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 **Anurag Mahadev Gaiwal** of **D15A** semester **VI,** have successfully completed necessary experiments in the **MAD & PWA Lab** under my supervision in **VES Institute of Technology** during the academic year **2023-2024.**

Lab Assistant

Subject Teacher

Mrs. Kajal Joseph

Principal

Head of Department

Dr. Mrs. Shalu Chopra

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

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

Faculty Incharge : Mrs. Kajal Joseph.

Lab Teachers **:** Mrs. Kajal Jewani.

Email **:** kajal.jewani@ves.ac.in

Programme Outcomes: The graduate will be able to:

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

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

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

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

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

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

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

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

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

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

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

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

Program specific Outcomes

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

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

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

MAD & PWA Lab

Journal

Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	17
Name	Anurag Mahadev Gaiwal
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	15

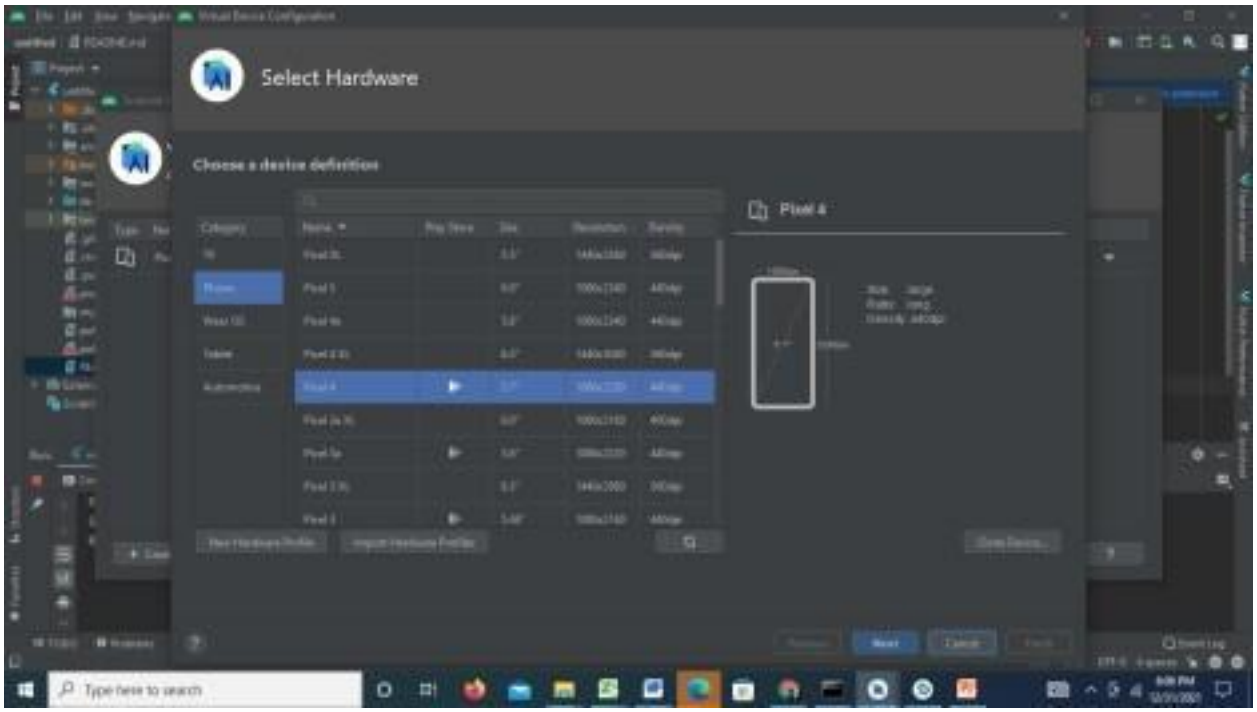
MPL Lab Exp 1

Name: Anurag Gaiwal **Roll No. 17**
Div: D15A **Batch: A**

Experiment 1: Create a 'Hello World App' using Flutter

Step 1 : Create the app

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

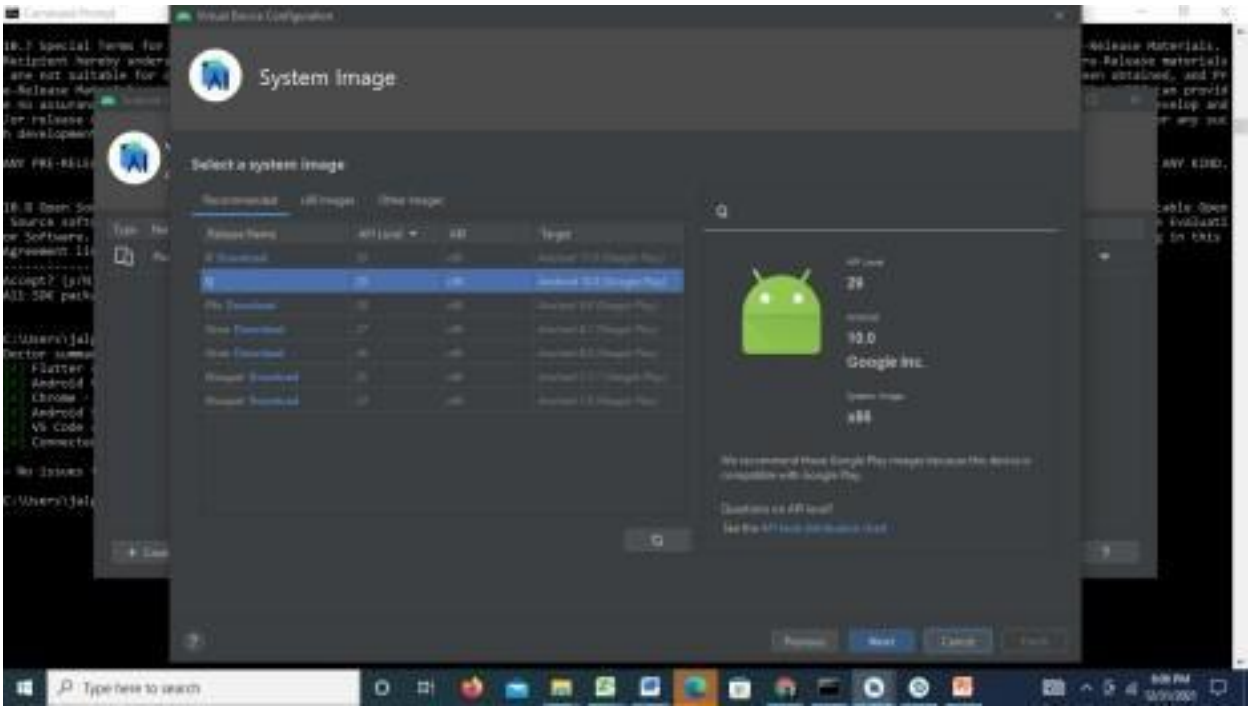


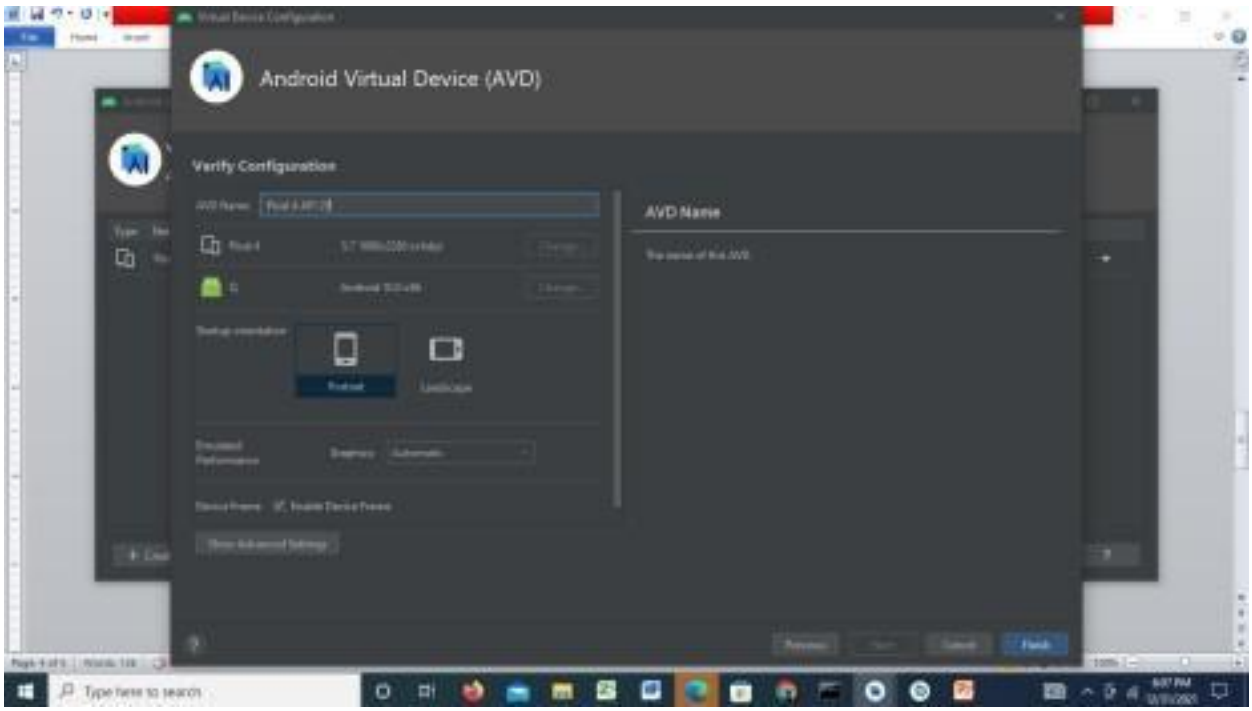
6. Wait for Android Studio to install the SDK and create the project.

Step 2: Run the app

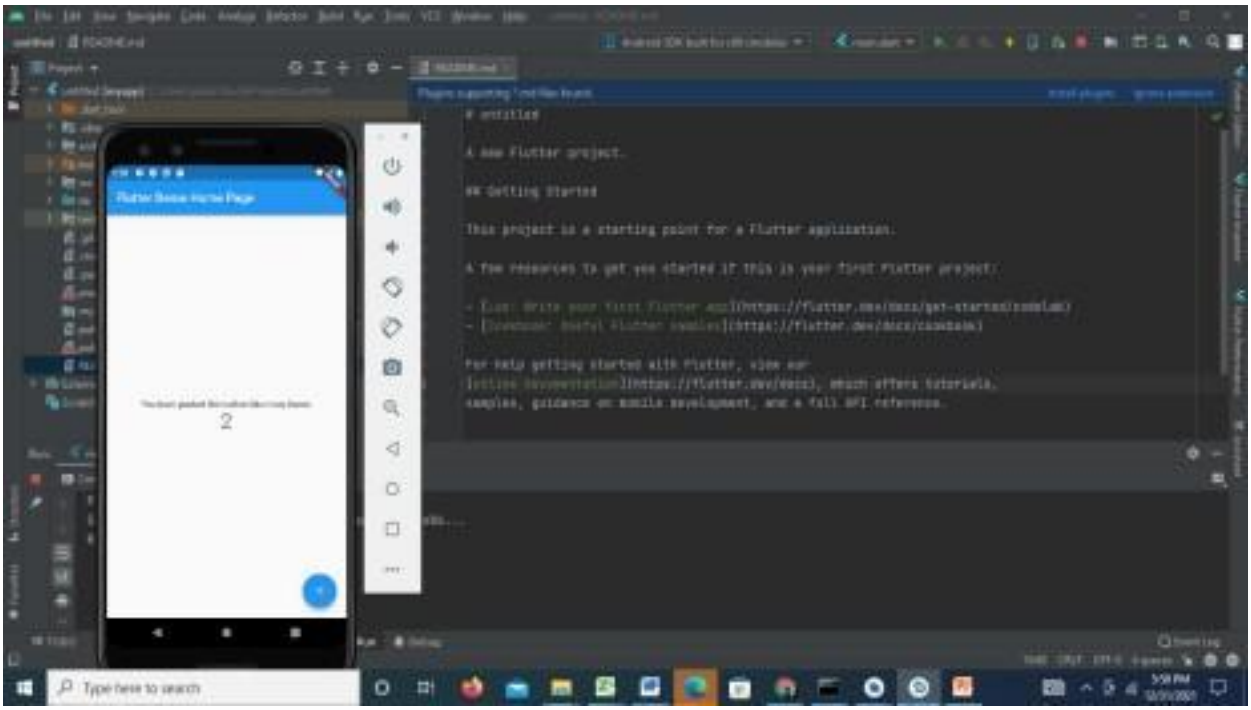
1. Locate the main Android Studio toolbar:

2. In the **target selector**, select an Android device for running the app. If none are listed as available, select **Tools > AVD Manager** and create one there.





3. Click the run icon in the toolbar, or invoke the menu item **Run > Run**.



Step 3 : Creating Hello world app

1. Replace the contents of lib/main.dart. - Delete all of the code from **lib/main.dart**. 2. Replace with the following code, which displays “Hello World” in the center of the screen.

```
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('Anurag Gaiwal From D15A 17'),
        ),
      ),
    );
  }
}
```

3. Run the app by selecting Run> Run „main.dart“ and see the output in emulator device.

MAD & PWA Lab
Journal

Experiment No.	02
Experimen t Title.	To design Flutter UI by including common widgets.
Roll No.	17
Name	Anurag Mahadev Gaiwal
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	11

MPL Lab Exp 2

Name: Anurag Gaiwal Roll No. 17
Div: D15A Batch: A

Aim: To design Flutter UI by including common widgets.

Theory:

We can split the Flutter widget into two categories:

1. Visible (Output and Input)
2. Invisible (Layout and Control)

1. Visible widget

The visible widgets are related to the user input and output data. Some of the important types of
this widget are:

1. Text

A Text widget holds some text to display on the screen. We can align the text widget by using
textAlign property, and style property allow the customization of Text that includes font, font
weight, font style, letter spacing, color, and many more.

2. Button

This widget allows you to perform some action on click. Flutter does not allow you to use the
Button widget directly; instead, it uses a type of buttons like a FlatButton and a RaisedButton.

3. Image

This widget holds the image which can fetch it from multiple sources like from the asset
folder or directly from the URL. It provides many constructors for loading image, which are given below:

o Image: It is a generic image loader, which is used by ImageProvider.

o asset: It load image from your project asset folder.

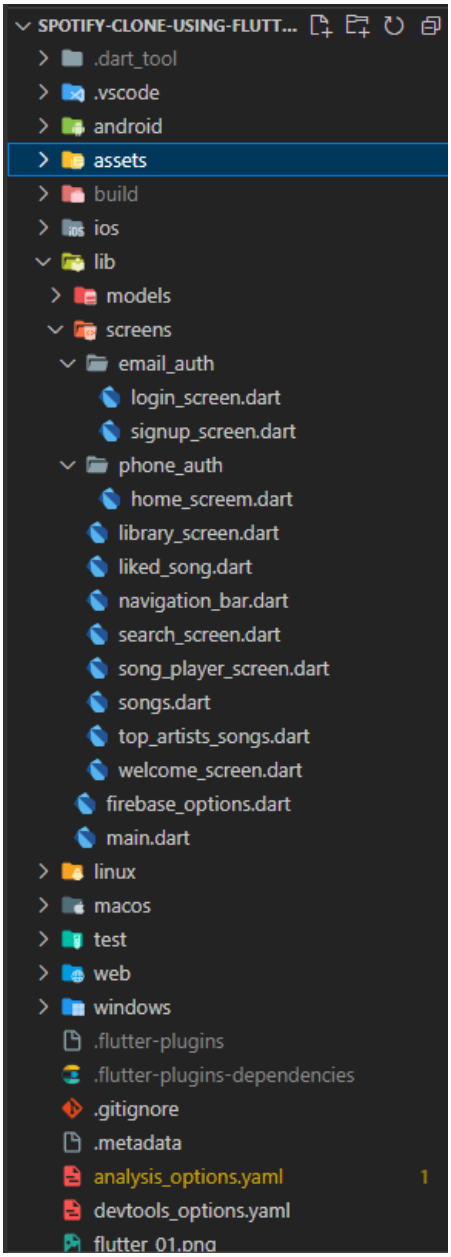
o file: It loads images from the system folder.

o memory: It load image from memory.

o network: It loads images from the network.

To add an image in the project, you need first to create an assets folder where you keep your images and then add the below line in pubspec.yaml file.

File structure:



Code:

main.dart

```
import
'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';
import
'package:jio_saavn_auth/firebase_options.dart';
import 'package:jio_saavn_auth/models/song.dart';
import
'package:jio_saavn_auth/screens/welcome_screen.dart';
import 'package:provider/provider.dart';

Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  // runApp(
  // ChangeNotifierProvider(
  //    create: (context) => LikedSongsModel(),
  //    child: MyApp(),
  // ),
  // );
  runApp(
    MultiProvider(
      providers: [
        ChangeNotifierProvider(create: (context) => LikedSongsModel()),
        // Add other providers if needed
      ],
      child: MyApp(),
    ),
  );
}
```

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
```

```
    return ChangeNotifierProvider(
      create: (context) => LikedSongsModel(),
      child: MaterialApp(
        title: 'Spotify',
        home: WelcomePage(), // Use your main screen here
      ),
    );
  }
}
```

Home_screen.dart:

```
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import
'package:jio_saavn_auth/screens/email_auth/login_screen.dart';
import "package:jio_saavn_auth/screens/library_screen.dart";
import 'package:jio_saavn_auth/screens/navigation_bar.dart';
import 'package:jio_saavn_auth/screens/search_screen.dart';
import
'package:jio_saavn_auth/screens/song_player_screen.dart';
import 'package:jio_saavn_auth/screens/top_artists_songs.dart';

class Song {
  final String name;
  final String path;
  final String imagePath;

  Song({
    required this.name,
    required this.path,
    required this.imagePath,
  });
}
```

```
List<Song> recommendedSongs = [
  Song(
```

```
    name: 'Alone',
    path:
    'assets/songs/song1.mp3',
    imagePath:
    'assets/song1.png'),
  Song(
    name: 'Kesariya',
    path:
    'assets/songs/song2.mp3',
    imagePath:
    'assets/song2.png'),
  Song(
    name: 'We own it',
    path:
    'assets/songs/song3.mp3',
    imagePath:
    'assets/song3.png'),
];

// Add this function outside the HomeScreen class
Future<List<String>> fetchTopSongsForArtist(String artistName) async {
  // Simulate fetching data, replace this with actual data fetching
  logic await Future.delayed(Duration(seconds: 2)); // Simulating a
  delay
  // Hardcoded top songs for each artist
  Map<String, List<String>> artistTopSongs = {
    'Without Me': ['song1.mp3', 'song2.mp3', 'song3.mp3'],
    'Heeriye': ['song4.mp3', 'song5.mp3', 'song6.mp3'],
    'Calm Down': ['song7.mp3', 'song8.mp3', 'song9.mp3'],
    // Add more artists and their top songs as needed
  };

  return artistTopSongs[artistName] ?? [];
}
```

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

  @override
  State<HomeScreen> createState() => _HomeScreenState();
}
```

```
class _HomeScreenState extends State<HomeScreen>
 { int _selectedIndex = 0;

 void logout() async {
  await FirebaseAuth.instance.signOut();

  Navigator.popUntil(context, (route) => route.isFirst);
  Navigator.pushReplacement(
   context,
   MaterialPageRoute(builder: (context) => LoginScreen()),
  );
 }

 Color _getColor(int index) {
  return _selectedIndex == index
     ? Colors.white
     : const Color.fromARGB(255, 128, 128, 128);
 }

 Widget _buildSuggestedItem(BuildContext context, Song song) {
  return GestureDetector(
   onTap: () {
    Navigator.push(
    context,
    MaterialPageRoute(
       builder: (context) => SongPlayerScreen(
        songPaths: [song.path],
        songName: song.name,
        imagePath:
        song.imagePath,
       ),
      ),
    );
```

```
    },
```

```
    child: Container(
      width: 120,
      margin: EdgeInsets.only(right: 5),
      child: Column(
        children: [
          Image.asset(
          song.imagePath,
          height: 100,
            width: 100,
            fit: BoxFit.cover,
          ),
          SizedBox(height: 5),
          Text(
            song.name,
            style: TextStyle(color: Color.fromARGB(255, 255, 255, 255)),
          ),
        ],
      ),
    ),
  );
}

Widget _buildImageRow(List<Song> songs) {
  return SizedBox(
    height: 150,
    child: ListView.builder(
      scrollDirection: Axis.horizontal,
      itemCount: songs.length,
      itemBuilder: (BuildContext context, int index) {
        return GestureDetector(
          onTap: () async {
            print('Tapped on ${songs[index].name}');
```

```
// Fetch the top songs for the selected artist
```

```
            List<String> topSongs =
                await fetchTopSongsForArtist(songs[index].name);
            print('Top songs for ${songs[index].name}: $topSongs');

            // Navigate to TopArtistSongsScreen with the fetched top songs
            Navigator.push(
              context,
              MaterialPageRoute(
                builder: (context) => TopArtistSongsScreen(
                  artistName: songs[index].name,
                  topSongs: topSongs,
                ),
              ),
            );
          },
          child: _buildSuggestedItem(context, songs[index]),
        );
      },
    ),
  );
}

@override
Widget build(BuildContext context) {
  List<Song> topArtists = [
    Song(
        name: 'Without Me',
        path: 'assets/songs/eminem.mp3',
        imagePath: 'assets/travis.png'),
    Song(
        name: 'Heeriye',
        path:
        'assets/songs/arjit.mp3',
        imagePath:
```

```
    'assets/sonu.png'),
Song(
```

```
    name: 'Calm Down',
    path:
    'assets/songs/selena.mp3',
    imagePath: 'assets/drake.png'),
];

List<Song> newReleases = [
 Song(
    name: 'Supriya sule - TRS
    hindi', path:
    'assets/songs/new2.mp3',
    imagePath:
 'assets/trsh_supriya_sule.jpg'), Song(
    name: 'Satranga',
    path: 'assets/songs/new3.mp3',
    imagePath: 'assets/satranga.png'),
 Song(
    name: 'Duniya jala denge',
    path:
    'assets/songs/new3.mp3',
    imagePath: 'assets/sariduniya.png'),
];

List<Song> jioSaavnPicks =
 [ Song(
    name: 'Challeya',
    path: 'assets/songs/pick1.mp3',
    imagePath:
    'assets/challeya.png'),
 Song(
    name: 'Zinda Banda',
    path: 'assets/songs/pick2.mp3',
    imagePath:
```

```
      'assets/zindabanda.png'),
  Song(
      name: 'Jawan Title Track',
      path:
      'assets/songs/pick3.mp3',
      imagePath: 'assets/jawan.png'),
];
```

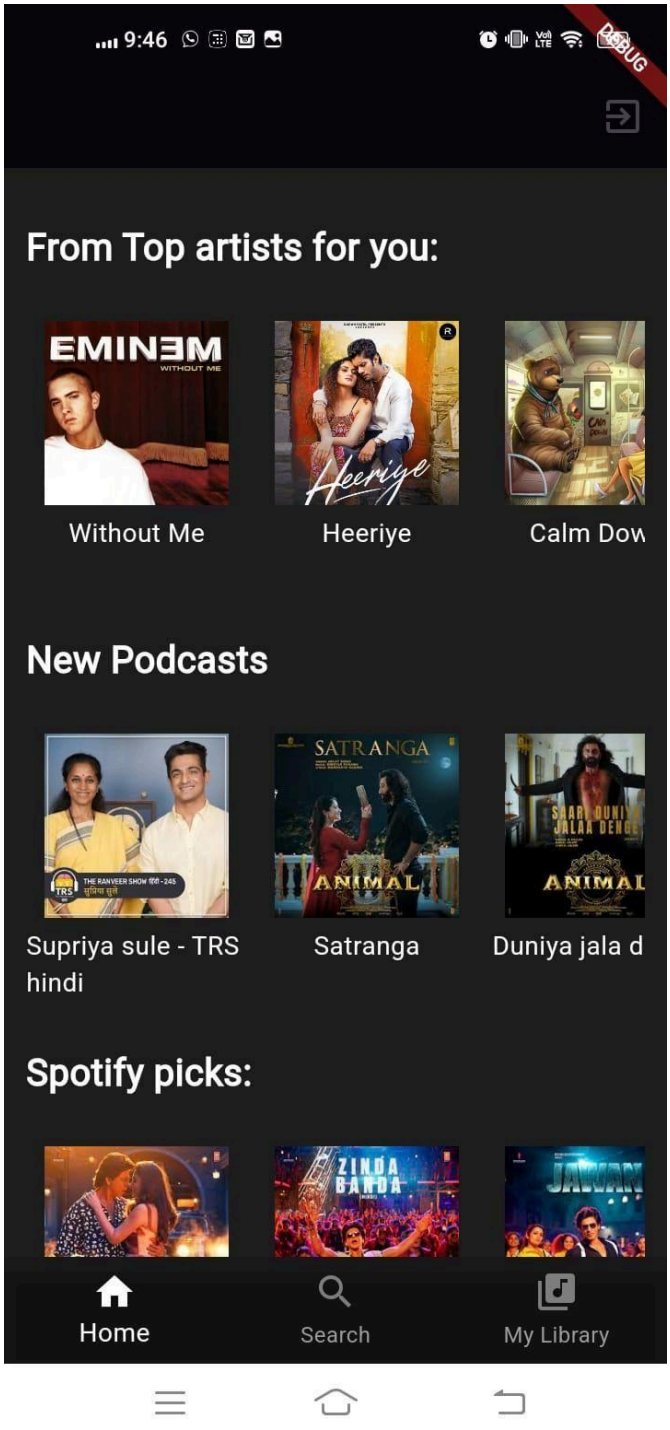
```
return Scaffold(
  appBar: AppBar(
    automaticallyImplyLeading: false,
    backgroundColor: Colors.black,
    actions: [
      IconButton(
        onPressed: () {
          logout();
        },
        icon: Icon(Icons.exit_to_app),
      ),
    ],
  ),
  backgroundColor: Color.fromARGB(255, 30, 30, 30),
  body: SingleChildScrollView(
    child: Padding(
      padding: const EdgeInsets.all(12.0),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Center(
            child: Align(
              alignment: Alignment.center,
              child: Image.asset(
              'assets/jiosaavn_name.png',
              height: 150,
                width: 200,
              ),
            ),
          ),
          SizedBox(height: 10),
          SizedBox(height: 20),
          Text(
```

```
  'Recommended Songs for
  you:', style: TextStyle(
    fontSize: 20,
    fontWeight: FontWeight.bold,
    color: Color.fromARGB(255, 255, 255, 255),
  ),
),
SizedBox(height: 25),
_buildImageRow(recommendedSongs),
SizedBox(height: 20),
Text(
  'From Top artists for
  you:', style: TextStyle(
    fontSize: 20,
    fontWeight: FontWeight.bold,
    color: Color.fromARGB(255, 255, 255, 255),
  ),
),
SizedBox(height: 25),
_buildImageRow(topArtists),
SizedBox(height: 20),
Text(
  'New
  Podcasts',
  style:  TextStyle(
  fontSize: 20,
    fontWeight: FontWeight.bold,
    color: Color.fromARGB(255, 255, 255, 255),
  ),
),
SizedBox(height: 25),
_buildImageRow(newReleases),
SizedBox(height: 20),
Text(
```

'Spotify picks:',

```
                style: TextStyle(
                  fontSize: 20,
                  fontWeight: FontWeight.bold,
                  color: Color.fromARGB(255, 255, 255, 255),
                ),
              ),
              SizedBox(height: 25),
              _buildImageRow( jioSaavnPicks),
            ],
          ),
        ),
      ),
      bottomNavigationBar: CustomBottomNavigationBar(
        selectedIndex: _selectedIndex,
        onItemTapped: (index) {
          setState(() {
            _selectedIndex = index;
          });
          if (_selectedIndex == 1) {
            Navigator.push(
            context,
              MaterialPageRoute(builder: (context) => SearchScreen()),
            );
          } else if (_selectedIndex == 2) {
            Navigator.push(
              context,
              MaterialPageRoute(builder: (context) => MyLibrary()),
            );
          }
        },
      ),
    );
  }
}
```

Home screen:


9:46
DEBUG
From Top artists for you:
EMINEM
WITHOUT ME
Without Me
Heeriye
Heeriye
Calm Dow
New Podcasts
THE RANVEER SHOW
Supriya sule - TRS hindi
SATRANGA
ANIMAL
Satranga
ANIMAL
Duniya jala d
Spotify picks:
ZINDA BANDA
JAWAN
Home
Search
My Library

Conclusion:

Flutter’s widget architecture offers great flexibility for building complex UIs.

Understanding key widgets and concepts is essential for effective Flutter development.

MAD & PWA Lab
Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	17
Name	Anurag Mahadev Gaiwal
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	11

MPL Lab Exp 3

Name: Anurag Gaiwal Roll No. 17
Div: D15A Batch: A

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

Theory:

A flutter app when built has both assets (resources) and code. Assets are available and deployed during runtime. The asset is a file that can include static data, configuration files,
icons, and images. The Flutter app supports many image formats, such as JPEG, WebP, PNG,
GIF, animated WebP/GIF, BMP, and WBMP.

Steps to Add an Image:

Step 1. Create a new folder

- It should be in the root of your flutter project. You can name it whatever you want, but assets are preferred.
- If you want to add other assets to your app, like fonts, it is preferred to make another subfolder named images.

Step 2. Now you can copy your image to images sub-folder. The path should look like assets/images/yourImage. Before adding images also check the above-mentioned supported
image formats.

Step 3. Register the assets folder in pubspec.yaml file and update it.

To add images, write the following code:

```
flutter:
assets:
- assets/images/yourFirstImage.jpg
- assets/image/yourSecondImage.jpg
```

If you want to include all the images of the assets folder then add this:

```
flutter:
assets:
- assets/images/
```

Step 4. Insert the image code in the file, where you want to add the image.

Step 5. Now you can save all the files and run the app, you will find the output as shown below.

Icon class in Flutter is used to show specific icons in our app. Instead of creating an

image for our icon, we can simply use the Icon class for inserting an icon in our app. For using
this class you must ensure that you have set uses-material-design: true in the pubsec.yml file of
your object.

Properties:
● color: It is used to set the color of the icon
● size: It is used to resize the Icon
● semanticLable: It comes in play while using the app in accessibility mode (ie, voice-over)
● textDirection: It is used for rendering Icon
Note: The semanticLabel are not visible in the UI.

Code in home_screen.dart:

```
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import
'package:jio_saavn_auth/screens/email_auth/login_screen.dart';
import "package:jio_saavn_auth/screens/library_screen.dart";
import 'package:jio_saavn_auth/screens/navigation_bar.dart';
import 'package:jio_saavn_auth/screens/search_screen.dart';
import
'package:jio_saavn_auth/screens/song_player_screen.dart';
import 'package:jio_saavn_auth/screens/top_artists_songs.dart';

class Song {
  final String name;
  final String path;
  final String imagePath;

  Song({
    required this.name,
    required this.path,
    required this.imagePath,
  });
```

```
}
```

```
List<Song> recommendedSongs = [
  Song(
     name: 'Alone',
     path:
     'assets/songs/song1.mp3',
     imagePath:
     'assets/song1.png'),
  Song(
     name: 'Kesariya',
     path:
     'assets/songs/song2.mp3',
     imagePath:
     'assets/song2.png'),
  Song(
     name: 'We own it',
     path:
     'assets/songs/song3.mp3',
     imagePath:
     'assets/song3.png'),
];

// Add this function outside the HomeScreen class
Future<List<String>> fetchTopSongsForArtist(String artistName) async {
  // Simulate fetching data, replace this with actual data fetching
  logic await Future.delayed(Duration(seconds: 2)); // Simulating a
  delay
  // Hardcoded top songs for each artist
  Map<String, List<String>> artistTopSongs = {
    'Without Me': ['song1.mp3', 'song2.mp3', 'song3.mp3'],
    'Heeriye': ['song4.mp3', 'song5.mp3', 'song6.mp3'],
    'Calm Down': ['song7.mp3', 'song8.mp3', 'song9.mp3'],
    // Add more artists and their top songs as needed
```

```
  };

  return artistTopSongs[artistName] ?? [];
}

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

```
  @override
  State<HomeScreen> createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen>
  { int _selectedIndex = 0;

  void logout() async {
    await FirebaseAuth.instance.signOut();

    Navigator.popUntil(context, (route) => route.isFirst);
    Navigator.pushReplacement(
      context,
      MaterialPageRoute(builder: (context) => LoginScreen()),
    );
  }

  Color _getColor(int index) {
    return _selectedIndex == index
        ? Colors.white
        : const Color.fromARGB(255, 128, 128, 128);
  }

  Widget _buildSuggestedItem(BuildContext context, Song song) {
    return GestureDetector(
      onTap: () {
        Navigator.push(
        context,
        MaterialPageRoute(
            builder: (context) => SongPlayerScreen(
              songPaths: [song.path],
              songName: song.name,
              imagePath:
              song.imagePath,
```

),

```
        ),
      );
    },
    child: Container(
      width: 120,
      margin: EdgeInsets.only(right: 5),
      child: Column(
        children: [
          Image.asset(
          song.imagePath,
          height: 100,
            width: 100,
            fit: BoxFit.cover,
          ),
          SizedBox(height: 5),
          Text(
            song.name,
            style: TextStyle(color: Color.fromARGB(255, 255, 255, 255)),
          ),
        ],
      ),
    ),
  );
}

Widget _buildImageRow(List<Song> songs) {
  return SizedBox(
    height: 150,
    child: ListView.builder(
      scrollDirection: Axis.horizontal,
      itemCount: songs.length,
      itemBuilder: (BuildContext context, int index) {
        return GestureDetector(
          onTap: () async {
```

```
            print('Tapped on ${songs[index].name}');

            // Fetch the top songs for the selected
            artist List<String> topSongs =
                await fetchTopSongsForArtist(songs[index].name);
            print('Top songs for ${songs[index].name}: $topSongs');

            // Navigate to TopArtistSongsScreen with the fetched top songs
            Navigator.push(
              context,
              MaterialPageRoute(
                builder: (context) => TopArtistSongsScreen(
                  artistName: songs[index].name,
                  topSongs: topSongs,
                ),
              ),
            );
          },
          child: _buildSuggestedItem(context, songs[index]),
        );
      },
    ),
  );
}

@override
Widget build(BuildContext context) {
  List<Song> topArtists = [
    Song(
        name: 'Without Me',
        path: 'assets/songs/eminem.mp3',
        imagePath: 'assets/travis.png'),
    Song(
        name: 'Heeriye',
```

```
      path:
      'assets/songs/arjit.mp3',
      imagePath:
      'assets/sonu.png'),
  Song(
      name: 'Calm Down',
      path:
      'assets/songs/selena.mp3',
      imagePath: 'assets/drake.png'),
];

List<Song> newReleases = [
  Song(
      name: 'Supriya sule - TRS
      hindi', path:
      'assets/songs/new2.mp3',
      imagePath:
   'assets/trsh_supriya_sule.jpg'), Song(
      name: 'Satranga',
      path: 'assets/songs/new3.mp3',
      imagePath: 'assets/satranga.png'),
  Song(
      name: 'Duniya jala denge',
      path:
      'assets/songs/new3.mp3',
      imagePath: 'assets/sariduniya.png'),
];

List<Song> jioSaavnPicks =
  [ Song(
      name: 'Challeya',
      path: 'assets/songs/pick1.mp3',
      imagePath:
```

```
    'assets/challeya.png'),
Song(
    name: 'Zinda Banda',
    path: 'assets/songs/pick2.mp3',
    imagePath:
    'assets/zindabanda.png'),
Song(
    name: 'Jawan Title Track',
```

```
      path:
      'assets/songs/pick3.mp3',
      imagePath:
      'assets/jawan.png'),
];

return Scaffold(
  appBar: AppBar(
    automaticallyImplyLeading: false,
    backgroundColor: Colors.black,
    actions: [
      IconButton(
        onPressed: () {
          logout();
        },
        icon: Icon(Icons.exit_to_app),
      ),
    ],
  ),
  backgroundColor: Color.fromARGB(255, 30, 30, 30),
  body: SingleChildScrollView(
    child: Padding(
      padding: const EdgeInsets.all(12.0),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Center(
            child: Align(
              alignment: Alignment.center,
              child: Image.asset(
              'assets/jiosaavn_name.png',
              height: 150,
                width: 200,
```

```
      ),
    ),
  ),
```

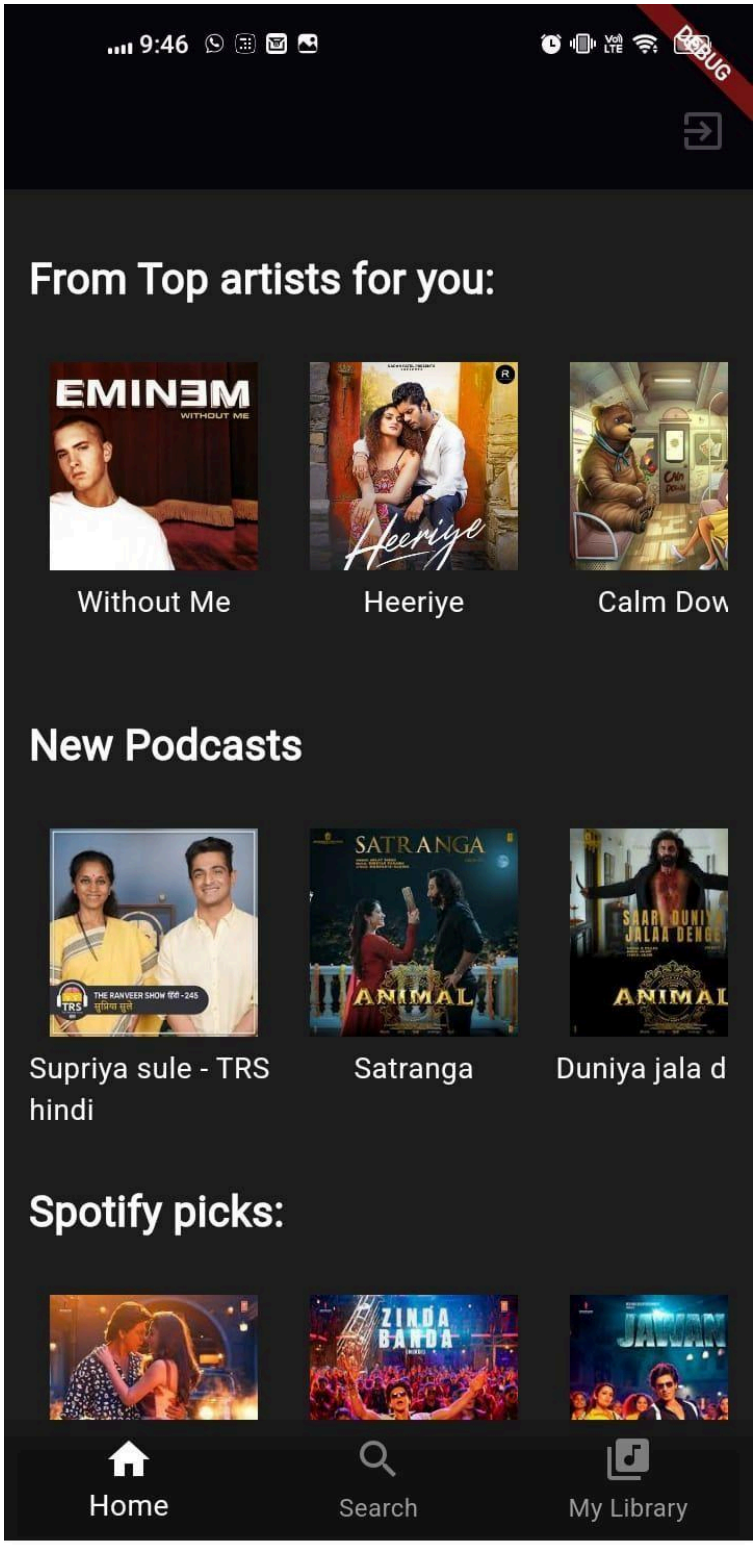
```
SizedBox(height: 10),
SizedBox(height: 20),
Text(
  'Recommended Songs for
  you:', style: TextStyle(
    fontSize: 20,
    fontWeight: FontWeight.bold,
    color: Color.fromARGB(255, 255, 255, 255),
  ),
),
SizedBox(height: 25),
_buildImageRow(recommendedSongs),
SizedBox(height: 20),
Text(
  'From Top artists for
  you:', style: TextStyle(
    fontSize: 20,
    fontWeight: FontWeight.bold,
    color: Color.fromARGB(255, 255, 255, 255),
  ),
),
SizedBox(height: 25),
_buildImageRow(topArtists),
SizedBox(height: 20),
Text(
  'New
  Podcasts',
  style:  TextStyle(
  fontSize: 20,
    fontWeight: FontWeight.bold,
    color: Color.fromARGB(255, 255, 255, 255),
  ),
),
SizedBox(height: 25),
```

```
_buildImageRow(newReleases),
```

```
        SizedBox(height: 20),
        Text(
          'Spotify picks:',
          style: TextStyle(
          fontSize: 20,
            fontWeight: FontWeight.bold,
            color: Color.fromARGB(255, 255, 255, 255),
          ),
        ),
        SizedBox(height: 25),
        _buildImageRow( jioSaavnPicks),
      ],
    ),
  ),
),
bottomNavigationBar: CustomBottomNavigationBar(
  selectedIndex: _selectedIndex,
  onItemTapped: (index) {
    setState(() {
      _selectedIndex = index;
    });
    if (_selectedIndex == 1) {
      Navigator.push(
      context,
        MaterialPageRoute(builder: (context) => SearchScreen()),
      );
    } else if (_selectedIndex == 2) {
      Navigator.push(
        context,
        MaterialPageRoute(builder: (context) => MyLibrary()),
      );
    }
  },
),
```

```
    );
  }
}
```

Output:


9:46
DEBUG
From Top artists for you:
EMINEM
WITHOUT ME
Heeriye
Without Me
Heeriye
Calm Dow
New Podcasts
THE RANVEER SHOW
SATRANGA
ANIMAL
ANIMAL
Supriya sule - TRS hindi
Satranga
Duniya jala d
Spotify picks:
ZINDA BANDA
Home
Search
My Library

MAD & PWA Lab
Journal

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget
Roll No.	17
Name	Anurag Mahadev Gaiwal
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	12

MPL Lab Exp 4

Name: Anurag Gaiwal Roll No. 17
Div: D15A Batch: A

Aim:
The aim of this experiment is to create an interactive form using Flutter's form widget. The form will include various form fields such as text fields, dropdown menus, and checkboxes, along with validation logic. Additionally, a button will be added to validate and submit the form data.

Theory:
In Flutter, forms are created using the `Form` widget, which provides a framework for form validation, saving form data, and handling form submission. Here's a brief overview of the key concepts used in this experiment:

Form Widget:
The `Form` widget in Flutter facilitates the creation of interactive user input forms. It manages state, validation, and submission of form data efficiently.

Email Field (TextFormField):
Utilized for email input, `TextFormField` provides validation features ensuring basic email format compliance. The `onSaved` property captures and stores entered email addresses.

Password Field (TextFormField with obscureText):
For secure password input, `TextFormField` with `obscureText` conceals characters. Validation logic ensures password strength, enhancing security and user experience.

By understanding these concepts and implementing them effectively, we can create an interactive form in Flutter that provides a smooth user experience with validation and submission functionality.

Code in main.dart:

```
// main.dart
import
'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';
import
'package:jio_saavn_auth/firebase_options.dart';
import 'package:jio_saavn_auth/models/song.dart';
import
'package:jio_saavn_auth/screens/welcome_screen.dart';
```

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

```
Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  // runApp(
  // ChangeNotifierProvider(
  //    create: (context) => LikedSongsModel(),
  //    child: MyApp(),
  // ),
  // );
  runApp(
    MultiProvider(
      providers: [
        ChangeNotifierProvider(create: (context) => LikedSongsModel()),
        // Add other providers if needed
      ],
      child: MyApp(),
    ),
  );
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return ChangeNotifierProvider(
      create: (context) => LikedSongsModel(),
      child: MaterialApp(
        title: 'Spotify',
        home: WelcomePage(), // Use your main screen here
      ),
    );
  }
```

```
}
```

Code in signup_screen.dart:

```
import 'dart:developer';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import
'package:jio_saavn_auth/screens/phone_auth/home_screem.dart';
import 'login_screen.dart';

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

  @override
  State<SignUpScreen> createState() => _SignUpScreenState();
}

class _SignUpScreenState extends State<SignUpScreen> {
  TextEditingController emailController = TextEditingController();
  TextEditingController passwordController = TextEditingController();
  TextEditingController cPasswordController = TextEditingController();

  String errorMessage = ''; // Added to manage error message

  void createAccount() async {
    String email = emailController.text.trim();
    String password = passwordController.text.trim();
    String cPassword = cPasswordController.text.trim();

    if (email == "" || password == "" || cPassword == "")
      { setState(() {
        errorMessage = "Please enter all the details!";
```

```
  });
} else if (password != cPassword) {
```

```
    setState(() {
      errorMessage = "Passwords do not match!";
    });
  } else {
    try {
      UserCredential userCredential = await FirebaseAuth.instance
          .createUserWithEmailAndPassword(email: email, password: password);
      if (userCredential.user != null) {
        // User created successfully, navigate to
        HomeScreen Navigator.push(
          context,
          MaterialPageRoute(builder: (context) => HomeScreen()),
        );
      }
      log("User created!");
    } on FirebaseAuthException catch (ex) {
      log(ex.code.toString());
      setState(() {
        errorMessage = "Error creating account. Please try again.";
      });
    }
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: const Color.fromARGB(255, 255, 255, 255),
    body: SafeArea(
      child: SingleChildScrollView(
        child: Column(
          children: [
            Padding(
              padding: const EdgeInsets.all(25.0),
```

```
child: Column(
 mainAxisAlignment: MainAxisAlignment.center,
 children: [
  Image.asset(
   'assets/jiosaavn_name.png'
   , height: 150,
   width: 200,
  ),
  SizedBox(height: 20),
  Text(
   'Welcome!, Your awesome music journey starts
   here!', style: TextStyle(
    fontSize: 14,
    fontWeight: FontWeight.normal,
    color: Color.fromARGB(198, 202, 189, 189),
   ),
  ),
  SizedBox(height: 15),
  TextField(
   controller: emailController,
   style:
      TextStyle(color: const Color.fromARGB(255, 0, 0, 0)),
   decoration: InputDecoration(
    border: OutlineInputBorder(),
    labelText: "Email", labelStyle:
    TextStyle(
       color: const Color.fromARGB(255, 0, 0, 0)),
    focusedBorder: OutlineInputBorder(
    borderSide: BorderSide(
         color: Color.fromARGB(255, 0, 0, 0)),
    ),
   ),
  ),
  SizedBox(
```

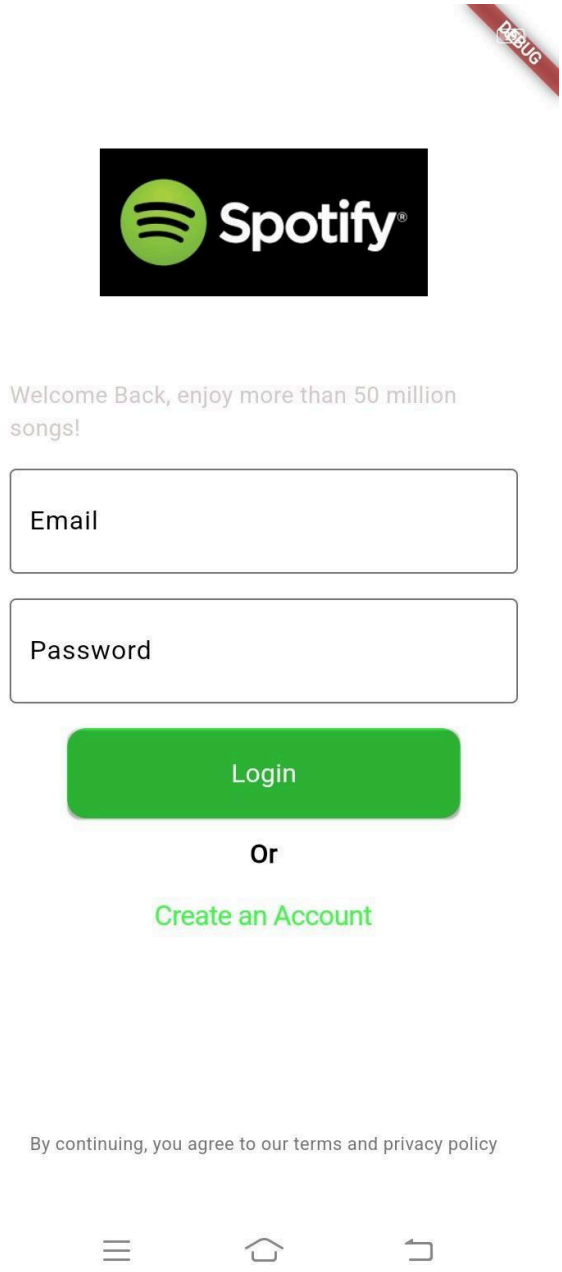
```
  height: 15,
),
TextField(
  controller: passwordController,
  style:
      TextStyle(color: const Color.fromARGB(255, 0, 0, 0)),
  decoration: InputDecoration(
    border: OutlineInputBorder(),
    labelText: "Password",
    labelStyle: TextStyle(
        color: const Color.fromARGB(255, 0, 0, 0)),
    focusedBorder: OutlineInputBorder(
    borderSide: BorderSide(
         color: Color.fromARGB(255, 0, 0, 0)),
    ),
  ),
),
SizedBox(
  height: 15,
),
TextField(
  controller: cPasswordController,
  style:
      TextStyle(color: const Color.fromARGB(255, 0, 0, 0)),
  decoration: InputDecoration(
    border: OutlineInputBorder(),
    labelText: "Confirm Password",
    labelStyle: TextStyle(
        color: const Color.fromARGB(255, 0, 0, 0)),
    focusedBorder: OutlineInputBorder(
    borderSide: BorderSide(
         color: Color.fromARGB(255, 0, 0, 0)),
    ),
```

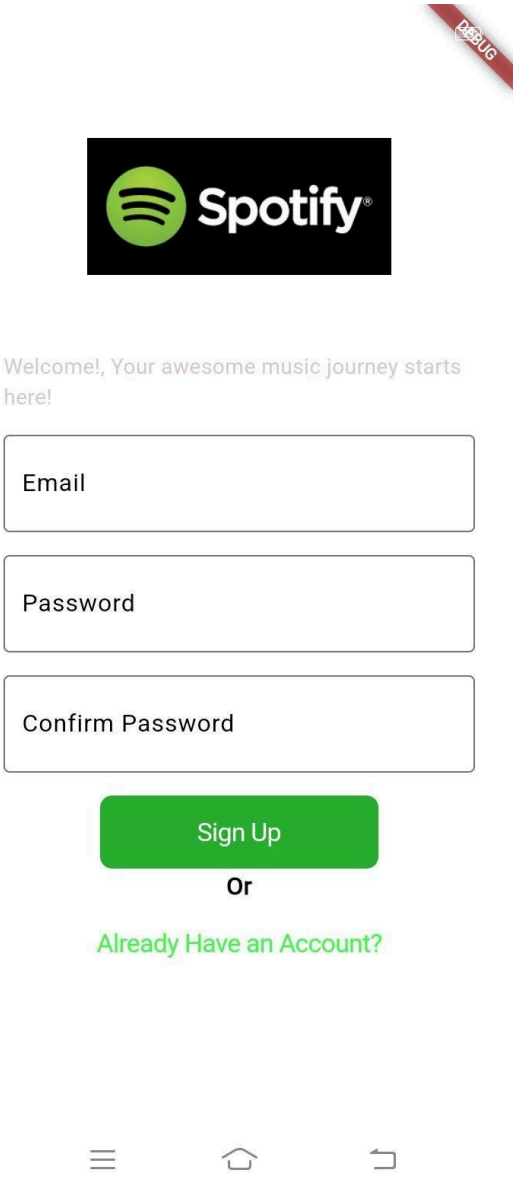
),

```
),
SizedBox(
  height: 15,
),
CupertinoButton(
  onPressed: () {
    setState(() {
      errorMessage
      =
          ''; // Clear error message on button press
    });
    createAccount(); // Call createAccount to handle sign-up logic
  },
  color: Color.fromARGB(255, 40, 175, 47),
  child: Text("Sign Up"),
),
Text(
  'Or'
  ,
  style: TextStyle(
    fontSize: 16,
    fontWeight: FontWeight.bold,
    color: const Color.fromARGB(255, 0, 0, 0),
  ),
),
CupertinoButton(
  onPressed: () {
  Navigator.push(
      context,
      CupertinoPageRoute(
        builder: (context) => LoginScreen(),
      ),
    );
```

```
},
child: Text(
  "Already Have an Account?",
```

```
                    style: TextStyle(
                        color: Color.fromARGB(
                            255, 58, 255, 68)), // Set the text color
                    ),
                  ),
                ],
              ),
            ),
            Padding(
              padding: const
              EdgeInsets.all(8.0), child: Text(
                errorMessage,
                style: TextStyle(
                fontSize: 14,
                color: Colors.red,
                ),
                textAlign: TextAlign.center,
              ),
            ),
          ],
        ),
      ),
    ),
  );
 }
}
```


DEBUG
Spotify
Welcome Back, enjoy more than 50 million songs!
Email
Password
Login
Or
Create an Account
By continuing, you agree to our terms and privacy policy


DEBUG
Spotify
Welcome!, Your awesome music journey starts here!
Email
Password
Confirm Password
Sign Up
Or
Already Have an Account?

MAD & PWA Lab
Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	17
Name	Anurag Mahadev Gaiwal
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	12

MPL Lab Exp 5

Name: Anurag Gaiwal Roll No. 17
Div: D15A Batch: A

Aim:
The aim of this experiment is to create an interactive form using Flutter's form widget. The form will include various form fields such as text fields, dropdown menus, and checkboxes, along with validation logic. Additionally, a button will be added to validate and submit the form data.

Theory:
Gesture:

In Flutter, gestures are user interactions with the application, such as taps, drags, and scrolls. The GestureDetector widget is used to detect various gestures and provides callbacks to handle these interactions. Here's a brief overview:

GestureDetector Widget: The GestureDetector widget wraps its child and detects various gestures applied to it. It provides properties like onTap, onDoubleTap, onLongPress, onPanUpdate, etc., to handle different types of gestures.

Gesture Detection Process: When a user interacts with the screen, the GestureDetector widget detects the gesture based on the user's input and invokes the appropriate callback function, allowing developers to respond to the gesture accordingly.

Navigation:

Navigation in Flutter refers to moving between different screens or routes within the application. Flutter's navigation system is built around the Navigator class, which manages a stack of routes. Here's how it works:

Routes: In Flutter, each screen or page is called a route. Routes are pushed onto and popped off the Navigator's stack to navigate between screens.

Navigator.push(): To navigate from one route to another, you use the Navigator.push() method. This method adds a new route to the stack, displaying the new screen on top of the current one.

Navigator.pop(): To return to the previous route, you use the Navigator.pop() method. This removes the top route from the stack, returning to the previous screen.

Steps to Implement Navigation and Gestures:

Create Two Routes: Define two separate screens or widgets to represent the two routes in your application.

Navigate to Second Route: Use Navigator.push() to navigate from the first route to the second route when a specific gesture or action is detected.

Return to First Route: Implement a mechanism, such as a button press or gesture detection, to trigger Navigator.pop() and return from the second route to the first route.

By combining gestures and navigation, you can create dynamic and interactive Flutter applications that provide a seamless user experience for navigating between different screens and responding to user input through gestures.

Code in main.dart:

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

import 'package:audioplayers/audioplayers.dart';

void main() {
  AudioPlayer player = AudioPlayer(); // Initialize audioplayers
  runApp(MyApp(player: player));
}

class MyApp extends StatelessWidget {
  final AudioPlayer player;
  const MyApp({Key? key, required this.player}) : super(key: key);

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

```
      scaffoldBackgroundColor:
      Colors.black, appBarTheme:
      AppBarTheme( backgroundColor:
      Colors.black, titleTextStyle: TextStyle(
         color:
         Colors.white,
         fontSize: 34.0,
         fontWeight: FontWeight.bold,
        ),
      ),
     ),
     home: SplashScreen(),
   );
  }
}
```

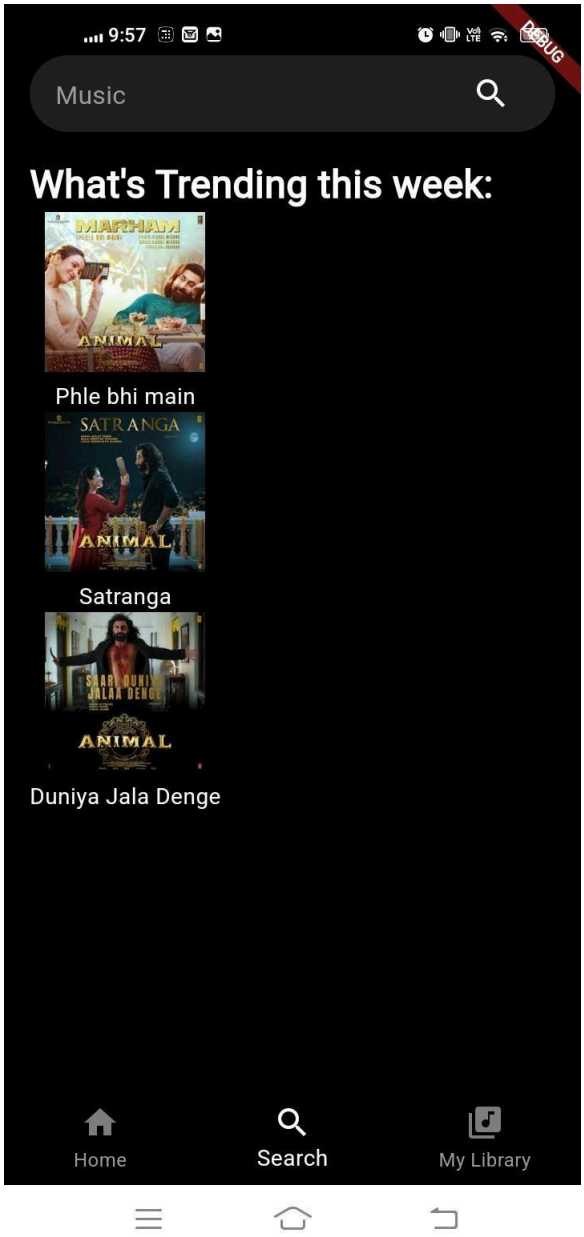
Code in navigation_bar.dart

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

class CustomBottomNavigationBar extends StatelessWidget {
  final int selectedIndex;
  final void Function(int) onItemTapped;
  final int currentIndex;

  CustomBottomNavigationBar({
   required this.selectedIndex,
   required this.onItemTapped,
   this.currentIndex = 0, // Provide a default value for currentIndex
  });

  @override
  Widget build(BuildContext context) {
   return BottomNavigationBar(
   currentIndex: selectedIndex,
```

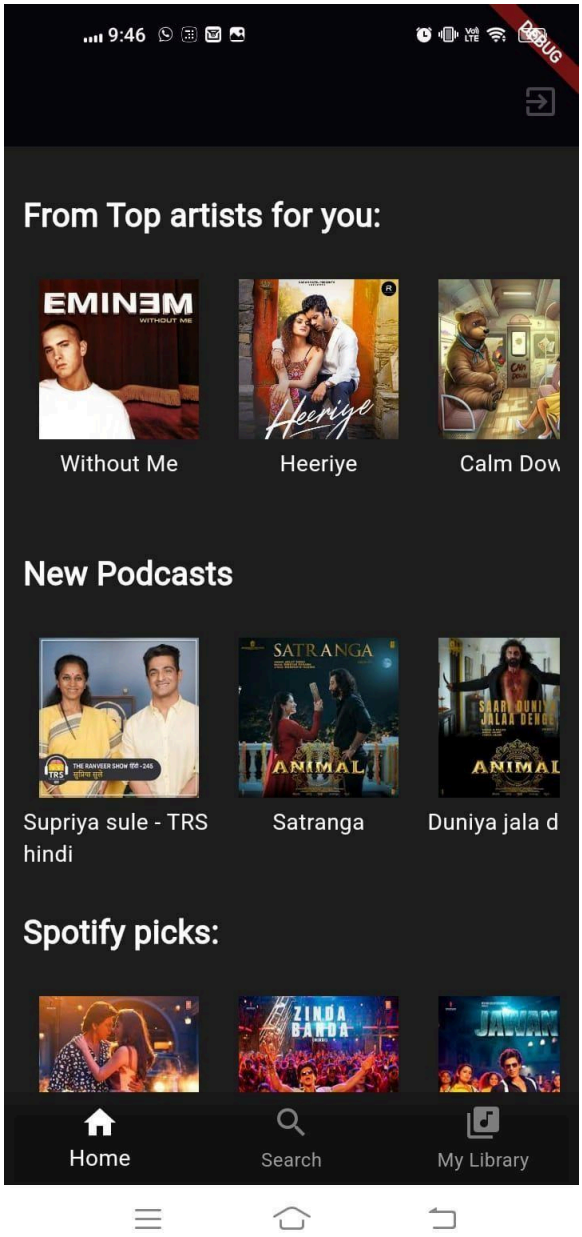
```
onTap: onItemTapped,
```

```
      items: [
        BottomNavigationBarItem(
          icon: Icon(Icons.home, color:
          _getColor(0)), label: 'Home',
        ),
        BottomNavigationBarItem(
          icon: Icon(Icons.search, color:
          _getColor(1)), label: 'Search',
        ),
        BottomNavigationBarItem(
          icon: Icon(Icons.library_music, color:
          _getColor(2)), label: 'My Library',
        ),
      ],
      backgroundColor: Color.fromARGB(31, 0, 0, 0),
      selectedItemColor: Color.fromARGB(255, 255, 255, 255),
      unselectedItemColor: Colors.grey,
    );
  }

  Color _getColor(int index) {
    return currentIndex == index
        ? Colors.white
        : const Color.fromARGB(255, 128, 128, 128);
  }
}
```

Switched to search screen



Home Screen



MAD & PWA Lab
Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	17
Name	Anurag Mahadev Gaiwal
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	13

MPL Lab Exp 6

Name: Anurag Gaiwal Roll No. 17
Div: D15A Batch: A

Aim: To Connect Flutter UI with FireBase database

Theory:

Prerequisites

To complete this tutorial, you will need:

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

Code. Create a Firebase Project:

First, log in with your Google account to manage your Firebase projects. From within the Firebase dashboard,

select the Create new project button and give it a name:

Go to the Firebase Console and create a new

project. Add your Flutter app to the Firebase project:

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

The most important thing here is to match up the Android package name that you choose here with the one

inside of our application.

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

name, and the application name:

com.example.flutterfirebaseexample

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

the Android package name:

android/app/build.gradle

...

defaultConfig {

// TODO: Specify your own unique Application ID

(https://developer.android.com/studio/build/application-id.html)

. applicationId 'com.example.flutterfirebaseexample'

...

} ...

Downloading the Config File

The next step is to add the Firebase configuration file into our Flutter project. This is important as it contains

the API keys and other critical information for Firebase to use.

Select Download google-services.json from this page:

2. Add Firebase to your Flutter project:

Add Dependencies:

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

yaml

Code:

main.dart

```
import
'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';
import
'package:jio_saavn_auth/firebase_options.dart';
import 'package:jio_saavn_auth/models/song.dart';
import
'package:jio_saavn_auth/screens/welcome_screen.dart';
import 'package:provider/provider.dart';

Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
```

```
);
// runApp(
// ChangeNotifierProvider(
```

```
  //    create: (context) => LikedSongsModel(),
  //    child: MyApp(),
  // ),
  // );
  runApp(
    MultiProvider(
      providers: [
        ChangeNotifierProvider(create: (context) => LikedSongsModel()),
        // Add other providers if needed
      ],
      child: MyApp(),
    ),
  );
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return ChangeNotifierProvider(
      create: (context) => LikedSongsModel(),
      child: MaterialApp(
        title: 'Spotify',
        home: WelcomePage(), // Use your main screen here
      ),
    );
  }
}
```

Firebase_options.dart:

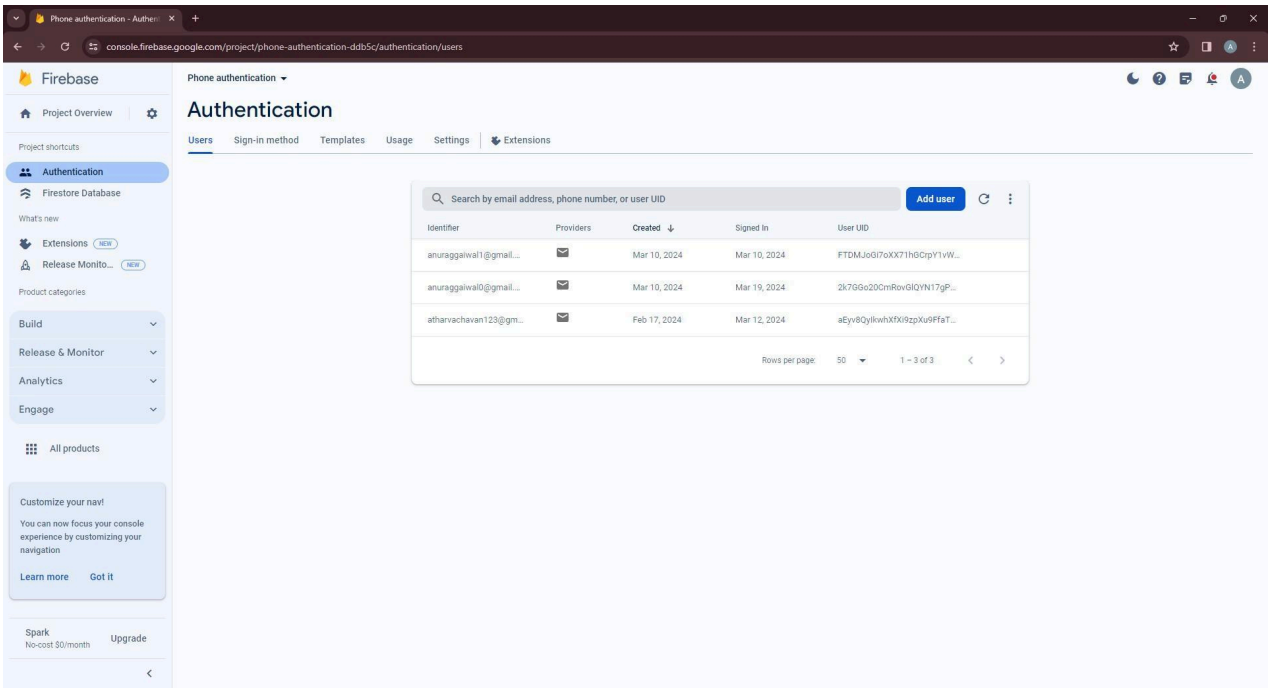
```
// File generated by FlutterFire CLI.
// ignore_for_file: lines_longer_than_80_chars,
avoid_classes_with_only_static_members
import 'package:firebase_core/firebase_core.dart' show FirebaseOptions;
```

```
import 'package:flutter/foundation.dart'
    show defaultTargetPlatform, kIsWeb, TargetPlatform;

/// Default [FirebaseOptions] for use with your Firebase apps.
///
/// Example:
/// ```dart
/// import 'firebase_options.dart';
/// // ...
/// await Firebase.initializeApp(
/// options: DefaultFirebaseOptions.currentPlatform,
/// );
/// ```
class DefaultFirebaseOptions {
  static FirebaseOptions get currentPlatform {
    if (kIsWeb) {
      throw UnsupportedError(
        'DefaultFirebaseOptions have not been configured for web - '
        'you can reconfigure this by running the FlutterFire CLI
        again.',
      );
    }
    switch (defaultTargetPlatform) {
      case TargetPlatform.android:
        return android;
      case TargetPlatform.iOS:
        throw UnsupportedError(
          'DefaultFirebaseOptions have not been configured for ios - '
          'you can reconfigure this by running the FlutterFire CLI
          again.',
        );
      case TargetPlatform.macOS:
        throw UnsupportedError(
          'DefaultFirebaseOptions have not been configured for macos -
```

```
    ' 'you can reconfigure this by running the FlutterFire CLI
    again.',
  );
```

```
      case TargetPlatform.windows:
        throw UnsupportedError(
          'DefaultFirebaseOptions have not been configured for windows - '
          'you can reconfigure this by running the FlutterFire CLI again.',
        );
      case TargetPlatform.linux:
        throw UnsupportedError(
          'DefaultFirebaseOptions have not been configured for linux - '
          'you can reconfigure this by running the FlutterFire CLI
          again.',
        );
      default:
        throw UnsupportedError(
          'DefaultFirebaseOptions are not supported for this platform.',
        );
    }
  }

  static const FirebaseOptions android = FirebaseOptions(
    apiKey: 'AIzaSyBCubxVCQHG3ieLoA11-3EzmohuAsMET8c',
    appId: '1:867583680837:android:81bfa85c82da3424599e2c',
    messagingSenderId: '867583680837',
    projectId: 'phone-authentication-ddb5c',
    storageBucket: 'phone-authentication-ddb5c.appspot.com',
  );
}
```

Firebase console:



Conclusion:

In this experiment, we have successfully connected firebase database and authenticated using google signin and email and password with out flutter application successfully.

MAD & PWA Lab
Journal

Experiment No.	07
Experiment Title.	To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”.
Roll No.	17
Name	Anurag Mahadev Gaiwal
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	15

PWA Lab Exp 7

Name: Anurag Gaiwal Roll No. 17
Div: D15A Batch: A

Theory:-

Regular Web App

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

Progressive Web App

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

Difference between PWAs vs. Regular Web Apps:

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

1. Native Experience

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

2. Ease of Access

Unlike other mobile apps, PWAs do not demand longer download time and make memory space available for installing the applications. The PWAs can be shared and installed by a link, which cuts down the number of steps to install and use.

These applications can easily keep an app icon on the user's home screen, making the app easily accessible to the users and helps the brands remain in the users' minds, and improving the chances of interaction.

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

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

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

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

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

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

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

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

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

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

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

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

Reactivable — Make it easy to reactivate the application thanks to capabilities such as web notifications.
Installable — They allow the user to “save” the apps that he considers most useful with the corresponding icon on the screen of his mobile terminal (home screen) without having to face all the steps and problems related to the use of the app store.

Linkable — Easily shared via URL without complex installations.

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

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

Hosted app link : https://todo-list-using-react-next.vercel.app/

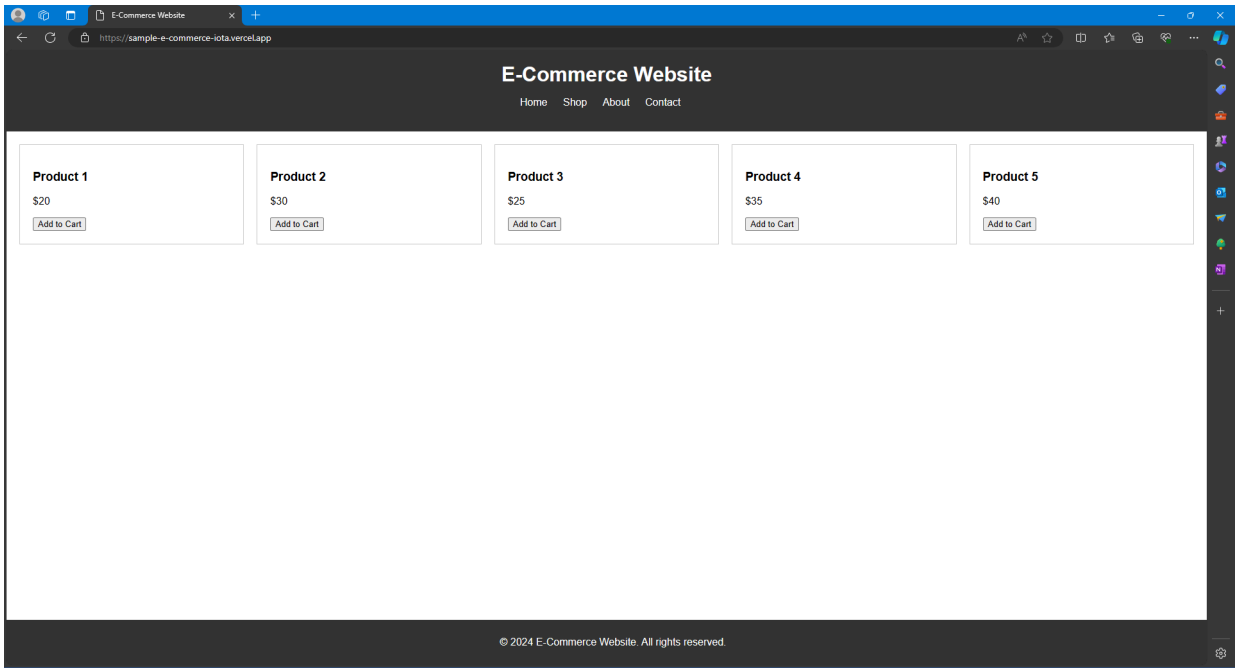
Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
    initial-scale=1.0">
    <title>E-Commerce Website</title>
    <link rel="stylesheet" href="styles.css">
    <meta name="theme-color" content="#4285f4">
    <link rel="apple-touch-icon" href="">
    <link rel="manifest" href="manifest.json">
</head>
<body>
```

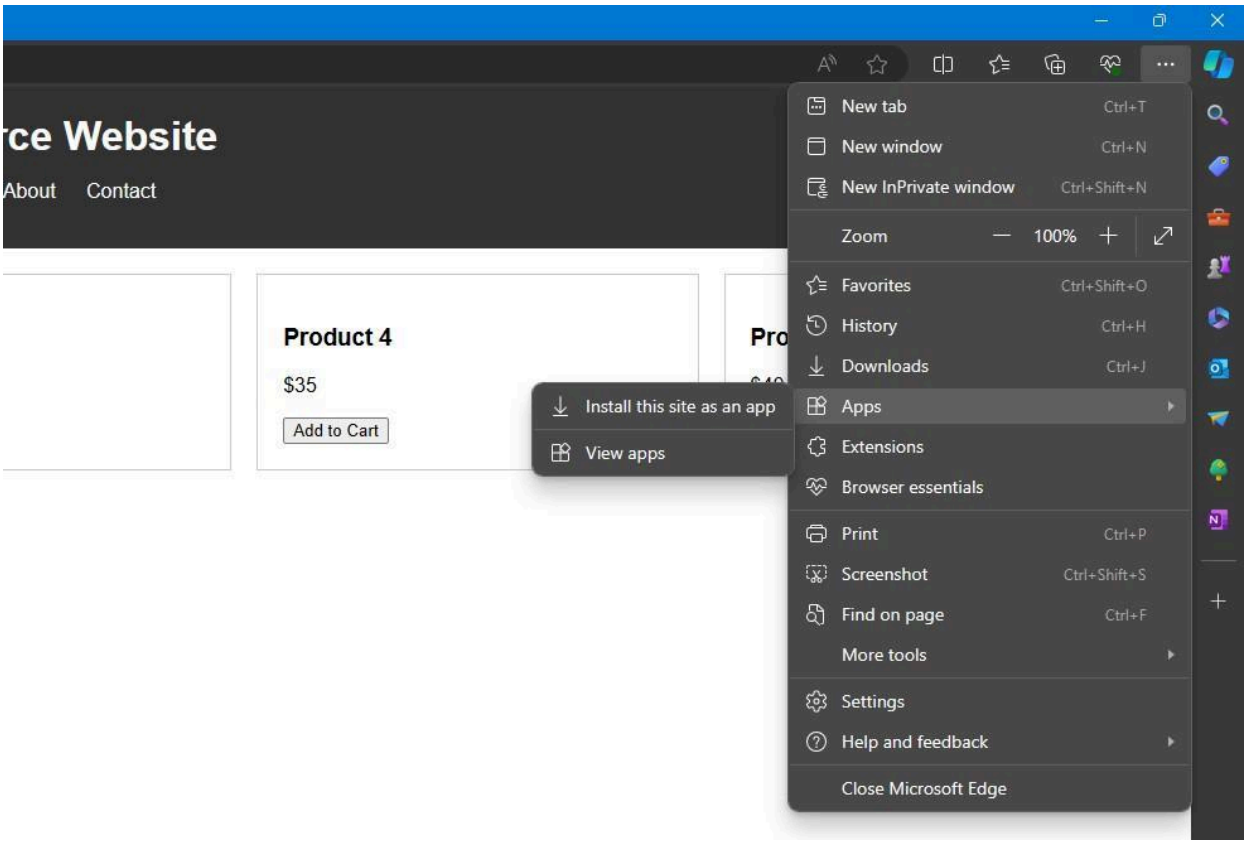
```
    <header>
        <h1>E-Commerce Website</h1>
        <nav>
            <ul>
                <li><a href="#">Home</a></li>
                <li><a href="#">Shop</a></li>
                <li><a href="#">About</a></li>
                <li><a href="#">Contact</a></li>
            </ul>
        </nav>
    </header>

    <main>
        <section class="products">
            <!-- Product cards will be dynamically generated here -->
        </section>
    </main>

    <footer>
        <p>&copy; 2024 E-Commerce Website. All rights reserved.</p>
    </footer>

    <script src="script.js"></script>
</body>
</html>
```

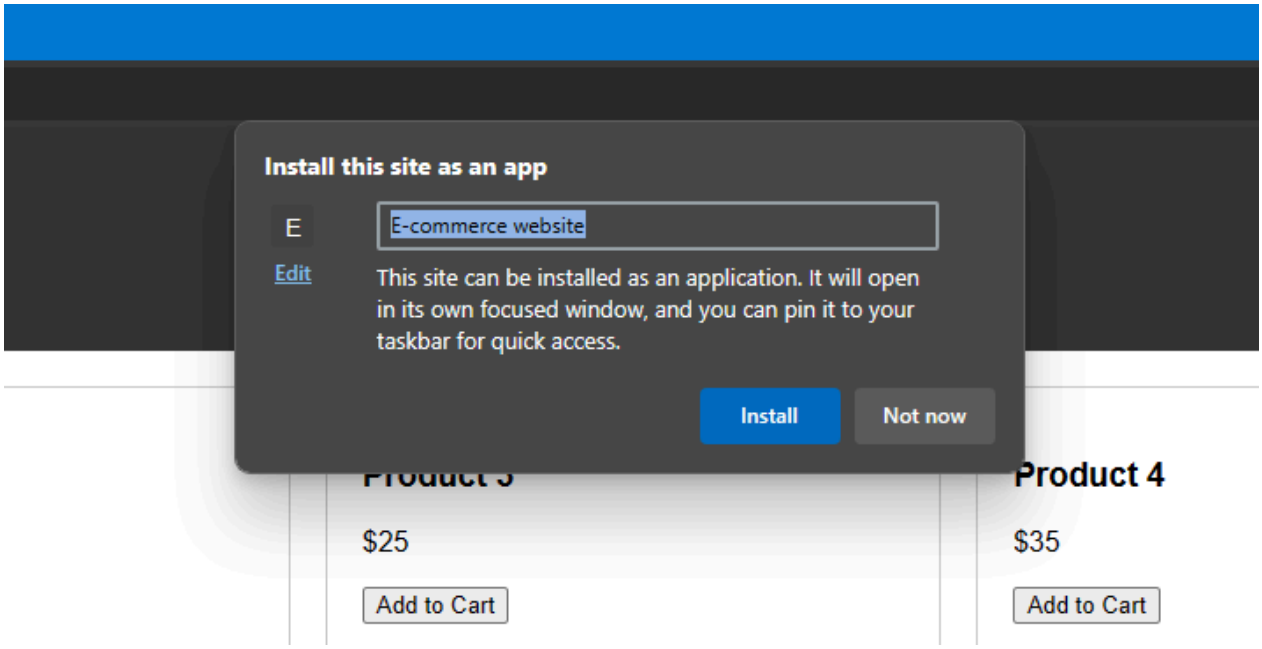
—

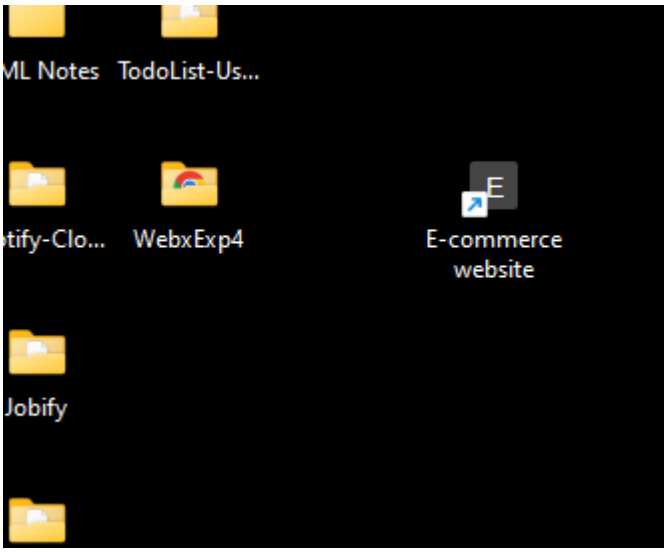
Open this on microsoft edge browser

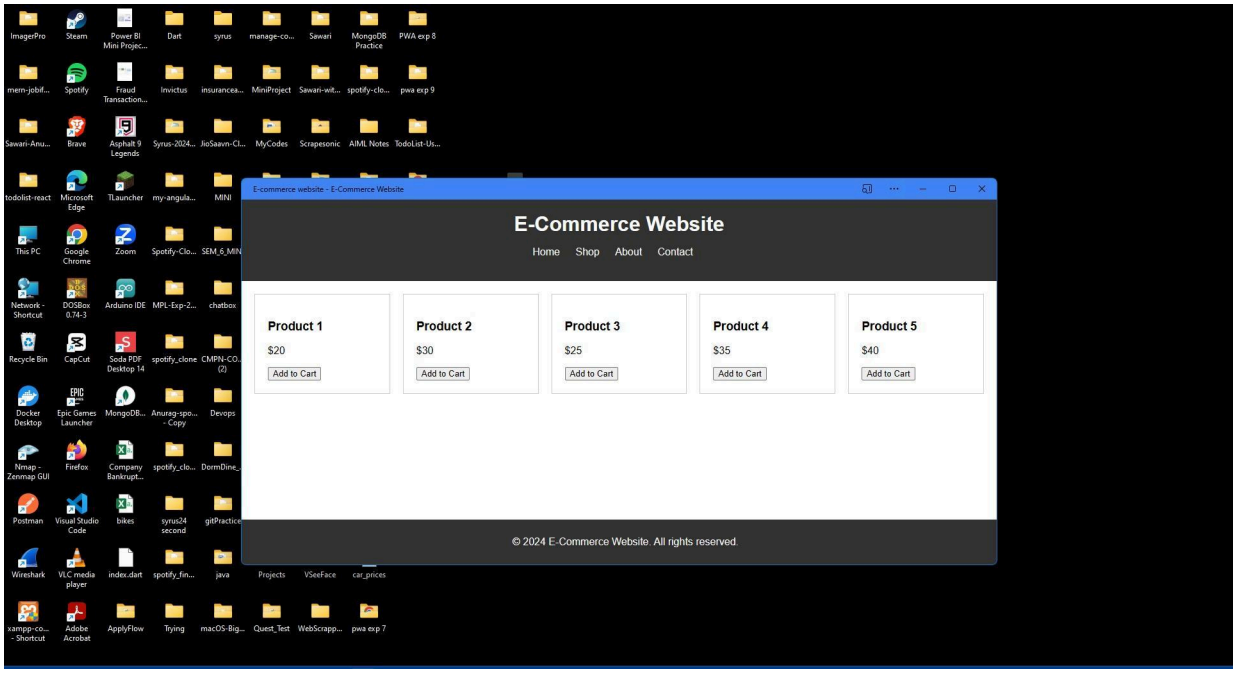


Click on 3 dots, and then applications, and then install this site as an app.




Install this site as an app
E
Edit
E-commerce website
This site can be installed as an application. It will open in its own focused window, and you can pin it to your taskbar for quick access.
Install
Not now
Product 4
$25
$35
Add to Cart
Add to Cart


ML Notes
TodoList-Us...
tify-Clo...
WebxExp4
E-commerce website
Jobify



Conclusion: In this experiment, we successfully created a basic progressive application of our website and installed it on desktop.

MAD & PWA Lab
Journal

Experiment No.	08
Experiment Title.	To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA
Roll No.	17
Name	Anurag Mahadev Gaiwal
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

PWA Lab Exp 8

Name: Anurag Gaiwal Roll No. 17
Div: D15A Batch: A

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

Theory:

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

Things to note about Service Worker:

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

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

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

You can manage Push Notifications
You can manage push notifications with Service Worker and show any information message to the user.
You can Continue
Although Internet connection is broken, you can start any process with Background Sync of Service Worker.
What can’t we do with Service Workers?

You can’t access the Window

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

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

Service Worker Cycle
A service worker goes through three steps in its life cycle:
Registration
Installation
Activation

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

main.js

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

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

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

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

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

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

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

main.js

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

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

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

service-worker.js

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

```
});
```

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

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

service-worker.js

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

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

Sample Code with Output

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>E-Commerce Website</title>
    <link rel="stylesheet" href="styles.css">
    <meta name="theme-color" content="#4285f4">
    <link rel="apple-touch-icon" href="">
    <link rel="manifest" href="manifest.json">
</head>
<body>
    <header>
        <h1>E-Commerce Website</h1>
        <nav>
            <ul>
                <li><a href="#">Home</a></li>
                <li><a href="#">Shop</a></li>
                <li><a href="#">About</a></li>
                <li><a href="#">Contact</a></li>
            </ul>
        </nav>
    </header>

    <main>
        <section class="products">
            <!-- Product cards will be dynamically generated here -->
        </section>
    </main>

    <footer>
        <p>&copy; 2024 E-Commerce Website. All rights reserved.</p>
    </footer>

    <script src="script.js"></script>
</body>
```

```
</html>
```

app.js

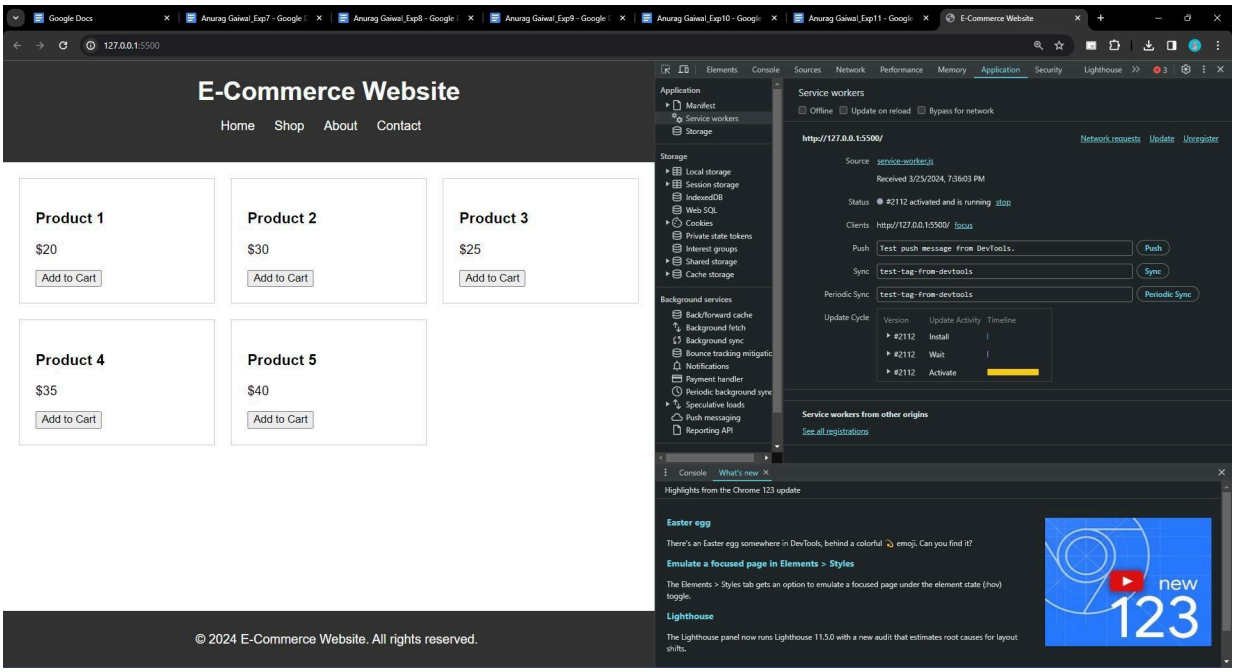
```
if ('serviceWorker' in navigator) {
    window.addEventListener('load', () => {
        navigator.serviceWorker.register('/service-worker.js')
            .then(registration => {
                console.log('Service Worker registered with
scope:', registration.scope);
            })
            .catch(error => {
                console.error('Service Worker registration failed:', error);
            });
    });
}
```

service-worker.js

```
// service-worker.js

const cacheName =
'ecommerce-pwa-v1'; const
assetsToCache = [
    '/',
    '/index.htm
    l',
    '/main.css'
    ,
    '/app.js'
    // Add more files and assets here as needed
];

self.addEventListener('install', event
    => { event.waitUntil(
        caches.open(cacheName)
            .then(cache => {
                return cache.addAll(assetsToCache);
            })
    );
});

self.addEventListener('activate', event => {
```

```
    event.waitUntil(
        caches.keys().then(cacheNames
            => { return Promise.all(
                cacheNames.filter(name => {
                    return name !== cacheName;
                }).map(name => {
                    return caches.delete(name);
                })
            );
        })
    );
});
```

Steps for Execution
Create a folder and put all 4 files main.css , service-worker.js, app.js, index.html
open visual studio
install extension Live server
open folder in visual studio open index.html
on bottom right corner click go Live
it will open html page in browser
go to developer tools and take following screenshots



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.	17
Name	Anurag Mahadev Gaiwal
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

PWA Lab Exp 9

Name: Anurag Gaiwal Roll No. 17
Div: D15A Batch: A

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

Theory:

Service Worker

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

Things to note about Service Worker:

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

Fetch Event

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

Of course, you can use many different methods but you can find in the following example a “cache first” and “network first” approach. In this example, if the request’s and current location’s origin are the same (Static content is requested.), this is called “cacheFirst” but if you request a targeted external URL, this is called “networkFirst”.

- **CacheFirst** - In this function, if the received request has cached before, the cached response is returned to the page. But if not, a new response requested from the network.
- **NetworkFirst** - In this function, firstly we can try getting an updated response from the network, if this process completed successfully, the new response will be cached and returned. But if this process fails, we check whether the request has been cached before or not. If a cache exists, it is returned to the page, but if not, this is up to you. You can return dummy content or information messages to the page.

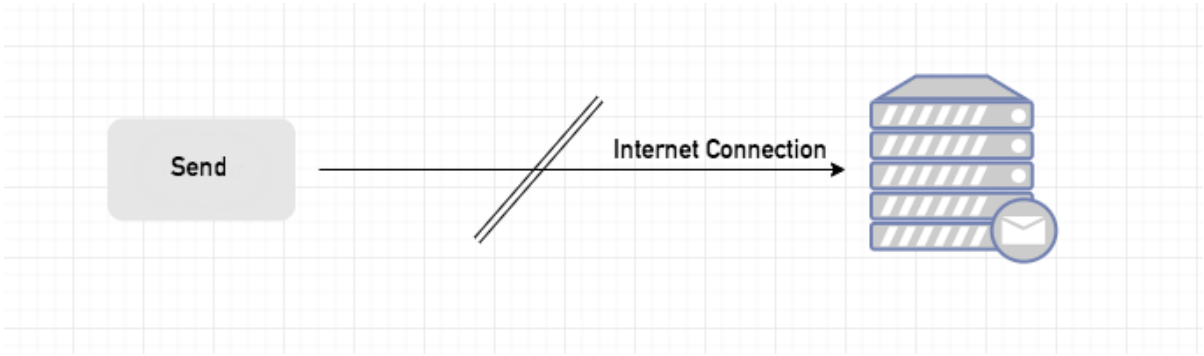
```
self.addEventListener("fetch", function (event) {
    const req = event.request;
    const url = new URL(req.url);

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

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

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

Sync Event

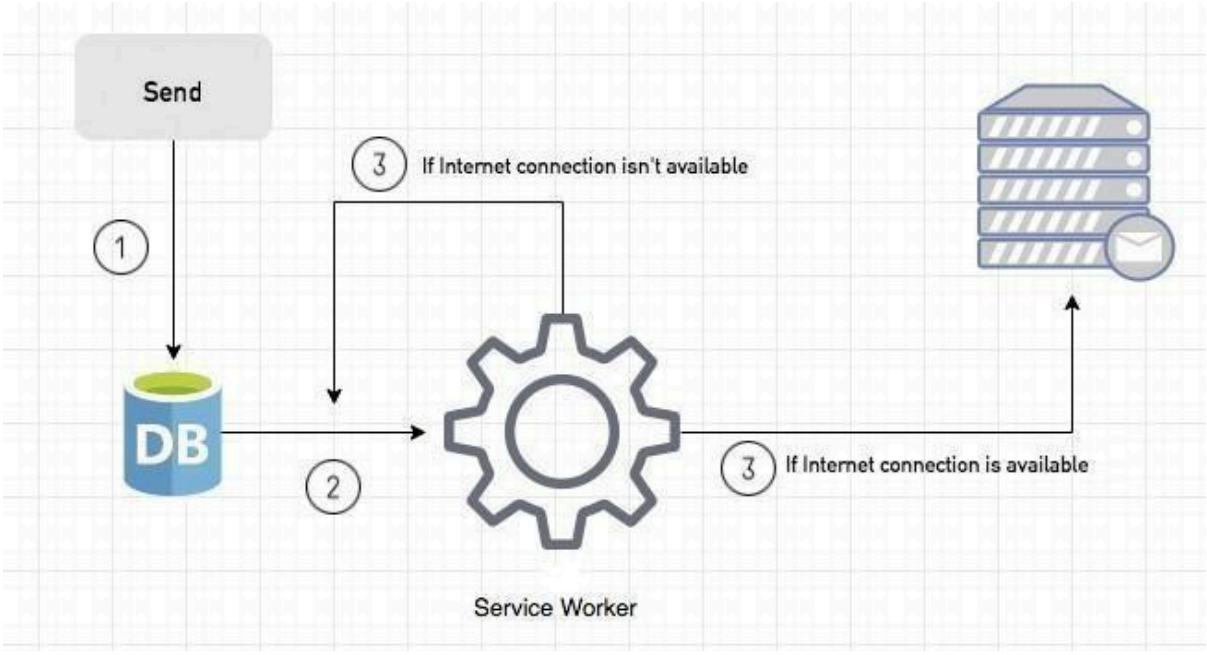
Background Sync is a Web API that is used to delay a process until the Internet connection is stable. We can adapt this definition to the real world; there is an e-mail client application that works on the browser and we want to send an email with this tool. Internet connection is broken while we are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

Here is a job for the Background Sync.

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



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



1. When we click the "send" button, email content will be saved to IndexedDB.
2. Background Sync registration.
3. **If the Internet connection is available**, all email content will be read and sent to Mail

Server.

If the Internet connection is unavailable, the service worker waits until the connection is available even though the window is closed. When it is available, email content will be sent to Mail Server.

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

Event Listener for Background Sync Registration

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

Event Listener for sw.js

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

Push Event

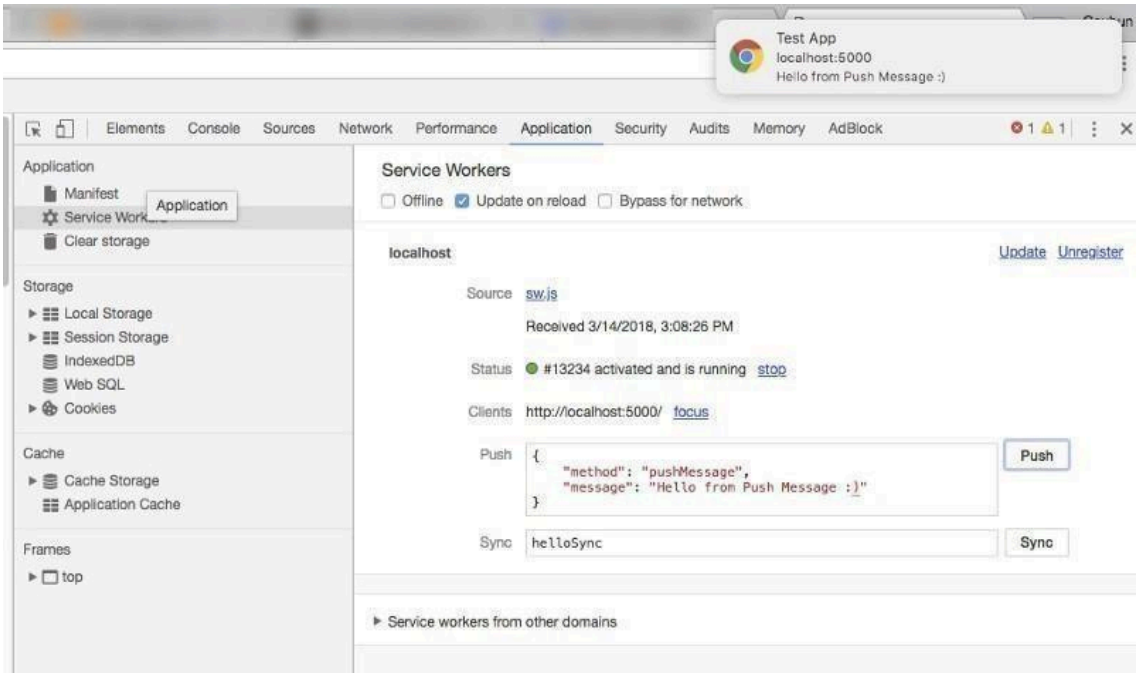
This is the event that handles push notifications that are received from the server. You can apply any method with received data.

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

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

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

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



Code:

sw.js

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

self.addEventListener("fetch", function (event) {
    event.respondWith(checkResponse(event.request).catch(function () {
console.log("Fetch from cache successful!") return
        returnFromCache(event.request);
}));
console.log("Fetch successful!") event.waitUntil(addToCache(event.request));
});

self.addEventListener('sync', event => { if
    (event.tag === 'syncMessage') {
```

```
    console.log("Sync successful!")
  }
});

self.addEventListener('push', function
  (event)  { if (event && event.data) {
    var data = event.data.json();
    if (data.method == "pushMessage")
      { console.log("Push notification
      sent");
      event.waitUntil(self.registration.showNotification("Omkar
        Sweets Corner", { body: data.message
      }))
    }
  }
})


var filesToCache
  = [ '/',
  '/menu',
  '/contactUs',
  '/offline.htm
  l',
];

var preLoad = function () {
  return  caches.open("offline").then(function  (cache)  {
    // caching index and important routes
    return cache.addAll(filesToCache);
  });
};
```

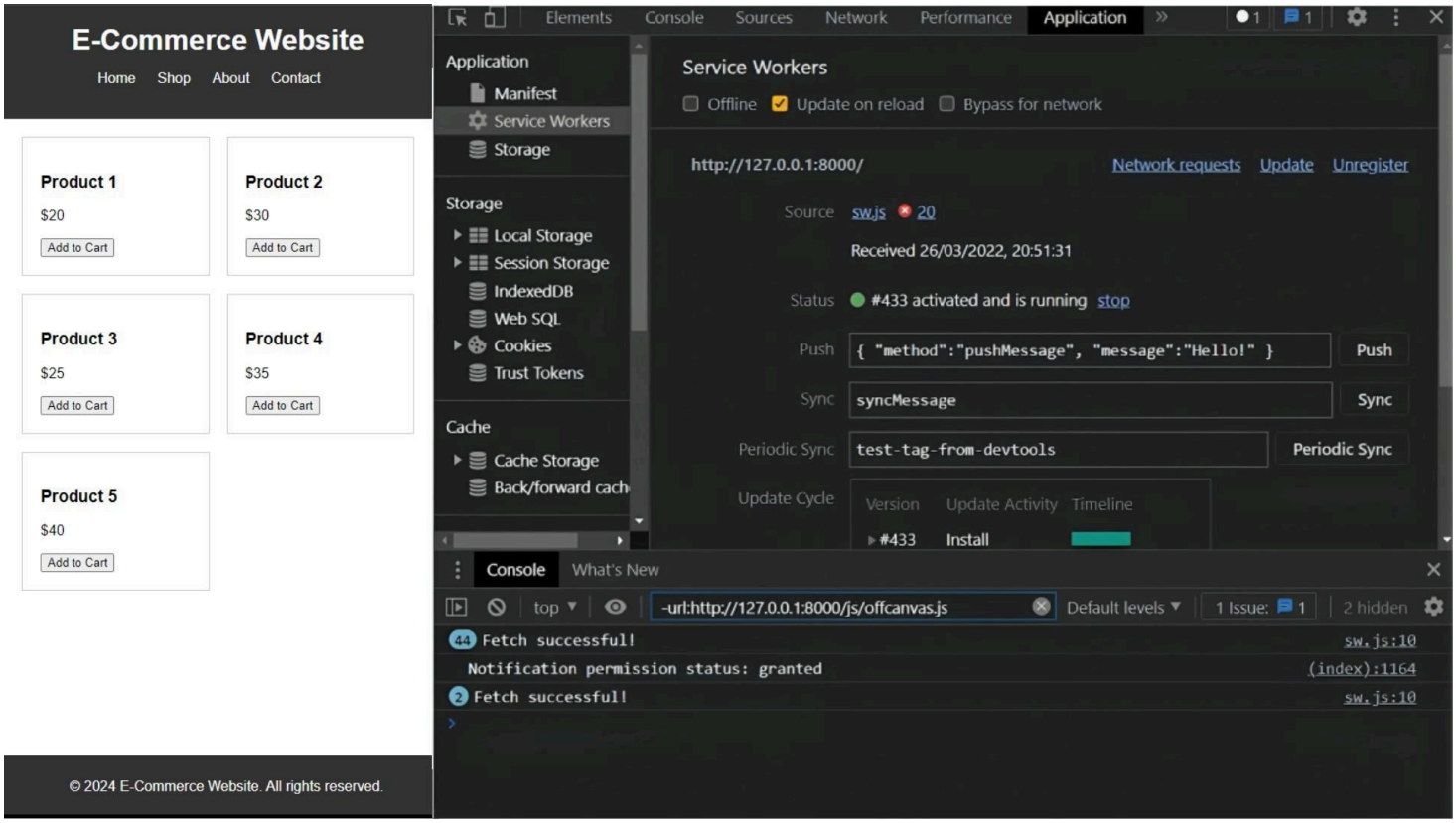
```
var checkResponse =   function
  (request)   { return new
  Promise(function (fulfill, reject)
  {
    fetch(request).then(function
      (response)    { if (response.status
      !== 404) {
        fulfill(response);
      } else {
        reject();
```

```
}
}, reject);
});
};

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

var returnFromCache = function (request) {
    return  caches.open("offline").then(function   (cache)   {
       return cache.match(request).then(function (matching) {
          if (!matching || matching.status == 404) {
             return cache.match("offline.html");
} else {
return matching;
       }
    });
});
};
```

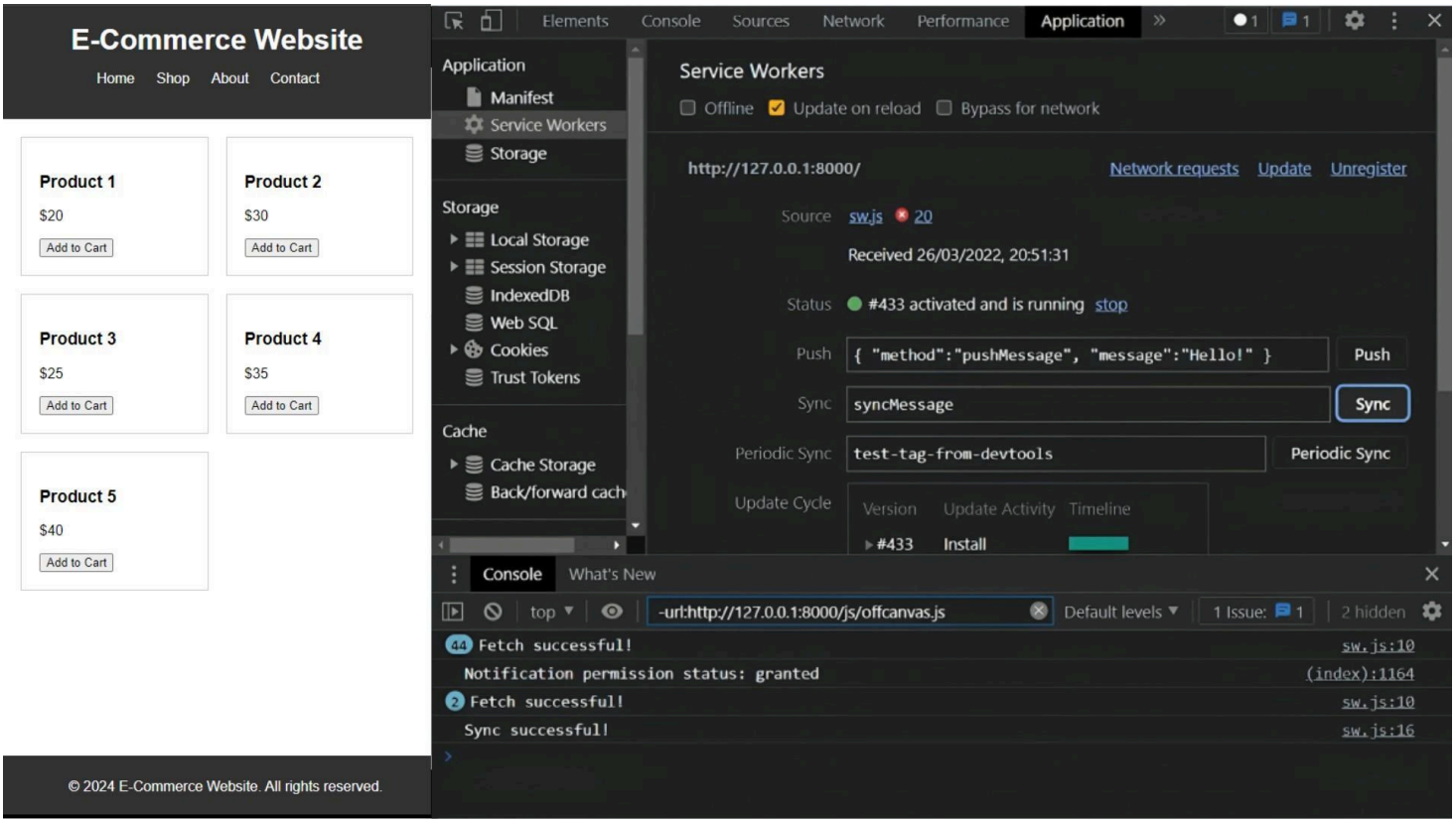
Output:

Fetch

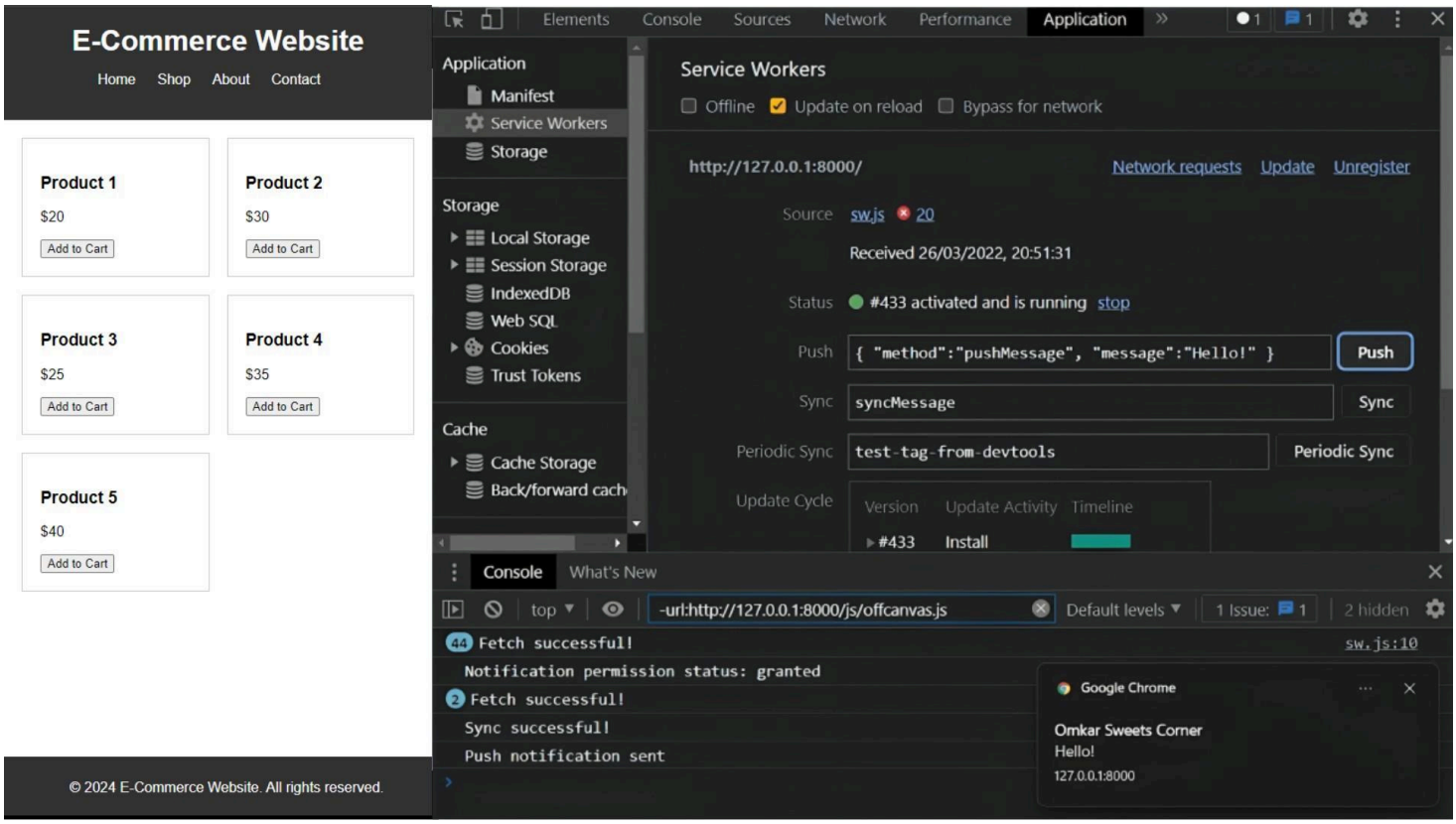
Event


E-Commerce Website
Home Shop About Contact
Product 1
$20
Add to Cart
Product 2
$30
Add to Cart
Product 3
$25
Add to Cart
Product 4
$35
Add to Cart
Product 5
$40
Add to Cart
© 2024 E-Commerce Website. All rights reserved.
Elements Console Sources Network Performance Application
Application
Manifest
Service Workers
Storage
Storage
Local Storage
Session Storage
IndexedDB
Web SQL
Cookies
Trust Tokens
Cache
Cache Storage
Back/forward cach
Service Workers
Offline Update on reload Bypass for network
http://127.0.0.1:8000/
Network requests Update Unregister
Source sw.js 20
Received 26/03/2022, 20:51:31
Status #433 activated and is running stop
Push { "method":"pushMessage", "message":"Hello!" } Push
Sync syncMessage Sync
Periodic Sync test-tag-from-devtools Periodic Sync
Update Cycle
Version Update Activity Timeline
#433 Install
Console What's New
top -url:http://127.0.0.1:8000/js/offcanvas.js Default levels 1 Issue: 1 2 hidden
44 Fetch successful! sw.js:10
Notification permission status: granted (index):1164
2 Fetch successful! sw.js:10

Sync event


E-Commerce Website
Home Shop About Contact
Product 1
$20
Add to Cart
Product 2
$30
Add to Cart
Product 3
$25
Add to Cart
Product 4
$35
Add to Cart
Product 5
$40
Add to Cart
© 2024 E-Commerce Website. All rights reserved.
Elements Console Sources Network Performance Application
Application
Manifest
Service Workers
Storage
Storage
Local Storage
Session Storage
IndexedDB
Web SQL
Cookies
Trust Tokens
Cache
Cache Storage
Back/forward cach
Service Workers
Offline Update on reload Bypass for network
http://127.0.0.1:8000/
Network requests Update Unregister
Source sw.js 20
Received 26/03/2022, 20:51:31
Status #433 activated and is running stop
Push { "method":"pushMessage", "message":"Hello!" } Push
Sync syncMessage Sync
Periodic Sync test-tag-from-devtools Periodic Sync
Update Cycle
Version Update Activity Timeline
#433 Install
Console What's New
top -url:http://127.0.0.1:8000/js/offcanvas.js Default levels 1 Issue: 1 2 hidden
44 Fetch successful! sw.js:10
Notification permission status: granted (index):1164
2 Fetch successful! sw.js:10
Sync successful! sw.js:16

Push event


E-Commerce Website
Home Shop About Contact
Product 1
$20
Add to Cart
Product 2
$30
Add to Cart
Product 3
$25
Add to Cart
Product 4
$35
Add to Cart
Product 5
$40
Add to Cart
© 2024 E-Commerce Website. All rights reserved.
Elements Console Sources Network Performance Application
Application
Manifest
Service Workers
Storage
Storage
Local Storage
Session Storage
IndexedDB
Web SQL
Cookies
Trust Tokens
Cache
Cache Storage
Back/forward cach
Service Workers
Offline Update on reload Bypass for network
http://127.0.0.1:8000/ Network requests Update Unregister
Source sw.js 20
Received 26/03/2022, 20:51:31
Status #433 activated and is running stop
Push { "method":"pushMessage", "message":"Hello!" } Push
Sync syncMessage Sync
Periodic Sync test-tag-from-devtools Periodic Sync
Update Cycle
Version Update Activity Timeline
#433 Install
Console What's New
top -url:http://127.0.0.1:8000/js/offcanvas.js Default levels 1 Issue: 1 2 hidden
44 Fetch successful! sw.js:10
Notification permission status: granted
2 Fetch successful!
Sync successful!
Push notification sent
Google Chrome
Omkar Sweets Corner
Hello!
127.0.0.1:8000

MAD & PWA Lab
Journal

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	17
Name	Anurag Mahadev Gaiwal
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

PWA Lab Exp 10

Name: Anurag Gaiwal Roll No. 17
Div: D15A Batch: A

Aim:

To study and implement deployment of website PWA to GitHub Pages.

Theory:

GitHub Pages

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

GitHub Pages provides the following key features:

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

Reasons for favoring this over Firebase:

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

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

Pros

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

Cons

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

Firebase

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

Some of the features offered by Firebase are:

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

Reasons for favoring over GitHub Pages:

1. Realtime backend made easy
2. Fast and responsive

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

Pros

1. Hosted by Google. Enough said.
2. Authentication, Cloud Messaging, and a whole lot of other handy services will be available to you.
3. A real-time database will be available to you, which can store 1 GB of data.

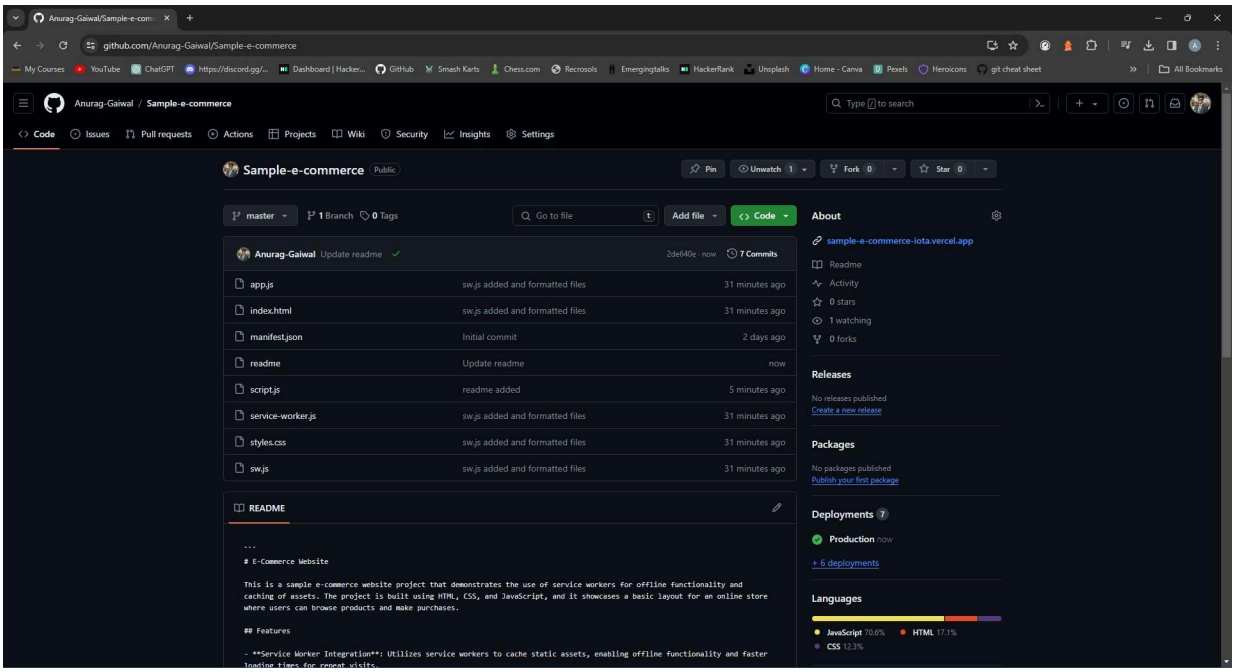
4. You'll also have access to a blob store, which can store another 1 GB of data.
5. Support for HTTPS. A free certificate will be provisioned for your custom domain within 24 hours.

Cons

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

Link to our GitHub repository:
https://github.com/Anurag-Gaiwal/Sample-e-commerce

Github Screenshot:



MAD & PWA Lab
Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	17
Name	Anurag Mahadev Gaiwal
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	15

PWA Lab Exp 11

Name: Anurag Gaiwal Roll No. 17
Div: D15A Batch: A

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

Theory :

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

Google Lighthouse :
Google Lighthouse is a tool that lets you audit your web application based on a number of parameters including (but not limited to) performance, based on a number of metrics, mobile compatibility, Progressive Web App (PWA) implementations, etc. All you have to do is run it on a page or pass it a URL, sit back for a couple of minutes and get a very elaborate report, not much short of one that a professional auditor would have compiled in about a week.
The best part is that you have to set up almost nothing to get started. Let's begin by looking at some of the top features and audit criteria used by Lighthouse.
Key Features and Audit Metrics

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

Performance: This score is an aggregation of how the page fared in aspects such as (but not limited to) loading speed, time taken for loading for basic frame(s), displaying meaningful content to the user, etc. To a layman, this score is indicative of how decently the site performs, with a score of 100 meaning that you figure in the 98th percentile, 50 meaning that you figure in the 75th percentile and so on.
PWA Score (Mobile): Thanks to the rise of Service Workers, app manifests, etc., a lot of modern web applications are moving towards the PWA paradigm, where the objective is to make the application behave as close as possible to native mobile applications. Scoring points are based on the Baseline PWA checklist laid down by Google which includes Service Worker implementation(s), viewport handling, offline functionality, performance in script-disabled environments, etc.
Accessibility: As you might have guessed, this metric is a measure of how accessible your website is, across a plethora of accessibility features that can be implemented in your page (such as the 'aria-' attributes like aria-required, audio captions, button names,

etc.). Unlike the other metrics though, Accessibility metrics score on a pass/fail basis i.e. if all possible elements of the page are not screen-reader friendly (HTML5 introduced features that would make pages easy to interpret for screen readers used by visually challenged people like tag names, tags such as <section>, <article>, etc.), you get a 0 on that score. The aggregate of these scores is your Accessibility metric score.
Best Practices: As any developer would know, there are a number of practices that have been deemed 'best' based on empirical data. This metric is an aggregation of many such points, including but not limited to:Use of HTTPS
Avoiding the use of deprecated code elements like tags, directives, libraries, etc.
Password input with paste-into disabled
Geo-Location and cookie usage alerts on load, etc.

Code:
Index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
    initial-scale=1.0">
    <title>E-Commerce Website</title>
    <link rel="stylesheet" href="styles.css">
    <meta name="theme-color" content="#4285f4">
    <link rel="apple-touch-icon" href="">
    <link rel="manifest" href="manifest.json">
</head>
<body>
    <header>
        <h1>E-Commerce Website</h1>
        <nav>
            <ul>
                <li><a href="#">Home</a></li>
                <li><a href="#">Shop</a></li>
                <li><a href="#">About</a></li>
                <li><a href="#">Contact</a></li>
            </ul>
        </nav>
    </header>

    <main>
```

```
        <section class="products">
            <!-- Product cards will be dynamically generated here -->
        </section>
    </main>

    <footer>
        <p>&copy; 2024 E-Commerce Website. All rights reserved.</p>
    </footer>

    <script src="script.js"></script>
</body>
</html>
```

Styles.css:

```
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
}

header {
    background-color: #333;
    color: #fff;
    padding: 20px;
    text-align: center;
}

header h1 {
    margin: 0;
}

nav ul {
    list-style-type: none;
    padding: 0;
}

nav ul li {
    display: inline;
    margin-right: 20px;
```

```
}

nav ul li a {
    color: #fff;
    text-decoration: none;
}

main {
    padding: 20px;
}

.products {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(200px,
    1fr)); grid-gap: 20px;
}

.product {
    border: 1px solid
    #ccc; padding: 20px;
}

footer {
    background-color:
    #333; color: #fff;
    text-align:
    center; padding:
    10px; position:
    fixed; bottom: 0;
    width: 100%;
}
```

script.js:

```
// Simulated data for products
const products = [
    { name: "Product 1", price: 20 },
    { name: "Product 2", price: 30 },
    { name: "Product 3", price: 25 },
    { name: "Product 4", price: 35 },
```

```
    { name: "Product 5", price: 40 }
];

// Function to generate product
cards function
generateProductCards() {
    const productsSection = document.querySelector('.products');

    products.forEach(product => {
        const productCard =
        document.createElement('div');
        productCard.classList.add('product');
        productCard.innerHTML = `
            <h3>${product.name}</h3>
            <p>$${product.price}</p>
            <button>Add to Cart</button>
        `;
        productsSection.appendChild(productCard
        );
    });
}

// Call the function to generate product
cards generateProductCards();
```

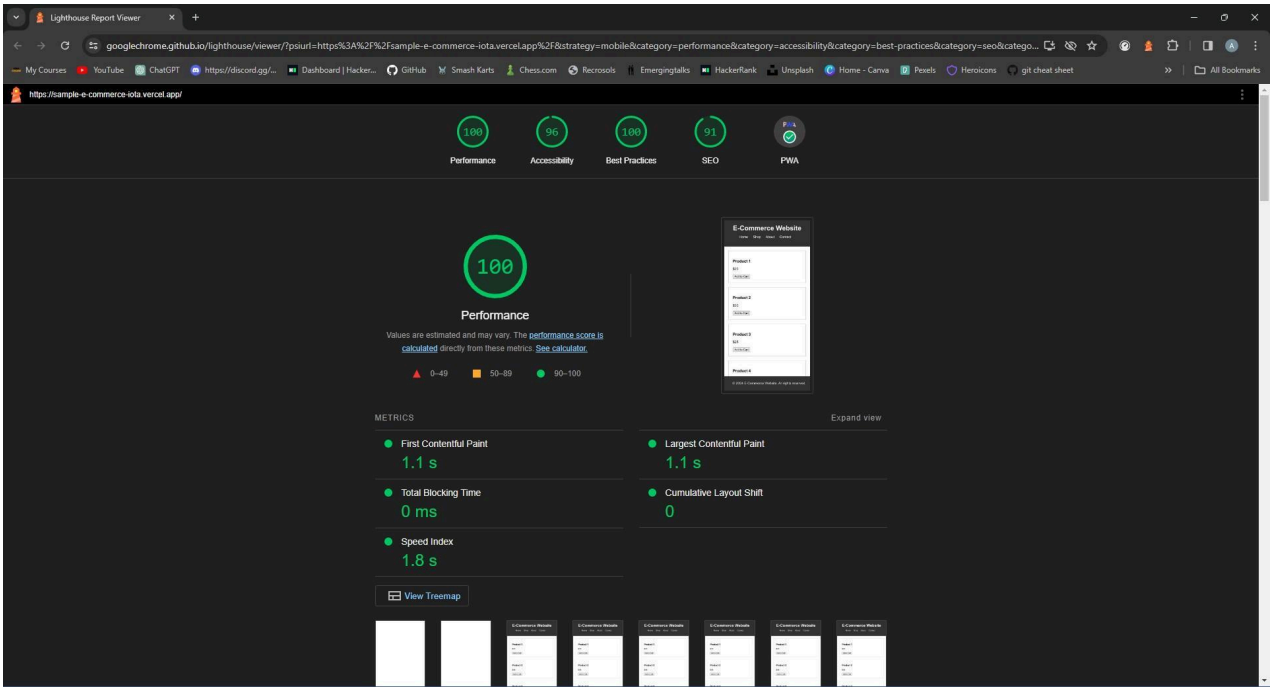
Manifest.json:

```
{
  "name": "E-commerce website",
  "short_name": "E-commerce",
  "start_url": "index.html",
  "scope": "./",
  "theme_color": "#ffd31d",
  "background_color": "#333",
  "display": "standalone",
  "icons": [
    {
      "src": "icon-1.png",
      "sizes": "192x192",
      "type": "image/png",
      "purpose": "any maskable"
    },
```

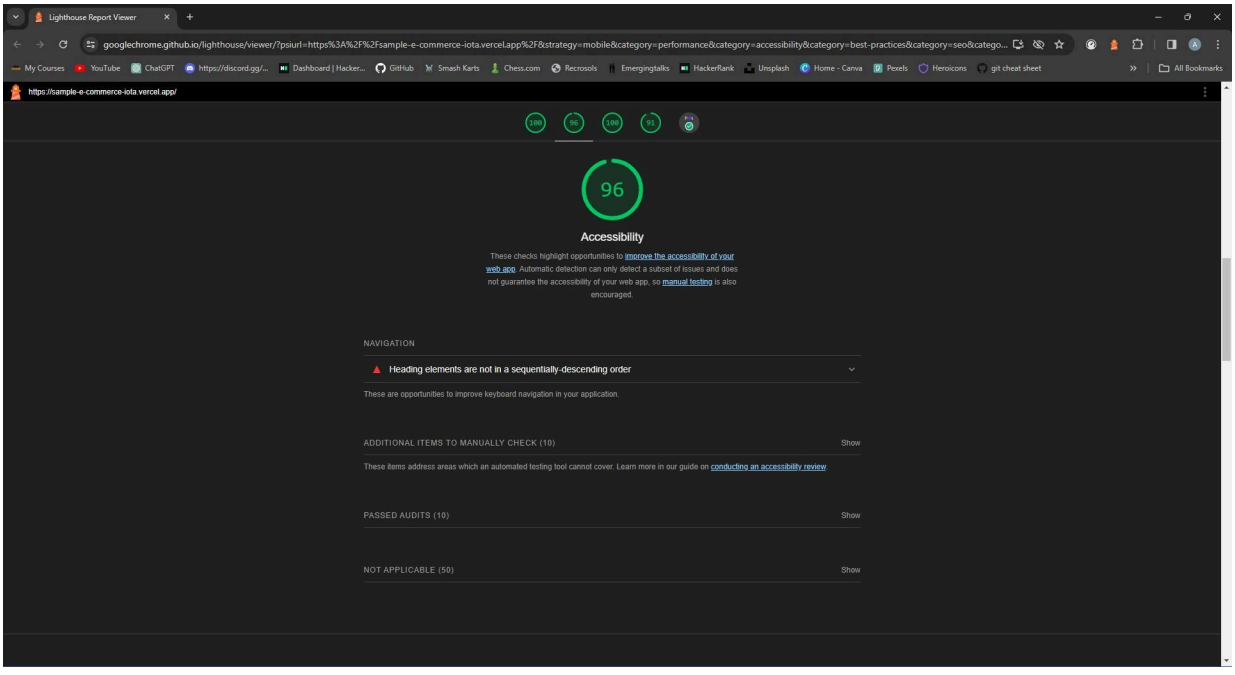
```
        "src": "icon-2.png",
        "sizes": "512x512",
        "type": "image/png",
        "purpose": "any
        maskable"
      }
    ]
}
```

Output:

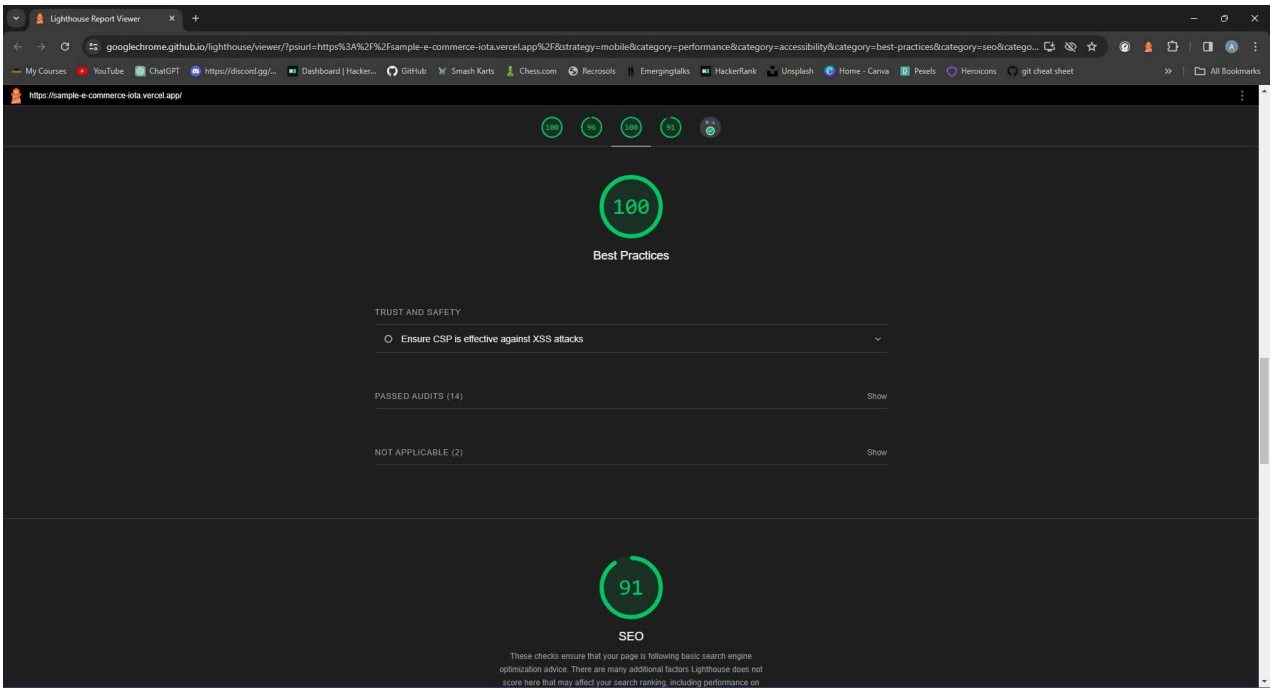
Performance -



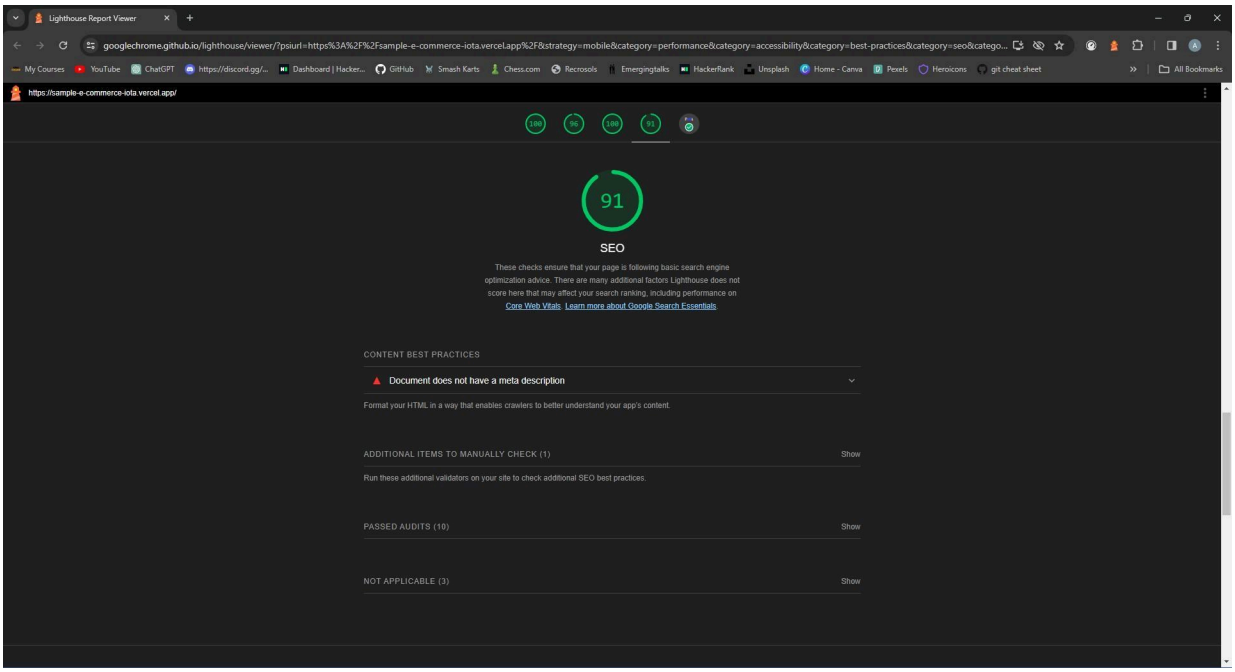
Accessibility -



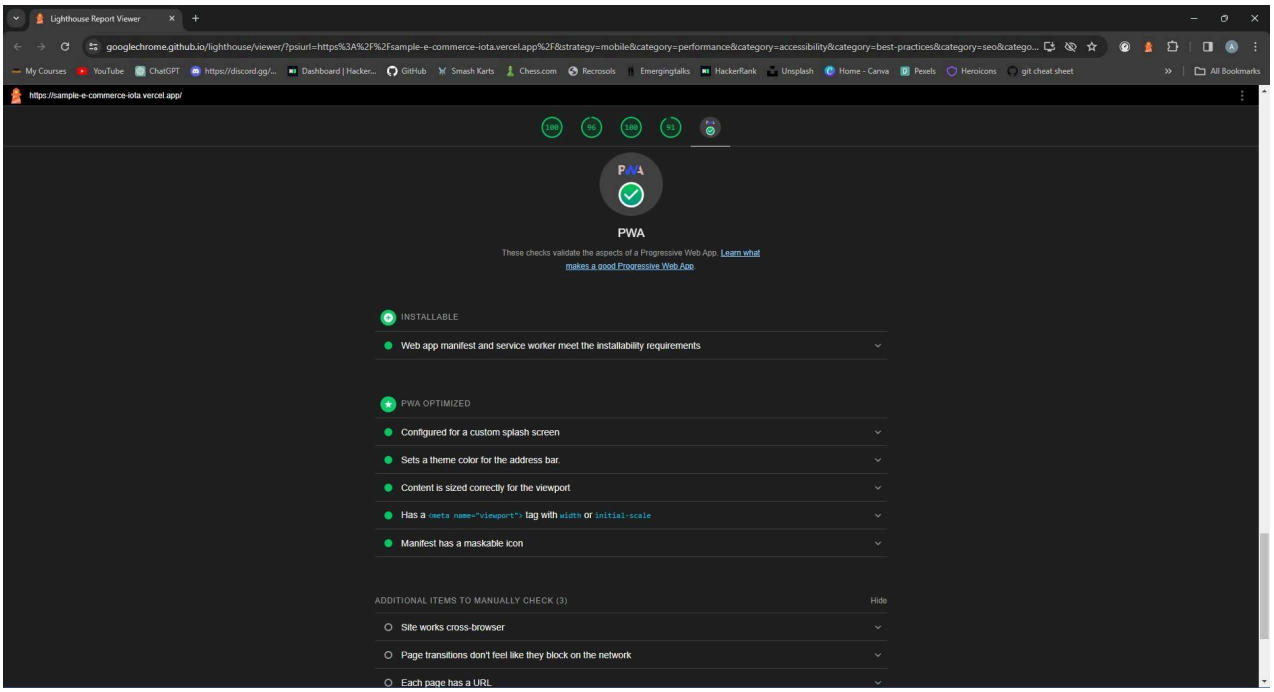
Best Practices :



SEO :



PWA :



Hosted Website : https://sample-e-commerce-iota.vercel.app/

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

MAD & PWA Lab
Journal

Experiment No.	Assignment-1
Assignment 1 Questions	1. Flutter Overview:Explain the key features and advantages of using Flutter for mobile app development. Discuss how the Flutter framework differs from traditional approaches and why it has gained popularity in the developer community. 2. Widget Tree and Composition: Describe the concept of the widget tree in Flutter. Explain how widget composition is used to build complex user interfaces. Provide examples of commonly used widgets and their roles in creating a widget tree. 3. State Management in Flutter: Discuss the importance of state management in Flutter applications. Compare and contrast the different state management approaches available in Flutter, such as setState, Provider, and Riverpod. Provide scenarios where each approach is suitable. 4. Firebase Integration in Flutter: Explain the process of integrating Firebase with a Flutter application. Discuss the benefits of using Firebase as a backend solution. Highlight the Firebase services commonly used in Flutter development and provide a brief overview of how data synchronization is achieved.
Roll No.	17
Name	Anurag Mahadev Gaiwal
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

-Anurag Gaiwal
-D15A
-17

TMPL LAB Assignment (1)

Q.1)

Ans: a) Key features of Flutter :

- Single codebase for multiple platforms - Flutter allows developers to write code and deploy it on both iOS & android.
- Hot reload:
  This enables developers to instantly see the results of code changes they make.
- Expressive UI:
  Developers have the ~~pr~~flexibility to create expressive and flexible UI.
- Integration with other tools:
  Flutter can easily integrate with other popular development tools & framework.

b) Advantages of Flutter:

a) Faster development:
Uses single codebase for multiple platforms.

- Consistent UI Across platforms.
  widgets provide a consistent look & feel across different platforms.
- cost-Efficiency:
  Development & maintaing single codebase. for iOS as well as Android.

c) Differs from Traditional Approach.

- Traditional approach uses a hierachial structure for UI components whereas Flutter uses a widget-based approach.

- Flutter compiles to native ARM code, providing performance comparable to native application.
- Hot Reloads allows to see changes made instantly.

Flutter's popularity is driven by increased productivity, a growing community, flexibility in UI design, cross-platform development capabilities and adoption by major companies.

Q.2)

Ans: Widget Tree :

- The widget tree is a hierarchical structure of widgets that defines the user interface of an application.
- Every visual element, from simple components to complex layout, is represented by a widget.
- widget can be categorised in 2 types:
  - Stateless widget : It is immutable & cannot change over time. Eg: Images, Text.
  - Stateful Wigdget: Widget that can change it's state over time, eg: buttons, forms.

Widget Composition:

- Widget composition in Flutter involves combining multiple simple widgets to create complex & compound widget.
- This compasability is a powerful concept that allows developers to built sophisticated user interfaces by nesting widgets within each other

Commonly used widgets:

1. Container:
   - A box model for padding, margin and decoration.
2. Column & Row:
   - Layout widgets for arranging children vertically or horizontally.
3. Stack:
   - Overlapping widgets, allowing them to be layered on top of each other.
4. List View:
   - A scrollable list of widgets.
5. AppBar:
   - A material design app bar typically at top of screen.
6. TextField:
   - An input field for user to enter text.
7. Button widget:
   - Interactive buttons for user actions.

Q.3)

Ans:
- State management is crucial in Flutter application because it ~~en~~involves managing the data that can change over time.
- Flutter is reactive, meaning the UI rebuilds when the underlying data changes.

Setstate	Provider	Riverpod
(1) Built-in Flutter method.	(1) External package named ('Provider')	(1) External package ('riverpod'
(2) Local state within a widget	(2) Global state within widget tree	(2) Global state with additional Features
(3) Limited scalability for large apps.	(3) Suitable for medium sized apps.	(3) Designed for large & complex apps
(4) May leads to code redundancy	(4) Balances simplicity & readability.	(4) Emphasizes readability & clean syntax.
(5) Testing can be more challenging	(5) Good testability support.	(5) Enhances testing experience
(6) Widely used, well established.	(6) Widely adopted well supported	(6) Gaining popularity growing community

Scenario where each is applicable

(1) setState:
- for small to moderately complex applicatior
- when managing local state within a widget

eg. Simple forms, UI components with local UI-spesific state.

(2) Provider
- For medium to large-sized applicatios.
- when a centralized state is needed accessible by multiple widgets.

eg. managing user authentication, theme changes or app-wide configuration.

(3) Riverpod:
- For large and complex applications.
- when testability and maintainability are top priorities.

eg. complex applications with multiple features dynamic UIs.

Q.4)

Ans: Integration:

1) Go to Firebase console and create a new project
2) Add Firebase SDK by including dependencies in pubspec.yaml.

```
dependencies:
    Firebase-core : ^version.
    Firebase_auth : ^version.
    cloud_firestore : ^version
```

3) Run Flutter pub get.

4) Initialise firebase by calling 'Firebase.initialiseApp()' in main.

```
import 'package:firebase_core/firebase_core.dart';
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(MyApp());
}
```

- Benefits of using firebase as backend.

(1) Real-time Database:
- Firebase offers a realtime NOSQL database.

(2) Authentication:
- Provides a secure and easy-to-implement solution for user authentication.

(3) Cloud firebase.
- Firebase's Cloud Firestore provides a secure and easy-to-implement scalable No SQL database that allows you to store and sync data in real time.

(4) Hosting.
- Firebase Hosting provides a simple & efficient way to deploy and host web applications.

- Data Synchronization :

(1) Real-time Database.

- When data changes on one client, it triggers event that automatically update data on other client's systems.

(2) Cloud Firebase :

- It notifies clients when data changes, allowing for seamless real-time updates.

(3) Authentication :

- If user sign in or out on one device, the authentication state is automatically reflected on other devices as well.

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.	17
Name	Anurag Mahadev Gaiwal
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

- Anurag Gaiwal
- D15A - 17
- Batch-A

PWA Assignment (2)

Q.1)

Ans: (1) Progressive web app (PWA) are a type of web application that combines the best features of Traditional web applications and native mobile application.

(2) PWA are built using web technologies like HTML, CSS and Javascript, but they can provide a user experience that is similar to a native app.

(3) Significance in modern web app development:

- PWA are significant in modern web development because they offer a number of advantages over traditional web applications and native mobile applications. They are:
  - Reachable: They can be accessed from any device with a web browser, regardless of the OS.
  - Reliable: They can work offline, thanks to service workers.
  - Installable: They can be installed on a user's home screen, just like a native app.
  - Engaging: They can provide a user experience that is indistinguishable from a native app.

(4) Key characteristics that differentiate PWAs from traditional mobile apps:

- PWA are built using web technologies, while native mobile apps are built using platform-specific programing language.
  eg. Swift for iOS, Java for Android.

- PWAs do not require installation from an app store, while native mobile apps ~~tra~~typically do.
- PWAs can work offline, while traditional mobile apps typically cannot.

Q.2)

Ans: (1) Responsive Web Apps Design (RWD) : It is web design approach that ensures that a website looks good and functions properly on all devices from desktop computers to tablets & smartphones.

(2) RWD is important for PWAs because it ensures that the app can be used on a variety of devices.

(3) Comparison of responsive, Fluid and adaptive web design approaches :

- Responsive web design : A responsive web design uses a ~~signate~~ single flexible layout that adapts to different screen sizes. This is the most common approach for PWAs.
- Fluid web design : A Fluid web design uses a flexible layout that scales to fit the width of the browser window. This can be a good option for websites that have a lot of content that needs to ~~the~~ be resized to fit different screens.
- Adaptive web design : An adaptive web design uses a different layout for different screen sizes. This can be a good option for websites that need to provide a different user experience for different devices.

Q.3)

Ans: ① Service workers : are scripts that run in the background of a web application. They can be used to enable features such as push notifications, offline functionality, & background synchronization.

② Lifecycle of service workers :

- Registrations : The service worker script is registered with the browser.
- Installation : The service worker script is downloaded and cached by the browser.
- Activation : The service worker takes control of a web page or group of web pages.

Q.4)

Ans: • IndexedDB : is a web API that provides a way for web applications to store data on the client side. Service workers can use IndexedDB to store data that needs to be available offline, such as cached content or user data.