("sync", function (event)
```

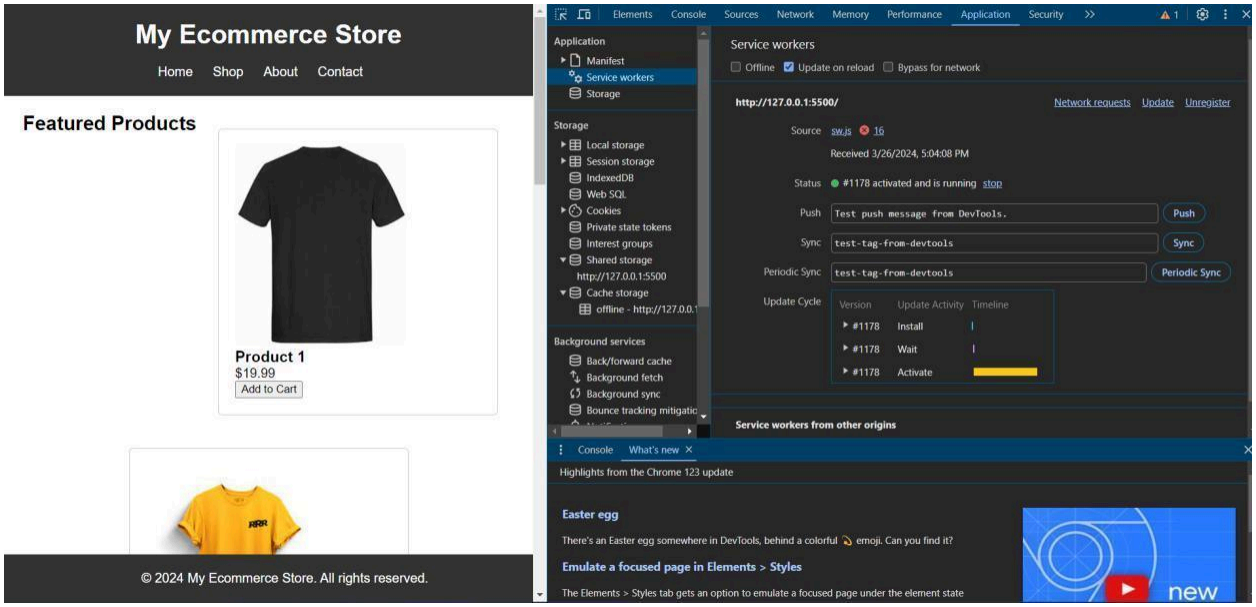
```
  if (event.tag === "syncMessage") {

    console.log("Sync successful!");

  }

 })
  ;

self.addEventListener("push", function (event)
  { if (event && event.data) {

    var data = event.data.json();

    if (data.method === "pushMessage") {
      console.log("Push notification
      sent"); event.waitUntil(

        self.registration.showNotification("Nilesh Ecom", {
          body: data.message,

        })

      );

    }

  }

 })
  ;

var filesToCache = ["/", "/index.html", "/styles.css"];

var preLoad = function () {

  return caches.open("offline").then(function (cache) {
```

```
    //    caching index and important
    routes return
    cache.addAll(filesToCache);
  });
};

var checkResponse = function (request) {
  return new Promise(function (fulfill, reject) {
    fetch(request)
```

```
      .then(function (response) {
        if (response.status !== 404) {
          fulfill(response);
        } else {
          reject();
        }
      })
      .catch(reject);
  });
```

```
};

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

    });
  });
};

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





Conclusion: We have Successfully Understood and implemented register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA.

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.	16
Name	Om Gaikwad
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

Experiment No:09

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.

Here, you can create any scenario for yourself. A sample is in the following for this case.

When we click the "send" button, email content will be saved to IndexedDB.

Background Sync registration.

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

Push Event

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

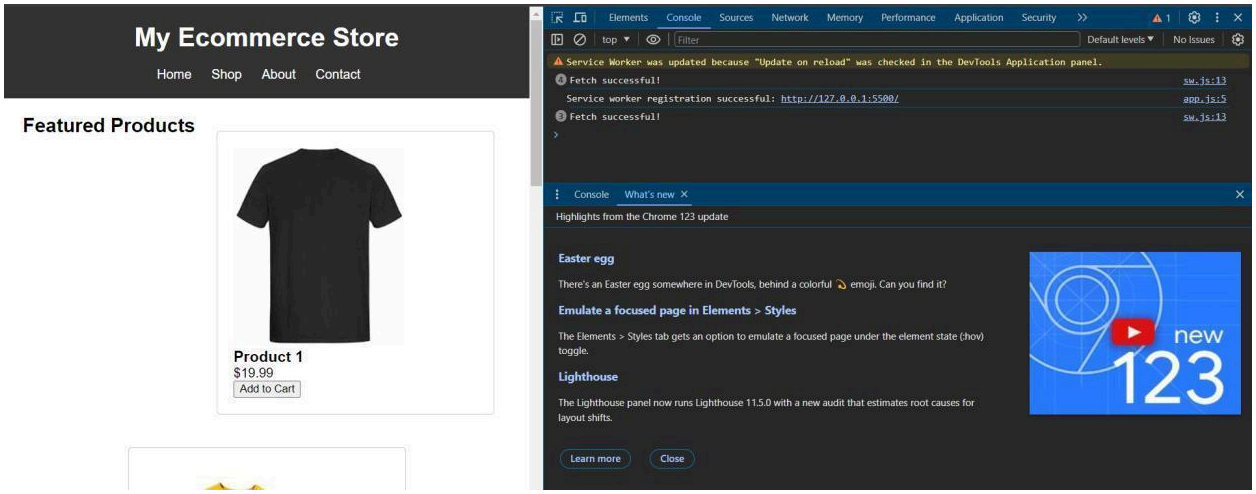
We can check in the following example.

"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.

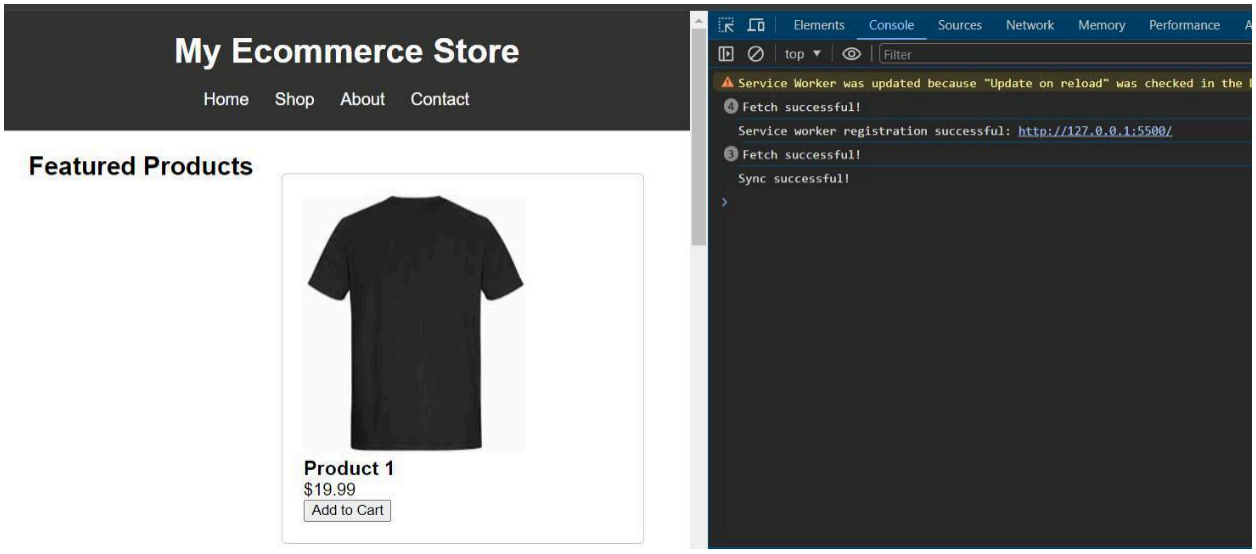
In the following code block is in sw.js file. You can handle push notifications with this event. In this example, I kept it simple. We send an object that has "method" and "message" properties. If the method value is "pushMessage", we open the information notification with the "message" property.

Implementation:

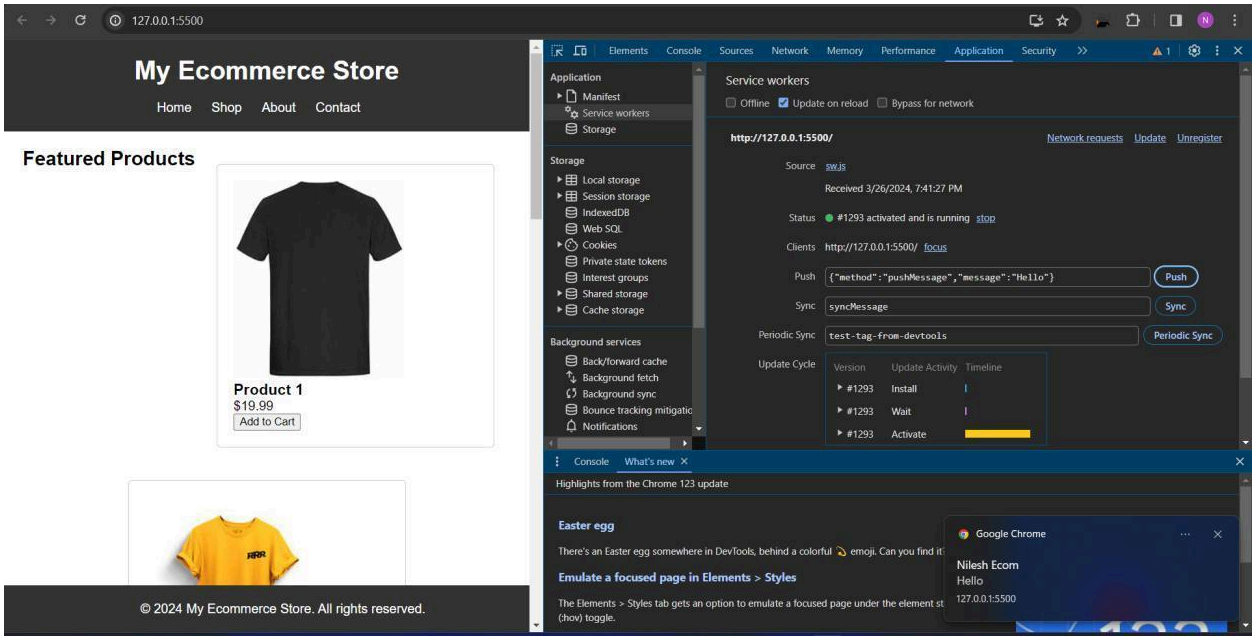
1.Fetch Event:



2.Sync Event:



3. Push Event:



Conclusion:

I have Successfully understood and implemented Service worker events like fetch, sync and push for E-commerce PWA

MAD & PWA Lab
Journal

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	16
Name	Om Gaikwad
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

ExperimentNo. 10

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

Theory:

GitHub Pages

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

GitHub Pages provides the following key features:

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

Reasons for favoring this over Firebase:

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

GitHub Pages is used by Lyft, CircleCI, and HubSpot.

GitHub Pages is listed in 775 company stacks and 4401 developer stacks.

Pros

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

Cons

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

Firebase

The Realtime App Platform. Firebase is a cloud service designed to power real-time, collaborative applications. Simply add the Firebase library to your application to gain

access to a shared data structure; any changes you make to that data are automatically synchronized with the Firebase cloud and with other clients within milliseconds.

Some of the features offered by Firebase are:

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

Reasons for favoring over GitHub Pages:

1. Realtime backend made easy
2. Fast and responsive

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

Pros

1. Hosted by Google. Enough said.

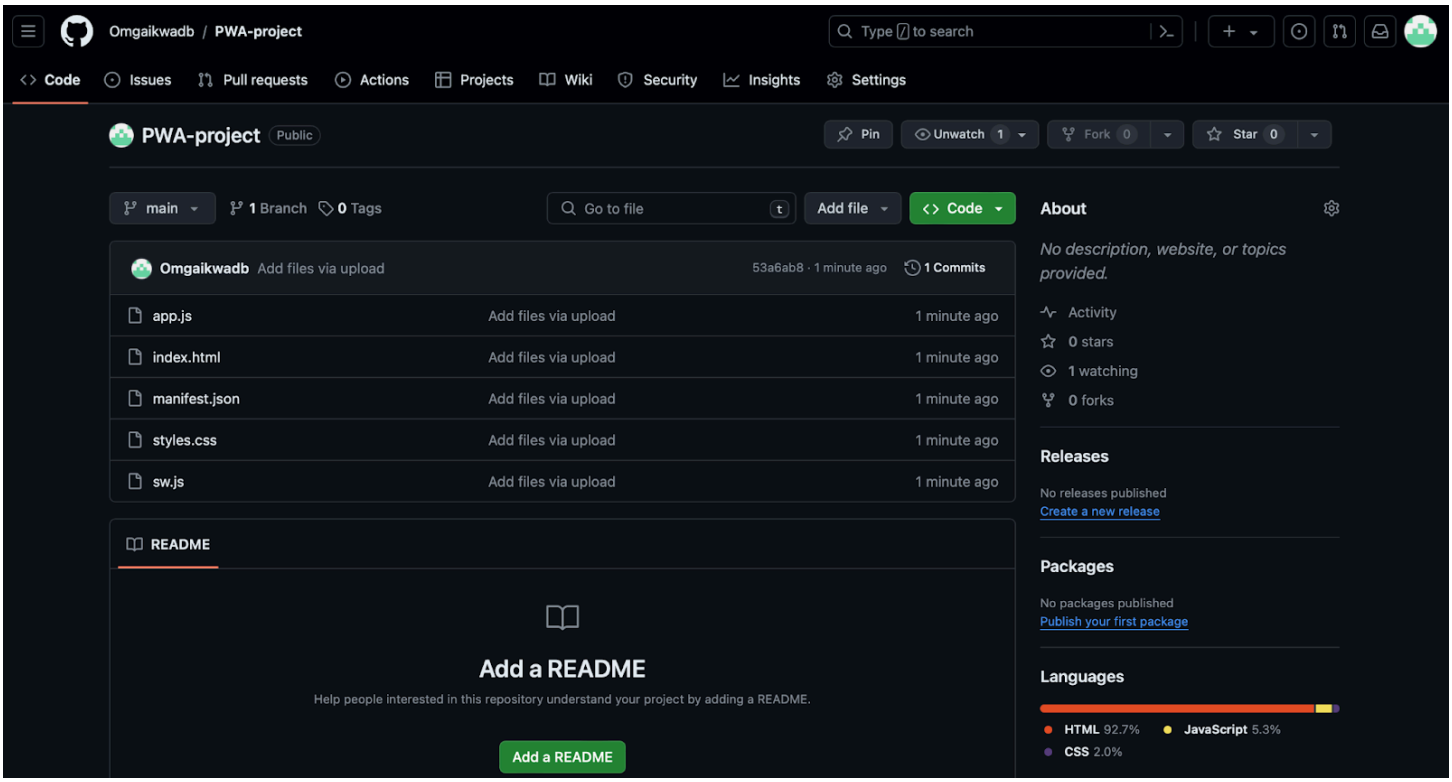
2. Authentication, Cloud Messaging, and a whole lot of other handy services will be available to you.
3. A real-time database will be available to you, which can store 1 GB of data.
4. You'll also have access to a blob store, which can store another 1 GB of data.
5. Support for HTTPS. A free certificate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

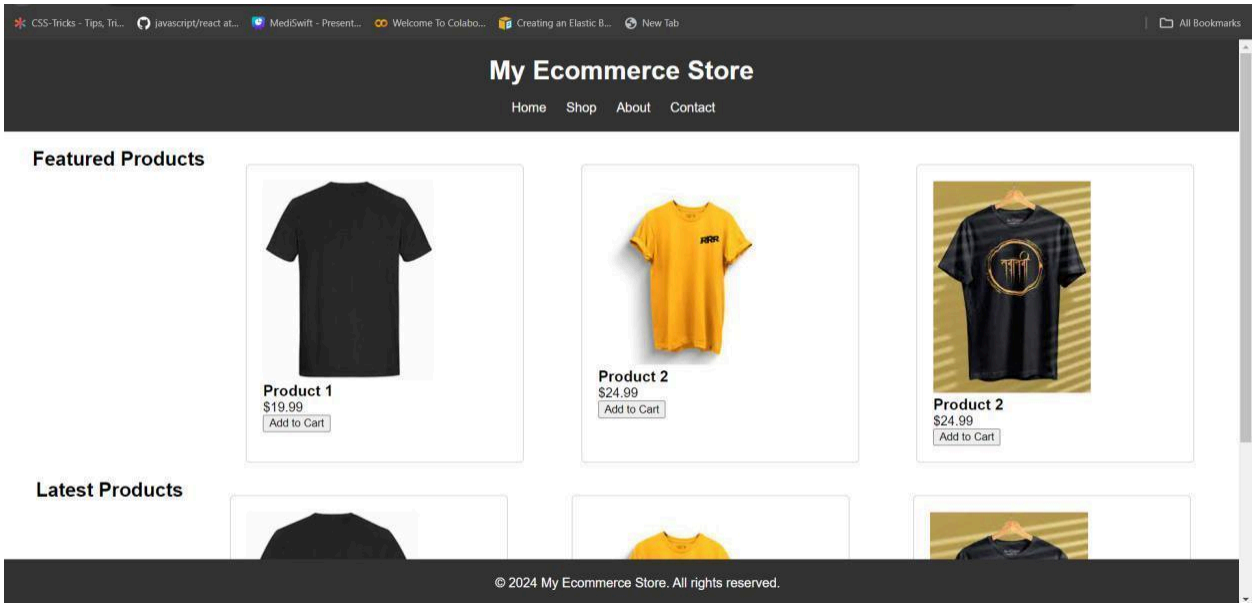
Cons

1. Only 10 GB of data transfer is allowed per month. But this is not really a big problem, if you use a CDN or AMP.
2. Command-line interface only.
3. No in-built support for any static site generator.

Link to our GitHub repository:

Github Screenshot:





Conclusion:I have Successfully studied and implemented deployment of Ecommerce PWA to GitHub Pages.

MAD & PWA Lab
Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	16
Name	Om Gaikwad
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	15

Experiment No:11

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

Theory:

Google Lighthouse :

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

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

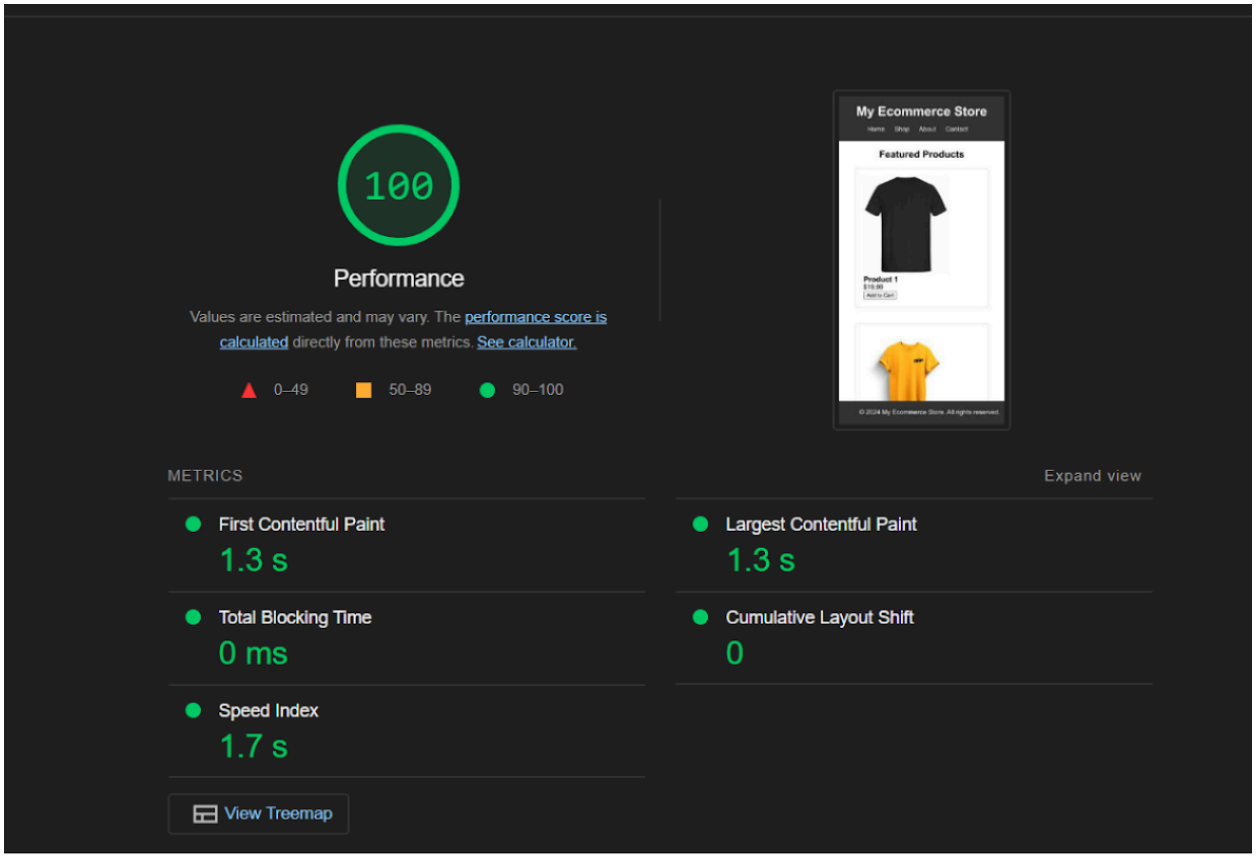
Key Features and Audit Metrics

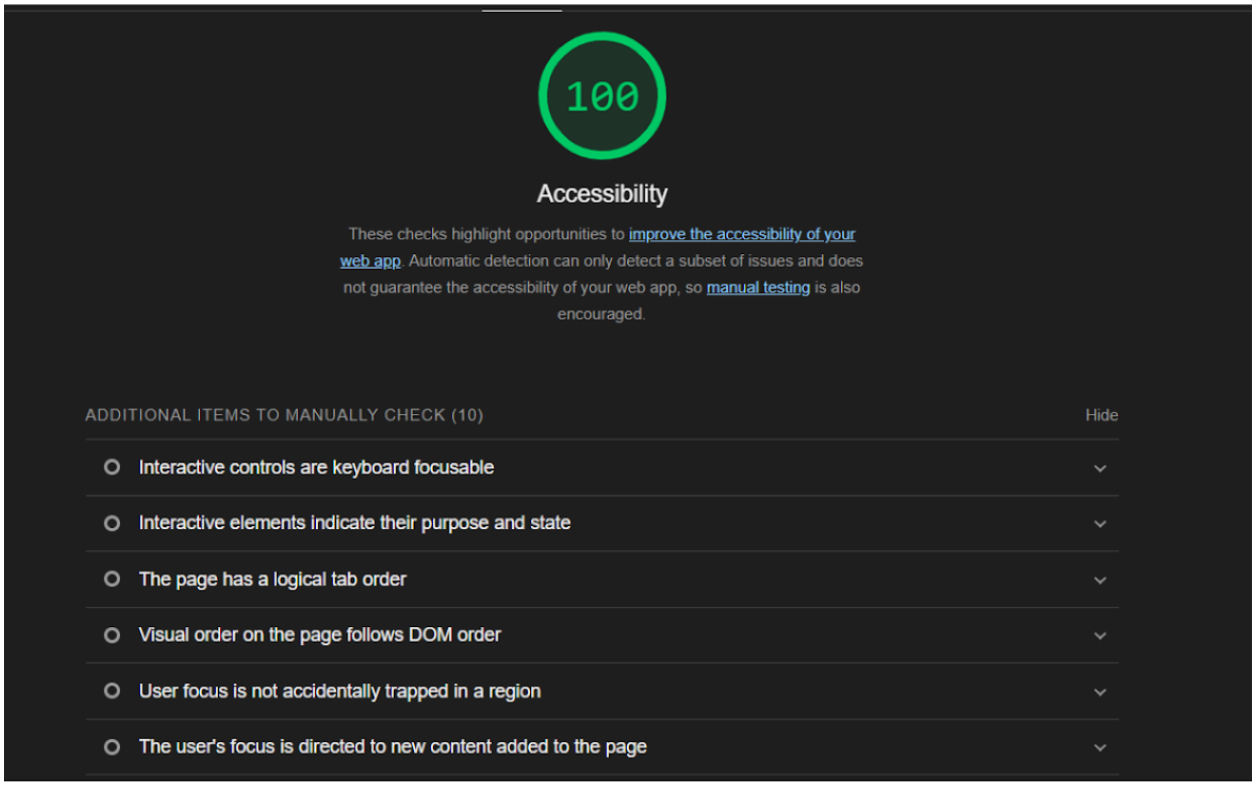
Google Lighthouse has the option of running the Audit for Desktop as well as mobile version of your page(s). The top metrics that will be measured in the Audit are:

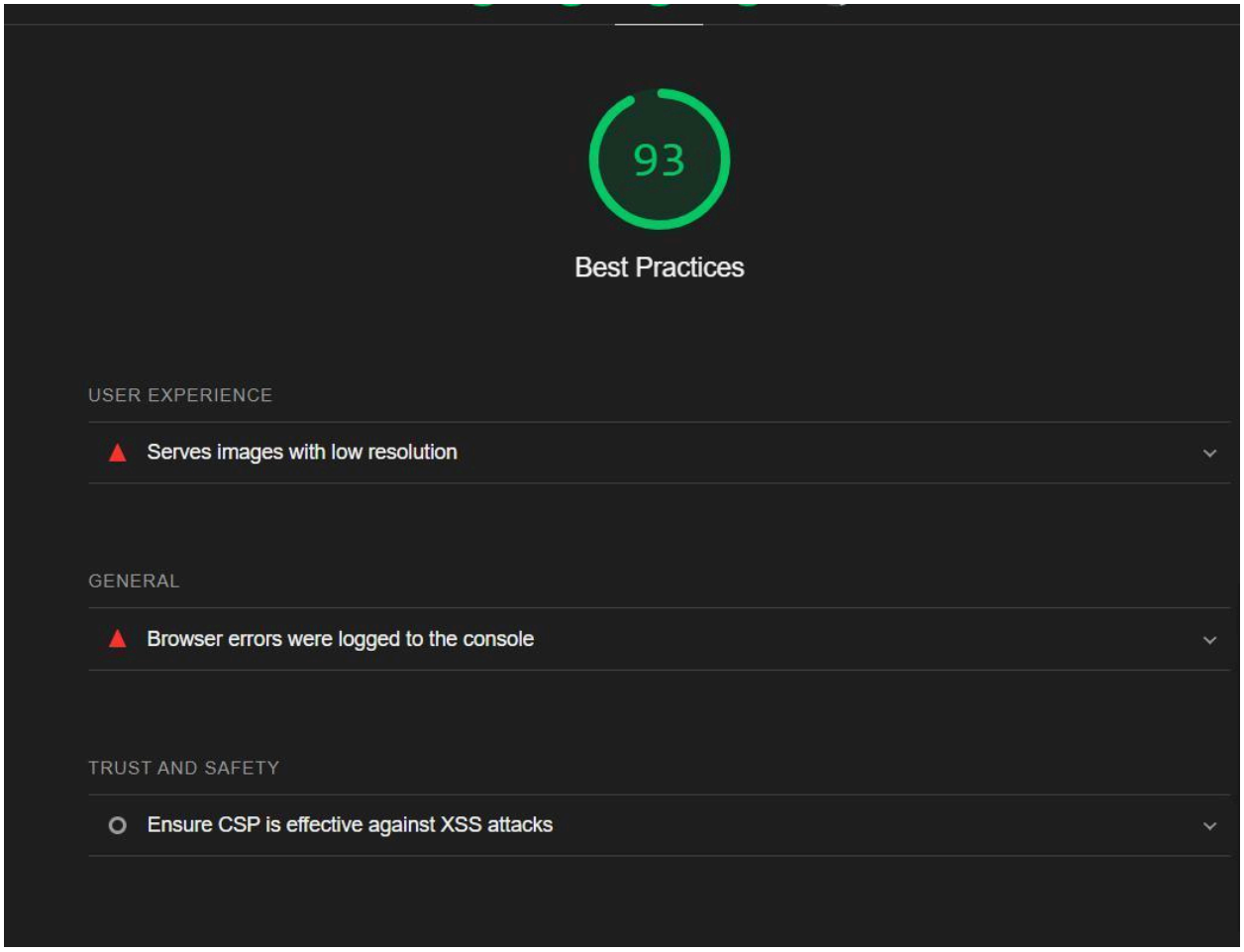
1. Performance: This score is an aggregation of how the page fared in aspects such as (but not limited to) loading speed, time taken for loading for basic frame(s), displaying meaningful content to the user, etc. To a layman, this score is indicative of how decently the site performs, with a score of 100 meaning that you figure in the 98th percentile, 50 meaning that you figure in the 75th percentile and so on.
2. PWA Score (Mobile): Thanks to the rise of Service Workers, app manifests, etc., a lot of modern web applications are moving towards the PWA paradigm, where the objective is to make the application behave as close as possible to native mobile applications. Scoring points are based on the Baseline PWA checklist laid down by Google which includes Service Worker implementation(s), viewport handling, offline functionality, performance in script-disabled environments, etc.
3. Accessibility: As you might have guessed, this metric is a measure of how accessible your website is, across a plethora of accessibility features that can be implemented in your page (such as the 'aria-' attributes like aria-required, audio captions, button names, etc.). Unlike the other metrics though, Accessibility metrics score on a pass/fail basis i.e. if all possible elements of the page are not screen-reader friendly (HTML5 introduced features that would make pages easy to interpret for screen readers used by visually challenged people like tag names, tags such as <section>, <article>, etc.), you get a 0 on that score. The aggregate of these scores is your Accessibility metric score.

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

Implementation:


100
Performance
Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.
0–49
50–89
90–100
My Ecommerce Store
Featured Products
METRICS
Expand view
First Contentful Paint
1.3 s
Largest Contentful Paint
1.3 s
Total Blocking Time
0 ms
Cumulative Layout Shift
0
Speed Index
1.7 s
View Treemap


100
Accessibility
These checks highlight opportunities to improve the accessibility of your web app. Automatic detection can only detect a subset of issues and does not guarantee the accessibility of your web app, so manual testing is also encouraged.
ADDITIONAL ITEMS TO MANUALLY CHECK (10)
Hide
Interactive controls are keyboard focusable
Interactive elements indicate their purpose and state
The page has a logical tab order
Visual order on the page follows DOM order
User focus is not accidentally trapped in a region
The user's focus is directed to new content added to the page



Conclusion: We have Successfully understood and used google Lighthouse PWA Analysis Tool to test 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.	16
Name	Om Gaikwad
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	5

Om Gaikwad
DISA 16
Batch - A

MAD PWA
Assignment : 1

Q1) Flutter overview :-

Key features →
1. Single Codebase: write once and run anywhere
2. Hot ~~reload~~ reload :- Instantly see code changes without expensive UI.
3. Rich widget library :- Customizable widgets for expressive UI.
4. Native performance : compiled to native ARM code for speed.
5. dart programming Language :- Simple, object oriented language

Advantages :-
1) Faster development : single codebase and hot reload speed up development.
2) cost effective : reduces costs with unified development
3) Consistent UI/UX : ensures a consistent experience across platforms
4) Community support :- Active community contributes to ecosystem.
5) Customization and flexibility : Rich widgets offer & design flexibility.

differences from traditional approaches :-
1) Compiled code : It compiles to native code for better performance
2) UI Components : uses its own widgets for consistent visuals
3) Language choice : dart is tailored for UI development
4) development workflow : Hot reload accelerates development

popularity :-

1) Ease of learning : dart's simplicity appeals to diverse developers.
2) productive gains : Hot reload and single codebase boost efficiency.
3) Growing ecosystem : Vibrant community enhances flutter capabilities.
4) Google's backing : Google's support ensures ongoing improvement.
5) Versatility : Suited for ~~divi~~ diverse applications from startups to enterprises.

Q2] Widget tree and Composition :-

→ It is hierarchical structure that represents the visual elements of a user interface. Every component, from basic elements like buttons and text to complex layouts and screens is represented by a widget.

Widget Composition for building complex UIs :-

→ It involves combining multiple smaller widgets to create more complex and feature-rich user interfaces.

1) Container widgets :- These are higher-level widgets that contain and arrange other. It includes :-
   → container
   → Column and Row
   → Stack
   → ListView and GridView

2. Leaf Widgets :- These are lower - level widgets representing specific UI elements. It includes :-

→ Text → Icon → Text field.
→ image → Button widgets

⇒ commonly used widgets and their roles in creating a widget tree :-

1) Container :- Role → A box model for sizing positioning and decorating child widgets

eg.

```
Container(
    margin : EdgeInsets.all(16.0),
    padding : EdgeInsets.symmetric(horizontal : 20.0,
                                   vertical : 10.0),
    color : Colors.blue,
    child : Text('Hello, Flutter!'),
)
```

2) Column and row :-

Role : Arrange children vertically or horizontally

eg

```
Column(
    children : [
        Text('Item 1'),
        Text('Item 2'),
        Text('Item 3'),
    ],
)
```

3. Stack :-
Role → overlap widgets using a stack - like layout

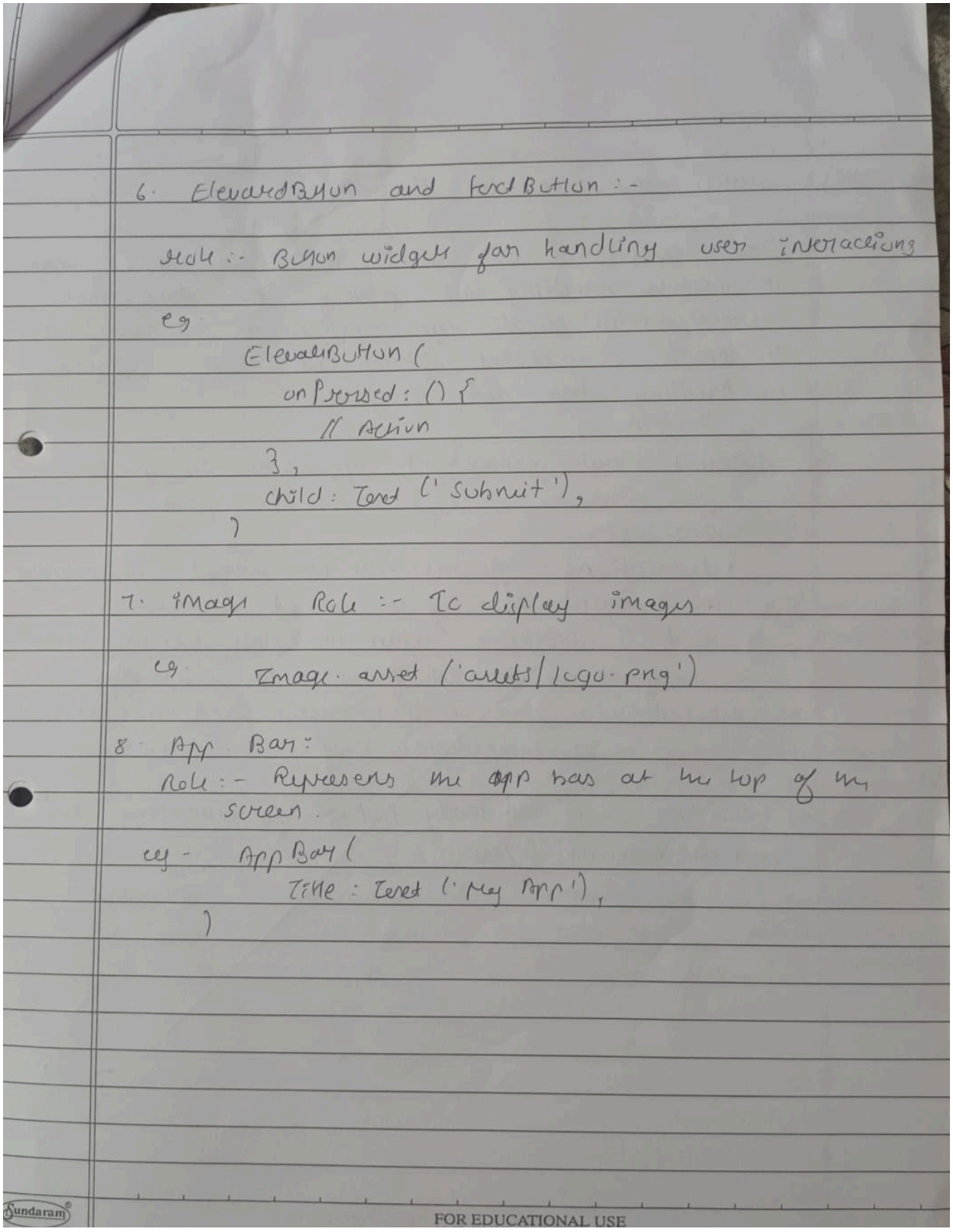
eg:
```
stack (
  children : [
    Positioned (
      top : 10.0,
      left : 10.0,
      child : Text ('Top left'),
    ),
    Text ('Center'),
  ],
)
```

4. ListView :
Role :- display a scrolling list of widgets

eg
```
ListView (
  children : [
  ListTile (title : Text ('Item 1')),
  List Tile (title : Text ('Item 2')),
  ],
)
```

5. Text field : Role → Accept user i/p for text.

eg
```
Textfield (
    decoration : InputDecoration (
      label Text : 'username',
    ),
)
```



6. ElevatedButton and TextButton :-

Role :- Button widgets for handling user interactions

eg

```
ElevatedButton(
  onPressed: () {
    // Action
  },
  child: Text('Submit'),
)
```

7. Image Role :- To display images

eg. `Image.asset('assets/logo.png')`

8. App Bar :

Role :- Represents the app bar at the top of the screen.

eg -

```
AppBar(
  Title: Text('My App'),
)
```

08) State management in flutter :-

State management is crucial in flutter apps because it involves handling and updating the data that determines the visual representation of the user interface. It ensures a consistent, efficient and scalable approach to handling data and UI updates.

different state management approaches in flutter :-

1) setState :-
   - description : flutters built in method to manage the internal state of a widget
   - How it works : when the state changes 'setState' triggers a rebuild of the widget, updating the UI.
   - Suitability :- for small to medium sized apps or simple UI components.

2) Provider :- A 3rd party package for managing state and dependency injection.

3)

→

Comparing different state management approaches.

Aspect	setState	Provider	Riverpod
Built-in Solution	Yes	No (3rd party)	No (3rd party)
Scope	Local to widget	global, centralized	global, centralized
Complexity	Simple	moderate to advanced	moderate to adv.
dependencies	minimal	Provider package	Riverpod package.
flexibility	limited	high	high
* usage scenario	small to medium sized apps	medium to large sized apps	medium to large sized apps.
ecosystem support	integrated info flutter	popular well-supported	growing community support.

q4) Flutter Firebase integration in flutter :-

Process of integrating firebase with flutter :-

1. Create a firebase project
2. Add flutter app to firebase project
3. Add firebase SDK dependencies
4. initialize firebase in flutter app.
5. use firebase services

Benefits of using firebase as a backend solution :-

1. Real time data sync :- it allows updates to be immediately reflected.
2. Authentication :- Supports various sign in methods.
3. Cloud firestore :- offers flexible and scalable solution.
4. Cloud functions :- serverless computing for executing backend logic
5. Cloud storage :- Store and serve user-generated content.
6. Authentication and authorization :- provides secure and simple authentication methods.

Data synchronization in firebase :-

1) Firestore :- Firestore is a NOSQL database that supports real-time updates.
- → when data changes in firestore connected flutter clients receive automatic updates in real time.

2) Realtime database :-
→ similar to firestore, the realtime database in firebase ensures real time data synchronization.

3) Cloud functions :-

→ cloud functions can be triggered in response to changes in Firebase data.

→ This allows developers to implement server side logic that automatically updates data or performs additional actions upon changes.

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.	16
Name	Om Gaikwad
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:	3

DISA (16)

Assignment - 2

Q1) PWA and its significance :-

→ It is a website that delivers an app-like user experience, PWA's are built using web technologies like HTML, CSS and Javascript but they are capable of working offline and other features typically associated with native apps like push notifications and the ability to be added to a home screen.

Significance :-

→ Significant in modern web development because they offer a way to bridge the gap between websites and native apps.
→ They provide users with more engaging and app like experience on the web without the need to download and install an app from an app store.

Key features :-

→ installability
→ offline functionality
→ push notifications
→ discoverability
→ cost

Teacher's Sign.: _____

Q2] Responsive web design :-

→ Provides :-
- improved user experience
- reduced development cost
- increased reach.

importance :-

1) Cross device compatibility :- PWA's are designed to work seamlessly across various devices and ensures that the PWA's interface adapts fluidly to different screen resolutions and orientations providing users with a consistent experience regardless of the device they use.

2) Enhanced user experience :- Responsive design plays a pivotal role in delivering an optimal user experience. By automatically adjusting the layout content and navigation elements based on the screen size and capabilities of the device.

Approaches	Pros	Cons
1) Responsive	• easy to implement • works well for most websites	• can be complex for websites with complex content
2) fluid	• flexible and can adapt to any screen size	• can be difficult to control the layout.
3) adaptive	• provides the best possible experience for each screen size	• Time consuming and expensive to develop and maintain.

Teacher's Sign.: _____

Q3] lifecycle of service worker consists of three main phases :-

1) Registration :-
→ The service worker script is registered with the browser.
→ Typically occurs in the main JavaScript file of the web application such as index.js or app.js
→ developers use the navigator.serviceWorker.register() method to register the service worker.

2) installation :-
→ The browser proceeds to install the service worker.
→ During installation the browser downloads the service worker script specified during registration.
→ developers define an event listener for various lifecycle event such as 'install', 'activate', 'fetch', and 'push'.

3) Activation.
→ once the service worker is installed, it enters the activating phase.
→ Activation occurs after the installation is completed and the service worker script is successfully executed.
→ service worker takes control of the web application and starts handling incoming network requests.

Q4] IndexedDB is a powerful client side storage mechanism available in web browsers that allows developers to store and retrieve large amounts of structured data.

Uses :-

1) offline data storage :-
→ service workers enable offline capabilities for web applications, allowing them to continue functioning even when the user is offline.
→ used to store essential application data locally on the user's device ensuring that the application remains functional even without an internet connection.

2) caching dynamic content :-
→ service workers can cache dynamic content received from network requests for future use.
→ IndexedDB provides a structural storage mechanism for caching dynamic data, such as API responses or user generated content.

3) Background data sync :- SW can enable background synchronization of data with the server, allowing apps to keep local data up-to-date without user intervention.

4) Persistent storage :-
→ IndexedDB provides persistent storage that persists even when the browser is closed or the device is restarted.

Teacher's Sign.: _____