Department of Information Technology

CERTIFICATE

This is to certify that **Sujal Taktani** of **D15A** semester **VI,** have successfully completed necessary experiments in the **MAD & PWA Lab** under my supervision in **VES Institute of Technology** during the academic year **2023-2024.**

Lab Assistant

Subject Teacher

Mrs. Kajal Joseph

Principal

Head of Department

Dr. Mrs. Shalu Chopra

Name of the Course : MAD & PWA Lab

Course Code : ITL604

Year/Sem/Class : D15A/D15B

A.Y.: 23-24

Faculty Incharge : Mrs. Kajal Joseph.

Lab Teachers : Mrs. Kajal Jewani.

Email : kajal.jewani@ves.ac.in

Programme Outcomes: The graduate will be able to:

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

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

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

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

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

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

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

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

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

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

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

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

Program specific Outcomes

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

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

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	9
2.	To design Flutter UI by including common widgets.	LO2	23/1	30/1	9
3.	To include icons, images, fonts in Flutter app	LO2	30/1	6/2	9
4.	To create an interactive Form using form widget	LO2	6/2	13/2	9
5.	To apply navigation, routing and gestures in Flutter App	LO2	13/2	20/2	9
6.	To Connect Flutter UI with fireBase database	LO3	20/2	5/3	9
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	11
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	11
9.	To implement Service worker events like fetch, sync and push for E-commerce PWA	LO5	19/3	26/3	11
10.	To study and implement deployment of Ecommerce PWA to GitHub Pages.	LO5	26/3	5/3	11
11.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.	LO6	5/3	2/4	10
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

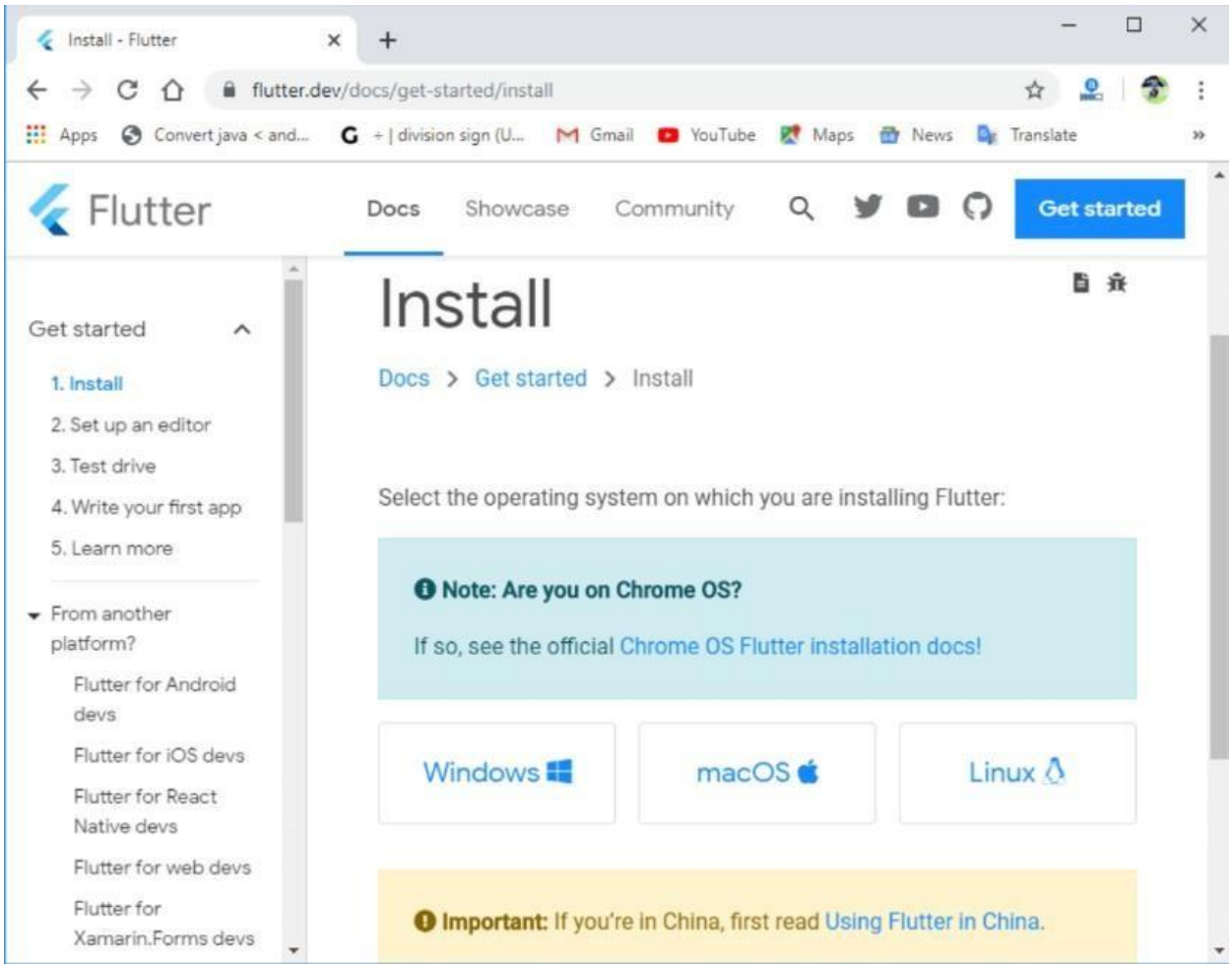
Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	60
Name	Sujal Taktani
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	9

MAD LAB EXP-1

Name- Sujal Taktani Div-D15A Roll-60

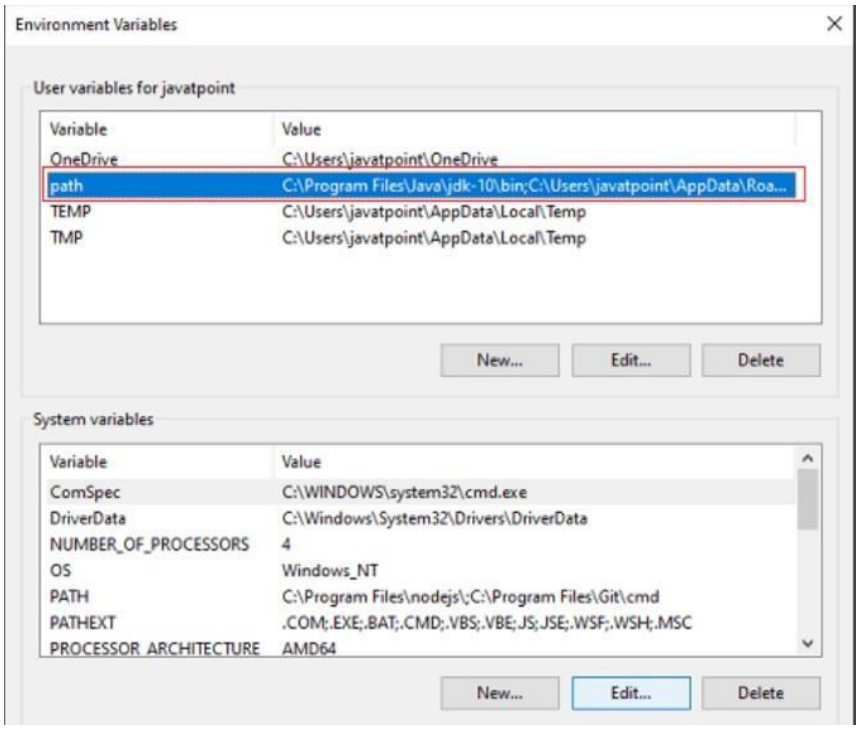
EXP-1 Installation of Flutter

Step 1- Install the flutter SDK, download the latest Flutter SDK,

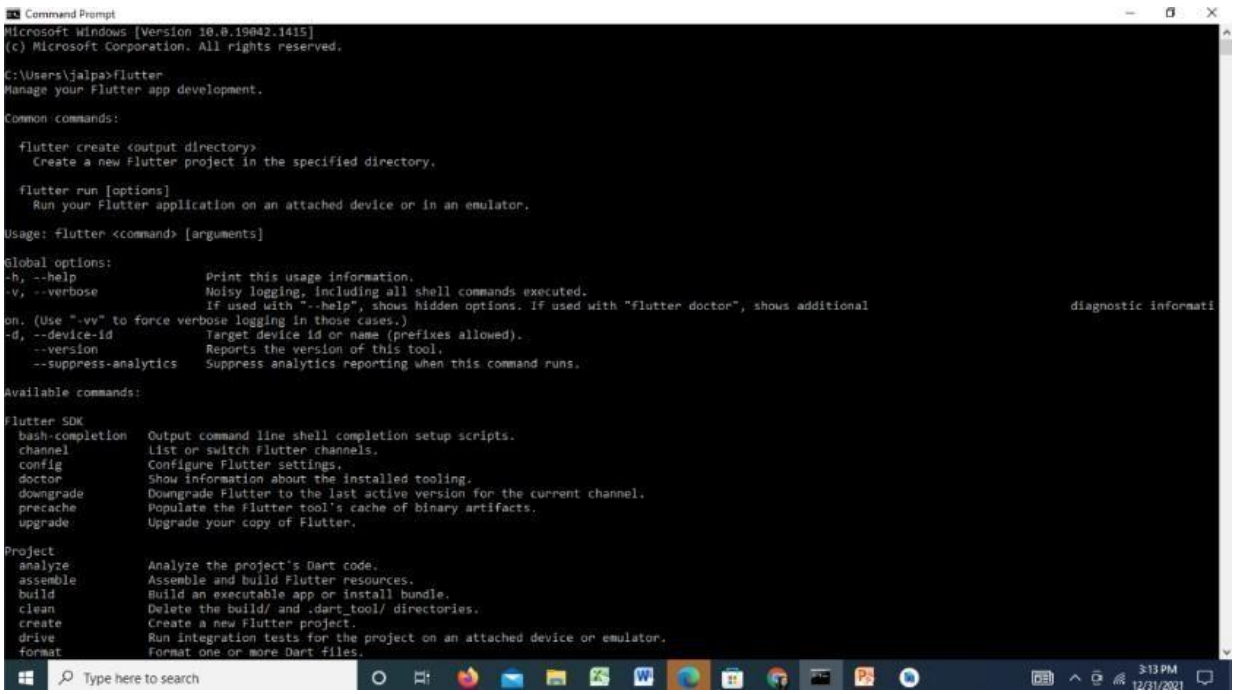


Step 2-When your download is complete, extract the zip file and place it inthe desired installation folder or location, for example, C: /Flutter.

Step 3- Now edit the environment variables.

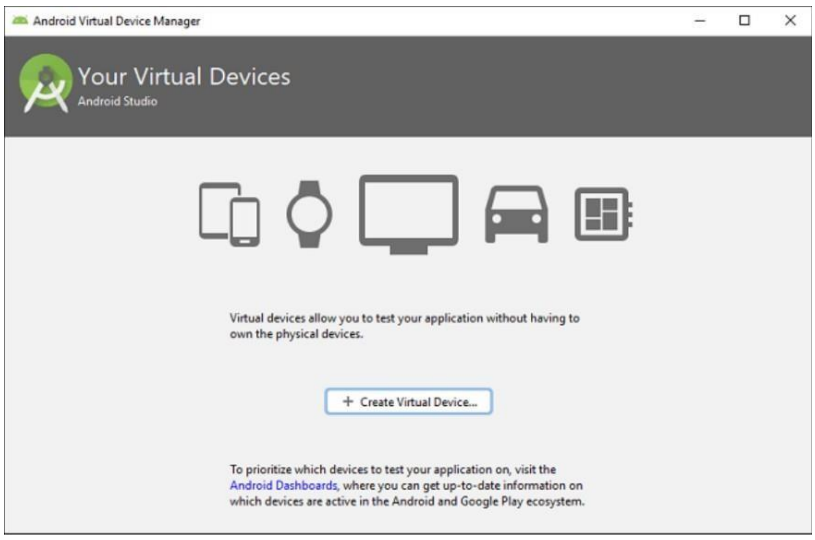


Step 4-Now, run the $ flutter and Flutter doctor command in command prompt.

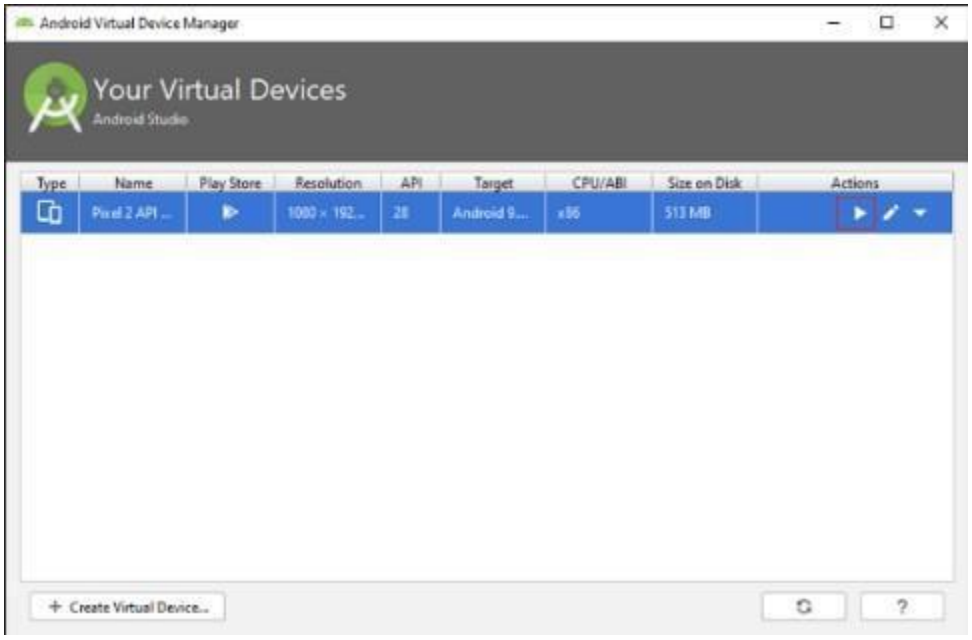


Step 5- Install the Android SDK. If the flutter doctor command does not find the Android SDK tool in your system, then you need first to install the Android Studio IDE.

Step 6- Next, you need to set up an Android emulator. It is responsible for running and testing the Flutter application.



Step 7- Select the system image for the latest Android version and click on Next. Now, verify the all AVD configuration. If it is correct, click on Finish. The following screen appears.



Step 8 - 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.

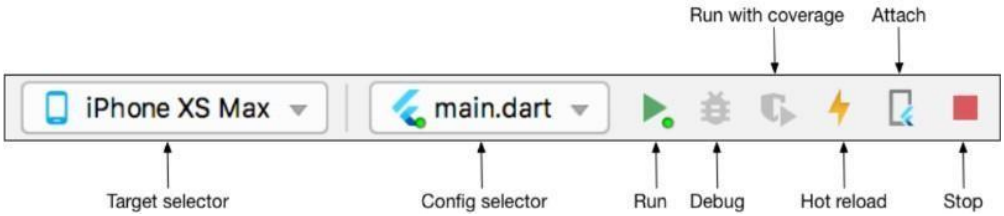
EXP 2 - RUNNING Hello World on Flutter

Step 1-Create the app.

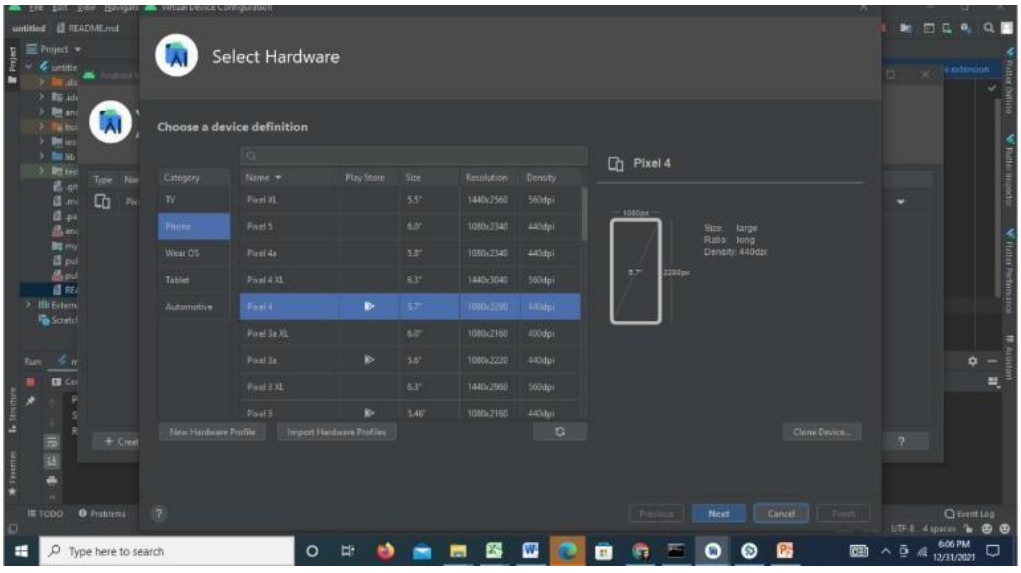
1. Open the IDE and select Create New Flutter Project.
2. Select Flutter Application as the project type. Then click Next.
3. Verify the Flutter SDK path specifies the SDK‟s location (select Install SDK... if the
text field is blank).
4. Enter a project name (for example, myapp). Then click Next.
5. Click Finish.
6. Wait for Android Studio to install the SDK and create the project.

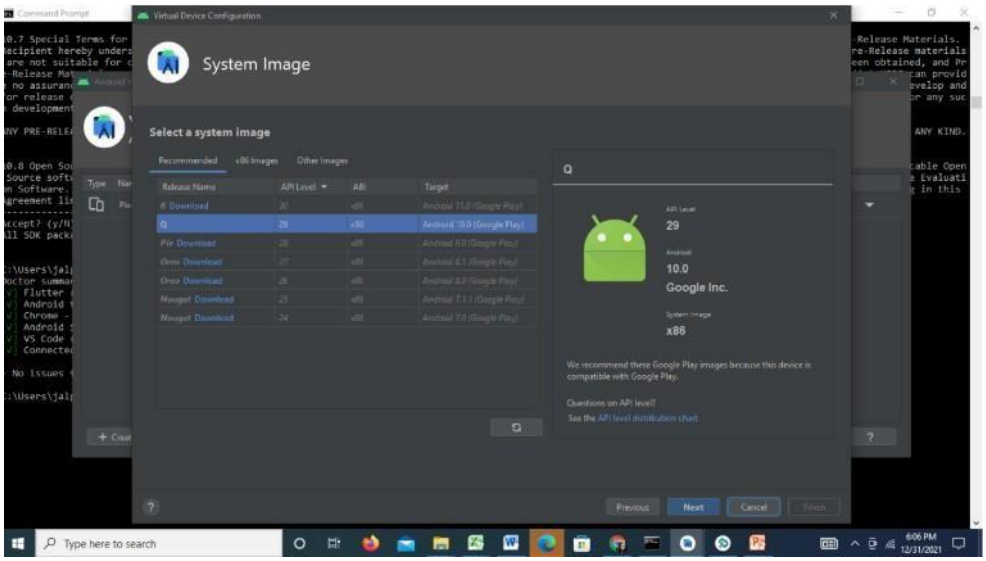
Step 2: Run the app.

Locate the main Android Studio toolbar:



Step 3-



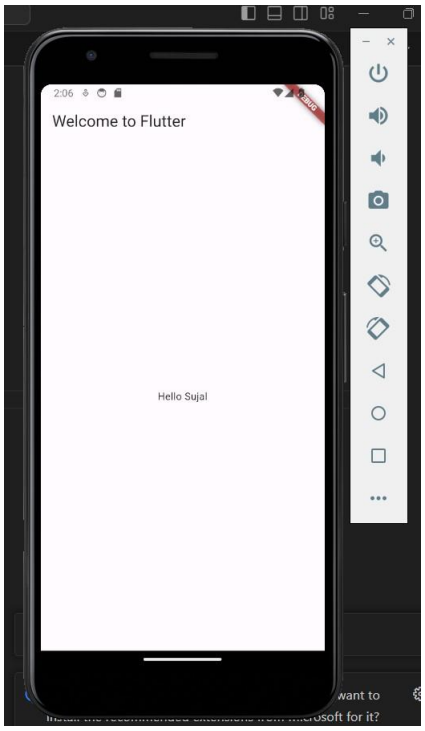


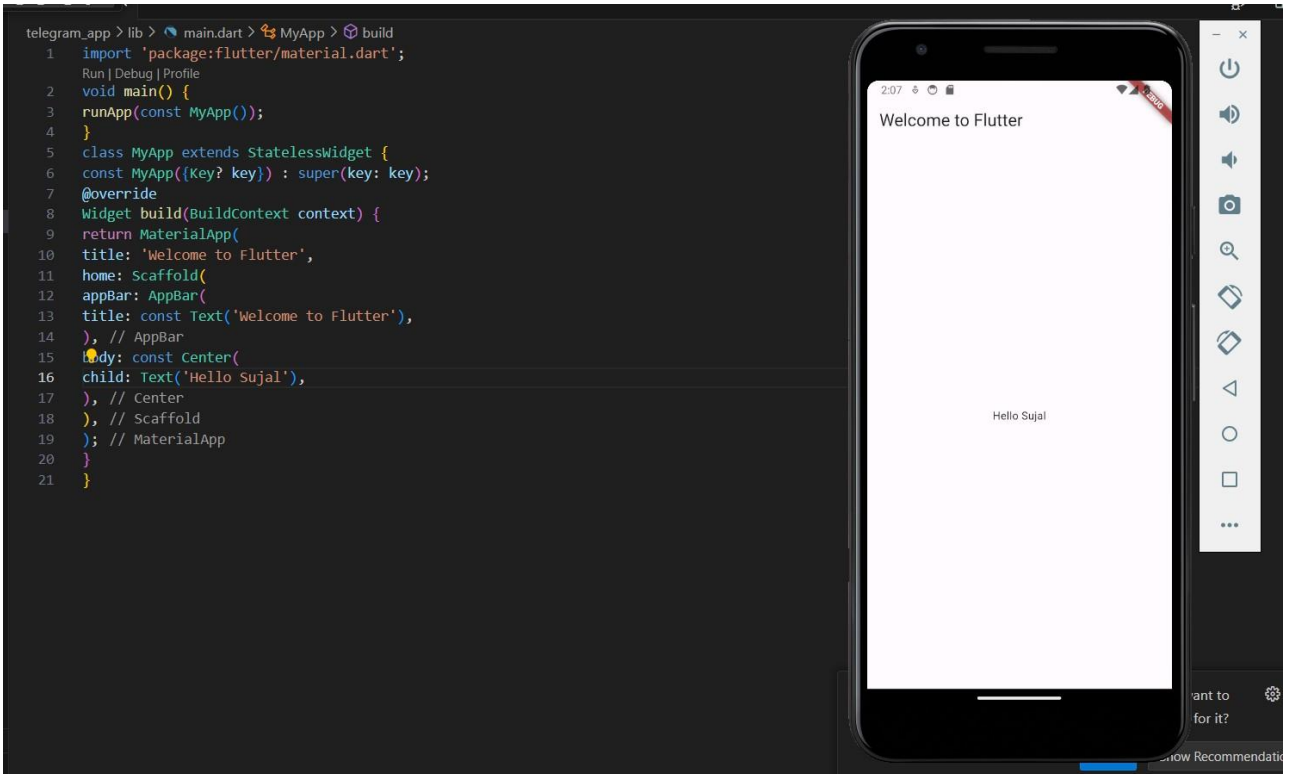
Step 3 : Creating Hello world app

Code-

```
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
          Sujal'),
        ),
      ),
    );
  }
}
```

OUTPUT:-


2:06
Welcome to Flutter
Hello Sujal


telegram_app > lib > main.dart > MyApp > build
import 'package:flutter/material.dart';
Run | Debug | Profile
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'),
), // AppBar
body: const Center(
child: Text('Hello Sujal'),
), // Center
), // Scaffold
); // MaterialApp
}
}
2:07
Welcome to Flutter
Hello Sujal

MAD & PWA Lab

Journal

Experiment No.	02
Experiment Title.	To design Flutter UI by including common widgets.
Roll No.	60
Name	Sujal Taktani
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	9

Sujal Taktani Batch :- C

D15A_60

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 orforms. Like Row, you have control over the alignment, spacing, and size of thechildren 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 verticallyor horizontally and supports both static and dynamic lists.

Code :-

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

import
'package:telegram_clone_flutter/screens/dark_mode/drawer_dark.d
art';

import
'package:telegram_clone_flutter/screens/models/chat_model.dart';


import 'chatting.dart';



class HomeScreenDark extends StatefulWidget
  {@override
  _HomeScreenDarkState createState() => _HomeScreenDarkState();
}


class _HomeScreenDarkState extends State<HomeScreenDark>
  {@override
  Widget build(BuildContext context)
    {return Scaffold(
      backgroundColor:
      Color(0xff1d2733),appBar: AppBar(
```

```
    backgroundColor:
    Color(0xff212d3b),title:
    Text('Telegram'),
    actions:
      <Widget>[
      Padding(
        padding: const EdgeInsets.all(8.0),
        child: Icon(Icons.search),
      )
    ],
  ),
  drawer: Theme(
  data: Theme.of(context).copyWith(
canvasColor: Color(0xff1d2733),
    //This will change the drawer background to blue.
//other styles
),
  child: DrawerScreenDark(),),



  body: Padding(
    padding: const EdgeInsets.only(top: 10),
    child: ListView.separated(
        itemBuilder: (ctx, i)
          {return InkWell(
          onTap: (){
```

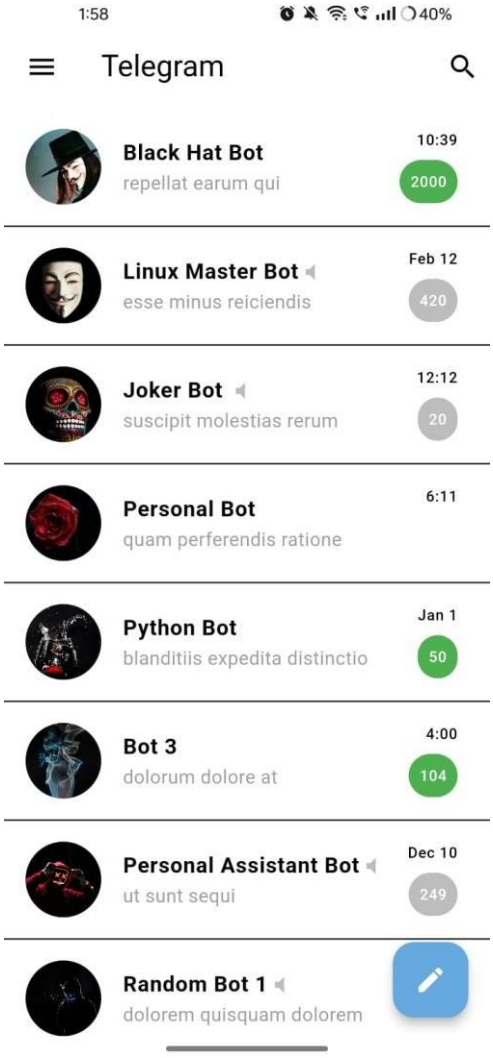
Navigator.push(

```
            context,
            MaterialPageRoute(builder: (context) =>
ChatScreenDark()),
            );
          },
          child: ListTile(
            leading:
            CircleAvatar(radius:
            28,
            backgroundImage: AssetImage(items[i].imgPath),
            ),
            title: items[i].status ?
            Text(items[i].name,style: TextStyle(fontWeight:
FontWeight.bold,color: Colors.white),):
            Row(children: [
              Text(items[i].name,style:
TextStyle(fontWeight:FontWeight.bold,color:
Colors.white),),
              Icon(Icons.volume_mute,size: 18,color:
Color(0xff7d8b98),)
            ],),
            subtitle:Text(items[i].message,style: TextStyle(color:
Color(0xff7d8b98)),),



            trailing:items[i].messNum!=null


            ?Column(
              crossAxisAlignment:
              CrossAxisAlignment.end,children: [
```

```
Text(items[i].time,style: TextStyle(color:

Colors.grey),),SizedBox(height: 7,),



Container(

 decoration: BoxDecoration(
   color:
items[i].status?Color(0xff64b4ef):Color(0xff3e5263),
   borderRadius: BorderRadius.circular(30)
 ),
 child:Padding(
  padding: const EdgeInsets.all(8.0),
  child: Text('${items[i].messNum}',style:
TextStyle(color:Colors.white,fontWeight: FontWeight.bold),),
 ),
)


 ],
):

Column(
 crossAxisAlignment:
 CrossAxisAlignment.end,children: [
```

```
                Text(items[i].time,style: TextStyle(color:

                Colors.grey),),SizedBox(height: 7,),

              ],)


          ),
        );
      },
      separatorBuilder: (ctx, i)
       {return Divider();
      },
      itemCount: items.length),
  ),
  floatingActionButton: FloatingActionButton(
    child: Icon(Icons.create,color:
    Colors.white,),backgroundColor:
    Color(0xFF65a9e0), onPressed: (){}),
 );
 }
}
```

Screenshot :-


1:58
40%
Telegram
Black Hat Bot
repellat earum qui
10:39
2000
Linux Master Bot
esse minus reiciendis
Feb 12
420
Joker Bot
suscipit molestias rerum
12:12
20
Personal Bot
quam perferendis ratione
6:11
Python Bot
blanditiis expedita distinctio
Jan 1
50
Bot 3
dolorum dolore at
4:00
104
Personal Assistant Bot
ut sunt sequi
Dec 10
249
Random Bot 1
dolorem quisquam dolorem

MAD & PWA Lab

Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	60
Name	Sujal Taktani
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	9

Sujal Taktani Batch :- C

D15A_60

Experiment no. :- 03

Aim :- To include icons, images, fonts in

Flutter app

Theory :-

1) Button: the Button widget is not a specific widget, but rather a category ofwidgets 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 edittext, 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 inthe 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 eventslike 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 likenavigation, opening menus, submitting forms, etc.

Code :-

```
import
'package:animated_login/animated
_login.dart';import
'package:async/async.dart';
import 'package:flutter/material.dart';

import 'auth.dart';



/// Example app widget.
class MyAppLogin extends StatelessWidget {
 /// Main app widget.
 const MyAppLogin({Key? key}) : super(key: key);

 @override
 Widget
  build(BuildContext
  context) {return
  MaterialApp(
   title: 'Animated
   Login', theme:
   ThemeData(
```

```
primaryColor:
Colors.blue,
 colorScheme: const ColorScheme.light(primary: Colors.blue),
 // useMaterial3: true,
 textTheme: Theme.of(context).textTheme.apply(
  // Note: The below line is required due to a current bug in Flutter:
  //
   https://github.com/flutter/flutter/issues/1
   29553 decorationColor: Colors.blue),
 inputDecorationTheme: const
  InputDecorationTheme(
  prefixIconColor: Colors.black54,
  suffixIconColor:
  Colors.black54,
  iconColor:
  Colors.black54,
  labelStyle:
  TextStyle(color:
  Colors.black54), hintStyle:
  TextStyle(color:
  Colors.black54),
 ),
),
debugShowCheckedModeBanner: false,
```

```
   initialRoute:
   '/login',routes: {
    '/login': (BuildContext context) => const LoginScreen(),
    '/forgotPass': (BuildContext context) => const
    ForgotPasswordScreen(),
   },
  );
 }


// static const Map<int, Color> color = {
//  50: Color.fromRGBO(4, 131, 184, .1),
//   100: Color.fromRGBO(4, 131, 184, .2),
//   200: Color.fromRGBO(4, 131, 184, .3),
//   300: Color.fromRGBO(4, 131, 184, .4),
//   400: Color.fromRGBO(4, 131, 184, .5),
//   500: Color.fromRGBO(4, 131, 184, .6),
//   600: Color.fromRGBO(4, 131, 184, .7),
//   700: Color.fromRGBO(4, 131, 184, .8),
//   800: Color.fromRGBO(4, 131, 184, .9),
//  900: Color.fromRGBO(4, 131, 184, 1),
// };
}


/// Example login screen
class LoginScreen extends StatefulWidget {
 /// Simulates the multilanguage, you will implement your own logic.
 /// According to the current language, you can display a text message
```

```
 /// with the help of [LoginTexts] class.
 const LoginScreen({Key? key}) : super(key: key);

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

class _LoginScreenState extends State<LoginScreen> {
 /// Example selected language, default is English.
```

```
LanguageOption language = _languageOptions[1];

/// Current auth mode, default is
[AuthMode.login]. AuthMode
currentMode = AuthMode.login;

CancelableOperation? _operation;

@override
Widget
 build(BuildContext
 context) { return
 AnimatedLogin(
  onLogin: (LoginData data) async =>
    _authOperation(LoginFunctions(context)
  .onLogin(data)), onSignup: (SignUpData
  data) async =>
    _authOperation(LoginFunctions(context)
  .onSignup(data)), onForgotPassword:
  _onForgotPassword,
  logo: Image.asset('assets/images/logo.gif'),
  // backgroundImage:
  'images/background_image.jpg',
  signUpMode: SignUpModes.both,
```

```
socialLogins:
_socialLogins(context),
loginDesktopTheme:
_desktopTheme,
loginMobileTheme:
_mobileTheme,
loginTexts:
_loginTexts,
emailValidator:
ValidatorModel(
  validatorCallback: (String? email) =>
'What an email! $email'),
changeLanguageCallback:
(LanguageOption? _language) {
 if (_language != null) {
  DialogBuilder(context).showR
  esultDialog(
    'Successfully changed the language
  to: ${_language.value}.'); if (mounted)
  setState(() => language = _language);
 }
},
changeLangDefaultOnPressed: () async
=> _operation?.cancel(),
```

languageOptions: _languageOptions,

selectedLanguage:

language,

initialMode:

currentMode,

```
  onAuthModeChange:
    (AuthMode newMode) async {
    currentMode = newMode;
    await _operation?.cancel();
  },
 );
}

Future<String?>
 _authOperation(Future<String?> func)
 async { await _operation?.cancel();
 _operation = CancelableOperation.fromFuture(func);
 final String? res = await
 _operation?.valueOrCancellation(); if
 (_operation?.isCompleted == true) {
  DialogBuilder(context).showResultDialog(res ?? 'Successful.');
 }
 return res;
}

Future<String?>
 _onForgotPassword(String email)
 async { await _operation?.cancel();
 return await LoginFunctions(context).onForgotPassword(email);
}
```

```
static List<LanguageOption> get
 _languageOptions => const <LanguageOption>[
 LanguageOption(
  value:
  'Turkish',
  code:
  'TR',
  iconPath: 'assets/images/tr.png',
 ),
 Language
  Option(
  value:
  'English',
  code:
  'EN',
  iconPath: 'assets/images/en.png',
 ),
];
```

```
/// You can adjust the colors, text styles, button styles, borders
/// according to your design preferences for DESKTOP view.
/// You can also set some additional display
options such as [showLabelTexts].
LoginViewTheme get _desktopTheme =>
_mobileTheme.copyWith(
 // To set the color of button text,
 use foreground color.
 actionButtonStyle: ButtonStyle(
  foregroundColor: MaterialStateProperty.all(Colors.white),
 ),
 dialogTheme: const
  AnimatedDialogTheme(
  languageDialogTheme:
  LanguageDialogTheme(
    optionMargin: EdgeInsets.symmetric(horizontal: 80)),
 ),
 loadingSocialButtonColor:
 Colors.blue, loadingButtonColor:
 Colors.white,
 privacyPolicyStyle: const
 TextStyle(color: Colors.black87),
 privacyPolicyLinkStyle: const
 TextStyle(
```

```
   color: Colors.blue, decoration: TextDecoration.underline),
);

/// You can adjust the colors, text styles, button styles, borders
/// according to your design preferences for MOBILE view.
/// You can also set some additional display
options such as [showLabelTexts].
LoginViewTheme get _mobileTheme =>
LoginViewTheme(
 // showLabelTexts: false,
 backgroundColor: Colors.blue, //
 const Color(0xFF6666FF),
 formFieldBackgroundColor:
 Colors.white, formWidthRatio: 60,
 actionButtonStyle: ButtonStyle(
  foregroundColor: MaterialStateProperty.all(Colors.blue),
 ),
 animatedComponentOrder: const
  <AnimatedComponent>[
  AnimatedComponent(
   component:
   LoginComponents.logo,
   animationType:
   AnimationType.right,
  ),
```

```
 AnimatedComponent(component:
 LoginComponents.title),
 AnimatedComponent(component:
 LoginComponents.description),
 AnimatedComponent(component:
 LoginComponents.formTitle),
 AnimatedComponent(component:
 LoginComponents.socialLogins),
 AnimatedComponent(component:
 LoginComponents.useEmail),
 AnimatedComponent(component:
 LoginComponents.form),
 AnimatedComponent(component:
 LoginComponents.notHaveAnAccount),
 AnimatedComponent(component:
 LoginComponents.forgotPassword),
 AnimatedComponent(component:
 LoginComponents.policyCheckbox),
 AnimatedComponent(component:
 LoginComponents.changeActionButton),
 AnimatedComponent(component:
 LoginComponents.actionButton),
],
privacyPolicyStyle: const
```

```
 TextStyle(color: Colors.white70),
 privacyPolicyLinkStyle: const
 TextStyle(
   color: Colors.white, decoration: TextDecoration.underline),
);

LoginTexts get
 _loginTexts =>
 LoginTexts( nameHint:
 _username,
 login:
 _login,
 signUp:
 _signup,
 // signupEmailHint: 'Signup Email',
 // loginEmailHint: 'Login Email',
 // signupPasswordHint: 'Signup Password',
 // loginPasswordHint: 'Login Password',
);

/// You can adjust the texts in the screen according to the current
language
/// With the help of [LoginTexts], you can create
a multilanguage scren. String get _username =>
language.code == 'TR' ? 'Kullanıcı Adı' :
```

'Username';

String get _login => language.code == 'TR' ? 'Giriş Yap' : 'Login';

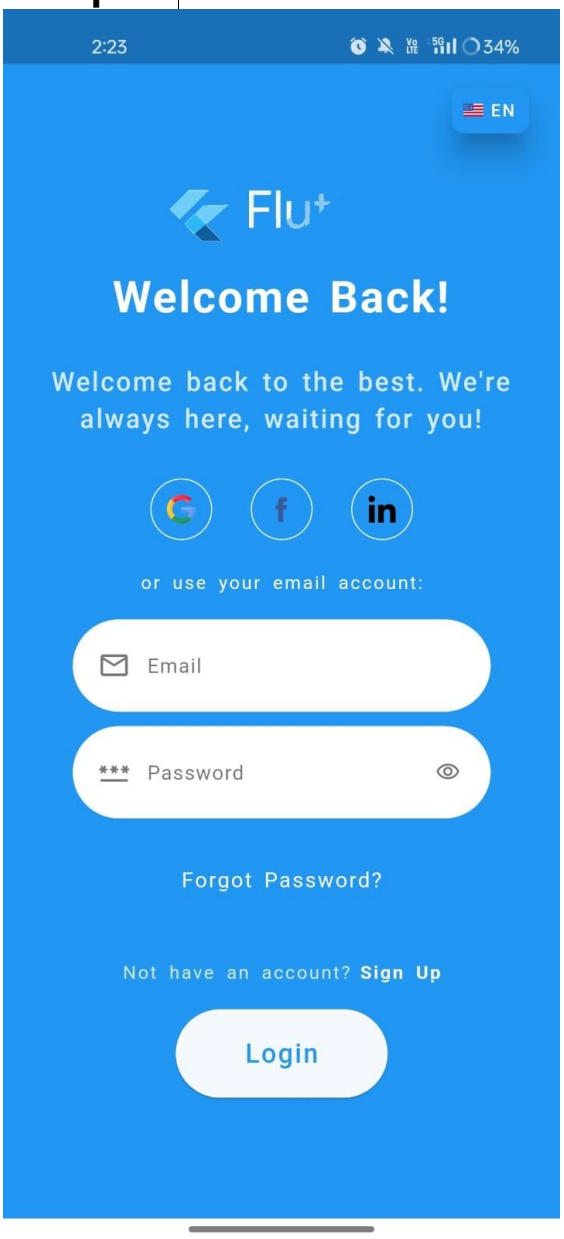
String get _signup => language.code == 'TR' ? 'Kayıt Ol' : 'Sign Up';

```
/// Social login options, you should provide callback function and
icon path.
/// Icon paths should be the full path in the assets
/// Don't forget to also add the icon folder to
the "pubspec.yaml" file. List<SocialLogin>
_socialLogins(BuildContext context) =>
<SocialLogin>[ SocialLogin(
   callback: () async =>
   _socialCallback('Google'),
   iconPath:
   'assets/images/google.png'),
 SocialLogin(
   callback: () async =>
   _socialCallback('Facebook'),
   iconPath:
   'assets/images/facebook.png'),
 SocialLogin(
   callback: () async =>
   _socialCallback('LinkedIn'),
   iconPath:
   'assets/images/linkedin.png'),
];

Future<String?>
 _socialCallback(String type)
```

```
  async { await
  _operation?.cancel();
  _operation =
    CancelableOperation.fromF
    uture(
    LoginFunctions(context).so
    cialLogin(type));
  final String? res = await
  _operation?.valueOrCancellation(); if
  (_operation?.isCompleted == true
  && res == null) {
  DialogBuilder(context)
     .showResultDialog('Internal Server Error in with $type.');
  }
  return res;
 }
}

/// Example forgot password screen
class ForgotPasswordScreen extends StatelessWidget {
 /// Example forgot password screen that user is navigated to
 /// after clicked on "Forgot Password?" text.
 const ForgotPasswordScreen({Key? key}) : super(key: key);

 @override
```

```
Widget
 build(BuildContext
 context) { return const
 Scaffold(
  body: Center(
   child: Text('FORGOT PASSWORD'),
  ),
 );
}
}
```

Output


2:23
34%
EN
Flu+
Welcome Back!
Welcome back to the best. We're always here, waiting for you!
or use your email account:
Email
Password
Forgot Password?
Not have an account? Sign Up
Login

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget
Roll No.	60
Name	Sujal Taktani
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	9

Sujal Taktani **Batch :- C**
D15A_60

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, 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 :-

```
import 'dart:convert';

import
'package:flutter/material
.dart'; import
'package:intl/intl.dart';
import 'package:http/http.dart' as http;
import
'../screens/light_mode/home_sc
reen.dart'; final formatter =
DateFormat.yMd();
 var
 enteredname=
 ''; var
 entereddate;

class SignUp extends StatefulWidget {
 const SignUp({Key? key}) : super(key:
 key); // Corrected super call @override
 State<StatefulWidget>
  createState() { return
  _SignUp();
 }
}
class DT{
DT({required
this.date}); final
DateTime date;
String get
formattefDate {
  return formatter.format(date);
 }

}




class _SignUp extends
 State<SignUp> {
 DateTime?
```

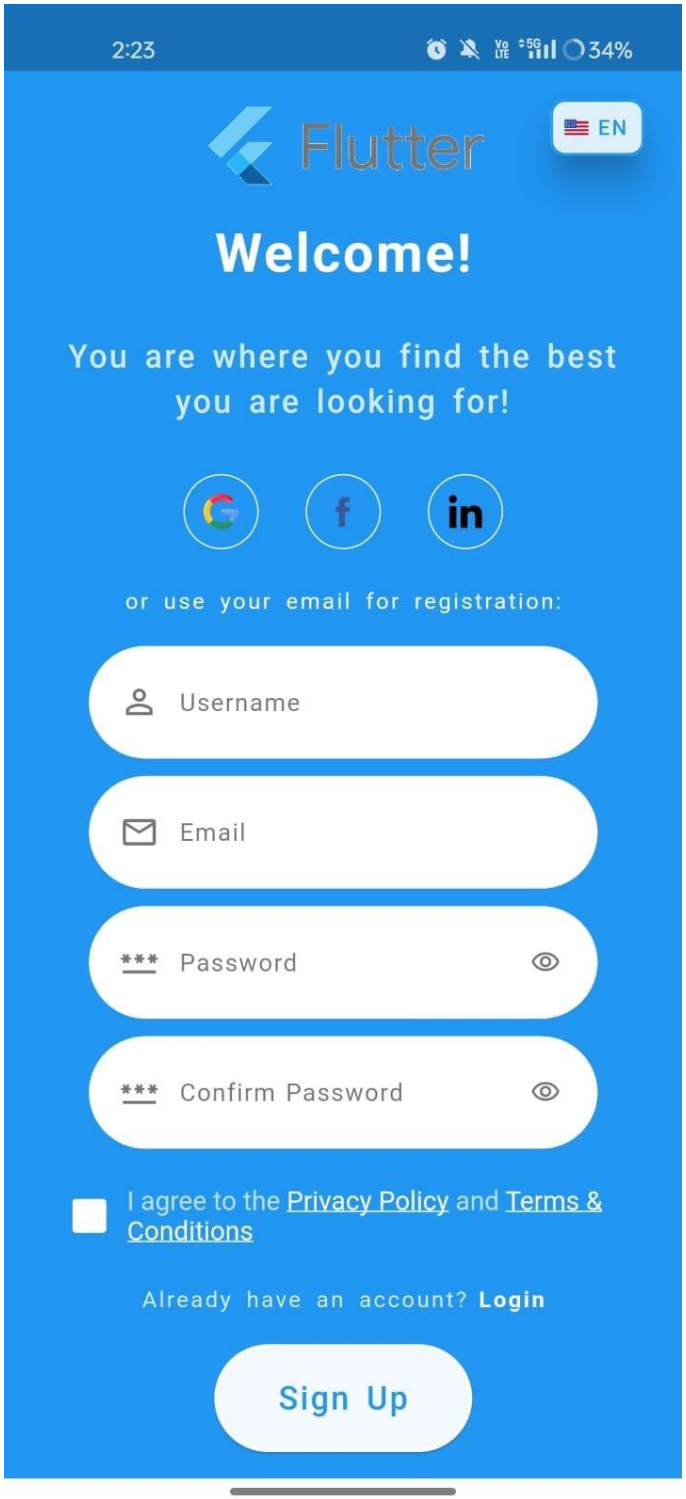
```
_selectedDate=DateTime.n
ow(); void
_presentDatePicker() async
{
 final now = DateTime.now();
 final firstDate = DateTime(now.year - 1, now.month, now.day);
 // final lastDate = DateTime(now.year + 3,
 now.month, now.day); final pickedDate =
 await showDatePicker(
   context:
   context,
   initialDate:
   now,
   firstDate:
   firstDate,
   lastDate:
   now);
  if(_selectedDate!=
 null){ setState(() {
   _selectedDate = pickedDate??_selectedDate;
 });}
}
 String get formattedDate {
 return formatter.format(DateTime.now());
}

void savedata()async{

enteredname=nameCon

troller.text;
final url=Uri.https('flutter-prep-5b74d-default-
rtdb.firebaseio.com','user-data.json');

 final response= await
   http.post(url,headers: {
   'Content-
   type':'application/json'
   },
  body: json.encode({
```

```
    'name': enteredname,
    'dateofbirth': formatter.format(_selectedDate!)
  })
   );

}

final _dateController =
TextEditingController(); final
nameController=TextEditingC
ontroller();

@override
Widget build(BuildContext
 context) { DateTime?
 selectedDate=DateTime.n
 ow();
 // var selectedDate = _selectedDate;

 return Scaffold(
  backgroundColor:
  Theme.of(context).primaryColorLi
  ght, body:  Center(
   child: Padding(
    padding: const
    EdgeInsets.all(12.0),
    child: Column(
     mainAxisSize:
     MainAxisSize.min,
     mainAxisAlignment:
     MainAxisAlignment.center,
     crossAxisAlignment:
     CrossAxisAlignment.center,
     children: [
      TextField(
       controller:
       nameController,
       keyboardType:
       TextInputType.name,
       decoration:
       InputDecoration(
       border:
       OutlineInputBorder(
          borderRadius:
```

```
BorderRadius.all(Radius.circular(
8))), iconColor: Colors.black,
label: Text(
 "Add Your
 Name",
 style:
 TextStyle(
 color:
 Colors.black,
  fontStyle: FontStyle.italic,
 ),
),
),
expands: false,
),
const SizedBox(height: 10,),
const
SizedBox(height:
10), Row(
 children:
  [
  Expand
  ed(
   child: TextField(
    controller:
    _dateController,
    decoration:
    InputDecoration(
     label: Text('Enter your
    D.O.B',style: TextStyle( color:
    Colors.black,
    fontStyle: FontStyle.italic,
   ),
   )
    ),
   ),
  ),
   Expan
  ded(
  child:
  Row(
   mainAxisAlignment:
   MainAxisAlignment.end,
   crossAxisAlignment:
   CrossAxisAlignment.center,
```

```
childre
 n: [
 Text(

selectedDate == null
    ? 'No Date Selected'
    :
    formatter.format(_selecte
    dDate!), style: const
    TextStyle(color:
    Colors.black)
 ),
 const SizedBox(width:
 10), IconButton(
   onPressed: () {
    _presentDatePicker();
   },
   icon: const Icon(
    Icons.calendar_
    month, color:
    Colors.black,
   )),
  // const SizedBox(width: 16),
  // Text(formattedDate),
 ],
 ),
 ),

 ],
),
const
SizedBox(height:
10,),
ElevatedButton(on
Pressed: (){
savedata();
 Navigator.of(context).push(MaterialPa
   geRoute(builder: (ctx)=>
   HomeScreen())
 );
}, child: const Text("Next")),
 ],
 ),
 ),
 ),
 );
 }
}
```

Output :-


2:23
34%
Flutter
EN
Welcome!
You are where you find the best you are looking for!
or use your email for registration:
Username
Email
*** Password
*** Confirm Password
I agree to the Privacy Policy and Terms & Conditions
Already have an account? Login
Sign Up

MAD & PWA Lab

Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	60

Name	Sujal Taktani
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	9

Sujal Taktani Batch :- C

D15A_60

Experiment no. :- 05

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

Theory :-

In Flutter, navigation, routing, and gestures are essential concepts for creatinginteractive 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 formanaging navigation and routing.

Routing: Routing is the mechanism used to define the paths or routes betweendifferent 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 providesvarious gesture detection widgets to handle user input.

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

Code :-

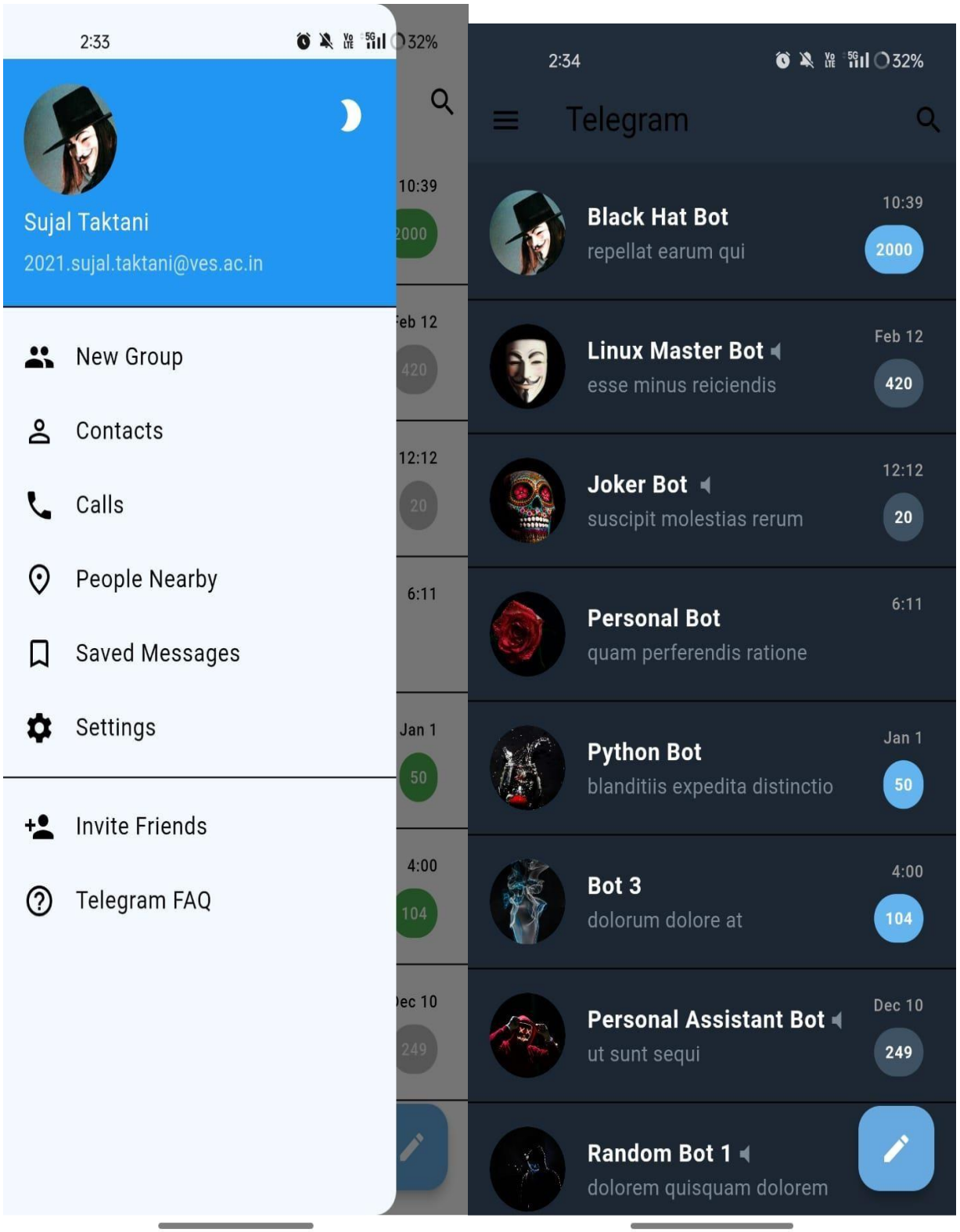
```
import 'package:flutter/material.dart';
import
'package:telegram_clone_flutter/screens/dark_mode/
drawer_dark.dart'; import
'package:telegram_clone_flutter/screens/models/cha
t_model.dart';

import 'chatting.dart';


class HomeScreenDark
 extends StatefulWidget {
 @override
 _HomeScreenDarkState createState() => _HomeScreenDarkState();
}

class _HomeScreenDarkState extends
 State<HomeScreenDark> { @override
 Widget
  build(BuildContext
  context) { return
  Scaffold(
   backgroundColor:
   Color(0xff1d2733),
   appBar: AppBar(

    backgroundColor:
    Color(0xff212d3b), title:
    Text('Telegram'),
    actions:
     <Widget>[
     Padding(
      padding: const
      EdgeInsets.all(8.0),
      child:
      Icon(Icons.search),
     )
    ],
   ),
   drawer: Theme(
```

```
 data:
Theme.of(context).cop
yWith( canvasColor:
Color(0xff1d2733),
  //This will change the drawer background to blue.
//other styles
),
 child: DrawerScreenDark(),),



 body: Padding(
  padding: const
  EdgeInsets.only(top: 10),
  child:
  ListView.separated(
    itemBuilder:
     (ctx, i) {
     return
     InkWell(
     onTap: (){
       Navigator
        .push(
        context,
        MaterialPageRoute(builder: (context) => ChatScreenDark()),
       );
      },
      child: ListTile(
       leading:
        CircleAvatar
        (radius: 28,
        backgroundImage: AssetImage(items[i].imgPath),
       ),
       title: items[i].status ?
       Text(items[i].name,style: TextStyle(fontWeight:
       FontWeight.bold,color: Colors.white),): Row(children: [
        Text(items[i].name,style: TextStyle(fontWeight:
        FontWeight.bold,color: Colors.white),),
        Icon(Icons.volume_mute,size: 18,color: Color(0xff7d8b98),)
```

```
],),
subtitle:Text(items[i].message,style: TextStyle(color:
Color(0xff7d8b98)),),

trailing:items[i].mes

sNum!=null ?

Column(
 crossAxisAlignment:
 CrossAxisAlignment.end,
 children: [


  Text(items[i].time,style: TextStyle(color:

  Colors.grey),), SizedBox(height: 7,),


  Container(

   decoration: BoxDecoration(
     color:
     items[i].status?Color(0xff64b4ef):Color(0
     xff3e5263), borderRadius:
     BorderRadius.circular(30)
   ),
   child:Padding(
    padding: const EdgeInsets.all(8.0),
    child: Text('${items[i].messNum}',style:
TextStyle(color: Colors.white,fontWeight:
FontWeight.bold),),
   ),
  )

 ],
):

Column(
 crossAxisAlignment:
 CrossAxisAlignment.end,
 children: [

  Text(items[i].time,style: TextStyle(color:
```

```
            Colors.grey),), SizedBox(height: 7,),

           ],)


         ),
        );
       },
       separatorBuilde
        r: (ctx, i) {
        return
        Divider();
       },
       itemCount: items.length),
   ),
   floatingActionButton:
     FloatingActionButton(
     child:
     Icon(Icons.create,color:
     Colors.white,),
     backgroundColor:
     Color(0xFF65a9e0),
     onPressed: (){}),
  );
 }
}
```

Output :-


2:33
32%
Sujal Taktani
2021.sujal.taktani@ves.ac.in
New Group
Contacts
Calls
People Nearby
Saved Messages
Settings
Invite Friends
Telegram FAQ
10:39
2000
Feb 12
420
12:12
20
6:11
Jan 1
50
4:00
104
Dec 10
249
2:34
32%
Telegram
Black Hat Bot
repellat earum qui
10:39
2000
Linux Master Bot
esse minus reiciendis
Feb 12
420
Joker Bot
suscipit molestias rerum
12:12
20
Personal Bot
quam perferendis ratione
6:11
Python Bot
blanditiis expedita distinctio
Jan 1
50
Bot 3
dolorum dolore at
4:00
104
Personal Assistant Bot
ut sunt sequi
Dec 10
249
Random Bot 1
dolorem quisquam dolorem

MAD & PWA Lab

Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	60
Name	Sujal Taktani
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:	9

Sujal Taktani **Batch :- C**
D15A_60

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

Code:

```
import 'dart:convert';

import
'package:flutter/material.dart'
; import
'package:http/http.dart' as
http;

class ChatScreen extends
 StatefulWidget { @override
 _ChatScreenState createState() => _ChatScreenState();
}

class _ChatScreenState extends
 State<ChatScreen> { List<Message>
 messages = [];
 TextEditingController _textEditingController =
 TextEditingController();

 @override
 Widget build(BuildContext
  context) { return Scaffold(
   appBar:
    AppBar(
    title:
    Text('Chat'),
   ),
   body: Column(
    children:
    <Widget>[
    Expanded(
      child:
       ListView.builder(
       itemCount:
       messages.length,
       itemBuilder:
       (context, index) {
```

```
          return _buildMessage(messages[index]);
        },
       ),
      ),
      _buildMessageInputArea(),
    ],
   ),
  );
}

Widget _buildMessage(Message
 message) { return Align(
  alignment: message.isUserMessage ? Alignment.centerRight :
  Alignment.centerLeft, child: Padding(
   padding: const
   EdgeInsets.all(8.0), child:
   Container(
    padding:
    EdgeInsets.all(12),
    decoration:
    BoxDecoration(
     color: message.isUserMessage ? Colors.blue :
     Colors.grey[300], borderRadius:
     BorderRadius.circular(12),
    ),
    child: Text(message.text),
   ),
```

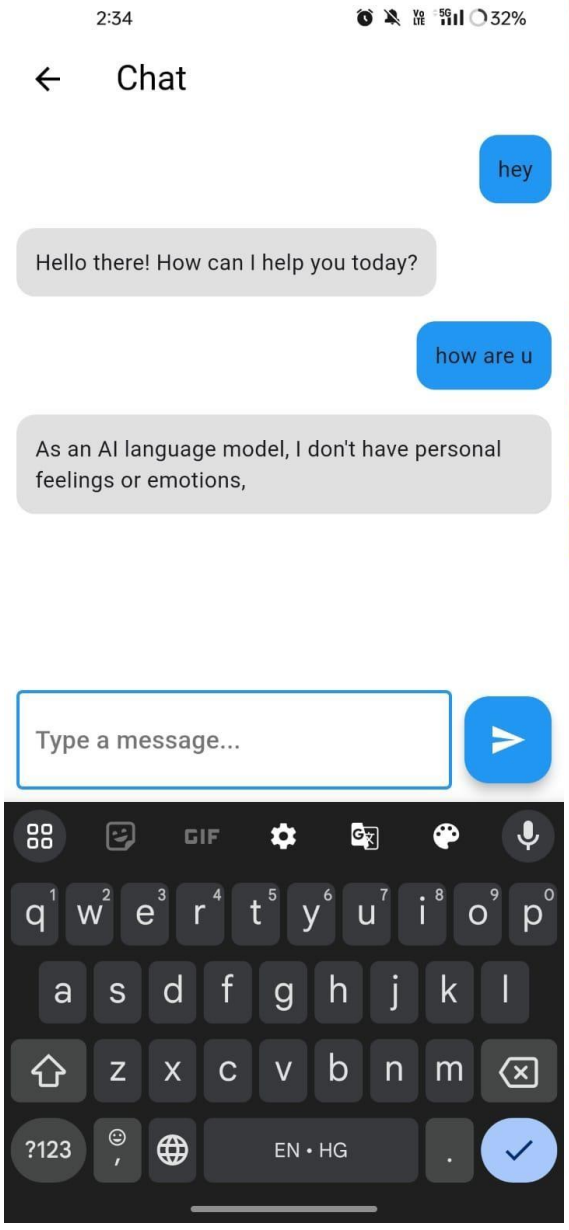
```
  ),
 );
}

Widget
 _buildMessageInputArea()
 { return Padding(
  padding: const
  EdgeInsets.all(8.0), child:
  Row(
   children:
    <Widget>[
    Expanded(
     child: TextField(
      controller:
      _textEditingController,
      decoration:
      InputDecoration(
      hintText: 'Type a
      message...', border:
      OutlineInputBorder(),
      ),
     ),
    ),
    SizedBox(width: 8),
    FloatingActionButton(
    onPressed: () {
      _sendMessage(_textEditingController.text);
     },
     child: Icon(Icons.send),
    ),
   ],
  ),
 );
}

void _sendMessage(String
 message) async { setState(() {
  messages.add(Message(message, true)); // Add user's message
  _textEditingController.clear(); // Clear text field
 });

 // Define the API endpoint URL
 var apiUrl =
```

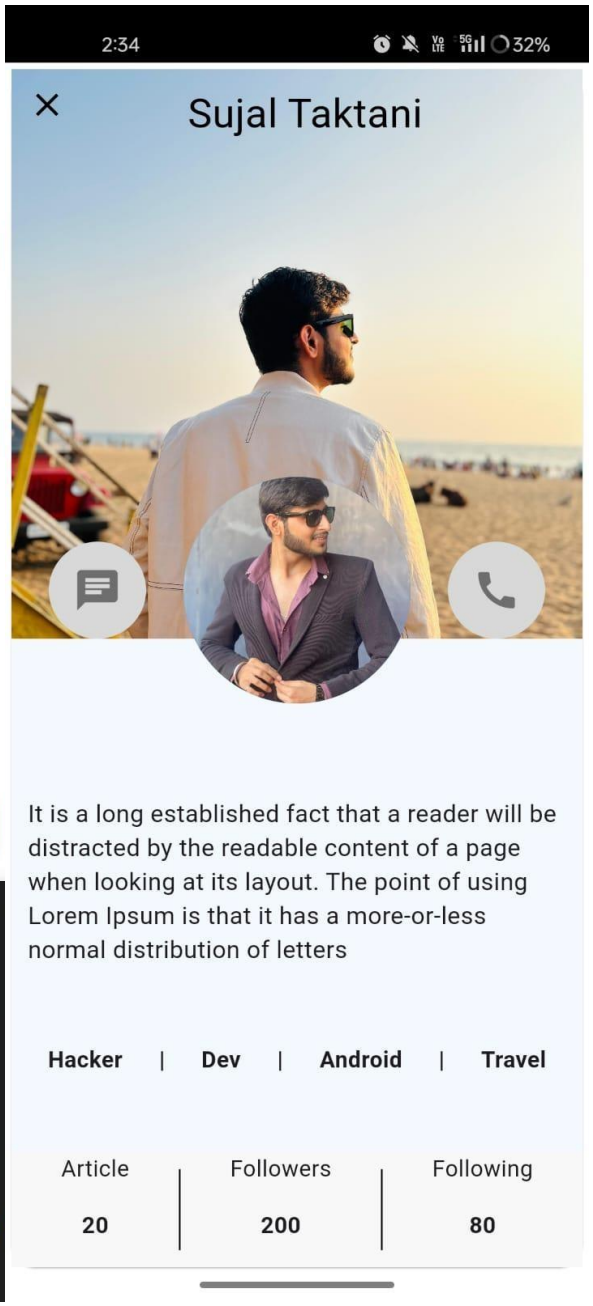
'https://generativelanguage.googleapis.com/v1/models/gemini-pro:streamGenerateContent?key=AIzaSyBUA19PoTh7qUmJPnpDiow5n7R2lvIHCKM';

```
  // Define the request body
  var requestBody =
   json.encode({ "contents": [
    {
     "role": "user",
     "parts": [{"text": message}]
    }
   ]
  });

  // Make the HTTP POST request var response = await http.post( Uri.parse(apiUrl),
   headers: {'Content-Type': 'application/json'}, body: requestBody,
  );
```

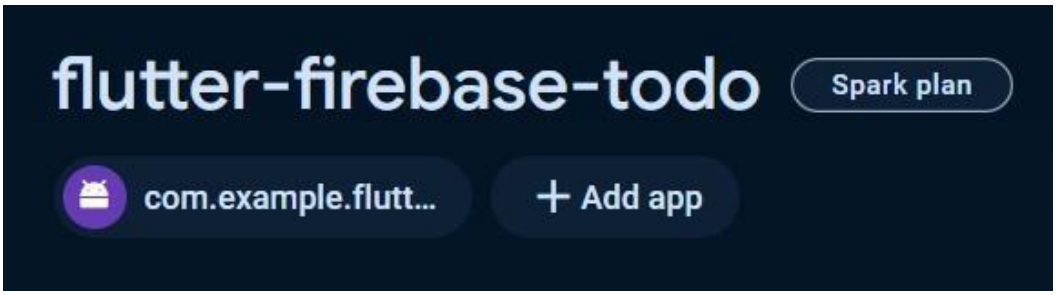
```
  // Check if the request was
  successful if
  (response.statusCode ==
  200) {
   // Parse the response JSON
   var responseBody = json.decode(response.body);

   // Extract the API response text from the JSON and
   add it to the messages list if (responseBody is List
   && responseBody.isNotEmpty) {
    var candidates =
    responseBody[0]['candidates'] as List;
    if (candidates.isNotEmpty) {
     var content =
     candidates[0]['content']; var
     parts = content['parts'] as
     List;
     if (parts.isNotEmpty) {
      var apiResponseText =
      parts[0]['text']; setState(() {
       messages.add(Message(apiResponseText, false)); // Add API
       response
      });
     }
    }
   }
  } else {
   // Handle the error if the
   request fails print('Error:
   ${response.statusCode}');
  }
 }
}

class
 Message {
 final String
 text;
 final bool isUserMessage;

 Message(this.text, this.isUserMessage);
}
```

Implementation:


2:34
32%
Chat
hey
Hello there! How can I help you today?
how are u
As an AI language model, I don't have personal feelings or emotions,
Type a message...
GIF
EN • HG


2:34
32%
Sujal Taktani
It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout. The point of using Lorem Ipsum is that it has a more-or-less normal distribution of letters
Hacker | Dev | Android | Travel
Article
20
Followers
200
Following
80

```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
```



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.	60
Name	Sujal Taktani
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	11

Name : Sujal Taktani
Div : D15A
Roll No : 60
Batch : C

PWA 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.

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

7. Lower Development Cost Progressive web apps can be installed on the user device like a native device, but it does not demand submission on an App Store. This makes it far more cost-effective than native mobile applications while offering the same set of functionalities.

Pros and cons of the Progressive Web App The main features are:

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

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

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

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

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

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

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

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

Linkable — Easily shared via URL without complex installations.

Offline — Once more it is about putting the user before everything, avoiding the usual error message in case of weak or no connection

. The PWA are based on two particularities: first of all the ‘skeleton’ of the app, which recalls the page structure, even if its contents do not respond and its elements include the header, the page layout, as well as an illustration that signals that the page is loading.

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

Folder Structure and icon size

Index.html

```
<!DOCTYPE html>
<html>


<head>
    <meta name="apple-mobile-web-app-status-bar" content="#aa7700">
    <meta name="theme-color" content="black">
    <link rel="manifest" href="manifest.json">
    <script src="service-worker.js"></script>
    <title>
        Index
    </title>
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.
css">
```

```
<body>


    <nav class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-
toggle="collapse" data-target="#mynavbar">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
```

```
                <li><a href="signup.html">
                        <span class="glyphicon glyphicon-user" /> Sign-Up
                        </a>
                </li>
                <li>
                    <a href="login.html">
                        <span class="glyphicon glyphicon-log-in" /> Login
                        </a>
                </li>
            </ul>
        </div>


    </div>


</nav>
<div class="banner-image">
    <div class="container">
        <div class="banner-content" style="margin-left :25%">
            <h1>All premium plants available here</h1>
            <p>Flat 30% to our new customers</p> <br>
            <a href="product.html" class="btn btn-danger btn-lg
            active">Shop
Now
</a>
        </div>
    </div>
</div>
<footer>
    <div class="container">
        <p style="text-align:center;">Copyright © Purity Plants. All
```

```
Rights Reserved and Contact Us: +91 8432777111 </p>
        </div>
    </footer>
    <script>
        // Add event listener to execute code when page loads
        window.addEventListener('load', () => {
            // Call registerSW function when
            page loads registerSW();
        });

        // Register the Service
        Worker async function
        registerSW() {
            // Check if browser supports Service
            Worker if ('serviceWorker' in
            navigator) {
                try {
```

```
                // Register the Service Worker named
                'serviceworker.js' await
                navigator.serviceWorker.register('service-
                worker.js');
            }
            catch (e) {
                // Log error message if registration fails
                console.error('ServiceWorker registration failed: ', e);
            }
        }
    }
    if ('Notification' in window) {
        Notification.requestPermission().then(function
        (permission) {
            if (permission === 'granted') {
                console.log('Notification permission
```

Manifest.json

```
{
  "name":"PWA
  Tutorial",
  "short_name":"PWA",
  "start_url":"index.html",
  "display":"standalone",
  "background_color":"#5
  900b3",
```

```
  "type":"image/png",
  "purpose":"any
  maskable"
  },
  {
  "src":"images/plant 512 X 512.png",
  "sizes":"512x512",
  "type":"image/png",
```

Service-worker.json

```
// service-worker.js


const CACHE_NAME = 'my-ecommerce-
app-cache-v1'; const urlsToCache = [
 '/',
 'cart.html',
 'index.html',
 'product.html',
 'shop.html',
 'style.css',
 'success.html',
 'service-
 worker.js',
 'manifest.json',
 'offline.html'
 // Add more files to cache as needed
```

```
  );
 });


self.addEventListener('activate', function(event) {
 // Perform activation steps
 event.waitUntil(
   caches.keys().then(function(cache
     Names) { return Promise.all(
       cacheNames.map(function(cac
         heName) { if (cacheName
         !== CACHE_NAME) {
           return caches.delete(cacheName);
         }
       })
     );
   })
  );
 });


// Fetch event listener
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));
});
```

```
// Sync event listener
self.addEventListener('sync',
function(event) {
 if (event.tag ===
    'syncMessage') {
    console.log("Sync
    successful!");
 }
});

// Push event listener
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);
    }
  }
});


self.addEventListener('activate', async ()
 => { if (Notification.permission !==
 'granted') {
   try {
     const permission = await
     Notification.requestPermission(); if (permission
     === 'granted') {
       console.log('Notification permission granted.');
     } else {
       console.warn('Notification permission denied.');
     }
   } catch (error) {
     console.error('Failed to request notification permission:', error);
   }
  }
});


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
```

Product.html

```
<!DOCTYPE html>
<html>


   <head>
   <title>
        product
   </title>
        <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min
.css" >


        <!--jQuery library-->
```

```
    </head>
        <body>
            <nav class = "navbar navbar-inverse navbar-fixed-top">
                    <div class ="container">
                        <div class ="navbar-header">
                            <button type="button" class
="navbar-toggle" data-toggle="collapse" data-
target="#mynavbar">
                                <span class="icon-bar"></span>
                                <span class="icon-bar"></span>
                                <span class="icon-bar"></span>
                                </button>
                                <a class="navbar-brand" href="index.html">Purity
                                Plants</a>


                        </div>
                        <div class="collapse navbar-collapse" id="mynavbar">
                            <ul class="nav navbar-nav navbar-right">
                                <li>
                                    <a href="cart.html">
                                        <span
class="glyphicon glyphicon-shopping-cart">  Cart
</span> </a>
                                </li>
                                <li>
                                    <a href="setting.html">
                                    <span class="glyphicon glyphicon-user">
Setting</span>
</a>
```

```
</li>                        <li>
                                <a href="index.html">
                                    <span class="glyphicon glyphicon-log-
                                    out">
Logout</span
></a>


                            </li>
                        </ul>
                    </div>


                </div>

            </nav>

            <div class =" container" style="margin-top: 5%;">

            <div class ="jumbotron">
```

```
<h1> Welcome  to our Purity Plants!  </h1>
<p>We have the best quality and rare breed of plants at our botany </p>
</div>


<div class="row text-center">
<div class="  col-md-3 col-sm-6  thumbnail " >
<img src="/product-images/1.webp">
<div class="caption">
<h2>Corpse Flower</h2>
<p>Large foul-smelling bloom.</p>


</div>
<div class=" btn btn-primary  btn-block btn-md btn-success">
300
</div>
</div>



<div class="  col-md-3 col-sm-6  thumbnail " >
<img src="/product-images/2.webp">
<div class="caption">
<h2>Jade Vine</h2>
<p>Turquoise flowers in clusters.</p>


</div>
<div class=" btn btn-primary  btn-block btn-md btn-success">
240
</div>
```

```
</div>


<div class="  col-md-3 col-sm-6  thumbnail " >
<img src="/product-images/3.webp">
<div class="caption">
<h2>Wollemi Pine</h2>
<p>"Living fossil" from Australia</p>

</div>
<div class=" btn btn-primary  btn-block btn-md btn-success">
290
</div>
</div>

<div class="  col-md-3 col-sm-6  thumbnail " >
<img src="/product-images/4.webp">
```

```
<div class="caption">
<h2>Ghost Orchid</h2>
<p>Ghostly white floating flowers.</p>
</div>


<div class=" btn btn-primary  btn-block btn-md btn-success">
500
</div>


</div>


</div>



<div class="row text-center">


<div class="  col-md-3 col-sm-6  thumbnail " >
<img src="/product-images/5.webp">
<div class="caption">
<h2>Lithops</h2>
<p>Succulents resembling rocks.</p>
</div>
<div class=" btn btn-primary  btn-block btn-md btn-success">
499
</div>
</div>
```

```
<div class="  col-md-3 col-sm-6  thumbnail " >
<img src="/product-images/6.webp">
<div class="caption">
<h2>Black Bat Flower</h2>
<p>Dark purple bat-like flowers.</p>
</div>
<div class=" btn btn-primary  btn-block btn-md btn-success">
129
</div>
</div>



<div class="  col-md-3 col-sm-6  thumbnail " >
<img src="/product-images/7.webp">
      <div class="caption">
     <h2>Venus Flytrap</h2>
     <p>Carnivorous plant trapping insects.</p>
```

```
    </div>
    <div class=" btn btn-primary  btn-block btn-md btn-
    success"> 199
</div>


</div>


<div class="  col-md-3 col-sm-6  thumbnail " >
<img src="/product-images/8.webp">
<div class="caption">
<h2>Kadupul Flower </h2>
<p>Night-blooming Sri Lankan flower.</p>
</div>
<div class=" btn btn-primary  btn-block btn-md btn-success">
209
</div>


</div>
</div>




<div class="row text-center">


<div class="  col-md-3 col-sm-6  thumbnail " >
<img src="/product-images/9.webp">
  <div class="caption">
<h2>Rainbow Eucalyptus</h2>
```

```
<p>Multicolored peeling bark.</p>
</div>
<div class=" btn btn-primary  btn-block btn-md btn-success">299 </div>


</div>


<div class="  col-md-3 col-sm-6  thumbnail " >
<img src="/product-images/10.webp">
  <div class="caption">
<h2>Corkscrew Vine</h2>
<p>Fragrant spiral-shaped flowers.</p>
</div>
<div class=" btn btn-primary  btn-block btn-md btn-success">300</div>
</div>


<div class="  col-md-3 col-sm-6  thumbnail " >
```

```
<img src="/product-images/11.webp">
  <div class="caption">
<h2>Night-blooming Cereus:</h2>
<p>Fragrant nocturnal blooms.</p>
</div>
<div class=" btn btn-primary  btn-block btn-md btn-success">249 </div>
</div>


<div class="  col-md-3 col-sm-6  thumbnail " >
<img src="/product-images/12.webp">
  <div class="caption">
<h2>Bleeding Tooth Fungus</h2>
<p>Blood-like liquid oozes.</p>
</div>
<div class=" btn btn-primary  btn-block btn-md btn-success">249 </div>
</div>
</div>
</div>
```

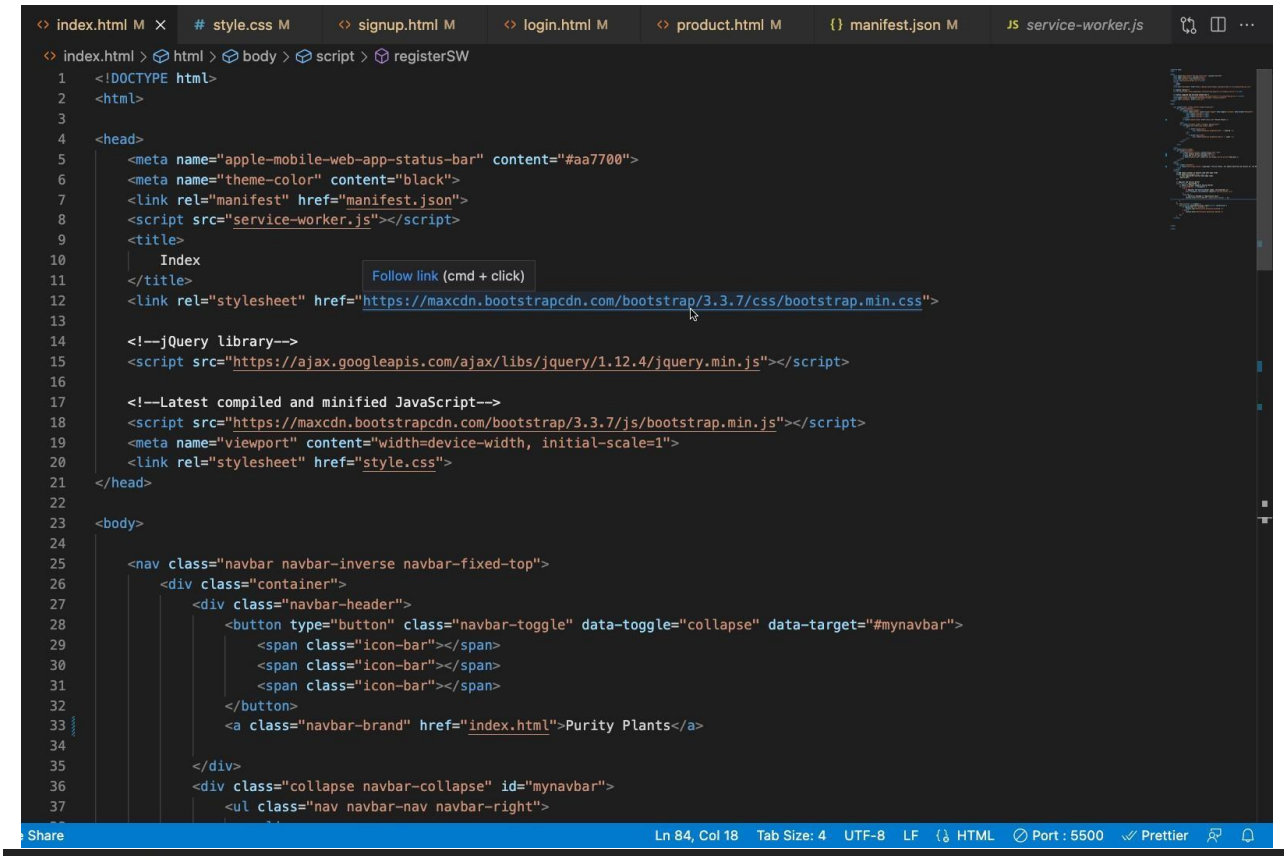
Style.css

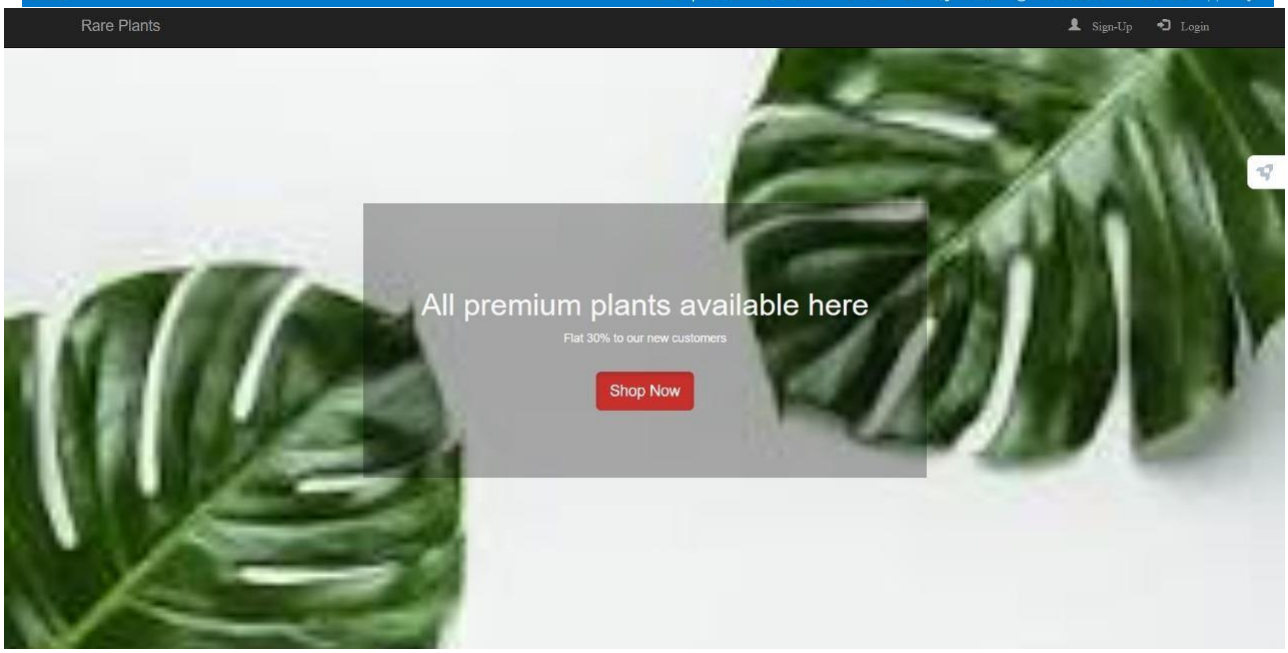
```
.banner-image
{padding-top:
75px; padding-
bottom: 50px;
text-align:
```

```
.banner-content{ position:
  relative; padding-top: 6%;
padding-bottom: 6%;


margin-top: 12%;
margin-bottom: 12%;
background-color: rgba(0, 0, 0, 0.3);
width: 50%;
text-align:center;
}
footer
{
padding: 10px 0;
background-color: #110011; color:#9d9d9d;
bottom: 0;
width: 100%;
}
.container{ width:90%;
margin:auto;
overflow:hidden;


}
```

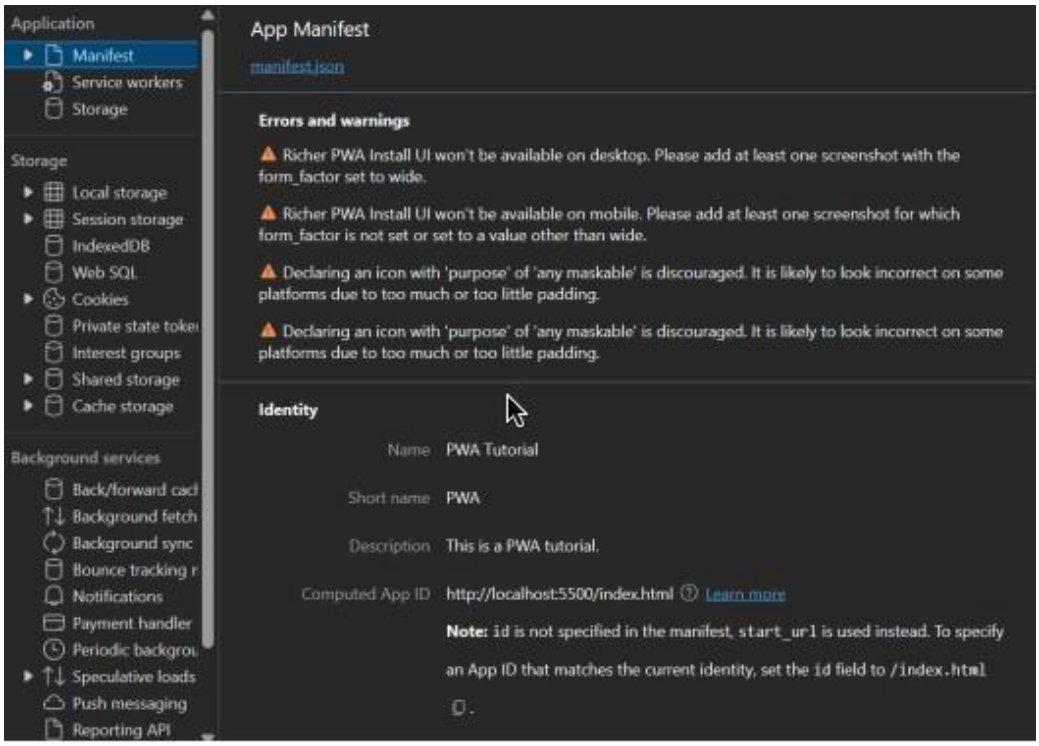
Starting the Server

:



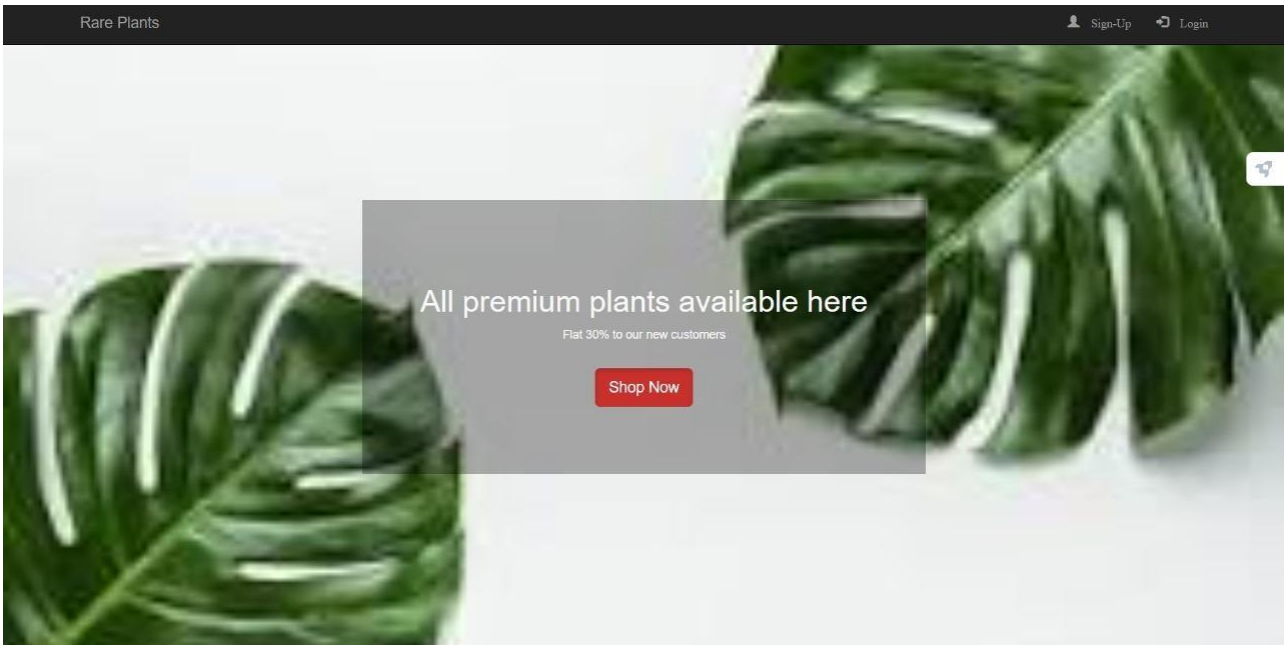


Now go to developer options -> Application->Manifest



Now to install PWA , click on Three dots(...) -> Apps -> Install PWA


flutter 2
Sublime Text
PWA Tutorial
2c4244fe-dcbd-42...1.jpg
ZEB-MAX_NI..._UM.pdf
Simulator Screen Recordi...8.56.mp4
java
removeVowels.java
Other
Screenshots



Conclusion: Hence We wrote meta data of our Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”. And it is currently added Successfully on the Desktop

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.	60
Name	Sujal Taktani
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	11

Name: Sujal Taktani
Division: D15A
RollNo:60
Batch:C

Experiment No 8

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

Service Worker

Service Worker is a script that works on browser background without user interaction independently. Also, It resembles a proxy that works on the user side. With this script, you cantrack 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 thatyou need to persist and reuse across restarts, you can use IndexedDB databases.

What can we do with Service Workers?

- You can dominate **Network Traffic**
  You can manage all network traffic of the page and do any

manipulations. For example, when the page requests a CSS file, you can send plain text as a response or when the page requests an HTML file, you can send a png file as a response. You can also send a true response too.

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

- You can manage **Push Notifications**

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

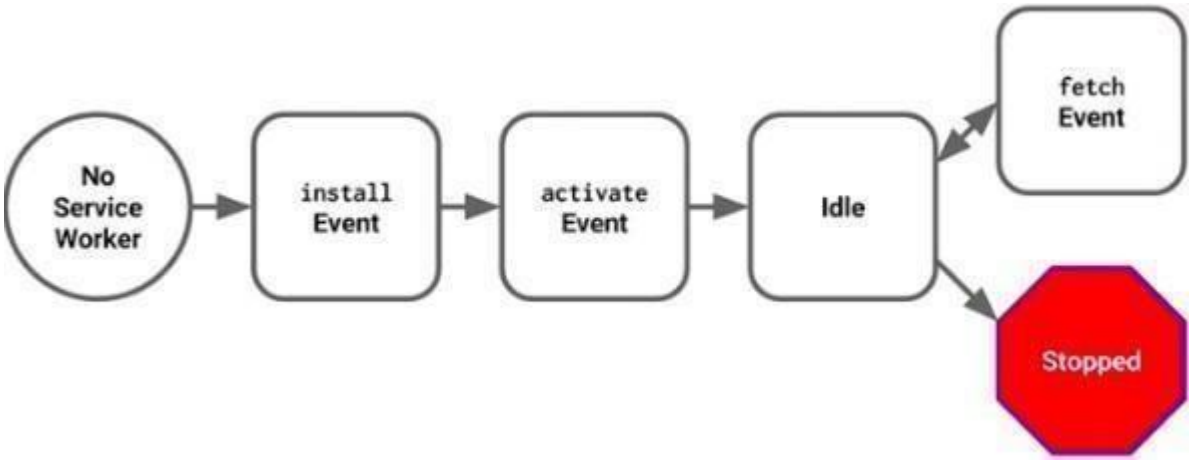
- You can **Continue**
  Although Internet connection is broken, you can start any process with Background Sync of Service Worker.

What can't we do with Service Workers?

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

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

Service Worker Cycle



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

- Registration
- Installation
- Activation

Registration

To install a service worker, you need to register it in your main JavaScript code. Registrationtells 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('/servi
 ce-worker.js')
 .then(function(registration) {
  console.log('Registration successful, scope is:', registration.scope);
 })
 .catch(function(error) {
  console.log('Service worker registration failed, error:', error);
 });
}
```

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

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

You can also set an arbitrary scope by passing in an additional parameter

when registering. For example: main.js

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

In this case we are setting the scope of the service worker to /app/, which means the

service worker will control requests from pages like /app/, /app/lower/ and /app/lower/lower, but not from pages like /app or /, which are higher.

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

main.js

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

Installation

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

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

service-worker.js

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

Activation

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

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

service-worker.js

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

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

Code in service-worker.js

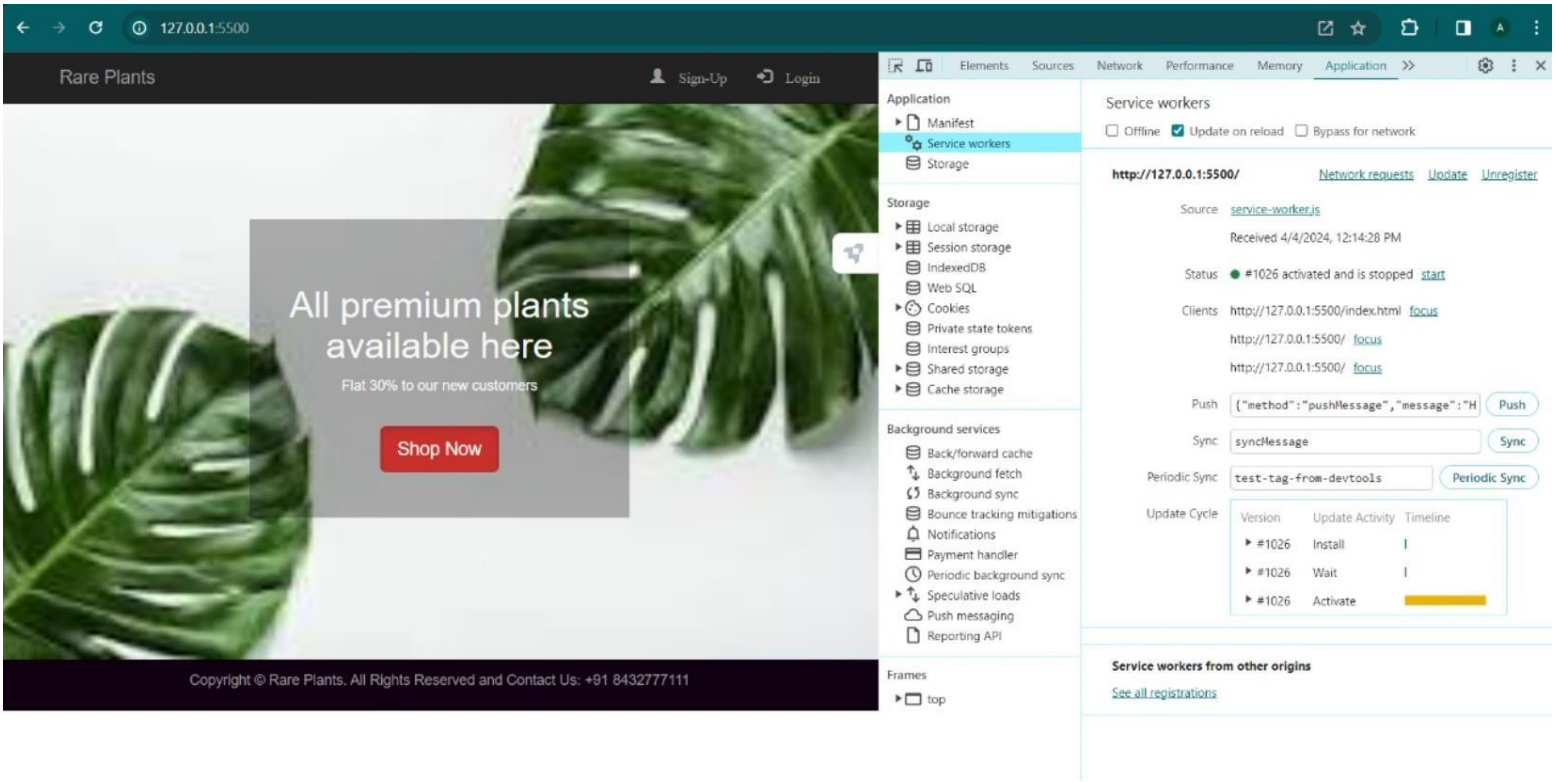
```
self.addEventListener('install', function(event) {
 event.waitUntil(
   caches.open(CACHE_NAME)
     .then(function(cache) {
       console.log('Opened cache');
       return cache.addAll(urlsToCache)
         .catch(function(error) {
           console.error('Cache.addAll error:', error);
         });
     })
 );
});

self.addEventListener('activate', function(event) {
 // Perform activation steps
 event.waitUntil(
   caches.keys().then(function(cacheNames) {
     return Promise.all(
       cacheNames.map(function(cacheName) {
         if (cacheName !== CACHE_NAME) {
           return caches.delete(cacheName);
         }
       })
     );
   })
 );
```

Code in index.html

```
<script>
      // Add event listener to execute code when page loads
      window.addEventListener('load', () => {
          // Call registerSW function when page loads
          registerSW();
      });
```

```
// Register the Service Worker
async function registerSW() {
    // Check if browser supports Service Worker
    if ('serviceWorker' in navigator) {
        try {
            // Register the Service Worker named 'serviceworker.js'
            await navigator.serviceWorker.register('service-worker.js');
        }
        catch (e) {
            // Log error message if registration fails
            console.error('ServiceWorker registration failed: ', e);
        }
    }
}
```

Output:



Conclusion: Hence We Successfully Registered our Service Worker on the Progressive Web App and it is activated as well as running

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.	60
Name	Sujal Taktani
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	11

Name: Sujal
Taktani Division:
D15A
Roll No:60
Batch : C

Experiment No 9

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.

```
self.addEventListener("fetch", function (event) {
    const req = event.request;
    const url = new URL(req.url);

    if (url.origin === location.origin) {
        event.respondWith(cacheFirst(req));
    }
    else {
        event.respondWith(networkFirst(req));
    }
});

async function cacheFirst(req) {
    return await caches.match(req) || fetch(req);
}

async function networkFirst(req) {
    const cache = await caches.open("pwa-dynamic");
    try {
        const res = await fetch(req);
        cache.put(req, res.clone());
        return res;
    } catch (error) {
        const cachedResponse = await cache.match(req);
        return cachedResponse || await caches.match("./noconnection.json");
    }
}
```

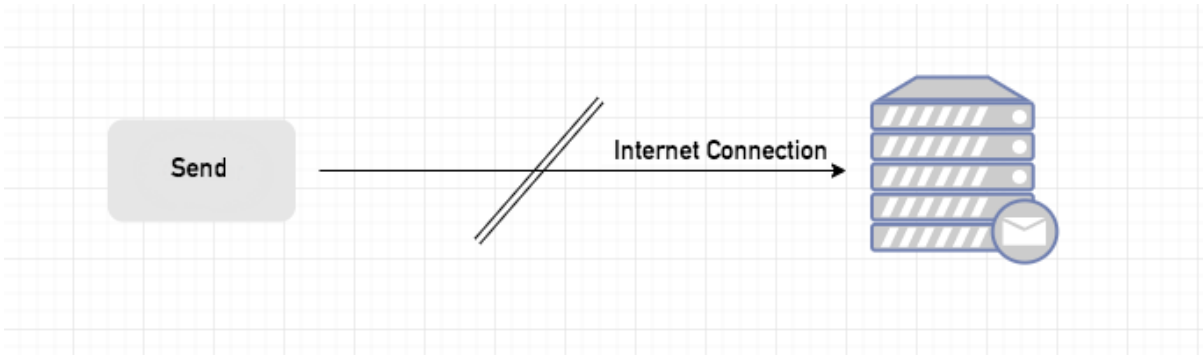
Sync Event

Background Sync is a Web API that is used to delay a process until the Internet connection is stable. We can adapt this definition to the real

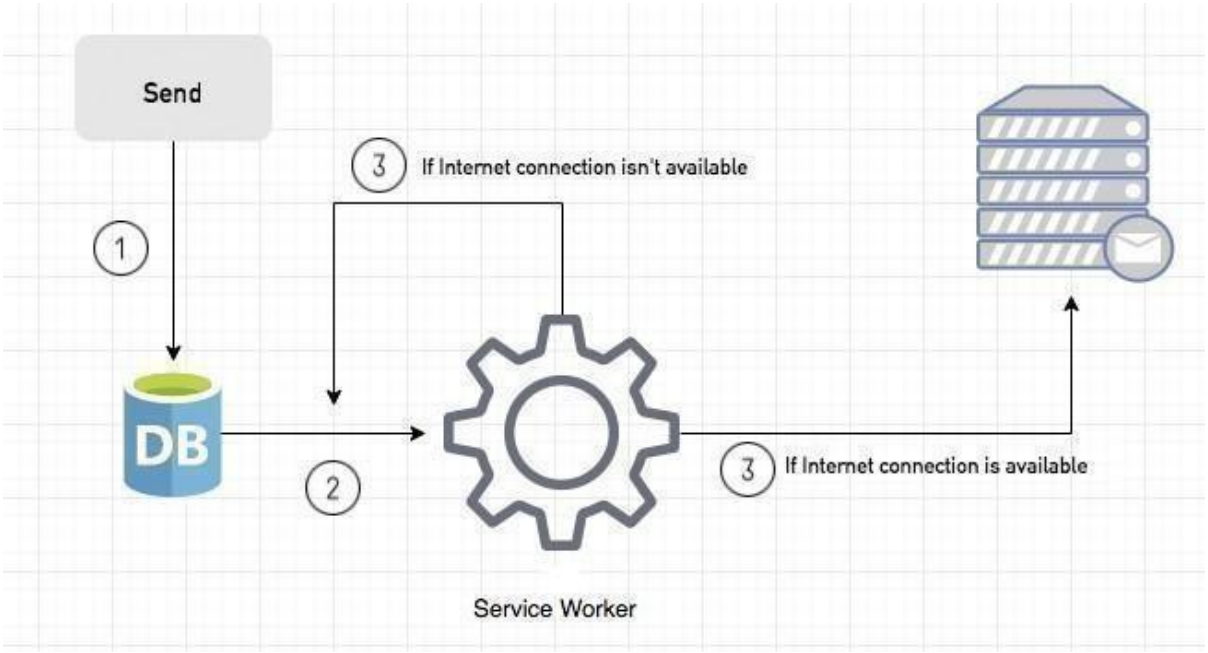
world; there is an e-mail client application that works on the browser and we want to send an email with this tool. Internet connection is broken while we are writing e-mail content and we didn’t realize it. When completing the writing, we click the send button.

Here is a job for the Background Sync.

The following view shows the classical process of sending email to us. If the Internet Connection is broken, we can't send any content to Mail Server.



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



1. When we click the "send" button, email content will be saved to IndexedDB.
2. Background Sync registration.
3. **If the Internet connection is available**, all email content will be read and sent to Mail Server.
   If the Internet connection is unavailable, the service worker waits until the connection is available even though the window is closed. When it is available, email content will be sent to Mail Server.

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

Event Listener for Background Sync Registration

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

Event Listener for sw.js

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

Push Event

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

We can check in the following example. “Notification.requestPermission();” is the necessary line to show notification to the user. If you don’t want to show any notification, you don’t need this line.

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

```
self.addEventListener('push', event => {
    if (event && event.data) {
        var data = event.data.json();
        if (data.method === "pushMessage") {
            event.waitUntil(self.registration.showNotification("Test App", {
                body: data.message
            }));
        }
    }
});
```

You can use Application Tab from Chrome Developer Tools for testing push notification
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”.

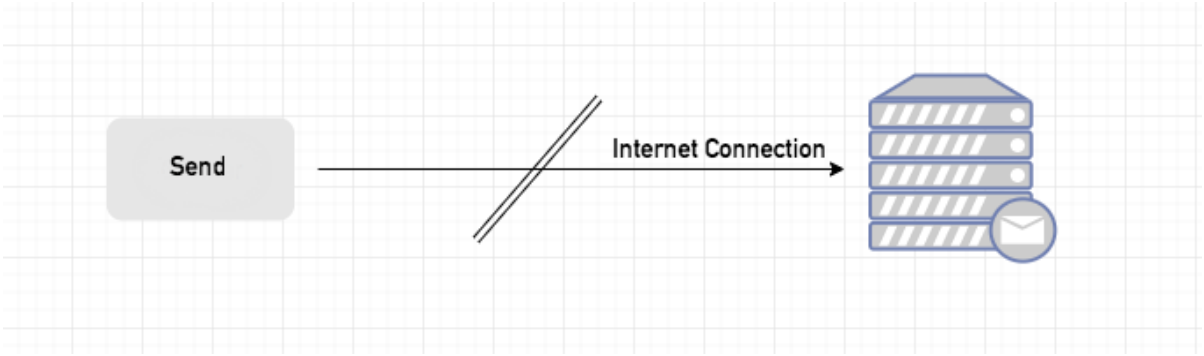
- **CacheFirst** - In this function, if the received request has cached before, the cached response is returned to the page. But if not, a new response requested from the network.
- **NetworkFirst** - In this function, firstly we can try getting an updated response from the network, if this process completed 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.

```
self.addEventListener("fetch", function (event) {
    const req = event.request;
    const url = new URL(req.url);
```

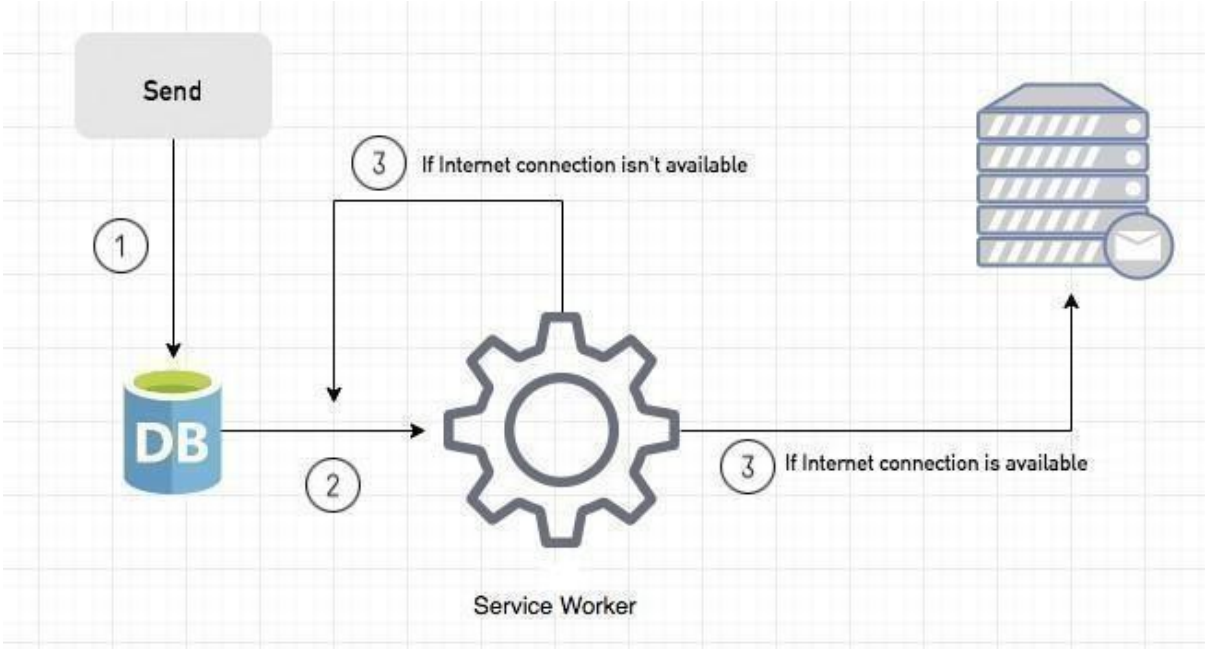
Sync Event

Background Sync is a Web API that is used to delay a process until the Internet connection is stable. We can adapt this definition to the real world; there is an e-mail client application that works on the browser and we want to send an email with this tool. Internet connection is broken while we are writing e-mail content and we didn't realize it. When completing the writing, we click the send button.

Here is a job for the Background Sync.
The following view shows the classical process of sending email to us. If the Internet Connection is broken, we can't send any content to Mail Server.



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



1. When we click the “send” button, email content will be saved to IndexedDB.
2. Background Sync registration.
3. **If the Internet connection is available**, all email content will be read and sent to Mail Server.
   If the Internet connection is unavailable, the service worker waits until the connection is available even though the window is closed. When it is available, email content will be sent to Mail Server.

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

Event Listener for Background Sync Registration

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

Event Listener for sw.js

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

Push Event

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

We can check in the following example.
“Notification.requestPermission();” is the necessary line to show notification to the user. If you don’t want to show any notification, you don’t need this line.

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

```
self.addEventListener('push', event => {
    if (event && event.data) {
        var data = event.data.json();
        if (data.method === "pushMessage") {
            event.waitUntil(self.registration.showNotification("Test App", {
                body: data.message
            }));
        }
    }
});
```

You can use Application Tab from Chrome Developer Tools for testing push notification. Code:

In index.html

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

This code sends notification permission to your Device , and click on allow to send push notification service-worker.js

```
// service-worker.js


const CACHE_NAME = 'my-ecommerce-app-cache-v1';
const urlsToCache = [
 '/',
 'cart.html',
 'index.html',
 'product.html',
 'shop.html',
 'style.css',
 'success.html',
 'service-worker.js',
 'manifest.json',
 'offline.html'
 // Add more files to cache as needed
];


self.addEventListener('install', function(event) {
 event.waitUntil(
   caches.open(CACHE_NAME)
     .then(function(cache) {
       console.log('Opened cache');
       return cache.addAll(urlsToCache)
         .catch(function(error) {
           console.error('Cache.addAll error:', error);
         });
     })
```

```
 );
});

self.addEventListener('activate', function(event) {
 // Perform activation steps
 event.waitUntil(
   caches.keys().then(function(cacheNames) {
     return Promise.all(
       cacheNames.map(function(cacheName) {
         if (cacheName !== CACHE_NAME) {
           return caches.delete(cacheName);
         }
       })
     );
   })
 );
});

// Fetch event listener
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));
});

// Sync event listener
self.addEventListener('sync', function(event) {
 if (event.tag === 'syncMessage') {
   console.log("Sync successful!");
 }
});

// Push event listener
self.addEventListener("push", function (event) {
 if (event && event.data) {
   try {
```

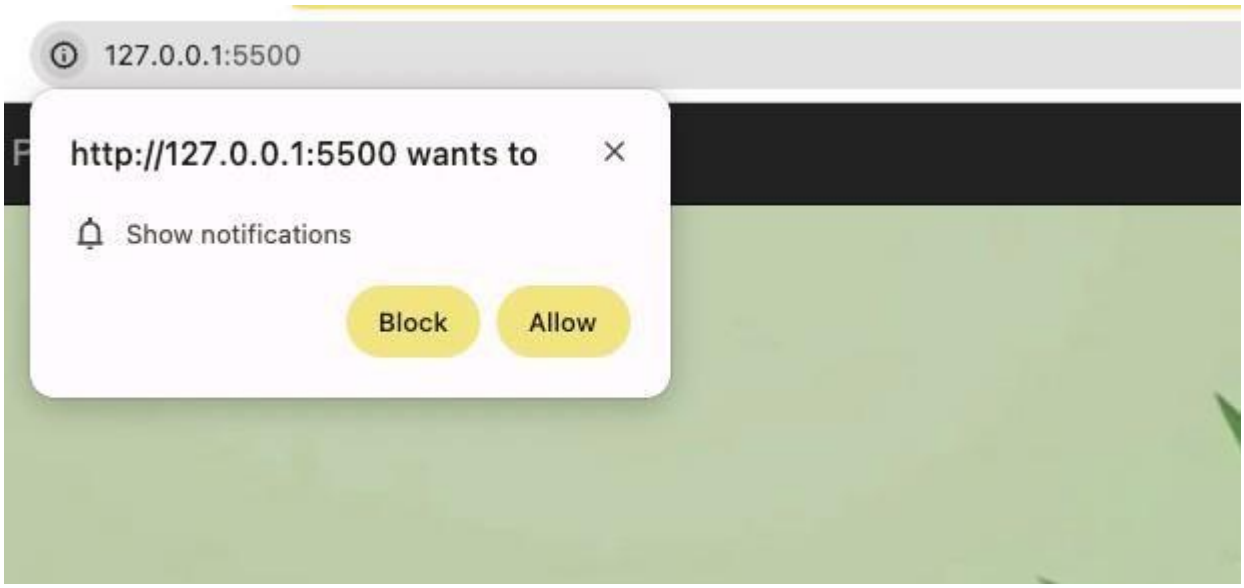
```
var data = event.data.json();
if (data && data.method === "pushMessage") {
```

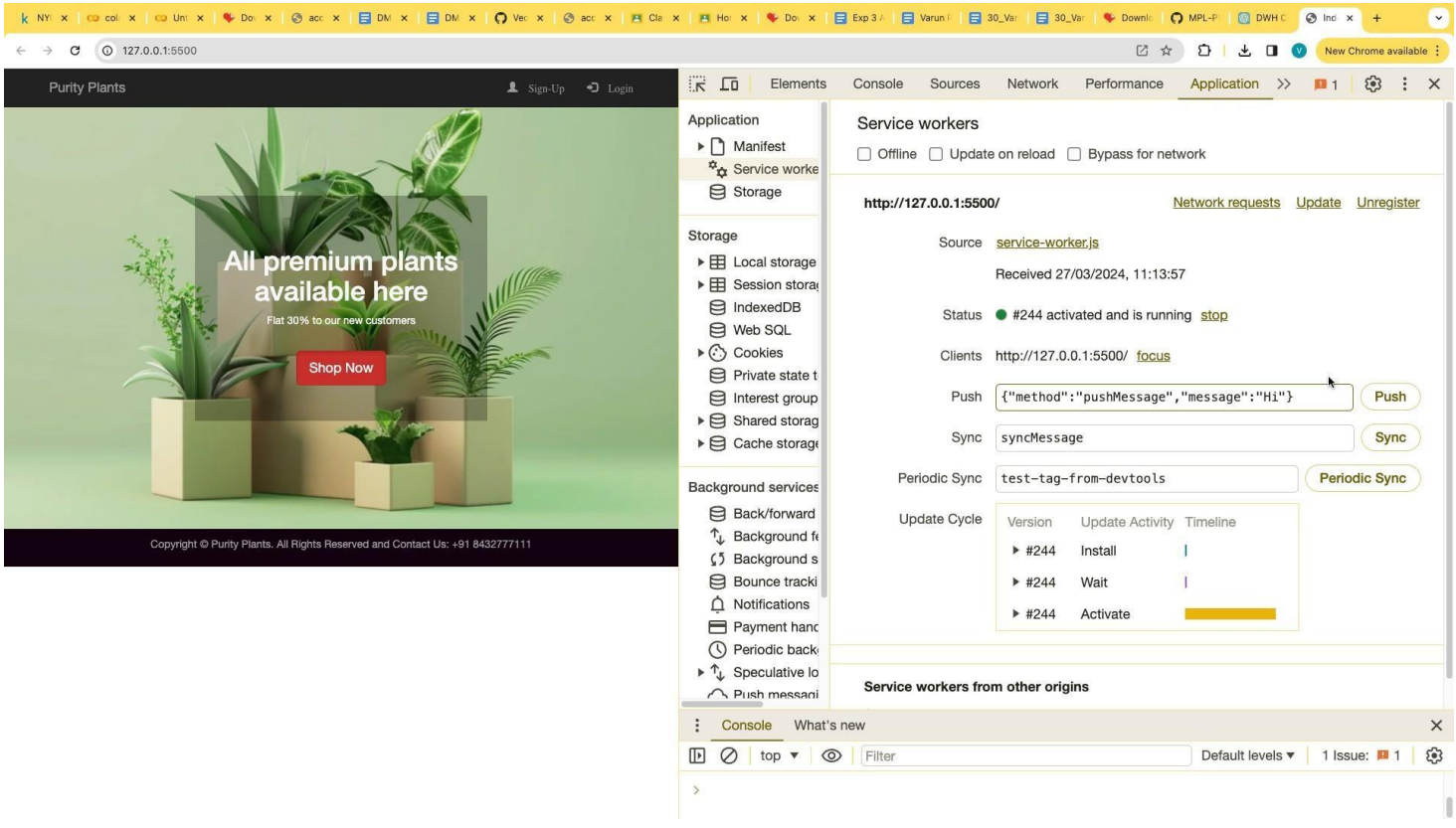
```
      console.log("Push notification sent");
      self.registration.showNotification("Ecommerce website", { body:
data.message });
    }
  } catch (error) {
    console.error("Error parsing push data:", error);
  }
}
});

self.addEventListener('activate', async () => {
 if (Notification.permission !== 'granted') {
  try {
    const permission = await Notification.requestPermission();
    if (permission === 'granted') {
      console.log('Notification permission granted.');
    } else {
      console.warn('Notification permission denied.');
    }
  } catch (error) {
    console.error('Failed to request notification permission:', error);
  }
 }
});

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"));
      };




 });
```

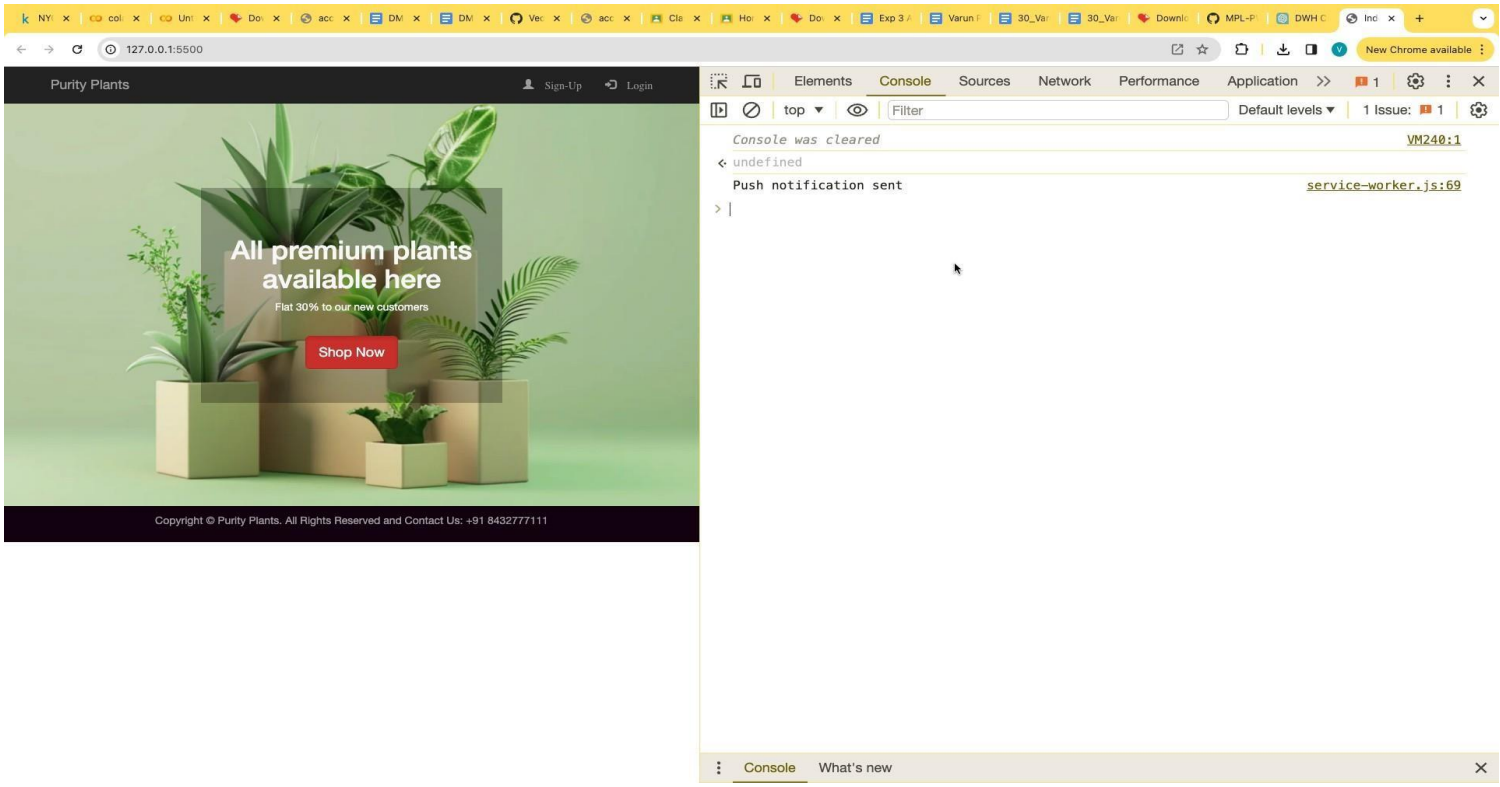
```
  }      r);
})     });
.
  c
  a
  t
  c
  h
  (
  f
  u
  n
  c
  t
  i
  o
  n

  (
  e
  r
  r
  o
  r
  )

  {

  r
  e
  j
  e
  c
  t
  (
  e
  r
  r
  o
```

```
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:


127.0.0.1:5500
http://127.0.0.1:5500 wants to
Show notifications
Block
Allow


127.0.0.1:5500
New Chrome available
Purity Plants
Sign-Up
Login
All premium plants available here
Flat 30% to our new customers
Shop Now
Copyright © Purity Plants. All Rights Reserved and Contact Us: +91 8432777111
Elements
Console
Sources
Network
Performance
Application
Application
Manifest
Service worke
Storage
Storage
Local storage
Session storag
IndexedDB
Web SQL
Cookies
Private state t
Interest group
Shared storag
Cache storage
Background services
Back/forward
Background f
Background s
Bounce tracki
Notifications
Payment hanc
Periodic back
Speculative lo
Push messagi
Service workers
Offline
Update on reload
Bypass for network
http://127.0.0.1:5500/
Network requests
Update
Unregister
Source
service-worker.js
Received 27/03/2024, 11:13:57
Status
#244 activated and is running
stop
Clients
http://127.0.0.1:5500/
focus
Push
{"method":"pushMessage","message":"Hi"}
Push
Sync
syncMessage
Sync
Periodic Sync
test-tag-from-devtools
Periodic Sync
Update Cycle
Version
Update Activity
Timeline
#244
Install
#244
Wait
#244
Activate
Service workers from other origins
Console
What's new
top
Filter
Default levels
1 Issue: 1



Conclusion: Hence we implemented methods like fetch,sync,and push on the service worker , and if we push the message, it says “notification recieved” on the desktop. So the push, sync , and fetch method is implemented successfull.

MAD & PWA Lab
Journal

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	60
Name	Sujal Taktani
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	11

Name: Sujal Taktani
Div: D15A
Roll No: 60
Batch C

Experiment No 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.

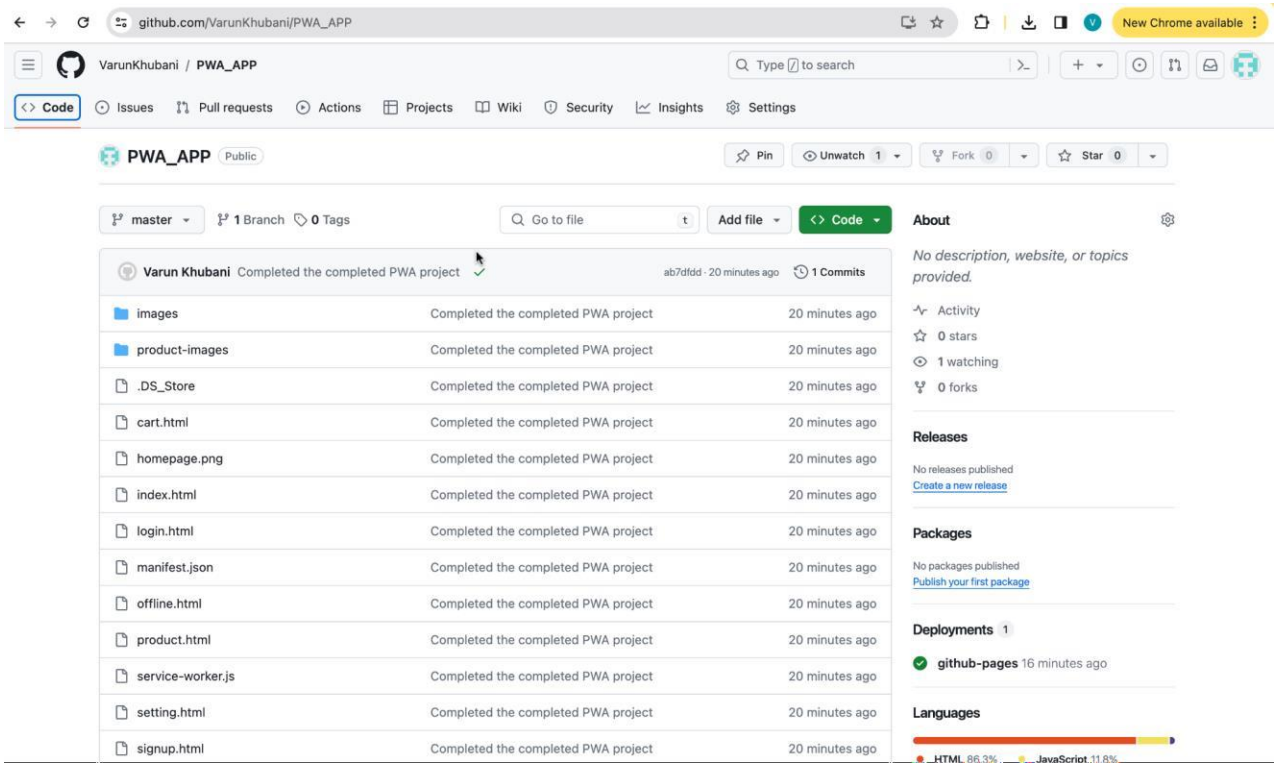
4. You’ll also have access to a blob store, which can store another 1 GB of data.
5. Support for HTTPS. A free certificate will be provisioned for your custom domain within 24 hours.

Cons

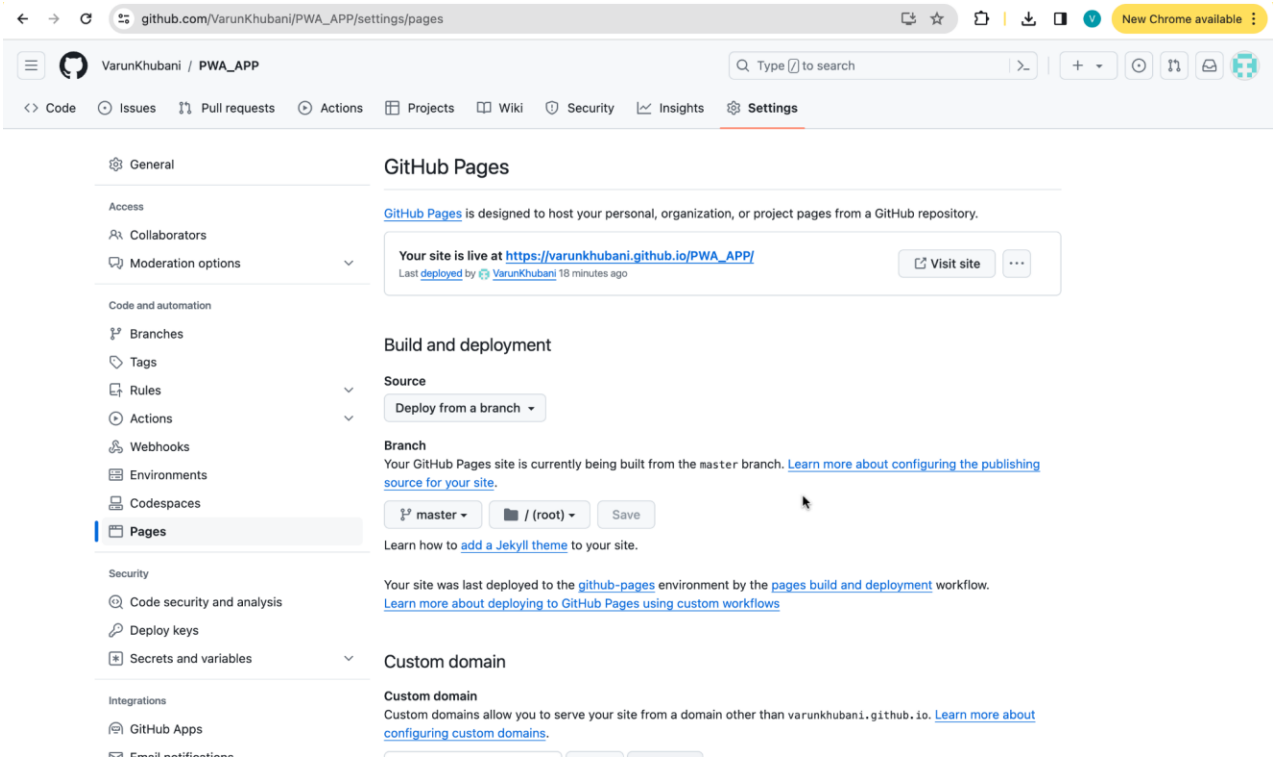
1. Only 10 GB of data transfer is allowed per month. But this is not really a big problem, if you use a CDN or AMP.
2. Command-line interface only.
3. No in-built support for any static site generator.

Link for GitHub: https://varunkhubani.github.io/PWA_APP/
GitHub ScreenShots

Step1: Make your GitHub Repository public and push your PWA into the repository

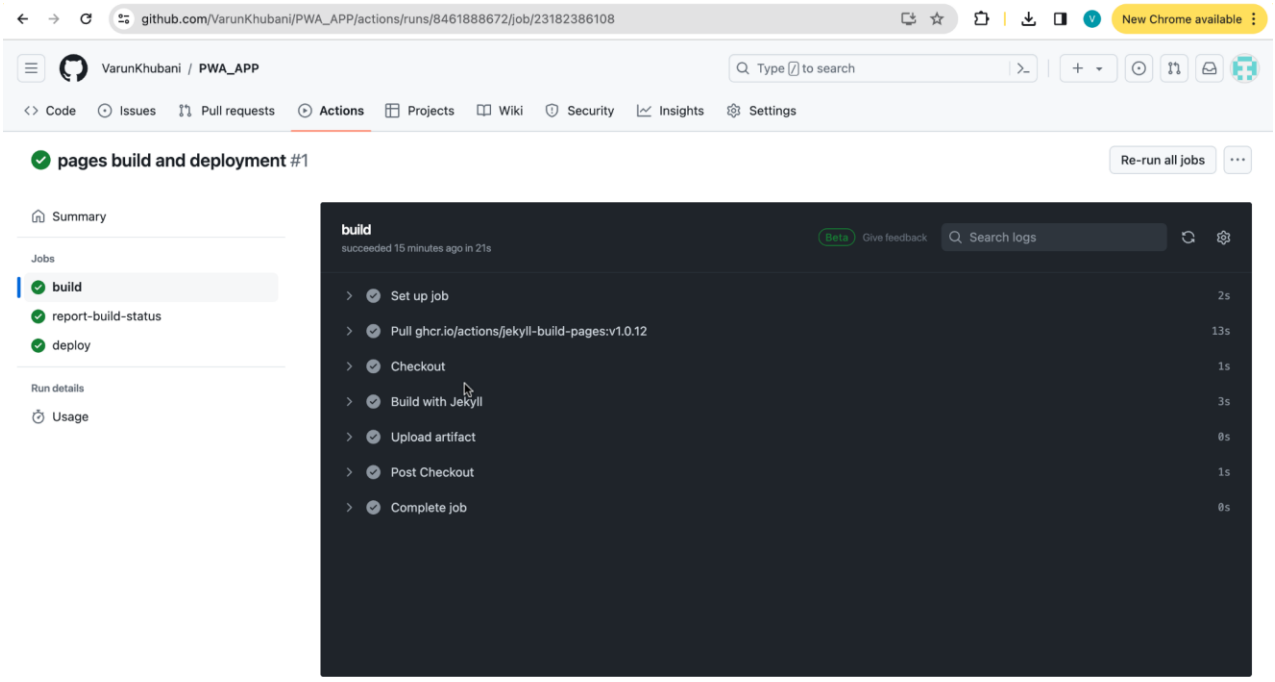


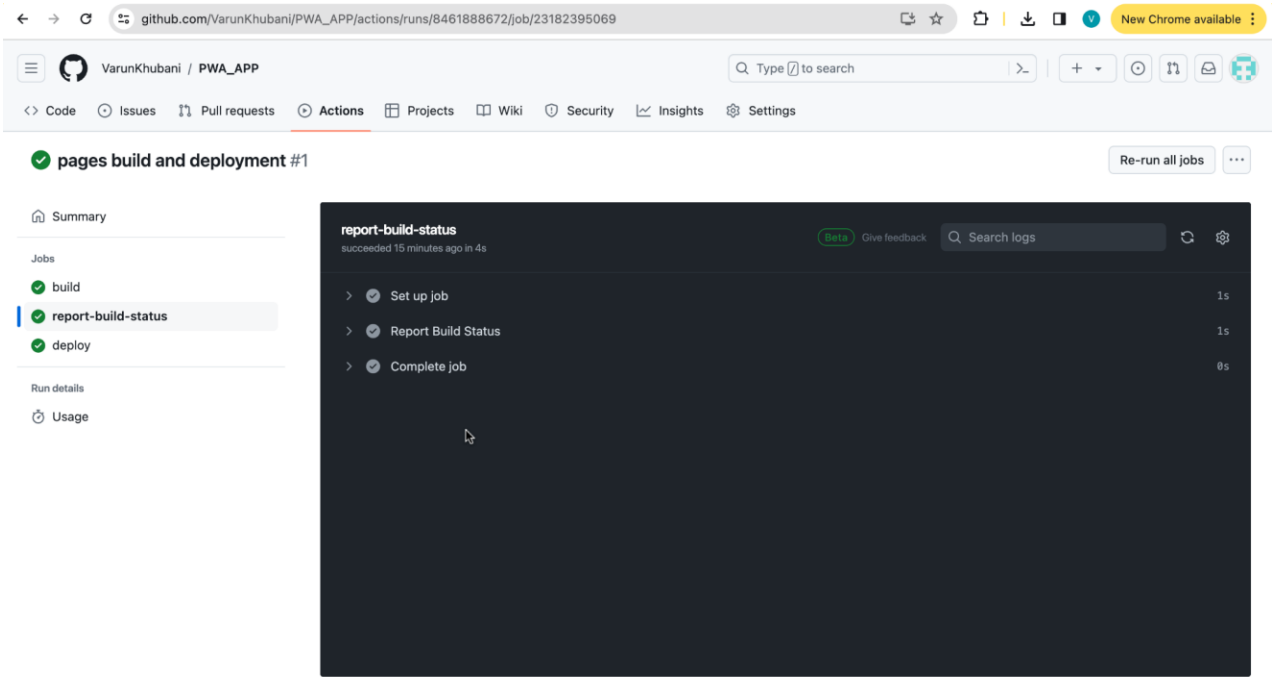
Step 2: Go to settings -> pages and choose your root directory and save it

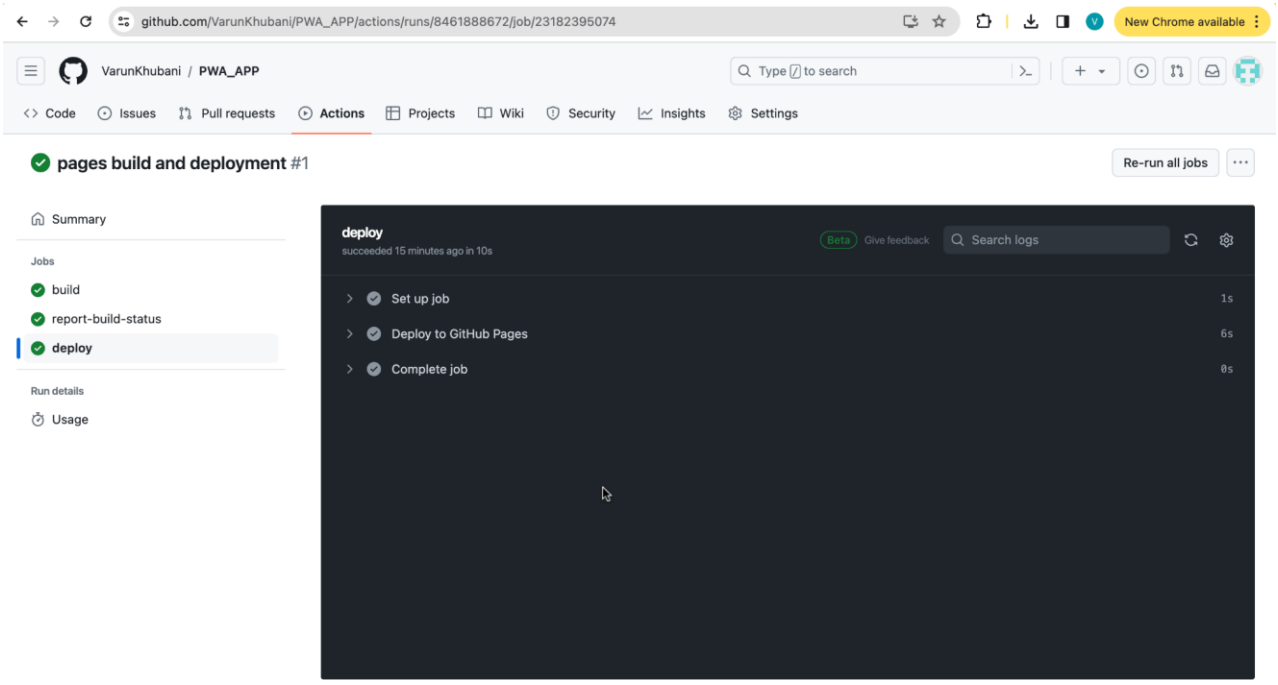


Step 3: Now go to your Code and you will see a small circle near your recent commit(Mine is finished deploying so i am getting a tick-mark sign)

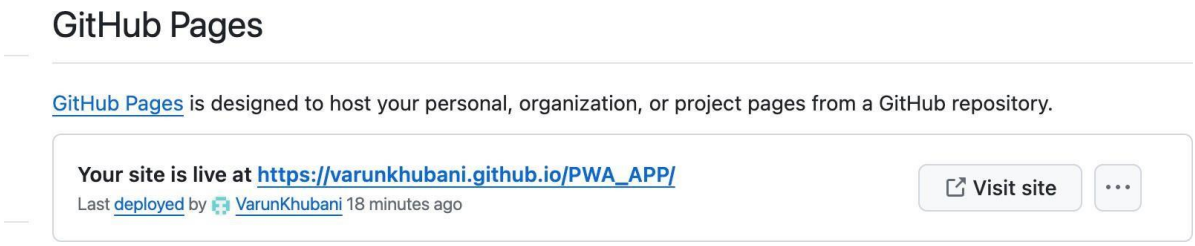
On clicking Logs of all the deployment is shown for convenience

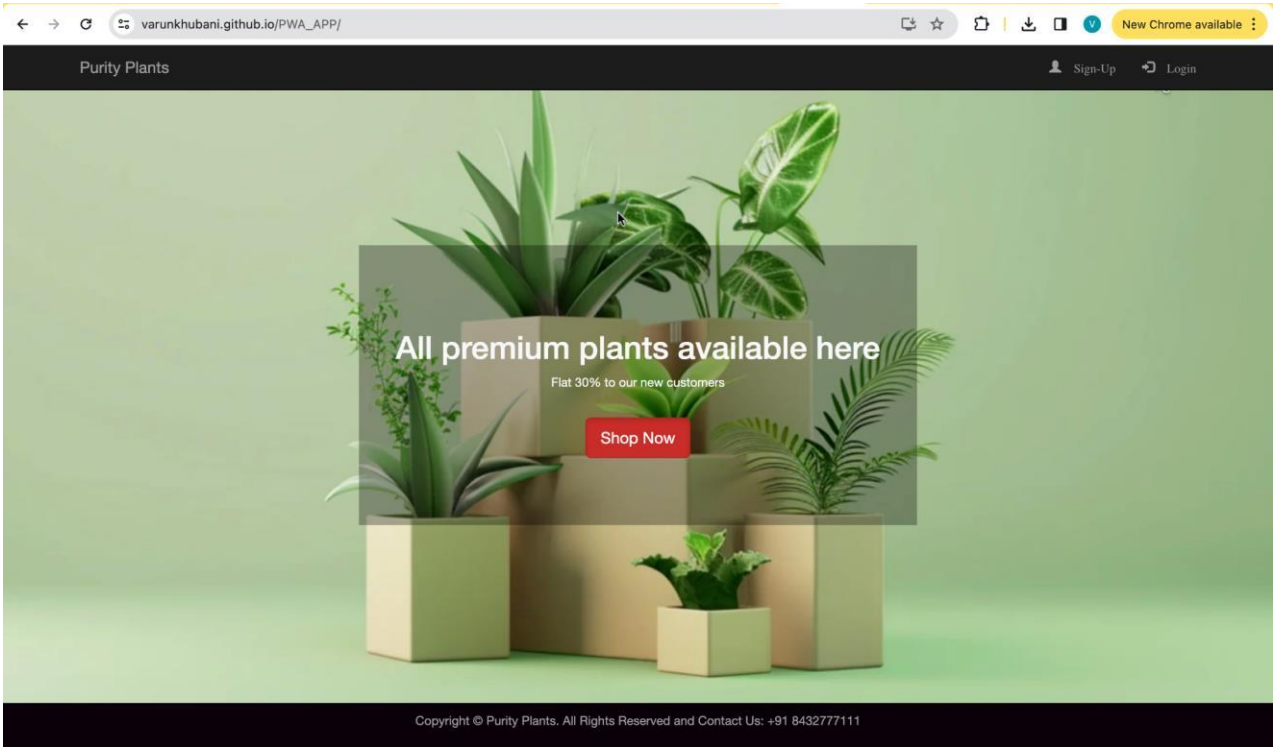

github.com/VarunKhubani/PWA_APP/actions/runs/8461888672/job/23182386108
New Chrome available
VarunKhubani / PWA_APP
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
pages build and deployment #1
Re-run all jobs
Summary
Jobs
build
report-build-status
deploy
Run details
Usage
build
succeeded 15 minutes ago in 21s
Beta
Give feedback
Search logs
Set up job 2s
Pull ghcr.io/actions/jekyll-build-pages:v1.0.12 13s
Checkout 1s
Build with Jekyll 3s
Upload artifact 0s
Post Checkout 1s
Complete job 0s


github.com/VarunKhubani/PWA_APP/actions/runs/8461888672/job/23182395069
New Chrome available
VarunKhubani / PWA_APP
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
pages build and deployment #1
Re-run all jobs
Summary
Jobs
build
report-build-status
deploy
Run details
Usage
report-build-status
succeeded 15 minutes ago in 4s
Beta
Give feedback
Search logs
Set up job 1s
Report Build Status 1s
Complete job 0s



Step4: Go to Settings -> Pages again , and you will see the pages has been deployed and a link is given





Conclusion: Hence we deployed our E-Commerce Progressive Web App Successfully on 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.	60
Name	Sujal Taktani
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	10

Name: Sujal Taktani

Division: D15A
Roll No:60
Batch: C

Experiment No 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

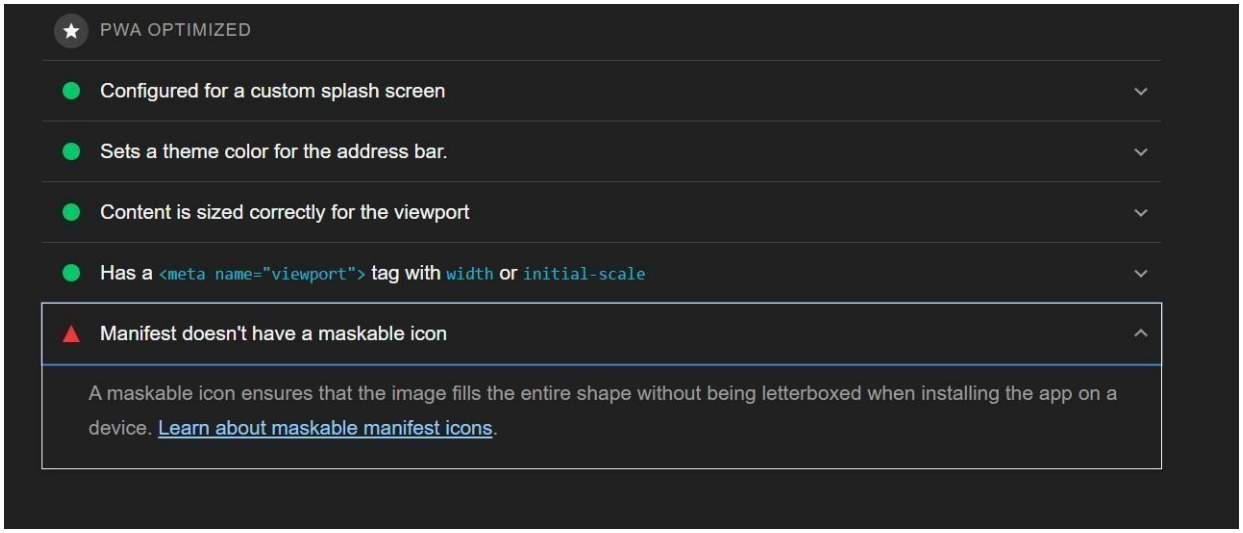
Avoiding the use of deprecated code elements like tags, directives, libraries, etc.

Password input with paste-into disabled

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

Output:

Before

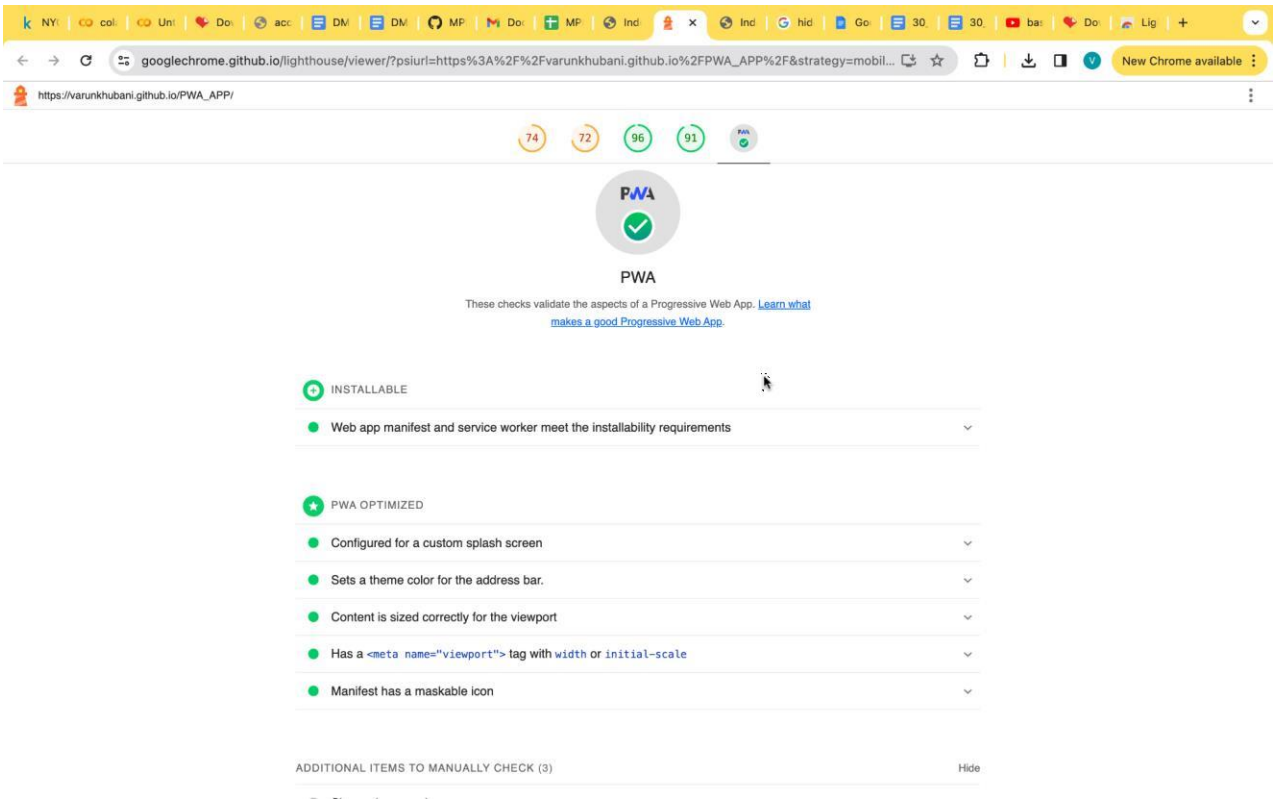


We encountered an issue here , it says “Manifest does not have a maskable icon”

Changes made to the code:

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

After:



Conclusion: Hence by making some changes to the code , we did google lighthouse analysisand our PWA is Fully Optimized and ready to go

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.	60
Name	Sujal Taktani
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:	5

Sujal. Takloni
D15A - 60

Assignment - 1 Flutter

Q1] Flutter Overview - Explain the key feature and advantage 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.

→ 1) Key Feature of Flutter:

ⓐ. Single codebase for multiple platform - allows to write code on iOS and android both.

ⓑ Hot Reload - To see the result in code.

ⓒ. Expressive UI - Flexibility to create.

ⓓ Integration with other Tools:

2) Advantage of Flutter:

ⓐ Faster Development. Uses single codebase

ⓑ. Consistent UI across platform widgets provide a consistent look.

ⓒ. Cost - Efficiency. Developing and maintaining single codebase.

3. Differ from Traditional Approach
- It uses a hierarchical structure for UI components.
- Flutter compiles to native ARM.
- Hot Reload allows to see changes.

Q.2] Widget Tree and Composition describe the concept of widget tree in Flutter. Explain how widget composition is used to build complex user interface.

→ Widget Tree:

- Widget Tree is a hierarchical structure of widgets.
- Every visual element, from simple component to complex layouts.
- Widget can be categorized into two types:
  a. Stateless widget.
  eg: Images, Text.
  b. Stateful widget.
  eg: Button, Forms.

Widget Composition

- Widget Composition in flutter involves combining multiple simple widgets
- This Composibility is a powerful concept that allow developer to build.

Commonly used widgets:

a. Container
b. Column and Row
c. Stack
d. List View
e. Grid View
f. App Bar
g. Text Field
h. Button widgets

3]. State management in Flutter.

→ • State management is crucial in Flutter application because it involves managing the data.
• Flutter is reactive meaning the UI rebuilds.

setState	Provider	Riverpod
① Built-in Flutter method	External package	External package (Riverpod)
② Local state within a widget.	Global state within widget state.	with additional feature.
③. Limited scalability.	For medium size apps	For large and complex
④ Lead to redundancy	Simplicity & readability	clean syntax.

Scenario when each is applicable

① Set State.
eg: Simple forms, UI components

② Provider
- for medium to large scale applications
eg: managing user authentication

③ Riverpod
eg: Complex applications with multiple features dynamic

6-4] Firebase Integration in Flutter

→ Integration:
1) Go to firebase console.
2) Add firebase SDK by including dependencies

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

3) Run flutter pub get
4) Initialize firebase by calling.

Benefits of using firebase

1) Real-time database + NoSql database

2) Authentication - Account & easy to implement solution

3) Cloud firestore :- Allow to store and sync data in realtime.

4) Hosting.
- Provide simple and efficient way to deploy.

Data Synchronization

1) Real-time database
- when data changes on one client, it triggers event.

2). Cloud firestore.
- It notifies client, when data changes allowing for real time update.

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

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.	60
Name	Sujal Taktani
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:	3

Sujal Takalani
D15A-60

Assignment - 2

1] Define progressive web app (PWA)

=> (i) Progressive web app (PWA)
- A type of web application that combines of web application

(ii) Significance in modern web development
- PWA an platform - agnostic meaning thing work seamlessly across different devices and operating system

(iii) Key characteristics
Traditional mobile apps
- offline capabilities
- Responsive design
- App shell architecture
- Push notifications
- Installation

2]

=> (i) Responsive web design ensures that a website adapts

(ii) Importance in Content of PWA
- Provides a consistent experience across device
- Ensure that PWA look and function

(ii)

Responsive	Fluid	adaptive web design
- Involves creating websites that automatically adjust and respond of flexible grids	- Fluid web design involve creating web that use proportional units of fixed units	- Adaptive web design. Involve creating multiple layout or templates for different devices
- Use a single codebase that dynamically Changes the layout & content based on screen sizes	- It focus on creating design that scale smoothly across different screen sizes	- Detects the user device or screen size and serves a predefined layout.
Eg- A responsive website rearranged image & adjust text sizes to provide an optimal viewing	- In a fluid design, elements like text, block and images expand or Contract smoothly	- Have separate layout. frontend. for desktop in landscape orientation,

3]

=> The lifecycle of service workers involves 3 main phases.

① Registration
② Installation
③ Activation

① Registration -
- Uses 'navigator.serviceWorker.register()' method to register the service worker.

- Begins when a JavaScript file containing the service worker code is registered in the web page

② Installation :
- During installation, the service worker caches essential resources like HTML, CSS and JS files.
- In the install event handler, developers can perform tasks

③ Activation -
- After installation is completed, the service worker enters the activation phase. During activation, the new service workers

4]

→ (i) Indexed DB is a low-level API that stores large amount of structured data on a user browser.

(ii) It uses index to enable high performance searches of this data.

(iii) Indexed DB is asynchronous, the user interface from rendering. while the data loads with rich query.

(iv) An event listener to handle message. The maximum limit is based on the browser and the disk space.

(v). For example, Chrome, and Chromium-based browser allow up to 80% disk space. Allows to categorize data using object store. Allows to store large amount of data.