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 **Ameya Chaudhary** 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

Project Title: **Roll No.**

Year/Sem/Class **:** D15A/D15B **A.Y.:** 23-24

Faculty Incharge : Mrs. Kajal Joseph.

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

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

Programme Outcomes: The graduate will be able to:

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

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

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

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

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

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

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

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

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

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

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

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

Program specific Outcomes

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

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

Project Title: **Roll No.**

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

Project Title: **Roll No.**

Index

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

Project Title: **Roll No.**

MAD & PWA Lab
Journal

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

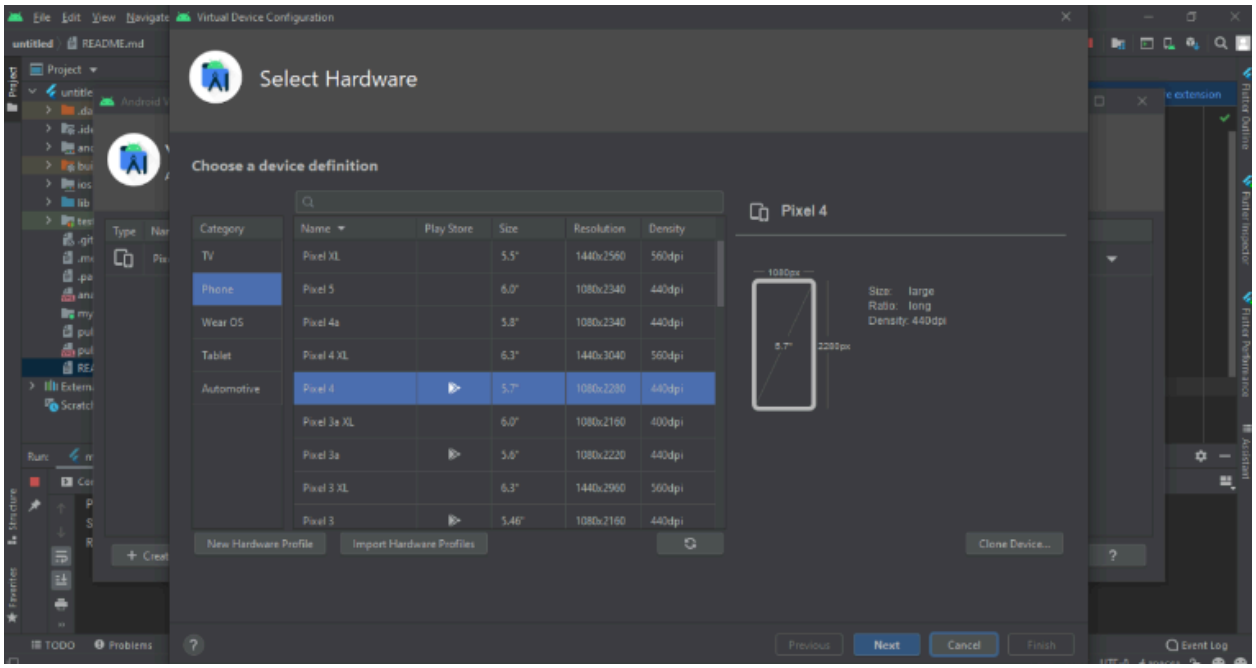
Project Title: **Roll No.**

EXPT - 1

AIM: To install and configure flutter environment

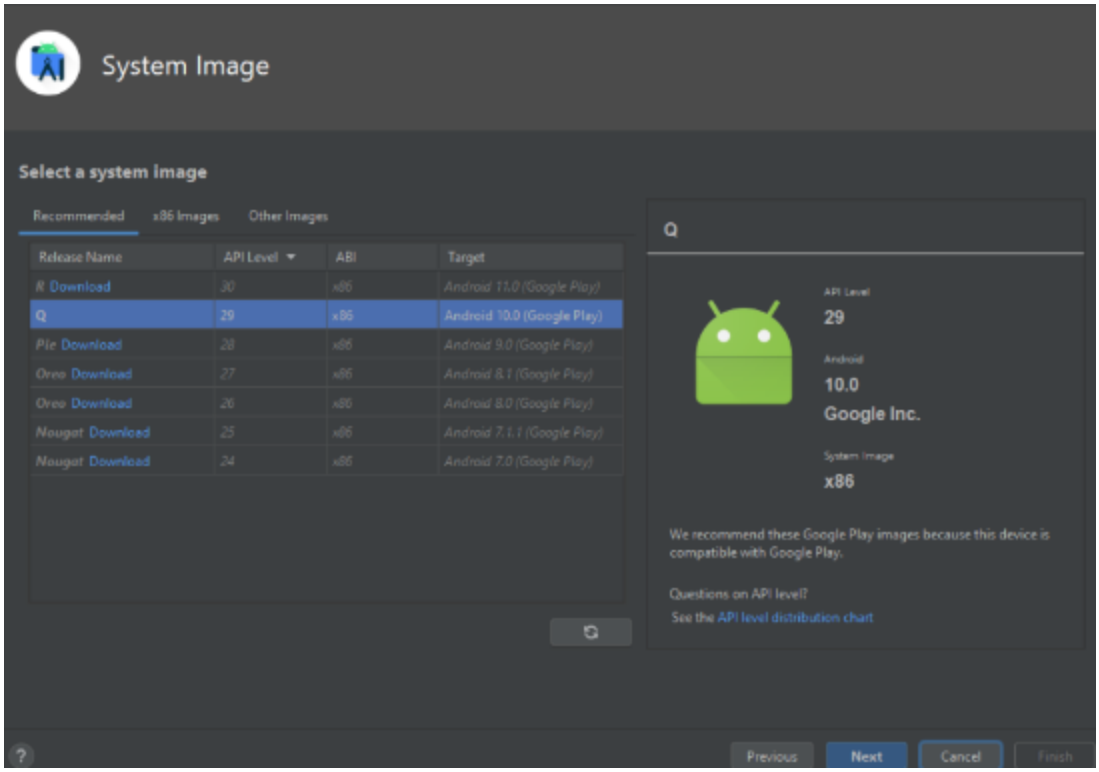
Step-1:Download Android studio for Windows and sdk

Step-2:Create a new virtual device



Download the ANdroid version

Project Title: **Roll No.**



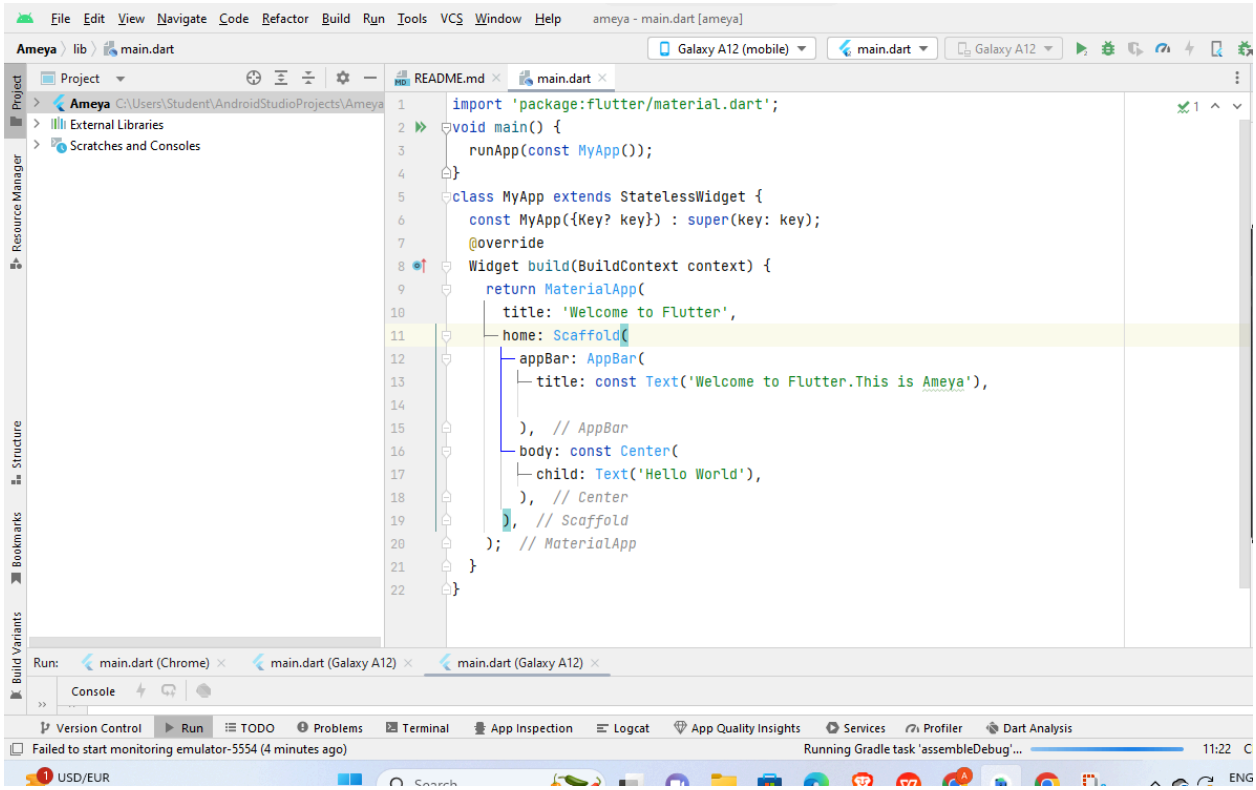
Create the app

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.

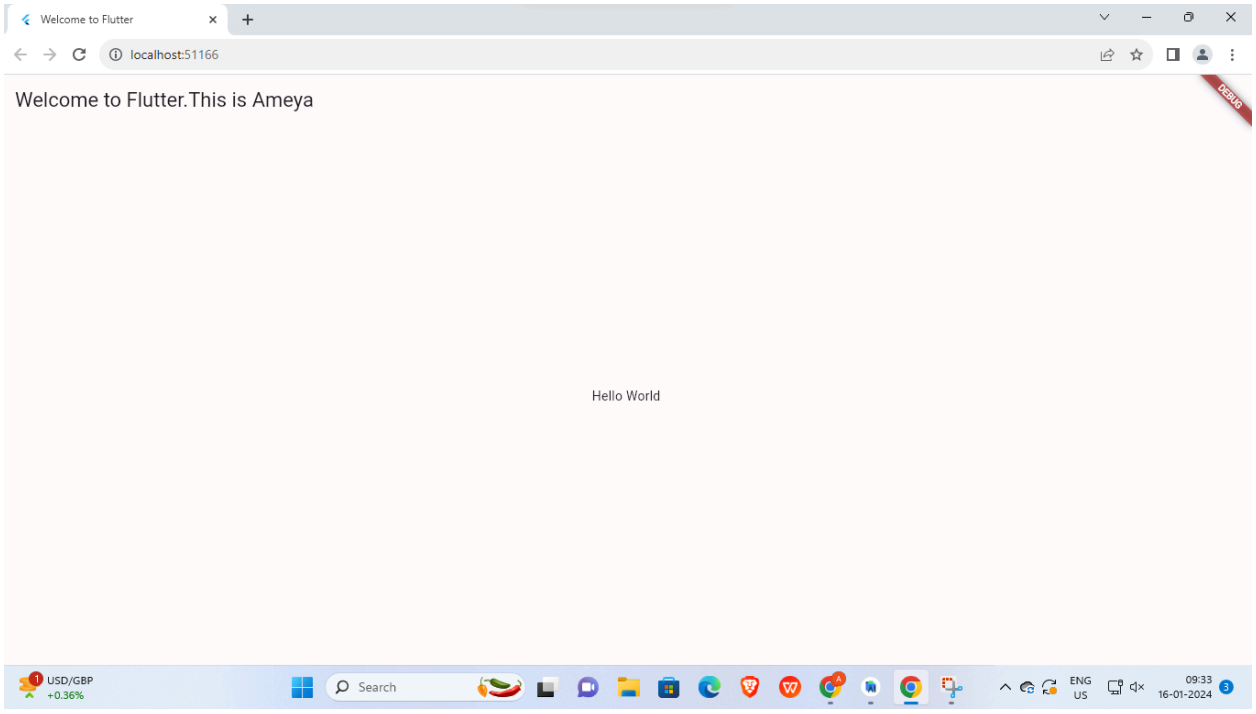
Step 2: Run the app

1. Locate the main Android Studio toolbar:

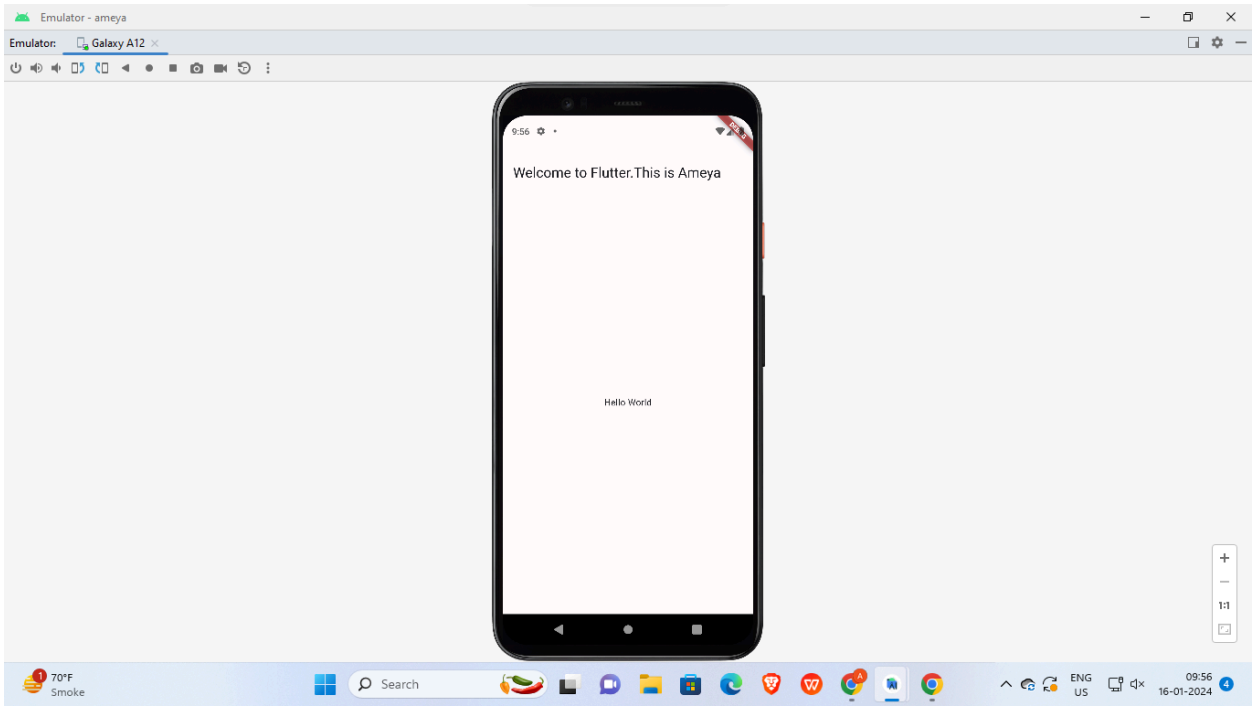
Project Title: **Roll No.**



```
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.This is Ameya'),

        ),  // AppBar
        body: const Center(
          child: Text('Hello World'),
        ),  // Center
      ),  // Scaffold
    );  // MaterialApp
  }
}
```



Project Title: **Roll No.**



Conclusion:

Thus,android studio and flutter is installed and app is created successfully

Project Title: **Roll No.**

MAD & PWA Lab
Journal

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

Project Title: **Roll No.**

MAD PWA LAB

EXPT - 2

Aim: To design flutter UI by including common widgets.

Theory:

Flutter App Customization

In this Flutter app, we have customized the appearance of the entire screen using the `MaterialApp` and `Scaffold` widgets. The goal is to create a visually appealing layout with a black background, a centered app title in white, and an image logo just below the title.

Theme Customization

The `MaterialApp` widget allows us to customize the overall theme of the app. We set the `scaffoldBackgroundColor` to black to create a dark background. Additionally, we customized the app bar theme (`AppBarTheme`) by setting the background color to black and specifying a white text color for the title using `titleTextStyle`.

```
theme: ThemeData(
  scaffoldBackgroundColor: Colors.black,
  appBarTheme: AppBarTheme(
    backgroundColor: Colors.black,
    titleTextStyle: TextStyle(
      color: Colors.white,
      fontSize: 24.0,
      fontWeight: FontWeight.bold,
    ),
  ),
),
```

APP TITLE AND LOGO

The app title is placed at the center of the app bar using the centerTitle property of the AppBar widget. The title is styled with a white color, a font size of 24.0, and a bold font weight.

To position the logo just below the title, we use a Column widget in the body of the Scaffold. Inside the Column, we include a SizedBox to create space between the title and the logo.
Adjust the height of the SizedBox to fine-tune the spacing between the title and the logo based on your design preferences.
This Flutter code provides a clean and organized way to customize the appearance of the entire screen, offering a visually appealing and well-structured user interface.

CODE:

```
import 'package:flutter/material.dart';
void main() {
 runApp(const MyApp());
}
class MyApp extends StatelessWidget {
 const MyApp({Key? key}) : super(key: key);
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     title: 'YouTube Music',
     theme: ThemeData(
       scaffoldBackgroundColor: Colors.black,
       appBarTheme: AppBarTheme(
         backgroundColor: Colors.black,
         titleTextStyle: TextStyle(
           color: Colors.red,
           fontSize: 34.0,
           fontWeight: FontWeight.bold,
         ),
       ),
     ),
     home: Scaffold(
       appBar: AppBar(
         title: const Text('YouTube Music'),
         centerTitle: true,
       ),
```

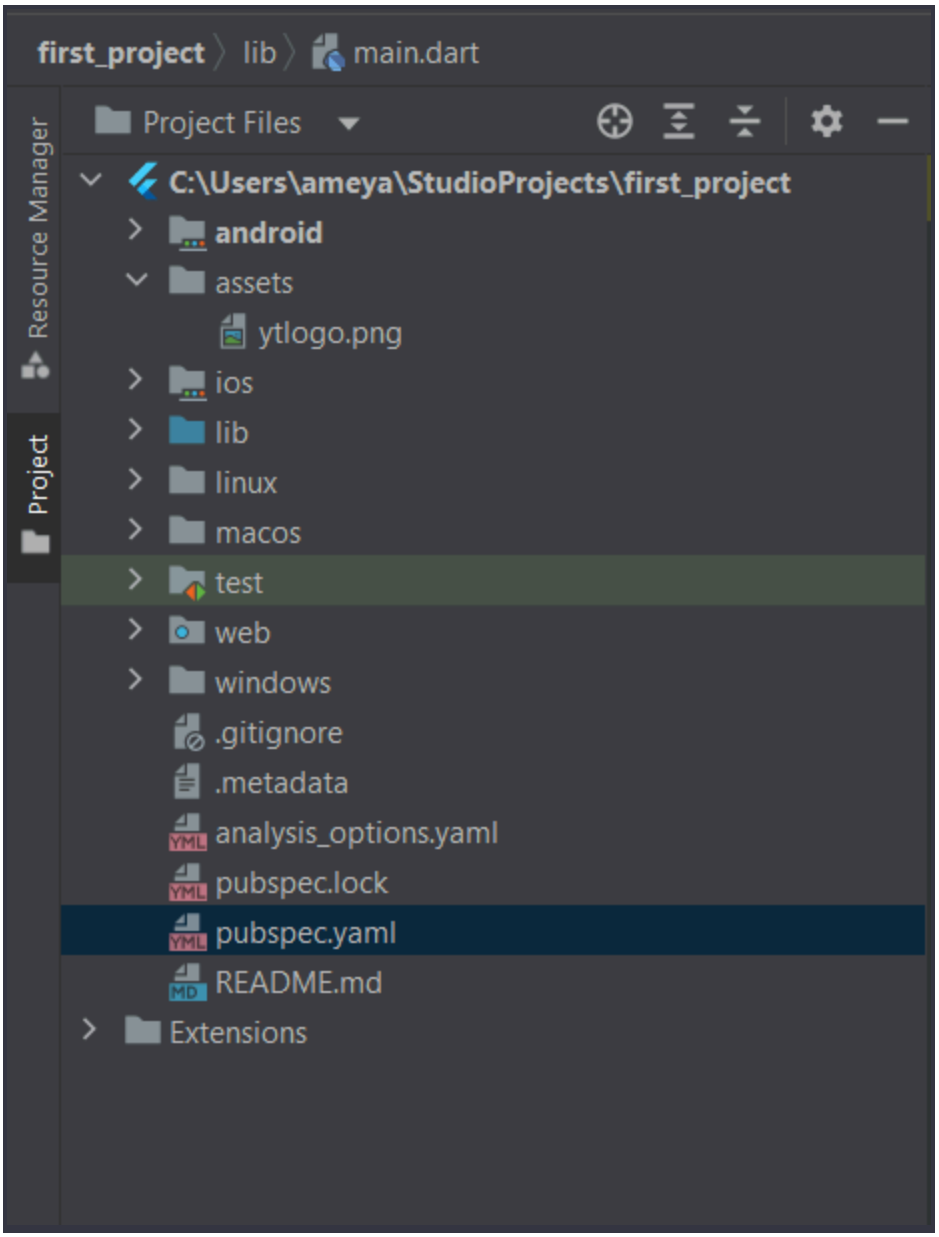
```
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            SizedBox(height: 30), // Adjust the height as needed
            Image.asset('assets/ytlogo.png'),
          ],
        ),
      ),
    ),
  );
 }
}
```

ASSETS:

-assets/ytlogo.png

FILE STRUCTURE:

Project Title: **Roll No.**



Project Title: **Roll No.**

README.md | main.dart | pubspec.yaml

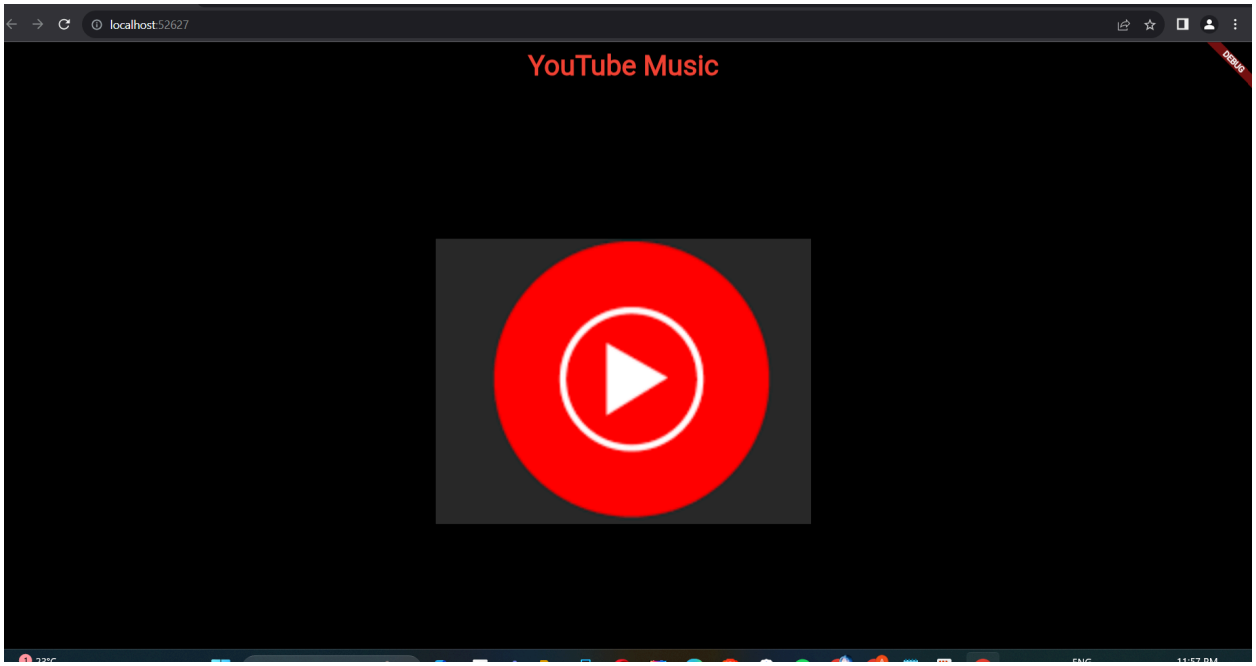
Pubspec has been edited — Get dependencies | Upgrade dependencies | Ignore

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

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

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'YouTube Music',
      theme: ThemeData(
        scaffoldBackgroundColor: Colors.black,
        appBarTheme: AppBarTheme(
          backgroundColor: Colors.black,
          titleTextStyle: TextStyle(
            color: Colors.red,
            fontSize: 34.0,
            fontWeight: FontWeight.bold,
          ),  // TextStyle
        ),  // AppBarTheme
      ),  // ThemeData
      home: Scaffold(
        appBar: AppBar(
          title: const Text('YouTube Music'),
          centerTitle: true,
        ),  // AppBar
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              SizedBox(height: 30), // Adjust the height as needed
              Image.asset('assets/ytlogo.png'),
            ],
          ),  // Column
        ),  // Center
      ),  // Scaffold
    );  // MaterialApp
  }
}
```

Project Title: **Roll No.**

OUTPUT:

Project Title: **Roll No.**



–

Project Title: **Roll No.**

MAD & PWA Lab
Journal

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

Project Title: **Roll No.**

Ameya Chaudhary
D15A
09

MAD PWA LAB

EXPT - 3

Aim: To include icons,images and fonts in flutter app.

Theory:

Overall Design Philosophy:

- User-Centric Design: Prioritize user experience by making navigation intuitive and content easily accessible.
- Consistent Branding: Use consistent colors, fonts, and styling to create a cohesive and branded look.
- Visual Hierarchy: Arrange elements in a way that guides the user's attention from most important to least important.

AppBar and Logo:

- Logo Placement: Position the logo prominently at the top, ensuring it's recognizable and reinforces the app's identity.
- AppBar Styling: Consider using a custom AppBar with a clean design that complements the overall aesthetic.

Song and Artist Images:

- Image Grid: Organize song and artist images in a visually appealing grid, creating a sense of order and simplicity.
- Image Quality: Ensure high-quality images for songs and artists, optimizing them for different screen sizes and resolutions.
- Consistent Sizing: Maintain consistency in image sizes for a polished and professional appearance.

Bottom Navigation Bar:

- Iconography: Use clear and recognizable icons for home, search, and playlist, ensuring users easily understand their purpose.
- Active State Indicator: Highlight the active page with a distinct indicator to provide visual feedback to users.
- Navigation Logic: Implement navigation logic to switch between different sections seamlessly.

Color Scheme:

- Theme Colors: Choose a color scheme that aligns with the app's branding and creates a visually pleasing atmosphere.
- It is Black and Red.
- Contrast and Readability: Ensure sufficient contrast between text and background colors for optimal readability.

CODE:

```
import 'package:flutter/material.dart';
import 'search_screen.dart'; // Import the SearchScreen
import 'song_details_screen.dart'; // Import the SongDetailsScreen

class HomeScreen extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text(
        'YouTube Music',
        style: TextStyle(color: Colors.red, fontSize: 34),
      ),
      centerTitle: true,
      backgroundColor: Colors.black,
    ),
    body: SingleChildScrollView(
      child: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Center(
              child: Text(
                'Welcome to YouTube Music!',
                style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold, color: Colors.pink),
              ),
            ),
            SizedBox(height: 10),
            Image.asset(
              'assets/welcome_image.png',
              height: 240,
              width: double.infinity,
              fit: BoxFit.cover,
            ),
```

```
          SizedBox(height: 20),
          Text(
            'Suggested Songs:',
            style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold, color: Colors.white),
          ),
          SizedBox(height: 25),
          Row(
            mainAxisAlignment: MainAxisAlignment.spaceBetween,
            children: [
              _buildSuggestedItem(context, 'Alone', 'assets/song1.png'),
              _buildSuggestedItem(context, 'Kesariya', 'assets/song2.png'),
              _buildSuggestedItem(context, 'We own it', 'assets/song3.png'),
            ],
          ),
          SizedBox(height: 20),
          Text(
            'Suggested Artists:',
            style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold, color: Colors.white),
          ),
          SizedBox(height: 25),
          Row(
            mainAxisAlignment: MainAxisAlignment.spaceBetween,
            children: [
              _buildSuggestedItem(context, 'Eminem', 'assets/eminem.png'),
              _buildSuggestedItem(context, 'Arjit Singh', 'assets/arjit.png'),
              _buildSuggestedItem(context, 'Selena Gomez', 'assets/selena.png'),
            ],
          ),
        ],
      ),
    ),
  ),
  bottomNavigationBar: BottomNavigationBar(
    items: [
      BottomNavigationBarItem(
        icon: GestureDetector(
          onTap: () {
            // Navigate to HomeScreen (Optional)
          },
          child: Image.asset('assets/home.png', height: 24, width: 24, color: Colors.red),
        ),
        label: 'Home',
      ),
      BottomNavigationBarItem(
```

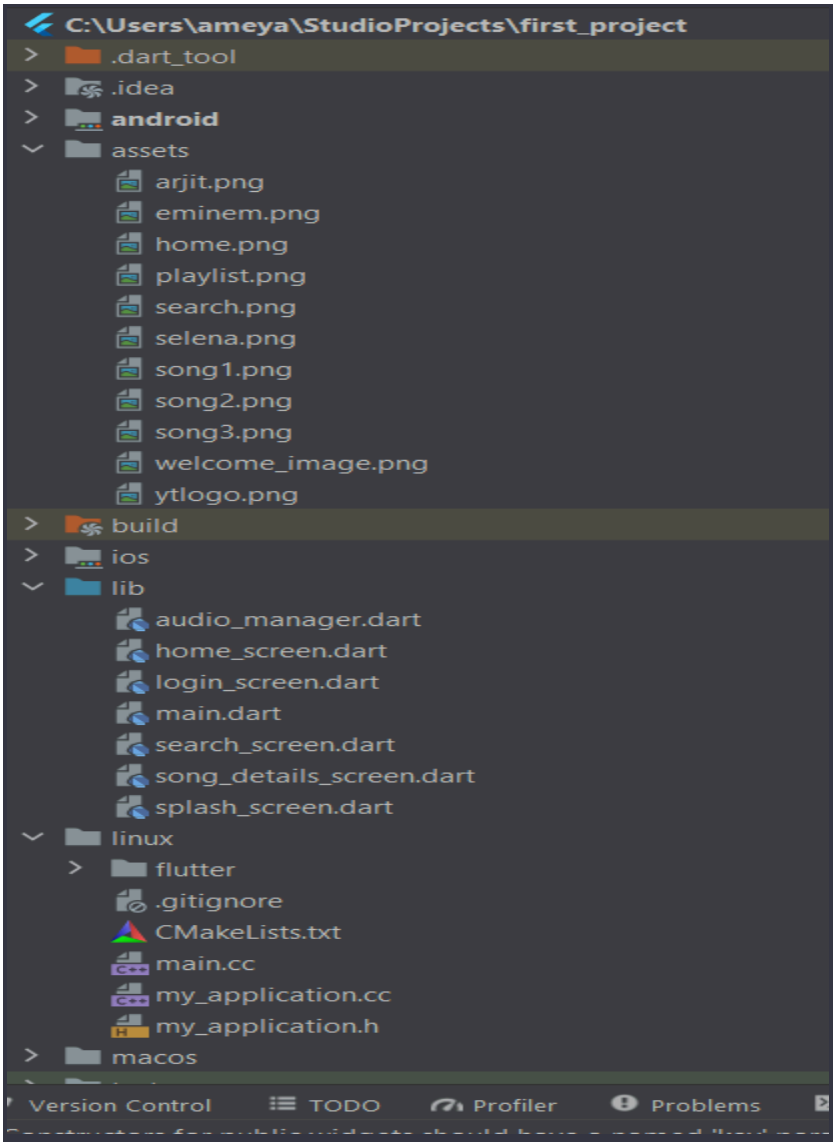
```
          icon: GestureDetector(
            onTap: () {
              Navigator.push(context, MaterialPageRoute(builder: (context) => SearchScreen()));
            },
            child: Image.asset('assets/search.png', height: 24, width: 24, color: Colors.grey),
          ),
          label: 'Search',
        ),
        BottomNavigationBarItem(
          icon: Image.asset('assets/playlist.png', height: 24, width: 24, color: Colors.grey),
          label: 'Library',
        ),
      ],
      backgroundColor: Colors.deepPurple,
      selectedItemColor: Colors.red,
      unselectedItemColor: Colors.grey,
    ),
  );
}

Widget _buildSuggestedItem(BuildContext context, String name, String imagePath) {
  return GestureDetector(
    onTap: () {
      // Navigate to SongDetailsScreen when the image is tapped
      Navigator.push(
        context,
        MaterialPageRoute(
          builder: (context) => SongDetailsScreen(songTitle: name, songImagePath: imagePath),
        ),
      );
    },
    child: Column(
      children: [
        Image.asset(
          imagePath,
          height: 80,
          width: 80,
          fit: BoxFit.cover,
        ),
        SizedBox(height: 5),
        Text(
          name,
          style: TextStyle(color: Colors.white),
        ),
```

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

ASSETS:

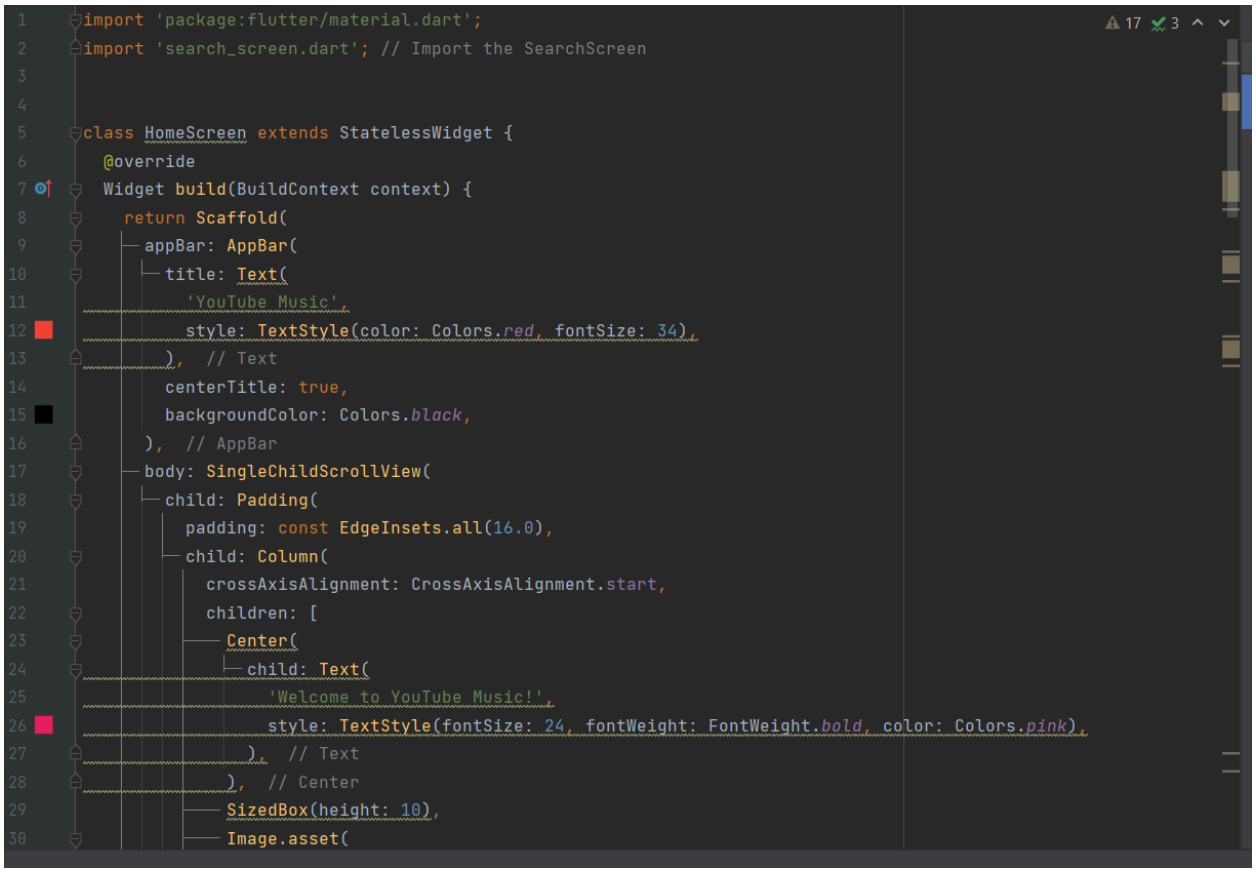
-assets/ytlogo.png
-assets/song1.png
-assets/song2.png
-assets/song3.png
-assets/selena.png
-assets/arjit.png

FILE STRUCTURE:



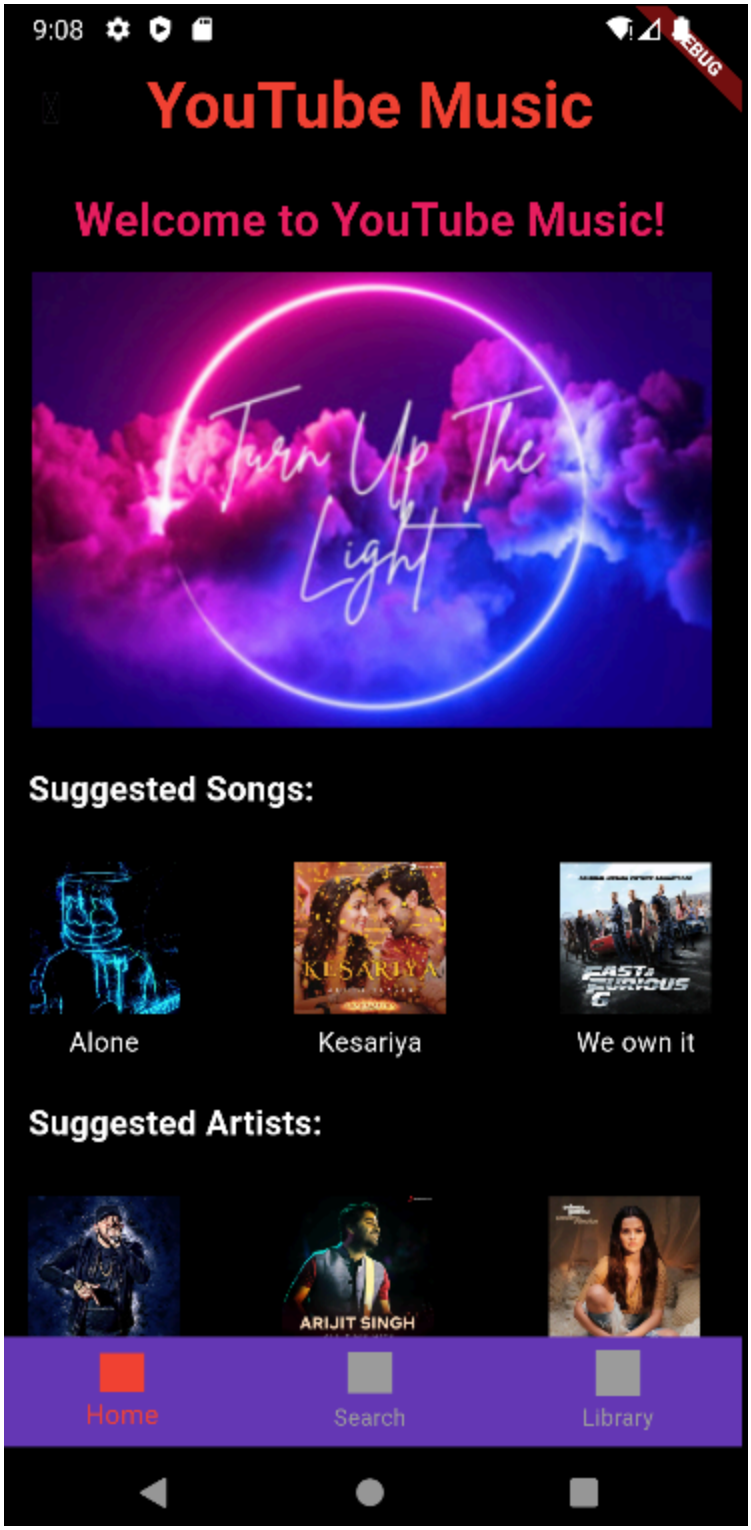
Project Title: **Roll No.**

OUTPUT:



```dart
import 'package:flutter/material.dart';
import 'search_screen.dart'; // Import the SearchScreen


class HomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(
          'YouTube Music',
          style: TextStyle(color: Colors.red, fontSize: 34),
        ),  // Text
        centerTitle: true,
        backgroundColor: Colors.black,
      ),  // AppBar
      body: SingleChildScrollView(
        child: Padding(
          padding: const EdgeInsets.all(16.0),
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Center(
                child: Text(
                  'Welcome to YouTube Music!',
                  style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold, color: Colors.pink),
                ),  // Text
              ),  // Center
              SizedBox(height: 10),
              Image.asset(
```

Project Title: **Roll No.**



Project Title: **Roll No.**

MAD & PWA Lab
Journal

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

Project Title: **Roll No.**

MAD PWA LAB

EXPT - 4

Aim: To create an interactive form using form widget

Theory:

Logo Placement:

- Branding: Position the YouTube Music logo prominently at the top of the login form to reinforce brand identity.
- Recognition: Ensure the logo is easily recognizable, providing users with a visual cue about the platform they are accessing.

Input Fields:

- Email Field:
  - Purposeful Labels: Clearly label the email input field to convey its purpose.
  - Input Validation: Implement validation to ensure that users enter a valid email address.
  - Auto-complete: Enable auto-complete for convenience, making the login process faster for returning users.
- Password Field:
  - Security: Design the password field with security in mind, hiding characters and providing an option to reveal them.
  - Strength Indicator (Optional): Consider adding a password strength indicator to guide users in creating secure passwords.

Login Button:

- Visibility: Make the login button stand out by using a distinct color or styling, encouraging users to proceed.
- Loading State: Implement a loading state to provide feedback to users while the authentication process is underway.

Form Styling:

- Consistency: Maintain a consistent color scheme and typography throughout the form for a cohesive and polished appearance.
- Whitespace: Use appropriate spacing between elements to improve readability and prevent visual clutter.

CODE:

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

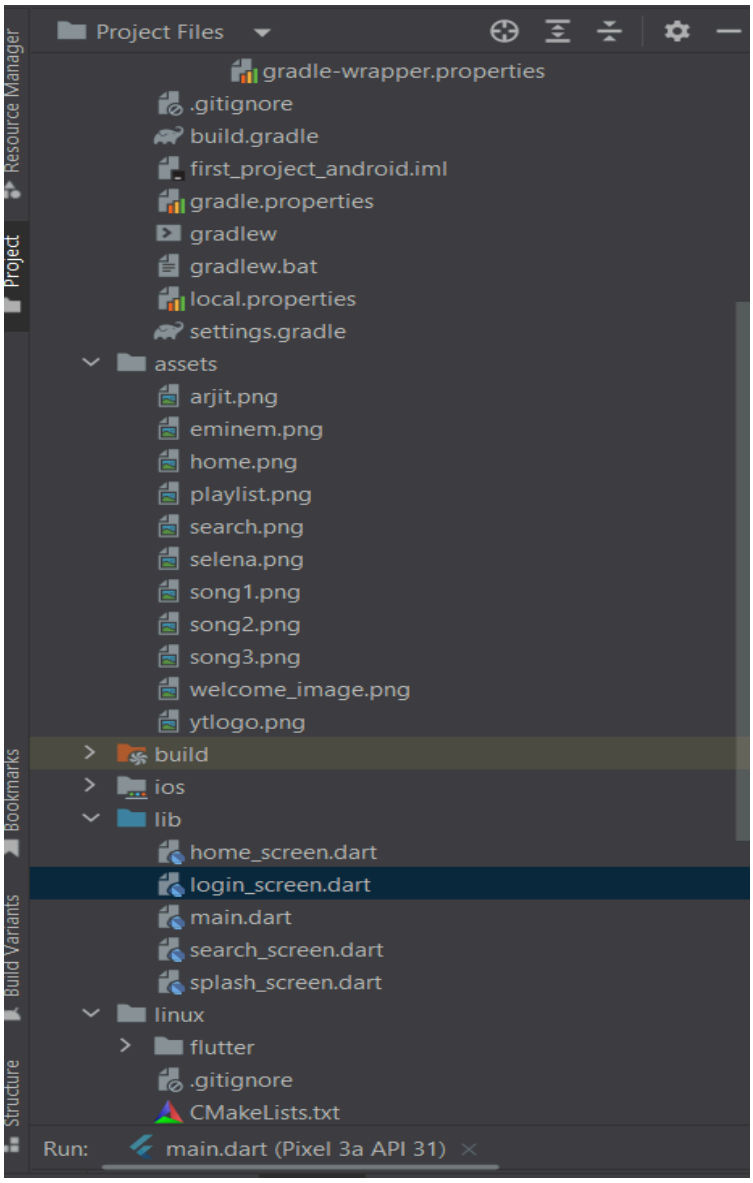
Project Title: **Roll No.**

```
import 'home_screen.dart'; // Import the HomeScreen

class LoginScreen extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Login'),
      centerTitle: true,
    ),
    body: Container(
      color: Colors.white, // Set the background color to white
      padding: const EdgeInsets.all(16.0),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Image.asset('assets/ytlogo.png', height: 100, width: 100),
          SizedBox(height: 20),
          TextField(
            decoration: InputDecoration(labelText: 'Email'),
          ),
          SizedBox(height: 10),
          TextField(
            decoration: InputDecoration(labelText: 'Password'),
            obscureText: true,
          ),
          SizedBox(height: 20),
          ElevatedButton(
            onPressed: () {
              // Handle login logic
              // For now, let's navigate to the HomeScreen on login success
              Navigator.pushReplacement(
                context,
                MaterialPageRoute(builder: (context) => HomeScreen()),
              );
            },
            style: ElevatedButton.styleFrom(
              primary: Colors.red,
            ),
            child: Text('Login'),
          ),
        ],
      ),
    ),
```

```
   );
 }
}
```

ASSETS:
-assets/ytlogo2.png

FILE STRUCTURE:

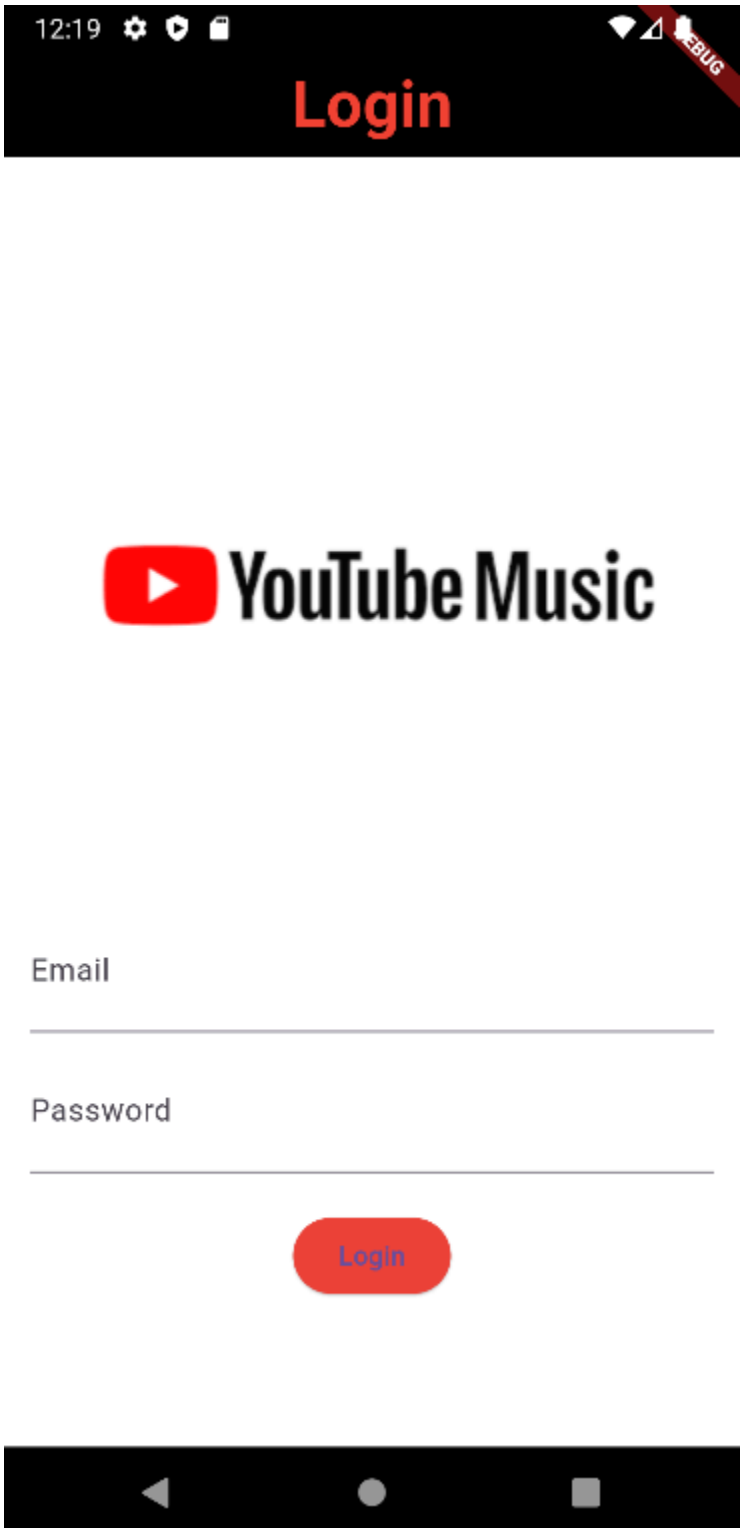


Project Title: **Roll No.**

```dart
import 'package:flutter/material.dart';
import 'home_screen.dart'; // Import the HomeScreen

class LoginScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Login'),
        centerTitle: true,
      ),  // AppBar
      body: Container(
        color: Colors.white, // Set the background color to white
        padding: const EdgeInsets.all(16.0),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Image.asset('assets/ytlogo.png', height: 100, width: 100),
            SizedBox(height: 20),
            TextField(
              decoration: InputDecoration(labelText: 'Email'),
            ),  // TextField
            SizedBox(height: 10),
            TextField(
              decoration: InputDecoration(labelText: 'Password'),
              obscureText: true,
            ),  // TextField
            SizedBox(height: 20),
            ElevatedButton(
              // Handle login logic
              // For now, let's navigate to the HomeScreen on login success
              Navigator.pushReplacement(
                context,
                MaterialPageRoute(builder: (context) => HomeScreen()),
              );
            },
            style: ElevatedButton.styleFrom(
              primary: Colors.red,
            ),
            child: Text('Login'),
            ),  // ElevatedButton
          ],
        ),  // Column
      ),  // Container
    );  // Scaffold
  }
}
```

Project Title: **Roll No.**

OUTPUT:



Project Title: **Roll No.**

MAD & PWA Lab
Journal

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

Project Title: **Roll No.**

MAD PWA
EXPT-5

AIM: To apply navigation ,routing and gestures in app

THEORY:

1. Navigator:

Flutter uses a Navigator to manage a stack of routes or pages. Each route represents a screen or page in your application.

2. Routes:

A route in Flutter is a way to navigate to a different screen or page. It is represented by the PageRoute class and typically involves a transition animation.

3. Named Routes:

Flutter supports named routes, where each route is associated with a unique name. Named routes make it easy to reference specific screens throughout your app.

4. Navigator.push:

To navigate to a new screen, you can use Navigator.push method. This adds a new route to the stack and transitions to the specified screen.

5. Navigator.pop:

To go back to the previous screen, you can use Navigator.pop. This removes the current route from the stack and returns to the previous screen.

6. Sending Data to a New Screen:

You can pass data to a new screen by providing arguments to the constructor of the new screen.

7. Named Routes Example:

Define named routes in the MaterialApp

8. Route Arguments with Named Routes:

You can pass arguments with named routes using ModalRoute.of:

dart
final String data = ModalRoute.of(context)!.settings.arguments as String;

9. Navigator.pushReplacement:

If you want to replace the current screen with a new one, you can use Navigator.pushReplacement. This is useful when, for example, you don't want the user to go back to the login screen after successfully logging in.
dart

These are the fundamental concepts of navigation and routing in Flutter. Understanding these concepts will help you efficiently manage the flow of your app and navigate between different screens.

CODE:

//home_screen.dart

```
import 'package:flutter/material.dart';
import 'search_screen.dart';
import 'song_details_screen.dart';
import 'library_screen.dart'; // Import the LibraryScreen

class HomeScreen extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(
       title: Text(
         'YouTube Music',
         style: TextStyle(color: Colors.red, fontSize: 34),
       ),
       centerTitle: true,
       backgroundColor: Colors.black,
     ),
     body: SingleChildScrollView(
       child: Padding(
         padding: const EdgeInsets.all(16.0),
         child: Column(
           crossAxisAlignment: CrossAxisAlignment.start,
           children: [
             Center(
               child: Text(
                 'Welcome to YouTube Music!',
```

```
    style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold, color: Colors.pink),
   ),
  ),
  SizedBox(height: 10),
  Image.asset(
   'assets/welcome_image.png',
   height: 240,
   width: double.infinity,
   fit: BoxFit.cover,
  ),
  SizedBox(height: 20),
  Text(
   'Suggested Songs:',
   style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold, color: Colors.white),
  ),
  SizedBox(height: 25),
  Row(
   mainAxisAlignment: MainAxisAlignment.spaceBetween,
   children: [
    _buildSuggestedItem(context, 'Alone', 'assets/song1.png'),
    _buildSuggestedItem(context, 'Kesariya', 'assets/song2.png'),
    _buildSuggestedItem(context, 'We own it', 'assets/song3.png'),
   ],
  ),
  SizedBox(height: 20),
  Text(
   'Suggested Artists:',
   style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold, color: Colors.white),
  ),
  SizedBox(height: 25),
  Row(
   mainAxisAlignment: MainAxisAlignment.spaceBetween,
   children: [
    _buildSuggestedItem(context, 'Eminem', 'assets/eminem.png'),
    _buildSuggestedItem(context, 'Arjit Singh', 'assets/arjit.png'),
    _buildSuggestedItem(context, 'Selena Gomez', 'assets/selena.png'),
   ],
  ),
  SizedBox(height: 20),
  ElevatedButton(
   onPressed: () {
    // Navigate to LibraryScreen when the Library button is tapped
    Navigator.push(
     context,
```

```
                MaterialPageRoute(builder: (context) => LibraryScreen()),
              );
            },
            style: ElevatedButton.styleFrom(
              primary: Colors.red,
            ),
            child: Text('Go to Library'),
          ),
        ],
      ),
    ),
  ),
  bottomNavigationBar: BottomNavigationBar(
    items: [
      BottomNavigationBarItem(
        icon: GestureDetector(
          onTap: () {
            // Navigate to HomeScreen (Optional)
          },
          child: Image.asset('assets/home.png', height: 24, width: 24, color: Colors.red),
        ),
        label: 'Home',
      ),
      BottomNavigationBarItem(
        icon: GestureDetector(
          onTap: () {
            Navigator.push(context, MaterialPageRoute(builder: (context) => SearchScreen()));
          },
          child: Image.asset('assets/search.png', height: 24, width: 24, color: Colors.grey),
        ),
        label: 'Search',
      ),
      BottomNavigationBarItem(
        icon: Image.asset('assets/playlist.png', height: 24, width: 24, color: Colors.grey),
        label: 'Library',
      ),
    ],
    backgroundColor: Colors.deepPurple,
    selectedItemColor: Colors.red,
    unselectedItemColor: Colors.grey,
  ),
 );
}
```

```
 Widget _buildSuggestedItem(BuildContext context, String name, String imagePath) {
   return GestureDetector(
     onTap: () {
       Navigator.push(
         context,
         MaterialPageRoute(
           builder: (context) => SongDetailsScreen(songTitle: name, songImagePath: imagePath),
         ),
       );
     },
     child: Column(
       children: [
         Image.asset(
           imagePath,
           height: 80,
           width: 80,
           fit: BoxFit.cover,
         ),
         SizedBox(height: 5),
         Text(
           name,
           style: TextStyle(color: Colors.white),
         ),
       ],
     ),
   );
 }
}
```

//search_screen.dart

```
import 'package:flutter/material.dart';
import 'home_screen.dart';
import 'library_screen.dart'; // Import the LibraryScreen

class SearchScreen extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(
       title: Text(
         'Search',
         style: TextStyle(color: Colors.white, fontSize: 24),
       ),
       centerTitle: true,
```

```
      backgroundColor: Colors.black,
    ),
    body: SingleChildScrollView(
      child: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            _buildSearchBar(),

            // Add the top trending songs title
            Text(
              'Top Trending Songs',
              style: TextStyle(fontSize: 26, fontWeight: FontWeight.bold, color: Colors.red),
            ),

            // Add the three images with their song names
            _buildSuggestedItem('Song 1', 'assets/song1.png'),
            _buildSuggestedItem('Song 2', 'assets/song2.png'),
            _buildSuggestedItem('Song 3', 'assets/song3.png'),
          ],
        ),
      ),
    ),

    bottomNavigationBar: BottomNavigationBar(
      items: [
        BottomNavigationBarItem(
          icon: GestureDetector(
            onTap: () {
              Navigator.pushReplacement(
                context,
                MaterialPageRoute(builder: (context) => HomeScreen()),
              );
            },
            child: Image.asset('assets/home.png', height: 24, width: 24, color: Colors.grey),
          ),
          label: 'Home',
        ),
        BottomNavigationBarItem(
          icon: Image.asset('assets/search.png', height: 24, width: 24, color: Colors.red),
          label: 'Search',
        ),
        BottomNavigationBarItem(
```

```
          icon: GestureDetector(
            onTap: () {
              Navigator.push(context, MaterialPageRoute(builder: (context) => LibraryScreen()));
            },
            child: Image.asset('assets/playlist.png', height: 24, width: 24, color: Colors.grey),
          ),
          label: 'Library',
        ),
      ],
      backgroundColor: Colors.deepPurple,
      selectedItemColor: Colors.red,
      unselectedItemColor: Colors.grey,
    ),
  );
}

Widget _buildSearchBar() {
  return Container(
    margin: EdgeInsets.symmetric(vertical: 16),
    padding: EdgeInsets.symmetric(horizontal: 16),
    decoration: BoxDecoration(
      color: Colors.grey[900],
      borderRadius: BorderRadius.circular(30),
    ),
    child: TextField(
      style: TextStyle(color: Colors.white),
      decoration: InputDecoration(
        hintText: 'Search for songs...',
        hintStyle: TextStyle(color: Colors.grey),
        suffixIcon: Icon(Icons.search, color: Colors.white),
        border: InputBorder.none,
      ),
    ),
  );
}

Widget _buildSuggestedItem(String name, String imagePath) {
  return Row(
    children: [
      Image.asset(
        imagePath,
        height: 80,
        width: 80,
        fit: BoxFit.cover,
```

```
      ),
      SizedBox(width: 16),
      Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Text(
            name,
            style: TextStyle(color: Colors.white),
          ),
        ],
      ),
    ],
  );
 }
}
```

//library_screen.dart

```
import 'package:flutter/material.dart';
import 'home_screen.dart';
import 'search_screen.dart';
import 'song_details_screen.dart';
import 'package:provider/provider.dart';

class LibraryScreen extends StatelessWidget {
 // Map to associate each song title with its image path
 final Map<String, String> songImageMap = {
   'Song 1': 'assets/song1.png',
   'Song 2': 'assets/song2.png',
   'Song 3': 'assets/song3.png',
   // Add more entries as needed
 };

 @override
 Widget build(BuildContext context) {
   List<String> likedSongs = Provider.of<LikedSongs>(context).likedSongs;

   return Scaffold(
     appBar: AppBar(
       title: Text(
         'Library',
         style: TextStyle(color: Colors.red, fontSize: 24),
       ),
       centerTitle: true,
       backgroundColor: Colors.black,
```

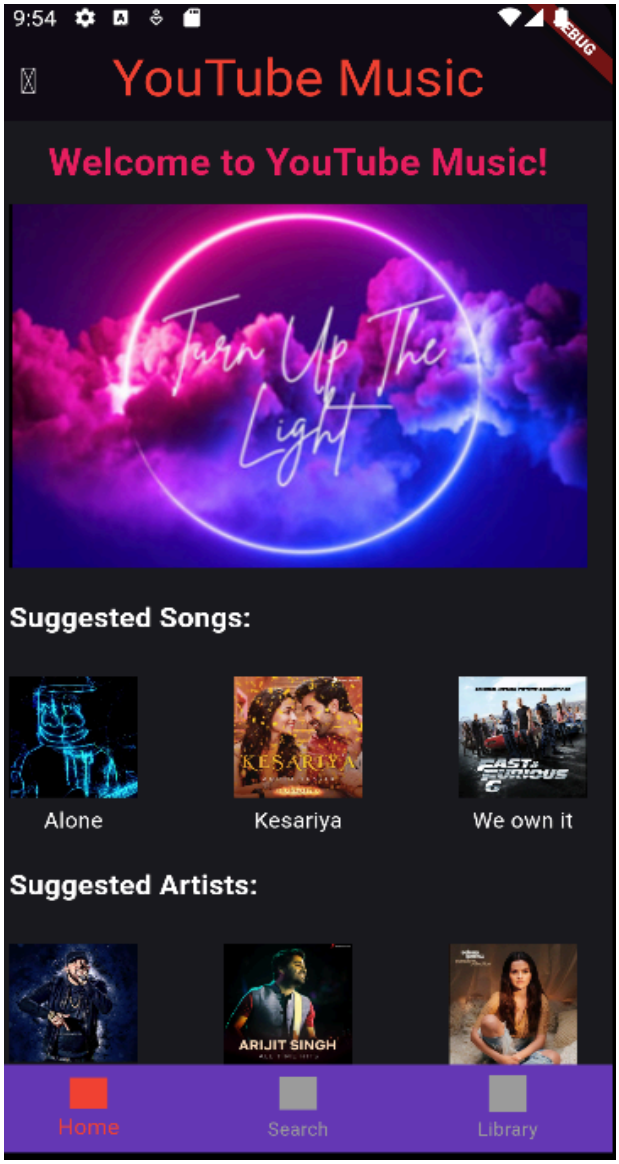
```
    ),
    body: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Text(
            'Liked Songs',
            style: TextStyle(fontSize: 26, fontWeight: FontWeight.bold, color: Colors.red),
          ),
          // Display liked songs here
          Expanded(
            child: ListView.builder(
              itemCount: likedSongs.length,
              itemBuilder: (context, index) {
                // Extract the song details from the likedSongs list
                String songTitle = likedSongs[index];
                String songImagePath = getSongImagePath(songTitle); // Get the image path based
on the song title

                return ListTile(
                  leading: Image.asset(
                    songImagePath,
                    height: 40,
                    width: 40,
                    fit: BoxFit.cover,
                  ),
                  title: Text(songTitle),
                  // Add more details if needed
                );
              },
            ),
          ),
        ],
      ),
    ),
    bottomNavigationBar: BottomNavigationBar(
      items: [
        BottomNavigationBarItem(
          icon: GestureDetector(
            onTap: () {
              Navigator.pushReplacement(
                context,
                MaterialPageRoute(builder: (context) => HomeScreen()),
```

```
            );
          },
          child: Image.asset('assets/home.png', height: 24, width: 24, color: Colors.grey),
        ),
        label: 'Home',
      ),
      BottomNavigationBarItem(
        icon: GestureDetector(
          onTap: () {
            Navigator.push(context, MaterialPageRoute(builder: (context) => SearchScreen()));
          },
          child: Image.asset('assets/search.png', height: 24, width: 24, color: Colors.grey),
        ),
        label: 'Search',
      ),
      BottomNavigationBarItem(
        icon: Image.asset('assets/playlist.png', height: 24, width: 24, color: Colors.grey),
        label: 'Library',
      ),
    ],
    backgroundColor: Colors.deepPurple,
    selectedItemColor: Colors.red,
    unselectedItemColor: Colors.grey,
  ),
 );
}

// Helper method to get the image path based on the song title
String getSongImagePath(String songTitle) {
  // Use the map to get the image path based on the song title
  return songImageMap[songTitle] ?? 'assets/song1.png';
}
}
```

OUTPUT:

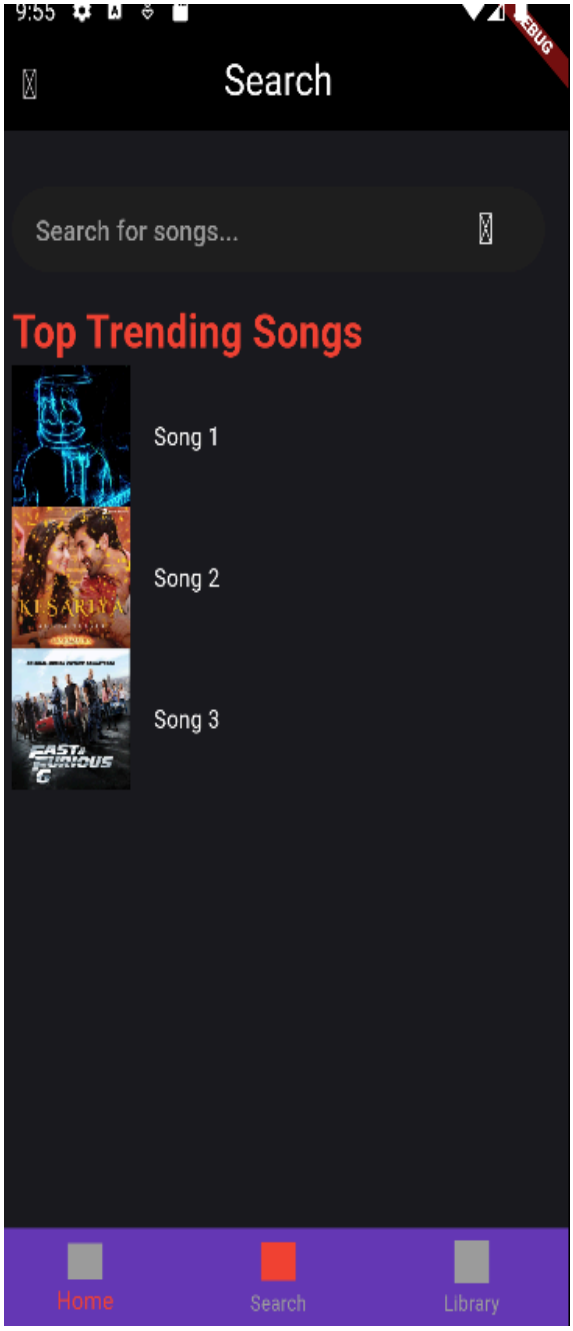
home_screen

Project Title: **Roll No.**



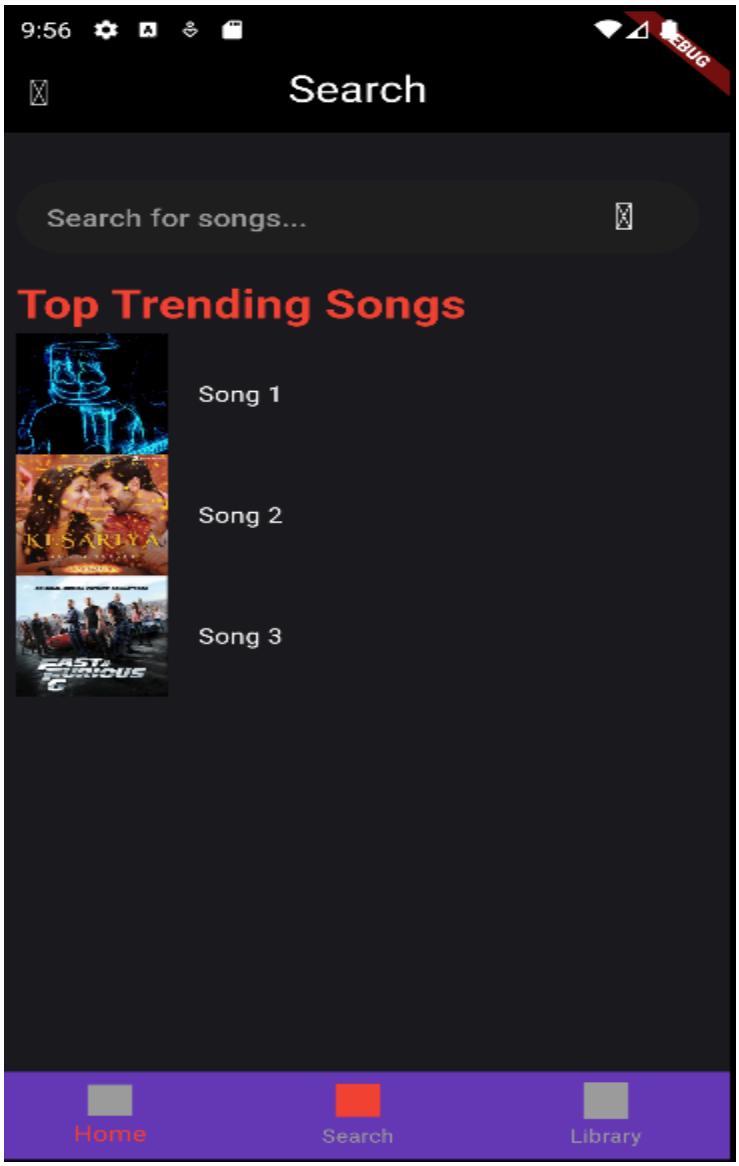
//search_screen

Project Title: **Roll No.**



//playlist_screen:

Project Title: **Roll No.**



CONCLUSION:

In conclusion, mastering navigation and routing in Flutter is essential for building sophisticated and user-centric applications. A well-designed navigation system enhances user engagement, simplifies code maintenance, and contributes to an overall positive user experience. As developers explore and apply these concepts, they gain the ability to create fluid and responsive mobile applications that meet the expectations of modern users.

Project Title: **Roll No.**

MAD & PWA Lab
Journal

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

Project Title: **Roll No.**

Ameya Chaudhary
D15a
09

EXPT 6

To connect Flutter UI with firebase database

THEORY:

Connecting a Flutter app with Firebase database and implementing email/password authentication through Firebase involves several steps

Set Up Firebase Project:

- Create a new project on the Firebase Console.
- Follow the prompts to add your app to the project. Download the config file (google-services.json for Android or GoogleService-Info.plist for iOS) and place it in the appropriate location in your Flutter project.

Add Dependencies:

- Open your pubspec.yaml file and add the necessary dependencies. For Firebase, you'll typically need:
- yaml
- Copy code

```
dependencies:
 firebase_core: ^latest_version
 firebase_auth: ^latest_version
 cloud_firestore: ^latest_version
```

Initialize Firebase:

- In your main Dart file or application entry point, initialize Firebase with the config file:
- dart
- Copy code

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

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

Project Title: **Roll No.**

Implementing Authentication

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

Connect to Firestore Database:

- Use the cloud_firestore package to interact with Firestore:
- dart
- Copy code

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

// Example to add data to Firestore
Future<void> addUserData(String uid, String name, String email) async {
 await FirebaseFirestore.instance.collection('users').doc(uid).set({
 'name': name,
 'email': email,
 });
}
```

Use Firebase Services in Your UI:

- Integrate authentication and database functions into your UI to enable user registration, login, and data retrieval.

Handle User Authentication State:

- Implement a mechanism to listen to the authentication state changes and update the UI accordingly. You can use a StreamBuilder or a state management solution like Provider or Bloc.

CODE:

//main.dart

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:first_project/firebase_options.dart';
import 'package:firebase_core/firebase_core.dart';
import 'home_screen.dart';
import 'song_details_screen.dart';
import 'library_screen.dart';
import 'login_screen.dart';
import 'splash_screen.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:just_audio/just_audio.dart'; // Import the just_audio package
import 'liked_songs.dart';
```

```
class AudioPlayerProvider extends ChangeNotifier {
 final AudioPlayer _audioPlayer = AudioPlayer(); // Create an instance of AudioPlayer

 AudioPlayer get audioPlayer => _audioPlayer;
}

void main() async {
 WidgetsFlutterBinding.ensureInitialized();
 try {
   await Firebase.initializeApp(options: DefaultFirebaseOptions.currentPlatform);
   runApp(
     MultiProvider(
       providers: [
         ChangeNotifierProvider<AudioPlayerProvider>(
           create: (_) => AudioPlayerProvider(),
         ),
         ChangeNotifierProvider(create: (_) => LikedSongs()),
       ],
       child: MyApp(),
     ),
   );
 } catch (e) {
   print('Error initializing Firebase: $e');
 }
}

class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     title: 'youtube_music',
     theme: ThemeData(
       primaryColor: Colors.black,
       scaffoldBackgroundColor: Colors.black,
       textTheme: TextTheme(
         bodyText2: TextStyle(color: Colors.white),
       ),
       appBarTheme: AppBarTheme(
         backgroundColor: Colors.black,
         foregroundColor: Colors.red,
       ),
     ),
```

```
    home: (FirebaseAuth.instance.currentUser != null) ? SplashScreen() : LoginScreen(),
   );
 }
}
```

// register_screen.dart

```
import 'package:flutter/material.dart';
import 'home_screen.dart';
import 'dart:developer';
import 'package:firebase_auth/firebase_auth.dart';

class RegisterScreen extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(
       title: Text('Register'),
       centerTitle: true,
       backgroundColor: Colors.black,
     ),
     body: Padding(
       padding: const EdgeInsets.all(16.0),
       child: SingleChildScrollView( // Wrap with SingleChildScrollView
         child: RegisterForm(),
       ),
     ),
   );
 }
}

class RegisterForm extends StatefulWidget {
 @override
 _RegisterFormState createState() => _RegisterFormState();
}

class _RegisterFormState extends State<RegisterForm> {
 final TextEditingController emailController = TextEditingController();
 final TextEditingController passwordController = TextEditingController();
 final TextEditingController cpasswordController = TextEditingController();

 void createAccount() async {
   String email = emailController.text.trim();
   String password = passwordController.text.trim(); // Corrected
   String cPassword = cpasswordController.text.trim(); // Corrected
```

```
  if (email == "" || password == "" || cPassword == "") {
    log("Please fill all the details");
  } else if (password != cPassword) {
    log("Password mismatched");
  } else {
    try {

      UserCredential userCredential = await FirebaseAuth.instance
          .createUserWithEmailAndPassword(email: email, password: password);
      log("User Created!");
      if(userCredential.user != null) {
        Navigator.pop(context);
      }
      // Redirect to HomeScreen or perform any other action
    } catch (e) {
      log("Error creating user: $e");
      // Handle the error, you can log it or show an error message
    }
  }
}

@override
Widget build(BuildContext context) {
  return Column(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      Image.asset('assets/ytlogo2.png', height: 300, width: 300),
      SizedBox(height: 20),
      TextField(
        controller: emailController,
        decoration: InputDecoration(labelText: 'Email'),
      ),
      TextField(
        controller: passwordController,
        decoration: InputDecoration(labelText: 'Password'),
      ),
      TextField(
        controller: cpasswordController,
        obscureText: true,
        decoration: InputDecoration(labelText: 'Confirm Password'),
      ),
      SizedBox(height: 20),
      ElevatedButton(
```

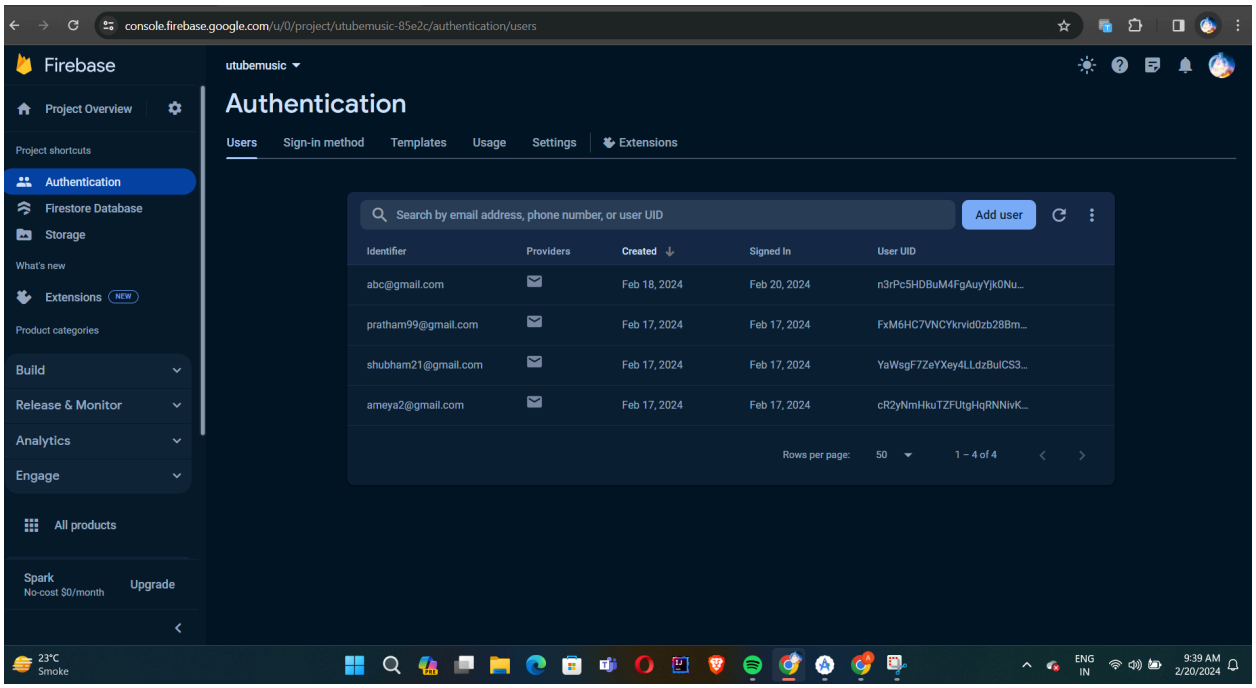
```
        onPressed: () {
          createAccount();
        },
        style: ButtonStyle(
          backgroundColor: MaterialStateProperty.all<Color>(Colors.red),
        ),
        child: Text('Register'),
      ),
    ],
  );
 }
}

//Login Search
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'home_screen.dart';
import 'register_screen.dart';

class LoginScreen extends StatelessWidget {
 TextEditingController emailController = TextEditingController();
 TextEditingController passwordController = TextEditingController();

 void login(BuildContext context) async {
  String email = emailController.text.trim();
  String password = passwordController.text.trim();

  if (email == "" || password == "") {
   print("Fill in all the fields");
  } else {
   try {
    UserCredential userCredential = await FirebaseAuth.instance
        .signInWithEmailAndPassword(email: email, password: password);
    if (userCredential.user != null) {
     Navigator.pushReplacement(
      context,
      CupertinoPageRoute(builder: (context) => HomeScreen()),
     );
    }
   } catch (e) {
    print("Error during login: $e");
   }
  }
```

```
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Login'),
      centerTitle: true,
    ),
    body: SingleChildScrollView(
      child: Container(
        color: Colors.white,
        padding: const EdgeInsets.all(16.0),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Image.asset('assets/ytlogo2.png', height: 300, width: 300),
            SizedBox(height: 20),
            TextField(
              controller: emailController,
              decoration: InputDecoration(labelText: 'Email'),
            ),
            SizedBox(height: 10),
            TextField(
              controller: passwordController,
              decoration: InputDecoration(labelText: 'Password'),
              obscureText: true,
            ),
            SizedBox(height: 20),
            ElevatedButton(
              onPressed: () {
                login(context);
              },
              style: ElevatedButton.styleFrom(
                primary: Colors.red,
              ),
              child: Text('Login'),
            ),
            SizedBox(height: 10),
            Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                Text("New?"),
                TextButton(
```

Project Title: **Roll No.**

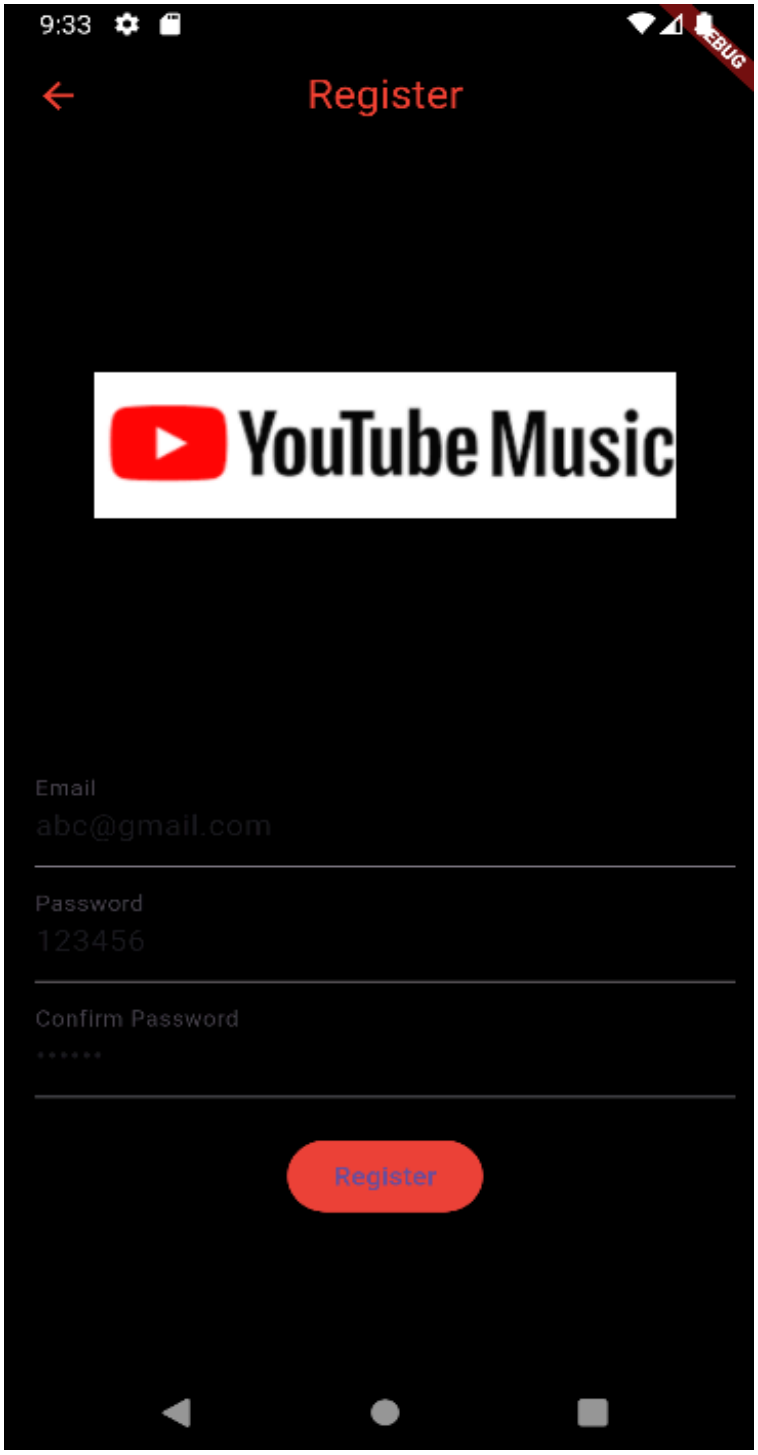
```
                onPressed: () {
                  Navigator.push(
                    context,
                    MaterialPageRoute(
                      builder: (context) => RegisterScreen(),
                    ),
                  );
                },
                child: Text('Register'),
              ),
            ],
          ),
        ],
      ),
    ),
  );
 }
}
```

FIREBASE CONSOLE:

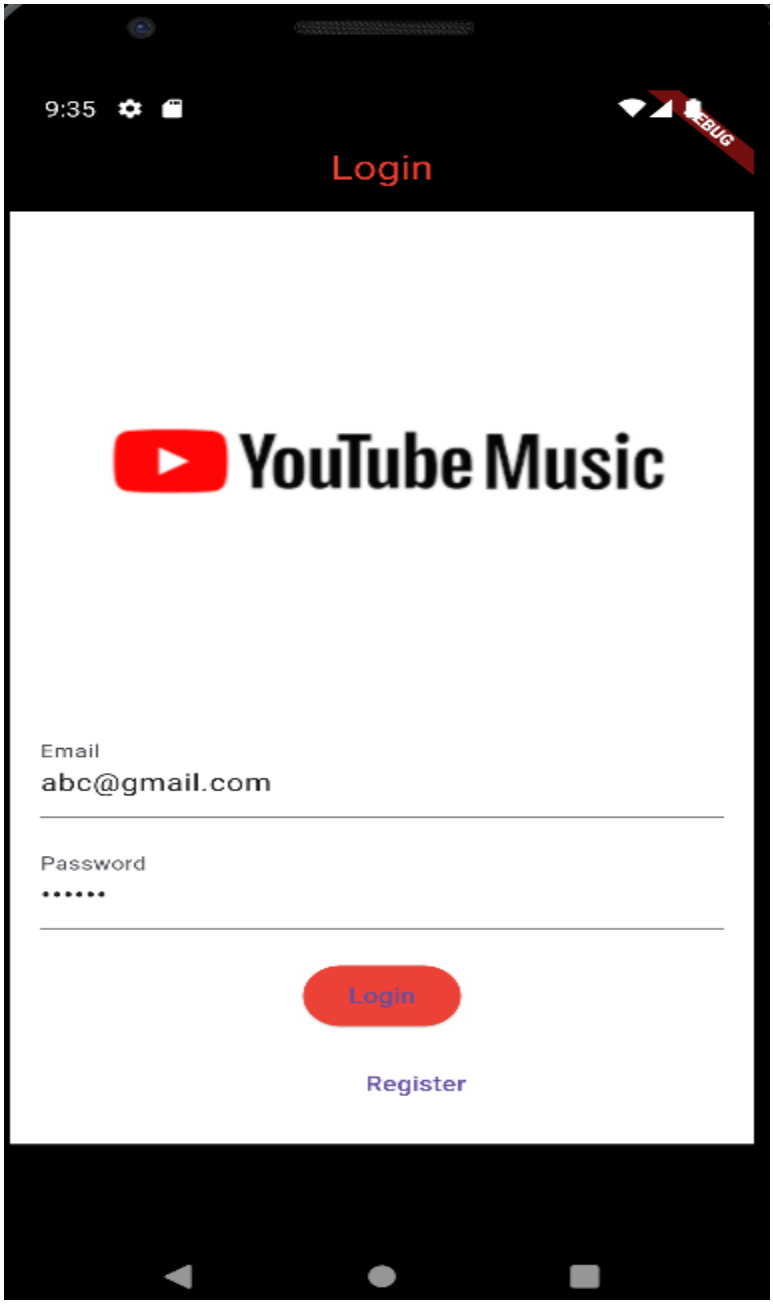


OUTPUT:

Project Title: **Roll No.**



Project Title: **Roll No.**



CONCLUSION:

In conclusion, integrating Firebase with a Flutter app for email/password authentication and database connectivity provides a robust and scalable backend solution. The combination of firebase_auth and cloud_firestore packages streamlines user authentication and real-time data management. Leveraging Firebase services enhances the overall functionality and user experience of your Flutter application, allowing seamless integration of authentication features and cloud-based storage.

Project Title: **Roll No.**

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

Project Title: **Roll No.**

Experiment No:7

Aim:

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

Theory:

Regular Web App:

- **A regular web app is a traditional website accessed through a browser.**
- **It relies on a server for most of its functionality and data processing.**
- **It does not necessarily offer offline access, relying on an active internet connection.**
- **It may not provide features like push notifications or access to device hardware.**
- **It does not have a manifest file defining its metadata or the ability to be installed on a device's homescreen.**

Progressive Web App (PWA):

- **A Progressive Web App (PWA) is a type of web application that provides a more app-like experience for users.**
- **It is built with modern web technologies like Service Workers, Web App Manifests, and**

HTTPS.

- **PWAs offer responsive design, adapting to various screen sizes and orientations.**
- **They can work offline or with a poor internet connection, using Service Workers to cache content.**
- **PWAs can send push notifications, enabling engagement even when the app is not open. ● They have a manifest file (manifest.json) that defines metadata such as the app's name, icons, colors, and start URL.**
- **Users can "install" a PWA on their device's homescreen, making it accessible like a native app, with an icon and full-screen mode.**
- **PWAs are discoverable and can be indexed by search engines, improving visibility.**
- **They provide a seamless, fast, and reliable user experience, leading to higher user engagement and retention.**

<u>Differences: Offline Functionality:</u>

- **Regular Web Apps:**
- **Typically rely on an active internet connection to function properly.**
- **When the user loses connectivity, they may see error messages or a lack of new data.**

Name: Ameya Chaudhary **PWA exp 7**
Roll no:9

● The app's functionality is limited when offline, as it cannot access previously loaded content. ●

Progressive Web Apps (PWAs):

- **Use Service Workers to cache content, allowing them to work offline or in low-connectivity situations.**
- **Users can continue to browse content, view pages, and interact with the app even without an internet connection.**
- **Updates made while offline are synchronized when the device reconnects to the internet.**

Installation:

- **Regular Web Apps:**
- **Accessed by typing the URL into a web browser.**
- **Do not have an option for installation on the device's homescreen.**

- **Progressive Web Apps (PWAs): ● Can be "installed" on the user's device, creating an icon on the homescreen.**

- **This installation is done through a browser prompt or a manual option from the browser menu.**

- **Once installed, PWAs launch in full-screen mode, resembling native mobile apps.**

Push Notifications:

- **Regular Web Apps:**

- **Lack the ability to send push notifications to users.**

- **Engagement with users outside of the app is limited.**

- **Progressive Web Apps (PWAs):**

- **Have the capability to send push notifications, even when the app is not actively open.**
  ● This feature helps in re-engaging users, announcing updates, promotions, or relevant information.

Metadata and Manifest:

- **Regular Web Apps:**

- **Do not have a manifest file (manifest.json) defining metadata.**

- **Metadata like icons, colors, and start URLs are typically not specified.**

- **Progressive Web Apps (PWAs):**
- **Use a manifest.json file to define metadata about the app.**
- **Developers can specify the app's name, icons for different device sizes, background color, theme color, and more.**
- **This metadata allows the PWA to appear more like a native app when installed on the homescreen.**

<u>Performance and Optimization:</u>

- **Regular Web Apps**
- **Performance is based on the browser and device capabilities.**
- **May not always provide a smooth, app-like experience due to varying browser support.**
- **Progressive Web Apps (PWAs):**
- **Optimized for performance and reliability.**
- **Utilize modern web technologies such as Service Workers and efficient caching strategies.**
- **Offer a faster, more responsive, and smoother user experience, especially on mobile devices. ● Can be added to the user's homescreen, launching quickly and providing a native-like feel.**

Discoverability and Indexing:

- **Regular Web Apps:**
- **Discoverability by search engines can be limited.**

- **May not appear in app stores or be easily found by users searching for similar apps.**
- **Progressive Web Apps (PWAs):**
- **Fully discoverable by search engines, improving visibility.**
- **Can be indexed and ranked in search results, similar to regular websites.**
- **Users searching for related topics may discover PWAs through search engine results.**

Implementation:

The below steps have to be followed to create a progressive web application:

Step 1: Create an HTML page that would be the starting point of the application. This HTML will contain a link to the file named manifest.json. This is an important file that will be created in the next step.

Step 2: Create a manifest.json file in the same directory. This file contains information about the web application. Some basic information includes the application name, starting URL, theme color, and icons. All the information required is specified in the JSON format. The source and size of the icons are also defined in this file.

Project Title: **Roll No.**

manifest.json about.html product.html

manifest.json > [ ] icons > {} 0

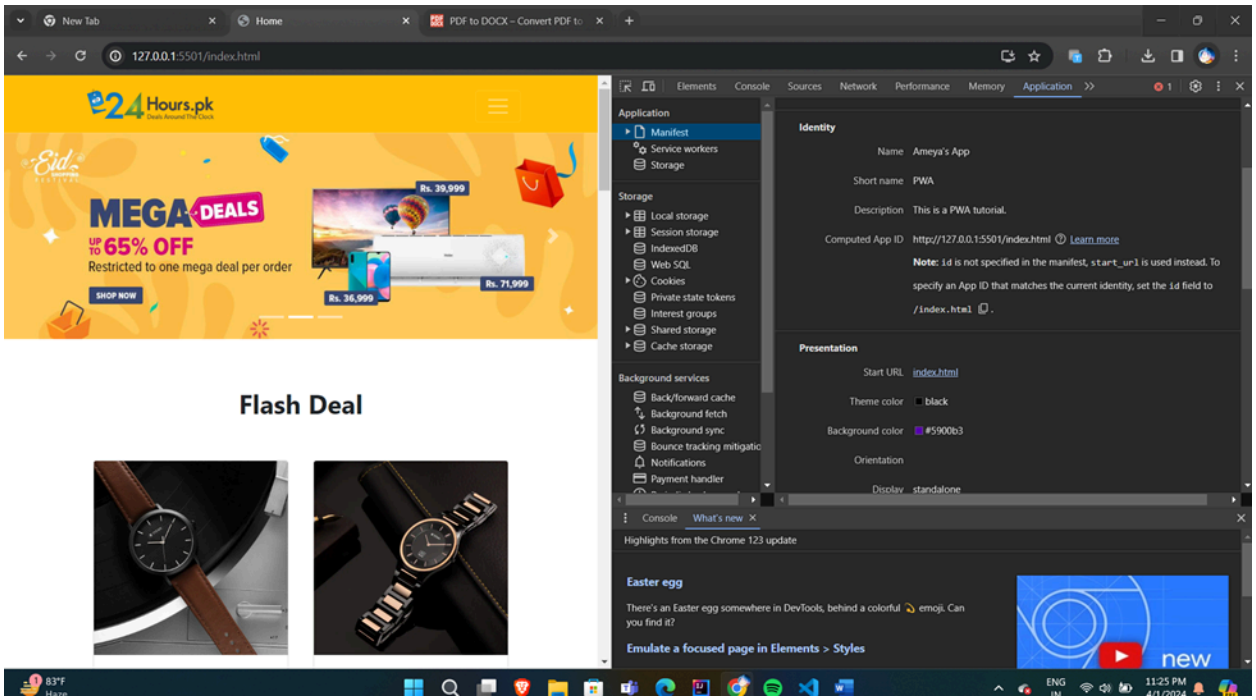
```
{
  "name":"Ameya's App",
  "short_name":"PWA",
  "start_url":"index.html",
  "display":"standalone",
  "background_color":"#5900b3",
  "theme_color":"black",
  "scope": ".",
  "description":"This is a PWA tutorial.",
  "icons":[
  {
  "src":"images/icon.png",
  "sizes":"192x192",
  "type":"image/png"
  },
  {
  "src":"images/icon.png",
  "sizes":"512x512",
  "type":"image/png"
  }
]
}
```

Project Title: **Roll No.**

Step 3: Create a new folder named images and place all the icons related to the application in that folder. It is recommended to have the dimensions of the icons at least 192 by 192 pixels and 512 by 512 pixels. The image name and dimensions should match that of the manifest file.

Step 4: Serve the directory using a live server so that all files are accessible.

Step 5: Open the index.html file in Chrome navigate to the Application Section in the Chrome Developer Tools. Open the manifest column from the list.



Step 6: Under the installability tab, it would show that no service worker is detected. We will need to create another file for the PWA, that is, serviceworker.js in the same directory. This file handles the configuration of a service worker that will manage the working of the application.

Project Title: **Roll No.**

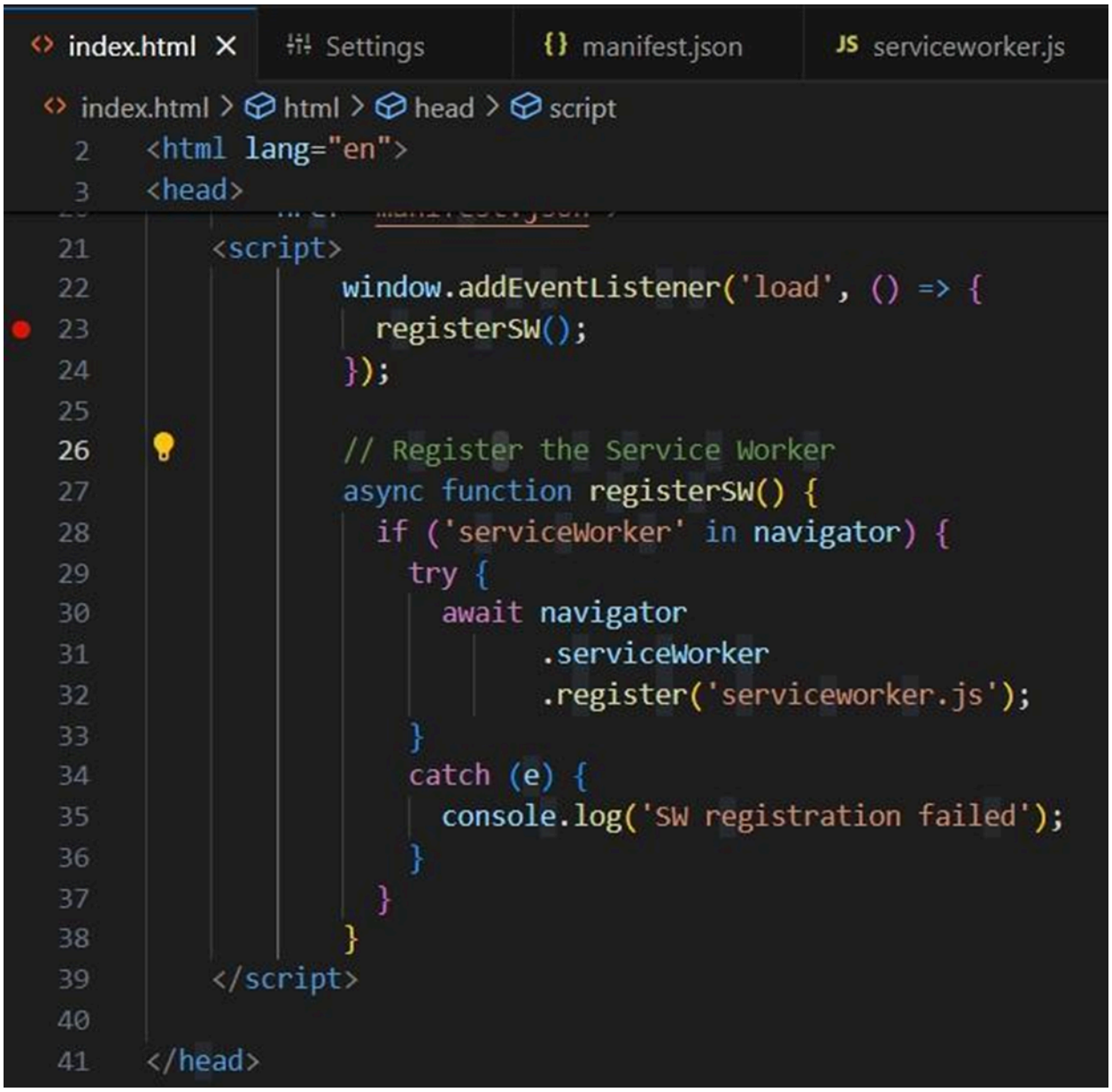
index.html | Settings | manifest.json | serviceworker.js

serviceworker.js > self.addEventListener("install") callback > then() callback

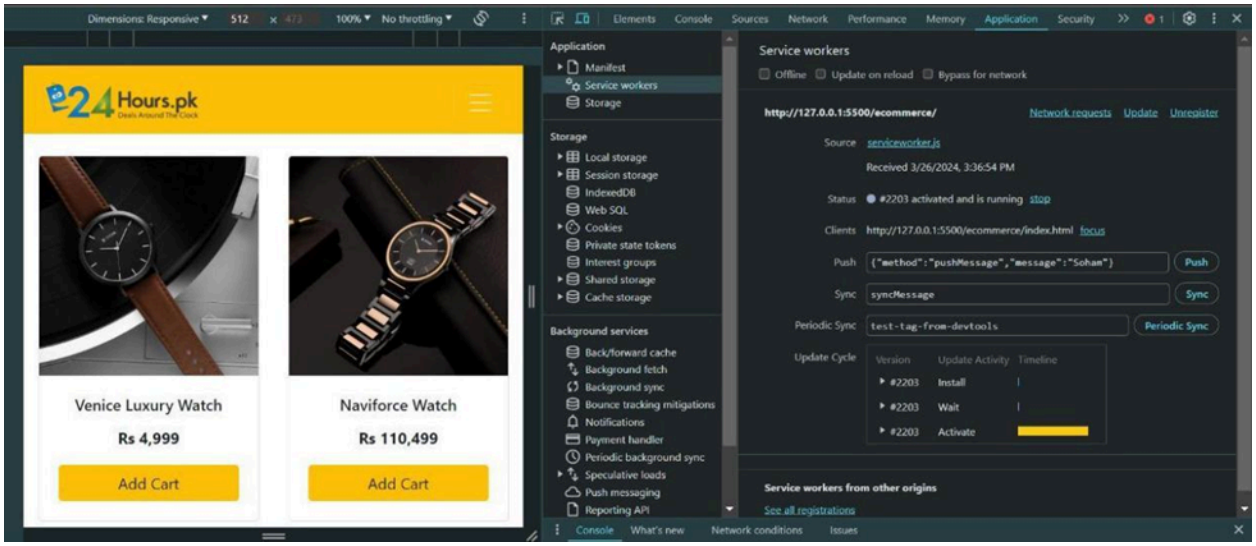
```
1   var staticCacheName = "pwa";
2
3   self.addEventListener("install", function (e) {
4     e.waitUntil(
5       caches.open(staticCacheName).then(function (cache) {
6         return cache.addAll(["/"]);
7       })
8     );
9   });
10
11  self.addEventListener("fetch", function (event) {
12    console.log(event.request.url);
13
14    event.respondWith(
15      caches.match(event.request).then(function (response) {
16        return response || fetch(event.request);
17      })
18    );
19  });
20
```

Step 7: The last step is to link the service worker file to index.html. This is done by adding a short JavaScript script to the index.html created in the above steps. Add the below code inside the script tag in index.html.

Project Title: **Roll No.**



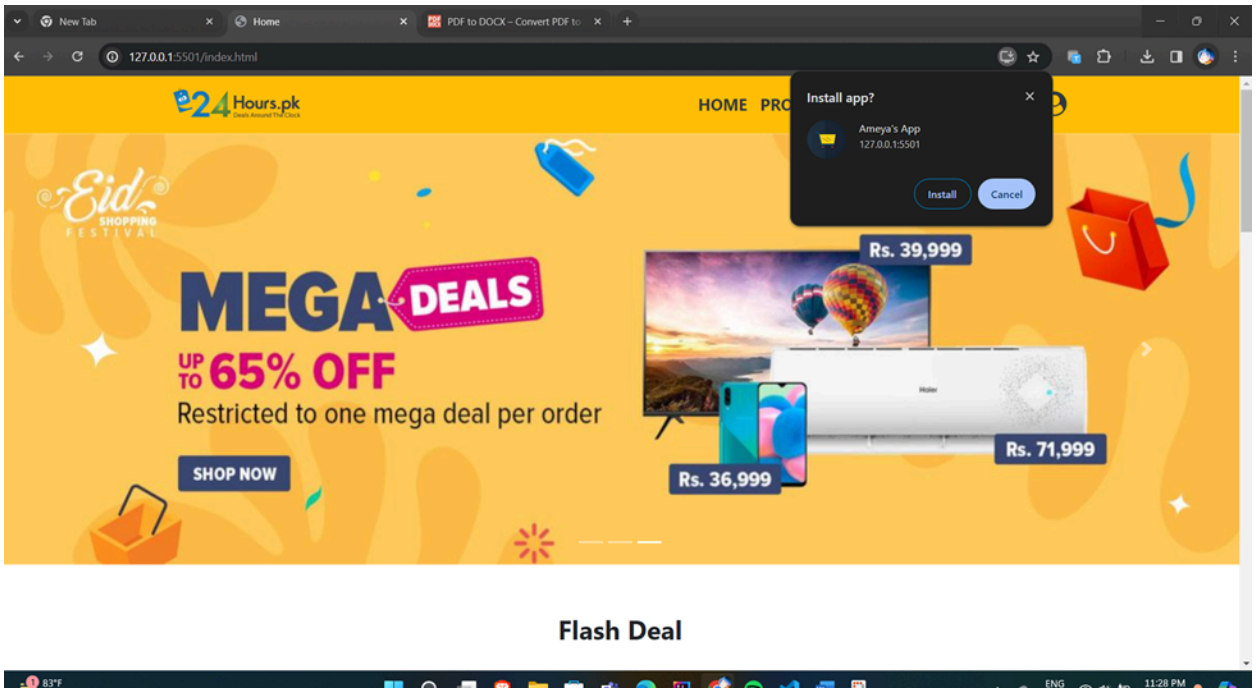
Project Title: **Roll No.**



Installing the application:

● Navigating to the Service Worker tab, we see that the service worker is registered successfully and now an install option will be displayed that will allow us to install our app. ● Click on the install button to install the application. The application would then be installed, and it would be visible on the desktop. ● For installing the application on a mobile device, the Add to Home screen option in the mobile browser can be used. This will install the application on the device.

Project Title: **Roll No.**



Conclusion :

This experiment involved creating a straightforward HTML E-commerce home page showcasing products with images, titles, prices, descriptions, and "Add to Cart" buttons. This foundational layout offers a starting point for developing a more robust E-commerce platform, with the potential for expansion into categories, navigation.

Project Title: **Roll No.**

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

Project Title: **Roll No.**

Experiment No. 8

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

Theory :

Service Worker

Service Worker is a powerful script that operates in the background of a browser, independently of user interaction. It acts as a network proxy, intercepting outgoing HTTP requests made by your web application. With Service Worker, you can manage network traffic, handle push notifications, and develop "offline first" web applications using the Cache API.

- **Network Proxy: Service workers intercept all outgoing HTTP requests, allowing you to handle these requests. For example, you can serve content from a local cache if available, improving performance and providing a better user experience.**

- **Offline Capabilities: Service workers enable offline functionality by caching essential application resources such as HTML, CSS, JavaScript, and images. When a user is offline, the service worker can retrieve the requested content from the cache, ensuring a seamless experience even without an internet connection.**

- **HTTPS Requirement: For security reasons, service workers require HTTPS connections. This ensures secure communication between the service worker, your application, and the server.**

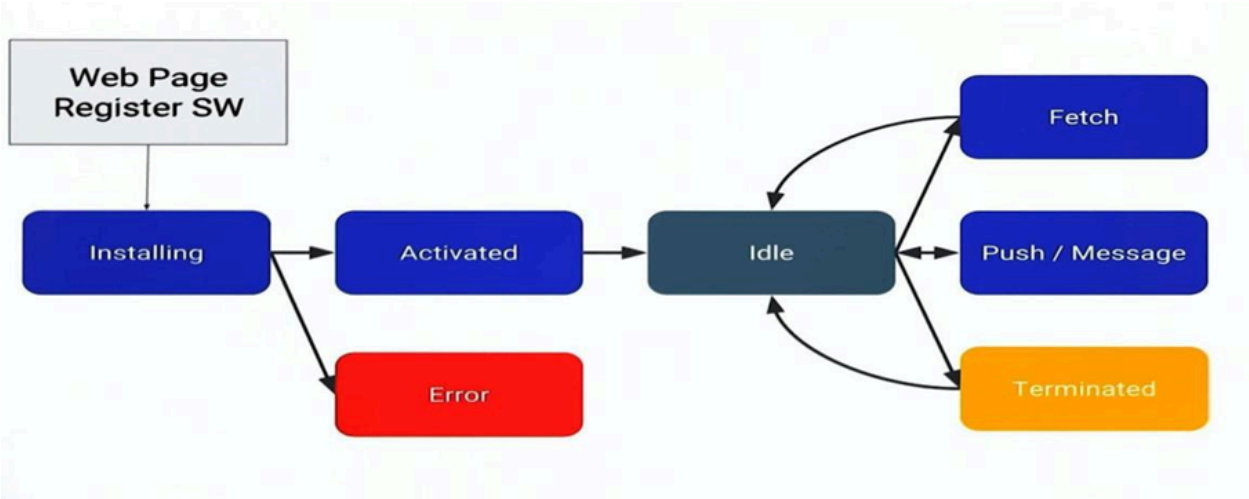
What can we do with Service Workers?

- **Network Traffic Control: Manage all network traffic of the page and manipulate requests and responses. For example, you can respond to a CSS file request with plain text or an HTML file request with a PNG file. You can also respond with the requested content.**
- **Caching: Cache any request/response pair with Service Worker and Cache API, allowing you to access offline content anytime.**

- **Push Notifications: Manage push notifications with Service Worker, enabling you to show informational messages to the user.**
- **Background Processes: Even when the internet connection is broken, you can start any process with the Background Sync feature of Service Worker.**

Service Worker Cycle

Project Title: **Roll No.**



Steps for coding and registering a service worker for your E-commerce PWA completing the install and activation process:

1. **Create the Service Worker File (sw.js):**

```
self.addEventListener('install', event => {
event.waitUntil(
    caches.open('offline').then(cache => {
        return cache.addAll([
            'index.html',
            'contact.html',
            'about.html',
            'images/1.png',
            'images/2.avif',
            'product.html'
        ]);
    })
);
});
```

Project Title: **Roll No.**

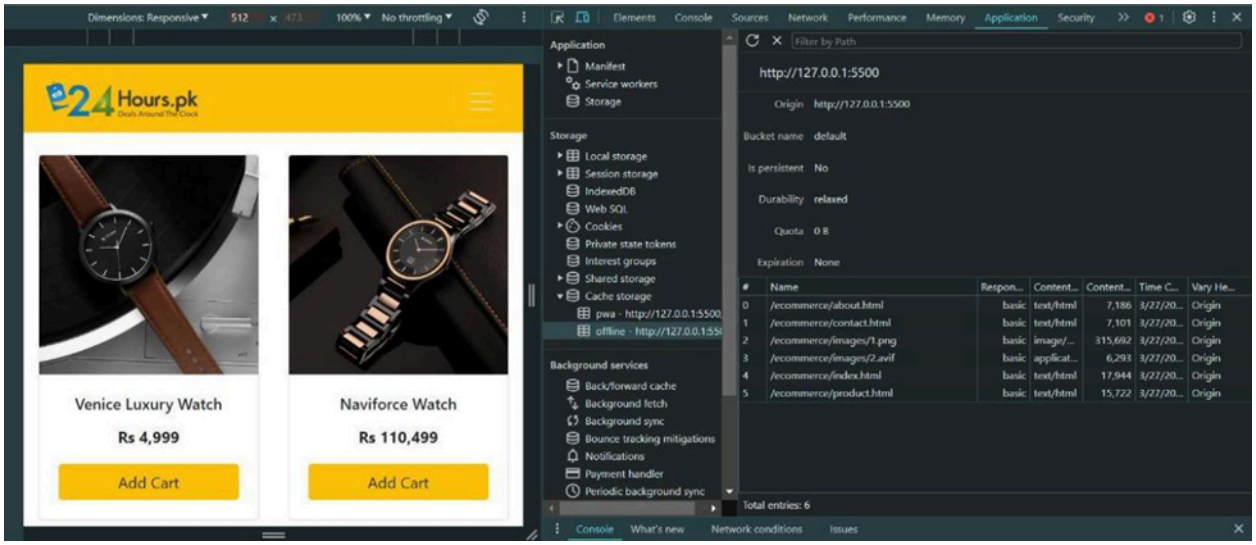
2. Register the Service Worker:

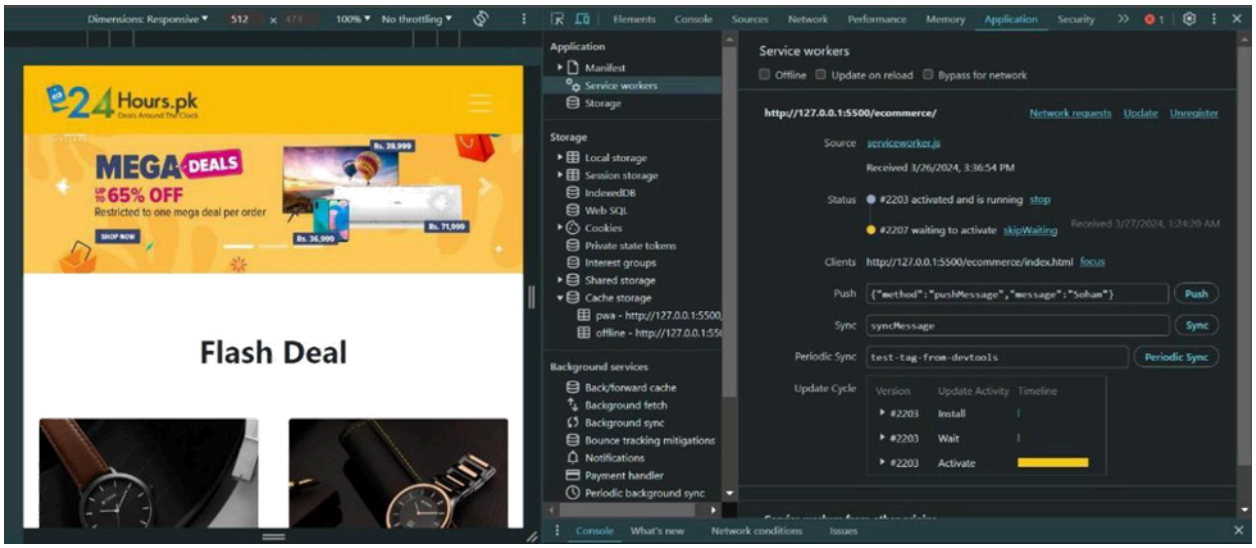
In your main JavaScript file (e.g., `main.js` or `app.js`), add the following code:

```
async function registerSW() {
  if ('serviceWorker' in navigator) {
    try {
      await navigator
            .serviceWorker
            .register('serviceworker.js');
    }
    catch (e) {
      console.log('SW registration failed');
    }
  }
}
```

Output

Project Title: **Roll No.**





Conclusion : I have understood and successfully registered a service worker, and completed the install and activation process for a new service worker for the E-commerce PWA.

Project Title: **Roll No.**

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

Project Title: **Roll No.**

EXPERIMENT NO. 9

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

Theory:

Service Worker

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

Things to note about Service Worker:

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

you need to persist and reuse across restarts, you can use IndexedDB databases.

- **Service workers make extensive use of promises, so if you're new to promises, then you should stop reading this and check out Promises, an introduction.**

Fetch Event

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

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

- **CacheFirst - In this function, if the received request has cached before, the cached response is returned to the page. But if not, a new response requested from the network.**

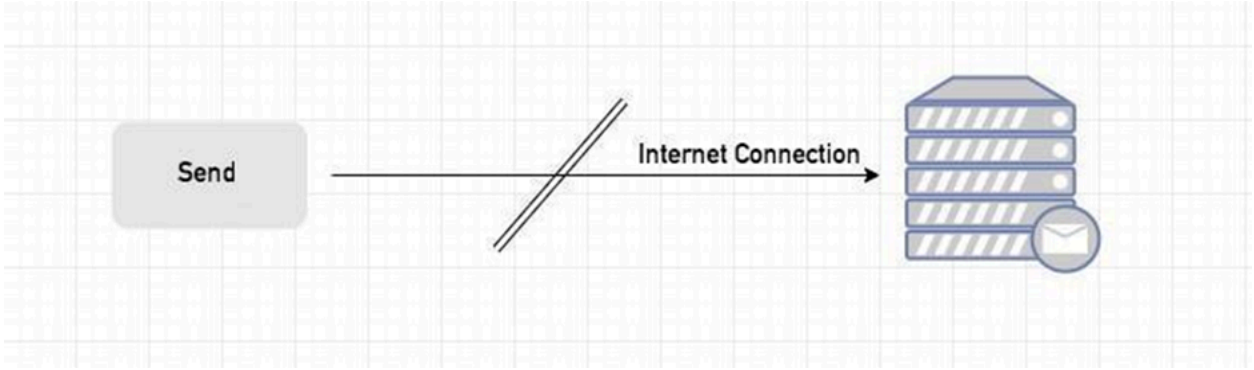
- **NetworkFirst - In this function, firstly we can try getting an updated response from the network, if this process completed successfully, the new response will be cached**

and returned. But if this process fails, we check whether the request has been cached before or not. If a cache exists, it is returned to the page, but if not, this is up to you. You can return dummy content or information messages to the page.

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

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

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

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

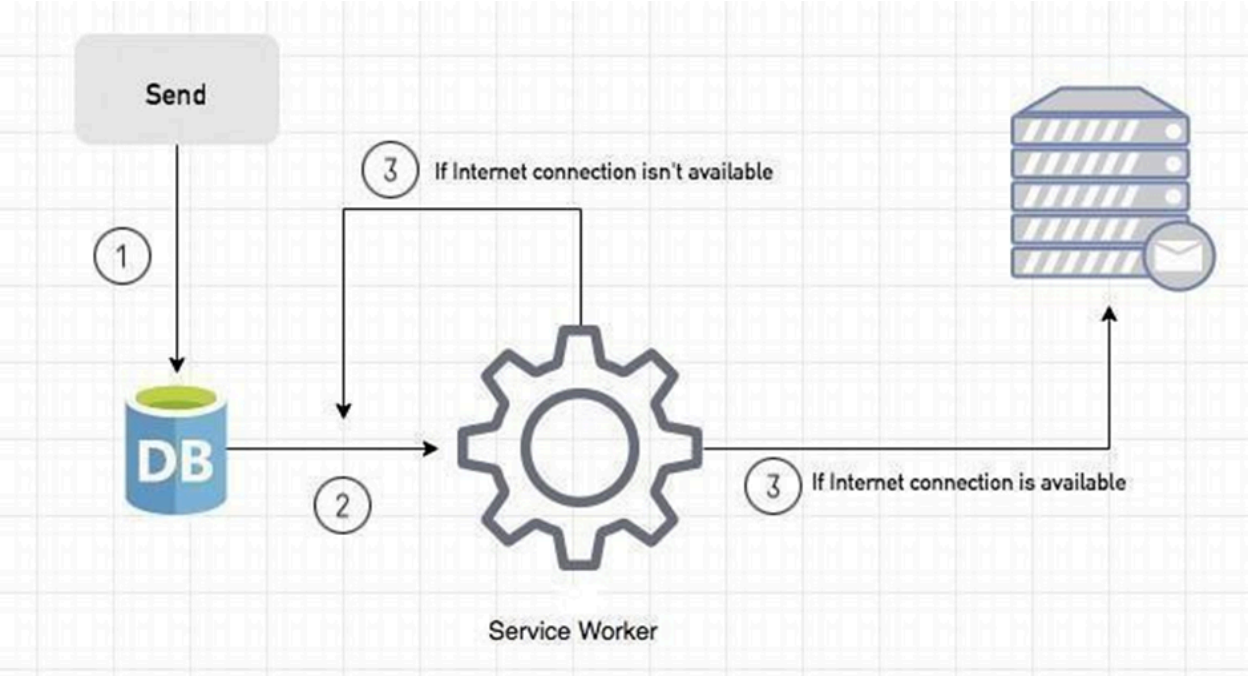
Sync Event

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



Here is a job for the Background Sync.

Project Title: **Roll No.**



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.

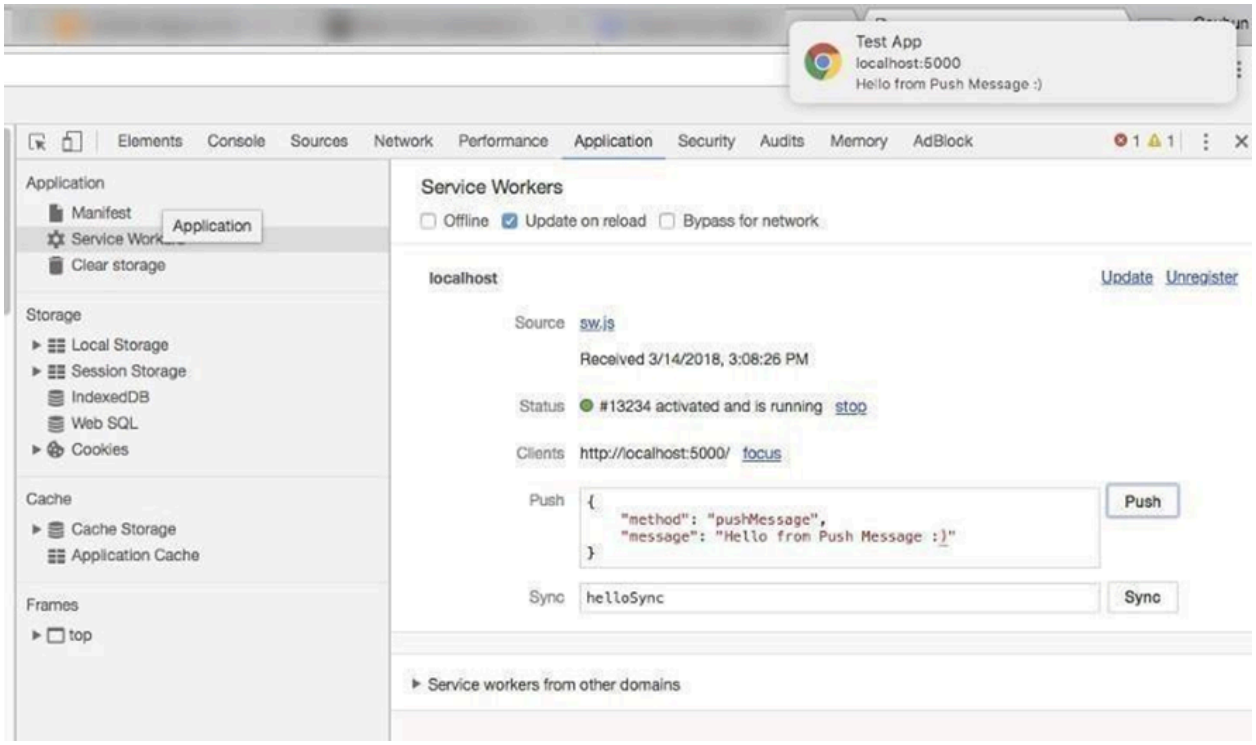
“Notification.requestPermission();” is the necessary line to show notification to the user. If you don’t want to show any notification, you don’t need this line.

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.

Project Title: **Roll No.**



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') {
```

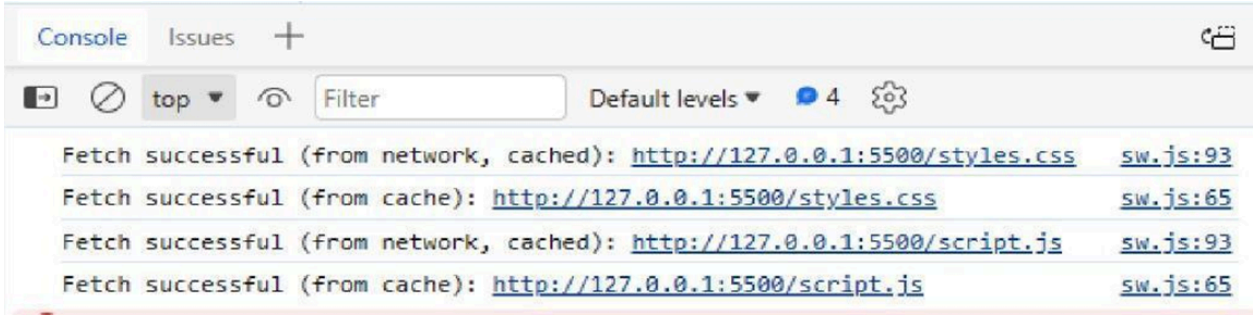
```
      console.log("Sync successful!")
  }
});

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.html',
];

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();
```

Project Title: **Roll No.**

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

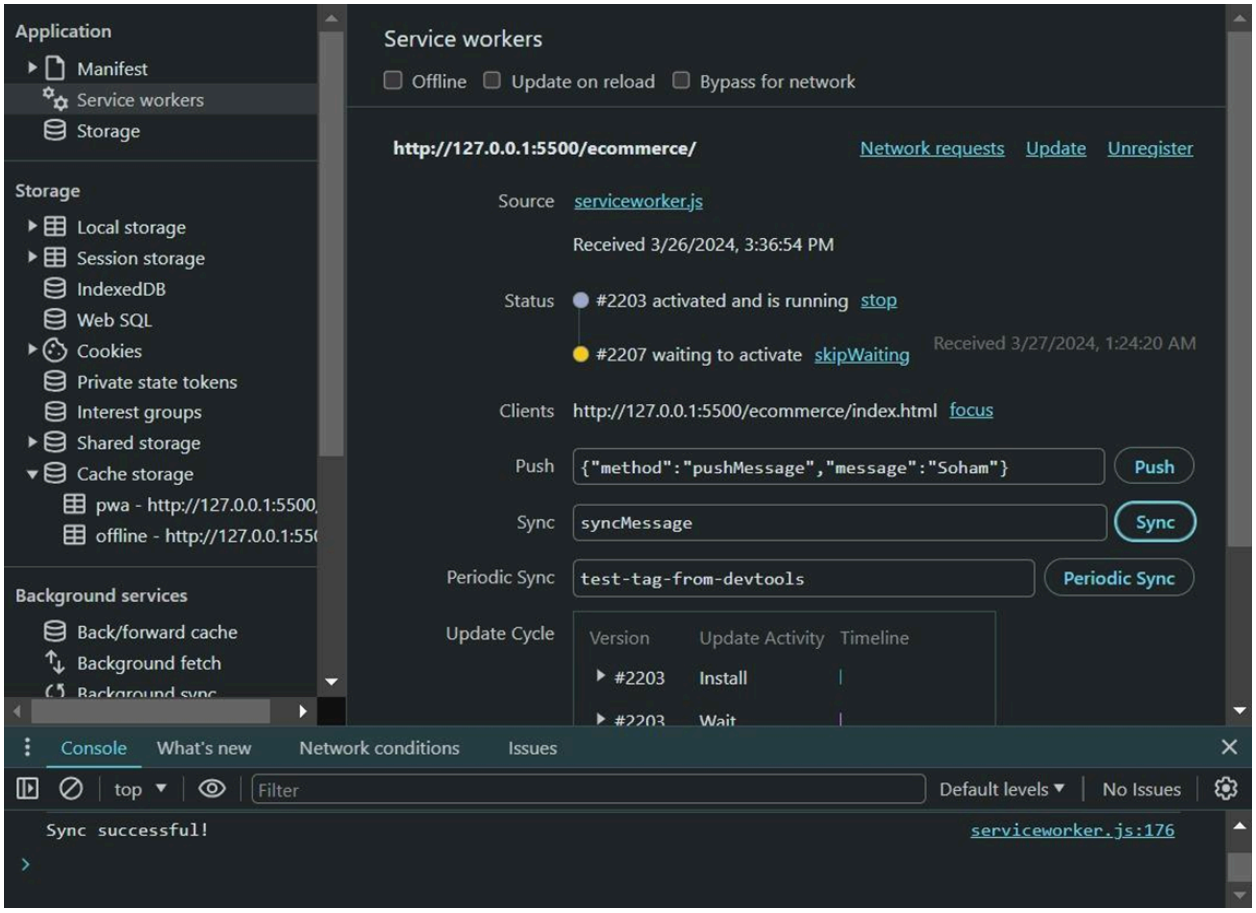
Output:

FetchEvent

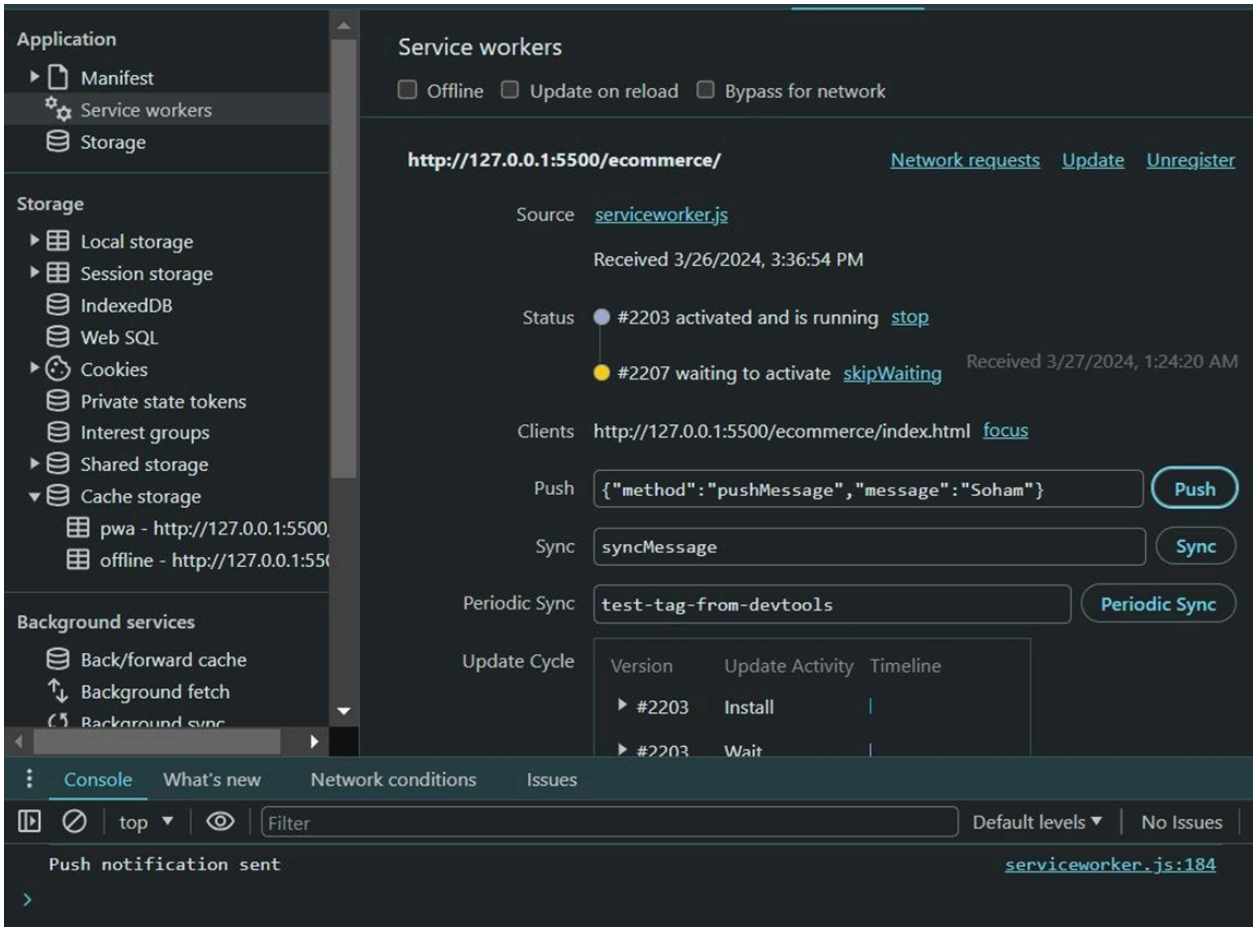


Sync event

Project Title: **Roll No.**



Project Title: **Roll No.**



Conclusion : We have understood and successfully implemented events like fetch , push and sync for our ecommerce pwa.

Project Title: **Roll No.**

MAD & PWA Lab
Journal

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

Project Title: **Roll No.**

Name: Ameya Chaudhary

D15A

09

Experiment No. 10

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

Theory:

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

GitHub Pages provides the following key features:

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

Project Title: **Roll No.**

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 Arecord 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-thebox, 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 realtime.

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. **Areal-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.**

Project Title: **Roll No.**

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.

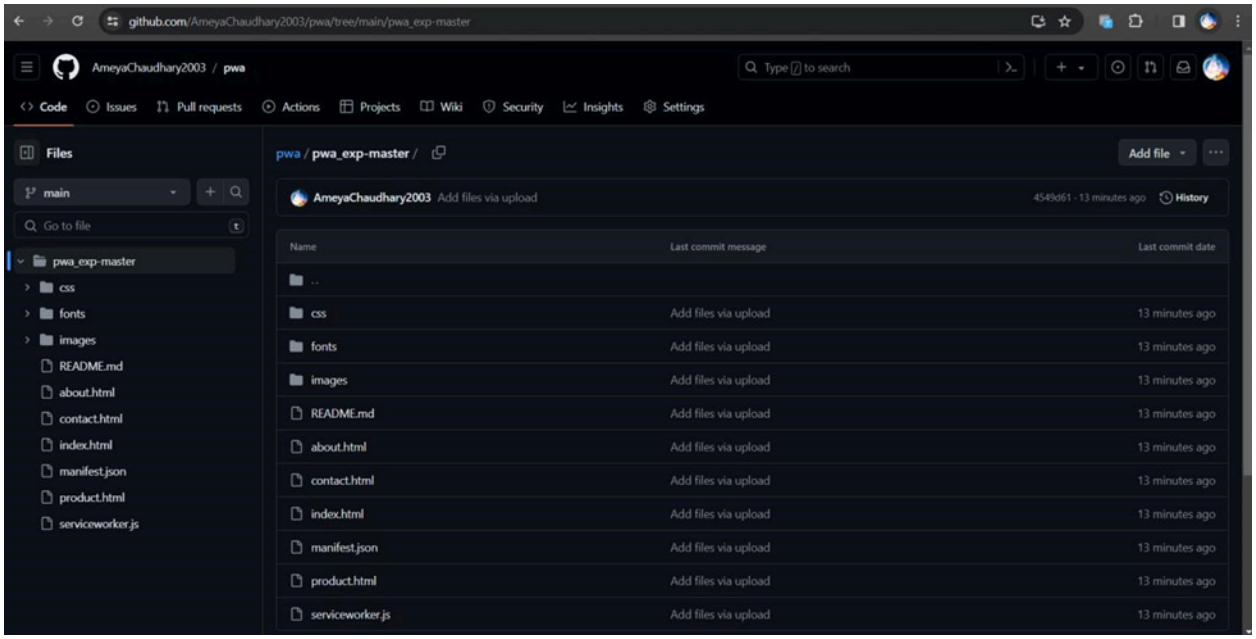
2. Command-line interface only.

3. Noin-built support for any static site generator.

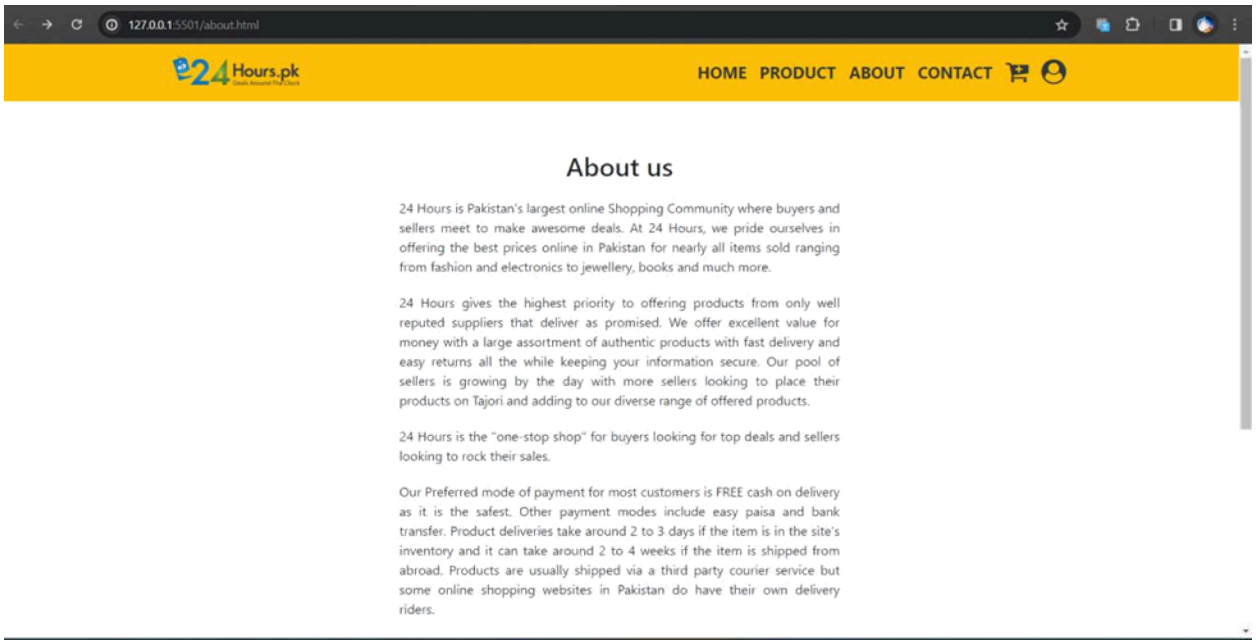
Link to github repo:

https://github.com/AmeyaChaudhary2003/pwa/tree/main/pwa_exp-master

Github Screenshot:



Project Title: **Roll No.**



Conclusion: We have deployed our Ecommerce Pwa via GitHub pages and understood the working of Github pages.

Project Title: **Roll No.**

MAD & PWA Lab
Journal

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

Project Title: **Roll No.**

EXPERIMENT NO.11

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

THEORY :

Google Lighthouse is an open-source tool developed by Google that helps developers improve the quality and performance of web pages and web applications. It is commonly used to audit and analyze various aspects of a website, including performance, accessibility, best practices, SEO (Search Engine Optimization), and Progressive Web App (PWA) functionality. Here's a breakdown of what Google Lighthouse is and its key features:

1. Audit Capabilities:

-Google Lighthouse can perform audits on different aspects of a web page or web application.

-It evaluates performance metrics such as load time, page speed, and resource optimization.
-It checks accessibility standards to ensure that websites are usable by people with disabilities.

-It assesses best practices to identify areas where coding standards can be improved. -It analyzes SEO factors to help improve search engine rankings.

-It examines PWA features to verify if a web app meets the criteria for being considered a Progressive Web App.

2. Scoring System:

-Lighthouse provides a scoring system for each audit category, ranging from 0 to 100. - Higher scores indicate better performance, accessibility, best practices, SEO, and PWA compliance.
-The scores are accompanied by detailed information and recommendations for improving each aspect.

3. Detailed Reports:

-After running an audit, Lighthouse generates a detailed report that includes scores, metrics, and recommendations.
-The report highlights areas of concern and provides actionable insights to optimize web pages and web apps.

4. Integration with DevTools:

- Lighthouse is integrated into Chrome DevTools, making it easily accessible for developers.

- It can be launched directly from DevTools to audit a specific web page or web

app.

5. Open-Source and Extensible:

- Lighthouse is an open-source tool, allowing developers to contribute to its development and customize its functionality.

- It supports plugins and extensions, enabling additional capabilities and integrations.

Project Title: **Roll No.**

6. Focus on Performance Optimization:

- One of Lighthouse's primary focuses is on performance optimization, helping developers identify and address issues that impact page load times and user experience.

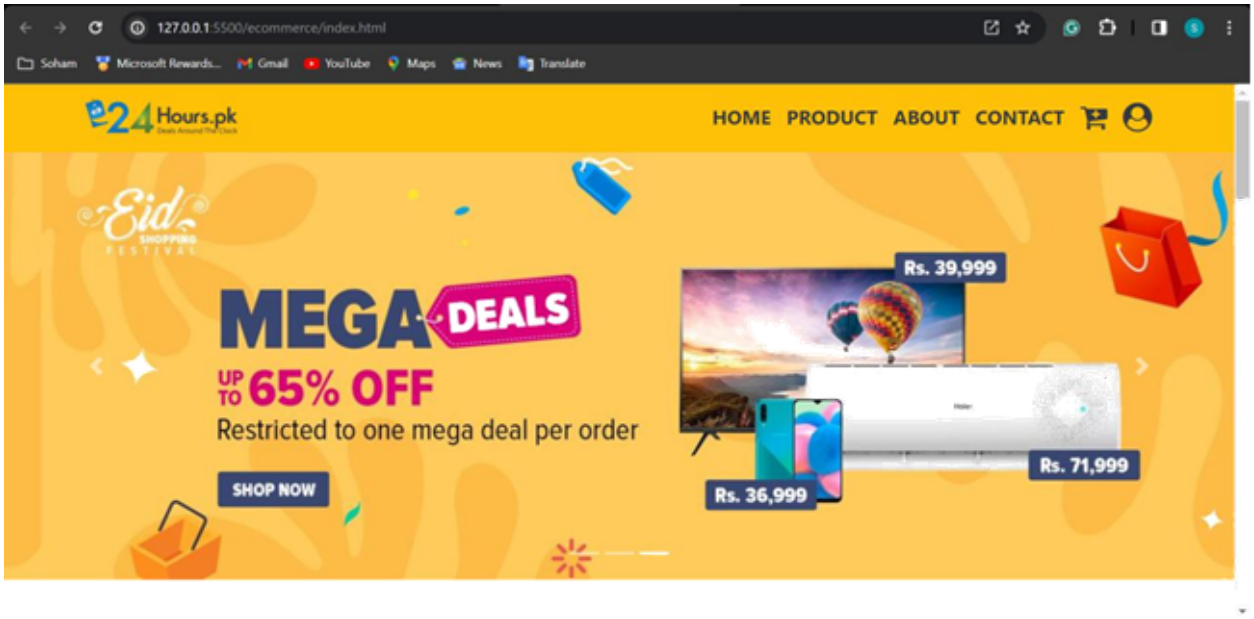
Steps :

To use Google Lighthouse to test the Progressive Web App (PWA) functioning, follow these steps:

1. Open Chrome DevTools:

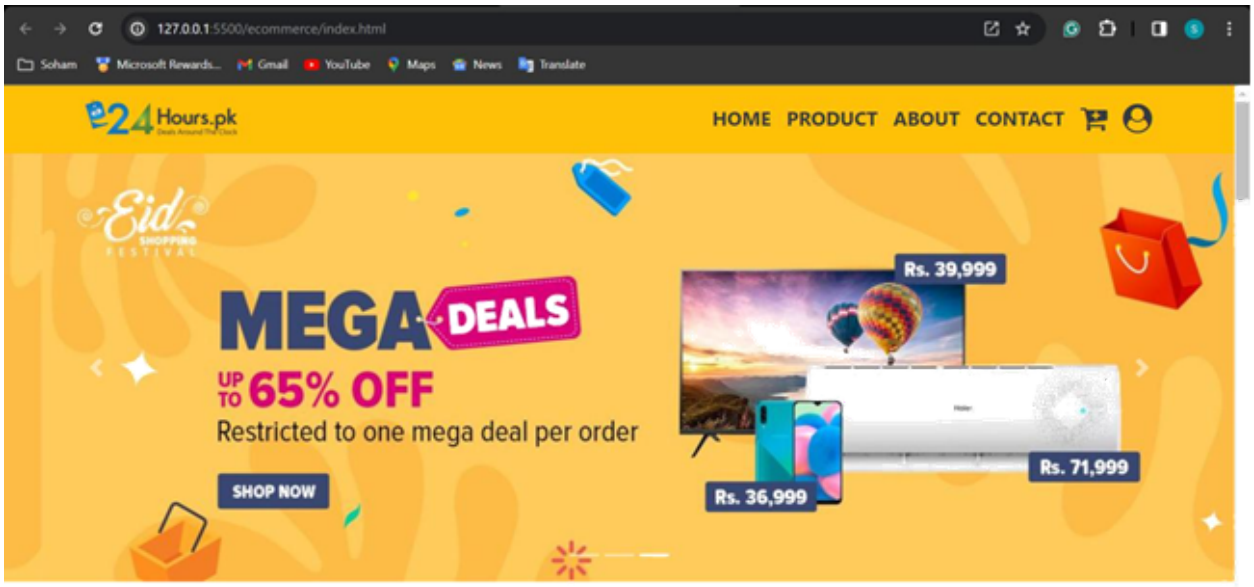
- **Open Google Chrome browser.**
- **Go to the website you want to test as a PWA.**
- **Right-click on the page and select "Inspect" or press `Ctrl+Shift+I` (Windows/Linux) or `Cmd+Option+I` (Mac) to open Chrome DevTools.**

Project Title: **Roll No.**



2. Navigate to the Lighthouse Tab:

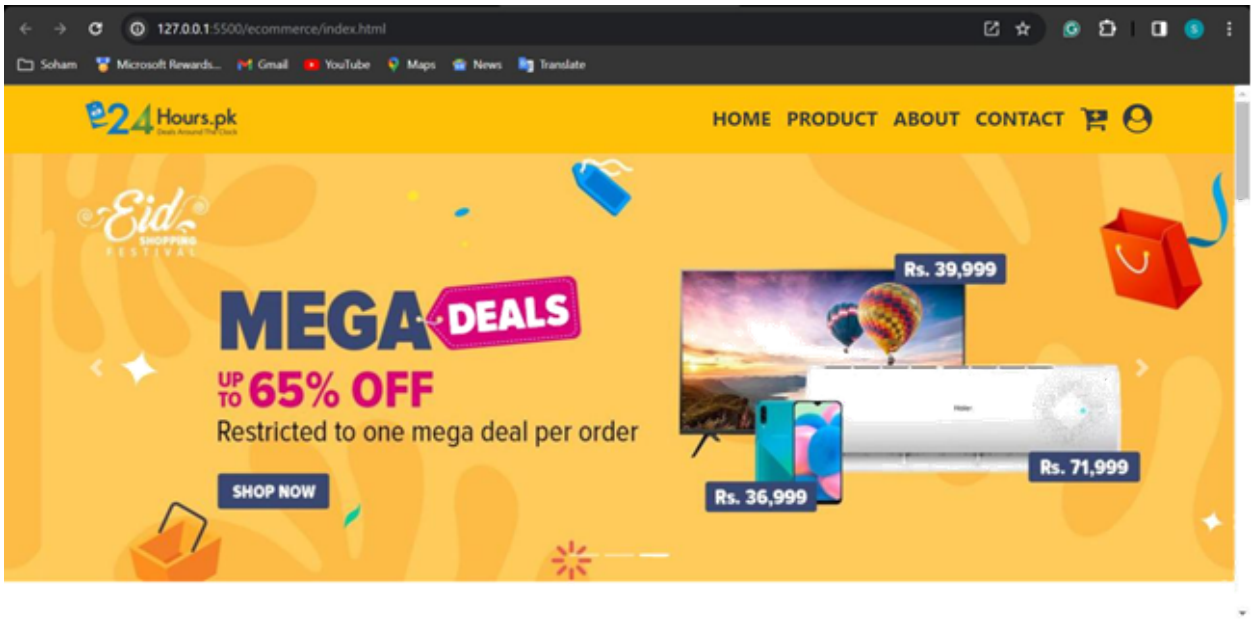
- In Chrome DevTools, click on the "Lighthouse" tab at the top of the DevTools panel.



Project Title: **Roll No.**

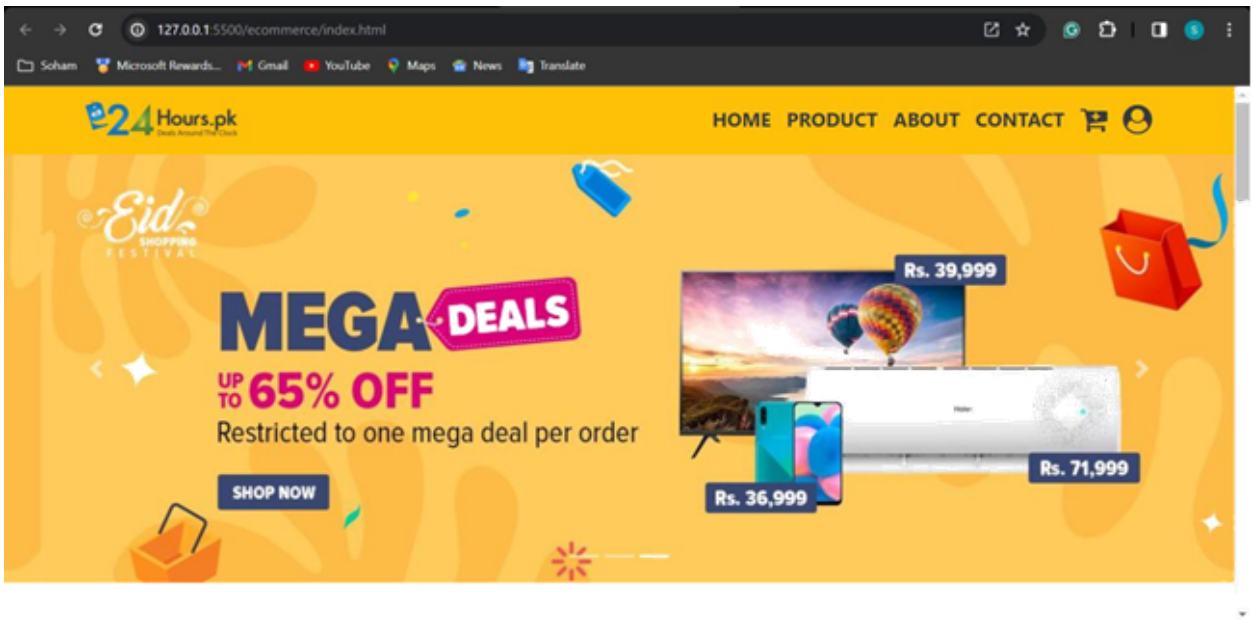
3. Run the Lighthouse Audit:

- Click on the "Generate report" button to start the audit process.



You can choose to audit for Performance, Accessibility, Best Practices, SEO, and Progressive Web App (PWA) functionality. Make sure to select the "Progressive Web App" checkbox.

Project Title: **Roll No.**



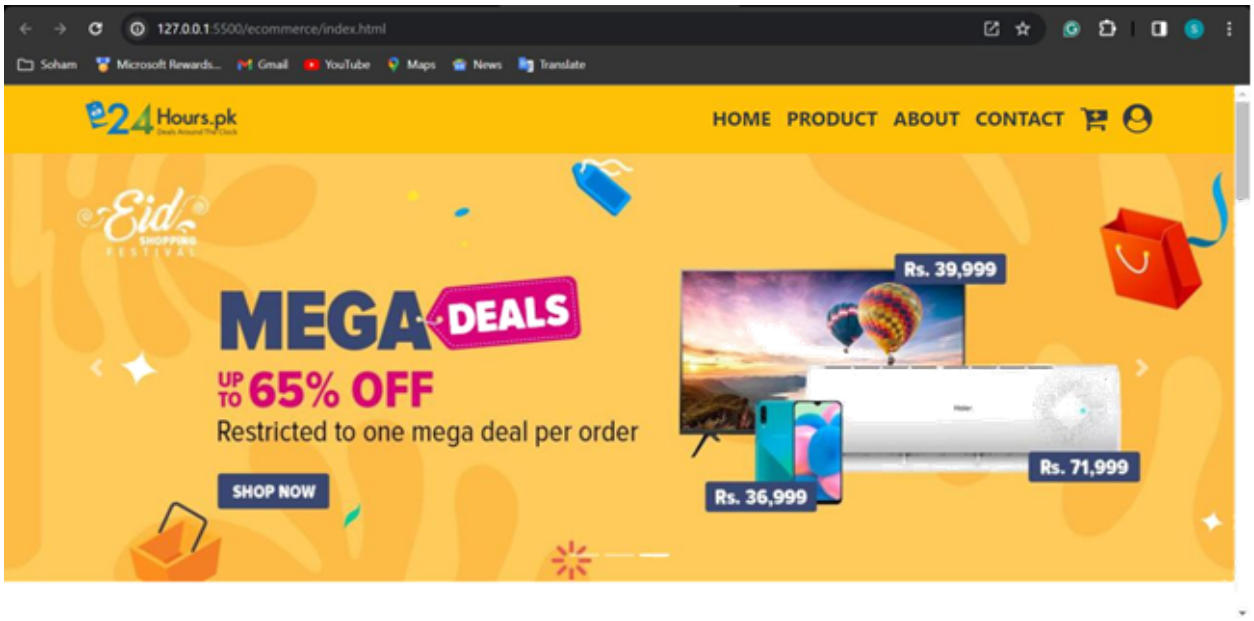
4. View the Audit Results:

-After the audit is complete, Lighthouse will display a report with scores and detailed information for each category.

-In the PWA section, you can check if your website meets the PWA criteria, such as having a service worker, being responsive on different devices, having a valid manifest file, etc.

-Lighthouse will provide suggestions for improvements and optimizations to enhance your PWA functionality.

Project Title: **Roll No.**



CONCLUSION : Hence we have understood the working of Google Lighthouse PWA analysis tool, and used Google Lighthouse tool to test and analyze the performance statistics of our E-commerce Progressive web application.

Project Title: **Roll No.**

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.	
Name	
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	

Project Title: **Roll No.**

DSA
09

Assignment -1 Flutter

Q1) Flutter overview - explain the key features and advantages of using flutter for mobile app development. Discuss how the flutter framework differs from traditional approaches.

→

1. key features of flutter
a) Single codebase for multiple platforms
Flutter allows developers to write code and deploy it.
b) Hot reload -
Enables users to instantly see the results of code changed.
c) Expressive UI -
developers have the flexibility to create expressive UI's
d) Integration with other tools
Flutter can easily integrate with other popular development tools and frameworks

2) Advantages of flutter :-
a) faster development -
b) consistant UI across platforms.
c) cost - efficiency.
d) differ from traditional approach
e) Hot reloads allows to see changes made instantly.

Project Title: **Roll No.**

Q.2) Widget Tree and composition: Describe the concept of widget tree in flutter. Explain how widget composition is used to build complex user interfaces.

→ 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 layouts, is represented by a widget

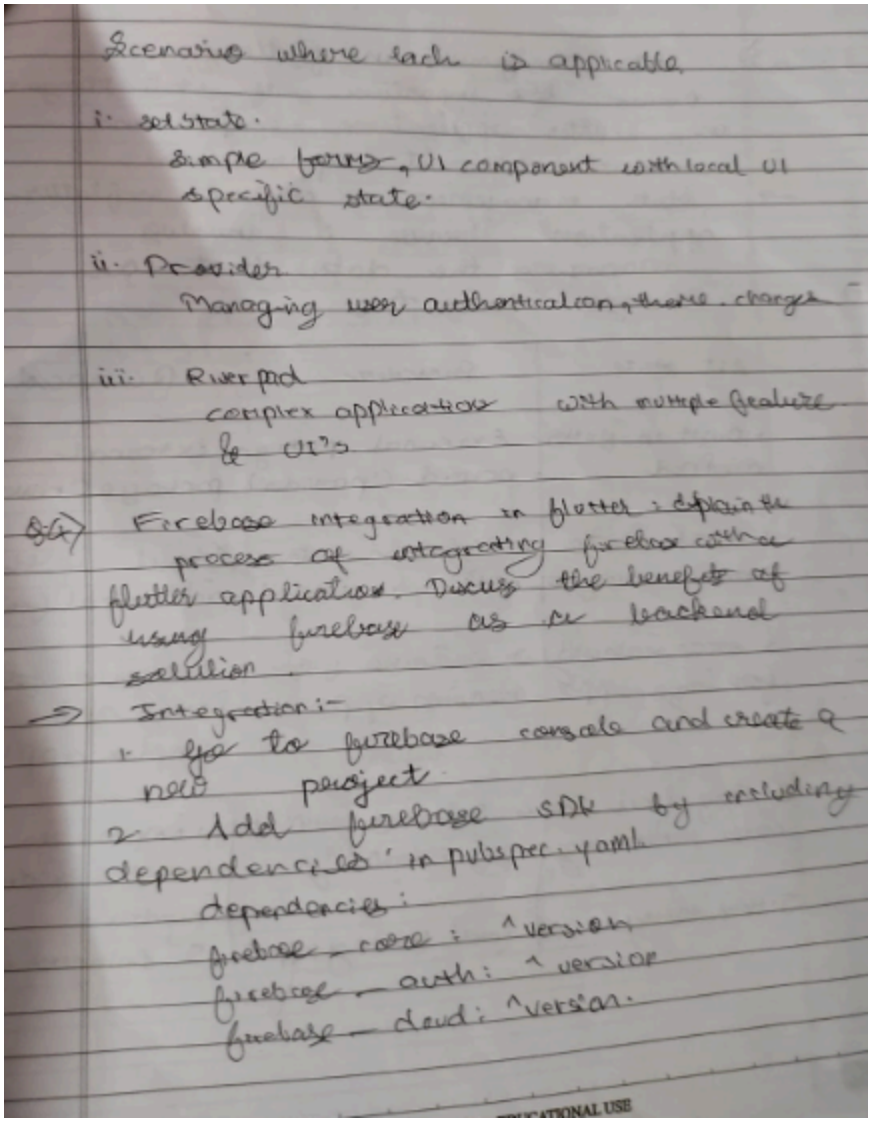
a) stateless widget
It is immutable and cannot change over time.
eg: images, text.

b) stateful widget
widget that can change its state over time.
eg - buttons, forms.

Widget composition

- Widget composition in flutter involves combining multiple simple widgets to create more complex and compound interest.
- This composability is a powerful concept that allows developers to build UI's

Project Title: **Roll No.**



Scenarios where each is applicable

i. setState.
Simple forms, UI component with local UI specific state.

ii. Provider
Managing user authentication, theme, charges

iii. Riverpod
complex applications with multiple features & UI's

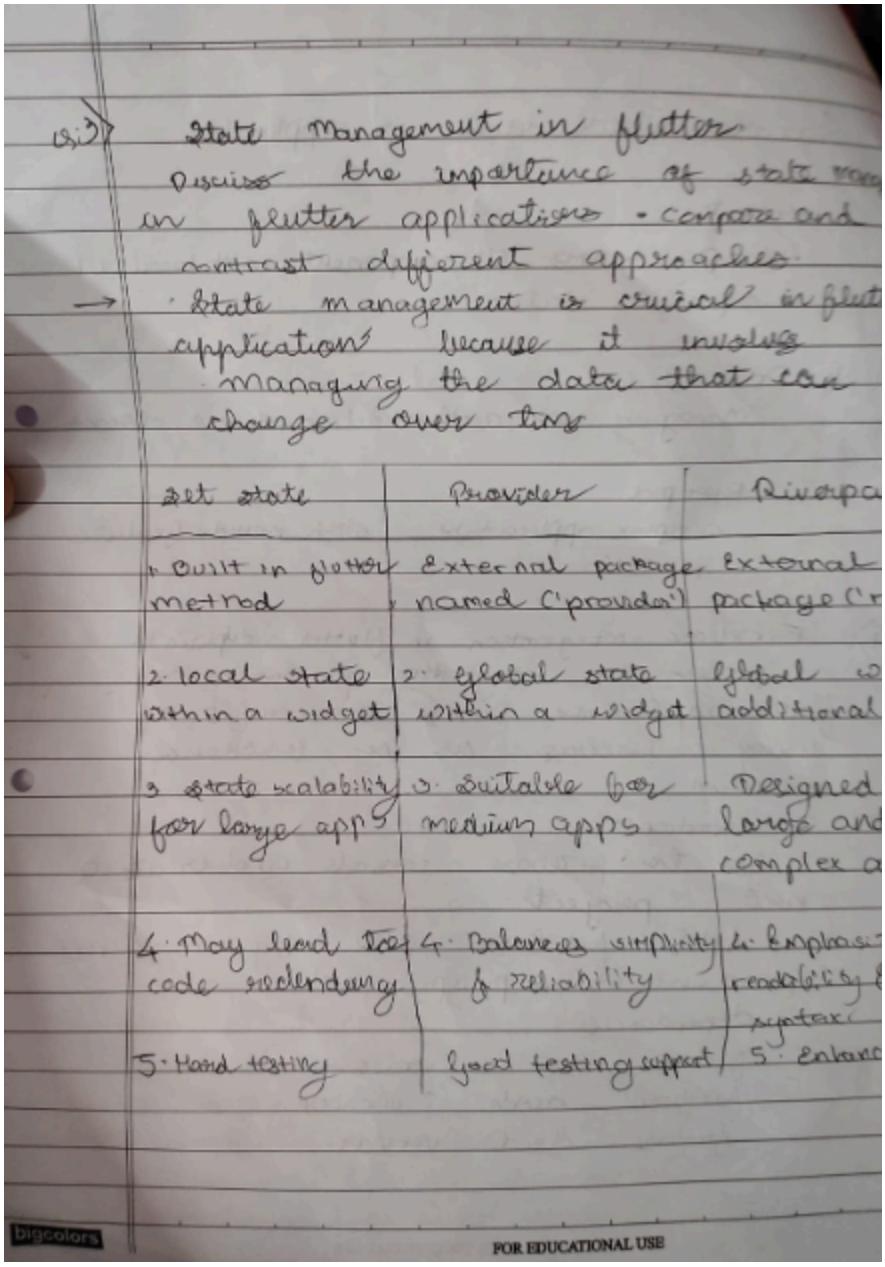
Q4) Firebase integration in flutter : explain the process of integrating firebase with a flutter application. Discuss the benefits of using firebase as a backend solution.

⇒ 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
firebase_cloud: ^version.

Project Title: **Roll No.**



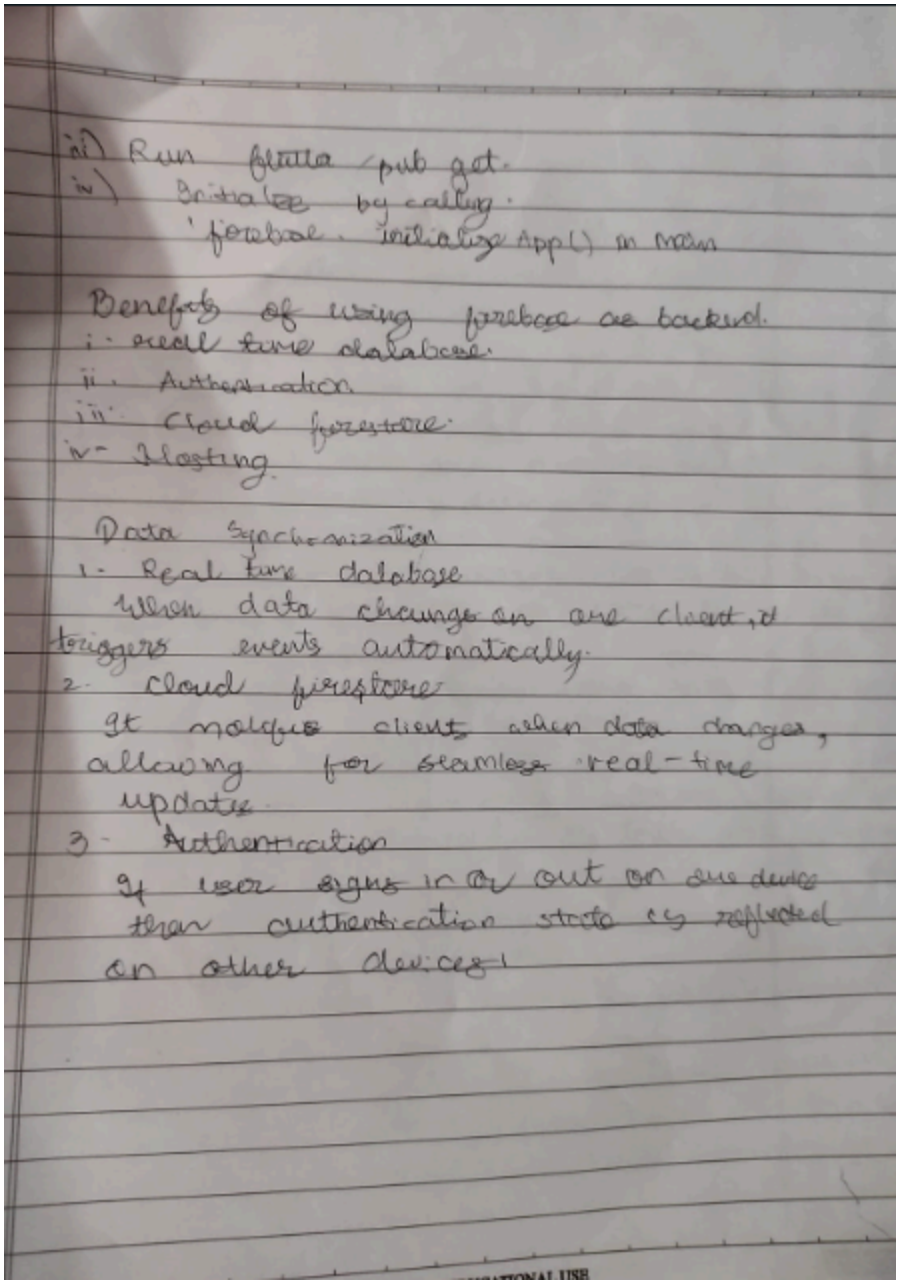
Q.3) State Management in flutter

Discuss the importance of state ma
in flutter applications. Compare and
contrast different approaches.

→ State management is crucial in flut
application because it involves
managing the data that can
change over time

set state	Provider	Riverpa
1. Built in flutter method	External package named ('provider')	External package ('r
2. local state within a widget	2. global state within a widget	Global s additional
3. state scalability for large apps	3. Suitable for medium apps	Designed large and complex a
4. May lead to code redundancy	4. Balances simplicity & reliability	4. Emphasi readability syntax
5. Hard testing	Good testing support	5. Enhanc

Project Title: **Roll No.**



iii) Run flutter pub get.
iv) Initialize by calling
'firebase_initializeApp() in main

Benefits of using firebase as backend.
i: real time database.
ii. Authentication
iii. Cloud firestore
iv- Hosting.

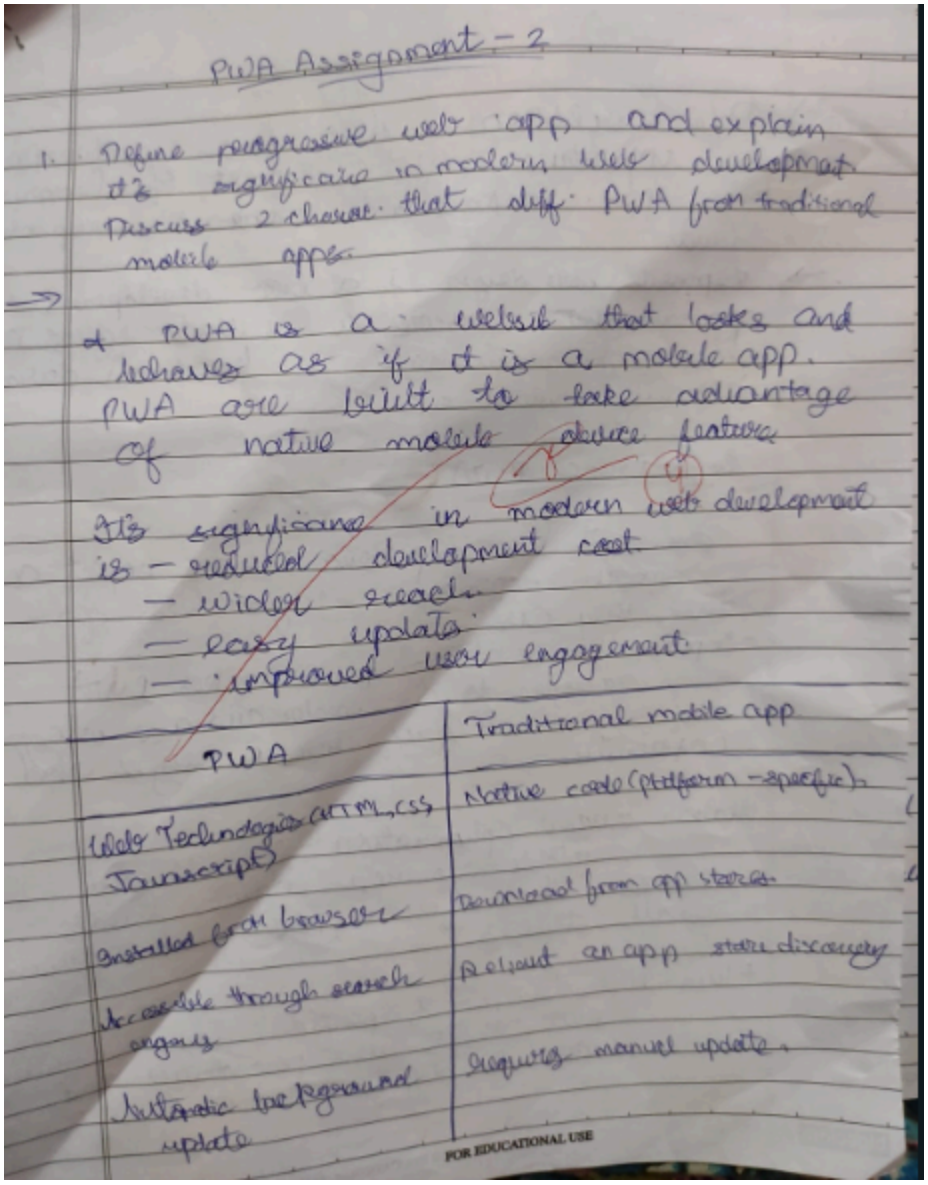
① Data synchronization
1. Real time database
When data changes on one client, it triggers events automatically.
2. Cloud firestore
It notifies clients when data changes, allowing for seamless real-time updates.
3. Authentication
If user signs in or out on one device then authentication state is reflected on other devices.

Project Title: **Roll No.**

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

Project Title: **Roll No.**



PWA Assignment - 2

1. Define progressive web app and explain it's significance in modern web development. Discuss 2 choose that diff. PWA from traditional mobile apps.

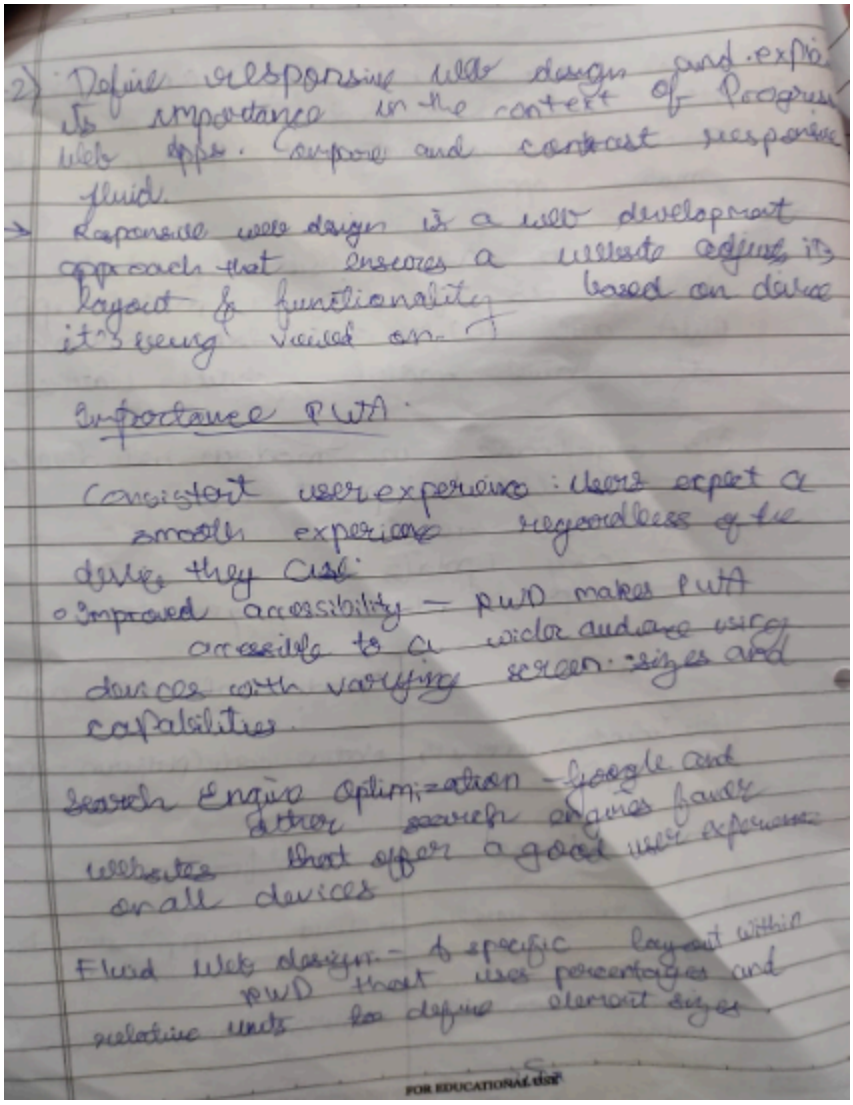
→ A PWA is a website that looks and behaves as if it is a mobile app. PWA are built to take advantage of native mobile device features.

It's significance in modern web development is - reduced development cost
- wider reach
- easy update
- improved user engagement

PWA	Traditional mobile app
Web Technologies (HTML, CSS, JavaScript)	Native code (platform-specific)
Installed from browser	Download from app stores
Accessible through search engines	Reliant on app store discovery
Instantic background update	Requires manual update.

FOR EDUCATIONAL USE

Project Title: **Roll No.**



2) Define responsive web design and explain its importance in the context of Progressive Web Apps. Compare and contrast responsive and fluid.

→ Responsive web design is a web development approach that ensures a website adjusts its layout & functionality based on device it's being viewed on.

Importance PWA:

Consistent user experience: Users expect a smooth experience regardless of the device they use.

- Improved accessibility — PWD makes PWA accessible to a wider audience using devices with varying screen sizes and capabilities.

Search Engine Optimization - Google and other search engines favor websites that offer a good user experience on all devices

Fluid Web design:- A specific layout within PWD that uses percentages and relative units to define element sizes.

Adaptive web design :- This approach involves multiple fixed width layout for device categories.

Q-3) A key player in this PWA universe is "Service Worker". The service worker is a Javascript File that runs on a seperate.

Three phases of lifecycle.

a) Registration

The first phase in the service worker's lifecycle is registering it to the browser. You either specify a scope for a service worker has to.

b) Installation.

Once the service worker is successfully registered, it is not ready to be installed. The service worker script is downloaded by the browser and the browser is attempt to install the service worker.

c) Activation

Once the installation phase is sucessfull, the next phase is the activation phase.

Project Title: **Roll No.**

None of the pages use the service worker active on that page.

Q.4) Uses of using IndexedDB with service worker:

- Offline data access - Improves user experience by allow interaction with the PWA even without an internet connection.

Faster load times - Cached data retrieval from IndexedDB can be served much faster than fetching it.

Improved reliability - Reduces dependences on a constant network connection making PWA more reliable.

Security - IndexedDB provides a secure storage within the browser.

Data management - PWA should have mechanisms to manage storage space avoid excessive data accumulation. In IndexedDB