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 **Komal Milind Deolekar** of **D15A** semester **VI,** have successfully completed necessary experiments in the **MAD & PWA Lab** under my supervision in **VES Institute of Technology** during the academic year **2024-2025.**

Lab Assistant

Subject Teacher

Mrs. Kajal Joseph

Principal

Head of Department

Dr. Mrs. Shalu Chopra

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

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

Faculty Incharge : Mrs. Kajal Joseph.

Lab Teachers : Mrs. Kajal Joseph.

Email : kajal.jewani@ves.ac.in

Programme Outcomes: The graduate will be able to:

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

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

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

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

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

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

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

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

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

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

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

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

Program specific Outcomes

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

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

Lab Objectives:

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

Lab Outcomes:

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

Index

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

MAD & PWA Lab
Journal

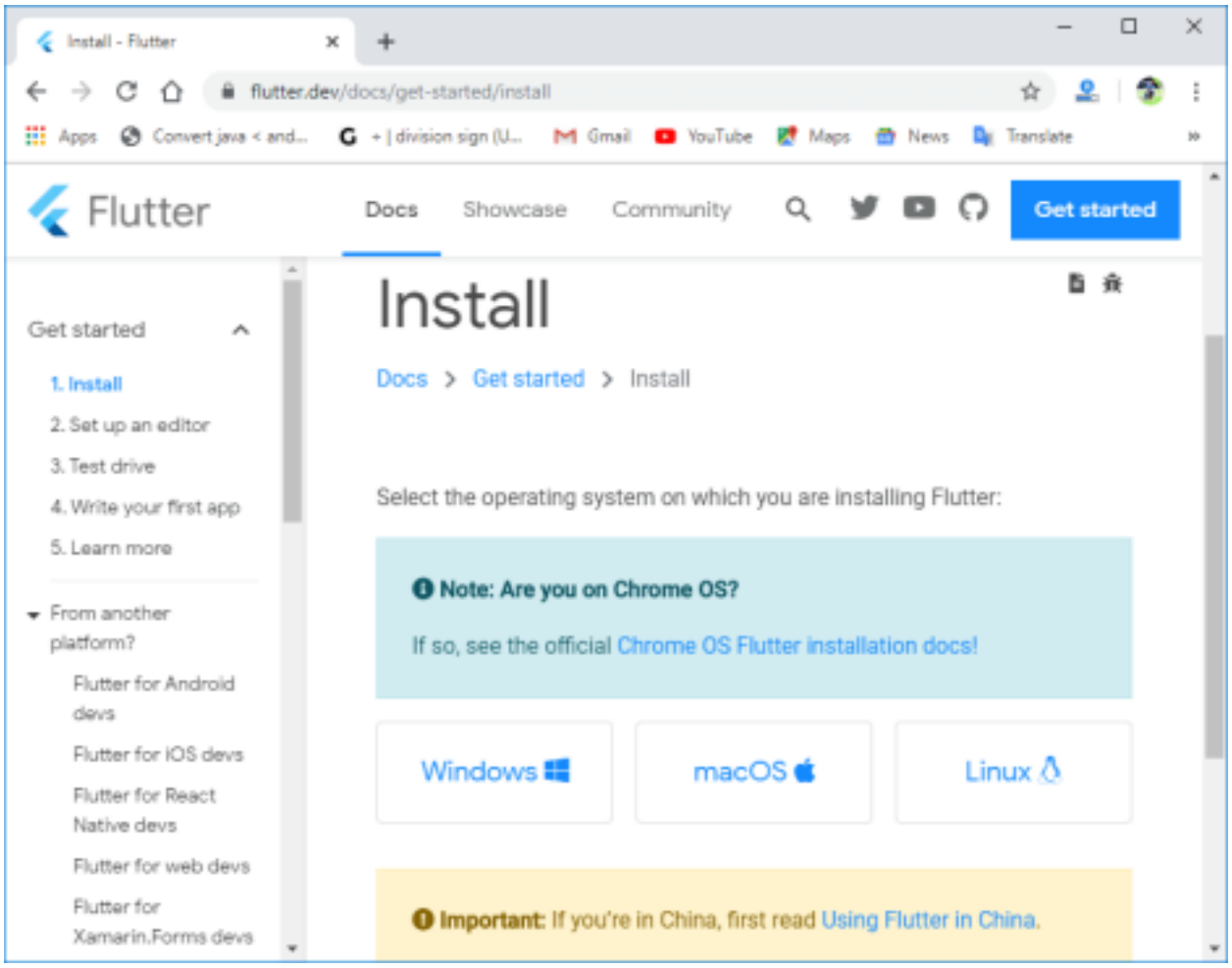
Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	14
Name	Komal Milind Deolekar
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	

AIM : To install and configure the Flutter Environment

Theory :

Install the Flutter SDK

Step 1: Download the installation bundle of the Flutter Software Development Kit for windows. To download Flutter SDK, Go to its official website https://docs.flutter.dev/get-started/install , you will get the following screen.

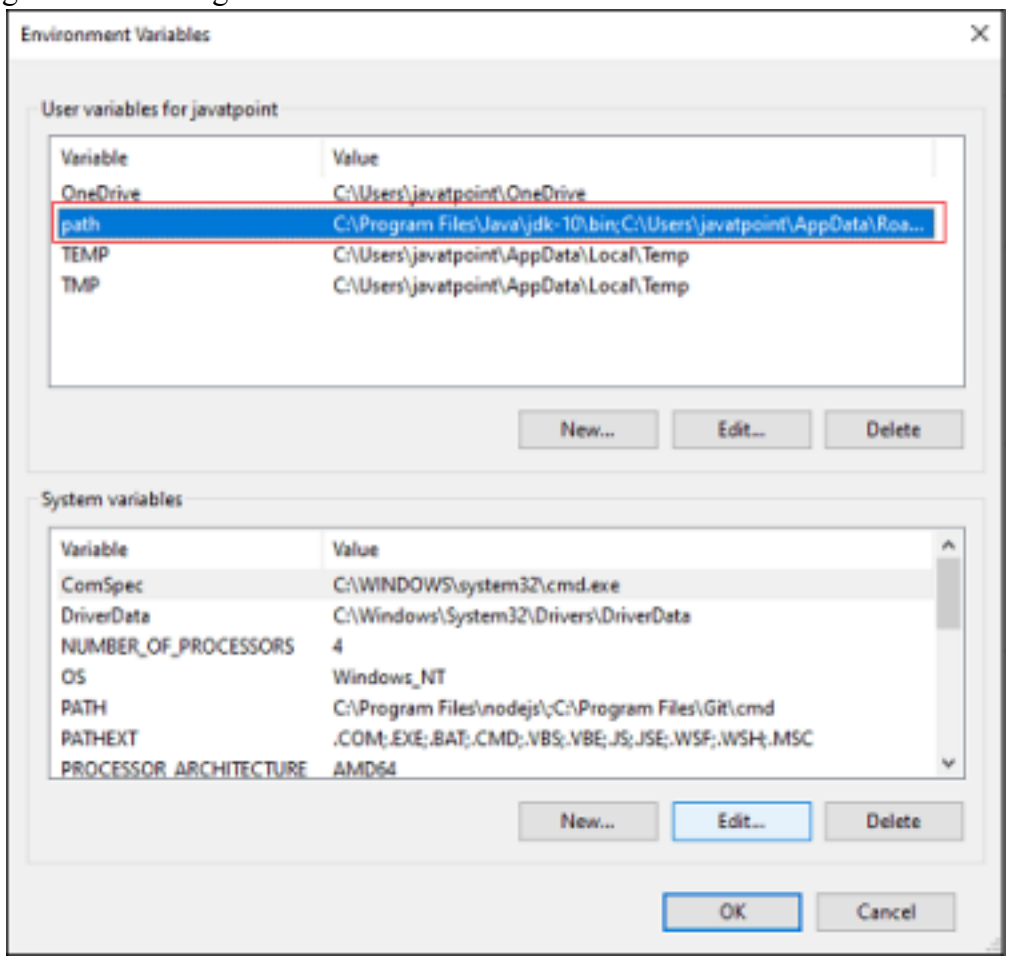


Step 2: Next, to download the latest Flutter SDK, click on the Windows **icon**. Here, you will find the download link for SDK.

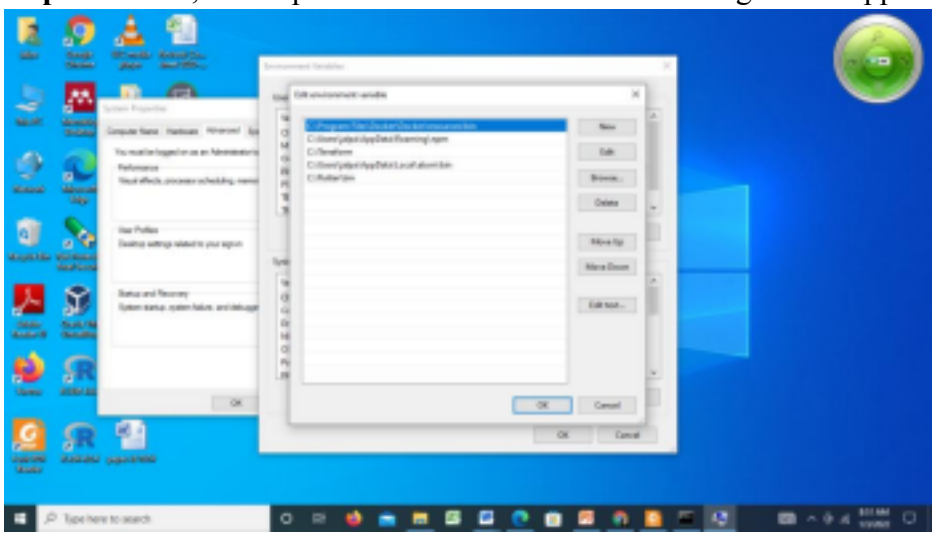
Step 3: When your download is complete, extract the **zip** file and place it in the desired installation folder or location, for example, C: /Flutter.

Step 4: To run the Flutter command in regular windows console, you need to update the system path to include the flutter bin directory. The following steps are required to do this:

Step 4.1: Go to MyComputer properties -> advanced tab -> environment variables. You will get the following screen.

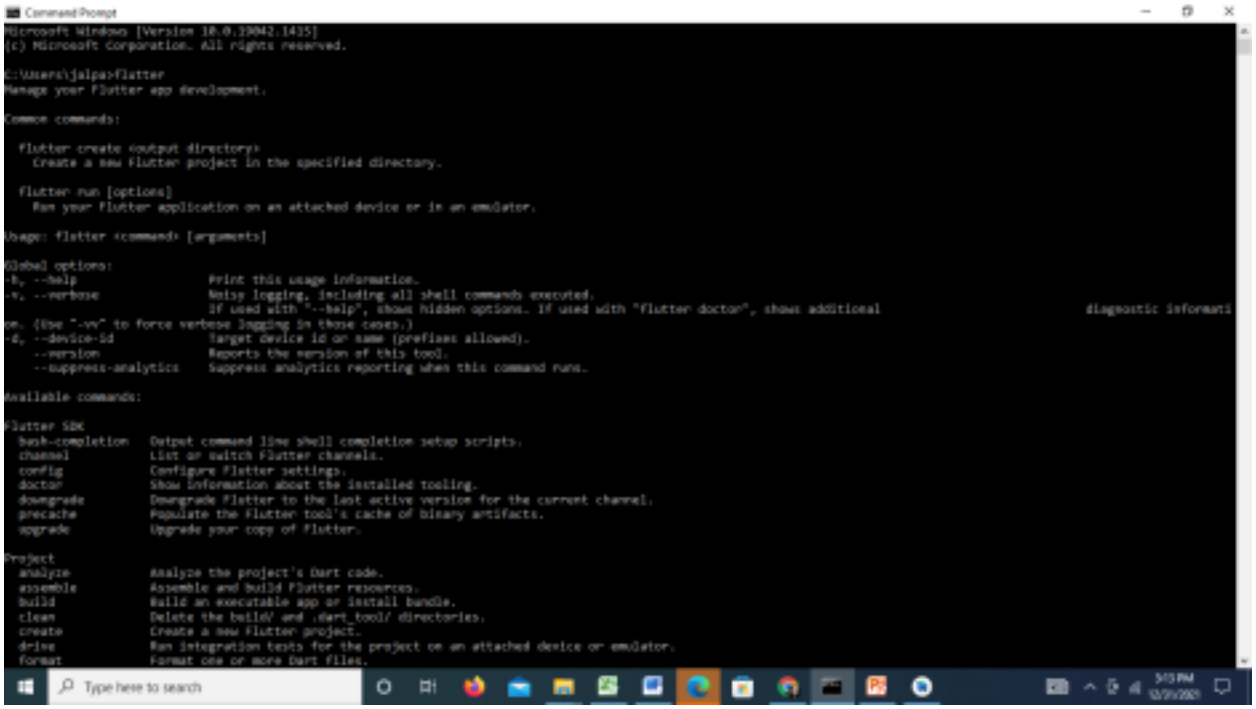


Step 4.2: Now, select path -> click on edit. The following screen appears

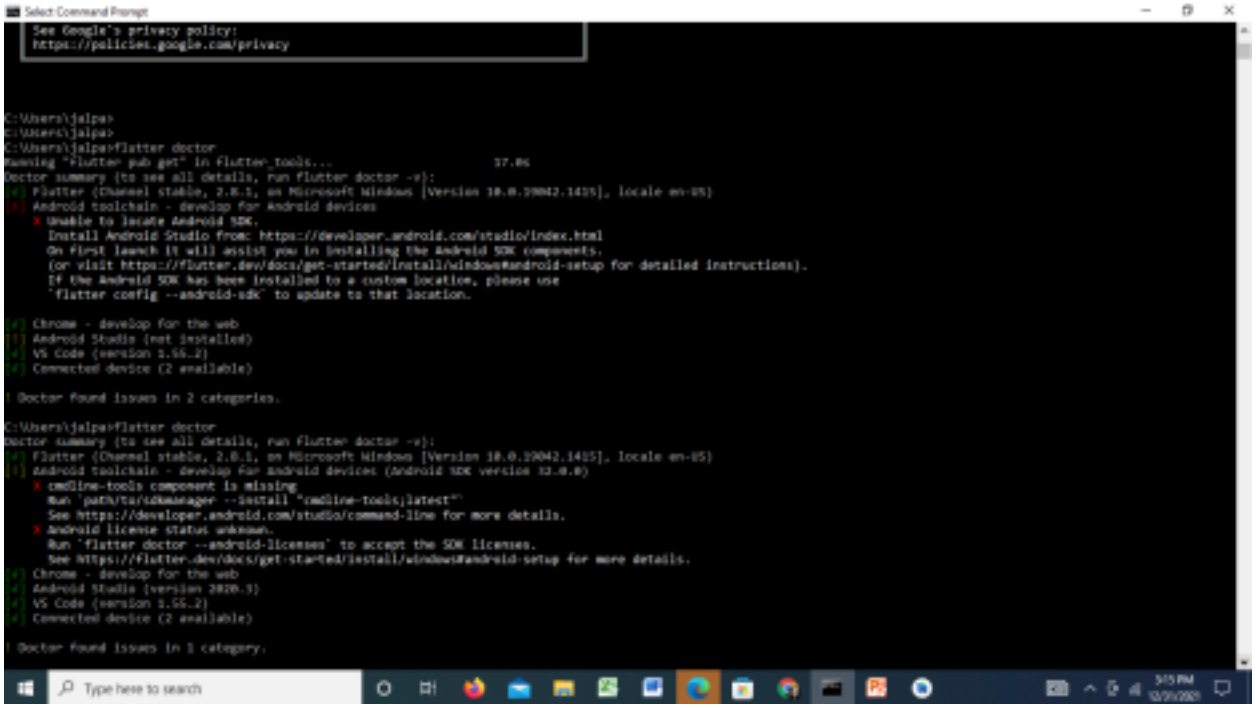


Step 4.3: In the above window, click on New->write path of Flutter bin folder in variable value - > ok -> ok -> ok.

Step 5: Now, run the $ **flutter** command in command prompt.



Now, 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.

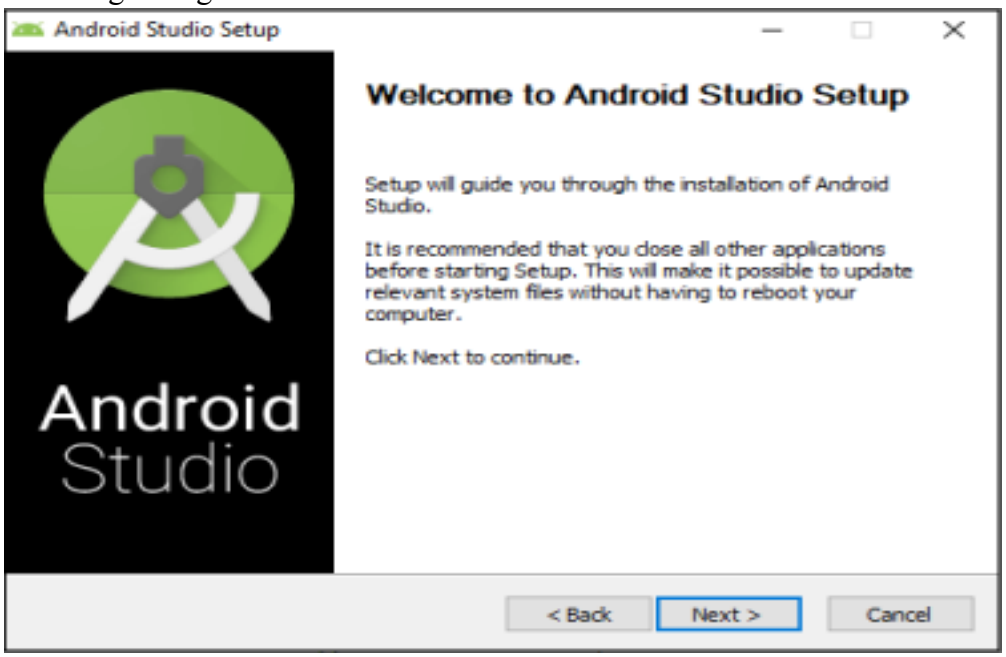


Step 6: When you run the above command, it will analyze the system and show its report, as shown in the below image. Here, you will find the details of all missing tools, which required to run Flutter as well as the development tools that are available but not connected with the device.

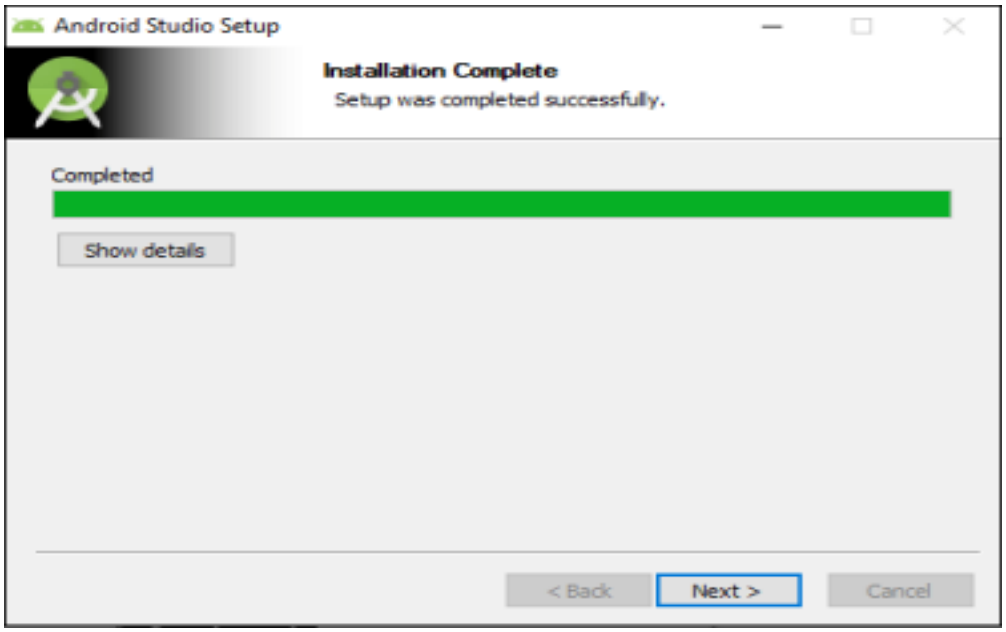
Step 7: Install the Android SDK. If the flutter doctor command does not find the Android SDK tool in your system, then you need first to install the Android Studio IDE. To install Android Studio IDE, do the following steps.

Step 7.1: Download the latest Android Studio executable or zip file from the official site.

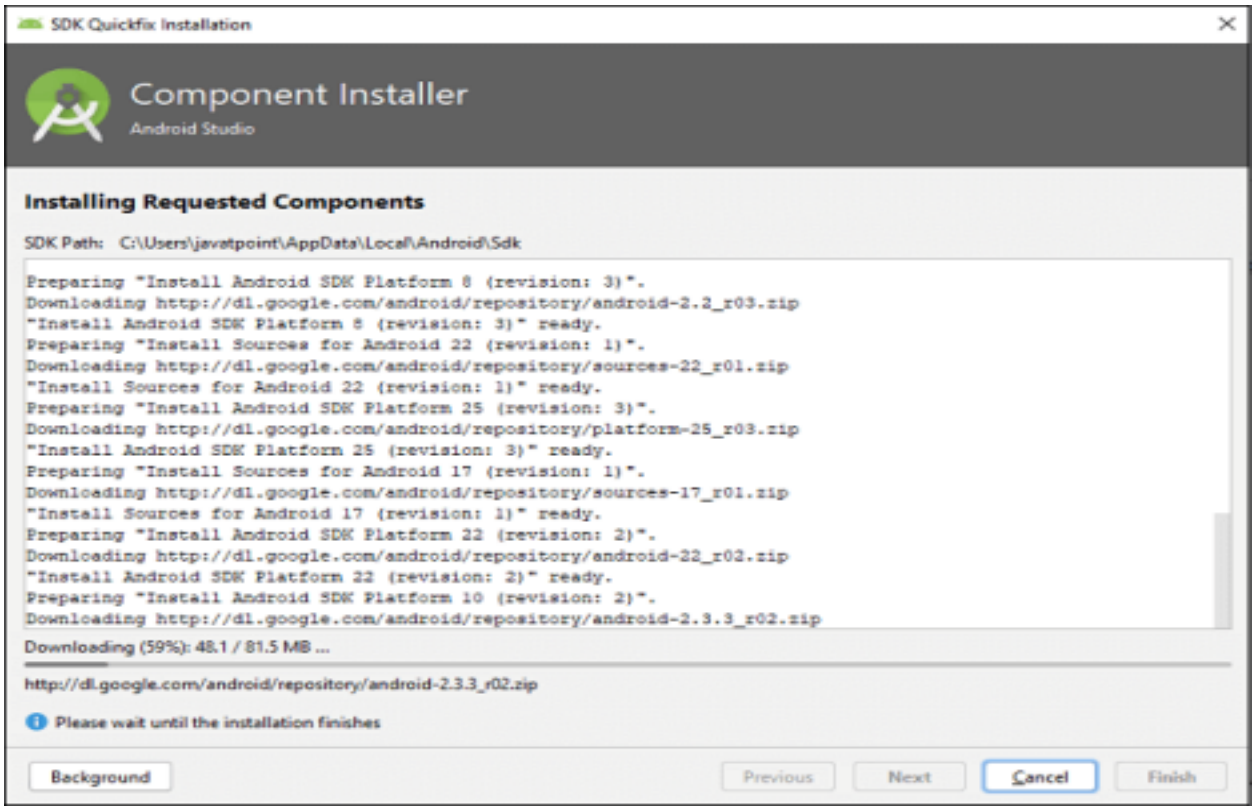
Step 7.2: When the download is complete, open the **.exe** file and run it. You will get the following dialog box.



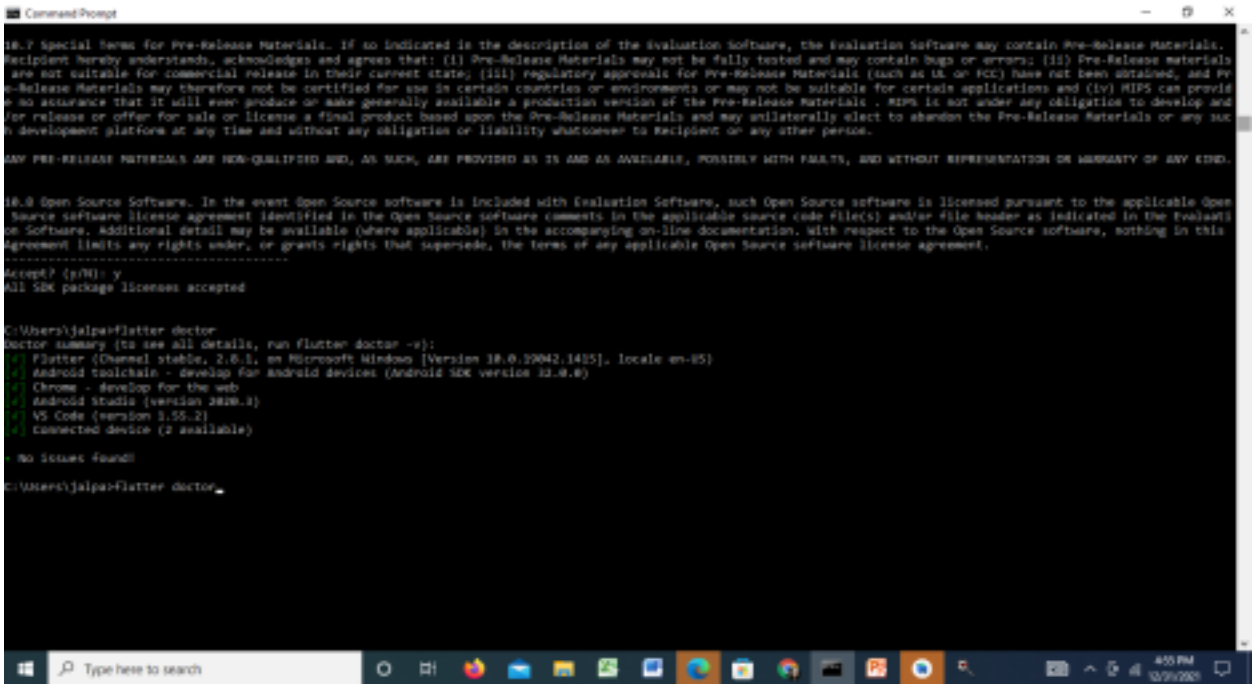
Step 7.3: Follow the steps of the installation wizard. Once the installation wizard completes, you will get the following screen.



Step 7.4: In the above screen, click Next-> Finish. Once the Finish button is clicked, you need to choose the 'Don't import Settings option' and click OK. It will start the Android Studio.



Step 7.5 run the $ **flutter doctor** command and Run flutter doctor --android-licenses command.



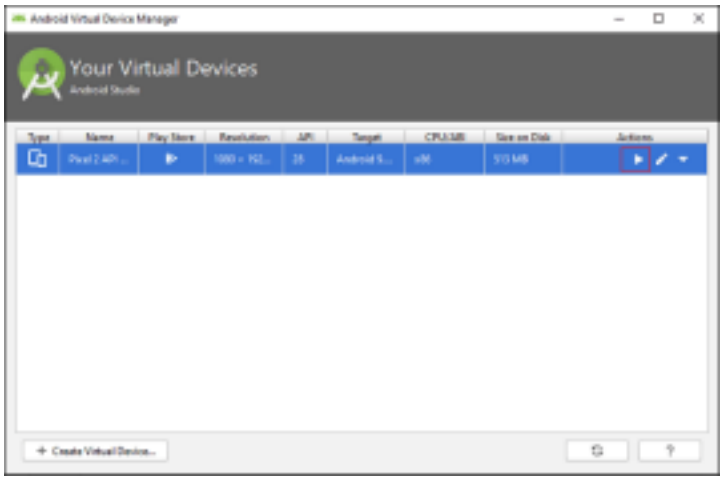
Step 8: Next, you need to set up an Android emulator. It is responsible for running and testing the Flutter application.

Step 8.1: To set an Android emulator, go to Android Studio > Tools > Android > AVD Manager and select Create Virtual Device. Or, go to Help->Find Action->Type Emulator in the search box. You will get the following screen.

Step 8.2: Choose your device definition and click on Next.

Step 8.3: Select the system image for the latest Android version and click on Next.

Step 8.4: Now, verify the all AVD configuration. If it is correct, click on Finish. The following screen appears.

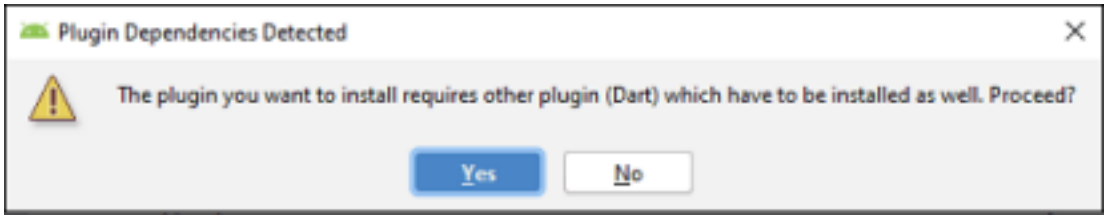


Step 8.5: Last, click on the icon pointed into the red color rectangle. The Android emulator displayed as below screen.

Step 9: 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. Do the following steps to install these plugins.

Step 9.1: Open the Android Studio and then go to File->Settings->Plugins.

Step 9.2: Now, search the Flutter plugin. If found, select Flutter plugin and click install. When you click on install, it will ask you to install Dart plugin as below screen. Click yes to proceed.



Step 9.3: Restart the Android Studio.

MAD & PWA Lab
Journal

Experiment No.	02
Experiment Title.	To design Flutter UI by including common widgets.
Roll No.	14
Name	Komal Milind Deolekar
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

AIM : To design Flutter UI by including common widgets.

Theory :

Flutter is a UI toolkit developed by Google for building natively compiled applications for mobile, web, and desktop from a single codebase. It provides a rich set of widgets that help in designing responsive and visually appealing user interfaces.

Flutter UI Structure

Flutter follows a widget-based architecture where everything is a widget, including the app itself. The UI is built using:

- **MaterialApp**: The root of a Flutter application that provides Material Design features.
- **Scaffold**: Provides a basic layout structure including an AppBar, body, floating action button, etc.
- **Widgets**: The building blocks of the UI.

Types of Widgets in Flutter

A. Basic Widgets

1. **Text**: Displays text with different styles.

```
Text('Hello, Flutter!', style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold))
```

2. **Image**: Displays images from assets or the internet.

```
Image.asset('assets/logo.png') // For local image
Image.network('https://example.com/image.png') // For online image
```

3. **Icon**: Displays an icon.

```
Icon(Icons.home, size: 30, color: Colors.blue)
```

B. Layout Widgets

1. **Container**: A box widget that can hold other widgets.

```
Container(
  padding: EdgeInsets.all(10),
  decoration: BoxDecoration(color: Colors.blue, borderRadius:    BorderRadius.circular(10)),
  child: Text('Hello World', style: TextStyle(color: Colors.white)),
)
```

2. **Column & Row**: Arrange widgets vertically or horizontally.

```
Column(
```

```
  children: [
   Text('Item 1'),
   Text('Item 2'),
  ],
)

Row(
 children: [
  Icon(Icons.star),
  Text('Rating: 5'),
 ],
)
```

3. **Stack**: Overlays widgets on top of each other.

```
Stack(
 children: [
  Image.asset('assets/background.png'),
  Positioned(top: 20, left: 30, child: Text('Overlay Text')),
 ],
)
```

C. Input Widgets

1. **TextField**: Allows text input.

```
TextField(
  decoration: InputDecoration(labelText: 'Enter Name', border: OutlineInputBorder()),
)
```

2. **Button Widgets**:
   a. **ElevatedButton**: Raised button with elevation.

```
ElevatedButton(
 onPressed: () {},
 child: Text('Click Me'),
)
```

   b. **TextButton**: Flat button with no elevation.

```
TextButton(
 onPressed: () {},
 child: Text('Click Me'),
)
```

   c. **IconButton**: A button with an icon.

```
IconButton(
 icon: Icon(Icons.add),
  onPressed: () {},
)
```

D. Navigation & Routing Widgets

1. **Navigator**: Handles screen transitions.

```
Navigator.push(context, MaterialPageRoute(builder: (context) => NewScreen()));
```

2. **BottomNavigationBar**: Used for switching between different screens.

```
BottomNavigationBar(
    items: [
      BottomNavigationBarItem(icon: Icon(Icons.home), label: 'Home'),
      BottomNavigationBarItem(icon: Icon(Icons.settings), label: 'Settings'),
    ],
  )
```

E. Interactive & Gestures

GestureDetector: Detects taps, swipes, and other gestures.

```
GestureDetector(
    onTap: () => print('Tapped!'),
    child: Container(color: Colors.blue, height: 100, width: 100),
  )
```

Responsive Design Considerations

- Use **MediaQuery** for screen size adaptation.
- Use **Expanded & Flexible** widgets for dynamic layouts.
- Implement **ListView** instead of Column for scrollable content.

State Management in UI

- **StatelessWidget**: Used for static UI components.
- **StatefulWidget**: Used for UI components that need updates (e.g., form inputs, animations).

Theming & Styling

Flutter provides **ThemeData** to define a consistent look.

```
ThemeData(
  primaryColor: Colors.blue,
  textTheme: TextTheme(
    bodyText1: TextStyle(fontSize: 18, color: Colors.black),
  ),
)
```

Code :

Home_page.dart

```
import 'package:flutter/material.dart';
import 'package:lucide_icons/lucide_icons.dart';
import 'fashion_store_screen.dart';
import 'food_store_screen.dart';

class HomePage extends StatelessWidget {

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   backgroundColor: Colors.white,
   appBar: AppBar(
    backgroundColor: Colors.blue.shade100,
    elevation: 0,
    title: Row(
     children: [
      Icon(Icons.location_on, color: Colors.pink),
      SizedBox(width: 5),
      Text("Charai", style: TextStyle(fontWeight:
FontWeight.bold)),
      Icon(Icons.keyboard_arrow_down),
     ],
    ),
    actions: [
     Container(
      padding: EdgeInsets.symmetric(horizontal:
10, vertical: 5),
      decoration: BoxDecoration(
       color: Colors.white,
       borderRadius: BorderRadius.circular(12),
      ),
      child: Row(
       children: [
        Text("300", style: TextStyle(fontWeight:
FontWeight.bold)),
        Icon(Icons.monetization_on, color:
Colors.yellow),
       ],
      ),
     ),
     SizedBox(width: 10),
     IconButton(icon: Icon(Icons.notifications),
onPressed: () {}),
    ],
   ),
   body: SingleChildScrollView(
    child: Column(
     crossAxisAlignment:
CrossAxisAlignment.start,
     children: [
      // Search Bar
      Padding(
       padding: EdgeInsets.all(12),
       child: TextField(
        decoration: InputDecoration(
         prefixIcon: Icon(Icons.search),
         hintText: "Search for stores, products,
brands...",
         filled: true,
         fillColor: Colors.grey[200],
         border: OutlineInputBorder(
          borderRadius:
BorderRadius.circular(12),
          borderSide: BorderSide.none,
         ),
        ),
       ),
      ),

      // Banner Section
      Padding(
       padding: EdgeInsets.all(12),
       child: ClipRRect(
        borderRadius: BorderRadius.circular(12),
        child:
Image.asset("assets/images/pizza_banner.jpg"), //
Change image
       ),
      ),


      Padding(
       padding: EdgeInsets.symmetric(horizontal:
12),
       child: Column(
        crossAxisAlignment:
CrossAxisAlignment.start,
        children: [
         GridView.count(
          shrinkWrap: true,
          physics: NeverScrollableScrollPhysics(),
          crossAxisCount: 2,
          childAspectRatio: 1.5,
          crossAxisSpacing: 10,
          mainAxisSpacing: 10,
          children: [
           categoryCard("Fashion",
"assets/images/fashion.png", "Going out",
FashionStoreScreen(), context),
           categoryCard("Food Delivery",
"assets/images/food_delivery.png", "Deals at ₹9",
FoodStoreScreen(), context),
```

```
                categoryCard2("Budget Dining",
"assets/images/budget_dining.png", "Going out" ),
                categoryCard2("Online Vouchers",
"assets/images/vouchers.png", "" ),
                categoryCard2("Fine Dining",
"assets/images/fine_dining.png", "UP TO 70% OFF"
),
                categoryCard2("The ₹1 Shop",
"assets/images/shop.png", "magicShop" ),
              ],
            ),
            Align(

              alignment: Alignment.center,
              child: TextButton(
                onPressed: () =>
showMoreCategories(context),
                child: Text("View All", style:
TextStyle(fontSize: 16, fontWeight:
FontWeight.bold)),
              ),
            ),

                    ],
        ),
      ),

      // Rewards Section
      Container(
        margin: EdgeInsets.all(12),
        padding: EdgeInsets.all(10),
        decoration: BoxDecoration(
          color: Colors.grey.shade200,
          borderRadius: BorderRadius.circular(12),
        ),
        child: Row(
          mainAxisAlignment:
MainAxisAlignment.spaceBetween,
          children: [
            Column(
              crossAxisAlignment:
CrossAxisAlignment.start,
              children: [
                Text("1 Pending Rewards", style:
TextStyle(fontWeight: FontWeight.bold)),
                Text("Your scratch cards are expiring
soon..."),
              ],
            ),
            ElevatedButton(
              onPressed: () {},
              child: Text("Claim Now"),
              style: ElevatedButton.styleFrom(
                backgroundColor:
Colors.blue.shade500,
                foregroundColor: Colors.white,
              ),
            )
          ],
        ),
      ),

      // Promotional Banner
      Padding(
        padding: EdgeInsets.all(12),
        child: ClipRRect(
          borderRadius: BorderRadius.circular(12),
          child:
Image.asset("assets/images/promo_banner.png"), //
Add new promo banner
        ),
      ),
    ],
  ),
 ),
);
}

 Widget categoryCard2(String title, String
imagePath, String subtitle) {
  return Container(
   decoration: BoxDecoration(
    color: Colors.white,
    borderRadius: BorderRadius.circular(12),
    border: Border.all(color: Colors.grey.shade300),
   ),
   padding: EdgeInsets.all(10),
   child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
     if (subtitle.isNotEmpty)
      Container(
       padding: EdgeInsets.symmetric(horizontal:
8, vertical: 3),
       decoration: BoxDecoration(
        color: Colors.blue.shade100,
        borderRadius: BorderRadius.circular(8),
       ),
       child: Text(subtitle, style:
TextStyle(fontSize: 12, fontWeight:
FontWeight.bold)),
      ),
     SizedBox(height: 5),
     Text(title, style: TextStyle(fontSize: 16,
fontWeight: FontWeight.bold)),
     Spacer(),
     Align(
      alignment: Alignment.bottomRight,
      child: Image.asset(imagePath, height: 50),
     ),
    ],
   ),
```

```
    );
  }

  Widget categoryCard(String title, String imagePath,
String subtitle, Widget page, BuildContext context) {
    return GestureDetector(
      onTap: () => Navigator.push(
        context, MaterialPageRoute(builder: (context)
=> page),
      ),
      child: Container(
        decoration: BoxDecoration(
          color: Colors.white,
          borderRadius: BorderRadius.circular(12),
          border: Border.all(color:
Colors.grey.shade300),
        ),
        padding: EdgeInsets.all(10),
        child: Column(
          crossAxisAlignment:
CrossAxisAlignment.start,
          children: [
            if (subtitle.isNotEmpty)
              Container(
                padding: EdgeInsets.symmetric(horizontal:
8, vertical: 3),
                decoration: BoxDecoration(
                  color: Colors.blue.shade100,
                  borderRadius: BorderRadius.circular(8),
                ),
                child: Text(subtitle, style:
TextStyle(fontSize: 12, fontWeight:
FontWeight.bold)),
              ),
            SizedBox(height: 5),
            Text(title, style: TextStyle(fontSize: 16,
fontWeight: FontWeight.bold)),
            Spacer(),
            Align(
              alignment: Alignment.bottomRight,
              child: Image.asset(imagePath, height: 50),
            ),
          ],
        ),
      ),
    );
  }

  void showMoreCategories(BuildContext context) {
    showModalBottomSheet(
      context: context,
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.vertical(top:
Radius.circular(16)),
      ),
      builder: (context) {
        return Padding(
          padding: const EdgeInsets.all(16.0),
          child: Column(
            mainAxisSize: MainAxisSize.min,
            children: [
              Text("More Categories", style:
TextStyle(fontSize: 18, fontWeight:
FontWeight.bold)),
              SizedBox(height: 10),
              GridView.count(
                shrinkWrap: true,
                crossAxisCount: 2,
                childAspectRatio: 1.5,
                crossAxisSpacing: 10,
                mainAxisSpacing: 10,
                children: [
                  categoryCard2("Budget Dining",
"assets/images/budget_dining.png", "Going out"),
                  categoryCard2("Online Vouchers",
"assets/images/vouchers.png", ""),
                  categoryCard2("Fine Dining",
"assets/images/fine_dining.png", "UP TO 70%
OFF"),
                  categoryCard2("The ₹1 Shop",
"assets/images/shop.png", "magicShop"),
                ],
              ),
            ],
          ),
        );
      },
    );
  }

}
```

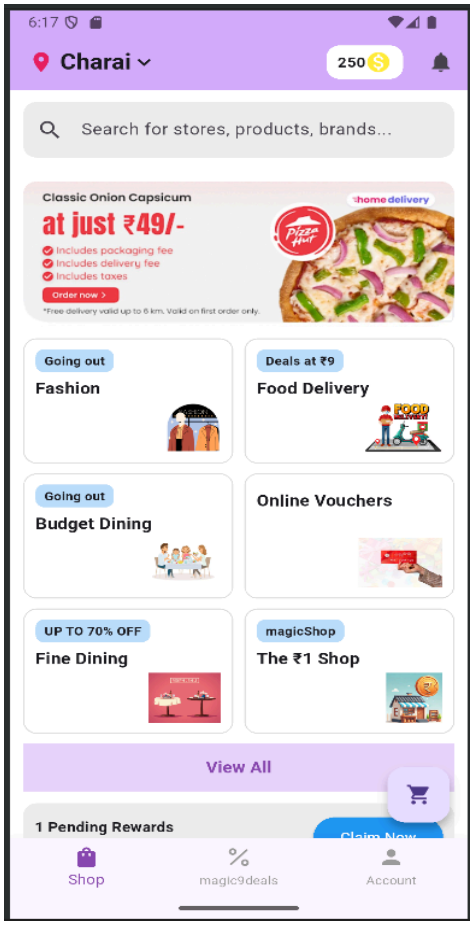
Magic9deals.dart

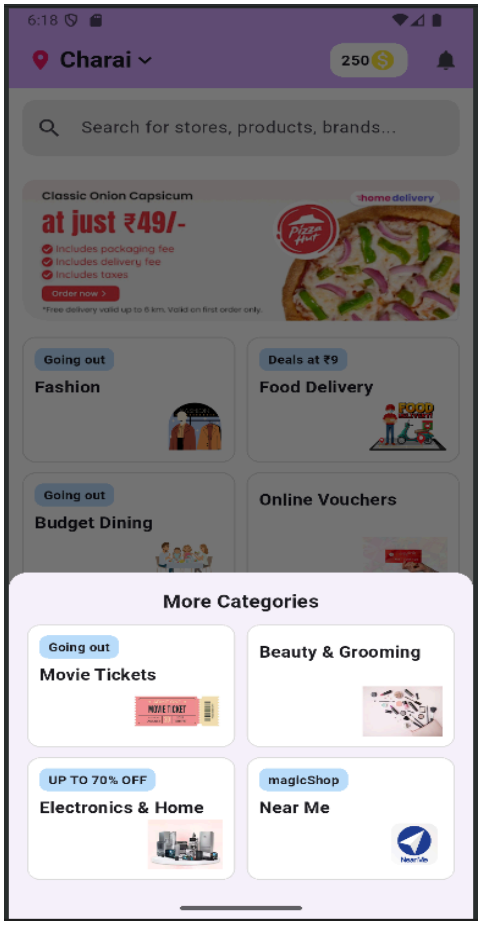
```
import 'package:flutter/material.dart';
import 'footer.dart';
import 'magic9_header.dart';
import 'user_provider.dart';
import 'package:provider/provider.dart';

class Magic9Deals extends StatefulWidget {
  @override
  _Magic9DealsState createState() =>
_Magic9DealsState();
}

class _Magic9DealsState extends
State<Magic9Deals> {

// class Magic9Deals extends StatelessWidget {
  @override
  void initState() {
    super.initState();
    Provider.of<UserProvider>(context, listen:
false).fetchUserData();
  }

// }
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.white,
      appBar: AppBar(
        backgroundColor: Color.fromRGBO(212, 171,
250,1),
        title: Column(
          crossAxisAlignment:
CrossAxisAlignment.start,

          children: [
            Text("Dombivli Railway Station", style:
TextStyle(fontSize: 16, fontWeight:
FontWeight.bold)),
            Text("Mumbai", style: TextStyle(fontSize: 12,
color: Colors.grey)),
          ],
        ),
        actions: [
          Consumer<UserProvider>(
            builder: (context, userProvider, child) {
              return Row(
                children: [
                  Container(
                    padding:
EdgeInsets.symmetric(horizontal: 10, vertical: 5),
                    decoration: BoxDecoration(
                      color: Colors.white,
                      borderRadius:
BorderRadius.circular(12),
                    ),
                    child: Row(
                      children: [
                        Text("${userProvider.magicPoints}",
style: TextStyle(fontWeight: FontWeight.bold)),
                        Icon(Icons.monetization_on, color:
Colors.yellow),
                      ],
                    ),
                  ),
                  SizedBox(width: 10),
                  IconButton(icon:
Icon(Icons.notifications), onPressed: () {}),
                ],
              );
            },
          ),
          IconButton(icon: Icon(Icons.search),
onPressed: () {}),
          IconButton(icon: Icon(Icons.notifications),
onPressed: () {}),
        ],
      ),
      body: ListView(
        padding: EdgeInsets.all(16),
        children: [
          Magic9Header(),
          SizedBox(height: 16),

          // Updated Category Cards with images on the
right
          CategoryCard(
            title: "Food Delivery",
            subtitle: "Menu starting at ₹9",
            buttonText: "Order now",
            imageUrl:
"assets/images/magic9_food_delivery.png", // Update
with actual image path
            bgColor: Colors.redAccent,
          ),
          CategoryCard(
            title: "Budget Dining",
            subtitle: "Deals starting at ₹9",
            buttonText: "Explore now",
            imageUrl:
"assets/images/magic9_budget_dining.png",
            bgColor: Colors.orangeAccent,
          ),
          CategoryCard(
            title: "Fashion",
            subtitle: "Bestsellers starting at ₹49",
            buttonText: "Visit store",
            imageUrl:
```

```
"assets/images/magic9_fashion.png",
          bgColor: Colors.purpleAccent,
        ),
        CategoryCard(
          title: "Pubs & Bars",
          subtitle: "Beer deals starting at ₹79",
          buttonText: "Explore now",
          imageUrl:
"assets/images/magic9_pubs_and_bar.png",
          bgColor: Colors.blueAccent,
        ),

        SizedBox(height: 16),
        AppFooter(),
      ],
    ),
    floatingActionButton: FloatingActionButton(
      onPressed: () {
        Navigator.pushNamed(context, '/cart');
      },
      child: Icon(Icons.shopping_cart),
    ),
  );
 }
}

// Updated CategoryCard with Image on the Right
class CategoryCard extends StatelessWidget {
 final String title;
 final String subtitle;
 final String buttonText;
 final String imageUrl;
 final Color bgColor;

 CategoryCard({
  required this.title,
  required this.subtitle,
  required this.buttonText,
  required this.imageUrl,
  required this.bgColor,
 });

 @override
 Widget build(BuildContext context) {
  return Card(
   shape: RoundedRectangleBorder(borderRadius:
BorderRadius.circular(16)),
   margin: EdgeInsets.symmetric(vertical: 8),
   elevation: 4,
   child: Container(
    decoration: BoxDecoration(
     borderRadius: BorderRadius.circular(16),
     color: bgColor.withOpacity(0.15),
    ),
    child: Padding(
     padding: EdgeInsets.all(12),
     child: Row(
      children: [
       // Expanded Text Section (Left Side)
       Expanded(
        child: Column(
         crossAxisAlignment:
CrossAxisAlignment.start,
         children: [
          Text(title, style: TextStyle(fontSize: 18,
fontWeight: FontWeight.bold)),
          SizedBox(height: 4),
          Text(subtitle, style: TextStyle(fontSize:
14, color: Colors.black54)),
          SizedBox(height: 8),
          ElevatedButton(
           onPressed: () {},
           style:
ElevatedButton.styleFrom(backgroundColor:
bgColor),
           child: Text(buttonText, style:
TextStyle(color: Colors.white)),
          ),
         ],
        ),
       ),

       SizedBox(width: 12),

       // Image Section (Right Side)
       ClipRRect(
        borderRadius: BorderRadius.circular(12),
        child: Image.asset(
         imageUrl,
         width: 80,
         height: 80,
         fit: BoxFit.cover,
        ),
       ),
      ],
     ),
    ),
   ),
  );
 }
}
```

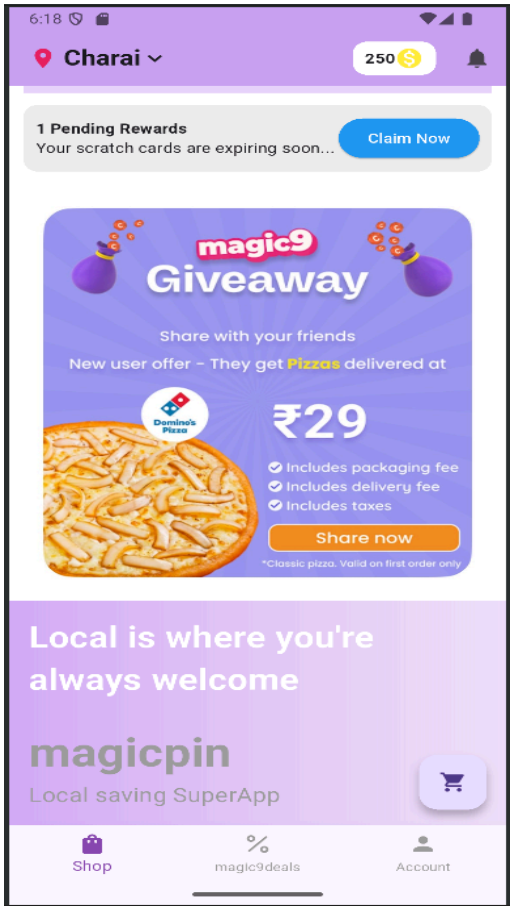
Project Title: MagicPin **Roll No.: 14**

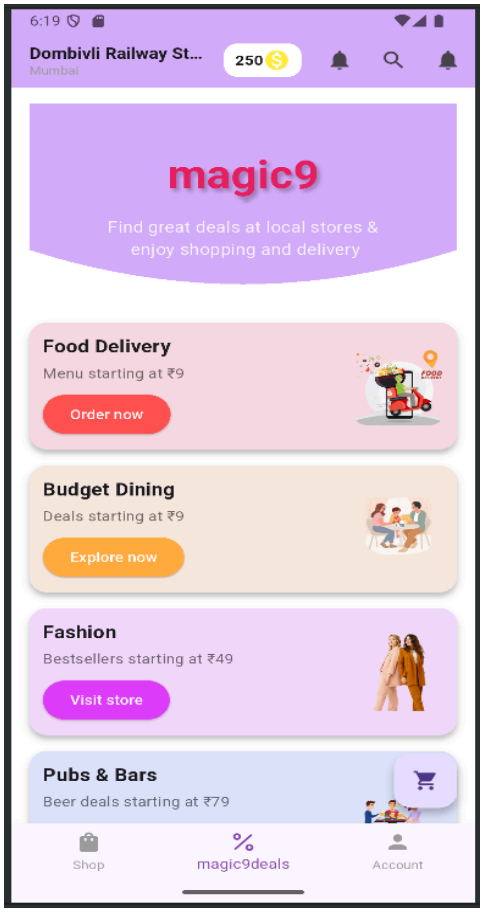
ScreenShots :





Project Title: MagicPin **Roll No.: 14**





MAD & PWA Lab
Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	14
Name	Komal Milind Deolekar
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

AIM : To include icons, images, fonts in flutter app.

Theory :

Flutter provides various ways to include icons, images, and custom fonts to enhance the visual appearance of an application. These assets can be added to the app's pubspec.yaml file and used efficiently in the UI.

Icons, images, and fonts are fundamental elements of UI design that contribute to the visual appeal, usability, and branding of an application. Proper implementation of these elements ensures a seamless user experience and helps create an aesthetically pleasing interface. In Flutter, managing these assets efficiently leads to a professional and engaging application design.

A Flutter app consists of both code and assets (resources). Assets include images, icons, fonts, and configuration files, which can be used dynamically during runtime. Properly managing these assets ensures a seamless and engaging app experience. Flutter supports various image formats, including JPEG, PNG, WebP, GIF, BMP, and animated WebP/GIF. Additionally, it allows developers to use both system and custom fonts to improve the typography of the application.

1. Adding Icons in Flutter

Icons are essential UI components used to represent actions, statuses, or navigation elements.

A. Using Built-in Material Icons

Flutter provides a built-in collection of Material Design icons through the Icons class.

```
Icon(Icons.home, size: 40, color: Colors.blue)
```

Commonly Used Icons:

- Icons.home
- Icons.settings
- Icons.search
- Icons.favorite
- Icons.shopping_cart

B. Using Custom Icons (FontAwesome, Custom TTF)

If you need custom icons, you can use external icon fonts like FontAwesome or your own .ttf icon files.

Steps to Use FontAwesome:

- Add dependency in pubspec.yaml:

```
dependencies:
  flutter:
```

```
    sdk: flutter
  font_awesome_flutter: ^10.6.0
```

- Import and use in Dart file:

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

IconButton(
  icon: FaIcon(FontAwesomeIcons.twitter),
  onPressed: () {},
)
```

C. Using Custom .ttf Icons

- Download a .ttf icon font and place it in the assets/fonts/ folder.
- Register it in pubspec.yaml:

```
flutter:
  fonts:
    - family: CustomIcons
      fonts:
        - asset: assets/fonts/custom_icons.ttf
```

- Use in the app:

```
Text(
  'Custom Icon',
  style: TextStyle(fontFamily: 'CustomIcons', fontSize: 24),
)
```

2. Adding Images in Flutter

Images are a crucial part of UI design, and Flutter supports images from assets and network sources.

A. Using Asset Images (Local Images)

- Place images inside the assets/images/ directory.
- Declare them in pubspec.yaml:

```
flutter:
  assets:
    - assets/images/logo.png
    - assets/images/background.jpg
```

- Use them in the app:

```
Image.asset('assets/images/logo.png', width: 100, height: 100)
```

B. Using Network Images

- To load an image from a URL:

```
Image.network(
  'https://example.com/image.png',
  width: 150,
  height: 150,
  loadingBuilder: (context, child, progress) {
    return progress == null ? child : CircularProgressIndicator();
  },
)
```

C. Using SVG Images

Flutter does not support SVGs natively, but you can use the flutter_svg package.

- Add dependency in pubspec.yaml:

```
dependencies:
  flutter_svg: ^2.0.9
```

- Use in Dart file:

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

SvgPicture.asset('assets/images/vector.svg', width: 100, height: 100)
```

3. Adding Custom Fonts in Flutter

Custom fonts help in creating unique UI designs.

- Download a font file (.ttf or .otf) and place it inside the assets/fonts/ folder.
- Register it in pubspec.yaml:

```
flutter:
  fonts:
    - family: Poppins
      fonts:
        - asset: assets/fonts/Poppins-Regular.ttf
        - asset: assets/fonts/Poppins-Bold.ttf
          weight: 700
```

- Use in the app:

```
Text(
  'Hello, Flutter!',
  style: TextStyle(fontFamily: 'Poppins', fontSize: 20),
)
```

Code :

Account_page.dart

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'sign_in_page.dart';
import 'edit_profile_page.dart';
import 'user_provider.dart';
import 'package:provider/provider.dart';

class AccountPage extends StatefulWidget {
  @override
  _AccountPageState createState() =>
_AccountPageState();
}

class _AccountPageState extends
State<AccountPage> {

  final FirebaseFirestore _firestore =
FirebaseFirestore.instance;
  final FirebaseAuth _auth = FirebaseAuth.instance;

  int _magicPoints = 0;

  @override
  void initState() {
    super.initState();
    // _loadMagicPoints();
    super.initState();
    Provider.of<UserProvider>(context, listen:
false).fetchUserData();
  }

  int _selectedIndex = 2;
  // final FirebaseAuth _auth = FirebaseAuth.instance;

  Future<void> _logout() async {
    await _auth.signOut();
    // Navigator.pushReplacement(context,
MaterialPageRoute(builder: (context) =>
LoginPage()));
    Navigator.pushReplacement(context,
MaterialPageRoute(builder: (context) =>
SignInPage()));
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.white, // Set
background color
      appBar: AppBar(
        title: Text("Account"),
        backgroundColor: Color.fromRGBO(212, 171,
250, 1),
        actions: [
          Consumer<UserProvider>(
            builder: (context, userProvider, child) {
              return Row(
                children: [
                  Container(
                    padding:
EdgeInsets.symmetric(horizontal: 10, vertical: 5),
                    decoration: BoxDecoration(
                      color: Colors.white,
                      borderRadius:
BorderRadius.circular(12),
                    ),
                    child: Row(
                      children: [
                        Text("${userProvider.magicPoints}",
style: TextStyle(fontWeight: FontWeight.bold)),
                        Icon(Icons.monetization_on, color:
Colors.yellow),
                      ],
                    ),
                  ),
                  SizedBox(width: 10),
                  IconButton(icon:
Icon(Icons.notifications), onPressed: () {}),
                ],
              );
            },
          ),
        ],
      ),
      body: SingleChildScrollView(
        child: Column(
          crossAxisAlignment:
CrossAxisAlignment.start,
          children: [
            SizedBox(height: 5),
            Container(
              padding: EdgeInsets.all(16.0),
              decoration: BoxDecoration(
                color: Color.fromRGBO(212, 171, 250, 1),
                borderRadius: BorderRadius.only(
                  bottomLeft: Radius.circular(20),
                  bottomRight: Radius.circular(20),
                ),
              ),
              child: Column(
                children: [
```

```
            Row(
              children: [
                CircleAvatar(
                  radius: 40,
                  backgroundColor: Colors.yellow,
                  child: Icon(Icons.person, size: 50,
color: Colors.white),
                ),
                SizedBox(width: 16),
                Column(
                  crossAxisAlignment:
CrossAxisAlignment.start,
                  children: [
                    Text(
                      _auth.currentUser?.displayName ??
"User",
                      style: TextStyle(
                        fontSize: 22,
                        fontWeight: FontWeight.bold,
                        color: Colors.white,
                      ),
                    ),
                    Text(
                      _auth.currentUser?.email ?? "No
Email",
                      style: TextStyle(
                        fontSize: 16,
                        color: Colors.white70,
                      ),
                    ),
                    TextButton(
                      onPressed: () {
                        User? user =
FirebaseAuth.instance.currentUser;
                        if (user == null) {
                          print("User is not signed in");
                          return;
                        }
                        else{
                          Navigator.push(
                            context,
                            MaterialPageRoute(builder:
(context) => EditProfilePage()),
                          );
                        }
                      },
                      child: Text(
                        "Edit Profile",
                        style: TextStyle(color:
Colors.white),
                      ),
                    )
                  ],
                )
              ],
            ),
            ],
          ),
        ),
        Padding(
          padding: EdgeInsets.all(16.0),
          child: Column(
            children: [
              buildOption(Icons.receipt_long, "My
Transactions"),
              buildOption(Icons.support, "Customer
Support"),
              buildOption(Icons.settings, "Settings"),
              Divider(),
              buildOption(Icons.card_giftcard, "Easy
Rewards", badge: "Save More"),
              buildOption(Icons.share, "Refer & Get
Free Food"),
              buildOption(Icons.star, "Join Pro
Membership", badge: "NEW"),
              buildOption(Icons.credit_card,
"magicCredit"),
              Divider(),
              buildOption(Icons.logout, "Logout",
onTap: _logout),
            ],
          ),
        )
      ],
    ),
  );
}

  Widget buildOption(IconData icon, String title,
{String? badge, VoidCallback? onTap}) {
    return ListTile(
      leading: Icon(icon, color: Colors.purple),
      title: Row(
        mainAxisAlignment:
MainAxisAlignment.spaceBetween,
        children: [
          Text(title),
          if (badge != null)
            Container(
              padding: EdgeInsets.symmetric(horizontal:
8, vertical: 4),
              decoration: BoxDecoration(
                color: Colors.red,
                borderRadius: BorderRadius.circular(8),
              ),
              child: Text(
                badge,
                style: TextStyle(color: Colors.white,
fontSize: 12),
              ),
            ),
```

```
      ],
    ),
    onTap: onTap,
```

Food_store_screen.dart

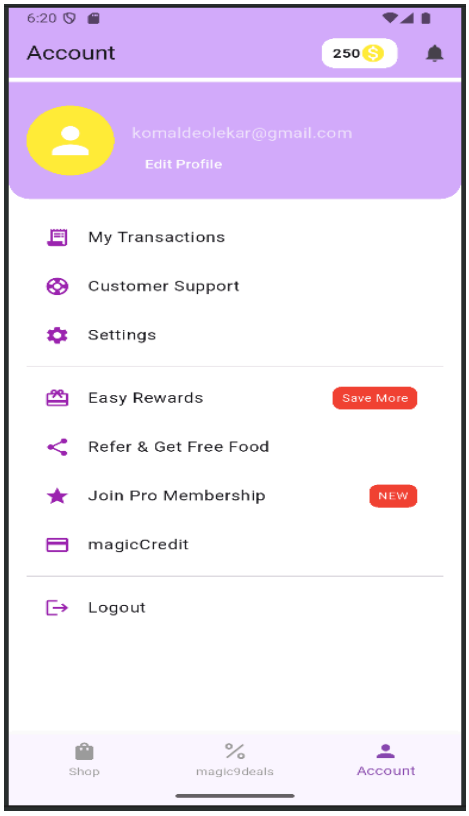
```
import 'package:flutter/material.dart';
import 'package:lucide_icons/lucide_icons.dart';
import 'package:flutter/material.dart';
import 'package:carousel_slider/carousel_slider.dart'
hide CarouselController;
import
'package:carousel_slider/carousel_controller.dart';
import 'footer.dart';
import 'package:flutter/services.dart'; // For loading
JSON
import 'dart:convert';
import 'food_store_details.dart';

class FoodStoreScreen extends StatefulWidget {
 @override
 _FoodStoreScreenState createState() =>
_FoodStoreScreenState();
}

class _FoodStoreScreenState extends
State<FoodStoreScreen> {

 final List<String> imagePaths = [
  'assets/images/food_add2.png',
  'assets/images/food_add2.png',
  'assets/images/food_add3.png',
  'assets/images/food_add4.png',
  'assets/images/food_add5.png',
  'assets/images/food_add6.png',
  'assets/images/food_add7.png',
 ];

 List<Map<String, dynamic>> foodItems = [];

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

 Future<void> loadFoodItems() async {

  try {
   String data = await
rootBundle.loadString('assets/data/food_items/food_i
tems.json');
   print("JSON Loaded: $data"); // Debugging
   setState(() {
    foodItems = List<Map<String,
dynamic>>.from(json.decode(data));
   });
   print("Parsed Data: $foodItems"); // Debugging
  } catch (e) {
   print("Error loading JSON: $e");
  }
 }

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   backgroundColor: Colors.white, // Set
background to white or another color
   appBar: AppBar(
    backgroundColor: Color.fromRGBO(212, 171,
250,1),
    title: Column(
     crossAxisAlignment:
CrossAxisAlignment.start,
     children: [
      Text('Food Delivery', style:
TextStyle(fontSize: 18, fontWeight:
FontWeight.bold)),
      Text('Charai, Mumbai', style:
TextStyle(fontSize: 14, color: Colors.grey)),
     ],
    ),
    ,
     Container(
      padding: EdgeInsets.symmetric(horizontal:
10, vertical: 5),
      decoration: BoxDecoration(
       color: Colors.white,
       borderRadius: BorderRadius.circular(12),
      ),
      child: Row(
       children: [
        Text("300", style: TextStyle(fontWeight:
FontWeight.bold)),
        Icon(Icons.monetization_on, color:
Colors.yellow),
       ],
      ),
     ),
    ],
   ),
   body: SingleChildScrollView(
    child: Padding(
     padding: EdgeInsets.all(10.0),
     child: Column(
      crossAxisAlignment:
```

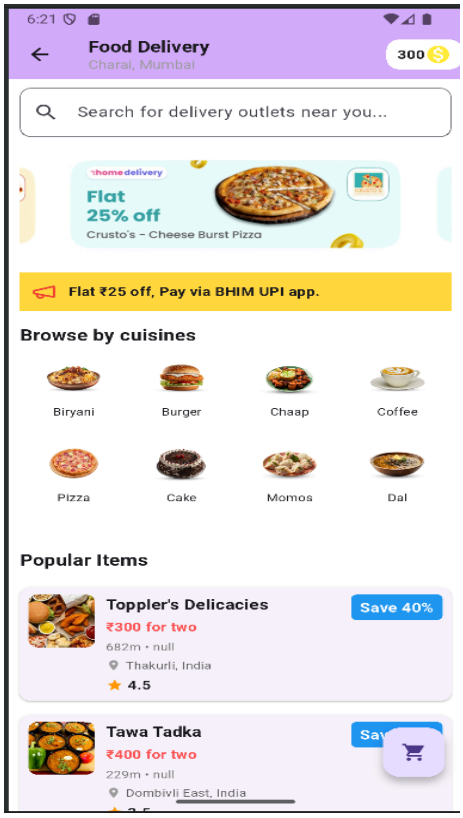
```
CrossAxisAlignment.start,
          children: [
            // Search Bar
            TextField(
              decoration: InputDecoration(
                hintText: 'Search for delivery outlets near
you...',
                prefixIcon: Icon(Icons.search),
                border: OutlineInputBorder(borderRadius:
BorderRadius.circular(10)),
              ),
            ),
                    Padding(
              padding: const
EdgeInsets.symmetric(vertical: 2.0),
              child: CarouselSlider(
                options: CarouselOptions(
                  height: 150.0,
                  autoPlay: true,
                  enlargeCenterPage: true,
                  aspectRatio: 16 / 9,
                  viewportFraction: 0.8,
                ),
                items: imagePaths.map((imagePath) {
                  return ClipRRect(
                    borderRadius:
BorderRadius.circular(12.0),
                    child: Image.asset(
                      imagePath,
                      // fit: BoxFit.cover,
                      fit: BoxFit.contain,
                      width: double.infinity,
                    ),
                  );
                }).toList(),
              ),
            ),
                  // Payment Offer Banner
            Container(
              padding: EdgeInsets.all(10),
              color: Colors.amberAccent,
              child: Row(
                children: [
                  Icon(LucideIcons.megaphone, color:
Colors.red),
                  SizedBox(width: 10),
                  Expanded(child: Text('Flat ₹25 off, Pay
via BHIM UPI app.', style: TextStyle(fontWeight:
FontWeight.bold))),
                ],
              ),
            ),
            SizedBox(height: 15),

            // Browse by Cuisines
            Text('Browse by cuisines', style:
TextStyle(fontSize: 18, fontWeight:
FontWeight.bold)),
            SizedBox(height: 10),
            GridView.count(
              shrinkWrap: true,
              physics: NeverScrollableScrollPhysics(),
              crossAxisCount: 4,
              children: [
                _cuisineItem('Biryani',
'assets/images/biryani.png'),
                _cuisineItem('Burger',
'assets/images/burger.png'),
                _cuisineItem('Chaap',
'assets/images/chaap.png'),
                _cuisineItem('Coffee',
'assets/images/coffee.png'),
                _cuisineItem('Pizza',
'assets/images/pizza.png'),
                _cuisineItem('Cake',
'assets/images/cake.png'),
                _cuisineItem('Momos',
'assets/images/momos.png'),
                _cuisineItem('Dal',
'assets/images/dal.png'),
              ],
            )

            Text('Popular Items', style:
TextStyle(fontSize: 18, fontWeight:
FontWeight.bold)),
            SizedBox(height: 10),
            ListView.builder(
              shrinkWrap: true,
              physics: NeverScrollableScrollPhysics(),
              itemCount: foodItems.length,
              itemBuilder: (context, index) {
                final item = foodItems[index];
                return Card(
                  margin: EdgeInsets.symmetric(vertical:
8),
                  shape:
RoundedRectangleBorder(borderRadius:
BorderRadius.circular(10)),
                  child: InkWell(
                    onTap: () {
                      Navigator.push(
                        context,
                        MaterialPageRoute(
                          builder: (context) =>
FoodShopDetailsPage(shop: item),
                        ),
                      );
                    },
                    child: Stack(
                      children: [
                        Padding(
```

```
                    padding: const EdgeInsets.all(8.0),
// Added padding for better spacing
                    child: Row(
                     crossAxisAlignment:
CrossAxisAlignment.start,
                    children: [
                     // Shop Image
                     ClipRRect(
                      borderRadius:
BorderRadius.circular(8),
                      child: Image.asset(
                       item['image'],
                       width: 60,
                       height: 60,
                       fit: BoxFit.cover,
                      ),
                     ),
                     SizedBox(width: 10), // Space
between image and text

                     // Shop Details
                     Expanded(
                      child: Column(
                       crossAxisAlignment:
CrossAxisAlignment.start,
                       children: [
                        Text(
                         item['name'],
                         style: TextStyle(fontWeight:
FontWeight.bold, fontSize: 16),
                        ),
                        SizedBox(height: 4), // Space
between name and price
                        Text(
                         "₹${item['price_for_two']}
for two",
                         style: TextStyle(color:
Colors.redAccent, fontWeight: FontWeight.bold),
                        ),
                        SizedBox(height: 4), // Space
between price and distance

                        // Distance & Delivery Time
                        Text(
                         "${item['distance']} •
${item['deliveryTime']}",
                         style: TextStyle(fontSize:
12, color: Colors.grey[600]),
                        ),

                        SizedBox(height: 4), // Space
between delivery time and address

                        // Address
                        Row(
                         children: [
                          Icon(Icons.location_on,
size: 14, color: Colors.grey),
                          SizedBox(width: 4),
                          Expanded(
                           child: Text(
                            item['location'], // Ensure
`address` is included in item
                            style:
TextStyle(fontSize: 12, color: Colors.grey[700]),
                            maxLines: 1,
                            overflow:
TextOverflow.ellipsis, // Prevents overflow issues
                           ),
                          ),
                         ],
                        ),

                        SizedBox(height: 4), // Space
before rating row
                        Row(
                         children: [
                          Icon(Icons.star, color:
Colors.orange, size: 16),
                          SizedBox(width: 4),
                          Text(
                           item['rating'].toString(),
                           style: TextStyle(fontSize:
14, fontWeight: FontWeight.bold),
                          ),
                         ],
                        ),
                       ],
                      ),
                     ),
                    ],
                   ),
                  ),

                  // Discount Tag
                  Positioned(
                   top: 8,
                   right: 8,
                   child: Container(
                    padding:
EdgeInsets.symmetric(horizontal: 8, vertical: 4),
                    decoration: BoxDecoration(
                     color: Colors.blue,
                     borderRadius:
BorderRadius.circular(5),
                    ),
                    child: Text(
                     "${item['discount']}",
                     style: TextStyle(color:
Colors.white, fontSize: 15, fontWeight:
FontWeight.bold),
                    ),
```

```
                    ),
                  ),
                ],
              ),
            ),
          );
          ;
        },
      ),
      AppFooter(),

    ],
  ),
 ),
),
floatingActionButton: FloatingActionButton(
  onPressed: () {
    Navigator.pushNamed(context, '/cart');
  },
  child: Icon(Icons.shopping_cart),
 ),
);
}

Widget _cuisineItem(String name, String imgUrl) {
 return Column(
  children: [
   Image.asset(imgUrl, height: 50),
   SizedBox(height: 5),
   Text(name, style: TextStyle(fontSize: 12)),
  ],
 );
}
}
```

ScreenShots :





MAD & PWA Lab
Journal

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget
Roll No.	14
Name	Komal Milind Deolekar
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

AIM : To create an interactive Form using form widget.

Theory :

Forms are an essential component of any application that requires user input. They allow users to enter and submit data, making them useful for login pages, registration forms, feedback sections, and more. In Flutter, the Form widget provides a structured way to handle user input efficiently, ensuring validation and state management.

In Flutter, the Form widget provides a structured way to handle user input efficiently while ensuring validation and state management.

Understanding the Form Widget

The Form widget in Flutter acts as a container for multiple input fields (such as text fields, checkboxes, radio buttons, or dropdowns), unlike using individual TextField widgets. It helps manage the form state and validation collectively, ensuring a smooth user experience.

Key Components of an Interactive Form in Flutter

1. TextFormField Widget: Used for user input fields such as name, email, and password.
2. FormKey: A unique global key that manages the form state and validation.
3. Validators: Functions that check if user input meets the required conditions (e.g., ensuring a field is not empty or an email format is valid).
4. Submit Button: Triggers validation and processes user input.
5. State Management: Helps retain user input and dynamically update the UI.

Importance of Interactive Forms in App Development

- User Engagement: Provides a structured way for users to interact with the app.
- Data Validation: Ensures correct and meaningful input before submission.
- State Management: Maintains form data efficiently for a better user experience. Retains user input even when UI changes.
- Seamless Navigation: Allows users to enter data smoothly with features like auto-focus and keyboard actions.
- Data Collection: Makes it easy to retrieve user data for further processing.

Working Mechanism of a Form in Flutter

1. Defining a Form with a Global Key:
    - The Form widget requires a GlobalKey<FormState> to manage its state and validation.
    - This key helps in performing actions like form validation and resetting inputs.
2. Using TextFormField for Input Fields:
    - Instead of TextField, the TextFormField widget is used within a Form because it supports built-in validation.

   - Each TextFormField can have a validator function to check whether the entered data meets specific conditions.
3. Adding Validation Logic:
   - Each TextFormField includes a validator function to check input validity.
   - When the user submits the form, the validate() function of the FormState is called, which runs all validator functions and ensures that the input is valid before proceeding.
4. Handling User Input and Submission:
   - The onSaved function stores user input for processing.
   - A button (e.g., ElevatedButton) is used to trigger the validation and submission process.
   - If the form is valid, it can proceed with actions like saving data or navigating to another screen.

Example Workflow of an Interactive Form

1. The user enters data into TextFormField fields (e.g., name, email, password).
2. When the submit button is clicked, the FormKey calls validate(), which runs all validator functions.
3. If validation passes, the onSaved function stores the data and the app processes it.
4. If validation fails, appropriate error messages are displayed.

Some Properties of Form Widget

- key: A GlobalKey that uniquely identifies the Form. You can use this key to interact with the form, such as validating, resetting, or saving its state.
- child: The child widget that contains the form fields. Typically, this is a Column, ListView, or another widget that allows you to arrange the form fields vertically.
- autovalidateMode: An enum that specifies when the form should automatically validate its fields.

Some Methods of Form Widget

- validate(): This method is used to trigger the validation of all the form fields within the Form. It returns true if all fields are valid, otherwise false. You can use it to check the overall validity of the form before submitting it.
- save(): This method is used to save the current values of all form fields. It invokes the onSaved callback for each field. Typically, this method is called after validation succeeds.
- reset(): Resets the form to its initial state, clearing any user-entered data.
- currentState: A getter that returns the current FormState associated with the Form.

The Flutter form uses TextField for input, ElevatedButton for submission, and Text for messages. Column and Row structure the layout, while Padding and Container manage spacing and styling. Icons enhance the UI, and Form ensures input validation. GestureDetector handles taps for navigation, while ThemeData and MaterialApp define the app's overall design.

Code :

login_page.dart

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

class PhoneNumberPage extends StatefulWidget {
  @override
  _PhoneNumberPageState createState() =>
_PhoneNumberPageState();
}

class _PhoneNumberPageState extends
State<PhoneNumberPage> {
  TextEditingController phoneController =
TextEditingController();

  // Function to handle navigation to OTP page
  void goToOTPPage() {
    String phoneNumber = phoneController.text.trim();

    // Simple validation for phone number
    if (phoneNumber.isNotEmpty &&
phoneNumber.length == 10) {
      Navigator.pushNamed(
        context,
        '/otp',
        arguments: phoneNumber, // Passing phone
number
      );
    } else {
      // Show error message if phone number is invalid
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text("Please enter a valid
10-digit phone number")),
      );
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Color(0xFF7D73E8), // Light
purple background
      body: Center(
        child: Padding(
          padding: const
EdgeInsets.symmetric(horizontal: 20.0),
          child: Column(
            mainAxisAlignment:
MainAxisAlignment.center,
            children: [
              SizedBox(height: 100),
              Text(
                'magicpin',
                style: TextStyle(
                  fontSize: 32,
                  fontWeight: FontWeight.bold,
                  color: Colors.white,
                ),
              ),
              SizedBox(height: 5),
              Text(
                'Local Savings SuperApp',
                style: TextStyle(
                  fontSize: 16,
                  color: Colors.white,
                ),
              ),
              SizedBox(height: 120),
              Text(
                "Hey, what’s your number?",
                style: TextStyle(
                  fontSize: 20,
                  fontWeight: FontWeight.bold,
                  color: Colors.white,
                ),
              ),
              SizedBox(height: 10),
              Text(
                "Your wallet balance will be linked to this
number. Don't worry, we will never make it public.",
                textAlign: TextAlign.center,
                style: TextStyle(
                  fontSize: 14,
                  color: Colors.white,
                ),
              ),
              SizedBox(height: 30),
              Container(
                decoration: BoxDecoration(
                  color: Colors.white,
                  borderRadius: BorderRadius.circular(10),
                ),
                child: Row(
                  children: [
                    Padding(
                      padding: const EdgeInsets.all(8.0),
                      child:
Image.asset('assets/images/india_flag.jpg', width: 24),
                    ),
                    Text(
                      '+91',
                      style: TextStyle(fontSize: 16,
fontWeight: FontWeight.bold),
                    ),
```

```
                Icon(Icons.arrow_drop_down),
                Expanded(
                  child: TextField(
                    controller: phoneController, // ✅
Added controller
                    keyboardType: TextInputType.phone,
                    style: TextStyle(color: Colors.black),
// ✅ Set text color to bla
                    decoration: InputDecoration(
                      hintText: 'Enter mobile no.',
                      border: InputBorder.none,
                      contentPadding:
EdgeInsets.symmetric(horizontal: 10),
                    ),
                  ),
                ),
              ],
            ),
          ),
          SizedBox(height: 20),
          ElevatedButton(
            onPressed: goToOTPPage, // ✅ Fixed
onPressed
            style: ElevatedButton.styleFrom(
              backgroundColor: Colors.grey[300],
              padding:
EdgeInsets.symmetric(horizontal: 100, vertical: 15),
            ),
            child: Text(
              'Login',
              style: TextStyle(color: Colors.black,
fontSize: 16),
            ),
          ),
          SizedBox(height: 30),
          Text(
            "By clicking 'Login', you accept our",
            style: TextStyle(color: Colors.white,
fontSize: 12),
          ),
          Text(
            "Privacy Policy",
            style: TextStyle(
              color: Colors.white,
              fontSize: 12,
              fontWeight: FontWeight.bold,
            ),
          ),
          Spacer(),
          SizedBox(
            width: MediaQuery.of(context).size.width,
// Takes full screen width
            child: Container(
              padding: EdgeInsets.all(10),
              color: Colors.white,
              child: Column(
                children: [
                  Text(
                    "₹500 worth 🟡 magicPoints",
                    style: TextStyle(fontSize: 14,
fontWeight: FontWeight.bold),
                  ),
                  Text(
                    "New users - Sign up now to claim
your bonus!",
                    style: TextStyle(fontSize: 12),
                  ),
                ],
              ),
            ),
          ),
        ],
      ),
    ),
  );
 }
}
```

otp_verification_page.dart

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

class OTPVerificationPage extends StatefulWidget {
 @override
 _OTPVerificationPageState createState() =>
_OTPVerificationPageState();
}

class _OTPVerificationPageState extends
State<OTPVerificationPage> {
 late String phoneNumber;
 TextEditingController otpController =
TextEditingController();
 int _resendTimer = 30;
 late Timer _timer;

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

 void startResendTimer() {
  _resendTimer = 30;
  _timer = Timer.periodic(Duration(seconds: 1),
```

```
(timer) {
      if (_resendTimer > 0) {
        setState(() {
          _resendTimer--;
        });
      } else {
        timer.cancel();
      }
    });
  }

  @override
  void dispose() {
    _timer.cancel();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    phoneNumber =
ModalRoute.of(context)!.settings.arguments as
String;

    return Scaffold(
      backgroundColor: Color(0xFF7B61FF), //
Magicpin purple background
      body: Column(
        children: [
          SizedBox(height: 100),
          Text(
            "magicpin",
            style: TextStyle(
              fontSize: 28,
              fontWeight: FontWeight.bold,
              color: Colors.white,
              letterSpacing: 1.5,
            ),
          ),
          Text(
            "Local Savings SuperApp",
            style: TextStyle(
              fontSize: 16,
              color: Colors.white70,
            ),
          ),
          SizedBox(height: 70),
          Text(
            "Verification Code",
            style: TextStyle(
              fontSize: 22,
              fontWeight: FontWeight.bold,
              color: Colors.white,
            ),
          ),
          SizedBox(height: 10),
          Text(
            "Waiting for OTP sent to $phoneNumber",
            textAlign: TextAlign.center,
            style: TextStyle(fontSize: 16, color:
Colors.white70),
          ),
          SizedBox(height: 20),
          Padding(
            padding: EdgeInsets.symmetric(horizontal:
40),
            child: PinCodeTextField(
              appContext: context,
              length: 6,
              controller: otpController,
              keyboardType: TextInputType.number,
              animationType: AnimationType.fade,
              enableActiveFill: true,
              pinTheme: PinTheme(
                shape: PinCodeFieldShape.box,
                borderRadius: BorderRadius.circular(12), //
Rounded squares
                fieldHeight: 55,
                fieldWidth: 50,
                inactiveColor: Colors.white,
                activeColor: Colors.white,
                selectedColor: Colors.white,
                inactiveFillColor: Colors.white,
                activeFillColor: Colors.white,
                selectedFillColor: Colors.white,
                borderWidth: 1,
              ),
              onChanged: (value) {},
            ),
          ),
          SizedBox(height: 20),
          ElevatedButton(
            onPressed: () {
              String otp = otpController.text;
              if (otp.length == 6) {
                Navigator.pushNamed(context, '/home'); //
Navigate after OTP verification
              } else {

ScaffoldMessenger.of(context).showSnackBar(
                  SnackBar(content: Text("Please enter a
valid 6-digit OTP")),
                );
              }
            },
                  style: ElevatedButton.styleFrom(
              backgroundColor: Colors.black,
              shape:
RoundedRectangleBorder(borderRadius:
BorderRadius.circular(10)),
              padding: EdgeInsets.symmetric(vertical: 15,
horizontal: 80),
            ),
```

```
          child: Text("Verify", style:
TextStyle(fontSize: 18, color: Colors.white)),
        ),
        SizedBox(height: 15),
        TextButton(
          onPressed: _resendTimer == 0 ?
startResendTimer : null,
          child: Text(
            _resendTimer == 0
                ? "Resend OTP"
                : "Resend in
${_resendTimer.toString().padLeft(2, '0')}",
            style: TextStyle(
              fontSize: 16,
              color: Colors.white70,
              fontWeight: FontWeight.bold,
            ),
          ),
        ),
        Spacer(),
        Container(
          padding: EdgeInsets.symmetric(vertical: 15),
          width: double.infinity,
          decoration: BoxDecoration(
            color: Colors.white,
            borderRadius: BorderRadius.vertical(top:
Radius.circular(20)),
          ),
          child: Column(
            children: [
              Text(
                "₹500 worth 🪙 magicPoints",
                style: TextStyle(
                  fontSize: 16,
                  fontWeight: FontWeight.bold,
                  color: Colors.black,
                ),
              ),
              SizedBox(height: 5),
              Text(
                "New users - Sign up now to claim your
bonus!",
                style: TextStyle(fontSize: 14, color:
Colors.black54),
              ),
            ],
          ),
        )
      ],
    ),
  );
 }
}
```

lib/theme/theme.dart

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

class AppTheme {
  static const Color primaryColor = Color(0xFF796EFF); // Adjust as needed

  static ThemeData lightTheme = ThemeData(
    primaryColor: primaryColor,
    scaffoldBackgroundColor: primaryColor,
    textTheme: TextTheme(
      headlineLarge: TextStyle(fontSize: 24, fontWeight: FontWeight.bold, color: Colors.white),
      bodyLarge: TextStyle(fontSize: 16, color: Colors.white),
    ),
  );
}
```

Main.dart :

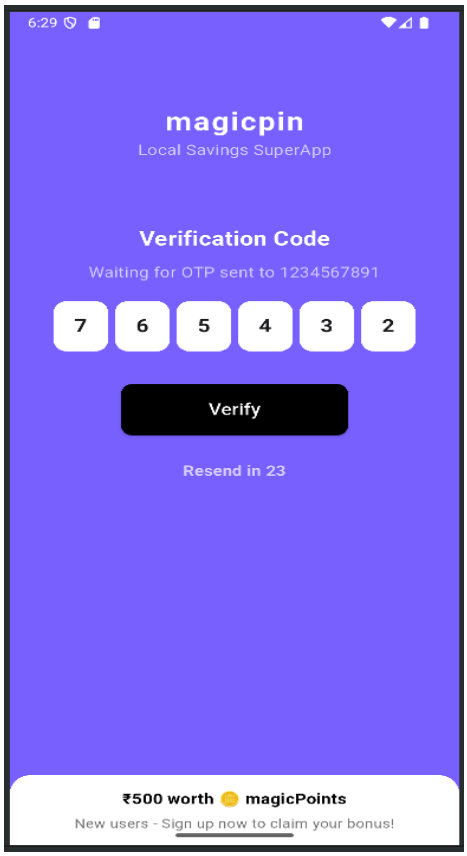
```
import 'package:flutter/material.dart';
import 'screens/login_page.dart';
import 'theme/theme.dart';

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

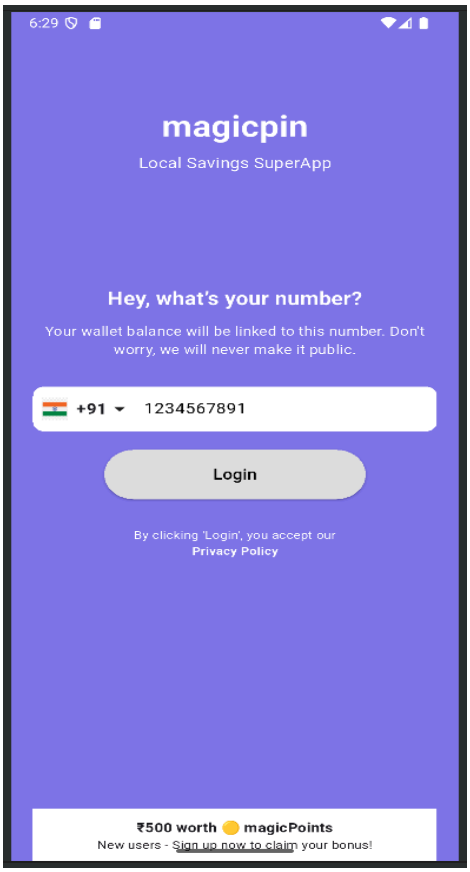
```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Magicpin Clone',
      theme: AppTheme.lightTheme,    //ThemeData(primarySwatch: Colors.blue),
      home: LoginPage(),
    );
  }
}
```

Pubspec.yaml

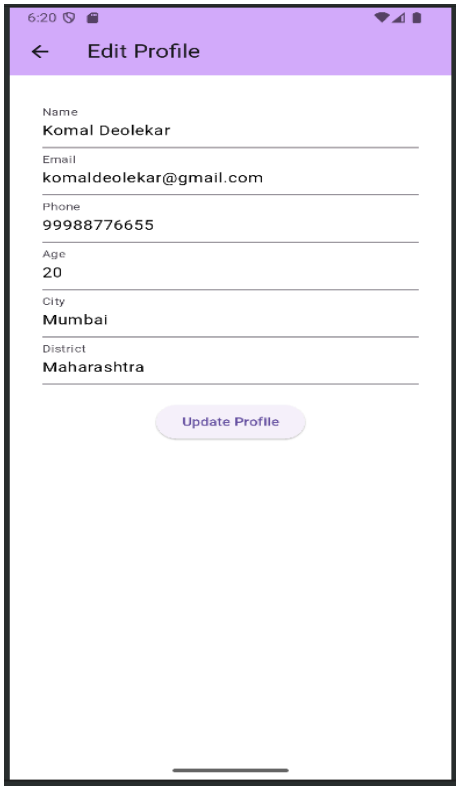
```
assets:
  - assets/images/india_flag.jpg
```

ScreenShots :





Project Title: MagicPin **Roll No.: 14**



MAD & PWA Lab
Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	14
Name	Komal Milind Deolekar
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	

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

Theory :

Navigation, routing, and gestures are essential for creating a seamless and interactive user experience in Flutter applications. These features allow users to move between different screens, perform actions through touch interactions, and improve overall app usability.

1. Navigation and Routing in Flutter

What is Navigation?

Navigation refers to the ability of an app to transition between different screens (pages). In Flutter, navigation is managed using the Navigator class, which maintains a stack of routes. Each screen is considered a route, and users navigate between them using push and pop operations.

Types of Navigation in Flutter

1. **Imperative Navigation (Using Navigator Class)**
    - Uses functions like Navigator.push() and Navigator.pop() to move between screens.
    - Each new screen is added to the stack when pushed and removed when popped.
2. **Declarative Navigation (Using Named Routes)**
    - Screens (routes) are predefined and assigned names in the MaterialApp widget.
    - Users navigate using the assigned names instead of direct widget calls.

Working of Navigation in Flutter

1. **Pushing a New Screen:**
    - Navigator.push(context, MaterialPageRoute(builder: (context) => NewScreen()))
    - Adds a new screen on top of the navigation stack.
2. **Popping a Screen:**
    - Navigator.pop(context)
    - Removes the current screen and returns to the previous one.
3. **Using Named Routes:**

- Define routes in MaterialApp:

```
routes: {
  '/home': (context) => HomeScreen(),
  '/profile': (context) => ProfileScreen(),
}
```

- Navigate using:
-

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

Importance of Navigation

- Allows users to transition between different screens.
- Provides a structured way to manage app flow.
- Enhances user experience by ensuring smooth screen transitions.

2. Gestures in Flutter

Gestures allow users to interact with an app through touch-based actions like tapping, swiping, pinching, and dragging. Flutter provides the GestureDetector widget to detect and respond to various gestures.

Common Gestures in Flutter

- **Tap Gesture:** Detects single or double taps on a widget.

```
GestureDetector(
  onTap: () {
    print("Tapped!");
  },
  child: Container(
    color: Colors.blue,
    height: 100,
    width: 100,
  ),
)
```

- **Long Press Gesture:** Detects when a user presses and holds a widget.

```
GestureDetector(
  onLongPress: () {
    print("Long Pressed!");
  },
)
```

- **Swipe (Drag) Gesture:** Detects horizontal or vertical swipes.

-
```
GestureDetector(
  onHorizontalDragUpdate: (details) {
    print("Swiped horizontally!");
  },
)
```

- **Double Tap Gesture:** Detects a quick double tap.

```
GestureDetector(
  onDoubleTap: () {
    print("Double Tapped!");
  },
)
```

Importance of Gestures in Flutter

- Enables user interaction with UI elements.
- Enhances accessibility and user experience.
- Makes apps more dynamic and responsive.

3. Combining Navigation, Routing, and Gestures

Navigation and gestures often work together to improve app usability. For example:

- A swipe gesture can navigate between screens.
- A long press gesture can open a context menu for navigation options.
- Tapping a button can trigger navigation using Navigator.push().

Example Use Case

1. The user taps a button → Navigates to a new screen.
2. The user swipes left → Moves to the previous screen.
3. The user long-presses an item → Opens a menu with navigation options.

Code :

main_screen.dart

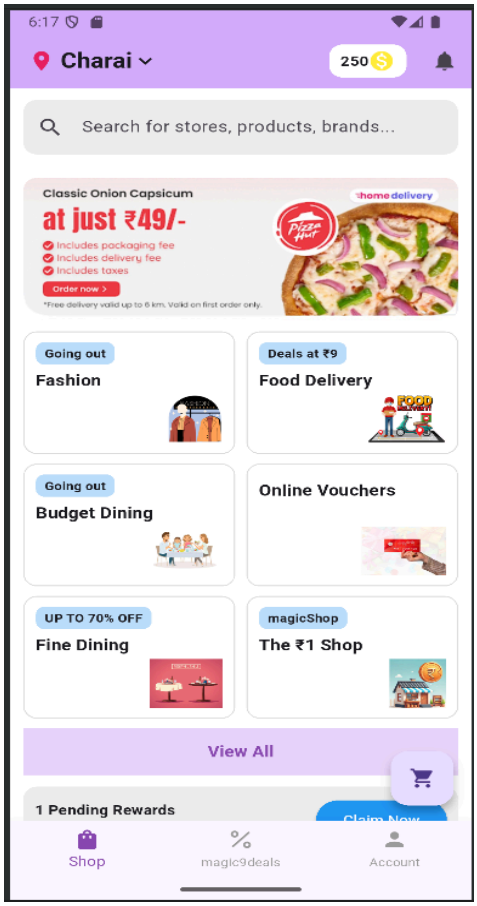
```
import 'package:flutter/material.dart';
import 'package:lucide_icons/lucide_icons.dart';
import 'home_page.dart';
import 'magic9_deals.dart';
import 'account_page.dart';

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

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

 // List of pages
 final List<Widget> _pages = [
  HomePage(),
  Magic9Deals(),
  AccountPage(),
 ];

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   body: IndexedStack(
    index: _currentIndex,
    children: _pages,
   ),
   bottomNavigationBar: BottomNavigationBar(
    currentIndex: _currentIndex,
    // selectedItemColor: Colors.blue,
    selectedItemColor: Color.fromRGBO(137, 74,
176,1),
    unselectedItemColor: Colors.grey,
    onTap: (index) {
     setState(() {
      _currentIndex = index;
     });
    },
    items: [
     BottomNavigationBarItem(icon:
Icon(Icons.shopping_bag), label: "Shop"),
     BottomNavigationBarItem(icon:
Icon(LucideIcons.percent), label: "magic9deals"),
     BottomNavigationBarItem(icon:
Icon(Icons.person), label: "Account"),
    ],
   ),
  );
 }
}
```

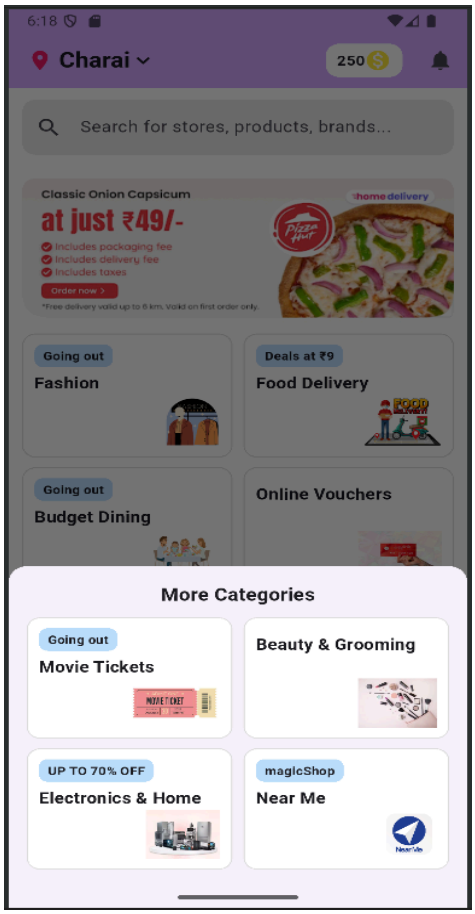
main.dart

```
import 'package:flutter/material.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:provider/provider.dart';
import 'screens/phone_number_page.dart';
import 'screens/otp_verification_page.dart';
import 'screens/main_screen.dart';
import 'theme/theme.dart';
import 'screens/cart_provider.dart';
import 'screens/cart_page.dart';
import 'screens/sign_in_page.dart';
import 'screens/sign_up_page.dart';
import 'screens/user_provider.dart';

void main() async {
 WidgetsFlutterBinding.ensureInitialized();
 await Firebase.initializeApp();

 runApp(
  MultiProvider(
   providers: [
    ChangeNotifierProvider(create: (context) =>
CartProvider()),
    ChangeNotifierProvider(create: (context) =>
UserProvider()),
   ],
   child: MyApp(),
```

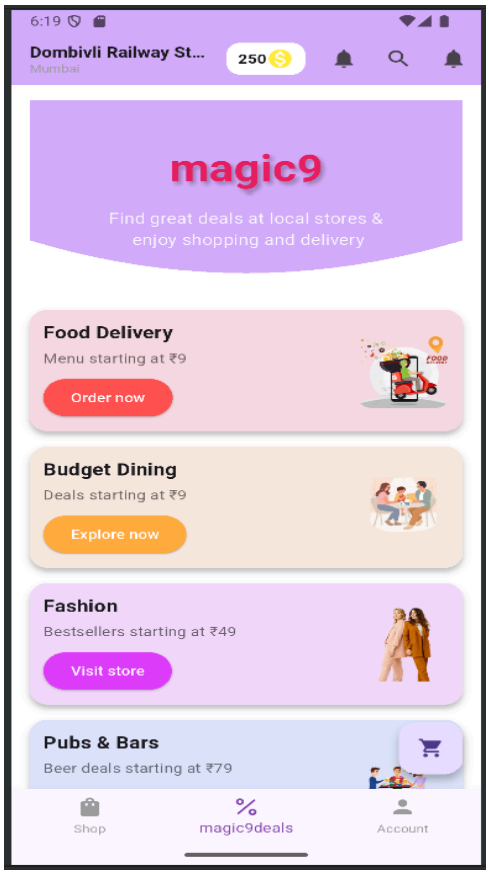
```
    ),
  );
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Magicpin Clone',
      theme: AppTheme.lightTheme,
      home: AuthWrapper(),  // Determines which page
to show
      routes: {
        '/signup' : (context) => SignUpPage(),
         '/login' : (context) => SignInPage(),
        '/phone' : (context) => PhoneNumberPage(),
        '/otp': (context) => OTPVerificationPage(),
        '/home': (context) => MainScreen(),
        '/cart': (context) => CartPage(),
        // '/login': (context) => LoginPage(),
      },
    );
  }
}

// ✅ AuthWrapper: Redirects user based on
authentication state
class AuthWrapper extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return StreamBuilder<User?>(
      stream:
FirebaseAuth.instance.authStateChanges(),
      builder: (context, snapshot) {
        if (snapshot.connectionState ==
ConnectionState.waiting) {
          return Scaffold(body: Center(child:
CircularProgressIndicator())); // Loading state
        } else if (snapshot.hasData) {
          return MainScreen(); // User is signed in
        } else {
          // return LoginPage(); // User is not signed in
          return SignInPage();
        }
      },
    );
  }
}
```

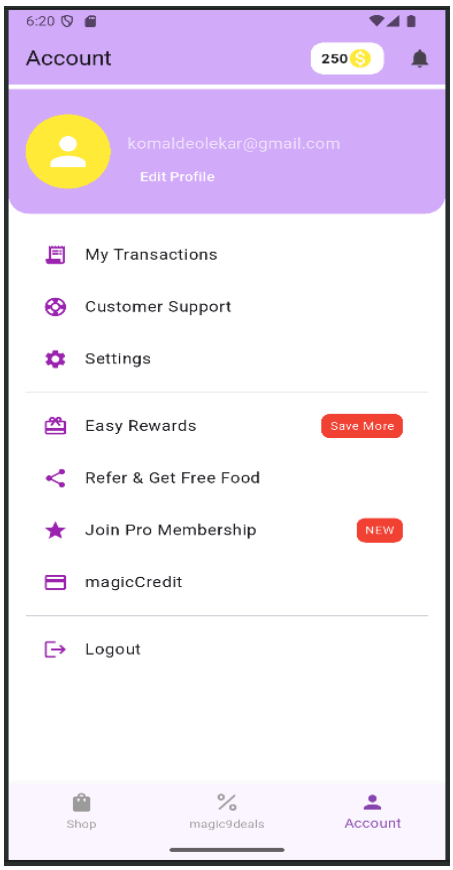
Project Title: MagicPin **Roll No.: 14**





Project Title: MagicPin **Roll No.: 14**





MAD & PWA Lab
Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	14
Name	Komal Milind Deolekar
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:	

AIM : To Connect Flutter UI with fireBase database

Theory :

Firebase provides a powerful backend solution for Flutter applications, enabling features like authentication, real-time database, and cloud storage. By integrating Firebase, developers can easily manage user authentication and store data without setting up a separate backend. Firebase is a cloud-based platform by Google that provides backend services for mobile and web applications, eliminating the need to manage servers. It integrates seamlessly with Flutter to enhance app functionality, security, and performance.

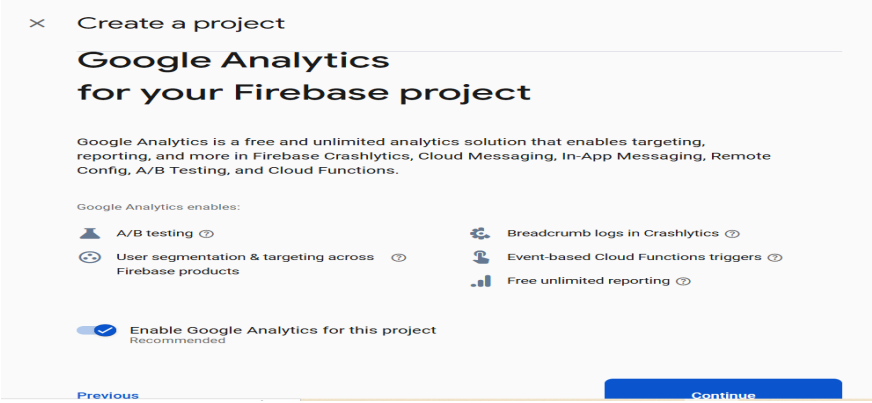
It includes **Authentication** (email, social logins, phone sign-in), **Cloud Firestore & Realtime Database** for real-time data syncing, and **Cloud Storage** for handling images, videos, and documents. **Cloud Messaging (FCM)** enables push notifications, while **Crashlytics** and **Performance Monitoring** help in debugging and optimizing app performance. **Remote Config** allows updating app features without requiring updates, and **Analytics** provides insights into user behavior. With Firebase, Flutter apps can be more scalable, secure, and feature-rich without complex backend management.

Steps to Set Up Firebase with Android Apps

Step 1: Create a Firebase Project

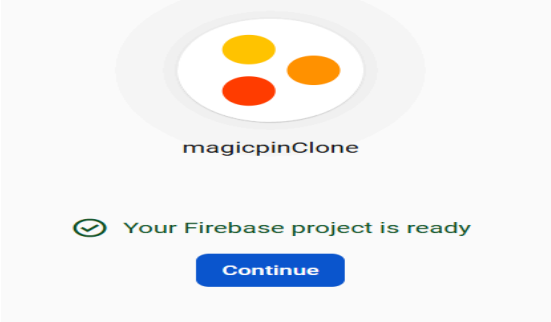
1. Go to Firebase Console.
2. Click on **"Add Project"**, enter your project name, and proceed.

3. Enable **Google Analytics** (optional) and complete the setup.



If you choose to use Google Analytics, you will need to review and accept the terms and conditions prior to project creation.

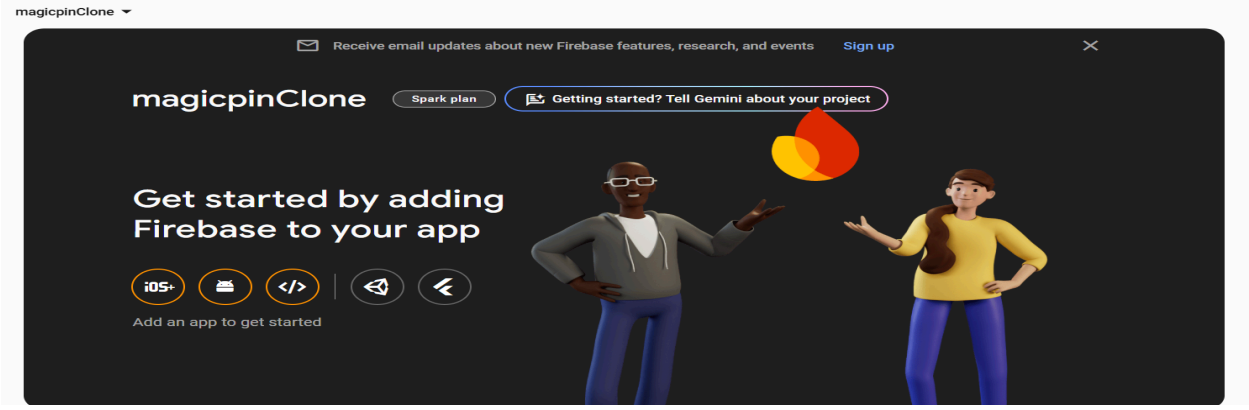
After pressing Continue, your project will be created and resources will be provisioned. You will then be directed to the dashboard for the new project.



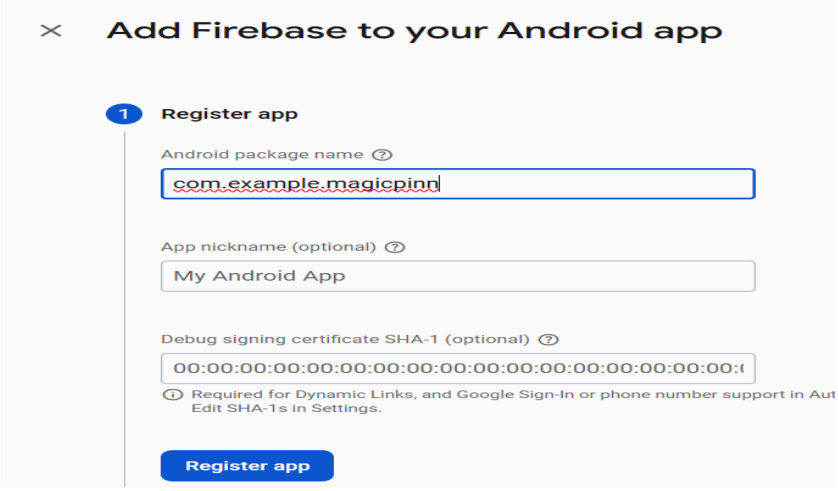
Step 2: Add Firebase to Your Flutter App

Adding Android support :

1. Click **"Add App"** → Select **Android**.



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



2. Enter the package name (found in android/app/build.gradle).

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

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

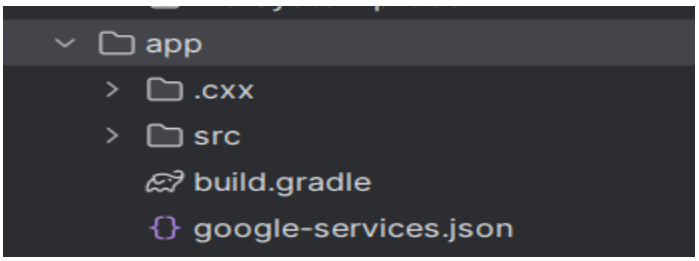
   `com.example.flutterfirebaseexample`

   You can copy it from `applicationId` in `android/app/build.gradle` in your code editor android/app/build.gradle

   You can skip the app nickname and debug signing keys at this stage. Select Register app to continue.

3. Download the **google-services.json** file and place it inside android/app/.

   This is important as it contains the API keys and other critical information for Firebase to use.



Adding the Firebase SDK

We'll now need to update our Gradle configuration to include the Google Services plugin.

Open android/build.gradle in your code editor and modify it to include the following:

```
dependencies {
    classpath 'com.google.gms:google-services:4.3.10'
}
```

android/buiild.gradle

```
buildscript {
    ext.kotlin_version = "1.9.22"
    repositories {
        google()
        mavenCentral()
    }
    dependencies {
        classpath 'com.google.gms:google-services:4.4.0'
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:1.9.22"
    }
```

Finally, update the app level file at android/app/build.gradle to include the following:

```
apply plugin: 'com.google.gms.google-services'
```

android/app/build.gradle

```
apply plugin: 'com.android.application'
// Add this line
apply plugin: 'com.google.gms.google-services'

dependencies {
  // Import the Firebase BoM
  implementation platform('com.google.firebase:firebase-bom:28.0.0')

}

plugins {
    id "com.android.application"
    id "kotlin-android"
    id "dev.flutter.flutter-gradle-plugin"
    id 'com.google.gms.google-services'
}

dependencies {
```

```
  // Import the Firebase BoM
   implementation platform('com.google.firebase:firebase-bom:33.9.0')
  // TODO: Add the dependencies for Firebase products you want to use
   // When using the BoM, don't specify versions in Firebase dependencies

  // Add the dependencies for any other desired Firebase products
// https://firebase.google.com/docs/android/setup#available-libraries
}
```

Or else

```
apply plugin: 'com.google.gms.google-services'
```
at the end

With this update, we're essentially applying the Google Services plugin as well as looking at how other Flutter Firebase plugins can be activated such as Analytics.

Step 3: Install Firebase Dependencies

Open pubspec.yaml and add:

```
dependencies:

  firebase_core: latest_version

  firebase_auth: latest_version

  cloud_firestore: latest_version
```

Then run :

```
flutter pub get
```

Step 4: Initialize Firebase in Flutter

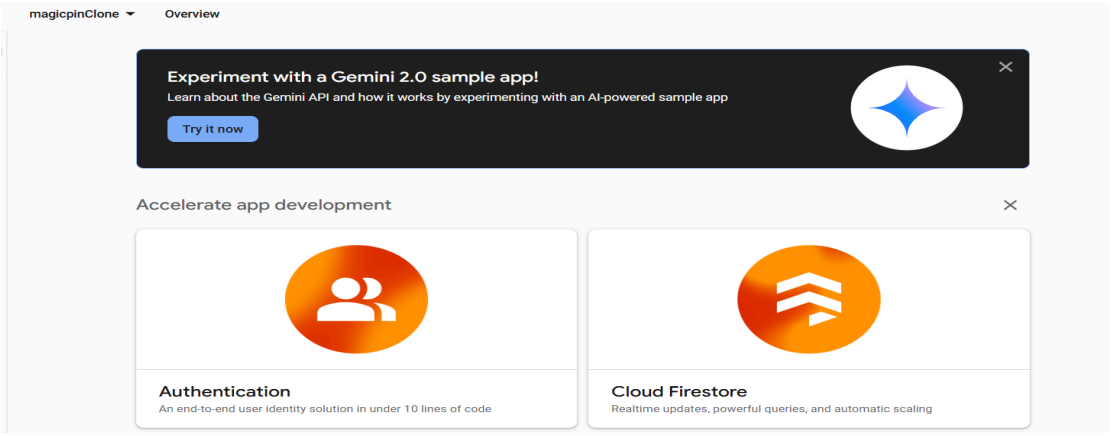
Open main.dart and update the main() function:

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

import 'package:firebase_core/firebase_core.dart';

void main() async {

  WidgetsFlutterBinding.ensureInitialized();

  await Firebase.initializeApp();

  runApp(MyApp());

}
```

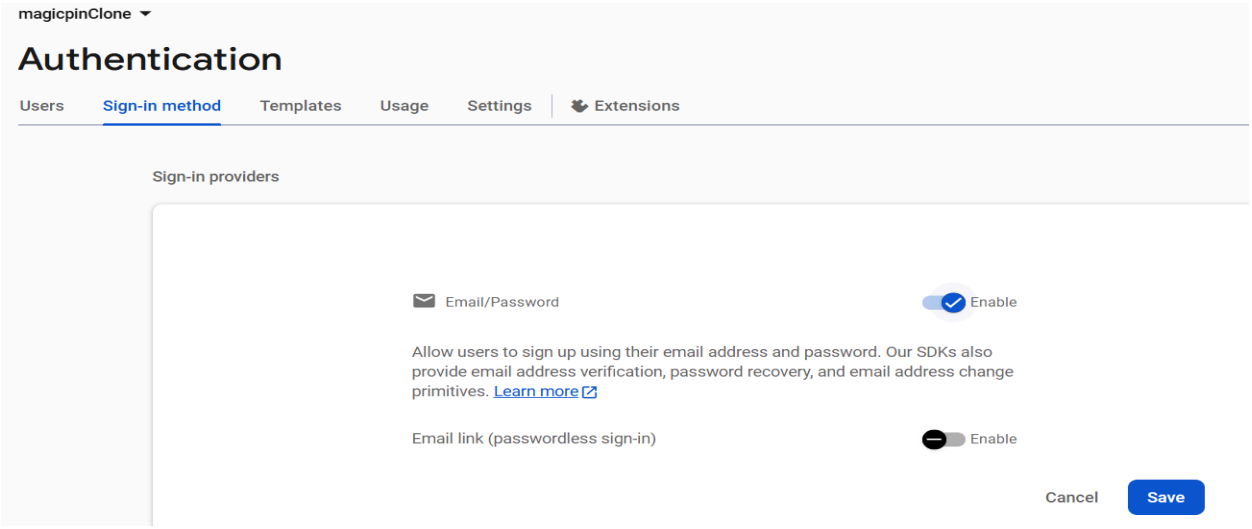
Ensure that MyApp() is wrapped inside MaterialApp().

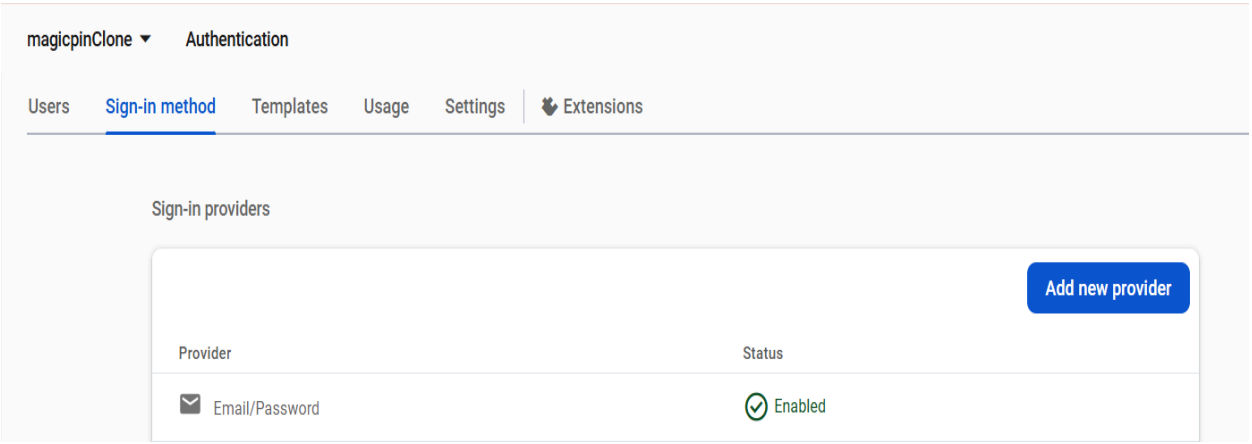
Step 5: Enable Authentication in Firebase

1. In Firebase Console, go to **Authentication** > **Sign-in method**.



2. Enable **Email/Password Authentication** (or any other method).





Step 6: Implement Firebase Authentication in Flutter

1. Register a New User (Signup)

```
import 'package:firebase_auth/firebase_auth.dart';
Future<void> signUpUser(String email, String password) async {
  try {
    UserCredential userCredential = await FirebaseAuth.instance.createUserWithEmailAndPassword(
      email: email,
      password: password,
    );
    print("User registered: ${userCredential.user?.uid}");
  } catch (e) {
    print("Signup failed: $e");
  }
}
```

2. Login User

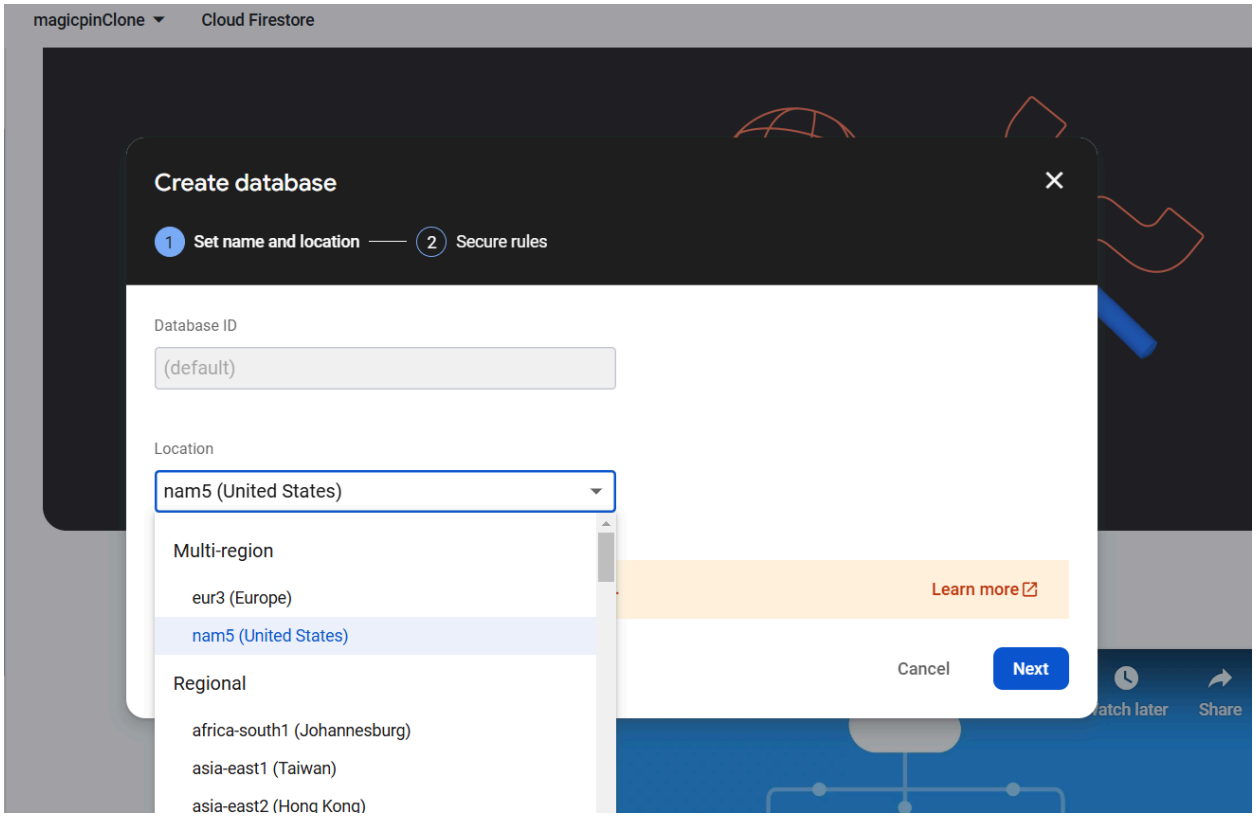
```
Future<void> loginUser(String email, String password) async {
  try {
    UserCredential userCredential = await FirebaseAuth.instance.signInWithEmailAndPassword(
      email: email,
      password: password,
    );
    print("User logged in: ${userCredential.user?.uid}");
  } catch (e) {
    print("Login failed: $e");
  }
}
```

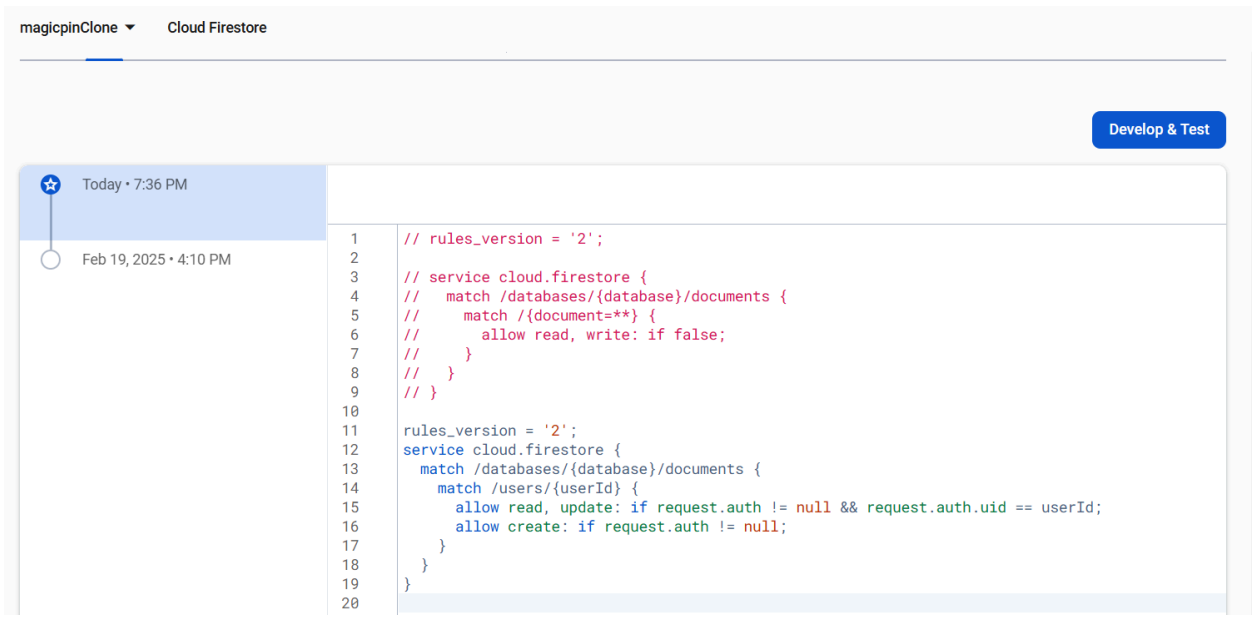
3. Logout User

```
Future<void> logoutUser() async {
  await FirebaseAuth.instance.signOut();
  print("User logged out");
}
```

Step 7: Enable Firestore Database in Firebase

1. In Firebase Console, go to **Firestore Database**.
2. Click **Create Database**, select **Start in Test Mode**, and enable Firestore.





Step 8: Perform Database Operations in Flutter

1. Add Data to Firestore

```
import 'package:cloud_firestore/cloud_firestore.dart';
void addUser() {
  FirebaseFirestore.instance.collection('users').add({
    'name': 'John Doe',
    'email': 'johndoe@example.com',
  });
}
```

2. Read Data from Firestore

```
void fetchUsers() {
  FirebaseFirestore.instance.collection('users').get().then((snapshot) {
    for (var doc in snapshot.docs) {
      print(doc.data());
    }
  });
}
```

3. Update Data in Firestore

```
void updateUser(String docID) {
  FirebaseFirestore.instance.collection('users').doc(docID).update({
    'name': 'Jane Doe',
  });
}
```

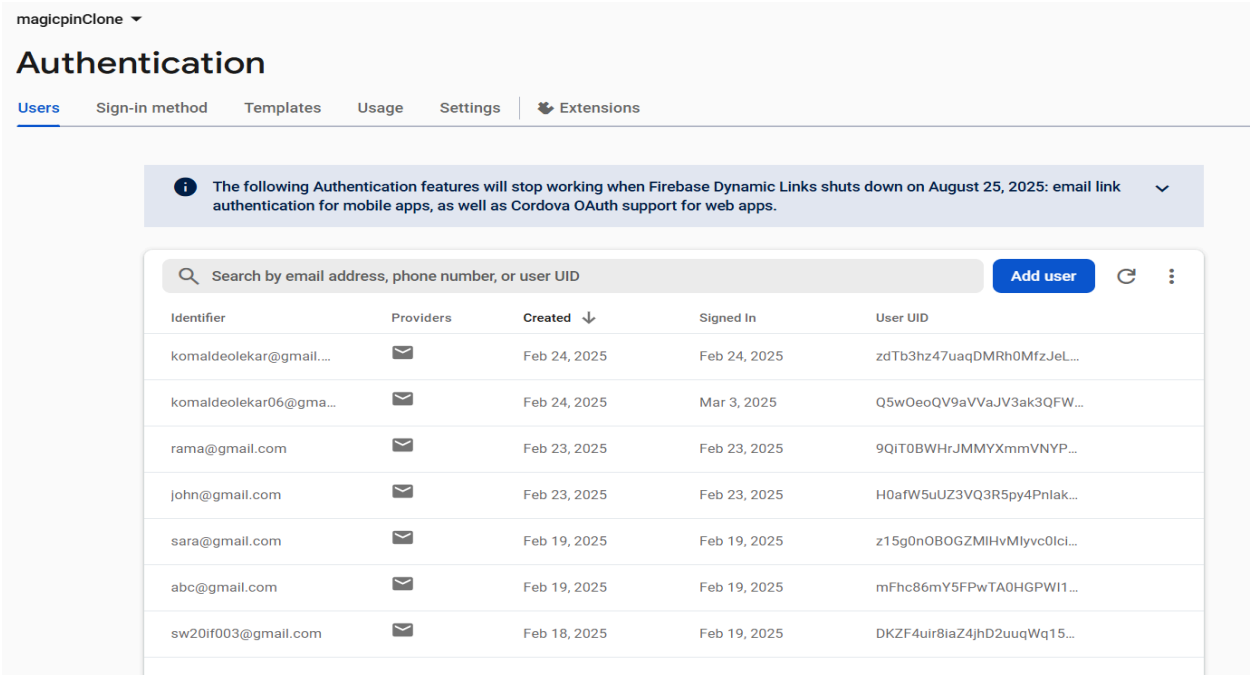
4. Delete Data from Firestore

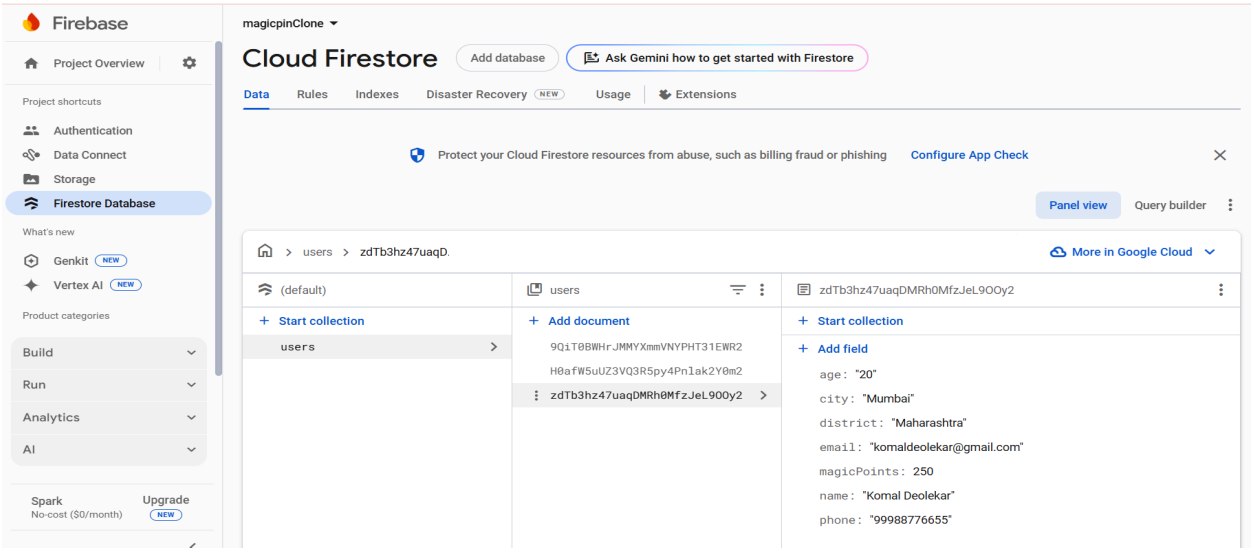
```
void deleteUser(String docID) {
  FirebaseFirestore.instance.collection('users').doc(docID).delete();
}
```

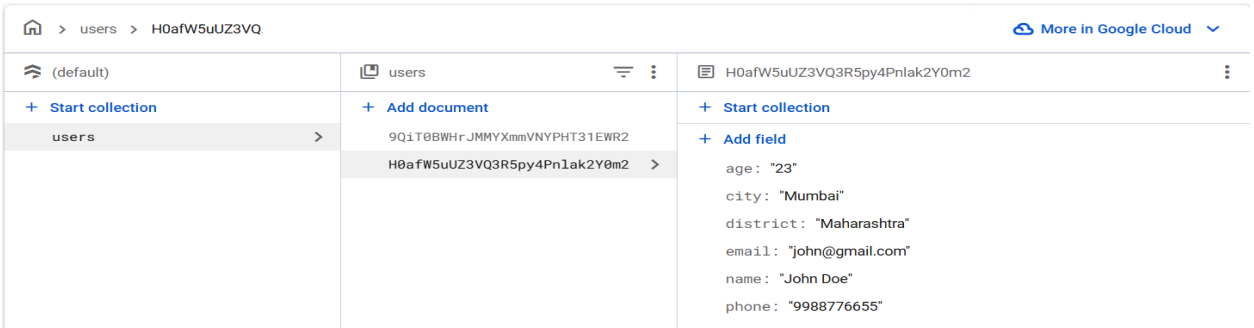
Step 9: Test Firebase Connection

Now update the codes and run the app using:
flutter run

1. Perform actions like adding, reading, updating, and deleting data.
2. Check Firebase Console to verify the changes.







Codes :

sign _up_page.dart

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

class SignUpPage extends StatefulWidget {
  @override
  _SignUpPageState createState() =>
_SignUpPageState();
}

class _SignUpPageState extends State<SignUpPage>
{
  final TextEditingController nameController =
TextEditingController();
  final TextEditingController emailController =
TextEditingController();
  final TextEditingController passwordController =
TextEditingController();
  final FirebaseAuth _auth = FirebaseAuth.instance;
  final FirebaseFirestore _firestore =
FirebaseFirestore.instance;
  bool isLoading = false;


  void signUp() async {
    if (nameController.text.trim().isEmpty ||
        emailController.text.trim().isEmpty ||
        passwordController.text.trim().isEmpty) {
      Fluttertoast.showToast(msg: "All fields are
required!");
      return; // Stop execution if any field is empty
    }

    setState(() => isLoading = true);

    try {
      UserCredential userCredential = await
_auth.createUserWithEmailAndPassword(
        email: emailController.text.trim(),
        password: passwordController.text.trim(),
      );

      // Save user data in Firestore
      await
_firestore.collection("users").doc(userCredential.user
!.uid).set({
        "name": nameController.text.trim(),
        "email": emailController.text.trim(),
        "magicPoints": 250,
      });

      Fluttertoast.showToast(msg: "Sign Up
Successful!");

      // Navigate to MainScreen only if sign-up is
successful
      Navigator.pushReplacement(
        context,
        // MaterialPageRoute(builder: (context) =>
MainScreen()),
          MaterialPageRoute(builder: (context) =>
SignInPage()),

      );
    } catch (e) {
      Fluttertoast.showToast(msg: "Sign Up Failed:
${e.toString()}");
    } finally {
      setState(() => isLoading = false);
    }
  }


  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Color(0xFF7D73E8),
      appBar: AppBar(
        centerTitle: true,
        backgroundColor: Color(0xFF7D73E8),
        elevation: 0, // Remove shadow
      ),
      body: Center(
        child: Padding(
          padding: const EdgeInsets.all(20.0),
```

```
        child: Column(
          children: [
            Text(
              "Magicpin",
              style: TextStyle(fontSize: 35, fontWeight:
FontWeight.bold, color: Colors.white),
            ),
            Text(
              'Local Savings SuperApp',
              style: TextStyle(
                fontSize: 16,
                color: Colors.white,
              ),
            ),
            SizedBox(height: 120),
            Card(
              elevation: 5,
              shape:
RoundedRectangleBorder(borderRadius:
BorderRadius.circular(15)),
              child: Padding(
                padding: const EdgeInsets.all(20.0),
                child: Column(
                  mainAxisSize: MainAxisSize.min,
                  children: [
                    Text("Sign Up", style:
TextStyle(fontSize: 24, fontWeight:
FontWeight.bold,color: Color(0xFF796EFF) )),
                    SizedBox(height: 20),
                    TextField(
                      controller: nameController,
                      style: TextStyle(color: Colors.black),
                      decoration: InputDecoration(
                        labelText: "Full Name",
                        border: OutlineInputBorder(),
                        labelStyle: TextStyle(color:
Colors.black),
                      ),
                    ),
                    SizedBox(height: 15),
                    TextField(
                      controller: emailController,
                      style: TextStyle(color: Colors.black),
                      decoration: InputDecoration(
                        labelText: "Email",
                        border: OutlineInputBorder(),
                        labelStyle: TextStyle(color:
Colors.black),
                      ),
                    ),
                    SizedBox(height: 15),
                    TextField(
                      controller: passwordController,
                      style: TextStyle(color: Colors.black),
                      decoration: InputDecoration(
                        labelText: "Password",
                        border: OutlineInputBorder(),
                        labelStyle: TextStyle(color:
Colors.black),
                      ),
                      obscureText: true,
                    ),
                    SizedBox(height: 20),
                    isLoading
                        ? CircularProgressIndicator()
                        : ElevatedButton(
                      onPressed: signUp,
                      style:
ElevatedButton.styleFrom(minimumSize:
Size(double.infinity, 50),backgroundColor:
Color.fromRGBO(212, 171, 250, 1)),
                      child: Text("Sign Up"),
                    ),
                    SizedBox(height: 10),
                    TextButton(
                      onPressed: () =>
Navigator.pushNamed(context, '/login'),
                      child: Text("Already have an account?
Sign In"),
                    ),
                  ],
                ),
              ),
            ),
          ],
        ),
      ),
    );
  }
}
```

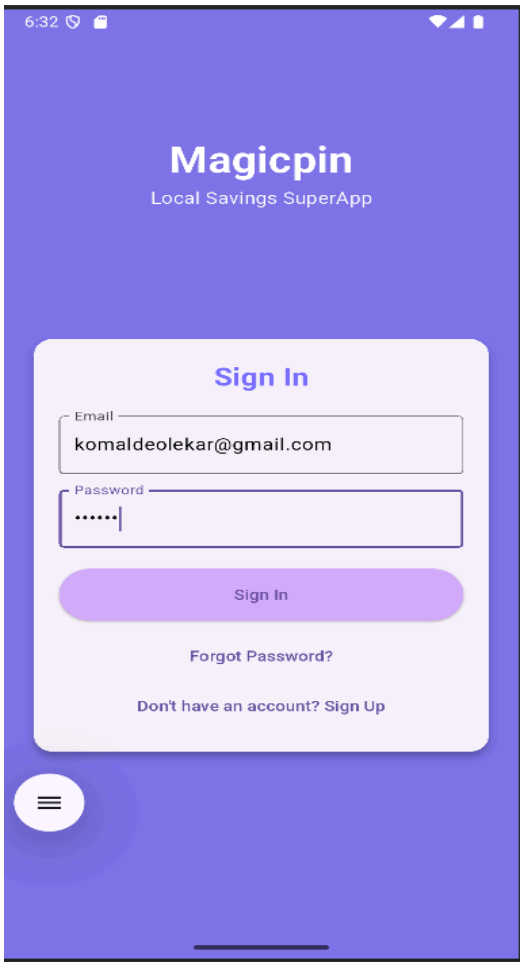
Sign_in_page.dart

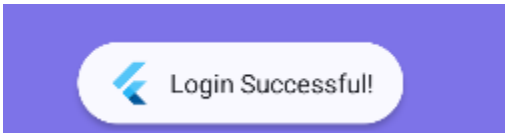
```
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:fluttertoast/fluttertoast.dart';
import 'main_screen.dart'; // Redirect to home after
login
import 'phone_number_page.dart';

class SignInPage extends StatefulWidget {
 @override
 _SignInPageState createState() =>
_SignInPageState();
}

class _SignInPageState extends State<SignInPage> {
 final TextEditingController emailController =
TextEditingController();
 final TextEditingController passwordController =
TextEditingController();
 final FirebaseAuth _auth = FirebaseAuth.instance;
 bool isLoading = false;

 void signIn() async {
  setState(() => isLoading = true);

  try {
   await _auth.signInWithEmailAndPassword(
    email: emailController.text.trim(),
    password: passwordController.text.trim(),
   );

   Fluttertoast.showToast(msg: "Login
Successful!");
   // Navigator.pushReplacement(context,
MaterialPageRoute(builder: (context) =>
MainScreen()));
   Navigator.pushReplacement(context,
MaterialPageRoute(builder: (context) =>
PhoneNumberPage()));
  } catch (e) {
   Fluttertoast.showToast(msg: "Login Failed:
${e.toString()}");
  } finally {
   setState(() => isLoading = false);
  }
 }

 void resetPassword() async {
  if (emailController.text.isNotEmpty) {
   try {
    await _auth.sendPasswordResetEmail(email:
emailController.text.trim());
    Fluttertoast.showToast(msg: "Password reset
email sent!");
   } catch (e) {
    Fluttertoast.showToast(msg: "Error:
${e.toString()}");
   }
  } else {
   Fluttertoast.showToast(msg: "Enter your email to
reset password");
  }
 }

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   backgroundColor: Color(0xFF7D73E8),
   body: Center(
    child: Padding(
     padding: const EdgeInsets.all(20.0),
     child: Column(
      // mainAxisSize: MainAxisSize.min,
      children: [
       SizedBox(height: 100), // Adds space at the
top
       Text(
        "Magicpin",
        style: TextStyle(fontSize: 35, fontWeight:
FontWeight.bold, color: Colors.white),
       ),
       Text(
        'Local Savings SuperApp',
        style: TextStyle(
         fontSize: 16,
         color: Colors.white,
        ),
       ),
       SizedBox(height: 120),
       Card(
        elevation: 5,
        shape:
RoundedRectangleBorder(borderRadius:
BorderRadius.circular(15)),
        child: Padding(
```

```
            padding: const EdgeInsets.all(20.0),
            child: Column(
              mainAxisSize: MainAxisSize.min,
              children: [
                Text("Sign In", style:
TextStyle(fontSize: 24, fontWeight: FontWeight.bold
, color: Color(0xFF796EFF))),
                SizedBox(height: 20),
                TextField(
                  controller: emailController,
                  style: TextStyle(color: Colors.black),
// Input text in black
                  decoration: InputDecoration(
                    labelText: "Email",
                    border: OutlineInputBorder(),
                  ),
                ),
                SizedBox(height: 15),
                TextField(
                  controller: passwordController,
                  style: TextStyle(color: Colors.black),
// Input text in black
                  decoration: InputDecoration(
                    labelText: "Password",
                    border: OutlineInputBorder(),
                  ),
                  obscureText: true,
                ),
                SizedBox(height: 20),
                isLoading
                    ? CircularProgressIndicator()
                    : ElevatedButton(
                  onPressed: signIn,
                  style:
ElevatedButton.styleFrom(minimumSize:
Size(double.infinity, 50),
                      backgroundColor:
Color.fromRGBO(212, 171, 250, 1), ),
                  child: Text("Sign In"),

                ),
                SizedBox(height: 10),
                TextButton(onPressed: resetPassword,
child: Text("Forgot Password?")),
                TextButton(
                  onPressed: () =>
Navigator.pushNamed(context, '/signup'),
                  child: Text("Don't have an account?
Sign Up"),
                ),
              ],
            ),
          ),
        ),
      ],
    ),
  ),
 );
 }
}
```

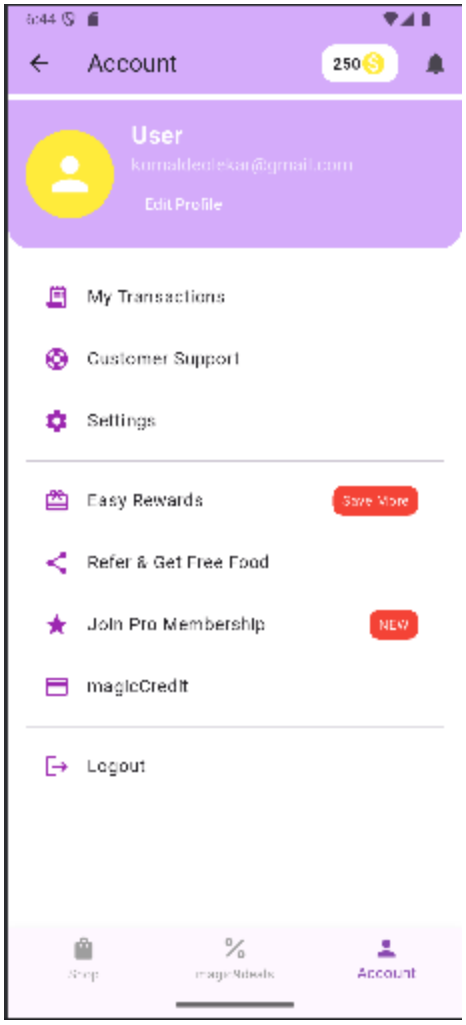
Project Title: MagicPin **Roll No.: 14**

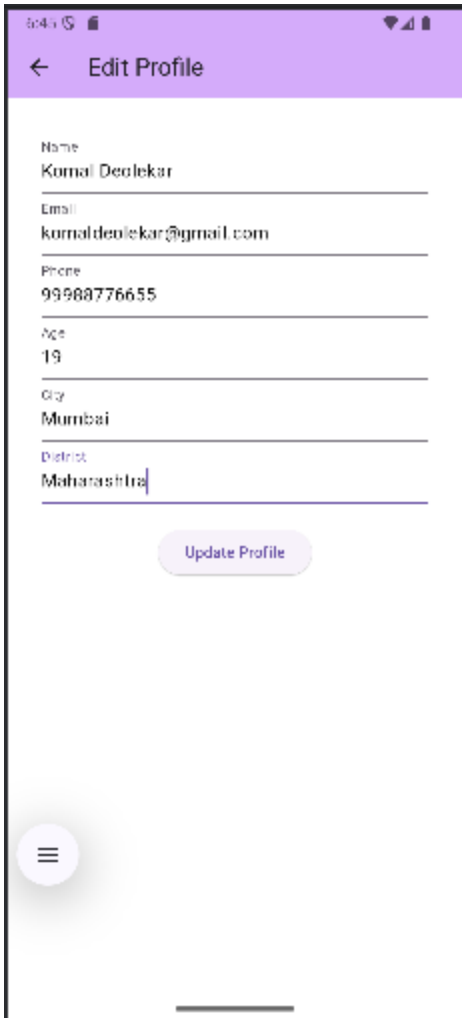
ScreenShots :





Project Title: MagicPin **Roll No.: 14**





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.	14
Name	Komal Milind Deolekar
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	

AIM : To write meta data of your PWA in a Web app manifest file to enable "add to homescreen feature".

Theory :

Online Reference:
https://developer.mozilla.org/en-US/docs/Web/Manifest
https://www.geeksforgeeks.org/making-a-simple-pwa-under-5-minutes/

Regular Web App

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

Progressive Web App

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

Difference between PWAs vs. Regular Web Apps:

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

1. Native Experience

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

2. Ease of Access

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

3. Faster Services

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

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

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

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

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

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

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

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

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

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

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

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

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

Linkable — Easily shared via URL without complex installations.

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

Weaknesses refer to:

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

Steps :

1. Create manifest.json file

Create a file named manifest.json in your project root or public/ folder.

```
{
 "name": "My Awesome App",
 "short_name": "AwesomeApp",
 "description": "An awesome PWA that does cool things!",
 "start_url": "/",
 "display": "standalone",
 "background_color": "#ffffff",
 "theme_color": "#1976d2",
 "icons": [
  {
   "src": "/icons/icon-192x192.png",
   "sizes": "192x192",
   "type": "image/png"
```

```
  },
  {
   "src": "/icons/icon-512x512.png",
   "sizes": "512x512",
   "type": "image/png"
  }
 ]
}
```

2. Link the manifest file in your HTML

In your main HTML file (index.html), add the following inside the <head> tag:

```
<link rel="manifest" href="/manifest.json">
<meta name="theme-color" content="#1976d2">
```

3. Add icons

Place your app icons in the specified path (e.g., /icons/icon-192x192.png and /icons/icon-512x512.png).

Make sure they meet these requirements:

- PNG format
- Sizes: 192x192 and 512x512 at least

4. Serve over HTTPS

PWA features like "Add to Home Screen" require HTTPS (except on localhost for development).

5. Register a service worker (mandatory for PWA)

In your JS file (e.g., main.js or index.js): (Here I stored in index.html)

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

6. Add a minimal service-worker.js

At the root level:

```
self.addEventListener('install', event => {
  console.log('Service Worker installed');
});

self.addEventListener('fetch', event => {
  // You can cache requests here
});
```

Done! Now your app will prompt “Add to Home Screen” when:

- It has a manifest.json
- A service worker is registered
- Served over HTTPS
- Visited more than once by the use

Code :

manifest.json:-

```
{
   "name": "VESIT college",
   "short_name": "VESIT",
   "start_url": "/",
   "display": "standalone",
   "background_color": "#5900b3",
   "theme_color": "black",
   "scope": ".",
   "description": "This is a PWA tutorial.",
   "icons": [
     {
       "src": "/vesitlogo.jpg",
       "sizes": "192x192",
       "type": "image/jpg"
     },
     {
       "src": "/google_logo.png",
       "sizes": "512x512",
       "type": "image/png"
     }
   ],
   "screenshots": [
  {
   "src": "/screenshot1.png",
```

```
    "type": "image/png",
    "sizes": "1280x720",
    "form_factor": "wide"
  },
  {
    "src": "/screenshot2.png",
    "type": "image/png",
    "sizes": "720x1280",
    "form_factor": "narrow"
  }
]

  }
```

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

index.html

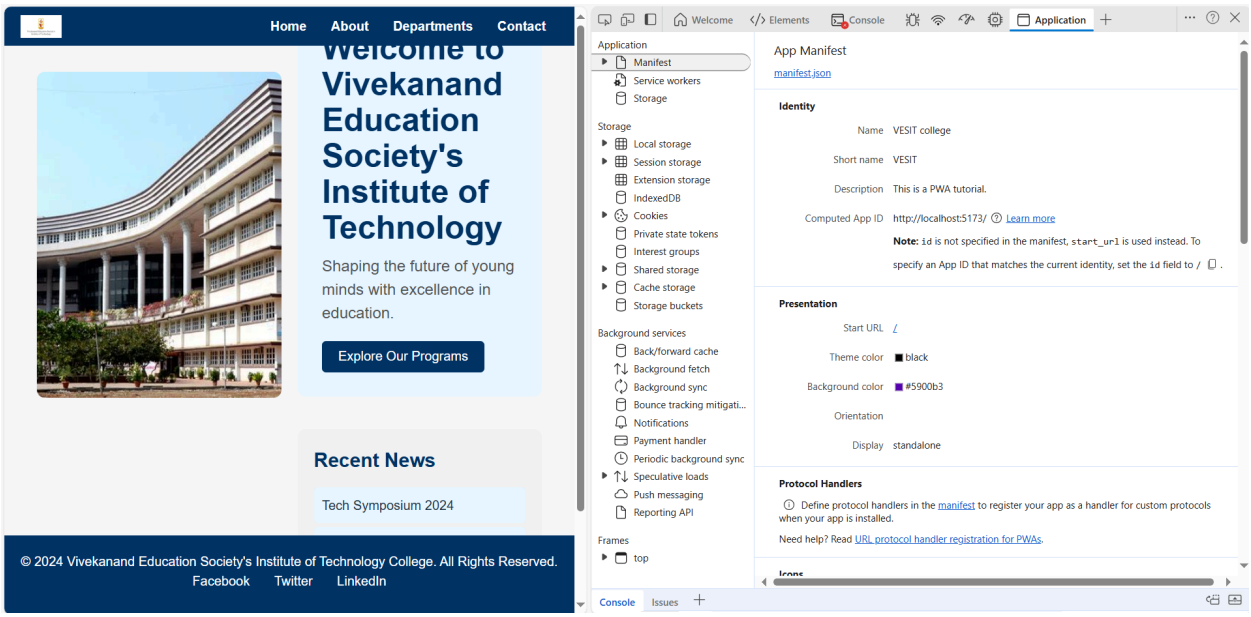
```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name=
"apple-mobile-web-app-status-bar"
                        content="#aa7700">
<meta name="theme-color"
                        content="black">

<!-- Manifest File link -->
<link rel="manifest"
                        href="/manifest.json">
    <title>Vite + React</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
    <script>
if ('serviceWorker' in navigator) {
  navigator.serviceWorker.register('/serviceworker.js')
    .then(function(registration) {
      console.log('Service Worker registered with scope:', registration.scope);
    })
    .catch(function(error) {
      console.log('Service Worker registration failed:', error);
    });
}
</script>
  </body></html>
```

Project Title: VESIT Website **Roll No.: 14**

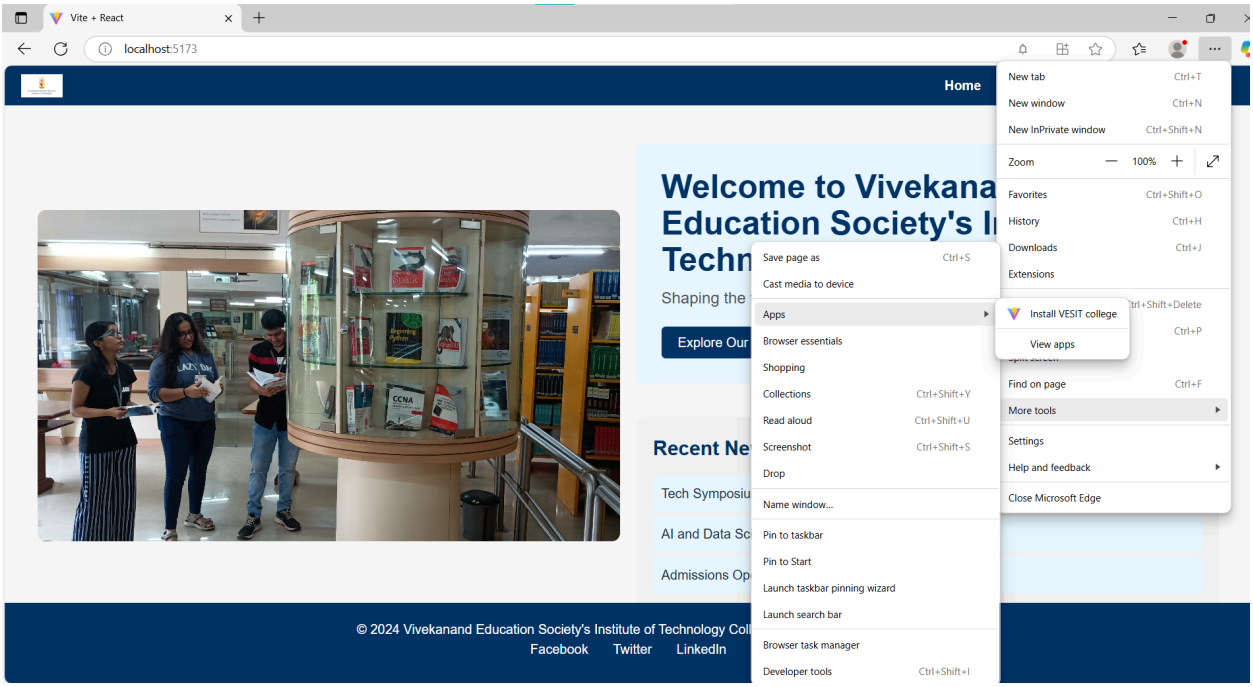
Output :

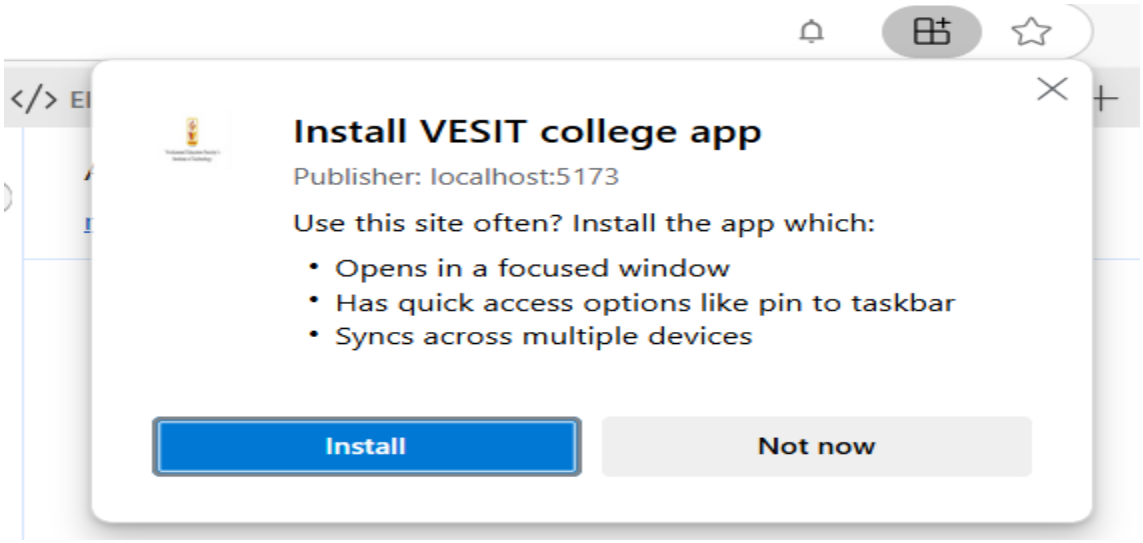
For Edge

open Developer tools options -> Application

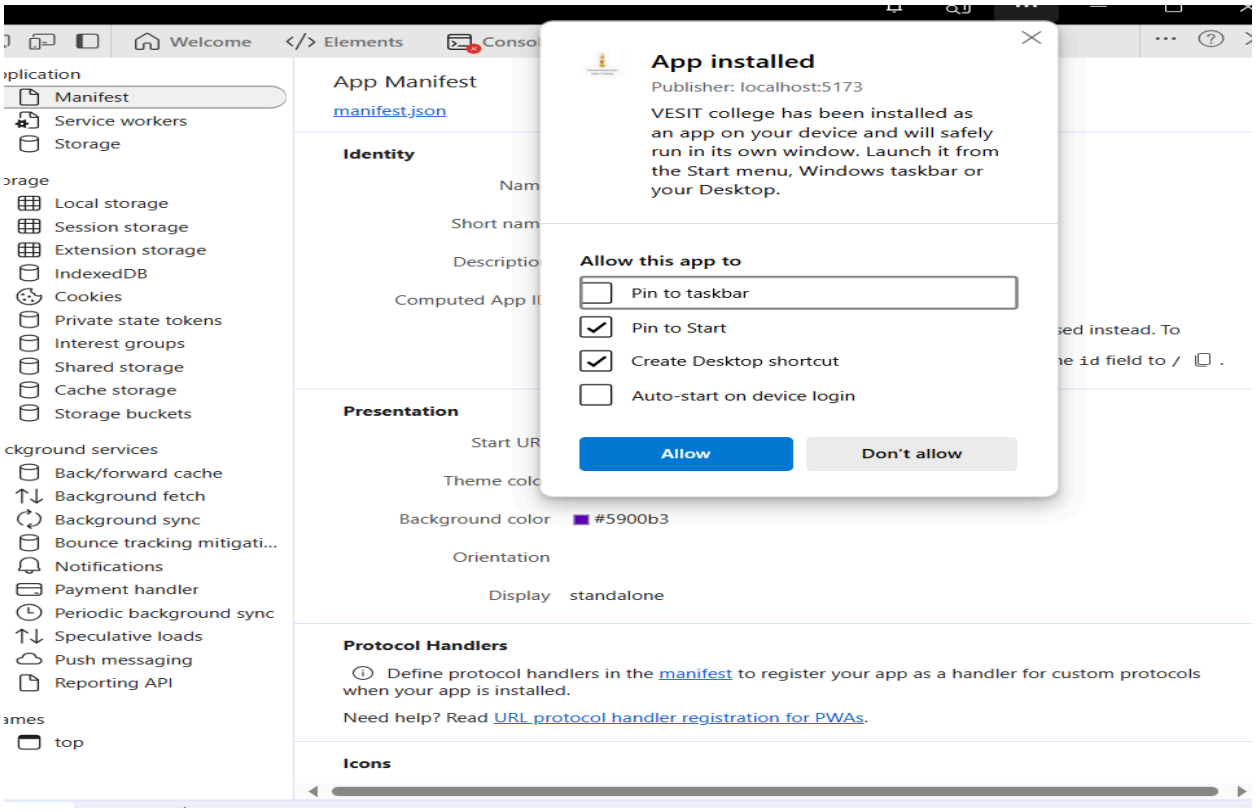


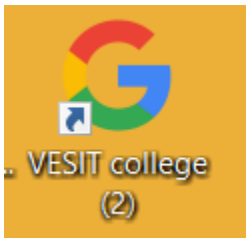
click on 3 dots on top right corner of browser from app option install this site as an app
3 dots -> more tools - > apps -> Install





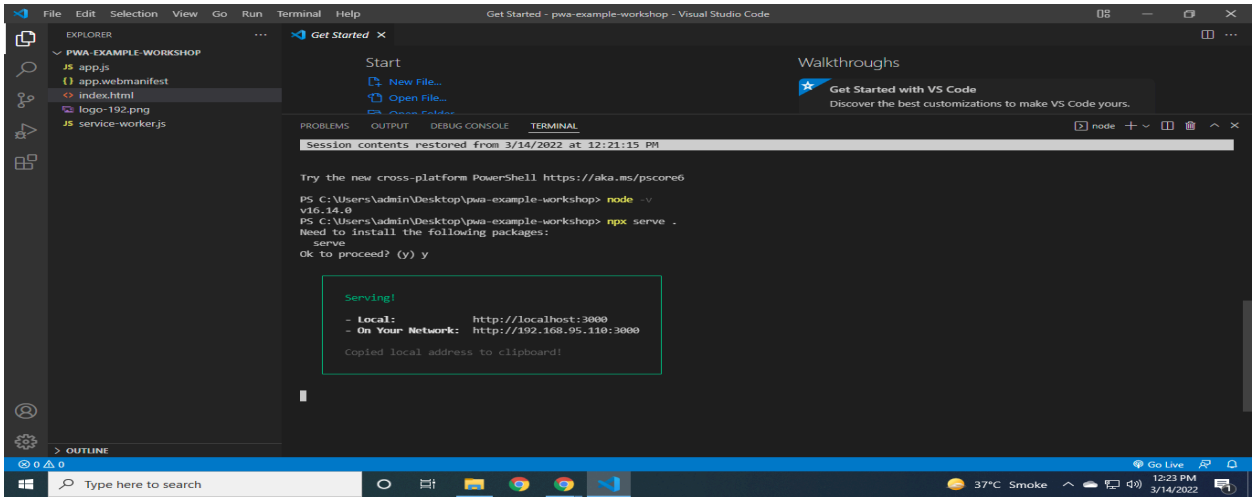
Check create desktop shortcut - > allow

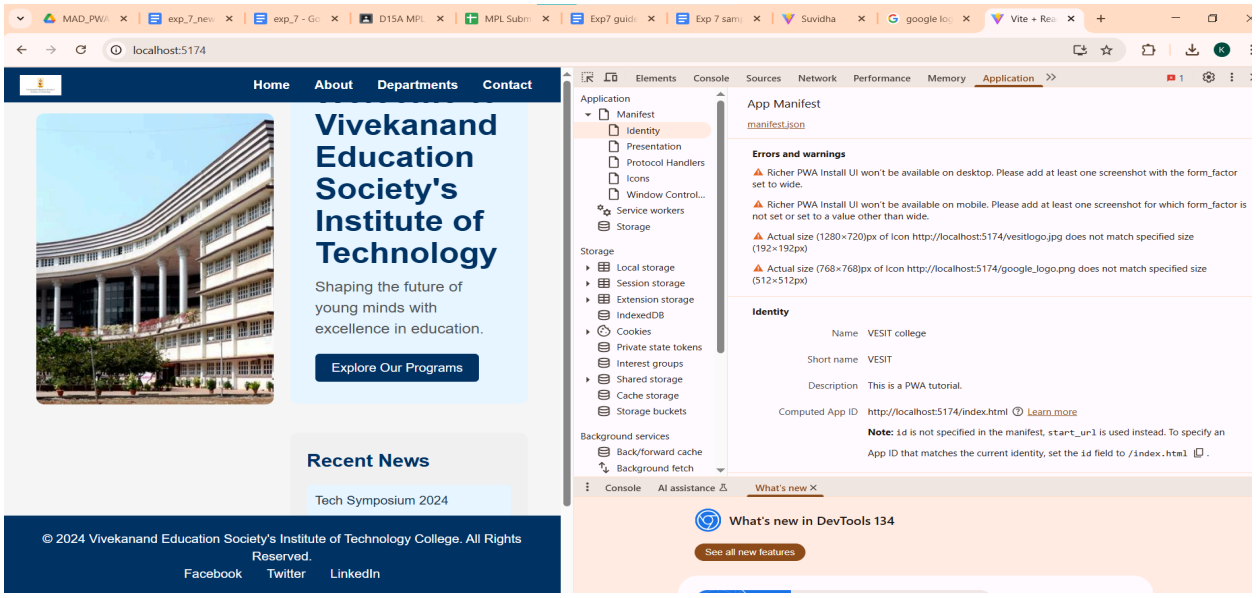


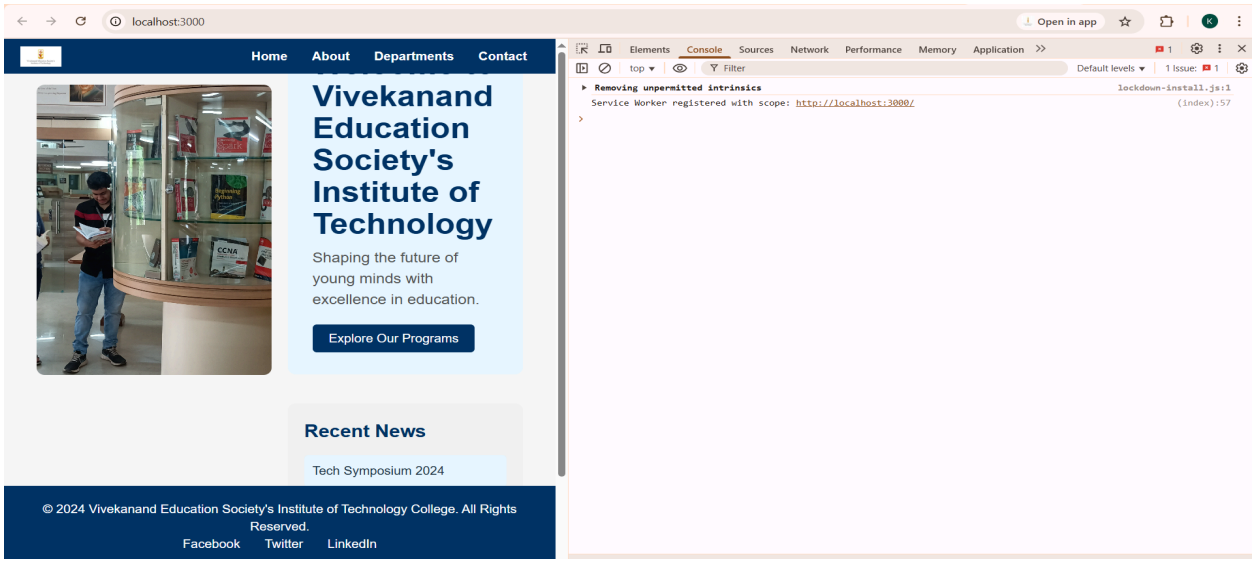


Project Title: VESIT Website **Roll No.: 14**

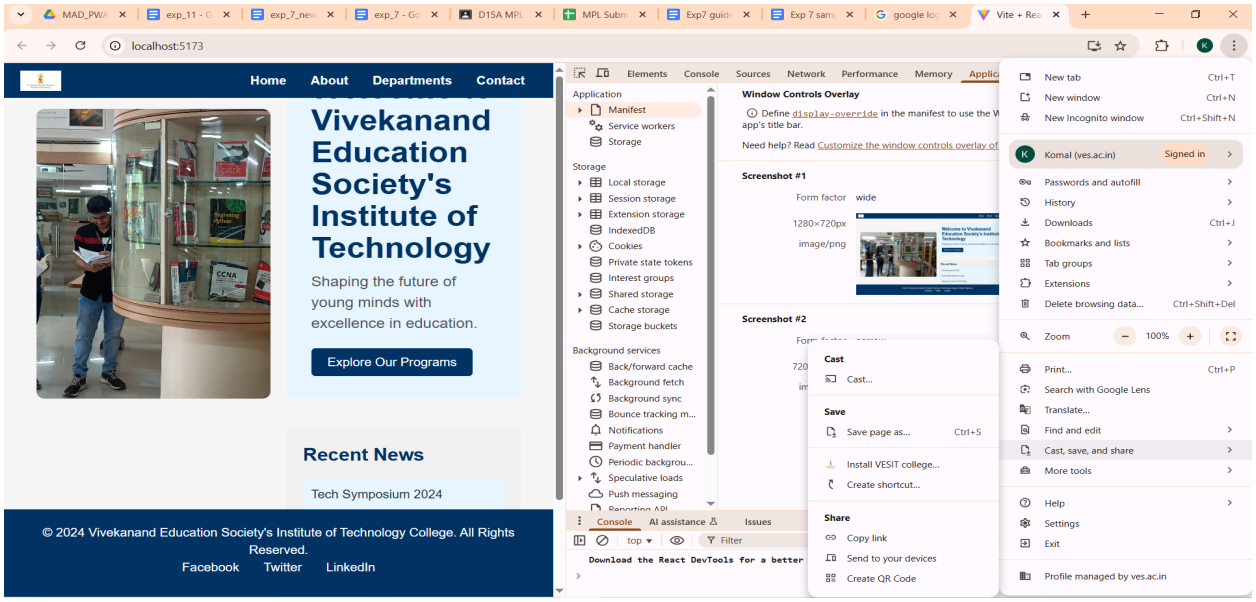
For Chrome

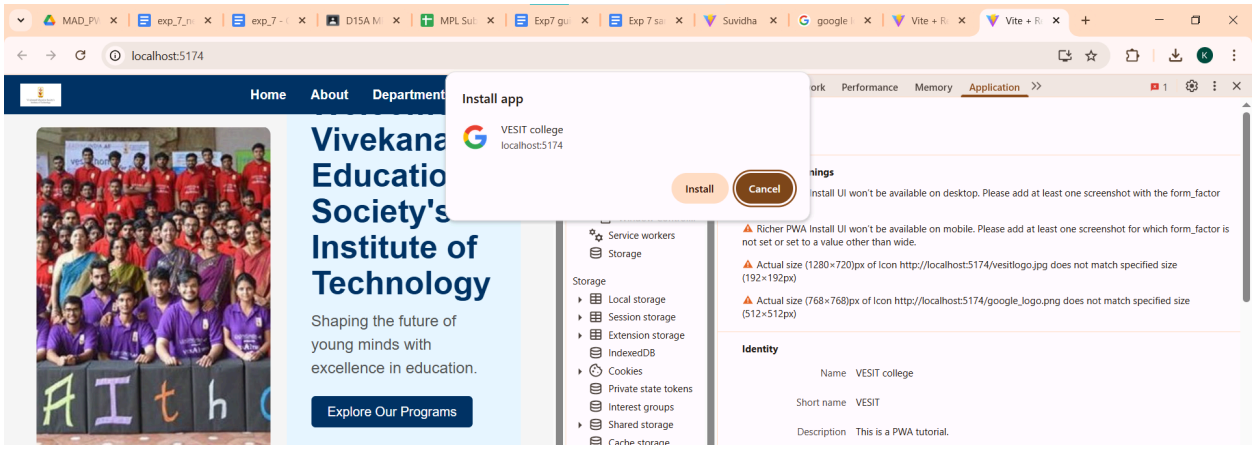


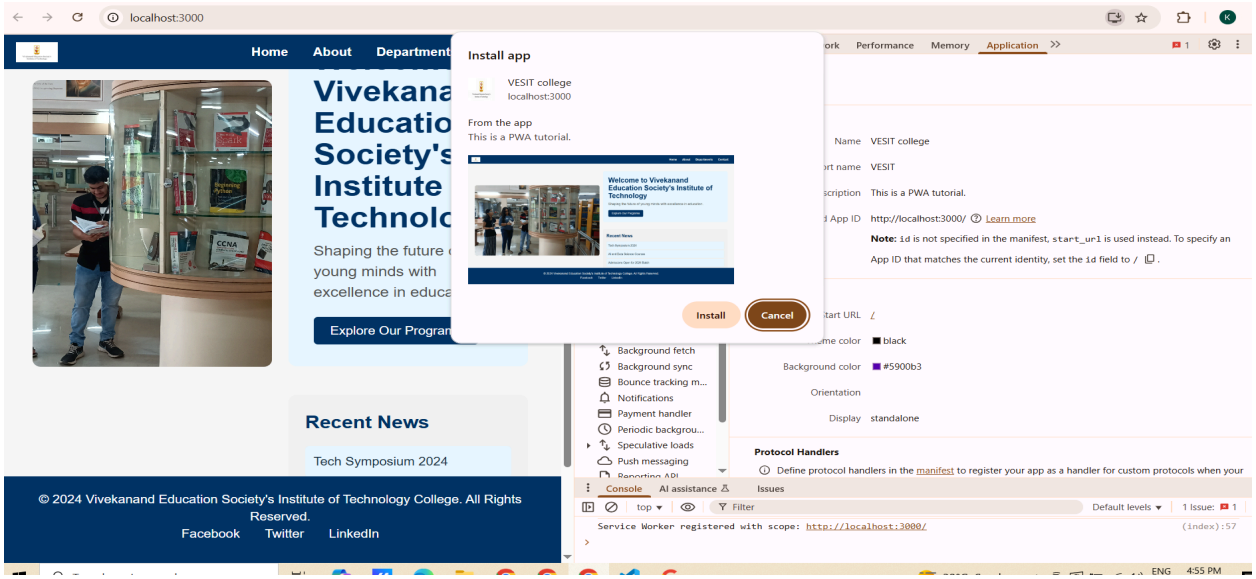




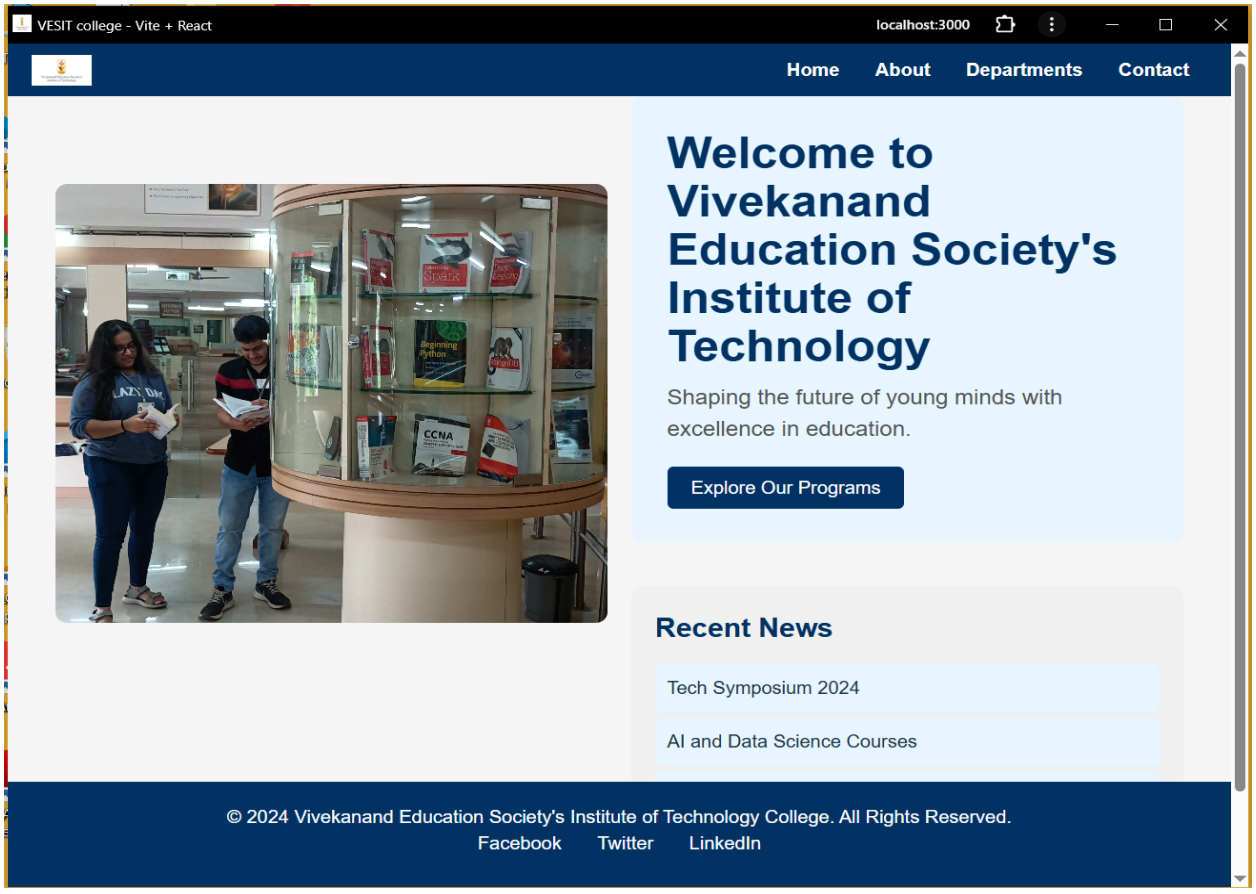
Project Title: VESIT Website **Roll No.: 14**

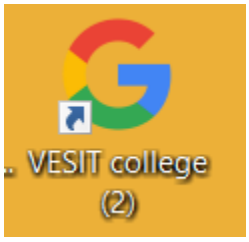






Project Title: VESIT Website **Roll No.: 14**





Conclusion :

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

https://medium.com/@svinkle/start-a-local-live-reload-web-server-with-one-command-72f99bc6e855

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.	14
Name	Komal Milind Deolekar
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

AIM : To code and register a service worker, and complete the install and activation process for a new service worker for the Website 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.

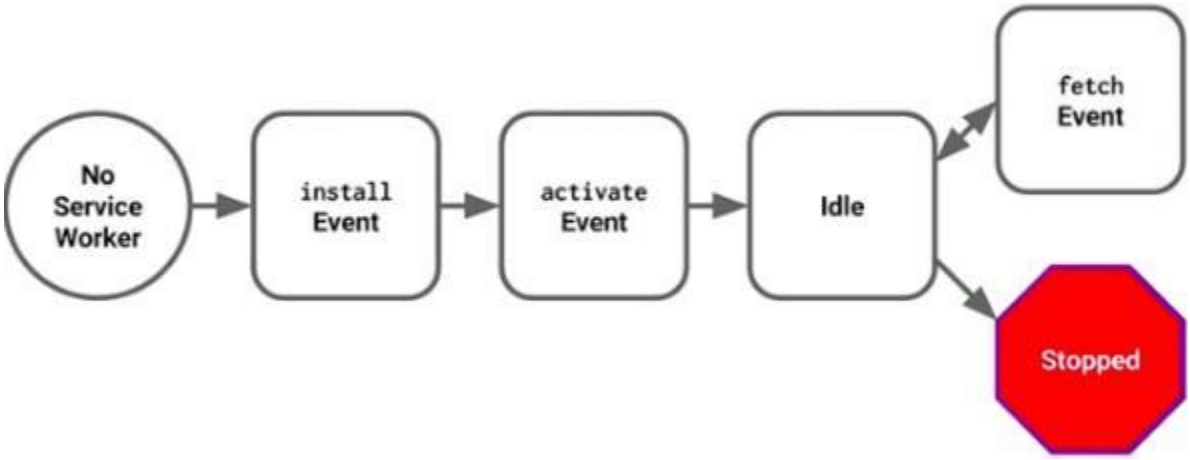
- You can manage **Push Notifications**
  You can manage push notifications with Service Worker and show any information message to the user.
- You can **Continue**
  Although Internet connection is broken, you can start any process with Background Sync of Service Worker.

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-work
  er.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 :

serviceworker.js

```
const cacheName = 'vesit-pwa-v1';
const assetsToCache = [
   '/',            // Make sure this loads index.html
 '/manifest.json', // Should be inside public/
 '/vite.svg'
   // Add more files and assets here as needed
];


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


self.addEventListener('activate', event => {
   event.waitUntil(
      caches.keys().then(cacheNames => {
         return Promise.all(
            cacheNames.filter(name => {
               return name !== cacheName;
            }).map(name => {
               return caches.delete(name);
            })
         );
      })
   );
});

self.addEventListener("fetch", function (event) {
            event.respondWith(
             caches.match(event.request).then(function (response) {
```

```
            return response || fetch(event.request);
          })
        );
});

self.addEventListener("fetch", function (event) {
        event.respondWith(
          caches.match(event.request).then(function (response) {
            return response || fetch(event.request);
          })
        );
});
```

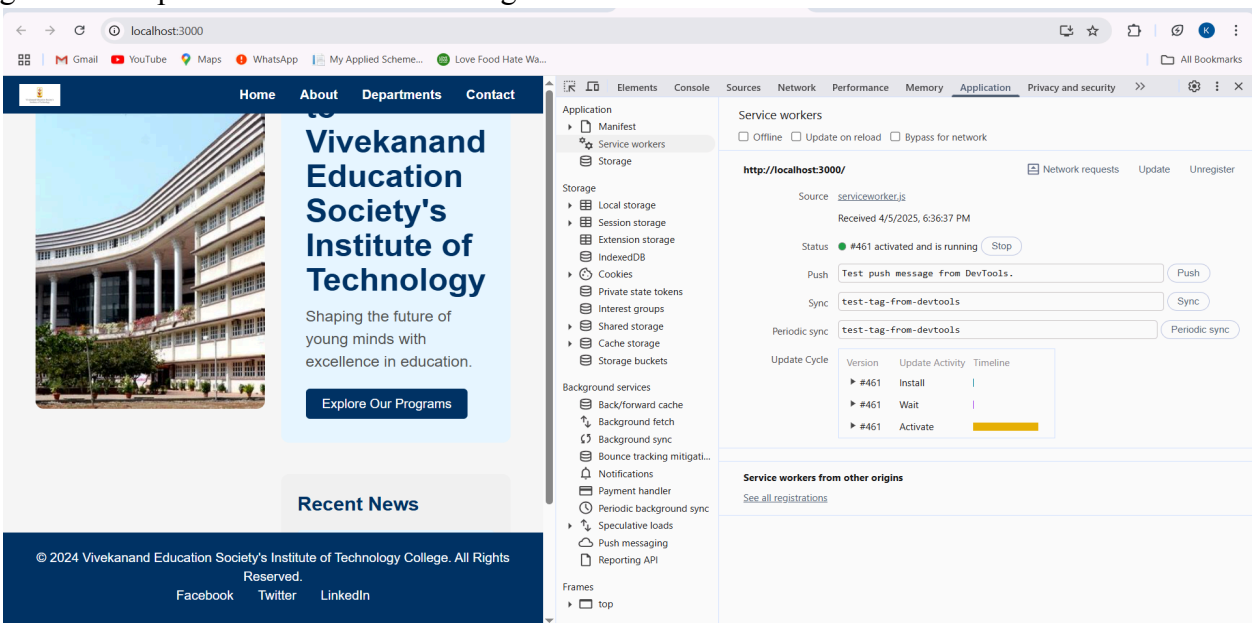
Index.html

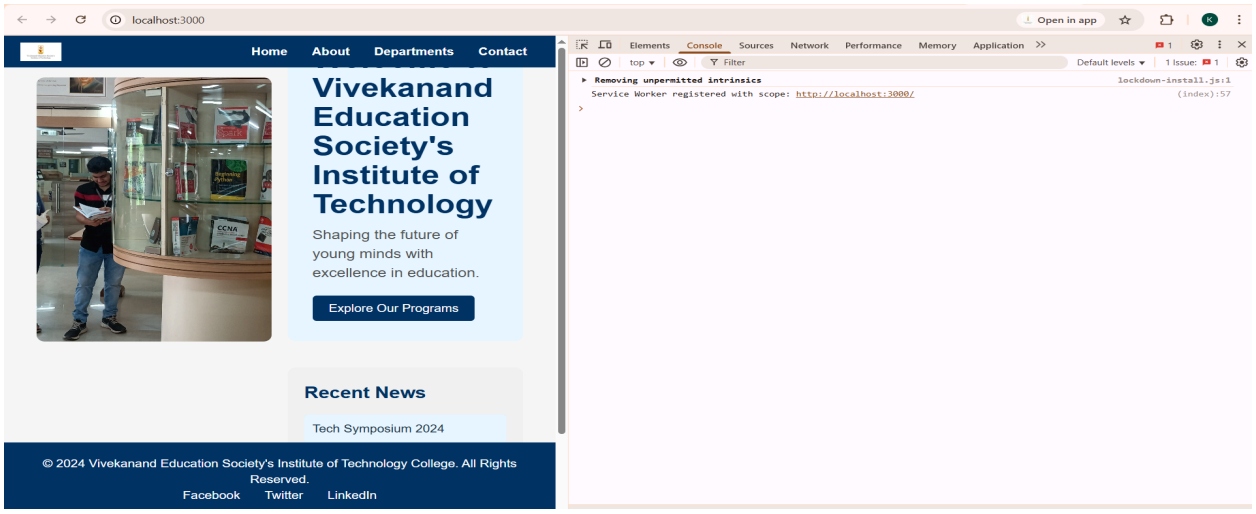
```
<!doctype html>
<html lang="en">
 <head>
  <meta charset="UTF-8" />
  <link rel="icon" type="image/svg+xml" href="/vite.svg" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name=
"apple-mobile-web-app-status-bar"
                content="#aa7700">
<meta name="theme-color"
                content="black">

<!-- Manifest File link -->
<link rel="manifest"
                href="/manifest.json">
  <title>Vite + React</title>
 </head>
 <body>
  <div id="root"></div>
  <script type="module" src="/src/main.jsx"></script>
  <script>
if ('serviceWorker' in navigator) {
 navigator.serviceWorker.register('/serviceworker.js')
  .then(function(registration) {
   console.log('Service Worker registered with scope:', registration.scope);
  })
  .catch(function(error) {
   console.log('Service Worker registration failed:', error);
  });
}
</script>
 </body>
```

Project Title: VESIT Website **Roll No.: 14**

Output :

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





Conclusion :

To enable PWA features, we code and register a service worker that caches essential assets for offline access. Completing the install and activate steps ensures the service worker takes control and enhances the app's reliability and performance.

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.	14
Name	Koml Milind Deolekar
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

AIM : To implement Service worker events like fetch, sync and push for Website PWA.

Theory:

Service Worker

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

Things to note about Service Worker:

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

Fetch Event

You can track and manage page network traffic with this event. You can check existing cache, manage “cache first” and “network first” requests and return a response that you want.
Of course, you can use many different methods but you can find in the following example a “cache first” and “network first” approach. In this example, if the request’s and current location’s origin are the same (Static content is requested.), this is called “cacheFirst” but if you request a targeted external URL, this is called “networkFirst”.

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

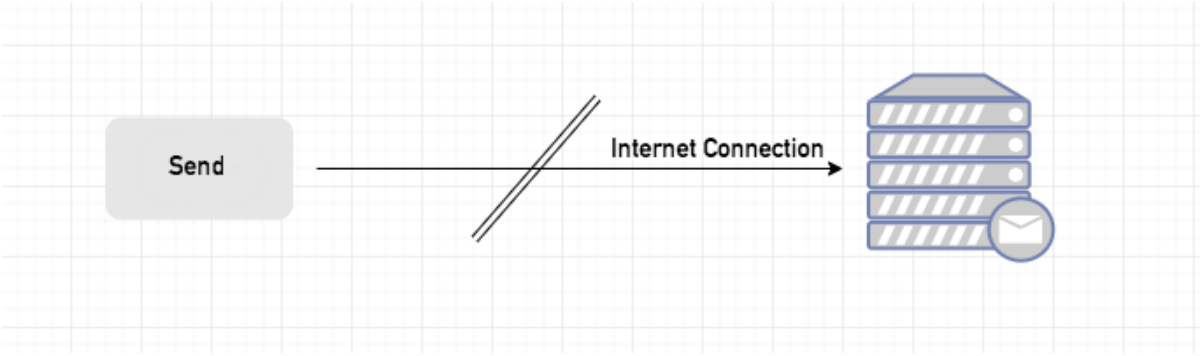
```
self.addEventListener("fetch", function (event) {
    const req = event.request;
    const url = new URL(req.url);

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

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

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

Sync Event

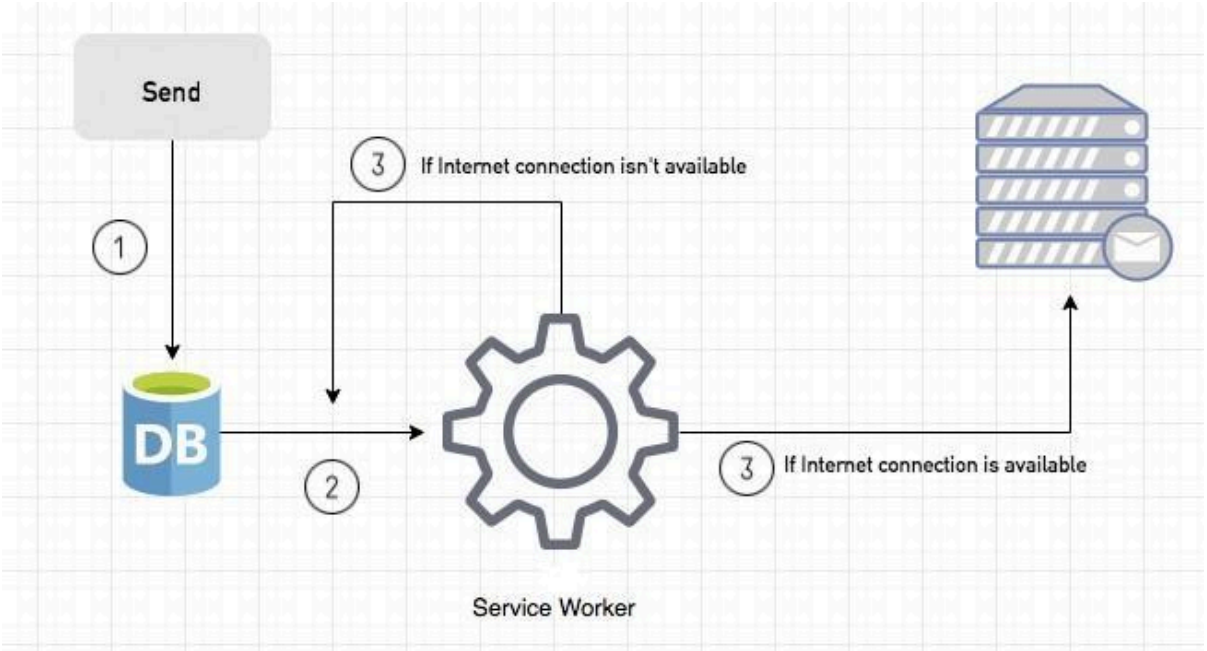
Background Sync is a Web API that is used to delay a process until the Internet connection is stable. We can adapt this definition to the real world; there is an e-mail client application that works on the browser and we want to send an email with this tool. Internet connection is broken while we are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

Here is a job for the Background Sync.

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



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



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

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

Event Listener for Background Sync Registration

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

Event Listener for sw.js

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

Push Event

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

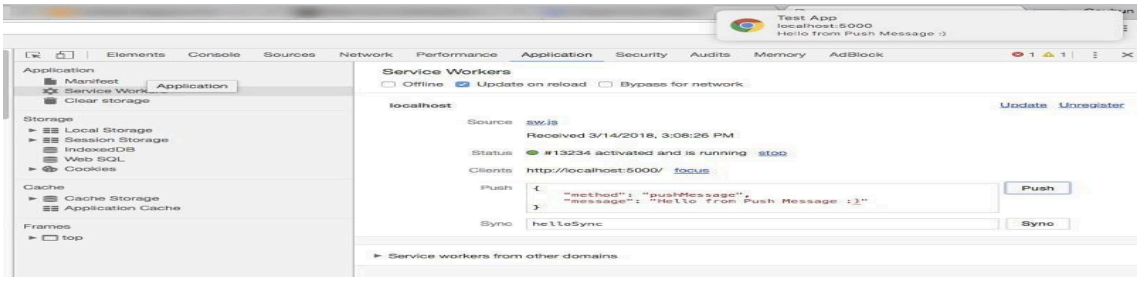
We can check in the following example.
“Notification.requestPermission();” is the necessary line to show notification to the user. If you don’t want to show any notification, you don’t need this line.

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

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

value is “pushMessage”, we open the information notification with the “message” property.

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



Code :

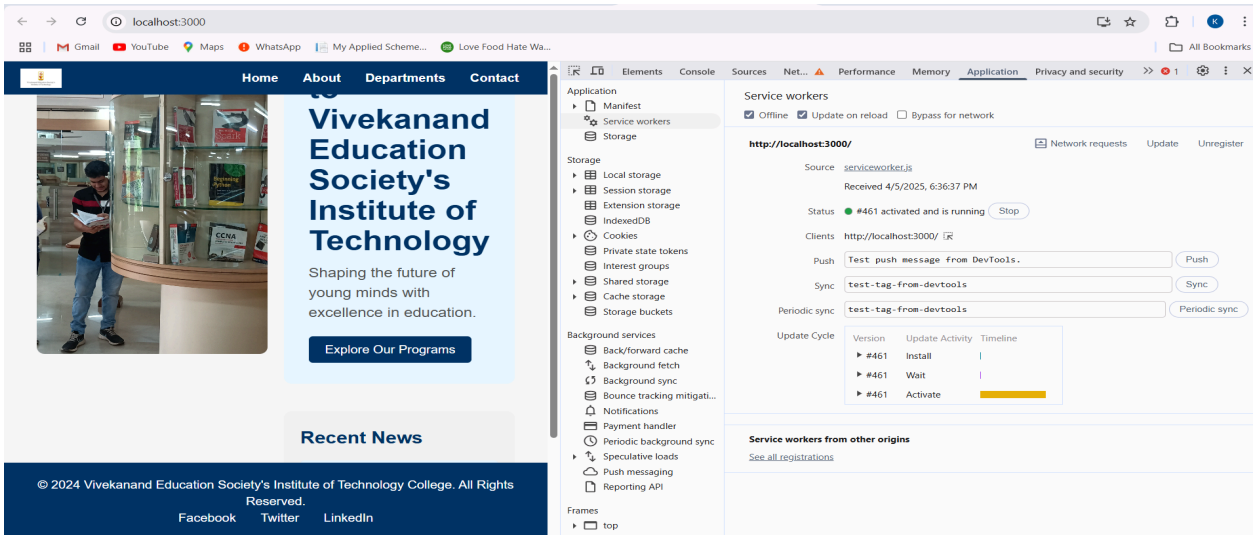
serviceworker.js

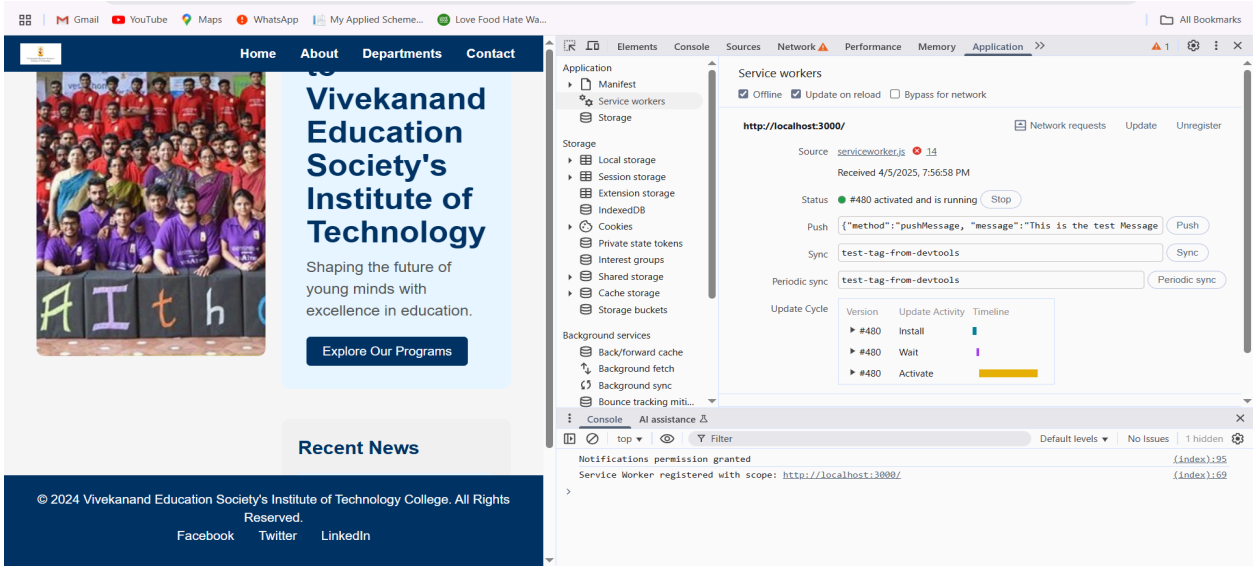
```
const cacheName = 'vesit-pwa-v1';
const assetsToCache = [
  '/',
  '/manifest.json',
  '/vite.svg',
  '/vesitlogo.jpg',
  '/screenshot1.png',
];

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

// ✅ Activate
self.addEventListener('activate', event => {
  event.waitUntil(
    caches.keys().then(keys => {
      return Promise.all(
        keys.filter(name => name !== cacheName).map(name => caches.delete(name))
      );
    })
  );
  self.clients.claim();
});

// ✅ Fetch (Offline support + Notification on failure)
self.addEventListener('fetch', event => {
  event.respondWith(
    caches.match(event.request).then(response => {
      return response || fetch(event.request).catch(() => {
        showNotification('Network Error', 'Failed to fetch: ' + event.request.url);
        return new Response('Offline', {
          status: 408,
          statusText: 'Offline'
        });
      });
    })
  );
});

// ✅ Sync (send data + show notification on success/failure)
self.addEventListener('sync', event => {
  if (event.tag === 'sync-feedback') {
    event.waitUntil(
      sendFeedbackData()
    );
  }
});

async function sendFeedbackData() {
  try {
    const offlineData = await getOfflineFeedback();
    await fetch('/api/submit-feedback', {
```

```
      method: 'POST',
      body: JSON.stringify(offlineData),
      headers: {
        'Content-Type': 'application/json'
      }
    });
    showNotification('Feedback Synced', 'Your feedback was sent successfully!');
  } catch (err) {
    showNotification('Sync Failed', 'Could not sync feedback.');
    console.error('Sync failed:', err);
  }
}

// Example mock data function
function getOfflineFeedback() {
  return Promise.resolve({
    name: 'Komal Deolekar',
    message: 'Great college app!',
    timestamp: new Date().toISOString()
  });
}

// ✅ Push (show incoming push notification)
self.addEventListener('push', event => {
  const data = event.data?.json() || {
    title: 'VESIT Update',
    body: 'Admissions open for 2025!',
    icon: '/vite.svg'
  };

  event.waitUntil(
    self.registration.showNotification(data.title, {
      body: data.body,
      icon: data.icon
    })
  );
});

// ✅ Helper: Show notification (reusable across all)
function showNotification(title, body) {
  self.registration.showNotification(title, {
    body,
    icon: '/vite.svg'
  });
}
```

Project Title: VESIT Website **Roll No.: 14**

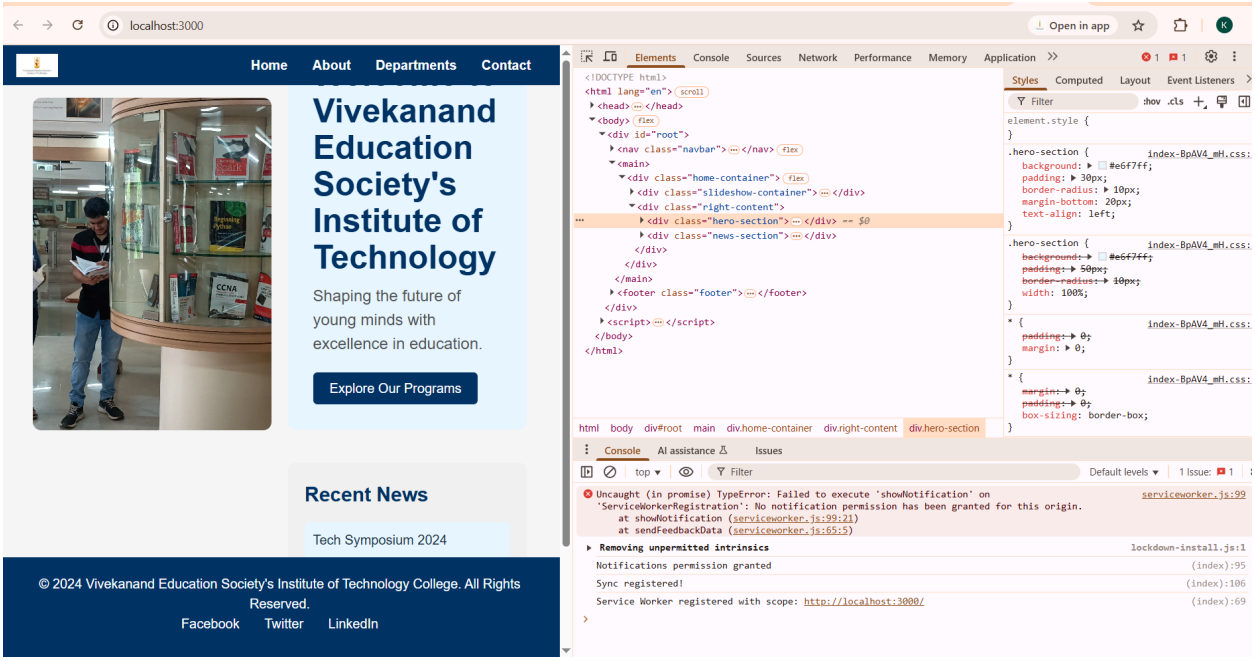
Output :

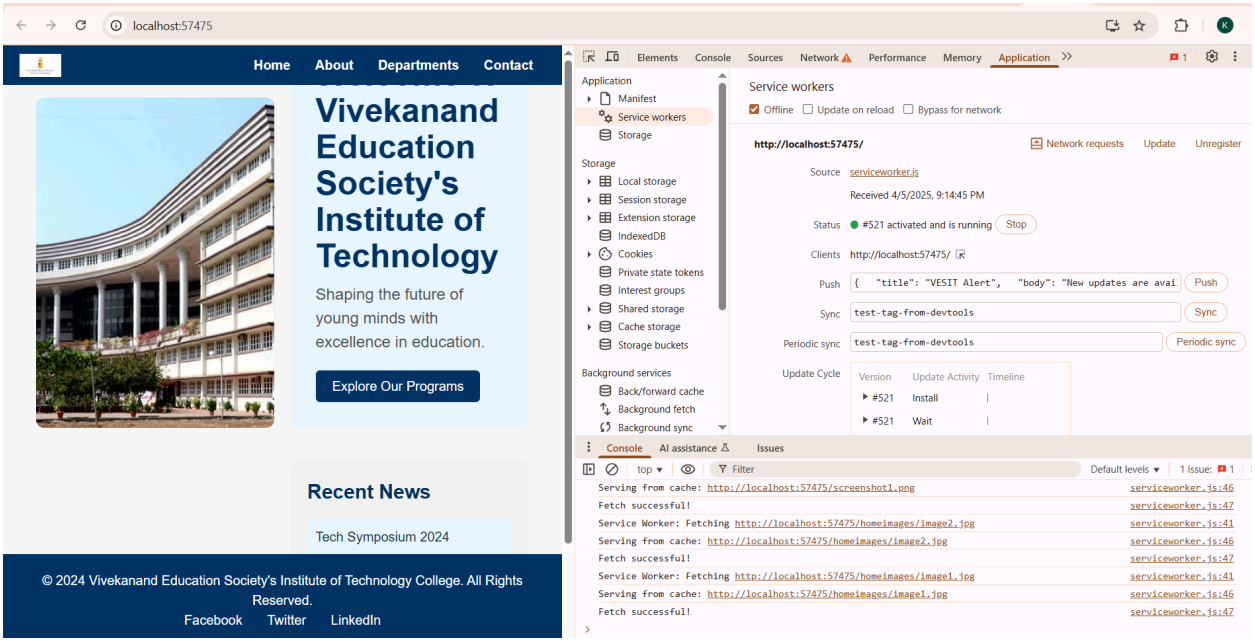




Project Title: VESIT Website **Roll No.: 14**

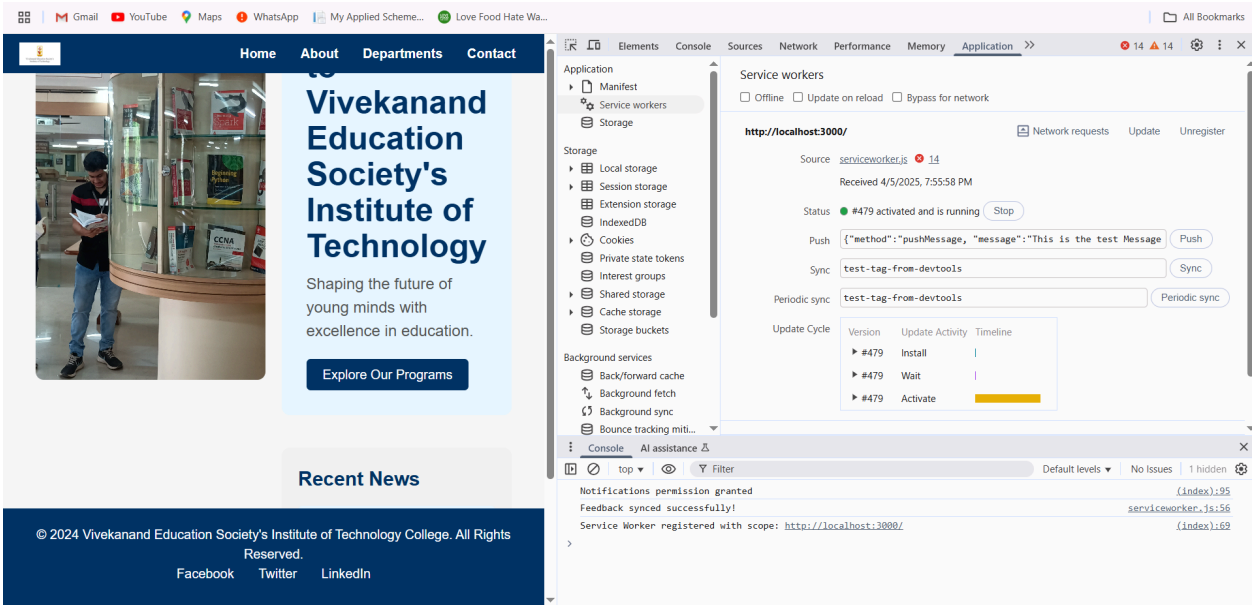
Fetch Event

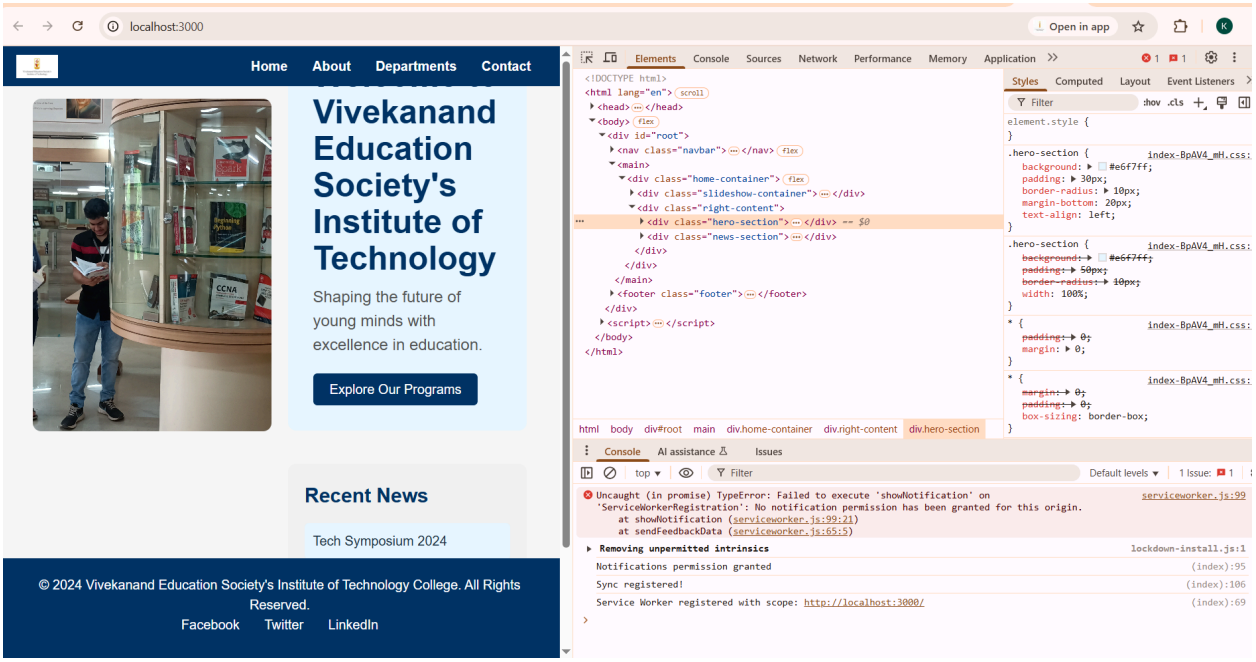




Project Title: VESIT Website **Roll No.: 14**

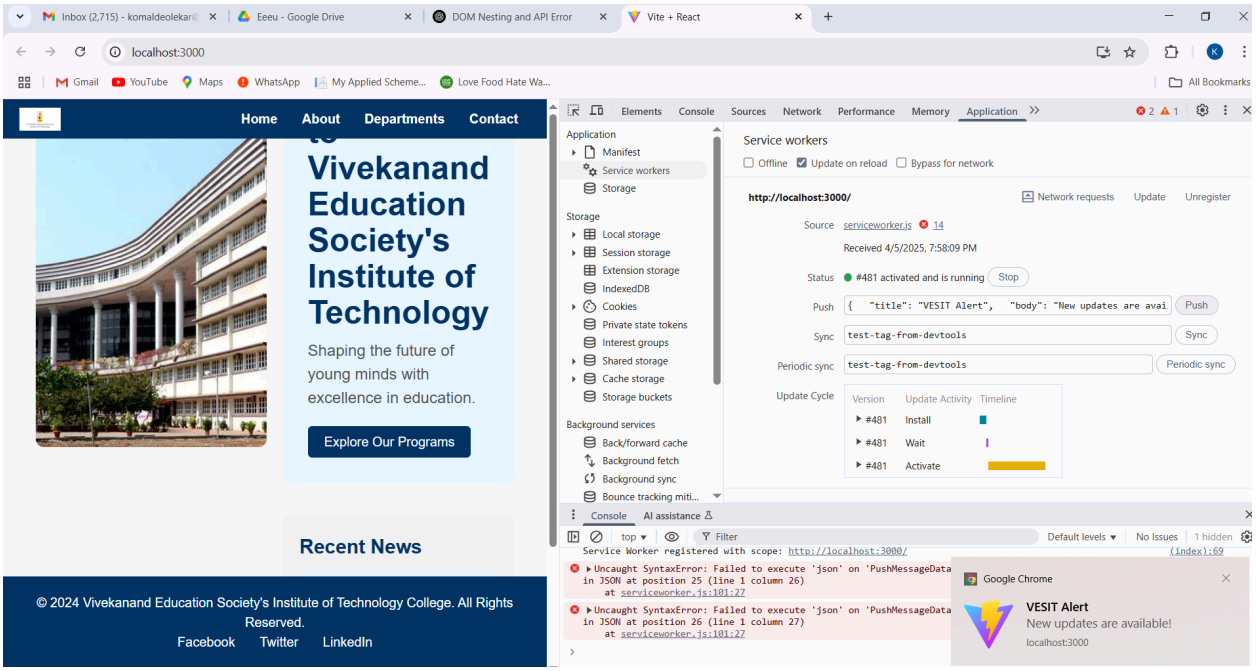
Sync Event





Project Title: VESIT Website **Roll No.: 14**

Push event



Conclusion :

By implementing service worker events like `fetch`, `sync`, and `push`, we enhance the Website PWA with offline access, background data synchronization, and real-time notifications. This ensures improved performance, reliability, and user engagement even in low or no network conditions.

MAD & PWA Lab
Journal

Experiment No.	10
Experiment Title.	To study and implement deployment of Ecommerce PWA to GitHub Pages.
Roll No.	14
Name	Komal Milind Deolekar
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	

AIM : To study and implement deployment of Website PWA to GitHub Pages.

Theory:

GitHub Pages

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

GitHub Pages provides the following key features:

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

Reasons for favoring this over Firebase:

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

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

Pros

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

Cons

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

Firebase

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

Some of the features offered by Firebase are:

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

Reasons for favoring over GitHub Pages:

1. Realtime backend made easy
2. Fast and responsive

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

Pros

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

Cons

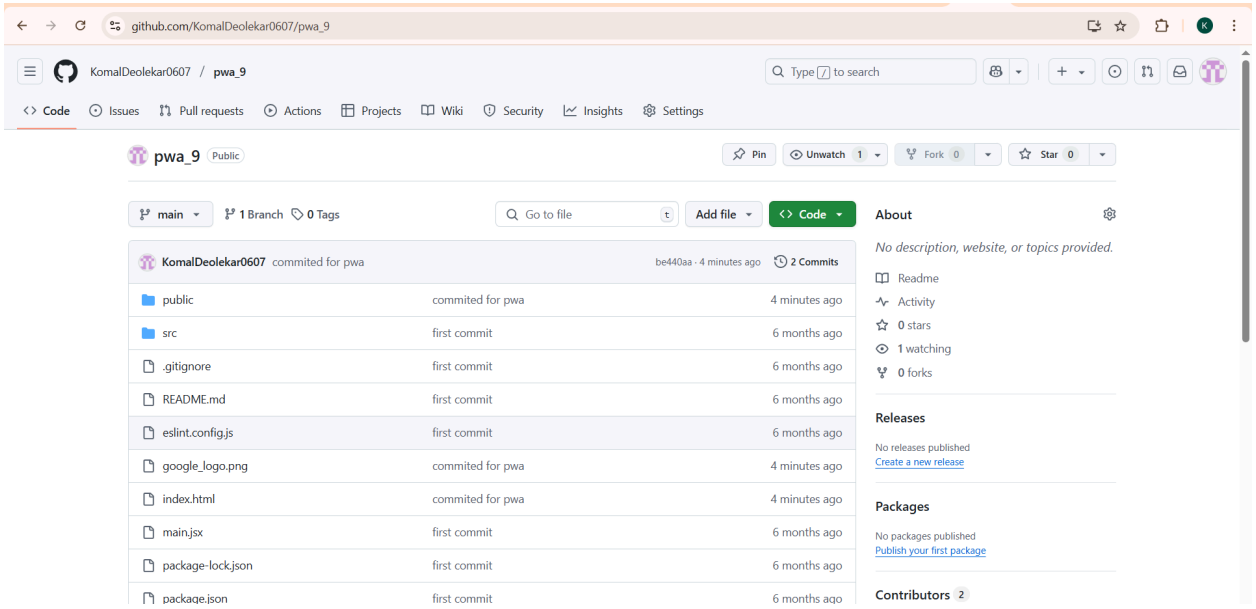
1. Only 10 GB of data transfer is allowed per month. But this is not really a big problem, if you use a CDN or AMP.
2. Command-line interface only.
3. No in-built support for any static site generator.

Link to our GitHub repository:

https://github.com/KomalDeolekar0607/pwa_9

Github Screenshot:

```
D:\Users\Komal\OneDrive\Desktop\sem 6\mpl_lab\college_website\college_site>git branch -M main

D:\Users\Komal\OneDrive\Desktop\sem 6\mpl_lab\college_website\college_site>git remote add origin https://github.com/KomalDeolekar0607/pwa_9.git
error: remote origin already exists.

D:\Users\Komal\OneDrive\Desktop\sem 6\mpl_lab\college_website\college_site>git remote
origin

D:\Users\Komal\OneDrive\Desktop\sem 6\mpl_lab\college_website\college_site>git remote add pwa https://github.com/KomalDeolekar0607/pwa_9.git

D:\Users\Komal\OneDrive\Desktop\sem 6\mpl_lab\college_website\college_site>git push -u pwa main
Enumerating objects: 52, done.
Counting objects: 100% (52/52), done.
Delta compression using up to 8 threads
Compressing objects: 100% (51/51), done.
Writing objects: 100% (52/52), 2.71 MiB | 99.00 KiB/s, done.
Total 52 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/KomalDeolekar0607/pwa_9.git
 * [new branch]      main -> main
branch 'main' set up to track 'pwa/main'.

D:\Users\Komal\OneDrive\Desktop\sem 6\mpl_lab\college_website\college_site>
```



Conclusion :

To deploy a Website PWA on GitHub Pages, we build the project and push the production-ready files to a GitHub repository. Hosting it on GitHub Pages makes the PWA accessible via HTTPS, enabling full PWA functionality like installability and offline support.

MAD & PWA Lab
Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	14
Name	Komal Milind Deolekar
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	

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:

1. **Performance:** This score is an aggregation of how the page fared in aspects such as (but not limited to) loading speed, time taken for loading for basic frame(s), displaying meaningful content to the user, etc. To a layman, this score is indicative of how decently the site performs, with a score of 100 meaning that you figure in the 98th percentile, 50 meaning that you figure in the 75th percentile and so on.
2. **PWA Score (Mobile):** Thanks to the rise of Service Workers, app manifests, etc., a lot of modern web applications are moving towards

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

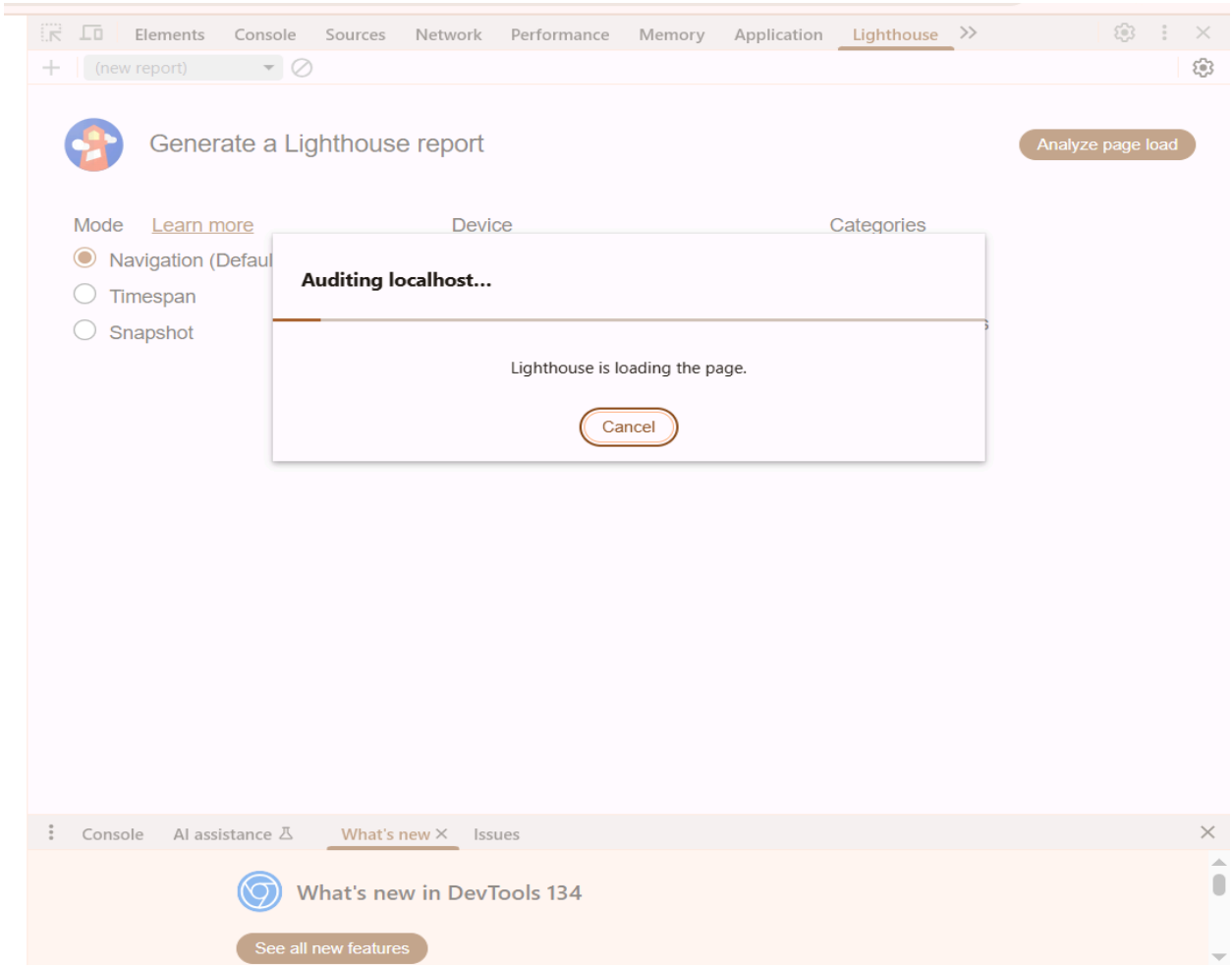
3. **Accessibility:** As you might have guessed, this metric is a measure of how accessible your website is, across a plethora of accessibility features that can be implemented in your page (such as the 'aria-' attributes like aria-required, audio captions, button names, etc.). Unlike the other metrics though, Accessibility metrics score on a pass/fail basis i.e. if all possible elements of the page are not screen-reader friendly (HTML5 introduced features that would make pages easy to interpret for screen readers used by visually challenged people like tag names, tags such as <section>, <article>, etc.), you get a 0 on that score. The aggregate of these scores is your Accessibility metric score.
4. **Best Practices:** As any developer would know, there are a number of practices that have been deemed 'best' based on empirical data. This metric is an aggregation of many such points, including but not limited to:Use of HTTPS

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

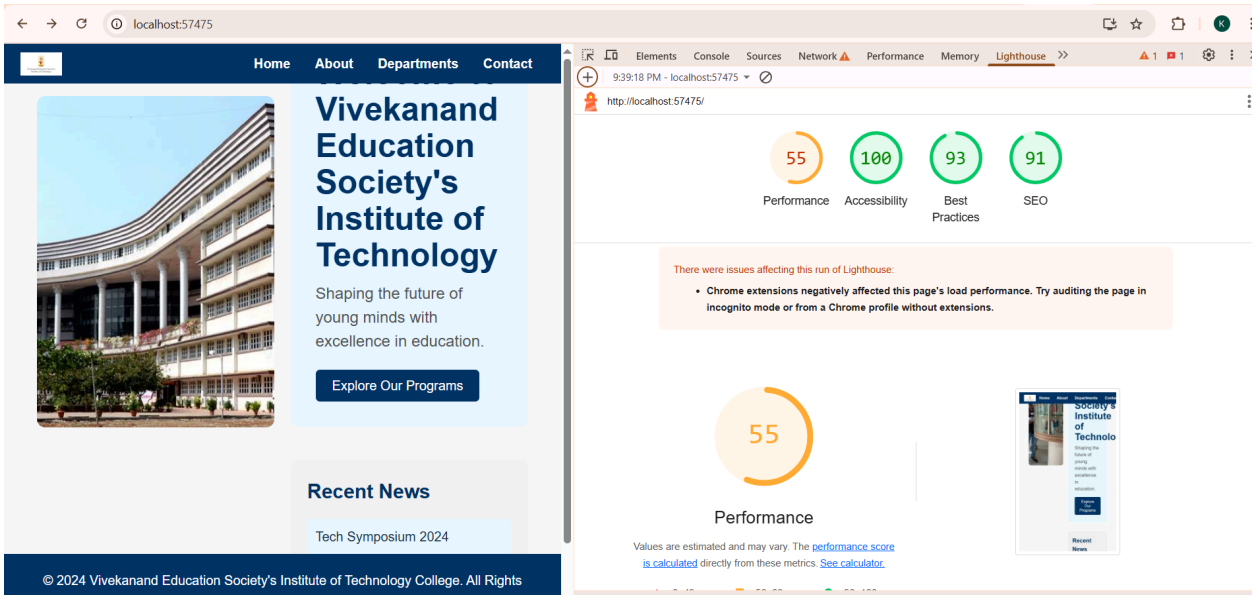
   Geo-Location and cookie usage alerts on load, etc.

Project Title: VESIT Website **Roll No.: 14**

Output :

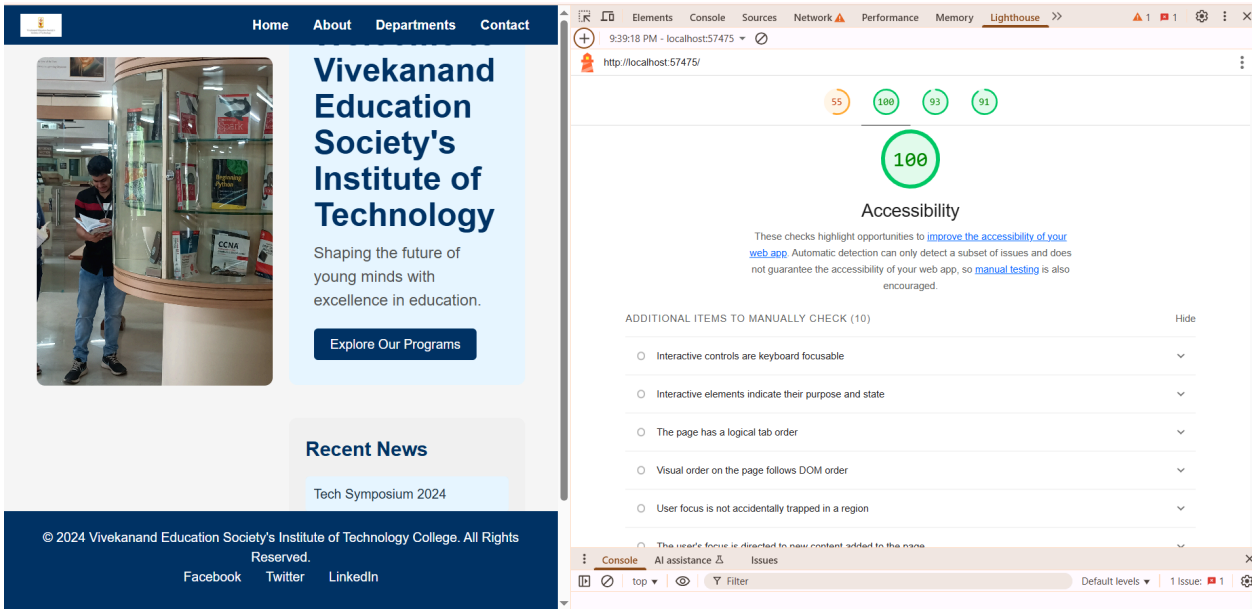


Performnce

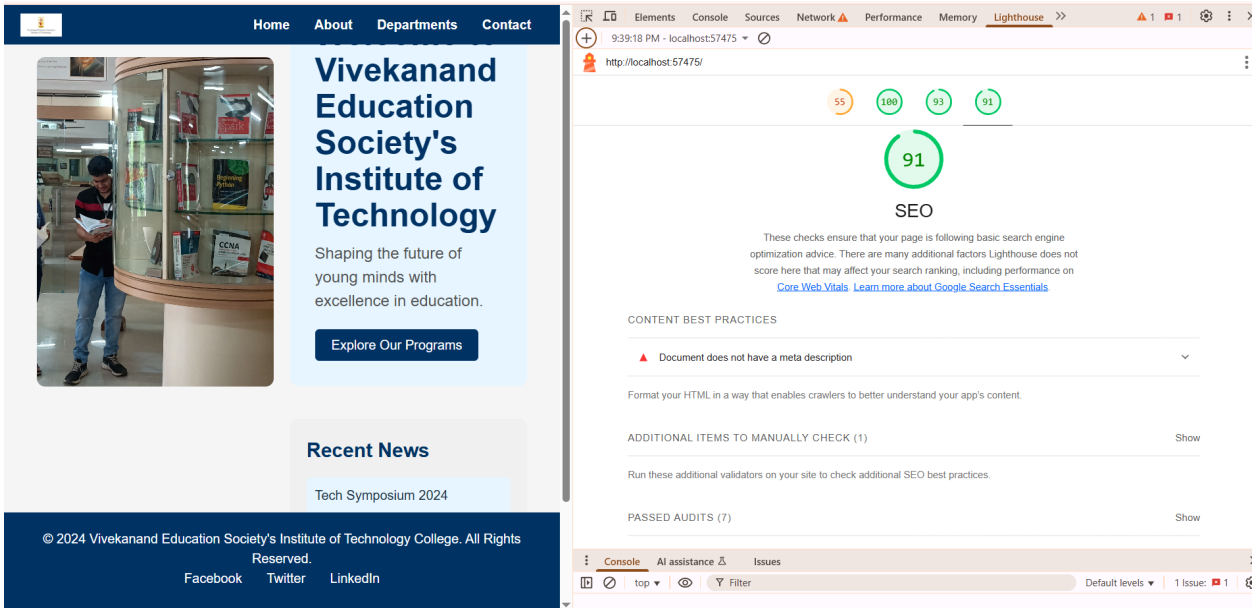


Project Title: VESIT Website **Roll No.: 14**

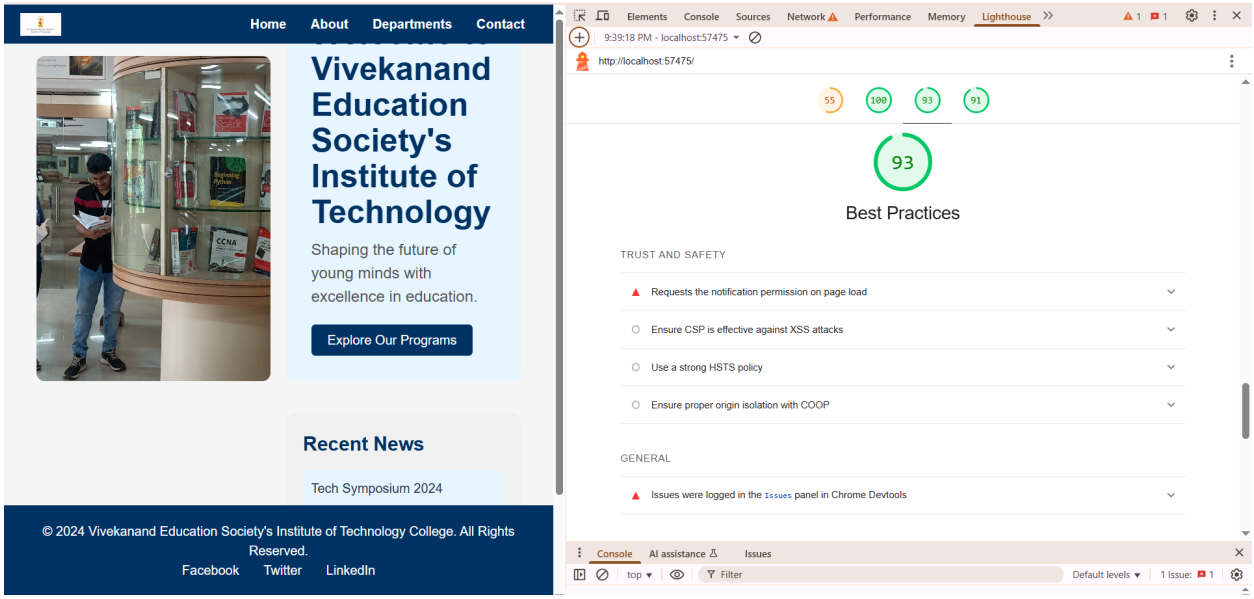
Accessibility



SEO



Best practices



Changes made to the code :

Converted Images to webp from jpg , png

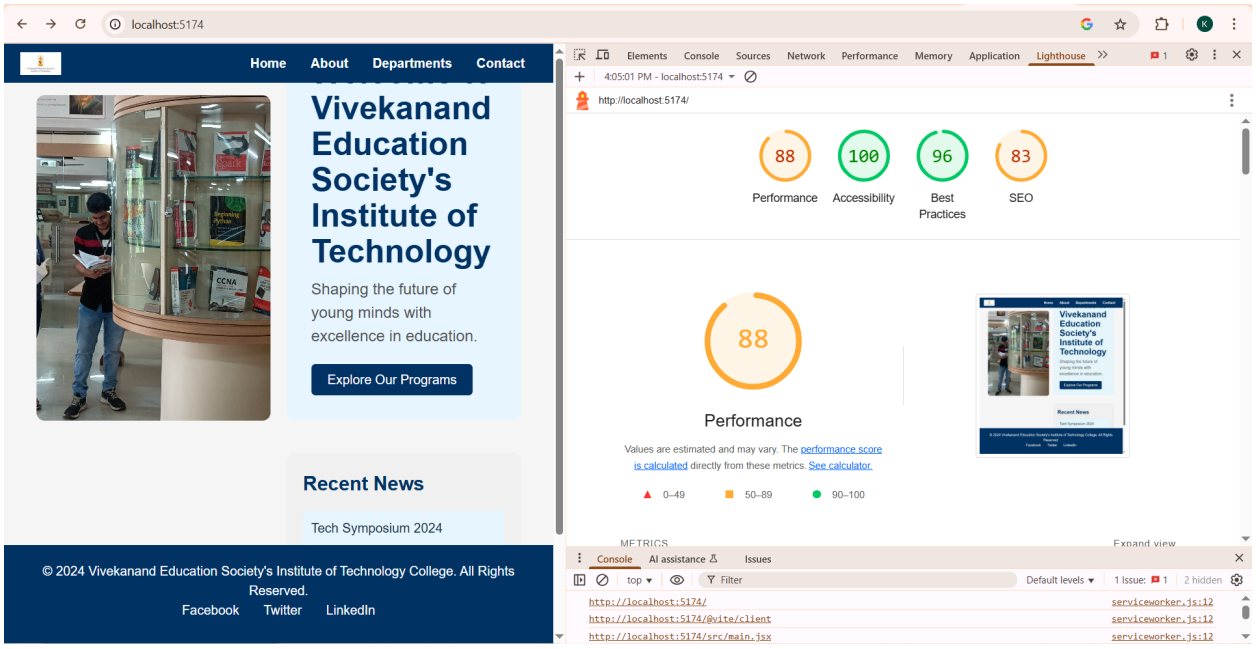
Changes in vite.config for adding compression

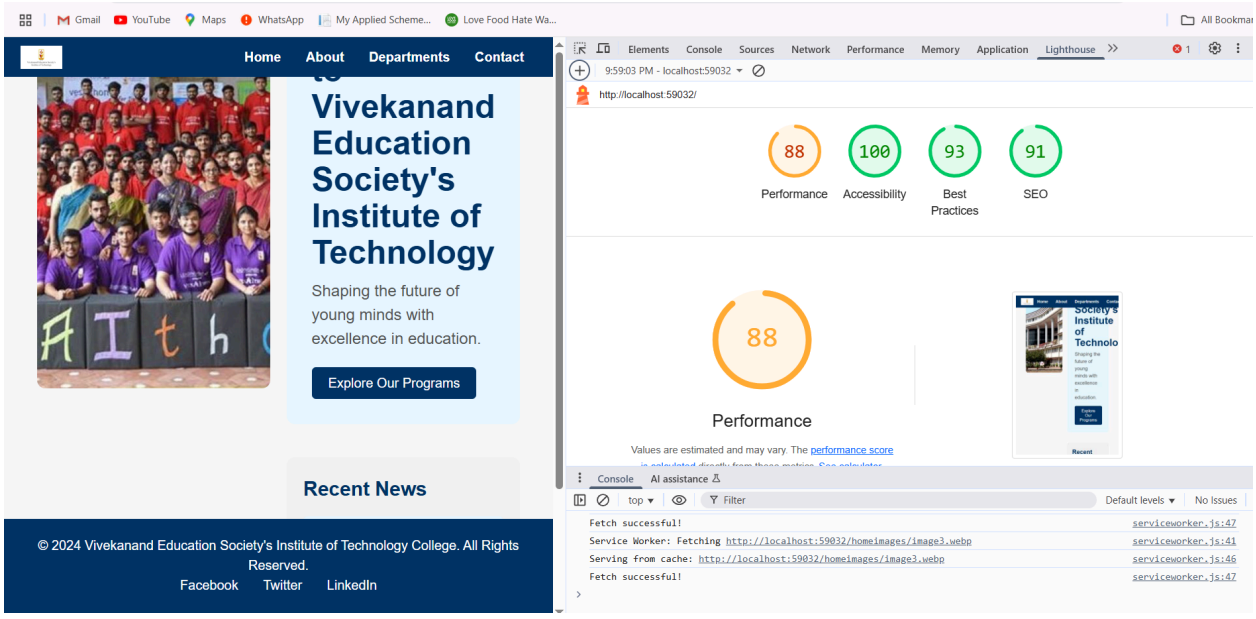
```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
import viteCompression from 'vite-plugin-compression';

// https://vitejs.dev/config/
export default defineConfig({
  plugins: [react(),viteCompression()],
  build: {
    minify: 'esbuild',
    terserOptions: {
      compress: {
        drop_console: true, // remove console logs
      }
    }
  }
})
```

For preloading this time taking asset

```
  <link rel="preload" href="/assets/index-DWhKUY4_.js" as="script">
```

Project Title: VESIT Website **Roll No.: 14**





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

MAD & PWA Lab
Journal

Experiment No.	Assignment-2
Assignment 2 Questions	1. Define Progressive Web App (PWA) and explain its significance in modern web development.Discuss the key characteristics that differentiate PWAs from traditional mobile apps 2. Define responsive web design and explain its importance in the context of Progressive Web Apps.Compare and contrast responsive, fluid, and adaptive web design approaches. 3. Describe the lifecycle of Service Workers, including registration, installation, and activation phases. 4. Explain the use of IndexedDB in the Service Worker for data storage.
Roll No.	
Name	
Class	D15A/D15B
Subject	MAD & PWA Lab
Lab Outcome	LO4:Understand various PWA frameworks and their requirements LO5: Design and Develop a responsive User Interface by applying PWA Design techniques LO6:Develop and Analyze PWA Features and deploy it over app hosting solutions
Grade:	