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 **Vedant Sanap** 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 **2024-2025.**

Lab Assistant

Subject Teacher

Mrs. Kajal Joseph

Principal

Head of Department

Dr. Mrs. Shalu Chopra

Name of the Course : MAD & PWA Lab

Course Code : ITL604

Year/Sem/Class **:** D15A **A.Y.:** 24-25

Faculty Incharge : Mrs. Kajal Joseph.

Lab Teachers **:** Mrs. Kajal Joseph.

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:Home Workout App **Roll No. 47**

Lab Objectives:

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

Lab Outcomes:

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

Index

Sr. No	Experiment Title	LO	DOP	DOS	Grade
1.	To install and configure the Flutter Environment	LO1			
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			

MAD & PWA Lab
Journal

Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	47
Name	Vedant Sanap
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	

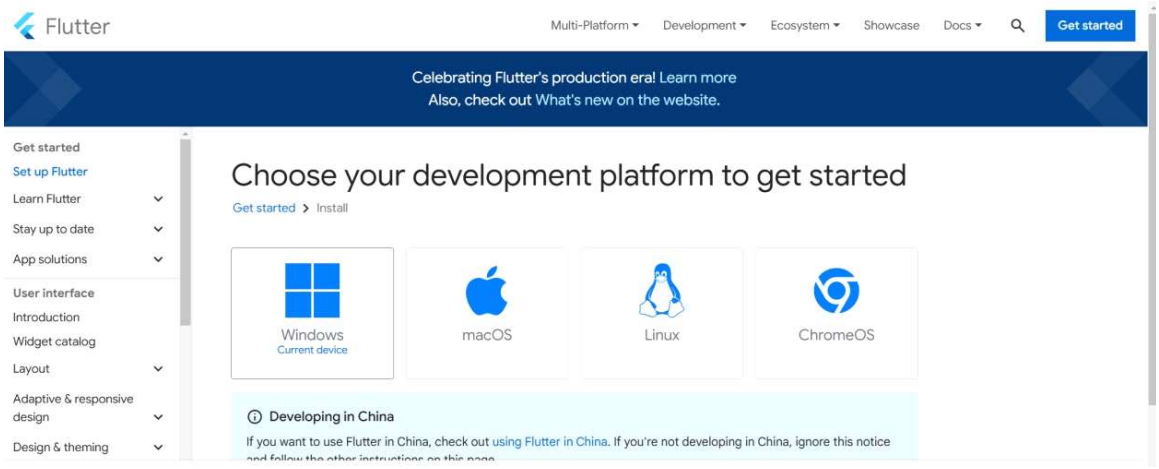
MPL Experiment 1

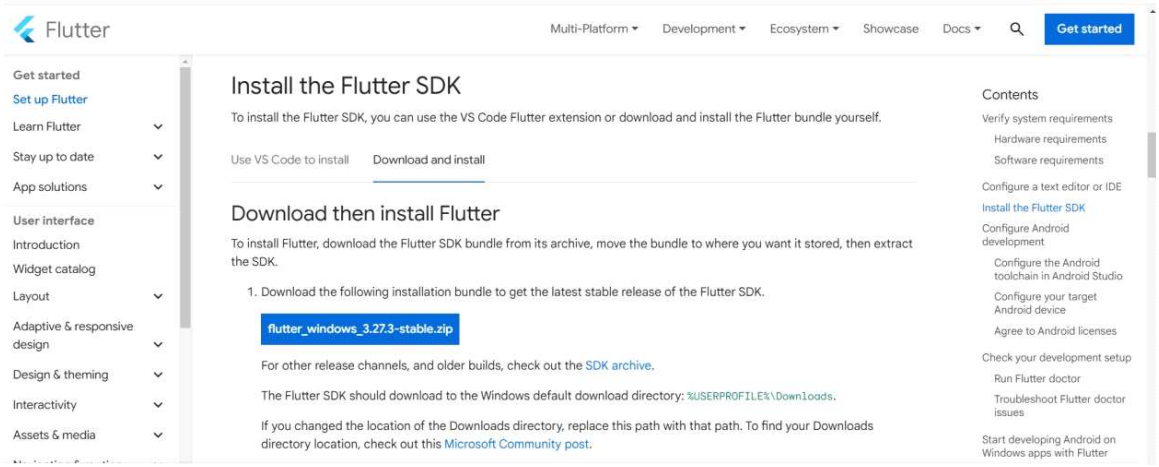
Name: Vedant Sanap **Class:** D15A **Roll No:** 47

Aim: Installation and Configuration of Flutter Environment.

Step 1: Install Flutter

- **Download Flutter SDK**
- Go to the official Flutter website.
- Click on "Get Started."
- Download the appropriate Flutter SDK for your operating system (Windows, macOS, or Linux).



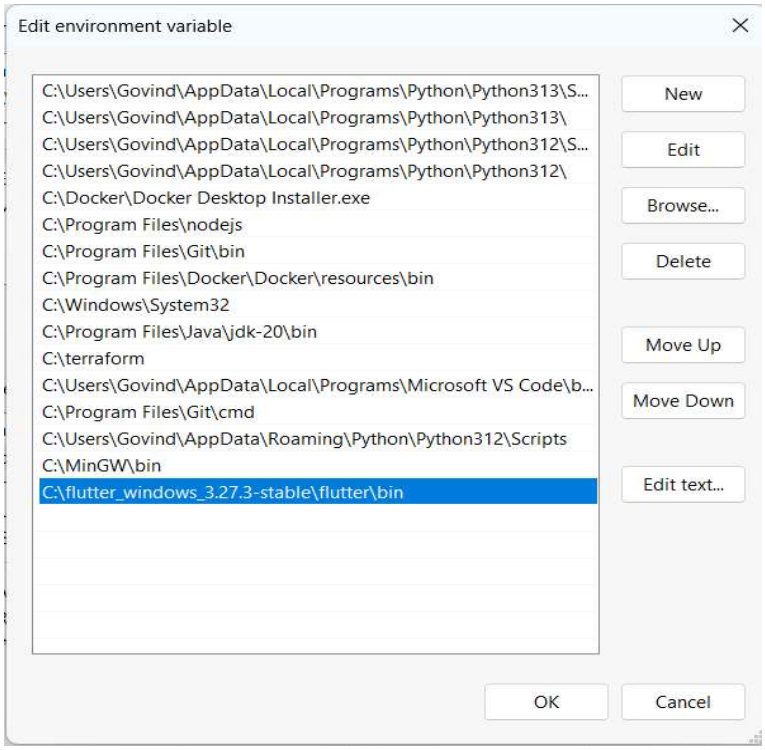


- **Extract the Flutter SDK**
- Extract the downloaded zip file to a preferred location on your computer

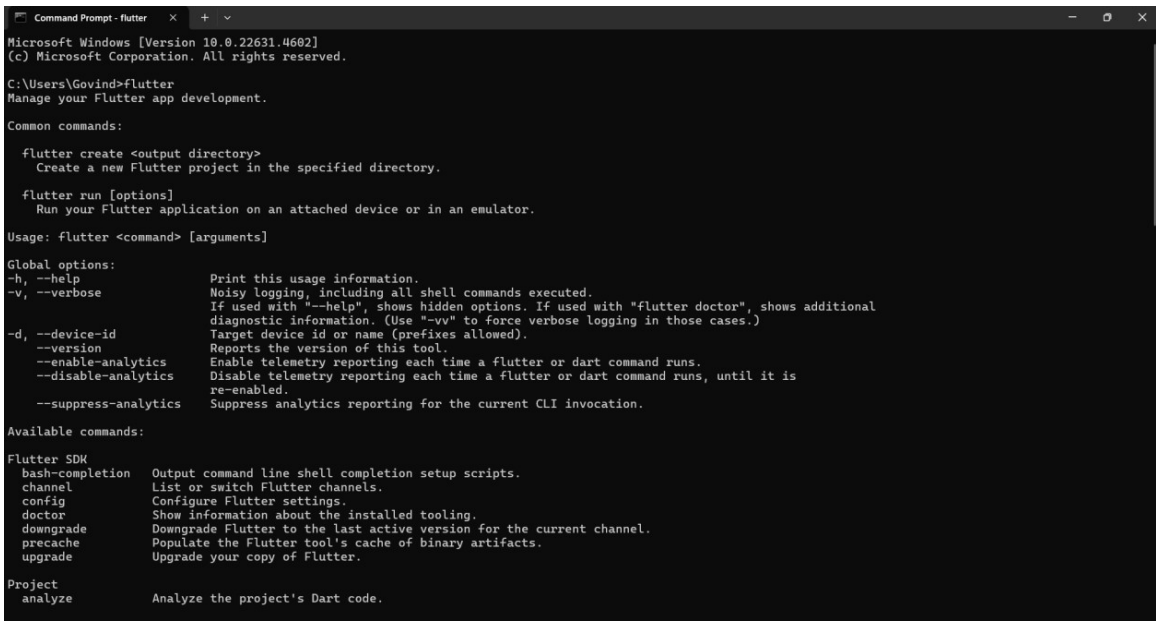
(e.g., C:\src\flutter for Windows).

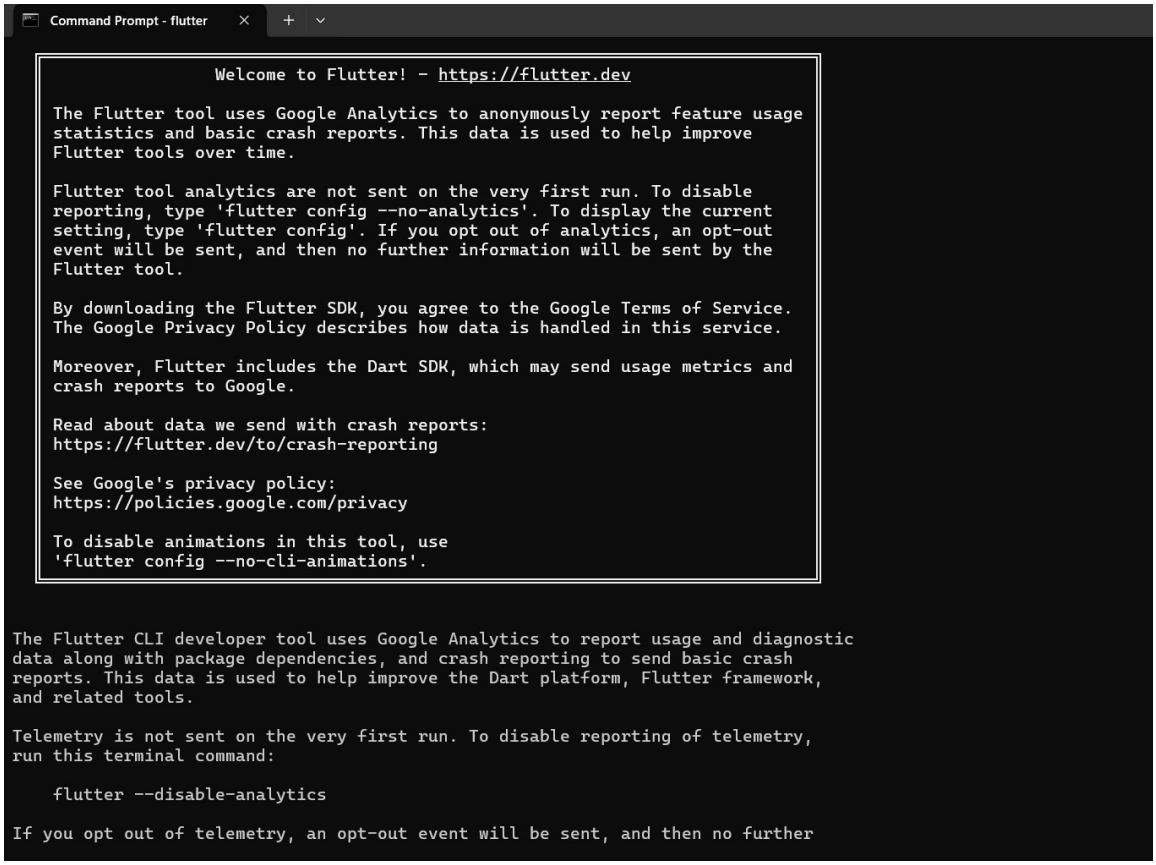
- **Add Flutter to the PATH**
- Locate the flutter\bin directory in the extracted Flutter folder.
- Add this directory to your system's PATH environment variable.

- Windows: Go to Environment Variables > Edit Path > Add the path to flutter\bin.



- **Verify the Installation**
- Open a terminal or command prompt.
- Run the command: flutter and flutter doctor.
- Follow any additional setup instructions displayed in the output.

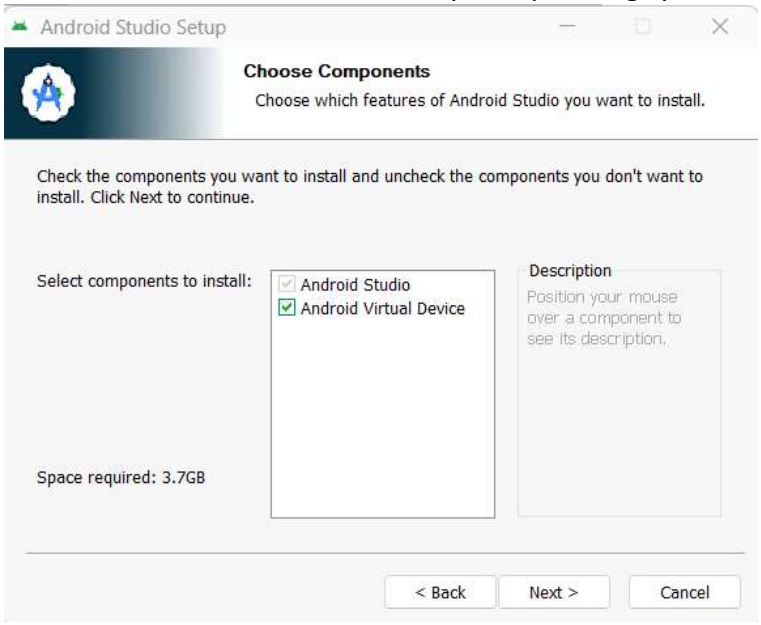




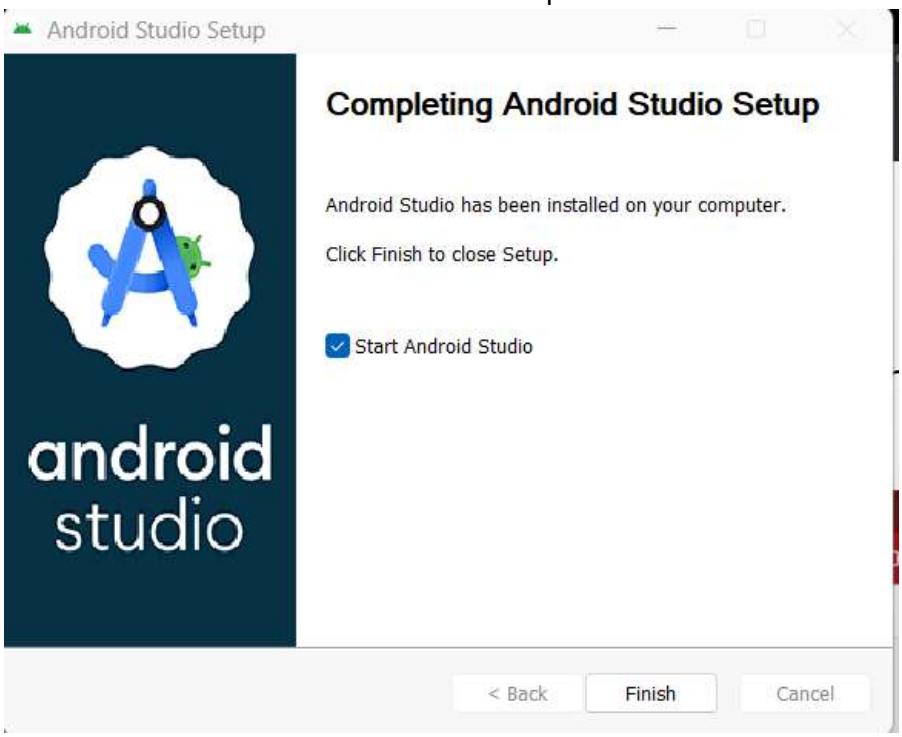
Step 2: Install Android Studio

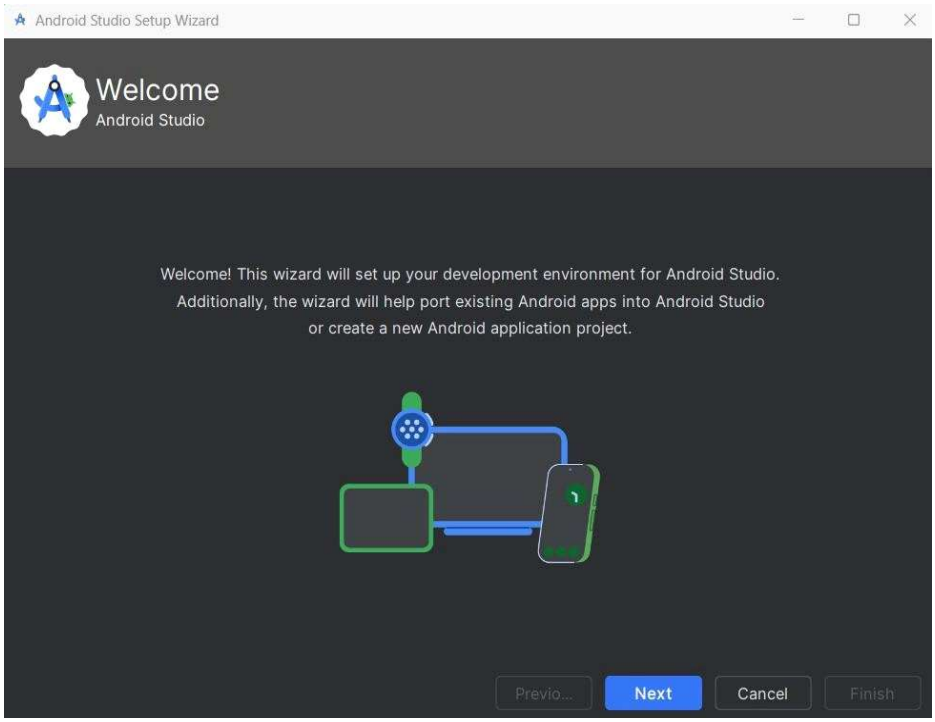
- Download Android Studio

- Go to the Android Studio website.
- Download the installer for your operating system.



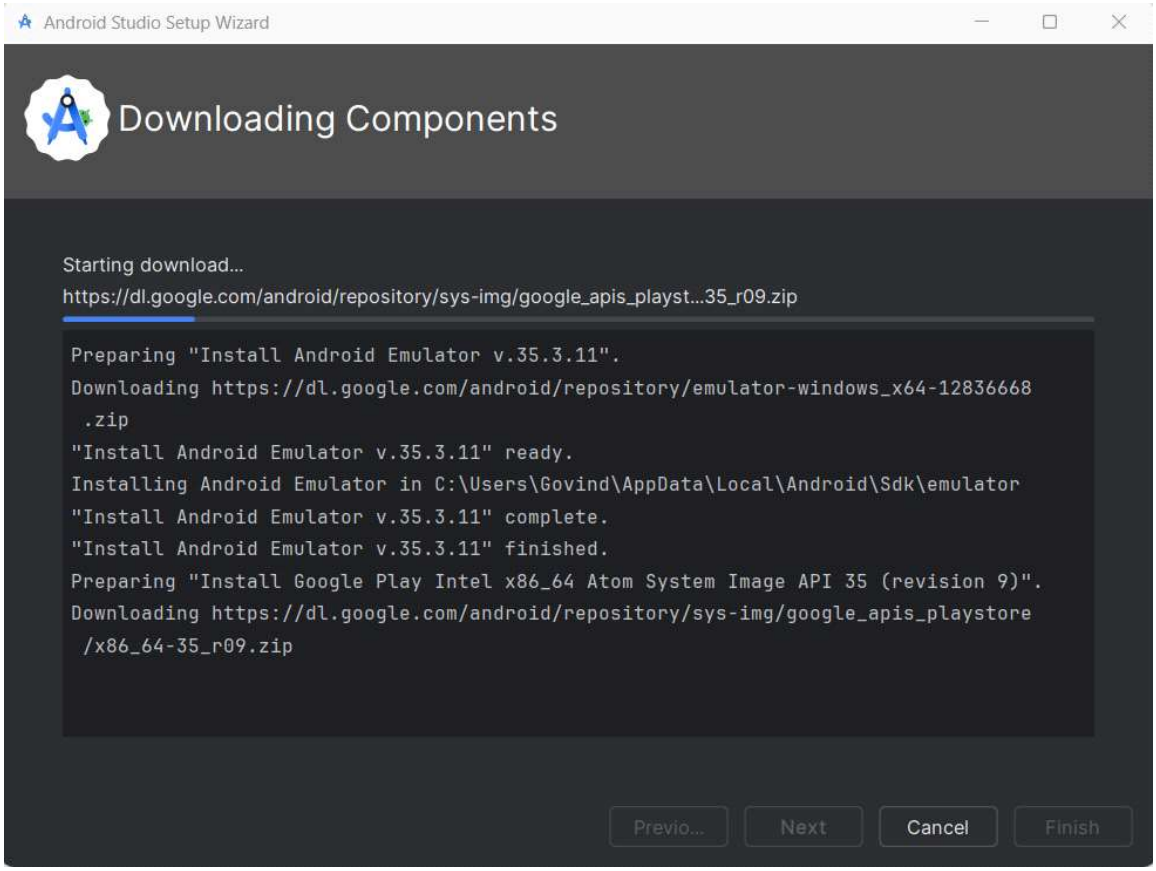
- **Install Android Studio**
- Run the installer and follow the setup wizard.
- Choose the standard installation option.



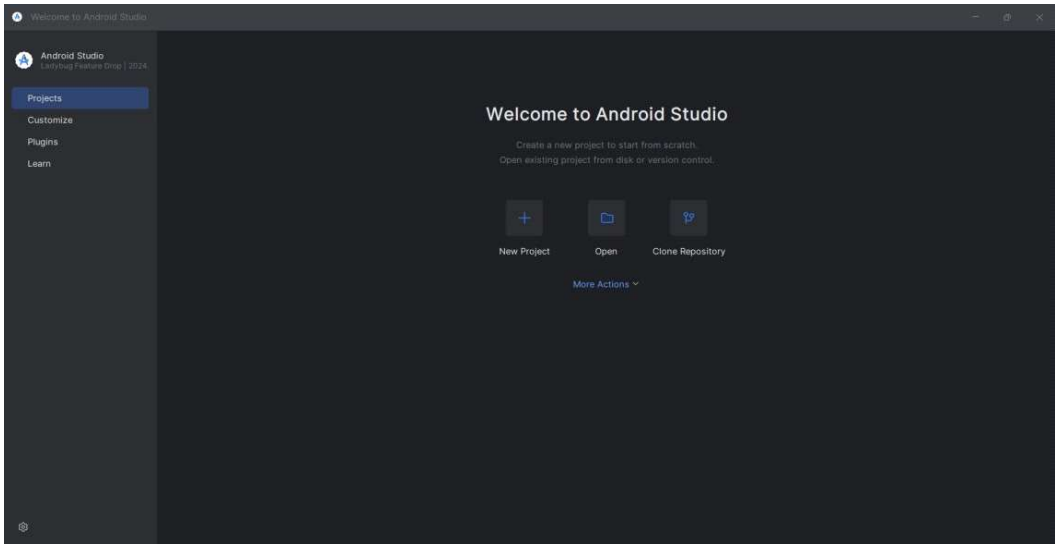


- **Install Android SDK Tools**
- Open Android Studio.

- **Go to Settings/Preferences > Appearance & Behavior > System Settings > Android SDK.**
- Select the latest Android API level.
- Ensure "Android SDK Platform" and "Android Virtual Device (AVD)" are selected.
- Click "Apply" and wait for the components to install**.**

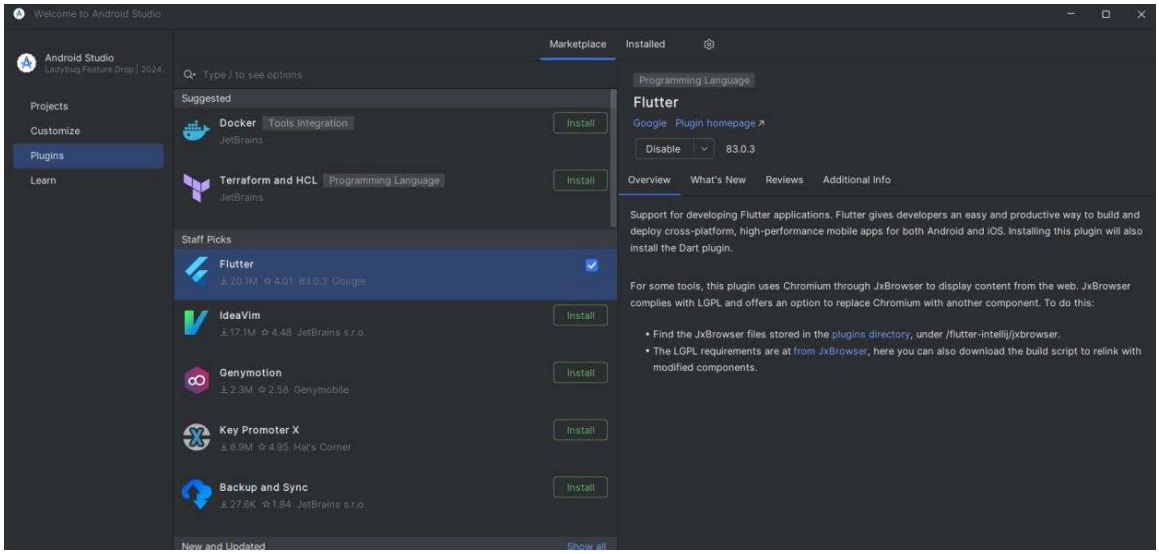


- **Set Up Android Emulator**
- Go to Tools > Device Manager.
- Click "Create Device."
- Select a hardware profile and system image.
- Configure the emulator and click "Finish."



Step 3: Connect Flutter with Android Studio

- Install Flutter and Dart Plugins
- Open Android Studio.
- Go to File > Settings (Windows/Linux) or Android Studio > Preferences (macOS).
- Navigate to Plugins.
- Search for "Flutter" and click "Install." Dart will be installed automatically.
- Restart Android Studio.



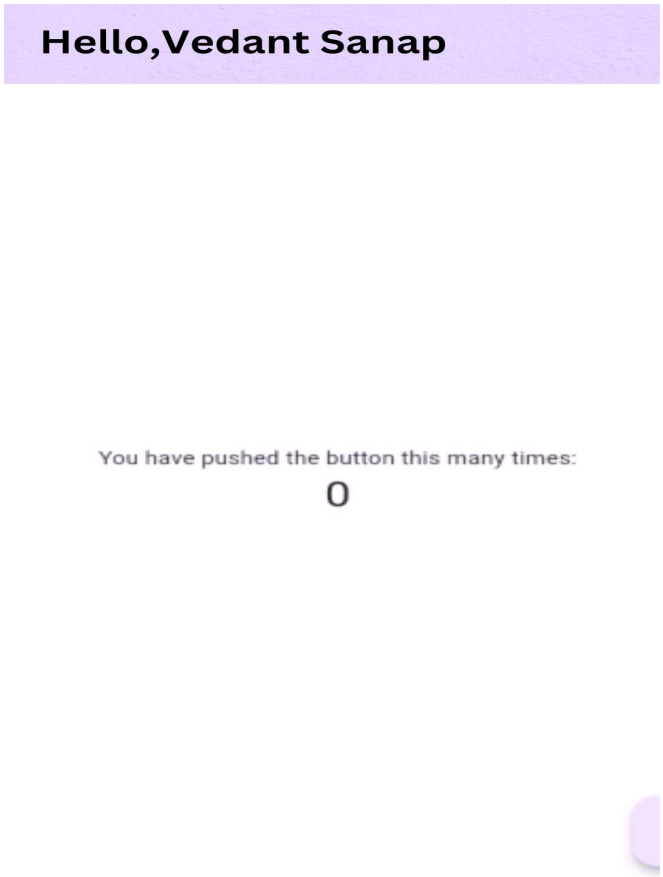
Step 4: Test Flutter Installation

- Create a New Flutter Project

- Open Android Studio.
- Click on New Flutter Project.

- Enter project details and select the Flutter SDK path.
- Click "Finish" to create the project.

Conclusion: Flutter and Android Studio setup is now completed.



MAD & PWA Lab
Journal

Experiment No.	02
Experiment Title.	To design Flutter UI by including common widgets.
Roll No.	47
Name	Vedant Sanap
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

MPL EXP-2

Vedant Sanap
D15A - 47

Aim: To Design Flutter UI by including common widgets

Theory:

Common Widgets used in Flutter app:

1. **Scaffold:** Provides the basic visual structure for a screen, including app bars, bodies, and bottom navigation.
2. **AppBar:** A widget that is displayed at the top of the screen.
3. **Text:** Displays an immutable piece of text.
4. **TextField:** Allows the user to enter text.
5. **Image.network:** Displays an image from a URL.
6. **SizedBox:** A box with a specified size.
7. **ListView.builder:** Creates a scrollable list of widgets that are built on demand.
8. **Card:** A panel with slightly rounded corners and a shadow.
9. **InkWell:** A rectangular area of a UI that responds to touch.
10. **Column:** Arranges its children in a vertical array.
11. **Row:** Arranges its children in a horizontal array.
12. **Padding:** Inserts space around another widget.
13. **Align:** Aligns its child within itself.
14. **Container:** A widget that combines common painting, positioning, and sizing widgets.
15. **CircularProgressIndicator:** A widget that indicates that a task is in progress.
16. **BottomNavigationBar:** A bar at the bottom of the screen for selecting different destinations.
17. **BottomNavigationBarItem:** An item in a bottom navigation bar.
18. **SingleChildScrollView:** A box in which a single widget can be scrolled.
19. **AnimatedOpacity:** Animates the opacity of a widget.
20. **Center:** Centers its child within itself.
21. **MaterialPageRoute:** A route that replaces the entire screen with a platform-adaptive transition.
22. **Navigator.push:** Method to push a route onto the navigator's stack.
23. **Icon:** A graphical icon widget.
24. **InputDecoration:** Styles the visual appearance of a TextField.
25. **OutlineInputBorder:** A border for TextField with a rectangular outlinE.

Code:

main.dart:

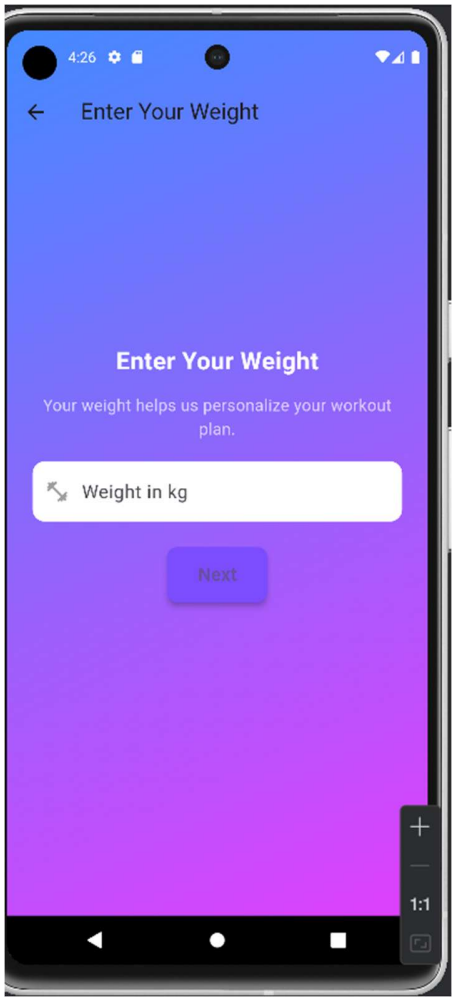
```
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:get/get.dart';
import '../screen/weight_page.dart'; // Import the weight entry page
import 'login.dart'; // Import the login page

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

  @override
  State<Homepage> createState() => _HomepageState();
}

class _HomepageState extends State<Homepage> {
  void selectGender(String gender) {
    Navigator.push(
      context,
      MaterialPageRoute(
        builder: (context) => WeightPage(selectedGender: gender),
      ),
    );
  }

  // Sign out function
  void signOut() async {
    await FirebaseAuth.instance.signOut();
    Get.offAll(() => Login()); // Redirect to login page after sign out
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      extendBodyBehindAppBar: true,
      appBar: AppBar(
```

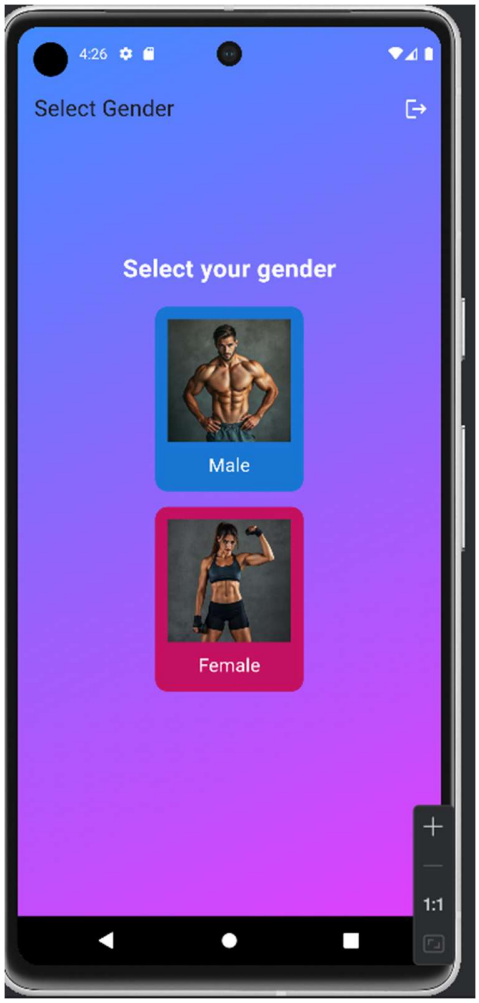
```
  title: Text("Select Gender"),
  backgroundColor: Colors.transparent,
  elevation: 0,
  actions: [
   IconButton(
    icon: Icon(Icons.logout, color: Colors.white),
    onPressed: signOut,
   ),
  ],
 ),
 body: Container(
  decoration: BoxDecoration(
   gradient: LinearGradient(
    colors: [Colors.blueAccent, Colors.purpleAccent],
    begin: Alignment.topLeft,
    end: Alignment.bottomRight,
   ),
  ),
  child: Center(
   child: Padding(
    padding: const EdgeInsets.symmetric(horizontal: 25.0),
    child: Column(
     mainAxisSize: MainAxisSize.min,
     children: [
      Text(
       "Select your gender",
       style: TextStyle(
        fontSize: 24,
        fontWeight: FontWeight.bold,
        color: Colors.white,
       ),
      ),
      SizedBox(height: 20),
      GestureDetector(
       onTap: () => selectGender("Male"),
       child: Container(
        decoration: BoxDecoration(
```

```
      color: Colors.blue.shade700,
      borderRadius: BorderRadius.circular(12),
    ),
    padding: EdgeInsets.all(12),
    child: Column(
      children: [
        Image.asset(
          "lib/assets/images/male.jpg",
          height: 120,
        ),
        SizedBox(height: 10),
        Text(
          "Male",
          style: TextStyle(fontSize: 18, color: Colors.white),
        ),
      ],
    ),
  ),
),
SizedBox(height: 15),
GestureDetector(
  onTap: () => selectGender("Female"),
  child: Container(
    decoration: BoxDecoration(
      color: Colors.pinkAccent.shade700,
      borderRadius: BorderRadius.circular(12),
    ),
    padding: EdgeInsets.all(12),
    child: Column(
      children: [
        Image.asset(
          "lib/assets/images/female.jpg",
          height: 120,
        ),
        SizedBox(height: 10),
        Text(
          "Female",
```

```
                    style: TextStyle(fontSize: 18, color: Colors.white),
                  ),
                ],
              ),
            ),
          ),
        ),
      ),
    ),
  );
 }
}
```

Output:


4:26
Enter Your Weight
Enter Your Weight
Your weight helps us personalize your workout plan.
Weight in kg
Next
1:1


4:26
Select Gender
Select your gender
Male
Female
1:1

MAD & PWA Lab
Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	47
Name	Vedant Sanap
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

MPL EXPERIMENT-3

Name: Vedant Sanap
Class/Roll No : D15A-47

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

Theory: -
Incorporating Visual Elements in Flutter: Icons, Images, and Custom Fonts
Flutter is a powerful open-source UI framework that enables developers to build natively compiled applications for mobile, web, and desktop platforms—all from a single codebase. One of Flutter's greatest strengths is its flexibility in crafting highly customizable UIs.
This practical guide focuses on integrating essential visual elements—icons, images, and custom fonts into a Flutter application. These elements enhance visual appeal, improve usability, and create a more engaging user experience.
Importance of Visual Elements in App Development

- **Enhanced User Experience** – Icons and images make applications more visually appealing and user friendly.
- **Efficient Communication** – Well-designed icons convey information quickly, reducing the need for lengthy text.
- **Brand Identity** – Custom icons and images reinforce branding, making an app more memorable.

Managing Assets in a Flutter App
When a Flutter app is built, it consists of both code and assets. Assets include static files such as images, icons, fonts, and configuration files, which are deployed and available at runtime. Flutter supports multiple image formats, including JPEG, WebP, PNG, GIF, BMP, and WBMP.

Adding Icons in Flutter
Flutter provides built-in Material Design icons through the Icons class. Custom icons can also be integrated using third-party packages like flutter_launcher_icons and font_awesome_flutter.
Example (Built-in Material Icons):

```
Icon(
 Icons.home,
 size: 40,
);
```

Adding Images in Flutter
Flutter supports images from three primary sources: Assets, Network, and Local Storage (Memory or File
System).

1. Using Asset Images (Local Project Files)
To use an image stored in the project folder:
Place the image inside the assets/images/ folder.
Declare it in pubspec.yaml:
flutter:
 assets:
 - assets/images/sample.png
Display it in the app:
Image.asset('assets/images/sample.png');

2. Using Network Images (Fetched from the Internet)
Flutter simplifies loading images from the web using Image.network. Additional properties like height,
width, fit, and color can be specified.
Example:
Image.network('https://example.com/sample.jpg');

3. Using Local Storage (Memory or File System)
Images stored on the user's device can also be displayed using packages like image_picker or file_picker.
Adding Custom Fonts in Flutter
By default, Flutter uses the Roboto font. However, custom fonts can be added to create a unique visual
Identity.

Steps to Add a Custom Font:
1. Download the font and place it in the assets/fonts/ folder.

2. Declare the font in pubspec.yaml:
yaml
flutter:
 fonts:
 - family: CustomFont
 fonts:
 - asset: assets/fonts/CustomFont.ttf

3. Use the font in your app
Text(
 'Custom Font Example',
 style: TextStyle(fontFamily: 'CustomFont', fontSize: 24),
);

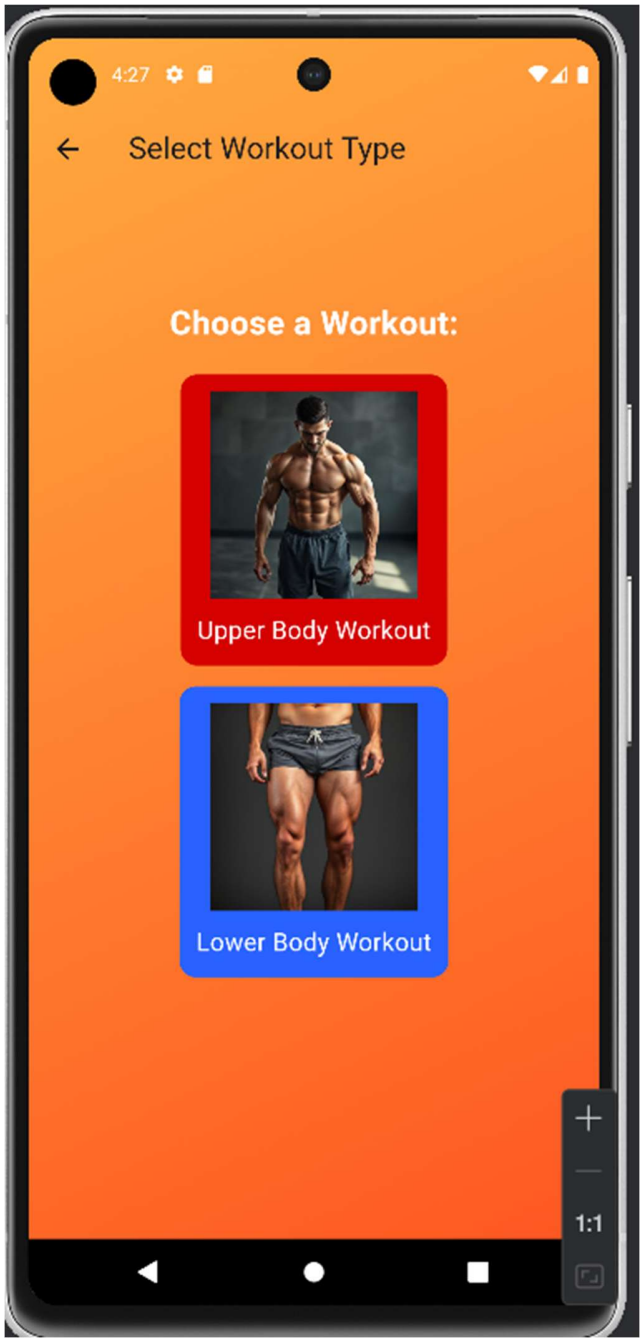
Code:

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

class WorkoutPage extends StatelessWidget {
  final String selectedGender;
  final double weight;

  const WorkoutPage({super.key, required this.selectedGender, required this.weight});

  void startWorkout(BuildContext context, String workoutType) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text("Starting $workoutType Workout")),
    );
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      extendBodyBehindAppBar: true,
      appBar: AppBar(
        title: Text("Select Workout Type"),
        backgroundColor: Colors.transparent,
        elevation: 0,
      ),
      body: Container(
        decoration: BoxDecoration(
          gradient: LinearGradient(
            colors: [Colors.orangeAccent, Colors.deepOrange],
            begin: Alignment.topLeft,
            end: Alignment.bottomRight,
          ),
        ),
        child: Center(
          child: Padding(
            padding: const EdgeInsets.symmetric(horizontal: 25.0),
            child: Column(
              mainAxisSize: MainAxisSize.min,
              children: [
                Text(
                  "Choose a Workout:",
                  style: TextStyle(
                    fontSize: 24,
                    fontWeight: FontWeight.bold,
                    color: Colors.white,
                  ),
```

```
),
SizedBox(height: 20),
GestureDetector(
  onTap: () => startWorkout(context, "Upper Body"),
  child: Container(
    decoration: BoxDecoration(
      color: Colors.redAccent.shade700,
      borderRadius: BorderRadius.circular(12),
    ),
    padding: EdgeInsets.all(12),
    child: Column(
      children: [
        Image.asset(
          "lib/assets/images/upper.jpg", // Replace with actual image
          height: 150,
        ),
        SizedBox(height: 10),
        Text(
          "Upper Body Workout",
          style: TextStyle(fontSize: 18, color: Colors.white),
        ),
      ],
    ),
  ),
),
SizedBox(height: 15),
GestureDetector(
  onTap: () => startWorkout(context, "Lower Body"),
  child: Container(
    decoration: BoxDecoration(
      color: Colors.blueAccent.shade700,
      borderRadius: BorderRadius.circular(12),
    ),
    padding: EdgeInsets.all(12),
    child: Column(
      children: [
        Image.asset(
          "lib/assets/images/lower.jpg", // Replace with actual image
          height: 150,
        ),
        SizedBox(height: 10),
        Text(
          "Lower Body Workout",
          style: TextStyle(fontSize: 18, color: Colors.white),
        ),
      ],
    ),
  ),
),
```

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

Output:


4:27
Select Workout Type
Choose a Workout:
Upper Body Workout
Lower Body Workout
1:1

MAD & PWA Lab
Journal

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget
Roll No.	47
Name	Vedant Sanap
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

MPL EXP-4

Ansh Sarfare D15A - 49

Aim: To create an interactive form using form widget

Code:

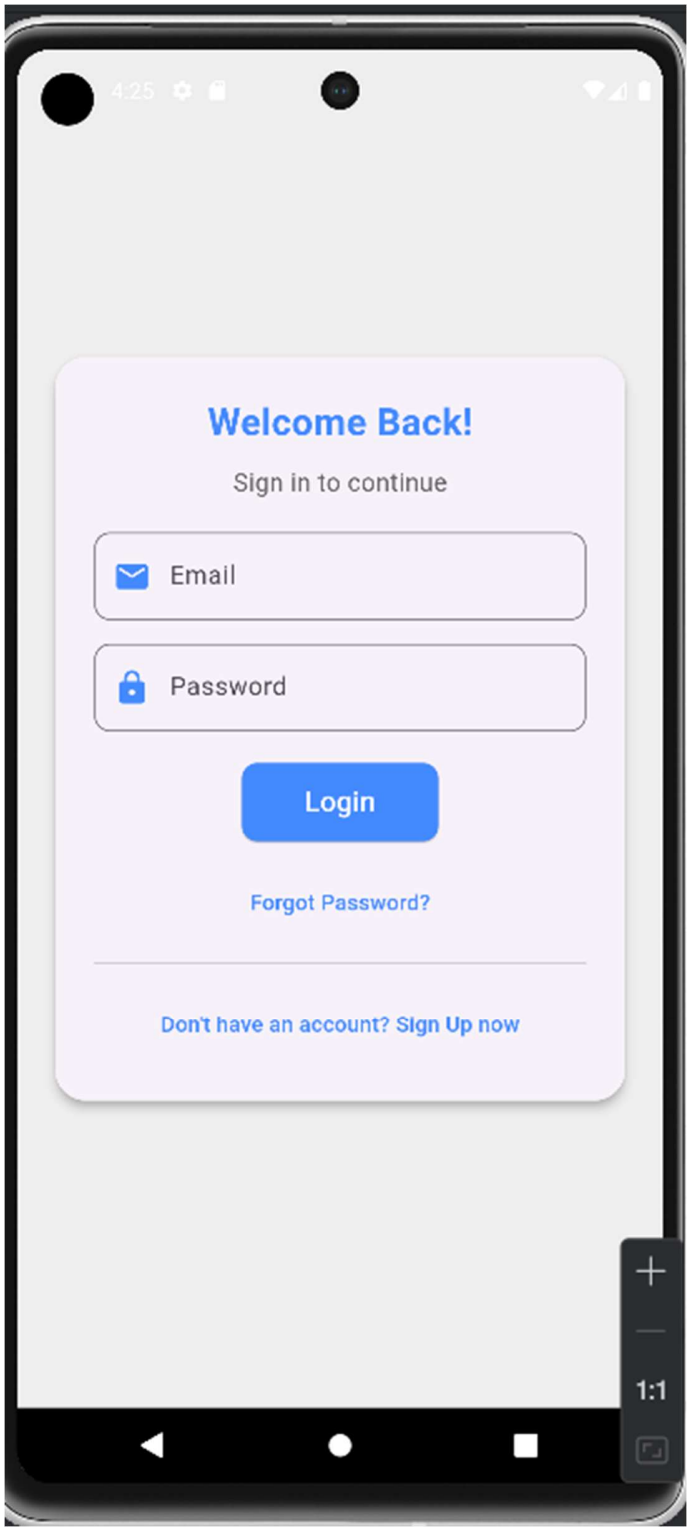
```
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:get/get.dart';
import 'signup.dart';
import 'forgot.dart';

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

  @override
  State<Login> createState() => _LoginState();
}

class _LoginState extends State<Login> {
  TextEditingController email = TextEditingController();
  TextEditingController password = TextEditingController();

  signIn() async {
    if (email.text.isEmpty || password.text.isEmpty) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text("Please fill in all fields"),
          backgroundColor: Colors.redAccent,
        ),
      );
      return;
    }

    try {
      await FirebaseAuth.instance.signInWithEmailAndPassword(
        email: email.text,
        password: password.text,
      );
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text("Login Successful!"),
          backgroundColor: Colors.green,
        ),
```

```
    );
    Get.offAllNamed('/home'); // Navigate to home page
  } catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text("Error: ${e.toString()}"),
        backgroundColor: Colors.red,
      ),
    );
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.grey[200],
    body: Center(
      child: Padding(
        padding: const EdgeInsets.all(20.0),
        child: Card(
          elevation: 5,
          shape: RoundedRectangleBorder(
            borderRadius: BorderRadius.circular(20),
          ),
          child: Padding(
            padding: const EdgeInsets.all(25.0),
            child: Column(
              mainAxisSize: MainAxisSize.min,
              crossAxisAlignment: CrossAxisAlignment.center,
              children: [
                Text(
                  "Welcome Back!",
                  style: TextStyle(
                    fontSize: 24,
                    fontWeight: FontWeight.bold,
                    color: Colors.blueAccent,
                  ),
                ),
                SizedBox(height: 10),
                Text(
                  "Sign in to continue",
                  style: TextStyle(fontSize: 16, color: Colors.grey[700]),
                ),
                SizedBox(height: 20),
```

```
TextField(
  controller: email,
  decoration: InputDecoration(
    labelText: "Email",
    prefixIcon: Icon(Icons.email, color: Colors.blueAccent),
    border: OutlineInputBorder(
      borderRadius: BorderRadius.circular(10),
    ),
  ),
  keyboardType: TextInputType.emailAddress,
),
SizedBox(height: 15),
TextField(
  controller: password,
  decoration: InputDecoration(
    labelText: "Password",
    prefixIcon: Icon(Icons.lock, color: Colors.blueAccent),
    border: OutlineInputBorder(
      borderRadius: BorderRadius.circular(10),
    ),
  ),
  obscureText: true,
),
SizedBox(height: 20),
ElevatedButton(
  onPressed: signIn,
  style: ElevatedButton.styleFrom(
    backgroundColor: Colors.blueAccent,
    foregroundColor: Colors.white,
    shape: RoundedRectangleBorder(
      borderRadius: BorderRadius.circular(10),
    ),
    padding: EdgeInsets.symmetric(vertical: 12, horizontal: 40),
  ),
  child: Text(
    "Login",
    style: TextStyle(fontSize: 18),
  ),
),
SizedBox(height: 15),
TextButton(
  onPressed: () => Get.to(() => Forgot()),
  child: Text(
    "Forgot Password?",
```

```
                    style: TextStyle(color: Colors.blueAccent),
                  ),
                ),
                Divider(thickness: 1, height: 30),
                TextButton(
                  onPressed: () => Get.to(() => Signup()),
                  child: Text(
                    "Don't have an account? Sign Up now",
                    style: TextStyle(color: Colors.blueAccent),
                  ),
                ),
              ],
            ),
          ),
        ),
      ),
    ),
  );
 }
}
```

Output:


4:25
Welcome Back!
Sign in to continue
Email
Password
Login
Forgot Password?
Don't have an account? Sign Up now
1:1

MAD & PWA Lab
Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	47
Name	Vedant Sanap
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

MPL EXPERIMENT-5

Name: Ansh Sarfare **Class/Roll No :** D15A-49

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

Theory: -

Navigation, Routing, and Gesture Handling in Flutter
In Flutter, screens or pages are referred to as routes, and each route is essentially a widget. This concept is similar to Activities in Android. Navigating between pages defines an app's workflow, and the mechanism for handling this is known as routing.
Flutter provides a built-in routing system using MaterialPageRoute, along with the Navigator.push() and Navigator.pop() methods to move between routes.
Additionally, gestures allow apps to respond to user interactions like taps, swipes, and drags, making applications more dynamic and user-friendly.

Navigation and Routing in Flutter,

1. Using the Navigator Widget

Flutter's Navigator widget manages a stack of routes, enabling seamless navigation between screens.
Pushing a Route: Moves to a new screen using Navigator.push().
Popping a Route: Returns to the previous screen using Navigator.pop().

Example:

```
ElevatedButton(
 onPressed: () {
 Navigator.push(
 context,
 MaterialPageRoute(builder: (context) => SecondScreen()),
 );
 },
 child: Text('Go to Second Screen'),
);
```

2. Using Named Routes

For larger applications, named routes provide a cleaner and more structured way to manage navigation.
Step 1: Define Routes in MaterialApp

```
MaterialApp(
```

```
 initialRoute: '/',
 routes: {
 '/': (context) => HomeScreen(),
 '/second': (context) => SecondScreen(),
 },
);
```

Step 2: Navigate Using Navigator.pushNamed()

```
Navigator.pushNamed(context, '/second');
```

Handling Gestures in Flutter

Gestures enable user interaction through taps, swipes, pinches, and drags. Flutter provides various widgets and gesture detectors to manage these interactions effectively.

1. Tap Gestures

Taps are one of the most common interactions and can be handled using:

GestureDetector

InkWell

ElevatedButton

Example (Tap Gesture using GestureDetector):

```
GestureDetector(
 onTap: () {
 print("Tapped!");
 },
 child: Container(
 padding: EdgeInsets.all(20),
 color: Colors.blue,
 child: Text('Tap Me'),
 ),
);
```

2. Long Press Gestures

Long-press interactions can be captured using the onLongPress callback in GestureDetector or InkWell.

```
InkWell(
 onLongPress: () {
 print("Long Pressed!");
 },
 child: Container(
 padding: EdgeInsets.all(20),
 color: Colors.red,
 child: Text('Long Press Me'),
 ),
);
```

3. Swipe and Drag Gestures

Flutter provides built-in methods like onHorizontalDragUpdate and onVerticalDragUpdate to detect
swipe and drag actions.
Example (Swipe Detection):

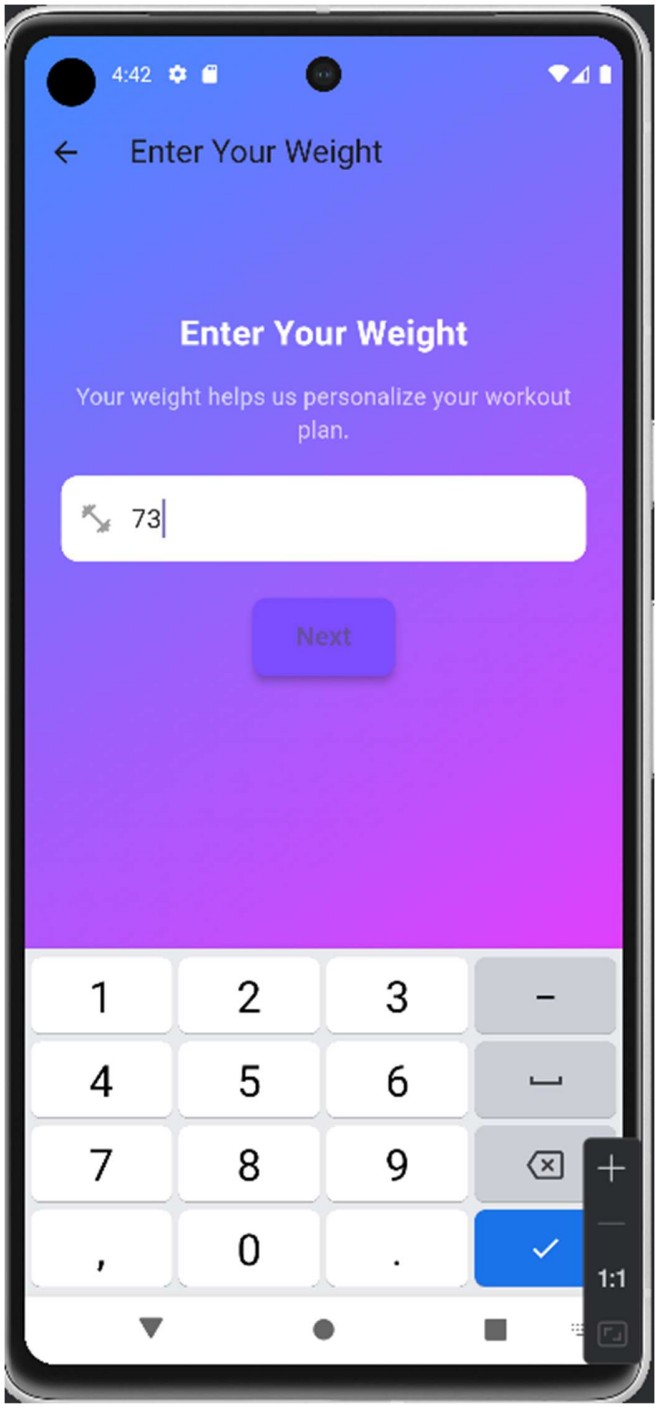
```
GestureDetector(
 onHorizontalDragUpdate: (details) {
 if (details.primaryDelta! > 0) {
 print("Swiped Right!");
 } else {
 print("Swiped Left!");
 }
 },
 child: Container(
 padding: EdgeInsets.all(20),
 color: Colors.green,
 child: Text('Swipe Me'),
 ),
);
```

Code:

```
import 'package:flutter/material.dart';
import 'workouts_page.dart'; // Import
the next page

class WeightPage extends
StatefulWidget {
 final String selectedGender;
 const WeightPage({super.key, required
this.selectedGender});

 @override
 State<WeightPage> createState() =>
_WeightPageState();
}

class _WeightPageState extends
State<WeightPage> {
 TextEditingController weightController
= TextEditingController();

 void proceedToWorkout() {
  if (weightController.text.isEmpty) {

ScaffoldMessenger.of(context).showSna
ckBar(
      SnackBar(content: Text("Please
enter your weight")),
    );
    return;
  }

  Navigator.push(
    context,
    MaterialPageRoute(
      builder: (context) => WorkoutPage(
        selectedGender:
widget.selectedGender,
        weight:
double.parse(weightController.text),
      ),
    ),
  );
 }

 @override
 Widget build(BuildContext context) {
  return Scaffold(
    extendBodyBehindAppBar: true,
    appBar: AppBar(
      title: Text("Enter Your Weight"),
      backgroundColor:
Colors.transparent,
      elevation: 0,
    ),
    body: Container(
      decoration: BoxDecoration(
        gradient: LinearGradient(
          colors: [Colors.blueAccent,
Colors.purpleAccent],
          begin: Alignment.topLeft,
          end: Alignment.bottomRight,
        ),
      ),
      child: Center(
        child: Padding(
          padding: const
EdgeInsets.symmetric(horizontal: 25.0),
          child: Column(
            mainAxisSize:
MainAxisSize.min,
            children: [
              Text(
                "Enter Your Weight",
                style: TextStyle(
                  fontSize: 24,
                  fontWeight: FontWeight.bold,
                  color: Colors.white,
                ),
              ),
              SizedBox(height: 15),
              Text(
                "Your weight helps us
personalize your workout plan.",
                style: TextStyle(fontSize: 16,
```

```
color: Colors.white70),
              textAlign: TextAlign.center,
            ),
            SizedBox(height: 20),
            TextField(
              controller: weightController,
              decoration: InputDecoration(
                filled: true,
                fillColor: Colors.white,
                border: OutlineInputBorder(
                  borderRadius:
BorderRadius.circular(12),
                  borderSide:
BorderSide.none,
                ),
                hintText: "Weight in kg",
                prefixIcon:
Icon(Icons.fitness_center, color:
Colors.grey),
              ),
              keyboardType:
TextInputType.number,
              style: TextStyle(fontSize: 18),
            ),
            SizedBox(height: 25),
            ElevatedButton(
              onPressed:
proceedToWorkout,
                style:
ElevatedButton.styleFrom(
                  padding:
EdgeInsets.symmetric(vertical: 14,
horizontal: 30),
                  shape:
RoundedRectangleBorder(
                    borderRadius:
BorderRadius.circular(10),
                  ),
                  backgroundColor:
Colors.deepPurpleAccent,
                  elevation: 5,
                ),
                child: Text(
                  "Next",
                  style: TextStyle(fontSize: 18,
fontWeight: FontWeight.bold),
                ),
              ),
            ],
          ),
        ),
      ),
    ),
  );
 }
}
```

Output:


4:42
Enter Your Weight
Enter Your Weight
Your weight helps us personalize your workout plan.
73
Next
1
2
3
4
5
6
7
8
9
,
0
.
1:1

MAD & PWA Lab
Journal

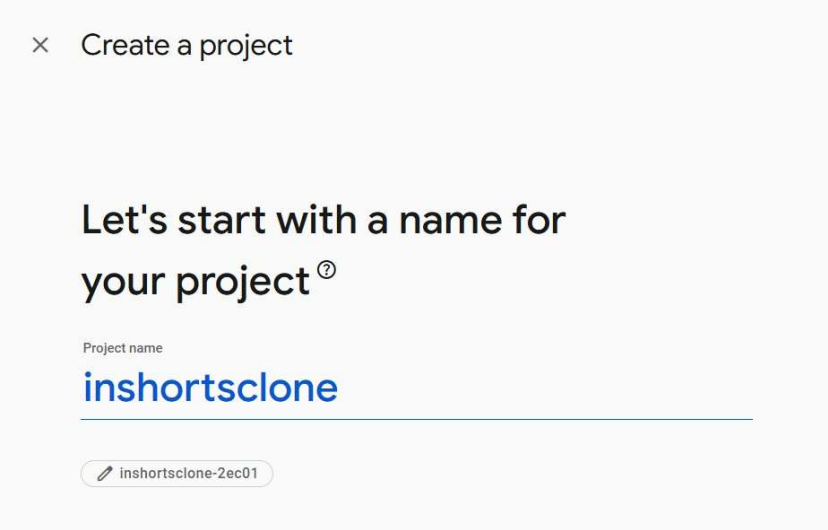
Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	47
Name	Vedant Sanap
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	

EXPERIMENT NO:- 6

Name:- Vedant Sanap D15A Roll-no:-47

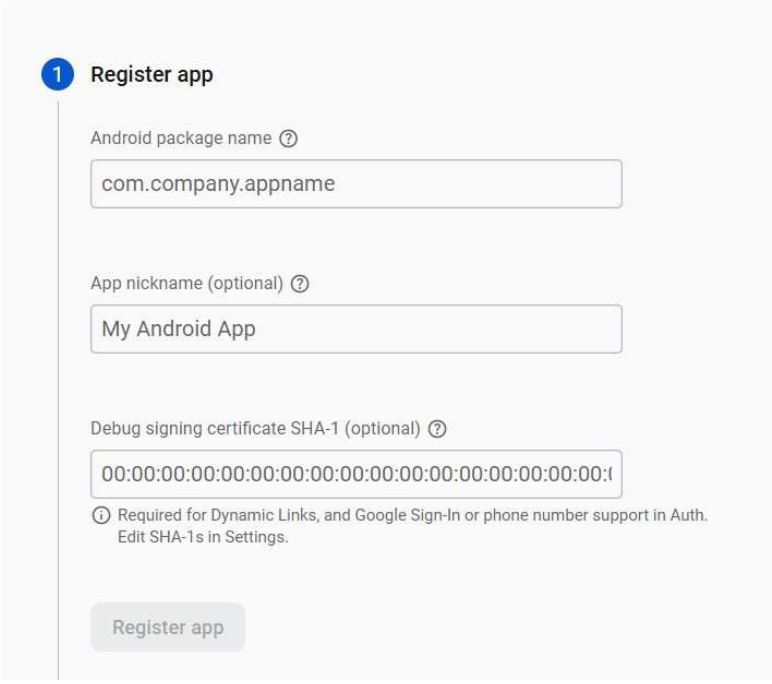
Aim:- To Connect flutter UI with firebase database

Creating a New Firebase Project



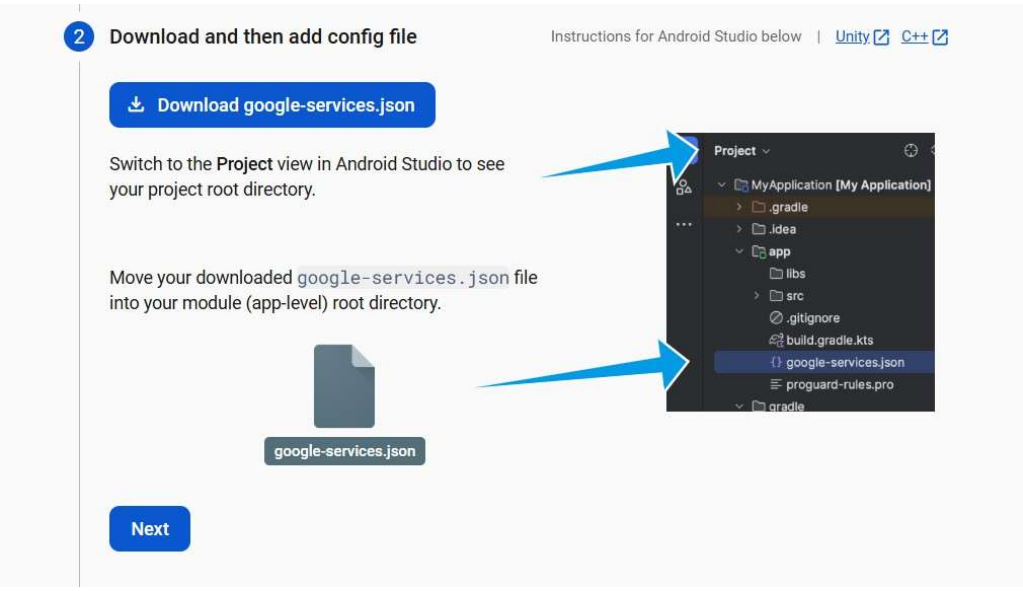
First, log in with your Google account to manage your Firebase projects. From within the Firebase dashboard, select the Create new project button and give it a name

In order to add Android support to our Flutter application, select the Android logo from the dashboard. This brings us to the following screen:

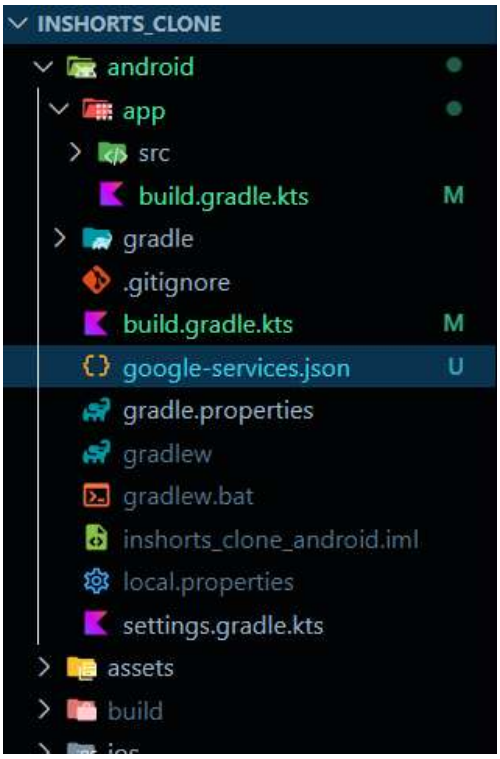


The most important thing here is to match up the Android package name that you choose here with the one inside of our application.

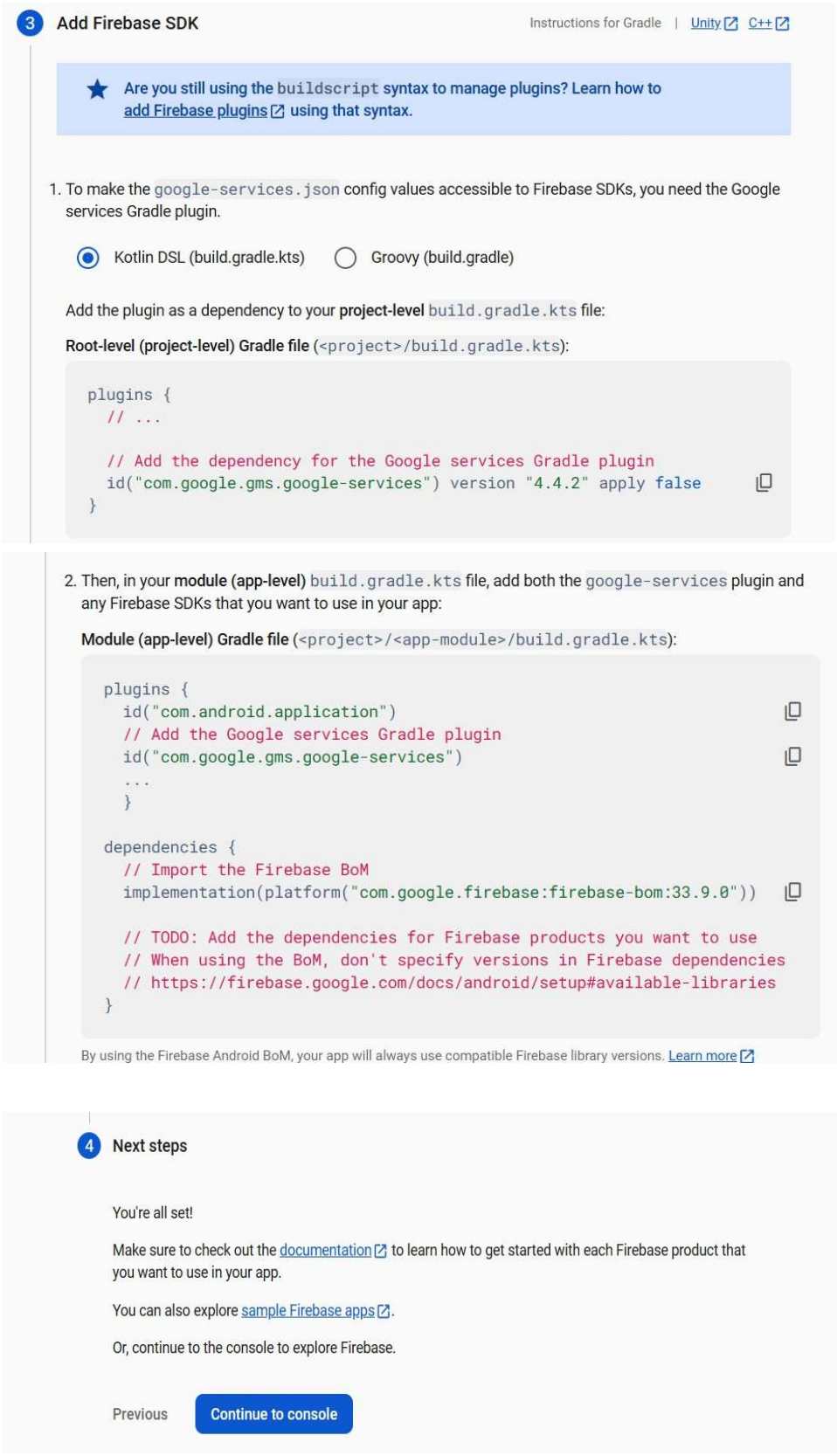
Then download the google-services.json file, that you will get.



put that file in the android folder (root level)



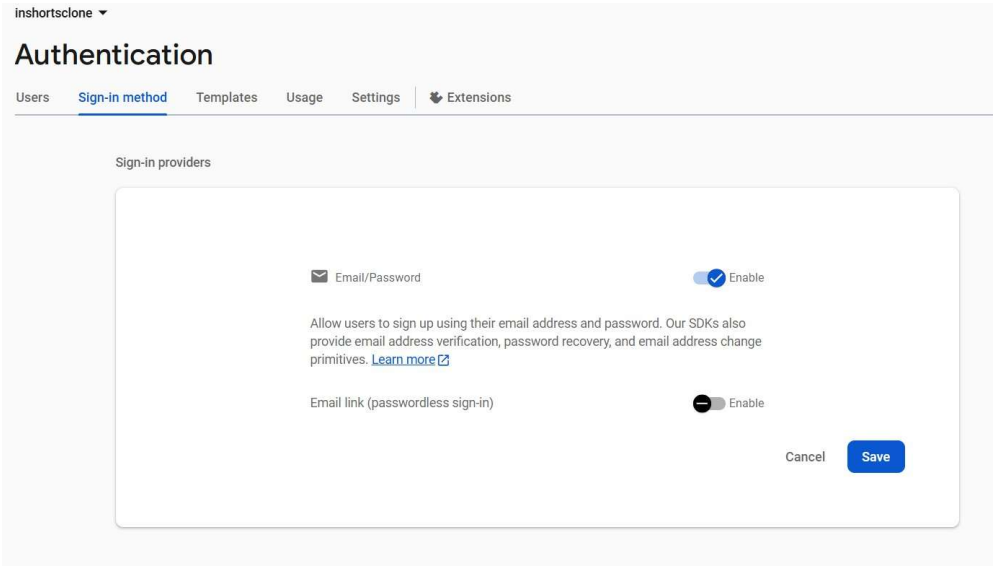
then select the build.gradle.kts (Kotlin DSL) part, and then follow the rest instructions



Generate the firebase_options.dart file, based on the google-services.json file

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

class DefaultFirebaseOptions {
  static FirebaseOptions get currentPlatform {
    return const FirebaseOptions(
      apiKey: "AIzaSyA_VIR7fj4d-b6tNZhW9qJ6GRRx5EXKqs0",
      appId: "1:388272292768:android:33180b2382688b18781ac5",
      messagingSenderId: "388272292768",
      projectId: "inshortsclone-848a9",
      storageBucket: "inshortsclone-848a9.firebasestorage.app",
      androidClientId: "1:388272292768:android:33180b2382688b18781ac5",
    );
  }
}
```

In your part select the sign-in method and enable it.



Code:-

LOGIN:

```
import 'package:flutter/material.dart';
import
'package:firebase_auth/firebase_auth.dart
';
import 'package:get/get.dart';
import 'signup.dart';
import 'forgot.dart';

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

  @override
  State<Login> createState() =>
_LoginState();
}

class _LoginState extends State<Login> {
  TextEditingController email =
TextEditingController();
  TextEditingController password =
TextEditingController();

  signIn() async {
    if (email.text.isEmpty ||
password.text.isEmpty) {

ScaffoldMessenger.of(context).showSnac
kBar(
        SnackBar(
          content: Text("Please fill in all
fields"),
          backgroundColor:
Colors.redAccent,
        ),
      );
      return;
    }

    try {
      await
FirebaseAuth.instance.signInWithEmailAn
dPassword(
        email: email.text,
        password: password.text,
      );

ScaffoldMessenger.of(context).showSnac
kBar(
        SnackBar(
          content: Text("Login Successful!"),
          backgroundColor: Colors.green,
        ),
      );
      Get.offAllNamed('/home'); // Navigate
to home page
    } catch (e) {

ScaffoldMessenger.of(context).showSnac
kBar(
        SnackBar(
          content: Text("Error:
${e.toString()}"),
          backgroundColor: Colors.red,
        ),
      );
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.grey[200],
      body: Center(
        child: Padding(
          padding: const EdgeInsets.all(20.0),
          child: Card(
            elevation: 5,
            shape: RoundedRectangleBorder(
              borderRadius:
BorderRadius.circular(20),
            ),
            child: Padding(
              padding: const
EdgeInsets.all(25.0),
              child: Column(
                mainAxisSize:
MainAxisSize.min,
                crossAxisAlignment:
CrossAxisAlignment.center,
                children: [
                  Text(
                    "Welcome Back!",
                    style: TextStyle(
                      fontSize: 24,
```

```
                fontWeight:
FontWeight.bold,
                  color: Colors.blueAccent,
                ),
              ),
              SizedBox(height: 10),
              Text(
                "Sign in to continue",
                style: TextStyle(fontSize: 16,
color: Colors.grey[700]),
              ),
              SizedBox(height: 20),
              TextField(
                controller: email,
                decoration: InputDecoration(
                  labelText: "Email",
                  prefixIcon: Icon(Icons.email,
color: Colors.blueAccent),
                  border: OutlineInputBorder(
                    borderRadius:
BorderRadius.circular(10),
                  ),
                ),
                keyboardType:
TextInputType.emailAddress,
              ),
              SizedBox(height: 15),
              TextField(
                controller: password,
                decoration: InputDecoration(
                  labelText: "Password",
                  prefixIcon: Icon(Icons.lock,
color: Colors.blueAccent),
                  border: OutlineInputBorder(
                    borderRadius:
BorderRadius.circular(10),
                  ),
                ),
                obscureText: true,
              ),
              SizedBox(height: 20),
              ElevatedButton(
                onPressed: signIn,
                style:
ElevatedButton.styleFrom(
                  backgroundColor:
Colors.blueAccent,
                  foregroundColor:
Colors.white,
                  shape:
RoundedRectangleBorder(
                    borderRadius:
BorderRadius.circular(10),
                  ),
                  padding:
EdgeInsets.symmetric(vertical: 12,
horizontal: 40),
                ),
                child: Text(
                  "Login",
                  style: TextStyle(fontSize:
18),
                ),
              ),
              SizedBox(height: 15),
              TextButton(
                onPressed: () => Get.to(() =>
Forgot()),
                child: Text(
                  "Forgot Password?",
                  style: TextStyle(color:
Colors.blueAccent),
                ),
              ),
              Divider(thickness: 1, height:
30),
              TextButton(
                onPressed: () => Get.to(() =>
Signup()),
                child: Text(
                  "Don't have an account?
Sign Up now",
                  style: TextStyle(color:
Colors.blueAccent),
                ),
              ),
            ],
          ),
        ),
      ),
    ),
  );
 }
```

}

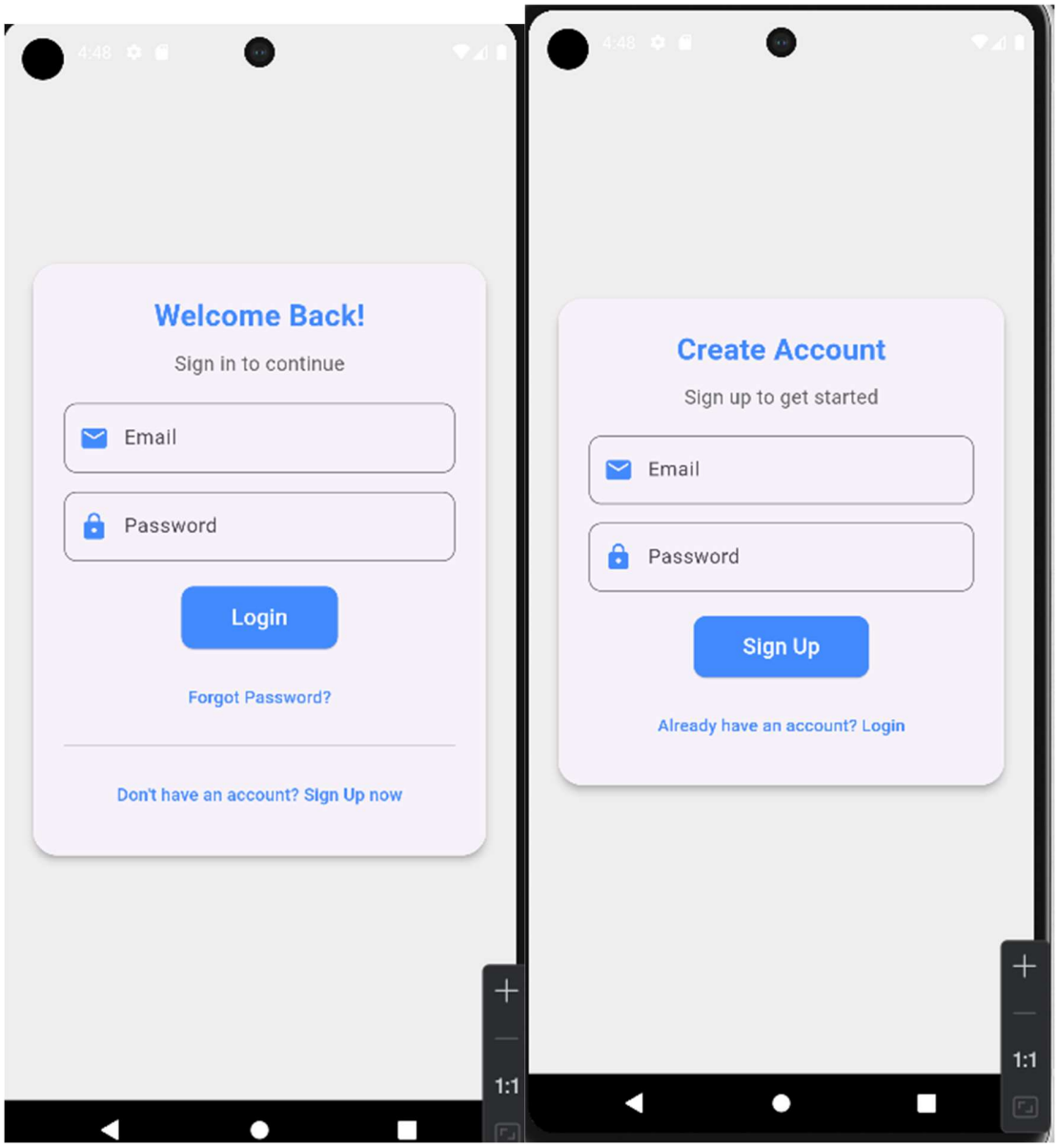
SIGN UP :

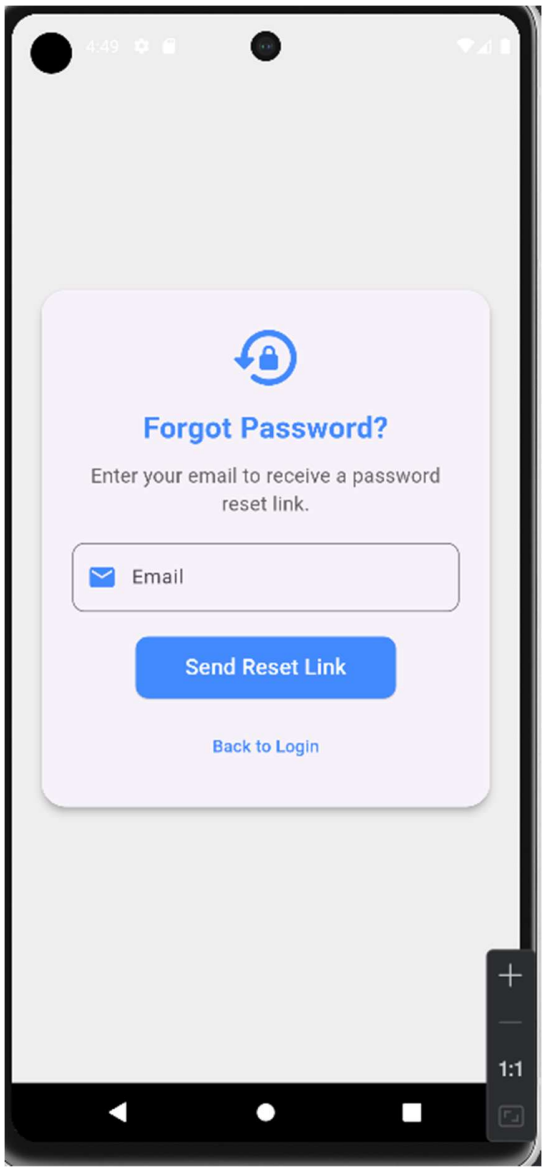
```
import '/wrapper.dart';
import
'package:firebase_auth/firebase_auth.dart
';
import 'package:flutter/material.dart';
import 'package:get/get.dart';

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

  @override
  State<Signup> createState() =>
_SignupState();
}

class _SignupState extends
State<Signup> {
  TextEditingController email =
TextEditingController();
  TextEditingController password =
TextEditingController();

  signup() async {
    if (email.text.isEmpty ||
password.text.isEmpty) {
      Get.snackbar(
        "Error",
        "Please fill in all fields",
        snackPosition:
SnackPosition.BOTTOM,
        backgroundColor: Colors.redAccent,
        colorText: Colors.white,
      );
      return;
    }

    try {
      await
FirebaseAuth.instance.createUserWithEm
ailAndPassword(
        email: email.text.trim(),
        password: password.text.trim(),
      );
      Get.offAll(Wrapper()); // Navigate to
Wrapper after successful signup
      Get.snackbar(
        "Success",
        "Account created successfully!",
        snackPosition:
SnackPosition.BOTTOM,
        backgroundColor: Colors.green,
        colorText: Colors.white,
      );
    } catch (e) {
      Get.snackbar(
        "Signup Failed",
        e.toString(),
        snackPosition:
SnackPosition.BOTTOM,
        backgroundColor: Colors.red,
        colorText: Colors.white,
      );
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.grey[200],
      body: Center(
        child: Padding(
          padding: const EdgeInsets.all(20.0),
          child: Card(
            elevation: 5,
            shape: RoundedRectangleBorder(
              borderRadius:
BorderRadius.circular(20),
            ),
            child: Padding(
              padding: const
EdgeInsets.all(25.0),
              child: Column(
                mainAxisSize:
MainAxisSize.min,
                crossAxisAlignment:
CrossAxisAlignment.center,
                children: [
                  Text(
                    "Create Account",
                    style: TextStyle(
                      fontSize: 24,
                      fontWeight:
```

```
FontWeight.bold,
                    color: Colors.blueAccent,
                  ),
                ),
                SizedBox(height: 10),
                Text(
                  "Sign up to get started",
                  style: TextStyle(fontSize: 16,
color: Colors.grey[700]),
                ),
                SizedBox(height: 20),
                TextField(
                  controller: email,
                  decoration: InputDecoration(
                    labelText: "Email",
                    prefixIcon: Icon(Icons.email,
color: Colors.blueAccent),
                    border: OutlineInputBorder(
                      borderRadius:
BorderRadius.circular(10),
                    ),
                  ),
                  keyboardType:
TextInputType.emailAddress,
                ),
                SizedBox(height: 15),
                TextField(
                  controller: password,
                  decoration: InputDecoration(
                    labelText: "Password",
                    prefixIcon: Icon(Icons.lock,
color: Colors.blueAccent),
                    border: OutlineInputBorder(
                      borderRadius:
BorderRadius.circular(10),
                    ),
                  ),
                  obscureText: true,
                ),
                SizedBox(height: 20),
                ElevatedButton(
                  onPressed: signup,
                  style:
ElevatedButton.styleFrom(
                    backgroundColor:
Colors.blueAccent,
                    foregroundColor:
Colors.white,
                    shape:
RoundedRectangleBorder(
                      borderRadius:
BorderRadius.circular(10),
                    ),
                    padding:
EdgeInsets.symmetric(vertical: 12,
horizontal: 40),
                  ),
                  child: Text(
                    "Sign Up",
                    style: TextStyle(fontSize:
18),
                  ),
                ),
                SizedBox(height: 15),
                TextButton(
                  onPressed: () => Get.back(),
                  child: Text(
                    "Already have an account?
Login",
                    style: TextStyle(color:
Colors.blueAccent),
                  ),
                ),
              ],
            ),
          ),
        ),
      ),
    );
  }
}
```

OUTPUT :

```
[3]: Edge (edge)
Please choose one (or "q" to quit): 2
Launching lib\main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...            20.9s
This app is linked to the debug service: ws://127.0.0.1:56889/EEaa4qKHJvU=/ws
Debug service listening on ws://127.0.0.1:56889/EEaa4qKHJvU=/ws

  To hot restart changes while running, press "r" or "R".
For a more detailed help message, press "h". To quit, press "q".

A Dart VM Service on Chrome is available at: http://127.0.0.1:56889/EEaa4qKHJvU=
The Flutter DevTools debugger and profiler on Chrome is available at: http://127.0.0.1:9101?uri=http://127.0.0.1:56889/EEaa4qKHJvU=
_Firebase initialized successfully!
```




4:49
Forgot Password?
Enter your email to receive a password reset link.
Email
Send Reset Link
Back to Login
1:1

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.	47
Name	Vedant Sanap
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	

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.	47
Name	Vedant Sanap
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	

Experiment No. 7

Title: To write meta data of your Ecommerce PWA

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

Theory:

Regular Web App

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

Progressive Web App

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

Difference between PWAs vs. Regular Web Apps:

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

1. Native Experience

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

2. Ease of Access

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

3. Faster Services

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Linkable — Easily shared via URL without complex installations.

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

Weaknesses:

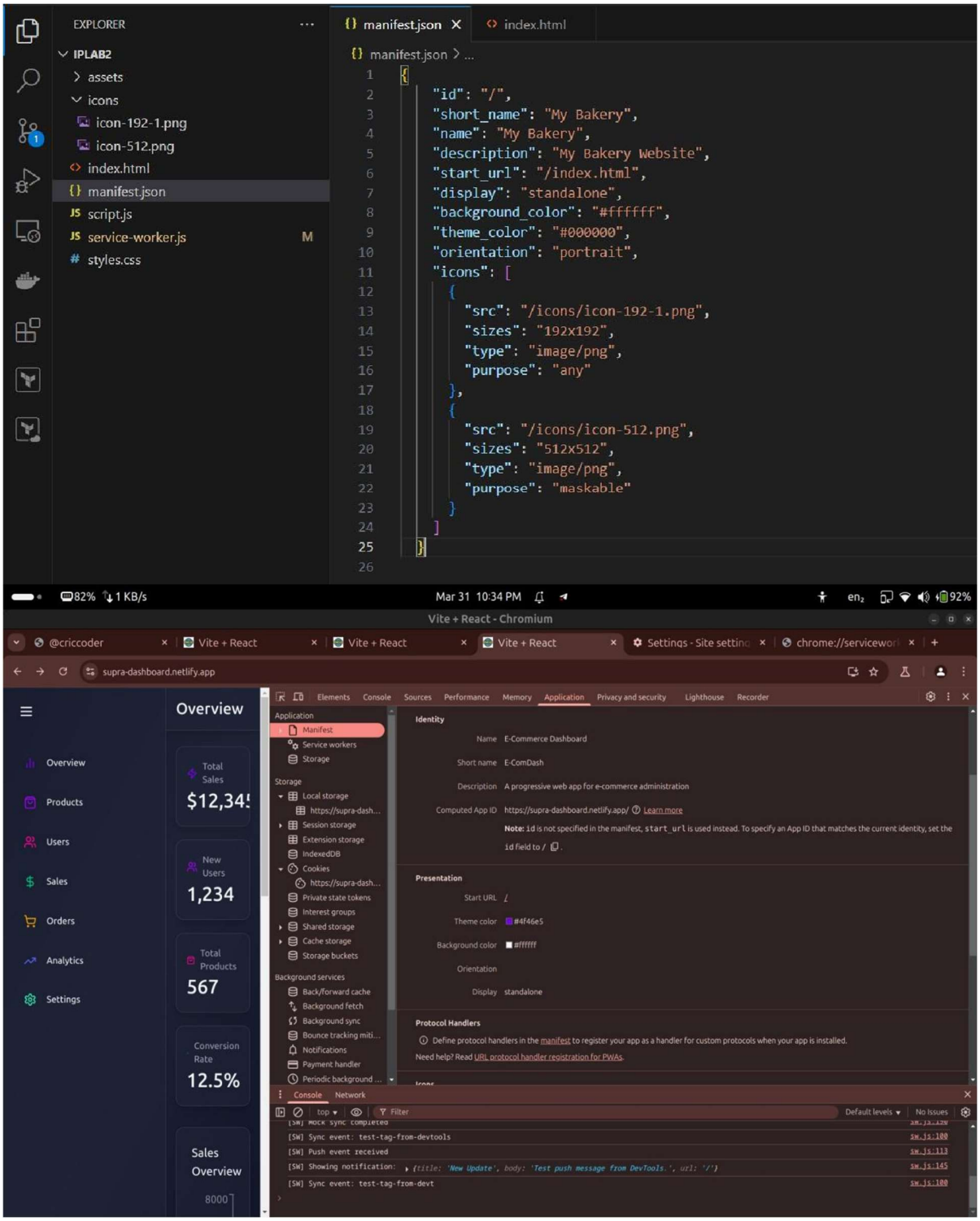
- IOS support from version 11.3 onwards
- Greater use of the device battery
- Not all devices support the full range of PWA features (same speech for iOS and Android operating systems)
- It is not possible to establish a strong re-engagement for iOS users (URL scheme, standard web notifications)
- Support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

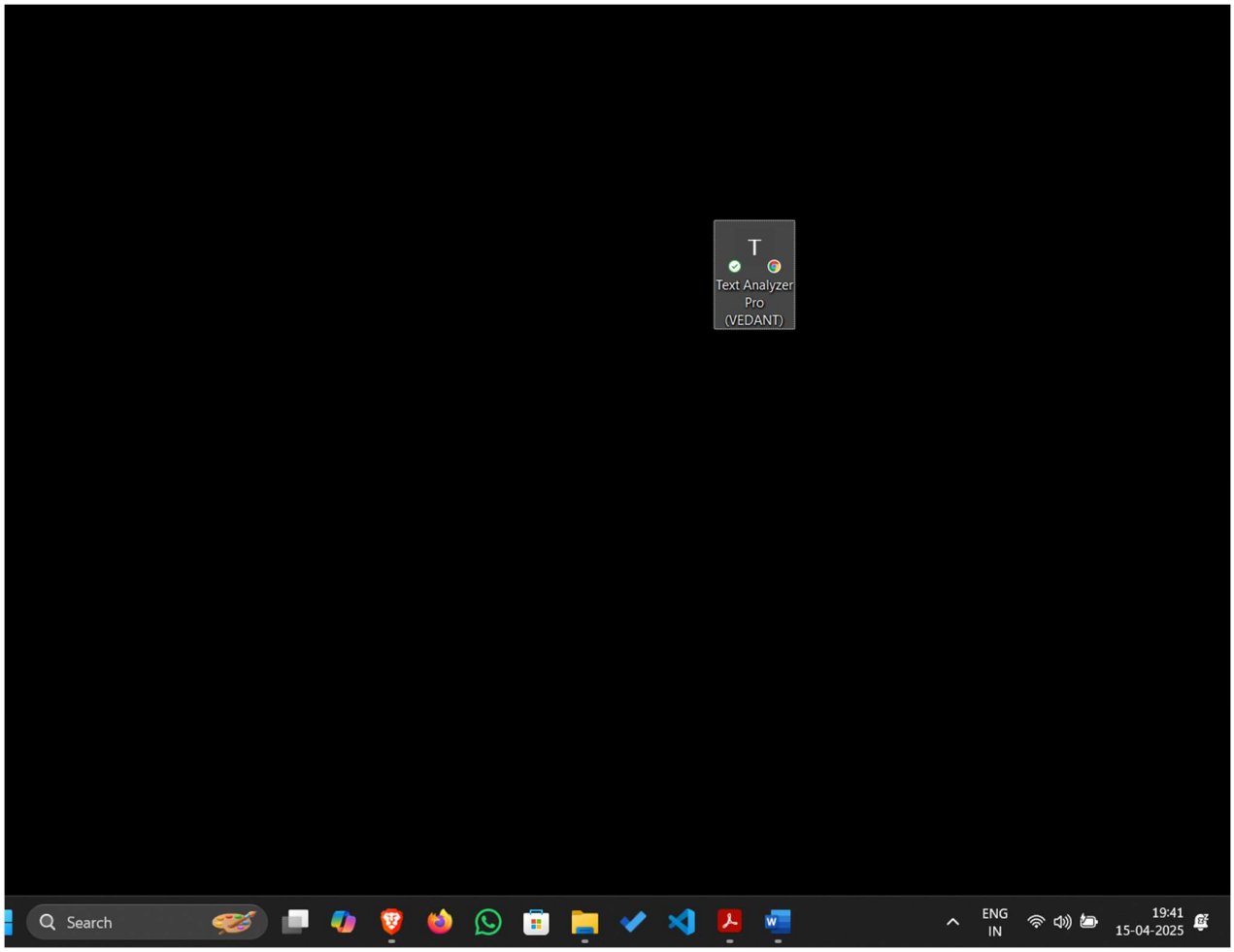
Code:

manifest.json:

```
{
    "id": "/",
    "short_name": "My Bakery",
    "name": "My Bakery",
    "description": "My Bakery Website",
    "start_url": "/index.html",
    "display": "standalone",
    "background_color": "#ffffff",
    "theme_color": "#000000",
    "orientation": "portrait",
    "icons": [
      {
        "src": "/icons/icon-192-1.png",
        "sizes": "192x192",
        "type": "image/png",
        "purpose": "any"
      },
      {
        "src": "/icons/icon-512.png",
        "sizes": "512x512",
        "type": "image/png",
        "purpose": "maskable"
      }
    ]
  }
```

Add the link tag to link to the manifest.json file


EXPLORER
IPLAB2
assets
icons
icon-192-1.png
icon-512.png
index.html
manifest.json
script.js
service-worker.js M
styles.css
manifest.json
index.html
{
"id": "/",
"short_name": "My Bakery",
"name": "My Bakery",
"description": "My Bakery Website",
"start_url": "/index.html",
"display": "standalone",
"background_color": "#ffffff",
"theme_color": "#000000",
"orientation": "portrait",
"icons": [
{
"src": "/icons/icon-192-1.png",
"sizes": "192x192",
"type": "image/png",
"purpose": "any"
},
{
"src": "/icons/icon-512.png",
"sizes": "512x512",
"type": "image/png",
"purpose": "maskable"
}
]
}
82% 1 KB/s
Mar 31 10:34 PM
en2
92%
Vite + React - Chromium
@criccoder
Vite + React
Vite + React
Vite + React
Settings - Site setting
chrome://servicework
supra-dashboard.netlify.app
Overview
Overview
Products
Users
Sales
Orders
Analytics
Settings
Total Sales
$12,345
New Users
1,234
Total Products
567
Conversion Rate
12.5%
Sales Overview
8000
Elements
Console
Sources
Performance
Memory
Application
Privacy and security
Lighthouse
Recorder
Application
Manifest
Service workers
Storage
Storage
Local storage
https://supra-dash...
Session storage
Extension storage
IndexedDB
Cookies
https://supra-dash...
Private state tokens
Interest groups
Shared storage
Cache storage
Storage buckets
Background services
Back/forward cache
Background fetch
Background sync
Bounce tracking miti...
Notifications
Payment handler
Periodic background ...
Identity
Name E-Commerce Dashboard
Short name E-ComDash
Description A progressive web app for e-commerce administration
Computed App ID https://supra-dashboard.netlify.app/ Learn more
Note: id is not specified in the manifest, start_url is used instead. To specify an App ID that matches the current identity, set the id field to /.
Presentation
Start URL /
Theme color #4f46e5
Background color #ffffff
Orientation
Display standalone
Protocol Handlers
Define protocol handlers in the manifest to register your app as a handler for custom protocols when your app is installed.
Need help? Read URL protocol handler registration for PWAs.
Console
Network
top
Filter
Default levels
No Issues
[SW] Sync event: test-tag-from-devtools sw.js:100
[SW] Push event received sw.js:113
[SW] Showing notification: {title: 'New Update', body: 'Test push message from DevTools.', url: '/'} sw.js:145
[SW] Sync event: test-tag-from-devt sw.js:100


T
Text Analyzer Pro (VEDANT)
Search
ENG
IN
19:41
15-04-2025

Conclusion:

Hence, we learnt how to write a metadata of our website PWA in a Web App Manifest File to enable add to homescreen feature.

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.	47
Name	Vedant Sanap
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

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.	47
Name	Vedant Sanap
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

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

Things to note about Service Worker:

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

What can we do with Service Workers?

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

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

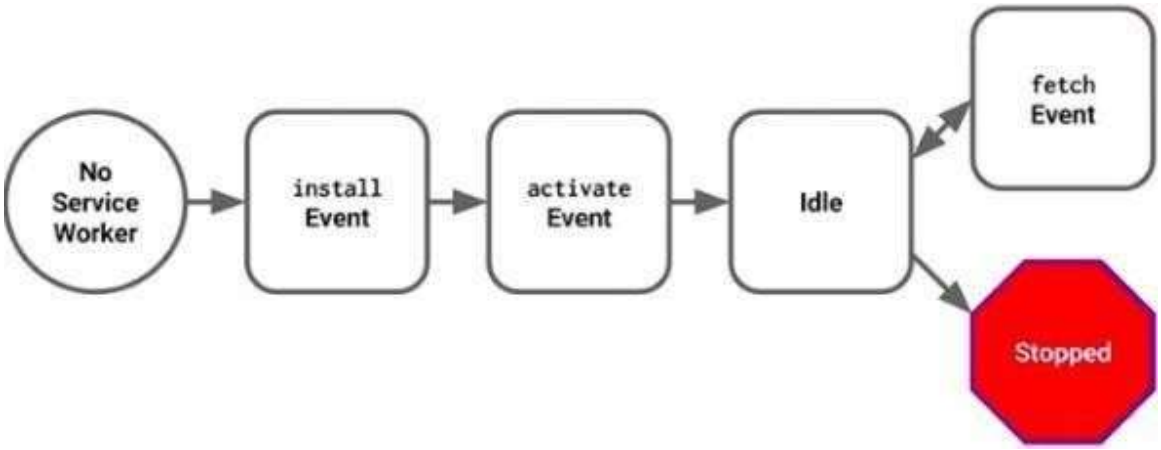
- You can manage **Push Notifications**
  You can manage push notifications with Service Worker and show any information message to the user.
- You can **Continue**
  Although Internet connection is broken, you can start any process with Background Sync of Service Worker.

What can’t we do with Service Workers?

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

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

Service Worker Cycle



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

- Registration
- Installation
- Activation

Registration

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

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

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

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

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

example: main.js

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

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

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

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

Installation

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

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

service-worker.js

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

Activation

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

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

service-worker.js

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

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

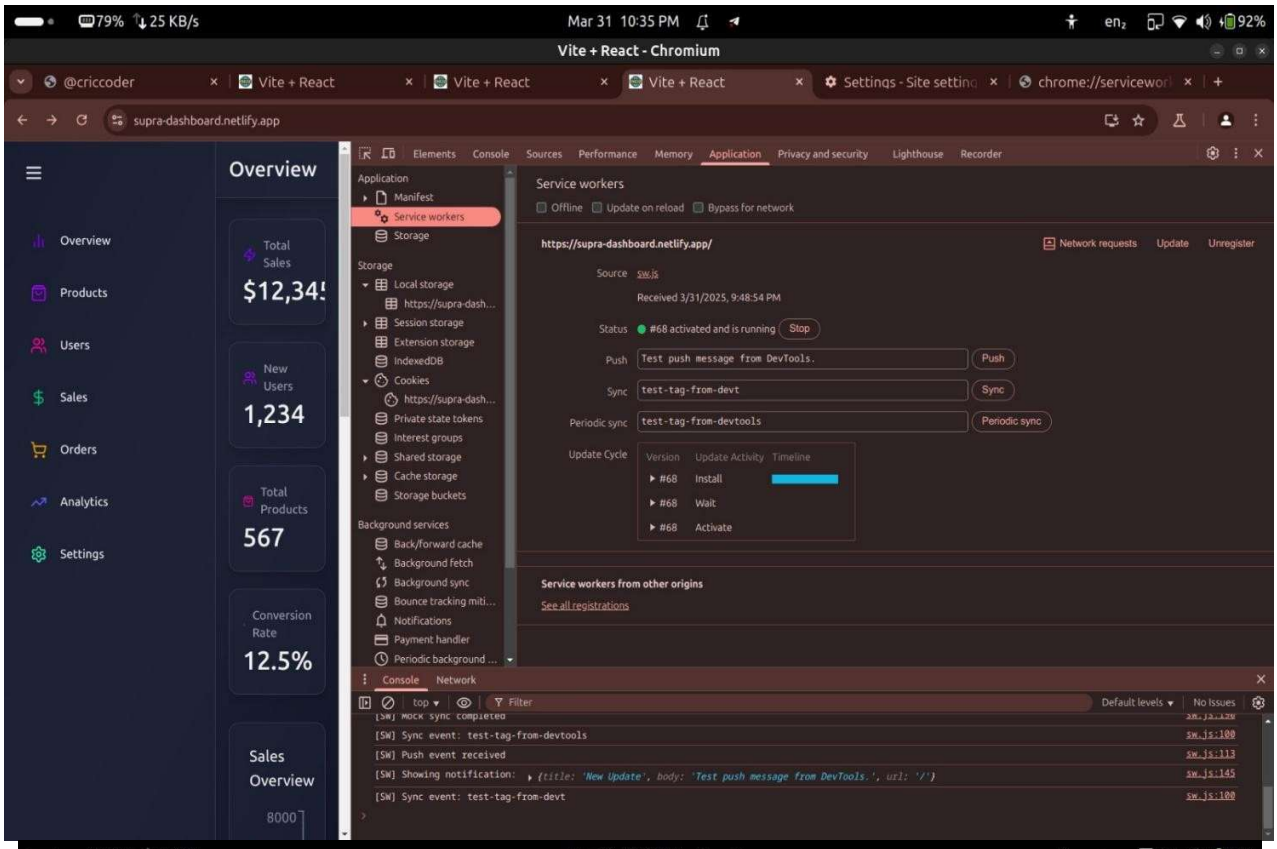
Code:

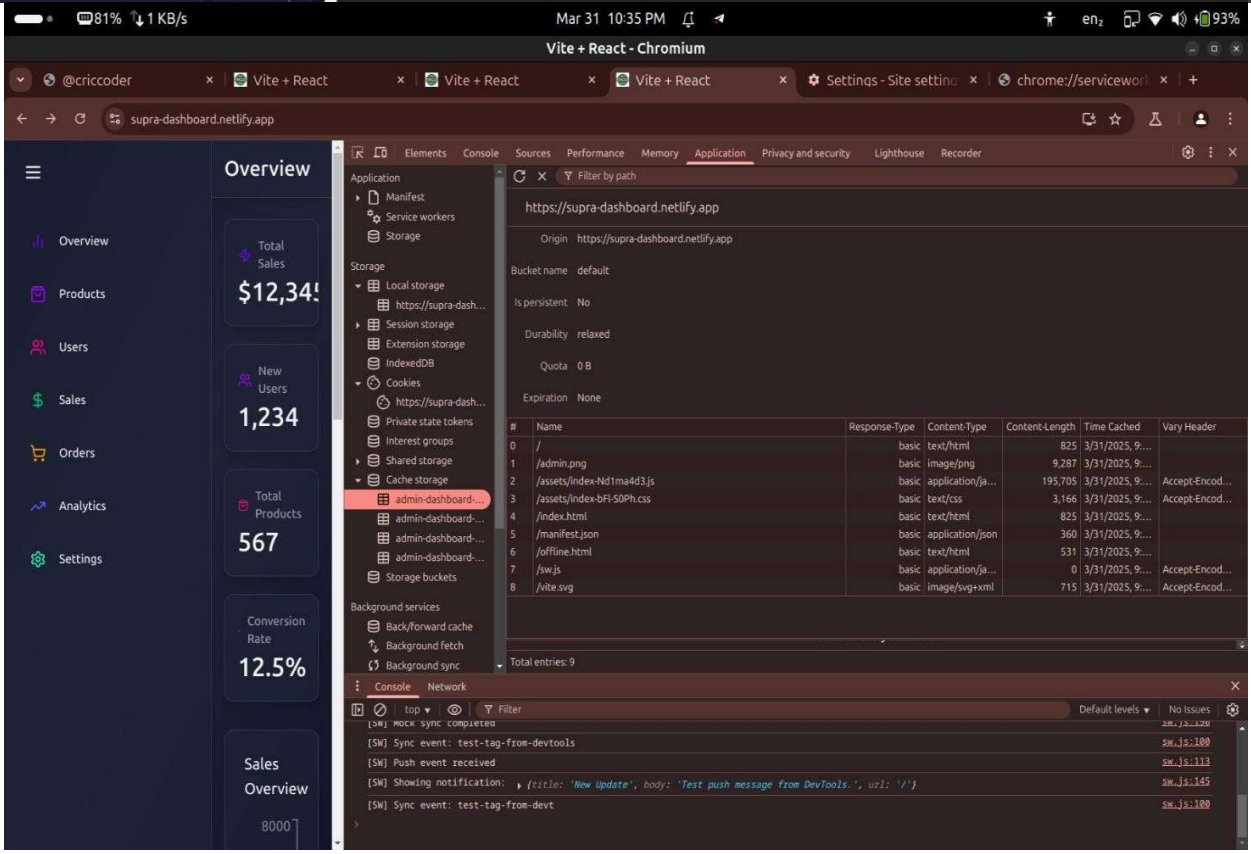
```
<script>
      if ('serviceWorker' in navigator) {
        window.addEventListener('load', () => {
        navigator.serviceWorker.register('/service-worker.js')
            .then((registration) => {
              console.log('Service Worker registered with scope: ', registration.scope);
       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

service-worker.js

```
const CACHE_NAME = 'my-pwa-cache-v1';
const urlsToCache = [
  '/',
  '/index.html',
  '/styles.css',
  '/icons/icon-192-1.png',
  '/icons/icon-512.png',
  '/assets/brownie.png',
  '/assets/cakes.png',
  '/assets/cookies.png',
  '/assets/donuts.png',
  '/assets/logo.png',
  '/assets/muffins.png',
  '/assets/pastry.png',
];

// Install service worker
self.addEventListener('install', (event) => {
event.waitUntil(
    caches.open(CACHE_NAME).then((cache) => {
      console.log('Opened cache');
      return cache.addAll(urlsToCache);
```

```
    })
  );
});

// Activate service worker
self.addEventListener('activate', (event) => {
const cacheWhitelist = [CACHE_NAME];
event.waitUntil(
    caches.keys().then((cacheNames) => {
      return Promise.all(
      cacheNames.map((cacheName) => {
          if (!cacheWhitelist.includes(cacheName)) {
            return caches.delete(cacheName);
          }
        })
      );
    })
  );
});

// Fetch content from the cache or network
self.addEventListener('fetch', (event) => {
event.respondWith(
    caches.match(event.request).then((cachedResponse) => {
      return cachedResponse || fetch(event.request);
    })
  );
});
```


79% 25 KB/s
Mar 31 10:35 PM
en2
92%
Vite + React - Chromium
@criccoder
Vite + React
Vite + React
Vite + React
Settings - Site setting
chrome://serviceworl
supra-dashboard.netlify.app
Overview
Overview
Products
Users
Sales
Orders
Analytics
Settings
Total Sales
$12,345
New Users
1,234
Total Products
567
Conversion Rate
12.5%
Sales Overview
8000
Elements
Console
Sources
Performance
Memory
Application
Privacy and security
Lighthouse
Recorder
Application
Manifest
Service workers
Storage
Storage
Local storage
Session storage
Extension storage
IndexedDB
Cookies
Private state tokens
Interest groups
Shared storage
Cache storage
Storage buckets
Background services
Back/forward cache
Background fetch
Background sync
Bounce tracking miti...
Notifications
Payment handler
Periodic background ...
Service workers
Offline
Update on reload
Bypass for network
https://supra-dashboard.netlify.app/
Network requests
Update
Unregister
Source sw.js
Received 3/31/2025, 9:48:54 PM
Status #68 activated and is running
Stop
Push
Test push message from DevTools.
Push
Sync
test-tag-from-devt
Sync
Periodic sync
test-tag-from-devtools
Periodic sync
Update Cycle
Version
Update Activity
Timeline
#68 Install
#68 Wait
#68 Activate
Service workers from other origins
See all registrations
Console
Network
top
Filter
Default levels
No Issues
[SW] Mock sync completed
[SW] Sync event: test-tag-from-devtools
sw.js:100
[SW] Push event received
sw.js:113
[SW] Showing notification: {title: 'New Update', body: 'Test push message from DevTools.', url: '/'}
sw.js:145
[SW] Sync event: test-tag-from-devt
sw.js:100


81% 1 KB/s
Mar 31 10:35 PM
en2
93%
Vite + React - Chromium
@criccoder
Vite + React
Vite + React
Vite + React
Settings - Site setting
chrome://serviceworl
supra-dashboard.netlify.app
Overview
Overview
Products
Users
Sales
Orders
Analytics
Settings
Total Sales
$12,345
New Users
1,234
Total Products
567
Conversion Rate
12.5%
Sales Overview
8000
Elements
Console
Sources
Performance
Memory
Application
Privacy and security
Lighthouse
Recorder
Filter by path
Application
Manifest
Service workers
Storage
Storage
Local storage
Session storage
Extension storage
IndexedDB
Cookies
Private state tokens
Interest groups
Shared storage
Cache storage
admin-dashboard-...
Storage buckets
Background services
Back/forward cache
Background fetch
Background sync
https://supra-dashboard.netlify.app
Origin https://supra-dashboard.netlify.app
Bucket name default
Is persistent No
Durability relaxed
Quota 0 B
Expiration None
Name Response-Type Content-Type Content-Length Time Cached Vary Header
0 / basic text/html 825 3/31/2025, 9:...
1 /admin.png basic image/png 9,287 3/31/2025, 9:...
2 /assets/index-Nd1ma4d3.js basic application/ja... 195,705 3/31/2025, 9:... Accept-Encod...
3 /assets/index-bFl-50Ph.css basic text/css 3,166 3/31/2025, 9:... Accept-Encod...
4 /index.html basic text/html 825 3/31/2025, 9:...
5 /manifest.json basic application/json 360 3/31/2025, 9:...
6 /offline.html basic text/html 531 3/31/2025, 9:...
7 /sw.js basic application/ja... 0 3/31/2025, 9:... Accept-Encod...
8 /vite.svg basic image/svg+xml 715 3/31/2025, 9:... Accept-Encod...
Total entries: 9
Console
Network
top
Filter
Default levels
No Issues
[SW] Mock sync completed
[SW] Sync event: test-tag-from-devtools
sw.js:100
[SW] Push event received
sw.js:113
[SW] Showing notification: {title: 'New Update', body: 'Test push message from DevTools.', url: '/'}
sw.js:145
[SW] Sync event: test-tag-from-devt
sw.js:100

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.	47
Name	Vedant Sanap
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

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.	47
Name	Vedant Sanap
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

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”.

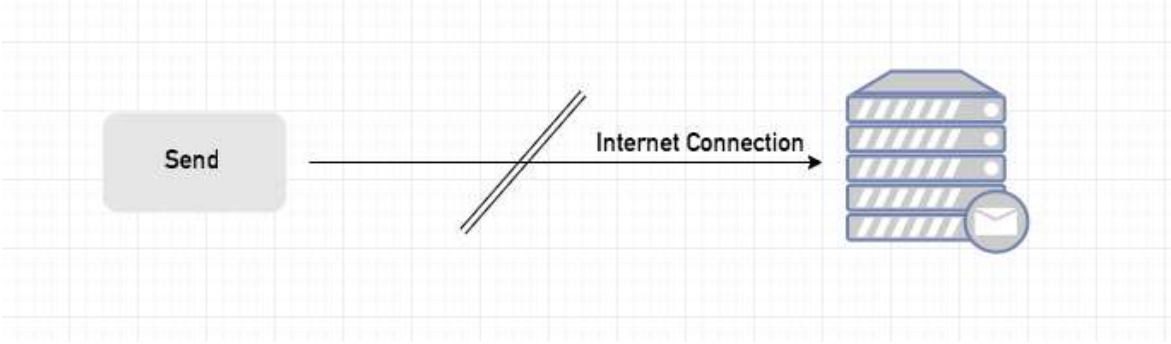
- **CacheFirst** - In this function, if the received request has cached before, the cached response is returned to the page. But if not, a new response requested from the network.
- **NetworkFirst** - In this function, firstly we can try getting an updated response from the network, if this process completed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

Sync Event

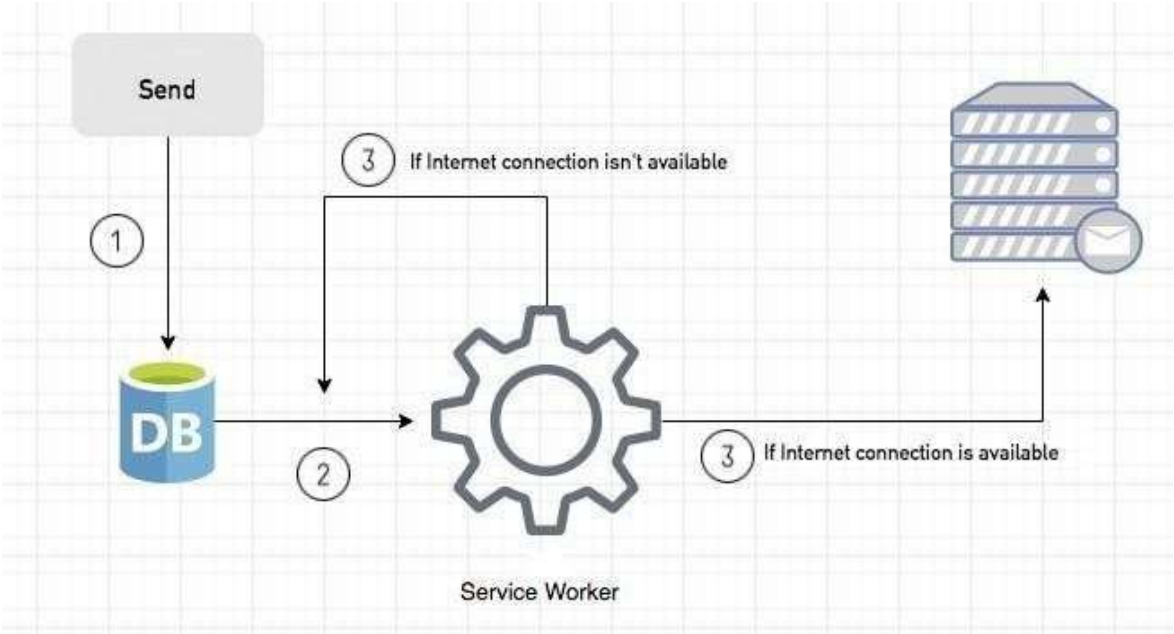
Background Sync is a Web API that is used to delay a process until the Internet connection is stable. We can adapt this definition to the real world; there is an e-mail client application that works on the browser and we want to send an email with this tool. Internet connection is broken while we are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

Here is a job for the Background Sync.

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



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



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

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

Event Listener for Background Sync Registration

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

Event Listener for sw.js

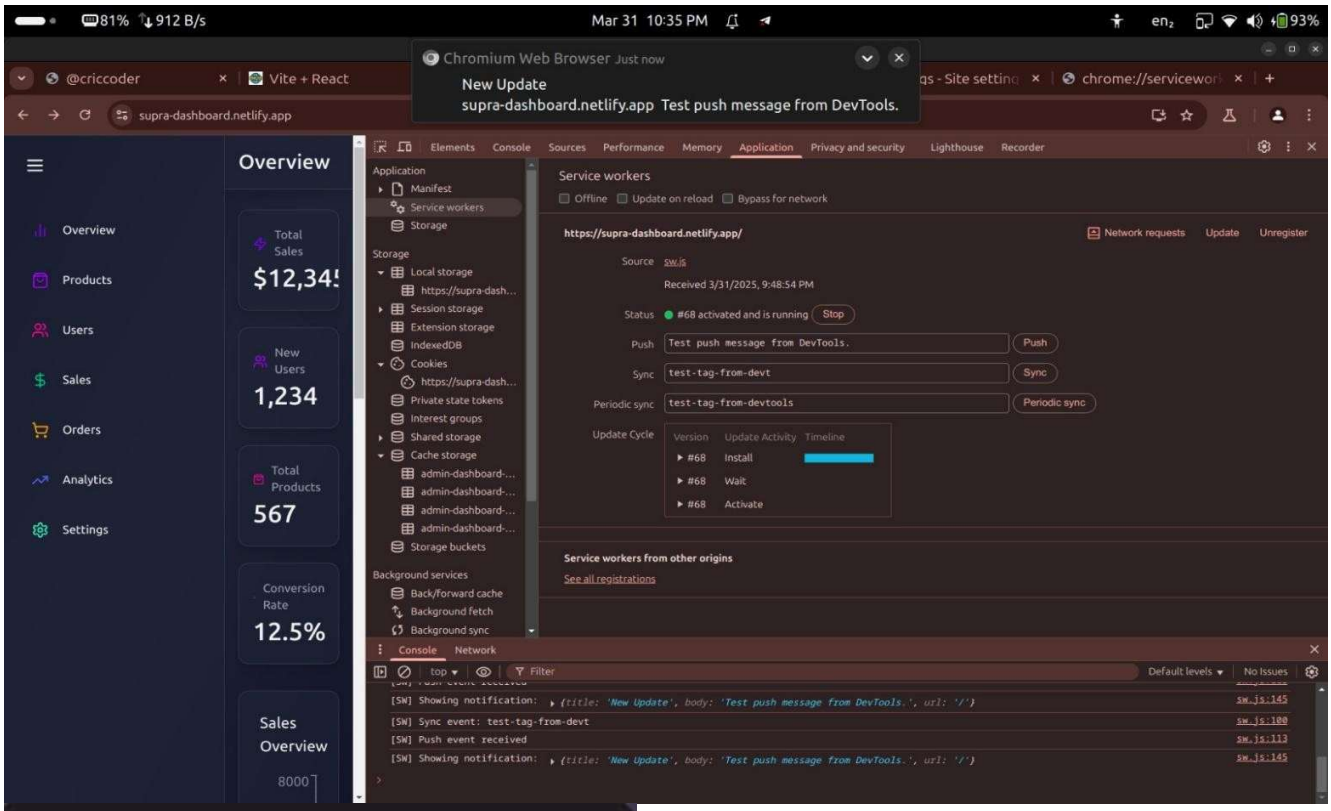
```
self.addEventListener('sync', event => {
    if (event.tag == 'helloSync') {
        console.log("helloSync [sw.js]");
    }
});
```

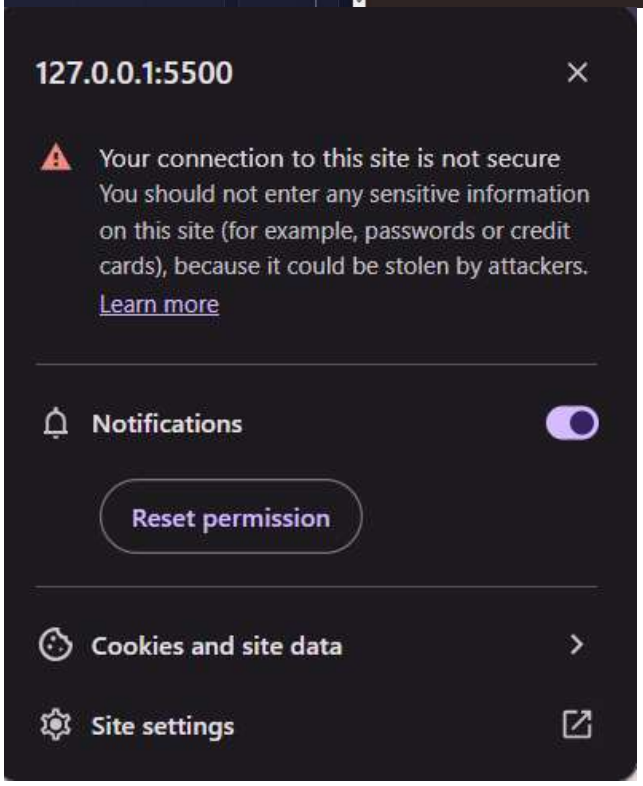
Push Event

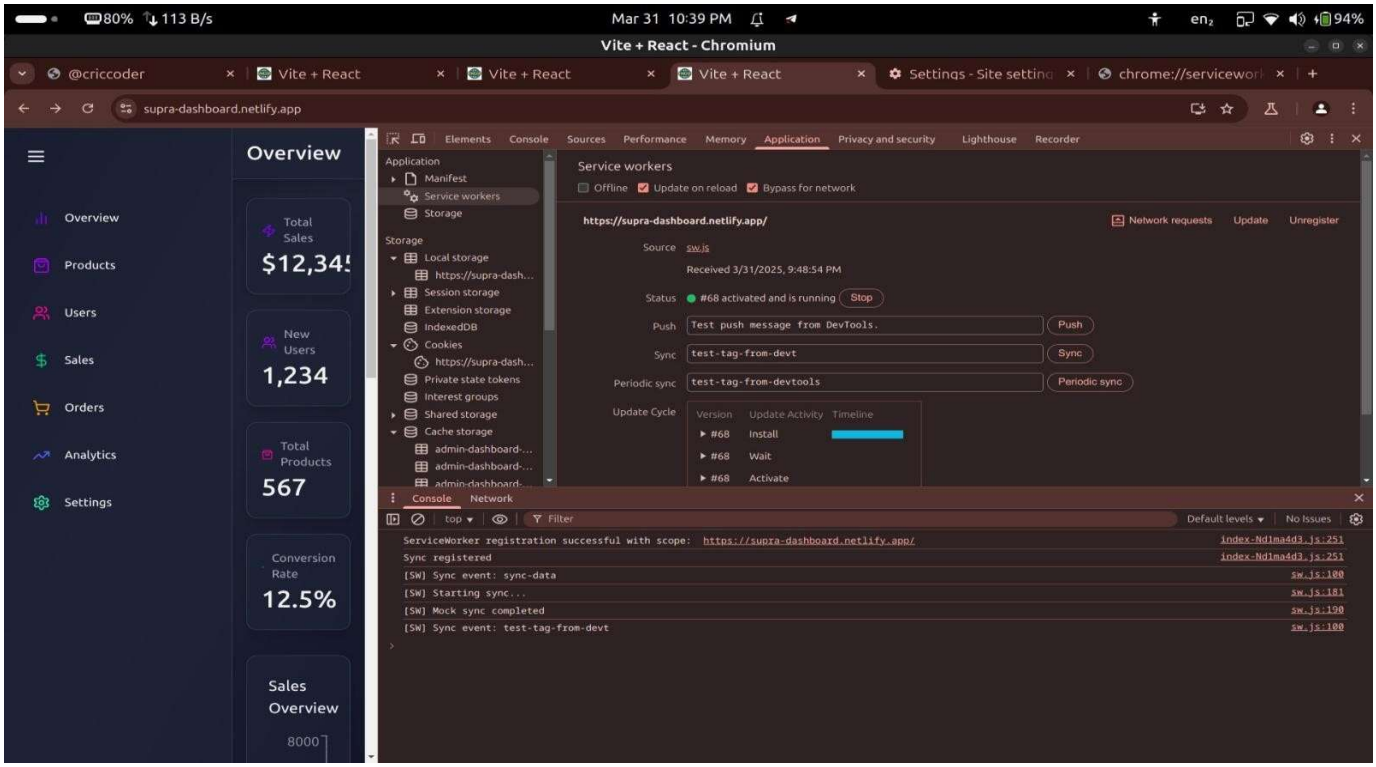
This is the event that handles push notifications that are received from the server. You can apply any method with received data.

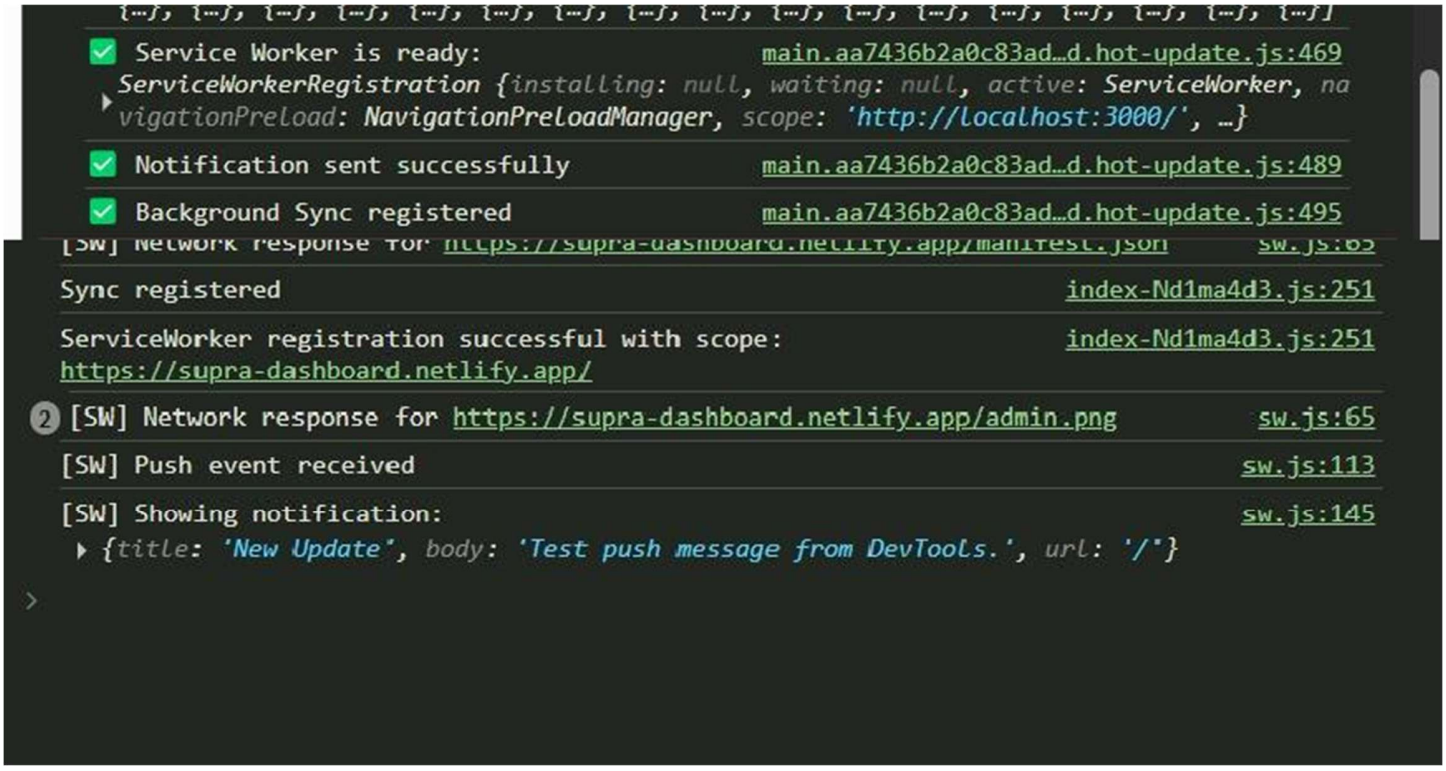
We can check in the following example.
“Notification.requestPermission();” is the necessary line to show notification to the user. If you don’t want to show any notification, you don’t need this line.

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


Chromium Web Browser Just now
New Update
supra-dashboard.netlify.app Test push message from DevTools.
supra-dashboard.netlify.app
Overview
Total Sales
$12,345
New Users
1,234
Total Products
567
Conversion Rate
12.5%
Sales Overview
Elements Console Sources Performance Memory Application Privacy and security Lighthouse Recorder
Service workers
Offline Update on reload Bypass for network
https://supra-dashboard.netlify.app/
Network requests Update Unregister
Source sw.js
Received 3/31/2025, 9:48:54 PM
Status #68 activated and is running Stop
Push Test push message from DevTools. Push
Sync test-tag-from-devt Sync
Periodic sync test-tag-from-devtools Periodic sync
Update Cycle
Version Update Activity Timeline
#68 Install
#68 Wait
#68 Activate
Service workers from other origins
See all registrations
Console Network
[SW] Showing notification: {title: 'New Update', body: 'Test push message from DevTools.', url: '/'}
[SW] Sync event: test-tag-from-devt
[SW] Push event received
[SW] Showing notification: {title: 'New Update', body: 'Test push message from DevTools.', url: '/'}


127.0.0.1:5500
Your connection to this site is not secure
You should not enter any sensitive information on this site (for example, passwords or credit cards), because it could be stolen by attackers.
Learn more
Notifications
Reset permission
Cookies and site data
Site settings


Vite + React - Chromium
supra-dashboard.netlify.app
Overview
Total Sales
$12,345
New Users
1,234
Total Products
567
Conversion Rate
12.5%
Sales Overview
Service workers
Offline
Update on reload
Bypass for network
https://supra-dashboard.netlify.app/
Network requests
Update
Unregister
Source sw.js
Received 3/31/2025, 9:48:54 PM
Status #68 activated and is running
Stop
Push
Test push message from DevTools.
Sync
test-tag-from-devt
Periodic sync
test-tag-from-devtools
Update Cycle
ServiceWorker registration successful with scope: https://supra-dashboard.netlify.app/
Sync registered
[SW] Sync event: sync-data
[SW] Starting sync...
[SW] Mock sync completed
[SW] Sync event: test-tag-from-devt


Service Worker is ready:
main.aa7436b2a0c83ad…d.hot-update.js:469
ServiceWorkerRegistration {installing: null, waiting: null, active: ServiceWorker, navigationPreload: NavigationPreloadManager, scope: 'http://localhost:3000/', …}
Notification sent successfully
main.aa7436b2a0c83ad…d.hot-update.js:489
Background Sync registered
main.aa7436b2a0c83ad…d.hot-update.js:495
Sync registered
index-Nd1ma4d3.js:251
ServiceWorker registration successful with scope:
https://supra-dashboard.netlify.app/
index-Nd1ma4d3.js:251
[SW] Network response for https://supra-dashboard.netlify.app/admin.png
sw.js:65
[SW] Push event received
sw.js:113
[SW] Showing notification:
sw.js:145
{title: 'New Update', body: 'Test push message from DevTools.', url: '/'}

MAD & PWA Lab
Journal

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	47
Name	Vedant Sanap
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

MAD & PWA Lab
Journal

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	47
Name	Vedant Sanap
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

Experiment No. 10

Aim:

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

Theory:

GitHub Pages

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

GitHub Pages provides the following key features:

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

Reasons for favoring this over Firebase:

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

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

Pros

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

Cons

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

Firebase

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

Some of the features offered by Firebase are:

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

Reasons for favoring over GitHub Pages:

1. Realtime backend made easy
2. Fast and responsive

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

Pros

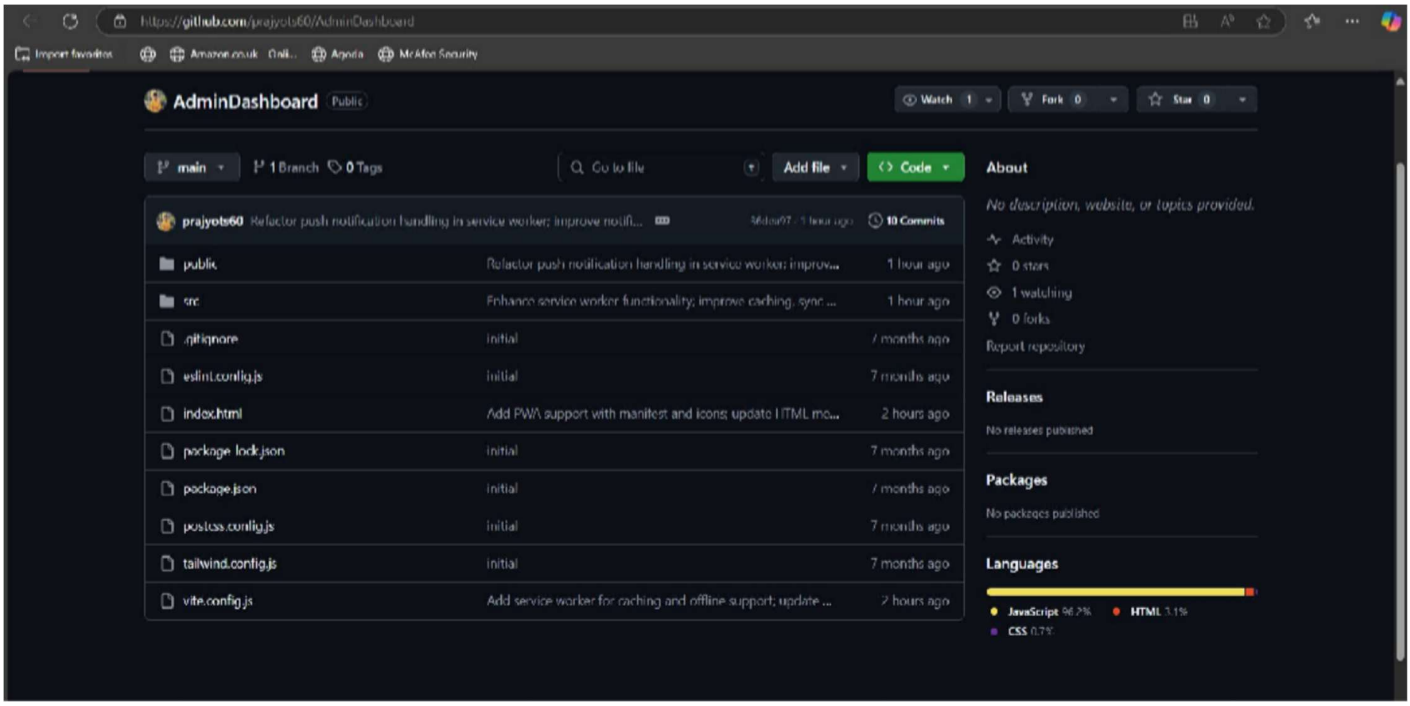
1. Hosted by Google. Enough said.
2. Authentication, Cloud Messaging, and a whole lot of other handy services will be available to you.
3. A real-time database will be available to you, which can store 1 GB of data.
4. You'll also have access to a blob store, which can store another 1 GB of data.
5.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

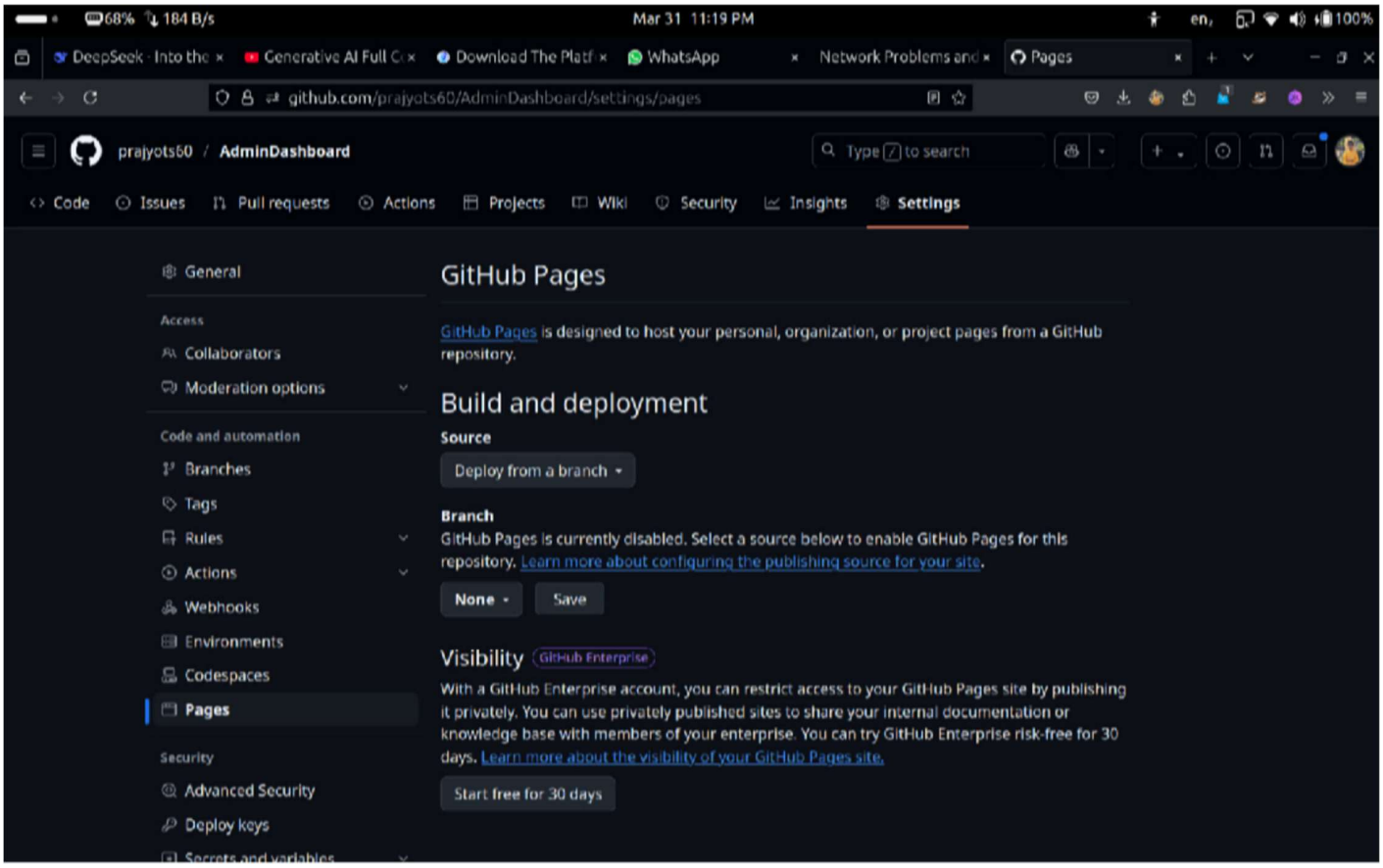
Cons

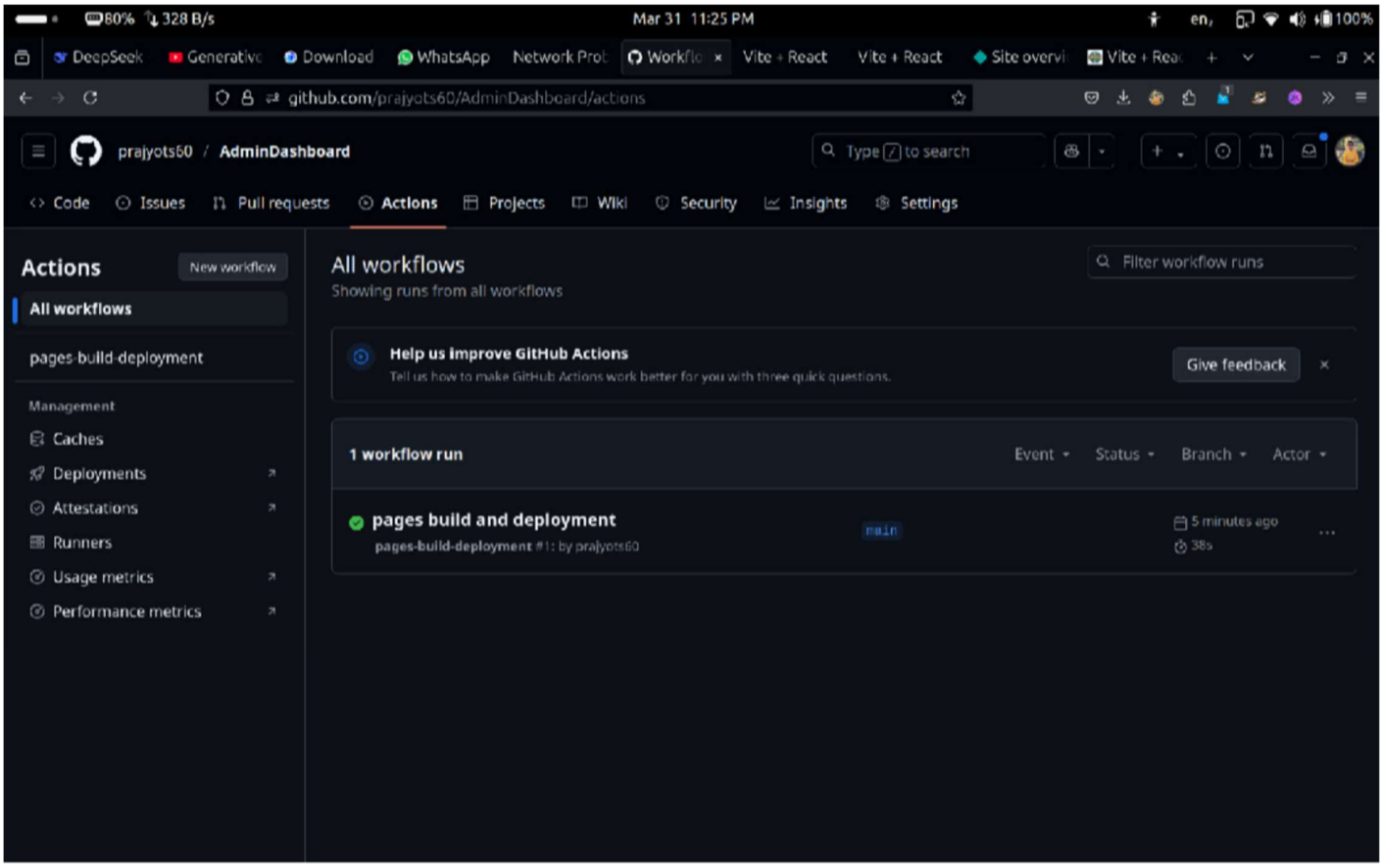
1. Only 10 GB of data transfer is allowed per month. But this is not really a big problem, if you use a CDN or AMP.
2. Command-line interface only.
3. No in-built support for any static site generator.

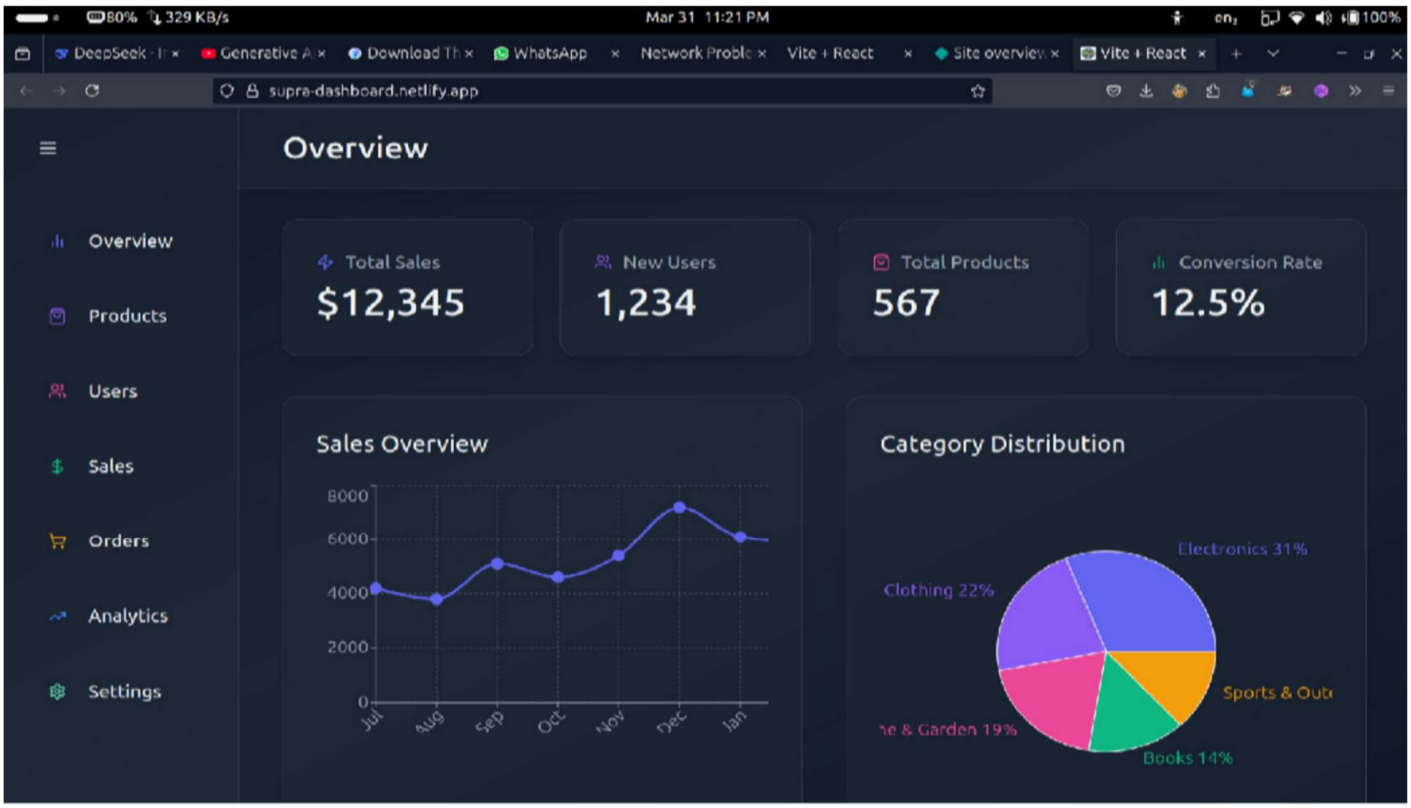
Link to our GitHub repository: https://github.com/VedantSanap0110/text_analyzer

Github Screenshot:









Deployed Link: https://github.com/VedantSanap0110/text_analyzer

MAD & PWA Lab
Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	47
Name	Vedant Sanap
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	

MAD & PWA Lab
Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	47
Name	Vedant Sanap
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	

Experiment No. 11

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

Theory :

Google Lighthouse :

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

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

Key Features and Audit Metrics

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

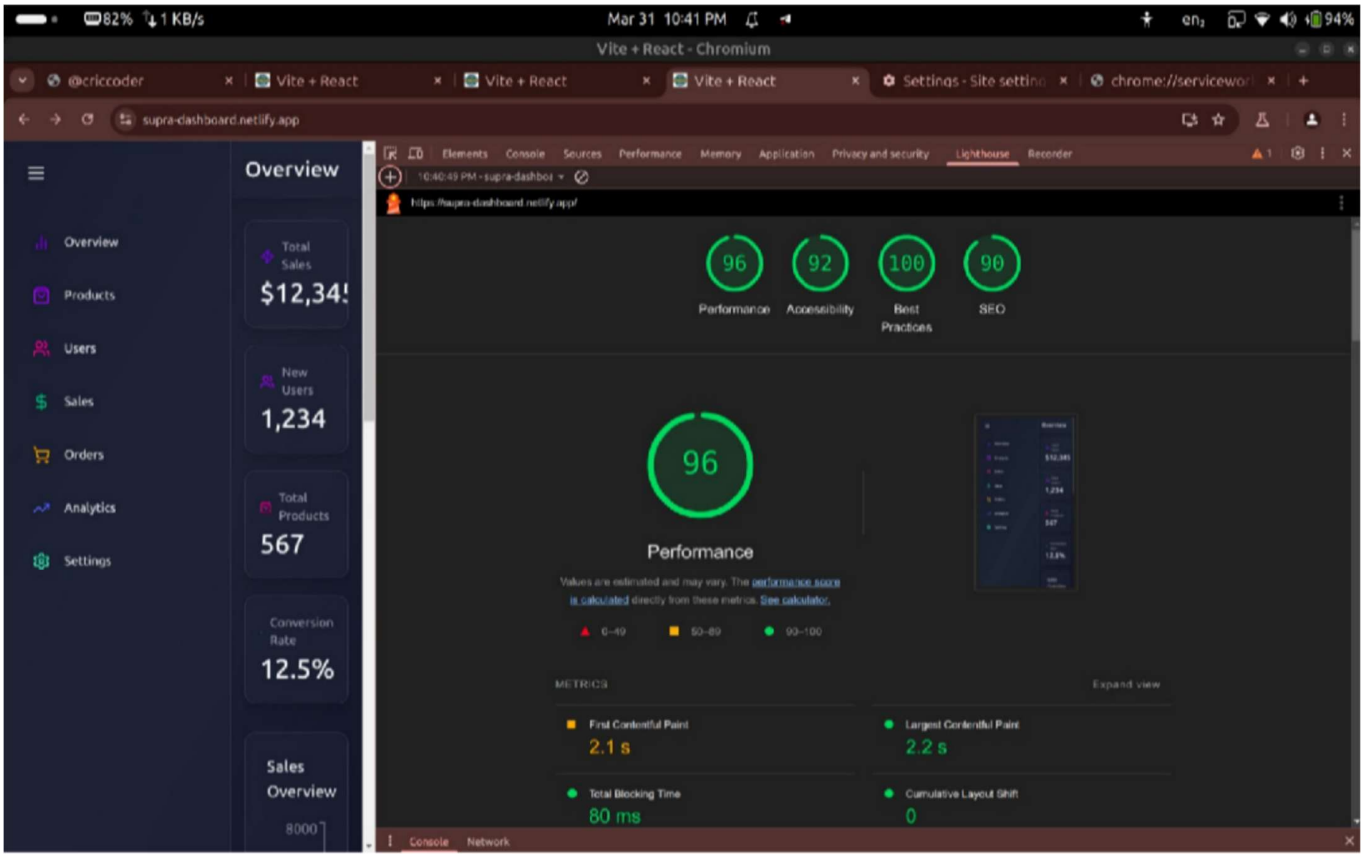
1. **Performance:** This score is an aggregation of how the page fared in aspects such as (but not limited to) loading speed, time taken for loading for basic frame(s), displaying meaningful content to the user, etc. To a layman, this score is indicative of how decently the site performs, with a score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

etc.). Unlike the other metrics though, Accessibility metrics score on a pass/fail basis i.e. if all possible elements of the page are not screen-reader friendly (HTML5 introduced features that would make pages easy to interpret for screen readers used by visually challenged people like tag names, tags such as <section>, <article>, etc.), you get a 0 on that score. The aggregate of these scores is your Accessibility metric score.

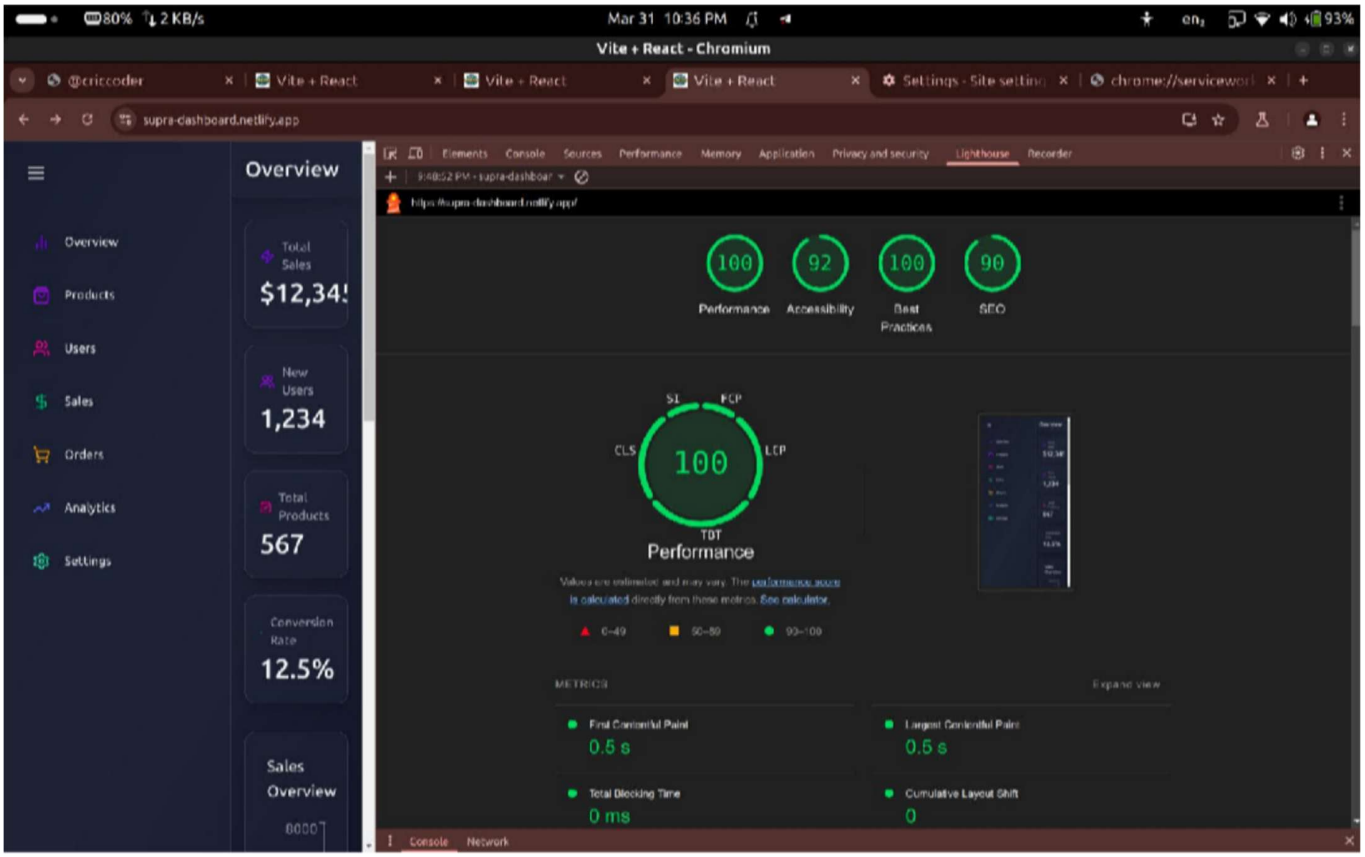
4. **Best Practices:** As any developer would know, there are a number of practices that have been deemed 'best' based on empirical data. This metric is an aggregation of many such points, including but not limited to:Use of HTTPS
   Avoiding the use of deprecated code elements like tags, directives, libraries, etc.
   Password input with paste-into disabled
   Geo-Location and cookie usage alerts on load, etc.

Screenshots:

Mobile: initial



Desktop



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

MAD & PWA Lab
Journal

Experiment No.	Assignment-1
Assignment 1 Questions	1. Flutter Overview:Explain the key features and advantages of using Flutter for mobile app development. Discuss how the Flutter framework differs from traditional approaches and why it has gained popularity in the developer community. 2. Widget Tree and Composition: Describe the concept of the widget tree in Flutter. Explain how widget composition is used to build complex user interfaces. Provide examples of commonly used widgets and their roles in creating a widget tree. 3. State Management in Flutter: Discuss the importance of state management in Flutter applications. Compare and contrast the different state management approaches available in Flutter, such as setState, Provider, and Riverpod. Provide scenarios where each approach is suitable. 4. Firebase Integration in Flutter: Explain the process of integrating Firebase with a Flutter application. Discuss the benefits of using Firebase as a backend solution. Highlight the Firebase services commonly used in Flutter development and provide a brief overview of how data synchronization is achieved.
Roll No.	47
Name	Vedant Sanap
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	

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.	47
Name	Vedant Sanap
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:	