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 **Vedang Vijay Wajge** 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

Project Title: PillMate(MAD) & Static PWA(PWA) **Roll No. 62**

Year/Sem/Class **:** D15A/D15B **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: PillMate(MAD) & Static PWA(PWA) **Roll No. 62**

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

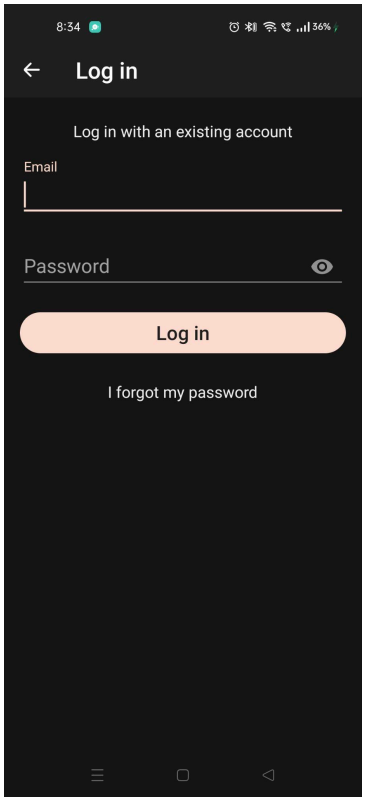
Sr. No	Experiment Title	LO	DOP	DOS	Grade
1.	To install and configure the Flutter Environment	LO1			
2.	To design Flutter UI by including common widgets.	LO2			
3.	To include icons, images, fonts in Flutter app	LO2			
4.	To create an interactive Form using form widget	LO2			
5.	To apply navigation, routing 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			

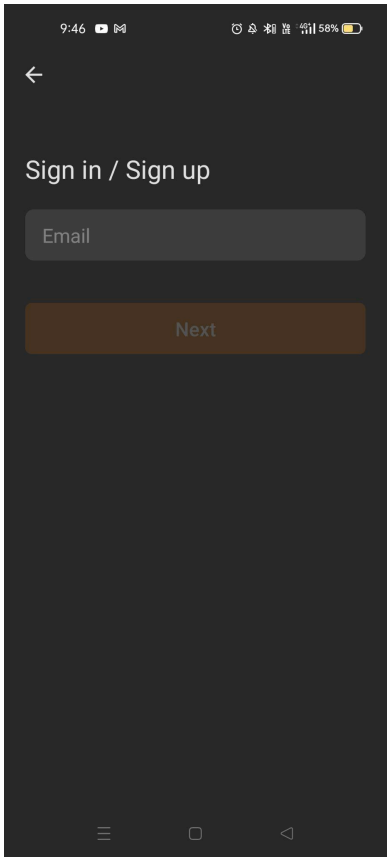
MPL & PWA LAB PREREQUISITE

Name: - Vedang Wajge **Class: -** D15A **Roll: No: -** 62

AIM: - Selecting features for application development, the features should comprise of:

1. Common Widgets
2. Should include icons, images, charts etc.
3. Should have an interactive form.
4. Should apply navigation, routing and gestures
5. Should connect with FireBase database

Screen Shot	Features
	• **Purpose**: Enables users to register or log in by Email or Google to access the app's features. • **Features**: 1. **Form Field** 2. **Google Sign in Button(Optional)** : A button to sign up with google and authenticate/register the user. • **Firebase Integration** Ensures secure login by storing user details in Firebase Authentication.



Login Page:

- **Purpose**:
  Enables users to register or log in by providing Email to access the app's features.

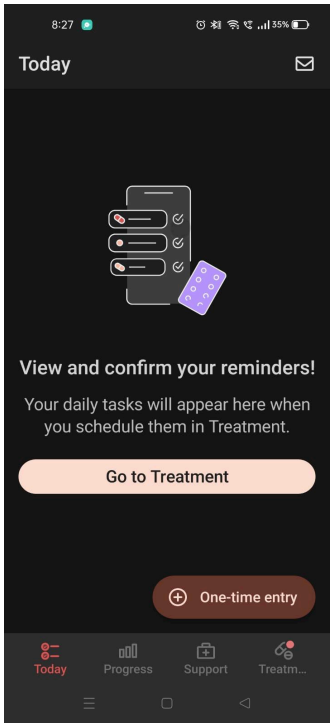
- **Features**:
  1. **Form Field**:
     Email
  2. **Next Button**:
     A button to submit the email and authenticate/register the user.

- **Firebase**

  Integration:

  Firestore Database:

  Saves user's email in a Firestore collection for real-time access and updates.



HomePage

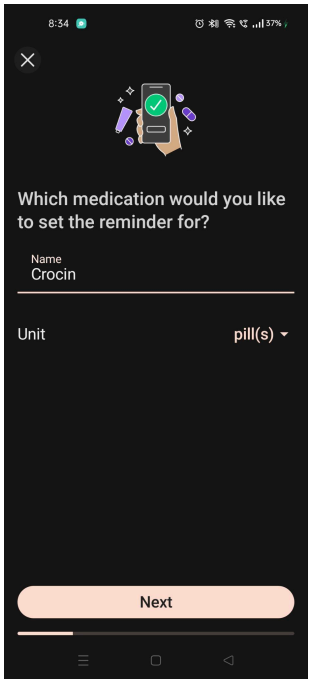
Bottom Taskbar:

- **Purpose**: A taskbar at the bottom with buttons to view reminders, calendar, progress etc..
- **Widgets Used**: Icons with select animations.

Feature Icons:

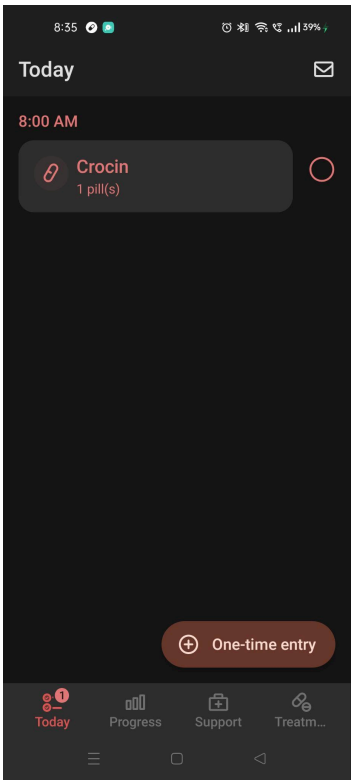
- **Purpose**: Interactive icons representing categories like add medicines, profile, etc
- **Widgets Used**: Icon Buttons.

.



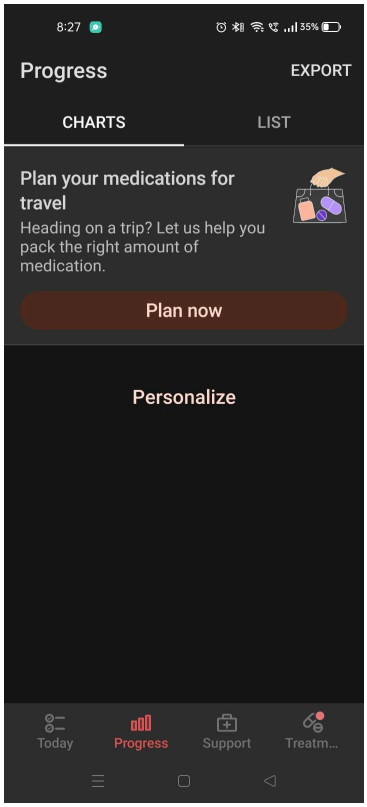
Add Medicine Section:

- **Purpose**: Provides user to add medicines by clicking the add pill button.
- **Widgets Used**:
  - Textbox to enter the pill and other buttons to add label, frequency, etc.
  - Elevated button to save the pill.



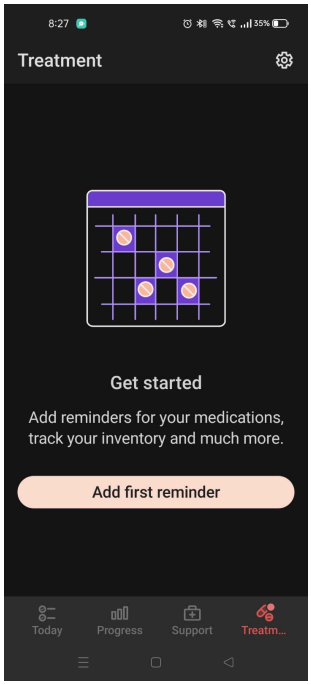
Body Section:

- **Purpose**: Provides user to interact with pills by editing or deleting them or marking them as taken.
- **Widgets Used**:
  - Hover button and checkbox.
  - List View for displaying pills for the day.



Reports Section:

- **Purpose**: Provides user to view pill adherence reports.
- **Widgets Used**:
  - Charts and reports of daily, monthly and yearly views.
  - List view for medication history.

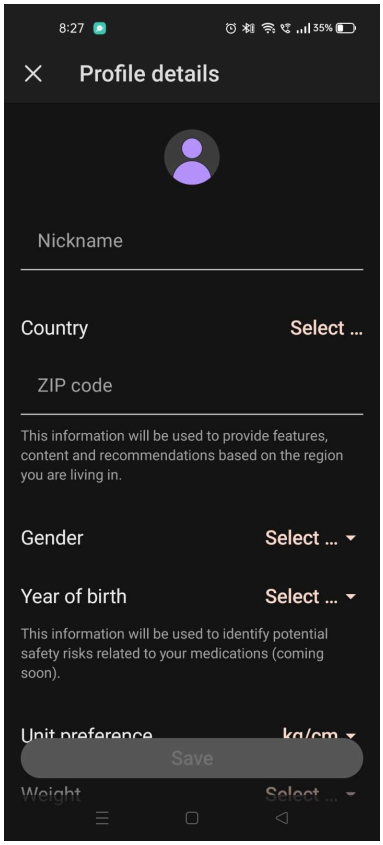


Calendar Section:

Purpose:
Provides user with full calendaric view to check pending and upcoming pills for the whole month.

Widgets Used:

- **Calendar**: To display the calendar in a clean and interactive format.
- **IconButton**: To mark a pill as taken or skip it.



User Account Page :

Purpose:
Allows users to view and edit their personal details, manage preferences, and update their account information seamlessly.

Features:

1. **Profile Picture**:
   A placeholder image that can be replaced with the user's chosen profile picture.
2. **Editable Fields**:
   Includes fields for Name, Phone and Email.
3. **Badges**:
   - Badges to showcase the consistency for the user.
   - A bit of statistical data of the user to monitor the activity.

Widgets Used:
Form Widgets, Buttons, ListView, Icons

MAD & PWA Lab
Journal

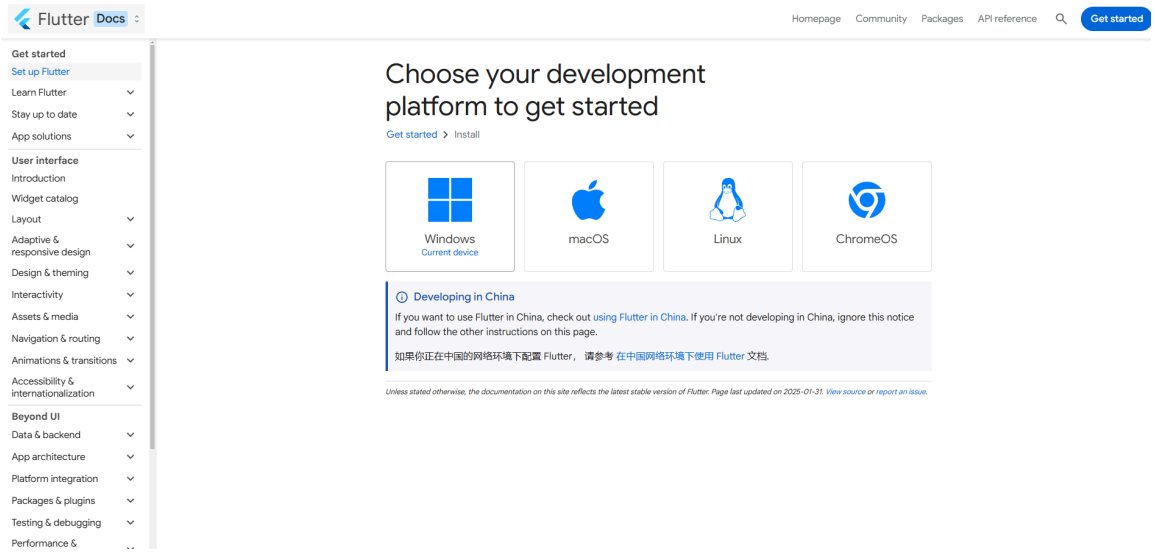
Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	62
Name	Vedang V. Wajge
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	

EXPERIMENT NO: 1

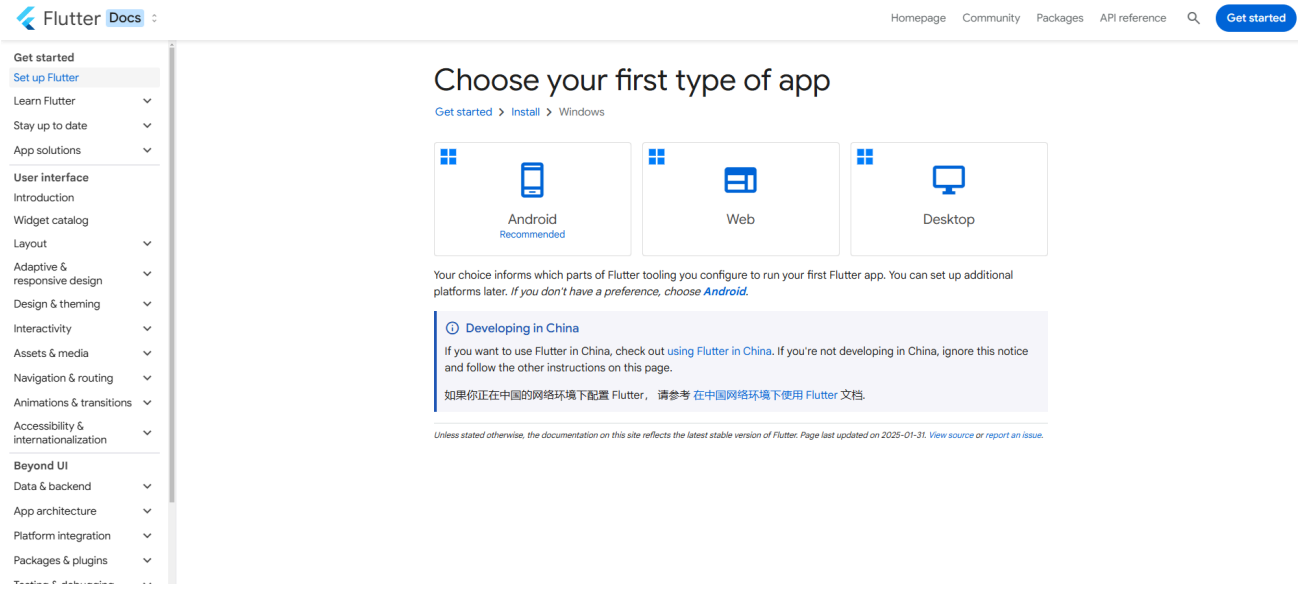
Name : Vedang V. Wajge
Class : D15A
Roll No: 62

AIM: - Installation and Configuration of Flutter Environment.

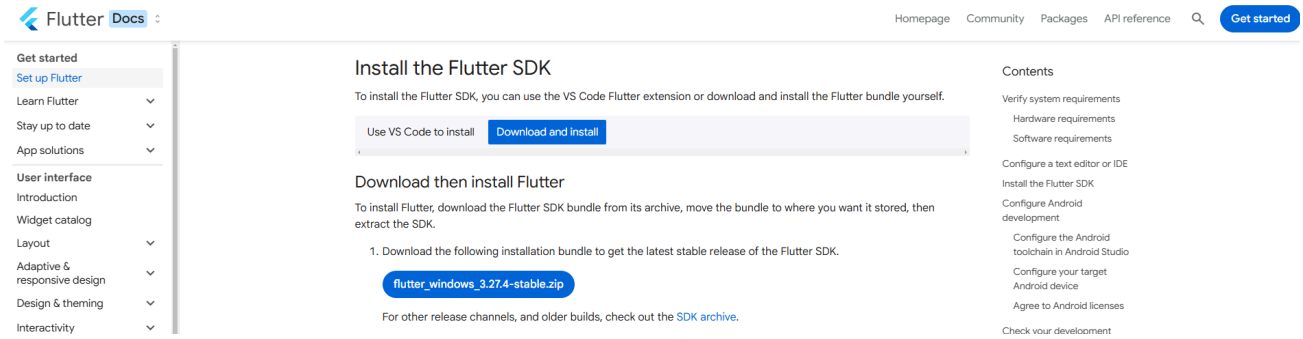
Step 1: Go to the official Flutter website: https://docs.flutter.dev/get-started/install



Step 2: To download the latest Flutter SDK, click on the Windows icon > Android

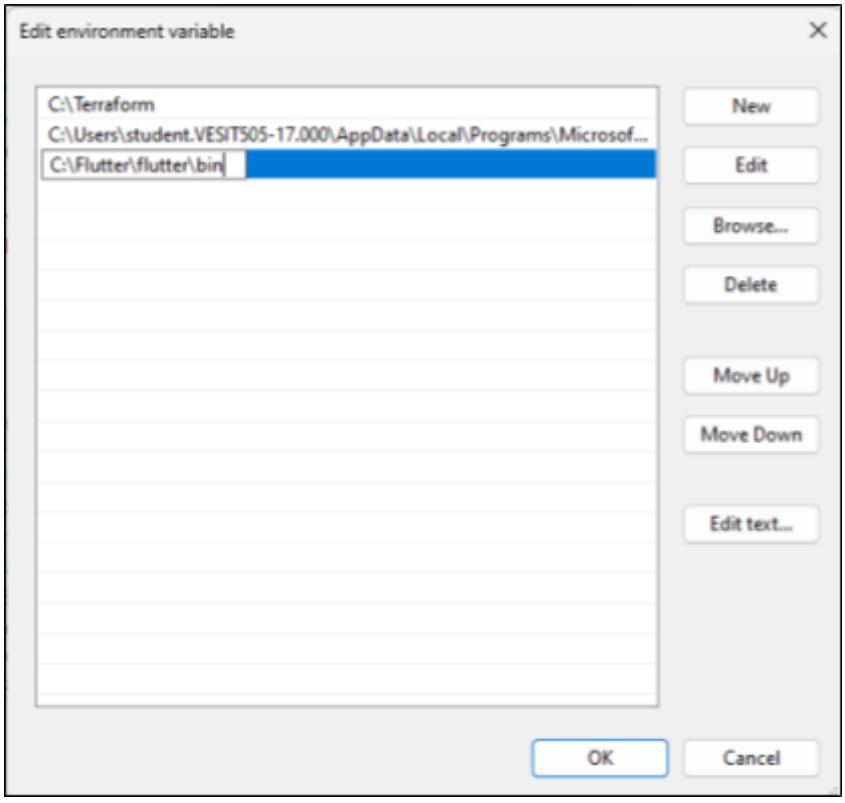


Step 3: For Windows, download the stable release (a .zip file).

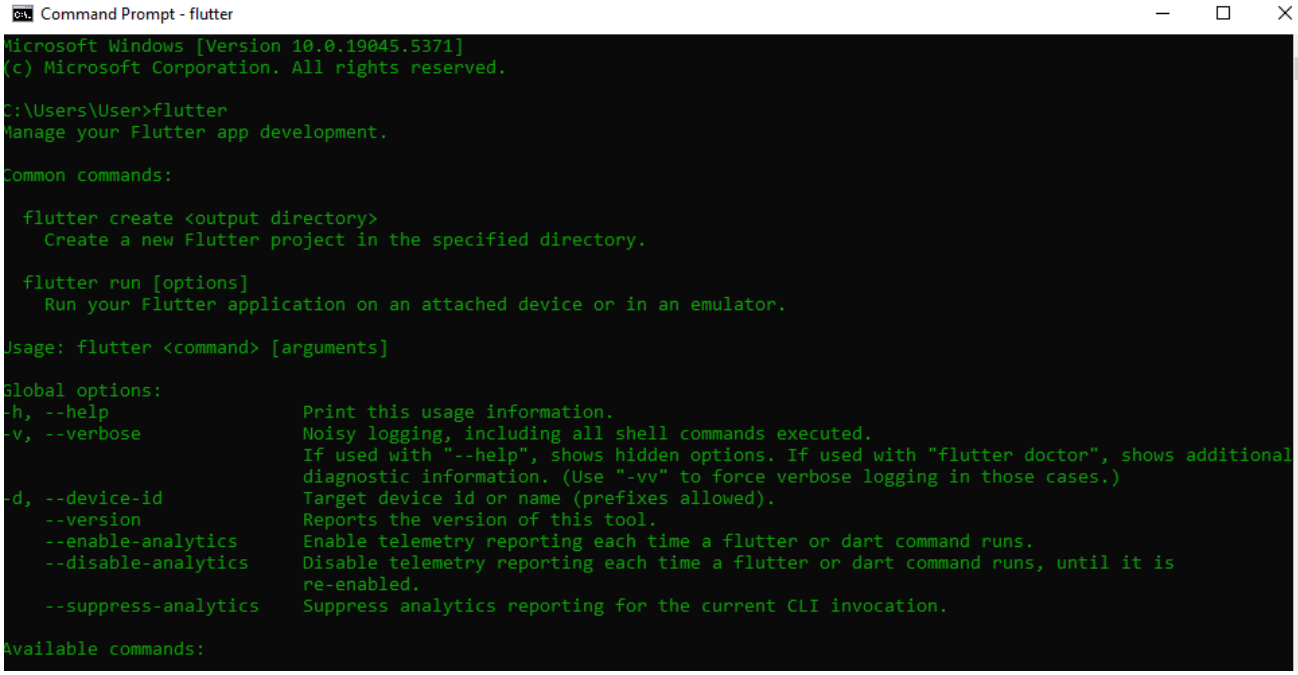


Step 4: Extract the zip file and copy paste the folder in C: Drive

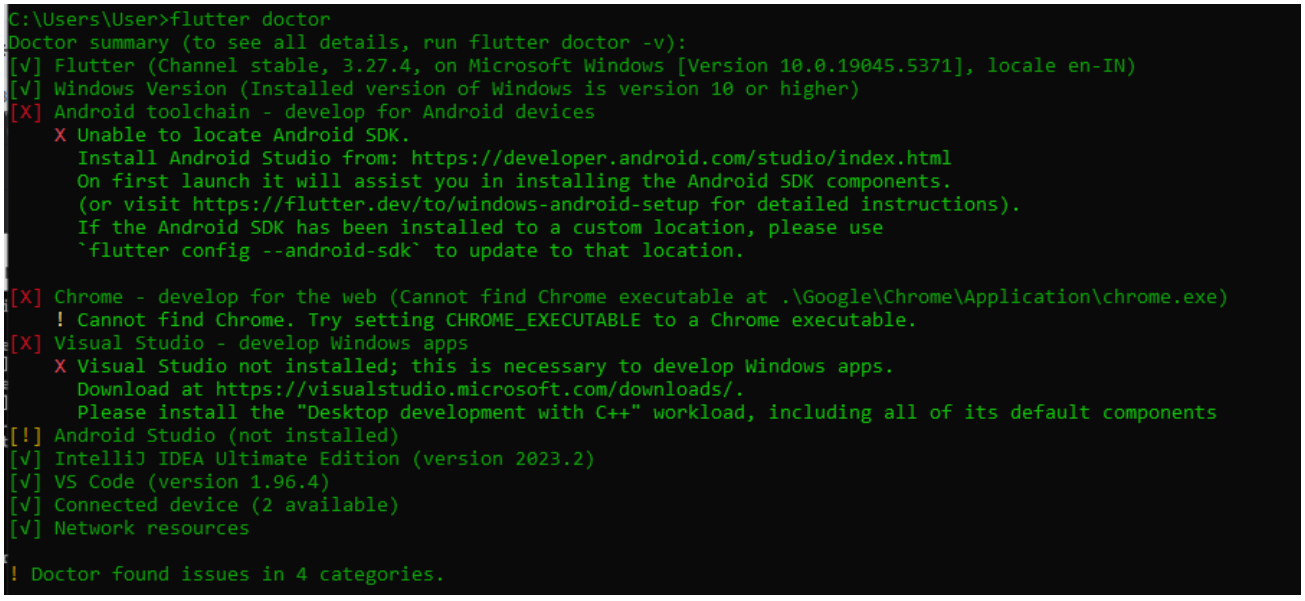
Step 5 :- Add Flutter to System PATH
Right-click on the Start Menu > System > Advanced system settings > Environment Variables.
Under System Variables, find Path and click Edit.
Add the full path to the flutter/bin directory (e.g., C:\flutter\bin).



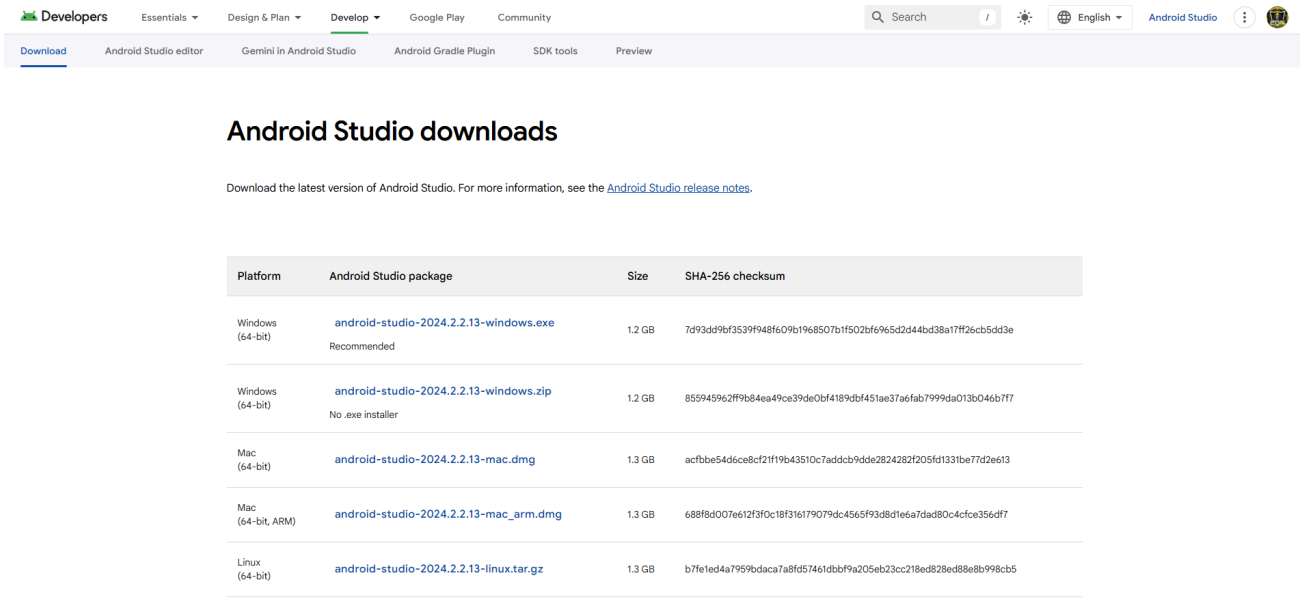
Step 6 : - Now, run the $ flutter command in command prompt



Step 7:- Run the $ flutter doctor command. This command checks for all the requirements of Flutter app development and displays a report of the status of your Flutter installation

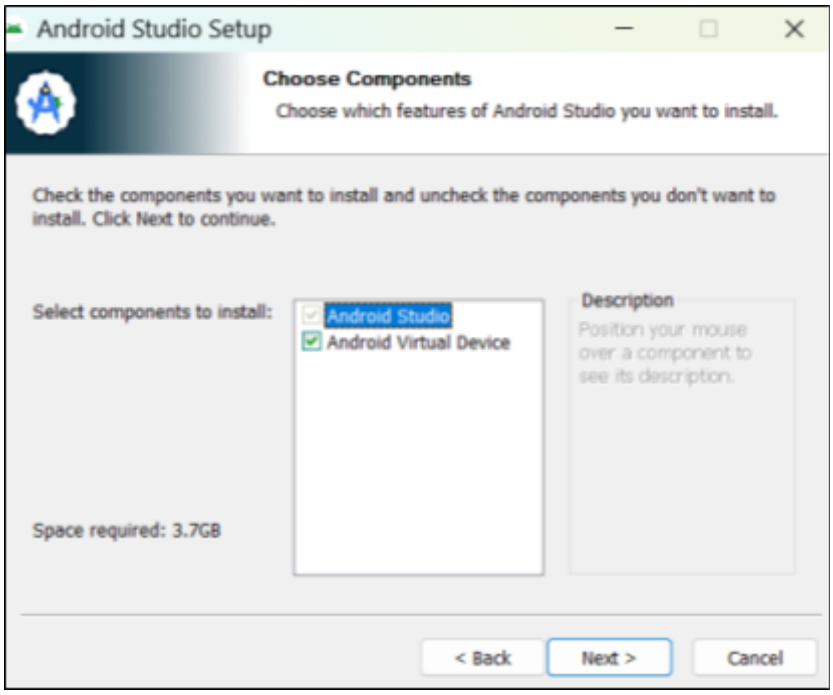


Step 8 : - Go to Android Studio and download the installer

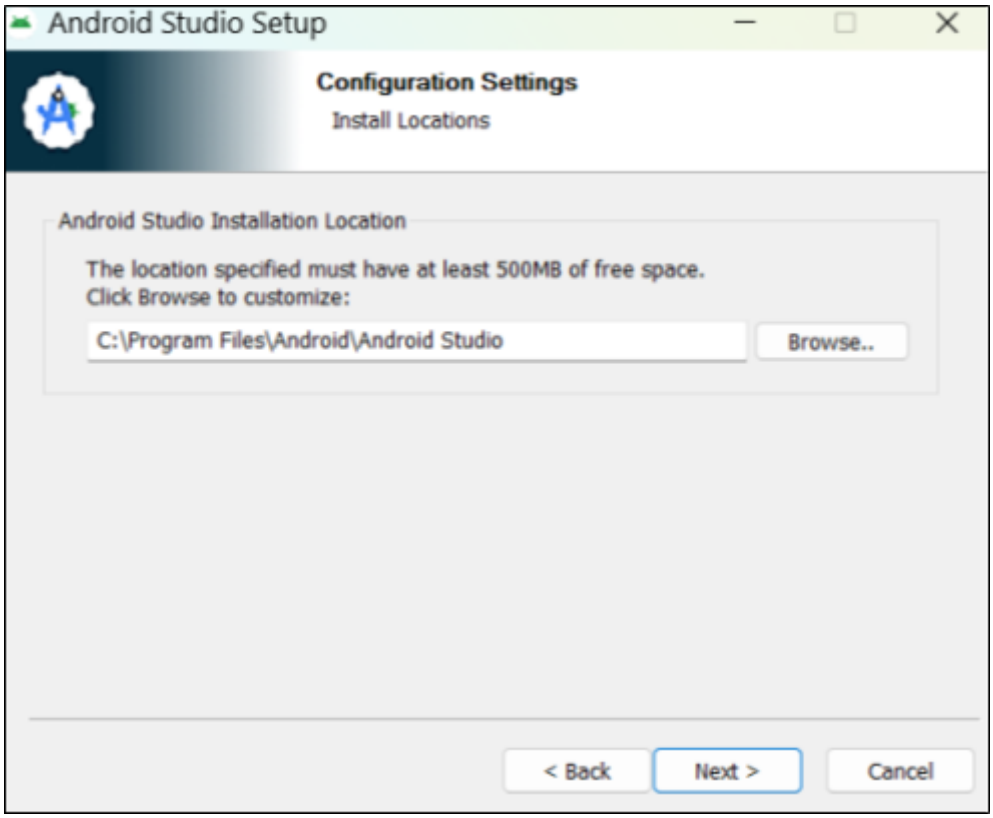


Step 8.1: - When the download is complete, open the .exe file and run it. You will get the following dialog box

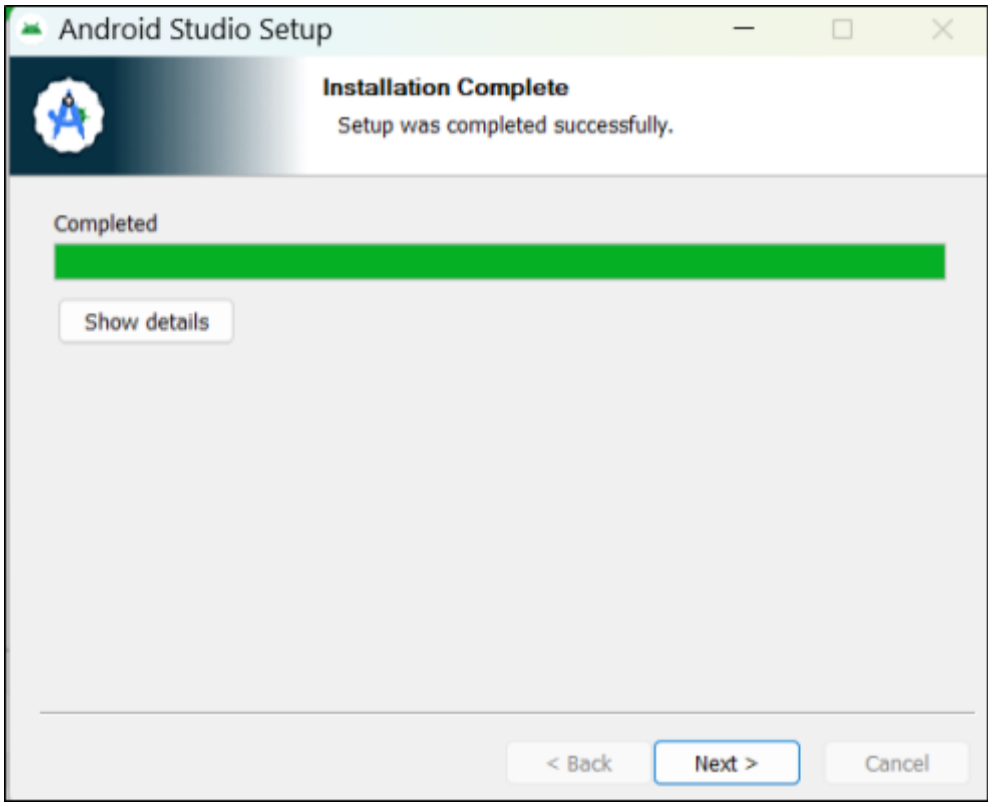
Step 8.2: - Select all the Checkboxes and Click on ‘Next’ Button.

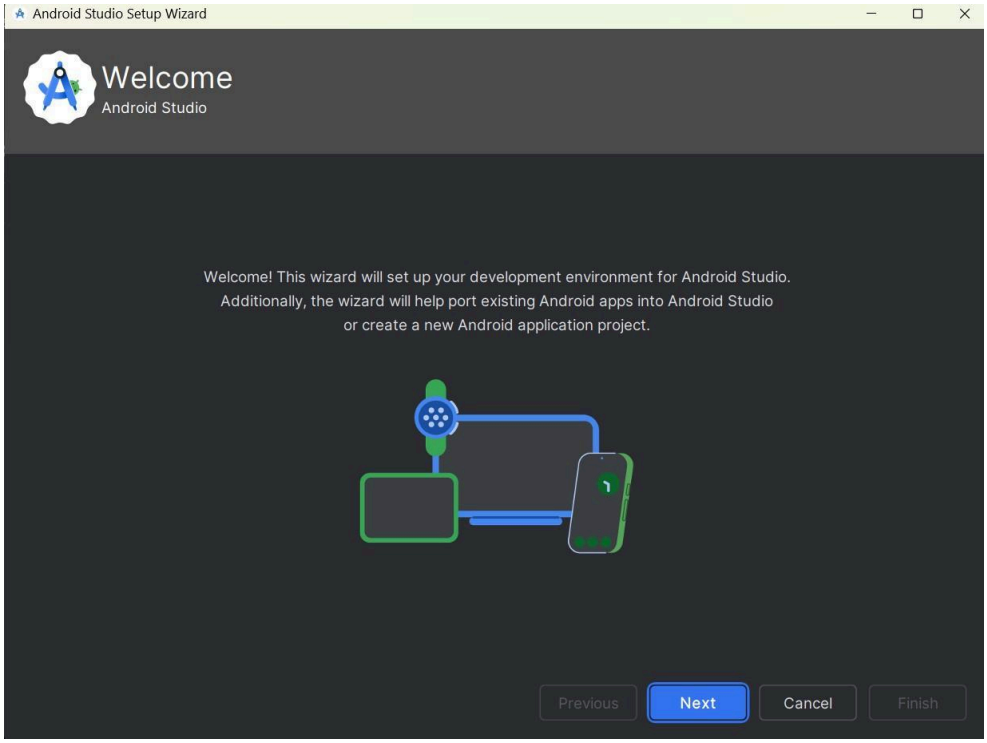


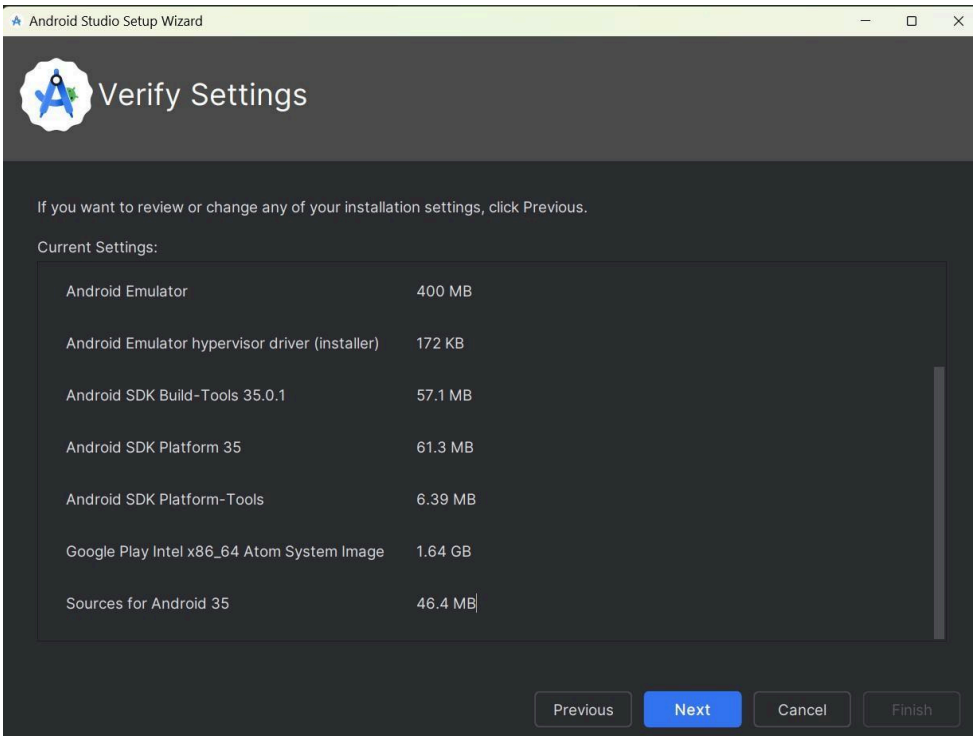
Step 8.3: - Change the destination as per your convenience and click on ‘Next’ Button.

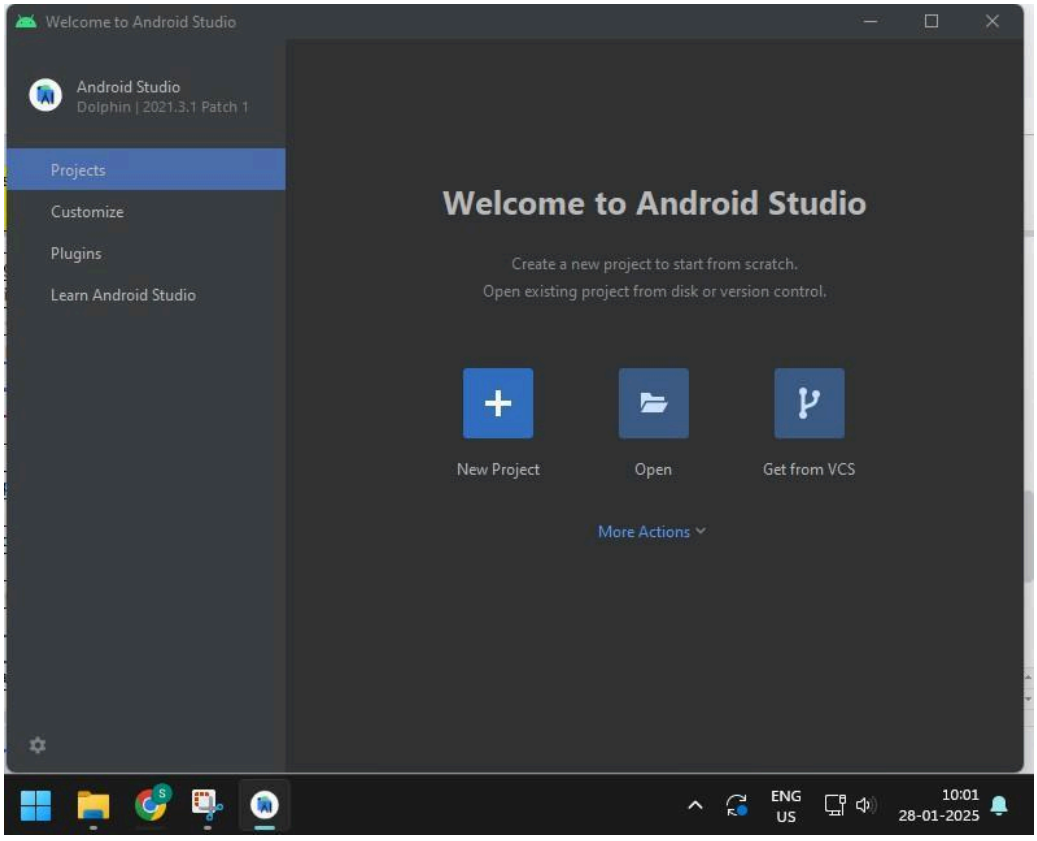


Step 8.4: - Follow the steps of the installation wizard. Once the installation wizard completes, you will get the following screen.

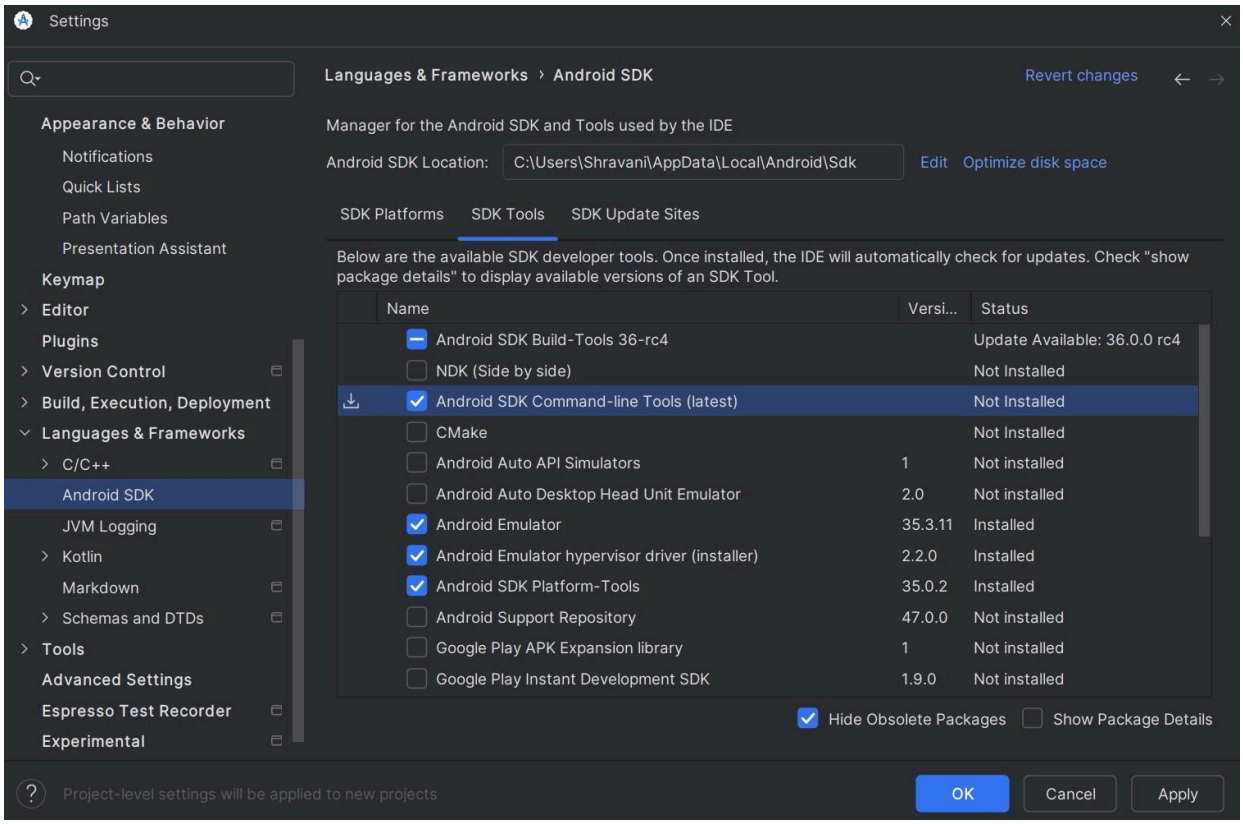





Android Studio Setup Wizard
Verify Settings
If you want to review or change any of your installation settings, click Previous.
Current Settings:
Android Emulator 400 MB
Android Emulator hypervisor driver (installer) 172 KB
Android SDK Build-Tools 35.0.1 57.1 MB
Android SDK Platform 35 61.3 MB
Android SDK Platform-Tools 6.39 MB
Google Play Intel x86_64 Atom System Image 1.64 GB
Sources for Android 35 46.4 MB
Previous
Next
Cancel
Finish


Welcome to Android Studio
Android Studio
Dolphin | 2021.3.1 Patch 1
Projects
Customize
Plugins
Learn Android Studio
Welcome to Android Studio
Create a new project to start from scratch.
Open existing project from disk or version control.
New Project
Open
Get from VCS
More Actions
ENG
US
10:01
28-01-2025

Step 8.5: - Go to Preferences > Appearance & Behavior > System Settings > Android SDK. Select the SDK Tools tab and check Android SDK Command-line Tools and Install it.

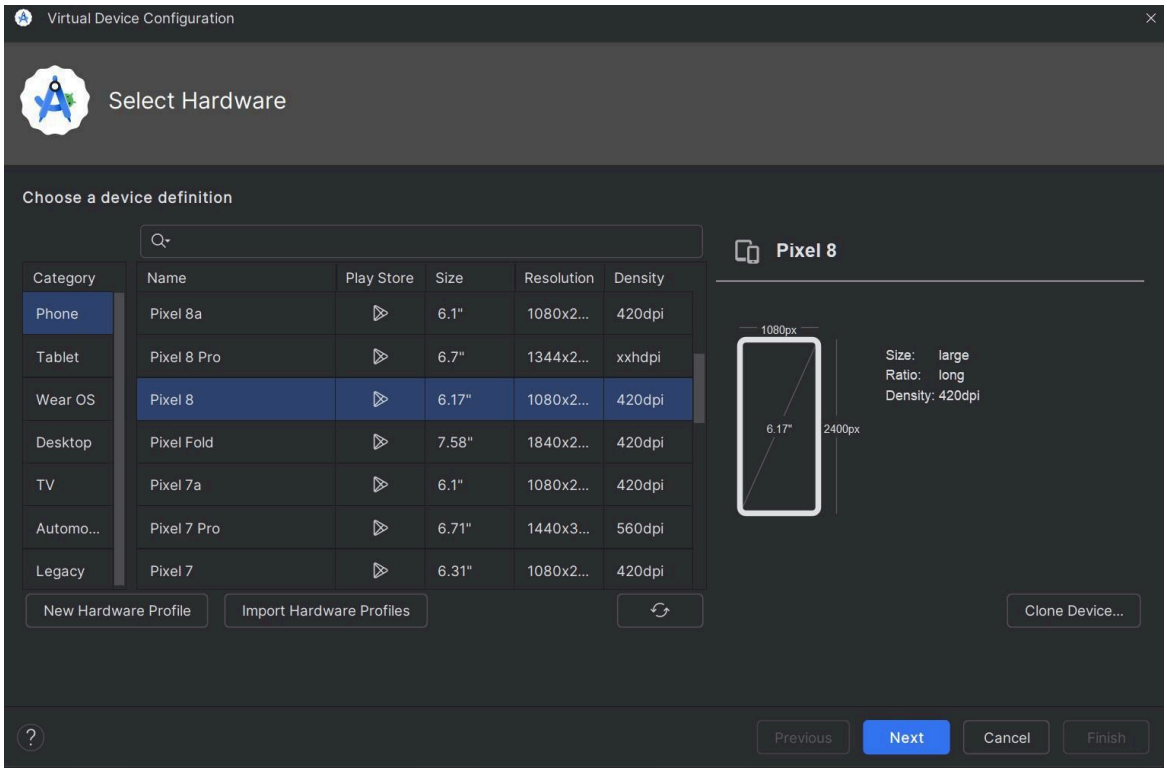


Step 9: - Open a terminal and run the following command

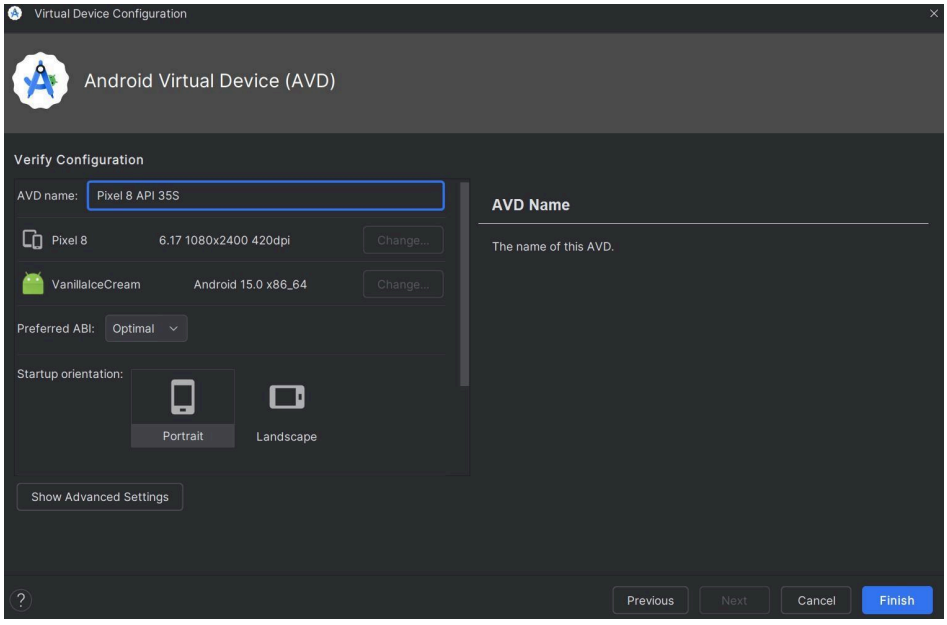
```
C:\Users\User>flutter doctor --android-licenses
[======================================] 100% Computing updates...
5 of 7 SDK package licenses not accepted.
Review licenses that have not been accepted (y/N)? y

1/5: License android-googletv-license:
---------------------------------------
Terms and Conditions

This is the Google TV Add-on for the Android Software Development Kit License Agreement.

1. Introduction
```

```
C:\Users\User>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, 3.27.4, on Microsoft Windows [Version 10.0.19045.5371], locale en-IN)
[√] Windows Version (Installed version of Windows is version 10 or higher)
[!] Android toolchain - develop for Android devices (Android SDK version 35.0.1)
    ! Some Android licenses not accepted. To resolve this, run: flutter doctor --android-licenses
[X] Chrome - develop for the web (Cannot find Chrome executable at .\Google\Chrome\Application\chrome.exe)
    ! Cannot find Chrome. Try setting CHROME_EXECUTABLE to a Chrome executable.
[X] Visual Studio - develop Windows apps
    X Visual Studio not installed; this is necessary to develop Windows apps.
      Download at https://visualstudio.microsoft.com/downloads/.
      Please install the "Desktop development with C++" workload, including all of its default components
[√] Android Studio (version 2024.2)
[√] IntelliJ IDEA Ultimate Edition (version 2023.2)
[√] VS Code (version 1.96.4)
[√] Connected device (2 available)
[√] Network resources

! Doctor found issues in 3 categories.
```

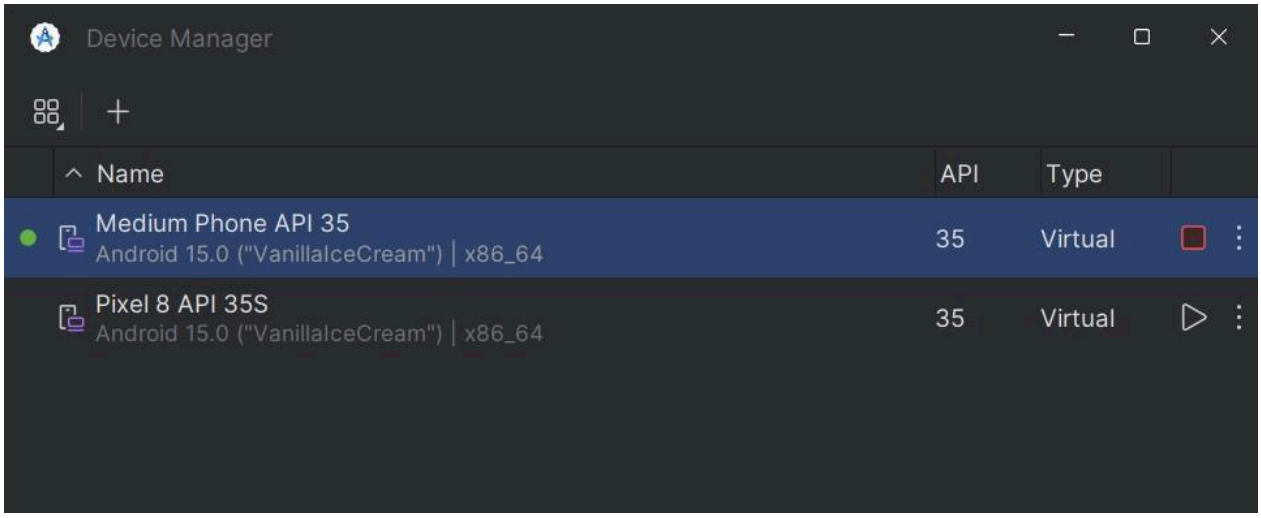
Step 10: - Next, you need to set up an Android emulator. It is responsible for running and testing the Flutter application

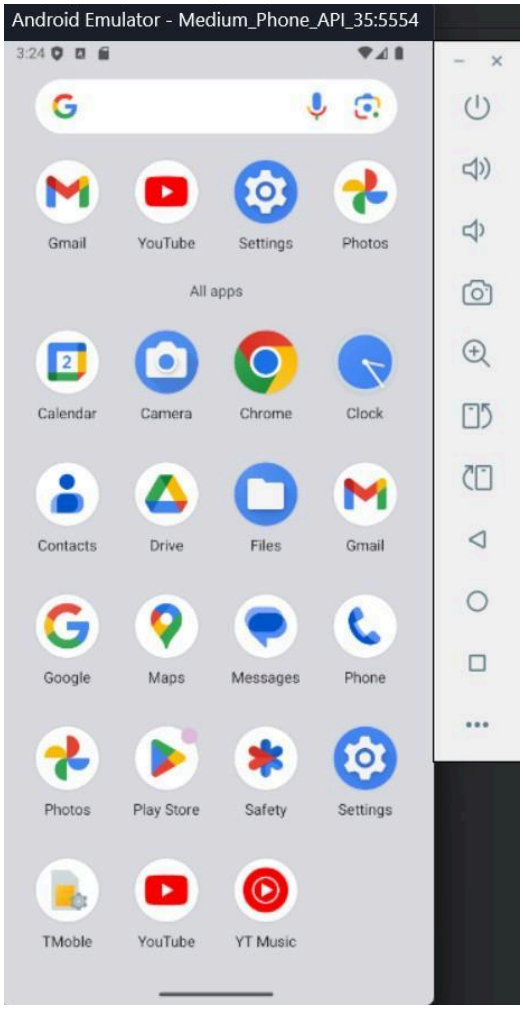


Step 10.1: - Open Android Studio and go to Tools > AVD Manager. Create a new virtual device

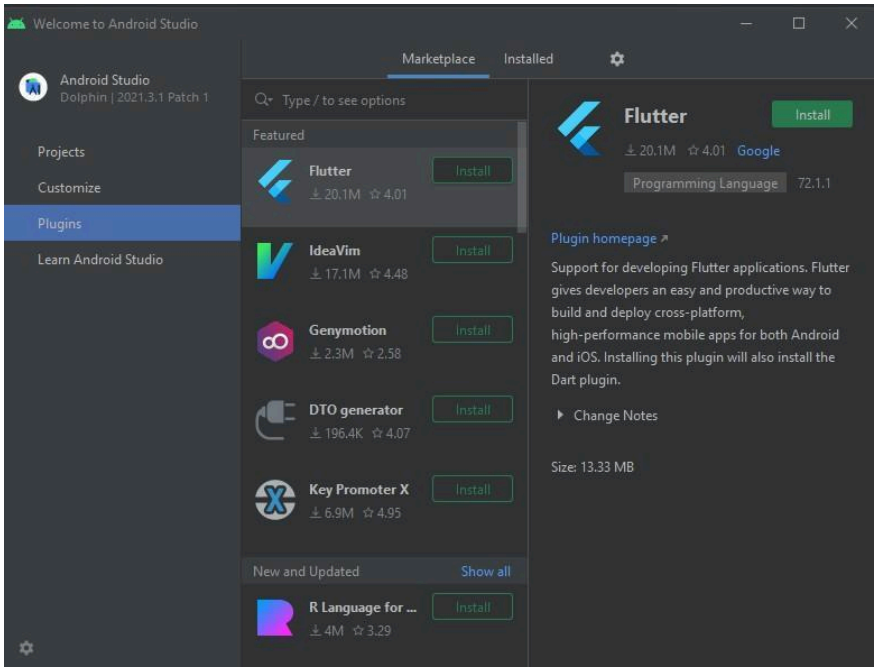


Step 10.2: - Click on the icon pointed into the red color rectangle. The Android emulator displayed as below screen

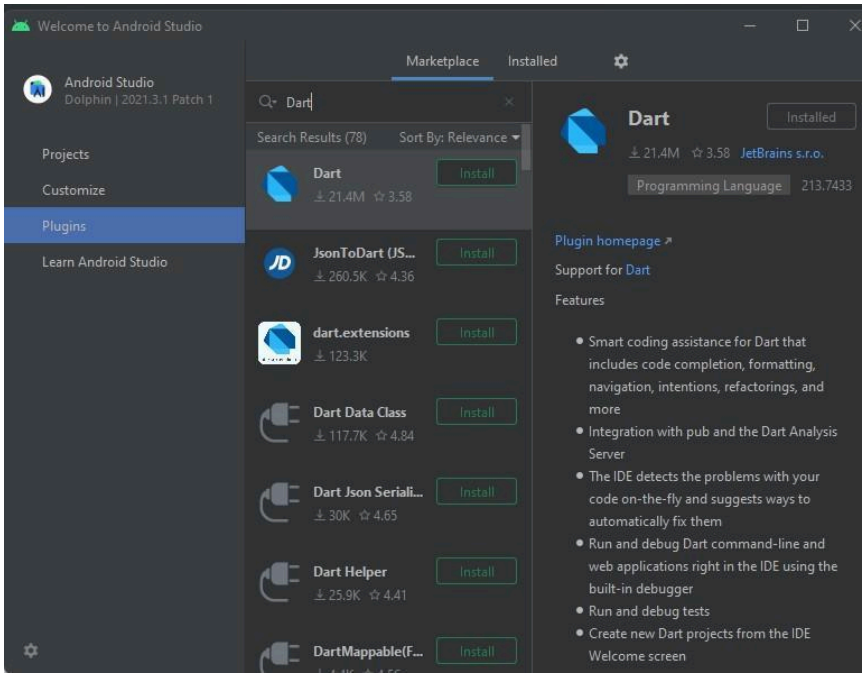




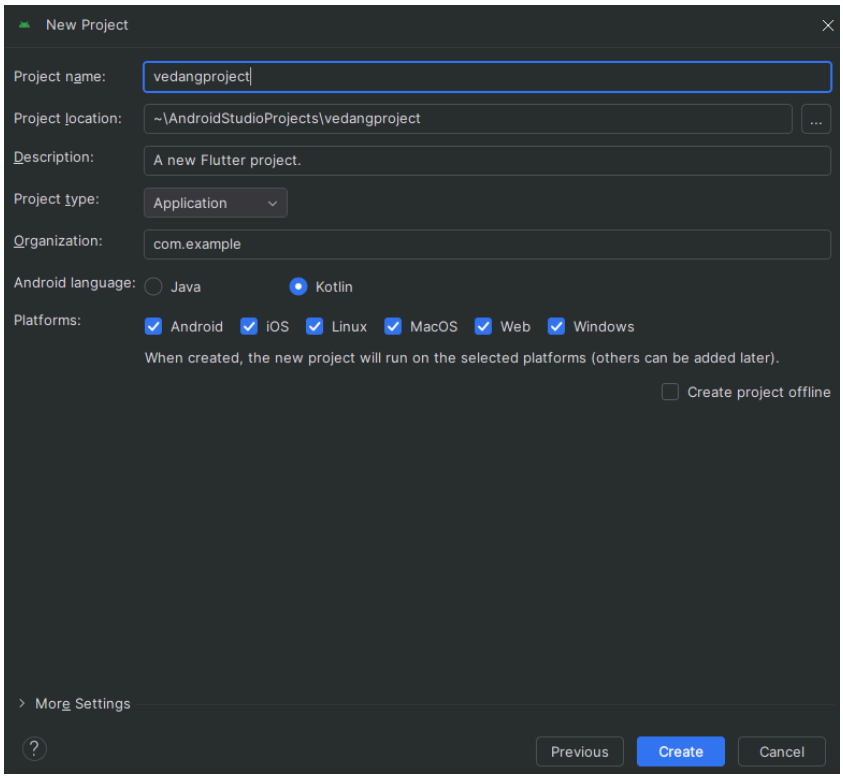
Step 11: - Now, install Flutter and Dart plugin for building Flutter application in Android Studio. These plugins provide a template to create a Flutter application, give an option to run and debug Flutter application in the Android Studio itself

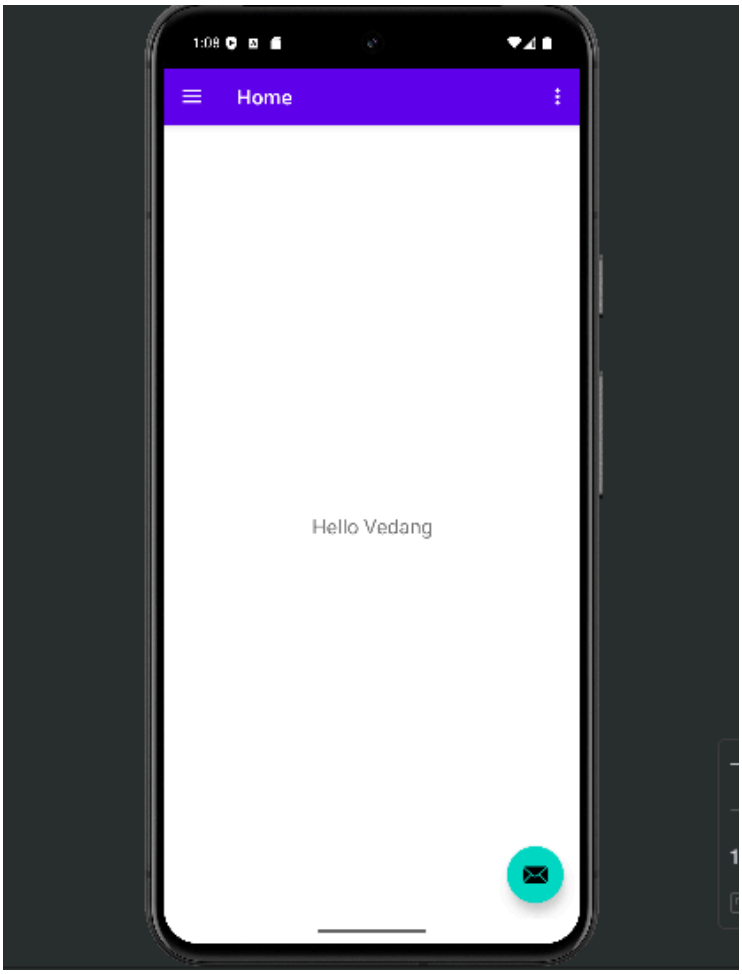


Step 11.1: - Open the Android Studio and then go to File->Settings->Plugins. Now, search the Flutter plugin. If found, select Flutter plugin and click insta



Step 12: Go to File > New Project select project name , location and click next





MAD & PWA Lab
Journal

Experiment No.	02
Experiment Title.	To design Flutter UI by including common widgets.
Roll No.	62
Name	Vedang V. Wajge
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

EXPERIMENT NO: - 02

Name:- Vedang Wajge **Class:-** D15A **Roll:No: -** 62

AIM: - To design Flutter UI by including common widgets.

Theory: -

Each element on the screen of the Flutter app is a widget. The view of the screen completely depends upon the choice and sequence of the widgets used to build the apps. And the structure of the code of apps is a tree of widgets.

When you made any alteration in the code, the widget rebuilds its description by calculating the difference of previous and current widget to determine the minimal changes for rendering in UI of the app.Widgets are nested with each other to build the app. It means the root of your app is itself a widget, and all the way down is a widget also. For example, a widget can display something, can define design, can handle interaction, etc.

The single child layout widget is a type of widget, which can have only **one child widget** inside the parent layout widget. These widgets can also contain special layout functionality. Flutter provides us many single child widgets to make the app UI attractive. If we use these widgets appropriately, it can save our time and makes the app code more readable.

The multiple child widgets are a type of widget, which contains **more than one child widget**, and the layout of these widgets are **unique**. For example, Row widget laying out of its child widget in a horizontal direction, and Column widget laying out of its child widget in a vertical direction. If we combine the Row and Column widget, then it can build any level of the complex widget.

Type of Widgetss

- **StatefulWidget**

  A StatefulWidget has state information. It contains mainly two classes: the state object and the widget. It is dynamic because it can change the inner data during the widget lifetime. This widget does not have a build() method. It has createState() method, which returns a class that

extends the Flutters State Class. The examples of the StatefulWidget are Checkbox, Radio, Slider, InkWell, Form, and TextField.

- **StatelessWidget**

  The StatelessWidget does not have any state information. It remains static throughout its lifecycle. The examples of the StatelessWidget are Text, Row, Column, Container, etc.

Some of the commonly used widgets

Container – A box widget used for styling with padding, margins, colors, borders, and constraints. It helps in layout structuring and positioning.

Row & Column – Used to arrange widgets in horizontal (Row) or vertical (Column) orientation. They manage spacing, alignment, and distribution of child widgets.

Stack – Overlaps widgets on top of each other, useful for creating layered UIs like banners, tooltips, or floating elements.

Text – Displays text on the screen with customizable font size, color, alignment, and styling options

Image – Loads and displays images from assets, network, or memory with scaling, fit, properties.

Scaffold – Provides a basic layout structure with an app bar, body, floating action button, and bottom navigation.

ListView – A scrollable list widget that efficiently renders large amounts of dynamic content. Supports both vertical and horizontal scrolling.

GridView – Displays widgets in a grid format, useful for galleries, product listings, or dashboards. It supports dynamic column adjustments.

SizedBox – Used to create space between widgets or define fixed width and height for layout adjustments.

ElevatedButton – A button with elevation that provides a raised effect, customizable with color, shape, and click actions.

TextField – A user input field that supports text entry, keyboard configurations, validation.

AppBar – A top navigation bar that includes a title, actions, and menu icons, commonly used in Scaffold.

BottomNavigationBar – A bar at the bottom of the screen used for navigation between different app sections with icons and labels.

Drawer – A side navigation panel that slides out from the left, typically used for app menus and quick navigation.

Card – A material design component that displays content inside a box with elevation.

Code: -

home_page.dart

```
import 'package:flutter/material.dart';
import 'myaccountpage.dart';
import 'pills_section.dart';
import 'crop_list_page.dart';
import 'weather_forecast_page.dart';

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: HomePage(),
    );
  }
}

class HomePage extends StatefulWidget {
  @override
  _HomePageState createState() =>
_HomePageState();
}

class _HomePageState extends
State<HomePage> {
  int _currentIndex = 0;

  // Method to handle the bottom navigation item
taps
  void _onItemTapped(int index) {
    setState(() {
      _currentIndex = index;
    });
    if (index == 1) {
      Navigator.push(
        context,
        MaterialPageRoute(builder: (context) =>
WeatherPage()),
      );
    }
    if (index == 3) {
      Navigator.push(
        context,
        MaterialPageRoute(builder: (context) =>
CropListPage()),
      );
    }
    if (index == 4) {
      Navigator.push(
        context,
        MaterialPageRoute(builder: (context) =>
MyAccountPage()),
      );
    }
  }


  // Returns the widget corresponding to the
selected index
  Widget _getSelectedPage() {
    switch (_currentIndex) {
      case 0:
        return Center(child: Text('Home Page'));
      case 1:
        return Center(child: Text('Weather Page'));
      case 2:
        return Center(child: Text('Disease
Detection Page'));
      case 3:
        return Center(child: Text('Crop'));
      case 4:
        return Center(child: Text('Profile Page'));
      default:
        return Center(child: Text('Home Page'));
    }
  }
```

```
  @override
  Widget build(BuildContext context) {
   return Scaffold(
    backgroundColor: Colors.white,
    appBar: AppBar(
     backgroundColor: Colors.white,
     title: Row(
      children: [
       Image.asset('assets/images/logo.png',
height: 40), // Your logo here
       SizedBox(width: 10),
       Text('AgriApp'),
      ],
     ),
     actions: [
      IconButton(
       icon: Icon(Icons.search),
       onPressed: () {
        // Add search functionality
       },
      ),
     ],
    ),
    body: SingleChildScrollView(
     child: Column(
      children: [
       BannerSection(),
       CategorySection(),
       DealsOfTheDaySection(),
       FreeDeliverySection(),
       NewsSection(),
      ],
     ),
    ),
    bottomNavigationBar:
BottomNavigationBar(
     currentIndex: _currentIndex, // Keep track
of the selected index
     onTap: _onItemTapped, // Update the index
when an item is tapped
     selectedItemColor: Colors.black, // Color
for the selected item
     unselectedItemColor: Colors.black54, //
Color for unselected items
     type: BottomNavigationBarType.fixed, //
Ensures icons stay the same size
     items: const <BottomNavigationBarItem>[
      BottomNavigationBarItem(
       icon: Icon(Icons.home),
       label: 'Home',
      ),
      BottomNavigationBarItem(
       icon: Icon(Icons.cloud), // Weather icon
       label: 'Weather',
      ),
      BottomNavigationBarItem(
       icon: Icon(Icons.camera_alt), // Disease
detection icon
       label: 'Detect',
      ),
      BottomNavigationBarItem(
       icon: Icon(Icons.local_florist), // Crop
practices icon
       label: 'Crops',
      ),
      BottomNavigationBarItem(
       icon: Icon(Icons.account_circle), //
Profile icon
       label: 'Profile',
      ),
     ],
    ),
   );
  }
}

class BannerSection extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
   return Padding(
    padding: EdgeInsets.all(14.0),
    child: ClipRRect(
     borderRadius: BorderRadius.circular(20),
     child: Container(
      width: double.infinity,
      height: 200,
      decoration: BoxDecoration(
       image: DecorationImage(
        image:
AssetImage('assets/images/banner.jpg'),
        //fit: BoxFit.cover,
       ),
      ),
```

```
        ),
      ),
    );
  }
}

class CategorySection extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Container(
      padding: EdgeInsets.symmetric(vertical: 20),
      child: GridView.count(
        crossAxisCount: 4,
        shrinkWrap: true,
        physics: NeverScrollableScrollPhysics(),
        children: [
          CategoryItem('Offers',
'assets/images/offers.png'),
          CategoryItem('Crop Tonics',
'assets/images/crop_tonics.png'),
          CategoryItem('Fertilizer',
'assets/images/fertilizer.png'),
          CategoryItem('Pesticides',
'assets/images/pesticides.png'),
        ],
      ),
    );
  }
}

class CategoryItem extends StatelessWidget {
  final String title;
  final String imagePath; // Changed from
IconData to String

  CategoryItem(this.title, this.imagePath);

  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        Container(
          width:80,
          height:80,
          decoration: BoxDecoration(
            color: Colors.green.withAlpha(50), //
Light green background
            shape: BoxShape.circle,
          ),
          padding: EdgeInsets.all(16),
          child: Image.asset(imagePath, width:
double.infinity, height: double.infinity,
fit:BoxFit.cover), // Load image instead of icon
        ),
        SizedBox(height: 5),
        Text(title, textAlign: TextAlign.center,
style: TextStyle(fontSize: 12, fontWeight:
FontWeight.bold)),
      ],
    );
  }
}

class DealsOfTheDaySection extends
StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Container(
      padding: EdgeInsets.symmetric(vertical: 10),
      child: Column(
        crossAxisAlignment:
CrossAxisAlignment.start,
        children: [
          // Row for title and "See All" button
          Padding(
            padding: EdgeInsets.only(left: 20),
            child: Row(
              mainAxisAlignment:
MainAxisAlignment.spaceBetween,
              children: [
                Text(
                  'Deals of the day!',
                  style: TextStyle(fontSize: 24,
fontWeight: FontWeight.bold),
                ),
                TextButton(
                  onPressed: () {
                    // Navigate to All Deals screen
                    Navigator.push(
                      context,
                      MaterialPageRoute(builder:
(context) => AllDealsScreen()),
                    );
                  },
```

```
              child: Text(
                'See All >',
                style: TextStyle(color: Colors.blue,
fontWeight: FontWeight.bold),
              ),
            ),
          ],
        ),
      ),
      SizedBox(height: 10),
      // Grid of products for Deals of the Day
      GridView.count(
        crossAxisCount: 2,
        shrinkWrap: true,
        physics: NeverScrollableScrollPhysics(),
        children: [
          ProductCard('00-00-50', '₹240.00',
'₹270.00', 'assets/images/product1.png'),
          ProductCard('12-61-00', '₹277.2',
'₹355.00', 'assets/images/product2.png'),
        ],
      ),
    ],
   ),
  );
 }
}

// Product Card widget
class ProductCard extends StatelessWidget {
 final String productName;
 final String price;
 final String originalPrice;
 final String imagePath;

 ProductCard(this.productName, this.price,
this.originalPrice, this.imagePath);

 @override
 Widget build(BuildContext context) {
  return Card(
   margin: EdgeInsets.all(8),
   //color: Colors.white, // White background
for card
   shape: RoundedRectangleBorder(
    borderRadius: BorderRadius.circular(8),
    side: BorderSide(color: Colors.grey, width:
1), // Black border
   ),
   child: Column(
    children: [
     Container(
      height: 100,
      width: 100,
      child: Image.asset(
       imagePath,
       fit: BoxFit.contain,
      ),
     ),
     Text(productName),
     Text(price, style: TextStyle(fontWeight:
FontWeight.bold)),
     Text(originalPrice, style:
TextStyle(decoration:
TextDecoration.lineThrough)),
    ],
   ),
  );
 }
}

class AllDealsScreen extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: Text('All Deals'),
   ),
   body: GridView.count(
    crossAxisCount: 2,
    padding: EdgeInsets.all(10),
    children: [
     // List of all product cards
     ProductCard('00-00-50', '₹240.00',
'₹270.00', 'assets/images/product1.png'),
     ProductCard('12-61-00', '₹277.2',
'₹355.00', 'assets/images/product2.png'),
     ProductCard('23-45-67', '₹500.00',
'₹600.00', 'assets/images/product3.png'),
     ProductCard('34-56-78', '₹350.00',
'₹400.00', 'assets/images/product4.png'),
     ProductCard('45-67-89', '₹220.00',
'₹280.00', 'assets/images/product5.png'),
     ProductCard('56-78-90', '₹310.00',
```

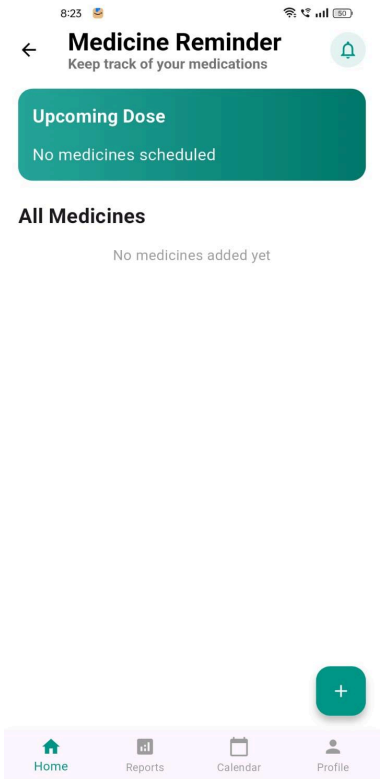
```
'₹370.00', 'assets/images/product6.png'),                    }
        ProductCard('67-89-01', '₹450.00',
'₹520.00', 'assets/images/product7.png'),
        ProductCard('78-90-12', '₹200.00',
'₹250.00', 'assets/images/product8.png'),
        ProductCard('89-01-23', '₹600.00',
'₹700.00', 'assets/images/product9.png'),
        ProductCard('90-12-34', '₹150.00',
'₹200.00', 'assets/images/product10.png'),
        ProductCard('01-23-45', '₹180.00',
'₹230.00', 'assets/images/product11.png'),
        ProductCard('12-34-56', '₹420.00',
'₹480.00', 'assets/images/product12.png'),
        // Add more ProductCard widgets here
       ],
     ),
   );
  }
}


class FreeDeliverySection extends
StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Container(
      padding: EdgeInsets.symmetric(vertical: 10),
// Vertical padding
      child: Center(
        child: Padding(
          padding:
EdgeInsets.symmetric(horizontal: 10), // Add
horizontal space (left & right)
          child: ClipRRect(
            borderRadius:
BorderRadius.circular(15), // Adjust the radius
as needed
            child: Image.asset(
              'assets/images/delivery.png',
              fit: BoxFit.cover,
            ),
          ),
        ),
      ),
    );
  }
```

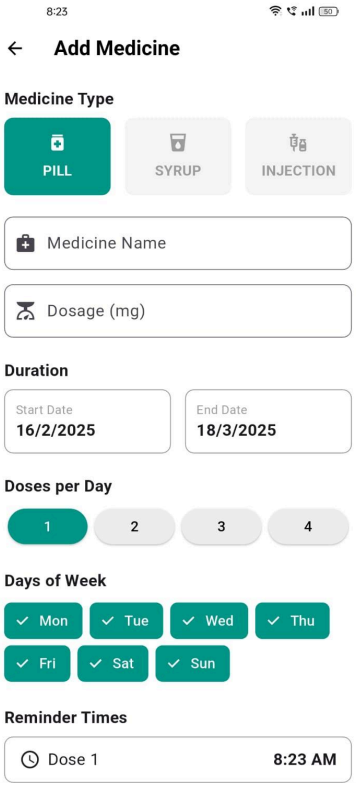
add_medicine.dart

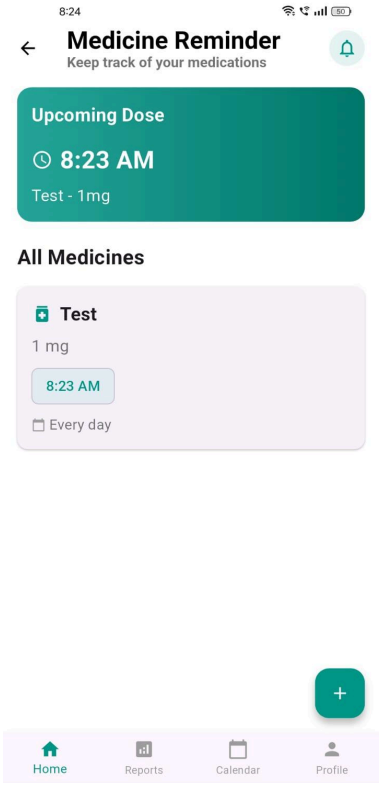
```
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
import 'dart:convert';

// Define the API endpoint
const String apiKey =
"7b40c2993a264008904439ac295309b0";
const String pillsApiUrl =

"https://pillsapi.org/v2/everything?q=Agriculture
&sortBy=popularity&apiKey=7b40c2993a2640
08904439ac295309b0";

// News Section widget
class NewsSection extends StatefulWidget {
  @override
  _NewsSectionState createState() =>
_NewsSectionState();
}

class _NewsSectionState extends
State<NewsSection> {
  List<dynamic> farmingNews = [];

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

  // Fetch the pills from the API
  Future<void> fetchNews() async {
    final response = await
http.get(Uri.parse(pillsApiUrl));
    if (response.statusCode == 200) {
      setState(() {
        farmingNews =
json.decode(response.body)['articles']; // Adjust
based on the response structure
      });
    } else {
      // Handle the error if the request fails
      throw Exception('Failed to load pills');
    }
  }

  @override
  Widget build(BuildContext context) {
    return SingleChildScrollView( // Wrap the
column in a scroll view
      child: Padding(
        padding: EdgeInsets.symmetric(vertical:
20),
        child: Column(
          crossAxisAlignment:
CrossAxisAlignment.start,
          children: [
            Padding(
              padding: EdgeInsets.only(left: 20),
              child: Text(
                'Latest Farming News',
                style: TextStyle(fontSize: 24,
fontWeight: FontWeight.bold),
              ),
            ),
            SizedBox(height: 10),
            // Container with horizontal scrolling for
pills
            Container(
              height: 150, // Set height for horizontal
scroll
              child: ListView.builder(
                scrollDirection: Axis.horizontal,
                itemCount: farmingNews.length > 4 ?
4 : farmingNews.length, // Limit to 4 items
                itemBuilder: (context, index) {
                  return Padding(
                    padding: EdgeInsets.only(left: 10),
                    child: Card(
                      child: Container(
                        width: 200,
```

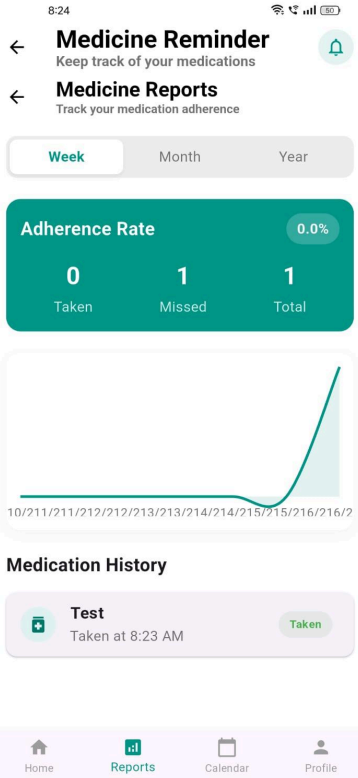
```
                padding: EdgeInsets.all(8),
                child: Column(
                 crossAxisAlignment:
CrossAxisAlignment.start,
                 children: [
                  Text(
                   farmingNews[index]['title'],
                   style: TextStyle(fontWeight:
FontWeight.bold),
                   maxLines: 2,
                   overflow:
TextOverflow.ellipsis,
                  ),
                  SizedBox(height: 10),
                  Text(

farmingNews[index]['description'] ?? 'No
description available.',
                   maxLines: 3,
                   overflow:
TextOverflow.ellipsis,
                  ),
                 ],
                ),
               ),
              ),
             );
            },
           ),
          ),
          // View All Button
          Padding(
           padding: EdgeInsets.only(left: 20, top:
10),
           child: TextButton(
            onPressed: () {
             // Navigate to the next screen to show
all the pills
              Navigator.push(
               context,
               MaterialPageRoute(
                builder: (context) =>
AllNewsScreen(pills: farmingNews),
               ),
              );
             },
             child: Text('View All'),
            ),
           ),
          ],
         ),
        ),
       );
      }
     }

     class AllNewsScreen extends StatelessWidget {
      final List<dynamic> pills;

      AllNewsScreen({required this.pills});

      @override
      Widget build(BuildContext context) {
       return Scaffold(
        appBar: AppBar(title: Text('All Farming
News')),
        body: ListView.builder(
         itemCount: pills.length,
         itemBuilder: (context, index) {
          return Card(
           margin: EdgeInsets.all(10),
           child: ListTile(
            title: Text(pills[index]['title']),
            subtitle: Text(pills[index]['description']
?? 'No description available.'),
           ),
          );
         },
        ),
       );
      }
     }
```

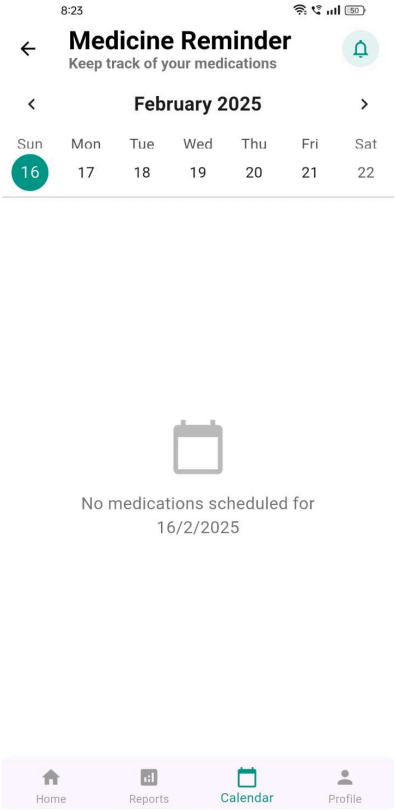
OUTPUT: -

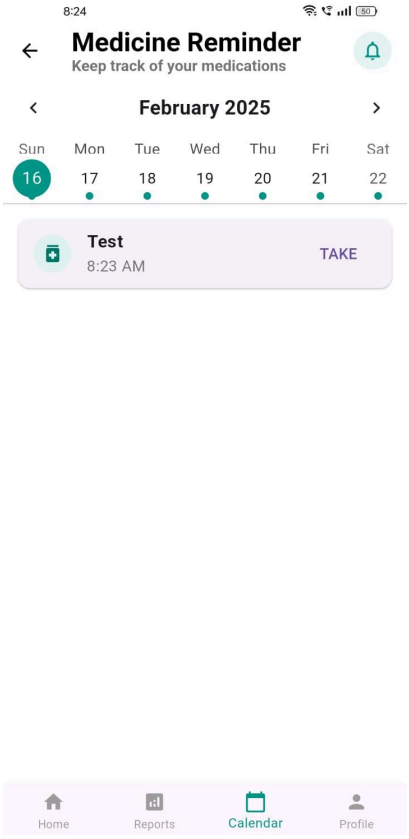

8:23
Medicine Reminder
Keep track of your medications
Upcoming Dose
No medicines scheduled
All Medicines
No medicines added yet
Home
Reports
Calendar
Profile


8:23
Add Medicine
Medicine Type
PILL
SYRUP
INJECTION
Medicine Name
Dosage (mg)
Duration
Start Date
16/2/2025
End Date
18/3/2025
Doses per Day
1
2
3
4
Days of Week
Mon
Tue
Wed
Thu
Fri
Sat
Sun
Reminder Times
Dose 1
8:23 AM


8:24
Medicine Reminder
Keep track of your medications
Upcoming Dose
8:23 AM
Test - 1mg
All Medicines
Test
1 mg
8:23 AM
Every day
Home
Reports
Calendar
Profile


8:24
Medicine Reminder
Keep track of your medications
Medicine Reports
Track your medication adherence
Week
Month
Year
Adherence Rate
0.0%
0
Taken
1
Missed
1
Total
Medication History
Test
Taken at 8:23 AM
Taken
Home
Reports
Calendar
Profile


8:23
Medicine Reminder
Keep track of your medications
February 2025
Sun
Mon
Tue
Wed
Thu
Fri
Sat
16
17
18
19
20
21
22
No medications scheduled for
16/2/2025
Home
Reports
Calendar
Profile


8:24
Medicine Reminder
Keep track of your medications
February 2025
Sun
Mon
Tue
Wed
Thu
Fri
Sat
16
17
18
19
20
21
22
Test
8:23 AM
TAKE
Home
Reports
Calendar
Profile

MAD & PWA Lab
Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	62
Name	Vedang V. Wajge
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

EXPERIMENT NO: - 03

Name:- Vedang Wajge **Class:-** D15A **Roll:No: -** 62

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

Theory: -

Flutter is a versatile open-source UI framework , which allows developers to build natively compiled applications for mobile, web, and desktop platforms from a single codebase. One of the key strengths of Flutter is its flexibility in creating highly customizable UIs. This practical focuses on incorporating essential visual elements—icons, images, and custom fonts—into a Flutter application. These elements enhance the visual appeal and usability of the app, providing an engaging experience for users.

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

Visual elements play a significant role in app development.

- **Enhanced User Experience**: Images and icons make your app visually appealing and user-friendly.
- **Information Conveyance**: They convey information quickly and intuitively. A well-chosen icon can replace lengthy text.
- **Branding**: Custom icons and images reinforce your app's branding, making it memorable.

Adding Icons in Flutter

Flutter provides built-in material design icons through the Icons class. Custom icons can also be added using third-party packages such as flutter_launcher_icons and font_awesome_flutter.

Icon(

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

Adding Images in Flutter

Flutter supports images from three sources:

1. **Assets** (Stored locally in the project)

   - Place the image inside the assets/images folder in the project.
   - Declare the image in pubspec.yaml

     ```
     flutter:
       assets:
         - assets/images/doctor1.png
     ```

   - Display the image in the app

     ```
     Image.asset('assets/images/sample.png');
     ```

2. **Network** (Fetched from the internet)

   Displaying images from the internet or network is very simple. Flutter provides a built-in method Image.network to work with images from a URL. The Image.network method also allows you to use some optional properties, such as height, width, color, fit, and many more.

   ```
   Image.network('https://example.com/sample.jpg');
   ```

3. **Memory or File** (Stored on the device)

Adding Custom Fonts in Flutter

By default, Flutter uses the Roboto font, but custom fonts can be added for a unique UI.

- Download the font and place it in the assets/fonts/ folder.
- Declare the font in pubspec.yaml
- Use the font in the app

```
Text(
  'Custom Font Example',
  style: TextStyle(fontFamily: 'CustomFont', fontSize: 24),
);
```

Code: -

support_page.dart

```
import 'dart:convert';
import 'package:flutter/services.dart';
import 'package:flutter/material.dart';
import 'crop_detail_page.dart';

class CropListPage extends StatelessWidget {
  Future<List<Map<String, dynamic>>> loadCrops() async {
    // Load the JSON file
    String jsonString = await rootBundle.loadString('assets/crops.json');
    List<dynamic> jsonResponse = json.decode(jsonString);
    return jsonResponse.map((crop) => crop as Map<String, dynamic>).toList();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Crop Practices')),
      body: FutureBuilder<List<Map<String, dynamic>>>(
        future: loadCrops(),  // Load crops data from JSON
        builder: (context, snapshot) {
          if (snapshot.connectionState == ConnectionState.waiting) {
            return Center(child: CircularProgressIndicator());
          } else if (snapshot.hasError) {
            return Center(child: Text('Error: ${snapshot.error}'));
          } else if (!snapshot.hasData || snapshot.data!.isEmpty) {
            return Center(child: Text('No crops available.'));
          } else {
            var crops = snapshot.data!;
            return GridView.builder(
              padding: EdgeInsets.all(10),
              gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
                crossAxisCount: 2,
                crossAxisSpacing: 10,
                mainAxisSpacing: 15,
                childAspectRatio: 1.2,
              ),
              itemCount: crops.length,
              itemBuilder: (context, index) {
                return GestureDetector(
                  onTap: () {
                    Navigator.push(
                      context,
                      MaterialPageRoute(
                        builder: (context) => CropDetailPage(crop: crops[index]),
                      ),
                    );
                  },
                  child: Column(
                    crossAxisAlignment: CrossAxisAlignment.center,
                    children: [
                      ClipRRect(
                        borderRadius: BorderRadius.circular(15),
                        child: Image.asset(
                          crops[index]["image"],
                          height: 130,
                          width: 180,
```

```
                  fit: BoxFit.cover,
                ),
              ),
              SizedBox(height: 5),
              Text(
                crops[index]["name"],
                style: TextStyle(fontSize: 16,
fontWeight: FontWeight.bold),
              ),
            ],
          ),
        );
      },
    );
  }
}
```

crop_details_page.dart

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

class CropDetailPage extends StatelessWidget {
  final Map<String, dynamic> crop;

  CropDetailPage({required this.crop});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text(crop["name"])),
      body: ListView(
        children: [
          Image.asset(
            crop["image"],
            height: 250,
            width: double.infinity,
            fit: BoxFit.cover,
          ),
          SizedBox(height: 20),
          buildListTile(context, "Introduction",
crop["introduction"]),
          buildListTile(context, "Climate",
crop["climate"]),
          buildListTile(context, "Soil",
crop["soil"]),
          buildListTile(context, "Sowing",
crop["sowing"]),
          buildListTile(context, "Land Preparation",
crop["land preparation"]),
          buildListTile(context, "Fertilizers",
crop["fertilizers"]),
          buildListTile(context, "Major Diseases
and their Management", crop["disease"]),
          buildListTile(context, "Harvesting",
crop["harvesting"]),
          buildListTile(context, "Yield",
crop["yield"]),
        ],
      ),
    );
  }

  Widget buildListTile(BuildContext context,
String label, String content) {
    return ListTile(
      title: Text(label),
      onTap: () {
        Navigator.push(
          context,
          MaterialPageRoute(
            builder: (context) => DetailsPage(label:
label, content: content),
          ),
        );
      },
    );
  }
}
```

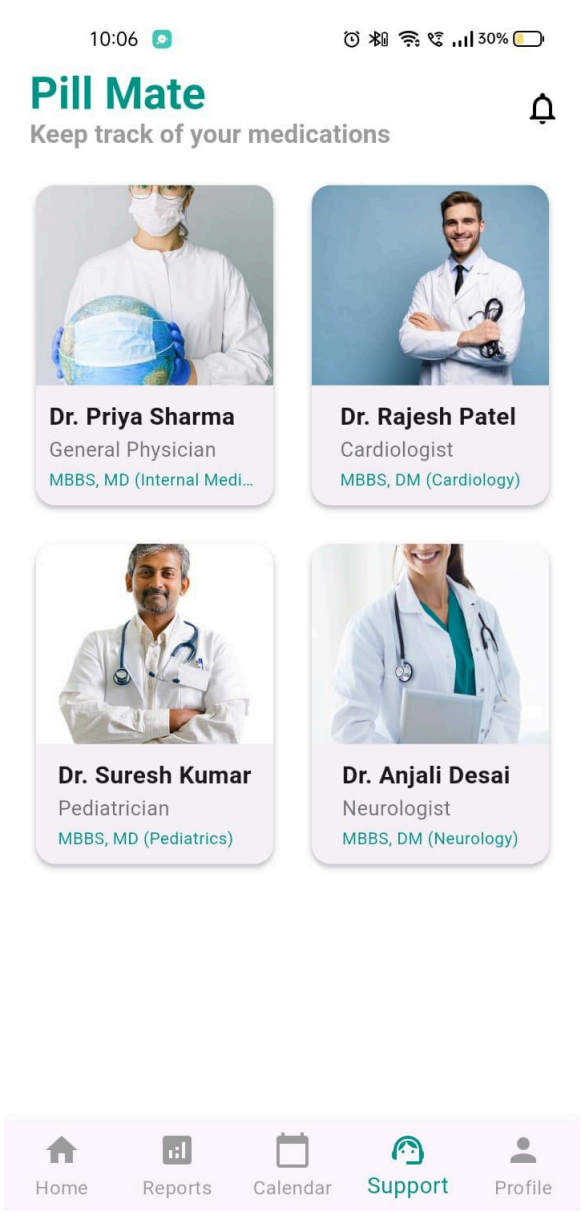
details_page.dart

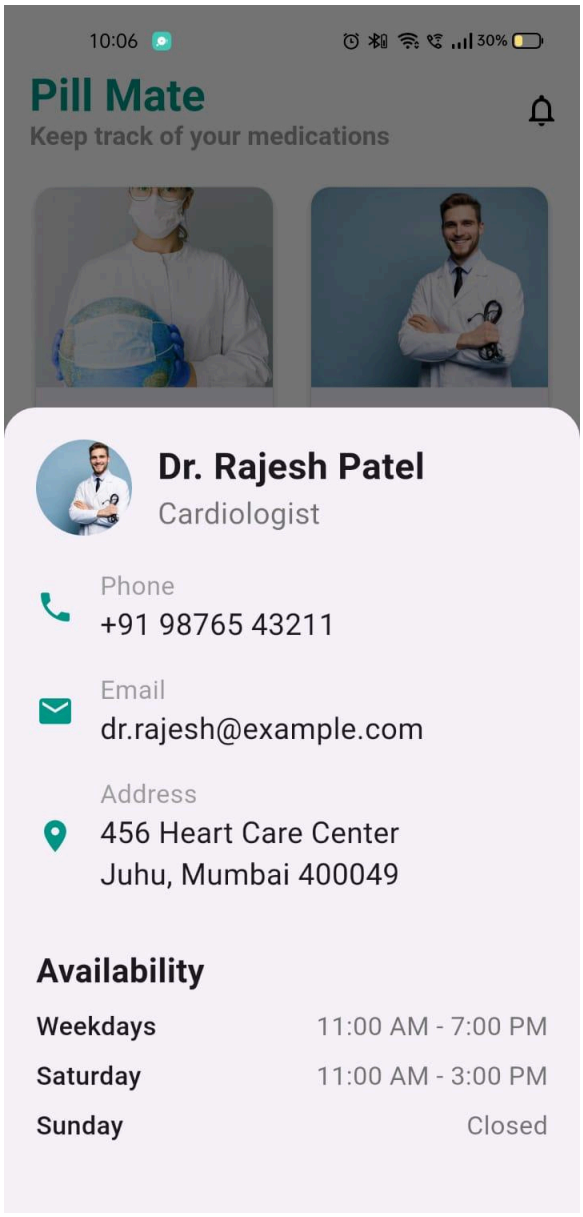
```
import 'package:flutter/material.dart';

class DetailsPage extends StatelessWidget {
  final String label;
  final String content;

  DetailsPage({required this.label, required
this.content});

  @override
  Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(title: Text('$label')),
     body: Padding(
       padding: const EdgeInsets.all(16.0),
       child: SingleChildScrollView(
         child: Text(
           content,
           style: TextStyle(fontSize: 18),
         ),
       ),
     ),
   );
  }
}
```

pubspec.yaml

```
flutter:
  assets:
    - assets/crops.json
    - assets/images/logo.png
    - assets/images/banner.jpg
    - assets/images/offers.png
    - assets/images/crop_tonics.png
    - assets/images/fertilizer.png
    - assets/images/pesticides.png
    - assets/images/delivery.png
    - assets/images/product1.png
    - assets/images/product2.png
    - assets/images/product3.png
    - assets/images/product4.png
    - assets/images/product5.png
    - assets/images/product6.png
    - assets/images/product7.png
    - assets/images/product8.png
    - assets/images/product9.png
    - assets/images/product10.png
    - assets/images/product11.png
    - assets/images/product12.png
    - assets/images/wheat.png
    - assets/images/paddy.png
    - assets/images/maize.jpg
    - assets/images/pearl_millet.png
    - assets/images/tomato.jpg
    - assets/images/cauliflower.jpg
    - assets/images/potato.jpg
    - assets/images/brinjal.jpg
    - assets/images/cabbage.jpg
    - assets/images/chilli.jpg
    - assets/images/onion.jpg
    - assets/images/coriander.jpg
    - assets/images/garlic.png
    - assets/images/sugarcane.png
```

OUTPUT: -


10:06
30%
Pill Mate
Keep track of your medications
Dr. Priya Sharma
General Physician
MBBS, MD (Internal Medi...
Dr. Rajesh Patel
Cardiologist
MBBS, DM (Cardiology)
Dr. Suresh Kumar
Pediatrician
MBBS, MD (Pediatrics)
Dr. Anjali Desai
Neurologist
MBBS, DM (Neurology)
Home
Reports
Calendar
Support
Profile


10:06
30%
Pill Mate
Keep track of your medications
Dr. Rajesh Patel
Cardiologist
Phone
+91 98765 43211
Email
dr.rajesh@example.com
Address
456 Heart Care Center
Juhu, Mumbai 400049
Availability
Weekdays
11:00 AM - 7:00 PM
Saturday
11:00 AM - 3:00 PM
Sunday
Closed

MAD & PWA Lab
Journal

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget
Roll No.	62
Name	Vedang V. Wajge
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 4

NAME: Vedang Wajge **ROLL NO:62** **CLASS:D15A**

AIM: To create an interactive Form using form widget

Theory:

1.Introduction

Forms are an essential part of any mobile application, allowing users to input and submit data. In Flutter, forms are managed using the Form widget along with form fields like TextFormField, DropdownButtonFormField, Radio, and Checkbox. Forms help in gathering user information, validating inputs, and processing data securely.

Flutter provides a simple yet powerful way to create and manage forms, ensuring a smooth user experience. This project demonstrates the implementation of **Login and Signup Forms** in Flutter, focusing on best practices for UI, validation, and navigation.

Creating an Interactive Form Using Form Widgets in Flutter:

Forms in Flutter are used to collect user input interactively. The Form widget acts as a container that holds multiple input fields and manages validation, submission, and state.

Key Components of a Flutter Form:

1. **Form Widget:**
    - This is the parent container that groups all input fields.
    - It requires a GlobalKey<FormState> to manage validation and submission.
2. **Form Fields:**
    - Flutter provides various widgets to capture user input, such as:
        - **Text Fields:** For text input like names and emails.
        - **Dropdowns:** To allow selection from a predefined list.
        - **Checkboxes & Switches:** For boolean choices (e.g., agree to terms).
        - **Radio Buttons:** To choose one option from multiple choices.
3. **Validation Mechanism:**
    - Each input field can have a validation function to check if the entered data is correct.
    - The form can be validated as a whole before submission.

4. **Managing State:**
    - Flutter allows state management within forms using controllers or stateful widgets.
    - The input values can be dynamically updated based on user actions.
5. **Handling Submission:**
    - A form should include a submit button that validates all fields before proceeding.
    - If validation passes, the data can be processed, stored, or sent to a server.

Steps to Create an Interactive Form:

1. **Define the Form:** Wrap all input fields inside a form container.
2. **Add Input Fields:** Use appropriate widgets for user input, such as text fields, dropdowns, or checkboxes.
3. **Implement Validation:** Ensure fields are correctly filled before submission.
4. **Manage User Input Dynamically:** Capture and store input values for further processing.
5. **Submit the Form:** Trigger validation and process the data when the user submits.

CODE:

SignUp Page:

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

class SignUpPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.teal.shade700,
      body: SafeArea(
        child: Column(
          children: [
            Spacer(),
            Text(
              'Create Account',
              style: TextStyle(fontSize: 26, color: Colors.white, fontWeight: FontWeight.bold),
            ),
            Spacer(),
            Container(
              width: double.infinity,
              padding: EdgeInsets.symmetric(horizontal: 20, vertical: 30),
              decoration: BoxDecoration(
                color: Colors.white,
```

```
            borderRadius: BorderRadius.only(topLeft: Radius.circular(30), topRight:
Radius.circular(30)),
            boxShadow: [BoxShadow(color: Colors.black12, spreadRadius: 1, blurRadius:
10)],
          ),
          child: Column(
            mainAxisSize: MainAxisSize.min,
            children: [
              _buildTextField("Full Name", Icons.person),
              SizedBox(height: 10),
              _buildTextField("Age", Icons.calendar_today, isNumber: true),
              SizedBox(height: 10),
              _buildTextField("Email", Icons.email),
              SizedBox(height: 10),
              _buildTextField("Password", Icons.lock, isPassword: true),
              SizedBox(height: 20),
              _buildButton(context, "Sign Up", Colors.teal, Colors.white, () {
                Navigator.pushReplacementNamed(context, '/home');
              }),
            ],
          ),
        ),
      ],
    ),
  ),
 );
}

 Widget _buildTextField(String label, IconData icon, {bool isPassword = false, bool
isNumber = false}) {
  return TextField(
    decoration: InputDecoration(
      labelText: label,
      prefixIcon: Icon(icon, color: Colors.teal),
      border: OutlineInputBorder(borderRadius: BorderRadius.circular(10)),
      filled: true,
      fillColor: Colors.grey.shade100,
    ),
    keyboardType: isNumber ? TextInputType.number : TextInputType.text,
    obscureText: isPassword,
  );
}
```

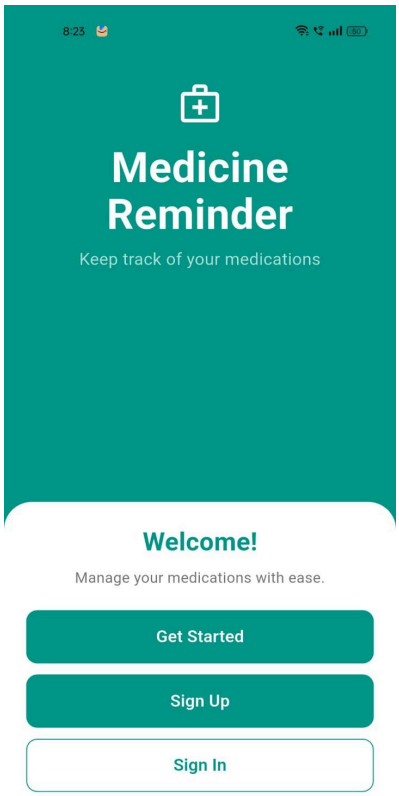
```
  // Button Widget
  Widget _buildButton(BuildContext context, String text, Color bgColor, Color textColor,
VoidCallback onPressed) {
    return SizedBox(
      width: double.infinity,
      child: ElevatedButton(
        onPressed: onPressed,
        style: ElevatedButton.styleFrom(
          backgroundColor: bgColor,
          padding: EdgeInsets.symmetric(vertical: 15),
          shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(10)),
        ),
        child: Text(text, style: TextStyle(fontSize: 18, color: textColor)),
      ),
    );
  }
}
```

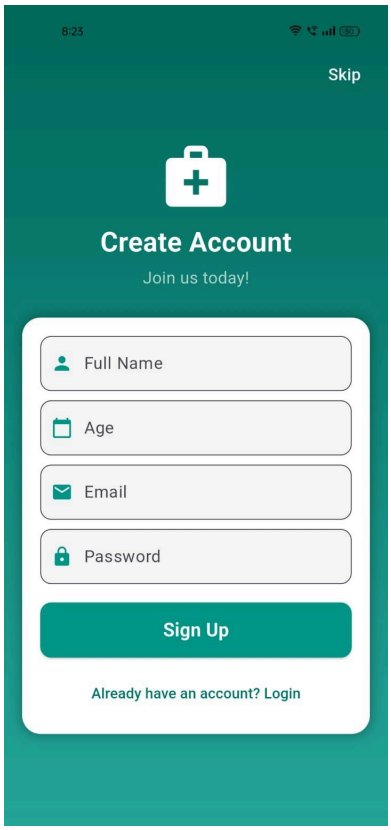
LoginIN Page:

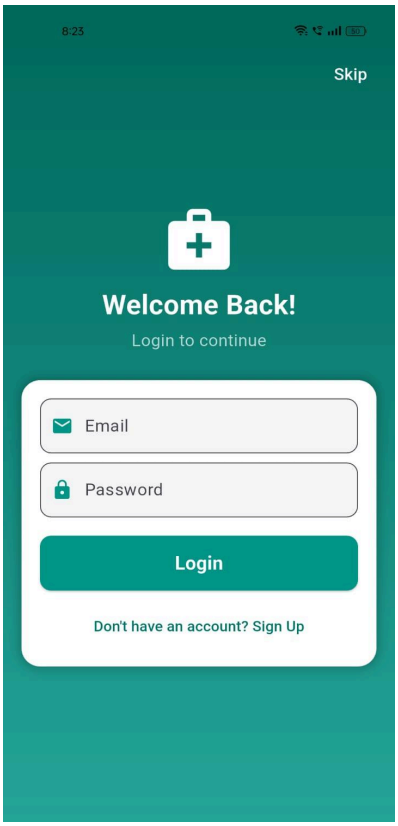
```
import 'package:flutter/material.dart';

class LoginPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.teal.shade700,
      body: SafeArea(
        child: Column(
          children: [
            Spacer(),
            Text(
              'Welcome Back to MindfulPlanner',
              style: TextStyle(fontSize: 26, color: Colors.white, fontWeight: FontWeight.bold),
            ),
            Spacer(),
            Container(
              width: double.infinity,
              padding: EdgeInsets.symmetric(horizontal: 20, vertical: 30),
              decoration: BoxDecoration(
                color: Colors.white,
                borderRadius: BorderRadius.only(topLeft: Radius.circular(30), topRight: Radius.circular(30)),
                boxShadow: [BoxShadow(color: Colors.black12, spreadRadius: 1, blurRadius: 10)],
              ),
              child: Column(
                mainAxisSize: MainAxisSize.min,
                children: [
                  _buildTextField("Email", Icons.email),
```

```
              SizedBox(height: 10),
              _buildTextField("Password", Icons.lock, isPassword: true),
              SizedBox(height: 20),
              _buildButton(context, "Login", Colors.teal, Colors.white, () {
                Navigator.pushReplacementNamed(context, '/home');
              }),
            ],
          ),
        ),
      ],
    ),
  ),
 );
}

Widget _buildTextField(String label, IconData icon, {bool isPassword = false}) {
 return TextField(
  decoration: InputDecoration(
   labelText: label,
   prefixIcon: Icon(icon, color: Colors.teal),
   border: OutlineInputBorder(borderRadius: BorderRadius.circular(10)),
   filled: true,
   fillColor: Colors.grey.shade100,
  ),
  obscureText: isPassword,
 );
}

Widget _buildButton(BuildContext context, String text, Color bgColor, Color textColor, VoidCallback
onPressed) {
 return SizedBox(
  width: double.infinity,
  child: ElevatedButton(
   onPressed: onPressed,
   style: ElevatedButton.styleFrom(backgroundColor: bgColor),
   child: Text(text, style: TextStyle(fontSize: 18, color: textColor)),
  ),
 );
}
}
```

OUTPUT:


8:23
Medicine Reminder
Keep track of your medications
Welcome!
Manage your medications with ease.
Get Started
Sign Up
Sign In


8:23
Skip
Create Account
Join us today!
Full Name
Age
Email
Password
Sign Up
Already have an account? Login


8:23
Skip
Welcome Back!
Login to continue
Email
Password
Login
Don't have an account? Sign Up

MAD & PWA Lab
Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	62
Name	Vedang V. Wajge
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

EXPERIMENT NO: - 05

Name:- Vedang Wajge **Class:-** D15A **Roll:No: -** 62

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

Theory: -

In Flutter, the screens and pages are known as routes, and these routes are just a widget. In Android, a route is similar to an Activity.

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

Gestures enable the app to respond to user interactions, making the application more dynamic and responsive.

Navigation and Routing in Flutter

Navigation is the process of moving between different screens or pages in an app. Flutter provides a simple and effective way to handle this through the use of the Navigator widget and routes.

1. Using Navigator Widget

The Navigator widget manages a stack of routes, allowing for pushing and popping routes on the stack.

- **Pushing a Route**: To navigate to a new screen, use Navigator.push().
- **Popping a Route**: To go back to the previous screen, use Navigator.pop().

```
ElevatedButton(
  onPressed: () {
    Navigator.push(
```

```
      context,
      MaterialPageRoute(builder: (context) => SecondScreen()),
    );},
);
```

2. **Named Routes**

   Flutter also allows the use of named routes to navigate, which can make the routing process cleaner, especially in larger applications.

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

Navigate to the route using Navigator.pushNamed()

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

Handling Gestures in Flutter

Gestures refer to user interactions with the app, such as taps, swipes, pinches, and drags. Flutter provides several widgets and gesture detectors to handle these interactions.

Tap Gestures

The most common gesture is the tap, which can be handled using the GestureDetector widget or specific buttons like InkWell or ElevatedButton.

Long Press Gesture

For long press gestures, Flutter provides the onLongPress callback in GestureDetector or InkWell.

Swipe and Drag Gestures

Flutter also provides swipe and drag gesture handling. The onHorizontalDragUpdate and onVerticalDragUpdate callbacks are used for dragging gestures.

Code: -

Welcome_page.dart

main.dart

```
import 'package:flutter/material.dart';
import 'pages/login_page.dart';
import 'pages/register_page.dart';
import 'pages/otp_verification_page.dart';
import 'pages/myaccountpage.dart';
import 'pages/home_page.dart';


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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'AgriApp',
      theme: ThemeData(
        primarySwatch: Colors.green,
        colorScheme: ColorScheme.fromSeed(seedColor:
Color(0xFF6A9A5B)),
      ),
      initialRoute: '/login', // Set initial page
      routes: {
        '/login': (context) => LoginPage(),
        '/register': (context) => RegistrationPage(),
        '/otp': (context) => OtpVerificationPage(),
        '/myaccount': (context) => MyAccountPage(),
        '/home': (context) => HomePage(),
      },
    );
  }
}
```

Login_page.dart

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

class LoginPage extends StatefulWidget {
  @override
  _LoginPageState createState() =>
_LoginPageState();
}

class _LoginPageState extends
State<LoginPage> {
  final _formKey = GlobalKey<FormState>();

  @override
  Widget build(BuildContext context) {
   return Scaffold(
    backgroundColor: Colors.white,
    body: SafeArea(
     child: SingleChildScrollView(
      child: Padding(
       padding: const EdgeInsets.all(16.0),
       child: Column(
        crossAxisAlignment:
CrossAxisAlignment.stretch,
        children: [
         SizedBox(height: 50),
         // Logo
         Center(
          child: Image.asset(
           'assets/images/logo.png',
           height: 150,
          ),
         ),
         SizedBox(height: 16),
         // Title
         Text(
          'AgriApp: The mix of Agriculture &
Smart, Scientific, Sustainable, Modern
Technology Methods for Precision Farming.',
          textAlign: TextAlign.center,
          style: TextStyle(
           fontSize: 15,
           color: Colors.black,
          ),
         ),
         SizedBox(height: 32),
         // Form
         Form(
          key: _formKey,
          child: Column(
           children: [
            _buildTextField(
             label: 'Email',
             hint: 'Enter your email',
             icon: Icons.email,
             validator: (value) {
              if (value == null ||
value.isEmpty) {
               return 'Email is required';
              } else if
(!RegExp(r'^[^@]+@[^@]+\.[^@]+').hasMatch(
value)) {
               return 'Enter a valid email
address';
              }
              return null;
             },
            ),
            SizedBox(height: 16),
            _buildTextField(
             label: 'Password',
             hint: 'Enter your password',
             icon: Icons.lock,
             isPassword: true,
             validator: (value) {
              if (value == null ||
value.isEmpty) {
               return 'Password is required';
              } else if (value.length < 6) {
               return 'Password must be at
least 6 characters';
              }
              return null;
```

```
                },
              ),
            ],
          ),
        ),
        SizedBox(height: 32),
        // Login Button
        ElevatedButton(
          onPressed: () {
            if
(_formKey.currentState!.validate()) {

Navigator.pushReplacementNamed(context,
'/home');
            }
          },
          style: ElevatedButton.styleFrom(
            backgroundColor: Colors.green,
            minimumSize: Size(double.infinity,
50),
            shape: RoundedRectangleBorder(
              borderRadius:
BorderRadius.circular(8),
            ),
          ),
          child: Text(
            'Login',
            style: TextStyle(fontSize: 18, color:
Colors.white),
          ),
        ),
        SizedBox(height: 16),
        // Registration link
        Center(
          child: TextButton(
            onPressed: () {
              Navigator.pushNamed(context,
'/register'); // Add registration route later
            },
            child: Text(
              'Don’t have an account? Register',
              style: TextStyle(color:
Colors.green), // Green color applied
            ),
          ),
        ),
      ],
```

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

Widget _buildTextField({
  required String label,
  required String hint,
  required IconData icon,
  bool isPassword = false,
  required String? Function(String?) validator,
}) {
  return TextFormField(
    obscureText: isPassword,
    decoration: InputDecoration(
      labelText: label,
      hintText: hint,
      prefixIcon: Icon(icon, color: Colors.green),
      border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(8),
      ),
      focusedBorder: OutlineInputBorder(
        borderSide: BorderSide(color:
Colors.green), // Green color applied
        borderRadius: BorderRadius.circular(8),
      ),
    ),
    validator: validator,
  );
}
}
```

add_medicine.dart

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

class PillMate extends StatefulWidget {
  @override
  _PillMateState createState() =>
_PillMateState();
}

class _PillMateState extends State<PillMate> {
  final _formKey = GlobalKey<FormState>();
  final TextEditingController _cityController =
TextEditingController();

  String? _city;
  String? _address;
  String? _updatedAt;
  String? _status;
  String? _temp;
  String? _tempMin;
  String? _tempMax;
  String? _windSpeed;
  String? _pressure;
  String? _humidity;
  String? _sunrise;
  String? _sunset;
  bool _isLoading = false;
  bool _isError = false;
  bool _isDataFetched = false; // Add this flag to
control the visibility of weather details

  Future<void> _fetchWeather() async {
    setState(() {
      _isLoading = true;
      _isError = false;
      _isDataFetched = false; // Reset this flag
when fetching new data
    });

    final response = await http.get(Uri.parse(

'https://api.openweathermap.org/data/2.5/weathe
r?q=$_city&units=metric&appid=73cbebdd0322
acd49bda6ede059b2b18'));

    if (response.statusCode == 200) {
      final data = jsonDecode(response.body);

      setState(() {
        _address = '${data['name']},
${data['sys']['country']}';
        _updatedAt = 'Updated At:
${DateTime.fromMillisecondsSinceEpoch(data['
dt'] * 1000).toString()}';
        _status =
data['weather'][0]['description'].toUpperCase();
        _temp = '${data['main']['temp']}°C';
        _tempMin = 'Min Temp:
${data['main']['temp_min']}°C';
        _tempMax = 'Max Temp:
${data['main']['temp_max']}°C';
        _pressure = 'Pressure:
${data['main']['pressure']} hPa';
        _humidity = 'Humidity:
${data['main']['humidity']}%';
        _windSpeed = 'Wind Speed:
${data['wind']['speed']} m/s';
        _sunrise =
DateTime.fromMillisecondsSinceEpoch(data['sy
s']['sunrise'] * 1000).toString();
        _sunset =
DateTime.fromMillisecondsSinceEpoch(data['sy
s']['sunset'] * 1000).toString();
        _isLoading = false;
        _isDataFetched = true; // Set the flag to true
after data is fetched
      });
    } else {
      setState(() {
        _isLoading = false;
        _isError = true;
        _isDataFetched = false; // Reset the flag if
```

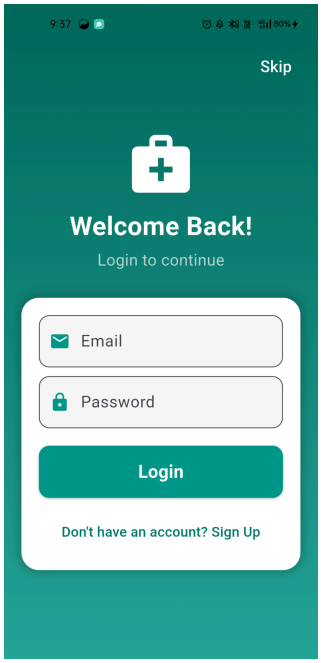
there is an error

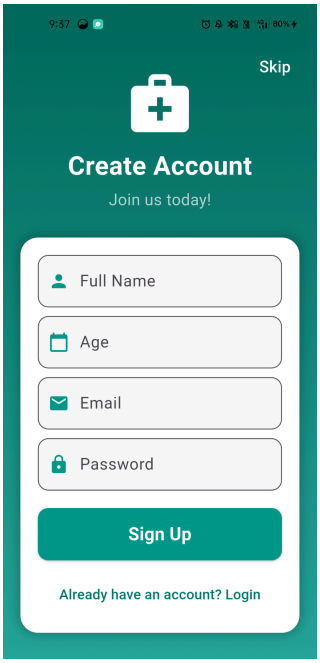
```
    });
  }
 }

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: Text('Weather Forecasting'),
    //backgroundColor: Colors.white,
   ),
   //backgroundColor: Colors.white,
   body: SafeArea(
    child: SingleChildScrollView(
     child: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Column(
       crossAxisAlignment:
CrossAxisAlignment.stretch,
       children: [
        SizedBox(height: 50),
        // Logo
        Center(
         child: Image.asset(
          'assets/images/logo.png', // Replace
with your logo
          height: 150,
         ),
        ),
        SizedBox(height: 16),
        // Title
        Text(
         'Weather Forecasting',
         textAlign: TextAlign.center,
         style: TextStyle(
          fontSize: 18,
          color: Colors.black,
         ),
        ),
        SizedBox(height: 32),
        // Form
        Form(
         key: _formKey,
         child: Column(
          children: [
           TextFormField(
            controller: _cityController,
            decoration: InputDecoration(
             labelText: 'City',
             hintText: 'Enter the city name',
             prefixIcon:
Icon(Icons.location_city, color: Colors.green),
             border: OutlineInputBorder(
              borderRadius:
BorderRadius.circular(8),
             ),
             focusedBorder:
OutlineInputBorder(
              borderSide: BorderSide(color:
Colors.green),
              borderRadius:
BorderRadius.circular(8),
             ),
            ),
            validator: (value) {
             if (value == null ||
value.isEmpty) {
              return 'City is required';
             }
             return null;
            },
           ),
           SizedBox(height: 16),
           ElevatedButton(
            onPressed: () {
             if
(_formKey.currentState!.validate()) {
              setState(() {
               _city = _cityController.text;
              });
              _fetchWeather();
             }
            },
            style: ElevatedButton.styleFrom(
             backgroundColor: Colors.green,
             minimumSize:
Size(double.infinity, 50),
             shape:
RoundedRectangleBorder(
              borderRadius:
BorderRadius.circular(8),
             ),
            ),
```

```
                child: Text(
                  'Get Weather',
                  style: TextStyle(fontSize: 18,
color: Colors.white),
                ),
              ),
            ],
          ),
        ),
        SizedBox(height: 32),
        _isLoading
            ? CircularProgressIndicator()
            : _isError
            ? Text(
          'Error fetching weather data.',
          style: TextStyle(color: Colors.red),
          textAlign: TextAlign.center,
        )
            : _isDataFetched
            ? Column(
          crossAxisAlignment:
CrossAxisAlignment.start,
          children: [
            Text(
              'Location: $_address',
              style: TextStyle(fontSize: 16,
fontWeight: FontWeight.bold),
            ),
            SizedBox(height: 8),
            Text('$_updatedAt'),
            SizedBox(height: 16),
            Text('Status: $_status'),
            SizedBox(height: 8),
            Text('Temperature: $_temp'),
            SizedBox(height: 8),
            Text('$_tempMin'),
            SizedBox(height: 8),
            Text('$_tempMax'),
            SizedBox(height: 8),
            Text('$_windSpeed'),
            SizedBox(height: 8),
            Text('$_pressure'),
            SizedBox(height: 8),
            Text('$_humidity'),
            SizedBox(height: 8),
            Text('Sunrise: $_sunrise'),
            SizedBox(height: 8),
            Text('Sunset: $_sunset'),
          ],
        )
            : Container(),
      ],
    ),
  ),
  ),
  ),
  );
 }
}
```

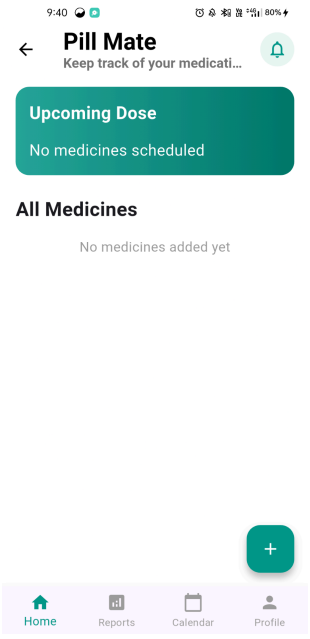
OUTPUT: -

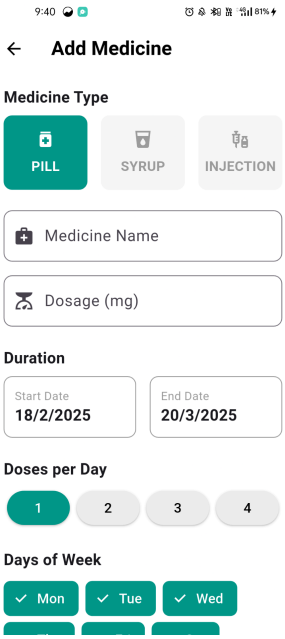
After clicking on Don’t have an account? it navigates to the registration page.





In home page, after clicking on “+” icon it navigates to the add medicine page.





MAD & PWA Lab
Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	62
Name	Vedang V. Wajge
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: - 06

Name:- Vedang Wajge **Class:-** D15A **Roll:No: -** 62

AIM: - To connect Flutter UI with Firebase database.

Theory: -

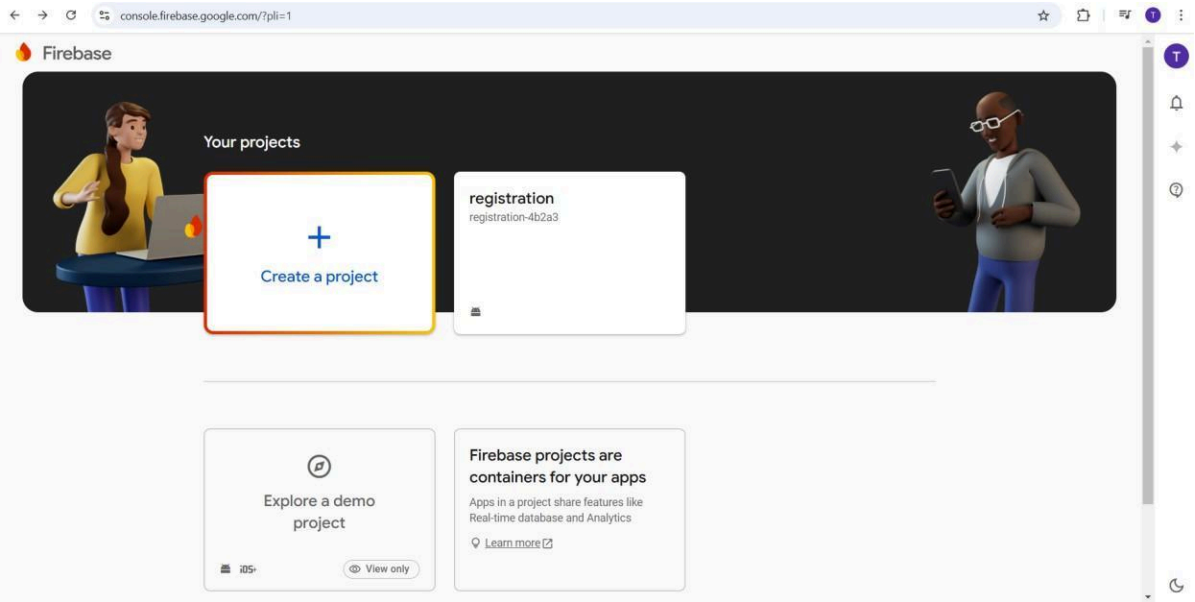
Flutter is an open-source UI toolkit developed by Google for building natively compiled applications for mobile, web, and desktop from a single codebase. Firebase, a Backend-as-a-Service (BaaS) platform, provides real-time database, authentication, and cloud storage services, making it a powerful backend solution for Flutter applications.

By integrating Firebase with Flutter, developers can store and retrieve data in real time, authenticate users, and manage cloud-based data efficiently. This is particularly useful for applications requiring dynamic content updates and user interactions.

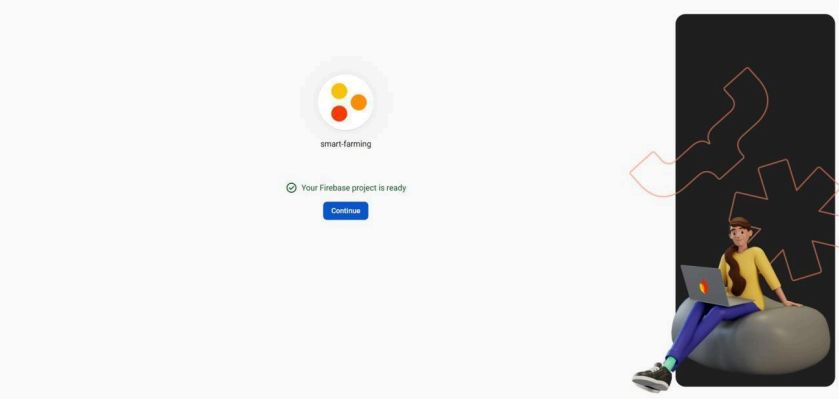
- **Steps to Connect Flutter UI with Firebase Database**

Step 1:

1.1) Go to Firebase Console and Create a Firebase Project

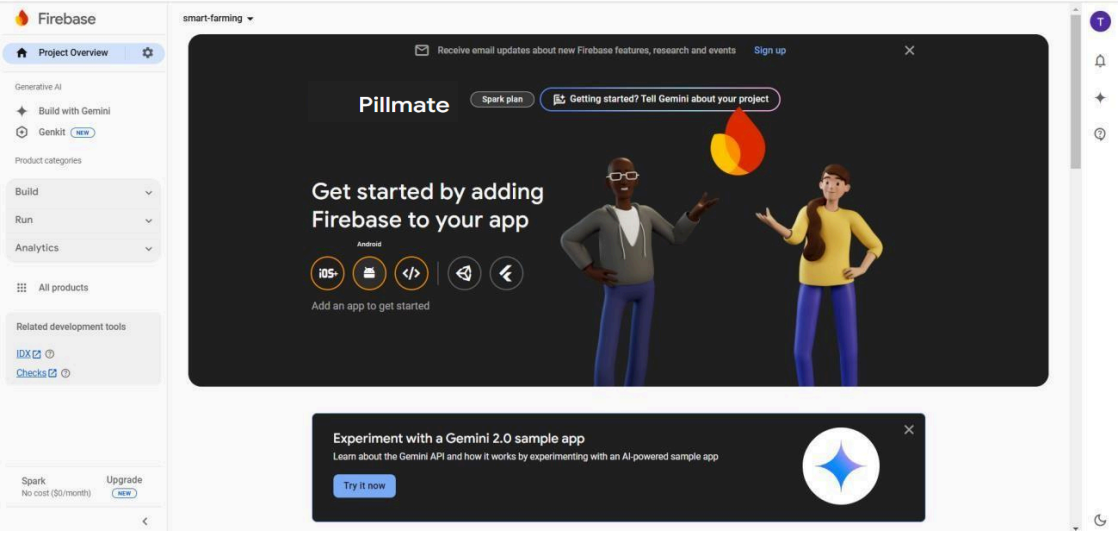


1.2) Click on Create a Project and give it a suitable name and enable Google Analytics (optional) & Click continue and complete the setup

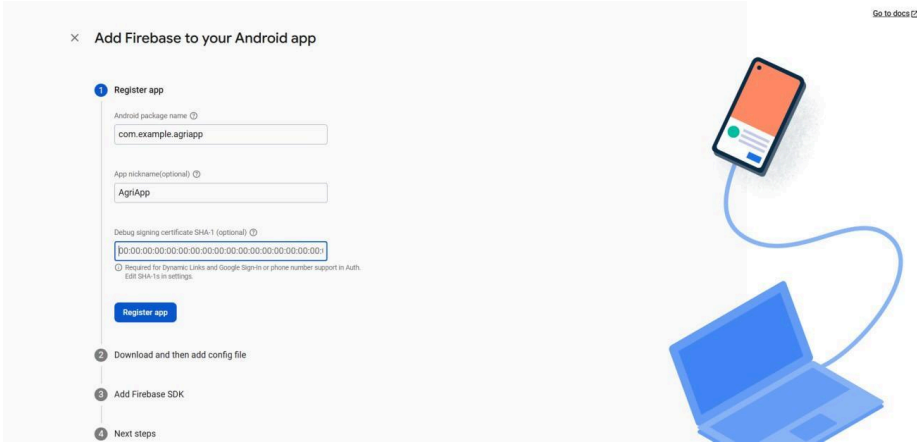


Step 2:- Add Firebase to Your Flutter App

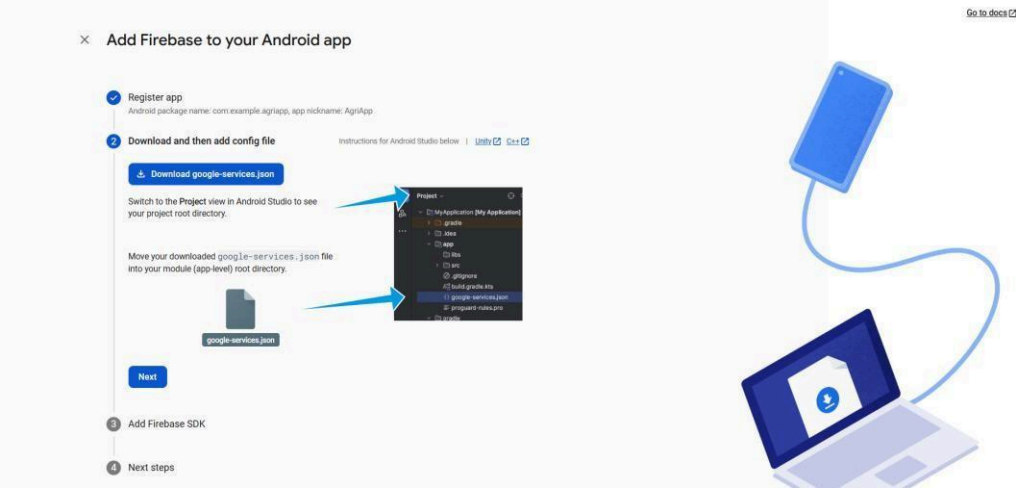
2.1) Click on Android/iOS/Web based on your Flutter application

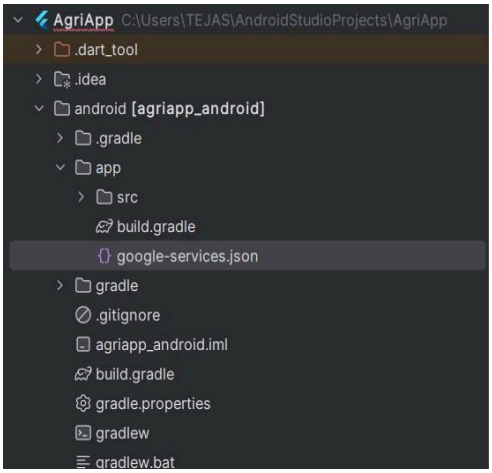


2.2) Register your app with a unique package name (found in android/app/build.gradle for Android).

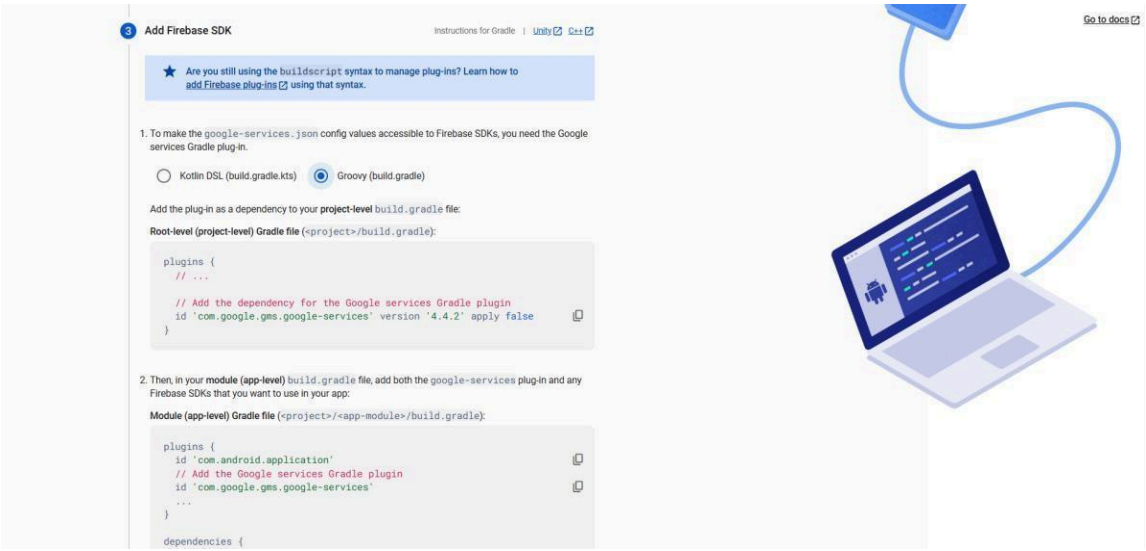


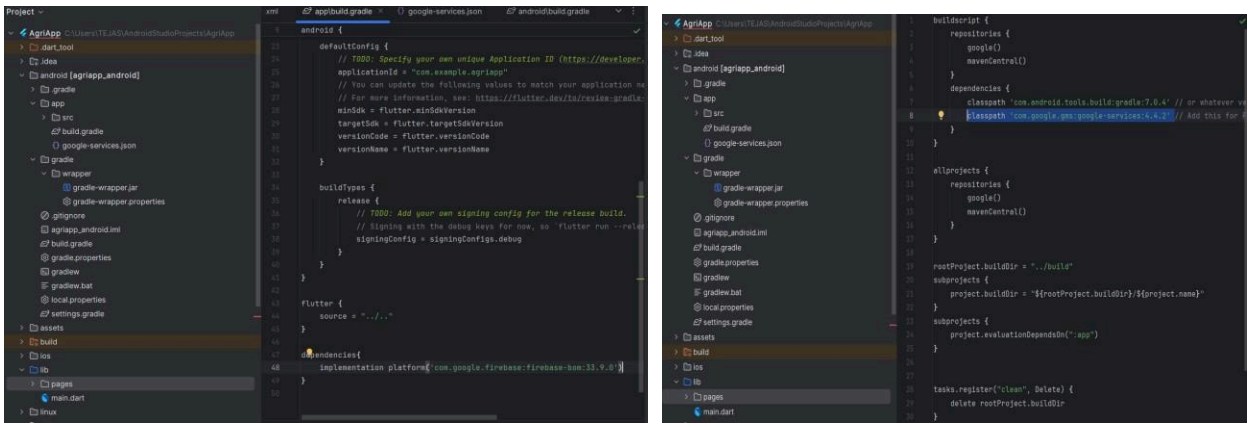
2.3) Download the google-services.json (for Android) & place the JSON file inside android/app/ directory.





2.4) Add Firebase SDK dependencies to android/build.gradle



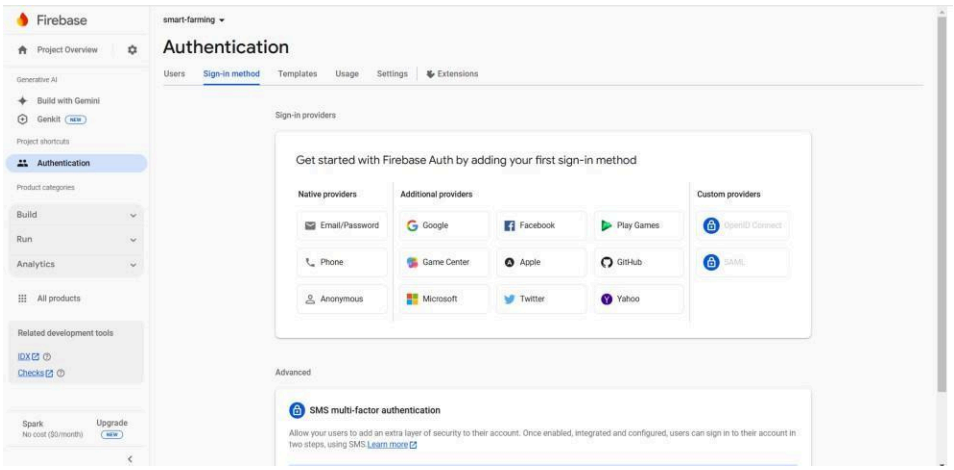


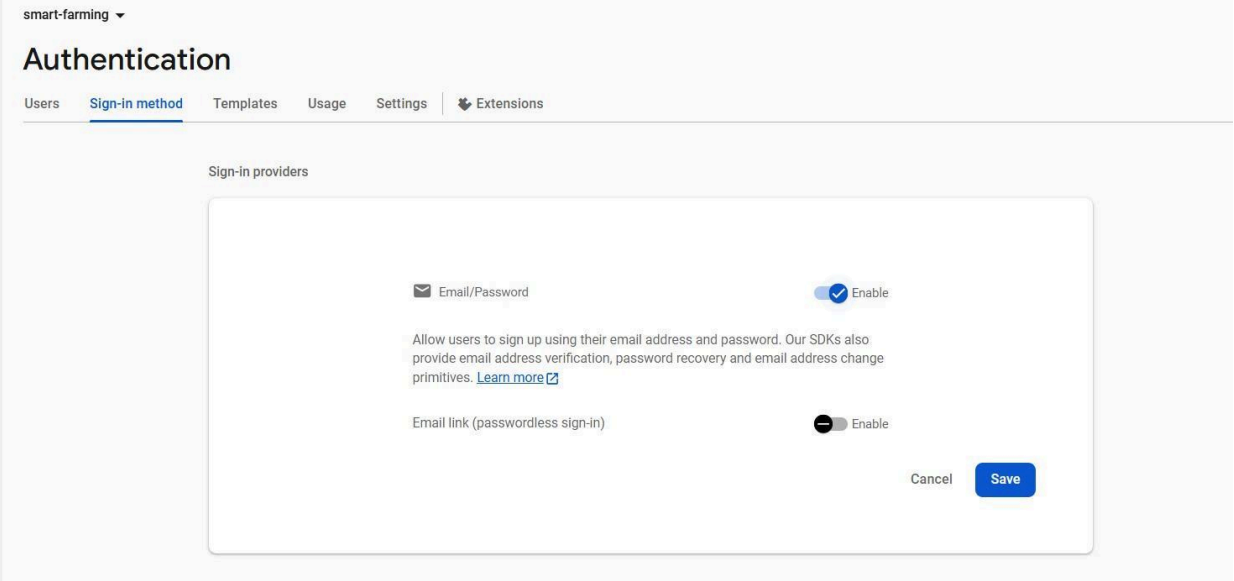
Step 3: - Add Firebase Authentication to Your App

3.1) Add Firebase Authentication Dependencies

```yaml
dependencies:
  flutter:
    sdk: flutter
  firebase_core: ^3.11.0
  firebase_auth: ^5.4.2  # For authentication
  cloud_firestore: ^5.6.3  # For Firestore, if you need it
  firebase_messaging: ^15.2.2
  http: ^0.13.3
  image_picker: ^1.0.4
  tflite_flutter: ^0.11.0
  image: ^3.2.0
  url_launcher: ^6.1.14
```

3.2) Enable Authentication in Firebase Console
Go to **Firebase Console →**
Authentication.
Click on **Sign-in method** and enable **Email/Password** (or any other method like Google).
Click Save





3.3) Implement Authentication in Flutter Modify main.dart

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

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

Step 4: -Configure Firebase Realtime Database

4.1) Go to Firebase Console → Realtime Database.
4.2) Click **Create Database** → Choose location → Set rules (for development, set read/write to true).
4.3) Click **Publish**.

Code:-

Register_page.dart

```
class _RegistrationPageState extends
State<RegistrationPage> {
  final _formKey =
GlobalKey<FormState>(); bool
_isLoading = false; String
_name = '';
  String _email = ''; String
  _state = ''; String
  _district = ''; String
  _password = ''; String
  _phone = '';

  final FirebaseAuth _auth =
FirebaseAuth.instance;

  // Firebase Registration
  Future<void> _handleRegistration() async
{
    if (_formKey.currentState!.validate()) {
      setState(() {
        _isLoading = true;
      });

      try {
        // Register user with Firebase
Authentication (Email and Password)
        UserCredential userCredential = await
_auth.createUserWithEmailAndPassword(
          email: _email,
          password: _password,
        );

        // Get the user ID from Firebase
Authentication user
        String userId =
userCredential.user!.uid;

        // Store extra user information in
Firestore
        await
FirebaseFirestore.instance.collection('users')
.doc(userId).set({
          'name': _name,
          'email': _email,
          'phone':
          _phone, 'state':
          _state, 'district':
          _district,
          // You can add other fields here
        });

        setState(() {
          _isLoading = false;
        });

        // Navigate to OTP verification
page after successful registration
        Navigator.pushReplacementNamed(
          context,
          '/otp',
          arguments: _phone, // Pass the phone
number here
        );
      } catch (e) {
        setState(() {
          _isLoading = false;
        });

ScaffoldMessenger.of(context).showSnack
B ar(
          SnackBar(content: Text("Registration
failed: $e")),
        );
    }
  }
}
```

login_page.dart

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

class LoginPage extends StatefulWidget {
  @override
  _LoginPageState createState() =>
_LoginPageState();
}




class _LoginPageState extends
State<LoginPage> {
  final _formKey =
GlobalKey<FormState>(); final
FirebaseAuth _auth =
FirebaseAuth.instance;

  TextEditingController _emailController =
TextEditingController();
TextEditingController _passwordController
= TextEditingController();

  bool _isLoading = false;
  String _errorMessage = '';

  Future<void> _loginUser() async {
    if (!_formKey.currentState!.validate())
return;

    setState(() {
      _isLoading = true;
      _errorMessage = '';
    });

    try {
      UserCredential userCredential = await
_auth.signInWithEmailAndPassword( email:
        _emailController.text.trim(), password:
_passwordController.text.trim(),
      );
      User? user =

      userCredential.user; if (user !=

      null) {

  Navigator.pushReplacementNamed(context
, '/home');
      } else {
        setState(
        () {
          _errorMessage = "Something went
  wrong. Please try again.";
        });
      }
    } on FirebaseAuthException catch (e) {
      setState(() {
        _errorMessage = e.message ?? "An
error occurred. Please try again.";
      });
    }

    setState(() {
      _isLoading = false;
    });
  }
```

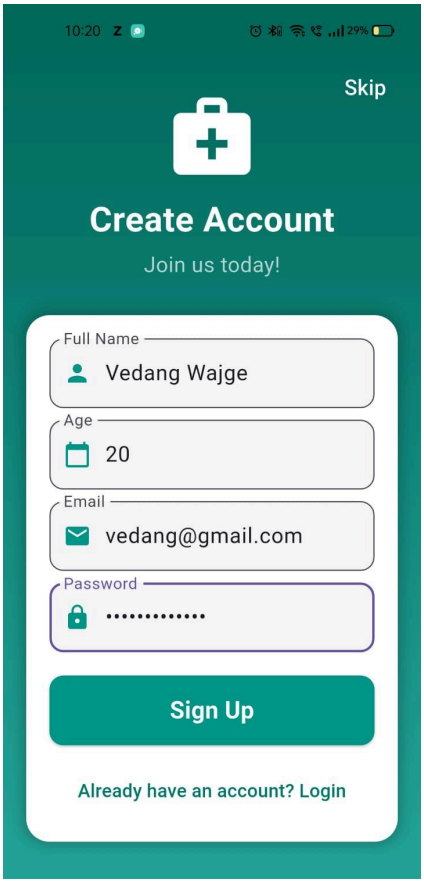
edit_profile.dart

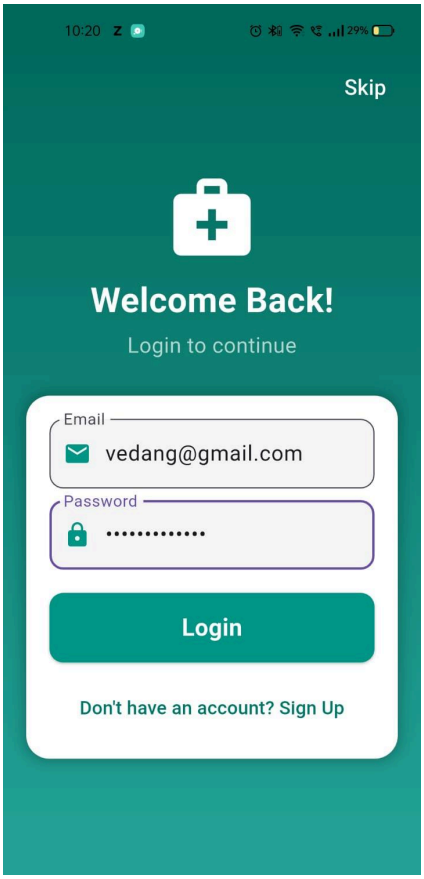
```
import
'package:cloud_firestore/cloud_firestore.dart
';
import
'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';

class MyAccountPage extends StatefulWidget
{
 @override
 _MyAccountPageState createState() =>
_MyAccountPageState();
}

class _MyAccountPageState extends
State<MyAccountPage> {
 final _formKey =
GlobalKey<FormState>(); String
name = '';
 String phone = '';
 String email = '';
 String state = '';
 String language = 'English';
 List<String> myCrops = [];
 bool isLoading = true;

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

 // Fetch data from Firestore
 _fetchUserData() async { User?
  user =
FirebaseAuth.instance.currentUser;

  if (user != null) { try
   {
    DocumentSnapshot doc = await
FirebaseFirestore.instance
        .collection('users')
        .doc(user.uid)
        .get();

      if (doc.exists) {
       setState(() {
        name = doc['name'];
        phone = doc['phone'];
        email = doc['email'];
        state = doc['state'];
        language = doc['language'];
        myCrops =
List<String>.from(doc['myCrops']);
        isLoading = false;
       });
      }
    } catch (e) {
      print("Error fetching user data: $e");
      setState(() {
       isLoading = false;
      });
    }
  }
 }

 // Save updated data to Firestore
 _saveUserData() async {
  User? user =
FirebaseAuth.instance.currentUser;

  if (user != null) {
   try {
    await
FirebaseFirestore.instance.collection('users')
.doc(user.uid).update({
       'name': name,
       'phone': phone,
       'email': email,
       'state': state,
       'language':
       language,
       'myCrops':
       myCrops,
      });
```

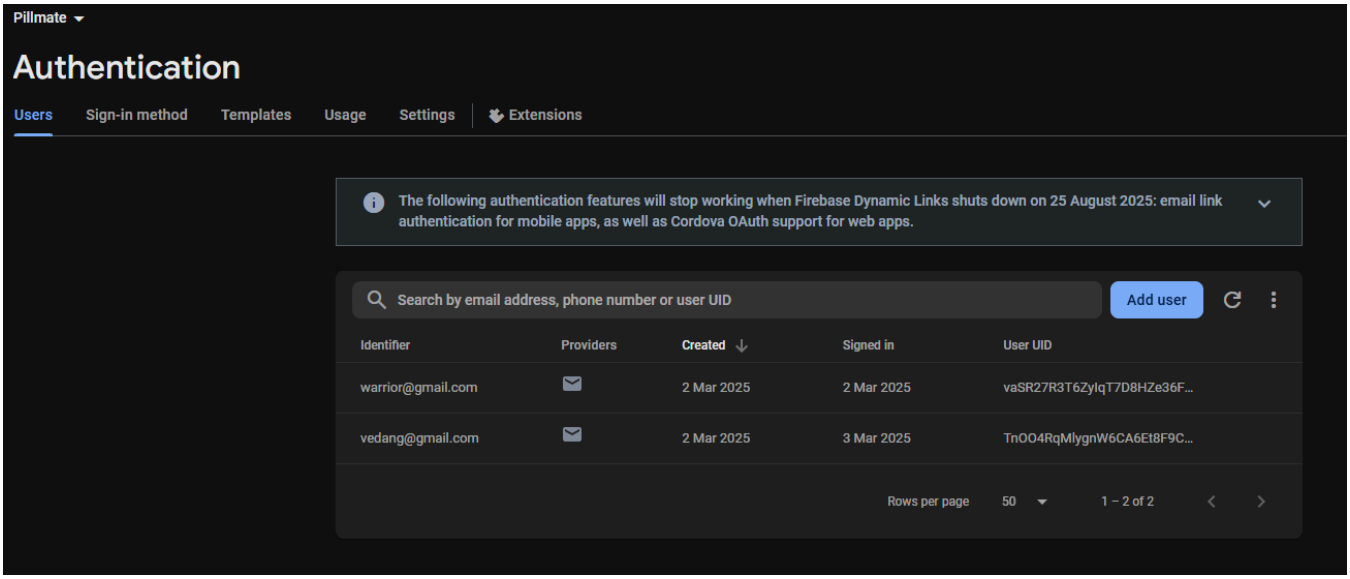
```
      // Show success message

ScaffoldMessenger.of(context).showSnackB
ar(SnackBar(content: Text('Changes
saved!')));
    } catch (e) {
      print("Error saving user data: $e");

ScaffoldMessenger.of(context).showSnackB
ar(SnackBar(content: Text('Error saving
changes')));
    }



  }
 }
```

Main.dart

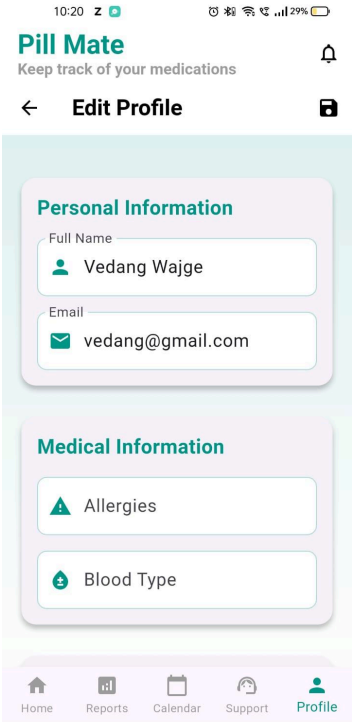
```
void main() async {
 WidgetsFlutterBinding.ensureInitialized();
 await Firebase.initializeApp();
 runApp(MyApp());
}

class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) { return
  MaterialApp(
  debugShowCheckedModeBanner: false,
  title: 'AgriApp',
    theme: ThemeData( primarySwatch:
      Colors.green, colorScheme:
ColorScheme.fromSeed(seedColor:
Color(0xFF6A9A5B)),
    ),
    home:
FirebaseAuth.instance.currentUser == null ?
LoginPage() : MainScreen(),
    routes: {
      '/login': (context) => LoginPage(),
      '/register': (context) =>
RegistrationPage(),
      '/otp': (context)
      =>
OtpVerificationPa
       ge(
       phoneNumb
       er:
ModalRoute.of(context)!.settings.argument
s as String,
      ),
      '/home': (context) => MainScreen(),
    },
   );
  }
 }

class MainScreen extends StatefulWidget {
 @override
 _MainScreenState createState() =>
  _MainScreenState();
 }

 class _MainScreenState extends
 State<MainScreen> {
  int _currentIndex = 0;

  final List<Widget> _pages
   = [ HomePage(),
   WeatherPage(),
   DiseaseDetectionPage(),
   CropListPage(),
   MyAccountPage(),
  ];
```







After Registering, the user details is saved in the database.



User details get fetched from the database in the profile page.

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.	62
Name	Vedang V. Wajge
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	

Experiment No. 7

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

Theory:

- **Regular Web App**

  A regular web app is a 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:**

  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.

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.

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.

- **The main features are:**

1. Progressive — They work for every user, regardless of the browser chosen because they are built at the base with progressive improvement principles.
2. Responsive — They adapt to the various screen sizes: desktop, mobile, tablet, or dimensions that can later become available.
3. Updated — Information is always up-to-date thanks to the data update process offered by service workers.
4. Secure — Exposed over HTTPS protocol to prevent the connection from displaying information or altering the contents.
5. Searchable — They are identified as “applications” and are indexed by search engines.
6. Reactivable — Make it easy to reactivate the application thanks to capabilities such as web notifications.
7. 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.
8. Linkable — Easily shared via URL without complex installations.
9. 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.

Code:

1. **Manifest.json**

```
{
    "name": "Simple PWA",
    "short_name": "PWA",
    "start_url": "index.html",
    "display": "standalone",
    "background_color": "#ffffff",
    "theme_color": "#000000",
    "description": "A simple Progressive Web App that explains what a PWA is.",
    "icons": [
        {
            "src": "pwa-banner.png",
            "sizes": "192x192",
            "type": "image/png"
        }
    ]
}
```

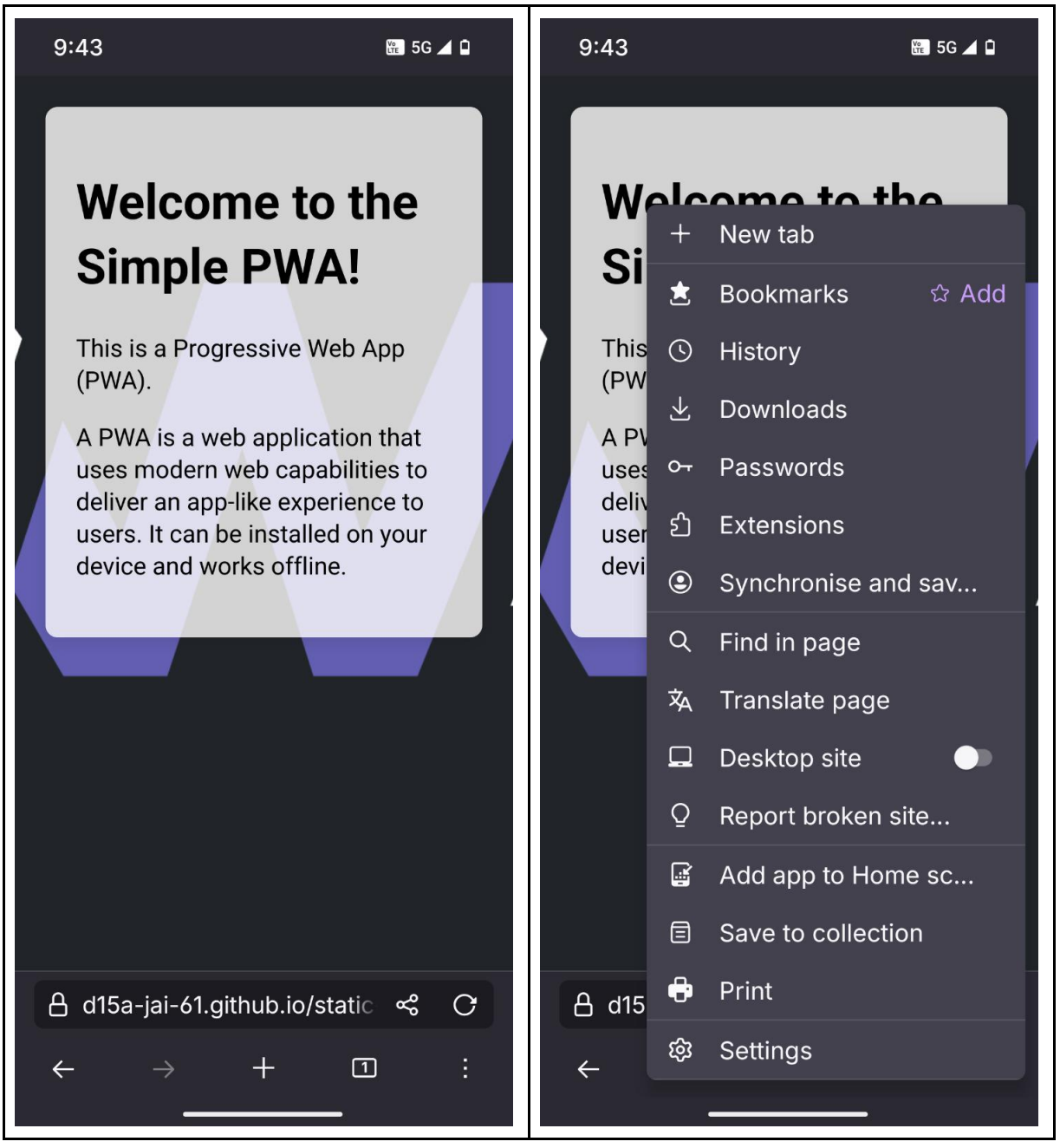
2. **Service-worker.js**

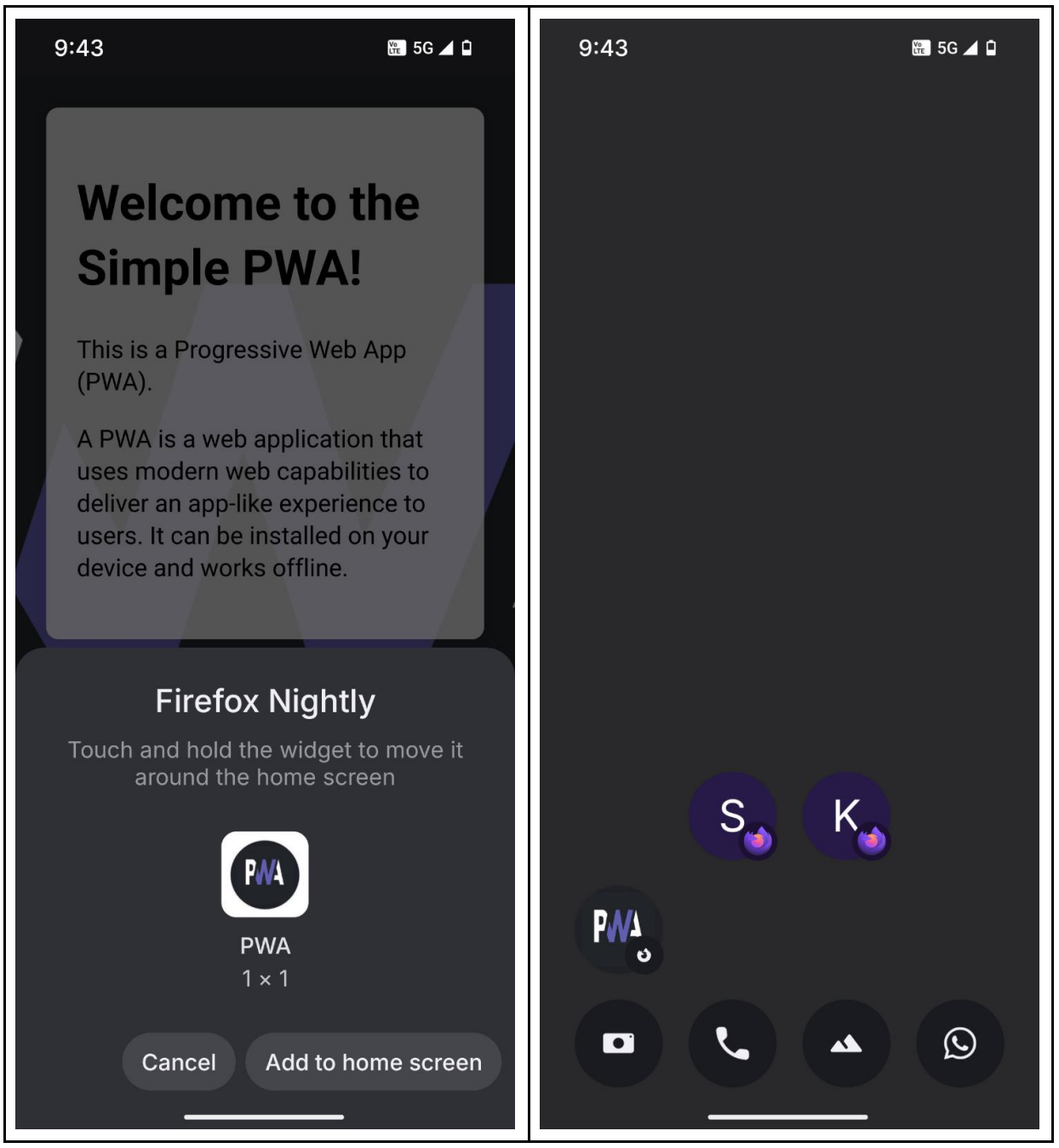
```
const CACHE_NAME = 'simple-pwa-cache-v1';
const urlsToCache = [
    'index.html',
    'offline-form.html',
    'styles.css',
    'manifest.json',
    'pwa-banner.png',
];

// Install the service worker and cache assets
self.addEventListener('install', (event) => {
    event.waitUntil(
        caches.open(CACHE_NAME).then((cache) => {
            console.log('[Service Worker] Caching essential assets');
            return cache.addAll(urlsToCache);
        })
    );
});

// Fetch event - serve cached assets or fetch from network
self.addEventListener('fetch', (event) => {
    event.respondWith(
        caches.match(event.request).then((response) => {
            if (response) {
                console.log('[Service Worker] Returning cached resource:',
```

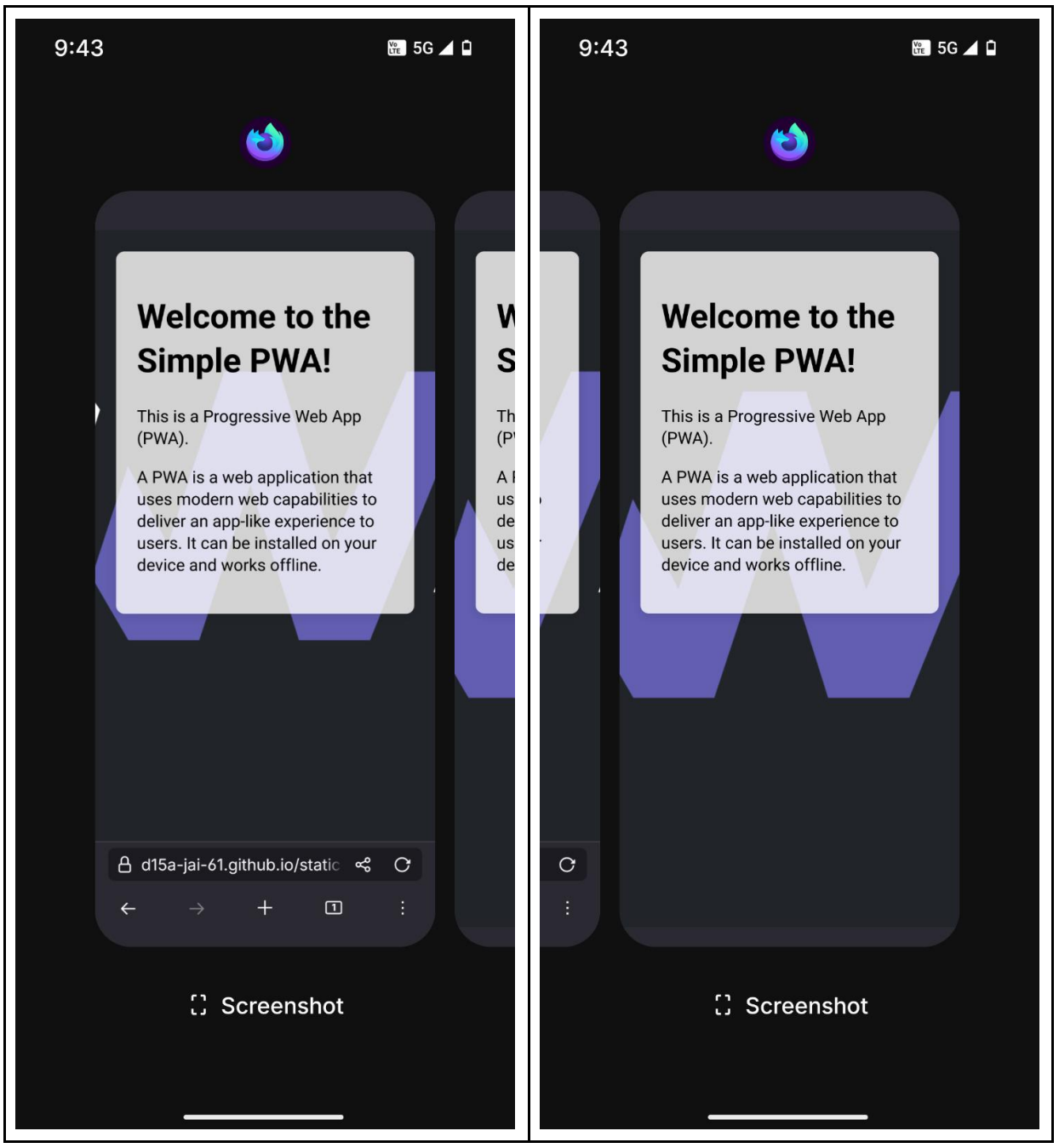
```
event.request.url);
                return response;
            }
            console.log('[Service Worker] Fetching from network:', event.request.url);
            return fetch(event.request);
        })
    );
});

// Sync event - retry offline form submissions
self.addEventListener('sync', (event) => {
    if (event.tag === 'sync-form') {
        console.log('[Service Worker] Sync event triggered: Submitting offline form
data...');
        event.waitUntil(syncFormData());
    }
});

function syncFormData() {
    return getFormDataFromIndexedDB().then((formData) => {
        if (formData) {
            console.log('[Service Worker] Syncing form data...');
            console.log('[Service Worker] Form Data:', formData);

            // Instead of sending to an API, let's store it back in IndexedDB or localStorage
for now
            saveFormDataLocally(formData);

            // Clear the form data from IndexedDB after "syncing"
            clearFormDataFromIndexedDB();
        }
    });
}

function getFormDataFromIndexedDB() {
    return new Promise((resolve, reject) => {
        const request = indexedDB.open('offlineFormData', 1);
        request.onsuccess = function() {
            const db = request.result;
            const tx = db.transaction('formData', 'readonly');
            const store = tx.objectStore('formData');
            const data = store.getAll();  // Get all stored form data
            data.onsuccess = function() {
                if (data.result.length > 0) {
                    resolve(data.result[0]);  // Return the first form data entry
                } else {
                    resolve(null);  // No data found
                }
            };
            data.onerror = reject;
        };
        request.onerror = reject;
    });
}
```

```
function saveFormDataLocally(formData) {
    // Save form data back into IndexedDB (simulating pushing data back into the app)
    const request = indexedDB.open('offlineFormData', 1);
    request.onsuccess = function() {
        const db = request.result;
        const tx = db.transaction('formData', 'readwrite');
        const store = tx.objectStore('formData');
        store.add(formData);  // Push data back into IndexedDB
        tx.oncomplete = function() {
            console.log('[Service Worker] Form data saved locally to IndexedDB:',
formData);
        };
    };
}

function clearFormDataFromIndexedDB() {
    const request = indexedDB.open('offlineFormData', 1);
    request.onsuccess = function() {
        const db = request.result;
        const tx = db.transaction('formData', 'readwrite');
        const store = tx.objectStore('formData');
        store.clear();  // Clear the form data from IndexedDB after syncing
        console.log('[Service Worker] Form data cleared from IndexedDB.');
    };
}

// Push event - handle push notifications (you can modify this part as per your needs)
self.addEventListener('push', (event) => {
    const options = {
        body: event.data.text(),
        icon: 'pwa-banner.png',  // Use pwa-banner.png for the notification icon
        badge: 'badge.png',      // Optional: You can use a badge if needed
    };

    event.waitUntil(
        self.registration.showNotification('New Content Available!', options)
    );
});

// Handle notification click
self.addEventListener('notificationclick', (event) => {
    event.notification.close();
    event.waitUntil(
        clients.openWindow('https://d15a-jai-61.github.io/static-pwa/') // Open your app's
homepage or a specific page
    );
});
```

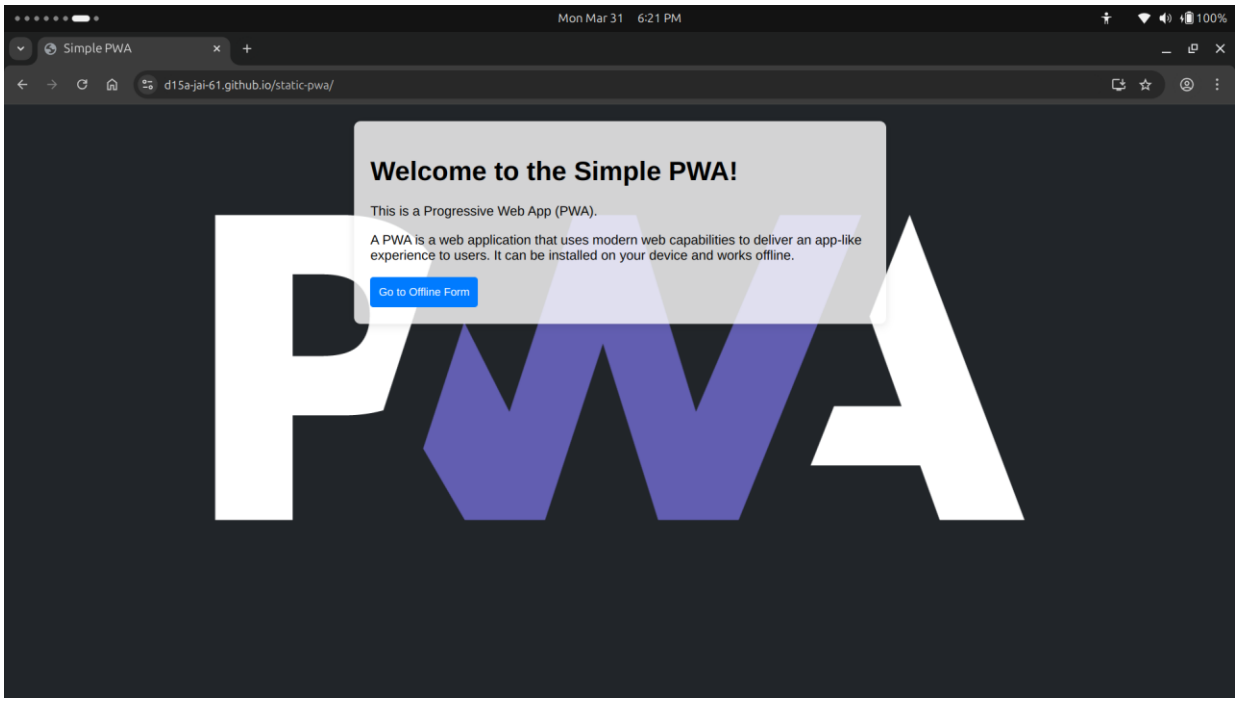
Output:

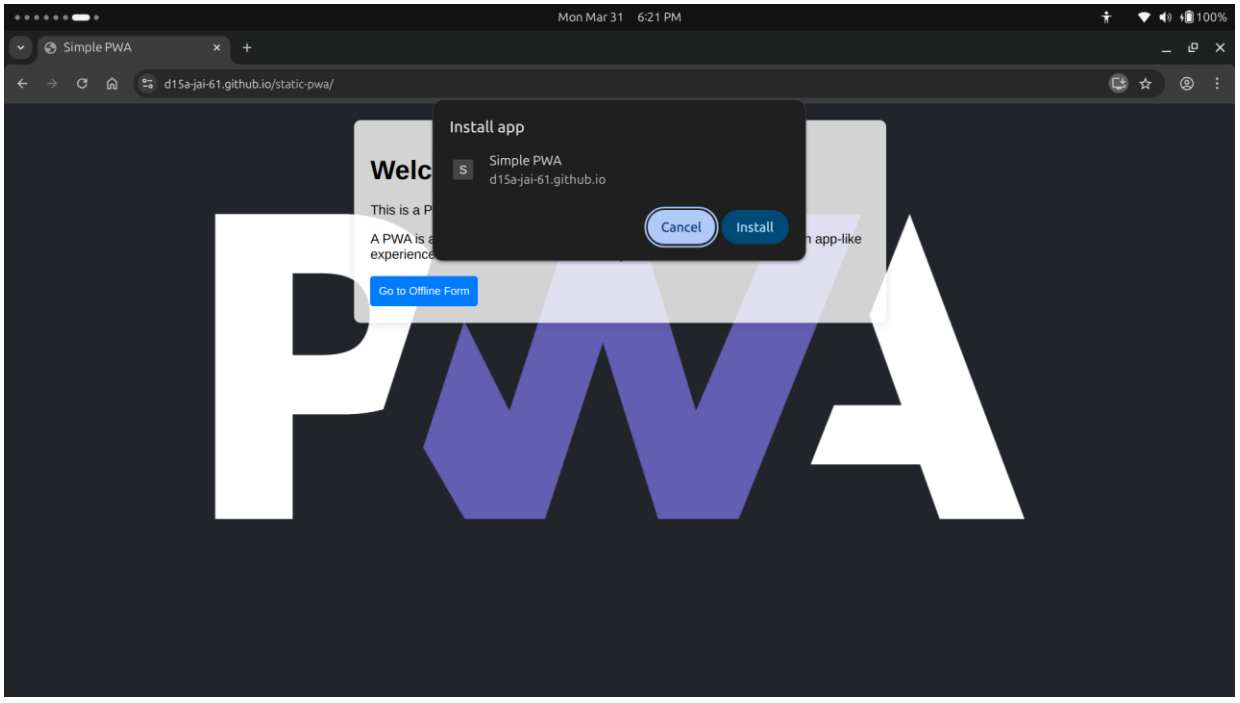

9:43
5G
Welcome to the Simple PWA!
This is a Progressive Web App (PWA).
A PWA is a web application that uses modern web capabilities to deliver an app-like experience to users. It can be installed on your device and works offline.
d15a-jai-61.github.io/static
9:43
5G
New tab
Bookmarks
Add
History
Downloads
Passwords
Extensions
Synchronise and sav...
Find in page
Translate page
Desktop site
Report broken site...
Add app to Home sc...
Save to collection
Print
Settings

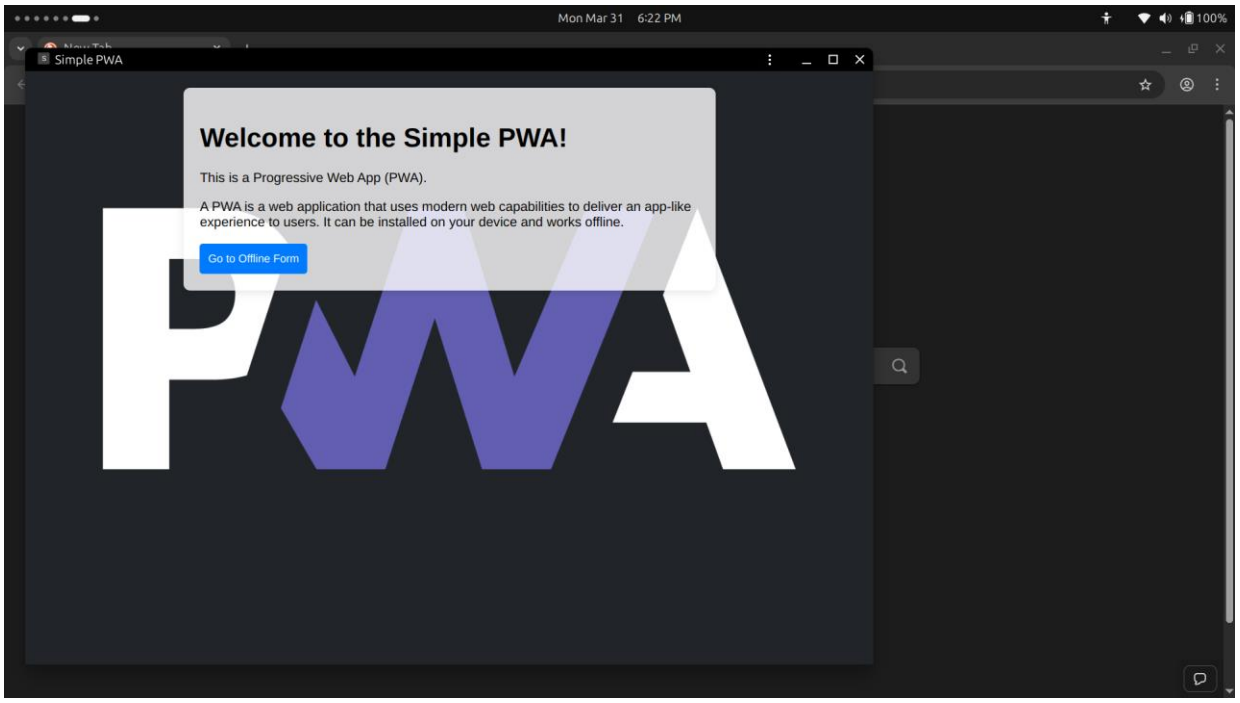

9:43
5G
Welcome to the Simple PWA!
This is a Progressive Web App (PWA).
A PWA is a web application that uses modern web capabilities to deliver an app-like experience to users. It can be installed on your device and works offline.
Firefox Nightly
Touch and hold the widget to move it around the home screen
PWA
1 × 1
Cancel
Add to home screen
9:43
5G
S
K
PWA

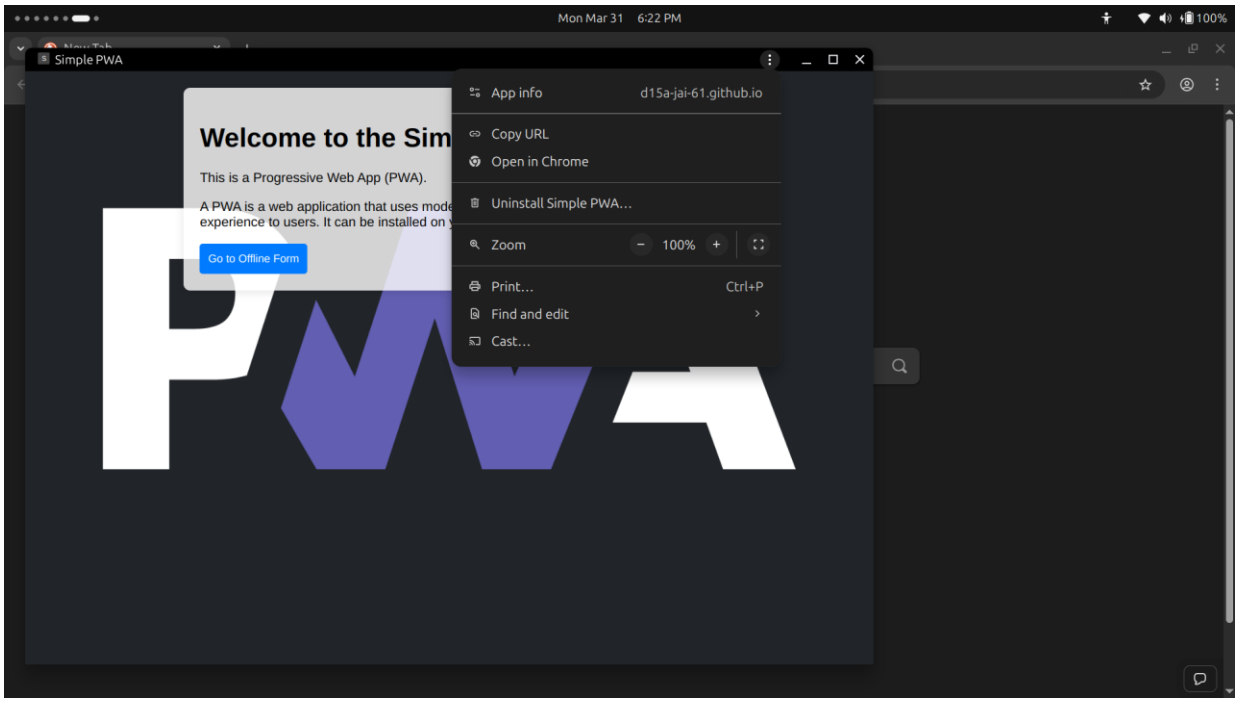

9:43
5G
Welcome to the Simple PWA!
This is a Progressive Web App (PWA).
A PWA is a web application that uses modern web capabilities to deliver an app-like experience to users. It can be installed on your device and works offline.
d15a-jai-61.github.io/static
Screenshot
9:43
5G
Welcome to the Simple PWA!
This is a Progressive Web App (PWA).
A PWA is a web application that uses modern web capabilities to deliver an app-like experience to users. It can be installed on your device and works offline.
Screenshot

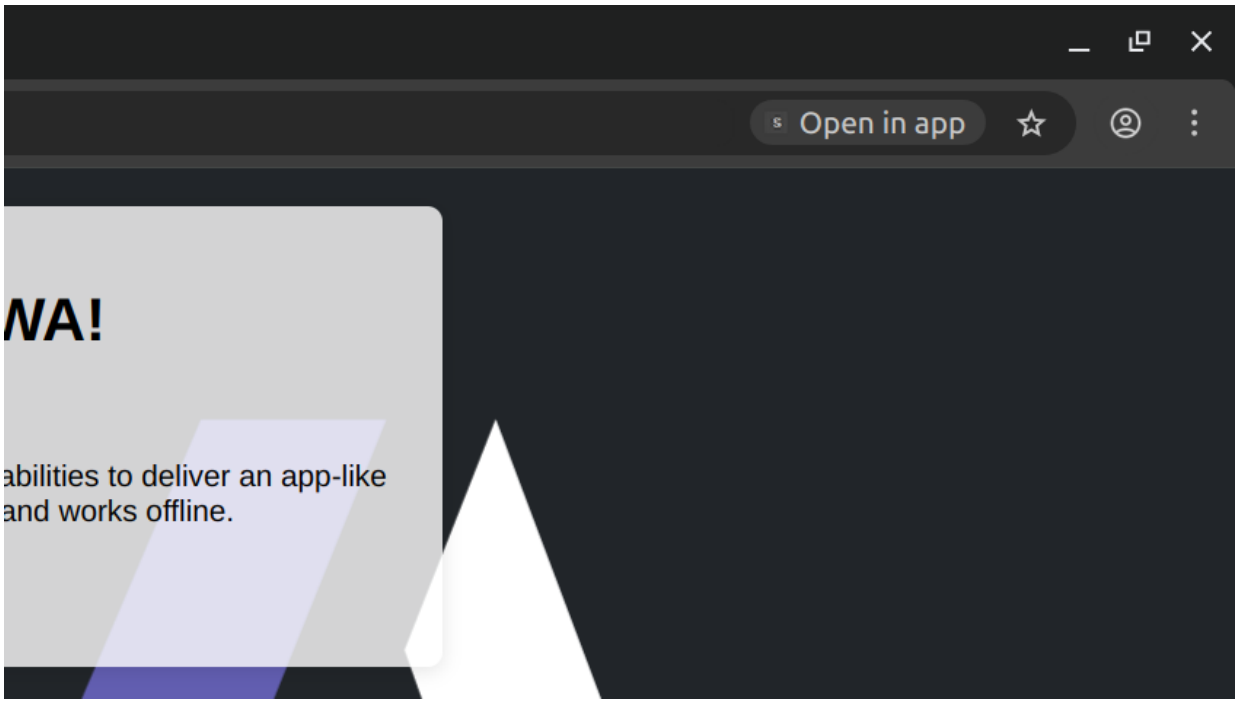

9:43
5G
Welcome to the Simple PWA!
This is a Progressive Web App (PWA).
A PWA is a web application that uses modern web capabilities to deliver an app-like experience to users. It can be installed on your device and works offline.

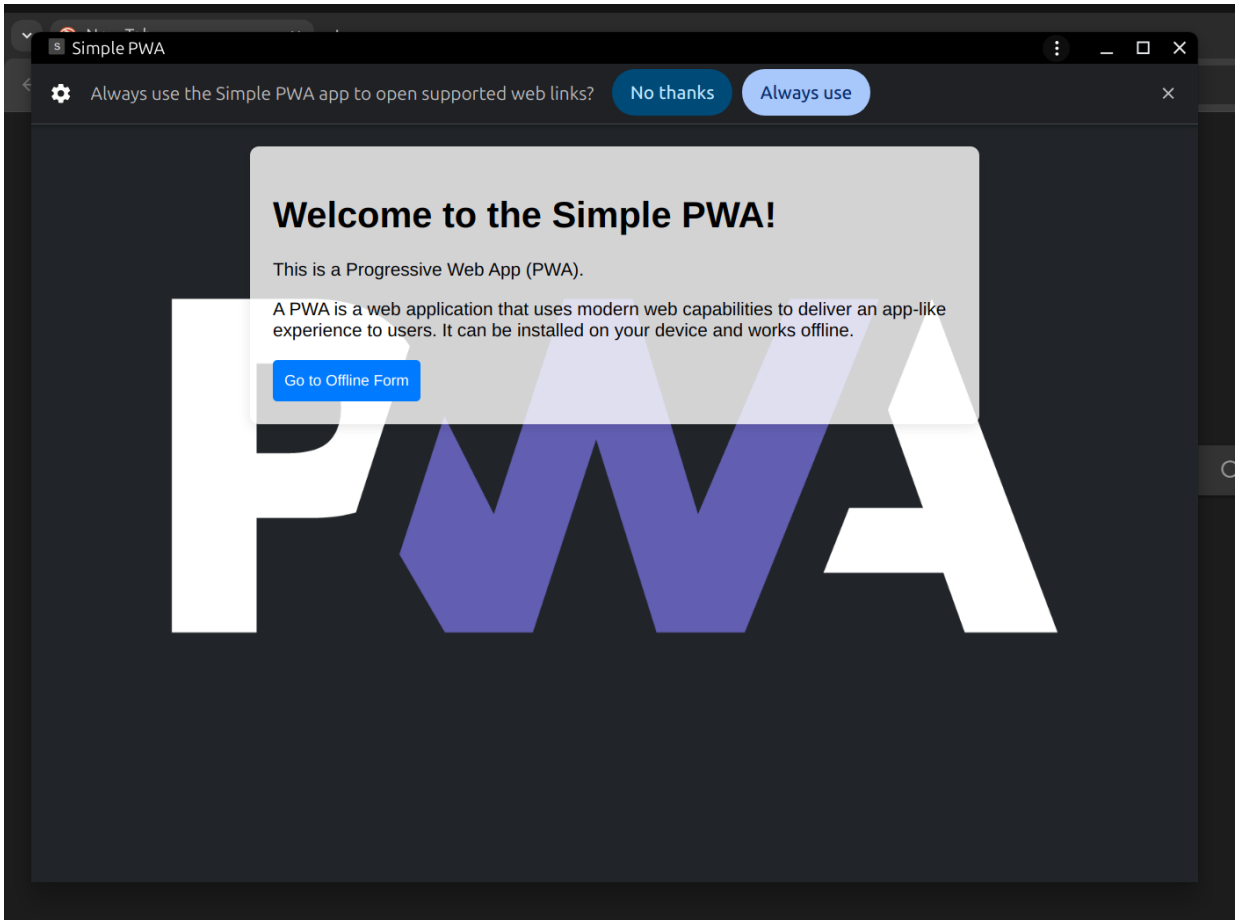

Mon Mar 31 6:21 PM
100%
Simple PWA
d15a-jai-61.github.io/static-pwa/
Welcome to the Simple PWA!
This is a Progressive Web App (PWA).
A PWA is a web application that uses modern web capabilities to deliver an app-like experience to users. It can be installed on your device and works offline.
Go to Offline Form
PWA


Mon Mar 31 6:21 PM
100%
Simple PWA
d15a-jai-61.github.io/static-pwa/
Install app
S
Simple PWA
d15a-jai-61.github.io
Cancel
Install
Go to Offline Form
PWA


Mon Mar 31 6:22 PM
100%
Simple PWA
Welcome to the Simple PWA!
This is a Progressive Web App (PWA).
A PWA is a web application that uses modern web capabilities to deliver an app-like experience to users. It can be installed on your device and works offline.
Go to Offline Form
PWA


Mon Mar 31 6:22 PM
100%
Simple PWA
Welcome to the Sim
This is a Progressive Web App (PWA).
Go to Offline Form
App info
d15a-jai-61.github.io
Copy URL
Open in Chrome
Uninstall Simple PWA...
Zoom
100%
Print...
Ctrl+P
Find and edit
Cast...
PWA


Open in app
NA!
abilities to deliver an app-like
and works offline.


Simple PWA
Always use the Simple PWA app to open supported web links?
No thanks
Always use
Welcome to the Simple PWA!
This is a Progressive Web App (PWA).
A PWA is a web application that uses modern web capabilities to deliver an app-like experience to users. It can be installed on your device and works offline.
Go to Offline Form
PWA

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.	62
Name	Vedang V. Wajge
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

Name: Diksha Utekar Jai Talreja Vedang Wajge D15A- 63/59/62

Experiment 8

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

Theory:

Service Worker

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

Things to note about Service Worker:

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

What can we do with Service Workers?

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

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

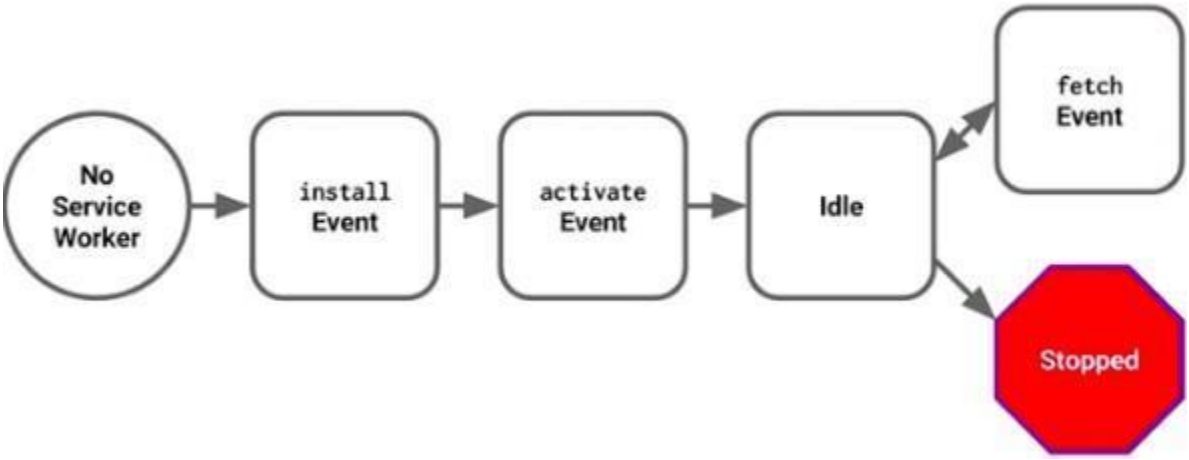
- You can manage **Push Notifications**
  You can manage push notifications with Service Worker and show any information message to the user.
- You can **Continue**
  Although Internet connection is broken, you can start any process with Background Sync of Service Worker.

What can’t we do with Service Workers?

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

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

Service Worker Cycle



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

- Registration
- Installation
- Activation

Registration

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

main.js

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

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

The scope of the service worker determines which files the service worker controls, in other words, from which path the service worker will intercept requests. The default scope is the location of the service worker file, and

extends to all directories below. So if service-worker.js is located in the root directory, the service worker will control requests from all files at this domain.

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

example: main.js

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

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

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

main.js

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

Installation

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

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

service-worker.js

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

Activation

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

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

service-worker.js

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

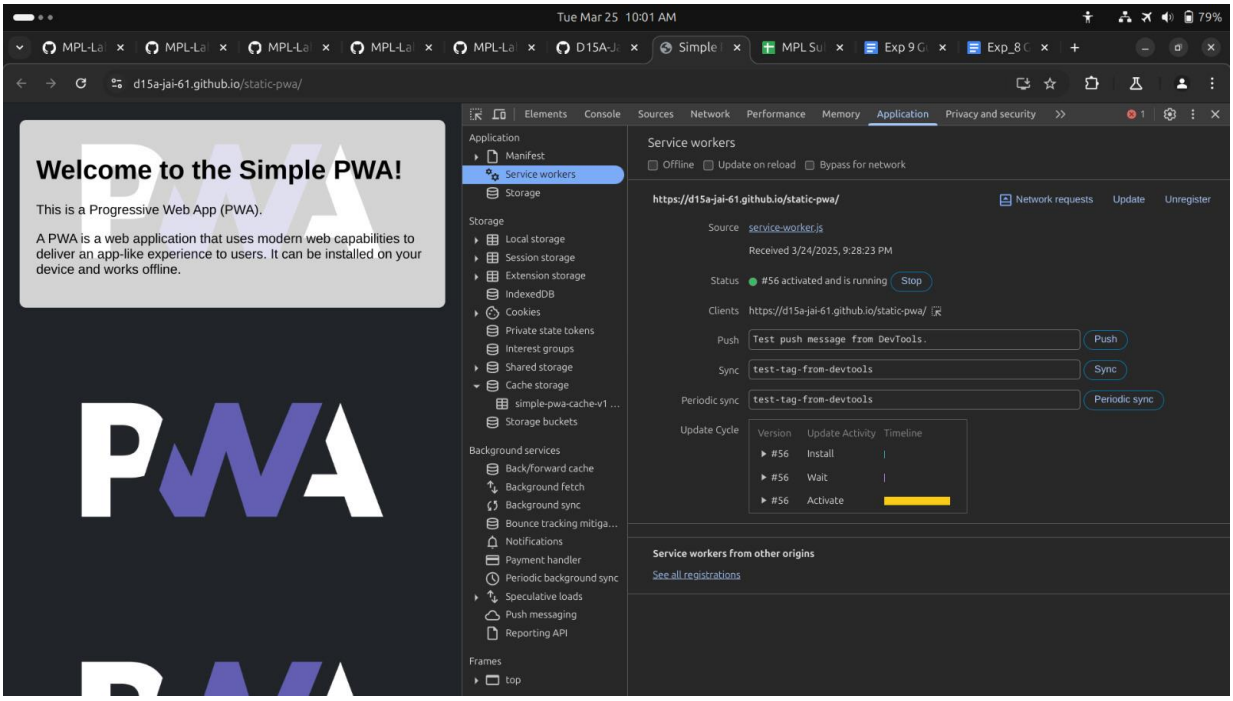
Once activated, the service worker controls all pages that load within its scope, and starts listening for events from those pages. However, pages in your app that were loaded before the service worker activation will not be under service worker control. The new service worker will only take over when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

Code under index.html:

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

Name: Diksha Utekar Jai Talreja Vedang Wajge D15A- 63/59/62

Output:



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.	62
Name	Vedang V. Wajge
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

Experiment 9

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

Theory:

Service Worker

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

Things to note about Service Worker:

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

Fetch Event

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

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

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

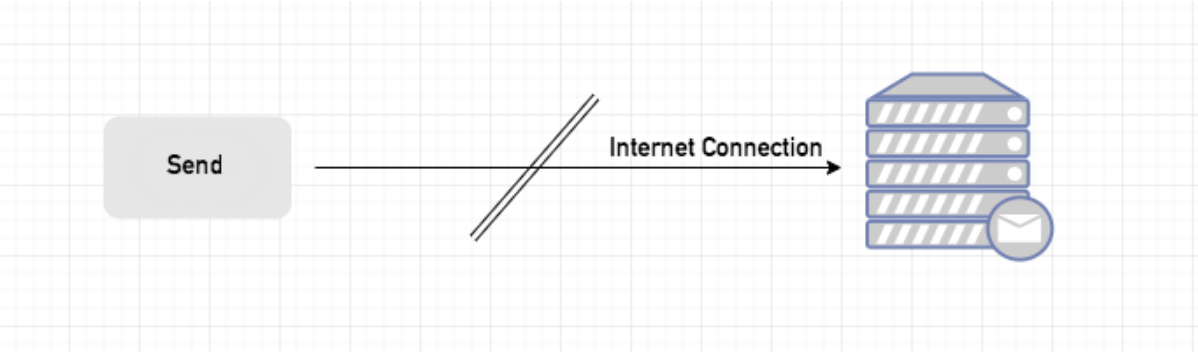
```
self.addEventListener("fetch", function (event) {
    const req = event.request;
    const url = new URL(req.url);

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

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

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

Sync Event

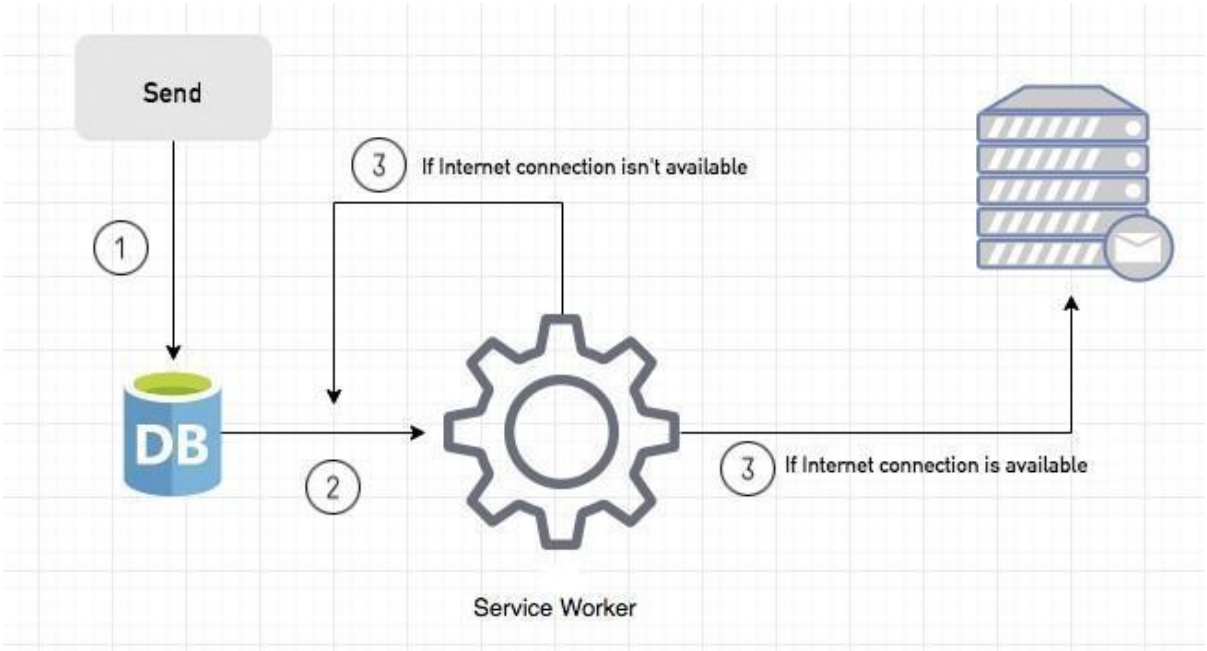
Background Sync is a Web API that is used to delay a process until the Internet connection is stable. We can adapt this definition to the real world; there is an e-mail client application that works on the browser and we want to send an email with this tool. Internet connection is broken while we are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

Here is a job for the Background Sync.

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



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



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

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

Push Event

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

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

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

Code:

service-worker.js :

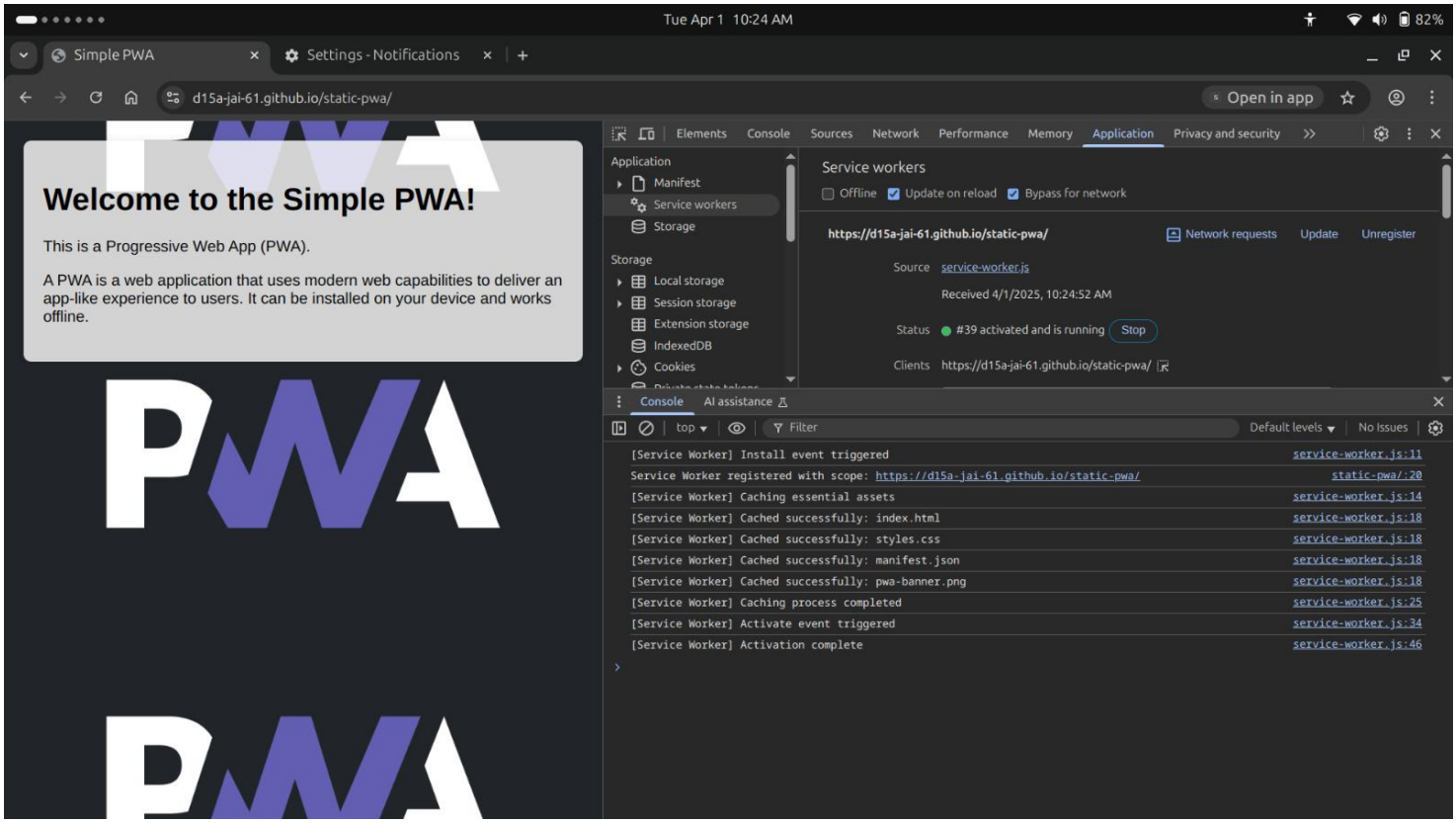
```
const CACHE_NAME = 'static-pwa-cache-v1';
const urlsToCache = [
    'index.html',
    'styles.css',
    'manifest.json',
    'pwa-banner.png',
];

// Install event - Cache essential assets
self.addEventListener('install', (event) => {
    console.log('[Service Worker] Install event triggered');
    event.waitUntil(
        caches.open(CACHE_NAME).then(cache => {
            console.log('[Service Worker] Caching essential assets');
            return Promise.all(
                urlsToCache.map(url =>
                    cache.add(url).then(() =>
                        console.log([Service Worker] Cached successfully: ${url})
                    ).catch(error =>
                        console.error([Service Worker] Failed to cache: ${url}, error)
                    )
                )
            );
        }).then(() => {
            console.log('[Service Worker] Caching process completed');
            return self.skipWaiting();
        })
        .catch(error => console.error('[Service Worker] Caching failed:', error))
    );
});

// Activate event - Cleanup old caches
self.addEventListener('activate', (event) => {
    console.log('[Service Worker] Activate event triggered');
    event.waitUntil(
        caches.keys().then(cacheNames => {
            return Promise.all(
                cacheNames.map(cacheName => {
                    if (cacheName !== CACHE_NAME) {
                        console.log('[Service Worker] Deleting old cache:', cacheName);
                        return caches.delete(cacheName);
                    }
                })
            );
        }).then(() => {
            console.log('[Service Worker] Activation complete');
            return self.clients.claim();
        })
    );
});

// Fetch event - Serve cached assets or fetch from network
```

```
self.addEventListener('fetch', (event) => {
    console.log([Service Worker] Fetch event triggered for: ${event.request.url});
    if (event.request.method !== 'GET') return;

    event.respondWith(
        fetch(event.request).then(networkResponse => {
            console.log([Service Worker] Network response for ${event.request.url});
            return caches.open(CACHE_NAME).then(cache => {
                cache.put(event.request, networkResponse.clone());
                return networkResponse;
            });
        }).catch(error => {
            console.log([Service Worker] Network failed, trying cache for ${event.request.url});
            return caches.match(event.request).then(cachedResponse => {
                if (cachedResponse) {
                    console.log([Service Worker] Serving from cache: ${event.request.url});
                    return cachedResponse;
                }
                if (event.request.mode === 'navigate') {
                    return caches.match(OFFLINE_URL);
                }
                return new Response('Offline', { status: 503, statusText: 'Service Unavailable' });
            });
        })
    );
});

// Sync event - Handle background sync
self.addEventListener('sync', (event) => {
    console.log([Service Worker] Sync event triggered: ${event.tag});
    if (event.tag === 'sync-demo') {
        event.waitUntil(
            handleSync().catch(error => {
                console.error('[Service Worker] Sync failed:', error);
            })
        );
    }
});

async function handleSync() {
    console.log('[Service Worker] Starting sync...');
    await new Promise(resolve => setTimeout(resolve, 2000));
    console.log('[Service Worker] Sync task completed successfully!');
}

// Push event - Handle push notifications
self.addEventListener('push', (event) => {
    console.log('[Service Worker] Push event received');
    const options = {
        body: event.data ? event.data.text() : 'New update available!',
        icon: 'pwa-banner.png',
        badge: 'badge.png',
        actions: [
            { action: 'open', title: 'View Now' },
            { action: 'dismiss', title: 'Dismiss' }
        ]
```

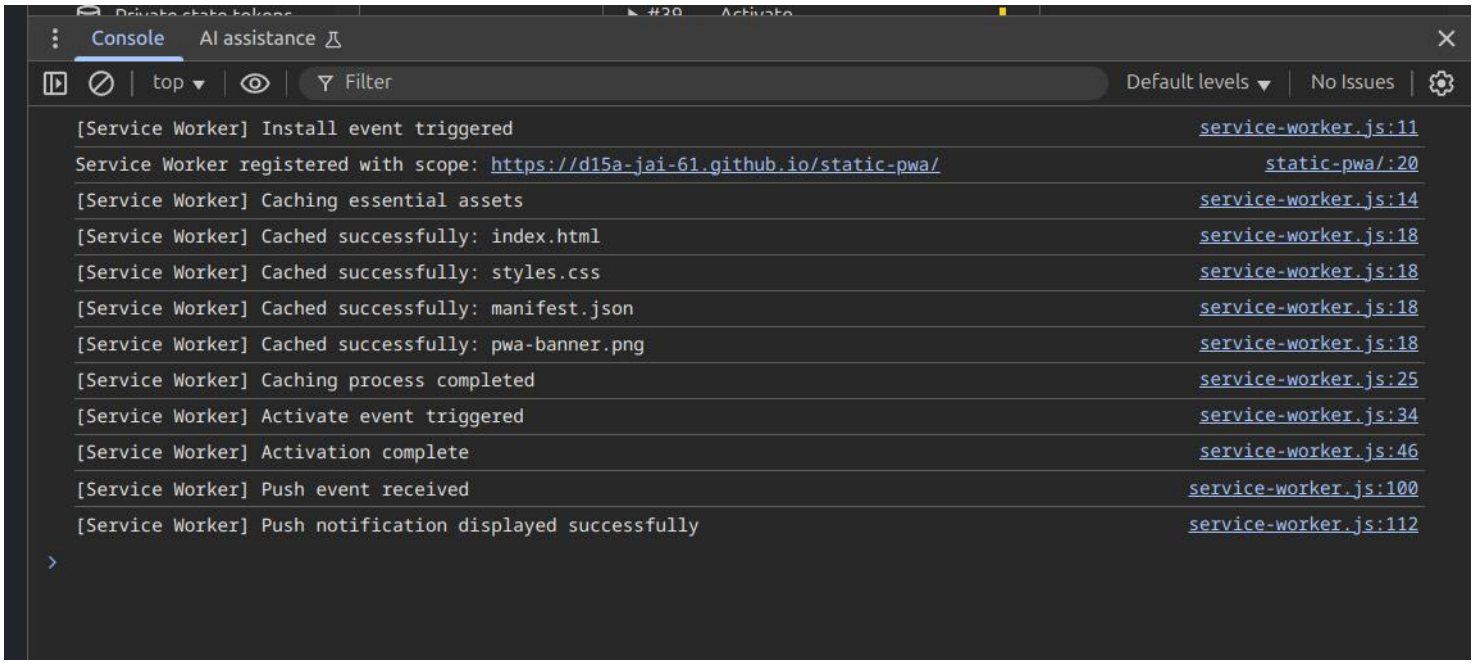
```
    };
    event.waitUntil(
        self.registration.showNotification('New Notification', options)
        .then(() => console.log('[Service Worker] Push notification displayed successfully'))
        .catch((error) => console.error('[Service Worker] Failed to display push notification:', error))
    );
});

// Handle notification click
self.addEventListener('notificationclick', (event) => {
    console.log([Service Worker] Notification clicked: ${event.notification.title});
    event.notification.close();
    if (event.action === 'open') {
        console.log('[Service Worker] Opening application');
        event.waitUntil(clients.openWindow('https://your-static-site.com'));
    } else {
        console.log('[Service Worker] Notification dismissed');
    }
});
```

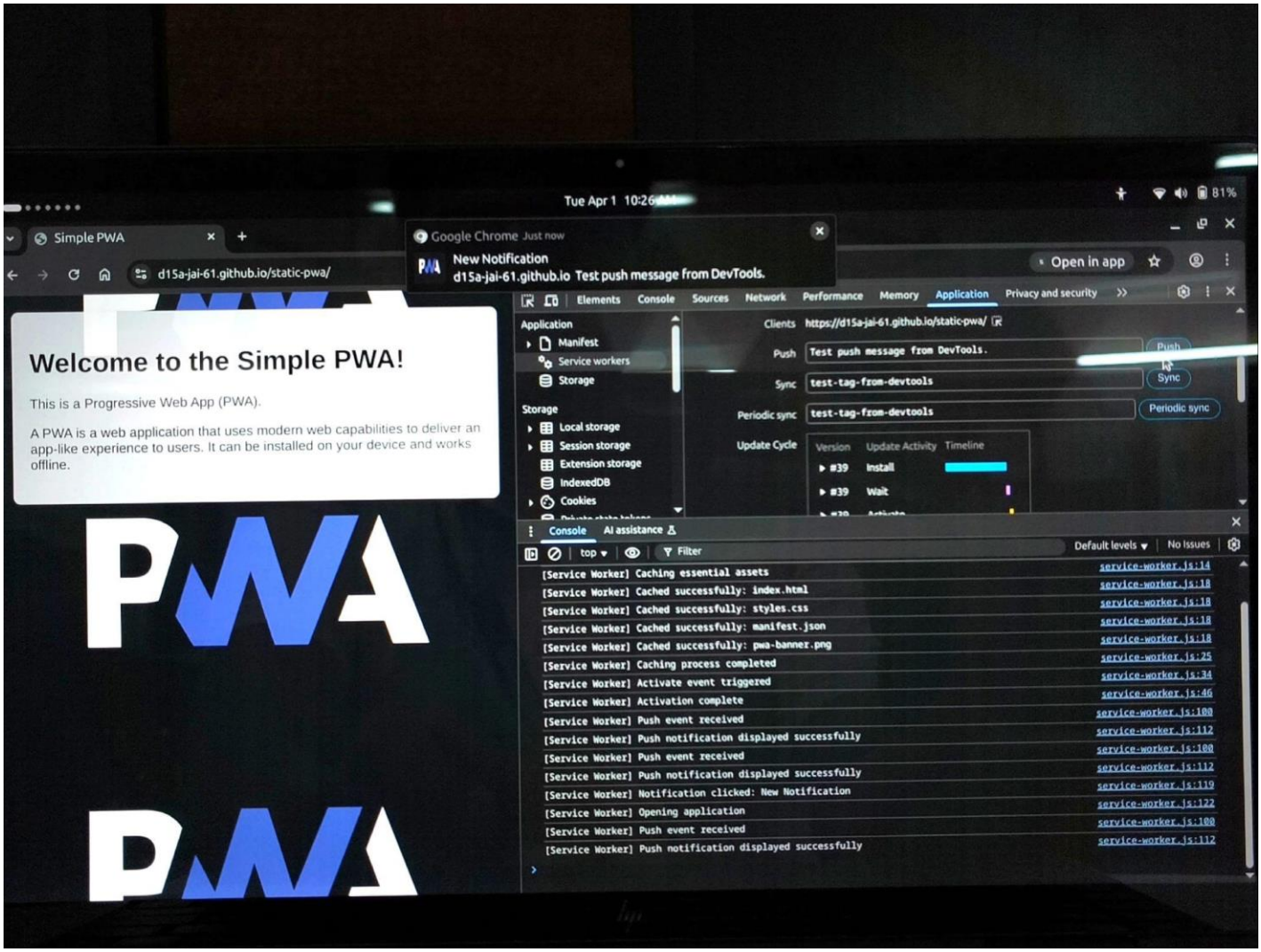
OUTPUT:

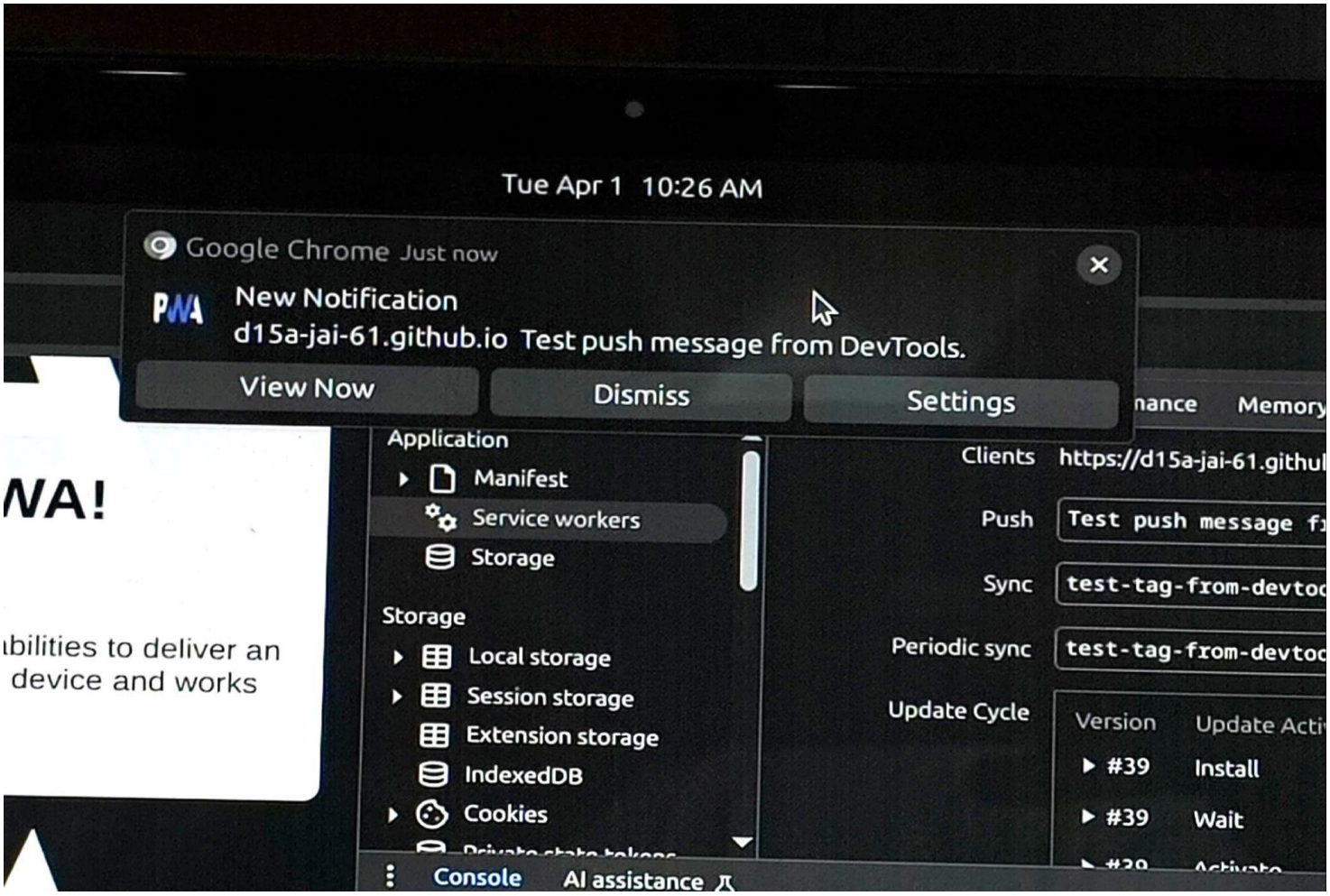
Fetch event



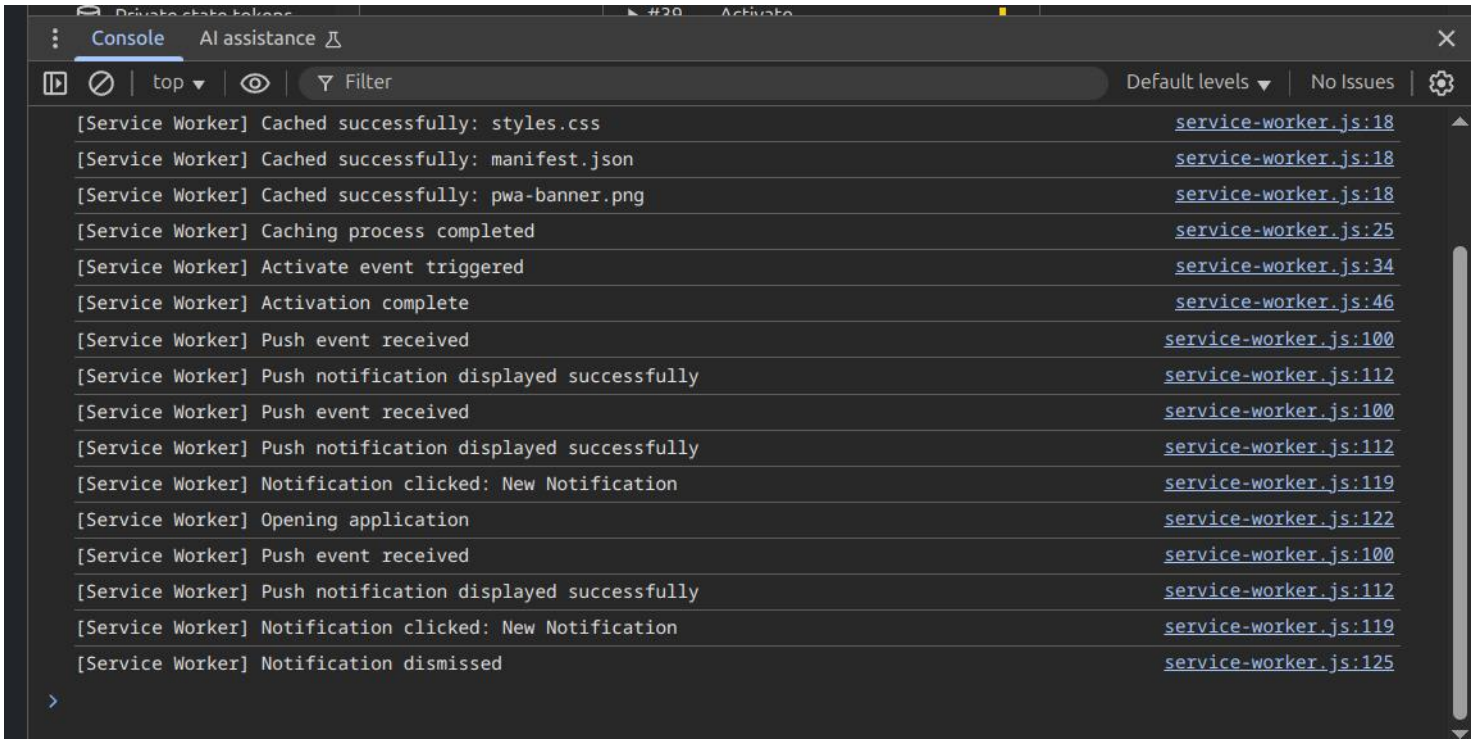
Push event


Console
AI assistance
top
Filter
Default levels
No Issues
[Service Worker] Install event triggered service-worker.js:11
Service Worker registered with scope: https://d15a-jai-61.github.io/static-pwa/ static-pwa/:20
[Service Worker] Caching essential assets service-worker.js:14
[Service Worker] Cached successfully: index.html service-worker.js:18
[Service Worker] Cached successfully: styles.css service-worker.js:18
[Service Worker] Cached successfully: manifest.json service-worker.js:18
[Service Worker] Cached successfully: pwa-banner.png service-worker.js:18
[Service Worker] Caching process completed service-worker.js:25
[Service Worker] Activate event triggered service-worker.js:34
[Service Worker] Activation complete service-worker.js:46
[Service Worker] Push event received service-worker.js:100
[Service Worker] Push notification displayed successfully service-worker.js:112


Simple PWA
d15a-jai-61.github.io/static-pwa/
Google Chrome Just now
New Notification
d15a-jai-61.github.io Test push message from DevTools.
Welcome to the Simple PWA!
This is a Progressive Web App (PWA).
A PWA is a web application that uses modern web capabilities to deliver an app-like experience to users. It can be installed on your device and works offline.
PWA
PWA
Open in app
Elements
Console
Sources
Network
Performance
Memory
Application
Privacy and security
Application
Manifest
Service workers
Storage
Storage
Local storage
Session storage
Extension storage
IndexedDB
Cookies
Clients https://d15a-jai-61.github.io/static-pwa/
Push Test push message from DevTools.
Sync test-tag-from-devtools
Periodic sync test-tag-from-devtools
Update Cycle
Version
Update Activity
Timeline
#39 Install
#39 Wait
Console
AI assistance
top
Filter
Default levels
No Issues
[Service Worker] Caching essential assets service-worker.js:14
[Service Worker] Cached successfully: index.html service-worker.js:18
[Service Worker] Cached successfully: styles.css service-worker.js:18
[Service Worker] Cached successfully: manifest.json service-worker.js:18
[Service Worker] Cached successfully: pwa-banner.png service-worker.js:18
[Service Worker] Caching process completed service-worker.js:25
[Service Worker] Activate event triggered service-worker.js:34
[Service Worker] Activation complete service-worker.js:46
[Service Worker] Push event received service-worker.js:100
[Service Worker] Push notification displayed successfully service-worker.js:112
[Service Worker] Push event received service-worker.js:100
[Service Worker] Push notification displayed successfully service-worker.js:112
[Service Worker] Notification clicked: New Notification service-worker.js:119
[Service Worker] Opening application service-worker.js:122
[Service Worker] Push event received service-worker.js:100
[Service Worker] Push notification displayed successfully service-worker.js:112



After dismissing the notification



Sync event


[Service Worker] Opening application service-worker.js:122
[Service Worker] Push event received service-worker.js:100
[Service Worker] Push notification displayed successfully service-worker.js:112
[Service Worker] Notification clicked: New Notification service-worker.js:119
[Service Worker] Notification dismissed service-worker.js:125
[Service Worker] Sync event triggered: test-tag-from-devtools service-worker.js:82

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.	62
Name	Vedang V. Wajge
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

Experiment 9

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

Theory:

Service Worker

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

Things to note about Service Worker:

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

Fetch Event

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

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

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

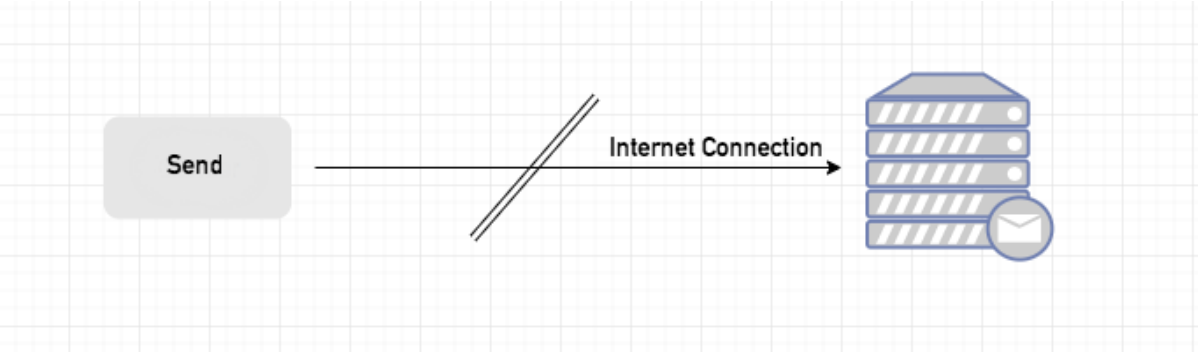
```
self.addEventListener("fetch", function (event) {
    const req = event.request;
    const url = new URL(req.url);

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

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

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

Sync Event

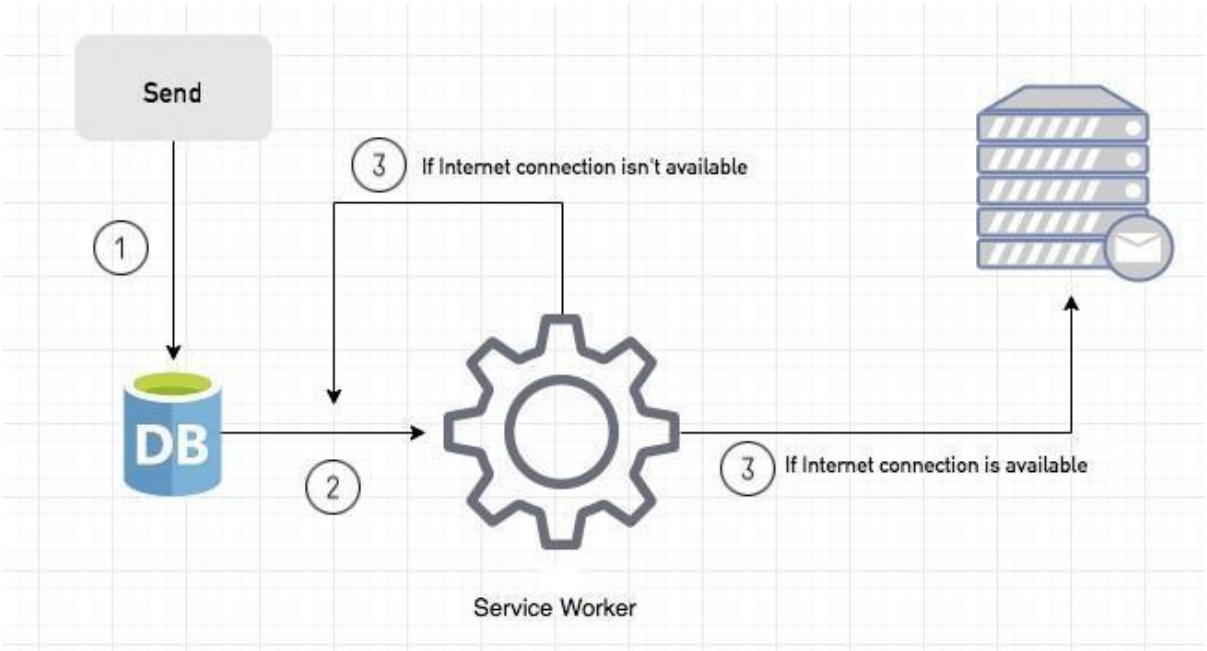
Background Sync is a Web API that is used to delay a process until the Internet connection is stable. We can adapt this definition to the real world; there is an e-mail client application that works on the browser and we want to send an email with this tool. Internet connection is broken while we are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

Here is a job for the Background Sync.

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



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



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

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

Push Event

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

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

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

Code:

service-worker.js :

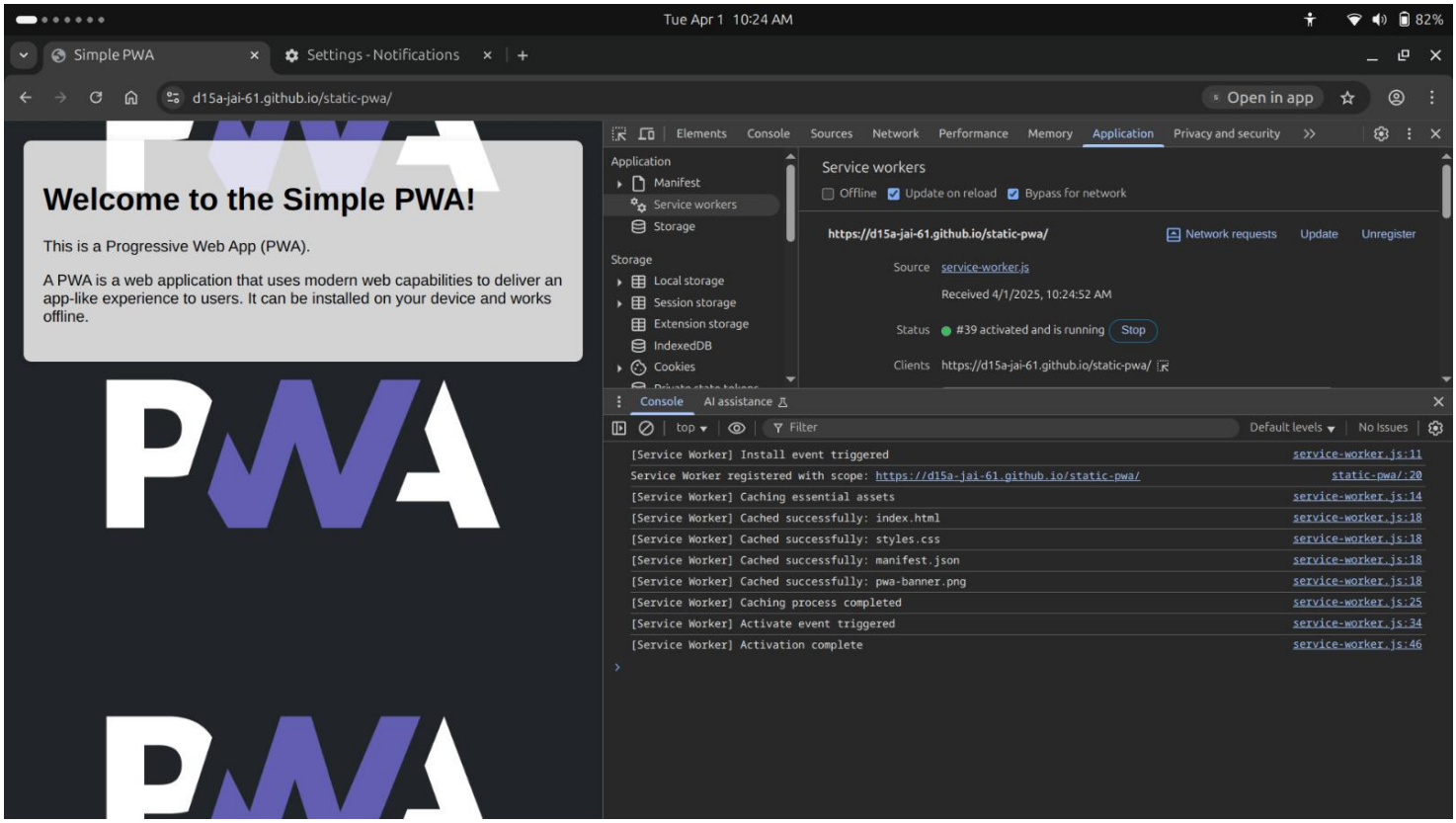
```
const CACHE_NAME = 'static-pwa-cache-v1';
const urlsToCache = [
    'index.html',
    'styles.css',
    'manifest.json',
    'pwa-banner.png',
];

// Install event - Cache essential assets
self.addEventListener('install', (event) => {
    console.log('[Service Worker] Install event triggered');
    event.waitUntil(
        caches.open(CACHE_NAME).then(cache => {
            console.log('[Service Worker] Caching essential assets');
            return Promise.all(
                urlsToCache.map(url =>
                    cache.add(url).then(() =>
                        console.log([Service Worker] Cached successfully: ${url})
                    ).catch(error =>
                        console.error([Service Worker] Failed to cache: ${url}, error)
                    )
                )
            );
        }).then(() => {
            console.log('[Service Worker] Caching process completed');
            return self.skipWaiting();
        })
        .catch(error => console.error('[Service Worker] Caching failed:', error))
    );
});

// Activate event - Cleanup old caches
self.addEventListener('activate', (event) => {
    console.log('[Service Worker] Activate event triggered');
    event.waitUntil(
        caches.keys().then(cacheNames => {
            return Promise.all(
                cacheNames.map(cacheName => {
                    if (cacheName !== CACHE_NAME) {
                        console.log('[Service Worker] Deleting old cache:', cacheName);
                        return caches.delete(cacheName);
                    }
                })
            );
        }).then(() => {
            console.log('[Service Worker] Activation complete');
            return self.clients.claim();
        })
    );
});

// Fetch event - Serve cached assets or fetch from network
```

```
self.addEventListener('fetch', (event) => {
    console.log([Service Worker] Fetch event triggered for: ${event.request.url});
    if (event.request.method !== 'GET') return;

    event.respondWith(
        fetch(event.request).then(networkResponse => {
            console.log([Service Worker] Network response for ${event.request.url});
            return caches.open(CACHE_NAME).then(cache => {
                cache.put(event.request, networkResponse.clone());
                return networkResponse;
            });
        }).catch(error => {
            console.log([Service Worker] Network failed, trying cache for ${event.request.url});
            return caches.match(event.request).then(cachedResponse => {
                if (cachedResponse) {
                    console.log([Service Worker] Serving from cache: ${event.request.url});
                    return cachedResponse;
                }
                if (event.request.mode === 'navigate') {
                    return caches.match(OFFLINE_URL);
                }
                return new Response('Offline', { status: 503, statusText: 'Service Unavailable' });
            });
        })
    );
});

// Sync event - Handle background sync
self.addEventListener('sync', (event) => {
    console.log([Service Worker] Sync event triggered: ${event.tag});
    if (event.tag === 'sync-demo') {
        event.waitUntil(
            handleSync().catch(error => {
                console.error('[Service Worker] Sync failed:', error);
            })
        );
    }
});

async function handleSync() {
    console.log('[Service Worker] Starting sync...');
    await new Promise(resolve => setTimeout(resolve, 2000));
    console.log('[Service Worker] Sync task completed successfully!');
}

// Push event - Handle push notifications
self.addEventListener('push', (event) => {
    console.log('[Service Worker] Push event received');
    const options = {
        body: event.data ? event.data.text() : 'New update available!',
        icon: 'pwa-banner.png',
        badge: 'badge.png',
        actions: [
            { action: 'open', title: 'View Now' },
            { action: 'dismiss', title: 'Dismiss' }
        ]
```

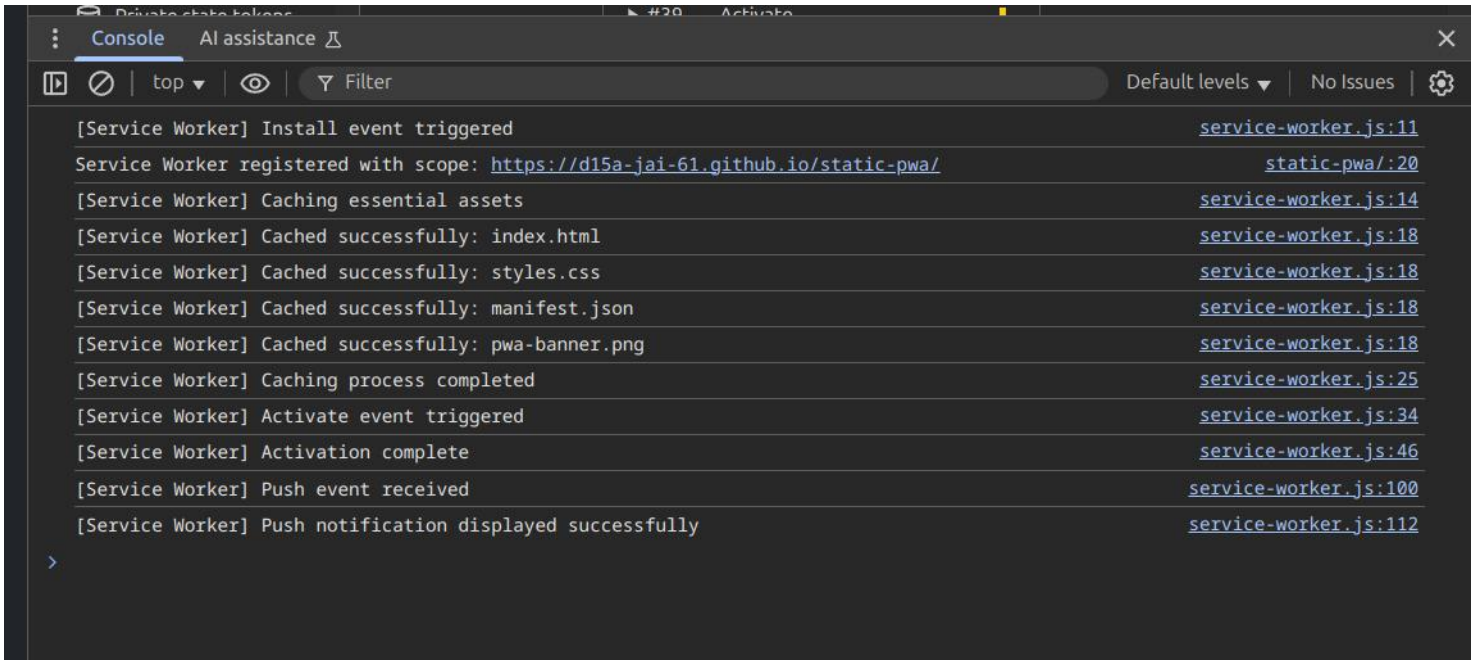
```
    };
    event.waitUntil(
        self.registration.showNotification('New Notification', options)
        .then(() => console.log('[Service Worker] Push notification displayed successfully'))
        .catch((error) => console.error('[Service Worker] Failed to display push notification:', error))
    );
});

// Handle notification click
self.addEventListener('notificationclick', (event) => {
    console.log([Service Worker] Notification clicked: ${event.notification.title});
    event.notification.close();
    if (event.action === 'open') {
        console.log('[Service Worker] Opening application');
        event.waitUntil(clients.openWindow('https://your-static-site.com'));
    } else {
        console.log('[Service Worker] Notification dismissed');
    }
});
```

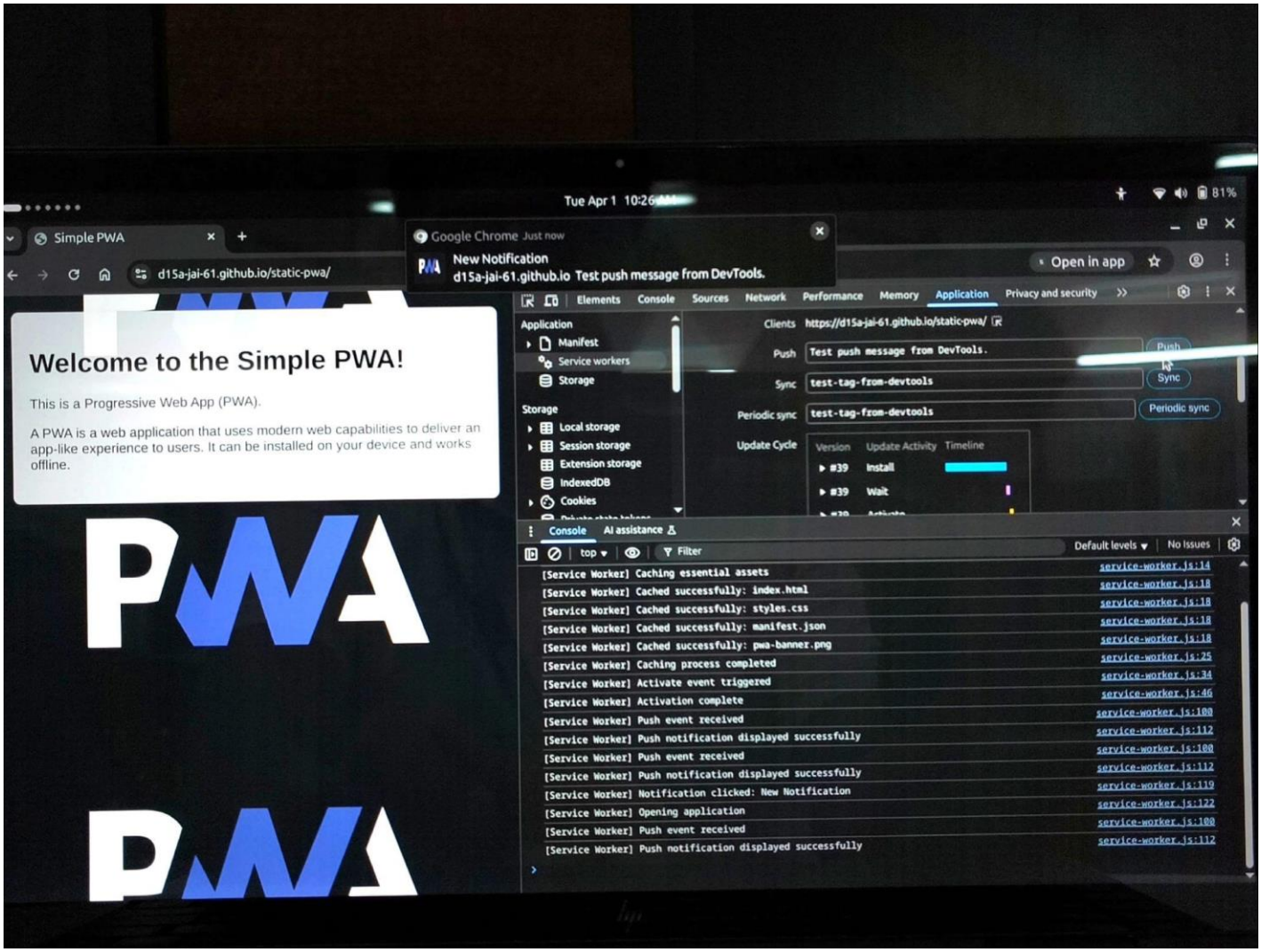
OUTPUT:

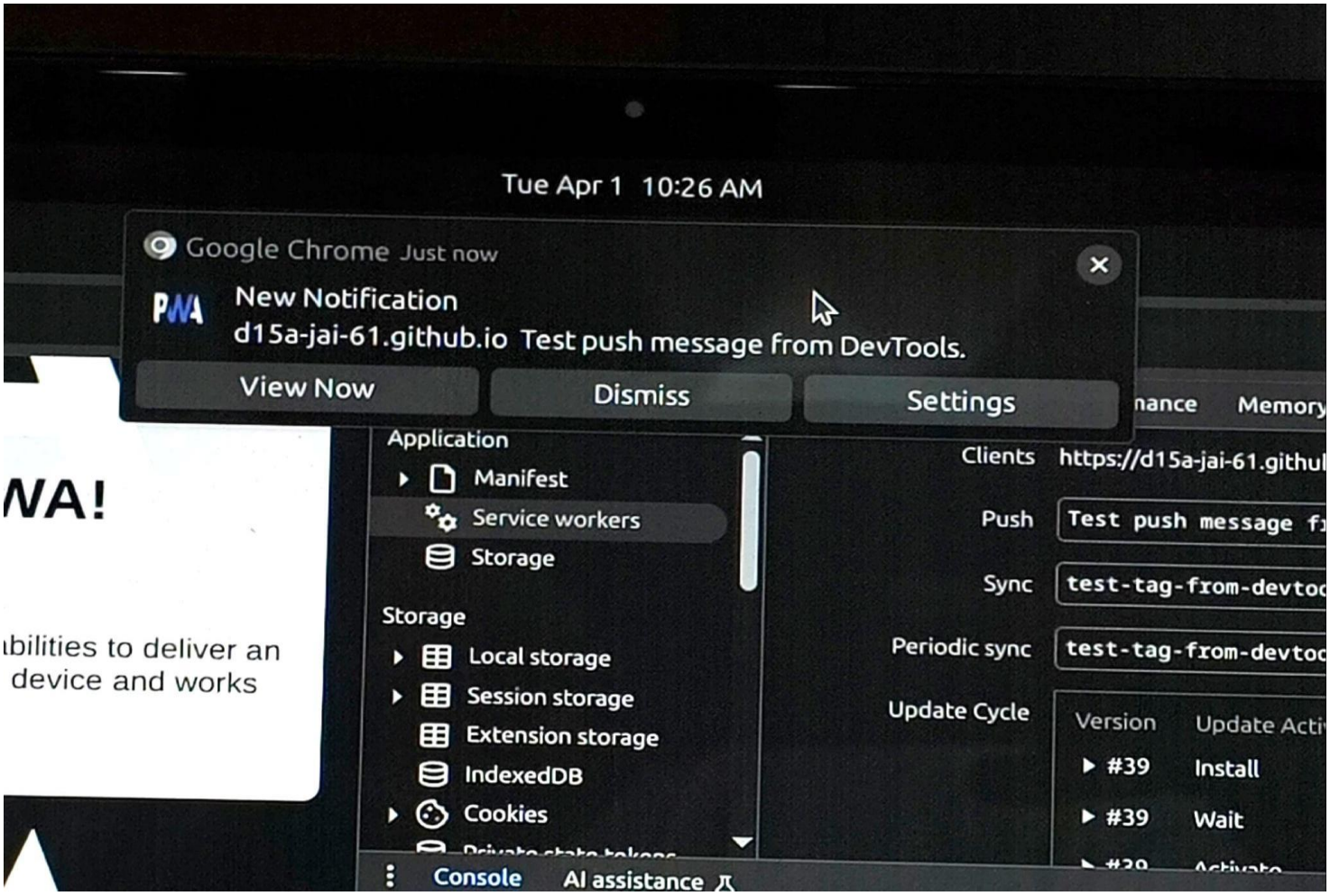
Fetch event



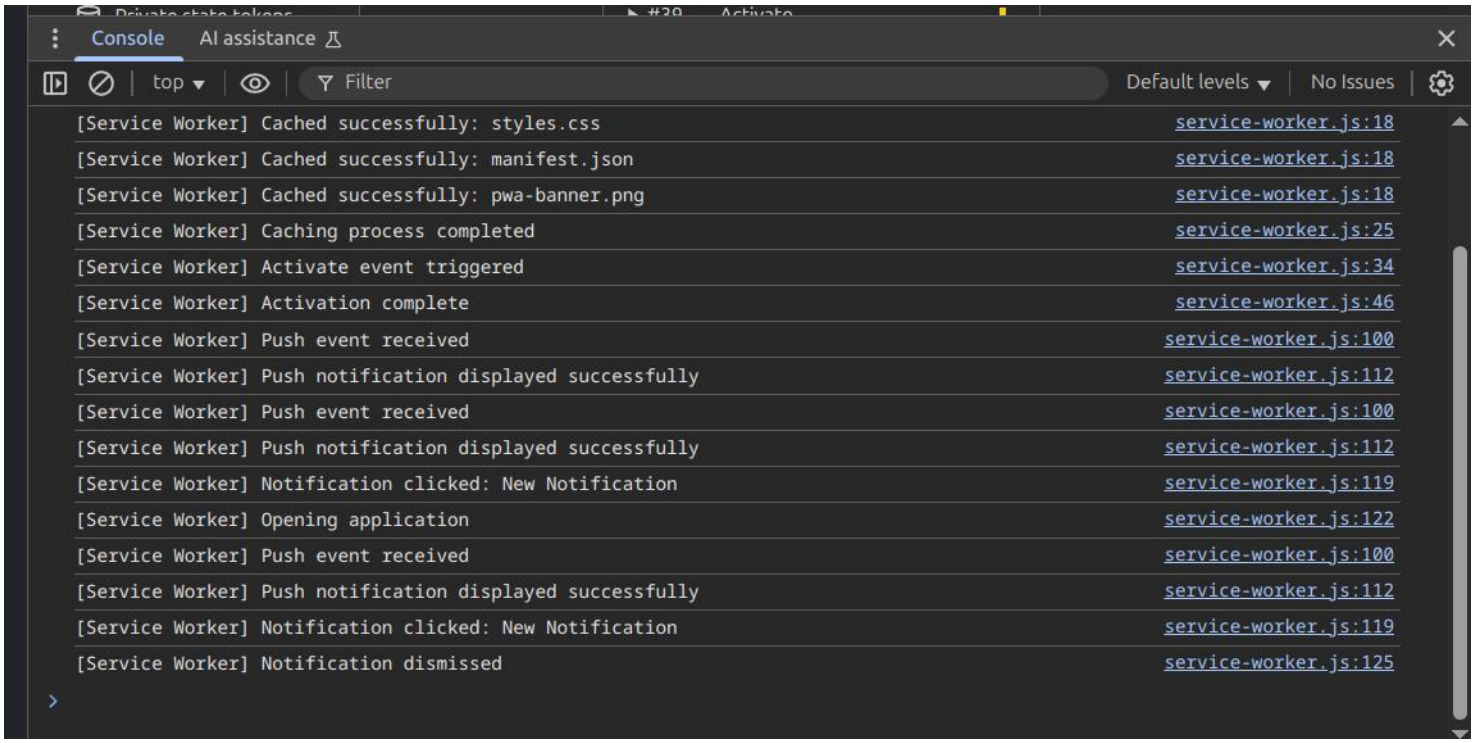
Push event


Console
AI assistance
top
Filter
Default levels
No Issues
[Service Worker] Install event triggered service-worker.js:11
Service Worker registered with scope: https://d15a-jai-61.github.io/static-pwa/ static-pwa/:20
[Service Worker] Caching essential assets service-worker.js:14
[Service Worker] Cached successfully: index.html service-worker.js:18
[Service Worker] Cached successfully: styles.css service-worker.js:18
[Service Worker] Cached successfully: manifest.json service-worker.js:18
[Service Worker] Cached successfully: pwa-banner.png service-worker.js:18
[Service Worker] Caching process completed service-worker.js:25
[Service Worker] Activate event triggered service-worker.js:34
[Service Worker] Activation complete service-worker.js:46
[Service Worker] Push event received service-worker.js:100
[Service Worker] Push notification displayed successfully service-worker.js:112


Simple PWA
d15a-jai-61.github.io/static-pwa/
Google Chrome Just now
New Notification
d15a-jai-61.github.io Test push message from DevTools.
Welcome to the Simple PWA!
This is a Progressive Web App (PWA).
A PWA is a web application that uses modern web capabilities to deliver an app-like experience to users. It can be installed on your device and works offline.
PWA
PWA
Open in app
Elements
Console
Sources
Network
Performance
Memory
Application
Privacy and security
Application
Manifest
Service workers
Storage
Storage
Local storage
Session storage
Extension storage
IndexedDB
Cookies
Clients https://d15a-jai-61.github.io/static-pwa/
Push Test push message from DevTools.
Sync test-tag-from-devtools
Periodic sync test-tag-from-devtools
Update Cycle
Version
Update Activity
Timeline
#39 Install
#39 Wait
Console
AI assistance
top
Filter
Default levels
No Issues
[Service Worker] Caching essential assets service-worker.js:14
[Service Worker] Cached successfully: index.html service-worker.js:18
[Service Worker] Cached successfully: styles.css service-worker.js:18
[Service Worker] Cached successfully: manifest.json service-worker.js:18
[Service Worker] Cached successfully: pwa-banner.png service-worker.js:18
[Service Worker] Caching process completed service-worker.js:25
[Service Worker] Activate event triggered service-worker.js:34
[Service Worker] Activation complete service-worker.js:46
[Service Worker] Push event received service-worker.js:100
[Service Worker] Push notification displayed successfully service-worker.js:112
[Service Worker] Push event received service-worker.js:100
[Service Worker] Push notification displayed successfully service-worker.js:112
[Service Worker] Notification clicked: New Notification service-worker.js:119
[Service Worker] Opening application service-worker.js:122
[Service Worker] Push event received service-worker.js:100
[Service Worker] Push notification displayed successfully service-worker.js:112



After dismissing the notification



Sync event


[Service Worker] Opening application service-worker.js:122
[Service Worker] Push event received service-worker.js:100
[Service Worker] Push notification displayed successfully service-worker.js:112
[Service Worker] Notification clicked: New Notification service-worker.js:119
[Service Worker] Notification dismissed service-worker.js:125
[Service Worker] Sync event triggered: test-tag-from-devtools service-worker.js:82

MAD & PWA Lab
Journal

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	62
Name	Vedang V. Wajge
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

Experiment 10

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

Theory:

GitHub Pages

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

GitHub Pages provides the following key features:

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

Reasons for favoring this over Firebase:

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

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

Pros

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

Cons

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

Firebase

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.

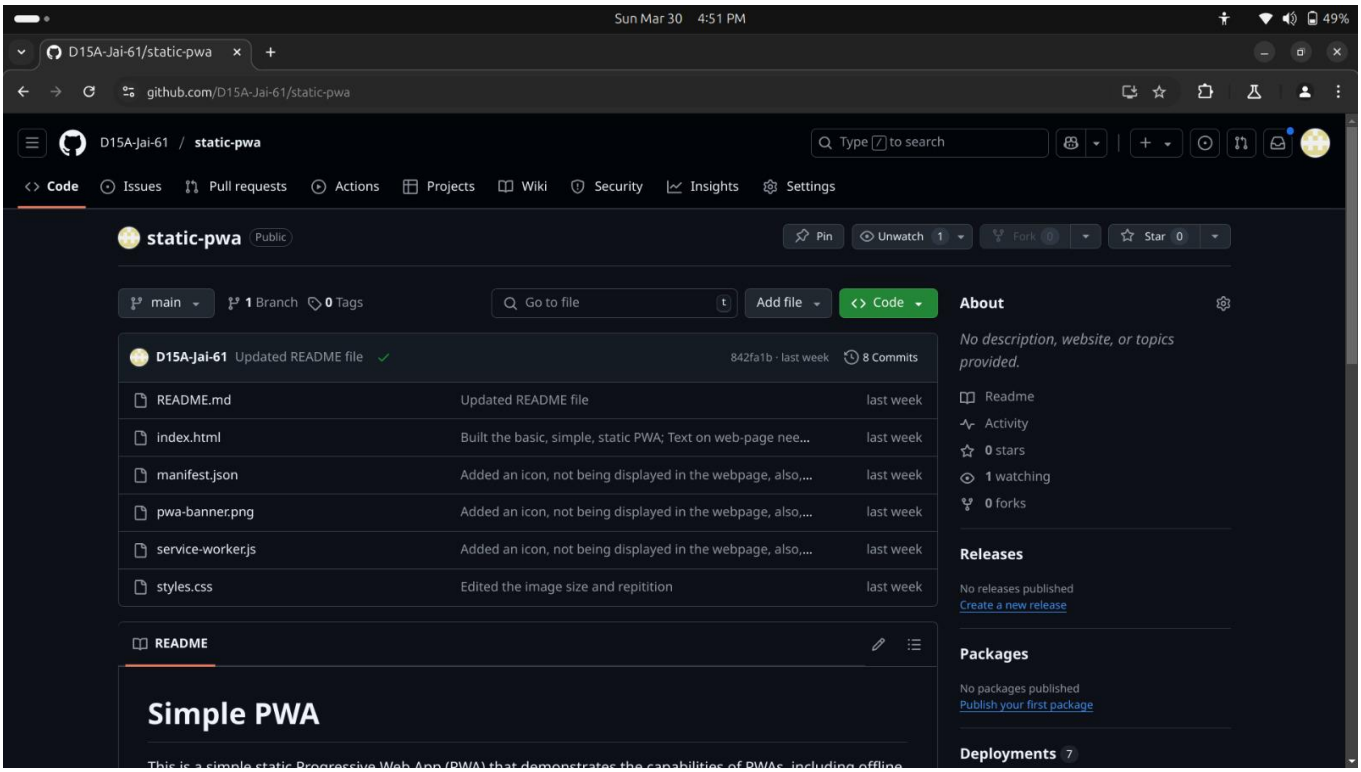
Cons

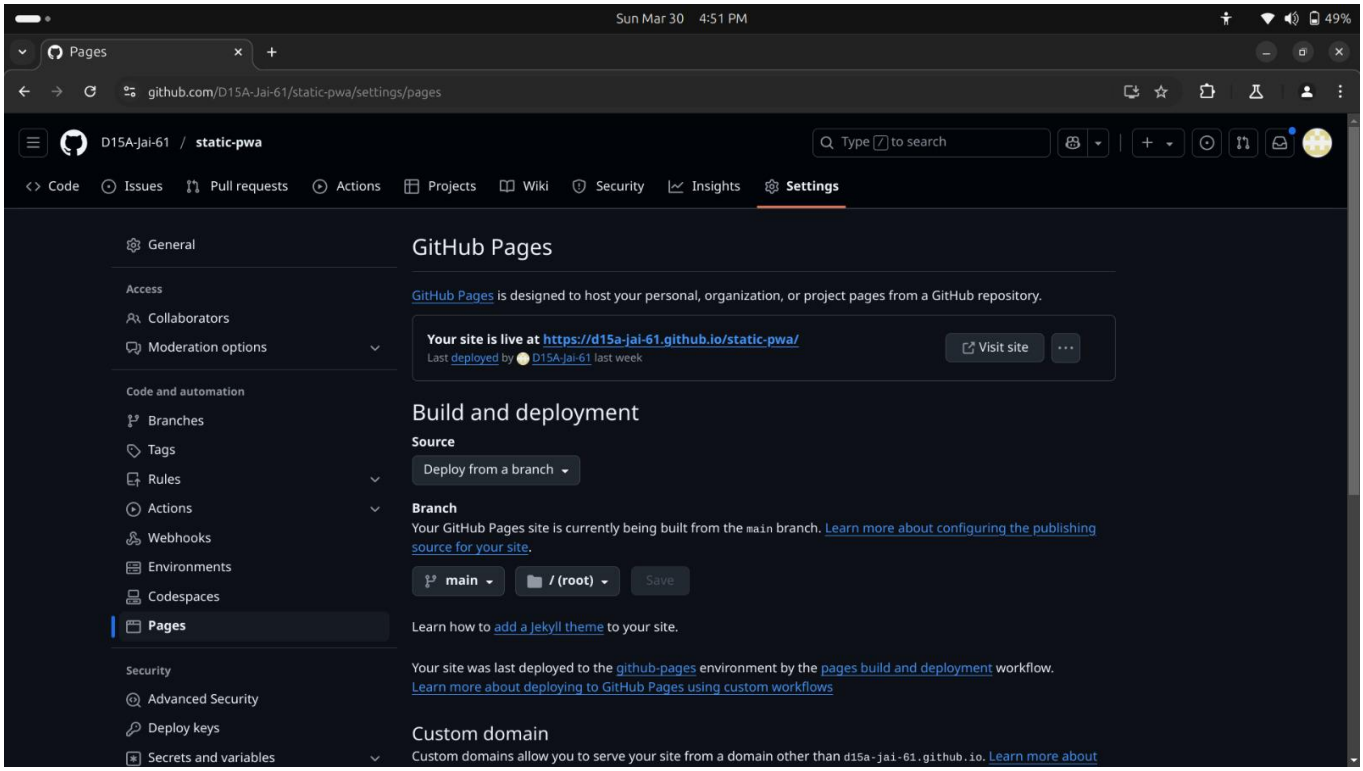
1. Only 10 GB of data transfer is allowed per month. But this is not really a big problem, if you use a CDN or AMP.
2. Command-line interface only.
3. No in-built support for any static site generator.

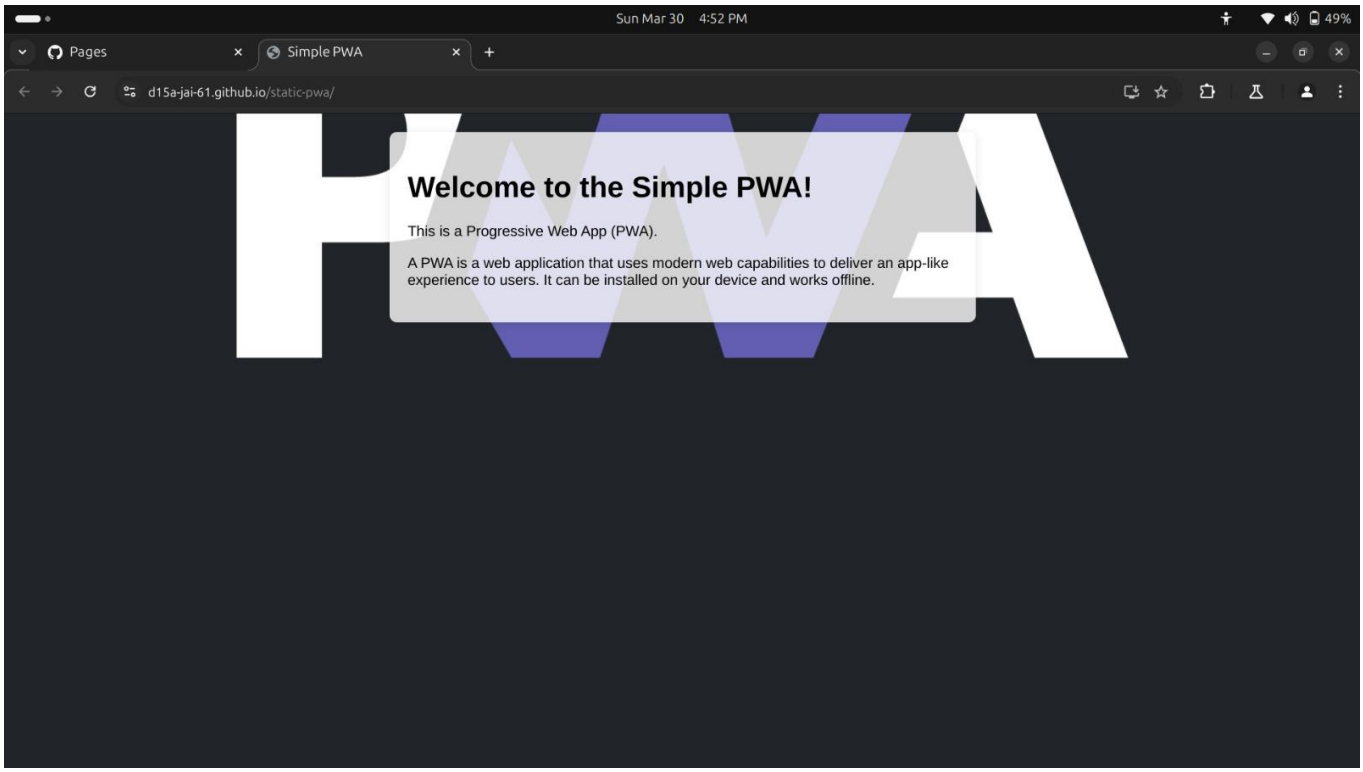
Link to our GitHub repository:

https://github.com/D15A-Jai-61/static-pwa.git

Github Screenshot:


Sun Mar 30 4:51 PM
D15A-Jai-61/static-pwa
github.com/D15A-Jai-61/static-pwa
D15A-Jai-61 / static-pwa
Code Issues Pull requests Actions Projects Wiki Security Insights Settings
static-pwa Public
Pin Unwatch 1 Fork 0 Star 0
main 1 Branch 0 Tags
Go to file
Add file
Code
D15A-Jai-61 Updated README file
842fa1b · last week 8 Commits
README.md Updated README file last week
index.html Built the basic, simple, static PWA; Text on web-page nee... last week
manifest.json Added an icon, not being displayed in the webpage, also,... last week
pwa-banner.png Added an icon, not being displayed in the webpage, also,... last week
service-worker.js Added an icon, not being displayed in the webpage, also,... last week
styles.css Edited the image size and repitition last week
README
Simple PWA
About
No description, website, or topics provided.
Readme
Activity
0 stars
1 watching
0 forks
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
Deployments 7


Sun Mar 30 4:51 PM
Pages
github.com/D15A-Jai-61/static-pwa/settings/pages
D15A-Jai-61 / static-pwa
Code Issues Pull requests Actions Projects Wiki Security Insights Settings
General
Access
Collaborators
Moderation options
Code and automation
Branches
Tags
Rules
Actions
Webhooks
Environments
Codespaces
Pages
Security
Advanced Security
Deploy keys
Secrets and variables
GitHub Pages
GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.
Your site is live at https://d15a-jai-61.github.io/static-pwa/
Last deployed by D15A-Jai-61 last week
Visit site
Build and deployment
Source
Deploy from a branch
Branch
Your GitHub Pages site is currently being built from the main branch. Learn more about configuring the publishing source for your site.
main / (root) Save
Learn how to add a Jekyll theme to your site.
Your site was last deployed to the github-pages environment by the pages build and deployment workflow.
Learn more about deploying to GitHub Pages using custom workflows
Custom domain
Custom domains allow you to serve your site from a domain other than d15a-jai-61.github.io. Learn more about


Sun Mar 30 4:52 PM
49%
Pages
Simple PWA
d15a-jai-61.github.io/static-pwa/
Welcome to the Simple PWA!
This is a Progressive Web App (PWA).
A PWA is a web application that uses modern web capabilities to deliver an app-like experience to users. It can be installed on your device and works offline.

MAD & PWA Lab
Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	62
Name	Vedang V. Wajge
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	

Experiment 11

Aim :

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

Theory :

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

Google Lighthouse :

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

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

Key Features and Audit Metrics

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

Performance: This score is an aggregation of how the page fared in aspects such as (but not limited to) loading speed, time taken for loading for basic frame(s), displaying meaningful content to the user, etc. To a layman, this score is indicative of how decently the site performs, with a score of 100 meaning that you figure in the 98th percentile, 50 meaning that you figure in the 75th percentile and so on.

PWA Score (Mobile): Thanks to the rise of Service Workers, app manifests, etc., a lot of modern web applications are moving towards the PWA paradigm,

where the objective is to make the application behave as close as possible to native mobile applications. Scoring points are based on the Baseline PWA checklist laid down by Google which includes Service Worker implementation(s), viewport handling, offline functionality, performance in script-disabled environments, etc.

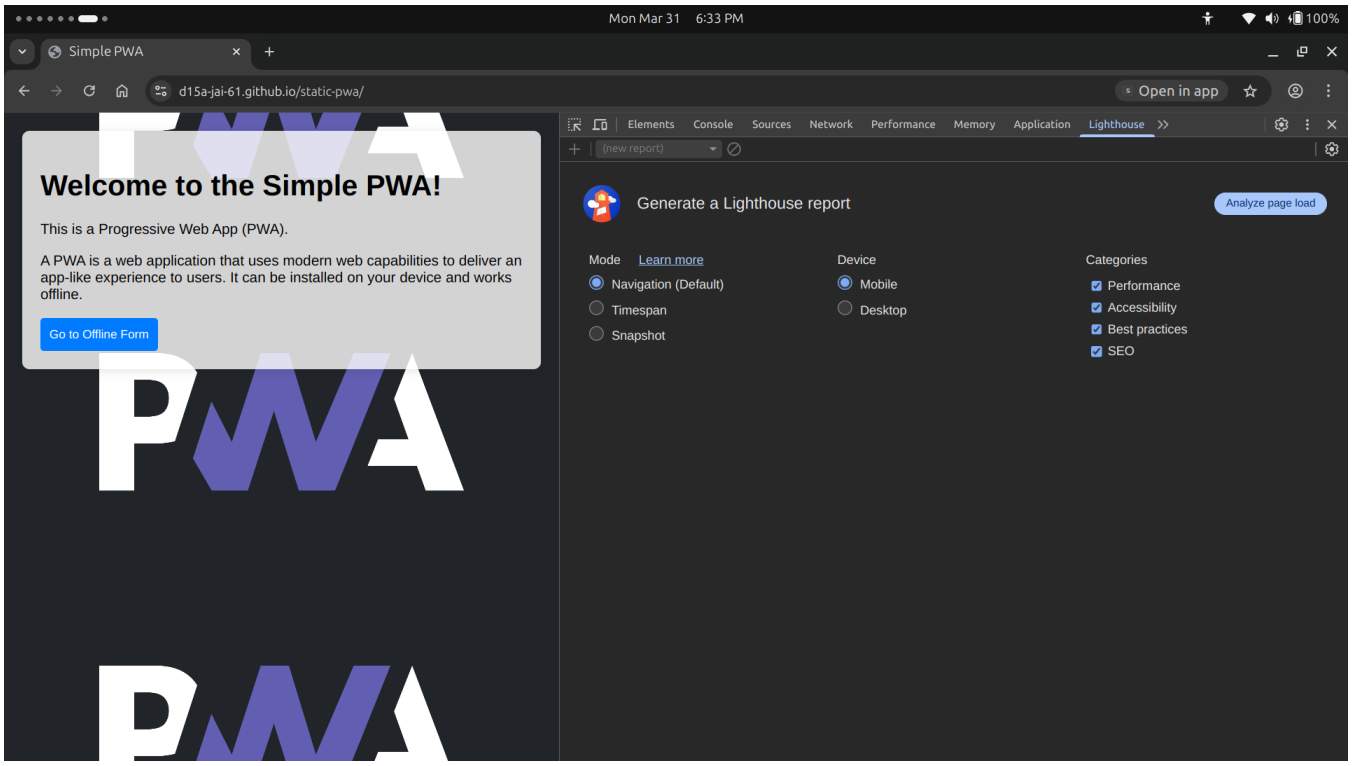
Accessibility: As you might have guessed, this metric is a measure of how accessible your website is, across a plethora of accessibility features that can be implemented in your page (such as the 'aria-' attributes like aria-required, audio captions, button names, etc.). Unlike the other metrics though, Accessibility metrics score on a pass/fail basis i.e. if all possible elements of the page are not screen-reader friendly (HTML5 introduced features that would make pages easy to interpret for screen readers used by visually challenged people like tag names, tags such as <section>, <article>, etc.), you get a 0 on that score. The aggregate of these scores is your Accessibility metric score.

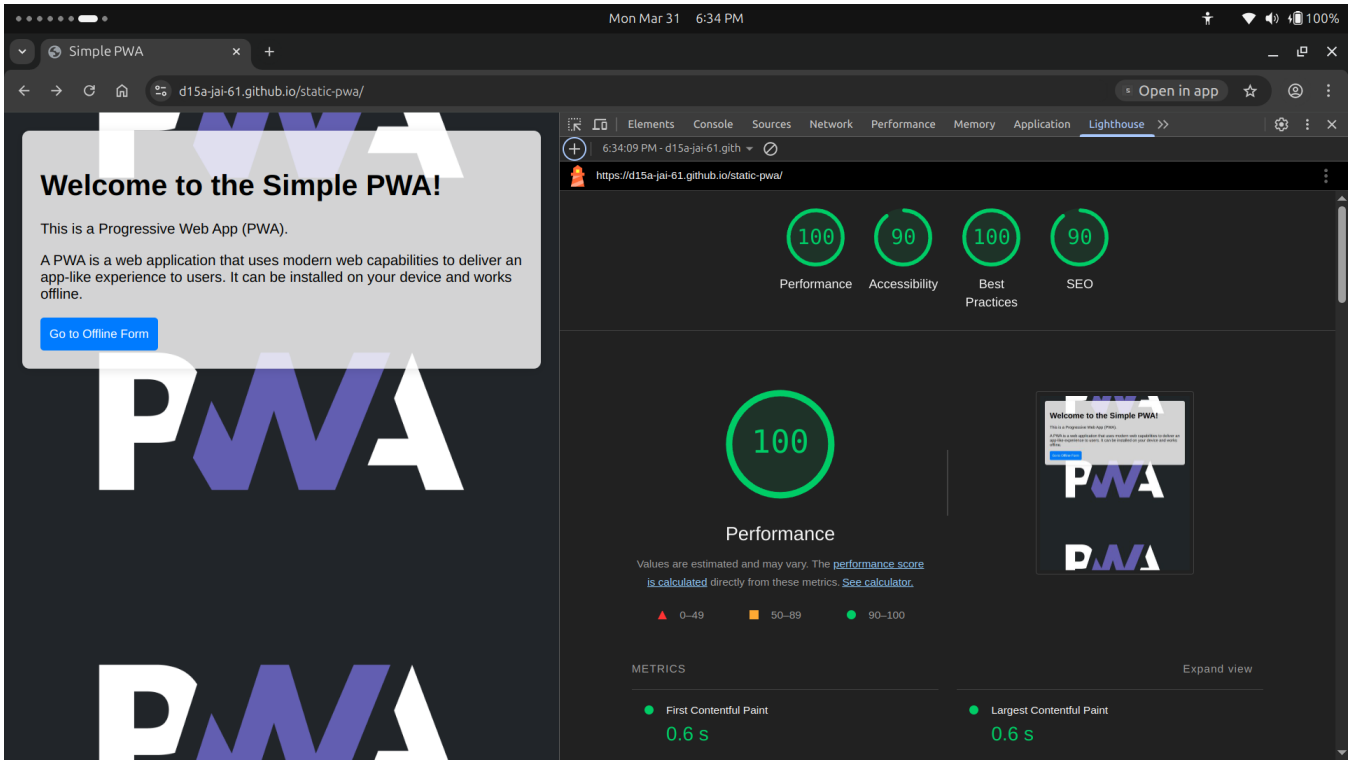
Best Practices: As any developer would know, there are a number of practices that have been deemed 'best' based on empirical data. This metric is an aggregation of many such points, including but not limited to:Use of HTTPS

Avoiding the use of deprecated code elements like tags, directives, libraries, etc. Password input with paste-into disabled

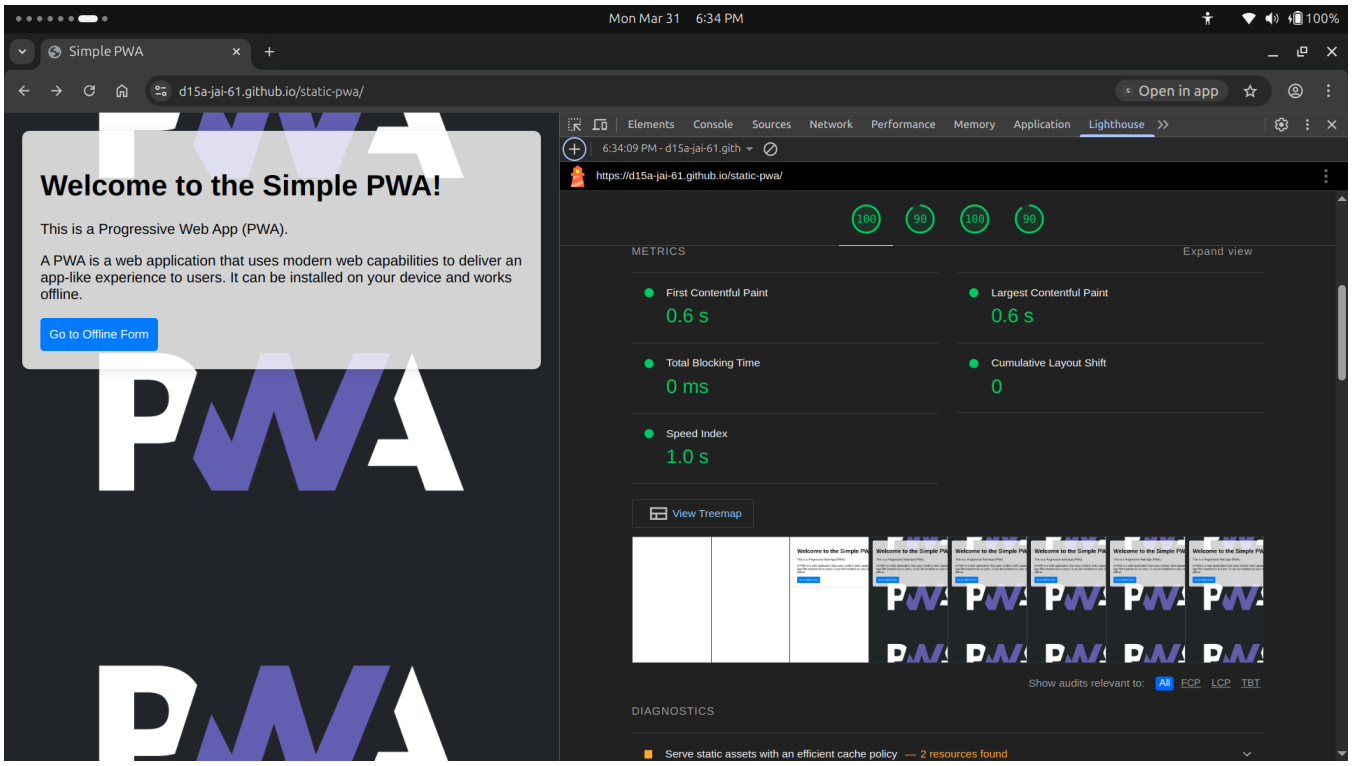
Geo-Location and cookie usage alerts on load, etc.

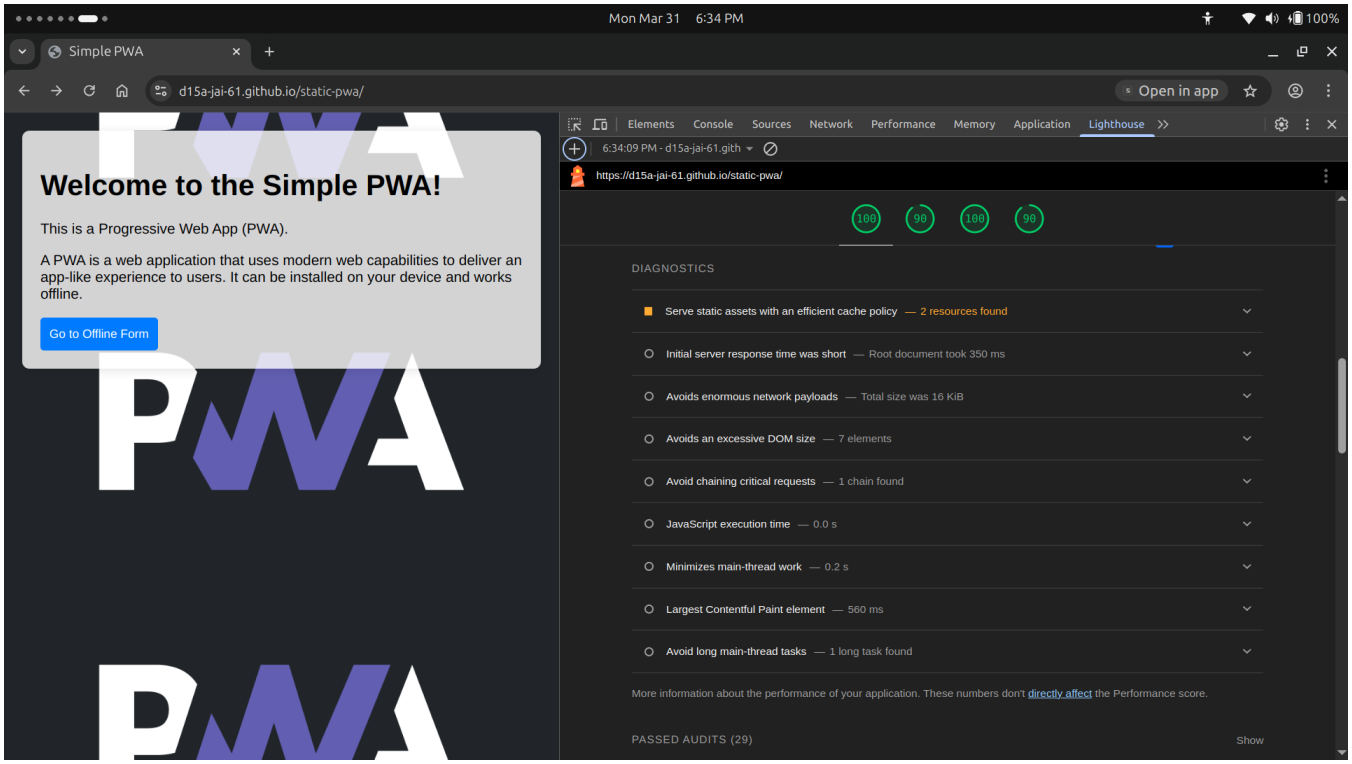
Diksha Utekar Jai Talreja Vedang Wajge D15A-63/59/62



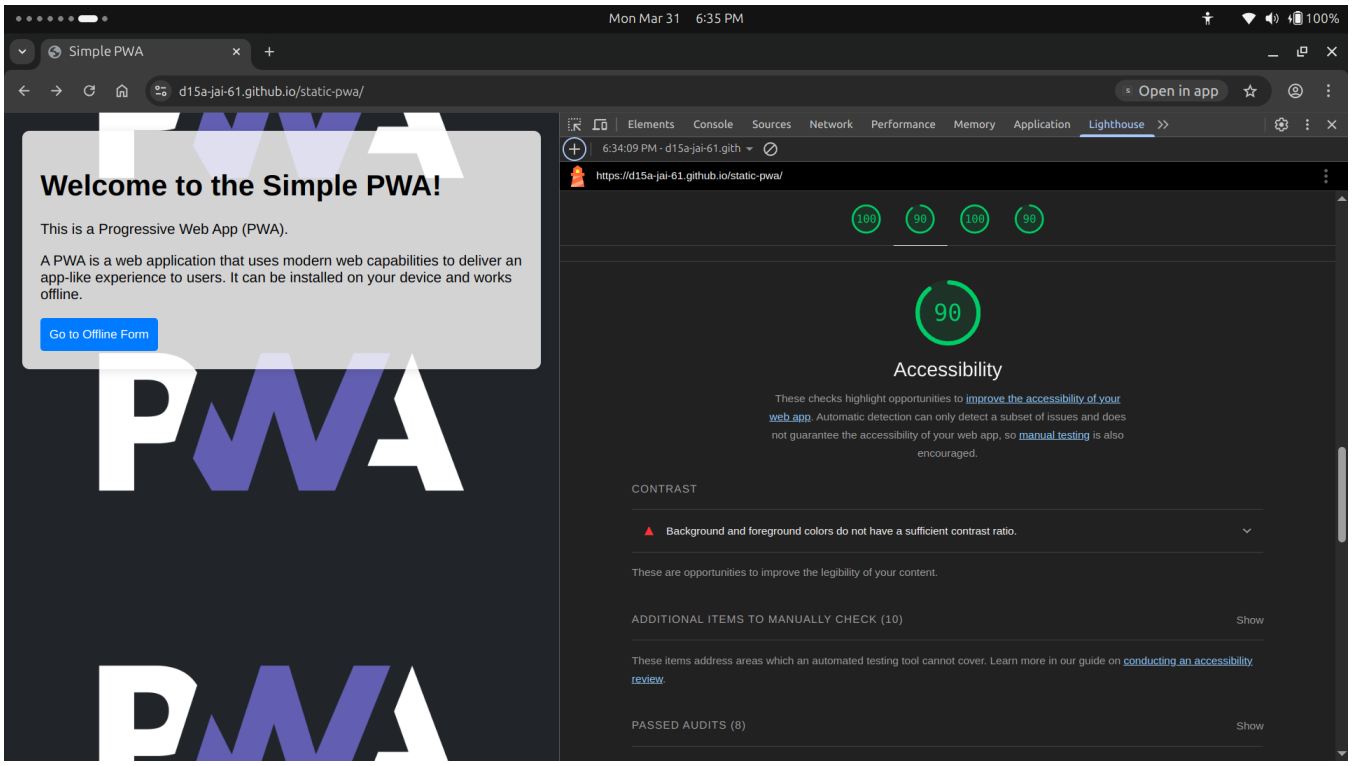


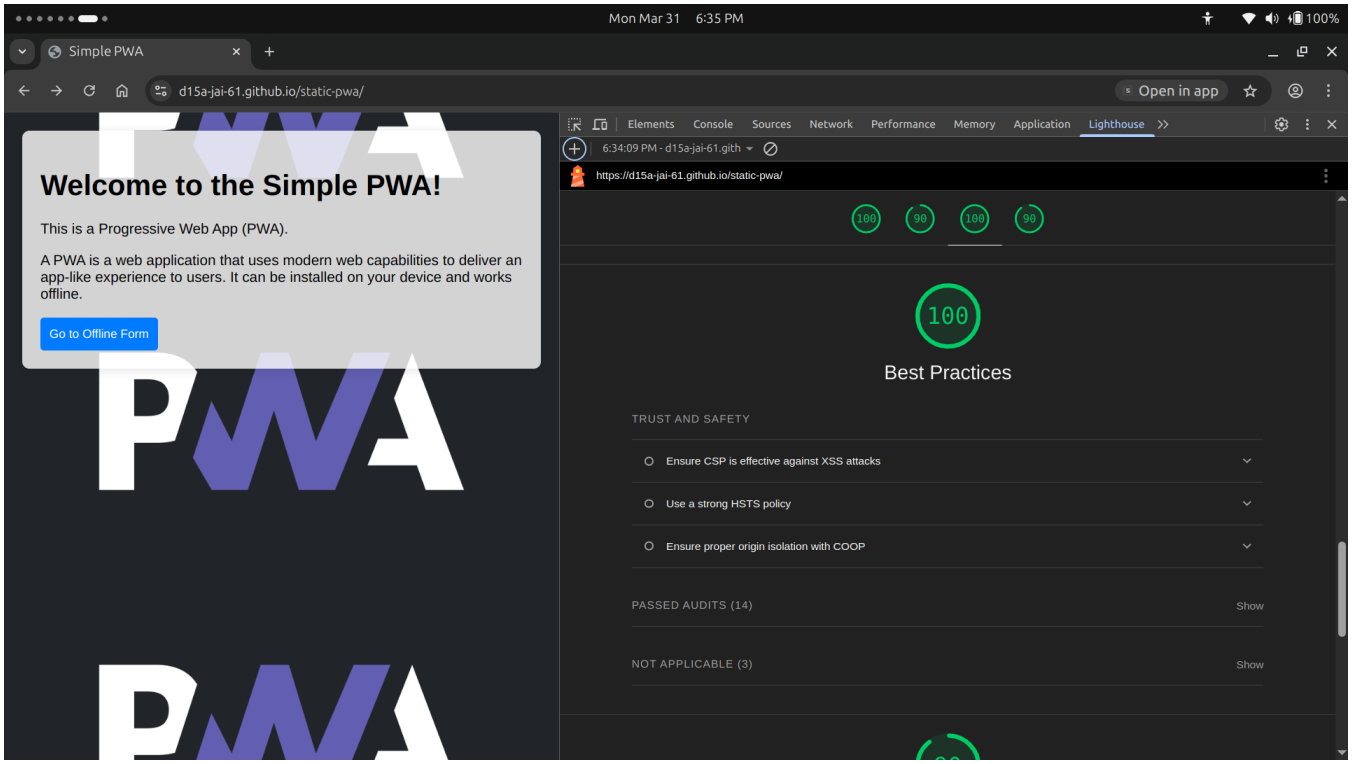
Diksha Utekar Jai Talreja Vedang Wajge D15A-63/59/62

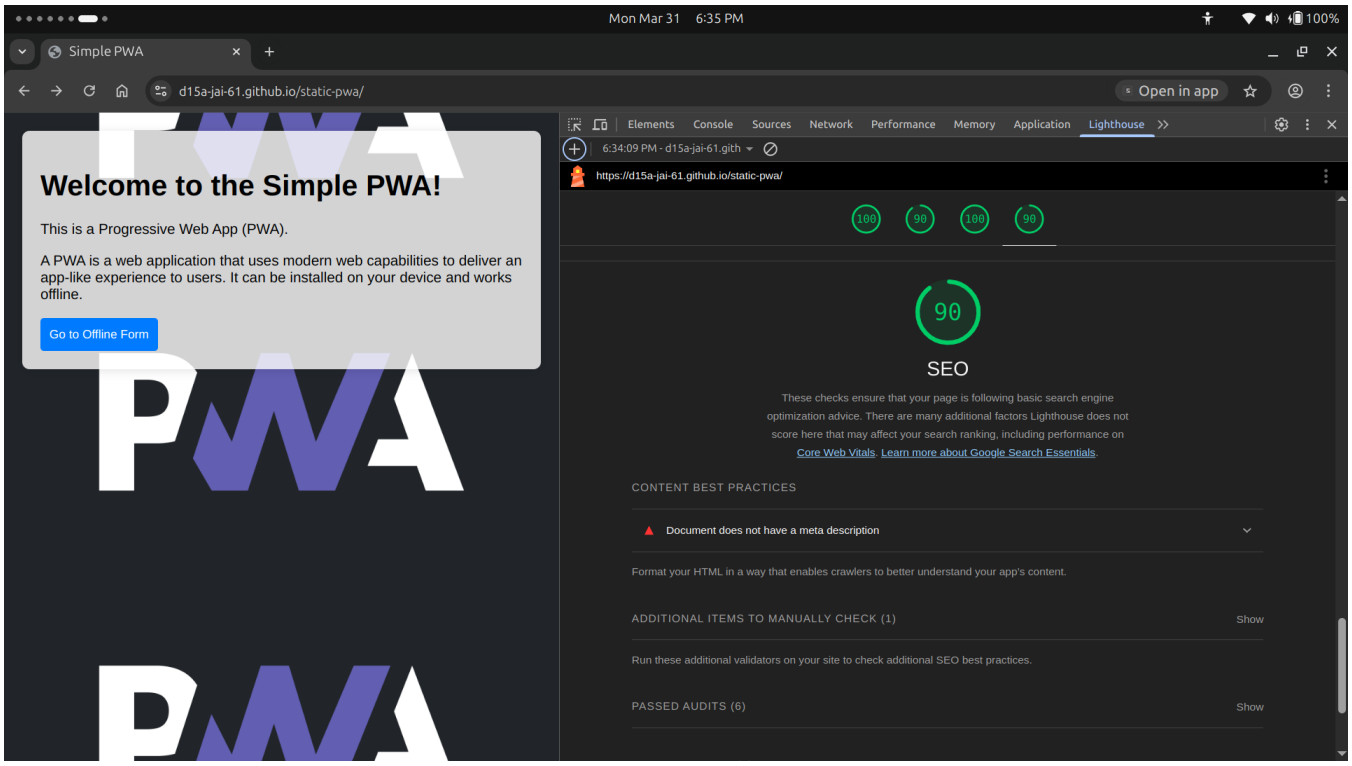




Diksha Utekar Jai Talreja Vedang Wajge D15A-63/59/62







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.	62
Name	Vedang V. Wajge
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.	62
Name	Vedang V. Wajge
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4:Understand various PWA frameworks and their requirements LO5: Design and Develop a responsive User Interface by applying PWA Design techniques LO6:Develop and Analyze PWA Features and deploy it over app hosting solutions
Grade:	