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 **Aditya Parmanand Tahiliani** of **D15-A** semester **VI,** has 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 Jewani

Principal

Head of Department

Dr. Mrs. Shalu Chopra

Name of the Course	**:** MAD & PWA Lab	**Course Code :**	ITL604
Year/Sem/Class	**:** D15A/D15B	**A.Y.:**	23-24
Faculty Incharge	: Mrs. Kajal Jewani.		
Lab Teachers	**:** Mrs. Kajal Jewani.		
Email	**:** kajal.jewani@ves.ac.in		

Programme Outcomes: The graduate will be able to:

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

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

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

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

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

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

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

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

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

PO10) Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as being able to comprehend and write

effective reports and design documentation, make effective presentations and give and receive clear instructions.

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

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

Program specific Outcomes

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

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

Lab Objectives:

Sr. No.	Lab Objectives
The Lab experiments aims:	
1	Learn the basics of the Flutter framework.
2	Develop the App UI by incorporating widgets, layouts, gestures and animation
3	Create a production ready Flutter App by including files and firebase backend service.
4	Learn the Essential technologies, and Concepts of PWAs to get started as quickly and efficiently as possible
5	Develop responsive web applications by combining AJAX development techniques with the jQuery JavaScript library.
6	Understand how service workers operate and also learn to Test and Deploy PWA.

Lab Outcomes

Sr. No.	Lab Outcomes	Cognitive levels of attainment as per Bloom's Taxonomy
On Completion of the course the learner/student should be able to:		
1	Understand cross platform mobile application development using Flutter framework	L1, L2
2	Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation	L3
3	Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS	L3, L4
4	Understand various PWA frameworks and their requirements	L1, L2
5	Design and Develop a responsive User Interface by applying PWA Design techniques	L3

6	Develop and Analyze PWA Features and deploy it over app hosting solutions	L3, L4

Index

Sr. No	Experiment Title	LO	DOP	DOS	Grade
1.	To install and configure the Flutter Environment	LO1	17/1/24	24/1/24	11
2.	To design Flutter UI by including common widgets.	LO2	24/1/24	31/1/24	11
3.	To include icons, images, fonts in Flutter app	LO2	31/1/24	7/2/24	11
4.	To create an interactive Form using form widget	LO2	7/2/24	14/2/24	11
5.	To apply navigation, routing and gestures in Flutter App	LO2	14/2/24	21/2/24	11
6.	To Connect Flutter UI with fireBase database	LO3	21/2/24	6/3/24	11
7.	To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”.	LO4	6/3/24	13/3/24	15
8.	To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA	LO5	13/3/24	20/3/24	15
9.	To implement Service worker events like fetch, sync and push for E-commerce PWA	LO5	20/3/24	27/3/24	15
10.	To study and implement deployment of Ecommerce PWA to GitHub Pages.	LO5	27/3/24	27/3/24	15
11.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.	LO6	27/3/24	27/3/24	15
12.	Assignment-1	LO1,LO 2,LO3	6/2//24	5/2/24	5
13.	Assignment-2	LO4,LO 5,LO6	20/3/24	21/3/24	4

MAD & PWA Lab
Journal

Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	59
Name	**Aditya P Tahiliani**
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	10

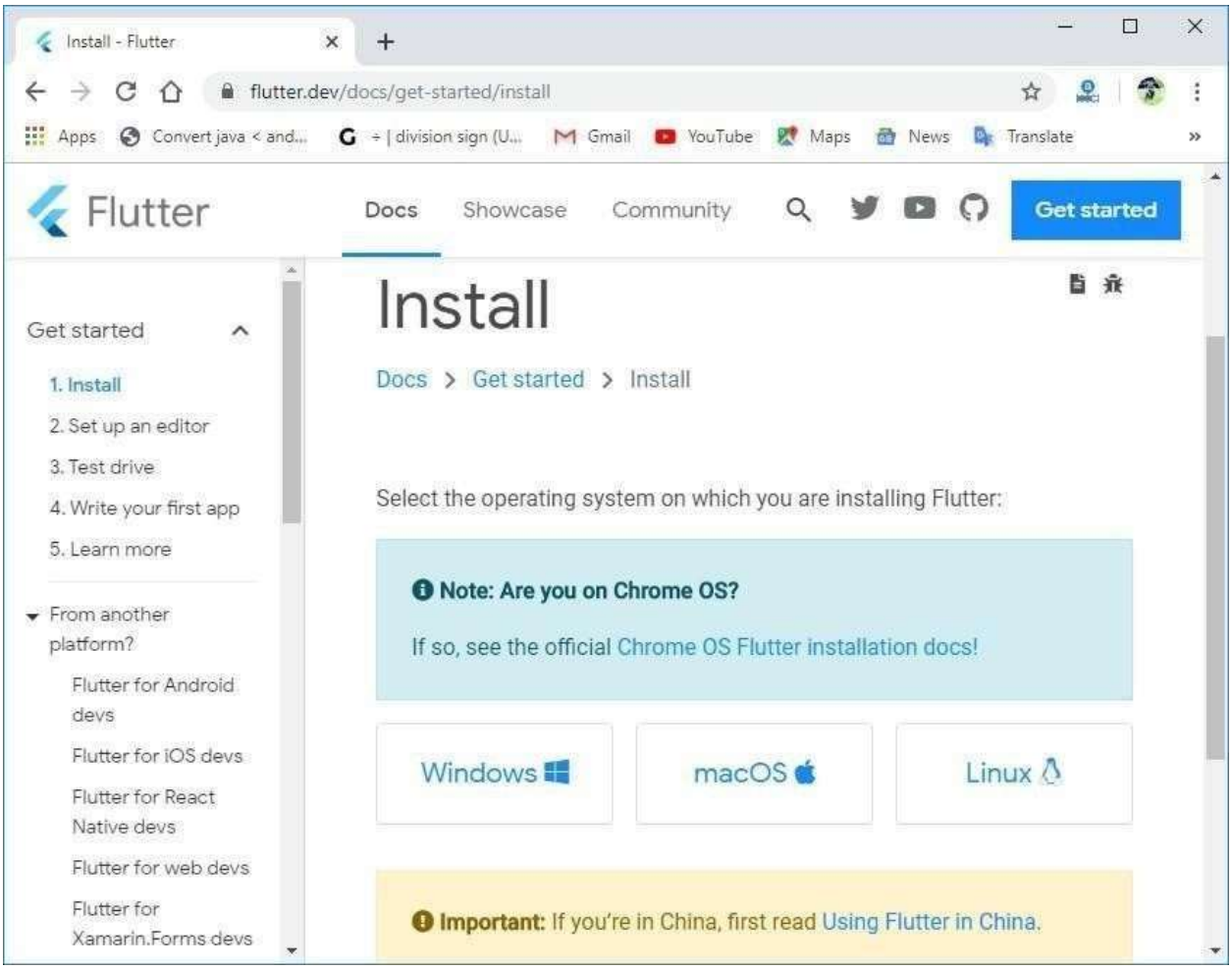
Experiment No: 01

Name:- Aditya Parmanand Tahiliani **Class:- D15-A Roll No:- 59**

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.

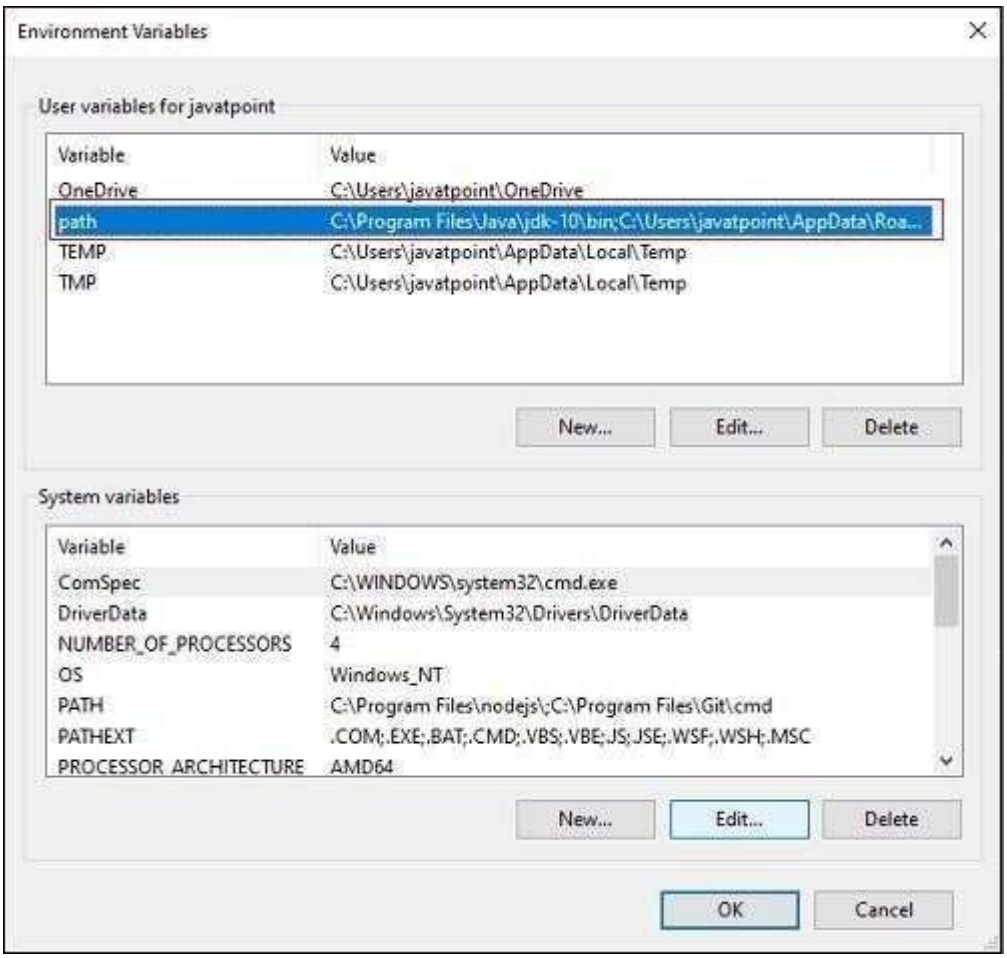


Step 2: Next, to download the latest Flutter SDK, click on the Windows **icon**. Here, you will find the download link for SDK.

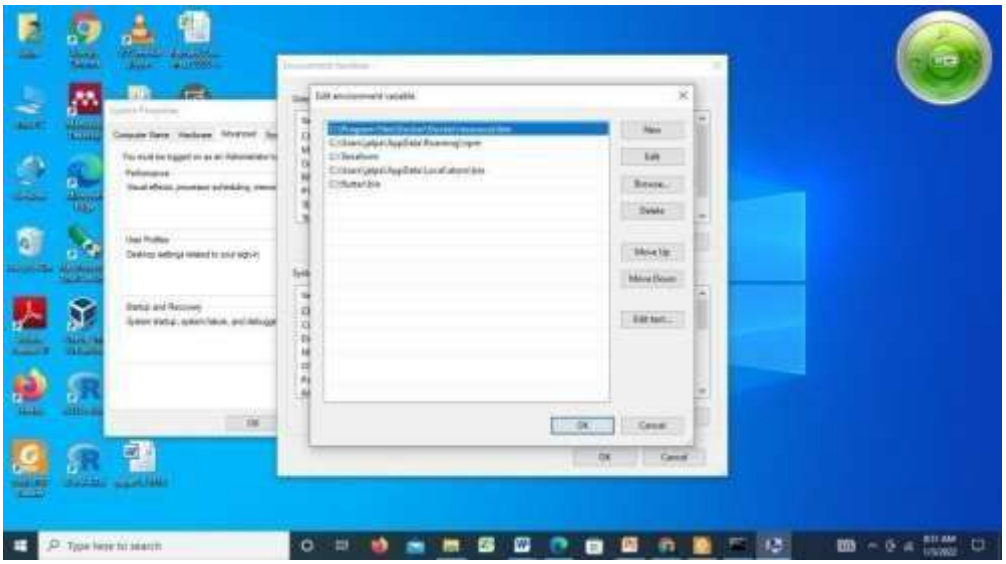
Step 3: When your download is complete, extract the **zip** file and place it in the desired installation folder or location, for example, C: /Flutter.

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.

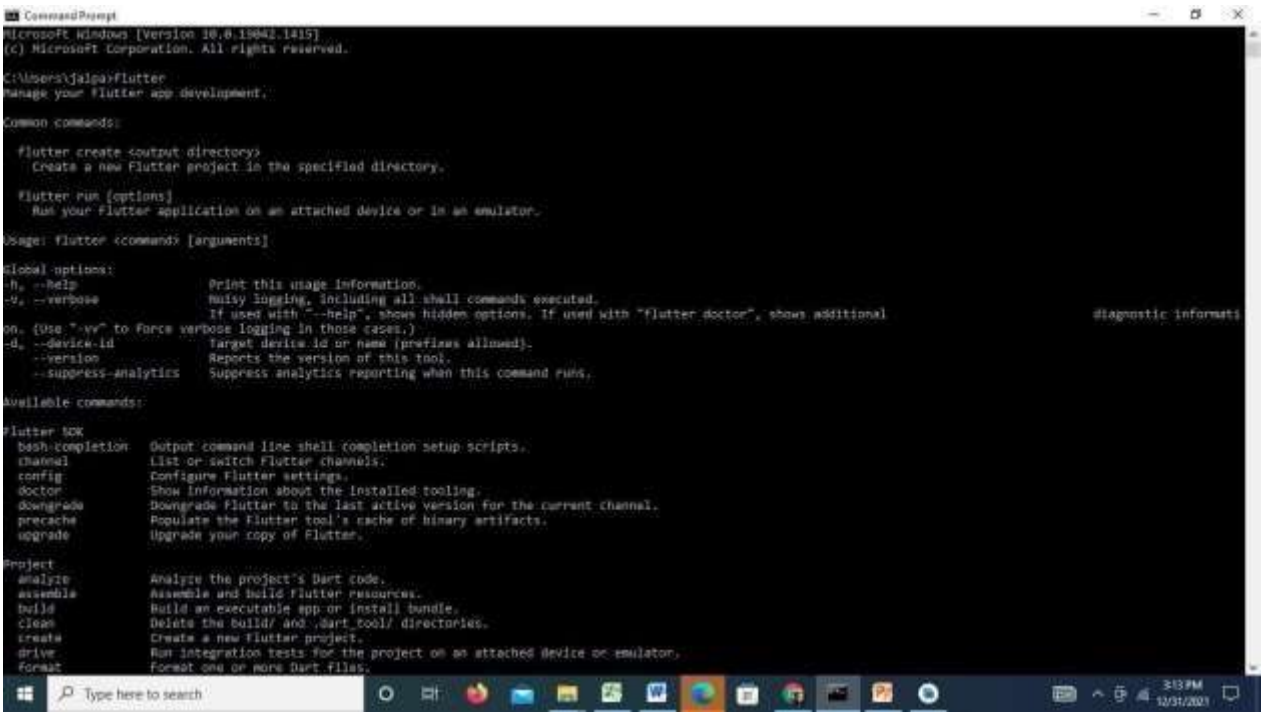


Step 4.2: Now, select path -> click on edit. The following screen appears

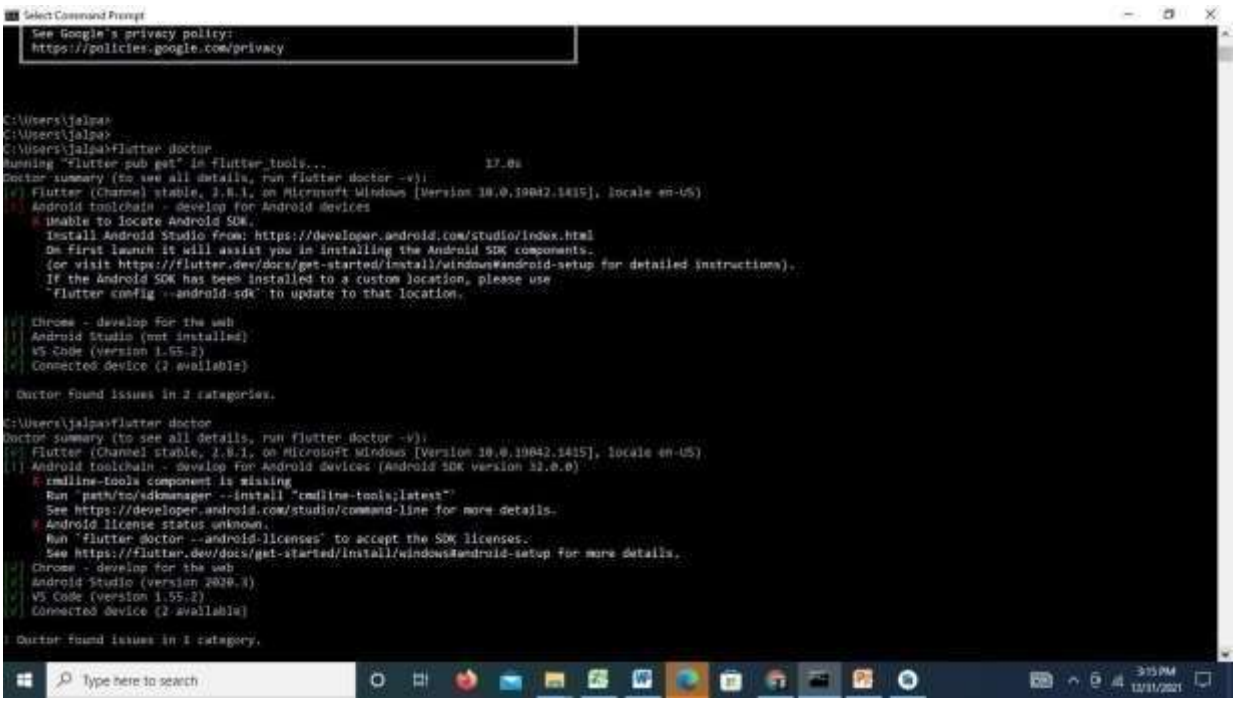


Step 4.3: In the above window, click on New->write path of Flutter bin folder in variable value > ok -> ok -> ok.

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.



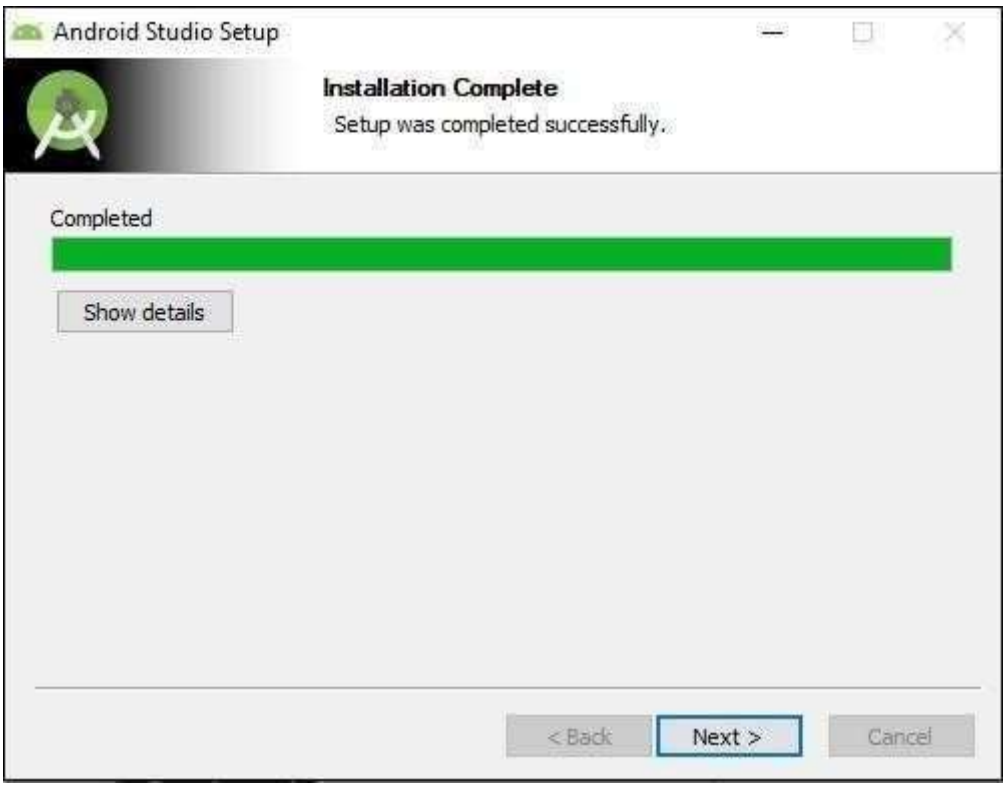
Step 6: When you run the above command, it will analyze the system and show its report, as shown in the below image. Here, you will find the details of all missing tools, which required to run Flutter as well as the development tools that are available but not connected with the device.

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.

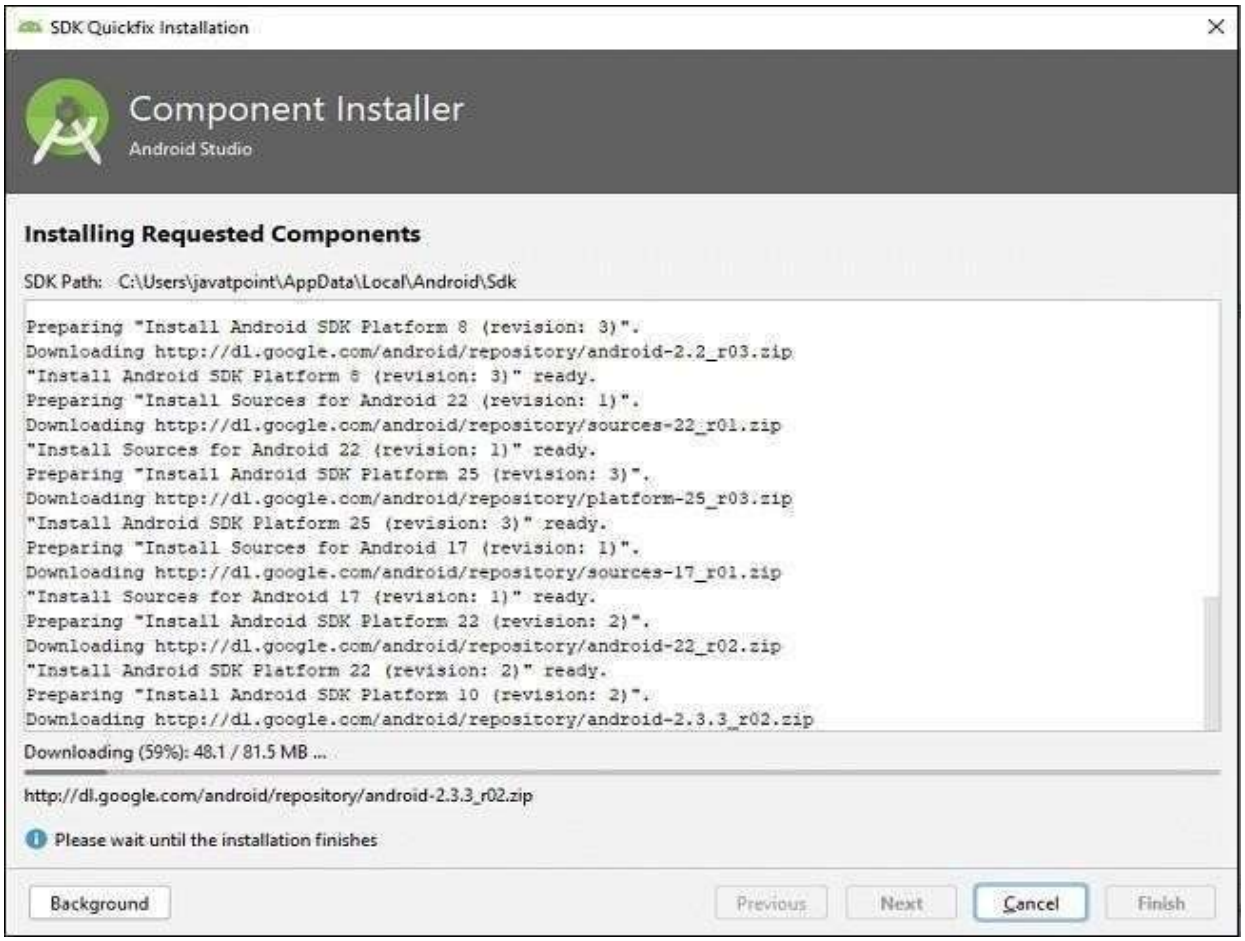
Step 7.1: Download the latest Android Studio executable or zip file from the official site.

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

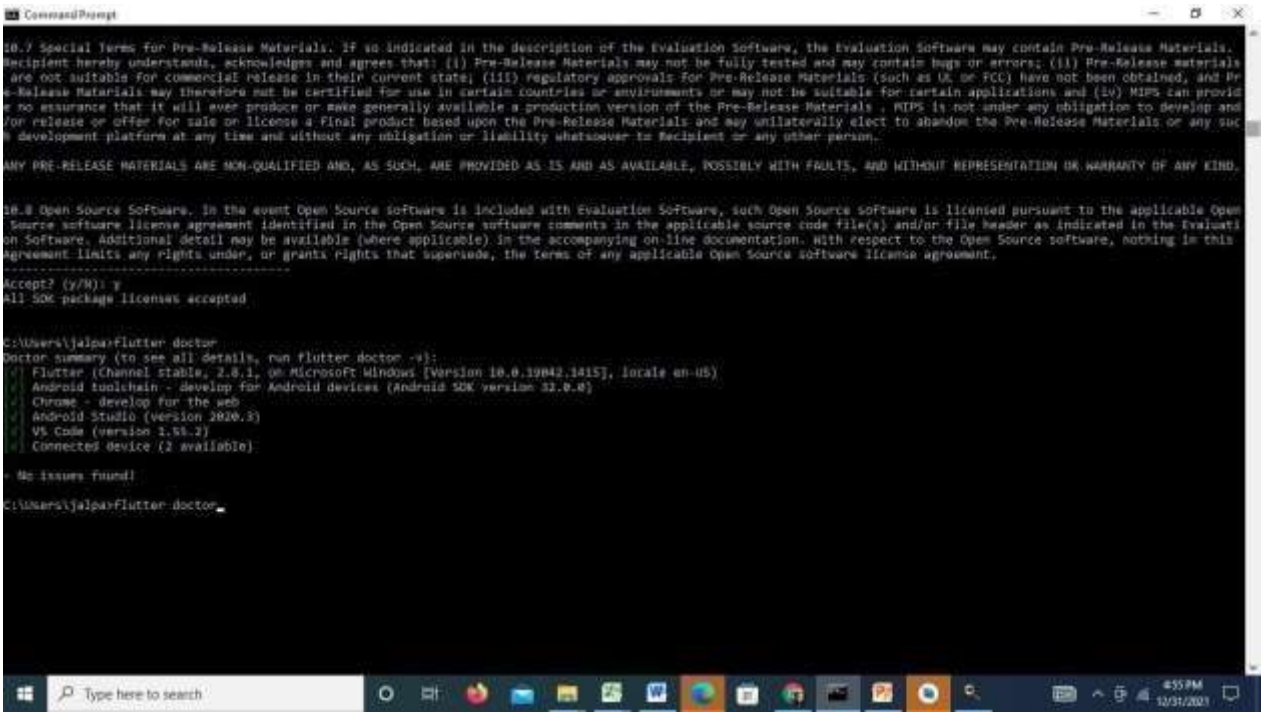
Step 7.3: Follow the steps of the installation wizard. Once the installation wizard completes, you will get the following screen.



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.

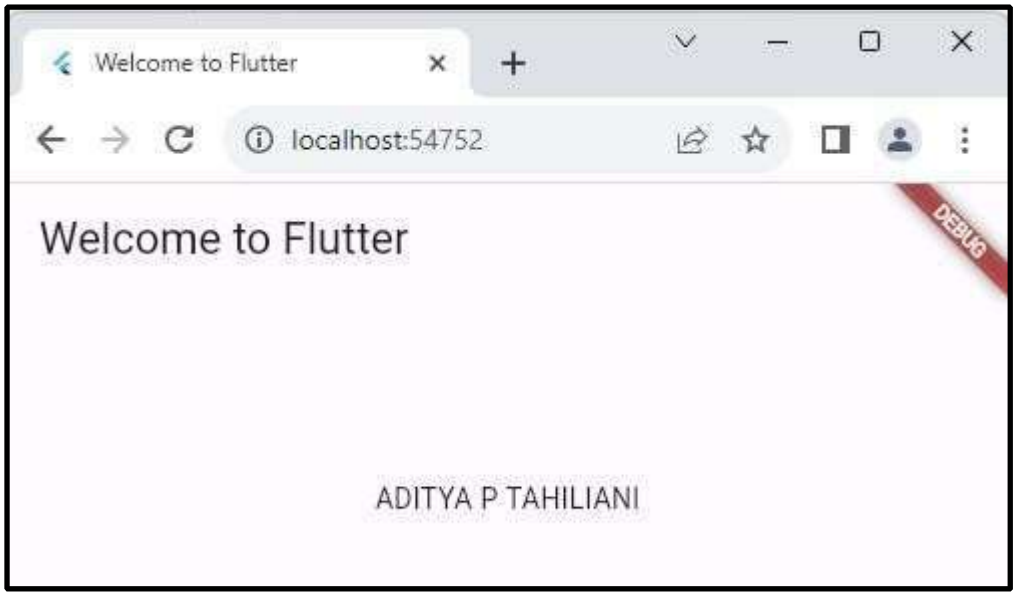
Step 9.2: Now, search the Flutter plugin. If found, select Flutter plugin and click install. When you click on install, it will ask you to install Dart plugin as below screen. Click yes to proceed.

Step 9.3: Restart the Android Studio.

Code : To print the *hello your_name* in the output.

```
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('ADITYA P TAHILIANI'),
       ),
     ),
   );
 }

}
```

Output :



MAD & PWA Lab
Journal

Experiment No.	02
Experiment Title.	To design Flutter UI by including common widgets.
Roll No.	59
Name	**Aditya P Tahiliani**
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

Aditya P Tahiliani
D15A_59

Batch :- C

Experiment no. :- 02

Aim :- **To design Flutter UI by including common widgets.**

Theory :-

In Flutter, widgets are the building blocks of the user interface, and several common widgets play crucial roles in creating engaging and interactive applications. Here's a brief overview of some fundamental Flutter widgets:

Text: Flutter's Text widget is used to display text on the screen. It supports various styles such as font size, font weight, color, alignment, and more. You can use it to display static text as well as dynamic text generated at runtime.

Image: The Image widget is used to display images in a Flutter application. It supports various image sources, including network images, local images, and even memory images. You can customize the image's dimensions, alignment, and more.

Icon: Flutter's Icon widget is used to display graphical icons from an icon library such as Material Icons or FontAwesomeIcons. Icons can be customized with properties like size, color, and alignment. They're commonly used for indicating actions or representing UI elements.

Container: The Container widget is a versatile widget used to create rectangular visual elements. It can contain a single child widget and supports styling options like color, padding, margin, border, and more. Containers are often used for layout purposes and to apply styling to other widgets.

Row: The Row widget arranges its children widgets horizontally in a single line. It's commonly used for laying out multiple widgets side by side. You can control the alignment, spacing, and size of the children widgets within the Row.

Column: Similar to Row, the Column widget arranges its children widgets vertically in a single line. It's useful for creating vertical layouts such as lists or forms. Like Row, you have control over the alignment, spacing, and size of the children widgets within the Column

ListView: The ListView widget is used to create scrollable lists of widgets. It's particularly useful when dealing with a large number of items that need to be displayed within limited screen space. ListView can display its children vertically or horizontally and supports both static and dynamic lists.

Code :-

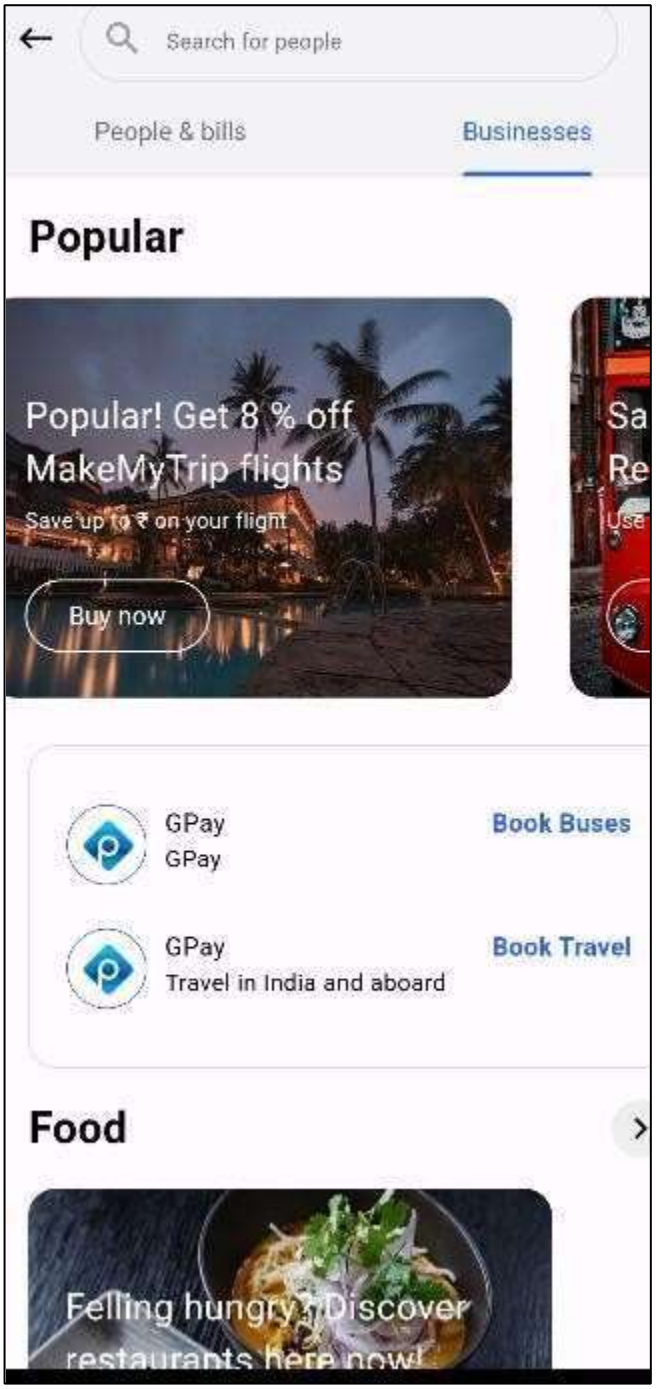
```
import 'package:flutter/material.dart';
import 'package:flutter_to_do_list/const/colors.dart';
import 'package:flutter_to_do_list/data/firestor.dart';
import 'package:flutter_to_do_list/model/notes_model.dart';
import 'package:flutter_to_do_list/screen/edit_screen.dart';


class Task_Widget extends StatefulWidget
 {Note  _note;
 Task_Widget(this._note, {super.key});

 @override
 State<Task_Widget> createState() => _Task_WidgetState();
}

class _Task_WidgetState extends State<Task_Widget>
 {@override
 Widget build(BuildContext  context)
  {bool isDone = widget._note.isDon;
  return Padding(
    padding: const EdgeInsets.symmetric(horizontal:  15, vertical: 10),child:
    Container(
```

```
width: double.infinity,
height: 130,
decoration:
 BoxDecoration( borderRadius:
 BorderRadius.circular(10),color:
 Colors.white,
 boxShadow:
  [BoxShadow
  (
   color: Colors.grey.withOpacity(0.2),
   spreadRadius: 5,
   blurRadius: 7,
   offset: Offset(0, 2),
  ),
 ],
),
child: Padding(
 padding: const EdgeInsets.symmetric(horizontal: 10),
 child: Row(
  children: [
   // image
   imageee(),
   SizedBox(width: 25),
   // title and subtitle
   Expanded(
    child: Column(
     crossAxisAlignment: CrossAxisAlignment.start,
     children: [
```

```
SizedBox(height: 5),
Row(
  mainAxisAlignment:  MainAxisAlignment.spaceBetween,
  children: [
    Text( widget._note.
      title,style:
      TextStyle( fontSize:
      18,
        fontWeight: FontWeight.bold,
      ),
    ),
    Checkbox(
      activeColor: custom_green,
      value: isDone,
      onChanged:  (value)
        {setState(() {
          isDone = !isDone;
        });
        Firestore_Datasource()
            .isdone(widget._note.id,  isDone);
      },
    )
  ],
),
Text( widget._note.subtitl
  e,
```

```
style:
    TextStyle(fon
    tSize: 16,
    fontWeight: FontWeight.w400,
    color: Colors.grey.shade400),
),
```

Screenshot :-



MAD & PWA Lab
Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	59
Name	**Aditya P Tahiliani**
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	10

Aditya P Tahiliani
D15A_59

Batch :- C

Experiment no. :- 03

Aim- **To include icon, images, fonts in Flutter APP**

Theory :-

1) Button: the Button widget is not a specific widget, but rather a category of widgets that are used to handle user interaction by triggering actions when pressed. Some commonly used button widgets include: Elevated Button , Textfield Button, Outlined button etc

2) Textfield with Icon: In Flutter, a TextField widget is used to allow users to input text. It is a fundamental part of many forms and input-based user interfaces. TextField provides a text input area where users can enter and edit text, and it comes with various customization options.

3) Image : This widget holds the image which can fetch it from multiple sources like from the asset folder or directly from the URL. To add an image in the project, you need first to create an assets folder where you keep your images and then add the below line in pubspec.yaml file.

4) Gesture Detection: To make an image interactive like a button, you need to detect user gestures such as taps. Flutter provides gesture detection widgets like GestureDetector .These widgets allow you to listen for various touch events like taps, swipes, and drags.I used this widget to make image as a button.

5) Icon Button: Flutter provides the IconButton widget, which combines an icon with a tappable area, making it easy to create interactive icons that respond to user taps. The IconButton widget is commonly used for actions like navigation, opening menus, submitting forms, etc.

Code :-

```
import 'package:flutter/material.dart';
import 'package:flutter_to_do_list/const/colors.dart';
import 'package:flutter_to_do_list/data/auth_data.dart';


class LogIN_Screen extends StatefulWidget
 {final VoidCallback show;
 LogIN_Screen(this.show, {super.key});

 @override
 State<LogIN_Screen> createState() => _LogIN_ScreenState();
}

class _LogIN_ScreenState extends State<LogIN_Screen>
 {FocusNode _focusNode1 = FocusNode();
 FocusNode _focusNode2 = FocusNode();

 final email = TextEditingController();
 final password = TextEditingController();

 @override
 void initState() {
  // TODO: implement initState
  super.initState();
  _focusNode1.addListener(()
    {setState(() {});
```

```
  });
  super.initState();
  _focusNode2.addListener(()
    {setState(() {});
  });
}

@override
Widget build(BuildContext context)
  {return Scaffold(
    backgroundColor: backgroundColors,
    body: SafeArea(
      child:
        SingleChildScrollView(child:
        Column(
          children:
            [ SizedBox(height:
            20),image(),
            SizedBox(height: 50),
            textfield(email, _focusNode1, 'Email', Icons.email),
            SizedBox(height: 10),
            textfield(password, _focusNode2, 'Password', Icons.password),
            SizedBox(height: 8),
            account(),
            SizedBox(height: 20),
            Login_bottom(),
          ],
```

```
Widget account()
 {return Padding(
  padding: const EdgeInsets.symmetric(horizontal: 15),
  child: Row(
   mainAxisAlignment: MainAxisAlignment.end,
   children: [
    Text(
     "Don't have an account?",
     style: TextStyle(color: Colors.grey[700], fontSize: 14),
    ),
    SizedBox(width: 5),
    GestureDetector( onTa
    p: widget.show, child:
    Text(
      'Sign UP',
      style:
        TextStyle( color:
        Colors.blue,
        fontSize: 14,
        fontWeight: FontWeight.bold),
     ),
```

```
Widget Login_bottom()
 {return Padding(
  padding: const EdgeInsets.symmetric(horizontal: 15),
  child: GestureDetector(
   onTap: () {
    AuthenticationRemote().login(email.text,  password.text);
   },
   child: Container(
    alignment: Alignment.center,
    width: double.infinity, height:
    50,
    decoration:
     BoxDecoration(color:
     custom_green,
     borderRadius: BorderRadius.circular(10),
    ),
    child:
     Text('Logl
     n',
     style:
      TextStyle( color:
      Colors.white,
```

```
            fontWeight: FontWeight.bold,
),
  Widget textfield(TextEditingController _controller, FocusNode _focusNode,
    String typeName, IconData iconss) {
   return Padding(
    padding: const EdgeInsets.symmetric(horizontal: 15),
    child: Container(
     decoration:
      BoxDecoration(color:
      Colors.white,
      borderRadius: BorderRadius.circular(15),
     ),
     child:
      TextField( controller:
      _controller, focusNode:
      _focusNode,
      style: TextStyle(fontSize:18, color: Colors.black),
      decoration: InputDecoration(
        prefixIcon:
         Icon(iconss,
         color: _focusNode.hasFocus ? custom_green : Color(0xffc5c5c5),
        ),
```

```
contentPadding:
    EdgeInsets.symmetric(horizontal: 15, vertical: 15),hintText:
typeName,
enabledBorder:
  OutlineInputBorder( borderRadius:
  BorderRadius.circular(10),borderSide:
  BorderSide(
    color: Color(0xffc5c5c5),
    width: 2.0,
  ),
),
focusedBorder:
  OutlineInputBorder( borderRadius:
  BorderRadius.circular(10),borderSide:
  BorderSide(
```

Output :-


1:58
People
Ria
John
Mohan
Sara
Noyura
Diya
Neha
Mohan
Bob
Judith
Lora
Ria
Explore
New Payment
Spots
Essentials
Shopping

MAD & PWA Lab
Journal

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget
Roll No.	59
Name	**Aditya P Tahiliani**
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	10

Aditya P Tahiliani
D15A_59

Batch :- C

Experiment no. :- 04

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, placeholdertext, 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 :-

```
import 'package:flutter/material.dart';
import 'package:flutter_to_do_list/const/colors.dart';
import 'package:flutter_to_do_list/data/auth_data.dart';


class SignUp_Screen extends StatefulWidget
 {final VoidCallback show;
 SignUp_Screen(this.show, {super.key});


 @override
 State<SignUp_Screen> createState() => _SignUp_ScreenState();
}


class _SignUp_ScreenState extends State<SignUp_Screen>
 {FocusNode _focusNode1 = FocusNode();
 FocusNode _focusNode2 = FocusNode();
 FocusNode _focusNode3 = FocusNode();


 final email = TextEditingController();
 final password = TextEditingController();
 final PasswordConfirm = TextEditingController();


 @override
 void initState() {
  // TODO: implement initState
  super.initState();
```

```
  _focusNode1.addListener(()
    {setState(() {});
  });
  _focusNode2.addListener(()
    {setState(() {});
  });
  _focusNode3.addListener(()
    {setState(() {});
  });
}

@override
Widget build(BuildContext context)
  {return Scaffold(
    backgroundColor: backgroundColors,
    body: SafeArea(
      child:
        SingleChildScrollView(child:
        Column(
          children:
            [ SizedBox(height:
            20),image(),
            SizedBox(height: 50),
            textfield(email, _focusNode1, 'Email', Icons.email),
            SizedBox(height: 10),
            textfield(password, _focusNode2, 'Password', Icons.password),
            SizedBox(height: 10),
```

```
            textfield(PasswordConfirm, _focusNode3, 'PasswordConfirm',
                Icons.password),
            SizedBox(height: 8),
            account(),
            SizedBox(height: 20),
            SignUP_bottom(),
          ],
        ),
      ),
    ),
  );
}

Widget account()
  {return Padding(
    padding:  const EdgeInsets.symmetric(horizontal: 15),
    child: Row(
      mainAxisAlignment: MainAxisAlignment.end,
      children: [
        Text(
          "Don you have an account?",
          style: TextStyle(color: Colors.grey[700], fontSize: 14),
        ),
        SizedBox(width: 5),
        GestureDetector(  onTa
        p: widget.show,
```

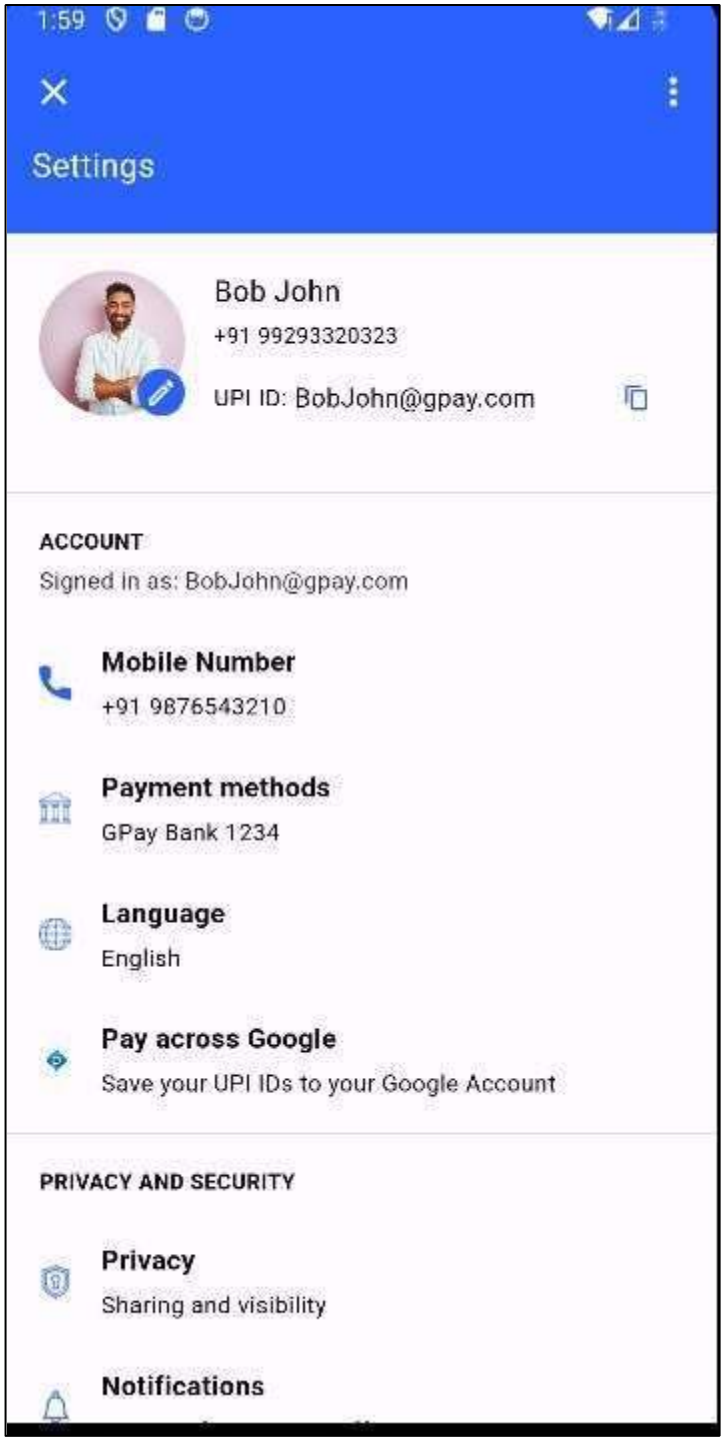
```
        child:
          Text('Logi
          n',
          style:
              TextStyle( color:
              Colors.blue,
              fontSize: 14,
              fontWeight: FontWeight.bold),
        ),
      )
    ],
  ),
 );
}

Widget SignUP_bottom()
 {return Padding(
   padding:  const EdgeInsets.symmetric(horizontal:  15),
   child: GestureDetector(
     onTap: ()
      { AuthenticationRemote
      ()
          .register(email.text, password.text, PasswordConfirm.text);
     },
     child: Container(
       alignment: Alignment.center,
       width: double.infinity, height:
       50,
       decoration: BoxDecoration(
```

```
          color: custom_green,
          borderRadius: BorderRadius.circular(10),
        ),
        child:
          Text('Sign
          Up',
          style:
            TextStyle( color:
            Colors.white,
            fontSize: 23,
            fontWeight: FontWeight.bold,
          ),
        ),
      ),
    ),
  );
}

Widget textfield(TextEditingController _controller, FocusNode _focusNode,
    String typeName, IconData iconss) {
  return Padding(
    padding: const EdgeInsets.symmetric(horizontal: 15),
    child: Container(
      decoration:
        BoxDecoration(color:
        Colors.white,
        borderRadius: BorderRadius.circular(15),
      ),
```

```
controller: _controller,
focusNode: _focusNode,
style: TextStyle(fontSize: 18, color: Colors.black),
decoration: InputDecoration(
  prefixIcon:
   Icon(iconss,
   color: _focusNode.hasFocus ? custom_green : Color(0xffc5c5c5),
  ),
  contentPadding:
    EdgeInsets.symmetric(horizontal: 15, vertical: 15),hintText:
  typeName,
  enabledBorder:
   OutlineInputBorder( borderRadius:
   BorderRadius.circular(10),borderSide:
   BorderSide(
    color: Color(0xffc5c5c5),
    width: 2.0,
   ),
  ),
  focusedBorder:
   OutlineInputBorder( borderRadius:
   BorderRadius.circular(10),borderSide:
   BorderSide(
    color: custom_green,
    width: 2.0,
   ),
  )),
),
```

```
      ),
    );
  }

  Widget image()
   {return
   Padding(
     padding: const EdgeInsets.symmetric(horizontal: 15),
     child: Container(
       width: double.infinity,
       height: 300,
       decoration:
       BoxDecoration(color:
         backgroundColors, image:
         DecorationImage(
           image: AssetImage('images/7.png'),
           fit: BoxFit.fitWidth,
         ),
       ),
     ),
    );
  }
}
```

Output :-


1:59
Settings
Bob John
+91 99293320323
UPI ID: BobJohn@gpay.com
ACCOUNT
Signed in as: BobJohn@gpay.com
Mobile Number
+91 9876543210
Payment methods
GPay Bank 1234
Language
English
Pay across Google
Save your UPI IDs to your Google Account
PRIVACY AND SECURITY
Privacy
Sharing and visibility
Notifications


2:01
Enter Bank Details
Account Number
12345667
Re-enter account number
12345667
Search for IFSC
IFSC
ADITYA P TAHILIANI
Recipient name
ADITYA P TAHILIANI
This information will be securely saved as per the GPay Terms of Services and Privacy Policy.
Continue

MAD & PWA Lab
Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	59
Name	**Aditya P Tahiliani**
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	10

Aditya P Tahiliani
D15A_59

Batch :- C

Experiment no. :- 05

Aim :- : **To apply navigation,routing and gestures in Flutter.**

Theory :-

In Flutter, navigation, routing, and gestures are essential concepts for creating interactive and navigable user interfaces.

Navigation: Navigation refers to the process of moving between different screens or pages within a Flutter app. Flutter provides the Navigator widget for managing navigation and routing.

Routing: Routing is the mechanism used to define the paths or routes between different screens in your app. Each route typically corresponds to a different widget or screen in your app.

Gesture Detection: Gestures allow users to interact with the app by tapping, dragging, swiping, or performing other touch-based actions. Flutter provides various gesture detection widgets to handle user input.

```
GestureDetector(
onTap: () {
print('Container tapped');
 },
child:
Container( widt
h: 200,
height: 200,
color: Colors.blue,
child: Center(
 child: Text('Tap Me'),
  ),
 ),
)
```

Code :-

```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:flutter_to_do_list/const/colors.dart';
import 'package:flutter_to_do_list/data/firestor.dart';

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

 @override
 State<Add_creen> createState() => _Add_creenState();
}

class _Add_creenState extends State<Add_creen>
 { final title = TextEditingController();
 final subtitle = TextEditingController();

 FocusNode _focusNode1 = FocusNode();
 FocusNode _focusNode2 = FocusNode();
 int indexx = 0;
 @override
 Widget build(BuildContext context)
  { return Scaffold(
    backgroundColor: backgroundColors,
    body: SafeArea(
     child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
       title_widgets(),
       SizedBox(height: 20),
       subtite_wedgite(),
       SizedBox(height: 20),
       imagess(),
       SizedBox(height: 20),
       button()
      ],
     ),
```

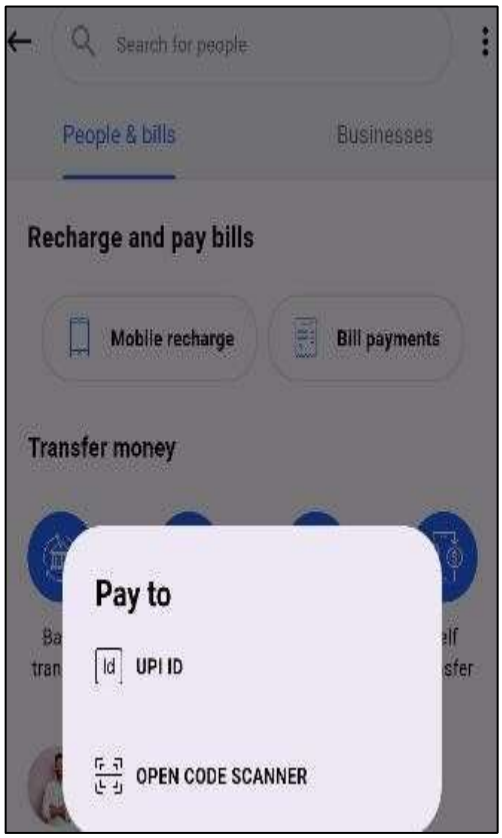
```
    ),
  );
}

Widget button()
 { return Row(
   mainAxisAlignment: MainAxisAlignment.spaceAround,
   children: [
    ElevatedButton(
    style:
       ElevatedButton.styleFrom( primary:
       custom_green, minimumSize:
       Size(170, 48),
     ),
     onPressed: () {
       Firestore_Datasource().AddNote(subtitle.text, title.text, indexx);
       Navigator.pop(context);
     },
     child: Text('add task'),
    ),
    ElevatedButton(
     style:
       ElevatedButton.styleFrom( primar
       y: Colors.red, minimumSize:
       Size(170, 48),
     ),
     onPressed: ()
       { Navigator.pop(context);
     },
     child: Text('Cancel'),
    ),
   ],
  );
}

Container imagess()
 { return
 Container( height:
 180,
   child: ListView.builder(
```

```
      itemCount: 4,
      scrollDirection: Axis.horizontal,
      itemBuilder: (context, index)
      { return GestureDetector(
          onTap: () {
            setState(()
              { indexx = index;
            });
          },
          child: Padding(
            padding: EdgeInsets.only(left: index == 0 ? 7 : 0),
            child: Container(
              decoration: BoxDecoration( borderRadius:
                BorderRadius.circular(10), border:
                Border.all(
                  width: 2,
                  color: indexx == index ? custom_green : Colors.grey,
                ),
              ),
              width: 140,
              margin: EdgeInsets.all(8),
              child: Column(
                children:
                  [ Image.asset('images/${index}.png'),
                ],
              ),
            ),
          ),
        );
      },
    ),
  );
}

Widget title_widgets()
  { return Padding(
    padding: const EdgeInsets.symmetric(horizontal: 15),
```

```
    child: Container( decoration:
      BoxDecoration( color:
      Colors.white,
        borderRadius: BorderRadius.circular(15),
      ),
      child:
        TextField( cont
        roller: title,
        focusNode: _focusNode1,
        style: TextStyle(fontSize: 18, color: Colors.black),
        decoration: InputDecoration(
          contentPadding:
            EdgeInsets.symmetric(horizontal: 15, vertical: 15),
          hintText: 'title',
          enabledBorder:
            OutlineInputBorder( borderRadius:
            BorderRadius.circular(10), borderSide:
            BorderSide(
              color: Color(0xffc5c5c5),
              width: 2.0,
            ),
          ),
          focusedBorder:
            OutlineInputBorder( borderRadius:
            BorderRadius.circular(10), borderSide:
            BorderSide(
              color: custom_green,
              width: 2.0,
            ),
          )),
      ),
    ),
  );
}

Padding subtite_wedgite()
  { return Padding(
    padding: const EdgeInsets.symmetric(horizontal: 15),
    child: Container(
      decoration: BoxDecoration(
```

```
          color: Colors.white,
          borderRadius: BorderRadius.circular(15),
        ),
        child:
          TextField( maxLine
          s: 3, controller:
          subtitle,
          focusNode: _focusNode2,
          style: TextStyle(fontSize: 18, color: Colors.black),
          decoration: InputDecoration(
            contentPadding: EdgeInsets.symmetric(horizontal: 15, vertical: 15),
            hintText: 'subtitle',
            enabledBorder:
              OutlineInputBorder( borderRadius:
              BorderRadius.circular(10), borderSide:
              BorderSide(
                color: Color(0xffc5c5c5),
                width: 2.0,
              ),
            ),
            focusedBorder:
              OutlineInputBorder( borderRadius:
              BorderRadius.circular(10), borderSide:
              BorderSide(
                color: custom_green,
                width: 2.0,
              ),
            ),
          ),
        ),
      ),
    );
  }
}
```

Output :-


Search for people
People & bills
Businesses
Recharge and pay bills
Mobile recharge
Bill payments
Transfer money
Pay to
UPI ID
OPEN CODE SCANNER


2:11
28%
Scan QR code to pay
Scan Spot Code to connect
Show my Spot Code

MAD & PWA Lab
Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	59
Name	**Aditya P Tahiliani**
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	10

Aditya P Tahiliani **Batch :- C**

D15A_59

Experiment no. :- 06

Aim: To Connect Firebase Database With Flutter UI

Theory:

Firebase is a great backend solution for anyone that wants to use authentication, databases, cloud functions, ads, and countless other features within an app.

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

This tutorial was verified with Flutter v2.0.6, Android SDK v31.0.2, and Android Studio v4.1.

Creating a New Flutter Project

This tutorial will require the creation of an example Flutter app.

Once you have your environment set up for Flutter, you can run the following to create a new application:

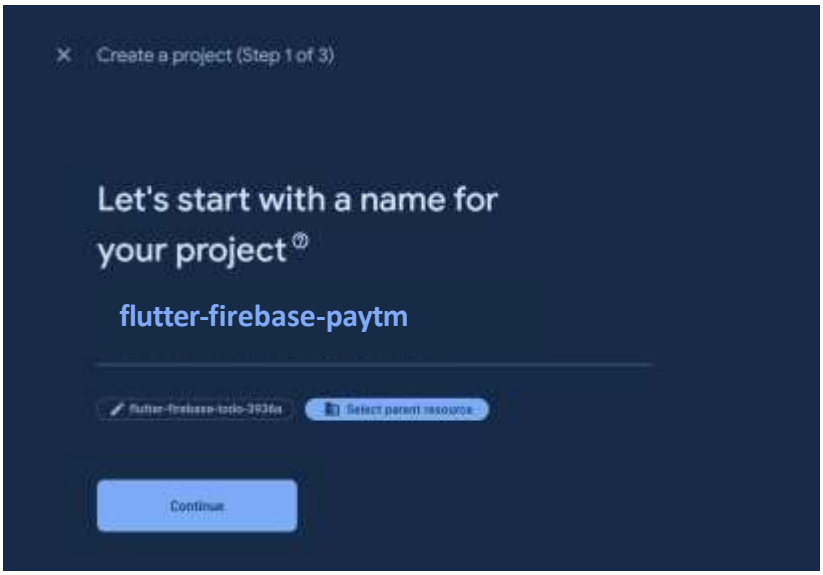
flutter create flutterfirebaseexample

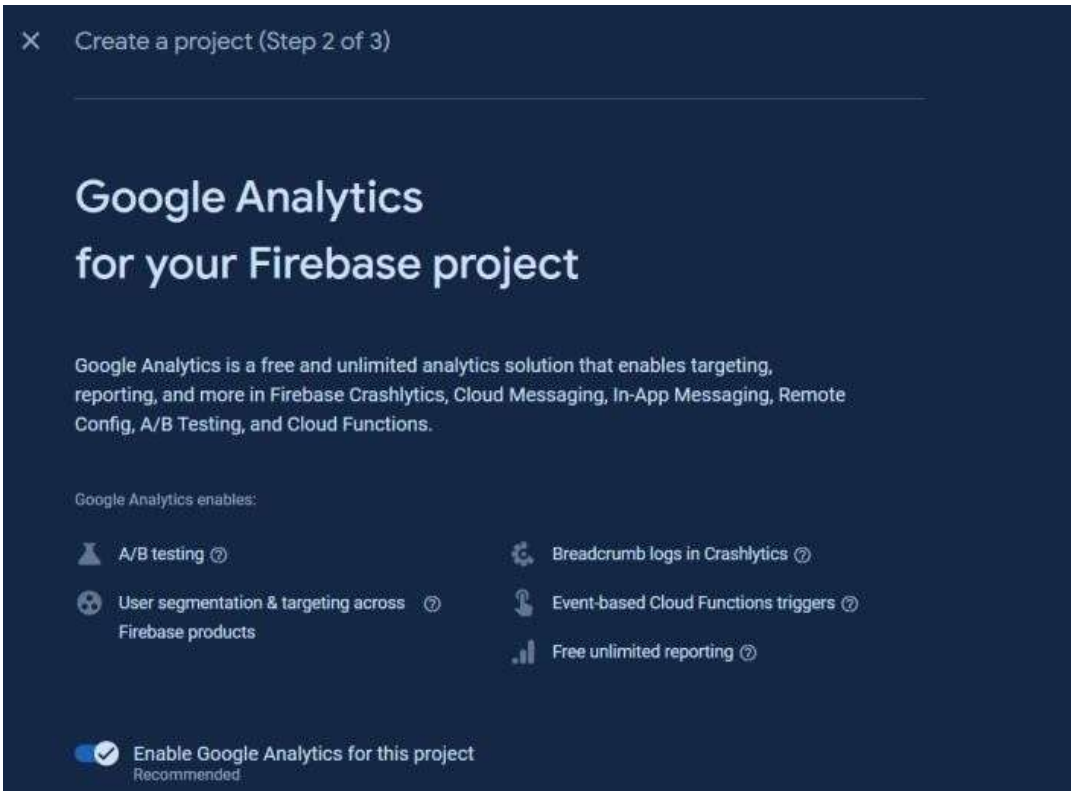
Using flutter create will produce a demo application that will display the number of times a button is clicked.

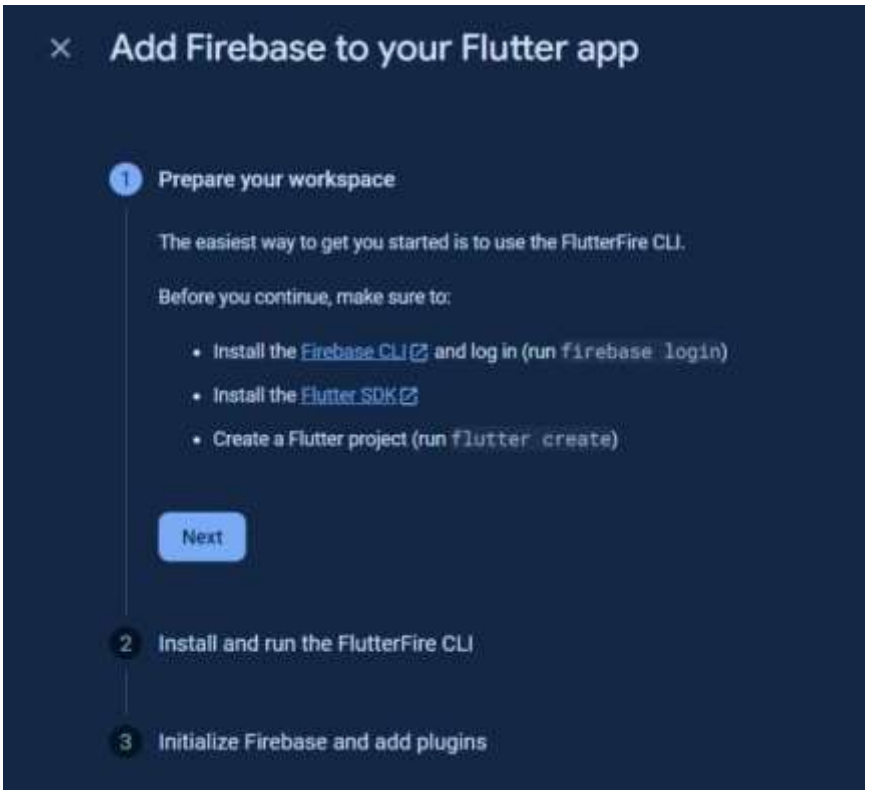
Now that we've got a Flutter project up and running, we can add Firebase.

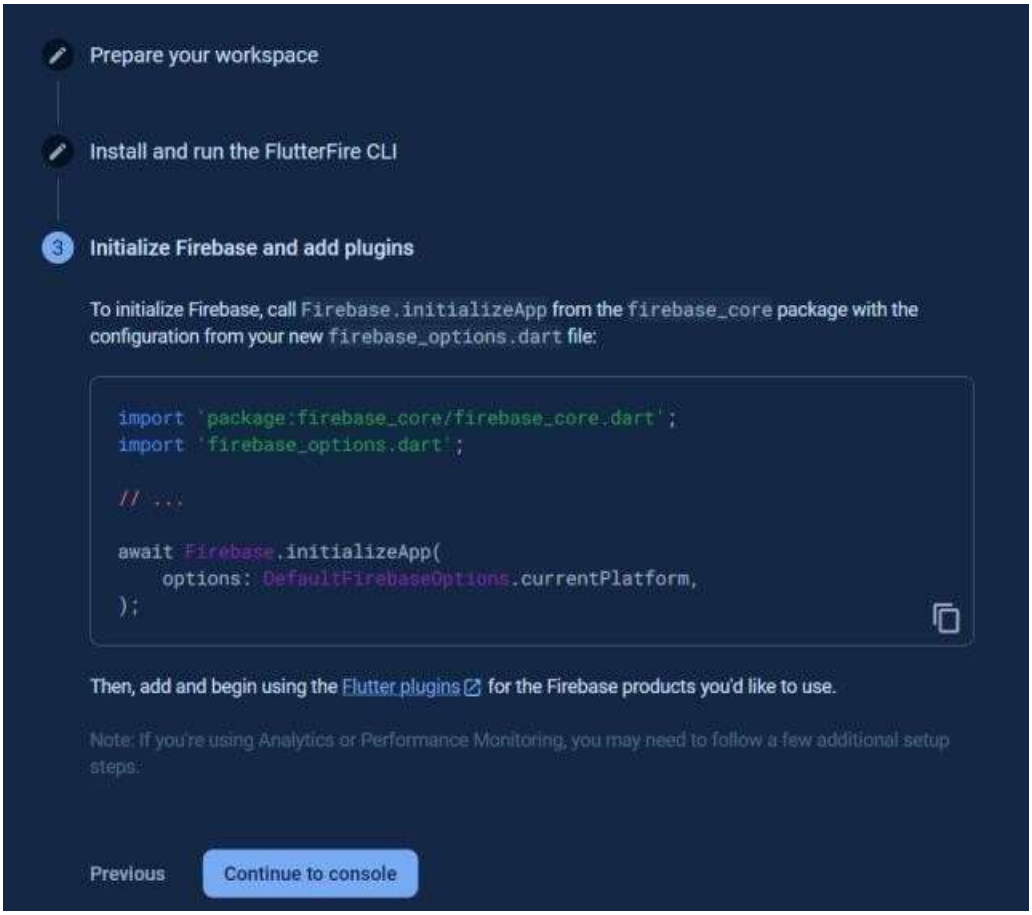
Creating a New Firebase Project

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:

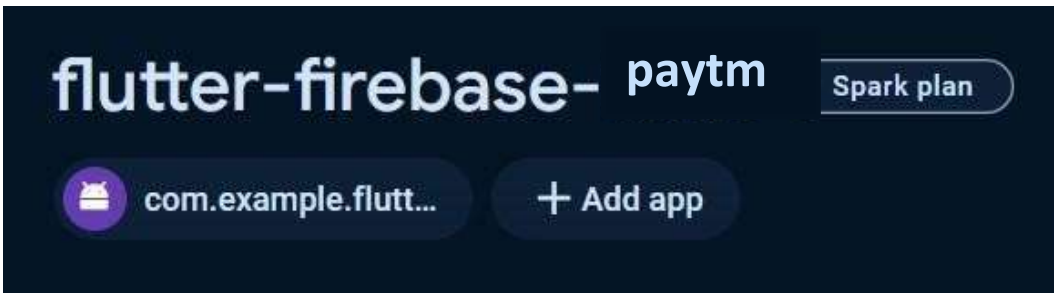





Add Firebase to your Flutter app
1
Prepare your workspace
The easiest way to get you started is to use the FlutterFire CLI.
Before you continue, make sure to:
Install the Firebase CLI and log in (run firebase login)
Install the Flutter SDK
Create a Flutter project (run flutter create)
Next
2
Install and run the FlutterFire CLI
3
Initialize Firebase and add plugins


Prepare your workspace
Install and run the FlutterFire CLI
3
Initialize Firebase and add plugins
To initialize 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,
);
Then, add and begin using the Flutter plugins for the Firebase products you'd like to use.
Note: If you're using Analytics or Performance Monitoring, you may need to follow a few additional setup steps.
Previous
Continue to console

```yaml
dependencies:
  any_link_preview: ^3.0.1
  cloud_firestore: ^4.14.0
  cupertino_icons: ^1.0.2
  dotted_border: ^2.1.0
  file_picker: ^6.1.1
  firebase_auth: ^4.16.0
  firebase_core: ^2.24.2
  firebase_storage: ^11.6.0
  flutter:
    sdk: flutter
  flutter_riverpod: ^2.4.9
  fpdart: ^1.1.0
  google_sign_in: ^6.2.1
  routemaster: ^1.0.1
  shared_preferences: ^2.2.2
  uuid: ^4.3.3
```



Conclusion :- Thus, we successfully Connected Firebase Database With Flutter User Interface.

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.	59
Name	**Aditya P Tahiliani**
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	15

Aditya P. Tahiliani
59, D15-A

Experiment - 7

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

Theory:

Regular Web App

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

Progressive Web App

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

Difference between PWAs vs. Regular Web Apps:

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

1. Native Experience

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

2. Ease of Access

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

3. Faster Services

PWAs can cache the data and serve the user with text stylesheets, images, and other web content even before the page loads completely. This lowers the waiting time for the end-users and helps the

brands improve the user engagement and retention rate, which eventually adds value to their business.

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

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

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.

Code:

```
import { useEffect, useState } from 'react';
import './App.css';
import { v4 as uuidv4 } from 'uuid';
import Header from './components/Header';
import { ProjectList } from './components/ProjectList';
```

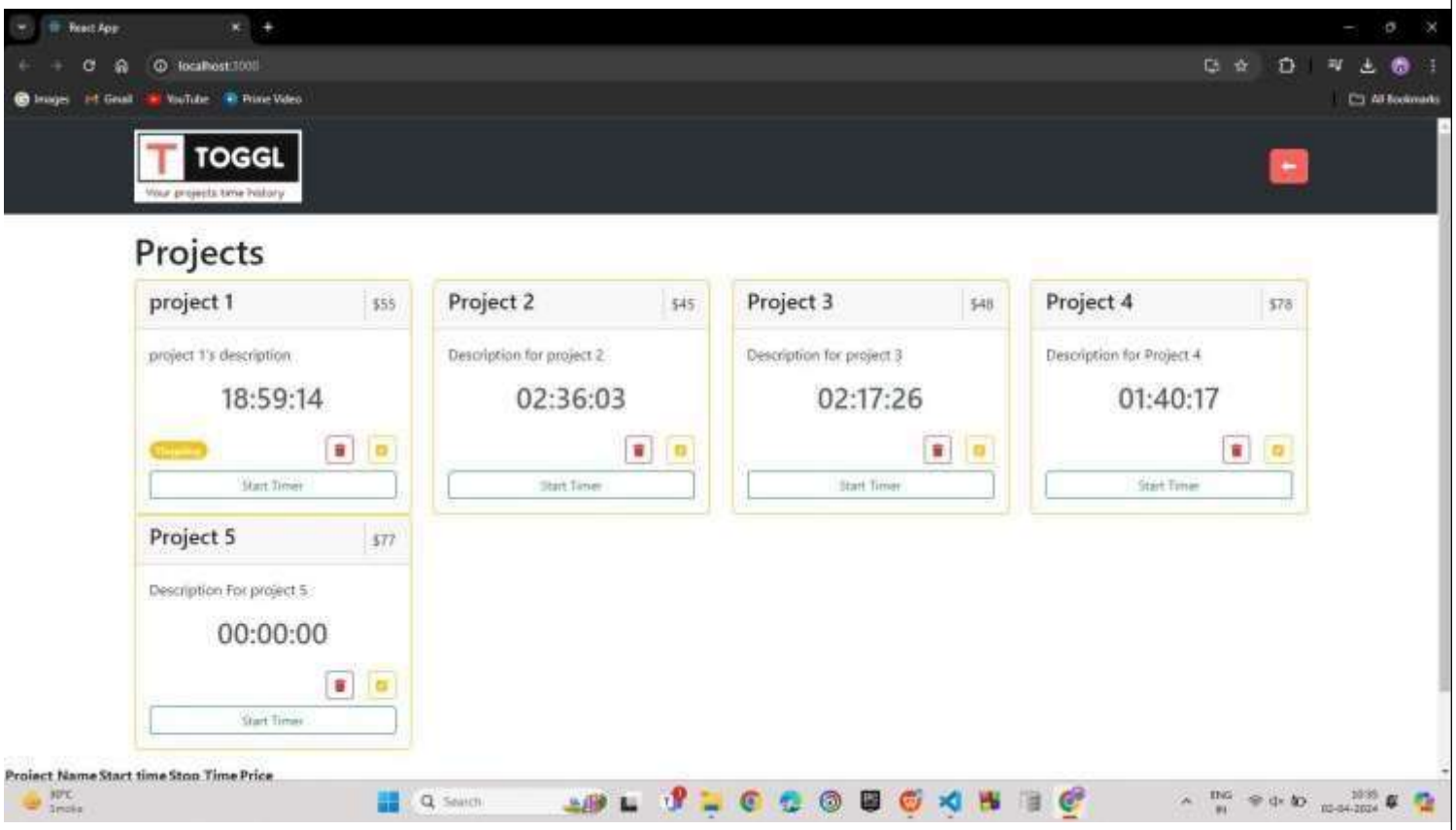
```
import { LogsTable } from './components/LogsTable';

function App() {
const [ projects, setProjects] =
useState(JSON.parse(localStorage.getItem('projects')) || []);
const [ isSidebarVisible, setIsSidebarVisible] = useState(false);
projects.map(project=> console.log(project.state))

useEffect(() => {
localStorage.setItem('projects', JSON.stringify(projects));
},[projects]);

const handleSidebarClose = () => setIsSidebarVisible(false);
const handleSidebarShow = () => setIsSidebarVisible(true);

const handleAddProject = (project) => {
  const projectToBeAdded = {...project, id: uuidv4(), elapsed:0,
runningSince:null,logs:[{Starttime:0,Endtime:0,Description:""}]};
setProjects([...projects, projectToBeAdded]);
setIsSidebarVisible(false);
}

const handleUpdateProject = (projectId, updatedProject) => {
  const updatedProjects = projects.map(project => {
  if(project.id === projectId) {
      return{
        ...updatedProject,
        elapsed: project.elapsed,
        runningSince: project.runningSince,
        id: projectId
      }
    }
    return project;
  });
  setProjects(updatedProjects);
}

const handleDeleteProject = (projectId) => {
  const updatedProjects = projects.filter(project => project.id !== projectId);
  setProjects(updatedProjects);
```

```
}

const handleStartTimer = (projectId) => {
  const updatedProjects = projects.map(project => project.id === projectId ?
{...project, runningSince: Date.now()}: project);
  setProjects(updatedProjects);
}
const handleStopTimer = (projectId) => {
  const updatedProjects = projects.map(project => {
    if(project.id !== projectId) return project;

    return {
      ...project,
      elapsed:project.elapsed + (Date.now() - project.runningSince),
      runningSince:null
    }
  });
  setProjects(updatedProjects);
}
  return (
    <>
      <Header
        onAddProject = { handleAddProject }
        onSidebarShow = { handleSidebarShow}
        onSidebarClose = { handleSidebarClose}
        isSidebarVisible = { isSidebarVisible }
      />
      <ProjectList
        onDeleteProject = { handleDeleteProject }
        projects={ projects }
        onSidebarShow= { handleSidebarShow }
        onStartTimer ={ handleStartTimer }
        onStopTimer = { handleStopTimer }
        onUpdateProject={ handleUpdateProject }
       />
       <LogsTable
        projects={projects}
       />
    </>
  );
```

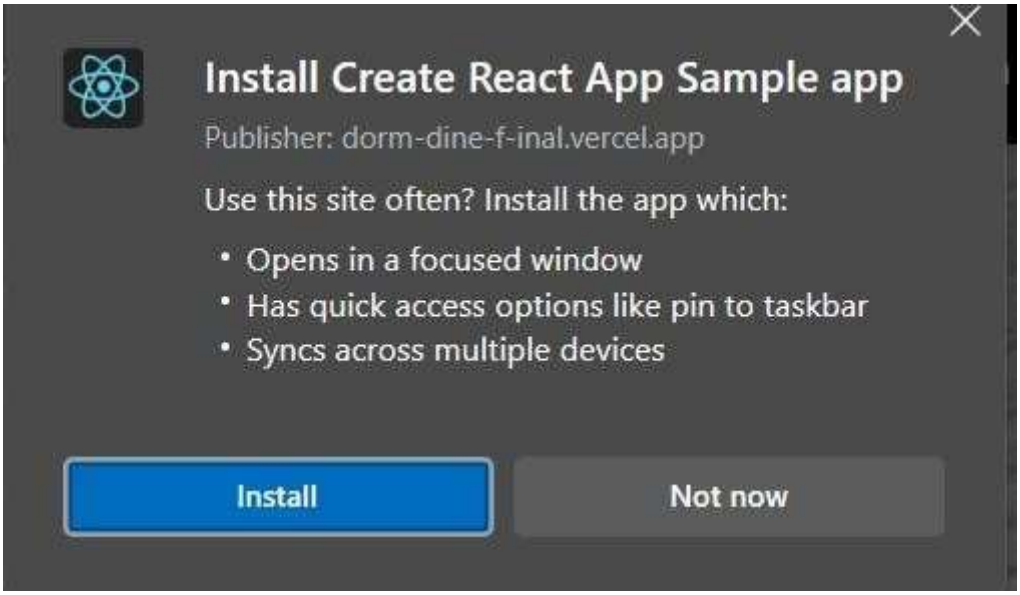
```
}

export default App;
```

Open folder in VS code and click go live at bottom right corner

Open your hosted site on Microsoft Edge



Click on 3 dots
click on Apps
select install app option
Click on Install

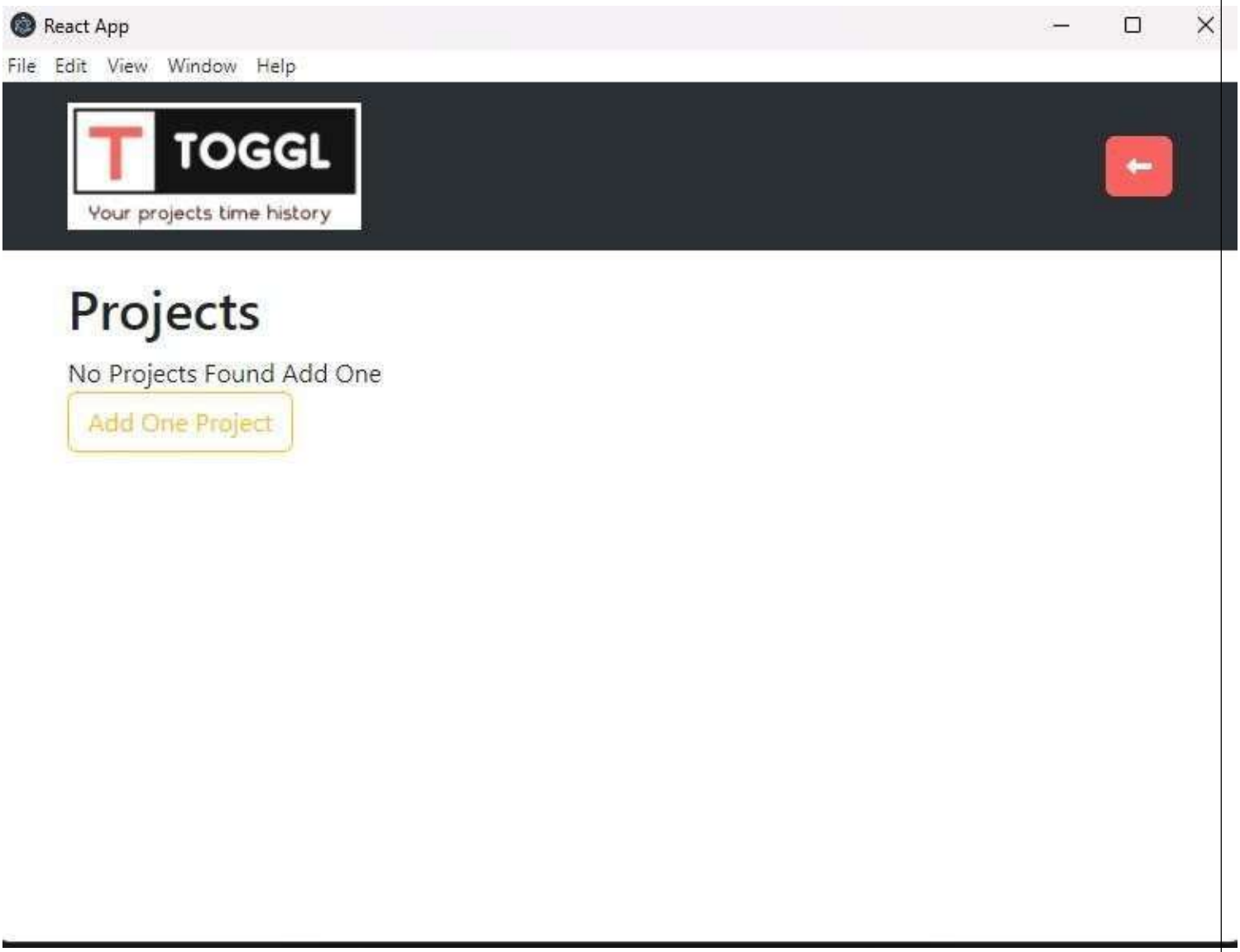


App icon will appear at the bottom

Output:

Desktop App Created Successfully.

Open Desktop App:



Conclusion:

In this experiment, we have successfully created a basic progressive web app of our web page and installed it in our desktop successfully.

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.	59
Name	**Aditya P Tahiliani**
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	14

\

Aditya P. Tahiliani
59, D15-A

Experiment No:08

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.

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('/s
 ervice-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.

\

Code

Index.html

manifest.json M | app.js U | service-worker.js U | index.html M

index.html > html > body.auth-body.dark > script

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="fuzzy" />
    <meta name="keywords" content="fuzzy" />
    <meta name="author" content="fuzzy" />
    <link rel="manifest" href="manifest.json" />
    <link rel="icon" href="assets/images/logo/favicon.png" type="image/x-icon" />
    <title>fuzzy</title>
    <link rel="apple-touch-icon" href="assets/images/logo/favicon.png" />
    <meta name="theme-color" content="#122636" />
    <meta name="apple-mobile-web-app-capable" content="yes" />
    <meta name="apple-mobile-web-app-status-bar-style" content="black" />
    <meta name="apple-mobile-web-app-title" content="fuzzy" />
    <meta name="msapplication-TileImage" content="assets/images/logo/favicon.png" />
    <meta name="msapplication-TileColor" content="#FFFFFF" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
</head>

<body class="auth-body dark">
  <!-- onboarding section start -->
  <section class="section-b-space">
    <div class="swiper intro slider-1">
      <div class="swiper-wrapper">
        <div class="swiper-slide">
          <div class="theme-logo pb-3">
            <img class="img-fluid logo-img" src="assets/images/logo/logo.png" alt="logo" />
          </div>
          <div class="onboarding-design">
            <img class="img-fluid design-img" src="assets/images/onboarding/design1.png" alt="b

            <img class="img-fluid slider-img1" src="assets/images/onboarding/1.png" alt="slider

            <img class="img-fluid vector1" src="assets/images/onboarding/vector1.png" alt="v1"
            <img class="img-fluid vector2" src="assets/images/onboarding/vector2.png" alt="v2"
            <img class="img-fluid vector3" src="assets/images/onboarding/vector3.png" alt="v3"
          </div>
          <div class="product-details">
            <h1>Office Furniture</h1>
            <span></span>
```

\

```
<!-- pwa install app popup start -->
<div class="offcanvas offcanvas-bottom addtohome-popup theme-offcanvas" tabindex="-1" id="offc
  <button type="button" class="btn-close text-reset" data-bs-dismiss="offcanvas" aria-label="C
  <div class="offcanvas-body small">
    <div class="app-info">
      <img src="assets/images/logo/48.png" class="img-fluid" alt="" />
      <div class="content">
        <h4>fuzzy app</h4>
        <a href="#">www.fuzzy-app.com</a>
      </div>
    </div>
    <a href="#!" class="btn theme-btn install-app btn-inline home-screen-btn m-0" id="install
  </div>
</div>
<!-- pwa install app popup start -->

<!-- swiper js -->
<script src="assets/js/swiper-bundle.min.js"></script>
<script src="assets/js/custom-swiper.js"></script>

<!-- iconsax js -->
<script src="assets/js/iconsax.js"></script>
```

Style.css

```
@-webkit-keyframes fireworkLine {
  0% {
    right: 20%;
    -webkit-transform: scale(0, 0);
            transform: scale(0, 0);
  }
  25% {
    right: 20%;
    width: 6px;
    -webkit-transform: scale(1, 1);
            transform: scale(1, 1);
  }
  35% {
    right: 0;
    width: 35%;
  }
  70% {
    right: 0;
    width: 4px;
    -webkit-transform: scale(1, 1);
            transform: scale(1, 1);
  }
  100% {
    right: 0;
    -webkit-transform: scale(0, 0);
            transform: scale(0, 0);
```

```
@keyframes fireworkLine {
  0% {
    right: 20%;
    -webkit-transform: scale(0, 0);
            transform: scale(0, 0);
  }
  25% {
    right: 20%;
    width: 6px;
    -webkit-transform: scale(1, 1);
            transform: scale(1, 1);
  }
  35% {
    right: 0;
    width: 35%;
  }
  70% {
    right: 0;
    width: 4px;
    -webkit-transform: scale(1, 1);
            transform: scale(1, 1);
  }
  100% {
    right: 0;
    -webkit-transform: scale(0, 0);
```

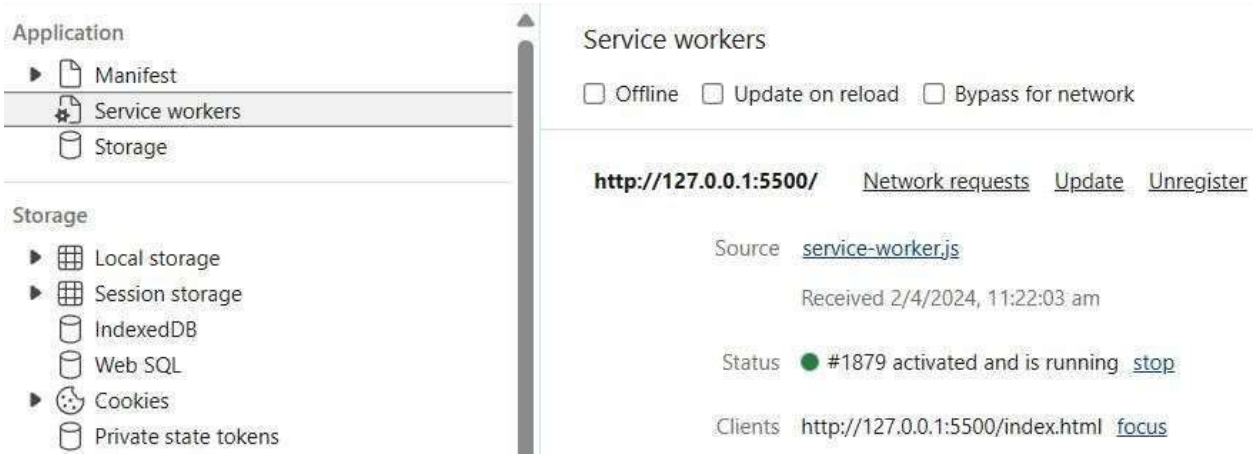
\

App.js

Service-Worker.js

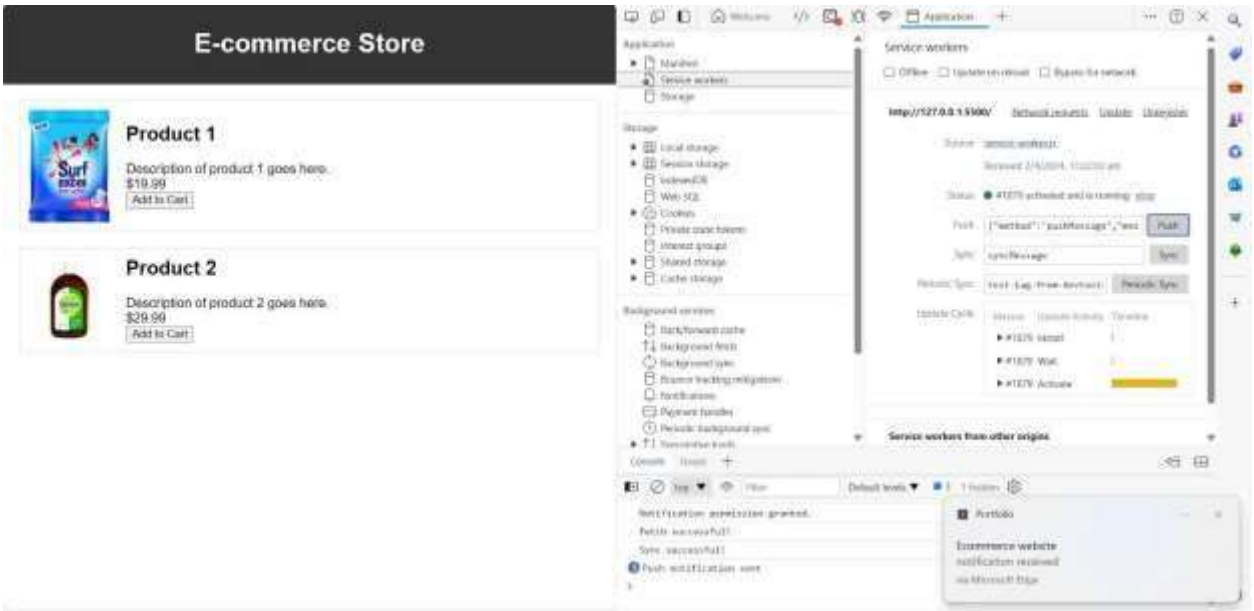
\

OUTPUT:


Application
Manifest
Service workers
Storage
Storage
Local storage
Session storage
IndexedDB
Web SQL
Cookies
Private state tokens
Service workers
Offline
Update on reload
Bypass for network
http://127.0.0.1:5500/
Network requests
Update
Unregister
Source
service-worker.js
Received 2/4/2024, 11:22:03 am
Status
#1879 activated and is running
stop
Clients
http://127.0.0.1:5500/index.html
focus

\

Cache Storage



Conclusion: In this experiment, we have registered a service worker, and completed the install and activation process for a new service worker for the E-commerce PW

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.	59
Name	**Aditya P Tahiliani**
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	14

Aditya P. Tahiliani
59, D15-A

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.

Code:

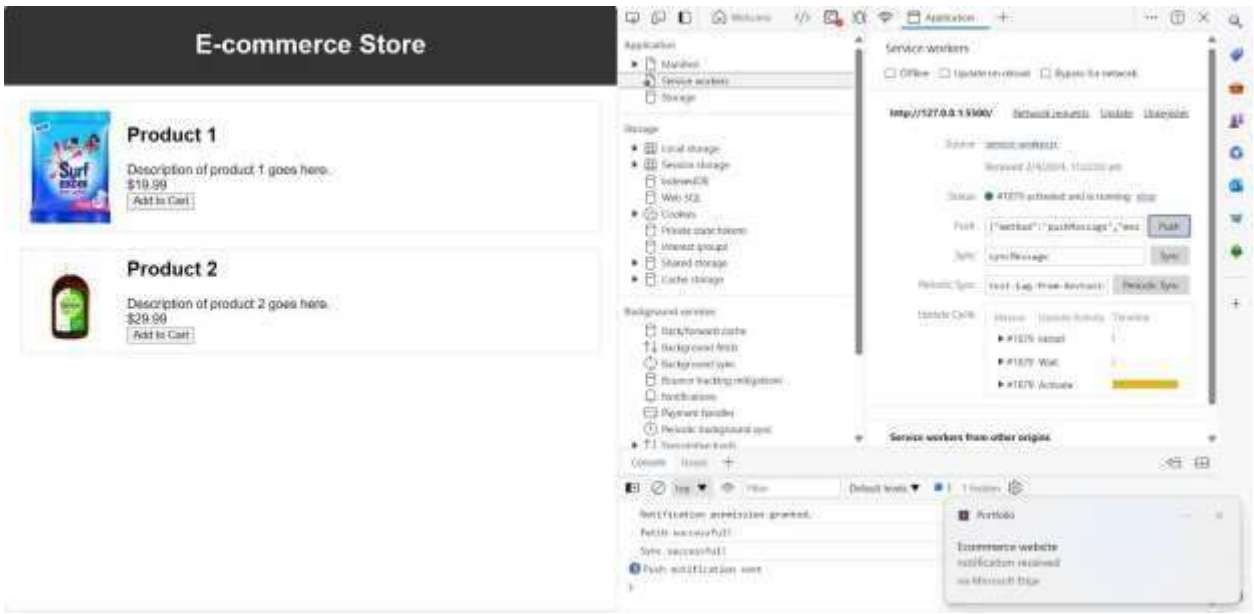
Service-Worker.js

```
self.addEventListener("install", function (event)
    { event.waitUntil(preLoad());
    });
    self.addEventListener("fetch", function (event)
    { event.respondWith( checkResponse(event.requ
    est).catch(function () { console.log("Fetch from
    cache successful!"); return
    returnFromCache(event.request);
    })
    );
    console.log("Fetch successful!");
    event.waitUntil(addToCache(event.requesst));
    });
    self.addEventListener("sync", (event) =>
    { if (event.tag === "syncMessage")
    { console.log("Sync successful!");
    }
    });
    self.addEventListener("push", function (event)
    { if (event && event.data) {
    try {
    var data = event.data.json();
    if (data && data.method === "pushMessage")
    { console.log("Push notification sent");
    self.registration.showNotification("Ecommerce website",
    { body: data.message,
    });
    }
    } catch (error) {
    console.error("Error parsing push data:", error);
    }
    }
    });
    var preLoad = function () {
    return caches.open("offline").then(function (cache) {
```

```
// caching index and important routes
return cache.addAll([
"/",
"/index.html",
"/landing.html",
"/profile.html",
"/shop.html",
"/cart.html",
"/css/main.css",
]);
});
};
var checkResponse = function (request)
{ return new Promise(function (fulfill, reject)
{ fetch(request)
.then(function (response) {
if (response.status !== 404)
{ fulfill(response);
} else {
reject(new Error("Response not found"));
}
})
.catch(function (error)
{ reject(error);
});
});
};
var returnFromCache = function (request) {
return caches.open("offline").then(function (cache)
{ return cache.match(request).then(function (matching)
{ if (!matching || matching.status == 404) {
return cache.match("offline.html");
} else {
return matching;
}
});
});
```

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

Output:



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

MAD & PWA Lab
Journal

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	59
Name	**Aditya P Tahiliani**
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	14

Aditya P. Tahiliani
59, D15-A

Experiment No. 10 MAD &PWA Lab

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.

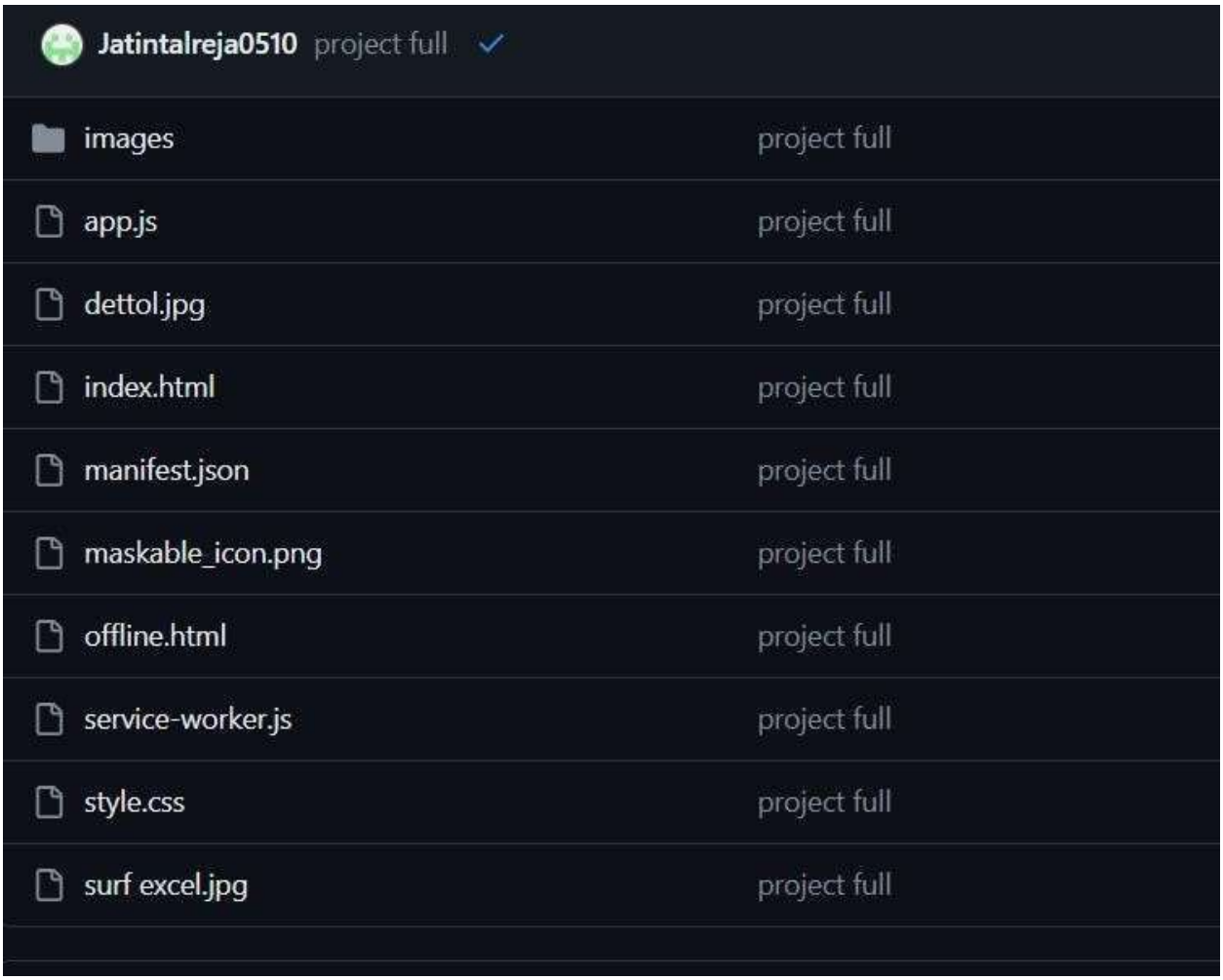
Cons

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: https://github.com/Jatintalreja0510/MPL-PWA-APP

Github Screenshot:




jatintalreja0510.github.io/MPL-PWA-APP/
E-commerce Store
Product 1
Description of product 1 goes here.
$19.99
Add to Cart
Product 2
Description of product 2 goes here.
$29.99
Add to Cart

MAD & PWA Lab
Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	59
Name	**Aditya P Tahiliani**
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	14

Aditya P. Tahiliani
59, D15-A

Experiment 11

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

Theory :

Reference : https://www.semrush.com/blog/google-lighthouse/

Google Lighthouse :

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

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

Key Features and Audit Metrics

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

1. **Performance:** This score is an aggregation of how the page fared in aspects such as (but not limited to) loading speed, time taken for loading for basic frame(s), displaying meaningful content to the user, etc. To a layman, this score is indicative of how decently the site performs, with a score of 100 meaning that you figure in the

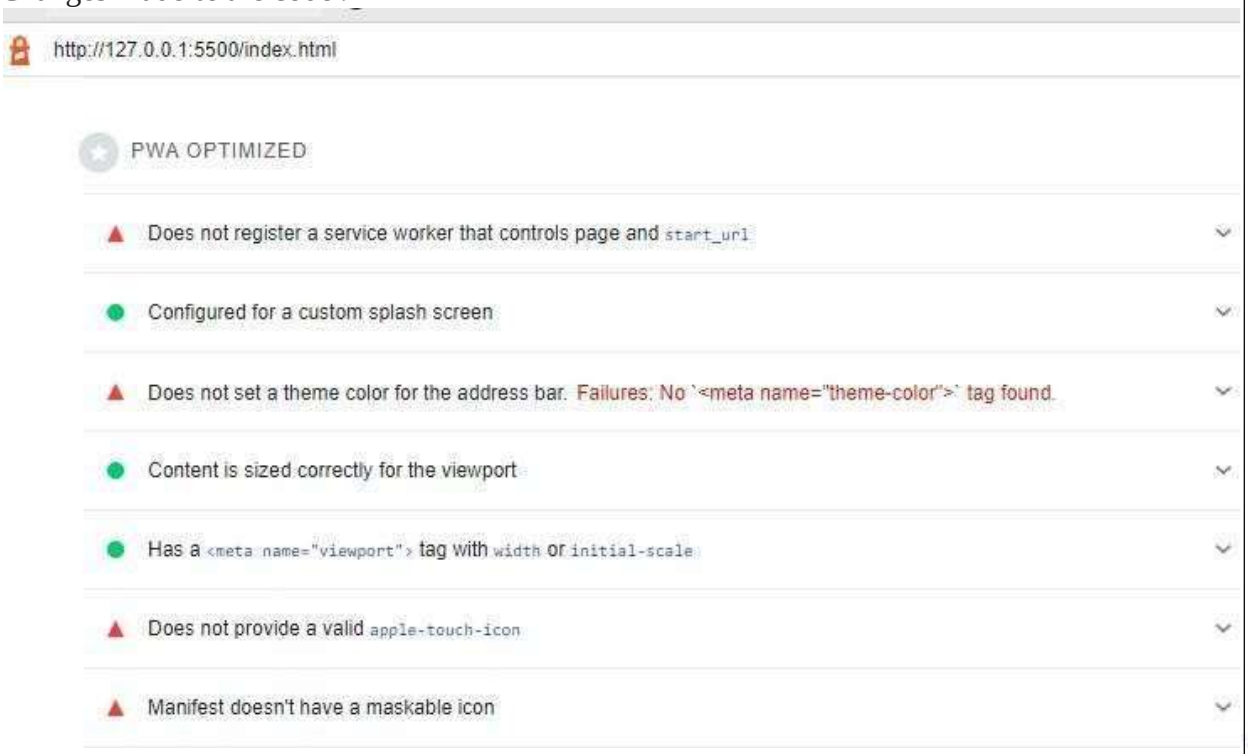
98th percentile, 50 meaning that you figure in the 75th percentile and so on.

2. **PWA Score (Mobile):** Thanks to the rise of Service Workers, app manifests, etc., a lot of modern web applications are moving towards the PWA paradigm, where the objective is to make the application behave as close as possible to native mobile applications. Scoring points are based on the Baseline PWA checklist laid down by Google which includes Service Worker implementation(s), viewport handling, offline functionality, performance in script-disabled environments, etc.
3. **Accessibility:** As you might have guessed, this metric is a measure of how accessible your website is, across a plethora of accessibility features that can be implemented in your page (such as the 'aria-' attributes like aria-required, audio captions, button names, etc.). Unlike the other metrics though, Accessibility metrics score on a pass/fail basis i.e. if all possible elements of the page are not screen-reader friendly (HTML5 introduced features that would make pages easy to interpret for screen readers used by visually challenged people like tag names, tags such as <section>, <article>, etc.), you get a 0 on that score. The aggregate of these scores is your Accessibility metric score.
4. **Best Practices:** As any developer would know, there are a number of practices that have been deemed 'best' based on empirical data. This metric is an aggregation of many such points, including but not limited to:Use of HTTPS

   Avoiding the use of deprecated code elements like tags, directives, libraries, etc. Password input with paste-into disabled

   Geo-Location and cookie usage alerts on load, etc.

Changes made to the code :



For theme color add a meta tag in index.html-

<meta name="theme-color" content="#4285f4">

For a maskable icon add "purpose": "any maskable" to the icons

in manifest.json file For apple touch icon add the following

meta tag in index.html-

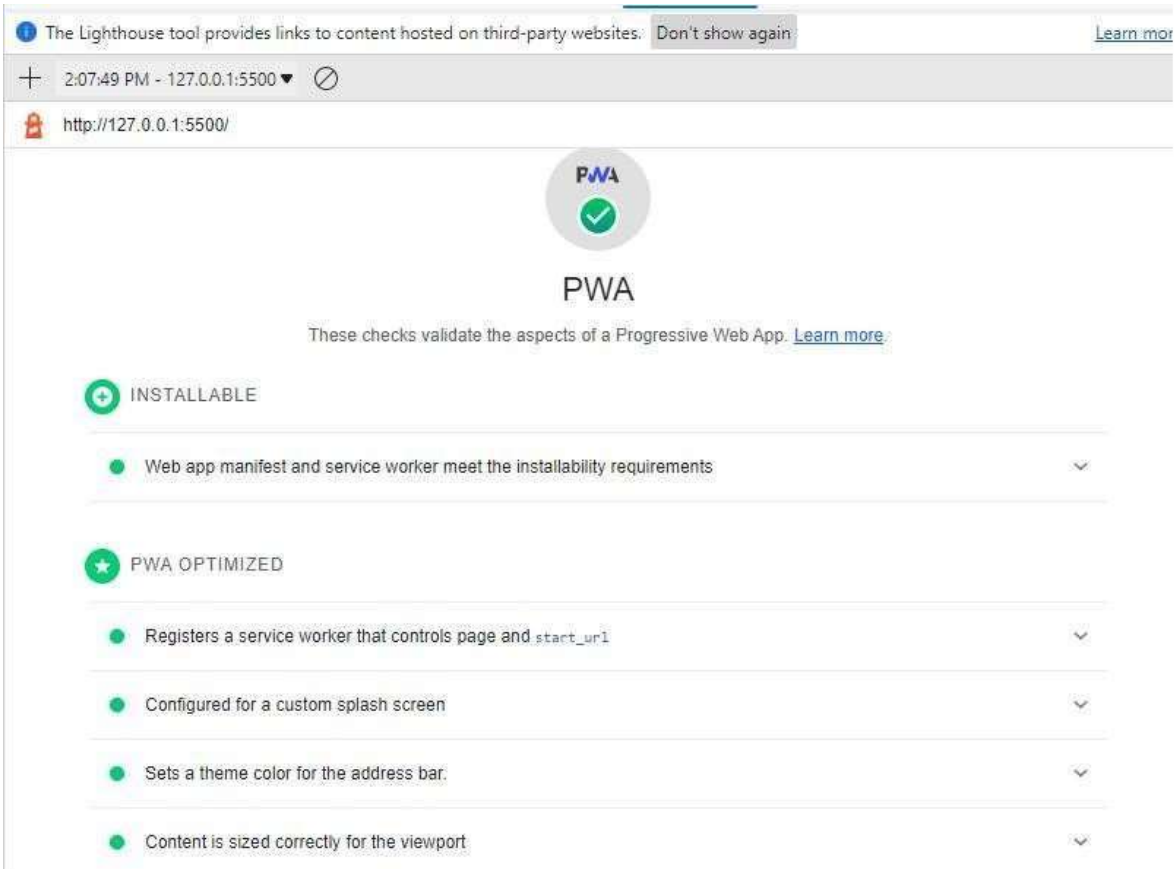
<link rel="apple-touch-icon" href="">

Changes in manifest.json

```
{
  "name":
  "flower
  shop
  website",
```

```
"short_na
me":
"flowers",
"start_url":
"index.htm
l", "scope":
"./",
"theme_
color":
"#ffd31d
",
"backgr
ound_co
lor":
"#333",
"display
":
"standal
one",
"icons":
[
  {
    "src": "icon-1.png",
    "size
    s":
    "192
    x192
    ",
    "typ
    e":
    "ima
    ge/p
    ng"
      “purpose”:"any maskable"

  },
  {
    "src": "icon-2.png",
    "size
    s":
```

```
        "512
        x512
        ",
        "typ
        e":
        "ima
        ge/p
        ng"
          “purpose”:"any maskable"

      }
    ]
}
```

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

MAD & PWA Lab
Journal

Experiment No.	Assignment-1
Assignment 1 Questions	1. Flutter Overview:Explain the key features and advantages of using Flutter for mobile app development. Discuss how the Flutter framework differs from traditional approaches and why it has gained popularity in the developer community. 2. Widget Tree and Composition: Describe the concept of the widget tree in Flutter. Explain how widget composition is used to build complex user interfaces. Provide examples of commonly used widgets and their roles in creating a widget tree. 3. State Management in Flutter: Discuss the importance of state management in Flutter applications. Compare and contrast the different state management approaches available in Flutter, such as setState, Provider, and Riverpod. Provide scenarios where each approach is suitable. 4. Firebase Integration in Flutter: Explain the process of integrating Firebase with a Flutter application. Discuss the benefits of using Firebase as a backend solution. Highlight the Firebase services commonly used in Flutter development and provide a brief overview of how data synchronization is achieved.
Roll No.	59
Name	**Aditya P Tahiliani**
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	4

Aditya P. Tahiliani
59, D15-A

05.02.24

Assignment 1

1. Flutter Overview.

(a) Explain the key features and advantages of using flutter for mobile app development.

→ Following are the key features & advantages of using flutter for mobile app development:-

1) Single Codebase : Develop for IOS and Android with one codebase
2) Hot Reload : Instantly see changes without restarting, speeding up development.
3) Rich widgets library and Extensive UI.
4) Open Source, Community Support & Ease of Adaptability.
5) Use of Dart Language with Easy Integration & High Performance.

(b). (b) Discuss how the flutter framework differs from traditional approaches and why it has gained popularity in the developer community.

→ Flutter is different from traditional approaches in the following ways:-

1) Single Codebase :- Flutter allows a Single code base for both IOS & Android whereas trad. Approach often involves sep. codebase for each platform.

2. Performance : Flutter's compiled code offers high perf., contrasting with potential performance variations in trad. cross platform solutions

3. Rich widgets :- Flutter provides a rich set of customizable widgets for UI, while trad'l tool approaches may involve more manual UI development.

4) Community & packages :- Flutter has a vibrant community & a rich ecosystem of packages, differing from potential fragmentation & limited packages in some trad. approaches.

The combination of features makes flutter an attractive choice for developers, offering a balance between performance, efficiency & a visually appealing user interface. Additionally Flutter's active and supportive community contributes to its continuous growth in popularity.

The Flutter's combination of features inclu
1) Single Code base
2) Hot Reload
3) Rich Widget Library
4) High Performance
5) Expressive UI
6) Open Source
7) Strong Community Support
8) Customization
9) Adaptability
10) Easy Integrat
11) Dart Languag
12) Responsive Framework

(2) Widget Tree & Composition

a) Describe the concept of the widget tree in Flutter.

→ • In Flutter, the widget tree represents the hierarchy of user elements organized as a tree structure. Each Element (called a widget) corresponds to a visual component or a layout piece in the user interface.

• The widget tree is rooted at the top level widget, typically the 'MaterialApp' or 'CupertinoApp' and extends down to the smallest, most granular widgets.

• Flutter follows a declarative approach, meaning that UI is described in code & rebuilt when needed, rather than imperatively updating the UI.

(b) Explain how widget composition is used to build complex user interfaces in flutter.

→ • Widget composition in flutter involves combining smaller, reusable widgets to create complex user interfaces.

• This modular approach enables developers to abstract and encapsulate functionality, encouraging reusability & maintainability.

• By organizing widgets hierarchically, developers can easily construct intricate UIs, fostering efficient develop

(c) Provide Examples of commonly used widgets and their roles in creating a widget tree

→ 1) Container :- A basic visual element that can contain other widgets and define its own styling.

2) Column & Row :- Organize child widgets vertically or horizontally in a linear arrangement.

3) Stack - Overlap widgets, allowing them to be positioned on top of each other.

4) ListView - Displays a scrolling, linear list of widgets, useful for displaying a dynamic set of Items.

5) GridView :- Arranges child widgets in a 2-D, scrollable grid.

6) AppBar :- Represents the top app bar that typically contains the app's title & actions.

7) Scaffold :- Provides a basic app structure, including the app's layout, navigation & theming.

3 State Management in Flutter :-

(a) (a) Discuss the importance of state management in flutter Applications.

→ • In the context of Mobile app development, "State" refers to any data that can change over time.
• This data can include user input, the content displayed on screen, network reponses or even device's orientation.
• Following are the reasons for why State management is important :-

a) Maintains Data Consistency.
b) Enhances & Enables Data Persistence.
c) Promote Clean Code.
d) Improves Scalability & Optimizes performance
e) Enhances testability.

(b) Compare & contrast the different State management approaches available in Flutter, such as SetState, Provider, and Riverpod. Provide Scenarios where each approach is suitable.

→ • The best Approach depends on your specific project requirements & preferences.

• Scenarios :-

1) <u>setState</u> : • Simple widget interactions (eg. button clicks)
• Managing state specific to a single widget.

b) Provider : • Sharing state between multiple widgets within a subtree
• Managing simple to moderate app-wide state.

c) Riverpod :- • Global state management across the entire app.
• Apps requiring find-grained control over data flow & updates.

Feature	setState	Provider	Riverpod.
Approach	Built in Flutter widget.	Package, dependency injection	Package Dependency Injection
Difficulty	Beginner Friendly	Moderate	Moderate.
Data Flow	Explicit, Manual updates	Declarative, Reactive	Declarative, Reactive.
Scalability	Limited for Complex Apps	More Scalable	More Scalable
Testing	Unit Testing possible	Easier Unit & Integration testing	Easier Unit & Integration Testing

4. Firebase Integration in Flutter :-

(a) Explain the process of Integrating Firebase with a Flutter Application.

→ Integrating Firebase with a Flutter application involves several steps:
1) Create a Firebase Project
2) Add Firebase to your App
3) Install Firebase SDK Dependencies
4) Initialize Firebase
5) Configure Firebase Services.
6) Implement Firebase Functionality.
7) Test & Deploy your flutter App.

(b) Discuss the benefits of using Firebase as a backend solution.

→ Following are benefits of using Firebase :-

1) Realtime Database with Synchronization capabilities
2) Authentication Supporting various methods.
3) Scalable NoSQL Cloud Firestore for flexible data Storage.
4) Cloud Functions for Serverless backend logic
5) Secure Cloud Storage for file storage.
6) Analytics for insights into user behaviour.
7) Performance Monitoring to monitor app performance.
8) Easy deployment with Firebase Hosting.
9) Built on Google Infrastructure.

(C) Highlight the Firebase services commonly used in Flutter development and provide a brief overview of how data synchronization is achieved.

→ Firebase services commonly used in Flutter Development include:-

1) Authentication.
2) Cloud Firestore
3) Realtime Database
4) Cloud Storage & Functions.

- Data Synchronization in Firebase is achieved through real-time listeners.
- When data changes on server, Firebase sends the updated data to all connected clients in real time using websockets or long-lived HTTP connections.
- Clients are notified of data changes, and they can update their local copies of the data accordingly, ensuring that all clients have the most up-to-date information.
- This real-time synchronization enables features like live updates, collaborative editing & real-time analytics in Flutter apps.

* * *

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.	59
Name	**Aditya P Tahiliani**
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO4:Understand various PWA frameworks and their requirements LO5: Design and Develop a responsive User Interface by applying PWA Design techniques LO6:Develop and Analyze PWA Features and deploy it over app hosting solutions
Grade:	4

Aditya P. Tahilliani
D15A/59

8/3/24

Assignment - 2

Q1 Define Progressive Web App (PWA) and explain its significance in modern web Development. Discuss the key characteristics that differentiate PWAs from traditional mobile Apps.

→ • A Progressive Web App (PWA) is a type of web Application that utilizes modern web capabilities to deliver an app like experience to users.

• PWAs are designed to work across all devices and platforms and can be accessed through a web browser without requiring users to download and install them from an app store.

• The significance of PWAs in modern web development lies in their ability to combine the best features of web and mobile applications. Some key characteristics that differentiate PWAs from traditional mobile apps include:

a) Cross-platform compatibility - PWAs are built using standard web technologies making them compatible with multiple platforms and devices including desktops, tablets and smartphones.

b) No Installation Required :- Unlike traditional mobile apps, PWAs can be accessed directly through a web Browser URL without need for installation from an app Store.

c) Offline Functionality : PWAs can work offline or in areas with poor connectivity by leveraging service workers, which cache content and enable basic functionality even when the device is offline.

d) Responsive Design :- PWAs are built with responsive design principles, ensuring that they adapt seamlessly to different screen sizes and orientations.

e) App-like Experience - PWAs offer an app-like experience to users, including features such as push notifications, home screen installation, and full-screen mode.

Overall, PWAs represent a significant advancement in web development, offering a compelling alternative to traditional mobile apps by combining the reach & accessibility of web with functionality & user experience of native applications.

Q2 Define Responsive web design and explain its importance in the context of Progressive Web Apps. Compare & Contrast responsive, fluid, and adaptive web design approaches.

→ • Responsive web design is an approach to designing and developing websites that ensures optimal viewing and interaction experiences across a wide range of devices & screen sizes.

Date
Page

- In the context of Progressive Web Apps (PWAs), responsive web design is crucial for several reasons:-

a) Device Compatibility
b) User Experience
c) Accessibility
d) Search Engine Optimization (SEO).
e) Maintainance and Development Efficiency.

Comparison of Responsive, fluid & adaptive web design approaches in table format:

Aspect	Responsive web Design	Fluid web Design	Adaptive Web Design.
Definition	Uses CSS media queries to adjust layout & content based on Device characteristics	Uses Relative Units for Flexible layout Scaling	Creates diff. layouts for specific devices.
Layout Flexibility	Dynamically adapts layout & content	Scales layout elements fluidly	Presets layouts for diff devices
Design Approach	Adjusts existing layout for various devices	Focuses on flexible scale of Elements	Creats tailored layouts for each device.
Implem. Effort.	Involves setting break points & adjusting CSS	Requires using relative units for layout Elements	Requires creating & maintaining multiple versions.

Date
Page

5

	Implifies	Requires
	Single code	
SEO.	Supports SEO	

ons.
pose
ages
ple
sions

ding
ssive

be the lifecycle of Serv

\,«sb R_øJÙ?+uñ1ż •,oi„«s 's=,+ml Õl « r··Y,

2) Installation &
3) Activation.

egistration:
• Initiated by the web application throug the 'navigator. serviceWorker. register()' me
• Browser download the service worker scri Specified in the registration call.

- This phase establishes the service worker's scope & prepare it for installation.

2. Installation:

- Occurs once the service worker script is downloaded and parsed.
- The browser installs the service worker, triggering the 'install' event within the service worker script.
- During installation, the service worker can cache static assets and set up its initial state.

3. Activation:

- After installation, the service worker enters the activation phase.
- Activation triggered by the 'activate' event within the service worker script.
- During activation, the service worker can perform tasks like cleaning up old caches, managing versioning, and setting up event listeners for fetch and message events.
- Once activated, the service worker becomes ready to intercept network requests, handle push notifications, and execute other background tasks as defined in its script.

Q4 Explain the use of IndexedDB in the service worker for data storage.

→ • IndexedDB is a client-side storage mechanism

that allows web applications, including Service Workers, to store and retrieve large amounts of structured data asynchronously.

- In the context of Service workers, IndexedDB can be used to :-

1) Cache Data :- Workers can utilize IndexedDB to cache resources such as HTML, CSS, JavaScript files, images, and other data needed for offline functionality.

2) Offline data Access :- IndexedDB allows Service workers to store data locally on the user's device, enabling offline access to previously fetched content.

3) Performance Optimization :- By storing frequently accessed data in IndexedDB, Service workers can improve the performance of web applications by reducing the need to fetch data from network repeatedly.

4) Background Data Synchronization :- Service workers can use IndexedDB to store data temporarily while the device is offline & synchronize it with the server once the network connection is restored.

* * *