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

Lab Assistant

Subject Teacher

**Mrs. Kajal Joseph**

Principal

Head of Department

**Dr. Mrs. Shalu Chopra**

**Project Title: Flutter:Swiggy Clone/PWA:Todo List** **Roll No. 55**

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

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

**Faculty Incharge** : Mrs. Kajal Joseph.

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

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

**Programme Outcomes**: The graduate will be able to:

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

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

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

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

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

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

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

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

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

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

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

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

**Program specific Outcomes**

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

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

**Project Title: Flutter:Swiggy Clone/PWA:Todo List** **Roll No. 55**

**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 | 19/01 | 02/02 | 15 |
| **2.** | To design Flutter UI by including common widgets. | LO2 | 26/01 | 02/02 | 15 |
| **3.** | To include icons, images, fonts in Flutter app | LO2 | 02/02 | 09/02 | 15 |
| **4.** | To create an interactive Form using form widget | LO2 | 09/02 | 16/02 | 15 |
| **5.** | To apply navigation, routing and gestures in Flutter App | LO2 | 16/02 | 23/02 | 15 |
| **6.** | To Connect Flutter UI with fireBase database | LO3 | 23/02 | 08/03 | 15 |
| **7.** | To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”. | LO4 | 08/03 | 22/03 | 15 |
| **8.** | To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA | LO5 | 15/03 | 22/03 | 15 |
| **9.** | To implement Service worker events like fetch, sync and push for E-commerce PWA | LO5 | 22/03 | 01/04 | 15 |
| **10.** | To study and implement deployment of Ecommerce PWA to GitHub Pages. | LO5 | 29/03 | 01/04 | 15 |
| **11.** | To use google Lighthouse PWA Analysis Tool to test the PWA functioning. | LO6 | 29/03 | 01/04 | 15 |
| **12.** | Assignment-1 | LO1,LO2 ,LO3 | 28/01 | 05/02 | 05 |
| **13.** | Assignment-2 | LO4,LO5 ,LO6 | 14/03 | 21/03 | 04 |

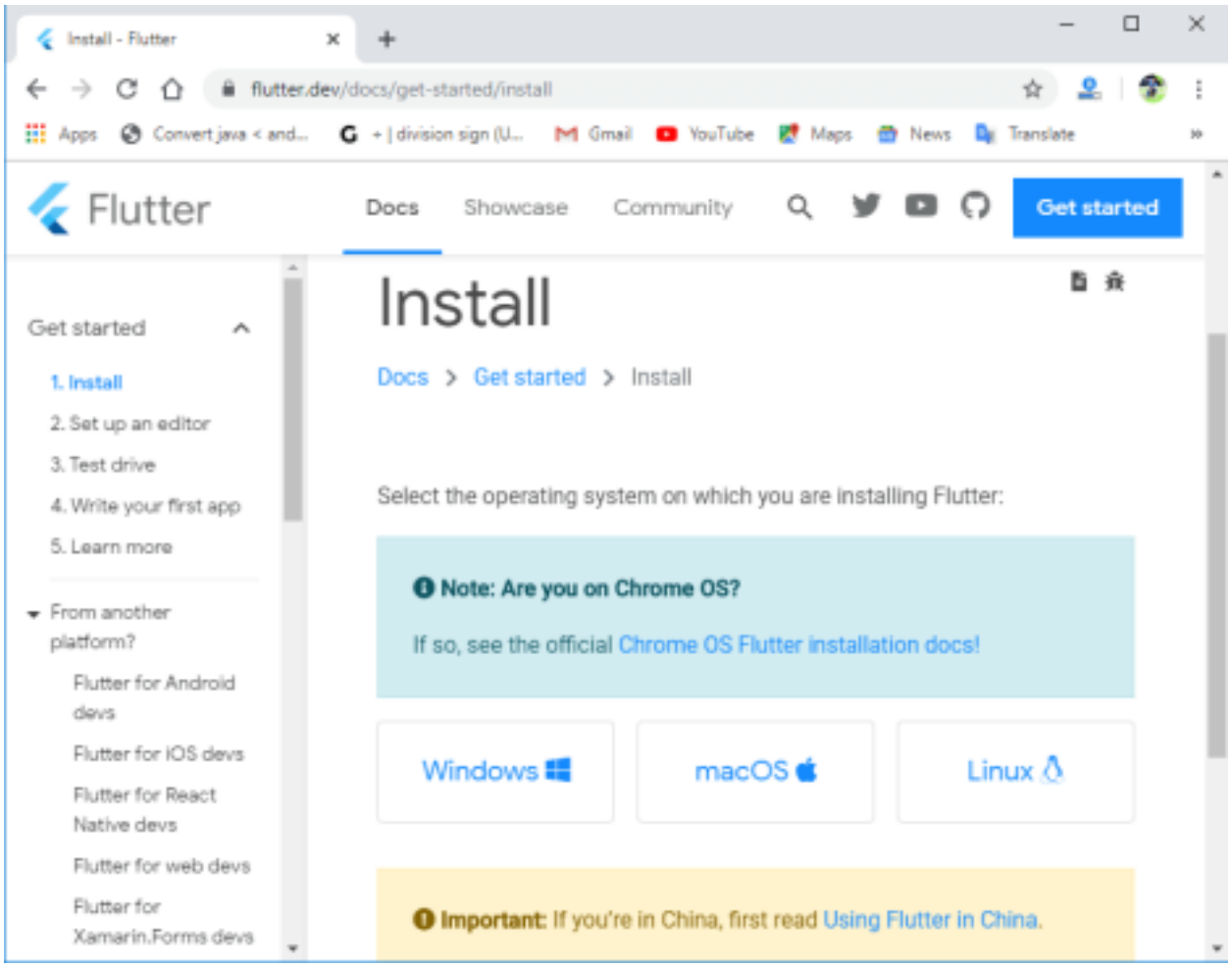
# MAD & PWA Lab
# Journal

| | |
|---|---|
| Experiment No. | 01 |
| Experiment Title. | To install and configure the Flutter Environment |
| Roll No. | 55 |
| Name | Mohd Sufiyan Siddiqui |
| Class | D15A |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO1: Understand cross platform mobile application development using Flutter framework |
| Grade: | 15 |

**Aim**: To install flutter and Android Studio and create a 'Hello World App' using Flutter

## 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.



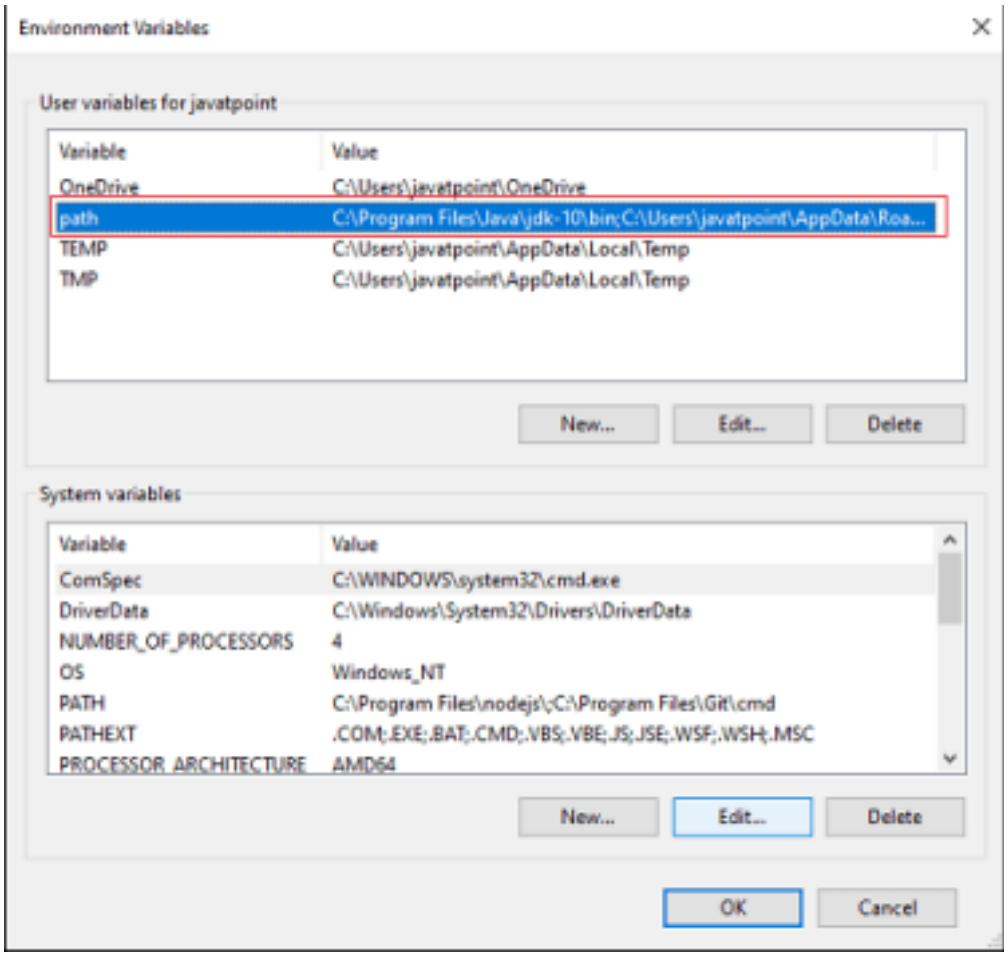

**Step 2:** Next, to download the latest Flutter SDK, click on the Windows **icon**. Here, you will find the download link for SDK.

**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.

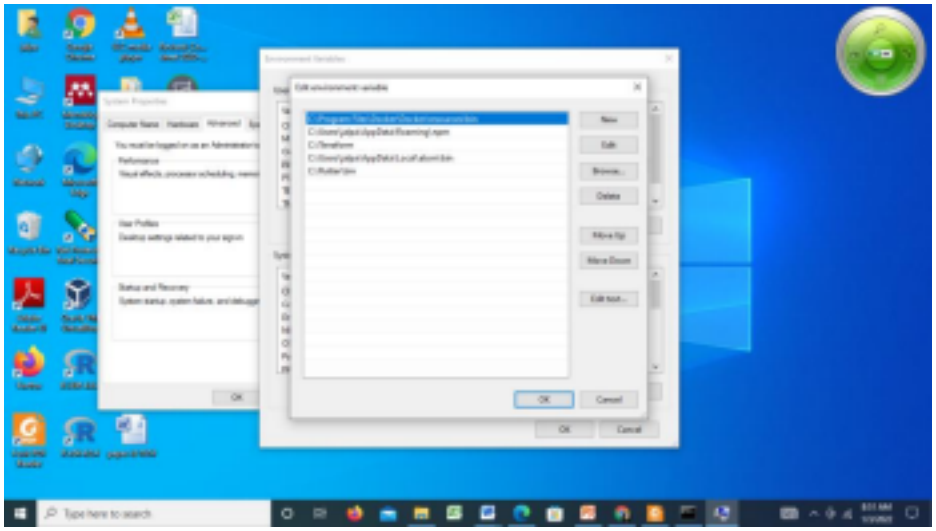
**Step 4:** To run the Flutter command in regular windows console, you need to update the

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.



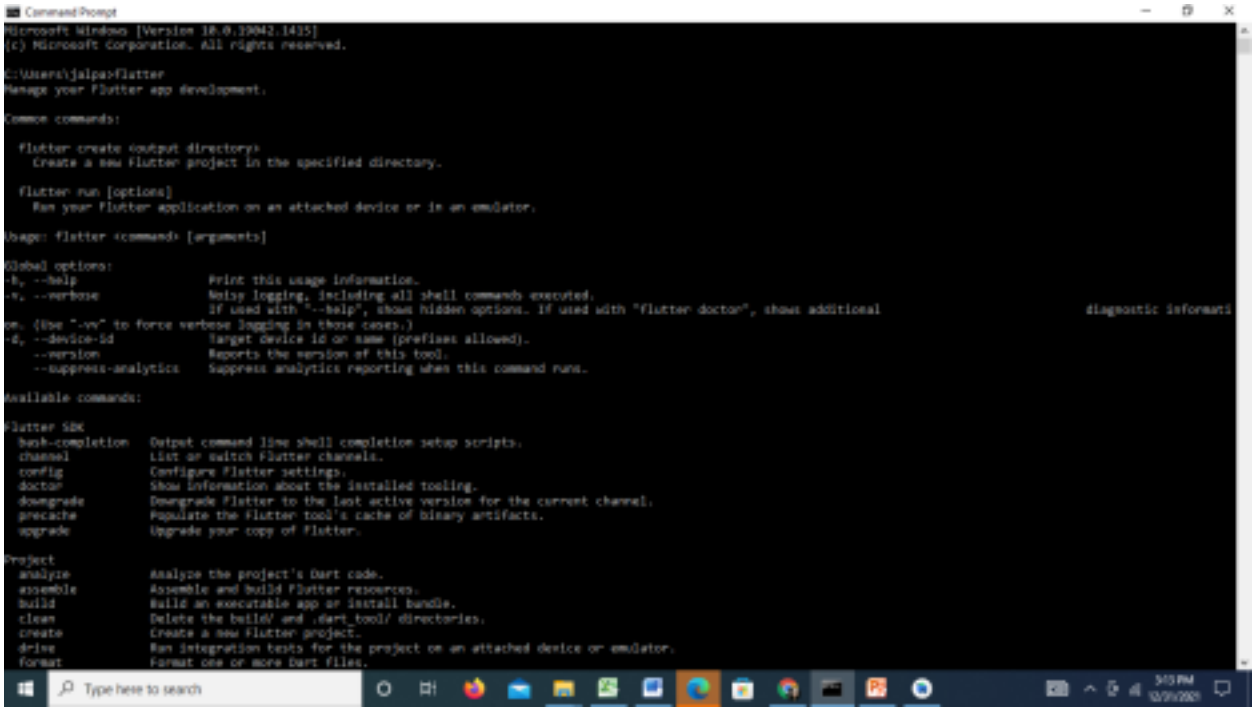

**Step 4.2:** Now, select path -> click on edit. The following screen appears



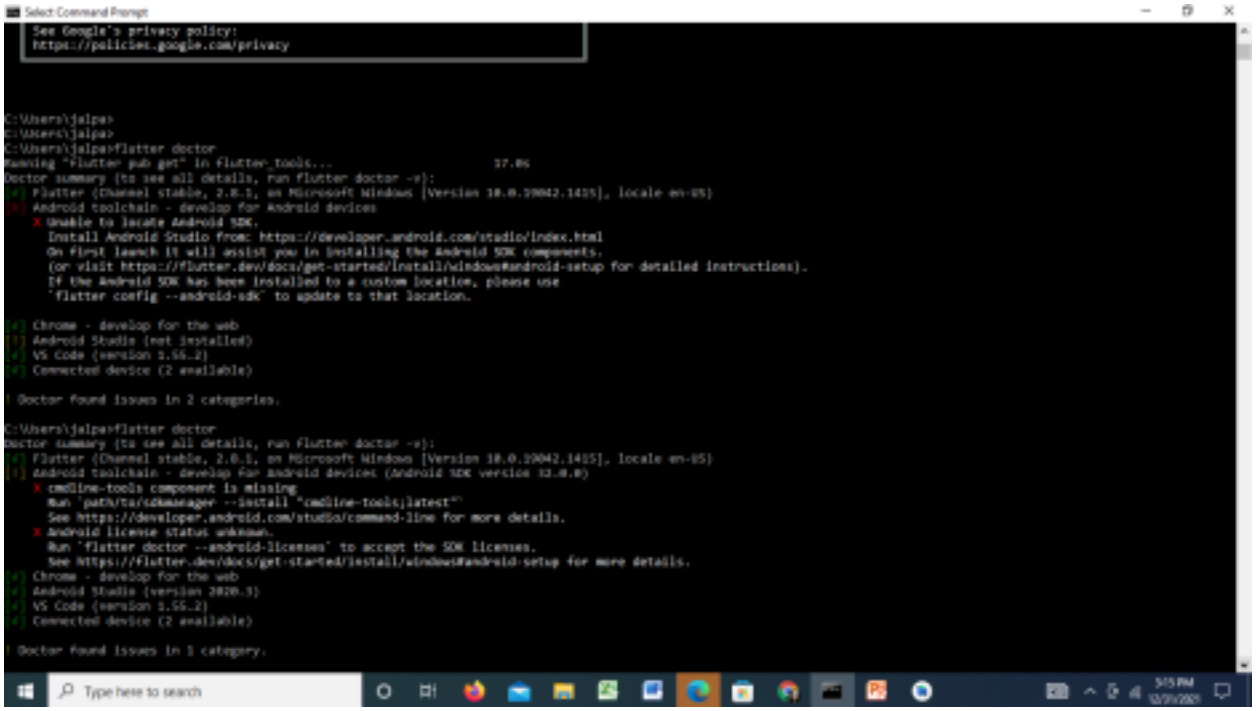
**Step 4.3:** In the above window, click on New->write path of Flutter bin folder in variable

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

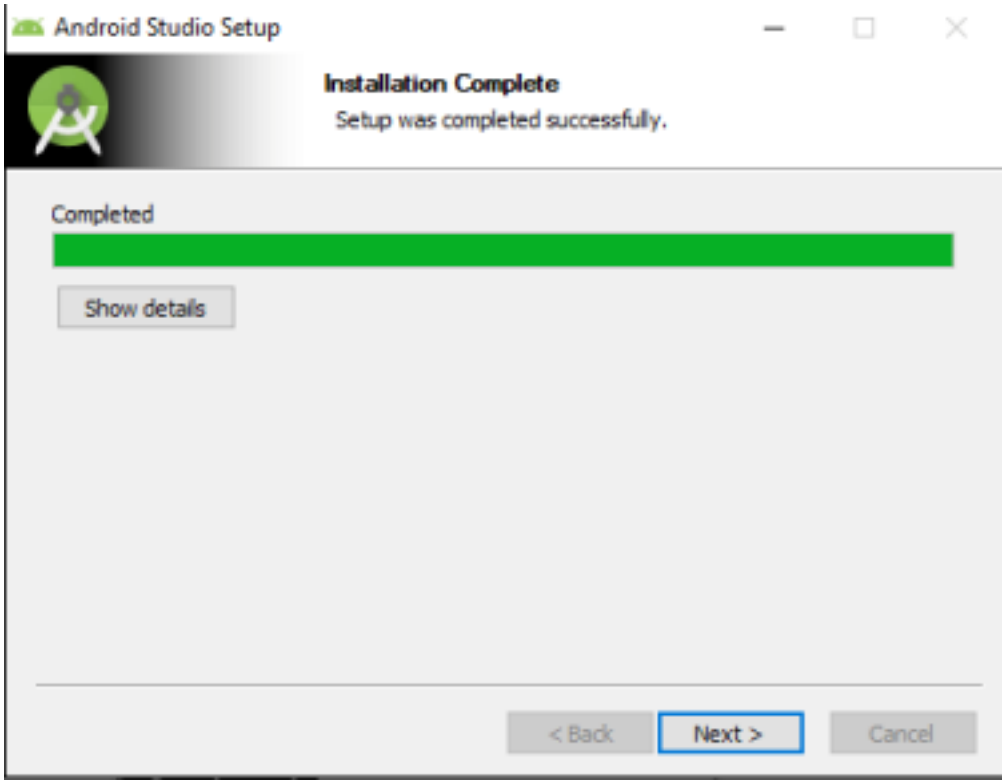
shown in the below image. Here, you will find the details of all missing tools, which required to run Flutter as well as the development tools that are available but not connected with the device.

**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.

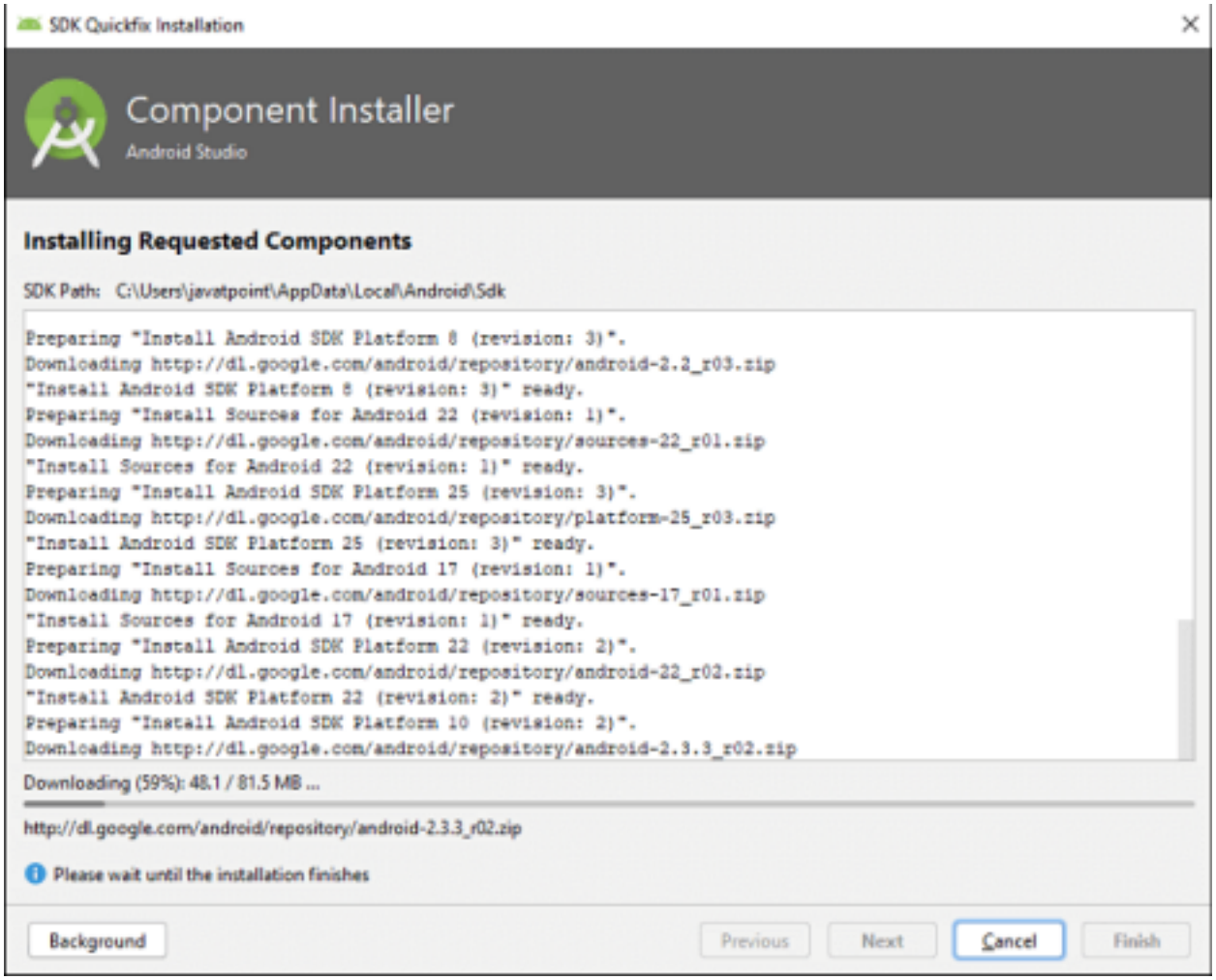
**Step 7.1:** Download the latest Android Studio executable or zip file from the official site.

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

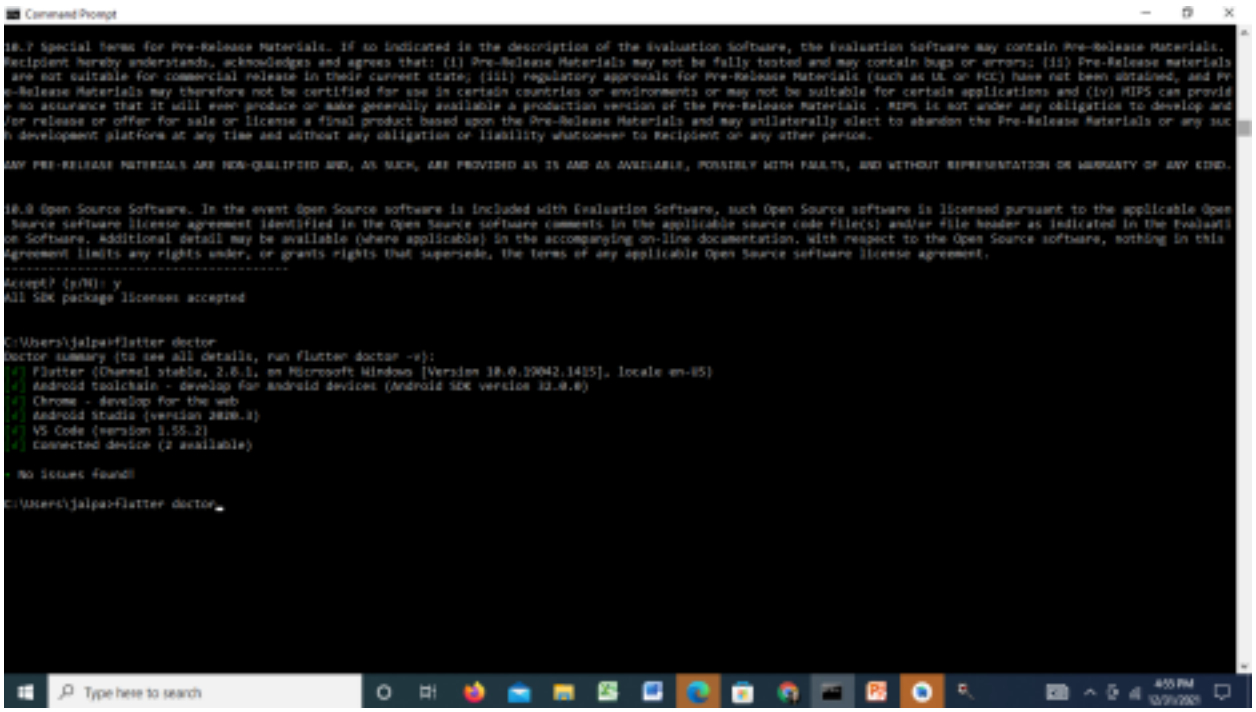

**Step 7.3:** Follow the steps of the installation wizard. Once the installation wizard completes, you will get the following screen.




**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.

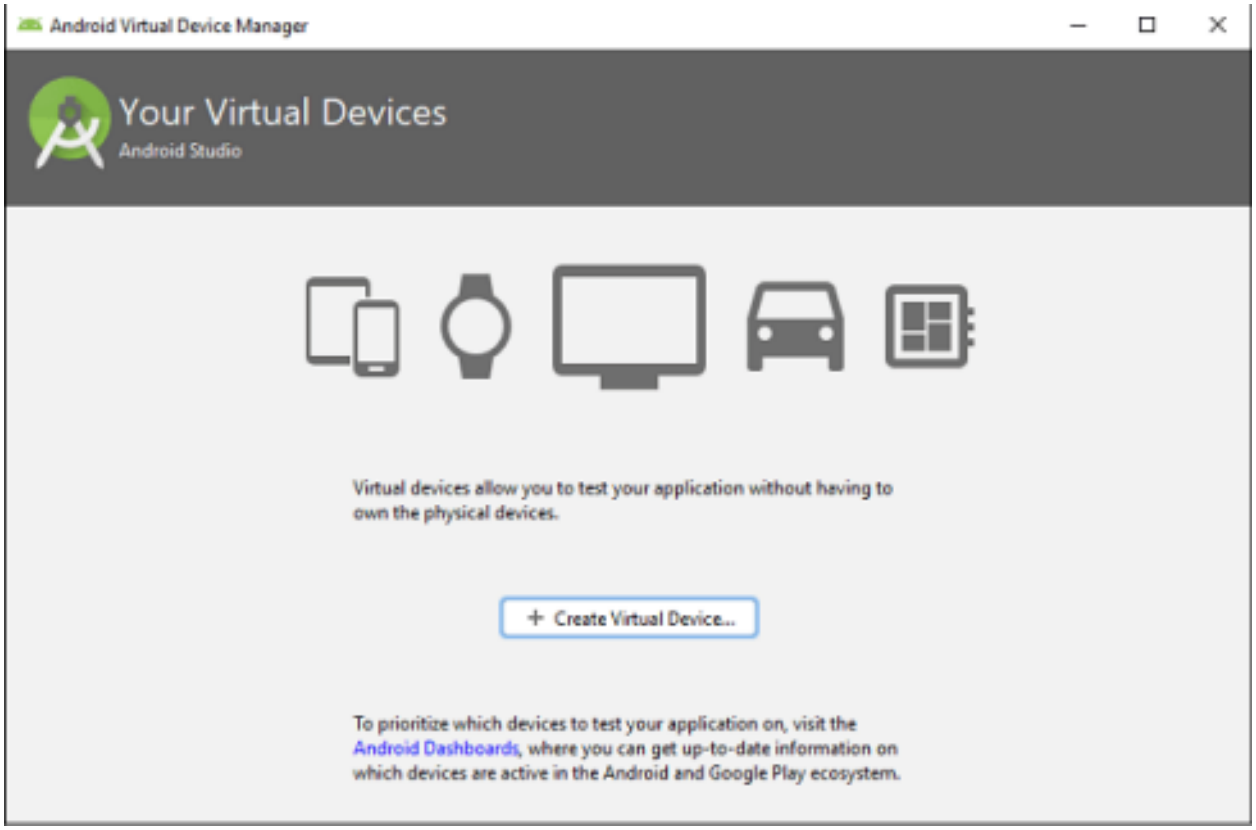



Step 7.5 run the $ **flutter doctor** command and Run flutter doctor --android-licenses command.




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

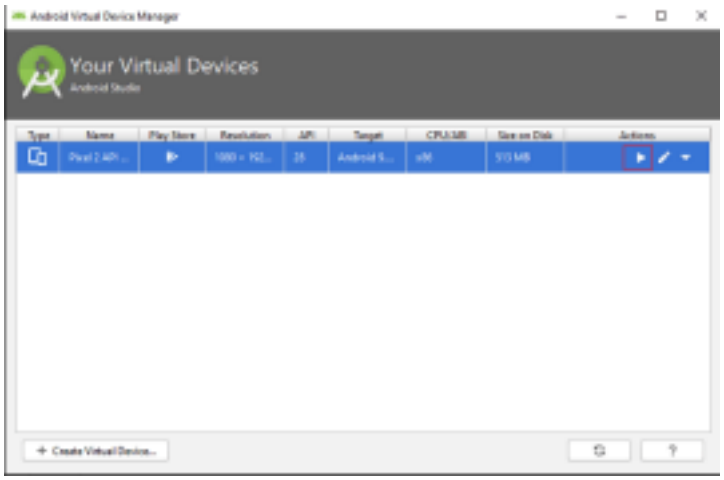
**Step 8.1:** To set an Android emulator, go to Android Studio > Tools > Android > AVD Manager and select Create Virtual Device. Or, go to Help->Find Action->Type Emulator in the search box. You will get the following screen.




**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.

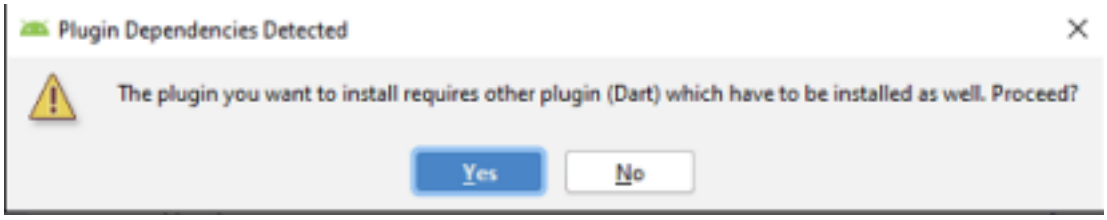



**Step 8.5:** Last, click on the icon pointed into the red color rectangle. The Android emulator displayed as below screen.

**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.

```
import 'package:flutter/material.dart';
void main() {
runApp(const MyApp());
}
class MyApp extends StatelessWidget {
const MyApp({Key? key}) : super(key: key);
@override
Widget build(BuildContext context) {
return MaterialApp(
title: 'Welcome to Flutter',
home: Scaffold(
appBar: AppBar(
title: const Text('Welcome to Flutter'),
),
body: const Center(
child: Text('Hello World'),
),
),
);
}
}
```

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


**Conclusion**: Thus,we installed flutter and android studio and created our first flutter application.

# MAD & PWA Lab
# Journal

| | |
|---|---|
| Experiment No. | 02 |
| Experiment Title. | To design Flutter UI by including common widgets. |
| Roll No. | 55 |
| Name | Mohd Sufiyan Siddiqui |
| Class | D15A |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation |
| Grade: | 15 |

**Project Title: Flutter:Swiggy Clone/PWA:Todo List** **Roll No. 55**

**Theory :**

Flutter: A Comprehensive Overview

Introduction:

Flutter is an open-source UI software development toolkit created by Google, known for its ability to build natively compiled applications for mobile, web, and desktop from a single codebase. Launched in 2017, Flutter has gained widespread popularity for its expressive and flexible framework, allowing developers to create visually appealing and performant applications.

Key Features:

1. Single Codebase, Multiple Platforms:
   Flutter allows developers to write code once and deploy it on various platforms, such as iOS, Android, web, and desktop. This not only saves development time but also ensures consistency across different devices.

2. Widget-based Architecture:
   Flutter's core building blocks are widgets, which are lightweight and reusable elements used to construct the user interface. Widgets can be combined to create complex UIs, and Flutter provides an extensive set of both Material Design and Cupertino widgets for a native look and feel on Android and iOS.

3. Hot Reload:
   One of Flutter's standout features is the Hot Reload functionality. Developers can see the impact of code changes in real-time without restarting the entire application. This speeds up the development process, making it more interactive and efficient.

4. Rich Set of Libraries:
   Flutter comes with a rich set of pre-designed material components and Cupertino widgets. Additionally, developers can use packages from the pub.dev repository to extend the functionality of their applications, covering a wide range of features such as authentication, databases, and more.

5. Dart Programming Language:
   Flutter uses Dart as its programming language. Dart is a modern, object-oriented language that provides features like a strong static type system and just-in-time (JIT) compilation. Its syntax is easy to learn for developers familiar with languages like Java or JavaScript.

Architecture:

Flutter follows a layered architecture comprising the following components:

1. Framework Layer:
   The framework layer contains the foundation of Flutter, providing the basic building blocks such as widgets, rendering, and animation. The framework is responsible for managing the flow of data and handling user input.

2. Engine Layer:
   The engine layer consists of C/C++ code responsible for rendering, gestures, and other low-level tasks. It interacts with the platform-specific code and communicates with the framework layer using a platform channel.

3. Embedder Layer:
   The embedder layer integrates Flutter into the respective platform (iOS, Android, web, desktop). It handles the initialization of Flutter, communication with the engine layer, and exposes platform-specific services to the framework.

Advantages:

1. Productivity:
   The ability to write code once and deploy it on multiple platforms significantly enhances developer productivity. Hot Reload ensures quick iterations and reduces development time.

2. Expressive UI:
   Flutter's widget-based architecture allows for the creation of highly customizable and visually appealing user interfaces. Designers and developers can collaborate seamlessly to bring designs to life.

3. Performance:
   Flutter's compiled code performs at native speed, providing a smooth and responsive user experience. The absence of a bridge between the application and the platform contributes to excellent performance.

4. Community and Ecosystem:
   Flutter has a vibrant and growing community, contributing to a vast ecosystem of plugins and packages. This makes it easy for developers to find solutions to common problems and integrate various functionalities into their applications.

Challenges and Considerations:

1. Learning Curve:
   While Dart is easy to pick up for developers with existing programming experience, there might be a learning curve for those unfamiliar with the language.

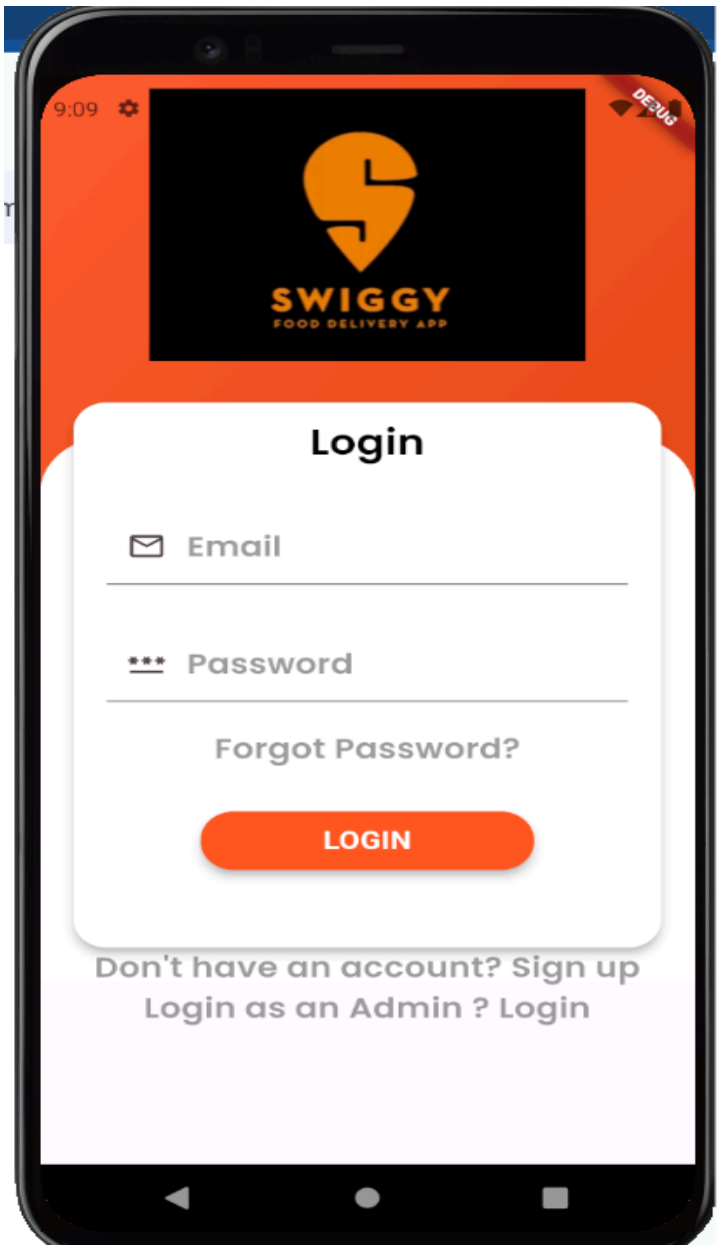
2. Size of the App:

Flutter apps may have a larger file size compared to native apps due to the inclusion of the Flutter engine. However, advancements are being made to address this concern.

3. Platform-specific Integration:

While Flutter provides a unified codebase, there might be cases where platform-specific features or integrations require additional effort.

**OUTPUT:**



**Code :**

```
import 'package:flutter/material.dart';
import 'package:animate_do/animate_do.dart';
void main() => runApp(
 MaterialApp(
   debugShowCheckedModeBanner: false,
   home: HomePage(),
 )
);

class HomePage extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return Scaffold(
     backgroundColor: Colors.white,
     body: SingleChildScrollView(
       child: Container(
         child: Column(
           children: <Widget>[
             Container(
               height: 400,
               decoration: BoxDecoration(
                 image: DecorationImage(
                   image: AssetImage('assets/images/b6.jpg'),
                   fit: BoxFit.fill
                 )
               ),
               child: Stack(
                 children: <Widget>[
                   Positioned(
                     left: 30,
                     width: 80,
                     height: 200,
                     child: FadeInUp(duration: Duration(seconds: 1), child: Container(
                       decoration: BoxDecoration(
                         image: DecorationImage(
                           image: AssetImage('assets/images/light-1.png')
                         )
                       ),
                     )),
                   ),
                   Positioned(
                     left: 140,
                     width: 80,
```

```
                height: 150,
                child: FadeInUp(duration: Duration(milliseconds: 1200), child: Container(
                  decoration: BoxDecoration(
                    image: DecorationImage(
                      image: AssetImage('assets/images/light-2.png')
                    )
                  ),
                )),
              ),
              Positioned(
                right: 40,
                top: 40,
                width: 80,
                height: 150,
                child: FadeInUp(duration: Duration(milliseconds: 1300), child: Container(
                  decoration: BoxDecoration(
                    image: DecorationImage(
                      image: AssetImage('assets/images/clock.png')
                    )
                  ),
                )),
              ),
              Positioned(
                child: FadeInUp(duration: Duration(milliseconds: 1600), child: Container(
                  margin: EdgeInsets.only(top: 50),
                  child: Center(
                    child: Text("Login", style: TextStyle(color: Colors.white, fontSize: 40, fontWeight:
FontWeight.bold),),
                  ),
                )),
              )
            ],
          ),
        ),
        Padding(
          padding: EdgeInsets.all(30.0),
          child: Column(
            children: <Widget>[
              FadeInUp(duration: Duration(milliseconds: 1800), child: Container(
                padding: EdgeInsets.all(5),
                decoration: BoxDecoration(
                  color: Colors.white,
                  borderRadius: BorderRadius.circular(10),
                  border: Border.all(color: Color.fromRGBO(143, 148, 251, 1)),
```

```
                boxShadow: [
                 BoxShadow(
                  color: Color.fromRGBO(143, 148, 251, .2),
                  blurRadius: 20.0,
                  offset: Offset(0, 10)
                 )
                ]
               ),
               child: Column(
                children: <Widget>[
                 Container(
                  padding: EdgeInsets.all(8.0),
                  decoration: BoxDecoration(
                   border: Border(bottom: BorderSide(color:  Color.fromRGBO(143, 148, 251,
1)))
                  ),
                  child: TextField(
                   decoration: InputDecoration(
                    border: InputBorder.none,
                    hintText: "Email or Phone number",
                    hintStyle: TextStyle(color: Colors.grey[700])
                   ),
                  ),
                 ),
                 Container(
                  padding: EdgeInsets.all(8.0),
                  child: TextField(
                   obscureText: true,
                   decoration: InputDecoration(
                    border: InputBorder.none,
                    hintText: "Password",
                    hintStyle: TextStyle(color: Colors.grey[700])
                   ),
                  ),
                 )
                ],
               ),
              )),
              SizedBox(height: 30,),
              FadeInUp(duration: Duration(milliseconds: 1900), child: Container(
               height: 50,
               decoration: BoxDecoration(
                borderRadius: BorderRadius.circular(10),
                gradient: LinearGradient(
```

```
                    colors: [
                      Color.fromRGBO(143, 148, 251, 1),
                      Color.fromRGBO(143, 148, 251, .6),
                    ]
                  )
                ),
                child: Center(
                  child: Text("Login", style: TextStyle(color: Colors.white, fontWeight:
FontWeight.bold),),
                ),
              )),
              SizedBox(height: 70,),
              FadeInUp(duration: Duration(milliseconds: 2000), child: Text("Forgot Password?",
style: TextStyle(color: Color.fromRGBO(143, 148, 251, 1)),)),
            ],
          ),
        )
      ],
    ),
  ),
 )
);
}
}
```

**Conclusion:**

Flutter has emerged as a powerful framework for cross-platform development, providing a seamless development experience, expressive UI, and excellent performance. Its growing community and continuous updates from Google contribute to its ongoing success. As the technology landscape evolves, Flutter is likely to play an increasingly significant role in shaping the future of multi-platform application development.

# MAD & PWA Lab
# Journal

| | |
|---|---|
| Experiment No. | 03 |
| Experiment Title. | To include icons, images, fonts in Flutter app |
| Roll No. | 55 |
| Name | Mohd Sufiyan Siddiqui |
| Class | D15A |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation |
| Grade: | 15 |

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

**Theory:**

Flutter is a UI toolkit developed by Google for building natively compiled applications for mobile, web, and desktop from a single codebase. YAML (YAML Ain't Markup Language) is a human-readable data serialization format often used for configuration files in Flutter projects.

Certainly! Flutter is a UI toolkit developed by Google for building natively compiled applications for mobile, web, and desktop from a single codebase. YAML (YAML Ain't Markup Language) is a human-readable data serialization format often used for configuration files in Flutter projects.

Below, I'll provide a basic example of a Flutter app that includes YAML configuration for image insertion. Let's create a simple Flutter app that displays an image using YAML for configuration.

We can split the Flutter widget into two categories:

Visible (Output and Input)

Invisible (Layout and Control)
Visible widget
The visible widgets are related to the user input and output data. Some of the important types of this widget are:
1. Text
A Text widget holds some text to display on the screen. We can align the text widget by using textAlign property, and style property allow the customization of Text that includes font, font weight, font style, letter spacing, color, and many more. We can use it as like below code snippets.

```
new Text(
'Hello, ALL!',
textAlign: TextAlign.center,
style: new TextStyle(fontWeight: FontWeight.bold),
)
```

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

```
//FlatButton Example
new FlatButton(
child: Text("Click here"),
onPressed: () {
// Do something here

},
),

//RaisedButton Example
new RaisedButton(
child: Text("Click here"),
elevation: 5.0,
onPressed: () {
// Do something here
},
```

3. Image
This widget holds the image which can fetch it from multiple sources like from the asset folder or directly from the URL. It provides many constructors for loading image, which are given below:
o Image: It is a generic image loader, which is used by ImageProvider.
o asset: It load image from your project asset folder.
o file: It loads images from the system folder.
o memory: It load image from memory.
o network: It loads images from the network.

**Project Title: Flutter:Swiggy Clone/PWA:Todo List** **Roll No. 55**

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

**Code:-**

```
import 'package:flutter/material.dart';
import 'package:flutter_application_1/widget/widget_support.dart';

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

  @override
  State<Home> createState() => _HomeState();
}

class _HomeState extends State<Home> {
  bool icecream = false, pizza = false, salad = false, burger = false;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Container(
        margin: EdgeInsets.only(top: 50.0, left: 20.0),
        child: Column(crossAxisAlignment: CrossAxisAlignment.start, children: [
          Row(
            mainAxisAlignment: MainAxisAlignment.spaceBetween,
            children: [
              Text("Hello Sufiyan,", style: AppWidget.boldTextFeildStyle()),
              Container(
                padding: EdgeInsets.all(3),
                decoration: BoxDecoration(color: Colors.black),
                child: Icon(
                  Icons.shopping_cart_outlined,
                  color: Colors.white,
                ),
              )
            ],
          ),
          SizedBox(height: 30.0),
          Text("Delicious Food", style: AppWidget.HeadlineTextFeildStyle()),
          Text(
```

```
  "Discover and get great food",
  style: AppWidget.LightTextFeildStyle(),
 ),
 SizedBox(
  height: 20.0,
 ),
 showItem(),
 SizedBox(height: 30.0),
 SingleChildScrollView(
    scrollDirection: Axis.horizontal,
    child: Row(
     // crossAxisAlignment: CrossAxisAlignment.start,
     children: [
      Container(
       margin: EdgeInsets.all(10),
       child: Material(
          elevation: 5.0,
          borderRadius: BorderRadius.circular(10),
          child: Container(
           padding: EdgeInsets.all(10),
           child: Column(children: [
            Image.asset(
             "images/salad2.png",
             height: 160,
             width: 160,
             fit: BoxFit.cover,
            ),
            Text("Veggie Taco Hash",
               style: AppWidget.boldTextFeildStyle()),
            SizedBox(
             height: 5.0,
            ),
            Text("Healthy and Fresh",
               style: AppWidget.LightTextFeildStyle()),
            SizedBox(
             height: 5.0,
            ),
            Text("\$30", style: AppWidget.boldTextFeildStyle()),
           ]),
          )),
      ),
      SizedBox(width: 10.0),
      Container(
       margin: EdgeInsets.all(10),
```

```
  child: Material(
      elevation: 5.0,
      borderRadius: BorderRadius.circular(10),
      child: Container(
        padding: EdgeInsets.all(10),
        child: Column(children: [
          Image.asset(
            "images/salad2.png",
            height: 160,
            width: 160,
            fit: BoxFit.cover,
          ),
          Text("Veggie Taco Hash",
              style: AppWidget.boldTextFeildStyle()),
          SizedBox(
            height: 5.0,
          ),
          Text("Healthy and Fresh",
              style: AppWidget.LightTextFeildStyle()),
          SizedBox(
            height: 5.0,
          ),
          Text("\$30", style: AppWidget.boldTextFeildStyle()),
        ]),
      )),
),
SizedBox(width: 10),
Container(
  margin: EdgeInsets.all(10),
  child: Material(
      elevation: 5.0,
      borderRadius: BorderRadius.circular(10),
      child: Container(
        padding: EdgeInsets.all(10),
        child: Column(children: [
          Image.asset(
            "images/salad2.png",
            height: 160,
            width: 160,
            fit: BoxFit.cover,
          ),
          Text("Veggie Taco Hash",
              style: AppWidget.boldTextFeildStyle()),
          SizedBox(
```

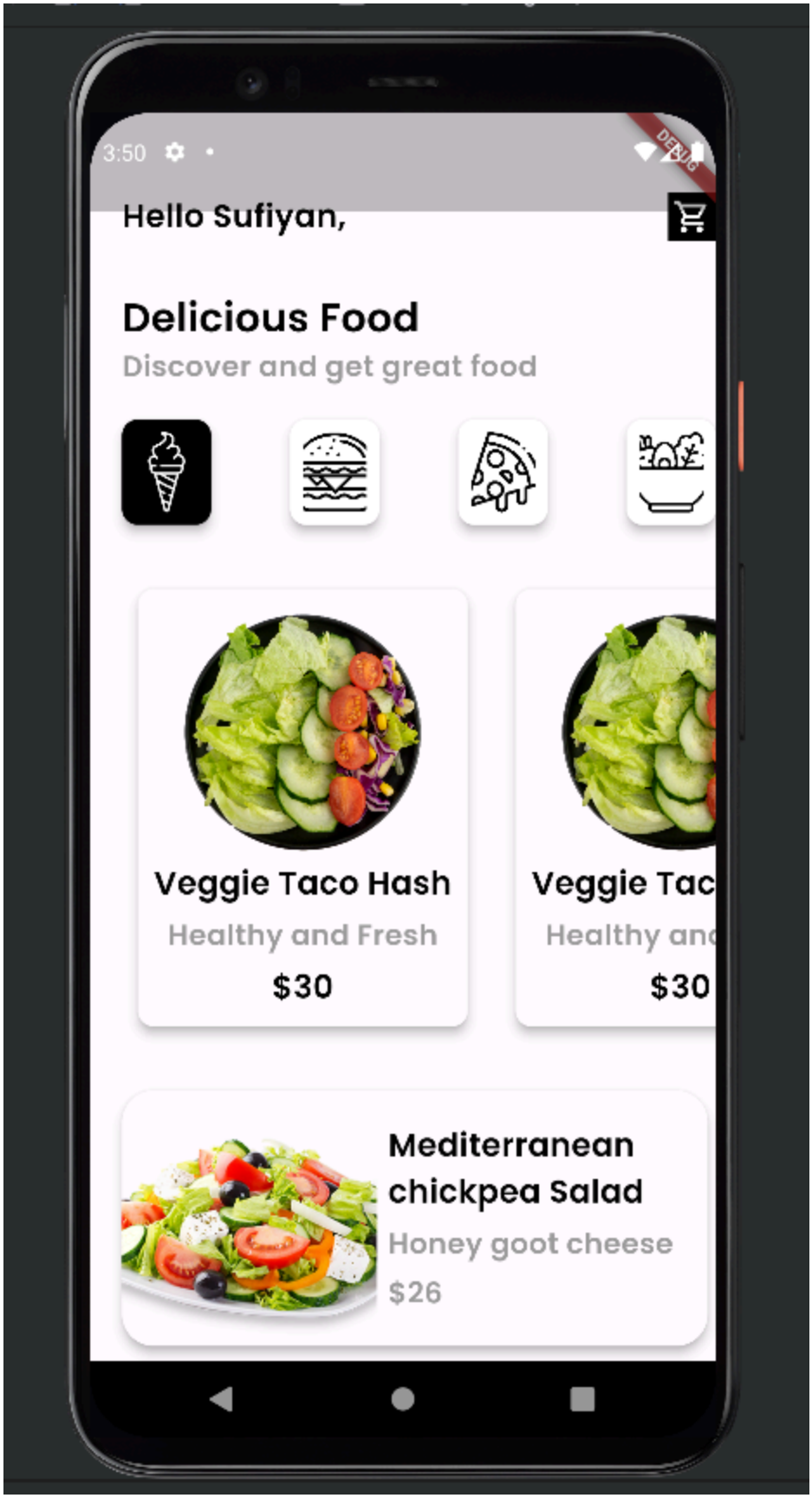
```
                  height: 5.0,
                ),
                Text("Healthy and Fresh",
                    style: AppWidget.LightTextFeildStyle()),
                SizedBox(
                  height: 5.0,
                ),
                Text("\$30", style: AppWidget.boldTextFeildStyle()),
              ]),
            )),
      )
    ],
  )),
SizedBox(
  height: 30.0,
),
Container(
    margin: EdgeInsets.only(right: 5.0),
    child: Material(
      elevation: 5.0,
      borderRadius: BorderRadius.circular(20),
      child: Row(children: [
        Image.asset("images/salad3.png",
            height: 160, width: 160, fit: BoxFit.cover),
        SizedBox(width: 7.0),
        Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Container(
                width: MediaQuery.of(context).size.width / 2,
                child: Text(
                  "Mediterranean chickpea Salad",
                  style: AppWidget.boldTextFeildStyle(),
                )),
            SizedBox(
              height: 5.0,
            ),
            Container(
                width: MediaQuery.of(context).size.width / 2,
                child: Text(
                  "Honey goot cheese",
                  style: AppWidget.LightTextFeildStyle(),
                )),
            SizedBox(
```

```
                  height: 5.0,
                ),
                Container(
                    width: MediaQuery.of(context).size.width / 2,
                    child: Text(
                      "\$26",
                      style: AppWidget.LightTextFeildStyle(),
                    ))
              ],
            )
          ]),
        ))
    ]),
  ),
 );
}

Widget showItem() {
 return Row(
  mainAxisAlignment: MainAxisAlignment.spaceBetween,
  children: [
   GestureDetector(
      onTap: () {
       icecream = true;
       pizza = false;
       salad = false;
       burger = false;
       setState(() {});
      },
      child: Material(
         elevation: 5.0,
         borderRadius: BorderRadius.circular(10),
         child: Container(
            decoration: BoxDecoration(
               color: icecream ? Colors.black : Colors.white,
               borderRadius: BorderRadius.circular(10)),
            padding: EdgeInsets.all(8),
            child: Image.asset("images/ice-cream.png",
               height: 50,
               width: 40,
               fit: BoxFit.cover,
               color: icecream ? Colors.white : Colors.black)))),
   GestureDetector(
      onTap: () {
```

```
        icecream = false;
        pizza = false;
        salad = false;
        burger = true;
        setState(() {});
      },
      child: Material(
          elevation: 5.0,
          borderRadius: BorderRadius.circular(10),
          child: Container(
              decoration: BoxDecoration(
                  color: burger ? Colors.black : Colors.white,
                  borderRadius: BorderRadius.circular(10)),
              padding: EdgeInsets.all(8),
              child: Image.asset("images/burger.png",
                  height: 50,
                  width: 40,
                  fit: BoxFit.cover,
                  color: burger ? Colors.white : Colors.black)))),
    GestureDetector(
      onTap: () {
        icecream = false;
        pizza = true;
        salad = false;
        burger = false;
        setState(() {});
      },
      child: Material(
          elevation: 5.0,
          borderRadius: BorderRadius.circular(10),
          child: Container(
              decoration: BoxDecoration(
                  color: pizza ? Colors.black : Colors.white,
                  borderRadius: BorderRadius.circular(10)),
              padding: EdgeInsets.all(8),
              child: Image.asset("images/pizza.png",
                  height: 50,
                  width: 40,
                  fit: BoxFit.cover,
                  color: pizza ? Colors.white : Colors.black)))),
    GestureDetector(
      onTap: () {
        icecream = false;
        pizza = false;
```

```
            salad = true;
            burger = false;
            setState(() {});
          },
          child: Material(
              elevation: 5.0,
              borderRadius: BorderRadius.circular(10),
              child: Container(
                  decoration: BoxDecoration(
                      color: salad ? Colors.black : Colors.white,
                      borderRadius: BorderRadius.circular(10)),
                  padding: EdgeInsets.all(8),
                  child: Image.asset("images/salad.png",
                      height: 50,
                      width: 40,
                      fit: BoxFit.cover,
                      color: salad ? Colors.white : Colors.black)))),
      ],
    );
  }
}
```

**Output:-**


3:50
DEBUG
Hello Sufiyan,
Delicious Food
Discover and get great food
Veggie Taco Hash
Healthy and Fresh
$30
Veggie Tac
Healthy and
$30
Mediterranean chickpea Salad
Honey goot cheese
$26

**Conclusion:**

Flutter has emerged as a powerful framework for cross-platform development, providing a seamless development experience, expressive UI, and excellent performance. Its growing community and continuous updates from Google contribute to its ongoing success. As the technology landscape evolves, Flutter is likely to play an increasingly significant role in shaping the future of multi-platform application development.

# MAD & PWA Lab
# Journal

| | |
|---|---|
| Experiment No. | 04 |
| Experiment Title. | To create an interactive Form using form widget |
| Roll No. | 55 |
| Name | Mohd Sufiyan Siddiqui |
| Class | D15A |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation |
| Grade: | 15 |

## Aim: To create an interactive Form using form widget

## Theory:

Creating an interactive form using the form widget in Flutter involves several steps. Here's a theoretical outline to guide you through the process:

1. Import Required Libraries : Begin by importing the necessary libraries in your Flutter project. Typically, you'll need the `flutter/material.dart` library for UI components and `flutter/widgets.dart` for widgets.

2. Design UI Layout : Plan the layout of your form UI. Decide on the fields you want to include (text fields, checkboxes, dropdowns, etc.) and how you want them arranged on the screen.

3. Implement Form Widget : Use the `Form` widget to wrap your form fields. The `Form` widget automatically handles form validation and submission.

4. Add Form Fields : Inside the `Form` widget, add form fields using widgets like `TextFormField`, `Checkbox`, `DropdownButton`, etc., depending on your requirements.

5. Validation : Implement validation for the form fields to ensure that users enter data correctly. You can use the `validator` property of `TextFormField` or custom validation logic.

6. Handle Form Submission : Implement logic to handle form submission. This typically involves attaching a callback function to the form's `onSubmitted` or `onSaved` property.

7. Display Validation Errors : If form validation fails, display error messages to guide users on what needs to be corrected. You can use the `errorText` property of form fields for this purpose.

8. Testing : Thoroughly test your interactive form to ensure that it behaves as expected. Test different scenarios, such as valid and invalid input, to verify that your validation logic works correctly.

9. Refinement and Optimization : Refine your form based on user feedback and optimize its performance if necessary. Consider factors like usability, accessibility, and responsiveness.

10. Documentation and Maintenance : Document your code to make it easier for others (and yourself) to understand. Also, be prepared to maintain and update your form as needed, especially if the underlying Flutter framework evolves.

Remember that this is a high-level overview, and the actual implementation details may vary depending on your specific requirements and preferences. Additionally, you can refer to the Flutter documentation and community resources for more detailed guidance and examples.

**Code:**

```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:flutter_application_1/pages/bottomnav.dart';
import 'package:flutter_application_1/pages/forgotpassword.dart';
import 'package:flutter_application_1/pages/home.dart';
import 'package:flutter_application_1/pages/signup.dart';
import 'package:flutter_application_1/widget/widget_support.dart';

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

  @override
  State<LogIn> createState() => _LogInState();
}

class _LogInState extends State<LogIn> {
  String email = "", password = "";

  final _formkey = GlobalKey<FormState>();

  TextEditingController useremailcontroller = new TextEditingController();
  TextEditingController userpasswordcontroller = new TextEditingController();

  userLogin() async {
    try {
      await FirebaseAuth.instance
          .signInWithEmailAndPassword(email: email, password: password);
      Navigator.push(
          context, MaterialPageRoute(builder: (context) => BottomNav()));
    } on FirebaseAuthException catch (e) {
      if (e.code == 'user-not-found') {
        ScaffoldMessenger.of(context).showSnackBar(SnackBar(
            content: Text(
          "No User Found for that Email",
          style: TextStyle(fontSize: 18.0, color: Colors.black),
        )));
      } else if (e.code == 'wrong-password') {
        ScaffoldMessenger.of(context).showSnackBar(SnackBar(
            content: Text(
          "Wrong Password Provided by User",
          style: TextStyle(fontSize: 18.0, color: Colors.black),
        )));
      }
```

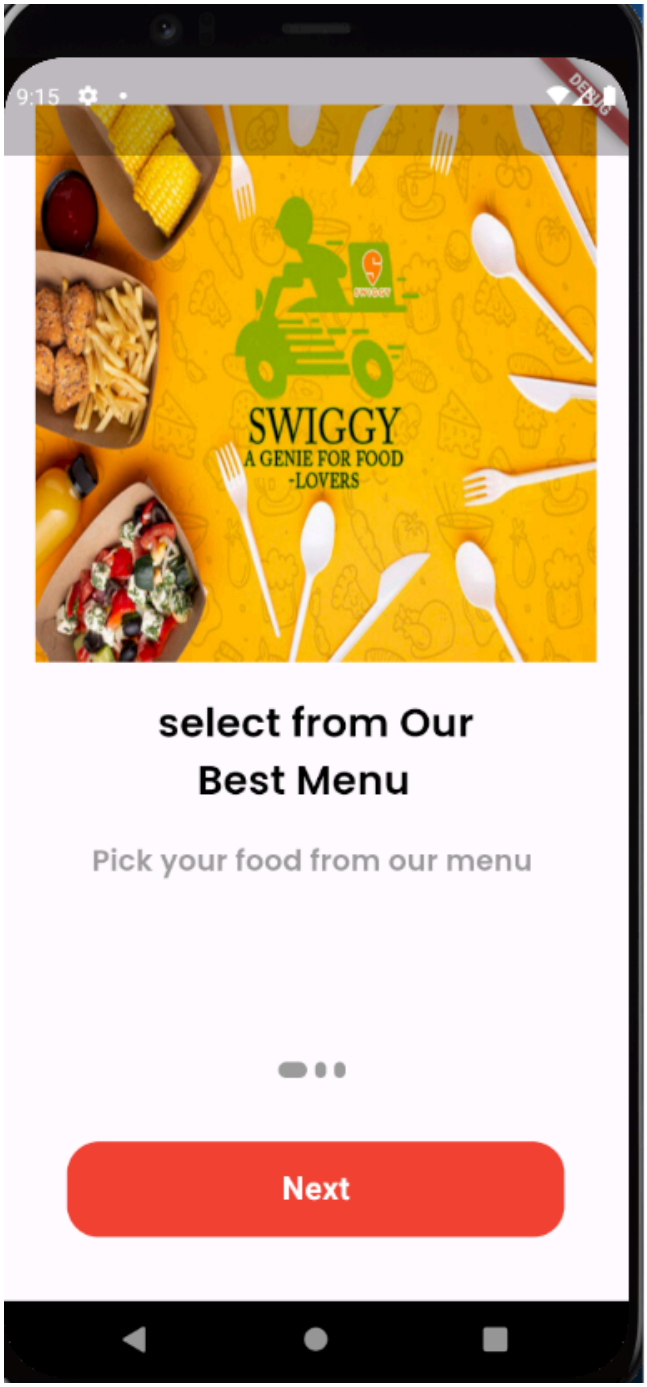
```
  }
 }

 @override
 Widget build(BuildContext context) {
  return Scaffold(
    body: Container(
      child: Stack(
        children: [
          Container(
            width: MediaQuery.of(context).size.width,
            height: MediaQuery.of(context).size.height / 2.5,
            decoration: BoxDecoration(
                gradient: LinearGradient(
                    begin: Alignment.topLeft,
                    end: Alignment.bottomRight,
                    colors: [
                  Color(0xFFff5c30),
                  Color(0xFFe74b1a),
                ])),
          ),
          Container(
            margin:
                EdgeInsets.only(top: MediaQuery.of(context).size.height / 3),
            height: MediaQuery.of(context).size.height / 2,
            width: MediaQuery.of(context).size.width,
            decoration: BoxDecoration(
                color: Colors.white,
                borderRadius: BorderRadius.only(
                    topLeft: Radius.circular(40),
                    topRight: Radius.circular(40))),
            child: Text(""),
          ),
          Container(
            margin: EdgeInsets.only(top: 10.0, left: 20.0, right: 20.0),
            child: Column(
              children: [
                Center(
                    child: Image.asset(
                  "images/logo.png",
                  width: MediaQuery.of(context).size.width / 1.5,
                  fit: BoxFit.cover,
                )),
                SizedBox(
```

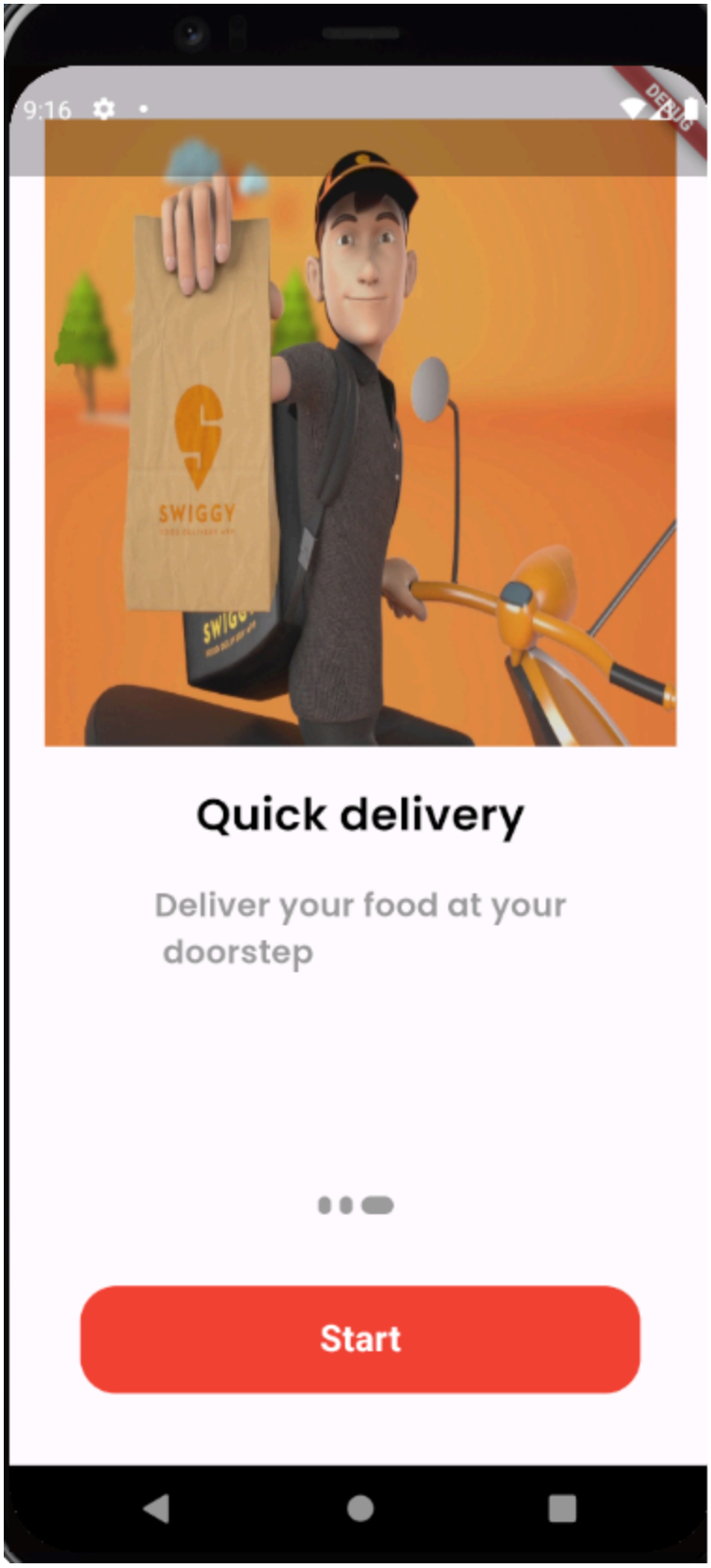
```
  height: 30.0,
),
Material(
  elevation: 5.0,
  borderRadius: BorderRadius.circular(20),
  child: Container(
    padding: EdgeInsets.only(left: 20.0, right: 20.0),
    width: MediaQuery.of(context).size.width,
    height: MediaQuery.of(context).size.height / 2,
    decoration: BoxDecoration(
        color: Colors.white,
        borderRadius: BorderRadius.circular(20)),
    child: Form(
      key: _formkey,
      child: Column(
        children: [
          SizedBox(
            height: 10.0,
          ),
          Text(
            "Login",
            style: AppWidget.HeadlineTextFeildStyle(),
          ),
          SizedBox(
            height: 30.0,
          ),
          TextFormField(
            controller: useremailcontroller,
            validator: (value) {
              if (value == null || value.isEmpty) {
                return 'Please Enter Email';
              }
              return null;
            },
            decoration: InputDecoration(
                hintText: 'Email',
                hintStyle: AppWidget.semiBoldTextFeildStyle(),
                prefixIcon: Icon(Icons.email_outlined)),
          ),
          SizedBox(
            height: 30.0,
          ),
          TextFormField(
            controller: userpasswordcontroller,
```

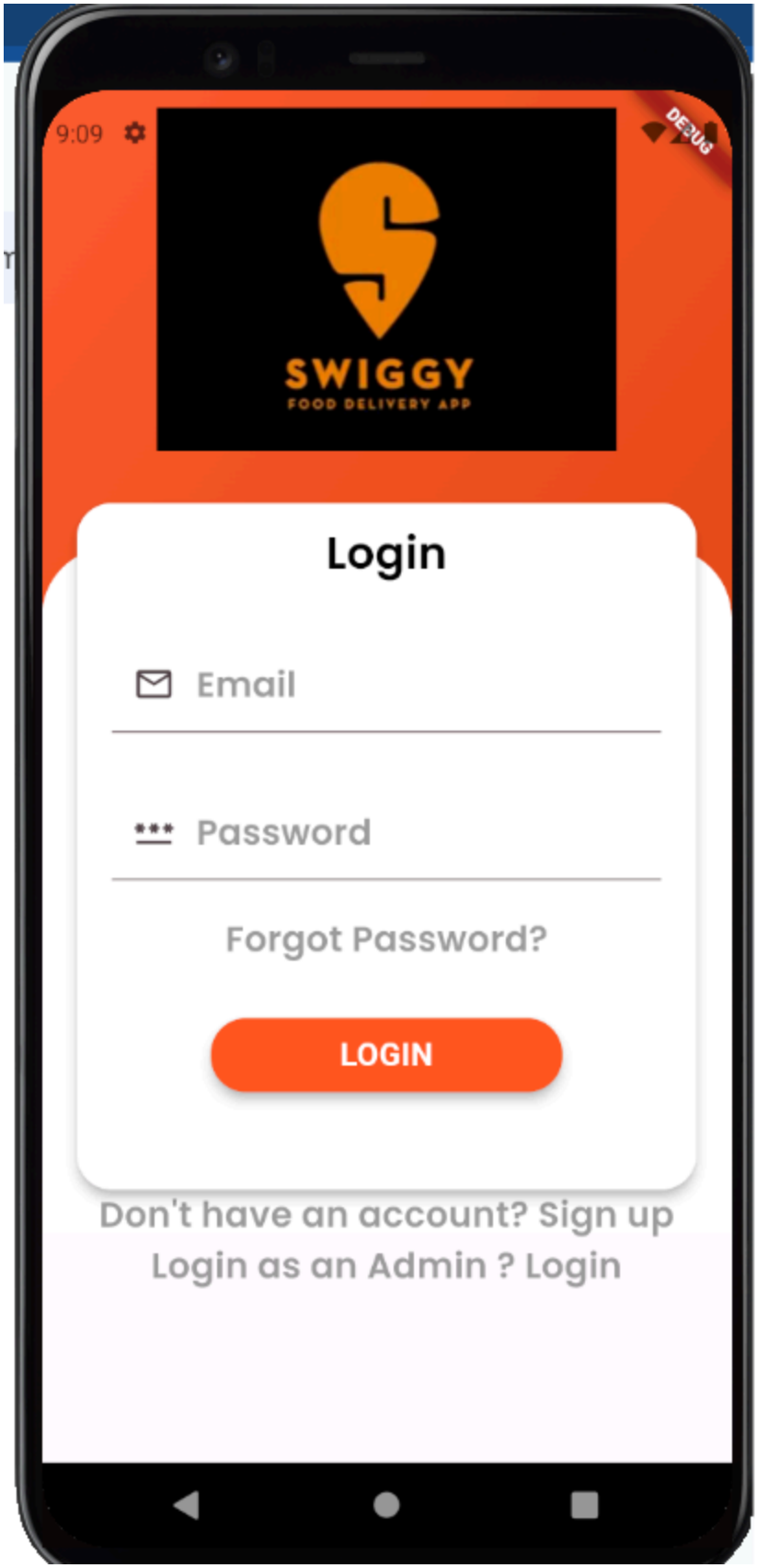
```
  validator: (value) {
    if (value == null || value.isEmpty) {
      return 'Please Enter Password';
    }
    return null;
  },
  obscureText: true,
  decoration: InputDecoration(
      hintText: 'Password',
      hintStyle: AppWidget.semiBoldTextFeildStyle(),
      prefixIcon: Icon(Icons.password_outlined)),
),
SizedBox(
  height: 20.0,
),
GestureDetector(
  onTap: () {
    Navigator.push(
        context,
        MaterialPageRoute(
            builder: (context) =>
                ForgotPassword()));
  },
  child: Container(
      // alignment: Alignment.topRight,
      alignment: Alignment.center,
      child: Text(
        "Forgot Password?",
        style: AppWidget.semiBoldTextFeildStyle(),
      )),
),
SizedBox(
  height: 30.0,
),
GestureDetector(
  onTap: () {
    if (_formkey.currentState!.validate()) {
      setState(() {
        email = useremailcontroller.text;
        password = userpasswordcontroller.text;
      });
    }
    userLogin();
  },
```

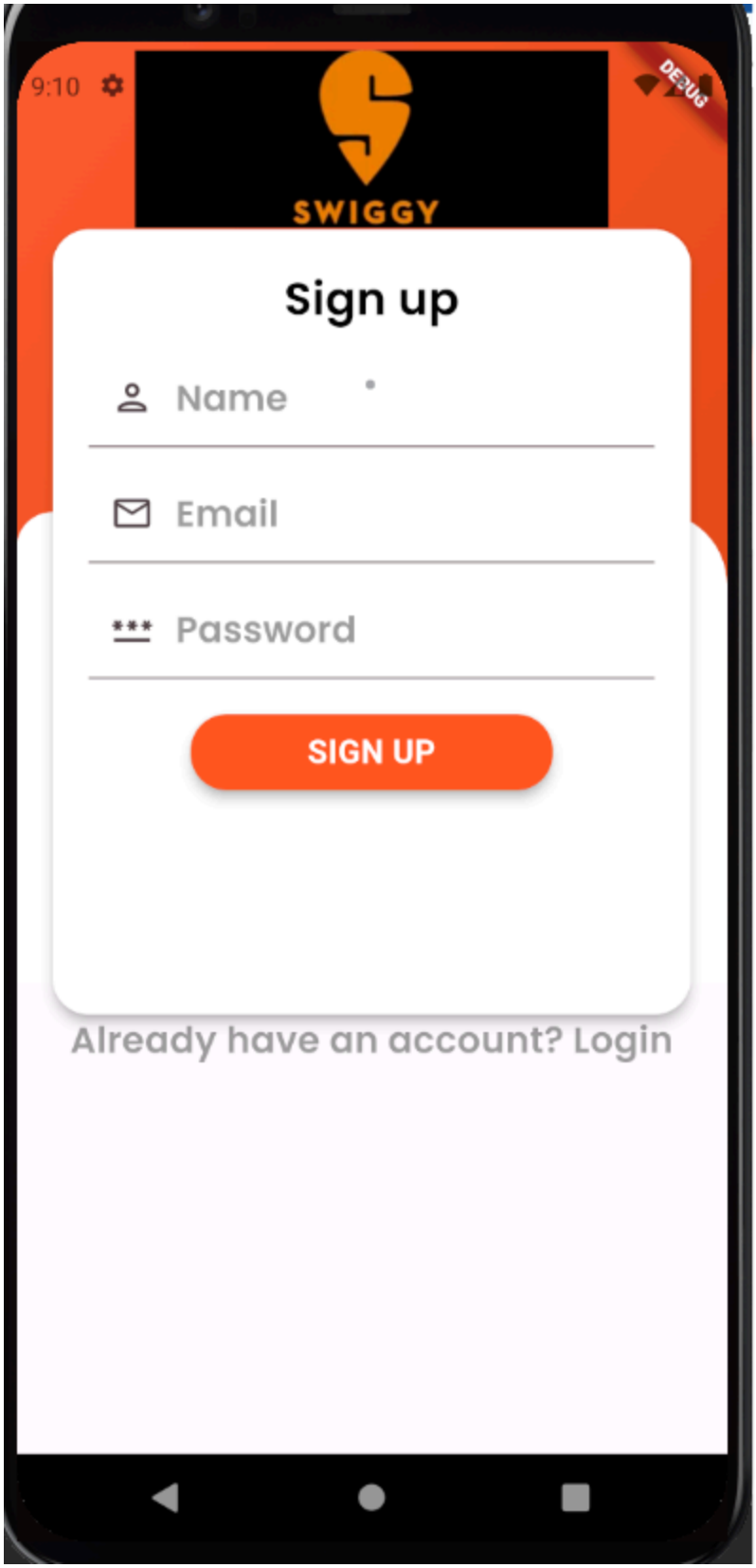
```
                child: Material(
                  elevation: 5.0,
                  borderRadius: BorderRadius.circular(20),
                  child: Container(
                    padding: EdgeInsets.symmetric(vertical: 8.0),
                    width: 200,
                    decoration: BoxDecoration(
                        color: Color(0Xffff5722),
                        borderRadius: BorderRadius.circular(20)),
                    child: Center(
                        child: Text(
                      "LOGIN",
                      style: TextStyle(
                          color: Colors.white,
                          fontSize: 18.0,
                          fontFamily: 'Poppins1',
                          fontWeight: FontWeight.bold),
                    )),
                  ),
                ),
              ),
            ],
          ),
        ),
      ),
      SizedBox(
        height: 0.0,
      ),
      GestureDetector(
          onTap: () {
            Navigator.push(context,
                MaterialPageRoute(builder: (context) => SignUp()));
          },
          child: Text(
            "Don't have an account? Sign up",
            style: AppWidget.semiBoldTextFeildStyle(),
          ))
    ],
  ),
)
],
),
),
```

```
    );
  }
}
```


9:15
DEBUG
SWIGGY
A GENIE FOR FOOD
-LOVERS
select from Our
Best Menu
Pick your food from our menu
Next


9:16
DEBUG
SWIGGY
Quick delivery
Deliver your food at your doorstep
Start


9:09
DEBUG
SWIGGY
FOOD DELIVERY APP
Login
Email
Password
Forgot Password?
LOGIN
Don't have an account? Sign up
Login as an Admin ? Login


9:10
DEBUG
SWIGGY
Sign up
Name
Email
Password
SIGN UP
Already have an account? Login

# MAD & PWA Lab
# Journal

| | |
|---|---|
| Experiment No. | 05 |
| Experiment Title. | To apply navigation, routing and gestures in Flutter App |
| Roll No. | 55 |
| Name | Mohd Sufiyan Siddiqui |
| Class | D15A |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation |
| Grade: | 15 |

**Project Title: Flutter:Swiggy Clone/PWA:Todo List** **Roll No. 55**

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

**Theory:**
Here's a theoretical outline on how to apply navigation, routing, and gestures in a Flutter app:

1. Understanding Navigation and Routing :
   - Navigation : Navigation refers to moving between different screens or pages within your app.
   - Routing : Routing involves defining the paths or routes that users can take to navigate through your app.

2. Implementing Navigation :
   - Use the `Navigator` widget to manage navigation within your app. This widget maintains a stack of routes and handles transitions between them.
   - Define named routes for each screen/page in your app using the `MaterialApp` or `CupertinoApp` widget's `routes` property.
   - Use the `Navigator.push()` method to navigate to a new screen, and `Navigator.pop()` to return to the previous screen.

3. Handling Route Transitions :
   - Customize route transitions using the `PageRouteBuilder` class or predefined transition animations like `MaterialPageRoute` or `CupertinoPageRoute`.
   - You can specify transition animations, duration, and other parameters to create visually appealing navigation effects.

4. Implementing Gestures :
   - Use gesture detection widgets like `GestureDetector` to detect user gestures such as taps, swipes, drags, etc.
   - Wrap the relevant widgets with `GestureDetector` and provide callback functions to handle specific gestures.
   - GestureDetector provides properties like `onTap`, `onDoubleTap`, `onLongPress`, `onHorizontalDrag`, `onVerticalDrag`, etc., to capture various types of gestures.

5. Adding Gesture Navigation :
   - Implement gesture-based navigation features, such as swiping to navigate between screens or using gestures to trigger specific actions.
   - Combine gestures with navigation methods like `Navigator.push()` to achieve seamless navigation based on user interactions.

6. Handling Complex Gestures :
   - For more complex gestures, consider using gesture recognizers like `DragGestureRecognizer`, `ScaleGestureRecognizer`, `PanGestureRecognizer`, etc., to detect and handle intricate user interactions.
   - Customize gesture recognizers to suit your app's requirements by defining custom gesture recognizer subclasses.

7. Testing and Refinement :
   - Thoroughly test your navigation and gesture implementation to ensure smooth and intuitive user experience.
   - Solicit feedback from users and iterate on your implementation to address any usability issues or bugs.
   - Continuously refine and optimize your navigation and gesture handling based on user feedback and evolving app requirements.

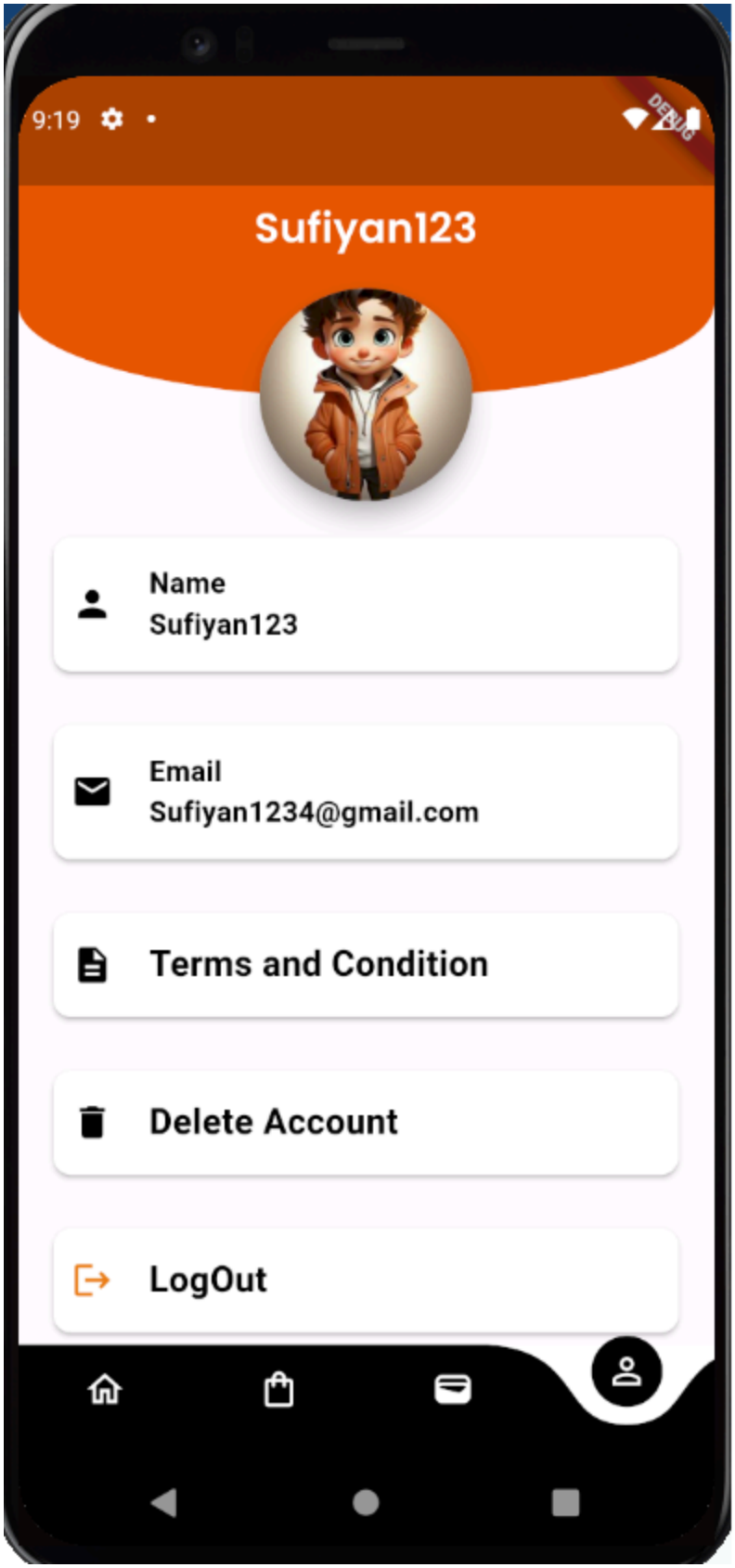
By following these steps, you can effectively incorporate navigation, routing, and gestures into your Flutter app to create a seamless and interactive user experience.

**Code:**

```
import 'package:curved_navigation_bar/curved_navigation_bar.dart';
import 'package:flutter/material.dart';
import 'package:flutter_application_1/pages/Home.dart';
import 'package:flutter_application_1/pages/order.dart';
import 'package:flutter_application_1/pages/profile.dart';
import 'package:flutter_application_1/pages/wallet.dart';

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

  @override
  State<BottomNav> createState() => _BottomNavState();
}

class _BottomNavState extends State<BottomNav> {
  int currentTabIndex = 0;

  late List<Widget> pages;
```

```
 late Widget currentPage;
 late Home homepage;
 late Profile profile;
 late Order order;
 late Wallet wallet;

 @override
 void initState() {
  homepage = Home();
  order = Order();
  profile = Profile();
  wallet = Wallet();
  pages = [homepage, order, wallet, profile];
  //pages = [homepage, order, profile, profile];
  super.initState();
 }

 @override
 Widget build(BuildContext context) {
  return Scaffold(
    bottomNavigationBar: CurvedNavigationBar(
        height: 65,
        backgroundColor: Colors.white,
        color: Colors.black,
        animationDuration: Duration(milliseconds: 500),
        onTap: (int index) {
          setState(() {
            currentTabIndex = index;
          });
        },
        items: [
          Icon(
            Icons.home_outlined,
            color: Colors.white,
          ),
          Icon(
            Icons.shopping_bag_outlined,
            color: Colors.white,
          ),
          Icon(
```

```
            Icons.wallet_outlined,
            color: Colors.white,
          ),
          Icon(
            Icons.person_outline,
            color: Colors.white,
          )
        ]),
    body: pages[currentTabIndex],
  );
 }
}
```


9:19
DEBUG
Sufiyan123
Name
Sufiyan123
Email
Sufiyan1234@gmail.com
Terms and Condition
Delete Account
LogOut

# MAD & PWA Lab
# Journal

| | |
|---|---|
| Experiment No. | 06 |
| Experiment Title. | To Connect Flutter UI with fireBase database |
| Roll No. | 55 |
| Name | Mohd Sufiyan Siddiqui |
| 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: | 15 |

**Project Title: Flutter:Swiggy Clone/PWA:Todo List** **Roll No. 55**

**AIM: To connect Flutter UI with firebase database**

**Theory:**

Flutter Connectivity with Firebase: A Comprehensive Guide

**Introduction**

In today's mobile-centric world, ensuring seamless internet connectivity is crucial for any application. Flutter, a popular framework for cross-platform development, offers robust tools for managing network connections, while Firebase, Google's cloud platform, provides powerful backend services. This note explores how to integrate connectivity management with Firebase in your Flutter app, ensuring a resilient and user-friendly experience even in fluctuating network conditions.

**Understanding Network Connectivity in Flutter**

1. **Flutter's Connectivity Plugin:**

    - Use the `connectivity` plugin to ascertain connectivity status (WiFi, mobile data, none).
    - Install the plugin: `flutter pub add connectivity`
    - Import and use the `Connectivity` class:

2. Dart

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

Future<void> main() async {
  var connectivityResult = await Connectivity().checkConnectivity();
  if (connectivityResult == ConnectivityResult.mobile) {
    // Use mobile data
  } else if (connectivityResult == ConnectivityResult.wifi) {
    // Use WiFi
  } else {
    // No internet connection
```

```
  }
}
```

3.
4.
5. **Network Change Monitoring:**

    - Subscribe to the `onConnectivityChanged` stream to receive updates on changes:

6. Dart

```
Connectivity().onConnectivityChanged.listen((ConnectivityResult result) {
  if (result == ConnectivityResult.none) {
    // Show an offline message
  }
});
```

7.
8.

**Connecting to Firebase**

1. **Firebase Setup:**

    - Create a Firebase project and configure authentication and databases as needed.
    - Follow the official guides: https://firebase.google.com/docs/flutter/setup

2.
3. **Firebase Services and Connectivity:**

    - Consider which Firebase services require internet access (e.g., Firestore, Authentication).
    - Handle offline behavior:

        - Local storage (e.g., SharedPreferences, Hive) for caching data.
        - Queueing operations for when connectivity resumes (e.g., using WorkManager).
    - 
4. 

**Best Practices and Considerations**

1. **Informative Error Handling:**
    - Provide clear and actionable error messages when offline or experiencing connectivity issues.
    - Use progress indicators to show data loading or syncing progress.
2. 
3. **Offline-First Design:**
    - Design your app to function even without internet access (e.g., showing cached data).
    - Prioritize essential features that work offline.
4. 
5. **Data Management:**
    - Manage offline data synchronization effectively:
        - Use Firestore's offline persistence and `enablePersistence` feature.
        - Implement conflict resolution strategies for merging local and remote data.
    - 
6. 

7. **Testing and Monitoring:**
    - Thoroughly test your app's behavior in offline scenarios.
    - Use Firebase Crashlytics or other tools to monitor network-related errors.
8.

**Advanced Techniques**

1. **Connectivity Plugins:**
    - Explore advanced plugins like `internet_connection_checker` for more granular insights.
2.
3. **Network-Aware Widgets:**
    - Leverage network-aware widgets (e.g., `OfflineIndicator`) for visual feedback.
4.
5. **Custom Offline Caching:**
    - Implement custom caching strategies for specific data types.
6.
7. **Background Tasks:**
    - Use WorkManager or other libraries to perform network-related tasks in the background.
8.

## Conclusion

Effectively managing network connectivity in Flutter with Firebase is essential for robust and user-friendly mobile apps. By incorporating the techniques and considerations discussed here, you can ensure that your app remains functional and provides a positive experience even in challenging network conditions.

**Additional Notes:**

- This guide provides a foundational understanding. Feel free to explore further based on your specific app's requirements.
- The Firebase SDK offers extensive documentation and examples for in-depth guidance.
- Continuously test and refine your connectivity logic for optimal performance and resilience.

**Code:**

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

import 'package:flutter/material.dart';

import 'package:flutter_application_1/pages/Home.dart';

import 'package:flutter_application_1/pages/bottomnav.dart';

import 'package:flutter_application_1/pages/onboard.dart';

import 'package:flutter_application_1/pages/signup.dart';

import 'package:flutter_application_1/widget/app_constant.dart';

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

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  Stripe.publishableKey = publishableKey;
  try {
    await Firebase.initializeApp(
        options: FirebaseOptions(
            apiKey: 'AIzaSyAbQE5bkqgk7E6PUWUfdjR6bWGX1G7_d68',
            appId: '1:955784621517:android:d83194633c4d1088ea2884',
            messagingSenderId: '955784621517',
            projectId: 'fooddeliveryapp-628c0'));
  } catch (e) {
    print('Error initializing Firebase: $e');
  }
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
```

```
  const MyApp({super.key});



  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        title: 'Flutter Demo',
        theme: ThemeData(
          colorScheme: ColorScheme.fromSeed(
              seedColor: Color.fromARGB(255, 185, 26, 108)),
          useMaterial3: true,
        ),


        home: Onboard());
  }
}
class MyHomePage extends StatefulWidget {
 const MyHomePage({super.key, required this.title});
```

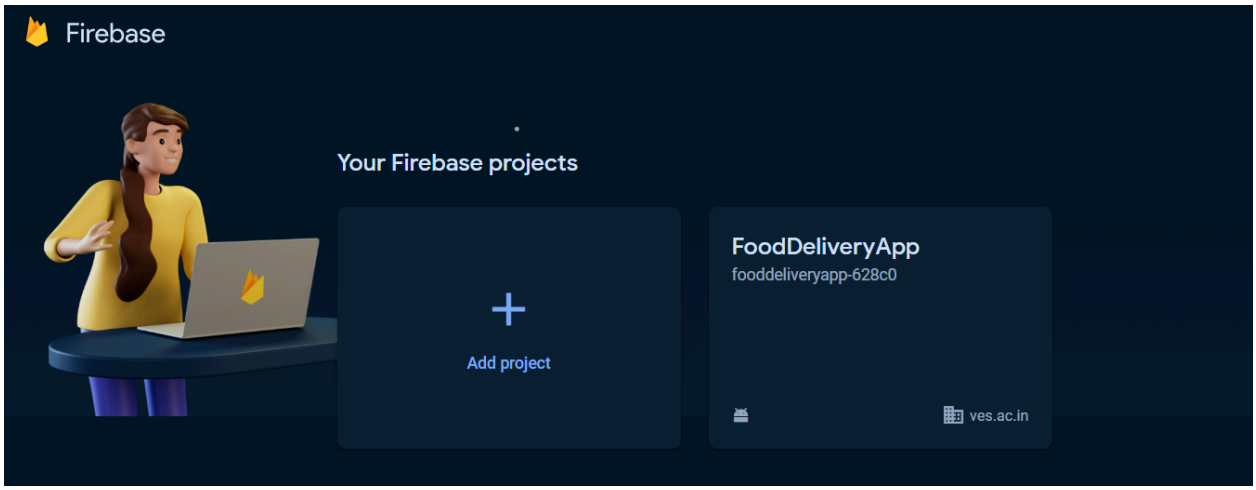
```
  final String title;

  @override

  State<MyHomePage> createState() => _MyHomePageState();

}



class _MyHomePageState extends State<MyHomePage> {

  int _counter = 0;



  void _incrementCounter() {

    setState(() {

      // This call to setState tells the Flutter framework that something has

      // changed in this State, which causes it to rerun the build method below

      // so that the display can reflect the updated values. If we changed

      // _counter without calling setState(), then the build method would not be

      // called again, and so nothing would appear to happen.

      _counter++;

    });

  }
```

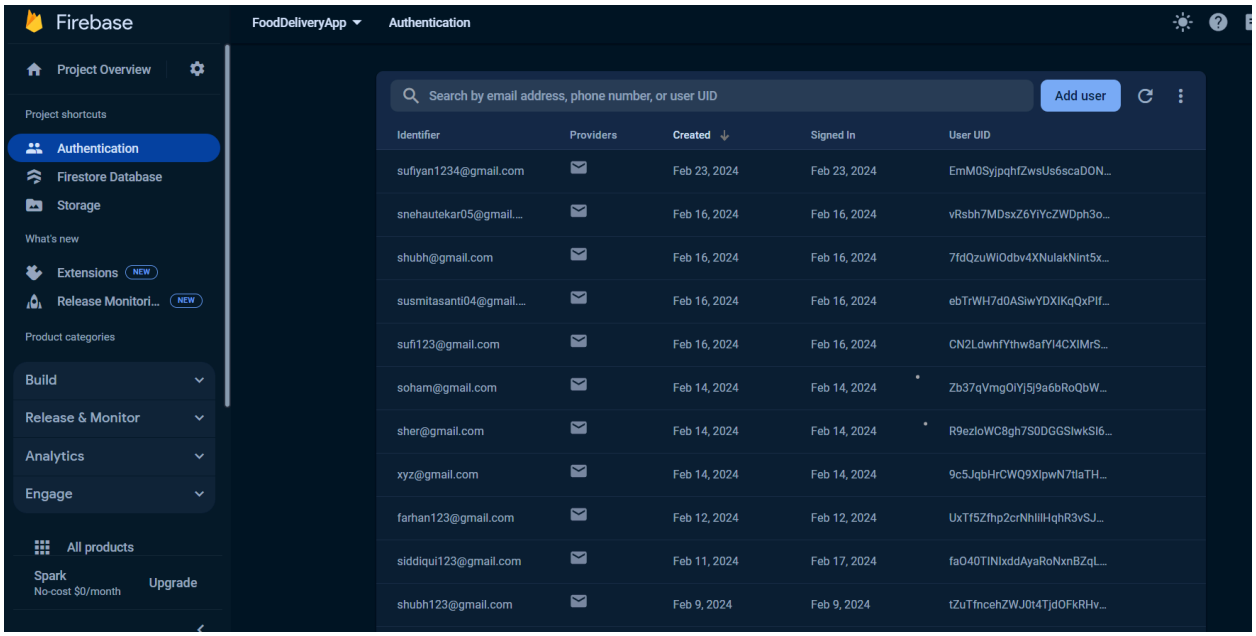
```
 @override
 Widget build(BuildContext context) {
  // This method is rerun every time setState is called, for instance as done
  // by the _incrementCounter method above.
  //
  // The Flutter framework has been optimized to make rerunning build methods
  // fast, so that you can just rebuild anything that needs updating rather
  // than having to individually change instances of widgets.
  return Scaffold(
    appBar: AppBar(


      ),
    body: Center(
      // Center is a layout widget. It takes a single child and positions it
      // in the middle of the parent.
      child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
```

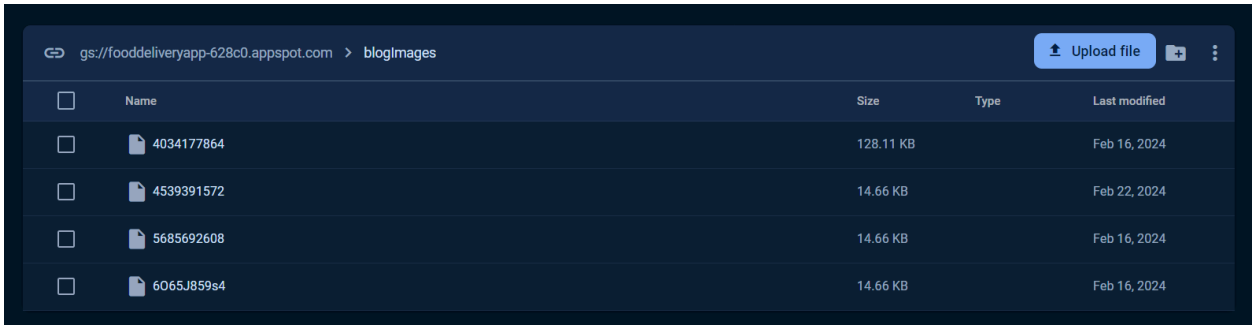
```
        children: <Widget>[
          const Text(
            'You have pushed the button this many times:',
          ),
          Text(
            '$_counter',
            style: Theme.of(context).textTheme.headlineMedium,
          ),
        ],
      ),
    ),
    floatingActionButton: FloatingActionButton(
      onPressed: _incrementCounter,
      tooltip: 'Increment',
      child: const Icon(Icons.add),
    ), // This trailing comma makes auto-formatting nicer for build methods.
  );
}
```

}

Screenshots:



# MAD & PWA Lab
# Journal

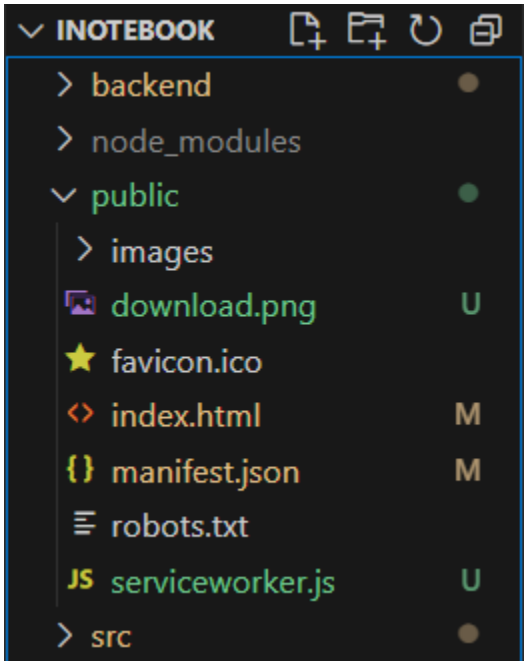
| | |
|---|---|
| Experiment No. | 07 |
| Experiment Title. | To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”. |
| Roll No. | 55 |
| Name | Mohd Sufiyan Siddiqui |
| Class | D15A |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO4: Understand various PWA frameworks and their requirements |
| Grade: | 15 |

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

**THEORY:**

Regular Web App

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




Progressive Web App

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

**manifest.json**

```
{
   "short_name": "React App",
   "name": "Create React App Sample",
   "icons": [
     {
       "src": "favicon.ico",
       "sizes": "64x64 32x32 24x24 16x16",
       "type": "image/x-icon"
     },
     {
       "src": "logo192.png",
       "type": "image/png",
       "sizes": "192x192"
     },
     {
       "src": "logo512.png",
       "type": "image/png",
       "sizes": "512x512"
     }
   ],
   "start_url": ".",
   "display": "standalone",
   "theme_color": "#000000",
   "background_color": "#ffffff"
 }
```

**serviceworker.js**

```
var staticCacheName = "pwa";

self.addEventListener("install", function (e) {
e.waitUntil(
   caches.open(staticCacheName).then(function (cache) {
   return cache.addAll(["/"]);
   })
);
});

self.addEventListener("fetch", function (event) {
console.log(event.request.url);

event.respondWith(
   caches.match(event.request).then(function (response) {
   return response || fetch(event.request);
```

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

**public/index.html**

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8" />
  <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <meta name="theme-color" content="#000000" />
  <meta name="description" content="Web site created using create-react-app" />
  <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
  <!--
    manifest.json provides metadata used when your web app is installed on a
    user's mobile device or desktop. See
https://developers.google.com/web/fundamentals/web-app-manifest/
  -->
  <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
  <!--
    Notice the use of %PUBLIC_URL% in the tags above.
    It will be replaced with the URL of the `public` folder during the build.
    Only files inside the `public` folder can be referenced from the HTML.

    Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
    work correctly both with client-side routing and a non-root public URL.
    Learn how to configure a non-root public URL by running `npm run build`.
  -->

  <!-- Bootstrap CSS -->
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.rtl.min.css"

integrity="sha384-PJsj/BTMqILvmcej7ulplguok8ag4xFTPryRq8xevL7eBYSmpXKcbNVuy+P0
RMgq" crossorigin="anonymous">

  <title>iNoteBook</title>
</head>
```

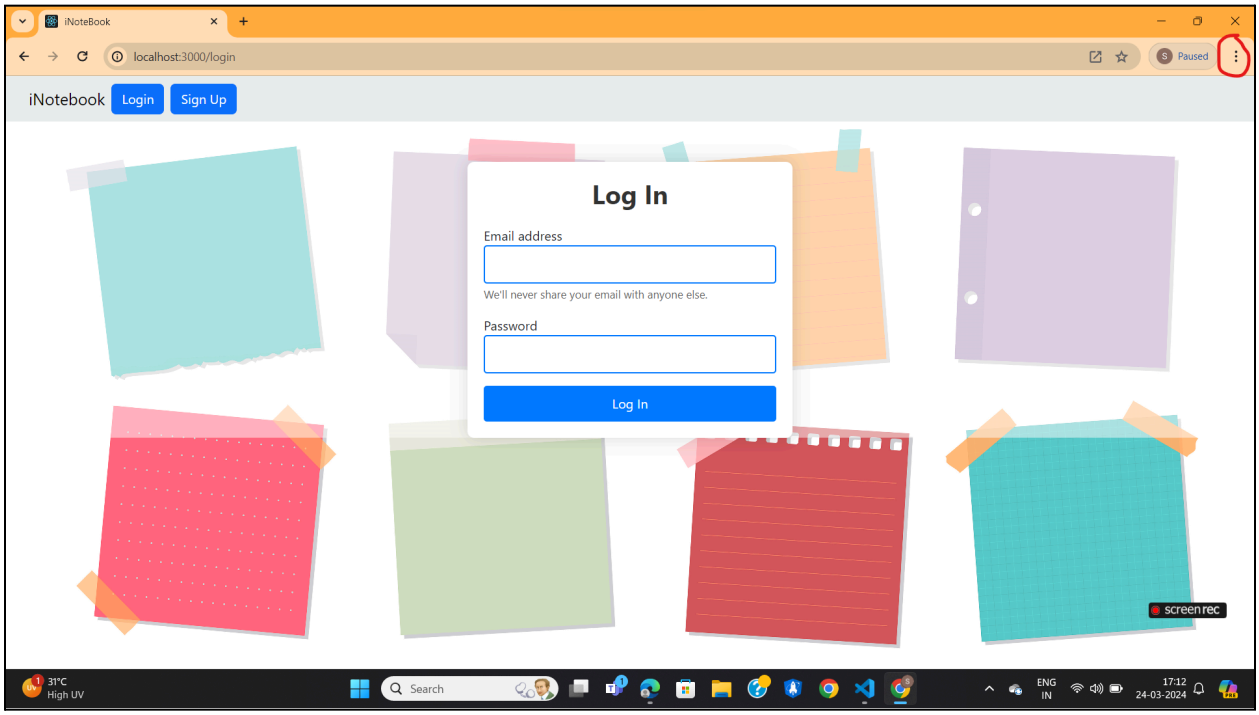
```
<body>
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root"></div>
  <!--
     This HTML file is a template.
     If you open it directly in the browser, you will see an empty page.

     You can add webfonts, meta tags, or analytics to this file.
     The build step will place the bundled scripts into the <body> tag.

     To begin the development, run `npm start` or `yarn start`.
     To create a production bundle, use `npm run build` or `yarn build`.
    -->

  <!-- Option 1: Bootstrap Bundle with Popper -->
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"

integrity="sha384-geWF76RCwLtnZ8qwWowPQNguL3RmwHVBC9FhGdlKrxdiJJigb/j/68SIy
3Te4Bkz"
    crossorigin="anonymous"></script>

    <script src="https://kit.fontawesome.com/e7a7193d30.js" crossorigin="anonymous"></script>

    <script>
        window.addEventListener("load", () => {
          registerSW();
        });

        // Register the Service Worker
        async function registerSW() {
          if ("serviceWorker" in navigator) {
            try {
              await navigator;
              ServiceWorker;
              register("serviceworker.js");
            } catch (e) {
              console.log("SW Registration Failed.");
            }
          }
        }
```

```
    </script>
</body>
</html>
```

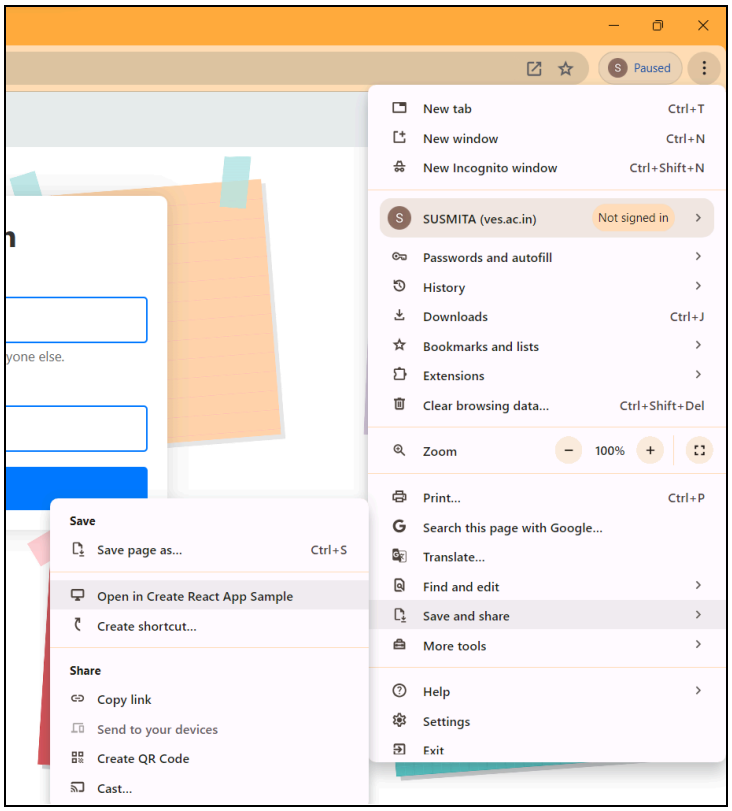
COMMAND: **npm run start**

```
You can now view inotebook in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.0.104:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

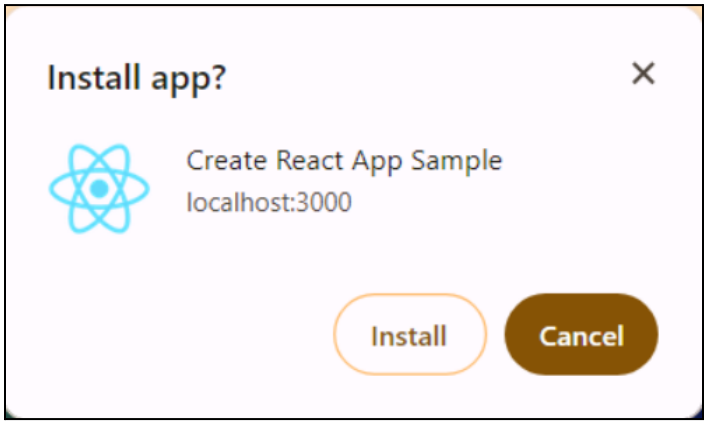
Open localhost:3000 on the browser




Click on 3 dots on the top right corner of the browser → go to **Save and Share** → Click on **Install Create React App Sample**

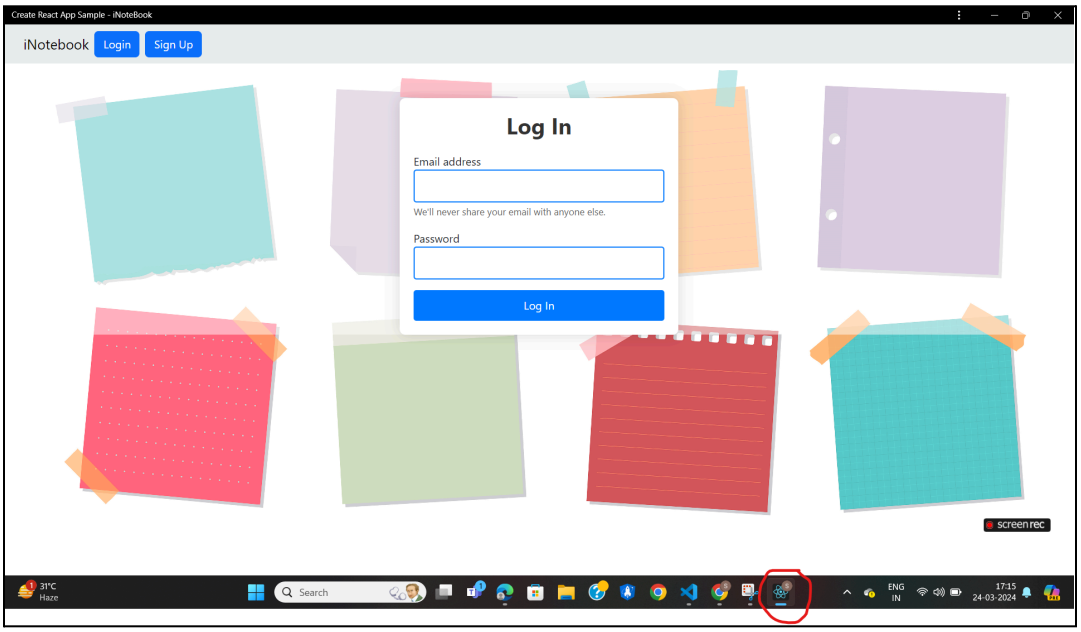



You will get the following pop-up:




Click on **Install**.

**App Icon will appear at the bottom:**




**CONCLUSION:**

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

# MAD & PWA Lab
# Journal

| | |
|---|---|
| Experiment No. | 08 |
| Experiment Title. | To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA |
| Roll No. | 55 |
| Name | Mohd Sufiyan Siddiqui |
| Class | D15A |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO5: Design and Develop a responsive User Interface by applying PWA Design techniques |
| Grade: | 15 |

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

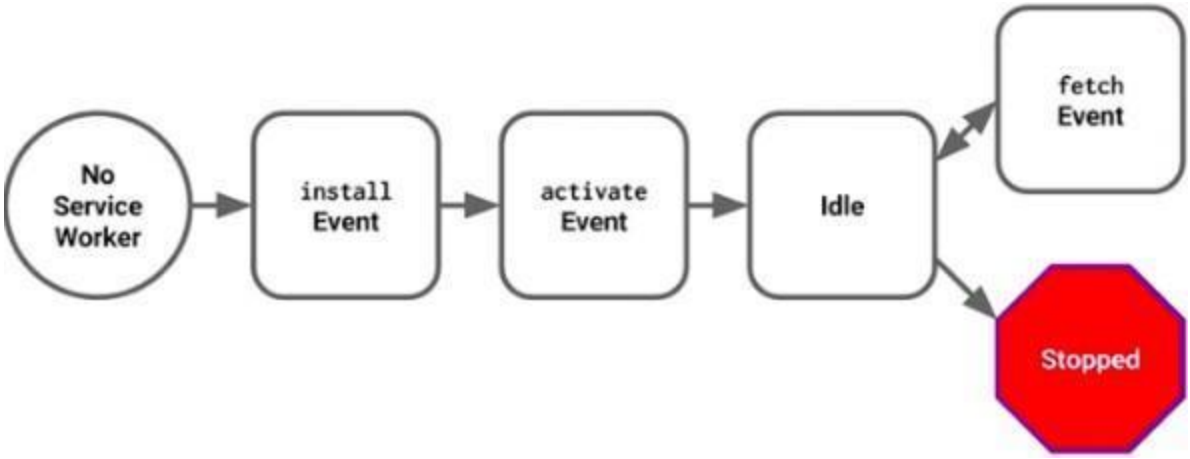
- You can manage **Push Notifications**
  You can manage push notifications with Service Worker and show any information message to the user.
- You can **Continue**
  Although Internet connection is broken, you can start any process with Background Sync of Service Worker.

**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.

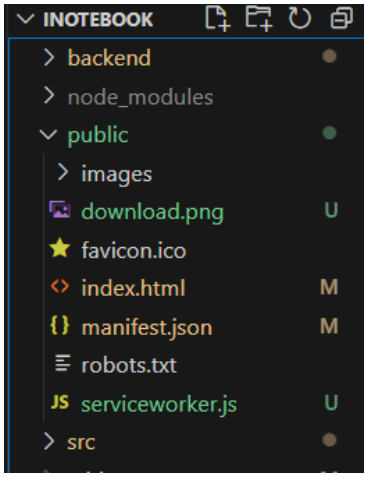
**Service Worker Cycle**




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

- Registration
- Installation
- Activation

My Project's folder structure:




**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.

**CODE:**

**public/index.html**

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop. See
https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->

    <!-- Bootstrap CSS -->
    <link
      rel="stylesheet"
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.rtl.min.css"

integrity="sha384-PJsj/BTMqILvmcej7ulplguok8ag4xFTPryRq8xevL7eBYSmpXKcbNVuy+P0
RMgq"
```

```
    crossorigin="anonymous"
  />

  <title>iNoteBook</title>
 </head>

 <body>
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root"></div>
  <!--
    This HTML file is a template.
    If you open it directly in the browser, you will see an empty page.

    You can add webfonts, meta tags, or analytics to this file.
    The build step will place the bundled scripts into the <body> tag.

    To begin the development, run `npm start` or `yarn start`.
    To create a production bundle, use `npm run build` or `yarn build`.
  -->

  <!-- Option 1: Bootstrap Bundle with Popper -->
  <script
    src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"

integrity="sha384-geWF76RCwLtnZ8qwWowPQNguL3RmwHVBC9FhGdlKrxdiJJigb/j/68SIy
3Te4Bkz"
    crossorigin="anonymous"
  ></script>

  <script
    src="https://kit.fontawesome.com/e7a7193d30.js"
    crossorigin="anonymous"
  ></script>

  <script>
    window.addEventListener("load", () => {
      registerSW();
    });

    // Register the Service Worker
```

```
    async function registerSW() {
      if ("serviceWorker" in navigator) {
        try {
          const registration = await navigator.serviceWorker.register(
            "serviceworker.js"
          );
          console.log("Registered Service Worker:", registration);
        } catch (error) {
          console.log("SW Registration Failed:", error);
        }
      }
    }
  </script>
 </body>
</html>
```

**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:

```
self.addEventListener("install", function (e) {
 e.waitUntil(
  caches.open(staticCacheName).then(function (cache) {
   return cache.addAll(["/"]);
  })
 );
});
```

*~This is done in serviceworker.js*

**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.

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

*~This is done in serviceworker.js*

Therefore, serviceworker.js finally looks like this:

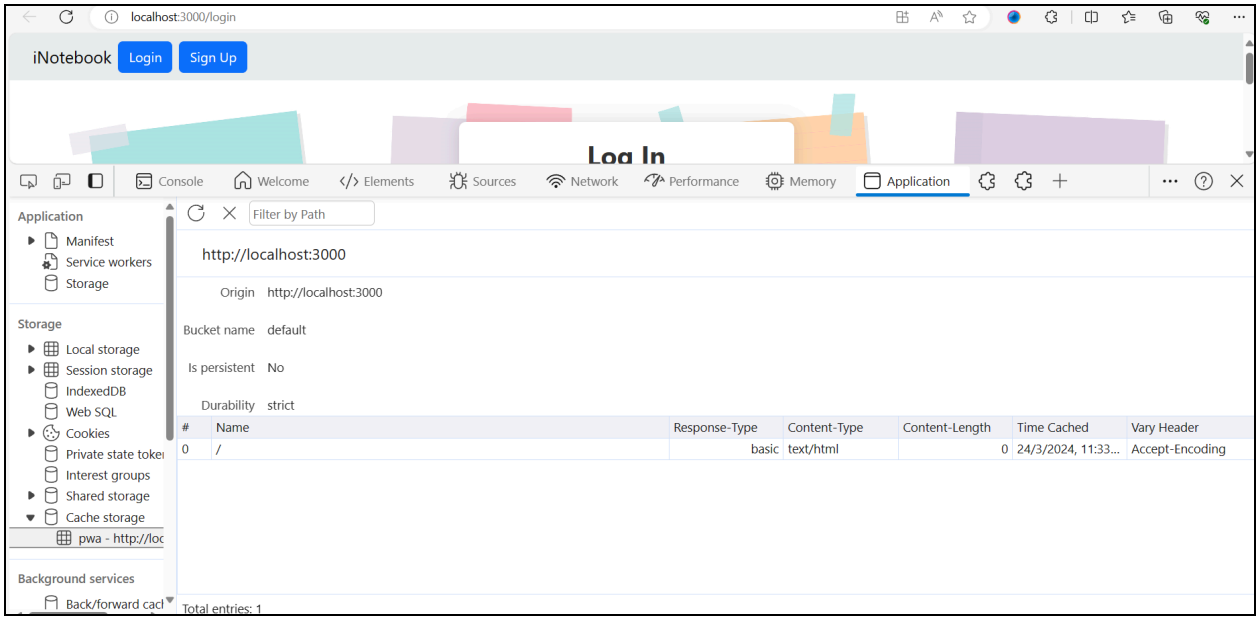
```
var staticCacheName = "pwa";

self.addEventListener("install", function (e) {
 e.waitUntil(
  caches.open(staticCacheName).then(function (cache) {
   return cache.addAll(["/"]);
  })
 );
});

self.addEventListener('activate', event => {
  event.waitUntil(
    caches.keys().then(cacheNames => {
      return Promise.all(
```

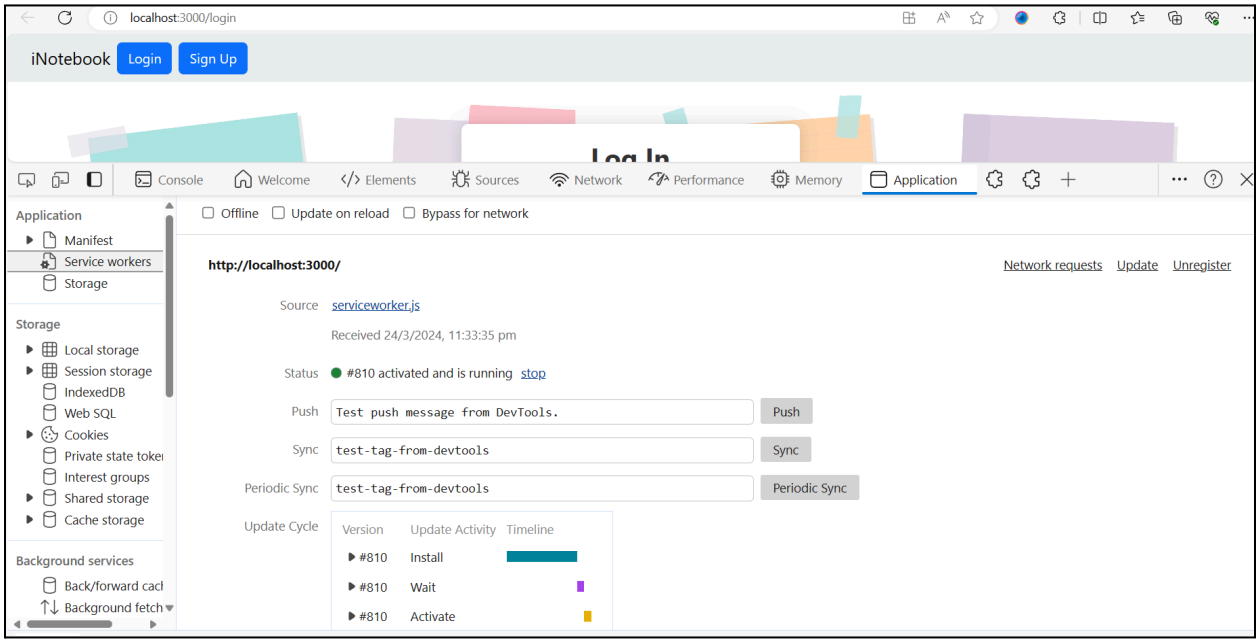
```
            cacheNames.filter(name => {
               return name !== staticCacheName;
            }).map(name => {
               return caches.delete(name);
            })
         );
      })
   );
});

self.addEventListener("fetch", function (event) {
  console.log(event.request.url);

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

We now run the app using the following command:
> npm run start

```
You can now view inotebook in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.0.104:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

On the browser:




## Conclusion:

The aim was to implement a service worker for aProgressive Web Application (PWA), enabling offline functionality and improving performance. This involved coding and registering the service worker (serviceworker.js) to intercept network requests, cache essential resources, and enhance the offline experience. By completing the install and activation process, the PWA gained features like offline access and faster page loads, enhancing user experience and resilience to network issues.

# 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. | 55 |
| Name | Mohd Sufiyan Siddiqui |
| Class | D15A |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO5: Design and Develop a responsive User Interface by applying PWA Design techniques |
| Grade: | 15 |

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

**Theory**:

**Service Worker**

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

Things to note about Service Worker:

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

**Fetch Event**

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

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

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

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

**Sync Event**

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

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

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

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

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

**Push Event**

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

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

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

### serviceworker.js

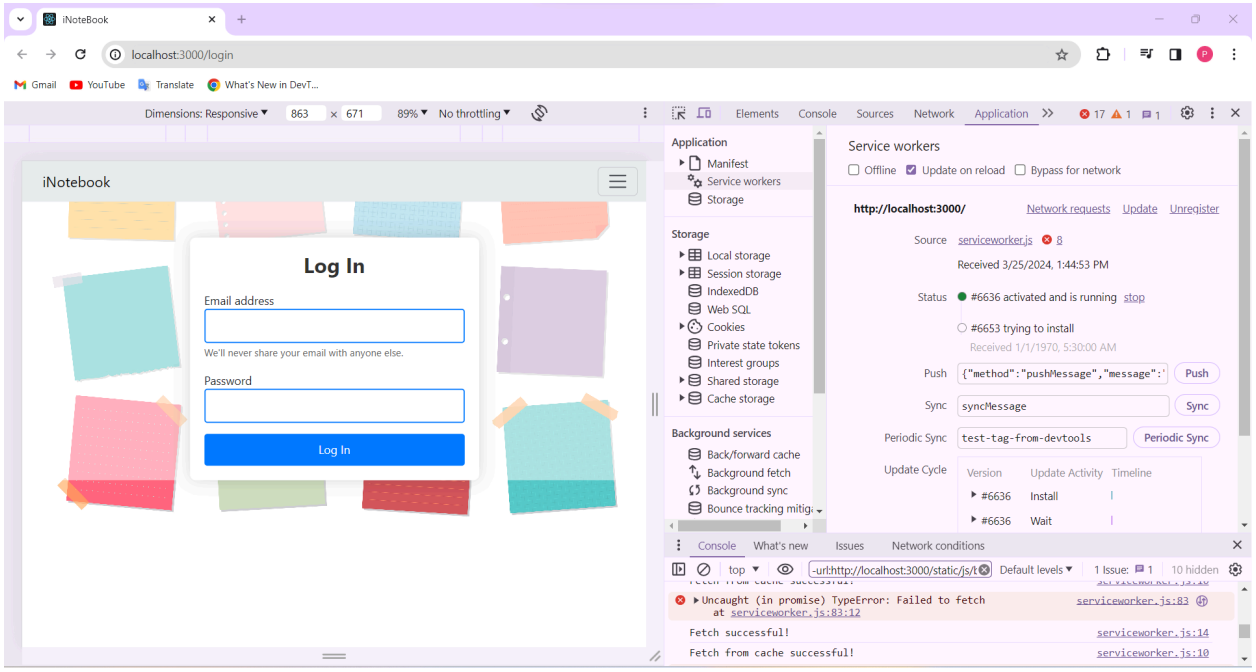
```
var staticCacheName = "pwa";

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

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

self.addEventListener("sync", (event) => {
  if (event.tag === "syncMessage") {
    console.log("Sync successful!");
  }
});

self.addEventListener("push", function (event) {
    if (event && event.data) {
      var data = event.data.json();
      if (data.method == "pushMessage") {
        console.log("Push notification requested");

        if (Notification.permission === "granted") {
          event.waitUntil(
            self.registration.showNotification("To Do List", {
              body: data.message,
            }).catch(function(error) {
              console.error("Error showing notification:", error);
            })
          );
        } else if (Notification.permission === "denied") {
          console.error("Notification permission denied.");
```

```
        // Handle the case where notification permission is denied, such as showing a message to
the user.
      } else {
        Notification.requestPermission().then(function(permission) {
          if (permission === "granted") {
            console.log("Notification permission granted, showing notification");
            self.registration.showNotification("To Do List", {
              body: data.message,
            });
          } else {
            console.error("Notification permission denied.");
            // Handle the case where notification permission is denied after requesting.
          }
        });
      }
    }
  }
 });


var filesToCache = ["/login", "/Home", "/About", "/index.html"];

var preLoad = function () {
  return caches.open("index").then(function (cache) {
    return cache.addAll(filesToCache);
  });
};

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

var addToCache = function (request) {
  return caches.open("index").then(function (cache) {
```

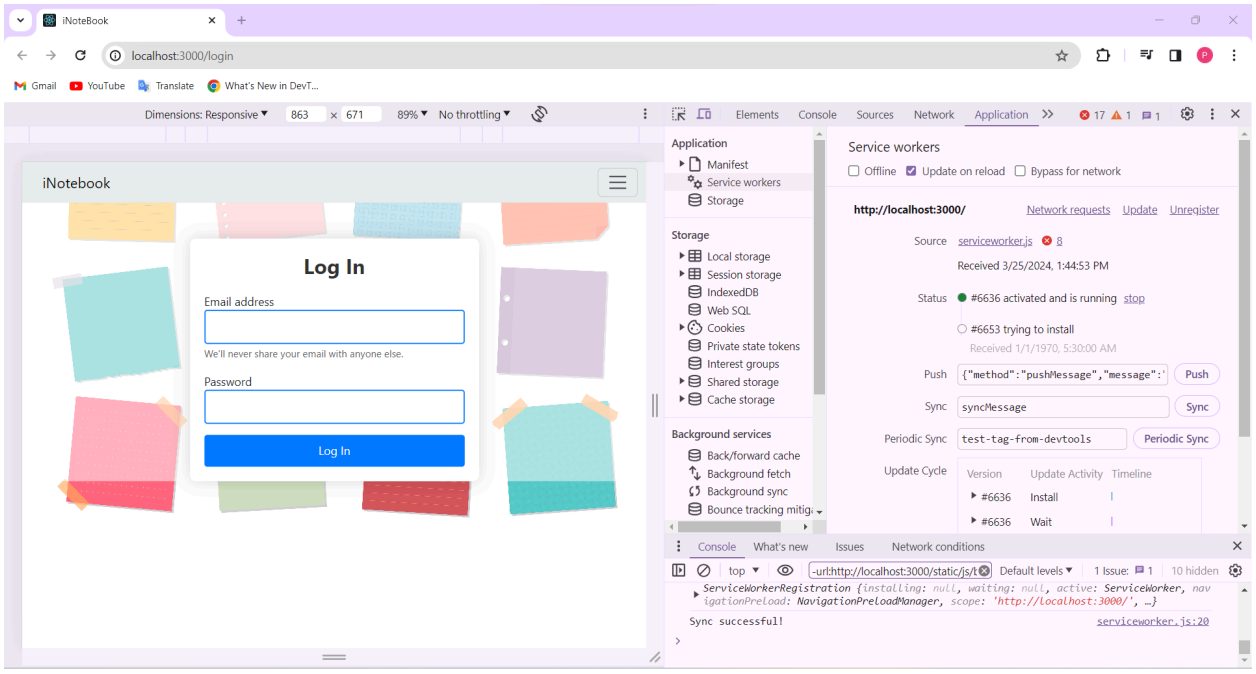
```
    return fetch(request).then(function (response) {
      return cache.put(request, response);
    });
  });
};

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

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

Fetch Event :

**Project Title: Flutter:Swiggy Clone/PWA:Todo List** **Roll No. 55**



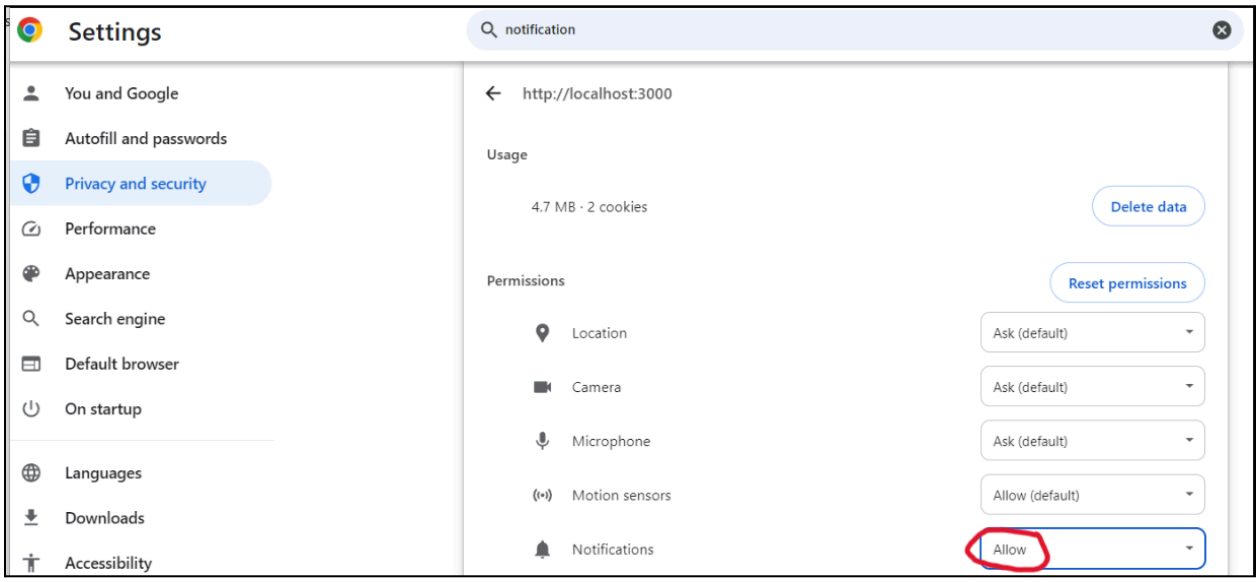

Sync Event :



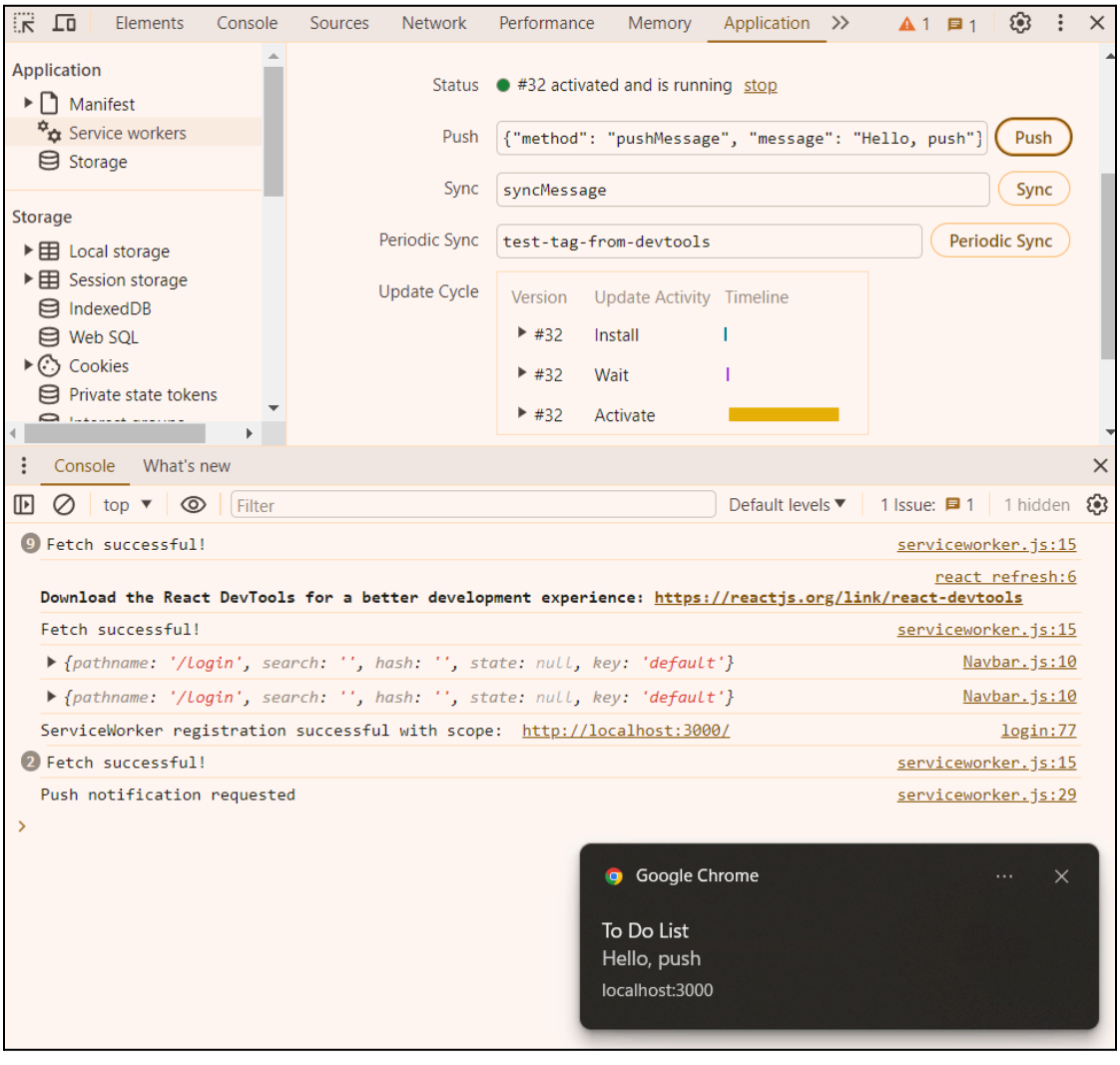

Push Event:

```
Push notification requested                                    serviceworker.js:29
▼Notification permission denied.                               serviceworker.js:42
(anonymous) @ serviceworker.js:42
```

Go to Site Settings and allow Notifications:




Now, we can see the desired result:



## CONCLUSION:

- The aim is to enhance the functionality and user experience of a Progressive Web App (PWA) by implementing service worker events such as fetch, sync, and push.
- These events enable the app to efficiently cache resources for offline access, synchronize data in the background, and deliver timely notifications to users, thereby improving performance, reliability, and engagement.

# MAD & PWA Lab
# Journal

| | |
|---|---|
| Experiment No. | 10 |
| Experiment Title. | To study and implement deployment of Ecommerce PWA to GitHub Pages. |
| Roll No. | 55 |
| Name | Mohd Sufiyan Siddiqui |
| Class | D15A |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO5: Design and Develop a responsive User Interface by applying PWA Design techniques |
| Grade: | 15 |

**AIM:**

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

**THEORY:**

**GitHub Pages**

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

GitHub Pages provides the following key features:

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

Reasons for favoring this over Firebase:

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

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

Pros

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

Cons

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

**Firebase**

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

Some of the features offered by Firebase are:

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

Reasons for favoring over GitHub Pages:

1. Realtime backend made easy
2. Fast and responsive

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

Pros

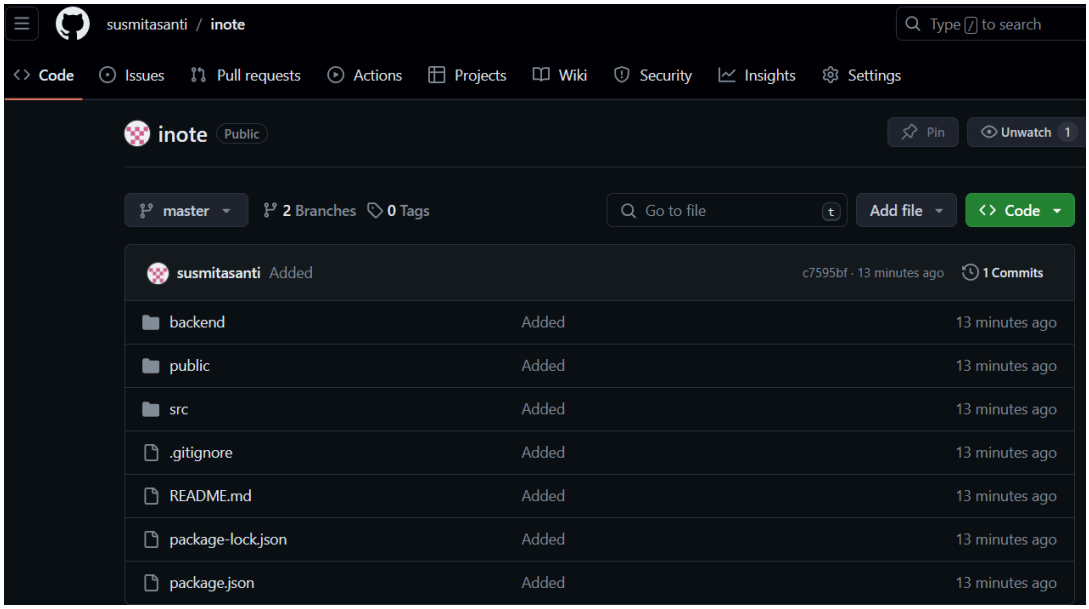
1. Hosted by Google. Enough said.
2. Authentication, Cloud Messaging, and a whole lot of other handy services will be available to you.
3. A real-time database will be available to you, which can store 1 GB of data.
4. You'll also have access to a blob store, which can store another 1 GB of data.
5.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

Cons

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

**Project Title: Flutter:Swiggy Clone/PWA:Todo List** **Roll No. 55**

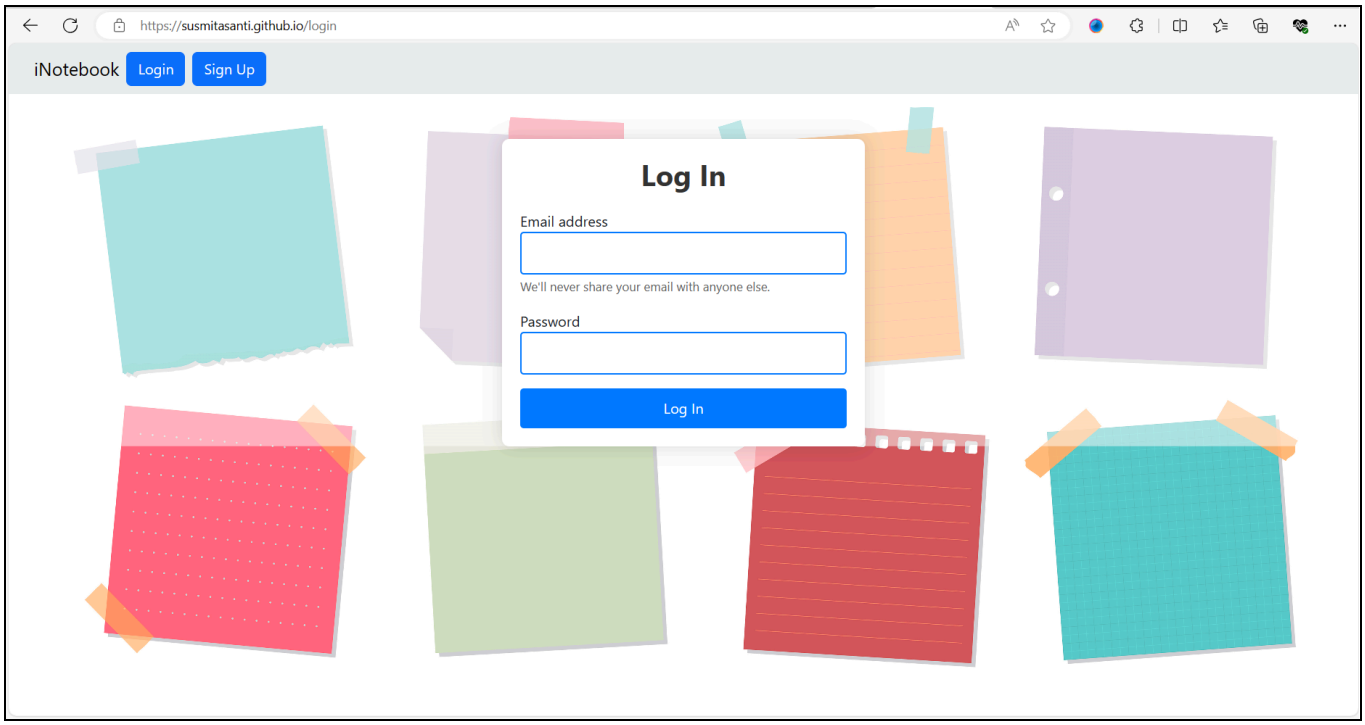
**Link to our GitHub repository: [susmitasanti/inote (github.com)]**




**HOSTED LINK:**
**[iNoteBook (susmitasanti.github.io)]**

**HOSTED IMAGE:**

**Project Title: Flutter:Swiggy Clone/PWA:Todo List** **Roll No. 55**



# MAD & PWA Lab
# Journal

| | |
|---|---|
| Experiment No. | 11 |
| Experiment Title. | To use google Lighthouse PWA Analysis Tool to test the PWA functioning. |
| Roll No. | 55 |
| Name | Mohd Sufiyan Siddiqui |
| Class | D15A |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO6: Develop and Analyze PWA Features and deploy it over app hosting solution |
| Grade: | 15 |

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

**THEORY:**

**Google Lighthouse :**

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

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

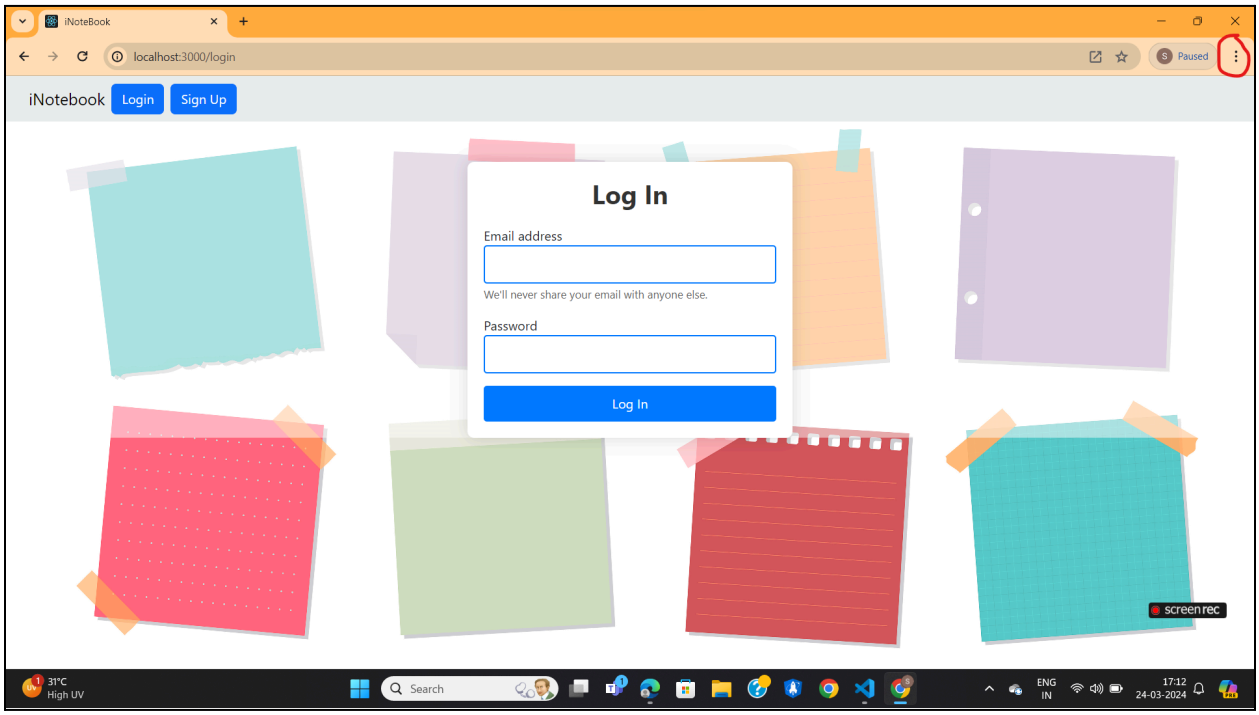
**Key Features and Audit Metrics**

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

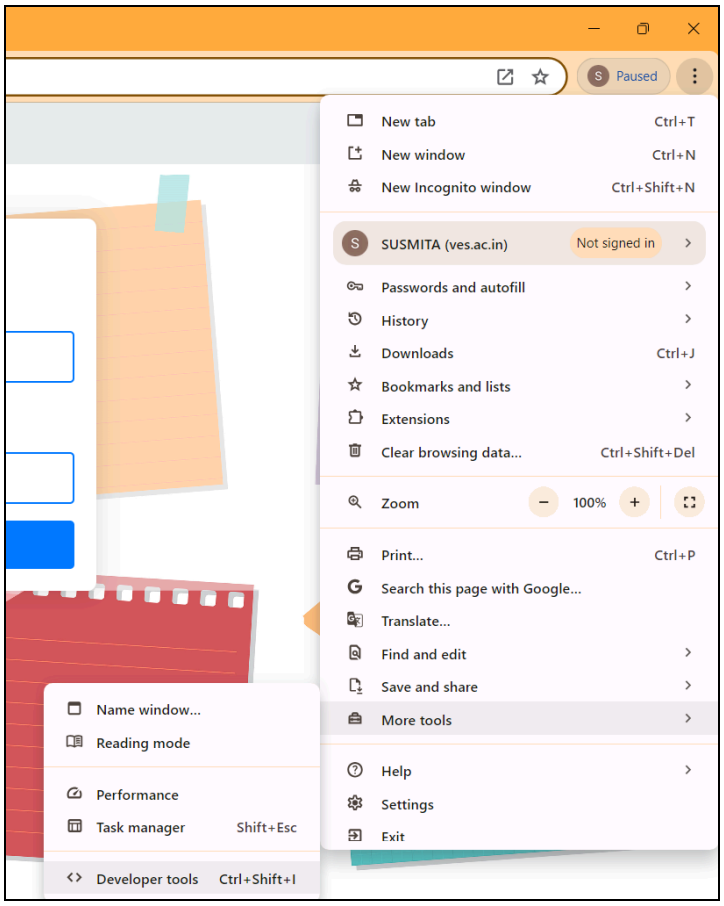
1. **Performance:** This score is an aggregation of how the page fared in aspects such as (but not limited to) loading speed, time taken for loading for basic frame(s), displaying meaningful content to the user, etc. To a layman, this score is indicative of how decently the site performs, with a score of 100 meaning that you figure in the 98th percentile, 50 meaning that you figure in the 75th percentile and so on.
2. **PWA Score (Mobile):** Thanks to the rise of Service Workers, app manifests, etc., a lot of modern web applications are moving towards the PWA paradigm, where the objective is to make the application behave as close as possible to native mobile applications. Scoring points are based on the Baseline PWA checklist laid down by Google which includes Service Worker implementation(s), viewport handling, offline functionality, performance in script-disabled environments, etc.
3. **Accessibility:** As you might have guessed, this metric is a measure of how accessible your website is, across a plethora of accessibility features that can be implemented in your page (such as the 'aria-' attributes like aria-required, audio captions, button names, etc.). Unlike the other metrics though, Accessibility metrics score on a pass/fail basis i.e. if all possible elements of the page are not screen-reader friendly (HTML5 introduced features that would make pages easy to interpret for screen readers used by visually challenged people like tag names, tags such as <section>, <article>,

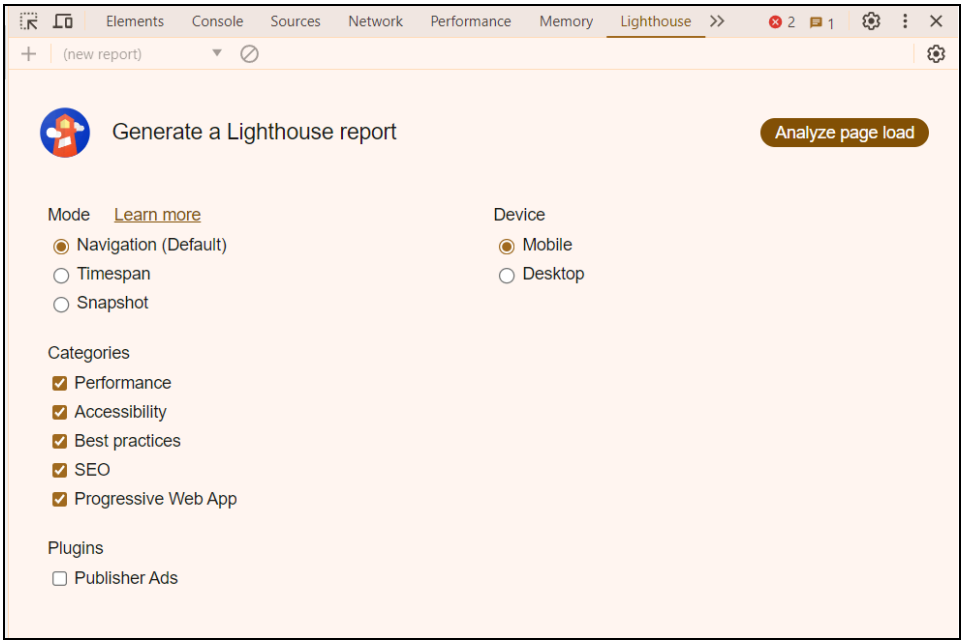
etc.), you get a 0 on that score. The aggregate of these scores is your Accessibility metric score.

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



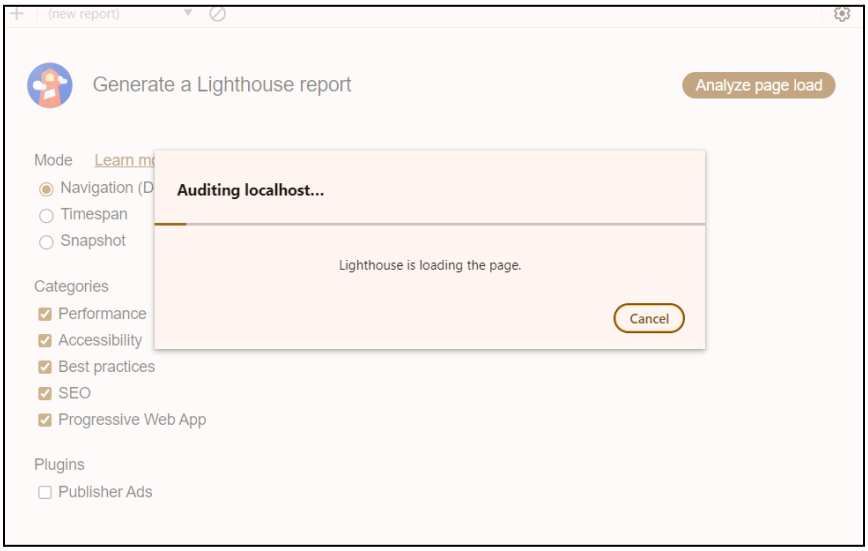

Click on the 3 dots on the top-right corner of the page → Click on **More Tools** → Click on **Developer Tools**

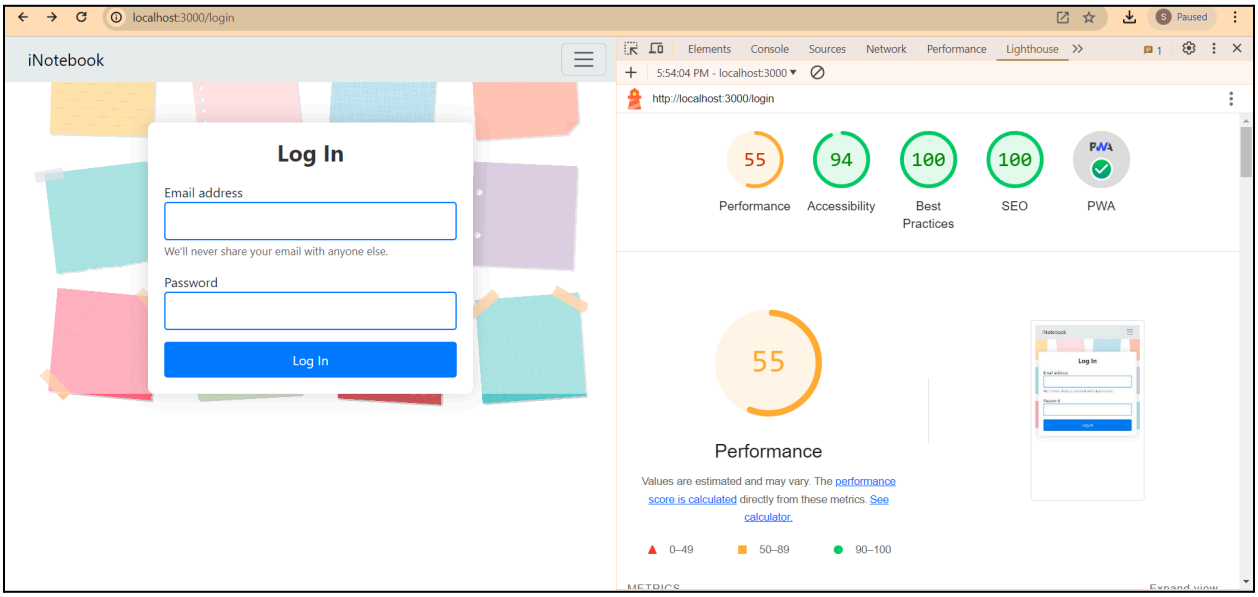


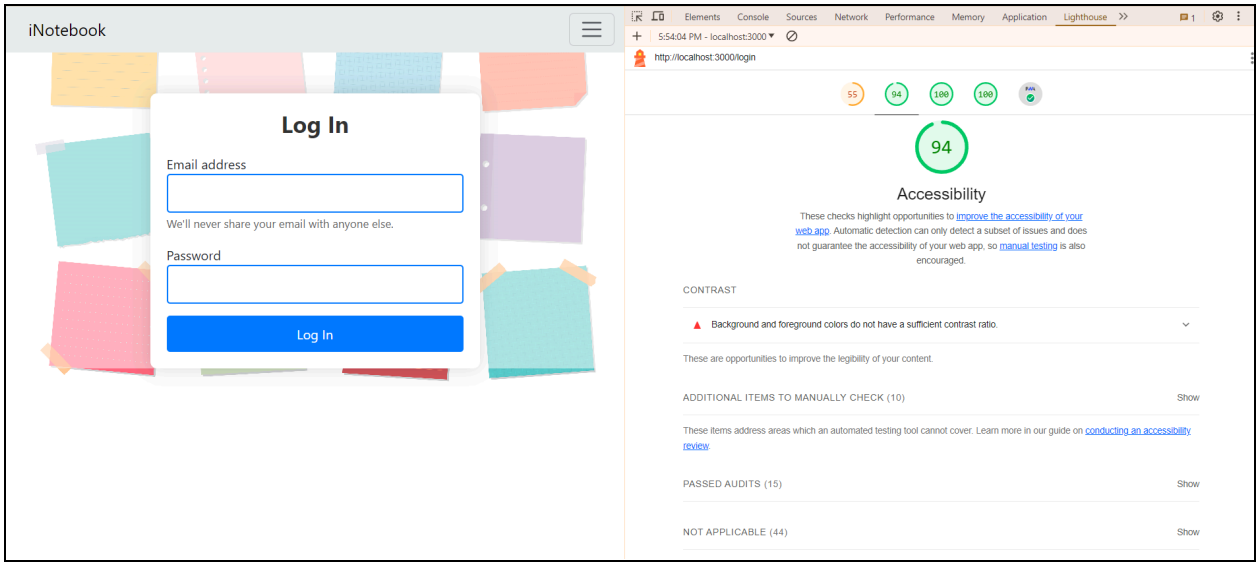

Click on **Analyze page load**

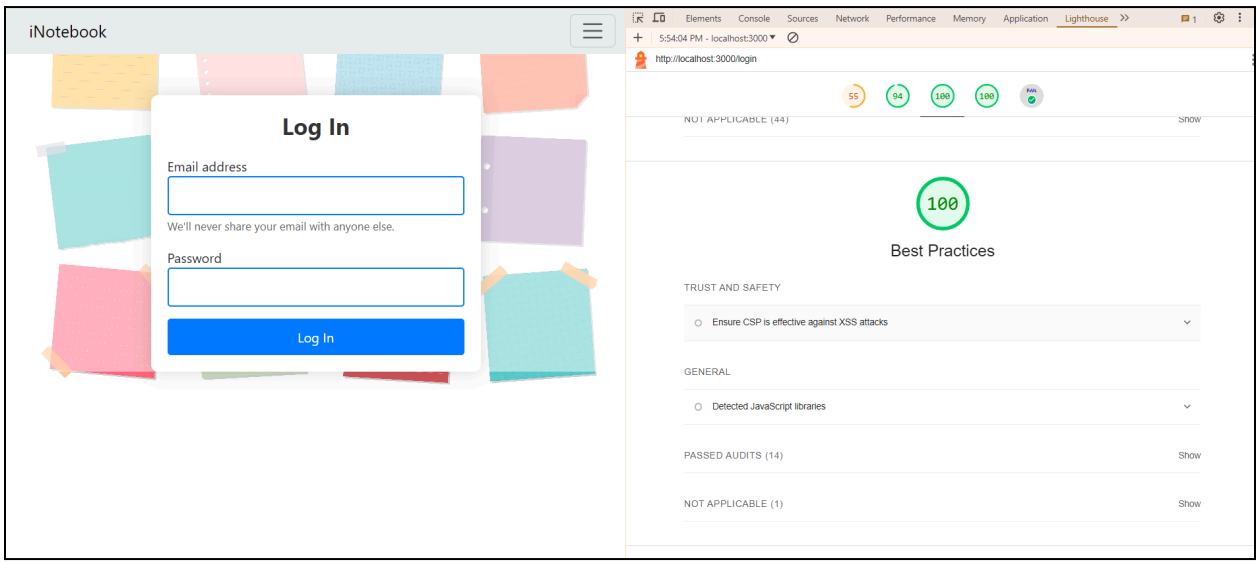
**Project Title: Flutter:Swiggy Clone/PWA:Todo List** **Roll No. 55**

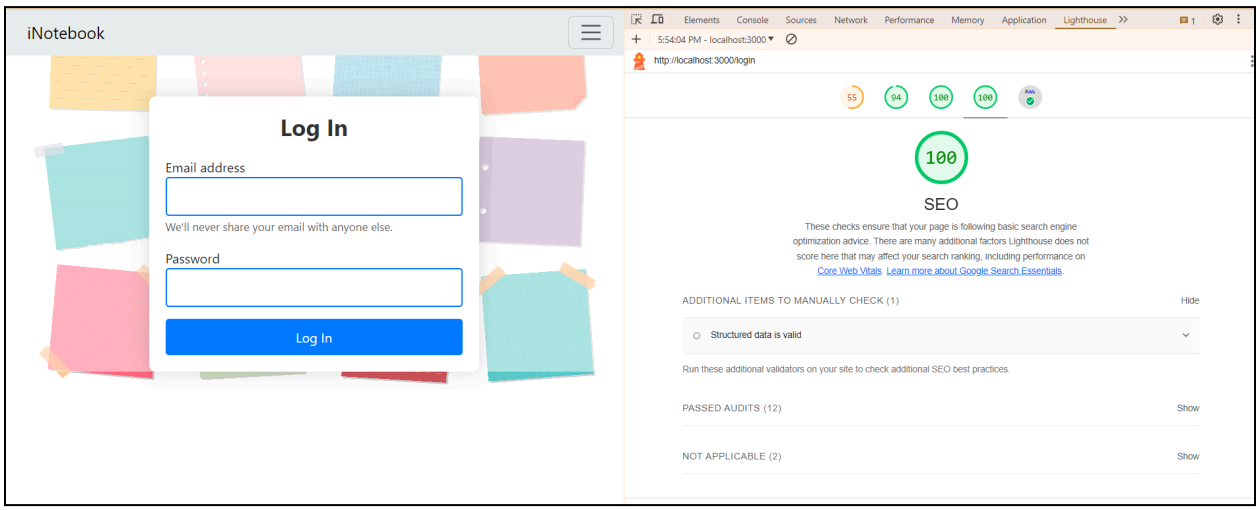


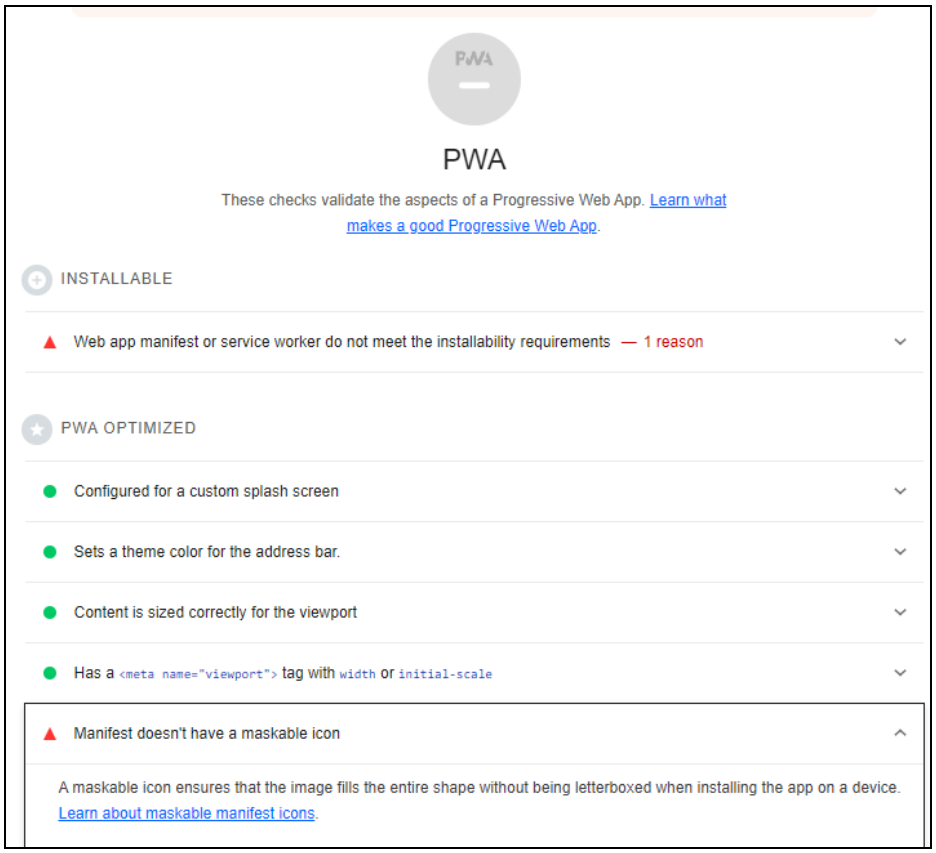



**Project Title: Flutter:Swiggy Clone/PWA:Todo List** **Roll No. 55**



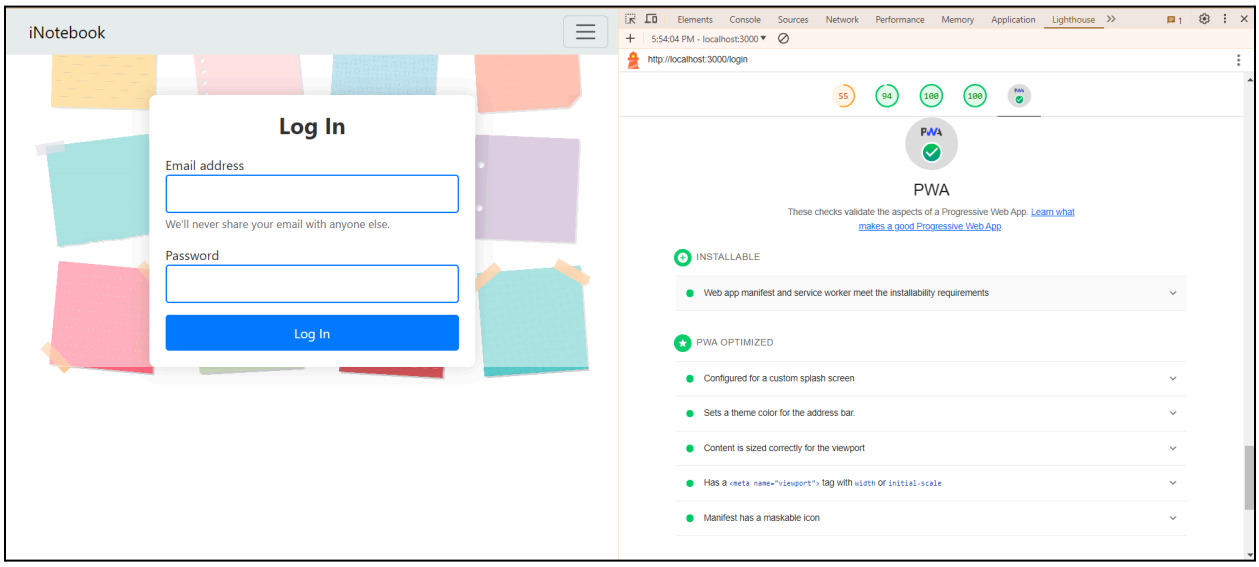






Made the following changes in **manifest.json**

```
{
  "short_name": "React App",
  "name": "Create React App Sample",
  "icons": [
    {
      "src": "favicon.ico",
      "sizes": "64x64 32x32 24x24 16x16",
      "type": "image/x-icon"
    },
    {
      "src": "logo192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
    {
      "src": "logo512.png",
      "type": "image/png",
      "sizes": "512x512",
      "purpose":"maskable"
    },
```

```
    {
      "src": "download.png",
      "type": "image/png",
      "sizes": "144x144",
      "purpose": "any"
    }
  ],
  "start_url": ".",
  "display": "standalone",
  "theme_color": "#000000",
  "background_color": "#ffffff"
}
```




**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.<br>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.<br>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.<br>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. | 55 |
| Name | Mohd Sufiyan Siddiqui |
| Class | D15A |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO1: Understand cross platform mobile application development using Flutter framework<br>LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation<br>LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS |
| Grade: | 05 |

Mohd Sufiyan Siddiqui
D15A/55

Mohd

ASSIGNMENT - 01

Q1 Flutter Overview :- Explain the key features and advantages of using flutter for mobile app development. Discuss how the flutter framework differs from traditional approaches and why it has gained popularity in the developer community.

→ Key features :-

① Single Codebase, Multiple Platforms :-
- Allows developer to write code once and deploy it on multiple platforms including iOs, Android, web and desktop.

② Hot Reload :-
- It enables developers to see the impact of code changes in real-time without restarting the entire application.

③ Expressive and flexible UI :-
- Uses a widget-based architecture to build UIs, providing a rich set of predefigned Material Design and Cupertino widget.

④ High performance :-
- It compiles to native ARM code, resulting in high-performance apps.

It is different from traditional approaches due to all features stated above. which has resulted in higher performance, enabling smooth animations and responsive interfaces

-• It has gained popularity among developer community for various reasons :-

Teacher's Sign.: _____

① Productivity and time savings.
② Consistent UI Across Platforms.
③ Growing Ecosystem.
④ Strong community support.
⑤ High Performance
⑥ Google Backing etc.

2.] Widget Tree and Composition:

→ The widget tree in flutter represents the hierarchical structure of widgets that compose the user interface of a flutter application. Everything in flutter is a widget, whether it's a simple text element or a complex layout. The widget tree is a fundamental concept that facilitates the construction of UI interfaces by nesting and combining various widgets.

The widget tree is structured in a hierarchical manner, with a single root widget at the top.
- Each widget in the tree can have child widgets, forming a parent-child relationship.
- Developers compose complex UI, by nesting and combining various widgets.
- Widgets can be reused, promoting a modular and maintainable code structure.

Teacher's Sign.:

Commonly Used widgets:-

- ① Container: A box model that can contain other widgets. It provides properties for customization like padding, margin, decoration, etc.
- ② Column & Row:- Layout widgets used to arrange children vertically or horizontally.
- ③ ListView:- A scrollable list of widgets that can display a large number of children.
- ④ Stack:- Widget overlays its children widget.
- ⑤ Text:- Displays a text string, etc.

3] State Management in flutter:

→ State management is a crucial aspect of flutter app development, determining how and when the UI updates in response to changes in the application's state. Flutter provides various approaches to manage state, each suitable for different scenarios.

- Some common state management techniques.

① Local State (setState):-

- Overview:-
  - flutter's simplest form of state management.
  - Used for managing state within a single widget.

Teacher's Sign.:

How it works :-
- The `setState` function is used to trigger a rebuild of the widget tree.
- The updated state is typically stored within the widget's stateful class.

② Provider :
- Third party package for simple and scalable state management.
- Implements the provider pattern.

How it works :-
- Uses the `ChangeNotifier` class to create a reactive model.
- Listens to changes and updates widgets accordingly.

③ Riverpod Package :
- Third Party package focused on simplicity and scalability.
- Provides providers for managing state.

How it works :
- Uses providers to expose and consume state.
- Supports both local & global state management.

Q) Firebase Integration in Flutter:-

→ Integrating firebase with flutter involves a series of steps to enable features such as authentication, real-time database, cloud storage, and more.

Step 1:- Set Up a Firebase Project:-

1. Create a Firebase Project.
2. Add an app to your Project.
3. Download and Add config file.
4. 

Step 2:- Configure flutter Project:-

1. Add dependencies:
   - Open pubspec.yaml

```
dependencies:
    flutter:
        sdk: flutter
    firebase_core: ^1.10.6
    firebase_auth: ^4.5.0
```

   - Run 'flutter pub get' to install

2. Initialize firebase.
3. Use Firebase services:
   - Authentication (Firebase Auth).
   - Cloud fire store

- Benefits of firebase integration.
  1. Real time Data Synchronization.

Teacher's Sign.:

② Authentication and User Management
③ Cloud Firestore NoSQL Database.
④ Firebase Cloud Functions.
⑤ Firebase Analytics, etc.

Commonly Used Firebase Services in Flutter.

① Firebase Authentication:
- Enables user authentication with various providers (email/password, Google, Facebook, etc.)
- Allows secure sign-in, sign-up, and password recovery.

② Cloud Firestore :-
- A NoSQL cloud database for storing and syncing data.

③ Firebase Cloud Functions:
④ Firebase Storage
⑤ Firebase Analytics.

Firebase achieves data synchronization through the use of real-time database like Cloud Firestore and the Realtime Database.

# 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<br>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.<br>3. Describe the lifecycle of Service Workers, including registration, installation, and activation phases.<br>4. Explain the use of IndexedDB in the Service Worker for data storage. |
| Roll No. | 55 |
| Name | Mohd Sufiyan Siddiqui |
| Class | D15A |
| Subject | MAD & PWA Lab |
| Lab Outcome | LO4:Understand various PWA frameworks and their requirements<br>LO5: Design and Develop a responsive User Interface by applying PWA Design techniques<br>LO6:Develop and Analyze PWA Features and deploy it over app hosting solutions |
| Grade: | 04 |

Mohd Sufiyan Siddiqui
D15A/55

Assignment - 2

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.

→ A progressive web App (PWA) is a type of web application that uses modern web technologies ~~that uses~~ to provide a native app-like experience to users. They are designed to work on any platform that uses a standards-compliant browser, such as desktop and mobile devices.

- key characteristics that differentiate PWAs.

(i) Cross-platform compatibility:-
PWAs work across different devices and platforms

(ii) Responsive design:-
It is built with responsive design principle.

(iii) Offline functionality:- It can work offline or with poor internet connection.

(iv) Applike experience:- It offer features such as push notification, homescreen installation, full screen mode making them feel like native app.

(v) Fast Performance

(vi) Discoverability:- It is discoverable through search engines and can be shared easily.

Overall, PWAs combine the best feature of web and mobile apps, offering a compelling alternative to traditional native app development.

Sham Sheyc
-4

Q.2 Define responsive web design and explain its importance in the context of PWA. Compare and contrast responsive, fluid & adaptive web design approach.

→ Responsive web design is an approach to web design that aims to create web pages that adjust and respond to the user's device and screen size. It uses a combination of flexible grids and layouts, images, & CSS media queries to ensure that a website looks and function optimally on any device, whether its a desktop, tablet or smartphone.

· Responsive v/s fluid v/s adaptive.

| Feature | Responsive | Fluid | Adaptive. |
|---|---|---|---|
| ① Layout | Flexibility, adapt to any screen size | continuously adjust based on relative units | uses multiple fixed width layouts. |
| ② Development Effort | Moderate | Less effort for basic layout | More effort to create multiple layout. |
| ③ Control | Good control over overall layout | less control over element appearance on extreme sizes | High control over layout for each category |
| ④ Suitability for PWAs | Ideal foundation due to its versatility | Can be used but may require adjustment for optimal experience | Less common for PWA's due to development overhead. |

Q.3 Describe the life cycle of service workers including registeration, installation and activation phase.

→ The life cycle of service workers involves several key phases.

1. Registration :- Developers register a service worker script in the main JavaScript file of the web application using the 'navigator.serviceworker.register()' method. This tells the browser to install and service worker for the specified scope.

2. Installation : During the installation phase, the browser downloads the service worker script specified in the registeration step. The service worker is then installed in the background, but it doesnot yet control any pages or intercept network request.

3. Activation :- After the service worker is installed, it goes through the activation phase. During activation, the service worker can start controlling pages & intercepting network requests.

Throughout its life cycle, a service worker operates independently of any specific web page, making it a powerful tool for building offline capable, resilient web applications with enhanced performance and user experience.

Q.4 Explain the use of indexedDB in the serviceworker for data storage.

→ IndexedDB is a low-level API for client-side storage of significant amounts of structured data, including files/blobs.

Here it is used in a service worker as

1.] Persistent Storage :- IndexedDB provides persistent storage, meaning data stored in IndexedDB persists even when the browser is closed or the device is restarted.

2.] Asynchronous Operation :- IndexedDB operates asynchronously, allowing the service worker to perform database operations without blocking the main thread.

3.] Key value storage :- Stores data in key-value pairs but also supports more complex data structures like object and arrays.

4.] Querying and Indexing :- IndexedDB supports indexing enabling efficient querying of stored data.

5.] Data Transactions :- IndexedDB transactions ensure data integrity by providing atomicity, consistency, isolation & durability (ACID) properties.

Overall, IndexedDB in service workers enables enhanced performance and deliver seamless user experience.