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

Lab Assistant

Subject Teacher

Mrs. Kajal Joseph

Principal

Head of Department

Dr. Mrs. Shalu Chopra

Name of the Course : MAD & PWA Lab

Course Code : ITL604

Project Title: Flutter: Food panda Clone/PWA:To-Do-List website **Roll No. 47**

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: Food panda Clone/PWA:To-Do-List website **Roll No. 47**

Lab Objectives:

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

Lab Outcomes:

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

Index

Sr. No	Experiment Title	LO	DOP	DOS	Grade
1.	To install and configure the Flutter Environment	LO1	19/01	26/01	15
2.	To design Flutter UI by including common widgets.	LO2	26/01	02/02	14
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	15/03	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	31/03	15
10.	To study and implement deployment of Ecommerce PWA to GitHub Pages.	LO5	29/03	31/03	15
11.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.	LO6	29/03	31/03	15
12.	Assignment-1	LO1,LO2, LO3	28/01	05/02	5
13.	Assignment-2	LO4,LO5, LO6	14/03	21/03	4

MAD & PWA Lab
Journal

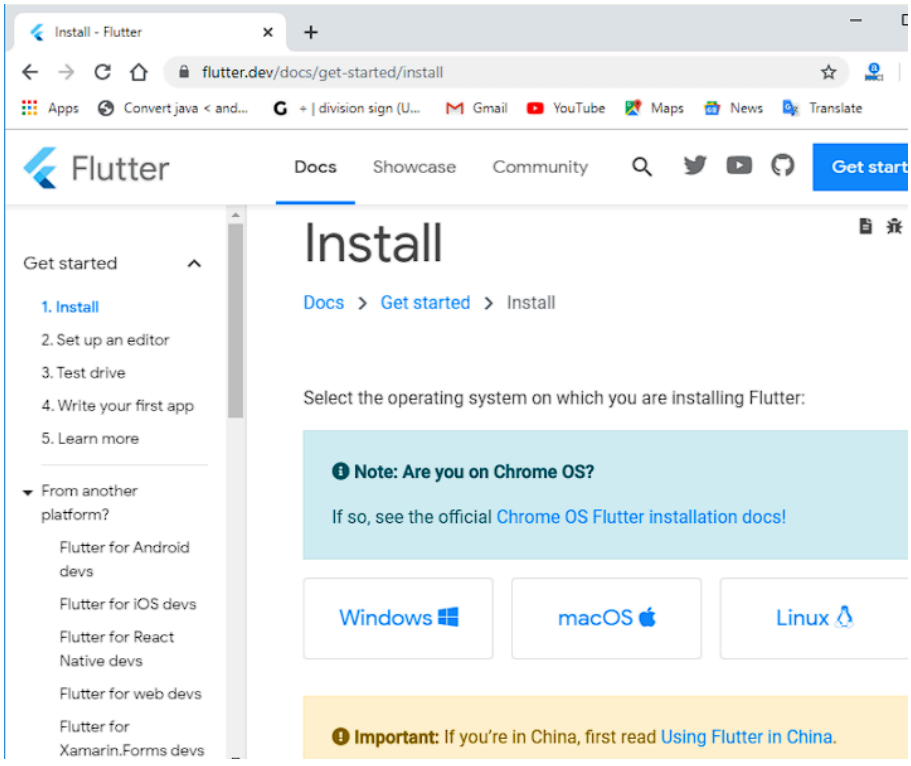
Experiment No.	01
Experiment Title.	To install and configure the Flutter Environment
Roll No.	47
Name	PRANAV SANDEEP RAIKAR
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework
Grade:	15

Project Title: Flutter: Food panda Clone/PWA:To-Do-List website **Roll No. 47**

PRANAV RAIKAR
D15A 47
Aim: To install and configure the Flutter Environment

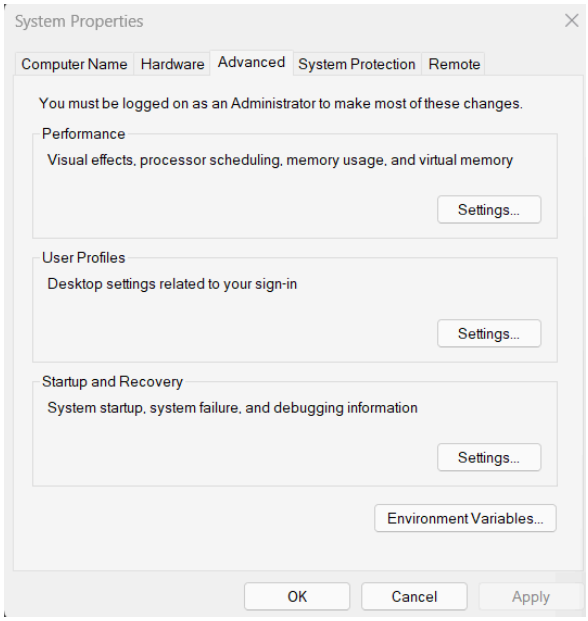
Theory: Flutter is an open source framework by Google for building beautiful, natively compiled, multi-platform applications from a single codebase. Fast: Flutter code compiles to ARM or Intel machine code as well as JavaScript, for fast performance on any device. Productive: Build and iterate quickly with Hot Reload. Update code and see changes almost instantly, without losing state. Flexible: Control every pixel to create customized, adaptive designs that look and feel great on any screen. Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It is available for download on Windows, macOS and Linux based operating systems. It is a replacement for the Eclipse Android Development Tools (E-ADT) as the primary IDE for native Android application development.

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.

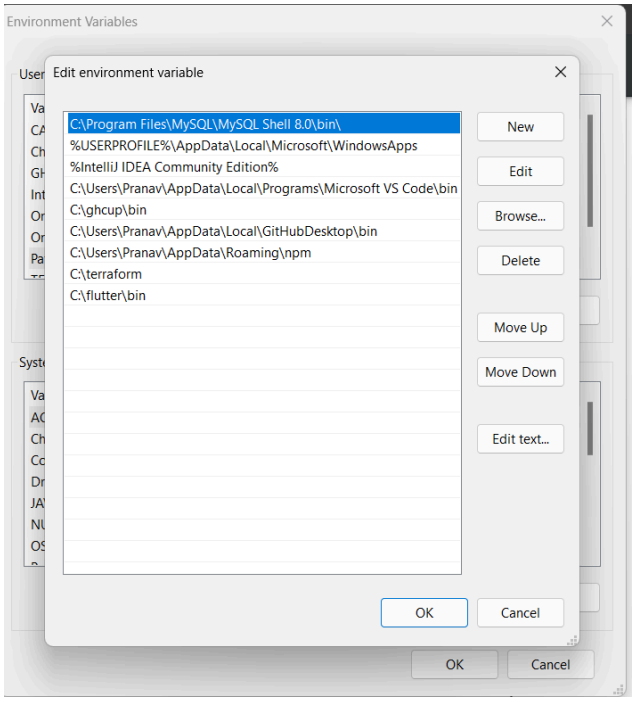


Step 2: Next, to download the latest Flutter SDK, click on the Windows icon. Here, you will find the download link for SDK.

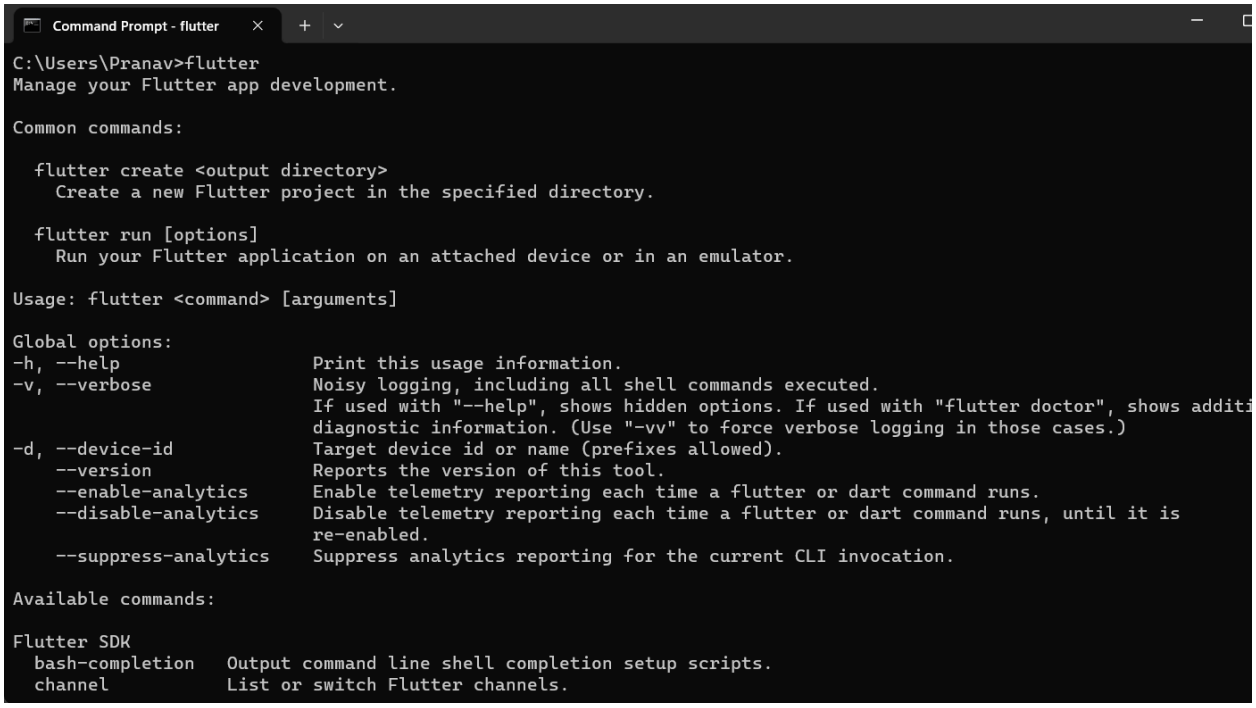
Step 3: When your download is complete, extract the zip file and place it in the desired installation folder or location, for example, C: /Flutter.
Step 4: To run the Flutter command in regular windows console, you need to update the system path to include the flutter bin directory. The following steps 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.

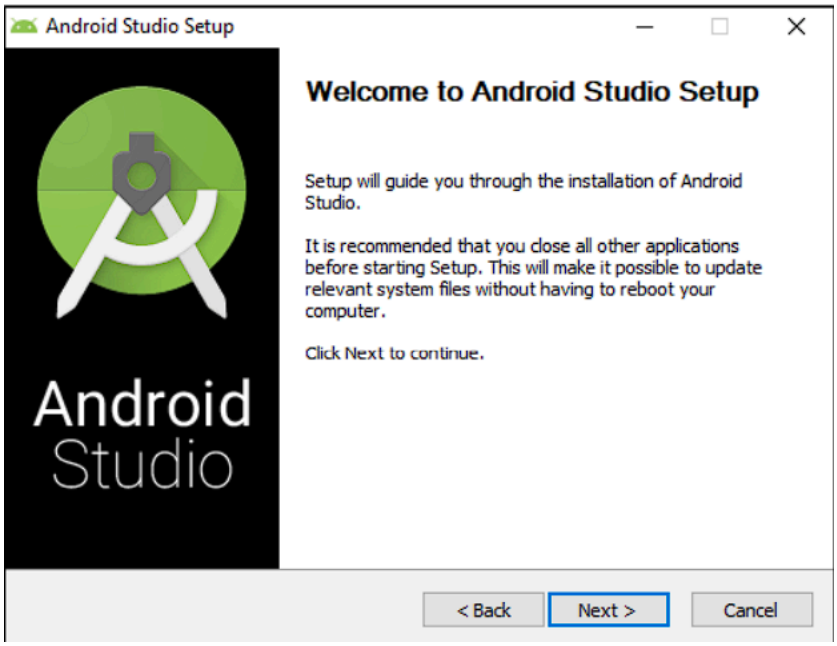


Step 6: When you run the above command, it will analyze the system and show its report, as shown in the below image. Here, you will find the details of all missing tools, which required to run Flutter as well as the development tools that are available but not connected with the device.

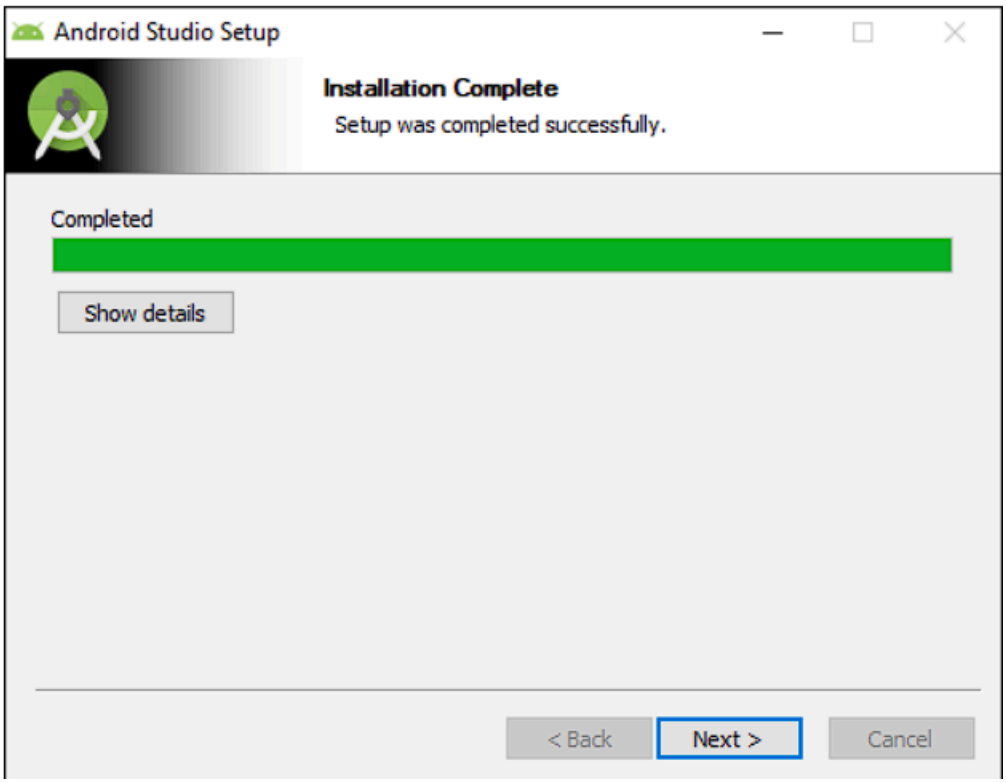
Step 7: Install the Android SDK. If the flutter doctor command does not find the Android SDK tool in your system, then you need first to install the Android Studio IDE. To install Android Studio IDE, do the following steps.

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

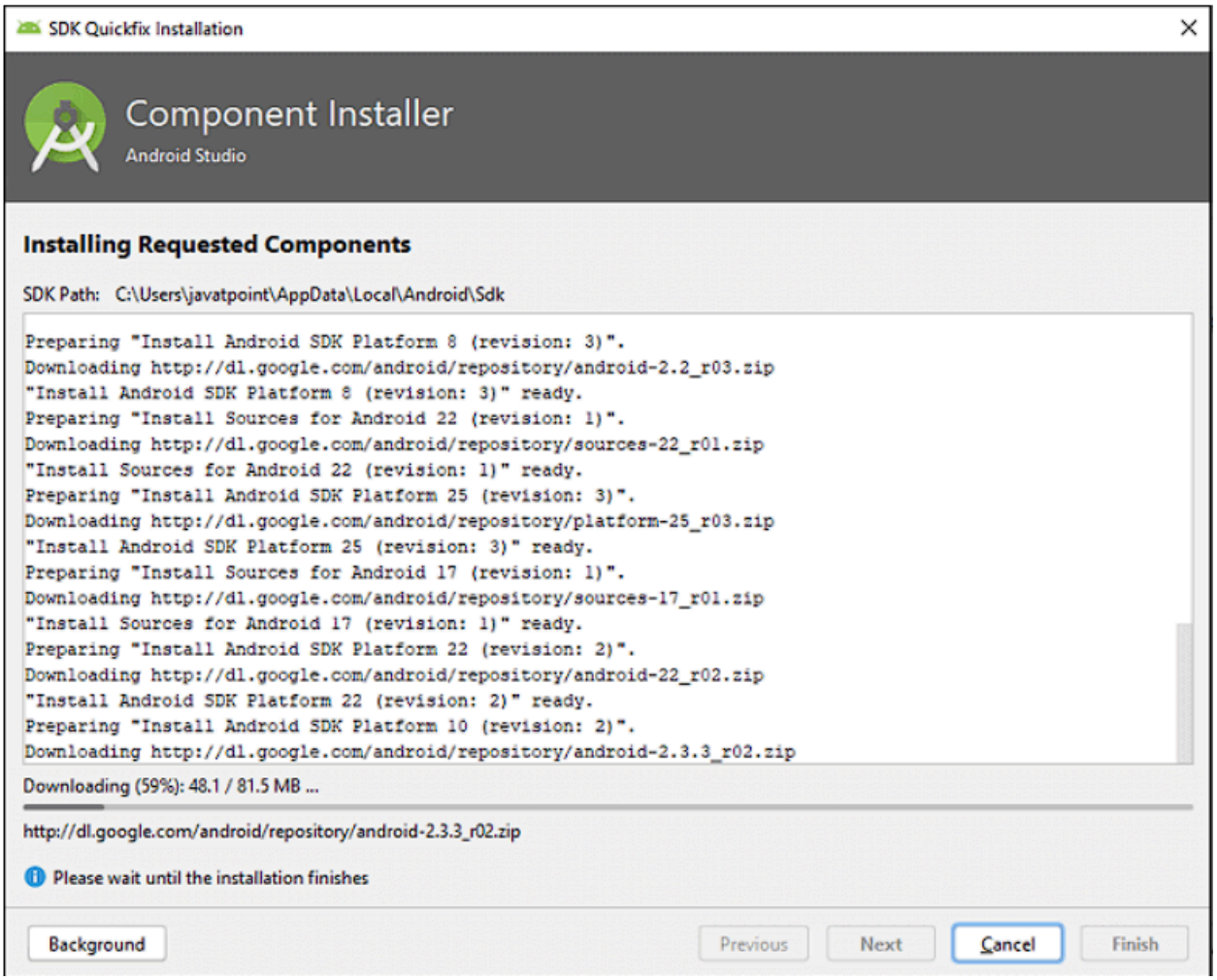
Step 7.2: When the download is complete, open the .exe file and run it. You will get the following dialog box.



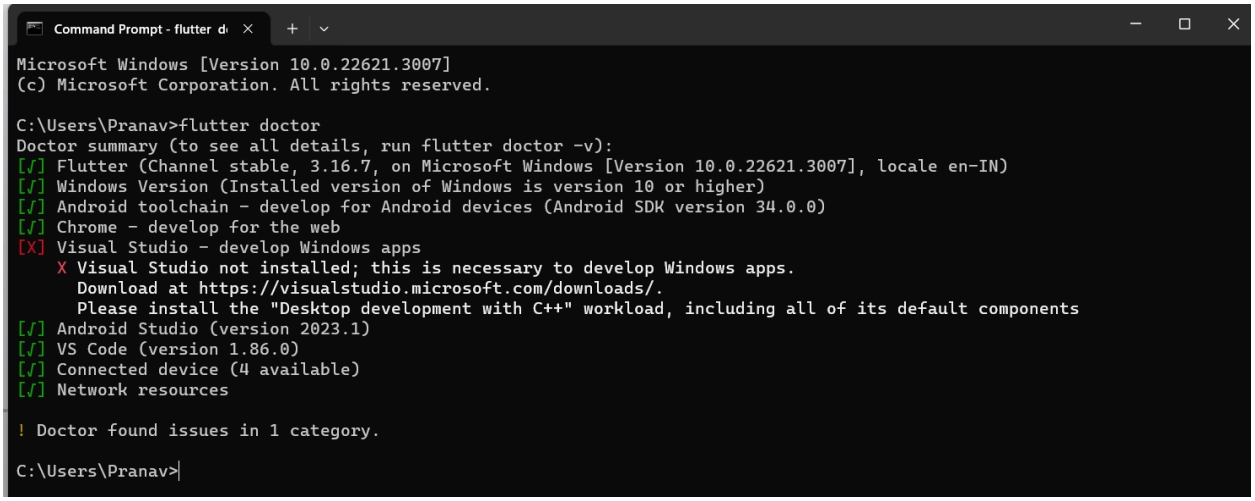
Step 7.3: Follow the steps of the installation wizard. Once the installation wizard completes, you will get the following screen.



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.



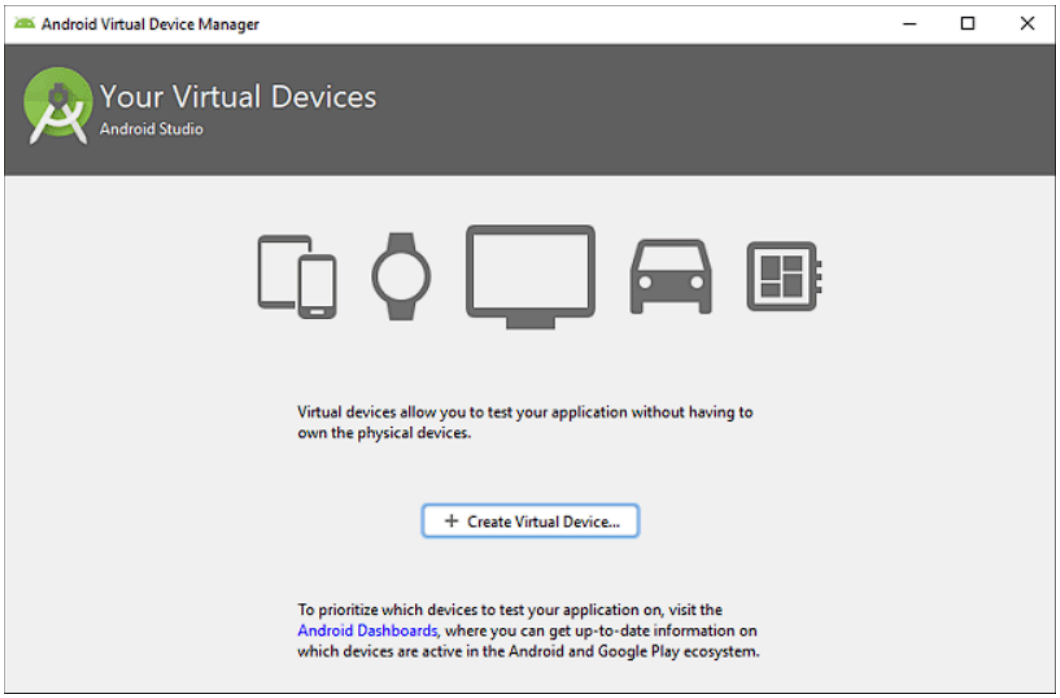
Step 7.5 run the $ flutter doctor command and Run flutter doctor --android-licenses command.



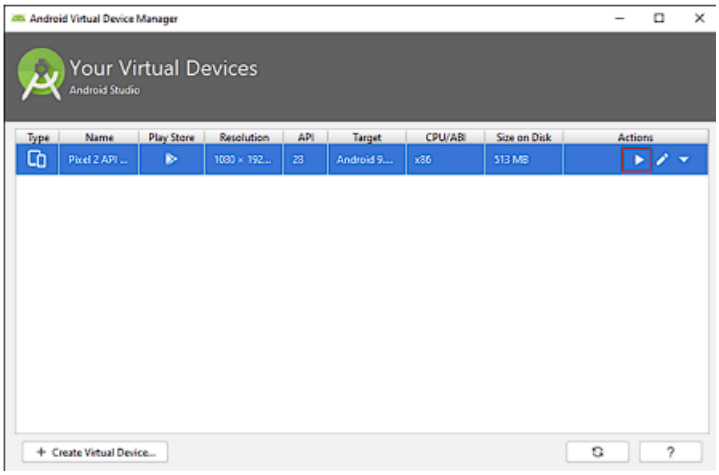
Step 8: Next, you need to set up an Android emulator. It is responsible for running and testing the Flutter application.

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.



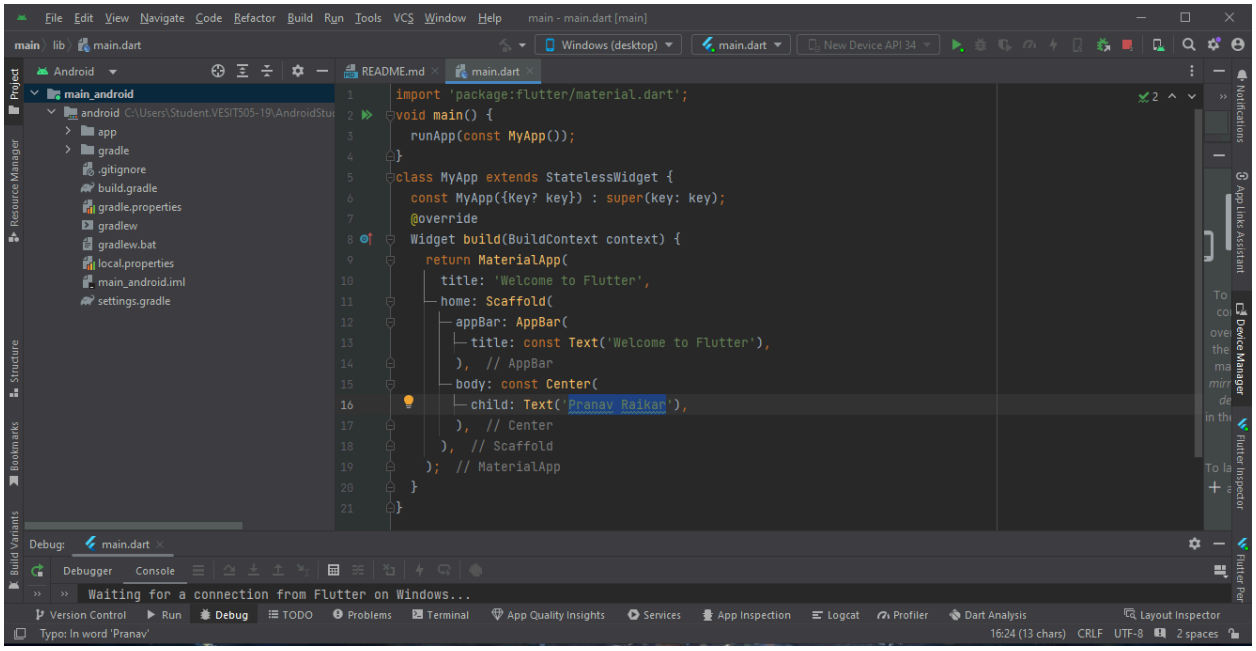
Step 8.5: Last, click on the icon pointed into the red color rectangle. The Android emulator displayed as below screen.

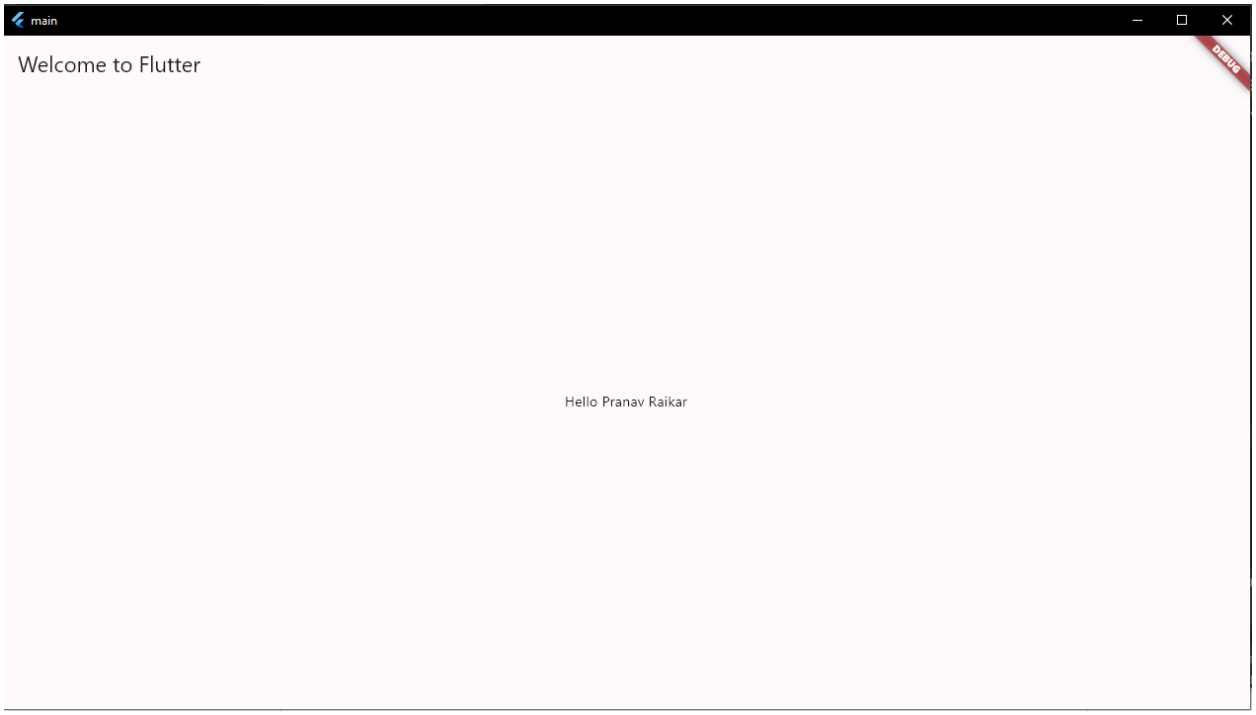
Step 9: Now, install Flutter and Dart plugin for building Flutter application in Android Studio. These plugins provide a template to create a Flutter application, give an option to run and debug Flutter application in the Android Studio itself. Do the following steps to install these plugins.

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.

Project Title: Flutter: Food panda Clone/PWA:To-Do-List website **Roll No. 47**





MAD & PWA Lab

Journal

Experiment No.	02
Experiment Title.	To design Flutter UI by including common widgets.
Roll No.	47
Name	PRANAV SANDEEP RAIKAR
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	14

PRANAV RAIKAR
D15A 47
Experiment 2
AIM: To design Flutter UI by including common widgets.

Theory:

In summary, Flutter widgets are fundamental components in constructing the user interface of a Flutter application. They can be broadly categorized into two types: `StatelessWidget` representing immutable parts of the UI and `StatefulWidget` representing mutable components that can change over time.

Some key Flutter widgets include:

1. Scaffold: The basic structure for a Flutter app, providing layout elements such as AppBar, BottomNavigationBar, and a body for main content.

2. Container: A versatile box model used for layout, padding, margin, decoration, and constraints, capable of containing other widgets.

3. Row & Column: Widgets for arranging child widgets horizontally (Row) or vertically (Column), essential for creating flexible and responsive layouts.

4. Text: Used for displaying text on the screen with support for various styling options like font size, color, and alignment.

5. TextField: Captures user input, such as text, numbers, or passwords, with the `onChanged` property for dynamic updates based on user input.

6. Buttons: Various button widgets like `ElevatedButton` or `TextButton` trigger actions when pressed, providing a means for user interaction.

7. Forms: The `Form` widget manages a group of `TextFormField` widgets, facilitating input validation and submission.

8. Icons: The `Icon` widget displays icons from libraries, enhancing visual elements and conveying meaning through symbols.

Key Design Principles highlighted include:

- Consistency: Common widget usage fosters a consistent design language throughout the app.

- Responsive Layouts: Widgets like `Row` and `Column` aid in creating responsive and flexible layouts, adapting to different screen sizes.

- User Input Handling: `TextField` and `Form` widgets facilitate proper handling, ensuring data integrity and validation.

- Interactive Elements: Buttons and icons contribute to interactivity and user engagement within the app.

- Visual Styling: The `Container` widget and styling properties of other widgets allow for visual customization and theming.

Common widgets is used for different type of fonts:

```
// ignore_for_file: prefer_const_constructors, non_constant_identifier_names

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

```
class AppWidget {
  static TextStyle boldTextFieldStyle() {
    return TextStyle(
        color: Color.fromARGB(255, 50, 32, 32),
        fontSize: 20.0,
        fontWeight: FontWeight.bold,
        fontFamily: 'Poppins');
  }

  static TextStyle HeadlineTextFieldStyle() {
    return TextStyle(
        color: Color.fromARGB(255, 50, 32, 32),
        fontSize: 24.0,
        fontWeight: FontWeight.bold,
        fontFamily: 'Poppins');
  }

  static TextStyle LightTextFieldStyle() {
    return TextStyle(
        color: Color.fromARGB(255, 50, 32, 32),
        fontSize: 15.0,
        fontWeight: FontWeight.w500,
        fontFamily: 'Poppins');
  }

  static TextStyle semiBoldTextFieldStyle() {
    return TextStyle(
        color: Color.fromARGB(255, 50, 32, 32),
        fontSize: 15.0,
        fontWeight: FontWeight.bold,
        fontFamily: 'Poppins');
  }
}
```

The above code is used in following below code of home.dart

```
import 'package:flutter/material.dart';
import 'package:food_panda/pages/details.dart';
import 'package:food_panda/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: SingleChildScrollView(
```

```
scrollDirection: Axis.vertical,
child: Container(
  margin: const EdgeInsets.only(top: 50.0, left: 20.0),
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Row(
        mainAxisAlignment: MainAxisAlignment.spaceBetween,
        children: [
          Text("Hello Pranav,", style: AppWidget.boldTextFieldStyle()),
          Container(
            margin: const EdgeInsets.only(right: 20.0),
            padding: const EdgeInsets.all(3),
            decoration: BoxDecoration(
                color: Colors.black,
                borderRadius: BorderRadius.circular(8)),
            child: const Icon(
              Icons.shopping_cart_outlined,
              color: Colors.white,
            ),
          )
        ],
      ),
      const SizedBox(
        height: 20.0,
      ),
      Text("Delicious Food", style: AppWidget.HeadlineTextFieldStyle()),
      Text("Discover and Get Great Food",
          style: AppWidget.LightTextFieldStyle()),
      const SizedBox(
        height: 20.0,
      ),
      Container(
          margin: const EdgeInsets.only(right: 20.0),
          child: showItem()),
      const SizedBox(
        height: 30.0,
      ),
      SingleChildScrollView(
        scrollDirection: Axis.horizontal,
        child: Row(
          children: [
            GestureDetector(
              onTap: () {
                Navigator.push(context,
                    MaterialPageRoute(builder: (context) => Details()));
              },
              child: Container(
                margin: const EdgeInsets.all(4),
                child: Material(
                  elevation: 5.0,
                  borderRadius: BorderRadius.circular(20),
                  child: Container(
                    padding: const EdgeInsets.all(14),
                    child: Column(
                        crossAxisAlignment: CrossAxisAlignment.start,
                        children: [
                          Image.asset(
                            "images/salad2.png",
                            height: 150,
                            width: 150,
```

```
                  fit: BoxFit.cover,
                ),
                Text("Veggie Taco Hash",
                    style:
                        AppWidget.semiBoldTextFieldStyle()),
                const SizedBox(
                  height: 5.0,
                ),
                Text("Fresh and Healthy",
                    style: AppWidget.LightTextFieldStyle()),
                const SizedBox(
                  height: 5.0,
                ),
                Text(
                  "\$25",
                  style: AppWidget.semiBoldTextFieldStyle(),
                )
              ]),
        ),
      ),
    ),
  ),
  const SizedBox(
    width: 15.0,
  ),
  Container(
    margin: const EdgeInsets.all(4),
    child: Material(
      elevation: 5.0,
      borderRadius: BorderRadius.circular(20),
      child: Container(
        padding: const EdgeInsets.all(14),
        child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Image.asset(
                "images/salad4.png",
                height: 150,
                width: 150,
                fit: BoxFit.cover,
              ),
              Text("Mix Veg Salad",
                  style: AppWidget.semiBoldTextFieldStyle()),
              const SizedBox(
                height: 5.0,
              ),
              Text("Spicy with Onion",
                  style: AppWidget.LightTextFieldStyle()),
              const SizedBox(
                height: 5.0,
              ),
              Text(
                "\$28",
                style: AppWidget.semiBoldTextFieldStyle(),
              )
            ]),
        ),
      ),
    ),
  ],
),
```

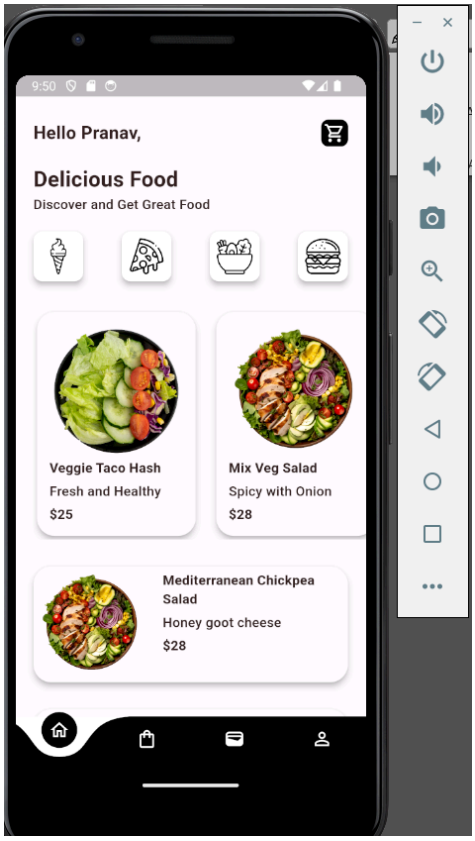
```
),
const SizedBox(
  height: 30.0,
),
Container(
  margin: const EdgeInsets.only(right: 20.0),
  child: Material(
    elevation: 5.0,
    borderRadius: BorderRadius.circular(20),
    child: Container(
      padding: const EdgeInsets.all(5),
      child: Row(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Image.asset(
            "images/salad4.png",
            height: 120,
            width: 120,
            fit: BoxFit.cover,
          ),
          const SizedBox(
            width: 20.0,
          ),
          Column(
            children: [
              Container(
                  width: MediaQuery.of(context).size.width / 2,
                  child: Text(
                    "Mediterranean Chickpea Salad",
                    style: AppWidget.semiBoldTextFieldStyle(),
                  )),
              const SizedBox(
                height: 5.0,
              ),
              Container(
                  width: MediaQuery.of(context).size.width / 2,
                  child: Text(
                    "Honey goot cheese",
                    style: AppWidget.LightTextFieldStyle(),
                  )),
              const SizedBox(
                height: 5.0,
              ),
              Container(
                  width: MediaQuery.of(context).size.width / 2,
                  child: Text(
                    "\$28",
                    style: AppWidget.semiBoldTextFieldStyle(),
                  ))
            ],
          )
        ],
      ),
    ),
  ),
),
const SizedBox(
  height: 30.0,
),
Container(
  margin: const EdgeInsets.only(right: 20.0),
```

```
            child: Material(
              elevation: 5.0,
              borderRadius: BorderRadius.circular(20),
              child: Container(
                padding: const EdgeInsets.all(5),
                child: Row(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    Image.asset(
                      "images/salad2.png",
                      height: 120,
                      width: 120,
                      fit: BoxFit.cover,
                    ),
                    const SizedBox(
                      width: 20.0,
                    ),
                    Column(
                      children: [
                        Container(
                            width: MediaQuery.of(context).size.width / 2,
                            child: Text(
                              "Veggie Taco Hash",
                              style: AppWidget.semiBoldTextFieldStyle(),
                            )),
                        const SizedBox(
                          height: 5.0,
                        ),
                        Container(
                            width: MediaQuery.of(context).size.width / 2,
                            child: Text(
                              "Honey goot cheese",
                              style: AppWidget.LightTextFieldStyle(),
                            )),
                        const SizedBox(
                          height: 5.0,
                        ),
                        Container(
                            width: MediaQuery.of(context).size.width / 2,
                            child: Text(
                              "\$28",
                              style: AppWidget.semiBoldTextFieldStyle(),
                            ))
                      ],
                    )
                  ],
                ),
              ),
            ),
          ),
        ],
      ),
    ),
  );
}

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: const EdgeInsets.all(8),
      child: Image.asset(
        "images/ice-cream.png",
        height: 40,
        width: 40,
        fit: BoxFit.cover,
        color: icecream ? 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: const EdgeInsets.all(8),
      child: Image.asset(
        "images/pizza.png",
        height: 40,
        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: const EdgeInsets.all(8),
            child: Image.asset(
              "images/salad.png",
              height: 40,
              width: 40,
              fit: BoxFit.cover,
              color: salad ? 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: const EdgeInsets.all(8),
            child: Image.asset(
              "images/burger.png",
              height: 40,
              width: 40,
              fit: BoxFit.cover,
              color: burger ? Colors.white : Colors.black,
            ),
          ),
        ),
      ),
    ],
  );
 }
}
```

Project Title: Flutter: Food panda Clone/PWA:To-Do-List website **Roll No. 47**



Conclusion :Thus I learnt to create and use common widgets

MAD & PWA Lab
Journal

Experiment No.	03
Experiment Title.	To include icons, images, fonts in Flutter app
Roll No.	47
Name	PRANAV SANDEEP RAIKAR
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

PRANAV RAIKAR
D15A 47

Experiment 3:

AIM: To include images and fonts in flutter app.

Theory:

Theory: Certainly! Here's a simplified process for adding images in Flutter:

1. Import Libraries:
Ensure that you have the necessary libraries imported in your Dart file. For images, you'll typically use `dart:ui` and other relevant Flutter packages.
2. Adding Local Images:
- Place your local images in the `assets` folder.
- Declare the images in the `pubspec.yaml` file.
3. Adding Network Images:
- Use the `Image.network` widget for displaying images from the internet.
4. Image Widget:
- Create an `Image` widget and provide it with an `ImageProvider`.
- Use `AssetImage` for local images and `NetworkImage` for network images.
5. ImageProvider:
- Understand that `AssetImage` and `NetworkImage` are subclasses of the `ImageProvider` class.
- You can create custom `ImageProvider` if needed.
6. CachedNetworkImage (Optional):
- If you want to cache network images, consider using the `cached_network_image` package.
7. Image Loading and Error Handling:
- Customize the loading and error behavior using `loadingBuilder` and `errorBuilder` properties of the `Image` widget or other relevant widgets.

Remember, the actual implementation details might vary based on your specific use case and the packages you choose to use. The key is to understand the concepts of working with local and network images and the various widgets and packages available in Flutter for handling images.

Code:

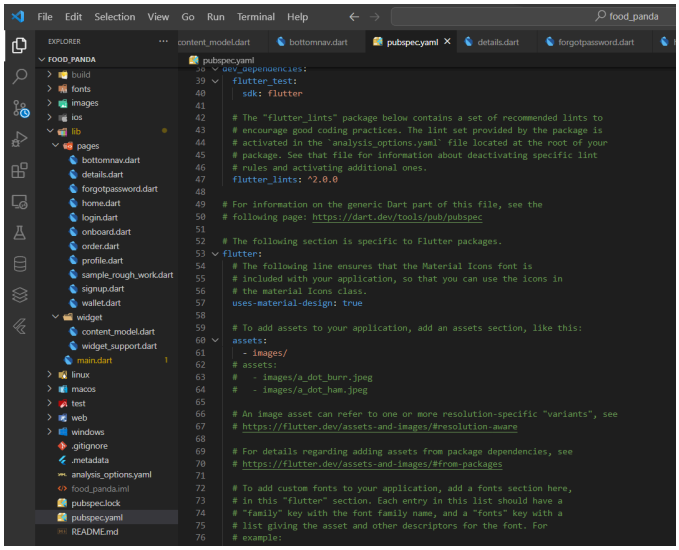
```
import 'package:flutter/material.dart';
import 'package:food_panda/widget/widget_support.dart';

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

  @override
  State<Details> createState() => _DetailsState();
}

class _DetailsState extends State<Details> {
  int a = 1;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Container(
        margin: EdgeInsets.only(top: 50.0, left: 20.0, right: 20.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
```

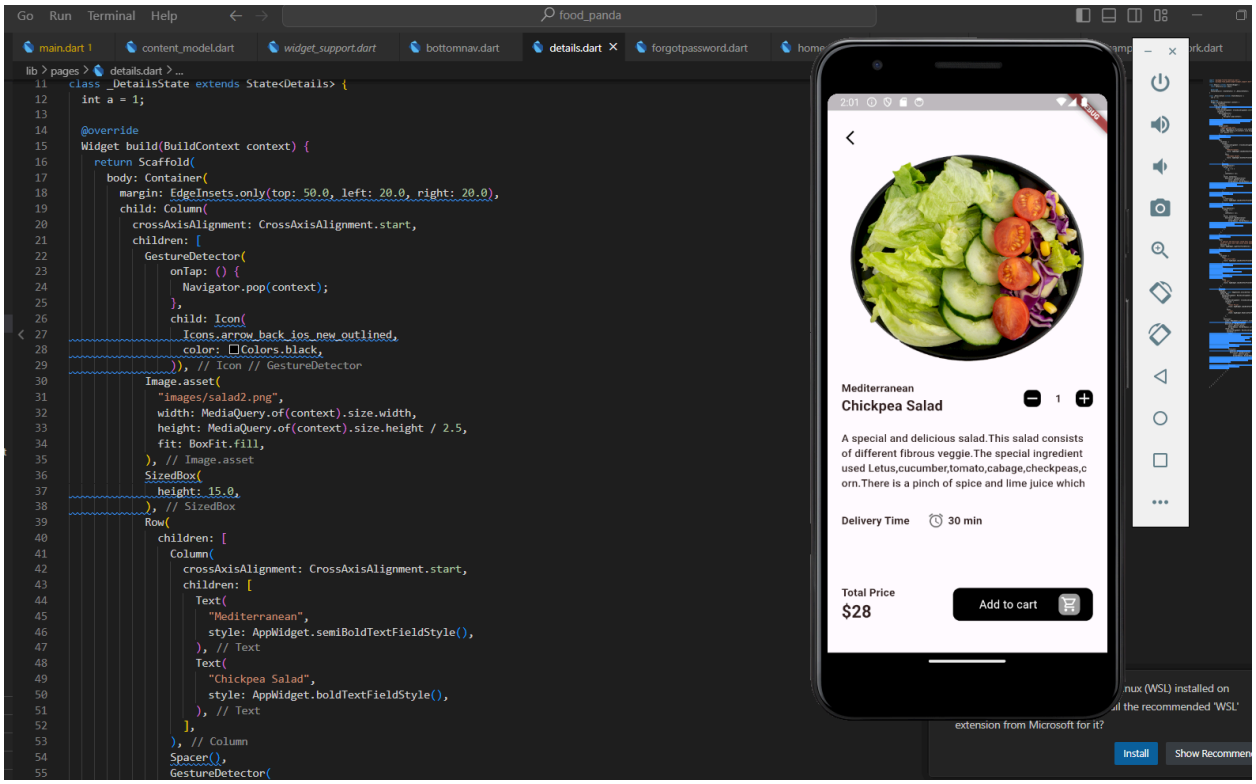
```
GestureDetector(
    onTap: () {
      Navigator.pop(context);
    },
    child: Icon(
      Icons.arrow_back_ios_new_outlined,
      color: Colors.black,
    )),
Image.asset(
  "images/salad2.png",
  width: MediaQuery.of(context).size.width,
  height: MediaQuery.of(context).size.height / 2.5,
  fit: BoxFit.fill,
),
SizedBox(
  height: 15.0,
),
Row(
  children: [
    Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Text(
          "Mediterranean",
          style: AppWidget.semiBoldTextFieldStyle(),
        ),
        Text(
          "Chickpea Salad",
          style: AppWidget.boldTextFieldStyle(),
        ),
      ],
    ),
    Spacer(),
    GestureDetector(
      onTap: () {
        if (a > 1) {
          --a;
        }
        setState(() {});
      },
      child: Container(
        decoration: BoxDecoration(
            color: Colors.black,
            borderRadius: BorderRadius.circular(8)),
        child: Icon(
          Icons.remove,
          color: Colors.white,
        ),
      ),
    ),
    SizedBox(
      width: 20.0,
    ),
    Text(
      a.toString(),
      style: AppWidget.semiBoldTextFieldStyle(),
    ),
    SizedBox(
      width: 20.0,
    ),
    GestureDetector(
```

```
            onTap: () {
              ++a;
              setState(() {});
            },
            child: Container(
              decoration: BoxDecoration(
                  color: Colors.black,
                  borderRadius: BorderRadius.circular(8)),
              child: Icon(
                Icons.add,
                color: Colors.white,
              ),
            ),
          )
        ],
      ),
      SizedBox(
        height: 20.0,
      ),
      Text(
        "A special and delicious salad.This salad consists of different fibrous veggie.The special ingredient used
Letus,cucumber,tomato,cabage,checkpeas,corn.There is a pinch of spice and lime juice which makes the salad more delicious",
        maxLines: 4,
        style: AppWidget.LightTextFieldStyle(),
      ),
      SizedBox(
        height: 30.0,
      ),
      Row(
        children: [
          Text(
            "Delivery Time",
            style: AppWidget.semiBoldTextFieldStyle(),
          ),
          SizedBox(
            width: 25.0,
          ),
          Icon(
            Icons.alarm,
            color: Colors.black54,
          ),
          SizedBox(
            width: 5.0,
          ),
          Text(
            "30 min",
            style: AppWidget.semiBoldTextFieldStyle(),
          )
        ],
      ),
      Spacer(),
      Padding(
        padding: const EdgeInsets.only(bottom: 40.0),
        child: Row(
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: [
            Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Text(
                  "Total Price",
```

```
                    style: AppWidget.semiBoldTextFieldStyle(),
                  ),
                  Text(
                    "\$28",
                    style: AppWidget.HeadlineTextFieldStyle(),
                  )
                ],
              ),
              Container(
                width: MediaQuery.of(context).size.width / 2,
                padding: EdgeInsets.all(8),
                decoration: BoxDecoration(
                    color: Colors.black,
                    borderRadius: BorderRadius.circular(10)),
                child: Row(
                  mainAxisAlignment: MainAxisAlignment.end,
                  children: [
                    Text(
                      "Add to cart",
                      style: TextStyle(
                          color: Colors.white,
                          fontSize: 16.0,
                          fontFamily: 'Poppins'),
                    ),
                    SizedBox(
                      width: 30.0,
                    ),
                    Container(
                      padding: EdgeInsets.all(3),
                      decoration: BoxDecoration(
                          color: Colors.grey,
                          borderRadius: BorderRadius.circular(8)),
                      child: Icon(
                        Icons.shopping_cart_outlined,
                        color: Colors.white,
                      ),
                    ),
                    SizedBox(
                      width: 10.0,
                    ),
                  ],
                ),
              )
            ],
          ),
        )
      ],
    ),
  ),
 );
 }
}
```

Project Title: Flutter: Food panda Clone/PWA:To-Do-List website **Roll No. 47**





In above screenshot I have used Image.asset method to add a image

Conclusion :Thus I learnt to how to add images in our flutter app

MAD & PWA Lab

Journal

Experiment No.	04
Experiment Title.	To create an interactive Form using form widget
Roll No.	47
Name	PRANAV SANDEEP RAIKAR
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

PRANAV RAIKAR
D15A 47
Experiment 4

AIM: To create interactive form using form widget.

Theory: In Flutter, a "Form" is a widget that represents a container for a collection of form fields. It helps manage the state of the form and facilitates the validation and submission of user input. Here are some key concepts and theories about forms in Flutter:

1. Widget Hierarchy:
Forms in Flutter are composed of the `Form` widget, which contains a list of `FormField` widgets. Each `FormField` represents an individual input field like text fields, checkboxes, or dropdowns.

2. Form State:
The `Form` widget maintains the state of the form, including the current values of the form fields and their validation statuses. The form state is automatically managed by Flutter.

3. Validation:
Forms provide built-in validation through the `validator` property of each `FormField`. Validators are functions that determine whether the input is valid. The form's overall validity is determined by the validity of all its fields.

4. Form Submission:
Form submission is typically triggered by a button press. The `onPressed` callback of the button can call the `FormState.save()` method, which invokes the `onSaved` callback for each form field and then calls the `onFormSaved` callback.

5. GlobalKey<FormState>:
To interact with the form state, a `GlobalKey<FormState>` is commonly used. This key allows access to the form state and is used to validate and save the form.

6. Auto-validation:
Flutter provides automatic validation by calling the `validator` function whenever the user input changes. This allows for real-time feedback to the user about the validity of their input.

7. Form Submission Lifecycle:
The form submission process involves validation, saving, and then handling the saved data. Developers can customize this process by providing their own logic within the `onSaved` and `onFormSaved` callbacks.

8.FocusManagement:
Formshandlethefocusof input fields,makingiteasytonavigatethroughtheformusing keyboardinputorprogrammaticallysettingfocusonspecificfields.

9.FormKeyandGlobalKey:
Usinga`GlobalKey<FormState>`allowsformorecontrolovertheform,suchastriggering formvalidationorresettingtheform. It isusuallydefinedasaglobalkeyinthewidget tree.

10.FormPersistence:
Formdatacanbepersistedacrossdifferentscreensorappsessionsbypassingthedata downthewidget treeorusingstatemanagementsolutionslikeProviderorRiverpod.
Formsplayacrucial roleinuserinteraction,datacollection,andvalidationinFlutter

applications,providingastructuredandefficientwaytohandleuserinput.

Code:

Login

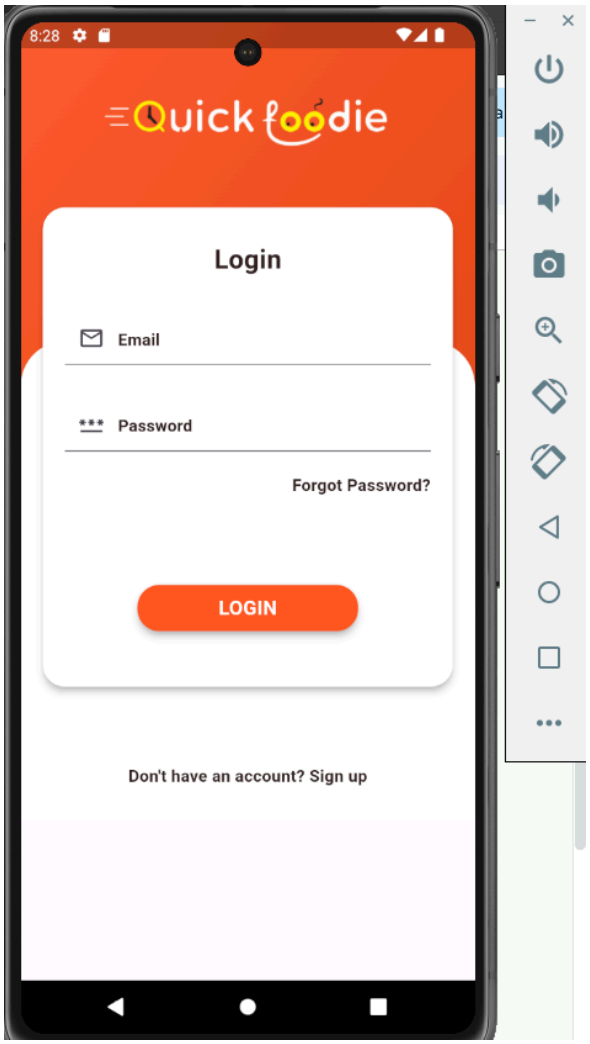
```
// import 'package:firebase_auth/firebase_auth.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:food_panda/pages/bottomnav.dart';
import 'package:food_panda/pages/forgotpassword.dart';
import 'package:food_panda/pages/signup.dart';
import 'package:food_panda/widget/widget_support.dart';
// import 'package:sample_flutter/pages/bottomnav.dart';
// import 'package:sample_flutter/pages/forgotpassword.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: 60.0, 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: 50.0,
              ),
              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: 30.0,
  ),
  Text(
    "Login",
    style: AppWidget.HeadlineTextFieldStyle(),
  ),
  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.semiBoldTextFieldStyle(),
        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.semiBoldTextFieldStyle(),
        prefixIcon: Icon(Icons.password_outlined)),
  ),
  SizedBox(
    height: 20.0,
  ),
  GestureDetector(
    onTap: () {
      Navigator.push(
          context,
          MaterialPageRoute(
              builder: (context) =>
                  ForgotPassword()));
    },
    child: Container(
        alignment: Alignment.topRight,
```

```
                child: Text(
                  "Forgot Password?",
                  style: AppWidget.semiBoldTextFieldStyle(),
                )),
            ),
            SizedBox(
              height: 80.0,
            ),
            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: 70.0,
),
GestureDetector(
    onTap: () {
      Navigator.push(context,
          MaterialPageRoute(builder: (context) => SignUp()));
    },
    child: Text(
      "Don't have an account? Sign up",
```

```
                    style: AppWidget.semiBoldTextFieldStyle(),
                  ))
            ],
          ),
        )
      ],
    ),
  ),
 );
 }
}
```



Signup:

Code:

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

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

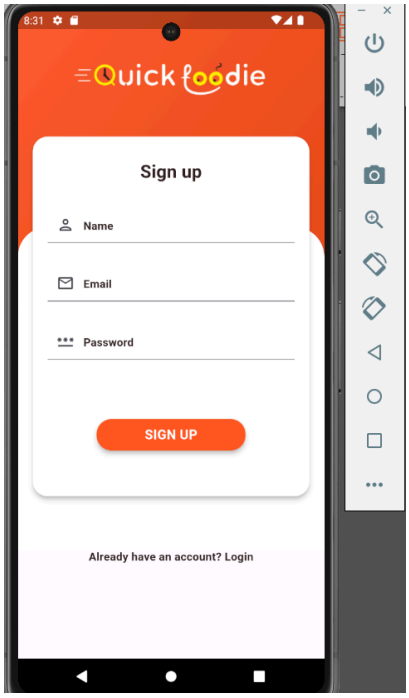
```
import 'package:food_panda/pages/login.dart';
import 'package:food_panda/service/database.dart';
import 'package:food_panda/service/shared_pref.dart';
import 'package:food_panda/widget/widget_support.dart';

import 'package:random_string/random_string.dart';
// import 'package:sample_flutter/pages/bottomnav.dart';

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

  @override
  State<SignUp> createState() => _SignUpState();
}

class _SignUpState extends State<SignUp> {
  String email = "", password = "", name = "";

  TextEditingController namecontroller = new TextEditingController();

  TextEditingController passwordcontroller = new TextEditingController();

  TextEditingController mailcontroller = new TextEditingController();

  final _formkey = GlobalKey<FormState>();

  registration() async {
    if (password != null) {
      try {
        UserCredential userCredential = await FirebaseAuth.instance
            .createUserWithEmailAndPassword(email: email, password: password);

        ScaffoldMessenger.of(context).showSnackBar((SnackBar(
            backgroundColor: Colors.redAccent,
            content: Text(
              "Registered Successfully",
              style: TextStyle(fontSize: 20.0),
            ))));
        String Id = randomAlphaNumeric(10);
        Map<String, dynamic> addUserInfo = {
          "Name": namecontroller.text,
          "Email": mailcontroller.text,
          "Wallet": "0",
          "Id": Id,
        };
        await DatabaseMethods().addUserDetail(addUserInfo, Id);
        await SharedPreferenceHelper().saveUserName(namecontroller.text);
        await SharedPreferenceHelper().saveUserEmail(mailcontroller.text);
        await SharedPreferenceHelper().saveUserWallet('0');
        await SharedPreferenceHelper().saveUserId(Id);

        // ignore: use_build_context_synchronously
        Navigator.pushReplacement(
            context, MaterialPageRoute(builder: (context) => BottomNav()));
```

```
    } on FirebaseException catch (e) {
      if (e.code == 'weak-password') {
        ScaffoldMessenger.of(context).showSnackBar(SnackBar(
            backgroundColor: Colors.orangeAccent,
            content: Text(
              "Password Provided is too Weak",
              style: TextStyle(fontSize: 18.0),
            )));
      } else if (e.code == "email-already-in-use") {
        ScaffoldMessenger.of(context).showSnackBar(SnackBar(
            backgroundColor: Colors.orangeAccent,
            content: Text(
              "Account Already exsists",
              style: TextStyle(fontSize: 18.0),
            )));
      }
    }
  }
}

@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: 60.0, 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: 50.0,
),
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 / 1.8,
    decoration: BoxDecoration(
        color: Colors.white,
        borderRadius: BorderRadius.circular(20)),
    child: Form(
      key: _formkey,
      child: Column(
        children: [
          SizedBox(
            height: 30.0,
          ),
          Text(
            "Sign up",
            style: AppWidget.HeadlineTextFieldStyle(),
          ),
          SizedBox(
            height: 30.0,
          ),
          TextFormField(
            controller: namecontroller,
            validator: (value) {
              if (value == null || value.isEmpty) {
                return 'Please Enter Name';
              }
              return null;
            },
            decoration: InputDecoration(
                hintText: 'Name',
                hintStyle: AppWidget.semiBoldTextFieldStyle(),
                prefixIcon: Icon(Icons.person_outlined)),
          ),
          SizedBox(
            height: 30.0,
          ),
          TextFormField(
            controller: mailcontroller,
            validator: (value) {
              if (value == null || value.isEmpty) {
                return 'Please Enter E-mail';
```

```
    }
    return null;
  },
  decoration: InputDecoration(
      hintText: 'Email',
      hintStyle: AppWidget.semiBoldTextFieldStyle(),
      prefixIcon: Icon(Icons.email_outlined)),
),
SizedBox(
  height: 30.0,
),
TextFormField(
  controller: passwordcontroller,
  validator: (value) {
    if (value == null || value.isEmpty) {
      return 'Please Enter Password';
    }
    return null;
  },
  obscureText: true,
  decoration: InputDecoration(
      hintText: 'Password',
      hintStyle: AppWidget.semiBoldTextFieldStyle(),
      prefixIcon: Icon(Icons.password_outlined)),
),
SizedBox(
  height: 80.0,
),
GestureDetector(
  onTap: () async {
    if (_formkey.currentState!.validate()) {
      setState(() {
        email = mailcontroller.text;
        name = namecontroller.text;
        password = passwordcontroller.text;
      });
    }
    registration();
  },
  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(
        "SIGN UP",
        style: TextStyle(
            color: Colors.white,
            fontSize: 18.0,
```

```
                                fontFamily: 'Poppins1',
                                fontWeight: FontWeight.bold),
                            )),
                          ),
                        ),
                      ),
                    ),
                  ],
                ),
              ),
            ),
          ),
          SizedBox(
            height: 70.0,
          ),
          GestureDetector(
              onTap: () {
                Navigator.push(context,
                    MaterialPageRoute(builder: (context) => LogIn()));
              },
              child: Text(
                "Already have an account? Login",
                style: AppWidget.semiBoldTextFieldStyle(),
              ))
        ],
      ),
    )
  ],
),
),
);
}
}
```



AddItem

Code:-

```
import 'dart:io';

import 'package:firebase_storage/firebase_storage.dart';
import 'package:flutter/material.dart';
import 'package:food_panda/service/database.dart';
import 'package:food_panda/widget/widget_support.dart';
import 'package:image_picker/image_picker.dart';
import 'package:random_string/random_string.dart';
// import 'package:sample_flutter/widget/widget_support.dart';

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

  @override
  State<AddFood> createState() => _AddFoodState();
}

class _AddFoodState extends State<AddFood> {
  final List<String> fooditems = ['Ice-cream', 'Burger', 'Salad', 'Pizza'];
  String? value;
  TextEditingController namecontroller = new TextEditingController();
  TextEditingController pricecontroller = new TextEditingController();
  TextEditingController detailcontroller = new TextEditingController();
  final ImagePicker _picker = ImagePicker();
  File? selectedImage;

  Future getImage() async {
    var image = await _picker.pickImage(source: ImageSource.gallery);
    selectedImage = File(image!.path);
    setState(() {});
  }

  uploadItem() async {
    if (selectedImage != null &&
        namecontroller.text != "" &&
        pricecontroller.text != "" &&
        detailcontroller.text != "") {
      String addId = randomAlphaNumeric(10);
      Reference firebaseStorageRef =
          FirebaseStorage.instance.ref().child("blogImages").child(addId);
      final UploadTask task = firebaseStorageRef.putFile(selectedImage!);

      var downloadUrl = await (await task).ref.getDownloadURL();

      Map<String, dynamic> addItem = {
        "Image": downloadUrl,
        "Name": namecontroller.text,
        "Price": pricecontroller.text,
        "Detail": detailcontroller.text
      };
```

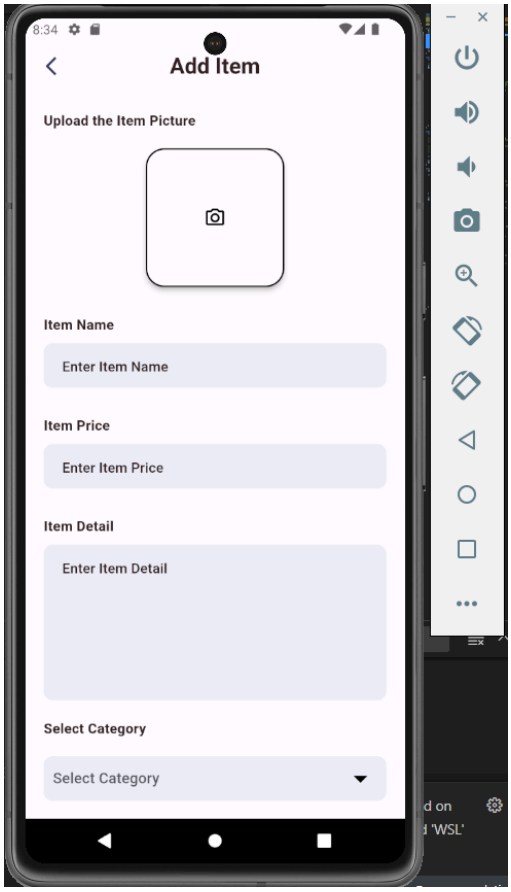
```
    await DatabaseMethods().addFoodItem(addItem, value!).then((value) {
      ScaffoldMessenger.of(context).showSnackBar((SnackBar(
          backgroundColor: Colors.redAccent,
          content: Text(
            "Food item added successfully",
            style: TextStyle(fontSize: 20.0),
          ))));
    });
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      leading: GestureDetector(
          onTap: () {
            Navigator.pop(context);
          },
          child: Icon(
            Icons.arrow_back_ios_new_outlined,
            color: Color(0xFF373866),
          )),
      centerTitle: true,
      title: Text(
        "Add Item",
        style: AppWidget.HeadlineTextFieldStyle(),
      ),
    ),
    body: SingleChildScrollView(
      child: Container(
        margin:
            EdgeInsets.only(left: 20.0, right: 20.0, top: 20.0, bottom: 50.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Text(
              "Upload the Item Picture",
              style: AppWidget.semiBoldTextFieldStyle(),
            ),
            SizedBox(
              height: 20.0,
            ),
            selectedImage == null
                ? GestureDetector(
                    onTap: () {
                      getImage();
                    },
                    child: Center(
                      child: Material(
                        elevation: 4.0,
                        borderRadius: BorderRadius.circular(20),
                        child: Container(
                          width: 150,
```

```
              height: 150,
              decoration: BoxDecoration(
                border:
                    Border.all(color: Colors.black, width: 1.5),
                borderRadius: BorderRadius.circular(20),
              ),
              child: Icon(
                Icons.camera_alt_outlined,
                color: Colors.black,
              ),
            ),
          ),
        ),
      )
    : Center(
        child: Material(
          elevation: 4.0,
          borderRadius: BorderRadius.circular(20),
          child: Container(
            width: 150,
            height: 150,
            decoration: BoxDecoration(
              border: Border.all(color: Colors.black, width: 1.5),
              borderRadius: BorderRadius.circular(20),
            ),
            child: Image.file(
              selectedImage!,
              fit: BoxFit.cover,
            ),
          ),
        ),
      ),
SizedBox(
  height: 30.0,
),
Text(
  "Item Name",
  style: AppWidget.semiBoldTextFieldStyle(),
),
SizedBox(
  height: 10.0,
),
Container(
  padding: EdgeInsets.symmetric(horizontal: 20.0),
  width: MediaQuery.of(context).size.width,
  decoration: BoxDecoration(
      color: Color(0xFFececf8),
      borderRadius: BorderRadius.circular(10)),
  child: TextField(
    controller: namecontroller,
    decoration: InputDecoration(
        border: InputBorder.none,
        hintText: "Enter Item Name",
        hintStyle: AppWidget.LightTextFieldStyle()),
```

```
  ),
),
SizedBox(
  height: 30.0,
),
Text(
  "Item Price",
  style: AppWidget.semiBoldTextFieldStyle(),
),
SizedBox(
  height: 10.0,
),
Container(
  padding: EdgeInsets.symmetric(horizontal: 20.0),
  width: MediaQuery.of(context).size.width,
  decoration: BoxDecoration(
      color: Color(0xFFececf8),
      borderRadius: BorderRadius.circular(10)),
  child: TextField(
    controller: pricecontroller,
    decoration: InputDecoration(
        border: InputBorder.none,
        hintText: "Enter Item Price",
        hintStyle: AppWidget.LightTextFieldStyle()),
  ),
),
SizedBox(
  height: 30.0,
),
Text(
  "Item Detail",
  style: AppWidget.semiBoldTextFieldStyle(),
),
SizedBox(
  height: 10.0,
),
Container(
  padding: EdgeInsets.symmetric(horizontal: 20.0),
  width: MediaQuery.of(context).size.width,
  decoration: BoxDecoration(
      color: Color(0xFFececf8),
      borderRadius: BorderRadius.circular(10)),
  child: TextField(
    maxLines: 6,
    controller: detailcontroller,
    decoration: InputDecoration(
        border: InputBorder.none,
        hintText: "Enter Item Detail",
        hintStyle: AppWidget.LightTextFieldStyle()),
  ),
),
SizedBox(
  height: 20.0,
),
```

```
Text(
  "Select Category",
  style: AppWidget.semiBoldTextFieldStyle(),
),
SizedBox(
  height: 20.0,
),
Container(
  padding: EdgeInsets.symmetric(horizontal: 10.0),
  width: MediaQuery.of(context).size.width,
  decoration: BoxDecoration(
      color: Color(0xFFececf8),
      borderRadius: BorderRadius.circular(10)),
  child: DropdownButtonHideUnderline(
      child: DropdownButton<String>(
   items: fooditems
       .map((item) => DropdownMenuItem<String>(
           value: item,
           child: Text(
             item,
             style:
                 TextStyle(fontSize: 18.0, color: Colors.black),
           )))
       .toList(),
    onChanged: ((value) => setState(() {
         this.value = value;
       })),
    dropdownColor: Colors.white,
    hint: Text("Select Category"),
    iconSize: 36,
    icon: Icon(
      Icons.arrow_drop_down,
      color: Colors.black,
    ),
    value: value,
  )),
),
SizedBox(
  height: 30.0,
),
GestureDetector(
  onTap: () {
    uploadItem();
  },
  child: Center(
    child: Material(
      elevation: 5.0,
      borderRadius: BorderRadius.circular(10),
      child: Container(
        padding: EdgeInsets.symmetric(vertical: 5.0),
        width: 150,
        decoration: BoxDecoration(
            color: Colors.black,
            borderRadius: BorderRadius.circular(10)),
```

```
                child: Center(
                  child: Text(
                    "Add",
                    style: TextStyle(
                        color: Colors.white,
                        fontSize: 22.0,
                        fontWeight: FontWeight.bold),
                  ),
                ),
              ),
            ),
          ),
        )
      ],
    ),
  ),
),
);
}
}
```



Conclusion :Thus I learnt to create interactive form widgets

MAD & PWA Lab
Journal

Experiment No.	05
Experiment Title.	To apply navigation, routing and gestures in Flutter App
Roll No.	47
Name	PRANAV SANDEEP RAIKAR
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation
Grade:	15

PRANAV RAIKAR
D15A 47
Experiment 5

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

Theory:

In Flutter, applying navigation, routing, and gestures are essential aspects of creating a smooth and intuitive user experience in your app. Let's break down each of these concepts:
Navigation and Routing:

- Navigation in Flutter involves moving between different screens or "routes" within your app.
- Routing is the mechanism by which you define and manage these routes.
- Flutter provides a Navigator class to handle navigation and route management.
- You can use named routes to define the routes in your app, making it easier to navigate between them.
- Navigation and routing are crucial for maintaining a smooth user flow and organizing the content of your app effectively.

Gestures:

- Gestures refer to user interactions such as tapping, swiping, pinching, etc.
- Flutter offers a variety of gesture recognizers to detect and respond to these user actions.
- Common gesture recognizers include GestureDetector, InkWell, DragGestureRecognizer, ScaleGestureRecognizer, etc.
- Gestures play a significant role in making your app interactive and responsive to user input.
- By implementing appropriate gestures, you can enhance the usability and engagement of your app.

Combining Navigation and Gestures:

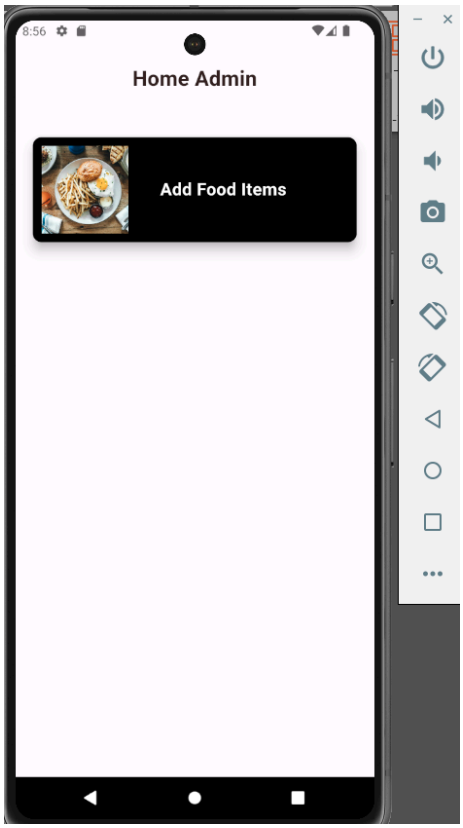
- Combining navigation and gestures allows you to create interactive user interfaces in your Flutter app.
- For example, you can navigate to a different screen when a specific gesture, such as tapping or swiping, is detected.
- By integrating navigation with gestures, you can create seamless user experiences that are both intuitive and engaging.
- It's essential to consider the context and purpose of each gesture to ensure it aligns with the overall navigation flow of your app.

Code:

Home_admin:

```
import 'package:flutter/material.dart';
import 'package:food_panda/admin/add_food.dart';
import 'package:food_panda/widget/widget_support.dart';
// import 'package:sample_flutter/admin/add_food.dart';

// import 'package:sample_flutter/widget/widget_support.dart';

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

  @override
  State<HomeAdmin> createState() => _HomeAdminState();
}

class _HomeAdminState extends State<HomeAdmin> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Container(
        margin: EdgeInsets.only(top: 50.0, left: 20.0, right: 20.0),
        child: Column(
          children: [
            Center(
              child: Text(
                "Home Admin",
                style: AppWidget.HeadlineTextFieldStyle(),
              ),
            ),
            SizedBox(
              height: 50.0,
            ),
            GestureDetector(
              onTap: () {
                Navigator.push(context,
                    MaterialPageRoute(builder: (context) => AddFood()));
              },
              child: Material(
                elevation: 10.0,
                borderRadius: BorderRadius.circular(10),
                child: Center(
                  child: Container(
                    padding: EdgeInsets.all(4),
                    decoration: BoxDecoration(
                      color: Colors.black,
                      borderRadius: BorderRadius.circular(10),
                    ),
                    child: Row(
                      children: [
                        Padding(
                          padding: EdgeInsets.all(6.0),
                          child: Image.asset(
                            "images/food.jpg",
                            height: 100,
```

```
                    width: 100,
                    fit: BoxFit.cover,
                  ),
                ),
                SizedBox(
                  width: 30.0,
                ),
                Text(
                  "Add Food Items",
                  style: TextStyle(
                      color: Colors.white,
                      fontSize: 20.0,
                      fontWeight: FontWeight.bold),
                )
              ],
            ),
          ),
        ),
      ),
    )
  ],
 ),
),
);
}
}
```



Add_food:

Code:

```
import 'dart:io';

import 'package:firebase_storage/firebase_storage.dart';
import 'package:flutter/material.dart';
import 'package:food_panda/service/database.dart';
import 'package:food_panda/widget/widget_support.dart';
import 'package:image_picker/image_picker.dart';
import 'package:random_string/random_string.dart';
// import 'package:sample_flutter/widget/widget_support.dart';

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

  @override
  State<AddFood> createState() => _AddFoodState();
}

class _AddFoodState extends State<AddFood> {
  final List<String> fooditems = ['Ice-cream', 'Burger', 'Salad', 'Pizza'];
  String? value;
  TextEditingController namecontroller = new TextEditingController();
  TextEditingController pricecontroller = new TextEditingController();
  TextEditingController detailcontroller = new TextEditingController();
  final ImagePicker _picker = ImagePicker();
  File? selectedImage;

  Future getImage() async {
    var image = await _picker.pickImage(source: ImageSource.gallery);
    selectedImage = File(image!.path);
    setState(() {});
  }

  uploadItem() async {
    if (selectedImage != null &&
        namecontroller.text != "" &&
        pricecontroller.text != "" &&
        detailcontroller.text != "") {
      String addId = randomAlphaNumeric(10);
      Reference firebaseStorageRef =
          FirebaseStorage.instance.ref().child("blogImages").child(addId);
      final UploadTask task = firebaseStorageRef.putFile(selectedImage!);

      var downloadUrl = await (await task).ref.getDownloadURL();

      Map<String, dynamic> addItem = {
        "Image": downloadUrl,
        "Name": namecontroller.text,
        "Price": pricecontroller.text,
        "Detail": detailcontroller.text
      };
      await DatabaseMethods().addFoodItem(addItem, value!).then((value) {
        ScaffoldMessenger.of(context).showSnackBar((SnackBar(
            backgroundColor: Colors.redAccent,
```

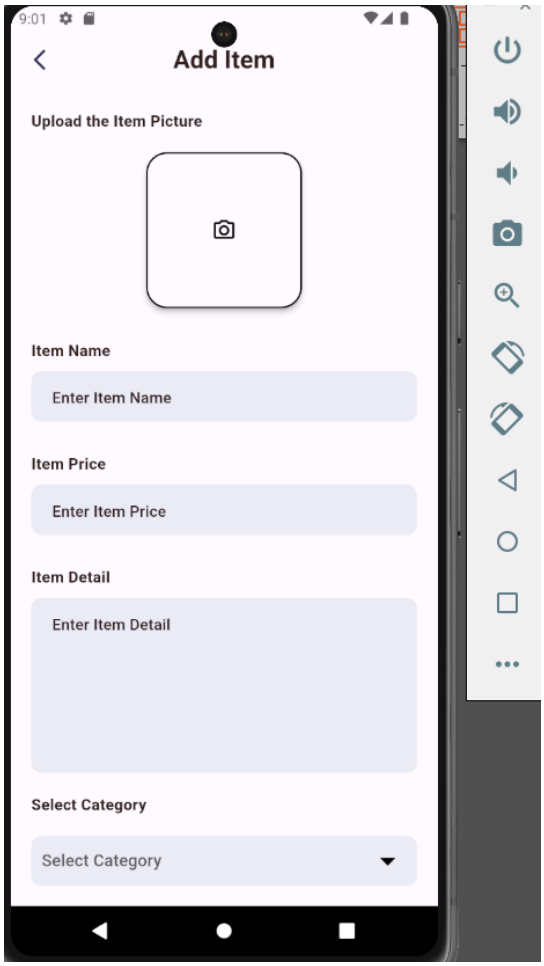
```
          content: Text(
            "Food item added successfully",
            style: TextStyle(fontSize: 20.0),
          )));
    });
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      leading: GestureDetector(
          onTap: () {
            Navigator.pop(context);
          },
          child: Icon(
            Icons.arrow_back_ios_new_outlined,
            color: Color(0xFF373866),
          )),
      centerTitle: true,
      title: Text(
        "Add Item",
        style: AppWidget.HeadlineTextFieldStyle(),
      ),
    ),
    body: SingleChildScrollView(
      child: Container(
        margin:
            EdgeInsets.only(left: 20.0, right: 20.0, top: 20.0, bottom: 50.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Text(
              "Upload the Item Picture",
              style: AppWidget.semiBoldTextFieldStyle(),
            ),
            SizedBox(
              height: 20.0,
            ),
            selectedImage == null
                ? GestureDetector(
                    onTap: () {
                      getImage();
                    },
                    child: Center(
                      child: Material(
                        elevation: 4.0,
                        borderRadius: BorderRadius.circular(20),
                        child: Container(
                          width: 150,
                          height: 150,
                          decoration: BoxDecoration(
                            border:
```

```
                Border.all(color: Colors.black, width: 1.5),
              borderRadius: BorderRadius.circular(20),
            ),
            child: Icon(
              Icons.camera_alt_outlined,
              color: Colors.black,
            ),
          ),
        ),
      ),
    )
  : Center(
      child: Material(
        elevation: 4.0,
        borderRadius: BorderRadius.circular(20),
        child: Container(
          width: 150,
          height: 150,
          decoration: BoxDecoration(
            border: Border.all(color: Colors.black, width: 1.5),
            borderRadius: BorderRadius.circular(20),
          ),
          child: Image.file(
            selectedImage!,
            fit: BoxFit.cover,
          ),
        ),
      ),
    ),
SizedBox(
  height: 30.0,
),
Text(
  "Item Name",
  style: AppWidget.semiBoldTextFieldStyle(),
),
SizedBox(
  height: 10.0,
),
Container(
  padding: EdgeInsets.symmetric(horizontal: 20.0),
  width: MediaQuery.of(context).size.width,
  decoration: BoxDecoration(
      color: Color(0xFFececf8),
      borderRadius: BorderRadius.circular(10)),
  child: TextField(
    controller: namecontroller,
    decoration: InputDecoration(
        border: InputBorder.none,
        hintText: "Enter Item Name",
        hintStyle: AppWidget.LightTextFieldStyle()),
  ),
),
SizedBox(
```

```
  height: 30.0,
),
Text(
  "Item Price",
  style: AppWidget.semiBoldTextFieldStyle(),
),
SizedBox(
  height: 10.0,
),
Container(
  padding: EdgeInsets.symmetric(horizontal: 20.0),
  width: MediaQuery.of(context).size.width,
  decoration: BoxDecoration(
      color: Color(0xFFececf8),
      borderRadius: BorderRadius.circular(10)),
  child: TextField(
    controller: pricecontroller,
    decoration: InputDecoration(
        border: InputBorder.none,
        hintText: "Enter Item Price",
        hintStyle: AppWidget.LightTextFieldStyle()),
  ),
),
SizedBox(
  height: 30.0,
),
Text(
  "Item Detail",
  style: AppWidget.semiBoldTextFieldStyle(),
),
SizedBox(
  height: 10.0,
),
Container(
  padding: EdgeInsets.symmetric(horizontal: 20.0),
  width: MediaQuery.of(context).size.width,
  decoration: BoxDecoration(
      color: Color(0xFFececf8),
      borderRadius: BorderRadius.circular(10)),
  child: TextField(
    maxLines: 6,
    controller: detailcontroller,
    decoration: InputDecoration(
        border: InputBorder.none,
        hintText: "Enter Item Detail",
        hintStyle: AppWidget.LightTextFieldStyle()),
  ),
),
SizedBox(
  height: 20.0,
),
Text(
  "Select Category",
  style: AppWidget.semiBoldTextFieldStyle(),
```

```
),
SizedBox(
  height: 20.0,
),
Container(
  padding: EdgeInsets.symmetric(horizontal: 10.0),
  width: MediaQuery.of(context).size.width,
  decoration: BoxDecoration(
      color: Color(0xFFececf8),
      borderRadius: BorderRadius.circular(10)),
  child: DropdownButtonHideUnderline(
      child: DropdownButton<String>(
    items: fooditems
        .map((item) => DropdownMenuItem<String>(
            value: item,
            child: Text(
              item,
              style:
                  TextStyle(fontSize: 18.0, color: Colors.black),
            )))
        .toList(),
    onChanged: ((value) => setState(() {
          this.value = value;
        })),
    dropdownColor: Colors.white,
    hint: Text("Select Category"),
    iconSize: 36,
    icon: Icon(
      Icons.arrow_drop_down,
      color: Colors.black,
    ),
    value: value,
  )),
),
SizedBox(
  height: 30.0,
),
GestureDetector(
  onTap: () {
    uploadItem();
  },
  child: Center(
    child: Material(
      elevation: 5.0,
      borderRadius: BorderRadius.circular(10),
      child: Container(
        padding: EdgeInsets.symmetric(vertical: 5.0),
        width: 150,
        decoration: BoxDecoration(
            color: Colors.black,
            borderRadius: BorderRadius.circular(10)),
        child: Center(
          child: Text(
            "Add",
```

```
                    style: TextStyle(
                        color: Colors.white,
                        fontSize: 22.0,
                        fontWeight: FontWeight.bold),
                  ),
                ),
              ),
            ),
          ),
        ),
      )
    ],
  ),
 ),
),
);
}
}
```



Conclusion :Thus I learnt to apply navigation, routing and gestures in Flutter App

MAD & PWA Lab
Journal

Experiment No.	06
Experiment Title.	To Connect Flutter UI with fireBase database
Roll No.	47
Name	PRANAV SANDEEP RAIKAR
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

PRANAV RAIKAR
D15A 47
Experiment 6

AIM: To Connect Flutter UI with fireBase database

Theory:
Connecting a Flutter app to Firebase involves several steps, including setting up a Firebase project, configuring the Flutter app to use Firebase, and integrating Firebase SDKs into your Flutter code.
1. Firebase Setup: First, you need to create a Firebase project in the Firebase console (https://console.firebase.google.com/). This project will host your app's data and services.
2. Add Firebase to Flutter App: You'll need to add the Firebase SDK to your Flutter app. This involves adding Firebase configuration files to your project and updating your Flutter app's dependencies to include Firebase SDKs.
3. Initialize Firebase: In your Flutter app, you'll initialize Firebase by calling Firebase.initializeApp(). This should typically be done at the beginning of your app's lifecycle, such as in the main() function or in the initState() method of your main widget.
4. Use Firebase Services: Once Firebase is initialized, you can use various Firebase services in your Flutter app, such as Authentication, Firestore, Realtime Database, Cloud Storage, Cloud Messaging, and more.

Main.dart:

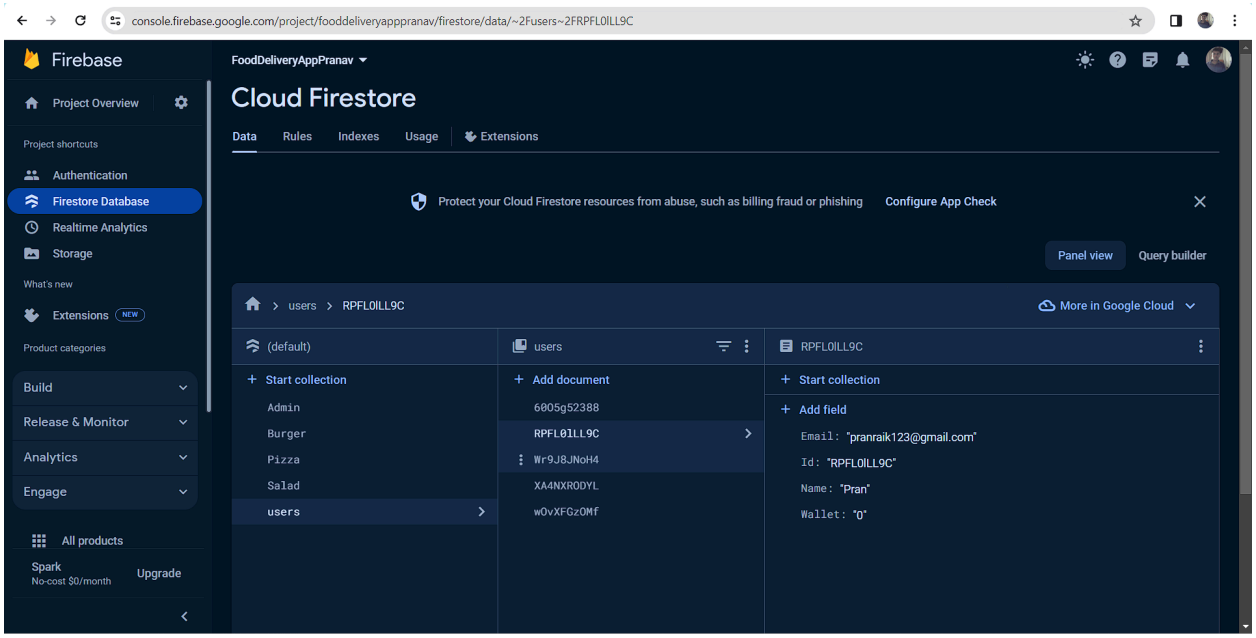
```
import 'package:flutter/material.dart';
import 'package:flutter_stripe/flutter_stripe.dart';
import 'package:food_panda/admin/admin_login.dart';
import 'package:food_panda/pages/bottomnav.dart';
import 'package:food_panda/pages/onboard.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:food_panda/widget/app_constant.dart';
import 'firebase_options.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  Stripe.publishableKey = publishableKey;
  await Firebase.initializeApp(
    options: FirebaseOptions(
      apiKey: 'AIzaSyBftcoBEue-M8hQfcB-M-7ZAbhm_5PVmck',
      appId: '1:726847533270:android:f8faab0dba68351c9b1977',
      messagingSenderId: '726847533270',
      projectId: 'fooddeliveryapppranav',
      storageBucket: 'fooddeliveryapppranav.appspot.com',
    ),
  );
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  // This widget is the root of your application.
  @override
```

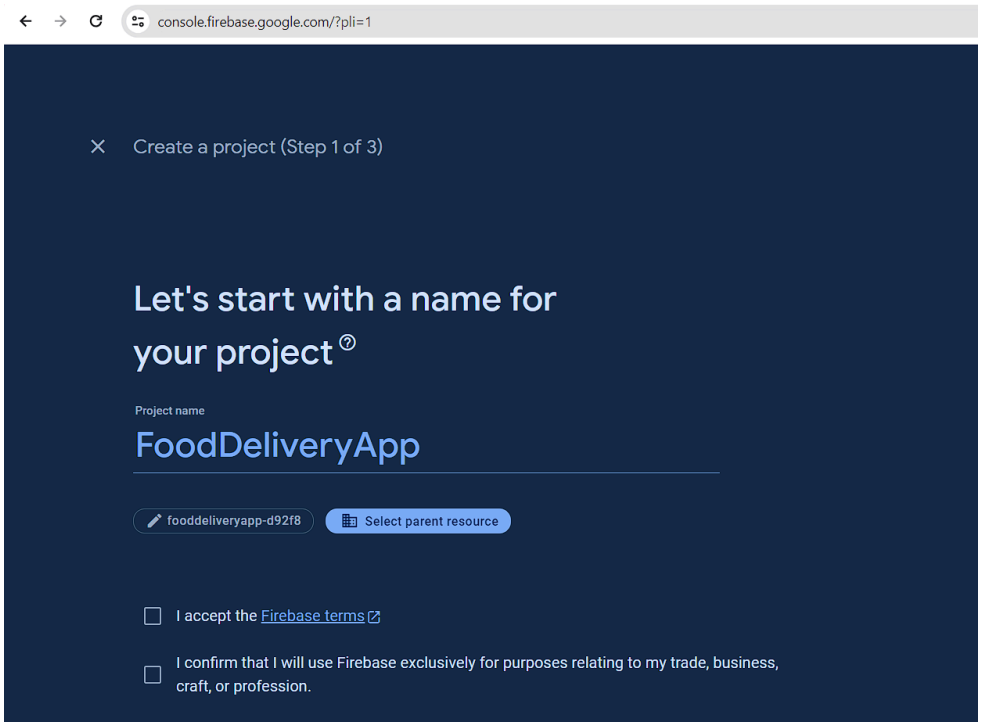
```
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Flutter Demo',
      theme: ThemeData(
        // This is the theme of your application.
        //
        // TRY THIS: Try running your application with "flutter run". You'll see
        // the application has a purple toolbar. Then, without quitting the app,
        // try changing the seedColor in the colorScheme below to Colors.green
        // and then invoke "hot reload" (save your changes or press the "hot
        // reload" button in a Flutter-supported IDE, or press "r" if you used
        // the command line to start the app).
        //
        // Notice that the counter didn't reset back to zero; the application
        // state is not lost during the reload. To reset the state, use hot
        // restart instead.
        //
        // This works for code too, not just values: Most code changes can be
        // tested with just a hot reload.
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),
      home: Onboard(),
      // home: AdminLogin(),

      // home: BottomNav(),
    );
  }
}

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

  // This widget is the home page of your application. It is stateful, meaning
  // that it has a State object (defined below) that contains fields that affect
  // how it looks.

  // This class is the configuration for the state. It holds the values (in this
  // case the title) provided by the parent (in this case the App widget) and
  // used by the build method of the State. Fields in a Widget subclass are
  // always marked "final".

  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++;
    });
```

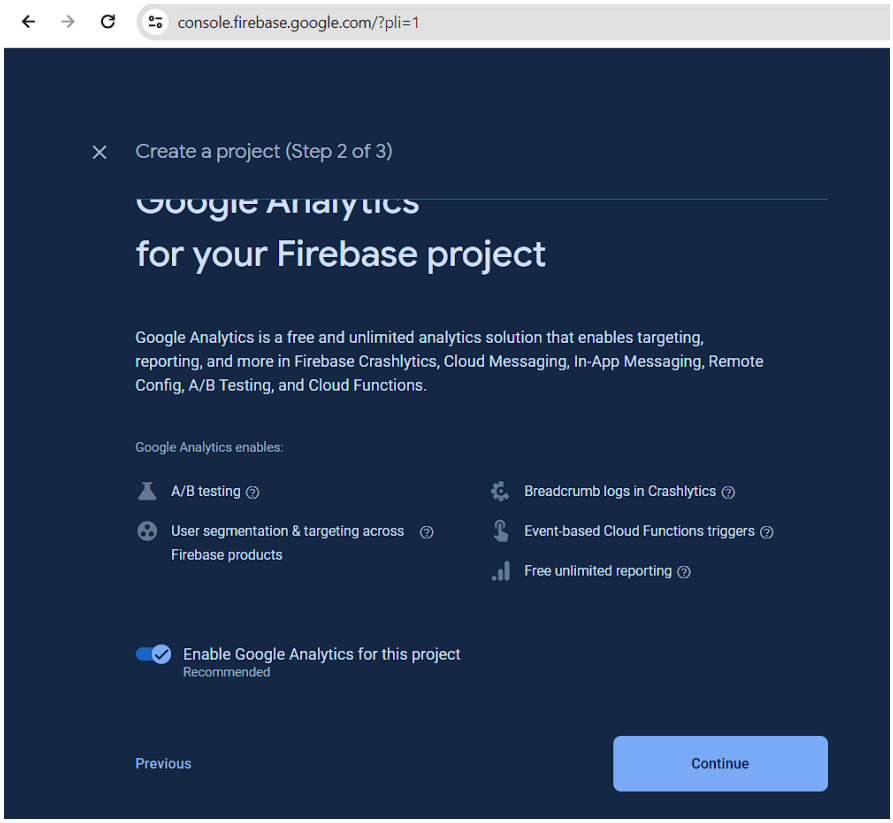
```
  }

  @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(
        // TRY THIS: Try changing the color here to a specific color (to
        // Colors.amber, perhaps?) and trigger a hot reload to see the AppBar
        // change color while the other colors stay the same.
        backgroundColor: Theme.of(context).colorScheme.inversePrimary,
        // Here we take the value from the MyHomePage object that was created by
        // the App.build method, and use it to set our appbar title.
        title: Text(widget.title),
      ),
      body: Center(
        // Center is a layout widget. It takes a single child and positions it
        // in the middle of the parent.
        child: Column(
          // Column is also a layout widget. It takes a list of children and
          // arranges them vertically. By default, it sizes itself to fit its
          // children horizontally, and tries to be as tall as its parent.
          //
          // Column has various properties to control how it sizes itself and
          // how it positions its children. Here we use mainAxisAlignment to
          // center the children vertically; the main axis here is the vertical
          // axis because Columns are vertical (the cross axis would be
          // horizontal).
          //
          // TRY THIS: Invoke "debug painting" (choose the "Toggle Debug Paint"
          // action in the IDE, or press "p" in the console), to see the
          // wireframe for each widget.
          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.
    );
  }
}
```

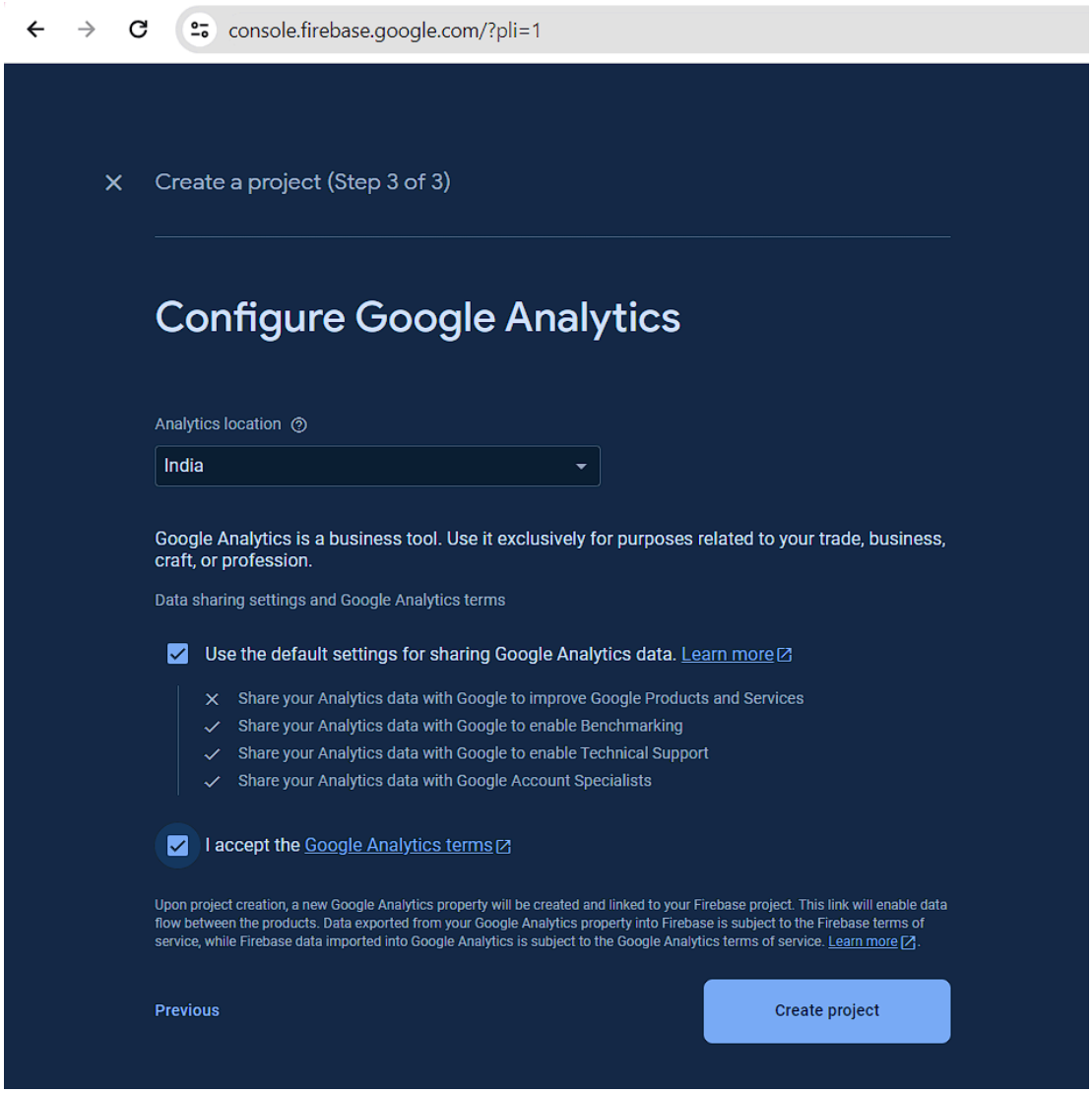
Database.dart:

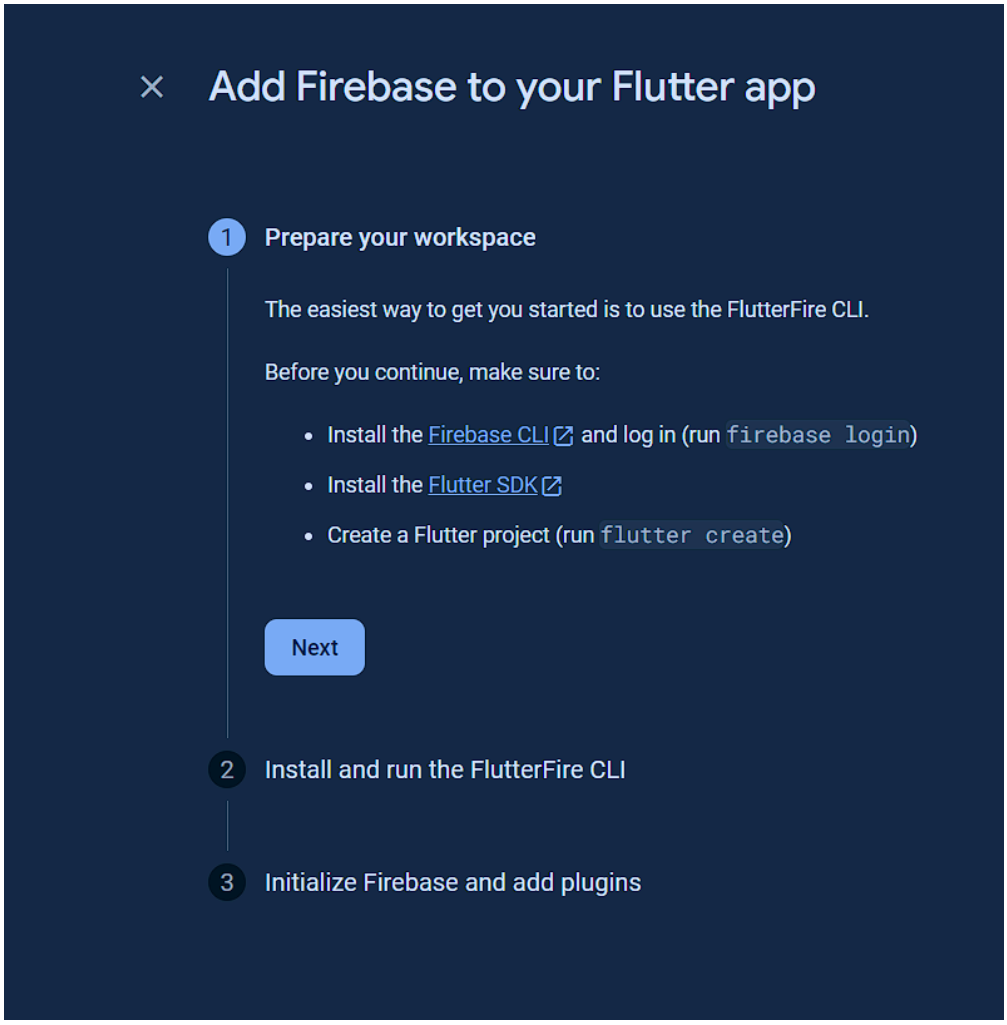
```
import 'package:cloud_firestore/cloud_firestore.dart';

class DatabaseMethods {
  Future addUserDetail(Map<String, dynamic> userInfoMap, String id) async {
    return await FirebaseFirestore.instance
        .collection('users')
        .doc(id)
        .set(userInfoMap);
  }

  UpdateUserwallet(String id, String amount) async {
    return await FirebaseFirestore.instance
        .collection("users")
        .doc(id)
        .update({"Wallet": amount});
  }

  Future addFoodItem(Map<String, dynamic> userInfoMap, String name) async {
    return await FirebaseFirestore.instance.collection(name).add(userInfoMap);
  }

  Future<Stream<QuerySnapshot>> getFoodItem(String name) async {
    return await FirebaseFirestore.instance.collection(name).snapshots();
  }
}
```

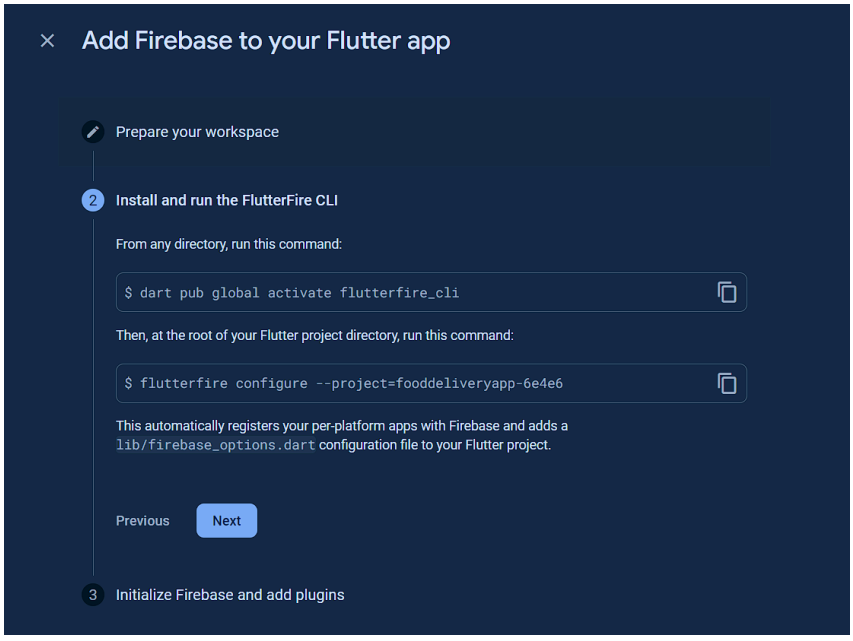



console.firebase.google.com/?pli=1
Create a project (Step 1 of 3)
Let's start with a name for your project
Project name
FoodDeliveryApp
fooddeliveryapp-d92f8
Select parent resource
I accept the Firebase terms
I confirm that I will use Firebase exclusively for purposes relating to my trade, business, craft, or profession.

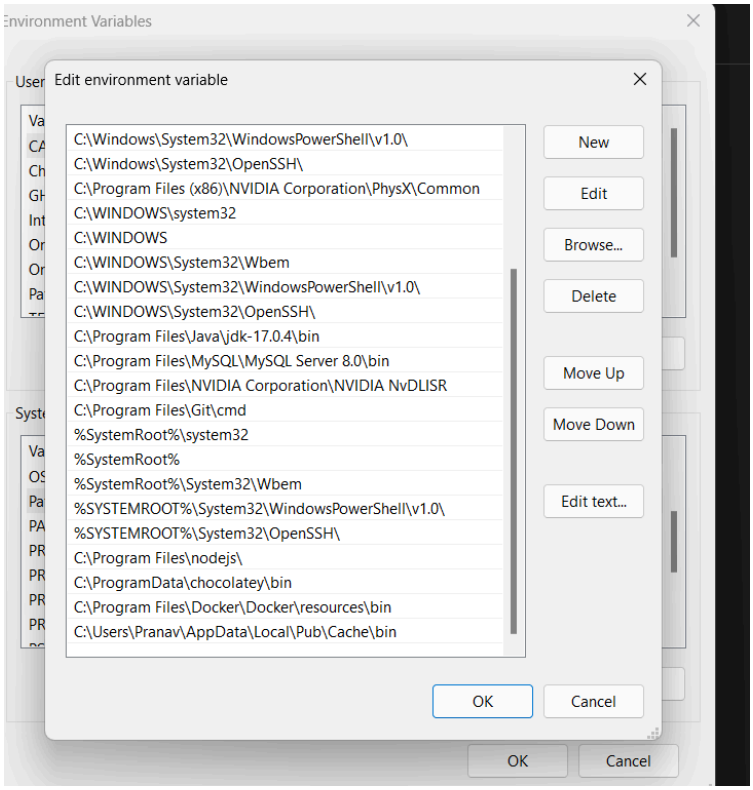

console.firebase.google.com/?pli=1
Create a project (Step 2 of 3)
for your Firebase project
Google Analytics is a free and unlimited analytics solution that enables targeting, reporting, and more in Firebase Crashlytics, Cloud Messaging, In-App Messaging, Remote Config, A/B Testing, and Cloud Functions.
Google Analytics enables:
A/B testing
User segmentation & targeting across Firebase products
Breadcrumb logs in Crashlytics
Event-based Cloud Functions triggers
Free unlimited reporting
Enable Google Analytics for this project
Recommended
Previous
Continue

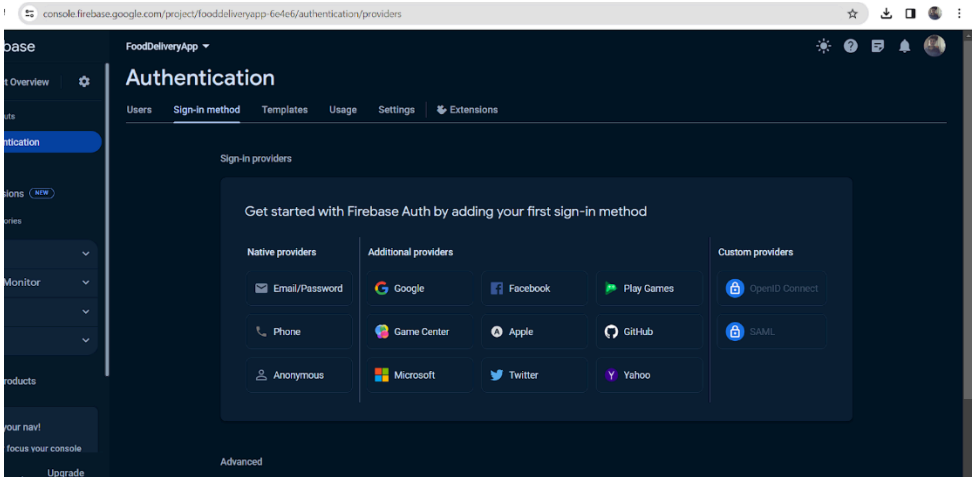

console.firebase.google.com/?pli=1
Create a project (Step 3 of 3)
Configure Google Analytics
Analytics location
India
Google Analytics is a business tool. Use it exclusively for purposes related to your trade, business, craft, or profession.
Data sharing settings and Google Analytics terms
Use the default settings for sharing Google Analytics data. Learn more
Share your Analytics data with Google to improve Google Products and Services
Share your Analytics data with Google to enable Benchmarking
Share your Analytics data with Google to enable Technical Support
Share your Analytics data with Google Account Specialists
I accept the Google Analytics terms
Upon project creation, a new Google Analytics property will be created and linked to your Firebase project. This link will enable data flow between the products. Data exported from your Google Analytics property into Firebase is subject to the Firebase terms of service, while Firebase data imported into Google Analytics is subject to the Google Analytics terms of service. Learn more.
Previous
Create project


Add Firebase to your Flutter app
1
Prepare your workspace
The easiest way to get you started is to use the FlutterFire CLI.
Before you continue, make sure to:
Install the Firebase CLI and log in (run firebase login)
Install the Flutter SDK
Create a Flutter project (run flutter create)
Next
2
Install and run the FlutterFire CLI
3
Initialize Firebase and add plugins

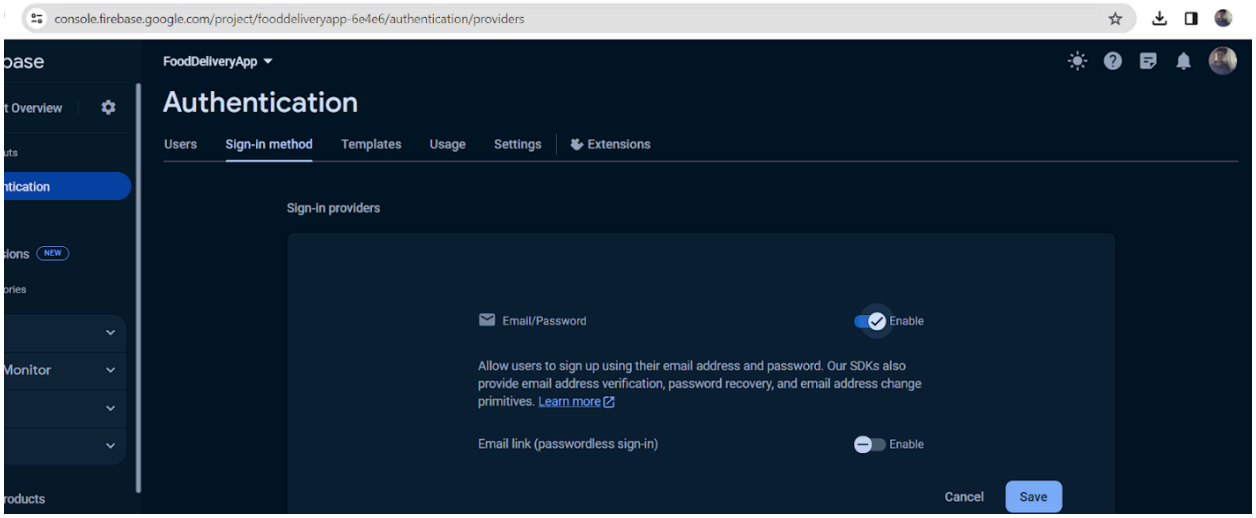

Add Firebase to your Flutter app
Prepare your workspace
2
Install and run the FlutterFire CLI
From any directory, run this command:
$ dart pub global activate flutterfire_cli
Then, at the root of your Flutter project directory, run this command:
$ flutterfire configure --project=fooddeliveryapp-6e4e6
This automatically registers your per-platform apps with Firebase and adds a lib/firebase_options.dart configuration file to your Flutter project.
Previous
Next
3
Initialize Firebase and add plugins

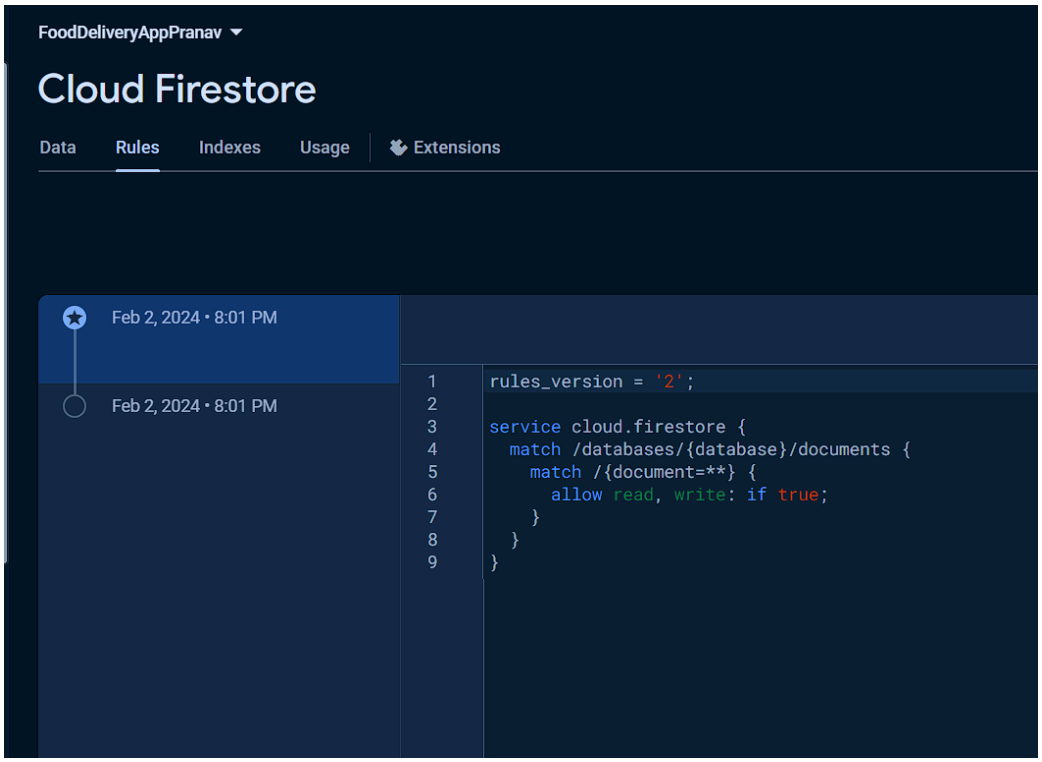
Project Title: Flutter: Food panda Clone/PWA:To-Do-List website **Roll No. 47**



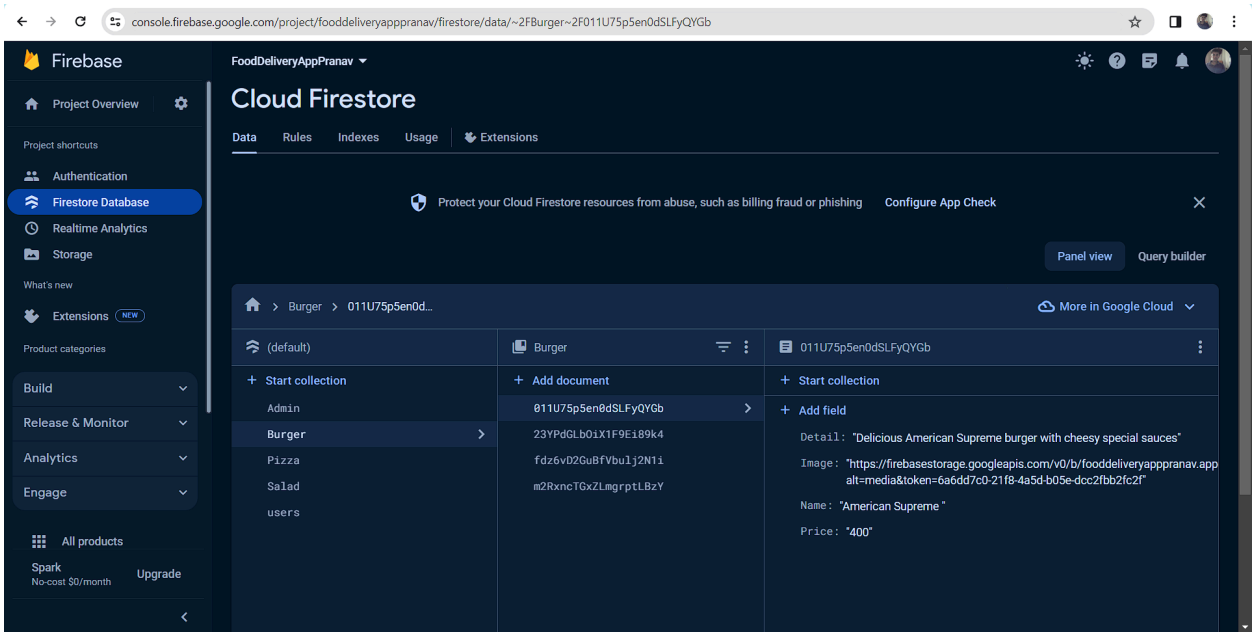


Project Title: Flutter: Food panda Clone/PWA:To-Do-List website **Roll No. 47**





Project Title: Flutter: Food panda Clone/PWA:To-Do-List website **Roll No. 47**



Conclusion :Thus I learnt to Connect Flutter UI with fireBase database

MAD & PWA Lab

Journal

Experiment No.	07
Experiment Title.	To write meta data of your Ecommerce PWA in a Web app manifest file to enable “add to homescreen feature”.
Roll No.	47
Name	PRANAV SANDEEP RAIKAR
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4: Understand various PWA frameworks and their requirements
Grade:	15

NAME:PRANAV SANDEEP RAIKAR
D15A 47

Practical 7

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

Theory:

Regular Web App

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

Progressive Web App

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

Difference between PWAs vs. Regular Web Apps:
A Progressive Web is different and better than a Regular Web app with features like:

Native Experience

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

Ease of Access

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

Faster Services

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

Engaging Approach

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

Updated Real-Time Data Access

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

Discoverable

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

Lower Development Cost

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

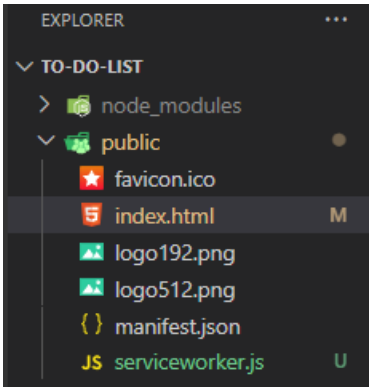
The main features are:

Progressive — They work for every user, regardless of the browser chosen because they are built at the base with progressive improvement principles.
Responsive — They adapt to the various screen sizes: desktop, mobile, tablet, or dimensions that can later become available.
App-like — They behave with the user as if they were native apps, in terms of interaction and navigation.
Updated — Information is always up-to-date thanks to the data update process offered by service workers.
Secure — Exposed over HTTPS protocol to prevent the connection from displaying information or altering the contents.
Searchable — They are identified as “applications” and are indexed by search engines.
Reactivable — Make it easy to reactivate the application thanks to capabilities such as web notifications.
Installable — They allow the user to “save” the apps that he considers most useful with the corresponding icon on the screen of his mobile terminal (home screen) without having to face all the steps and problems related to the use of the app store.
Linkable — Easily shared via URL without complex installations.
Offline — Once more it is about putting the user before everything, avoiding the usual error message in case of weak or no connection. The PWA are based on two particularities: first of all the ‘skeleton’ of the app, which recalls the page structure, even if its contents do not respond

and its elements include the header, the page layout, as well as an illustration that signals that the page is loading.

Code and output:

Adding the manifest and serviceworker files:



Inserting the code:

```
//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

```
/* eslint-disable no-restricted-globals */
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);
    })
  );
});
```

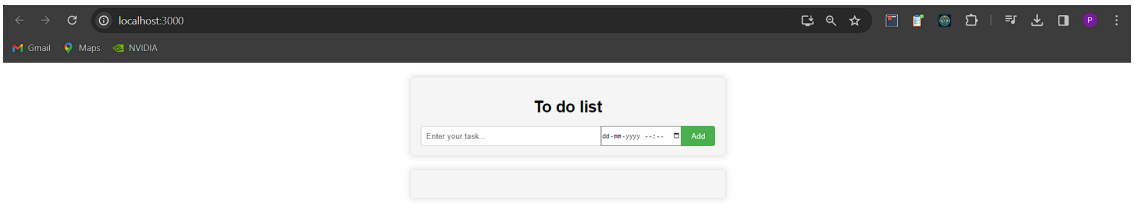
//index.html

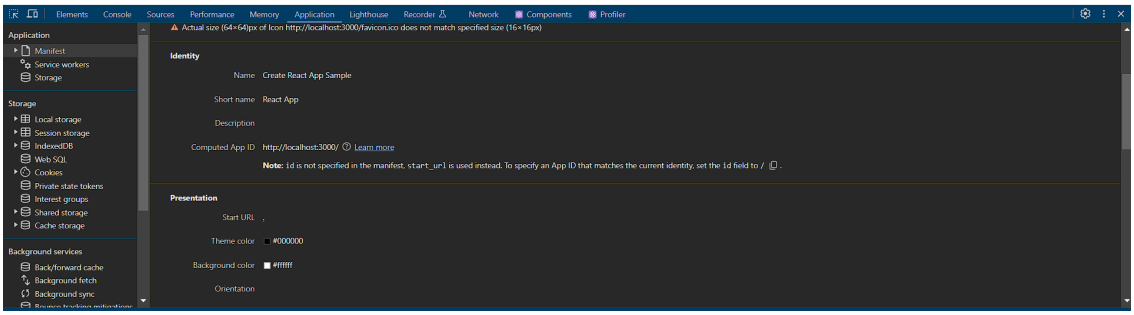
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web 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`.
    -->
    <title>Pranav's To do list</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`.
    -->
```

```
    <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>
```

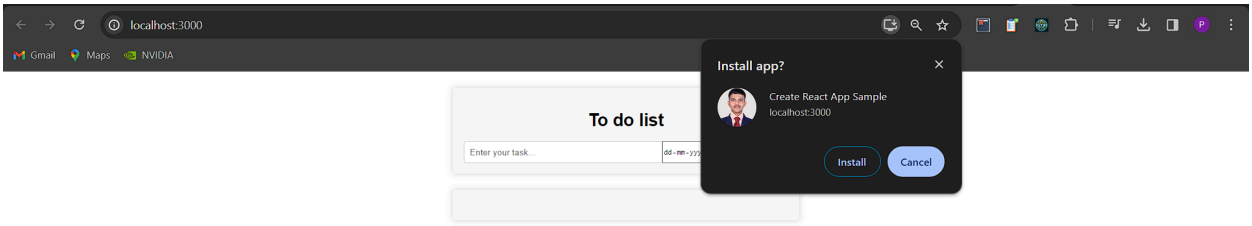
Open localhost:3000 on the browser:

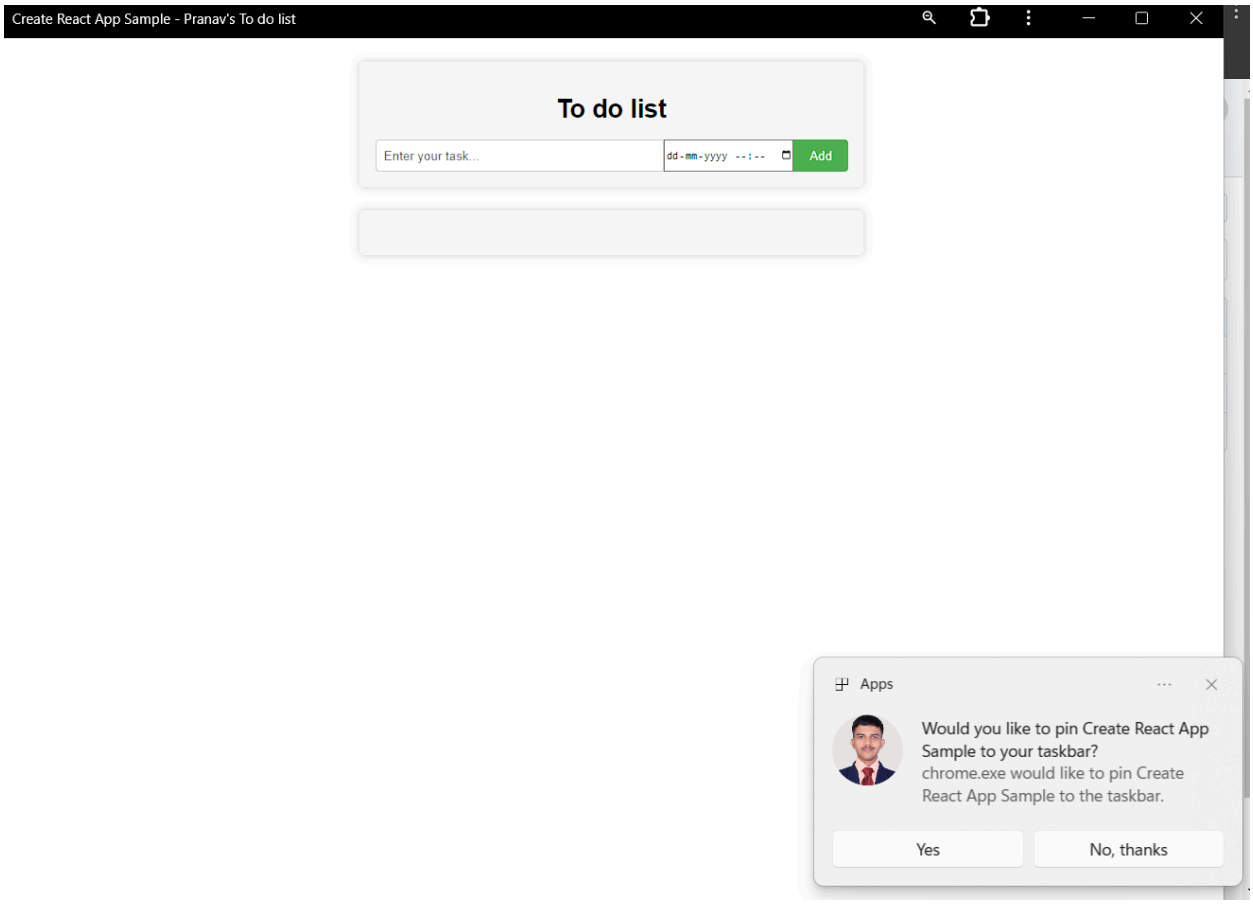
Click on the three dots on the top right corner-> more tools-> developer tools:





Click on the three dots-> Apps-> Install this site as an app





Conclusion: Thus, the "add to homescreen feature was successfully enabled by adding a manifest file to our web

MAD & PWA Lab
Journal

Experiment No.	08
Experiment Title.	To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA
Roll No.	47
Name	PRANAV SANDEEP RAIKAR
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

NAME:PRANAV SANDEEP RAIKAR
D15A 47

EXP 8

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.

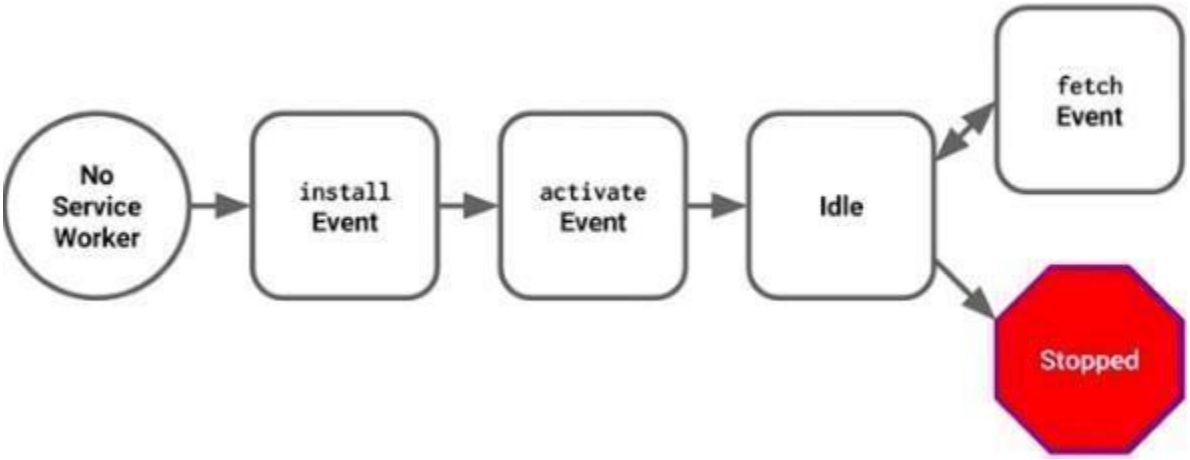
- You can **Continue**
  Although Internet connection is broken, you can start any process with Background Sync of Service Worker.

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.

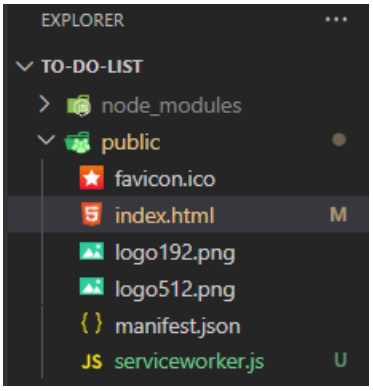
Service Worker Cycle



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`.
    -->
    <title>Pranav's To do list</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`.
    -->
    <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>
```

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:

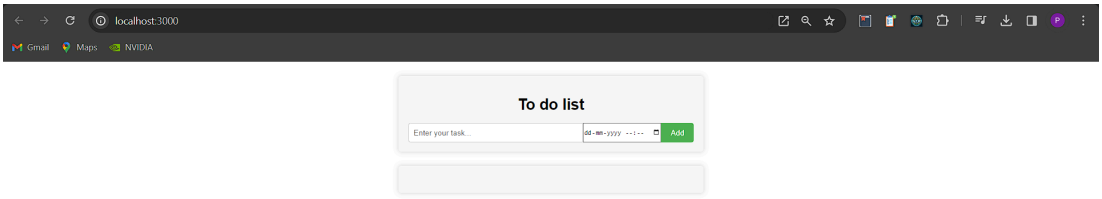
```
var staticCacheName = "pwa";
self.addEventListener("install", function (e) {
e.waitUntil(
    caches.open(staticCacheName).then(function (cache) {
      return cache.addAll(["/"]);
    })
);
```

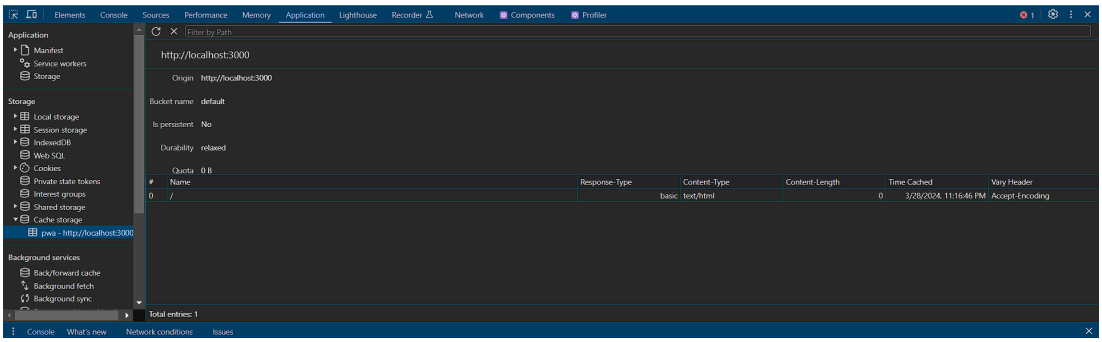
```
});

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

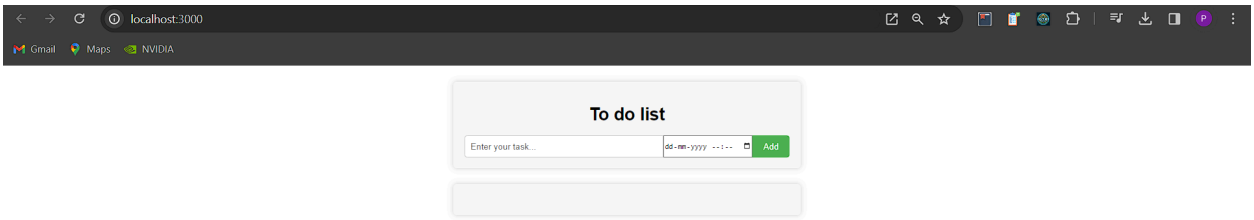
We now run the app using the following command:
> npm run start

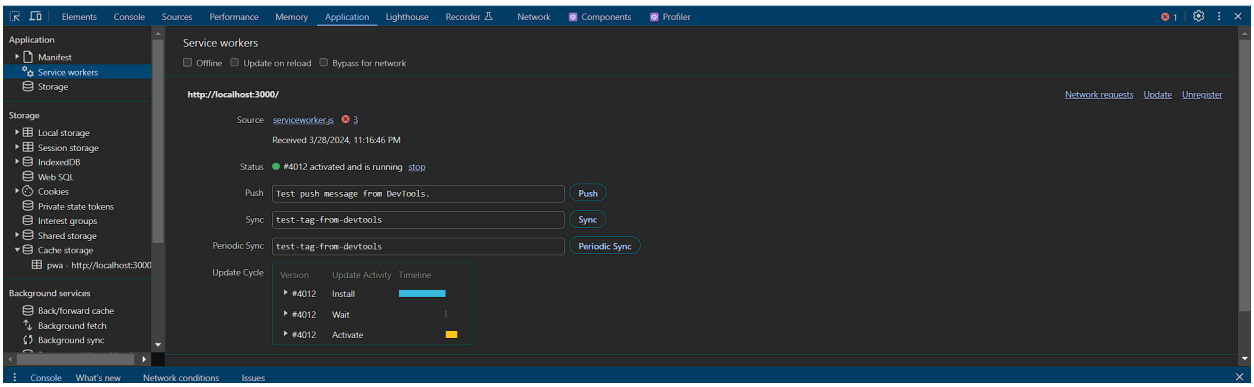
```
You can now view inotebook in the browser.

  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 a Progressive 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.	47
Name	PRANAV SANDEEP RAIKAR
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

NAME:PRANAV SANDEEP RAIKAR
D15A 47

EXP 9

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

Theory:

Service Worker

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

Things to note about Service Worker:

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

Fetch Event

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

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

- **CacheFirst** - In this function, if the received request has cached before, the

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

Sync Event

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

Here is a job for the Background Sync.

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

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

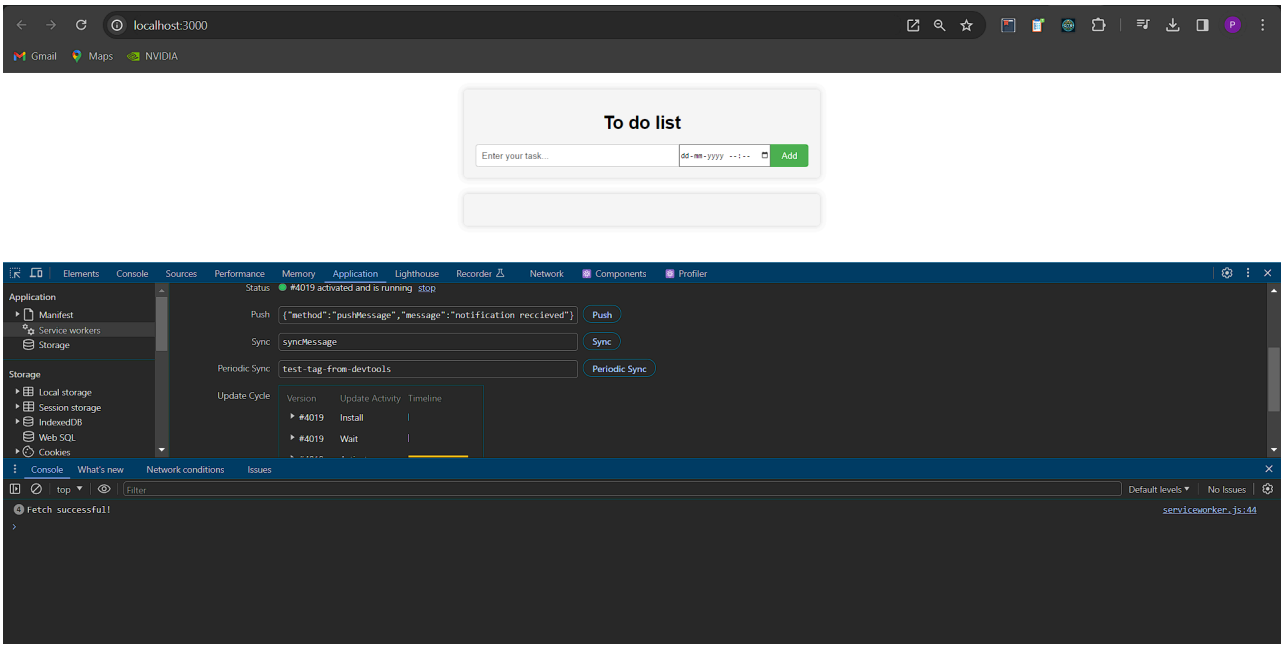
```
var staticCacheName = "pwa";
self.addEventListener("install",
function (event) {
  event.waitUntil(preLoad());
});
self.addEventListener("fetch",
function (event) {
  event.respondWith(

checkResponse(event.request).ca
tch(function () {
      console.log("Fetch from cache
successful!");
      return
returnFromCache(event.request);
    })
  );
  console.log("Fetch successful!");

event.waitUntil(addToCache(event
.request));
});
self.addEventListener("sync",
function (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().t
hen(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 = ["/App",
"/HeaderForm", "/Task",
"/TaskList", "/index.html"];
var preLoad = function () {
 return
caches.open("index").then(functio
n (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);
 });
};
```

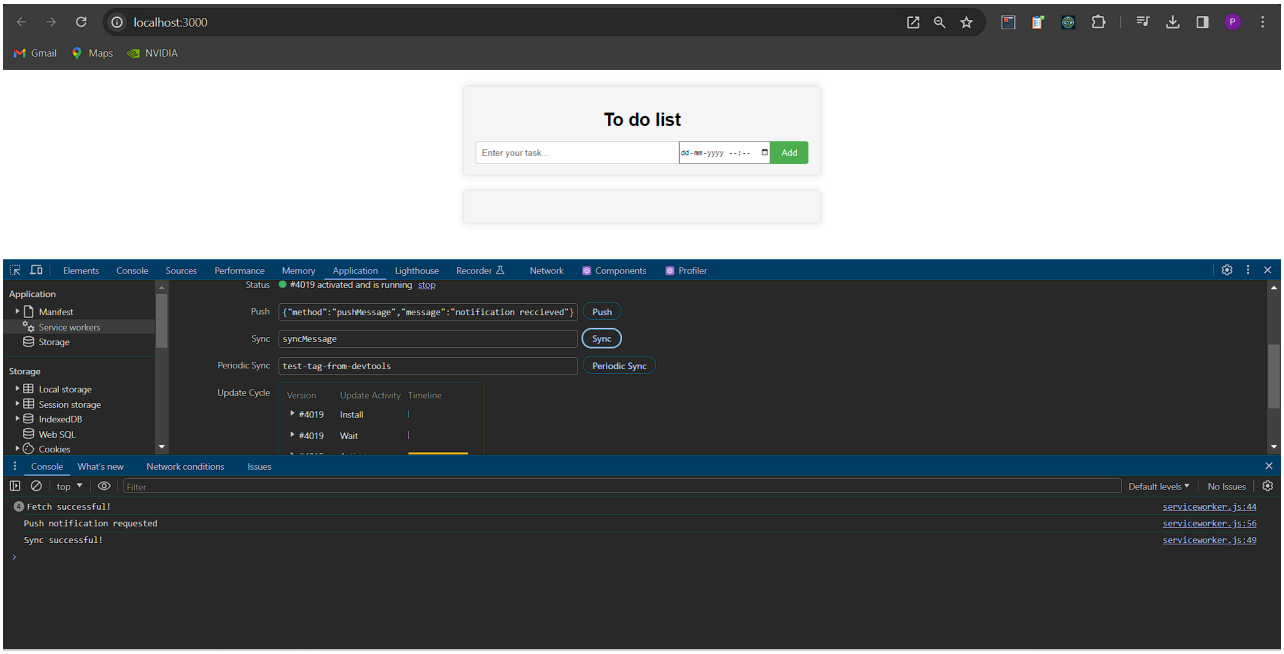
```
var addToCache = function
(request) {
  return
caches.open("index").then(functio
n (cache) {
    return
fetch(request).then(function
(response) {
      return cache.put(request,
response);
    });
  });
};
var returnFromCache = function
(request) {
  return
caches.open("index").then(functio
n (cache) {
    return
cache.match(request).then(functio
n (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: Food panda Clone/PWA:To-Do-List website **Roll No. 47**

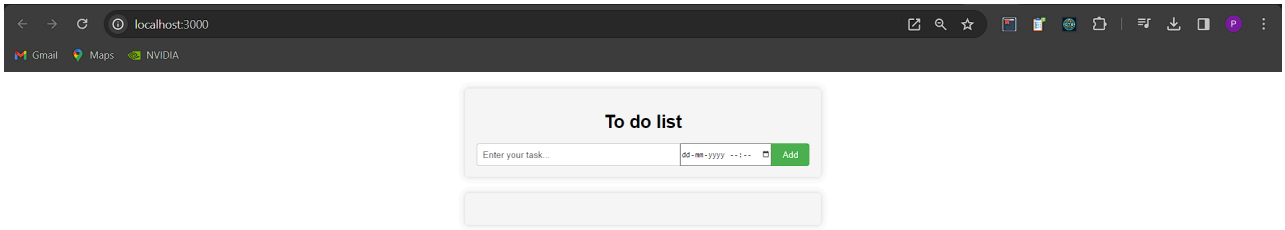


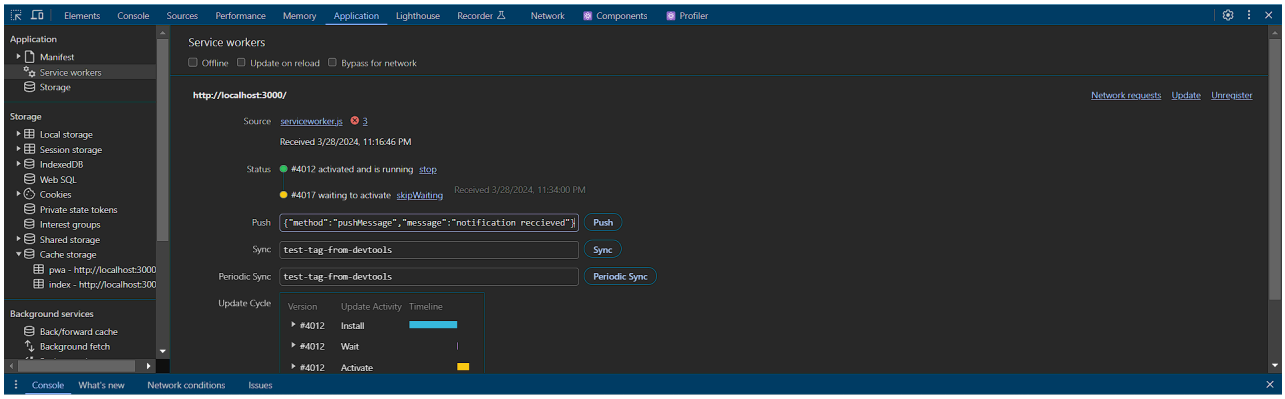
Sync Event :

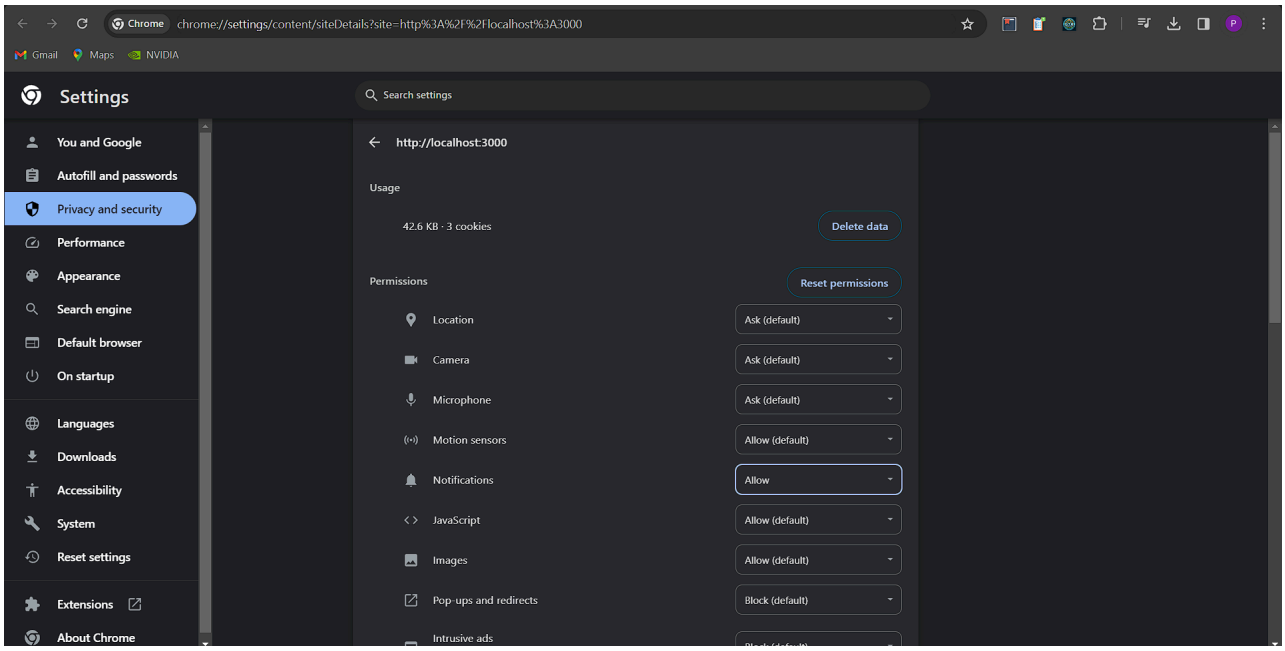


Push Event:

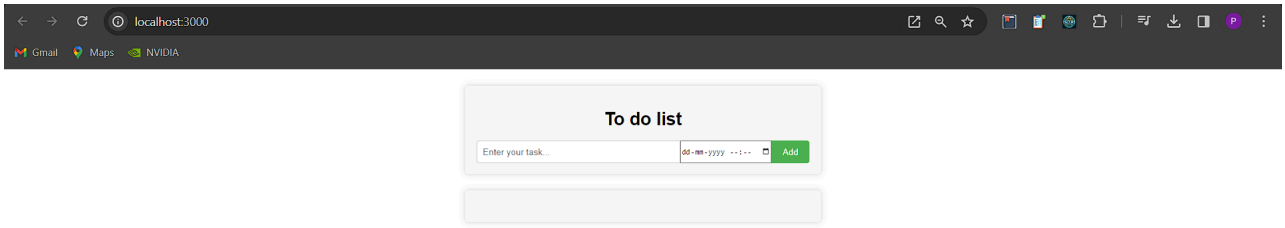
Project Title: Flutter: Food panda Clone/PWA:To-Do-List website **Roll No. 47**

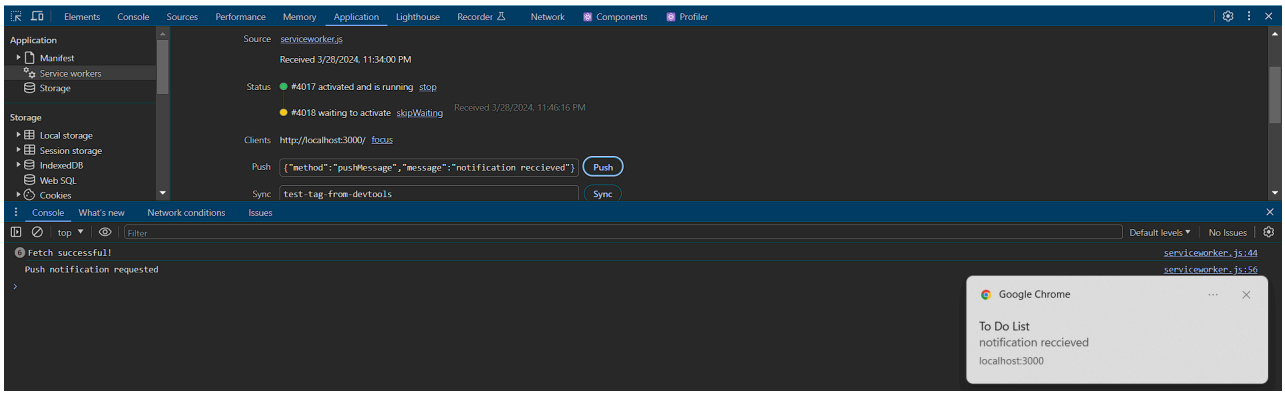






Project Title: Flutter: Food panda Clone/PWA:To-Do-List website **Roll No. 47**





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.	47
Name	PRANAV SANDEEP RAIKAR
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO5: Design and Develop a responsive User Interface by applying PWA Design techniques
Grade:	15

NAME:PRANAV SANDEEP RAIKAR
D15A 47

EXP 10

AIM:

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

THEORY:

GitHub Pages

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

GitHub Pages provides the following key features:

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

Reasons for favoring this over Firebase:

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

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

Pros

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

Cons

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

Firebase

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

Some of the features offered by Firebase are:

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

Reasons for favoring over GitHub Pages:

1. Realtime backend made easy
2. Fast and responsive

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

Pros

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

Cons

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

Link to GitHub repository: https://github.com/pranavraikar01/pwa-project

package.json M

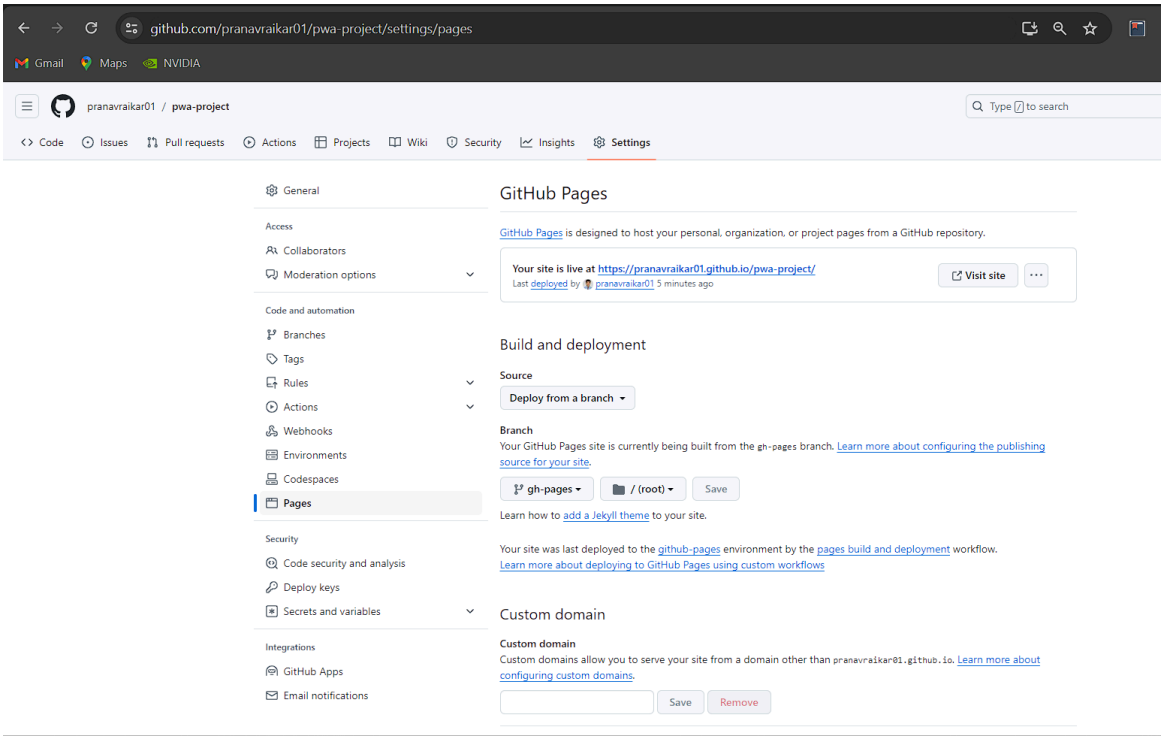
package.json > { } scripts

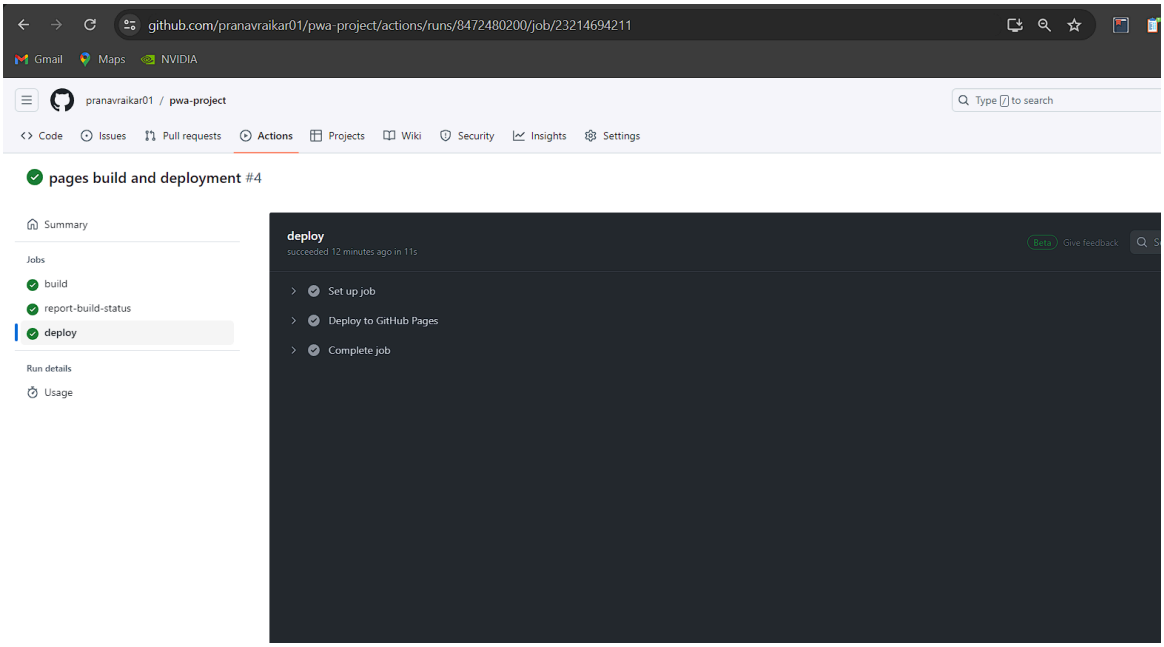
```
{
  "homepage": "https://pranavraikar01.github.io/pwa-project",
  "name": "to-do-list",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.17.0",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.16.0",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  ▷ Debug
  "scripts": {
    "start": "react-scripts start",
    "predeploy":"npm run build",
    "deploy":"gh-pages -d build",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS D:\Full_stack_projects\PWA LAB\pwa-project> npm run deploy

> to-do-list@0.1.0 predeploy
> npm run build


> to-do-list@0.1.0 build
> react-scripts build

Creating an optimized production build...
Browserslist: caniuse-lite is outdated. Please run:
  npx update-browserslist-db@latest
  Why you should do it regularly: https://github.com/browserslist/update-db#readme
One of your dependencies, babel-preset-react-app, is importing the
"@babel/plugin-proposal-private-property-in-object" package without
declaring it in its dependencies. This is currently working because
```

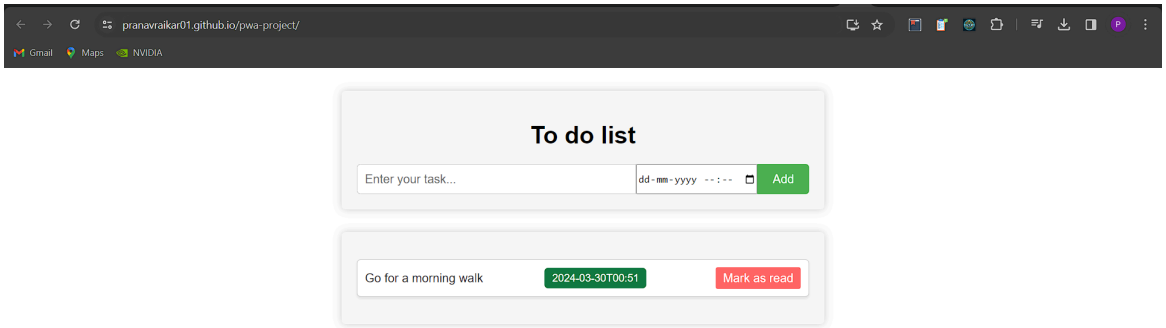
Project Title: Flutter: Food panda Clone/PWA:To-Do-List website **Roll No. 47**





HOSTED LINK:
https://pranavraikar01.github.io/pwa-project/

HOSTED IMAGE:



Conclusion:Thus we have implemented deployment of PWA website to GitHub Pages.

MAD & PWA Lab
Journal

Experiment No.	11
Experiment Title.	To use google Lighthouse PWA Analysis Tool to test the PWA functioning.
Roll No.	47
Name	PRANAV SANDEEP RAIKAR
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO6: Develop and Analyze PWA Features and deploy it over app hosting solution
Grade:	15

PRANAV SANDEEP RAIKAR
D15A 47

Practical 11

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

Theory:

Google Lighthouse :

Google Lighthouse is a tool that lets you audit your web application based on a number of parameters including (but not limited to) performance, based on a number of metrics, mobile compatibility, Progressive Web App (PWA) implementations, etc. All you have to do is run it on a page or pass it a URL, sit back for a couple of minutes and get a very elaborate report, not much short of one that a professional auditor would have compiled in about a week. The best part is that you have to set up almost nothing to get started. Let's begin by looking at some of the top features and audit criteria used by Lighthouse.

Key Features and Audit Metrics

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

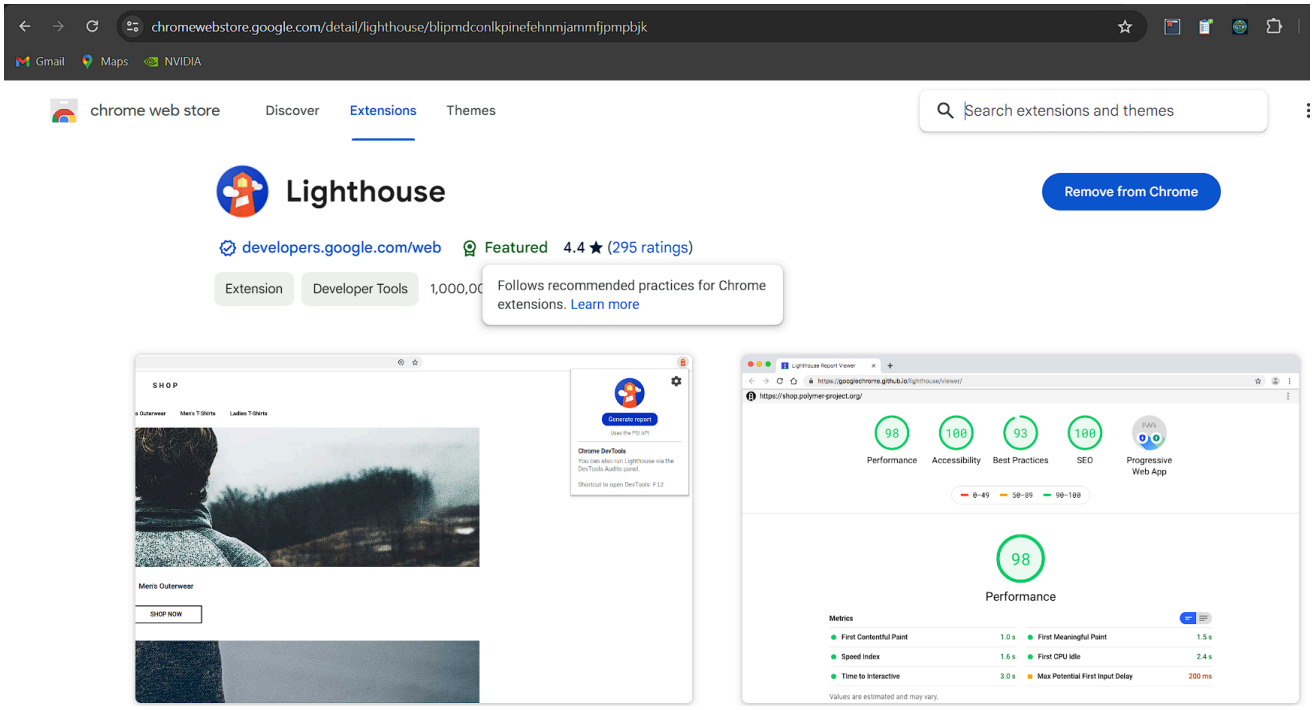
1. Performance: This score is an aggregation of how the page fared in aspects such as (but not limited to) loading speed, time taken for loading for basic frame(s), displaying meaningful content to the user, etc. To a layman, this score is indicative of how decently the site performs, with a score of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

2. PWA Score (Mobile): Thanks to the rise of Service Workers, app manifests, etc., a lot of modern web applications are moving towards the PWA paradigm, where the objective is to make the application behave as close as possible to native mobile applications. Scoring points are based on the Baseline PWA checklist laid down by Google which includes Service Worker implementation(s), viewport handling, offline functionality, performance in script-disabled environments, etc.

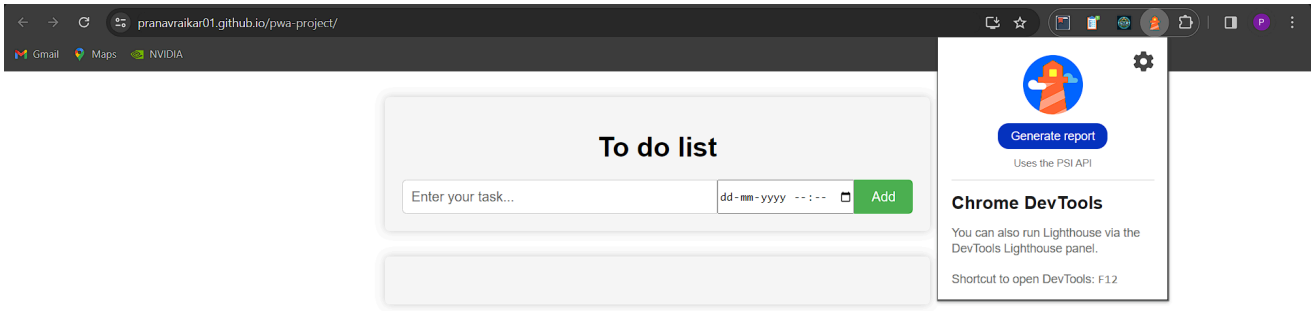
3. Accessibility: As you might have guessed, this metric is a measure of how accessible your website is, across a plethora of accessibility features that can be implemented in your page (such as the 'aria-' attributes like aria-required, audio captions, button names, etc.). Unlike the other metrics though,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

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

Project Title: Flutter: Food panda Clone/PWA:To-Do-List website **Roll No. 47**



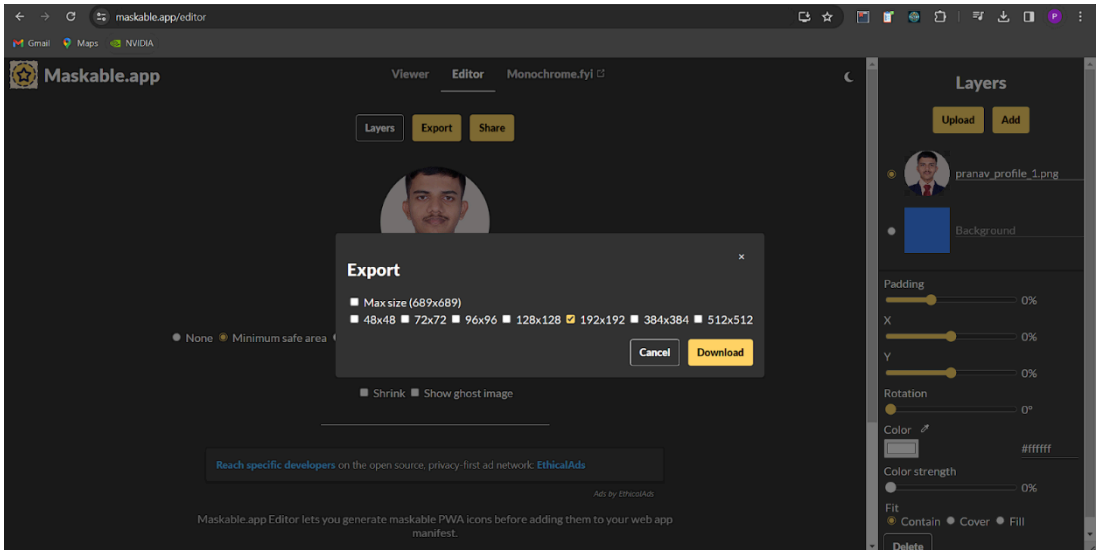
Website for report: https://pranavraikar01.github.io/pwa-project/



Click on Generate report.

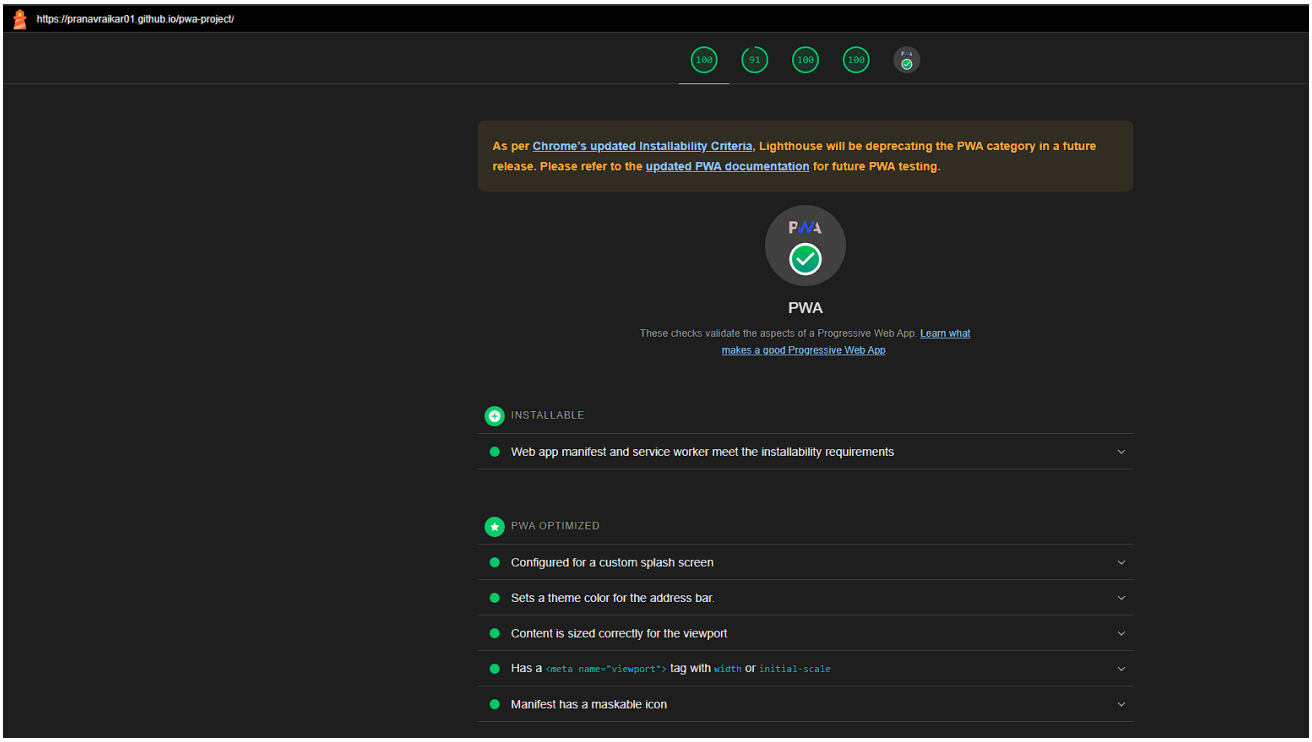
PWA steps is not yet completed for that we have to add maskable image icon

Project Title: Flutter: Food panda Clone/PWA:To-Do-List website **Roll No. 47**



manifest.json:{

```
 "short_name": "Pranav Raikar",
 "name": "Pranav Raikar",
 "icons": [
  {
    "src": "pranavprofile.jpeg",
    "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"
  },
  {
    "src": "maskable_icon_x192.png",
    "sizes": "192x192",
    "type": "image/png",
    "purpose": "maskable"
  }
 ],
 "start_url": ".",
 "display": "standalone",
 "theme_color": "#000000",
 "background_color": "#ffffff"
}
```



Conclusion:
In this experiment we have successfully used Google Lighthouse PWA analysis tool for testing the PWA functioning

MAD & PWA Lab
Journal

Experiment No.	Assignment-1
Assignment 1 Questions	1. Flutter Overview:Explain the key features and advantages of using Flutter for mobile app development. Discuss how the Flutter framework differs from traditional approaches and why it has gained popularity in the developer community. 2. Widget Tree and Composition: Describe the concept of the widget tree in Flutter. Explain how widget composition is used to build complex user interfaces. Provide examples of commonly used widgets and their roles in creating a widget tree. 3. State Management in Flutter: Discuss the importance of state management in Flutter applications. Compare and contrast the different state management approaches available in Flutter, such as setState, Provider, and Riverpod. Provide scenarios where each approach is suitable. 4. Firebase Integration in Flutter: Explain the process of integrating Firebase with a Flutter application. Discuss the benefits of using Firebase as a backend solution. Highlight the Firebase services commonly used in Flutter development and provide a brief overview of how data synchronization is achieved.
Roll No.	47
Name	PRANAV SANDEEP RAIKAR
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO1: Understand cross platform mobile application development using Flutter framework LO2: Design and Develop interactive Flutter App by using widgets, layouts, gestures and animation LO3: Analyze and Build production ready Flutter App by incorporating backend services and deploying on Android / iOS
Grade:	5

DISA 47

3/2/24 MAD PWA LAB Assignment 1

(1) Flutter overview :-

→ Flutter is an open source Framework by Google for building mobile, web and desktop applications from a single codebase. Its key features include a hot reload for quick development, a rich widget library for expressive UI, high performance and a supportive community. Flutter's advantages lie in its cost effectiveness, consistent UI/UX easy learning curve and widespread adoption. It differs from traditional approaches by enabling cross platform development with a single codebase, using a custom rendering ~~agents~~ engine, and employing a widget based UI. Its popularity is driven by community support.

Features :-

(1) Single codebase :- Flutter offers a single codebase that can be used to create applications for multiple platforms.

(2) Hot Reload :- Developers can see the changes made in the code instantly reflected in the app.

(3) High performance :- Flutter uses the SKIA graphics engine to render visuals contributing to smooth animation and a ~~re~~ responsive render experience.

(4) Access to native features :- Flutter provides seamless access to native features and API's allowing developers to integrate device specific functionalities without compromising on performance.

Q2 Widget tree and composition :

→ In flutter, the widget tree represents the hierarchy of UI elements in an app. It's a tree structure where each node is a widget, forming the visual components of user interface. Widget composition is the process of combining simple widgets to create more complex ones.

① Container : A basic rectangular box often used to group and style other widgets.
② Column / Row : Widgets for arranging children vertically or horizontally.
③ Text : Displays a string of text.
④ Stack : Used to put one widget on top of the other.
⑤ Image : Used to display an image on the UI.

By combining these widgets, you can create intricate UI's. For instance, a login screen might use a Column with Text widgets for labels, and TextFields within a container for input. This hierarchical arrangement of widgets forms the widget tree, facilitating the construction of diverse and sophisticated user interfaces in Flutter.

Q3 States Management in Flutter : Discuss the importance of state management in flutter application Compare and contrast the different state management

approaches available in Flutter, such as setState, Provider and Riverpod. Provide scenarios where each approach is suitable.

→ State management in Flutter is crucial for maintaing and updating the application's state. It ensures a responsive and dynamic user interface. In flutter effective state management is vital for building responsive and dynamic applications. It involves handling and updating the state of your widgets to reflect changes in the user interface.

Different approaches include :-

① setState :- This is the most straight forward.

Use case :- • Suitable for small to medium sized applications
• Ideal for managing simple UI specific states.

Pros :- • Straight forward & easy to use.
• No external dependencies

Cons :- • Limited to the scope of a single widget.
• May lead to widget rebuilds higher in widget tree

② Provider :- A popular state managent solution that provides global & easily accessible state.

• Use case :- • Good for managing global state dependency injection and sharing data between widgets

• Pros :- • Simple & easy to implement.
• Offers good performance and efficiency.
• Well suited for dependency injection.

Cons:- • can be considered overkill for small application.

⑤ Riverpod:- An extension of provider that introduces improvements in terms of readability, scalability and testability
Use case:- offers improved syntax and advanced features compared to provider.
pros:- Provides improved readability and testability.
Cons:- Slightly deeper learning curve compared to provider

Scenarios:-
① setState():- example - Basic calculator app where state changes are isolated to a single server.

② Provider:- eg:- A weather app with multiple screens where the ~~co~~chosen location and temperature unit need to be shared across different widgets.

③ Riverpod:- eg:- An e-commerce app with complex user applications shopping cart management and real time updates where riverpod, advanced features are beneficial.

Q4 Firebase integration in Flutter :- Explain the process of integrating firebase with a flutter application. Discuss the benefits of using Firebase as a backend solution. Highlight the firebase servie commonly used in flutter development and provide a brief overview of how data synchronization is achieved.

→ Integrating firebase with flutter applications.

① Create firebase project.

② Register your app by clicking on Add app and choose the appropriate platform

③ Add Firebase SDK to flutter project:-

Add Flutter SDK dependencies to pubsec.yaml file

firebase_core: ^1.10.6

firebase_auth: ^4.1.0.6

④ Initialize firebase in your App i.e. main.dartfile 'firebase.initializeApp().

⑤

Benefits of using firebase as backend solution:-

① Real time database

② Authentication

③ Cloud firestore

④ Cloud functions.

⑤ Cloud storage

⑥ Easy integration

⑦ Scalibility.

⑧ Authentication providers

⑨ Analytics and crash reporting

Data synchronization in Flutter is achieved through various techniques like:

1. State management
2. Streams and Futures
3. Collections and documents
4. Real time updates
5. Offline support.

MAD & PWA Lab
Journal

Experiment No.	Assignment-2
Assignment 2 Questions	1. Define Progressive Web App (PWA) and explain its significance in modern web development.Discuss the key characteristics that differentiate PWAs from traditional mobile apps 2. Define responsive web design and explain its importance in the context of Progressive Web Apps.Compare and contrast responsive, fluid, and adaptive web design approaches. 3. Describe the lifecycle of Service Workers, including registration, installation, and activation phases. 4. Explain the use of IndexedDB in the Service Worker for data storage.
Roll No.	47
Name	PRANAV SANDEEP RAIKAR
Class	D15A
Subject	MAD & PWA Lab
Lab Outcome	LO4:Understand various PWA frameworks and their requirements LO5: Design and Develop a responsive User Interface by applying PWA Design techniques LO6:Develop and Analyze PWA Features and deploy it over app hosting solutions
Grade:	4

Pranav . S. Raikar
D15A 47

PWA Assignment - 2

Q1 Define Progressive Web App (PWA) and explain its significance in Modern web development. Discuss the key characteristics that differentiate PWA's from traditional mobile app.

→ A progressive web app (PWA) is a type of web application that uses modern web capabilities to deliever an app-like experience to users. PWA's are built using standard web technologies such as HTML, CSS & Js but are designed to provide a more native-like experience for users, especially on mobile devices.

Key characteristics of Progressive web Apps are -

① Progressive Enhancement - They are built using progressive enhancement principles, meaning they provide a basic experience for all users.

② Responsive design : PWAs are designed to feel and behave like native mobile apps.

③ Reliability - PWA's are designed to be reliable, even when users are offline or has slow or unreliable internet connection. They cache content and resources to ensure that the app remains functional.

④ Fast Performance.

⑤ Discoverability - PWA's are easily discoverable and shareable as they are built using standard web technologies and can be accessed via in URL.

⑥ Security - PWA's are served over HTTPS to ensure security.

⑦ Cross-platform compatibity.

The significance of PWA's in modern web development lies in their ability to bridge the gap between web and native app experiences. By combining the reach and accessibility of the web with functionality and engagement of native apps. They allow developer to create fast reliable and engaging experience that work seamlessly across wide range of devices and platforms.

Q2 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.

→ Responsive web design is a web design approach that ensure a website adapts its layout and content to seamlessly function and display well across various screen sizes from desktops to smartphones.

Importance of PWA :-

① Foundation for App like experience - RWD ensures proper layout and navigation across devices, mimicking the adaptability of native apps.

② Accessibility - A responsive design make PWA accessible to wider audience using diverse devices

③ SEO - benefits - Responsive design makes a ~~Responsive~~ positive ranking factor in search engine optimization.

④ Responsive web design VS Fluid VS Adaptive

Feature	Responsive	Fluid	Adaptive
Layout	① Flexible, adaptive to any screen size	① Continuously adjusts based on relative units	① Uses multiple fixed width layouts
Development effort	① Moderate	① Less effort for basic layout	① More efforts to create multiple layouts
Control	⑤ Good control over overall layout	③ Less control over element appearance	③ High control over layout for each device category
Suitability for PWA	④ Ideal Foundation due to its versatility	⑤ Can be used but may require adjustments	⑤ Less common for PWA.

Q3 Describe the lifecycle of service workers including registration, installation and activation phases.

→ Service workers are crucial components of progressive web Apps (PWA's) that enable features such as offline support, push notification and background synchronization.

① Registration :- 1st phase of lifecycle. Occurs in the main javascript file of web application.

- Registration is done using the navigator.serviceWorker.register() method which takes the path to service worker file as parameter.
- This method returns a promise that resolves to a service worker registration object which can be used to

interact with service worker.

④ Installation - Once service worker registered, browser downloads and starts installation process of service worker script.

- During installation the service worker script is executed and the install event is fired. Inside #install event handler, developers can define ~~the~~ what resources should be cached for offline access using techniques like caching strategies or ~~pre~~ precaching
- Installation process is critical for setting up the service worker's initial cache and preparing it to take control of web application

⑤ Activation :- After installation phase, Activation phase. It occurs when service worker is ready to take control of the web application

- During activation, 'activate' event is fired giving service worker an opportunity to clean up any old caches or perform other tasks necessary for application functionality.
- Developers can use the 'activate' event handler to manage cache cleanup versioning and other activation related tasks

Q4 Explain the use of indexed DB in the service worker for data storage

→ Indexed DB is a low-level ~~AB~~ API for client side storage of significant amounts of structured data, including files and blobs. It's particularly

useful for storing data in web applications that need to work offline or with large datasets that may not fit comfortably in memory.

(4) • Persistent storage - Service workers themselves are temporary by nature indexed DB provides persistent storage on the user's device, allowing PWAs to save data even when offline.

• Structured data - Indexed DB stores data in a structured format using object stores and keys making it ideal for complex data models used by PWAs.

• Offline data storage - service workers are background run seperately from web pages and enable features like push notifications and offline functionality. Indexed DB provides a way for service workers to store data locally on the user's device, allowing web applications to work offline and improve performance by caching frequently accessed data.

(3) Structured data storage - IndexedDB stores data in key-value pairs within object stores similar to how data is organized in a database. Service workers can create access and manage object stores to store different types of data required by the web application, such as user preferences cached content or application state.