Vivekanand Education Society's Institute of Technology

An Autonomous Institute Affiliated to University of Mumbai

Hashu Advani Memorial Complex, Collector Colony, Chembur East, Mumbai - 400074.

Department of Information Technology

CERTIFICATE

This is to certify that **Yukta Bhatia** 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

Project Title: Quick Chats/ PWA : Purity Plants **Roll No. 7**

Name of the Course : MAD & PWA Lab **Course Code :** ITL604

Year/Sem/Class : D15A **A.Y.:** 23-24

Faculty Incharge : Mrs. Kajal Joseph.

Lab Teachers : Mrs. Kajal Jewani.

Email : kajal.jewani@ves.ac.in

Programme Outcomes: The graduate will be able to:

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

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

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

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

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

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

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

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

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

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

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

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

Program specific Outcomes

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

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

Project Title: Quick Chats/ PWA : Purity Plants **Roll No. 7**

Lab Objectives:

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

Lab Outcomes:

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

Index

Sr. No	Experiment Title	LO	DOP	DOS	Grade
1.	To install and configure the Flutter Environment	LO1	16/1	23/1	15
2.	To design Flutter UI by including common widgets.	LO2	23/1	30/1	15
3.	To include icons, images, fonts in Flutter app	LO2	30/1	6/2	15
4.	To create an interactive Form using form widget	LO2	6/2	13/2	15
5.	To apply navigation, routing and gestures in Flutter App	LO2	13/2	20/2	15
6.	To Connect Flutter UI with fireBase database	LO3	20/2	5/3	15
7.	To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”.	LO4	5/3	12/3	15
8.	To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA	LO5	12/3	19/3	15
9.	To implement Service worker events like fetch, sync and push for E-commerce PWA	LO5	19/3	26/3	15
10.	To study and implement deployment of Ecommerce PWA to GitHub Pages.	LO5	26/3	2/4	15
11.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.	LO6	5/3	12/3	15
12.	Assignment-1	LO1,LO2 ,LO3	2/2	5/2	5
13.	Assignment-2	LO4,LO5 ,LO6	19/3	21/3	4

MAD & PWA Lab
Journal

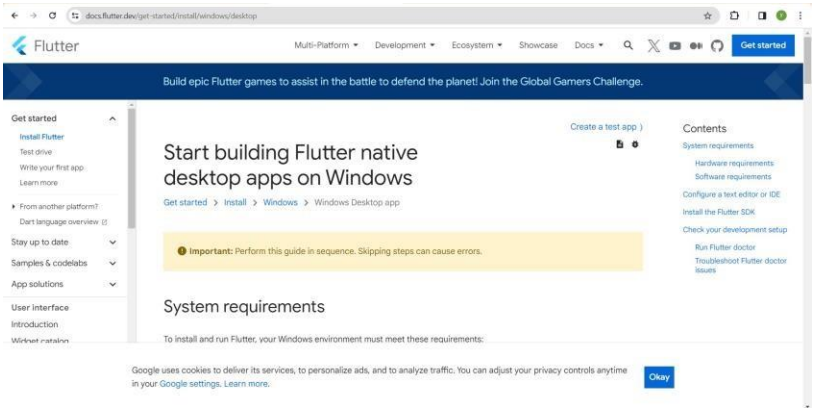
Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	07
Name	Yukta Bhatia
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	15

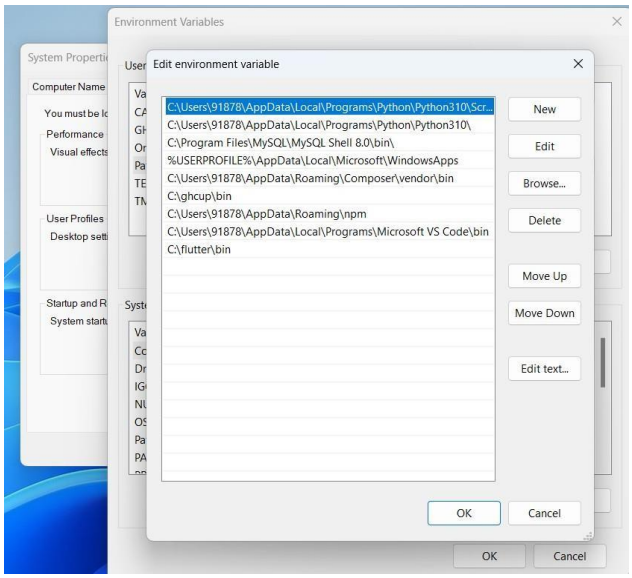
Project Title: Quick Chats/ PWA : Purity Plants **Roll No. 7**

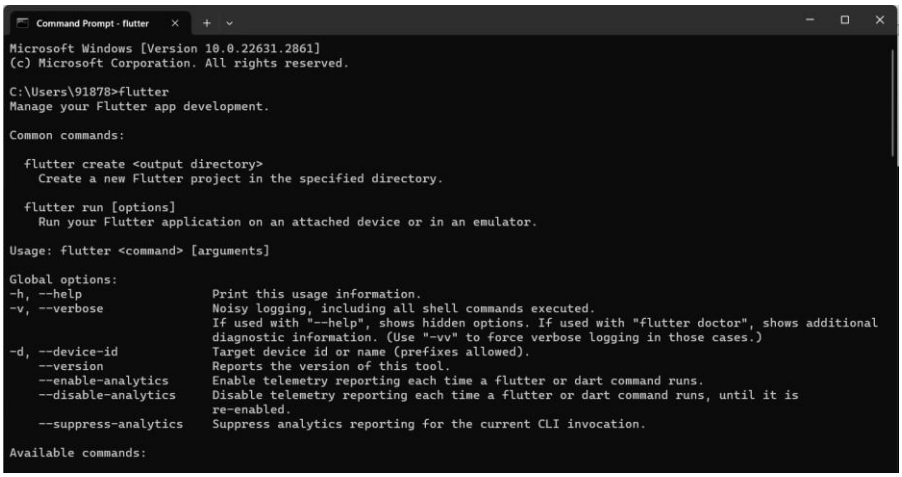
Experiment-1

Aim: Create a ‘Hello World’ using

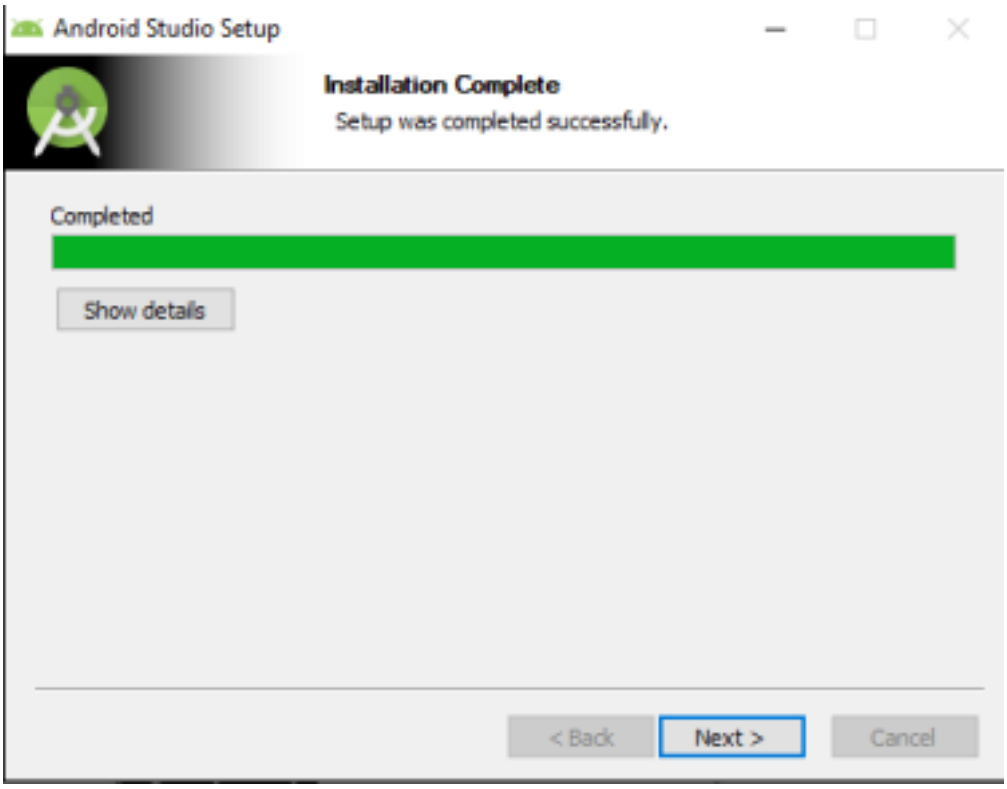
FlutterInstalled Flutter SDK

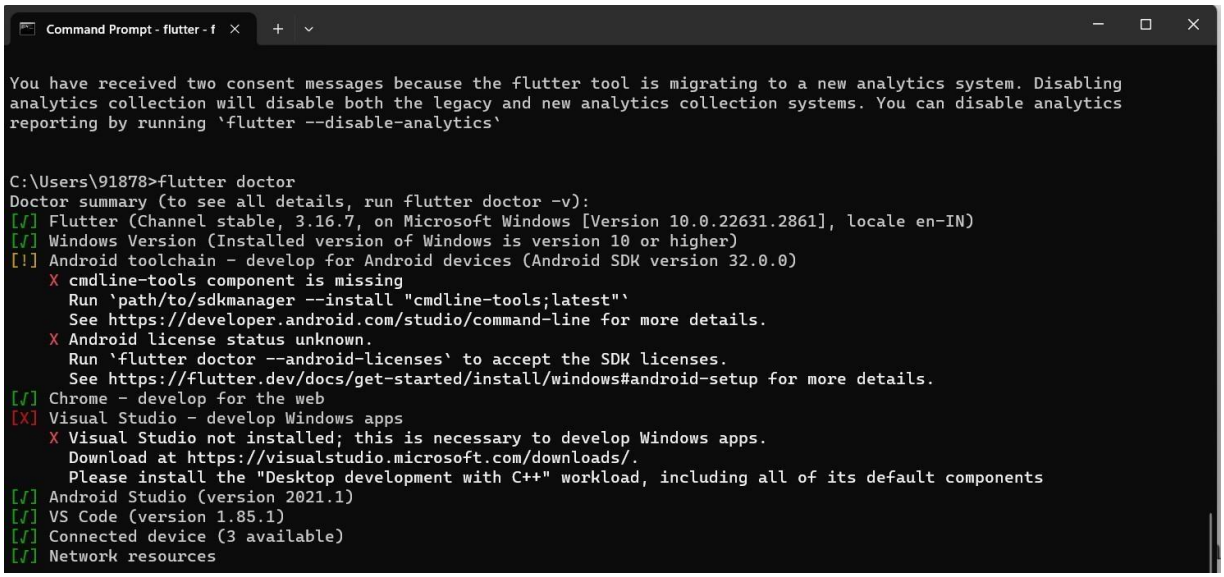






Installed Android SDK.


Android Studio Setup
Installation Complete
Setup was completed successfully.
Completed
Show details
< Back
Next >
Cancel

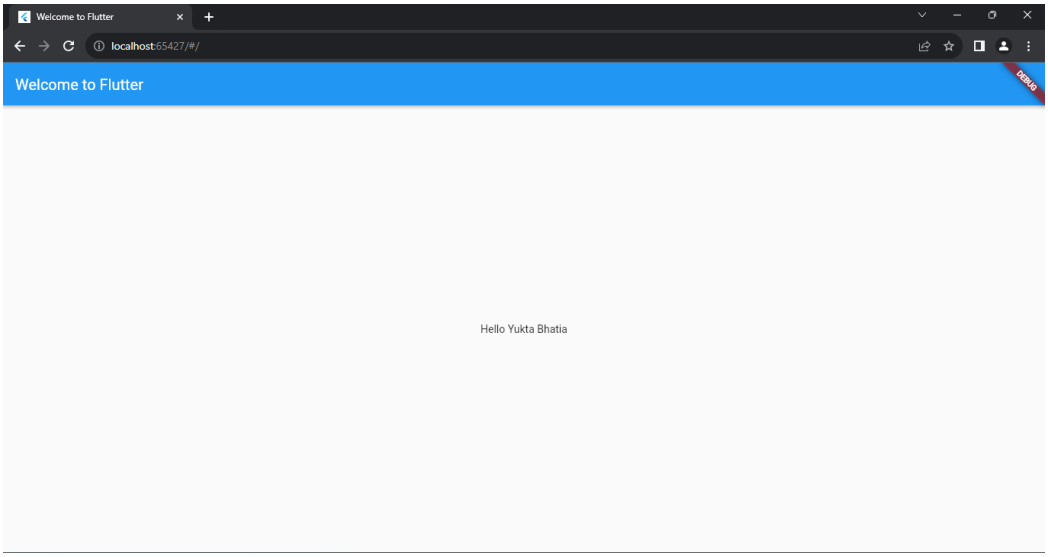

Command Prompt - flutter - f
You have received two consent messages because the flutter tool is migrating to a new analytics system. Disabling analytics collection will disable both the legacy and new analytics collection systems. You can disable analytics reporting by running `flutter --disable-analytics`
C:\Users\91878>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, 3.16.7, on Microsoft Windows [Version 10.0.22631.2861], locale en-IN)
[√] Windows Version (Installed version of Windows is version 10 or higher)
[!] Android toolchain - develop for Android devices (Android SDK version 32.0.0)
X cmdline-tools component is missing
Run `path/to/sdkmanager --install "cmdline-tools;latest"`
See https://developer.android.com/studio/command-line for more details.
X Android license status unknown.
Run `flutter doctor --android-licenses` to accept the SDK licenses.
See https://flutter.dev/docs/get-started/install/windows#android-setup for more details.
[√] Chrome - develop for the web
[X] Visual Studio - develop Windows apps
X Visual Studio not installed; this is necessary to develop Windows apps.
Download at https://visualstudio.microsoft.com/downloads/.
Please install the "Desktop development with C++" workload, including all of its default components
[√] Android Studio (version 2021.1)
[√] VS Code (version 1.85.1)
[√] Connected device (3 available)
[√] Network resources

Project Title: Quick Chats/ PWA : Purity Plants **Roll No. 7**

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 Yukta Bhatia'),

        ),

      ),
    );
  }
}
```

Output:



MAD & PWA Lab
Journal

Experiment No.	02
Experiment Title.	To design Flutter UI by including common widgets.
Roll No.	07
Name	Yukta Bhatia
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

Name: Yukta Bhatia
RollNo: 07
Class: D15A

Experiment-2

Aim: To design Flutter UI by including common widgets.

Theory:

Flutter widgets are the building blocks of a Flutter app's user interface. They are the basic visual elements developers use to create user interfaces and define the app's functionality.

A Flutter widget can be defined as a self-contained, reusable piece of code that describes how part of the user interface should be displayed. Widgets can be thought of as Lego blocks, which can be combined and arranged in many different ways to create complex user interfaces.

Code:

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

class _MessageCardState extends State<MessageCard> {
  @override
  Widget build(BuildContext context) {
    bool isMe = APIs.user.uid == widget.message.fromId; return
    InkWell(
        onLongPress: () {
          _showBottomSheet(isMe);
        },
   }

  void _showBottomSheet(bool isMe) { showModalBottomSheet(
        context: context,
        shape: const RoundedRectangleBorder( borderRadius:
            BorderRadius.only(
                topLeft: Radius.circular(20), topRight: Radius.circular(20))), builder:
        (_) {
          return ListView(
```

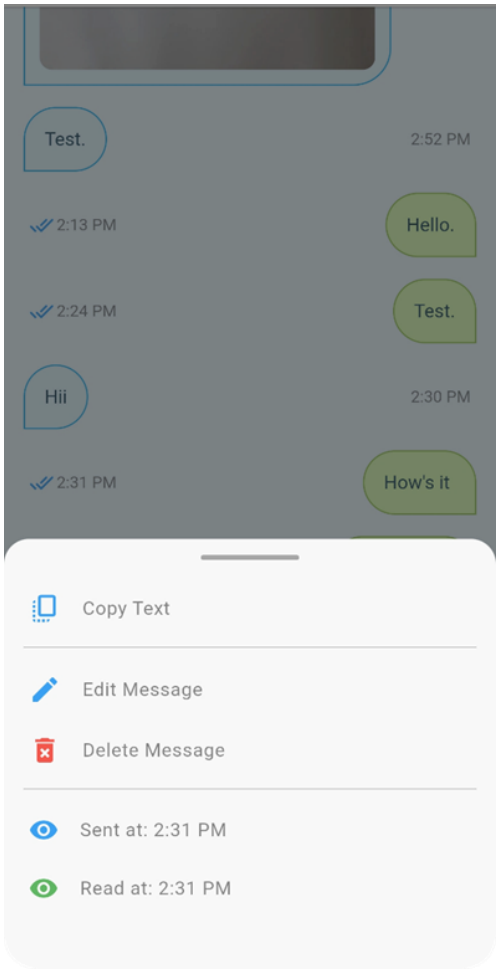
```
shrinkWrap: true,
children: [
  //black divider
  Container(
  height: 4,
    margin: EdgeInsets.symmetric(
        vertical: mq.height * .015, horizontal: mq.width * .4), decoration:
    BoxDecoration(
        color: Colors.grey, borderRadius: BorderRadius.circular(8)),
  ),

  widget.message.type == Type.text
      ?
      //copy option
      _OptionItem(
          icon: const Icon(Icons.copy_all_rounded, color:
              Colors.blue, size: 26),
          name: 'Copy Text', onTap:
          () async {
            await Clipboard.setData(
                  ClipboardData(text: widget.message.msg))
                .then((value) {
              //for hiding bottom sheet
              Navigator.pop(context);

              Dialogs.showSnackbar(context, 'Text Copied!');
            });
          })
      :
      //save option
      _OptionItem(
          icon: const Icon(Icons.download_rounded, color:
              Colors.blue, size: 26),
          name: 'Save Image',
          onTap: () async {
          try {
            log('Image Url: ${widget.message.msg}');
            await GallerySaver.saveImage(widget.message.msg,
                  albumName: 'We Chat')
                .then((success) {
              //for hiding bottom sheet
              Navigator.pop(context);
              if (success != null && success) {
                Dialogs.showSnackbar(
```

```
                    context, 'Image Successfully Saved!');
                }
              });
            } catch (e) {
              log('ErrorWhileSavingImg: $e');
            }
          }),

//separator or divider if
(isMe)
  Divider(
    color: Colors.black54,
    endIndent: mq.width * .04,
    indent: mq.width * .04,
  ),

//edit option
if (widget.message.type == Type.text && isMe)
  _OptionItem(
      icon: const Icon(Icons.edit, color: Colors.blue, size: 26), name:
      'Edit Message',
      onTap: () {
        //for hiding bottom sheet Navigator.pop(context);

        _showMessageUpdateDialog();
      }),

//delete option
if (isMe)
  _OptionItem(
      icon: const Icon(Icons.delete_forever, color:
          Colors.red, size: 26),
      name: 'Delete Message',
      onTap: () async {
        await APIs.deleteMessage(widget.message).then((value) {
          //for hiding bottom sheet Navigator.pop(context);
        });
      }),

//separator or divider
Divider(
  color: Colors.black54,
```

```
            endIndent: mq.width * .04,
            indent: mq.width * .04,
          ),

          //sent time
          _OptionItem(
              icon: const Icon(Icons.remove_red_eye, color: Colors.blue), name:
                  'Sent At: ${MyDateUtil.getMessageTime(context: context, time: widget.message.sent)}',
              onTap: () {}),

          //read time
          _OptionItem(
              icon: const Icon(Icons.remove_red_eye, color: Colors.green), name:
              widget.message.read.isEmpty
                  ? 'Read At: Not seen yet'
                  : 'Read At: ${MyDateUtil.getMessageTime(context: context, time: widget.message.read)}',
              onTap: () {}),
        ],
      );
    });
  }

class _OptionItem extends StatelessWidget { final
  Icon icon;
  final String name;
  final VoidCallback onTap;

  const _OptionItem(
      {required this.icon, required this.name, required this.onTap});

  @override
  Widget build(BuildContext context) {
    return InkWell(
        onTap: () => onTap(), child:
        Padding(
          padding: EdgeInsets.only(
              left: mq.width * .05,
              top: mq.height * .015, bottom:
              mq.height * .015),
          child: Row(children: [
            icon,
```

```
            Flexible(
                child: Text('     $name',
                    style: const TextStyle(
                        fontSize: 15,
                        color: Colors.black54,
                        letterSpacing: 0.5)))
          ]),
        ));
  }
}
```

Output:



MAD & PWA Lab
Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	07
Name	Yukta Bhatia
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

Name: Yukta Bhatia
RollNo: 07
Class: D15A

Experiment-3

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

Code:

main.dart

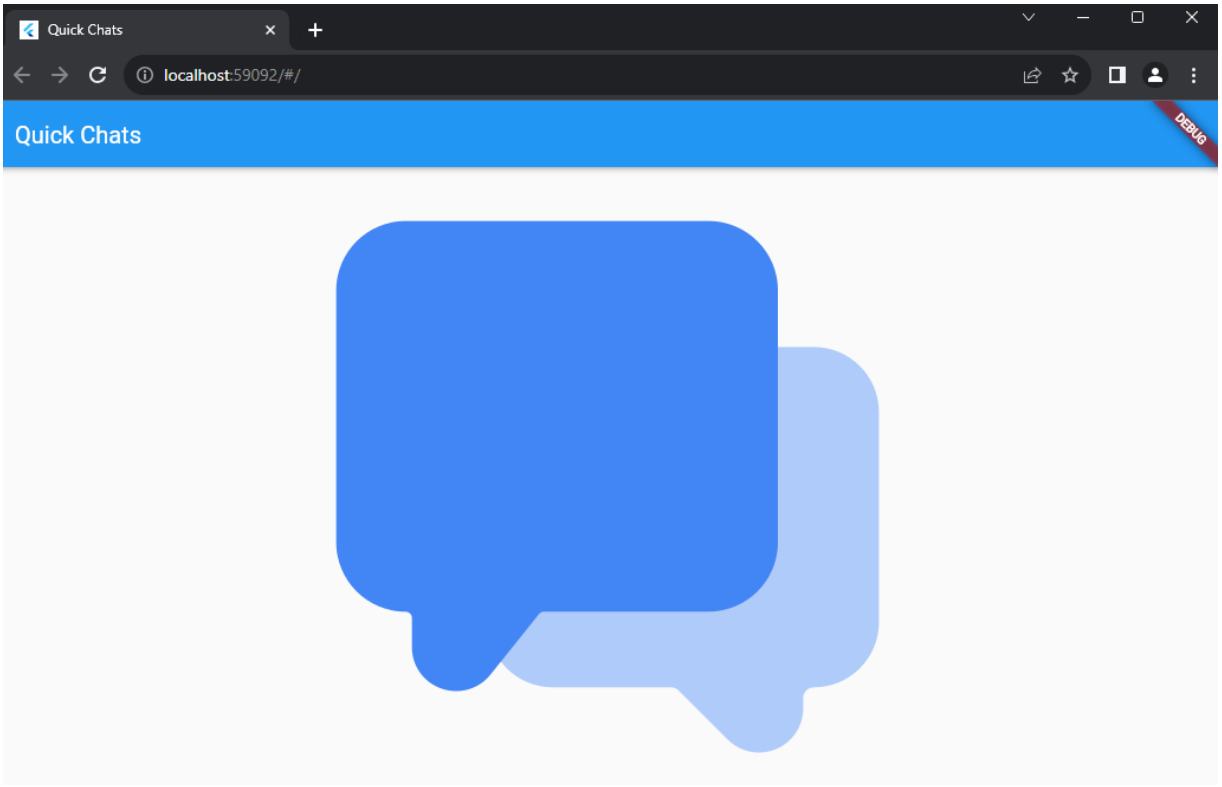
```
import 'package:flutter/material.dart';void
main() {
 runApp(const MyApp());
}
class MyApp extends StatelessWidget { const
 MyApp({Key? key}) : super(key: key);
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     title: 'Quick Chats',
     home: Scaffold(
     appBar: AppBar(
         title: const Text('Quick Chats'),
       ),
       body: Center(
         child: Image.asset('assets/comp.png'),
       ),
     ),
   );
 }
}
```

pubspec.lock

```
# To add assets to your application, add an assets section, like this:
assets:
   - assets/comp.png
#  - images/a_dot_ham.jpeg
```

Output:


assets
comp.png
build


Quick Chats
localhost:59092/#/
Quick Chats
DEBUG

MAD & PWA Lab
Journal

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget
Roll No.	07
Name	Yukta Bhatia
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

Name: Yukta Bhatia
RollNo: 07
Class: D15A

Experiment-4

Aim: To create an interactive Form using form widget

Theory:

Flutter Form Widget

The Form widget in Flutter provides a way to create and manage a group of related form elements. It helps with form validation, submission, and data storage. When using the Form widget, you typically also use TextFormField for text input and other form-related widgets.

TextFormFields

Use TextFormField widgets for text input fields within the form. These widgets handle user input, validation, and saving data.

Form Submission

Typically, a button triggers form submission. The onPressed callback of the button calls a function, which in turn validates and saves the form data.

Code:

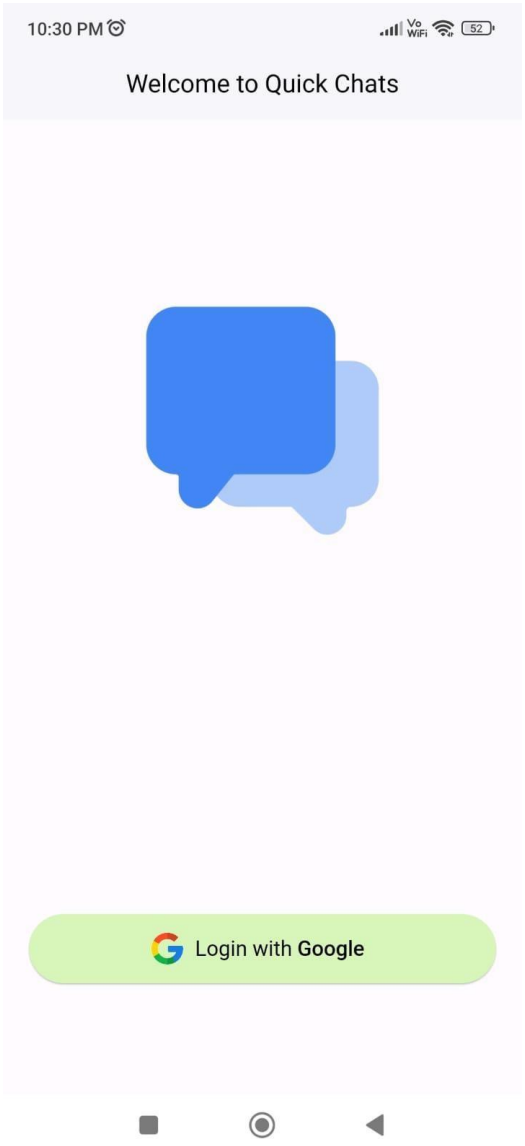
```
import 'dart:developer';
import 'dart:io';

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

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

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

class _LoginScreenState extends State<LoginScreen> {
  bool _isAnimate = false;

  @override
  void initState() {
    super.initState();

    Future.delayed(const Duration(milliseconds: 500), () {
```

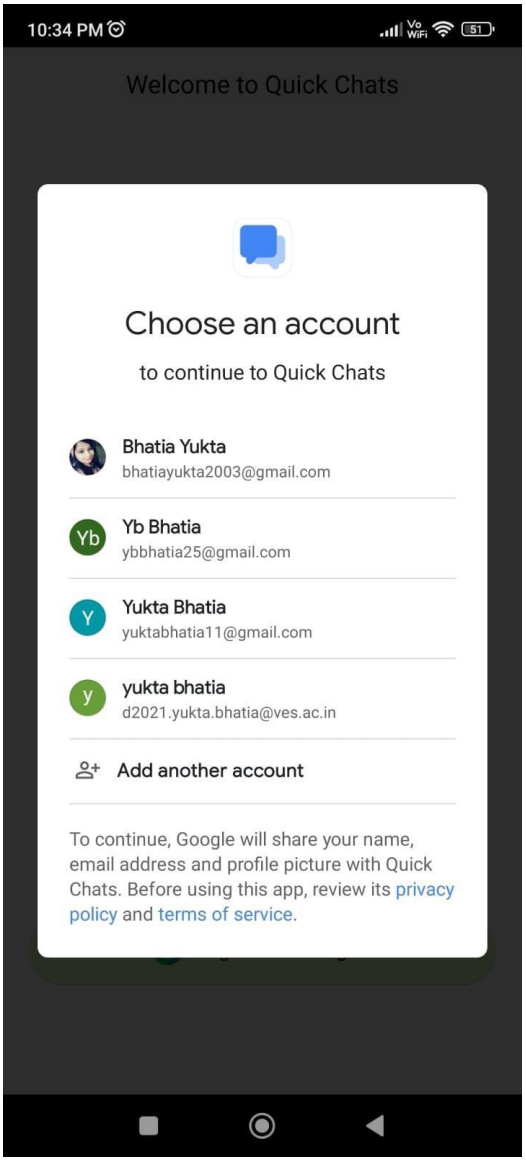
```
    setState(() => _isAnimate = true);
  });
 }

 _handleGoogleBtnClick() {
  //for showing progress bar
  Dialogs.showProgressBar(context);

  _signInWithGoogle().then((user) async {
   //for hiding progress bar
   Navigator.pop(context);

   if (user != null) {
    log('\nUser: ${user.user}');
    log('\nUserAdditionalInfo: ${user.additionalUserInfo}');

    if ((await APIs.userExists())) {
     Navigator.pushReplacement(
         context, MaterialPageRoute(builder: (_) => const HomeScreen()));
    } else {
     await APIs.createUser().then((value) {
      Navigator.pushReplacement(
          context, MaterialPageRoute(builder: (_) => const HomeScreen()));
     });
    }
   }
  });
 }

 Future<UserCredential?> _signInWithGoogle() async {
  try {
   await InternetAddress.lookup('google.com');
   final GoogleSignInAccount? googleUser = await GoogleSignIn().signIn();

   // Obtain the auth details from the request
   final GoogleSignInAuthentication? googleAuth =
       await googleUser?.authentication;

   // Create a new credential
   final credential = GoogleAuthProvider.credential(
    accessToken: googleAuth?.accessToken,
    idToken: googleAuth?.idToken,
   );

   // Once signed in, return the UserCredential
   return await APIs.auth.signInWithCredential(credential);
  } catch (e) {
   log('\n_signInWithGoogle: $e');
   Dialogs.showSnackbar(context, 'Something Went Wrong (Check Internet!)');
   return null;
  }
 }
```

```
@override
Widget build(BuildContext context) {
  //initializing media query (for getting device screen size)
  mq = MediaQuery.of(context).size;

  return Scaffold(
    //app  bar
    appBar: AppBar(
      automaticallyImplyLeading: false,
      title: const Text('Welcome to Quick Chats'),
    ),

    //body
    body: Stack(children: [
      //app logo
      AnimatedPositioned(
          top: mq.height * .15,
          right: _isAnimate ? mq.width * .25 : -mq.width * .5,
          width: mq.width * .5,
          duration: const Duration(seconds: 1),
          child: Image.asset('images/icon.png')),

      //google login button
      Positioned(
          bottom: mq.height * .15,
          left: mq.width * .05,
          width: mq.width * .9,
          height: mq.height * .06,
          child: ElevatedButton.icon(
              style: ElevatedButton.styleFrom(
                  backgroundColor: const Color.fromARGB(255, 223, 255, 187),
                  shape: const StadiumBorder(),
                  elevation: 1),
              onPressed: () {
                _handleGoogleBtnClick();
              },

              //google icon
              icon: Image.asset('images/google.png', height: mq.height * .03),

              //login with google label
              label: RichText(
                text: const TextSpan(
                    style: TextStyle(color: Colors.black, fontSize: 16),
                    children: [
                      TextSpan(text: 'Login with '),
                      TextSpan(
                          text: 'Google',
                          style: TextStyle(fontWeight: FontWeight.w500)),
                    ]),
```

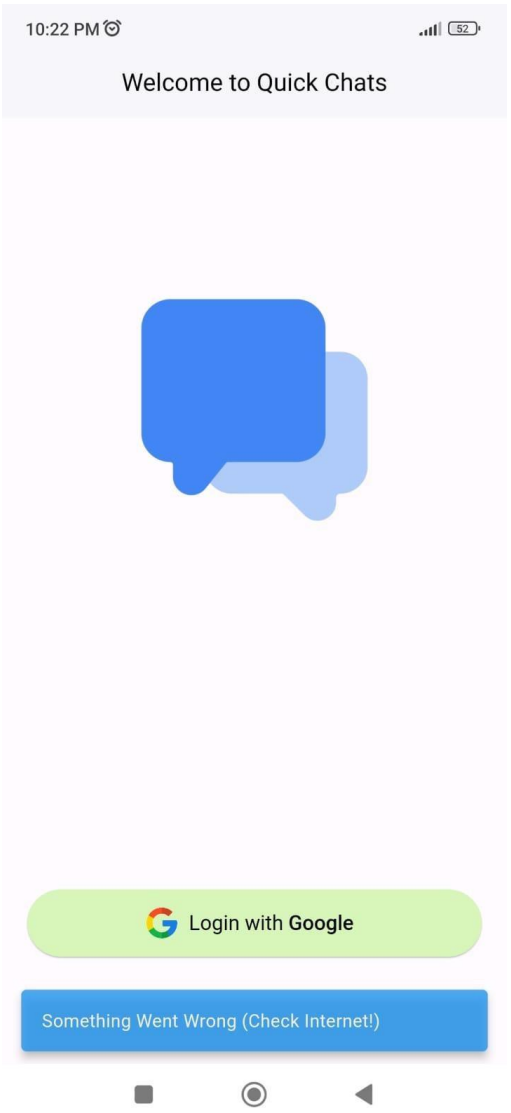
```
            ))),
      ]),
    );
  }
}
```

Output:


10:30 PM
Welcome to Quick Chats
Login with Google


10:34 PM
Welcome to Quick Chats
Choose an account
to continue to Quick Chats
Bhatia Yukta
bhatiayukta2003@gmail.com
Yb Bhatia
ybbhatia25@gmail.com
Yukta Bhatia
yuktabhatia11@gmail.com
yukta bhatia
d2021.yukta.bhatia@ves.ac.in
Add another account
To continue, Google will share your name, email address and profile picture with Quick Chats. Before using this app, review its privacy policy and terms of service.

Project Title: Quick Chats/ PWA : Purity Plants **Roll No. 7**



MAD & PWA Lab
Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	07
Name	Yukta Bhatia
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

Name: Yukta Bhatia
RollNo: 07
Class: D15A

Experiment-5

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

Theory:

Flutter Navigation and Routing

Navigation and routing are some of the core concepts of all mobile application, which allows the user to move between different pages. We know that every mobile application contains several screens for displaying different types of information. For example, an app can have a screen that contains various products. When the user taps on that product, immediately it will display detailed information about that product.

In Flutter, the screens and pages are known as routes, and these routes are just a widget. In Android, a route is similar to an Activity, whereas, in iOS, it is equivalent to a ViewController.

In any mobile app, navigating to different pages defines the workflow of the application, and the way to handle the navigation is known as routing. Flutter provides a basic routing class MaterialPageRoute and two methods Navigator.push() and Navigator.pop() that shows how to navigate between two routes. The following steps are required to start navigation in your application.

Code:

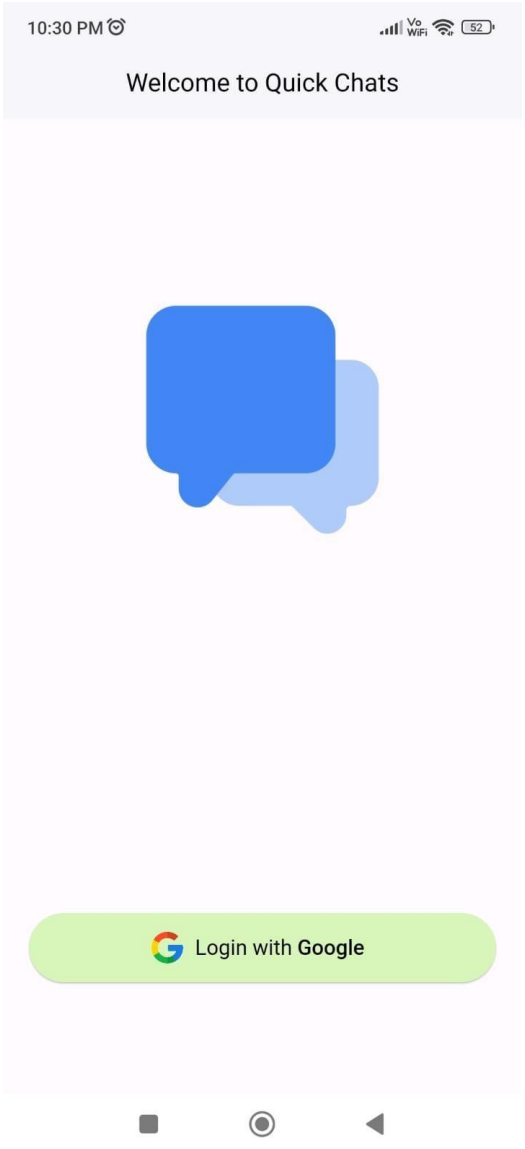
```
import 'dart:developer';
import 'dart:io';

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

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

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

class _LoginScreenState extends State<LoginScreen> {
  bool _isAnimate = false;

  @override
  void initState() {
    super.initState();
```

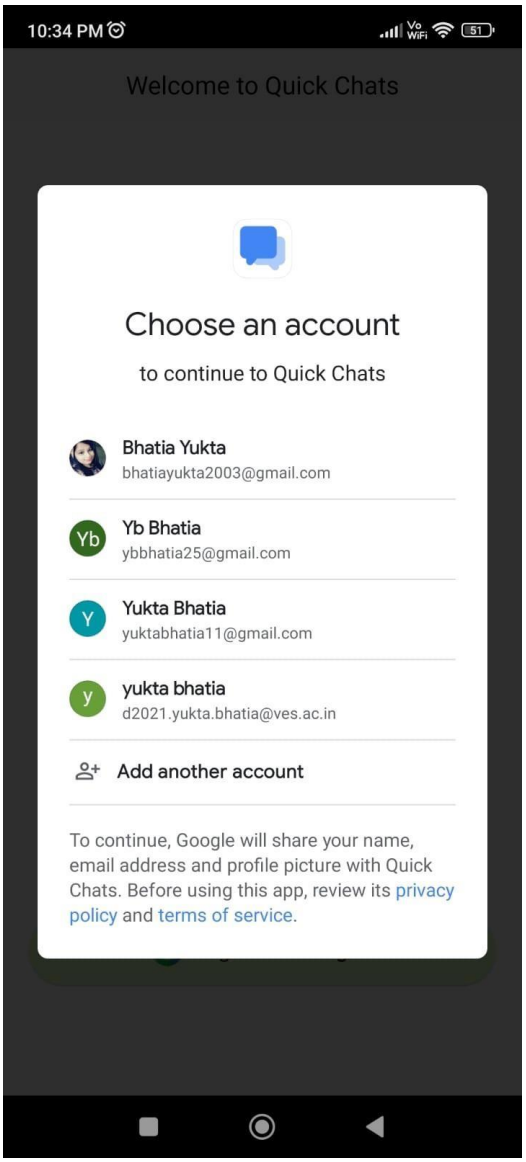
```
  Future.delayed(const Duration(milliseconds: 500), () {
    setState(() => _isAnimate = true);
  });
}

_handleGoogleBtnClick() {
  //for showing progress bar
  Dialogs.showProgressBar(context);

  _signInWithGoogle().then((user) async {
    //for hiding progress bar
    Navigator.pop(context);

    if (user != null) {
      log('\nUser: ${user.user}');
      log('\nUserAdditionalInfo: ${user.additionalUserInfo}');

      if ((await APIs.userExists())) {
        Navigator.pushReplacement(
            context, MaterialPageRoute(builder: (_) => const HomeScreen()));
      } else {
        await APIs.createUser().then((value) {
          Navigator.pushReplacement(
              context, MaterialPageRoute(builder: (_) => const HomeScreen()));
        });
      }
    }
  });
}

Future<UserCredential?> _signInWithGoogle() async {
  try {
    await InternetAddress.lookup('google.com');
    final GoogleSignInAccount? googleUser = await GoogleSignIn().signIn();

    // Obtain the auth details from the request
    final GoogleSignInAuthentication? googleAuth =
        await googleUser?.authentication;

    // Create a new credential
    final credential = GoogleAuthProvider.credential(
      accessToken: googleAuth?.accessToken,
      idToken: googleAuth?.idToken,
    );

    // Once signed in, return the UserCredential
    return await APIs.auth.signInWithCredential(credential);
  } catch (e) {
    log('\n_signInWithGoogle: $e');
    Dialogs.showSnackbar(context, 'Something Went Wrong (Check Internet!)');
    return null;
```

```
  }
 }


 @override
 Widget build(BuildContext context) {
  //initializing media query (for getting device screen size)
  mq = MediaQuery.of(context).size;

  return Scaffold(
   //app  bar
   appBar: AppBar(
    automaticallyImplyLeading: false,
    title: const Text('Welcome to Quick Chats'),
   ),

   //body
   body: Stack(children: [
    //app logo
    AnimatedPositioned(
       top: mq.height * .15,
       right: _isAnimate ? mq.width * .25 : -mq.width * .5,
       width: mq.width * .5,
       duration: const Duration(seconds: 1),
       child: Image.asset('images/icon.png')),

    //google login button
    Positioned(
       bottom: mq.height * .15,
       left: mq.width * .05,
       width: mq.width * .9,
       height: mq.height * .06,
       child: ElevatedButton.icon(
          style: ElevatedButton.styleFrom(
             backgroundColor: const Color.fromARGB(255, 223, 255, 187),
             shape: const StadiumBorder(),
             elevation: 1),
          onPressed: () {
           _handleGoogleBtnClick();
          },

          //google icon
          icon: Image.asset('images/google.png', height: mq.height * .03),

          //login with google label
          label: RichText(
           text: const TextSpan(
              style: TextStyle(color: Colors.black, fontSize: 16),
              children: [
               TextSpan(text: 'Login with '),
               TextSpan(
                  text: 'Google',
```

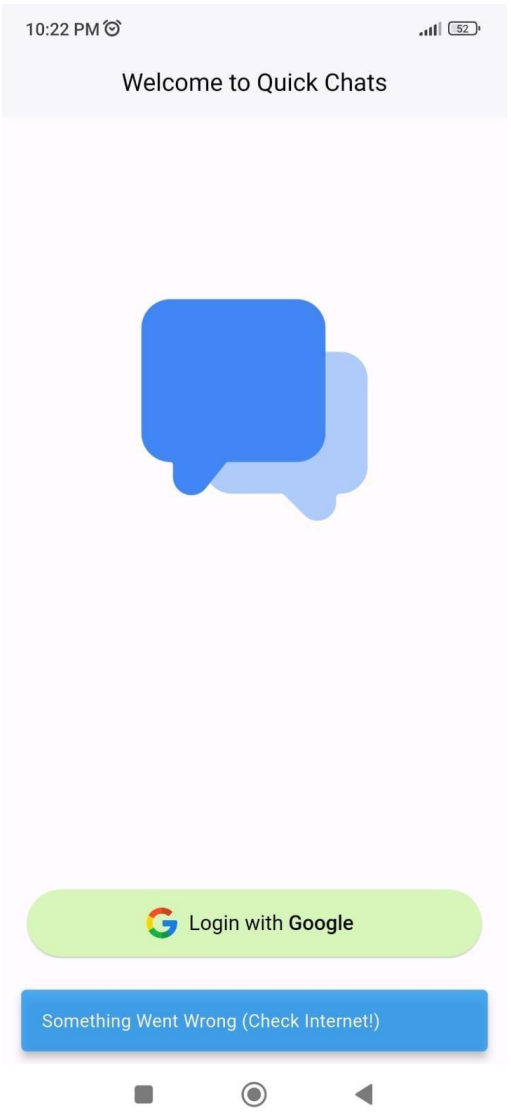
```
                    style: TextStyle(fontWeight: FontWeight.w500)),
                ]),
            ))),
      ]),
    );
  }
}
```

Output:





Project Title: Quick Chats/ PWA : Purity Plants **Roll No. 7**



MAD & PWA Lab
Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	07
Name	Yukta Bhatia
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	15

Name: Yukta Bhatia

RollNo: 07

Class: D15A

Experiment-6

Aim: To connect Flutter UI with Firebase Database.

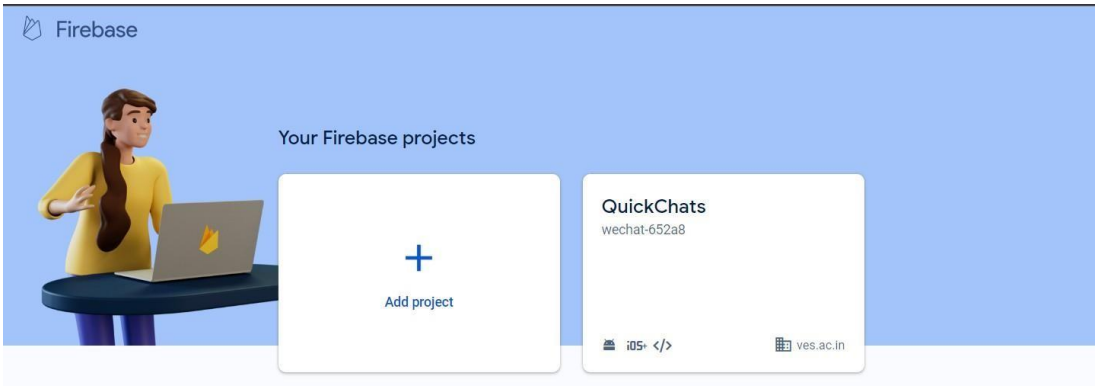
Theory :

Prerequisites :

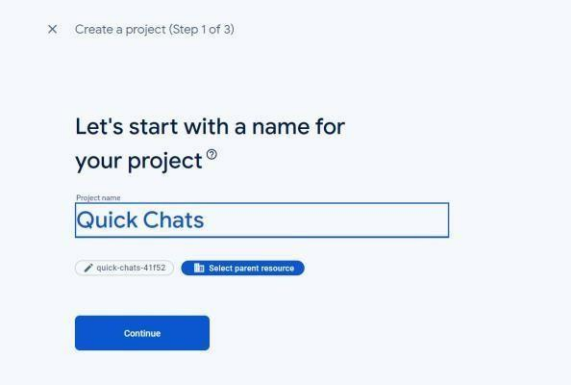
To complete this tutorial, you will need:

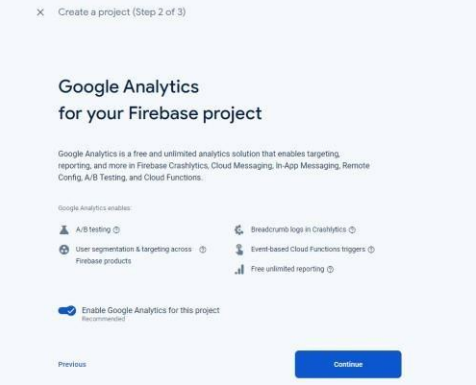
- A Google account to use Firebase.
- Developing for iOS will require XCode.
- To download and install Flutter.
- To download and install Android Studio and Visual Studio Code.
- It is recommended to install plugins for your code editor:
    - Flutter and Dart plugins installed for Android Studio.
 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

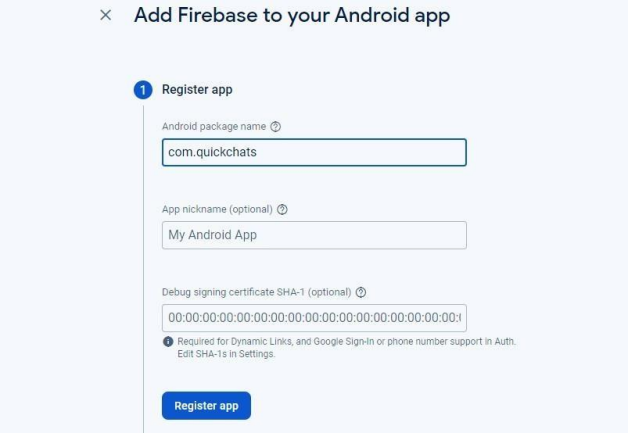
Creating a New Firebase Project :



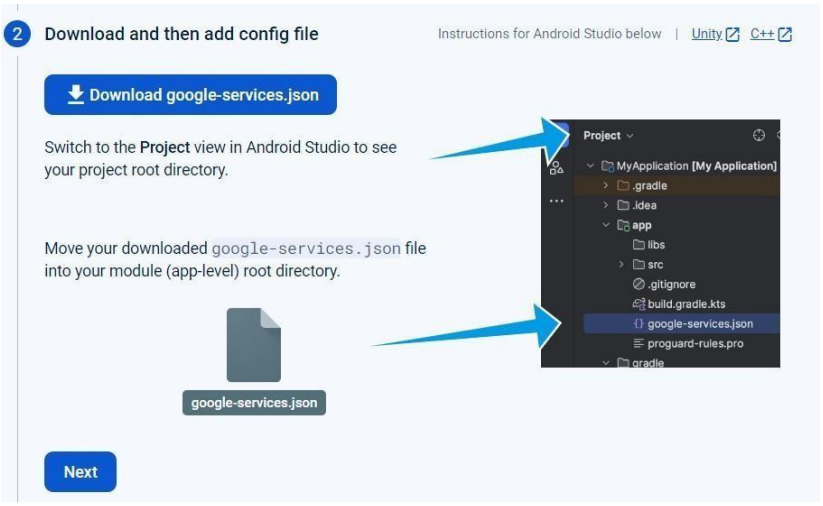
Adding Android support:







Downloading the Config File:



Adding dependencies to the project:

```
dependencies:
  flutter:
    sdk: flutter
  # For using cupertino icons
  cupertino_icons: ^1.0.2
  firebase_core: ^2.2.0
  firebase_auth: ^4.1.3
  google_sign_in: ^5.4.2
  cloud_firestore: ^4.1.0
  cached_network_image: ^3.2.3
  image_picker: ^0.8.6
  firebase_storage: ^11.0.6
  emoji_picker_flutter: ^1.5.1
  firebase_messaging: ^14.2.0
  http: ^0.13.5
  flutter_notification_channel: ^2.0.0
```

Code:

login_screen.dart

```
import
'dart:developer';
import 'dart:io';

import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:google_sign_in/google_sign_in.dart';

import '../../api/apis.dart';
import '../../helper/dialogs.dart';
import '../../main.dart';
import '../home_screen.dart';

//login screen -- implements google sign in or sign up feature for app
class LoginScreen extends StatefulWidget {
  const LoginScreen({super.key});

  @override
```

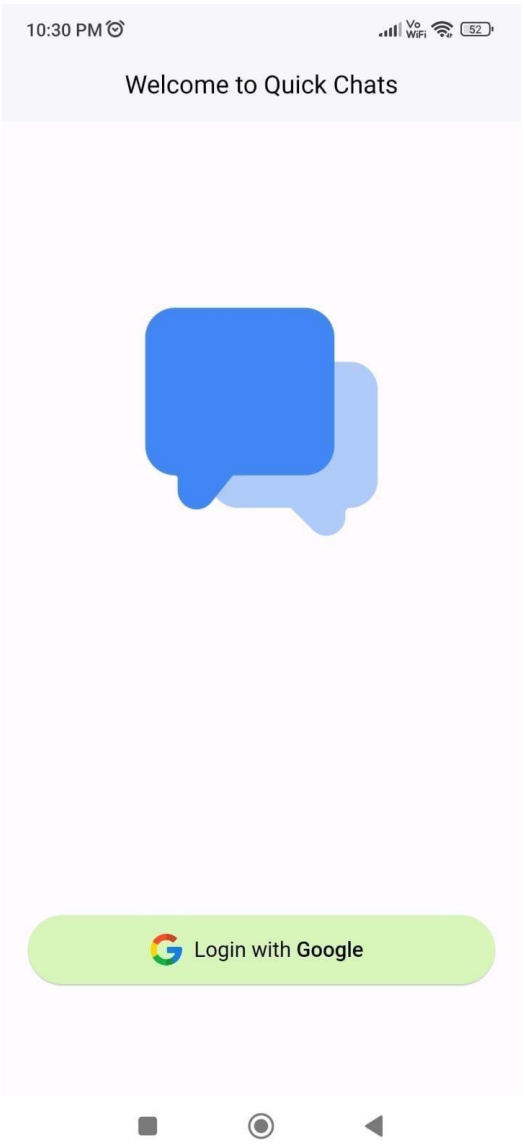
```
  State<LoginScreen> createState() => _LoginScreenState();
 }

 class _LoginScreenState extends State<LoginScreen> {
  bool _isAnimate = false;

  @override
  void initState() {
   super.initState();

   //for auto triggering animation
   Future.delayed(const Duration(milliseconds: 500), () {
    setState(() => _isAnimate = true);
   });
  }
  _handleGoogleBtnClick() {
   //for showing progress bar
   Dialogs.showProgressBar(context);

   _signInWithGoogle().then((user) async {
    //for hiding progress bar
    Navigator.pop(context);

    if (user != null) {
     log('\nUser: ${user.user}');
     log('\nUserAdditionalInfo: ${user.additionalUserInfo}');

     if ((await APIs.userExists())) {
      Navigator.pushReplacement(
          context, MaterialPageRoute(builder: (_) => const HomeScreen()));
     } else {
      await APIs.createUser().then((value) {
```

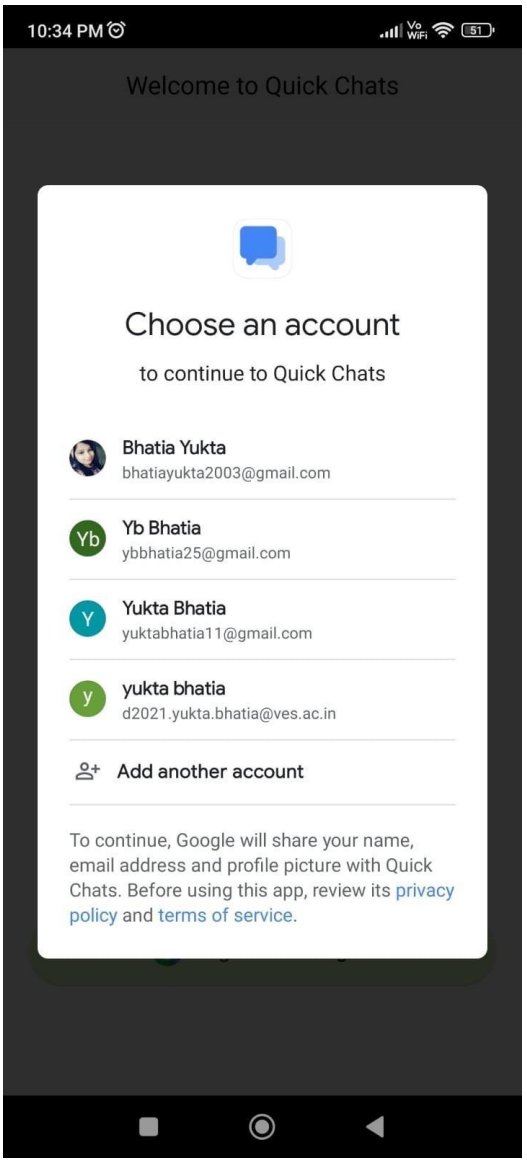
```
          Navigator.pushReplacement(
              context, MaterialPageRoute(builder: (_) => const HomeScreen()));
        });
      }
    }
  });
}

Future<UserCredential?> _signInWithGoogle() async {
  try {
    await InternetAddress.lookup('google.com');
    // Trigger the authentication flow
    final GoogleSignInAccount? googleUser = await GoogleSignIn().signIn();

    // Obtain the auth details from the request
    final GoogleSignInAuthentication? googleAuth =
        await googleUser?.authentication;

    // Create a new credential
    final credential = GoogleAuthProvider.credential(
      accessToken: googleAuth?.accessToken,
      idToken: googleAuth?.idToken,
    );

    // Once signed in, return the UserCredential
    return await APIs.auth.signInWithCredential(credential);
  } catch (e) {
    log('\n_signInWithGoogle: $e');
    Dialogs.showSnackbar(context, 'Something Went Wrong (Check Internet!)');
    return null;
  }
}
```

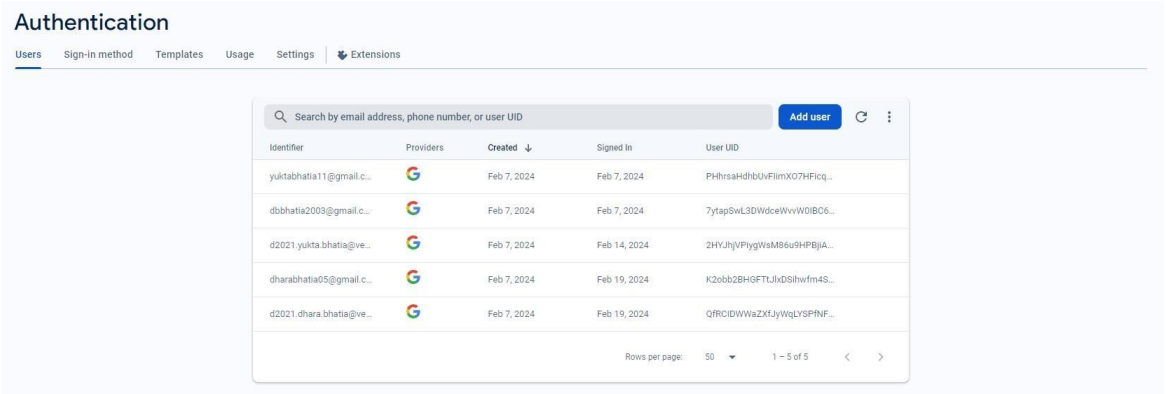
```
@override
Widget build(BuildContext context) {
  //initializing media query (for getting device screen size)
  mq = MediaQuery.of(context).size;

  return Scaffold(
    //app  bar
    appBar: AppBar(
      automaticallyImplyLeading: false,
      title: const Text('Welcome to Quick Chats'),
    ),

    //body
    body: Stack(children: [
      //app logo
      AnimatedPositioned(
          top: mq.height * .15,
          right: _isAnimate ? mq.width * .25 : -mq.width * .5,
          width: mq.width * .5,
          duration: const Duration(seconds: 1),
          child: Image.asset('images/icon.png')),

      //google login button
      Positioned(
          bottom: mq.height * .15,
          left: mq.width * .05,
          width: mq.width * .9,
          height: mq.height * .06,
          child: ElevatedButton.icon(
              style: ElevatedButton.styleFrom(
                  backgroundColor: const Color.fromARGB(255, 223, 255, 187),
```

```
            shape: const StadiumBorder(),
            elevation: 1),
          onPressed: () {
           _handleGoogleBtnClick();
          },

          //google icon
          icon: Image.asset('images/google.png', height: mq.height * .03),

          //login with google label
          label: RichText(
           text: const TextSpan(
              style: TextStyle(color: Colors.black, fontSize: 16),
              children: [
                TextSpan(text: 'Login with '),
                TextSpan(
                  text: 'Google',
                  style: TextStyle(fontWeight: FontWeight.w500)),
              ]),
          ))),
    ]),
  );
 }
}
```

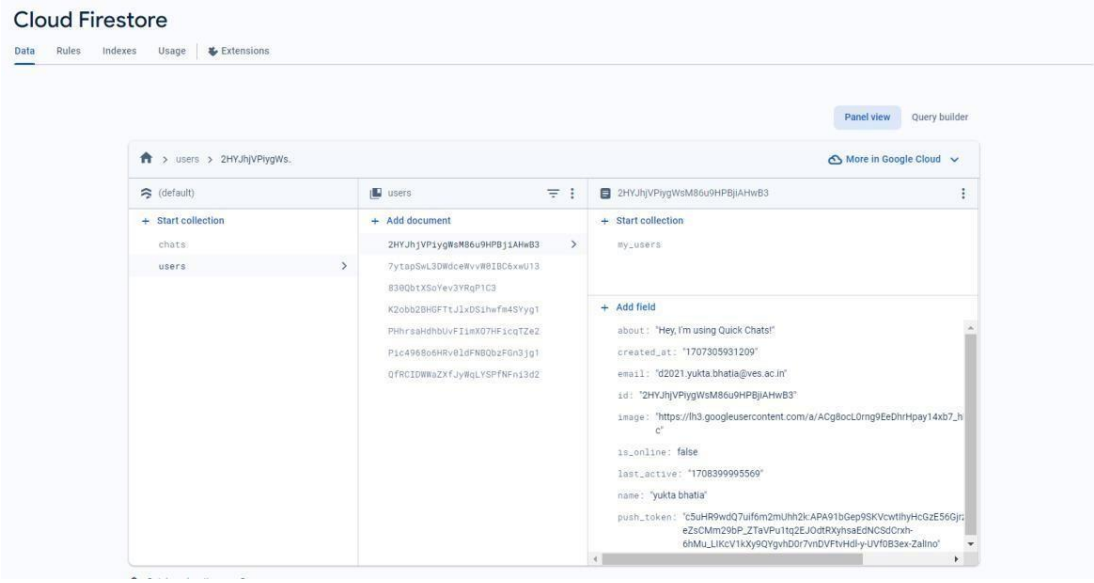
Output:


10:30 PM
Welcome to Quick Chats
Login with Google


10:34 PM
Welcome to Quick Chats
Choose an account
to continue to Quick Chats
Bhatia Yukta
bhatiayukta2003@gmail.com
Yb Bhatia
ybbhatia25@gmail.com
Yukta Bhatia
yuktabhatia11@gmail.com
yukta bhatia
d2021.yukta.bhatia@ves.ac.in
Add another account
To continue, Google will share your name, email address and profile picture with Quick Chats. Before using this app, review its privacy policy and terms of service.


Authentication
Users
Sign-in method
Templates
Usage
Settings
Extensions
Search by email address, phone number, or user UID
Add user
Identifier
Providers
Created
Signed In
User UID
Rows per page:
1 – 5 of 5

Project Title: Quick Chats/ PWA : Purity Plants **Roll No. 7**



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.	07
Name	Yukta Bhatia
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	15

Name: Yukta Bhatia
RollNo: 07
Class: D15A

Experiment-7

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

Theory:

Regular Web App

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

Progressive Web App

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

Difference between PWAs vs. Regular Web Apps:

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

1. Native Experience

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

2. Ease of Access

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

3. Faster Services

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

4. Engaging Approach

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

5. Updated Real-Time Data Access

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

The main features are:

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

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

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

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

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

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

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

Code:

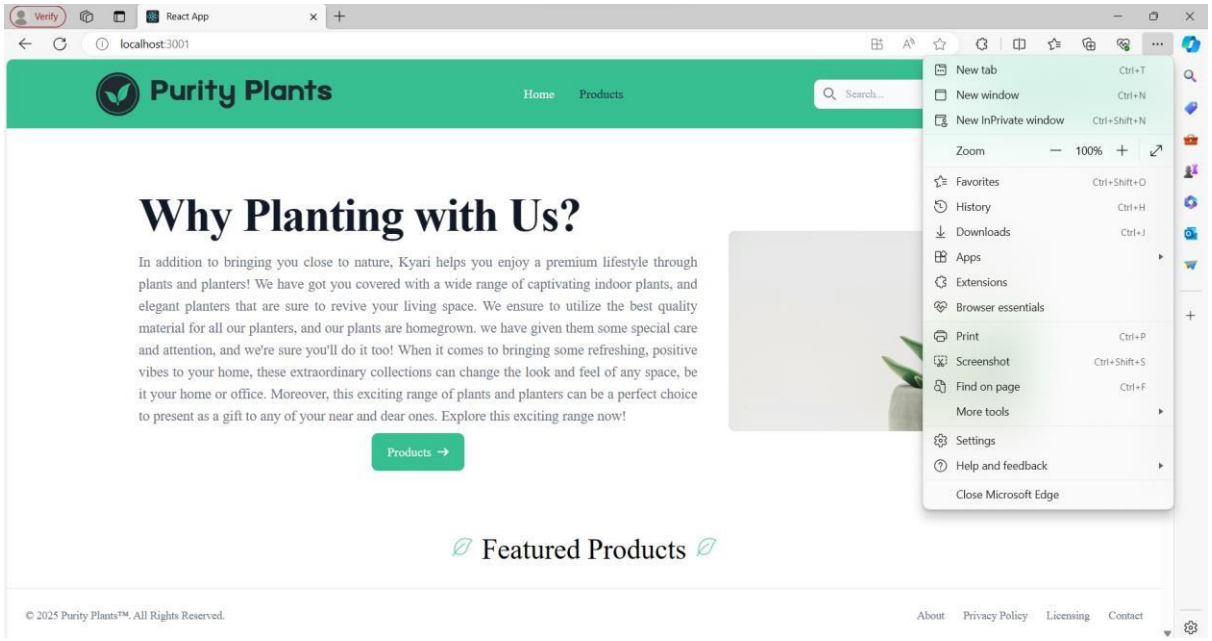
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
```

```
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />

    <meta name="description" content="Web site created using create-react-app"/>
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />

    <!-- Manifest File link -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />

    <meta name="apple-mobile-web-app-status-bar" content="#aa7700">
    <meta name="theme-color" content="black">
</head>
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
     <script>
      // Add event listener to execute code when page loads
      window.addEventListener('load', () => {
        // Call registerSW function when page loads
        registerSW();
      });

      // Register the Service Worker
      async function registerSW() {
        // Check if browser supports Service Worker
        if ('serviceWorker' in navigator) {
          try {
            // Register the Service Worker named 'serviceworker.js'
            await navigator.serviceworker.register('serviceworker.js');
          }
          catch (e) {
            // Log error message if registration fails
            console.log('SW registration failed');
          }
        }
      }
    </script>
  </body>
</html>
```

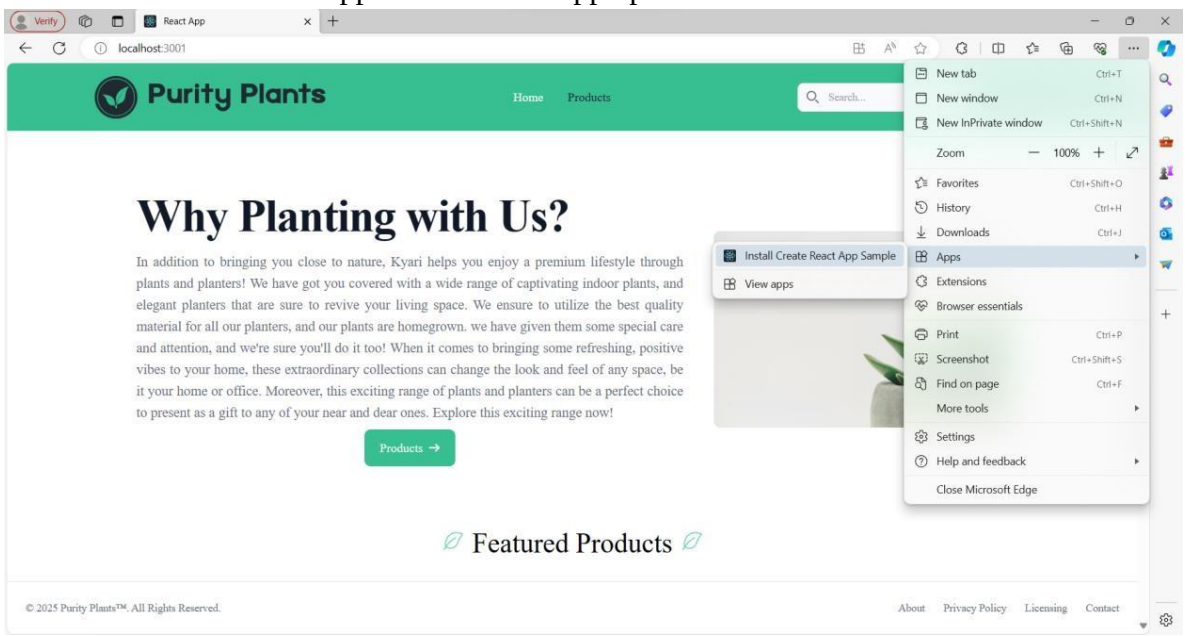
Output:

Open folder in VS code and click go live at bottom right corner
Open your hosted site on Microsoft Edge

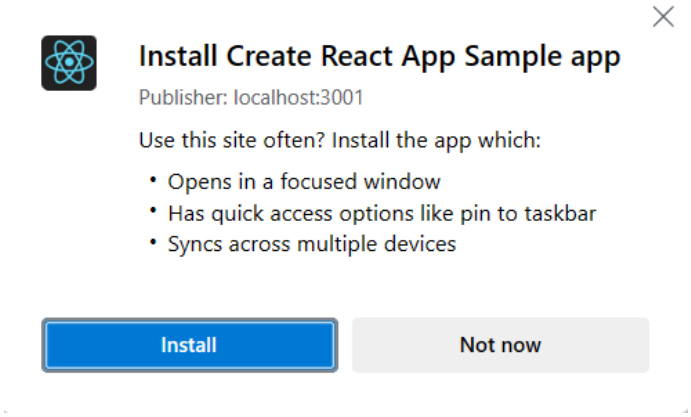
Project Title: Quick Chats/ PWA : Purity Plants **Roll No. 7**

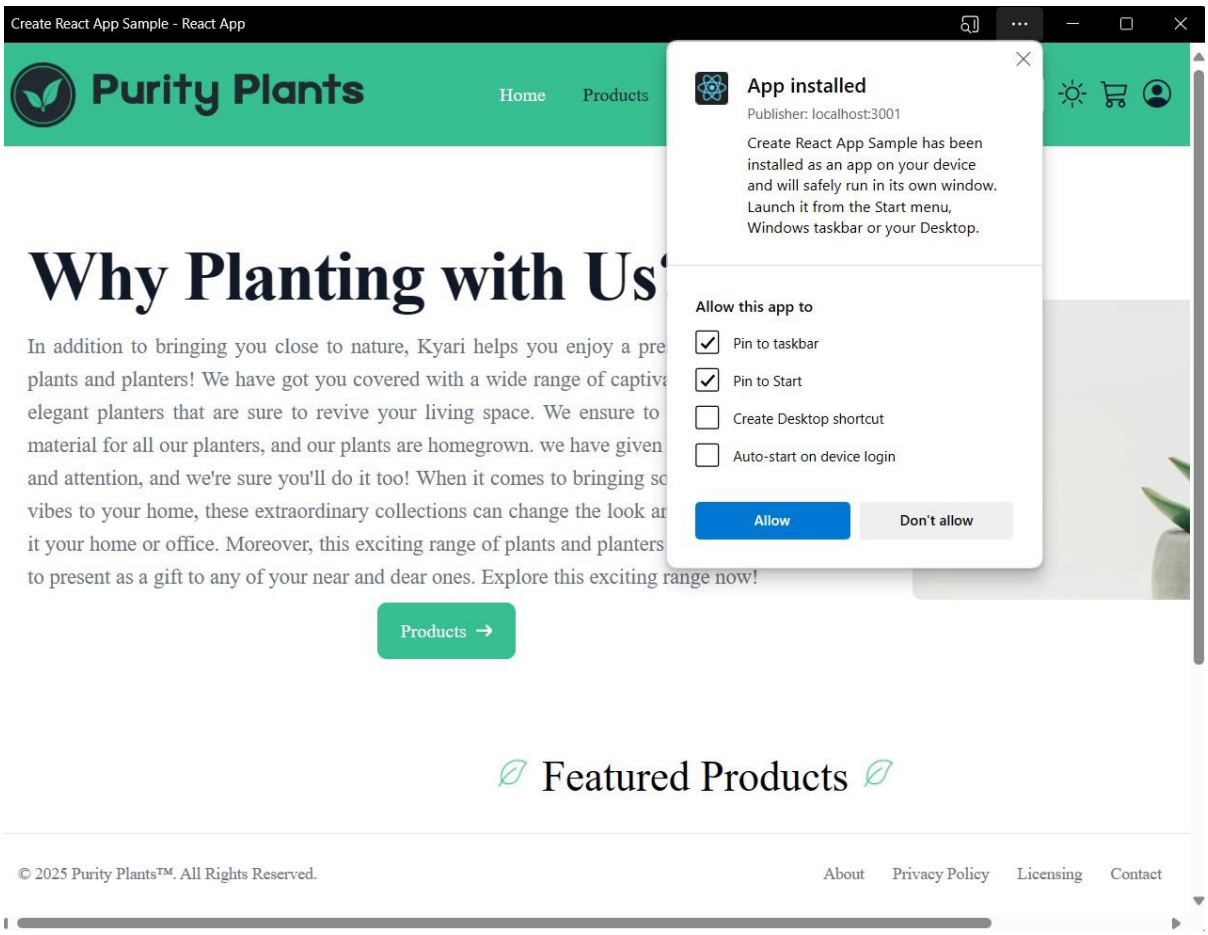


Click on 3 dots click on Apps select install app option



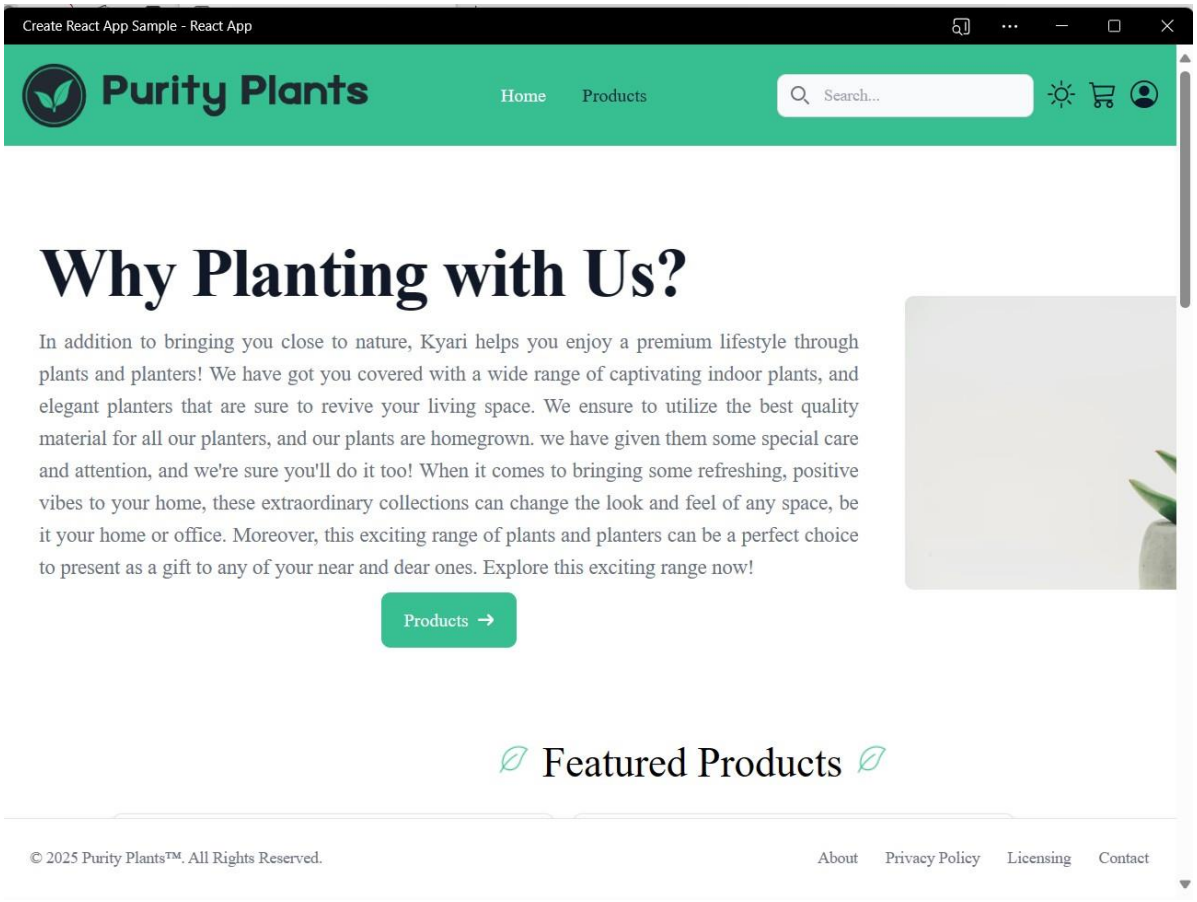
Click on Install





Desktop App Created Successfully

Open Desktop App:



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

MAD & PWA Lab
Journal

Experiment No.	08
Experiment Title.	To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA
Roll No.	07
Name	Yukta Bhatia
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

Name: Yukta Bhatia
RollNo: 07
Class: D15A

Experiment-8

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

Theory:

Service Worker

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

Things to note about Service Worker:

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

What can we do with Service Workers?

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

- You can **Cache**
  You can cache any request/response pair with Service Worker and Cache API and you can access these offline content anytime.
- You can manage **Push Notifications**
  You can manage push notifications with Service Worker and show any information message to the user.

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.

Code:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />

    <meta name="description" content="Web site created using create-react-app"/>
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />

    <!-- Manifest File link -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />

    <meta name="apple-mobile-web-app-status-bar" content="#aa7700">
    <meta name="theme-color" content="black">
</head>
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
     <script>
      // Add event listener to execute code when page loads
      window.addEventListener('load', () => {
        // Call registerSW function when page loads
        registerSW();
      });

      // Register the Service Worker
      async function registerSW() {
        // Check if browser supports Service Worker
        if ('serviceWorker' in navigator) {
          try {
            // Register the Service Worker named 'serviceworker.js'
            await navigator.serviceworker.register('serviceworker.js');
          }
```

```
        catch (e) {
          // Log error message if registration fails
          console.log('SW registration failed');
        }
      }
    }
  </script>
 </body>
</html>
```

index.css

```
* {
 font-family: 'Open Sans, sans-serif';
}

main {
 min-height: 83vmin;
 padding-bottom: 69px;
 width: 1200px;
 margin: auto;
}

.slick-next:before,
.slick-prev:before {
 color: #37BE91!important;
 font-size: 2rem!important;
}

.slick-dots li button:before{
 color: #37BE91!important;
}

.sectionTitle {
 position: relative;
 align-items: center;
 text-align: center;
}

.sectionTitle::before {
 content: '';
 width: 30px;
 height: 30px;
 margin-right: 10px;
 background: url('./assets/leaf.png') no-repeat;
 background-size: cover;
 display: inline-block;
}
.sectionTitle::after {
 content: '';
```

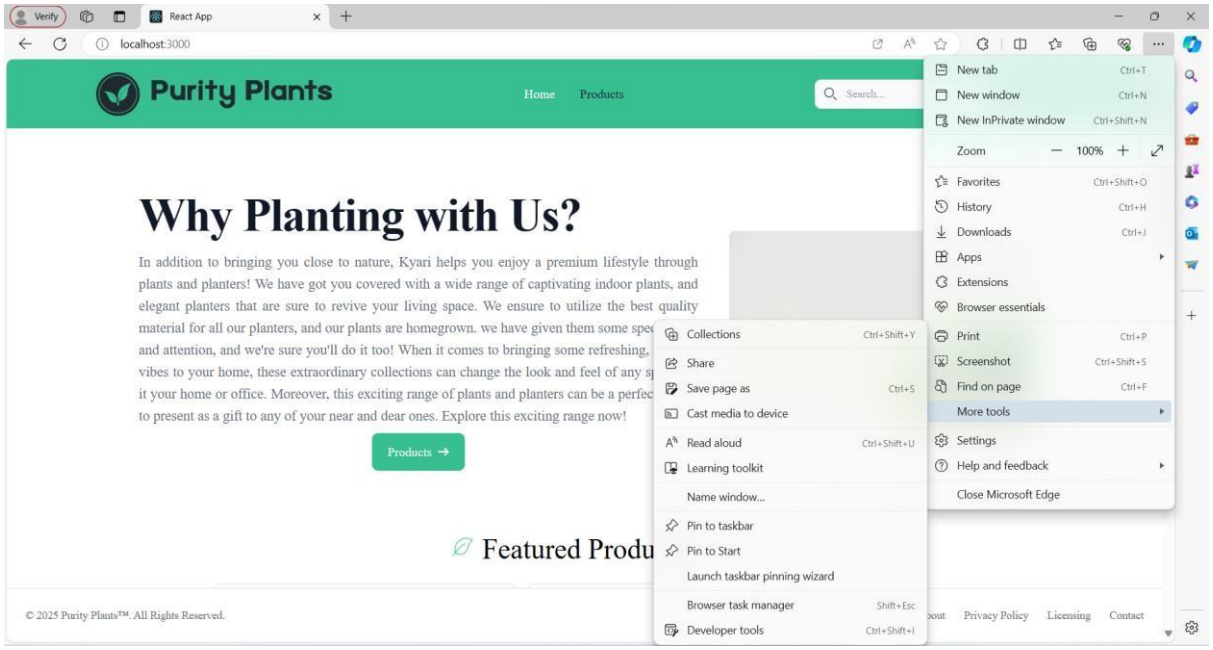
```
  width: 30px;
  height: 30px;
  margin-left: 10px;
  background: url('./assets/leaf.png') no-repeat;
  background-size: cover;
  display: inline-block;
}
```

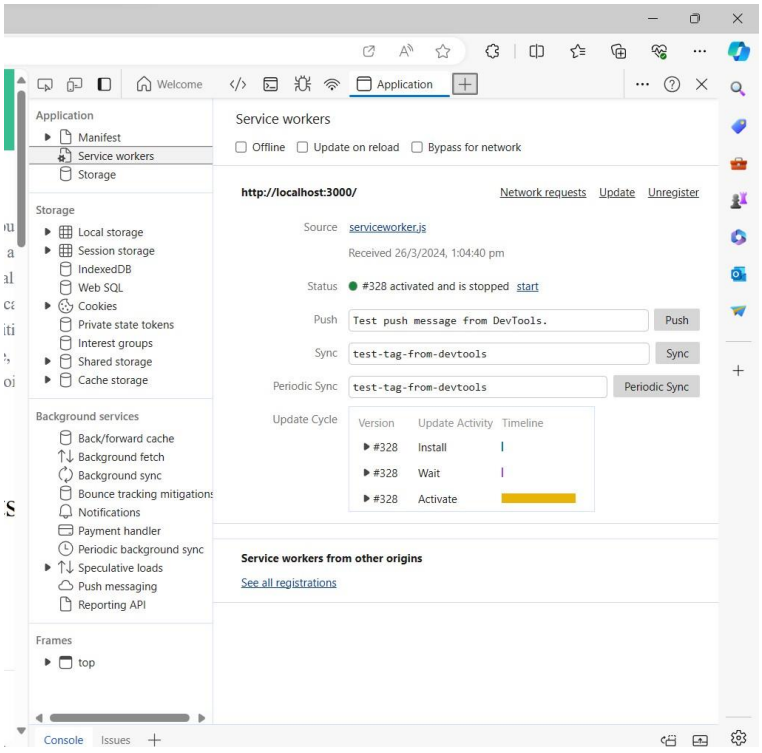
serviceworker.js

```
const cacheName = 'Purity Plants';
const assetsToCache = [
        '/',
        '/index.html',
        '/index.css',
]
this.addEventListener('install', event => {
        event.waitUntil(
                caches.open(cacheName)
                        .then(cache => {
                        return cache.addAll(assetsToCache);
                })
        );
});
this.addEventListener('activate', event => {
        event.waitUntil(
                caches.keys().then(cacheNames => {
                        return Promise.all(
                                cacheNames.filter(name => {
                                return name !== cacheName;
                                }).map(name => {
                                return caches.delete(name);
                                })
                        );
                })
        );
});
```

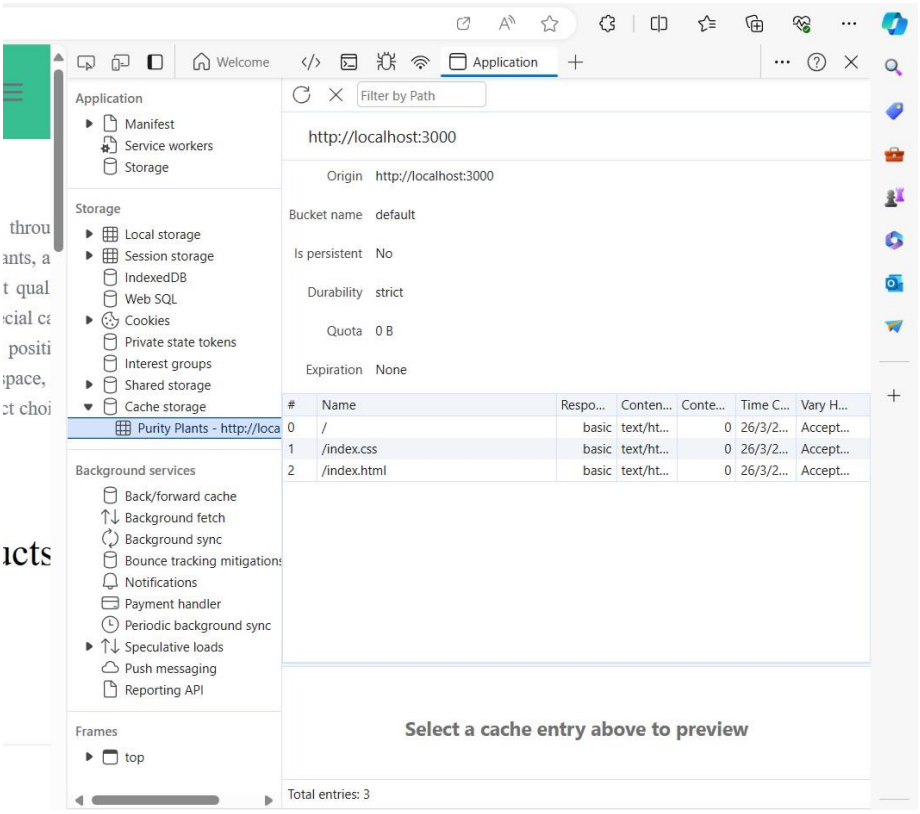
Project Title: Quick Chats/ PWA : Purity Plants **Roll No. 7**

Output:





Project Title: Quick Chats/ PWA : Purity Plants **Roll No. 7**



Conclusion: Registered a service worker, and completed the installation and activation process for a new service worker for the PWA.

MAD & PWA Lab
Journal

Experiment No.	09
Experiment Title.	To implement Service worker events like fetch, sync and push for E-commerce PWA
Roll No.	07
Name	Yukta Bhatia
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

Name: Yukta Bhatia
RollNo: 07
Class: D15A

Experiment-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 networkrequests 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.
- 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 thenetwork, 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");
    }
}
```

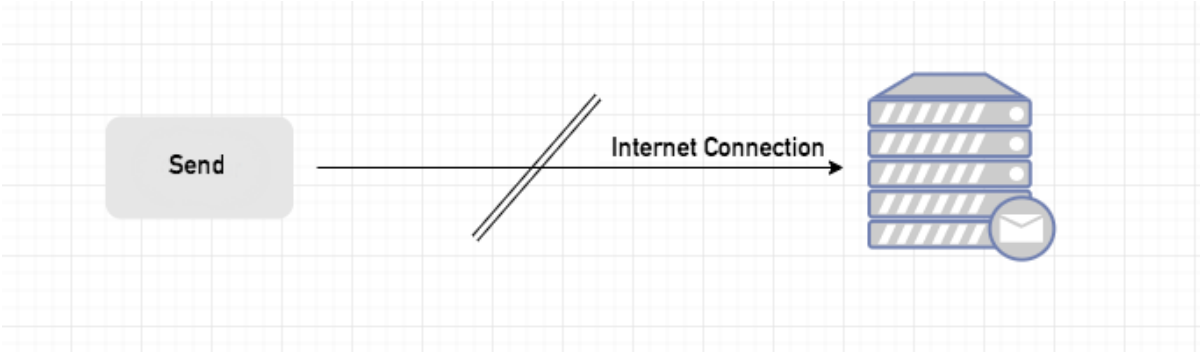
Sync Event

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.

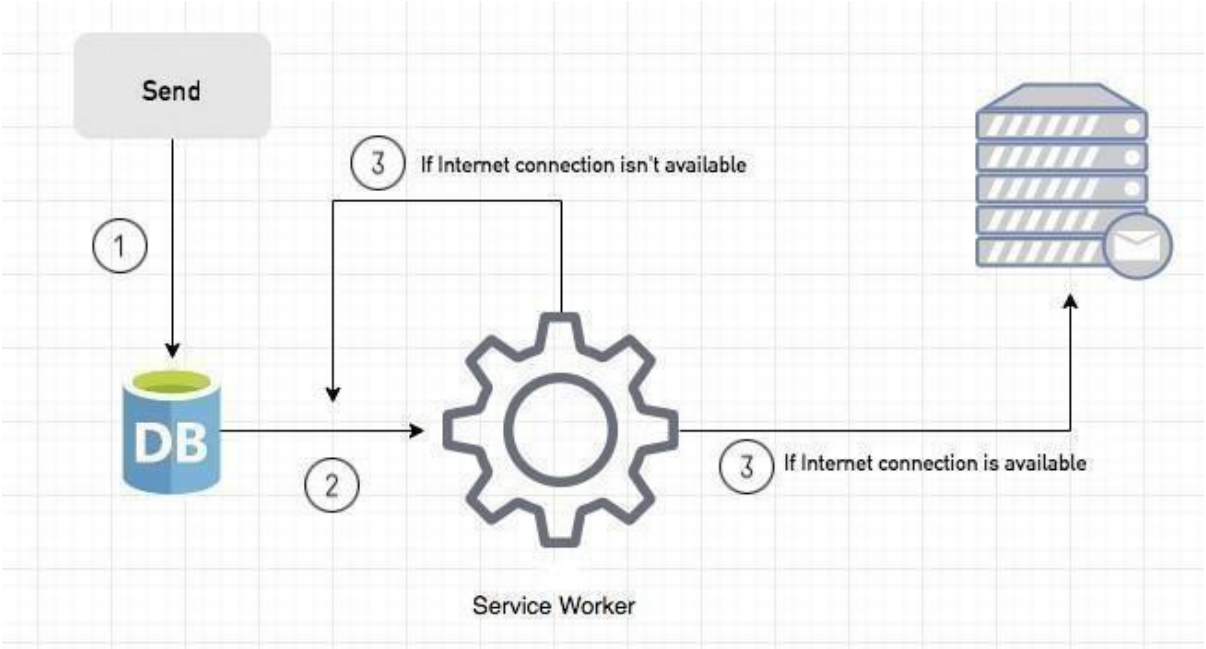
Here is a job for the Background Sync.

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.

Push Event

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

"Notification.requestPermission();" is the necessary line to show notification to the user.

Code:-

serviceworker.js

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

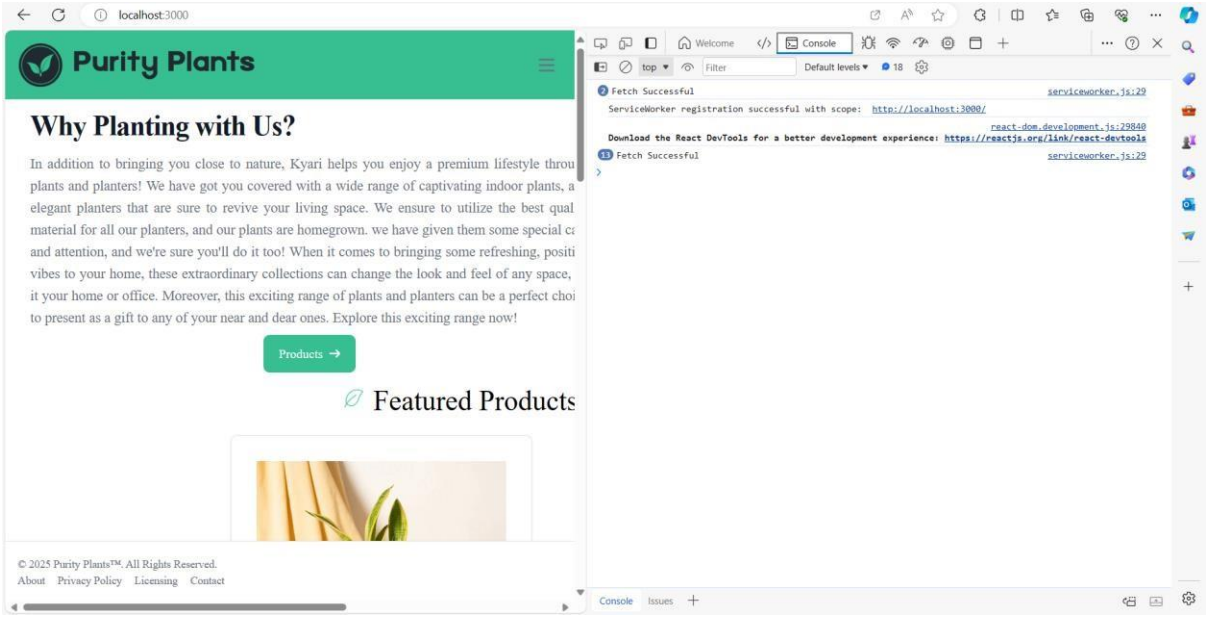
```
});

this.addEventListener("fetch", function (event) {
   event.respondWith(
      checkResponse(event.request).catch(function () {
         console.log("Fetch from cache successful!");
         return returnFromCache(event.request);
      })
   );
   // Moved inside respondWith to correctly reflect fetch success
         console.log("Fetch Successful");
   event.waitUntil(addToCache(event.request));
});

this.addEventListener("sync", (event) => {
   if (event.tag === "syncMessage") {
      console.log("Sync successful!");
   }
});

this.addEventListener("push", function (event) {
         if (event && event.data) {
         try {
         var data = event.data.json();
         if (data && data.method === "pushMessage") {
         console.log("Push notification sent");
         this.registration.showNotification("Ecommerce website", {
         body: data.message,
         });
         }
         } catch (error) {
         console.error("Error parsing push data:", error);
         }
         }
         });

function preLoad() {
   return caches.open("offline").then(function (cache) {
      // caching index and important routes
      return cache.addAll([
         "/",
         "/index.html",
         "/index.css",
         "/offline.html"
      ]);
```

```
    });
}

function checkResponse(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);
            });
    });
}

function returnFromCache(request) {
    return caches.open("offline").then(function (cache) {
        return cache.match(request).then(function (matching) {
            return matching || cache.match("/offline.html");
        });
    });
}


function addToCache(request) {
    return caches.open("offline").then(function (cache) {
        return fetch(request).then(function (response) {
            if (!response.ok && response.status !== 404) {
                throw new Error(`Failed to fetch and cache: ${request.url}, Status: ${response.status}`);
            }
            return cache.put(request, response.clone()).then(function () {
                return response;
            });
        }).catch(error => {
            console.error(`Fetch failed for ${request.url}:`, error);
            throw error;
        });
    });
}
```

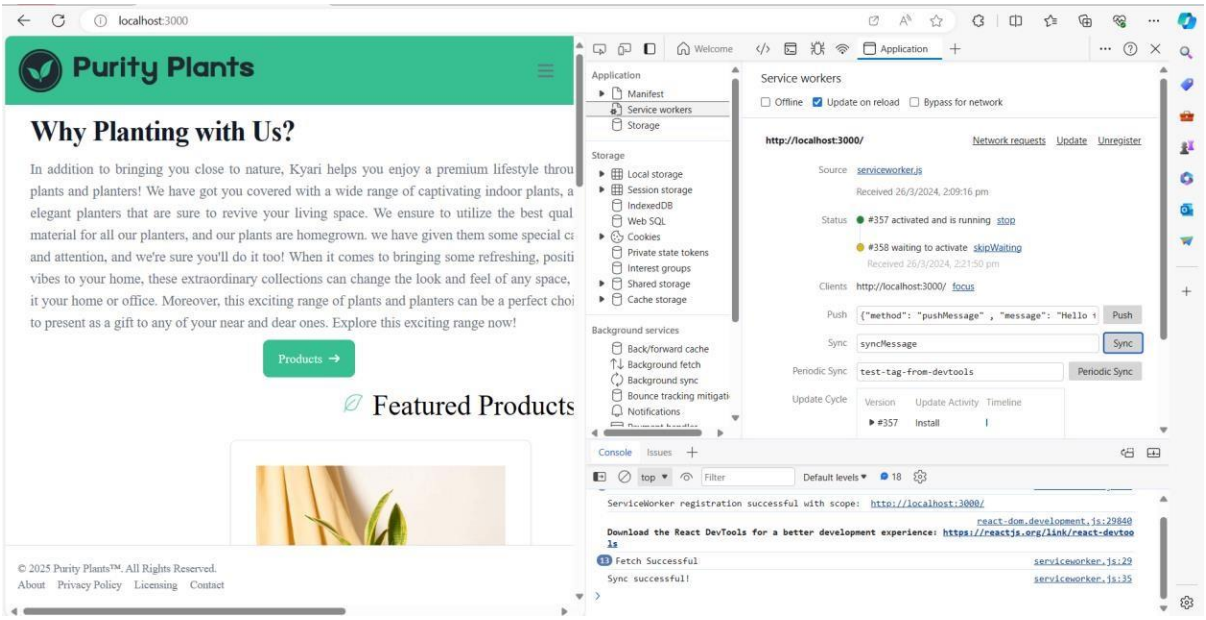
Project Title: Quick Chats/ PWA : Purity Plants **Roll No. 7**

Output:

FETCH

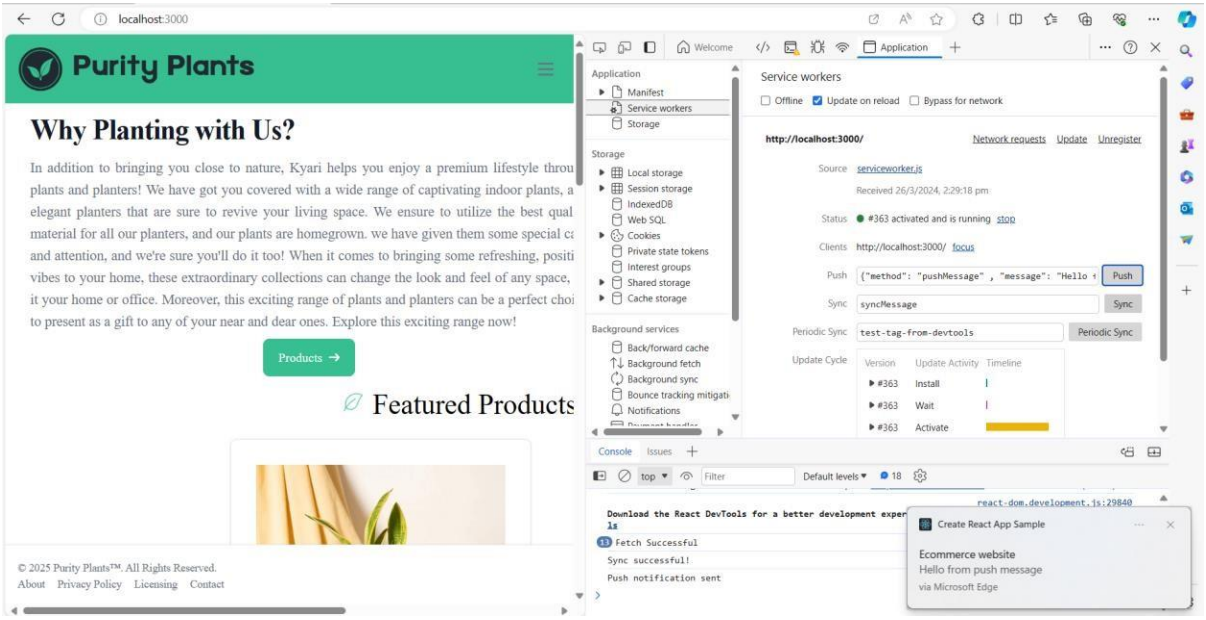


SYNC



Project Title: Quick Chats/ PWA : Purity Plants **Roll No. 7**

PUSH



Conclusion:- In this experiment we have implemented service worker events like fetch, push and sync for E-commerce PWA

MAD & PWA Lab
Journal

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	07
Name	Yukta Bhatia
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

Project Title: Quick Chats/ PWA : Purity Plants **Roll No. 7**

Name: Yukta Bhatia
RollNo: 07
Class: D15A

Experiment-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

Project Title: Quick Chats/ PWA : Purity Plants **Roll No. 7**

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

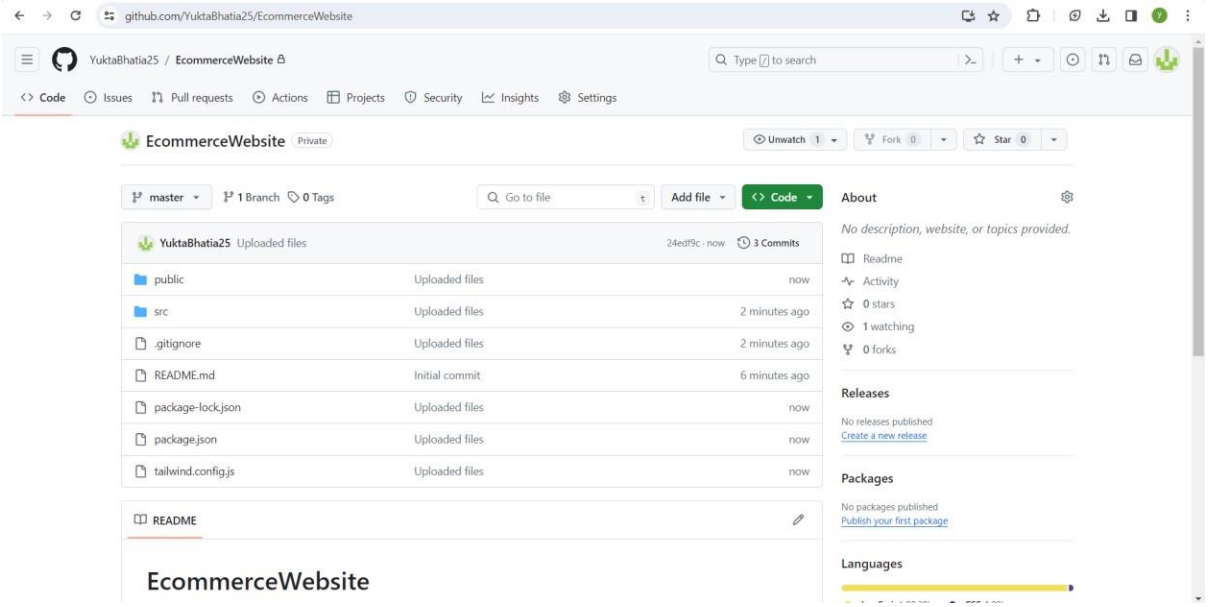
1. Hosted by Google. Enough said.
2. Authentication, Cloud Messaging, and a whole lot of other handy services will be available to you.
3. A real-time database will be available to you, which can store 1 GB of data.
4. You'll also have access to a blob store, which can store another 1 GB of data.
5.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

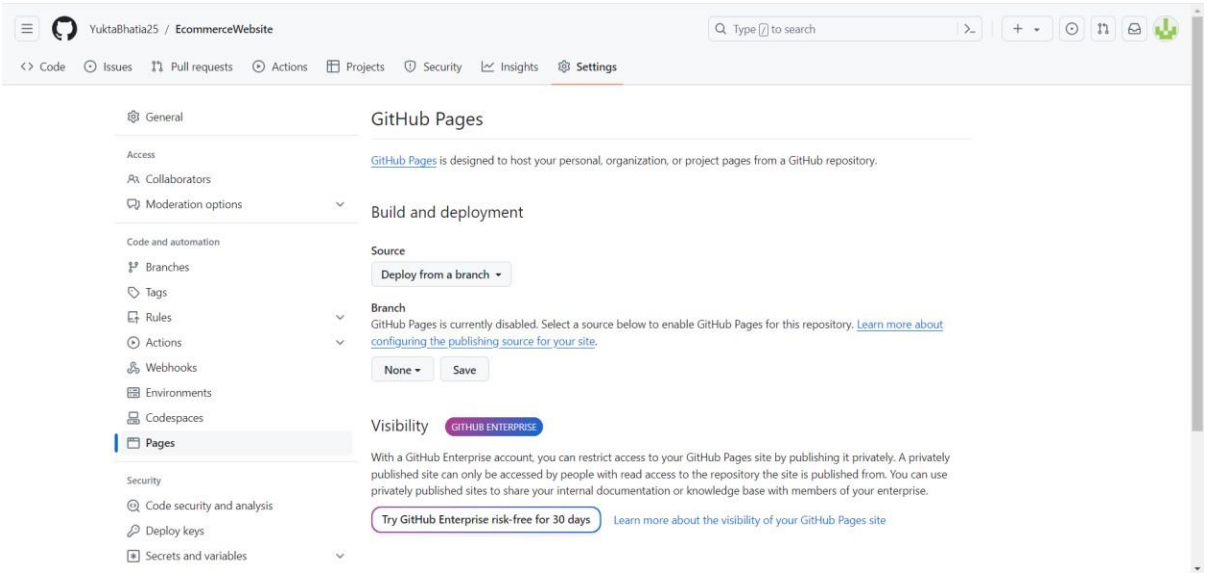
Cons

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.

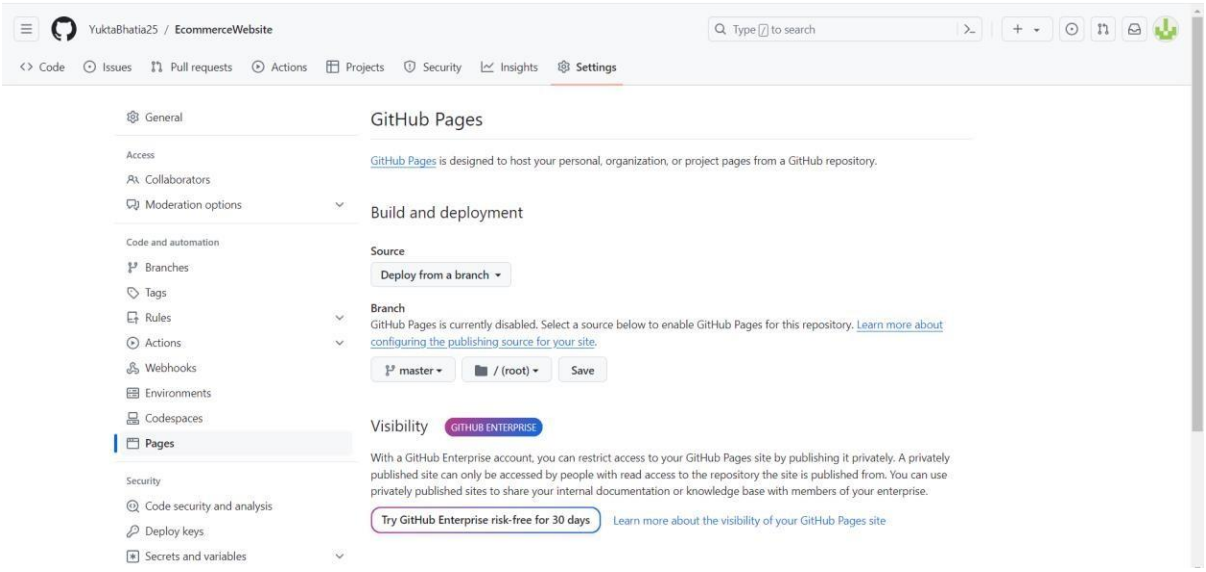
Project Title: Quick Chats/ PWA : Purity Plants **Roll No. 7**

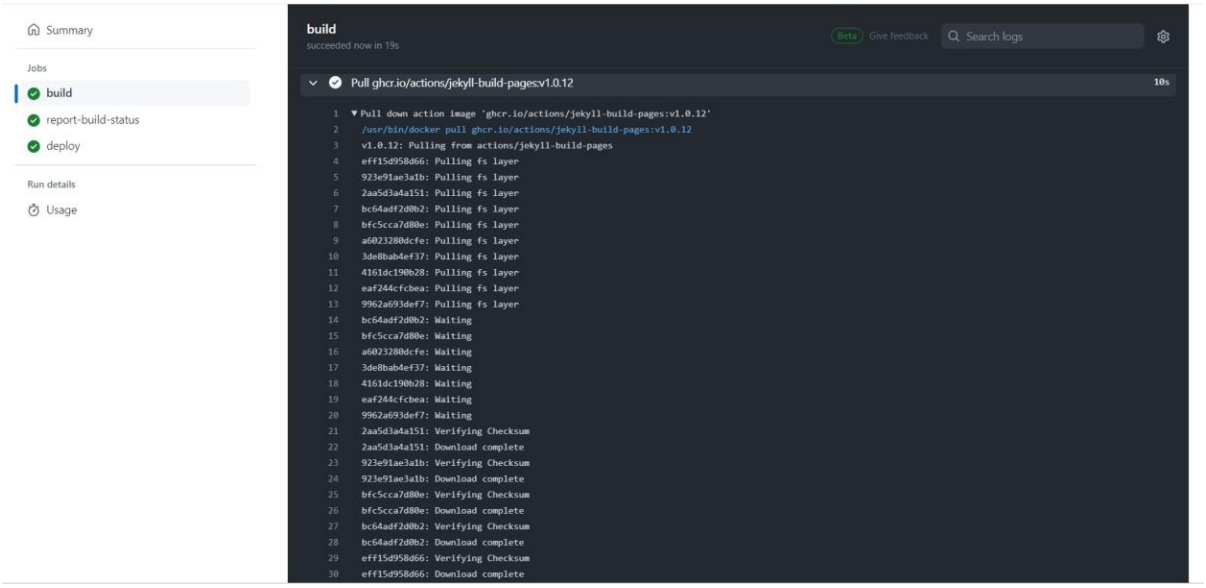
Implementation:

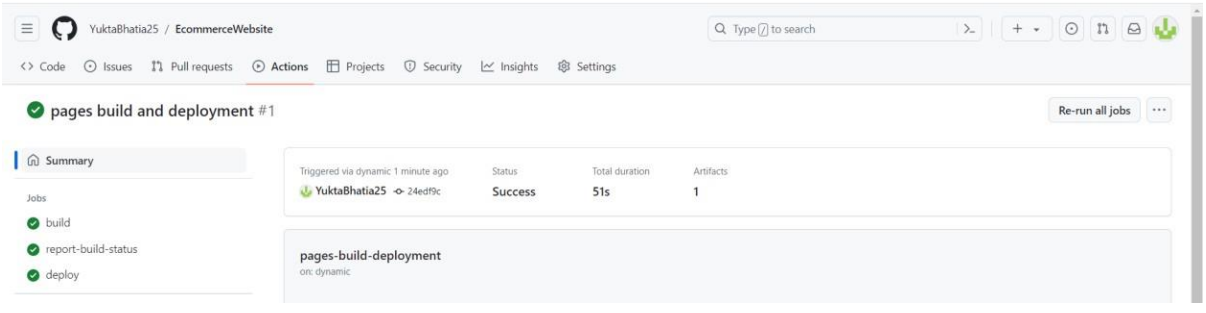




Project Title: Quick Chats/ PWA : Purity Plants **Roll No. 7**







Github repository:- https://github.com/YuktaBhatia25/EcommerceWebsite

Conclusion:- In this experiment, Website has been deployed 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.	07
Name	Yukta Bhatia
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	15

Project Title: Quick Chats/ PWA : Purity Plants **Roll No. 7**

Name: Yukta Bhatia
RollNo: 07
Class: D15A

Experiment-11

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

Theory:

Google Lighthouse

Lighthouse is an open-source, automated tool for improving the quality of web pages. You can run it against any web page, public or requiring authentication. It has audits forperformance, accessibility, progressive web apps, SEO and more.

You can run Lighthouse in Chrome DevTools, from the command line, or as a Nodemodule. You give

Lighthouse a URL to audit, it runs a series of audits against the page, and then it generates a report on how well the page did. From there, use the failing audits as indicators on how to improve the page. Each audit has a reference doc explaining whythe audit is important, as well as how to fix it.

Features of Lighthouse

Google Lighthouse gives a breakdown of your site into the accompanying metrics.Here is a brief explanationof each of the aforementioned metrics:

1. Performance

Performance is generally viewed as the most valuable metric given by the Google Lighthouse tool. Like the PageSpeed Insights, the Performance area of the Lighthousereport contains a few helpful metrics you can use to advance your site to climb Google's rankings. The Performance segment of the Lighthouse report joins the Opportunities, Field Data, Lab Data, and Diagnostics metrics of the PageSpeed Insights tool.

A great example is the opportunities metric as it flags three types of render-blockingURL's namely stylesheets, scripts, and HTML imports. This merged perspective on performance metrics gives an exact and valuable analysis of your site's performanceand any progressions you should make to expand your site's exhibition.

2. Accessibility

The first of the new regions of Google Lighthouse is the Accessibility metric. Basicallywhat this metric does is feature potential chances to improve the availability and clientexperience of your mobile app or website.

Following the accessibility improvement report will guarantee that your clients can without much of a stretch explore and utilize your site. Just as guaranteeing that youhave the most obvious opportunity with regards to positioning better on web search engines.

3. Best Practices

Another segment new to Google's analysis tools is the Best Practices metric. This region of the Lighthouse report doesn't carefully give execution related data. However,it will give you recommendations which can improve both your exhibition and client experience, particularly for mobile sites.

4. SEO

The latest and most dynamic of the highlights in Google's Lighthouse instrument is the SEO metric.

PageSpeed Insights doesn't offer this tool. This is why most web designers and SEO specialists prefer to utilize Google Lighthouse to analyze a website. The SEO metric gives fundamental tools to examine your page's streamlining for search engine results rankings. While there are numerous more factors which Lighthouse doesn't consider orquantify, the most essential focuses are secured.

5. Progressive Web Applications

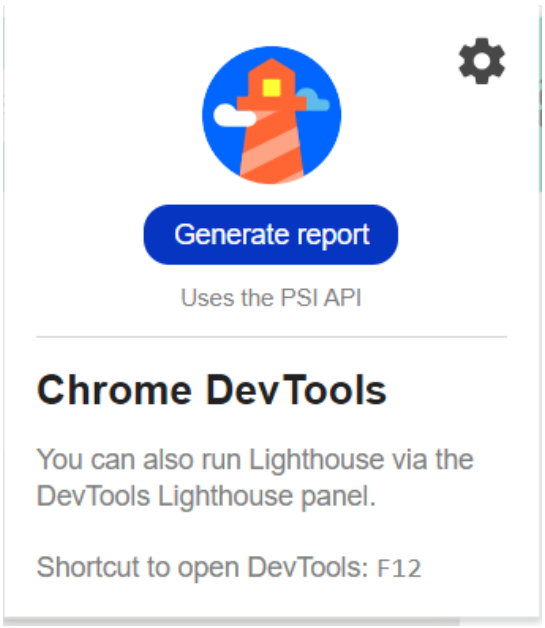
The Progressive Web App area is another of Google's most up to date execution measurements incorporated into its Lighthouse tool. While the meaning of a Progressive Web App (PWA) hasn't been especially clear, Google's documentation expresses that there are a few key variables which make a site a PWA. A great feature of this metric is registering service workers which allow you to enable push notificationson your web app

Code:

Manifest.json

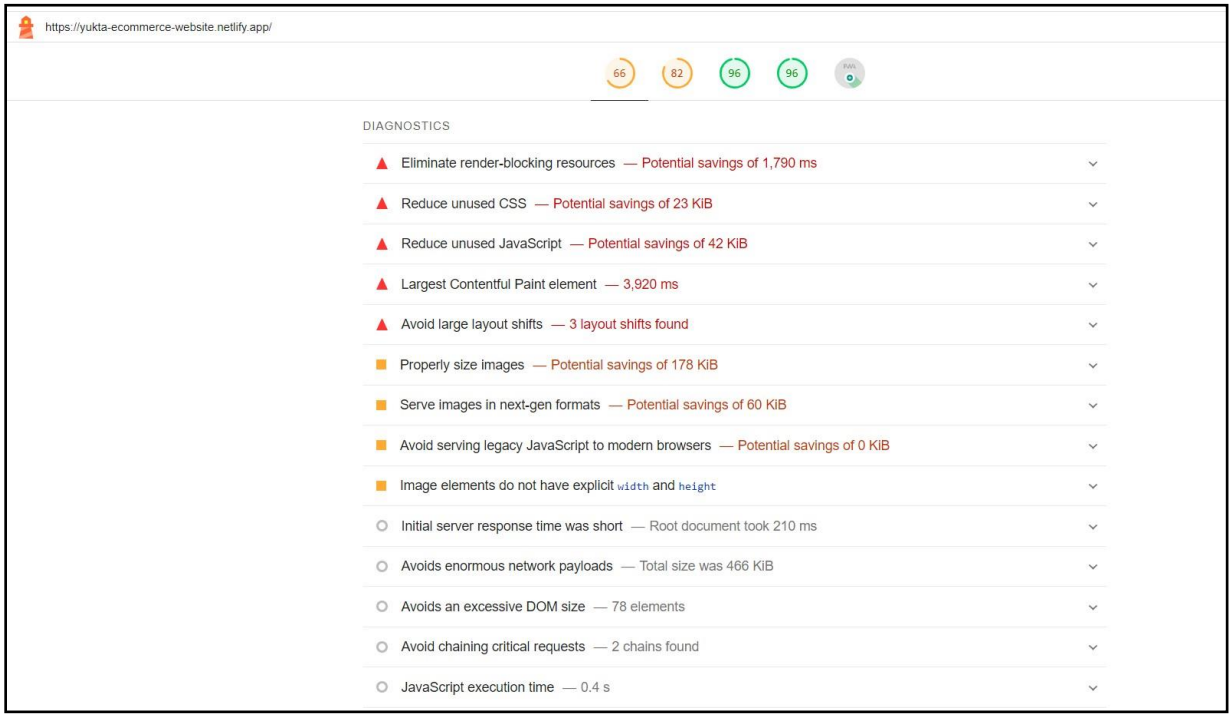
```
{
  "short_name": "React App",
  "name": "Create React App Sample",
  "icons": [
    {
      "src": "favicon.ico",
      "sizes": "64x64 32x32 24x24 16x16",
      "type": "image/x-icon",
      "purpose":"any maskable"

    },
```

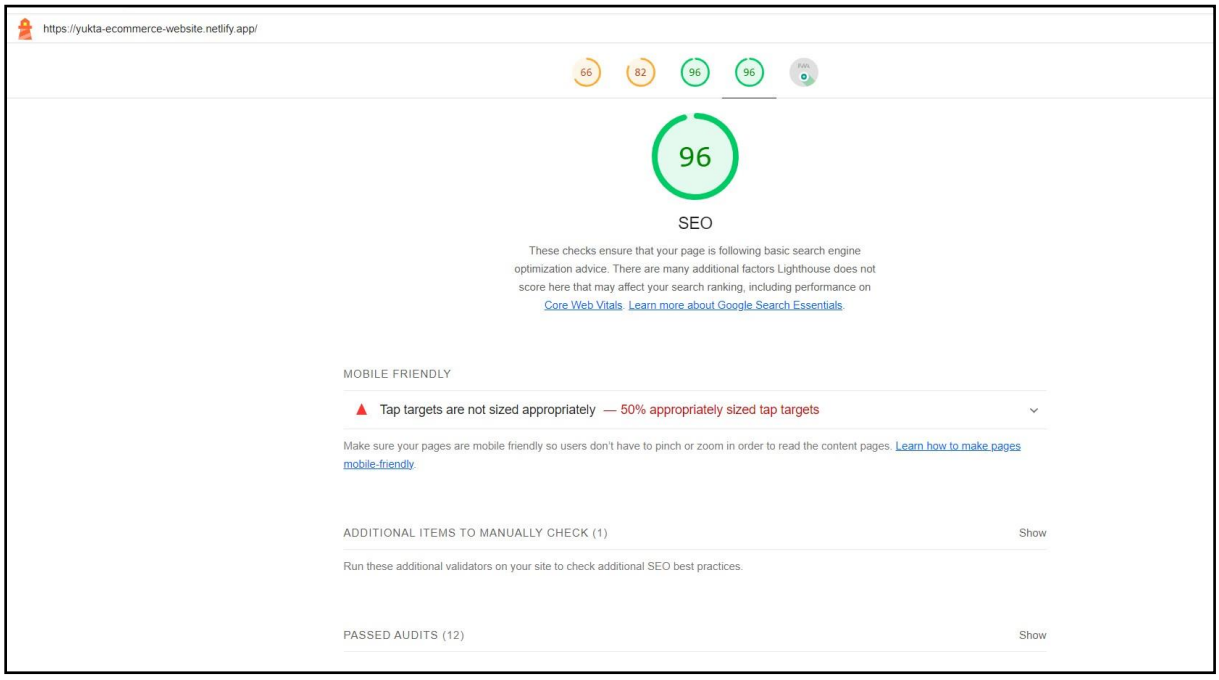
```
    {
      "src": "logo192.png",
      "type": "image/png",
      "sizes": "192x192",
      "purpose":"any maskable"
    },
    {
      "src": "logo512.png",
      "type": "image/png",
      "sizes": "512x512",
      "purpose":"any maskable"
    }
  ],
  "start_url": ".",
  "display": "standalone",
  "theme_color": "#000000",
  "background_color": "#ffffff"
}
```



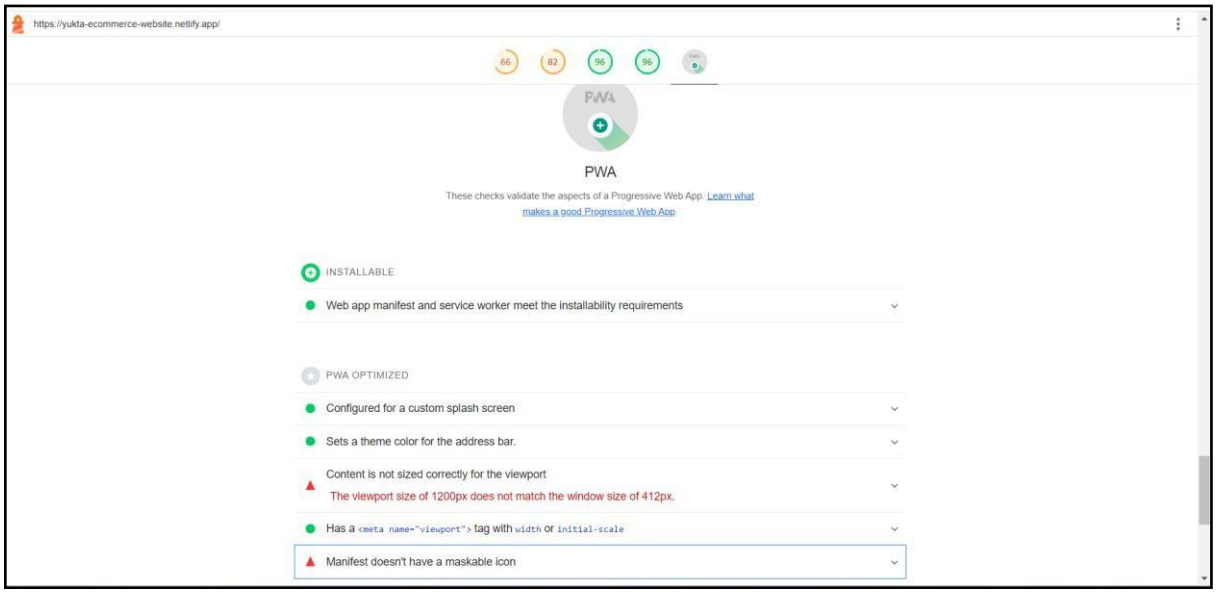
Project Title: Quick Chats/ PWA : Purity Plants **Roll No. 7**

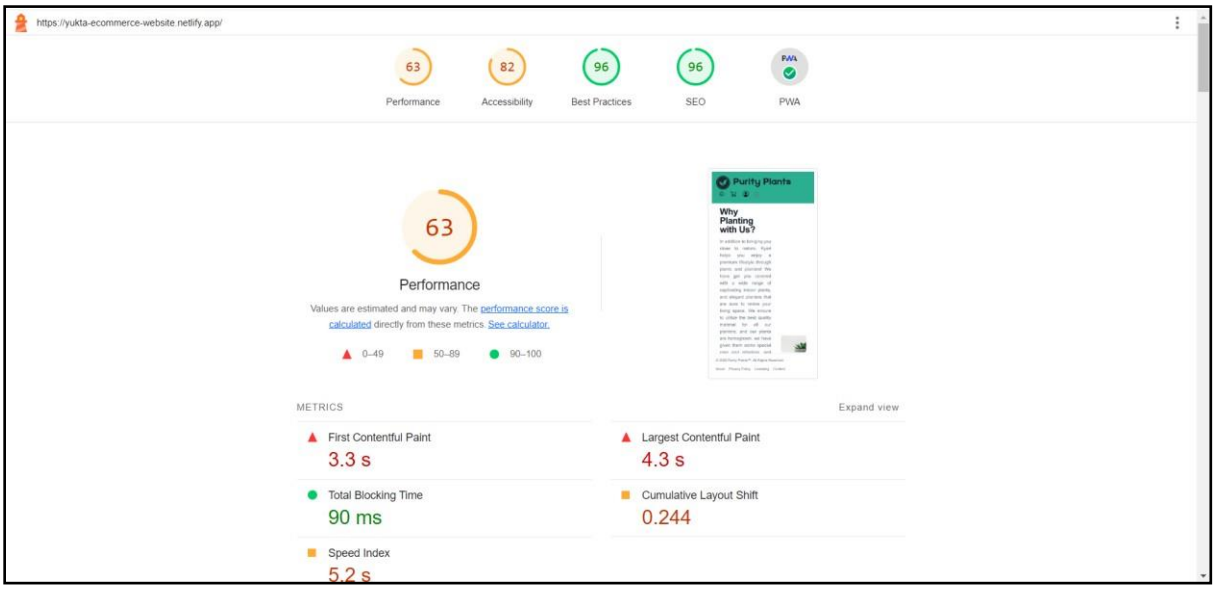
Report Generated:

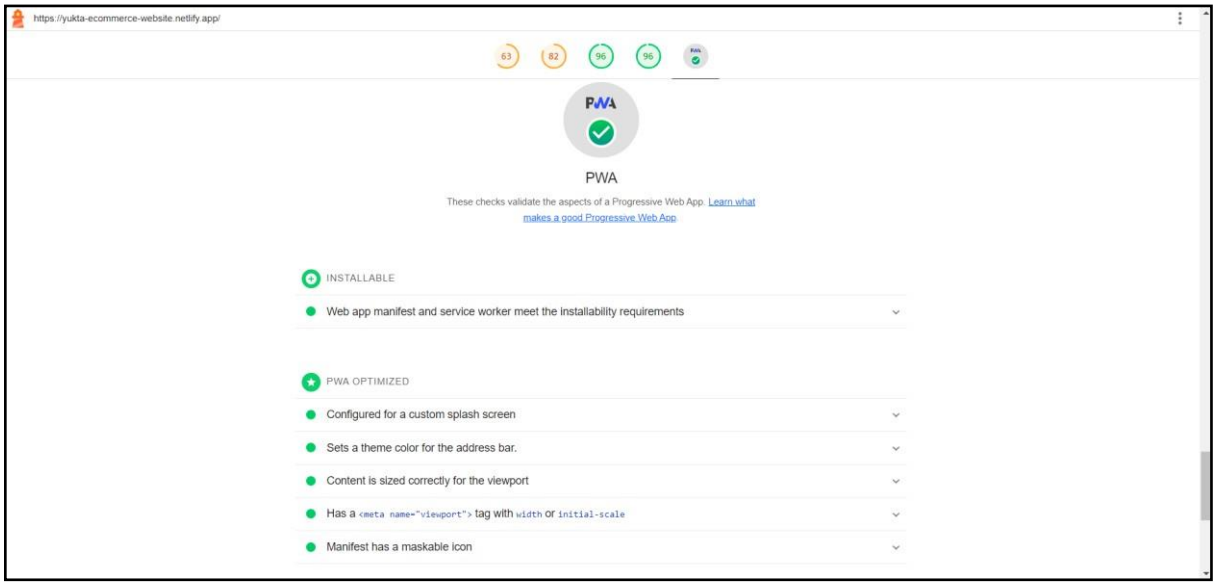




Project Title: Quick Chats/ PWA : Purity Plants **Roll No. 7**







Conclusion: In this experiment, we have successfully used Google Lighthouse PWA Analysis Tool fortesting the PWA functioning.

MAD & PWA Lab
Journal

Experiment No.	Assignment-1
Assignment 1 Questions	1. Flutter Overview:Explain the key features and advantages of using Flutter for mobile app development. Discuss how the Flutter framework differs from traditional approaches and why it has gained popularity in the developer community. 2. Widget Tree and Composition: Describe the concept of the widget tree in Flutter. Explain how widget composition is used to build complex user interfaces. Provide examples of commonly used widgets and their roles in creating a widget tree. 3. State Management in Flutter: Discuss the importance of state management in Flutter applications. Compare and contrast the different state management approaches available in Flutter, such as setState, Provider, and Riverpod. Provide scenarios where each approach is suitable. 4. Firebase Integration in Flutter: Explain the process of integrating Firebase with a Flutter application. Discuss the benefits of using Firebase as a backend solution. Highlight the Firebase services commonly used in Flutter development and provide a brief overview of how data synchronization is achieved.
Roll No.	07
Name	Yukta Bhatia
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	5

Name:- Yukta Bhatia
RollNo:- 07
Class:- D15A

MAD Assignment-1

1) 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

→ Flutter is an open-source UI software development kit created by Google. It is used to develop applications for Android, IOS, Linux, Mac, Windows and the web from a single codebase.

Key Features:

1. Hot Reload:
This feature allows developers to see the results of their changes almost instantly, without losing the current application state. It enhances productivity and makes it easier to experiment with the code.

2. Dart Programming Language:
Flutter uses Dart, a language optimized for fast apps on any platform. Dart allows for both Just In Time (JIT) compilation for the developmed phase (enabling hot reload) and Ahead Of Time (AOT) compilation for the release phase

Advantages:

1. Reduced Development Time
The single codebase and hot reload feature significantly reduce development time and effort as changes can be made in real-time and one app can be deployed on multiple platforms.

2. High Performance
Flutter apps compile to native code, which helps in achieving performance that is comparable to native applications, with smooth animations and transitions.

3. Rich Ecosystem
The flutter ecosystem includes many widgets and tools that covers many common use cases. Additionally, it's easy to use third-party plugins to access native features like camera, GPS, etc.

Differentation from Traditional Approaches

1. Development Efficiency
Traditional app development often involves writing and maintaining multiple codebases, which is more time-consuming and costly. Flutter's single codebase approach streamlines this process.

2. Productivity Tools
Features like hot reload are not typically available in traditional development environments, which can slow down the development process. Flutter's development tools and ecosystem are designed to boost productivity and efficiency.

Popularity Among Developers

1. Rapid Development and Iteration

Features like hot reload and a single codebase for multiple platforms make Flutter an attractive option for startups and businesses looking to quickly bring their products to market.

2. Strong Community and Support
The active and growing developer community around Flutter, combined with strong support from Google, ensures developers have access to a wealth of resources, libraries and tools.

2) 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.

→ The concept of the widget tree in Flutter is crucial for understanding how Flutter constructs and organizes user interface (UIs). Widgets, the core building blocks of a Flutter app's UI are organized into a hierarchical structure known as the widget tree. This hierarchy represents the arrangement and nesting of widgets, enabling Flutter to efficiently render the UI

Widgets in Flutter can be either stateful or stateless, depending on whether they depend on some state for their rendering. Stateful widgets can update their apperance based on changes in their state, while stateless widgets render once and do not change state.

Flutter emphasizes widget composition, where developers build complex UIs by nesting simpler widgets with each other. This approach allows for the creation of highly customizable and flexible interfaces. For example, a simple UI might consist of a Scaffold widget containing a Center widget, which in turn contains a Column widget. The Column widget could hold a Text widget for a title and a RaisedButton for an interactive button. This structure forms a widget tree starting from the Scaffold at the root, down to the Text and RaisedButton as leaves.

Commonly used widgets include:

1. Scaffold: Provides the basic material design layout structure
2. AppBar: A toolbar typically used at the top of an app screen.
3. Text: Displays text with various styling options.
4. Row/Column: Organizes children widgets in a horizontal or vertical manner.
5. Container: A versatile widget for styling, positioning or sizing its child.
6. ListView: Display a scrollable list of widgets.
7. Stack: Layers widgets on top of each other.

Through the widget tree and widget composition, Flutter enables the development of complex, responsive and UIs with a single codebase across multiple platforms.

3) 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.

→ State management in Flutter is a critical aspect of app development, as it deals with managing the data that changes over time in our app and how these changes affect the UI.

Importance of State Management:

1. UI Consistency: Ensures the UI reflects the current state of the app accurately.
2. Code Maintainability: Simplifies the codebase by seperating business logic from UI code, making it easier to maintain and update.
3. Performance Optimization: Efficient state management helps in minimizing unnecessary widget rebuilds, enhancing app performance.
4. Scability: As the app grows, a robust state management system can help manage complexity and interactions across different parts of the app.

Comparison of State Management Approaches:-

Feature	setState	Provider	Riverpod
Description	Built-in Flutter mechanism for managing state.	State management library that allows data sharing across widgets	A more flexible and powerful evolution of Provider.
Suitability	Small apps or widgets with localized state changes.	Medium-to large apps with more complex state management needs.	Suitable for all app sizes, offering high scalability and flexibility.
Scalability	Not suitable for large apps or extensive state management	Good scalability and maintainability for larger applications	Excellent scalability suitable for complex and large applications.
Performance	Can lead to inefficient in larger apps	Optimizes performance by prioritizing unnecessary widgets	High optimized state management, minimizing unnecessary widgets.

Scenarios for each approach:-

SetState: Best for simple or demo apps where you're experimenting with Flutter or when the state is confined to a single widget.
Provider: Suitable for most production apps that have a moderate level of complexity. It's great when the app has a clear separation between business logic and UI and requires efficient state sharing between widgets.
Riverpod: Ideal for complex, large-scale applications with a need for a highly testable, scaleable and flexible state management solution.

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.

→ Integrating firebase with flutter:-

1) Set Up firebase Project:-

- Create a firebase project and register your flutter application in the firebase Console.

2) Configure flutter app:-
- Add firebase dependencies in 'pubspec.yaml' and initialize firebase in the main.dart using 'firebase_initializeApp()

3) Using firebase Services
- Include firebase services in your flutter app like Authentication ('firebase_auth'), firestore ('cloud_firestore') and realtime database ('firebase_database')

4) Authentication Example
- Sign in with Email and password using 'Firebase Auth. Instance. SignInWithEmailAndPassword'

5) Firestore Example:
- Read data from firestore using 'Firebase Firestore. instance. collection('users'). doc('user id'). get();

Benefits of using Firebase as a Backend Solution:-

1) Real-time data Sync:-
Firebase provides real-time data synchronisation, allowing updates to be immediately reflected in all connected clients

2) Scalability:-
Firebase automatically scales to handle the growth of your app, ensuring it remains performant as user number increases.

Firebase achieves real-time data synchronisation through web stockets. When data changes on the server, firebase sends updates to the connected clients.

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.	07
Name	Yukta Bhatia
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4:Understand various PWA frameworks and their requirements LO5: Design and Develop a responsive User Interface by applying PWA Design techniques LO6:Develop and Analyze PWA Features and deploy it over app hosting solutions
Grade:	4

Name:- Yukta Bhatia
Roll No:- 7
Class:- D15A

PWA Assignment-2

1) Define Progressive Web App (PWA) and explain its significance in modern web design development. Discuss the key characteristics that differentiate PWAs from traditional mobile apps.

→ Progressive Web Apps are a type of application software delivered through the web, built using common web technologies including HTML, CSS and JavaScript. It was introduced by Google in 2015 to describe web apps that took advantages of new features supported by modern browsers, including service workers and web app manifests, that enable users to have a user experience akin to that of mobile apps.

Significance in Modern Web Development:

1. Cross-Platform compatibility: PWAs can run on any device with a web browser, reducing the need for platform-specific development
2. Cost-Effective: Developing and maintaining a PWA is generally less expensive than maintaining seperate native apps for multiple platforms.
3. Performance and Enagegement: With service workers, PWAs can load quickly, even on slow network connections, and can send push notifications, improving user enagegement.

Key characteristics differentiating PWA from Traditional Mobile Apps:

1. Installability :- While traditional mobile apps require downloading from app stores, PWAs can be added to a device's home screen directly from the browser, often with minimal

Storage space requirements.

2. Offline Operation: Service workers allow PWAs to function offline or on low-quality networks, a feature traditionally associated with native apps.
3. Update Process: PWAs update automatically as the web content updates, removing the need for users to download updates from an app store.
4. Access to Device Features: While traditional mobile apps have extensive access to device APIs, PWAs have historically had limited access. However, this gap is narrowing as new APIs become available.

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.

→ Responsive Web Design is an approach to web design aimed at creating sites to provide an optimal viewing experience- easy reading and navigation with a minimum of resizing and scrolling across a wide range of devices, from desktop computer monitors to mobile phones.

Importance

1. Seamless User Experience Across Devices: PWAs are intended to work on any device with a standards-compliant browser. Responsive design ensures that a PWA provides a consistent

and engaging user experience, whether it's accessed on a phone, tablet, laptop or desktop which is crucial for user retention and satisfaction.

2. Improved Discoverability and SEO: Because PWAs are accessible via the web, their discoverability is heavily influenced by search engine optimization (SEO). Responsive design is a ranking factor for search engines like Google, which prioritize mobile-friendly websites.
3. Future Scalability: As new devices and screen size continue to emerge, a responsive PWA can adapt to these changes without requiring significant redesigns. This scability ensures that the PWA remains functional and visually appealing across future devices, protecting the investment in its development.

Comparison of responsive, fluid and adaptive web design

Aspect	Responsive Web Design	Fluid Web Design	Adaptive Web Design
Definition	An approach to web design that makes web pages render well on various devices and window or screen sizes	A type of web design where layout fluidly adjusts regardless of screen size using percentage for widths	A type of web design that detects the screen size and loads the appropriate layout for it.

Images and Media	Images and media can be flexible or fixed. Can use CSS and JS to adjust images/ media dynamically	Images can scale based on the fluid grid, potentially using CSS to ensure they remain within their container without distortion	Images and media are chosen based on detected layout size, potentially loading different sizes for different screens.
Techniques Used	Media queries, flexible grid layouts, flexible images and media.	Fluid grids, percentage-based sizing.	Media queries, multiple fixed-width layouts
User Experience	Consistently optimal across different devices adjusting layout and content dynamically	Provides a continuous adjustment of layout proportions, offering a more tailored fit to various screen sizes.	Offers a customized experience at specific breakpoints but may not be as seamless between those breakpoints.

3) Describe the lifecycle of Services Workers, including registration, installation, and activation phases

→ Service Workers play a crucial role in the functionality of Progressive Web Apps by acting as a proxy between the web application and the network. The lifecycle of Service worker includes several phases: registration, installation activation and others like fetching and messaging.

1. Registration

A service worker is basically a Javascript file. One thing that differentiate a service worker file from a normal Javascript file, is that a service worker, we must register that it as background process. This is the first phase of the cycle.

```
if ('serviceWorker' in navigator)
    navigator.serviceWorker.register('sw.js')
    .then(function (registration) {
        console.log('Service Worker registered!');
    })
    .catch(function(err) {
        console.log('Registration failed!');
    })
}
```

2. Installation

This is the second phase of the cycle. This phase is an opportunity to cache static assets (HTML, CSS, JavaScript files, images) required for offline support. The service worker's 'install' event is where these assets are usually cached. This phase is considered successful if all the files specified for caching are downloaded and cached successfully. If any file fails to download and cache, the installation fails and the service work will be discarded.

3. Activation.

The activation phase is critical for managing old caches. When a new service worker is activated, it can clear old caches that are no longer needed. A service worker will only be active in any of these cases.

1. If there is no service worker currently active.
2. If the self.skipWaiting() is called in the install event handler of the service worker script.
3. If the user refreshes the page.

After activation, the service worker is ready to handle other events like fetch, push and sync.

4) Explain the use of IndexedDB in the Service Worker for data storage.:

→ Uses of IndexedDB in Service Workers :-

1. Offline Data storage: It is most common use of IndexedDB in service workers is for storing app data for offline use. When a user accesses the app while online, the service worker can fetch and cache dynamic content, such as user-generated content. This ensures that the app can still function and display this data when the user is offline.
2. Performance Optimization: By caching data in IndexedDB, service workers can reduce reliance on network requests, leading to faster load times and a smoother user experience. Data can be retrieved from the local IndexedDB store rather than needing to fetch it from a server each time, which is especially beneficial for users with slow or unstable internet connections.
3. Background Syncing: Service workers can use IndexedDB in combination with the Background Sync API to store user actions performed while offline and synchronize them with a server once the user is back online. This ensure that no user data is lost, even if actions are taken while the app is offline.
4. Push Notifications Handling: For PWAs that use push notifications, IndexedDB can store payloads received from push notifications when the application is not active or when the user decides to interact with a notification at a later time.

5. State Management: Service Workers can use IndexedDB to persist application state across sessions. Since service workers are terminated when not in use and restarted as needed, using IndexedDB can help maintain state information that needs to persist beyond the lifecycle of the service worker.